Peer to Peer File Sharing Networks : A Review

Kousshik Raj, Satyam Porwal, Prashant Ramnani, Robin Babu, Shivam Jha

March 24, 2021

1. INTRODUCTION

In the current era, where data is becoming more prevalent, it is an infrequent occurrence to find a standalone node. Instead, we often see a network of interconnected nodes responsible for a part of the network's resources and services. Such networks can be primarily grouped into centralised systems and distributed systems. In a centralised solution, a single node is treated as the core of the network. It is responsible for all the processing taking place in the network. The other network components usually take care of other tasks such as storage, input and output. For example, a desktop connected with a projector, printer, mouse, keyboard, and storage devices through the LAN comes under such a classification.

On the contrary, in a distributed network, the computation is shared among the network's different nodes. These networks can be further segregated into a client-server network model and a peer-to-peer (commonly known as p2p) network model based on how the distribution happens.

The client-server model is somewhat similar to the centralised systems in that this network revolves around a single (or a set of) core node(s). In this model, a client (periphery nodes) requests the server (a core node) for content or services without sharing any of its own to the rest of the network. An example of such a system would be a network of nodes that run a DBMS application where a central node (the server) stores the entire database and the periphery nodes query the server for various details. Such a network can either be flat (have a single server) or hierarchical (layers of servers) to improve the network's stability.

Whereas, the p2p model offers a novel solution to the network operations of a distributed system as opposed to the traditional client-server systems, which can be extended to many application domains. In this model, the network comprises many heterogeneous and highly dynamic nodes, also called the peers, where they share a part of their content/resources such as storage, processing power, or files like multimedia software with the rest of the network. Such p2p networks can either be fully decentralised (pure network) or have some centralisation (hybrid network). This classification is exemplified by Fig. 1. This report will primarily focus on the p2p network model - its architecture, structure, and design challenges.

What makes p2p networks different from the others is that the participating nodes can both act as clients and servers simultaneously. Each node can access other nodes in the network without any intermediary. As can be seen from their names, a pure p2p system will not suffer from a node often coming in and pulling out of the network. Such coming in and pulling out is often referred to as *churn* in the network. However, in the case of a hybrid network, as it uses some centralised entities to manage the network, such high churn will not always be benign.

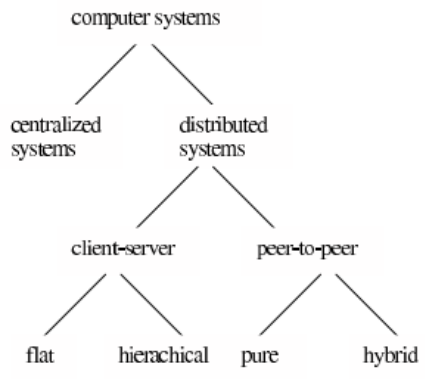

Fig. 1. Classification of System Networks

P2P networks are often associated with the concepts of resource sharing, decentralisation, and self-organisation. As we have seen, every node of the network shares some of the resources with the network, resulting in the decentralisation of the network to a certain extent. In a completely decentralised network, there is no entity to manage the network globally. Hence, the network nodes self organise themselves by the information it obtains from locally reachable nodes. As a consequence of such local operations, global behaviour can be seen throughout the network.

2. P2P ARCHITECTURES

As we discussed earlier, complete decentralisation is not an integral aspect of all p2p networks. As such, not all networks have the same amount of decentralisation. Hence, based on the degree of decentralisation, the p2p networks are further classified into pure and hybrid architectures.

2.1. Purely Decentralised

These systems are also commonly called as *pure* p2p systems. This is because they act without any centralised means of control, and thereby they exemplify a completely decentralised system. This implies that every node is equivalent to one another and is often called *servant* (SERver + cliENT), which highlights the capability of peers in a p2p network to act as both clients as well as servers. CAN [1], Chord [2], Freenet [3], Gnutella [4], etc. are P2P systems that fall under this class.

The most notable advantage of systems of this class is that they are inherently scalable and fault-tolerant. The amount of centralised actions necessary usually restricts scalability in a network, and as there are no such limitations in this architecture, these networks can grow indefinitely. Moreover, since a loss of one or more peers does not significantly affect the system and can be easily compensated because of the equivalence of the nodes, these systems are inherently fault-tolerant.

However, along with the advantages, they also come with specific detriments. These systems do not have any means to support fast new information discovery, thereby affecting the quality of service provided. This often results in flooding the network to ensure a guaranteed lookup. Also, these systems lack predictability, as there is no view of the system globally.

2.2. Hybrid Architecture

In hybrid p2p systems [5], there exist one or more central entities governing the system's activities by maintaining a meta-data of the registered users and the content shared in the network. However, even in such a system, end-to-end interactions happen only between peers. There are two types of hybrid systems: centralised indexing and decentralised indexing.

In centralised indexing, the central server maintains an index of all the data currently shared in the network by the active peers. Each peer maintains a passive connection to the server, to which it sends a query for the required resource when necessary. After locating the required resource, data is exchanged in a peer-to-peer fashion. This architecture can

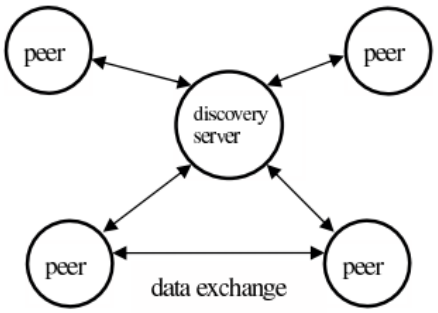

Fig. 2. Centralised Indexing Hybrid System

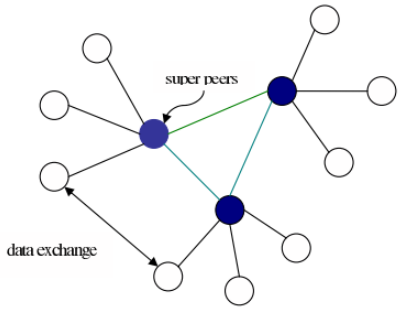

Fig. 3. Decentralised Indexing Hybrid System

be found in Napster. Fig. 2 shows the basic idea behind such a system.

The mode of operation of such systems is elementary and straightforward. They are very efficient in propagating the finding of new information, and thus the searches are quite comprehensive. On the other hand, they are quite vulnerable to malicious attacks, and because of the presence of central servers, there is a single point of failure. Furthermore, unlike the pure p2p systems, they do not support the network's indefinite growth as the central server limits it due to its capacity to respond to queries and its database size.

In decentralised indexing, we again have a central server to register the users to facilitate the peer discovery process. However, unlike the centralised indexing system, here, some of the peers assume a more important position than the rest of the nodes in the network. These are collectively called *SuperNodes* [5]. These nodes are responsible for indexing the information shared by the local peers connected to that SuperNode, and they also proxy the search requests on behalf of these peers. Thus, the queries are sent to the nearest SuperNode rather than all the neighbouring peers like with the other system models. Kazaa [6] and Morpheus are instances of the decentralised indexing hybrid systems. Fig. 3 demonstrates such a system in action.

The interesting fact in such networks is that a peer is automatically qualified to become a SuperNode if they have sufficient bandwidth and processing power. Once such a node joins the network, it becomes a SuperNode and establishes a connection with similar nodes. However, if it does not receive a required number of connections from the nearby clients within a certain period of time, it is demoted back to a normal node. When a new node registers with the central server, the server provides it with the address of one or more SuperNodes that it can connect to.

Though this system offers a slower information discovery time than the centralised indexing system, it is still much better than the pure p2p architectures. Moreover, with the SuperNodes proxying the requests on behalf of the local peers, the query traffic, and the central server's workload reduce drastically. This system is also quite resilient to faults, as the failure of even multiple SuperNodes can be compensated by the election of new SuperNodes in their place.

3. DISCOVERY METHODS

One of the most essential parts of distributed peer-to-peer file sharing systems is the discovery mechanism for locating the data within the distributed system.

The first P2P systems, Napster, started with using a centralised structure for this purpose. The second generation of systems, Gnutella, used a flooding-based approach. More recent

systems have implemented a Distributed Hash Table (DHT) based approach, for example, systems like Chord and Kademlia-based IPFS.

Each of these discovery methods come with different classes of overlay networks: Structured and Unstructured. We will discuss more about this in part IV. For now, let us discuss certain discovery methods:

3.1. Centralised indexes and repositories

Commonly used in hybrid systems, this model requires the peers to interact with a centralised server that stores all information of location and resource usage. Whenever a node requests, the central server will process it and find the best peer in its directory that matches the request requirements. The best can be in terms of speed, cost or availability, depending on the needs.

This model was introduced in Napster, one of the pioneers of the P2P file-sharing system. In Napster, a central directory is maintained, which stores the metadata of all the files, a table of registered user connection information, and a table listing the files that each user holds and shares in the network.

When a client requests the central server, it also sends the list of files it maintains. After processing the request, the server returns a list of users that hold the matching file. The client then opens a direct connection to one of the peers and requests and downloads the file.

3.2. Query Broadcast Flooding

Often used in purely decentralised P2P systems, each peer publishes information about its shared contents in the P2P network. This means no single node is aware of all the resources in the network.

Thus for resource discovery, flooding must be performed. Request broadcasting is performed until the request is answered or external constraints like Time-to-Live (TTL) are met. Thus, the search network is built in an ad hoc manner, without restricting a priori which nodes can connect or what information can be exchanged.

Without any external constraints, the network would be swamped with requests and other messages. Furthermore, accurate discovery is not guaranteed in flooding mechanism, and optimal results are seen in a small to an average number of peers, hence not a scalable approach. To overcome these problems, TTL-limiting flooding can be used.

This effectively segments the network into various subsets, imposing a virtual horizon beyond which the message cannot be reached. Although the value of TTL would depend on various factors, Gnutella found seven hops as their horizon standard. Typically, a seven hop radius along with the network constraints would imply around 10,000 nodes are reachable.

3.3. Routing Model/ DHT

Commonly used in more structured systems, where the resources are stored using distributed hash tables. In this protocol, we try to provide a mapping between the resource identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to nodes with the desired resources.

We try to reduce the number of P2P hops that are required to locate the resource. The look-up service is implemented to organise in a structured overlay network and routing messages through the overlay to responsible peers.

Chord, Content Addressable Network (CAN) and Pastry are some examples of systems that use Distributed Hash Table [7] (DHT)-based approach for their look-up services.

All the DHT topologies share some variety of key-based routing. Beyond the basic routing correctness, two essential constraints on the topology are to guarantee that the maximum number of hops in any route is the low

and maximum number of neighbours of any node is low to minimise the maintenance overhead. Moreover, shorter routes require a higher degree. The most common choice $O(logN)$ degree/routing length is not optimal in terms of degree/route length tradeoff; hence these topologies allow more flexibility to choose neighbours based on features like latency.

Other than routing, many systems exploit the network structure for sending messages to nodes in a DHT. These are used to perform overlay multicast, range queries or collect statistics. Structella [8] implements flooding and random walks on a Pastry overlay and DQ-DHT, which implements a dynamic querying search algorithm over Chord Network [9].

4. P2P NETWORK STRUCTURES

The peer-to-peer mechanism is usually considered a technology where the data is shared across many peers. This is a much more successful approach than the centralized approach [10, 11]. P2P networks can be classified in many ways. One of the more common ways to do it is by identifying if the network contains some structure or is more ad-hoc.

The technical meaning of structured is that P2P overlay network topology is tightly controlled, and contents are placed at specific known locations and not spread at random peers, making subsequent queries much more efficient. These structured P2P systems use DHT, where data is placed deterministically using a unique key. The topology in these networks is tightly controlled.

Whereas in unstructured networks, the placement of the data is entirely unrelated to the topology and peers in-place. In such cases, neighbours have no information about each other's data, and flooding is usually involved in searching in these systems.

We will discuss key features of Structured P2P and Unstructured P2P overlay networks, their operational functionalities, evaluate them and discuss various developments in each.

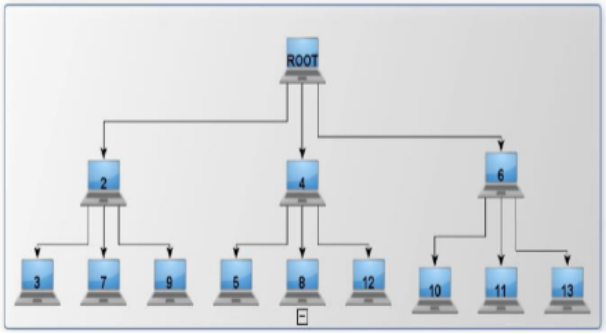

Fig. 4. Structured Peer to Peer

4.1. Structured P2P Networks

In this, the overlay network assigns keys to data and organises its peers into a graph, mapping each data key to a peer. The structured graph ensures efficient discovery of data using the keys. However, handling complex queries is not so simple in this case.

In structured P2P systems, the resource indices are organised, and an inflexible structure is employed to interconnect different peers. DHT and a distributed indexing service based on hashing are used in these systems [12]. The hash function used here is also used to map the resources and peers to a keyspace. Fig. 4 presents the abstraction view of structured peers.

Chord [2] is the first structured P2P system introduced in 2001. Chord uses consistent hashing to assign keys to its peers [13]. It is designed to let peers enter and leave the network with minimal interruptions. This decentralised scheme tends to balance the load on the system. The lookup process follows something similar to a binary search. The index of all resources whose keys fall in the range of peer, each peer stores it if it is between the predecessor and its own key.

More systems with structured overlay are Pastry, Tapestry and CAN. Studies in [14] pre-

sented an Ant-Colony-Optimisation (ACO) algorithm for the discovery of resources in large P2P grid systems. Each query message can be thought of like an ant, moving in the network from nest to nest to locate the resource and return via the same path to the original nest upon finding the required resources. This prevents not only large-scale flat flooding but also supports a multi-attribute range query. Parallelism can be employed by using multiple ants.

4.2. Unstructured P2P Networks

Unstructured peer-to-peer networks do not impose a particular structure on the overlay network by design but are formed by nodes that randomly form connections.

Since there is no structure globally imposed on these, unstructured networks can easily build and optimise different regions of the overlay. Each peer evaluates the query on the local content and supports complex queries, thereby making it inefficient for content that are not widely replicated. The roles of all the peers in the network are the same. Compared to a structured network, unstructured networks are highly robust when there is a high rate of churn - the number of peers joining and leaving is high.

The major limitations of the network arise from the unstructured nature as well. Since flooding is involved, a high amount of signalling traffic is caused. Popular content will be available to several peers, but searching for not so popular/rare data will more likely result in failure. Some unstructured P2P networks are FreeNet, Gnutella, FastTrack/KaZaA and BitTorrent.

In addition to these classifications, recent systems have seen a hybrid model [15] which tries to use a combination of P2P and client-server model to develop a model with better performance than either pure structured or unstructured networks overcoming some of their tradeoffs, such as searching is easier on a centralised system and also benefit from the decentralised aggregation of nodes from an unstructured network [16].

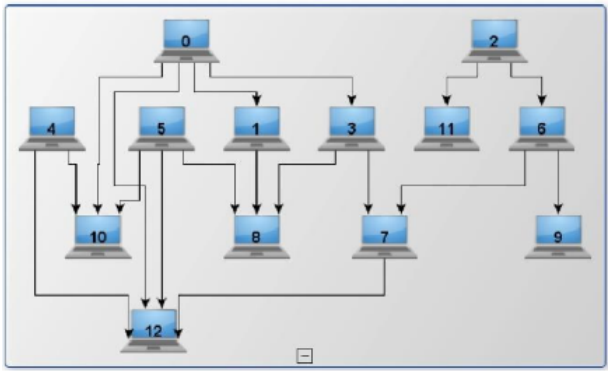

Fig. 5. Unstructured Peer to Peer

5. DESIGN CHALLENGES

P2P systems offer a large number of advantages over traditional Client/Server systems. But they are not all silver bullets. They have their own set of problems. Some challenges and problems while designing P2P systems are explained in detail below.

5.1. Security

The intrinsic structural and operational differences between the P2P system and the classic client-server paradigm cause many security solutions developed for the latter to be almost useless for the former. And in most cases, a major adaptation rework is required.

In P2P systems the nodes are dynamic and also the peers don't trust each other, and thus achieving a high level of security in P2P systems is more difficult than traditional Client-Server Systems. Normal security mechanisms to protect systems from intruders and attacks such as firewalls can not protect P2P systems since they are essentially globally distributed and also these mechanisms can inhibit P2P communication. Therefore new security concepts are required that allow interaction and

processing in P2P systems.

Research have been concentrated on specific security points - providing anonymity to users and data [17]. Work has also been done towards establishing and managing trust relationships among users. Solutions focused on the avoiding of Denial of Service (DoS) attacks and the abuse of multiple identities (Sybil attack) [18] have also surfaced in the recent years. In general, solutions exist to handle various security problems with P2P systems - and most of the times these issues are interlinked with other challenges of P2P systems (such as fairness).

5.2. Reliability

A Reliable system is a system which can be recovered after some failure or faults. The factors necessary for a reliable system are data replication, node failure detection and recovery, existence of multiple guarantees for location information to avoid a single point of failure and the availability of multiple paths to data. Corrupted or fake files also diminish the reliability of the P2P service due to downloading (and subsequent redundancy by spreading) of useless contents [19].

The advantage of scalability generates a new risk. Since the shared file is no longer at a single trusted server location, peers may offer a corrupted version of a file or parts of it. This is referred to as *poisoning* [20]. When the number of corrupt chunks is large, the distribution of the original file is disrupted.

For providing reliability in unstructured networks, dynamically adding redundant links to the systems is known to work. In this way no single disconnection should cause disconnection of the whole system. In a structured network, messages need to be routed, and the routing stage has to be updated automatically when a node enters or leaves the network. To maintain reliability for such conditions, nodes must use part of their bandwidth to maintain the routing states. Further techniques can be used to optimise this so that the cost is manageable in the normal operating conditions.

5.3. Flexibility

One of the most important services a P2P system provides to its users is the ability to join and leave at their will. Modern P2P systems are characterised by decentralised control, large scale and extreme dynamic environments. Adaptation and self-organisation is very important for such dynamic environments. In most cases, anonymity of data, user, and the user's information is also crucial for the system. Offering flexibility with anonymity and other non-essential features (faster sequencing, downloads, etc.) is a challenge for P2P systems.

In modern unstructured P2P systems like Kazaa [6], queries are forwarded only to super-nodes. The super-nodes maintain a list with the file names of their connected peers, avoiding overloading all peers of the system. In standard structured P2P systems, static identifiers are assigned to nodes and based on these distributed data structures are constructed. So, the overlay network structure is determined by these identifiers and in turn any self-organization of the system is prevented. Standard structured systems based on distributed hash tables should perform look-ups quickly and consistently while nodes arrive and leave the system.

5.4. Load Balancing

Load balancing in peer-to-peer networks is a mechanism to spread various kinds of loads like storage, access and message forwarding among participating peers in order to achieve a fair or optimal utilization of contributed resources such as storage and bandwidth. The management of the distribution of the data and the computation across the network is very important for the efficient functioning of the

network. In the context of P2P systems, the initial works on load-balancing relied on consistent hashing [21]. It was proposed in 1997 to deal with load-balancing in web caches with minimal movement of data.

Consistent hashing was used to achieve storage load-balancing in many early distributed hash table P2P networks proposed around early 2000s. We employ, the same, a Distributed Hash Table (DHT), where all the data that is stored in the network is mapped to a unique identifier. To distribute the load, the identifier space is partitioned into a wide range of subsets and is distributed among the nodes, and each node is responsible for maintaining its portion of the partition space.

The concept and handling of load balancing is interlinked with fairness of the P2P system.

5.5. Fairness

Unfairness is a fundamental challenge of P2P networks. Many peers free-ride by account for large portions of download bandwidth and little or no upload bandwidth. If a P2P system can implement fairness, i.e. a peer receives bandwidth equal to what it contributes, the system would be able to guarantee a certain level of experience to each of the user (and exactly equivalent of their contribution in a perfectly fair scenario).

The fairness in P2P systems is difficult to achieve for three major reasons.

1. there is no central authority controlling the distribution of resources
2. the amount of bandwidth resources available is not known in advance
3. peers cannot be trusted upon to specify their resources honestly

The most popular fairness solution is the Tit-For-Tat (TFT) heuristic used by BitTorrent [6]. TFT splits a peer's upload bandwidth equally among a subset of neighboring peers for a time duration called a round, then adjusts this subset each round based on estimates of their bandwidth rates from the previous round. There have been attempts at improving fairness over TFT. Block-based TFT [22, 23, 24], used by BitTyrant [25] peers among one another, attempts to improve on TFT by augmenting TFT with hard limits on the amount of data one peer can owe another. PropShare [26] also attempts to improve on TFT by using the Proportional Response algorithm [27] to split the peer's upload bandwidth in proportion to the contribution received from its neighbors in the previous round.

6. CASE STUDIES

6.1. Gnutella

Gnutella [28, 29] is among the first of many decentralized technologies to reshape the Internet and revolutionize our approach towards network applications. It put to stop the traditional thought process of resorting to a hiearchical client-server system for any kind of network applications. A decentralised system has a lot of advantages that a traditional client-server system fails to deliver and Gnutella is a point of proof that such technologies, while young, are viable.

An interesting fact is that, Gnutella in itself is not a software. Its a language of communication, a protocol. To be precise, Gnutella is a file sharing protocol that defines the way distributed nodes communicate over a p2p network. Any software that supports it is a Gnutella-compatible software. The term Gnutella nowadays is also used to refer to the virtual network of Internet accessible hosts running Gnutella-speaking applications and a number of smaller, and often private, disconnected networks.

6.1.1. Architecture

The Gnutella network is based on a completely decentralized architecture, making it a true P2P network that does not involve central servers for authentication, indexing, assisting in file searches, or any of the network operations whatsoever. This architecture eliminated the problems associated with a centralized server, including traceability and scalability, thus allowing the network to grow as users connect to it indefinetely, theoretically.

Each peer in the Gnutella network is called a *Gnutella servent*. As previously discussed, this term is used to highlight the capability of the nodes to act as a client and a server for the network. Thus, all of them are considered equal to one another, capable of transmitting searches throughout the network and sending and receiving files.

6.1.2. Joining

In order for a new Gnutella servent to join the network overlay, it must know the IP Address of at least one other servent already present in the Gnutella network. Gnutella servents tries to connect to several Internet Protocol (IP) addresses that were installed with the client software. To further facilitate this bootstrap operation, the new servent receives the immediate neighbours of the client whose address it knows and it further tries to connect with them. After the servent has connected to the network and is aware of the IP addresses of nearby peers, it is capable of utilizing the features of the network, such as searching for files and downloading shared files.

Towards this end, the protocol uses **PING** and **PONG** (group membership) messages. A node joining the network initiates a broadcasted PING message to announce its presence. When a node receives a PING message it forwards it to its neighbors and initiates a back-propagated PONG message. The PONG message contains information about the node such as its IP address and the number and size of its shared files.

6.1.3. Sharing

Since Gnutella follows a decentralised network architecture, the node indexes the files it is going to share locally. Whenever a query for a particular content comes to this node, it searches for matches with the index and then responds. This means that all shared files are stored and hashed locally.

6.1.4. Search and Fetch

When a servent initiates a file search in the network, it starts by querying all the servents it is connected to. In turn, those servents send the query to the servents they are connected to. This is similar to the *Query Brodcast Flooding* discussed earlier in Section 3. This process continues as search results are sent to the requesting servent. For downloading files from the network, there is a somewhat similar process that happens. As the process for downloading a file begins, the servent requesting the download contacts the servent that answered its search request. If that download does not start, the download request is passed from the reporting servent to the other servents that it is connected to. Eventually, the download request will be answered by a servent, which contacts the servent that forwarded the original download request. This process is depicted in Fig. 6. The messages used for this purpose are:-

- **QUERY** and **QUERY RESPONSE** (Search) messages:- QUERY messages contain a user specified search string, each receiving node matches against locally stored file names. QUERY messages are broadcasted. QUERY RESPONSES are back-propagated replies to QUERY messages and include information necessary to download a file.

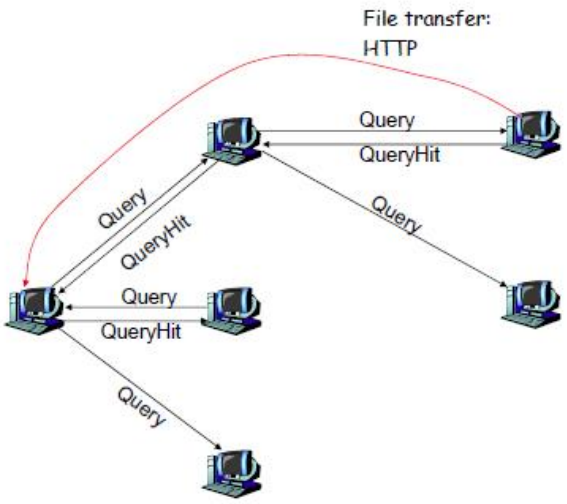

Fig. 6. Querying in Gnutella

- **GET** and **PUSH** (File Transfer) messages:- File downloads are done directly between two nodes using GET/PUSH messages, where the GET message contains request for a file and the PUSH message has the actual contents of the requested file.

As can be seen, such uncontrolled flooding of QUERY messages might result in a exponential increase in the network traffic, owing to which each such message is limited by TTL (Time-to-Live). Typically, TTL is 7 hops. But, since Gnutella is a shallow network, a servent has a lot of neighbouring servents and because of this, in 7 hops a good percentage of the network is covered.

6.1.5. Properties

We now discuss some of the important properties of the Gnutella p2p network:

- **Decentralisation:** The system is highly decentralised and each peer is truly equal in nature.
- **Scalability:** In theory the Gnutella network should be highly scalable as each peer is equal and ideally there should be no central point for congestion of traffic. However it was observed that flooding of queries hogs the network bandwidth causing huge scalability issues. This also leads to fragmentation of the network into smaller clusters as certain nodes become unresponsive to huge amounts of network bandwidth utilization. Furthermore users aren't forced to store or share files. This leads a lot of users simply being present in the networth without any contribution, which leads to huge performance issues and lower probability of being able to find the required files.
- **Anonymity and Security:** Anonymity is preserved to some extent by virtue of the system being a true P2P system and messages can only come through local nodes. However it is possible for nodes to store information regarding incoming and outgoing traffic that may affect anonymity to some extent. The security of the system is quite low as the system is prone to unwanted flooding, malicious attacks, hijacking of queries and denial of service attacks.
- **Lookup Completeness:** The search is incomplete as it may not search across all clients due to the presence of TTL. Thus it is possible for the search query to not yield any result despite the existence of the required file in the system.
- **Fault Resilience:** The system is quite tolerant to the fault of a node. The system is still capable of communicating with each other without any problem and the system will continue to work correctly without issues, however the faulting of certain nodes can cause performance issues or inability to obtain files present in that node as there is no inherent replication of data in the system.

6.2. BitTorrent

BitTorrent[30, 31] is a decentralised communication protocol for Peer-to-Peer file sharing system. Unlike other P2P file sharing systems, BitTorrent does not focus on storing files efficiently in the P2P system and does not provide a method to search for files in the system. However it provides a protocol for highly efficient file fetching. BitTorrent leverages the concept of flash mobs, that is a large number of users interested in the same file at a particular time. In traditional P2P systems the download and upload speed of the users act as bottlenecks in the transfer of a file. In BitTorrent, however the file is broken into multiple chunks and spread among the swarm of users who wish to download the file. Thus each user who wishes to download the file will download different chunks of it from multiple users simultaneously thus, the upload limit of a particular user does not act as a significant bottleneck. We shall see in more detail how the BitTorrent system manages this.

6.2.1. Architecture

The BitTorrent architecture is a distributed hybrid P2P system. The users who own the BitTorrent Client are the peers in the system, other than that we have a central server known as a tracker that facilitates the file transfer amongst the various peers in the network.

The peers in the BitTorrent system come under two main types:

- **Seeder**: A seeder is any user who contains the full file downloaded and can upload chunks of the file to users who are attempting to download the file.
- **Leecher**: Any user who is currently downloading the file and may or may not upload the file is called a Leecher. The efficiency in BitTorrent's fetching derives from the presence of multiple leechers who simultaneously upload already downloaded chunks to other leechers. A leecher who does not upload any data is called a **Freeloader**, and the presence of freeloaders harms the system's efficiency.

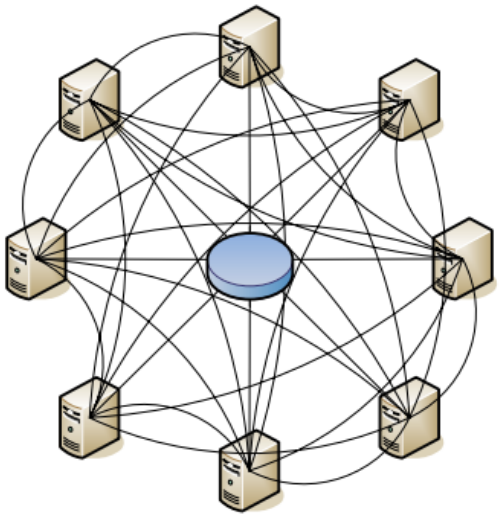

Fig. 7. BitTorrent Architecture [30]

6.2.2. Sharing

In order for a user to upload a file through BitTorrent the user needs to first publish the file. In order to do so a user must create a torrent file which contains the following information:

- URL of a tracker
- Metadata:
  - Name of the file
  - SHA-1 Hash of each Piece of the file
  - Piece Size
  - Number of Pieces

The BitTorrent client system does not allow one to search for torrents for particular files, instead one can email the torrent file after publishing it or host it on a website where users can

take the torrent file and download from their BitTorrent Client.

6.2.3. Join

Once the torrent for a file has been obtained, the user must contact the tracker server from the torrent and periodically needs to give the information regarding its IP Address, Port number and the amount of data it has already uploaded and downloaded. The tracker in return sends the information of the other users who are seeding or leeching the corresponding file. Thus the user joins the swarm of users who are trying to download the same file, and then uses the information it has regarding the distribution of chunks amongst the various user to then download and upload chunks of the file it requires. It is important to note that every swarm requires atleast one seed node who contains the full file downloaded to ensure that the full file is available to download.

6.2.4. Fetch

In this section we discuss the algorithms used by BitTorrent to decide which piece of the file should be downloaded by the client and from which peer. Every file is divided into pieces which are typically 256Kb in size, these pieces are further divided in to sub-pieces which are 16Kb in size, thus 16 sub pieces.

Our main goal is to download different pieces from different peers and choose these pieces in an efficient manner so that the fetching of the file is done as quickly as possible. This is done so through the following policies:

1. **Strict Policy**: This policy ensures that if a sub-piece is currently being downloaded, then only other sub-pieces of that particular piece will be downloaded until the rest of that piece is downloaded entirely.
2. **Rarest Piece Policy**: While choosing which piece to obtain the peer chooses the piece which is present least in number amongst its peers. This is beneficial for the following reasons:
   (a) This policy allows the pieces that are less in quantity to propagate through the system, thereby allowing all pieces to spread through the network in a sort of equivalent fashion thus all pieces have multiple peers which contain the piece allowing further users to download it quickly.
   (b) The churn in the system may cause rare pieces to get lost over time, this becomes a problem especially if the seed node itself leaves the network. Thus prioritising rare pieces reduces the chances of this happening.
3. **Random Piece First Policy**: The Rarest piece policy chooses pieces that would take some time to get downloaded as they are present amongst very less peers. This causes a problem as it would take too much time until the client is able upload data thus the benefit of the client joining the network will not be observed quickly. In order to help with this, the very first piece is chosen randomly so that it can obtain the first piece at a faster rate than the rarest piece policy and immediately help other peers in downloading. After this the policy switches to the Rarest Piece policy.
4. **End Game mode**: Towards the end of the download of the file it often happens that the download gets stalled due to the low bandwidth of a peer causing very low download rates. The BitTorrent client generally maintains a queue of requests for different sub-pieces so that

it can simultaneously request different sub-pieces from different peers (typically 5 requests are done in parallel). When it occurs that all the remaining sub-piece requests are requested from a single peer (indicating a possible slow bandwidth peer), then all the remaining requests are broadcasted to all peers to quickly finish the downloading of the final sub-pieces of the file.

We have discussed the policies which decide who to download from. These policies are done so that the client can download as efficiently as possible and it will send requests to ensure that. However, each client must also prioritise who they upload data to. Ideally we would like to do it such a way as to penalise Freeloaders in the system, i.e. clients who only download but do not upload. This is done via a *tit-for-tat strategy* also known as *choking*. The following are the main parts of this strategy:

- **Choking Algorithm**: A refusal to upload to a particular peer is called Choking. At any particular time amongst all the requests the client typically unchokes a constant number of peers. Amongst these peers, the client prioritizes those peers who have recently uploaded to the client, thus forming the essence of the *tit-for-tat* strategy. Amongst the peers who have recently uploaded to the client, the peers are prioritised based on the download speed so as to ensure fast upload which is not bottlenecked by a peer's low download rate.

- **Optimistic unchoking**: Intermediately BitTorrent chooses an additional peer to be unchoked temporarily to observe if it has a better performance over time based on its upload/download rate. If this peer does infact perform better, then it replaces one of the unchoked peers. This is just to improve upon having missed on any good connections while the initial unchoking phase.

- **Anti-Snubbing**: It is obvious to see that Freeloaders and peers with low bandwidth are penalised, causing the download of the files in their respective systems to be significantly slower than responsive high bandwidth peers. However it can occur that a particular node becomes snubbed, i.e it has been choked by all its peers. In retaliation the node also stops uploading to all its peers and it performs *optimistic unchoking* at a higher rate to help speedup the download process.

6.2.5. Properties

We now discuss some of the important properties of the BitTorrent P2P network:

- **Centralisation:** The system follows a centralised model with Tracker keeping track of peers.

- **Scalability:** BitTorrent is a robust and scalable system which ensures high uplink bandwidth utilization. The system ensures that bandwidth of the origin server is bounded in spite of increasing number of nodes. It is ensured that server sends unique packets to the network quicker than what can be diffused by the peers.

- **Anonymity and Security:** Anonymity is ensured in the BitTorrent traffic. The IP that the swarm sees is not the actual IP address. The security of the systems is moderate. Centralized Tracker manage file transfer and allows more control which makes it much harder to fake IP address, port number, etc. The user can also encrypt and anonymize using proxy/tunneling to add another layer of

protection. Although this would slow down the connection.

- **Lookup Completeness:** The system provides are guarantee to locate data. Although the performance is only guaranteed for content with high number of seeders, or simply put popular content.
- **Fault Resilience:** The system is not affected by fault of a node significantly as long as there is at least one seeder. Similarly, one or more trackers should always exist in the system to propagate peers information. Choking can also be avoided by changing the peer that is choked once every 10 seconds.

6.3. Kazaa

Kazaa is a P2P file sharing application that uses the FastTrack protocol. This application came out after the demise of Napster. It is known as a second generation P2P file sharing system as it was developed to over come the shortcomings of the earlier First generation P2P protocols, in particular it is a combination and thus an improvement over the Gnutella and Napster systems.

In Napster the main fault was that all clients have to contact a central server. This allows very efficient and fast search, however, this can be a point of fault as the failure of the server will cause the failure of the system. On the other hand Gnutella had a fully decentralised P2P system, which had very inefficient search but was very robust to churn in the system.

6.3.1. Architecture

The Kazaa Architecture leverages heterogeneity to impose a form of hierarchy amongst the nodes. However there is no central server, and all nodes, irrespective of hierarchy, are a part of the system involved in downloading and uploading of files. The system is unstructured in nature with minimal amount of centralisation in the form of nodes which hold some amount of responsibility. The nodes are thus classified into two types, Ordinary Nodes and Super Nodes. Super nodes are chosen based on the capability of the node and a certain *reputation* that it has attained in the system.

Every node that joins the system is assigned to a Super Node. This Super Node consists of information such as the IP Address and ports of all the peers that are connected to it, as well as the metadata of the files that these nodes have published. Thus the network is hierarchical in nature and the Super Nodes form a distributed overlay network. This network is robust to churn as even if a Super Node leaves the network, its set of peers can be distributed among all the other Super Nodes.

6.3.2. Join

Whenever a user activates Kazaa, node connects to a bootstrap node that is always assumed to be present, and this node informs the user if it is a Super Node or an Ordinary Node based on a certain *reputation* that it calculates based on the node's bandwidth, availability, CPU power, etc. Once this has been done, the node receives a list of all the active Super Nodes present in the network at the moment. If the Node is an Ordinary Node, then it sends UDP requests to some of the Super Nodes it has in its list, typically 5 and then establishes a TCP connection with one of the Super Nodes, and it informs the Super Node of its IP Address and Port. File fetch requests by the node then go through this Super Node. For each TCP connection the Super Node and Ordinary Node exchange encryption keys to ensure security. This is how a new node joins the Kazaa network.

6.3.3. *Publish*

Once a node joins the Network it needs to publish the set of files that it decides to share with the Kazaa Network. Every node does it through the Super Node it is connected to. It sends the following file meta data to the Super Node:

- *File Name*
- *File Size*
- *File Hash:* To identify identical copies of the same file, helps in parallel downloading of a file.
- *File Descriptors:* To help in finding the file when another node queries for it.

6.3.4. *Search*

When a particular Node requires a file, it sends the search query to the corresponding Super Node that it is connected to. The Super Node then floods the network of Super Nodes looking for this particular file, with a particular TTL (Time-To-Live). It then returns the information gathered back to the querying node. Thus the querying node now has the search results which include the IP addresses of the nodes where they are present and the corresponding meta data. The user can then choose the file to download and even download the same file from multiple nodes if present. We observe that flooding the network of Super Nodes is much more efficient than the flooding that was done in the Gnutella network, which flooded all nodes in the system.

6.3.5. *Fetch*

The fetching in the system is fairly straight forward, once the information regarding the meta data of the files and the IP addresses of the nodes, are returned to the querying node, the node initiates a request directly with that node to download the corresponding file.

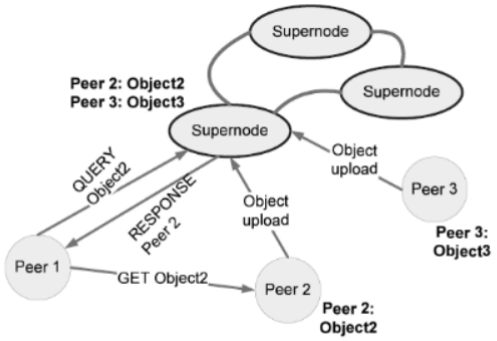

Fig. 8. Search and Fetch in Kazaa

It is important for us to observe that the File Hash that is present allows us to detect the presence of multiple copies of the same file in the system. This allows us to download in parallel from multiple nodes different chunks of the same file, which is done through the HTTP byte-range header. This allows us to efficiently download the file.

6.3.6. *Properties*

We now discuss some of the properties of the Kazaa System:

- **Decentralisation:** The system is decentralised in the perspective of search and querying as the Super Nodes form a purely decentralised overlay network. However there is hierarchy present amongst the nodes as the Ordinary Nodes perform queries through the corresponding Super Nodes.
- **Scalability:** The Network is fairly scalable, however the certain Super Nodes may get overloaded with the presence of too many Ordinary Nodes that it may not be able to handle efficiently due to its own limitation in resources. This could serve as a bottleneck in scaling the system to a larger number of nodes.

- **Anonymity and Security:** The Ordinary Nodes are anonymous with respect to each other, however the Super Nodes do have access to all the IP Addresses of the nodes under them. It is still more anonymous than the Napster network in this manner.

  Security is not very high, however malicious attacks through flooding largely affect the Super Nodes and can be deal with them, thus Ordinary Nodes aren't as much at risk. The presence of encryption between the nodes also helps preserve the security of the system.

- **Lookup Completeness:** Lookup is incomplete as it is done through flooding. However it is much more efficient as the flooding spans only the Super Nodes of the system.

- **Fault Resilience:** The system is pretty fault resilient as it can tolerate large amounts of churn since it is a largely decentralised system. When an Ordinary Node fails, the Super Node, can just remove its information, and if a Super Node fails, its corresponding Ordinary Nodes can be redistributed with the help of the Bootstrapping node.

7. CONCLUSION

System Architecture depends upon the applications requirements. Till now major architectures have been Client-Server Systems. But with the emergence of technologies like Napster, Torrent, IPFS, things have started to change. Due to cons of the standard Client-Server model, efforts are being put into the exploration of P2P systems.

In the last few decades, a lot has been done for the development of scalable, fault-tolerant and autonomous P2P systems. Many architectures, each with their pros and cons have emerged. Depending upon the requirements for their particular applications, developers should choose a topology for the platform that matches their needs.

In pure peer-to-peer networks every peer is given equal responsibility irrespective of its capabilities. From this it is evident that this can lead to reduction of performance as less powerful nodes are added. Also, pure p2p systems lack manageability since every peer is its own controller.

Unstructured pure p2p systems use blind flooding for search. This creates the problem of scalability as large scale systems require a large number of messages to be exchanged. Scalability issues can be solved by using structured systems. But in standard structured systems it is hard to maintain the structure required for routing in a very transient node population, in which nodes join and leave at a high rate.

Pure p2p systems are fault tolerant, since failure of any particular node does not impact the rest of the system. Hybrid p2p systems solve the manageability problem of pure p2p systems, so that the control servers act as a monitoring agent for all the other peers and ensures information coherence. Regarding distributed indexing and centralized indexing systems, drawbacks associated with centralized indexing systems are a single point of failure when the central server goes down and also not being scalable because of the capacity of the server to maintain database and to respond to queries. Distributed indexing systems alleviate these shortcomings by using super-peers. Although super-peer clusters are efficient, scalable and manageable, in order to avoid a single point of failure for the clients in a cluster, some policies of super-peer redundancy should be taken into account. As in the case of failover super-peer, these strategies should be able to take over the job of the primary super-peer.

8. REFERENCES

[1] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A scalable content-addressable network," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 161–172, Aug. 2001.

[2] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, 2001, SIGCOMM '01, p. 149–160, Association for Computing Machinery.

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, pp. 46–66, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[4] Matei Ripeanu, Ian T. Foster, and Adriana Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *CoRR*, vol. cs.DC/0209028, 2002.

[5] B. Yang and Hector Garcia-Molina, "Designing a super-peer network," in *Proceedings - International Conference on Data Engineering*, 04 2003, pp. 49 – 60.

[6] Nathaniel S. Good and Aaron Krekelberg, "Usability and privacy: A study of kazaa p2p file-sharing," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2003, CHI '03, p. 137–144, Association for Computing Machinery.

[7] Wojciech Galuba and Sarunas Girdzijauskas, *Distributed Hash Table*, pp. 903–904, Springer US, Boston, MA, 2009.

[8] Miguel Castro, Manuel Costa, and Antony Rowstron, "Should we build gnutella on a structured overlay?," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 131–136, Jan. 2004.

[9] Domenico Talia and Paolo Trunfio, "Enabling dynamic querying over distributed hash tables," *Journal of Parallel and Distributed Computing*, vol. 70, no. 12, pp. 1254–1265, 2010.

[10] H. A. Ali and M. A. Ahmed, "Hprdg: A scalable framework hypercube-p2p-based for resource discovery in computational grid," in *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*, 2012, pp. 2–8.

[11] Adriana Iamnitchi, Ian Foster, and Daniel Nurmi, "A peer-to-peer approach to resource location in grid environments," 04 2002.

[12] Y. Tan, "A multi-agent approach for p2p based resource discovery in grids," in *2009 International Joint Conference on Artificial Intelligence*, 2009, pp. 43–45.

[13] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM STOC*, 02 2001.

[14] Yuhui Deng, Frank Wang, and Adrian Ciura, "Ant colony optimization inspired resource discovery in p2p grid systems," *The Journal of Supercomputing*, vol. 49, pp. 4–21, 07 2009.

[15] Vasilios Darlagiannis, *21. Hybrid Peer-to-Peer Systems*, pp. 353–366, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[16] Beverly Yang and Hector Garcia-Molina, "Comparing hybrid peer-to-peer systems," in *Proceedings of the 27th International Conference on Very Large Data Bases*, San Francisco, CA, USA, 2001, VLDB '01, p. 561–570, Morgan Kaufmann Publishers Inc.

[17] Clay Shields and Brian Neil Levine, "A protocol for anonymous communication over the internet," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2000, CCS '00, p. 33–42, Association for Computing Machinery.

[18] John R. Douceur, "The sybil attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Berlin, Heidelberg, 2002, IPTPS '01, p. 251–260, Springer-Verlag.

[19] Kenji Leibnitz, Tobias Hossfeld, Naoki Wakamiya, and Masayuki Murata, "Peer-to-peer vs. client/server: Reliability and efficiency of a content distribution service," 01 2007, vol. 4516, pp. 1161–1172.

[20] Nicolas Christin, Andreas S. Weigend, and John Chuang, "Content availability, pollution and poisoning in file sharing peer-to-peer networks," in *Proceedings of the 6th ACM Conference on Electronic Commerce*, New York, NY, USA, 2005, EC '05, p. 68–77, Association for Computing Machinery.

[21] David R Karger, Eric Lehman, Tom Leighton, Matt Levine, Daniel Lewin, and Rina Panigrahy, "Relieving hot spots on the world wide web," in *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pp. 654–663.

[22] Ashwin Bharambe, Cormac Herley, and Venkata Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," 04 2006.

[23] Karthik Tamilmani, Vinay Pai, and Alexander E. Mohr, "Swift: A system with incentives for trading," 2004.

[24] Seung Jun and Mustaque Ahamad, "Incentives in bittorrent induce free riding," in *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems*, New York, NY, USA, 2005, P2PECON '05, p. 116–121, Association for Computing Machinery.

[25] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani, "Do incentives build robustness in bittorrent?," in *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, Cambridge, MA, Apr. 2007, USENIX Association.

[26] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, New York, NY, USA, 2008/// 2008, ACM, SIGCOMM '08, pp. 243 – 254, ACM.

[27] Fang Wu and Li Zhang, "Proportional response dynamics leads to market equilibrium," in *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 2007, STOC '07, p. 354–363, Association for Computing Machinery.

[28] Andy Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly amp; Associates, Inc., USA, 2001.

[29] "Foreword contributor," in *Securing Im and P2P Applications for the Enterprise*, Paul L. Piccard, Brian Baskin, Craig Edwards, George Spillman, and Marcus H. Sachs, Eds., p. x. Syngress, Burlington, 2005.

[30] Jahn Arne Johnsen, "Peer-to-peer networking with bittorrent," 2005.

[31] Matei Ripeanu, Miranda Mowbray, Nazareno Andrade, and Aliandro Lima, "Gifting technologies: A bittorrent case study," *First Monday*, vol. 11, 04 2006.