BitTorrent Discovery and Performance Enhancement using DDS QoS Policies

BY

Anas Ahmed Abu-Dagga

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 Networks

December 2014

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Anas Ahmed Abu Dagga** 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.**

[illegible]

Dr. Basem Al madni
(Advisor)

Dr. TYPE NAME
(Co-Advisor)

[illegible]

Dr. Tarek Sheltami
(Member)

[illegible]

Dr. Marwan Abu-Amara
(Member)

Dr. TYPE NAME
(Member)

[illegible] 25/12/2014

Dr. Ahmad Almulhem
Department Chairman

[illegible]

Dr. Salam A. Zummo
Dean of Graduate Studies

5/1/15

Date

To my lovely family; parents, brothers, and sisters

ACKNOWLEDGMENTS

All the praises and thanks are to Allah; Oh Allah how can I praise you, how can I thank you. Oh Allah help me for remembering you much, thanking you much. There is no power but with Allah.

Who does not thank the people, do not thank Allah. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this thesis. I would like to thank few people in particular. To My esteemed advisor, Dr.Basem Al-madani, for his kind treatment, attention to detail and correctness, motivation and encouragement. Besides my advisor, I would like to thank the rest of my thesis committee: Dr.Tarek Sheltami and Dr. Marwan Abu Amara, for their encouragement and insightful comments. Thanks to Dr.Ashraf Mahmoud, he gives me good solutions for networking issue. I would also like to thank King Fahd University of Petroleum and Minerals for the support extended towards my research and for granting me the opportunity to pursue graduate studies, special thanks are given to the Department of Computer Engineering. Thanks to the Real-Time Innovation Inc. (RTI) for its support with all tools that I have used in this thesis work. Heartfelt thanks to my family for their support and encouragement throughout my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

P2P: Peer to Peer

RTT: Round Trip Time

BDP: Bandwidth Delay Product.

RF: Rarest First.

PS: Publish/Subscribe.

RTPS: Real Time Publish Subscribe.

JMS: Java Message Service.

DDS: Data Distribution Service.

COM+: Component Object Model.

OMG: Object Management Group.

QoS: Quality of Service.

DCPS: Data Centric Publish Subscribe.

GDS: Global Data Space.

RTI: Real Time Innovation.

PEX: Peer Exchange.

DHT: Distributed Hash Table.

UDP: User Datagram Protocol.

TCP: Transmission Control Protocol.

ISP: Internet Service Provider.

IOP: ISP-Owned Peers.

BGP: Border Gateway Protocol.

JDK: Java Development Kit.

API: Application Programming Interface.

LAN: Local Area Network.

WAN: Wide Area Network.

CPU: Central Processing Unit.

PC: Personal Computer.

RAM: Random Access Memory.

DCPS: Data Centric Publish Subscribe.

GNS3: Graphical Network Simulator.

HB: Heart Beats.

GPL : General Public License.

ABSTRACT

Full Name	:	Anas Ahmed Abu Dagga
Thesis Title	:	BitTorrent Discovery and Performance Enhancement using DDS Middleware
Major Field	:	Computer Networks
Date of Degree	:	Dec 2014

BitTorrent is the most worldly adopted peer to peer (P2P) file distribution application protocol that constitutes a huge part of today's Internet traffic. P2P model benefits BitTorrent's peers in file exchanging process, by eliminating a single point of congestion. Beside the main P2P implementation of BitTorrent for file exchanging, it also has client-server communication between the peers and tracker. The tracker is used in BitTorrent network for peers' discovery. However, the tracker does not benefit from P2P characteristics. Mainly, the tracker is considered to be a single point of failure, also, scalability and load-balancing are other traker's issues. Another problem is that BitTorrent uses eleven overhead messages which help in distributing the file pieces among the peers. In this research work, we aim to have a pure P2P BitTorrent application, and minimize the messages overhead. For Discovery, we propose a novel architecture to decentralize the tracker and make it distributed among the peers. The proposed method reduce both communication overhead and node searching complexities to *O(1).* For Dissemination, we re-implement the existing BitTorrent using Data Distribution Service (DDS), which is a Real Time Publish Subscribe (RTPS) middleware. We study the performance in terms of file downloading time and goodput for both the original and the proposed dissemination protocols. The results show that the proposed solution can minimize the BitTorrent overhead achieving high goodput, and speed up the file downloading process in most cases. The proposed approaches are tested and validated only over Intranet.

ملخص الرسالة

الاسم الكامل : أنس أحمد محمد أبو دقة

عنوان الرسالة : تحسين الاستكشاف والأداء في البت تورنت باستخدام أدوات تحسين الجودة الموجودة في تقنية توزيع البيانات (دي دي اس) الوسيطة.

التخصص : شبكات الحاسب الآلي

تاريخ الدرجة العلمية: ديسمبر 2014

البت تورنت هو بروتوكول مشاركة الملفات بطريقة النظير للنظير ، وهو الأكثر انتشارا حول العالم حيث يستهلك الجزء الأكبر من مساحة تبادل البيانات عبر الإنترنت. استفاد البت تورنت من طريقة الاتصال المسماة النظير للنظير في عملية تبادل الملفات الكبيرة من غير أن يكون هناك خادم مركزي بين هؤلاء النظراء يسبب الاختناق وتزاحم البيانات وتأخر نقلها . إلا أن البت تورنت أيضا يستخدم نموذج الاتصال المسمى الخادم والعميل بين النظراء وبين خادم مركزي يسمى المتابع. المتابع في نظام البت تورنت يُستَخدم في عملية استكشاف النظراء . العيب الرئيسي لهذا المتابع أنه بتعطله عن العمل قد يتعطل جميع النظام عن العمل، أيضا عدم قابليته للتوسع مشكلة أخرى. المشكلة الأخرى في البت تورنت أنه يستخدم إحدى عشر رسالةً زائدة لتسهيل عملية نقل الملفات بين النظراء .في هذا البحث نهدف لجعل البت تورنت يعمل بالكامل على نموذج الاتصال المسمى النظير للنظير والتقليل من عدد الرسائل الزائدة المستخدمة في عملية نقل الملفات. لاستكشاف النظراء, قمنا بعمل مخطط للتخلص من المتابع وتوزيع مهمته بين النظراء . طريقة البحث المقترحة تقلل الأحمال الزائدة على الشبكة وتقلل عملية الوصول للنظير المطلوب إلى رقم ثابت يساوي الواحد الصحيح .لإرسال البيانات بين النظراء ، قمنا بإعادة تصميم وبرمجة البت تورنت على أساس استخدام خدمة توزيع البيانات (دي دي اس)،

هذه الخدمة تعتبر وسيط للنشر والاشتراك في الوقت الحقيقي. وقمنا باختبار كفاءة كلا من البت تورنت الأصلي والمقترح الجديد. النتائج أظهرت أن المقترح الجديد يستطيع تقليل الأحمال الزائدة التي يرسلها البت تورنت الأصلى مع الملفات المرسلة. أيضا النتائج أظهرت أن المقترح الجديد يساهم في تسريع عملية تحميل الملفات في معظم الحالات.

CHAPTER 1

INTRODUCTION: PEER-To-PEER NETWORKS

The distributed computing architecture known as peer-to-peer (P2P) networks are designed to allow users to share resources without any intermediary control or authority centralized coordination services. Conceived and developed in the specific context of the Internet, these systems are inspired by a simple and fundamental principle: ability to adapt to a highly dynamic environment while maintaining substantially unchanged reliability and connectivity. In a peer-to-peer environment, resources can be considered as missing since their availability is generally considered independent of the availability of the node that initially shared the resources. Specific techniques are implemented for this purpose and they are varied according to the particular architecture. They are based on the redundancy of resources, and the decentralization of routing and discovery algorithms.

The P2P overlay networks are distributed systems that have no hierarchical organization and no centralized control. Peers are self-organized over the IP networks, offering a set of characteristics such as robust routing architectures and efficient data search.

The P2P model is the antithesis of the classic client-server architecture in which each node is a server or a client that depends on a central authority; the data are stored on a server which sends them to those clients who request it. The P2P model, on the other hand, means that each node behaves as both a server and client depending on who is the supplier or the requester of a particular resource. Client-server and P2P networks are depicted in Figure 1.1and Figure 1.2 respectively.

1.1 Comparison: P2P vs. Client-Server

As mentioned above the P2P model is quite different from the client-server model. Table 1-1 shows the main differences between the two models.

Table1-1: P2P Model vs. Client-Server Model

Client-Server	P2P
• Asymmetry: client and server are distinct and play different tasks. • Global Knowledge: servers have a global view of the network. • Centralized Approach: the communication and management are centralized. • Single Point of Failure: the failure of the server involves the malfunction of the entire system • Limited Scalability: servers are subject to an overhead which limits the network scalability. • High Cost:	• Symmetric: each node has the same functions and can be both client and server. • Local Knowledge: peers know only a subset of the network nodes. • Decentralized Approach: there is no global knowledge but only local interactions. • Robustness: the consequences of one node failure is malfunctioning of minimal or null nodes. • High Scalability: due to the distribution of the load and the high aggregate capacity, the network is highly scalable.

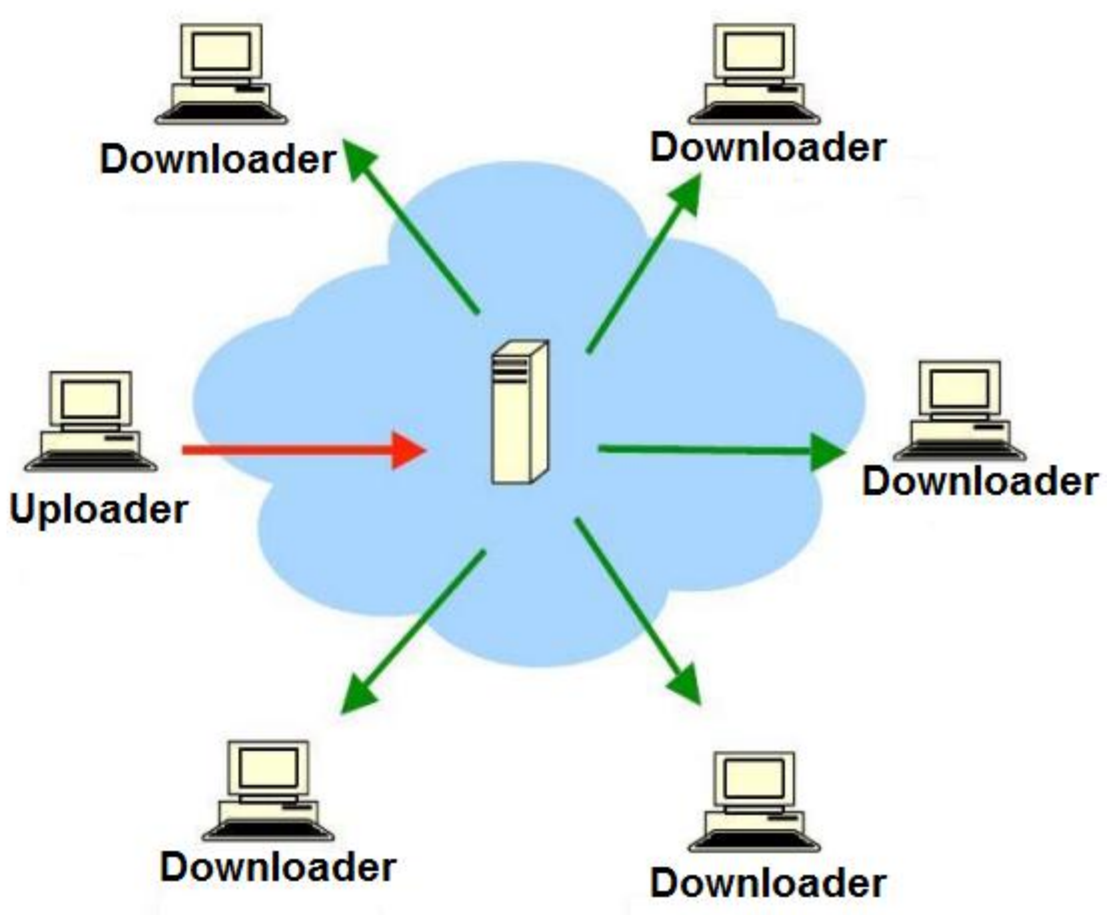

Figure 1.1 Client-Server Model

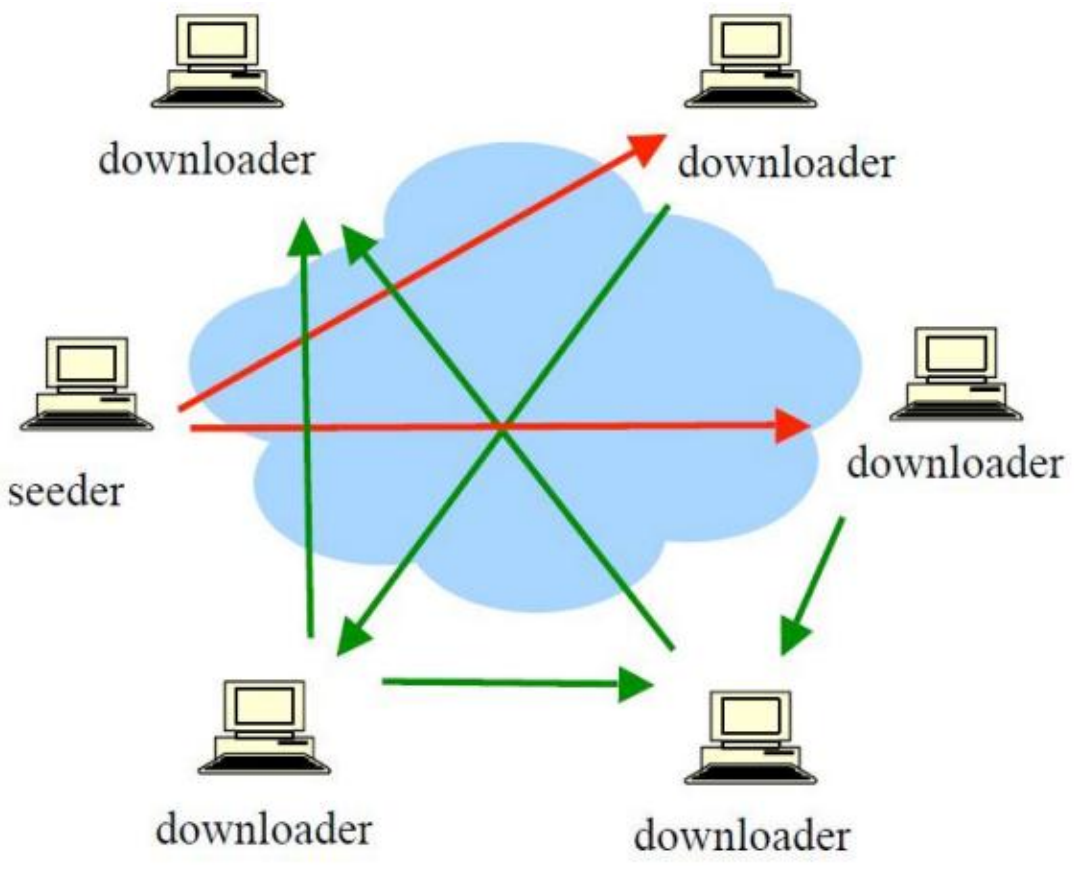

Figure 1.2: P2P Model

1.2 P2P Architectures

It should be noted that a full decentralized P2P cannot be realized on a large scale due to the need of one or more nodes which have the task of providing the parameters of the initial communication to the new nodes that want to join the system. Beyond this, the P2P networks are commonly classified according to the relationship between the location of resources in the network and the topology of the network.

The main architectures referred to when talking about P2P systems are: centralized or decentralized, and structured or unstructured [63]. In order to compare the different solutions available, the following two main aspects must be taken into account:

1. Scalability of the system: For each node you have to check the overhead of communication (number of steps to reach the node that stores information), and

the used memory (size of the routing table) as a function of number of nodes in the system (N).

2. Robustness and adaptability to frequent changes and malfunctions.

1.2.1 Centralized and decentralized:

The distinction between P2P architectures is related to the centralized and decentralized presence or absence of a coordination node [18]. Some architectures, in fact, are based on a central infrastructure that performs services resources indexing, freeing the burden of peers to distribute the resources themselves. Figure 1.3 represents the organization of centralized P2P architecture. Clients connect to a central unit which has the task of maintaining the following:

- A table of connected users (IP address, port number, if any connection information such as the bandwidth, etc.).
- A table of shared resources by each user, possible accompanied by metadata (information that describes a set of data).

Upon the joining, clients contact the central unit by publishing a list of resources that are willing to share. Queries (requests) are forwarded to the central server that locates in its tables the peer (or possibly peers) sharing a resource that meets the query conditions. The subsequent communication occurs between the peers. The next communication between clients takes place directly with one or more direct connections between the peer that requested the resources and its peers that distribute the resources. If the advantage of the centralized model lies in the simplicity and reliability of the protocol, the main problem is the presence of a single point of failure and the vulnerability to censorship. In fact, the development of Napster [19], one of the first examples of this architecture, has been discontinued due to the in slew of lawsuits filed against Napster rather than because of technological limitations.

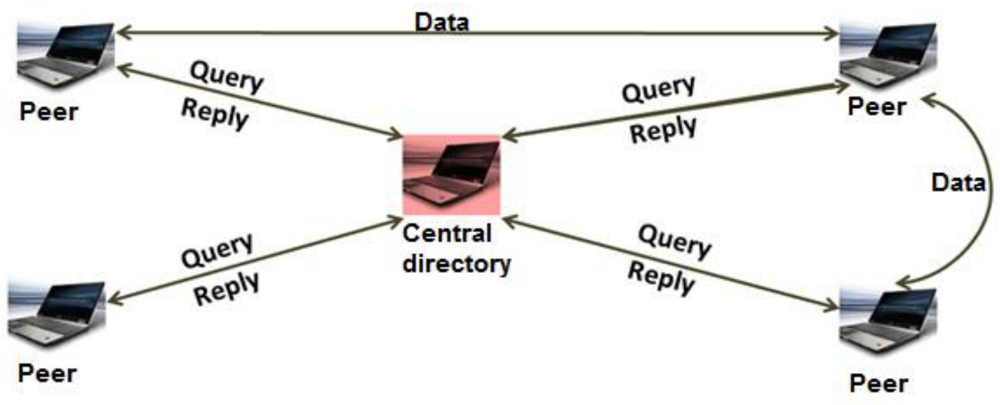

Figure 1.3 : Centralized P2P architecture.

In a network of this type, the node that owns the resource is found after *O(1)* steps because you just forward a request to the centralized server. The server will store amount of data equal to *O(N)* where N equal to the number of available resources (files) in the system.

Peer-to-peer decentralized architecture is characterized by the lack of a single point of “break”; characteristic due to the fact that in the system there is no privileged node required by the operation. There is no centralized control of the network, and each user application acts simultaneously as both a client and a server. Such a user application is referred to as a servent. An example is the Gnutella network [20] in which the communication between the servents are regulated by a protocol that defines four types of messages:

- ping: represents an announcement to a servent presence on the network.
- Pong: is a reply message to a ping. Contains the IP address and port of the sender of the message, plus the number and size of the file shared.
- Query: a resource request that includes also information on bandwidth requirements.
- Query hits: a response to a query that contains IP address and port number to which to connect to download information relative to the bandwidth and the number of files that match the requirements of the request.

An example of communication in a peer-to-peer that is purely decentralized is shown in Figure 1.4 .

After a node joins to the network, it sends a ping message to each of its neighbors, all those nodes which it directly knows the IP addresses and their port numbers. These, in turn, respond with a message of pong identifying amount of data that nodes shared, and propagate the ping message to their neighbors. The location of a resource on the network is found by sending a query message that is propagated in the network until the exhaustion of a time-to-live(TTL); any replies are forwarded to the node that originated the query following the reverse path. When a node receives a query hit, indicating that the resource has been localized to a certain peer it establishes a direct connection with that peer for downloading. The scalability of the system is guaranteed by the TTL of the messages, which determines a limit beyond which the messages cannot propagate, avoiding the collapse of the network.

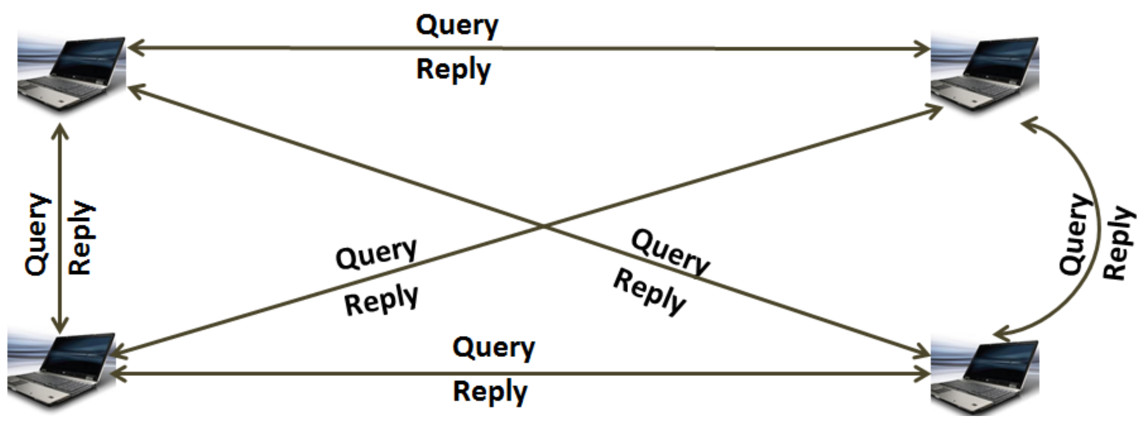

Figure 1.4: P2P decentralized architecture.

The complexity of P2P decentralized architecture is $O(N^2)$ in terms of looking for and getting a resource, and the looking up results are not guaranteed, since the lifetime of the query message is limited by TTL. On the other hand, the information of routing is shared and does not depend on the number of nodes in the system, so each node will store a quantity equal to $O(1)$.

A middle way is represented by hybrid systems that employ the concept of super-peers, which forward the received query to other super-peers with which they are connected in a totally decentralized topology. These nodes are elected based on the computing power and bandwidth. These super-peers enhance the overall network performance by benefiting from both centralized and decentralized architectures, in the other words, resource allocation can be done by both decentralized and centralized search techniques. Maintaining the scalability of decentralized P2P systems, and the speed of resource locating of centralized P2P systems. The structure of a peer-to-peer hybrid is shown in Figure 1.6.

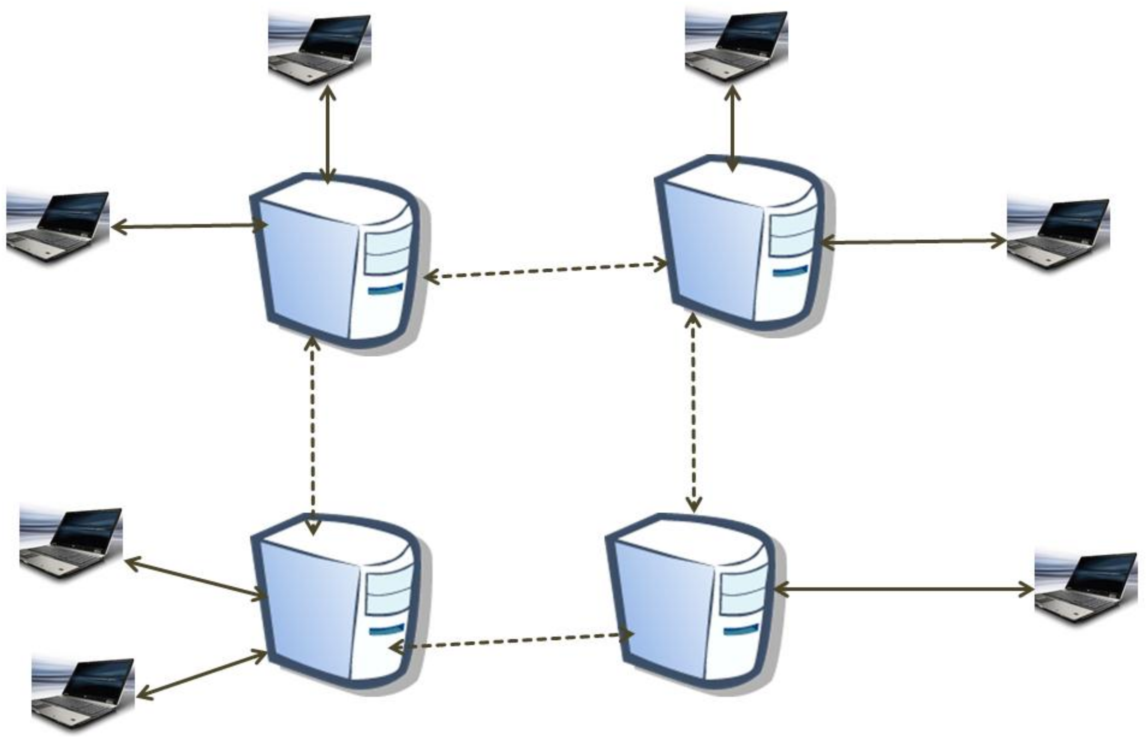

Figure 1.5: P2P hybrid architecture

1.2.2 Structured and unstructured

The networks can be classified, also, in structured and unstructured. This classification is based on the relationship between the network topology and the location of a resource in the network.

In unstructured networks, there is no relationship between the location of a resource and the network topology. The overlay network uses techniques such as non-deterministic flooding (requests are forwarded to all participants) or the random walks (requests are forwarded to a subset of the participants chosen randomly) for the location of a resource on the network. Each peer evaluates the query received locally and sends it to the peer sending list of all content that matches the request. It should be noted that while the flooding-based techniques are well suited to those systems where peers come and go at high frequency and in which the data are high redundant, they are not very suitable for locating data not very common because, as they are in possession few peers, queries should be sent to almost all of them. The basic problem of unstructured P2P networks are: peers become easily overloaded, the network is not scalable in the case it should manage the aggregated queries with high frequency and further sudden increase in the size of the system which causes a proportional increase of requests within the network. Examples of unstructured P2P overlay networks are: Gnutella [20], KaZaA [22], eDonkey2000 [21], and BitTorrent which will be discussed in detail in later chapters.

In structured networks, however, resources are stored on the nodes chosen in a deterministic way according to a specific algorithm which provides a mapping between the content and nodes, or, more precisely, between the respective identifiers in the form of routing tables that are used to route the queries efficiently to those peers who own the particular resource. The structured P2P networks are able to locate data in an efficient rare, since the routing based on deterministic algorithms is scalable. Examples of structured P2P overlay networks are Distributed Hash Table (DHT) algorithms such as: Tapestry [23], Chord [24], and Kademlia [25]. A Distributed Hash Table is a data structure distributed that allows you to store the pairs<key,value>, obtained by a hash function, in a way that is efficient, reliable, and robust. This approach is to assign the unique identifier (key), selected in the space of identifiers, data, and simultaneously assigning identifier (value), selected from the same space, to a set of nodes that possess the information to which the key refers. Operations such as “put (key, value)” and “value=get(key)” can be invoked to store and retrieve the data corresponding to the key. This involves the exchange of routing messages to the peer which is associated with the same key. Each node maintains small routing tables in which stores the identifiers of neighboring peers and their IP addresses. Requests for localization or routing messages are forwarded to the peers in the overlay in a progressive manner considering the identifiers of the nodes which are next to the space key identifiers. In theory, systems based on DHT can ensure that any data can be located, on average, in a number of steps equal to O(log N), and that each node possesses the routing tables with O(log N) entries. It should be noted that the path between two nodes, the underlying physical network, can be very different from the DHT overlay network based on the approach, this may causing a delay in the localization phase which lead to a deterioration of the performance of the network. The DHT has a limit due to the transient nature of the P2P clients. The graceful exit (announced) of a node usually results in O(log N) operations to maintain consistent data structures, on the other hand, a kind of graceless (no ad budget and transfer of state information) have worse outcomes.

1.3 Applications of P2P Systems

The P2P systems are used in a wide range of applications, characterized by substantial independence from authority coordination. These applications can basically fall into three main categories:

- File sharing and content distribution.
- Communication and collaboration.
- Distributed computing.

1.3.1 File sharing and Content Distribution

Most P2P systems available today fall into this category, to the point that in some settings, the P2P technology has been implemented as a synonym for piracy and copyright infringement. In fact, this type of application ranging from simple tools to complex file-sharing platforms for publishing, cataloging, retrieval and distribution of digital content. Examples of file-sharing applications are already mentioned, Napster (1999), Gnutella and eDonkey (2000), KaZaA (2001), eMule and BitTorrent (2002).

Applications for Content Distribution, however, are the distribution of real-time audio and video data over the network (P2P live media streaming). Examples of this type are the open-source software PPLive [28] [27], SopCast [29], and CoolStreaming [26].

1.3.2 Communication and Collaboration

In this category, the framework of the infrastructure is built to facilitate the direct communication between applications or users on the Internet. Typical examples are chat, instant-messaging and P2P VOIP. Skype [30] is a famous example of P2P usage in VoIP; it was built on top of the infrastructure of P2P file-sharing network, Kazaa [22]. The bandwidth is shared and the sound or video in real-time are shared as resources. The central server is available only for the existence information and calculating invoices for the system users whenever they make a call that has charges.

Document collaboration is important for a team or a company. Collaboration with P2P makes it much effective and simple rather than using a centralized server. Groove [31] is a Microsoft application with P2P capabilities for document collaboration. Groove offers Microsoft Office based solutions, instant messaging, and video conferencing solutions. It, also, provides role and user based security, which is one of the most important aspects of P2P for an organization.

1.3.3 Distributed Computing

The distributed computing requires a moderate commitment arbitrage to divide a job into fragments that are then sent to other computers for processing. The results are subsequently conveyed to a centralized repository.

Distributing computing is important for scientific research. P2P plays a role in enabling high performance computing by sharing of resource like computation power, network bandwidth, and disk space. SETI@HOME [32] is a popular project enables users to search for extraterrestrial intelligence. It is a voluntary project with more than 3.3 million users in 226 countries. It has used 796,000 years of CPU time and analyzed 45 terabytes of data in just two and a half years of operation.

1.4 Problem Description

In the recent years, BiTorrent protocol got a lot of adoption as peer to peer file sharing system, and still being the dominant P2P traffic on the Internet. However, this protocol suffers from the dependency on a single server that is called a tracker for the coordination and the content routing between its peers; this is a single point of failure (SPOF) problem. Also, this tracker is vulnerable to Denial-of-Service (DoS) attack. Furthermore, the tracker is limited in terms of scalability and availability.

The other problem with BitTorrent is that during the dissemination of the files between the peers, the BitTorrent uses many overhead messages. These messages can increase the load on the network.

CHAPTER 2

BACKGROUND

2.1 BitTorrent Systems

BitTorrent is a protocol for peer-to-peer file sharing, designed in Python by Brahm Cohen in 2002 with the aim to facilitate the dissemination of large files over unreliable networks [5]. BitTorrent protocol takes the utility of uploading bandwidth of all peers in the swarm for downloading files for some other peers. The responsibility of tracking the updated information regarding file pieces and the peers that currently joining and participation in the swarm falls on centralized server called “Tracker”. The tracker sends to a new joiner peer a list of peers and chunks available with each one of those peers. The new peer is going to select a single or multiple peers from the list and start downloading the file chunks. The centralized tracker forms a critical part in the system where its failure will stop file exchanging between the peers which makes it as single point of failure (SPOF) [2].

Current BitTorrent consists of the five main elements [5] [7] to work properly: Tracker server: it was elaborated. Metafile(.torrent file):contains a metadata or information about the shared file/content . Details about the structure and the content of this file can be found in [5]. BitTorrent Client: it is an application that implements the BitTorrent protocol. There are a lot available versions of them such as BitTorrent, BitComet [39], uTorrent [40], etc. Initial peer (seeder): it is the original user who has the published file, the existence of this component is essential since it is the source of the data, and any other peer that had the completed file and start uploading is also becoming a seeder. The fifth and last element is Downloader (leecher): it is the consumer of the published data, it changes its state to “seeder” when it completes file downloading.

2.1.1 How BitTorrent Works:

Figure 2.1 illustrates the principle of operation of the BitTorrent protocol that we analyze in detail by distinguishing four main phases:

-*Initial Phase:*

The first step that you must do if you are interested in downloading a file through the BitTorrent application, is to obtain a .torrent file called metainfo file from a webserver or an email. This is followed by the execution of this file by running the BitTorrent client.

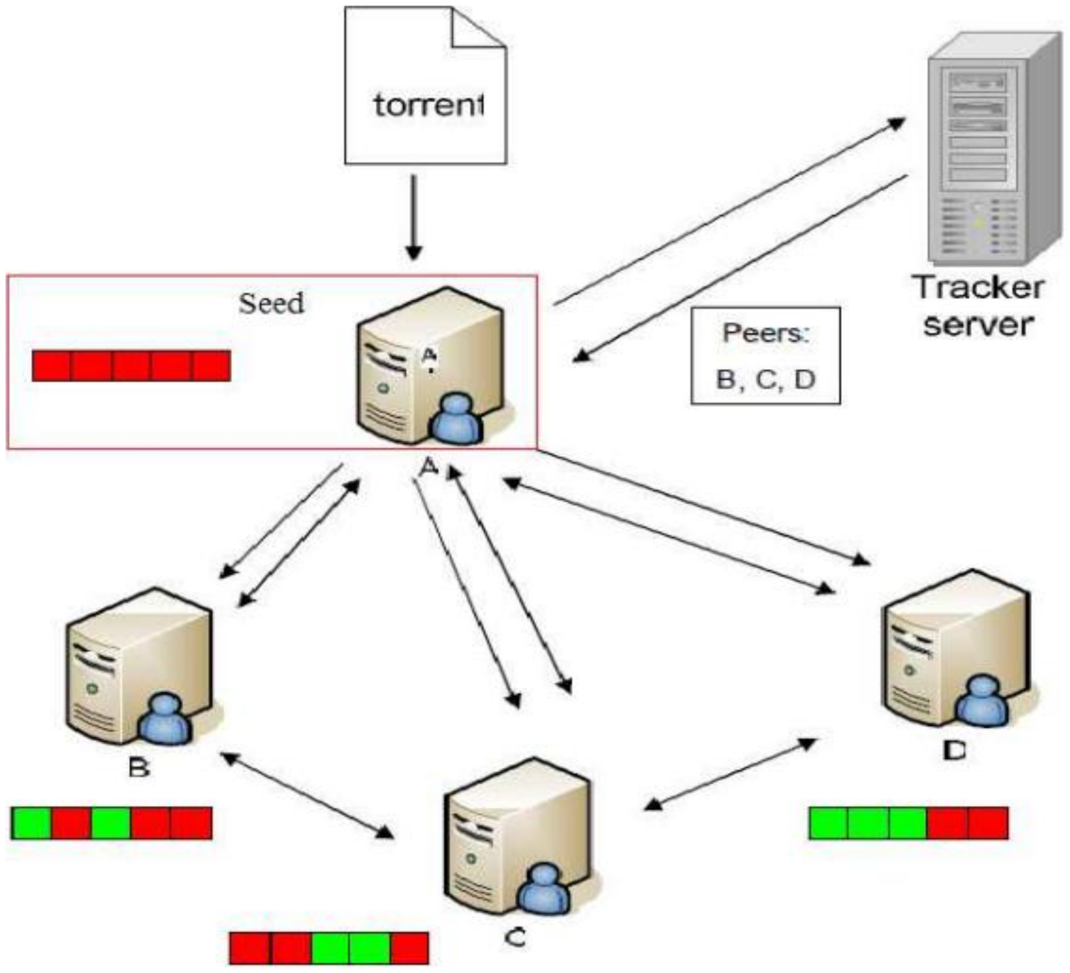

Figure 2.1: Standard BitTorrent components and how it works

- *Communication with the tracker:*

Secondly, you will contact the tracker that specified in the torrent file. The tracker provides HTTP/HTTPS that respond to an HTTP GET request. Tracker is used to manage the participation of users involved in the torrent (known as peers). A peer has no information about the other participants until it receive a response from the tracker. When a peer connects to the tracker it tells which pieces of the file the others peers have. In this way, when the peer queries the tracker, it returns a list ordered randomly with peers that are participating in the torrent, and who possess the required piece. By default, the list of peers returned by the tracker consists of 50 elements chosen by the tracker randomly. The tracker can choose to implement a mechanism for the selection of peers more intelligent. The clients can send requests to the tracker more frequently than the specified interval if a specified event occurs (completed or stopped) or if they need to increase the list of peers that they connected to.

- *Establishing Connections*

After obtain the list of peers participating in the swarm, the client establishes a bidirectional TCP connection with each of them; beginning the phase of data exchange. The BitTorrent client uses, in general, TCP ports 6881-6999. To find an available port, the client will choose, initially, the one with lowest number progressing until find one that can be used. This means that BitTorrent client will use only one port, and running another client will choose another port. When a peer receives a request for a particular piece from another peer, it may refuse to offer it. If this happens, we say that the peer is chocked. This is mainly due to the fact that by default, the client maintains a fixed number of simultaneous uploads (max_uploads), therefore, future requests will be suffocated (choked).

By virtue of this behavior, the client must maintain state information for each connection created with the others remote peers. Such information is specified by the value of the following variables:

- Choked: indicates whether the peer is, or not, “choked” by the remote peer. When a peer chokes, it notifies the client that does not respond to any request by the latter until it is unchoked (unlocked). The client, in this case, should not try to send requests for blocks and should consider all pending request will be ignored by the peer;

- Interested: indicates whether the peer is, or not, interested in one or more piece held by the client. If the peer is in this state it will starts as soon as possible to request blocks if it unchoked by the client;

At the establishment of the connection, the client will be choked and not interested.

- *Messaging and Data Exchanging:*

The first message that is send by the client is called handshake. It has a length of 49 bytes. The initiator of the connection should transmit handshake immediately, while, the peer on the other side of the connection waits for the initiator's handshake. After the handshake sequence has been completed and before any other message is exchanged, a bitfield message is sent. It is optional and, in case the client does not have any piece, it is not necessary to send it. The payload of this message is the pieces possessed by the peer that sent the message. The most significant bit of the first byte corresponds to piece index 0 and every single bit specifies whether the particular piece is owned (1) or not (0). Complete the exchange of bitfield among peers in the swarm, beginning the demands of the missing pieces by sending a request message. This message has a fixed length and its payload is the information about the piece required. If a peer is able to fulfill a request passes through the request message, the peer replies with a piece message. This message has a variable size that depends on the block of the piece x that has been selected. In the payload are the indices for the chosen piece and the actual data block. Just a piece has been downloaded, it is verified by the hash function and its receipt should notify the majority of the peers that he gets this piece by sending a have message. Now, peers, which have received have message, may request this piece from its new owner. If a client is no longer interested in a block which had requested, it sends a cancel message. To keep a connection between two peers open, a keep-alive message is sent periodically. This message has a length of 0 bytes and is specified with the prefix length set to zero. The peers who do not receive any message within a certain period of time can break down the connection; the keep-alive must then sent to keep the connection open if no other message has been sent for a given time, usually two minutes. Choke, unchoke, intereseted, and not intersted, these messages are used to update the status information of the connection. They are all characterized by of a fixed message length, and the absence of the payload.

2.1.2 BitTorrent Techniques and Algorithms

We illustrate in the following subsections the techniques introduced to improve the performance of BitTorrent protocol [41]:

2.1.2.1 Pipeling

This is a technique that allows you to increase your download speed when transferring data via TCP, as in the case of the application BitTorrent. The peers remain in the queue a number of unfulfilled requests for each connection. It works in this way because otherwise it would have to wait an entire round trip time (RTT) between the download of two successive blocks (round trip time between piece message and the subsequent request). In link with a high bandwidth-delay-product (BDP) this would result in a substantial loss of performance. The BitTorrent achieves this by segmenting the pieces in subparts (blocks), typically 16KB, and keeping a certain number of requests, generally 5, queued. Each time a block has been received a new request is sent.

2.1.2.2 Piece Selection

The selection algorithm of pieces downloading is highly important to get good system performance. In fact, by wrong choices there is a risk to reach at a situation in which the pieces of the file owned by the client are not required by any other peer. Consequently, we need to find a strict policy for downloading of pieces that compose the file. The original specification of the protocol requires that clients can download pieces in a pure random way, but subsequently new techniques were introduced to improve performance [56]. The use of these techniques depends on the amount of data possessed by peers at the time of selection. In the following, the techniques are introduced in detail

- *Strict priority:*

Is the first policy used by BitTorrent; if a single block of a given piece has been received, the remaining parts of this particular piece will take precedence with respect to requests relating to a new piece. This technique allows completing as quickly as possible whole parts of the file. Consider the example illustrated in Figure 2.2. The file that shared has been divided into 8 pieces, each consists of 6 blocks. For simplicity, we consider a swarm composed of only two peers; A, and B. Peer A, is a seeder and therefore possesses the whole file and peer B which is a leecher that possesses only the pieces 2, 3 completely and the first block of the piece 1. Before peer B can request for blocks belonging to any other piece, if the strict priority strategy is applied, it has to complete the incomplete piece.

- *Rarest First (RF):*

When a peer decides to select a new piece to download, it always chooses the one with the lowest occurrence within the swarm. The behavior of this rule allows the peer to be competitive, because, the ownership of pieces that have a high demand, allows the peer more easily to swap in return for others. Operating in this manner very lightens the load on the original seeder, especially, when it introduces a new torrent. The client can determine the rarest piece by keeping the information received during the exchange of bitfield with other peers and updating the receipt of each message have. Thus the client may request the piece which has the fewest number of occurrences.

It should be noted that the Rarest First strategy should also include a mechanism for random choice among the less common pieces to avoid that all peers end up with requiring the same rare piece, making this technique less productive. Figure 2.3 is an example of a selection of the rarest pieces. Consider a swarm composed of 5 peers, one seeder, and 4 leechers. After, peer B completing the download of one piece, it have to choose which piece to request later. Since, B has received the bitfield relating to peers A, C, D, and E, and it knows which pieces are owned with each one of them (represented in the figure by the green color and 1). B will calculate the number of occurrences within the swarm for each of the pieces that do not have. It will notice that: the pieces 4 and 6 are held by three peers (A, C, E), and pieces 5 and 7 are owned by two peers (A, D), while the piece 8 is only possessed by peer A. Consequently, the choice will fall on the latter piece, which is the rarest within the swarm.

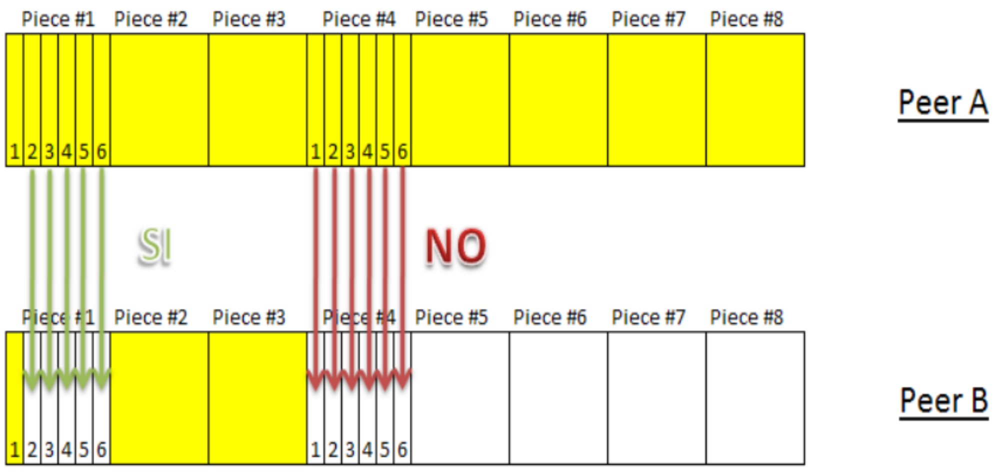

Figure 2.2: Strict priority piece selection strategy.

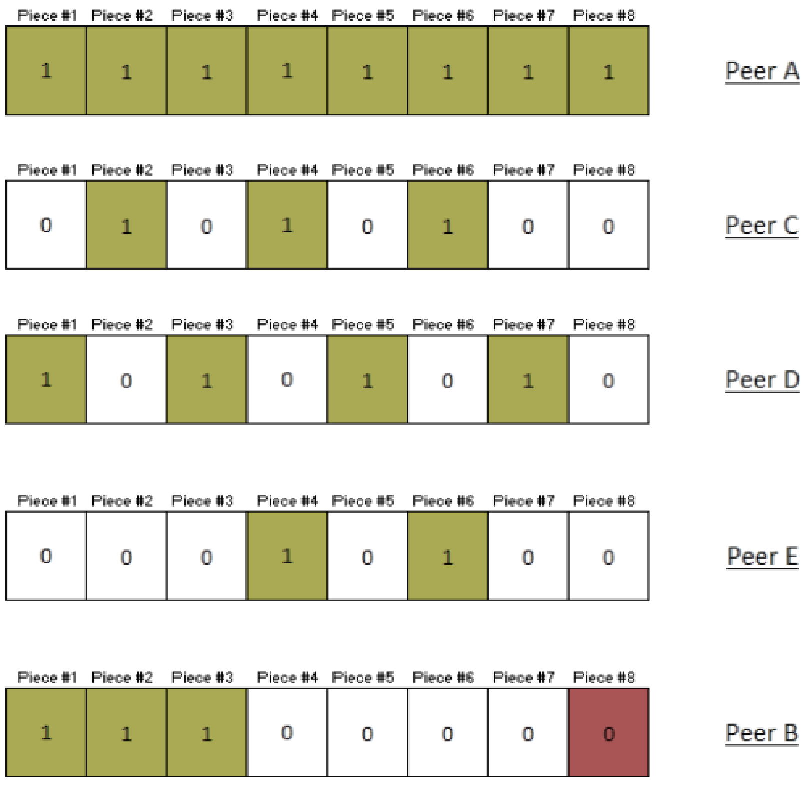

Figure 2.3: Rarest first piece selection strategy.

- *Random first:*

The only exception to the RF rule is happened when a peer starts downloading a file and the peer is not the owner of any piece. In this situation, it is likely that the rarest pieces of a file are in possession of a few peers, and, if the RF policy applied, the download would be slowed down. Consequently, the first piece to be downloaded is chosen randomly and after the completion of the download of this piece, applies RF. It should be noted that it is important that a leecher beginning as soon as possible to send the blocks that owns since download speed depends on its upload. So when a peer has not any block to distribute is preferable to adopt an algorithm that allows him to quickly get a full piece to be exchanged.

- *Endgame mode:*

In some situations, a piece may be requested by a peer with a transfer capability is very low. This situation does not cause particular problems in the intermediate stage of the file downloading, but it can be potentially harmful when it is coming in the end. At the phase of completion of file download it can make it faster if the client broadcasts the request of the last few blocks to all peers to which it is connected. To avoid this situation becomes inefficient, the client will send a broadcast message to cancel each block has been received.
When to enter this mode is still under discussion. Some clients will enter when it have all the required pieces except one, others wait until the number of remaining blocks of the last piece is smaller than those already received, or in any case the number of remaining blocks do not exceeds 20.

2.1.2.3 Choking Algorithm:

In BitTorrent protocol, there is no centralized resource allocation and each peer tries to maximize its download rate. A peer is faced with the problem of cooperation (download/upload) similar to the iterated prisoner's dilemma [38].The choking algorithm ensures collaboration among peers eliminating this dilemma and allowing you to achieve pareto optimality by applying a tit-for-tat strategy.

The act of choking algorithm is a temporary refusal to provide the data, but not to receive them, and then the download can continue and the connection must be renegotiated when this condition is ended.

Every good choking algorithm should follow a number of criteria:

1. Must avoid the occurrence of the phenomenon called “fibrillation” in which there is a fast switch between chocked state to unchocked state that resulting a considerable overhead within the network.
2. Should allow the client to reciprocate the upload bandwidth provided by those peers from which it is downloading (principle of reciprocity).
3. Must be able to check the connections not active to see if there is someone who can be more advantageous (optimistic unchoking).

The phenomenon of fibrillation is avoided by only repeating the selection of choked peers once every 10 seconds (choking_interval).

Each client participates in the file sharing unchokes a fixed number of peers (default 4) among all those which has established with them a TCP connection. The problem is thus reduced to the choice of which of them do not “choke”. The client makes unchoking to four peers that provide the highest download rates and that are interested in it. These four peers are called downloaders.

Peers who have a bettor upload speed than the current downloaders, but aren’t interested get unchoked. If they become interested, the downloader with the lowest upload speed gets choked. It should be noted that if a client becomes a seeder (owns the complete file), it will use its upload rate rather than its download rate to choose which peers to unchoke.

For optimistic unchoking, a single peer is unchoked regardless of its upload speed. Every 30 seconds rotation (optimistic_unchoke_interval), a peer is optimistically unchoked. Newly peers are three times as likely to start as the current optimistic unchoke as anywhere else in the rotation. This gives them a good opportunity to get a complete piece to upload.

2.1.2.4 BitTorrent Tracker:

Tracker is a server that allows peers and seeds to communicate using the BitTorrent protocol. It plays a vital role on BitTorrent application since it can trace out a list of clients that participating in the network. In addition, peers know nothing of each other until a response is received from the tracker. Therefore, the peers connect to the tracker server to obtain the related information about the file that they want to download. The role of the tracker ends once peers have known each other. A BitTorrent client must communicate with the tracker before starting downloading the file as well as during downloading in progress to report their own downloading information and also can gain the new seed information. Trackers use a simple protocol layered on top of HTTP. Moreover, the role of the tracker ends once peers have known each other. From then on, communication is done directly between peers, and the tracker is not involved. Therefore, the bandwidth of the tracker is very low since peers only connect to the tracker for a very short time in long time intervals (usually 30 minutes). The total amount of bandwidth used by the tracker is currently around a thousandth the total amount of bandwidth used [42].

Tracker drawbacks are: it is a single point of failure (SPOF), vulnerable to Denial-of-Service (DoS) attack, and, is limited in terms of scalability and availability.

2.2 Real Time Publish Subscribe (RTPS) Middleware

The publish-Subscribe architecture, Figure 2.4, is a data centric design permitting direct control of information exchange among different nodes in the architecture [44]. It is a sibling of the message queue pattern, and is one part of a large message-oriented middleware system. It generally relies on asynchronous message passing, as opposed to request-response architecture. It connects anonymous messages publishers with anonymous messages subscribers. The property of decoupling publish and subscriber in time (data when you want it), in location (publisher and subscriber can be located anywhere) and in platform (connect any set of systems) make the publish-subscribe communication model more appropriate for formidable scale and loosely coupled distributed Real-Time systems than traditional models such as client-server models[]. Client-server communication drawbacks, e.g., server bottleneck, single points of failure and high bandwidth load in many-to-many communication are resolved by publish-subscribe communication model [43]. Unlike client-server interaction model, data in publish-subscribe interaction model is pushed by the producers to “topics” or “destinations” where consumers will receive all data distributed to the topics to which they subscribe immediately after the data is produced without the need of a request, and thus subscribers and can get the data in Real-Time. In addition, publish-subscribe architecture releases the producer (publisher) from waiting for an acknowledgement by the consumer (subscriber). As a result, the publisher can quickly move on to the next receiver within deterministic time without any synchronous operations which is desirable for a large scale distributed Real-Time systems [3]. Recently, the publish-subscribe communication model has become popular in different middleware such as Java Message Service (JMS), Microsoft Component Object (COM+) and Data Distribution Service (DDS). DDS is a high performance middleware standardized by the Object Management Group (OMG) for QoS-enabled publish-subscribe communication aimed at distributed Real-Time and embedded systems [10].

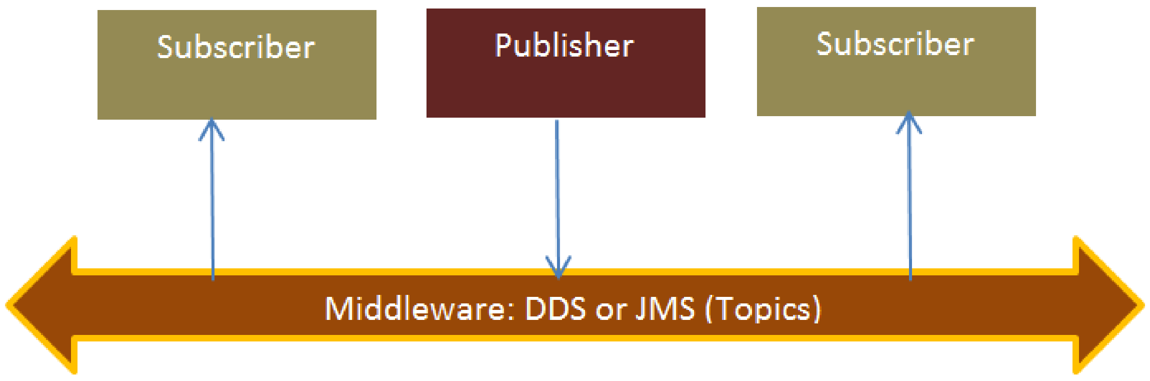

Figure 2.4: Publish/Subscribe (PS) model

2.2.1 OMG Data-Distribution-Service (DDS):

The Object Management Group, Inc. (OMG) is an international organization founded in 1989. The OMG promotes the theory and practice of object-oriented technology in software development [8]. The OMG's goals are the portability, reusability, and interoperability of object-based software in distributed, heterogeneous environments. Ten years ago, the Data Distribution Service (DDS) has been risen as OMG standard for topic-based publish/subscribe Middleware. The OMG Data Distribution Service for Real-Time Systems (DDS) is considered to be the first open international middleware standard directly addressing publish-subscribe communications for real-time systems. The main goal of the DDS specification is to make the dissemination of data in heterogeneous distributed environments efficient and easy [8]. At the core of DDS is the Data Centric Publish-Subscribe (DCPS) layer that is targeted towards the efficient delivery of the proper information to the proper recipients for applications running on heterogeneous platforms [11].DCPS builds on a Global Data Space (GDS), Figure 2.5, by which applications or participants running on heterogeneous platforms can share information by publishing data under one or more topics of interest to other participants. On the other hand, applications or participants can use the GDS to declare their intent to become subscribers and access data of interested topics. Each topic represents a logical channel for connecting publishers to all interested subscribers. DDS has several implementations; these implementations can be categorized as free (open source) such as OpenSplice [45] and OpenDDS [46], and commercial such as CoreDX [47] and RTI-DDS [35]. For our work, we have chosen RTI-DDS middleware due to its efficient implementation [14]. Moreover, DDS is a publish-subscribe standard with a diverse set of Quality of service (QoS that ensures high performance and low delay of transmission).

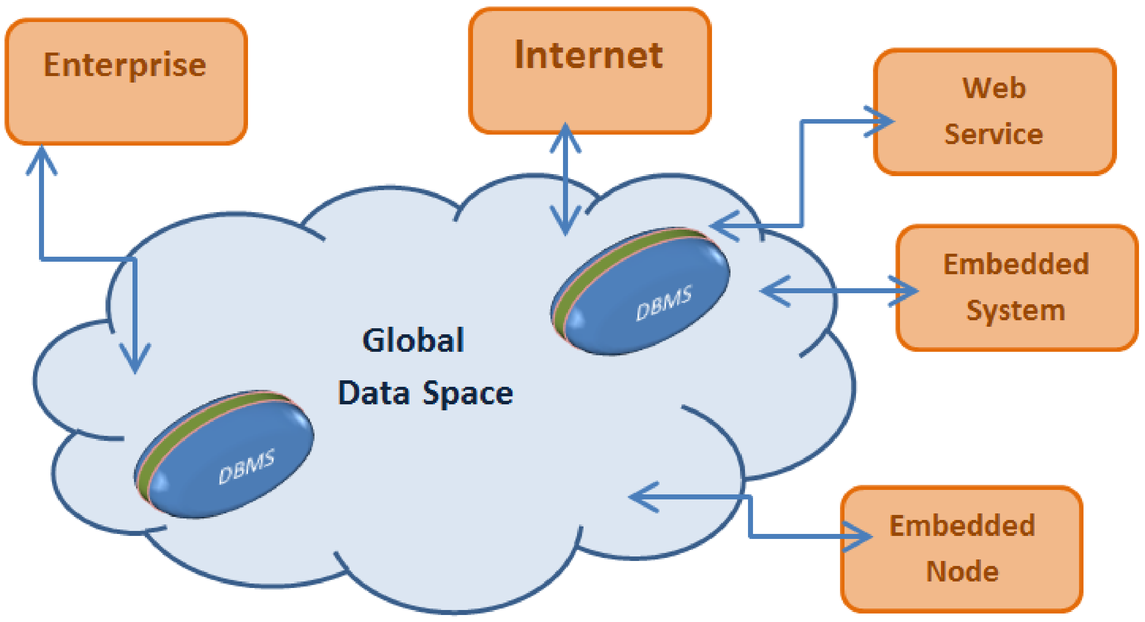

Figure 2.5: Global Data Space (GDS).

2.2.2 DDS Quality of Service Policies (QoS)

Perhaps, the most important advantage of DDS it the fine control over real-time Quality of Service (QoS). DDS relies on the use of QoS to tailor the service to the application requirements. QoS policies are implemented as a list of qualities of service that must meet the component to which it is associated. All components of a communication system may have an associated set of quality of service. The QoS, which is requested by a subscriber, must be met by a publisher. Each publisher-subscriber pair can establish particular quality of service agreements. QoS parameters control every aspect of the DDS model and the underlying communications mechanisms. Many parameters of QoS are implemented as a contract between publisher and subscribers; publisher offers, and subscriber requests, levels of service (like a negotiation mechanism). The responsibility of the middleware is to determine if the offering can match the request, thereby initialing the connection or showing an incompatibility exception.

It has been decided to organize QoS policies in groups, considering the functionality offered or scope of communication in which they operate. Here are the name of each group and its QoS policies:

- Volatility group: contains five QoS policies; Durability (store or not previous published data), History (how much data to store), ReaderDataLifecycle (manages the lifecycle of the data that it has received), WriterDataLifecycle (how Datawriter controls the lifecycle of the instances that manages), and Lifespan (determines how long should consider data sent to be valid).
- Infrastructure group: contains two QoS policies; Entity Factory (controls the behavior of an entity as a factory of other entities), Resource Limits (determines amount of memory is allocated for middleware entities).
- Delivery group: contains four QoS policies; Reliability (controls the protocol reliability e.g. best effort, or reliable), Time Based Filter (specifies a minimum time period before new data is provided to a DataReader), Deadline (related to samples elapsed time), and Content Filters.
- User group: contains three QoS policies; User Data, Topic Data, and Group Data, they attach discoverable meta-data at the writer/reader level, the producer/consumer, the topic level, respectively.
- Redundancy group: contains three QoS policies; Ownership ,Ownership Strength, they specify if a subscriber can get new samples from multiple publishers at the same time, and Liveliness (allows subscriber to detect when publisher becomes dead, or disconnected).
- Transport group: contains two QoS policies; Latency Budget (suggests how much time is allowed to deliver data), and Transport Priority (gives some data different priority than other data).

More details about Quality of Service policies can be found at the DDS specification [10], and RTI QoS reference guide [48]. In4, and chapter 5, we mention wider details about QoS policies that we have been used to accomplish our proposed work.

2.2.3 DDS Discovery

The DCPS model provides anonymous, transparent, many-to-many communications.

Applications that use DDS discover one another in an automatic, dynamic P2P fashion; they do not need any brokers or centralized node in order to send messages. Applications in the discovery mode automatically send announcements to one another when the following events take place: a new connection is created, or a new DataWriter or DataReader is created.

An application goes through an operation titled matching, in which the new publisher or subscriber is compared against the local publishers or subscribers to decide whether or not they can communicate. A publisher and subscriber are considered to be matched if:

- They registered the same Topic.
- And, they have compatible QoS.

After a publisher and subscriber have been matched, data published by the publisher will begin to be received by the subscriber.

The consumed time for connections to discover one another and for publisher-to-subscriber matching to accomplish is on the order of one or two seconds when all peers are in the same subnet considering modestly system's size. Based on the number of hops, the loads on the network, the system size, and, the load on the target CPUs, this time can differ greatly [12] [49].

CHAPTER 3

LITERATURE REVIEW

In general, research studies in BitTorrent peers Discovery are limited in number. The most three important solutions for tackling tracker availability problem are: multiple central trackers, Distributed Hash Table (DHT), and Peer Exchange (PEX)

3.1 Multiple Central Trackers

Since, the first version of BitTorrent, the idea of multiple trackers [52] has been used. A single torrent file contains multiple trackers addresses. This enables redundancy, if one tracker goes down, then the remaining trackers can go ahead to reply new coming peers with the peers list. Multi-tracker has introduced two disadvantages: 1) it becomes possible to have multiple swarms for one torrent where some peers can join to a special tracker whereas being unable to bind to another [51]. 2) Extra resources (multiple servers) are needed, and much time is needed to replicate the torrents file. The main advantage of this approach is that the network (search) complexity is *O(1)* or in the worst case is *O(S)*, where S is the number of tracker servers, whereas, the complexity terms of memory consumption is *O(N)*, where N is the number of torrent files available in the tracker. An example of multi-tracker environment is depicted in Figure 3.1.

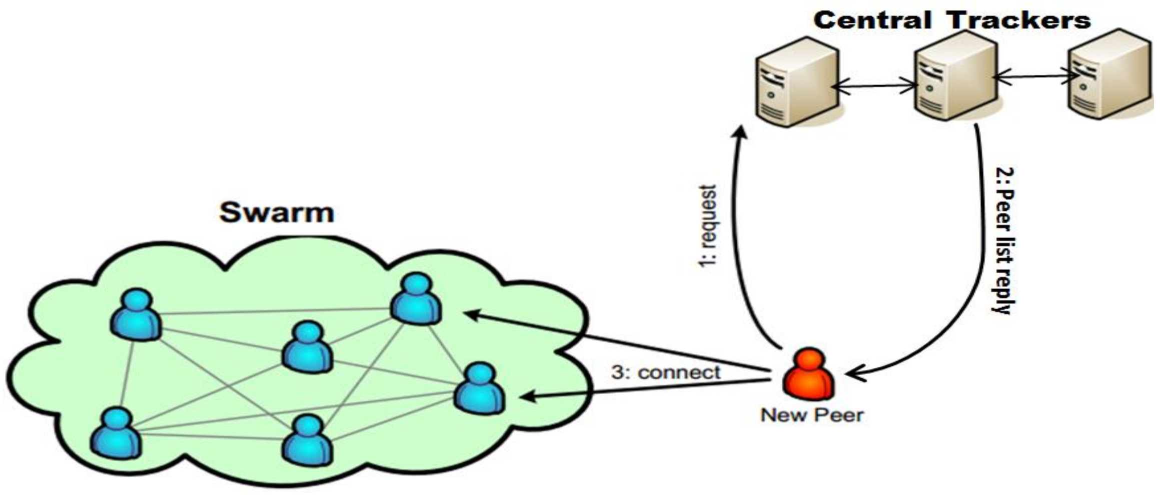

Figure 3.1: A new peer connects to a swarm using multiple central trackers approach [55].

3.2 Distributed Hash Table (DHT):

Andrew et.al [50] proposed a DHT BitTorrent for storing peers information for, what so called, "Trackerless" torrents. The standard and implementation of DHT protocol is described on the official website of BitTorrent [50]. Terms peer, node, DHT, distance metric, and routing table all refer to members at the P2P network. In brief, terminology and mechanism are illustrated as follows:

A. Terminology:-

- Peer: is a client or a server listens on TCP port implementing BitTorrent protocol.
- Node: is a client or a server listens on a UDP port that implements the DHT protocol.
- DHT: is consists of nodes and stores the site of peers.
- Node ID: is a unique identifier assigned to every node. This identifier is chosen randomly from the same 160-bit space.
- Distance metric: is used to compare two node IDs or node ID and infohash for closeness.
- Finger Table (Routing Table): contains the peers IDs for a small number of other nodes in the system.

B. Mechanism:

Firstly, a node, which tries to locate peers for a specified torrent, compares the infohash of the torrent file with the IDs of the nodes in its finger table. This comparison is done by using the distance metric. Secondly, the node checks out the nodes it knows about with IDs closest to the infohash and inquires for the communication information of peers in the swarm. Thirdly, if a connected node has knowledge about peers for this torrent, a response with the peer communication information is returned. Otherwise, the connected node has to answer with the communication information of the nodes which are closest to the torrent's infohash. Finally, the original node iteratively inquires nodes which are close to the target infohash till it cannot find any closer node.

DHT still need a bootstrap node which is a node provides initial configuration information to newly joining nodes so that they may successfully join the overlay network . The storage and search complexity of DHT should not increase than O(logN). An example of new peer joining a DHT overlay network is depicted in Figure 3.2.

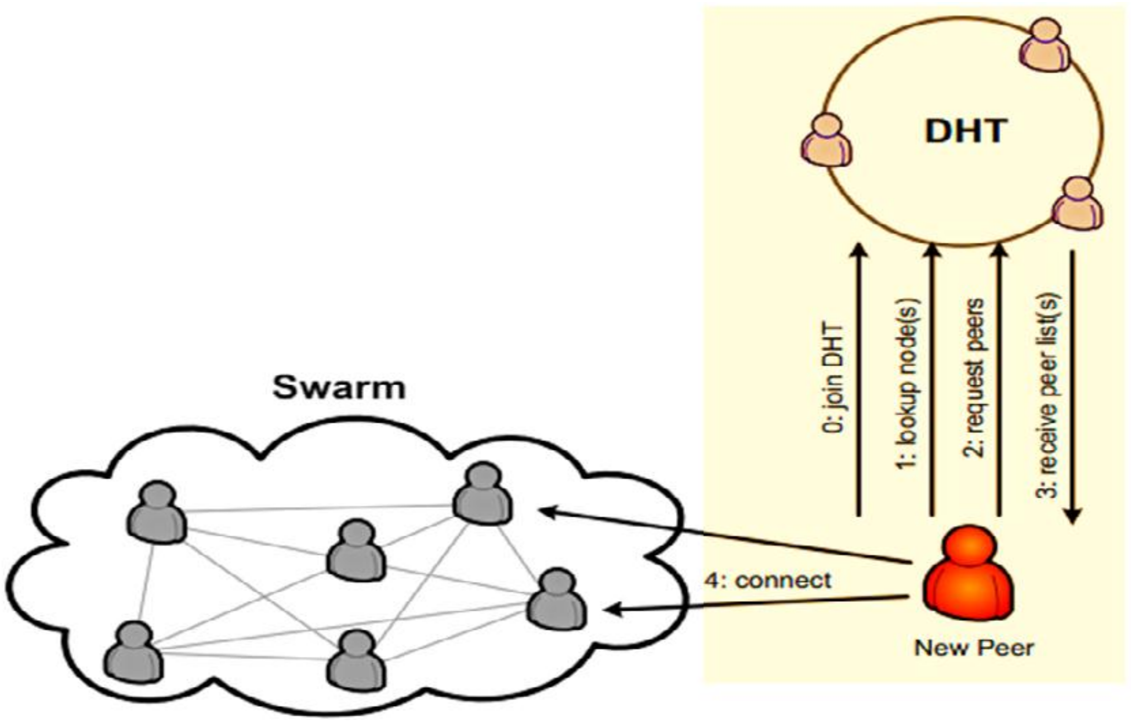

Figure 3.2: BitTorrent’s DHT environment (A new peer joining process) [55].

3.3 Peer Exchange (PEX):

Another method to design a distributed tracker is by using a gossip protocol like PEX. PEX [53][54] is an extension to BitTorrent standard protocol aimed to accelerate nodes discovery. Instead peers connect the central tracker to update their peers list; a peer can share his own neighborhood group with his neighbors. After a peer has interchanged his own peer lists with another peer, it might connect to the newly discovered peers.

The steps of PEXing process are as follows. For each PEX-capable link, a peer maintains a group of peer addresses it has already sent to the other entity. When a peer decides to send a new PEX message, it sends the difference between its current neighborhood set and its set of peers already sent, or a subset hereof if the resulting set it too big. Computed using these same two sets, the same PEX message also contains a group of previously sent peers which the peer is no longer bounded to since the last PEX message.

In PEX, the need of tracker is not eliminated completely. Actually, PEX just lessens the load in the tracker. Each new joining peer must connect the tracker in order to get its first peers list. Figure 3.3 illustrates the PEX process graphically.

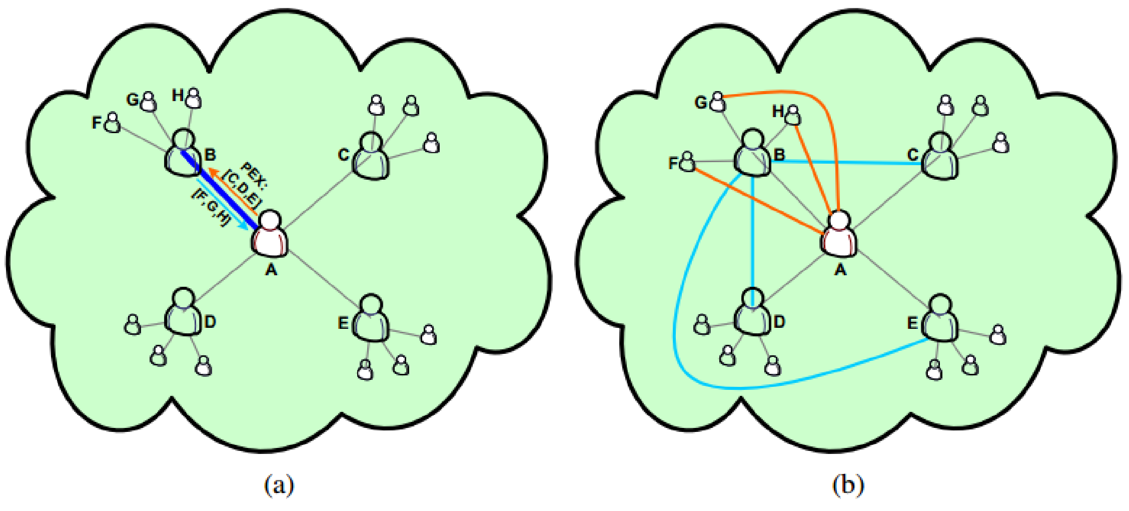

Figure 3.3: (a) Peers A and B change their lists information (b) Peers A and B can contact the discovered peers directly [55].

3.4 Other Related Methods:

Fabio V et.al [6] proposed balanced Tracker (B-Tracker), a pull-based, and fully decentralized tracker. It refers to a seeder as a provider. In initial stage, B-Tracker depends on a DHT overlay structure for tracker detection. In this stage, the primary trackers, which are peers have peerID nearest to the resourceID, are responsible for storing the set of providers (seeders) of the resource. The complexity of essential trackers discovery is *O(log n)* , where n refers to the number of nodes (peers) in the network. It is not always true that the primary trackers of resources are providers for these resources. In next stages and after a peer has got a provider list from the primary tracker, subsequent queries can be sent to any provider. A provider is considered a secondary tracker for the resources it provides. The idea of secondary trackers makes B-Tracker scalable, because resources with many providers are capable of spreading the load among primary trackers, as well as, secondary trackers. Also, fairness can be improved by sharing the load among peers who interested in providing the resource. Simulation of B-tracker showed that it achieves better higher efficiency and load balancing than the other distributed trackers (DHT and PEX).

Lareida et.al [4] proposed RB-Tracker. RB-Tracker is essentially based on B-Tracker [6]. However, RB-Tracker manages the overlay network automatically and does replication of content. The motivation of RB-Tracker is to minimize traffic peaks and inter-domain traffic. The main idea of RB-Tracker is that in non-peak hours duplicating the content like a CDN and utilizing locality. In a fully distributed network, RB-Tracker combines three methods. First, replication of popular and of interest to the user content to the local cache in order to make files closer to their users. The goal of this is to reduce peak loads, consequently, favorite files are duplicated, since they causes the dominant traffic. Second, identify the status of the network to choose the replication time. This is done by sending messages; contain a time stamp flag, between peers. A peer measures delay and builds statistics based on its measurements, as a result, a peer can identify when delay is rising. If the delay is raised between two neighbor peers, the neighbor is not an appropriate source for duplication and another neighbor needs to be detected. Third, determines the group of close peers to duplicate from, as a result, additional intra domain traffic is avoided. For two IP addresses, a locality function is used to identify how two peers are close. A peer decides, if it should duplicate from a neighbor peer using the number of AS hops between them. A trace route tool with an IP to AS map is used to find the AS hop distance vector. An example which illustrates this mechanism can be found in the original paper.

Charles P et.al [2] shown BitTorrent swarm at any point in time as a simple graph G=(V,E) V={1,....,n) is the set of peers and E (V x V) is the set of neighbor relations. Random walk mechanism, which it used to randomly select nodes from graphs. Using biased random walks to select initial neighbors for joining nodes and to replace failed nodes. As consequence, it removes any dependence on the tracker. This accomplish by Entry Points which help new joining nodes to get a random set of neighbors. So, Entry Points play the same function of the tracker. Communications between entry points is not mandatory, they can operate concurrently. The authors compared graphs generated by their approach to those created by the centralized tracker, two logs are used, RedHat tracker log, and Debian tracker log.

Ioanna et.al [1] proposed ISP-Owned Peers (IoPs) to enhance BitTorrent Performance. IoP is a node which targets to increase the level of traffic locality within an ISP and to improve the performance of P2P applications. IoP could be a regular entity but highly active peer (HAP) that is given extra resources by the ISP, or could be as a part of an ISP infrastructure, so, it's controlled by the ISP. IoP runs the typical overlay protocol like others peers in the swarm with some parameters changes that benefit other peers, e.g. IoP can unchoke more peers than the classical ones, in order to take the advantage of its extra uplink capacity. Also, it can store the content downloaded and of course uploading it back to the network. There are two methods to deploy an IoP in BitTorrent network:

A. Plain insertion: BiTtorrent original protocol is run by all peers; there is no mechanism like awareness of locality is employed, and no consideration of any agreement with the overlay provider. Therefore, the tracker is not conscious about the existence of IoP as a specific peer but deals it as a regular entity. In this approach, the other peers prefer the IoP because of the tit-for-tat mechanism run by unchocking algorithm and due to its high uplink capacity. The IoP follows the tit-for-tat principle taking the advantage of the immediate incentives of the latter that are directly related to the underlay.

B. Integration with locality awareness methods: the run of locality awareness methods that impact the overlay network's structure is considered as being imposed by the ISP. The implementation of these methods could be either: 1) clear (transparent) to the peers (run the same original protocol) or 2) non-transparent (an adjusted version of the protocol is introduced). Metrics that can be considered are RTT and hops' number to remote peers, the identity of peers' autonomous system, and BGP information. According to these metrics, the IoP is mostly preferred by peers that are 'closer' to it.

Simulation has shown that the deployment of the IoP achieves good decrease in the inter-domain traffic that get in the AS where it is deployed. Further enhancement is achieved when the IoP deployment is integrated with locality-awareness mechanisms. Moreover, the deployment of IoP in a pure BitTorrent overlay network within the use locality-aware leads to higher reduction in the inter-domain traffic.

3.5 Research Objective

The aims of this research are as follows:

- Removing BitTorrent Tracker completely, and distributing the tracker role among swarm peers. Making BitTorrent protocol a pure P2P protocol.
- Propose a novel architecture for BitTorrent discovery protocol.
- Reduce both communication overhead (network overhead) and node searching complexities to *O(1)* for most cases.
- Implement BitTorrent dissemination protocol using RTPS middleware (DDS). For our best of knowledge this is the first implementation for BitTorrent over DDS.

- Benefiting from publish/subscribe paradigm in minimizing BitTorrent overhead messages.
- Compare the DDS-BitTorrent performance with the original BitTorrent protocol.

CHAPTER 4

DYNAMIC POINTERS: NOVEL DISCOVERY PROTOCOL FOR BITTORRENT BASED ON RTPS MIDDLEWARE

Based on the problem of the BitTorrent tracker stated in section 1.4, we formalized the problem in this chapter and present our implementation architecture. For our discovery architecture, we benefit from tuned RTI-DDS discovery scenario which is called "One-way Communication with high Fan-Out"[27], and we use it as a skeleton to implement our discovery architecture. The following section will summarize this scenario and how it is related to our architecture.

4.1 Discovery Scenario: "One-way Communication with High Fan out"

In this scenario that is shown in Figure 4.1, a standalone publisher distributes messages to a large number of subscribers. These subscribers do not interchange any messages with each other. As a result, the performance of these subscribers will be improved and the memory footprint of them will be decreased, since, they do not need to discover each other. Therefore, IP unicast is used to send messages from the subscribers to the publisher. It should be noted that the default DDS discovery uses IP multicast, in which, all peers in the network discover each other. By using IP unicast, we ensure that the network interfaces and CPUs of the other subscribers computers will not be burdened with the additional multicast traffic. Messages sent from the publisher to the subscribers are distributed using IP multicast for general messages, and IP unicast for subscriber's specified messages.

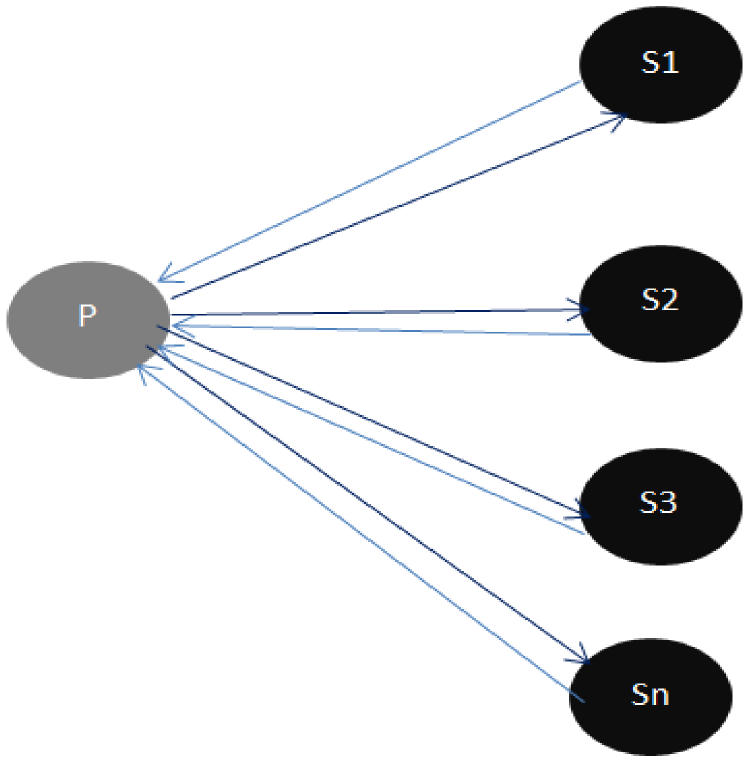

Figure 4.1 RTI-DDS discovery scenario :"one way communication".

The QoS related to the discovery configuration to achieve this scenario for both publisher and subscribers are shown in Figure 4.2 and Figure 4.3, respectively.

```
<discovery>

      <initial_peers>
<!-- Multicast address to talk to subscribers:-->

            <element>239.255.0.1</element>
        </initial_peers>
        <multicast_receive_addresses>
        <!-- Empty: only listen over unicast -->
        </multicast_receive_addresses>

</discovery>
```

Figure 4.2 Publisher discovery QoS xml configuration

```
          <discovery>

          <!-- just talk to the IP which was included in < initial peers> list -->
          <accept_unknown_peers>false</accept_unknown_peers>
          <initial_peers>
          <!-- Publisher's unicast address -->
                  <element>192.168.1.100</element>
          </initial_peers>
          <multicast_receive_addresses>
                  <!-- Multicast address to talk to publisher: -->
                          <element>239.255.0.1</element>
          </multicast_receive_addresses>
          </discovery>
```

Figure 4.3: Subscriber discovery xml configuration

BitTorrent discovery protocol typically works in the same manner as the above scenario, in which, peers only interact with the tracker, and they know no information about each other.

4.2 Dynamic Pointer Quality of Service

For more information about DDS QoS policies see section 2.2.2, and RTI QoS reference guide [48]. In this section, we describe in details about two DDS QoS policies; Ownership and Domain.

4.2.1 Ownership and Ownership-Strength QoS's

Ownership QoS provides fast, robust, transport replacement for failover. By default, subscribers can get data from any matching publisher for the same topic; this is known as the "shared" setting for the Ownership QoS policy. If the "exclusive" setting is used, subscribers only receive data from one publisher at a time. The setting of Ownership in the subscriber side must be the same on the publisher side to be connected. Either both sides must be shared or both sides must be exclusive. When the setting of the Ownership is "exclusive", we use the Ownership-Strength QoS policy to specify which publisher is the owner of the data (allowed to send data). The publisher who has the highest value for the Ownership-Strength is considered as the owner of the data. When there are many publishers, and the publisher with the highest value of ownership strength leaves, the middleware will change the ownership of the data to a publisher with the highest ownership-Strength from the remaining publishers.

An important point to be known about the Ownership QoS policy is that it is a network overhead. Actually, data is sent by all the publishers, and the middleware at the subscriber side drops all data except these that are sent by the owner publisher (the strongest publisher). In fact, the filtering process is done at the subscriber side after consuming and wasting the network bandwidth.

4.2.2 Domain QoS

A domain is a logical network that overlays the physical network. Every connection in DDS application belongs to exactly one domain; therefore, domains form a technique for isolating subsystems or entire distributed applications from one another. A unique integer value, domain ID, is used to distinguish one domain from another. An application participates in a domain by creating a DomainParticipant for that domain ID. A domain creates a “virtual network” linking all applications that share the same domain ID. Applications run on the same set of physical computers and share the same physical network but using different domains are isolated from each other.

4.3 Dynamic Pointer

From the BitTorrent specification, we know that the availability of the initial seeder that has all the pieces that make up the content is mandatory for the success of the file sharing process. This principle is maintained in our proposed solution with a minor modification to the overall operation. Instead of adding the IP of the tracker in .torrent file, the seeder puts its IP address so that the new joiners contact the original seeder directly. The seeder plays the role of the tracker in terms of coordination and updating the swarm list. “Self-tracker” is the name of the initial seeder and other seeders.

The Dynamic Pointers architecture benefits from the “One-way communication with High Fan-out” discovery scenario, discussed in section 4.1. We use the publisher as the initial seeder (self-tracker) and the other new joiners as subscribers. Each new joiner needs to contact only the initial seeder to get its swarm list using IP unicast. The details of how the Dynamic Pointers architecture works are as follows:

1. Initial seeder shares the .torrent file which contains its IP address and domain-ID on a web server or via email and waits new joiner peers to contact it.
2. A peer that is interested in that file downloads the corresponding .torrent and contacts the initial seeder (self-tracker). Then, the peer subscribes to a DDS topic has the same name of the .torrent file, the connection established by using IP unicast. The self-tracker publishes to the new joining peer the swarm list (see Figure 4.4).
3. Now, if a peer completes the file download (say peer B in Figure 4.4), it starts its publisher (seeder) application, and publishes a unicast message to the self-tracker (peer A) telling it that “I’m a seeder”.
4. The initial seeder (peer A) would publish multicast message to all other peers telling them that peer B became a seeder. Consequently, all peers save the IP address of peer B in a special list (trackers list).

5. After that, peer B changes its discovery mode from unicast to multicast and, adds and publishes its IP address with .torrent file.
6. Subsequently, if the self-tracker (Peer A) leaves the swarm or crashes for some reason (see Figure 4.5 (a)), then the other peers will know that their self-tracker is not available by missing its heartbeats (timeout period). So, they invoke their tracker-list and get the IP address of peer B and contact it, Figure 4.5 (b), by adding B's IP to "initial_peers" element in discovery QoS.

The previous steps repeat themselves and the process continues in this manner. Also, note that the configuration and failover in step six are done completely automatically without any intervention.

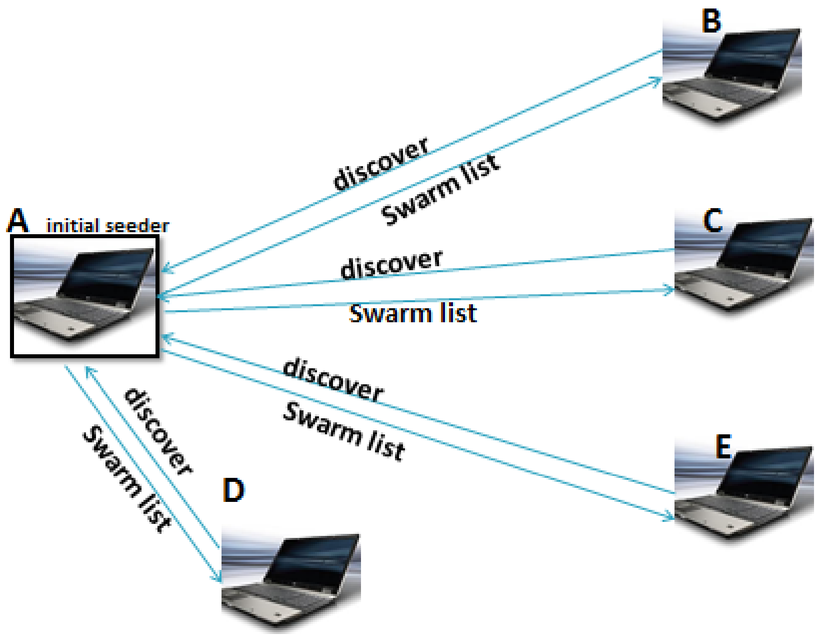

Figure 4.4: Discovery mechanism in Dynamic pointers approach

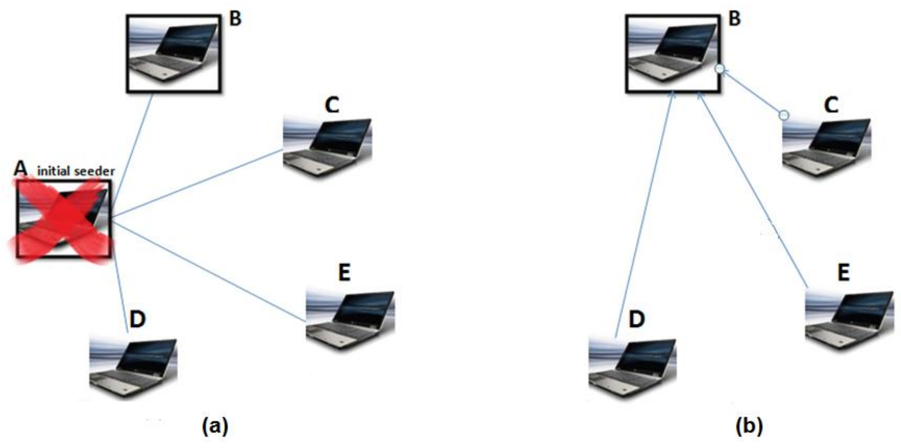

Figure 4.5: (a) Failure of initial seeder, (b) Peers contact directly to the new self-tracker

When there are more than one seeder, new joiners have the option to contact any of the available seeders. For example (see Figure 4.6), if peer F is a new joiner, it may contact seeder B while new peer G may contact seeder A. Additionally to the swarm list, the particular seeder sends the IPs of other seeders in the swarm (A sends IP address of B to G) to make the new peer ready for any changes in the swarm (e.g. leaving or crashing of designated seeder). The peer will be ready to apply step six.

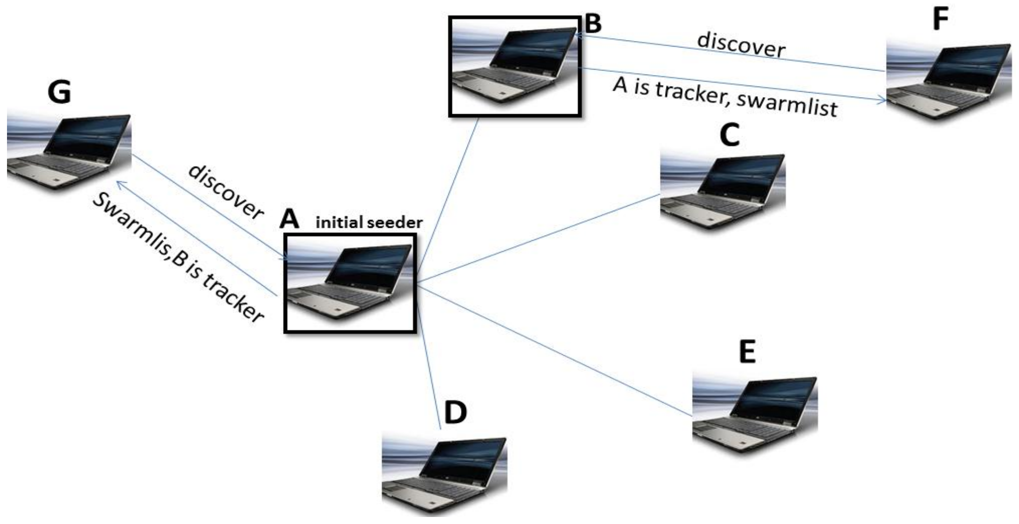

Figure 4.6: new joiners can easily contact any seeder (self-tracker).

The formula of the timeout period, that is used to determine that a designated seeder is disconnected, is depicted in Equation 4.1 and Equation 4.2, and the pseudo code for calculating and adjusting this period is depicted Figure 4.7.

$$\text{initally:} \qquad \text{timeout} = t_0 + \mathrm{X} \qquad (4.1)$$

$$\text{timeout} = \max(t_i, t_{i-1}) + X \qquad (4.2)$$

t_i is the arrival period of the heartbeat messages, i=0, 1, 2, …, e.g. the first heartbeat message arrival time is t0 and so on, and X is an extra period constant used as an assurance period before deciding the collapse of the self-tracker (a seeder); the value of X is determined according to the system size. If the timeout period is expired and no heartbeat message arrived then the current self-tracker is considered not available.

```
// initially set the timeout period to a large period (y)
sufficient to the particular network (consider multi-hops).

1. BEGIN
2.  SET timeout←y;  x ←3 sec;  i←0;
3.  START a timer T

4.   REPEAT
5.     $t_i$ ← read(timer T value)
6.     IF HBi  received THEN   //a heartbeat message is rec
7.         IF i equl 0 THEN
8.             timeout ← $t_i$ +x
9.         ELSEIF  $t_i > t_{i-1}$ and  $t_i < t_{i-1}$+x THEN
10.                      timeout ← $t_i$ +x
11.                  ElSE
12.                  $t_i = t_{i-1}$
13.                  timeout ← $t_i$+x   // not change
14.                  ENDIF
15.                  RESTART T
16.                  INCREMENT i
17.        ENDIF
18.  UNTIL $t_i$ >  timeout   // same as $t_i > t_{i-1}$+x
19.  CONNECT new Seeder.
```

Figure 4.7: timeout period adjusting pseudo code.

The main feature of our proposed solution is fully decentralized for file dissemination, auto configuration, and, instantaneous failover, which achieves high scalability and high availability. The complexity of the proposed solution also has an improvement over multiple trackers, DHT, and PEX methods. The communication overhead (network overhead) is *O(1)* because the requester just has to know one seeder. The complexity in terms of memory consumption is *O(1)* since, the seeder usually has one .torrent file, and if the seeder has more than one file, the seeder can isolate them from each other by using the Domain QoS (designate a unique domain ID for each file).

The key limitation in our solution is the dependence on the availability of seeders. The system may fail if no seeder is found. The following two subsections describe enhancements to the dynamic pointers architecture

4.3.1 Guarantee Discovery using IP Multicast

It may happen that a new peer gets a .torrent file with an IP address of a seeder that crashed or left the swarm. In this case, this new joiner can't join the swarm and download the file.

To solve this problem we have made an addition to the .torrent file. We add a multicast IP address beside the unicast IP address of the seeder. This multicast group is essentially joined by the seeders of the swarm, and the discovery scenario becomes as follow:

- Initially, the new joiner tries to connect to the self-tracker (initial seeder) directly using the unicast IP address.
- If the new joiner fails for any reason, it uses the multicast IP address and searches for any other seeder.
- As a new seeder is found, the new joiner gets the IP address of the new seeder and contacts it using IP unicast approach.

The flowchart of this process is shown in Figure 4.8. Using this approach, the discovery is guaranteed as long as, there is a single seeder.

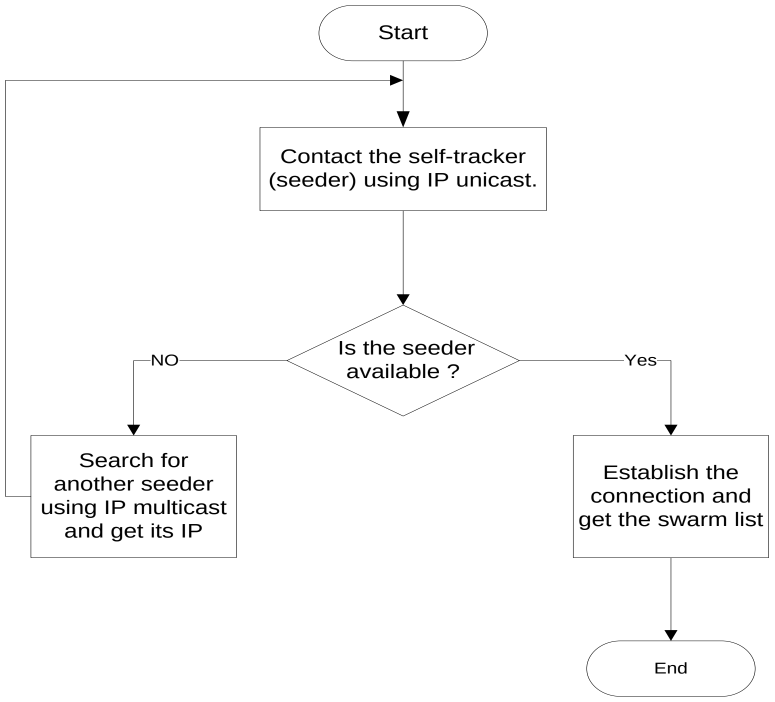
Start
Contact the self-tracker (seeder) using IP unicast.
Is the seeder available ?
NO
Yes
Search for another seeder using IP multicast and get its IP
Establish the connection and get the swarm list
End

Figure 4.8 : Guarantee discovery process flowchart.

4.3.2 Recovery Speed up Using Ownership QoS

The recovery process that is used in the Dynamic Pointers architecture can be accelerated by doing two things:

A. Set the setting of Ownership QoS to “exclusive” for all the publishers (seeders) and all subscribers (leechers), and vary the values of the OwnershipStrength QoS for all seeders.
B. Instead of storing the IP of a new seeder in the trackers list, the subscribers should connect to the new seeder immediately as they receive its IP.

Applying Ownership QoS policy in this situation is reasonable, and the Ownership QoS’s drawback (network overhead) could be ignored, because the messages between peers and the self-tracker in the discovery phase are very small in the size. But if the system is huge and the bandwidth is limited, Ownership QoS should be carefully used.

4.4 Implementation and Experimental Work

Java JDK version 7 [33] as a programming language, NetBeans version 7.4 [34] as an environment, and RTI-DDS API [35] as a middleware are used to complete our proposed solution implementation. We benefit from tuned RTI-DDS discovery scenario which called "One-way communication with High Fan-out" [27] and we use it as a reference to implement our discovery architecture. The testbed of our experiment consists of four PCs connected to each other via LAN. The specifications for these PCs are Intel® Core i3 2.93 GHz CPU, 4 GB RAM, and Windows 7 32-bits OS. Each of these PCs has its own IP and they are connected to Internet as well. Scalability and Availability are the two metrics that we focus in them.

For scalability test, we first run a single publisher (seeder) then, we run new subscribers (leechers) sequentially to test how latency delay increases as the new leechers increase. Two groups of leechers are selected. The first group, which is small, contains subgroups which are one-leecher, two-leechers, four-leechers, and eight-leechers. The second group, which is large, contains subgroups which are twenty-leechers, thirty-leechers, forty-leechers, and fifty-leechers; these numbers of leechers are selected to cover wide range of possibilities of leechers per a swarm. Packets of sizes 2,4,8,16,32 KB are used; these packet sizes are suitable for BitTorrent discovery phase where the exchange messages are only small text messages. 100 samples of each packet size are sent to every leecher in the subgroups then the average delay is calculated. The total trials that are done in the test are 40 trials (number of subgroups x number of packet sizes).

For availability test, we ran two seeders and some leechers. Then, we intended to crash one of the seeders to know how much time required for the leechers to transfer automatically to the other seeder. Three scenarios of this test have been done:

- When the seeder and the leechers are in the same network segment,
- When the seeder is two hops away from the leechers,
- When the seeder is five hops away from the leechers.

In each scenario, we have done five trials. Each trial has been repeated ten times and the average was taken. The last two scenarios are accomplished by using Graphical Network Simulator (GNS3) simulation tool [36], which, provides capabilities to design and simulate a complex network while being as close as possible to the way real networks perform.

Also, availability test is re-experienced when the Ownership QoS is applied.

4.5 Evaluation and Results

First, we have compared the complexity of our proposed approach with Centralized Tracker BitTorrent, DHT BitTorrent and PEX. Table 4-1 summarizes the complexity of our solution versus the others three approaches.

Table 4-1 Complexity of dynamic-pointers Vs. others approaches

Approaches	***Memory consumption***	***Network overhead***
MCT	O(N)	O(1)
PEX	O(N)	O(1)
DHT	O(logN)	O(logN)
Dynamic-Pointers	O(1)	O(1)

For the scalability test, the results come as expected. As the number of leechers increases, the latency of messages from the seeder to these leechers increases. However, the latency increase is due to the increase of the leechers is smooth and it has no harmful effect on the scalability of the network. Table 4-2 and Figure 4.9 show the latency (μs) results in case of 1, 2, 4 and, 8 leechers, respectively. We can see from the table that the average latency proportional to the number of leechers and that makes sense because when the number of leechers increases, the network traffic increases and then the packet latency will increase as well. We can also see that the latency increase is not sharp (the difference between sending 32-Kbytes to one leecher and eight leechers is less than 0.5 millisecond).

Table 4-3 and Figure 4.10 show latency (ms) results in case of 20, 30, 40, and 50 leechers, respectively. We can see that the difference between sending 16-Kbytes packet to twenty leechers and fifty leechers is about 1.5 ms.

Table 4-2 Latency (µs) variations for small number of leechers

Packet size(KB)	# of Leechers			
	1-Leecher	*2-Leechers*	*4-Leechers*	*8-Leechers*
2K	228.9	341.75	386.1	561.65
4K	275.7	331.45	398.6	532
8K	289.4	322.0	385.0	592.55
16K	325.35	332.15	488.35	669.9
32K	346.65	356.0	495.85	763.25

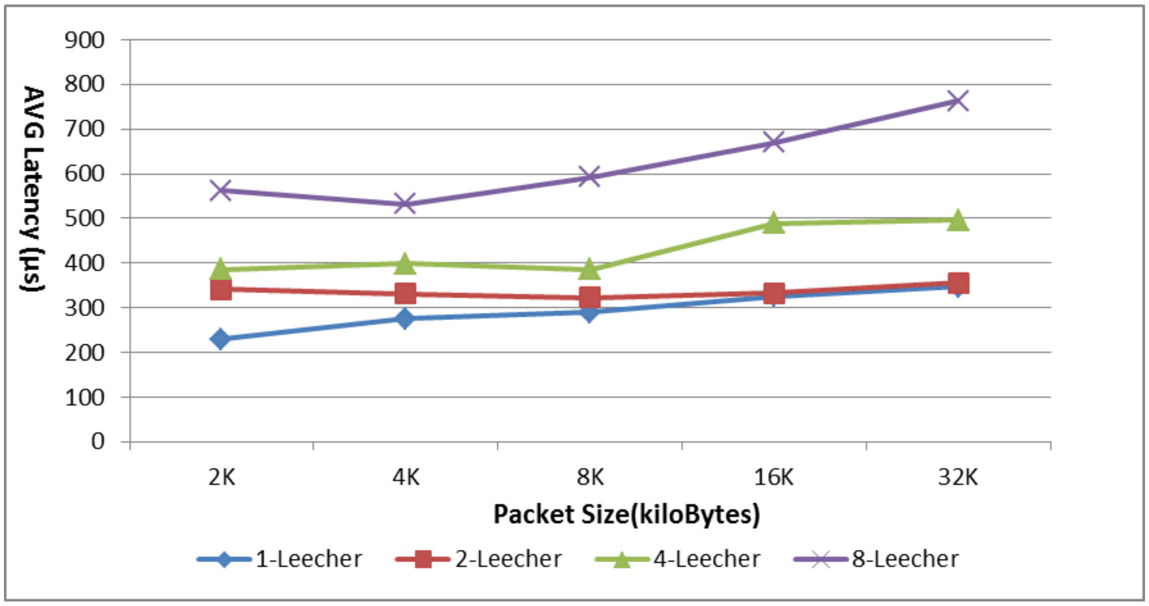

Figure 4.9 Latency(µs) Vs. Packet size for different numbers of leechers

Table 4-3: Latency (ms) variations for large number of leechers

Packet size(KB)	# of Leechers			
	20-Leecher	30-Leechers	40-Leechers	80-Leechers
2K	1.121	1.66855	1.8284	2.34275
4K	1.0801	1.5788	1.8288	2.2403

Packet size(KB)	**# of Leechers**			
	20-Leecher	30-Leechers	40-Leechers	80-Leechers
8K	1.2074	1.61725	1.8778	2.5099
16K	1.1988	1.9245	1.9638	2.63445
32K	1.36155	1.93405	2.1324	2.99125

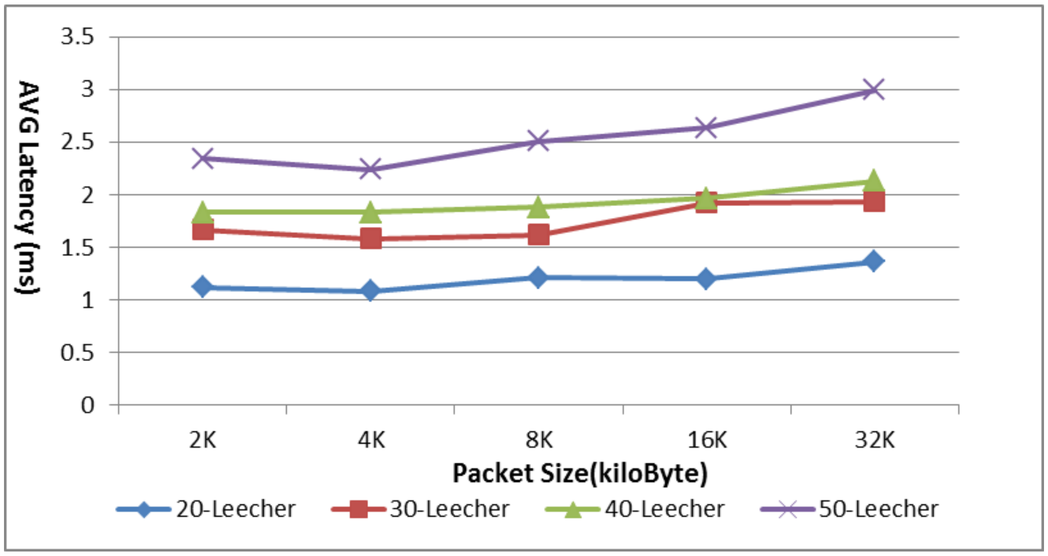

Figure 4.10 Latency (ms) Vs. Packet size for different numbers of leechers

For the availability test, the results show that our network could automatically failover when a crash or a failure occurs. Table 4-4 and Figure 4.11 show all the results. Figure 4.11 (a), where the seeder and the leechers are in the same subnet, shows that the recovery period ranging from 30 to 60 seconds. Figure 4.11 (b) where the seeders are two hops away from the leechers, shows that the recovery period is in between 50 and 70 seconds. Figure 4.11 (c) where the seeders are five hops away from the leechers, shows that the recovery period starts from a minute and half, whereas, the network is congested, the recovery period falls in between two minutes and half to three minutes. These results are very suitable for BitTorrent where a peer contact the tracker one time every 30 minutes for coordination.

For the availability test when the Ownership QoS is applied, the results show a significant enhancement over the previous availability test. Table 4-5and Figure 4.12 show all the results. Figure 4.12 (a), where the seeder and the leechers are in the same subnet, shows that the recovery period ranging from 8 to 10 seconds. Figure 4.12 (b) where the seeders are two hops away from the leechers, shows that the recovery period is in between 10 and 14 seconds. Figure 4.12 (c) where the seeders are five hops away from the leechers, shows that the recovery period mostly falls in 15, or 16 seconds, whereas, the network is congested, the recovery period falls in 30 seconds. Figure 4.13 shows a comparative study between applying Ownership QoS and without applying it.

Table 4-4 AVERAGE RECOVERY TIME (SEC) PER EACH TRIAL FOR DIFFERENT HOPS

Trials	**Distance between Seeders and Leechers**		
	0-hops	***2-hops***	***5-hops***
Trial#1	44.5	68.9	94.6
Trial#2	25	61.1	102
Trial#3	33	62.2	112.3
Trial#4	62.25	53.5	134.5
Trial#5	40	63.5	164.5

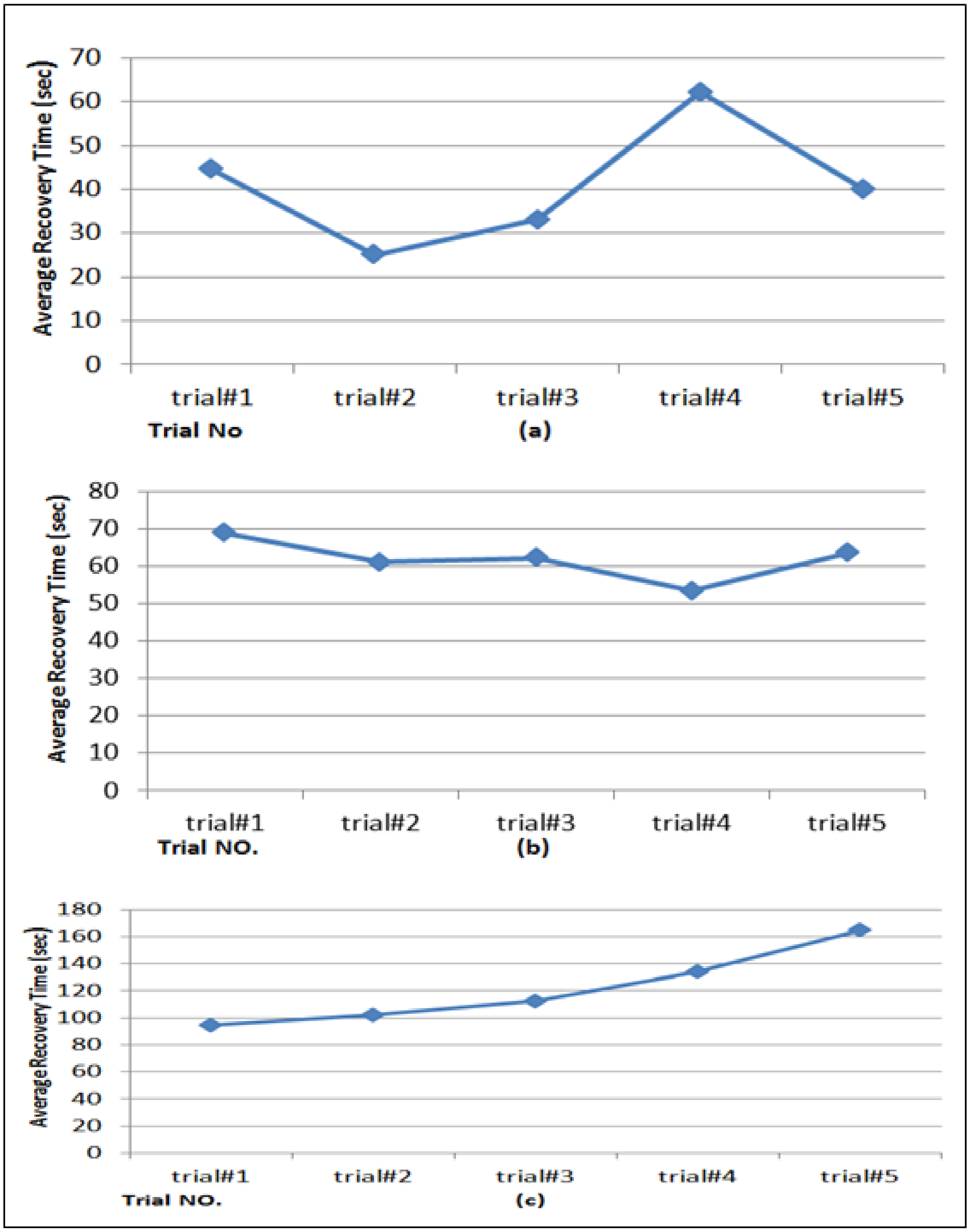

Figure 4.11 Average recovery time (sec) in each trial for different distance scenarios (a) 0-hops (b) 2-hops (c) 5-hops

Table 4-5: Average recovery time (sec) per each trial for different hops (applying ownership QoS)

Trials	**Distance between Seeders and Leechers**		
	0-hops	***2-hops***	***5-hops***
Trial#1	9.1651	10.0126	30.6984
Trial#2	8.6454	12.8452	15.9066
Trial#3	8.9528	10.2288	15.0708
Trial#4	8.8468	14.3095	16.1462
Trial#5	8.9468	13.3884	15.5934

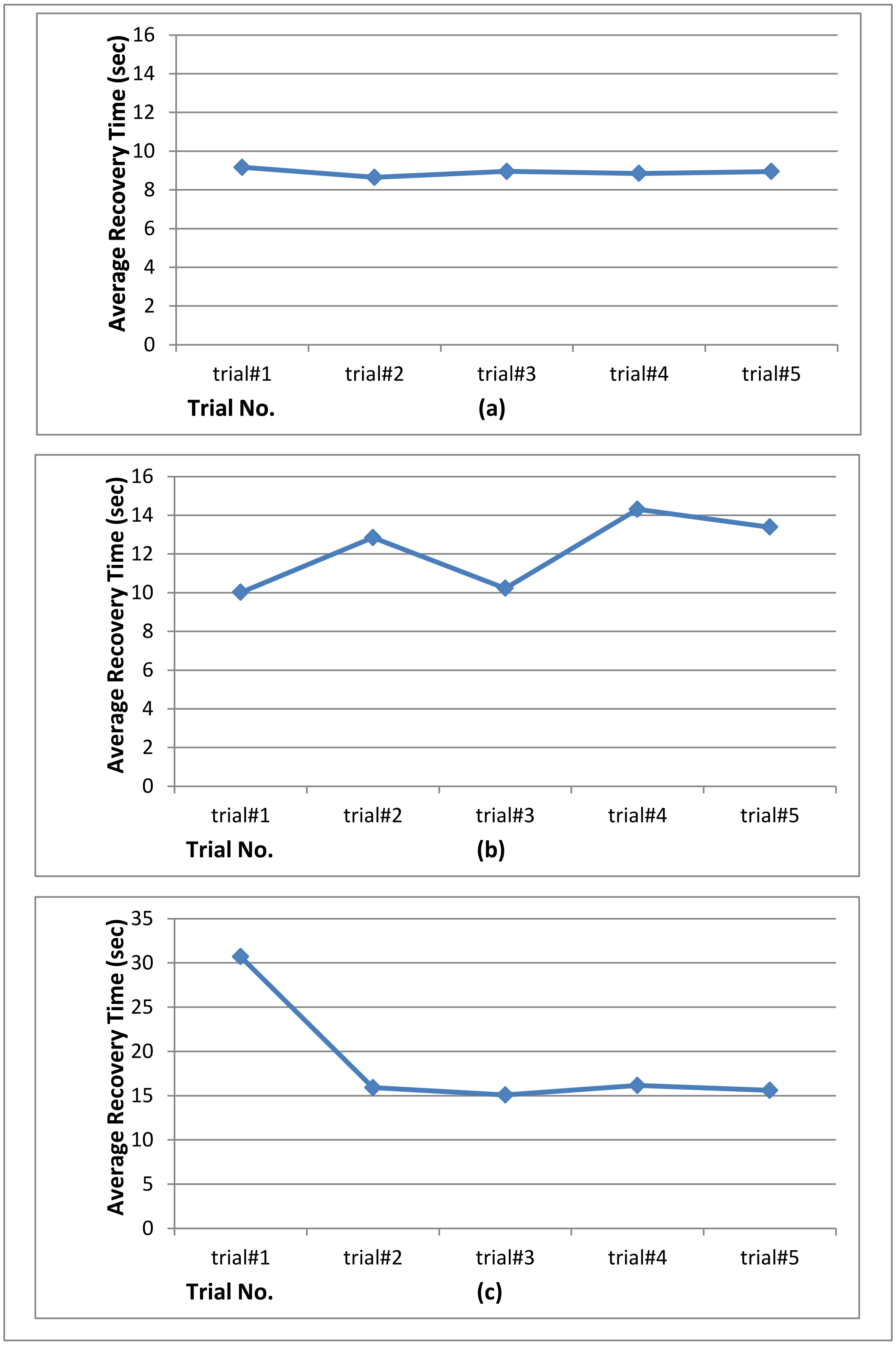

Figure 4.12 : Average recovery time (sec) in each trial for different distance scenarios (a) 0-hops (b) 2-hops (c) 5-hops (applying ownership QoS).

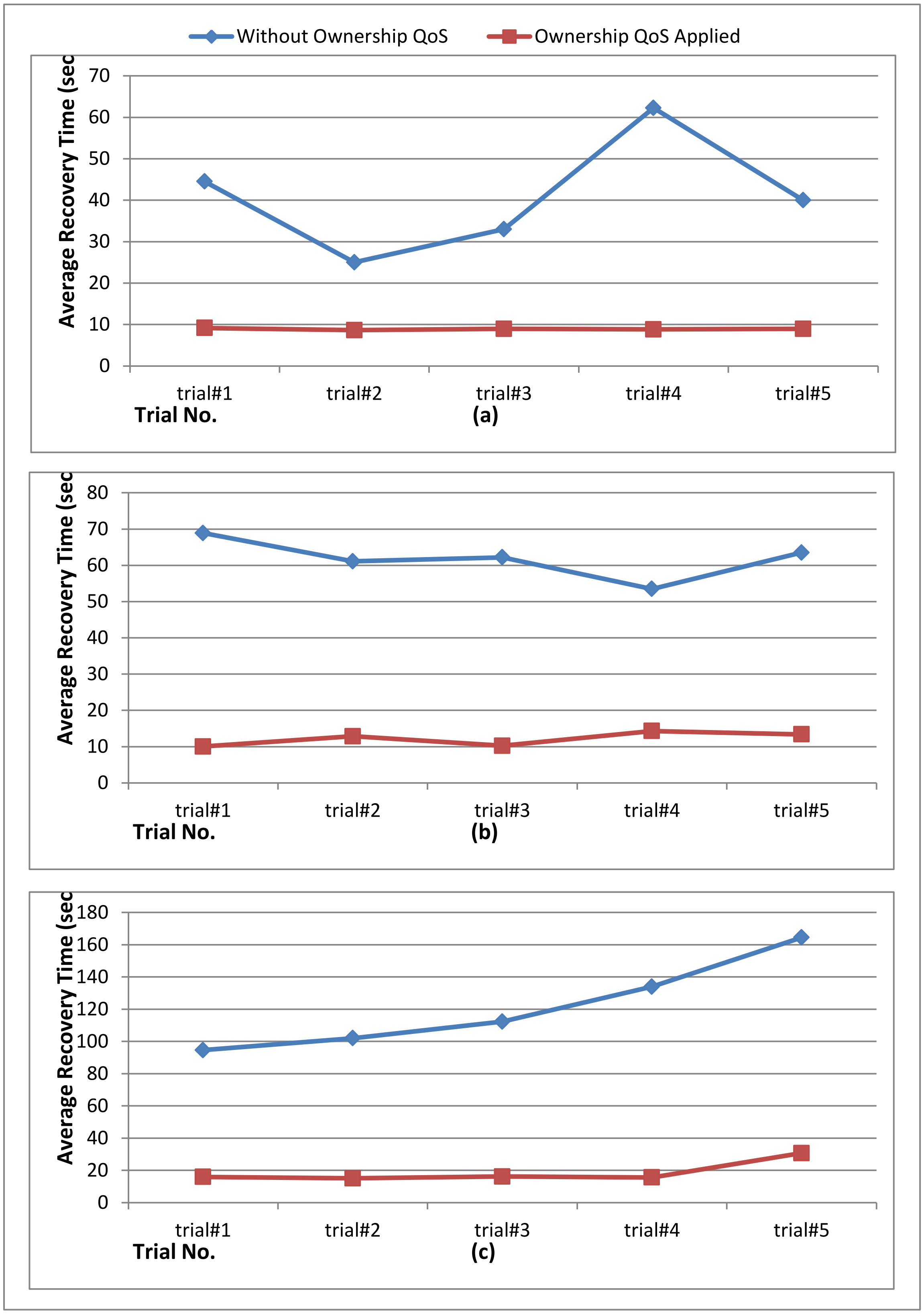

Figure 4.13: Average recovery time (sec) in each trial for different distance scenarios (a) 0-hops (b) 2-hops (c) 5-hops (A comparative study).

4.6 Summary

In this chapter, we presented Dynamic Pointers, a novel tracker-less discovery protocol for BitTorrent, we have introduced an evaluation for our proposed architecture. Dynamic-pointers framework is fully decentralized such that the tracker is completely eliminated and its role shared among some peers. Results show that our framework can achieve high scalability and reliability under churn. Also, it can achieve high availability and auto recovery under crashing or leaving of nodes. Table 4-6 holds a comparison among the three methods, centralized tracker, DHT, and Dynamic pointers, in terms of dependency, scalability, fault tolerance, security, and complexity.

Table 4-6: CHARACTERISTICS COMPARISON BETWEEN CENTRALIZED-TRACKER, DHT, AND DYNAMIC-POINTERS.

characteristic	**Approach**		
	Centralized Tracker	***Decentralized DHT***	***Dynamic-Pointers***
Dependency	Highly dependent on the tracker	Initially, depends on a bootstrap node.	Partial dependent on self-tracker which dynamic and recoverable.
Scalability	*A tracker is bottleneck*	*High scalablity*	*High scalability in LAN (need to be tested in WAN).*
Fault Tolerance	*Single Point Of Failure (SPOF)*	*High Reliability (No SPOF)*	*High Reliability (No SPOF*
Security	*Tracker could have a built-in security measures, but, it vulnerable to Denial of Service (DoS) attack.*	*No built-in security measures*	*Benefits from DDS security (there is a basic security).*
Complexity	*Network Overhead: O(1)* *Memory Consumption: O(N)*	*Network Overhead: O(logN)* *Memory Consumption: O(logN)*	*Network Overhead: O(1)* *Memory Consumption: O(1)*

CHAPTER 5

DDS BitTorrent: Implementation of BitTorrent Dissemination Protocol using RTPS middleware

In this chapter, we implement a BitTorrent dissemination protocol using RTPS middleware (DDS). To the best of our knowledge this is the first implementation for BitTorrent over DDS. We aim to enhance BitTorrent performance by benefiting from the publish/subscribe paradigm and by applying DDS QoS policies. The motivation behind this work is that DDS QoS are designed originally to improve P2P applications performance.

The following sections present DDS-BitTorrent QoS, theoretical analysis, design and implementation issues, experimental works, and evaluation and results.

5.1 DDS-BitTorrent Quality of Service:

5.1.1 DDS Reliable Delivery Model and Reliability QoS

The DDS-middleware uses the UDP transport to make communications between the peers. As a result, it uses by default, the best effort delivery model which means, no guarantee that all the instances published are received.

An important feature of DDS-middleware is that it can offer the reliability on top of a very wide diversity of transports like the unreliable UDP transport layer, packet based transports, multicast capable transports, or high latency transports. The middleware achieves this by implementing an application layer reliable protocol that sequences and acknowledges messages and observes the liveliness of the link [57]. The publisher maintains a send queue with space to hold the last X number of samples sent. Also, a subscriber maintains a receive queue with space for consecutive X expected samples. The send and receive queues are used to temporarily cache samples until the middleware is sure the samples have been delivered and are not needed anymore. There are three types of messages for the DDS reliable protocol:

- DATA Message: contains the value of data-objects and associated with a sequence number that middleware uses to identify them within the publisher history.
- Heart Beats Message (HB): announces to the subscriber that it should have got all data instances up to the one tagged with a range of sequence numbers. Also, it required by the subscriber to send acknowledgement back. For example, HB (0-2) informs the subscriber that it should have received messages tagged with sequence numbers 0,1, and 2 and asks the subscriber to confirm this.
- ACK/NACK Message: communicates to the publisher that particular instances have been successfully received and stored in the subscriber history. ACK/NACK also tells the publisher which instances are missing on the subscriber side. The sequence number of ACK/NACK message indicates which one the subscriber is missing. For example, ACKNACK(3) indicates that instances with sequence numbers 0,1, and 2 have been successfully received and stored in the subscriber history, and that 3 has not been received. The ACK/NACK messages are only sent as a direct response to HB messages

An important note to be mentioned is that the middleware can bundle multiple of the above messages in a single packet. This provides a higher performance communications. more details about this protocol can be found in [57].

For BitTorrent, which is a file sharing protocol, we need to use reliable protocol to transfer the designated file from publishers to a subscriber. For this purpose the Reliability QoS must be set to REALIABLE.

5.1.2 History QoS

Controls how the middleware manages the samples payload sent by the publisher's DataWriter or received by the subscriber's DataReader. It helps tune the reliability between publishers and subscribers.

For BitTorrent, we need restrict reliability which can be achieved by setting the History QoS to KEEP_ALL value. This means that the samples sent must be kept in the memory until they are acknowledged.

5.1.3 Partition

This QoS can be used to add additional conditions for the publish-subscribe matching. In a normal manner, DataWriters are matched to DataReaders of the same topic. By using the Partition QoS, additional condition is used to decide whether a DataWriter samples are allowed to be sent to a DataReader. This QoS helps in making the communication model one-to-one per a thread. More details about how to use this QoS within BitTorrent will be discussed in section5.2.2.

5.1.4 Durability

Specifies whether the middleware should store and deliver previously published samples to late subscribers.

Since, the file sharing process usually works in one-to-one communication model, and to prevent publisher memory from overflowing, this QoS should be set to VOLATILE value. That means, the samples will be removed from the memory as soon as they are acknowledged.

5.1.5 DDS-BitTorrent: Traffic Measurement Theoretical Analysis

The main principle that DDS-BitTorrent follows is that "just join us, or just subscribe to us, and everything will come to you", you do not need to create connections with the other peers, or to request each piece of the whole file to complete downloading. Simply, as just subscribe to a topic (file name) the pieces of the file will come to you from all a topic publishers without the needing to establish a unique connection to each publisher and send extra overhead messages (e.g. request, interested, bitfield, …). Figure 5.1 shows the difference behavior between the DDS-BitTorrent and the standard BitTorrent, assuming that the computer in the middle is a new joiner peer.

Using the publish/subscribe approach Within BitTorrent will help in eliminating the application layer overhead (BitTorrent messages). Also, since the DDS middleware is implemented by standard over UDP, this will minimize the transport layer overhead [15]. The overhead caused by the DDS reliable application protocol is very small, and is limited in two messages; ACK/NACK, and HB which can be piggybacked within the data message.

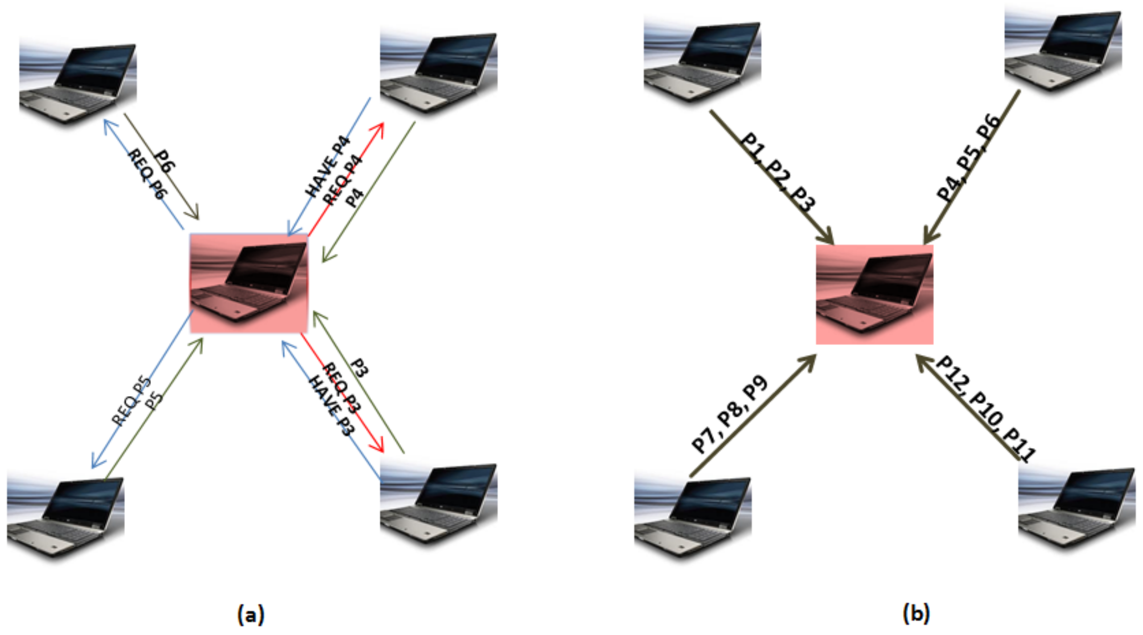

Figure 5.1 (a) standard BitTorrent behavior (b) DDS-BitTorrent behavior

The theoretical analysis for the total traffic transferred in the network for both standard BitTorrent and DDS-BitTorrent can be demonstrated after presenting the following characteristics:

- BitTorrent messages are twelve messages, excluding the Piece message which is the actual data; the remaining eleven messages are overhead. While all the overhead messages needed in DDS-BitTorrent are two messages.

- The number of the ‘Have’ messages that should be sent is equal to the number of the file pieces times the number of the swarm peers.
- The number of the ‘Request’ messages that should be sent is equal to the number of the file pieces.
- The number of the ‘Handshake’ messages that should be sent is equal to the number of the swarm peers.
- In DDS-BitTorrent, the number of the HB messages that should be sent is usually equal to TCP sequence messages. And, the number of the ACK messages sent is usually equal to the number of TCP ACK messages.

Having a file with size M bytes and this file is chunked into K pieces. Also, assuming a swarm with R peers, S is the number of the other BitTorrent messages are sent, such that S<K. Assuming that all these messages have the same size (usually 60 bytes), the formulas for the total traffic transferred through the network for both standard BitTorrent and DDS-BitTorrent are depicted in Equation 5.1 and Equation 5.2, respectively.

$$TotalTraffic_{st.BitTorrent} = M + (K * R)\ Have\ message + K * Request\ message + R * Handshake\ message + (S * 8)\ anonymous\ message + TCP_{seq}\ messages + TCP_{ack} messages \qquad (5.1)$$

$$TotalTraffic_{DDS.BitTorrent} = M + HB\ messages + ACK\ messages \qquad (5.2)$$

Remove TCP_{seq} messages and TCP_{ack} messages from Equation 5.1, and remove HB messages and ACK messages from Equation 5.2 we can conclude that $totalTraffic_{DDS.BitTorrent} < totalTraffic_{st.BitTorrent.}$

It’s clear that the total traffic transferred by DDS-BitTorrent is much less than the total traffic transferred by standard BitTorrent.

From the transport layer respective, each BitTorrent block is encapsulated in a TCP segement must accompany 20-bytes of overhead, while a UDP packet uses only 8-bytes overhead [16].

5.2 DDS-BitTorrent: Design and Implementation Issues

This section covers the design and implementation of the proposed DDS-BitTorrent. To propose a good architecture for implementing BitTorrent to work over DDS, we are trying to make a simple mapping between BitTorrent's main components and RTPS-DDS's main components as in the following table:

Table 5-1: Mapping BitTorrent main components to DDS components.

BitTorrent	**DDS**
Tracker Server	DDS-BitTorrent is tracker-less, so, either using dynamic pointers discovery protocol, or using default DDS simple discovery protocol (SDP). depends on the system size
Seeder	DDS publisher
Leecher	DDS subscriber
Torrent file	Topic name
Piece	Instance (Each piece represents an instance of the topic).
Block	Sample (each block is a sample of the instance)

Since, the communication paradigm in DDS is a data oriented and not a node oriented, two issues during the implementation are appeared; these issues and their solutions are illustrated in the following subsections

5.2.1 Issue 1: Redundant Pieces Delivery

As a new subscriber subscribes to a particular file, all the publishers start publishing pieces of that file. Multiple publishers may have the same pieces. Figure 5.2 shows the process of publishing the same pieces from multiple publishers, this will cause in wasting network bandwidth and result in redundancy delivery at the subscribing side.

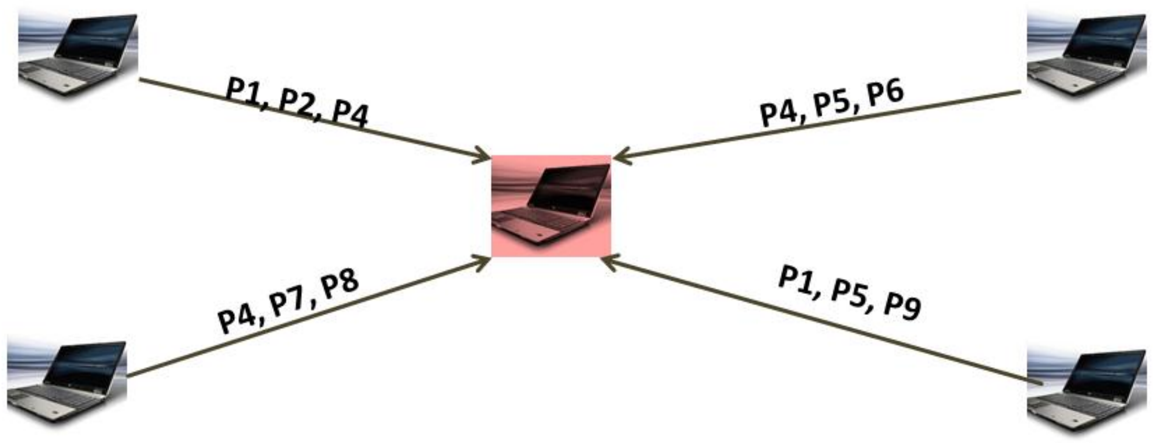

Figure 5.2: multi publisher send the same pieces to one subscriber

Using the Ownership QoS does not solve the problem since the Ownership is network overhead, it just solve the delivery redundancy problem. By using the Ownership QoS the overhead ratio may reach 100% or even x*100 % of the file size, where x is the number of the seeders.

A good solution to this issue should achieve load balancing, and fairness among the publishers. Collaborative piece sharing algorithm is our proposed solution for this problem. In this algorithm, the publisher only shares the pieces that are in a particular list, called “Owning Piece List”. Two terms should be distinguished between them; having a piece and owning a piece. Having a piece means that the peer has downloaded the piece successfully, owning the piece means that this peer is the responsible for sharing this piece to new joiners peers (subscribers). The procedure of how this algorithm works and how the Owning Piece List is managed are as follows:

- Initially: the initial seeder owned all the file pieces (all the file pieces are in the “Owning Piece List”), so as new subscriber joins, it will receive all the pieces of the file from the initial seeder.
- The initial seeder gives to the first subscriber 50% of its owning pieces, which are in this case 50% of the file pieces.
- As the second subscriber joins, it will get its complete file pieces as follows; 50% from the initial seeder, and 50% from the first subscriber. The second subscriber

owning piece list will be filled by 25% of the file pieces from the initial seeder, and 25% form the first subscriber.

- After the second subscriber having all the pieces, the scene can be shown as follows:
    - The initial seeder owns 25% of the file pieces.
    - The first subscriber owns 25% of the file pieces.
    - The second subscriber owns 50% of the file pieces.
- The process continues as the third, fourth, ..., etc. peers join the swarm
- When the peer's owning pieces becomes 5% of the file pieces, it publishes pieces without transferring the ownership of these pieces to others three times, then the peer can owns the 5% pieces to another peer and becomes free.
- Finally, if a subscriber does not get all the file pieces for any reason, the subscriber sends a message to initial seeder requesting the remaining pieces.

To conclude, each time a new peer joins the swarm, each previous peer available in the swarm gives to the new peer 50% of its owning pieces. The ratio of total owning pieces from all peers must be 100%. Also, the last joiner peer always shares the biggest ratio of the file pieces; the share process starts as soon as a new peer joins and the owning piece list is not empty. This achieves load balancing, fairness, and, mitigation of free riding. The pseudo code of this algorithm from the publisher perspective is depicted in Figure 5.3, and from the subscriber perspective is depicted in Figure 5.4.

```
Algorithm: Collaborative Pieces Sharing (share pieces)
Input:  list ownPiecelist (OPL),
Output: new ownPiecelist
Steps:
    1.  BEGIN
    2.  SET int lastPiecesCounter ← 0
    3.  IF OPL is empty THEN     // make sure that my ownPieceList is not empty
        3.1  exit                    // you havn't pieces to send so exit
  // compare the elements in my ownPieceList with the number of the whole file pieces
    4.  int fifthPercent ← 0.05 *filePieces.size
    5.  IF OPL.size > fifthPercent THEN
        5.1  int OwntoAnother ← OPL.size()/2   // own 50% of my own pieces to another peer
    6.  ELSE
        6.1  Increment lastPiecesCounter  //share the pieces three time
    7.  ENDIF

//loop through the OwnPieceList and pusblish the pieces
    8.  FOR i ← 0 to i ← OPL.size()
        8.1     IF i < OwntoAnother  OR lastPiecesCounter > 3 THEN
                8.1.1     SET Piece_i.ownedFlag to true          //give this piece to another peer
                8.1.2     PUBLISH Piece_i
                8.1.3     REMOVE Piece_i from  OPL
        8.2     ELSE   //share but keep this piece owned
                8.2.1     SET Piece_i.OwnedFlag to false
                8.2.2     SHARE Piece_i
    9.  ENDFOR
    10. END
```

Figure 5.3: Collaborative piece sharing algorithm pseudocode (share pieces).

```
Algorithm: Collaborative Pieces Sharing (receive pieces)
Input:  list ownPiecelist (OPL), piece i
Output: new ownPiecelist
Steps:
    1.  BEGIN
    2.  VERIFY $piece_i$
    3.  IF $piece_i$ is verified
        3.1  Save $piece_i$
        3.2  IF $piece_i$.ownflag is equal true
                3.2.1    ADD piecei to OPL
        3.3  ENDIF
    4.  ENDIF
    5.  FINALLY
        5.1  IF Received_Pieces are equal to File_Pieces    // test if receive all the pieces
                5.1.1    exit
        5.2  ELSE
                5.2.1    Contact the initial_seeder
                5.2.2    Request  Remaining_Pieces
    6.  ENDIF
    7.  END
```

Figure 5.4: Collaborative piece sharing algorithm pseudocode (receive pieces)

5.2.2 Issue 2: Subscriber buffer Overflowing

Since there are many publishers that publish pieces to a single subscriber at the same time, it’s very likely that the subscriber buffer overflows as shown in Figure 5.5. As a result, many pieces will be discarded and needs republishing, which will result in a huge network overhead.

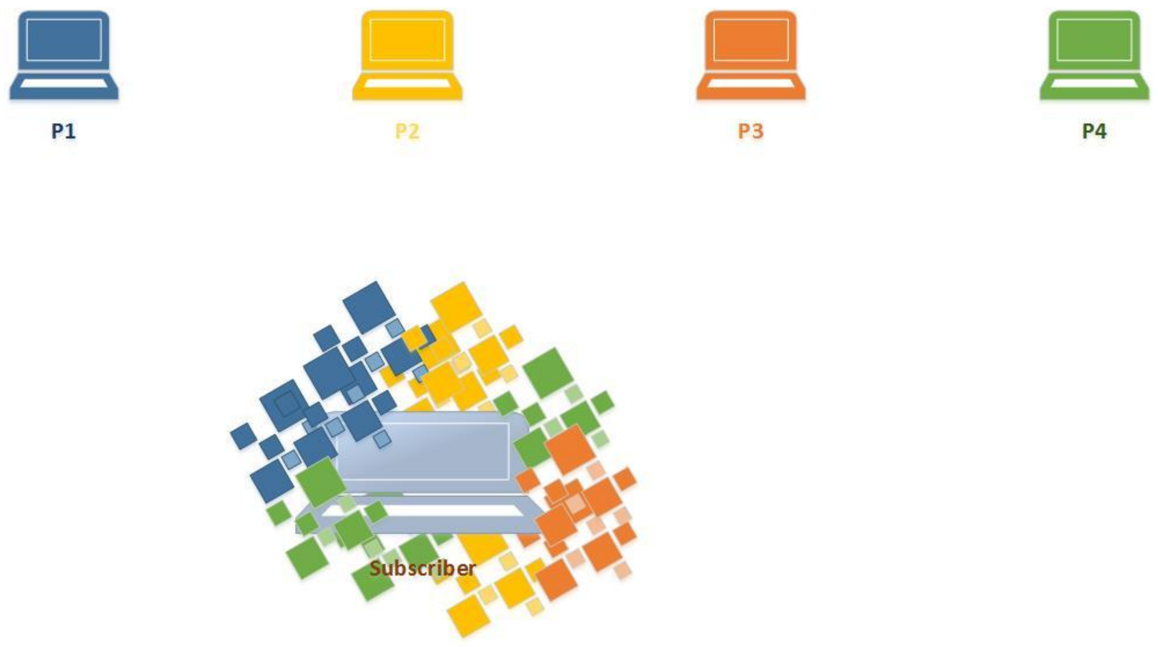

Figure 5.5 multi-publishers overwhelm single subscriber with the pieces

The trivial solution for this problem is to use multithreading. However, using this solution standalone does not solve the problem, and results in that each thread is overwhelmed with pieces from all the publishers as shown in Figure 5.6. The reason behind this situation is that DDS is data oriented and not node oriented; all the children threads and the parent thread have the same IP and port.

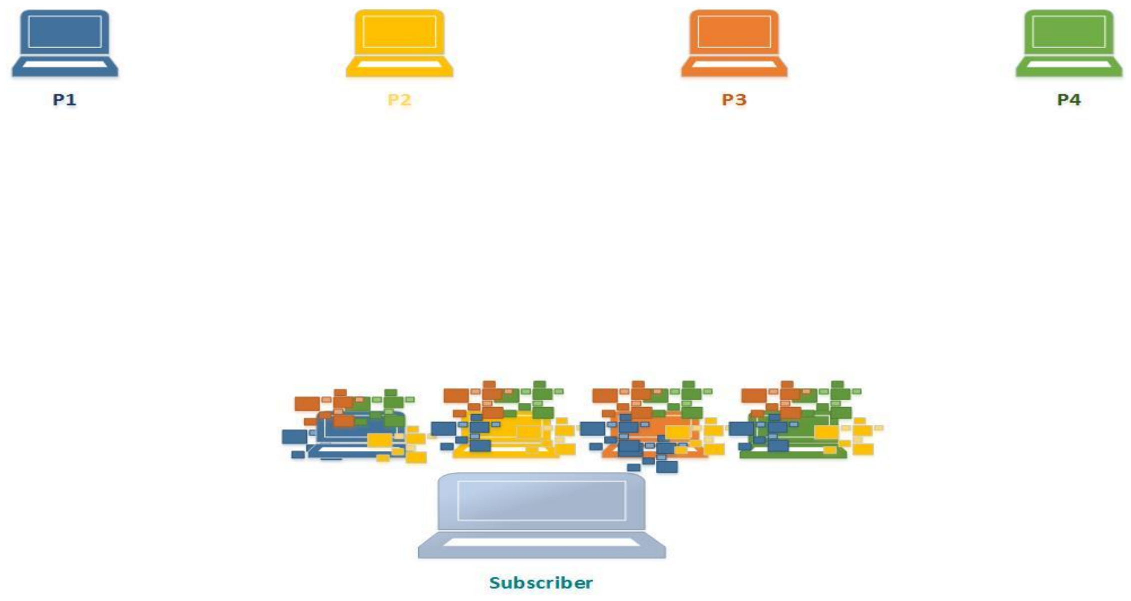

Figure 5.6: Multithreading (multi-subscribers).

Using Partition QoS is a good solution for this issue. By using partition QoS, we can make each subscriber thread match only one publisher. The parent subscriber creates children subscribers equal to the number of publishers.

Initially, on the publishing side, the parent publisher will create new publisher as soon as the parent publisher notified that new subscriber has joined. The child publisher sets the Partition string to unique string (e.g. Publisher IP+ a unique number) and then sends this string to the subscriber; the parent subscriber creates a child subscriber with Partition QoS value equal to the unique string that has been sent from the publisher. This subscriber only matches with one publisher as shown in Figure 5.7.

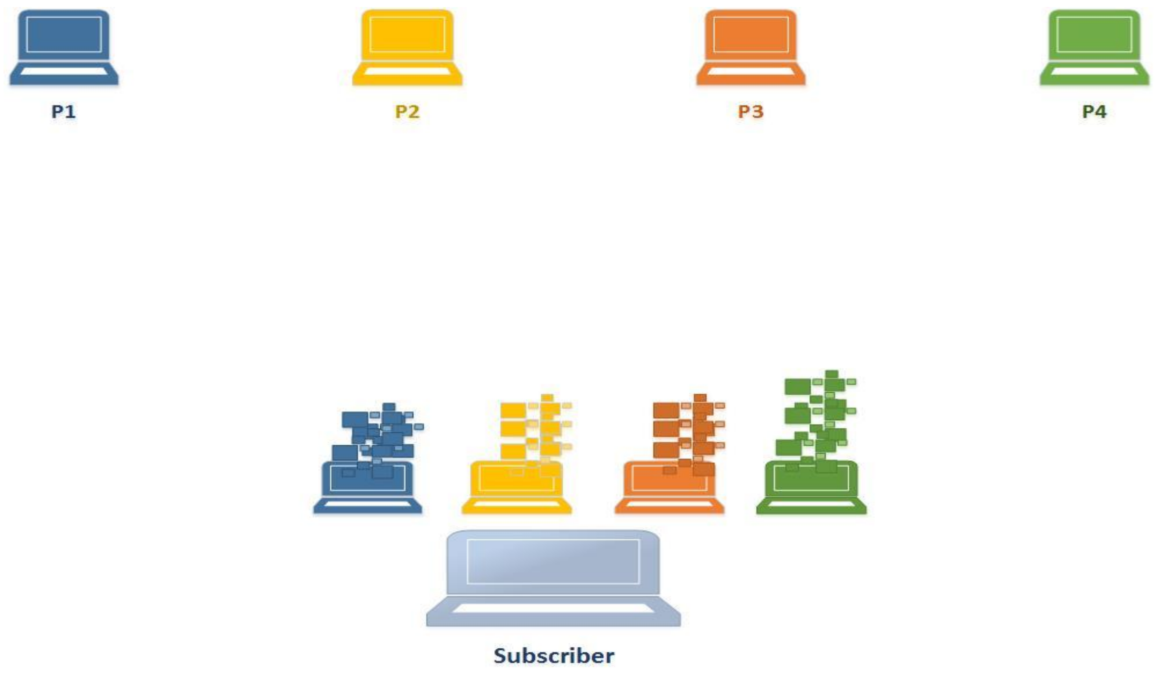

Figure 5.7: Multithreading (multi-subscribers) using Partition QoS.

5.3 Experimental Work

First of all, we construct our network for both standard BitTorrent and DDS-BitTorrent by using GNS3 [36] emulator. The network contains 5 virtual PCs created using Oracle VM VirtualBox [58], and connected to each other via Ethernet switch as shown in Figure 5.8 ; additional PC is needed to work as tracker for standard BitTorrent as shown in Figure 5.9 . Each PC has Intel Xeon 3.47 GHz CPU, 2 GB RAM, and windows XP 32-bit. Four of these PCs run as seeders or publishers, and one runs as a subscriber or a leecher.

Java JDK version 7 [33] as a programming language, NetBeans version 7.4 [34]as an environment, and RTI-DDS API [35] as a middleware are used to complete the implementation of DDS-BitTorrent. For the purpose of Comparison, we download JBitTorrent library [17], which is an open source implementation of the BitTorrent protocol in Java under General Public License (GPL) 2.

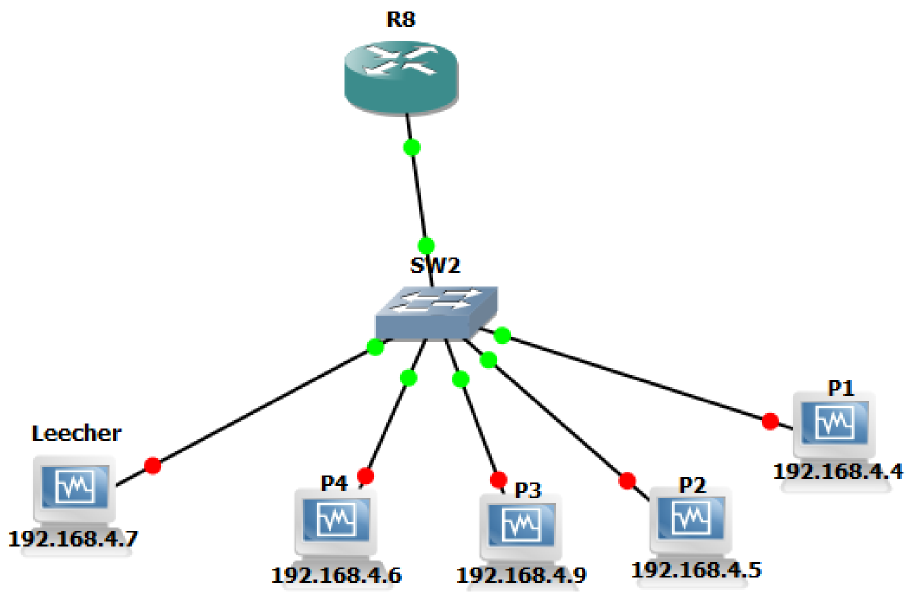

Figure 5.8: DDS-BitTorrent network topology

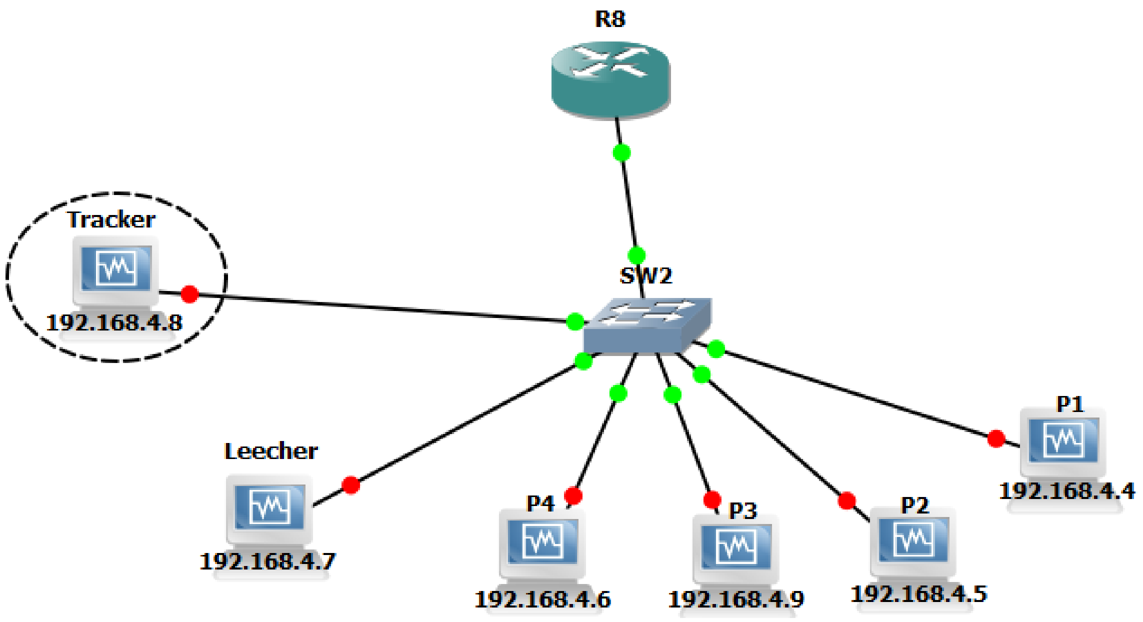

Figure 5.9: Standard BitTorrent network topology

File downloading time, and goodput are the two metrics used to complete the comparison test. Goodput which is the application layer throughput is defined as the number of useful data bits delivered by the network to a particular destination per a time unit. Equation 5.3 defines the goodput.

$$goodput = \frac{actaul\ file\ size\ (Mb)}{total\ link\ traffic\ (Mb)} x\ R\ Mb/s \qquad (5.3)$$

Where, R Mb/s is the Ethernet link data rate. Wireshark [59] which is a network analysis tool that is used for packets capturing and calculating the total traffic (include incoming and outgoing) of the subscriber link.

For both file downloading time and goodput tests, four files with different sizes (54Mb, 129Mb, 494Mb, 2 GB) are used. For each test, three important scenarios per each file are done which are based on the variance of piece and block sizes, as the following:

- Piece size:256 KB and block size:16 KB
- Piece size: 512 KB and block size: 32 KB
- Piece size: 1024 KB and block size: 64 KB

For each scenario, the four files are published from the publishers to the subscriber five times for each, and the average file downloading time and the average goodput are calculated. This is done for both standard BitTorrent and DDS-BitTorrent.

5.4 Evaluation and Results

To evaluate our proposed DDS-BittTorrent, we compare it with the standard BitTorrent.

When piece size is 256 KB and block size is 16 KB as shown in Table 5-2and Figure 5.10, DDS-BitTorrent outperforms standard BitTorrent. The results show that the ratio of the downloading time of DDS-BitTorrent is less than standard BitTorrent in about 50%. For goodput as shown in Table 5-3 and Figure 5.11, the results show that DDS-BitTorrent goodput is always between 90 and 95 Mb/s, this is because of its very small overhead. On the other hand, standard BitTorrent goodput starts at 87 Mb/s and decreases as the file size increases; this is due to the fact that the bitTorrent overhead is directly proportional to the file size

When piece size is 512 KB and block size is 32 KB as shown in Table 5-4 Figure 5.12, DDS-BitTorrent also can download the file faster and its goodput (see Table 5-5 and Figure 5.13) still between about 90 and 95Mb/s. the standard BitTorrent performance is get better. In this scenario, pieces number is half pieces number of the first scenario; as a result, the BitTorrent overhead messages and TCP overhead are decreased to about 50%.

Table 5-2: Avg.File downoalding time (sec) for Both standard and DDS BitTorrent (piece:256KB,block: 16KB).

File size (MB)	Standard BitTorrent	DDS-BitTorrent
54	16.2	7.8
129	40.11	15.2
494	111	78.47
1998	544.9	283.9

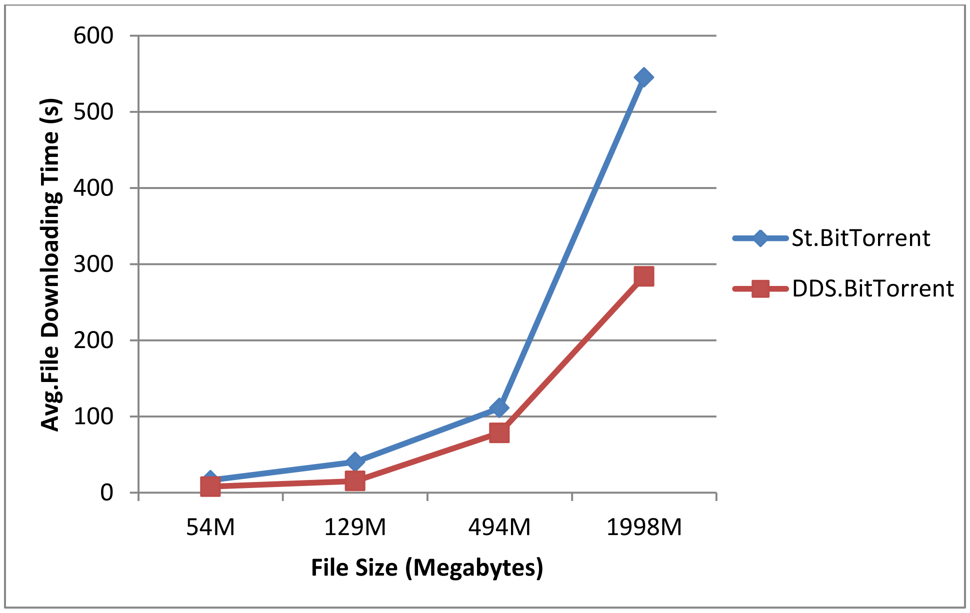

Figure 5.10: Average file downloading time (sec) for different file sizes (piece: 256KB, block: 16KB)

Table 5-3: Standard vs. DDS BitTorrent goodput (Mb/s) for different files (Piece size: 256 KB, block size: 16 KB).

File size (MB)	Standard BitTorrent		DDS-BitTorrent	
	Link traffic (MB)	Goodput	Link traffic (MB)	Goodput
54	61.6	87.6	59	91.5
129	152	84.8	139	92.8
494	609	81.1	521	94.8
1998	3092.4	64.5	2109	94.6

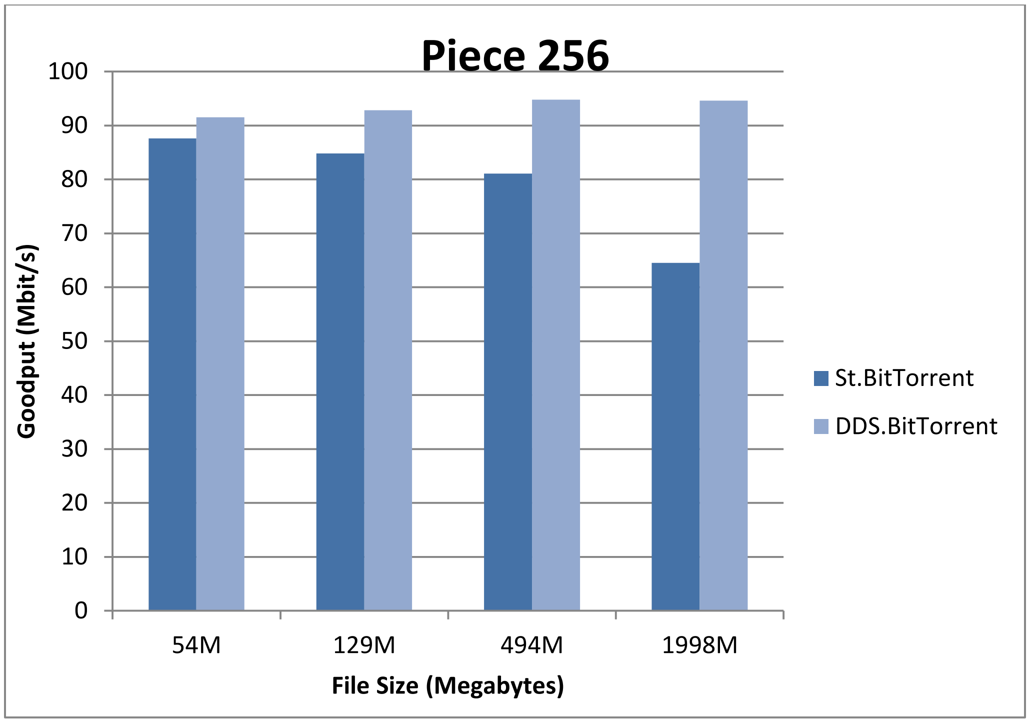

Figure 5.11: Standard vs. DDS BitTorrent goodput (Mb/s) for different files (Piece size: 256 KB, block size: 16 KB).

Table 5-4: Avg.File downoalding time (sec) for Both standard and DDS BitTorrent (piece:512KB,block: 32KB).

File size (MB)	Standard BitTorrent	DDS-BitTorrent
54	13.7	10.3
129	28.9	24.2
494	107.4	96.6
1998	486	296.4

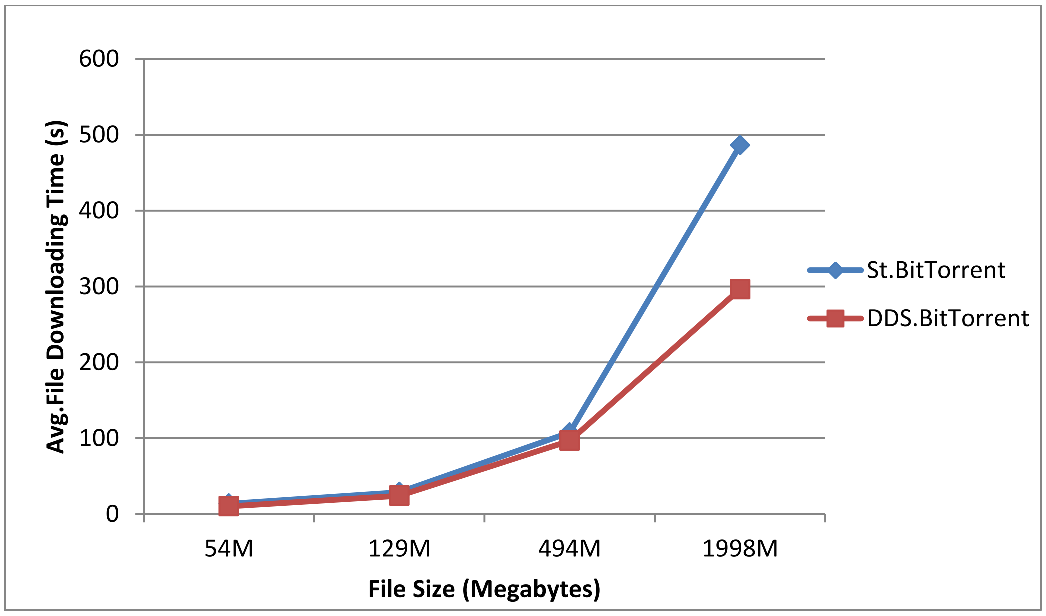

Figure 5.12: Average file downloading time (sec) for different file sizes (piece: 512KB, block: 32KB)

Table 5-5: Standard vs. DDS BitTorrent goodput (Mb/s) for different files (Piece size: 512 KB, block size: 32 KB)

File size (MB)	Standard BitTorrent		DDS-BitTorrent	
	Link traffic (MB)	Goodput	Link traffic (MB)	Goodput
54	60.6	89.1	60	90
129	149	86.5	140	92.1
494	585	84.4	521	94.8
1998	2621	76.1	2132	93.7

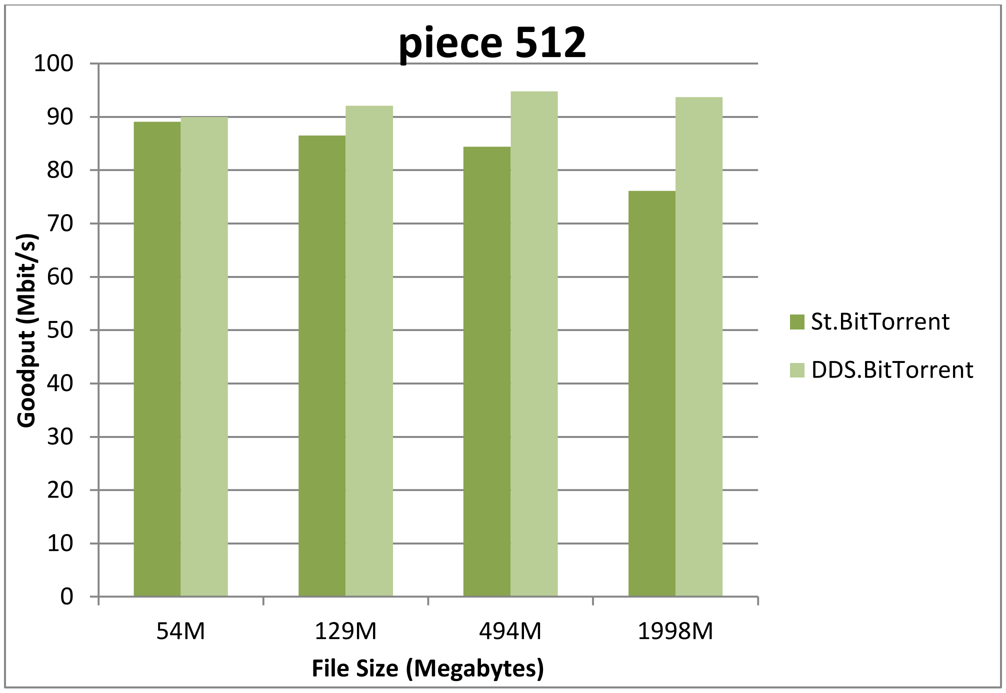

Figure 5.13: Standard vs. DDS BitTorrent goodput (Mb/s) for different files (Piece size: 512 KB, block size: 32 KB)

When piece size is 1024 KB and block size is 64 KB as shown in Table 5-6 and Figure 5.14, the standard BitTorrent works better than the DDS-BitTorrent. Actually, In DDS which is implemented over UDP, a 64 KB block will be segmented into 44 (64 KB/1500 bytes) Ethernet frames and then reassembled on the subscribing side; the loss of anyone of these frames will make reassembly impossible leading to an effective loss. This will result in a huge overhead and degrading the goodput as depicted in Table 5-7 Figure 5.15.

Table 5-6: Avg.File downoalding time (sec) for Both standard and DDS BitTorrent (piece:1024KB,block: 64KB).

File size (Mbytes)	Standard BitTorrent	DDS-BitTorrent
54	10.5	15.9
129	24.7	32.5
494	85.14	119.1
1998	444.4	328.1

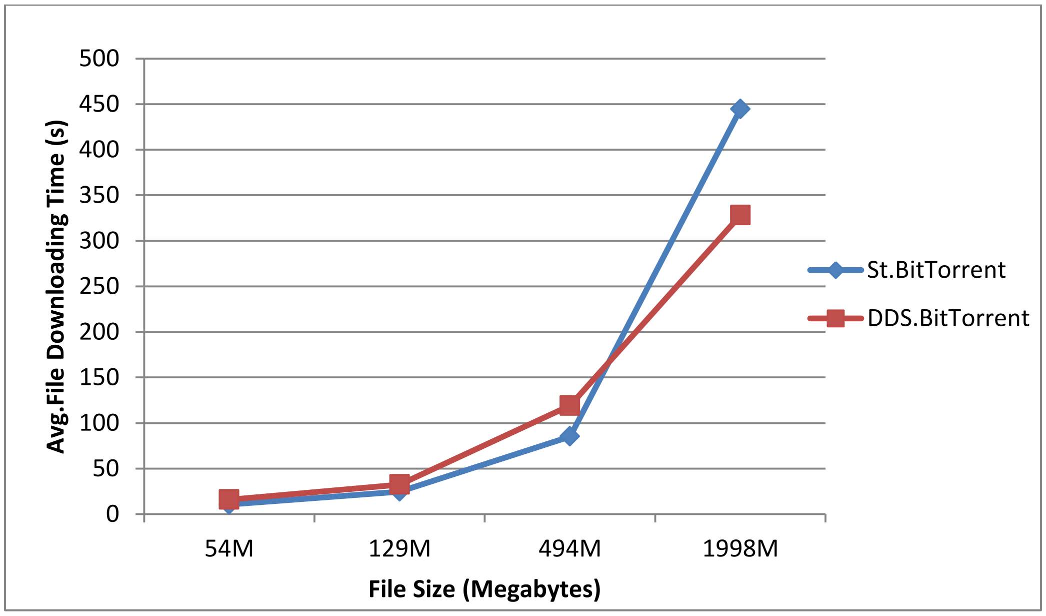

Figure 5.14: Average file downloading time (sec) for different file sizes (piece: 1024KB, block: 64KB)

Table 5-7: Standard vs. DDS BitTorrent goodput (Mb/s) for different files (Piece size: 1024 KB, block size: 64 KB).

File size (Mbytes)	Standard BitTorrent		DDS-BitTorrent	
	Link traffic (Mbytes)	Goodput	Link traffic (Mbytes)	Goodput
54	59.7	90.4	72	72.7
129	143	90.2	170	75.8
494	563	87.7	585	84.4
1998	2549	78.3	2481	80.5

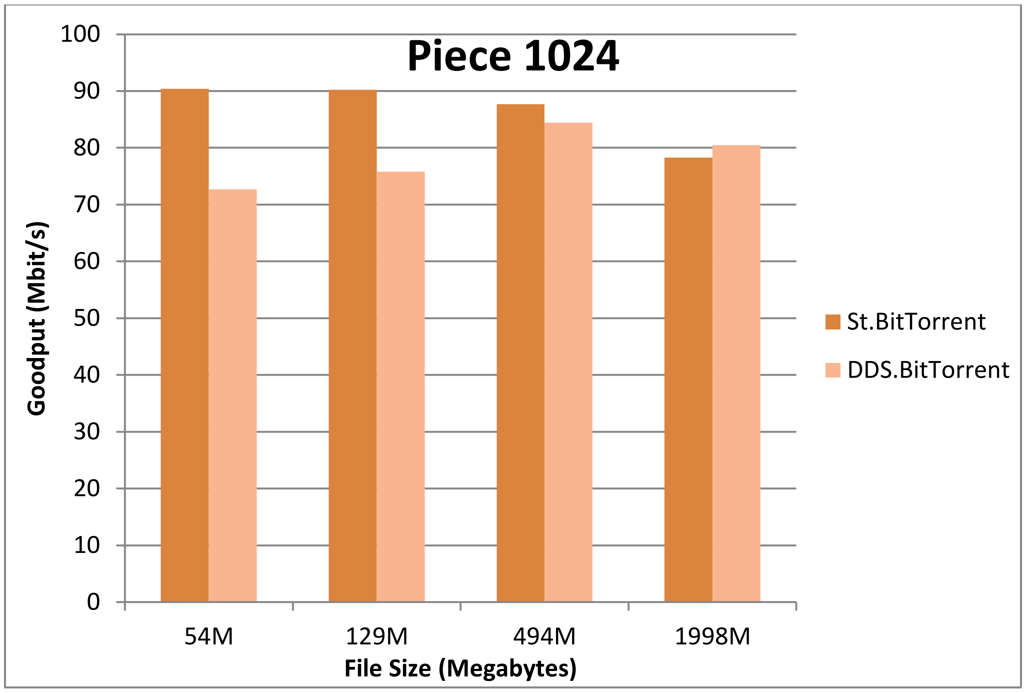

Figure 5.15: Standard vs. DDS BitTorrent goodput (Mb/s) for different files (Piece size: 1024 KB, block size: 64 KB).

To conclude, DDS goodput is not related to the file size, it use the same small overhead for all the file sizes (Figure 5.16); degradation of the DDS-BitTorrent happened because of the loss of packets which requires packets retransmission. The block size recommended to use with DDS is between 16k and 32k.

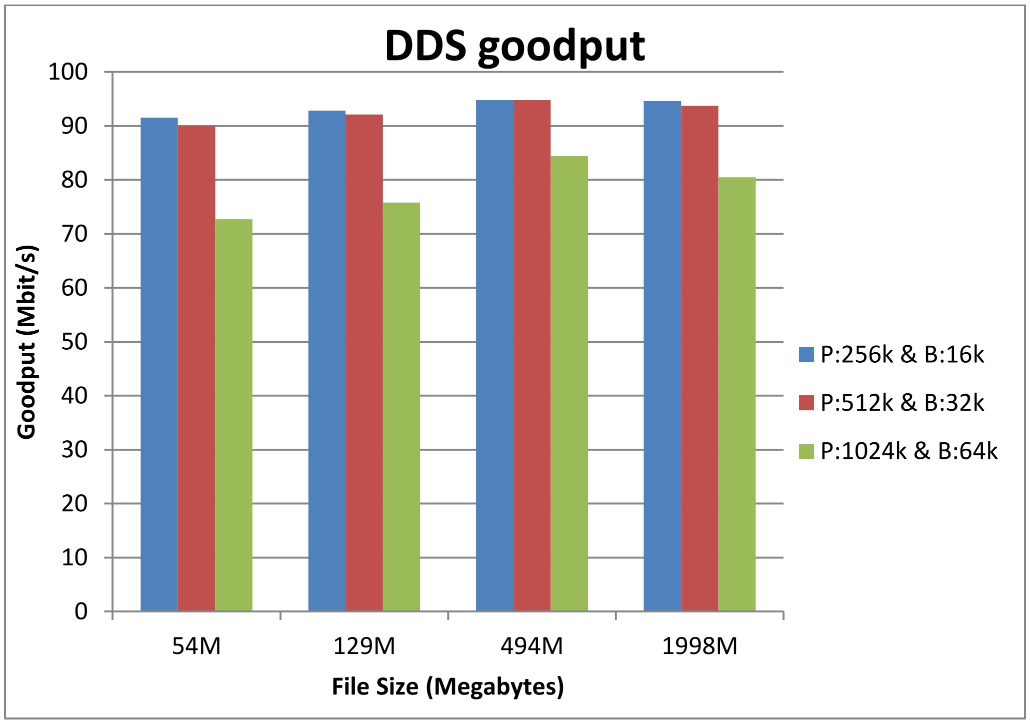

Figure 5.16: DDS-BitTorrent goodput (Mbit/s) for different file sizes and pieces.

5.5 Summary

In this chapter, we introduced DDS-BitTorrent, a BitTorrent dissemination protocol that is based on DDS middleware. We aim to enhance BitTorrent performance by benefiting from the publish/subscribe paradigm and by applying DDS QoS policies. Theoretical analysis, design, and implementation are presented. DDS-BitTorrent is fast, has low network overhead, and achieves load balancing, fairness, and mitigation freeriding among peers. A comparative study between the proposed DDS-BitTorrent and the standard BitTorrent was conducted. Results show that our solution outperforms the original BitTorrent when the block size is less than or equal 32 KB; this block size is suitable for BitTorrent [9] [41][60]. On the other hand, when the block size was set to 64 KB, DDS-BitTorrent resulted in huge overhead because the blocks must be fragmented, and as a result, they will be subjected to being lost.

Chapter 6

Conclusion and Future Work

In the recent years, BitTorrent protocol got a lot of adoption on peer to peer file sharing systems, and still being the dominant traffic on the Internet. However, this protocol suffers from the dependency on a single server called a tracker for the coordination and the content routing between its peers; this is a single point of failure (SPOF) problem. The other problem with BitTorrent is that during the dissemination of the files between the peers, the BitTorrent uses many overhead messages.

In this research, we studied both the discovery and dissemination protocols of BitTorrent. For discovery, we presented Dynamic Pointers, a novel tracker-less discovery protocol for BitTorrent, we have provided an evaluation for our proposed architecture. Dynamic-pointers framework is fully decentralized such that the tracker is completely eliminated and its role is shared among some peers. Results show that our framework can achieve high scalability and reliability under churn. Also, it can achieve high availability and auto recovery under crashing or leaving of nodes. Moreover, dynamic pointers minimized both the network overhead and memory consumption complexities to *O(1).*

For dissemination, we analyzed BitTorrent dissemination protocol theoretically then we proposed DDS-BitTorrent by re-implementing the standard BitTorrent using DDS middleware benefiting from the publish/subscribe paradigm and DDS QoS policies in reducing overhead messages. To evaluate our proposed DDS-BittTorrent, we compared it with the standard BitTorrent; goodput and file downloading time are two metrics used to complete the comparative study. The results show that DDS-BitTorrent outperforms the standard BitTorrent when the block size is less than or equal 32 KB; this block size is suitable for BitTorrent. On the other hand, when the block size was set to 64 KB, DDS-BitTorrent results in huge overhead because the messages are subjected to being lost.

This work is a first and basic step. So for future work, this work can be extended and improved. For dynamic-pointers, it needs to be tested in a wider network (e.g. WAN). For DDS-BitTorrent, impact of the peers churn should be shown and tested. Also, the multi-hops should be considered. The algorithms used need more improvements.

References

[1] Papafili, Ioanna, Sergios Soursos, and George D. Stamoulis. "Improvement of bittorrent performance and inter-domain traffic by inserting isp-owned peers."Network Economics for Next Generation Networks. Springer Berlin Heidelberg, 2009. 97-108.

[2] Fry, Charles P., and Michael K. Reiter. "Really truly trackerless bittorrent."School of Computer Science, Carnegie Mellon University, Tech. Rep (2006): 06-148.

[3] S. Oh, J. hoon Kim, and G. Fox, \Real-Time Performance Analysis for Publish/Subscribe Systems," 2009.

[4] Lareida, Andri, et al. "RB-tracker: A fully distributed, replicating, network-, and topology-aware P2P CDN." Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on. IEEE, 2013.

[5] Cohen, Bram. "The BitTorrent protocol specification." (2008): 28.

[6] Hecht, Fabio Victora, Thomas Bocek, and Burkhard Stiller. "B-Tracker: improving load balancing and efficiency in distributed P2P trackers." Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on. IEEE, 2011.

[7] Bindal, Ruchir, et al. "Improving traffic locality in BitTorrent via biased neighbor selection." Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on. IEEE, 2006.

[8] Schlesselman, Joseph M., Gerardo Pardo-Castellote, and Bert Firebaugh. "OMG data distribution service (DDS): architectural update." Military Communications Conference, 2004. MILCOM 2004. 2004 IEEE. Vol. 2. IEEE, 2004.

[9] Sherman, A., Nieh, J., & Stein, C. (2012). FairTorrent: a deficit-based distributed algorithm to ensure fairness in peer-to-peer systems. IEEE/ACM Transactions on Networking (TON), 20(5), 1361-1374.

[10] OMG. “Data Distribution Service for Real-time systems”. Object Management Group, 1.2 formal/07-01-01 edition, January 2007.

[11] The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification, Version 2.2, January 2014, OMG:formal/2014-09-01 http://www.omg.org/spec/DDSI-RTPS/2.2/.

[12] An, K., Gokhale, A., Schmidt, D., Tambe, S., Pazandak, P., & Pardo-Castellote, G. (2014, May). Content-based filtering discovery protocol (CFDP): scalable and efficient OMG DDS

discovery protocol. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems* (pp. 130-141). ACM.

[13] Corsaro, A., Querzoni, L., Scipioni, S., Piergiovanni, S. T., & Virgillito, A. (2006). Quality of service in publish/subscribe middleware. *Global Data Management*, *19*, 20.

[14] Bellavista, P., Corradi, A., Foschini, L., & Pernafini, A. (2013, July). Data Distribution Service (DDS): A performance comparison of OpenSplice and RTI implementations. In *Computers and Communications (ISCC), 2013 IEEE Symposium on* (pp. 000377-000383). IEEE.

[15] Xiong, M., Parsons, J., Edmondson, J., Nguyen, H. and Schmidt, D. “Evaluating Technologies for Tactical Information Management in NetCentric Systems”, in Proceedings of the Defense Transformation and NetCentric Systems Conference, 2007.

[16] Sawashima, H., Hori, Y., Sunahara, H., & Oie, Y. (1997, June). Characteristics of UDP packet loss: Effect of tcp traffic. In Proceedings of INET’97: The Seventh Annual Conference of the Internet Society

[17] JBitTorrent Project , http://jbittorrent.sourceforge.net/.

[18] De Boever, Jorn. Peer-to-peer networks as a distribution and publishing model. na, 2007

[19] Napster, L. L. C. "Napster." URL: http://www. napster. com (2001).

[20] Adar, Eytan, and Bernardo A. Huberman. "Free riding on Gnutella." First Monday 5.10 (2000).

[21] eDonkey2000, http://www.edonkey2000.com, 2003

[22] Kazaa, “Kazaa Homepage”, http://www.kazaa.com/us/index.htm, 2003

[23] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubia-towicz, “Tapestry: A Resilient Global-scale Overlay for Service Deployment”, IEEE Journal on Selected Areas in Communications , 22(1):41–53, 2004

[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications”, In Proceedings of the 2001 ACM Sigcomm Conference, pp. 149–160, ACM Press, 2001.

[25] P. Maymounkov and D. Mazieres, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric”, In International Workshop on Peer-to-Peer Systems (IPTPS’02), 2002

[26] Xie, S., Li, B., Keung, G. Y., & Zhang, X. (2007). Coolstreaming: Design, theory, and practice. Multimedia, IEEE Transactions on, 9(8), 1661-1671.

[27] RTI Message Service, "Configuration and Operation Manual," Real-Time Innovations, Inc., December 2013

[28] Huang, Gale. "PPLive: A practical P2P live system with huge amount of users." Proceedings of the ACM SIGCOMM Workshop on Peer-to-Peer Streaming and IPTV Workshop. 2007

[29] Lu, Y., Fallica, B., Kuipers, F. A., Kooij, R. E., & Van Mieghem, P. (2009). Assessing the Quality of Experience of SopCast. IJIPT, 4(1), 11-23

[30] Baset, S. A., & Schulzrinne, H. (2004). An analysis of the skype peer-to-peer internet telephony protocol. arXiv preprint cs/0412017.

[31] Ozzie, R. (2005). Microsoft, Groove Networks to Combine Forces to Create Anytime, Anywhere Collaboration. Microsoft News Center.

[32] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). SETI@ home: an experiment in public-resource computing. Communications of the ACM, 45(11), 56-61

[33] "Java SE | Oracle Technology Network | Oracle." [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html

[34] "NetBeans IDE" [Online]. Available at https://netbeans.org.

[35] RTI-DDS is available at http://www.rti.com.

[36] GNS3. Ltd, http://www.gns3.net/.

[37] Quental, N. C., & da S Goncalves, P. A. (2011, January). Exploiting application-layer strategies for improving BitTorrent performance over MANETs. In Consumer Communications and Networking Conference (CCNC), 2011 IEEE (pp. 691-692). IEEE

[38] Nowak, M., & Sigmund, K. (1993). A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game. Nature, 364(6432), 56-58.

[39] DevGroup, B. BitComet. 2006-4-30]. http://www. bitcomet.com.

[40] Hazel, G. (2012). uTorrent Transport Protocol library.

[41] BitTorrent Specification 2014, https://wiki.theory.org/BitTorrentSpecification.

[42] Cohen, B. (2003, June). Incentives build robustness in BitTorrent. In Workshop on Economics of Peer-to-Peer systems (Vol. 6, pp. 68-72)

[43] M. AnisMastouri and S. Hasnaoui, "Performance of a Publish/Subscribe Middleware for the Real-Time Distributed Control systems Summary," 2007

[44] Schneider, S., & Farabaugh, B. (2009). Is DDS for You?. A Whitepaper by Real-Time Innovations, 1-5.

[45] OpenSplice is available at http://www.prismtech.com.

[46] OpenDDS is available at http://www.ociweb.com.

[47] CoreDX is available at http://www.twinoakscomputing.com/.

[48] RTI Connext, "Core Library and Utilities, QoS Reference Guide," Real Time Innovations, Inc., 2013

[49] RTI Message Service, "user's Manual," Real-Time Innovations, Inc., December 2013.

[50] DHT Protocol - BitTorrent.org, http://www.bittorrent.org/beps/bep_0005.html.

[51] Neglia, G., Reina, G., Zhang, H., Towsley, D. F., Venkataramani, A., & Danaher, J. S. (2007, May). Availability in BitTorrent Systems. In INFOCOM(pp. 2216-2224).

[52] Hoffman, J. "Multitracker metadata entry specification." (2004).

[53] PeerExchange,"http://wiki.vuze.eom/w/Peer_Exchange", last visited: April: 2014

[54] Wu, D., Dhungel, P., Hei, X., Zhang, C., & Ross, K. W. (2010, August). Understanding peer exchange in bittorrent systems. In Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on (pp. 1-8). IEEE.

[55] R.Vliegendhart. "Swarm Discovery in Tribler using 2-Hop TorrentSmell". MSc thesis, Delft University of Technology, May 2010.

[56] Luo, J., Xiao, B., Bu, K., & Zhou, S. (2013). Understanding and Improving Piece-Related Algorithms in the BitTorrent Protocol. IEEE Transactions on Parallel and Distributed Systems, 1.

[57] RTI Connext, "Core Library and Utilities, User's Manual," Real Time Innovations, Inc., 2012

[58] Oracle, V. M. (2012). VirtualBox. Programming Guide and Reference. 2012-06-20]. http://download. virtualbox, org/virtual-box/SDKRef, pdf.

[59] Combs, G. (2007). Wireshark. Web page: http://www.wireshark.org/ last modified, 12-02.

[60] Ben Jones. "How to Make the Best Torrents". Available at http://torrentfreak.com/how-to-make-the-best-torrents-081121/.

[61] Bindal, R., Cao, P., Chan, W., Medved, J., Suwala, G., Bates, T., & Zhang, A. (2006). Improving traffic locality in BitTorrent via biased neighbor selection. In Distributed

Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on (pp. 66-66). IEEE.

[62] Izal, M., Urvoy-Keller, G., Biersack, E. W., Felber, P. A., Al Hamra, A., & Garces-Erice, L. (2004). Dissecting bittorrent: Five months in a torrent's lifetime. In Passive and Active Network Measurement (pp. 1-11). Springer Berlin Heidelberg.

[63] Quang Hieu Vu, Mihai Lupu, & Beng Chin Ooi. (2010). Architecture of Peer-to-Peer Systems. In: Handbook of Peer-to-Peer Computing. Springer Berlin Heidelberg. pp. 11-37.

Vitae

Name: Anas Ahmed Abu Dagga

Nationality: Palestinian

Date of Birth:3/29/1986

Email: anas-ahmed@msn.com

Address: King Fahd University of Petroleum and Minerals Dhahran, 31261, Saudi Arabia

Education:

2003 to 2004	**General Secondary School Certificate**- GPA "90.8%", Palestine- Gaza Strip- Khanyounis.
2004 to 2009	**Bachelor in Computer Engineering**, GPA "82.6". Islamic University of Gaza (IUGAZA). Gaza-Palestine.
2012 to 2014	**MSc degree in computer networks**, King Fahd University for Petroleum and Mineral, GPA "3.57 out of 4", Dhahran-Saudi Arabia.

Publication

Basem Al-madani, Anas Abu-Dagga. Dynamic Pointers: Novel Discovery Protocol for BitTorrent Based on Real Time Publish Susbcribe (RTPS) Middleware. (to be published).

Basem Al-madani, Anas Abu-Dagga. DDS-BitTorrent: BitTorrent Implementation and Performance Enhancement using Publish-Subscribe Paradigm. (to be published).