Zeta: A Novel Network Coding Based P2P Downloading Protocol

Candidate: Wang Jing

Major: Communication and Information System

Supervisor: Associate Professor. Huang Jiaqing

Huazhong University of Science and Technology

Wuhan, Hubei 430074, P. R. China

June, 2010

摘要

许多现有的对等下载协议（如 BT 协议中）中都存在着一个问题：当文件下载接近完成的时候，下载的速度会大幅降低。这主要是因为拥有一些关键文件块的节点可能较早的离开了这个网络导致某些文件块在网络中稀缺，大部分节点因为缺乏这些文件块而无法完成最终的下载。着篇文章中提出了一种全新的 p2p 下载协议，它使用网络编码解决了这样的问题。

和 BT 协议相比，Zeta 协议中的节点不仅仅是单纯的转发数据，而是会事先将已有的数据进行线性组合，然后发送出去。每一个线性组合中包含有所有文件块的信息，所以不会出现某些关键分块的丢失影响整个文件下载的情况。当接受节点接收到了足够多的线性组合，它就能够重新生成原来的数据。

微软剑桥研究院曾经提出了一个名为 Avalanche 的类似的协议。这个协议结合了 BT 协议和网络编码的优点。但是 Avalanche 备受争议并被指责为低效的和不实用的。 Zeta 协议在很大程度上面借鉴了 Avalanche 协议的一些思想。与 Avalanche 不同的是，Zeta 引入了许多机制来增加协议的效率，使得协议更加的实用。第一，Zeta 协议中使用了更加简洁的消息交互机制。第二，Zeta 协议使用改进的 UDP 协议而不是 TCP 协议来减少维持连接所需要的额外成本。 第三，Zeta 协议使用完全的 PUSH 机制来进一步减少对等节点之间不必要的信息交互。第四，Zeta 协议使用了渐进式的解码方式来缓解网络编码计算量过大的问题。

为了方便后来的研究者对 Zeta 协议的仿真，我们开发了一套基于 NS2 的 Zeta 协议仿真工具包。这个名为 ZetaSim 的仿真工具包大大的方便了研究者大家基于 Zeta 协议的仿真环境的工作量。值得一提的是，ZetaSim 良好的结构设计使得它具有较高的灵活性和可移植性。基于 ZetaSim，本文做了一系列的仿真实验并得出了一些有用的结论。

Abstract

Many existing peer-peer downloading protocols suffer from lose-key-block problem, that the lose of key block paralyze the whole network. A novel p2p downloading protocol named Zeta is proposed in this paper which aims to solve this problem by introducing network coding techniques.

Compared with BT protocol, nodes in Zeta Protocol combines existing blocks before sending new data. When the receiver has got enough linearly independently blocks it will be able to re-generate the original data. This techniques make the network more robust with the lose of key block.

Microsoft Research once proposed a similar protocol named Avalanche which combines the advantages the BT protocol and network coding. However, Avalanche has been criticized as low efficient and impractical.

Zeta protocol is based on Avalanche and we improve it from the following aspects. First, a better and more efficient message exchange scheme is proposed. Second, Zeta uses UDP instead of TCP as transportation protocol to reduce overhead. Third, Zeta introduces PUSH scheme to reduce the unnecessary data exchange thus to increase the transmission efficiency. Last, Zeta protocol uses progressive decoding method to reduce the calculation overhead of network coding.

To facilitate the simulation of Zeta Protocol, we develop ZetaSim, a friendly and scalable simulation suite for NS2. Based on the architecture of GnutellaSim, ZetaSim is well structured and portable. With little exchange it can be easily to be recoded as a real software.

After a large number of simulation in both regular network and real network structure, we prove the efficiency of Zeta Protocol and give some suggestions on setting parameters.

Table of Contents

1 Introduction

The peer-to-peer (P2P) system is a novel approach for people to share data files and computing resources. As many P2P based applications, like BitTorrent and PPlive, have been widely deployed, the research of p2p system becomes a hot spot in recent years.

BitTorrent[**1**] is one of the most successful P2P downloading protocol till now. It has been estimated that it accounts for more than 27% of all Internet Traffic.

The most important feature of BT protocol is to cut the original files into many pieces. User downloads each piece from different sources. Because of it, BT users will enjoy higher downloading speed when the number of downloaders increases. However, BT users often suffer from a harassing problem that they need to wait for a long time to download the last few pieces. Besides, when the number of receivers increases, it becomes harder to do optimal scheduling of pieces. To avoid the calculation overhead of optimization, The existing BT protocol uses heuristic methods to get sub-optimal solutions, which lead to slower download speed.

Random Network Coding[**7**] has been proved as an effective way to solve this problem. Instead of sending pieces directly to other peers, senders produce linear combination of the blocks they already had first. After receiving enough linear combinations, a peer can decode the entire file from these combinations. The main advantage is that peers don't need to find specific pieces in the systems, since each linear combination contains the information of the entire files. Although Avalanche has introduced Network Coding to solve the problems of BT protocol. It doen't take full advantages of Network Coding and ignores some important features of random network coding. As a result, Avalanche is unsatisfying in many senses and satirized by Bram Cohen, the inventor of BT protocol, as a “vaporware”.

Based on the pioneering work of Avalanche[**2**,**3**,**4**], I design a novel random network coding based p2p file distribution protocol called Zeta and develop a simulation suite in NS2 to facilitate the simulation process of Zeta

Protocol. By integrating the advantages of Network Coding and UDP protocol instead of TCP protocol, Zeta successfully reduces the overhead and improve the transmission efficiency while ensuing the high robustness of network.

Here is a brief introduction of the thesis. Chapter 2 describes the existing research work in this field. Chapter 3 is a detailed specification of Zeta Protocol. Chapter 4 deals with the architecture of ZetaSim, a NS2 based simulation suite for Zeta protocol. The last 2 chapters are the simulations results and the conclusions based on these results, respectively.

2 Previous Work

2.1 BitTorrent

BitTorrent (BT) is designed by programmer Bram Cohen in April 2001 [1]. The first release of BT client comes out in 2 July 2001. In BT protocol, each client is able to preparing, querying and transferring data files over a network. A peer is any computer running an instance of a client.

In order to share a data file, a peer need to create a small file called “torrent”, which contains the information of the file waiting to be shared and the address of the tracker, the computer that coordinates the file distribution.

Then this torrent will be put on the web server. Peers who are willing to download the the file should download the torrent file first. All the peers holding the same torrent file form a subnet.

Compared with classic downloading method with HTTP or FTP, BT has two important features:

1. Instead of downloading the entire file from one machine, BT client cuts the file into many pieces and downloads each piece from different machines through multi TCP connections.
2. Classical methods download file in a sequential way. However, BT

downloads pieces randomly or in a "rarest-first" approach to improve the robustness of the network.

These features make BT an efficient and scalable downloading protocol. However, every coin has two sides, they also have some drawbacks.

First, in BT protocol, peers needs to maintain many cost-roaring TCP connections to other machines. it usually takes a long time for a peer to reach the full speed. In contrast, regular downloading methods only needs to construct a TCP connection thus rise to full speed very quickly and maintains this speed throughout.

Second, in BT protocol, each pieces are downloaded independently. During the distribution process, there are many copies of each piece in the network. Each peer should decide which piece to download and form whom it require a piece. The optimization problem is very hard to solve when the size of the network is large. As a result, the existing version of BT protocol uses some heuristic rules, like "rarest first" rule, to coordinate the piece downloading process, which are quite inefficient in many cases.

2.2 Random Network Coding

Network Coding is a novel and effective mechanism to improve the utilization of network[**5**,**6**]. Instead of just transmitting packets without any change, intermediate nodes in network coding are allowed to encode packets.

Random Network Coding (RNC) is one kind of network coding proposed by T.Ho, et.al.[**7**] which using random codes. In random network coding, network nodes independently choose random coefficients and combine the blocks using these coefficients. To recover symbol, receivers requires degrees of freedom, in which case the coefficient matrix is invertible.

2.3 Avalanche

Avalanche is a research project funded by Microsoft Research Cambridge which aims to enable a cost effective, Internet scalable and fast file distribution solution[2,3,4]. It fixes the problems of BT protocol using network coding.

Instead of sending the file pieces directly, Avalanche clients generate a random combination of the network coding. The generated combination and corresponding parameters are sent together. When peers receive enough enough linear combinations, they will be able to decode the original blocks through solving the linear equations.

The main advantage of network coding is that peers don't need to find specific pieces in the system to completely download the file because every blocks contains information of the entire file. This advantage helps reduce the complexity of the coordination. Besides, in Avalanche network, no peer becomes a bottleneck since no block is more important than the other. Last, network bandwidth is fully utilized because network coding has proved as a efficient method to improve the network throughput.

The introduction of network coding makes Avalanche an effective systems as node has no need to know the information of its peers and the overhead caused by coding is negligible.

2.4 GnutellaSim

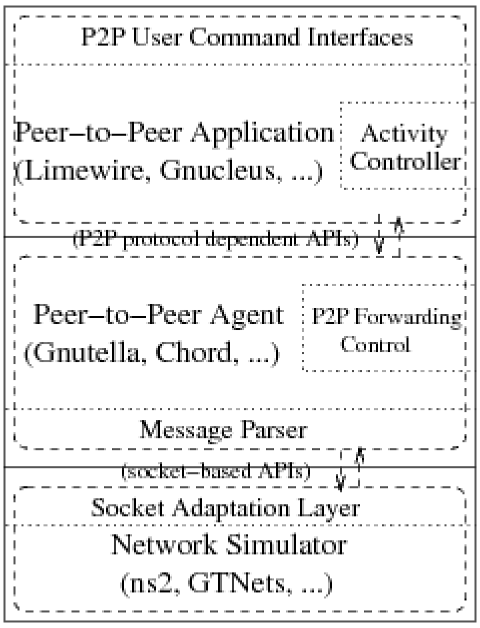

P2P User Command Interfaces
Peer–to–Peer Application (Limewire, Gnucleus, ...)
Activity Controller
(P2P protocol dependent APIs)
Peer–to–Peer Agent (Gnutella, Chord, ...)
P2P Forwarding Control
Message Parser
(socket–based APIs)
Socket Adaptation Layer
Network Simulator (ns2, GTNets, ...)

Fig. 1: Peer Simulator Architecture

GnutellaSim is a scalable packet-level simulator for Gnutella Protocol which enables evaluation of the Gnutella system[**8**,**9**,**10**]. The framework of GnutellaSim is designed to be portable among many existing network simulators, such as NS2, PDNS and PTNet. But usually GnutellaSim runs above PDNS and NS2.

Each peer in GnutellaSim consists of a 3-layer architecture shown in Fig. [1], add a group of peer nodes forms the whole p2p system. The most uppermost layer, denoted as peer-to-peer application layer, embodies the behavior of a peer-to-peer application. It also includes a activity controller which models the user behavior using model proposed in [**11**]. The corresponding class in this layer is PeerApp, PeerSys and ClassSpec. The peer-to-peer agent performs message transmission and processing. The APIs between PeerApp and PeerAgent correspond to messaged defined in p2p protocol.

GnutellaSim is designed as a flexible and portable platform. To solve the problem that existing network simulator often have different APIs, GnutellSim adds socket adaptation layer between network simulator and peer agent layers.

3 Zeta Protocol

3.1 Overview

As noted above, Avalanche has successfully solved some problems in BT through the introduction of network coding. However, there are still some harassing problems left. First, both Avalanche and BT peers must maintain a lot of TCP connections to their peers, which leads to soaring connection costs. If we restrict the number of connection to reduce the cost, peers will be not able to enjoy the uploading bandwidth of many peers at the same time.

Second, in BT protocol, a peer must send a request to its peers before it can receive a packet-so call PULL scheme, which greatly increases the delay and reduces the throughput.

To solve the first problem, Zeta protocols is designed to run above mainly on UDP protocol instead of TCP protocol, some simple techniques are introduced to control the flow rate. Besides, Zeta protocol uses PUSH scheme. Peer nodes deduce the needs of its peers and send linear combinations actively to them. Network coding techniques greatly reduce the complexity of deduction because source nodes don't need to guess which block they should transmit. With the parameters of each nodes, it is easy for source nodes to decide to which node they should send combinations.

3.2 Network Coding Techniques

To make network coding possible, we divide each files into several generations, while each generation is then divided in several blocks. A block consists of many code words. Fig. 2 Illustrates the relationship between them.

Suppose the size of Code Word, block, generation, and file are S_w , S_b , S_g

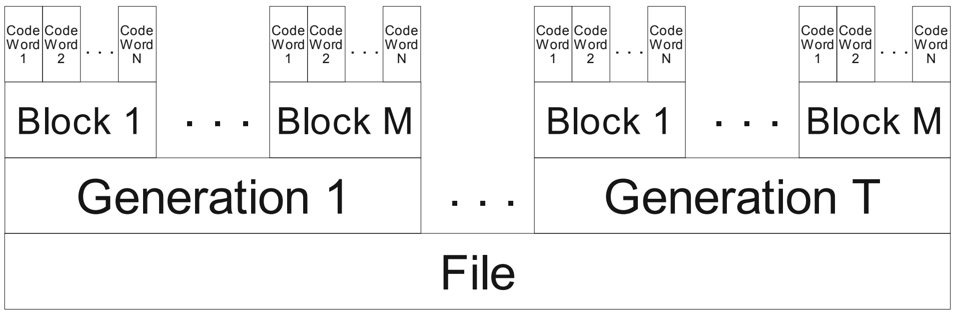


Fig. 2: The Strucutre of File

and S_f respectively. There are N code words in each block, M blocks in each generation and T generations in each file.

Blocks can only be combined with other blocks in the same generation, that is to say, any two combinations between two different generations are independent. When it is time for a peer node to send a packet to other nodes , it produces a combination all the existing blocks it holds in that generation. The operation can be best illustrated in Fig. 3

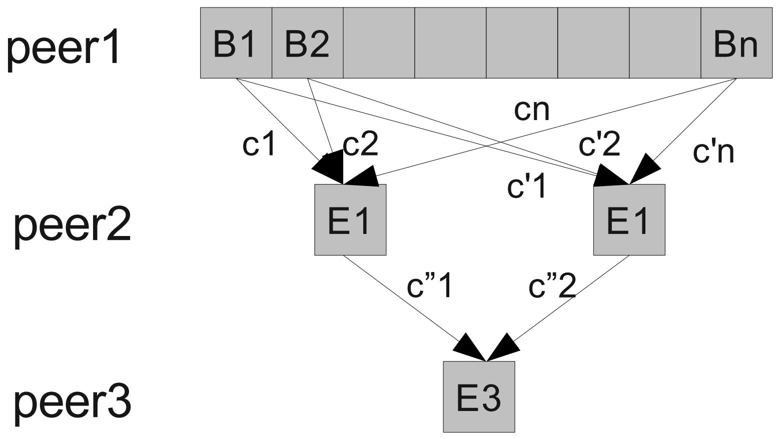


Fig. 3: Illustration of Network Coding System

Because of the independence between different generations, we only consider one generation to simplify the process. Suppose peer 1 is a seeder who hold all the blocks of this generation. When it sends a combinations to peer 2, it produce a linear combination E1 as follows. First, it picks random coefficients $c_1, c_2, ..., c_n$, the size of the each coefficient is equal to code word size. It then multiply each element of block i with c_i , and finally add the results together. All the operations take place in a finite field. It is possible that two nodes picks the same set of nodes and construct the same block coincidently. If the field is very small, such kind of collision may reduce the performance of the system. A field size of 2^{16} is enough in most cases. Please refer [7] for more discussion of the impact of the field size.

Peer 1 will send the calculate linear combination as well as the coefficient vector $\vec{c} = (c_i)$ to peer 2. Suppose when peer2 becomes a seeder and start to send data, it hold another block E2. It will choose two coefficients $c_1"$ and $c_2"$ randomly and combines E1 and E2 together. The result combination E3 equals $c_1" E_1 + c_2" E_2$, the new coefficient vector $\vec{c"}$ becomes $c_1^{"} \cdot \vec{c} + c_2^{"} \cdot \vec{c'}$.

Network Coding brings many benefits to the network. Without network coding, each peer needs to decide which block to receive and from whom it should request this block. However, with network coding, such kind of decision becomes unnecessary, the combination contains the information of all the blocks that sender holds and thus is useful to receivers in most cases. The only situation that the combined block is useless is that the same block has been produced in other places. According to [7], the probability is quite small.

3.2.1 The Specification of Parameters

It is easy to determine the code word size S_w become each code word should have the same number of bits as finite field size. Suppose the finite size is w , then $S_w = w/8$. Usually w can be 8, 16, 32.

Next we will deduce the maximum size of a generation if we want to enclose block data and corresponding coefficients in to a single packet. As defined in last section, the block size is S_b , the generation size is S_g . So there are $M = S_g / S_b$ blocks in each generation. The size of each coefficient is equal to code word size S_w , so the total size of coefficient vector is :

$$S_{cv} = M \times S_w = S_g S_w / S_b$$

Suppose the maximum length of a single in this network is MTU. If we want to enclose block data and corresponding coefficient vector in the same packet to avoid packet slicing in lower layer, the follow restriction must be satisfied.

$$S_{cv} + S_b \leq MTU$$

$$S_g S_w / S_b + S_b < MTU$$

We can transform it as:

$$S_g = (MTU - S_b) \times S_b / S_w$$

When $S_b = MTU/2$, S_g is maximized as

$$S_g^{max} = \frac{MTU^2}{2S_w}$$

If MTU = 1500 Bytes, the most common value in the Internet, and field size is 8, then

$$S_g^{max} = 1.125MB$$

If we want to enclose the block data and coefficient vector into the same packet, which is the most efficient method, those files larger than this value must be divided into several generations.

3.3 Node Type

There are three kinds of nodes in a P2P network—boot server, tracker and peer node. Boot server should always be open and it stores and manages the information of Tracker.

In BT network, the tracker address of a shared file is recorded in torrent file. This scheme has two disadvantages. First, the distribution of torrent files is through web server and it is quite hard for users to share and query a file. Second, the relationship between file and tracker is fixed and all the torrent files will become invalid if a tracker server is closed. That's the reason why there are so many invalid torrent files in the Internet.

Boot server incorporates the functionality of web server and torrent files. One the one hand, when a node connects to the network, it will register all the files it is willing to share automatically in boot server. This information will also be periodically updated. Users who wants to share a new file are no longer required to build a torrent file and upload it to a web server. Besides, boot server also maintains the corresponding relationship between a file and a tracker. If a tracker is closed, boot server will assign a new tracker for this file automatically.

Tracker in Zeta protocol has some differences with its counterpart in BT protocol. In Zeta protocol, tracker is dynamically assigned by boot server and one tracker is usually in charge of only one file. Tracker stores the address set of all peers who are downloading this file. Peers will periodically update its information in tracker. A peer node can be tracker and downloader at the same time.

We refer those peer nodes who are downloading file A as the downloaders and those who are sending blocks to other peers as seeders. The tracker of file A maintains the data structure which records who are seeders and who are downloaders.

A peer node can be downloader and seeder at the same time. So in the subnet of file A, there are three kinds of peer nodes---1. pure downloader, 2. pure seeder, 3. downloader & seeder.

When a peer node starts downloading, it is a pure downloader. After a while, it will receive enough packets and start seeding for others. As a result, it is downloader as well as seeder at this stage. When it finishs downloading, it will becomes a pure seeder.

3.4 Message Type

Usually a message is enclosed in a Zeta packet. Each Zeta packet consists of two parts: 1. packet header and 2. extend information.

The format of header is as follows:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|_________________________Node ID_____________________________|
|  Msg   Type |             (for Extend Use)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4: Packet Header in Zeta Protocol

The first 4 bytes is the Node ID of the peer node. The following byte represents the message type. One number corresponds to a message and receiver will parse the packet according to this byte. The current version of Zeta Protocol has 16 messages, so only 4 bits are utilized. The rest 3 bytes are reserved for extend use.

Since different messages have different extend information, we will illustrate them one by one in the following parts. As noted above, there are 16 types of messages, which can be categorized as 3 groups—messages sent by boot servers, messages sent by tracker and messages sent by peer node. The following table is the list of all messages:

Table 1: List of Other Messages

Messages Sent By Boot Servers		
UPDATE_SHARED_FILES_ACK,	QUERY_NOT_HIT,	QUERY_HIT
REQUEST_START_SEEDER,	SELECT_TRACKER,	
Messages Sent By Tracker		
SELECT_TRACKER_ACK	CONNECT_TRACKER_ACK	UPDATE_PEERS_REPLY
Messages Sent By Average Peers		
CONNECT_BOOT_SERVER	CONNECT_BOOT_SERVER_REPLY,	UPDATE_SHARED_FILES,
QUERY	CONNECT_TRACKER	HAS_FINISHED_DOWNLOADING,
UPDATE_PEERS	FILE_TRANS	

3.5 Boot Process

This section illustrates what happens during file distribution using Zeta protocol. The distribution process consists of 3 parts—boot, query and file distribution.

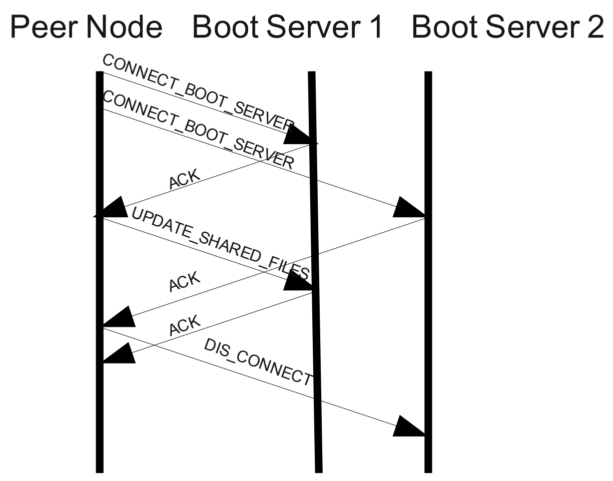


Fig. 5: The Description of Boot Process

Boot process is described as inf Fig. 5. Usually a Zeta system has a lot of boot servers, the address of these well known boot servers are cached by peer nodes. Any peer who are willing to join in the network will send CONNECT_BOOT_SERVER message to the cached boot server one by one.

When this peer receive a CONNECT_BOOT_SERVER_ACK message from a boot server, it will set state as ONLINE and that boot server as its father boot server. If it receives CONNECT_BOOT_SERVER_ACK messages from other boot server later, it will then send a DIS_CONNECT_BOOT_SERVER message back.

Then the peer node will scan its shared folder and update its file list waiting to be shared. After receiving CONNECT_BOOT_SERVER_ACK message from its father boot server, it will send back a UPDATE_SHARED_FILES message. Then its father boot server will update the database according to this message and send a UPDATE_SHARED_FILES_ACK back. Till now, peer node successfully join in to network and will be able to query files.

Father boot server contains the shared file list of its child peer nodes. And different boot servers periodically exchange information with each other.

3.6 Query Process

After booting successfully, Zeta client will wait the query from users. Then it will send a query message to its father boot server. As a noted above, each boot server maintains a database which stores the corresponding relationship between peer node and its shared files.

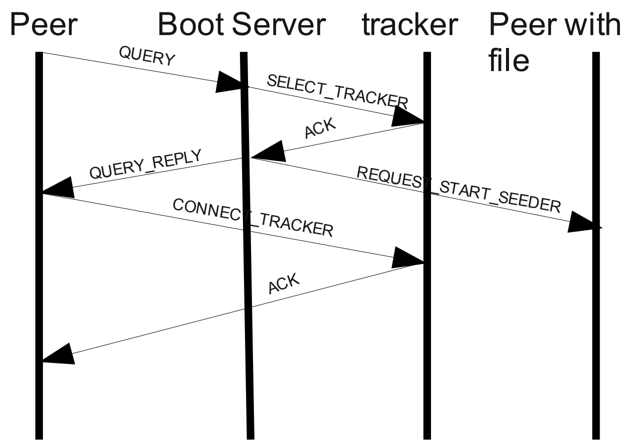


Fig. 6: Query Process

In responding to this query message, boot server will check whether there is a existing tracker server for this file. It there is, the boot server will return the address of the boot server. Otherwise, the boot server will select a tracker for this file.

There are many criterion of choosing tracker. First, tracker should have good network condition because it is the center of the subnet of this file from which all the seeders and downloaders periodically request information. Second, the average stay time of tracker should be long. If a tracker is closed down, boot server will have to select a new tracker for this subnet, which brings extra overhead. Last, it is highly recommended, but not required, for the tracker to have a full copy of the file waiting to be distributed. If the number of seeders is small in this subnet, trackers will help seed the file and accelerate the distribution process.

The tracker receiving SELECT_TRACKER message will send a SELECT_TRACKER_ACK message as a response. After a tracker has been successfully selected, the boot server will send REQUEST_START_SEEDER message to all nodes who has registered this file as their shared file before.

3.7 File Distribution

3.7.1 The Role of Tracker

During the initialization process , a sender sends CONNECT_TRACKER message to its tracker. The tracker will register this node with some parameters contained in the message, including max upload bandwidth of this seeder and the max number of downloaders it is willing to seed, and send a CONECT_TRACKER_ACK back.

After receiving the ACK from tracker, it will send UPDATE_PEERS message to tracker to request information of the downloading peers periodically. Tracker will return a proper number of downloaders according to the parameters of this seeder.

The structure of UPDATE_PEERS message and UPDATE_PEERS_ACK information can be illustrate in Table 2.

Table 2: Information in Message between Tracker and Seeder

UPDATE_PEERS	UPDATE_PEERS_ACK
Max upload bandwidth	The address set of downloaders
Max downloaders it will seed	Max download speed of downloaders

At the same time, downloaders also send UPDATE_PEERS message to tracker to request information of the seeders. The information consists of two parts. The first part is the address set of validated seeders in this subnet. With the information of validated seeders, downloaders will be able to reject data from vicious computers, thus is improved the network security. The second part is the sending rate of each seeder to this downloader. This information will be used in congestion control.

Table 3: Information in Message between Tracker and Downloader

UPDATE_PEERS	UPDATE_PEERS_ACK
Max download speed	The address set of Seeders
Receiving rate from each seeder	Sending rate of each seeder

Since both downloaders and seeders need to exchange information with tracker, tracker plays an important role in the distribution subnet of this file. As result, tracker is generally not involved in seeding file data. However, when the number of seeders is very small, tracker will also seed the file to accelerate the rate of file distribution.

3.7.2 PUSH v.s. PULL

Many P2P downloading protocol use PULL scheme. Only when a node receives a request from its peers, it starts to send data to the requester. Zeta protocol uses PUSH scheme, a seeder deduces the needs of its peers and sends data to them actively.

Because of this, seeders have to request information of downloaders through sending UPDATE_PEERS message to tracker periodically. With the UPDATE_PEERS_REPLY message from tracker, seeders knows who needs data.

Besides, the things downloaders need to do are simple. They first send CONNECT_TRACKER message to tracker to register themselves in the tracker. After receiving CONNECT_TRACKER_ACK from tracker, it will wait passively for the data in most of time. Although it also need to update the list of seeder through seeding UPDATE_PEERS message to tracker, the frequency of update is much lower than that of seeders.

However, PUSH scheme brings two problems. First, as Zeta Protocol is built above UDP protocol, receiver doesn't send feedback to sender after successfully receiving a packet. So sender needs to control the sending rate by itself. Second, since there are many downloaders, seeders need to distribute its upload bandwidth reasonably.

3.7.3 Flow Control Scheme

Since Zeta protocol is based on UDP protocol, if seeders send packets continuously without any flow control scheme, it is highly possible that congestion will happen.

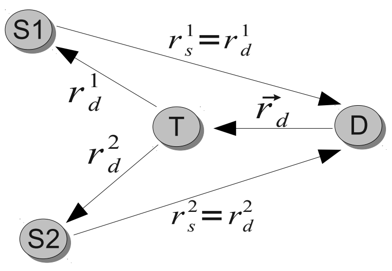


Fig. 7: The flow control in Zeta

Zeta protocol introduce a simple flow control scheme to avoid this situation. Each time when downloaders send UPDATE_PEERS message in tracker, it will place its own receiving rate there. If seeder find that its receiving rate is smaller than seeder's sending rate. It will assume that congestion happens and reduce the sending rate to that downloader. Take the simple topology in Fig. 7 as an example, S1 and S2 are two seeder, T is tracker and D is the downloader. Suppose D's receiving rate from Seeder i is r_d^i ▹ Downloader D will send its receiving rate vector $\vec{r_d} = \{r_d^i\}$ to tracker T periodically. This information will be distributed to each seeder through UPDATE_PEERS_ACK message. After seed i get r_d^i from tracker, it will check whether congestion has happened or not and update it sending rate.

Although the scheme is simple, it is enough because Zeta protocol is robust with congestion. One reason for congestion to be a serious problem in TCP is that the packet drop caused by congestion will invoke re-transmission, which in return makes the matter worse. However, all the blocks in Zeta protocol are equivalent to each other, so it is not necessary for a Zeta client to retransmit a specific packet.

3.7.4 The Process of Sending Data

It is quite important for seeder i to distribute its upload bandwidth U_i as there may be many downloaders at the same time. On one hand, it should be able to punish "free riders", those nodes who enjoy other seeders' bandwidth but don't seed for others. On the other hand, a serious problem in p2p file distribution network is the discrepancy between real network topology and overlay network. This kind of discrepancy reduce network throughput and thus must be considered.

BT uses a simple "tip for tap" technique to achieve the first goal and completely ignores the second factor. Zeta protocol make the several improvements.

3.7.4.1 When to Send Packet?

As shown in section 3.2.1 , the size of each packet :

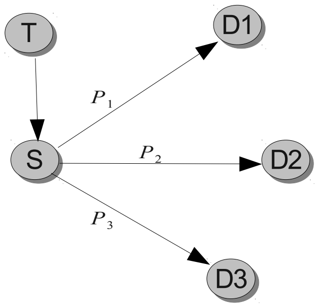
$$S_p = S_{cv} + S_b$$

Suppose the max upload bandwidth of peer i is B_i , then the interval between two consecutive packets is:

$$T_i = \frac{S_p}{B_i}$$

3.7.4.2 To Whom It Should Send Packet?

Each time when a seeder needs to send a packet, it chooses a destination from the downloading peer set randomly. Let P_i^j be the probability of downloader j to be chosen as destination by seeder i. P_i^j is related with downloading history, current upload speed and distance with seeder.



First, suppose H_i^j is the size of data peer i has gotten from peer j before. Large H_i^j means peer j has done much thing for node i thus node i should reciprocate the favor. Current upload speed is also an important factor, seeder should also assign high priority to those new-coming node with large upload speed.

Second, seeder i will preferably send data to closer downloader, which helps reduce the discrepancy between overlay network and real network. A simple metric is defined to represent the distance. Let IP_i be the IP address of the peer i and IP_j be the IP address of the peer j. then the distance between peer i and peer j is:

$$D_i^j = |IP_i - IP_j|$$

Let S_d be the set of downloaders. Here we give a formula to calculate the probability to send packet to downloader j :

$$P_i^j = \frac{H_i^j \times B_j / D_i^j}{\sum\limits_{j \in S_d} P_j^i}$$

In equation, P_j^i is proportional to data peer i has gotten for peer j and the upload speed of B_j . It is also inversely proportional to the distance between peer i and peer j .

3.8 Random Network Coding Operation

Since different generations are independent with each other, we only consider the encoding and decoding of one generation.

3.8.1 Encoding in Source Node

Source Node is a seeder who owns the complete version of this generation. Suppose there are m blocks $[b_1, b_2, ..., b_m]^T$ in this generation. To generate a combined block x_j , it needs to select a set of coding coefficients $[c_{j1}, c_{j2}, ..., c_{jm}]$ in the field size of $GF(2^{8\times codewordsize})$ and produce x_j using the following formula[**12**,**13**]:

$$x_j = \sum_{i=1}^{m} c_{ji} \cdot b_i$$

3.8.2 Encoding in Other Seeders

Suppose a node contains t independent blocks $[x_1, x_2, ..., x_t]^T$. To generate a new block x'_j, it first chooses a set of coding coefficients $[c'_{j1}, c'_{j2}, ..., c'_{jt}]$ in the field size of $GF(2^{8\times codewordsize})$, each of them for a received block. Then x'_j will be:

$$x'_j = \sum_{i=1}^{t} c'_{ji} \cdot x_i$$

The new coefficient vector will be:

$$c_j" = \sum_{i=1}^{t} c_{ji} \cdot c_i$$

3.8.3 Decoding

If a node owns m linearly independently blocks , in will be able to decode this generation. It first forms a $m \times m$ coefficient matrix C , Each row in coefficient matrix C is the coefficient vector of one received combination. It then be able to recover the original data

$$b = C^{-1}X$$

C $^{-1}$ is calculate through Gauss Elimination Method. Indeed, this techniques can during downloading, not after finishing the downloading process.

To accelerate the decoding process, Zeta protocol uses progressive decoding techniques. Instead of decoding after collecting enough packets, a downloader try to predecode the received packets each time when it receives a packet.

First, the downloader cascades the coefficient vectors of all the received packets. These coefficient vectors form a matrix. The downloader try to keep this matrix as a upper-triangle form. Each time when a new blocks arrives, the downloader will check whether it is linear independent with existing packets. If it is, it will add the new coefficient vector as the last row.

The process can be illustrated using the following simple example. If the block number in each generation is 5, a downloader has received 3 linearly independent blocks before. Then the coefficient matrix of this downloader can be:

$$\left(\begin{array}{ccc|cc} 1 & 2 & 3 & 4 & 3 \\ 0 & 5 & 6 & 5 & 4 \\ 0 & 0 & 9 & 8 & 6 \end{array}\right)$$

A dependent block's coefficient vector can be as follows:

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 \end{pmatrix}$$

A independent block's coefficient vector can be as follows:

$$\begin{pmatrix} 1 & 2 & 3 & 1 & 2 \end{pmatrix}$$

To check whether this block is independent with existing blocks, we need only to check the last two variables are zero or not. If any one of them is nonzero, then the new block will be independent with existing blocks. Then it will use Gauss Elimination to eliminate the first 3 variables.

In general, if the block number in each generation is M , the existing linearly independent blocks is k , then checking dependences requires only $n–k$ comparisons. Progressive decoding requires only $\frac{(2M-k)\times k}{2}$ multiplications and additions

4 Architecture of ZetaSim

Many existing analysis of P2P network uses simplified model which doesn't considering the model of the underlying network and most P2P system models comes from classic Client/Server model [**14**, **15**, **16**]. One reason for the ignorance of underlying network characteristics is that there is no mature p2p network simulation software suite.

The GnutellaSim developed by Q. He has greatly reduced the complexity to do simulation of P2P network[**17**]. However, GnutellaSim is designed for Gnutella protocol thus its structure is not suitable for the simulation of Zeta protocol. For example, there is no encoding and decoding process in GnutellaSim.

Based on the pioneering work of Q. He[**9**], I modify the architecture of GnutellaSim and develop a novel NS2 simulation suite for Zeta Protocol which is helpful in analyzing performance of Zeta Protocol.

4.1 Overview

Each node in ZetaSim, including tracker and boot server, has the three-layer architecture as in Fig. 8. The bottom layer is network simulator, for example NS2. Above the network simulator is socket adaption layer. This layer increase the extensibility of the simulation suite. There are many network simulators. However, the transport layer messaging services provided by different network simulators often have different APIs. Socket adaption layer level out the difference of transportation functions and provide a universal interface to the upper layer. The middle layer is messager which deals with protocol dependent messages. Usually, each type of message is corresponding to a message function, the layer of message provide a set of message API to the upper layer.

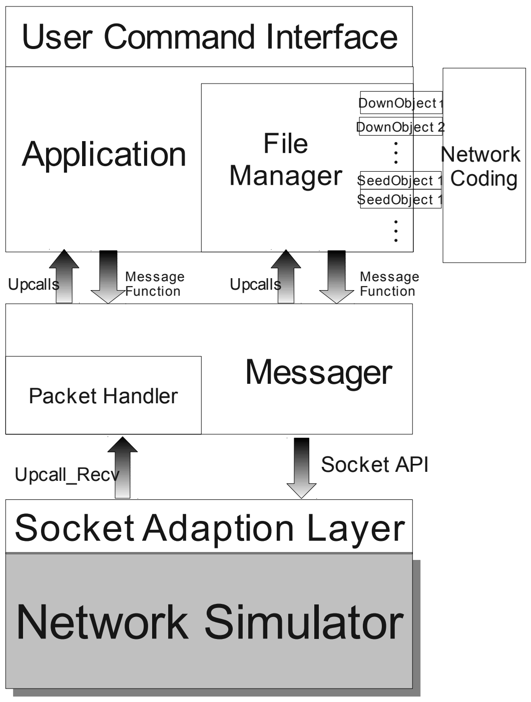


Fig. 8: ZetaSim Architecture

The application entity performs application related action such as join, leave, search and provide user command interface to users. File Manager maintains the list of download object and seeder object. Each download object is in charge of the download of a specific file and each seed object is in charge of a file. I implement ZetaSim as a set of C++ classes, whose brief introduction is in Table 4.

Table 4: C++ classes in ZetaSim

Class Name	**Function**
NSSocket	Provide universal socket API to upper layers.
ZetaPacket	Assemble and parse packet
ZetaMsger	Provides protocol dependent message API
ZetaFilemanager	Manage all downloader objects and seeder objects
ZetaApp	Provide user command interface
Seeder	Seed a specific file for other peers
Downloader	In charge of the download of a specific file
File	Encode and Decode file.

4.2 Describing the Simulation

With the help of ZetaSim, it is quite easy to to the simulation of Zeta Protocol. A p2p simulation scenario usually consists of a set of p2p application running on physical node. Each peer dynamically goes online dynamically and sending file searching request.

Simulation in ZetaSim usually consists of following steps:

1. Set up network topology and select peer node
2. Create socket agent and attach to the selected node
3. Initialize zeta application and attach it with socket agent
4. Set the share file list in each node
5. Control the activity of each node

As the first step of the simulation, we need to set up a physical network. You can either set network topology manually in Tcl script or use the topology generated by three-party tools. Then users must select peer nodes who take part in the node into from the network nodes. Usually only a small portion of the nodes is peer node. Socket agent is in charge of network transmission. After attached with socket agent and physical node, zeta application will be able to run properly. Besides, you must manually specify the shared file list and activity of each node, both of which is not part of ZetaSim.

4.3 Specification of Parameters

ZetaSim has many simulation parameter. Some of them specify the network codition and some other set the state of Zeta application. Table 5 is the list of all parameters.

Table 5: Parameters of ZetaSim

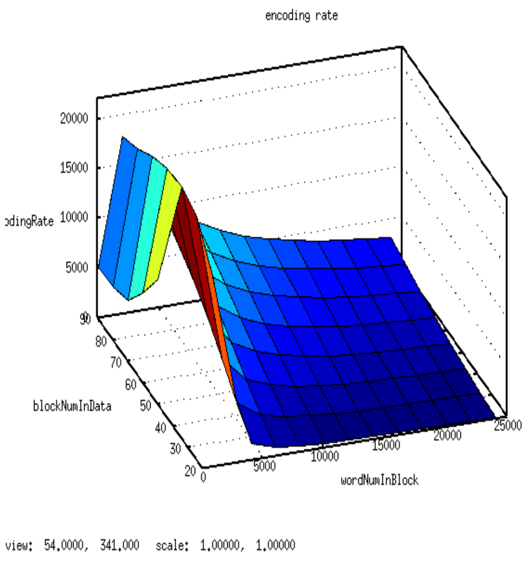
Parameters	***Meaning***
	General Parameters
TotalNum	The total number of nodes
Num	The number of peer nodes in the p2p network
BserverNum	The number of boot server
Duration	The duration of simulation
	The Parameters of Zeta Application
dataSize_	The size of the file that will be distributed
generationSize_	The size of each generation
BlockSize_	The size of each block
codeWordSize	$fieldsize = codewordsize \times 8$
maxUploadSpeed_	The max upload speed of each peer node
isBootServer_	Whether this zeta application is boot server or not
tracker_threshold	When the seeder num < tracker_threhold, tracker will help seed
updatePeersInterval_	The interval between two Update_Peers message
startSeederTh_	Seeder starts when downloaded blocks percentage surpass it

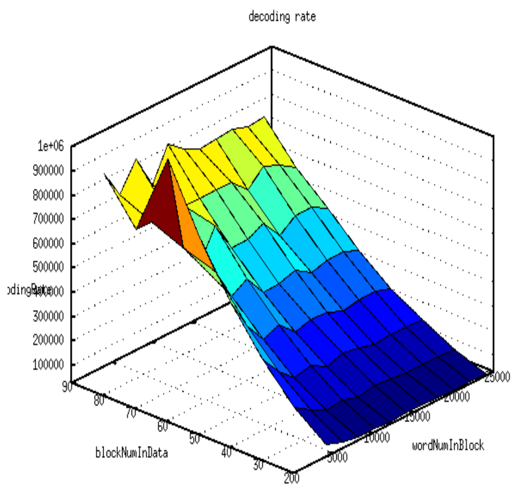
5 Simulation Results

5.1 Network Coding Parameters

The parameters of network coding operations, including code word size, block size and data size, have much influence on the performance of the system. It is well known that encoding and decoding process in network coding is time consuming. Much previous work has been done to accelerate the process. In this section, we analyze the influence of network coding parameters on the encoding and decoding rate.

To exclude the disturbance causes by network uncertainty. We extract the network coding part from ZetaSim so that we can concentrate on network coding. The simulation scenario consists of two nodes, one source node and on e sink node. The source node continuously generates random linear combination of blocks. Instead of waiting until enough blocks are collected, the sink code try to pre-decode all received blocks as soon as a new block arrives.





From the result, we can easily get that the encoding rate is no sensitive with block number but sensitive with word number in block. when word number in block increase from 0 to 25000. The encoding rate increasing rapidly and then decrease.

Besides, decoding rate is not sensitive with word Number In block but quite sensitive with block number in data.

The simulation results are reasonable. Since encoding process involve a large number of byte by byte multiplication in finite field. More word in a block means seeder must do more byte-by-byte multiplication to generate a single block, which will increase calculation overhead.

The situation of decoders are quite different . The most frequency operation to a decoder is Gauss Elimination. The number of multiplication and addition in Gauss Elimination is related to the number of variable, which in this case is the bock number in data. The change of word Number in block can only influence the time of each operation, which is comparably small.

Another interesting phenomenon is that when word number in block and block number in data are 2500 and 90, respectively. The encoding rate increase rapidly while decoding rate decrease rapidly. If you want to encode easily, decoding process will be harder. When we set up network coding parameters, we should find a trade off between encoding rate and decoding rate.

5.2 Simulation In Full Connected Topology

First, we simulate the performance of Zeta Protocol in regular network. Usually, the average downloading time of the nodes in this network to download a file is the most intuitive metric of the performance of the network. The smaller the average downloading time is, the better the p2p downloading protocol is. In spite of average downloading time, we also take consideration of several other parameters.

As noted above, only when the condition of boot server is good, zeta protocol can work well. Usually the burden of boot server is large because it must maintains users' registration and query. In Zeta protocol, some of the boot server's work is transferred to trackers. As a result, we analyze the influence of this techniques.

Third, after sending query, user must wait before successfully set a connection to tracker and start to download. This time is called delay time. If the delay time is long, then user may actively terminate the downloading process, which results in the network resource waste.

In the following section, we will analyze the influence of network size on this three metrics in full connected topology. Although full connected Topology is not practical in most real cases, it is a very useful theoretical tool. Besides, since p2p protocols works above the overlay network, full connected topology can be considered as a abstraction of a overlay network. The simulation scenario consists of 10 nodes. Node 0 is not only the boot server but also the tracker of the network. Node 1, 2, 3 hold the file at the beginning of the simulation. The 10 nodes are full connected and the max bandwidth of each link is 100 Kb. Update peers interval of each peer node is 0.5 seconds the tracker_threshold is 3. The maximum size a NSSocket packet is 50Kb. The code word size is 1 Byte, block size is 20KB, the total size of the file is 3Mb. The threshold to start seeder is 0.5.

Table 6: Simulation Result in Full Connected Graph

Node Number	Ave. Down Rate(KB/s)	Ave. Query Time(s)	Boot Server & Tracker Load(KB/s)	Average Node Load(KB/s)
10	48.503	0.03400	219.3	61.98
30	41.198	0.03408	274.06	59
50	39.13	0.03654	335.2	68
70	43.5823	0.03815	405	85

Table 6 is the simulation result. From the table we know, with the increase of node number, the average query time keep almost constant, which means out network is scalable in terms of querying delay.

Fig. 9 shows the average downloading rate, the average load of boot server & tracker and the average load of peer node. From the picture we know both average downloading rate and boot server's load increase with the node number.

5.3 Simulation In Transit-stub Topology

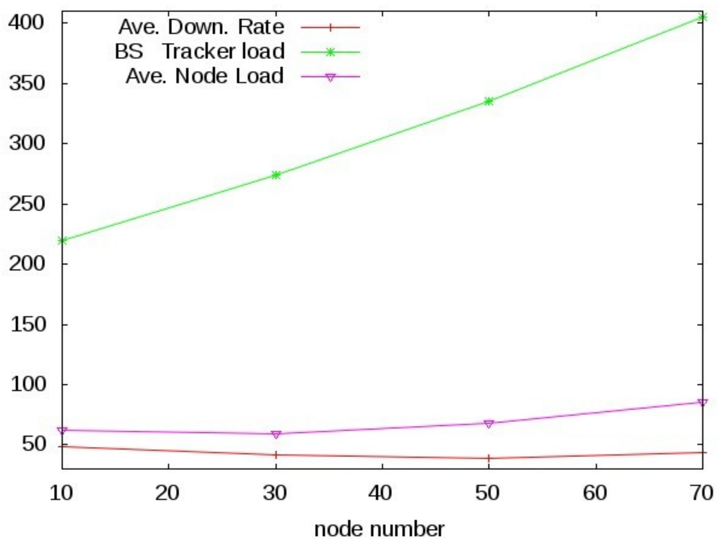


Fig. 9: Simulation Results in Full Connected Topology

Good performance in theoretical overlay network is not good enough. The real Internet structure is quite complicated and the network condition is transitive. To further analyze the performance of Zeta Network, we do some simulation with the aid of ZetaSim and gt-itm.

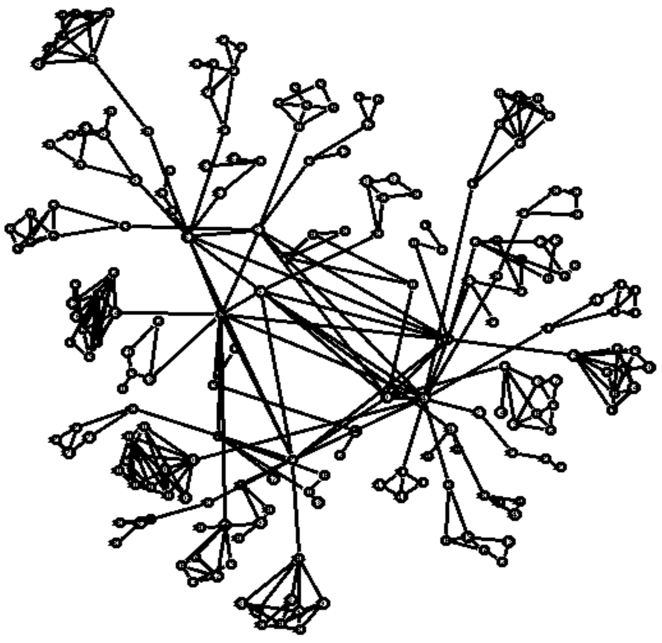


Fig. 10: Transit-stub Topology Produced by gt-itm

We use gt-itm to generate the transit-stub graph topology shown in the Fig. 10. The total number of nodes in this graph is 200, and a portion of the node number participate in the p2p transmission. The simulation result is as follows:

Table 7: Simulation in Transit-Stub Topology

Node Number	Ave. Down Rate(KB/s)	Ave. Query Time(s)	Boot Server & Tracker Load(KB/s)	Average Node Load(KB/s)
100	245.3269	5.4141	1325.85	320.45

6 Conclusion and Future Work

With the simulation of network coding parameters, we find that encoding rate is not sensitive with block number but sensitive with word number in block. Besides, decoding time is not sensitive with word number in block with quite sensitive with block number. Another useful conclusion that usually we must get trade off between encoding time. In many cases, higher encoding time means we must sacrifice some decoding rate and vice versa.

Through observing the simulation result Zeta Protocol in full connected network and Transit-stub topology, we draw the conclusion that Zeta protocol is a scalable and efficient p2p transmission protocol.

Our future work includes improving the flow-control scheme in Zeta Protocol and implementing a real Zeta-Protocol client.

7 Acknowledgement

I would like to thank my supervisor Jiaqing Huang for his guidance and support. He has given much freedom to do what I like and encourage to challenging the existing theory. Without his supervisor, I could not finish the course work in time.

[1]http://en.wikipedia.org/wiki/BitTorrent_(protocol)

[2]Christos Gkantsidis, John Miller, and Pablo Rodriguez, Comprehensive view of a live network coding P2P system, in IMC, Association for Computing Machinery, Inc., October 2006

[3]C. Gkantsidis, John Miller, and P. Rodriguez, Anatomy of a P2P Content Distribution System with Network Coding, in IPTPS'06, February 2006

[4]Christos Gkantsidis and Pablo Rodriguez, Network Coding for Large Scale Content Distribution, in IEEE INFOCOM, March 2005

[5]Raymond W. Yeung, Information Theory and Network Coding, Springer 2008

[6]Raymond W. Yeung et al., Network Coding Theory, now Publishers, 2005

[7]T. Ho, R. Koetter, M. Medard, D. R. Karger and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting" 2003 IEEE International Symposium on Information Theory.

[8]http://www.cc.gatech.edu/computing/compass/gnutella/

[9]Q. He, M. Ammar, G. Riley, H. Raj and R. Fujimoto,Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems, MASCOTS 2003.

[10]http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[11]Z. Ge, D. R. Fiegueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In INFOCOM, 2003

[12]H. Shojania, B. Li, X. Wang, Nuclei: GPU-accelerated many-core network coding, Proc. Of IEEE INFOCOM'09 Rio de.

[13]R G. Dimakis, P. Brighten Godfrey, Y. Wu, M O. Wainwright, K. Ramch, Network Coding for Distributed Storage Systems in Proc. Of IEEE INFOCOM.

[14]D. Nogueira, L.Rocha, J. Santos, P.Araujo, V.Almeida, and W.Meira. A methodology for workload characterization of file-sharing peer-to-peer networks.

[15]K. Kant and R.lyer. A performance model for peer to peer file-sharing serivices, 2001

[16]Z. Ge, D. R. Fiegueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In INFOCOM, 2003.

[17]Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto. Mapping Peer Behaviror to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems.