GnuSim: A General Purpose Simulator for Gnutella and Unstructured P2P Networks

Murat Karakaya, İbrahim Körpeoğlu and Özgür Ulusoy
Computer Engineering
Bilkent University Ankara, Turkey
{muratk,korpe,oulusoy}@cs.bilkent.edu.tr

Abstract— **Peer-to-peer networks have attracted a significant amount of interest as a popular and successful alternative to traditional client-server networks for resource sharing and content distribution. Since a P2P network consists of many nodes, thousands or millions, it is hard to evaulate the performance of a P2P network and some related protocols analytically using only mathematical models. Most often we need to do simulations. Therefore it is important to have a simulation tool specifically designed for P2P network and protocol simulation. We developed such a tool, and in this report we present the design and implementation of this simulation tool, which we call GnuSim. The report includes both the simulation model and the code details. This work will be useful for researchers doing simulation study in P2P systems domain.**

I. Introduction

Peer-to-peer (P2P) networks have attracted a significant amount of interest both in the Internet community and in the academic world as a popular and successful alternative to traditional client-server networks for resource sharing and content distribution. P2P networks are implemented as overlay networks on top of the existing Internet infrastructure. There have been many system proposals and applications in the main functional areas of P2P paradigm such as data placement, file lookup, replication, etc. Most of these efforts aim to provide efficient, effective and fast exchange of files between peers.

Although there exist different architectural designs and applications for P2P file sharing, in nearly all P2P systems files are stored at peers, searched through the P2P network mechanisms, and exchanged directly between peers using the underlying network mechanisms. In an ideal case, peers share their content with other peers and a file that is downloaded by a peer is automatically opened for sharing with other peers. However, peers can, and frequently do, abstain from sharing any content to economize on their own resources such as bandwidth. Therefore, the primary property of P2P systems, the implicit or explicit functional cooperation and resource contribution of peers, may fail and lead to a situation called free riding.

In this report, we introduce a new simulation tool for modelling Gnutella and unstructured P2P networks. As part of this simulation tool, we implemented an unstructured P2P model with many number of parameters to observe the effect of many factors and mechanisms applied on a P2P network. The implementation has many levels of detail such as the number of peers, network topology, content distribution and replication, free riding, time-to-live value, query pattern and query generation rate, etc. The model can be extended to simulate other types of P2P networks as well.

The goal of the simulation study is to validate the schemes that are proposed to use in P2P networks, to evaluate the performance of the schemes and the cost of them. Furthermore, the model can be used to measure and compare the performance metrics of different models of P2P network functions such as querying content and downloading.

The organization of the report is as follows. Sections II is devoted to related work and background information about P2P networks and related concepts. In Section III describes the simulation model in detail. In Section IV, we present detailed information about our simulation code. The performance report, which is produced as an output of the simulation program and the trace file which may used to debug and control of the flow of the simulation are presented in Section V. The conclusions are provided in Section VI.

II. Background

In this section, we describe some basic P2P systems concepts and protocols upon which our simulation model is built. Much more details about P2P networks and protocols can be found in [7], [24], [4], [5]. The observations and findings about P2P network traffic measurement, modelling, and peer behaviors can be found [2], [6], [10], [9], [8]. Some of the important P2P applications exist on the Internet are [28], [29], [22], [21], [17], [30], [20], [27], [31]. Information about free riding issue can be found in [1], [3], [25], [26], [32], [11], [12], [13], [15], [16].

We focus on unstructured P2P networks like Gnutella, because of their popularity and well-known protocols [24]. Unstructured P2P networks have the distinct properties that can be summarized as [4]:

- no central coordination
- no central database
- no peer has a global view of the system
- global behavior emerges from local interactions
- all existing data and services should be accessible
- peers are autonomous and anonymous
- peers and connections are unreliable

These features enabled unstructured P2P networks to be very successful, but also brought some problems. Among the problems of such networks is the so-called reputation problem.

In an unstructured P2P network such as Gnutella, peers interact with unknown peers and have no information about their reputations. In other words, they do not know to what extent they can trust the other peers and the data provided by them.

A. P2P General Scenario

The basic service that a P2P network provides is answering user queries and enabling file downloads. A requesting peer may send queries to the P2P network, wanting the network answer the query. The query can be, for example, "list me the source peers that have the file x". The outcome of this request can be one of the following: success (the P2P system could find the file and respond with a query hit); failure (the P2P system could not find a node sharing the file in the search process). If the peer could get a list of source peers that have the file, the requesting peer then can select one source peer and try to download the file from that source peer. Again, the outcome of this request can be one of the following: success (the source peer really has the file and provides the file, the network resources are enough for the download operation, and the download is successfully realized); failure (the source peer lied by sending a query hit and it really does not have the file, or the network resources are not enough to realize the download operation).

B. Phases in P2P communication

In an unstructured P2P network, a peer may go through four main phases:

- Connection phase: In this phase, the peer tries to find some other peers which have already been connected to the P2P network. On finding some peers, it announces its existence to these peers. These connections will be used to broadcast any search requests of the peer. Furthermore, connected peers may communicate with the peer for the search requests initiated by other peers.
- Search Phase: The peer needs a file and initiates its search operation by broadcasting a search message through its neighbors. It then waits for replies.
- Downloading Phase: If the peer has received a hit message, then it may begin to download the file from the source peer through a direct connection.
- Local Search and Routing Phase: The peer can have some search queries delivered to itself by neighboring peers. It first checks its local resources. If it has the file it returns a hit message to the neighboring node. Either it has the file or not, it decreases Time-To-Live (TTL) value of the search message, and if TTL value is greater than 1, it forwards it to all neighbors other than the one which has delivered the search. If any hit message arrives, the peer routes it back to the requesting peer.

These phases are implemented with descriptors in Gnutella Protocol [24] (See Table III).

C. Free Riding Implemented in Scenario

As a P2P networking concept, free riding (FR) means exploiting P2P network resources (through searching, downloading objects, or using services) without contributing to the P2P network at desirable levels.

User traffic on Gnutella network is extensively analyzed in [1] and it is observed that 70% of peers do not share any file at all. Furthermore, 63% of the peers who share some files do not respond to any queries. That is, they are sharing some files but nobody is interested in them and therefore no queries are generated searching for these files. Another interesting observation is that 25% of the peers provide 99% of the whole content in the network.

In the following discussion, we define three possible free riding types and identify some clues that can be used to detect them [1]. A summary of free riding Types' properties is given in Table I.

- **The peer does not share anything at all or shares uninteresting files.** It may be observed that a neighboring peer does not return any `QueryHit` messages to the queries that it receives. There may be two reasons for that: either the peer does not have any files matching the queries, or the peer does not share any files at all. This kind of free riding peers can be called a *non-contributor*.
- **The peer consumes much more resource than that it shares.** A peer may share some documents and upload them to the system. However the number of its downloads highly exceeds the number of its uploads. The comparison of these two numbers indicates that any peer may consume more than it share. This kind of peers may be considered as a free-rider and named as a *consumer*.
- **The peer drops others' queries.** A peer may drop P2P protocol messages of other peers to save its resources. Especially, it may not route the neighboring peers' `Query` and/or `QueryHit` messages. We call this type of free riders *droppers*.

Researchers have observed the existence of high degrees of free riding in P2P networks and they suggest that free riding may be an important threat against the existence and efficient operation of P2P networks [1], [9]. Adar and Huberman argue that "free riding leads to degradation of the system performance and adds vulnerability to the system. If this trend continues copyright issues might become moot compared to the possible collapse of such systems" [1].

In a more recent work, Saroui et al. confirm that there is a large amount of free riding in Gnutella network as well as in Napster [9]. Another interesting observation is that 7% of the peers together provide more files than all of the other remaining peers. The authors suggest that the system should not treat its peers equally, on the contrary it should provide the right incentives and rewards for peers to provide and exchange data.

Considering the importance of free riding for P2P networks, in our simulation tool, we provide several parameters to simulate FR phenomena and observe its effect on the system.

[1]The free riding types defined here are not an exhaustive list. There could be a more and detailed classification of possible FR types. We believe that, considering the purpose of the work, the defined FR types are enough to cover and enable the detection of most of the free riding behaviors which may seen in a P2P network.

Free Riding Type	Sharing Content?	Request Generation Speed	Routing Messages?
NONE	Yes, much	Normal	Yes
NON-CONTRIBUTOR	No	Normal	Yes
CONSUMER	Yes, but little	Higher	Yes
DROPPER	No	Normal	No

TABLE I
SUMMARY OF FREE RIDING TYPES

D. Performance Metrics

As stated before, the aim of the work is to enable researchers to observe the performance of a P2P network under different scenarios and different mechanisms. Therefore, we set up three parameters for Quality of Service (QoS) and their metrics.

For example, a work aiming to implement a scheme reducing free riding in a P2P network may use these metrics. Researchers may want to observe that Quality of Service (QoS) for non-free riders is increased, while being diminished for free riders.

The QoS improvement of a P2P system has to be measured by using some metrics. We would like to use the following QoS parameters and related metrics to evaluate the performance of a P2P system that applies a proposed method. We classify QoS parameters under three topics. Summary information about parameters and metrics given in Table II. Below, details of each parameter is presented. All these metrics are observed during simulation and presented in the result report for each peer type.

- *Availability*: The availability of content and services in P2P network can be an important issue. For example, if we consider upload capacity of peers, we may recognize that some peers may have upload bottleneck due to high level of downloading requests. When they reach the limit of upload capacity, they begin to refuse new requests. The number of refused download requests (unsuccessful downloads) may be used as a metric for availability of content. Another metric for availability parameter may be the number of downloaded files. Furthermore, Query Hit Ratio can be calculated using number of files requested to download and number of queries submitted.
- *Load Sharing* : A large number of search and download operations may go towards few peers and this may lead a bottleneck. In ideal case in P2P system, the load on peers can also be shared by peers. This will help the system to be more efficient so that larger number of search queries and download operations can be executed on the system successfully. Specifically, the number of uploads done by each peer type may be used as a sample metric for this reason.
- *Scalability*: One of the important scalability issues in P2P networks is the flooding of the messages. As the number of messages routed in the network increases each peer needs to handle more messages. Network congestion can be occurred for large amounts of messages. Network congestion can affect several services such as login to network, querying for the content, and downloading files [18]. Therefore, we decide to observe the number of P2P network protocol messages as a metric related to network congestion.

III. ASSUMPTIONS AND PARAMETERS OF THE SIMULATION MODEL

We have implemented an event-driven P2P network and protocol simulator using CSIM 18 [19] simulation library and C++ programming language on the WINDOWS OS. The basic characteristics of the model are set to be similar to those of Gnutella network by implementing the protocol described in [24]. Main message types of the protocol are presented in Table III.

Before discussing details of the model, we present the main parameters of the simulation environment in Tables IV, V, VI, VII. All these parameters are set up in *parameter.h* file given in Appendix D.

In the following subsections,, details of the important parameters and related assumptions are provided as classified into those subsections: Network, Peers, Content, Request. Before that however, we present first the parameter that is not classified into one of those subsections: simulation time.

Simulation Time: This is a parameter defining how long a simulation run will last. The unit used by the parameter is *simulation time unit*. When the specified simulation time expires during a simulation run, a simulation report file is generated and the simulation program terminated. The report file can then be used for analysis.

We next describe other parameters.

A. Network

Network Topology: The network topology defines the connectivity between peers. We assume that number of peers (nodes) and their interconnections (links) are fixed and never changes during a simulation run. Many previous works observe that P2P networks manifest small world phenomena. That is, most of the peers have small number of neighbors (e.g., 3 or 4) whereas few number of peers have large numbers of neighbors. However, as Power-Law states, a few peers can have many neighbors. This configuration can be created by using a topology generator. The simulation program reads topology information from an input file defined by parameter NEIGHBOURS_TABLE_SETUP_FILE_NAME. The file organization is very simple. In the first line, total number of peers is written. In each of the following line, there are *ids* of two peers separated by a space character which indicates that these

QoS Parameter	Observed Metric(s)
Availability	# downloads
Availability	# unsuccessful downloads
Availability	Query Hit Ratio
Load Sharing	# uploads
Scalability	# P2P messages

TABLE II
SUMMARY OF QUALITY OF SERVICE PARAMETERS AND METRICS

Descriptor	Description	Content
Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.	Nothing
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.	IP and port of responding host, number and size of files shared
Query	The primary mechanism for searching the distributed network. A servent receiving a `Query` descriptor will respond with a `QueryHit` if a match is found against its local data set.	Minimum speed requirement of the responding host;search string
QueryHit	The response to a `Query`. This descriptor provides the recipient with enough information to acquire the data matching the corresponding `Query`.	IP and port, speed of responding host; number of matching files and their indexed result set
Push	A mechanism that allows a firewalled servent to contribute file-based data to the network.	Responding host id; file index; IP and port of requesting peer

TABLE III
GNUTELLA PROTOCOL DESCRIPTORS

Parameter	Definition	Default Value
MESHEDGE	Number of peers in one edge of mesh, if mesh structure is used for topology generation.	20
NUMPEERS	Total number of peers in the simulation.	MESHEDGE * MESHEDGE
maxNoOfCon	Maximum number of neighbors that a peer can be connected to.	Depends on topology

TABLE IV
TOPOLOGY PARAMETERS

two peers are neighbors. A sample topology file is presented in Appendix C.

In performance tests, we use a mesh structure to model the network topology for the sake of simplicity (see Fig. 1). The average number of connections (NAC) is between 3 and 4. Given a mesh topology, the exact value for the average number of connections can be calculated as:

$$NAC = (4mn - 2(m + n))/mn \tag{1}$$

where m and n are the dimensions of the mesh. For example, for a 20x20 mesh, NAC is 3.8.

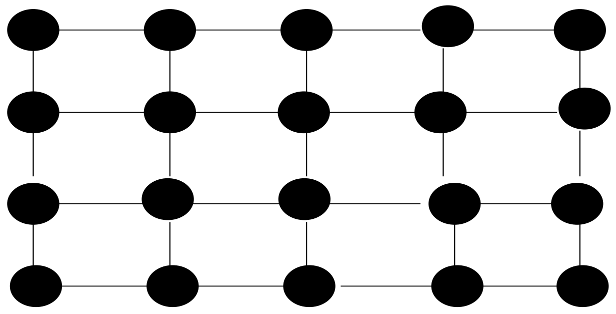

Fig. 1. A mesh topology for network connections.

Messaging: There are two mailboxes within each peer. One mailbox is used for P2P network messages, and the other is used for download requests and downloads. In this way, each mailbox simulates a port in Gnutella and TCP/IP protocol stack running on a peer.

Connection Duration: A peer is assumed to stay connected in the network during the whole simulation lifetime.

Pinging Frequency: To check the validity of the connections with its neighbors, each peer submits a PING message at every PINGFREQ seconds.

Time-To-Live (TTL): In unstructured P2P networks, messages are broadcasted into the network. The TTL parameter is a technique used to limit the broadcast horizon in the network. In the simulation, we assume that maximum TTL value for any P2P protocol message may be set to 7.

B. Peers

Peers and Peer Types: We simulate a population of peers, *NUMPEERS*, constituting both free riders and contributors. Peers are grouped according to the given *NUMBER_OF_PEER_TYPES* parameter and the corresponding properties that are given in Table V. We selected the default values in accordance with the observations done in [1].

Ratio of Free Riders: At the beginning of the simulation, peers are grouped into different types according to the NUM_OF_PEER_TYPES parameter. The number of peers in each type is determined according to the POPULATION_RATIOS parameter considering the total number of peers (NUM_PEERS). For each peer type, we can set FREE_RIDING_TYPE to determine FR type of peers in that group. The possible values of the FREE_RIDING_TYPE parameter are: NONE, NON_CONTRIBUTOR, CONSUMER, DROPPER, and MIXED. MIXED means that the peers in that type are equally distributed to each of the three free riding types. Peers' free riding types will not change during a single run of a simulation (i.e. during the simulation lifetime).

Upload Bandwidth Capacity: We assume that each peer has a limited bandwidth capacity to download and upload files. Download capacity is assumed to be 1. That means there is only one download operation that can be executed at a time. However, a peer can do more than one uploads at the same time which is limited by the value of NO_OF_MAXIMUM_UPLOADS parameter.

Download Attempts: If a peer reaches to NO_OF_MAXIMUM_UPLOADS, it can refuse more uploads. If a requesting peer is refused by a resource peer, it can try another source peer if there is any in the queryHitList. MAX_DL_ATTEMPT_NUMBER specifies how many times a peer should try to download the same file from different source peers if any peer refuses to upload the requested file.

C. Content

Content Distribution: We distribute the content to peers uniformly. First of all, peers are grouped into different types based on the NUM_OF_PEER_TYPES parameter. Later, according to the given SHARED_FILE_RATIOS parameter, the number of files to be distributed for each peer type is calculated. At last, for each type of peers, the determined number of files are distributed uniformly. However, if a free rider peer is specified as a *dropper* or a *non-contributor*, no files are distributed to it. The files saved from this kind of peers are redistributed to *consumer* peers of the same peer type. Furthermore, at the beginning of the every simulation run, the distribution of the files to the peers is the same.

Content Replication During Simulation: The settings given above are valid for the beginning of the each run of the simulation. During the simulation the settings can be changed according to peers' property to replicate the downloaded files. If peers' REPLICATION property is set "true" then the downloaded files are replicated and shared. Therefore, the content distribution in the system would be dynamic during the simulation time.

Size of files: We assume that each file has the same size and download time for all the files are the same which is defined by DOWNLOAD_TIME parameter.

Uniqueness of the content: At the beginning of the every simulation run, the distinct files (DISTINCT_FILES) are copied according to given three parameters:

- COPY
- RANGE
- SKEWNESS

The first parameter specifies the number of copies of each distinct files. For example if the number of distinct files (DISTINCT_FILES) is 100 and the COPY parameter is 2 then, it means that the total number of files (TOTAL_FILES) in the simulation would be 200 and there are 2 copies for each file. If we want some files have more copies than the other files we use the second and third parameters. For example, if we want that only the first 10 files have 2 copies but others have only one copy then the setting should be as follows:

Parameter	Definition	Default Value
NUM OF PEER TYPES	Number of peer types in the simulation.	3
POPULATION RATIOS	Population ratios of each peer type.	{0.10, 0.20, 0.70}
FREE RIDING TYPE	Free riding types of each peer type.	{NONE, NONE, MIXED}
SHARED FILE RATIOS	Ratio of shared files of each peer type to total files in the simulation	{0.87, 0.12, 0.1}
NO OF MAXIMUM UPLOADS	Maximum number of uploads a peer can provide at the same time.	{3, 3, 3}
QUERY GENERATION MEAN	The mean value of the exponential distribution that defines the time between two consecutive generated queries.	{60, 60, 60}
CONSUMER QUERY GENERATION MEAN	The mean value of the exponential distribution that defines the time between two consecutive queries generated by Consumer peers.	60
REPLICATION	If peers' REPLICATION property is set "true" then the downloaded files are replicated and shared.	{true, true, false}

TABLE V
PEER TYPE PARAMETERS

Parameter	Definition	Default Value
DISTINCT FILES	Total number of distinct files in the simulation.	NUMPEERS * 10
COPY	Number of copies of each distinct files.	2
RANGE	Range of files to be replicated more than COPY.	0
SKEWNESS	Number of extra copies of the files in the RANGE.	0
TOTAL FILES	Total number of files in the system.	DISTINCT FILES*COPY + RANGE*SKEWNESS

TABLE VI
CONTENT PARAMETERS

DISTINCT_FILES 100
COPY 1
RANGE 10
SKEWNESS 1

The above setting means that each file has only one copy, but the first 10 files (0..9) have 1 more copy. Therefore, TOTAL_FILES would be 110.

D. Request

Request-File Matching: We assume that the system replies the queries with exact matches only. In reality, Gnutella can also reply with partial matches. For example, if we execute a query for file "barismanco.mp3", we will get a reply if only the system has a file with name "barismanco.mp3" in a peer and if the query has reached to that peer. We do not simulate the system with keyword searches for the time being.

Request Pattern: Peers randomly (uniform distribution) select a file to be requested from the P2P network. After selecting a FileId to be requested, it is checked if the peer itself has the file. If the peer does not have the file then it generates a Query message and submits it to its neighbors. All the files have equal probability for getting requested.

Request Generation Rate: The inter arrival time distribution of requests generated by a peer follows exponential distribution with a mean value QUERY_GENERATION_MEAN which is supplied to the simulation as a peer type parameter to observe the effect of the request pattern.

The request rate (queries/second) originating from a CONSUMER free rider peer is assumed to be larger than the request rate originating from other peers and the CONSUMER_QUERY_GENERATION_MEAN parameter is set up for this reason.

IV. Code Details

In this section we present the details of the simulation code.

A. Main Classes

1) List Class: The List class is an implementation of a linked list structure. It consists of a node and this node has an integer variable and a node pointer to the next node in the list. We used list to hold three types of data:

- Each peer has a *files* list to hold the *FileId*s owned by that peer.
- Each peer has a *download list* to select a peer, who sends a QueryHit to its Query, to download the file.
- Each peer type has a list consisting of peers of that type.

2) TimedList Class: The TimedList class is an implementation of a linked list structure. The difference between List and TimedList is that TimedList has three variables in its node. One of them holds the id of the node, which can be a query ID or a neighbour ID or a peer ID. One of them holds extra values needed for different uses of TimedList and the last of them, which is a double floating number, holds the timeout value of the node. The most important role of the TimedList class in simulation is that it can apply the current simulation time to the nodes timeout and delete all the nodes whose timeout has expired. We used TimedList to hold three types of data:

- Each peer has a routing table to cache incoming queries and pings and route their query hits and pongs back. Id value of the query is query ID, extra value of the query is routed peer ID. We delete the queries after the messages' timeout has expired or the query has passed HITWAITTIME.
- Each peer has a neighbors table to have up to date neighbors. After each Ping or Pong arrival from a peer in the neighbors table we update the timeout value of that peer. If the peer does not send a Ping or reply with a Pong in 3 * PINGFREQ time, we delete that peer from the neighbors table.
- Each peer has an upload list to count uploads and update it. When a peer starts an upload it adds the fileId and the peerId to the upload list and sets its timeout value to DOWNLOADTIME. When the download time finishes, the system automatically deletes this upload from the list, so new empty connections can be established.

3) Message Classes: There are two different message classes. One for P2P protocol and the other is for TCP/IP protocol.

P2P protocol message class: There are 4 types of simulated messages for P2P protocol: Query (Q), QueryHit (QH), PING, PONG. To cover all necessary information of each message type, a generic message class is created. Properties of the class are given in Table VIII.

Details of each implemented P2P protocol message type are presented below.

- QUERY MESSAGE: Any peer searching for a file submits Query message with a QueryId, requested FileId, and TTL (set to 7). PeerId will be the id of the intermediate peers which are routing the Query message. The peer waits until HIT_WAIT_TIME expires or the number of QUERYHIT messages exceeds the SATISFIED_QUERY_HIT parameter. After then, if there is at least one hit, it starts the download cycle. Otherwise, if there is no hit to its query, it waits for QUERY_GENERATION_MEAN time to generate a new query.
- QUERYHIT MESSAGE: Any peer which has the requested file replies with QueryHit message including QueryId of the Query message, requested and found FileId, its PeerId. TTL is the TTL of the Query message.
- PING MESSAGE: Any peer desiring to connect to a peer or to check the connection submits a Ping message to its neighbors with a proper QueryId. FileId is null. PeerId is the id of the peer itself and TTL is 7.
  When a peer receives a Ping message, it first checks if the owner of the Ping message is its neighbor. If the owner is its neighbor, it sends a Pong message as a reply. If the owner is not a neighbor, the peer checks whether it has available connections to connect this peer itself. If there is an available slot for a new connection in this peer, it replies this Ping with a Pong message.
  If there is not an available slot to connect the owner of the Ping message, and if TTL value of the message is greater than 1, it changes peerId of the message to its own peerId and forwards it to other peers.
  To check the validity of the connections with its neighbors, each peer submits a PING message at every PINGFREQ time. If a peer does not get a PING message or a PONG message from its neighbor for a long time (currently 3 * PINGFREQ), it deletes this neighbor from neighbors table.
  To check the validity of the connections with its neighbors, each peer submits a PING message at every PINGFREQ seconds. If it did not get a PONG message from existing neighbors, it deletes this neighbor from NEIGHBORS table.
- PONG MESSAGE: Any peer receiving a Ping message replies with a Pong message if either the connection is alive or it can accept the pinging peer as a new connection. If the connection is not alive and if there is no empty connection capacity (reached MaxNumOfCon) it only routes Ping messages as other messages.
  For simulation purposes, at the beginning of simulation it is not necessary to create Ping, Pong and Push messages as Gnutella protocol suggests. Because, at the beginning of the simulation, each peer is connected to some other peers according to the given network topology.

Parameter	Definition	Default Value
MESSAGE PROCESSING TIME	Time to process any message.	0.1
MESSAGE WAITING TIMEOUT	The time duration of a peer to listen to its mailbox for any incoming message.	0.1
MESSAGE STORING TIMEOUT	The time for keeping information about a message: when timeout occurs all information about that message is deleted from routing table.	5.0
HIT WAIT TIME	The time for a peer to wait the QUERY HIT messages to arrive itself before beginning to download process.	5.0
PINGFREQ	The time between two consecutive Ping messages.	20.0
SATISFIED QUERY HIT	The minimum number of QUERY HITS arrived to requesting peer to begin download process.	3
TIME TO LIVE	The maximum number of hops that a message can be transferred.	3
MAX DL ATTEMPT NUMBER	The maximum number of attempts to download a file from arrived QUERY HITS.	3

TABLE VII
P2P PROTOCOL PARAMETERS

Property	Definition
messageType	messageType:Query,RoutedQuery,QueryHit
queryId	query id
fileId	the file which is looked for
routingPeerId	the peer that routing the message
replyingPeerId	the peer that replying the message
TTL	remainin life time of message(in terms of number of hopes)

TABLE VIII
P2P PROTOCOL MESSAGE CLASS PROPERTIES

However, during the simulation, peers may leave or be disconnected from the network due to some reasons. Therefore, we set several rules. First of all, the maximum number of connection for a peer is maxNoOfCon. A disconnected peer should find a peer with a current connection number less than maxNoOfCon to reconnect. A disconnected peer can search for such a peer submitting Ping messages. Any peer with a current connection number less than maxNoOfCon answers with a Pong message. If a peer receives more than one Pong messages, it uses all of them. It does exploit them to fulfill all the empty connections until it reaches maxNoOfCon.

TCP/IP protocol message class: Two types of messages for the TCP/IP protocol are implemented: DOWNLOAD_REQUEST (DR) and DOWNLOAD_REQUEST_REPLY (DRR). To cover all necessary information of each message type, a generic message data structure is used. Properties of the class are given in Table IX.

- DOWNLOAD_REQUEST MESSAGE: Any peer requesting for the file which is found at the source peer (known from QH message located at QueryHit List) submits DR message with the FileId of the file and PeerId of itself. Initially reply type is 0.
- DOWNLOAD_REQUEST_REPLY MESSAGE: Any peer receiving a DR message replies it with a DRR message. If the resources are available for the upload, reply type is set to Download_Acknowledged. If the file is not available in the requested peer, then it replies with the type Wrong_File_ID. If the requested peer reached its upload capacity before this request, than it replies with the reply type No_Upload_Capacity.

4) Peer Types Class: Class peerTypes encapsulates the properties of a peer type. Initialization of the peers and phases of life cycles of the peers are determined by the properties given in Table X.

Property	Definition
messageType	Type of TCP message: DOWNLOAD_REQUEST_REPLY or DOWNLOAD_REQUEST
peerId	peer Id
fileId	the file which is looked for
replyType	if the request is accepted: Download_Acknowledged, Wrong_File_ID or No_Upload_Capacity

TABLE IX
TCP MESSAGE CLASS PROPERTIES

Property	Definition
populationRatio	the ratio of population of the given peer type to entire population
sharedFileRatio	the ratio of files shared by the given peer type to entire files
noOfMaxConnections	the maximum number of connections a peer can have in a given peer type
noOfMaxUploadConnections	maximum number of uploads a peer can make in a given peer type
queryGenerationTimeMean	the exponential mean value of sleep time for a peer in a given peer type
freeRidingType	free riding type of the peers in this type
replicationEnable	determines if the peers in the given peer type replicate the files they download

TABLE X
PEERTYPES CLASS PROPERTIES

B. Main Functions

- `void queryHitnPongForward(int peerId, msg * message)`
  This function finds the routed peer who sent the query or ping message to this peer using routing table, copies the message and sends it to routed peer. After that it adds one message to allMessagesQtable using `note_entry()` function.
- `void sendDownloadRequest(int peerId, int fileId, int & attemptNumber)`
  This function finds the source peer to download the file with the help of attemptNumber, creates a download request and sends it to source peer. After that it increments attemptNumber by one, adds one message to allMessagesQtable using `note_entry()` function.
- `void sendDownloadRequestReply(int peerId, tcpMsg *tcpMessage, int replyType)`
  This function creates a download request reply message with the help of tcpMessage, and sends the message to tcpMessage's peerId. After that it adds one message to allMessagesQtable using `note_entry()` function.
- `void sendQueryHit(int peerId, msg * message)`
  This function creates a query hit message with the help of message and sends the message to the peer who routed that query to itself. After that it adds one message to allMessagesQtable using `note_entry()` function.
- `int selectPeerToDownload(List *list, int attemptNumber)`
  This function returns a new peer to download the file using attemptNumber and downloadList.
- `bool alreadyReceived(int queryId, int peerId)`
  This function checks whether the peer with peerId got the query before using routing table of the peer.
- `void forwardToNeighbour(int peerId, int neighbourId, msg * message)`
  This function copies the message and sends it to peer with Id neighbourId. After that it adds one message to allMessagesQtable using `note_entry()` function.
- `void sendQueryToNeighbour(int peerId, int neighbourId, int fileId, int querySeqNo)`
  This function creates a query message using fileId, peerId and querySeqNo. Then sends it to peer with ID neighbourId. After that it adds one message to allMessagesQtable using `note_entry()` function.
- `void pingNeighbour(int peerId, int neighbourId, int querySeqNo)`
  This function creates a ping message using peerId and querySeqNo. Then sends it to peer with ID neighbourID.

After that it adds one message to allMessages qtable using `note_entry()` function.

- void sendPong(int peerId, msg * message)
  This function creates a pong message with the help of message and sends the message to the peer who routed that query to itself. After that it adds one message to allMessages qtable using note_entry( ) function

C. Initialization of the Simulation Model

First, we initialize the following tables and the lists.

- files, array of List pointers,
- routing Tables, array of TimedList pointers;
- neighbors Table, array of TimedList pointers;
- download List, array of List pointers;
- uploadList, array of TimedList pointers;
- pTypes, array of List pointers;

After initialization of these lists and tables, we filled Neighbors table with the topology data gathered from the file specified by the NEIGHBOURS_TABLE_SETUP_FILE_NAME parameter. Later, we initialized the CSIM Table data structures to obtain statistics from the system.

Peer types are created according to the data gathered from the parameters defined in Section III-B. The number of peers in each peer type is computed and randomly chosen peers are assigned to these peer types. The number of files of each peer type is determined according to the shared file ratio of each peer type to the number of total file number.

Following the global initialization of the simulation the initialization of each peer starts. The download and upload lists of peers are initialized. After the initialization of these lists, the free rider type of the peer is decided if it is a free rider peer. The maximum number of connections value of the peer is set to a number that is one more than the number of its current neighbors. The first ping time and the first query time of the peer is set, and the life cycle of the peer starts.

D. Life Cycle of a Peer

Every peer continuously executes a cycle during the simulation. The life cycle of a peer consists of five phases which simulate the main P2P communication phases defined in Section II-B.

A peer starts every cycle with applying timeout to its TimedLists, which are routing tables, upload lists and neighbors table. If any node's timeout value has expired, that node is deleted from these lists. Then, with the refreshed information, the peer starts its life cycle as described below.

The first phase of a peer's life cycle - Message Processing: The peer calls *timed_receive* function of CSIM library to receive the message from the P2P protocol mailbox if there exists. If there is no message in the mailbox, the peer passes the first phase and starts the second phase. If there is a message in the mailbox, and the peer receives it successfully, the peer increments the received message number and processes the received message according to its type as defined below.

- Query: The first control on the query is whether we have received this query before or not. This check can be done by searching routing table. If we have received this message before, we don't need to process it again. Therefore we delete this message and proceed with the following phase of the peer life cycle. If this message is a new one, then the life cycle continues with a hold for a message processing time. After that the peer records the query information to its routing table to be able to route QueryHit message back successfully to its owner. After adding this query to the routing table, the life cycle continues according to the free rider type of the peer. If the peer is a DROPPER, then it deletes the query and ends the first phase of its life cycle. If the peer is not a CONSUMER, then it executes the query on its own files and replies with a QueryHit if it finds the requested file among its files. If the peer is a NON_CONTRIBUTOR, then it does not look for the requested file and passes to the next step. The next step is forwarding the Query message if its time to live (TTL) is not expired. If the message's TTL is greater than 1, then the peer copies the message and forwards the message to all the other neighbors. After this step the peer deletes the message and ends the first phase of its life cycle.
- Query Hit: We wait MESSAGE_PROCESSING_TIME to pass at the start of processing query hits. Then, we check whether this peer is the owner of the query. If this peer is the owner, it again checks whether the file of the query hit is the same with the file of its last query. If they are the same, then the peer adds the owner of the query hit to its download list. If they are not the same, then the peer deletes the message and completes the first phase of its life cycle. If this peer is not the owner of the query then it routes the query hit message to the owner of the query with the help of the information in its routing table. After sending the message, the peer completes the first phase of its life cycle.
- Ping: Processing ping is nearly the same as processing a query. We start with a control of the ping in our routing table. If we find the ping's query ID in the routing table, then it means that we have received this Ping message before. So we delete this message and return to the start of the life cycle. If this message is a new, one then we add this message to the routing table. Then, we continue to execute controls. We check whether the owner of the Ping message is one of our neighbors. If it is, then we update its timeout value in the neighbors table and send a Pong message to it. If it is not, we check whether we have empty connections to connect him. If we have empty connections, then we add the owner of the Ping message to the neighbors table and send a Pong message to it. If we do not have any available connection slots, then we check whether the message has a TTL value greater than 1. If it does, we forward the Ping message to the other neighbors of this peer. After sending the message, the peer completes the first phase of its life cycle.
- Pong: Processing Pong messages is very similar to processing QueryHit messages. We first check whether the peer is the owner of the Ping message. If the peer is the owner, it again checks whether the owner of the Pong

message is a neighboring peer. If it is a neighbor, then we update its timeout value in the neighbors table. If it is not a neighbor, we add this peer to our neighbors list if we have available connections. If this peer is not the owner of the Ping message, then it routes the message to the owner of the Ping message with the help of information in its routing table. After sending the message back, the peer completes the first phase of its life cycle.

The second phase of a peer's life cycle - Pinging: In this phase the peer checks whether its next ping time (PINGFREQ) has expired. If it is over, the peer sends a new ping message to its neighbors and sets its next ping time by adding PINGFREQ to current time.

The third phase of a peer's life cycle - Download Request Messages Processing: The peer calls timed_receive function of CSIM library to receive the message if there are any in the TCP mailbox. If there is no messages in the TCP mailbox, the peer passes this phase and starts the fourth phase. If there is a message in the mailbox, and the peer receives it successfully, we process the message according to its type.

- Download Request: We wait MESSAGE_PROCESSING_TIME to pass at the beginning of processing Download Requests. Then, we check whether the peer has reached its upload capacity. If it has, then it replies this request with NO_UPLOAD_CAPACITY as the reply type. If the peer has not reached the upload capacity and the requested file is actually among its files, then it adds this download to its download list and sets the timeout value to current time + DOWNLOADTIME. After that it sends a reply to the request with DOWNLOAD_ACKNOWLEDGED as the reply type. If the requested file is not in this peer's files, then it replies the request with WRONG_FILE_ID as the reply type.
- Download Request Reply: We process download request replies according to their reply types. If the reply type is DOWNLOAD_ACKNOWLEDGED, we remove all peers in the download list, we set the attemptNumber to 0, we reset our query file ID, last query time and set next Query time by adding DOWNLOADTIME and exponential of peer type's queryGenerationMean to current clock. If the peer's type allows replication, we replicate the file by adding this file to our files list. We increment all downloads by one and increment the peer's type's downloads by one. If the reply type is not DOWNLOAD_ACKNOWLEDGED, we first check whether we have other peers in our download list and we have not exceeded the maximum attempt number. If there are other peers in the list and we have not exceeded the maximum attempt number, then we select a new peer and send Download request to that peer. Otherwise, we set attemptNumber to 0, remove all peers in the download list, reset the query file ID and the last query time and set the next query time by adding the peer's query generation time mean to current clock.

The fourth phase of a peer's life cycle - Download Request: At the beginning of this phase, we first check if we need to make a download request. If there is not an active query, then we should not make a download request. Similarly, if there is an active request, but HIT_WAIT_TIME has not passed yet or there are not enough hits to reach SATISFIED_QUERY_HITS, then we do not make a download request and exit this phase. Otherwise, we need to make a download request. We first check if we have requested any file before. If we requested any file before, we do not make a new download request and wait for the reply for this file. If it is the first request, than we check if there is any peer in the download list. If there are not any files, then we reset our query file ID, reset the last query time and set the next query time by adding peer type's query generation mean value to current clock. If there is a file, then we send a download request to the first peer in the download list.

The fifth phase of a peer's life cycle - Query Generation: At next query time of the peer, the peer starts the query generation phase. First, next query time is set to -1, in order to be sure that this phase is done only once for a query. Then, we increment the query sequence number of the peer and select a random file which this peer does not own. After determining the file to search, we wait MESSAGE_PROCESSING_TIME to pass and send the query to all neighbors. We increment all query counters by one. We set the peer's query file ID to the requested file and set the last query time to current clock.

V. Performance Result Report and Trace File

We prepare a custom report for observing metrics defined in Section II-D. The report consists of a list for parameters and their corresponding values, observed metric values (including download, upload, P2P messages, unsuccessful downloads), and information about free riders and content distribution. A sample output is provided in Appendix A. Other observations can be made by using these facts such as query hit ratio, unsuccessful download ratio, etc.

We have also produced a trace file to be able to understand and control the simulation run time flow. The trace file is generated for a specific peer, Peer-To-Trace (PTT). At each important phase of the simulation, the output is written to the file. Furthermore any error captured by the simulation is written to the trace file. The behavior of PTT can be monitored using this file. A sample trace file is presented in Appendix B.

VI. Conclusion

In this work, we have designed and implemented a simulation tool to model an unstructured P2P network such as Gnutella network. We have simulated Gnutella protocol in details. Our model is sophisticated enough to observe and compare the performance of a P2P network under various parameters and their different values.

We are currently extending the simulation program to implement and evaluate a mechanism to be used against free riding. We aim to compare our proposed system with existing networks using this simulation tool.

Acknowledgement: The authors would like to thank Ahmet Murat Özdemiray and Eyüphan Bulut for their invaluable contributions for developing initial codes of the simulation program.

REFERENCES

[1] Eytan Adar and Bernardo A. Huberman, “Free Riding on Gnutella”, “http://www.firstmonday.dk/issues/issue5_10/adar”, 2000.

[2] Evangelos P. Markatos, “Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella”, Proceedings of the second IEEE International Symposium on Cluster Computing and the Grid, 65-74, May 2002.

[3] Lakshmish Ramaswamy and Ling Liu, “Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems”, 36th Annual Hawaii International Conference on System Sciences (HICSS’03), - Track7,Big Island, Hawaii, January, 2003.

[4] Karl Aberer and Manfred Hauswirth, “Peer-to-Peer Information Systems: Concepts and Models, State-of-the-Art, and Future Systems”, 18th International Conference on Data Engineering (ICDE),

[5] Karl Aberer and Manfred Hauswirth, “An Overview of Peer-to-Peer Information Systems”, WDAS, 2002.

[6] M. Jovanovic and F.S. Annexstein and K.A. Berman, “Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella”, Technical Report, University of Cincinnati, 2001.

[7] Jordan Ritter, “Why Gnutella Can not Scale. No, Really”, “http://www.darkridge.com/ jpr5/doc/gnutella.html”, February, 2001.

[8] Matei Ripeanu and Ian Foster and Adriana Iamnitchi, “Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design”, IEEE Internet Computing, Journal special issue on peer-to-peer networking, Volume.6, 2002

[9] Stefan Saroiu and P. Krishna Gummadi and Steven D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems”, Proceedings of the Multimedia Computing and Networking, January, 2002.

[10] Krishna P. Gummadi and Richard J. Dunn and Stefan Saroiu and Steven D. Gribble and Henry M. Levy and John Zahorjan, “Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload”, Proceedings of the 19th ACM Symposium on Operating Systems Principles, (SOSP-19), October, 2003.

[11] Ramayya Krishnan and Michael D. Smith and Zhulei Tang and Rahul Telang, “The Impact of Free-Riding on Peer-to-Peer Networks”, Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS’04) - Track 7, January, 2004.

[12] Makoto Iguchi and Masayuki Terada and Ko Fujimura, “Managing Resource and Servent Reputation in P2P Networks”, Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS’04) - Track 7, January, 2004.

[13] Philippe Golle and Kevin Leyton-Brown and Ilya Mironov, “Incentives for Sharing in Peer-to-Peer Networks”, Proceedings of the Electronic Commerce’01, 2001.

[14] Vivek Vishnumurthy and Sangeeth Chandrakumar and Emin Gun Sirer, “KARMA: A Secure Economic Framework for P2P Resource Sharing”, Proceedings of the Workshop on the Economics of Peer-to-Peer Systems, June, 2003.

[15] Sepandar D. Kamvar and Mario T. Schlosser and Hector Garcia-Molina, “Addressing the Non-Cooperation Problem in Competitive P2P Networks”, First Workshop on Economics of P2P Systems, June 2003.

[16] Jeff Shneidman and David Parkes, “Rationality and Self Interest in Peer to Peer Networks”, Proceedings of the IPTPS Workshop, February 2003.

[17] Ares Web Site, “http://www.aresgalaxy.org”, 2004.

[18] Atip Asvanund and Karen Clay and Ramayya Krishnan and Michael Smith, “An Emprical Analysis of Network Externalities in Peer-To-Peer Music Sharing Networks”, Proceedings of the 23rd International Conference on Information Systems (ICIS), pp. 15-18, December, 2002.

[19] Herb Schwetman, “CSIM: A C-based, process oriented simulation language”, Proceedings of the 1991 Winter Simulation Conference, pp. 387-396, 1991.

[20] eDonkey Web Site, “http://www.edonkey2000.com”, 2004.

[21] eMule Web Site, “http://www.emule-project.net”, 2004.

[22] FreeNet Web Site, “http://www.freenet.com”, 2004.

[23] N.S. Glance and B.A. Huberman, “Dynamics of Social Dilemmas”, Scientific American, March, 1994.

[24] Clip2, “The Gnutella Protocol Specification v0.4 (Document Revision 1.2)”, “http://www9.limewire.com/developer/gnutella protocol0.4.pdf”, June, 2001.

[25] B.A. Huberman and N.S. Glance, “Beliefs and Cooperation”, Modeling Rational and Moral Agents, Oxford University Press, pages 210-235, 1996.

[26] B.A. Huberman and R.Lukose, “Social Dilemmas and Internet Congestion”, Science, V. 277, March, 1997.

[27] iMesh Web Site, “http://www.imesh.com”, 2004.

[28] Kazaa Web Site, “htpp://www.kazaa.com”, 2004.

[29] Kazaa Lite Web Site, “http://www.k-lite.tk”, 2004

[30] LimeWire Web Site, “http://www.limewire.com”, 2004

[31] MP2P Web Site, “http://www.mp2p.net”, 2004

[32] Leander Kahney, “Cheaters bow to peer pressure”, “http://www9.wired.com/news/tecnology/0,1282,41838,00.html”, 2001.

[33] M. Karakaya and I. Korpeoglu and O. Ulusoy, “A Distributed and Measurement-Based Framework Against Free Riding in Peer-to-Peer Networks”, Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P’04), August 2004.

[34] Nazareno Andrade and Francisco Brasileiro and Walfredo Cirne and Miranda Mowbray, “Discouraging Free-riding in a Peer-to-Peer Grid”, Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC13), June, 2004.

[35] Nazareno Andrade and Miranda Mowbray and Walfredo Cirne and Francisco Brasileiro, “When Can an Autonomous Reputation Scheme Discourage Free-riding in a Peer-to-Peer System”, Proceedings of the 4th Workshop on Global and Peer-to-Peer Computing(GP2PC), April 2004.

Appendix A: Sample Report

```
   C++/CSIM Simulation Report (Version 18.3 for MSVC++ V4.0)

   Fri Nov 05 21:10:55 2004

   Ending simulation time: 0.000
Elapsed simulation time: 0.000
CPU time used (seconds): 0.000

   *************** PARAMETER VALUES ***************


MESH_EDGE                    20
SIMTIME                      1000
MESSAGE_PROCESSING_TIME      0.10000
PINGFREQ                     20.000  HIT_WAIT_TIME              5.000
DOWNLOAD_TIME                60.000  SATISFIED_QUERY_HIT        3
TIME_TO_LIVE                 7       MAX_DL_ATTEMPT_NUMBER      3
DISTINCT_FILES               4000    COPY                       2
RANGE                        0       SKEWNESS                   0
TOTAL_FILES                  8000

NUM_OF_PEER_TYPES            3
POPULATION_RATIOS            0.10    0.20                       0.70
SHARED_FILE_RATIOS           0.10    0.19                       0.71
NO_OF_MAXIMUM_UPLOADS        3       3                          3
QUERY_GENERATION_MEAN        60.00   60.00                      60.00
CONSUMER_QUERY_GEN_MEAN      30.00
REPLICATION                  1       1                          0
FREE_RIDING_TYPE             NONE    NONE                       MIXED




   *************** OBTAINED RESULTS ***************
C++/CSIM Simulation Report (Version 18.3 for MSVC++ V4.0)

   Fri Nov 05 21:12:53 2004
   Ending simulation time: 1000.100
Elapsed simulation time: 1000.100
CPU time used (seconds): 117.825

   *************** QUERIES ********************
The number of queries of all peers is : 6505
The number of queries of type A peers is : 544 0.08
The number of queries of type B peers is : 1096 0.17
The number of queries of type C peers is : 4865 0.75
*************** DOWNLOADS ********************
The number of downloads done by all peers is : 836
The number of downloads done by type A peers is : 71 0.08
The number of downloads done by type B peers is : 142 0.17
The number of downloads done by type C peers is : 623 0.75
*************** UPLOADS ********************
The number of uploads done by all peers is : 836
The number of uploads done by type A peers is : 95 0.11
The number of uploads done by type B peers is : 178 0.21
The number of uploads done by type C peers is : 563 0.67
*********** UNSUCCESSFULL DOWNLOADS ************
The number of unsucessful downloads done by all peers is : 80
The number of unsucessful downloads done by type A peers is : 5 0.06
The number of unsucessful downloads done by type B peers is : 21 0.26
The number of unsucessful downloads done by type C peers is : 54 0.68
**************** PING PONG MESSAGES **************
The number of ping pong messages in the network is : 150675
**************** P2P PROTOCOL MESSAGES **********************
total messages send :1066169
total messages received :1065872
****************************************************************
FREE RIDING TYPES AND POPULATION
****************************************************************
consumers :105
droppers :77
NON-contributors :98
noneFR :120
****************************************************************
PEERS AND FILES
****************************************************************
peers_having_zero_file :175 0.44
peers_having_some_file (¡=10) :175 0.44
```

Appendix B: Trace File

```
SIM STARTED!
init started!
init_tables() started!
init_NeighboursTable() started!
reading file started!
NUMBER OF PEERS READ FROM neighbourTable20.txt: 400
init_NeighboursTable() ended!
init_FRLT() started!
init_FRLT() ended!
init_tables() ended!
initPeerTypes() started!
initPeerTypes() ended!
init_Stats() started!
init_Stats() ended!
identication of peer types started!
PEER TYPES AND THEIR MEMBERS
PEER TYPE 0 : 205 70 123 213 ...
PEER TYPE 1 : 391 50 85 ...
PEER TYPE 2 : 0 1 2 3 4 5 ...
FREE RIDING TYPES OF ALL PEERS
PeerId: 0 FreeRiding Type: DROPPER
PeerId: 1 FreeRiding Type: CONSUMER
PeerId: 2 FreeRiding Type: CONSUMER
...........
...........
identication of peer types ended!
identication of peer file quotas started!
peer type 0 [num of peers:40]
equallyDistributed 800
remainders 0
distributed 0
totalDistributed 8000
peer type 1 [num of peers:80]
equallyDistributed 1520
remainders 0
distributed 0
totalDistributed 8000
peer type 2 [num of peers:280]
equallyDistributed 5600
remainders 80
distributed 0
totalDistributed 8000
identication of peer file quotas ended!
distributing files to peers started!
distribute files out of skewness range started!
distributing files to peers ended!
transfer files between peers started!
The cycle 6 begins at: 1.2 checking for new message at 1.2 (number of msg in mabox 0)
```

```
********** * Message came at 1.3 * **********
********** * Message Info * **********
Message type (txt): QUERY
QueryId: 200278017
Query Sequence no.: 1
Owner of the Query: 191
Routing PeerId: 191
Replying PeerId: -1
TTL: 7
file ID: 2954
current msg number at mailbox : 0
********** * Message Processing Begins* **********
I forward this message to my neighbour: 210
I forward this message to my neighbour: 212
I forward this message to my neighbour: 231
********** * Message Processing Ends at 1.4 * **********
The cycle 6 ends at: 1.5
.........
.........
checking for new message at 999.9 (number of msg in mabox 0)
NO new message at 1000
The cycle 5036 ends at: 1000.1
************ Files at the end of simulation *************
Total Number of Files:0
*********** PEER LIFE CYCLE ENDS **************
SIM ENDED!
***************************************
```

Appendix C: Topology File

```
100
0 1
0 10
1 2
1 11
2 3
2 12
3 4
3 13
4 5
4 14
......
......
```

Appendix D: parameter.h File

```
/*****************************************************************

This header contains information about the parameters used in
simulation system.
The names of the parameters are tried to be chosen carefully
for easy understanding of the function of the parameters. However
the comments are also written to explain the meanings of the
parameters.

*****************************************************************
/

#ifndef _PARAMETERS_H
#define _PARAMETERS_H
```

```
//******************************************************************
// Since the system is designed as a medge with square structure
// at the beginning of simulation, we should define how many peer
// will be in one edge of the medge.
//******************************************************************
#define MESH_EDGE 20
const int NUM_PEERS=MESH_EDGE * MESH_EDGE;
//******************************************************************
// The total simulation run time. The program ends when the clock
// of CSIM reachs to that time.
//******************************************************************
#define SIMTIME 1000.0

//******************************************************************
// During the life time of a peer,it does several operations.
// It processes messages, it waits for a while between two
// consecutive generated queries, it waits for a specific time for
// HIT messages of its queries and it downloads files from other
// peers for some time.
//
// Below these parameters are specified and set some numbers
//******************************************************************

   // The time passing while a peer processes a message
#define MESSAGE_PROCESSING_TIME 0.1

   // Timeout value for message receiving
#define MESSAGE_WAITING_TIMEOUT 0.1

   // The time for keeping information about a message: when timeout occurs
// all information about that message is deleted from routing table
#define MESSAGE_STORING_TIMEOUT 5.0



   //******************************************************************
// The time frequency for a peer to ping its neighbours
//******************************************************************

   #define PINGFREQ 20.0

   // The time for a peer to wait the QUERY HIT messages arrive to
// itself. After that time elapses, download begins.
#define HIT_WAIT_TIME 5.0

   // The time to download files. Download times for all peers is the
// same.
#define DOWNLOAD_TIME 60.0

   //******************************************************************
// When a peer broadcasts a query, it starts to wait for query hits
// The time for waitng is defined either by the number of QUERY HITS
// arrived to peer or by the HIT_WAIT_TIME.
// Normally a peer intends to wait the HIT messages during a
// HIT_WAIT_TIME after it has done the query. However if the number
// of QUERY HIT messages reachs the value SATISFIED_QUERY_HIT
// it stops waiting the HIT messages and starts to download.
//******************************************************************
#define SATISFIED_QUERY_HIT 3

   //******************************************************************
// The peers in the system share some files among possible number
// of files. The total number of distinct files in the system is
// defined by parameter NO_OF_DISTINCT_FILES
//******************************************************************

   #define NO_OF_DISTINCT_FILES (NUM_PEERS*10)

   #define FILE_DISTRIBUTION_REPLICATION 2
   #define FILE_DISTRIBUTION_REPLICATION_SKEWNESS_RANGE ((NO_OF_DISTINCT_FILES /100)*5)

   #define FILE_DISTRIBUTION_REPLICATION_SKEWNESS_RANGE 0

   #define FILE_DISTRIBUTION_REPLICATION_SKEWNESS 0

   #define TOTAL_NO_OF_FILES (int)(NO_OF_DISTINCT_FILES *
FILE_DISTRIBUTION_REPLICATION+
FILE_DISTRIBUTION_REPLICATION_SKEWNESS_RANGE *
FILE_DISTRIBUTION_REPLICATION_SKEWNESS)

   //*******************************************************************
// In Gnutella, the number of hops that a message can go over is
// defined by a TIME_TO_LIVE(TTL) value. In every hop the TTL of
// message is decreased by one.
//*******************************************************************
#define TIME_TO_LIVE 7
//*******************************************************************
// When a peer starts to download files, it may be refused by the
// peers that it applied to download the specific file from. At
// that moments it tries to download the file from another peer
// that it take QUERY HIT from. The number of attempts for a peer
// to download a file is defined by the parameter below.
//*******************************************************************
#define MAX_DL_ATTEMPT_NUMBER 3

   //*******************************************************************

   #define NUM_OF_PEER_TYPES 3

   //**************************************************************
// Free riding types
//**************************************************************
#define NONE 0
#define NON_CONTRIBUTOR 1
#define CONSUMER 2
#define DROPPER 3

   #define MIXED 4
//peers are equally distributed to each seven free riding types

   #define NUM_OF_FR_TYPES 3

   #define CONSUMER_QUERY_GENERATION_MEAN 30.0
#define POPULATION_RATIOS  0.1 , 0.2 , 0.7
#define SHARED_FILE_RATIOS  0.1, 0.19 , 0.71
#define NO_OF_MAXIMUM_UPLOADS  3 , 3 , 3
#define QUERY_GENERATION_MEAN  60.0 , 60.0 , 60.0
#define REPLICATION  true, true , false
#define FREE_RIDING_TYPE  NONE, NONE, MIXED

   #define SIMULATION_REPORT_FILE_NAME "REPORT.txt"
#define SIMULATION_TRACE_FILE_NAME "TRACE.txt"

   #define NEIGHBOURS_TABLE_SETUP_FILE_NAME "neighbourTable20.txt"
//*******************************************************************
// The peers in the system communicate eachother using messages
// There are different types of messages acccording to the Gnutella
// structure. Query, RoutedQuery, QueryHit and DownloadRequest, Ping, Pong
//*******************************************************************
#define QUERY 1
#define QUERY_HIT 2
#define DOWNLOAD_REQUEST 3
#define DOWNLOAD_REQUEST_REPLY 4
#define PING 5
#define PONG 6

   //**************************************************************
// Download request reply messages
```

```
//***********************************************************
#define DOWNLOAD_ACKNOWLEDGED 99
#define WRONG_FILE_ID 199
#define NO_UPLOAD_CAPACITY 299

  ;

  #endif
```