NAT Traversal Issues and Solutions for Peer-To-Peer Networks

Bahareh Mousavi Jazayeri
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: Bahareh@mail.upb.de

Abstract

Peer-to-Peer communication will be blocked on Network Address Translation (NAT) if there is no mapping address for that on the NAT. So deploying Peer-to-Peer technology in a perfect way concerning NAT traversal approaches, up to now several approaches introduced, some of them are more practical and robust. In this paper I document and analyze three primary NAT traversal techniques of relaying, hole punching and STUN. Then two relevant approaches will be considered that the first one locates propriety peers in the overlay to establish direct TCP connection between two endpoints, and the other one implements superpeers to utilize STUN and manage NAT traversal. At the end I discuss FreePastry built-in configuration for traversing NAT that enables UPnP otherwise it suggests to implement one of three mechanisms by the overlay.

1 Introduction

Nowadays networking technology is moving to a new direction in which there will be less need to implement and maintain servers with considerable resources. Although client-server fashion networks are still popular and dominating, Peer-to-Peer technologies have been emerging fast in the recent decade. Statistics [1] show that the portion of clients resources is 20 times bigger than all server resources around the world, and only the Peer-to-Peer technology can make these valuable but distributed resources usable as an autonomous system. There are many big projects like Skype, Bittorrent, Fasttrack which attracted millions of users, and motivated them to participate in the Peer-to-Peer networking.

There are some issues which still make obstacles to implement a Peer-to-Peer technology, one of the major difficulties is that the Internet architecture is designed to satisfy client-server interaction requirements. One of the highly problematic issues in this architecture is NAT [14], as the main goal of NAT is to save the global IP address consumption and the conservation of the private network, it avoids Peer-to-Peer connections to be established by default.

Up to now some techniques have been put into use, and deployed in the commercial and non-commercial Peer-to-Peer products. For instance in IP telephony the proprietary protocols like SIP [13] are utilized, in Skype [4] messages are routed through the non-NATed nodes and superpeers, locating specific nodes on the Internet to relay messages in online gaming, and other techniques like spoofing, port prediction and port forwarding are the examples of implemented approaches.

In this paper, the most considerable NAT traversal methods are explained and then the most promising approaches are discussed. In Section 2, I have a look at the NAT functionality and consequently the NAT traversal issue, then in Section 3 relaying [7], hole punching [7], STUN [16] and STUNT [8] will be considered in details, as they are the base mechanisms for the most recent approaches like NatTrav [6] and STUN-superpeers [15] which will be explained in Section 4. After that I consider FreePastry [2] NAT/Firewall configuration that is the main subject related to the Project Group seminar in Section 5. Section 6 discusses the other related works, and I conclude in Section 7.

2 NAT and NAT Traversal

NAT stands for Network Address Translation, it is a middle box which its main goal is to serve the internal network with the private IP addresses and pass private clients connections out to the public network. The private address can be used many times in different internal networks that cause to prevent a lot of global IP address consumption. When the client is going to establish a connection to the outside, the application uses the private IP address and private port number for starting the connection, then sends the the packets to the Internet gateway, and the gateways are usually equipped with NAT, here NAT receives the connection from client, then by binding a public IP address and public port number enables the packets go through the Internet, then NAT saves the done mapping on a table. In contrast, inbound connection is allowed to pass to the private network if there exists

a related mapping to the initiator of inbound connection on the NAT table. So in addition to saving public IP address, NAT have limited some of the security concerns and attacks.

Although it brings the world of networking a lot of benefits, it introduces a problem named NAT traversal issue. That means internal network can send outbound packets to the public network, but inbound connection from outside networks will be accepted only if they are wanted from inside network, otherwise NAT will drop these connections.

Still some technologies like IP telephony, gaming, file sharing and Peer-to-Peer application have considerable obstacles in traversing NAT/Firewall. So when a client in the private network needs to be contacted before any connection from inside network initialized, then NAT traversal approaches come in.

Different approaches are presented to traverse NAT which most of them are based on different behaviors that NAT may treat. These are some features that NAT producer companies may include in their products. These differences are more based on two parameters; first how NAT performs address mapping, and second who can connect to the internal client from outside by knowing its public IP address and public port. [16] generally presents four types of NAT that are as following:

- Full Cone NAT (FC)
- Restricted Cone NAT (RC)
- Port Restricted Cone NAT (PRC)
- Symmetric NAT (SYM)

Cone-mapping NAT types bind the same mapping public endpoint to the same client, no matter the client in the internal network is connecting to which destination client. Contrary a Symmetric NAT changes the public endpoint address that is mapped to the same internal client when connecting to different destinations in the outside, clearly it is endpoint-dependent.

Furthermore any client can traverse FC by contacting the public address mapped to the private client by its NAT, but if it is RC, the IP address belongs to the client in the outside network will be checked, whether it has been contacted before or not. PRC restricts both the IP address and the port number but still it is not as restrict as SYM NAT which allows a unique endpoint to pass on that are mapped to the internal client in the past.

Most of the NAT traversal approaches are designed and implemented for cone-mapping NAT types, also there are approaches which consider symmetric NAT traversal. However the academic communities are optimistic that NAT market is going to decrease symmetric NAT production in the future, to cope with IP telephony and other technologies more than before.

3 NAT Traversal Solutions

In this section the most well-known methods to the NAT traversal are presented which are the base of the most recent NAT traversal approaches.

3.1 Relaying

In this approach, the clients behind NAT are already connected to a well-known server or generally a well-known peer. As clients are behind NATs they are unable to have direct connection, so this server passes on all communications between these two end points. Although it is attempted to emulate Peer-to-Peer connection, it still works in client-server fashion.

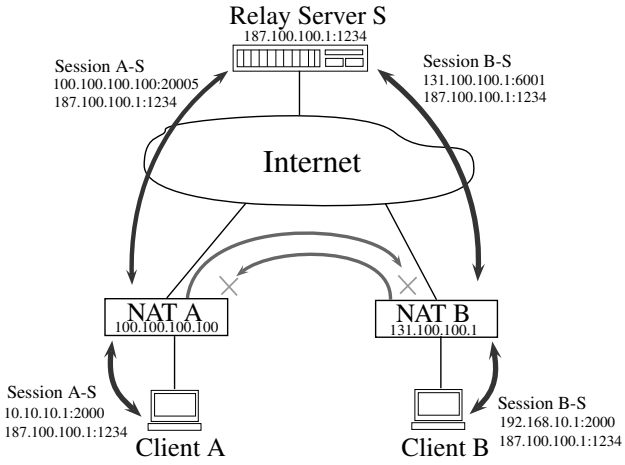


Figure 1. NAT traversal is done between A and B through the relaying server [7]

As shown in Figure 1, client A is willing to connect to client B but it cannot traverse NAT B, then it sends its connection request message to the server S, who has already established the client-server connection with A and B. So here server S is responsible to relay this message from A to the real receiver means client B. When B is willing to answer to A, it follows the same relaying procedure. A and B stay connected to each other until they have connection with server S.

As all communications from A and B are done through the relay server, this server should have valuable resources such as high network bandwidth and high processing power to carry the network traffic and avoid problems like the single point of failure and the bottleneck. Although this approach has some advantages, it works with any kind of NAT even symmetric one, moreover there is no limitation to have TCP connection or UDP session. So it can be the final reliable NAT traversal method in case of the failure of other approaches.

Turn protocol [12] is the relaying protocol, it helps both

clients and server to implement a client-server base connection in the sufficient way. It does not allow servers to assign famous port to relaying purpose, this keeps security, and specially it has benefits for symmetric NAT as in this way symmetric NAT all the time maps outbound and inbound related to a specific client to a well-known peer in the outside network (relaying server), so its role will change to the port-restricted NAT .

First TURN client generates TURN request to the TURN server, TURN protocol makes the clients only in role of receiver of this connection. Here TURN server is the middle path relaying peer which is receiving TURN requests from the clients, then sends the TURN responses back, its main role is to receive data and forward them to the other client.

3.2 Hole punching

This NAT traversal method is used if both side of Peer-to-Peer endpoints are behind cone-mapping NAT types, because this approach relays on using the public IP address and public port number given to the client, so when NAT (symmetric) changes port assigned to the client for the next time mapping, it would not be practical to punch a hole in symmetric NAT by acquired IP address and port. Although there are some countermeasures like port prediction but they are costly and non-standard in some cases.

Rendezvous server is an important concept which must be explained before discussing hole punching mechanism.

Rendezvous server- Before implementing real Peer-to-Peer connection between two endpoints which are behind NAT, addressing must be relayed between two clients by a third party, named rendezvous server. It is not necessarily a server, it is a middle point between the clients with which both endpoints have alive connection. It is actually a server or an ordinary peer who can connect both endpoints and have active UDP or TCP connection with them.

Each endpoint registers in the rendezvous server by providing the local IP address and port number, this address named private endpoint, and client claims that it can be contacted by using this address. Rendezvous server extracts the private address from body of registration request which is sent by client. The public endpoint contains the public IP address and port number which is bound to the client by NAT. This address will be found by rendezvous server from header of IP packet and its port from header of UDP datagram. By these two addresses- private and public IP address and port- rendezvous server can register each client and then keeps its active UDP session by receiving periodic registration request.

By these explanations, it is supposed that both end client who are willing to have direct contact, are already registered in rendezvous server. So I can go forward to the hole punching mechanism.

UDP hole punching- To establish UDP session from the client A to the client B, first A as the initiator of connection should contact the rendezvous server, this is shown in Figure 2, while the first attempt from A to B will be fail in NAT which B is behind. It is supposed that client A and B have already active sessions with the rendezvous server. So A asks the rendezvous server to forward the contact information of A, include public and private endpoint, to B, then asks for the contact information of B. Then in second step the rendezvous server sends public IP address and port number, by that B can be contacted, to A. It forwards connection request message of A to B. In the third step A and B try to contact each other, as A makes a hole in its NAT by the previous connection attempt to B, B will be successful in traversing NAT A.

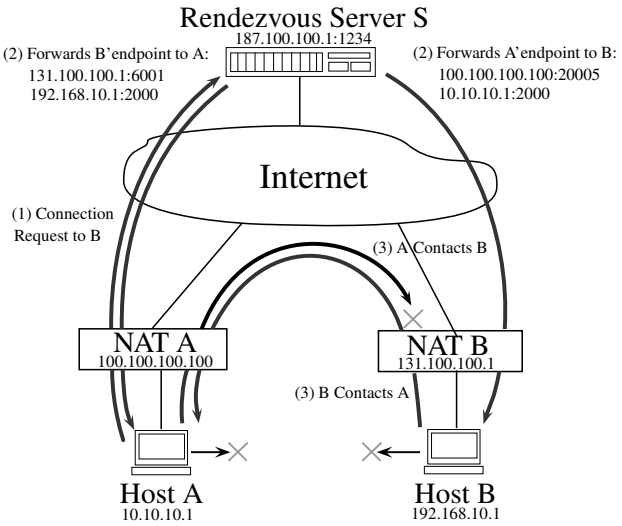


Figure 2. UDP Hole Punching [7]

TCP hole punching- If end hosts are going to establish a direct TCP connection, NAT traversal can be applied by hole punching. In this case establishing TCP connection will face some challenges, but still the hole punching is the main approach to have TCP connection between two peers behind NAT. The point is that the TCP is connection-oriented, it provides unique TCP socket from the initiator A to the receiver B. When A as the initiator sends SYN packets it connects on a unique local TCP port to the receiver, automatically this TCP port is different to the local port on which A is listening when it is waiting to receive the respond from B. Technically if A starts a TCP connection to B a 4-tuple socket will be assigned to this connection request, another 4-tuple socket will be assigned to TCP connection from B to A. So by default, B is not going to use the same port number when answering to the connection request of A, this makes the hole punching useless when B does not benefit from hole made by TCP connection request of A in NAT A.

The countermeasure to the problem is an application

layer setting on the endpoints. Today most of the operating systems support this option that named $SO - REUSEADDR$, as [7] proposes this enables each client to bind multiple TCP sockets to the same local TCP port that forces endpoints to use the same socket stream in the connection from A to B and from B to A. Figure 3 demonstrates the idea in an abstract view. It can be seen that listen socket which is used to receive the expected connection, connection to S and connection to the public and private endpoint on the other side all are bound to the one local TCP port on each host.

So TCP hole punching will be applied while host initiate connection to each other on the same TCP port when they are connected to rendezvous server S.

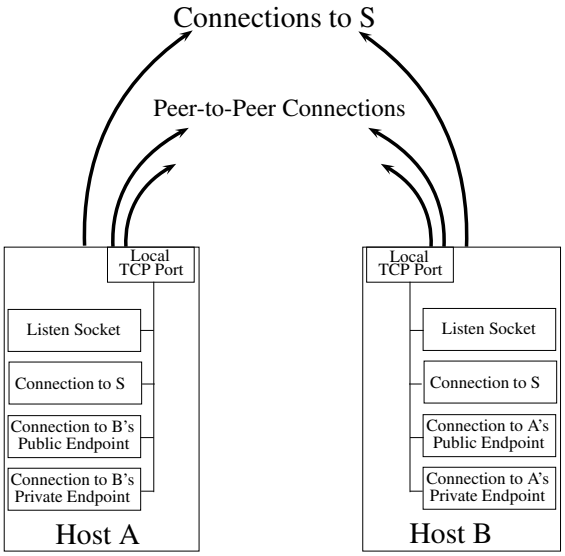


Figure 3. All sockets are bound to the one TCP local port on each client [7]

TCP simultaneous connection- As some TCP connections must be established at the same time from the both clients, timing should be managed by the third party, again the rendezvous server. Here client A and B send SYN message to the other client at the same time, or at least before SYN from A get to NAT B and SYN from B get to the NAT A, that means hole in the both NAT A and NAT B is made before the connection request from the other side arrives. This simultaneous connection establishment has some benefits; first, it is fast specifically TCP connection will be established at the first attempt, so it does not need to do sequential steps as I will see in Section 4.1. Second the probability of failure is less while in some cases TCP connection will be timeout or dropped by the other side.

3.3 STUN and STUNT

STUN [16] is the UDP base protocol to traverse cone-mapping NAT types. STUNT [16] has been introduced by [10] to improve the functionality of STUN protocol and to omit its considerable limitation as it is for TCP connections too. Still both of them are not applicable with symmetric NATs.

STUN, Simple Traversal of UDP Through NAT, is introduced as a standard protocol in RFC 3489 which helps client behind NAT, first to discover if it is NATed, and if so, STUN determines the NAT type, and second to find out the public IP address and port which are assigned to the client by NAT. In addition STUN is a standard protocol which needs to apply no change in NAT functionality.

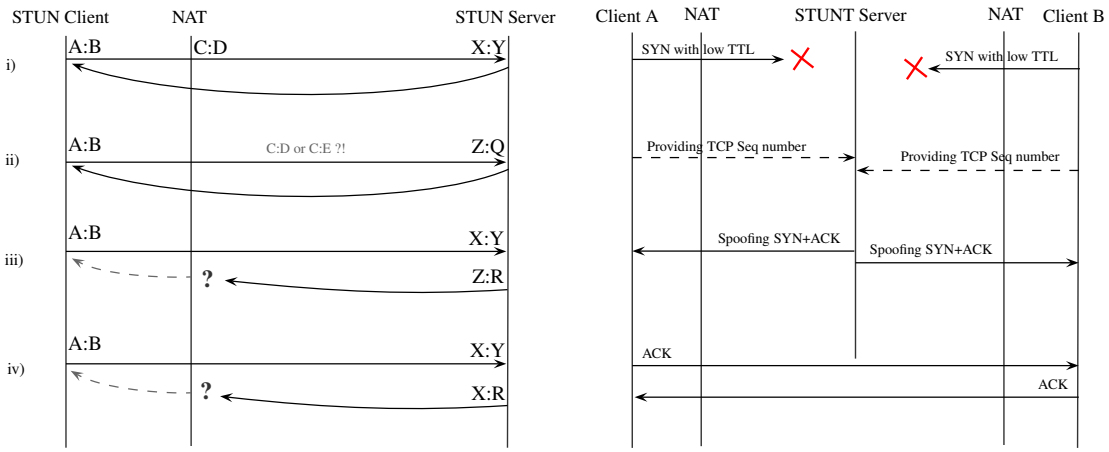
STUN defines two roles; the first is STUN client, it is an ordinary client who sends STUN request and interested in finding the type of its NAT and the bound public IP and port during mapping process. The second role is STUN server which receives STUN requests and answers them. STUN servers are nodes which are connected to the the public Internet and can be used for free.

See Figure 4(a). At the first phase (i) STUN client sends a STUN binding request to STUN server, there is the private endpoint means IP address:port number (A:B) in STUN Binding request. Later STUN server receives this request, puts the source address of STUN Binding request into the STUN Binding response and sends the response back to the client, in this step STUN server answers with original IP address and port number in the source header of STUN Binding request (C:D) which bound to the client by NAT (if there exists). The STUN client receives this response, by comparing the IP address returned by STUN server (C:D) with its local IP address , client will find if it is NATed or not. In this step the client could find out its public endpoint, if NAT is full cone, after this step only by listening on this port the client can receive the inbound connections traversed NAT from the out side.

As still the client does not know the type on its NAT, at the second phase (ii) client sends a request to the STUN server from A:B but this time to a different IP address (Z:Q), again this time STUN server send back the STUN Binding response, if client receives a different IP address from the respond in the first phase inside the packet, that means the NAT is symmetric which assigns a unique mapping to each connection from the client to different destinations.

To find out if the NAT is full cone or not, at (iii) the client sends the STUN Binding request to the STUN server (X:Y) from A:B, and asks it to respond with a different IP address (Z:R), if the client receives this respond, it will result that it is behind a full cone NAT type.

Moreover the NAT may be restricted or port restricted one, so at (iv) the client (A:B) sends the STUN Binding request to the server (X:Y) and asks it to answer from the



(a) Functionality of STUN protocol

(b) Functionality of STUNT protocol [9]

Figure 4. STUN is the UDP based protocol, and STUNT improves STUN functionality to TCP connections too

same IP but a different port number (X:Q), here if the packet can be delivered to the client, the client will know that its NAT is just restricted one, otherwise it is port restricted NAT type.

During this process by STUN protocol, the client first discovers if it is settled behind a NAT and its public endpoint, and second it finds the NAT type.

STUNT stands for Simple Traversal of UDP Through NAT and TCP too, presented by the team who introduced NUTSS [10]. It is aimed to establish the direct TCP connection by STUNT. As it is shown in Figure 4(b), each client sends SYN messages with low TTL to the other one, then it sends the information related to this TCP connection to the STUNT server which both clients have already access, based on the provided sequence number by client, STUNT server responds to the clients by spoofed SYN+ACK, consequently the clients sends ACK as usual and the direct TCP connection will be established.

4 Recent Approaches

In this section two relevant approach have been considered, both of them do a portion of NAT traversal task in the Peer-to-Peer overlay. the first approach, NatTrav is an application approach which applies TCP hole punching in sequential steps, and it is useful for applications that are not able to bind the different TCP sockets to only one local port. The second approach has mainly combined STUN protocol with the other approaches means *connection reversal*, hole punching and relaying.

4.1 NatTrav: a software approach to traverse cone-mapping NAT types

NatTrav [6] as a software approach presented in 2002. It is a java-base application to be installed on peers. This approach explains a method to establish the TCP connection between ordinary peers behind cone-mapping Nat types by implementing sequential procedure. It is claimed that the connection will be established in the Peer-to-Peer fashion without manipulation of any protocol, security and NAT functionality. In this section this approach is explained more in details.

Here, three roles are supposed; first is the initiator peer, the one who is willing to connect to another peer directly, the second role named recipient which receives the connection from the network and registers in the peers in the third role named connection brokers. They are intermediate nodes who have two main tasks to do; first, connection brokers supply the contact information of initiator for the recipient, next they help initiator and recipient to connect each other directly. If the recipient is not NATed, connection reversal [11] will be applied as NAT traversal method. But if the recipient is behind NAT hole punching will be implemented. **Registration-** all recipients must be registered at connection broker if they are interested in being contacted by connection broker later, specifically each recipient provides its URI -Universal Resource Indicator- to the connection broker. The recipient sends its contact info by UDP connection, this UDP connection keeps the recipient alive for the connection broker, so the connection broker can forward connection request to the recipient.

Although the registration process can be done by TCP connection, the problem is that for every TCP connection

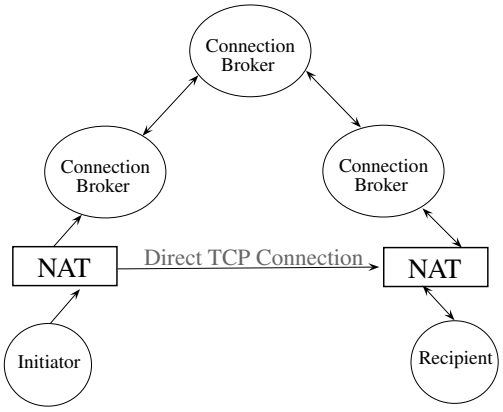


Figure 5. NatTrav Architecture [6]

from the recipient to the connection broker, the connection broker has to establish one TCP session that would be too many of them after awhile, so here UDP session is preferred. The peer which connects to the network only in TCP would use another peer with UDP connection capability as a proxy to the connection broker. To stay connected with the connection broker, registered recipients should periodically send registration message to it. In registration step to avoid attacker to register in connection broker and lookup itself a specific security protocol can be used to authenticate connection brokers and recipients. Figure 5 shows that the recipients have the active session with the connection brokers, and the initiator sends the lookup to the connection brocker and connection brokers are connected and replicated with each other.

Lookup- The initiator peer who is willing to connect directly to another peer provides the desired URI to the connection broker. If connection broker has no connection to the recipient then forwards the connection request to other connection brokers, finally the connection broker sends the public IP address and public port of recipient back to the initiator.

Recipient punches a hole in its NAT- If the recipient is behind NAT, hole punching must be facilitate by connection broker between the initiator and the recipient. If suppose, the recipient is already registered in the connection broker, Figure 6 shows how the initiator starts a lookup for the desired URI, and finally establishes a direct TCP connection to the recipient. R/I and NR/NI respectively mean private and public IP address of recipient/initiator.
In step 1, the initiator sends a lookup request to the connection broker, then in step 2, the connection broker forwards the connection request to the recipient. In step 3 the recipient sends the connection request ACK to the connection broker, here by receiving this ACK connection broker decides whether hole punching is needed to be implemented, connection broker performs that by comparing the private IP address to the public IP address, if they are identical the recipient is not NATed, otherwise hole punching must be done in the recipient NAT.

In case of hole punching, the connection broker provides the recipient peer with the public IP address and public port number of the initiator and asks it to close the current TCP connection with connection broker in step 4, here there is a trick; when the connection broker gets the information of recipient TCP connection asks it to close this connection then restarts it with the initiator in the next step. So during this process TCP socket is identical, later in step 5 the recipient peer tries to establish TCP connection with the initiator, but as the initiator is behind NAT NI, the request will be failed however the main goal was to create a hole in NAT NR. After that the recipient starts a new TCP connection with connection broker to tell him that he is listening on port j to receive the TCP connection from initiator (step 6). The time is to benefit from the hole in NAT NR, so now the connection broker answers the lookup request asked by the initiator at the beginning in step 1, it forwards the information of TCP connection with the recipient and asks the initiator to establish the TCP connection with the desired recipient (step 7). At last in step 8, the initiator sends SYN message to the recipient, by traversing NAT NR, there is no problem in having the direct connection between the initiator and the recipient.

This approach helps peers behind the cone-mapping NAT types to connect directly. Specially for the TCP connection, the problem is more severe which is claimed to be solved by this approach. But TCP traversal of symmetric NATs stays unsolved however authors in [6] suggest port prediction extension in order to symmetric NAT traversal.

4.2 NAT Traversal by STUN-Superpeers

This approach [15] combines other NAT traversal approaches to implement them as the extra functionality in Peer-to-Peer overlay. Here superpeers must be defined, so heterogeneity must be considered in the overlay. The superpeers should be behind FC NAT or no NAT at all. They register in DHT, so their address can be retrieve by a lookup in DHT. [15] expresses NAT traversal in two phases which are explained more in details in the following.

In the first phase each peer must find the type of NAT which is behind, this will be done by STUN protocol, as STUN servers are freely available in the public Internet, it is possible to connect them however this needs the administrative privilege on the end point. So internal STUN servers are maintained in Peer-to-Peer overlay which are mentioned superpeers. Every two superpeers can act as a STUN server, as the STUN server should have two public IP address to perform STUN tests, after performing a lookup to find superpeers, a peer sends STUN request to the superpeers then they will determine its NAT type.

By finding NAT type in the both endpoints of connection,

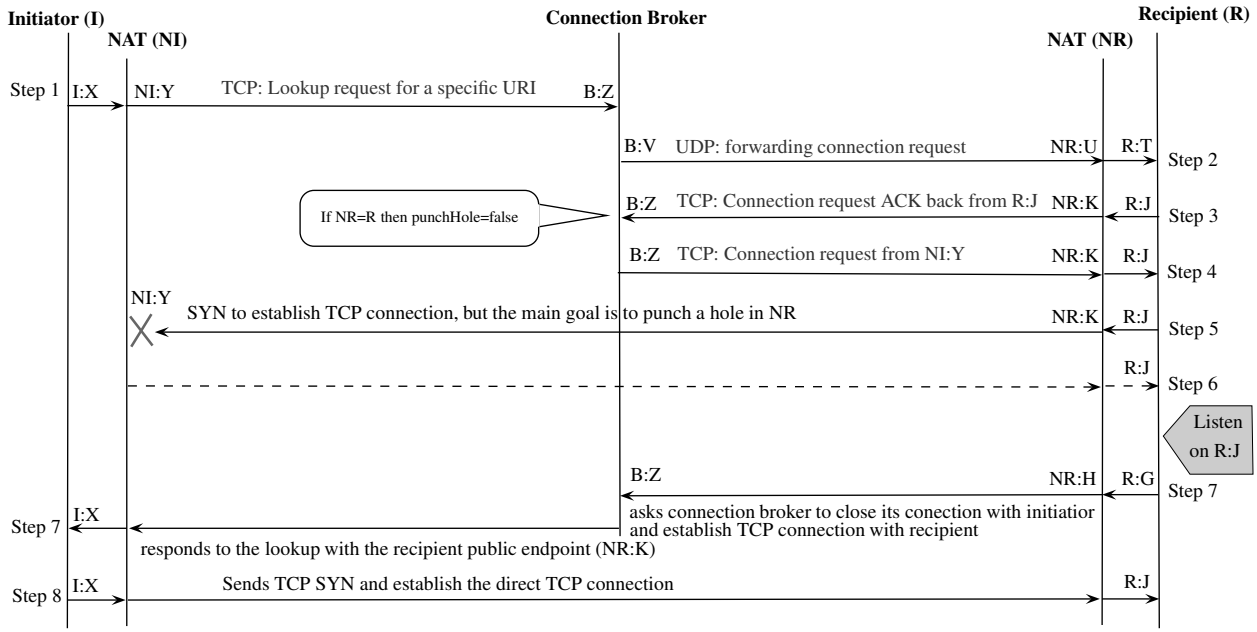


Figure 6. Establishing a direct TCP connection between two clients by Connection Broker [6]

the initiator and the recipient of connection, NAT traversal method can be determine. Table 1 is presented as a guidance to choose the appropriate NAT traversal method that can be managed at runtime.

In this approach many main tasks like making simultaneous connection, routing and hole punching are supposed to be managed in the overlay. For instance if relaying is going to be as NAT traversal solution, superpeers will relay messages between two peers in the FreePastry overlay.

NAT	FC	RC	PRC	SYM
FC	Direct	Reversal	Reversal	Reversal
RC	Direct	Hole Punching	Hole Punching	Hole Punching
PRC	Direct	Hole Punching	Hole Punching	Relaying
SYM	Direct	Hole Punching	Relaying	Relaying

Table 1. selection of NAT traversal method depends on NAT types in the both endpoints [15]

5 FreePastry NAT/Firewall Configuration

Only a small number of clients in the global network have the routable IP address. So the majority of nodes are behind NATs and Firewalls, which makes nearly impossible to connect clients behind them without any extra configuration. As FreePastry is a Peer-to-Peer overlay, it must overcome NAT and Firewall traversal obstacles. In this section, FreePastry NAT/Firewall configuration is discussed specifically. First the existing implemented approach will be explained, according to that the FreePastry overlay has already been configured to work. While the existing approach does not satisfy NAT traversal of all NAT types, in Section 5.2 final approaches will be extended by combining the recent approaches discussed in this paper with suggested approaches by FreePastry community.

5.1 Existing Solution

FreePastry project [2] considers the possibility of the direct communication between two hosts in different situations. It has been applied built-in the configuration in the overlay. If a node which is behind the NAT/Firewall succeeds to join the FreePastry ring, it means its contact information will be available in the overlay, so it gives the chance to be accessible by node ID that the overlay has assigned to the peer after joining the overlay, in the FreePastry, for instance, each node has a 160-bit identifier which is registered in the DHT. So one of the most important step towards NAT traversal would be for a new node when it is going to join the overlay. In this step the overlay gathers the information of NAT/Firewall configuration related to the new node, and exploits these information to overcome the NAT traversal challenges of the proprietary node in the future.

FreePastry has supposed that a new node has one of these states before joining the overlay:

- It has a global IP address which is reachable without trying any traversal solution
- It is behind a cooperative NAT/Firewall, here cooperative NAT/Firewall means a device which has already enabled port forwarding capability

- It is settled behind a non-cooperative NAT/Firewall

The best one is a node with the Internet routable IP address, by default and with no additional NAT traversal configuration this node will join FreePastry ring, afterwards it will be accessible in the overlay. In addition FreePastry uses this type of node as the bootstrap node, the rendezvous point and the relaying node to help a new node to join the ring and to traverse NAT/Firewall. So when a new node is willing to join FreePastry overlay, in the first step it will contact with the bootstrap nodes in *probeAddresses* that is a list of bootstrap nodes offered by FreePastry.

The bootstrap node will obtain NAT/Firewall information of the new node. In the second step, port forwarding must be enabled on NAT/Firewall, hence FreePastry checks if UPnP [3] is enabled on the proprietary NAT/Firewall, if so, it reuses the port forwarding information otherwise FreePastry attempts to enable the port forwarding by *SBBI*, which is the one of UPnP libraries. As some of the NAT/Firewalls does not support UPnP protocols, this attempt may be failed. In this case, FreePastry asks user to enable port forwarding manually. Therefore user must find if the addressing schema done by Internet Service Provider (ISP) is either dynamic or static, if it is dynamic the same port must be set up for UDP and TCP.

After the attempts to enable port forwarding, the next is to test the configuration by a *Firewall Test*. The Firewall Test determines whether The NAT/Firewall is cooperative or uncooperative. Uncooperative means NAT/Firewall does not have port forwarding. Here FreePastry provides another node in addition to the bootstrap node, then asks the new node which was willing to join the overlay to send a packet to the bootstrap node, and finally receive it from the third node in the overlay. If Firewall Test fails, FreePastry results that NAT/Firewall is the uncooperative one, then marks the node which was willing to join the overlay as Firewalled. The Firewall Test ensures the two way connectivity of the new node to the ring [2]. Figure 7 demonstrates the discussed procedure, for simplicity the FreePastry overlay is shown in the form of a simple ring.

[2] proposes that in order to establish a direct connection between two nodes in the FreePastry overlay, information related to NAT/Firewall configuration must be provided. If at least one of two nodes have routable IP address or is behind a cooperative NAT/Firewall, then the messages should be routed to the firewalled node, After that the firewalled node opens the TCP session or UDP connection, the proposed mechanism is the same as connection reversal method.

Although the existing built-in configuration traverses NAT/Firewall which has the capability of port forwarding, but what if the client is behind uncooperative NAT/Firewall? Surely it can join the overlay, but it has a one way connection to the overlay, then it can contact the other nodes, but it may not be accessible from the other nodes in the overlay, here the chance of two way connectivity depends on the type of NAT. So the obligation of enabling port forwarding is still a considerable limitation and only port forwarding is not an absolute solution. The next section provides prospective solutions to traverse different types of NAT which are more feasible than other approaches to employ in the FreePastry overlay.

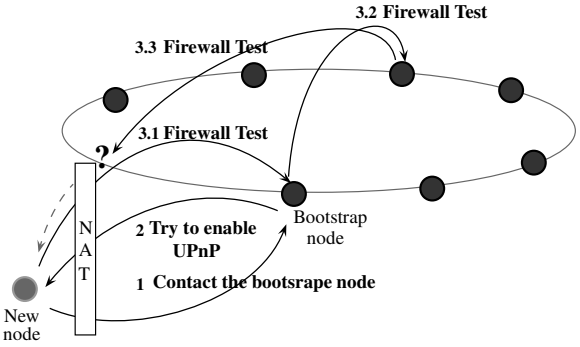


Figure 7. Information of the NAT/Firewall configuration must be detected before joining the overlay

5.2 Prospective Candidate Solution

Suppose a node is wishing to establish the direct connection to another one, If FreePastry determines that the both nodes are firewalled, that means they are behind the uncooperative NAT/Firewalls. So one of the mentioned NAT traversal methods, which are relaying,hole punching and STUN (STUNT), must be applied. To do so the first requirement is to find the third node in the overlay which is capable to act as the relaying or the rendezvous node, this capability is based on owning a public IP address or settling behind a cooperative NAT/Firewall, I call such a this node superpeer. In addition at least one of the such a this node must exist in LeafSet of each endpoint, to handle relaying traffic between nodes. [2] suggests that when a firewalled node is going to establish the direct connection to another firewalled node, hole punching and relaying can be the solutions for the NAT traversal , and it is going to be managed by STUN (STUNT) protocol.

Furthermore superpeers must be detectable in the overlay, means their node ID can be retrieve by the lookup. The next point is that there must be nodes which support STUN (STUNT) protocol. The nodes that satisfy such these requirements can be contacted by endpoints, hence as explained in the Section 4.2 they will determine NAT types and send the bound public address of each endpoint back to it. Then NAT traversal method will be selected and applied

through the overlay by employing superpeers.

Moreover to establish the direct TCP connection still the superpeers can perform *connection broker* responsibilities discussed in the Section 4.1. Although the direct TCP connection could be established by STUNT, [9] discusses the remained challenges to this procedure like the difficulty of setting SYN message with low TTL and the challenges concerning spoofing in the process done by STUNT. So the discussed approach in the Section 4.1 reveals its importance where it does not manipulate any protocol and applies standard functionality which needs no administrative privilege on the clients. In addition it is feasible for the Windows operating systems prior to Windows XP Service Pack 2 which are not enable to use the $SO - REUSEADDR$ functionality mentioned in the Section 3.2 [7].

6 Related Work

UPnP [3] includes networking protocols to ease the end client to join other devices on the network, it has been introduced in 1999. More than 972 companies have supported the protocols. It automatically enables end client to find its name, IP address and port number on the network. By IGD protocol that is inside UPnP, port forwarding on the NAT/Firewall would be enabled, in this case the NAT/Firewall should support UPnP. Unfortunately UPnP supposes a fully trustable system, and no authentication mechanism is involved, then the security concerns make some companies hesitated in supporting UPnP on their products. So as [6] criticizes UPnP, it may not be an absolute solution when either the end user does not have administrative access to enable UPnP or NAT does not support UPnP.

SIP [13] stands for Session Initiation Protocol, invented to manage multimedia communication between users. By SIP protocols users start, keep available and terminate sessions with each other. In its distributed architecture, server named proxy routes the messages to the final destination. One of its most important usage is in telephony.

NUTSS [10] is the different approach base on STUNT to establish a direct TCP connection, it gathers different techniques such as STUNT, hole punching, port prediction and SYN spoofing to make a unique functionality.

Here end clients have connection with the rendezvous server, and let it know about their public and private endpoints, later they use these information to predict the port number which may be assigned to the other client by its NAT, and they send low TTL SYN message to the other client, then contact STUNT server to determine type of NATs and spoof SYN+ACK messages to the NATs from the outside, here STUNT server manages the connection to seems a two way connection but the sequence number gained by the end client itself. After that endpoints will send the final ACK to establish the direct TCP connection.

Although the mechanism is planed to work with even the hardest NAT type means symmetric one, there are some risky mechanisms in this approach that would fail in the real world, first SYN+ACK messages must be spoofed by STUNT server, today many ISPs filter such a this spoofed packets. Second SYN message must be send in low TTL as much as getting to its NAT but not to the NAT in the other side, it is practically impossible when two client are using a common interface on NAT. In addition this process needs administrative privilege from the end user [5] [9].

NATBLASTER proposed by Biggadike [5] is an approach designed for traversing the most restrictive NATs and establishing direct TCP connection. It applies port prediction for the symmetric NATs by multi hole punching that means peer behind the less restrictive NAT sends some SYN messages with different port numbers to establish the connection with peer behind the restrictive NAT, then it is managed that second peer starts to answer to this connection, although the port assigned to that by restrictive NAT is not an absolute port number, it probably matches to one of the SYN message sent by the first peer. Here a rendezvous server again coordinates sequence numbers of TCP segments between two clients. But still [11] expresses that port prediction has its own challenges, and IP loose source must be provided in this approach which due to security concerns such these packets are sometimes blocked.

7 Conclusion

To overcome NAT traversal issue in the Peer-to-Peer communications three mechanisms are obtained as the base of traversal functionality, that are relaying, hole punching and STUN (STUNT). Relaying works for all NAT types while it is an inefficient approach practically. If at least one of two clients is behind a full cone NAT type or no NAT at all, traversal can be done by connection reversal, this case is the easiest one. But if they are NATed by cone-mapping NAT types the hole punching come in mind, [6] proposes a mechanism in which peers register in specific nodes named connection broker before applying the hole punching in a sequential procedure. Although [15] expresses more overlay dependent approach in which there is no need to register beforehand, superpeer addresses can be retrieve by lookup in the overlay as they store themselves in DHT. Then superpeers manage NAT traversal by implementing STUN.

On the other hand, Freepastry as an Peer-to-Peer overlay configures UPnP to enable port forwarding on NAT which is not an absolute solution, because of security concerns still many products do not support UPnP protocols. So [2] first counts on enabling port forwarding however for the next step hole punching and relaying are suggested as NAT traversal in more complex situation which must be managed all in the Peer-to-Peer overlay.

References

[1] URL: www.statista.com, Last accessed in May 2012.

[2] Rice University, FreePastry. URL: www.freepastry.org/, Last accessed in May 2012.

[3] UPnP Forum. http://www.upnp.org/, Last accessed in May 2012.

[4] S. A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *Computing Research Repository*, abs/cs/041, 2004.

[5] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig. NAT-BLASTER: Establishing TCP Connections Between Hosts Behind NATs. In *Proceedings of ACM SIGCOMM ASIA Workshop*, 2005.

[6] J. L. Eppinger. TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem. *Technical Report CMUISRI-05-104*, 16, 2005.

[7] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-Peer Communication Across Network Address Translators. In *Proceedings of USENIX Technical Conference*. ACM, 2005.

[8] S. Guha and P. Francis. Simple traversal of UDP through NATs and TCP too (STUNT). URL: http://nutss.gforge.cis.cornell.edu/stunt.php, Last accessed in May 2012.

[9] S. Guha and P. Francis. Characterization and Measurement of TCP Traversal through NATs and Firewalls. In *Proceedings of the International Conference on Internet Measurement*. ACM, 2005.

[10] S. Guha, Y. Takeda, and P. Francis. NUTSS: a SIP-based approach to UDP and TCP network connectivity. In *Proceedings of ACM SIGCOMM Conference*. ACM, 2004.

[11] H. Peters. Analysis of NAT approaches and explicit signaling for NAT traversal, 2006.

[12] J. Rosenberg, R. Mahy, and C. Huitema. Internet draft: TURN-Traversal Using Relay NAT. 2006.

[13] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. *IETF RFC 3261*, 2002.

[14] P. Srisuresh. IP Network Address Translator (NAT) Terminology and Considerations. *IETF RFC 2663*, 1999.

[15] A. Wacker, G. Schiele, S. Holzapfel, and T. Weis. A NAT Traversal Mechanism for Peer-To-Peer Networks. In *Proceedings of International Conference on Peer-to-Peer Computing*. IEEE, 2008.

[16] J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). *IETF RFC 3489*, 2003.

Streaming and live streaming

Jens Janiuk
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Distributed Framework for Social Networks
Email: jjaniuk@mail.uni-paderborn.de

Abstract

Nowadays a typical multicast live streaming solution is using a central server to broadcast the live media from one towards different users. This approach has a lot of disadvantages like the high load on the server, the dependency on the server, the single attack point for potential hacker attacks and many others. This paper presents some general and promising, existing approaches to make multicast (live) streaming in Peer-to-Peer networks possible.

1 Introduction

Centralized live-streaming to large audiences cause a huge amount of server load and bandwidth. Since streaming and live-streaming over the internet becomes more popular and the standards of the streaming quality continually increased, Peer-to-Peer live-streaming has become a popular research topic.
It offers a huge number of advantages on both sides. By using the capacity of the consumers itself the publisher of the live content do not have to offer huge upload capacity to serve all consumers. The system becomes much more stable and attackers don't have single attack points. Also it becomes very easy for private users to offer live-streams towards a huge audience.
To develop an useful Peer-to-Peer live-streaming network it has to fulfill a lot of requirements besides the basic requirements of a general Peer-to-Peer network like a high robustness against frequently joining, leaving, or even failing nodes, a fast routing mechanism, etc. Because of this, the general idea is to build an overlay on top of an existing network structure to support P2P live streaming.
But let's focus on the requirements of a P2P live streaming network. First of all, streaming live content always means having time sensitive traffic. A single piece of a video stream might only be valid for a short time and has to arrive by the viewer before this deadline finishes, otherwise the receiving nodes might miss parts of the content or have to deal with a huge amount of delay.
Another point is the huge bandwidth requirement of such a network. Let's assume we have a live-video stream with one streaming node and k receiving nodes and the compressed live video-stream has a data rate of 400kb/s. To forward this content without losses to each node the complete upload-capacity of the network has to be $(k-1) \cdot 400Kb/s$ else we would't be able to serve each node. In a typical network there are nodes with different capacities, even if we have the required bandwidth capacity in total, some nodes might have less or more upload-capacity than the rate of the video stream. The network has to manage these different capacities to guarantee a stable content forwarding to each node.
Regarding to the bandwidth requirement it must be ensured that nodes width high bandwidth-capacities do not hide it in the network and only downloading the content without forwarding. To solve this problem a fairness policy is needed.
To sum up we can define the following basic requirements for live-streaming P2P systems:

- Robustness against frequently joining, leaving, or even failing nodes
- Short delivery delay
- Low data miss rate
- Balancing upload capacities
- Fairness

To fulfill these and many other requirements there exist different approaches. First this paper will present the two basic approaches of live streaming in section 2, which includes the tree-based approach 2.1 and the mesh-based approach 2.2. Chapter 3 contains some special mesh-based approaches with reference to BitTorrent and chapter 4 will discuss some other approaches to streaming live content in a P2P system. In the last chapter 5, the presented approaches are compared to each other and the possibilities to implement the systems into our current framework will be discussed.

2 General approaches

The current solutions to build a P2P based streaming system can be summarized into two basic approaches based on

their network overlay structure. There is the tree-based approach (presented in section 2.1) and the mesh-based approach (presented in section 2.2).
An overlay represents a network-structure upon an existing network which will be usually the internet with its IP-based structure during this paper. This overlay network uses the existing underlaying network to communicate or root data and messages to other network members, so called peers. A peer represents a node in the network witch offers services to other peers.
The basic idea of the tree-based approach is to have a solid, mostly static tree-based overlay structure and send data from one peer to the children of this peer whereas the mesh-based approach uses a more dynamical overlay structure, where peers connect to each other based on their available content and bandwidth.

2.1 Tree-based approach

In a tree-basted approach the peers are connected to each other in a tree overlay structure. The peer providing the live content is the root node of this tree and is connected with two other peers which also have a maximum of two child-peers. This tree overlay structure is shown in Figure 1 a).
The routing of the content inside this overlay is very easy. The source root node streams his live-content towards his two child-peers, which receive and forward this content to its child-peers.
But this kind of overlay has a lot of weaknesses. First of all the peers at a higher level of the tree receives the live-content with a higher delay then nodes in lower levels. Another point is the stability of such an overlay if we have rapidly joining and leaving nodes. If one peer in the middle of the tree is leaving the network, the hole subtree under this peer don't have access to the live content anymore, which is shown in Figure 1 b). To reestablish connection there have to be a mechanism that reorders the tree-structure as soon as possible like in Figure 1 c).
For this reordering mechanism there are some centralised approaches, where for example the root node is responsible to manage joining and leaving nodes based on their locality and their capacity, which only works well for small numbers of peers, because the capacity of the root-node causes a bottleneck. For large numbers of peers there exists some distributed algorithms. The exact functionality of this reordering-mechanisms will not be discussed here.
Another big disadvantage is, that each peer, which is not a leaf node, represents a bottleneck for each peer in the underlying subtree and in addition the bandwidth capacity of the leaf-nodes is not used. To make use of the bandwidth capacity of the leaf nodes and to improve the bottleneck problem there exists a multi-tree streaming approach, where the source content is split up into several sub-streams. Each peer that wants to receive the live content has to join in each of this subtrees. In this scenario a peer with lower bandwidth-capacity can be placed as a leaf-node in most of the subtrees and only a few times as a forwarding node in the middle of some subtrees.

2.2 Mesh-based approach

The mesh-based streaming approach try to solve some of the problems of the tree-based approach by using a more dynamic network overlay. One peer can establish connection to multiple other peers within the streaming network and also download and upload the live-content to multiple nodes. If one node leaves the network a node, which had a connection to this node, is still able to download the content from his remaining connected nodes and has some time to replace the disconnected node.
"Similar to P2P file sharing systems like BitTorrent, a mesh streaming system has a tracker to keep track of the active peers in the video session" [5]. If a node wants to join the streaming network it has to register with the tracker. After that the tracker sends a random list of already registered nodes within the streaming network. The new node can try to establish connection to some of these nodes from the list with best possible conditions like bandwidth, workload, delay, etc. and start to exchange content. This process is shown in Figure 2.
In contrast to the tree-basted approach the mash based approach is extremely robust against peer churn. If a peer disconnects from the network it informs the tracker and all its connected peers. If a node crashes the tracker and the connected nodes delete the dead peer after a predefined timeout. To find new peers a peer can ask the tracker or its remaining connected nodes for an actual list with active nodes and establish new connections.
To exchange the live content the source node split its content into small chunks with unique increasing sequence numbers and forward the chunks to its connected peers. Each peer buffers this chunks and individually pushes them forward. The multimedia player can display the live content by reordering the chunks according to their sequence numbers. One disadvantage of the push-forward approach is redundancy. One node may receive the same chunk from multiple nodes which cause unnecessary high traffic. This can be avoided with the pull approach.
In a pull approach each peer offers a buffer map containing all its current buffered chunks. A peer can look up the buffer maps of its neighbours and send requests to pull the missing chunks. Although this approach solves the redundancy problem, it requires a lot of communication among the peers which can delay the chunk retrieval.

3 Solutions/approaches based on BitTorrent

3.1 General functionality of BitTorrent

BitTorrent uses the upload capacities of downloading peers just like an ordinary P2P filesharing protocol to offer

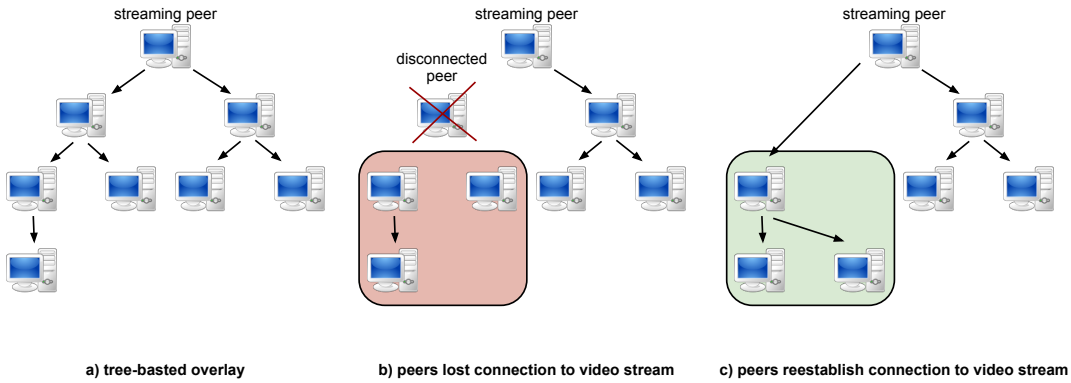


Figure 1. tree-based P2P streaming overlay [5]

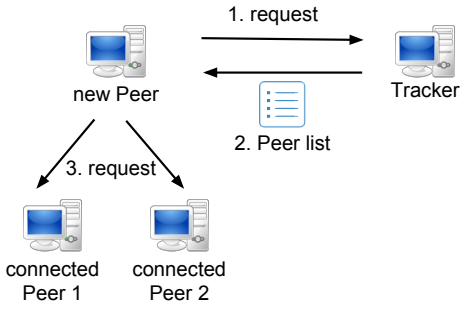


Figure 2. connecting to a mesh-based P2P streaming network [5]

a fast and efficient distribution of large files over the internet.

There are two different roles of peers, the seeds and the leeches. A seed have already successfully downloaded the content and keeps uploading whereas a leech is still downloading the content.

In BitTorrent the source file is split into several pieces with a typical size of 256KB. By these peers are able to simultaneously download missing pieces and upload already finished pieces.

As mentioned in the mesh-based approach above, BitTorrent uses a centralised tracker. This tracker keeps track of all active peers in the network and its upload and download ratios. In addition it holds information about the file, the pieces of the file, and its hash values for validation. Peers offers information over all finished pieces to all its neighbours. A peer can simply search for missing pieces on all connected neighbours. To make sure that all pieces of a file stay in the network as long as possible, a special piece selection mechanism makes sure to prefer rarest pieces.

To download a special content over the BitTorrent network at least the IP address of the responsible tracker is needed. Usually this is handled with a Torrent-File (.torrent) which contains information about the Tracker and a lot of information, which are also stored onto the Tracker like the file, the pieces of the file, and the hash values for validation. A peer can now connect to the tracker and ask for active peers just like in Figure 2.

Each Torrent has its own tracker and the trackers don't have connection to each other, which means that each Torrent has its own network where only the original content offered by the publisher be downloaded.

The success of BitTorrent relies on the ability to prevent selfish peers downloading content with high bandwidth from the network by offering only a some or none of its upload capacity (Free-Riders phenomenon). If a Peer stops or reduces his upload all its neighbours immediately stop or reduce uploading to this selfish peer by a so called Choking mechanism (Tit-for-Tat policy). This mechanism guarantees a large data transfer rate among all downloading peers. BitTorrent works best with large files and large numbers of downloading peers.

3.1.1 Limitations in streaming

The original BitTorrent system cannot handle any kind of time sensitive traffic with deadlines. Each piece is valid all the time and the piece selection mechanism is based on rare pieces to allow the complete file to stay in the network as long as possible.

To make live streaming possible, the piece selection mechanism has to be modified in such a way that it prefers pieces based on deadlines [9].

Furthermore time sensitive traffic requires a stable bandwidth to keep conditions of the deadlines. With the Tit-for-Tat policy of BitTorrent this would not be a problem. If a peer needs more download bandwidth it has to offer more of its upload bandwidth otherwise it would not be able to

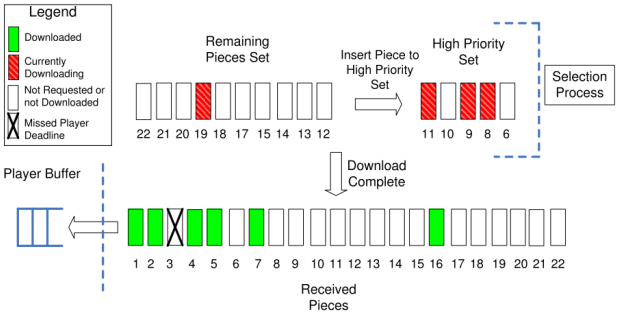


Figure 3. BiToS approach for supporting straiming with BitTorrent [9]

receive the live content in time.
But one problem that can occur is, if the total upload capacity of all nodes is smaller than the required download capacity for all nodes connected to the live stream.

3.2 BitTorrent Streaming (BiToS)

The BiToS (BitTorrent Streaming) approach makes live-streaming with BitTorrent possible by simply replace the piece selection mechanism of BitTorrent.
To handle the bandwidth problem BiToS assumes that each peer in the streaming network has at least the same upload capacity as the required download capacity. That means if we want to watch a video stream with a quality of 400Kbps we need at least an upload-capacity of 400Kbps.
Figure 3 shows the core ideas of the new piece selection mechanism. It divides the pieces of the stream into three different roles: *Received Pieces*, *High Priority Set*, and the *Remaining Pieces Set*. The *Received Pieces* contains all the pieces the peer has knowledge from. Their are marked as Downloaded, Not-Downloaded or Missed (wasn't able to download in time). All the downloaded pieces inside this set can be shared with the connected neighbours. The *High Priority Set* has a fixed size and contains all the pieces which are close to be required from the multimedia-player. The pieces in this set can be marked as Not-Requested or Currently-Downloading. At least the *Remaining Pieces Set* which "contains the pieces that have not been Downloaded, are not Missed, and are not in the High Priority Set. A piece can be in the Not-Requested or Currently-Downloading state" [9].

The actual selection mechanism works as follows. With a probability p the peer decides to download the next piece out of the *High Priority Set* and with a probability of 1-p to download the next piece out of the *Remaining Pieces Set*. To improve the algorithm it is planed to adjust the probability p dynamically depending on missed pieces, shorter deadlines, download rate etc. but this is still future work and not completely worked out yet.

If the peer decided to download out of the *High Priority Set* the piece with the closest deadline inside this set will be downloaded first. Otherwise the rarest piece out of the *Remaining Pieces Set* will be downloaded. Each peer can download multiple pieces at once up to a fixed defined maximum number.
If a piece was successfully downloaded it will be marked as downloaded in the *Received Pieces*. If this piece was downloaded out of the *High Priority Set* it will be replaced by the piece with the closest deadline within the *Remaining Pieces Set*.
Another part of the piece selection mechanism is the *Piece Deadline function* that checks whether a piece can be downloaded in time before its deadline finishes. If this is not the case the piece will be marked as Missed and ignored from the selection mechanism.
A fist implementation of this approach has shown that this approach is feasible and "can be easily incorporated into the original BT protocol" [9].

3.3 PULSE

"PULSE, an unstructured mesh-based peer-to-peer system designed to support live streaming to large audiences under the arbitrary resource availability as is typically the case for the Internet." [7]. PULSE focuses on flexibility, scalability, and robustness and makes use of the BitTorrent Tit-for-Tat policy to reward user participation and to prevent peers keeping back their upload-capacities.
Joining nodes are placed in the network according to their capacities. If a node with a high upload-bandwidth is joining the network it will be placed near the source with the ability to forward maximum amount of recent data to its neighbours. According to the Tit-for-Tat policy the node providing huge upload benefits with low delay and the rest of the network benefits with higher reception performance. On this way PULSE makes use of the maximum upload-capacities of all nodes.
Connections to other peers are established very dynamically and depend on pairwise negotiations, data exchanges, topological characteristics and capacities.
To stream the live-content the source node splits the content into several chunks of a fixed size and tags them with their encoding time. These information can be used at the receiving nodes to rebuild the original stream and to calculate their own delay.
Nodes can freely communicate and exchange information between each other. Except the source node each node has the same functionality. For the data exchange between nodes a node has three main components, the *data buffer*, the *knowledge record*, and the *trading logic*. The *data buffer* of a node is shown in Figure 4 where the chunks are ordered according to their time-tag. Newest chunks are on the left and oldest on the right. All chunks of the source stream are collected and stored in this buffer. The *Sliding Window*

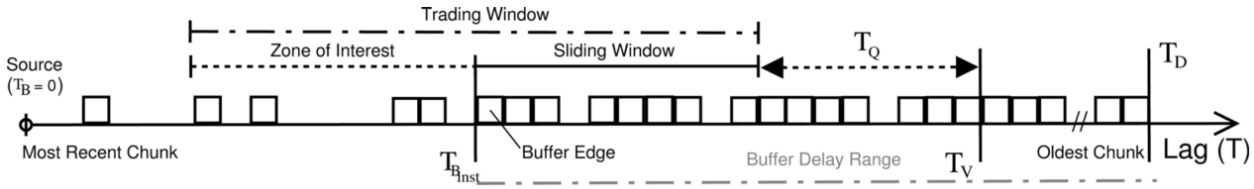


Figure 4. Data buffer of a PULSE node [7]

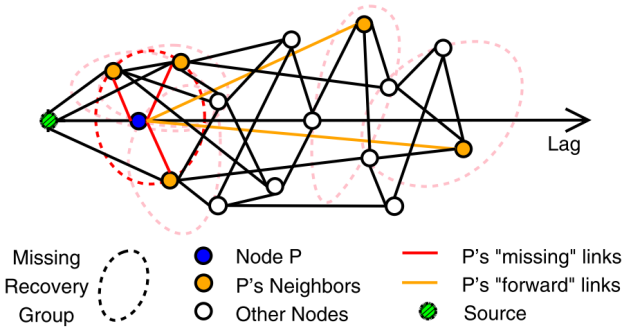


Figure 5. Data exchange sets of a PULSE node [6]

contains some chunks that are currently displayed and regulates the stream reception. The *Zone of Interest* contains all chunks which are close to be displayed. Together they are labeled as *Trading Window*.

The *knowledge record* is shown in Figure 5. All neighbours of a peer are divided into two sets (*MISSING* and *FORWARD*). The missing set contains all nodes with an overlapping trading window to this peer, because of the overlapping trading window both peers have chunks the other one needs. This peers establish a bidirectional connection and forward chunks to each other. The forward set of a peer contains only nodes which don't have an overlapping trading window and the peer is only forwarding chunks to. The connection between this nodes is unidirectional. This mechanism allows a low streaming delay.

At least the *Trading Logic* is responsible to manage the data exchange based on the information out of the buffer and the knowledge recording mechanism. It processes incoming chunk requests and calculates which chunks are forwarded to which neighbours.

Even if there is currently no complete implementation, first simulation results have shown that this approach works reasonably well.

4 Other solutions/approaches

4.1 SplitStream

In this approach SplitStream [2] is only a part of the overall live-streaming solution which is based on Pastry [8], Scribe [2] and SplitStream [2]. In this System Pastry takes on the task of Routing and Scribe makes use of the routing mechanisms of Pastry to construct multicast trees and offer the ability to multicast data. Finally SplitStream uses the functionality of Scribe to generate multiple multicast trees for one multicast. Overall this whole system represents a multi-tree streaming solution.

In Scribe a new multicast tree is generated as follows. The source peer generates a *topicId* for the new multicast tree and makes the node with the id closest to this *topicId* responsible for it. This node now becomes the root of the tree and is called *rendez-vous point*. Each node can join the multicast by sending a special *subscribe* message containing the *topicId*. This message is routed via Pastry to the *rendez-vous point*. At each visited node during the routing process it will be checked if the current node is also a subscriber to this topic. If so, this node adds the requesting node to its forwarding list and forward from now on the content of the topic to this node.

Scribe without SplitStream provides a single-tree streaming solution. The main problems of such a single tree solution were introduced in section 2.1. Single nodes in the tree might not have enough capacity to forward the live content to all its neighbours in time whereas the capacity of the leafnodes is not used within the streaming network. SplitStream is now using Scribe to split the main multicast stream into multiple multicast trees called *stripes* to eliminate these problems and distribute the load over all peers within the streaming-network while taking into consideration the different capacities of single peers.

The basic approach of how this is done is shown in Figure 6 where the source content stream is split into two stripes. In this example each node except the root-node is only one internal node in one subtree, receiving content from two nodes of different stripes, and forwards his content towards two child-nodes within the same stripe. It is assumed that each node has the same bandwidth-capacity which has to be at least an upload and download-rate of the bandwidth requirement for the live-stream.

In general if there are different nodes with different upload and download capacities the source content is split into several stripes where nodes width a download rate lower than the live stream bandwidth requirement may just receive a

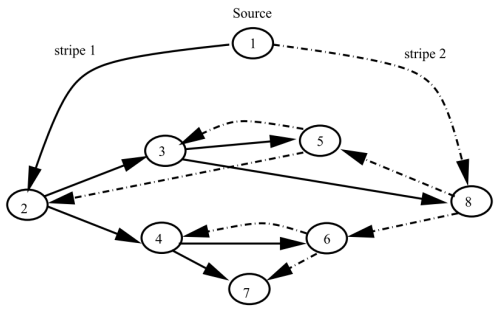


Figure 6. Basic approach of SplitStream multiple multicast trees [2]

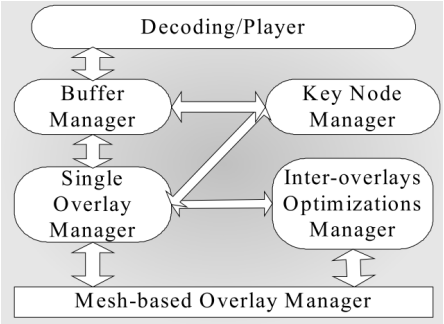


Figure 7. basic workflow of an Anysee node [4]

subset of the stripes. The other way around nodes width a lower upload rate may only forward to some nodes less then the number of stripes. Nodes with higher upload rates can compensate this weak nodes and forward the live-content to much more nodes. The exact functionality of this balancing mechanism is rather complex and can be found in [2].
SplitStream is also quite robust against node failures. If a node fails its neighbours miss only content of one stripe and are still able to display the live-stream with lower quality while trying to replace the failing node.
The SplitStream approach has already been successfully implemented in Java on top of FreePastry. The source-code and some tutorials to integrate it into FreePastry can be found at [1] which can be quite useful to integrate SplitStream into the current P2P-Framework.

4.2 Anysee

Anysee [4] uses a mesh based streaming approach. It solves the streaming delay problem by connecting peers based on their locality and their delay, even if the nodes have different overlays. It also provides a solution to balance the load over all network members. The whole system is quite complex, therefore we will only discuss some basic functionalities in this paper.
Figure 7 shows the basic workflow of an Anysee node. First of all a Mesh-based overlay is constructed.
The *Mesh-based Overlay Manager* manages the underlying network topology of the node and find neighbours based on their locality, their capacities and their joining time. Therefore it communicates with some initial peers.
The *Single Overlay Manager* is responsible for the joining and leaving of the node into the streaming overlay. It calculates delays and establishes an active streaming path in the network where the live-content is streamed towards the node based upon the calculated delays. It should also be pointed out that a streaming path can consists of several nodes forwarding the live-content towards our current node.
The *Inter-overlay Optimization Manager* manages and searches for backup streaming paths using the mash-based-overlay. If the current streaming path fails or a path with lower delay was found, the *Inter-overlay Optimization Manager* reestablish a new connection to this one of its backup paths.
The *Key Node Manager* deals with forwarding requests and is responsible for an optimal recourse utilization.
At least the *Buffer Manager* receives the live-content from the active streaming path and display it with the player according to timestamps and deadline conditions.
Anysee was successfully implemented and released in CERNET (China Education and Research Network) of China and its source is free to be downloaded.

5 Comparison

In this paper some P2P live streaming approaches have been presented. The research in this area is huge, therefore the presented approaches are only a small part of the overall existing approaches. Most likely each approach has its own focus and its own strength and disadvantages. In section 1 we defined some basic requirements on a P2P based live streaming network. Table 1 gives an overview which requirements and some additional properties the presented approaches can fulfill.
First of all the *BiToS* system which is based upon the mesh-based BitTorrent adopts all the great properties of BitTorrent like the Robustness against frequently joining, leaving, or even failing nodes, fairness caused by the tit for tat policy, etc. but its ability to have a short delivery delay is currently not completely worked out. It has no balancing mechanism for different upload capacities, because it assumes that each node in the network has at least an upload bandwidth equal to the required stream bandwidth, which is not the case in a typical network. There are only some simulation based implementations and even if an integration in our current framework might be possible by taking BitTorrent and simply replacing the selection mechanism, there might be better

Requirements/Properties	BiToS	PULSE	SplitStream	Anysee
Robustness against frequently joining, leaving, or even failing nodes	✓	✓	✓	✓
Short delivery delay	(✓)	✓	✓	✓
Low data miss rate	✓	✓	✓	✓
Balancing upload capacities	-	✓	✓	✓
Fairness	✓	✓	-	-
Running implementation	(✓)	(✓)	✓	✓
integration in our current framework	(✓)	-	✓	(✓)
Mesh-based	•	•		•
Tree-based			•	

Table 1. Overview which Approaches fulfill the requirements specified in Section 1. (✓) marked requirements are indirectly or only partially fulfilled.

solutions.

PULSE seems to fulfill all our defined requirements. Its mesh based structure is robust against frequently joining, leaving, or even failing nodes and the fact that nodes are placed in the network according to their capacities offer a full usage of their upload bandwidth and causes a short delivery delay. Its ability to find neighbours width an overlapping delay-window causes low data miss rates and the tit-for-tat policy of BitTorrent solves the fairness problem. Unfortunately there exists no complete implementation, except some simulations, which makes an adoption into our current framework nearly impossible.

Unlike the previous approaches, *SplitStream* is already successfully implemented and the sourcecode is free to download. It is based upon Pastry and Scribe and uses a multi-tree-based overlay structure. While the multi-tree structure can handle failing nodes, the underlying Pastry network ensures low delivery delays and miss rates. *SplitStream* splits the main stream into several multicast trees to balance the load over all nodes and handle different kinds of capacities. There is no special mechanism to force fairness. It is assumed that the nodes are willing to offer their upload-capacity. Over all it can fulfill almost all of our basic requirements and is build upon Pastry which is also used in our current framework. An integration should be possible with reasonably afford.

Finally the mesh-based approach *Anysee* establishes connections between nodes based on their locality and their delay which causes a short delivery delay and a low data miss rate. It also continuously search for faster and backup streaming paths to improve the delay and dealing with failing nodes. The nodes itself are responsible for an optimal recourse utilization which offers a good usage of the upload capacities but can make some problems in reference to the fairness. There also exists a complete implementation which is free to download but it would make much more afford to integrate this system into our current framework.

In this paper we focused largely on live streaming. Concluding it's also worth mentioning that there are also some approaches to offer P2P based Video-on-Demand (VoD) services like BASS (BitTorrent Assisted Streaming System for Video-on-Demand) [3], which makes use of BitTorrent by adding an external media server to the network which contains the video file. The only modification made on BitTorrent is to disallow the download of data until the current playback point to make use of the tit-for-tat policy and the rarest first mechanism. If the limits of the media server are reached, the nodes are able to forward the data over BitTorrent.

References

[1] SplitStream Implementation. https://svn.mpi-sb.mpg.de/trac/DS/freepastry/wiki/tut_splitstream.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the ACM symposium on Operating systems principles*. ACM, 2003.

[3] C. Dana, D. Li, D. Harrison, and C.-N. Chuah. BASS: BitTorrent Assisted Streaming System for Video-on-Demand. In *Workshop on Multimedia Signal Processing*. IEEE, 2005.

[4] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng. AnySee: Peer-to-Peer live streaming. In *Proceedings of the International Conference on Computer Communications*. IEEE, 2006.

[5] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1, 2008.

[6] F. Pianese, J. Keller, and E. W. Biersack. Pulse, a flexible p2p live streaming system. In *Proceedings of the International Conference on Computer Communications*. IEEE, 2006.

[7] F. Pianese, S. Member, D. Perino, J. Keller, and E. W. Biersack. Pulse: An adaptive, incentive-based, unstructured p2p live streaming system. *IEEE Network*, 9, 2007.

[8] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the International Conference on Distributed Systems Platforms Heidelberg*. Springer-Verlag, 2001.

[9] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: enhancing BitTorrent for supporting streaming applications. In *Proceedings of the International Conference on Computer Communications*. IEEE, 2006.

Insecure FreePastry: Routing Attacks

Payman Alavi
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: Payman@Mail.Uni-Paderborn.de

Abstract

Pastry is one of the most prominent peer-to-peer DHT overlays which is used in variety of applications since its release in 2001. Pastry provides an application-level routing for the nodes which are connected via the internet. In case of no adversary, a message will be delivered to the destination with a high probability but, even a small fraction of malicious nodes can prevent correct routing in Pastry and reduce the performance and functionality of the overlay dramatically. In this paper, the routing attacks and the ways to achieve a secure message forwarding in Pastry/FreePastry are studied. In addition to the routing attacks, the Sybil attack and the Eclipse attack are also reviewed because although they are not routing attacks, but they can be used to prevent the correct message delivery in the overlay.

1 Introduction

Peer-to-peer networks are decentralized overlay networks that can be used for resource sharing and various distributed applications. Pastry [19] is a peer-to-peer network overlay and Freepastry [1] is the open-source release of it. Each node that is connected to the internet and runs Pastry client software can interact with other Pastry nodes through the overlay. Each node can join the overlay at any time and leave it without any previous warning. This continuous leaving and joining the overlay is called churn. Although Pastry handles the churn successfully but even a small proportion of the malicious nodes can stop correct routing in the overlay. In routing attacks the adversaries can stop message forwarding or route the message to a wrong destination.

Secure routing is an important subject of study in Pastry and there are several different available approaches to secure Pastry against routing attacks. These different attempts reveal that secure routing is not an easy goal to reach.

In addition to routing attacks, the Sybil [11] attack and the Eclipse [21] attack are also important in the field of secure routing. Although the Sybil attack and the Eclipse attack are not routing attacks, but they can be used for this purpose.

In the Sybil attack, the adversary creates many fake identities. By the use of these fake nodes, the adversary get a better overview of the network and take the control of the overlay. consequently these malicious nodes can route the messages to the wrong destinations or stop forwarding them.

In the Eclipse attack, the adversary poison the routing table of the other nodes by providing wrong information. The target nodes lose the ability to communicate with other nodes except the adversary.

Studies [5, 26] show that in order to have a secure routing in Pastry, besides secure message forwarding, secure nodeID assignment and secure routing table maintenance must be concerned. The rest of the paper is organized as follows. Section 2 gives some background about Pastry and possible attacks. In section 3, 4 and 5, the Sybil attack, the Eclipse attack and the routing attacks are presented and in each section some of the available solutions are reviewed. At the end, section 6 provides the related work, and section 7 concludes.

2 Background

The information in this section is based on [5, 19]. Pastry is developed by Microsoft labs research, Rice university, Purdue university and university of Washington [20]. FreePastry is a modular, open-source release of Pastry, which is introduced by Rice university as an prototype of Pastry for deployment in the internet. A random ID is assigned to each Pastry node when it joins the network. In comparison to Pastry the node identifier space in FreePastry is larger than Pastry. The node ID in Pastry is 128 bits but in FreePastry it is 160 bits.

Pastry is a structured peer-to-peer overlay, which is based on the distributed hash table (DHT) system. In structured overlay networks each node has a unique identifier and is responsible for some of the objects in the network. The DHT is a distributed indexing table which is used to improve the lookup function. Every node is responsible to

maintain part of the table. In order to use the DHT, Pastry assigns a key to every object in the overlay and each key is stored in a node which is responsible for that object.

2.1 Pastry node state

Each Pastry node maintains a routing table, a neighbor set and a leaf set. The routing table consists of $\lceil \log_{2^b} N \rceil$ rows and $2^b - 1$ columns. b is a configuration number which is usually set to 4 and N is the number of available nodes in the overlay.

There are 2^b entry in the routing table and each entry in the row r of the routing table contains the ID of a node which shares the first r digits of its nodeID with the local node's nodeID. In the row r, the column which corresponds to the r+1th digit of the present node's nodeID stays empty until a node with an appropriate prefix is found.

Each node in Pastry maintains also a Leaf set which is consisted of L nodes that their nodeID numerically is closer to the local node. $|L|/2$ of the leaf set node's nodeID are numerically lower and the other $|L|/2$ are higher than the local nodeID.

The neighborhood set contains the nodes that are physically closer to the local node. Usually the neighborhood set is not used for the routing purpose. Figure 1 shows a Pastry node with the nodeID of 10233102.

Node-ID 10233102

Leaf Set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing Table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighbourhood Set

13021022	10200230	11301233	31301233
02212102	223011203	31203203	33213321

Figure 1. state of a node with nodeID of 10233102, leaf set $l = 8$, $b = 4$ and all numbers are in base 4. The top row number in the routing table is zero. The shaded cells show the nodeID of the current node. The nodeIDs have been split to show the number of common prefix with 10233102 nodeID. [19]

2.2 Routing in Pastry

To find the responsible node for a key, first each node compares the key with the entries in its leaf set. If the key is equal to one of nodeIDs in the leaf set, the request is forwarded directly to the destination node. If there is no match in the leaf set, the Pastry node checks its routing table and looks for an entry which shares at least one more common prefix with the key. If there is no such an entry available in the routing table, it looks for a nodeID with the same number of common shared prefix but numerically closer to the key. Such a node will be available at the leaf set. At each step of the routing process the request will get closer to the destination because it is sent to a node which shares more common prefix digits or numerically closer than the current nodeID. Pastry is very efficient in routing and the maximum number of hops from each node to any destination is $\log_{2^b} N$. Figure 2 depicts the routing of a message to a node.

2.3 Node arrival

When a new node X wants to join the Pastry overlay, first it generates a nodeID which is generally the hash of its IP address. Then it sends a request to the physically closest node A in the overlay. The node A forwards the request to the node Z, which is numerically closest to the newly arrived nodeID. X gets its neighbor set from Z and completes its routing table from the state table of all the nodes between A and Z. At the end X announces its existence by sending its state table information to all the nodes in its routing table, neighbor set and leaf set. These nodes will also send their updates to node X.

2.4 Node failure

Nodes in Pastry may fail or leave without any previous warning. Each node periodically checks its table entries to see whether they are alive or not. The node is considered failed If it does not respond to these checking messages. If the failure happened in the leaf set, then the node updates with the leaf set of the furthest node. To cope with failure in the routing table, after detecting the failure, another node in the same row will be used instead of the failed one. If all the nodes in a row fail, then the nodes in the higher row will be used instead.

2.5 Security issues

In a real peer-to-peer overlay, adversaries are part of the network. The adversary can be a malicious node that follows a protocol which is harmful to the network. The malicious node can generate arbitrary packets with forged source IP or receive only the packets that are addressed to itself. The erroneous nodes which do not follow the protocol

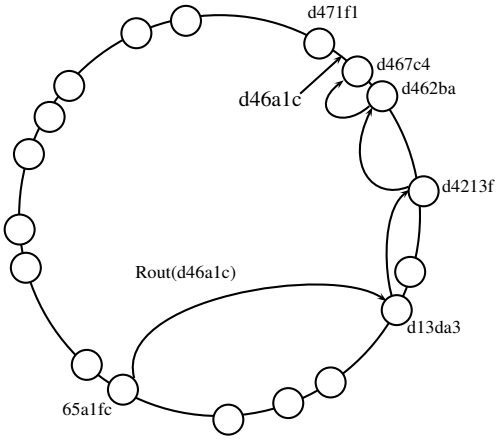


Figure 2. Routing a message from nodeID 65a1fc with key of d46a1c. The small circles represent the live nodes in Pastry's circular identifier space. [26]

correctly are also considered as an adversary in peer-to-peer networks.

Attacks can occur at the application layer or at the network layer. At the application layer, an adversary can provide false information. For example, it can deny the existence of a data, or a node may report less bandwidth than its real bandwidth and in this way receives less traffic. Free-riding is also a problem that the selfish nodes benefit from the overlay by using the others files but do not share anything.

As a network layer attack, a node can forward a message to a non-existent address or a wrong destination. This leads to increase in network traffic and decrease in bandwidth usage efficiency . An adversary can also provide false information for the other nodes and pollute their routing tables with incorrect information. Another problem can be occurred in bootstrap phase.The action of sending request to join the overlay is called bootstrapping. If a new node which wants to join the network sends the join request to a malicious node, then the malicious node can lead the newly arrived node to a virtual overlay which consists of only malicious nodes. This situation is called network partitioning. If a new node bootstrap to such a partitioned network all its actions can be observed, and it can be controlled by the adversary easily.

There are some other types of attacks in Pastry like returning wrong data or deny the existence of a certain data in the network. Distributed denial of service (DDOS) is also another type of attack, which is not related to the topic of this paper. In DDOS attack, the malicious nodes try to poison the routing tables of the other nodes and address a popular file to a wrong node. Consequently, a large number of requests will be forwarded to the target node, and this node reaches its threshold and stops responding to the other nodes. As mentioned before, undoubtedly two important types of attacks in the DHT overlays are the Sybil attack and the Eclipse attack which can be used also for routing attacks.

2.6 Sybil attack

In the Sybil attack the adversary creates many fake nodes and joins the network with numerous forged identities. As a result, the adversary gets a better view of all the network and can control other nodes more easily. In addition when all the fake nodes join the network the bootstrapping action cause a lot of stabilization traffic in the network. The problem will get worse if the overlay design, needs to transfer a lot of data in the bootstrapping phase. On the other hand, when all the forged nodes fail suddenly it causes to decrease the robustness of the network dramatically. These fake nodes potentially have the power to partition the network and create a parallel network to hijack the newly arrived nodes.

2.7 Eclipse attack

In all types of the DHT overlays a node has some links to some of its neighbors. In Pastry routing table entries are considered as links. If the adversary keeps the control of the neighbors of a good node, then this valid node can be Eclipsed from the rest of the nodes. The attack can be done by sending inaccurate updates to the valid nodes, consequently the routing table of the good nodes will be filled by totally wrong information after a while. That is why the Eclipse attack is also called routing table poisoning in some documents like [16, 6]. The adversary can stop forwarding of the good node's messages to the rest of the network and also can hide the good node from the other nodes of the overlay.

The Sybil attack can be used to perform the Eclipse attack and the Eclipse attack can be used to smooth the way of the routing attack. Thus, in this paper the approaches to defend Pastry against the Sybil and the Eclipse attack will be reviewed in addition to the routing attacks.

3 Secure assignment of node identifiers

In order to have a secure assignment of node identifiers the overlay must be protected from the possibility of the Sybil attack. The Sybil attack is important to study because it was a possible attack in the DHT overlays, and it can cause other kinds of attacks like routing attacks. The Sybil attack is first studied by Douceur [11]. According to his book, each physical entity can have a logical identity in a system. If the system cannot control the relation between entities and identities, then the adversary can create numerous identities and deceives the other nodes. In Pastry, if an adversary creates many fake identities, it becomes easier to pollute the routing table of the honest nodes. With many virtual identities controlling the network becomes easy. In

order to diminish the possibility of routing attacks, secure assignment of the node identifier is essential. This section of the article reviews the solutions for the Sybil attack which is the most important threat against the secure assignment of node identifier.

3.1 Certificates signed by a trusted authority, possibly payed

Castro et al. [5] that, the best practical way to have a secure node assignment is the use of some trusted authorities. When a node wants to join the network, it sends a request to one of these trusted authorities to get an identifier. An identifier includes of a random number, the IP address of the entity and its public key. In this way making several identities becomes more difficult. Douceur [11] also mentions this method as an only practical solution.

Castro et al. also suggest to charge the new arrival nodes with money. As a result in the large networks making large number of identities to control the overlay costs too much. This method cannot be useful in small networks, because there is no need to purchase numerous certificates. Charging the users can also lead to the slower increase in the number of the overlay's nodes because even some of the honest users stop joining the overlay because of the charge.

3.2 Use of physical network characteristics to identify nodes

Wang et al. [27] claim that the Castro et al. [5] ideas to use the IP address in the identification of the nodes and charging the users with money in order to obtain a new identifier are not practical and reject them. They focus on using the physical network overlay information to improve the performance and security of the DHT overlay. They propose to use an special combination as an identifier consisted of the IP address of the node's default router, Media Access Control(MAC) address and Round Trip Time(RTT) between the node, and some randomly selected nodes in the same sub-network. They call this combination net-print and claim that using of this method reduce the possibility of cheating in the nodeID assignment dramatically. They mentioned that even if the adversary creates fake nodes, the scope of the problem is limited to the size of that node's sub-network which is small in comparison to the whole network and can be detected easily. Since their solution is based on the physical network layer, every change in the network and nodes location can result in the change of the nodes identity and the overlay. As an instance, a change in a node location, changes its default router and this leads to the change of the node identification. Also many nodes identifications face change because of the roll of the RTT as an element of the identifier space.

3.3 Use of network coordinates

Bazzi and Konjevod [2] solution is based on the physical location of the nodes. They propose to use a geographical location-based certificate to prohibit the adversary from creating several malicious nodes. In their approach, nodes are categorized to two categories of the beacons and the applicants. The beacons are used to locate the physical location of the applicants. In fact, they are used to create a geometric certificate based on the distance between the beacons and the applicants. The distance is obtained by assessing the delay of a certain message between the beacons and the applicants. The way of calculating the location and the distance between the nodes can be compared to the GPS system as they share some common methods. However, if the adversary controls several nodes in different places this defence method becomes disable to find it out. Furthermore, Szymaniak [24] shows that the method which is used to calculate the distance between nodes may generates false information, thus the unique distance parameter between nodes is not always a reliable source of information to be used.

3.4 Use of social network

Yu et al. [29] suggest the use of social network concept. In this approach, the overlay graph edges are considered as the human friendship. For example, the owner of a node in the overlay who trust the owner of the other node can connect to it, and a trust relation will be created between these two nodes. Each node creates a network of its trusted nodes. An adversary can create a large number of corrupt nodes, but they are not a threat to the network as long as these nodes are not connected to the honest nodes. Each node has a route to its friends nodes and when a new node X asks to connect a node N, N checks the intersection routes between X and itself. If half of the N routes intersects with any routes in X, then N accepts X because it is more likely that X is a friend or a friend of a friend. By connecting to more trusted nodes the possibility of the adversary decreases.

Each node creates its own partition of network based on random route, which is a modified version of the random walk method. "Random walks is a typical search schema in unstructured peer-to-peer network" [30]. Two important specification of random rout are as follows; First, Merging the intersected routs which are in the same direction and second, the implementation of back-traceable routs. The authors also present some approaches to securely establish these routes such as using a pair of public/private key, a symmetric key for each social and besides maintaining two tables for each node. These tables are called the register table and the witness table which are equal to each other, but the orders of the rows are inverted. These tables are used to prevent nodes from lying about their routs.

3.5 Hierarchical Computational puzzles

Rowaihy et al. [18] propose a tree structured hierarchical approach in which the root of the tree is a completely trusted node. When a new node X wants to join the network first it

generates a public/private key pair and an identifier which is the hash of the public key. Then it sends a request to a leaf node Y. The node Y provides a cryptographic puzzle [17] which consists of two parts: a known and an unknown part. The known part contains the hash of the random number of the unknown part, and the goal of the player is to discover the unknown part by using its hash. If X solves the puzzle, then it receives a token from Y and afterwards by showing the token to Y's parent it receives another puzzle to solve. This process repeats until X reach the root node and by solving the puzzle provided by the root, it receives a special token which is acceptable to the entire network.

Initially, the newly accepted node cannot admit the other new nodes until it stays in the network for a long period of time. To promote to the admission tree the authenticated Diffie-Hellman key exchange method [9] is used.This method is used to establish a shared key with its parent to admit newly arrived nodes.

If a node wants to leave the network, it informs all of its children and one of them which has been stayed in the network for a longer period of time is replaced with the parent. In case of node failure all of its children must execute the procedure of re-joining the network . If the attacker is a member of the tree, then it can accept the new nodes without asking to solve a puzzle and reduce the number of puzzles that must be solved. This problem can be discovered by the parent of the malicious node by checking the rate of token requests. If the adversary is not a member of the tree, then it can only obtain nodes slowly by solving the puzzles. As the size of the tree grows it becomes more time-consuming for an adversary to solve many puzzles and join the network. Rowaihy et al. approach does not prevent the Sybil attack, but it can limit it in this way.

4 Secure routing table maintenance

If an adversary takes the control of the neighbors of a node, it can isolate this node from the rest of the network. One of the ways for taking the control of the nodes in Pastry is to send them wrong updates. By polluting the routing table of the nodes, the adversary becomes able to do various kinds of attacks like the Eclipse and the routing attacks. Therefore, the nodes should maintain their routing table in an appropriate way. They should use some mechanisms to distinguish wrong updates from the other nodes. The Eclipse attack is the most prominent attack that target the routing table maintenance of the nodes in the overlay and benefits from the poisoning of them. In this section of the article proposed solutions against the Eclipse attack is reviewed.

4.1 Use of two routing tables

Castro et al. [5] propose the use of two routing tables. One table which is used for proximity information and one, which does not exploit proximity. The first one is called the Optimized routing table and the second one is called the verified routing table. Normally the Optimized Routing Table is used for routing purpose and the Verified Routing Table is just used in case of the failure of the efficient routing with the Optimized routing table. They modified Pastry to use this method. In this method, the bootstrap node plays an important role, so it is essential to have several redundant bootstrapping node to ensure that at any time at least one of them works properly. To choose the bootstrap node, they propose the use of nodeID certificate. However, Condie et al. [7] shows that the Optimized Routing Table is vulnerable, and it can be poisoned after a while. In this situation efficient routing with optimized routing table becomes impossible and the Verified Routing Table is the only option for routing. Checking the poisoned routing table and then referring to the Verified Routing Table increases the routing cost consequently. Hence, they claim that this method increase the cost of routing dramatically after a while.

4.2 Resetting of optimized table entries and continuous induced churn

As mentioned in previous section Condie et al. showed that using two routing tables is not a good way of preventing the Eclipse attack because the Optimized Routing Table is in danger of poisoning over time. To decrease the chance of poisoning, they introduce induced churn. In this way, they limit the routing table updates since it is the source of the problem. The authors claim that by moving the good nodes in the identifier space continuously, the adversary cannot attack them easily so they propose that at each reset of the routing table each node receives a random number as a new identifier and changes its position in the identifier space. This fresh randomly generated identifier is created by a trusted timed randomness service, which signs and sends it securely. This method can reduce the probability of the Eclipse attack, but by changing the identifier all the data should also move to a new place, and it creates another problem in the field of data-storage, which needs to be considered.

4.3 Use of routing table entries based on network proximity

Hildrum and Kubiatowicz [15] propose some modification in Pastry routing table to make its maintenance more secure. As described in the background section of this document each row of the Pastry routing table consists of the nodes with some common prefix. They propose to select and store the physically closest nodes, in each row of the table. This method can be helpful to reduce the Eclipse attack because in this way it is not possible for the attacker to appear as the locally closest node to a large fraction of the good nodes. In case of the availability of a trusted distance measurement this method is useful and simple to implement.

4.4 Control of the in-degree and out-degree of overlay nodes via anonymous auditing

Singh et al. [21] test the Hildrum and Kubiatowicz approach [15] and claim that this method is practical only if the nodes are separated in a vast area, otherwise the concept of physically closest nodes becomes meaningless. They also claim that by using Hildrum and Kubiatowicz approach, controlling the Eclipse attack can be possible just in the small networks.

In the Eclipse attack the in-degree of the adversary becomes more than the average in-degree of the rest of the overlay nodes. This can be controlled by preventing the good nodes from selecting the nodes with in-degree of more than a certain threshold in the process of routing table maintenance. On the other hand, the malicious nodes can increase the in-bound degree of the valid nodes by pointing them and increasing their in-degree. Preventing the good nodes from selecting the nodes with a large in-degree can prevent other nodes to select these valid nodes too. To solve this problem, limiting the out-degree of the nodes is also proposed.

In this approach, each node X implements a list which is referred as Backpointer set . Node X only accept to forward the messages from the nodes which are in its Backpointer list. This list contains the neighbors of the node X and X periodically challenges them to be sure that they are not malicious nodes. If a node from the mentioned list answer less than k challenges out of the n challenges, then it is considered as malicious node and is wiped out of the list. In this method, the probability of a correct node to be considered as a malicious node is:

$$\sum_{i=0}^{k-1} \binom{n}{i} f^{n-i}(1-f)^i$$

Where f is the fraction of malicious nodes in the network and n is the number of challenges. The probability that a malicious node passes the challenge undetected is:

$$\sum_{i=k}^{n} \binom{n}{i} [f + (1-f)c/r]^i [(1-f)(1-c)]^{n-i}$$

Where c is the probability of the malicious node to answer the challenge and r is the ratio of the size of its true set versus the maximal allowed size. Node X challenges the nodes in its Backpointer set and removes a node from this list if the response to the challenge does not include X or if the response to the challenge has more entries than the in-degree threshold.

5 Secure message forwarding

As mentioned before, the Sybil and the Eclipse attack are not considered as the routing attack, but they can be used for this purpose. In order to forward a massage to the destination in Pastry basically several nodes are responsible, because in each step of the routing the responsible node forwards the message to another node which is closer to the target. If an malicious node appears as one of the intermediate nodes between the origin of the request and the destination, the routing system can face a problem. One of the important reasons is that the initiator has no control over the intermediate nodes as displayed in figure 3. As a routing attack, the adversary can forward a request to a wrong destination or simply stop forwarding the messages. It can also pretend that it is the destination node. Therefore [5] the probability of successful routing between two un-faulty nodes is $(1-f)^{h-1}$ where f is the fraction of faulty nodes in the network and h is the number of the hop counts from the source to the destination node, which in Pastry is $\log_{2^b}(N)$. In this part some proposed solutions against routing attacks in Pastry are studied.

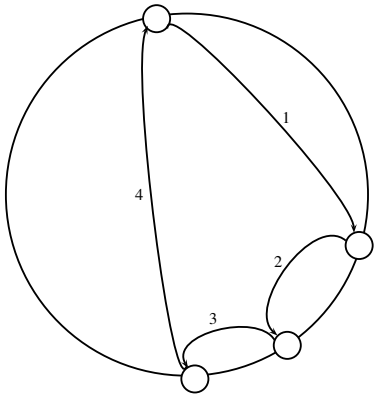


Figure 3. In the Recursive Routing method the initiator has no control over the intermediate nodes [28]

5.1 Sit and Morris guidelines

Sit and Morris [22] propose some general guidelines for this issue. Their solution is based on three elements. First, they propose to use the iterative routing strategy. As it is shown in figure 4, in this method when the intermediate node receives the message from the initiator, it sends the IP address of the hop ,which is closest to the destination, to the initiator of the message instead of forwarding it by itself. Afterward, the initiator sends the message to the next hop and gets closer to the destination in each step. This process continues until it reaches the destination. In this way, the initiator can check the lookup progress and detect anomalies such as responses that are far from the requested key.

Second they propose to use a verifiable identifier such as the hash of the node's IP and port number. They claim

that in this way, it becomes difficult for an adversary to pretend being responsible for a specific key. The last part of their solution is to use public key as an identifier. Thus each node can verify the origin of a message and the validity of their content. They also claim that having at least two replications of the data in the overlay is necessary and all the nodes must be responsible for maintaining the replicas.

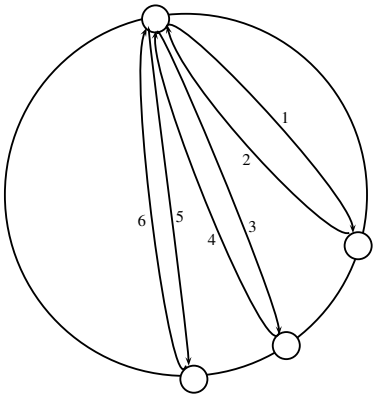


Figure 4. Iterative Routing method [28]

5.2 Redundant routing

Castro et al. [5] reject the use of iterative routing and introduce the redundant routing. The goal of Castro et al. to solve this problem is to send at least one copy of the message and the key, to all correct replicas of the destination node. As described in section 4.1 they propose the use of two routing tables. First, the node use the Optimized Routing Table, and if the efficient routing fails then it use the verified routing table. To discover the failure of the Optimized Routing Table, they introduce a failure test which decides whether or not to use the verified routing table. In this approach the Verified Routing Table is more expensive and time consuming than the optimized routing table. The failure test function, receives the key and set of root nodes for this test. It sends a message to the roots of the given key and starts a timer, if the node reply to the message late, then the test function returns positive otherwise it returns negative and the Verified Routing Table will be used.

To increase the probability of successful routing, they use redundant routing, which is sending a message to all the replicas of the destination through different routes. When the replicas are near the key node this solution is not effective enough. Therefore The message is sent also by different neighbors of the initiator and every honest node that receives the message returns its own ID certificate, which is issued by certification authority. The sender collects the certificates, and in case of time out, sends this collection as a list, to all members of this list. These nodes responsibility is to send the message to all of their neighbors which are not included in the list. Consequently, the node receives some more new certificates as an update. The author claims that by using this method the probability of reaching all replicas is:

$(1-f)^{1+\log_b N}$

Where b is the base for node identifiers, f is the fraction of the malicious nodes, and N is the number of nodes in the overlay. Again as discussed in section 5.1 the Optimized Routing Table is vulnerable to be poisoned, and in case of the use of this approach this problem must be concerned.

5.3 Use the combination of the tracer routing method with peer-ID

Xu Xiang and Tan Jin [28] claim that generating the nodeID based on IP is not secure enough because generating a fake IP address is not impossible. For this reason, they propose the ID Selector service(IS) and Private Key Generator(PKG). The nodeID is a randomly generated number by IS and to prevent the Eclipse attack, they suggest to charge the nodes a little amount of money or get a valid credit card number. Each node has a public key which is derived from its nodeID and a private key which is generated by the PKG. This private key is used for signing the messages.

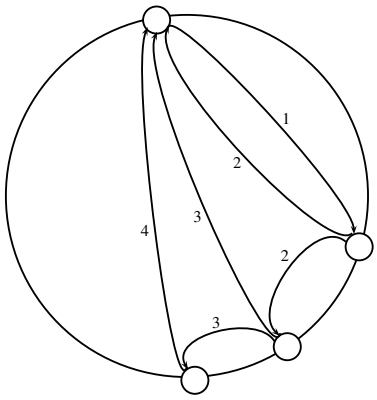


Figure 5. Tracer Routing method [28]

For message forwarding, they propose to use the combination of the tracer routing method with peer-ID. In Tracer routing method in each step an intermediate hop forwards the message to the next hop and the initiator simultaneously. By combining Tracer routing and the Peer-ID method when node Y sends a message, the intermediate node X receives the message and verifies it by using the initiator's public key which is its peer-ID. Next it forwards the message to the next hop and an acknowledgment to the message's initiator. The acknowledgment message is included of the next hop Peer-ID and Y signature which is generated by the use of private key of X . By the use of this method the initiator trace the message until it reaches the destination. In figure 5 the tracer routing method is illustrated.

6 Related work

There are some other solutions for the Sybil attacks. Dinger and Hartenstein [10] propose a distributed registration for Chord [23]. In their system, the node identifier is the hash of the node's IP and port number. When a node wants to join the network, it sends a request to the nodeID which is the combination hash of the local node's IP and a randomly generated number. In addition, the bootstrap node checks the IP address of the requester and rejects the node if it is registered more than a certain number of identity in the overlay.

Bazzi et al. [3] propose to use the hop count between each node and some beacons to prevent nodes from creating fake Identities. Danezis et al. [8] and Borisov [4] also propose some modifications for Chord overlay. For example the use of the computational puzzle that is proposed by Borisov to make the Chord overlay Sybil resistant.

For secure message forwarding, Ganesh and Zhao [13] propose a method which is called existence proof. This method can show if the next hop is truly the responsible node in Pastry. Harvesf and Blough [14] and Fiat et al. [12] also propose some development in chord to improve the security of message forwarding.

7 Conclusion

In this paper the possible routing attacks in Pastry and some of the available solutions were introduced. In order to have a secure routing system in Pastry, secure assignment of node identifier and secure routing table maintenance are essential. The Sybil attack and the Eclipse attack are two well-known threats to assignment node identifier and routing table maintenance respectively. For each type of attacks, some of the solutions reviewed, and some were named in the previous section of this paper. The variety of proposed solutions for secure routing in the DHT overlays show the complexity of the problem. As [25] concludes " securing a DHT requires secure assignment of node identifiers, a low fraction of malicious nodes, malicious nodes spread over the identifier space, data replication, and a routing mechanism that provides a high probability of reaching a correct replica set." However, still a comprehensive secure routing method in the completely decentralized peer-to-peer overlays like Pastry needs more work.

References

[1] Rice University, FreePastry. URL: www.freepastry.org/, Last accessed in MAY 2012.

[2] R. A. Bazzi and G. Konjevod. On the establishment of distinct identities in overlay networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*. Springer, 2005.

[3] R. A. Bazzi, Y. ri Choi, and M. G. Gouda. *Hop Chains: Secure Routing and the Establishment of Distinct Identities*. Springer, 2006.

[4] N. Borisov. Computational Puzzles as Sybil Defenses. In *Proceedings of the International Conference on Peer-to-Peer Computing*. IEEE, 2006.

[5] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. *Operating Systems Review*, 36, 2002.

[6] T. Condie, V. Kacholia, S. Sank, J. M. Hellerstein, and P. Maniatis. Induced Churn as Shelter from Routing-Table Poisoning. In *Proceedings of the International Conference on Network and Distributed System Security Symposium*. ISOC, 2006.

[7] T. Condie, V. Kacholia, S. Sank, J. M. Hellerstein, and P. Maniatis. Induced Churn as Shelter from Routing-Table Poisoning. In *Proceedings of the International Symposium on Network and Distributed System Security*. ISOC, 2006.

[8] G. Danezis, C. Lesniewski-laas, M. F. Kaashoek, and R. J. Anderson. *Sybil-Resistant DHT Routing*. Springer, 2005.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22, 1976.

[10] J. Dinger and H. Hartenstein. Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. In *Proceedings of the International Conference on Availability, Reliability and Security*. IEEE, 2006.

[11] J. R. Douceur. *The Sybil Attack*. Springer, 2002.

[12] A. Fiat, J. Saia, and M. Young. *Making Chord Robust to Byzantine Attacks*. Springer, 2005.

[13] L. Ganesh and B. Y. Zhao. Identity theft protection in structured overlays. In *Proceedings of IEEE Workshop on Secure Network Protocols*. IEEE, 2005.

[14] C. Harvesf and D. M. Blough. The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables. In *Proceedings of the International Conference on Peer-to-Peer Computing*. IEEE, 2006.

[15] K. Hildrum and J. Kubiatowicz. *Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks*. Springer, 2003.

[16] H. Lin, R. Ma, L. Guo, P. Zhang, and X. Chen. Conducting routing table poisoning attack in DHT networks. In *Proceedings of the International Conference on Communications, Circuits and Systems*. IEEE, 2010.

[17] R. C. Merkle and R. L. Rivest. Secure communications over insecure channels. *Communications of The ACM*, 21, 1978.

[18] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta. Limiting sybil attacks in structured p2p networks. In *Proceedings of the IEEE International Conference on Computer Communications*. IEEE, 2007.

[19] A. I. T. Rowstron and P. Druschel. Pastry: Scalable distributed object location and routing for large-scale peer-to-peer systems. *Lecture notes in computer science*, 2001.

[20] X. Shen, H. Yu, J. Buford, and M. Akon, editors. *Handbook of Peer-to-Peer Networking*. Springer, 2010.

[21] A. Singh, T. wan Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proceedings of IEEE INFOCOM*. IEEE, 2006.

[22] E. Sit and R. Morris. *Security Considerations for Peer-to-Peer Distributed Hash Tables*. Springer, 2002.

[23] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to- Peer Lookup Service for Internet Applications. In *Proceedings of the International ACM SIGCOMM Conference*. ACM, 2001.

[24] M. Szymaniak, G. Pierre, and M. V. Steen. Scalable Cooperative Latency Estimation. In *Proceedings of the International Conference on Parallel and Distributed Systems*. IEEE, 2004.

[25] G. Urdaneta, G. Pierre, and M. V. Steen. A survey of DHT security techniques. *ACM Computing Surveys*, 43, 2011.

[26] D. S. Wallach. A Survey of Peer-to-Peer Security Issues. In *Proceedings of the International Symposium on Software Security*. ACM, 2002.

[27] H. Wang, Y. Zhu, and Y. Hu. An Efficient and Secure Peer-to-Peer Overlay Network. In *Proceedings of IEEE Conference on Local Computer Networks*. IEEE, 2005.

[28] X. Xiang and T. Jin. Efficient Secure Message Routing for Structured Peer-to-Peer Systems. In *Proceedings of the International Conference on Networks Security, Wireless Communications and Trusted Computing*. IEEE, 2009.

[29] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: defending against sybil attacks via social networks. In *Proceedings of ACM SIGCOMM Conference*. ACM, 2006.

[30] Q. Zheng, Y. Wang, and X. Lu. *WCBF: Efficient and High-Coverage Search Schema in Unstructured Peer-to-Peer Network*. Springer, 2004.

Best Practices for Data Storage in P2P Networks

Alexander Mäcker
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: amaecker@mail.upb.de

Abstract

Distributed data storage is an important aspect of peer-to-peer systems since it facilitates storing and retrieval of files. To provide a good quality, it is necessary that a data storage system satisfies requirements such as persistency, security and performance. This paper focuses on several approaches that could be used to improve the quality of the peer-to-peer storage utility PAST, based on the distributed lookup service Pastry, that is utilized in the online social network under consideration. It addresses several aspects of quality of a peer-to-peer data storage system: the question of suitability of whole-file replication and erasure coding in order to provide high availability is considered; an approach to redistribute load based on the notion of virtual servers that respects heterogeneity of nodes is presented; a design for quota management based on distributed auditing as well as an attribute-based query mechanism are described.

1 Introduction

This paper mainly presents some approaches to improve the quality of the peer-to-peer (P2P) storage management PAST, based on Pastry, in terms of its utilization in the online social network (OSN) which is under consideration.

To recall the fundamentals, one can consider a structured P2P data storage system as a functional architecture consisting of three layers [15]. At the lowest level the *distributed lookup service* (DLS) is located. In the considered case this layer is implemented by Pastry [16]. Pastry mainly provides the possibility to route a message to an appropriate node, identified by its *nodeID*, in usually $\mathcal{O}(\log N)$ steps. Furthermore, it handles joining and failures of nodes by maintaining the affected routing tables. Since being advantageous for the DSS layer it should be mentioned that each Pastry node additionally maintains a so called *leaf set* that stores its $\frac{l}{2}$ numerically closest predecessor and $\frac{l}{2}$ numerically closest successor nodes. On top of the DLS layer the *distributed storage service* (DSS) is layered, which is basically responsible for the storage of files in the DHT. For that purpose the DSS PAST [16] provides an insert mechanism to store a file, identifiable by a key or $fileID$, at k different nodes and a lookup operation that retrieves a copy of a file identified by its $fileID$. Finally, the highest layer is the *distributed data management service* (DDMS) that allows to add semantic to objects stored by the DSS layer. This layer is not implemented by PAST.

For each of these functional layers several requirements have been formulated [5]. Because this paper focuses on the DSS and DDMS layer requirements concerning those are of main interest. One important requirement is the robustness of a P2P data storage system. The system should be able to handle removals and failures and the constant joining and leaving of nodes. Despite this issue data should be stored persistently, i.e. it should be highly available and should not get lost although nodes may leave the system. Additionally, it should be able to distribute resources in such a way that fast access to resources is possible, hotspots are avoided and (heterogeneous) nodes are utilized according to their capabilities. Furthermore, security is an important issue. Since nodes are usually untrusted it is necessary to prevent malicious nodes from doing harm to the system and to use techniques like access mechanisms and encryption of data to protect information from being accessed or altered by unauthorized nodes.

Several of the mentioned requirements are addressed in this paper. Detailed information about solutions in PAST regarding the considered aspects are described in the corresponding sections in order to motivate and understand the need of the elaborated improvements. The remainder of this paper is organized as follows. In section 2 several replication techniques and the idea of erasure coding are presented and compared afterwards which leads to the idea of a hybrid approach. Section 3 presents a simple idea how to support heterogeneity and redistribute load based on virtual servers. The issue of quota management in the field of security is considered in section 4 and a possible solution using auditing is stated. Section 5 deals with the question how to provide a search mechanism on top of a DHT using

an attribute-based approach. Finally, a conclusion is given in section 6.

2 Redundancy Management

Due to churn, the leaving and joining of nodes, it is important to introduce redundancy into the system to ensure that files are still available if the responsible node is offline or has lost the file. Main properties that should be taken into consideration are high availability of content and low storage overhead. In literature there are mainly two mechanisms that more or less meet these requirements and which are presented and compared in this section: replication and erasure coding.

2.1 Replication

The probable most simple approach to introduce redundancy into a P2P-system is replication. The basic idea is to maintain several replicas (identical copies) of a file at k different nodes. Thereby the choice of k is up to the user and can be chosen e.g. according to the demanded availability. If a node storing the file fails, it will be still available by accessing another available node that holds a replica.

2.1.1 Replication Strategies

In literature several kinds of replication techniques have been described. To be able to decide on the question whether PAST can be improved regarding its replication strategy three kinds of replication techniques, which mainly differ concerning the question where to store replicas, are presented in the following.

Leaf Set Replication Originally PAST utilizes a leaf set replication [2] and the basic functionality works as follows. If a user wants to insert a file, PAST first computes a 160-bit identifier, the fileID. Then the file is routed to and stored at the k nodes for which the node identifiers are numerically closest to the (128 most significant bits of the) fileID. This is done by routing towards the fileID and as soon as one of the k closest nodes is reached it stores the file and forwards it to the remaining $k-1$. Because those nodes are stored in the leaf set of the routing table, which is maintained by Pastry, this approach is called leaf set replication.

Path Replication Another approach called path replication is used in Tapestry [23]. Using path replication a file is stored at one unique node that is responsible for it. Each other node that holds a replica of the file publishes that fact by sending a message to the responsible node. At each node that is traversed a mapping (fileID, nodeID) is created.

To retrieve a file a query is sent towards the responsible node. As soon as a node that has a mapping for the corresponding file is reached the query is directly forwarded to its destination.

Symmetric Replication The last replication that is presented is symmetric replication, which is an instance of multi publication key replication [6]. The notion of that kind of replication is that a key is mapped to k different locations and thus k replicas are stored at different nodes. To be more precise, a key x is mapped to $x_1, \ldots, x_k$, for example, using the mapping $h(x_i) = (x + (i * \frac{N}{k})) \mod N$.

To retrieve a file with key x, a node can compute x_i and issue a request. Therefore, also parallel lookups are possible simply by requesting different x_i's that are associated with the same file x.

2.1.2 Comparison

In [6] successor replication, path replication and symmetric replication has been compared concerning several performance measures. Although in [6] the emulation is based on Chord it is claimed that the results are mainly independent of the underlying DHT. Furthermore it might be feasible to apply the results for successor replication as an estimation for the performance of leaf set replication, which is used in PAST. This is due to the fact that successor replication only differs from leaf set replication in the fact that replicas are not stored at the k closest nodes (predecessors and successors) but only at the k closest successors.

To mention the main results here, it turned out that symmetric replication provides the best availability while generating clearly the highest traffic load, whereas path replication provides much lower look up latency and traffic load but a worse availability. Successor replication, however, seems to be always in between the two other strategies. Basically none of the strategies has turned out to be the best. Thus, it is necessary to weigh which one is best for the purpose of an OSN.

2.1.3 Chunking

A further choice that have to be made regards the approach of chunking. In [2] it is mentioned that non-acceptance of queries, if they occur, are mostly related to the insertion of large files. Additionally, in [5] it is criticized that storing a whole file, as PAST does, is neither efficient nor fault-tolerant. The approach of chunking, the splitting of files into smaller blocks or fragments before storing, is, for example, provided by the storage system CFS [1]. This technique overcomes the problem that large files cannot be stored because a single node has not enough storage space. Furthermore it makes a parallel retrieval possible. If one decided to employ chunking, then a further possible improvement should be taken into consideration: erasure coding.

2.2 Erasure Coding

Techniques of erasure coding are related to chunking but more sophisticated strategies are employed to improve the

performance. As it is done when using chunking the first step is to split a file into m smaller blocks. Afterwards a coding technique, e.g. Reed-Solomon-Codes [10], is used to compute n blocks out of the given m, where $n > m$. Those blocks have the property that accessing any m blocks - that have roughly the same size as the original file - allow to reproduce the original file and they can be stored at different nodes.

Because the underlying mathematical coding techniques cannot be treated in this paper a simplified, hopefully more or less intuitive, example (based on Reed-Solomon-Codes [10]) is given instead to give an impression how such a code could work. Assume that a file a should be stored. First, it is spit into m fragments $a_0, \ldots, a_{m-1}$. Additionally, for simplicity reasons assume that each fragment can be described by a number from a finite field, for example $\mathbb{F}_p = \{0, \ldots, p-1\}$ for a prime p together with multiplication and addition modulo p, that is $a_i \in \mathbb{F}_p$. Now, the file a is interpreted as a polynomial of degree $m-1$

$$a(x) = a_{m-1}x^{m-1} + a_1x^1 + \ldots + a_0$$

using the fragments a_i as coefficients. If the m fragments should be coded into n fragments one can define n different $u_i \in \mathbb{F}_p$ and compute $a(u_i)\ \forall 1 \leq i \leq n$. This $a(u_i)$ are then stored in the system at different nodes. If the original file a should be restored it is sufficient to retrieve m of the coded fragments $a(u_1), \ldots, a(u_n)$ because, as known from mathematics, m different pairs $(u_i, a(u_i))$ uniquely determine a polynomial of degree $m-1$ which then is the polynomial $a(x)$ that describes the original file and can be, for example, determined using a system of linear equations or Lagrange interpolation.

2.3 Replication or Erasure Coding?

At this point the question what the advantages and drawbacks of erasure coding (in comparison to whole-file replication) are arises. This question has been studied in several works, e.g. [20, 14, 8], and the main results are presented in the following.

2.3.1 Advantages

There are mainly three advantages that are presented in the mentioned literature, namely the benefit of chunking, storage savings and increased availability.

Chunking As already mentioned in the section about replication splitting a file into smaller blocks has the advantage that a parallel retrieval and a better load balancing as well is possible. In principle this also holds if those blocks are erasure coded.

Bandwidth Another advantage that is mentioned in [20] is that a system that employs erasure coding needs much less bandwidth in comparison to a replicating system while having a similar mean time to failure.

Availability The probably most important benefit of erasure coding regards two properties influenced by a redundancy mechanism: file availability and storage overhead S (where $S = \#\text{replicas}$ and $S = \frac{n}{m}$, respectively). Basically it holds that if replication is utilized a file is available if at least one out of the S replicas is available. If erasure coding is used instead, at least m out of the $S * m$ blocks must be accessible. Under the assumption that all nodes have the same availability (or average node availability is used, respectively) and their failures are independent and identically distributed it was figured out in [14, 20] that erasure coding provides higher file availability than replication while using the same storage overhead. Or vice versa erasure coding needs less storage for the same file availability.

For instance, in [14] each file is divided into $m = 7$ fragments and it turned out that to obtain a given availability replication needs up to three times more storage than erasure coding needs (cf. Figure 1). Furthermore, it has been figured out that the smaller the node availability is the larger is that factor. Thus, the conclusion is that erasure coding performs considerably better especially for low node availabilities and up to availabilities of 0.9.

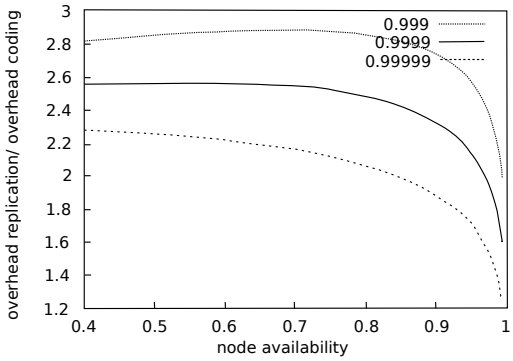


Figure 1. Node availability against ratio of storage overhead of replication and overhead of erasure coding for different levels of file availability (based on [14])

2.3.2 Drawbacks

However, there are also some disadvantages that one has to be aware of and which are described subsequently.

Dependence of Parameters One issue are the strong dependencies between node availability, file availability, stor-

age overhead and the choice of the number of fragments m as the authors of [8] figured out.

For example, in contrary to the advantage mentioned above, for certain combinations of small storage overhead and low node availability whole-file replication obtains better file availabilities. But as the storage overhead might get larger the achievable file availability is higher for erasure coding.

At last it is claimed that the ratio of node availability and storage overhead is most important to make the right decision about the question whether erasure coding is suitable and unless node availabilities cannot be measured precisely enough whole-file replication is recommended.

Complexity One important disadvantage of employing erasure coding is that it leads to a more complex system design because coding and decoding mechanisms have to be implemented. Furthermore, the retrieval and maintenance of files might be more complex as it is necessary to gather m blocks instead of accessing the entire file [14].

Access Costs Another further drawback is mentioned in [21]. Whenever a file should be downloaded m nodes have to be contacted instead of only the node that holds the whole-file replica. Furthermore, this implies that the latency is determined by the m^{th} closest node that stores a fragment instead of the closest node that holds a replica.

2.4 Hybrid Approach

Since erasure coding as well as replication have different advantages and drawbacks, as discussed above, the idea of an hybrid approach, that uses both ideas, has been studied in [21, 22, 7]. The notion is to separate mechanisms for high availability from those for latency reduction, as argued in [20].

Approving to this notion in [21] it is proposed to use a redundancy scheme in which one (or two) primary copies (of the whole original file) are hold as well as erasure coded blocks of the file. That approach should have the advantage that on the one hand the storage efficiency of erasure coding is used. On the other hand the primary copy allows a simpler and faster access to the stored object and a simpler reconstruction of lost blocks by preferring the primary copy instead of gathering and decoding m coded blocks and producing a new one. Figure 2 shows how a file is stored. In this example the file a is split into $m = 4$ blocks, encoded into $n = 6$ blocks and afterwards the whole file is stored at the responsible node and the coded fragments are stored at the nodes in its leaf set. A similar approach where fragments are stored at the successors of the responsible node is proposed by [17].

Simulations based on a Farsite trace with average node availability of about 81% have shown that for a file availability of 0.9999 the storage overhead and maintenance costs are lower for the hybrid approach than for replication [21]. Figure 3 shows some simulation results for a Farsite trace. Maintenance costs of 0.025 means that on average 2.5% of "the user-stored data [is moved] in order to maintain the current state" [21]. This can be explained because if the primary copy does not fail then to reconstruct a failed coded block a replacement block can be generated with low cost of $\frac{1}{m}$. Considering an Overnet trace with average node availability of about 15% it also turned out that the hybrid approach has lowest maintenance costs and storage overhead [21].

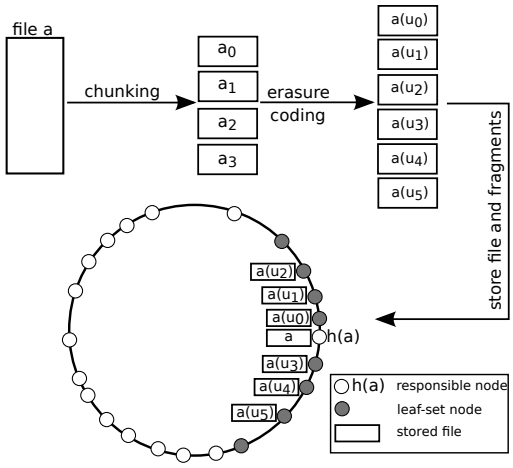


Figure 2. Example for hybrid approach

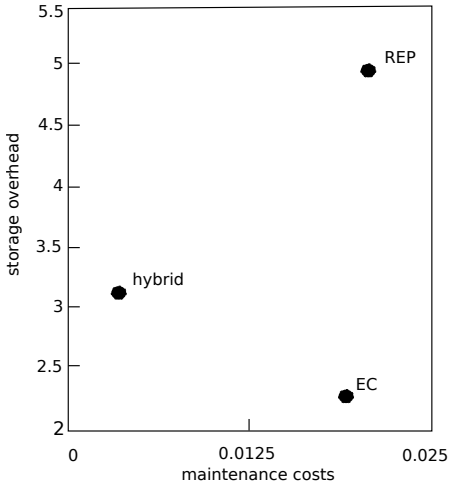


Figure 3. Maintenance costs against storage overhead for 0.9999 file availability for Farsite trace (based on [21])

Together with further theoretical analysis it is finally recommended that replication should be used for highly avail-

able nodes and the hybrid approach is preferable if node availabilities are low like e.g. in the Farsite or Overnet trace. To conclude, there is no definite answer to the question whether erasure coding or a hybrid approach should be used instead of replication and the decision must be weighed properly. If the node availabilities are rather low, which might be the case in an OSN, storage overhead and maintenance costs can be decreased and file availability increased using a hybrid approach. Additionally, the higher access costs of erasure coding are improved by storing one whole-file replica. Thus, if one is aware of and can cope with the additional complexity of the storage system due to coding techniques, retrieval of files by gathering enough fragments or maintenance of fragmented files a hybrid approach might be suitable.

3 Heterogeneity Support

Originally PAST does not support heterogeneity of nodes which results in the fact that nodes are basically responsible for a certain ID space independent of their storage and bandwidth capacities. However, nodes can be highly heterogeneous, for example, if one take into consideration that servers or strong computers but mobile devices as well may participate in an OSN.

3.1 PAST

Although there is no explicit mechanism to support heterogeneity of nodes PAST provides some support based on load balancing. In [16] it is argued that among other things because of different storage capacities of nodes in the network a situation can arise where the k closest nodes that should store replicas are not able to handle the request due to lack of free storage. However, there are k nodes in the corresponding k leaf sets that are able to store the replicas. At this point a mechanism called *replica diversion* is employed. Whenever a node A that is one of the k closest to the fileID that should be inserted receives a storing request and is not able to handle the request itself, it chooses a node B that has maximal free storage space, is within A's leaf set, is not one of the k closest and does not hold a diverted replica yet. Then, A sends a request for storing a replica to B and in turn stores a pointer to the diverted replica. More precisely, a replica is diverted if $\frac{filesize}{freespace(A)} > t$. That is, if the file "... would consume more than a given fraction t of ... [A's] remaining free space" [16]. Thus, this load balancing also implies a kind of implicit heterogeneity support since nodes with little storage space more likely divert replica to stronger nodes.

3.2 Virtual Servers

A further possibility to take heterogeneity of nodes into account are *virtual servers* [1]. Using this mechanism each physical node are assigned several nodeIDs and thus, each physical node acts as several (virtual) nodes and is responsible for several ranges in the DHT.

3.2.1 Virtual Servers in CFS

This notion is employed in CFS [1] to allow heterogeneity support. For that purpose the number of virtual servers per physical node is roughly proportional to its capacity and in principle the stronger a node is the larger the range a physical node is responsible for in the DHT is. Furthermore, this number can be adjusted from time to time. A node that is overloaded just shuts down virtual servers and nodes that have enough capacity and are able to handle more load just add new virtual servers. All maintenance is automatically done by the underlying mechanisms responsible for joining and leaving of nodes.

To overcome the drawback that the number of hops may increase when using this approach it is proposed that virtual servers on one physical node may use the routing tables of each other. However, the drawback that more storage space is used for routing tables in each physical node since each virtual server has to maintain one remains. Especially, if the overall system utilization is low, many nodes will create virtual servers and thus, increase the storage space needed for routing tables.

3.2.2 Rendezvous Mechanism

In [11] another complete distributed algorithm has been proposed. A main advantage is that this algorithm addresses a problem that could occur if the CFS mechanism is used: if a node shuts down a virtual server and a neighbour becomes responsible for the corresponding files the neighbour also can get overloaded and the process can further continue in this way.

To avoid this issue, the notion of *light* and *heavy* nodes is introduced. A node i has a certain capacity c_i and a certain load l_i which is simply the sum of the objects that are stored by node i. Thus, the utilization of node i is determined by $\frac{l_i}{c_i}$ and a node is called heavy if $\frac{l_i}{c_i} > 1$ and light otherwise. A light node A periodically selects another node B that participates in the DHT. If B is a heavy node, it will choose its lightest virtual server that dissolves its overload and transfers it to node A as long as this will not make A a heavy node. This approach guarantees that no node gets overloaded because of the transfer of a virtual server and thus, the issue of the mechanisms employed in CFS is avoided.

Additionally, this rendezvous mechanism can be extended by a further operation. If in a rendezvous there is no virtual server with the properties described above, node B can transfer the heaviest virtual server that does not leads to an overload of A to A and thus, increase the probability that B will find another light node that allows to make B a light node.

To get an impression how well this mechanism performs, the authors of [11] figured out, that about 15% to 45% of the entire load must be moved for system utilization between 50% and approximately 97% in order to make all nodes light (assuming 5 virtual servers per node).

3.2.3 Redistribution of Load using Directories

Another approach, which has been proposed in [4], especially improves the scalability in dynamic and heterogeneous systems where heterogeneity refers either to size or popularity of an object and either to storage space or bandwidth of nodes accordingly. However, since files in an OSN are rather small heterogeneity in popularity of objects and heterogeneity of nodes' bandwidth might be more important.

The basic concept is that of *directories*. A directory collects information about the load of several nodes and finally computes a reassignment of virtual servers in order to achieve a better load distribution. Assume that there are $d \ll N$ directories. Then, a known hash function h is used and a directory $i \in [0, d)$ is stored at the node that is responsible for $h(i)$. Intuitively, this approach should perform better than the rendezvous mechanism because more information is taken into consideration when transfers are computed. Interestingly, it even has pointed out that this algorithm barely performs worse if the number of directories increases although an increasing number of directories leads to more distributed decisions each based on less information.

Periodic Balancing The directories are used as follows. Each node periodically (each t seconds) selects a directory at random and sends information about its capacity c_i and the loads of its virtual servers to it. A directory collects these information and after a certain time t it computes a new assignment of virtual servers to nodes in order to improve the load distribution. This is done by a greedy algorithm that works as follows. For each node n that reported load information to the directory its least loaded virtual server is moved into a set $pool$ until the ratio of n's load and its capacity is below a certain value k (thus, removing those virtual servers leads to a utilization of at most k). Afterwards the virtual servers in $pool$ are sorted in a decreasing order and each $v \in pool$ is assigned to the node m that minimizes $\frac{l_m + l_v}{c_m}$ (notice that nodes that have a low utilization are preferred). Finally, the computed reassignment is reported to the nodes, the nodes transfer the corresponding files (if the new node that should host the virtual server does not get overloaded) and the directory restarts its collection process. Depending on the choice of k and t one gets a tradeoff between load movement and quality of load distribution; it is proposed to set k depending on the average utilization of the reporting nodes. Thus, a virtual server is only transferred if the node's utilization is larger than the average utilization of those nodes that sent information to the directory.

Emergency Balancing In spite of the periodic balancing this approach is used when a node gets overloaded, i.e. its utilization becomes larger than a certain value k'. In this case it directly reports its information to a random directory. The directory in turn detects the situation and instantly computes a reassignment using the greedy algorithm described for periodic balancing.

4 Storage Quotas

Quota management is a mechanism that determines how much storage might be occupied by a node. The quota of a node could for example be based on the storage space a node provides. Thus, it addresses mainly two issues in a distributed storage system. On the one hand it gives an incentive to provide storage space and therefore to actively participate in the distributed storage system. On the other hand it helps to overcome the security issue that (malicious) nodes exhaust the storage space in a crucial and selfish way. Since original PAST only supports storage quota by using smart cards, which is not reasonable for an OSN, another completely distributed approach is needed.

4.1 Quota Managers

One possible solution one can think of is the use of quota managers [19]. In PAST [2] the smart card stores information about how much storage may be used and how much is contributed by a node. This information is maintained during the insertion and reclaim of files and it is assured that the quota is kept. Now, the basic idea is as follows. Instead of using smart cards the neighbours of a node emulate them, store quota information and pay attention that the quota is kept. A node A that receives an insert request from node B simply asks the neighbours of node B if B is allowed to use certain storage space and then acts like the majority.

Nevertheless, the main problem that arises is that nodes have no intrinsic reasons to do so and, instead, it would be simpler not to keep care and allow arbitrary insertion of files. Thus, another approach that actually gives an incentive to participate in a quota management system is required.

4.2 Audit Architecture

The basic idea of auditing [9] is that each node in the P2P system stores information about its usage of storage as well as its contribution. This information can be accessed by other nodes to verify - or *audit* - these information and by that prevent a node from using without providing storage.

4.2.1 Usage File

As mentioned above each node stores information about certain storage. The idea is that a node can barter local storage for remote storage which is necessary to insert data into the DHT. For that purpose each node maintains a *usage file*. This usage file includes the *advertised capacity* as well as two logs: the *local list* stores (nodeID, fileID, size) of all files that are stored locally and the *remote list* stores (fileID, size) of all own files that are stored elsewhere. Now, whenever a node wants to store a file it can be verified whether the used storage space does not exceed the advertised capacity and thus, the node is allowed to store the file.

Furthermore, it is also possible that a node wants to use more storage space than it is allowed according to its advertised capacity. As long as there are also nodes that are willing to contribute more storage space than they want to use there is a simple way how they can trade storage with each other. For that purpose a node A that wants to "sell" an amount of $value$ storage space to a node B just stores an entry $(B, value$ trade$)$ in its remote list and B stores an entry $(A, value$ trade$)$ in its local list. In that case when a node wants to insert a file it might be necessary to verify whether the used storage space does not exceed the sum of the filesizes mentioned in its local list (because checking advertised capacity against used storage space is not sufficient).

4.2.2 Auditing

However, at this point it is not ensured that a node does not fake its usage file and there are mainly two possible ways to do so in order to take advantage.

1. A node can reduce the remote list by omitting entries of actually stored files in order to be able to use more storage than it is allowed.
2. A node can extend the local list by adding files that are actually not stored in order to contribute less storage than it specified.

Normal Auditing The first issue can be overcome by a mechanism called *normal auditing*. On the one hand a node that stores a file is interested in auditing the remote list of the file owner and will delete the corresponding file if it is not mentioned there. On the other hand a file owner is interested in auditing the storing node and will delete the entry of the corresponding file in its own remote list if it is actually not stored. This auditing can be done by requesting a random piece of the file and verifying whether the storing node can respond. The normal auditing can be performed at random points in time.

Nonetheless, there is still an issue regarding normal auditing. Suppose that a node A stores a file at node B and a file at node C. If A knows whether it is audited by either B or C it can fake its remote list simply by removing the entry of B when audited by C or the entry of C when audited by B. Thus, it is necessary that audits are performed anonymously. It is proposed that this can be done in a similar way to routing in crowds [12]. Therefore, the auditing node chooses a node at random and sends the message to the chosen node. This node in turn either forwards the message with a certain probability to a further intermediate node or routes the request to the actual destination. Eventually the request arrives at the node that should be audited without revealing which node actually wants to audit it. Thus, it is prevented that nodes use storage for which they do not "pay".

Random Auditing The second issue can be tackled by *random auditing*. The goal is to verify that a node really stores the files that are mentioned in its local list. For that purpose each node periodically picks another one at random, requests its usage file and then checks whether the mentioned files in the local list are also mentioned in the corresponding remote lists of the file owners. Assume that usage files are signed with a private key, then, if a cheat is discovered there will be a proof of the node's misbehaviour and it can be punished.

A problem that remains is how a misbehaving node can be excluded from the system. But this problem is rather a general security issue than a specific problem of quota management. Unfortunately, it is an open question how that can be achieved effectively.

4.2.3 Checking Replicas

The last point regards replicas. Normal audits are only performed by the file owner on the node that stores the (original) file. If files are replicated other nodes will store the file (which is not mentioned in any local list and thus random auditing does not cover this file) and so provide storage, too. Therefore, another mechanism, called *challenge* is required. Each node that stores a file chooses a replica holder periodically at random and requests a hash of (some blocks of) the file. To ensure that this node itself does not request the hash, the verifying node should inform the other replica holders in advance.

4.2.4 Performance Results

In [9] several simulations have been performed. The main results state that the required bandwidth per node is approximately constant in the number of nodes. Similar results has been obtained for the bandwidth depending on average node lifetime but it slightly increases for lower lifetimes. Because they only simulated lifetimes of ≥ 1 day the ques-

tion remains how auditing performs for shorter values. Furthermore, it turned out that the required bandwidth grows linearly depending on the number of files per node and it is argued that this might be acceptable because the overhead is quite small, especially as files are usually large. Because this possibly does not hold for an OSN one has to try whether it performs well.

5 Query Support

Due to the fact that PAST is based on top of a lookup service it is only possible to retrieve documents by knowing their fileID. Nonetheless, it could be necessary and it is desirable to be able to search for certain documents using a kind of keyword search for example to search for people from a certain area or with a certain age. This section elaborates on the question how such a mechanism could be designed on top of PAST.

5.1 Requirements

The basic approach that is utilized by most of the systems is to use meta-data that is associated to a file and describes its content. Different systems mainly differ in the way meta-data is structured, the operators that can be used in queries and how queries are evaluated [15].

The first question regarding query support that arises is how queries in an OSN are assumed to look like and how much (or which) information should be considered when answering a query. Answers to these questions influence the choice of an appropriate approach. Concerning the first question consider assumed queries in an OSN. It is rather searched for documents with certain properties like e.g. people from a certain city, people with a certain age, events taking place at a certain date etc. than for freely chosen (unstructured) search phrases like one usually does when using search engines such as Google. Strongly related to this is the second question. Usually a full-text search is not required but rather a search for documents with several properties as mentioned above.

In [15] a survey on different approaches of querying systems is given. Mainly three kinds are distinguished: systems that are based on a semi-structured model, systems that utilize a relational model and systems that use meta-data consisting of tuples (attribute, value). Considering the different mechanisms it has turned out that an attribute-based system might be most suitable while others are less because for example they are designed for broad queries, offer too much services, for the purposes mentioned above, and thus add unnecessary complexity (e.g. PIER that supports 40 operators similar to those in a relational database) or make assumptions that are not met by Pastry or PAST (e.g. MAAN assumes a special hash function, MLP a special structure of nodes, or KadoP that a file is stored at the node that owns the file) (cf. [15]). The remainder of this section focuses on a system called PinS [18] which has the advantage that it fits the requirements mentioned above and can be implemented on top of Pastry and PAST.

5.2 Attribute-based Model

The idea of an *attribute-based model* is that a file is described by meta-data consisting of a set of (attribute, value)-tuples. For example a profile in an OSN can be described by tuples like $\{(name = \text{John Average}), (yearofbirth = 1986), (location = \text{Paderborn}), (gender = \text{male}), \ldots\}$. After defining such meta-data for a certain file it is necessary to store this data and somehow make use of it in order to provide query support. How this can be done is described in the subsequent two sections.

5.2.1 Storing Meta-Data

The basic idea is to store meta-data in the DHT, like any usual data, by using the storage service layer. Whenever a file should be inserted into the DHT it is possible to store also meta-data that describes the file in order to locate that data afterwards without knowing its fileID. For that each meta-data tuple is hashed and then inserted into the DHT and therefore, stored at the responsible node together with the fileID of the corresponding file it describes. This stored data is then referred to as *location data*. For example, if John Average's profile should be stored, the corresponding document is inserted as usual by obtaining its fileID, e.g. $fileID = 1121$. Furthermore, each meta-data tuple is hashed individually and stored at the responsible node together with the fileID. If the tuple $(location = \text{Paderborn})$ is hashed to 2312 then the node responsible for $fileID = 2312$ stores $((location = \text{Paderborn}), 1121)$ (and possibly other fileIDs of profiles of people that are also from Paderborn). So, an inverted index with entries $< attributeName = value, \{fileID1, fileID2, \ldots\} >$ that map (attribute, value)-tuples to a set of files that contain this tuple is created and distributed in the DHT.

5.2.2 Fundamental Query Support

Now, if a node wants to search for certain documents it first has to define its query by (attribute, value)-tuples. Assuming that such a query consists of conjunctions and/ or disjunctions of terms of the form $(attribute = value)$, there are different solutions how a query evaluation can be done in detail. The first solution [18] assumes that any node can get a query. The receiving node computes the hash of each individual term and contacts the responsible nodes. These nodes reply with the fileIDs of files that contain the searched combination of attribute and value by using their location data. Afterwards the conjunction and/ or disjunction is computed and the resulting list of fileIDs is sent to the query initiator. One advantage of this approach is that in principle it might be possible to send

queries to particular nodes, e.g. strong nodes that have high bandwidth. Furthermore, if, for example, the query is $(location = \text{Paderborn}) \wedge (gender = \text{male})$, then one can lookup the fileIDs for $(location = \text{Paderborn})$ and for $(gender = \text{male})$ in parallel and then compute the intersection afterwards.

[13] proposes a slightly different approach. Based on a trace of (keyword-) search requests it is argued that most searches consist of a combination of 2-5 keywords which motivates a design that improves performance of conjunctive searches. Assuming that the observation also holds for an attribute-based model, the proposed approaches should be taken into consideration. The basic idea is that, unlike the approach described above, different lookups are not done in parallel and the conjunction is computed afterwards, but the conjunction is computed step by step. That means if, for example, there is a set of fileIDs for $(location = \text{Paderborn})$ and a set for $(gender = \text{male})$ then the first set is computed and sent together with the second term to the node responsible for the second term. This node can then directly answer with the intersection. The advantage is that less data is transferred and network bandwidth is saved, especially if the intersection is much smaller which might be the case for the example.

Bloom Filters The latter approach has furthermore the advantage that one can use *bloom filters* to decrease the used bandwidth when computing the result of a conjunctive query [13]. A bloom filter [3] is in a sense a compact representation of a set such that it is possible to determine whether a certain element belongs to this set. However, this membership test can give false positives, a positive result although the element does not belong to the set.

The basic idea is as follows [3]. At first k independent hash functions $h_1, \ldots, h_k$ are chosen. They are independent and each maps to the set $\{1, \ldots, m\}$. Furthermore a bitvector of length m bits is created. For each element a in the given set that should be represented as a bloom filter the positions $h_1(a), \ldots h_k(a)$ are set to one. Now, if a membership test for an element b should be applied again $h_1(b), \ldots, h_k(b)$ are computed. If at least one of this positions in the bitvector is 0 then b is not an element of it. Otherwise it can be an element but there is also the possibility of a false positive. For example if four hash functions are used and $m = 6n$, so the number of bits is six times the number of elements in the set under consideration the probability of a false positive is only 0.0561 and in general there is obviously a tradeoff between the size m of the bloom filter and the probability of false positives.

This approach can be used to save communication bandwidth for conjunctive queries [13]. In the example mentioned above where the intersection of $(location = \text{Paderborn}) \wedge (gender = \text{male})$ should be computed, the first node only sends a bloom filter of the set of fileIDs that contain $(location = \text{Paderborn})$. The second node then computes the intersection with the set that is obtained for $(gender = \text{male})$. This intersection can then be either sent back to the querying node or again back to the first node to remove the false positives and then to the querying node.

5.2.3 Inequality- and Like-Terms

Until now it is only possible to handle queries that consist of conjunctions and disjunctions of equal-terms, i.e. of the form $(attribute = value)$. Because it could be also useful to support queries that contain comparisons like $\leq, \geq$ or $like$ an extension is proposed in [18]. The basic element for that are so called *local catalogs*. Each node has a local catalog that describes the files that are locally stored. Thus, for each element and each term that describes this element "... the term identifier, the attribute name, the attribute value and the list of local objects satisfying the term ..." [18] are stored.

The following strategy allows to evaluate terms that consist of at least one equal-term together with a comparison- or like-term connected by a conjunction and possibly arbitrary further terms. At first a *query coordinator* is chosen. This coordinator decomposes the query and sends the corresponding parts of the query to other nodes, i.e. to nodes that further decompose the query (sub-coordinators) or those that store location data of the query as described above. In the next step the query is evaluated. At first each node that was contacted in the first step checks whether it has some matching objects in its own local catalog. Notice that using the local catalog it is possible to evaluate inequality- and like-terms since meta-data is stored as plain text. Then, the local location data is used to determine further nodes that store (possibly) matching files and the query is forwarded to them. Notice that at this point it is usually not sufficient to lookup the fileIDs for certain (attribute, value)-tuples in the local location data since inequality- or like-terms have to be evaluated at the nodes that actually store the data. Therefore, the nodes that are responsible for the fileIDs in turn compute an answer using their local catalogs and finally send it, for example, back to the query coordinator that computes the answer and replies to the query initiator.

To simplify the understanding a short example (based on [18]) is considered in the following. Assume that a query $(location = \text{Paderborn}) \wedge (yearofbirth \geq 1980)$ is issued. At first the node that stores the location data for $(location = \text{Paderborn})$ (the node that is responsible for the fileID obtained by hashing the tuple) is contacted. This node checks its local catalog to get the fileIDs of files that satisfy the query and are stored locally. This could give some matching files. In the next step the node

uses its location data to get the fileIDs of files that satisfy $(location = \text{Paderborn})$. The entire query is sent to the corresponding nodes, which in turn use their local catalogs to determine matching files and finally send their answers to the first node.

In addition to the support of inequality- and like-terms this design also allows further possibilities. For instance, the query coordinator can be chosen according to its capabilities. The decomposition is flexible and the number of terms a node gets responsible for in principle adjustable. Furthermore, it is stated that timeouts can be specified and considered by decreasing the initial value by a certain constant each time the query is forwarded.

6 Conclusion

This paper elaborates on the question how the quality of data storage based on PAST for an OSN could be improved. For that purpose several different aspects have been considered and enhancing approaches have been presented.

Located in the DSS layer the question concerning the usage of whole-file replication or erasure coding has been considered in section 2 leading to the conclusion that it might be reasonable to use erasure coding as long as one is aware of the additional complexity.

Section 3 motivates the need of a heterogeneity support and presents a simple approach that can be implemented on top of a DHT without changing the underlying design.It provides load redistribution based on virtual servers that takes heterogeneity of nodes into account.

In order to give nodes an incentive to actively participate in the storage system and thus, contribute storage, a design of distributed auditing has been presented in section 4. Although the question remains how to exclude malicious nodes from the system after they have been detected the design might achieve an incentive for well behaving nodes.

Finally, section 5 elaborates on a solution that enables a user to search for certain documents. Since in a DHT it is only possible to retrieve documents for which the fileID is known the presented attribute-based approach might be a reasonable feature on top of the DSS layer.

To conclude, this paper presents several possible approaches that focus on the fact that they respect the design of Pastry and PAST. Actually the presented solutions might be able to improve the quality of the data storage and might be suitable for being implemented on top of PAST.

References

[1] F. Dabek, F. M. Kaashoek, D. Karger, R. Morris, and Stoica. Wide-area cooperative storage with CFS. In *Proceedings of ACM symposium on Operating systems principles (SOSP)*. ACM, 2001.

[2] P. Druschel and A. I. T. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceeding of Workshop on Hot Topics in Operating Systems*. IEEE Computer Society, 2001.

[3] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. *IEEE/ACM Transactions on Networking*, 8, 2000.

[4] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load Balancing in Dynamic Structured P2P Systems. In *Conference of the IEEE Computer and Communications Societies (INFOCOM)*. IEEE Computer Society, 2004.

[5] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. H. Campbell. A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC)*. IEEE Computer Society, 2005.

[6] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod. Performance Evaluation of Replication Strategies in DHTs under Churn. In *Proceedings of International Conference on Mobile and Ubiquitous Multimedia (MUM)*. ACM, 2007.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, S. E. Czerwinski, P. R. Eaton, D. Geels, R. Gummadi, S. C. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Y. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of International Conference on Architectual Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2000.

[8] W. K. Lin, D. M. Chiu, and Y. B. Lee. Erasure code replication revisited. In *Proceedings of International Conference on Peer-to-Peer Computing (P2P)*. IEEE Computer Society, 2004.

[9] T.-W. Ngan, D. S. Wallach, and P. Druschel. Enforcing Fair Sharing of Peer-to-Peer Resources. In *Revised Papers from International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2003.

[10] J. S. Plank. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems. *Software Practice and Experience*, 27, 1997.

[11] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *Revised Selected Papers from International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2003.

[12] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1, 1998.

[13] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Proceedings of International Middleware Conference*. Springer, 2003.

[14] R. Rodrigues and B. Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *Revised Selected Papers from International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2005.

[15] C. Roncancio, M.-D.-P. Villamil, C. Labbé, and P. Serrano-Alvarado. Data Sharing in DHT Based P2P Systems. *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, 1, 2009.

[16] A. I. T. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *Proceedings of ACM symposium on Operating systems principles (SOSP)*. ACM, 2001.

[17] R. Sonar and D. M. Thakore. Data Management for a Distributed Hash Table. In *International Conference on*

Distributed Computing and Networking (ICDCN). Springer, 2006.

[18] M.-D.-P. Villamil, C. Roncancio, and C. Labbé. PinS: Peer-to-Peer Interrogation and Indexing System. In *Proceedings of International Database Engeneering and Applications Symposium (IDEAS)*. IEEE Computer Society, 2004.

[19] D. S. Wallach. A Survey of Peer-to-Peer Security Issues. In *Revised Papers from International Symposium on Software Security (ISSS)*. Springer, 2002.

[20] H. Weatherspoon and J. Kubiatowicz. Erasure Coding Vs. Replication: A Quantitative Comparison. In *Revised Papers from International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2002.

[21] C. Williams, P. Huibonhoa, J. Holliday, A. Hospodor, and T. J. E. Schwarz. Redundancy Management for P2P Storage. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE Computer Society, 2007.

[22] F. Wu, T. Qiu, Y. Chen, and G. Chen. Redundancy Schemes for High Availability in DHTs. In *Proceedings of International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. Springer, 2005.

[23] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.

Rights and liability in operating a p2p based social network

Kangping Liu
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: kangping@mail.upb.de

Abstract

P2P technology drives internet to a new age, it enable users to get free communication, free file sharing etc. Due to the characteristics of P2P network, which are uncontrollable and decentralized, right and liability issues come out. Different entities are involved in, such as developers, end-users, internet service providers etc. Present that the future of peer-to-peer technologies are unpredictable, many legal provisions and constraints are still in discussion now, This paper discusses the right and liability issues in P2P based social network. It mainly introduces and discusses copyright, privacy violation and covers some child-inappropriate content issue in p2p network.

Key words: peer-to-peer, copyright, privacy, social network

1 Introduction

According to the report from German internet research institution ipoque [5], peer-to-peer has thoroughly dominated the internet by showing that 43 percent to 70 percent traffic is generated by peer-to-peer file sharing. Although according to the report, p2p is now growing slower than past years, it is still the dominate one. Among all the P2P protocols, BitTorrent is the number one. P2P enables people to not passively wait for the resources provided in central service, but actively share resources with each other by private channel without restriction like registration, payment etc. Also with this enabler user not only can share resources, but also private voice communication, like Skype, or video streaming, like IPTV and so on.

At the meantime, another new emerging force also wins public's favorite quickly, which is known as online social network, and now plays a essential role in people's daily life. Examples for online social network as Facebook for personal information, Twitter for short message and Youtube for video sharing. Such applications form an incredibly fascinating virtual community that people can share their daily life, find friends etc. While online social network brings a huge convenient to people's daily life, the providers who support these applications suffer from a huge burden on their servers, due to the rapidly growing number of client, these servers are facing a challenge of over capacity. Not only the burden of servers is an issue, the privacy problem in social network is also worried by user. For these client/server applications, clients have to rely on the server to keep their data and this leads to many questions, such as can these servers be trusted, or what if I want remove my account thoroughly? Because it is known that in Facebook an account can only be deactivated but not deleted.

The combination of the fascinating and multifunctional online social network, and the p2p technology which enables user wholly control their own data without any auxiliary, seems to be very promising. "LifeSocial" is a completely distributed social network built on structured p2p overlay (FreePastry). User can create personal profile; find friends, post pictures and some other basic social network functionalities. And LifeSocial is more than this, it allows user to create their own social network, the size can be from a network for several friends to global social network. Because of its p2p-based nature, the cost for operating a global network will be no more than operating a small group. From another perspective, LifeSocial is a plug-in-based application, which means that its functionalities can be indefinitely extended; more features will be added in the future.

2 Overview on issues in p2p-based social network

Right and liability issues are always a debatable topic in P2P world. The initialization of the issue is distribution of questionable content. Because of the nature of P2P that no central server and central censorship, after connecting to the overlay, every peer is able to publish whatever it wants without inspection. Once data has been distributed over the internet, data subjects lose any possibility to further control the circulation of the innumerable copies of them that are created on the web.[7]

The undesirable content distributed in p2p-based social network comprises three parts. (i)Copyright issue must be the one stands in the breach. It is mainly caused by the freedom provided by p2p technique. Nowadays nearly all the resources can be kept in digital storage, even the copyrighted resources, thus every end-user could commit a direct infringement once he has published, distributed or reproduced copyrighted content without authorized by the copyright owner. In addition, p2p tool maker may also be involved in as a contributory, vicarious or inducement infringer if they make their p2p tool not working as a neutral role and helping user to infringe copyright. This issue is discussed in section [3].(ii)Privacy is a main issue in online social network, with p2p technique, this issue becomes even more serious. Since all the information published to public in the p2p network has not been checked or censored in advance. So it is easy for a third party to publish someone else's privacy freely. In this violation procedure, the service provider could be also judged liable of privacy violation under some circumstances which is introduced in section [4]. (iii)Illegal content is also an important issue in p2p network. Vicious end-user could spreads plagiarized or (child-) pornographic profile pictures through independent channel. Since usually the only victim of such content is children and their parents and they usually can't do anything but keep away from p2p network. What can a p2p network do for them is talked in section [5]. Copyright issue is discussed in U.S. copyright law and privacy violation is discussed in E-commerce directive. The evaluation of all these issues and guidelines for developers are introduced in section [6]. Then the general penalty of copyright infringement and privacy violation are introduced in section [7].

2.1 Roles in issues

Each issue refers to several roles and each role holds various points of view towards the issue. Of course they all consider themselves mostly, such as in copyright issue, end-users may simply argue that p2p tool which is able to freely distribute content is not developed by them, why they should be accounted for the content? Developers also may argue that they just provide the platform and they have no idea about the content published by end-user. This section introduces the basic functionalities and behaviours of each role. Basically, the roles will be introduced in this paper are following:

- Service provider
- Copyright owner and privacy victim
- End-user

Service provider The definition contained in Article 1(2) of Directive 98/34/EC as amended by Directive 98/48/EC, considers a service provider as any natural or legal person providing "any service normally provided for remuneration, at a distance, by electronic means and at the individual request of a recipient of services" and recipient of services as "any natural or legal person who, for professional ends or otherwise, uses an information society service, in particular for the purposes of seeking information or making it accessible".[7] In U.S. copyright law, a service provider is defined as a provider of online services or network access, or the operator of facilities. Once a wide scope of unlawful actions are performed in p2p network, it is nearly impossible to investigate users one by one, but it is easy to control the service provider. Hence service provider has to always carefully conduct its software and system which brings a huge burden. However many constraints and safe harbors are provided to support service provider.

Copyright owner and privacy victim Copyright owner holds an exclusive rights to the original work, usually for a limited time. It has the right to reproduce or distribute the work in copies or phonorecords, perform or display the work publicly and so on.[1] Copyright owners usually dislike p2p network, since it gives consumers opportunity to get their copyrighted work for free(e.g. Napster), so they persistently put pressure on court to fight against infringement in p2p network. Privacy victim is someone whose privacy has been violated by a third party in the p2p network. Copyright owner and privacy victim have rights to notice the operator of p2p network to remove the inappropriate content if it has truly harmed himself.

End-user End-user is the user of p2p tool. They can be the victim of child-inappropriate content, but usually they are the initiator of most issues. In copyright issue end-user is regarded as direct infringer. In privacy issue end-user publishes others' personal information without authority; in illegal content issue end-user spreads pornographic pictures or terrorist message. Myriad behaviors can be performed by end-users and end-users are uncontrollable in a p2p network. However, if the copyright infringement or privacy violation is performed by a large amount of end-user, it is almost impossible to investigate every one of them. The general solution is to handle the p2p tool used by end-user. For example, in Napster case, after the serious damage to music industry made by a large amount end-users of Napster, the court mainly put concentrate on Napster to verify its secondary liability, although the end-users are the direct infringer.

3 Copyright

In the world of p2p network, there are two things viewed as usual, one is digital file and the other is transmission be-

tween peer and peer. But rules getting stricter when copyright law are imposed on this world. First, every digital file should qualify as copyrighted work. Second, every unauthorized transmission which results in a reproduction, distribution and possibly a public performance (in the world of copyright law, "public performance" includes the act of transmitting a copyrighted work to the public) requires an explanation.[8] In a p2p based social network, which is not a file sharing network, the dedicated unauthorized transmissions seem not to be an issue. But there still can be copyright issues as it is a social network. Users may publish photos scanned from copyrighted works to their public album (which is often seen in a Chinese social network called "RenRen"). Under this circumstance, as explained above, it results in an unauthorized public performance. The infringement defined in U.S. and Europe legislation are almost same except that in EU legislation, it is allowed to copy copyrighted work for non-commercial private use.

3.1 U.S. legsilation

According to the U.S. copyright law principles, infringement is first divided into direct and secondary infringement, and then it clearly classifies the infringement into the following four parts:

- Direct infringement(End-user)
- Secondary infringement: Contributory infringement(Service provider)
- Secondary infringement: Vicarious infringement(Service provider)
- Secondary infringement: Inducement liability(Service provider)

Direct infringement This kind of infringement is mostly caused by end-users. As the name "direct" implies, enderusers reproduce, distribute files by themselves without authorization of copyright owners which directly harms the copyrighted works. These end-users are called as "direct infringer".

Direct infringer has a great many mass base. Usually people are willing to not only download for free, but also upload for free. In today's music industry, according to the digital music report 2008[6] from IFPI (International Federation of the Phonographic Industry) , Music sales via online and mobile channels have risen from zero to an estimated 2.9 billion dollars (15 per cent of industry sales) over the last five years, making music more digitally advanced than any entertainment sector except games. Tens of billions of illegal files were swapped in 2007. The ratio of unlicensed tracks downloaded to legal tracks sold is about 20 to 1. The countermeasures to direct infringement maybe constraint end-users to only share authorized files.

Contributory infringement Contributory infringement arises where a third party with knowledge of the infringing activity, induces, causes or materially contributes to the infringing conduct of another.[4] The premise of contributory infringement is there at least has been a direct infringement performed by end-user. In this kind infringement infringer may not gain any benefit from the infringement, but it somehow assists direct infringers to achieve the infringement. Examples are: providing access to an authorized work copies, such as links on websites or helping code crack.

Vicarious infringement Vicarious infringement means that a person is liable if he has the right and responsibility to supervise the direct infringer and also has a direct financial interest in his activities.[8] The conclusions can be drawn from the definition are first, same as contributory infringement, there is at least a existing direct infringement as precondition. Second, there are another two conditions should be hold, (i) The vicarious infringer has the right and ability to supervise the direct infringer. It means that, for example, the developer of "LifeSocial" has the ability to terminate or block users' account while there are on-going direct infringement. (ii)The vicarious infringer also gain a direct financial interest from the infringing activity. Here the "financial interest" refers to a wide range of definition that any kind of benefit can be considered in, such as attracting more users, making advertisements etc.

Inducement infringement It is a new rule originated from Grokster's case which will be discussed in section [3.2]. It means that a third party who distributes a device with the object of promoting its use to infringe copyright, as shown by clear expression or other affirmative steps to foster infringement, is liable for the users resulting acts of infringement[4].

In U.S. copyright law, chapter 5.12, it provides limitations on liability related to material online which enable a service provider being not liable under certain circumstances. It consists of four main categories[3]:

- Transitory digital network communications: covers conduit systems that transmit and route data between users and store copies only as part of that process.
- System caching: Protects caching systems, which temporarily retain copies for users requesting it from its original source.
- Information residing on systems or networks at direction users: Protects storage systems that allow users to store information on their networks, such as a web hosting service or a chat room.
- Information location tools: Covers information location tools such as search engines or directories.

However when these safe harbors are enacted, p2p is not concerned within scope, p2p service provider is a little bit difficult to perfectly fit into these safe harbors. In a p2p network, user stores files in their own computers and exchange files directly. In this circumstance, service providers can be considered as a conduit in this process and treated under safe harbor 512(a). In 512(a) provider has to implement a policy for terminating repeat infringer and inform subscriber about it after being noticed[3].

3.2 Case study:Copyright

Here are two cases of copyright infringement, one of them straightly judged as liable and presents a standard infringement, but the other's case is more winding and proved to be liable of infringement at last. These two cases show a criterion that how copyright issues are handled in reality.

Napster case Napster is a company established in 1999 in Silicon Valley. It develops software which uses p2p technique to provide user a service of sharing mp3 files. Anyone can log in the Napster network after downloading and installing the software (called "MusicShare") on his/her own computer. All the Napster users can freely share mp3 files with each other by "MusicShare", it means that the client who request for files does not have to pay for the client who provides files, Napster or copyright owners. Moreover, Napster also helps searching and laying out of the target files and listing ip addresses of clients who provide target files.

Since Napster provides a great convenient to music fans to download music for free, it causes hatred and protest of many record companies and copyright owners. Napster was sued by A&M and other 18 record companies in December, 1999, in the name of contributory and vicarious infringements.

The disputed focuses of Napster case are followings :(i) Whether Napster knew there were infringements in its network. (ii) Whether Napster has assisted infringement. The judgment of the court concluded that:(i) Napster already knows there were infringing files in its system. But it did not prevent the infringers log in its system, instead assisted and provided techniques to help users to download infringing files. According to U.S. copyright law, this results in contributory infringement. (ii) Napster did not monitor the files in its system, and it continuously gained benefit from the direct infringement. This results in vicarious infringement. The court draw a conclusion from the provided facts that Napster has the ability to supervise the sharing files of clients, but actually Napster did not provide a valid supervise. And the infringements of copyrighted files bring Napster more and more users, which also can be viewed as a financial benefit. Finally, Napster lost the lawsuit.

Grokster case Grokster and Streamcast are two internet service companies who provide user download service. Grokster had developed software called "Grokster" which helps user to download the software called "Kazaa" (developed by a company called Sharman). Kazaa helps user to achieve file sharing through p2p technology. After installed "Grokster" and "Kazaa", users are able to retrieve music files stored in their computers with each other and upload or download files. But in this procedure, Grokster does not provide any original music files or copies.

In the meantime, Streamcast also developed software called "Morpheus". There is a difference between "Morpheus" and "Grokster", since "Morpheus" uses p2p technology by itself, it allows user to achieve file sharing without any other software. But it still doesn't directly provide a service of original music files or copies upload and download.

Of course, Streamcast and Groskter incurred the hatred of record companies. In October, 2001, Streamcast and Grokster were sued by M-G-M and some other companies.

In M-G-M's view, the defendants help users to download p2p-based software enable end-users to share files, and among these files most are copyrighted work. Downloading and uploading copyrighted work for free result in direct infringement and in this infringement, end-users used the software provided by defendants. Therefore, defendants have to be responsible for contributory infringement. But in April, 2003, the Federal Court rejected the plaintiff's claim. The Federal Court concluded that, although part of end-users of defendants' software is suspected of being involved in direct infringement, actually defendants have no idea about this infringement. One more thing, defendants' software also fulfills "Capable of substantial non-infringing uses".

Because always taking care of unlawful acts of end-users impose a great pressure on p2p service providers. Thus Supreme Court proclaims an exemption that, as long as the software is capable of substantial non-infringing uses, it can survive from contributory infringement, but can't survive from vicarious infringement. Hence in Groskter's case, the court finally concluded that, Streamcast and Groskter didn't contribute to provide convenient or materials for direct infringement, they are not responsible for contributory infringement.

However, M-G-M did not buy this result, it sued Grokster again in the Supreme Court. At last, the Supereme Court proved Grokster to be liable for the copyright infringement by three justifications:(1) Grokster's software is able to help the direct infringement and most end-users involved in direct infringement are also the end-user of Napster. Moreover Based on the documents of the development of Grokster's software, the court considered that the software is developed based on the idea and model of Napster,

and even the name of Grokster comes from Napster. So the main goal of Grokster is for copyright infringement. (2) Grokster and StreamCast did not develop any filtering program to prevent infringements. Even though this evidence has no relation to this case because Grokster and StreamCast are lack of the ability to supervise end-users, it still reveals that these two companies help end-users' direct infringement. (3) The court found that, the more end-users, the more advertisements come to these two companies so that they earn more money. The commercial benefit of Grokster and StreamCast mainly depends on the use of software by end-users, and most of the uses are infringements. All these three justifications can prove Grokster and StreamCast are liable of copyright infringement individually, but the combination of these three proved that Grokster and StreamCast are liable. Based on this case, the Supreme Court also creates a new kind of secondary infringement which is introduced in section [3.1]:induce infringement.

In these two cases, although all of them are proved to be liable, there are still some differences. In Napster's case, even though Napster is not the provider of mp3 files, but from the beginning to the end, it participated in end-user's file sharing activities. End-users are able to share files only if they have logged in Napster, and they can't do anything if Napster's servers shut down. In addition, as an service provider, as mentioned in vicarious infringement, Napster has to supervise its end-users' actions, but it did not, instead it gained benefits from end-users' direct infringement(music sharing). According to U.S. copyright law principal, Napster accounts for vicarious infringement and contributory infringement.

However, in Groskter's case, the defendants Grokster and Streamcast just provide a p2p-based software. After the installation of "Kazaa" or "Morpheus", even if the central server is shut down, end-users still can share their music files with each other. Also compared to Napster, Grokster and Streamcast fulfill "Capable of substantial non-infringement uses". In this case, the aim of Grokster and Streamcast is to provide software that help end-user to share authorized files. Moreover, they left the system as anarchy, so they did not have the ability to monitor end-users' actions. However, if they have provided filtering programs in their network and have not gained any benefit from end-user's infringement, the final result would be much more different.

What can be concluded from these two cases are first, "if you want to monitor your user's you have to be able to control them to behave legally, otherwise just give up control". Second do not participate in end-user's activities, what software provide should be just a platform, then leave the system as anarchy unless the provider is very confident to keep all activities being legal in its network. Last, keep the functions of software in a legal direction and do not ever try to gain any benefit from end-user's infringement, it will definitely incur troubles.

4 Privacy violation

In the era of web 2.0, user-generated content becomes a majority of web content. Internet service providers change into a role behind the scenes, which means, they just provide the facilities for user to create web content (mainly HTML pages) by themselves. With the web-server, software, database provided by service provider, user has the most freedom to create whatever they want on the internet.

LifeSocial, Facebook and Youtube are the very service provider described above. It is surreal to imagine a social network without users. Nearly all the content in a social network is user-generated, such as blog, album, personal profile etc. service provider simply provides a platform for user to create, distribute and share content. Nowadays people actively participate in these social networks to interact with their friends. Of course usually service provider doesn't provide all these support for free, as it gains a large amount of users, it makes profit by adding advertisement, optional paid service etc. In 2000, when the E-commerce Directive was passed[2], web hosting consisted mainly in websites (html pages and related documents) completely developed by the recipient of the hosting service, including the way the content was posted, the structure of the websites and so on[7]

With this high degree of freedom, user can create whatever they want, but content can't be expected to be all legitimate. To avoid privacy violation, at least user should publish content with the consent of the data subject and authorization of the data protection authority, even if the data subject is publisher itself. Without authorization of the data protection authority, it will commit a punishable offence. But this kind of violation is usually just literal, the practical bad case is that the privacy violation concerning third party, such as Alice published Bob's picture without the consent of Bob.

There is a tradeoff between privacy violation and the freedom of speech. It means that, everyone has rights to express its speech, but it may also violate other's privacy. The judgment of this tradeoff depends on concrete case. But this paper concentrates on the liability of service provider in privacy violation concerning third party. It is mainly discussed Europe legislation, because dislike U.S. legislation, privacy gains more attention and is treated as fundamental rights in Europe. Same case can see a very different result between Europe and U.S. legislations.

If users of provider's service violate others privacy, not only the user should be responsible, but the provider could be also liable if the provider is considered responsible for data protection offences committed by using its platform[7]. There is a number of liability exemptions

for service provider described in e-Commerce directive (E-Commerce directive is short for the Electronic Commerce Directive, adopted in 2000, set up a Internal Market framework for electronic commerce, which provides legal certainty for business and consumers alike. It establishes harmonized rules on issues such as the transparency and information requirements for online service providers, commercial communications, electronic contracts and limitations of liability of intermediary service providers[2].).

4.1 EU legislation

The definition provided for by Article 14 of e-Commerce Directive[2] comprises an information society service "that consists of the storage of information provided by a recipient of the service". In that situation, the Directive exempts the service providers from liability regarding the content generated by users under two conditions:(a) the provider has no actual knowledge of the illegal information or (b) it does act expeditiously to remove or to disable access to the information after obtaining such knowledge[7].

According to the recital 42 of the e-Commerce Directive, the exemptions of liability cover only cases where the activity of the service provider "is of a mere technical, automatic and passive nature, which implies that the information society service provider has neither knowledge nor control over the information which is transmitted or stored[7]." In EU legislation, according to 2000/31/EC directive[2], service provider can be ensured not liable under the following three main concept:

- Mere conduit
- Caching
- Hosting

Mere conduit Member States shall ensure that the service provider is not liable for the information transmitted, on condition that the provider:[2]

1. does not initiate the transmission;
2. does not select the receiver of the transmission; and
3. does not select or modify the information contained in the transmission

Caching Member States shall ensure that the service provider is not liable for the information transmitted, on condition that the provider:[2]

1. the provider does not modify the information;
2. the provider complies with conditions on access to the information;
3. the provider complies with rules regarding the updating of the information, specified in a manner widely recognised and used by industry;
4. the provider does not interfere with the lawful use of technology, widely recognised and used by industry, to obtain data on the use of the information; and
5. the provider acts expeditiously to remove or to disable access to the information it has stored upon obtaining actual knowledge of the fact that the information at the initial source of the transmission has been removed from the network, or access to it has been disabled, or that a court or an administrative authority has ordered such removal or disablement.

Hosting Member States shall ensure that the service provider is not liable for the information transmitted, on condition that the provider:[2]

1. the provider does not have actual knowledge of illegal activity or information and, as regards claims for damages, is not aware of facts or circumstances from which the illegal activity or information is apparent; or
2. the provider, upon obtaining such knowledge or awareness, acts expeditiously to remove or to disable access to the information.

However the precondition for these exemptions is that "an information society service is provided that consists of the storage of information provided by a recipient of the service"[2]. In p2p all the information is distributed in the network and service provider provides no storage for the user-generated information. So same as circumstance in U.S. legislation, if p2p service provider has no knowledge of and does contribute to the privacy violation in its network, the duty for it is to remove illicit content expeditiously once has been noticed.

4.2 Case study:Privacy violation caused by service provider

Although situations in privacy violation caused by service provider is not as tricky as it caused by third party, it is a common issue for many tycoons of online social network. These tycoons are sued for multifarious reasons. In November, 2011, after two-year long investigation, Federal Trade Commission(here after FTC) concluded there were many privacy violations existing in Facebook. The privacy issues pointed out by FTC were like: (1)Facebook claimed that third party application was only able to access the needed information of user. But actually third party application could access almost all the personal information, including

needless information. (2) Facebook made some user information, which should be classified, to be public. (3) Facebook shared user personal information with advertiser. (4) Facebook claimed that all the photos and videos can't be accessed after user having disabled or deleted its account. However Facebook still allowed all these accesses. Mr. Zuckerberg who is the chief executive of Facebook admitted all these problems and promised to commit himself to privacy protection. However, recently, Facebook has been involved in many privacy violation cases again. One case is that Facebook adds a "like" button for users to note the content they like in a third party website. However, once user relogged facebook, the corresponding advertisement is added into user's page. The other case is that Facebook, Apple, Twitter and other companies or third party service provider are sued for the reason that they upload user contact information without user's authorization. Facebook even sued for 15 billion dollar penalty for it tracking users even they having logged out. Most cases are still in investigation. For some issues, Facebook argued that they are caused by third party applications. For others, Facebook also compromised to accusation. For the tricky issues, U.S. government appeals that a clear and detailed principal has to be made for privacy protection.

All these cases present a rough idea about the privacy issues usually caused by service provider it self. From these cases, a truth if revealed that online social network conflict with privacy law at the very beginning. Since on one side, users want to share their information to public, on the other side users want totally control their information. Now their is no such system to handle both two points very well. A good choice for service provider is to try best to avoid touching user data. However, it is more clear the responsibility of service provider concerning third party violation.

4.3 Case study:Privacy violation caused by third party

In recent years, there is a hot online phenomenon called "human-powered search". Generally, "human-powered search" is unlike traditional internet information search, it is the information delivered person by person on the internet. Mostly it is used to find someone's identity, contact information or address.

In early phase, a famous case in China called "maltreat of cat" was strongly accused. A girl maltreated her cat at home and made it a video, then upload it to the internet. It made people very angry and asking everyone to find this girl. Finally, someone knew this girl published girl's home address and contact information on internet. From then on, the girl got molested every day. Strangers came to her home or called her phone to accuse her which turned her life into a mess. In this case, this girl's information was published without consent and the girl's privacy was strongly violated. Human-powered search played as a negative role in this case.

In human-powered search, internet service provider plays roles as "search tool provider" and "prospective publisher". In the procedure of human-powered search, participants have to do the search with the help of information search tool. These search tools are provided by internet service provider by the means of every kind of search engine. On the other side, internet service provider also acts as somehow like a "publisher". Among these service providers, there are BBS (Bulletion Board System) providers, e-mail providers or chat-room providers. They act like traditional news media or publisher, but the difference is that, traditional news media or publisher can control or filter the information before it is published to public. But providers such as BBS do not have the ability to control in advance.

The required characteristics of service provider in human-powered search can be concluded as :(i) This service is neutral. It means that distribution or transmission of the information is totally based on user. The provider itself doesn't participate in this procedure. What a provider offers is only the storage of information and corresponding techniques. (ii) This service is automatic and treats all the information equally without priority.

From a worldwide view, the liability of service provider is transited from strict liability to fault liability. In the early time, service provider is identified as being responsible for the censorship of the activities of its network. Once it carelessly neglects its censorship, it is identified as strict liability. However, it brings a huge burden on service provider. In practice, the courts also think it is unwise to bring provider into trouble if it acts as a neutral role. Later in U.S. Digital Millennium Copyright Act(DMCA), it formulate "safe harbor defense" to relax the restrictions for provider. It is now a worldwide tendency to transform the strict liability to fault liability.

The progress of Chinese legislation is later than other developed countries, but it learns experience from developed countries. It defines two regulations for the censorship of service provider. The first one is active obligation, it requires provider, once the illicit transmission is found, provider has to cease and report it. The second one is passive obligation, once the provider is informed by copyright owner or the victim of privacy violation, it has to remove the content expeditiously.

In practice, it is prior to emphasize passive obligation to provider. From the experience in the infringement of human-powered search, usually the searched target's information is provided by anonymous user, or could be only reproduced, therefore it is necessary to find the original publisher. Consequently it is hard for developers to find the true identity of publisher. Based on existing technology, it

is unable to censor the information in the network automatically. If this censorship is done manually, it raises cost for provider and decreases the efficiency of the distribution of information. So generally speaking, provider should be more concentrate on passive obligation. In certain circumstances, if human-powered search becomes a focused event, even if no one notices provider, provider has to consciously find and remove the illicit content, otherwise it would be accounted for fault liability.

In human-powered search, first, although provider can't control or filter the information in advance, but it should apply some filtering program and remove the illicit content within a reasonable time period. Once the provider is noticed by someone, it has to eliminate corresponding content. If provider has done nothing after it being noticed and leaves the damage expanding, it is regarded as liable.

5 illegal, harmful and child-inappropriate content

Except copyright and privacy issue, illegal content is also a serious problem in p2p network. Here the illegal content mostly deals with violent, hateful, pornographic and other potentially harmful content. Downloading or distributing child pornography is serious crimes in both United States and Europe. For the violent or other potentially harmful content, they are largely not illegal, but they are indeed harmful to children. Unlike copyright infringement and privacy violation, there is usually no specific entity would declare that it is offended by child pornography or other kinds of illegal content in p2p network except parents. However, it is usually useless for individual to notice p2p service provider to remove these content, so what can only be done by parents is to keep their child from p2p network and there are also many speeches declaring that preteens should not be allowed to use p2p network. However many public help groups devote themselves to prevent the distribution of such content. They may send the notification of removing child-inappropriate content to service providers.

To escape from filtering programs, pornography traders use other seemingly regular file names to mask the true contents. It is possible for someone innocently downloads these illegal files instead of true files. Therefore, user in the p2p network should report the child pornographic content they found and developer should try to ban these content and distributor in their network.

6 Evaluation

As many issues introduced above, problems in law can be very complicated and depends on many temporal situations. P2P is a new rising technology in this century while it also brings many new tough problems, so the current legislation in most regions has not covered it entirely. But from a developer's view, there are some basic conclusions can be drawn for all these issues. Since many issues are concerned with the storage of service provider, but storage mechanism is different in p2p world.

6.1 Storage in P2P network

In a traditional central network, user data is usually stored in storage disks provided by service provider. Service provider provides the storage space and security for user data. It seems convenient to leave the work of data protection to provider, but it is a single point of failure and provider has a high chance to be affirmed that it has the knowledge of illegal data stored by user, although provider is supposed to not access user data at all in case violating user privacy. What a provider can do is to carefully warn its user not to provide illicit information or violate privacy of third parties. Once it gets the accusation that there is illegal data in its system, it has to remove or disable it right away. This kind of accusation could be very common in a social network and provider could be exhausted.

Compared to the method of storage in a central network, what is done in a p2p network is in a very different way. In a p2p network, user data is decentralized stored in every peer. Each peer contributes one part of its storage space and bandwidth to the network. User retrieves the data depends on a distributed hash table. In a distributed hash table, every value, which is file in this circumstance, is mapped into a key. File is able to be retrieved by using the corresponding key. Since all files are decentralized stored in every peer, each peer contributes part of its bandwidth to ensure the data accessible.

However, one notable issue in p2p network is that, since no centre control exists in this network, every peer is uncontrollable. It may go online and off line any time. The activity that peers getting in and out of the network is described by word "churn". To make data is always available even at a high churn rate, two main strategies are used :(i) Splitting each file into small blocks. (ii) Replicating each block. The purpose of splitting files into small block is for saving storage space and load balancing. For instance, first if two files share one same block, then the storage can be saved. Second, the size of each file is different and out of control, but each block is able to be same so that the load of each peer can be well balanced. And obviously, replicating each block is in case of one peer getting off line with necessary blocks.

In "LifeSocial", the storage of data is decentralized. Due to data is distributed in every peer and there is no central storage space, provider has no access to data which also means that provider has no knowledge of user data. User is the only one has privilege to access files and define the access rules of files, while provider only specifies the mechanism for security of files in software. User is the only

owner and controller of their own data and only the content publisher has rights to delete or remove the content.

Compare to the exemptions provided in Article 14 of the E-commerce Directive. First, conditions in "mere conduit", since transmissions in p2p network is between peer and peer and they are initiated by each individual peer. It is obvious that service provider cannot be the initiate of the transmission; it has no relation with the selection of receiver of transmission and it cannot modify or select information contained in the transmission. Second, conditions in "caching", due to the characteristics described in the storage mechanism of p2p network, service provider is supposed to not cache any information of user. Third, conditions in "hosting", p2p service provider usually will not have actual knowledge of illegal activity or harmful information. However, according to the exemption, as conduit providers have to remove or disable access to such information expeditiously upon obtaining such knowledge or awareness[2].

6.2 Notice and take down

Due to the characteristics of P2P storage, "LifeSocial" or other p2p based social network is able to fulfill the exemption which is "the provider has no actual knowledge of the illegal information". Once user has installed the software, data is entirely controlled by user itself and service provider has no chance to store or access it. In privacy violation concerning third party in p2p network, service provider takes a passive responsibility. The most important and at least duty of p2p service provider is to remove or disable the access of illicit content expeditiously.

"Notice and take down" provision provided in both U.S. and Europe legislation. In e-Commerce Directive, service providers have no liability for illegal content that they hosts as long as they were not responsible for writing it and did not know about its illegal content, and act expeditiously to remove or disable access upon discovery about its illegality[2].

It is clear the meaning of this procedure, after a valid notice from someone, service provider has to remove or disable the content as soon as possible. It is not necessary to describe the detail legal requirements about "notice" and "take down" in this paper, what this paper is interested in is about the word "expeditiously". There is no explicit definition of expeditious by courts. However, in common law, it is described as not immediate, but not undue delay. Some external factors have to be taken into the judgement of expeditious. First, when provider gets an valid notice, it should first consult a qualified counsel to get the legal advice about what should be "taken down" to fulfill notice. However it is easy for big commercial company to get such qualified counsel, possibly there is a law department in such company. For small not-for-profit providers, even maybe held by volunteers, it may not that fast to get a counsel. So it is reasonable for small not-for-profit companies to be over 24 hours. But for big commercial companies, they should inform user the illegal content will be removed or inform the user to remove illegal content within 12 hours. But it is still a great burden for service providers if they have a lot work to do dealing with the content remove. Second the notice delivered to providers could be on holiday or weekend, it is usually not easy for provider to get content publisher and other users who have the copies informed immediately during such time period. Last, the way the notice being sent is also a vital factor, if it is sent by post or fax, there should be a lag between the sender and receiver who is able to act upon it to receive it. Usually the court will take all these reasonable factors into consideration if provider has tried its best.

6.3 Guidelines for P2P developers

If "notice and taken down" can be regarded as on-demand solution and mainly deals with privacy violation concerning third party(but it also can be the solution to copyright infringement), here are some proactive solutions to copyright infringement gained and concluded from past cases.

1. Design the software in a neutral way. The software is ought to be only a platform for user to provide or retrieve content. Do not support priority of content or guide user to certain content. Try to make sure the functions of software do not induce end-user to perform unlawful activities.

2. Keep innocent from stored information in the network. This should be not an issue for p2p network, since all the content is decentralized stored in peers and controlled by publisher itself. But if there is illicit content needed to be removed, service provider should be able to remove it or find the corresponding user and inform him to remove it expeditiously.

3. Give warning and suggestions to users before they join the network. Advise user to do not generate, distribute or reproduce any illicit information. Implement filtering programs to decrease potential illicit content.

4. Keep monitoring the network, once there are illicit activities, remove or cease them as soon as possible. It will be good if all these events are limited to a small range and killed at an early stage.

5. Once get a valid notice, provider should first consult a counsel to get some idea about the extent of infringement and remove the infringing content as soon as possible.

Here are some more guidelines and lessons described as legal landscape in Senior Intellectual Property Attorney Fred von Lohmann's paper "Peer-to-Peer File Sharing and Copyright Law: A Primer for Developers"[8].

1. Choose to monitor users and control them, or just leave it as anarchy. In principal, if the network is kept in anarchy, once service provider is informed by copyrighted

owner that there is infringement in the software. Service provider fulfills the exemption that it has no knowledge of unlawful activities and then stop the infringement as soon as possible. Thereby service provider will not be accused for the secondary infringement.

2.In Napster's case, one fatal drawback is that Napster helps users to search, lay out of target files and lists file providers' ip addresses. Users cannot use Napster's service once the server shuts down. But in Groskter's case, user can still use the software independently without providers assistant. Here is a difference between on-going service and stand-alone software. If the software is designed stand-alone, developers needn't worry about what users are doing, they can also plausibly claim that they have no idea what users are doing, whereas for an on-going service, developers has an easy chance to be accused for secondary infringement. Keep the software decentralized, the more centralized the higher possibility to incur troubles.

What can be further derived from stand-alone software is that, do not even provide any user on-line support or user hotline, leave these things in user community and let them discuss by themselves. Obviously these kinds of communities can be seen in open source software, thereby, open source is another better choice.

3."Capable of substantial non-infringing uses" is a good choice to avoid secondary infringement. This feature born with "LifeSocial", a large amount of functions can be achieved, such as private communication channel, live streaming, publish pictures etc. It is an awful choice to make the software merely be capable of copyright infringement.

4. Be careful with marketing, there mustn't be anything relevant to infringement uses.

5. Of course do not make any financial benefit from end-users' infringement, it will no doubt put providers in dire strait.

6. Here are some more detailed suggestions of technical aspects provided in Lohmann's paper:

- Disaggregate functions of the software.
- Give up the EULA(End-user license agreements).
- Do not support any "auto-updates".

7 Penalty

Penalty usually comes down to money. If service provider is sued and it loses, at first provider has to give up all profit it made. Then, provider may also be accounted for the cost of court(the plaintiff's) and damages as well.

In copyright infringement, the law provides a range from 200 dollar to 150,000 dollar for each infringed work. Recently, a master student in Boston University is sued for illegal downloading and distributing 30 music pieces. According to U.S. Fedral Law, for each infringing music piece, recording company has the right to ask for the penalty from 750 to 30 thousand dollars. At last, he was judged to pay 670,000 dollar for four recording companies.

In privacy violation, the disclosure of personally identifiable information by individual or group, is considered as a misdemeanor and charged for a fine up to 5,000 dollar.

Penalty depends on the extent of infringement or privacy violation. Under certain circumstance it is also possible for the infringer to go to jail. It is always a good choice for provider to limit the rang of infringement or violation, and cease them at early stage as soon as possible.

8 Conclusion

With the increasing number of users in p2p network, to actively or automatically censor network and filter illicit content seem to hardly impossible. But it is still a good suggestion for developers to put attention on "what's happening" in the network. Since nowadays courts put more concentrate on the passive obligation of service providers, keep in mind first keep innocent of user data and be able to remove the noticed illicit content or inform the corresponding publisher to remove it as soon as possible. It won't be a big issue if the illicit content is removed quickly after getting a notice.

Inevitably the nature of online social network conflicts privacy and copyright at the very beginning. Moreover with the high freedom provided by p2p network, users are enabled to share information to a maximum extent, at the same time users want to control their own data and providers have to control user and censor data in the network. It results in a contradiction. Many new and tough issues are happening now or will happen in the future. The current law has not perfectly covered every aspect and made a clear bound between issues. But p2p actually indeed brings a new world to internet and has a promising future. As a p2p tool developer or a p2p service provider, it is always important to keep in touch with counsel and be careful with every questionable part of the software.

References

[1] Copyright Basics. `http://www.copyright.gov/circs/circ01.pdf`.

[2] Article 14 of the E-commerce Directive, 2000. `http://en.wikipedia.org/w/index.php?title=Plagiarism&oldid=5139350`.

[3] U.S. Copyright Law Section 512, Limitations on liability relating to material online, 2010. `http://www.copyright.gov/title17/92chap5.pdf`.

[4] A. N. Dixon. *Liability of users and third parties for copyright infringements on the Internet: overview of international developments*, chapter Introduction: peer-to-peer file sharing

and secondary liability in copyright law, pages 12–42. Edward Elgar Publishing Limited, 2009.

[5] K. M. Hendrik Schulze. Internet Study 2008/2009. Technical report, ipoque, 2009.

[6] J. Kennedy. IFPI Digital Music Report 2008. Technical report, IFPI, 2008.

[7] M. Viola de Azevedo Cunha, L. Marin, and G. Sartor. Peer-to-Peer Privacy Violations and ISP Liability: Data Protection in the User-Generated Web. In *http://ssrn.com/paper=1953904*, 2011.

[8] F. von Lohmann. Peer-to-Peer File Sharing and Copyright Law: A Primer for Developers. In *Peer-to-Peer Systems II*, volume 2735. Springer Berlin / Heidelberg, 2003.

Plugins and their GUI in LifeSocial

Alexander Morast
University of Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Peer-to-Peer Framework for Social Networks
Email: alm@mail.upb.de

Abstract

This paper refers to the project "LifeSocial - A P2P based Online Social Network" based on the work of Kalman Graffi and different students at Technical University of Darmstadt and University of Paderborn since 2007 . As "LifeSocial" is using a p2p framework as basement for the plugin-based achtitecture of the social network application, the communication in between the plugins and their GUI is very important for the functionality and is presented in this paper. In order to overcome the restrictions due to lack of modularisation support in Java a better solution had to be found. Therefore the project group of year 2011 decided to use the OSGi Framework Equinox that implements the OSGi specifications, which imply dividing software into a collection of plugins. By that the software became very flexible, because the different plugins can be added, modified and deleted at runtime. In addition Eclipse RCP, which is built on top of Equinox, was chosen in order to manage both GUI and plugins easily. To improve "LifeSocial" new plugins have to be developed. Therefore the requirements for developing such a plugin and its GUI are analysed and clarified in this paper.

1 Introduction

Since Web 2.0 the sharing of personal information on the internet and thus social networks became very popular. Nowadays online social networks are in fact some of the most visited websites on the internet [1]. The reason for that are the different extensions offered by the social networks. Users for example have the opportunity to publish user-generated content or search for content shared by other users in the social network. Furthermore they can interact with each other by chatting or *Voice over IP* (VoIP).
To provide such a social network different factors are important. On the one hand the providers are interested in attracting new users to increase the earnings by involving advertisements into the social network. But with it as a result of increasing popularity of the online social networks the rising bandwith is attached to high operating costs. So in addition they have to reduce the operational costs, represented by expensive internet connections, to be able to solve all the requests with smallest delay, just as the costs for the mass of servers, that have to be provided and maintained.
On the other hand for the users especially the security of the data is very important because they are sharing personal information with almost everyone in the internet.
To overcome the given problems for both sides the software *LifeSocial*[7] was developed. Since 2007 Kalman Graffi and different students at Technical University of Darmstadt and University of Paderborn worked on *LifeSocial* with the goal to create a completely distributed social network [4]. Therefore the decision was made to implement the online social network on top of a p2p framework [10].
During the work on *LifeSocial* different problems came up. The software became static in case of adding improvements to the social network. But the software should be flexible and this is why the decision was made to use the *OSGi Service Platform*[8] or more specifically the *OSGi Framework Equinox*. From then on the social network consisted of a collection of plugins, which offer the functionalities for the social network.
In correspondance to the plugins the *Graphical User Interface*(GUI) of *LifeSocial* is also divided into different components. That means that the User Interface is separated from the functionality where each component of the *GUI* works with different functions offered by the plugins. This fact supports *Rapid Application Development(RAD)* that improves the development phase of the software project. Since the *Rich Client Platform Eclipse RPC* is using *Equinox*, it combines the dynamic plugin model with a *GUI*. That means, that even different *GUI* elements can be developed independently from each other and combined afterwards.
The entire structure of *LifeSocial* is shown in Figure 1. As result of the shown structure new functionalities can be added easily and an introduction how to implement new plugins for *LifeSocial* will be given in this paper.

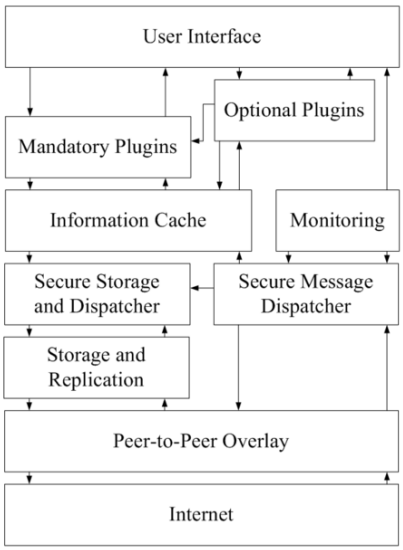


Figure 1. Structure of LifeSocial [7].

In Chapter 2 *OSGi* will be presented in general. Therefore the *OSGi Service Platform* is introduced and the *OSGi Framework Equinox*, which is used in *LifeSocial*, is presented. Additionally *Eclipse RCP* will be introduced in order to explain the *GUI* architecture of *LifeSocial*. Chapter 3 considers the use of *Equinoix* in *LifeSocial* and a concrete example of a working plugin in *LifeSocial* will be presented. In addition the communication in between the plugins and the GUI is presented as well. In the last chapter a conclusion will be given which requirements have to be considered to implement new plugins for *LifeSocial*.

2 OSGi

In order to explain how the plugins in *LifeSocial* work, *OSGi* will be introduced in this chapter. For that purpose the *OSGi Service Platform*, which clarify the requirements for developing a dynamic module system for Java, is presented in section 2.1. In section 2.2 *Equinox* will be introduced as the chosen *OSGi Framework*, which is used in *LifeSocial*. Furthermore *Eclipse RCP* will be introduced as the *Rich Client Platform* on top of *Equinox* in Chapter 2.3.

2.1 OSGi Service Platform

In 1999 the *OSGi Alliance* was founded among others by IBM and Sun Microsystems with the goal to create standards for the development of module-based systems. To enable the platform-independent development of such a system, Java with its underlying *Java Virtual Machine*(JVM) was picked as the favored programming language. But since the *Java ARchive* (JAR) files are used as units in a system developed in Java there is no dependencies management between JAR files. Furthermore the JVM does not provide a dynamic module system. Hence the JAR files are lacking version control(in that case known as *JAR Hell*[12]) and dynamic. In fact JAR files can't be used as modules to develop and manage a module-based system so this is why a new approach had to be found.
As result the *OSGi Service Platform* was developed in order to clarify the specifications which have to be implemented to encourage a service-oriented module system [9]. Since *OSGi* itself only provides the specifications, different frameworks were implemented and build the core of the *OSGi Service Platform Specifications*. Such a framework is divided into different layers to provide the development of a module-based application. In the following the layers are presented:

- Applications(Bundles): Bundles are represented by regular JAR files with an additional manifest file. Moreover the JAR files contain the classes, which represent the functionalities of the bundle, and resources like images and so on.

- Security Layer: The Security Layer controls the boundaries of bundle functionalities with respect to the security of each bundle.

- Module Layer: The Module Layer is responsible for the encapsulation and declaration of dependencies, which are defined in the manifest file.

- Life Cycle Layer: This layer is based on the module and security layer. Further the Life Cycle Layer offers an API to control the life cycle management of the bundles.

- Service Layer: The Service Layer cooperates with the Life Cycle Layer and provides a dynamic collaborative model. Thus bundles can register services in form of objects at the service registry. These services are used for the communication between the bundles. The layer provides different kinds of services including the Declarative Services.

- Execution Environment: The Execution Environment defines the Runtime Environments which can be set as requirement for a bundle.

The entire achitecture of the *OSGi Framework* is shown in Figure 2.

Consequently bundles can be added to a system on runtime. In order to use a bundle in a running system, it has to be installed first. Thereby an *URL* has to be sent to the framework, which leads to the JAR file of the bundle. The framework controls the bundle on the basis of the manifest file. If the manifest file contains the required information the bundle will be included into the frameworks storage. After a successfull installation the bundle has to be resolved

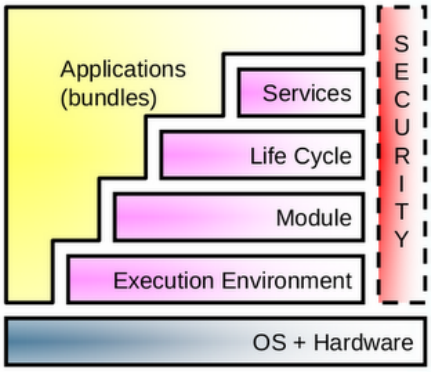


Figure 2. OSGI Framework Architecture[6]

.

in order to be started. The framework checks the dependencies defined in the manifest file for existence. If the dependencies are checked and all required bundles are available, the bundle can be activated. The activator, which is defined in the manifest file of the bundle, will be invoked and starts the bundle. While it is activated, the bundle can be stopped at all times. Thereby the services, that are registred by the bundle, will be unregistered and afterwards the bundle can be uninstalled or started again. Futher by uninstalling, the bundle will be removed from the frameworks storage and completely deleted from the framework.

2.2 Equinox

Equinox is an implementation of the *OSGi Framework Specifications* and was developed to reorganize the architecture of *Eclipse*[2], which is an *Integrated Development Environment*(IDE) platform, towards a dynamic plugin-based architecture. Since version 3.0 of *Eclipse*, which was released in 2004, *Equinox* is used to organize the plugins on runtime in *Eclipse*. Furthermore *Eclipse* itself offers a *Plugin Development Environment*(PDE) to develop *OSGi* bundles. Compared to other Frameworks the *PDE* allows an easier development of bundles and this is why it was chosen to be used in *LifeSocial*.
In order to use the framework you can either run *Equinox* from *Eclipse* or directly in the console as a standalone OSGi server. By starting *Equinox* in the console, the `osgi>` prompt appears and commands can be send to the framework. After installing a bundle by sending the `install <FileURL>` command, where `FileURL` leads to the JAR-file of the bundle, the framework replies by display the id of the bundle given by the framework. The output of the console looks as follows: `Bundle id is <#ID>`. To use the installed bundles you can use either the name of the bundle or the given id. With the command `ss` the framework lists a *short status* of all bundles installed at the framework. Usually the system bundle is installed and activated by default and has the id 0. Moreover you can start and stop a bundle by sending the `start` or `stop` command followed by the id or the bundles name. So as to shut down the whole framework the *close* command has to be invoked.
Figure 3 shows the complete life cycle of OSGi bundles.

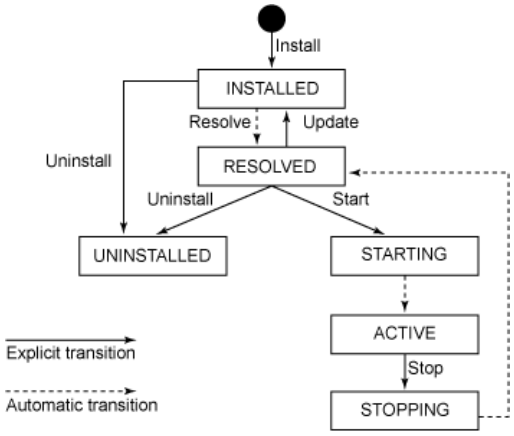


Figure 3. States of a OSGi bundle[5].

In order to use *Equinox* in the *Eclipse IDE* a *Run Configuration* has to be created and customized. By that all required plugins can be included to the configuration and do not have to be started by hand.

2.3 Rich Client Platform

Eclipse provides a *Rich Client Platform*(RCP) called *Eclipse RCP*, which contains *Equinox* as runtime environment. *Eclipse RCP* combines the dynamic plugin system managed by *Equinox* with *GUIs* supported by the *GUI Framework Standard Widget Toolkit*(SWT) The entire architecture of *Eclipse RCP* is shown in Figure 4.

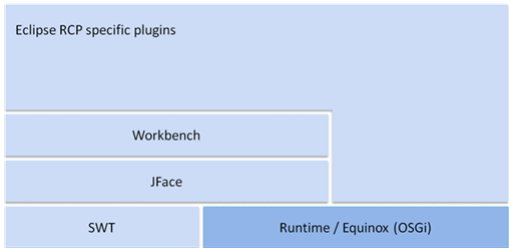


Figure 4. Eclipse RCP Architecture.

By using *Eclipse RCP*, *GUI* applications can be developed very easy, because the *GUI* can be divided into different parts. Therefore a *Workbench* with its structure is provided by the *RCP* and only have to be filled with views and editors, which also can be added by different bundles. The views contain *GUI* elements that are provided by the

GUI Framework SWT. Since the other *GUI Frameworks AWT* and *Swing* are slow in performance, *SWT* is a good choice to implement *GUI* applications. Another argument for using *SWT* for *GUIs* is the fact that *SWT* is using native widgets. That means that the look of an application is dependent of the operating system on which it is used.
In correspondance with *SWT*, *JFace*, which is included on top of *SWT* in the *Eclipse RCP* architecture, provides the *Model-View-Controller*(MVC)[3] programming approach. By using *JFace*, widgets provided by *SWT* that can contain data e.g. like a list, can be managed more simply. In addition *JFace* extends *SWT* by providing wizards and dialogs.

3 LifeSocial's Plugins

Since *LifeSocial* is based on the *OSGi Framework Equinox*, everything in *LifeSocial* is a bundle. As *LifeSocial* provides its own plugin-system, we have to distinguish between core-bundles and plugin-bundles. By that the development phase of plugins can be improved because the GUI can be developed independently from the plugins, which store the functionalities of the social network. Plugins in *LifeSocial* can be either mandatory or optional. A screenshot of the GUI of *LifeSocial* is shown in Figure 5.
According to [7] *LifeSocial* includes following plugins, which provide basic online social networking-functionalities:

- Login: The login plugin allows the registration and the login into the social network.
- Profile: This plugin provides a description and an image of the user.
- Friends: The friends plugin contains a list of links to the profiles of the users friends.
- Messaging: The messaging plugin offers the inbox-, outbox- and message composer-views.
- Photo: This plugin countains a list of photo albums, which link to user photos stored in the p2p network.
- Groups: The groups plugin offers a list of users that joined in a common interest group.
- Tweets: List of status updates of a user and its followers
- Chat: This plugin allows direct communication by user to user text messaging.
- File transfer: With this plugin the users can send files to other user.
- Tic-Tac-Toe: This plugin offers the game Tic-Tac-Toe for two players. Additionally for spectators the plugin allows to observe the game session.
- Whiteboard: The whiteboard plugin provides a collaborative graphical editing of a canvas.
- Calendar: With the calendar plugin the user can manage a collaborative time schedule. Therefore the users can edit and share their time schedules.
- Multicast: This plugin allows the creation of multicast groups and publishing to multicast groups

Besides the mentioned plugin-bundles, *LifeSocial* contains bundles, which provide the p2p platform. Furthermore it contains bundles, that provide the core functions for the social network. These functions allow the plugins to receive and to store the needed data from the p2p network. In addition interfaces for the communication between the GUIs and the plugins are provided. In fact, both new GUI-bundles and plugin-bundles can reuse functions from already existing plugins in *LifeSocial*.
In order to understand how the GUI corresponds with the plugins, the source code has to be analysed deeply. Therefore the requirements for creating plugins are presented by introducing an existing plugin of *LifeSocial* in Chapter 3.1 first. Afterwards the GUI-bundles are presented in Chapter 3.2 in order to analyse how the GUI-bundles use the functions offered by the plugin-bundles.

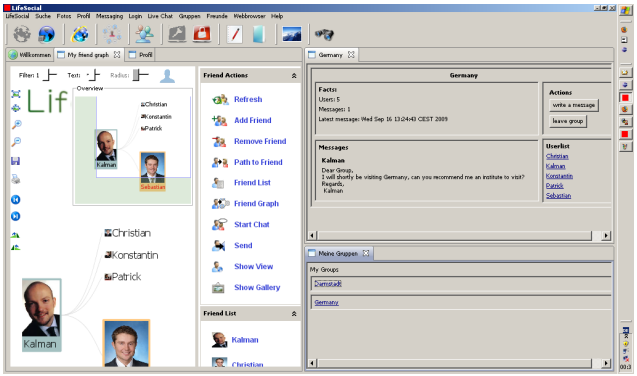


Figure 5. Screenshot of LifeSocial[7]

3.1 Plugins

Besides the fact that *LifeSocial* consists of different bundles, an own plugin-system was developed in order to improve the application more easy. Especially the separation of functionality and GUI into different bundles supports the RAD.
In *LifeSocial* different bundles provide core funtionalities for the application. All the bundles beginning with `org.lifesocial.core.*` offer interfaces e.g. in order to manage data handling or the communication between plugins and GUI-bundles. The bundle `org.lifesocial.core.main` merge all

these core bundles with the bundles providing the p2p framework. Furthermore it includes the package `de.tud.kom.lifesocial.core.commands`, that is responsible for the administration of the commands, that should be used by the GUI-bundles. So this bundle is the main requirement for future plugins in order to be runnable in the *LifeSocial* environment. Further the `org.lifesocial.core.utilities` is also required for additional plugins, because this bundle offers an interface for administration of plugin identifiers as well as some helper classes.

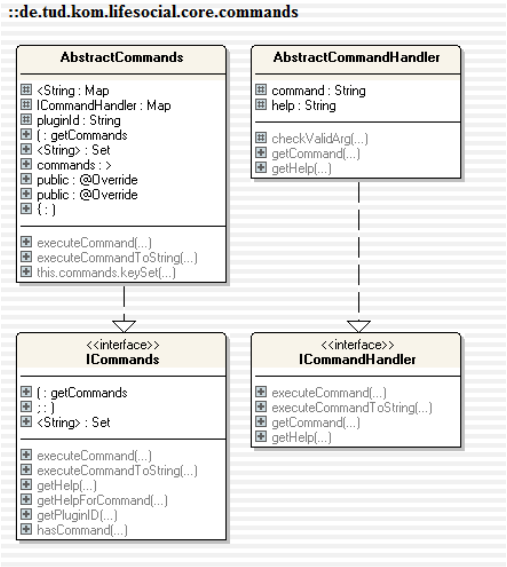


Figure 6. Classes in package org.lifesocial.plugin.commands

Since the plugin-bundles in *LifeSocial* provide declarative services, 3 steps for developing new plugins have to be considered: the definition of the API which has to be provided for other bundles, the definition of the bundle-dependencies and the configuration of the declarative services. These 3 steps will be presented below by analysing the plugin-bundle `org.lifesocial.plugin.searchplugin`.

3.1.1 API

Plugins in *LifeSocial* are responsible for the high-level functionality of the application and provide an interface for the GUI. In order to understand how the GUI-plugins are able to use the functions provided by the plugins, the structure of a plugin-bundle has to be analysed. Therefore the components of the `org.lifesocial.plugin.searchplugin` bundle will be presented below (also see Figure 7).

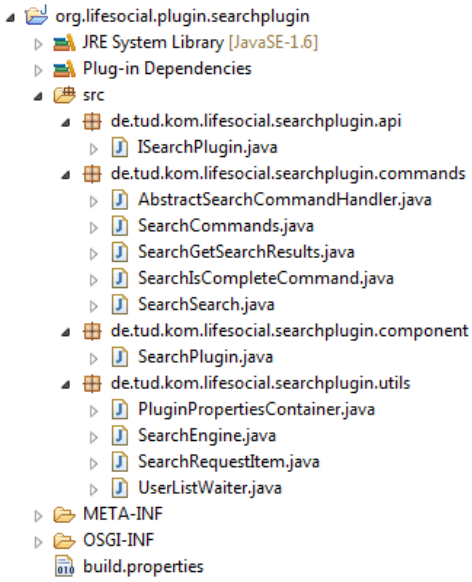


Figure 7. Content of the bundle org.lifesocial.plugin.searchplugin

The mentioned bundle contains 4 packages. The first package `de.tud.kom.lifesocial.searchplugin.api` only includes an interface. The interface `ISearchPlugin` includes all the methods that will be provided for the other bundles. In correspondance to this package the package `de.tud.kom.lifesocial.searchplugin.component` is created. This package includes also only one class. This class is called `SearchPlugin` and provides the implementation of the methods defined in the interface `ISearchPlugin`. The method `activate`, that is defined in this class, is important for the *Declarative Services*. These services are explained in Chapter 3.1.3.

The package `de.tud.kom.lifesocial.searchplugin.commands` (Figure 8) contains the abstract class `AbstractSearchCommandHandler` which extends `AbstractCommandHandler` provided by the package `de.tud.kom.lifesocial.core.commands` of the bundle `org.lifesocial.core.main` (see Figure 6). This class holds an object of the type `ISearchPlugin`. The including methods `setSearchPlugin` and `unsetSearchPlugin` control the binding of this object. Moreover for each method that should be accessible by other bundles, an own class is created and is extending the `AbstractCommandHandler`. Thus all these classes include the methods `executeCommand` and `executeCommandToString` which are responsible for the execution of the exported methods. In Figure 9

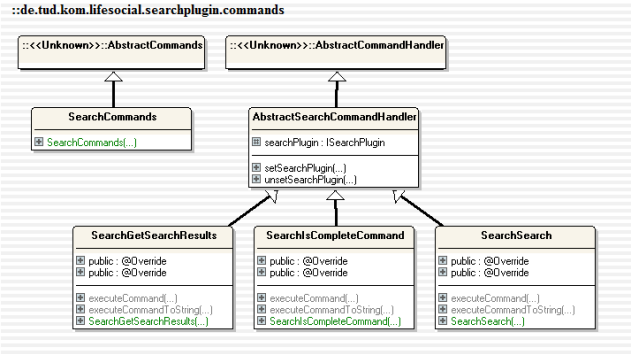


Figure 8. Content of the package de.tud.kom.lifesocial. searchplugin.commands

the implementation of the method `executeCommand` of the class `SearchGetSearchResults` in shown. In addition the class `SearchCommands` represents the commands which are provided for the other bundles.

```
@Override
public Object executeCommand(String arg) {
    if (this.checkValidArg(arg)) {
        int ticket = -1;
        ticket = Integer.valueOf(arg.trim());
        Collection<String> results = this.searchPlugin.getResults(ticket);
        return results;
    }
    return PluginPropertiesContainer.ParameterMissingMsg;

}
```

Figure 9. Implementation of the method executeCommand for the command GetSearchResults.

The last package of the plugin-bundle is called `de.tud.kom.lifesocial.searchplugin.utils` and provides the properties for the plugin. Especially the definition of the names of the exported methods are important. Furthermore the package includes helper classes, which are needed in order to execute the methods.

3.1.2 Manifest

In order to insert a new plugin into the application, several requirements have to be accomplished. Especially the dependencies of the plugins are important, since the plugins have to provide the functionality in order to be used by the GUI bundles. Therefore the `MANIFEST.MF` file of a bundle is very important. This file has to be put into the `META-INF` folder of the bundle. Inside of the manifest file the dependencies of the bundle have to be defined just as the metadata like version number and name.
To understand how the manifest file works, an example is given by analysing the manifest file of the bundle called `org.lifesocial.plugin.searchplugin`, that is presented in Listing 1.

Listing 1. MANIFEST.MF of the Search Plugin

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: SearchPlugin
Bundle-SymbolicName:
 de.tud.kom.lifesocial.searchplugin
Bundle-Version: 1.0.0
Bundle-Vendor: Luciana Alvite
Bundle-RequiredExecutionEnvironment:
 JavaSE-1.6
Require-Bundle:
 de.tud.kom.lifesocial.
   utilities;bundle-version="1.0.0",
 de.tud.kom.lifesocial.
   core;bundle-version="1.0.0"
Import-Package:
 de.tud.kom.lifesocial.logger.api,
 de.tud.kom.lifesocial.middlewarestorage.
   informationcache.main,
 de.tud.kom.lifesocial.middlewarestorage.
   informationcache.utils,
 de.tud.kom.lifesocial.middlewarestorage.
   storagedispatcher.api,
 org.osgi.service.component;version="1.0.0"
Export-Package:
 de.tud.kom.lifesocial.searchplugin.api
Service-Component:
 OSGI-INF/searchenginecomponent.xml,
 OSGI-INF/searchplugincomponent.xml,
 OSGI-INF/searchrequestitemcomponent.xml,
 OSGI-INF/userlistwaitercomponent.xml,
 OSGI-INF/searchcommandscomponent.xml,
 OSGI-INF/searchgetsearchresultscommand.xml,
 OSGI-INF/searchsearchcommand.xml,
 OSGI-INF/SearchIsCompleteCommand.xml
Bundle-ActivationPolicy: lazy
```

In the first lines of the presented manifest file, the metadata of the bundle is defined. Especially the manifest header `Bundle-SymbolicName` is important since it is the only required entry. It defines a unique identifier for the bundle in correspondance with the header `Bundle-Version`, which is also important for the version control. Furthermore the `BundleRequiredExecutionEnvironment` header defines the runtime environment that is required for the usage of the bundle. By setting the `Bundle-ActivationPolicy` to lazy, the bundle will be only started when it is needed first. This option allows to improve the startup performance just as the handling

performance of the applications.
Up next the requirements for the usage are defined. Therefore the `Require-Bundle` and the `Import-Package` header have to be defined. Both headers have the same functionality. The only difference is that by requiring bundles all the exported packages are accessible and by importing only the referenced package. But as far as possible importing packages should be prefered[11]. Since every plugin needs to use the classes from the package `de.tud.kom.lifesocial.utilities` which is the only package of the bundle it could also be imported as package. In addition all plugins use packages from the bundles with the prefix `org.lifesocial.core.*`. In order to connect these packages with the plugin, either the bundle `org.lifesocial.core.main` has to be required or the specific packages have to be imported by hand.
Since the plugins have to provide functionalities, they also have to export some packages. By defining the `Export-Package` header, the packages from the plugin can be provided for the use by other plugins just as the GUI-bundles.
If the bundle is going to provide declarative services, the header `Service-Component` has to be defined. This header includes all the names of the declarative services that will be provided by the bundle.

3.1.3 Declarative Services

In order to provide functionalities for other bundles, services have to be defined and registered at the service registry. Since there are service providers and service consumer, the plugin-bundles have to be defined as service providers and the GUI-bundles as service consumer.
There are two ways of defining a service. It can be done either by using the class *Activator* to register the service and use the service tracker. Or by the declaration approach of defining services. Thus the services are defined beyond the class `Activator` and are not registered automatically when its bundle is started. The *Declarative Services* are configured in XML and have to be put into the `OSGI-INF` folder of the bundle. In order to understand what have to be considered to configure a declarative service, the configuration files of the bundle `org.lifesocial.plugin.searchplugin` are presented below.
Service Components are created for every class that implements a command in the package `de.tud.kom.lifesocial.searchplugin.commands`.
In addition the implementation of the interface `ISearchPlugin` got its configuration file just as the helper classes in the package `de.tud.kom.lifesocial.searchplugin.utils`.
As example the configuration file for the class `SearchIsCompleteCommand`. This configuration file is shown in Figure 10.
At first the component is registered with its name called `SearchIsCompleteCommand` at the service registry. This component is providing a service to the service registry under the interface `de.tud.kom.lifesocial.searchplugin.commands.AbstractSearchCommandHandler` implemented by the class `de.tud.kom.lifesocial.searchplugin.commands.SearchIsCompleteCommand`.
Moreover a reference is defined. This reference says that this component has a dependency on the service represented by the interface `de.tud.kom.lifesocial.searchplugin.api.ISearchPlugin`. The `bind` command defines a method which will be invoked by the declarative service when a `ISearchPlugin` service is registered at the service registry. This command is linked to the method `setSearchPlugin` of the class `AbstractSearchCommandHandler` in order to link to the exported methods. The `unbind` command will be called when the binded service is not anymore available. The entire XML code of the described configuration file is shown in Figure 10.

```
<?xml version="1.0"?>
<component name="SearchIsCompleteCommand">
    <implementation class="de.tud.kom.lifesocial.searchplugin.commands.SearchIsCompleteCommand"/>
    <service>
        <provide interface="de.tud.kom.lifesocial.searchplugin.commands.AbstractSearchCommandHandler"/>
    </service>
    <reference name="SEARCHPLUGIN"
        interface="de.tud.kom.lifesocial.searchplugin.api.ISearchPlugin"
        bind="setSearchPlugin"
        unbind="unsetSearchPlugin"
        cardinality="1..1"
        policy="dynamic"/>
</component>
```

Figure 10. Configuration file for the command SearchIsComplete.

3.2 GUI

LifeSocial uses SWT as framework for its GUI just like in Eclipse, since it is based on *Eclipse RCP*. By using perspectives and views the GUI can be changed dynamicaly. Hence the GUI can be customized at runtime by adding and moving the views.
As mentioned above GUI-bundles represent the service consumer. Thus they have to use the API provided by the plugin-bundles. Futhermore the architecture of the GUI-bundles is also important in order to understand how the API for the search plugin is used by its GUI-bundle.

3.2.1 SWT

First of all a main GUI-bundle was created in order to handle the upcoming GUI-bundles. This bundle is

called `org.lifesocial.ui` and provides all the needed classes in order to run the GUI. Futher this bundle provides the `Workbench` that is hosting the perspecitves and views. By that this bundle is the central bundle of the entire application and has to be invoked in order to start *LifeSocial*. This is why the bundle `org.lifesocial.ui` is required by each GUI-bundle.

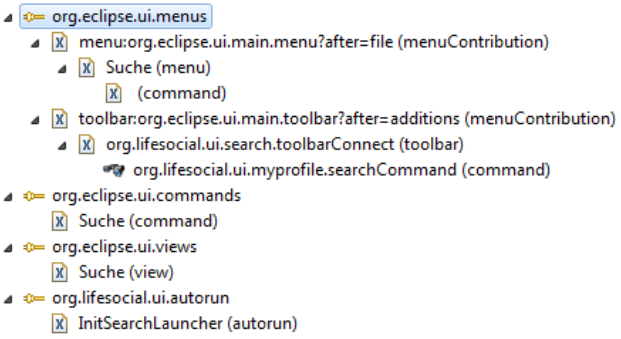


Figure 11. Extensions of the bundle org.lifesocial.ui.search

The GUI-bundles provide the menus and views that are relevant for the specific plugin-bundle. These have to be defined in the manifest file(see Figure 11). The framework SWT provides different widgets that can be included into the views. For that each view that should be provided by the GUI-bundle gets its own class. In this GUI-bundle it is only the class `SearchView`. In addition the class `SimpleViewHandler` was created in order to control the view.

3.2.2 Architecture

The API that is provided by the plugin-bundles has to be imported by the GUI-bundles. Therefore the bundles need to import the package `de.tud.kom.lifesocial.commands.api` that is provided by the bundle `org.lifesocial.plugin.commands`. The entire content of this bundle is presented in Figure 12.

To use the API methods provided by the plugin-bundles, these methods are defined in the class `PluginPropertiesContainer`. These methods have to be named equivalent to the methods defined by the plugin-bundle.

Moreover the methods need to be fetched. Therefore a declarative service is created (see Figure 14). Its configuration file connects the class `Activator`, that is used as the `Bundle-Activator`, with the package `de.tud.kom.lifesocial.commands.api.ICommandsComponent`. The class `ICommandsComponent` contains a set of objects with the type

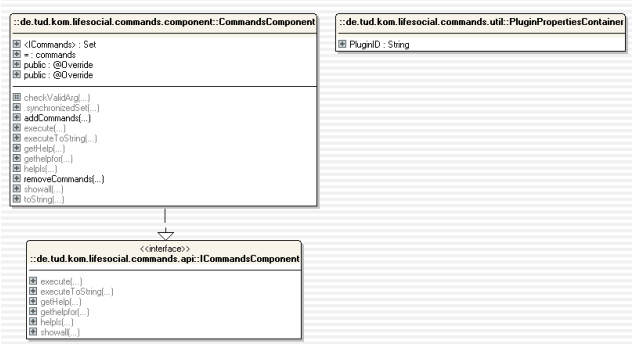


Figure 12. Content of the bundle org.lifesocial.plugin.commands.

```
public class PluginPropertiesContainer extends PropertiesContainer {

    // plugin ID
    public static final String PluginID = "SearchPlugin";

    /** Plugin Commands **/

    public static final String CommandSearch = PluginID.concat(".").concat(
            "Search");
    public static final String CommandSearchIsComplete = PluginID.concat(".")
            .concat("SearchIsComplete");
    public static final String CommandGetSearchResults = PluginID.concat(".")
            .concat("GetSearchResults");
```

Figure 13. Definition of the methods in the PluginPropertiesContainer

`ICommands` (see Figure 6). The objects of the type `ICommandsComponent` is binded with the `addCommands(ICommandsComponent command)` to the GUI-plugin in order to be able to use the methods provided by the plugin-bundles.
In addition the class `Startup` is created. This class implements the interface `IAutorun`, which is provided by the bundle `org.lifesocial.ui`, in order to start the GUI-bundle automatically by invoking the class `Activator`. Futher an extension in the `plugin.xml` is created to enable the autostart of the bundle.

```
<?xml version="1.0"?>
<component name="SearchPluginGUIcomponent">
    <implementation class="org.lifesocial.ui.search.Activator"/>
 <reference bind="addCommands"
            cardinality="1..1"
            interface="de.tud.kom.lifesocial.commands.api.ICommandsComponent"
            name="COMMANDS"
            policy="dynamic"
            unbind="removeCommands"/>
</component>
```

Figure 14. Configuration file for the GUI-bundle.

4 Conclusion

By using *Eclipse RCP* with its underlying *OSGi Framework Equinox*, *LifeSocial* consists of several bundles. Thus the bundles are separated into plugin-bundles and GUI-bundles in order to divide the functionality from the GUI. To implement new plugins for *LifeSocial* different requirements have to be considered just as the fact that the GUI-bundles have to fetch the functionalities from the plugin-bundles.

To develop a plugin-bundle 3 different parts are important. At first the API, which have to be exported, have to be defined. For that especially the bundles with the prefix `org.lifesocial.core.*` have to be considered. These bundles offer functions in order to work with the data which is stored in the online social network. Further the manifest has to be configured in order to specify the requirements of the plugin-bundle. In the end the *Declarative Services* have to be defined for each method that has to be exported to the GUI-bundles.

The GUI-bundles are considered to be the service consumer. That is why these bundles have to fetch the methods, that are provided by the plugin-bundles. The GUI-bundles need to import the package `de.tud.kom.lifesocial.commands.api` that is provided by the bundle `org.lifesocial.plugin.commands` in order to use the methods inside of the GUI-bundles. Moreover each GUI-plugin only have to concentrate on the views that relevant. The bundle `org.lifesocial.ui` collects all the views and provides them for the application.

All in all, the use of *Eclipse RCP* is very convenient, since it combines the *OSGI Service Platform* with the GUI Framework *SWT*. Thus plugin- and GUI-bundles can be developed independently. In addition a bundle repository could be provided easily, since each bundle can exist without other bundles.

References

[1] Alexa.com. The top 500 sites on the web. Online available at http://www.alexa.com/topsites . Last accessed on June 4th, 2012.

[2] Eclipse. Homepage of Eclipse. Online available at http://www.eclipse.org . Last accessed on June 4th, 2012.

[3] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[4] K. Graffi, C. Groß, P. Mukherjee, A. Kovacevic, and R. Steinmetz. LifeSocial.KOM: A P2P-based Platform for Secure Online Social Networks. In *IEEE P2P '10: Proceedings of the International Conference on Peer-to-Peer Computing*, 2010.

[5] IBM. Understanding Eclipse's new bundle-management mechanism. Online available at http://www.ibm.com/developerworks/opensource/library/os-eclipse-bundlemgmt/index.html . Last accessed on June 4th, 2012.

[6] JBoss Community. Introduction to OSGi. Online available at https://docs.jboss.org/author/display/JBOSGI/Introduction . Last accessed on June 4th, 2012.

[7] LifeSocial. Lifesocial, 2012. Online available at http://www.lifesocial.org . Last accessed on June 4th, 2012..

[8] OSGi Alliance. Benefits of Using OSGi. Online available at http://www.osgi.org/About/WhyOSGi . Last accessed on June 4th, 2012.

[9] OSGi Alliance. OSGi Service Platform Core Specification Release 4.3, 2011. Online available at http://www.osgi.org/Release4/Download . Last accessed on June 4th, 2012.

[10] Project Group - A Distributed Framework for Social Networks. Final Report, 2011. University of Paderborn.

[11] L. Vogel. Eclipse Plug-in dependencies Required Bundle vrs. Import-Package. Online available at http://www.vogella.com/blog/2009/03/27/required-bundle-import-package/ . Last accessed on June 4th, 2012.

[12] H. Yang. Class Loading Problem - JAR Hell. Online available at http://www.herongyang.com/JVM/ClassLoader-Class-Load-Problem-JAR-Hell.html . Last accessed on June 4th, 2012.

Distributed App-Store: Hosting and Search of Software Bundles

Yevgen Mexin
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: yevgen@mail.upb.de

Abstract

For storage, installation and the start-up of plug-ins, LifeSocial uses the OSGi Framework. Currently, OSGi standard does not include the technology for the remote loading of the components. Some OSGi realisations maintain centralised solutions for connecting to the remote server as well as loading components. Such approaches have some inherent problems with performance and stability. The decentralised p2p approach eliminates these problems. There are various approaches to achieve decentralised p2p networks organization, and each approach has its benefits and disadvantages. In this article four solutions - POSGi, D^2CM, Chord4S and RSP - based in different technologies will be presented. All of them can be used for the distribution of OSGi components.

1 Introduction

LifeSocial, a distributed framework for social networks, needs a mechanism for the distributed storing and managing plug-ins, the latter will expand its functionality. For managing plug-ins, LifeSocial uses Open Services Gateway Initiative (OSGi) framework which is the de-facto standard for component-based and service-oriented Java applications.

OSGi is a very powerful tool for managing of components. However, at this time, OSGi standard does not include the technology for the remote loading of the components. Some OSGi realisations (for instance, OSGi Bundle Repository (OBR)) maintain the centralised solutions for connecting with the remote server and the loading of the components.

However, such approaches have some inevitable problems with performance (bottleneck problem) and stability (single point of failure problem). Besides, the support of a special server, which would send the requested components to clients, is associated with certain expenses of monetary (premise rent, electricity payment) and human (server administration) resources.

The decentralised approach solves the abovementioned problems. For this purpose, each remote device will be a peer in a peer-to-peer network. Peers distribute the components amongst themselves.

There are various technologies for establishing the decentralised organisation of the p2p networks. Some are based on Distributed Hash Tables (DHTs), for example Pastry and Chord. Each technology has its corresponding benefits and disadvantages.

Approaches for the discovery, storage and the exchange of components are created based on these p2p technologies. The features of such solutions depend on the chosen base p2p technology. Each solution should realise mechanisms of distribution, discovery and identification of components.

In this article four solutions based on different technologies will be presented. All of them can be used for the distribution of OSGi components.

This article consists of five parts. The first section is introduction, which is followed by a chapter entitled "Fundamentals", in which base knowledge of OSGi, p2p technologies, plug-ins and security by using of plug-ins will be provided. The third section, "Distributed approaches", the four approaches for component distribution will be shown. The fourth part will demonstrate, which of the provided solutions is most suitable for the management of plug-ins in LifeSocial. The small conclusion finishes the article.

2 Fundamentals

2.1 OSGi

OSGi [13] [1] is an open platform for distributing and executing service-oriented and component-based applications. OSGi is provided by OSGi Alliance. Currently, OSGi framework is a de-facto standard for component-based Java applications.

The components in OSGi are called bundles. Technically, bundles are Java Archives (JARs) containing applications and libraries. The bundles are described by a special manifest file, that contains metadata (e.g. bundle name, version, dependencies and other properties).

The bundles can be installed and uninstalled, started and stopped or updated. OSGi manages this bundle life cycle.

The changes in the bundle state can be made at runtime.

The bundles maintain services, that can be published and used by other bundles. OSGi services are Java objects which produce functionalities and are described by service interfaces. OSGi contains the service registry, which manages all the services published by the bundles. The package, that wishes to publish a certain service should register it in the registry. The service registry then informs all the bundles about any changes in the states of the published services through the special events.

There are several implementations of OSGi. The most popular implementations are Oscar, Knopflerfish, Apache Felix and Oscar. The OSGi specification is widely used in the industry.

2.2 Structured p2p overlays

Peer-to-peer technology is self-organizing decentralised distributed system architecture. The network members (peers) share their system resources with each other.

The p2p overlays are divided into unstructured and structured overlays. The unstructured p2p overlay networks (e.g. Freenet, Gnutella) are mostly inefficient.

The structured p2p overlays use some technologies to organise the nodes and to create efficient mechanisms for sharing, discovery and loading of contents. On the other hand, the structured p2p overlays need some additional maintenance and use more resources to support their organised structures.

The structured p2p network has to realise a balanced mapping of resources between the peers. The Overlay technologies define some primitives, for example a routing between peers. Some popular structured overlays are Pastry, Controller Area Network (CAN), Chord and Tapestry.

2.3 Pastry

Pastry [10] is the structured p2p overlay technology based on the DHT. Pastry organise the nodes in a structured p2p network. Pastry is scalable and can also be used in large-scale p2p networks.

Each node in Pastry becomes a unique identifier. The data transferred and stored in the network also becomes a unique key. The data with key $= x$ will be stored by a node, the identifier of which is nearest to x. The identifiers and keys of nodes and data objects are chosen randomly from range $[0, 2^{128})$.

Pastry uses the prefix-based routing method to search the requested object with some key. (See Figure 1)

2.4 Benefits of plug-ins

Plug-ins can be used in LifeSocial for expansion of its base functions as well as simple and fast addition of various new features. LifeSocial supports many useful services, but the necessity to expand the available functionality of the program can arise at any moment.

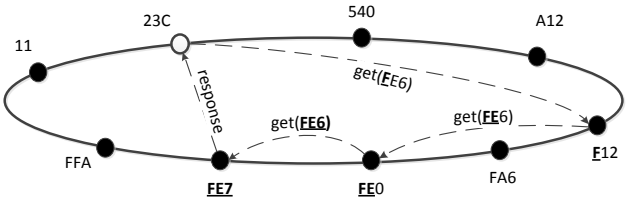


Figure 1. Prefix-based discovery in Pastry

Plug-ins allow to easily add new optional functions into client applications. The modification of the main application for this purpose is not required, it isnt necessary to release patches and to distribute them among all users. Plug-ins add new features, that are optional, dont change the code of the basic program and can be installed, started, stopped and uninstalled by the user at his or her will. Usually, plug-in management requires the restarting of the main application for the subsequent appropriate work with the plug-in. The realisation of plug-ins by means of the OSGi-bundles allows eliminating this inconvenience. The OSGi platform carries out the installation and uninstalling as well as the starting and stopping of the OSGi-bundles in the manner such that these actions can be completed at runtime.

Plug-ins can be created not only by project developers, but also by third-party developers or even by the users. The developers, that have created the Application Programming Interface (API) for the interaction with the main program, distributed the Software Development Kit (SDK) for the creation of plug-ins and published the necessary documentation, can shift the work on minor functionalities to the third-party developers and so concentrate their attention on improving and solving the errors in the main software product. The developers that support such initiatives can create Internet resources and forums for communication with plug-ins developers, thus creating special Internet communities. The information in support for such Internet communities, feedback by the users, and any increase in the interest of the user in creating of new plug-ins will be a reasonable step for project developers. For example, contests for best plug-ins can be held for encouraging such participation.

2.5 Plug-ins security

Maintaining the security of the system when using plug-ins is necessary to support third-party plug-ins. The API is used by the plug-ins to interact with the main application, and, whenever possible, should not give an access to functions, that can directly affect the working capacity of the application and users operating system, provide an access to users files or especially give unlimited remote access to the system. A check of the plug-ins hash-sums or even digital signatures of plug-ins and developers will not be superfluous.

By means of these and other precautions it will be possible to guarantee relative security against the harmful code, added in plug-ins by intruders. Such precautions can be excessive in private p2p networks which unite one enterprise, university or group of persons and in which every user can be identified and is reliable. However, in large public networks security is very important.

Use of license agreements, that allow users to distribute and to publish the plug-ins only as open source, can be a useful tactical step from the point of view of security. Thus, the harmful code can be relatively quickly detected and the unsafe plug-ins can be removed by users. The publication of plug-ins with an open source code also brings the usual advantages of such software, such as quick detection of errors and holes in the security or improvement of the plug-in code by third-party programmers.

3 Distributed approaches

In this section various approaches for distribution and discovery of OSGi components and services will be described. At the conclusion, an attempt to choose the most suitable solution for LifeSocial will be made.

For the discovery and the distribution of the components in a p2p network, first of all, a description method must be chosen. This seemingly trivial problem defines in many respects the basic features of solution and strongly depends on the chosen p2p technologies.

The most basic property of a component is its name. This property is included in any set of properties used for search and storage of components. The use of additional characteristics expands the functionality of the technology.

Properties of components can be conditionally divided either in functional or non-functional properties. Functional properties describe the utility of components, for instance, a plug-in for editing of images. Non-functional properties describe other qualities of the components, such as performance or additional features. Compatibility (e.g. operating system, hardware, language, and platform) and possible dependencies on other components are essential properties of bundles.

The properties used for the description of the components can be additionally classified as mandatory and optional. Mandatory properties, for example the component name, the component version or demanded operating system, are used directly for search and selection of suitable bundles.

In systems that allow for the use of optional properties, additional processes of evaluation of the founded bundles and the choice of the most suitable component can be realised. Optional properties can be any, for example, presence of unessential features.

The possibility to use properties by means of the distribution and discovery of components directly depends upon the chosen p2p technology. For example, technologies based on DHT permit to use only the properties included in the component identifier. On the other hand, Juxtapose (JXTA) Technology allows using a much wider set of properties described in special advertisements used by the JXTA.

3.1 POSGi

The following approach from Stephane Frenot and Yvan Royon [2] describes the distributed sharing and discovery of the OSGi bundles in the decentralised structured p2p networks. POSGi is based on Pastry technology. The given approach supports version management and resolves simple search and installation issues of the last version of the components. By means of the distribution and discovery in a p2p network, POSGi considers only name and version of bundle. Additional characteristics, such as non-functional properties, are not taken into consideration.

All peers can share components in this type of network. To register a bundle in the network, the peer should send a deposit request. For the support of version management, two such requests are necessary. By means of the hash-function known to all nodes in the network, the source-peer calculates from a component name its key and peer with the ID nearest to a received key ($rootnodeID = hash(< bundleName >)$), which stores the number of last version of a bundle. Then the node, the ID of which is the nearest to $hash(< bundleName > _ < version >)$, stores the bundle under a name $< bundleName > _ < version >$. The following listings are taken from the article [2].

```
publish(hash(<bundleName>), <version>);
publish(hash(<bundleName>_<version>),
bytecode);
```

The peer, wishing to install a component with name $< bundleName >$, should first request a peer, whose ID is the closest to $hash(< bundleName >)$, the number of a newest version of the resource. Using received number of the version, the peer sends a request to node with the identifier, the value of which is the nearest to $hash(< bundleName > _ < version >)$.

```
retrieve(hash(<bundleName>));
retrieve(hash(<bundleName>_<version>));
```

The Figure 2 shows an example of search of a bundle with the name PhotoEditorPlugIn. A 12-bit Cyclic Redundancy Check (CRC) function is used as a hash-function. Two peers are involved in the sharing of one component. The peer with $id = \text{FE7}_{16}$ contains only the number of newest component version, and the peer with $id = 540_{16}$ contains the component itself. The peer 23C_{16} receives at first the number of the newest component version an the the component.

As well as in the usual Pastry-networks, it makes sense to use the PAST-algorithm for a data replication. Authors

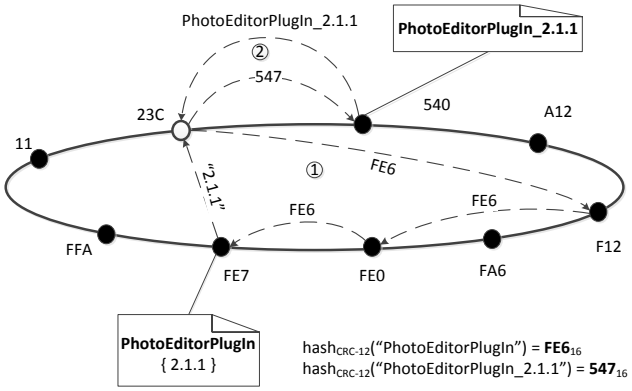


Figure 2. Discovery of bundle in POSGi

implement POSGi on FreePastry and Oscar. POSGi has following advantages and shortcomings.

Benefits of POSGi:

- Simplicity of realisation
- Uses Pastry
- Supports work with bundle versions

Shortcomings of POSGi:

- Does not support functional and non-functional properties
- There is no possible to evaluate components with identical functional, but with various non-functional properties.

3.2 D^{2}CM

Decentralised Dynamic Code Management (D^{2}CM) [11] is a solution based on JXTA [3] platform for the distributed sharing, search, selection and loading of bundles in the p2p networks. D^{2}CM supports discovery and evaluation of the components based on their functional and non-functional properties. Also, packages can be chosen considering compatibility and, if necessary, the dependences of the packages can be resolved. Thus, D^{2}CM supports the processing of several realisations of one component.

Three kinds of advertisements are used for the description of bundles in D^{2}CM: Resource Advertisement (RA), Code Description Advertisement (CDA) Interface Description Advertisement (IDA), which can reference each other. RA represents the resource, IDA announces an existence of component interface, while the CDA describes concrete realisation of the interface and is used for description of non-functional properties.

Thus, a component in D^{2}CM can be localised by its interface name or by its keywords by means of IDA, and based on its non-functional properties by means of CDA.

Special manifests are used for generation of IDA and CDA. Listing from the article [11] shows an example of a CDA manifest in which a printer driver bundle is described.

```
Name:de.uulm.impl.printer
Version:1.3
Interface:de.uulm.Printer
InterfaceVersion:1.0
InternalName:uulmp 1 3

compatibility
{
        os.name::linux
        os.version:version:[2.4;2.6)
}
properties
{
        mode::duplex
        color:boolean:true
}
```

For searching and loading a component considering additional non-functional criteria, the interface name and non-functional properties are used. Non-functional properties can be classified into mandatory and optional properties. Mandatory properties are used directly for the discovery of bundles, and optional properties allow for the evaluation of the successfully found and loaded components, as well as to choose the most suitable to installation. For this purpose, a certain value can be appropriated to each optional property, the values of the executed properties are summarised and the received value is used for the evaluation of bundles.

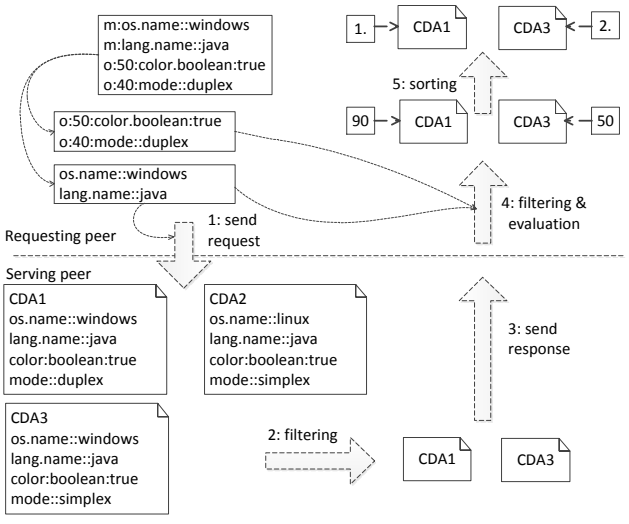


Figure 3. Discovery, loading and evaluation of component in D^{2}CM

The Figure 3 based on the Figure 7 from the article [11] shows the process of the discovery and loading of the component based on non-functional properties. This process consists of five steps:

1. Splitting the properties into mandatory and optional. The mandatory properties are used for request creation. The requesting peer sends the request to the serving peer.
2. The serving peer filters and chooses the bundles that meet the mandatory properties.
3. Chosen CDAs return to the requesting peer.
4. The requesting peer receives and evaluates CDAs and appropriates it values by means of optional properties.
5. The requesting peer sorts CDA by their values and chooses the best CDA.

D^{2}CM has following shortcomings and advantages.
Benefits of D^{2}CM:

- Possible to discover bundles based on their non-functional properties
- Use of optional non-functional properties for the evaluation of components.
- Support of the automatic resolution of dependences.

Disadvantages of D^{2}CM:

- Use of a heavy JXTA platform.
- For evaluation, only optional properties, which are expressed in Boolean values (true or false), are used, while the numerical values are not supported.

3.3 Chord4S

Chord4S [4] is an approach, based on Chord [12] technology, for the distribution and discovery of components or services in p2p networks. A special service identified is used for the description of components or services in Chord4S.

The resource identifier consists of two parts, functional and provider parts. The functional part represents a hierarchy of layers, which describes the functional properties of a bundle. For example, "PlugIn.Image.Editor.SimpleEditor" is a correct identifier. The quantity of levels can vary and is established by the manufacturer of application The second part of service identifier is provided in bits, which are used for the description of the provider-specific information.

Functional and provider parts are hashed by means of a Secure Hash Algorithm (SHA-1) algorithm. Each layer is coded separately and the received bits are concatenated with each other. The formed chain of bits is the resource identifier (see Figure 4 based on Figure 1 from the article [4]).

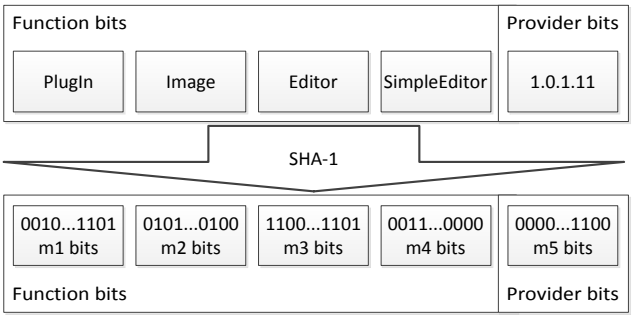


Figure 4. Service identifier in Chord4S

$$\begin{aligned} service\ identifier_m = hash_{m1}(''PlugIn'') \\ +hash_{m2}(''Image'') + hash_{m3}(''Editor'') \\ +hash_{m4}(''SimpleEditor'') + hash_{m5}(''10.0.0.1'') \\ = m1 + m2 + m3 + m4 + m5 \end{aligned}$$

Functional bits are used for resource discovery, while the provider bits allow to distribute the components with identical functionality. The maximum number of functionally identical components that can be distributed in the network is directly proportional to length of provider bits.

In Chord4S components are grouped in virtual segments, which correspond to layers of the component identifier. Figure 5 based on the Figures 2 and 9 from the article [4] shows an external "PlugIn.Image" and an internal "PlugIn.Image.Editor" virtual segment. Components are published in successor nodes which identifier are equal or are nearest to identifiers of components.

Functionally identical components, whose identifiers differ only by provider bits, are located in a virtual segment. The size of provider bits allows to distribute functionally identical components on all nodes of the virtual segment. Thus, the availability of data in case of failure of some nodes remains.

For discovery of the necessary component, the requesting peer generates a target service identifier similar to a usual service identifier. Much like the service identifier, the target service identifier consists of functional bits. The filed which is responsible for provider bits is filled with zero.

$$\begin{aligned} target\ service\ identifier_m = hash_{m1}(''PlugIn'')+ \\ hash_{m2}(''Image'') + hash_{m3}(''Editor'')+ \\ hash_{m4}(''SimpleEditor'') + (0000...0000)_{m5} \end{aligned}$$

Chord4S allows for use of wildcards by query generation. For this purpose one or more layers of functional bits in target service identifier is filled with zero.

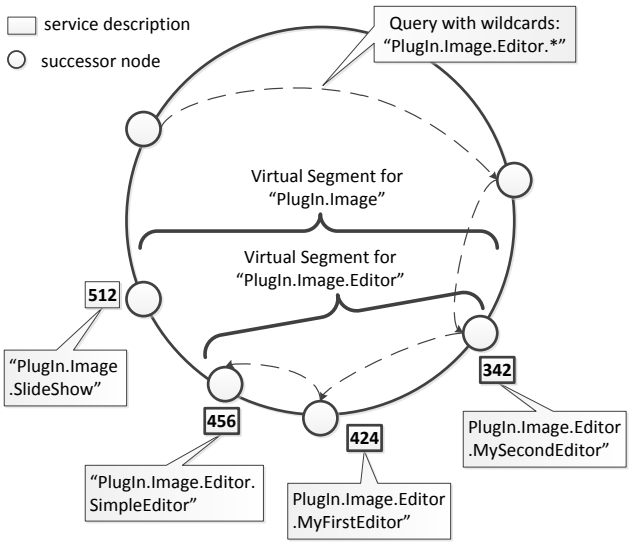


Figure 5. Virtual segments and service discovery in Chord4S

The search of a component the query from consumer peer is routed to all nodes in a virtual segment and thus the system finds all components corresponding to the specified target service identifier. To check the identity of the component to the required service the bit operation AND is used. If the equation

$$target\ service\ identifier AND \\ succeeding\ service\ identifier \\ = target\ service\ identifier$$

is true, means the required component is found. The discovery with wildcards happens similarly. They are used in the service target identifier. The Figure 5 shows a search with the help of wildcards.

Chord4S allows using of the additional Quality of System (QoS) information by discovery and publication of components. The special advertisements containing non-functional properties of components are used to this end. Chord4S supports numerical, boolean and enumerated QoS attributes. The evaluation of components on the basis of non-functional properties begins after the search process of relevant components described above, by component discovery with QoS specification.

Benefits of Chord4S:

- Supports wildcards.
- Supports QoS.
- Supports distribution of additional number of functionally identical components which protect the application from loss of accessible components through node failure.

Disadvantages of Chord4S:

- The Chord4S approach uses specific functions and features of Chord technology and using of Chord4S in other environments may be a heavy or even an impossible problem.

3.4 RSP

Remote Service Sharing (RSP) [5] is a p2p solution for distribution of OSGi services. RSP is based on pure OSGi framework and does not use any other technologies. RSP is a OSGi bundle which allows to connect local OSGi frameworks to remote OSGi and to use services they provide. For this purpose, the possibility of a connection of the local device with remote device through the Internet is required.

RSP allows OSGi framework in the same manner to start and give services without dependence from their location: local or removed. So is reached the transparency by using of remote services.

RSP inform all remote devices about all changes occurring to services, such as registration, unregistration or updating of services. Thus reliability is achieved.

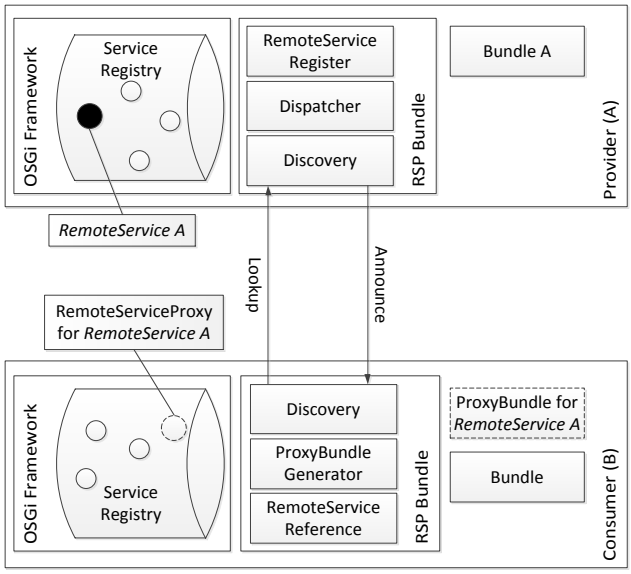


Figure 6. Structure of RSP

The Figure 6 based on the Figure 2 from the article [5] shows the structure of RSP consisting of five parts: RemoteServiceRegister, RemoteServiceReference, Discovery, ProxyBundleGenerator and Dispatcher.

RemoteServiceRegister registers the chosen services as remote in the local OSGi service registry. To this end, a special OSGi service attribute "service.remote" is used, which has the boolean value true for remote services and false for the local.

RemoteServiceReference finds proxy reference on remote OSGi service if such proxy service for required remote service is already registered. If required remote service is not yet registered, RemoteServiceReference waits for the registration of proxy service for remote service.

The basic work in RSP is done by the Discovery mechanism, which is used for the announcement and search of remote OSGi services. Discovery utilises the p2p mechanism and the local OSGi framework with started RSP is used as a peer in this p2p network. If peer wishes to publish some remote service, it uses RemoteServiceRegister for the registration of remote service. Afterwards, the Discovery mechanism sends the multicast message to all peers. This message contains the service name, service identifier, the Internet Protocol (IP) address of publisher peer and number of the port which the Dispatcher service will listen to.

If the peers, having received this multicast message, do not contain bundles waiting for the remote service, they ignore the message. If such bundles exist, Discovery service calls ProxyBundleGenerator mechanism to generate a proxy bundle. Similar multicast messages are sent by any changes in remote service status.

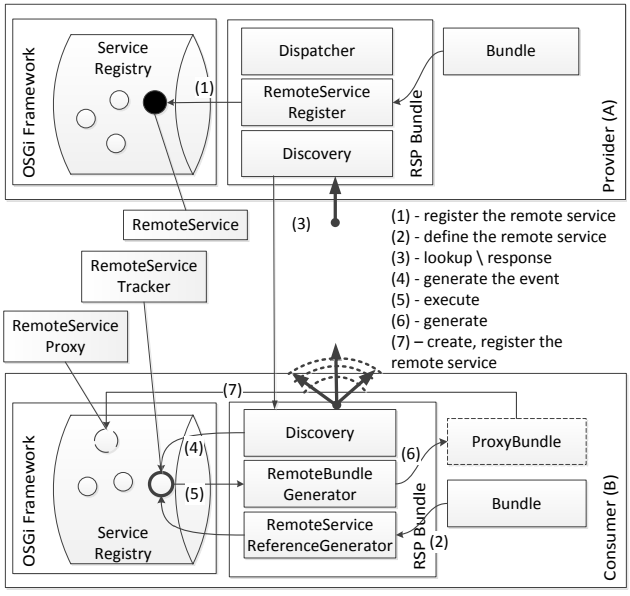


Figure 7. Discovery process in RSP

To search for remote service, RemoteServiceReference uses Discovery mechanism to send multicast queries to network peers. The peer having required remote service uses the special information contained in the received message to call back to the consumer peer by means of the Discovery service. The Figure 7 based on the Figure 3 from the article [5] shows discovery process of remote service.

ProxyBunderGenerator is used for creation of proxy bundles consisting of RemoteServiceProxy services for remote OSGi services.

Requests for the remote services from clients are received and processed by special Dispatcher service which is active continuously.

Benefits of RSP:

- Universality for all systems using OSGi, because of refusal to use of specialised p2p technologies
- Hight Speed
- There is no need to create difficult p2p systems, such as Pastry, Chord or JXTA.

Disadvantages of RSP:

- In the presence of an already organised p2p network based on one of popular technologies, RSP does not make use of that technology.

3.5 Other works

In article provided, the four most well described and powerful approaches are presented.

In addition to the abovementioned approaches, there is a set of other solutions, the majority of which are based on popular p2p technologies Chord (e.g. R-Chord [8], Peer-to-Peer Based Web Service Discovery (PWSD) [7], M-Chord [9]) and JXTA (WebPeer [6]). The approaches based on Pastry are very seldom mentioned in the literatures.

4 Evaluation

From the four solutions described above, two are based on the technologies that are distinct from Pastry: D^2CM is based on JXTA and Chord4S is based on Chord. They use specific features of corresponding technologies and their modification for use in other environments would be impossible or unnecessarily cumbersome.

P2P framework for social networks LifeSocial uses p2p technology Pastry, specifically, its realisation, FreePastry. The plug-ins, used in LifeSocial, represent OSGi bundles. From this follows that the simplest solution would be to use POSGi, which is based on Pastry and allows to load and use OSGi bundles distributed in p2p networks.

POSGi represents a solution for the distribution of OSGi bundles that would be simple in realisation. The basic disadvantage of POSGi is the absence of the possibility of discovery and the selection of components on the basis of functional and non-functional properties. The single piece of information about components which is used by POSGi is the name of the bundle and its version.

However, this restriction is not critical in reference to a problem of distribution and discovery of plug-ins for the single application. If the plug-ins are started on identical platforms (Java Virtual Machine), in identical environments, with identical characteristics and in an identical application (LifeSocial), then the need to distinguish functionally identical plug-ins by their non-functional properties is most likely excessive. At the same time, the support of version management for plug-ins would be useful.

However, there are additional functions that could be useful in a p2p system of distribution and discovery of plug-ins. For example, the Chord4S concept of creation of the component identifier from a hierarchy of layers of functional properties and the feature of wildcards searching can give very powerful additional function. Nevertheless, POSGi can still be used in LifeSocial for distribution and search of plug-ins.

The other solution for LifeSocial can be the RSP approach. However, this leads to some problems and complexities. Firstly, the realisation of RSP as an independent plug-in can be very complex. Secondly, RSP is designed for use of the remote OSGi services, without loading them on the local device. Active use of remote services will load p2p traffic and will depend on stability of connection. Casual fail of successor peer will lead to a crash. For a plug-ins framework it would be much more logical to use such approaches at which the requested plug-in will be loaded only once, and further, used only locally. Thus, after plug-in loading, the local application will not depend on remote devices and quality of connection. Thus, the traffic spent on loading of a plug-in depends only on the size of this plug-in.

5 Conclusions

Four solutions for the distributed storage and discovery of the components and services in p2p networks have been described in this article. We have attempted to choose the most suitable one and one powerful enough to carry out distribution and search of plug-ins for LifeSocial, this solution is POSGi, due to the following two reasons firstly, it based on the p2p Pastry technology, which is used by LifeSocial for organization of the p2p network. Secondly, the functions given from POSGi are sufficient for the creation of the distributed plug-ins store in LifeSocial.

List of Abbreviations

API Application Programming Interface.

CAN Controller Area Network.
CDA Code Description Advertisement.
CRC Cyclic Redundancy Check.

D^{2}CM Decentralised Dynamic Code Management.
DHT Distributed Hash Table.

IDA Interface Description Advertisement.
IP Internet Protocol.

JAR Java Archive.
JXTA Juxtapose.

OBR OSGi Bundle Repository.
OSGi Open Services Gateway Initiative.

PWSD Peer-to-Peer Based Web Service Discovery.

QoS Quality of System.

RA Resource Advertisement.
RSP Remote Service Sharing.

SDK Software Development Kit.
SHA-1 Secure Hash Algorithm.

References

[1] N. Bartlett. *OSGi In Practice*. 2009. DRAFT PREVIEW.

[2] S. Frénot and Y. Royon. Component deployment using a peer-to-peer overlay. In *Component Deployment*, pages 33–36, 2005.

[3] L. Gong. Industry report: Jxta: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, 2001.

[4] Q. He, J. Yan, Y. Yang, R. Kowalczyk, and H. Jin. A decentralized service discovery approach on peer-to-peer network. *IEEE Transactions on Services Computing*, 99(PrePrints), 2011.

[5] E. Kim, K. Yun, and J. Choi. Rsp: A remote osgi service sharing scheme. *Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, 0:318–323, 2009.

[6] R. Li, Z. Zhang, Z. Wang, W. Song, and Z. Lu. Webpeer: A p2p-based system for publishing and discovering web services. In *IEEE SCC*, pages 149–158, 2005.

[7] Y. Li, F. Zou, Z. Wu, and F. Ma. Pwsd: A scalable web service discovery architecture based on peer-to-peer overlay network. In *APWeb*, pages 291–300, 2004.

[8] J. Liu and H. Zhuge. A semantic-based p2p resource organization model r-chord. *Journal of Systems and Software*, 79(11):1619–1631, 2006.

[9] D. Novak and P. Zezula. M-chord: a scalable distributed similarity search structure. In *Infoscale*, page 19, 2006.

[10] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[11] H. Schmidt, J. H. Yip, F. J. Hauck, and R. Kapitza. Decentralised dynamic code management for osgi. In *Proceedings of the 6th workshop on Middleware for network eccentric and mobile applications*, MiNEMA '08, pages 10–14. ACM, 2008.

[12] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[13] The OSGi Alliance. OSGi service platform core specification, release 4.1. http://www.osgi.org/Specifications, 2007.

HTML5 opportunities with Servlets

Marcel Sander
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: jmsander@mail.upb.de

Abstract

In the current version of the peer-to-peer based social network LifeSocial the GUI is implemented on top of the Eclipse Platform. This was convenient because it allowed for fast and simple development while focusing on the peer-to-peer algorithms behind. Now to bring LifeSocial to a more mature version, it is necessary to pay attention to the usability of the GUI. Furthermore, users of the social network may want to participating using mobile devices which are not supported by the Eclipse Platform.

The evolution of web technologies in the last years has shown that it is nowadays possible to create web applications which feel like desktop applications regarding responsiveness and usability. This paper analyzes the advantages and disadvantages that a HTML5 GUI combined with Java Servlets may offer and outlines an architectural approach for the LifeSocial GUI.

1 Introduction

In the last decade online social networks have evolved to platforms where millions of people get in touch with their friends and business contacts. Moreover, they share photos, videos, and other large files with each other. Usually, the software for the social networks is run centralized at a company's datacenter. They are typically implemented as web applications, so that the clients can acccess the social networks with any devices that run a modern *Hypertext Markup Language* (HTML) browser (e.g. laptops, tablets, mobile phones,...).

On the one hand, the offered features simplify and fasten the communication. On the other hand there are some major problems that can be identified because of the fact that social networks are served by single companies:

- The big amount of data has to be processed and stored in huge datacenters. This is expensive and needs a lot of energy.
- Storing personal information or business information at a third-party company results in privacy/security issues.
- The company that runs the social network is a single point of failure. If the company became insolvent, the servers and therefore the social network would be shut down.

The idea of *LifeSocial* is to overcome these issues by creating a social network as a peer-to-peer (P2P) network that runs on the devices of the users. Each user has to provide a small amount of the resources of his device to participate in the network.

1.1 Motivation

At first, as a research project, the development of LifeSocial was focused on the data structures and algorithms that are needed to store the information in the peer-to-peer overlay. The Eclipse Platform was convenient because it offers a simple way to build graphical user interfaces (GUIs). However, it is limited concerning platform-independence (e.g. Android, iOS) and graphical design.

To bring LifeSocial to a level in which it can compete with well established online social networks, it must be able to run on mobile devices and provide better usability. Hence, this paper analyses the possibilities of HTML5 for the GUI that interacts with a Java backend based on the Java Servlet API. Below, the Java backend is referred to as the *server-side* and the HTML frontend is referred to as *client-side*.

The paper is structured as follows: At first it will provide a short history of the web development in Section 1.2 and an overview over the components of modern web applications in Section 1.3. Then it will outline the new features of HTML5 in Section 2 and give a short introduction to server-side development with Java in Section 3. In Section 4 an architectural approach for the LifeSocial GUI will be proposed. Afterwards, occurring security issues are discussed in Section 5. Eventually, a conclusion over the opportunities of a HTML5 GUI for LifeSocial will be given in Section 6.

1.2 History of web development

The descriptive markup language HTML was originally created to interchange and to link text-based documents using the *Hypertext Transfer Protocol* (HTTP). The users were not intended to interact that intensively with these documents as it is possible today. The documents were mostly static, which means they were stored on the web server and sent to the clients upon requests. Later the idea came up to generate these responses on-the-fly and the Common Gateway Interface (CGI) was born. Websites were now able to process the user input and to show dynamic information. This evolution allowed programmers to create web applications like booking sites, forums, and blogs.

The scripting language JavaScript, officially released in 1995 [17], can be embedded in web pages for reacting to user input and for changing the document by the browser itself. Since the technique *Asynchronous JavaScript and XML (AJAX)* has been developed, it is even possible to use JavaScript to make HTTP requests and change the document as a callback to the response. This allows to react to user input by changing parts of the web page without reloading the whole page. For the users this feels a lot better because the UI is still responsible during the HTTP request. An example is Google Maps [5], which loads the current map section in the background while it is still possible to scroll.

Nevertheless, using AJAX, the information displayed in the user's web browser is still a snapshot of the information stored on the server. The snapshot is created after the user made the HTTP request: Imagine you want to book seats for the show on Friday and you are using the offered web application for it, where you can choose the exact places where you want to sit. After you have opened the page, another guest makes a booking. In your browser you will still see these seats as available.

The reason for this is the architecture of the HTTP Protocol that was designed with a request-response paradigm. Each connection must be started by the client side. Apart from several third-party browser plugins, mainly two solutions have been used to overcome this problem:

The first solution is repeatedly polling the web server for changes. A disadvantage of this technique is that the server has to look for changes each time, even if there are no changes. In some scenarios this might trigger some expensive computations. Also, the network overhead for the connections and the "nothing changed" responses might be avoidable.

The second solution is to use a trick, the so-called *Long Polling* [19]. The idea is to make a HTTP request to the server with a very long timeout. The server does not send a response until a change occurred that is relevant for the client. When a change occurred, the response is sent and the client makes a new request afterwards. This way it is possible to simulate pushing changes from the server to the client.

The new version of HTML (5) [8] is currently under heavy development but already partly implemented in many browsers. It supersedes these solutions by supporting a native bidirectional protocol for connections between servers and clients through *Websockets* (Section 2.6). HTML5 also introduces many further enhancements that are described in Section 2.

1.3 Components of modern web applications

In Figure 1 the interaction between the different components of modern web applications is outlined and will be explained in this section. The new features of HTML 5 will not be included here, since they are discussed in Section 2 in detail.

Visiting a website typically begins by the user typing the URL of a HTML document in the browser's address bar. Thus, the browser sends a request of the web page to the specified server: As an example, typing in the URL "http://www.lifesocial.org/index.php" would result in a request of "/index.php" to the server "www.lifesocial.org". As seen in the figure, the request first goes to the web server. There it is decided whether the request has to be processed by the application server. If this is the case, the application server handles the user's input and generates the response. It might also communicate with the data layer, which is often a SQL database, but in the case of LifeSocial the P2P network.

With the first response, the client receives the HTML document that contains the structure and the content of the web page. The HTML document triggers requests to other files including Cascading Style Sheets(CSS), JavaScript, images, audio files, videos, etc.. . These files are mostly static, which means the requests can be directly answered by the web server.

The HTML document is parsed on the client-side and internally represented by the Document Object Model (DOM). The style of the document objects is defined by CSS properties. As already mentioned in the last section, JavaScript is used to make the web page more interactive. It can modify the DOM and react to DOM events. DOM events are e.g. clicking on buttons or pressing keys. Moreover, JavaScript can make HTTP requests by itself and process the responses for manipulating the DOM. In response to these AJAX requests the server may also send the data encoded as XML or JSON, instead of server-side generated HTML. The DOM can then be build up by JavaScript according to the received data.

This ability allows the generation of the web page on the client side, while the data, JavaScript and CSS is provided by the server. Until now, web pages that are fully generated client-side are not widely spread because search

engines like Google do not process JavaScript. Nevertheless, they may offer a better user experience for LifeSocial and are therefore discussed in Section 3 and Section 4.

2 Fundamentals of HTML5

This section will explain some new elements, which came with HTML5, in more detail. HTML is the abbreviation for *Hypertext Markup Language*. As the name says, it is a descriptive markup language for web pages. Beside structuring content, formatting the contained text, and including graphic, audio and video files, it contains hyperlinks to other documents. It can include CSS files that describe how the structure elements and the content should be displayed in the browser, and JavaScript files to provide interactivity. For a more detailed documentation, the reader is referred to the W3C specification [8]. The changes of HTML5 in comparison to the previous version are fully described in [9]. There are also many books that may help getting started with HTML5 like [3].

2.1 Structure and Semantics

HTML5 introduces some new tags to describe the type of the content. Prior to this, parts like the navigation, the header of the web page, the article and the footer were all structured using the same elements. To improve the structure to reveal more semantic and allow better machine-readability, these content parts can now be semantically classified using the following new tags:

article The *article*-element can be used to represent an independent part of the web page, like a news article or a forum post.

section The *section*-element describes thematically related parts of the content. Inside an article you can e.g. differentiate between the introduction, the main part, and the summary.

aside This can be used for additional information that is not necessary to understand the main content of the page (e.g. term definitions).

nav This element defines a navigation area. A page may (and probably should) contain multiple nav elements.

header The *header*-element contains a header of a web page. This does not only include the typical header at the top, containing the logo and primary navigation, but also headings of sections .

These elements are one of the major improvements of HTML5. Although they probably do not offer any practical advantages for the LifeSocial GUI in comparison to previous HTML versions, they are named here to give a better overview over the changes of HTML5. In fact there are some more new elements that can also be found at [9].

2.2 Video, Audio

HTML5 introduces the new media elements *audio* and *video*. Before, the common way to include video and audio files in web pages was using third-party plugins like the *Adoble Flash Player*. Now these media elements provide a native and officially specified approach.

2.3 Canvas / SVG

The canvas and SVG elements allow the client-side creation of graphics. SVG is the abbreviation for *Scalable Vector Graphics* and is a XML-based format for vector graphics that can be used on web pages. The canvas element defines an area on the web page, in which bitmaps can be drawn by JavaScript. In comparison, SVG has the advantages of vector graphics like resolution independence, while canvas is faster in redrawing [23].

2.4 Geolocation

The Geolocation API [16] specifies an interface to the location of the client's device. This may be especially useful for mobile applications like maps or augmented reality, but also for social networks. The interface provides information like the latitude, longitude, altitude, speed and direction, and the accuracy.

2.5 Web Storage

The Web Storage specification [7] introduces two new methods for storing data on the client. They are both similar to HTTP cookies but are not sent to the web server with each request.

The *localStorage* can be used to store a huge amount of structured data on the client. This can be used e.g. for settings that only affect the client-side code, or for the storage of data to achieve a better performance.

The *sessionStorage* is bound to the browser window and not persistent between browsing sessions. It can be used for example for wizard applications. If you open a wizard twice in two seperate windows, you do not want the actions you take in one window to affect the other window.

2.6 Websockets

The HTTP architecture was not designed for bidirectional communication which is needed for modern real-time interactive web applications. The reason for this is the request-response paradigm that is not meant to push information from the server to the clients. Former solutions use long-lived HTTP connections, i.e. keeping a HTTP connection open for server pushes. The misuse of the HTTP protocol comes with several issues related to the HTTP overhead (e.g. sending HTTP headers every request), latency, and resources (e.g. multiple connections per client) [11].

They are now superseded by the new *Websockets API* in terms of performance, bandwidth usage and latency [6] [2]. It specifies native bidirectional connections between the

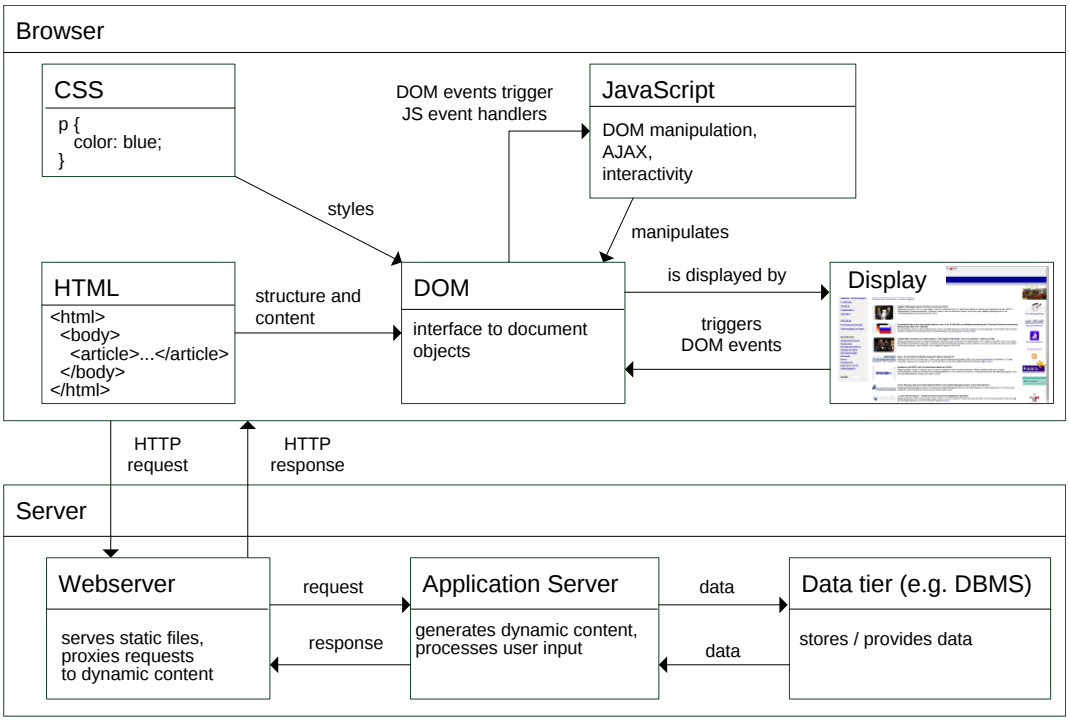


Figure 1. Structure of web applications

browser and the server. Websocket connections start similar to normal HTTP requests. The client performs an HTTP request which proposes the server to upgrade to a Websocket connection. If the server accepts this, the HTTP connection is not closed but used as the Websocket connection afterwards.

2.7 Cross-document messaging

For security reasons, browsers do not allow JavaScript to access objects of other pages from different domains. This is called the *Same-Origin-Policy* [18]. To still allow the communication between different documents, that are e.g. opened in different frames or different windows, a messaging system has been introduced [10]. It allows to send messages to other pages using the *postMessage* function and reacting to messages using an event handler.

3 Java Webservers

The server-side of a web application can be written in many programming languages. Based on the fact that the current LifeSocial version is written in Java, it is obvious that the backend of the HTML5 GUI should also be written in Java. Therefore, this section will introduce Java-based web applications.

Traditionally, in web applications the server generates HTML code upon a client's request. Thereby it doesn't matter whether the request was created using AJAX. The HTML code is transferred to the client and displayed in the browser.

Java web servers, supporting this approach, consist of a *http web server*, a *web container*, and *web components* [13]:

web server The we bserver implements the HTTP protocol. It handles the HTTP requests and communicates with the web container to generate dynamic responses.

web container The web container maps each request to the responsible web component and executes it. The web component's *lifecycle* is managed by the web container.

web component A web component contains the actual generation of the dynamic response. Web components can be *Java Servlets* or *JavaServer Pages*. A Servlet is a Java class that processes a request and generates a response. Also it contains the application logic that communicates with the data tier (e.g. DBMS or P2P-overlay). JavaServer Pages offer an easier way to build text-based responses. They are basically text files that contain some Java code. Technically they are also executed as servlets.

Another approach for generating dynamic websites is to create the HTML code client-side. Therefore the server just needs to send the data to the client in form of XML or

JSON. The client can then process it using JavaScript and create the appropriate HTML.

This approach is naturally very similar to the Model-View-Controller pattern [4]. The view layer is completely executed in the browser using JavaScript. The controllers and models run on the server-side. In addition, based on the actual implementation, the browser needs to update only the elements in the DOM that have really changed. This may provide a more responsive user experience.

In most web applications it is important that the clients are able to read the web page without supporting the execution of JavaScript. The reason for this is that search engines like Google are not supporting JavaScript. Because the turnover of web pages mainly relies on the amount of visitors, they are required to be linked by search engines.

However, for LifeSocial this is not an issue, because the web server runs locally and is not intended to provide a HTTP interface to the internet. Hence, this paper analyses this concept and proposes it for the LifeSocial GUI in Section 4. In more detail, it will use the Websockets API (Section 2.6) because it offers bidirectional communication between the server and the client.

Until now, there is no standard Java API for Websockets but a request has been filled by Oracle [15]. Though, there are already some Java implementations. While the web containers like *Grizzly* [14], *Jetty* [22] and *Tomcat* [21] provide different APIs for using Websockets, there is a layer called *Atmosphere Framework* [1] which can be run in any of these servers. It allows to develop the server-side part of the web application in a way that the server, that executes it, is interchangeable. Additionally, it provides a client-side JavaScript library which abstracts from browser specific issues. If the client doesn't support Websockets, the Atmosphere Framework can automatically fallback to a legacy concept like long-polling.

4 Architecural approach for the GUI

In the following, the previously described technologies will be used in a sketch for the architecture for the new LifeSocial GUI. Therefore in Section 4.1 the requirements for the GUI are collected. Afterwards, the server-side is described in Section 4.2 and the client-side in Section 4.3

Analysing different libraries is not part of this paper because it would require extensive and well explained tests to give scientific arguments.

4.1 Requirements

In contrast to most other web applications, LifeSocial and therefore the web server runs on the device of the user. This results in slightly different technical requirements than traditional web applications.

The web application does not need to be optimized for search engines and not even be able to be read by search engines. Thus, the LifeSocial GUI can fully focus on the usability.

Another point is the idea, that LifeSocial should support plugins. These plugins should be developed by third-party developers. Therefore, it is on the one hand required to offer appropriate interfaces. On the other hand security issues have to be taken into account. The latter will be discussed in Section 5.

The third point, that has to be considered, is that LifeSocial is intended to run on many different devices. This can of course be basically achieved by using HTML and having a backend that runs on these devices. Nevertheless, the layout for small screens (e.g. mobile phones) must be different from the one for larger screens. In addition, input devices like touchscreens may require a different approach than traditional input devices.

In conclusion this results in the following requirements:

- optimization for usability, performance, responsiveness
- support for plugins
- device- and plattform-independence

4.2 Server-Side

The server-side is responsible for talking to the P2P network and for providing an interface for the client-side. Currently the P2P-part is already implemented in combination with a GUI based on the Eclipse Platform. It is thus sufficient to replace the current GUI by the new interface.

A major requirement for social networks is soft realtime, which is important for features like instant messaging or playing online games. In many use cases events will be triggered by other users, sent over the P2P network, and received by the server-side of the user. Such an event has then to be immediately forwarded to the client-side. Since the server-side should not be required to know anything about the client-side, this is only possible if there is an opened communication channel between the client-side and the server-side. Using HTTP, which is the traditional protocol for communication within a web browser, connections are initiated by the client and normally closed after the requested data has been sent. Fortunately, as described in Section 2.6, the Websocket API solves this problem by allowing constant bidirectional connections and is therefore suitable for the needs of LifeSocial.

As a consequence, the server-side must provide a HTTP server for transferring static files to the web browser and implement the Websocket API. Static files include e.g. images, CSS files, and JavaScript files. Also a static HTML document which tells the browser to load a specific JavaScript file must be included. This is explained in detail in the next section.

There are several Java servers which have to be considered for this purpose. Basically looking for a web server and web container with support for Websockets is a good starting point. Further, the decision for a particular software must include the requirement for platform- and device-independence. Some possible examples have been named in Section 3, though they are not analysed within this paper.

Another concern that involves the server-side is the need for supporting plugins. These plugins may be developed by third-party developers and are therefore potentially malicious. This requires some interfaces, whose specification is security-critical. This is discussed in Section 5.

A proposal for an architecture of the server-side is shown in Figure 2. The component *HTTP/Websocket Server* is basically the webserver and webcontainer. Servlets and static files can be provided by *plugins* and the *core controllers*. Core controllers contain basic important features that are an inherant part of the LifeSocial application. Further, they can manage the plugins through the *plugin manager*. In contrast to plugins, the core controllers are not potentially malicious. For simplicity, the *P2P* part of the application is shown as a single component, because it doesn't concern the GUI. It is used by the plugins and controllers for storage and for sending events to other users.

4.3 Client-Side

The client-side is liable for creating the user interface and reacting to user input. Because of the requirement for device- and platform-independence, HTML5 and JavaScript is suitable for this purpose. It can be displayed in most modern web browsers.

Therefore the propose is as follows: A basic HTML file is served by the server-side and opened by the user in the web browser. It refers to a basic JavaScript file, that is immediately loaded. This JavaScript file can load any additional files and modify the DOM. It can hence build up the whole user interface.

There are many JavaScript libraries for using HTML templates which ease the development. Directly building up complete documents would be quite complex in JavaScript and its code would quickly grow up to a size with a counterintuitive structure. Templating libraries allow to write HTML files with placeholders for data and simple programming constructs like loops and conditions.

To talk to the server-side, the Websocket API is used. The server-side provides the data that is inserted in the templates. This data has to be requested through Websockets, which is similar to the traditional HTTP request-response paradigm but avoids the HTTP overhead. If events from the server-side occur, they can be pushed to the browser through the Websocket connection and be handled by JavaScript.

Relating to the introduction of HTML5 in this paper, there are some further new features of HTML which can be advantageous for the LifeSocial GUI. Some examples are:

- The *Geolocation API* can be used to implement features that are especially interesting for mobile devices. For example, a plugin could show all contacts that are near by.
- The *localStorage* can, for example, be used to store settings that are just related to the GUI and do not affect the server-side. They are stored within the browser.
- The audio and video elements can be used for streaming video or audio chats for example.

Because all JavaScript code can generally access the whole DOM, there are some security aspects concerning third-party plugins that are discussed in the next chapter.

5 Security issues concerning plugins

As said before, LifeSocial is supposed to support plugins that may be developed by third-party developers. These plugins will be distributed by the P2P network itself. Obviously, these plugins are potentially malicious. They might, for instance, contain spyware or send spam mails. This issue requires some actions to be taken for prevention.

On the sever-side, the plugin classes must be restricted to using predefined secure interfaces. Technically this can be achieved by implementing a *Java Classloader* to prevent the usage of forbidden classes. This approach can be even extended to implement different permissions for plugins. When the user installs a plugin, LifeSocial could show the permissions, the plugin needs, and require the user to confirm these permissions. This is a similar approach to the one that is used in Android [20].

An example for a secure interface would be the *IGUIFunctionProvider* shown in Figure 2. It is used to provide data, functions, and files for the client-side. The *HTTP/Websocket* component can force each class, that implements this interface, to be mapped to a specific URL. Similar, the interfaces for the P2P network can follow security aspects. As a consequence, the Classloader can restrict the plugins to not being able to use Java's native network classes on their own.

Another more difficult issue concerns the client-side: The JavaScript of the plugins can modify the whole DOM and therefore break the GUI. It can also send any data with the Websocket API to the server-side and hence influence other plugins or the core controllers. The same restriction problem affects the CSS files, because they can define styles not only for the plugin's elements, but for every element in the DOM.

One solution to overcome this problem is to modify the plugin's code on the server-side, such that it can only access specific objects [12]. However, this is a difficult task that possibly oversees some security holes.

Another solution is to embed the plugins using iframes. Iframes are elements in a page which contain another page

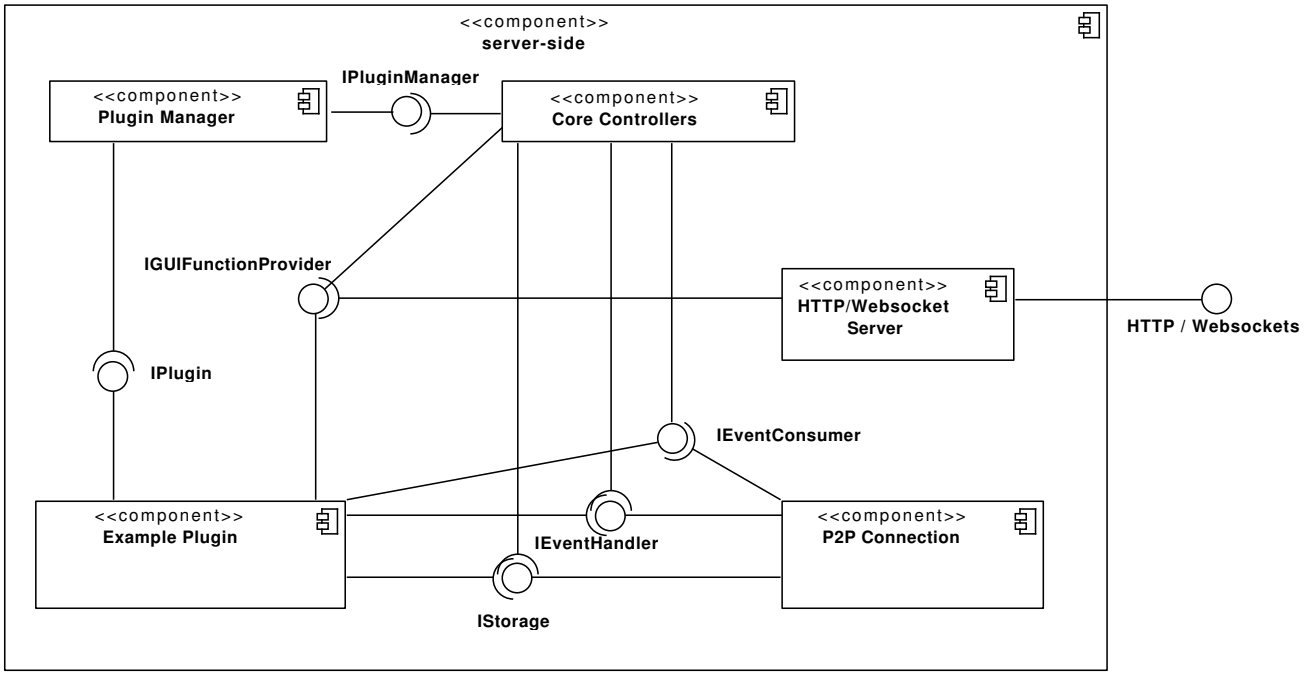


Figure 2. component diagram: server-side

with a different URL. In combination with *cross-document messaging* (Section 2.7) the following approach can be used:

- The *main* page lies at *http://home.lifesocial.org* (which points to localhost). For each plugin, that shall be loaded, it creates an iframe pointing to the plugin's URL.
- Each plugin is available at a different domain (e.g. *Plugin 1* is available at *http://plugin1.lifesocial.org* that also points to localhost).
- The iframes and the main page may communicate using defined interfaces with the postMessage function.

This setup restricts the plugins to their own iframes. They are not able to modify the DOM elements of the main page directly and are not able to include CSS for the main document. Moreover, they can be restricted to their own server-side functions but can though offer interfaces for interaction with other plugins. A basic example how the code could look like is in the appendix.

6 Conclusion

This paper has first given a short history of web applications to explain afterwards what new types of opportunities HTML5 offers. The Java way of implementing the server-side was summarized and the differences between client-side and server-side HTML rendering were discussed.

Eventually an architectural approach for the LifeSocial GUI has been proposed. This approach uses client-side rendering and a HTTP/Websocket server. The security issues that concern plugins can be solved as sketched in Section 5.

As a conclusion, HTML5 is very promising as a GUI for LifeSocial. In respect to the MVC pattern, it allows a clean and natural way of separation between the view and the controllers/models. It also supports the device- and platform independence. On the server-side Java is one of the most famous languages for web applications and there are many libraries that fulfil the needs for the purpose.

As follow-up to this paper, the libraries for the concrete implementation have to be analysed in terms of functionality and device- and platform-independence. Also, the interfaces in the architectural approach need to be specified in more detail.

A Client-side security

This is the example code for safely embedding the plugins in the browser. Setup your web server to serve the file for the main page at the domain *home.lifesocial.org* and the file for the example plugin at the domain *plugin1.lifesocial.org*. Be sure to add the following lines to your hosts file. This is required because there exists no appropriate DNS entry yet.

```
127.0.0.1 home.lifesocial.org
127.0.0.1 plugin1.lifesocial.org
```

Linux: /etc/hosts
Windows: C:\Windows\system32\drivers\etc\hosts

A.1 home.lifesocial.org

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      function parentFunction(msg) {
        alert("parentFunction was called:"+msg);
      }

      window.addEventListener("message",
                              receiveMessage,
                              false);
      function receiveMessage(event)
      {
        //define an interface"
        //for the plugin "plugin1"
        if (event.origin ==
            "http://plugin1.lifesocial.org") {
          var data = event.data;
          if(data.func=="parentFunction") {
            parentFunction(data.param);
          }
        }
      }
    </script>
  </head>
  <body>
    This is the parent frame....
    <iframe
      src="http://plugin1.lifesocial.org/">
    </iframe>
  </body>
</html>
```

A.2 plugin1.lifesocial.org

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      function callWithInterface() {
        //access the main page's interface,
        //only if the main page has the
        //domain home.lifesocial.org
        parent.postMessage(
          {"func":"parentFunction",
           "param":"Plugin1 loaded"},
          "http://home.lifesocial.org");
      }
      function callDirectly() {
        //This will raise a security error:
        parent.parentFunction("Plugin1 loaded");
      }
    </script>
  </head>
  <body>
    This is the plugin frame....
    <button onclick="callWithInterface();">
      Call parentFunction using the interface
    </button>
    <button onclick="callDirectly();">
      Call parentFunction directly
    </button>
  </body>
</html>
```

References

[1] N. Colomer. *Comet/WebSocket? Introducing the Atmosphere framework*. http://www.ncolomer.net/2012/03/-cometwebsocket-introducing-the-atmosphere-framework/, Mar. 2012.

[2] I. Fette and A. Melnikov. *The WebSocket Protocol*. Internet RFC 6455, Dec. 2011.

[3] K. Förster and B. Oggl. *HTML5 Guidelines for Web Developers*. Addison-Wesley, July 2011.

[4] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, USA, Oct. 1994.

[5] Google Inc. *Google Maps*. http://maps.google.com, May 2012.

[6] I. Hickson. *The Web Sockets API*. Candidate Recommendation, W3C, Dec. 2011.

[7] I. Hickson. *Web Storage*. Candidate Recommendation, W3C, Dec. 2011.

[8] I. Hickson. *HTML5 - A vocabulary and associated APIs for HTML and XHTML*. Working Draft, W3C, Mar. 2012.

[9] I. Hickson. *HTML5 differences from HTML4*. Working Draft, W3C, Mar. 2012.

[10] I. Hickson. *HTML5 Web Messaging*. Working Draft, W3C, Mar. 2012.

[11] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. Internet RFC 6202, Apr. 2011.

[12] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. *Caja - Safe active content in sanitized JavaScript*. Technical report, Google Inc., June 2008.

[13] R. Mordani. *JSR-000315 Java Servlet Specification 3.0 Rev a*. Technical report, Oracle America Inc., Feb. 2011.

[14] Oracle America Inc. *Grrrr: Welcome to Project Grizzly*. http://grizzly.java.net/, May 2012.

[15] Oracle America Inc. *JSR-000356 JavaTM API for WebSocket*. Specification request, Feb. 2012.

[16] A. Popescu. *Geolocation API Specification*. W3C Proposed Recommendation, W3C, May 2012.

[17] PR Newswire. *Netscape and Sun announce Javascript*. http://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html, Dec. 1995.

[18] J. Ruderman. *The Same Origin Policy*. http://www-archive.mozilla.org/projects/security/components/same-origin.html, Aug. 2001.

[19] A. Russel. *Comet: Low latency data for browsers*. Mar. 2006.

[20] J. Tam, R. W. Reeder, and S. Schechter. *I'm Allowing What? Disclosing the authority applications demand of users as a condition of installation*. Technical report, Microsoft Research, May 2010.

[21] The Apache Software Foundation. *Apache Tomcat*. http://tomcat.apache.org/, May 2012.

[22] The Eclipse Foundation. *Jetty*. http://www.eclipse.org/jetty/, May 2012.

[23] W3Schools. *HTML5 Canvas vs. SVG*. http://www.w3schools.com/html5/html5_canvas_vs_svg.asp, Apr. 2012.

Sharing large files through BitTorrent

Hongying Zhao
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: hongying@mail.upb.de

Abstract

Peer- to-peer protocols are playing more and more important roles in the Internet file sharing world. BitTorrent is one of the most popular protocols that provide robustness and the resource utilization. Compare with the traditional server-client mode, user can get more efficient transmission rate with BitTorrent. That is why it has started to dominate internet traffic of the world and been widely used. This paper describes the definition and the deployment of BitTorrent, relevant approaches, algorithm and some variations of BitTorrent. As conclusion, there will be some advice for the Project group.

1 Introduction

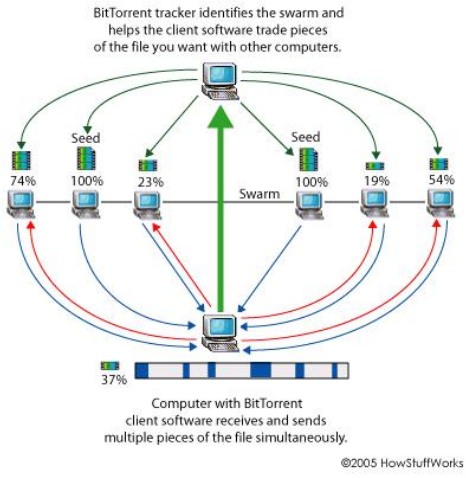


Figure 1. BitTorrent
[3]

1.1 Definition and history

BitTorrent is a file sharing protocol, which is based on peer-to-peer technology. It is used for distributing large files or data over the internet [2]. BitTorrent is designed by Bram Cohen in April 2001. The first release was on July 2. 2001. From 2010 BitTorrent dominates the P2P in most districts of the world, while P2P dominates the upstream traffics. In 2011 from the report of sandvine, BitTorrent dominates both Upstream and downstream traffic in Europe. But in north and South American, Ares and HTTP still dominate the downstream traffic. As of January 2012, BitTorrent has 150 million active users according to BitTorrent, Inc. The monthly users can be more than a quarter billion [9] [2].

Europe, Fixed Access, Peak Period, Top Applications by Bytes

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	59.68%	BitTorrent	21.63%	BitTorrent	28.40%
2	Skype	7.16%	HTTP	20.47%	HTTP	18.08%
3	HTTP	7.02%	YouTube	14.13%	YouTube	11.93%
4	PPStream	3.64%	RTMP	4.58%	RTMP	3.90%
5	Spotify	2.91%	Flash Video	3.99%	Flash Video	3.38%
6	SSL	2.66%	iTunes	3.65%	SSL	3.09%
7	eDonkey	1.76%	SSL	3.18%	iTunes	3.07%
8	YouTube	1.76%	NNTP	2.73%	Skype	2.44%
9	Facebook	1.42%	Facebook	1.71%	NNTP	2.30%
10	Teredo	1.18%	Skype	1.42%	PPStream	1.77%
	Top 10	89.19%	**Top 10**	77.49%	**Top 10**	78.36%

SOURCE: SANDVINE NETWORK DEMOGRAPHICS

sandvine

Figure 2. BitTorrent in europe
[5]

1.2 Strong points of BitTorrent

The normal download hosts provide the download service to every user. So when there is large amount of users, who try to access the server to get the service at same time, the download server will be under very high server impact. The users may lose the service. BitTorrent solves this problem with the new strategy. The download

host (Seeds) distributes the files to 1 or several downloader first, and then the downloader can share the files or chunk of the files to other downloader. This can reduce the server impact. So the Server can host multiple or even large amounts of download requests at the same time, without wasting lots of resource [4].

Secondly, the normal HTTP download service will be provided from the host, and then the downloader start to accept the files they want, that is only download progress. In BitTorrent, the downloaders get the data from the seeds and parallel they share the data to each other, which means the download and upload progresses occur at the same time. It gives a better upload bandwidth utilization [4].

1.3 Challenge

To deploy BitTorrent we must face some difficult problems, both logistical and robustness. Such as how to divide the files in chunks, how large should each chunk be? And some problems, which need more resources to solve them, such as how can peers know, whom their chunks should be sent to and from where they can get the chunks they want. Another problem is the churn rates: not many peer stays online for long time. And finally the fairness problem: User chooses BitTorrent, because they want to download safely and quickly. Mathematically the total download should be equaled to the total upload rate. So if we make the download rate of the peers proportional to the upload rate, all the peers should be happy. But in practice is now so easy to realise. Because we cannot avoid the download rate of a certain user from falling to zero [4]. All the problems will be talked in the following sections.

2 Technical basics

In this chapter. The basical technical content of BitTorrent will be introduced. First is the basical terminology, then the whole BitTorrent will be discussed with detailed explanation of the important algorithm.

2.1 Basic terminology

Torrent
A torrent is the set of peers, which want to download the same file with BitTorrent from the internet [6].

Tracker
A tracker is the server, which collects information of peers and exchange the information it gets to all peers which want to download same file. Help them to connect with each other and transmit data [6].

Pieces (Chunks) and sub pieces
BitTorrent will cut the large files into small pieces (or Chunks), and further to sub pieces. Each piece has SHA-1 hashes. Peers can transmit subpieces, but can share only full pieces with each other after the verifying. A piece can be shared, only if the peer has all the subpieces of it.

.torrent file
We can call it also metainfo file, which contains all necessary information for the download. Such as the amount of pieces, all SHA-1 hashes for the pieces, IP address and port number of the tracker server [6].

Local and remote Peers
The Local peer is the peer which we are standing at. All other peers in the peer set of the local peer are called remote peers.[6]

Leecher and seed
When a peer has already finished the downloading and get the full copy of the file. We can call it seed. All the peers, which do not finish the downloading, are called leecher.

Rarest first algorithm
By piece (Chunk) selection, the peer always chooses the rarest piece in the peer set, is called rarest first algorithm. The peer maintains a list of interested pieces with amount of copies, and always chooses piece with the least copy amount as the next download piece. The list will be refreshed, when there is any peer in the peer set, who announces that it acquired a full piece [6].

Choking algorithm
The choking algorithm is the peer choosing strategy. In BitTorrent client, the peer will reward the best few peers in its peer set for uploading to it. Conversely it will choke the other peers in the peer set temporary. So it will stop uploading to these choked peers, only download from them [6].

Interested and choked
A peer A is insterested from peer B, when peer A has at least 1 piece of data, which peer B does not have. If the pieces of data, which peer B has, is only subset of the data which A has, then we say A is not interested in B. When A does not send any pieces of data to B, we say A choked B. The Choking is only temporary refuse and after the choke, only the upload is stopped, A can still download from B [6].

Favor upload
Favor upload is a approach when choking algorithm combined with seed state. Because as seed a peer does not have any download, the choking algorithm must be modified to suit the seed state. It will always choose few peers with best download rate from him.

2.2 Deployment

To start to use BitTorrent, a .torrent file must be placed on a normal web server, which contains the information of the file, which is to be shared, such as the file name, the file size, the hashing information and the address of the tracker. The tracker is a server, which can help the peers to get the information of other peers. The peers can tell the tracker server information about the files they want via HTTP and get the information about other peers or seeds, which have the files [4]. So first of all, a seed must register on the tracker server, which has a full version of the file. After the registration of the first downloader, it will get the connect information about the seed, so it can start downloading from the seeds. The second downloader can connect to the seed and the first downloader after its registration.

2.3 Cut files into chunks and chunks exchange strategy

In order to help the peers to get better download effects, tracker must know what content each peer has. BitTorrent cuts the files into small pieces with fixed size, normally 0.25 megabyte (256k). Peers must tell other peers which pieces of the file it has, and only if a peer has the whole content of a piece, it can say that it has the piece [4]. It is very important to choose the pieces to download or to upload. A bad algorithm will lead to having all pieces which the peer want in use or cannot upload any piece the peer has to other. BitTorrent cuts the pieces deeper in to sub pieces to get the pipelining effect, in order to avoid the delay between pieces being sent [4]. So first the peers should try to download the sub pieces first to have a full piece. If a sub piece of a certain piece is chosen to be downloaded, other sub pieces of this piece will have higher priority than sub pieces from other pieces. The rarest first algorithm is the most popular for the piece selection now. It means all the peers will choose the pieces, which is the rarest in their peer sets. So the common content will be always put to later download. This can make the whole system in an optimal state. Because if all the peers always try to download the common contents first. It will be easy for them to get these contents at first, but after that all the peers will try to get the rarest content, which makes the system worse and worse. The rarest first algorithm is especially useful to avoid some risk when the seed goes offline: If the downloader peers try to get the rarest contents first, then all the contents on the network will be in balance. So if the seed goos offline by chance, the peers still can share all the pieces from other peers in the network. That can be thought as a strong point on robustness [4][7].

An exception for using the rarest first algorithm is on the initial state. At that time all the piece is rarest in the network or only one or few peers have the rarest pieces. It will waste some time for downloading such pieces. So the peer should choose a random piece to download first. After the download of the first pieces, the rarest first algorithm should be deployed. By end of the download progress, the peer sends request to all the peers for all the sub pieces it does not have. Once sub pieces arrive, a cancel request will be sent to all other peers to make more bandwidth free. Because such period is short, not too much bandwidth will be wasted, and it can avoid from downloading from peers with low transfer rate [4].

2.4 Peers selection

In order to get the optimal download rate, peers must choose the most suitable peers to contract with. BitTorrent uses Choking Algorithm to do that. Peers download from whoever they want, and upload to those via tit-for-tat. So upload to those, who cooperate, and choke other. Choking here is only a temporary refuse and it stops only the uploading, the download works still. So if the choking finished, peers do not need to rebuild the connection [4]. As a temporary refuse, peers will recalculate data receiving rates from all other peers in their peer sets, to choose the peers which to be choked again [6].

2.5 Choking algorithm

BitTorrent does no central resource allocation. So peers respond for the download speed themselves. Each peer in BitTorrent have contract with 4 other peers, problem is which peers it should connect with [4]. In BitTorrent, we use choking algorithm to do it. For each peer, there are 2 different states: as Downloader (Leecher State) or as Seed (Seed State). We now discuss the choking algorithm in these 2 states.

2.5.1 Choking algorithm in leecher state

BitTorrent tries to attempt pareto efficiency using Choking Algorithm [4]. So peers reciprocate other peers which upload much to them with their upload. And the free riders or peers, who do not do much upload, will be choked. In order to keep the download and upload parallel, each peer in BitTorrent will keep connecting with 4 other peers from their peer sets. These 4 peers should be also interested for the local peer, it mean they much have at least 1 piece that the local peer does not have. As we have mentioned, the choking is a temporary refuse. Choke and unchoke within very short time slot will waste resources, so choking algorithm chooses 10 seconds as the period. Every 10 seconds

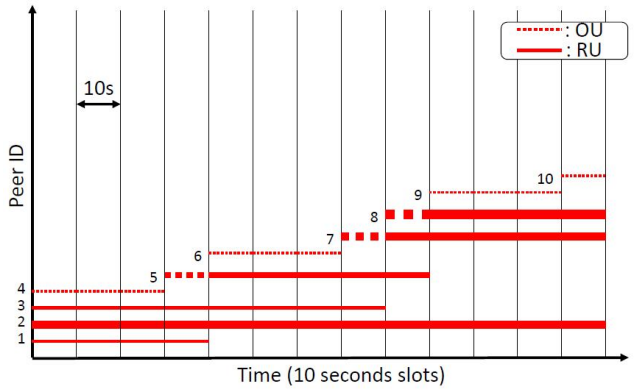


Figure 3. Choking algorithm in leecher state
[5]

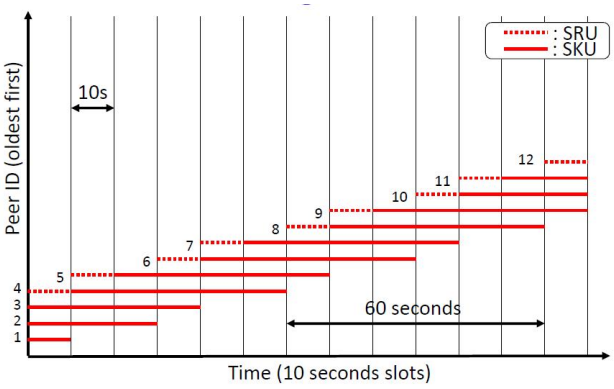


Figure 4. Choking algorithm in seed state
[5]

will all the interested peers in the peer set of local peer will be ordered according to their download rate to the local peer. The 3 fastest peers will be chosen and unchoked until the next period. All the peers, who do not do any contributions, will be ignored. So it guarantees the reciprocating only on peers that contributed. This 10 seconds are long enough for TCP to reach the full capacities of download and upload. Whenever an unchoked and interested peer leaves the peer set, or whenever an unchoked peer switches its interest state, the choking algorithm will be called also. It means the executing time for the choking algorithm may be shorter than the 10 seconds rechoke period [6][4][5].

Problem will be occurred if we just do choking algorithm like that, because the currently unused connections cannot be discovered. To solve this problem, each peer of BitTorrent has an optimistic unchoke every 30 seconds. This optimistic unchoke does not consider the download rate. It is a random choose. And the 30 seconds is long enough for local peer to reciprocate the other peers. If the candidate of the optimistic unchoke is unchoked by the normal unchoke, then the local peer will choose another candidate for the optimistic unchoke, until candidate is found or no more peers can be chosen as candidate[4][6][7][5]. This is shown in Figure 3.

2.5.2 Choking algorithm in seed state

Once a peer completes its download progress, it becomes a seed. And it has no more download rate for the choking algorithm to choose the unchoke candidate. At this situation, we have 2 methods for the peer selection. The old one is the favor upload. It means the peers in the peer set of the seed will be ordered according no more to the download rates, which do not exist anymore for the seed, but according to the data transmission rate from the seed. So the peers with higher download capacity will become the favor because they may do more contribution for the entire network after they get more pieces rapidly from the seed. But problem is, there will be free riders, which download quick and much from the seed while doing no contribution to other peers. Another side, a free rider may get most of pieces and upload bandwidth of the seed because of the high download speed from the seed. Both of the problems hurt the robustness of the algorithm [6][7][5].

A news issue is developed for the new version of the BitTorrent. It uses the similar approach as the leecher state, just with another peer selection criterion. As shown in Figure 4: All from the local peer interested peers are ordered according the time they were last unchoked, also every rechoke period, 10 second. During 2 consecutive rechoke periods, the 3 first peers and a 4th random interested peer are unchoked. After the 2 consecutive periods, the 4 first peers are unchoked [6][7].

According to the new issue, the seeds will serve a leecher for at most 6 rechoke periods. This can avoid the leecher from monopolizing the seed. And it tries to prevent to send the duplicate piece to same leecher before the seed upload the whole version of the file [6][7].

The new version of the choking algorithm is clearly more robust than the old one for free riders. Interestingly, the old version of the choking algorithm is large and widely used nowadays, because the new version might be less efficient for the legal content and the compliant peers [6][7].

Free riding is a difficult problem for all the BitTorrent developers. The result of the experiment shows, limited free riding is possible in BitTorrent under specific situations, but such free ridings do not have severely impact. The free riding peers do not have high download speed than normal.

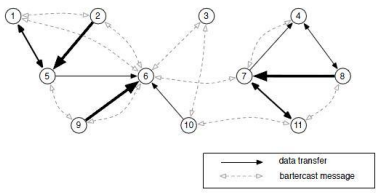
So if peers want to get the highest download rate, they should obey the protocol [6][7].

3 BarterCast

BarterCast is fully distributed mechanism for reputation management. In BarterCast, real-time download and upload information are broadcasted to all the peers who are found with an epidemic protocol. Peers computer each other the reputation based on the information they get from the broadcast. Lies spreading can be limited by applying the maxflow algorithm in computing indirect contribution of one peer to another peer. After that we can use the BitTorrent to distribute the data, with the help of the information (Reputation) from BarterCast [8].

3.1 History

File sharing systems are never stopping trying to find a strategy to reward the users according to their contribution to the file sharing progress. But it is hard to find the reputation. Old version of some software used the mechanism that each peer broadcasts its own upload rate. But its not secure enough, because it is easy to broadcast a false value to others. Some websites use closed BitTorrent trackers to do the reputation management. User need to reach a minimum download-upload ratio to stay in the group who can continue to download. Such system is vulnerable to technical and human failure. So we need to build a secure, full distributed mechanism for reputation management, which is BarterCast [8].



data and BarterCast messages have been exchanged.
The thickness of the data lines indicates the amount of data that has been transferred.

Figure 5. Network with data and BarterCast messages

[8]

3.2 BarterCast protocol

The BarterCast Protocol uses epidemic protocol for discovering unknown peers. All the peers exchange the list of random peers they have and take buddies with all peers they meet. The data transmission rate between local peer and his buddies will be exchanged with all rest peers. This exchange will be realized with BarterCast messages. The statistics in the BarterCast message contains two sorts of information: one is the data transmission rate between local peer and his buddies, the other one is the data transmission rate between the buddies and third party. The BarterCast message from one peer is only the first sort. Both sort of information will together used to build a graph. In order to avoid lies in the statistics, we use maxflow algorithm to calculate the maximum up- and download transfers between peers. The result from maxflow algorithm can be used to calculate the reputation of the peers [8].

$$R_i(j) = \frac{\arctan(\gamma(f_{j\to i} - f_{i\to j}))}{\pi/2}$$

Figure 6. Reputation caculation

[8]

If the result is positive, it means the data flow from j to i is more than the data flow from i to j. So the peer j uploads to I more than it take from i. If the result is negative, it means the data flaw from j to I is less than the data flow from I to j. So j contributes to I less it takes from i. As in figure 7, the result is between -1 and 1. And a peer may have high reputation to one peer but low reputation to another [8].

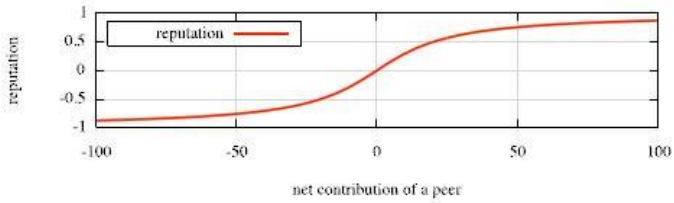


Figure 7. Reputation of a peer

[8]

3.3 Avoid from spreding lies

A peer may lie and exaggerate the contributing it made in order to get higher reputation. The reputation is caculated with the direct and indirect contribution with the maxflow algorithm. The result is bounded by the smallest edge. Figure 8 is the example:[8]

Example 1 in figure 8 shows, if B wants to tell A that it uploaded 1000 to A, and A knows that B uploaded only 10 from the received amount. Exmple B shows, if B wants

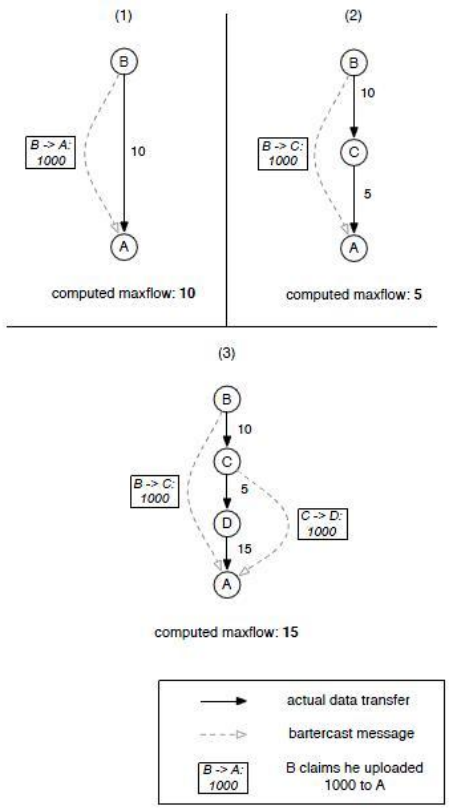


Figure 8. Examples that B lies to get higher reputation

[8]

to tell A that it uploaded 1000 to C, and A knows from maxflow that 1000 is far more larger than the actual upload of B. Example 3 shows, if B wants to tell A that it uploaded 100 to C and C wants to tell A that it uploaded 1000 to D. A gets the maxflow from D is only 15. So B and C cannot lie by A. In summary, a peer can get a higher reputation at another peer only if it did more contribution to this peer or to peers already have high reputation to this peer [8].

3.4 Reputation based BitTorrent

Normally uses BitTorrent tit-for-tat mechanism to for choking algorithm. All the interested peers from the peer set of local peer will be ordered according to their download rate to the local peer. The first 3 best peers will be unchoked until next rechoke period. There will be one random peer every 30 seconds, which is unchoked from the optimistic unchoking. So the new peers have also the possibility to get in to the network before they do any contributions [8].

This tit-for-tat mechanism is only short-term behavior. We can add a new policy to reward the long-term reputation. Like the websites, user can download only if they did enough contributions (Uploads). In reputation based BitTorrent, we setup a threshold with negative value. Only peers, whose reputation is above this threshold can take part in to the download. The negative value of the threshold guaranteed the new peers the possibility to join the download. The low the threshold is, the less reputation will be needed. So lower threshold mean more peers can take part in the download. If a peer joins the download, and it uploads not so much as the download rate, the reputation of this peer will be dropped. If the reputations lower than the threshold, it cannot download anymore. The only way it can download again is to make more upload to earn more reputation. At the time it gets enough reputation, it can start to download again. After the peers complete the download, they can still do some more uploads to earn more reputations for the next download [8].

The reputation based BitTorrent balances the short-term behavior and the long-term reputation. The tit-for-tat behavior guarantees the choking of the peers, which have high reputation but low return bandwidth. The long-term reputation can guarantee the unchoking of the peers, which have low reputation, to join the download again when they earn enough reputations through uploading data. None of both can dominate the protocol [8].

Other side, the reputation based BitTorrent solved the problem of making contribution during downloading successfully. User must do some upload to ensure the download possibility. That is a large improvement than the normal BitTorrent [8].

From figure 9 we can see the reputation of freeriders within the reputation based BitTorrent system decreases over time while the reputation of the sharers increases a little. The decrease of the reputation means the decrease the download rate. We can get the result here, that the reputation based BitTorrent can solve the freeriders problem more or less [8].

4 More implementations of BitTorrent

Amazon S3

Amazon S3 (Simple Storage Service) is an online storage service provided by Amazon. It stores more than 905 billion objects as of April 2012. S3 provids services such as web hosting, image hosting, and storage for backup systems. Every item can be served up as a BitTorrent file. The S3 store can work as a seed for hosting the torrent. User can download the items with lower transmission costs from the S3 server. With BitTorrent user can download from

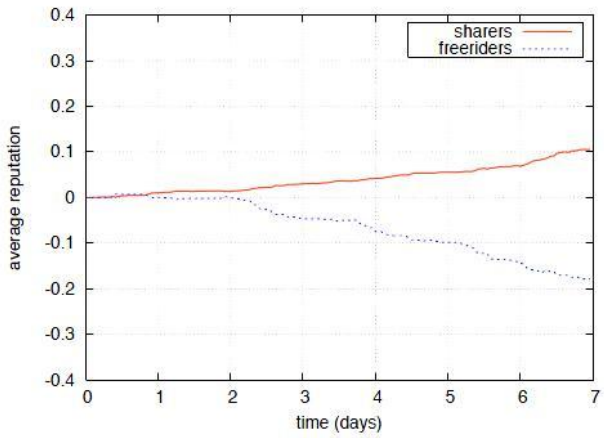


Figure 9. Average reputationover time for sharers and freeriders

[8]

each other. The reduce of the bandwidth and transmission traffic means the reduce of the cost. Therefore is it an improvement than using the traditional server-client mode for data storage [1].

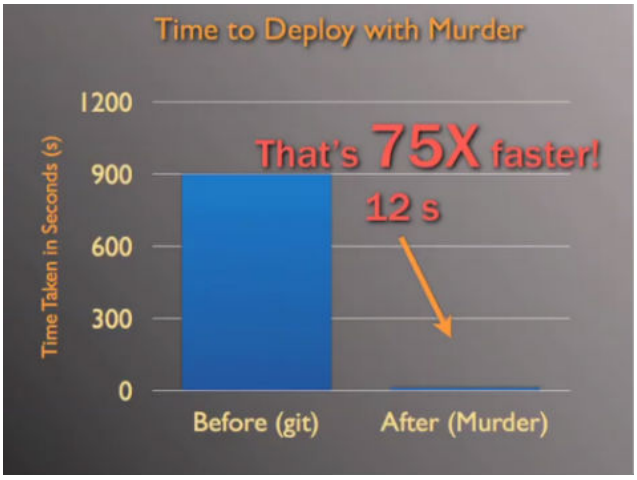


Figure 10. BitTorrent makes server deployment of Twitter 75x faster

[10]

Facebook and Twitter

As one of the largest web services provider, Facebook need thousands of servers to manage the data that sent from its users. Normally, the update can take very long time and problems can also accour. According to Tom Cook of Facebooks systems engineering group, Facebook use BitTorrent-powered distribution system to make daily code updates. With this system, Facebook can send a few hundred MB to tens of thousands of their servers in just one minute. Twitter is doing the same as Facebook, its implementation is based on the BitTornado BitTorrent client. It is open source and under the free software Apache License.It makes each server to become a peer, so the servers not only download, but also upload to other servers. That makes the turned a 40 minute deploy process into one that lasted just 12 seconds [10]. This is shown in figure 10.

Blizzard downloader

Blizzard Downloader is a combination of BitTorrent and http client used for Blizzard Entertainment software products. It is used for downloading the full gaming installer package and the periodic updates for the game. The game client is becoming larger and larger these years, now a game client larger than 3 or 5 GB is easy to find. With BitTorrent the user can quickly download them. The Blizzard Entertainment host many online games also. The server pressure is heavy, so the Blizzard Downloader can save some server and reduce the bandwidth impact. Users will upload some part of the game client to other user while downloading. But for the user, download rate is the most important factor [11].

5 Conclusion

P2P technology is becoming more and more popular for the internet market nowadays. As the most popular protocol, BitTorrent has its improvement. Compare to the traditional server-client mode. BitTorrent has effective robustness and resource utilization. It allows user to parallel download and upload. This download and upload is no more from server, but from seeds and from other peers. That makes the download quickly because server bandwidth is no more a problem. At the same time, with 1 tracker server we can host multiple file sharing progress, because peers do not need much bandwidth from tracker server like before. With strategy such as rarest first, peers can download more efficiently and without the problem, which the seeds goes offline by chance. Choking Algorithm is used to solve the peer selection problems. With the choke strategy, we can solve the fairness problem and make the download more effective

BarterCast is a full distributed reputation system. It is practice feasible and can avoid lying and exploiting. The reputation based BitTorrent has more improvement compared with the normal BitTorrent. It solves the seeder problem by adding reputation in the account to make users do more contribution for the downloading. The balance of short-term behavior and long-term reputation guarantees the system efficiencies. The free riders will be punished by

being lowed the download rate.

In the future, DHT system provides the possibility, which BitTorrent can work without the tracker server. So that one of the bottlenecks of BitTorrent is solved.

Our PG is doing research on P2P social networks. We can learn from the BarterCast and the implementation from Amazon S3, Facebook and Twitter. So we can share the large files with Reputation based BitTorrent. And as the first step, we can make the code update for the server with BitTorrent, when the update is very large.

References

[1] Amazone.com. Amazone Simple Storage Service. 2012.

[2] BitTorrent.com. BitTorrent and uTorrent Software Surpass 150 Million User Milestone; Announce New Consumer Electronics Partnerships. 2012.

[3] C. Carmack and howstuffworks.com. How BitTorrent Works. 2005.

[4] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*. 2003.

[5] A. Legout. Peer-to-PeerApplications FromBitTorrent to Privacy. 2012.

[6] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in bittorrent systems. 2007.

[7] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. ACM, 2006.

[8] M. Meulpolder, J. Pouwelse, D. Epema, and H. Sips. BarterCast: Fully Distributed Sharing-Ratio Enforcement in BitTorrent. 2008.

[9] Pouwelse, Johan, Garbacki, Pawel, Epema, Dick, Sips, and Henk. The bittorrent p2p file-sharing system: Measurements and analysis. In Castro, Miguel, van Renesse, and Robbert, editors, *Peer-to-Peer Systems IV*. Springer Berlin / Heidelberg, 2005.

[10] torrentfreak.com. BitTorrent Makes Twitters Server Deployment 75x Faster. 2012.

[11] wowpedia.org. Blizzard Downloader. 2012.

What does the p2p-framework offer?

Deepak Chandra
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: deepakc@mail.upb.de

Abstract

The technology that enables two or more peers to collaborate spontaneously through appropriate information with the help of communication systems is called peer to peer systems (P2P)[6].peer-to-peer systems became an integrated part of the Internet fabric attracting millions of users. According to recent evaluations, peer-to-peer traffic now exceeds Web traffic, once the dominant traffic on the Internet. While the most popular peer-to-peer applications remain file sharing and content distribution[9]. In Peer-to Peer systems there is no need for servers, but the entire nodes act as one. This paper examines the Peer-to-Peer framework, which is taken form Distributed Framework for Social Networks project group of 2011, which took place under the guidence of Dr.Kalman Graffi. It explains the Interfaces and Commands available in the framework. To do so, the packages in the framework are analyzed one by one, and it also gives you an overview how this Interfaces and Commands are practically used.

1 Introduction

P2P applications organize and manage resources at the edges of the Internet in a decentralized manner, with no interaction with centralized servers. The resources may be storage or content (e.g. file sharing applications), processor cycles (e.g. programs which use your computer to perform part of some large, distributed task when it is idle), or human presence (e.g. instant messaging). The 'edges' of the Internet are machines such as the average home PC which has just a single line (dial-up, DSL(Digital subscriber line), cable, etc.) to connect it to the vast Internet cloud[8].

As remote pc gets the ability to work as client/server sharing content (text, audio and video etc) to distance location, or finding the content from some other location is made possible. The main aspect is about security which becomes major concern, to solve this problem you look for something which is in a peer, which belongs to trusted community of your own network. To get the basic idea of how it works the framework is analyzed by looking the packages available in it, which contains classes and interfaces which help to setup and run the peer-to-peer network. This framework gives higher degree of potential and freedom to its users.

2 A brief look on Classes and Interfaces available in Framework

Before talking about Peer-to-Peer framework we should know what framework is? A framework is a set of code or libraries which provide functionality common to a whole class of applications. While one library will usually provide one specific piece of functionality, frameworks will offer a broader range which is all often used by one type of application. Rather than rewriting commonly used logic, a programmer can leverage a framework which provides often used functionality, limiting the time required to build an application and reducing the possibility of introducing new bugs [1].

This section includes the description about the packages, which shows classes and interfaces available in peer-to-peer framework and explains their functionality. Moreover it shows how to connect to a network which is explained in section 2.2, transfer messages explained in section 2.3, store data using distributed data structures explained in sections 2.4 and 2.8, streaming data is explained in section 2.6. The security mechanisms are described in section 2.5.

2.1 com.p2pframework

com.p2pframework is main package of the peer-to-peer framework, Figure 1. shows this package which has network, message, storage, security, stream, user and dds as its sub packages, it has five classes which helps to configure and manage the framework.

The Framework class holds the entire configuration needed to run the framework and it is also a main management class. An instance need to be created to use the framework and it provides access to all separate modules of the framework. The Configuration class holds needed configuration values for the framework, it contains address and

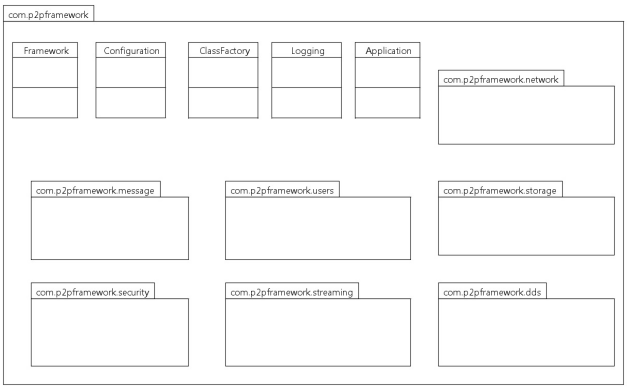


Figure 1. com.p2pframework package

port of the bootstrap host in the network, the local port and address the framework should bind to.

The ClassFactory class is an abstract class which is used to exchange individual modules of the framework, to do so an own ClassFactory class need to be implemented, it's also used to create instances of different management classes of the module, it uses Abstract Factory Pattren and can be configured using Configuration class. Abstract factory Pattren Provides an interface for creating families of related or dependent objects without specifying their concrete classes[11].

The Logging class uses the default Java logging mechanism to manage logging functionality of the framework and it's used by the user to set the log level and write log messages inside the framework.

The Application class is an abstract class which helps user in quick and easy way to use the framework by defining an initial structure to hook into. The Application class is only used for simple applications; if more than simple application is need the Framework class is directly used.

2.2 com.p2pframework.network

The Figure 2. shows the classes available in com.p2pframework.network package.

Network Interface helps you to log in to the network which is already existed, if not it creates a new one so that you can log on if it doesn't exist yet, all parameters need for this are defined in configuration class, just need to provide with the username and password to connect to the network and it also provides an option of log in as anonymous user. Using the username and password provided it creates current ID in the network, which is a unique identifier in the network. An IP is need to use the underlying internet instead of p2p overlay to establish direct connection, its useful if message latency is important, in this case user may want to save the IP in order to go around and re-fetching slowly over p2p network.

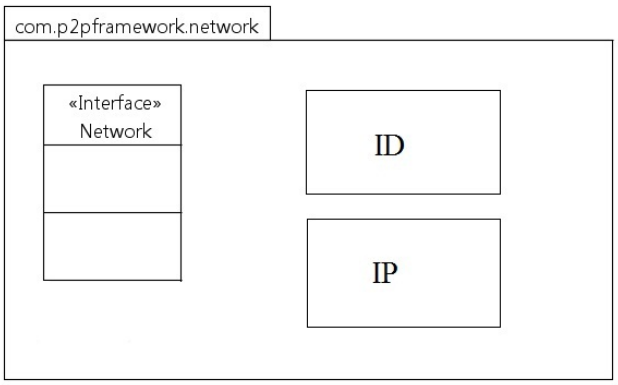


Figure 2. com.p2pframework.network package

2.3 com.p2pframework.message

The Figure 3. shows the class and interface available in com.p2pframework.network package.The main management interface of message package is MessageDispatcher which is used to create different types of channel instances, these channels are used to send message and notifies a listener to register. The three instances are MessageChannel, TopicChannel and AggregateChannel.

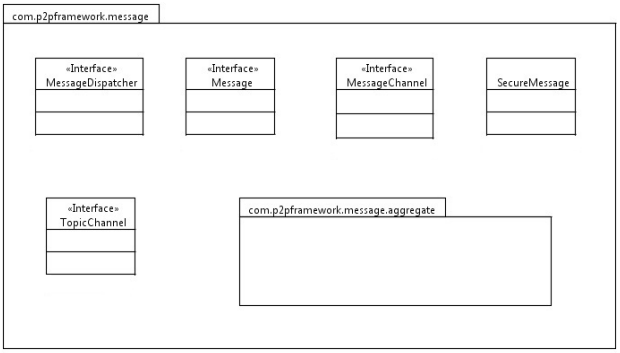


Figure 3. com.p2pframework.message package

The interface MessageChannel is used for one-to-one bidirectional communication it also provides one-to-N communication, it helps to send message to the defined address and also allows to register a listener that get notified on the other hand the TopicChannel interface sends message not only for a single receiver but every node which are registered in the TopicChannel, to accomplish this each TopicChannel is recognized using a unique name and to communicate participants has to create instance of the TopicChannel implementation using the same name and subscribe to

it, after this all subscribers can publish message to the TopicChannel.

The requirements needed to send the message in the network is given by the message interface, the message which is needed to be sent should have a sender and receiver, the user will set the receiver, the sender of a Message is always enforced by the different channels to be the current ID of the user. To send own messages in the network, it is needed to implement this interface. The payload of the Messages is automatically defined by the member variables of the class implementing this interface.The SecureMessage is a class which implements Message interface, this class provides the functionality to sign and secure the message that is sent over the network[3].

2.3.1 com.p2pframework.message.aggregate

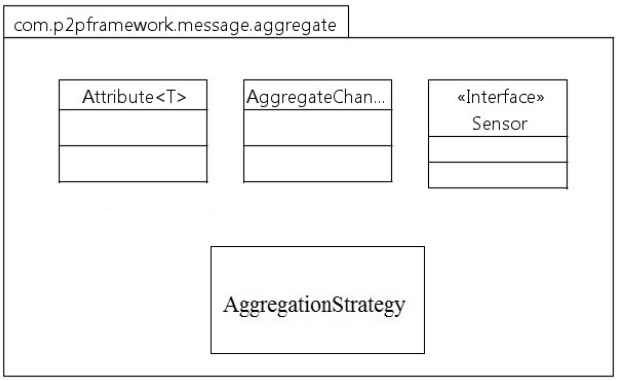


Figure 4. com.p2pframework.message.Aggregate package

The com.p2pframework.message package contains a sub package called com.p2pframework.message.Aggregate, The Figure 4. shows the content of the aggregate package, it has two classes an interface and a enumeration. An enumeration is an object that generates elements one at a time, used for passing through a collection, usually of unknown size[10].

The AggregateChannel class is used to combine the data in the large-scale level, to provide the data it will add the interface Sensor and to receive and combined the pieced data it will register a callback. The interface Sensor provides the attributes to merge all the pieced data over the network. The class Attribute <T >holds a single measurement, its name, unit, aggregationstrategy and the sensor it was measured[3], these are the basic items which are merged over the entire network, here <T>represents attribute data. The main attribute of the AggregationStrategy is to aggregate the data attributes which are isolated over the network.

2.4 com.p2pframework.storage

The Figure 5. gives the classes and interfaces available in the com.p2pframework.storage package.

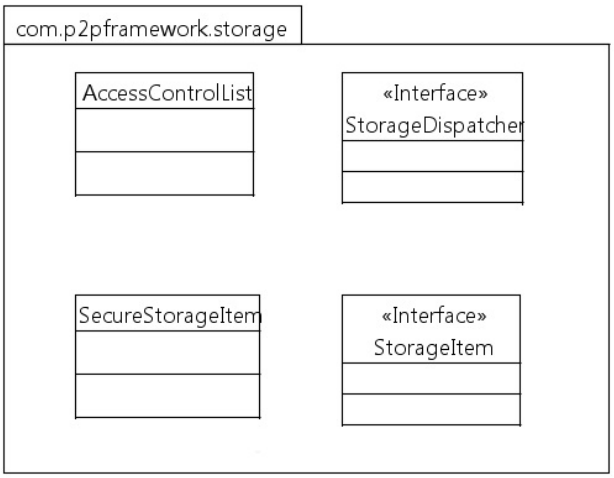


Figure 5. com.p2pframework.storage package

The StorageItem is an interface which is used to store the object in the network, which is implemented by SecureStorageItem class, this object is encrypted using AccessControlList(ACL) which deals with the encryption and decryption mechanism, it also assigns access permission for the stored object which is a reading right to all authentic users, problem is all users can access the object and read, to solve this problem the key which is used to decrypt the object is encrypted with the public key of the original user this means only the user with original decryption key can decrypt.

This object is stored, retrieved and deleted using methods which are provided by StorageDispatcher interface which is implemented by PastryStorageDispatcher class[5].

2.5 com.p2pframework.security

Figure 6. shows the class and interfaces available in com.p2pframework.security package. All the objects sent through the network should be secured via encryption and decryption mechanism, to do so Peer-to-Peer framework makes use of Elliptic curve and AES cryptography approach[2]. The various bytes of serialized object which is encrypted and decrypted or signed and verified using AsymmetricKey and SymmetricKey interfaces which are implemented by KeyFactory class which is a main management class which creates instences using factory pattern[11].

2.6 com.p2pframework.streaming

The classes available in the com.p2pframework.stream package are used to transmit either large or small chunk of

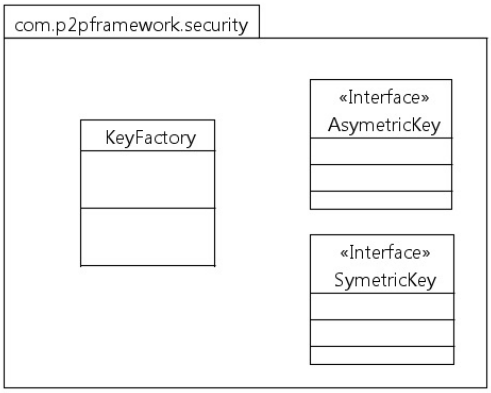


Figure 6. com.p2pframework.security package

data over the network, there are two kinds of transfer one is live stream data which sends stream of Bytes or Arrays, the second type is to send whole data file through the network at once, to look after these processes some listener are provided. Figure 7. shows the classes and interfaces available in the com.p2pframework.stream package.

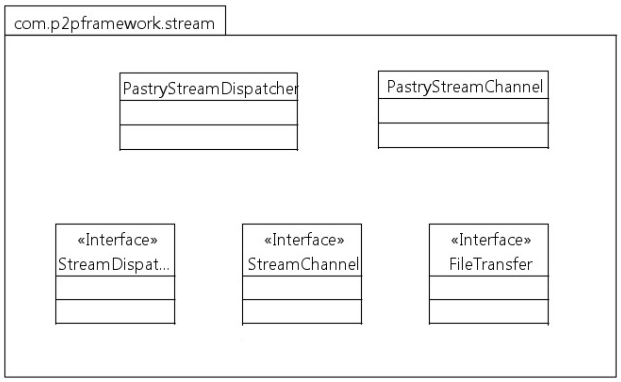


Figure 7. com.p2pframework.streaming package

The main management interface is StreamDispatcher which is implemented by StreamDispatcher class which provides further function to retrieve data form other nodes. StreamChannel is used to send and receive the data from the specified stream which it represents, another advantage of StreamChannel is it can adapt to the bandwidth of the network condition and retransmit the lost package.

A file transfer over the network is represented by FileTransfer interface, which manages the transfer and informs listener about the transfer status. The FileTransferListener is an interface which listens to the file transfer and gets back with the information about statues changes, bandwidth and timing etc, it can be used in both sender and receiver side, like wise the FileTransferListener the StreamListener interface is used to get statues information about the stream, even a second changes in the stream are sent.

2.7 com.p2pframework.users

Figure8.shows the classes available in the com.p2pframework.users package.

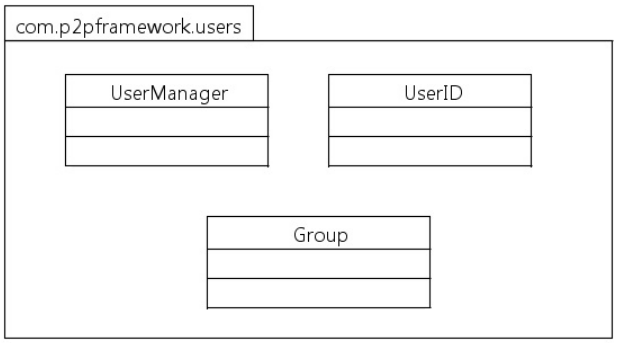


Figure 8. com.p2pframework.users package

The UserManager class is used to create and modify the user and group related information, it gives methods to map between UserID, NodeID and username, every user in a network should have unique identification number and the message to be sent should have valid UserID and it should be verified. This absolute ID of the user is created by UserManager, even if u change username and password UserID won't change, Like wise every node in the network has its own NodeID. Group class holds the information of users that forms a group, the group is updated when a member leaves or new member joins the group

2.8 com.p2pframework.dds

Before diving in to Distributed Data Structures, we should know what is a Data Structure? A data structure is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms[7]. But distributed data structures in peer-to-peer network are spread over many nodes to scale well with large content.

Figure 9. shows the classes available in the com.p2pframework.dds framework.

Peer-to-Peer framework has three Distributed data structures which are DistributedSet<E>, PrefixHashTree<E>and DistributedLinkedList<E>, which

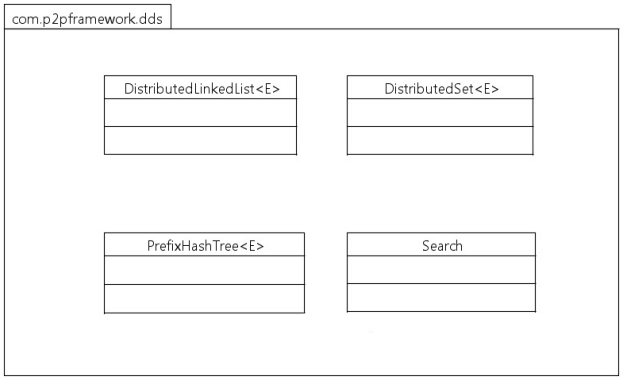


Figure 9. com.p2pframework.dds package

are used as the local data structure. In fact, they implement the known interface of their local pendants[3].

The DistributedSet class is used to represent the all class set in distributed manner, also here, almost all methods of set interface are implemented, which are used just like local set. But the DistributedLinkedSet class has class list in distributed manner; this list can be used like a local list. The PrefixHashTree class is used to store and retrieve the key-value pairs, in a given key range its efficiently retrieves all the key values needed.

The Search class holds all the search methods given by framework, by using these classes it's much easier to carry out search operation than through the distributed data structures

3 The practical usages of the classes and interfaces

The practical usage gives you an idea of how the peer-to-peer framework code is used to setup and run the peer-to-peer network.

3.1 How to Initialize, Connect and Send Message using the code avilable in the Framework

The Table 1 shows the code for how to initialize, connect and send message in a network, if the network is already exists, to connect for that particular network you just need to initialize the framework and its quite simple, only instance of Framework class and Configuration class is needed to do so.

After initializing in an existing network it configures network specific parameters like the address and port of the bootstrap, to create new network address of current node should defined as bootstrap address and it should provide username and password to login for the network. In the code you can see instance of the network is retrieved form the framework which is used to establish the connection,

1. code for initialization

```
Configuration config = new Configuration();
Framework framework = new Frameworke(config);
```

2. code for connect

```
config.setBootstrapAddress ( ”localhost” ) ;
config.setBootPort (9001) ;
Network network = framework . getNetwork ( ) ;
try  network . connect ( ”MyUser ” , ”MyPassword ”) ;
catch ( ConnectionFailedException e )
catch ( UserNotRegisteredException e )
```

3. code for send message

```
MessageDispatcher msgDisp = framework.
getMessageDispatcher( );
MessageChannel msgChan =
msgDisp.createMessageChannel(”MyMessages”);
Message msg = new GenericMessage
<String>(”Hello World”);
msg.setReceiver(framework.getNetwork()
.getOwnID( ));
msgChan.sendMessage(msg);
```

Table 1. Shows how to Initialize, Connect and Send Message[3]

this connect method takes username and password as argument and throws exception, which are ConnectionFailedException and UserNotRegisteredException.

To send message in the network MessageChannel is created using MessageDispatcher, to do so it retrieves an instance of the MessageDispatcher form the framework and creates MessageChannel using the string ”MyMessage” as an identifier for the class, the message itself is represented by a class that implements the message interface

3.2 How to receive a Message, Store and Retrieve data in the Network.

The Table 2 shows how to receive message, store and retrieve data.To receive the message in the network MessageChannel is used, the registered callback is called every time messages arrive. To receive message an instance of the MessageDispatcher is retrieved from the framework, and it creates a new instance of MessageChannel, the identifiers are same as in the send message but an additional parameter is given which implements MessageCallback interface. The StorageDispatcher is used to store and retrieve the data in the network, to do so an instance of it is taken from the

1. code for receive message

```
MessageDispatcher msgDisp = framework.
getMessageDispatcher( );
MessageChannel msgChan =
msgDisp.createMessageChannel(”MyMessages”,new
MessageCallback({
public void receiveMessage
(Message message){
System.out.println(message);}});
```

2. code for store data

```
StorageDispatcherstorage= framework
.getStorageDispatcher( );
StorageItem item = new GenericStorageItem
<String >(”Hello World”);
try{
storage.storeItem(item,”MyItem”);}
catch (StoringFaileException e ) {}
```

3. code for retrive data

```
StorageDispatcher storage = framework
.getStorageDispatcher( );
try
{
StorageItem item = storage.retrieve Item(”MyItem”);
}
catch(ItemNotFoundException e){}
```

Table 2. Shows how Receive Message, Store and Retrive data[3]

framework, a complete class can be stored in the network and can be retrieved again, and a predefined generic class is needed and an identifier to store and retrieve the data.

3.3 How to use Distributed Data Structures

The Table 3 shows how to use Distributed data structure specially Distributed Linked List (DLL) and also shows how to use Prefix Hash Tree.

The DLL is created using rights management, so a group with name is created and it's automatically admin. While creating DLL the data structure which is managed under the group will be given as last parameter, other parameters are the bucket size is defined and the name of the data structure[3]. The DLL which is created under the same name means all will have the same structure, they are kept consistent between all nodes, and all the methods can be

1. code for creation of distributed linked list

```
int bucket = 10;
String group = ”Groupi”;
UserManager um = framework.getUserManager( );
um.createGroup(group);
DistributedLinkedList<String>list =
new DistributedLinkedList<String >
(framework,name,bucket,group);
list.add(”2”);
if(list.contains(”2”))
System.out.println(list.size( ));
```

2. code to use the created distributed linked list

```
DistributedLinkedList<String>list= // [ . . ]
Collection<String>localList = // [ . . ]
list.add(localList);
Iterator <String>i = list.iterator( );
while(i.hasNext( )) {
String lala = i.next( );
if(lala.equals(”hit”))
i.remove( );}
String arr[ ] = list.toArray( );
for(String item:arr)
System.out.printLn(item) ;
```

Table 3. Shows how Create and use DLL, how to Stream and to use Prefix Hash Tree[3]

used like a local Java list, but it supports more than Java List interface. The table shows the advance cases, one is adding of LocalList which is very important method. An item after item is retrieved using the iterator. Internally the items get loaded in whole buckets. The iterator supports, as shown in table, the removal of the last item retrieved. Shows the creation of an array out of all elements of the DLL. As mentioned before the iterator is better suited for retrieval because it only requests as many items as needed. All items of a big collection can not be possibly shown in the GUI. Thus it is not advised on big collections[3].

The Prefix Hash Tree[4] stores keys and key value pairs which allow range search over its keys, to do so an alphabet which defines the possible keys is created, which is an vector of characters, this characters order is also the same order which is used to find greater or smaller keys in the key range. The Table 4 shows the bucket size of 10 items and a binary alphabet which is used to create Prefix Hash Table, here PHT stores integer values after that a toggle for lookup table plan is switched to parallel lookup which is latency friendly but utilizes more throughput.

The key should be represented using long string, it's very important because as they read the little endian with the most significant bit left. The key would get larger number if the starting zeros are omitted. It's not a problem to start with the large key space.

The range search is done to get the key value in between the range, then these searched keys are omitted and stored in the collection. But the iterator is better solution to retrieve large number of items, the Table 4 shows the key range which is same for iterator and getRange, this iterator is then used to retrieve the pairs and check for item with the given value, in the code table the value is 51 which is deleted afterwards. Another important thing is, there can be many values associated with one keys.

1. usage of prefix hash tree

```
int bucket = 10;
Vector<Character>alphabet = new Vector <Character >();
for(int i =0; i <2; i++)
alphabet.add((" "+i).charAt(0));
PrefixHashTree<Integer>tree =
new PrefixHashTree<Integer>(framework, name,
bucket, alphabet);
tree.setOptimizeLookupForLatenzy(true);


PrefixHashTree¡Integer¿tree= // [ . . ]
tree.add("00001",51);
Collection<Integer>d = tree.getRange
("00000","00010").values( );
Iterator<Entry<String,Integer>>it =
tree.getRange Iterator("00000","00010");
while(it.hasNext( )){
int value = it.next( ).getValue( );
if(value = 51)
it.remove( );
}
```

Table 4. Shows how to use Prefix Hash Tree[3]

3.4 How to use Streaming

The Table 5 shows the code for creating streaming channel, sending data and receiving the stream. Two arguments the name of the channel and the number of stripes to use are needed for the StreamDispatcher. After this some configuration technique is accepted by the channel where the maximum bandwidth to use, stripes redundancy and use checksum can be set, redundancy gives the number of stripes sending the same data and checksum get sent with the data such that the stream has a small amount of forward correction.

1.code for streaming

```
StreamDispatcher dispatcher = framework( )
.getStreamDispatcher( );
StreamChannel channel =dispatcher.createStreamChannel
("jackieChan",4);
channel.setBandwidth(500);
channel.setRedundancy ( 2 ) ;
channel.setUseChecksum(false);
```

2. code to stream data from a file

```
StreamChannel channel = [ . .];
BufferedInputStream in =
new BufferedInputStream(new FileInputStream(path));
channel.connectBufferedInputStream(in);
channel.startPublish( );
channel.setBandwidth(channel.getBandwidth( )+10);
```

3. code to receive data from a file

```
StreamChannel channel =[ . .];
BufferedOutputStream out =
new BufferedOutputStream(new FileOutputStream(path));
channel.connectBufferedOutputStream(out);
```

Table 5. Shows how to use Streaming[3]

After creating the stream channel intended data can be sent, code in the table 5 shows how to use channel for sending data. BufferInputStream is best suited method for this purpose, to do so it just has to be connected to the channel. The data to be transmitted through this connection is defined, the only thing remaining is starting the transmission with the startPublish( ) method. Just like the sender the receiving side also should create the StreamChannel, but here it has to connect to a BufferOutputStream, The received data gets buffered and written into this stream. The Table 5 also shows how to save the streaming data directly to a file[3].

4 Conclusion

This paper shows the power and opportunities offered by the P2P(Peer-to-Peer) framework. To show full potential of P2P Framework an overview on the Interfaces and Commands available in the framework are given and analyzed. All sections provide the main packages needed to setup and run the P2P framework in the network successfully, and explain the classes with methods, these classes implements many interfaces, it also gives some functionality to connect to the existing network and to send the chunk of data and

store and retrieve to all nodes in very secure manner. The P2P framework offers many suitable features that easy the development of a social network. It can be easily extended by implementing more distributed data structures if needed.

References

[1] DocForge Website. http://docforge.com/wiki/Framework (AccessDate:27-05-2012).

[2] J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. On the security of 1024-bit rsa and 160-bit elliptic curve cryptography. Cryptology ePrint Archive, Report 2009/389, 2009. http://eprint.iacr.org/.

[3] K. Graffi. A Distributed Framework for Social Networks. Technical report, Universitat Paderborn, 2012.

[4] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief announcement: prefix hash tree. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, New York, NY, USA, 2004. ACM.

[5] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems., 2001.

[6] P. D. D. Schoder. peer to peer systems. http://www.wim.uni-koeln.de/Social-Media-Web-Science-Social-Network-Analysis.441.0.html?L=1.

[7] searchsqlserver.techtarget.com. http://searchsqlserver.techtarget.com/definition/data-structure(AccessDate:07-06-2012).

[8] C. Shirky. http://cs.berry.edu/ nhamid/p2p/(AccessDate:07-06-2012).

[9] springer website. http://www.springer.com//communication +networks/book/978-3-540-29192-3(AccessDate:25-05-2012).

[10] tech-recipes website. http://www.tech-recipes.com/rx/1181 /java-enumeration-and-iterators/ (AccessDate:15-05-2012).

[11] J. M. Vlissides. reference on the topic, is design patterns: Elements of reusable object-oriented software.

Best practices for routing in p2p networks

Chintan Parekh
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Distributed Framework for Social Networks
Email: cjparekh@mail.upb.de

Abstract

Peer-to-peer (P2P) network based on distributed hash tables (DHTs) have gained a considerable importance since the last decade. Any DHT based overlay could be identified by the way it performs the routing operations. Routing in any DHT based network can be defined as fault less transfer of message from the source node to destination node. Routing can be characterized by three different parameters: overhead, latency and security. DHTs have come up with a balance between overhead and latency but unfortunately routing in a DHT based network is prone to attack by malicious nodes in various ways. This paper discusses some of the best practices proposed by various authors for fault free routing. We review the advantages and disadvantages of all the approaches discussed and try to propose a collaborative unique solution which could be implemented by ***FreePastry****, a prototypical implementation of Pastry to perform routing under adversary condition.*

1 Introduction

P2P based applications have grown tremendously. Almost half of the current internet population uses P2P system in their regular life. The popularity of P2P system is mainly because of its core feature, direct communication without any mediator. P2P networks can be broadly classified into two major areas, Structured and Unstructured Overlays.

1. Unstructured P2P Overlays: Every node here is responsible only for its own item. Due to its simple structure it gives us an advantage of performing complex search in the network but performance decreases with the growth of network.

2. Structured P2P Overlays: It can be defined as the one, where every node and object has a unique identifier with every node responsible for a given set of objects. Due to a concrete distributed structure lookup time could be bounded and network could be extended for many nodes.

The use of unstructured and structured overlay depends totally on the problem statement with the former offering the advantage of complex search and latter offering the strength to support more nodes exclusively. Our major concern here would be the real world scenarios where we have a huge pool of nodes with both legitimate and adversary participants. From now on we consider only structured overlay networks in this paper.
Structured overlay networks are basically implemented with the help of DHT with every node having information about a set of other nodes and hence the whole network. The major reason for the success of DHT is due to a bounded lookup time and hence reduced latency. Routing in any DHT based network like Chord [10] and Pastry[9] is performed using the local routing tables. However, in a real world network both legitimate and adversary nodes exists and hence performing routing securely becomes a major concern. Establishing a secure routing protocol is a process with 3 important constituents: Secure nodeId assignment, routing table maintenance and fault free message forwarding.

1.1 Important Definitions

1. Routing: A process of transfer of message from the source node to destination node can be defined as routing.

2. Latency: It can be defined as the overall delay in the successful transfer of message.

3. Overhead: The average amount of information that every node has to store can be defined as overhead.

4. Malicious Node: A node which circumvents the correct delivery of the message can be defined as a malicious node.

5. Legitimate Node: An honest node in the network which tries to do some work.

6. Colluding Nodes: A group of malicious nodes may come to a secret understanding to launch attack on the

network. Such a group of nodes can be termed as Colluding Nodes.

7. Secure Routing: A secure protocol can be defined as a protocol which performs routing efficiently with high success ratio in the presence of a fraction f of malicious nodes.

1.2 Organization

The organization of this paper is as follows: In section 2 we discuss the routing mechanism used by Chord and Pastry. This section lays down the basic understanding of how the routing is done in the absence of malicious node. Section 3 discusses some of the most prominent attacks on P2P networks and solutions to those attacks as proposed by various authors. In the last section we study a collaborative unique solution which could be implemented by **FreePastry**, a prototypical implementation of Pastry to perform routing under adversary condition. Lastly we conclude our paper with some review on the related work and then summarize the paper.

2 Routing Approaches

Structured P2P overlays implement routing using DHT but every protocol has its own strategy to organize nodes and perform the routing. In this section we understand some of the basic concepts used by popular overlays like Chord and Pastry which would be used in consecutive sections to understand the attacks and the offered solutions.

2.1 CHORD

2.1.1 Node State and Routing Table Organization

Chord organizes its node in a virtual ring like structure. Every node in Chord maintains a finger table of fixed size, $\lceil log_2 n \rceil$ such that every node points to the node succeeding it in the circle by the following formula.

$$n.f(i) = (2^{i-1} + n) \mod N$$

$$n.f'(i) = Successor(n.f(i)) \quad when, f(i)! = active$$

$$n.f'(i) = n.f(i) \quad when, f(i) == active$$

Where, $n.f'(i)$ and $n.f(i)$ represents the finger it should point and finger it is pointing respectively and N represents the total number of nodes in the system. Figure 1 shows how node 4 organizes its finger table.

2.1.2 Routing

Routing in Chord is done by a recursive process where every node forwards to the closest node to the target from its local finger table. For example, in figure 1, a lookup request for 52 at 4 is established by following steps: 4 to 36 , 36 to 54(as 54 is the immediate active successor after 52).

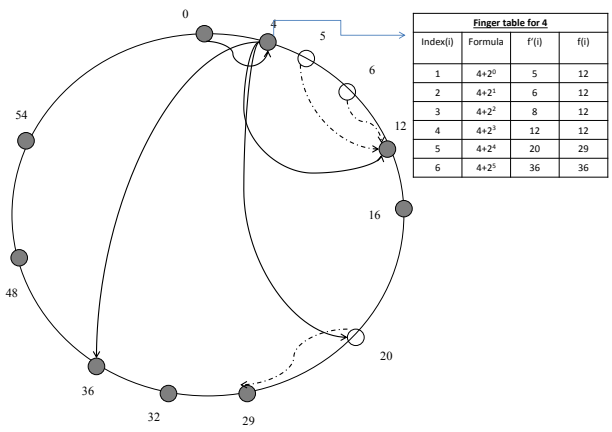


Figure 1. Chord Finger Table (Notation: Dark nodes and Light nodes are present and absent respectively)

2.2 Pastry

2.2.1 Node State and Routing Table Organization

In Pastry every node maintains a leaf set, a neighborhood set and a routing table. The size and number of hops depends on the programmable parameter b also called as base parameter.

1. Routing Table: Routing table is basically organized into $\lceil log_{2^b} N \rceil$ rows with $2^b - 1$ entries in each. A cell (i, j) shares the first i digits with the current node n and the $(i+1)^{st}$ digit represents the column number j. Hence each entry in the routing table is identified by a potential node which shares the appropriate prefix.

2. Leaf Set: It contains a set of numerically closest nodes with fraction $\frac{|L|}{2}$ numerically higher and fraction $\frac{|L|}{2}$ numerically smaller value where $|L|$ is usually 2^b or $2 * 2^b$.

3. Neighborhood Set: It contains $|M|$ geographically closest node where $|M|$ is usually 2^b or $2 * 2^b$.(Used to maintain the locality information and unlikely to be used for routing.)

Figure 2 shows the routing table together with the leaf and neighborhood set for clear understanding.

2.2.2 Routing

Pastry basically performs a prefix based routing. A request at any node follows 3 simple steps, leaf set, routing table and if none then returns the current node if it is numerically closest. Hence, if the request lies in the scope of leaf

NodeID 12030201			
Leaf Set			
12030202	12030213	12030233	12030301
12030003	12030013	12030123	12030130
Routing Table			
-0-1203020	1	2-1203232	3-12322323
1-0-203022	1-1-223322	2	1-3-310230
0	12-1-22112	12-2-13222	-
	120-1-1212		3
0	1203-1-121	1203-2-111	1203-3-210
12030-0-03	12030-1-23	2	12030-3-01
0	120302-1-3	120302-2-2	120303-3-1
	1		
Neighborhood Set			
13020322	21300111	23102103	32321010
11020301	00231222	12032302	10203010

Figure 2. State of a hypothetical Pastry node with nodeId 12030201, b = 2, and l = 8. All numbers are in base 4. The top row of the routing table is row zero.

set it replies immediately by forwarding it to the respective node or else it forwards the message to the routing table and performs prefix based routing to the node which is the most closest(if none then the current node is the destination). In reality routing in Pastry is performed even less than $\lceil log_{2^b} N \rceil$ hops due to its organization. Consider figure 2, assuming that the same node is queried with 2 values following happens:

1. 12030013: Direct forwarding to 12030013 as it is in the scope of leaf set.
2. 12031125: Routing table is searched based on prefix matching and forwarded to the most closest node(1203-1-121 in this case) and if value is absent then it returns the other closest node(if still not found then the current node is the destination).

3 Routing attacks and Practices

Structured P2P systems are subjected to various routing attacks. Here we discuss some of the major attacks and provide a solution on top of every attack. To avoid any confusion we consider a maximum of two solutions for a given problem. We distinguish all the attacks by following 4 categories:

1. Forward the request to incorrect or malicious node.
2. Pretend to be the destination.
3. Refuse to forward the request.
4. Provide invalid data as response.

3.1 Sybil Attack

3.1.1 Description

The Sybil attack is one of the most common attack in distributed system where it exploits the identification protocol. If the network is unable to provide a unique identification for each physical entity then an attacker can take an advantage of such protocol and create many such entities and thus controlling the network as a whole. For example, if there are many malicious identities, it becomes easier to moderate the routing table with the corrupted entries. However many DHTs work on the idea of having a considerable low fraction f of faulty nodes but Sybil attack technically breaks this defense by gradually increasing the quantity.

Category:

Sybil Attack forms the basis of all the four routing attacks named above and hence we need to take complete care to avoid it.

3.1.2 Solution: Secure Id Generation using CA

To prevent Sybil attack it is really important to generate id securely. Chord and Pastry generates nodeId using predefined random identifier space. It is really important to prohibit a single physical entity to generate many ids. Authors in [2] suggest to use trusted certification authorities (CAs) to assign nodeIds to principals and to sign nodeId certificates, which bind a random nodeId to the public key that speaks for its principal and an IP address. Moreover, with the help of public key, a secure encryption channel could be established. Also the CAs could guarantee a completely random assignment. For example, Pastry prepares its routing table by measuring the network delay between two nodes and one advantage of including IP address in the overlay disallows malicious nodes from swapping the nodeId certificates for benefits.
Additionally the same authors also propose to use a payment architecture where a node is granted a certificate based on a base payment. Payment architecture is important as even with a central CA a node could generate a lot of malicious nodeId certificates. A proper value to the base payment restricts the malicious node to control a significant part of the network. Even with a value as small as 20$, controlling 10% of the 1000 nodes costs 2000$. With the growth of network the amount also grows and hence it becomes increasingly difficult to control a significant part of the network.

Disadvantage:

1. Represents Single Point of Failure.

2. Payment gateways restrict many potential users.

3. Suitable for a small network or considerably small group of nodes.

3.1.3 Solution: SybilGuard Protocol

Authors in [13], explore the trust-relationship model in a social network. It exploits the fact that an edge between 2 nodes or peers represents a trust. A malicious user could create several entities but few trust relationships. Thus, there is a disproportionately-small cut in the graph between the colluding Sybil nodes and the honest nodes. SybilGuard restricts a malicious node to create more relationship with the legitimate node.

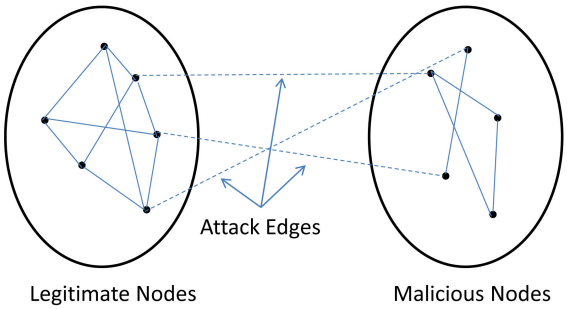


Figure 3. Trust-Relationship model(Edges represent a friendship or trust bind between two nodes)

Protocol ensures that the number of attack edges is independent of the number of Sybil identities, and is limited by the number of trust relation pairs between malicious users and honest users. If now a malicious user tries to create more entities then the graph would have a distinct subgraph which could identified by the cut as they won't have a huge trust relationship to the legitimate nodes and hence could be removed.

Protocol: It uses special kind of random walk known as random route. In random route, each node uses a precomputed one-to-one mapping from incoming to outgoing edge and hence multiple random routes entering a honest node always exits by the same edge (convergence). Every outgoing edge uniquely determines the incoming edge and thus helps for backtracking. Protocol generates random route for both verifier and the suspect and if there is a high percentage or intersection then only the suspect is added or else it is held guilty. Authors guarantee that even if a malicious node convinces a legitimate node it won't be able to create more edges with the legitimate node due to the definition of random route.

Disadvantage:

1. Suitable only for trust-relationship model.

3.2 Identity Attack

3.2.1 Description

In any DHT based overlay it is difficult to trust the identity of the replied node. While routing to the destination, a malicious node in between could claim to be the root of the key. Such kind of attack is called Identity Attack. Figure 4 shows a sample identity attack in a Pastry network.

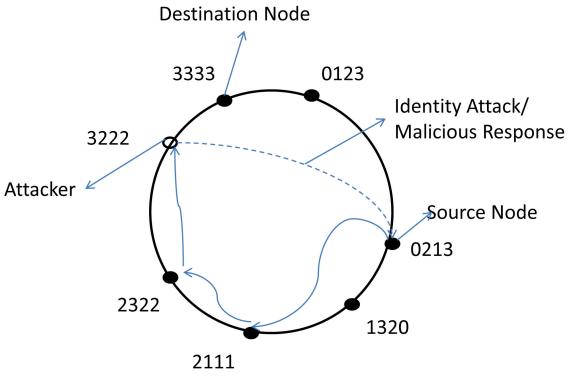


Figure 4. Identity Attack in a Pastry Network with base 4

Category:

This attack falls under category 2 were a node pretends to be the destination.

3.2.2 Solution: Proof-Of-Life Certificates

In [6], authors propose a solution by making every node to sign a certificate. Every node signs a proof-of-life certificate for partial nodeIds and distributes them randomly to the proof managers in the network. Any node could detect the false guarantees by querying the proof managers. Nodes sign the proof certificate periodically and store at the proof mangers to define its availability. When a node responds to a message with key K, the source node uses the namespace to determine whether the responded entry is true. If not, the source node guesses the prefix that K's real root will share with K, and sends verification requests to the proof manager responsible for that prefix. To establish this, offline CA provides every key pair a valid public-private key pair for encryption. For example, in Pasty, a node with id 1234, receives key pairs for 1, 12, 123 and 1234. To determine the proof managers for prefix group P, N applies a SHA-1

hash to P with several salts (i.e., 1, 2, 3) to generate several random keys in the namespace.
Also, verification at every step would result in a high overload and hence it is only performed when the node is found suspicious. Existence check is performed only if the length of the longest prefix column with all entries filled in is greater than the responder. Authors also understand that certifying all the prefix would lead to a considerable overload and hence they provide a solution with the help of cusp region. A cusp region, as defined, is the first combination of empty and non-empty column entries. Authors assume the cusp size to be 3 as they define it to be a good solution. Figure 5 shows the cusp region for 1232020. According the definition only the prefix of length 3, 4 and 5 are stored in the proof manager.

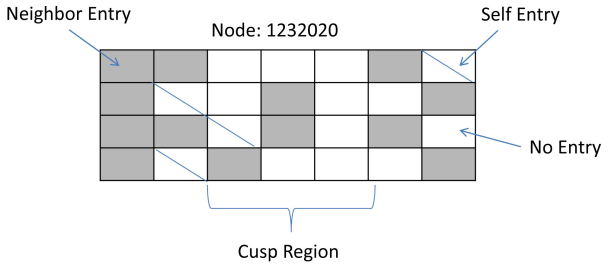


Figure 5. Cusp region for node 1232020

Authors[6] further measure the results using 2 metrics: the trigger rate, which measures how often an attack triggers a verification; and the verification rate, which measures how often an attack can be proved. Simulation results shows almost 100% verification rate when false positives are not considered. It shows a fall to 80% with 20% malicious node and 2 proof managers.

Disadvantage:

1. Storage attack is still possible.
2. Key Distribution needs a secure pathway.

3.3 Eclipse Attack

3.3.1 Description

Every node maintains a neighbor set and if a group of malicious nodes could control a larger fraction of these neighbors then the legitimate node is said to be eclipsed by this set and hence the name. This is other most common attack also known as routing table poisoning and it may lead to various other routing attacks. For example, in Pasty, routing table is built based on prefix matching so malicious node could provide false update thus poisoning the table.

Category:

This attack again falls under all the categories and hence need to be handled carefully.

3.3.2 Problem:Proximity Advantage

A malicious node from a group of such nodes intercepts the probe message by defining itself or other corrupted node as the destination and hence poisoning the routing table.

3.3.3 Solution: Central CA

This kind of attack could be protected using a strong communication model. Binding IP address to the identity would disallow faulty node from intercepting messages forwarded to others IP address[2]. Attack could also be avoided if we could assure that maximum fraction of colluding nodes c remains smaller than the fraction f of malicious node.

3.3.4 Problem: Provide False routing Updates

In this attack unlike to the previous attack of manipulating proximity, a faulty node provides update to the legitimate node and thus poisoning the routing table. It is really difficult to determine whether the update is received from a valid node or not and hence a malicious node could take a complete advantage of this weakness. For example, in Pastry nodes receives update when they join the overlay or even when other nodes join the overlay and hence becomes impossible to determine the validity of the solution. In [2], authors prove that fraction f of faulty nodes tends to 1 over a period of time and hence it is really important to prevent the attack.

3.3.5 Solution 1: Constrained Routing Table

Authors in [2] propose the use of additional routing table called constrained routing table. The idea behind implementing constrained routing table is to impose strong constraints on the value in each slot of the routing table. For example, the entry in each slot can be constrained to be the closest node id to some point in the id space as in Chord. This could be easily implemented on top of Pastry. A locality aware normal pastry table as shown in figure 2 and explained in section 2.2, in every slot (i, j) of node n contains a node which shares the first i digits with n and $(i+1)^{st}$ digit with j. In a constrained routing table we impose the same requirements plus we also demand the digits after j to be same as n. For example, in figure 2 in slot (2,2) we see node 12-2-13222 and in constrained routing table we would be finding a node n numerically close to 12-2-30201(only j^{th} entry is different rest all entries are same as n).

Routing Table Initialization: Any node contacts the bootstrap node to join the network. The initialization of locality aware Pastry table remains the same as they fetch the row entries from the nodes in the path of key search. In constrained routing table we perform a secure search of node close to particular point p in the routing space. This search is similar to Chord as explained in section 2.1.
Routing: Routing proceeds normally as described under section 2.2. In case of routing failure (we could determine this by the proposed failure test [2]), we use the constrained routing table for routing.

Disadvantage:

1. Fraction of malicious node in local routing table will tend to 1 as time increases.
2. Additional cost for performing failure test.
3. Additional cost for performing routing with constrained routing table (This cost increases with time).

An immediate solution to this disadvantage would be to use only constrained routing table for routing.

3.3.6 Solution 2: Reset + Constrained Routing Table

Authors in [3] proposed the solution on top of the solution proposed earlier to try to overcome the discussed disadvantage. The local routing table keeps getting poisoned and gradually the fraction of malicious node tend to 1 and hence the authors propose a periodically reset option. They propose a group reset at a time to disregard the overhead cost by reset all option. A reset here means that a node gains a new node id and then computes the new routing table. Simulation results show a considerable increase in use of local optimized table compared to the previous solution.

3.4 Denial of Service attack

3.4.1 Description

What if a malicious node just ignores to reply? Such kind of attack is called Denial of Service (DoS) attack.

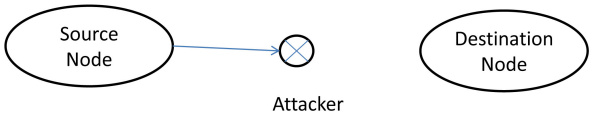


Figure 6. Denial of Service Attack

Category:

It falls under category 2 where a node refuses to forward the request to the destination node and hence resulting in the failure of message delivery.

3.4.2 Solution: Multiple Routes

In [12], authors propose to have multiple redundant routes from the source to destination. The solution is not framed for this kind of attacks but on close study works fine under such attacks. One can do this in Pastry by sending the message from the source node to all the neighbors in the leaf set. Due to consistent hashing all numerically close neighbors belong to geographically diverse location and hence delivery of a message depends on h such neighbors. Assuming a normal model with the fraction f of nodes being malicious, a message could be routed successfully with probability $(1 - f)^h$ where, h is a user defined parameter which is usually the same as L as defined in earlier section. Simulation results performed by authors show almost 99.9% success by bounding the fraction f of malicious node to 30%. Advantage of this approach is that we do not compensate by doing replication which could also be an alternate approach. Also generating more routes increases the success ration by a considerable amount.

Disadvantage:

1. It results in message overhead as many routes are generated for a single message.

3.4.3 Solution: Redundant Routing

In [2], authors propose to use redundant routing in case of failure. Every node can detect the failure based on the proposed test [2] and if there is a failure then authors use the constrained routing table as defined in section 3.3.5 to deliver the message. Message delivery could also be improved by replication. In Pastry, one could establish this by replication on every node in the leaf set. Hence, on failure one can deliver message to various replica roots. As long as the fraction of legitimate nodes are higher than the fraction f of malicious node a good success rate could be achieved.

Disadvantage:

1. If replication is used then an additional storage cost has to be included in the solution.
2. Fraction of malicious nodes in local routing table will tend to 1 as time increases.
3. Additional cost for performing failure test.
4. Additional cost for performing routing with constrained routing table (This cost increases with time).

4 Collaborative Solution

We have discussed various attacks with their solutions and some of the disadvantages. In this section we propose a unique solution by merging all the feasible solutions. The solution here could be implemented on the top of Pastry overlay to enhance the routing in a social network. Every node uses a combination of CA and SybilGuard Protocol to acquire identity securely. SybilGuard in addition could also help in identifying a malicious node and removal with an incentive based model. For routing table maintenance we could use a constrained routing table with periodic reset of local optimized table. As seen earlier this results in a considerable improvement due to periodic random id generation. Lastly DoS attack could be handled effectively by the constrained routing table. Figure 7 shows the described architecture.

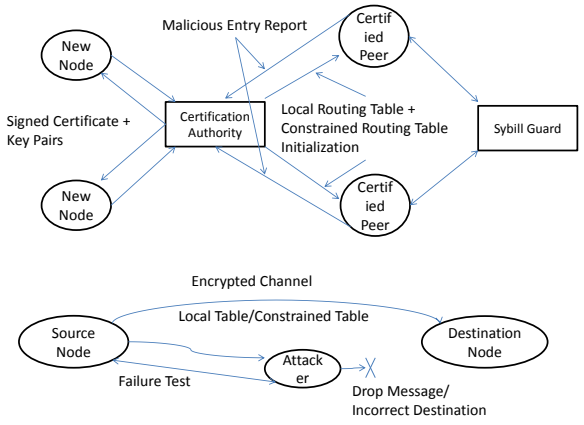


Figure 7. Collaborative Solution for FreePastry

5 Related Work

Security has always played an important role in research and hence there have been various research and analysis related to security in DHT. In [11], authors discuss and provide solutions to Sybil attack, Eclipse attack and Routing and Storage attack on all DHT based network. In [7], authors provide a distinctive comparison of both structured and unstructured P2P overlay. In [8], discuss in detail to have a robust P2P network and research mainly concentrates of methods to implement search on such a network.
In addition there are many more papers which concentrate on a single attack. In [5], authors discuss address proofing attack. They propose the use of micro cash for such attacks. In [4] author provides a defense mechanism against Sybil attack. They propose an idea of distributed registration which removes the dependency on central authority but the idea does not secure attack against large number of IP address. In [1], authors propose another defense mechanism against Sybil attack. It restricts the attacker by the number of computational puzzles it could solve. However it forces honest node to spend long time in solving computational puzzle and overall it is unable to acquaint in heterogeneous environment. Many more research exists around these topics.

6 Conclusion

Structured P2P networks have gained importance due to bounded lookup also technically defined as reduced latency. But such networks are prone to security threats. Here, firstly we revised the common routing mechanism in the absence of malicious entities by taking the example of Chord and Pastry. Both the overlays are implemented using DHT and are prone to various routing attacks. This paper, discusses some well-known attacks like Sybil Attack where a malicious node produces many such nodes, Identity Attack where a node gives false identification, Eclipse Attack where a group of malicious nodes eclipse a legitimate node and finally DoS attack where a node blocks the message forwarding by not responding to the request. We identify every attack by inserting it in at least one of the four categories of routing attack described in the section 3. Finally, the paper provides a collaborative unique solution which could be implemented on the top of Pastry for secure routing.

References

[1] N. Borisov. Computational puzzles as sybil defenses. In *6th International Conference on Peerto-Peer Computing*, 2002.

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *5th Symposium on Operating System Design and Implementation*, 2002.

[3] T. Condie, V. Kacholia, S. Sankararaman, J. M. Hellerstein, and P. Maniatis. Induced churn as shelter from routing-table poisoning. In *13th Annual Network and Distributed System Security Symposium (NDSS)*, 2006.

[4] J. Dinger and H. Hartenstein. Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. In *1st International Conference on Availability, Reliability and Security (Ares)*, 2006.

[5] J. Douceur and J. S. Donath. The sybil attack. In *1st International Workshop on Peer-to-Peer Systems*, 2002.

[6] L. Ganesh and B. Y. Zhao. Identity theft protection in structured overlays. In *1st Workshop on Secure Network Protocols(NPSEC)*, 2005.

[7] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 2005.

[8] J. Risson, T. Moors, J. Risson, and T. Moors. Survey of research towards robust peer-to-peer networks: Search methods. Technical report, Computer Networks, 2004.

[9] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware*, 2001.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *ACM SIGCOMM*, 2001.

[11] G. Urdaneta, G. Pierre, and M. V. Steen. A survey of dht security techniques. *ACM Computing Surveys(CSUR)*, 2011.

[12] D. S. Wallach. A survey of peer-to-peer security issues. In *International Symposium on Software Security*, 2002.

[13] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: Defending against sybil attacks via social networks. In *ACM SIGCOMM*, 2006.

Evaluation: Proper Testing and Large-Scale Tests

Anastasia Petkau
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: apetkau@mail.upb.de

Abstract

This paper deals with the various methods for testing peer-to-peer (P2P) networks. We discuss the centralized and distributed architecture. And then we compare them with respect to reaction time in the dependence to the number of nodes. Finally, a test environment PlanetLab and a brief introduction for its implementing will be represented.

1 Introduction

In recent decades, P2P systems have grown exceedingly, even more than it has been originally planned. Nowadays, the file-sharing applications like i.e. Napster, etc. expanded the application of P2P networks for different tasks: significant increase of video streaming, voice communication, gaming applications and social networks. It became apparent that the various applications require different types of P2P-overlays with significant differences in the network architecture. A test controls and evaluates P2P network to meet the defined requirements for their efforts and to measure their quality. The gained knowledge will be used for detection/identification and correction of bugs.

A test is no guarantee for a programs correctness. It can only show that certain test cases were successful. Dijskra wrote: "Program testing can be used to show the presence of bugs, but never show their absence!" [17]. The goal of testing is to measure the quality of the software system. The testing can be divided into multiple stages, as shown in figure 1.

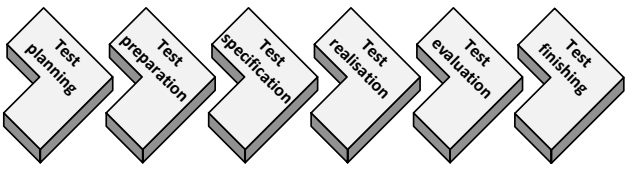


Figure 1. The phase model for testing [16]

The *test plan* is created for each new project and defines the entire testing process. It includes the following aspects, such as test objectives and criteria for starting the test, test strategy, procedure, test environment, test organization, etc. The *test preparation* is based on the test plan. At this stage for example the test environment is built and the documents for the test basis are provided. In the *test specification* requirements are set, which are necessary for the execution of specific test cases. It is recommended to run the test by independent persons and not by the developers themselves. In the *test analysis* stage are the actual results compared to the expected results. Afterwards it is decided whether the objectives have been achieved or not.

There are also various types of tests, such as alpha-tests (first developer tests) and beta-tests [14]. Alpha-tests are the first stage in the developing process. This includes unit testing, component testing, and system testing. The second stage is called beta-test. In this phase the product is tested by a big group of users. A distinction is made in "open beta" and "closed beta". The "open beta" is, as the name implies, open to everyone to evaluate the product, to describe the errors that have occurred and to give a feedback on usability and navigation. The "closed beta" can only be performed by persons who are invited to the test.

2 Benchmarking

Benchmarkings are standardized measurement and evaluation procedures that will help you to identify the performance of computer systems or system classes to compare these with each other according to certain criteria.

- Using the *workload specification* the type of requests and the frequency of the submitted requests to the system under test (SUT) can be determined. The system under test is a system which must be tested or compared [11].

- The *specification of metrics* defines which values are supposed to be measured. For example, the following values can be measured: throughput, average response times, CPU consumption, or failure rates [11].

Next, the qualitative aspects are given, which are used for the evaluation.

- *Validity, meaning that the system responses are complete and correct [10] .*
- *Efficiency, defined as the ratio of performance and costs [10] .* The performance refers to the number of operations, the costs of the number of Ressources.
- *Fairness, meaning that both the costs for operating the system and the availability of services are distributed over the participating entities (peers) such that a given fairness criteria holds [10] .* There are different types of fairness criteria, this depends on requirements.
- *Scalability, the quantitative adaptability of the P2P system to a changing number of entities (peers or services) in the system, while preserving validity, efficiency, and fairness [10] .* So, it is the adaptability of the system, e.g. when the number of peers changed.
- *Robustness, the persistence of a P2P system when crucial parts of a system fail [10] .* This means that a P2P system detects the error and deals correctly with such situations, so that a correct result will be provided despite unfavorable operations.
- *Stability, the persistence of the P2P system under system perturbations such as intensive or frequent use of system functions [10] .* This means that a P2P system behaves under as many conditions as possible, so that there is neither data loss nor unwanted security breaches or system failures.

3 Testing Architectures

The goal of testing is to detect the software errors. In this case an expected behavior is compared with the actual. The goal of test cases is to check the system. The review is based on various criteria that are defined in the specification. Next, some terms are defined. P2P systems and distributed applications should be tested on the basis of the specificity of Transportation. The tester observes the events and controls whether the nodes arrive in the correct order. There are different test architectures which can be helpful for the evaluation: *Centralized Tester Architecture* and *Distributed Architecture*.

3.1 Centralized Tester Architecture

Centralized Tester Architecture is a classical architecture for testing of distributed systems. The figure 2 shows its structure. The test controller sends the test inputs, receives the outputs or local verdicts from each node of the SUT and simultaneously controls the synchronization of distributed systems. The SUT is a system under test. The tester may interpret the results [2]. As we shall see later, the Centralized Tester Architecture is not suitable for large scale systems. The writing test case is not easy because the tests have to deal with larger numbers of peers. The sequence of the actions has to be considered.

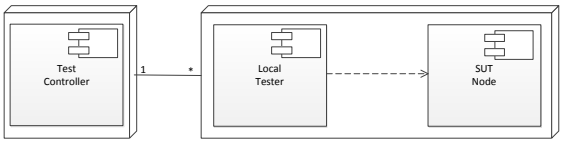


Figure 2. Centralized Tester Architecture [5]

3.1.1 Test Case Example

First, some basic concepts [6] have to be defined, before we start with the test case example.

Definition 1 (Distributed test case) A distributed test case noted τ is a tuple $\tau = (A^\tau, T^\tau, L^\tau, S^\tau, V^\tau, \varphi^\tau)$ where $A^\tau \subseteq A$ is an ordered set of actions $\{a_0^\tau, ..., a_n^\tau\}, T^\tau \subseteq T$ a set of testers, L^τ is a set of local verdicts, S^τ is a schedule, V^τ is a set of variables and φ^τ is the level of acceptable inconclusive verdicts.

Definition 2 (Schedule) A schedule is a map $S : A \mapsto \Pi$ where Π is a collection of tester sets $\Pi = \{T_0, ..., T_n\}$, and $\forall T_i \in \Pi : T_i \subseteq T$.

Definition 3 (Action) A test case action is a tuple $a_i^\tau = (\Psi, \kappa, T')$ where Ψ is a set of instructions, κ is the interval of time in which Ψ should be executed and $T \subseteq T$ is a set of testers that executes the action.

Definition 4 (Test sequence) A test sequence is a sequence of test actions $TS(A)$, where each action a_n^A has a hierarchical level $h_{a_n}^A$. Actions with lower levels are executed before actions with higher levels.

Definition 5 (Local verdict) A local verdict is given by comparing the expected result, noted E, with the result itself, noted R. E and R may be a single value or a set of values from any type. However, these values must be comparable. The local verdict v of τ within κ is defined as follows:

$$l_\kappa^\tau = \begin{cases} pass\ if R = E \\ fail\ if R \neq E \\ inconclusive\ if R = 0 \end{cases}$$

The following example [3] illustrates the operation of Centralized Tester Architecture.

This includes three testers test case T = {t0, t1, t2}, seven actions A = {a1, ..., a7}. If the data is retrieved in a4 and a6, the same were inserted in a2, it says the verdict is pass. In the other case, if the data are unequal, the verdict is fail. Local verdict is calculated and sent to the test coordinator.

Action	Testers	Action
(a1)	0,1,2	join()
(a2)	2	Insert the string One at key 1; Insert the string Two at key 2;
(a3)	*	Pause
(a4)	0	Retrieve data at key 1; Retrieve data at key 2;
(a5)	1	Leave the system
(a6)	0	Retrieve data at key 2; Retrieve data at key 2;
(a7)	0, 2	Leave the system
(v0)	0	Calculate a verdict

Table 1. Test Case Example

3.2 Distributed Architecture

In this chapter, another alternative to the Centralized Tester Architecture is presented. The idea is to classify the nodes in a tree structure. So, the test controller will be replaced by a hybrid tester. The hybrid tester plays the role of the controller and the tester. Furthermore, three methods are presented that allow testers to build a tree[4].

3.2.1 Balanced - Tree

This architecture is a matter of the testers in a tree of the order m structure to organize [6]. As mentioned above, the idea was to drop the test controller. The control over the execution of the test cases is taken from the root node. In the figure 3, an example of a balanced tree is shown. The tree has a root node, two hybrids and four leaf testers, this tree is of order 1.

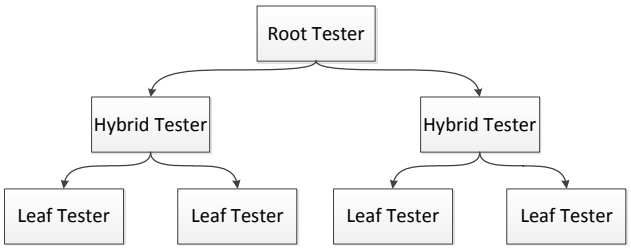


Figure 3. Balanced - Tree [3]

The root node sends the actions to his children. If all actions are performed, the leaves send their results to their parents. The results come from the node to root. Afterwards, the root node can send further actions to his children. This organization of nodes gives better results as Centralized Tester Architecture with regard to global execution time, as we shall see later.

3.2.2 Optimized Tree

In this architecture, a better balance is achieved by testers. A physical node contains one hybrid tester at most [2]. The figure 4 shows a possible design of an optimized tree. There are a total of nine testers, root 1, 2 and 6 leaf hybrid testers. The tree is of order two.

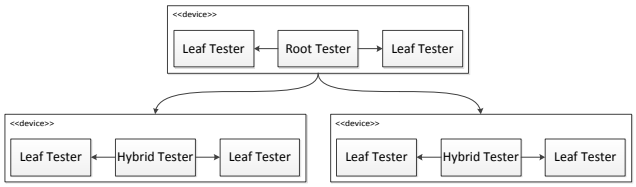


Figure 4. Optimized Tree [3]

The hybrid tester has control over m children, all the tester from some physical node, and their dependents. Dependents are the leaf nodes. The number of leaf nodes is not fixed.

3.2.3 Gossiping

Gossiping is an other distributed architecture. This architecture used for the multicast address for sending messages to a group of testers. The first tester t0 is the first in the system, followed by the other tester (t1-tn). Identifier tn assigns the action to the node. The tester t0 generates a multicast address for each test action. The test is divided into pieces. A tester can play two roles [7].

- *Busy tester.* This tester performs an action. When he finishes, it sends a message to multicast address and afterwards this tester becomes an idle tester.
- *Idle tester* waits for an action [5]. If he gets all the gossip, he begins with the action and becomes a busy tester.

The figure 5 represents a possible Gossiping architecture. The execution starts at the t0. If t0 is finished with the job, he gives notice to the other nodes {Tester1, Tester2, Tester3} through multicast address which can then start parallel with the execution of Actions. If the action is done, the tester sends a gossip to the multicast address [2].

3.3 A comparative analysis of different testing architecture

In this chapter, the above described architectures will be compared relating to the reaction time. The fake test cases will be sent in order to measure the response time. This test case contains empty actions. The execution time is measured for every action. Thus, the execution time of the remote invocations and the confirmations are measured by the test controller. The following diagram 6 represents the results of experiment [7].

A test case consists of eight actions, the number was chosen arbitrarily. As you can see immediately that when the

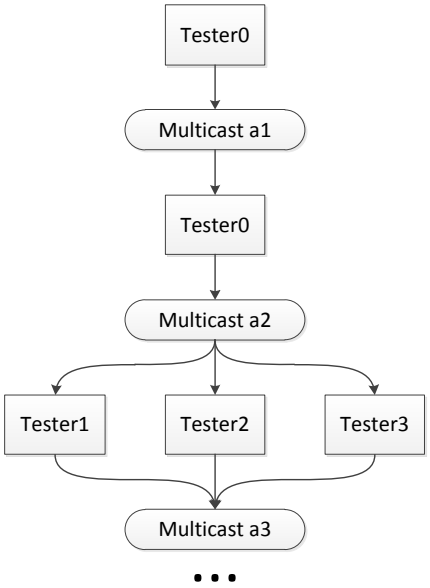


Figure 5. Gossiping [7]

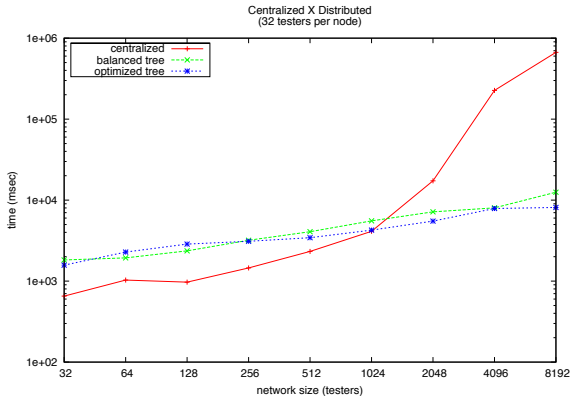


Figure 6. Analysis of Different Testing Architecture, [2]

number of testers grows, the reaction time of the centralized architecture increases linearly. So, this architecture is more efficient for small environments than distributed architecture and is not suitable for testing in large-scale environments.

4 PlanetLab

PlanetLab is a global test environment, which can be used for testing new services among others for testing P2P networks. This is a group of computers that consists of 1116 nodes at 543 sites (http://www.planet-lab.org/), see fugure 7. In 2002 David Culler and Larry Petersen had a vision for the creation of PlanetLab. Intel research labs took over the administration in 2004 and donated the first 100 machine.

A special emphasis was placed on three principles in the design of PlanetLab [8]: Distributed Virtualization, Application Centric Interfaces and Unbundled Management.

Distributed Virtualization: This principle states that the test environment does not command the central scheduling mechanism. The new lab will allow the test to be parallel, continuous and run for a long period of time.

*Application Centric Interfaces:*This principle is a well-known and widely used interface in the network area. The user can work with the application without long learning periods.

Unbundled Management: This point indicates that the system is supposed to meet the future requirements.

These concepts are reflected in the implementation. Distributed Virtualization is done by slice and sliver concept; these are the basic abstractions of the hardware. A slice describes the totality of all resources that are allocated to

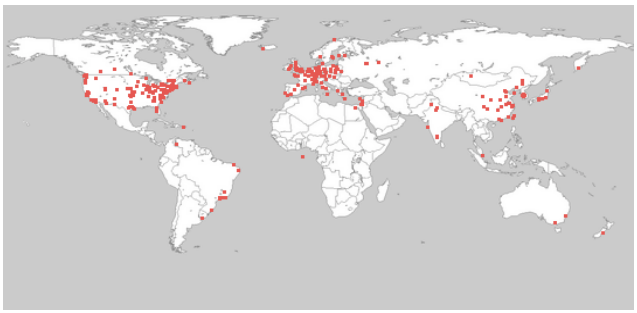


Figure 7. PlanetLab [12]

a user. A sliver however describes the hardware abstraction, similar to a virtual machine that is realized by Vserver. Vserver is a Linux - kernel extension that ”Slivers” produced. Another module of PlanetLab is VNET. An important function principle of VNET is the trace of network connections and their assignment to a VServer. Only connections are permitted that have been assigned to the slice or initiated by him. Another service provided by PlanetLab is the Proper. The Proper enables the slices to access operating system services, even with the limited user rights. A central boot server is a PlanetLab Central. This boot server represents the available node program updates and hold information about all nodes in a database. Moreover, it gives information about other nodes. The PlanetLab consists of these services, or modules [9].

Since the PlanetLab has been founded, it prospered steadily. However, the future development in the form of a decentralized structure is uncertain. This could lead to the risks of split-up. For example, the European headquarters stand under the administration of OneLab which is funded by the EU, so that it is possible that the European interest might grow over time. This might lead to incompati-

bilities between the various parts of continental PlanetLab [15]. Therefore, this will make the global usage at some point impossible. PlanetLab initiators still see it as a lasting Experiment. Hence, it does not introduce a substitute for accurate measurements, but rather an important addition to this option. PlanetLab offers the user different options to test its applications in a real distribution operation on giant system effects, fault tolerance and self-healing.

4.1 command line utility and shell

In addition to that, there is a presentation of some easy-to-install and-use programs. These tools allow to perform almost all tasks that arise in connection with the use of a slice on PlanetLab. Thus, one can for example copy your application on all nodes, starting and monitoring the application. These programs are robust and run on almost all systems; the graphical user interface is missing, however.

4.1.1 plcsh

The plcsh (PlanetLab Central Shell) was developed by Princeton University. It is used for interacting with PlanetLab Central database.

4.1.2 pssh

The Pssh is a collection that consists of four command line utilities: pssh, pscp, pnuke and prsync. This toolkit allows an interaction with a slice. The programs can be quickly and easily install, as described in [1].

4.1.3 vxargs

vxargs is a wrapper that was written in Python. It was developed at the University of Pennsylvania. vxargs from Unix command to create a parallel version, i.e. the tool can be used by the commands to run in parallel.

4.2 Simple PlanetLab Starter HowTo

This chapter deals with a short introduction [8] of what steps are needed to transform an application on PlanetLab nodes and start. There are several ways, only one is presented.

4.2.1 Preporations

First you need a computer with a Fedora Core 2 or Core 4 installation. Next, register for a PlanetLab account is required. This one can be found on the homepage of PlanetLab: https://www.planet-lab.org/db/persons/register.php. Its function is to fill the form with all required information and then to wait for a confirmation email.

4.2.2 Key Generation

Afterwards you create an SSH key pair in open-ssh format by using the following call [13]:

```
me@test: ~ $ ssh-keygen-t rsa-f ~
/ .ssh / id_rsa
```

Private key should be between 10 and 30 characters long. You can start at the beginning of the session ssh-agent and give him the pass phrase, which does not have to be re-entered.

```
me@test: ~ $ ssh-agent
me@test: ~ $ ssh-add ~ / .ssh / id_rsa
```

So, with this call, you can reach that one should only enter the passphrase once per current session.

Next you need to login on the PlanetLab website, and via the "My Account" $\rightarrow$ Key Management" enter of the generated public key. In the best case the private key should be stored on a USB stick. Using PlanetLab regularly for a long period of time, the pair of keys must be newly created/regenerated.

4.2.3 First Login

First login attempt can be made with [13]:

```
me@test: ~ $ ssh-l huh_slice-i ~ / .ssh
/ id_rsa pl1.hu-berlin.de
```

It is believed that the name of slice slice-huh, and the computer-berlin.de pl1.hu belongs to the slice.

4.2.4 Adding Nodes to Your Slice

In this chapter it is shown how a node can be added to the slice. There are several ways, only one is described in the following. In order to add more nodes, it is appropriate to use the tool plcsh. The plcsh is a shell for PlanetLab Central database, which is implemented in Python. The plcsh tool will be started with the following call [13]:

```
me@test:~$ plcsh -h
https://www.planet-org/PLCAPI -u pl_user
```

The next step is to call the following:

```
[pl_user]>>> file = open('hostlist', 'r')

[pl_user]>>> s = file.read()
```

```
[pl_user]>>> AddSliceToNode
("hawh_slice", s.splitlines())
```

If you want to check whether the actual node has been added to the slice, use the following function:

```
[pl_user]>>> GetNodes()
```

4.2.5 Application distributed on the nodes:

This chapter deals with the application of scp tool which is to be distributed to the nodes. The scp tool is located in open-ssh package. But the ssh-agent should be first started and the test folder must be copied to the home directory of the nodes a-plab-node of the slice. The call is then as follows [13]:

```
me@test:~$ scp -B -i ~/.ssh
/id_rsa -r test hawh_slice@a_plab_node:
```

But if you have more than five nodes, this call is not recommended. The support can be accomplished by the lower standing Python script.

Listing 1. Application distributed on the nodes

```
#!/usr/bin/python
# pscp.py
import sys
import os
import commands
from optparse import OptionParser
scp = '/usr/bin/scp'
home = os.getenv('HOME')
# enter here if the path to my private keys
# or alternatively it is also a parameter passing
# In the command line possible
defaultKeyFile = home + '/.ssh/id_rsa'
# Enter here my slice name
username = hawh_slice
# enter here the path to my hostfile
# hostfile: Textfile with Hostname or IP-Adress.
#           one name per line
hostfile = home + ' /hostfile'
parser = OptionParser(usage='\n%prog [OPTIONS] (FILE|DIR)')
parser.add_option("-r", "--recursive", dest="recursive", default=False,\
                help='copy dir recursively or not, default is %default')
parser.add_option("-i", "--key-file", dest="keyfile",\
                default=defaultKeyFile,\
                help='the key-file to be used, default is %default')
parser.add_option("-d", "--destination", dest="destination", default='',\
                help='the directory the file(s) will be copied to,' +\
                'default is %default')
(options, args) = parser.parse_args()
if not(len(args) == 1):
  parser.print_help()
  sys.exit(256)
file2scp=args[0]
f = open(hostfile, 'r')
s = f.read()
hostlist = s.splitlines()
if not options.recursive:
  command = scp + ' -B -i ' + options.keyfile +' '+\
```

```
            file2scp +' '+ username +'@'
  command = scp + ' -B -i ' + options.keyfile +' -r '+\
            file2scp +' '+ username +'@'
for host in hostlist:
  cmd = command + host +':'+ options.destination
  (status, output) = commands.getstatusoutput(cmd)
  print host
  print output
  print status
  print "\n"
sys.exit(0)
```

Assuming, the script is stored under the name of script.py, then copy the above works on all the nodes in our slice as follows:

```
me@test:~$ script.py -r test
```

4.2.6 Parallel exports commands:

Next, it is shown how to start a project, so you can follow the process afterwards. In the simple case, it's back on best with command-line tool ssh. The call looks like this:

```
me@test:~$ ssh -i ~/.ssh
/id_rsa hawh_slice@a_plab_node command
```

But if you have more than five nodes, this call is not recommended. The support can be accomplished by the lower standing Python script .

Listing 2. Parallel exports commands

```
#!/usr/bin/python
# pcmd.py
import sys
import os
import commands
from optparse import OptionParser
ssh = '/usr/bin/ssh'
home = os.getenv('HOME')
# enter here the path to my private keys
# or alternatively it is also a parameter passing
# In the command line possible
defaultKeyFile = home + '/.ssh/id_rsa'
# Enter here my slice name
username = hawh_slice
# enter here the path to my hostfile
# hostfile: Textfile with Hostname or IP-Address.
#           one name per line
hostfile = home + ' /hostfile'
parser = OptionParser(usage='\n%prog [OPTIONS] -c COMMAND')
parser.add_option("-i", "--key-file", dest="keyfile",\
                default=defaultKeyFile,\
                help='the key-file to be used, default is %default')
parser.add_option("-c", "--command", dest="command", default='',\
                help='the command to execute on remote host,' +\
                'default is %default')
(options, args) = parser.parse_args()
if command == '':
```

```
  parser.print_help()
  sys.exit(256)
f = open(hostfile, 'r')
s = f.read()
hostlist = s.splitlines()
command = ssh + ' -i ' + options.keyfile +' '+\
          username +'@'
for host in hostlist:
  cmd = command + host +' '+ options.command
  (status, output) = commands.getstatusoutput(cmd)
  print host
  print output
  print status
  print "\n"
sys.exit(0)
```

Assuming, the script is stored under the name of script.py, the following call will start the application on all nodes in our slice:

```
me@test:~$ script.py -c "command"
```

5 Conclusion

In this paper we have discussed various test architectures. It was also shown under what circumstances which architecture is more suitable for testing. So, for example a large number of testers said the distributed architecture is faster regarding the reaction time. As a possible test environment PlanetLab was introduced and at the end a short tutorial on the commissioning of PlanetLab was given.

References

[1] B. Chun. pssh howto. http://www.theether.org/pssh/docs/0.2.3/pssh-HOWTO.pdf. Last accessed on 08.06.2012.

[2] E. C. de Almeida, J. E. Marynowski, G. Sunyé, Y. L. Traon, and P. Valduriez. Efficient distributed test architectures for large-scale systems. In *ICTSS*, 2010.

[3] E. C. de Almeida, G. Sunyé, Y. L. Traon, and P. Valduriez. A framework for testing peer-to-peer systems. In *ISSRE*, 2008.

[4] E. C. de Almeida, G. Sunyé, Y. L. Traon, and P. Valduriez. Testing peers' volatility. In *ASE*, 2008.

[5] E. C. de Almeida, G. Sunyé, Y. L. Traon, and P. Valduriez. Testing peer-to-peer systems. *Empirical Software Engineering*, 15(4), 2010.

[6] E. C. de Almeida, G. Sunyé, and P. Valduriez. Action synchronization in p2p system testing. In *DaMaP*, 2008.

[7] E. C. de Almeida, G. Sunyé, and P. Valduriez. Testing architectures for large scale systems. In *VECPAR*, 2008.

[8] M. Dramburg. Planetlab - eine übersicht.

[9] B. Karisch. The planetlab platform. http://users.informatik.haw-hamburg.de/~schmidt/p2p/sup/karisch06.pdf. Last accessed on 08.06.2012.

[10] M. Lehn, T. Triebel, C. Gross, D. Stingl, K. Saller, W. Effelsberg, A. Kovacevic, and R. Steinmetz. Designing benchmarks for p2p systems. In *From Active Data Management to Event-Based Systems and More*, 2010.

[11] D. A. Menascé. Load testing, benchmarking, and application performance management for the web. In *Int. CMG Conference*, pages 271–282, 2002.

[12] PlanetLab. Planetlab website. http://www.planet-lab.org/. Last accessed on 08.06.2012.

[13] PlanetLab. Hello world tutorial. http://svn.planet-lab.org/wiki/HelloWorldTutorial, 2008. Last accessed on 08.06.2012.

[14] B. Reese. Alpha and beta testing. http://www.majordojo.com/uploads/AlphaBetaTesting.pdf, 2006. Last accessed on 08.06.2012.

[15] M. Wenzel. Das planetlab eine übersicht. http://users.informatik.haw-hamburg.de/~schmidt/p2p/sup/Wenzel_m-PlanetLab.pdf. Last accessed on 08.06.2012.

[16] Wikipedia. Softwaretest. http://de.wikipedia.org/wiki/Softwaretest. Last accessed on 08.06.2012.

[17] A. Zeller. Software-test. http://www.st.cs.uni-saarland.de/edu/einst/12-testen.pdf. Last accessed on 08.06.2012.

The Software Licenses used in the Peer to Peer Framework

Subramanya Kuduvally Gurumurthy
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: skg@mail.uni-paderborn.de

Abstract

This paper identifies and analyses the different types of licenses associated with the libraries and or framework used in the current project. Also it takes a close look with the compatibility of these licenses. Libraries or any proprietary software component is governed by a license and licensor would impose few legal constraints on using this libraries for commercial or non-commercial purpose. Typically, Software systems involve interaction between lot of components and this interaction may lead to incompatible license for the resulting system, there by leading to legal consequences. Hence it is important to verify for compatibility of licenses for the system at various stages. In this we identify the type of license associated with each library and verify for compatibleness. Libraries with incompatible licenses are required to be replaced by alternatives so that resulting software system contains libraries with compatible licenses.

1 Introduction

There is a lot of effort spent in building a new software product or a system. Essentially, while creating a new software system it need not be built from scratch, i.e. every component and every piece of code need not be written from scratch. Rather it could be derived or used from the previously built similar applications or components. This is called as reusability. This re-usability may usually take place through using those software components as libraries, plug-ins or as frameworks in the current system. In doing so, we need to analyze the compatibility of using these libraries, plug-ins or frameworks due to its associated copyright or Intellectual Property. The analyses of these types of Copy Rights or Intellectual Property compatibility can be realized through Software Licenses associated with them. Before trying to understand," what is software license?" and the need for it let us look at a case study where there was an issue regarding the breach of this agreement on reusing this libraries or APIs developed by another company.

Case Study : SDKs and APIs: Do they have copyright protection?

In the below case study we look at an on-going issue on Copyright/Intellectual Property Infringement between two major Software product companies, Oracle America Inc., and Google Inc., (Case No. 3:10-cv-03561-WHA). This case study is completely based on articles of [1], [9] and [10].

The issue is regarding what scope of protection is given to functional elements in an SDK (Software Development Kit) or an API (Application Programming Interface). Oracle America Inc., alleged that Google Inc., infringed the copyright in Java code when it built the Android Software platform.

Specifically, Oracle America Inc., argued that Google Inc., copied design specifications of some 37 APIs for Java libraries, as well as Java source code, object code and comments in 12 software files. In its defense, Google Inc., claimed that the APIs are purely functional elements and are not eligible for copyright protection. Google had articulated a compelling argument that there was no copyright in the Java APIs. In a recent decision the court disagreed that there was any copyright or patent infringement from Google Inc., with no prospects for royalty payments to Oracle America Inc., but it is still left out deciding whether SDKs and APIs can be subject to Copyrights/Intellectual property rights. Due to some inherent weakness in the Oracle America Inc's., copyright infringement assertions Google Inc., rescued from being sued.

Therefore, it is important to clearly understand the governing License, Copyrights or any Intellectual property associated with the software components before reusing them. So to know them we need to answer the below two questions
,

1. What is Software License ?
2. What is the need for Software License?

2 Software Licenses

Software License is an agreement or a legal contract made for the piece of code, a component or whole system

being developed. This legal agreement is done in between the developer or the publishing organization of that software and user of the software. The user of the software is obliged to adhere to the constrains in the license agreement.

Software Licenses protect the intellectual property rights on the software on behalf of the author or organization beholding that license (licensor), meanwhile providing limited or full (depending on the license) rights to the user (licensee) in using that software thereby saving lot of efforts, costs in developing new software and in the same time licensor may be monetary benefitted.

2.1 Software License: The Concepts

In general a software system will be composed of many software components. Each component may be realized as a set of software packages. Each of these software package may be using one or many predefined libraries and frameworks. Each library may be associated with a license type and this type may have numerous variants. Also these licenses may evolve in future. So it is very important to monitor on the license type and its variant from time to time. Also it is important to update and check the compatibility of the new license with our existing system when it evolves. The interactions between various software components and also configurations of these components may have different implication on the license of the whole software system. To note, configuring components at compile (build-time) time and run-time may have different implication on the license. For example, few components in the software system may be bound to each other only at static time (i.e. compile time) and few may interact only at dynamic time (run-time) and thus it is very important to consider all these factors when releasing the software product and all licensing issues has to be taken care. Licensing issues also has to be taken care at times, when software architecture is restructured or when new components are introduced to existing system [11].

2.2 Software Licenses: Rights and Obligations

The rights associated with the software can be realized through patents or copyrights. Essentially these copyrights legally give permission to the author to use the software, modify the software, distribute and sell the copies of software to other individuals or organizations. These copyrights to a product may enter to the public domain after few years, until then the copyright can be obtained through license. However the licensee is obliged to adhere to the license [11]. Licenses may constrain or impose obligations on licensee to realize the his rights on the licensed software product. Few imposed obligations include such as the licensee is obliged to publish the modified code at no cost, or to publish all the build-time source code at no cost. However there would be conflict in meeting obligations when two or more license involve in a product. For instance, one copyright may oblige the licensee to release the source code and other may restrict him doing so. In this kind of situation it is not possible to adhere to any of the license constraints. Hence a careful study is required even on the license of the components during the system design. Therefore rights are granted to licensee only when the specified obligation is fulfilled[11].

3 Terminologies related to Software License and Use of Software License

A typical license document will contain the following declaration: *"Subject to the provisions of this Agreement, Licensor grants to Licensee a perpetual, personal, nonassignable, nontransferable, nonexclusive object code license to use the Software solely for Licensees internal business purposes in the United States "*[12].

Each terms in the above declaration has a special meaning which dictates the rights of licensee and licensor.

3.1 Definition of the Licensee and Licensor

- **Licensee** The word Licensee is considered to be important in the aspect of financial and legal terms. In broader sense the term licensee defines the parties or persons who will be authorized to use the software may be at higher monetary benefits. In narrower sense, it tries to restrict the licensees use of software including any third parties or potential opponents of the licensor from accessing the software[12].

  Some Licenses allow affiliates or subsidiaries of the licensee to use the software. There might be some restriction when we mean affiliation or subsidiary to the licensee, like there might be a limit on the number of users or there might be a software access grant to the licensees parent company or any subsidiary companies owning at least 51 percent stake in the licensee company[12]. It is also important for the licensor to consider implications of allowing parent companies as well subsidiaries of licensee companies to access the software in case of any dispute, may be related to the usage of the software. It may be implied that, in the event of dispute all the individual or subsidiaries may sue the licensor separately and individually claiming that they are the third party beneficiary of the licensed software and there is breach of license. Thus it is important for the licensor to include clauses related to allowing third party beneficiaries[12].

- **Licensor** The term Licensor is important from the view of licensee. There are certain things which the licensee has to make sure before buying a licensed product from the licensor. The licensee must make sure that the licensor actually owns the intellectual property

right on the software being purchased and also the licensor is financially enough strong to pay the penalties in case of license breach from the licensor[12] .

3.2 Term of License

It is always better to start the term of license from the day the software is delivered, rather than from the day on which the license was signed to protect from any uncertainties on usage of software prior to acceptance of license. The license should also clearly state that it would protect the licensors interest on delivery of the licensed software and the acceptance of software does not occurs until the license is accepted by the licensee[12].

3.3 Use of License

- **General** License place a restriction on how the licensed software can be used by the licensee. Most of the time, exception being in case of open-source software the prime objective behind restricting the use of software is the financial criteria. At sometimes the objective might also be that the licensee is a competitor[12]. To avoid any misinterpretation on the restriction of using, the licensor must define the term "use" in relation to software.

  A typical "use" statement in the license would look like as below :

  " "Use" shall mean the ability to run, execute display, duplicate and distribute internally the licensed software in its Object code form. In the event that the Source code for the Licensed Software is released from escrow , then "Use" shall also mean the right to modify and create derivates of the Source Code"[12].

  While drafting the grants on the license, the licensor and the licensor must take the below three measures to meet their requirements:

  1. Both the parties must consider the fundamental rights a copyright owner is granted:
     - to reproduce the copyrighted work,
     - to prepare derivative works and
     - to distribute copies of the derivative work.
  2. The parties must consider any special rights that are unique to the system.
  3. Finally any industry specific terms such as right to "create with", in case of any tools being used.

  An example license grant would read as follows:

  "Licensor grants to Licensee a transferable, nonexclusive, right to Use the Licensed Software during the Term and any renewal period"[12].

- **Internal Use** Sometimes there would be an inherent confusion on the kind of environments the licensed software can be used. Typically a license would state that the licensed software can be used for "internal purpose only". The main objective of this kind of statement is to bind the usage of software to internal business environment and prevent it from being outsourced. It is very much required to mention this in the license grant to avoid any dispute over interpretation at later times. However, the term "internal-use" is quite ambiguous as most of the time the software licensed would be used to either build a product which would be then marketed or service an external product, which would lead to breach of "internal-use" term in either of the cases. Thus it is required to clearly state in the license, what do we mean by saying for "internal-use"[12].

- **Nonexclusive or Exclusive Use** In some cases when a software is being developed for a particular company, that company reserves an exclusive right on the software. In such situations the license has to be issued on exclusive basis for that software. This may occur when the company thinks that software developed for it requires competitive protection. If the license is stated as "nonexclusive", it means that the licensor reserves the right to sell or license the same software to other licensees [12].

- **Scope of Use** Scope of Use, is a kind of criteria that can be imposed by the licensor on the licensee. It may include , limit on the number of central processing units, to a class of computers, to a particular geographic location, or to a specific duration of time or to a predefined number of users etc. This criteria may benefit the licensor at time when the company expands or change the mode for usage of the licensed software. Therefore, it is required for the licensee to carefully look for a license which offers flexibility for the future needs [12].

- **Creation of Derivative Works and the Prohibition of Reverse Engineering**

  According to Copyright Act (Section 101)[12]: *"a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgement, condensation or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications, which, as a whole, represent an original work of authorship, is a derivative work"*.

Section 106(2) of Copyright Act prohibits the creation of derivation works without the permission from copyright owner. Sometimes it is required to create derivative works of the software being licensed. In such situations, it is required to know in prior on authors stand for granting permission to create derivative work on the software being licensed. Any use of Copyrighted materials in derivative work that exceeds the scope of license may result an infringement. A license need not explicitly mention that a copyright in a derivative work will be the copyright property of the owner of the original work. If licensor is granting any special permission to licensee for any modifications of the original work, then it can be done through the Copyright Act (section 117).It is required for the licensor to carefully observe whether sum of these modifications will raise to the level of derivative work and also to seek the copyright for any modifications done. This would prevent licensees from claiming the ownership on portions of the original software[12].

To avoid any reverse engineering on the licensed software, the licensor has to include a clause in the license stating the prohibition of any reverse engineering , decompiling or recompiling.

- **Object Code License**

  Object code refers to binary, machine readable form of the software. Object code of any software permits the licensee to use the software but it does not provide a way for any modifications, derivative works or future enhancements. For this purpose source-code is required. By providing license for source-code , the licensor would be under risk that licensee would become a potential competitor. Therefore, while granting a source code license, the licensor must impose restriction on the licensee from creating any derivative work , enhancements or any modifications[12].

4 Software Licenses involved in "Peer-to-Peer framework for Social Networks" project

Based on the libraries and the framework used in the project "Peer-to-Peer Framework for Social Networks" below are the set of licenses involved:

- **GNU GPL:** GPL Stands for "General Public License". As per GPL License : "GPL permits users to publish their modified versions. By using GPL it doesnt require us to release our modified version, or any part of it. We are free to make modifications and use them privately, without ever releasing them. This applies to any companies too; an organization can make a modified version and use it internally without ever releasing it outside the organization. But if we release the modified version to the public in some way, the GPL requires us to make the modified source code available to the programs users, under GPL"[4],[8]. But the main disadvantage of GPL is that, if we sell our product to someone and he deserves the right on the product to sell to other parties with or without fee. By this it means that we are transferring our rights equally to the selling party. Also GPL imposes a constraint that every copy of the product must contain a copy of GPL notice, so that the buyer knows his rights[4], [8].

- **GNU Lesser General Public License**: GNU LGPL states that: *"Using the Lesser GPL for any particular library constitutes a retreat for free software. It partially abandon the attempt to defend the users freedom, and some of the requirements to share what is built on top of GPL covered software"*. By using Lesser GPL we mean that we are not making our program a free software. GNU Lesser GPL gives permission to distribute the software under any conditions to any other parties as long as they are given right to modification for using and debugging purpose[5], [4].

- **Apache License:** Apache license permits us to use Apache licensed libraries in our program, being the constraint that we need to include a copy of license document in the software which we distribute and also have to provide attribute to Apache Software Foundation. It doesnt require us to include any Apache software itself in the software we redistribute. We also need to add prominent notice in any modified sources to indicate that it has been modified. It is more of liberal in nature[7].

- **Pastry License:** Pastry license [3] allows us to use, redistribute in any source or binary form with or without modifying with the following conditions:

  - Redistribution of source code must contain Free Pastry license copy and disclaimer contained in the license.
  - It allows to include the name of the RICE University (Provider of the Pastry software) or the name of the contributors to promote our software.

- **Oracle Binary Code License Agreement :** Oracle Binary Code License Agreement is applicable to all the products of Java SE Platform and any components which is proprietary part of oracle. According to the BCL Agreement [6] it states that *" Oracle grants a non-exclusive, non-transferable, limited license without fees to reproduce internally and use internally the Software complete and unmodified for the purpose of designing, developing, and testing your Programs"*.

However to distribute the Software , the Supplementary Agreement of Oracle BCL[6] states that, Oracle grants a non-exclusive, non-transferable, limited license without fees to reproduce and distribute the Software, provided that :

- you distribute the Software complete and unmodified and only bundled as part of, and for the sole purpose of running, your Programs,
- the Programs add significant and primary functionality to the Software,
- you do not distribute additional software intended to replace any component(s) of the Software,
- you do not remove or alter any proprietary legends or notices contained in the Software,
- you only distribute the Software subject to a license agreement that: You dont use Commercial Features to run the program ; Software is used for the internal development, design and testing purpose and
- you agree to defend and indemnify Oracle and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software'.

- **Eclipse Public License-v 1.0:** Eclipse Public License grants a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form. This whole section is based on content from [2].
  Few terminologies mentioned in the Commercial Distribution of the License are as below: **"Contributor"** means any person or entity that distributes the Program. **"Licensed Patents"** mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program. **"Program"** means the Contributions distributed in accordance with this Agreement. **"Recipient"** means anyone who receives the Program under this Agreement, including all Contributors. Eclipse Public License-v 1.0 [2] states the below requirements : A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

  1. it complies with the terms and conditions of this agreement; and
  2. its license agreement:
     - effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
     - effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
     - states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
     - states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

  When the Program is made available in source code form:

  1. it must be made available under this Agreement; and
  2. a copy of this Agreement must be included with each copy of the Program. Contributors may not remove or alter any copyright notices contained within the Program. Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution. It also states that in case if the product is commercially distributed then it will be sole responsibility of the commercial contributor(The product seller).

5 Licenses in Current Scenario

In our project, "A Peer-to-Peer Framework for Social Networks" there are number of java libraries and APIs which are being referred. Each of these libraries are governed by their own license as defined by the author of that library. So a particular java package might be using two or more libraries or APIs. Hence a particular java package might be under scanner for different types of licenses. It is very much required to determine the final license which implies to the package as a whole when it is governed by more than one license.

In general, the software system will consists of numerous packages, with each package using more than one library. Therefore it is difficult analyze the final or total license that implies to the software system. In order to license our final software product it is necessary to identify

and analyze the type of the license associated with each of the library and the packages using this library. The resulting license of the final software system will be based on these individual license involved. The figure 1 shows the list of libraries , their type of license and the packages using them. Most of the libraries use Apache v2.0 license

LIBRARY	Contained Package	LICENSE
guava-10.0.1.jar	com.p2pframework	Apache V2.0
commons-io-1.4.jar	org.lifesocial.core.filesystem	
commons-beanutils.jar	org.lifesocial.ui org.lifesocial.plugin.wallplugin	
jcommon-1.0.17.jar	quap2p.dims.util.ds.osgi.core	
log4j-1.2.15.jar	quap2p.dims.util.ds.osgi.core	
servlet.jar	quap2p.dims.util.ds.osgi.core	
xerces.jar	quap2p.dims.util.ds.osgi.core	
nebula_compositetable.jar	com.sign.ui.game org.lifesocial.ui.friendsgraph org.lifesocial.ui org.lifesocial.ui.friends	ECLIPSE Public LICENSE
CoDec-build17-jdk13.jar	com.p2pframework	GPL
FlexiProvider-1.7p3.signed.jar	com.p2pframework	
xmlpull_1_1_3_4a.jar	rice.freepastry com.p2pframework org.lifesocial.core.main	Lower GPL
jfreechart-1.0.14.jar	quap2p.dims.util.ds.osgi.core	
swtgraphics2d.jar	quap2p.dims.util.ds.osgi.core	
FreePastry-2.1.jar	com.p2pframework	PASTRY
xpp3-1.1.3.4d_b2.jar	rice.freepastry com.p2pframework	
FreePastry-2.1-docs.zip	rice.freepastry org.lifesocial.core.main	
FreePastry-2.1-source.zip	rice.freepastry org.lifesocial.core.main	
pastry.jar	rice.freepastry org.lifesocial.core.main	
TableLayout-bin-jdk1.5-2007-04-21.jar	quap2p.dims.util.ds.osgi.core	Oracle BCL
mailapi.jar	rice.freepastry org.lifesocial.core.main	

Figure 1. List of Libraries and their corresponding license type in "Peer-to-Peer Framework for Social Network" project.

and Pastry license. Other than these two we have libraries which are covered under GPL (General Public License), L-GPL(Lesser GPL), EPL(Eclipse Public License) and Oracle BCL. So our software system is composed of components governed by all these licenses. Some licenses may impose certain kind of restriction on derivative works., for example: GPL says that any derivative work of GPL must also be a derivative work. So to keep clean prospects for the final project to be marketed, it is better to restrict ourself from including libraries using GPL and Oracle BCL. Since our current project "Peer-to-Peer Framework for Social Network" uses libraries covered under GPL and Oracle BCL the concern is to determine alternative for them. In this regard the libraries for which alternative has to be determined are :

- CoDec-build17-jdk13.jar,
- FlexiProvider-1.7p3.signed.jar,
- mailapi.jar and
- TableLayout-bin-jdk1.5-2007-04-21.jar.

Not to the least, dynamic linkage of libraries and careful decisions has to be taken in future project implementation phase so that licensing issue is taken care. The figure 2 depicts the percentage wise different types of licenses involved in the project. For instance, Apache v2.0 license is

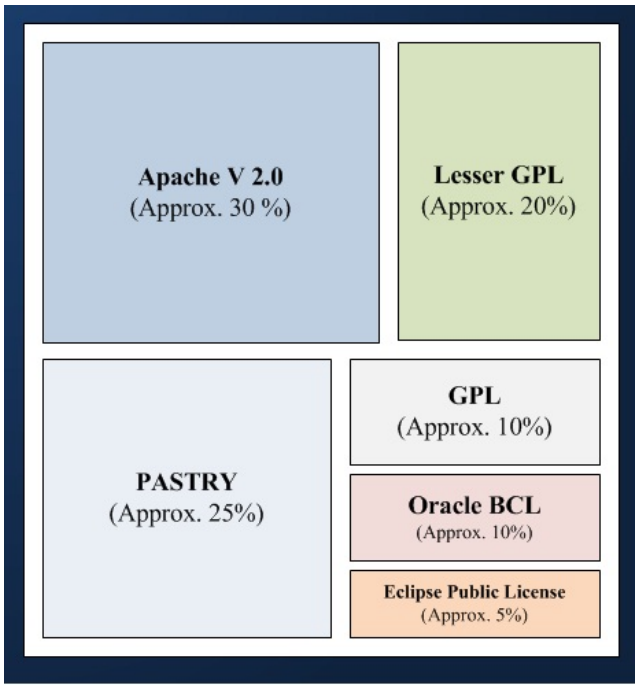


Figure 2. Block Diagram showing the percentage of libraries in the project using a particular license.

used by approximately 30 percent of the libraries involved in the project, where as GPL is 10 percent and Oracle BCL is used by 10 percent of libraries. Hence to make our product commercialized we have to take care of 20 percentage of the existing libraries as well as in future must use license compatible libraries. And also the compatibility with the JRE (Java Run Time) also has to be taken care , as the current license would hold only for developing, designing and testing for internal and non commercial purpose. In broader aspect, the Application Server which hosts this project has also to be brought under the scanner for license compatibility. Hence it is required to understand in a collective manner keeping all the constrains in mind before designing or redesigning the architecture of the software system so that it would not ultimately result in any license complexities.

6 Compatibility of Licenses

Figure 3 illustrates how sources from different open source system and commercial products with different license types, can be involved in creating a proprietary system [14]. As discussed in earlier sections it is very much necessary to identify the compatibility between different license types used by different components at static as well as dynamic times i.e. during compilation and as well as at run time. The implication of license may be totally different at static(compile-time) and dynamic time. Therefore, while designing or re-designing our framework we need to take care such that interaction between components at static and dynamic time won't change the final resultant license leading to incompatible license to use or merchandise our product. In the previous section the libraries which were listed for determining alternatives, were examined on the basis of static linking. Generally, library license issues are considered from the static view point as in most of the cases libraries are linked at static or during compilation time. As our vision is to build a final product with our own custom license, it is necessary for us restrict libraries with GPL and Oracle BCL. GPL does not permit to have restrictions in license which exceed the GPL terms. Hence license identification plays an important role in determining the final resultant license. While identifying the license for a component or a library specific version must verified, because software license evolve and there are chances that the license used by the library is different from the one which is distributed[14]. Figure 2 shows approximate percentage of libraries in the project using a particular license in comparison with the all the other license used by other libraries. To check the compatibility of the involved license it is required to manually identify the license type of each libraries or components of the software system and finally validate for compatibility with other interacting components license. This can be done by a team member who has expertise in both technical as well as legal aspects of the software system [14]. In the current scenario of our project to achieve compatibilty, for whatever incompatible-libraries that have been identified at static time; replacing the library or a solution can be found at implementation phase. At implementation phase, there are chance of encountering further libraries with incompatible license as a concrete interaction can be know only during implementation stage. Therefore, at implementation stages it is required to find solution for the current incompatible problem as well as to include new libraries only with a proper compatible license.

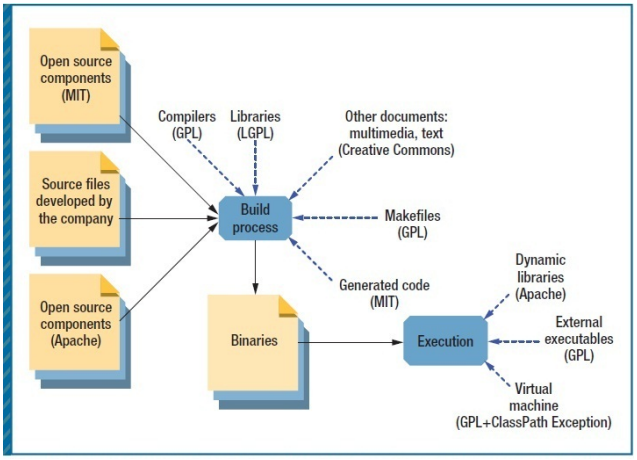


Figure 3. A proprietary software system might involve different source systems with many different source licenses[14].

7 Conclusion

In software systems with many components, which are governed by different types of license makes merely impossible to meet the criterions of all the licenses. This problem is referred as License Incompatibility Problem. A probable solution for this was to relicense the component code into a set of disjunctive licenses and allow the licensee to choose one of them based on the requirement. In another case when Oracle Inc, considered to license Java under GPLv2, concerns aroused that any Java program could be considered that it was a derivative of Java SDK (Software Development Kit). To provide a solution for this, The Free Software Foundation came up with a new concept of Classpath Exception, i.e. whenever Java library is licensed under GPL2v with Classpath Exception, the program that links to this library can be considered as a derivative work of the library, but the program can be distributed with any license (Open source or not)[13], [14].

In our project "Peer-to-Peer Framework for Social Network", the incompatible license which has to be taken care is GPL and Oracle BCL. The libraries or components using these licenses is 20 percent. Therefore it is required to find alternatives to these libraries at future implementation stages. Considering these modifications the resulting software system would be composed of libraries or components with compatible license.

References

[1] SDKs and APIs : DO they have copyright protection ?, 2012 (accessed May 25, 2012). http://www.ipblog.ca/?p=556.

[2] Eclipse Public License-v 1.0, 2012 (accessed May 25, 2012). http://www.eclipse.org/org/documents/epl-v10.php.

[3] Free Pastry Peer-to-Peer Application Development Substrate, 2012 (accessed May 25, 2012). http://www.freepastry.org/FreePastry/license.htm.

[4] Frequently Asked Questions about GNU Licenses., 2012 (accessed May 25, 2012). http://www.gnu.org/licenses/gpl-faq.html.

[5] GNU Lesser General Public License. Version-3, 2012 (accessed May 25, 2012). http://www.gnu.org/licenses/lgpl-3.0.html.

[6] Oracle Binary Code License Agreement for the Java SE Platform Products and JavaFX, 2012 (accessed May 25, 2012). http://www.oracle.com/technetwork/java/javase/terms/license/index.html.

[7] The Apache Software Foundation Version 2.0, 2012 (accessed May 25, 2012). http://www.apache.org/licenses/LICENSE-2.0.html.

[8] The Free Software Foundation. GNU General Public License., 2012 (accessed May 25, 2012). http://www.gnu.org/licenses/gpl.html/.

[9] Groklaw : Oracle vs Google- Trial Briefs, Special Verdict Forms, and Jury Instructions, 2012 (accessed May 28, 2012). http://www.groklaw.net/articlebasic.php?story=20111017164938222.

[10] Update on Oracle vs Google , 2012 (accessed May 28, 2012). http://www.ipblog.ca/?p=644.

[11] T. A. Alspaugh, H. U. Asuncion, and W. Scacchi. Analyzing software licenses in open architecture software systems. In *In Proceedings of 2nd International Conference on Emerging Trends in FLOSS Research and Development (FLOSS)*, May 2009.

[12] H. W. Classen. *A Practical Guide to Software Licensing for Licensee and Licensors*, chapter Grant of License, pages 21–60. American Bar Association, 2nd edition, 2005.

[13] D. M. Germán and A. E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *In Proceedings of International Conference on ICSE*. IEEE, 2009.

[14] D. M. German and M. D. Penta. A method for open source license compliance of java applications. *IEEE Software*, 29:58–63, 2012.

OPPORTUNITIES OF INTERACTION SUPPORT FOR ONLINE COLLABORATION

Shyam Sandeep Polineni
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: polineni@mail.upb.de

Abstract

The Peer-to-Peer system enables the sharing of the information among different entities in a network by using a simple client and server architecture but the special feature that makes them different is that there are no centralized servers or specific clients assigned in the whole system instead they collaborate with each other like peers for sharing of resources(music, videos etc.). However the sharing of certain files like different texts, documents and draw boards pose certain problems during the online collaboration among the peers. For instance while sharing documents like a draw board which is being shared among different peers the changes made by one peer will not reflect the change in other members who shared the same document. This paper gives a brief description about how documents, texts and draw boards create problem in online collaboration and opportunities of sending text, document files.

1 Introduction

Computers are communicating with each other by using peer-to-peer network since earlier days. Computers in peer-to-peer networks are called peers[8]. The computer only act as client and server there is no special server to run peer-to-peer networks. Any commercial and non-commercial data can be shared in this network. Many companies are using the peer-to-peer networks, several companies are working for development of peer-to-peer networks. Microsoft, IBM, Intel and other companies are also involved for the development of peer-to-peer standards.[8] Opportunities of interaction in a collaborative social network are limited, as there no sufficient awareness about the benefits of these networks. Interaction on common documents, calendars, text, draw boards is served by social networking websites. The main challenge is to edit the text document online in real time situation. Although there are some applications are allowing the online editing like adobe acrobat but not P2P technique. There are many applications providing P2P sharing the all kind of documents, files, etc. Finally there is a need of P2P technology with online collaborative changes.

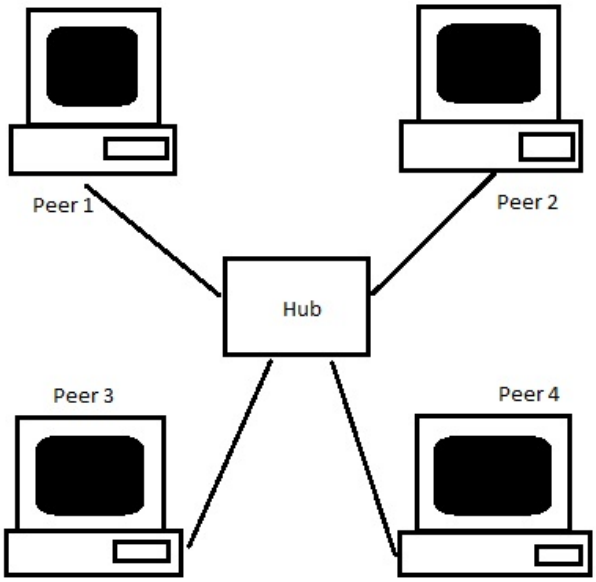


Figure 1. hub

The Figure 1 shows, group of 4 peers are contacted with each other using a hub.

The main structure of the paper is

- Online collaboration.
- How to share the files in peer to peer networks
- Interaction support
- Opportunities of online collaboration
- Advantages
- Problems faced in sharing files in online collaboration.

2 Integrating Peer-to-Peer collaboration:

Peer-to-Peer interaction may occur spontaneously to design delivery to include opportunities and activities that require collaboration between learners [1].Many of the tools provide operations to work together.[1]

2.1 Co-authoring:

Learners can use collaborative tool to discuss and edit the content that they have done. Collaborative problems solving and collaborative research: Learners who are working together to develop an application then they will discuss the problems that are facing in approaching tasks and solve the problems by communicating each other.

2.2 Mentoring:

Most of the popular persons, professors, experts and more experienced members can act as guides for specific learners who are learning their researches.

2.3 Online peer review:

Learners will submit their information in the network, so that other learners who are doing the same research can discuss on it and give their comments regarding on it and lots of discussion can be done for getting the better results.

2.4 Resource making:

Learners who have done their research in particular study will post their subject in online, so that other new learners who want to do research in same topic may use the previous papers of that research. This resource is mainly useful in this type of sharing.

2.5 Representing a document via tree

By viewing the shared document as hierarchical or multiple level structure.[3] We are able to reduce it in smaller structure, so that we can reduce communication and computational costs.[3] According to the documents there are nodes or structure, the structure of the paper is defined by sections, subsections, paragraphs, sentences, etc.[3] In the paper the main structure will be mentioned as main tree root and subsections of the paper are referred as leaf nodes. Number of leaf nodes can be appeared, because in papers there will be many subsections in the main section. And the length of the leaf nodes is maximum. The length will be adjusted by using algorithms. Sentences, paragraphs which are related to the documents are arranged in tree-form, so that length, cost and time are reduced. And is very easy to understand rather than making complicated matters dividing in sections and sub sections.

3 Features of peer to peer network

3.1 Online collaboration:

Currently peer-to-peer networks are playing key role in sharing the different types of content like audio and video files. As the same like audio and video files it can share different kinds of documents, files, draw-boards, etc by using these networks. At present Facebook like applications are very popular in sharing pictures and videos and communication. And in Skype peer-to-peer framework will be used in screen sharing tool. Different people will communicate from different locations and main advantage of this p2p network,there is no need of special server, every system in the network acts as a client-server architecture[2]

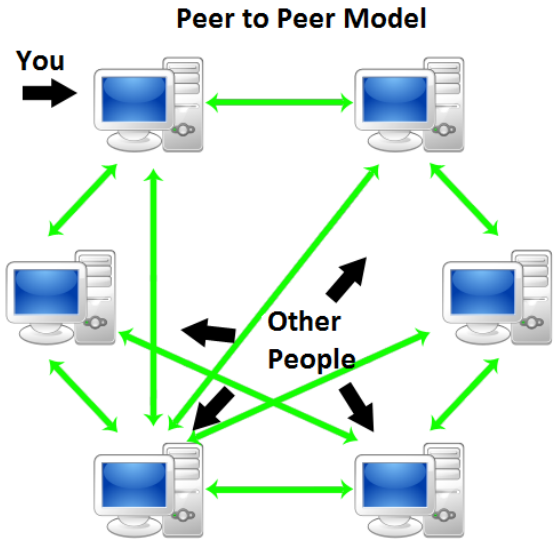


Figure 2. peer to peer model

The Figure 2 shows how the file sharing get started from one system to other within the network.

The term P2P refers to peer-to-peer networking. A peer-to-peer network allows computer hardware and software to function without the need for special server devices. P2P is an alternative to client-server network design.[4] Peer-to-Peer network there is a person to person sharing by online collaboration, we can upload file and that can be downloaded by all the network members. By using peer-to-peer networks we can share any number for files in the network, in the same way we can share draw boards, documents can be shared as like the audio and video files. Finally P2P acronym has acquired a non-technical meaning as well, some people have described this second meaning of P2P is people-to-people. For this perspective, P2P is a model for developing software and growing business that help individuals on the internet meet each other and share common interests. So called social networking technology is an example of this concept.

3.2 How to share the files in peer to peer networks?

P2P is a popular technology for file sharing software applications like Kazza, WinMx and Overnet, etc.[6] P2P technology helps the P2P client applications upload and download files over the P2P network services. Users can use software that connects in to a peer-to-peer network to search for shared files on the computers of other users connected to the network. Files of interest can then be downloaded directly from users on the network. Typically, large files are broken down into similar chunks, which may be obtained from multiple peers and then reassembled by the downloader. Peer-to-peer applications allow you to mutually share your files with others, but besides the benefit of access to a wealth of information, they can also put you at risk: you could download copyright-protected material, pornography, or malware (viruses, spyware and Trojans) without meaning to or you could mistakenly allow other people to copy files you donot mean to share.[6] There is no legal right to distribute music even if you are paid for the p2p application or for a copy of the music (such as CD that you copied onto your computers hard drive).[2] For protecting your data, configure file sharing software carefully, so you know what is being shared.[8] Check the proper settings so other users of the file sharing network should not have access to your private files, folders, or sub-folders. Disable your file sharing applications when you are not actively using them. Locate and disable the option for the application to start automatically boot up.[6] File sharing software and operating system should be up-to-date. P2P technology can also be found other places.

3.3 Interaction Support

By interaction support people can communicate each other. In this lot of information can be stored and they can argue, solve and discuss about the problem in the topic which was shared. Different people can express their feelings about the documents, draw-boards etc. therefore for good results and genuine opinion this interaction support is necessary. Any kind of development can be done by interaction. Bidding can be done for our documentation by interaction and faults can be justified and solved. By participating in interaction support one can know the mistakes that are done during development.[5]

The Figure3 shows the interaction between differnt systems.

4 Opportunities of online collaboration:

The main opportunity of Online Collaborative network is to get connected with people. In organization like all members can communicate and collectively work with each other in this online collaboration.[1] This results a innovative creations and thoughts and can be improved in this p2p

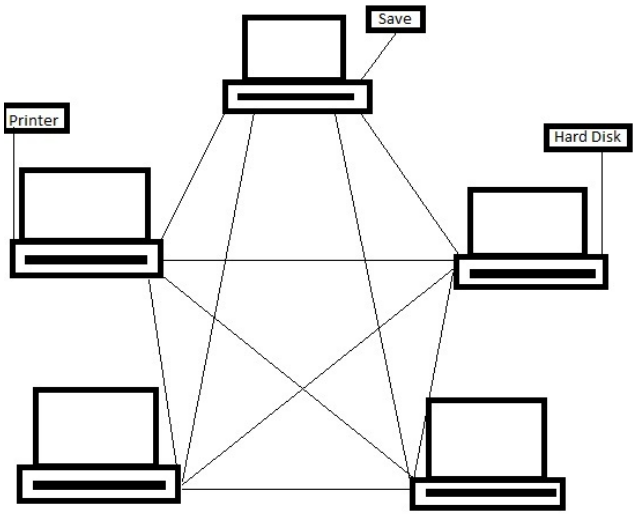


Figure 3. Interacting system

network, comparison of good and service, also plays a vital role in these networks and get the better results in the real life. By this there is a main opportunity that the travel costs can be reduced means of communication, thus developing a application which allows to round table conference and board meeting or business meeting, among the group of branches located in different countries, participates in the discussions.

4.1 Collaborative learning

A group of students can work together with collaborative learning by discussing, arguing and evaluating anything . The collaboration is done for solving a task; this can be solved by reporting, slides or notes, opinions of different members in that group. But not every group is considering as collaborative learning.

4.2 Knowledge sharing in communities of practices

[9] The concept of communities of practice (cops)[9], describes informal group of people who create, share, and explicit their knowledge and experience [9]. These are self-organizing structures, for any group of members can organize these cops like software, families, organizations etc. Group of members will join in their community to share the files, documents etc .for their use. These communities will strengthen the levels by views of different members. In knowledge sharing, communities of practices will filter the information, in many groups or organizations the members in the group will post their knowledge by adding unstructured information. The cops can separate the unstructured information from structured one, cops will behave as a engine for knowledge sharing from separating unstructured information.

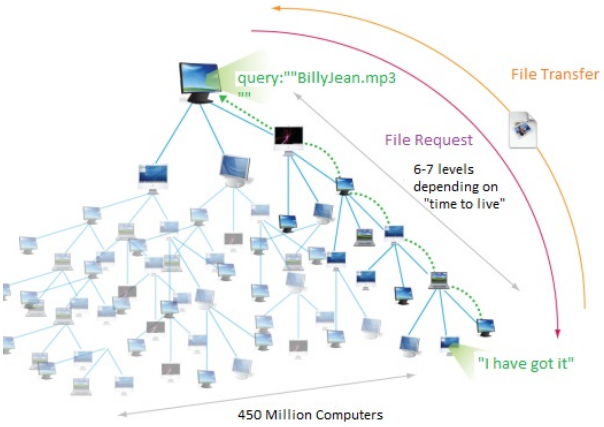


Figure 4. file transfer

The Figure 4 represents the prototype search engine model, locating the file and transferring to the user system.

4.3 Spreading Ideas through Draw boards

One can share his ideas by giving a pictorial representation thus using draw boards. These draw boards are easy to create and understand, these include sketches, artwork, painting etc.

4.4 Collaborative consumption:

It is used to share, swap the information i.e. changing the information and goods and services are directly exchanged in this process. Trading is done in this process.

4.5 Consumer- to- consumer

In consumer-consumer a new marketing technique can be developed where there will be facility for consumers to place items for auction and other consumers can place bids for the same.

4.6 Peer-to-Peer banking:

In banking system it allows individual to do the financial transactions of their own accounts, they can transfer the amount to the particular person. Loans can be offered up to a specific level for specific interest.

4.7 Peer-to-Peer car rental

This network is used to know about the information of cars for rent. The car owners will mention rental amount and rental period and customers who like the deal will contact him. This is all can be done in this network.

4.8 Peer-to-Peer property rental:

In this the owner of the particular house may post the details of the house for sale or rent in the network. The interested person will contact him and discuss about the rent that he is expecting. For any type of renting this p2p network is very helpful for communicating people directly in websites and knowing the information that they are needed. This is one of the great opportunities for group of people to interact with each other.

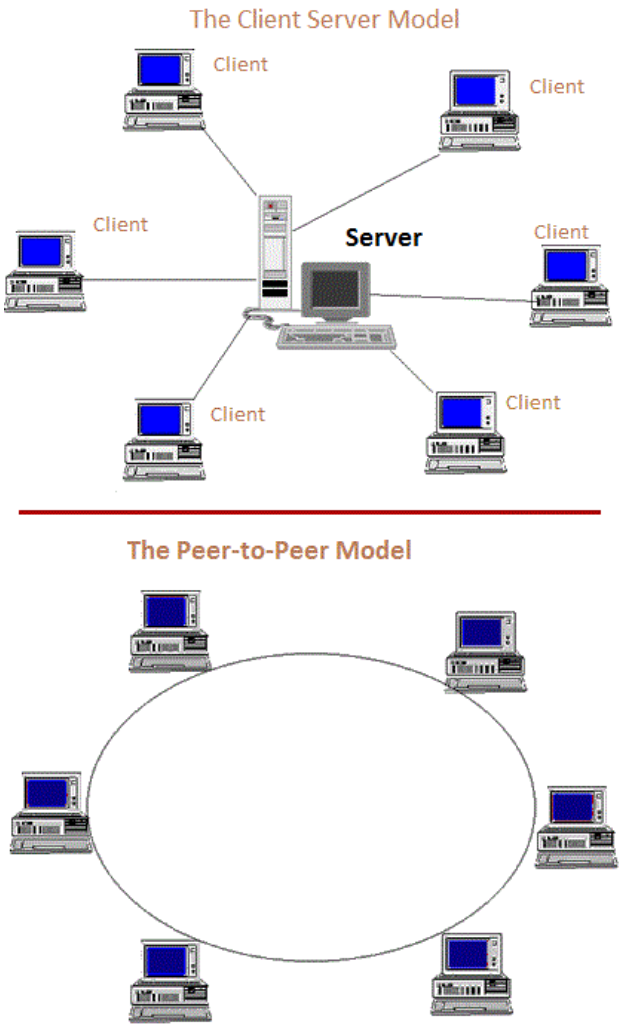


Figure 5. client-server model

The Figure 5 shows, the difference between client server architecture model and peer to peer architecture model.

4.9 Knowledge sharing:

The subjects that are taught by professors can be shared in this network, any kind of knowledge based materials can be shared for e.g. subjects, seminar papers, materials that are not available in books and any other needed valuable information can be uploaded and shared in p2p network. This can be done by using documents, videos, draw boards etc.[1] Any new scientific, research, space inventions that are done can be shared in p2p networks, so that people are

aware of latest developments going on around the World. Anyone can communicate and discuss about these inventions and can give their valuable suggestions.

4.10 Warning about Natural calamities:

Weather reports that are forecasted in different regions can be shared in P2P networks and Natural calamities that may occur in any place can be intimated by using these p2p networks.[1] Thus we can minimize the impact of those Natural calamities and also save many people. we can also monitor the impact of a Natural calamity by using P2P file sharing system. To evaluate the impact of this particular calamity we can analyze the application and network data from peers located in a P2P network. The use of online collaboration tools is not widespread despite the technology being established and offering significant benefits. There are many opportunities with online collaboration. By selecting a group of people in our network and we can share the information to all of them. They have a chance to justify whether the information is varying or not and can be discussed in the collaboration. In draw-boards most of the information can be show in a diagrammatic form. Using drawing board to create a simple drawing and invite friends and family to vote for it. Lots of information can be transferred by a single diagram. Various kinds of p2p networks are there in todays life. Torrents like web sites are very famous in now-a-days, we can upload and download files very easily. Lot of documents, files etc. information can be shared here. Sending files through emails to many people is very difficult.

5 Discussion

5.1 Advantages of interaction support for online collaboration in p2p:

- In p2p framework by using interaction support, information can be shared with different people in form of documents, files, draw-boards etc.
- Group of people can have a discussion about their feelings, argues, drawbacks etc.
- There is no need of special server. Here every system in the network act as client-server architecture.
- Uploading and downloading files is very easy in this process when compared to other process.
- Uninterrupted downloading and sharing due peer to peer architecture.
- Any kind of file can be shared.
- Knowledge sharing.

These are the main advantages by using p2p framework for social networks in online collaboration.

5.2 Problems faced in interaction support for online collaboration in p2p:

In fact there are some draw backs while sharing the information to group of people.

- Misuse the opportunity by sharing illegal files.
- Unknown member can download the files there is no security.
- In group of people , chances made in the file by one member cannot be identifed by the remaining members in the group.
- Virus can be spread.

6 P2P security:

Security is essential for every system, and it is very important in p2p systems for protecting the secure data that has to be stored in it. The main topics of the security are [7]:

- The need for security.
- Consequences of poor security,
- Current security methods.
- Future security.

6.1 Need for security:

The main security for this p2p systems are connection control, operation control, anti-virus and the protection of the idea stored on systems[7]. The following diagram shows the security process.

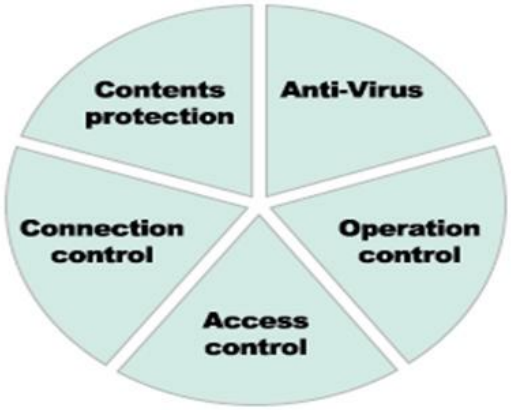


Figure 6. security
[7]

The Figure 6 represents the various methods of security in p2p file sharing.

6.2 Security key techniques:

In this technique the sender and receiver should have a key for sharing a file. If the person who want to download that file should have the password to download it[7].

6.3 P2P security in future:

In future all users should have login facility, and sharing file should be in genuine version so piracy content files canot found in the network.

7 Conclusion

In p2p systems there are many advantages instead of disadvantages; by using p2p networks we can share audio and video files, text documents, draw-boards and many more. In general P2P networks are not widely used due lack of awareness. If we can educate the people about the P2P networks then there will be usage of the networks. The only way to educate the people about P2P networks is, implementing it as a subject for Bachelor Course and conducting open Seminars. Many of the sites in p2p are illegal, it have to be done legal so many of the people in different countries will get benefitted. Sharing illegal files should be banded to safe guard future of these networks.

References

[1] Designing an implementing e-learning. http://designing.flexiblelearning.net.au/gallery/support/peer$_{t}o_{p}eer.htmyear$: 2012, .

[2] CIS:computing and information services. 2010. http://www.brown.edu/cis/policy/p2p.php.

[3] S. Aluru, M. Parashar, R. Badrinath, and K.Prasanna. P2p document tree management in a real-time collaborative editing system. In *Lecture Notes In Computer Science*, 2007.

[4] T. Bradley. Peer-to-peer(p2p)network security. 2012. http://netsecurity.about.com/od/newsandeditorial1/a/p2psecurity.htm.

[5] B. Mitchell. P2p networking and p2p software. 2012. http://compnetworking.about.com/od/p2ppeertopeer/a/p2pintroduction.htm.

[6] B. Mitchell. Top 10 Free File Sharing Programs-Free P2P Software. 2012. http://compnetworking.about.com/od/p2ppeertopeer/tp/p2pfilesharing.htm.

[7] D. Murphy, J. Kelly, K. C. andJohn Vickery, and D. o'Keeffe. P2p security. 2012. http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p10.html.

[8] G. Vrana. Peering through the peer-to-peer fog. July 19,2001. http://edn.com/design/communications-networking/4338332/Peering-through-the-peer-to-peer-fog.

[9] C.-Y. Wang. Using peer-to-peer technology for knowledge sharing in communities of practices. 2008. http://www.sciencedirect.com/science/article/pii/S0167923607000978.

Multimedia Libraries in Java for Audio and Video Streaming

Luqman Ahmad
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Project Group - A Peer-to-Peer Framework for Social Networks
Email: ahmad@mail.upb.de

Abstract

By enabling the delivery of dynamic, interactive media content across the network, streaming media is changing the way people communicate and access information [1]. This paper will explore the Multimedia libraries in Java for live streaming for audio and video. It will discuss the working of these libraries and their positives and negatives. We will also see the alternatives of these libraries while remaining in Java.

1 Introduction

Internet growth in last decade is bigger than one could thought. With the growth of internet, the methods in software engineering are totally changed. Today softwares are more distributed and so Java has become a major language for software development on internet.

As presented in [2], very fast speed of internet and power of computers has resulted different multimedia applications. There has been growth in multimedia traffic on internet in last decade. Some of very good multimedia applications that are used over the internet are:

- video conferencing,
- media-on-demand,
- access to multimedia database over Internet,
- interactive distance education,
- collaborative computing,
- virtual reality based scene navigation,
- interactive games rich in 3D graphics, etc.

For giving the fully control over multimedia applications on internet, Java has given a set of API's called Java Media API's. Java Media API's allows Java programmers to integrate multimedia to their applications.

According to [3] Java Media API's contains:
Java 2D : Java 2D API are set of classes for advanced 2D graphics and imaging.

Java 3D : Java 3D API are object oriented interfaces to control the behaviour of 3D objects.

Java Advanced Imaging : Java Advanced Imaging (JAI) are for images to be manipulated easily in Java applications and applets.

Java Binding for OpenGL : Java Binding for the OpenGL API (JOGL) provides hardware-supported 3D graphics to applications written in Java.

Java Image I/O : Java Image I/O API provides a pluggable architecture for working with images stored in files and accessed across the network.

Java Media Framework : Java Media Framework API (JMF) enables audio, video and other time-based media to be added to applications.

We will concentrate on Java Media Framework as it is used for live audio and video streaming.

2 Java Media Framework (JMF)

The Java Media Framework is an application programming interface (API) for inoarporating time based media in to Java applications and applets. This is an optional package which can capture, playback, stream and transcode mulitapl media formats, extends the Java 2 platform. [4] [5] [1]

2.1 What JMF can do ?

With JMF we can work on any type of media but we see some main work, as presented in [2] [6] [1], that we can do with JMF:

- Present multimedia contents
- Capturing audio through microphone and video through cameras

- Enable the development of media streaming and conferencing application in Java
- Enable advance developers to implement custom solution based on existing API
- Access to raw media data
- Progress media
- store media to file
- Handle stored audio and video

2.2 Features of JMF

From references [2], [7] and [8], JMF supports popular media types Microsoft Audio/Video Interleave (.avi), Macromedia Flash 2 movies (.swf), Future Splash (.spl), MPEG Layer 3 Audio (.mp3), Musical Instrument Digital Interface (MIDI; .mid or .rmi extensions), MPEG-1 videos (.mpeg, .mpg), QuickTime (.mov), Sun Audio file format (.au extension), and Macintosh AIFF file format (.aif or .aiff extension).
It also supports popular media access protocols such as file, HTTP, HTTPS, FTP, RTP, and RTSP.

2.3 Why JMF ?

Implementation of media players are platform dependent thats why they can not be useful on web. JMF provides independency of platform so that they becomes portable across platform. JMF hides these implementations by providing abstraction to developer. So JMF provides platform neutral framework for handling multimedia. You can add in your Java applications and applets with audio, video and animations. [2] [7] [8]

2.4 Working with Time-Based Media

Any media that can be changed with respect to time is time based media like audio clips, video clips etc. This data can be taken from local file, over internet or over network.

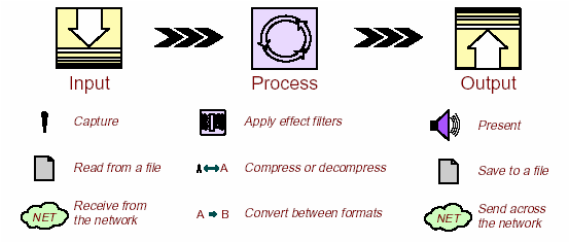


Figure 1. Processing Model in JMF [1]

According to figure 1, we see that in JMF any type of input and output is possible.

2.4.1 Streaming Media

In time based media [1], there is a time deadline in input and output. If flow of media is started, you have to meet time deadlines that is why time based media often called Streaming Media. If time deadline can not be meet at input then you can not show data on output as well that will cause delay in output or skipping data in output.

2.4.2 Media Streams

A Media Stream is media that can be get from local file, over network or from camera or microphone. Media streams have many channels to get data cladded tracks. A Media Stream having multiple tracks called multiplexed and extracting individual tracks from multiplexed called demultiplexing.
Media Streams have two categories as stated in [5]:

- Pull : When data transfer is initiated and controlled by client side
- Push : When data transfer is initiated and controlled by server side

Format	Content Type	Quality / CPU Req. / Bandwidth Req.
Cinepak	AVI QuickTime	Medium / Low / High
MPEG-1	MPEG	High / High / High
H.261	AVI RTP	Low / Medium / Medium
H.263	QuickTime AVI RTP	Medium / Medium / Low
JPEG	QuickTime AVI RTP	High / High / High
Indeo	QuickTime AVI RTP	Medium / Medium / Medium

Table 1. Common Video Types [5]

Format	Content Type	Quality / CPU Req. / Bandwidth Req.
PCM	AVI QuickTime WAV	High / Low / High
Mu-Law	AVI QuickTime WAV RTP	Low / Low / High
ADPCM	AVI QuickTime WAV RTP	Medium / Medium / Medium
MPEG-1	MPEG	High / High / High
G.723.1	WAV RTP	Medium / Medium / Low
GSM	WAV RTP	Low / Low / Low

Table 2. Common Audio Types [5]

Table 1 and Table 2 will describe the common media formats. You have to care about media formats when working on your application. For example, if you are working on an application that will be delivered on web, you have to keep in mind Bandwidth Requirements.

2.4.3 Media Presentation

As claimed in [5], Streaming Media like audio and video can be presented through speakers and monitors or it can be transmitted over the network. Destination sometimes referred as data sink.
When the destination is network then there is a delay to start the actual stream. This delay called start latency. One can experience this like when we click on start button to see stream, it takes little time to start.
When we are combining streams from different sources like audio from microphone and video from camera, we have to take care of start latency otherwise audio and video will be messed up to show strange effects.

Presentation Quality

Presentation Quality can be depend on following things as described in [1]:

- compression scheme
- processing capability
- bandwidth

2.4.4 Media Processing

In most cases, media stream is manipulated before sending it to output. Usually a series of processing is applied before sending it to user as represents in [1].
1. multiplexed streams are extracted into individual tracks.
2. compressed tracks will be decoded.
3. changing the format of tracks.
4. Effect filters can be applied.

2.4.5 Media Capture

A media stream [5] can be get from a live source for processing. An audio track can be capture from microphone and video track can be get from camera. Usually the capturing phase is referred as input phase of media processing model.

For capturing media, you need a specialized device like microphone, camera etc. A video camera can be give us audio track as well as video track. Capture device can be categories in push or pull sources.

2.5 High-Level Architecture of JMF

As it is shown in figure 2, we see in a daily life, we record the video tape from some camera and then we present media file to VCR in video tape format and VCR shows the output to television or speakers.

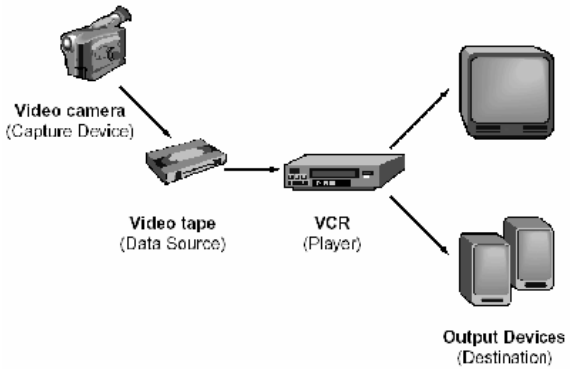


Figure 2. Recording, processing, and presenting time-based media [5]

JMF uses the same scenario. A data source encapsulate media stream like media tape and player provides processing similar to VCR. High-level JMF architecture is shown in figure 3.

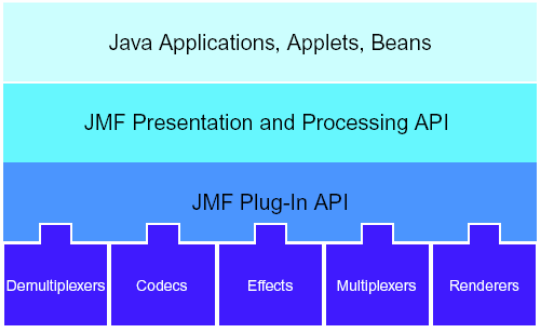


Figure 3. High-level JMF architecture [1]

2.5.1 Presentation

In JMF, the presentation process is modeled by the Controller interface. Controller defines the basic state and control mechanism for an object that controls, presents, or captures time-based media.
According to [1], JMF has to type of controllers:

- Players
- Processors

Normally players and processors used for only specific data and it can not be reused for another data.

Players
A players takes the input stream and renders it to a specific time.

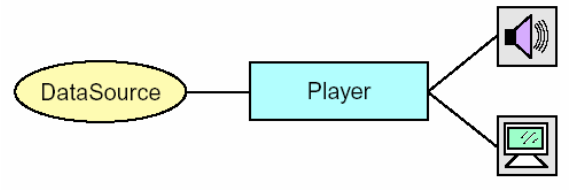


Figure 4. JMF Player Model [1]

As shown in figure 4, players take media stream from DataSource and renders it to output according to the input media stream.

Processors
A Processor is same like Player but it can also sends output to another DataSource.

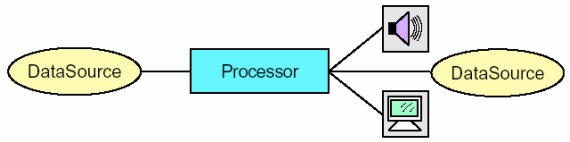


Figure 5. JMF Processor Model [1]

A Processor has all the qualities of Player plus data can be sent to another DataSource or another Player or Processor as shown in figure 5.

2.5.2 Processing

According to [5], a Processor is a Player that can be used to get input from DataSource and can send output to DataSource.Processor does some pre defined processing by developer before sending it to another DataSource or another DataSink.

Here are the types of processing on media data:

Demultiplexing
If an input stream have multiple tracks, we can parse it into single track.

Pre-Processing
With Pro-Processing, we apply the effects on parsed tracks.

Transcoding
In Transcoding, the format of each track of input can be converted to another format. Transcoding includes encoding and decoding.

Post-Processing
It is about applying effect algorithms to already decoded tracks.

Multiplexing
If multiple tracks are combined to make a single track, the whole procedure is called Multiplexing.

Rendering
It means that presenting the output to user.

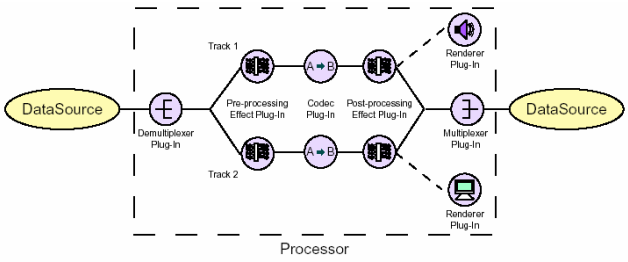


Figure 6. Processing Stages [5]

2.5.3 JMF plug-ins

The processing is done by different components called plug-ins. There are five types of plug-ins in JMF as follows:

- **Demultiplexer** Parses media streams
- **Effect** Giving the special effects on a single track
- **Codec** Data encoding and decoding
- **Multiplexer** Combines multiple data tracks into a single tracks
- **Renderer** Showing data to user as output

2.6 Working with Real-Time Media Streams

When a streaming media is presented to user in real-time, she can start the stream without downloading it fully. In many cases, the streams don't have predefined durations.
We can see streaming media in real-time now a days quite often on web like live radio, television broadcasting, video conferencing etc.
Streaming media has changed the way of peoples to interact each other, getting informations etc.

2.6.1 Protocols for Streaming Media

For transferring media data in real-time on web, we need high network throughput. There can be large delays or lost

of data. As described in [5], there are three type of protocols for transferring real-time media data.

- **TCP (Transmission Control Protocol)** TCP guarantees that data will be reached on destination. If data is lost or corrupted, it sends it again to make sure that data should be reached to destination that is why it can not be used in real-time media streams because of so much delays.
- **UDP (User Datagram Protocol)** This protocol does not give guarantee for packets to reach on destination also packets order can be different. So this also can not be used for streaming media.
- **RTP (Real-Time Transport Protocol)** Today the standard for streaming media is RTP.

2.7 Real-Time Transport Protocol (RTP)

According to [1], *"RTP provides end-to-end network delivery services for the transmission of real time data"*. RTP uses multicast and unicast network services. In unicast data is sent from one source to all destinations and in multicast data is sent from one source to network and then it is responsibility of network to send copies to all destinations. Multicasting is more efficient in video conferencing.

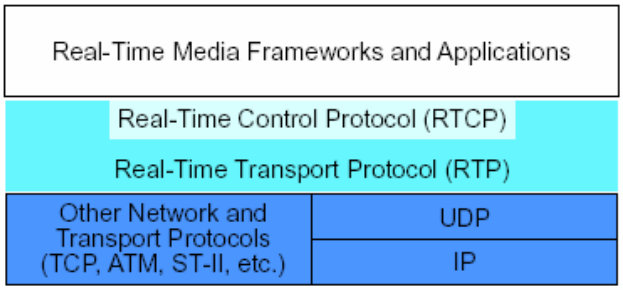


Figure 7. Processing Stages [5]

2.7.1 RTP Services

RTP provides many services, some of them as follows:

- Type of data to be transmitted.
- Determine the order of packets.
- Synchronization of media streams.

RTP does not give guarantees about the order of data packets that are sent from initial source. In fact, it does not give guarantee that packets will reach to destination or not. For this purpose, JMF use control protocol (RTCP) over RTP to ensure control and identification mechanisms.

2.7.2 Criticism on JMF

Still nothing is perfect so JMF also has some deficiencies. In [9], [2], [10], [11] and [12], criticism on JMF has been done like :

- Since the last improvement of JMF was in 2003 so it is now 9 years and new and better video codecs have been developed and the need for streaming media files over the Internet has largely increased.
- JMF is lacking in performance before native codes.
- JMF should be installed on both ends for working.
- JMF plug-ins are not easy to install specially in LINIX/UNIX platform.
- JMF only supports Flash 2.

3 Alternative to Java Media Framework

Depending on developer's use and needs, there are many alternatives to JMF that may be useful for developers. There are many alternative framework in many languages but we will see the alternatives that are in Java. Most of these are dependent to JMF and because JMF gives independence so that developer can use JMF to make another platform that will use JMF API's.
So there are many alternatives to JMF that is built on java and those are:

- Freedom for Media in Java (FMJ).
- gstreamer-java.
- DirectShow - Java wrapper.
- Xuggler.

3.1 Freedom for Media in Java (FMJ

According to FMJ (Freedom for Media in Java) official website [13], they claimed that FMJ is an open source project and best alternative to Java Media Framework. It is compatible with JMF API's. They are trying to give a single Framework which can be used to capture, playback, process, and stream media across multiple platforms.

3.1.1 FMJ Supported Media Formats

Wrapper for	OS	Formats
DirectShow	Windows	Pretty much anything!
Quicktime for Java	Mac OS X for Java	Pretty much anything!
GStreamer	Any	Pretty much anything!

Table 3. Native wrapper playback only [14]

Wrapper for	OS	Formats
FFMPEG	Any	Pretty much anything!

Table 4. Native wrapper processing and playback [14]

Container	Formats	Decode (D), Encode (E)
RTP	JPEG/RTP	D,E
	ULAW/RTP	D,E
	ALAW/RTP	D,E
	SPEEX/RTP	D,E
	ILBC/RTP	D,E
WAV	LINEAR (PCM)	D,E
AU	LINEAR (PCM)	D,E
	ULAW	D
AIFF	LINEAR (PCM)	D,E
multpart/x-mixed-replace	JPEG	D,E
	GIF	D,E
	PNG	D,E
Ogg	VORBIS	D
	THEORA	D
MP3	MP3	D

Table 5. Pure Java processing and playback [14]

3.1.2 Goals and Features of FMJ

Some goals and features of FMJ as described in [9], [14], [15], [16] are as follows:

- FMJ does not dependable on JMF.
- It has its own API's but still compatible with JMF.
- It supports all the functionality of JMF as for as supporting more codecs.
- FMJ can use JMF's plug-ins and vice versa.
- It has BSD license.
- Provide drop-in replacement for JMF.
- It provides native performance and pura Java flexibility where possible.

Type	Technology	OS
Audio	JavaSound	All
Video	LTI-CIVIL using DirectShow	Windows
	LTI-CIVIL using V4L2	Linux
	LTI-CIVIL using Quicktime for Java	Mac OS X

Table 6. Capture [14]

3.1.3 Line of Code of FMJ

Figure 8 shows the lines of code of FMJ increased with respect to year.

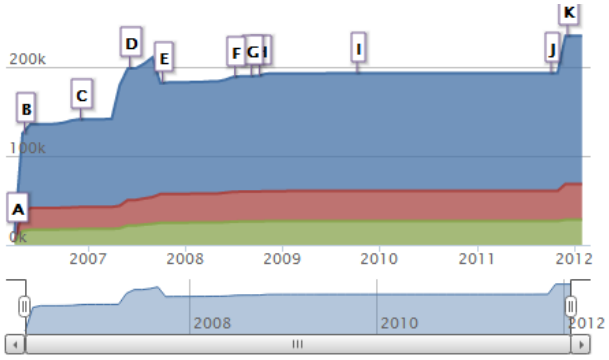


Figure 8. Line of Codes of FMJ [17]

3.2 Gstreamer - Java

Gstreamer-Java [18] claimed that this is a java binding of framework gstreamer. It Supports MacOSX, Linux and Windows. To use gstreamer-java, you should have the knowledge of Java language as well as knowledge of gstreamer framework which is working on C and Python languages.

As gstreamer-java is java bindings for gstreamer so that it has all features that gstreamer has. Let us see some features of gstreamer as listed in their website [19].

- Multiplatform.
- Comprehensive Core Library.
- Intelligent Plugin Architecture.
- Broad Coverage of Multimedia Technologies.
- Extensive Development Tools.

3.3 DierctShow - Java Wrapper (DSJ)

Like gstreamer-java DSJ is also a Java abstraction layer over Microsoft's DirectShow API. Some features of DierctShow are described in [18] and those are as follows:

DSJ Features :

- It gives media support for desktop or online Java applications.
- Works on 64-bit system.
- Control a firewire DV or HDV camera.

- Play DVD in Java.

As DSJ is a wrapper on Microsoft DirectShow thats why it has nearly all features that DirecShow has. On 2007 DirectShow gets the Emmy Award for best Streaming Media Architectures and Components.

3.4 Xuggler

Xuggler official website [20] claimed that Xuggler is an easy way to manipulate any media files or streams for Java. Xuggler uses FFmpeg to compress and decompressing media.
Xuggler uses types of API's:

- **MediaTools API :** For encoding and decoding audio and video.
- **Xuggler API :** Advanced API's for encoding and decoding audio and video.

4 Conclusion and Future

We have seen Java Media Framework (JMF) in detail and some alternatives to JMF in less details. As for as the live streaming in audio and video is needed, we can use some part of Java Media Framework (JMF) and some part of Freedom for Media in Java (FMJ). Other alternatives can be used but as they are mostly wrappers to some frameworks thats why we have to work very hard if we want to use it for audio and video streaming.

For future work, because JMF is not fully updated and does not support modern codecs, so we can combine JMF and FMJ for better solution.

References

[1] S. SunoAllah, M. Eljueidi, A. A. Hallaq, and M. Al-Serhy, "Java media framework." http://saadjomaa.com/projects/vu/ch5.pdf. [Last accessed 29-June-2012].

[2] T.G.Venkatesh, *Developing Multimedia Applications With The JavaTM Media Framework (includes JMF 2.1.1 and MMAPI)*. Cosmos Softwares, 1st ed., 2004.

[3] Oracle, "Java media apis." http://java.sun.com/javase/technologies/desktop/media/. [Last accessed 29-June-2012].

[4] Oracle, "Java media framework api." http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html. [Last accessed 29-June-2012].

[5] Oracle, "Java media framework api guide." http://java.coe.psu.ac.th/Extension/JMF/jmf2_0-guide.pdf. [Last accessed 29-June-2012].

[6] M. Mastracci, "Java media framework." http://www.slideshare.net/Ronny72/java-media-framework-matthew-mastracci. [Last accessed 29-June-2012].

[7] Deitel, "Playing video and other media with the java media framework." http://www.deitel.com/articles/java_tutorials/20060422/PlayingVideowithJMF/. [Last accessed 29-June-2012].

[8] N. Patare, "Developing internet based multimedia application then why jmf?." http://niteshpatare.blogspot.de/2010/09/developing-internet-based-multimedia.html. [Last accessed 29-June-2012].

[9] S. Zehnder, "Live P2P Video StreamingFramework," in *Live P2P Video StreamingFramework*, University of Zurich, 2008.

[10] JDJ, "Web conferencing using the java media framework (jmf)." http://java.sys-con.com/node/45832. [Last accessed 29-June-2012].

[11] J. Lobby, "Does jmf have a future?." http://www.javalobby.org/java/forums/t8551.html. [Last accessed 29-June-2012].

[12] C. Adamson, "Jmf: A mistake asking to be re-made." http://www.oreillynet.com/mac/blog/2005/12/jmf_a_mistake_asking_to_be_rem.html. [Last accessed 29-June-2012].

[13] FMJ, "Freedom for media in java." http://fmj-sf.net/. [Last accessed 29-June-2012].

[14] FMJ, "Supported media formats." http://fmj-sf.net/fmj/supported_formats.php. [Last accessed 29-June-2012].

[15] Artima, "Freedom for media in java (fmj) project." http://www.artima.com/forums/flat.jsp?forum=276&thread=164831. [Last accessed 29-June-2012].

[16] FMJ, "Roadmap and status." http://fmj-sf.net/roadmap.php. [Last accessed 29-June-2012].

[17] Ohloh, "Fmj - freedom for media in java." http://www.ohloh.net/p/fmj. [Last accessed 29-June-2012].

[18] gstreamer java, "gstreamer multimedia framework." http://gstreamer.freedesktop.org/features/. [Last accessed 29-June-2012].

[19] gstreamer, "gstreamer open source multimedia framework." http://code.google.com/p/gstreamer-java/. [Last accessed 29-June-2012].

[20] Xuggler, "Xuggler." http://www.xuggle.com/xuggler. [Last accessed 29-June-2012].

Overlay Heterogeneity and Capacity-Awareness in Structured P2P Systems

Lars Bremer
Project Group - A Peer-to-Peer Framework for Social Networks
Universität Paderborn, Supervisor: Dr.-Ing. Kalman Graffi
Email: lbremer@mail.upb.de

Abstract

Capacity awareness in peer-to-peer (p2p) systems is a widely discussed topic in academia. In this paper I focus on structured peer-to-peer systems. One property of such systems is that even in case of a system with homogeneous nodes a worst case load imbalance of a factor of $O(\log n)$ is possible. However, in reality, most peer-to-peer systems have a very heterogeneous node set. Some nodes may be strong workstations whereas other nodes are mobiles that are limited in storage, processing, and bandwidth capacity. In this paper, multiple methods to balance the load in a p2p environment with heterogeneous nodes are presented and evaluated.

1 Introduction

Peer-to-peer systems have been a heavily utilized for data transportation in the last decades. In recent years; however, software developers introduced applications that use the p2p functionality differently. Among those are applications for video streaming, communication, computation, and social networks.

Simultaneously to the development of p2p applications, mobile devices like smart-phones and tablet computers were developed. This leads to the situation where it is possible to use a variety of p2p applications on a number of mobile devices as well as on laptops and strong workstations. Users gain from this situation that allows them to use their applications in about any situation imaginable. However, this shift towards a heterogeneous environment with weak mobile devices and strong machines in the same system introduces new challenges for application developers. They have to take into account, that some users are accessing the application from strong workstation others may use a smart-phone with less bandwidth, processing, and storage capacity. A p2p system, in which each participating machine is not only consumer of a service but also provider for other machines has to take this heterogeneity into account.

One approach to achieve capacity awareness in a p2p system is the usage of hybrid p2p systems. These systems distinguish per definition between at least two kinds of nodes. One type experiences usually less load or needs to be less reliable. However, in general they do not provide functionality to fit the load in a gradient way. This means, there are only very little roles a node can play. Either it has a lot of capacity, or it has only very little capacity. Nodes with a average capacity may be either underloaded in case they join the group of weak peers or overloaded when they join the other group.

Additionally to hybrid p2p systems, capacity-aware protocols for unstructured p2p systems like [11] have been introduced. However, in this paper I will focus on structured p2p system. Such systems do not flood the network with queries, instead they use Distributed Hash Tables (DHTs) to identify data objects and other nodes. The direct lookups with the help of DHTs allow messages to reach their destination within $O(\log n)$ steps. Examples for those systems are Chord [13], Pastry [10], and Kademlia [7], which are all based on distributed hash tables as their key element.

Structured p2p systems use a DHT to spread nodes randomly among the address space. Therefore, the expected distance between all nodes in the address space is equal. This distance is closely related to the workload, in p2p systems, that do not take heterogeneity into account, which means that in such a system, each node has about the same workload. However, due to the heterogeneity of the node set, such a system is unfair and unbalanced. Nodes with little capacity may be overloaded with the assign fraction of the workload. Other nodes, however, may still have free capacity because they are stronger machines. The aim of the algorithms presented in this paper is to balance the load in a way that each node has the same resilient capacity or the same fraction of load/total capacity.

In the remainder of this paper I will state different metrics that can be used to quantify the capacity. Subsequently, four different methods to balance the load in a structured p2p environment are presented. The paper will conclude with the comparison of the fours described methods and the conclusion.

2 Metrics

To build a capacity aware p2p system, developers have to identify useful metrics that quantify the capacity, in particular the free capacity, of nodes. These metrics assign each node in the system a number to express its free capacity.

The Oxford Dictionary defines the word capacity as

> [...] the maximum amount that something can contain.

For the purpose of a capacity aware p2p system however, not the total capacity of a node is important. To spread the load evenly among all nodes it is far more important how much free capacity a node has left or what the ratio between load and capacity is. The node with the highest capacity in this context can then be described as the node with the most free capacity or with the lowest ratio load/capacity.

Each of the approaches for capacity awareness presented in this paper requests the capacity from the nodes at some point. The nodes must therefore be able calculate their own capacity. In the remainder of this section I will present multiple metrics to quantify the capacity of a node and exemplify how a node can calculate these.

Peer Lifetime When a peer is expected to leave the overlay soon, it may not be worth to transfer load to this peer any more. Stable nodes with a long expected lifetime are preferable. A mechanism to estimate the total lifetime and the remaining lifetime is given in [12].

Bandwidth The bandwidth of a node is an important metric for p2p networks that involve high data transfer such as video streaming systems. The bandwidth of a node may be determined by querying the underlying operating system of a node, or by probing the network.

Storage Capacity Storage capacity may just be the free space on the hard disk. More likely, it is the already occupied space subtracted from the maximal storage allowed by the p2p application or the user of this application. These numbers can simply be requested from the operating system.

CPU Capacity The processing capacity has the same properties as the storage capacity. The CPU capacity may be limited to a certain percentage for the application and can be requested from the underlying operating system.

Other A number of other metrics are imaginable. It is also possible to deny workload on a peer at all by setting its capacity to zero. This may be useful in case nodes are hidden behind a NAT service, for instance.

It is important to note, that each of the following systems for capacity aware load balancing can only deal with one value for a node's capacity. However, it is possible to combine multiple metrics using a linear combination. This additionally allows to factorize and therefore distinguish the importance of different metrics.

The following four sections of this paper describe different methods to add capacity awareness to structured p2p systems.

3 Hashing Assisted Load Balancing

A simple load balancing solution using a series of hash functions was introduced 2003 in [1]. The basic idea is to use k hash functions that identify k different locations. The node with the most free capacity among all nodes responsible for the k locations is assigned the load.

The load balancing algorithm is base on "the power of two choices" as described in [8]. The authors use the following example for their argumentation. When n balls are independently thrown in n bins, the maximum number of balls in a bin ends up to be in $\Theta(\log n / \log \log n)$ with high probability. This number of balls can be significantly reduced when d bins are chosen independent at random and the ball ins thrown in the one of those two, that contains the least balls. This results in a maximum load per bin at $\log \log n / \log d + O(1)$ with high probability.

This scenario is very similar to the load balancing case in a distributed hash table as argued in [1]. Plain distributed hash tables are generally using only one hash function to identify exactly one responsible node for a certain id. This step is closely related to throwing a ball into the bins. One node will end up to be responsible for $\Theta(\log n / \log \log n)$ data objects for n data objects and nodes. Instead of only having one possible node to store data at, the proposed algorithm uses k hash functions to identify k nodes that can be responsible for a certain data object. This balances the load of the nodes just as it does for the ball-bins example.

The identifier of a data object x is hashed with a series of hash functions h_i resulting in a set of hash values $A := \{h_i(x) \mid i \in [0, k]\}$. When a node wants to store x, it contacts all nodes B that are responsible for the ids in A, and requests information of their available capacity. After the requesting node received the replies it decides where to store x based on the collected data.

In a p2p system that uses the metric storage capacity, as described in Section 2, a node might decide to store x at the node $n \in B$ that has the highest ratio of *free storage/ total storage*. Additionally, only those nodes are considered that have enough free storage to store x.

The authors proposed two simple protocols for data lookup. In the first protocol the requesting node sends a query to all nodes in B. Subsequently, only the node that stores the requested data answers.

The second approach minimizes the overhead due to messages that are sent to nodes not containing x. In this

approach, each node of the set B holds a pointer to the one node $n \in B$ that actually stores x. To lookup x, a node sends the request to only one randomly chosen node $b \in B$. When b contains x it answers itself, if that is not the case it redirects the request to the node that stores x using its pointer.

This concept of redirection has not only the advantage of reducing the overhead. It also allows easy load-stealing and load-shredding. When the capacity of a node increases at runtime this node can move data from referenced other nodes and vice versa. This balances the load in a dynamic environment where the capacity of nodes or the demand of data change.

4 HiGLOB

Another way to balance the load in a structured peer-to-peer system based on histograms is HiGLOB which was introduced in [15]. Each node maintains a histogram which is used to locate under- or overloaded nodes. When such a node is found, nodes change their location to balance the overlay.

Each node in HiGLOB maintains two key components, a histogram manager and a load-balancing manager. The histogram manager collects information of the load distribution in the network. This data is used to determine whether a node is under- or overloaded. The load balancing manager is used to redistribute load when a node joins or leaves the overlay network (statical load balancing) and at runtime (dynamical load balancing).

4.1 Histogram Manager

The histogram manager collects statistical data of the load distribution in the network. This data is stored in multiple buckets by each node. A bucket stores the average utilization of a group of nodes. These groups maintain two important properties.

- All groups are disjoint. This means no node exists in two buckets of the same histogram.
- Each bucket contains one neighbor of the nodes maintaining the bucket.

An example of this model is illustrated in Figure 1 where node A maintains eight buckets, one for each neighbor (B, C, D, E, F, G, H, I). Each of the buckets additionally contains other nodes but no node is contained in two buckets.

HiGLOB uses the numbering of nodes in a structured p2p network to create the disjoint groups. A node that holds references to two other nodes a and b might, for instance, partition all nodes in the system in the following two sets. The first set consists of all nodes with $id \in [h(a), h(b))$. The second set consists of all elements with id $id \in [h(b), h(a))$, which are exactly those nodes that are not contained in the first set.

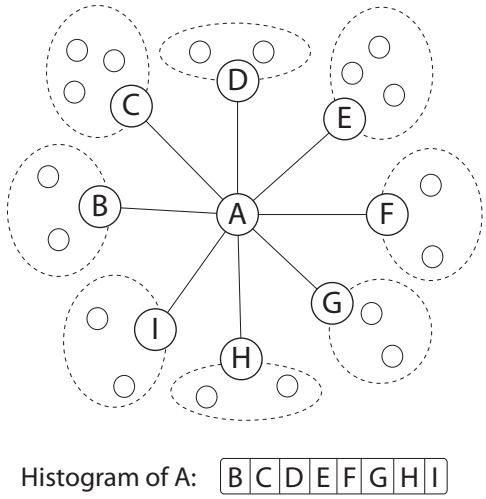


Figure 1. Example of a Histogram

The same idea can also be applied for a much larger number of references. Figure 2 shows the partition of a chord overlay with 6 nodes. Node D maintains three references in its finger table. Each of those referenced nodes are placed in a different bucket. Additionally nodes with other ids are placed in the bucket of its closest predecessor. Node G, for instance, is placed into the bucket F since this node is its closest predecessor.

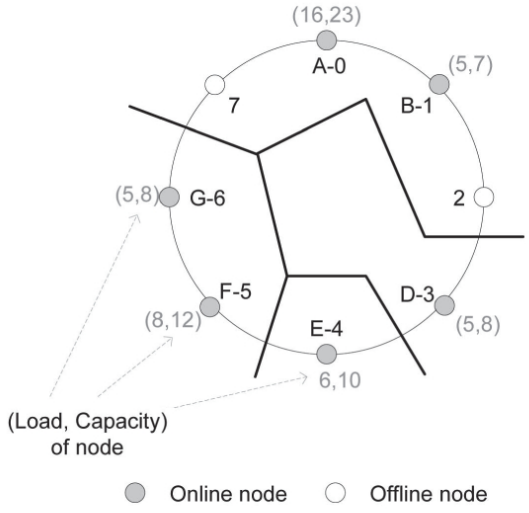
4.1.1 Histogram Creation

A node creates the histogram when it joins the overlay. To do so, it requests capacity data from each neighbor. This barely creates any message overhead since a new emerging node has to contact its neighbors anyway. The requests and the reply including the statistical data can be piggybacked in those messages needed for the initial contact to nodes in the routing table.

4.1.2 Histogram Maintenance

When the load or the capacity of a node changes dynamically at runtime the node informs its referenced nodes about the change. Subsequently, those nodes propagate the new information recursively. To reduce the amount of messages sent due to dynamical load balancing, only significant changes are propagated.

The level of significance Θ is described as the ratio between the old and the updated load value. Significant changes of these values at a node are not propagated very far due to the design of the system. The effect of the change on most other nodes in the network is too low to change the average capacity more than Θ.



Finger table of node D

start	int	succ
4	[4,5)	E-4
5	[5,7)	F-5
7	[7,3)	A-0

Histogram values of node D

.60	.65	.70
E	F	A

Figure 2. Overlay Partition in Disjoint Groups in the Example of Chord [15]

4.2 Load-Balancing Manager

The load-balancing manager is responsible to find low or heavily utilized nodes and redistributes the load. HiGLOB distinguishes between two general types, statical and dynamical, load balancing.

Statical node balancing takes place when a node joins or leaves the overlay network. A joining node creates its histogram as described in Section 4.1.1. After the process succeeded the node uses the histogram to find a heavily loaded node n and joins next to it which reduces the load at n. A leaving node, on the other hand, has to find a low utilized node to shed the load to it.

Dynamic load balancing is used when the load or capacities of nodes change significantly at runtime. A node initiates dynamic load balancing when its load is greater than twice as much as for any of the average loads in the buckets. In this case the node is overloaded. Analogue, a node is underloaded when its load is smaller than half the average load in any of the buckets.

Using this definition, a overloaded node tries to find a low utilized node from its adjacent nodes. If a suitable node is found, the node starts a process called local load balancing. This process balances the load between the two nodes without any node leaving its location.

If the local load balancing is unsuccessful, a different approach called network load balancing is executed. A suitable node is not only chosen from the set of adjacent nodes but from the set of all nodes in the network. An overloaded node finds a node with low utilization as follows. The requesting node n chooses form all buckets the one with the lowest average load. Then it sends a query to the neighboring node of n that is contained in the bucket. Such a neighbor does exist for every single bucket due to the definition of the buckets.

The query sent to the neighboring node contains a parameter to limit the search. This parameter is the id range of the identified bucket. In the example illustrated in Figure 2 the bucket with the lowest utilization is bucket E. This bucket has the id range $[4, 5)$ which would be sent as the parameter to node E.

The receiver of the query compares its own load with the average loads of the buckets that fall into the id range of the parameter. When its own load is less, it answers n and the balancing process can start. In contrast, when there is a bucket with a lower average load the node forwards the message to the neighboring node from this bucket in a recursive way.

The maximum number of steps to find a low or heavily utilized node is the maximum number of steps for a query in the overlay because the same routing system is used. Therefore it will take $O(\log n)$ steps to find a suitable node, in most structured p2p systems.

4.3 Node Movement

HiGLOB uses a different approach to balance load than seen in Adaptive Chord. The latter changes transfers the load to different nodes, HiGLOB on the other hand, transfers the nodes to different locations instead. When an overloaded node x identifies a bucket B in a bucket with an average capacity of half the capacity of x, the following three cases for node movement are distinguished by the system.

Case 1: In case bucket B contains only one element e, x compares the load of its two adjacent nodes. When the combined load of both is less than twice the load of x, then it can safely leave its position with both adjacent nodes still having less load than x did before. Subsequently x rejoins the overlay next to node $e \in B$.

Case 2: When B contains an even number of elements, then there must be two adjacent nodes with a combined capacity of less than than half the capacity of x. This is the case due to the definition of the bucket B that has an average capacity of half the capacity of x.

After identifying these two elements, x leaves its position in the overlay and rejoins between the two identified nodes.

Case 3: When the number of nodes in the bucket is uneven the set of nodes is divided into groups of two elements

plus one set with only one element. Those sets allow to execute a combined approach of case 1, where the singleton element is dealt with and case 2 for the other elements.

All cases result in a better balanced system. However, changing the position of a node for load balancing may raise the amount of churn significantly.

5 Adaptive Chord

A third way to manage the capacity in a p2p overlay is the use of a capacity overlay on top of the actual p2p overlay as it done in [14, 5].

Figure 3 illustrates such an environment using the example of a chord ring called *Adaptive Chord* proposed in [14]. Each node in Adaptive Chord resides on two overlays. One position is on the data overlay which is the standard chord overlay. The node additionally resides on the introduced capacity overlay. The position of a node on this second overlay reflects the free capacity of the node. A overloaded node can find a suitable node to shed workload using this second overlay.

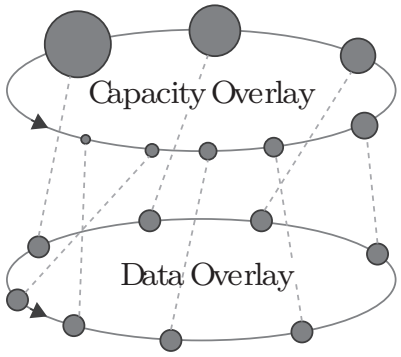


Figure 3. Example of Adaptive Chord [14]

5.1 Finding High Capacity Nodes

Nodes in this upper overlay in Figure 3 are ordered according to their free capacity. Each node that follows a node a counter-clockwise in the capacity overlay has more free capacity than node a. The node with the highest capacity has the highest id in the capacity overlay. This allows to lookup a node with a specific free capacity in $O(\log n)$ using the same lookup algorithm in the capacity overlay as in chord. However, no uniform random hash function is used to assign nodes ids on the capacity overlay. Instead, nodes are mapped to ids of the capacity overlay with a function that takes the capacity into account.

The id space of the capacity overlay is partitioned since multiple nodes may have the same capacity but the id of a node in the overlay must be unique. The function that maps the nodes to ids on the capacity overlay only returns id that mark the beginning of a partition p_0 of the ring. All nodes with that capacity are assigned an id $p_0 + rand(0, \text{partitionlength})$. To find a node with a free capacity of x, the requesting node uses the same function to identify the id of a node with requested capacity. This allows an overloaded node to find one random node of all nodes with the requested free capacity to shed load, and vice versa.

The capacity of a node may change at runtime. This may happen when a data object becomes more popular for instance. When the capacity changes by more than a constant factor in Adaptive Chord, the node changes its position in the capacity overlay. After this movement of a node, the id has to be recalculated and the successor tables and finger tables have to be recreated.

5.2 Balance the Load

Many approaches for load-balancing assign a greater id-space to nodes that are underloaded. The authors of Adaptive Chord use a different approach called *Object Mirrors*. Nodes that are responsible for the id of an object do not necessarily store those objects, they may just store references to other nodes that store the object. These nodes storing the object are called mirrors. This allows a node that has low capacity to shed the load of an object to the mirror. The responsible node itself only stores the reference and forwards requests for the data object to the mirror. The mirror, on the other hand, stores the object and provides network capacity to other nodes accessing the object.

When a node is overloaded it identifies a node with the required free capacity using the capacity overlay. Subsequently, the data object with the highest load is transferred to the identified node that will act as mirror for this object.

At runtime a mirror m may get overloaded, due to a changing popularity for its data objects, for instance. If that is the case m can contact the node that is responsible for the data object that creates the highest load and request its removal. The responsible node must then provide the object itself or find a new mirror.

6 Virtual Servers

One of the most popular methods to spread the workload in a structured p2p system is using virtual servers as it is done in [4, 3, 2] for instance. Each node in such an environment maintains a set of virtual servers to balance the load. According to [9] each of those virtual servers acts like a single peer to the overlay. However, virtual servers can be moved easily from a heavily loaded node to a node that has some free capacity.

This paper focusses on one promising protocol using the idea of virtual servers. The protocol is called Y_0 and was introduced by Godfrey and Stoica in [4].
In a system with a total of n nodes each node v has a certain

capacity c_v that is normalized. This means, the expected capacity is $c_v = 1$. An important parameter of a system using virtual servers is the number of servers per unit of capacity α. This value is not easily set since a small value like $\alpha = 1$ means that even node with low capacity need to maintain at least one virtual server. On the other hand, when the value is set too high, unnecessary servers are maintained that require references and maintenance messages which may congest the network.
The authors of Y_0 reduced the problems that emerge when a too small value for alpha is chosen by not considering nodes that have a capacity $c_v < \frac{1}{2}$. This means, each node that has a capacity less than half the average capacity in the network is not considered to maintain a virtual server. This reduces the need to choose a low value of $\alpha = \Theta(\log n)$ in order to deal with low capacity nodes. Simulations showed that only 10% of the capacity of a real world system are lost by not considering these nodes.

In many systems using virtual servers, the ids for the servers are chosen at random. This leads to a finger table size of $\Theta(c_v \log^2 n)$ for a system with $\log n$ virtual servers per unit of capacity c_v and $\log n$ fingers for each of those servers. An illustration of such a system implemented in chord is shown in Figure 4.

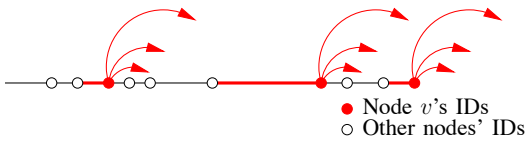


Figure 4. Basic Virtual Server Example [15]

Godfrey and Stoica reduce the number of fingers maintained by one node with the Low Cost Virtual Server Selection (LC-VSS) scheme that selects ids for virtual servers. Instead of choosing each server id randomly, only a starting point p_v is chosen. From this starting point, $\Theta(c_n * \log n)$ virtual servers are created in the counter-clockwise direction. The id of a server is chosen in a way, that each one of them has distance $\Theta(\frac{1}{n})$ between each other. This results in a cluster of virtual servers of range $\Theta(\frac{c_v \log n}{n})$.

The number of references that has to be maintained in a system using virtual servers is reduced in Y_0 because each server shares the same finger table as illustrated in Figure 5. This reduces the size needed for finger tables from $\Theta(c_v \log^2 n)$ in the basic virtual server example to $\Theta(c_v \log n)$ which is on average the same as in original chord without capacity awareness.

Y_0 reduces the storage needed for the lists of successors held by each node in the original chord design by following the same principle as done for the finger tables.

Routing in Y_0 follows the same principles as in the original chord. The only difference is, when a request reaches its destination the node is not necessarily responsible for the requested id. The reason for this behaviour is that the simulated range a node is responsible does not equal the real range. However, the real range contains all ranges of the virtual servers of a node as shown in Figure 5. Due to the clustering of the virtual servers it is possible for the node that receives the request to forward it to the actual owner of the id.

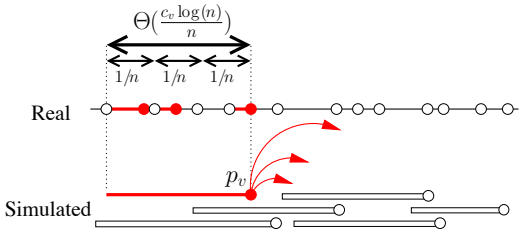


Figure 5. Virtual Server Example with LC-VSS [15]

7 Comparison

In Sections 3, 4, 5, and 6, four different systems to balance the load in a heterogeneous environment were introduced. Each of them balances the load by taking the capacity of a node into consideration. However, despite the common goal, each of the introduced systems use very different methods to achieve a fairly balanced system. The remainder of this sections compares the key aspects of the introduced algorithms and discusses their advantages and drawbacks.

The load balancing algorithm based on multiple hash functions presented in Section 3, balances the load only by giving an object multiple nodes to be stored at. A node that wants to shed workload or store a data object requests capacity information of all nodes identified by multiple hash functions. The replying node with the highest free capacity is then chosen to take the load. This behaviour distributes the load in the p2p overlay and does not require any storage overhead at the nodes. A lookup takes $O(\log n + k)$ steps, where k is the number of hash functions. This overhead exists since each node has to be queried for the requested object. This kind of hashing assisted load balancing is summarized in Table 1 as version (a). However, an additional reference to the actual owner of a data object can be introduced to reduce the lookup costs (b). The new lookup takes only one step more than in the original p2p system but the storage overhead rises.

HiGLOB uses a histogram based approach to find underloaded nodes. This approach introduces some storage overhead at each node to store the statistical data. However, this is only a constant number of averaged capacity informa-

	Storage Overhead	Maintenance Overhead	Lookup Overhead	Overload Reduction
Hashing Assisted LB (a)	none	none	one additional step	Maximum load reduction from
Hashing Assisted LB (b)	references for objects	none	k additional steps	$\Theta(\log n/\log\log n)$ to $\log\log n/\log d + O(1)$
HiGLOB	statistical data	$O(\log n)$	none	83% reduction of overloaded nodes
Adaptive Chord	second overlay references	doubled	none	85% reduction of overloaded nodes
Simple Virtual Servers	$O(k * \log n)$	k times	none	no data
Virtual Servers (Y_0)	none	none	$O(1)$	Load imbalance ≤ 3.9

Table 1. Comparison of Approaches for Capacity Awareness

tion per bucket. Each node does only maintain one bucket per neighbor, the overhead introduced is therefore not more than the storage needed to maintain the references to neighbors at a node. More significant is the message overhead introduced to keep the histogram updated. This action creates an overhead of $O(\log n)$.

The hashing assisted load balancing does not require any maintenance as described before but it introduces some lookup overhead. HiGLOB does not change the lookup process and the lookup time of the underlying p2p system is therefore not influenced. Simulations run by the authors showed a significant reduction of overloaded nodes by 83% in a chord system.

Adaptive Chord uses the approach of a second overlay, the capacity overlay. In the hashing based load balancing, data objects are moved to suitable nodes. HiGLOB moves the nodes to suitable positions. In adaptive chord a node stays responsible for a data object even when the object is hosted by another node.

The capacity overlay essentially doubles the references maintained at a node. Additionally the number of maintenance messages is also twice as much compared to p2p systems without capacity awareness. However, adaptive chord does not introduce any lookup overhead because the system uses only the data overlay for lookups, just like it is done in the original chord.

One drawback of adaptive chord is that it does not allow nodes to have no load at all. Even a node that signals 0 capacity is still obliged to forward requests to the host of a data object due to its load balancing approach using object mirrors.

In simulations of 1000 nodes the authors of adaptive chord showed that 160.4 nodes were overloaded in the original chord while in adaptive chord only 24.5 nodes were overloaded. This is a reduction of about 85%.

The simple virtual server approach introduces a storage overhead of $O(k * \log n)$, when the average node maintains k virtual servers, for references to other nodes. The virtual servers introduce a communication overhead that is k times higher than in a capacity unaware system.

Y_0 on the other hand reduces this overhead using the clustering as described in Section 6. The paper shows that Y_0 creates a load imbalance of only 2.7 to 3.6 in the simulations.

Each one of the presented systems have advantages and disadvantages. Introducing a capacity overlay as it is done in Adaptive Chord, for instance, involves a great developing effort and creates new points of failure in the complex system. However, it does create a significantly more balanced system than the standard Chord. On the other hand, a simple load balancing algorithm using a second hash function results in a notable reduction of imbalance in the system as well. This however introduces barely any overhead and does not require much programming work.

8 Conclusion

The development of internet devices towards smaller and more mobile units introduces challenges concerning capacity awareness in applications. Those applications that rely on the capacity of the machines like p2p systems are especially influenced by this development.

The overload reduction, that can be achieved using capacity aware systems, summarized in Table 1 shows that a peer-to-peer system operating in a heterogeneous environment gains significantly from a load balancing scheme.

Adding capacity awareness to an existing p2p system comes at a price. The table shows how the different approaches effect storage overhead, maintenance overhead for communication messages and lookup overhead. For each of these categories there is an approach that does not influence it at all.

When Storage capacity is the major concern, for instance, a system should consider to implement the Hashing Assisted Load Balancing (a), or Y_0. Both do not introduce any additional storage overhead.

Thus, even simple systems operating in such an environment should have a load balancing scheme implemented. The decision which one to choose depends greatly on the application, environment, and the level of node heterogeneity.

References

[1] J. W. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. pages 80–87. Springer, 2003.

[2] F. Dabek, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *18th ACM Symposium on Operating Systems Principles*, pages 202–215, 2001.

[3] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *INFOCOM*, 2004.

[4] B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *INFOCOM*, pages 596–606. IEEE, 2005.

[5] K. Graffi, A. Kovacevic, S. Xiao, and R. Steinmetz. Skyeye.kom: An information management over-overlay for getting the oracle view on structured p2p systems. In *ICPADS*, pages 279–286. IEEE, 2008.

[6] M. F. Kaashoek and I. Stoica, editors. *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, volume 2735 of *Lecture Notes in Computer Science*. Springer, 2003.

[7] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors, *IPTPS*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2002.

[8] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[9] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load balancing in structured p2p systems. In Kaashoek and Stoica [6], pages 68–79.

[10] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *Middleware*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[11] M. Srivatsa, B. Gedik, and L. Liu. Large scaling unstructured peer-to-peer networks with heterogeneity-aware topology and routing. *IEEE Trans. Parallel Distrib. Syst.*, 17(11):1277–1293, 2006.

[12] M. Steiner, T. En-Najjary, and E. W. Biersack. Long term study of peer behavior in the kad dht. *IEEE/ACM Trans. Netw.*, 17(5):1371–1384, 2009.

[13] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[14] unknown. Adding capacity awareness and load balancing to homogeneous distributed hash tables. 2012.

[15] Q. H. Vu, B. C. Ooi, M. Rinard, and K.-L. Tan. Histogram-based global load balancing in structured peer-to-peer systems. *IEEE Trans. Knowl. Data Eng.*, 21(4):595–608, 2009.