International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 553-560 Jan 2019-June 2019 www.csjournals.com

Design and Implementation of file sharing with the support of suitableTopology

Ruchi Sodhi[a], Ghanendra Kumar[b] and Chakresh Kumar*[a]
[a]University School of Information, Communication & Technology, Guru Gobind Singh IP University, New Delhi
[b]Department of Electronics and Communication Engineering, National Institute of Technology, Delhi-110040, India
*mail-id: chakreshk@gmail.com

Abstract: In the world of today the number of devices connected to the Internet or any other type of network are ever increasing, and the demand to share resources, whether they be files, disk space, computer cycles, connected peripherals etc., between them is also rising exponentially. To cater this need Peer-to-peer networks were created. It was initially popularized by the file sharing system Napster. Through the years that have passed, since they were first used, they have involved significantly. Due to this evolution each organisation has come to adopt different network overlays and algorithms for network maintenance and resource sharing in order to provide fast and easy access to their customers for the sharing of their concerned resources. The major concern till date remains about the structure used to create and maintain the network overlay, in addition to the algorithms deployed to handle the real world conditions to share resources and provide better fault resilience and scalability. The objective of this project is to implement a network based on B – Tree which aims at providing a new structure for network creation and maintenance along with better fault tolerance. On top of this overlay a simple text based file sharing system is to be built. This system utilizes Internet as its backbone.
Keywords: Topology, binary, data rate, point to point.

I. INTRODUCTION

In the most basic form, a Peer-to-Peer (P2P) network is established when two or more computers are connected with each other, and the resources are shared between the users without any need of having a server computer. P2P network can also be a permanent arrangement that can link half-dozen computers in a small workplace over copper wires, or a Peer to Peer network can be a network on a much bigger scale in which dedicated protocols and applications are used to set up an uninterrupted relationships between users over the Internet[1-3].

A. Distributed file sharing system

A typical implementation of a video streaming system would include a single watching node taking the entire video data from a single streaming node. Though this architecture is simple, it comes with its own drawbacks [4-5].

1) The streaming speed will always be limited by the upload speed of the streaming node.
2) If the streaming node goes down while streaming, the streaming process will fail.

Single Streaming Peer

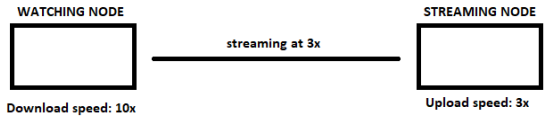


Fig. 1 single Streaming peer

Multiple Streaming Peer

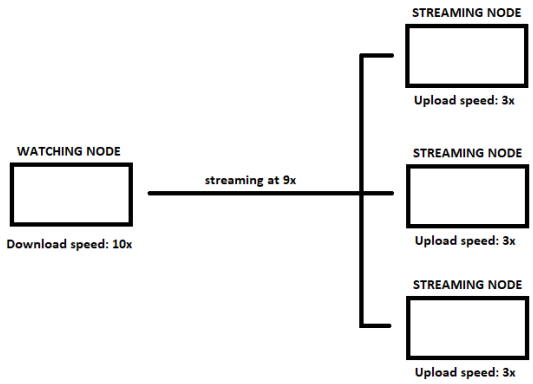


Fig.2 Multiple Streaming Peer

A distributed video streaming system helps us in overcoming these 2 disadvantages. In a distributed environment, a given video is streamed by taking segments/frames of the file from multiple streaming nodes simultaneously. This will help us is attaining the maximum streaming speed and will also remove the dependency from a single streaming node.

B. Network Topology

Network Topology

The logical and hierarchical manner in which computers are computers are connected is termed as Network topology. It defines how the computers are connected with each other in the network. Network Topology is also called as “network architecture.”

Devices on the network are known as 'nodes.' Network Topologies are classified on the basis of manners in which the nodes are connected with each other. Network topologies

International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 553-560 Jan 2019-June 2019 www.csjournals.com

determine the way nodes communicate with each over the network. There are few types of network topologies such as bus, Point-to-point, star, mesh and ring.

1) Bus: One main cable is used in the Bus Topology. Nodes are directly connected to the cable. The main cable is the backbone of the network

2) Point-to-Point: Point-to-point topology is the most basic simplest network topology. The nodes have a direct link between them.

3) Star: Star Network Topology consists of a central hub to which each node is connected using point-to-point connection.

4) Mesh: Every node is connected to the every other node in the Mesh Network topology. This topology is not efficient in handling high volume of traffic

5) Ring: In ring topology, each node is connected to the other two nodes forming a ring like structure. Data travels node to node in one direction.

II. ARCHITECTURE

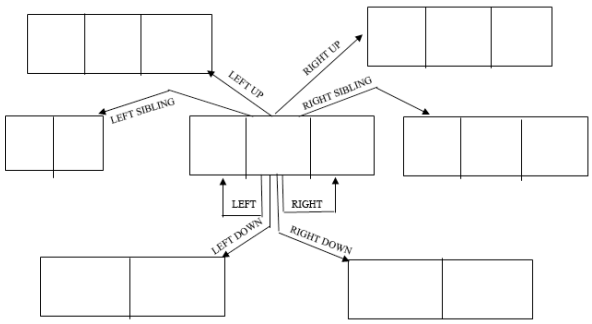
The structure for establishing the network overlay is based completely on the backbone of internet i.e. if a node is connected to internet then only can it be connected to the nodes in the B Tree network. architecture. It uses a modified B-tree for the arrangement of nodes in node packs based on the joining time (epoch time), as and when they join a network. The B-tree has been modified in terms of methods by which searching, insertion and deletion takes place in the overlay, the basic structure remains the same.

A particular node in the B Tree network architecture is assigned a unique nodeId which is based on the epoch time at which the node is inserted in the network, meaning that a node is assigned an Id after it has been added to the overlay network. In addition to this each node is responsible for maintaining eight different IP address pointers for storing the addresses of its corresponding neighbours in the structure. The different neighbours possible in the network are left node, right node, left parent, right parent, left sibling, right sibling, left child and right child (as indicated in the figure 2.1). The value for these pointers can either contain the IP addresses of the nodes that have been added to those particular locations or the value will be NULL.

Moreover each node also maintains the number of nodes on its right and left in a particular node pack and increments/decrements it according to the addition/deletion of node in its own pack. This count is used to keep track of the total number of nodes in a pack and when this count exceeds 5 (taken in the architecture, it can be assumed to k where k is any odd number value such that k is sufficiently large in order to provide proper scalability of the network) a decision for splitting of the node pack is made and suitable algorithm, explained in later section, is deployed in order to achieve that. Also a mechanism is provided in the network overlay such that when a particular node decides to go offline or gets disconnected from the network, all of its neighbours come to know about this event and hence are able to update their count and suitable pointers. In accordance to this, the structure is evaluated and suitable changes are made in the levels of the remaining nodes of the node pack if and only when required.

The network also takes into consideration the minimum number of nodes that are required in the node pack i.e. 2, if the node count falls below this suitable restructuring takes place in the overlay. Moreover the process of restructuring is completed in such a way that it ensures the proper working of the remaining nodes and it accomplishes this task in minimum delay. Thus the system developed is fault-resilient and scalable up to a large extent.

The node pack defined in the network overlay is acts an independent entity which is capable of making decisions for structuring as well as message forwarding in the network. A maximum size of node pack is also defined in order to provide efficiency in routing and location of nodes. On exceeding the size of a particular pack it splits up and correspondingly a decision is made for promotion/demotion of levels of the nodes in the pack



2.1 Different pointers maintained by a node in a nodepack

III. FUNTIONING

INSERTION

- In the Balancing Tree architecture a node can either decide to create its own network or join a pre-existing network. In the latter case the requesting node is required to know the IP address of any node that has already joined the network which it wishes to be a part of. After obtaining the IP address the requesting node sends an 'Insert Request' which is a JSON packet containing the following information:
- IP address : It has the IP address of the requesting node
- Key : Information about the key of requesting node
- Port Number : Indicated the port number on which the application is running on the requesting node.

- Direction : This indicates the position from which the Insertion request is coming, initially it is set to "Equal_Level"
- Now after this JSON packet has been received, it becomes the responsibility of the receiving node to find suitable position for the requesting node to be inserted in the network by following the various algorithms. The various cases that can occur are as follows:
- Receiving node is a leaf node : The receiving in the following case has two primary functions to perform depending on the value of 'Direction' field specified in the received JSON packet, they are –
- Direction = 'From_Above' : On encountering a packet with such a field it would insert the pack in the same node pack.
- Direction = 'Equal_Level' : When this packet is seen by the receiving node it changes the 'Direction' field of the packet to 'From_Below' and passes this packet to its above level i.e. its parent.
- Receiving node is not a leaf node : In such a case the receiving node again has to make a split decision based on the value of 'Direction' field of the received JSON packet, they are –
- Direction = 'From_Split' : When the JSON packet contains the following information it inserts the requesting node in the same node pack as its own.
- Any other value specified in 'Direction' field : On encountering this type of packet the receiving node checks the value of 'Key' field in the JSON packet and against its own, different cases arise here –
- If the value of JSON packet is smaller than its own and greater than the key of the left node of the receiving node then it changes the 'Direction' of the packet to 'From_Above' and transfer it to the 'Left_Down' node.
- If the value of the packet is smaller than its own and even from the its left node the receiving node transfers the packet to its left node for further handling.
- When no left node exists transfer the packet to 'Left_Up' node and set the packets 'Direction' field to 'From_Below'
- If the value of packet is greater than the receiving node but smaller from its right node then it sets the'Direction' field to 'From Above' and transfers it to 'Right_Down'
- If the value of packet is greater than its own and from its right node then transfer the packet to its right node for further handling.
- When no right node exists transfer the packet to 'Right_Up' node and set the packets 'Driection' field to 'From_Below'
- This process continues till the request reaches a leaf node, as the insertion can take place only on the leaf nodes. On reaching a leaf node two different cases can be encountered, they are :
- Inserting the requesting node in the same pack : When the packet reaches the leaf node it check the value of 'Key' in the packet against its own and if the key is greater/smaller than its own and no right/left node exists in the node pack the requesting node is inserted on the right/left side of the receiving node and its suitable pointers are updated.
- Inserting the requesting node between two nodes : The receiving node performs the comparison of the value of 'Key' field in the packet against its own , if the key is smaller/greater than its own it checks the key with the key of its left/right node and the value of 'Key' of packet falls between its own and its left/right node the requesting node is inserted in between these two nodes and the suitable pointers are update for all the concerned nodes.
- After the requesting node has been inserted in a node pack updation of pointers and counters takes place within the pack. And the total number of nodes in the pack are checked against the maximum and minimum nodes permissible in a node pack and if any violations.

Example of inserting nodes to network

1) NODE_151 is the first node to start the network. Initially NODE_51 exists as a stand-alone system.

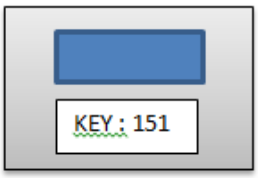


Figure 3.1.1

2) NODE_156 wishes to join the network created by NODE_151. Therefore, it requests NODE_151 which adds NODE_156 to the right of NODE_151(based on the key).

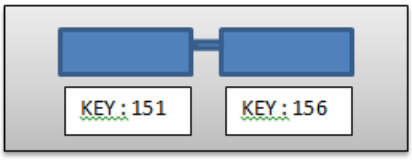


Figure 3.1.2

3) NODE_159 wishes to join the network created by NODE_151. Therefore, it requests NODE_151 or NODE_156 which adds NODE_159 to the right of NODE_156(based on the key).

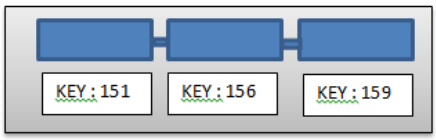


Figure 3.1.3

4) NODE_161 wishes to join the network created by NODE_151. Therefore, it requests NODE_151, NODE_156 or NODE_159 which adds NODE_161 to the right of NODE_159(based on the key).

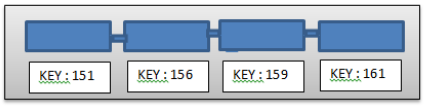


Figure 3.1.4

5) NODE_162 wishes to join the network created by NODE_151. Therefore, it requests NODE_151, NODE_156, NODE_159 or NODE_161 which adds NODE_162 to the right of NODE_161(based on the key).

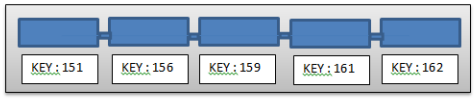


Figure 3.1.5

6 (A) NODE_164 wishes to join the network created by NODE_151. Therefore, it requests any node in the existing network which adds NODE_164 to the right of NODE_162(based on the key).

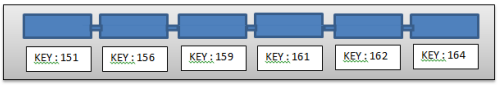


Figure 3.1.6a

6(B) The pack now has more than 5 nodes. So the pack splits and sends the central node to the parent pack(root in this case).

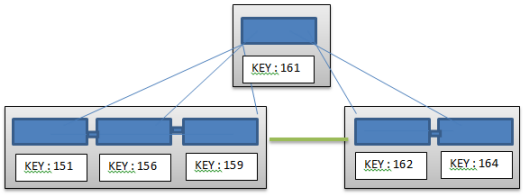


Figure 3.1.6b

7) NODE_144 wishes to join the network. Therefore, it requests any node in the existing network which adds NODE_144 to the left of NODE_151(based on the key).

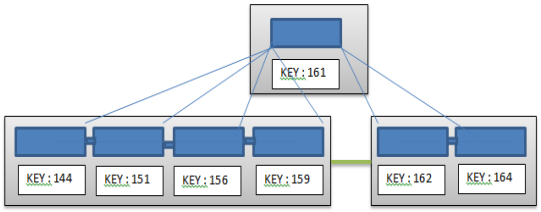


Figure 3.1.7

8) NODE_131 wishes to join the network. Therefore, it requests any node in the existing network which adds NODE_131 to the left of NODE_144(based on the key).

International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 553-560 Jan 2019-June 2019 www.csjournals.com

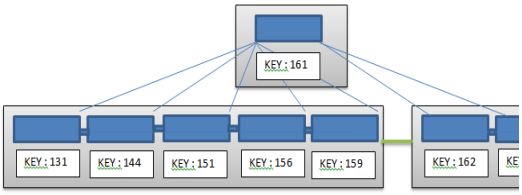


Figure 3.1.8

Figure 3.1.8

9 (A) NODE_120 wishes to join the network. Therefore, it requests any node in the existing network which adds NODE_120 to the left of NODE_131(based on the key).

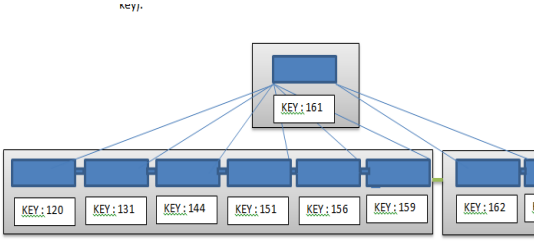


Figure 3.1.9a

Figure 3.1.9

9 (B) A pack now has more than 5 nodes. So the pack splits and sends the central node to the parent pack.

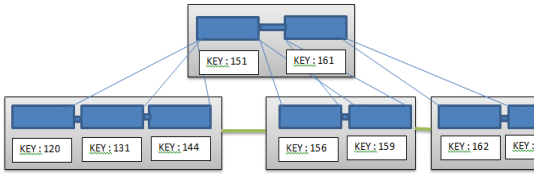


Figure 3.1.9b

SPLITTING

When violations to the maximum number of nodes to present in a node pack occurs then splitting and restructuring of the network overlay takes place. In our system this splitting occurs when the number of nodes in a node pack reach 6. In the following scenario the splitting always occurs at the third node in the node pack, we reach that node by checking the condition “Number of nodes on left+1 =Number of nodes on right” when it is met the 3rd node is found and splitting occurs at that node. The nodes on the right and left of this node go to a lower level and become the right down and left down nodes of the node chosen for splitting. Suitable updation of all the pointers takes place for all the nodes in the vicinity node packs. After this check has been performed the splitted node right up and left up nodes are checked and if they are present a request for insertion is generated with the ‘Direction’ set to ‘From_Split’.

DELETION

Whenever a node gets disconnected from the network while it goes offline or is fails due to unseen reasons all of its neighbours come to know about this through the disruption in the sockets previously created. Thus it becomes the responsibility of the left and right nodes to maintain the integrity of the network overlay and hence they generate certain requests in order to handle the rising situations. The different cases for node deletion are as follows:-

• Node deleted belongs to leaf pack : In this scenario there are two different cases possible they are :

The node deleted is at the extreme of the pack : When such a situation occurs the immediate neighbours of this node (left node and right node) update their pointers and also request their sibling and parent node packs to update their pointers to accommodate for the deletion of the node.The node deleted is not at the extreme of the pack : In such a situation the left and right nodes of the deleted nodes find each other by sending a request to their parent node packs and after finding each other they inform their sibling node packs to update their pointers respectively.

• Node deleted belongs to an internal pack: On encountering such a situation the left most node the right child of node deleted sends a request of merging to the right most node the left child of the node deleted. This request of merging is propagated down to each level for these nodes. It might lead to the rising of a situation where the number of nodes in the newly formed merged node packs is greater than 5 and hence appropriate splitting takes place.

There is a possibility in deletion that the minimum number of nodes required in pack i.e. 2 is violated and this is known as ‘Single node anomaly’, in order to encounter such situation the overlay maintains its integrity by either borrowing a node from its sibling or merging the node with one of its siblings node pack. This decision is based on the following cases :

International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 553-560 Jan 2019-June 2019 www.csjournals.com

• If the sibling pack has more than two nodes , a node is borrowed from the sibling
• If the sibling has exactly two nodes the overlay merges the two sibling by bringing down the common parent.

Example of Deleting a nodes from the network

Let the structure given below represents the current network state.

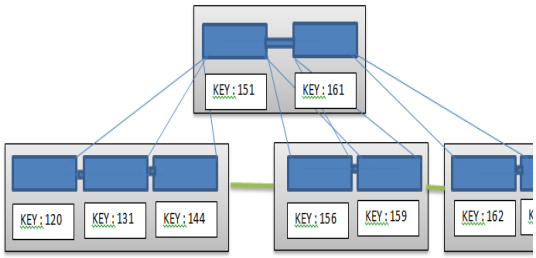


Figure 3.3

1) NODE_120 wants to leave the network. Since it is a leaf node, It informs NODE_131 and NODE_151 before leaving. The sibling and parent node make pointer adjustments accordingly.

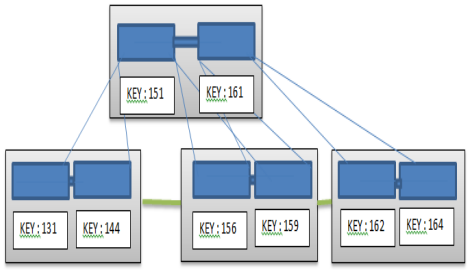


Figure 3.3.1

2) A) NODE_151 wants to leave the network. Since it is an internal node, It commands its children to merge with each other

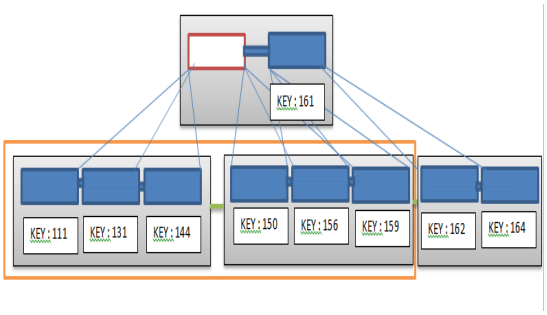


Figure 3.3.2a

2) B) the merged leaf sends a split up packet which splits every pack with more than 5 nodes

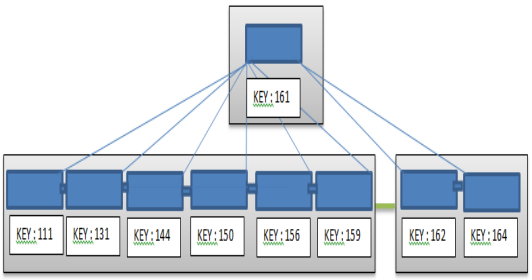


Figure 3.3.1b

2) c) A pack splits up an sends NODE_150 up.

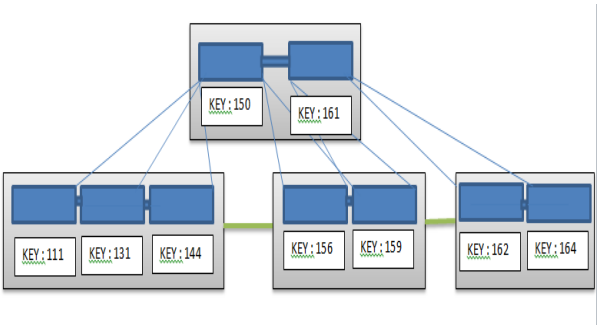


Figure 3.3.2c

SEARCHING

In order to search for a particular node in B Tree architecture the searching node must know about the key of the node to be searched. It does so by issuing a searching request which is a JSON packet, containing the same information as that contained by the insertion JSON packet

IV CONCLUSION

We have successfully designed a peer-to-peer distributed system, where one node can share a file with any other system on the same network. The network application built exhibits features like-

1) Completely decentralised

The network architecture built is free from any kind of central server making our system completely decentralised. Removing all kind of servers has helped us to remove a bottleneck value for the network. Also, P2P is reliable as there is no dependency on the server in center. If one peer fails, it will have no affect on the other peers of the network. All the resources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources. There is no need for full-time System Administrator. Every user is the administrator of his machine. The over-all cost of building and maintaining this type of network is comparatively very less.

2) Efficient Searching of systems-

The system is characterized in the form of a B-Tree, where each node represents a distinct pair of IP address and port number. Let us search here the key named k. We begin from root and do simultaneously go to the bottom. We return to positive if every non vivited leaf has the key. Otherwise we come back to the apt child till we reach the leaf node. The searching takes O(logn) complexity, which considerably saves time.

For example, a network which has about 220 nodes, a node can be searched with at most 15 hops.

3) Key based Routing-

The system successfully implements key based routing. The keys are assigned to a system based on the timestamp, when the system requested to connect to the network. These keys are used to for searching a system in the network. Every system in the network should have a unique key to as to maintain system integrity.

Another feature of our architecture is that no central server is used to maintain a record of the key or assigning a key to a system. The network handles all this features by itself.

4) File sharing-

The application built is able to successfully share a file with any other system on the network. To get a file, all a system needs to know is the key of the host. The searching algorithm returns the IP address and port number of the host. Based on this information a socket is set up between the two systems(request generator and file host). The file is then transferred by the host using the socket.

5) Network Integrity-

Whenever a node gets deleted, the network readjusts itself to maintain the property of a b-tree. This happens in the case of inserting a node to the network. Maintaining this property helps in stopping to network to turning into a linear list. As a result, we could always search a node in log n time.

6) Cross-platform

The application built can be run on multiple platforms. This is because, it has been programmed in Java, which help it target multiple platforms. Java is cross platform in the sense that a compiled Java program runs on all platforms for which there exists a JVM. This holds for all major operating systems, including Windows, Mac OS and Linux.

FUTURE SCOPE

1) Though the application is successfully implemented, and sharing of file is done effectively, there are situations where this application can be enhanced or modified to be utilized more efficiently. The system implemented is a general architecture and can be specialized to take full advantage of the B-tree architecture. Given below are a few possibilities this application can be extended to-

2) Allocation of keys based on Geographical topology-

In the current system, keys are assigned based on the timestamp of the system to assign a unique key. Keys play an important role is assigning location to the nodes in the B-tree. Generally, nodes with keys close to each other are close to each other in the tree.
If we can assign nodes key based on their geographical locations, we could reduce the time for sharing a file. In the present system, to search a system few meters away, a packet might be sent via a node miles away. If keys are assigned based on the geographical location, nodes closer to each other will have keys closer to each other. This will help us put those nodes near to each other in the B-tree helping us remove overhear of sending packets miles away unnecessarily.

3) Use of Distributed hash Tables-

In the current system, a system has 6 links to other nodes in the tree. Based on the key to be searched, the packet follows a fixed path to find a node in the network. This load on the root node and could result in loss of packets. There could also be an issue of congestion, if the path is kept fixed.
Dynamically changing distributed hash table could solve this issue.

4) Implementing security-

Security is one of the major issues in peer-to-peer systems. Authentication of a node trying to connect to the network is an important functionality that can be implemented in our application. Currently, any node with IP address and port number of any node on our network could connect to our network. Based on the requirements, authentication can be implemented on this architecture.
Authorization is another important functionality that has to be implemented based on the requirements. There may be a case in which a host node wishes to share its resources with a few nodes and nod not ever node on the network. Thus only a few nodes should be authorized to access resources in this case.

5) Implementing for Ad hoc network-

The present system implemented using the internet, i.e. packet are sent over the TCP/IP. This architecture and algorithm can be modified to be used in implementing a mobile ad hoc network. A mobile ad hoc network is a continuously self-configuring of devices connected without wires.
Each device in a network can move independently in any direction of its choice. Each one will have to behave as a router.
We can assign closer keys to nodes closer to each other. By this, we could even share files with devices that are out of our wireless range. This kind of network can be used in universities for sharing files, The slight modification is that instead of sharing file directly from the host, now the packets will be sent through various nodes in the network.

REFERENCES

[1] homas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algortihms(2010)

[2] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany (Print ISBN 978-3-540-42800-8)

[3] Detti A., Pomposini M., Blefari-Melazzi N., Salsano S., and Bragagnini A., (2012) “Offloading cellular networks with information-centric networking: The case of video streaming,” in 2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), (pp. 1–3).

[4] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Newsletter ACM SIGCOMM Computer Communication Review – Proceedings of the 2001 SIGCOMM conference (Volume 31 Issue 4)

[5] Harami C., Gazdar A., Jamelli I., and Belgith A., (2015) "Study of VOD streaming on BitTorrent," IEEE International Symposium on Networks, Computers and Communications (ISNCC).