Peer-to-Peer Networks and Issues Involving Them

CS625 : Advanced Computer Networks
Era Jain(Y9227209)
Dept. of CSE

Mentor:
Prof. Dheeraj Sanghi
Dept. of CSE

Abstract

The following study reviews peer-to-peer (P2P) networks and the various issues involving them. It starts with a comparison between P2P and client-server paradigm, giving details on the P2P architecture and highlights some important P2P file sharing protocols. It talks about various indexing routing techniques used in a P2P network. Next it addresses the problem of free riders and discuss several kinds of algorithms proposed to counter it. Lastly it discuss the NAT traversal problem and several security issues involved in a P2P network.

1 What are peer-to-peer networks?

1.1 History

Peer-to-peer systems is not a very new concept. Many companies and universities have been using this kind of architecture for more than 30 years. The very first P2P kind of a system came out in late 1960s. It was called the ARPANET and it connected UCLA, Stanford Research Institute, UC Santa Barbara and the University of Utah, where every host on the internet could FTP or telnet into every other host. In 1979, two graduate students from Duke University and one from University of North Carolina developed Usenet based on UUCP (Unix-to-Unix-copy protocol), where each unix machine could connect with another machine, exchange files with it and disconnect. They used Usenet to exchange data between the two schools.

P2P networks became very popular with the advent of Napster in May 1999, developed by a freshman at NorthEastern University. All members of Napster could search the local hard disks of other members for desired mp3 files with the help of a central server which kept record of the files belonging to each member. This way the members could share mp3 files with each other. A lawsuit was filled against Napster by a recording company following which it closed down in July 2001, not before resetting the trend of P2P systems.

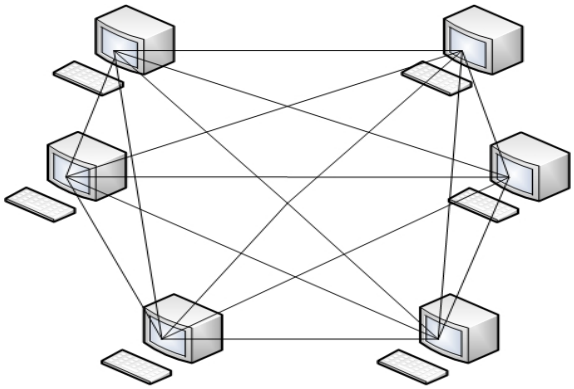

Figure 1: A peer-to-peer network

1.2 Definition

Peer-to-peer network refer to a concept where two equal peers can collaborate with each other to exchange files etc. without the need of a central authority to coordinate them. There are three important characteristics of a P2P network:

1. Each element of the system can act as a client as well as a server. All nodes share their resources such as files, bandwidth, storage and processor cycles in order to provide service to other peers in the system.

2. Decentralization - There's no central authority. All peers are equal. Each peer provides services to other elements as well as request services from other elements.

3. Autonomy - Each peer can decide autonomously when and to what extent it will provide resources to other nodes

2 Peer-to-peer vs Client/Server Model

Peer-to-peer models are a counterpart to client/server systems where a central server controls the entire network and is responsible for providing services to the clients. Client/Server model has several limitations where P2P models could perform better. Following is a brief comparison of the two kinds:

Scalability

- P2P system leverages the processing and storage capacity of all peers
- Addition of a user to the system increases the load on system but also increases the processing and storage capacity whereas in client/server model, load on the server keeps on increasing which might result in bottlenecks and single point of failure eventually.
- In P2P systems there's no typical need to update any central server to deal with more users.

Robustness and Reliability

- Centralized servers are highly available whereas users in a P2P network can leave the system at any time unreliably. Data replication techniques can be used to counter this.
- If a central server crashes, huge loss is incurred whereas a single peer leaving the network in P2P systems would not affect the system that much on a whole.

Performance

- P2P distributed computing (grid computing) can perform better in tasks that can be done in parallel however for tasks involving database queries, a centralized database at a central server would be faster
- Performance can be highly affected by lack of cooperation in P2P systems. Free Riding is highly prevalent in most of the P2P systems these days (to be discussed later)

Energy Consumption

- Server farms concentrate high reserves of energy in a single geographic location whereas in P2P networks too peer has to be up and running all the time

Security Issues

- Central servers are more prone to attack since all data is stored at a single place whereas this risk factor is less in a P2P network
- P2P networks can encounter other kinds of attacks. A malicious peers can enter the overlay and drop routing requests, upload malicious content, several peers can get together to prevent service to a node or bombard it with queries or indulge in Sybil Attack(to be discussed later)

3 Classification of P2P architecture

P2P systems can be classified based on: 1. *Degree of Decentralization* and 2. *Degree of Structure*

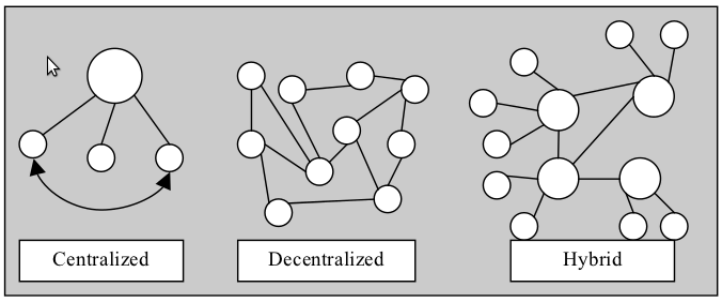

Figure 2: Degree of Decentralization

3.1 Degree of Decentralization

In peer-to-peer systems at least some elements must be decentralized. Classification can be done on the extent of decentralization:

1. Centralized: All important functions are performed by a central server which keeps track of all the nodes in the system and the resources provided by each of them. All the queries are addressed to the server however peers connect to each other directly without any intervention from its side. E.g.-Napster

2. Pure Decentralized: These are self organizing systems where the peers are completely responsible for the functioning of the system. There's no central component at all. These are highly scalable and have high fault tolerance. However QoS is pretty low, because only a limited proportion of the network can be reached at times. E.g.- Gnutella, Freenet, Chord etc.

3. Hybrid: These have both pure centralized as well as pure decentralized components thus combining the advantages of both types of systems like efficient node search, scalability and avoiding the drawbacks such as bottlenecks and low QoS. Several nodes which have higher capacities and resources are promoted to be the super nodes which are responsible for complete functioning and organization of the system. Instead of having a single server, work load is distributed among these super nodes. E.g.- KaZaA, Gnutella 0.6

3.2 Degree of Structure

Depends on the positioning of nodes and data in the network

1. Unstructured: Nodes are present in an adhoc manner with no fixed positions. Data is distributed randomly among nodes, location of data is not connected to the network topology. This results in inefficient search methods such as query flooding model. Being so disorganized, there's no guarantee that the data available in the network could be found for sure in some query. However these systems are quite resilient, unaffected by peers leaving at any time. E.g.- Napster, Gnutella, KaZaA

2. Structured: Nodes placed in a structured manner, data can be located more easily which increases scalability. Routing tables are distributed, thus successful search can be achieved in a few hops. Its more reliable however in case of transient nodes the structure needs to be reconfigured constantly. E.g.- Chord, CAN, Tapestry (all implement DHT)

3.3 Classification based on services provided

1. File Sharing - Napster, BitTorrent, Gnutella

2. Distributed Computing - SETI@home (Data distributed among members who compute and send back the results to the main computer)

3. Storage Services - FreeNet, OceanStore

4. Cooperative P2P - Instant messaging, chat, online games

5. P2P Multimedia Service - skype (follows P2P as well as client-server paradigm)

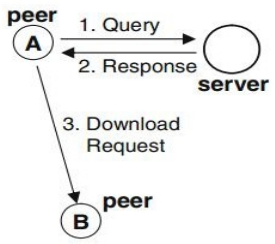

Figure 3: Napster

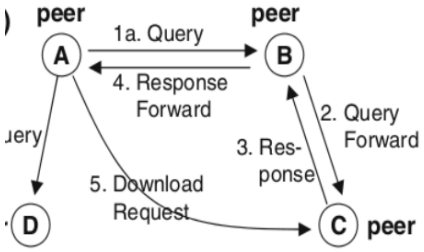

Figure 4: Gnutella

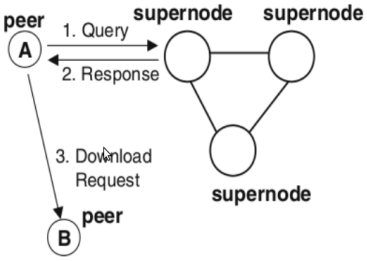

Figure 5: Gnutella 0.6

4 Some famous P2P protocols

4.1 Napster - A centralized approach

A central server maintains an index with meta data of all files in the system, a list of all registered peers and a list showing the files that each peer shares. All the search queries are directed to this server. Suppose, A wants to search for some file, it will query the central server(Figure 3). The server returns a list of some peers that hold the file. Peer A selects which peer to download the file from, say peer B. A then directly connects to B and downloads the file from it. The advantages with Napster like protocols is that they are easy to implement however they are not scalable, have a single point of failure and are vulnerable to DDos attacks.

4.2 Gnutella - A fully decentralized approach

Every peer in this system has a local Index pertaining to the data it has and is unaware of the indices at other peers. Whenever a new peer joins the system it sends a PING message to a list of peers hosted at *http://gnutellahosts.com*. Peers respond to this PING with a PONG message which is routed back along the path of the PING. PONG also forwards the PING to additional peers to which it has the links after decrementing the TTL by 1. The new peer will collect all the PONGs which have the IP addresses of the respective peers.

Search is done using *Query Flooding method*(Figure 4)). To find a specific file, the user enters a search term into the Gnutella interface. His peer then sends the QUERY message to all the peers it discovered through PINGs. If a match is found at some peer, a QUERY HIT message is sent back to the user's peer via the path of the query. This message contains the IP address of the peer with the file and the matching file name. Newer versions allow the user to set a time out for the search. The user peer after receiving a QUERY HIT downloads the file from the respective peer via http request. If the peer with the file is behind a firewall, then it drops this http request. In this case the user peer issues a PUSH request which is broadcasted through the network until it gets to the recipient. The recipient then connects to the user peer and sends the file.

Issues with Flooding is low search efficiency and arrival of duplicate query messages at several nodes. Another method called the *R*ing Expansion could be used where a query is sent with a low TTL value which keeps on incrementing unless the search succeeds. *M*ultiple Random Walk is another algorithm which is used. In a random walk a peer randomly selects a neighbor and forwards it a query. In order to reduce search time, a peer can start many random walks simultaneously.

4.3 Gnutella 0.6 - A hybrid approach

It has a hierarchical structure comprising of leaf nodes and ultrapeers. When a new peer enters the system, it is kept as a leaf at the edge of the network. A leaf is not responsible for any routing, it simply sends queries to its ultrapeers. Peers with high capacities and processing abilities are promoted to ultrapeers and are responsible for routing the queries using the *Query Routing Protocol(QRP)*. All leaves connect to these ultrapeers. Usually, each leaf connects to 3 ultrapeers, and each ultrapeer connects to more than 32 other ultrapeers. Also due to a high out degree the maximum number of hops in the system is reduced to 4 which results in high scalability and search efficiency. When a leaf tries to connect to another leaf via http request to download a file from it, the presence of a firewall around the target leaf will drop this http request. To prevent this the ultrapeers of a leaf act as proxies for it known as *PUSH proxies*. So the leaf instead of connecting to the target leaf, connects to its PUSH proxy which in turn sends a PUSH request to the target leaf on behalf of the sender leaf. The target leaf then initiates a connection with the sender leaf.

Query Routing Protocol: Each leaf will create its respective Query Routing Table(QRT) which contains a mapping of each filename (of the files present at the leaf) to its hashed keyword. All the leaves will send their QRTs to their ultrapeer which will combine them all along with its own QRT(if its sharing any file) and exchanges it with other ultrapeers its connected to. Any ultrapeer receiving a query will hash it to its keywords and check if all the words belong to its query. If yes then its passed along to a leaf or a subordinate ultrapeer.

4.4 BitTorrent

A BitTorrent system comprises of three parties: Seeder, Leecher and Tracker. Seeder is the one who wishes to contribute a file to the system. It breaks the file into chunks of equal size and creates a *.torrent* file containing the meta-data of the file and publishes it on some public website. The leecher is the one who wishes to download this file and hence downloads the *.torrent* file from the website. A tracker keeps track of the all the leechers and seeders who are downloading and uploading the file respectively. It also knows the locations of several file pieces in the system. All these three together constitute a *swarm*.
Each peer who wishes to download the file sends a request to the tracker which sends it a list of 50 peers who are currently downloading and uploading the file. The peer will then try to establish a TCP connection with each leecher and seeder in that list and start downloading file pieces (downloads from the seeder and exchanges with other leechers) following *Local Rarest First* scheme - download the file piece with fewest copies in the swarm. When the neighbors of a peer fall below 30 it again requests the tracker to send it a fresh list of peers. In order to maximize each peer's download rate and counter free riding, BitTorrent follows a *Tit for Tat* strategy which will be discussed in later sections.

4.5 Distributed Hash Tables

DHT as the name suggests are systems with a distributed (decentralized) structure utilizing the lookup techniques of hash tables. DHT have three main properties:

1. Autonomous and Decentralized
2. Fault tolerant

3. Scalable

All these three properties can be satisfied if every node interacts with a limited number of nodes.

4.5.1 Structure

It comprises of a keyspace which contains n-bit strings. All nodes in a systems are identified by n-bit identifiers. These identifiers and keys belong the same space. The keys are then partitioned into sets based on some keyspace partitioning scheme and each node is assigned a set. An overlay network decides how many and which nodes, a particular node connects to. To insert a file say *(filename, data)* in the system, the filename is first hashed to its n-bit key k. Then a *put(k, data)* message is sent to any node in the DHT and is transferred hop by hop from one node to other unless it reaches the node which is responsible for the key k. The data is then stored at that node against k in its hash table. to retrieve data from the system for a file with key k a *get(k,data)* message is generated and circulated in the network unless it reaches the desired node with key k.

4.5.2 Keyspace Partitioning

DHT use some consistent hashing technique to map filenames to keys. A notion of *similarity* is defined for two keys k1 and k2 as $\delta(k1, k2)$. A node 'N' with an identifier 'i' is assigned all those keys k for which $\delta(k, i)$ is least. In Chord DHT all keys in the n-bit space are arranged on a ring and $\delta(k1, k2)$ is the clockwise distance between k1 and k2 along the ring. If i1 and i2 are the identifiers for two adjacent nodes then all the keys between i1 and i2 (clockwise) belong to i2.
Routing algorithms:

1. Hop-by-hop: Each node sends the query to its nearest neighbor. $O(n)$
2. Finger Table: Each node memorizes the location of several other nodes: a, a+2, a+4, a+8..... Send the query to that node which keys closest to the key in the query. $O(log(n))$

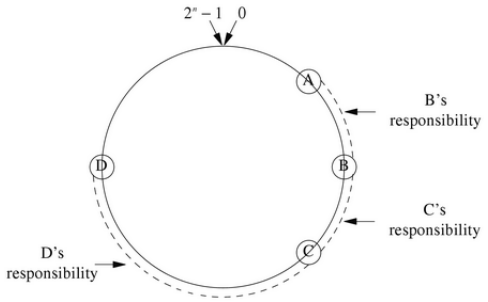

Figure 6: The Chord Ring

4.5.3 Overlay Network

Every node has a set of links to other nodes which are its neighbors. These links together form the overlay network. A node picks its neighbors depending on the network's topology. For any key k,

each node either owns k or has a link to a node whose ID is closer to k. Thus each node forwards the query message to the node whose $\delta(ID, k)$ is least following the greedy approach which is not necessarily optimal. This is known as *key-based routing*

Advantages with DHT in terms of structure is that joining of nodes is very easy. In case of Chord DHT only the keys between the joining node and its successor have to be split without disturbing the entire system. Similarly when a node leaves the system or fails, redistribution of keys can be done easily.

5 Free Riding

5.1 What is Free Riding?

Peers in a P2P network utilize resources shared by other peers. However a few peers consume resources but do not share their own. They are are known as free riders. Free riding is the situation when a peers cooperation and resource contribution towards the system fails. Eytan Adar and Bernardo A. Huberman did the first ever survey on free riding in file sharing P2P systems and found that 70 percent of peers didnt share any files at all while 25 percent provided 99 percent of all query hits in the network.

5.2 Impact of Free Riding

- A very small number of peers serve the population of a network. This results in scalability issues and single point of failure. System becomes vulnerable to attacks.
- Free riders occupy considerable network capacity and resources and cause delays and congestion in the network for other peers.
- In file sharing systems, renewal of interesting content might decrease with time.
- In P2P CPU-sharing, system utility decreases and system can collapse due to insufficient CPU resources

5.3 Methods to combat Free Riding

5.3.1 Monetary-Based approach

Money-based approaches Charge peers for any services they receive so a peer cannot endlessly free ride the system since it'll cost him everytime he uses a service. There's a Central server to keep track of a peers balance and all the transactions it makes. There can be two modes of payment *online* and *offline*. In case of online payment the peer has to immediately pay the relevant amount to the server for using a service, for this the server has to be up all the time. In case of offline payment, the peer's transaction gets recorded and he can pay later whenever the server becomes online.
Limitations:

1. Centralization and communication overhead - A single server maintains all the transactions and every peer's balance. This can cause single point of failure and scalability issues and also cause a lot of communication overhead on the network.

2. Persistent Identifiers - Each peer must have an identity since all his transactions and balance are recorded in the central database. This could be an issue with systems which allow their peers to be anonymous. Also peers can change their identity again and again to cheat the system.

3. Mental transaction costs - Users will have to think before every service whether it is worth a few bucks.

5.3.2 Reciprocity-Based approach

This approach is based on a mutual action where the service quality received by a peer is directly proportional it the amount of contribution it has made to the network. No long-term history about a peer is maintained, only current session's contribution is take into account. There are two kinds of models: *mutual reciprocity* and *indirect reciprocity*. In mutual reciprocity, a peer decides on what service quality should be provided to the other peer based only on the direct contribution made by the other peer to it. E.g.- Tit for Tat strategy. In case of indirect reciprocity, each peer monitors its neighbor's contribution to the network and provides service to it based on that.
Limitations

1. Fake services - Peers can publish fake services in order to increase their contribution level in the eyes of their neighbours.

2. Contribution-level credibility - In some systems the peer itself provides information on its contribution. A malicious peer could hack into the client program and provide misleading information.

3. Peer identity management - Peers are given identified based on their identities. A peer can get a new identity timely and receive better quality services meant for a newcomer.

5.3.3 Reputation-Based approach

Each peer's reputation is constructed on the basis of feedback given by other peers who have interacted with it in the past. Unlike reciprocity, long term behavior of a peer is recorded, thus a peer cannot become a good peer by providing services in just one session if it has free rided in the past. There can be two reputations of a peer in the system -*Local Reputation* which is maintained at every peer about every peer it has interacted with directly and *Global reputation* where local reputations is distribution among peers through 'gossip mechanism' and these are merged to create a central pool of reputations of all the peers in the system. In gossip mechanism, these reputations are piggy-backed on the P2P protocol messages
Limitations

1. Reputation reliability - Peers can mislead the system by providing wrong information on the reputation of some peer either promoting it or defaming it.

2. Centralization and communication overhead

3. Persistent identifiers

5.4 Tit for Tat strategy

It's used by the BitTorrent clients to optimize their download rates. Two main mechanisms are involved - *Regular Unchoking* and *Optimistic Unchoking*. Each peer has a limited number of upload slots say 'b', which it allocates to other peers from which it downloads. Initially all its upload slots are empty. It then selects first 'b' neighbours who has provided it with the highest download rates and allocates its upload slots to them. Consequently, when all its upload slots are filled, it will use a tit-for-tat strategy. If the peer finds that a neighbour is not uploading to it then it chokes the connection with that uncooperative peer. Selection for unchoking can be based on regular unchoking where every 10s it will unchoke top 'b' links which provides it with highest download rate. Also, an optimistic unchoking takes place every 30s where it gives a chance to a random new peer by unchoking it for some time. If the new link has a upload rate higher than any of the b-links of the peer, the peer replaces that link with it, otherwise it unchokes another new link in a round robin fashion.
Drawbacks of TFT:

1. Large-view-exploit - There's no mechanism in BitTorrent which could prevent a peer from establishing a larger view that is a free rider if choked by all its neighbors could request the tracker to provide it with a fresh list of peers over and over again. Since new peers are optimistically unchoked for a while, a free rider can take advantage of the system in this manner.

2. Whitewashing - A peer can enter the system as a new peer everytime its recognized as a free rider,

3. Sybil attack - A single peer can have multiple identities in the system. Thus if it has 'n' identities in the system then it can download from 'n*b' peers at a time!

5.5 Treat before Trick for BitTorrent

In TFET, free riders are prevented by encrypting the file using (t, n) threshold secret sharing. Any file encrypted using this technique has 'n' keys of which only 't' keys are required to decrypt it. The seeder encrypts the file, obtains 'n' subkeys and divides the file into 'm' pieces. It initially provides each leecher with one file piece and one subkey. The leecher then exchange file pieces among each following a simple rule: Any peer who wishes to download a file piece from its neighbor must upload a valid subkey (out of the 'n' generated) to the neighbor first. Free riding is prevented since even if a person downloads all the file pieces he cannot decrypt the file unless he has 't' keys for which he has to upload at least 't' file pieces. Optimal download completion time occurs when the threshold value 't[is equal to number of file pieces 'm'. In this algorithm, changing identities won't help the free riders neither would the large-view-exploit be effective.

5.6 Incentives in BitTorrent - Solution from Prisoner's Dilemma

A BitTorrent peer's dilemma in choosing whether or not to upload to its neighbor can be compared with the famous iterated prisoner's dilemma problem in game theory. The prisoner's dilemma can be explained as as follows: Consider two traders sitting at the two ends of a magic tunnel. Each has to decide whether it should send an item from his side or not. A player is in a dilemma since if he cooperates and the other defects then he will be at loss, however if he defects and the other defects

he'll receive no profit as compared to the case when he as well as the other player cooperates. The following table summarizes the profit and loss received by each player in each scenario.

P1 \ P2	Cooperation	Defect
Cooperation	R=3, R=3	S=0, T=5
Defect	T=5, S=0	P=1, P=1

The solution to the iterated prisoner's problem is to use a simple tit for tat strategy where a player will always cooperate in the first step however it will copy the moves of the opposite player from 2^{nd} step onwards. So it cooperates if player 2 cooperates and defects if player 2 defects. A player following this strategy can never lose an IPD tournament. Similar strategy can be applied in BitTorrent. Let 'd' be the amount of bytes a peer downloads from its neighbor and 'u' be the amount of bytes it uploads to the same neighbor, then the Cooperation-Defect matrix can be formed as follows: $R = d - u; T = d; S = -u; P = 0$. A player will upload to its neighbor as long as the $u - d \leq c$, for some constant c. If the difference becomes greater than c then the player will choke the neighbor. After this the neighbor shall be unchoked only if it cooperates first. The key points of this strategy are:

1. Never Defect First
2. Retaliatory
3. Forgiving
4. Clear behavior of a peer

5.7 A reputation based technique on Clustering and LP

Any peer i will calculate the local reputation of j based on to what extent its demands have been met in time period p. If $demand = D_i$ and peer j contributes x_i in return, local reputation of j at i, shall be calculated as:

$$LR_i(j) = \sum_{r_i}(x_i/D_i)$$

where r_i are all the requests sent by i to j in period p.
Next k-means clustering algorithm is applied over $LR_i(j)$ for all peers $i \in (i_1, i_2, ...i_m)$ so as to cluster the peers into two clusters based on the similarity in their reputations of peer j. The initial centroid given to the algorithm ($k = 2$) are the min and max values of $LR_i(j)$. Obviously, the cluster which is larger in size will represent the actual reputation of j in the system. The normalized distance from the centroid of this larger cluster of any peer i will give the similarity between the actual reputation of j in the system and reputation held by peer i. Let this similarity be denoted by $SIM_i(j)$. Then, global reputation GR_j of any peer j is calculated as:

$$GR_j = \sum_{i=1}^{m}(SIM_i(j)LR_i(j)GR_i)$$

This can be solved using fixed point iteration.

Allocation strategy based on reputation GR_i: Given a peer j with maximum uploading

capacity as C_j. Say it receives n requests with each request i claiming a demand D_i. Let j provide a bandwidth x_i to each request i proportional its global reputation GR_i. Following will be the LP constraints that j needs to solve to calculate x_i:

$$y = max \sum_{n} (x_i * G_i / D_i);$$
$$\sum (x_i) \leq C_j \ \forall i; \ \ x_i \leq D_i \ \forall i$$

5.8 Some Common Attacks

1. Collusion - A group of malicious peers get together to counter the free-riding prevention mechanisms. For e.g., they can promote a certain free rider and damage a good peer's reputation.

2. Whitewashing - A free rider can repeatedly change its online identity and take the advantages and rights of a newcomer.

3. Fake services/content - Malicious peers can upload files with fake filenames resembling the popular ones. Thus they upload fake data and gain reputation in the system

4. Spoofing attack - In these attacks a free rider can spoof the identity of another peer with high reputation. It generates messages which appear to generate from this good peer and hence can gain all advantages on its behalf.

5. Sybil attack - A single peer generates multiple identities and gain a significantly large influence over the network. It highly affects the reputation based systems.

6 NAT Traversal

Currently used NAT devices are designed primarily to work with the client/server model where the client machines inside a private network sitting behind a NAT initiate connections from inside to the public server whose IP addresses and DNS names are stable and known in the network. However in P2P networks, the source and the target, both might lie behind different NAT devices where neither of them has a publicly available IP. Thus NAT devices might create problems for peer-to-peer connections because hosts behind a NAT device normally have no permanently visible public IPs and ports on the Internet.

6.1 Important terminology

1. Endpoint- independent Mapping NAT (EIM-NAT) - If a private node behind this NAT initiates a connection to some public nodes outside, the NAT maps this private IP to some public IP and port say X:x. After this whenever this node tries to connect to 'any' external node, its mapped to same IP and port X:x.

2. Endpoint dependent Mapping NAT (EDM-NAT) - These behave is just the opposite manner and assigns a new IP and port to the internal node everytime it initiates a connection to some some external node.

3. Hairpinning - If two nodes, say N1 and N2 are behind the same NAT and exchanging traffic, NAT may allocate a public address to N2 say X2:x2. Now, if N1 sends traffic to X2:x2, it goes to the NAT, which relays it from N1 to N2. So basically, N1 and N2 cannot exchange data directly, it has to go through the NAT.

6.2 Various techniques used by P2P applications to traverse NATs

In all the techniques there are two peers A and B behind NAT-A and NAT-B respectively. Both NATs are EIM-NATs. Peer A has a private IP and port - X1:x1 and peer B has a private IP and port - X2:x2. A central server is present who public IP and port - S:s is visible on the internet. Peer A has initiated a connection with server S and has been allocated a public IP and port - Y1:y1 by NAT-A. Similarly peer B has also initiated a connection with the server and has been allocated a private IP and port - Y2:y2 by NAT-B

6.2.1 Relaying

If peer A wishes to connect to peer B, it cannot send it a request directly since NAT-B will drop it. However A can send its message to server S which then relays it to B. Since NAT-B identifies S, it will not drop its message. Similarly,if B wishes to send a message to A it can do so via S. This method has high latency

6.2.2 Connection Reversal

If peer A is behind NAT-A and peer B has a public IP and wishes to connect to A, it can send a connection request to S and ask it to deliver it to A on its behalf. A can then on receiving the request initiate a connection B.

6.2.3 UDP Hole Punching

- **Peers A and B are behind different NATs:** If A sends a connection request to B's public address Y2:y2, NAT-B drops it (since the request arrives from an unknown host for NAT-B who only know S). However in UDP hole punching what A can does is that it sends a connection request to B via S and then initiates a connection to B's public address. B on receiving the connection request from A via S initiates a connection to A's public address Y1:y1. Since A had already initiated a connection with B it won't drop its request. Similarly, since B has already initiated a connection to A, it won't drop A's request either and hence a P2P connection is established between the two.
- **Peers A and B are behind same NAT:** In case both the peers are behind the same NAT, there's no point communicating via NAT since it will cause unnecessary delays. Rather what A and B can do is that while sending a connection request to each other they can send their private IPs along with the public ones. So A's request to B will have two IPs - X1:x1 and Y1:y1. Similarly, B's request to A will have two IPs - X2:x2 and Y2:y2. Now A will send its message to both IPs of B and will use that IP from which it receives the response faster (will be the private IP in this scenario). B will do the same.

6.2.4 TCP Hole Punching

A and B will perform *Simultaneous TCP Open*. A and B will send SYN request simultaneously to each other's public IP Y1:y1 and Y2:y2 respectively. However these requests should be timed in a manner such that the SYN request from B arrives at NAT-A only after A's SYN request to B leaves NAT-A. So NAT-A will not drop B's SYN request since A had already initiated a connection to B and NAT-A has recognized that. Similar goes for B

6.2.5 UDP Port Number Prediction

Until now what we saw were EIM-NATs which assign the same IP to a private peer irrespective of the external peer it connects to. However what if there are EDM-NATs in the system? A guess work technique is used here. So A knows B's IP address w.r.t S which is Y2:y2. Now A will make an assumption that B's NAT allocates port numbers to each connection sequentially and hence it will send a connection request to Y2:(y2+1) and keeps on incrementing the port number unless the connection is successfully established. This is a very inaccurate method since NATs can follow a random policy for port allocation or by the time A's request reaches B, the latter might have established many other connections hence increasing the port number to a higher number in the sequence.

6.2.6 TCP Port Number Prediction

Same as UDP Port number Prediction, however SYN packets are sent here to establish connections.

7 References

[1] Detlef Schoder (University of Cologne, Germany), Kai Fischbach (University of Cologne, Germany) and Christian Schmitt (University of Cologne, Germany). "Core Concepts in Peer-to-Peer Networking".

[2] Xuemin Shen, Heather Yu, John Buford, Mursalin Akon. "Handbook of Peer-to-Peer Networking", 2010, XLVIII, 1500 p.

[3] Jorn De Boever. "Peer-to-Peer Networks as a Distribution and Publishing Model"

[4] Murat Karakaya, brahim Krpeolu, and zgr Ulusoy. 2008. Counteracting free riding in Peer-to-Peer networks. Comput. Netw. 52, 3 (February 2008), 675-694. DOI=10.1016/j.comnet.2007.11.002 http://dx.doi.org/10.1016/j.comnet.2007.11.002

[5] Kyuyong Shin; Reeves, D.S.; Injong Rhee, "Treat-before-trick : Free-riding prevention for BitTorrent-like peer-to-peer networks," Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on , vol., no., pp.1,12, 23-29 May 2009, doi: 10.1109/IPDPS.2009.5161007

[6] Michael Sirivianos and Jong Han and Park Rex and Chen Xiaowei Yang. "Free-riding in BitTorrent Networks with the Large View Exploit", In IPTPS 07, 2007.

[7] J. J. D. Mol ; J. A. Pouwelse ; M. Meulpolder ; D. H. J. Epema ; H. J. Sips; Give-to-Get: free-riding resilient video-on-demand in P2P systems. Proc. SPIE 6818, Multimedia Computing and Networking 2008, 681804 (January 28, 2008); doi:10.1117/12.774909.

[8] T. Klingberg, R. Manfredi. “Gnutella Protocol Development: Gnutella 0.6”, June 2002.

[9] J. A. Pouwelse and P. Garbacki and D.H.J. Epema and H. J. Sips. “The Bittorrent P2P File-Sharing System: Measurements and Analysis”, 4TH INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS (IPTPS), 2005.

[10] Jian Liang, Rakesh Kumar, and Keith W. Ross. 2. 006. The FastTrack overlay: a measurement study. Comput. Netw. 50, 6 (April 2006), 842-858. DOI=10.1016/j.comnet.2005.07.014 http://dx.doi.org/10.1016/j.comnet.2005.07.014

[11] Oliver Heckmann, Axel Bock, Andreas Mauthe, Ralf Steinmetz. “The eDonkey File-Sharing Network”

[12] P. Srisuresh, Kazeon Systems; B. Ford, M.I.T.; D. Kegel, kegel.com. “RFC 5128: State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)”, March 2008.

[13] G. Camarillo, Ed.For the IAB. “RFC 5694: Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability”, November 2009.

[14] J. Risson, T. Moors, University of New South Wales. “RFC 4981: Survey of Research towards Robust Peer-to-Peer Networks: Search Methods”, September 2007.

[15] Minaxi Gupta, Paul Judge, and Mostafa Ammar. 2003. A reputation system for peer-to-peer networks. In Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video (NOSSDAV ’03). ACM, New York, NY, USA, 144-152. DOI=10.1145/776322.776346 http://doi.acm.org/10.1145/776322.776346

[16] M. Meulpolder and J. A. Pouwelse and D. H. J. Epema and H. J. Sips. “BARTERCAST: A PRACTICAL APPROACH TO PREVENT LAZY FREERIDING IN P2P NETWORKS”. Parallel and Distributed Systems Group, Department of Computer Science, Delft University of Technology, the Netherlands.

[17] Lei Yang, ZhiGuang Qin, Hao Yang, Can Wang, Chao Sheng Feng. “A Reputation-Based Method for Controlling Free-ride in P2P Networks”

[18] Oliver Heckmann, Axel Bock. “The eDonkey 2000 Protocol”, KOM Technical Report 08/2002, Version 0.8, December 2002.

[19] Eric Recorla. “Introduction to Distributed Hash Tables”