Lookup Analyzer in BitTorrent Mainline DHT
Look@MLKademlia

Master Thesis in Computer Science

Sara Dar

TSLab. Royal Institute of Technology (KTH)

Stockholm, September 28th, 2010

Author
Sara Dar (Student MS Internetwroking, Royal Institute of Technology)

Title
Lookup Analyzer for BitTorrent Mainline DHT (Look@MLKademlia)

Presentation Date
September 28th, 2010

Graduation Committee

Examiner
Björn Knutsson, (Associate Professor) Royal Institute of Technology (KTH)

Supervisor
Raúl Jiménez (PhD Student) Royal Institute of Technology (KTH)

Abstract

Study and analysis of deployed DHTs requires monitoring of DHT traffic using some tool. Various types of tool have been developed and used by the researchers targeted to their own requirements, hence lack generalness and free availability.

This thesis work targets to analyze mainline DHT (MDHT) based on Kademlia by developing a generic, open source tool named – Look@MLKademlia. Look@MLKademlia is helpful for a large community of researchers, developers and students to study and analyze lookup behavior of MDHT. The tool not only presents data for analysis but also do some analysis using graphs and statistical data.

The scope of Look@MLKademlia is targeted to highlight Kademlia implementation behavior on a single node like an active monitor. It works on the real deployments where millions of nodes are present. It presents that data in graphical user interface form which makes it easier and quicker to investigate and explore the overall behavior of MDHT implementation in general and its lookup behavior in particular.

Look@MLKademlia is an exploratory tool which can indicate different behaviors of various implementations of Kademlia. One such example was seen when it was used by a student in his thesis while he was doing experiments on Kademlia network.

Look@MLKademlia has been tested with multiple captures from different BitTorrent clients using MDHT to evaluate the research capability of the tool.

Abstrakt

Analys av uppställda DHT:er behöver monitorering av DHT trafik med hjälp av någon slags verktyg. Olika typer av verktygen har utvecklats och används av forskarna men dessa verktygen är väldigt behovsspecifika för forskarna dvs verktygen är inte alllmänt tillgängliga.

Denna avhandling analyserar mainline DHT (MDHT) som är baserad på Kademlia, genom att utveckla en generisk, öppen källkod verktyg och heter Look@MLKademlia. Forskare, utvecklare och studenter kan använda Look@MLKademliaför att analysera MDHTs uppslagnings bteende. Verktyget presenterar data för analys samt analyserar det med hjälp av grafer och statistisk data.

Look@MLkademlia omfattar analys av Kademlias beteende på en nod som en aktiv monitor. Den använder verkliga grupperingar som innehåller miljoner noder. Den presenterar data i grafisk användargränssnitt som underlättar undersökning av MDHT implementation generellt och dess uppslagnings beteende specifikt.

Look@MLKademlia är en undersökningsverktyg som kan indikera olika beteende mönster hos olika implementationer av Kademlia. Ett sådant exempel träffade vi på när en student testade verktyget i sina experiment på Kademlia network.

Look@MLKademlia har testats med flera datafång från olika BitTorrent klienter som använder MDHT.

Acknowledgements

First, Thanks to my Allah, the most Gracious, and the most Merciful for opening the doors of knowledge on me and for making things easy for me to understand.
My special thanks to my examiner Björn Knutsson who gave me his precious time throughout this thesis. I believe that his every word was a valuable source of knowledge for me. I would also like to express my deep gratitude to my supervisor Raúl Jiménez whose continuous guidance and wonderful ideas made my thesis work easy and successful. Thanks to my colleague Ismael Saad Garcia for using my software and giving me feedback on it. Finally, thanks to my parents and other family members for providing me moral support during my studies.

Sara Dar
TSLab, KTH.
Stockholm. September 28th, 2010

Glossary

BitTorrent protocol	P2P file-sharing protocol
BitTorrent client	Application using BitTorrent protocol
Churn	The rate of nodes joining and leaving the DHT in a specific time interval.
DHT	Distributed Hash Table. One of the uses is, distributed trackers for peer discovery.
Mainline DHT	One of DHTs' implementations using Kademlia
Leecher	A peer that does not have all pieces of file.
Node	Member of DHT overlay.
Peer	Entity in P2P system, act as both client and server for sharing data
P2P	Peer-to-Peer
PEX	Peer Exchange. One of the techniques for implementing distributed tracker.
Seeder	A peer having all pieces of data and ready to share it.
Swarm	A group of peers sharing the same object
Tracker	Server which keeps information of all peers in a swarm
.torrent	Meta-data file which contains information about the file length, File name, infohash, and the URL of central tracker

Table of Contents

Abstract ..... **I**

Abstrakt ..... **I**

Acknowledgements ..... **III**

Glossary ..... **IV**

Table of Contents ..... **V**

List of Figures ..... **VIII**

Introduction **1**
1.1 Problem Statement 1
1.2 Motivation 2
1.3 Contribution 2
1.3.1 Comparison with Existing Tools 3
1.3.2 Features of Look@MLkademlia 3
1.3.3 Look@MLKademlia in the Research Perspective 4
1.4 Scope 5
1.5 Outline 5

Background **6**
2.1 Peer-to-Peer networks 6
2.2 Peer-to-Peer Applications 7
2.3 Centralized Peer Discovery (Lookup) 8
2.3.1 Napster - Central Directory Architecture 8
2.3.2 KaZaA - Super Nodes 10
2.3.3 BitTorrent - Central Tracker 11
2.4 Decentralized Peer Discovery (Lookup) 12
2.4.1 Gnutella (LimeWire) - Query Flooding 12
2.4.2 Peers Exchange 13
2.4.3 Distributed Hash Table 14
2.5 Kademlia 15
2.5.1 Kademlia Remote Procedure Calls (RPCs) 18

Methodology **19**
3.1 Literature Study 19
3.2 Scope Determination 20
3.3 Tools Selection 21
3.4 Platform 21
3.5 Data Display 21
3.6 Testing Strategies 22

3.6.1 Data Validity Testing 22
3.6.2 Performance Testing 22
3.6.2 Comparative Analysis 22
3.6.3 Research Capability Evaluation 22

Design and Development 23
4.1 Design 23
4.2 Development 24
4.2.1 Graphical User Interface (GUI) 24
4.2.2 File Reading 25
4.2.3 Binary encoded data parsing 25
4.2.4 Query, Responses and Errors Parsing 25
4.2.5 Graphs 25
4.2.6 IP and ID aliasing 25
4.2.7 Geographical Location of Nodes 26

Testing and Evaluation 27
5.1 Data Validity 27
Observations 27
5.2 Performance Testing 28
Observations 28
5.3 Comparative Analysis 28
5.3.1 Offline and Online Packet Capturing 28
5.3.2 Bencoded Dictionaries 29
5.3.3 Queries and Responses Dissection 29
5.3.4 Filters 33
5.3.5 Lookup Behavior 34
5.3.6 Graphs 35
5.3.7 Quick Statistical View 38
5.3.8 Queries with no responses and Vice Versa 39
5.3.9 Highlighting some Known Unusual Behavior 39
5.3.11 Geographical locations of nodes 42
5.4 Research Capability Evaluation 43
5.4.1 Kademlia Behavior 43
5.4.2 Corroborating Tool for Results of Experiments 55
5.5 Discussion 57

Related Work 58
6.1 Existing DHTs 58
6.1.1 Chord 58
6.1.2 Pastry 59
6.1.3 Kademlia 59
6.2 Monitoring and Measurement Tools 62
6.2.1 Passive Monitors 62
6.2.2 Montra 62
6.2.3 Mistral 63

6.2.4 Blizzard 64
6.2.5 Cruiser, kFetch and kLookup 64

Conclusion and Future Work 65
7.1 Conclusion 65
7.2 Future Work 66

References 67

Appendix A 70
Kademlia Queries in Mainline DHT 70

Appendix B 72
User Manual 72

List of Figures

1.1 Scope of Look@MLKademlia 5
2.1 Client/Server and P2P Models 7
2.2 Cluster of Central Indexing Servers in Napster 9
2.3 Central Directory based Architecture of Napster 9
2.4 Super Nodes based KaZaA Architecture 10
2.5 BitTorrent File Sharing 12
2.6 Gnutella Architecture 13
2.7 Iterative Routing 15
2.8 Recursive Routing 15
2.9 k-bucket Concept in Kademlia 16
2.10 Kademlia Lookup. Node 0011… searching for 1110… in network 17
3.1 Methodology Adopted in Thesis 19
3.2 Scope of the Tool-Look@MLKademlia 20
4.1 Module diagram of Look@MLKademlia 23
5.1 Get_peers Query in Wireshark 30
5.2 Get_peers Response in Wireshark 31
5.3 Get_peers Query and Response in Look@MLKademlia 32
5.4 Writing Filter in Look@MLKademlia 33
5.5 Filter Source Address and Infohash in Look@MLKademlia 34
5.6 Lookup Behavior in Look@MLKademlia 35
5.7 Lookup Convergence Graph in Look@MLKademlia 36
5.8 Round Trip Time from a Source in Look@MLKademlia 36
5.9 Total traffic generated in Look@MLKademlia 37
5.10 Queries and responses trend in Look@MLKademlia 38
5.11 Statistical Data in Look@MLKademlia 38
5.12 Queries with no Response in Look@MLKademlia 39
5.13 ID Aliasing in Look@MLKademlia 40
5.14 IP Aliasing Data in Look@MLKademlia 41
5.15 Errors in Look@MLKademlia 41
5.16 Errors in Wireshark 42
5.17 Geographical Location in Look@MLKademlia 42
5.18 uTorrent Lookup Convergence 44
5.19 NextShare Lookup Convergence 45
5.20 Mainline BitTorrent Convergence 46
5.21 Zoom Image of Figure 5.20 (Lookup convergence in Mainline BitTorrent) 47
5.22 uTorrent Generated Traffic 48
5.23 NextShare Generated Traffic 48
5.24 Mainline BitTorrent Generated Traffic 49
5.25 Queries and Responses in Mainline BitTorrent 50
5.26 Zoom Image of lowest part of Figure 5.25 (Queries and Responses in Mainline BitTorrent) 51
5.27 Queries and Responses in uTorrent 52
5.28 Zoom Image of 5.27 (Queries and Responses in uTorrent) 52
5.29 Queries and Responses in NextShare 53

5.30 Round Trip Time of get_peers Queries in Mainline BitTorrent 54
5.31 Round Trip Time of get_peers Queries in uTorrent 54
5.32 Round Trip Time of get_peers Queries in NextShare 55
6.1 Chord Lookup Process and Finger Table..... 59
6.2 Montra Architecture..... 63

Chapter 1

Introduction

This chapter presents the problem that is addressed in this thesis work. It also explains the unique contribution and provides the readers with general and specific features of the tool developed during this thesis work.

1.1 Problem Statement

"Developing an open source tool that can analyze Kademlia lookup process in Mainline Distributed Hash Table (MDHT)"

Kademlia [1] has been studied and analyzed in different perspectives where some of them are churn, lookup latency, parallelism, redundancy, session times, and geographical location of nodes. Moreover we see that there are multiple implementations of Kademlia used in different file sharing protocols. For instance Kademlia used in eMule and eDonkey is named as Kad while one used in some BitTorrent clients is MDHT and the other ADHT used in Azureus.

One aspect that is observed by the author of the thesis is that every researcher has used its own tool to study, measure and analyze Kademlia to draw conclusions of his/her experiments. Some of them do not mention how they took measurements but some have given description of their tools. Most of the tools we came across were crawlers which take snapshots of the network by sending different messages and use these snapshots to evaluate the DHT in hand. For example Cruiser [9], Blizzard [6], kFetch [9] are some of the crawlers used by researchers. Stutzbach et al. [9] have used a tool named kLookup for taking lookups over Kad network by using randomly selected source and randomly selected destination. The tools other than crawlers were active or passive monitors which capture traffic for offline analysis. (See details about all these tools in Chapter 6)

Besides researchers, a big community of developers is also working on different implementations of Kademlia to optimize and improve it. These developers need some analyzer or some testing tool that can show the effect of their changes in source code, better if in real environment. In addition to the researchers and the developers, the community of students is also in need of some easier and quicker way of learning Kademlia.

According to the best of my knowledge, there is no tool available publicly that particularly caters the needs of researchers, developers and students who want to understand and analyze Kademlia implementations.

The work done in the thesis targets to the community of the researchers, the developers and the students who want to explore, investigate and understand different aspects of

Mainline DHT implementation based on Kademlia in general and its lookup process in particular.

1.2 Motivation

Traffic monitoring is a commonly used method of analyzing working and behavior of any network protocol. If such monitoring is done in real environment, it can highlight many important aspects of the protocol under study. It can indicate unusual behaviors such as attacks on the network. It can also show effect of changes done on the protocol design.
In brief, for understanding multiple aspects of a network protocol, monitoring its traffic can be very helpful.

The same statement applies to Mainline Distributed hash table (MDHT) protocol based on Kademlia design. As mentioned in the problem statement, Kademlia has been studied and analyzed by a vast community of researchers.

Traffic monitoring requires some tool which can capture traffic and present it in useful format. There are many traffic capturing tools available like Wireshark but according to the author's knowledge, there is no tool available as open source which can help particularly in the analysis of Kademlia.

The goal of the thesis targets to develop an open source tool which will provide several benefits including real time traffic monitoring and analysis of Kademlia in Mainline DHT, testing of Kademlia implementation and easy learning of Kademlia lookup process.

1.3 Contribution

Due to the motivation, discussed in the previous section, a tool named Look@MLKademlia has been developed in Python during this thesis work which presents Mainline DHT (based on Kademlia) traffic in a form which makes it easier to be studied and analyzed.

The unique contribution lies in the design and development of this GUI tool. Also the testing and evaluation in different perspectives and finally this report are part of the unique contribution.

For researchers, the tool highlights some important parameters that are needed to be measured and analyzed. Graphs and statistical data presented in the tool accelerate and ease the analysis process. Geographical location of nodes shown by the tool helps the researchers understand geographical distribution of the Kademlia nodes. The tool also highlights the convergence of the protocol towards the destination key.

If Look@MLKademlia is analyzing some closed implementation of Kademlia, one can make very good guess about hidden parameters like parallelism, timeout value,

publishing behavior and management policies. When these parameters are already known, their effect on lookup latency can be easily studied using this tool. Moreover, different behaviors like ID and IP aliasing, errors are also highlighted in the tool.

Look@MLKademlia is equally helpful for the developers who want to observe impact of changes they make in the code. For example, if some developer is changing degree of parallelism or timeout value in the source code and wants to readily see the impact of that change, Look@MLKademlia is able to do this job.

This tool presents the overall lookup process in the graphical user interface form that makes understanding and learning Kademlia easier for the students.

1.3.1 Comparison with Existing Tools

Look@MLKademlia is different from crawlers as it does not take snapshots from the network and tells you the status of the whole network. It is more like a traffic analyzer tool which by presenting that traffic in a useful form, provides you the platform to see how a given Kademlia implementation works. In other words, it is analyzing single node's behavior in contrast to crawlers which target to the behavior of large number of nodes or the large part of the network. To elaborate more, Look@MLKademlia is a tool for investigating, exploring and discovering overall behavior of Mainline DHT based on Kademlia on one node in general and its lookup behavior in particular.

Look@MLKademlia has some similarity with kLookup (described in chapter 5) as it also shows how lookup process in Kademlia works. For example kLookup gives you an overview that how degree of parallelism effects the overall lookup process. The same thing can be tested by this tool if you have control to change the degree of parallelism. But there is one known limitation of kLookup – it is not open source. Also the other tools mentioned above are not available publicly. This limitation makes these tools unable to be used by anyone except their owners.

Look@MLKademlia seems very much similar to Wireshark. That is why a detailed discussion has been done in Chapter 6 to highlight its advantages over Wireshark in perspective of Kademlia analysis.

To sum up, since Look@MLKademlia is available as an open source, it can help a large academic community easier and can save much of their time and effort.

1.3.2 Features of Look@MLKademlia

The most attractive features of Look@MLKademlia are

- ✓ Easy to use filters which can filter data on parameters like source and destinations addresses, port numbers, query type, query and response time, Round Trip Time, transaction ID and some other parameters.
- ✓ Graphs representing different parameters such as lookup latency, number of queries and responses, traffic generated and convergence of the protocol.

- ✓ Statistical tables showing number of successful and unsuccessful queries along with the average round trip time.
- ✓ Display of queries and relative responses in parallel to have a quick look.
- ✓ Show convergence of lookup towards its destination key
- ✓ Readable form of binary encoded data
- ✓ Identifying geographical location of nodes.
- ✓ Some known unusual behaviors such as IP aliasing, ID aliasing.

1.3.3 Look@MLKademlia in the Research Perspective

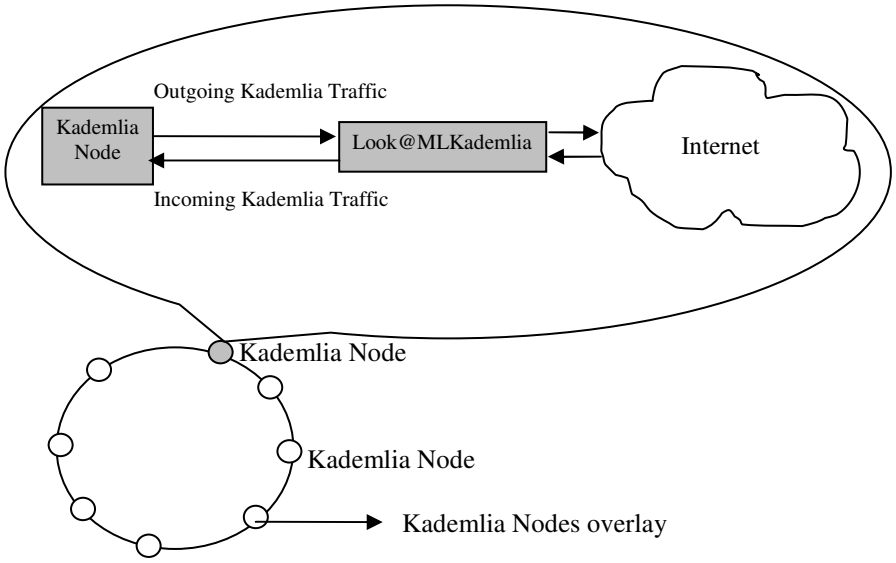
Briefly, Look@MLKademlia helps answering multiple questions; some of which are as follows.

- ✓ How a Kademlia implementation is working?
- ✓ What is the lookup performance?
- ✓ How is the convergence behavior? Is the protocol converging right? Is the convergence slow or fast?
- ✓ If some parameter like parallelism is changed, what will be the effect on lookup?
- ✓ How much traffic is being generated?
- ✓ What is the pattern of queries and responses?
- ✓ What is Round Trip Time (RTT) of all queries and each type of query?

Look@MLKademlia is not limited to provide answers of the above questions only. It presents the real deployment traffic in a friendly way which can answer many questions of the user who is interested to know the Kademlia behavior on a single node. It provides the user, several ways for exploring MDHT based on Kademlia. It can be a powerful tool for corroborating results of experiments done on MDHT. For example, in this report, a thesis work by another student has been discussed who has used Look@MLKademlia for confirmation of his experiments' results.

1.4 Scope

Look@MLKademlia is targeting to highlight single node behavior as shown in the Figure 1.1. (Refer to Section 3.2 for details)



1.1 Scope of Look@MLKademlia

1.5 Outline

The rest of the report is arranged in the following chapters. Chapter 2 gives a detailed literature review on Peer-to-peer networks, different peer discovery mechanisms and Kademlia. Chapter 3 discusses methodology adopted to achieve the thesis goals. Design and Implementation details are covered in Chapter 4. Chapter 5 focuses on Testing and evaluation of the work done. The related research work on various DHTs is given in Chapter 6. Chapter 6 also throws light on various tools used by researchers to monitor Kademlia traffic. Chapter 7 concludes the work and defines future work.

Chapter 2

Background

This chapter helps the reader to establish background knowledge on peer-to-peer networks and related technologies. It also gives a detailed discussion on different peer discovery mechanism. Moreover, the chapter provides a detailed overview of Kademlia, the main focus of this thesis.

Since the focal point of this thesis is peer discovery or lookup process in MDHT which is based on Kademlia design, various discovery mechanism have been discussed in this chapter in detail. The purpose of this detailed discussion is to make the reader clear about how the peer discovery or lookup mechanism works in different P2P applications and how these discovery mechanisms are different from each other. This discussion is important to make the position of Kademlia DHT design clear for the readers.

The major purpose of the chapter is to help those readers who do not have sufficient knowledge about P2P networks, P2P applications, Peer discovery mechanisms and Kademlia. The readers, who are well aware of these topics, can skip this chapter.

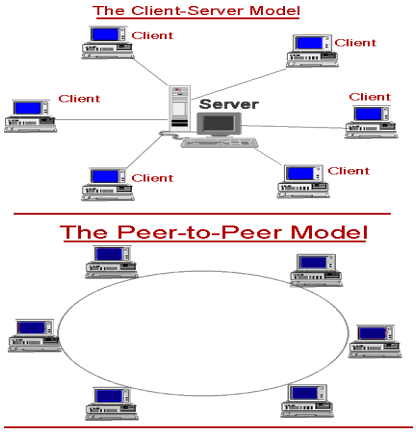
2.1 Peer-to-Peer networks

The main incentive for embracing peer-to-peer architecture over the last few years is its completely different architecture from the long-established client/server architecture which experiences the problems of single point of failure, restricted resources and limited scalability. Schollmeier [11] gives two precise definitions on peer-to-peer networks which are quoted here.

Definition 1"A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servant)."

Definition II "A distributed network architecture has to be classified as a pure Peer-to Peer network, if it is firstly a Peer-to-Peer network according to Definition 1 and secondly if any single, arbitrary chosen Terminal Entity can be removed from the network without having the network suffering any loss of network service."

The above definitions clearly state that in peer-to-peer networks, there are multiple advantages over the client/server architecture. Among them, resource sharing is the major advantage which makes P2P architecture the pragmatic solution for heavy resources demanding applications. P2P networks virtually provide unlimited resources over the network in the form of CPU, memory, power and bandwidth. Figure 2.1 shows the difference between P2P and client/server architecture.



2.1 Client/Server and P2P Models

Reproduced from http://www.ibiblio.org/team/intro/search/

Using the above definitions, we can define peer-to-peer network as a network where every peer is availing services from others like a client as well as providing services to others like a server without becoming the single point of failure.

Due to inherent scalability and robustness in the P2P networks, it seemed to be a realistic solution for today's internet where there is an ever growing trend of increasing users and network applications. These network applications have significantly increased the demand for resources such as bandwidth, storage, processing power. In this all time resource hungry environment, emergence of peer-to-peer (P2P) technology is not less than a blessing. Due to this incentive, many P2P applications came into the sight in the last few years.

2.2 Peer-to-Peer Applications

As discussed earlier, many P2P applications came into sight in the last few years, among which a few of them gained popularity by showing practical implementation in the real world. Among these few applications, file sharing applications using BitTorrent protocol [26] have shown significant acceptance by users [27, 28, 29]. Other popular P2P applications are eMule[1], LimeWire [2](Gnutella Client), Skype[3], PPLive[4], KaZaA[5].

Although all of the features of P2P are under study and research, peer discovery mechanisms of peers and routing between those peers has witnessed significant attention

[1] eMule, www.emule.com Last Visited July, 2010
[2] LimeWire, http://www.limewire.com/en Last Visited July,2010
[3] 'Skype', http://www.skype.com/intl/en/home, Last Visited July, 2010
[4] 'PPLive', http://pplive.en.softonic.com/, Last Visited July, 2010
[5] KaZaA, http://www.kazaa.com/, Last Visited July, 2010

by the researchers. Peer discovery can be highly centralized such as in Napster, highly distributed like Gnutella or somewhere in between. [35]

2.3 Centralized Peer Discovery (Lookup)

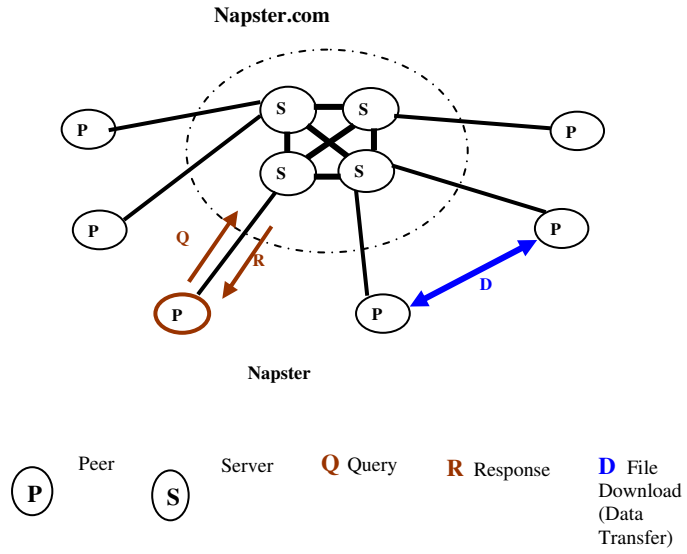
In this section, various peer discovery mechanisms have been discussed such as central directory used in Napster, super nodes in KaZaA and central tracker in BitTorrent.

All of the applications described in the following subsections are unstructured P2P applications. According to Qiao et al.[23], in unstructured P2P applications, the overlay graph is random and difficult to characterize. The data objects are placed randomly in the graph. In such a random environment, the peer discovery mechanisms ensure that the query covers a large set of peers in the network.

2.3.1 Napster - Central Directory Architecture

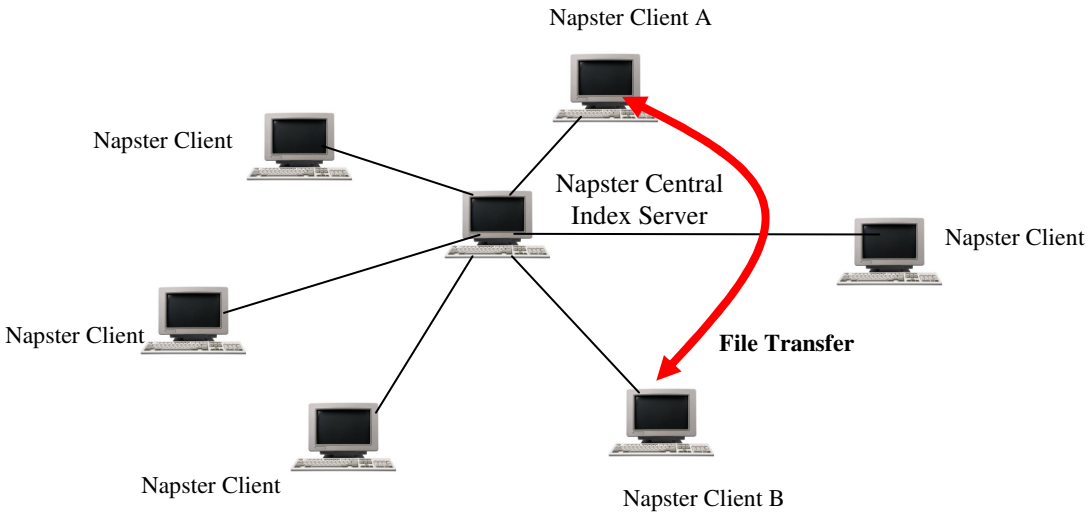
Napster was created in June 1999 by Shawn Fanning, student of Northeastern University. After a year or so, Napster was shut down when sued by the Recording Industry Association of America (RIAA) with the charge of copyright infringement.[6] It came into being again in 2003 as a legal music download site. Napster architecture was not a pure P2P as its indexing was centrally located. Users used to contact the index server to know the location of other users responsible for a certain piece of data.[7] In other words Napster architecture depended upon the central directory for searching the users who had the required data. That central directory is usually a cluster of many servers which communicate with each other to respond to the queries of users. [19] (Figure 2.2)
In Figure 2.3, it can bee seen that all Napster clients are connected to the central server. When Napster client A wants to download some file, it requests to central indexing server for the file details and its location. The server responds with the list of clients who have data. Client A contacts directly to some or all of the clients in the list for requesting the file. Say for example, client B respond to client A request for data. Data transfer is done directly between these clients without intervention of any server.

[6] Napster's High and Low Notes - Businessweek - August 14, 2000
http://www.businessweek.com/2000/00_33/b3694003.htm
[7] 'Napster', http://computer.howstuffworks.com/file-sharing.htm, Last visited July 2010.



2.2 Cluster of Central Indexing Servers in Napster

Reproduced from [19]



2.3 Central Directory based Architecture of Napster

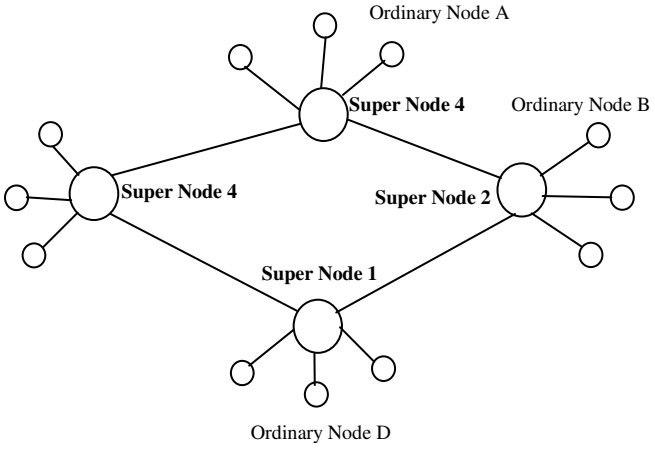
Reproduced from http://computer.howstuffworks.com/napster.htm

2.3.2 KaZaA - Super Nodes

At the same time when Napster was shut down and Gnutella (Section 2.4.1) set foot in the market, KaZaA also showed up. KaZaA resembles Gnutella that both do not depend upon centralized index server as in Napster. But unlike Gnutella, where all peers are at the same level, KaZaA architecture has two types of nodes: super nodes and ordinary nodes.

Super nodes have more responsibility than ordinary nodes and they are better in resources than ordinary nodes. Each ordinary node is associated with some super nodes. Super nodes serve as a small index server responsible for keeping record of files and keeper of those files. Unlike Napster, super node is not a dedicated system. It is any node with better resources such as bandwidth, memory and CPU power, thus exploited the heterogeneity of the network where all peers or nodes can never be at equal level in terms of resources. When an ordinary node wants a file, it sends query to its super node. If the super node has information about the file, it sends back the response. Otherwise it contacts other super nodes and forwards the query to them. [20]

To elaborate it more, Figure 2.4 shows that an ordinary node A is attached with a super node 4. When the node A wants to download some file it will contact super node 4. If the super node 4 has information of peers (users) who have the requested file, it will return that list of peers in response to node A. If it does not have that list, it will contact other super nodes to get that list of peers.



2.4 Super Nodes based KaZaA Architecture

Reproduced from [16]

2.3.3 BitTorrent - Central Tracker

BitTorrent is a P2P application for file sharing which is based on BitTorrent protocol [26]. Applications using BitTorrent protocol are known as BitTorrent clients. Many flavors of BitTorrent clients exist using this protocol. Some of them are KTorrent[8], uTorrent [9], Azureus[10], Mainline[11].

In BitTorrent protocol the peers that are participating in downloading the same file make a group of peers known as swarm. The peers join swarm by reading .torrent file which keeps info hashes of all pieces of the file to be downloaded. .torrent also has the information of tracker which informs about peers responsible for storing files.

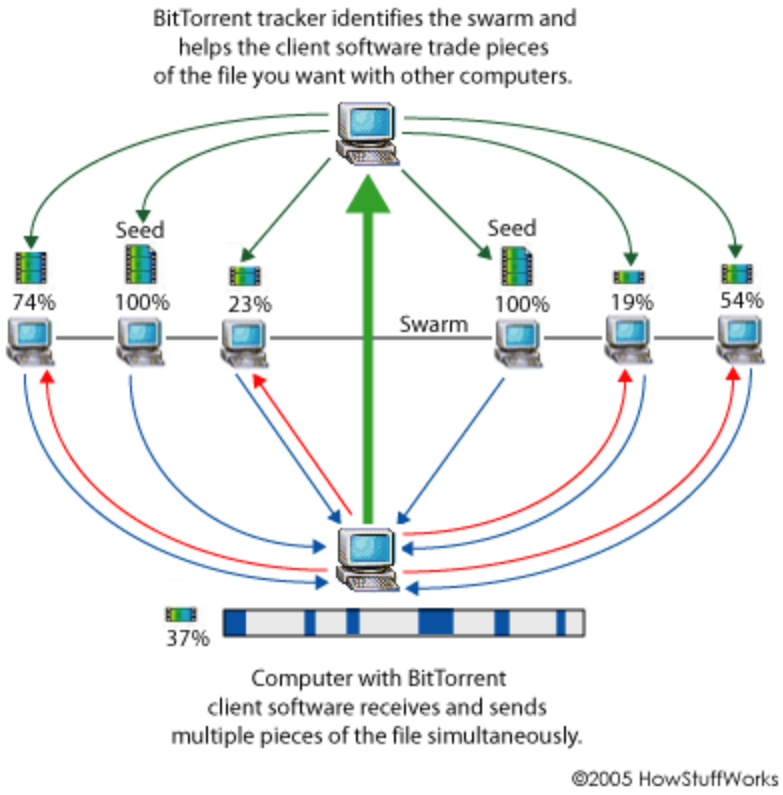
Information of the trackers is found from .torrent files. The information about location of these torrent files is found through the websites like mininova.org or legaltorrents.com. But torrent files themselves are not on the website server, rather the website links to torrent servers. User gets .torrent metadata file from these server to get tracker information from them. Trackers give list of peers (peers who are downloading the file) to the user (peer) who has asked for it [40].

When the peers are found with the help of trackers, they start downloading data from each other. Peers that have downloaded all pieces are known as seeders and the peers which are still not finished with downloading are known as leechers. Although, the seeders are the peers that are finished with downloading, remain in the swarm to let leechers download data from them. They do so due to tit-for-tat system of BitTorrent to avoid parasitic behavior of peers or in other words to avoid free riders who download data and vanish. This tit-for-tat system is for fair resource sharing in the system. [40]

Figure 2.5 is explaining BitTorrent protocol working using central tracker. From the Figure it can be seen that peers in the swarm who have done 100% downloading of the file are still in the swarm to share that data to other peers who need it. Central tracker here is providing peers with the information of other peers either seeders who have downloaded data or leechers who are still downloading data. This way peers come to know about each other and request for data.

---

[8] KTorrent, http://KTorrent.org/, Last Visited April, 2010
[9] uTorrent, http://www.utorrent.com/, Last Visited July, 2010
[10] 'Azureus', http://www.vuze.com/, Last Visited August, 2010
[11] BitTorrent, http://www.bittorrent.com/, Last Visited August, 2010



2.5 BitTorrent File Sharing

Reproduced from www.howstuffworks.com

According to Pouwelse et al. [41], in BitTorrent there are multiple central components – a tracker, indexing website, .torrent server and P2P moderation system to filter fake files. Dependency on these components makes system availability vulnerable to failures.

This limitation puts the need of making the system less dependent on the central components. The decentralized peer discovery mechanism was seen as a solution to reduce dependency on central trackers. This mechanism is realized by using distributed hash tables – DHTs. Peers exchange (PEX) is also a solution for distributed tracking.

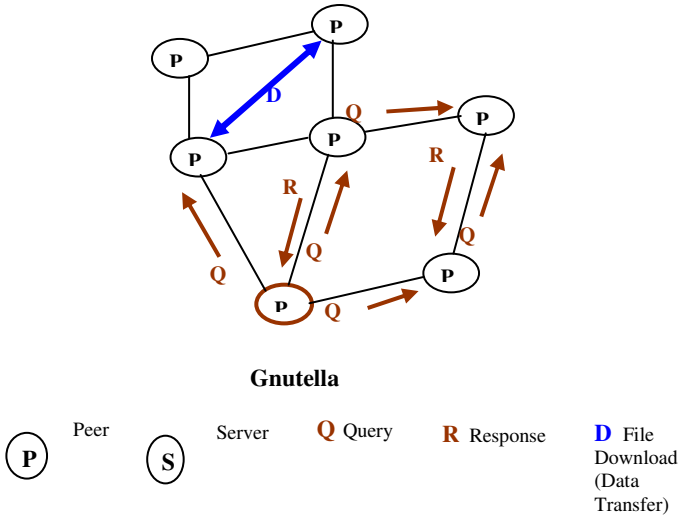
2.4 Decentralized Peer Discovery (Lookup)

Due to the single point of failure problem in centralized peer discovery, efforts were put to decentralized peer discovery where the discovery is not dependent on a central component. This results into query flooding architecture of Gnutella and Peer Exchange method in BitTorrent in unstructured overlays and DHTs in structured overlays.

2.4.1 Gnutella (LimeWire) - Query Flooding

After the decline of Napster in the early 2000, Gnutella came into being with a better architecture. It had no dependency on the central indexing server. The query flooding architecture of Gnutella made it independent of central server where a query is sent by a

user to all neighboring users which in turn, forward that query to their neighbors. The query, as a result is flooded to many peers. If a peer has requested data, it sends response back to the query originator. Data is transferred the same way as in Napster. To control the query being flooded forever, time to live TTL is used which is decremented with every hop visited [19]. Figure 2.6 is showing Gnutella architecture. It can be seen that in Gnutella Peers P flood query of other peer and find the peers which are responsible for the requested data. Here 'Q' stands for query, 'R' stands for response and 'D' is for data. Data transfer is done directly without intervention of other peers.



2.6 Gnutella Architecture

Reproduced From [19]

2.4.2 Peers Exchange

Peer exchange is only used in BitTorrent applications. In this method, peers exchange list of active peers directly with each other. In other words, the peers gossip with each other to know the list of active peers. In PEX, a peer needs to know atleast one other peer before it can start gossiping with peers for the list of peers. The method of finding that peer might be some peer discovery mechanism e.g. central tracker according to Wu et al [2] or some DHT. The bootstrapping mechanism of PEX is limited to one swarm at a time. To enter into the other swarm, it needs to bootstrap again.

According to Wu et al. [2], currently there is no standard of PEX. There are multiple PEX implementations used in BitTorrent clients such as AZ_PEX in Azureus, BT_PEX in BitComet and UT_PEX in uTorrent and kTorrent.

2.4.3 Distributed Hash Table

Qiao et al. [23] define structured P2P applications as applications where there is a tight control on the overlay structure and placement of data. Both, data object and nodes are assigned identifiers and query for data search is routed to the node responsible for it.

Commonly the overlays using Distributed hash tables are called structured overlays.

Due to the scalability issues in earlier unstructured P2P applications like Napster, Gnutella and KaZaA researchers' attention was diverted towards structured P2P applications based on distributed hash tables [23], commonly known as DHTs.

Due to their robust and scalable nature [30,31,32], DHTs proved to be attractive choice as distributed trackers for peer discovery, thus witnessed extensive research and analysis by the researchers. Many DHTs were introduced among which Kademlia [1], Chord [13], Pastry [14] and Tapestry [15] gained much attention in the researchers' community. (Refer to Chapter 6 for more information about DHTs)

DHTs is the same centralized hash table using <key, value> pair with the only difference that it is distributed over the whole overlay network. Any node in the network can retrieve value from any other node in the network using the key associated with that required value.

Many standards of DHTs have been proposed in the last few years; among them are Chord [13], Pastry [14], Tapestry [15] and Kademlia [1]. (Refer to Chapter 6 for details about these DHTs). All DHTs work on <key, value> system where files are associated with keys and nodes are responsible for storing a range of these keys. The value is the address of node associated with a certain range of keys. In DHTs, user looks for a file by using key. General name for this function is lookup (key). Lookup (key) has two operations; 'get' and 'put' which allows the user store and retrieve files on the node based on their associated key. [18]

In DHTs there is a need for a bootstrapping technique to enter in swarm. In BitTorrent clients, Bootstrapping servers provide the initial contacts to enter in the swarm. Once the node is entered in the swarm, it is part of whole overlay, unlike PEX which can only join one swarm at a time.

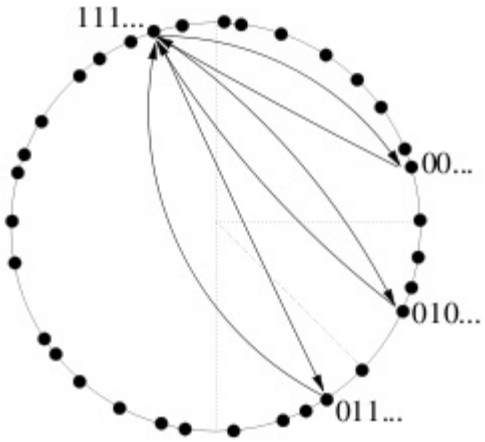
In DHT, nodes form an overlay where each node has several neighbors. When lookup(key) query is issued, it is routed through several nodes until it reaches the node which is responsible for the key. Ratnasamy et al. [18] mention in their paper that the scalability of the DHTs algorithms is tied to the efficiency of routing algorithms. All of the routing algorithms use essentially either of the two routing techniques; iterative routing or recursive routing.

In recursive routing (Figure 2.8) the request traverses through multiple nodes until it reaches the destination node. The selection of traversed peers is not in hands of the node

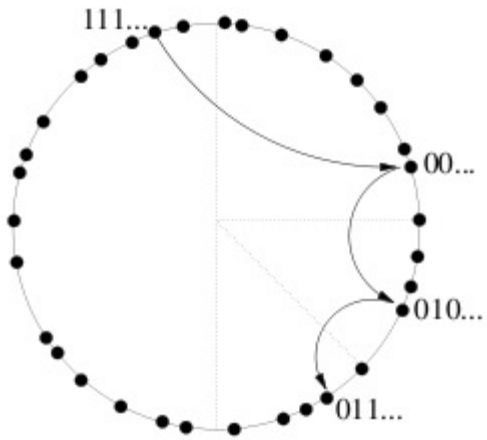
who is sending request. According to Kunzmann [17], the benefit of recursive routing is that it is fast and has less overhead.

In iterative routing, (Figure 2.7) the node sending the request gets back the response of each query sent. Response is either address of another node to which the request can be sent or address of node responsible for data. In iterative routing multiple queries can be sent at a time.

Kunzmann [17] mentions in his paper that iterative routing has two advantages. The originator of lookup can keep track of lookup route and can react to any problem in the route. The other advantage is that if originator notices lookup failure due to some node, it can change its lookup track. In brief, the originator has more control over the lookup process.



2.7 Iterative Routing
Reproduced from [37]



2.8 Recursive Routing
Reproduced from [37]

2.5 Kademlia

Kademlia [1] is one of the most used DHT overlay with some of its implementations exceeding over one million users [9].

Kademlia DHT has multiple implementations. One used in eMule[12] is known as Kad [5, 6, 7]. According to my knowledge, other two are Mainline DHT (MDHT) and Azureus DHT (ADHT). MDHT is used in some BitTorrent clients such as uTorrent, KTorrent, and

[12] Official eMule Homepage, http://www.emule-project.net/home/perl/general.cgi?l=1 Last Visited April,2010

Mainline while ADHT is employed in Azureus which is also a BitTorrent client. Both of ADHT and MDHT has been discussed in details by Crosby et al [3]. This thesis work targets to Mainline DHT implementation (MDHT) based on Kademlia.

Kademlia approach is similar to other DHTs where key is i60-bit identifier (e.g. SHA-1 hash of some larger data). Participating entity is node which is also having identifier from the same space of 160-bit identifier.<key,value> pairs are stored on the node whose identifier is closer to the key where the term closer refers to the XOR distance between the two IDs.

Routing Table

In Kademlia, each node's routing table keeps a list of <IP address, UDP port, Node ID> for nodes having distance in the range $[2^i , 2^{i+1})$ from the node where $0 <= i < 160$. These lists are known as k-buckets. Each k-bucket has contacts or nodes which are put in the same buckets because of sharing a common prefix with each other.

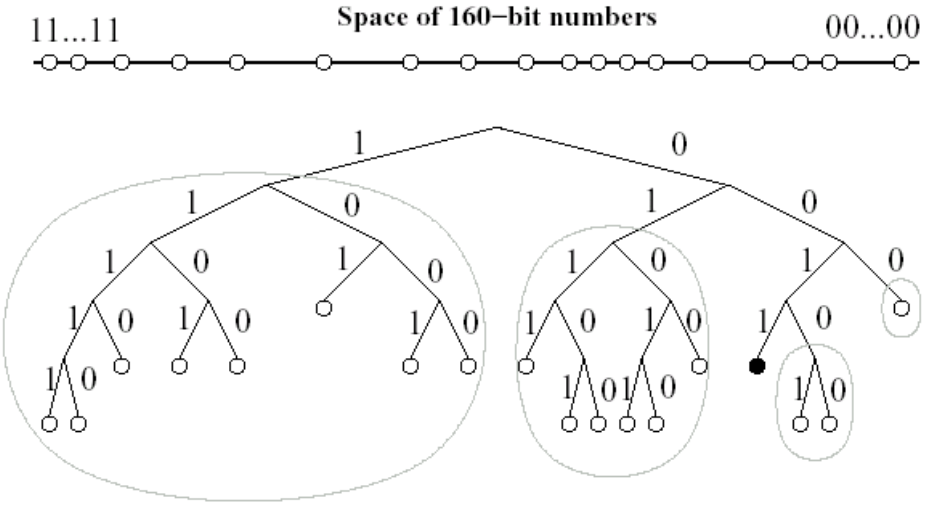


Fig. 1: Kademlia binary tree. The black dot shows the loation of node 0011... in the tree. Grey ovals show subtrees in which node 0011... must have a contact.

2.9 k-bucket Concept in Kademlia

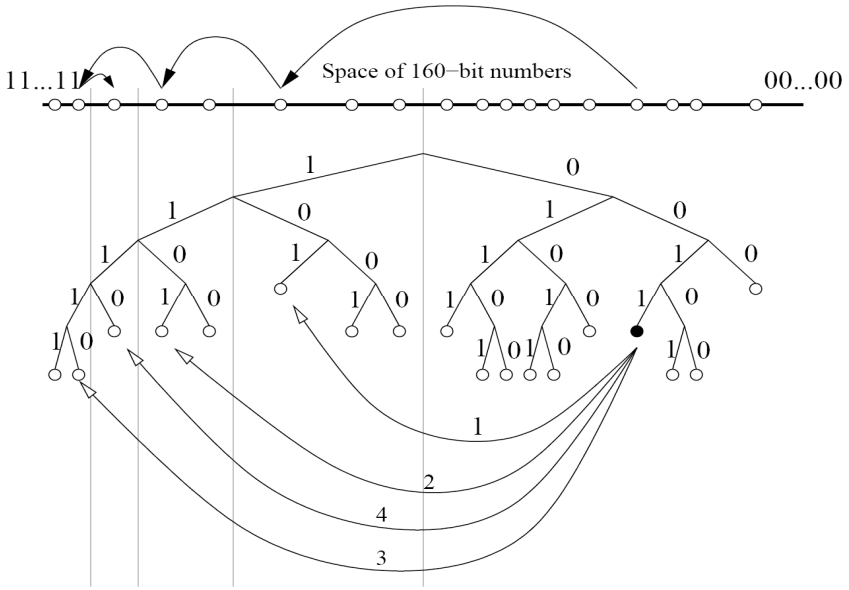
Reproduced from [1]

In Figure 2.9, we think of Kademlia network as a binary tree where nodes are arranged according to their node IDs. Taking node 0011(…) we see that this node has contacts in every subtree. Subtrees are shown as grey ovals. Each of these subtrees can be taken as k-bucket.

Lookup Process

For lookup process Kademlia uses iterative routing where the node which is starting lookup is responsible for the whole lookup process. The node starts with some nodes from its routing table and sends them query. These nodes are selected from the k-buckets in the routing table. The source node waits for the response from other nodes. On the basis of response, it sends more queries to other nodes which are closer to the ObjectID

or infohash. This process continues until source node does not receive any new nodes closer to the ObjectID.
In Figure 2.10, a node 0011… is searching for a node with identifier 1110…. in network. In the first iteration it contacted node with one prefix matched with the target i.e. 1. In the second iteration it is closer with first two prefixes matching i.e. 11. In the third iteration three prefixes matched and in the fourth iteration it reached its target.



2.10 Kademlia Lookup. Node 0011… searching for 1110… in network

Reproduced from [1]

Updating Routing Table
Routing table in the node keeps on updating itself by adding new nodes and removing stale nodes. Stale nodes are those who do not respond to 5 RPCs. (RPCs discussed in the next section). Refreshing of routing table entries is done in two ways.

One way of refreshing routing table works as a side effect of lookup process. When a node receives lookup query responses from other nodes, it updates its routing table on the basis of those responses.

In the other method, the routing table entries are refreshed by sending ping messages to the nodes in the routing table. If the nodes do not reply, they are considered stale and thus removed from the routing table.

When a node is to be added in the particular bucket, it is checked whether it is already in the bucket. If it is already there, then it is brought to the tail of the bucket where most recently seen nodes are placed. If the node is not in the bucket and bucket is not full then node is added in the bucket. If the bucket is full then least recently seen node is checked for aliveness. If it is not alive then new node replaces it. If it is alive then new node is discarded.

2.5.1 Kademlia Remote Procedure Calls (RPCs)

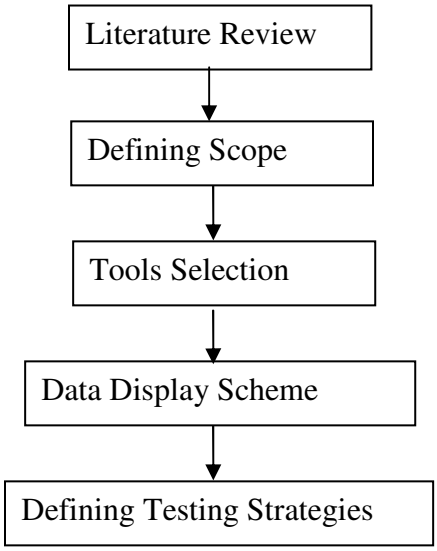
Kademlia design consists of four remote procedure calls: Ping, FIND_NODE, FIND_VALUE, and STORE. PING probes the node to check it aliveness. STORE is used to store <key, value> pair on a node for later retrieval. FIND_NODE sends 160-bit ID as an argument. In response of this RPC, <IP address, UDP Port, Node ID> triples are sent. The triples belong to the nodes that are closest to the target ID in view of recipient of RPC. FIND_VALUE behaves like FIND_NODE unless the recipient node receives STORE RPC for the key, it sends that stored value.

Chapter 3

Methodology

The overall goal of the thesis is to develop an open source tool which acts like a learning tool for students, a research instrument for the researchers and a testing tool for the developers at the same time.

To achieve this goal, I have gone through the following steps which were based on the idea of establishing sufficient knowledge on Kademlia, determination of scope, right tools selection for development and defining strategies for testing and evaluation.



3.1 Methodology Adopted in Thesis

3.1 Literature Study

Before the tool development could be started, it was important to study Kademlia in detail. Since the sole purpose of the tool was to highlight important aspects of mainline DHT based on Kademlia with one node perspective, it was important to know how the researchers, developers and students work on it.

Literature study was carried out in three steps where in the first step background knowledge on various peer discovery or lookup mechanisms was established. Also, Kademlia lookup process, its routing table's structure and it messaging structure was studied.

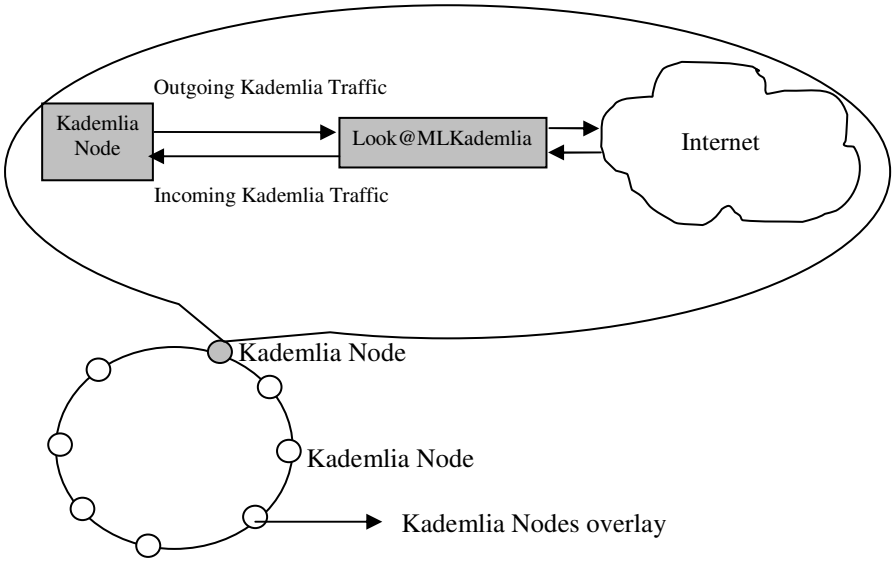
In the second step it was studied that how researchers have seen Kademlia. This step revealed that Kademlia has seen vast research in various perspectives including churn, lookup performance, parallelism, replication of data, publishing and searching relationship.

In the third step, it was investigated that which tools have been developed by the researchers for studying and analyzing Kademlia. It was also studied that what was the scope of these tools and how did they work.

3.2 Scope Determination

After having sufficient knowledge on Kademlia and MDHT, it was decided that the tool that would be developed during this thesis, will target Kademlia behavior on a single node. This excludes those behaviors that have been observed by the researchers over the whole network including churn, replication of data on multiple nodes and connectivity issues. It includes those behaviors that were related to single node including lookup latency, lookup cost, round trip time, lookup convergence and affect of different parameters like degree of parallelism.

In the above perspective, it is clear that the tool was not going to be a crawler that takes snapshot from multiple nodes or from a big part of the network. Rather, the tool would be capturing Kademlia traffic like an active monitor, sitting on a node which sends and receives queries. Further, the tool will highlight important characteristics of the Kademlia implementation on that node.



3.2 Scope of the Tool-Look@MLKademlia

3.3 Tools Selection

For development, Python[13] was selected because it was easy to use and understand. But the most incentivizing thing was the library 'kadtracker' that is developed in Python. This library has been developed in python by Raul[14] in TSLab[15], KTH. This library provides log distance between two node identifiers and it provides parsing of binary encoded data into messages form that made this thesis work easier.

Since the purpose of the tool was doing Kademlia analysis, graphs were also included as they provide a fast method of analysis. For graphs wx.lib.pot[16] was decided to be used as it was easy to learn and use. Also, it was fulfilling requirements of thesis.

GUI has been developed using wxPython GUI toolkit[17]. There are many strengths of wxPython such as its availability as open source, its flexibility as cross-platform toolkit and its ease to use and write programs.

The other libraries used are pcap and dkpt[18] for file reading, PyGeoIP for getting geographical location of nodes. PyGeoIP is a python API for MaxMind GeoIP databases[19].

3.4 Platform

The tool is developed in Ubuntu. Although, it is not tested on Windows and Mac, it is expected to work on these systems as libraries used in the code are cross-platform.

3.5 Data Display

For displaying data, it was decided that it would not be displayed like it is in Wireshark where data is displayed in time sequence. The problem in this display was that you have to find responses of the queries you are interested in. The tool developed in this thesis display queries and their related responses in one line. Each query and its response are in the same line, parallel to each other. This method of displaying data makes it quicker for the researcher, to analyze data. This way one can see query and response details in one go. No filtering is needed for it. But if the person working on the tool wants to see responses in time sequence, the tool provides the ability of sorting.

---

[13] 'Python', http://www.python.org/
[14] 'Raúl Jiménez', http://www.tslab.ssvl.kth.se/raul
[15] 'TSLab, KTH', http://tslab.ssvl.kth.se/
[16] 'wx.lib.plot', http://www.wxpython.org/docs/api/wx.lib.plot-module.html
[17] 'wxPython', http://www.wxpython.org/what.php
[18] 'dpkt', http://code.google.com/p/dpkt/
[19] 'Maxmind Database', http://www.maxmind.com/app/geolitecountry, Last Accessed July, 2010.

3.6 Testing Strategies

Multiple testing strategies were designed to test and evaluate the tool's results. This testing involves results validity, performance testing, comparative analysis and tool's research capability testing. (See Chapter 5 for details on Testing and Evaluation of the tool)

3.6.1 Data Validity Testing

In this test it was planned to check whether the tool is displaying the right data and also whether the tool is displaying the data right. For this purpose it was decided to match results with Wireshark.

3.6.2 Performance Testing

Performance testing was planned with the help of a tool named RunSnakeRun (Details in Chapter 5). The purpose of this testing was to check if the design of the tool is efficient.

3.6.2 Comparative Analysis

A detailed comparison was planned to highlight features of the tool that are not found in Wireshark. Also, the purpose of this comparison was to establish the importance of the tool over Wireshark when both are used particularly for Kademlia analysis. Moreover, this comparison emphasize that there are some features missing in Wireshark which hamper the process of analysis of Kademlia in Wireshark. (Details in Chapter 5)

3.6.3 Research Capability Evaluation

This test was decided to show that the tool developed was meeting promises, it made in the start of this thesis. These promises claimed that the tool would answers to the research questions mentioned in the introduction section of this work. For doing this test, three BitTorrent clients were used. All of these clients were Mainline DHT enabled. 8 captures of each client were taken to ensure the reliability of tests. (Details in Chapter 5)

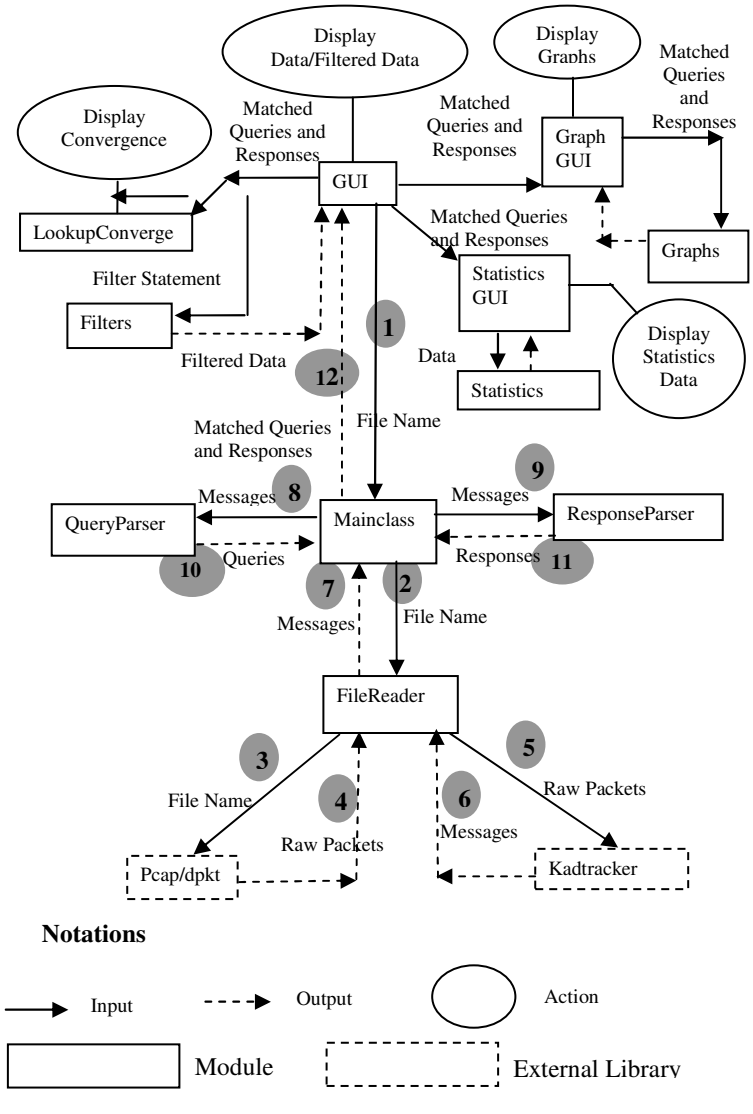
Chapter 4

Design and Development

This chapter presents the design and implementation of Look@MLKademlia.

4.1 Design

Figure 4.1 gives and overview of modules and their interaction with each other in Look@MLKademlia. According to the figure, there are 11 modules and two external libraries.



Note: At step 6, Messages means data has been parsed from binary encoded format.

4.1 Module diagram of Look@MLKademlia

The numbers mentioned in the module diagram in Figure 4.1 are indicating the flow of data when the user gives the file name through graphical user interface. Briefly, the file name given by user is sent to the module 'mainclass'(Step 1) which sends it to 'filereader'(Step 2) and finally to external libraries; 'dpkt' and 'kadtracker'(Steps 3 & 4). The file is read here and binary encoded data is parsed into 'messages'(Step 5).These packets are parsed separately into queries and responses by the 'queriesparser' and 'responsesparser' modules (Steps 10 and 11 respectively). The parsed queries and responses are sent to 'gui' module through main class. 'Mainclass' module matches these queries and responses on the basis of transaction ids and sends a list having queries and their matching responses to 'gui' module (Step 12). 'Gui' module displays them. 'gui' module keeps them throughout the execution time of the program so that all the process of parsing is not to be done again. This way, external libraries, 'filereader' module, 'queriesparser' module and 'responsesparser' module are accessed only once. This design saves much of the execution time, although it holds system memory uptill execution time.

Rest of the modules is event driven. They depend upon the input from the user and click of button or toolbars. For example, 'filters' module takes filter expression from the user and do filtering on queries and responses list saved in buffer. In the same way, 'graphs' module takes input from the user and plot graphs on the data taken from the same queries and matched responses list.

4.2 Development

Look@MLKademlia is developed in Python. Here the brief description is given about development of its major functions.

4.2.1 Graphical User Interface (GUI)

Module: gui.py
Class: GUI
In Look@MLKademlia code, gui.py is the main graphical interface class which interacts with other classes and display data. 'gui.py' takes parsed queries and responses from other classes and stores them in buffer for the whole time of code execution. Although using this method holds system memory uptill execution time, it saves multiple hard disk access for reading pcap file. Look@MLKademlia performance improved this way especially when pcap file was very large. Program does not need to read the file from hard disk more than once.
It became more important, when it was discovered that 'kadtracker' performs slower on large files (more than one hour capture).

The design of program works by reading file once and using kadtracker at the time of reading file. Data being accessed is stored in buffers throughout the running of the program.

4.2.2 File Reading

'filereader.py ' reads pcap file from disk using dptk package. In the same class kadtracker is used to parse binary encoded data into messages format. As discussed in the above section, pcap file is read only once and so does the kadtracker library. This is for improving performance of the program and for avoiding kadtracker performance issues, as discussed in the previous section.

Module: filereader.py
Class: FileReader

4.2.3 Binary encoded data parsing

Bencoded data conversion has been done using external library, kadtracker. 'Kadtracker' has broad functionality as it is developed as a standalone DHT client but in this thesis it has been used only for parsing binary encoded data into messages and for finding log distances between two node identifiers.

4.2.4 Query, Responses and Errors Parsing

In Look@MLKademlia, queries are matched with corresponding responses and errors by using transaction ID. Three classes are used for parsing queries and responses.

Module: queryparser.py
Class: QueriesBisector
This class is responsible for bisecting queries. It has five container classes. Queries class and four inherited classes; GetPeers, FindNode, AnnouncePeer and Ping.

Module: responseparse.py
Class: ResponseBisector
ResponseBisector class parse responses. It has one container class Responses.

Module: errorparser.py
Class: ErrorBisector
ErrorBisector class parses errors. It has one container class Errors.

4.2.5 Graphs

Wx.plotlib library is used for plotting graphs. This library is although limited in functionality than other library like matplotlib, but it was quick to learn and very easy to use. Also, it was fulfilling the requirement for plotting required graphs.
Modules: graphs.py and graphgui.py
Classes: GraphGui and Plot.
GraphGui takes processed data from class Plot to plot the graphs.

4.2.6 IP and ID aliasing

IP aliasing means multiple peers having the same identifier. 'Aliasing' class has two functions for finding IP and ID aliasing. These functions are ip_aliasing and id_aliasing.

‘ip_aliasing’ function collects all of the peers who have the same identifier and then removes redundant data. ‘id_aliasing’ works in the same way. It collects all peers having same IP but different identifiers.

Module: aliasing.py
Class: Aliasing

4.2.7 Geographical Location of Nodes

Module: nodeaddr.py
Class: Node

The geographical location of nodes has been added to see where the nodes are located which have been looked up for peer discovery.

Chapter 5

Testing and Evaluation

Multiple captures from three BitTorrent clients have been used for testing its results. These BitTorrent clients are KTorrent, uTorrent and NextShare[20]. These BitTorrent clients are all Mainline DHT enabled.

Testing has been done in four perspectives. In the first perspective it was checked whether Look@MLKademlia is displaying all the captured data. In the second perspective, it was tested that if Look@MLKademlia is performing well with large captures. The third perspective is the most important and detailed. Here we check that is Look@MLKademlia meeting all the promises it made in this work. In other words, it was checked if the tool is able to answer those research questions that are mentioned in the introduction section.

5.1 Data Validity

In this phase it is checked that if Look@MLKademlia is displaying all the data that has been captured by Wireshark. The importance of this test was obvious because if significant amount of data were missing, it could lead to misleading results and could affect overall goals of the thesis. That is why this test was done multiple times with different captures from different clients.

The captures from the above mentioned BitTorrent clients were taken and results of the tool were compared with packet capturing and analyzing tool, Wireshark. While doing this test, number of queries, responses (including errors) were matched. All other parameters like timestamps, identifiers, queries and responses data was matched by using multiple captures.

Observations

It was observed that a small number of responses were not seen in Look@MLKademlia. The reason was that those responses did not echo the same transaction ID as defined in their respective queries. Look@MLKademlia works by matching queries and responses on the basis of their transaction IDs. That is why it could not match those responses whose transaction IDs were different from their respective queries. It was observed from multiple captures that number of such responses is 1% of total responses that were received.

Although it is put in the future work, it is important to note that this limitation is not affecting the lookup analysis because from the application point of view, the responses

[20] 'NextShare', http://www.livinglab.eu/

not seen by the tool are not processed by the application. From the application point of view these responses are equal to those responses that never reached.

But from the user point of view, the absence of this small amount of responses might be misleading. User might consider these responses which were not received. That is why, it is suggested in future work to look into this problem.

5.2 Performance Testing

It was very important to do this test to find the performance bottlenecks.

RunSnakeRun

For testing the performance of Look@MLKademlia, RunSnakeRun[21] was used. It is a GUI utility for viewing python profiling dump.

Observations

RunSnakeRun revealed some performance bottlenecks. It was observed that the external library, kadtracker (Refer to Section 3.3 to know more about kadtracker) was exhibiting slow performance on large files (capture of approx 1 hour). The initial design of Look@MLKademlia was calling functions of this library many times, thus causing the tool to work slowly. To cope with this problem, design of Look@MLKademlia was significantly changed. Now it is referring to the library only once; when the tool starts. That is why Look@MLKademlia is observed slower when the file is uploaded. The rest of the time it works fine.

RunSnakeRun also revealed that IP aliasing and ID aliasing functions were making data display very slow because functions finding IP and ID aliasing were called when regular data was displayed. To solve this issue, IP and ID aliasing are not included in regular data display. Rather, it is put as an option for the user. If the user is interested to see aliasing, he/she can click on color pallets.

5.3 Comparative Analysis

A detailed comparative analysis between Look@MLKademlia and Wireshark was done to show differences between them and to highlight those features of Look@MLKademlia which make it better than Wireshark when both are used for Kademlia analysis.

5.3.1 Offline and Online Packet Capturing

Look@MLKademlia parses pcap file that is captured by Wireshark , tcpdump or any other tool available for online packet capturing.

---

[21] 'RunSnakeRun', http://www.vrplumber.com/programming/runsnakerun/

5.3.2 Bencoded Dictionaries

Kademlia DHT's messages (queries and responses) are sent as bencoded dictionaries over UDP. Each message is a dictionary having at least two keys. The two keys are 't' and 'y'. 't' is for transaction ID which is unique in each query and echoed in response so that queries and responses can be correlated. 'y' is for type of message which might be 'q' for query, 'r' for response and 'e' for error. Rest of the keys depends upon the type of message. Wireshark presents bencoded data at it is while Look@MLKademlia shows it in readable format.

To parse bencoded dictionaries into messages, the external library 'kadtracker' is used.

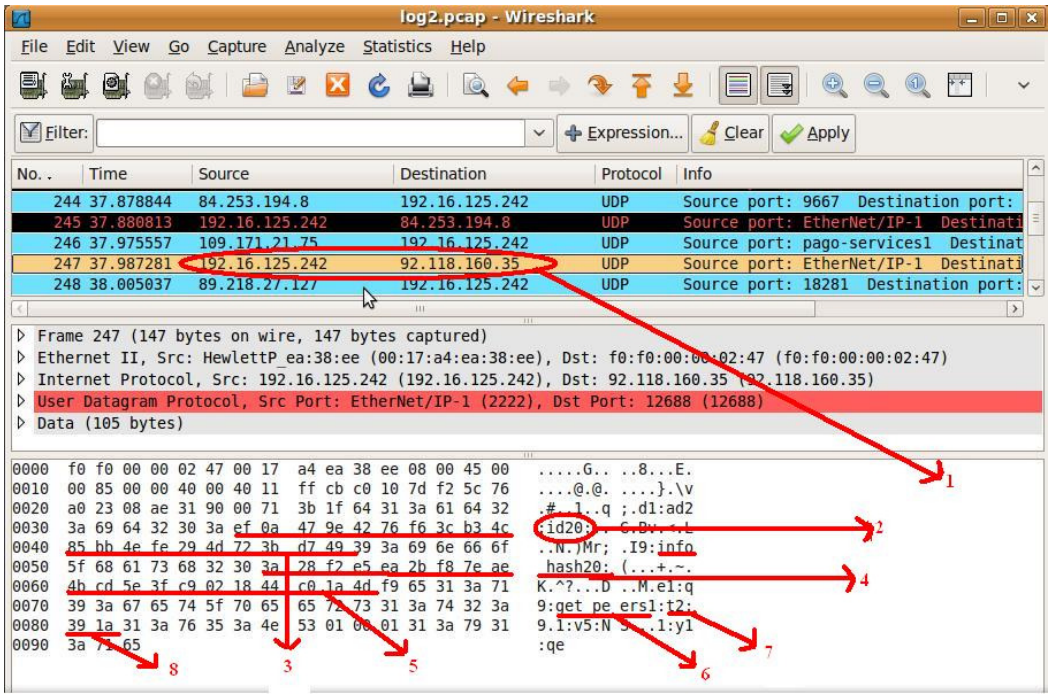
In Wireshark: In Wireshark bencoded data is displayed as it is.

In Look@MLKademlia: The bencoded data has been parsed into easily readable messages.

5.3.3 Queries and Responses Dissection

Look@MLKademlia dissects all queries and responses in the format which makes the information readily available. Here we take example of a get_peer query and its response and see how Look@MLKademlia is lot easier than Wireshark when the same information is extracted from both.
In get_peer query and response, the following information is extracted.

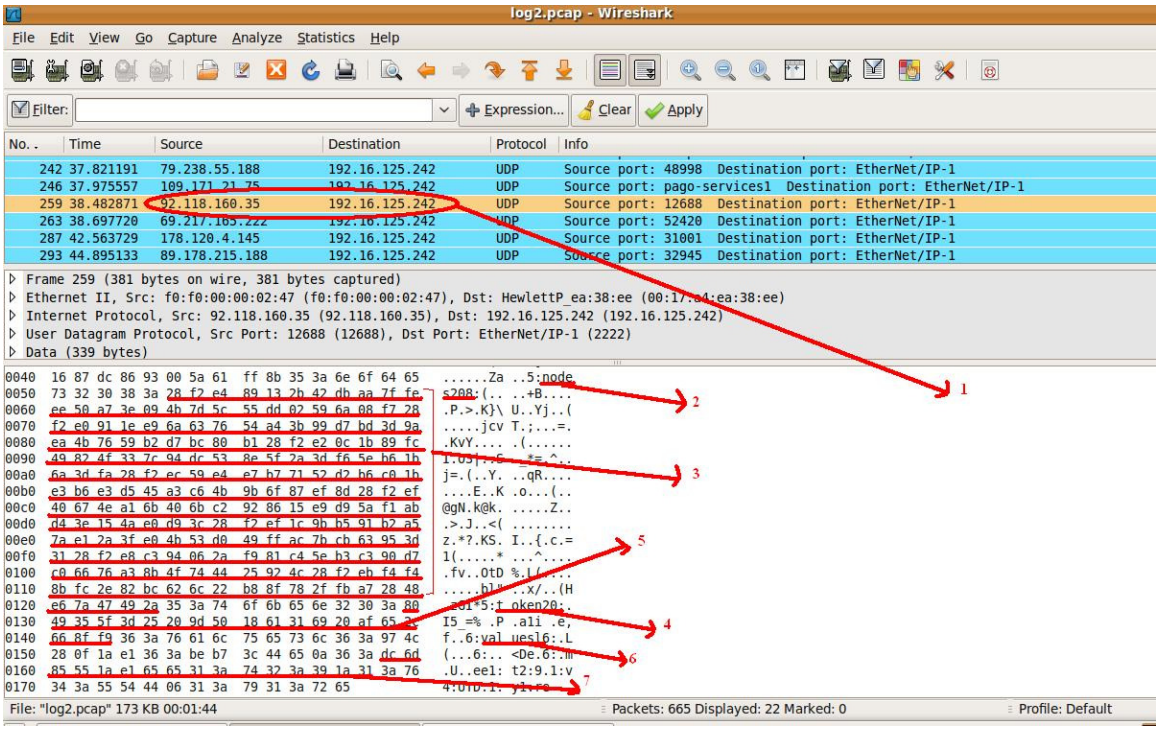
- Source and destination IP addresses
- Source and destination ports
- Infohash
- Sender and receiver node identifiers
- Nodes IP addresses
- Peers IP addresses



5.1 Get_peers Query in Wireshark

In Wireshark first we need to look at query to extract some information and then we look at response to get rest of the information. The Figure 5.1 shows information we need to get from get_peers query. '1' indicates source and destinations address. '2' shows 'id20' which means sending node identifier is 20 bytes long. To get that identifier look at hexa dump and count 20 bytes as shown by '3'. The next information is infohash indicated by '4'. For getting its value count 20 bytes in hexa dump as shown in '5'. '6' shows type of query that is 'get_peers' and '7' points to 't2' which means transaction Id which is 2 bytes long. '8' points to value of transaction Id.

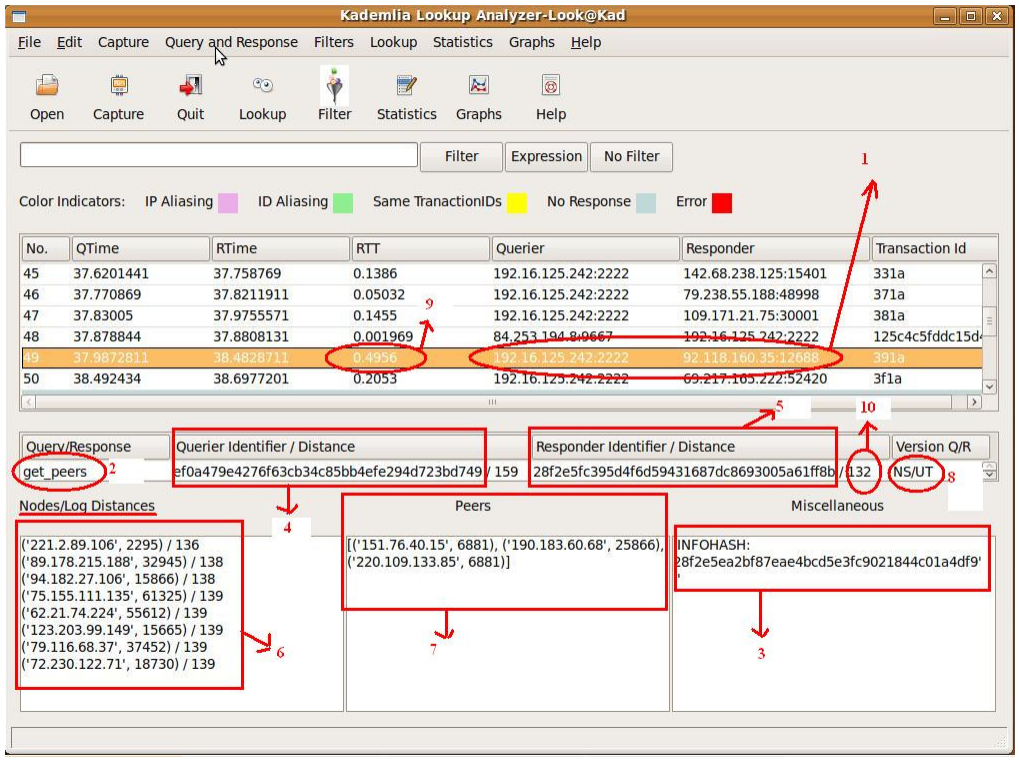
But this is half information. Now look at Figure 5.2 showing response of this query. The Figure 5.2 shows source and destination addresses pointed by '1'. '2' indicates 'nodes208' which means response has nodes having length of bytes 208. Go to hexadump and start counting 208 bytes to get values of nodes as pointed by '3'. But these hexa values needed to be converted into IP addresses and ports to get useful information.



5.2 Get_peers Response in Wireshark

‘4’ points to token having 20 bytes which are again needed to be counted from hexa dump as pointed by ‘5’. ‘6’ shows ‘values16’ which means peers. Again getting 16 bytes will not be enough. It needs to be converted into IP addresses and ports.

Now we see how Look@MLKademlia shows the same data. Figure 5.3 shows the same get_peers query and response together in one line.



5.3 Get_peers Query and Response in Look@MLKademlia

The Figure 5.3 is showing the following information.

- '1' Querier and responder IP addresses in one line long with ports.
- '2' query type which is get_peers here
- '3' infohash
- '4' Querying node identifier
- '5' Responding node identifier
- '6' Nodes in form of IP addresses and ports. Also log distances from nodes' identifiers from infohash.
- '7' Peers in form of IP addresses and ports
- '8' Version of BitTorrent clients. (querier and responder)
- '9' RTT between query time and response time
- '10' Distance of identifier from infohash.

It can be seen that in one go the whole information of query and response can be acquired. There is no need to count bytes for every piece of information. There is no need to convert nodes and peers into IP addresses and ports. Also extra information in the form of RTT and log distance can be obtained without any effort.

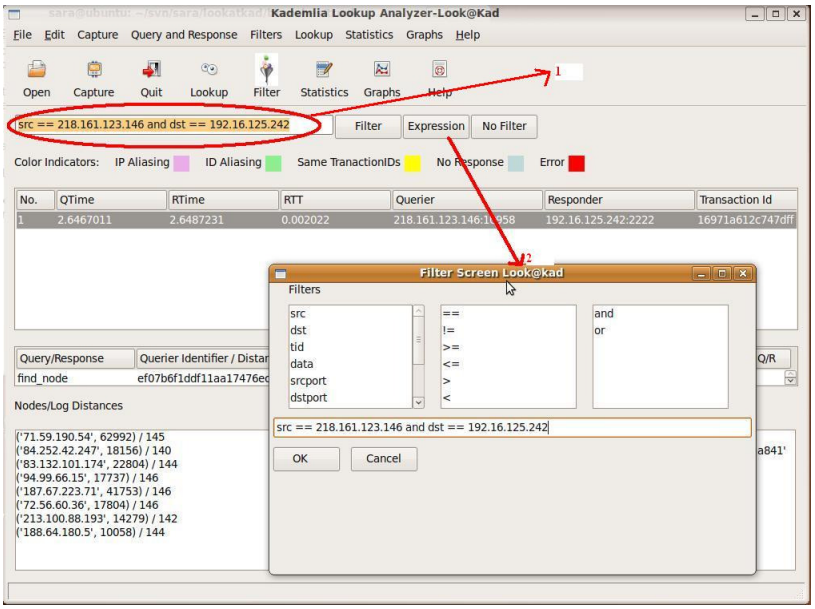
This is the case of only one query. It can be very well imagined that how difficult it would be to get information from Wireshark when several queries and their responses are needed to be checked and analyzed.

5.3.4 Filters

A variety of filters choice is a powerful feature of Look@MLKademlia. Filters can be written directly in the filter text box (Figure 5.4, '1') or filter screen can be opened by pressing 'Expression' button (Figure 5.4, arrow pointing '2'). Here comes the similarity with Wireshark. This is the same way filters in Wireshark are used with the slight differences.

In Look@MLKademlia, there are multiple parameters which can be used for filtering data such as source and destination address, transaction ID, Round Trip Time (RTT), Query and Responses time, Query type (get_peers, find_node, ping, announce_peer), destination and source port, token.

The flexibility of these filters allow several combinations of parameters using 'and' and 'or' operators. Multiple mathematical operators can be used (such as '==', '<=', '>=','<','>') to make filters more useful.

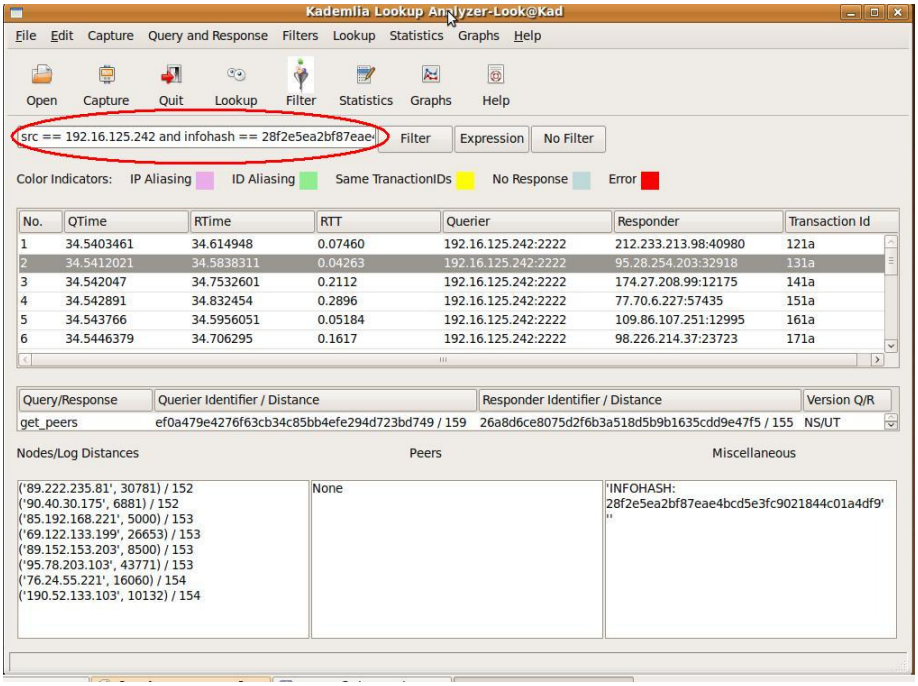


5.4 Writing Filter in Look@MLKademlia

Figure 5.4 shows two ways to give filter expression. Either write filter directly in the filter box or use 'Expression' button to open filter screen. Filter screen is helpful for learning usage of filters in the tool. Filter screen helps a user to create filter by selecting parameters and operators from the list. This is similar to Wireshark. Once 'Ok' button is pressed the filter expression is copied to the main window filter box shown in Figure 5.4 in oval marked as '1'. Pressing 'Filter' button beside the text box displays filtered data.

In Wireshark filters are very flexible and useful but when it comes to look at specifically Kademlia traffic filters in Wireshark can not be of much help. This limitation is not due to filters themselves, but it is due to the way data is presented in Wireshark- one query or response per line. For example, if one wants to see get_peer

queries and responses from a specific source and for a specific infohash, one can filter all get_peer queries but can not filter their corresponding responses at the same time. On the other hand, it is very easy to do in Look@MLKademlia as shown in Figure 5.5 where oval is showing filter expression. Writing this simple expression brings all get_peers queries and responses related to given infohash from given source.



5.5 Filter Source Address and Infohash in Look@MLKademlia

5.3.5 Lookup Behavior

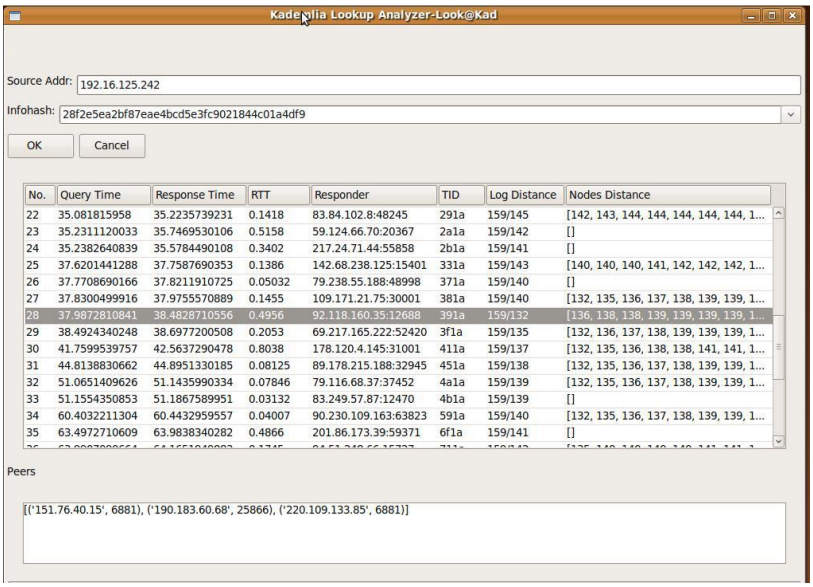
The main purpose of Look@MLKademlia is to show how lookup works in Kadmelia. Log distance of querier and responder IDs from the infohash and Nodes distances can clearly show how the lookup is converging to its destination ID i-e. infohash. Log distance is the logarithmic value of XOR distance between two node identifiers. The tool also shows Round Trip Time (RTT) to analyze lookup latency. See Figure 5.6.

In Wireshark: It is not easy to see lookup convergence in Wireshark. One has to filter the required data and collect it in some software such as Excel to do analysis. Wireshark does not show log distance of nodes from infohash which is an important metric to understand convergence process.
If Wireshark is used to extract the information shown in the Figure 5.6 taken from Look@MLKademlia, the following steps are essential.

- Collect all get_peers queries and responses from a given infohash and source address.
- Get queries and responses times and subtract them to get RTT values.
- For each response, calculate log distance of responder identifier from infohash.

- For each response, calculate log distances of the identifiers of all the nodes in the response from infohash.
- For each response, convert hexa data of nodes and peers into IP addresses and ports to get useful information.

Since Wireshark does not do it automatically, collect data in some other tool like Excel and do necessary calculations.
In Look@MLKademlia, all above steps are done on a single click. Right clicking any line gives IP addresses of nodes and geographical location of those nodes.

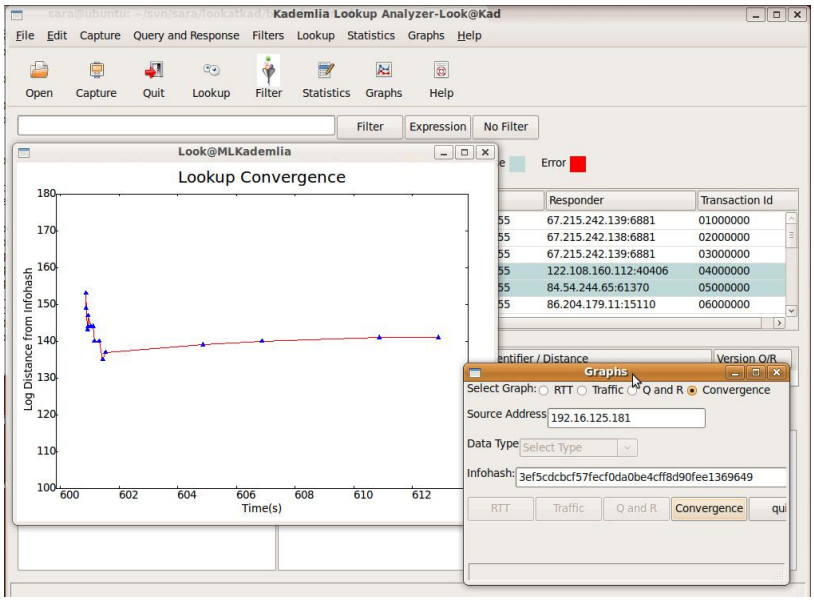


5.6 Lookup Behavior in Look@MLKademlia

5.3.6 Graphs

Graphs make analysis much quicker and easier. There are multiple graphs choices when the user clicks the 'Graphs' toolbar on the main screen.
Graphs in Wireshark: None of the graphs described below are drawn or can be drawn from Wireshark. These graphs help a lot in understanding Mainline Kademlia behavior.

Lookup Convergence Graph
Figure 5.7 shows the lookup convergence behavior. It shows how the lookup process comes closer and closer to its destination key (or infohash). Y-axis shows log distances of nodes from infohash while X-axis shows time taken to complete the lookup. We can see that when the lookup process is going to complete its log distance is relatively lower than previous values. This graph also reflects the effect of changes on lookup process due to different parameters. If some researchers or developer is changing degree of parallelism to study its effect on lookup, this graph is a quick tool to provide the required knowledge.
It can be seen that Figure 5.7 is graphical representation of Figure 5.6. (This graph has been discussed in more detail in Section 5.4.1)

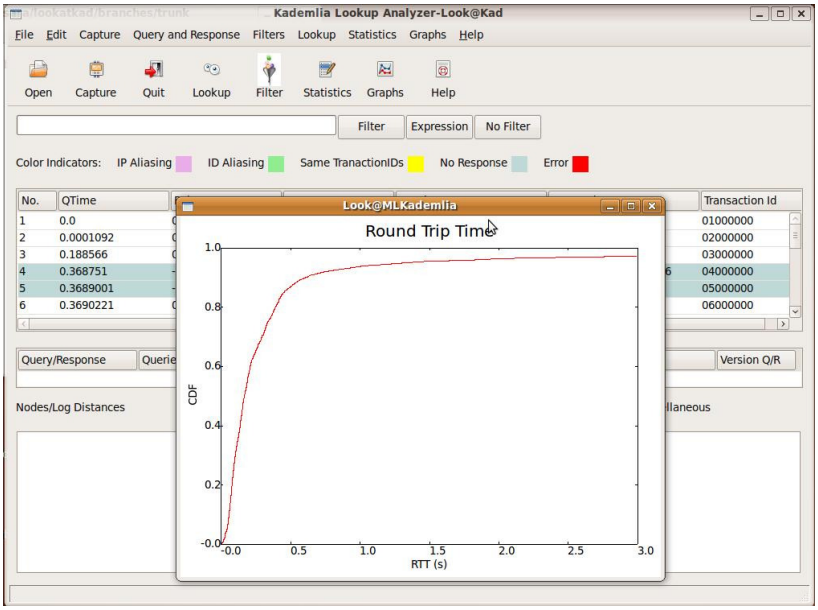


5.7 Lookup Convergence Graph in Look@MLKademlia

CDF Graph for Round Trip Time

Round trip time can be defined as time difference between sending query and receiving response. RTT defines network latency which is an important performance metric.

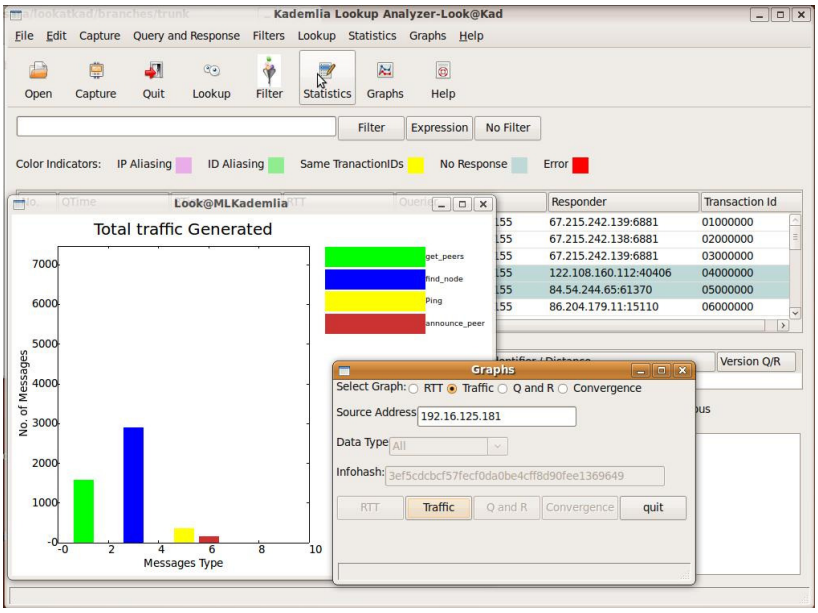
Figure 5.8 shows cumulative distribution function graph of Round Trip Time. In Look@MLKademlia multiple RTT graphs can be seen such as RTT graph for each type of query (get_peers, find_node, ping, announce_peer) and overall RTT for all queries from one source.



5.8 Round Trip Time from a Source in Look@MLKademlia

Total traffic generated graph

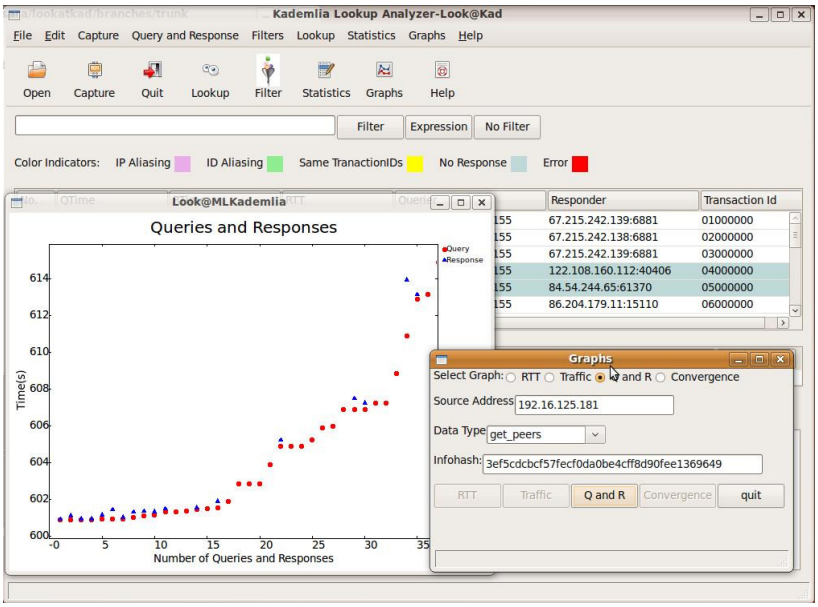
Figure 5.9 shows graph of traffic generated by a source address. It shows traffic of each query separately so that it is easy to identify that lookup traffic and maintenance traffic. Total traffic generated by a node helps to know the cost of lookup in terms of packets generated.



5.9 Total traffic generated in Look@MLKademlia

Queries and Responses Pattern in Time

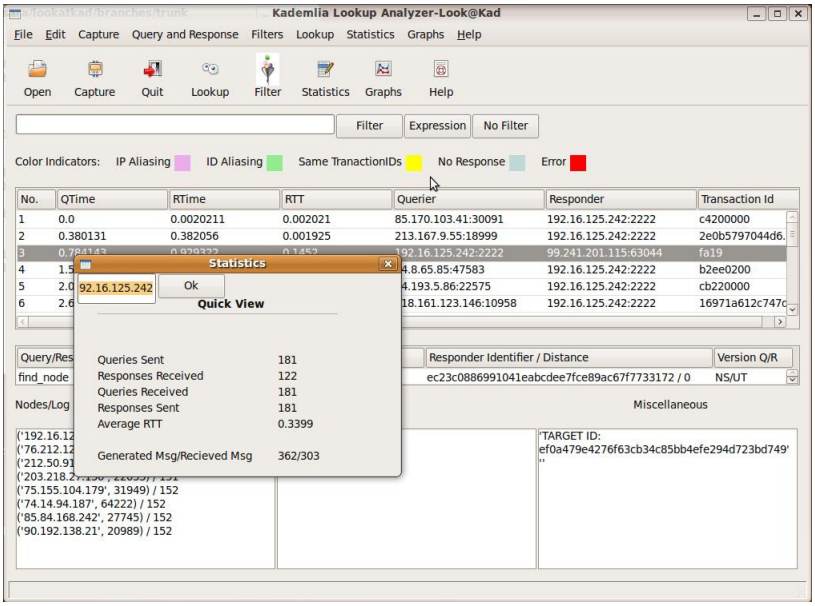
Queries and Responses pattern over time, shown in Figure 5.10, is very useful graph. This graph can be generated by each query separately or collectively. This graph can help to guess degree of parallelism, it can also help in knowing publishing behavior of the node. Moreover, it tells clearly about time when maintenance traffic is generated. (Refer to section 5.4.1 for detail discussion on this graph)



5.10 Queries and responses trend in Look@MLKademlia

5.3.7 Quick Statistical View

When any list item is right clicked on the main screen, some statistical data can be viewed. It is related to the source address in that list item. In Figure 5.11, we see that on right clicking the list item a small screen appears displaying number of queries and responses sent and received along with the mean RTT. If the user opens the window by toolbar 'Statistics', he/she will have to give source IP address.



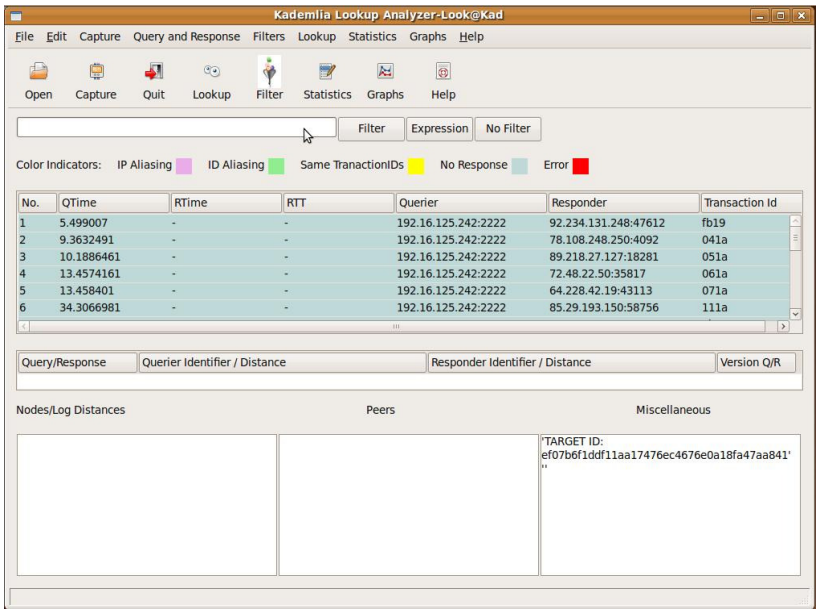
5.11 Statistical Data in Look@MLKademlia

5.3.8 Queries with no responses and Vice Versa

Look@MLKademlia can track those queries which are not responded (Figure 5.12). But right now Look@MLKademlia can not show responses without related query. (Details discussed in Section 5.2).

The first case where queries are not responded might be due to any reason such as the responding node might be dead, it might be behind firewall or may be it is overloaded due to which it is unable to respond to all queries. This might be done deliberately. For example someone is testing some implementation but does not want to get involved in the network activities.

The second case where responses are not matched to any query might be due to some attack where some attacker just sends abundant of response messages to a node to overload it. One more reason of such responses is observed. It is observed a query to destination say X with transaction ID '1' was sent. 'X' responded to that query but did not echo the same transaction ID '1'. Such type of responses are considered as they are not responded from the application point of view as the application keeps the track of queries and corresponding responses by transaction ID. (Details discussed in Section 5.2)



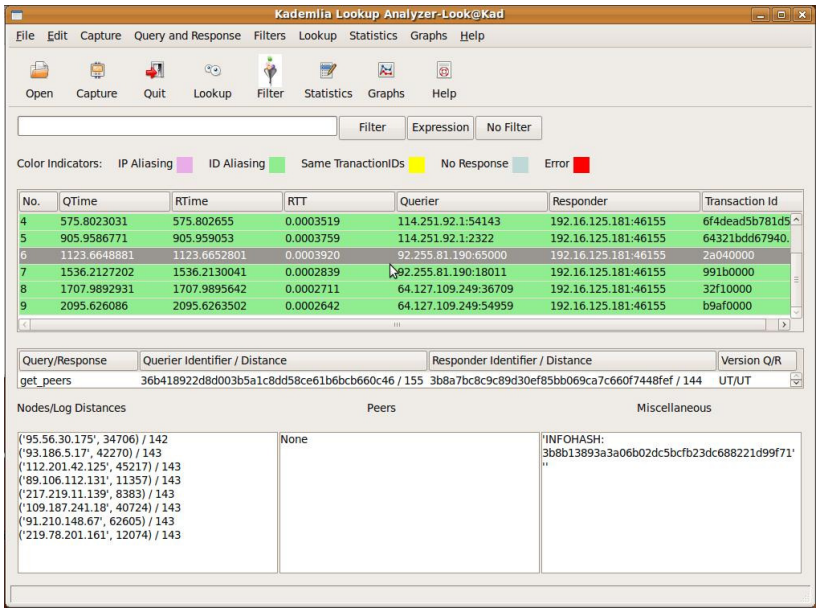
5.12 Queries with no Response in Look@MLKademlia

5.3.9 Highlighting some Known Unusual Behavior

Different colors are used to highlight the unusual behavior in packets. Two unusual behaviors are shown in Look@MLKademlia; ID aliasing and IP aliasing.

ID Aliasing

ID aliasing means peers having same IP address but different IDs. A detailed discussion on ID aliasing is given in Section 6.1.3. The obvious reason of ID aliasing might be NAT or it is done deliberately for testing purposes. Figure 5.13 is showing ID aliasing in Look@MLKademlia.

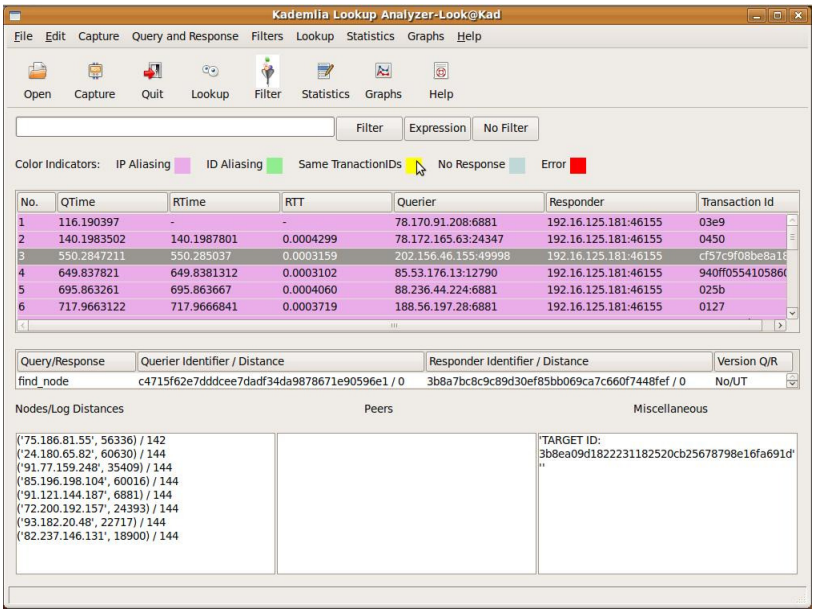


5.13 ID Aliasing in Look@MLKademlia

IP Aliasing

IP Aliasing means peers having the same ID with different IP addresses**.** A detailed discussion on IP aliasing is given in Section 6.1.3. One reason behind IP aliasing might be ISPs' policy of changing users' IP addresses every 24 hours.

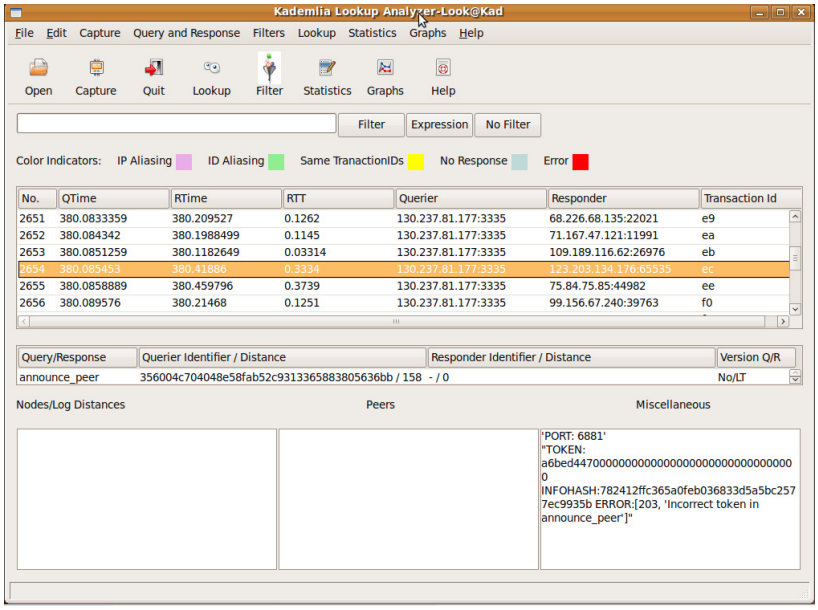
Figure 5.14 is showing IP aliasing in Look@MLKademlia.



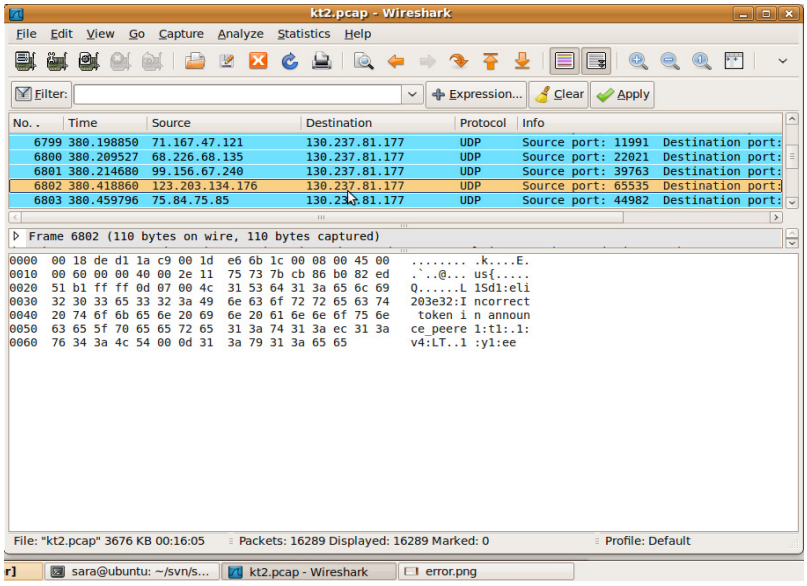
5.14 IP Aliasing Data in Look@MLKademlia

5.3.10 Errors

In Look@MLKademlia, the errors are highlighted in red. In Wireshark errors are displayed as other responses, thus difficult to be traced. It requires somewhat same amount of effort that is required for getting information about get_peers queries and responses discussed in Section 5.3.3.



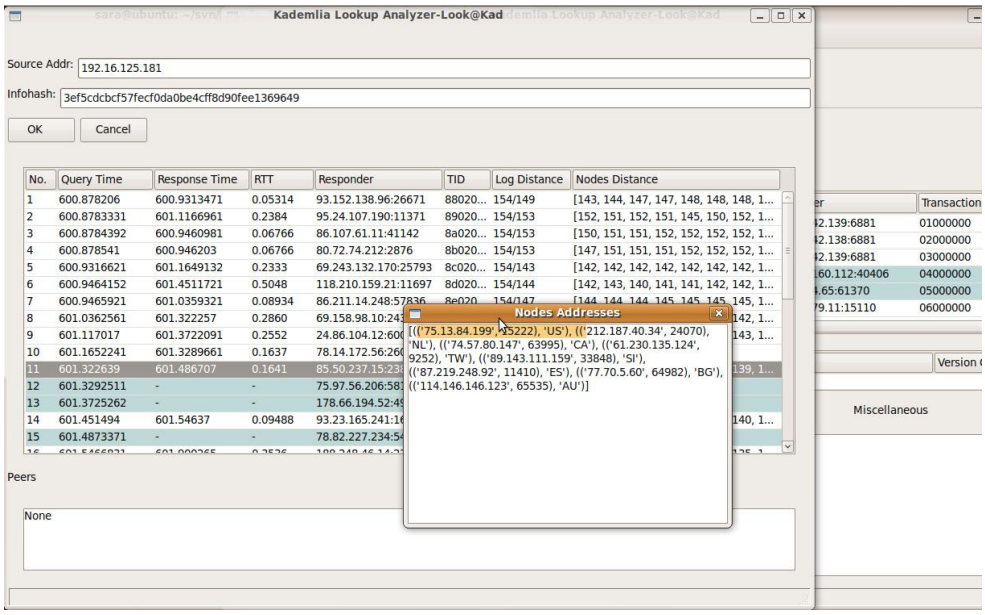
5.15 Errors in Look@MLKademlia



5.16 Errors in Wireshark

5.3.11 Geographical locations of nodes

When the lookup list item is right clicked, IP addresses of nodes are shown along with geographical location of these nodes.



5.17 Geographical Location in Look@MLKademlia

5.4 Research Capability Evaluation

This testing is done to see if Look@MLKademlia is helpful for the researchers' community. Two perspectives have been tested here. In the first, Kademlia behavior has been tested to see that how the tool helps to know how Kademlia works. In the second perspective it is seen that Look@MLKademlia can also be helpful for verification of results of experiments done on Kademlia network.

5.4.1 Kademlia Behavior

The testing has been done to see if the tool is answering the questions mentioned in the introduction section of this report. Captures of three BitTorrent clients were taken for testing. All of the captures were taken in one hour time duration. Some of the captures were also taken from my Supervisor Raul Jiménez and by my colleague Ismael Garcia.

	BitTorrent Client	**Short Name**	**Version**
1	uTorrent	UT	2.0
2	Mainline BitTorrent	ML	6.4
3	NextShare[22]	NS	

Lookup Performance

We define lookup latency as the difference between time of first get_peers query and first response of get_peers query with list of peers. Since the lookup behavior is separately shown in Look@MLKademlia, it does not take time to calculate lookup latency.

From Look@MLKademlia it was observed that 8 captures of UT showed 0.68 seconds as average lookup latency. Average lookup latency from 8 captures of ML showed 1.02 seconds. Four captures of NS showed 0.66s, 0.58s, 0.50s and 1.07s respectively (These captures had changed parameters in each).

Lookup Convergence

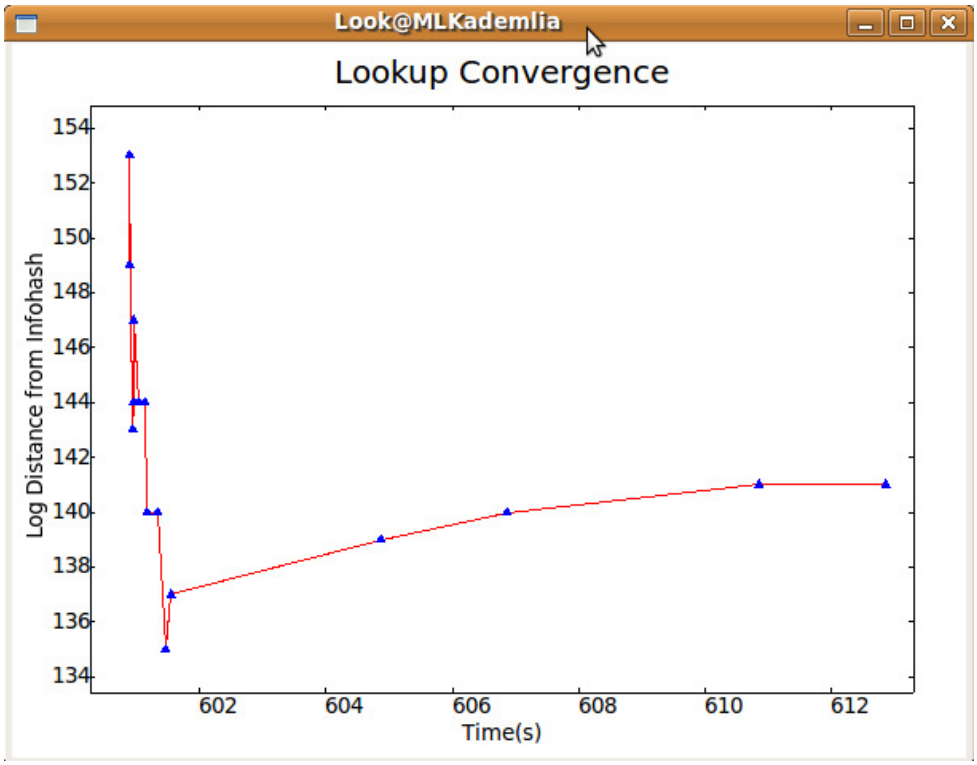
Lookup convergence is process of getting closer to the destination key when doing lookup. The convergence behavior should show that the distance towards the destination key is becoming smaller with time. The following graphs show four convergence behaviors from four MDHT enabled BitTorrent clients.

From the following graphs, it can be seen clearly that every client has its different way of converging towards its destination key (Infohash). These differences might be due to different parameter values such as timeout and degree of parallelism (Known as α). Using

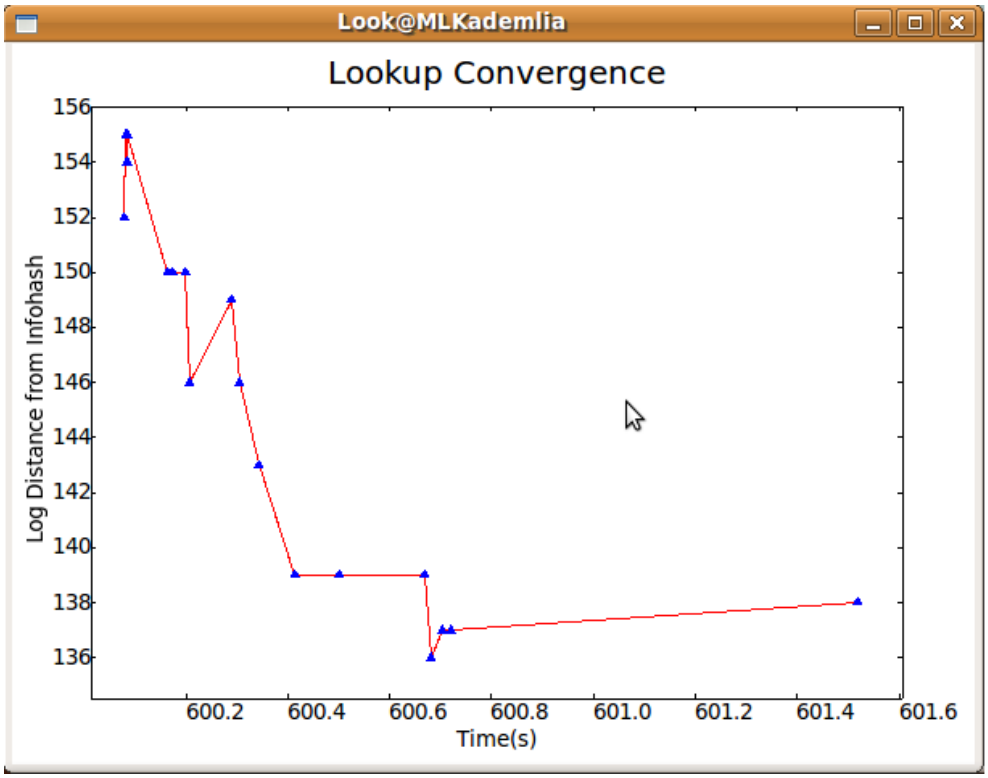
[22] 'NextShare', http://www.livinglab.eu/

Look@MLKademlia a guess of degree of parallelism is made according to which UT and ML have α=4 while it is 8 in NS and KTorrent.

In Figures 5.19 and 5.18, the y-axis is log distance from infohash while x-axis is time. It can be seen in x-axis that time does not start from '0'.Rather than scaling it from 0 it was decided that the real time when the lookup starts would be used as starting point.

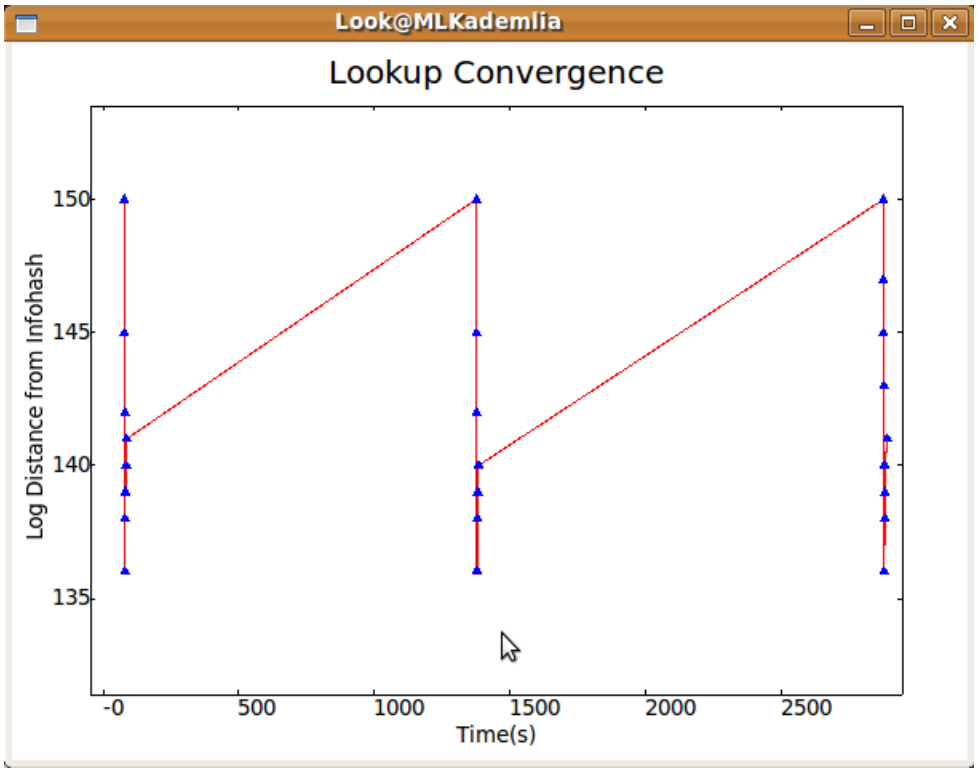


5.18 uTorrent Lookup Convergence



5.19 NextShare Lookup Convergence

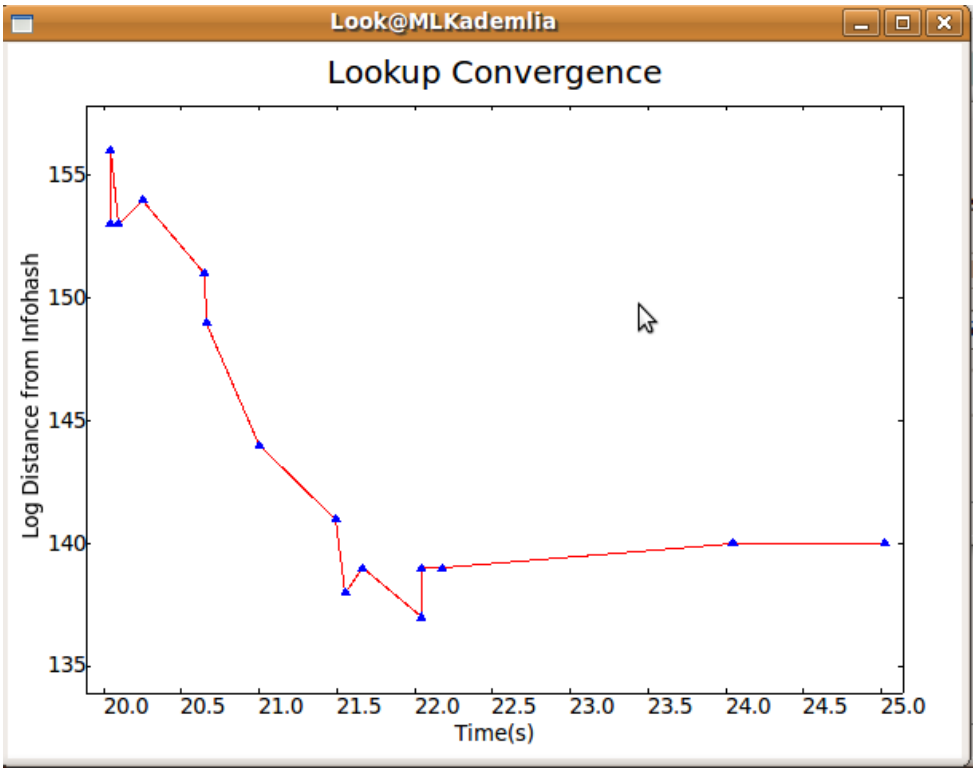
Look@MLKademlia gets all of the lookups for one infohash. This is why Figure 5.20 is showing strange graphs. Figure 5.21 is zoomed image of first part of Figure 5.20 (graph of Mainline BitTorrent). This is the single lookup for the infohash. The other two lookups are done when the node wants to send announce_peer query. Before sending announce_peer query, the node needs to know some closet nodes. These lookups are done after every 25 minutes to know the closest nodes and then announcement are made about the infohash to those nodes.



5.20 Mainline BitTorrent Convergence

Figure 5.21 is zoom Image of first part of Figure 5.20. By first part, we mean first lookup done in one hour. Figure 5.21 can be obtained in two ways.

1- Zoom the Figure 5.20 while using Look@MLKademlia.
2- Truncate the one hour capture to 20 minutes capture with the help of Wireshark. Then put 20 minutes file to Look@MLKademlia. In this way, we will see graph of single infohash.



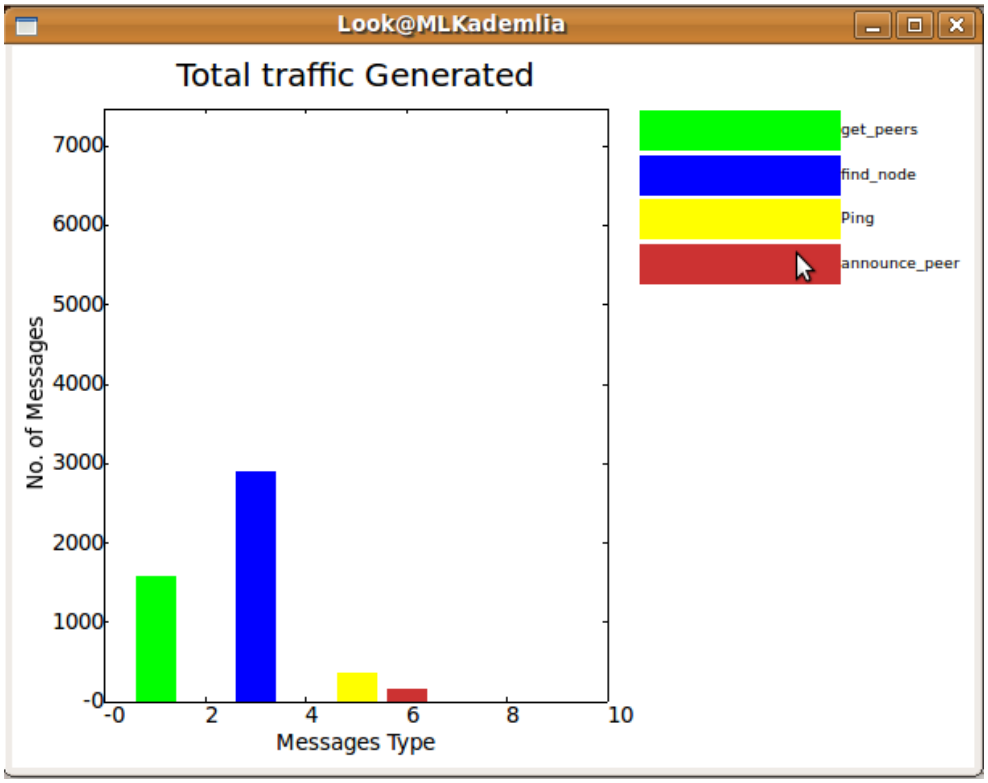
5.21 Zoom Image of Figure 5.20 (Lookup convergence in Mainline BitTorrent)

Effect of Parallelism on Lookup

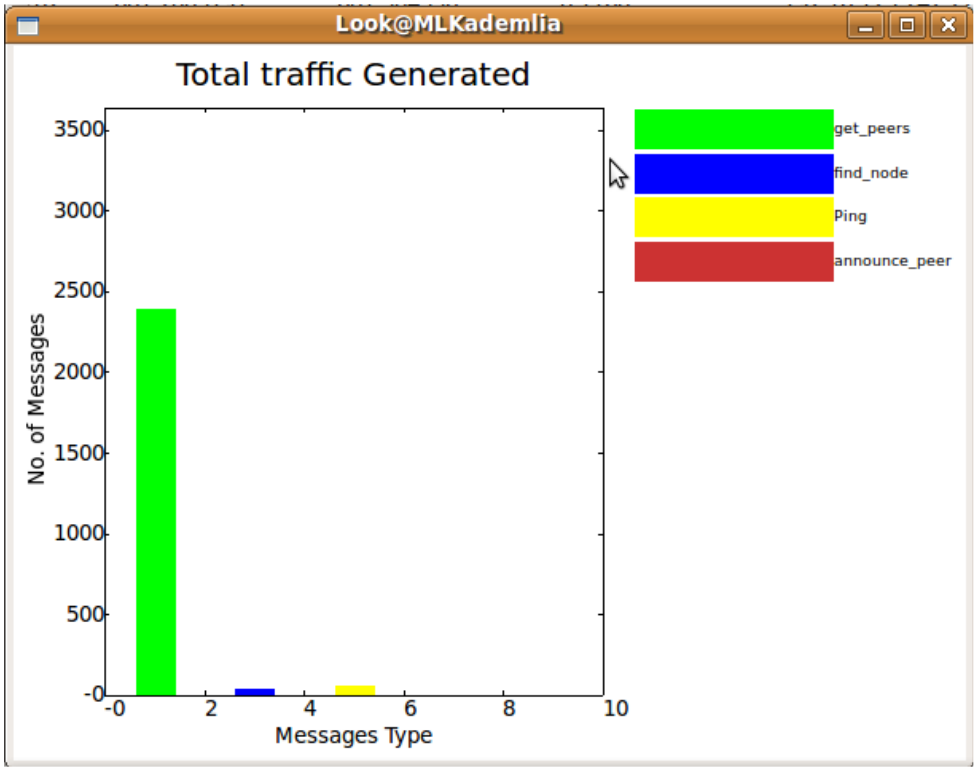
The discussion on above questions clearly state that lookup process can be observed quickly in Look@MLKademlia. The above graphs can also show effect of parallelism on lookup. If some one is working on Kademlia implementation and wants to observe the affect on lookup while changing different parameters, Look@MLKademlia can help in this regard.

Lookup Cost in Terms of Traffic Generated

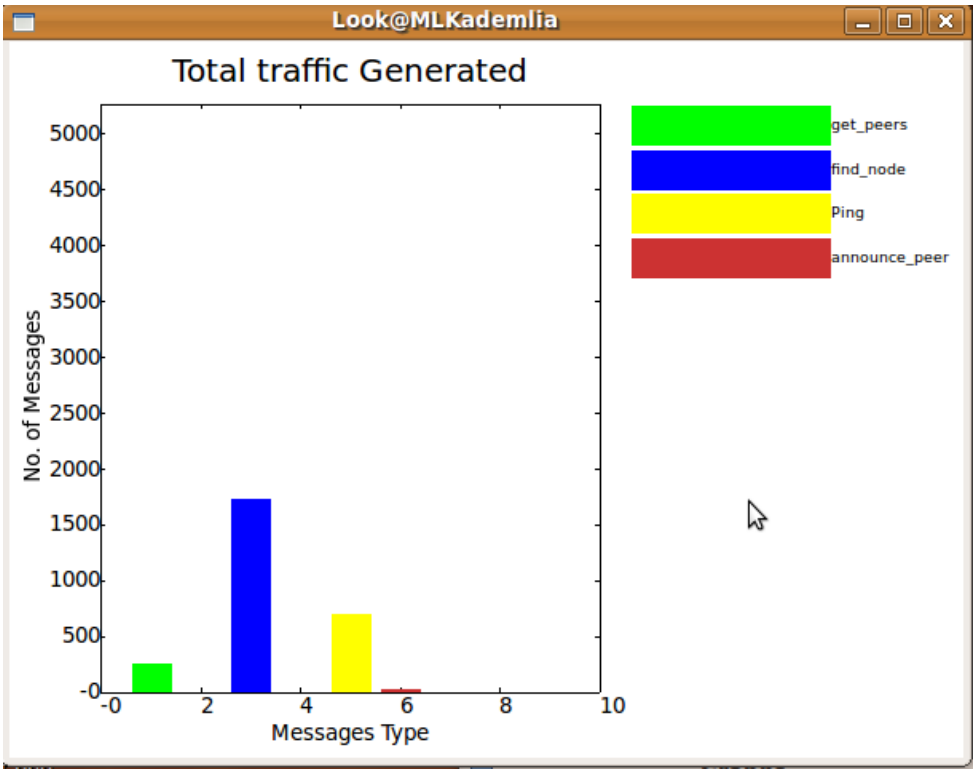
Look@MLKademlia shows number of get_peers, ping, find_node and announce_peer queries separately. This helps the user to know how much lookup and maintenance and publish traffic is generated.



5.22 uTorrent Generated Traffic



5.23 NextShare Generated Traffic



5.24 Mainline BitTorrent Generated Traffic

A consistent observation about ML and UT is that their find_node queries are significantly higher as compared to other queries. But this is not observed in NS (Figure 5.23). It is observed for NS that the policy of this implementation is to keep the maintenance traffic less.

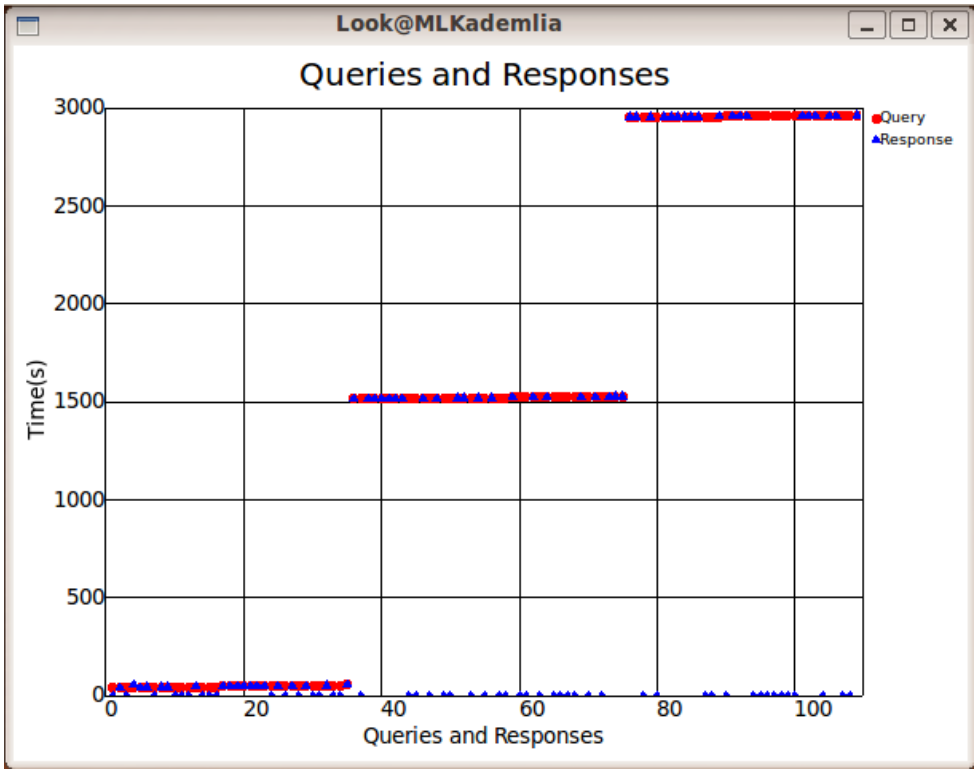
Traffic Pattern

Pattern of queries and responses shown in Look@MLKademlia, provides the user information about how the queries and response are thrown. It can help user to make a good guess about some of the parameters like timeout value and degree of parallelism. For example, these patterns show that UT and ML have α=4 while it is 8 in NS and KTorrent. The following figures show patterns of queries and responses from three clients. These are only get_peers queries and responses with the same infohash.

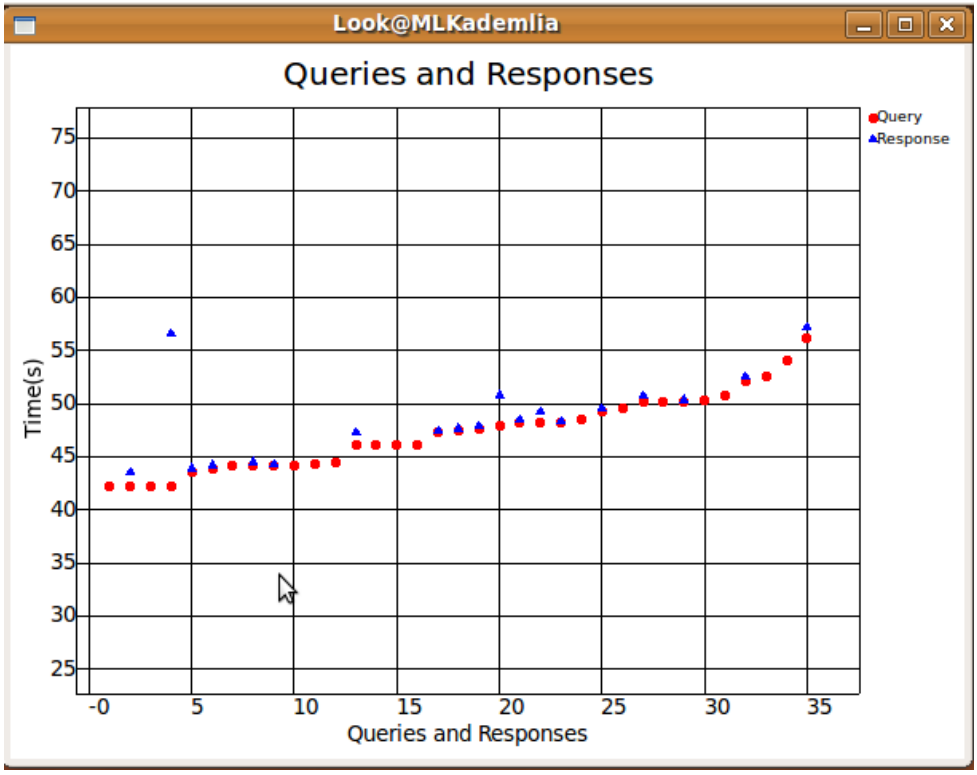
The y-axis in the Figures 5.31 and 5.32 starts the time when first query is thrown rather than starting it from 0.

Figure 5.25 and Figure 5.26 show queries and responses in Mainline BitTorrent. Figure 5.26 is zoom image of the lowest part of Figure 5.25. We can see that the number of queries thrown first in parallel is 4.

Look@MLKademlia shows all lookups related to one infohash. If some one wants to see the single lookup, he/she can zoom that area of the graph as is done in Figure 5.26. Figure 5.26 is zoom image of first lookup in Figure 5.25.

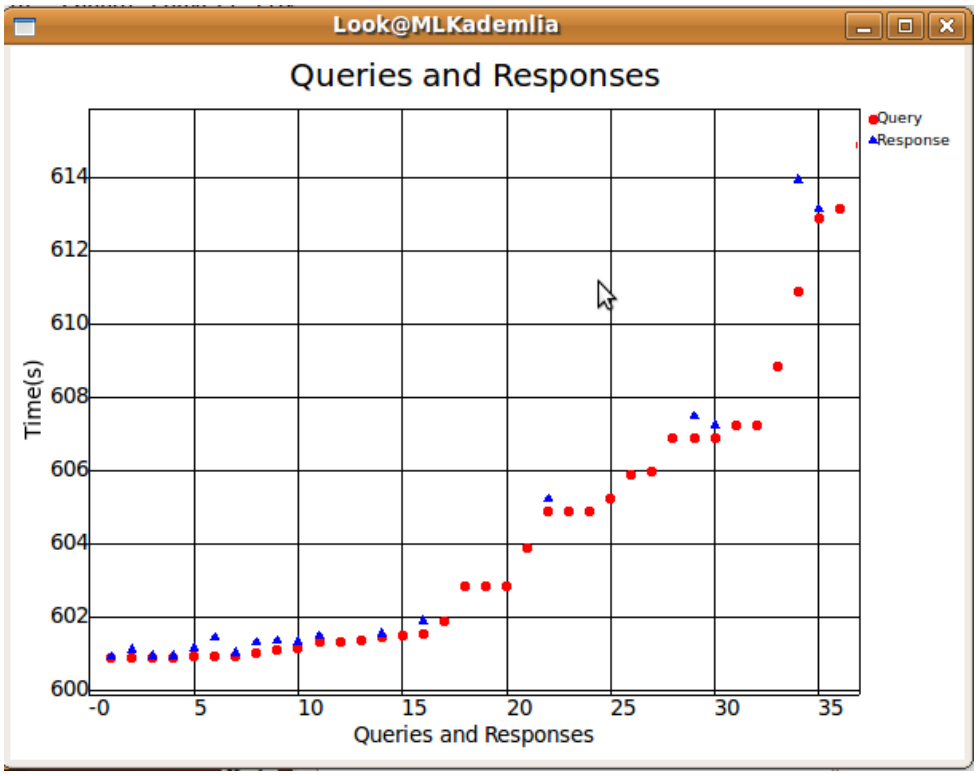


5.25 Queries and Responses in Mainline BitTorrent

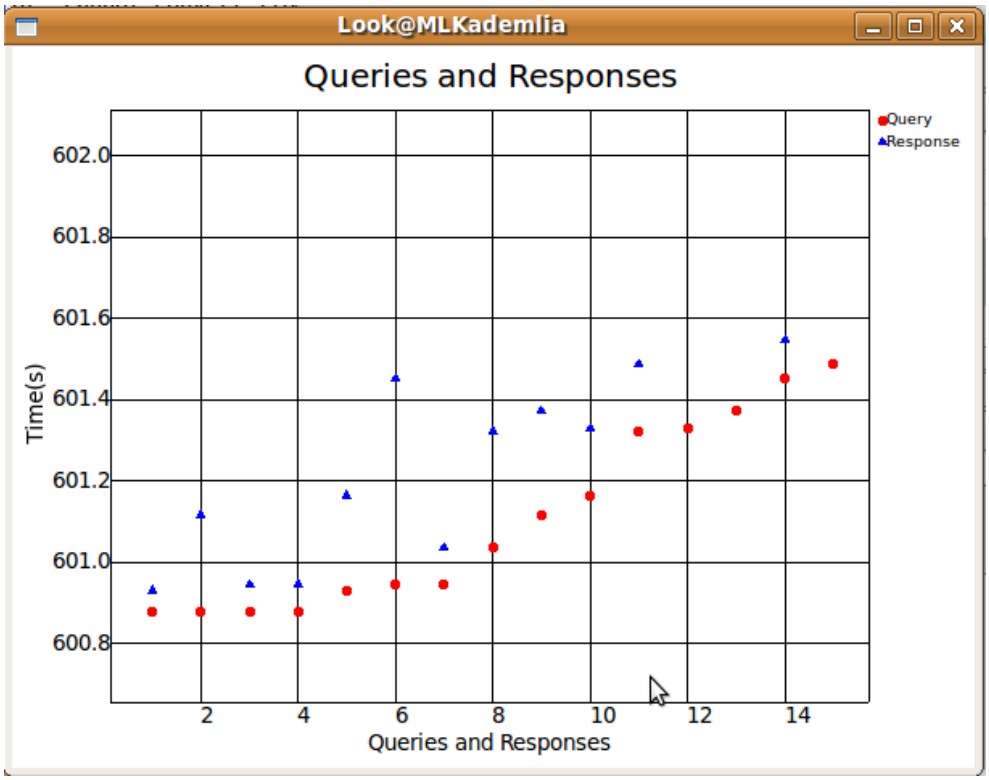


5.26 Zoom Image of lowest part of Figure 5.25 (Queries and Responses in Mainline BitTorrent)

In figure 5.27 queries and responses from UT are shown. The zoom image of Figure 5.27 is shown in 5.28. This zoom image gives us the idea that degree of parallelism is 4.

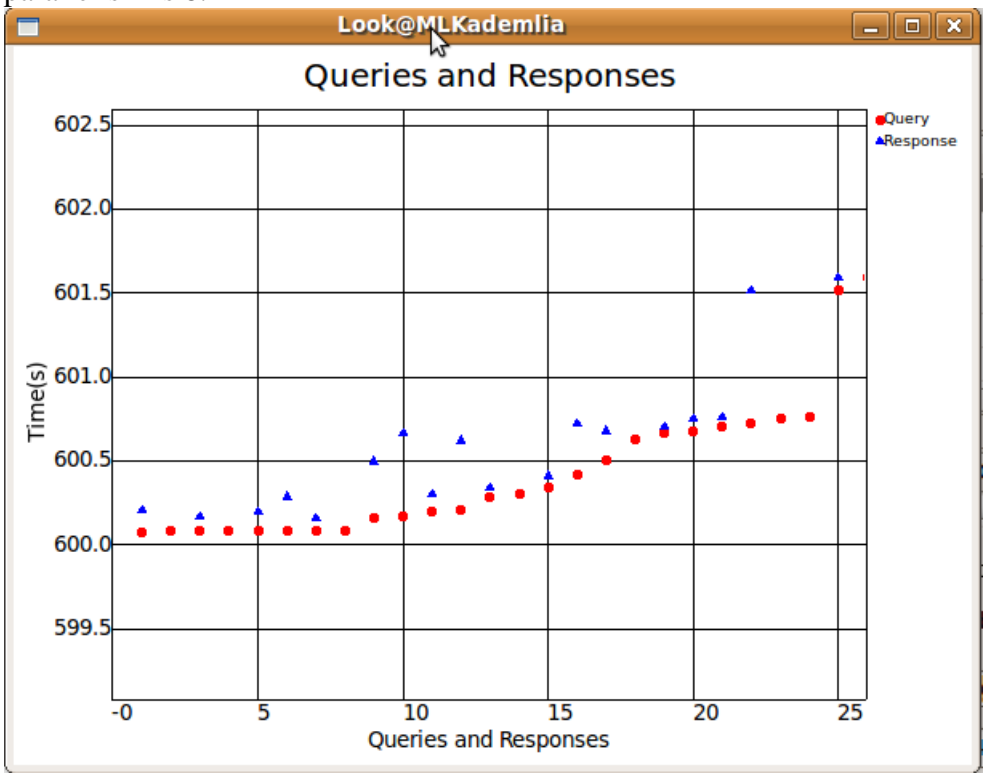


5.27 Queries and Responses in uTorrent



5.28 Zoom Image of 5.27 (Queries and Responses in uTorrent)

Figure 5.29 is showing results from NextShare. Here we see that degree of parallelism is 8.



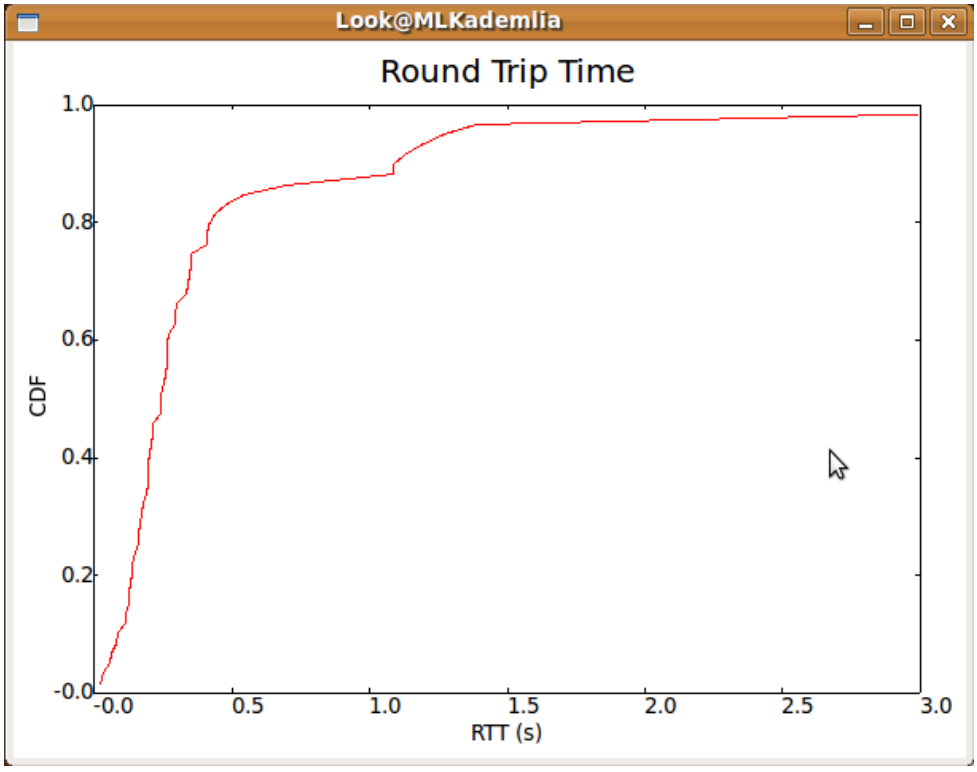
5.29 Queries and Responses in NextShare

Publishing Behavior

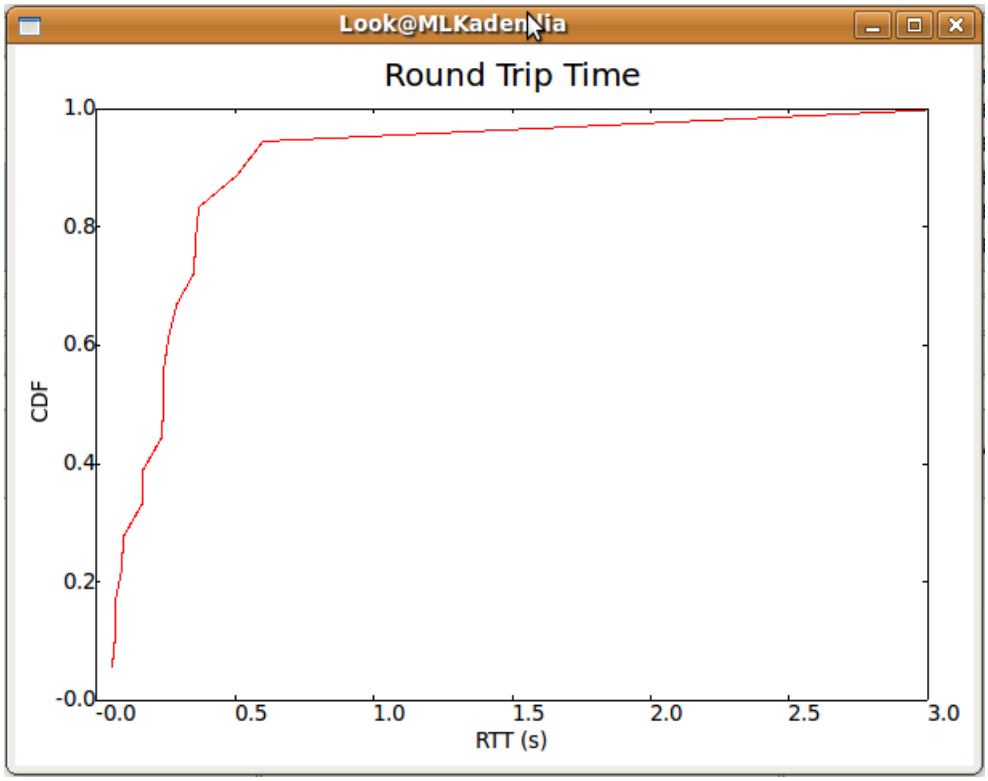
In the above graphs, it was discussed that the degree of parallelism can be observed. In Figure 5.25, we observe three lookups, each after 25 minutes. This shows the publishing behavior of node. After every 25 minutes, the node looks for closest nodes where it can announce the infohash. Announcement is done by sending announce_peer query.

Round Trip Time

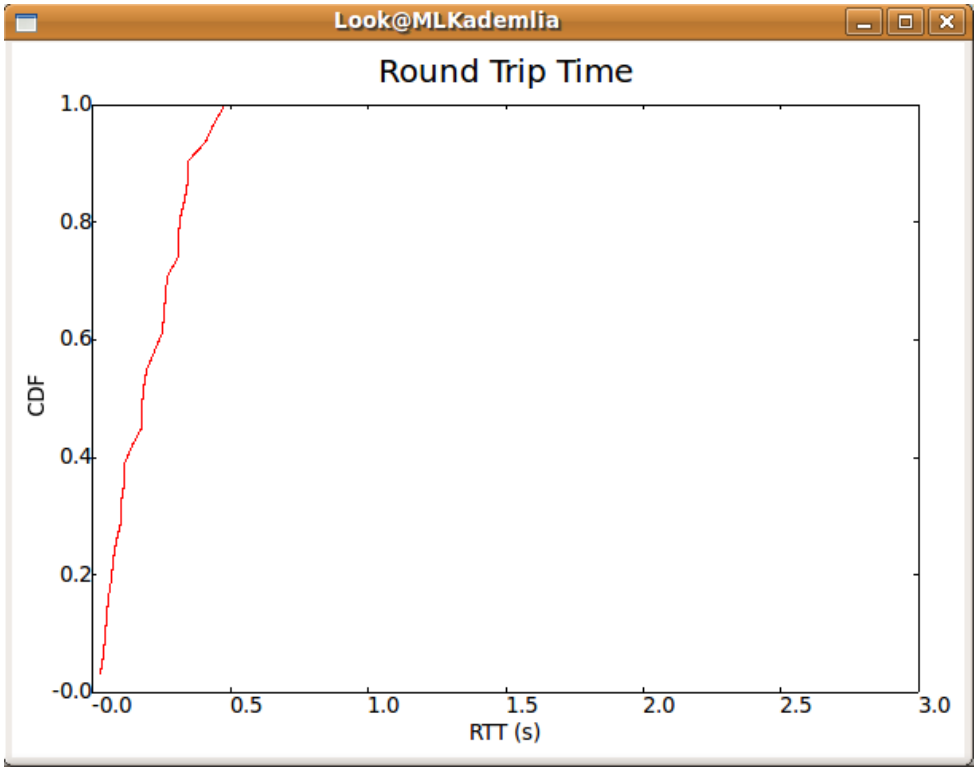
Look@MLKademlia provides multiple graphs about RTT. The user can see RTT of all of the queries together and can also see RTT for each of the query type separately. Figures 5.30, 5.31 and 5.32 are showing RTT for get_peers queries for ML, UT and NS respectively.



5.30 Round Trip Time of get_peers Queries in Mainline BitTorrent



5.31 Round Trip Time of get_peers Queries in uTorrent



5.32 Round Trip Time of get_peers Queries in NextShare

5.4.2 Corroborating Tool for Results of Experiments

Although first goal of Look@MLKademlia is to show lookup behavior of Kademlia, it has other broader benefits; it can verify the results of experiments done on Mainline DHT. In TSLab[23], KTH a thesis presented by Ismael Garcia with title "Censorship in BitTorrent"[24]. He targeted those BitTorrent clients who were MDHT enabled. He developed some scripts to do his experiments. In the mean time Look@MLKademlia was ready to use. He used this tool to verify his results.

His statement about Look@MLKademlia is:

"In my thesis, the possibility of censorship was studied. According to the design of the DHT, when a node sends an announce_peer message, it sends it to the three closest nodes to the corresponding info hash. In this case, if a client was modified to choose its identifier and three nodes were positioned closer to the info hash than those containing the list of peers, in a short period of time these nodes would be the only nodes containing the list of peers. If these nodes were modified not to provide a list of peers (or to provide a fake list of peers), the access to the object would be censored. It was checked experimentally that this way of censorship did not work; the reason was searched in some of the Mainline BitTorrent clients. The goal was analyzing the distance to the info hash of those nodes where different clients choose to announce. In some cases,

[23] 'TSLab, KTH', http://tslab.ssvl.kth.se/
[24] 'Content Censorship in BitTorrent', http://tslab.ssvl.kth.se/thesis/node/1273

it was found out that they don't follow the design rules. For example, KTorrent doesn't announce in the closest nodes (because, for example, sometimes it chooses nodes of distance greater than 150 to the info hash for the announcements and it is almost impossible that these nodes are part of the closest), it was also checked that some clients announce in more than 3 nodes. The tool **Look@MLKademlia** was useful to find out this behavior by watching the distances of nodes where a client chooses to announce and the number of nodes.”

His general statement about Look@MLKademlia is:

“This report is written from the point of view of a user of Look@MLKademlia who had a previous knowledge of how the DHT of BitTorrent works. As I am doing my master thesis about a related topic I could start using it immediately.

My first impression of the tool was how easy it was to use, I did not even need to read any user manual to find out how it worked, this is a big advantage because it is easy to get results fast. As the interface is very similar to Wireshark, for users who have used this program before, Look@MLKademlia is very easy to use.

The usage of this tool is clear; it provides a traffic analyzer for the DHT of BitTorrent. When I worked on my master thesis, sometimes I wanted to look at some specific data of the traffic generated; the choices I had were either doing a program for it (which could take more too much time) or using Wireshark to find it. Using Wireshark was not easy because it could take a long to find the wished data. Looking at the data in a message was quite tricky but the hardest was matching queries and responses. In the case I wanted to find some data, I usually wanted to calculate the some data with it (like log distance), Look@MLKademlia avoids the necessity of calculating this data.

As at the beginning I could not use the tool (because it was being developed), I made some script to find the information I wanted in Wireshark captures, once the tool was ready I could compare the results my tool provided with those provided by Look@MLKademlia, it was very useful to debug and check the correct behavior of my programs.

I think the most interesting property of Look@MLKademlia is its generalness, this means that, as every BitTorrent client has to follow the message protocol, any captured traffic from any BitTorrent client with MDHT enabled, can be analyzed with Look@MLKademlia.

In my work, I had to analyze the behavior of different BitTorrent clients, I could have looked at the source code of some of them, but it would have taken too much time. With Look@MLKademlia it had been very easy to process the captured traffic and watch the behavior I wanted.”

5.5 Discussion

The testing and evaluation of Look@MLKademlia has exhibited several features of this tool. The first and foremost feature is its capability to highlight those Kademlia aspects on which active research is going on. They include lookup latency, lookup convergence, lookup cost, traffic patterns, degree of parallelism, round trip time, geographical distribution of nodes, IP and ID aliasing, incoming and outgoing traffic. These are some of the major key points about Kademlia behavior on a single node.

Besides, Look@MLKademlia is helpful in confirming the results of experiments done on Kademlia node. For example, in his thesis, Ismael Garcia, a student in TSLab, used Look@MLKademlia to see how different BitTorrent Clients using MDHT, are announcing infohash to other nodes. From his experiments he observed that the announcing behavior of some implementations is different from the defined standard. He confirmed his results with the help of Look@MLKademlia.

Look@MLKademlia is helpful if some closed implementation is tested to know its key parameters including its lookup behavior, its degree of parallelism, its traffic patterns and some of its policies. As exhibited in this evaluation it was observed that one of the client' policy was to keep management traffic as small as possible. It was also seen that degree of parallelism was different in different clients which was expected to be 8 in each of them. Similarly, in the above paragraph, it was found by the student that announcement of infohash does not follow the standard in some clients.

One key point that is needed to be insisted upon is that the Look@MLKademlia should not be seen bounded to the evaluation done in this thesis. Since Look@MLKademlia is an exploratory tool which is presenting traffic in a friendly format, its more capabilities can be found if it is tested in various scenarios. Although it is not checked, but due to Look@MLKademlia features, it is believed that if it is placed near some popular infohash, it can bring out useful information like searching and publishing behavior. Moreover, its statistical data can detect unusually high traffic which might be due to denial of service attack on that node.

Chapter 6

Related Work

The chapter discusses the related work in two aspects. In the first section it discusses some other DHTs designs besides Kademlia. In the second section, the related work focuses on the tools that have been developed to monitor measure and analyze various Kademlia implementations.

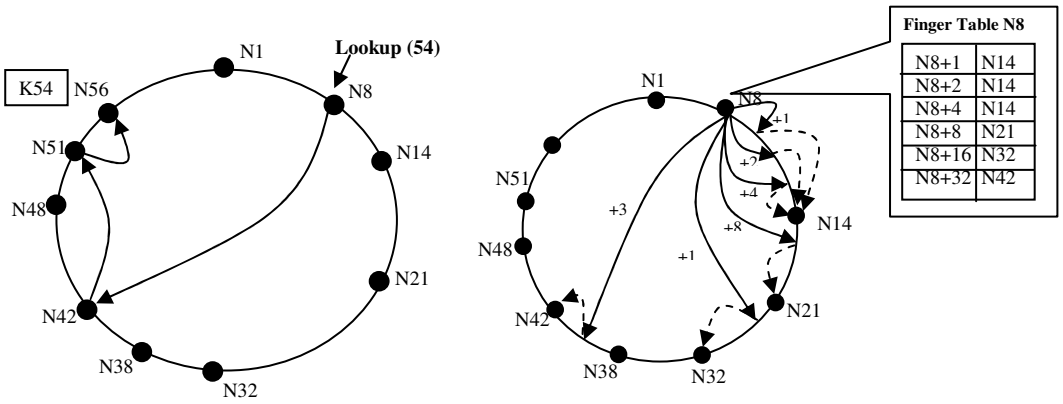
6.1 Existing DHTs

Some of the most popular DHTs which caught the attention of the researchers and have seen the real world deployments are Kademlia [1], Chord [13], Pastry [14], and Tapestry [15].

6.1.1 Chord

Chord [12], like Kademlia is one of the popular DHT protocol over the Internet. But it is quite different from Kademlia. Unlike Kademlia, chord sends one query at a time. One more difference is seen in the metric. In Kademlia metric is symmetric. If the nodes are arranged in a virtual ring, metric in case of Kademlia will remain the same either seen clockwise or anticlockwise while in case of chord metric is asymmetric. It works only in one direction.

In chord the node identifiers and keys are arranged on the same circle known as identifier circle or the chord ring. The node that has higher identifier in the ring is successor of its previous node or key and predecessor of its next node or key. Each node keeps track of its successor and predecessor which becomes more important and challenging as well where network faces significant churn. For stable behavior chord node needs to keep its successor and predecessor pointers updated along with it finger table. Finger table is the routing table in chord having node identifiers along with nodes' IP addresses and port numbers.
In an N-node System Chord maintains information about O(logN) nodes and a node requires O(logN) messages to complete its lookup.



6.1 Chord Lookup Process and Finger Table

Reproduced from [12]

6.1.2 Pastry

Pastry is another popular DHT like chord and Kademlia.
In pastry node identifier is 128-bit randomly chosen node Id. Node Ids are evenly distributed in the circular namespace ranges from 0 to 2^{128} -1.

In Pastry a node contains three tables - leaf set, routing table and neighborhood set. If L is the size of leaf set, leaf set of a node A contains L/2 smaller and L/2 larger nodes than A. L is generally 16 or 32. Leaf set is the first place to check when lookup process for key K starts. If key is not found within the range of the leaf set, routing table is contacted. Routing table starts with matching prefixes like Kademlia. The lookup message is forwarded to the node whose id is one digit closer to the key K than the current nodeId. If no such node is found, then message is forwarded to the node whose number of matching digits is exactly equal to that of current nodeId's matching digits but numerically closer to key K than the current nodeId. Routing of messages take O(logN) steps to destination where N is number of nodes in the system.

Neighborhood table contains closest nodes in terms of proximity metric. It is not used in routing, but used for locality properties. The purpose of proximity metric is to minimize the distance messages travel. Proximity metric might be number of hops, geographical location etc.

6.1.3 Kademlia

In section 2.5 Kademlia working has been described in detail. Here we describe how the researcher community has seen Kademlia.
Kademlia has been widely studied and analyzed in several aspects using various methods. Some people have studied and analyzed it in real-life setup while some examined it in simulated environment. Similarly, some have monitored Kademlia traffic passively while some have done it actively by using crawlers.

Using various tools and methodologies, Kademlia is studied is many perspectives such as finding session time and timeout value, analyzing lookup performance, measuring lookup cost, latency and parallelism. Finding effects of redundancy, replications and dynamics of peers or churn.

Lookup Latency and Routing Table Efficiency

Crosby and Wallach [3] have discussed two Kademlia based DHTs; Mainline DHT (MDHT) used in BitTorrent and Azureus DHT(ADHT) used in Azureus. They have examined both DHTs in perspective of churn, reachability, latency and liveness of nodes and they have identified various problems and their possible solution. They have identified several bugs in implementations of both DHTs. Their work emphasize on the problem of lookup latency due to timeouts. They believe that lookup time can be improved to median of 5 to15 seconds for MDHT and ADHT, respectively by decreasing timeout value. Further, they discussed the causes and locations of dead nodes in the routing table. According to their study, they found that due to the bugs in implementations of the DHTs, the lower layers of routing tables have significant number of dead nodes. They suggest aggressive pinging as solution to this problem. Moreover, they believe that along with refreshing routing tables effectively and increasing concurrency can improve lookup latency; improving timeouts can also be one of major strategy to hide latency. To elaborate it more, they suggested the idea of finding timeout dynamically rather than using a static value. To sum-up, the researchers have discussed in detail about how different parameters can affect overall lookup process and how improvements can be made in parameters and in the routing table refreshment policies to make the lookup process better and faster.

Churn

Steiner et al. [5] have studied Kademlia in the perspective of churn. They have analyzed nodes sessions times in KAD which is Kademlia based implementation in eMule and eDonkey. They see the behavior of nodes and discuss the magnitude of long lived and short lived nodes. They have discussed in detail about the session and inter-session times of peers. For achieving their goals, they have actively analyzed the network by using their own crawler named Blizzard. The practical implication of their work is shown at the end of their paper where they emphasize on selecting peers carefully on the basis of session and inter-session times of peers for storing information reliably in KAD, which as a result make KAD more stable.

IP and ID Aliasing

ID aliasing means peer with the same IP address but different IDs. Steiner et al. [6] have again actively monitored the KAD using the same crawler ‘Blizzard’ as used in [5]. According to them large number of peers in KAD (KAD is Kademlia implementation in eMule and eDoneky) have been observed whose IP addresses are same with different IDs. 1) Peers having same IP but carefully chosen IDs. They state that pattern of IDs can tell us that they are chosen by hand. They believe that such Peers might belong to the same organization due to the way their IDs are chosen. 2) Peers with same IP address and

randomly chosen IDs. These peers might be the clients behind NAT sharing same IP address.
In another paper Steiner et al. [7] discuss more about KAD ID aliasing. According to their observations, a significant number of users in some geographical regions keep on changing their IDs, thus making KAD IDs non-persistent, which according to them make it difficult to do some measurements such as membership turnover and individual behavior of peers. But they could not find the reason of this behavior of changing KAD IDs while having the same IP address

IP aliasing in peers means peers with same ID but different IP addresses. One reason seen by Steiner et al. [7] is that some ISPs give new IP addresses to the users after every 24 hours. They believe that number of IP addresses per peer is strongly associated with the life time of that peer.

Bhagwan et al. [28] have discussed IP aliasing problem in context of hosts' availability. They argue that counting number of hosts on the basis of IP addresses is misleading as one host might have multiple IP addresses, most probably due to DHCP.

Proximity Routing
In proximity routing, the routing choice is not only based on distance of node from the key. It is also taken into account that which node is closest in terms of latency. In proximity routing the underlying physical network is also considered for making routing decisions.
Kaune et al. [8] have investigated the importance of proximity in Kademlia DHT. They believe that lookup messages might needlessly routed across the long paths which results in high cost to ISPs. It also results in high lookup latencies, they believe. They have given some solutions for improving proximity routing in Kademlia DHT and they have experimentally showed that it improves lookup latency.

Parallelism and Replication
Stutzbach and Rejaie [9] have discussed in their paper, the significance of degree of lookup parallelism and degree of replication of data on different nodes. They have studied Kademlia implementation in e-mule. They believe that DHTs can be made resilient to churn by seeking either the DHT-based solution, which includes improvements in the degree of redundancy or frequency of updates in routing tables or Client-based solution which emphasize on improving lookup parallelism and degree of replication. Although they have done empirical studies of DHT-based solution and they have proved that DHT-based solution can improve lookup performance, changing routing tables in Kad was not under their control. For this reason, in their work, they have suggested Client-based solution as an alternative to increase lookup performance in presence of inaccurate routing tables. They have discussed that the lack of lookup performance due to inconsistencies of the routing tables can be handled by improving the degree of replication and degree of parallelism. They measured effect of alpha on different parameters including number of hops, latency, number of messages sent and concluded that a=3 is the sweet spot where performance is better than for higher or lower values of alpha. In addition, the researchers have discussed how degree of replication of

data can counter the effect of problem due to inconsistent and stale routing entries. They demonstrated experimentally that if identifiers are associated with more than one peer, lookup performance will be better despite of stale routing entries in the routing tables.
They have used three tools for taking measurements – a crawler named 'Blizzard', kFetch and kLookup.

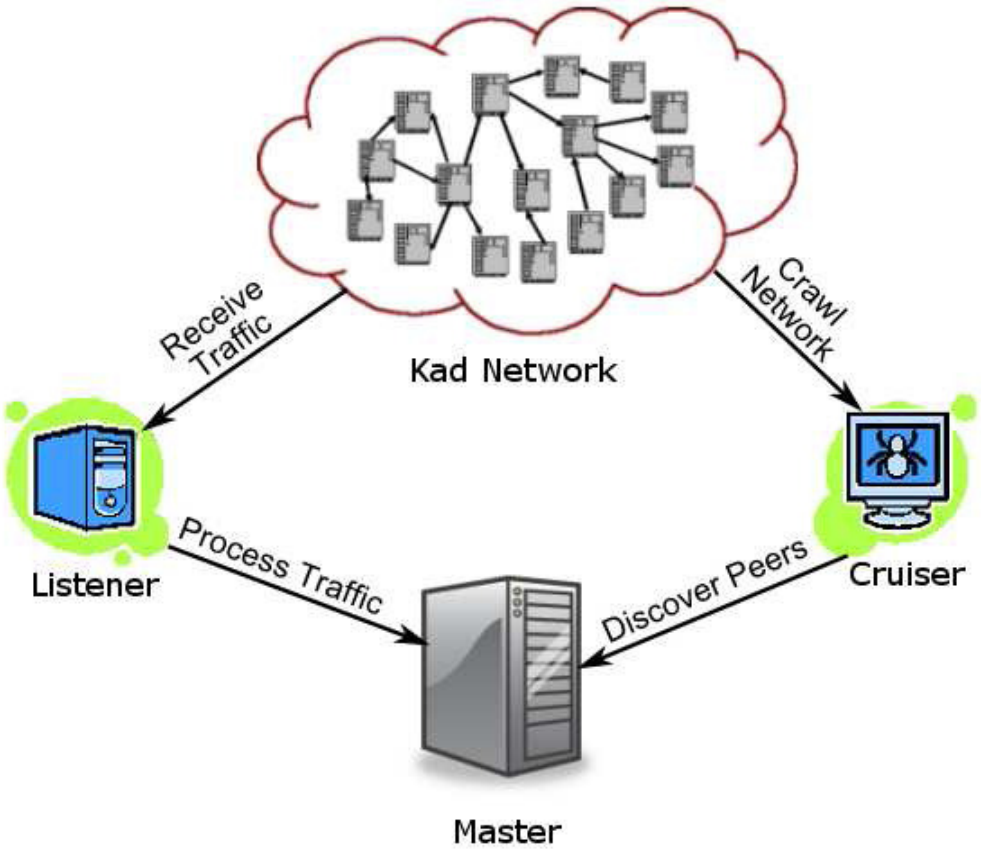
6.2 Monitoring and Measurement Tools

6.2.1 Passive Monitors

One approach of capturing traffic is to add ordinary peers who act passively in the network and record exchanged messages. Passive peers do not send any query but answer normally. Such type of approach has been used by Falkner et al [22] to study availability of content and session time in Kademlia-based Azureus DHT network and Qiao et al. [23] to monitor Kademlia-based Overnet DHT traffic. The problem with this approach is that it requires large number of peers because small number of peers can not provide sufficient details about the network. On the other hand, large number of instrumented peers may disrupt the normal DHT traffic pattern and provide with misleading results.

6.2.2 Montra

Another approach is used by Memon et al. [24]. They have implemented a tool named 'Montra' for monitoring DHT traffic. In 'Montra' they have introduced idea of minimally visible monitors to avoid DHT traffic disruption in case there are a large number of monitors. These monitors add themselves to only one target peer and remain invisible to others. They do not respond to other peers except the target peer. Multiple monitors are placed in a zone named as 'monitoring zone' where each monitor is associated to some target peer and responsible for monitoring destination traffic. To cope with churn they have used crawler which keeps on looking for new peers and readily attach monitor to each of them. Further, they have defined some extensions in 'Montra' to get more information such as content metadata. Using their tool, they have measured different traffic characteristics such as rate of publishing and searching contents and relation between publishing and searching files.



6.2 Montra Architecture

Reproduced from [24]

6.2.3 Mistral

Steiner et al. [25] introduced another traffic monitoring tool. It is meant to investigate publishing process in kad. Mistral works like Sybil attack [33,34]by introducing multiple Sybils - their own peers, in the network**.** These peers which are on the same machine are distributed over the whole identifier space and can gain control over the whole or a part of network.

To make sybils work, they have used crawler 'Blizzard' to know about a set of peers in the network. Then hello message is sent to these peers by their sybils. This way sybils are added to the routing tables of real peers. When real peers search query reaches a sybil, it respond with a fake reply pointing the content to an other sybil. That sybil just record the request but does not provide content. This way searching and publishing requests are recorded by sybils. According to their findings, publishing messages are 10 times more than searching messages and also publishing message is 10 times bigger than searching message because it contains metadata information along with keyword. Moreover they argue that some peers have heavy load on them which is due to popular keywords that are known as stop word and are meaningless. That is why they have proposed a stop word exclusion scheme in Kad to reduce this load. They believe that this exclusion will not affect usability of the system.

6.2.4 Blizzard

Blizzard is a crawler developed by Steiner et al [6] to actively monitor Kad. They used blizzard on one machine unlike some other crawlers which run on multiple machines. They believe that using multiple machines to run crawler consumes CPU time for synchronizing between machines. The working of crawler is started by contacting with as seed peer for getting a list of peers which in turn are contacted to get information about other peers in the network. In this work they measured churn in the network with the help of snapshots taken by 'Blizzard' and proposed their ideas to cope with churn effectively and efficiently.

Again blizzard was used by them [7] to detect ID aliasing in network.(See section 6.1.3) They also used the same crawler to help their tool 'Mistral' for finding active peers in the network. [25] (See above section to know more about 'Mistral')

6.2.5 Cruiser, kFetch and kLookup

Stutzbach and Rejaie [9] have used crawler named 'Blizzard' to actively monitor Kad traffic. Along with this tool they have used tool named **kFetch** for extracting routing table entries of Kad peers and **kLookup** tool for performing lookups over Kad. kFetch select a peer at random and download its complete routing table. Then it probes the entries in routing table actively to detect stale entries. For selecting random peer, kFetch generates a random identifier and sends lookup query to find peer closest to that identifier. kLookup emulates lookup from any source ID to any destination ID. It does this by locating a peer close to source. It uses kFetch to get routing table of source peer. Then it uses routing table of the source peer to do lookup for destination ID. The purpose of all these tools was to find effect of degree of parallelism and replication on lookup performance. (Details about their work they did using these tools can be read from section 6.1.3)

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The major goal of this thesis work was to develop a tool to present MDHT based on Kademlia traffic in the format that makes Kademlia easy to understand, explore and investigate.

To the best of my knowledge, there is no tool available publicly which is particularly designed and developed for Kademlia analysis. Look@MLKademlia is the first open source tool designed and developed for this purpose.

The testing and evaluation of Look@MLKademlia has highlighted several features of the tool which makes it an attractive choice for doing analysis of mainline DHT based on Kademlia.

The comparative analysis of Look@MLKademlia with Wireshark shows that Look@MLKademlia is better, easier and many times faster when both are particularly used for Kademlia analysis. The distinguishing features of Look@MLKademlia include the parallel display of queries and responses, the lookup convergence graph, queries and response patterns graph, outgoing traffic graph, RTT graphs, quick statistical data, IP addresses and ports instead of hexa data, log distances of identifiers from infohash, geographical location of nodes in lookup, IP and ID aliasing.

Look@MLKademlia research capabilities with respect to a single node are evaluated. This evaluation has revealed that Look@MLKademlia is able to show lookup performance in terms of lookup latency, lookup cost in terms of traffic generated and lookup convergence. Moreover it is able to reflect in the lookup process, the effect of changes done in degree of parallelism or in other parameters. The pattern of queries and responses opens several hidden aspects if closed implementation is under analysis. This pattern reveals degree of parallelism used, it makes one able to guess about timeout values, it shows republishing behavior of node, it exposes the times at which management traffic is generated in regular intervals. Further, it shows geographical distribution of nodes in lookup process. Also, it highlights IP and ID aliasing.

All of the research aspects mentioned in the above paragraph are considered to be highlighted in Look@MLKademlia after looking at the research work done on Kademlia. All of them are discussed in the related work in Chapter 6 which establishes their importance in the research community and justifies their presence in the tool.

Besides analyzing single node behavior, Look@MLKademlia is evaluated for hypothesis testing capability. This tool has been used in the thesis done by Ismael Garcia in TSLab to confirm the results of his experiments. He did these experiments on MDHT network.

On the basis of the capabilities of Look@MLKademlia, it is believed that if the arrangements are made to place the tool near some popular infohash, it can reveal important behaviors such as publishing and searching behaviors for that infohash. It can also inform about the incoming and outgoing traffic in that 'hotspot' area.

Also, the statistical data shown in the tool can be used as indicator for possible denial of service attacks on the node.

To sum up, Look@MLKademlia is an open source tool which provides the fast and quick analysis of Kademlia node by analyzing its traffic. It also acts as corroborating tool for confirmation of experiments done on Kademlia node. Moreover, it is believed to bring useful information if it is placed near a popular infohash.

7.2 Future Work

Although the tool works well with offline capturing of data, it would be more convenient to take these captures online.

During testing, it was found that a small number of responses are not shown by Look@MLKademlia due to the way it is implemented. Although, this small number does not affect the over all analysis of the implementation of Kademlia, this issue is needed to be addressed.

References

1. P. Maymounkov and D. Mazi`eres. Kademlia: A peer-to-peer information system based on the XOR metric. In Proceedings of IPTPS, Cambridge, MA, Mar. 2002.
2. D. Wu, P. Dhungel, X. Hei, Chao Zhang, Keith W. Ross, "Understanding Peer Exchange in BitTorrent Systems", Proc. 10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P'10), Delft, the Netherlands, August 25-27, 2010.
3. Scott A Crosby and Dan S Wallach An Analysis of BitTorrent's Two Kademlia-Based DHTs Technical Report TR-07-04, Department of Computer Science, Rice University, June 2007
4. A. Binzenhöfer and H. Schnabel, Improving the Performance and Robustness of Kademlia-based Overlay Networks, University of Wurzburg, Institute of Computer Science, Report No. 405, April 2007.
5. Steiner M., En-Najjary T., Biersack E. W.: Long. Term Study of Peer Behavior in the KAD DHT. IEEE/ACM Transactions on Networking. 2009
6. M. Steiner, E. W. Biersack, and T. En-Najjary, "Actively monitoring peers in KAD," in Proc. 6th Int. Workshop on Peer-to-Peer Systems (IPTPS), 2007
7. Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack A global view of KAD Proc. of Internet Measurement Conference (IMC), October 2007, San Diego, USA.
8. S. Kaune, “Embracing the Peer Next Door: Proximity in Kademlia”. 8th International Conference on Peer-to-Peer Computing 2008, P2P '08, September 8-11, 2008.
9. D. Stutzbach, and R. Rejaie, "Improving Lookup Performance over a Widely-Deployed DHT," in Proc. of INFOCOM, Barcelona, Spain, April 2006
10. T.M Le, J.But , “BitTorrent Traffic Classification”, Centre for Advanced Internet Architectures. Technical Report 091022A, Swinburne University of Technology Melbourne, Australia
11. R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", First International Conference on Peer-to-Peer Computing (P2P'01), 2001
12. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H.Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM SIGCOMM (San Diego, 2001).
13. Stoica, Ion et al. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". Proceedings of SIGCOMM'01 (ACM Press New York, NY, USA 2001)
14. A. Rowstron and P. Druschel. "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany(Nov 2001): 329–35
15. Zhao B. Y., Huang L., Stribling J., Rhea S. C., Joseph A. D., Kubiatowicz J. D. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE J-SAC, 22(1), January 2003

16. J B Meisel, "Entry into the Market for Online Distribution of Digital Content: Economic and Legal Ramifications", (2008) 5:1 SCRIPTed 50 http://www.law.ed.ac.uk/ahrc/script-ed/vol5-1/meisel.asp
17. Gerald Kunzmann, “Recursive or iterative routing? Hybrid!”, KiVS Kurzbeiträge und Workshop 2005: 189-19
18. Ratnasamy S., Shenker S. and Stoica I., Routing Algorithms for DHTs: Some Open Questions, 1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, USA, 2002.
19. S. Saroiu, P. Gummadi and S. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems”, MMCN 2002
20. Jian Liang, Rakesh Kumar, Keith W.Ross, "Understanding KaZaA”
21. R. Bhagwan, S. Savage, and G. Voelker, “Understanding availability,” in Proc. 2nd Int.Workshop on Peer-to-Peer Systems (IPTPS), 2003, pp.256–267.
22. J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a million user DHT. In Proceedings of IMC 2007.
23. Y. Qiao and F. E. Bustamante. Structured and unstructured overlays under the microscope: a measurement-based view of two P2P systems that people use. In Proceedings of ATEC ’06.
24. Ghulam Memon, Reza Rejaie, Yang Guo and Daniel Stutzbach, “Large-Scale Monitoring of DHT Traffic.” Proceedings of 8th International Workshop Workshop on Peer-to-Peer Systems (IPTPS '09), April 2009
25. M. Steiner, W. Effelsberg, T. En Najjary, and E. W. Biersack. Load reduction in the Kad peer-to-peer system. In DBISP2P ’07.
26. ‘BitTorrent Protocol Specification’ http://www.bittorrent.org/beps/bep_0003.html, Last Visited July, 2010.
27. ‘Internet Study 2007’, http://www.ipoque.com/resources/internet-studies/internet-study-2007, Last visited July, 2010.
28. ‘Cache Logic Study’ http://www.slyck.com/story914_CacheLogic_Study_P2P_is_Changing, Last visited July 2010
29. ‘iPoque Internet Study 2008/2009’ http://www.ipoque.com/news-and-events/news/ipoque-internet-study-2008_2009-finds-web-and-streaming-outgrows-p2p-traffic.html, Last visited July 2010.
30. B. Awerbuch and C. Scheideler, “Towards a Scalable and Robust DHT,” in SPAA, 2006, pp. 318–327.
31. Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, J. Hellerstein and S. Shenker, “A Case Study in Building Layered DHT Applications,” in Proceedings of ACM SIGCOMM 2005
32. A. Soro, C. Lai, Range-capable Distributed Hash Tables, in Third International Workshop on Geographic Information Retrieval - GIR'06, Seattle - USA, 2006.
33. Jochen Dinger and Hannes Hartenstein. “Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for self-Registration”. IEEE, Proceedings of the First international Conference on Availability, Reliability and Security, 2002.
34. John R. Douceur, The Sybil Attack, in Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS), 2002

35. Milojicic, D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z., "Peer-to-Peer Computing", HPL Technical Report HPL-2002-57
36. E. Sit, R. Morris "Security Considerations for Peer-to-Peer Distributed Hash Tables". Workshop on Peer-to-Peer Systems, March 2002.
37. S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In USENIX Annual Technical Conference, 2004.
38. K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault tolerance in peer-to-peer networks. In Proceedings of 17th International Symposium on Distributed Computing, Sorrento, Italy, Oct. 2003.
39. Amos Fiat , Jared Saia, Censorship resistant peer-to-peer content addressable networks, Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, p.94-103, January 06-08, 2002, San Francisco, California.
40. Pouwelse, J.A., Garbacki, P., Epema, D.H.J. & Sips, H.J. An introduction to the BitTorrent Peer-to-Peer File-Sharing System. In 19th IEEE Annual Computer Communications Workshop, IEEE Technical Committee on Computer Communications, October 17-20, 2004. Bonita Springs, Florida, USA
41. J. A. Pouwelse, P. Garbacki, D.H.J. Epema, and H. J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. 2005.

Appendix A

Kademlia Queries in Mainline DHT

As discussed earlier, Mainline DHT (MDHT) is implementation of DHT based on Kademlia design. There are four types of queries and responses in Mainline DHT. All of them have 'id' key and value with identifier of sender node.

Ping

Ping query is very simple. It has id key with value as sender identifier. Following is the example of ping query and response. Bencoded format [26] is also given.

Query = {"t":"aa", "y":"q", "q":"ping", "a":{"id":"abcdefghij0123456789"}}
bencoded = d1:ad2:id20:abcdefghij0123456789e1:q4:ping1:t2:aa1:y1:qe

In Query, 't' is transaction ID, 'y' stands for type of message, 'q' is for type of query and 'id' value is identifier of sender node.

Response = {"t":"aa", "y":"r", "r": {"id":"mnopqrstuvwxyz123456"}}
bencoded = d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

Find-Node

Find_Node returns the closest nodes to the ID given in find_node query besides ID of sender node.

Query = {"t":"aa", "y":"q", "q":"find_node", "a": {"id":"abcdefghij0123456789", "target":"mnopqrstuvwxyz123456"}}
Here 't' is for transaction ID, 'y' is for message type , 'q' is for query type and 'target' value is identifier of node whose closest nodes are requested.

bencoded = d1:ad2:id20:abcdefghij01234567896:target20:mnopqrstuvwxyz123456e1:q9:find_node1:t2:aa1:y1:qe

Response = {"t":"aa", "y":"r", "r": {"id":"0123456789abcdefghij", "nodes": "def456..."}}
bencoded = d1:rd2:id20:0123456789abcdefghij5:nodes9:def456...e1:t2:aa1:y1:re

In response, 't', 'y' have the same meaning as described above. 'r' means response. Here 'nodes' key is extra. Its value has nearest nodes to the target ID, given in query.

Get_peers

Get_peers query is the lookup query. It is associated with infohash. The query has two values; ID of sender and infohash for which lookup is done. Response either contains peers responsible for that infohash or closest nodes to the infohash that might provide with the list of peers.

Query = {"t":"aa", "y":"q", "q":"get_peers", "a": {"id":"abcdefghij0123456789", "info_hash":"mnopqrstuvwxyz123456"}}

In query, ‘t’, ‘y’, ‘q’ are the same as in above queries. ‘info_hash’ is identifier to which the closest nodes or peers are required.

bencoded = d1:ad2:id20:abcdefghij01234567899:info_hash20:mnopqrstuvwxyz123456e1:q9:get_peers1:t2:aa1:y1:qe

Response with peers = {"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "token":"aoeusnth", "values": ["axje.u", "idhtnm"]}}
bencoded = d1:rd2:id20:abcdefghij01234567895:token8:aoeusnth6:valuesl6:axje.u6:idhtnmee1:t2:aa1:y1:re

Response with closest nodes = {"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "token":"aoeusnth", "nodes": "def456..."}}
bencoded = d1:rd2:id20:abcdefghij01234567895:nodes9:def456...5:token8:aoeusnthe1:t2:aa1:y1:re

In response of get_peers query, there are either closest nodes to the ifo_hash or peers having the data related to info_hash or both of them. Here the key ‘nodes’ is for closest nodes and the key ‘values’ is for list of peers.

Announce_peer

This query is sent by the peer to announce that it is downloading data related to a particular infohash on a given port. This query contains id of sender, infohash, port and token. Token is used for authenticating the querying node. On receiving this query, queried node updates its peer information about the infohash.

Query = {"t":"aa", "y":"q", "q":"announce_peer", "a": {"id":"abcdefghij0123456789", "info_hash":"mnopqrstuvwxyz123456", "port": 6881, "token": "aoeusnth"}}
bencoded = d1:ad2:id20:abcdefghij01234567899:info_hash20:
mnopqrstuvwxyz1234564:porti6881e5:token8:aoeusnthe1:q13:announce_peer1:t2:aa1:y1:qe

Response = {"t":"aa", "y":"r", "r": {"id":"mnopqrstuvwxyz123456"}}
bencoded = d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

Appendix B

User Manual

Start the tool
The tool is tested in Ubuntu. To start in Ubuntu open the terminal and add following text.
python gui.py

Main screen will open. Go to file menu or 'open' toolbar and open saved pcap file. You will see queries and their corresponding responses in line. On clicking the list item, you will see more details about that query and responses in the second list and text boxes.

Filters
If you have not already used filters, it is better to go to filter screen by clicking 'Expression' button on the main screen. There you can see filter parameters along with the operators that can be used. For example to filter data related to some source address and destinations address you will write

src == 192.16.125.242 and dst == 192.143.222.22

Click 'ok' button. You will see your expression in the filter text box on the main screen. Click 'Filter' button to filter the data.
There are many parameters on which filtering can be done such as source IP address, destination IP address, query time, response time, RTT, Infohash, transaction ID, data (get_peers, find_node, ping, announce_peer).

If you have learned how and which filters are used you can directly write filter in the filter text box instead of opening expression screen. To get the original data again click on 'No Filter' button.

Note: Keep space between every word while writing filter.

Lookup
To see how the lookup process works in Kademlia click 'Lookup' toolbar item. A window will open. Enter Source IP address and Infohash, you want to see lookup of.
The list will display 'get_peers' queries and responses along with other information like query and response time, RTT, log distance of source address from infohash and log distance of responders from infohash. This log distance is a clear indicator of convergence of the protocol towards destination key or infohash. You can see that it is getting shorter and shorter. Along with this information, the lookup window is also showing nodes distances of the nodes that have been sent in responses. If you right click on the list item, you can see IP addresses of those nodes along with their geographical location. List of peers that is obtained by some responders is shown in the text box at bottom.

Graphs

There are four types of graphs generated by Look@MLKademlia. Go to 'Graphs' toolbar item. A small dialog box will appear. Select the type of graph you want to see. The following are the graphs generated by the tool

1- **RTT Graph**
   This cdf graph shows the round trip time taken by different types of queries to get responses. It clearly shows you lookup latency if you are looking at 'get_peers' queries type graph.
2- **Queries and Responses Graph**
   This graph shows you the pattern of queries and responses.
3- **Traffic Generated Graph**
   It shows you how much traffic has been generated from a specific IP address. This bar graph displays four bars each for one type of query that is get_peers, find_node, ping, announce_peer.
4- **Convergence Graph**
   Convergence graph is time distance graph where x-axis is showing time while y-axis is displaying log distances. One thing is important that convergence is related to lookup queries. That is why convergence graph requires source address and infohash. Using that source address and infohash, it gets get_peers queries and plot the distances of responding nodes to show how graph converges to infohash value. It is worth mentioning here that convergence graph is exact graphical representation of lookup process you see by pressing 'Lookup' toolbar item.

Statistical Data

On right clicking the list in the main screen you can see small window with the source address of the list item you clicked. On pressing 'Ok' button you will see some statistical information regarding that source address such as queries and responses generated and received along with average RTT.

IP and ID Aliasing

Clicking on the colored pellets you can filter IP aliasing or ID aliasing data. To know what is IP and ID aliasing go to section 5.4.

Same Transaction ID and Errors

Similarly clicking on other pellets you can see same transaction IDs data and Errors.

Sorting Columns

You can see sorted data on the basis of three columns. Query time, response time and RTT. Clicking on these columns will give you sorted data on the basis of column clicked.