IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 2, Ver. IX (Mar-Apr. 2014), PP 104-114
www.iosrjournals.org

Peer to Peer (P2P) Social Networks: An Improvement on Availability in a Trusted Way

Muhammed Tawfiqul Islam[1], Sabbir Ahmed[2], Kazi Chandrima Rahman[3], Dr. Syed Faisal hasan[4]

[1](Department of Computer Science and Engineering, University of Dhaka, Bangladesh)
[2](Department of Computer Science and Engineering, University of Dhaka, Bangladesh
[3](Department of Computer Science and Engineering, University of Asia Pacific, Bangladesh)
[4] (Department of Computer Science and Engineering, University of Dhaka, Bangladesh)

Abstract: *Online Social Networks (OSNs) provide the easiest way to stay connected to the people dear and important to us. People are using social networking sites for sending messages, sharing files and expressing their thoughts to others. Popular social networking sites like Facebook, MySpace are web based. Users register with their services and agree to provide their private data to the OSN providers. OSN providers use centralized server. User data can be shared and updated easily in this fashion. But OSN providers give users a limited control over their data. Social networking sites actually contain huge amount of personal data which is a great source for learning the habits of consumers and can be used for commercial benefits. As the social networking sites are centralized, they also exhibit scalability and connectivity problems. So, the new generation social networks need to be distributed, free from third party and users will have full control over their data. Peer to Peer (P2P) infrastructure has become really important for designing distributed social networks. The features of a P2P based system are: P2P scales so well, locality of peers can be taken into account and social networking can be done without any internet connectivity in some mobile devices, people will have full control over their content. So, P2P based social networks can be our new generation of social networks with high degree of privacy and scalability. But the dimension shift from the centralized paradigm to a P2P paradigm makes the whole system really challenging. How the information will be stored, shared and cached in secured way are the major challenges of a P2P based social network. Existing P2P based social networks use friend peers as a trusted medium for caching other user data to improve the availability. In this research, we propose a new idea for caching data in non-friend trusted peers in a secured way. We will show system architecture for implementing this idea, protocols and algorithms for this technique which can be used by any existing P2P based social network for improving availability.*

I. Introduction

Social networking has become a part of our life that we live today. Online Social Networks (OSNs) like Facebook, Myspace, and Twitter are extremely popular among all regardless of age and gender. They enable people to stay in touch with their friends, find people they have lost contact, making new friends, forming social groups etc. People are using social networking sites for sending messages, sharing files and expressing their thoughts to the community. But most of the social networking sites are centralized and exhibit privacy, scalability and connectivity problems.

The information of a user is stored in a centralized server which is owned by the OSN provider. Users can adjust their privacy settings to protect their content and also limit access by other users. But there is no protection against the OSN provider itself. They can process the user data to gather valuable marketing information or proposing undesirable advertisements. Social networking sites actually contain huge amount of personal data which is a great source for learning the habits of consumers and can be used for commercial benefits. Facebook's 'Beacon' service is the example of such a tool that was designed to help people to point out interesting products. But it was actually an advertising tool which was targeting user's buying habits mapped to their social graphs.

Centralized social networks also have scalability and connectivity problems. Internet connectivity is a must to contact with a centralized server. So, social networks are not accessible from local devices. Social networking sites are growing day by day. It is really hard to handle tons of requests from a lot of users by some centralized servers. So, scalability is another major concern to the current social networking sites. Peer to peer (P2P) based infrastructure can be a solution to the existing problems.

Peer to peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. All content is transferred directly between ordinary peers without passing through third-party servers. Thus, P2P file sharing takes advantage of the resources (bandwidth, storage and CPU) in a large collection of peers, sometimes millions! In other words, P2P system is highly scalable.

Although no centralized, third-party server gets involved in the file transfer, it is important to keep in mind that P2P still relies on the client-server paradigm. Indeed, the requesting peer is the client and the chosen peer is the server. Since any peer can request or can be chosen, all peers must be capable of running both the client and server sides of the file transfer protocol.

An advantage of a P2P infrastructure is that it is not centralized, not owned by a single entity. The central storage of user information and ownership by a company which is a major privacy concern could be better addressed by a P2P approach, with encryption and appropriate key management. Centralized web-based social networks do not match the inherent P2P nature of both social networks themselves and of participatory media creation. By mapping P2P application to a P2P infrastructure, direct connections can be exploited such that locality can be taken into account. This enables peers to be mobile and independent from internet connectivity. User control of data, as provided by a P2P and secured social network has consequences beyond privacy and freedom from advertisement. One such consequence is that users can also have control over the content they create in terms of intellectual property. User control in this sense means control over who can access their content and what they are allowed to do with that content. Another huge advantage of P2P infrastructure is that P2P scales so well that it can support hundreds of millions users without any problem.

There are many challenges when we shift the total concept from client-server architecture to P2P architecture. Providing security and privacy is more difficult in a distributed system. As there is no centralized server, we need to ensure some distributed storage system. Of course we can use storage devices in the client machines. But we also need some caching or distributed storage when a client is offline. Caching and distributed storage will provide more availability but cached data needs to be validated, it means we have to ensure cache coherency.

Availability is a major challenge in P2P based social networks. In this research we will propose an improved technique which can be used to increase the availability. We will also propose a distributed trust model that will work with this technique to provide privacy in caching. At last we will present our implemented prototype for testing our idea. We will simulate our prototype and show the performance improvements.

II. Background Study

2.1 Centralized System

In telecommunication, a centralized system is one in which most communications are routed through a central hub, peer or a server. A centralized server is a mainframe computer with a lot of resources. It has huge memory and great processing power. It accepts requests from various clients and processes their request. Multiple threads are used to provide parallel service. A great advantage of a centralized system is that the ease of updating and maintaining shared data. But the biggest flaw of a centralized system is that it has a single point of failure. If the central server goes down, all system will go down. There are also some privacy concerns for the clients as they trust a centralized server and share their information with it. Centralized web servers are the example of centralized systems.

2.2 Distributed System

Tanenbaum [1] describes that distributed system is a collection of independent computers that appears to its users as a single coherent system. One important characteristic of a distributed system is that the differences between the various computers and the ways in which they communicate are hidden from users. The same holds for the internal organization of the distributed system. Another important characteristic is that users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.

Distributed systems should also be relatively easy to expand or scale. This characteristic is a direct consequence of having independent computers, but at the same time, hiding how these computers actually take part in the system as a whole. A distributed system will normally be continuously available, although perhaps certain parts may be temporarily out of order. Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.

To support heterogeneous computers and networks while offering a single system view, distributed systems are often organized by means of a layer of software that is logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems. Such a distributed system is sometimes called middleware.

2.3 Client Server Model

It is the most popular distributed architecture which works by distributing the workloads among service providers, called servers and service requesters called clients. A server is a high speed, high capacity computer for servicing requests. Clients are ordinary user machines. In the Internet system, information is stored in servers. Servers can also perform some processing task like supporting queries, other mathematical operations

etc. A client finds an appropriate server for its desired information or task. Then it requests the service to the server. If the service can be provided by a server it accepts the request. A server can have multiple incoming connections to support multiple queries. They actually use request queues for multiple requests. Multiple threads of a server are actually used to provide faster service time.

Communication between a client and a server can be both connection-oriented and connectionless. Communication can be implemented by simple connectionless protocol if the underlying network is reliable. Connectionless protocol like UDP (User Datagram Protocol) has a biggest advantage that it is really efficient and is suited for real time applications like online multiplayer gaming, video streaming, voice/video chatting etc. But if reliability has to be ensured, reliable connection-oriented protocols like TCP (Transmission Control Protocol) is used. They are suited for file transfer, mail transfer etc. where packet loss can't be tolerated. Reliable protocols also provide some other features like flow control, congestion control etc.

2.4 Peer to Peer (P2P) Architecture

P2P architecture is distributed application architecture. It works by dividing the workloads among the peers. A peer is a node which can be a simple user machine that participates in the application. All peers have same privilege. They share storage, processing power and network bandwidth which are available to others. In centralized systems, a node can be client or a server. But in P2P based architecture, peers are both clients and servers. When a peer shares a resource it can be thought as a server. When it consumes a resource from other peers, it is a client. This architecture can run without any central authority.

A P2P network can be of two types: structured or unstructured. In a structured P2P network, a policy is maintained to organize the peers. There are algorithms about how a peer look-up can be done and how information will route through peers. Structured P2P networks actually use Distributed Hash Table (DHT) based indexing. But in unstructured P2P networks no algorithms are used to manage and optimize network connections. In pure P2P systems no infrastructure nodes are used and the total system is decentralized. But hybrid systems also exist where a 'Super Node' is responsible for maintaining the P2P infrastructure. There are centralized systems where a server is responsible for indexing and bootstrapping function of the whole system.

2.5 Distributed Hash Table (DHT)

A distributed Hash Table (DHT) is a class of decentralized distributed system that provides a lookup service similar to a hash table; (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the values associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. DHT can form an infrastructure that can be used in Web caching, distributed file systems, domain name services, instant messaging, P2P file sharing and content distributions. [2]

2.6 P2P Storage Systems

In P2P based systems, a policy must be maintained on how to store data on peers. There are many issues like privacy, accessibility, availability and controllability. Various existing P2P storage solutions are available like:

Oceanstore [3] is a P2P infrastructure where file system operations are supported. Data is distributed and can be cached among other peers. So availability and fast accessibility is provided here. Data is encrypted and stored in peers so security is ensured. But a third party access provider is used to ensure the security.

Ivy [4] is a multiuser read/write P2P file system that used DHash [5] to store data. Each user has a private log which can be modified to perform file updates. So it has the advantage that it is free from locking. But the biggest disadvantage is that it doesn't encrypt the files. The files are distributed in clear text to other peers.

Pastis [6] uses DHT to provide routing and storage. It is similar to Ivy as it also supports multiuser read/write like Ivy [4]. But the use of DHT improves the performance of Pastis over Ivy and Oceanstore. But Pastis is only able to maintain write access on files; no read access can be prevented. So ensuring privacy on stored files is impossible if encryption is not done in the client side.

2.7 Trust Models

The Oxford Dictionary of English defines trust as the "firm belief in the reliability, truth, or ability of someone or something" [7]. In P2P based systems, trust between peers needs to be established in order to provide a secured communication. No centralized server is used in a P2P based system so trust managing in these systems is a major challenge. There are many existing trust models that can be used in distributed systems to manage trust.

Global Trust Model [8] is a binary trust model. Here an agent can be trustworthy or not. Trustworthiness is measured by the transactions among the agents. An agent checks whether a transaction is performed correctly or not with another agent. If it happens that another agent is a cheater, transaction can't be performed correctly and complain about that agent is filed. To make this information (complain) global, this information is forwarded to other agents in the systems. All the agents save **this** information and do not share any data with untrustworthy agents. If a cheater agent files complain about many agents, all the agents can detect this cheating by receiving complain about that particular cheater agent from other agents.

NICE Model [9] is used in the background to determine the good peers in P2P systems. This model can be used to prevent the malicious peers which can cause privacy and security threats. Each peer creates a cookie after each transaction where it saves the feedback about the other peer. These cookies are exchanged among peers. If a transaction was successful, then the value of the cookie will be positive. Otherwise for unsuccessful transactions, negative value of cookie is stored. In NICE model if a peer wants to request for data or other service, that peer can show a cookie signed by the provider itself. The validity of a cookie will be justified by the provider. If a cookie is right, then it is proof of trustworthiness of a peer.

Eigentrust [10] is designed for the reputation management in the P2P systems. The Eigentrust algorithm is based on the notion of transitive trust: If a peer *i* trusts any peer *j*, it would also trust the peers trusted by *j*. Each peer *i* calculates the local trust value s_{ij} for all peers that have provided it with authentic or fake downloads based on the satisfactory or unsatisfactory transactions that it has had:

$$s_{ij} = \text{sat(i, j)} - \text{unsat(i, j)}.$$

where sat (i, j) refers to the number of satisfactory responses that peer I has received from peer j, and unsat (i, j) refers to the number of unsatisfactory responses that peer I has received from peer j. the local value is normalized, to prevent malicious peers from assigning arbitrarily high local trust values to colluding malicious peers and arbitrarily low local trust values to good peers. The normalized local trust value c_{ij} is then

$$c_{ij} = \max(s_{ij}, 0) / \sum \max (s_{ij}, 0).$$

The local trust values are aggregated at a central location or in a distributed manner to create a trust vector for the whole network. Based on the idea of transitive trust, a peer *i* would ask other peers it knows to report the trust value of a peer *k* and weigh responses of these peers by the trust peer *i* places in them.

Trust for Large Scale P2P Systems [11] is a challenging issue. P2P networks can be decentralized so no centralized authority can be found to manage trust in these systems. This model tries to provide a distributed reputation based mechanism for trust. In P2P system, peers establish trust relationship among themselves by forming ratings. This rating can be binary rating, where a ZERO indicates that a peer is untrustworthy and a ONE is assigned to set a peer as trustworthy if previous transactions were successful. Each peer in these systems is connected with several other peers called neighbors. Rating is distributed by flooding the rating information to the neighbors. Each peer collects rating information from its neighbors and aggregate to calculate trusts of other unknown peers.

2.8 Replication

Replication means maintaining copies of data in multiple machines. A replica is an exact copy of a data item. To improve availability, fault-tolerance, response time and reliability, replication is used. In P2P systems, data is shared between the clients. If P2P file system is considered, every peer has the whole file or some portion of the file. The peers which have full file and uploading only are known as seeders. Other peers have some portions of the file and they are both downloading remaining portion and uploading existing portion. These peers are known as leechers. The structure of this system is replication based. That means after some time, every peer will have the same data. The data is replicated over all the peers. But if a distributed application is considered where a data or file can be modified over time, replication can create more overhead due to high update cost. For example, a P2P based social network can be considered. We have replicated a post of a user across many peers. If the post is about some current issue and is modified frequently, we have to propagate updates frequently. So many replicated copies mean many update operations. But the bigger the number of replica, more availability can be ensured. So there is a tradeoff between maintaining consistency by update and replication.

Replication is really essential in P2P based social networks. We don't have any centralized server to hold user data. Even though a user is offline, the profile information or contents of that user have to be provided to other friends to the user. So replication is used to provide availability. Faster response time also can be acquired if parallel connections can be used with many peers to get the same content. Another advantage is that locality of peers can be considered. Data can be taken from a local peer really fast. Recently, mobile devices with Wi-Fi networks are becoming really popular. If data is replicated to a local machine which is also

connected to other devices with Wi-Fi like media, mobile devices can be used for social networking without any internet connections.

Update operations on a distributed and replicated data store generally initiated at a client and subsequently forwarded to one of the copies. From there, the update should be propagated to the other copies, while at the same time ensuring consistency. There are different designed issues to consider with respect to propagating updates.

- State versus Operations [1]: An important design issue concerns what is actually to be propagated. There are three possibilities:
  1. Propagate only a notification of an update: It is actually the invalidation protocol. In this protocol, other copies are informed that an update has taken place and that data is no longer a valid one. The advantage of this approach is that it uses little network bandwidth. If the data is not frequently accessed or the date is large in size, this technique is suitable. Updates can be propagated on a later time when the network has little congestion or the updated data is needed.
  2. Transfer data to one copy to another: It is a straight-forward approach for update propagation. Updates will be propagated whenever it is modified. If the data is a frequently accessed one, it should be kept updated all the time by using this approach. In P2P social networks, frequently accessed data like status messages, comments on posts needs to be propagated whenever possible.
  3. Propagate the update operation to other copies: This technique also uses little network bandwidth and distributes the workload among peers. If the update is like an operation e.g. databases update operations; the updated data is not propagated if the other replica holders have processing abilities to perform the operation. So, the operation is propagated among the peers and they update the data by themselves. In our P2P based network we don't have this kind of update operations. We will just consider the first two cases of update propagations.
- Pull versus Push protocols [1]: It is a design issue that determines whether updates are pulled or pushed. In a push-based approach, also referred to as server-based protocols, updates are propagated to other replicas without those replicas even asking for the updates. Push-based approaches are often used between permanent and server-initiated replicas, but can also be used to push updates to client caches. Server-based protocols are applied when replicas generally need to maintain a relatively high degree of consistency. So, social networking sites actually follow this technique. In contrast, in a pull-based approach, a server or client request another server to send it any updates it has at that moment. Pull-based protocols, also called client-based protocols, are often used by client caches. In P2P based social networking, pull-based protocols are suitable as there is no central authority. The clients are responsible for updating their cache to maintain consistency.

III. Improving the Availability in a Trusted Way

P2P social networks use distributed storage systems to store user data. Some information of the user are also cached in some peer that are known (friend) to a user. Information can be cached in a friend peer only if it was requested by that peer. So even if a user is offline, his/her profile can be accessed from these distributed storage systems or friend peers. Data is cached only if a peer can be trusted. So a trust manager is maintained to determine the trust of peer when data is shared or replicated. In the following sections, a new idea to improve the availability will be discussed. Then prototype architecture for supporting the idea will be described.

3.1 Idea to Improve the Availability

Availability can be increased if more information can be stored in a friend peer. Current solutions only store information that was requested by a peer. Other frequently accessed information which may be needed in future by other peers can also be cached in a friend peer. To provide more degree of availability, trustworthy peers that are not friends to a user can also be used. Suppose, some information is replicated/cached in a friend peer. That friend peer can further replicate the information to other peers that are trustworthy and friend to it. The new peers may not be friend to the original information provider. This technique will increase availability and high speed parallel access to an information. Actually, modern day computers have huge storage and a bigger portion of the storage may remain unused. So, sacrificing some storage to improve the social networking performance can't be a bigger problem. But some policies need to be established on when to use the bandwidth, what amount of storage can be used in a specific peer etc.

3.2 Architecture

To implement the idea, we have designed a system that will use this technique to improve availability. It also has a trust model to determine trustworthy peers before replicating data. The architecture can be divided into two parts; the components, and the protocols.

3.2.3 Components

There are two components in our system. A Superpeer and many clients connected to the Superpeer. A client can also connect to other clients for exchanging data and sharing o information. The whole structure of the system can be easily understood by looking at the figure below:

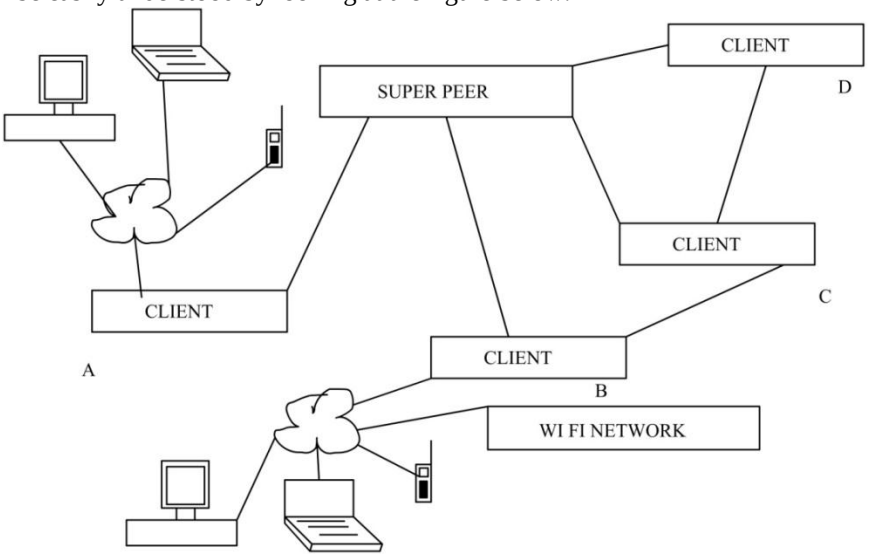

Figure: A P2P based Social Network Architecture.

As we can see, super peer is connected to all the clients. Some client machine is also the member of other Wi-Fi networks. If these internal networks have some mobile devices, they can be used for social networking too.

- **SuperPeer** is special peer which maintains the infrastructure of the social network. It is used to store the client online/offline information, bootstrapping, other client locating, finding cache locations, gathering time for validating or updating cache etc. This peer is similar to the Distributed Hash Table (DHT). The whole system may look centralized now when we have introduced a peer with more power and responsible for maintaining the structure. But it is completely different from a centralized social network because it doesn't store information about any peer. The information of a user is shared directly between with another user who requests it. SuperPeer is the medium for finding a user. When multiple replicas of content/user information are present, the SuperPeer can provide all the locations where the replicas can be found. It is also responsible for providing the timestamp of a data so that client caches can validate themselves by invoking the SuperPeer. If they have backdated data, they collect updated cache location and update themselves.

The SuperPeer maintains two tables. A **Client Table** which stores client id, availability and the locations where the information of that particular client are cached. The Replication Table is used to store the caching information for every available/online client. A timestamp is also assigned with contents. This timestamp is updated whenever the original information provider modifies the file. For maintaining the consistency, all the client programs consult the SuperPeer Replication Table for last updated time of their cached information. If any client finds that it has an invalid data (it finds it has an older timestamp for information), it requests the SuperPeer for updated cache locations. Then it can establish direct connections with one or more updated cache to update its own cache. The following figure shows the basic architecture of a Super Peer.

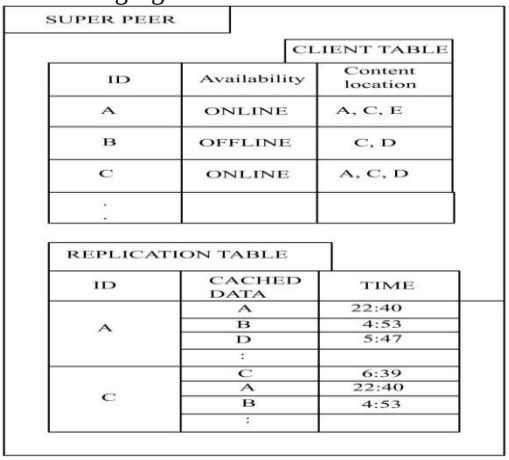

Figure: SuperPeer

- **Client** is the user machine that will run the social networking application. Clients will communicate with the SuperPeer to register their availability. When a client wants to join the network, it sends a message to the SuperPeer. The SuperPeer records its availability as ONLINE. The client will send the information about the replicated contents it has. SuperPeer will maintain the replication table where it will record the list of cached contents of that client. When another client will try to access any data that is cached by that client, the SuperPeer can easily send the location.

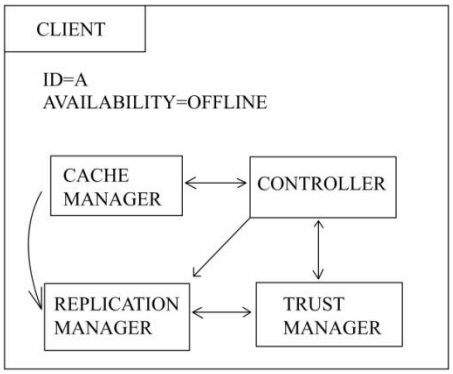

Figure: Client

The client has four important components that will control the whole social networking operation. Each client application has a main Controller, Cache Manager, Replication Manager and Trust Manager.

Controller:

It is the coordinator of the whole application. It has a central algorithm running for controlling the application. Interrupt handler is used to notify the controller about new post from the user, caching request from friend peers and data access request from other peers. It has a timer which is used to synchronize the Client with the SuperPeer for cache validation. When the timer is UP, Controller generates a message including the Client's cached contents id and caching time to the SuperPeer. SuperPeer checks the timestamps to decide whether that client has an updated cache or not. If SuperPeer sends new update information to the Client Controller, it contacts directly to peers those have updated data. Controller maintains other important components like Trust Manager, Cache Manger, Replication Manager for performing these operations.

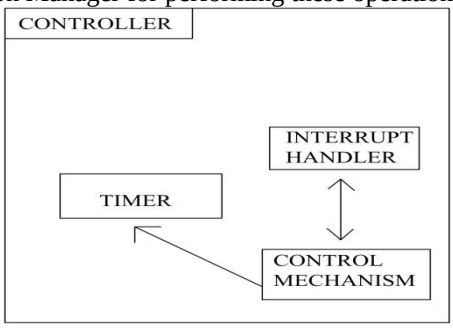

Figure: Controller

Cache Manager:

Cache Manager is responsible for storing and maintaining cached data in a client machine. The main Controller program requests the Cache Manager to receive new data for caching through the I/O interface. Controller can also request the Cache Manager to retrieve any cached data through the I/O interface. The Dispatcher is used to accept requests from the Controller. It has a queue for accepting multiple incoming or outgoing requests.

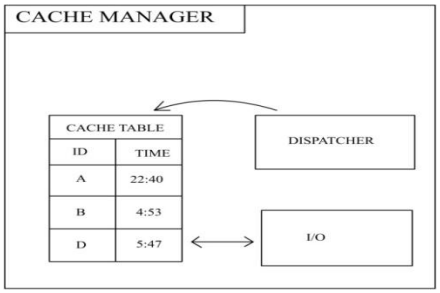

Figure: Cache Manager

Replication Manager:

Replication manager is used to send data for replication to other peers. The system can send any of the cached data for replication. That means we are not replicating the client's information only, we can further replicate other user data that are cached in that machine. This technique will achieve more availability in the P2P social network. Controller program works with the Trust Manager to determine the trusted peers where data can be replicated. Replication manager only gets request for sending a data packet to a given address. It generates the message, establishes connection and sends the message to that peer.

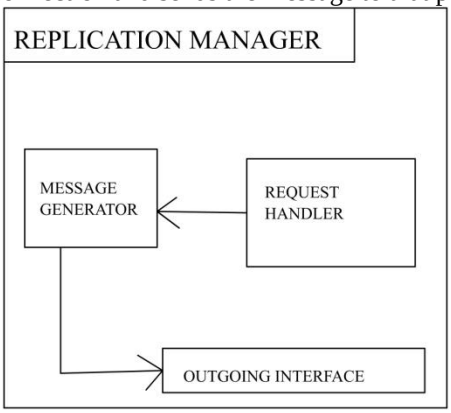

Figure: Replication Manager

Trust Manager:

Trust Manager is an important component in the client application. In our idea, we are now caching data in a peer where it may not have the access right for that data. So, maintaining trust relationship is a must here. The data will be encrypted and if a peer is not a friend to the data provider it can't have the key to decrypt the content. But for providing higher degree of privacy, we can't replicate data to any peer. A malicious peer can cause an attack to decrypt any cached information. A trust model can be implemented which will be best suited for the environment of a P2P Social Network. In our simple implementation for improving availability, we assume that the system is using a binary trust model like the Global Trust Model [8].

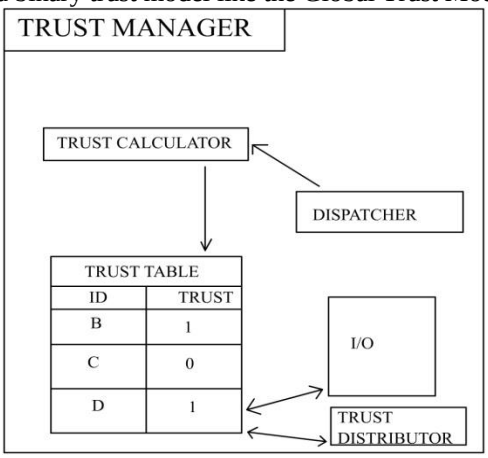

Figure: Trust Manager

IV. Implementation

We have implemented a prototype of a simple P2P based social network that only supports simple contents like messages. We are actually interested on improving the performance by increasing the availability. So, only the parts necessary for testing our idea is implemented here.

The whole code is done using Socket Programming and UDP as the transport layer protocol. The pseudo codes for all the components of the system are given below:

Client:

1 Controller;
2 Cache Manager;
3 Replication Manager;
4 Trust Manager;
5 Timer;
6 Interrupt Handler;

Controller:

```
1       send_to_super_peer_i_am_online(my_id);
2       wait() until any interupt or massage available from user
3       if interrupt() from timer
4               send_to_super_peer(frndlst);
5               time=get_data_times_from_super_peer();
6               for each frnd i
7                       if time[i]>cached_time[i]
8                               update_time(i);
9                               list=get_users_who_have_data_from_super_peer(i);
10                              sl=select_random_from(list);
11                              info=get_addrss_of_a_peer_from_server(sl);
12                              data=get_data_of_frnd(sl,i);
13                              send_data_to_Cache_Manager(i,data);
14                              send_data_to_Replication_Manager(data);
15
16      if interrupt() from Interrupt Handler
17              if another_peer_ask_for_data
18                      (id,peer)=id_and_addr()
19                      data=ask_data_from_cache_Manager(id);
20                      send_data_to_peer(peer,data);
21              if another_peer_ask_to_cache_data()
22                      (id,data)=to_cache_data();
23                      send_data_to_Cache_Manager(id,data);
24              send_to_super_peer_that_i_have_a_new_data_of_another_one(id);
25      if massage available from user
26              msg=msg_from_user();
27              send_data_to_Cache_manager(my_id,msg);
28              send_to_super_peer_that_i_have_a_new_data(my_id);
```

Cache Manager:

```
1       if ask_to_save_data()
2               (id,data)=data_to_save();
3               write_to_file(id,data);
4               send_to_super_peer_that_i_have_ones_data(id);
5       if ask_to_send_data();
6               id=which_data();
7               data=read_from_file(id);
8               send_to_controller(data);
```

Replicatin Manager:

```
1       data=asked_for_replication();
2       trusted_friend_list=trust_manager();
3       for each trusted_friend_list i
4               send_to_peer(i,data);
```

Trust Manager:

```
1       list=Manage_Trusted_list();
2       send_trusted_list(list);
```

Timer:

```
1       wait(120 sec);
2       send_interrupt_to_controller();
```

Interrupt Handler:

```
1       wait for any data available from any peer;
2       if(data_available)
3               send_interrupt_to_controller();
```

SuperPeer:

```
1       wait() for a request from a client;
2       msg=recv_msg();
3       (id,query)=parse(msg);
4       if query=online
5               update_to_CLIENT_TABLE(id,online,id.addr,id.port);
6       if query=offline
7               update_to_CLIENT_TABLE(id,offline,id.addr,id.port);
8       if query=is_updated
9               who=parse(msg);
10              list=all_peer_who_has_data_from_replication_table(who);
11              send_to_client(id,max_of(list.times));
12      if query=who_have_data
13              who=parse(msg);
14              list=all_peer_who_has_data_from_replication_table(who);
15              for each peer x in list
16                      if x is online from CLIENT TABLE
17                              send_to_client(id,x);
18      if query=get_addrs
19              who=parse(msg)
20              addrs=get_addrs_from_Client_Table(who);
21              send_to_client(id,addrs);
22      if query=have_new_data
23              time=system_time();
24              update_Replication_table(id,time);
25      if query=have_another_ones_data
26              who=parse(msg);
27              time=system_time();
28      update_Replicatoin_table(id,who,time);
```

V. Conclusion and Future Works

In this research, we have shown the problems of the existing centralized social networks. P2P based social networks are the solution and we have discussed the challenges we have to face when we create this system. We have discussed on existing P2P based Social Networks and we also mentioned how they faced the challenges. We have concentrated on the availability of a P2P Social Network. Existing P2P networks use caching and distributed storage systems to improve availability. We have implemented a better technique to achieve more availability.

5.1 Research Summary

Availability can be increased if more information can be stored in a friend peer. Current solutions only store information that was requested by a peer. Other frequently accessed information which may be needed in future by other peers can also be cached in a friend peer. To provide more degree of availability, we have shown that trustworthy peers that are not friends to a user can also be used. Suppose, some information is replicated/cached in a friend peer. That friend peer can further replicate the information to other peers that are trustworthy and friend to it. The new peers may not be friend to the original information provider. This technique will increase availability and high speed parallel access to an information. We have provided the architecture of our implemented prototype which supports this technique. We have included the pseudo codes which can be used by any existing system for better performance.

References

[1] Tanenbaum, Andrew S.; Steen, Maarten V.: *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2002. – ISBN 0130888931.

[2] http://end.wikipedia.org/wiki/Distributed_hash_table.

[3] Kubiatowicz, John ; Bindel, David ; Chen, Yan ; Czerwinski, Steven ; Eaton, Patrick ; Geels, Dennis ; Gummadi, Ramakrishna ; Rhea, Sean ; Weatherspoon, Hakim ; Wells, Chris ; Zhao, Ben: OceanStore: An architecture for global-scale persistent storage. In: *SIGARCH Comput. Archit. News* 28 (2000), Nr. 5, S. 190–201. http://dx.doi.org/http://doi.acm.org/10.1145/378995.379239.

[4] Muthitacharoen, Athicha ; Morris, Robert ; Gil, Thomer M. ; Chen, Benjie: Ivy: a read/write peer-to-peer file system. In: *SIGOPS Oper. Syst. Rev.* 36 (2002), Nr. SI, 31-44. http://dx.doi.org/http://dx.doi.org/10.1145/844128.844132.

[5] Dabek, Frank; Kaashoek, M. F.; Karger, David; Morris, Robert; Stoica, Ion: Wide-area cooperative storage with CFS. In: *SOSP '01: Proceedings of the eighteenth* ACM, 2001. – ISBN 1–58113–389–8, S. 202–215.

[6] Busca, Jean michel ; Picconi, Fabio ; Sens, Pierre: Pastis: A highly-scalable multi-user peer-to-peer file system. In: *in Euro-Par 2005*, 2005.
[7] Soanes, Catherine (Hrsg.) ; Stevenson, Angus (Hrsg.): *"trust noun" The Oxford Dictionary of English (revised edition)*. Oxford University Press http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t140.e82219
[8] Karl Aberer and Zoran Despotovic Managing Trust in a Peer-2-Peer Information System. In *CIKM'01*, November S-10,2001, Atlanta, Georgia, USA.
[9] Hai Ren. Comparison of Trust Model in Peer to Peer System.
[10] Sepandar D. Kamvar, Mario T. Schlosser and Hector GarciaMolina. The EigenTrust Algorithm for Reputation Management in P2P Networks.
[11] Bin Yu, Munindar P. Singh and Katia Sycara. Developing Trust in Large-Scale Peer-to-Peer Systems. IEEE 2004.