International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 565-570 Jan 2019-June 2019 www.csjournals.com

Peer-to-Peer Distributed Video Streaming System Using Binary Tree Topology

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: client server architecture offers a great deal of benefits over monolithic systems, but they come with many disadvantages too. The biggest one is the dependency of the entire network on a central server. This central dependency introduces a single point of failure for the entire network. So, if the main server crashes or is unreachable, the network is no longer operational. There are a few cases where we can achieve our purpose without a central server(It is not always possible to get rid of the central server). A Peer-to-Peer or adhoc network is formed when a collection of multiple nodes/computers offering a similar service come together eliminating the need of a central server. These Peer-to-Peer networks can be used for vast applications like video streaming, file sharing, distributed computation, etc.
Here I aim at designing a Peer-to-Peer on-demand video streaming system for faster video streaming of a given video that is present over multiple nodes/computers on that network. The node that wished to watch the video will be referred as the WATCHING node. Also, the nodes that host the video and stream video segments for the WATCHING node will be called the STREAMING nodes. The entire video can also be streamed from a single peer that hosts that video. But since the intake speed is usually lesser than the outgoing speed, the overall streaming speed for the WATCHING node will be limited by the STREAMING peer's upload speed. To overcome this problem, the WATCHING node/peer can take fragments of the video from more than one STREAMING nodes that host the video. With this, the WATCHING node/peer can stream in the video without any buffering or lag.
Here, we are going to use binary trees as the Network topology. This binary-tree topology will help us in generating routing addresses to the nodes which will help us in forwarding the video packets/segments to the destination using the most efficient path.
Keywords-Topology, binary, data rate, point to point.

I. INTRODUCTION

A Peer-to-Peer (P2P) network is created when many identical computers/nodes are connected with each other. These nodes share some resources (hardware, file) between them. P2P network can also be a permanent arrangement that can link a few computers in a small university or office over LAN connection. A P2P network can also be a network on a much bigger scale in which dedicated applications and protocols are used to set up an uninterrupted relationships between users over the Internet[1-2].

A. Distributed video streaming 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 outgoing speed of the streaming node.
2) If the streaming node goes down while streaming, the streaming process will fail.

Single Streaming Peer

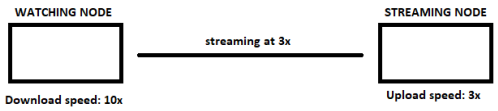

Fig. 1 single Streaming peer

Multiple Streaming Peer

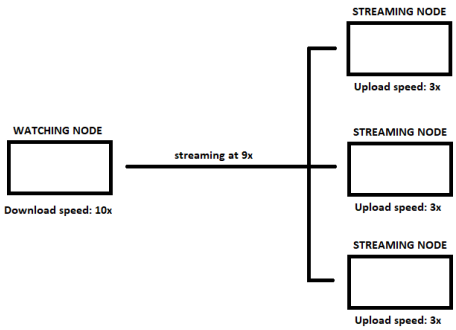

Fig.2 Multiple Streaming Peer

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 achieving the maximum

International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 565-570 Jan 2019-June 2019 www.csjournals.com

streaming speed. It will also remove the dependency from a single streaming node.

B. Network Topology

Network Topology

Network topology refers to the logical and hierarchical manner in which computers are computers are connected to each other(or some network device). It describes the layout pattern of the interconnections between the computers. The Network Topology can also be called as "network architecture."

An application running on machines connected to the binary-tree network is known as a 'node.' Network Topologies are classified on the basis of manners in which the nodes are connected with each other. Network topologies determine the way machines/apps transfer data to one another over a given network network. There are a number 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) **P2P:** 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. BINARY TREE NETWORK TOPOLOGY

Binary tree is a hierarchal data-structure, that is generally used for quick searching and sorting of data in memory. We are going to use this data structure as a network topology for quick searching of a given video over the network. Having a binary tree like hierarchal structured topology, we get the advantage of routing address generation and allotting them to the nodes. The transfer of data packets from given streaming nodes/peers to the watching node is also efficient.

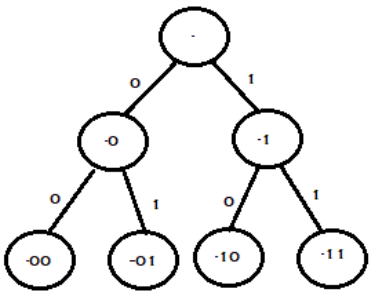

Fig. 3: Binary-Tree Network Topology

Above is an illustration of a binary-tree P2P network, where different nodes come together to form a binary-tree like structure. The root node is assigned the address "–".

1. If a peer/node gets added to the left of a node with id x, the address of the new node will be **x0**.

2. If a peer/node gets added to the left of a node with id x, the address of the new node will be **x1**.

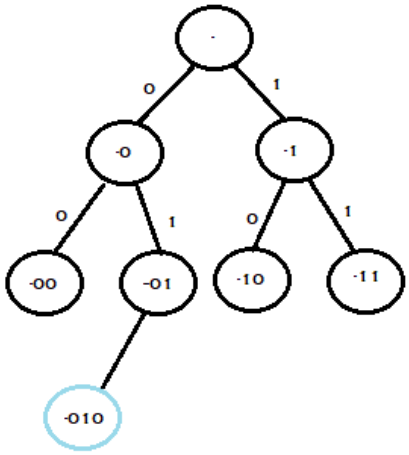

Fig. 4 Addition of new node

In the example above, since the new node (blue) is added to the left of 10-, its id is 010-

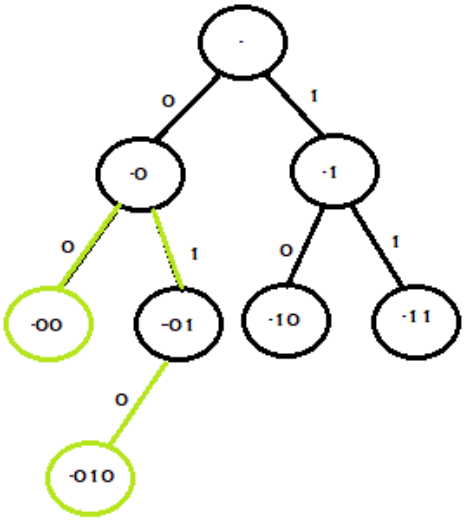

Fig. 5 Routing message from one peer to another

If (**NODE-010**) wants to send message to (**NODE-00**) then it will send it via (**NODE-0**) which is their common parent node.

A. Message Routing Algorithm

NODE-010 (from)

NODE-00 (to)

International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 565-570 Jan 2019-June 2019 www.csjournals.com

We can see that in (**NODE-010**) and (**NODE-00**), (**NODE-0**) is the common part. This is the routing address of their common ancestor.

In (**NODE-010**) and (**NODE-0**), **NODE-0** is common and 01 is different. Since length of 01 is 2, the video frame has to be sent to 2 levels above to reach **NODE-0** (common ancestor).

Further, in (**NODE-00**) and (**NODE-0**), **NODE-0** is common and 0 if different. But since the message has already reached (**NODE-0**), we will only be concerned about the difference. Since 0 represents left, the message has to be sent to the left child of (**NODE-0**). That will be **NODE-00**

B. Why are we using Binary Tree Topology

Binary tree give us an advantage that is not possible in many topologies. There is a hierarchical relationship between nodes. This hierarchical relationship can be used to allot routing address to the nodes/peers, which can be used in routing of video frames.

C. Socket

A socket is bounded to a specific port number of a machine on which the program is. The client then makes a connection request to the server program, which could be on same or different machine, and server application waits and listens to the socket when client makes a request. The client application identifies itself to the server application and is bound to a local port number which is used during the connection.

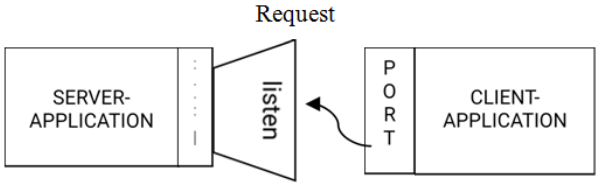

Fig. 6 Connection Request

Server then accepts the request, if everything is going as per the server' requirement. After accepting the request, the server catches a new socket destined to the same local port. The server's end point is set to the address and port of the client. It needs a new socket to continue listening to the original socket for connection requests.

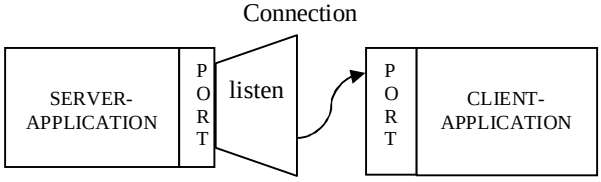

Fig. 7 Socket Connection

If the connection is accepted on the client side, a socket is created successfully and the client then uses the socket to communicate with the server.

A communication is now build up between client and a server and they communicate by writing to and reading from their sockets. An end point is comprised of a port number and an IP address. Given these two end points, every TCP connection is uniquely identified. This gives us the liberty of having multiple connections between our host and the server.

III. IMPLEMENTATION

Technology used: JAVA + Socket programming

There are different types of messages that we are using to be transferred in our applications. These messages will be used for establishing connection, requesting for a video and sending the video frames.

Type of messages communicated:

- **ADD_STREAMING_NODE**
- **ID_ASSIGNMENT**
- **ID_ASSIGNMENT_FAILURE**
- **VIDEO_SEARCH**
- **VIDEO_SEARCH_RESPONSE**
- **VIDEO_FRAME_REQUEST**
- **VIDEO_FRAME_RESPONSE**

*A. **ADD_STREAMING_NODE***

This type of message is a request to connect to the network that is sent from a connecting/new node to a network/tree node (that is already part of the tree network). It is sent only once.

Message direction: NEW NODE -> TREE NODE

RESPONSE OF THE NODE THAT RECIEVES THE MESSAGE:

When a tree node receives this request/message it will try to add the new node as its left or right child. Once added successfully, it will send an ID_ASSIGNMENT message to the new node which means that the new node is now a part of the network. However, if It already have both right and the left child, it will send an ID_ASSIGNMENT_FAILURE message, which means the new node cannot be connected to the tree/network by requesting to this node.

MESSAGE FORMAT:

```
{
 "type": "ADD_STREAMING_NODE",
 "content": null
}
```

Identify applicable funding agency here. If none, delete this text box.

International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 565-570 Jan 2019-June 2019 www.csjournals.com

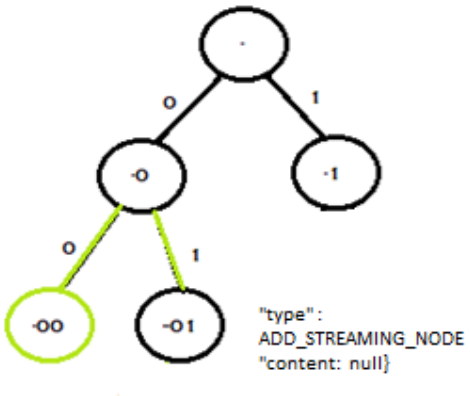

Fig. 8 Adding streaming node

B. *NODE_ID_ASSIGN*

This message is a response to the ADD_STREAMING_NODE message. When a node succeeds adding a new node as its left or right child, it sends this message to the new node. The content of this message is the Logical id of the new node that is being connected to the network and its value is created by adding 0 or 1 in front of the id of the existing node.

RESPONSE OF THE NODE THAT RECIEVES THE MESSAGE:
When the new node receives this message, it marks the node that sent this message as its parent.

SAMPLE MESSAGE:
{
"type": "NODE_ID_ASSIGN",
"content": "01-"
}

So, 1- adds the new node as its left child an sends the above message to that node. The new node now knows that its id is 01- from the content.

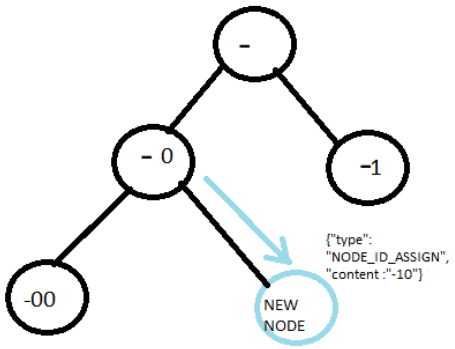

Fig. 9 Assigning Node Id

C. *NODE_ID_ASSIGN_FAILURE*

This type of message is a response to the ADD_STREAMING_NODE message. When an existing node fails to add a new node as its left or right child (it already had a left/right children), it sends this message to the new node. The content of this message is null

RESPONSE OF THE NODE THAT RECIEVES THE MESSAGE:

"Failed to connect to the network."

SAMPLE MESSAGE:
{
"type": "NODE_ID_ASSIGN_FAILURE",
"content": null
}

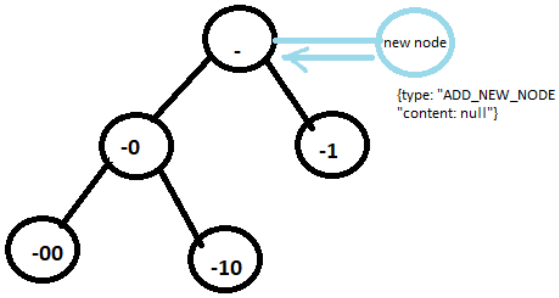

Fig. 10 Node ID assignment failure

D. *VIDEO_SEARCH*

Whenever a node wants a see a particular video, it will broadcast this type of message. Since a node only knows about 3 adjacent nodes (left child, right child and parent), it will send this message to them. The message content will contain the ID of the original node that requests that video information and the video name.

RESPONSE OF THE NODE THAT RECIEVES THE MESSAGE:

Case 1: Video is present with the node that receives the request:
Whenever a node receives file request message, it will 1st check if it has that video. If it is present, it will send the VIDEO_SEARCH_RESPONSE message to the requesting node with information like the number of frames (The video search request message contains the address). Oher than that, it will also forward the video search message to other 2 adjacent links.

International Journal of Electronics Engineering (ISSN: 0973-7383)
Volume 11 • Issue 1 pp. 565-570 Jan 2019-June 2019 www.csjournals.com

Case 2: Video is not present with the node that receives the request:
If the video is not present, the node will just forward the message to its other 2 adjacent link. For example, if a node receives the request from its left child and it does not have that video; it will forward the request to its parent and its right child.

SAMPLE MESSAGE:

{
"type": "VIDEO_SEARCH_REQUEST",
"content": "requesting_node:0-,file_name:BIKE_RIDE"
}

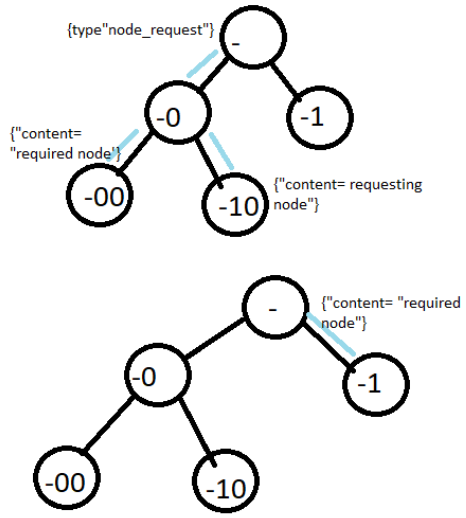

Fig. 11 Node Request

*E. **VIDEO_SEARCH_RESPONSE***

Whenever a node receives a request for a video search that it has, it will create a response message and send it on the appropriate path. The content of the message will be the destination address (which is the same as the requesting node in the FILE_REQUEST), the information of the video (file name, number of frames, etc.) and the logical address of the node that hosts the video.

RESPONSE OF THE NODE THAT RECIEVES THE MESSAGE:

Case 1: The Destination address is the same as the receiving node address:
If the destination address in the message is the same as the address of the receiving node, it will start requesting the video segments from all those nodes from which it has received the video search response.

Case 2: The Destination address is different from the receiving node address:
If the destination address in the message is different from the address of the receiving node, it will forward the message to its appropriate adjacent node. This can be found by comparing the destination id with the receiving node's id. It will be done using the following algorithm:
We see that in (NODE-**010)** and (NODE-**00)**, (NODE-0) is the common part, which is the address of their common parent.
In 010- , 0- is common and 01 is different. Since length of 01 is 2, the message has to be sent to 2 levels above to reach 0-(common parent).
Now in (NODE-00) NODE-0 is common and 0 if different. But since the message has already reached **(0-)**, we are only concerned about different. Since 0 represents left, the message has to be sent to the left child of 0-. That will be 00-.

SAMPLE MESSAGE:
{
"type": "VIDEO_SEARCH_RESPONSE",
"content": "destination_address:0-,
video_frame_count:200, video_name:photo.jpg",
streaming_node_address: 10-
}

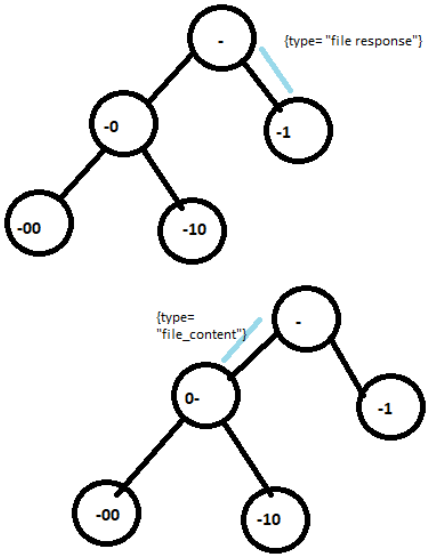

Fig. 12 Video Search Response

*F. **VIDEO_FRAME_REQUEST***

Whenever a node wants a specific frame of a video, it will send this type of message to one of the streaming node (from which it has received the video search response message). The message routing logic will be the same as video search response. In this message, the watching node will request for a single frame of a given video. So it will provide the video name, frame number, watcher node's address (requesting_node) and the node from which we are requesting the frame (destination_node).

RESPONSE OF THE NODE THAT RECIEVES THE MESSAGE:

Case 1: Node is the destination node:
Whenever a node receives file request message, it will 1st check if it is the destination node or not, by comparing its ID with the destination ID. If it is the destination node, it will send the VIDEO_FRAME_RESPONSE message to the requesting node (The video frame request message contains the address).

Case 2: Node is not the destination node:
If the node is not the destination node, the node will just forward the message to its other 2 adjacent link. For example, if a node receives the request from its left child and it is not the destination node; it will forward the request to its parent and its right child.

SAMPLE MESSAGE:
{
"type": "VIDEO_FRAME_REQUEST",
"content": "requesting_node:0-, destination_node:10-, video_name:BikeRide, frame_number:12 "
}

*G. **VIDEO_FRAME_RESPONSE***

Whenever a destination node receives a request for a video frame, it will create a response message and send it on the appropriate path. The content of the message will be the destination address (which is the same as the requesting node in the FILE_REQUEST), the data of the video frame and video's full name and frame number.

RESPONSE OF THE NODE THAT RECIEVES THE MESSAGE:

Case 1: The Destination address is the same as the receiving node address:
If the destination address in the message is the same as the address of the receiving node, it will save the video frame in the video displayer.

Case 2: The Destination address is different from the receiving node address:
If the destination address in the message is different from the address of the receiving node, it will forward the message to its appropriate adjacent node. This can be found by comparing the destination id with the receiving node's id. It will be done using the following algorithm:

We see that in (**010-**) and (**00-**), (0-) is the common part, which is the address of their common parent.
In **NODE-010**, 0- is common and 01 is different. Since length of 01 is 2, the message has to be sent to 2 levels above to reach 0-(common parent).
Now in (00-) 0- is common and 0 if different. But since the message has already reached **(0-)**, we are only concerned about different. Since 0 represents left, the message has to be sent to the left child of 0-. That will be 00-.

MESSAGE FORMAT:
{
"type": "VIDEO_FRAME_REQUEST_RESPONSE",
"content": "destination_address:0, video_frame_data:"hjdgfs76", video_name:BikeRide, video_frame_number:12"
}

IV. CONCLUSION

The major disadvantages for any client-server architecture are its live dependency on a central server. This dependence introduces a single point of failure for the network which means that if the main server fails, the network stops functioning. Though getting rid of the central server is not always possible, there are some cases that could be solved more effectively with a server-less architecture. In this paper I have successfully assigned address to each peer which is acting as a node of the binary tree, for the quick look up of the file and transferring the video from one streaming node to another.

REFERENCES

[1] Sodagar I., (2011) "The mpeg-dash standard for multimedia streaming over the internet," IEEE multimedia, vol. 18, no. 4, (pp. 62–67)
[2] Lua E.K., Crowcroft J., Pias M., Sharma R., Lim S. (2005) A Survey and Comparison of Peer to-Peer Overlay Network Schemes. IEEE Communications Surveys and Tutorials, Second Quarter, 7(2).
[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] Daswani N., Molina H.G., and yang B., (2003) "Open Problems in Data-sharing Peer-to-peer Systems," 9th International Conference on Database Theory.
[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).