# *A Research Study on Peer to Peer Networks with Latest Enhancements*

Rubul Kumar Bania, Rofiqul Alam Pramanik, Uttam Malakar

M.Tech Students, Tezpur Central University

**Abstract:**

This work presents basic structure of peer to peer network and latest enhancements of technologies using p2p network till date. Despite recent excitement generated by the peer-to-peer (P2P) paradigm and the surprisingly rapid deployment of some P2P applications, there are few quantitative evaluations of P2P systems behavior. We are focusing on the enhancements of p2p in current network, like Application using Chord algorithm, Massive Multiplayer Online games (MMOG), Kosha**,** ODISSEA: A Peer to Peer Architecture for Scalable Web Search and Information Retrieval, Bittorent. We have explored Bittorent and have proposed a few solutions for Bittorent to overcome some security threats like Fake-Block Attack and Uncooperative-Peer Attack.

***Keywords***: P2P Systems, Chord, Gnutella Network, Bit Torrent.

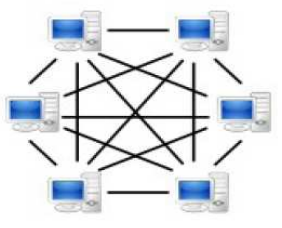
## 1. Introduction:

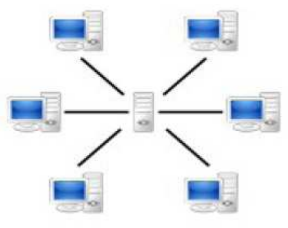
Peer-to-peer (P2P) systems are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality. These systems have recently received significant attention in both academia and industry for a number of reasons. The lack of a central server means that individuals can cooperate to form a P2P network without any investment in additional high-performance hardware to coordinate it. Furthermore, P2P networks suggest a way to aggregate and make use of the tremendous computation and storage resources that remain unused on idle individual computers. Finally, the decentralized, distributed nature of P2P systems makes them robust against certain kinds of faults, making them potentially well-suited for long-term storage or lengthy computations. P2P systems are, of course, distributed systems, and much traditional distributed systems research is relevant to them. Relatively unusual, however, is the assumption in P2P systems that nodes are continuously joining and leaving the system. This makes challenging several issues which are trivial in a system with a fixed membership. In particular, data items must migrate as nodes come and go, this makes location of a data item at any given time nontrivial. For some applications, that node might be responsible for storing a value associated with the key; for others, it might perform computation on that key. The easiest way to implement a content addressable network is to maintain a directory of key assignments. Unfortunately, maintaining this directory consistently in a distributed environment is too resource-intensive to scale.

[ *The authors Rubul Kumar Bania, Rofiqul Alam Pramanik,Uttam Malakar {rubul.bania, rasselpramanik1,uttam.malakar}@gmail.com are the student in the Department of Computer Science & Engineering Tezpur University,Napaam,Sonitpur,Assam-784028 INDIA* ]

## 1.1 File Sharing

File sharing is the practice of distributing or providing access to digitally stored information, such as computer programs, multi-media (audio, video), documents, or electronic books. It may be implemented in a variety of storage, transmission, and distribution models. Common methods are manual sharing using removable media, centralized computer file server installations on computer networks, World Wide Web-based hyperlinked documents, and the use of distributed peer-to-peer (P2P) networking. The emergence of file-sharing systems such as Napster and Gnutella [1] make the term peer-to-peer (P2P) popular. In a peer-to-peer system all nodes are symmetric and there is no central control o hierarchy. In a typical peer-to-peer system, each user has some information that may be of Interest to other users. This information may be free software, music, photographs, and so on. If there are large numbers of users, they will not know each other and will not know where to find what they are looking for. One solution is a big central database, but this may not be feasible. Firstly, nobody wants to take the responsibility to host and maintain it. Secondly, if the database crashes, the functioning of the entire network fails. Thus, the problem comes down to how a user finds a node that contains what he is looking for in the absence of a centralized database or even a centralized index. This can be achieved by searching for the files sequentially in all the nodes present in the network. But this is not very efficient as there may be lakhs of users in a network. Searching a file sequentially would consume a lot of bandwidth. This problem can be reduced by using a distributed hash table which hashes the information of the files to the various nodes and the search can then be performed in an efficient way using the famous "Chord Algorithm".


A peer-to-peer system of nodes without a central infrastructure.


Centralized server-based service model

## 1.2 Architecture of P2P systems

Peer-to-peer networks are typically formed dynamically by *ad-hoc* additions of nodes. In an *'ad-hoc'* network, the removal or addition of nodes has no significant impact on the network. The distributed architecture of an application in a peer-to-peer system provides enhanced scalability and service robustness. In *structured* peer-to-peer networks, connections in the overlay are fixed. They typically use distributed hash table-based (DHT) indexing, such as in the Chord system.

*Unstructured peer-to-peer* networks do not provide any algorithm for organization or optimization of network connections. In particular, three models of unstructured architecture are defined. In *pure peer-to-peer* systems the entire network consists solely of equipotent peers. There is only one routing layer, as there are no preferred nodes with any special infrastructure function. *Hybrid peer-to-peer* systems allow such infrastructure nodes to exist, often called *supernodes*. In *centralized peer-to-peer* systems, a central server is used for indexing functions and to bootstrap the entire system. Although this has similarities with a structured architecture, the connections between peers are not determined by any algorithm.

## 2. *Latest Enhancement of Peer-to-Peer (P2P) Network*:

### 2.1 The Peer-to-Peer network application using Chord algorithm:

A peer-to-peer system does not have any centralized control or hierarchical organization. Every node in a peer-to-peer system has equal functionality. The core operation in most peer-to-peer systems is efficient location of data items. The Chord algorithm [8] provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data pair at the node to which the key maps. The Chord algorithm provides fast distributed computation of a hash function mapping keys to nodes responsible for them. Chord assigns keys to nodes with consistent hashing, which has several desirable properties. Chord improves the scalability of consistent hashing by avoiding the requirement that every node know about every other node. A Chord node needs only a small amount of “routing” information about other nodes. Because this information is distributed, a node resolves the hash function by communicating with other nodes.

### 2.2 Peer-to-Peer Architecture for Media Streaming Systems :

The success of peer-to-peer (P2P) [1] applications in both commercial and research fields. However, a practical area has received little attention in the context of P2P to date: media streaming [2]. Given the fact that the current Internet does not widely support IP Multicast while content distribution- networks technologies are costly, P2P could be a promising start for enabling large-scale streaming systems. Peer-to-peer (P2P) computing has been of interest for quite long and numerous file-sharing systems based on its concepts have been developed .The applicability of P2P to the problem of streaming *live* media. It is so-called Zigzag approach [2]; the propose method for clustering peers into a hierarchy called the administrative organization for easy management, and a method for building the multicast tree a top this hierarchy for efficient content transmission. In the P2P streaming architecture, the delivery tree is built rooted at the source and including all and only the receivers. A subset of receivers get the content directly from the source and the others get it from the receivers in the upstream. Consequently, this paradigm promises to address many critical problems in large-scale streaming systems: (1) the network bandwidth bottleneck at the media source; (2) the cost of deploying extra servers, which would be incurred in content distribution networks; and (3) the infeasibility of IP Multicast on the current internet.

### 2.3 Kosha: A Peer to Peer Enhancement for the Network File System:

Kosha[6] is a peer-to-peer (p2p) enhancement for the widely-used Network File System (NFS). Kosha harvests redundant storage space on cluster nodes and user desktops to provide a reliable, shared file system that acts as a large storage with normal NFS semantics. P2p storage systems provide location transparency, mobility transparency, load balancing, and file replication-features that are not available in NFS. On the other hand, NFS [6] provides hierarchical file organization, directory listings, and file permissions, which are missing from p2p storage systems. By blending the strengths of NFS and p2p storage systems, Kosha provides a low overhead storage solution. Our experiments show that compared to unmodified NFS, Kosha introduces a 4.1% fixed overhead and 1.5% additional overhead as nodes are increased from one to eight. For larger number of nodes, the additional overhead increases slowly. Kosha achieves load balancing in distributed directories, and guarantees 99:99% or better availability.

### 2.4 ODISSEA: A Peer to Peer Architecture for Scalable Web Search and Information Retrieval

Due to the large size of the Web, users increasingly rely on specialized tools to navigate through the vast volumes of data, and a number of search engines, directories, and other IR tools have been built to fill this need. While there is a plethora of smaller specialized engines and directories, the main part of the search infrastructure of the web is supplied by a handful of large crawl-based search engines, such as Google, AllTheWeb, AltaVista, and a few others. Such search engines are typically based on scalable clusters, consisting of a large number of low-cost servers located at one or a few locations and connected by high-speed LANs or SAN. A lot of work has focused on optimizing performance on such architectures, which support up to tens of thousands of user queries per second on thousands of machines. The problem of building a P2P-based search engine for massive document collections. A prototype system called ODISSEA [10] (Open DIStributed Search Engine Architecture) that is currently under development. ODISSEA provides a highly distributed global indexing and query execution service that can be used for content residing inside or outside of a P2P network. ODISSEA is different from many other approaches to P2P search in that it assumes a two-tier search engine architecture and a global index structure distributed over the network.

### 2.5 Enhancement of P2P Architecture for Massive Multiplayer Online Games (MMOG):

Massive Multiplayer Online Games with their virtual gaming worlds grow in user numbers as well as in the size of the virtual worlds. With this growth comes a significant increase of the requirements for server hardware. Today an MMOG [3] provider usually faces the problem of serving thousands of users with entire server clusters. Peer-to-Peer networks with their high scalability and flexibility meet the requirements of connecting hundreds of thousands of people all over the world without a central server. In doing so the network bandwidth requirements remain at a reasonable level. In this model ,Combine MMOGs with a Peer to- Peer network. Introducing a game architecture capable of exploiting the flexibility and scalability of P2P networks. A P2P architecture based on an overlay network using distributed hash tables(DHT) with support for persistent object storage and event distribution has been developed to meet MMOG requirements.

## 2.6 Gnutella Network:

The Gnutella[1] protocol is an open, decentralized group membership and search protocol, mainly used for file sharing. The term Gnutella also designates the virtual network of Internet accessible hosts running Gnutella-speaking applications (this is the "Gnutella network" we measure) and a number of smaller, and often private, disconnected networks. As most P2P file sharing applications, Gnutella protocol was designed to meet the following goals:

- *Ability to operate in a dynamic environment*. P2P applications operate in dynamic environments, where hosts may join or leave the network frequently. They must achieve flexibility in order to keep operating transparently despite a constantly changing set of resources.
- *Performance and Scalability*. P2P paradigm shows its full potential only on large-scale deployments where the limits of the traditional client/server paradigm become obvious. Moreover, scalability is important as P2P applications exhibit what economists call the "network effect" the value of a network to an individual user increases with the total number of users participating in the network. Ideally, when increasing the number of nodes, aggregate storage space and file availability should grow linearly, response time should remain constant, while search throughput should remain high or grow.
- *Reliability*. External attacks should not cause significant data or performance loss.
- *Anonymity*. Anonymity is valued as a means to protect privacy of people seeking or providing information that may not be popular.

Gnutella nodes, called ***servents*** by developers, perform tasks normally associated with both **SERV**ers and cli**ents**.

## 2.7 Bittorent:

BitTorent [7] is a P2P protocol for content distribution and replication designed to quickly, eficiently and fairly replicate data. Among all available peer-to-peer Internet applications, BitTorrent has become the most popular for file sharing. Recent reports have indicated that near 75 % of all the current P2P Internet traffic is due to BitTorrent . One of the reasons of BitTorrent's popularity is that it provides very efficient file sharing; allowing downloads to scale well with the size of the downloading population. This efficiency is obtained by breaking up each large file into hundreds or thousands of segments, or pieces, which, once downloaded by a peer, can be shared with others while the downloading continues. All these features have made BitTorrent a leading P2P system in the Internet. However, BitTorrent has a serious vulnerability, it utilizes a central server, known as a "tracker", to coordinate connec- tions between peers in a "swarm", a term used to describe a BitTorrent ad-hoc file sharing network for a file (or a set of files) provided by a file distributor. The tracker of a swarm is specified by the swarm's original file distributor. The tracker doesn't implement selection mechanism on new arrivals and doesn't apply any control measures on the data declared by the system peers either. Moreover all peers in the swarm trust the tracker without implementing any authentication or verification procedures. Malicious nodes can take advantage of these facts and use them to their ends, and attack the infrastructure of BitTorrent.

### 2.7.1 How Bit Torrent Works

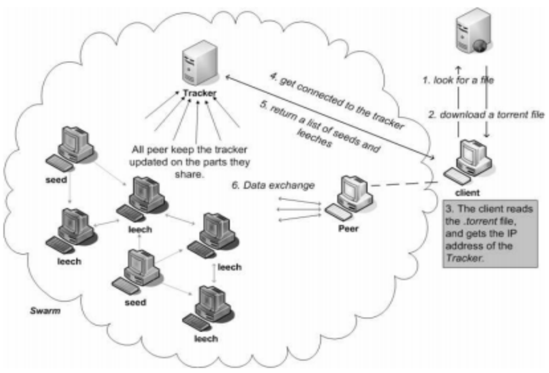

Tracker
4. get connected to the tracker
5. return a list of seeds and leeches
1. look for a file
2. download a torrent file
All peer keep the tracker updated on the parts they share.
seed
leech
6. Data exchange
Peer
client
3. The client reads the .torrent file, and gets the IP address of the Tracker.
leech
leech
seed
leech
Swarm

*Fig: How bit Torrent Works*

In the above figure, it is illustrate how a client downloads a file from a BitTorrent swarm. The tracker is installed on a machine which is located in a swarm represented by a cloud. Let's imagine a scenario, where a client shows an interest in downloading a certain file. The client first searches for the desired file by consulting a known website (step 1). The client would then download a torrent file which its metadata matches the desired file (step 2). Next, the client will read the content of the torrent file, and get the tracker address (step 3). Once, the client obtains the tracker address, he gets connected to it, announces its will to download the shared file and asks the tracker about other peers (step 4). When asked for peers, a tracker will return a random list of other peers currently in the swarm. As the number of peers in a single swarm may become very large for popular files, the size of the returned list is usually bound; a maximum of 50 peers is typical (step 5). Once a client has obtained a list of other peers, it will contact them to try to fetch the data it is looking for. In BitTorrent, file content is split into small-sized pieces and each client maintains the list of the pieces it holds. After a handshake, peers exchange their piece lists so that each of them may determine whether the other has some lacking pieces. The bandwidth being a limited resource, a single client cannot serve every peer interested in pieces it holds at the same time. The maximum number of peers served concurrently (i.e. the number of available slots) is configurable by the user and depends on the available bandwidth. All other peers connected to a client (whether they are interested or not) which are not being served are said to be choked. In consequence, each client implements an algorithm to choose which peers to choke and un-choke among those connected to him over time. The strategy proposed by BitTorrent is named "tit-for-tat", meaning that a client will preferably cooperate with the peers cooperating with him. Practically, this means that each client measures how fast it can download from each peer and, in turn, will serve those from whom it has the better download rates. When a client has finished downloading a file, the choking algorithm is applied by considering upload rate instead. Peers are selected based on how fast they can receive the upload. This spreads the file faster.

### 2.7.3 Peer Distribution

All logistical problems of file downloading are handled in the interactions between peers. Some information about upload and download rates is sent to the tracker, but that's just for statistics gathering. The tracker's responsibilities are strictly limited to helping peers find each other.

Although trackers are the only way for peers to find each other, and the only point of coordination at all, the standard tracker algorithm is to return a random list of peers. Random graphs have very good robustness properties. Many peer selection algorithms result in a power law graph.In order to keep track of which peers have what, BitTorrent cuts files into pieces of fixed size, typically a quarter megabyte.Each downloader reports to all of its peers what pieces it has.To verify data integrity, the SHA1 hashes of all the pieces are included in the .torrentfile, and peers don't report that they have a piece until they've checked the hash. Erasure codes have been suggested as a technique which might help with file distribution, but this much simpler approach has proven to be workable.

### 2.7.4 Pipelining

When transferring data over TCP, like BitTorrent does, it is very important to always have several requests pending at once, to avoid a delay between pieces being sent, which is disastrous for transfer rates. BitTorrent facilitates this by breaking pieces further into sub-pieces over the wire, typically sixteen kilobytes in size, and always keeping some number, typically five, requests pipelined at once. Every time a sub-piece arrives a new request is sent. The amount of data to pipeline has been selected as a value which can reliably saturate most connections.

### 2.7.5 Piece Selection

Selecting pieces to download in a good order is very important for good performance. A poor piece selection algorithm can result in having all the pieces which are currently on offer or, on the flip side, not having any pieces to upload to peers we wish to.

### 2.7.6 Strict Priority

BitTorrent's first policy for piece selection is that once a single sub-piece has been requested, the remaining sub-pieces from that particular piece are requested before sub-pieces from any other piece. This does a good job of getting complete pieces as quickly as possible.

### 2.9.7 Rarest First

When selecting which piece to start downloading next, peers generally download pieces which the fewest of their own peers have first, a technique we refer to as 'rarest first'. This technique does a good job of making sure that peers have pieces which all of their peers want, so uploading can be done when wanted. It also makes sure that pieces which are more common are left for later, so the likelihood that a peer which currently is offering upload will later not have anything of interest is reduced. Information theory dictates that no downloaders can complete until every part of the file has been uploaded by the seed. For deployments with a single seed whose upload capacity is considerably less than that of many downloaders, performance is much better if different downloaders get different pieces from the seed, since redundant downloads waste the opportunity for the seed to get more information out. Rarest first does a good job of only downloading new pieces from the seed, since downloaders will be able to see that their other peers have pieces the seed has uploaded already. For some deployments the original seed is eventually taken down for cost reasons, leaving only current downloader have to upload. This leads to a very significant risk of a particular piece no longer being available from any current downloader's. Rarest first again handles this well, by replicating the rarest pieces as quickly as possible thus reducing the risk of them getting completely lost as current peers stop uploading.

### 2.7.8 Random First Piece

An exception to rarest first is when downloading starts. At that time, the peer has nothing to upload, so it's important to get a complete piece as quickly as possible. Rare pieces are generally only present on one peer, so they would be downloaded slower than pieces which are present on multiple peers for which it's possible to download sub-pieces from different places. For this reason, pieces to download are selected at random until the first complete piece is assembled, and then the strategy changes to rarest first.

## 3. Security issues in Bittorent and our Proposal:

**Fake-Block Attack***:* As mentioned previously in BitTorrent each file is divided into pieces, where each piece is usually 256kb (depending on the overall size of the file). Each piece is further divided into blocks, typically 16 blocks in a piece. When downloading a piece, a client requests different blocks for the piece from different peers [4]. A non-malicious attack in nature the fake-block attack seeks to prolong your download times. The attacker joins the swarm sharing the file by registering with the tracker. Then it begins to advertise it has a number of pieces from the file. The victim receives the message and requests the attacker to send its blocks. The attacker instead of sending an authentic block will send a fake one. After the victim finishes downloading the block and the entire piece, a hash check is performed across the entire piece. The hash check will of course fail because of fake blocks and the user will then have to re-download the entire piece again. The victim just wasted 256kb of bandwidth, which in itself is not a lot but it is the bigger picture we must look at. Some people suggest a possible solution to this attack is to give the user an option in their BT client to ban certain seeders. If the client fails a hash check, the client searches for the IPs related to the blocks that failed the test and eliminates them from the individual's swarm. But as we know that there will be a number of seeds in a swarm which carries the different parts of the file. Checking for each individual seed will waste a lot of time and also the network bandwidth.

***Suggestion***: What we want to suggest is that when a seed advertises it has a number of pieces of a file, the tracker should take the responsibility of computing the hash values of the pieces to determine whether the pieces are genuine or not. If the pieces are not genuine then the tracker removes that particular seed from the swarm and sends the user an appropriate message.

**Uncooperative-Peer Attack**: In an uncooperative-peer attack, the attacker joins the swarm and establishes TCP connections with victim peers. After the connection is made it never provides any blocks to the peers. A common version of this attack is called the chatty peer attack. The attacker engages in a handshake message, which is the first connection that is established between two peers. Afterwards the attacker advertises it has a number of pieces available from the file. When the victim queries the attacker for a block they do not receive anything. The attacker then resends its handshake message and the process repeats itself. The victim never receives any blocks and wastes time dealing with the attacker, when it could have connected to a legitimate peer.

***Suggestion:*** The solution to this problem is to set an auto retry limit (a threshold) for each client. Each time the client resends a request for a particular piece the counter increments by one. When this counter exceeds threshold value, the client informs the tracker about this activity and the tracker removes the seed from the swarm accordingly.

## 4. Conclusion

Peer to Peer systems are very much efficient for several uses. P2P networks suggest a way to aggregate and make use of the tremendous computation and storage resources that remain unused on idle individual computers. This system is a low cost alternative to the traditional Client/Server model. The recent systems are scalable. Though this system has a lot of advantages, it has disadvantages too. It is still not a perfect system because there are issues like trust and certification, anonymity, security which are yet to overcome.

## *References*

1. Matei Ripeanu , Department of Computer Science, The University of Chicago ,Peer-to-Peer Architecture Case Study: Gnutella Network,.

2. Duc A. Tran, *Member, IEEE,* Kien A. Hua, *Senior Member, IEEE,* and Tai T. Do ,A Peer-to-Peer Architecture for Media Streaming ,.

3. Thorsten Hampel, Thomas Bopp, Robert Hinn, University of Paderborn, 33102 Paderborn, Germany ,A Peer-to-Peer Architecture for Massive Multiplayer Online Games,.

4. Sinah Hatahet, Yachine Challal, Abdelmadjid Bouabdallah, Heudiasyc UMR 6559, Universite de Technologie de Compiegne, BitTorrent Worm Sensor Network: P2P Worms Detection and Containment,

5. Bin Cheng , Xiuzheng Liu , Zheng Zhang , Hai Jin, Lex Stein, Xiaofei Liao, Huazhong University of Science and Technology, Wuhan, China ,Evaluation and optimization of a peer-to-peer video-on-demand system.

6. Ali Raza Butt, Troy A. Johnson, Yili Zheng, and Y. Charlie Hu, Purdue University West Lafayette, IN 47907, Kosha: A Peer to Peer Enhancement for the Network File System,.

7. Bram Cohen ,Incentives Build Robustness in BitTorrent .

8. Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, "*Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications*" in IEEE/ACM Transactions on Networking, vol. 11(1), February 2003.

9. Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger Robert Morris, Ion Stoica, Hari Balakrishnan *MIT Laboratory for Computer Science, "Building Peer-to peer systems with Chord, a distributed lookup service"*

10. Torsten Suel, Chandan Mathur, JoWen Wu, Jiangong Zhang Department of Computer and Information Science Polytechnic University Brooklyn, NY 11201 ODISSEA: A Peer to Peer Architecture for Scalable Web Search and Information Retrieval

********