Algorithms for Reliability in Large Scale Structured and Unstructured Peer-to-Peer Overlay Multicast Networks for Live Streaming

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

by

Ashutosh Singh

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY 2016

CERTIFICATE

It is certified that the work contained in the thesis entitled "*Algorithms for Reliability in Large Scale Structured and Unstructured Peer-to-Peer Overlay Multicast Networks for Live Streaming*" by *Ashutosh Singh* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

20 July 2016

[illegible]

Dr. Yatindra Nath Singh
Professor,
Department of Electrical Engineering,
Indian Institute of Technology Kanpur,
Kanpur, U.P. - 208016 India.

Let not Ambition mock their useful toil,

Their homely joys, and destiny obscure;

Nor Grandeur hear with a disdainful smile

The short and simple annals of the poor.

Thomas Gray in *"Elegy Written in a Country Churchyard"*

Dedicated

to

those lesser knowns,

who devote their life working

honestly in their area, with a passion

to make our world a better place

to live, without any ambition

for fame, awards

or accolades.

Acknowledgements

I would like to express my sincere gratitude toward my thesis supervisor, Dr. Yatindra Nath Singh, Professor, Electrical Engineering Department, IIT Kanpur, Kanpur. His vision and suggestions throughout the thesis work from choosing the problem, solving and presenting has been fundamental in the completion of the thesis. He always encouraged me to apply my own thinking and come up with my own ideas. This helped me a lot in improving my research abilities. His dynamic approach to the research problem and guiding the students is remarkable. His cooperation, understanding and patience are incredible. His ideologies and ethics has molded my attitude too. The confidence he has shown in me, has had a immeasurable effect on me and on my work. He taught me what research is, how it is performed, and how it is communicated. He also provided me with lots of hints to improve my professional behaviour. He taught me to see the silver lining in every dark cloud, helping me to develop analytical thinking and making me firmly believe that research is an attitude, a philosophy of life. He is like a guiding lighthouse for me, where I can see upon whenever I am uncertain.

My colleagues at HBTI Kanpur and lab mates at IIT Kanpur deserve more than a mention here. I acknowledge my colleagues Dr. Rachna Asthana, Dr. A.K. Shankhwar, and Mrs. Rajani Bisht for their for their companionship, camaraderie and support. I acknowledge my labmates Anurag, Prashant, Rameshwar, Anupam, Sateesh, Basavraj and Upkar for their help. My family members have always been a constant source of encouragement. I am grateful to them for bestowing unconditional love throughout my life and for their unflagging support. I sincerely acknowledge my parents, wife, daughter Shatakshi and son Shashwat for their affection, understanding and patience.

(Ashutosh Singh)

SYNOPSIS

Name of the student:	Ashutosh Singh
Roll number:	Y7104093
Degree for which applied:	Doctor of Philosophy
Department:	Electrical Engineering
Thesis Title:	Algorithms for reliability in large scale structured and unstructured peer-to-peer overlay multicast networks for live streaming
Name of the thesis supervisor:	Professor Yatindra Nath Singh
Month and year of submission:	July 2016

The advances in Information and Communication Technology (ICT) mingled with the explosion in growth of internet users in 21st century has completely changed the teaching and learning scenario and complemented the traditional classroom teaching. We can expect a Live Lecture Delivery Systems (LLDS) to be used for instruction, where a large heterogeneous population across the different corners of the world get a multicast delivery from a source simultaneously with provisions for two-way communication, thus creating a virtual classroom environment.

Initial multicast solutions were based on IP (Internet Protocol) or network layer multicast technology. IP multicast systems though works efficiently but have many limitations too. Since networks were primarily designed for unicast communication, IP multicast requires changes at infrastructure level in the Internet and hence is not easily deployable. In fact, multicast exists only in fragmented islands here and there in the internet. Further IP multicast is based on best-effort data delivery, and hence cannot support QoS. Moreover it requires routers to maintain per-group state which leads to scaling constraints.

With the growth of peer-to-Peer (P2P) network research, many protocols have been written for multicast over application layer, where a self organized overlay of participating hosts is formed and multicast-related functionalities are moved to the end-hosts. Application Layer Multicast (ALM) achieves multicast via piece-wise unicast connections. Though not as efficient as IP multicast, the overlay multicast has alleviated the problems due to fragmented IP multicast islands and QoS constraints in IP multicast. Two important mechanisms, one for resource discovery and another for data distribution are required to run a P2P multicasting network (ALM) for any application.

In this paragraph, a brief background of P2P networks is given. Their discovery mechanisms are described in the next one. In a Client/Server (C/S) model, the server is a central registering unit and the only provider of content and services. A client only requests content or the execution of services, without sharing any of its own resources. In contrast to C/S model, in P2P model, processing load and network bandwidth use is distributed among all the nodes due to decentralization. Participants in P2P network can act as a server and as well as a client at the same time. They are accessible to other nodes directly without needing any intervention of a central controller. P2P systems constitute highly dynamic networks of peers with complex topologies that create a self organizing overlay network. P2P applications however need sophisticated discovery mechanisms to enable peers to find, identify and communicate with other peers which hold the content or services of interest.

On the basis of discovery mechanism used, P2P networks fall into two categories, viz. unstructured and structured networks. In *unstructured approach*, the placement of the data is completely unrelated to the overlay topology and searching is mostly based on the flooding or random walk. Another alternative for query search in unstructured networks is based on Gossip protocol. In *structured approach*, the query network topology is tightly controlled and the indexes are placed at specific locations. The lookup service is implemented by organizing the peers in a structured overlay network. A mapping between the keyword identifier and location is provided in the form of a distributed routing table called Distributed Hash Table (DHT). Examples of such networks are Freenet, Chord, CAN, Pastry and Tapestry.

Initial ALM proposals used P2P infrastucture as substrate and targeted small group sizes. First phase ALM protocols were divided into two categories on the basis of whether peers were directly involved in the process of overlay construction (pure P2P, *e.g.* Yoid) or on their behalf, service nodes form the overlay and peers connect to these service nodes (proxy based, *e.g.* Scattercast). As far as live video streaming application is concerned, the challenege in both these architectures was how to form an overlay that is efficient in both bandwidth as well as latency. Based on the sequence of construction of data topology and control topology, protocols were classified as Direct Tree (*e.g.* Yoid) and Mesh-first protocols (*e.g.* Narada).

In the second phase, scalable protocols will have members being assigned addresses based on some abstract coordinate space (*e.g.* Delaunay Triangulation (DT) based and Content Addressable Networks (CAN) based overlays, or being organised in hierarchies of clusters (*e.g.* NICE and Kudos) that made the requirement of a formal message routing protocol redundant. In the third phase, protocols made use of DHT based lookup protocols for resource mapping with increased scalability and efficiency. Pastry based Scribe protocol which is used as large scale event notification system, is a representative protocol of this phase.

For streaming ALM protocols, research efforts are towards higher reliability, fair load distribution, free riding (uncontrolled noncooperative users' behavior) prevention and flash crowd handling. The characteristics and thereby the issues that needs to be investigated in an ALM, specifically designed for large scale Live Lecture Delivery System (LLDS) application, are the following.

(i) The live streaming applications are a delay as well as bandwidth sensitive and demand high QoS. The quality degrades with every byte of data loss and it becomes intolerable if interruption happens due to some node or link failure. Thus reliability is a major issue

(ii) A low overhead mechanism is required in unstructured approach to help efficient query of resource

(iii) Since achieving reliability is the primary goal in a live streaming application, more than one spare paths need to be maintained. Even after failure, reorganisation of network is needed to reestablish more than one transmission paths. Thus both proactive and reactive components are needed

(iv) Since we target a large scale ALM protocol, an efficient mechanism for Resource mapping (for resource address storage and retrieval) is required in the structured approach so that every query is satisfied without bothering one single root node. It is done by caching responses to the queries

(v) With the changes in the network dynamics, the distribution tree may become inefficient with time and hence continuous tree optimization is required

(vi) In case of homogeneous environment, we need to ensure uniform load sharing among nodes and in heterogeneous environment, high capacity nodes should be brought near the source node and low capacity nodes should be moved toward the periphery as leaf nodes through continuous topology reorganisation

(vii) In the beginning of a session, a large number of subscribers join the session in a very small duration. This phenomenon is known as flash crowd that results in increased joining latency and refusal rate. To handle a large crowd in the beginning of a session, a single root node may not be enough. Also, we can not maintain many root nodes specially after flash crowd. Thus a mechanism is required that increases and decreases the number of root nodes in proportion to the amount of flash crowd arriving in the network.

(viii) a subscriber may be a member of many different streams launched simultaneously, and groups may have large number of members. An efficient mechanism to cater large population and large number of simultaneous live streams is required.

The work done has been organized in seven chapters in the present thesis. We start with an introduction to the Peer-to-Peer (P2P) network in Chapter 1. The advantages of P2P model and its discovery mechanism are described. The possible approaches for multicast and use of overlay multicast for Live Lecture Delivery is also discussed in this chapter.

In chapter 2, a survey of Application Layer Multicast (ALM) protocols is given. The chapter begins with the definition of performance metrics of

ALM. Classification of ALM protocols from different perspectives, and then description of some popular ALM protocols is given. Open issues in ALM are dicussed finally.

Reliability approaches in Overlay Multicast Networks are discussed in Chapter 3. A survey of Reliability approaches is given. Classification of reliability approaches, brief description of major reliability protocols for streaming and the performance metrics for a reliability protocols are discussed in this chapter.

The approaches for the maintenance of biconnectivity for reliability in unstructured overlay network are proposed, analyzed and compared in Chapter 4. Algorithms for data distribution with duplicate paths are also discussed in this chapter.

In chapter 5, the dual path approach for reliability in query network based (structured) overlaid multicasting is discussed. The three approaches for feed managemant are given. PeerSim based simulation results verify that the differential delay in two paths and startup delay are well within tolerable limits and with moderate to high failure rates, only a small fraction of nodes get deprived of feed only for a small duration.

The issue of flash crowd is taken up in chapter 6. The distributed and scalable query handling algorithm with cache updation and feed forwarders list maintenance algorithm are given and simulation results verify the effectiveness of the algorithms.

Finally, conclusions, scope for future work and summary of contributions of the present thesis have been presented in chapter 7.

List of Abbreviations and Acronyms

ALM	**A**pplication **L**ayer **M**ulticast
CAN	**C**ontent **A**ddressable **N**etwork
CDN	**C**ontent **D**istribution **N**etwork
DHT	**D**istributed **H**ash **T**able
DT	**D**elaunay **T**riangulation
DVMRP	**D**istance **V**ector **M**ulticast **R**outing **P**rotocol
FEC	**F**orward **E**rror **C**orrection
HMTP	**H**ost **M**ulticast **T**ree **P**rotocol
LLDS	**L**ive **L**ecture **D**elivery **S**ystem
MDC	**M**ultiple **D**escription **C**oding
MST	**M**inimum **S**panning **T**ree
NICE	**N**ice is the **I**nternet **C**ooperative **E**nvironment
OMNI	**O**verlay **M**ulticast **N**etwork **I**nfrastructure
OMTP	**O**verlay **M**ulticast **T**ree **P**rotocol
PRM	**P**robabilistic **R**esilient **M**ulticast
RTT	**R**ound **T**rip **T**ime
SPT	**S**hortest **P**ath **T**ree
TFRC	**T**CP **F**riendly **R**ate **C**ontrol
TTL	**T**ime **T**o **L**ive
URI	**U**niversal **R**esource **I**dentifier

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to special biological characteristics, humans can produce innumerable vocal signals to communicate their need and experience. Human communication was revolutionized with speech approximately 100,000 years ago [4]. Communication via speech is a unique blessing to humans that differentiates them from other living beings. Survival of any human society depends on communication. With time a symbolic system called language got developed that helped to communicate not only by utterance but also by writing and also allowed context free communication. No languageless community has ever been found [5]. Language has alphabets that help to convey the message beyond the boundary of time and space to some extent. The communication process comprises a sender, a message and the intended recipients. The most basic communication system consists of two human beings with the start of message from one's mouth and ending on another one's ear (Mouth-to-Ear communication). Even in most complex communication system, the end points remain the same.

With advancement in technology, the restriction of geographical and temporal boundaries got relaxed further. The information generated at one instant of time can be sent to a number of places simultaneously at the same instant or it can be preserved and reproduced again and again.

Electronic communication is what expanded the horizons of communication and really boosted the communication industry. Sending information at almost the speed of light was possible by converting it in the form of electrical signals in Telegraph which started in 1837. Four decades later after the invention of Telegraph, the invention of Telephone by Alexender Graham Bell in 1876 was the first major leap towards all the modern methods of communication. The telephone system though still existing and supporting voice signals, the transmission however has evolved from circuit switched to packet switched and from using copper cables to optical fibers.

During the twentieth century, the key developments has been for information storage, processing and its distribution that includes invention of Television (John Logie Baird, 1926), satellite broadcasting, Computer (1945), Internet (1973) and World Wide Web (1990). Optical fiber technology has created almost limitless opportunities for digital networks with greatly enhanced capabilities. Public broadband integrated service networks that provide integrated data, voice and video on a global level became technologically feasible.

Now-a-days internet has become the most important source of information and communication throughout the world. Through Internet the live telecast of an event is possible to millions of people situated all around the globe. The internet is linked to almost every major organization, business and government. In the era of high-definition and 4G one can see a football game as if he is sitting in the stands. With 4G wireless technology, the transmission of crystal clear sounds and pictures is possible. Now, we can download information at the speed of light, practically, can dial up a long-distance telephone number in a matter of seconds, and can get new programs on our computer through our phone lines.

1.1 Computer Networks

The merging of computers and communications has changed the way of computation. The old model of a single computer serving all of the organization's computational need has been replaced by the one in which a large number of separate but interconnected computers do the job. These systems are called computer networks. Computation in these networks could be centralized or distributed. In centralized computation, one node processes the entire application locally while in distributed computation; processing steps of application are divided among the participating nodes.

1.1.1 Client-Server model or Peer-to-Peer model

In *Client-Server (c-s) model*, the server is central registering unit and the only provider of content and services. A client only requests content or the execution of services, without sharing any of its own services. The c-s model could be a flat system with single server or it could be a hierarchical system for improved scalability. In c-s model, in order to provide access with acceptable response time; sophisticated load balancing and fault tolerance algorithms have to be applied. Also limitation on network bandwidth adds the bottleneck problem.

Peer-to-Peer (P2P) model distributes processing loads and network bandwidth among all nodes and solves bottleneck problem. P2P systems constitute highly dynamic networks of peers with complex topologies that create an overlay network. P2P applications need sophisticated discovery mechanisms to enable peers to find, identify and communicate with each other. Participants can act as a server as well as a client at the same time. They are accessible by other nodes directly. The drawback with P2P model is that it needs complex algorithms.

1.1.1.1 Pure P2P and Hybrid P2P

The pure P2P networks [6] are purely decentralized. All nodes are equivalent in functionality. Any single, arbitrary chosen terminal entity can be removed from the network without loss of any of the network service. Pure P2P networks are inherently scalable and fault tolerant as there is no central point of failure. The examples are CAN, Chord, Gnutella and Freenet. The limitations with pure P2P model is that there is slow information discovery, no guarantee about quality of services and it is difficult to predict the system behavior because of the lack of a global view.

In Hybrid P2P networks, there is a central server that maintains directories of information about registered users in the network, in form of metadata. Thus it reduces discovery time as well as traffic between nodes.The hybrid P2P networks may have centralized indexing server or a cluster of decentralized indexing servers.

In Centralized approach, peers connect to a centralized directory servers that maintains an index with metadata (file name, time of creation etc.) of all files in the network and a table of connection information of registered users (IP addresses, connection speeds etc.). Upon receiving a request from a peer, the central index will match the request with the best peer in its directory and responds with the URI (Universal Resource Identifier) of the resource.The data exchange will occur directly between the two peers. Examples of such networks are Napster and DC++.

In Decentralized Indexing, Peers with sufficient bandwidth and processing power are elected as Superpeers which connect with the other superpeers forming a flat unstructured network of superpeers. Superpeers maintain the central indexes for the information shared by local peers connected to them and proxy search requests on behalf of these peers.

If a superpeer receives at least the required number of connections from client nodes within a specified time, it remains a superpeer, otherwise it turns into a regular client node. A central server provides new peers with a list of one or more superpeers with which they can connect. Thus in decentralized indexing there is reduced load on central server but there is slower information discovery. Example of Hybrid P2P network with decentralized indexing are Kazaa, and Gnutella.

1.2 P2P Discovery Mechanisms

On the basis of discovery mechanism, P2P networks can be classified as Unstructured and Structured P2P networks.

1.2.1 Unstructured Approach

In Unstructured approach, the placement of the data is completely unrelated to the overlay topology and searching amounts to (query matching or flooding based) random search. The unstructured approach is easy to implement and require little maintenance but lacks scalability. Number of messages exchanged for a search grows with number of peers. The limitation with the unstructured approach is that the search protocol is very sensitive to the number of edges in the network graph. With small number of links, all nodes will not be reachable in reasonable time, and with too many links, numerous identical copies of the query message will arrive at many nodes from different directions, resulting in wasted bandwidth.

The discovery Mechanism in unstructured P2P, could be Flooding based (used in pure P2P) e.g. in Gnutella [7] , Random Walk based or Gossip based.

1.2.1.1 Flooding based

In Flooding based approach, each peer publishes information about the shared content in the network. Any request from a peer is flooded to the directly connected peers, which flood it further until the request is answered or the query traverses the maximum number of hops. Gnutella uses this mechanism. It uses four types of messages to implement this mechanism:

1. *Ping:* a request from certain host to announce itself

2. *Pong:* reply to Ping contains the IP/port of the responding host and number and size of files shared

3. *Query:* search request contains a search string and the minimum speed requirements of the responding host

4. *Query hits:* reply to Query contains the IP/port, speed of the responding host, the number of matching files and their indexed result set

In flooding based approach, accurate discovery of peers is not guaranteed. It gives optimal results in a network with a small to average number of peers and is not scalable to very large numbers. TTL (Time-to-live) effectively segments the network into subsets, imposing on each user a virtual horizon beyond which their messages cannot reach.

1.2.2 Structured approach

In structured approach, the topology is tightly controlled and the indexes are placed at specific locations.The lookup service is implemented by organizing the peers in a structured overlay network. A mapping between the keyword identifier and location is provided in the form of a distributed routing table.

The discovery Mechanism here is DHT based. The advantage with this approach is that it reduces the number of P2P hops that must be taken to locate a resource. Peers also need to maintain information about resources, with neighboring peers to enable the index to be accessible when a node quits. This increases the maintenance efforts during situations in which nodes join and leave at a high rate. Examples of such networks are Freenet, Chord, CAN, Pastry and Tapestry.

1. *Freenet:* Freenet is an efficient decentralized file sharing protocol that supports dynamic storage and routing. It uses location independent key based routing for storage and retrieval of files. Requests are routed to the most likely physical location of data. The protocol keeps anonymity for both uploading and downloading peers.

   In Freenet each peer is assigned a random ID and Hash based ID is assigned to the documents to be shared. Each peer routes the document towards the peer with the ID that is most similar to the document ID. Freenet provides a mechanism to map the queried keyword to the document ID.

   Each request has a pseudo unique random identifier and starts with a hops-to-live (HTL) limit, which is decremented at each intermediate peer. With Identifier a request reaching to the same peer again can be detected thus avoiding loops. Request is routed to the node whose ID is closest to the requested document ID. If the data is found, it is cached on each node along the path. Therefore popular object will be cached more widely. Thus protocol is adaptive to usage pattern and hence efficient. Files are dynamically replicated in regions where it is more in demand and and deleted from regions where there is no interest.

2. *Chord:* Chord is a lookup protocol that stores key/value pairs for distributed data items. Given a key, it maps key to a node responsible for storing the key's value. Each node maintains routing information of about O (log N) other nodes, and resolves all lookups via O (log N) messages to the other nodes. Nodes' leaving/joining requires $O(log^2 N)$ messages to update the routing information.

3. *CAN:* CAN is a mesh of N nodes in virtual d-dimensional dynamically partitioned coordinate space. Each peer keeps track of its neighbours in each dimension. New peer randomly chooses a point in the space, the peer currently responsible for that point splits the space indexed by it into two and the new peer becomes responsible for one of the halves. New peer then contacts its neighbours to update their routing entries. Discovery mechanism comprises two operations: local hash-based look-up of a pointer to a resource and routing the lookup request to the pointer. Deterministic discovery in $O(N^{1/d})$ steps is guaranteed in this mechanism.

4. *Tapestry:* Plaxton *et.al.* [8] introduced the first algorithm known as Key Based Routing, for locating P2P objects and query routing based on mapping the object identifiers to the address space of peers. Local routing maps at each peer gradually route a message to the destination ID, digit by digit. This routing method guarantees the search com-

pletion within $log_B N$ hops where N is the total number of peers in the system and B is the configuration parameter. Peer's local routing map has a size of $B(log_B N)$. The PRR design is based on static set of nodes and does not cover the issues related to overlay formation and maintenance.

Tapestry [9] uses distributed search structure of [8], with additional algorithms for peer join, peer leave and overlay maintenance. In PRR's mesh, data objects are connected to one root peer, while in Tapestry there are multiple roots for each data object to avoid single point of failure. Tapestry uses surrogate routing to select root peers incrementally during both publishing and search of key-value pairs. It reduces routing overhead. The OceanStore large scale storage utility is based on Tapestry.

5. *Pastry:* In pastry[3], the routing table size complexity is O(logN), and the complexity of routing steps required are O(log N). Splitstream application layer multicast implementation is based on Pastry. The Pastry and Tapestry differ in the way of handling network locality and data object replication.

1.3 Unicast, Broadcast and Multicast in Networks

The transmission of information packets from a source node to a single uniquely identifiable destination node in networks is called Unicasting. The computer network was primarily developed for unicast applications such as http, e-mail, ftp, telnet etc.. All LANs (e.g. Ethernet) and IP networks support the unicast transfer mode. Even today Unicast is responsible for most of the traffic on internet.

In Broadcast, a source node sends packets and which are delivered to all the nodes in the whole subnet e.g. a LAN. Common methods to achieve broadcast are uncontrolled flooding, controlled flooding and spanning tree broadcast. Broadcast protocols are used at both the application and network layers [10]. Gnutella broadcasts queries for content among peers through application layer broadcast where it uses a form of sequence number based controlled flooding.

In multicast, a subset of network nodes form a group to exchange information among themseleves. There may be one or more sources that transmit packets to all the other nodes in the subset. Internet routers, unless specially provisioned, do not pass multicast traffic by default. The example applications include Live lecture delivery to a set of subscribers, video conferencing, video gaming etc..

1.4 Approaches for Multicast

Both network layer and application layer provide support for multicasting. To describe the different approaches for multicast, we take an example network as shown in figure 1.1 and we show different approaches applied on this network in subsequent figures.

1.4.1 Network Layer Multicast

Initially multicast service was implemented at network layer known as IP multicast. In network layer multicast, the duplication efforts and group membership management is done by routers (at network layer), thus spe-

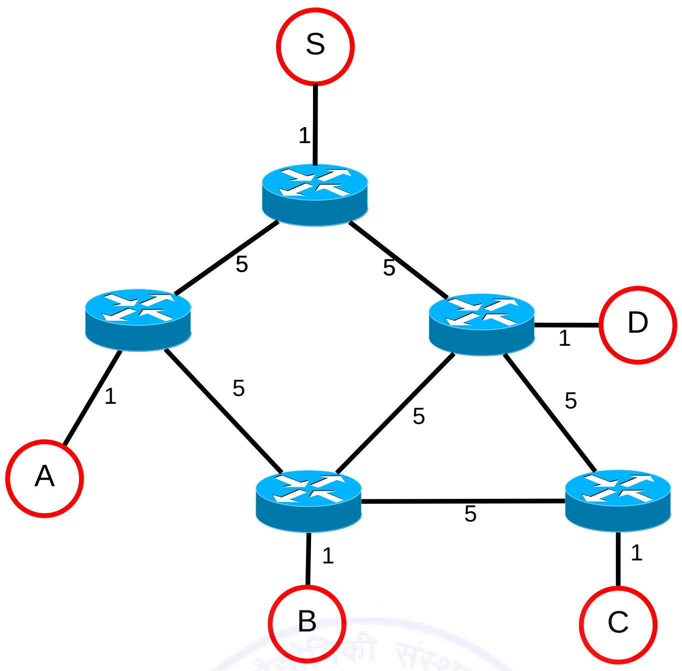

Figure 1.1: Example network

cial multicast enabled routers are required. The multicast enabled routers can decide which sub-networks have members for any multicast group and attempt to minimise the transmission of packets to the parts of the network where there are no active group members. The format of IP multicast packets is identical to that of unicast packets and is distinguished only by the use of a special class of destination address (class D IPv4 address) which denotes a specific multicast group.

1.4.2 Application Layer Multicast (ALM)

In Application layer multicast the duplication of packets and group membership management is done at end hosts participating in the multicast without any support from the network routers. This effort can be done at the source alone or it could be a joint effort of all the hosts participating.

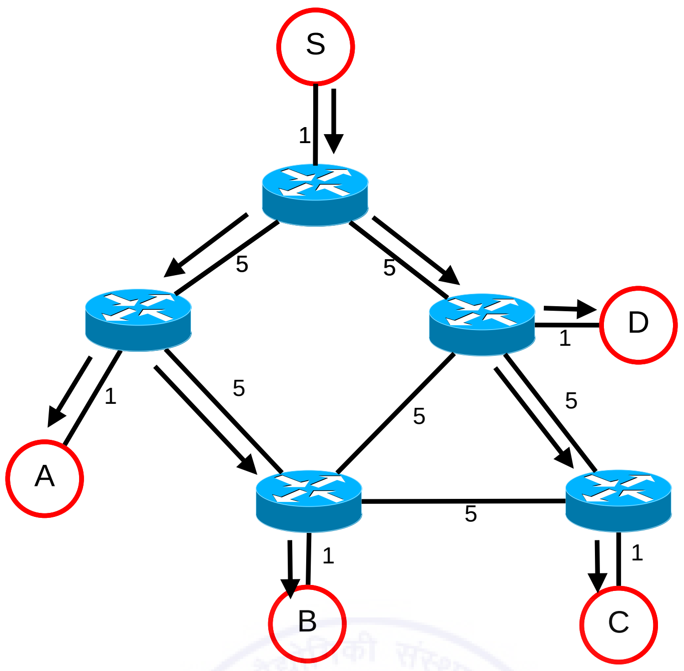

Figure 1.2: IP multicasting

1.4.2.1 Multiple Unicast

The multicast at application layer can be done in multiple unicast way without requiring routers or other participating hosts to make any packet duplication efforts and group management. The source itself explicitly does transmission of the same packet to N targeted members independently. Thus N unicasts are done to give an effect of multicast. This is the most inefficient way of multicasting.

1.4.2.2 Overlay Multicast

In overlay multicast, an overlay of group members is formed and group management and duplication efforts are done jointly by members in order to deliver the data efficiently by utilizing unicast among different pairs of hosts.

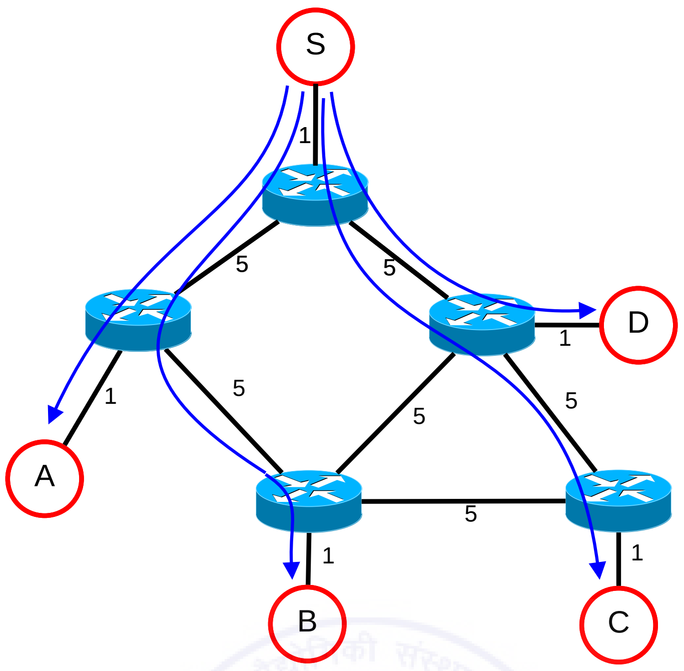

Figure 1.3: Naive unicasting (multiple unicast way)

1.4.3 Network Layer Multicast Vs Overlay Multicast

The network layer multicasting is the most efficient way of multicasting as there is no wastage of network bandwidth. In IP multicasting, decision of forwarding is taken at multicast enabled router, and the packets are forwarded only in the directions where one or more active members are there. A packet never traverses the same link more than once. The major limitation with the IP multicasting is that it requires infrastructural changes and hence can not be deployed at large scale without updating the routers.

The overlay multicast though not as inefficient as multicast by way of multiple unicast, but it is also not as efficient as IP multicasting. However, it is easily deployable in the network as all the multicast related efforts are done by the endhosts and it does not require specialized routers with multicast capability. The participating hosts in the session form an overlay and only

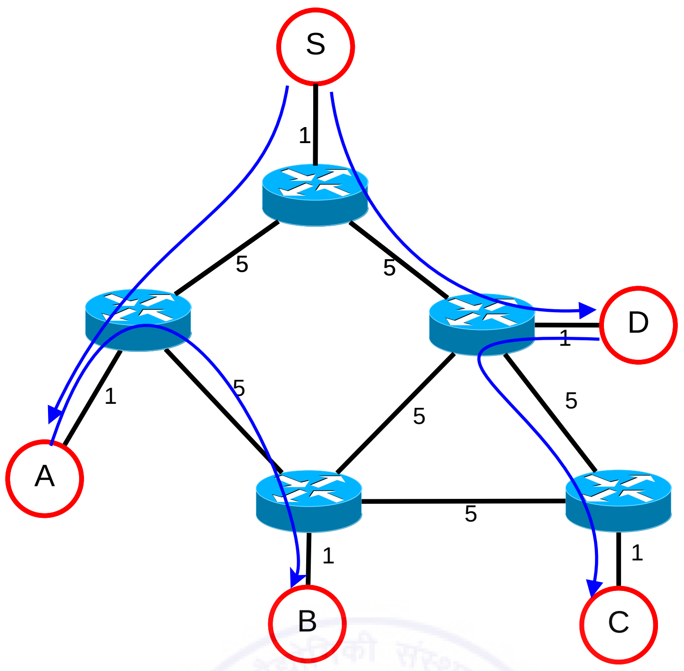

Figure 1.4: Application layer multicasting

using unicast among pairs of hosts data distribution is done among them.

1.5 Overlay Multicast for Live Lecture Delivery and challenges

The advancement in technology has completely changed the teaching and learning scenario. People have found the way to complement the traditional classroom teaching where the delivery resembles to one-size-fits-all, where an advanced learner feels like wasting his time whereas the for the novice it could be too fast. With the explosion in Information and Communication Technology (ICT), it is possible for different universities to increase their reach to any user across the world and to fill the voids of traditional class room teaching. The universities now can put their archived lectures on their website, and any user can see them at his convenience. He can navigate at his own pace (on-demand viewing). These are sometimes supplemented with the discussion groups and bulletin boards where the learners can communicate among themselves. The limitation with these archived lectures put on website is that it is still a one-way communication where a learner can not interact with the instructor.

According to World Internet Usage and Statistics, the internet users have grown from 361 million to 2.8 billion in the first 13 years of this century [11]. With the growing number of internet users along with the popularity of world wide web, we can expect a Live Lecture Delivery Systems (LLDS) to be used for instruction, where a large heterogenous population can be served across the different corners of the world creating a virtual classroom environment. In live lecture delivery, where two-way communication is possible, the learner can immediately seek clarifications by interacting with the instructor. We can also have other live interactive media like shared white board. This LLDS differs from a videoconferencing system in many aspects. While video conferencing is limited to have a much smaller number normally limited to few tens of participants, in LLDS, the participants could be as large as few millions. While video conferencing is done usually among users of a single organisation or within an autonomous system, in LLDS, we do not expect such restriction, and the participants in LLDS may be connected with different (heterogenuous) bandwidth and processing capabilities. Since IP multicast demands infrastructural changes, peer-to-peer overlay multicast is the only hope at such a large scale.

Though there have been many attempts to write an efficient and scalable overlay multicast protocols, and many protocols are available which are designed for specific applications, to the best of our knowledge no protocol is available to cater the needs of a large scale LLDS. Some issues which are specifically important for LLDS are as follows:

1. *Delay as well as Bandwidth sensitive:* The live streaming multicast application is sensitive to both delay as well as bandwidth. The delay as well as jitter should be within a limit for timely and continuous delivery. Excessive delay imposes the requirement of large buffers at each host and jitter creates glitches in video. Also video transmission requires higher bandwidth starting from few hundreds of kbps to few Mbps. Thus *Reliable Routing* is needed to maintain delay and bandwidth within bounds.

2. *Heterogeneous and dynamic environment:* The application is expected to deal with a large scale participation where the users may have different capacity. Also the hosts can join and leave at their will thus giving rise to the problem of *Churn*.

3. *Flash crowd in the beginning:* Since in our application, the information about transmission of live video is preannounced, most of the hosts would like to connect to the stream just before the beginning of session. This creates a problem of handling larger number of nodes joining the session almost at the same time. It is termed as *Flash Crowd* problem.

4. *Uncontrolled Users' Behavior:* In the live multicast application, it is tacitly assumed that the users always cooperate honestly and never cheat others during the transmission, during the formation of overlay, in passing the signaling information and in data streaming to others. The users' behavior, if found adverse to what is assumed, generates new issues collectively known as *Free Riding*.

The present thesis is an attempt to address above mentioned problems. The efforts have been made to find remedial algorithms to combat these issues which are at the very core of LLDS. The focus in the thesis is towards the *Design of a Query Network based Overlay Multicast Network for Live Lecture Delivery to Millions of Users.*

1.6 Thesis Organization

Chapter 1 gives an introduction to the Peer-to-Peer (P2P) networks. The advantages of P2P model and its discovery mechanism are described. The possible approaches for multicast and use of overlay multicast for Live Lecture Delivery is also discussed in this chapter.

Chapter 2 gives a survey of existing application layer multicast (ALM) protocols with their classification. The chapter begins with the definition of performance metrics of ALM. Classification of ALM protocols from different perspectives, and then description of some popular ALM protocols is given. Open issues in ALM are dicussed finally. A comparative chart for some popular ALM protocols is given at the end for the quick reference.

Chapter 3 gives a survey of existing application layer multicast (ALM) protocols with their classification. An analytical discussion of reliability approaches is also presented. Classification of reliability approaches, brief description of major reliability protocols for streaming and the performance metrics for a reliability protocols are discussed in this chapter. A comparative chart for the protocols and reliability approaches is given at the end for the quick reference.

Chapter 4 presents a biconnctivity based approaches for resilience in unstructured overlay multicast. These approaches are then analyzed and compared. Algorithms for data distribution with duplicate paths are also discussed in this chapter.

Chapter 5 gives schemes to address the reliability issues in structured (lookup protocol based) overlay multicast. Dualpath based approaches are presented for streaming to enhance reliability. The three schemes for maintaining dualpaths in data distribution overlay are discussed and compared. The tree optimization algorithm is also included in this chapter as a remedy for excessive latency. The PeerSim based simulation for the best dualpath approach is done and then it is compared with other available reliability schemes. Simulation results verify that the differential delay in two paths and startup delay are well within tolerable limits and with moderate to high failure rates, only a small fraction of nodes get deprived of feed only for a small duration.

Chapter 6 considers the problem of Flash Crowd. Algorithms for faster overlay creation in the situation of flash crowd are presented and analysed. The results prove the efficiency of the algorithm. The distributed and scalable query handling algorithm with cache updation and feed forwarders list maintenance algorithm are at the core of this chapter. Simulation based evaluation is done to verify the effectiveness of the algorithms.

Chapter 7 finally concludes the thesis work. The summary of specific contributions of the thesis and future scope is also presented in this chapter.

Chapter 2

A Review of ALM (Application Layer Multicast) Protocols

Applications such as Internet-TV, e-learning system for large groups, live transmission of sports events often require live multicast for their delivery. There have been many efforts in supporting live media streaming multicast in last two decades. The long surviving traditional method for implementing multicast related functionality has been IP layer based multicasting. However, more than 15 years after the initial proposal, IP multicast has concerns related to scalability, network management, deployment and support for higher layer functionality such as error, flow and congestion control. Most of the networks have not enabled multicasting or it is being provided as a value added service. Application layer multicast (also called Overlay Multicast) emerged as an attractive alternative solution, where multicast-related functionalities are moved to end-hosts. This peer-to-peer technology does not require support from internet routers and network infrastructure and thus is cost effective and easily deployable along with its capability to deal with the problems of scalability, group dynamics and heterogeneity.

Application layer multicast (ALM) builds an overlay topology consisting of end-to-end unicast connections between participating hosts. This overlay performs topology construction and data relaying at the application layer. The shifting of multicast support from routers to end systems has the potential to address most problems associated with non availability of IP multicast. In past few years, numerous algorithms and protocols have been proposed for Application Layer Multicasting. Application layer overlays, however, incur a performance penalty over router level solutions. In ALM, multiple overlay edges may traverse the same physical link and thus create redundant traffic and reduces the efficiency of the network. Also communication between two end systems requires traversing other end systems and thus increases latency. Therefore the major concern is how to route data along the topology efficiently.

The deployment issues in IP multicast, characterization of overlay multicast tree structure, overlay tree construction and maintenance algorithms, example overlay multicast networks, their basic mechanism and relative merits have been surveyed in this chapter. Current trends and direction for the future research have also been given.

2.1 Introduction

The native design of Internet was developed and optimized for one-to-one applications such as reliable file transfer and electronic mail. Its growth however has given birth to the new applications that are inherently one-to-many, such as video-on-demand and live media streaming; or many-to-many, such as video conferencing and multiplayer games. These applications put strain on the available resources and make inefficient use of one-to-one or unicast-only infrastructure. The need for efficient support of one-to-many and many-to-many applications led to the proposal for the implementation of multicasting. A unicast packet has a single source IP address and a single destination IP address. A multicast packet has a single source IP, but it has a multicast destination IP address, also called the group address. The multicast packet is delivered to all those receivers which are members of the group identified by the group address.

Thus an IP Multicast network allows one or more sources to efficiently send data to a group of recipients whereby the source transmits only one copy of the data and the appropriate network nodes efficiently make duplicate copies along the way to each receiver. Since the infrastructural support is essential for IP Multicast, its deployment at large scale has remained an elusive goal. Many issues such as group management, address allocation, authorization and security, Quality of Service (QoS) and scalability, are unaddressed. Internet connections to homes provided by local Internet Service Providers (ISPs) rarely have the ability to be a part of an IP Multicast session.

In response to the serious scalability and deployment concerns with IP multicast, in recent years, application layer multicast (overlay multicast) networks have become an effective alternative to IP multicast for efficient point to multipoint communication across the Internet. Here, end systems implement all multicast related functionality including membership management and packet replication. This shifting of multicast support from routers to end systems has the potential to address most of the problems associated with the IP multicast. However, it introduces duplicate packets on physical links and incurs large end-to-end delays than IP multicast. An overlay can be formed directly among participating end systems or among Multicast Service Nodes (MSNs) distributed in the network, providing multicast services to a set of end-hosts. This later type is called two-tier proxy based infrastructure. Overlay multicast tree construction and maintenance is a major challenge in designing application layer multicast (ALM) protocols. Depending on methods of building data delivery tree, the existing ALM protocols may be classified as mesh-based or tree-based (Mesh and tree protocols, and Direct tree protocols).

In this chapter, we start with a brief discussion of issues related to IP Multicasting that led to the development of ALM. A survey of ALM protocols is then given. Our approach here is to identify properties that are significant specially for live media streaming application, and characterize the protocols architectures accordingly. These properties include application domain, group configuration, routing protocols, and other characteristics that typically lead to trade-offs in design decisions such as mesh-first approach versus tree-first approach (group management), minimum spanning tree or clustering structure (routing), multi-source versus single source (application domain), and many other characteristics. A comparison table highlighting merits and demerits of different ALM protocols against characteristic parameters is given at the end. At the end of the chapter, current trends and direction for future research are presented.

2.2 Application Layer Multicast

2.2.1 Performance Metrics

An overlay network among a set of end hosts can be defined over an underlying physical network where each overlay link between end hosts consists of multiple physical hops including, source end host to its access router, one or more router-to-router hops in the backbone and the last hop between destination end host and its access router. Thus an overlay link may traverse many routers in the underlying physical network. An example overlay network shown in figure 2.1 describes the building of overlay tree over the underlying physical network containing a source and four receivers. An overlay tree edge comprises many physical links and routers in the underlying network.

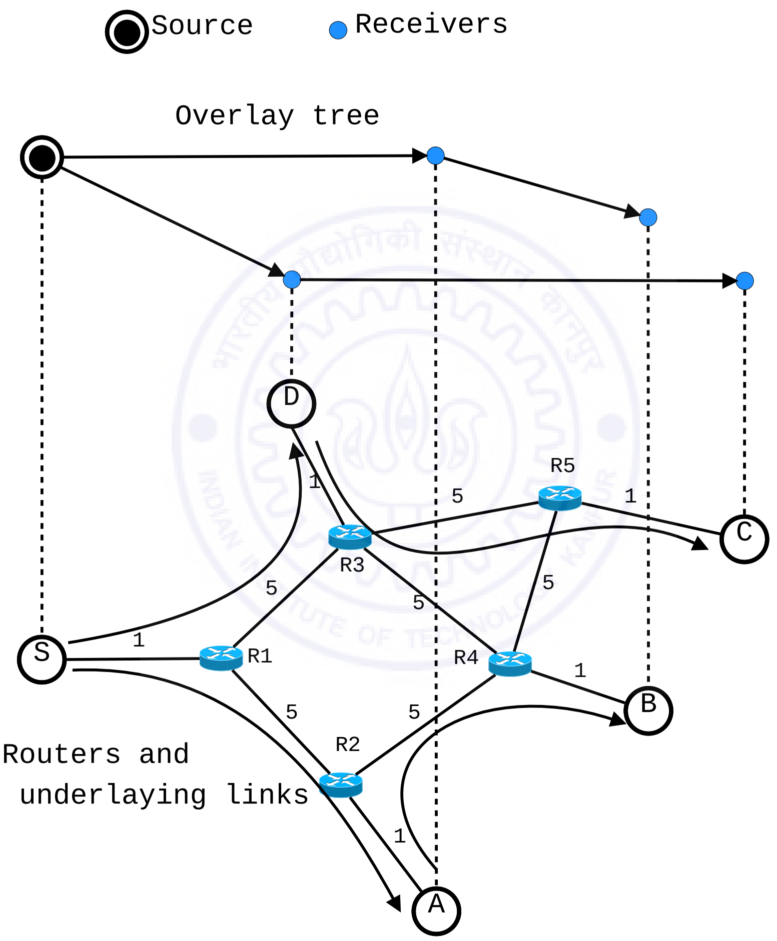

Figure 2.1: Example underlying network with one source and multiple receivers and overlay multicast tree thereof

Some of the parameters that characterize quality of the data path in ALM are following [12].

1. *Link Stress:* The stress of a physical link is defined as the total number of identical copies of a packet it carries. For IP Multicast, where there is no redundancy of such type, the value of link stress is found to be unity.Thus link stress measures how much inefficient an ALM scheme is as compared to IP multicast.

2. *Overlay Cost:* The Overlay cost of an overlay network is defined as the total number of underlying hops traversed by all the overlay links.

3. *Resource Usage:* This metric is defined as the sum of the (delay $*$ stress) over all the links that participate in the data transmissions. This metric gives an idea of network resources consumed in the process of data delivery to all the receivers. Here we have assumption that links with high delays are more costly. Resource usage metric becomes equal to the overlay cost when the delay for each link is unity.

4. *Relative Delay Penalty (RDP):* Assume a source host h_s delivers data to a destination host h_d alongwith many other hosts h_{d_1}, h_{d_2}, ..., h_{d_n} on the way, the Relative Delay Penalty (RDP) is defined as

$$RDP\ (h_s\ to\ h_d) = \frac{latency\ (h_s\ to\ h_d)}{delay\ (h_s\ to\ h_d)} \tag{2.1}$$

where delay(h_s to h_d) is the direct unicast delay between the source and destination hosts, and

$$latency\ (h_s\, to\, h_d) = delay\ (h_s\, to\, h_{d_1}) + \sum_{i=1}^{n-2} delay\ (h_{d_i}, h_{d_{i+1}}) + delay\ (h_{d_{n-1}}\ to\ h_d) \tag{2.2}$$

Considering a single source transmitting data to many receivers, the RDP for different receivers may be different, therefore the average of RDPs of all receivers in a particular overlay is calculated to characterize the overlay with better significance.

5. *Stretch for a member:* Like RDP, Stretch also compares an ALM scheme with IP multicast.The only difference is that stretch denotes the relative number of hops instead of the relative latency used in RDP.

$$stretch = \frac{hops\ (h_s\ to\ h_d)\ on\ the\ overlay}{rrh(h_s, h_d)\ in\ the\ backbone\ network\ +\ 2}$$
$$= \frac{(rrh\ (h_s\ to\ h_d) +\ 2) + \sum_{i=1}^{n-2}(rrh\ (h_{d_i}, h_{d_{n-2}}) + 2) + (rrh\ (h_{d_{n-1}}\ to\ h_d) + 2)}{rrh\ (h_s, h_d)\ in\ the\ backbone\ network + 2} \tag{2.3}$$

Here, rrh (h_s, h_d) denotes the total number of router-to-router hops betwen the two hosts h_s and h_d.

There exists a tradeoff between the latency and the stress (bandwidth) metrics.

6. *Losses after Failures:* This metric counts the average number of packet losses after an ungraceful failure of a single node and measures the robustness of ALM protocol. It highlights robustness in the occurrence of unpredicted events.

7. *Time to First Packet:* It defines the time required for a new member to start receiving a data flow when joining an on-going session.

8. *Control Overhead:* To maintain the overlay topology, the participating overlay nodes exchange control information (e.g. refresh messages in

NARADA) among themselves which constitute the control overhead at routers, links and participating members. While some overhead is necessary to keep the topology intact, the amount of control overhead decides the scalability of a protocol.

2.3 Classification of ALM proposals

2.3.1 Architecture: Peer-to-peer or Proxy-based

In pure P2P (also called single tier) distributed architecture, the overlay is built across end-users with all functionalities of group management and data replication implemented at end hosts. ALM networks with P2P architecture are easily deployable and inherently scalable. Despite being heterogeneous with regard to their physical connectivity, computing power, they can be individually harnessed according to their capabilities. P2P systems are redundant in the sense that a single failure does not affect the network substantially. Narada, NICE and Yoid are representative protocols under distributed architecture.

In proxy based (two-tier) architecture; the overlay is created between dedicated proxies to improve the performance. Being dedicated, homogeneous and better provisioned than individual hosts, the proxies based parchitecture is more reliable and robust to failure. Usually Proxies are deployed by ISPs and are stable than an individual host. They are more intelligent than end-hosts as they can provide value-added services such as being pre-configured with application specific components to make them application aware. Proxies can be positioned at strategic positions such as co-locating with IP routers or at hotspots to provide more efficient services. The problems with proxy based architecture are with regard to their acceptance and deployment. Proxies are static and not responsive with changing network conditions. OMNI, Overcast and Scattercast are some examples of proxy-based ALM protocols.

2.3.2 Control: Centralized or Distributed

A centralized approach to the overlay tree creation problem maintains a central controlling entity responsible for the group management, overlay computation and topology optimization. The controller maintains group information, handles group membership, collects the measurements from all the members, computes optimal distribution tree and disseminates the routing tables to all the members. Routing becomes an easy task with such an approach and the problems of tree partitioning and routing loops are alleviated. For small groups, the centralized approach is a good choice, however such protocols are not scalable and susceptible to a central point of failure. In the distributed approach, the responsibilities of group membership and overlay topology computation is distributed among peers. The distributed approach is more scalable and robust, however, it has more overhead and may not converge to optimal and efficient solution as fast as in a centralized approach.

2.3.3 Tree Construction: Mesh First, Tree First or Implicit

Any ALM protocol, requires efficient data distribution among peers while ensuring robustness of the overlay topology. This leads to tree like data topology alongwith a highly connected mesh-like control topology. Depending on sequence of construction of these topologies, ALM protocols are classified as Mesh-first, Tree-first or Implicit type [13].

In Mesh-first approach, a mesh control topology is first built among peers with multiple paths among them. A source specific tree rooted at any member can then be created using Reverse Path Forwarding (RPF) construction of data topology. Example protocol is Narada. Mesh based approaches have improved delay performance but needs a dynamic routing mechanism in order to achieve loop free data delivery which limits the scalability.

In Tree-first approach, a shared data distribution tree is built first, followed by the control connections between nodes in the tree. Example protocols are Yoid and HMTP.Each arriving member searches for its parent on its own with the help of a bootstrapping node called Rendezvous Point (RP). RP either gives a list of already connected members (as in Yoid) or indicates the root of the tree (as in HMTP). In the first case, one of the members in the supplied list becomes parent and in the later case; the parent search starts from the rootcoming gradually downward in the tree till an appropriate parent is found. An already connected member can become parent if it has available degree for new nodes. Once placed in the tree, each member in the tree discovers a few other members of the group that are not the neighbours on the overlay tree and maintains additional control links to these members.

In Implicit approach, the control and data paths are defined simultaneously. A control topology with some specific properties is created, and data delivery paths are implicitly defined on this control topology by some packet forwarding rule. The protocols under this approach have been designed to scale for large groups. Example protocols are NICE, CAN based multicast and Scribe. NICE creates a hierarchy of clusters, i.e. sets of nodes "close" to each other. An arriving node recursively cross this hierarchy to find the appropriate cluster. Tree based solutions are simple and scalable but more fragile than mesh based solutions. Also there is longer end-to-end delay as all data must pass through the tree's root.

2.3.4 Design Objective: Efficiency or Scalability

Another way of classifying ALM protocols could be based on data delivery mechanisms used in the protocol [14]. The data delivery can be based on a single shared tree, source rooted trees or multiple shared trees. It has been observed that as we increase the number of trees in data distribution overlay, the delay decreases and robustness increases, but the overhead increases and hence the efficiency decreases.

2.3.4.1 Source Based Trees

Narada protocol uses source based trees. This approach incurs lower delay and is suitable and efficient for a small group. However as the group size increases, control overheads increase very rapidly with the number of trees hence not efficient for large groups due to high protocol overhead.

2.3.4.2 Single Shared Tree

YOID protocol uses single shared tree. It can support a fairly large group. But since all the members are in one single tree and each node has bounded out degree, due to limited bandwidth and processing ability of end user terminals, the depth of the tree will be high and hence the latency will be high. Such approach is suitable and scalable for non-interactive applications e.g. Video-on-Demand but is not efficient.

2.3.4.3 Multiple Shared Trees

Tan *et. al.* [14] proposed an approach in which more than one multicast trees are there but far less than the total number of sources. This approach is extremely useful when both the number of senders and the group sizes are very large. With this approach, on the one hand, total number of trees is small and hence overhead is not too large; also, tree depth is not large resulting in not so large latency. For n nodes and s sources, m trees can be formed, where $1 < m \ll s$. Thus the protocol cost is only m times higher while the delay is within control. This approach is useful for multi-sender applications such as live lecture delivery, video conferencing and multi-party network gaming. Through simulations, it has been shown [14] that the proposed multiple shared trees approach offers a well balanced solutions to multi-source applications.

2.4 Example Application Layer Multicast Protocols

Under this section we describe the representative protocols from each class as discussed above.

2.4.1 Narada

Narada [15] is a representative protocol for mesh first approach in which source based trees are formed. The Narada protocol was one of the first application layer multicast protocols that demonstrated the feasibility of implementing multicast functionality at the application-layer. The protocol proposed a peer-to-peer, self organizing, fully distributed, mesh based topology with source based trees for data distribution. The target application for this protocol is audio and video conferencing with sessions that lasts for tens of minutes. It can support small groups (tens of participants), where any member can be the source (single source at any point in time) transmitting data at a fixed rate.

Meshes allow optimized tree construction for an individual source. Group management functions can be implemented at the mesh level rather than in each individual tree.The protocol constructs a well connected mesh first and then runs a DVMRP-like protocol to construct shortest path spanning tree for the data delivery. The overlay is optimized for both latency and bandwidth, but bandwidth is given priority over latency. If there are multiple paths with the same bandwidth, shortest one is chosen. Network path characteristic information (i.e. bandwidth and latency) is obtained by passive monitoring as well as active measurements. The estimates of bandwidth and latency are further improved by exponential smoothing algorithm.

2.4.1.1 Group Management Component

To ensure that overlay remains connected even with dynamic joins and failures, every member maintains a list of all other members in the group. Each member periodically generates a monotonically increasing sequence number which is disseminated in the refresh packets from source to neighbors and then via routing table exchanges over the mesh.

Table 2.1 describes the database at member i. Each member i maintains the following information for every other member (say) k in the group: member address k, last sequence number s_{ki}, that i knows k has issued , and the local time at the node i when it first received information that k has issued s_{ki}. Each member periodically exchanges its knowledge of group membership with its neighbors in the mesh with increasing sequence numbers. A message from member i to member j contains first two columns of the table 2.1 along with the last generated sequence number. On receiving a message from a neighbor j, member i updates its table. If no update is received about the member k for certain threshold time T_m ; the node i assumes that k is either dead or has partitioned away from i.

Member Address	**last sequence number that i knows that member has issued**	**local time at *i* when *i* first received the information that member issued this sequence number**
$\boldsymbol{k}$	s_{ki}	t_i *at the arrival of* s_{ki}
$\boldsymbol{l}$	s_{li}	t_i *at the arrival of* s_{li}
$\boldsymbol{m}$	s_{mi}	t_i *at the arrival of* s_{mi}
–	—–	——————

Table 2.1: Table maintained at member i to keep track of every other member in the group which are known to it. All known members are listed in the table

Member Join: It is assumed that a member will be able to get a list of group members (with at least one active member in the list) after arriving in the network through bootstrap mechanism from special designated hosts, also called the Rendezvous Points (RPs). The request is sent to all the listed members, and the responses are received from some of them. After joining, new member starts exchanging refresh message with the neighbors.

Leave/Failure: The protocol assumes fail-stop failure model. In case of a node leaves, it notifies its neighbors, and this information is propagated along the mesh. In case of abrupt failure, failure is detected locally (neighbors send probe messages, if no response is received, the node is assumed to be dead), and propagated to others. Dead members' information can be flushed after sufficient amount of time.

Repairing Mesh Partition: When the network get partitioned, members on each side of the partition stop receiving refresh messages from members on other side. Each member maintains a queue of such members. It runs a scheduling algorithm that periodically and probabilistically deletes a member from the head of the queue. The deleted member is probed and it is either determined to be dead, or a link is added to it if it is found to be reachable. Scheduling algorithm is adjusted so that no entry remains for more than a bounded period of time. Probability value is chosen so that in spite of several members simultaneously attempting to repair partition, only a small number of links are added at a time. Figure 2.2 considers an example mesh topology of 7 nodes named as A, B, C, D, E, F, and G. The data distribution overlay tree over this topology, that covers all the nodes without any loop is shown with dark lines. Figure 2.3 considers the mesh partitioning due to failure of node E. Two new links (D-F and D-G) are added to repair this partition as per algorithm explained above.

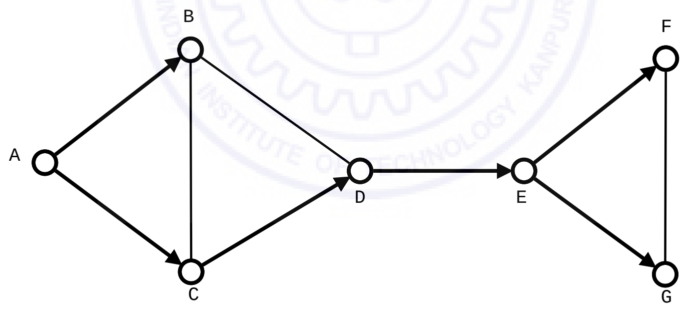

Figure 2.2: An example mesh topology

2.4.1.2 Overlay Optimization Component

This component ensures that the overlay quality remains good for all the time. An active measurements and passive monitoring of performance is done and links are added/dropped accordingly.

Addition of links: A member i computes the utility gain if a link is added connecting it to member j based on the number of members to which, link to j improves the performance for i and the amount of improvement. This way, periodically, members evaluate the utility of adding a link to a random member which is not a neighbor and evaluates the utility of adding a link to this member. A link is added to this random member if utility gain exceeds a given threshold.

Dropping of links: The cost $Cost_{ij}$ of a link between i and j in i's perception is the number of group members for which i uses j as next hop. Periodically, a member computes the consensus cost of its links to every neighbor.Consensus cost of a link is defined as the **max** ($Cost_{ij}$,$Cost_{ji}$). Every member periodically computes the consensus cost of its links to every neighbor. Link with lowest consensus cost is dropped if its consensus cost falls below a certain threshold. Dropping will not partition the network because as long as the drop threshold is lower than half of the group size, at least one edge will always be maintained from the node. .

Figure 2.4 shows an instance of new node joining in the mesh. A new member H joins the mesh and sets up links with two randomly chosen neighbors in the mesh. Figure 2.5 shows how the mesh topology optimizes itself after new node joining by dropping an existing link of low utility and adding some new links of high utility and thus transforming the mesh in to a new final form (figure 2.6).

Data delivery: Each member maintains the routing cost and the path to every other member. Routing updates between neighbors contain both the cost to the destination and the path that leads to such a cost. Per source trees are constructed from the reverse shortest path between each recipient and the source as done in DVMRP (Distance vector Multicast Routing Protocol). TFRC (TCP friendly rate control) is used as the underlying transport protocol on each overlay link.

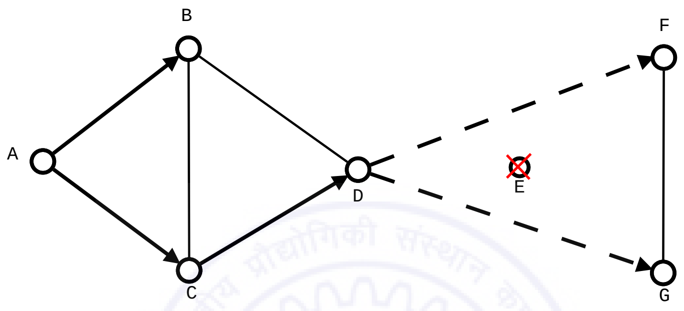

Figure 2.3: Mesh partitions as node E fails

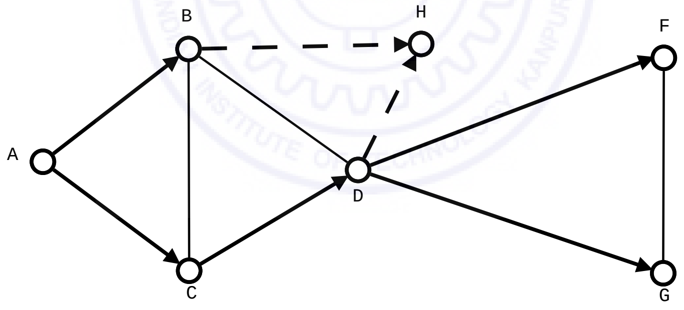

Figure 2.4: New node joining in example mesh topology of figure 2.2

2.4.1.3 Performance Evaluation

Each member in the group keeps state of all other members that are part of the group. This information is also periodically refreshed. Distribution

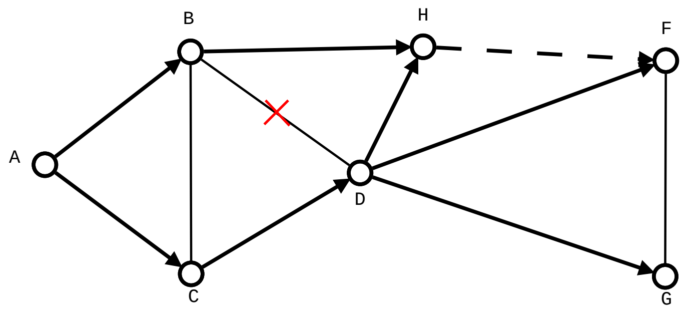

Figure 2.5: Topology optimization through add/drop of links

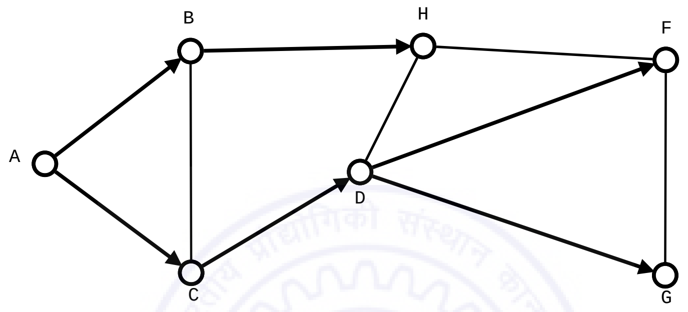

Figure 2.6: Final mesh topology after add/drop of links

of such state information about each member to all other members leads to relatively high with order ($O(N^2)$) aggregate control overhead, where N is the group size. The Narada protocol was designed for a small group size and traded off high control overheads for greater robustness where every member maintains a list of all other members in the group. Bandwidth performance is comparable to IP multicast. Mean receiver latencies are 1.3 to 1.5 times than that in IP multicast.

2.4.2 NICE

NICE (recursive acronym for NICE is the Internet Cooperative Environment) protocol [16] is an efficient, scalable and distributed tree building protocol which does not require any underlying topology information. It targets low bandwidth, loss tolerant realtime data stream applications with large receivers sets e.g. News and sports ticker services. It arranges group members into sets, forming a hierarchical control topology. As new members join and existing members leave the group, the protocol creates and maintains the hierarchy as its basic operation. The hierarchy implicitly defines the multicast overlay data paths and is crucial for scalability of this protocol to large groups. The members at the bottom of the hierarchy maintain state of constant number of other members at the same level, while the members at the top maintain such state for about O(logN) other members.

Members are assigned to different layers of the hierarchy. Hosts in each layer are partitioned into a set of clusters. Members that are close are mapped to the same clusters. Each cluster is of size between k and (*3k-1*), where k is a constant. These bounds can be justified as following. Since members in the group keep joining and leaving, there is a provision of merge and split operaion among clusters in the protocol. With, split/merge opeartions permitted the most elememntary choice for the upper and lower bound is k and *2k-1*. The problem with this choice is that a series of leaving and joining processes by members causes split and merge operation in the same sequence resulting in oscillations. For example a cluster with an initial size *2k-1* splits into two when a new member joins, resulting in to two clusters of size k each. Further if a member leaves in any of these newly formed cluster makes that cluster size less than lower bound and invokes again a merging of two clusters. The next better choice for the size range is [k, *3k-1*].

All hosts together form layer L_0. The cluster leader is the graph theoretic centre of the cluster. The cluster leaders of all the clusters in layer L_i join layer L_{i+1}. There are at most log_kN layers and the highest layer has only a single member. This hierarchical overlay structure is used to distribute control messages and to form data delivery paths.

2.4.2.1 Protocol Description

The protocol assumes the existence of a bootstrapping node called as RP (Rendezvous Point). It is a special host which acts as the leader of the single cluster in the highest layer of hierarchy. All the members will know about it a-priori. RP interacts with other members on control path but is bypassed while forming data delivery path.

Host Joining: A Joining host (JH) contacts the RP which responds with the list of hosts that are present in the highest layer. The JH contacts all the members in the highest layer to find the closest member. This closest member informs JH about its other members in the lower layer. Then the closest member is identified in the lower layer. This is done until JH finds its L_0 cluster. Joining Latency happens to be *O(log N)* RTTs, and the message overhead complexity is *O(k log N)* query response pairs.

The joining delay is defined as the duration from the instant when joining request was sent to the instant when the JH receives its first data packet. To reduce the joining delay to single RTT, JH on data path temporarily peers with the leader of the cluster of the current layer it is querying.

Cluster Maintenance/Refinement: Each member host (H) of a cluster periodically sends a message, called Heart Beat Message (HBM) to each of its cluster peers. Cluster leader includes complete updated cluster membership in HBM to all other members. This allows other Cluster Members to know about new joining hosts.

Split/Merge: Each cluster leader periodically checks the size of its cluster and does appropriate split or merge. If size exceeds the upperbound *(3k-1)*, the leader of the cluster splits them into two equal sized clusters, such that maximum of the radii among the two clusters is minimized and then the leadership is transferred to the new leaders (centres of the two partitions) through *LeaderTransfer* message. In case size of a cluster in layer L_i falls below the lower bound (k), its cluster leader selects its closest layer L_{i+1} peer and hands over all its members (including itself) to this L_{i+1} peer, which is also a layer L_i leader by sending a *ClusterMergeRequest* to it.

Inaccurate attachment: Supercluster for a cluster member (in layer L_i) is defined as the cluster of leaders of layer L_i. Each member in any layer (say at layer i), periodically probes all members in its supercluster (members of layer *i+1*) to identify closest member in the supercluster (say J). If presently J is not its cluster leader, it leaves present L_i cluster and joins L_i cluster whose leader is *J*. This way, the heirarchical topology keeps refining.

Leave/Failure: While leaving gracefully, the departing host (H) sends a *Remove* message to all clusters of which it is a member. In failure case, abrupt departure is detected through nonreceipt of *HeartBeat* message. If the departing host was the leader of the cluster, new leader (centre among the remaining members) is selected.

Control and Data topology: Control path peers of a node include all the nodes which belong to the same cluster in which this node lies in all the layers to which this node belongs to. Each member of a cluster exchanges *Refresh* message with all the remaining members of the cluster.

Source delivers first to all the members of the cluster to which it belongs to. Whenever a member receives data, it sends it further to all its cluster members. Thus, given a source; data delivery topology is a source specific tree.

2.4.3 Application-Level Multicast Infrastructure (ALMI)

ALMI [17] consists of a session controller and multiple session members. ALMI provides a centralized solution for multicast groups of relatively small size (several tens) with many-to-many service mode.

Session controller (centralized entity) creates a minimum spanning tree (MST) which consists of unicast connections between end hosts. Latency between members is used as the link cost of the MST. Session controller ensures connectivity of MST when members join/leave the session. It is also responsible for periodic tree refinement based on the measuremens received from session members. Session data is disseminated along the MST, while control messages are unicast between each member and the controller. The controller receives updates from each member and periodically re-computes the MST. Routing information of the MST is then communicated to all the members. Ideally, since the MST is centrally computed, it will be loop-free. However, due to the losses and delays in obtaining updates from members and disseminating the different versions of MST to members, loops and partitions may occur. To check this, version number is assigned whenever MST is refined before spreading it to all the members. Members maintain a cache of the different versions of the routing tables.

Whenever a member (source) generates a packet, it includes latest tree number in the header. Receiving node, after receiving this packet, checks the tree version in its cache. If any of tree versions stored in the cache matches with that in the received packet, packet is forwarded with that tree version; otherwise discarded. Packets with older tree version (not stored in the cache) are discarded. If a packet with higher tree version is received, a copy of new version of tree is obtained from session controller and then the packet is forwarded accordingly.

ALMI multicast trees have been shown to be close to source-rooted multicast trees in efficiency with low performance tradeoff though with higher control overheads O(N), due to the maintenance of the different tree versions.

2.4.4 HMTP and OMTP

Both HMTP [1] and its upgraded form OMTP [2] are representative protocol for tree first approach in which single shared tree is formed.

2.4.4.1 HMTP (Host Multicast Tree Protocol)

Join Procedure: The joining member first discovers the root of the tree with the help of a bootstrapping node called Renzdevous Point (RP). the search for parent starts from the root. Lower levels of tree are explored gradually till a node with free degree is found. HMTP iteratively compares the RTT between the joining nodes and existing nodes in the tree network starting from the root. A node with minimal RTT is chosen as parent by the joining node. An example of node joining in HMTP is shown in figure 2.7.
Member leave and tree repair procedure:

Member hosts periodically exchange REFRESH and PATH packets. In case a Host leaves Gracefully, it notifies its parent and children. In case a Host Crashes, repeatedly missing REFRESH and PATH packets are used to identify the missing node and thus repair is initiated.
Limitations: HMTP suffers from the following problems.

1. As tree grows, number of queries and RTT measurements increases with the depth of tree. Thus for sparse large groups, very high join latency is expected.

2. It also has uneven and unfair processing load distribution which is resolved by Foster children concept.

3. Flash Crowd Problem, for which a solution is to make each host wait for a random delay before restarting the join procedure.

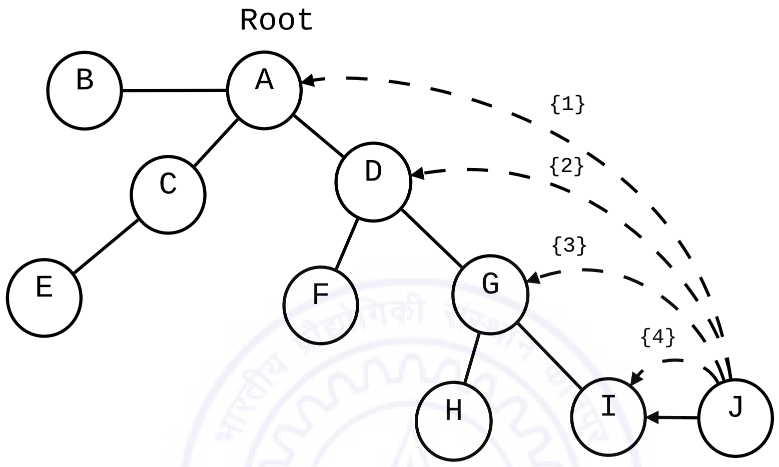

Figure 2.7: Join procedure in HMTP [1]

2.4.4.2 OMTP (Overlay Multicast Tree Protocol)

In Overlay Multicast Tree Protocol (OMTP), by leveraging on the IP hierarchical addressing locality, the formation of Overlay Multicast tree is made faster, and efficiency of tree maintenance is enhanced. Both the bandwidth availability and round-trip-time (RTT) are taken into consideration when a newcomer selects its parent node.

Join algorithm: To make the root of the tree less affected from flash crowds and to distribute the load, a dedicated always-up host, called rendezvous point (RP) is used. All newcomers first contact RP, RP redirects them to different selected member host in the tree. RP identifies the nearest network cluster by matching IP address of newcomer with the prefixes of existing hosts and then using greedy join algorithm is used to locate the nearest peer host. The host with the longest prefix match is first set as parent initially and then children list of this parent is obtained. This list never include the address of such children with all their out degree exhausted. Finally the children with smallest RTT becomes the parent. Thus both RTT and bandwidth availability are taken into consideration in the algorithm. An example of node joining in OMTP is shown in figure 2.8.

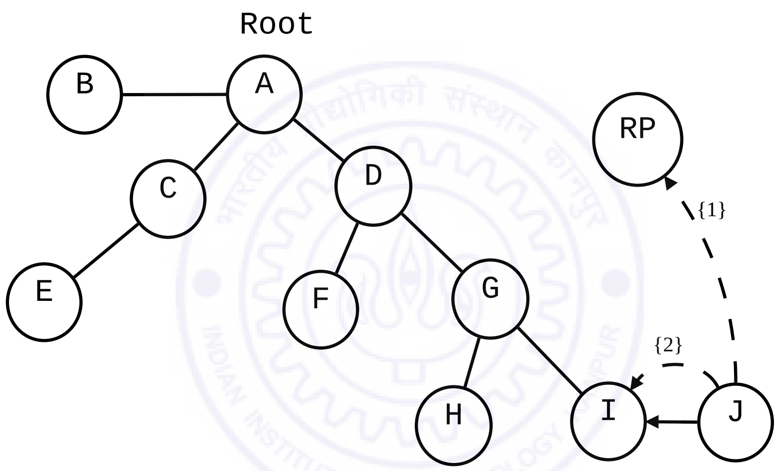

Figure 2.8: Join procedure in OMTP [2]

Tree maintenance algorithm:

Every child periodically sends an UPDATE message (heart beat packet) containing its IP addresses, prefix, root path, vacancy status, and prefixes of its descendents to its parent. Thus children list is updated periodically at each node. Once a parent receives an UPDATE message sent by its children, it sends back a PATH message to its children containing its own root path. The parent compares the children‘s IP prefix with its own. If these are found different, the children's IP prefix is appended into its own descendants' prefix table which will be eventually sent to RP for longest prefix matching purposes. The entries in prefix table received by the RP are therefore unique and represent the nodes in upper hierarchy, i.e. near the root.

The root sends UPDATE message to the RP to make it aware of current root and the IP prefix set of all the childeren in the tree.

Member leave and Tree repair algorithm:

If a member leaves gracefully, it will broadcast the address of one of its children chosen randomly to its other children. The elected child will replace the leaving node. If a member leaves suddenly, each child picks up a potential parent with which it has the second shortest RTT, rather than all of them choosing the same grand parent. A member with second shortest RTT is unlikely to be the same parental candidate of another child.

The join latency in OMTP is reduced by as large as 50 % as compared with HMTP.

2.5 Open Issues

2.5.1 Two Conflicting Design Goals

Another open issue in ALM is balancing the following two conflicting design goals:

1. Minimizing the length of the paths (usually in terms of hops) to the individual destinations

2. Minimizing the total number of hops to forward the packet to all the destinations

The minimum spanning tree (MST) and the shortest path tree (SPT) are two well-known data distribution methods in ALM. The MST optimizes the resource usage of the multicast tree but the pair-wise paths may not be optimal and can cause large end-to-end delays. Hence it is suitable for non-interactive data dissemination when end-to-end delays are not an issue. In SPT, the distribution tree will consist such that path from a node to source in tree is same as that used by a unicast connectivity from the node to source. It is optimal from the source to the receiver in terms of end-to-end delay but it require more network resources. Scaleable ALM systems usually use a different approach by creating hierarchical distribution tree. It naturally uses clustering. The advantage of a hierarchical clustering is the reduction in control overhead as nodes keep states only for all the nodes in its cluster and for very few other nodes. It allows faster joining and efficient group management at the cost of sub-optimal tree.

2.5.1.1 Adapted Routing Algorithm in the Overlay Multicast

Wang *et. al.* [18] developed an adapted routing algorithm for overlay networks that balances delay and bandwidth consumption by integrating the algorithms which minimizes the delay and bandwidth consumption.

Figure 2.9 shows a sample overlay multicast network with each of its link having some delay and bandwidth. Consider that figures 2.10 and 2.11 describe two different trees resulted from two different routing algorithms applied. Figure 2.10 shows the shortest path tree (SPT) that minimizes delay while in figure 2.11 shows the minimum spanning tree (MST) that minimizes bandwidth consumption. In the figure 2.12, both delay and bandwidth consumption have been considered by applying proposed adapted algorithm [18] to optimize the trade-off between the two QoS criteria. Optimal Balance of

Delay and Bandwidth consumption (OBDB) is formulated as follows.

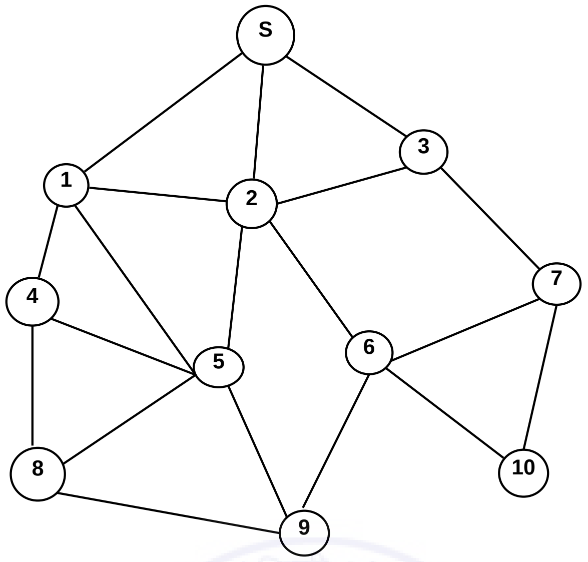

Figure 2.9: An Example Multicast Network

$$OBDB = (1 - \alpha)D + \alpha B \tag{2.4}$$

Where D is the minimum delay criteria and B is the minimum bandwidth consumption criteria respectively to SPT and MST. It is known that each one lead to give excellent performance in one criteria but lead to bad performance in another criteria. Equation 2.4 gives a theoretical bound for balancing D and B, where α is a weight indicating the relative performance of delay and bandwidth.

Wang's adapted algorithm generates a tree that balances delay and bandwidth consumption requirements. The algorithm generates a delay that is only a few percent higher than that of SPT and a bandwidth consumption that is a few percent larger than than that of MST.

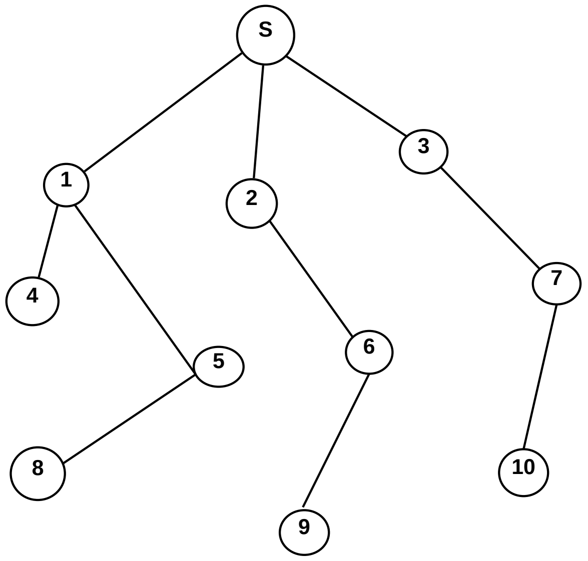

Figure 2.10: Shotrest Path Tree (SPT) resulted from the mesh shown in figure 2.9

2.5.2 Tree Refinement

An open issue for all ALM protocol is that of tree refinement i.e. the reorganization or shuffling of the nodes in the tree. This is usually done to enhance the system performance. In ALM, the quality of the path between any pair of members is same as that of the unicast path between them. Typically a lower diameter tree performs better than a higher diameter tree. Hence, refinement is a way to improve the quality of the ALM structure once it has already been constructed, by constantly reducing the tree diameter. If a node with zero out-degree joins a multicast session, the tree can not be extended beyond this node. It ultimately increases the tree diameter. To handle such situations, incremental refinement is the solution. But it is an expensive operation and thus should be applied as sparingly as possible. This is because the refinement protocols require too much information to perform. Research to find efficient mechanisms to determine whether or not refinement should

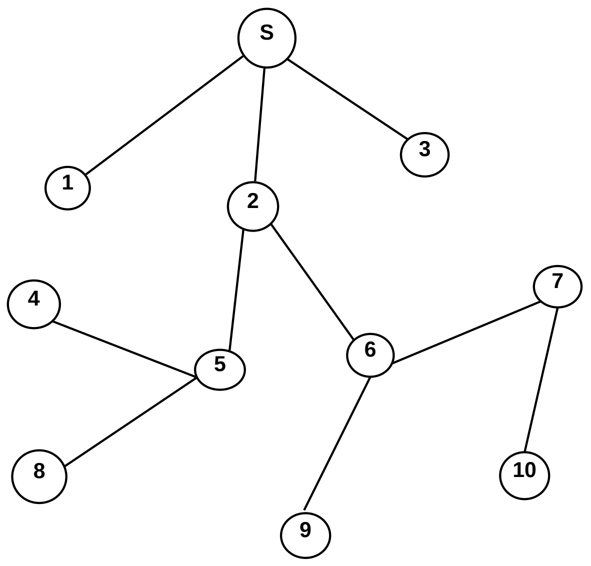

Figure 2.11: Minimum Spanning Tree (MST) resulted from the mesh shown in figure 2.9

be triggered at a particular node is needed. How much the refinement improves the performance of the system, say in terms of average latency or any other parameter is worth investigation. The protocol should also be aware of the transient period of the refinement when it actually takes place. It should be explored whether it affects its dependent nodes and by how much. The protocols must also consider the churn effect due to tree refinement as it can make the system inconsistent intermittently. For example, OMNI [19] uses local transformations (child promote, parent-child swap, iso-level-2 transfer, aniso-level-1-2 swap) and probabilistic transformations (simulated annealing) to refine its structure. As it is an expensive operation and requires extra care, frequent refinement may adversely affects the system performance. Most of the ALM protocols strategically and infrequently apply refinement operation.

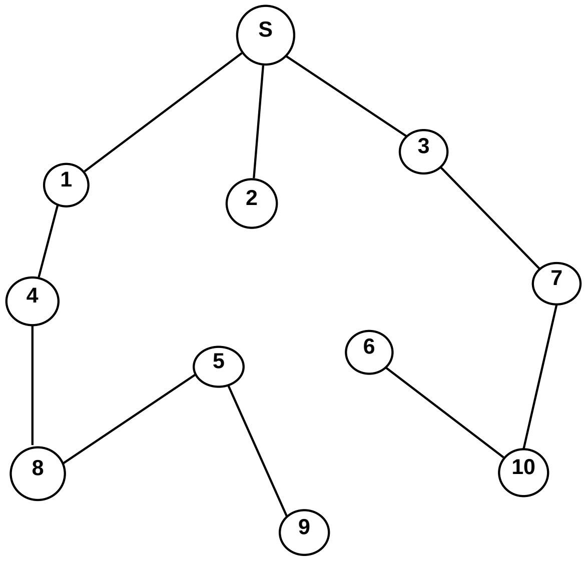

Figure 2.12: Tree obtained from the mesh as shown in figure 2.9 as Adapted Routing Algorithm is applied

2.5.3 Reliability

Due to the inherent dynamic nature (undefined join and leave pattern of peers) of P2P networks, reliability has always been one of the major concerns in all large scale application layer multicast solutions suggested so far ([20], [21]). Any ALM protocol leverages on the capability of end hosts to replicate and forward the data in the network, while these are not as stable as routers in IP multicast. Most of the ALM protocols build a tree based data distribution topology for efficient use of network resources, but in this topology a single node's departure disrupts the streaming to all the peers located in downstream segment. The protocol proponents either introduce reliability in the basic topology building components itself e.g. in Narada [15] or run a separate reliability enhancing protocol *e.g.* PRM over NICE [22].

ALM reliability enhancement approaches are broadly classified as proactive and reactive. In a proactive approach, redundant packets are sent along with the data packets which can be used to reconstruct the original data in case some of the data packets are lost. Some of the proactive approaches are Forward Error Correction (FEC) [23], Digital Fountain [24], Network Coding and layered coding scheme e.g. Multiple Description Coding (MDC) with multipath transmission, and Kunichika's approach [25].

In a reactive approach, lost packets are retransmitted after the receiver requests for the lost packets. Probabilistic resilient multicast (PRM) includes both proactive and reactive components.

In FEC-(d, r), [23] [26] source takes a set of d data packets and encodes them into a set of $d + r$ packets and sends them. A receiver can recover data packets if it receives any d of the $d + r$ encoded packets. Overhead of the scheme is r/d and resilience increases with the overhead. With 100 % overhead (i.e. $r = d$), performance improves with higher values of d (or r) while delivery latency increases. FEC based approaches can recover from network losses. However they alone are not sufficient when overlays are used. Overlay nodes are the processes on regular end hosts and are more prone to failures than the network routers.

In Kunichika's approach [25], all the nodes always keep one free out-degree by force, to accommodate a deprived node in case of a failure. Any end hosts with an out-degree n caters only to $n - 1$ children. Redundant structure of tree avoids exhaustive search of a backup parent and simplifies backup route calculations. Layers to which the backup route calculation is applied are limited at worst to the grandchild layer. The limitation of this approach is that it needs to maintain redundant degrees at each node permanently.

In probabilistic resilient multicast (PRM) enhanced NICE (Nice is the Internet Cooperative Environment) protocol, Banerjee *et. al.* [22] introduced multicast data recovery scheme with two components. A proactive component called randomized forwarding in which each overlay node chooses a constant number of other overlay nodes uniformly at random and forward data to each one of them with a low probability, and a reactive component called triggered NAKs to handle data losses due to link errors and network congestion.

Reliability approach suggested in [27] is based on dynamic mapping of nodes in the tree based on their relative stability. The proposal is based on the fact that the participating users' lifetime follows a Pareto (skewed, heavy-tailed) distribution [28]. The effect of this distribution is twofold: (i) peer's expected remaining lifetime becomes directly proportional to current age and (ii) a small fraction of peers dominate the majority of the lifetime, i.e., only few peers remain alive throughout the whole session, and most peers are short-lived. The algorithm in [27], organizes peers into a logical hierarchy based on their relative stability. We can conclude that full reliability is a difficult goal for large scale streaming networks.

2.5.4 Scalability

Scalability is a major issue when the application at hand needs to create a large heterogeneous network at global level. Many schemes which perform considerably well for small network, fail when network size grows. The first ALM protocol Narada [15] was designed for few tens of peers as it required each peer to maintain updates about every other peer in the network. Many ALM protocols with distributed construction and maintenance algorithm have been suggested in last one decade. CAN [29], DT based [30] Multicast and NICE [16] are few of them.

There exists a family of mesh based ALM protocols based on the infrastructure created using DHTs (Distributed Hash Tables) e.g. Pastry [3] based Scribe [31], CAN [29] based multicast and Chord [32] based Multicast. DHTs form a structured P2P network for efficient storage and retrieval of data where the responsibility of maintaining the mapping from object keywords to object values is distributed among the participating nodes and any join/leave of nodes causes a minimal amount of disruption. This DHT based design achieves high scalability by efficient routing while supporting continuous node arrivals, departures, and failures.

Chapter 3

Reliability Approaches in Overlay Muticast Networks

The overlay multicast network comprises large population of heterogeneous and dynamic peers. Shifting group management and packet forwarding responsibility from routers to end hosts gives deployment advantage over IP multicast and helps realizing large scale group communication, but places challenges in performance due to unpredictable and autonomous behavior of peers. In past few years, many resilience improvement schemes have been suggested for ALM protocols to improve delivery ratio while maintaining end-to-end latency and control overhead within a limit under high degree of transiency. Different schemes come with different tradeoffs among these parameters making one scheme to work better than other for a particular application.

Classification of reliability approaches, performance metrics of reliability, prominent resilience approaches with their basic mechanism and relative merits have been surveyed in this chapter.

3.1 Introduction

Use of application layer multicast (ALM) for bandwidth intensive streaming application for large number of autonomous and heterogeneous users gives scalability advantage of multicast and avoids deployment issues of a network (IP) level solution; though it has many reliability issues. In ALM, the network is built continuously on-the-fly with the end systems whose behavior is unpredictable. Unlike routers in IP multicast, participating nodes in ALM may join and leave at their will. Peers' transiency studies show that peers' median session time ranges from ninety to one minute [33]. In most of the ALM protocols, tree topology is a preferred choice for data distribution to save bandwidth. Thus high degree of transiency in participating nodes places a big challenge in maintaining reliability with minimum additional overhead in ALM streaming protocols.

In recent years, many authors have proposed schemes to improve resilience either by utilizing path diversity to make data delivery independent of nodes' behavior [22], [34], [35] or by adding data redundancy to the stream through coding [23], [26] [24]. In this chapter, a review of reliability schemes for ALM streaming protocols is presented. A classification of reliability approaches is given at the outset, some popular schemes are discussed in detail, performance evaluation metrics for reliability scheme are defined, and then, finally a comparative evaluation of schemes is done.

3.2 Classification of Reliability Approaches in ALM Streaming Protocols

Reliability approaches in ALM protocols can be classified in three broad categories: (i) Path diversity based approaches, (ii) Data redundancy based approaches and (iii) Combined approaches

3.2.1 Path diversity based approaches

A node failure, anywhere in the path from source to receiver may cause interruption in the reception of data at the end receiver; the obvious way to ensure reliability in such situation is to switch the transmission over an alternate path which is being maintained in advance. This basic concept has been applied in many forms in reliability solutions for tree based ALM protocols. Data transmission through an alternate path is maintained probabilistically [22] or deterministically [36] in some manner to make data delivery independent of nodes' transiency. Considering tree topology for data distribution, the redundant transmission can be done in any of the following forms.

3.2.1.1 Single tree topology

In reliability approaches of this type, extra links are added to the original single multicast tree topology for data distribution. The way redundancy is introduced in ALM protocols has two forms viz. Cross-link redundancy and In-tree redundancy. PRM (over NICE) [22] is an example of former type while Nemo [34] protocol is representative of the later type. The difference between these approaches is described in the next section where PRM is described in detail.

In single tree topology, redundancy can also be introduced by maintaining dual tree deterministically as suggested in [36]. In this protocol, topology is built incrementally while maintaining dual feeds of the media stream to any node from the source with minimum differential delay in receiving the packets via both the alternatives. The availability of a P2P query search network with distributed indexing service e.g. Chord, is assumed to be present for maintaining the list of presently active potential forwarders for the two different feeds. It helps in searching of feeds by a newly arrived node. These two feeds (named as feed 1 and feed 2) come via two different node-disjoint paths from the source and thus scheme is resilient to single failure at a time. The details of this scheme are given in chapter 5.

3.2.1.2 Multiple-tree topology

In reliability approaches of this type, several overlapping trees are created for data distribution. The data stream is divided into several substreams, each one being sent along each of the trees in multiple tree structure. SplitStream over Scribe [35] is a representative scheme of this class. Scribe is an ALM protocol that runs over Pastry, a DHT based structured peer-to-peer overlay. SplitStream is the reliability enhancement protocol that creates multiple node disjoint trees over which the substreams of data are disseminated. The detail of SplitStream is described in subsection 3.3.2.

Since in all the path diversity based approaches, redundant transmission is done in advance of the occurence of any loss, these schemes are also referred to as *Proactive* approaches.

3.2.2 Data Redundancy Based Approaches

In reliability approaches of this type, redundant data packets are sent along with the primary data packets, which can be used to reconstruct the original data in case some of the data packets are lost. This redundancy can be introduced through some source coding scheme. The representative protocols include FEC [23], Digital Fountain [24], Network Coding and Layered Coding Scheme e.g. Multiple Description Coding (MDC) with multi-path transmission [37]. Whereas, in most of the schemes based on data redundancy, additional redundant packets are sent in advance alongwith the primary data packets; there exists some schemes in which retransmission of packets is done after receiving Negative Acknowledgement (NAK) from the receiver side *e. g.*, in reactive component of PRM [22].

Thus depending on whether the redundancy is added in advance or added when triggerred by receiver; data redundancy based approaches may be categorised as *Proactive* approaches or *Reactive* approaches.

3.2.3 Combined Approaches

Some researchers have come up with the reliability approaches in which they introduce redundancy both in network path as well as in data itself e.g. Padmanabhan's approach [38] applied to Coopnet [39]. In [38], multiple, diverse distribution trees are used to provide path redundancy; and multiple description coding (MDC) [40] is used for data redundancy. Thus a combined approach is used, though both used in *proactive* form. The detail of [39] is described in the next section.

PRM over NICE [22] can also be categorized as using combined approach in the sense that it has two components viz. *proactive* and *reactive* simultaneously.

3.3 Example Reliability Approaches

3.3.1 PRM over NICE

The detailed description of NICE (recursive acronym for NICE is the Internet Cooperative Environment), a scalable ALM protocol, has been given in section 2.4.2. In probabilistic resilient multicast (PRM) enhanced NICE protocol, Banerjee *et. al.* [22] introduced multicast data recovery scheme with two components. A proactive component called randomized forwarding in which each overlay node chooses a constant number of other overlay nodes uniformly at random and forward data to each one of them with a low probability, and a reactive component called triggered NAKs to handle data losses due to link errors and network congestion. The PRM scheme is efficiently scalable in which, with increasing number of participants, the overhead at each overlay node decreases to zero asymptotically. The description of the two components of this scheme is as given below.

3.3.1.1 Randomized Forwarding

Randomized Forwarding, the proactive component of the scheme, introduces some cross edges in the data delivery tree by way of some additional transmissions with small probability, along randomly chosen overlay edges apart from regular tree edges. This helps in fast recovery of data under high failure rates of nodes. An example of proactive randomized forwarding is shown in figure 3.1 and 3.2. Each overlay node selects a small number (r) of nodes other than its children. On receiving the first copy of a data packet, it forwards the packet to these nodes with a small probability (β) apart from forwarding to its children in the delivery tree. Due to this additional forwarding, some nodes may receive multiple copies of the same packet. Such duplicate packets are detected and suppressed. A duplicate suppression cache is maintained that temporarily stores the data packets. Packets received after the latency deadlines are dropped. The cache size is limited by the latency deadline of the application. The overhead of the scheme is $\beta\ r$. The value of probability β is kept small (for example 0.01) and r takes value in the range from 1 to 3.

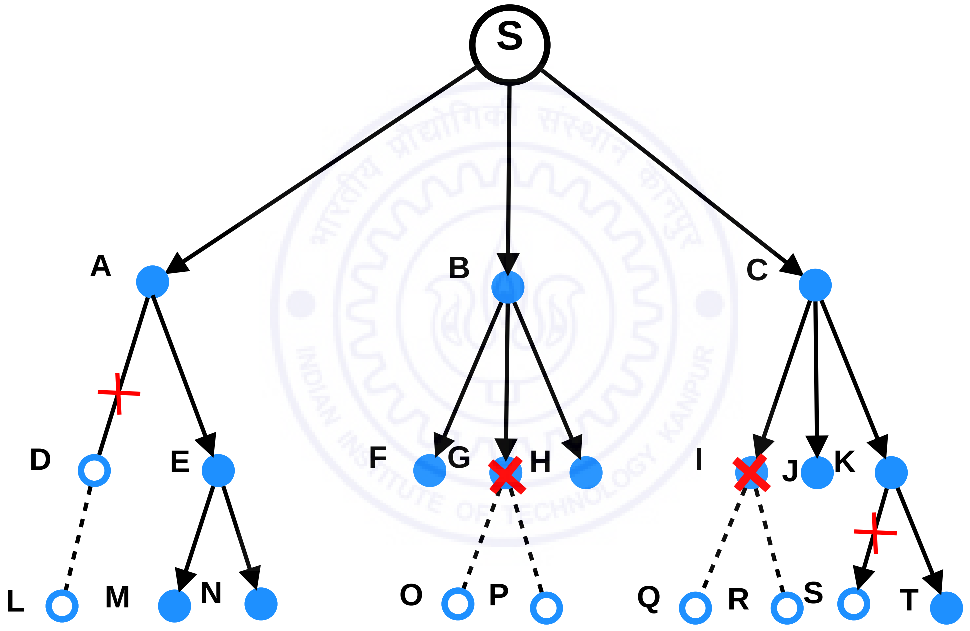

Figure 3.1: An overlay multicast network. The circles represents overlay nodes and crosses indicate the failures at links and nodes. The arrows indicate the data flow

Random Nodes Discovery: Analysis shows that if nodes for additional transmission are chosen uniformly at random, delivery of packet to each overlay node can be guaranteed with a high probability. Each node periodically dis-

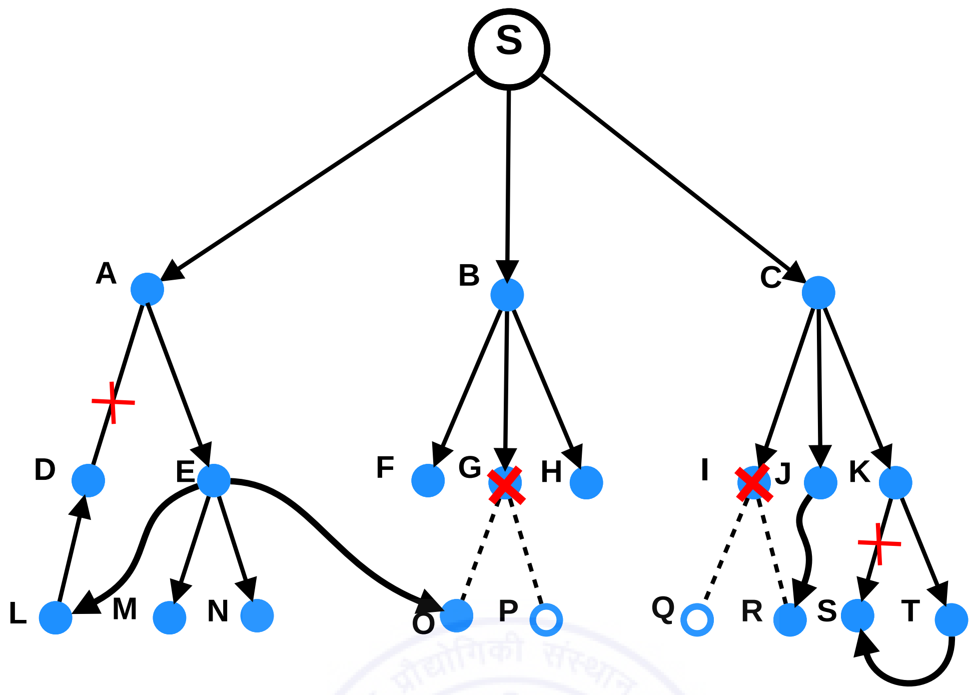

Figure 3.2: Randomized forwarding in PRM scheme. The curved edges show the randomized forwarding

covers a set of random nodes to whom additional transmission is to be done. The discovering node transmits a discover message with a TTL (time-to-live) to its parent. Parent then forwards it to one of its randomly selected neighbor which in turn forwards to another neighbor and so on, without retracing its path along the tree. The TTL value is decremented at each hop. The node at which TTL reaches to zero is selected as one of the random nodes.

Effectiveness of Randomized Forwarding: With high failure rate, a portion of the data delivery tree may get partitioned. If the cross edges over which additional data transmission is done are chosen uniformly at random, the number of cross edges on a selected subtree increases with the size of partition. Therefore larger the size of partition, higher is the probability of repair using cross transmission.

3.3.1.2 Triggered NAKs

The data packets are assigned a monotonically increasing sequence number so that interruption in the delivery can be detected using gaps in the sequence numbers. On detection of a gap, NAK-based retransmission is triggered. The information about correctly received and missed prior sequence numbers by a node is piggybacked in each packet forwarded to its children or the cross nodes. It makes the receipient node aware of the gaps in sequence numbers its parent has and prevents it from making NAK-based retransmission request for these sequence numbers. The receipient node will send NAK based retransmission request for those sequence numbers which its parent possesses but are not received by him. Once the parent node obtains missing sequence numbered packets, it immediately sends them to the children nodes. This saves network from unnecessary traffic.

An example of triggerred NAKs is shown in figure 3.3. Node K receives sequence number 65 from its parent C. The received packet also indicates that its parent has received sequence numbers 61, 62 and 63 but has missed sequence number 64. Node K has missed sequence numbers 62 and 64. It will send NAK only for sequence 62 but not for 64 as it is aware that even C does not have the sequence 64. The same way, node T, on receiving sequence 65 from K, sends NAK for 63 but not for 62 and 64. On receiving sequence 62 from C, K forwards it to T without waiting for any request from T for that sequence.

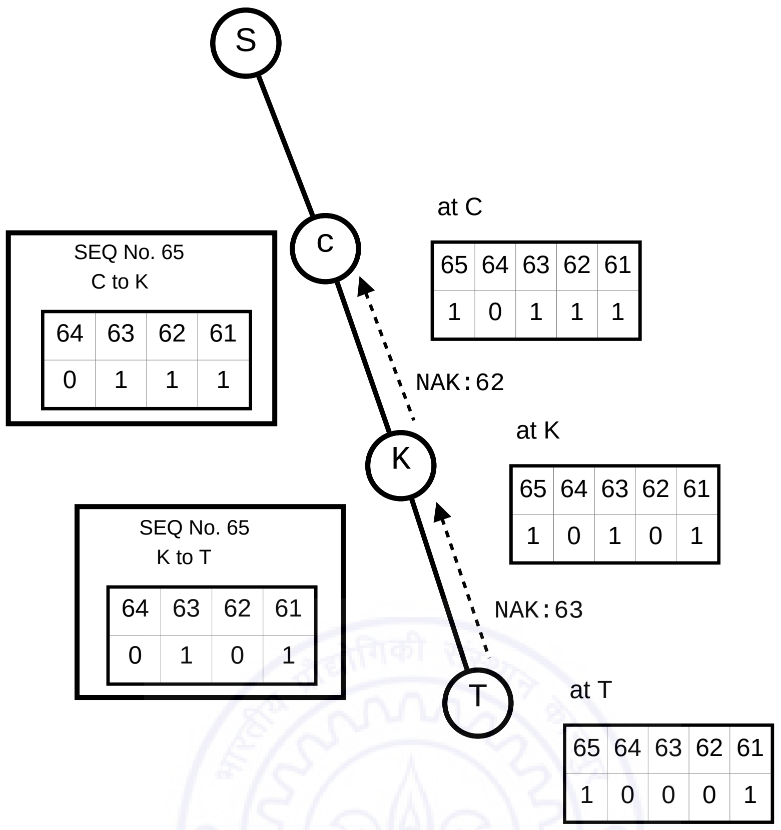

Figure 3.3: Triggerred NAKs in PRM with bit-mask length equal to 4

3.3.2 SplitStream over Scribe

In conventional tree-based multicast systems, a relatively small set of nodes are responsible for forwarding all the multicast messages. This may introduce bottlenecks in the forwarding topology as the induced load may easily overwhelm a specific end host. To address this problem, Castro *et. al.* [35] proposed the use of multiple interior-node-disjoint trees (a forest) over which stripes of the data stream are disseminated. A node can be leaf node in many trees but interior node in at least one tree. By forwarding different stripes over each tree and making each peer an interior node in at least one tree, the multiple-tree redundancy approach distributes the forwarding load more equally among the participating peers. DHT routing model based Scribe [31] makes creation and maintenance of this forest efficient. It neither requires expensive network monitoring nor it requires a centralized coordinator.

3.3.2.1 Pastry

Pastry [3] is an efficient, completely decentralized and self organized peer-to-peer look-up protocol for object location and routing. It distributes responsibility uniformly among participating nodes while keeping delay and link stress within acceptable limit. Pastry routes to any node in an overlay network of N nodes in O(log N) steps while maintaining only O(log N) entries in routing tables at each node.

Nodes are assigned a 128-bit nodeID that places it in a circular nodeID space ranging from 0 to $2^{128} - 1$. Node ID is assigned randomly to a node when it joins in and it can be a hashed version of its IP address or of its public key resulting in uniform distribution of nodeIDs in the circular space. Every node maintains a routing table, a neighborhood set and a leaf set. Routing table has $\lceil log_{2^b} N \rceil$ rows with $2^b - 1$ entries each. The neighborhood set contains nodeIDs and IP address of $|M|$ nodes that are closest (proximity metric) to the local node. The leaf set contains set of nodes with the $|L|/2$ numerically closest larger nodeIDs and $|L|/2$ numerically closest smaller nodeIDs. Typically $|L|$ and $|M|$ are 2^b. Configuration parameter b involves trade-off between the size of the populated portion of routing table and the maximum number of hops required in routing between any pair of nodes and its typical value is 4. Pastry routes with guarantee (unless $\lceil |L|/2 \rceil$ nodes with adjacent nodeIDs fail simultaneously) to the numerically closest node to the hash of a given key in less than $\lceil log_{2^b} N \rceil$ steps. Table 3.1 presents the state of some arbitrary Pastry peer with nodeID 10233102 (base 4).

When a new node (with nodeID X) arrives, it initializes its state tables and informs other nodes of its presence. We assume that it already knows about a bootstrapping node (nearby pastry node with nodeID A). Node X asks A to route the 'join' message whose key is X. Pastry routes it to the existing node Z whose ID is numerically closest to X. In response, A, Z and all nodes on the path from A to Z send their state table to X. Node X inspects these tables and it may further request state from additional nodes in order to initialize it own state table and that of the other affected peers.

A pastry node is considered failed when immediate neighbouring nodes in nodeID space can no longer communicate with it. To replace the failed peer in the leaf set of its neighbours, its neighbours in the NodeID space contact the live peer with the largest index on the side of the failed peer, and request its leaf table. An appropriate node from this accumulated set is chosen, then verified if it is still alive and then inserted in to its set. The neighbourhood set is not used for routing but to maintain locality properties.

3.3.2.2 Scribe

Scribe builds a scalable infrastructure over Pastry [3] for application layer multicast (ALM). It can support large number of groups with a wide range of group sizes and dynamic membership.

Any Scribe node may create a group with a groupID and a few others may join this group. Group members can multicast messages to other members of the group. Multicast tree of the group is formed by the union of the Pastry routes from each group member to the groupID's root. Being based on the proximity-aware Pastry substrate, group membership management in Scribe is highly efficient and decentralized. Though Scribe provides only best-effort delivery without conforming to a specific order; it offers framework

NodeID 10233102			
Leaf Set	*SMALLER*	*LARGER*	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
Routing Table			
-0-2212102	**1**	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	**2**	10-3-23302
102-0-0230	102-1-1302	102-2-2302	**3**
1023-0-322	1023-1-000	1023-2-121	**3**
10233-0-01	**1**	10233-2-32	
0		102331-2-0	
		2	
Neighborhood Set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Table 3.1: Pastry peer's routing table, leaf set, and neighbour set. An example of routing path for a pastry peer with ID 10233102, b=2 and ID length = 16 bit. All numbers are in base 4. Top row is row zero. Single digit cells in each row of routing table show the digits of this peer ID. The node IDs in each cell is is splitted to show the common prefix with 10233102 - next digit - rest ID.[3]

for implementing reliability.

3.3.2.3 SplitStream

In conventional tree-based multicast systems, a participating node may act as a leaf node or as an interior node. However only interior nodes share the responsibility to forward the data. In a full binary (fan-out = 2) tree there are almost equal number of leaf nodes and interior nodes. Consider a full binary tree of four levels, where root is at level-0, and other hierarchical levels are numbered as 1, 2 and 3. In level-1, there will be 2 nodes, 4 nodes at level-2 and at level-3, there will be 8 nodes. Thus there are 7 interior nodes whereas 8 leaf nodes (level-3 nodes), thus only half of the participating nodes share the forwarding load. The fraction of leaf nodes increases with the fan-out of participating nodes. Consider a full tree with all the nodes having a fan-out of 16. When all outdegrees are filled upto last but one level in this tree, 90 percent of the peers are leaves and only 10 percent of the peers contribute in forwarding. Thus single tree based multicast has uneven forwarding load distribution.

To address this issue, Castro et al.[35] proposed the use of a forest of multiple interior-node-disjoint trees over which stripes of the data stream are disseminated. A set of trees is said to be interior-node-disjoint if each node is an interior node in at most one tree, and a leaf node in other trees. By forwarding different stripes over different trees the forwarding load is more equally distributed among the participating peers. This multiple-tree forest can be created and maintained efficiently in a distributed way by availing the inherent properties of Pastry in Scribe [31]. Efficient SplitStream forest can be constructed in which each peer has forwarding bandwidth equal to the received bandwidth.

SplitStream can also handle heterogeneity by accommodating peers with different bandwidth capacities. Consider that the original media stream has a bandwidth requirement of B and splitting of stream is done into k stripes. Now the in-bound bandwidth requirement and out-bound bandwidth requirement can be controlled in steps of B/k. A media stream can be encoded using MDC (multiple description coding) so that the video can be reconstructed from any subset of the k stripes with video quality proportional to the number of stripes received and thus permits low bandwidth clients to receive the reduced quality video by explicitly requesting for fewer stripes. This also helps when some interior node in a stripe tree fails. The deprived peers can continue with the reduced quality video until the stripe tree with failed node is repaired.

The SplitStream Approach: Figure 3.4 describes how splitstream balances the forwarding load among participating peers. Consider a simple example where the original stream (banwidth B) is split in to two stripes (each having bandwidth $B/2$) and multicast in two different trees. There are total 8 participating nodes each with fan-out equal to 2. Each peer is an internal node in only one tree and forwards the stripe to two children.

The SplitStream Design: Splitstream utilizes the pastry routing to construct interior-node-disjoint trees. In Pastry, a message is forwarded toward nodes whose nodeID share progressively longer prefixes with the hash of the message's key. As the Scribe tree is formed by the union of the Pastry routes from each group member to the groupID's root, the nodeID s of all interior nodes share some number of digits with the tree's groupID. Thus if groupIDs of the k Scribe trees are chosen such that these all differ in the most significant digit, the trees will have a disjoint set of interior nodes. Figure 3.5 describes the example Splitstream forest construction. The nodeIDs of interior nodes

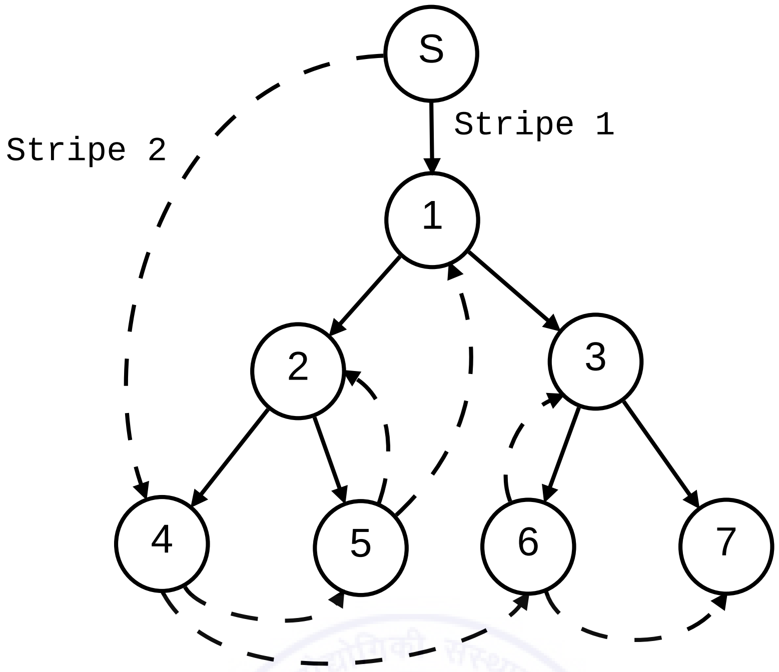

Figure 3.4: The Splitstream approach. The stream is divided into two stripes and each stripe is sent over independent multicast trees. Any peer is an interior node in one tree and is a leafnode in the other

share a prefix with the stripID, thus they must be leaves in the other trees. For example node M with a nodeID starting with 1 is an interior node in the tree for the stripID starting with 1 and a leaf node in other trees.

3.3.3 Resilient Coopnet (Cooperative Networking)

In Coopnet [39] , Padmanabhan *et. al.* suggested a self scaling and cost effective approach to address flash crowd problem in traditional client-server system, occurring during events of great interest; for example, the incident of September 11, 2001 terrorist attack swamped the major news websites such as CNN and MSNBC, that made sites unavailable and response times went over 45 seconds [41]. In most such cases, network bandwidth is the major

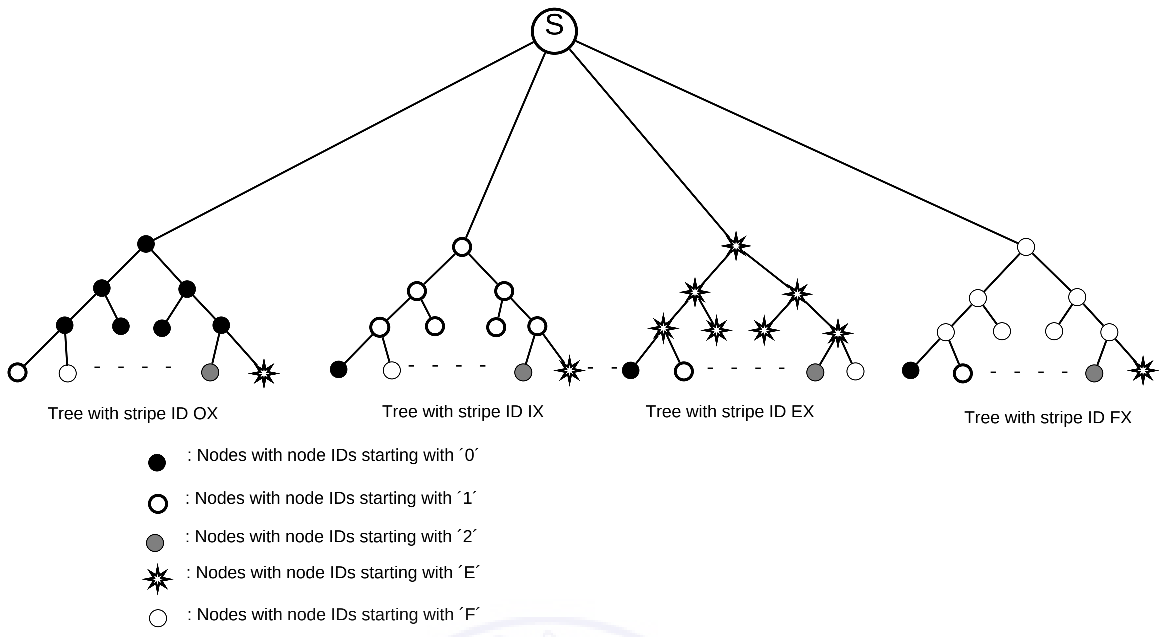
S
Tree with stripe ID OX
Tree with stripe ID IX
Tree with stripe ID EX
Tree with stripe ID FX
: Nodes with node IDs starting with ´0´
: Nodes with node IDs starting with ´1´
: Nodes with node IDs starting with ´2´
: Nodes with node IDs starting with ´E´
: Nodes with node IDs starting with ´F´

Figure 3.5: Forest construction in Splitstream. Trees have disjoint set of interior nodes. The content is splitted and each stripe is multicast in its designated tree. Each stripe's *stripID* starts with a different digit

constraint, rather than server CPU resources. There are three solutions for this problem:

1. *Proxy Server Caching:* It installs distributed clusters of servers. In this approach, wide deployment of proxy caches is necessary. Intermediate caching of objects reduces bandwidth consumption and hence reduces network traffic. But it has low scalability and needs to avail the services of a Content Distribution Network (CDN).

2. *CDN Based Approach:* It requires outsourcing the services of an infrastructure based content distribution networks (CDN). Infrastructure based CDNs, for example Akamai [42] applies dedicated machines to store and distribute the content on behalf of server, and ensure high availability of content both during flash crowd as well as normal load.

It maintains multiple points of presence with web server replicas called surrogate servers. It is highly scalable and works well even for resource hungry applications like media streaming, but for small websites it is not an affordable solution.

3. *Peer-to-Peer Content Distribution:* P2P approach is low cost, scalable and effective. Whereas, first two approaches can be applied without any involvement of clients, the P2P approach engages end systems to cooperate in storage and distribution.

3.3.3.1 Coopnet

The Coopnet [39] approach spurs cooperation among clients and thus triggers the system for peer-to-peer distribution, where clients become peers to each other. With P2P approach, the bandwidth available to serve content scales with demand. Coopnet is unique in the sense that cooperation among clients is necessary only during flash crowd, its P2P feature become dormant when normal situation resumes. The central server remains there even when network has invoked P2P behaviour. Existence of central server simplifies the content location as compared to expensive distributed content search in a P2P system. End hosts arrange like a P2P network to improve the network performance only when it is necessary.

Coopnet asks clients to serve content to other clients and thus reducing load on the server. Server can redirect some of the requesting clients to other clients that have downloaded content in recent past. Clients then resend the request to one or more of these peers. Evaluation of Coopnet was done using simulation based on traffic traces collected at the MSNBC (Microsoft National Broadcasting Company) website during 9/11 flash crowd.

3.3.3.2 Resilient Coopnet Protocol

In resilient coopnet [38], redundancy is provided in both network paths as well as in data paths. The multiple tree approach is used for diversity in distribution and Multiple Descriptions Coding (MDC) gives redundancy in data paths, resulting in upto a 22 dB improvement in peak signal-to-noise ratio (PSNR). Resilience has been taken as the prime objective in Resilient Coopnet, while efficiency has been put as the secondary goal.

Tree Management: Nodes arrive and depart frequently and contribute their resources as long as they are interested in receiving content. The The trees with short depth minimize the probability of disruption due to churn; therefore the balanced and bushy trees are the preferred. Out degree of nodes is kept as large as its bandwidth allows.

Diversity in distribution trees requires that the set of ancestors of a node in each tree should be as disjoint as possible. A distributed tree management would have supported scalability much better but it requires longer join and leave processing times, therefore Coopnet opted for centralized tree management. In practice, a central node which is the root of the tree (source of data stream) coordinates tree construction and maintenance. Root is usually more resourceful, its availability is expected to be high and it should be able to support quick join and leave.

Join: A joining node contacts the root node first and root node responds with information about designated parent nodes in each tree. The joining node then connects with these parents and start getting data.

Leave: In graceful departure, the leaving node informs to the root node. Root node then finds a new parent for the deprived children and notifies their identities to them. To handle abrupt leave, every node keeps monitoring the packet loss rate of incoming stream in each tree. If the loss rate exceeds a threshold, the node checks with its parent to confirm whether parent too is facing loss on that tree. If the parent is also experiencing the loss, the node holds off for a while with a hope that the parent will resolve the problem. In case, parent is not facing loss or he could not resolve the problem, the node contacts the root node requesting for a new parent. Root responds with a new parent and records a complaint against the old parent. This information is used in future parent selection.

Deterministic Tree Construction:

The centralization of tree construction helps respecting the bandwidth constraints of each node. Making each node interior in just one tree makes the trees more bushy and shorter rather than having randomized tree construction. It also contributes to diversity and robustness. When a new node joins, it has to be fertile (interior node) in one tree and sterile (leaf node) in all the other trees. The count of fertile nodes in each tree is kept track of. The tree with least number of fertile nodes becomes the fertile tree for a new node, thus balancing the number of fertile nodes in each tree. In its fertile tree, the new node starts from the root and comes to the lower levels until a level is found that either has a node with room or a node with sterile child. In case a node with a room is found, that node becomes parent. Otherwise node with sterile child becomes parent of the new node and new parent for the sterile child is found. The idea behind is to make the upper level of tree populated by fertile nodes.

There may be situations when large number of fertile nodes from the same tree departs making that tree unable to support new nodes. In such situations, a fertile node is picked from the tree which has largest number of fertile nodes and migrated to the tree which is capacity starved.

3.3.4 Kunichika's Approach for Reliability

In Kunichika's approach [25], targeted for single source live streaming application, authors proposed a proactive backup route maintenance over redundant overlay tree. A degree-constrained spanning overlay tree is obtained in such a way that all the nodes always keep one free out-degree by force. The total outdegree of an end host is defined as the ratio of it's connection bandwidth divided by media playback rate. Redundant structure of tree avoids exhaustive search of a backup parent and simplifies backup route calculation. Since backup parents are pre-calculated, these are applied immediately after a failure.

Through simulations, it is verified that in this approach, the recovery latency reduces drastically without increase in the control overhead since layers to which the backup route calculation is applied are limited at worst to the grandchild layer. The control overhead in this scheme is far less than the schemes where additional data traffic is continuously sent, like in PRM approach that amounts to a heavy traffic overhead in applications like media streaming.

3.3.4.1 Host Joining

In this approach, any end hosts with an out-degree n caters only to $(n-1)$ children. Forcing to keep one free outdegree simplifies and expedites backup route discovery and contributes to reduced control overhead. Any newly arrived nodes starts its search for parent from the root. If root has free outdegree it accommodates it otherwise redirects the request to its children, and if even at this level no node is found with two free out-degree it is again redirected and so on. In the host joining example shown in figure 3.6, it is assumed that each node has total outdegree equal to four, out of which one is kept reserved for backup route in failure cases. A newly arrived node, node 8 first requests to the root and is refused as it had only one free outdegree and is redirected to its children 1, where it gets connected.

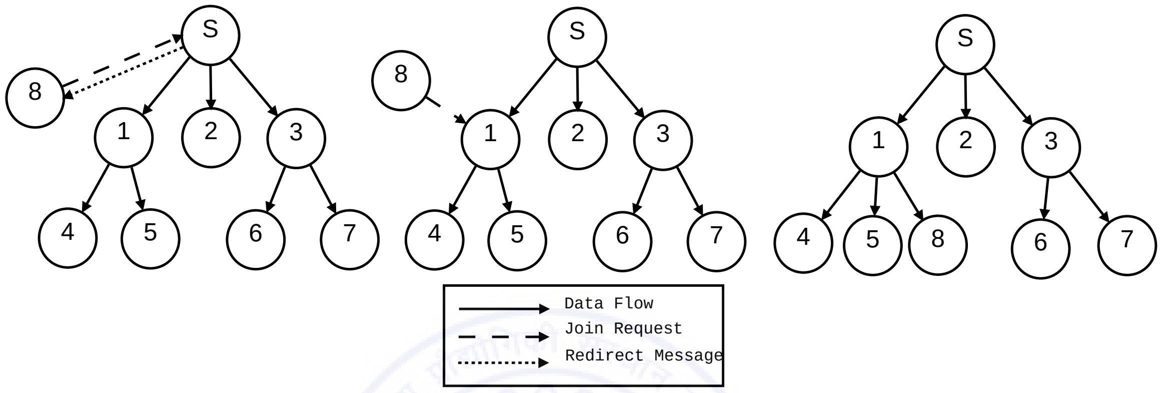

Figure 3.6: New node joining in Kunichika's approach

3.3.4.2 Backup Route Calculation

Backup route calculations are done on every incident of node joining and node leaving. As soon a new node joins in the distribution overlay, its grandparent needs to redetermine the backup route for that newly joined node as well as for all its siblings. Each node keeps track of its grandchildren and calculates the backup for them as follows. If number of grandchildren are more than the accommodation capacity of grandparent it arranges all of its grandchildren in increasing distance order based on RTT measurements. In case its child fails, corresponding deprived grandchildren are rearranged for feed. The nearest grandchild connectes directly to its grandparent, the second nearest grandchild get connects to the nearest grandchild and so on. An example is shown in figure 3.7, where on failure of node 1, all its children, in order of their distance from source node, are fed from failed node's parent (here source) directly or in sequence. The nearest one, node 4 get connected to it directly, the second nearest node, *i.e.* node 5 connects to node 4 (parent-child way) and then then finally node 8 connects to node 5 (parent-child way).

once the immediate remedy has been done for the failure, and redundancy has been utilized, the tree is reconstituted and backup routes are recalculated in order to regain the redundancy for any future failures.

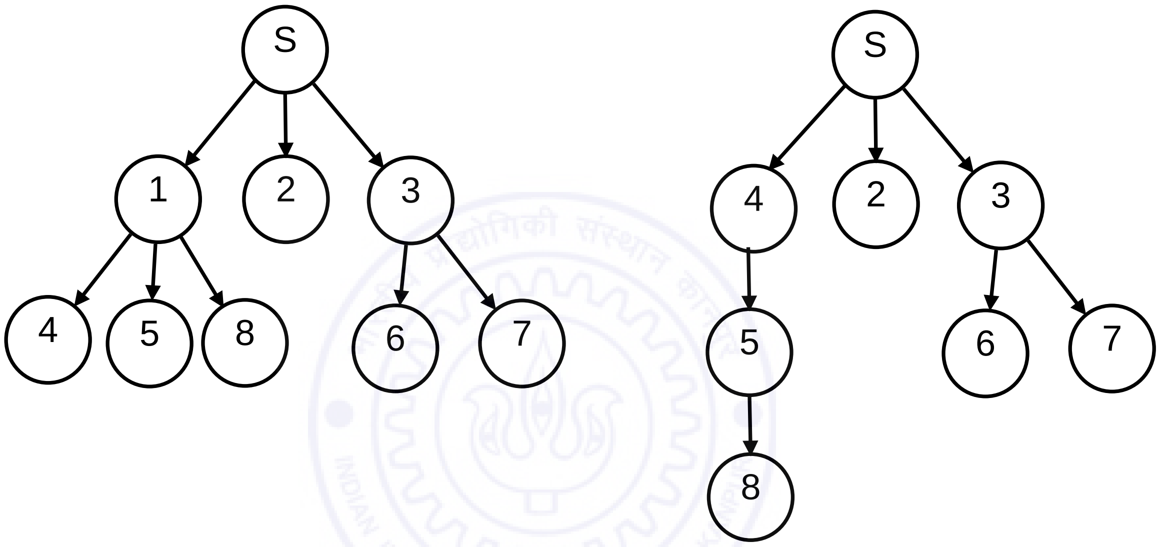

Figure 3.7: Rearrangement of chilldren nodes as a node fails. Node 1 of example overlay tree (shown left) fails and its children nodes, *i.e.* node 4, 5 and 8 are rearranged in sequence (shown right)

3.3.4.3 Limitations

The limitation of this approach is that it needs to maintain permanently redundant degrees at each node. Proactive backup routes reduce control overhead but at the cost of latency as the free out-degree at each node amounts to a larger tree size for a given number of participants.

3.4 Performance Metrics for Reliability

A reliability scheme can be evaluated in terms of improvement in application performance perceived at users' end and the cost involved. Following parameters may be used to quantify the performance of a reliability scheme:

Recovery Latency: It is defined as the time to reestablish transmission through alternate path after the interruption.

Delivery Ratio: It is defined as the ratio of subscribers receiving a packet within a fixed time bound

Overhead: It is defined as the amount of additional network traffic generated when reliability approach is applied due to its message overhead

3.5 Conclusion

For live media streaming multicast in a large scale network without a central coordinator, reliability is a challenge. It may be because of node churn at high rate, instant arrival of thousand of nodes at the start of the session, heterogenity in connection bandwidth and connection type (wired as well as wireless to accommodate mobile devices in the network) all contributing to the loss of reliability in the system. For live streaming application, to avoid any interruption at user end, a combination of proactive and reactive approach is required. It has been seen that in many schemes, there is a tradeoff between the level of reliability achieved and the overhead required. Efforts are still on to achieve a higher reliability in a large scale, dynamic and heterogeneous network.

Chapter 4

Maintaining Biconnectivity in Unstructured Overlay Network

Application layer multicast (ALM) also called Overlay Multicast, is an attractive alternative solution to most of the problems associated with IP multicast. In ALM, multicast-related functionalities are moved to end-hosts. Application layer multicast builds a peer-to-peer (P2P) overlay topology consisting of end-to-end unicast connections between end-hosts. The key advantages, overlays offer, are flexibility, adaptability and ease of deployment [12]. The general approach to build an application layer multicast architecture involves tracking network characteristics and building appropriate topologies by allowing the end users to self organize into logical overlay networks for efficient data delivery. The major concern in designing ALM protocol is the mechanism to build and maintain a topology and to route data efficiently and reliably in this topology. In this chapter, we propose a two-fold dynamic overlay tree construction and maintenance scheme in which a mesh-like topology is first built. In the mesh, an arriving host connects to two already connected hosts. This ensures that two node and link disjoint paths are always maintained between every possible pair of nodes. Once the mesh is formed, on top of it, a single or multiple data delivery tree(s) are built using a suitable protocol. An algorithm is run in the nodes of the overlay topology to maintain the biconnectivity by inserting new links and deleting the redundant links as a continuous process.

4.1 Introduction

The Internet has seen an unprecedented growth due to the success of one-to-one applications such as reliable file transfer, electronic mail and http based information access over web. IP multicast [10] at network layer defines an efficient way for multicasting whereby the sources transmit only one copy of the data and intermediate multicast enabled routers make the required number of copies for onward transmission. However Internet lacks in multicast support as it evolved primarily for unicast applications since beginning [43]. Thus, most of the Internet's infrastructure is unicast-only and does not provide efficient support for real time multicasting applications viz. Internet-TV, Video conferencing, Live Lecture Delivery Systems (LLDS) and content delivery networks over Internet. In these applications, copies of a message need to be transported to multiple recipients at different locations. IP multicast requires routers to maintain group states on per-group basis; that leads to scaling constraints when number of groups become huge. Although, possibly using aggregation techniques, the number of states can be reduced, IP multicast still needs changes at infrastructure level (deployment/enabling of multicast routing protocols in the routers) in the Internet.

Internet Protocol (IP) is based on best-effort data delivery, and hence cannot support QoS. This is not desirable in applications where real time synchronous data delivery is needed such as multi-party gaming and multi-party conferencing. It may be noted that the QoS gaurantees are not possible until they are implemented in all the lower layers of the network. Finally IP multicast provides only a limited support for group management, multicast address allocation and network management.

In IPv4, multicast is an optional service as it evolved and stabilized much later than the implementation of IP for Internet. As a result, most of the networks have not enabled multicast or provide it only as a value added service. Therefore multicast has been mostly limited to 'islands' of network domains under single administrative control or in local area networks.

In order to overcome the above limitation, the application layer multicast (ALM), also known as Overlay Multicast has been studied extensively as an attractive alternative by moving multicast-related functionalities to the end-hosts. We can see this as a mechanism to provide multicast services using whatever available unicast network services. The key advantages, overlays offer, are flexibility, adaptability and ease of deployment [12]. Since the multicast connections are based on the end hosts, there is no need of multicast enabled network routers. ALM achieves multicast via piece-wise unicast connections.

However ALM incurs a performance penalty over IP Multicast. Links near the end users carry redundant copy of data and also the delay to some of the end users is more than what would have been in the case of IP multicast. The major concern in ALM is formation of an efficient topology for reliable media transport. Switching off a single node has the potential to partition the whole overlay multicast network thereby interrupting media feed distributions to some of the subscribing nodes.

To enhance the reliability in Unstructured Overlay Networks, we propose the creation and maintenance of a dynamic overlay tree built over bi-connected mesh overlay topology. In order to create and maintain bi-connected mesh, a new host connects to two already connected hosts in the overlay network. This results in at least two node-disjoint paths between any pair of nodes. A dynamic algorithm is run in the mesh to maintain biconnectivity by suitably adding new links when needed while also deleting the redundant links. The detail of algorithm to construct and maintain a biconnected mesh is given in subsection 4.3.1. Approaches for biconnectivity in tree-first ALM protocols are suggested in subsection 4.3.2.

Once the mesh is formed, on top of that a data delivery tree is built using an appropriate distributed algorithm. Our simple broadcast based algorithm (subsection 4.4.2) ensures the constant tracking of an alternate path (every node has knowledge of an alternate route to source) for data, so that if feed stops from the current path, it can switch over to the alternate path by sending the feed request in that direction. The algorithm ensures that the next best path to source is again identified. The detail of the algorithm is given in section 4.4.

The next section gives a brief survey of related work. Section 4.3 describes proposed approaches toward maintaining the biconnectivity. In section 4.4, algorithm for data distribution tree overlay in bi-connected topology has been suggested. In section 4.5, a brief evaluation of suggested algorithms is done. Finally we conclude in the section 4.6.

4.2 Related Work

In this section, we describe the reliability schemes proposed in the major unstructured ALM protocols. In Narada protocol, Chu et. al. [15], proposed a mesh based topology design for small group size. Data delivery trees are constructed entirely from the overlay links present in the mesh. Shortest path spanning trees are constructed per source by running distance vector algorithms on top of the mesh. Periodically, each member generates a refresh message with monotonically increasing sequence number and exchanges its knowledge of group membership with its neighbors in the mesh. Mesh partition is detected, when members on one side of the partition stop receiving sequence number updates from members on the other side. Each of such members is probed to determine if it is dead, else a link is added to it. Each of the members on one side may attempt to add new links to some partitioned member on other side; this situation is probabilistically resolved such that in spite of several members simultaneously attempting to repair partition, only a small number of new links are added.

To improve the mesh quality, dynamic addition and dropping of links is also done. Each member periodically probes some random member that is not a neighbor, and evaluates the utility of adding a link to this member. Link is added if expected utility gain exceeds a given threshold. To drop a link, members periodically compute the consensus cost of its link to every neighbor. The consensus cost of a link (i, j) is defined as $max(cost_{ij}, cost_{ji})$, where $cost_{ij}$ is the number of members for which i uses j as next hop for forwarding packets, and $cost_{ji}$ is the number of members for which j uses i as next hop for forwarding packets. The link with lowest consensus cost is dropped if the consensus cost falls below a certain threshold.

In PRM enhanced NICE protocol, Banerjee *et. al.* [22] introduced multicast data recovery scheme called Probabilistic Resilient Multicast (PRM) with two components. A proactive component called randomized forwarding in which each overlay node chooses a constant number of other overlay nodes uniformly at random and forward data to each of them with a low probability and a reactive component called triggered NAKs to handle data losses due to link errors and network congestion.

4.3 Our Approaches toward Biconnectivity for Resilience

The work presented in this chapter suggests approaches toward maintaining bi-connectivity in data distribution topology in the application layer multicast networks.

The basic principle behind these approaches is that if two or more nodes form a ring among them, then between any node pair inside the ring, two node-disjoint paths exist. Further if two such rings (or two bi-connected components) share a common link or more than one connected links then those bi-connected components are again bi-connected.

4.3.1 Construction and maintenance of bi-connected mesh

In the First approach, an algorithm is suggested to construct a bi-connected Mesh over which a distribution tree can be formed. Such a biconnected mesh can be the foundation for mesh-first ALM protocols.

4.3.1.1 Adding newly arriving Node in the Network

Each new node connects to two already existing nodes in the network. Each node has a unique identity number that may be the function of its IP address (for example), known as its identifier. An example bi-connected mesh formation is described in figure 4.1. In the beginning, when first node comes, there is no network and the source node, denoted as circled **'S'** is the first node in the network. The second node connects to the first node. The third node connects to the first and second node forming a triangle. Fourth node connects to any two of the three nodes. The fifth arriving node sees the source node and node 3 as having least degree hence it connects to them.

The rule followed is that each arriving node connects to two already existing nodes that are connected with least number of nodes. In case more than two nodes have same degree, any two can be used for the connection with equal probability. Continuing this way, a bi-connected mesh can be formed with any number of nodes where any node pair maintains two node disjoint paths. An example mesh with 12 nodes (assuming first node is source node) is shown in figure 4.1. The similar algorithm can be applied to obtain any general k-connected mesh, where each node pair maintains k node-disjoint paths.

Further it is assumed that any already existing node while connecting to any newly arrived node tells that node about all his neighbors to whom it is already connected. Also, already existing node informs about newly joined node to all its previously existing neighbors. This information is used in mesh repairing as explained later in this subsection.

4.3.1.2 Analysis of the Mesh so formed

In the mesh so formed, the average degree of nodes in the mesh, as defined below, increases slightly with the number of nodes and reaches to a constant value of 4 as the number of nodes increases to more than 100 (shown in figure 4.2). The average degree of nodes in a network with N number of nodes can be obtained as below.

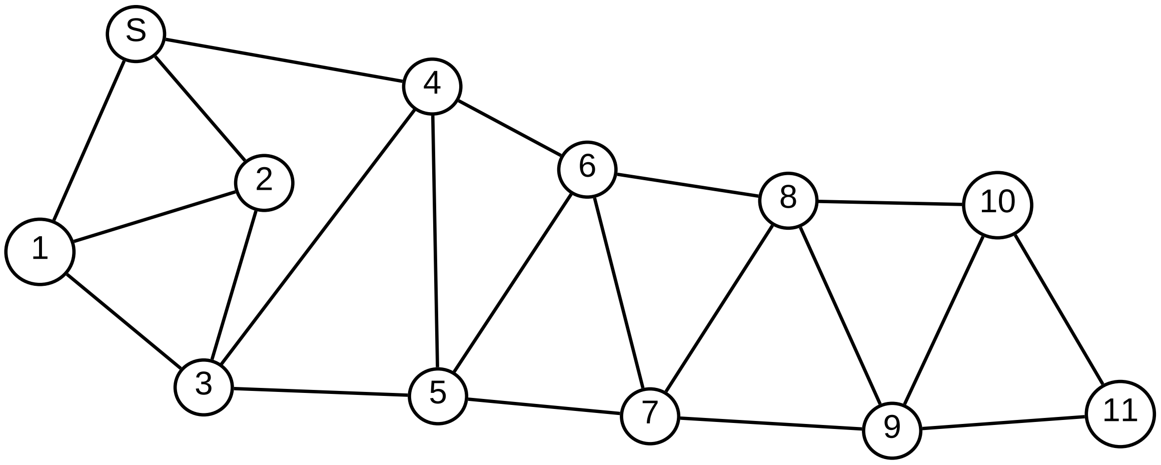

Figure 4.1: A bi-connected mesh with 12 nodes

$$Average \quad degree \quad of \quad nodes = \frac{\sum_{i=1}^{N} \quad degree \quad of \quad node \quad i}{total \quad number \quad of \quad nodes} \tag{4.1}$$

Average number of hops required to be travelled for any source destination node pair in the mesh is calculated and plotted in figure 4.3. The average has been taken over all source destination pairs. It is observed that the number of hops increase almost linearly with the number of nodes.

4.3.1.3 Deleting redundant Links from the Network

For each existing link in the network, the decision is taken periodically whether to maintain that link or to delete it. If the nodes connected by a link have two node disjoint paths even when the link is removed, that link is declared to be redundant and can be deleted from the mesh.

4.3.1.4 Steps taken in case of node failure

In case any node fails in the network, the biconnectivity to those nodes is affected which were directly connected to the failed node. In the biconnected

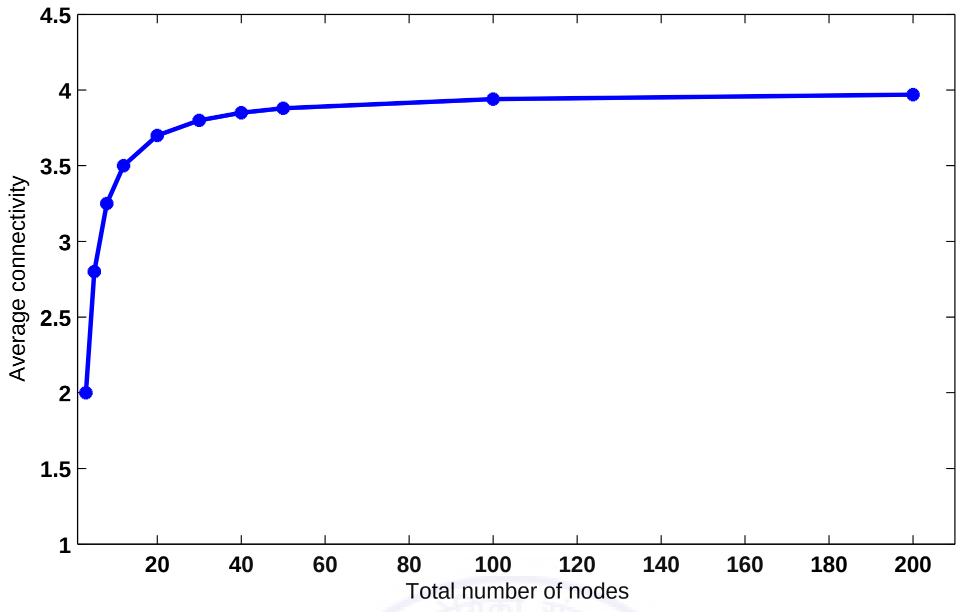

Figure 4.2: Average node degree versus network size

mesh shown in figure 4.1, it can be easily observed that on failure of any node (except for the first two nodes connected to source), exactly four nodes are affected. Therefore repair process attempts to recreate a ring among these nodes. To regain the lost bi-connectivity, links are added between these four affected nodes. While adding new links, condition that degree of a node does not exceeds four is fulfilled. It results in a deterministic solution where any affected node connects to the neighbor of its direct neighbor. This fact is confirmed with the example described below.

We consider the example case of failure of three adjacent nodes in sequence, one after another. In the example topology considered as shown in figure 4.1, if node 6 fails, its associated links do not work and thus biconnectivity destroys for nodes 4, node 5, node 7 and node 8 (figure 4.4). As assumed initially, in the start of this subsection, all the affected nodes know

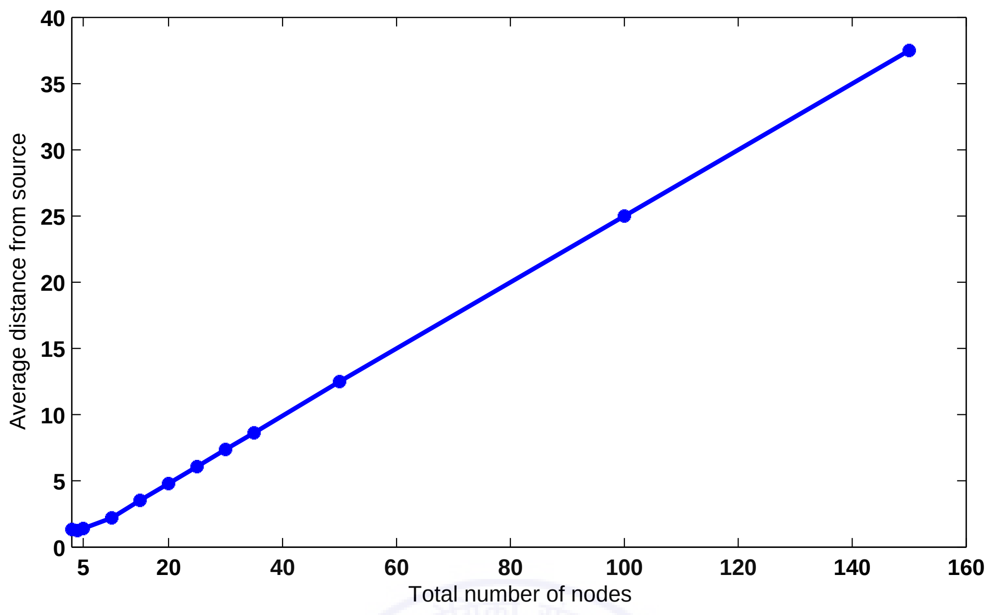

Figure 4.3: Average number of hops; averaged over all source destination pairs

in advance about all other neighbors of node 6, thus every affected node knows who are the other affected nodes due to this failure. New links are added between affected nodes with the degree condition, as stated above, is fulfilled. As a unique possible solution, two links are added; one between node 4 and node 7 and the other between node 5 and node 8, thus re-establishing bi-connectivity.

Further, after sometime, the adjacent node of node 6, *i. e.* node 7 fails. The repair algorithm creates two new links; one between node 4 and 8 and another between node 5 and node 9 (figure 4.5). Further if node 8 also fails, to maintain biconnectivity, again two new links are created; one between node 4 and node 9 and another between node 5 and node 10 (figure 4.6). The final form of the repaired topology is shown in figure 4.7. In the final topology, it can be easily confirmed that the original connectivity pattern is maintained even after successive failure of three adjacent nodes.

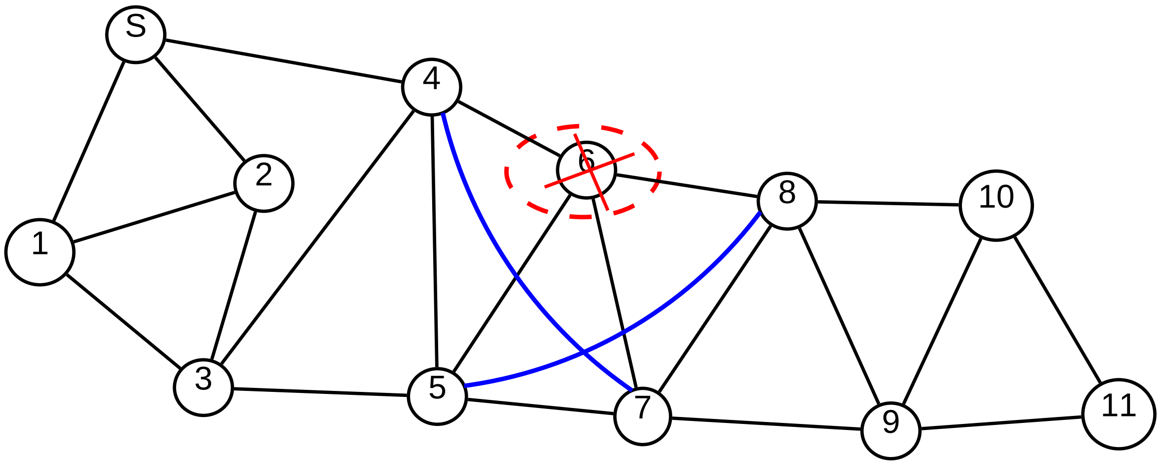

Figure 4.4: Failure of node 6. Addition of new links; one between node 4 and node 7, other between node 5 and node 8 reestablish bi-connectivity

4.3.1.5 Advantages and limitations to the Approach

In the mesh so formed, maximum degree that a node can have is limited to 4. But the maximum number of hops between any pair of nodes increases linearly as the number of nodes increase.

4.3.2 Approach for biconnectivity in the tree overlays

It is also possible to create a tree topology overlay without creating an underlying bi-connected mesh overlay network. For such overlay topologies, we can introduce bi-connectedness property for achieving reliability. This bi-connectivity is achieved through rings formation and then bringing every node in at least one ring. In the achieved bi-connected topology, a distribution tree needs to be maintained using the algorithm presented in section 4.4.

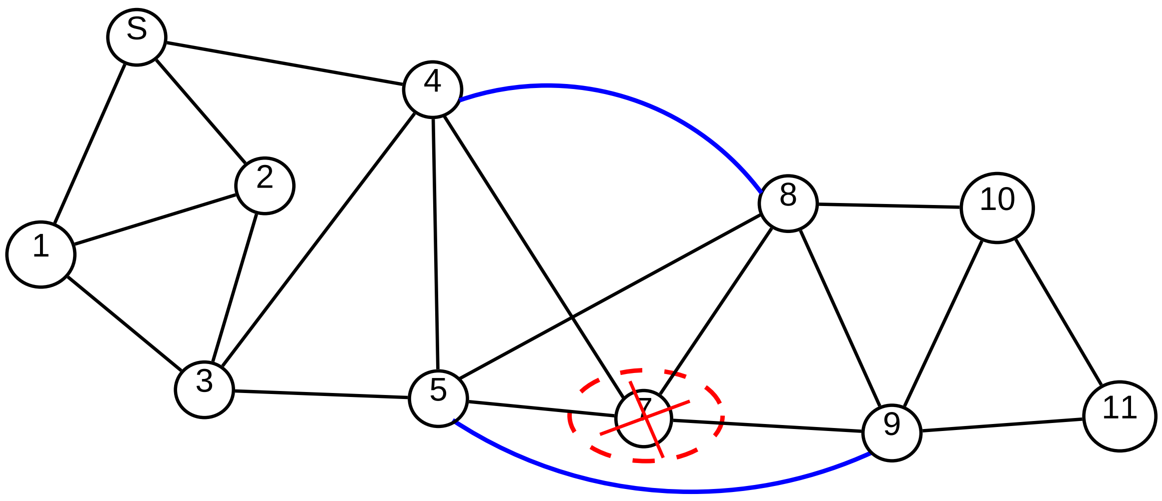

Figure 4.5: Failure of node 7. Addition of new links; one between node 4 and node 8, other between node 5 and node 9 reestablish bi-connectivity

4.3.2.1 Connect-to-grandparent approach

Biconnectivity in this approach is achieved by connecting all the nodes to their grand parent, which provides an alternate path to get the feed in case their parents fail. Wherever it is not possible to connect a node to its grandparent, that node is connected to its siblings. An example binary tree, having 4-level hierarchy, shown in figure 4.8 is considered. Figure 4.9 shows the newly added links that are required to be added in the original tree topology of figure 4.8, to create rings among nodes and then bringing every node within some ring.

In the child-grandparent approach, for a binary tree, the number of additional links required is $(N - 2)$; where N is number of nodes in the network.

4.3.2.2 Connected adjacent-leaf-nodes approach

Biconnectivity in this approach is achieved by connecting all the adjacent leaf nodes in pair. By connecting leaf nodes this way, any node pair in the

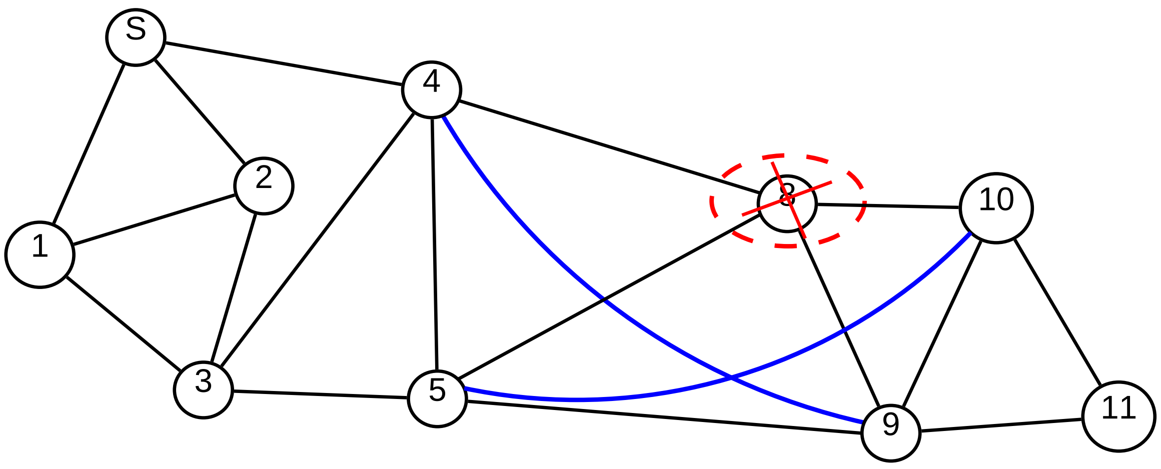

Figure 4.6: Failure of node 8. Addition of new links; one between node 4 and node 9, other between node 5 and node 10 reestablish bi-connectivity

tree gets bi-connectivity, as rings are formed through the leaf nodes. Here again we consider the same example binary tree, as earlier (shown in figure 4.8). Once the adjacent leaf nodes are connected, shown in figure 4.10, rings are formed between any pair of nodes. Any node can get information about its adjacent leaf nodes through his parent (if the adjacent leaf node is its sibling), by his grandparent (if adjacent leaf node is its cousin) and so on.

The number of additional links required to achieve bi-connectivity in this approach is $(N-1)/2$; where N is the number of nodes in the network.

4.3.2.3 Connected least-correlated-leaf-nodes approach

Biconnectivity in this approach is achieved by first dividing the tree in to two symmetric equal halves and then any leaf node of one half of the tree pairs with a leaf node in the other half of the tree, as shown in figure 4.11. By connecting leaf nodes this way, any node pair in the tree gets bi-connectivity, as in earlier two approaches but with smallest number of additional links (half of the additional links required in Connected adjacent-leaf-nodes approach).

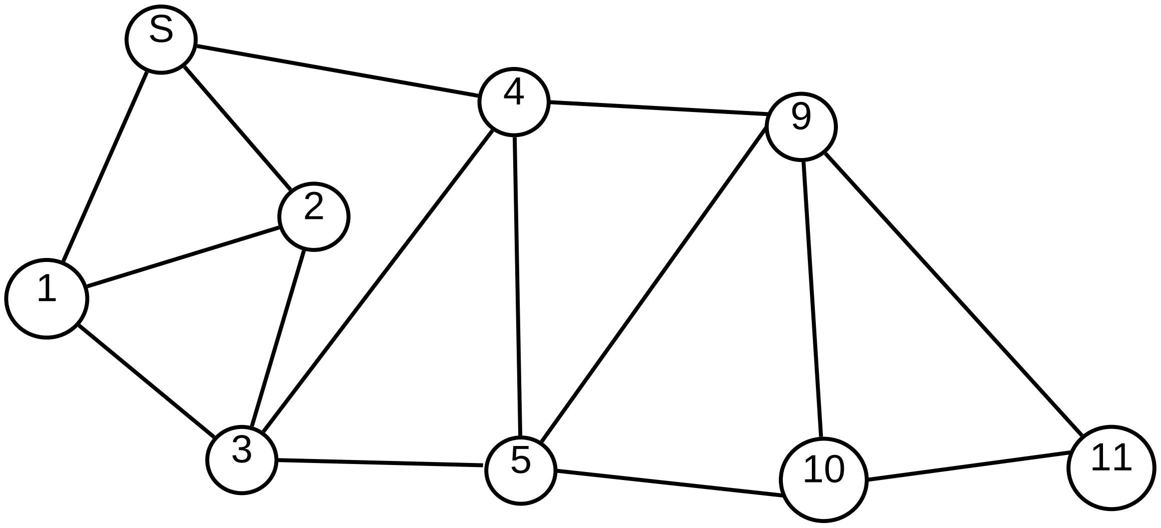

Figure 4.7: Final form of topology after successive failures of three adjacent nodes, *i. e.* node 6, 7 and 8. Biconnectivity pattern remains maintained.

Algorithm for this approach can be stated as following. Assuming binary tree, the two children of any node are given direction index as left (L) and right (R) children, for being situated either at left or right side in the next level. Assuming source at level 0, each node cumulates and forwards this information to its children in the next level and thus the direction index string is maintained at every node. This string can form the basis for defining the correlation of paths from source to any two nodes in the tree. Thus every node knows for every level, whether its direction index is left (L) or right (R). Any leaf node can pair with any other leaf node given their level 1 direction index is different. This actually ensures that path from source to these two nodes are least correlated and tree gets biconnectivity with least number of additional links, as shown in figure 4.11.

In case, no leaf node is available with different level-1 direction index, then the difference of level-2 direction index can be the next best choice for pairing but such pairing may require more number of links to make the tree

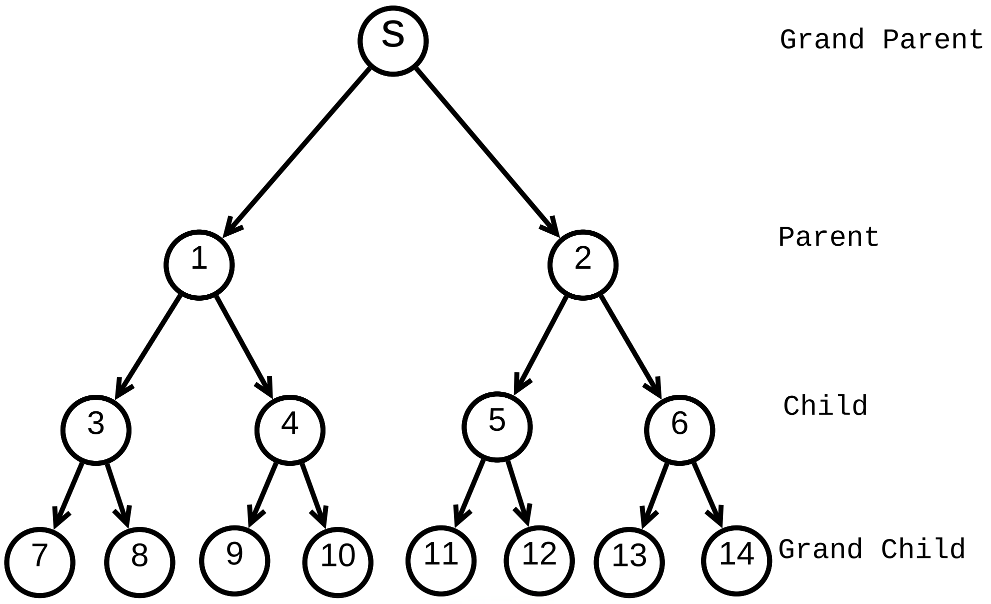

Figure 4.8: An example 4-level binary tree

biconnected.

The number of additional links required to achieve bi-connectivity in this approach is $(N + 1)/4$; where N is the number of nodes in the network.

Figure 4.12 gives comparison of the above three approaches considering additional links required as a metric.

4.4 Algorithm for data distribution tree overlay in biconnected topology

We assume initially that there exists a mesh containing all the nodes who are participants in a session. The existence of a bi-connected topology is assumed. Further in the case of node or link failure, we assume that repair mechanisms do exist which will restore the bi-connectedness in the topology. This will ensure resilience to any further failures.

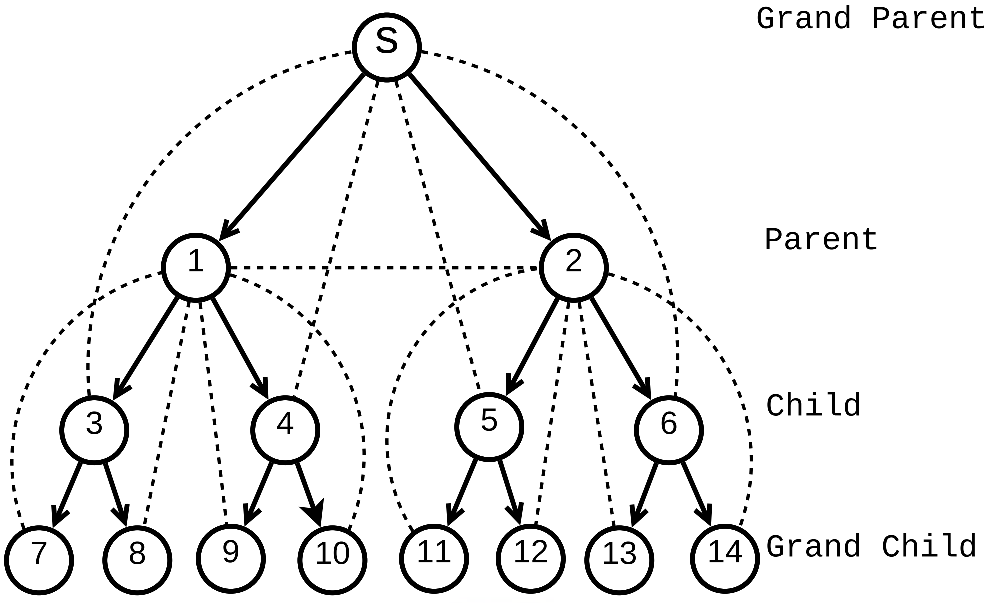

Figure 4.9: Connect-to-grandparent approach. All nodes are connected to their grandparent and if grandparent is absent then connected to their siblings

4.4.1 Data Forwarding

For distributing the data packets belonging to a live stream, each node maintains a **forwarding table**. Packets coming from the shortest path to source (*i.e.* the earliest arrived one) are forwarded to the nodes listed in the table. When a join request is received from a neighbor, its address is added to the forwarding table, and packets are forwarded to it also thereafter.

4.4.2 Topology Maintenance

4.4.2.1 Obtaining two best paths

Following steps are performed to obtain two best paths toward source:

(i) The source of the stream periodically broadcasts signaling packets (beacons) containing source id and sequence number to all the neighbors.

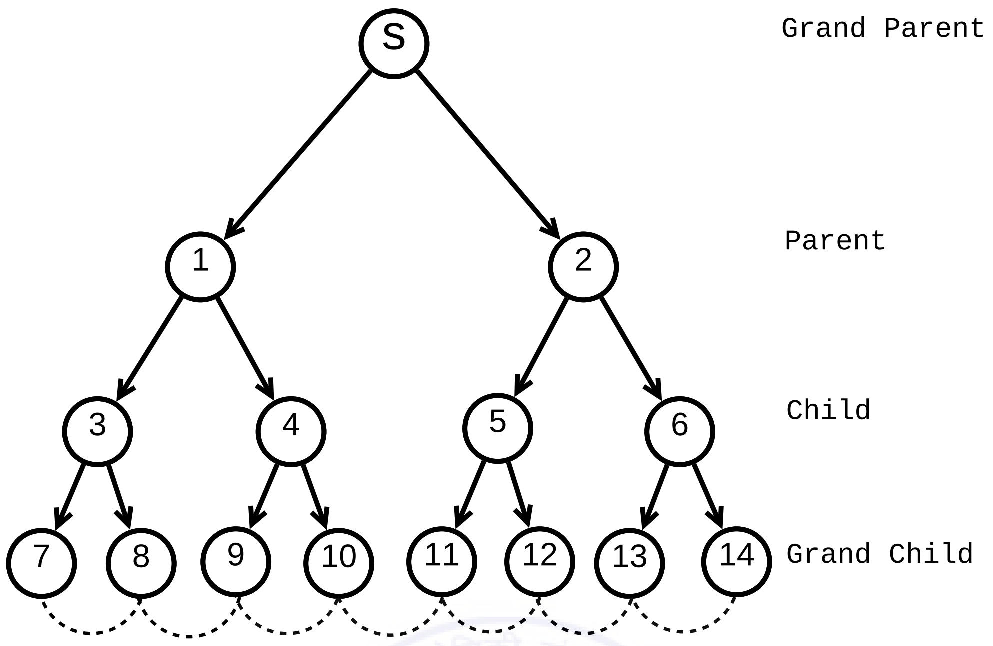

Figure 4.10: Connected adjacent-leaf-nodes approach. Adjacent leaf nodes are paired to achieve bi-connectivity

The format of signaling packet is shown in Table 4.1. The sequence number is incremented by the source for every new packet which is broadcasted.

(ii) In {source_id, Sequence_number, from_neighbor, next_hop_to_source} format, the packet detail is included in a table and listed as first best path towards the source (refer Table 4.2), whenever a node gets a packet with new sequence number from some source ID. This packet is then further broadcasted to all the neighbors except to the one from which it was received.

(iii) When a packet with the same sequence number (the one already listed with the node) arrives at a node from a different neighbor, it is also listed as second best path toward the source in the 'next hop to source

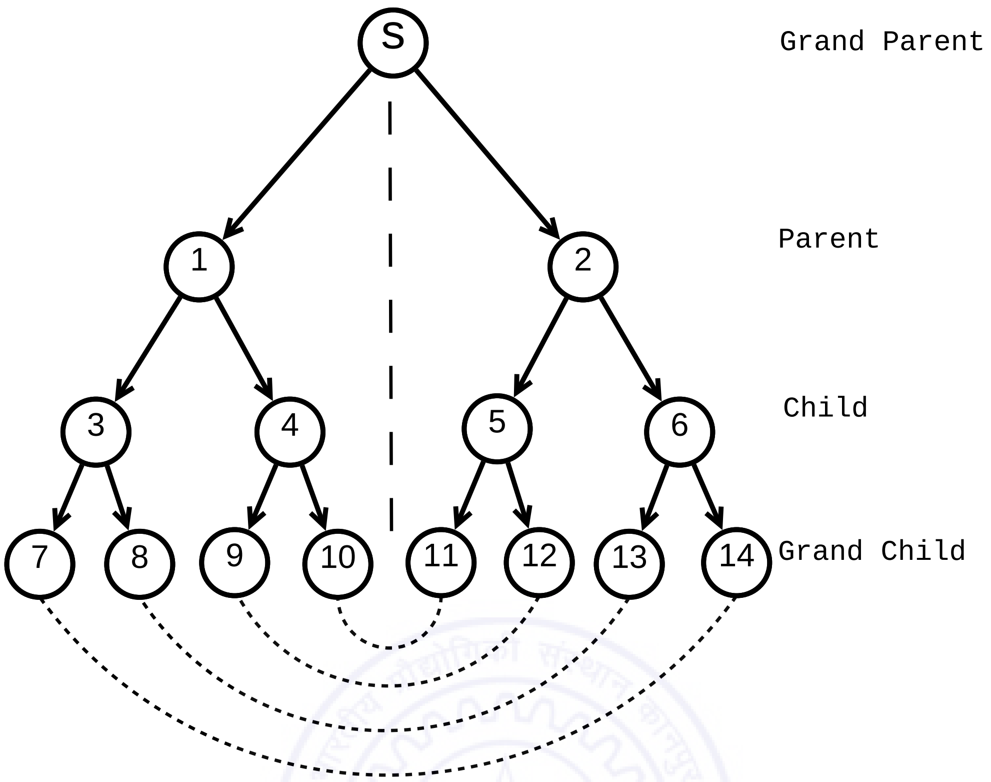

Figure 4.11: Connected least-correlated-leaf-nodes approach. Leaf nodes from two different halves of the tree get paired to achieve bi-connectivity

table'. Thus the alternate paths to source are always available at every node. This packet is forwarded only to the next hop towards source via the first best path.

(iv) Any further packets with the same sequence number from any more of the neighbors are simply discarded.

Each node thus maintains the two neighboring nodes which provide the two best paths towards the source. A join request will be forwarded to the best available neighbor node whenever a node wants to receive a data stream.

Life Time is a number that starts with a fixed value proportional to the expected expiry time desired and it decreases with time. It determines the

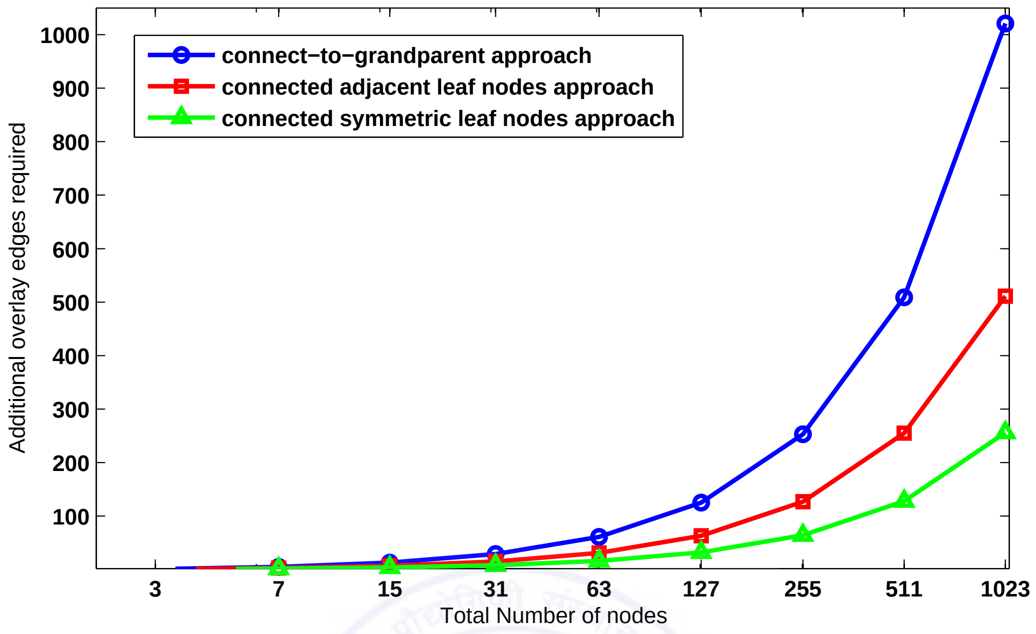

Figure 4.12: Number of additional links required to achieve biconnectivity by different approaches

From source ID	Sequence number	Hop distance from source	Life time

Table 4.1: Format of signaling packet

validity of a beacon packet.

The entry **next hop toward source ID** helps tracing back the route toward the source. At each node, a record of received sequence numbers is kept in the next hop to source table. This information is stored at each node, so that when a packet (same or different) comes from the same source, it can be compared with the already received packets. Any node receiving the above signaling packet will update these entries before broadcasting it further to all the outgoing links except the one from which it is received. If new and larger sequence number is received from a different neighbor, it indicates that a new and shorter route exists and the next hop entry is updated. The two neighbors from whom first and second packets of same sequence number are received are updated as first next hop and second next hop (to be used in case of failure of reception from first one).

Source ID	Last sequence number received	**First path**			**Second path**		
		Next hop toward source	Hop distance from source	Timer value	Next hop toward source	Hop distance from source	Timer value

Table 4.2: Format of 'next hop to source' table at each node

Also the life time value from the packet is copied to timer value in the table. The timer value will keep on decreasing with time. With the timeout, the information for that path is deleted. The alternate path information is again entered when beacon packet from same source is received from another neighbor.

Usually 16-bit binary **sequence number** is used and this length can generate 65536 different numbers before repetition occurs. Since the rule is that the entries are updated only if sequence number of received packet is more than the sequence number in the table entry, the timeout allows the update of tables when sequence number 0 is used after completion of a cycle or due to resetting of counter at the source.

4.4.2.2 Failure handling

Since packets are broadcasted from the source, a downward node can obtain the same packet from different routes earlier or later. This way the node can find many paths to the source, but it stores only two best paths. When a node leaves the overlay, the overlay path to some nodes may fail. Though the transmission can immediately be resumed from the alternate path that is already stored in the next hop to source table.

4.4.2.3 New node joining

When a new node joins the overlay, it will also receive beacon after some time and will know the two best next hop neighbors towards source. It will send a JOIN request packet to best next hop neighbor towards source. If the neighbor is receiving the feed, it will update the forwarding table for multicast and start sending the feed to the new node. If it is not receiving the feed, it will make the entry of new node in the child table for feed and further send the JOIN to the next hop to source. This happens till the JOIN reaches a node (in the worst case to the source) which is getting the feed.

4.4.2.4 Adaptive beacon broadcast rate

Beacon packets broadcasted from the source play an important role in establishing the paths toward the source. In the situation of failures, the beacon broadcast rate can be increased with the increase in the failure rate of nodes so that sudden multiple failures do not interrupt the data delivery. The node that detects a failure of a particular node may request directly to the next best hop towards the source along with the failed node ID and a request to increase the beacon broadcast rate. Then source doubles the rate of broadcast after that particular failure for a certain period. This enables the replenishment of second best path entry in the nodes which are left with only one entry due to the feed failures.

4.5 Evaluation

Our approach for bi-connectivity maintains an underlying resilient bi-connected mesh. On top of it, a data distribution overlay algorithm is run. Every node maintains the knowledge of two best paths to source. In the situation of failure of one path, feed is made available from the second best path while the broken path is repaired. If no packets are received for certain duration, a node can assume the failure of first feed path, and can send the data delivery request to the second best path. Since there is continuous broadcast of beacon packets, a new path to source will be registered shortly after failure as the second best path. Once the repair of broken path is done, after a while a beacon packet is sure to come from this repaired path and again the two best paths are updated and feed is taken from the first best path.

The time from the instant, a node fails to the instant when a repaired path is registered can be considered as restoration time (T_{res}) of the algorithm. The restoration time thus includes the time to detect a failure (T_{det}), time to establish additional links at mesh level (T_{add}) and time to sense and register optimum path (T_{reg}) after repair is done.

$$T_{res} = T_{det} + T_{add} + T_{reg} \tag{4.2}$$

If the restoration time does not exceed the time between two successive failures, uninterrupted data delivery is maintained from the duplicate paths already stored.

In our approach, the restoration time depends on network diameter, since a new path can be registered only on arrival of fresh beacon packet from the source. This puts the limitation in large networks under high failure rate.

4.6 Conclusion and future work

Bi-connectivity of a topology is essential for provisioning of reliable multicast transmission. Three different approaches toward maintaining bi-connectivity have been presented in this chapter.

As an extension of the approaches discussed in the work discussed in this chapter, for the networks where all nodes are not equally equipped and some nodes are more vulnerable than others, we can opt for tri-connected mesh instead of bi-connected. If the data distribution overlay algorithm (discussed in section 4.4) with the similar extension is applied over this tri-connected mesh, it results in three best paths to source. Since only signaling information is required to be propagated to maintain these paths, only a small bandwidth is consumed and hence it proves as an viable option for vulnerable nodes to maintain three best paths.

As a further extension, a large (heterogeneous) network can be divided into different protective zones; where different zones can be protected with bi-connected, tri-connected or n-connected mesh, and 2, 3 or n best paths can be maintained depending on the vulnerability of different zones.

Chapter 5

Dualpath Approach for Reliability in Overlay Multicast

Major concern in designing ALM formation and management protocol is how to build and maintain a topology to route data efficiently and reliably. In this chapter, we propose a scheme in which the topology is built incrementally while maintaining dual feeds of the media stream to any node from the source with minimum differential delay in receiving the packets via both the alternatives. We have assumed the availability of a P2P query search network. This enables building of multicast tree directly as an overlay. There is no need of maintaining an overlaid mesh and running multicast routing protocol to maintain a multicast tree in this mesh. The proposed scheme is much more simplified than the ones proposed in the earlier works and has almost full reliability in receiving packets from the source.

5.1 Introduction

To enhance the reliability in Structured Overlay Networks, variants of Dual-Path approach are proposed in this chapter. In these variants, the availability of peer-to-peer (P2P) query search network has been assumed. One representative peer-to-peer query search (look-up) protocol is Chord [32] protocol. The brief description of Chord look-up mechanism is given in subsection 5.1.1. In this network, the distributed index of resources is maintained. For ALM implementation, the resources will typically be end nodes who can potentially act as media forwarders (equivalent of multicast routers). Each node desirous of receiving the media feed searches for the resource in distributed index in the P2P overlay. It connects to one of the nodes found in the distributed index who could potentially provide the feed. Once it gets the feed, then it adds itself as a potential resource (feed) provider in the distributed index. Once a node is acting as forwarder for sufficient number of nodes, it can remove itself from the distributed index as, now it may not be able to accept requests for the feed from the new nodes without appreciable degradation in the performance. It shall be noted that at any point of time, there are two overlays, one for maintaining the distributed resource index, also called query network and the second one the multicast data transport network. It is possible that some nodes simply receive the feed and do not add themselves to index, thus acting as free riders. In the scope of this chapter, we are assuming that free-riders do not exist. Exploring the strategies for identifying and then penalizing the free riders has been left as future scope.

In the mesh based overlays, where the multicast tree is created through a separately running multicast routing protocol in the overlay, the mesh itself is created in such a way that a node or link failure does not partition the overlay network. Since the data delivery overlay in our scheme is a tree, the partition problem is prominent. Failure of a node can fragment the multicast topology unless the recovery mechanism is built. Moreover a streaming application usually has a playback deadline by which data delivery and loss recovery have to be accomplished. This makes stringent requirements on the recovery mechanisms.

In the proposed query search network based scheme, data forwarding topology is incrementally built. The topology governs the data forwarding in such a way that it maintains the double feed to any node from the source. These two feeds come via two different node-disjoint paths from the source and thus resilient to single failures in the overlay network. The query search network with distributed indexing service maintains the list of presently active potential forwarders for the two different feeds viz. feed 1 (f_1) and feed 2 (f_2). Any newly arrived node finds two nodes that can provide it duplicate feeds *i.e.* f_1 and f_2 using this query network.

All the Peers form a structured overlay for forwarding the streaming media. Each node maintains a table of next hop clients to whom media stream has to be forwarded. It also maintains links with parents from which duplicate feeds (f_1 and f_2) are received. The algorithm further has provision for healing of topology after a failure to keep the two feeds intact to every node via two node-disjoint paths from source.

5.1.1 Chord Look-Up Mechanism

Structured ALM protocols assume the support of query search network for resource finding service, over which data distribution overlay is formed. Chord is a scalable version of its predecessor look-up protocol, Consistient Hashing [44], that has a good load balancing property. Chord stores key/value pairs for distributed data items. Given a key, Chord maps the key to a node responsible for storing the key's value known as root node for that key. Each node maintains routing information of about O (log N) other nodes, and resolves all lookups via O (log N) messages to the other nodes. Nodes' leaving/joining requires $O(log^2 N)$ messages to update the routing information. A brief decription of Chord look-up mechanism is presented below.

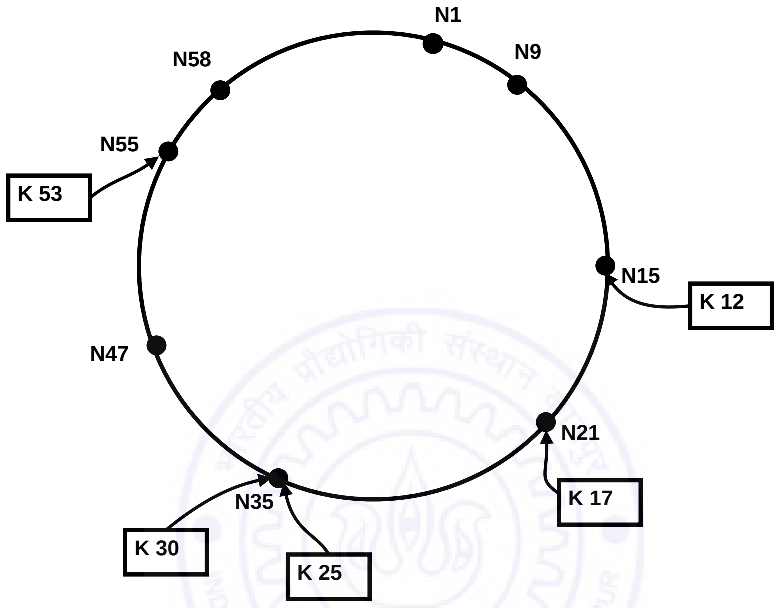

Figure 5.1: The basic Chord ring: Identifier circle having 8 nodes and 5 keys with their unique identifiers

Chord has an identifier circle defining a large addressing space. All the nodes and keys are assigned unique m-bit ID on this circle by applying some hashing function (SHA-1) on node's IP address/key. Identifiers are ordered in this circle modulo 2^m. An example Chord identifier circle with m = 6 (2^6 identifiers defined) is shown in figure 5.1. A key is assigned to the first peer whose identifier is equal to or follows k in the identifier circle, called as *successor(k)*.

Each node maintains a finger table. The i^{th} entry in the table at node n

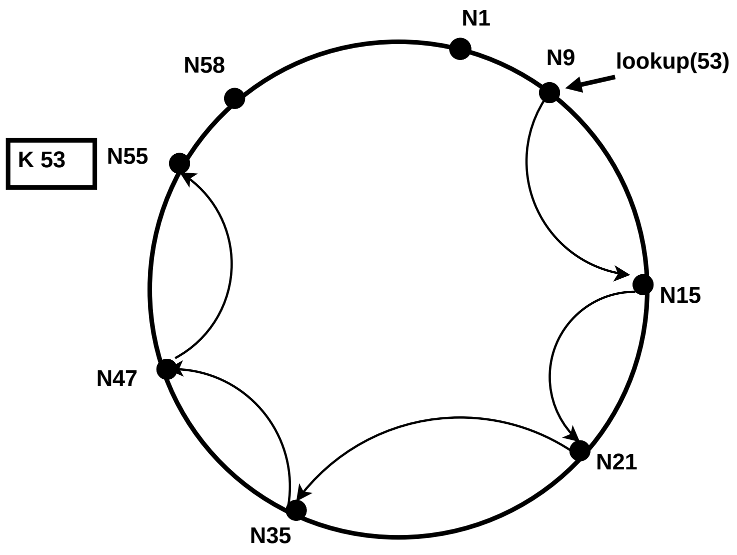

Figure 5.2: Finding successor node of given identifier ID: Slow routing with Basic look-up mechanism where every node keeps information only about its immediate neighbor

contains the identity of the first node, s, that succeeds n by at least 2^{i-1} on the identifier circle, i.e., $s = successsor(n + 2^{i-1})$, where $1 \leq i \leq m$. Node s is called i^{th} finger of node n. A finger table at node 9 in our example chord ring is shown in figure 5.3. The finger tables' pointers at repeatedly doubling distances cover half of the circle combinely. Thus each iteration halves the distance toward the given target. Figure 5.2 shows a slow look-up without applying Chord protocol whereas in figure 5.4, fast lookup is performed with Chord applied.

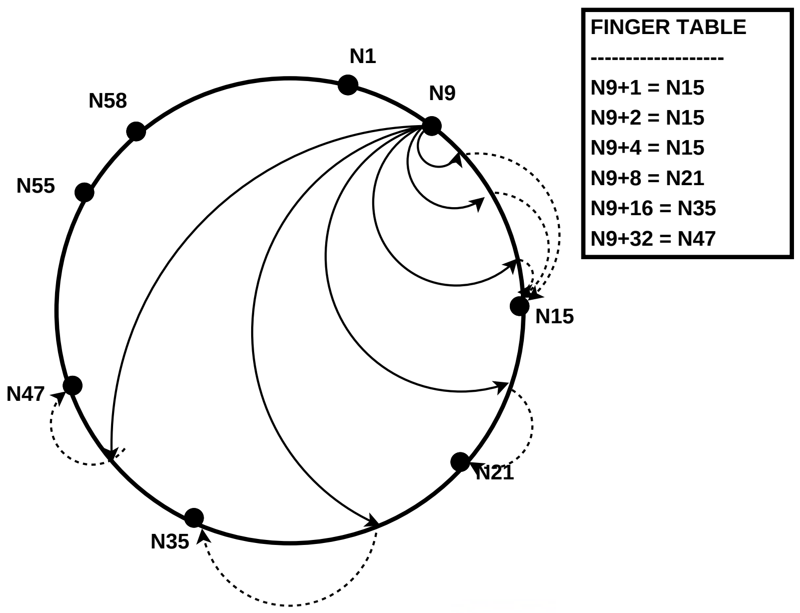

Figure 5.3: In Chord Protocol, each node maintains a finger table. Finger Table entries for node 9 are shown

5.2 Related Work

5.2.1 Scalable overlaid multicast topology creation

Shi and Turner [45] took the case of an Overlaid Multicast Network (OMN) that provides multicast services for real-time audio and video streaming applications through a set of distributed MSNs (Multicast Service Nodes), which communicate with hosts and with each other using standard unicast mechanisms. Authors in [45] have attempted to optimize end-to-end delay and MSN interface bandwidth usage at the routing sites. To solve the minimum diameter, degree limited spanning tree problem, Compact Tree (greedy) algorithm is used that builds a spanning tree incrementally. To solve limited diameter, residual-balanced spanning tree problem, Balanced Compact Tree algorithm is used, which proved to be effective in achieving a good balance

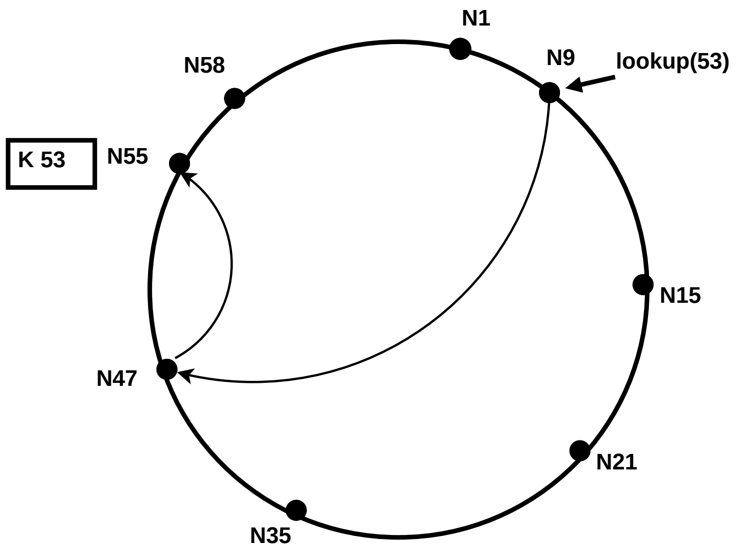

Figure 5.4: Finding successor node of given identifier ID: Fast routing with Chord look-up mechanism applied

between residual degree and diameter bound.

Banerjee *et. al.* [16] described a low overhead hierarchical clustering scheme called NICE for ALM. It is meant for low bandwidth real-time data applications with large receiver sets. This protocol is robust in the sense that failure of any number of group members does not affect the other members in the group. Members are assigned to different layers. Hosts in each layer are partitioned into a set of clusters. The cluster size lies between k and $3k-1$, where k is a constant. All hosts join to the lowest layer L_0. The leaders of all the clusters in lower layer join next higher layer. There are at most $log_k N$ layers, where N is the total number of member nodes. The highest layer has only a single member. In every layer, each cluster forms a clique (fully connected mesh) as the control topology and star as the data topology.

5.2.2 ALM loss recovery approaches

ALM loss recovery approaches are broadly classified as proactive and reactive. In a proactive approach, redundant data packets are sent along with the regular data packets which can be used to reconstruct the original data in case some of the data packets are lost. Proactive approaches include Forward Error Correction (FEC) [23], Digital Fountain [24], Network Coding and layered coding scheme e.g. Multiple Description Coding (MDC) with multi-path transmission. Kunichika's approach [25] is also proactive.

In a reactive approach, lost packets are retransmitted after the receiver requests for them. Probabilistic resilient multicast (PRM) [22] includes both proactive and reactive components.

In [46], it has been mentioned that a good loss recovery approach should have low residual loss rate, low recovery latency, low recovery overhead and low deployment overhead.

In FEC-(d, r) [23] [26], the source takes a set of d data packets and encodes them into another set of $d+r$ packets with redundancy incorporated and sends them. A receiver can recover d data packets if it receives any d of the $d+r$ encoded packets. Overhead of the scheme is r/d and resilience increases with the overhead. With 100% overhead (i.e. $r = d$), performance improves with higher values of d (and r) and while delivery latency increases. FEC based approaches can recover from network losses; however they alone are not sufficient when overlays are used. Overlay nodes are the processes on regular end hosts and are more prone to failures than network routers.

In Kunichika's approach [25], all the nodes always keep one free out-degree by force, to accommodate a deprived node in case of failure. Any end hosts with an out-degree n caters only to $n-1$ children. Redundant structure of tree avoids exhaustive search of a backup parent and simplifies backup route calculation. Layers to which the backup route search reaches are limited at worst to the grandchild layer. The limitation of this approach is that it needs to maintain permanently redundant degrees at each node.

In probabilistic resilient multicast (PRM) enhanced NICE (Nice is the Internet Cooperative Environment) protocol, Banerjee *et. al.* [22] introduced multicast data recovery scheme with two components. A proactive component called randomized forwarding in which each overlay node chooses a constant number of other overlay nodes uniformly at random and forward data to each one of them with a low probability, and a reactive component called triggered NAKs to handle data losses due to link errors and network congestion.

Efforts toward reliability in structured ALM networks has been described in chapter 3 in detail. Splitstrem [35], one prominent reliability solution in this category has been described in section 3.4.

5.3 Our Approach Toward resilience

The work presented in this chapter, illustrates three approaches with their algorithms, towards resilience of live streaming traffic in an application layer multicast network. Our approaches and PRM [22] fall in the same category of redundant transmission based resilient approaches. But there are two main differences. Our approach builds a data forwarding topology based on P2P query network. The second difference is that we deterministically maintain dual tree at all times for dual feed to each node, while in PRM redundant feeding is done on randomly selected links. Thus we expect high reliability in our design. The basic aspects common to all three approaches are discussed in this section. Specific details of the approaches are discussed in the next section.

5.3.1 Location and search for the feed

In the proposed scheme, the distributed index using a Distributed Hash Table (DHT) maintains a list of presently active sources for the two feeds f_1 and f_2 of the same video stream. If there are more than one live streams, each one of them will have two feeds. For example for stream 1, the two feeds are denoted as $1 - f_1$ and $1 - f_2$; for stream 2, these will be $2 - f_1$ and $2 - f_2$. The nodes which are willing to act as forwarders for a feed, publish their advertisement in the distributed index in the P2P query network. Using the keyword describing a feed, a new node can find the potential forwarders for a feed from this distributed query network. For example, all the nodes willing to forward $1 - f_1$ can be searched by the keyword $1 - f_1$. For maintaining distributed index, the query network can be built using any of the distributed Peer-to-Peer lookup protocol viz. Chord, Tapestry, Pastry etc. The distributed index maintains a list of some minimum number of active forwarders (say 5 or 6) for each of the two feeds. For a moderate growth rate of network, this number is sufficient. Any query is provided with the best source for each feed. The selection for the best can be based on the parameter such as:

(i) differential delay between the two feeds ($1 - f_1$ and $1 - f_2$) of the same stream should be as minimum as possible,

(ii) the distance of the forwarders from the source which also should be as minimum as possible, and

(iii) the feed forwarding capacity of the nodes. Nodes having higher capacity should connect higher up in the tree.

The node directly makes connection with the selected forwarders after getting the response. Every node gets a feed each from one of the f_1 as well as one of f_2 forwarders. It also puts up advertisement for being potential forwarders either for f_1 or f_2. The nodes which are already forwarding to sufficient number of nodes can remove their advertisements to avoid overloading.

5.3.2 How a search query progresses and replied

Whenever a node declares itself as a source of a particular feed, corresponding feed-URI (Universal Resource Identifier) pair is published in the Distributed Hash Table (DHT). Thus list of available stream servers for different stream feeds is stored as a list of feed-URI pairs in the DHT. The feed-URI pairs for a feed can be stored (retrieved) at (from) the root node responsible for that feed. Any media stream in the network is searched in the distributed index and corresponding URI of potential sources of this stream are retrieved. The node then makes a direct connection with one of the feed sources and gets the feed.

While a request moves, in the DHT query network, each node knows the address of the node that has passed the query to him. Ultimately, when the query reaches the node where index is maintained (root node for the keyword), the response is sent back to node which passed the query. The response thus travels back to the origin of the query. Each of the intermediate nodes keeps the response in the cache for a defined time. In case a node finds the response in the cache on receipt of a query, the cached response is sent back.

5.3.3 Node Design

A node receiving dual feeds of a stream should use buffering to take care of differential delay in the duplicate data received from two feeds to avoid interruption during play back.

Two feeds will be giving packets with different delays most of the time. In case of failure, and hence interruption of one feed, the playback of the data feed should remain uninterrupted for user. The stream should be buffered before being given to media player so that we have some packets to display when a feed get failed and hence ensuring interruption free playback.

Figure 5.5 and Table 5.1 describe an example scenario. Buffering ensures that even when a feed gets interrupted, the user gets the playback of multicast transmission without interruption.

time / packet numbers	from faster feed	from slower feed	player is playing	packets stored in the buffer			
t=0	70	66	65	66	67	68	69
t=-1	69	65	64	65	66	67	68
t=-2	68	64	63	64	65	66	67
t=-3	67	63	62	63	64	65	66
t=-4	66	62	61	62	63	64	65
t=-5	65	61	60	61	62	63	64

Table 5.1: An example to show the working of buffer at a node

5.3.4 Join Process

Any new node that wants to get the feed, will find the list of peers from a distributed P2P query network who are willing to provide the feed; with their

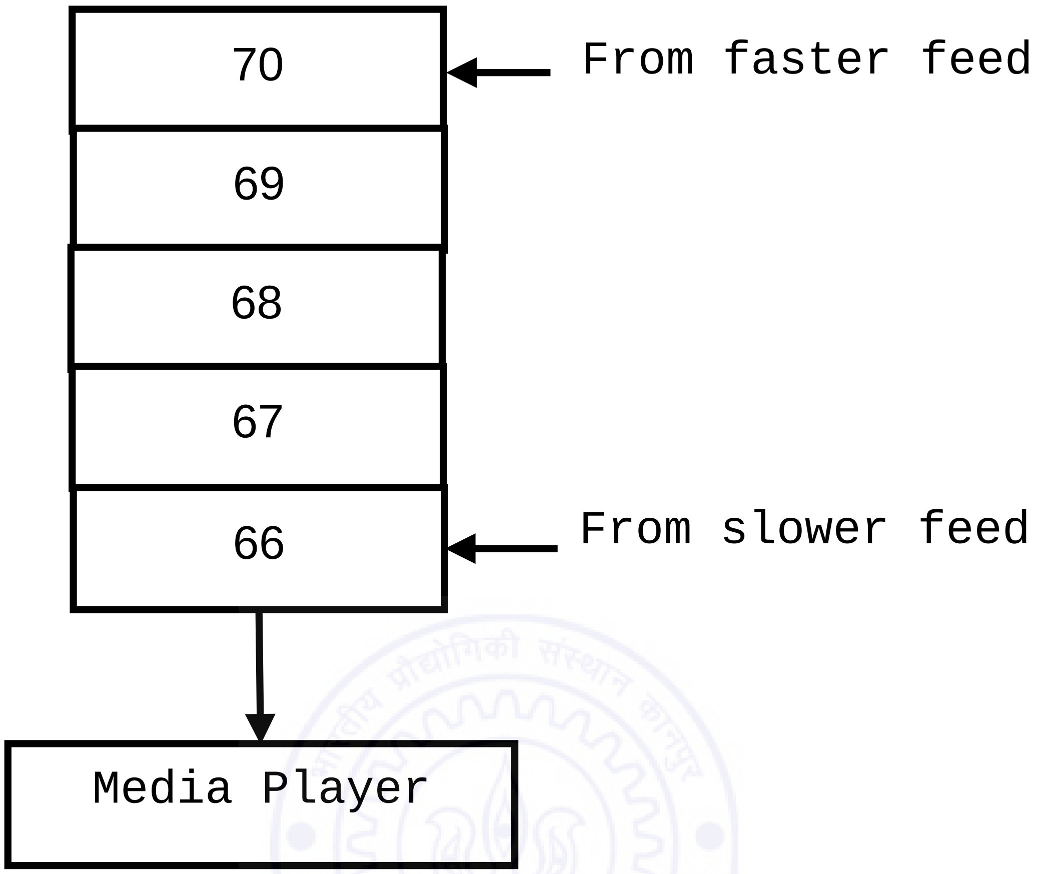

Figure 5.5: Single buffer created for two feeds from which media player plays the stream

parent nodes indicated. The query network that does the job of an indexing server keeps record of these peers in a table format as shown in Table 5.2. Feed id in the first column is the keyword for indexing.

The new node connects with the one forwarder from each set viz. forwarder set for f_1 and forwarder set for f_2. It also registers itself as forwarder for one of the feeds. Though the node receives feed in duplicate, but it becomes forwarder for only one type of feed i.e. either f_1 or for f_2. It also keeps the record of its grandparent for each feed. This information can be used to heal the distribution tree in case the feed from a parent stops coming. Source

Feed ID ($f_1 or f_2$)	Peer ID	Degree	Parent node ID

Table 5.2: Record of peers at an indexing server

node is an exception; it is source for both types of feeds (f_1 and f_2) of the stream.

5.3.5 Departure of node from multicast tree

When a member leaves gracefully, it informs all of its children in advance and these children in turn inform further to their children and so on. In this way the whole subtree rooted at the leaving node, is informed about the departure of leaving member. Any node in this subtree, who may have advertised itself as source of feed in the query network, will un-publish the advertisement. Now the nodes who are informed of the leaving node and who does not have any feed source i.e. next immediate children of leaving node will search for the new feed source and attach to them. Once the attachment is done, each such node can inform the same to subtree rooted at them permitting them to republish the advertisement of being a potential feed source. This ensures that the deprived nodes attach to some node which is not part of their own subtree, thus avoiding loop formation.

The abrupt departure of any member may cause interruption in playback if sufficient number of packets has not been buffered since it may take time to notify the failure to each node in subtree and to resume the removed feed from an alternate source.

An easier solution in this case is to get the lost feed from the departing node's parent. We recall that, during join process, when a node gets sources for f_1 and f_2, it also gets information about its grandparent. This grandparent node information can be used and feed-deprived nodes can request the feed from their grandparent. To avoid flash crowd, grandparent node then replaces failed node with one of its grandchildren selected on the basis described in the next paragraph. All other deprived siblings of selected grandchildren attach to some leaf node in the subtree of the selected grandchildren.

We define *cumulative children* of a node i (CC_i), as the total number of children nodes in the sub-tree rooted at this node. When a node fails, all its children get deprived of feed. The children with the maximum CC_i value may replace its failed parent by attaching to its grandparent. Remaining children of the failed node will get connected to the leaf nodes of the subtree of this child or the subtree of its uncle (failed parent's sibling nodes). With this solution applied, the reconstituted tree will be more balanced and the orphaned nodes would be able to get the feed deterministically.

5.3.6 Handling Loop Formation in the Event of Failure of a Node

In case a node in the multicast tree fails, it may interrupt data stream to downward nodes in either f_1 or f_2 tree (Approach 1 and 2) or in both (Approach-3). This node failure situation can be handled individually for f_1 or f_2 tree independently.

The information regarding this failure proceeds down the tree gradually and it may take time to inform the down most node about this failure. In the mean time the immediate children of the failed node may start searching for new source and it may happen that they choose some node in their own sub tree as their parent and thus creating a loop. To circumvent such loop formation, the following solutions are suggested.

Solution 1: In case of failure of a node (N), all of its children i.e. immediate children (N_{g1} nodes), grand children (N_{g2} nodes) and all the nodes in the sub tree rooted at failed node explicitly un-publish their advertisement for being a possible source from the query network. And once N_{g1} nodes search and connect to a new source and data streaming resumes, thereafter the N_{g1} nodes and the nodes in the lower level of the sub tree can again publish advertisement of being a possible forwarder.

Solution 2: In case of failure of a node (N), all its children i.e. immediate children (N_{g1} nodes), grand children (N_{g2} nodes) and eventually all the nodes in the sub tree rooted at failed node will stop getting the feed. Before this failure occurs, each of the node in the sub-tree rooted at failed node (N) were registered as potential source for the feed and after failure of root-node of this sub tree, all these get deprived of the feed. First of all, the Ng1 nodes detect that their parent has failed and they start searching for a new source. To avoid the possibility that any node down in the sub tree, which was registered as potential source; become new parent of Ng1 nodes, immediately after detecting the failure, all the N_{g1} nodes forward a message to their immediate children with an instruction to forward it further downward unless it is a leaf node. This message contains instruction not to respond to feed request from any nodes for the transition time $t_{transition}$ (time required to heal up the failure) duration. Thus this message gradually goes downward in the sub tree up to leaf nodes. Thus in this solution, if any node in the sub tree of failed node is in the DHT index, which could act as a possible source, is retained in the DHT but an INE (ineligible) tag is applied for estimated transition duration (say 30 units of time) to make them ineligible to act as a source for that duration. This will avoid chances of looping as N_{g1} nodes cannot select inactive sources.

With such remedy applied, even if a node in the sub tree of the failed node has its advertisement as forwarder, on getting request, it will not respond for the $t_{transition}$ duration.

Solution 3: This solution assumes that the distribution tree is binary. The two children of any node are named as left (L) and right (R) children. Each node maintains a string **path2src** for each feed which captures the node's position in that feed tree. String length is the maximum possible levels in the tree for a given number of total nodes. Each character, **path2src**[**i**], represents a level (starting from 1) in the tree and can have 3 values: U, L and R. It is assumed here that the source is at level 0 and at every hop from source, level increases by 1. The value U represents that the node is positioned at a level less than i. The values L (R) represents that the node can be reached by moving along the left (right) node in level i. For example the nodes 4, 11 and 2 in figure 5.11 in Approach-3 would be represented by LRUU, RLLU and RUUU respectively; assuming maximum 4 levels are there in the tree. We can use this information to deny connection request from all those nodes that are above the node in the feed tree.

5.3.7 Data Feed Management example

Our basis for reliable multicast is to maintain double feed distribution with node disjoint paths to each node. We discuss here the three possible approaches for overlay creation in which double feed distribution can be done, along with their merits and limitations. A comparative analysis is done on the basis of average latency in the network, average differential delay between the two feeds a node is fed with, and the out degree requirement of nodes. Out degree of nodes in a tree plays a crucial role to control the tree depth and hence the average latency.

The source of media stream S, in each case, transmits the same content as two duplicate feeds i.e. f_1 and f_2. Two separate trees are maintained for the two feeds and each node joins both the trees. Thus availing two different feeds from node disjoint paths, as evident in the respective structure shown in figures 2, 3 and 4 drawn for each approach. In the figures, f_1 feed tree links are shown as bold lines, while f_2 tree links are shown as dashed lines. It is possible that in one tree a node is a leaf node while in another tree it is at some intermediate position. In each approach, there are two kinds of nodes, one which will forward only f_1 version of the stream and the other one which will forward only f_2 version.

In each approach, a node forwards the information about its parents for both the feeds along with the distance from the source to all of its immediate children when they join it as their parent. This information can be used in the situation of abrupt failure.

5.3.8 Delay optimization in data distribution tree

With time, the data distribution tree may lose uniform distribution of load i.e. some portions of the tree may be denser as compared to others due to which some nodes might be facing inordinate latency, while there may be free out-degree in some other parts of the tree. Following periodic optimization algorithm tries to minimize the average latency in the whole tree.

To reduce average latency in the overall tree, nodes at lower level should be shifted toward root if there is a free out degree in an upper level node. At any level, all the nodes in that level will periodically tell their value of CC_i , the cumulative children of a node i (defined above in section 5.3.5) to their parent. Parent node on the basis of values obtained from all of its children, decides about modification in the tree structure.

Filling free out-degree with a grandchild: If a node has free out-degree, it is filled with one of its grandchildren, who is selected as follows. The node with free out-degree requests all of its children to send the node ID and value of their child with highest value. The grandchild with highest value is selected out of this set and offered to fill that free out-degree with its grandparent.

In case, a node does not have any children and hence grandchildren, its free degree can be filled by the grandchildren of its sibling in same manner.

CC_i **value based parent-child swapping:** If a node does not have free out-degree, this node can exchange its position with one of its children under certain conditions. A node i becomes eligible to replace its parent if it satisfies the following condition.

$$CC_i \quad > \quad (\sum_{\text{j= all siblings of i}} CC_j \quad + \quad 1) \quad + \quad 1 \tag{5.1}$$

The above inequality is self-explanatory and guarantees reduction is average delay in the network. As a child node swaps its position with parent, its parent and all its siblings along with their children get an increment in delay while all of its own children get a reduction in delay. This swapping will be advantageous if the decrease in delay is more than the increase. The inequality shows that the delay will reduce because at the left hand side it counts the number of children and at the right hand side it counts the number of siblings and their children plus the parent node. The tree organizing algorithm can be run periodically by nodes leading to optimization of performance.

5.4 The three approaches for dual feed

The detailed description of the three approaches for dual feed with specific detail is given in this section.

5.4.1 First Approach (Approach-1)

In the first approach, we assume that the out degree of each participating node is two except for the source node (S) which needs to have out degree three. In fact source or any of two earliest joining nodes can have out degree three to create a sensible overlay but here source is given this responsibility as it seems logical that the source has some extra capacity than other nodes. The overlay structure for the scheme is shown in figures 5.6, 5.7 and 5.8 . Source can use these three out degrees in delivering at least one f_1 and one f_2 feed; the third one could be either f_1 or f_2. Except source, any other node forwards only one type of stream (either f_1 or f_2).

Join Algorithm: The first two nodes *i.e.* node 1 and node 2 get feed f_1 and feed f_2 respectively directly from the source. The first node publishes itself as f_1 feed forwarder while the second node publishes itself as f_2 forwarder in the query network. Now both these nodes get complimentary feed from each other thus resulting in dual feed from source. Any further arriving nodes search for forwarders in the query network. Once a node with free out degree (one each in f_1 tree and in f_2 tree) is found, join request to the identified node is sent and connection is made. If more than one feed for the

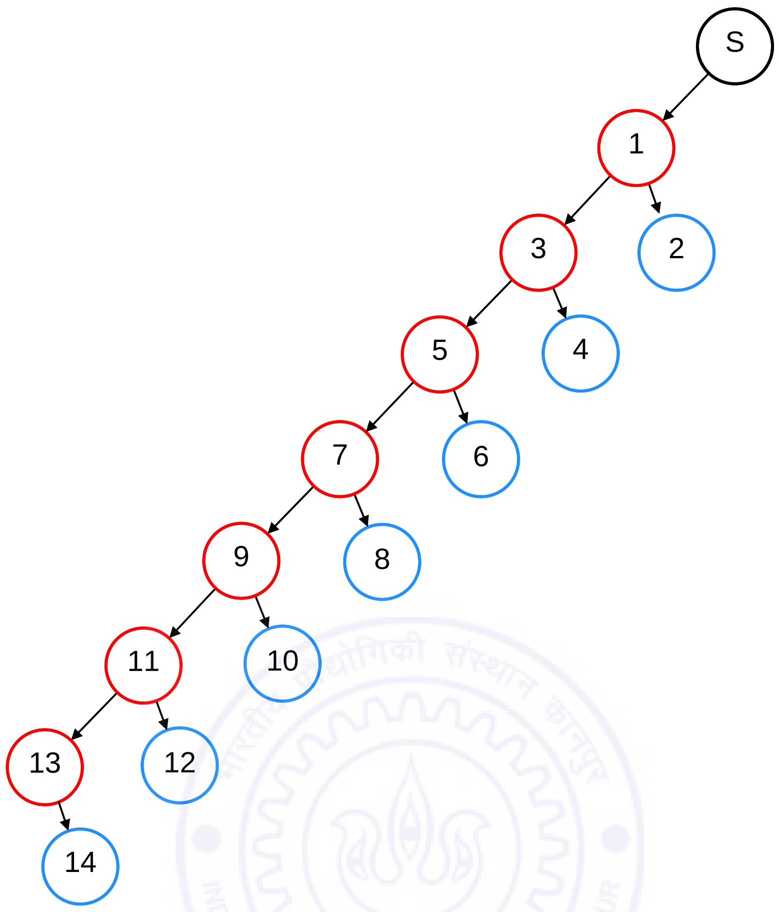

Figure 5.6: Original f_1 tree for approach-1 for example double feed distribution with 14 nodes before any failure occurs

same feed type are available, the one nearest to the source is preferred given it causes least differential delay in the two streams. The overlay tree thus gradually builds up. An example of double feed distribution with 14 nodes (including source) is shown in figure 5.6, 5.7 and 5.8.

Performance: In this approach, differential delay of zero or one is achieved at each node as is visible in figure 5.8. This is achieved by placing some restrictions during tree construction such that each node is placed at the levels

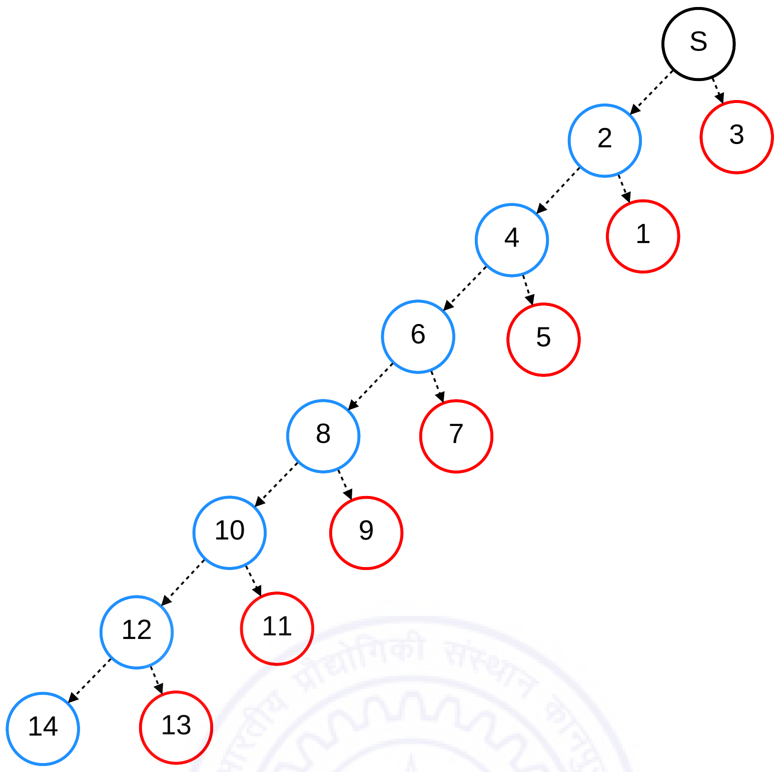

Figure 5.7: Original f_2 tree for approach-1 for example double feed distribution with 14 nodes before any failure occurs

differing by at most one in both the streaming trees and thus forming a deterministic topology. The limitation with this approach is that the average latency increases linearly with the number of nodes.

Failure recovery: To explain the recovery mechanism, the steps taken are listed below with reference to an example network of 14 nodes as shown in figure 5.8.

1. Let us assume that an f_1 forwarder node placed at some intermediate position fails. It leaves a forwarder vacant position (fvp) in f_1 tree and leaf vacant position (lvp) in f_2 tree. The respective parent nodes in

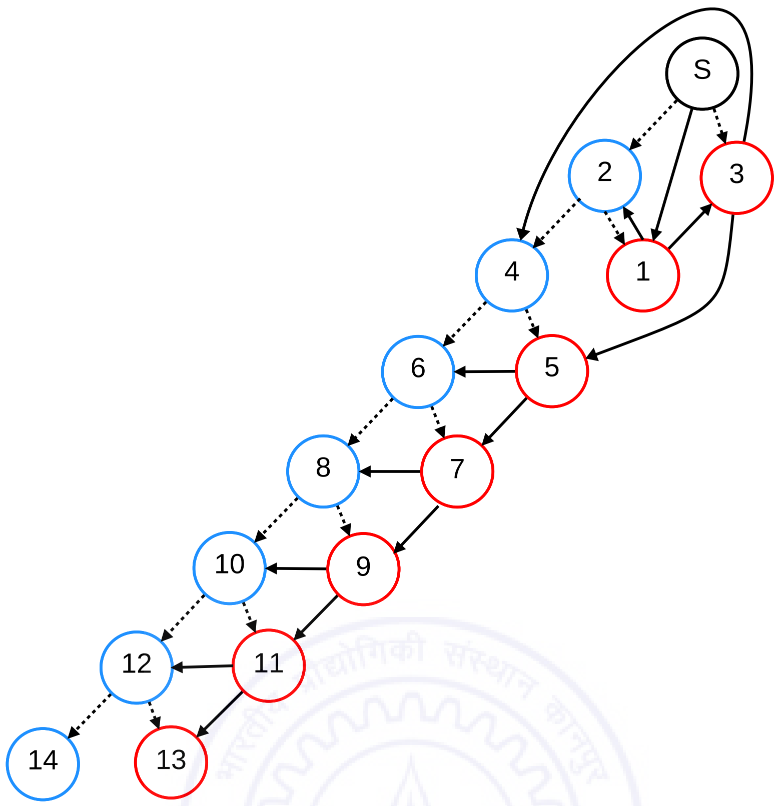

Figure 5.8: Original f_1 tree and f_2 tree put together for approach-1 for example double feed distribution with 14 nodes before any failure occurs

both trees which have gained free out degree due to this node failure, advertise for the free out degree in the query network.

2. **Changes in f_1 tree:** Failed node had two children in f_1 tree, one leaf child node (lcn) and one forwarder child node (fcn). These are orphaned and need to be replaced in f_1 tree. In f_1 tree, 'fcn' places itself at its failed parent's position with its whole subtree. The 'lcn' node finds its parent in f_1 tree through a fresh search in the query network.

3. **Changes in f_2 tree:** Since the failed node was a leaf node in f_2 tree,

it does not affect the transmission to other nodes and no change is required in f_2 tree.

4. **Tracking of differential delay:** Each node constantly observes the difference of delay in its two feeds. If it exceeds beyond a certain threshold, it tries to adjust its position in one or both trees by switching to a new forwarder either in one or both trees based on the information available in the query network. During this adjustment, if node leaves its present position in a tree, it is considered as vacant position and is advertised in the query network. Affected nodes get appropriate placement as per steps 1 to 3.

5. **Multiple Node failure of same type in sequence:** It may happen that the nodes of the same forwarding type fail in sequence resulting in shortage of forwarders of that type. The recovery mechanism has provision for change in forwarding behavior of nodes so that network never faces shortage of forwarding nodes of any type.

   Whenever a node that is orphaned in one tree (say in f_1 tree) but forwarder in other tree (say in f_2 tree) finds no feed forwarder for the feed type (say f_1) it is searching for, it converts his forwarding behavior (from f_2 forwarder to f_1 forwarder) and becomes forwarder for other feed type it was searching for. It replaces a leaf node at the bottom of the tree, places itself there and adopts that removed leaf node as its child. Immediately after that, it informs this conversion in the query network.

   Since after converting its forwarding behavior, it is no longer forwarding the earlier feed type (f_2), it removes itself from forwarding position in f_2 tree. Its ‘fcn’ and ‘lcn’ will find new parents as per steps 1 to 3.

6. Further complexity may arise, if failure happens at many places in the network simultaneously, and everywhere nodes see the deficiency of one type of forwarder and try to convert their forwarding behavior simultaneously. As a result of localized decisions taken at many nodes, a situation may be created where other extreme occurs. Suddenly one type of forwarders (which were earlier in deficiency) grows large in number and deficiency occurs for the other type and this cycle may keep repeating. To check this situation, a probabilistic approach is suggested below.

   With all other conditions fulfilled, a node to change its forwarding behaviour, waits for a random time $U[0, T]$, where T depends on the expected network size. Once this wait time is over it again checks if the deficiency of one type of nodes still exist or not, and acts accordingly. Instead of immediate change, it thinks twice and saves the network from the oscillatory situation as stated above.

5.4.2 Second Approach (Approach-2)

Join Algorithm: Figures 5.9 and 5.10 show an example network (binary tree) created as per join algorithm of approach 2. It assumes the presence of a central entity that applies a flag to all nodes indicating their type i.e. whether a node is f_1 forwarder or f_2 forwarder. It actually stamps every alternate node with f_1 type and f_2 type. Thus one half of the arriving nodes in the network are f_1 forwarder and the other half comprises f_2 forwarders.

Further in f_1 tree, f_1 forwarders are preferred and in f_2 tree, f_2 forwarders are preferred. This preferential placement puts the f_1 forwarders in f_1 tree as near to source as possible. The similar placement happens for f_2 forwarders in f_2 tree. The nodes can be swapped in a tree to maintain the preferential order. The topology builds gradually but eventually it takes a form that is deterministic for a given number of nodes. The final form for 14 nodes is shown in figure 5.9 and 5.10.

Due to this preferential placement, a node finds its position as leaf node in one tree and connects as intermediate node in another tree.

After getting both the feeds, a node places its advertisement in the query network as f_1 forwarder (or f_2 forwarder) according to its type already provided by central entity so that further arriving nodes can find the nodes with free out degree and request them to become their parent.

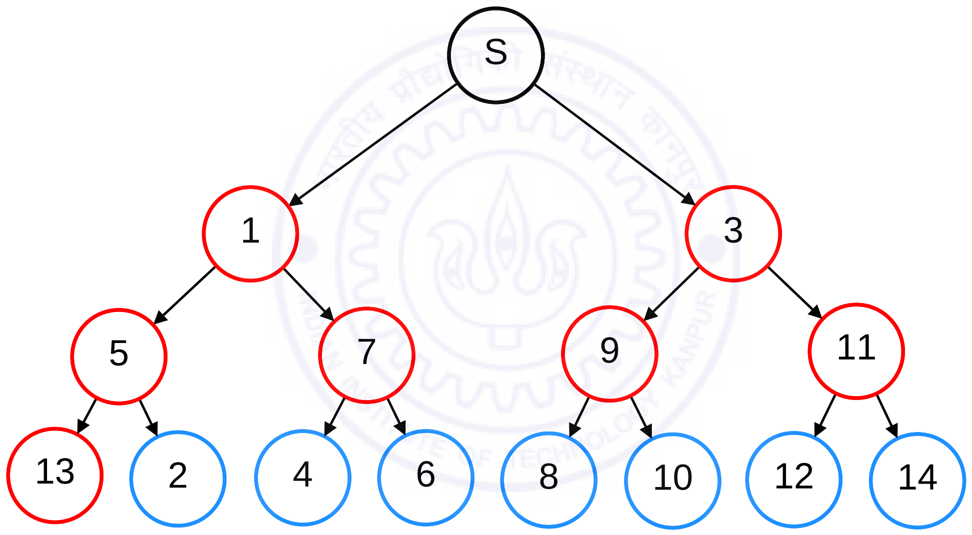

Figure 5.9: f_1 tree in approach-2 for example double feed distribution with 14 nodes

Performance: One merit of this scheme is that average latency increases only logarithmically with the number of nodes. The downside of this approach is that the structure formed is dynamic and the differential delay for most of the nodes will increase as the number of nodes increase. However, the increase happens to be logarithmically with the number of nodes.

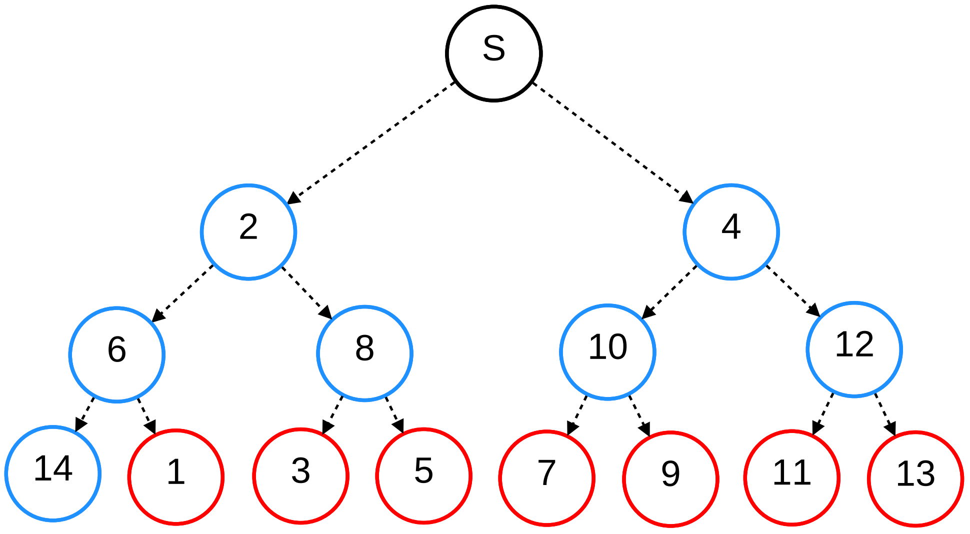

Figure 5.10: f_2 tree in approach-2 for example double feed distribution with 14 nodes

Failure recovery: Consider the failure of an intermediate node in f_1 tree. Whenever an intermediate node fails its two children are deprived. One child can place itself at failed node's position while the other one gets its new parent from the advertisements available in the query network assuming availability of an f_1 forwarder with free out degree. The loop avoidance algorithm as described in section 5.3.6 needs to be used.

Which one of the two deprived children will replace its parent is decided on the following basis. If one of them is f_1 forwarder and other is f_2 forwarder, the f_1 forwarder replaces its parent. If both are f_1 forwarders or both are f_2 forwarders, the child with higher CC_i value (cumulative children of a node i) replaces its parent.

Consider the situation when there is no f_1 forwarder available with free out degree in the query network when a node fails and two of its children are to be accommodated. In such situation, one of the forwarder nodes in f_2

tree with low CC_i value becomes f_1 forwarder. A leaf node which is nearest from source in f_1 tree and farthest from source in f_2 tree is preferred for this purpose. As soon a leaf node becomes forwarder in f_1 tree, it has to become a leaf node in f_2 tree. In doing so the children of that node in f_2 tree may get deprived. One of them however replaces its parent and the other gets parent from the query network advertisement.

5.4.3 Third Approach (Approach-3)

Join Algorithm: Figures 5.11 and 5.12 show an example network created as per join algorithm of approach 3. Any arriving node joins to both the trees. In f_1 tree, it attaches to a vacant position nearest to source. In f_2 tree, it attaches in the same layer to which it is attached in the f_1 tree. Once layer in f_2 tree is decided, it attaches to a node in that layer, which has nothing in common in its path to source in f_1 tree, which happens to be unique. The final form for 14 nodes is shown in figure 5.11 and 5.12.

Performance: The average latency increases only logarithmically with the number of nodes and the differential delay between two feeds to each node is zero. The only additional assumption it takes is that all the nodes excluding source node, need to have out degree 4.

5.5 Performance Evaluation

In the proposed scheme, a dual-tree is formed as data distribution topology. The two feeds are received at any node; one from f_1 tree and another one from f_2 tree. Both feeds will have node-disjoint paths from the source. Thus we need double the bandwidth in the current scheme of reliable media stream.

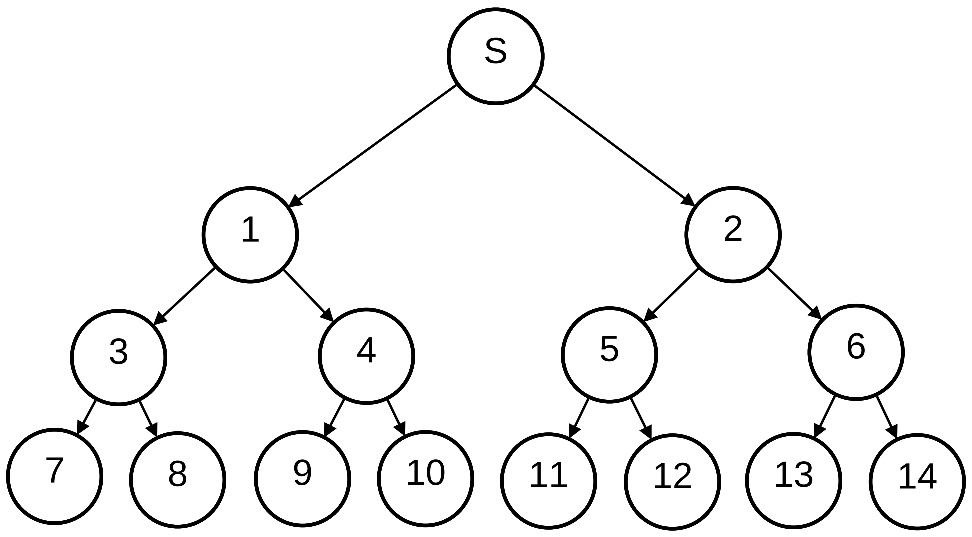

Figure 5.11: f_1 tree in approach-3 for example double feed distribution with 14 nodes

Since, two feed may have different delays, we need buffers at each node for merging the two feeds for uninterrupted playback in case one of the feeds fails. The size of buffer depends on the hop wise path difference in the two feeds.

The proposed scheme is based on P2P query network. As new host arrives and joins, it knows about available f_1 and f_2 feed sources from the query network. It connects to one of f_1 feed forwarders and one of f_2 feed forwarders. Thus the new host joining time is equal to the lookup time in the query network and the connection setup time. Similarly when a node fails, one of the feeds to the subtree rooted at this node will stop. As explained above, the recovery time has an upper bound which is equal to sum of DHT lookup time and connection setup time.

The three approaches discussed in section 5.4 has been compared in Table 5.3. In approach-1, average latency increases linearly while differential delay remains zero, as number of participating nodes increase. In approach-3,

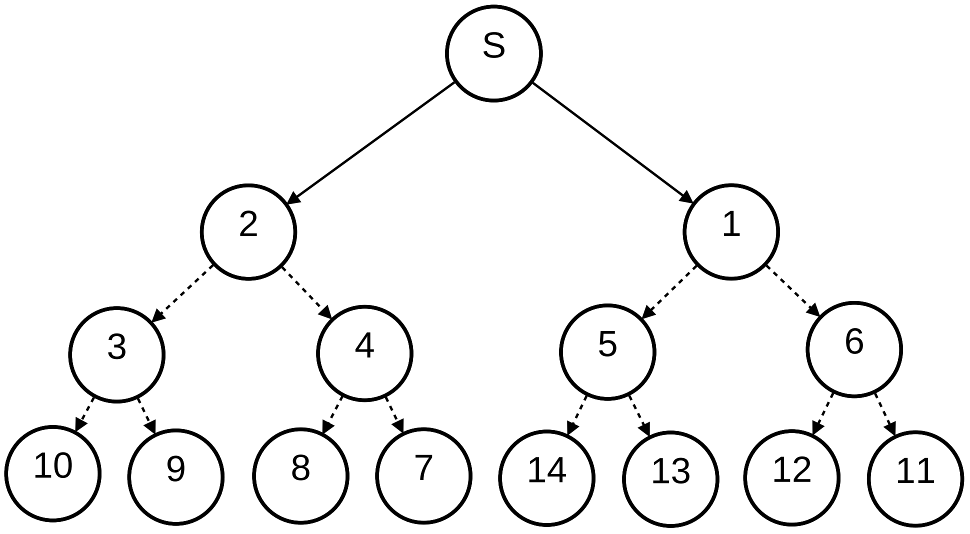

Figure 5.12: f_2 tree in approach-3 for example double feed distribution with 14 nodes

average latency increases only logarithmic and differential delay remains zero but out degree requirement for nodes is 4. The approach-2 seems to be most advantageous in which latency increases logarithmically, differential delay, though not zero but remains controllable and there is no need to have high out degree assumption as in approach-3. Therefore we do the simulation for approach 2 and do the analysis of the observations thus obtained.

The additional overhead in the scheme is the overhead to maintain a P2P query network to satisfy the requirement of a structured network. One can use Chord [32], Pastry, and Tapestry for P2P lookup. We assume here the availability of Chord protocol for query search. In Chord, in the steady state, for an N node network, each node maintains states for only about $O(logN)$ other nodes, and resolves all lookups via $O(logN)$ messages to other nodes. Updates to the routing information for nodes leaving and joining require only $O((logN)^2)$ messages. There is no need of maintaining an overlaid mesh and running multicast routing protocol to maintain a multicast tree in this

	Approach-1 (two trees, but no priority for any node)			**Approach-2** (two priority trees)			**Approach-3** (nodes placed at the same level in both trees		
number of nodes	number of levels	average latency	average differential delay	number of levels	average latency	average differential delay	number of levels	average latency	average differential delay
6	3	2	0	2	1.33	0.67	2	1.33	0
14	7	4	0	3	2.43	1.07	3	2.43	0
30	15	8	0	4	3.27	1.47	4	3.27	0
62	31	16	0	5	4.16	1.68	5	4.16	0
...	...	...	...	...	...	...	...	...	...
1022	511	256	0	9	8.02	1.96	9	8.02	0

Table 5.3: Comparison of the three different approaches used for data feed management

mesh. Thus the scheme is much more simplified than the earlier schemes of overlay mesh creation and management and then creation and management of multicast tree over this mesh.

We have evaluated the efficiency of our scheme based on following performance metrics: tree structure (comparison with perfectly balanced tree), distribution of differential delay, startupdelay, scalability and system behavior in the face of moderate to high failure rates.

5.5.1 Startup booster

If a node leaves the streaming overlay, it also causes disruptions in the underlying chord overlay. Hence, measurement of performance of multipath approach in face of failure will also include undesirable effects of the disruption caused in the chord overlay. To minimize this effect, some additional nodes are added to the chord overlay right in the beginning, which do not take part in streaming and only ensure Chord protocol's stability. These additional nodes are actually virtual nodes all staying at the source node. All these have same IP address as the source but distinct addresses (identifiers) on chord ring. The important point is that these virtual nodes are added in the beginning when there are no other nodes in the system. In this way, the addition of virtual nodes will not create any churn in the system. If the virtual nodes are added when there are some other nodes, then the virtual nodes will compete with real nodes for being added to the chord cycle. Thus, the performance would be reduced.

The number of virtual nodes must be sufficient enough to prevent the chord protocol from breaking down even when all real nodes have left. Also, it must not be too large to increase the chord diameter significantly. Ideally, it will depend on failure rate. As an example, let there be 1000 virtual nodes added in a system with 4000 real nodes. Now if 100 nodes fail, out of total 5000 nodes, 4900 nodes (98% nodes) are still intact, since 1000 virtual nodes remain unaffected due to failure. Even if 1000 nodes fail, out of total 5000 nodes, 4000 nodes (80% nodes) remain still intact. This proves the usefulness of virtual nodes introduced in the chord ring in the beginning.

These additional nodes will have the negative impact of increasing the chord diameter and hence the startup delay. But the positive impact of increased stability in the network outperforms the negative impact and we get an overall reduction in system scale and startup delay time. The additional nodes are only logical entities and they can be created by the source node to boost startup process.

5.5.2 Simulation results

PeerSim [47] [48], a java based event driven simulation engine, is used to run the simulation. The following parameters are used in the simulation.

N:	total number of nodes considered
d:	the peers' out-degree; taken as 2 as per approach-2
λ:	the peer arrival rate with uniform arrival pattern
μ:	the peer failure rate
D:	hop-difference between the feeds from f_1 tree and f_2 tree
B:	Size of buffer available at a node, should be taken so as to accommodate packets transmitted in the duration of maximum differential delay expected

The delay between the nodes is modeled by a uniform random variable between 45 and 55 milliseconds. To simulate the feed distribution, a feed message is pushed on each tree every 50 milliseconds. These feed messages are supposed to contain video information of 50 ms. Hence, by pushing them at every 50 ms, video playback at the nodes is maintained. As we do not consider overhead calculations, message size is irrelevant. Also, the size of the messages will depend upon the quality of video and the coding scheme used. Since, these are not covered in our simulation; size of the feed message does not matter.

Time is measured in time slots where one time slot denotes the time taken for a message transfer between two directly connected peers. The simulation is run for approach-2. Though there is no minimum number, at any time in our simulation, we have more than one active feed forwarder. The nodes register themselves when they have free out-degree and remove themselves from the list only when both their out-degrees are consumed. The recovery scheme as described in section 5.3.5, has been considered. For avoiding loop formation, solution-3 has been considered in our simulation.

5.5.2.1 Tree Structure

Tree structure (tree depth or maximum number of levels and average latency) is observed and compared with the optimal values i.e. with a completely balanced tree. Findings are listed in Table 5.4. The nodes arrive with the rate of 200 nodes/second for 10 seconds. The values are averaged over 10 experiments.

Number of Nodes	Max Level			Average Latency		
	Feed 1	Feed 2	Optimum	Feed 1	Feed 2	Optimum
2000	13	12.6	10	9.2	9.2	9.0

Table 5.4: Comparison of feed tree (approach-2) with perfectly balanced tree

5.5.2.2 Distribution of hop difference in the two feeds

Through simulation, we find the probability distribution curve of hop difference (differential delay) in the two feeds at all nodes, once all the nodes are connected. Differential delay decides the buffer size required at nodes. Distribution as shown in figure 5.13 indicates that around 70% of nodes have differential delay less than or equal to two, and less than 10% nodes have differential delay more than 4.

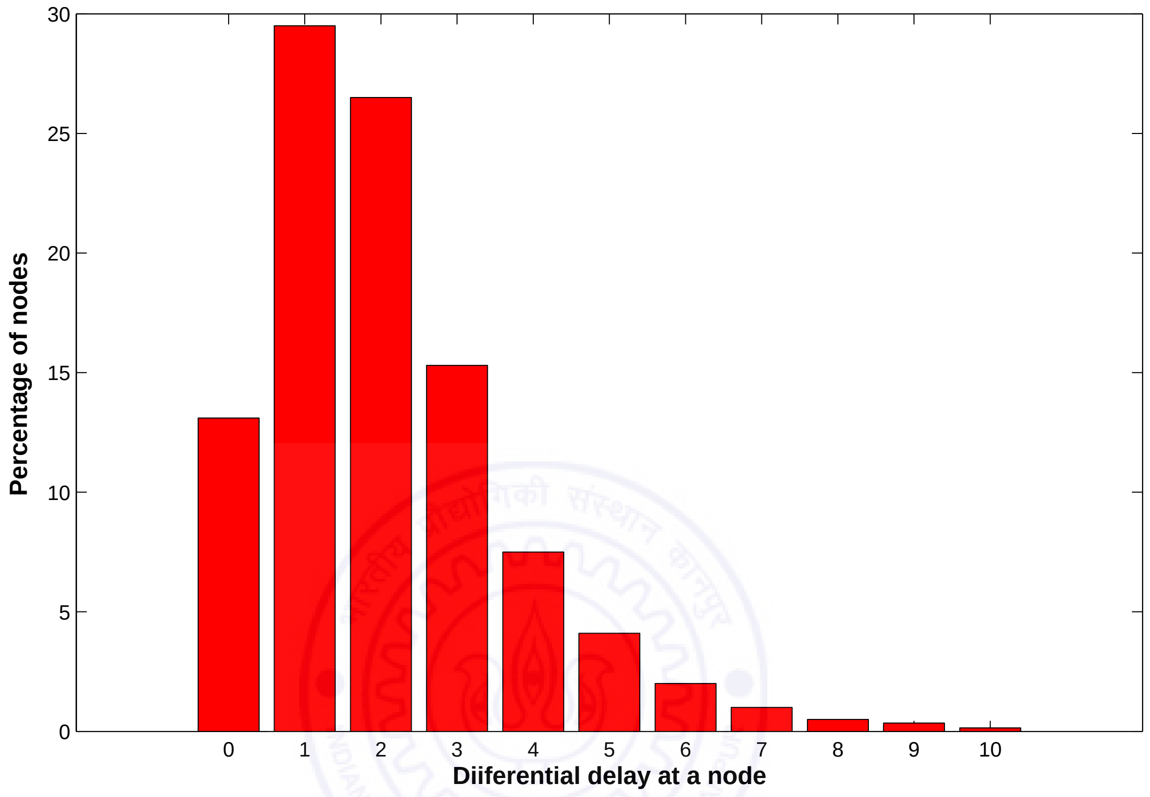

Figure 5.13: Distribution of differential delay

5.5.2.3 System scale and startup delay

In our algorithm, there may be situations when an arriving node does not get feeds within the given constraints. Any node deprived of any feed keep attempting periodically for that feed. We plot the system scale and the cumulative distribution of startup delay in figure 5.14 and 5.15 respectively. The *start-up delay* is defined as the time taken by the peer after its arrival, to get connected to a parent peer already receiving the stream. The nodes arrive with the rate of 200 nodes/second for 10 seconds. The values are averaged over 10 experiments.

As stated earlier, with startup booster process applied, the additional nodes though increase the chord diameter and hence the startup delay but support stability of the network. Thus overall, the delay performance improves with startup booster applied. The observations as shown in Figure 5.14 and 5.15 respectively confirm this fact.

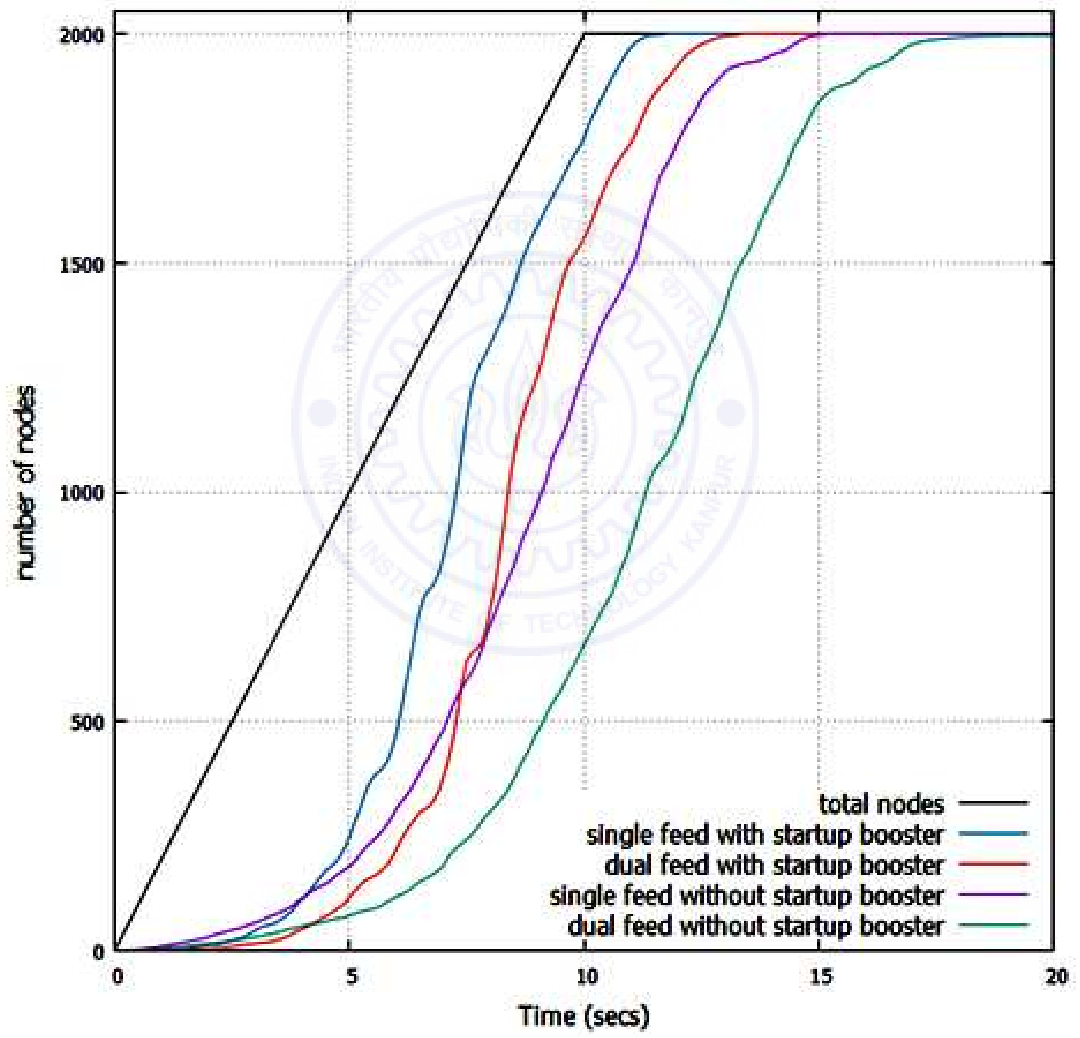

Figure 5.14: System scale with time

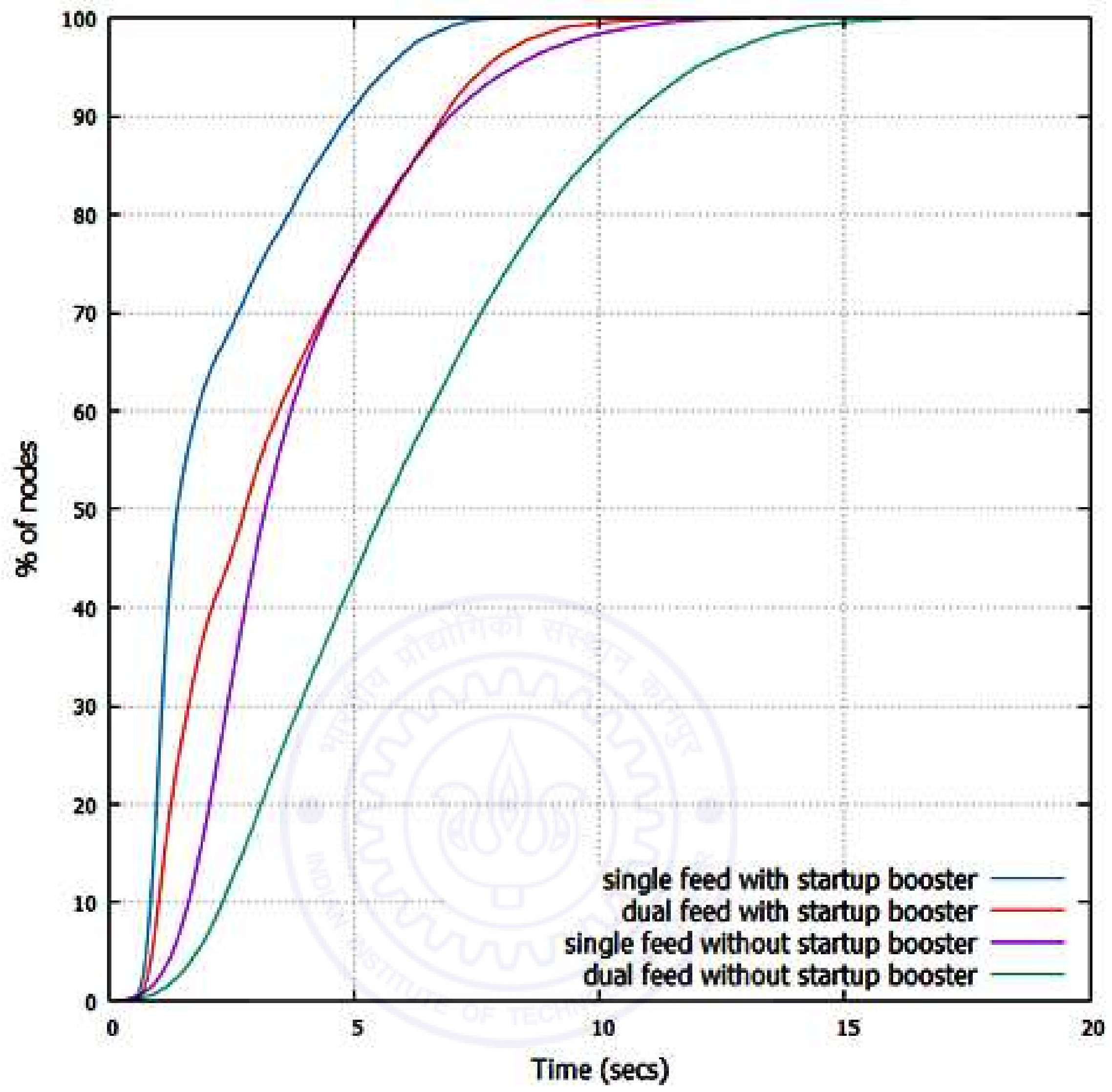

Figure 5.15: Cumulative distribution of startup delay

5.5.2.4 Network behavior in face of failure of nodes

Though in our scheme, each node is supplied with the two independent feeds, high failure rate may give interruption in the stream. The nodes arrive with the arrival rate of 200 nodes/second for 10 seconds. Now, starting with the 15th second, the nodes leave the network with the failure rate of μ nodes/sec for 10 seconds. The performance for different values of failure rate is plotted in Figure 5.16. Percentage of nodes receiving at least one feed in face of different failure rates is observed. Plots indicates that with failure rate of 50 nodes/per second in a network of 2000 nodes; 75% nodes still manage to get at least one feed.

Next we observe the behavior of network again in face of failure of nodes but with maintaining the moderate arrival rate. Plots in figure 5.17, 5.18 and 5.19 describe this scenario where percentage of nodes having at least one feed is plotted. Each figure has two plots. The first one shows the percentage with respect to the total number of nodes in the network including nodes which have just joined the network but have not started to receive the feed. The second plot finds the percentage with respect to total connected nodes without including newly joined nodes thus giving better picture.

In the beginning many nodes join the network at a fast rate as high as 200 nodes/sec for 10 seconds. After 10 seconds onwards, arrival does not stop and it continues at reduced value say 10/20/50 nodes per second respectively for figure 5.17, 5.18 and 5.19. Failure starts from 15th second onwards at the rate 10/20/50 nodes per seconds respectively for figure 5.17, 5.18 and 5.19 and it continues.

Finally, this is to mention that for each data point, multiple simulations have been performed untill the standard deviation is less than 5% of mean value. Thus for each point, the number of runs are different. Since confidence interval is small and uniform, it has not been shown in the results.

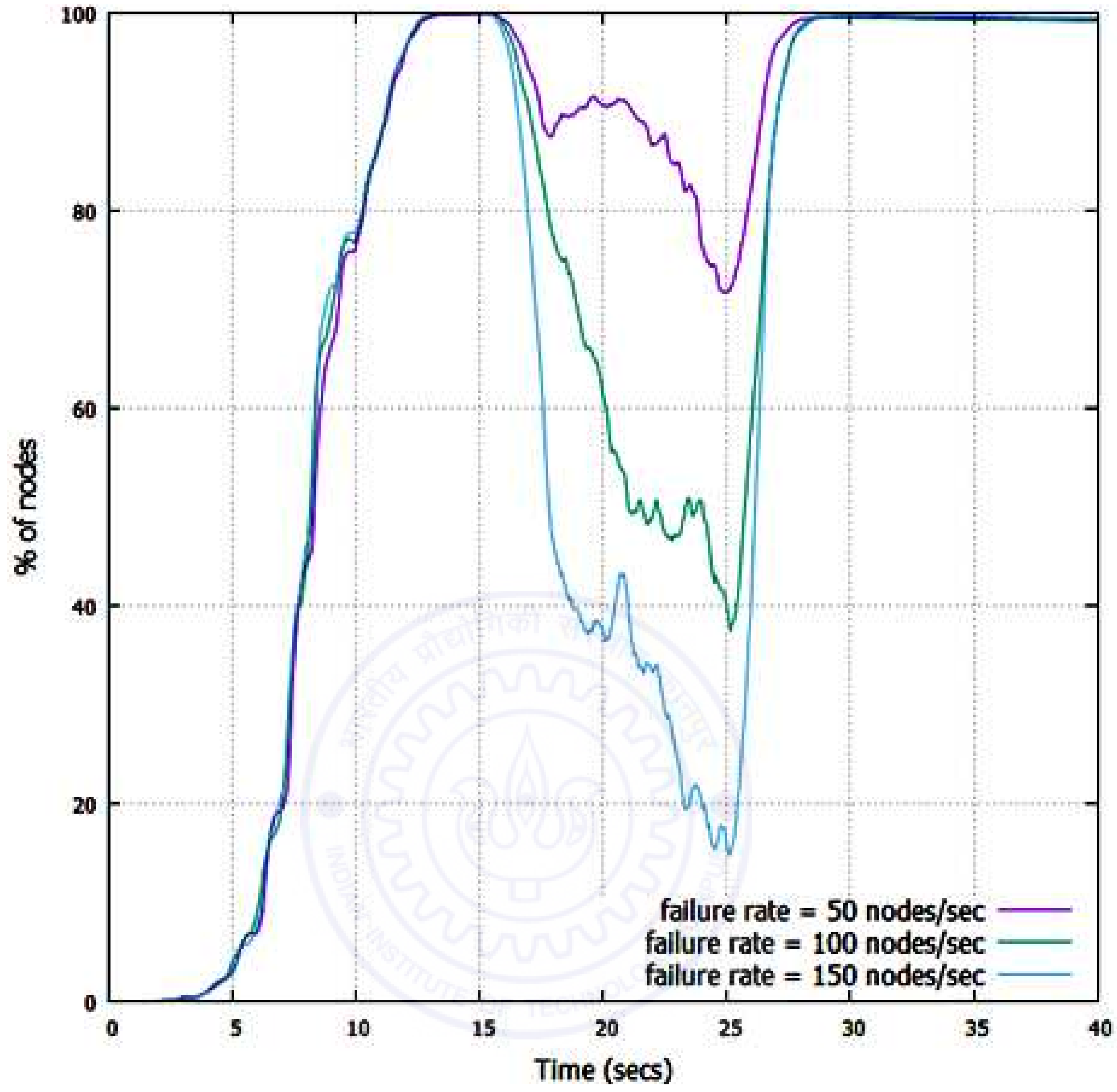

Figure 5.16: Percentage of nodes receiving at least one feed

5.6 Conclusion and future work

We have suggested an approach for maintaining dual-feed to increase the reliability in overlaid multicast streaming. The dual feeds are maintained while optimizing the delay. Some minimum outbound degree is needed for every node for this mechanism to work. We can make the system better if

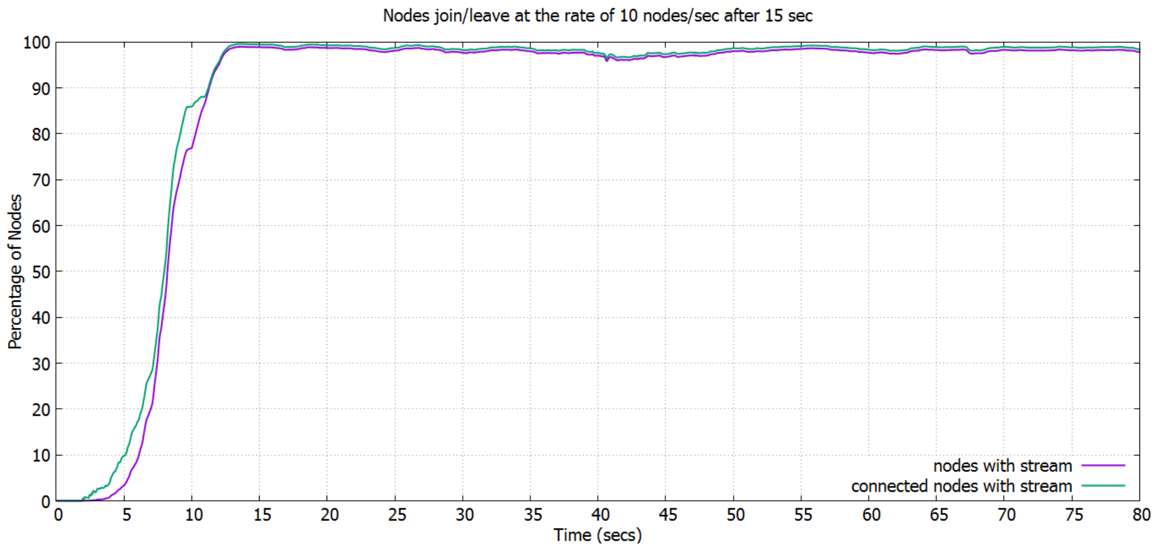

Figure 5.17: Percentage of nodes receiving at least one feed with continued failure and reduced arrival rate, rate = 10 nodes/second. The upper (green coloured) plot in the figure calculates the percentage with respect to total connected nodes excluding newly joined nodes in the network

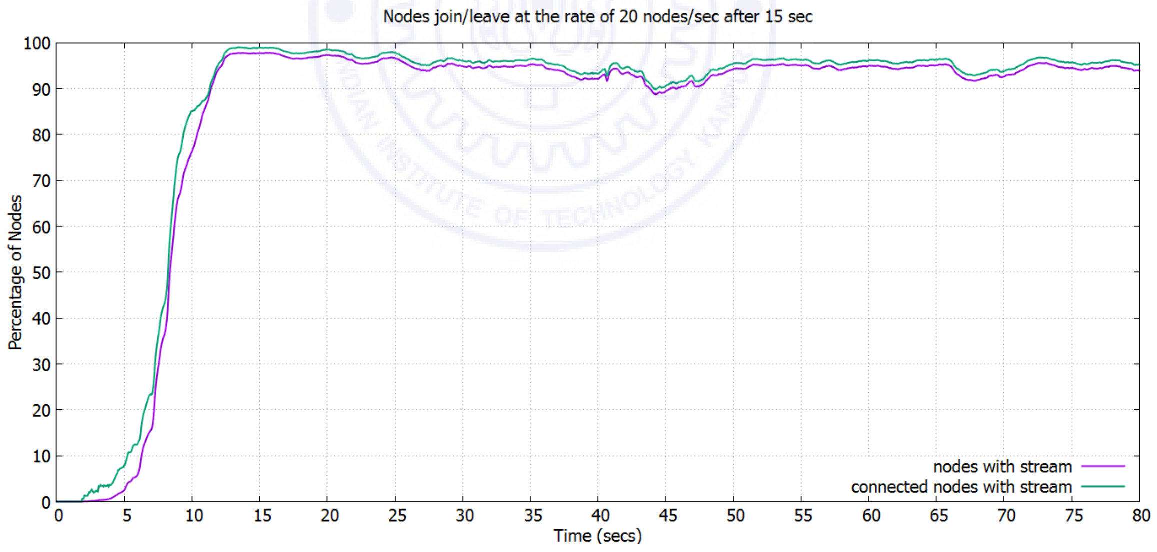

Figure 5.18: Percentage of nodes receiving at least one feed with continued failure and reduced arrival rate, rate = 20 nodes/second

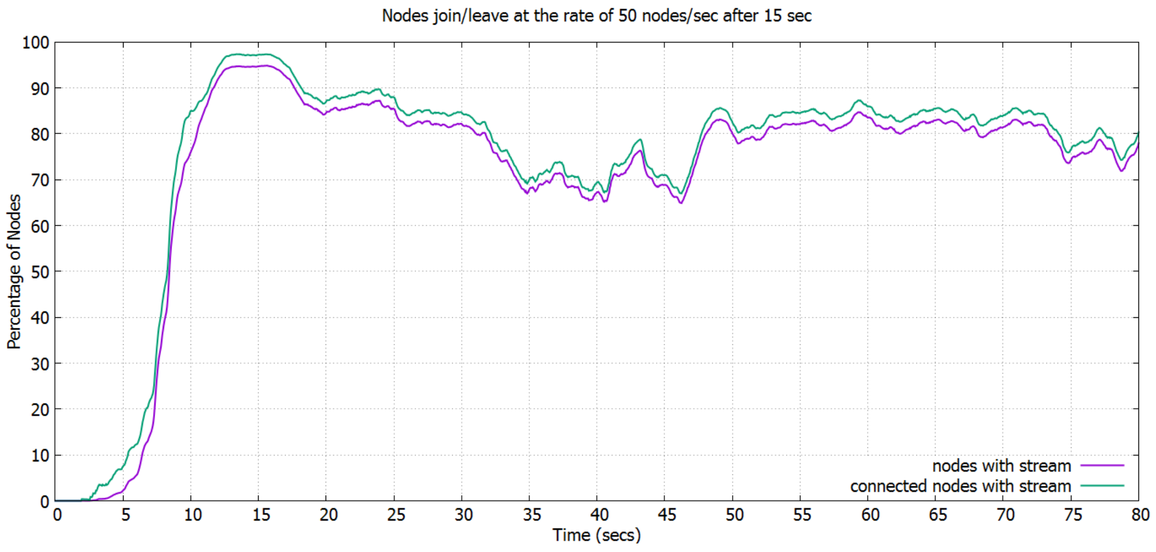

Figure 5.19: Percentage of nodes receiving at least one feed with continued failure and reduced arrival rate, rate = 50 nodes/second

each node advertises its distance from the root node. Thus, when a node gets multiple options to get the new feed, it can choose the parents based on the advertised hop count from the root node. If a node only wants to receive and does not transmit further, it is like free-riding. One needs to study the mechanisms to discourage this practice to ensure that the network functions without any problem.

The results presented here are the part of ongoing research to investigate mechanism for more reliable and scalable Brihaspati Sync [49] Live Lecture Delivery System (LLDS) development to cater to millions of users worldwide. The work presented here will be implemented in Brihaspati Sync in future.

Chapter 6

Faster Overlay Creation under High Growth Rate

Live Streaming Multicast Systems are needed for applications like Live Lecture Delivery. In these systems, a very large number of subscribers may join a session in a very small duration leading to a problem called flash crowd handling. The result of flash crowd is that most of the subscribers do not get the desired feed in first attempt and thus waiting time increases excessively. We consider this problem in query network based overlaid multicasting [36] systems, where the information about available feeders is stored distributedly in Distributed Hash Tables (DHTs). The DHT based look up service is used to efficiently locate the available feeders for a particular data stream. The feed is identified by its keyword which is hashed to find the corresponding root node. The root node maintains the list of forwarders for this feed. When a new node joins the data overlay, it becomes a potential feeder for further transmission and gets itself published in the list of available feeders.

At normal growth rate, maintaining a list of only few available feeders is sufficient, however during flash crowd, a list of large number of feed forwarders needs to be maintained. Two different algorithms are proposed in this chapter; one to alleviate the load of root nodes and the other to guarantee with high probability the provisioning of feed to any new arriving node even at very high growth rate. In the proposed scheme, there is a mechanism that the number of root nodes for the same keyword increases or decreases in tune with the growth rate. The simulation of proposed algorithm has been done for the Chord [32] based overlaid multicasting system with very high growth rate. Simulation results verify the effectiveness of these algorithms for the network in sustaining high growth rate in the network.

6.1 Introduction

P2P systems are distributed systems without any centralized command where all nodes are equivalent in functionality. The advantage in P2P systems as compared to client-server systems, lies with the fact that any single, arbitrary chosen terminal entity can be removed from the network without any loss of network service. This distributed nature makes them inherently scalable and fault tolerant. Among P2P applications, file sharing, video on demand and live media streaming are the most extensively studied and used. In live media streaming applications, delay deadlines are more stringent compared to file sharing. In file sharing system, a peer can wait for hours in retrieving a file, while in live streaming system, a newly arrived peer needs an immediate access to the stream. The total delay in live streaming application comprises the delay in finding an active peer for receiving the desired feed, the connection time and end-to-end delay along the overlay distribution tree.

In applications where an excessively large number of peers try for the same feed in a small duration, say in the beginning of a session, finding an appropriate feed forwarder may involve long delay and system may fail to accommodate such a large incoming crowd. Following are the two example scenarios where the growth rate of network is very high in the beginning. In Live Lecture Delivery (LLD) meant for very large number of expected subscribers, during first 10-15 minutes, in the beginning of lecture, a large number of peers may join the session. In football match telecast, the number can be as large as few hundred millions. The growth rate of the overlay network will be very high during this period. Chen Y. *et. al.* [50] characterized the severity of a flash crowd by the parameter called shock level, defined as the ratio of peers arriving rate during and before the flash crowd. The capacity of multicast system is defined as the maximum shock level that the system can survive [50].

Distributed Hash tables (DHTs) are query networks optimized for search task. In DHT based systems, the communication cost and the state maintained by each node scales logarithmically with the number of nodes. Therefore DHT based overlay networks are considered scalable and bandwidth efficient. We consider here a DHT query network based overlay multicasting system [36]. Chord [32] based multicast is an example of this. The Chord based look up service is used to efficiently locate the available feeders for a particular data stream. Chord supports the mapping from requested feed's key to a node where the URLs (uniform resource locators) for current feed forwarders are available. When a new node joins the multicast overlay, it becomes a potential feeder for further transmission and gets itself published in the list of available feeders at the root node of that feed. Root node for a feed is the node which has the same or most similar identifier as is the hash of feedID. The feedID is normally the unique string identifying the multicast feed. A list of few (say 5 or 6) available feeders needs to be maintained in the query network at normal growth rate. When the growth rate is very high in the beginning, rate of joining is limited by the number of feeder nodes listed in the query network and most of new nodes will not be able to get a data feed if very few feeders are maintained in the list. Moreover, since any newly joined node may publish itself as feed forwarder in the query network, it is logical to think that with high growth rate, the number of feed forwarders will also increase exponentially. In this situation, only a few nodes would need to maintain a large number of entries for a single feedID. These entries could be a list thousands of nodes who become potential feeders for a particular feed. Therefore the maintenance of an updated list of available feed forwarders is an important issue for supporting large growth rate of nodes.

We propose two different algorithms to handle high growth rate of nodes; these are:

(i) Push based cache update algorithm alongwith Purge message based stale information removal algorithm to alleviate the load of root nodes, and

(ii) Feed forwarders' list maintenance algorithm to guarantee with high probability the provision of feed to any newly arriving node under the condition of high growth rate.

Rest of the chapter is organized as follows. The next section gives a brief survey of related work. Section 3 describes the proposed algorithms. Simulation results for the proposed algorithms are presented in section 4. Finally conclusion is given in the section 5.

6.2 Related Work

Chen Z. *et. al.* [51] discovered the fundamental tradeoff between the time and scale. In their model for P2P live video streaming, they focused on the peer joining process during initial surge and concluded that the peer startup time increases with arrival rate and decreases with initial system capacity and peer uploading capacity. It is proved in their work that system scale was bounded by timing requirement.

Chen Y. *et. al.*'s work [50] is the pioneering effort to model and analyze the performance of P2P live streaming system under flash crowd. He used a fluid model to characterize a chunk based P2P live streaming system with admission control implemented at the tracker server. When a peer wants to get the feed, it first contacts a tracker server to register itself and obtains a neighbors' list. Media data is divided into small chunks. Peers exchange availability information of chunks they have fetched, and then requests chunks from neighbors accordingly. A newly arrived peer called startup peer can request multiple chunks in parallel but it does not upload chunks to others until it obtains a sufficient number of chunks (say η_0 chunks). Once it obtains these minimum number chunks, it becomes stable peer and can upload to other peers. A stable peer prefers to upload data to other stable peers so that transmission to existing peers is not affected. When a stable peer starts serving a startup peer, it does not accept the other startup peers' request until the served startup peer becomes stable peer. To evaluate the worst case system performance, Chen et. al. [50] assumed that the newly-arrived peers do not leave the system before they finish the startup phase.

It was found that a system without admission control had limited capacity to handle a flash crowd. System sustained at a new stable state when the size of flash crowd is small or moderate, but collapses when the flash crowd is excessively large. Chen proved that P2P live streaming system with admission control had excellent capacity to handle flash crowd and a startup peer's waiting time scales logarithmically with the size of flash crowd.

In [52], Dwivedi *et. al.* inferred that collaboration instead of competition among the peers is required to achieve better system scale. For the first time in [52], multiple tree based solution is designed to handle flash crowd in live streaming system. Earlier, in Splitstream [35], Castro et. al. used multiple tree based solution to handle issues related with resource under utilization and fault tolerance. Their two step approach first arranges the newly arrived peers in different levels of different sub stream trees and then peers are connected with each other to distribute the stream.

6.3 Proposed Algorithm

The problem of flash crowd can be handled at two fronts. In our first algorithm, earlier responses from the root nodes for a (key, value) pair are cached at intermediate nodes so that further queries for the same key can be handled in the DHT network by intermediate nodes only, and root node is not overwhelmed by queries. Further since a single root node cannot handle the publication of the list of thousands of feed forwarders, the second algorithm makes a number of sets of forwarders; each set is maintained at a different root node. The number of sets varies with the growth rate of the network in the proposed algorithm with at least one set being maintained all the time.

6.3.1 Distributed and scalable query handling algorithm

In the situation where a large number of subscribers try to join the session almost simultaneously and send query to the indexing server or to the query network to get the URL of an available feed forwarder, the indexing server may not be able to respond to all the queries simultaneously. The scheme to handle this problem is as follows. The response to each query is stored in caches at the intermediate nodes. Any other node querying for the same feed, if happens to follow a route that has some common nodes with the earlier query response route, it can get the information from the intermediate nodes without going up to the root node holding the index of feed forwarders. Thus the load on the indexing server can be greatly reduced. For a more popular key (feed), there will be more intermediate nodes that will be storing its index. This way scalability is supported, and unlimited number of queries for the same key can be handled. Though the cache based mechanism is a popular mechanism, we have added methods to update the cached responses and to purge the outdated information, as described below.

6.3.1.1 Push based Cache information updating

Whenever a new node joins the network and becomes eligible as feed forwarder for some feed, it publishes itself at the root node for that feed. In case the information is updated at the root node, the cache update message is sent to all downward nodes by the root node. For implementing such a system, root node maintains every entry in the list of feed forwarders with an associated timer. The timer allows purging the obsolete information. The list allows sending the update, after updating the timers, to the downward caches. We call it push based cache update.

6.3.1.2 Purge message based stale information removal

With time, the key-to-URL mapping information stored in intermediate nodes may become stale and useless. One possible improvement can be to store the cached information with successively reduced expiry period than what is stored in the previous node. Thus there remains no stale information and it expires after due time. Also nodes which are far away from root node, will never cache the information if the expiry time for them becomes zero.

It is also possible that the information becomes useless even before the expiry time is over e.g., when all the feed forwarder URLs for the desired feed as replied in earlier responses have exhausted and the current list at the root node now contain altogether different forwarder in the list. As and when, it is detected by some node down the tree, (which has been responded with the useless list by some intermediate node), that node should sends a *Purge* message towards root node informing that the cached list is stale and useless and thus leading to early purge of useless list. This message goes up to the root node. Then the root node can send the appropriate instruction to all the downward nodes which are maintaining the cache, to flush this information stored in cache. With the new requests and further fresh reply from the root node, the new list information populates the caches again at intermediate nodes.

6.3.2 Feed forwarders list maintenance algorithm

In situations, when network grows at a high rate, there is a need to maintain a list of large number of feed forwarders and it seems wiser to partition this single large list into different sets and maintaining them at different root

Algorithm 1 Actions taken by a newly arrived node to get feed from some already connected node in the network

```
1: Begin
2: Let $K = 0\ to\ K - 1$ be the set containing set numbers present in the network
3: Let $S =$ be the empty set in the beginning and records all set numbers that have been queried
4: while $K \neq \phi$ do
5:     Select set X randomly from K
6:     $S = S \cup X$
7:     $K = K–X$
8:     while forwarder list from (X) $\neq \phi$ do
9:         select a forwarder Y randomly from forwarder list from (X)
10:        try connecting to Y for getting feed
11:        if success then
12:            break
13:        else
14:            forwarder list from (X) = forwarder list from (X) $- Y$
15:        end if
16:    end while
17: end while
18: print ‘no forwarder available’
19: End
```

Algorithm 2 Actions taken by a newly connected node to get published as feed forwarder in the network

```
1: Begin
2: Let $K$ = 0 to $K - 1$ be the set containing set numbers present in the network
3: Let $S =$ be the empty set in the beginning and records all set numbers that have been queried
4: while $K \neq \phi$ do
5:   select a number X uniformly at random from {0, 1, 2, ....., K-1} - S
6:   $S = S \cup X$
7:   $K = K–X$
8:   if $X \neq 0$ then
9:     send register message to set# X
10:    if register message reply received then
11:      publish at set# X
12:      break
13:    else
14:      send register message to set# 0
15:      if register message reply received then
16:        Publish at set# 0
17:      else
18:        $k++$
19:        publish at newly generated set# K
20:      end if
21:    end if
22:  end if
23: end while
24: End
```

nodes. The need for more than one set of feed forwarders and their dynamics is discussed now.

6.3.2.1 Multiple sets of feed forwarders

In the proposed algorithm, we have more than one root nodes for the same feed and each one of them stores a different set of feed forwarders. We also store the count of total number of sets denoted as K at a single root node decided by the hash of string [feedID]_noofsets. At the onset of flash crowd, the value of K increases and afterwards it decreases at a fixed rate. It is crucial that the value of K is updated at all the intermediate nodes as soon as it is updated at root node ([feedID]_noofsets). To update the value of K, push based cache update is used as described in subsection 6.3.1. Whenever K changes at the root node ([feedID]_noofsets), this change is broadcasted down the tree.

6.3.2.2 Increase and decrease of number of sets

The number of required sets k (it is always greater than or equal to 1) is determined dynamically by the rate of arrival of subscribers at the root node of set#0. k may increase or decrease many times until the initial surge settles down and then it finally stays at one i.e. only the set#0 is maintained.

Any newly arrived node first hashes the [feedID]_noofsets and queries for the value of k from the root node of this hash. Let the value of k is found to be K. The node then randomly selects one of values from the range *0* to (*K-1*), say 'i'. The node now hashes [feedID]_set#i and finds root node for set#i. This root node maintains list of feed forwarders in set#i with corresponding URLs. The node seeks the list of feed forwarders from the root node and then connects to one of the feed forwarders from the received list. If request for connection is not accepted, it tries all other feed forwarders in this set one by one for connection. Eventually if there is no success with anyone in the list, it selects another set randomly from *0* to *(K-1)* and repeats the procedure the same way. The pseudo code for the join process is given in Algorithm - 1.

The newly joined node once starts receiving the feed from one of the feed forwarders, will find the latest value of k again by hashing [feedID]_noofsets and sending a query for k. Once the value of k is obtained (say K), it publishes itself at one of the randomly chosen sets (from range set#0 to set#(*K-1*)) with equal probability subject to availability of indexing space at the root node of the chosen set. In case there is no room at the randomly selected set, it again samples a number between *0* to *K-1* excluding earlier used set number(s) where no indexing space was found. This process continues till either a set is obtained with available indexing space or eventually set#*0* is sampled. This is to mention here that whenever a new node gets feed from any node in set#*0*, it necessarily publishes itself at set#*0*.

As a new node tries to publish itself at set#*0* indexing node (first root node) as feed forwarder for the feedID and if there is no room for it to be accommodated in set#*0*, the value of k is incremented by one, by the root node of set#*0* ([feeID]_set#0 node) by updating the published advertisement for [feedID]_noofsets. Thus a new set, set#K is created. The new node publishes itself as forwarder in the root node of this new set, set#K

While k increases as triggered by the root node ([feedID]_set#0 node), it is decreased by one unit at regular interval by the root node of [feedID]_noofsets. By doing so, it is ensured that k does not increase without limit, and achieves a value of *1* under steady state. In case, a node cannot get a free feed forwarder or is unable to publish itself in one of the set, it again chooses another set randomly and whole procedure is repeated. The pseudocode for the publish process is given in Algorithm - 2.

6.3.2.3 Removal of entries from a set

All the entries in sets except fixed number of entries in set#*0* are maintained with expiry time. An entry may expire even before its expiry time when all its fan-out capability is exhausted. Once all the feed forwarders' entries of a set are expired, that set also expires.

Certain numbers of entries in set#*0* are maintained without expiry, these entries are deleted only when their fan out capability exhausts. Moreover, as already mentioned that whenever a new node gets feed from any node in set#*0*, it necessarily publishes itself at set#*0*. Thus there is a guarantee that some minimum entries are always there in set#*0*. Thus, the value of [feedID]_noofsets always remain more than or equal to one.

When rate of joining of new nodes reduces to low value after initial burst, then all the sets except set#*0* will expire after some time and number of sets field will also adjust itself to unity. In set#*0*, there will be some predefined number of feed forwarders which will not expire unless their fan-out capability is consumed. All nodes over and above the minimum will have expiry after which they will be removed from the feed forwarders' list.

The set#*0* is special in many senses. This is the only set which always has listing of some minimum number of feed forwarders for the whole session time. As set#*0* root node is crucial, its backup is maintained at nearby nodes at 2-3 places so that the network does not fail if this root node fails.

6.3.3 Algorithms applied for an example application

Consider a Live Lecture Delivery system (LLDS), where we have different streams in the system for different course codes e.g. EE605, EE679 etc. with each having duplicate feeds (feed-1 and feed-2), which can be denoted as StreamID_FeedID (e.g. $EE605\_f_1$, $EE605\_f_2$, $EE679\_f_1$, $EE679\_f_2$ etc.).

We assume a typical Chord like distributed lookup protocol for query search. A new node, who wishes to join the overlay for a particular feed, hashes the keyword for that feed and gets the 256-bit identifier for that key and starts search for the root node for that key identifier. This root node along with few backup nodes in DHT holds the index of feed forwarders.

The key-to-feed forwarder node mapping (indexing) for a key will be available at the root node for that key. Whenever a query for a particular feed is resolved, it is responded with all the URLs where that feed is available. The joining node contacts its indexing server two times, once to get the list of available feed forwarders and again to publish itself as a potential forwarder for that feed.

Initially for a particular feed there exists only one set i.e. [feedX]_set#0, the only one root node for the feed, indexing all the available feed forwarders for this feed. At the start of a session, when a large number of subscribers for a course attempts to join almost simultaneously, the list of forwarders becomes too large to be accommodated at a single indexing node. To handle this situation the forwarder nodes are divided into different sets. In the changed scenario, the query for a particular feed then also contains the set ID e.g. $EE605\_f_1Set1$, $EE605\_f_1Set2$ etc. The first set (set#0) controls the creation and expiry of sets by updating the number of sets index entry.

6.4 Simulation Results

PeerSim [7] [8], a java based event driven simulation engine, is used to run the simulation. The following parameters are used in the simulation.

D: the fixed decay rate of number of sets (dk/dt),

d: the peers' feed-forwarding capacity,

Ssize: number of feed forwarders in a set (set size),

etime: feed forwarder entry expiry time,

λ: peer arrival rate, and

N: number of sets in steady state under continued arrival for sufficient long time

The link layer delay is modeled as a random variable uniformly distributed between 45 milliseconds to 55 milliseconds. Though the processing delay at each node is not zero, it is considered negligible as today's computers usually have very high processing capabilities. The routing delay is not modeled by the simulator as modeling these would prevent large scale simulations. It is considered that, all the nodes form Chord overlay for query network. Also, any node can directly send messages to any other node in the network if its IP address is known.

Rate	**d**	**Ssize = 5**		**Ssize = 10**		**Ssize = 20**		**Ssize = 30**		**Ssize = 40**		**Ssize = 50**	
		ST	**SD**	**ST**	**SD**	**ST**	**SD**	**ST**	**SD**	**ST**	**SD**	**ST**	**SD**
100	2	7.4	2.9	7.1	2.2	6	2	6.8	2.2	7	2.2	7.6	2.2
	4	5.3	1	5.3	0.9	5.3	1	5.3	0.9	5.3	0.9	5.3	0.9
	6	5.3	0.8	5.3	0.8	5.3	0.8	5.3	0.7	5.3	0.7	5.3	0.7
	8	5.3	0.7	5.3	0.7	5.3	0.7	5.3	0.7	5.3	0.7	5.3	0.7
200	2	9.6	4.3	8.5	3.6	6.9	2.8	7.2	2.9	8.2	3.2	9.4	3.3

Continued on next page

Table 6.1 – continued from previous page

Rate	d	Ssize = 5		Ssize = 10		Ssize = 20		Ssize = 30		Ssize = 40		Ssize = 50	
		ST	SD	ST	SD	ST	SD	ST	SD	ST	SD	ST	SD
	4	5.8	1.5	5.3	1.2	5.3	1.3	5.3	1.3	5.4	1.3	5.5	1.3
	6	5.3	1	5.3	1	5.3	1.1	5.3	1	5.3	1	5.4	0.9
	8	5.3	0.9	5.3	0.9	5.3	1	5.3	0.8	5.3	0.8	5.3	0.8
400	2	11.7	5.9	9.8	4.8	10.2	4.2	8.2	3.7	8.4	3.8	9.5	4.1
	4	6.7	2.4	5.7	1.6	5.4	1.7	5.8	1.7	6.2	1.8	6.4	1.9
	6	5.6	1.4	5.3	1.2	5.3	1.3	5.6	1.4	6.4	1.6	6.5	1.5
	8	5.4	1.1	5.3	1.1	5.3	1.2	5.6	1.2	5.6	1.1	5.8	1.2
1000	2	15.6	9.2	13.6	7.2	12.3	6.3	12	6	10.8	5.2	9.7	5
	4	8.2	3.7	8.1	3	5.6	2	6	2.2	7.1	2.4	7.9	2.5
	6	6.9	2.4	5.6	1.6	5.4	1.6	5.8	1.9	6.7	2.1	7.8	2.3
	8	6.4	1.9	5.4	1.3	5.4	1.5	5.8	1.7	6.7	1.9	6.9	1.9
2000	2	19.1	11.9	15.8	9.4	13.5	7.4	14.7	8	15.1	7.8	14.1	6.9
	4	10.6	5.2	8.8	4.1	7.4	2.8	6.4	2.6	7.2	2.8	8.4	3.1
	6	7.9	3.5	8	2.8	5.5	1.9	5.9	2.1	6.9	2.5	8	2.7
	8	7.1	2.7	5.8	1.7	5.4	1.7	5.8	1.9	6.9	2.2	7.9	2.5
4000	2	23.6	14.8	19	12.1	17.2	10	15.9	9.3	16.5	9.5	17	9.4
	4	12.9	7.2	11.4	5.4	10.8	4.6	7.9	3.4	7.5	3.3	8.6	3.8
	6	10	5	9	3.8	6.7	2.4	6	2.5	7.1	2.8	8.1	3
	8	8.4	3.7	8.2	2.9	5.5	1.9	5.9	2.1	7	2.5	8.1	2.8
10000	2	31.2	12.1	24.3	16.6	20.8	13.5	20.4	12.3	18.5	11.4	18.6	11.5
	4	17.3	10.5	14	7.9	12.3	6.4	13	6.2	12.6	5.4	8.7	4.3
	6	13.6	7.3	11.2	5.3	11.2	4.4	7.2	3.1	7.8	3.2	8.5	3.8
	8	11.1	5.6	9.4	4.4	7.2	2.6	6	2.5	7.1	2.8	8	2.9

Continued on next page

Table 6.1 – continued from previous page

Rate	d	Ssize = 5		Ssize = 10		Ssize = 20		Ssize = 30		Ssize = 40		Ssize = 50	
		ST	SD	ST	SD	ST	SD	ST	SD	ST	SD	ST	SD
20000	2	38.5	28	29.2	20.6	24.2	16.6	24	14.9	23.1	14.2	22.1	13.8
	4	21.3	14.1	16.4	10	13.8	7.6	14.4	7.6	16	7.7	15.1	7.3
	6	16.3	10	12.9	7	11.9	6	11.6	5.4	7.7	3.7	8.3	3.8
	8	13.6	7.8	12.1	5.6	11.2	4.5	7.2	3.1	7.2	3.2	8.2	3.5
40000	2	48.1	35.8	36.1	26.3	28.8	20.3	26.8	17.8	26.1	16.9	25.6	16.5
	4	26.3	18.1	20.1	13.1	16.2	9.7	15	8.7	15.6	8.9	15.3	9
	6	20.3	13	15.6	8.4	12.9	7.1	13.9	7	13.6	6.5	14.5	5.6
	8	17	10.5	13.5	7.5	11.8	6.1	11.9	5.4	10.2	4	18.5	3.9

Table 6.1: Stabilization Time (ST) and average Startup Delay (SD) in seconds at different arrival rates (nodes/second) for different out degree values and for different set size values

6.4.1 Estimation of optimum system parameters

The *stabilization time* is defined as the time taken for providing feed to more than 90% of the newly arrived peers. The *start-up delay* is defined as the time taken by the peer after its arrival, to get connected to a parent peer already receiving the stream.

Some parameters are kept fixed such as decay rate for sets, D = 1 set/second, and feed forwarder entry expiry time, *etime* = 10 seconds. The parameters *d* and *Ssize* are varied to capture the effect of their variation on the stabilization time and on the average startup delay.

Initially, the system contains a single set having a single feed forwarder entry. Nodes arrive with a uniform arrival rate (λ) for a period of 5 seconds. It was restricted to 5 seconds because even if we take this period slightly high say 10 seconds; the simulation takes a lot of time and does not contribute any new information in the result. Further, the feed forwarder entry expiry time would be irrelevant in face of high arrival rate since the entry would be removed after all of its out degrees are utilized even before it expires. The data obtained through simulation has been put in a table form as well as in 2-d plots. The arrival rate considered is as low as 100 nodes/second to as high as 40000 nodes/second. The corresponding values for stabilization time (ST) and average startup delay (ST) have been obtained for different values of out-degree (feed-forwarding capacity, d).

Stabilization time in our simulation has been observed as follows. We allow arrival for 5 seconds. We have a count of nodes that has arrived in these 5 seconds. Say with rate of 2000 nodes/second, 10000 nodes arrive in 5 second. Now we observe, at what time instant, 9000 nodes (90% of the total arrived nodes) get the feed. We start clock at the arrival of first node and stop the clock when 90% of the total arrived nodes have got the feed.

From the table as well as from the plots obtained, following can be inferred.

(i) At a fixed set size, higher out-degree reduces stabilization time and average startup delay (ST and SD) for sure but this reduction is more pronounced only at high arrival rates. At low arrival rates reduction is not significant. It depicts that increasing out degree gives more advantage at high arrival rates than at low arrival rates. The variation range for out degree considered is from 2 to 8 and the minimum ST as well as SD is obtained at 8.

(ii) At a fixed out degree value, higher value of set size reduces stabilization time and average startup delay in general, but there are exceptions where optimum is obtained before the highest set size value. We have considered here set sizes ranging from 5 to 50. At higher arrival rates, minimum is obtained at $Ssize = 50$, but at low rates, minimum is sometimes seen at $Ssize = 20$. It depicts that increasing set size any further at low arrival rate is not always advantageous.

Figure 6.1 and 6.2 plot the stabilization time for different values of out-degree at $Ssize = 5$ and 20 respectively as function of arrival rates. As seen from the figure, as we increase d from 2 to 4 and up to 8, the reduction in ST becomes less and less pronounced. An out degree value 4 can be considered as optimum as it gives the maximum advantage with the increase in arrival rate. Therefore $d = 4$ is taken for the rest of the simulations.

Figure 6.3 and 6.4 plot the stabilization time for different values of *Ssize* at out-degree d = 2 and 4 respectively as function of arrival rates. As seen from the figures, with increase in *Ssize* from 5 to 20 and then up to 40, the reduction in ST is not always guaranteed and at low arrival rates minimum is seen at $Ssize = 20$. Even at high arrival rate, though minimum is obtained at $Ssize = 50$, as seen from the table 6.4, but the advantage is not significant. Therefore $Ssize = 20$ is considered as optimum and this value is taken for the rest of the simulations.

Figure 6.5 and 6.6 plot the average startup delay against arrival rate once for varying out degree (*Ssize* constant) and then for varying *Ssize* (outdegree constant). These plots also have same nature of variation as in figures 6.1, 6.2, 6.3 and 6.4 for stabilization time. The out degree = 4 and set size = 20 again come as values with maximum advantage.

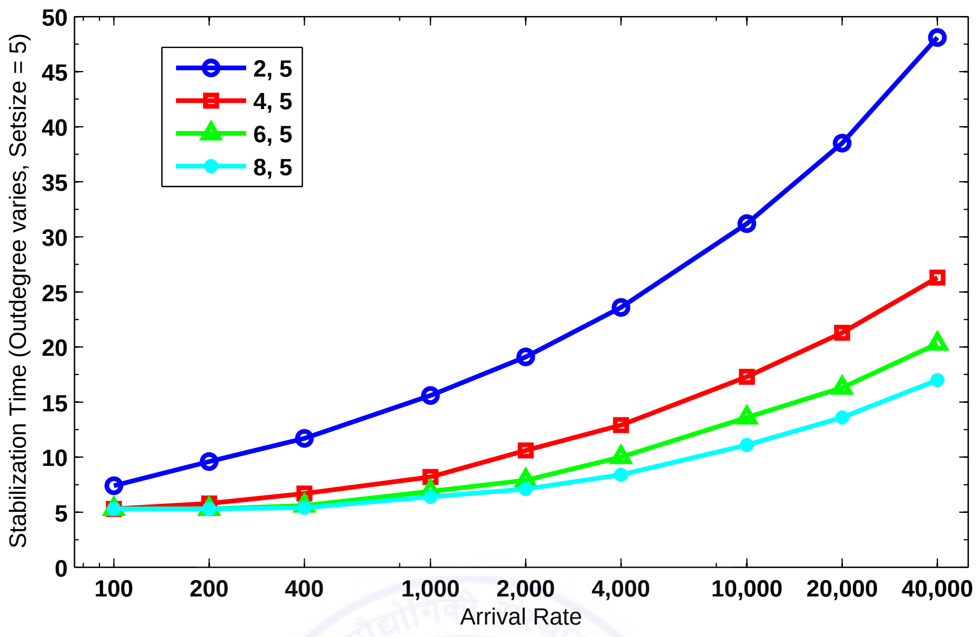

Figure 6.1: Effect of increasing arrival rate on the stabilization time for various out degree values for set size = 5

6.4.2 Flash crowd handling at different rates

The plots for stabilization time and average peer startup delay against arrival rate are obtained on a log scale in figures 6.1 to 6.4 and in figures 6.5 to 6.6 respectively. As seen from these figures (observing any single plot line at a time), when arrival rate is slow, the system stabilizes immediately after all the nodes have arrived. On increasing the rate further, the stabilization time increases, but the slope decreases for very high rates. Figures clearly reveal that the peer start-up delay as well as stabilization time varies logarithmically with the arrival rate. Hence, our algorithm is able to work for very high growth rates

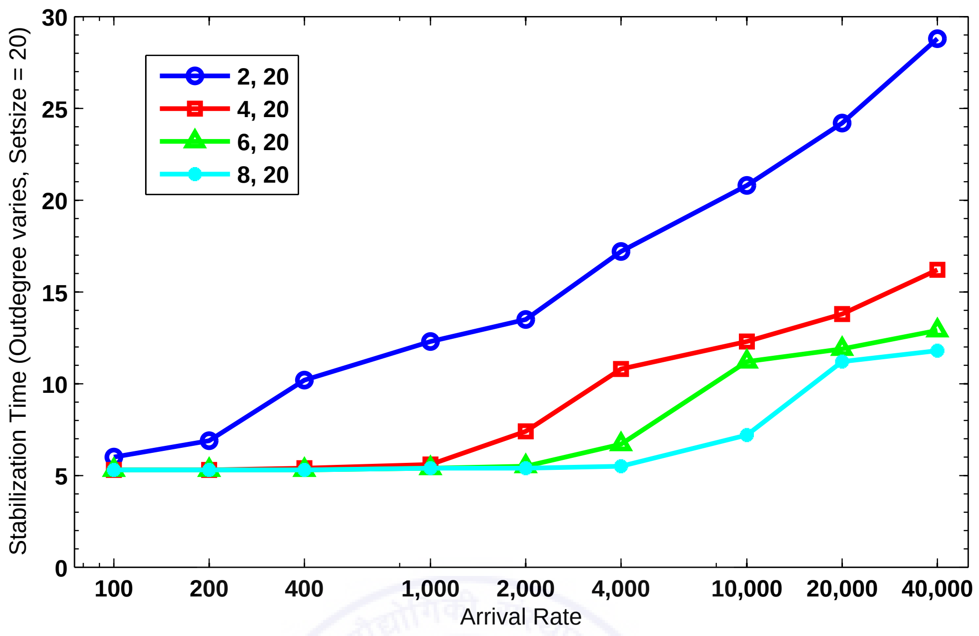

Figure 6.2: Effect of increasing arrival rate on the stabilization time for various out degree values for set size = 20

6.4.3 System scale and peer start-up delay

In the simulation to see the scaling behaviour, we assumed that initially the system contains only the source node. Thereafter, in next 5 seconds, total 200,000 nodes are added at the rate of 40,000 nodes/second. The values of other parameters are set as following: $D = 1$/seconds, $etime = 10$ seconds, $d = 4$ and $Ssize = 20$. Figure 6.7 shows how the system scales with time. When the flash crowd arrives, initially the system grows slowly as limited resources are available in the system, but when sufficient feed-forwarders are available; the system grows very fast and even surpasses the node arrival rate.

Figure 6.8 shows the distribution of peer start-up delay for the arrival rate of 32,000 nodes/second with other parameters set as above.

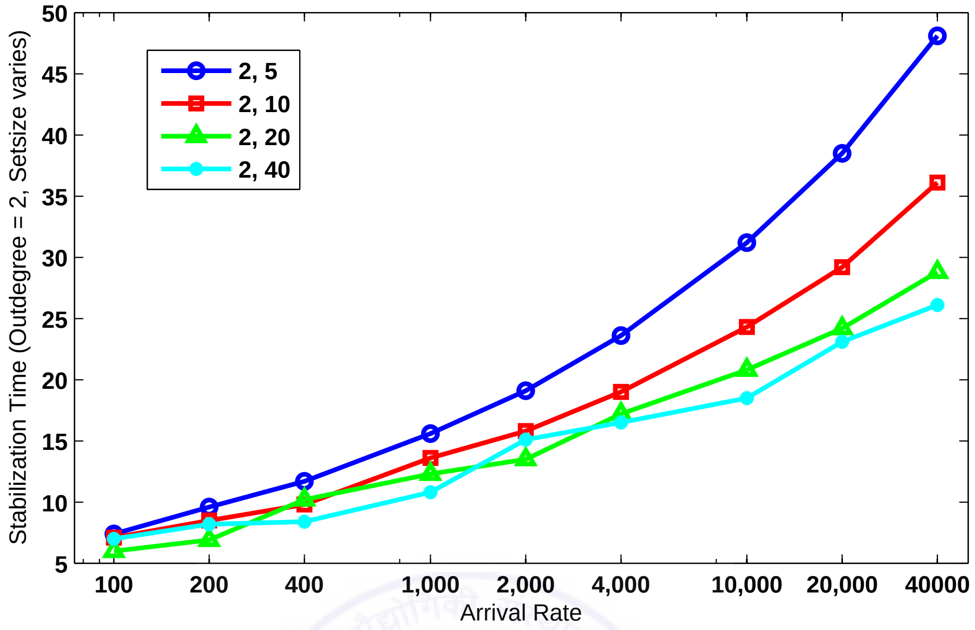

Figure 6.3: Effect of increasing arrival rate on the stabilization time for various set size values for out degree = 2

6.4.4 Steady state behaviour

Finally, we see the behavior of network with continuous constant arrival rate for sufficiently long time. It is observed that the value of N (number of sets), the number of sets increases monotonically with time and finally saturates after certain time. If entry expiry time is increased, N at saturation increases, while increasing Ssize decreases N at saturation. Observations for number of sets against time at continued arrival rate of 8000 nodes/second, for variations in out-degree (d) are shown in figure 6.9. The two plots for out-degree values 4 and 8 respectively are obtained for constant set size ($Ssize$ =50) and entry expiry time ($etime$ =10s). Observations reveal that the final value of N does not depend much on degree.

Observations for number of sets against time for variations in set size for

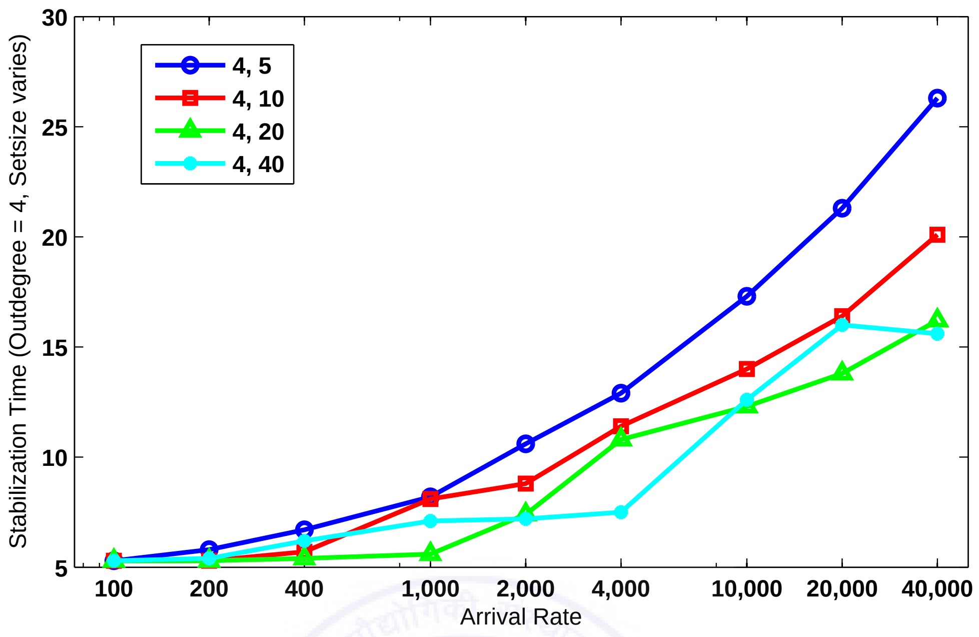

Figure 6.4: Effect of increasing arrival rate on the stabilization time for various set size values for out degree = 4

arrival rates 2000, 4000 and 8000 nodes/second are shown in figures 6.10, 6.11 and 6.12 respectively. These observations are taken for set size variation between 10 to 50 while out-degree (d=8), and entry expiry time ($etime$=10s) are kept constant.

In all these plots, there appears almost a flat region sandwiched between two distinct regions with different slopes and finally saturation occurs. The reason for the flat region is following. At the arrival of surge in the beginning, many new sets are formed almost simultaneously, though none of them have more than 1 or 2 entries and it takes time to fill these sets. This flat region denotes the time elapsed while the average occupancy of these sets increases from 1 to the maximum value.

Further, this is to state that for each data point, multiple simulations have

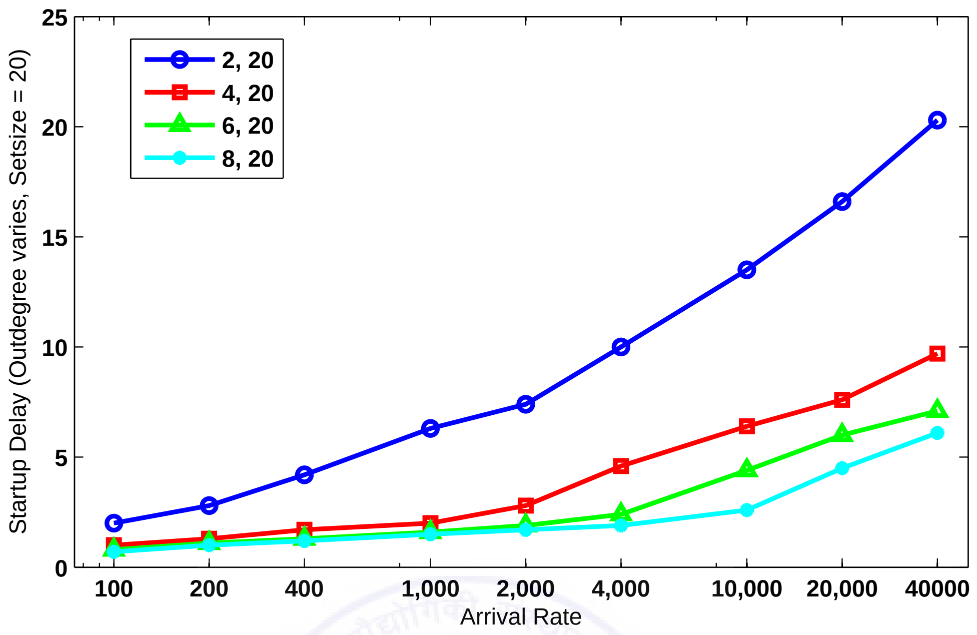

Figure 6.5: Effect of increasing arrival rate on the startup delay for variation in out degree; setsize remains constant at 20

been done untill the standard deviation is less than 5% of mean value. Thus for each point, the number of runs are different. Since confidence interval is small and uniform, it has not been shown in the results.

6.4.4.1 Analysis for the steady state behaviour

Since there are constant number of sets in the steady state, total number of feed forwarders will also be constant. It implies that total number of arriving nodes (that become feed forwarders) is equal to total number of feed forwarders exhausted per second. Since every second $1/etime$ part of total forwarders expires (from the definition of entry expiry time). Now, there are $(N.\ Ssize)$ feed forwarders under steady state.

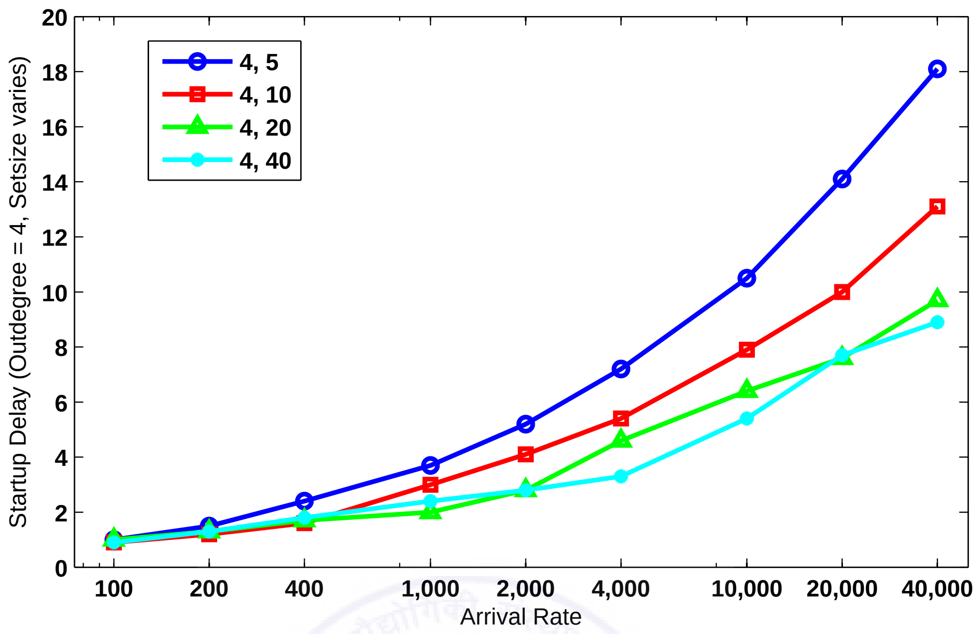

Figure 6.6: Effect of increasing arrival rate on the startup delay for variation in set size as out degree remains constant at 4

$$\lambda = \frac{N.Ssize}{etime} \tag{6.1}$$

$$\Rightarrow \quad N = \frac{\lambda.eime}{Ssize} \tag{6.2}$$

The value of N calculated as above matches with the value of N obtained by simulations.

Since arrival rate is λ, λ feed forwarders are selected per second. One feed forwarder is selected per 1/ λ second.

Assume no node is expiring and N is very large. On every new arrival, a feed forwarder is selected with probability

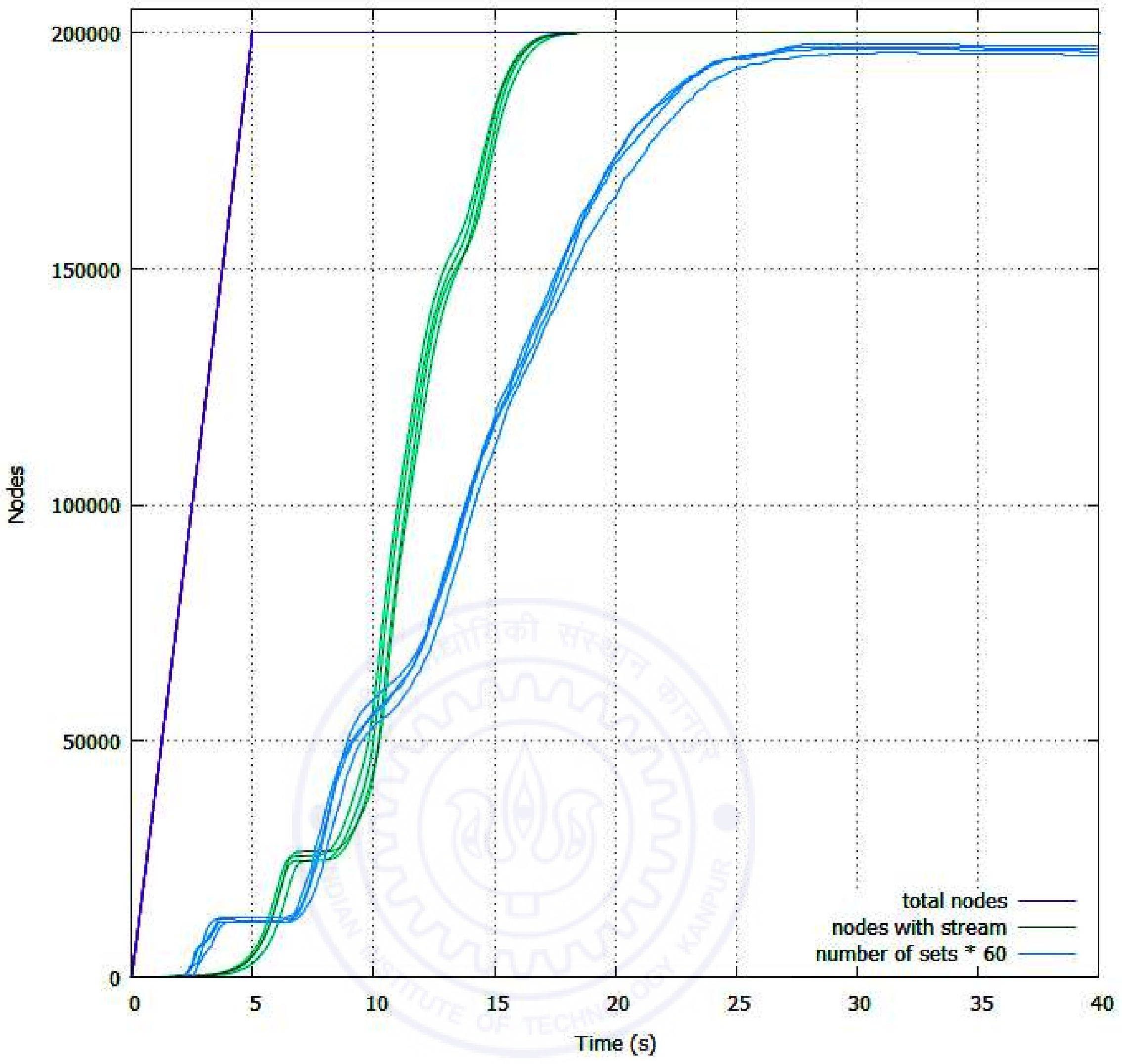

Figure 6.7: System scales with time when nodes are arriving with a rate = 40000 nodes/second for 5 seconds

$$P(\text{any of existing feed forwarders is selected}) = \frac{1}{N.Ssize} \qquad (6.3)$$

Since there are λ arrivals per second, a feed forwarder is selected λ .(1 / $N.$ $Ssize$) times per second. Therefore time needed to select it d times will be ($N.$ $Ssize.$ d) / λ. And thus approximate time to exhaust all d degrees is

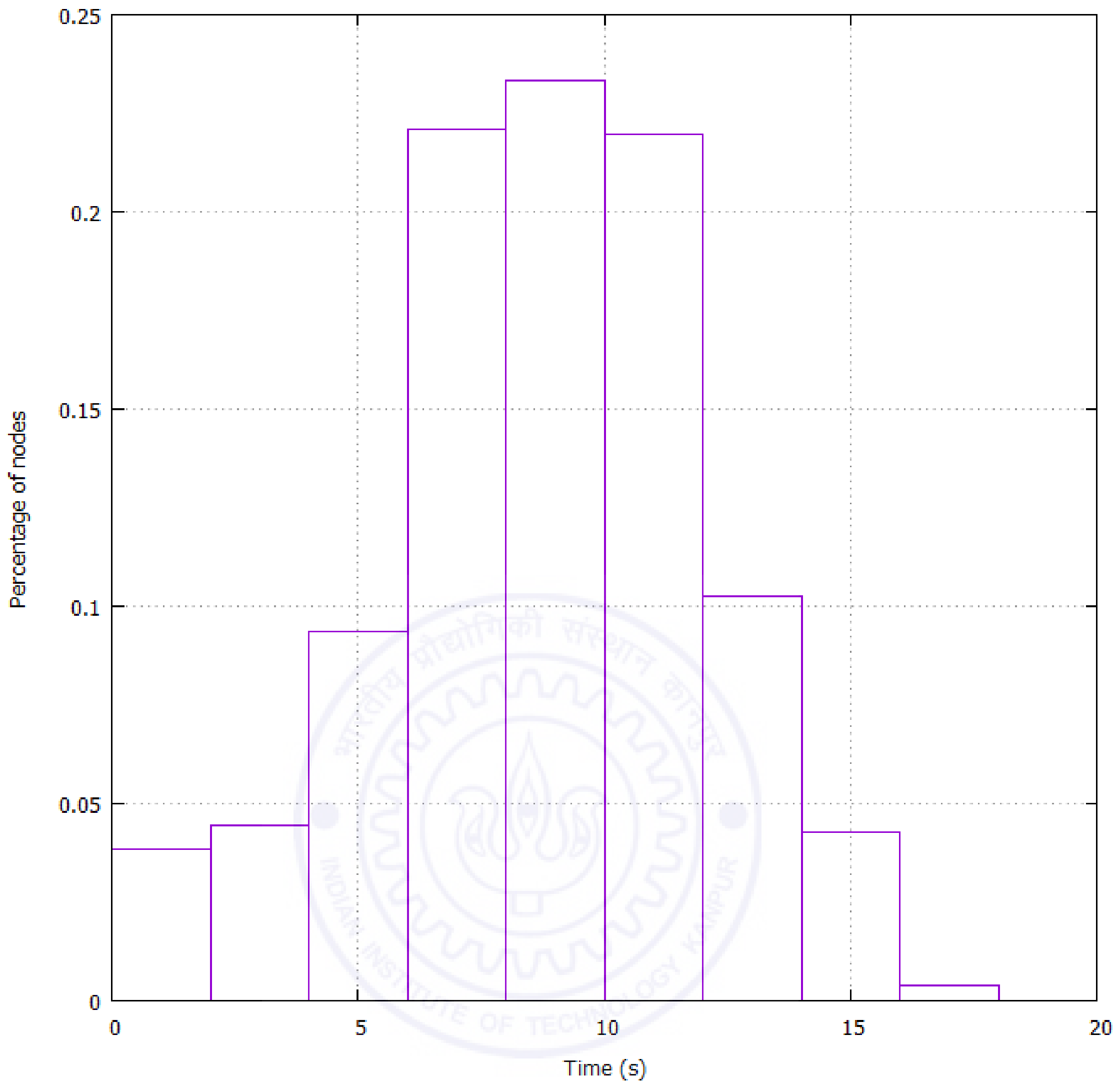

Figure 6.8: Distribution of startup delay for rate = 32000 nodes/second

$(N.\ Ssize.\ d)\ /\ \lambda$.

Finally we can comment that in the steady state, out-degree of forwarding nodes will be around 1 and not d since in the steady state N is constant and hence $N.\ Ssize$ is constant i.e. number of forwarders is constant. Therefore under steady state, as many nodes are added, that many numbers of feed

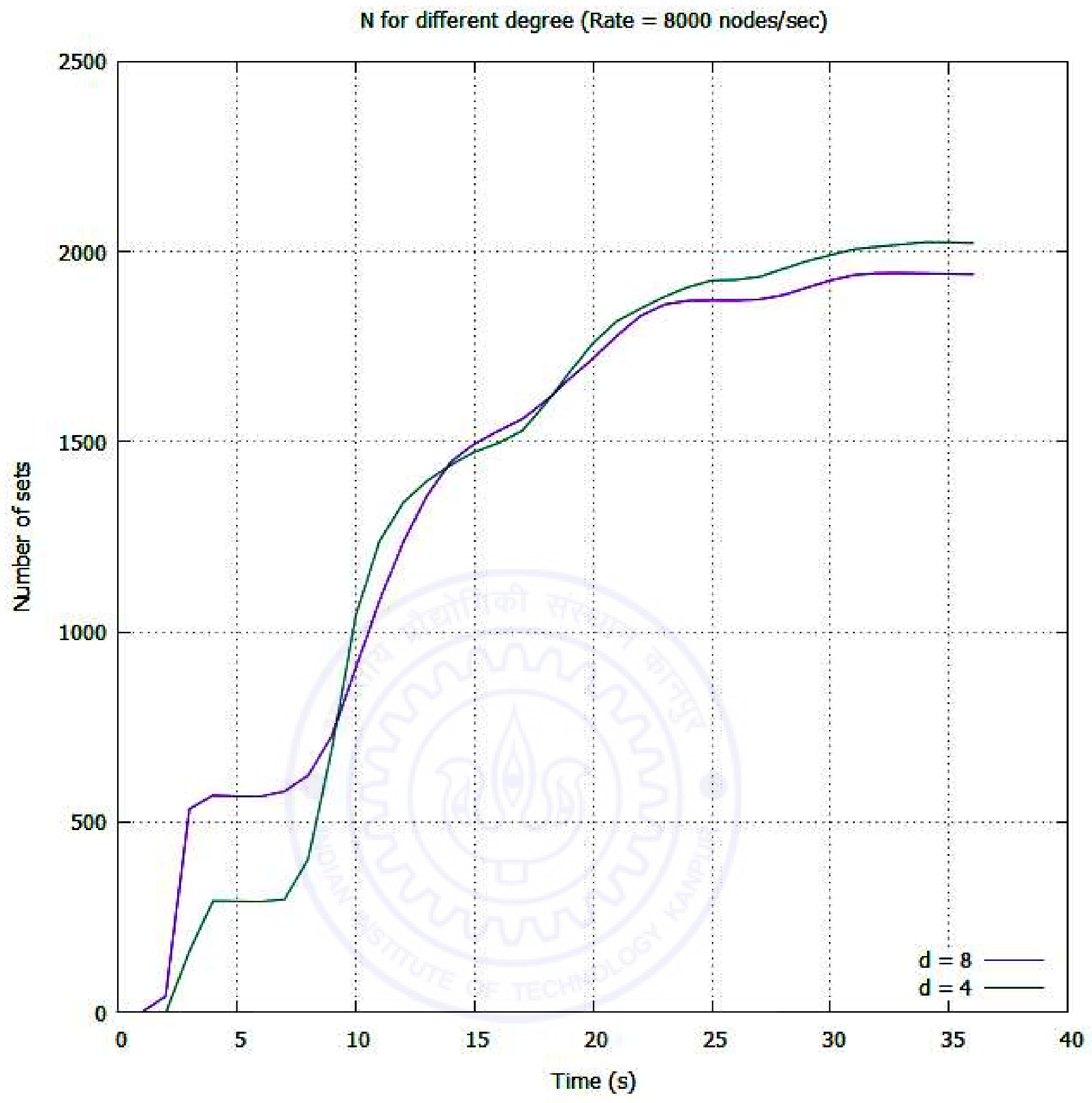

Figure 6.9: Variation of N against time for continued arrival rate for out degree values as d = 4 and 8; set size (= 50) and *etime* (= 10 seconds) remain constant, Rate = 8000 nodes/second

forwarders are consumed. Thus only one out-degree is filled in the feed forwarders. They are removed due to expiry timer.

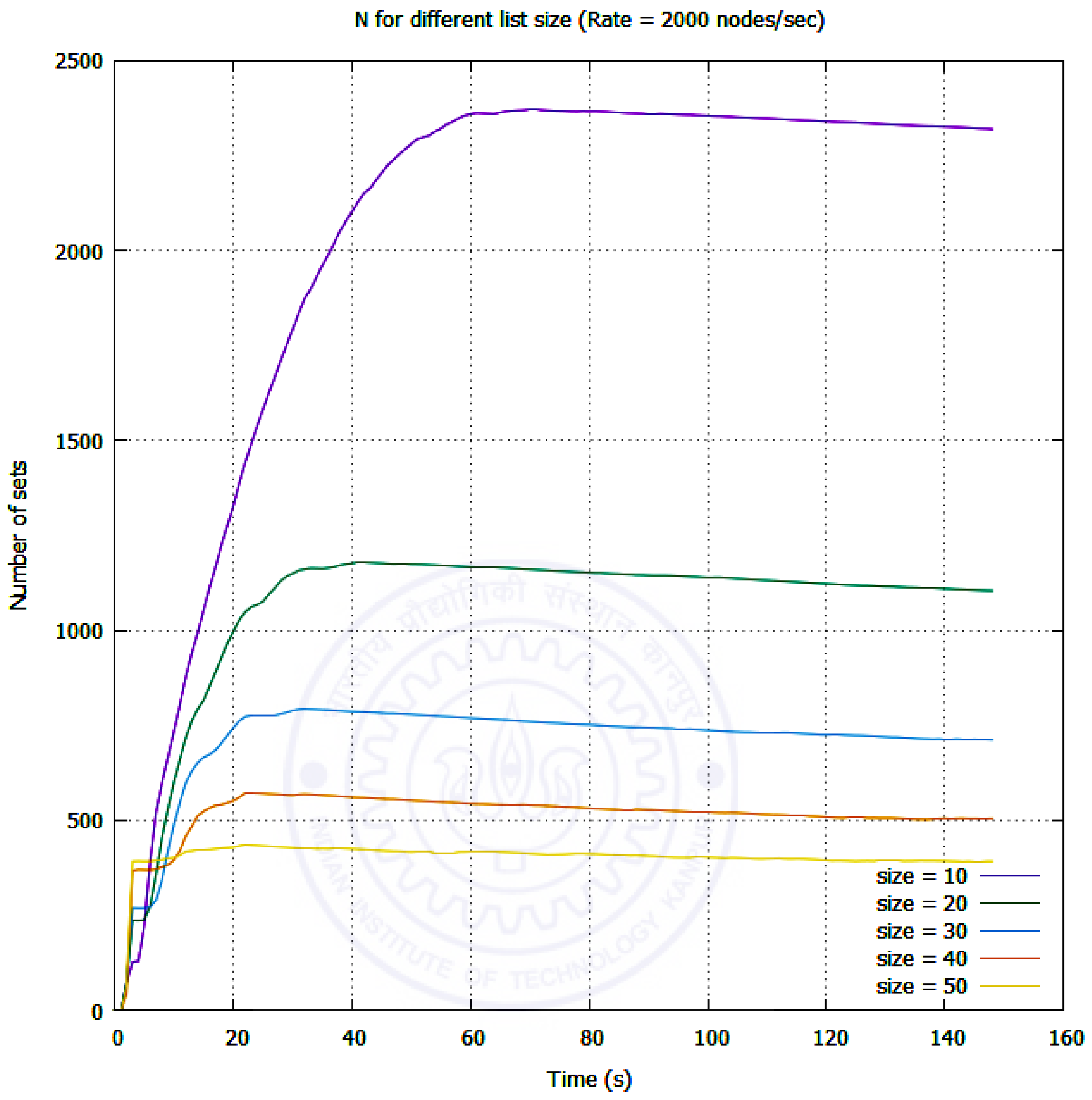

Figure 6.10: Variation of N against time for continued arrival rate for different set size values; peer's feed forwarding capacity d (= 8) and *etime* (= 10 seconds) remain constant, Rate = 2000 nodes/second

6.5 Conclusion

The algorithms presented to handle flash crowd in this chapter reduce the load at root nodes by dynamically changing their numbers and thus new

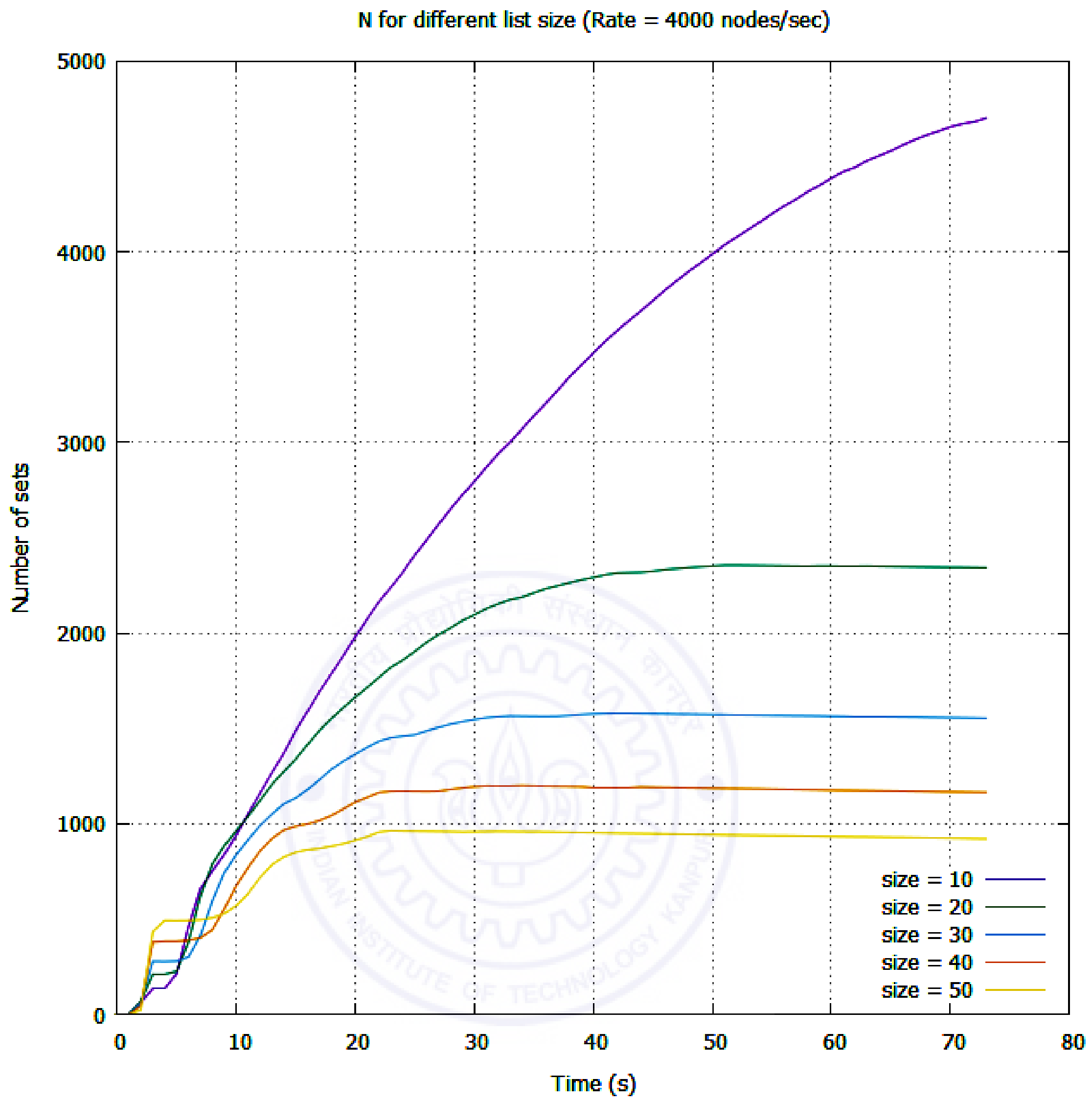

Figure 6.11: Variation of N against time for continued arrival rate for different set size values; peer's feed forwarding capacity *d* (= 8) and *etime* (= 10 seconds) remain constant, Rate = 4000 nodes/second

arriving nodes are able to get the stream feed even at very high network growth rate. The number of sets (each root node maintains one set of feed forwarders) i.e., the number of root nodes starts increasing at the onset of

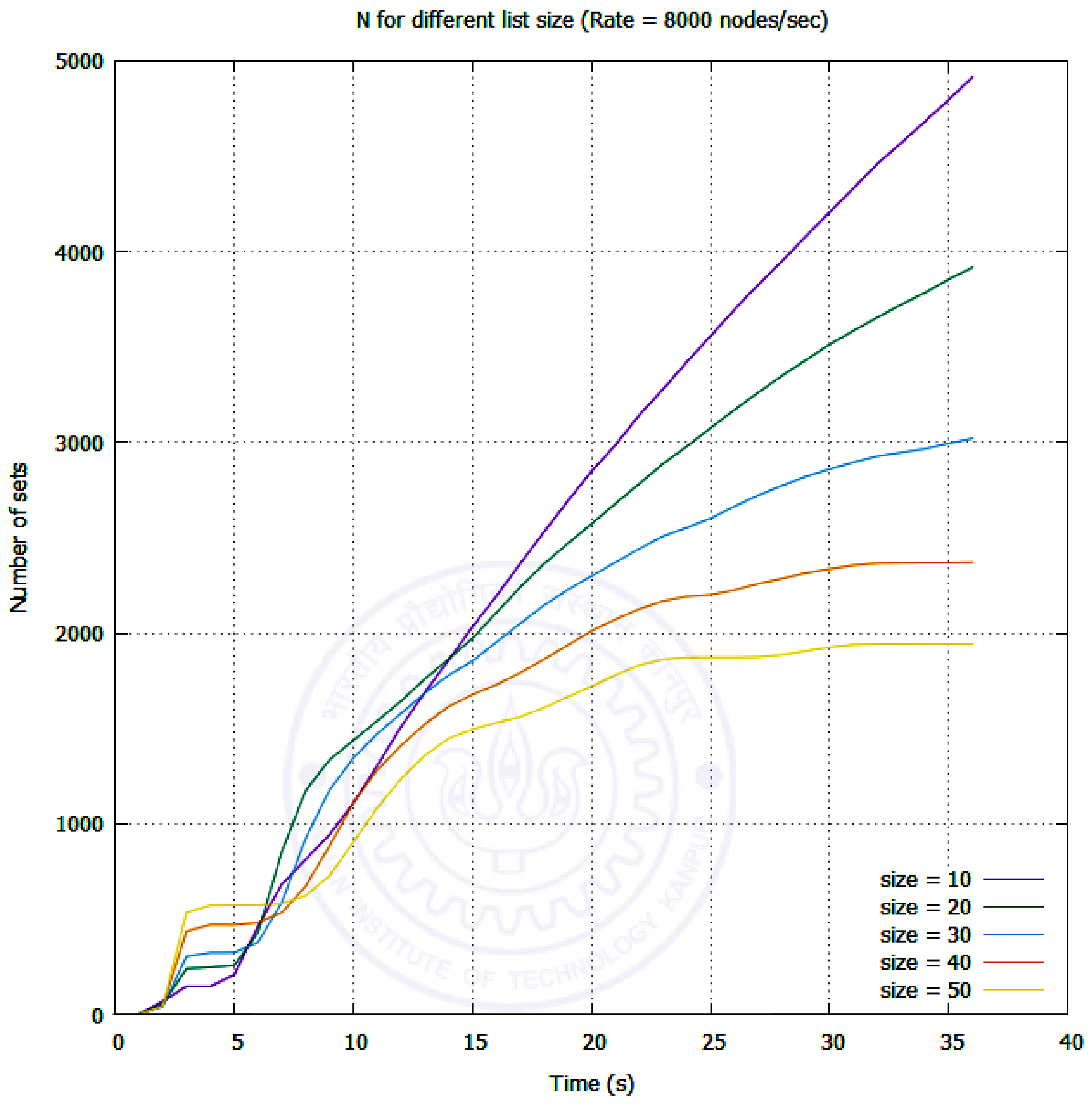

Figure 6.12: Variation of N against time for continued arrival rate for different set size values; peer's feed forwarding capacity *d* (= 8) and *etime* (= 10 seconds) remain constant, Rate = 8000 nodes/second

flash crowd, and then decays at a fixed rate. But at least one set is always maintained for all arrival rates. PeerSim based simulation results indicate that the proposed algorithm is effective.

There exists an optimum value for the pair (out-degree, set size) for which minimum stabilization time is obtained. For a given arrival rate, initially the system grows slowly but it soon surpasses the arrival rate as the feed forwarders' density increases. The scheme has provision to accommodate flash crowd with very high shock level yet it does have start up delay within a reasonable limit.

The results obtained are better than our earlier work [52] where distributed population control algorithm allows the system to scale up better as compared to the system without admission control. Though it was not a complete P2P design; as every node had to contact the bootstrapping server once before joining the network but the work done by the bootstrapping server was kept to a minimum. Through bootstrapping a nodeRank was assigned to each new arriving node. In [52], only 80% peers got the feed within 6 time slots, whereas 90% peers get the feed within 6 time slots with flash crowd handling algorithm presented here. The results presented here are part of ongoing research for proposed reliable and scalable Brihaspati Sync LLDS System [49].

Chapter 7

Conclusions and Future Scope

In the present chapter, the conclusions of this thesis work have been presented by revisiting the lessons learned during the course of research work. Directions for further research in this area are also mentioned at the end.

7.1 Conclusions

Exponential growth in the internet access to people all over the world and the advent of peer-to-peer (P2P) technology has revolutionized the way of human interaction in last two decades. P2P technology, that initially came up with an aim to optimize resource sharing among systems in the network gave birth to many interesting applications. P2P network can support multicast, also called as Application Layer Multicast (ALM) for millions of people distributed across the globe. ALM has been proven to be a good alternative of IP (network layer) Multicast where multicast-related functionalities are moved to end-hosts. In ALM, peers self organize themselves and an overlay topology is built for data distribution.

There have been many attempts to write an ALM protocol for cost effective, scalable live streaming but still an ideal solution has not come. For different sets of intended applications, there are different sets of issues that can be organized in a matrix form for more clarity with rows and columns corressponding to applications and problems respectively. Our contribution has attempted to address a small subset of issues from this matrix.

Whether we take structured or unstructured P2P network as substrate, the major concern in ALM is to route data efficiently and reliably in the overlay topology right from the beginning of a session.

The main contributions of the thesis are the following.

1. An algorithm for construction and maintenance of bi-connected mesh in unstructured P2P network is proposed. Algorithms to create biconnectivity in tree overlays are presented. Further, algorithm for creating distribution tree overlay over the bi-connected mesh is given. Comparative analysis of these algorithms is given at the end.

2. Structured P2P networks have been considered next. Three variants of dual-feed data distribution approach are proposed. Use of query network enables building of multicast tree directly as an overlay. Loop formation avoidance scheme in the event of failure is explained. Delay optimization schemes for distribution tree, and start up booster concept for Chord stability has been introduced in this chapter.

3. Flash crowd handling has been taken care of finally. Two different algorithms, one to alleviate the load of root nodes and the other to guarantee the availability of feed to any arriving node even at very high growth rate are proposed.

7.2 Summary of Important Findings

The basic terminology, performance metrics, and detailed description of few initial research efforts in the direction of development of streaming ALM protocols are given in tutorial form in chapter 2. It is evident from this survey that resource search becomes a major issue in a large P2P network and the scalable services of DHT based distributed lookup protocols become necessary.

Critical Comparison of available reliability approaches for ALM protocols is given in chapter 3. It has been found that the available approaches are insufficient to maintain perfect reliability which is necessary for live streaming application.

In chapter 4, Reliability in unstructured P2P networks is investigated. The algorithm for development of bi-connected mesh among participating hosts is found to be characterized by following.

1. Average out-degree requirement saturates soon, while the network diameter increases linearly with the number of nodes.

2. Two node-disjoint paths exist between any node pair in the network

Out of two approaches suggested for creating bi-connectedness in tree-first protocols, connected leaf nodes approach proves to be superior to child-grandparent approach. Algorithm for data distribution overlay maintains first hop nodes to two best paths towards source from any node.

Reliability issues in structured overlaid multicasting are investigated in chapter 5. Out of the three proposed approaches for dual-feed data distribution, approach-2, one with two priority trees, proves to be the best in terms of both differential delay and average latency. The differential delay does not increase beyond two, while average latency increases logarithmically with number of nodes. System scales soon with time and its failure performance is found to be as expected. Start up booster concepts plays a positive role for the Chord stability in case of high failure rates and the algorithms give better results whenever start up booster concept is invoked.

Chapter 6 covers the issue of flash crowd handling under high growth rate in overlay streaming protocols. Fully distributed algorithms, one to alleviate the load of root nodes, and the other to regulate the availability of forwarders in synchronism with the current requirement have been developed. Simulation results prove the effectiveness of the proposed algorithms at high growth rate.

7.3 Directions for Future Research

The present work has taken up the reliability issues at priority and suggested the solutions for associated concerns such as scalability and abrupt growth in the network for live media streaming. The prime objective has been to devise an algorithm which can be later implemented in BrihaspatiSync, an open-source live lecture streaming system. The issue of complexity has not been taken up in the thesis. Since the algorithms are distributive, the computation complexity at each node will be lower, however overall message complexity will be of significance and it is expected to be linear with factor N.(logN), if all N nodes are generating the querries. As further evaluation of algorithms, their complexity analysis can be done.

The proposals suggested in the thesis have been tested in PeerSim simulator. Further evaluation of the algorithms developed in this thesis can be done on an experimental testbed, that will give a better evaluation.

During the course of the investigations carried out in this thesis, following issues are identified for the future research in this area. These are as follows: prevention of free riding, Handling of heterogeneity among nodes, fair distribution of load among nodes.

Bibliography

[1] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users," *In proceedings of IEEE Infocom, NY, USA*, pp. 1366–1375, June 2002.

[2] T.-M. T. Kwan and K. L. Yeung, "On overlay multicast tree construction and maintenance," *In proceedings of IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, San Jose, CA, USA*, 2005.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," *In proceedings of 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, vol. 11, pp. 329–350, November 2001.

[4] Wikipedia, "History of communication," *http://en.wikipedia.org/wiki/History of Communication*, 05:46; June 28th, 2014.

[5] Matthews, Stephen, C. Bernard, and P. Marcia, "Atlas of languages: The origin and development of languages throughout the world," *New York: Facts on File*, 1996.

[6] B. Pourebrahimi, K. Bertels, and S. Vassiliadis, "A survey of peer-to-peer networks," *In proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, Veldhoven, The Netherlands*, pp. 1–8, November 2005.

[7] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," *In proceedings of the First IEEE International Conference on Peer-to-Peer Computing, Linkoping, Sweden*, pp. 99–100, August 2001.

[8] C. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," *Theory of Computing Systems*, vol. 32, no. 3, pp. 241–280, 1999.

[9] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on selected areas in communications*, vol. 22, no. 1, pp. 41–53, January 2004.

[10] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the internet*, 3rd ed. Pearson Education, 2005.

[11] internetworldstats.com, "Internet world stats: Usage and population statistics," http://www.internetworldstats.com/stats.htm, 2014.

[12] S. Fahmy and M. Kwon, "Characterizing overlay multicast network and their costs," *IEEE/ACM Transaction on Networking*, vol. 15, no. 2, pp. 373–386, April 2007.

[13] S. Banerjee and B. Bhattacharjee, "A comparative study of application layer multicast protocols," http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.2832, 2010.

[14] S. W. Tan, G. Waters, and J. Crawford, "A multiple shared trees approach for application layer multicasting," *In proceedings of the IEEE International Conference on Communications*, vol. 3, pp. 1456–1460, 2004.

[15] Y.-h. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Santa Clara, CA, USA*, pp. 1–15, June 2000.

[16] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," *In proceedings of ACM SIGCOMM Conference on applications, Technologies, Architectures and Protocols for Computer Applications, Pittsburgh, Pennsylvania, USA*, vol. 32, no. 4, pp. 205–217, October 2002.

[17] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "Almi: An application level multicast infrastructure," *In proceedings of the Third USENIX symposium on Internet technologies and Systems (USITS), San Francisco, CA, USA*, 2001.

[18] Z. Wang, X. Cao, and R. Hu, "Adapted routing algorithm in the overlay multicast," *In Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems*, pp. 634–637, November 28-December 1, 2007.

[19] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructutre for real-time applications," *In proceedings of 22nd Annual Joint Conference of*

the IEEE Computer and Communication Societies (INFOCOM), March-April 2003.

[20] X. Zhang, J. Liu, L. B., and T.-S. P. Yum, "Donet/coolstreaming: A data driven overlay network for peer-to-peer live media streaming," *In proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Miami, FL*, pp. 2102–2111, March 2005.

[21] "Pplive," *http://www.pplive.com/en/index.html*.

[22] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient multicast using overlays," *IEEE/ACM transaction on networking*, vol. 14, no. 2, pp. 237–248, April 2006.

[23] C. Huitema, "The case for packet level fec," *In proceedings of 5th international IFIP workshop on protocols for High Speed Networks*, 1996.

[24] J. W. Byers, M. Luby, and M. Mitzenmacher, "A digital fountain approach to asynchronous reliable multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, October 2002.

[25] Y. Kunichika, J. Katto, and S. Okubo, "Application layer multicast with proactive route maintenance over redundant overlay trees," *In proceedings of Conference on Advances in Multimedia Information Processing-PCM 2004, 5th Pacific Rim Conference on Multimedia, Tokyo, Japan*, pp. 306–313, November-December 2004.

[26] A. Mahanti and et. al., "Scalable on-demand media streaming with packet loss recovery," *In proceedings of IEEE ACM Special Interest Group on Data Communication (SIGCOMM) conference*, 2001.

[27] B. Rang, I. Kalil, and Z. Tari, “Reliability enhanced large scale application layer multicast,” *In proceedings of IEEE Global telecommunication System Conference (GLOBECOM), San Francisco , CA, USA*, 2006.

[28] F. E. Bustamante and Y. Qiao, “Friendship that last: Peer lifespan and its role in p2p protocols,” *In proceedings of Workshop on Web Control caching and Distribution*, Septeber 2003.

[29] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content addressable network,” *In proceedings of the ACM conference on Applications, technologies, architectures, and protocols for computer communications (ACM SIGCOMM’01)*, pp. 161–172, 2001.

[30] J. Liebeherr, M. Nahas, and W. Si, “Application layer multicast with delaunay triangulations,” *IEEE Journal of Selected Areas in Communications*, vol. 20, no. 8, pp. 1472–1488, 2002.

[31] M. Castro, D. P., A. M. Kermarrec, and R. A., “Scribe: A large-scale and decentralized application-level multicast infrastructure,” *IEEE Journal on Selected Areas in Communication*, vol. 20, no. 8, pp. 1489–1499, October 2002.

[32] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *In proceedings of ACM Special Interest Group on Data Communication (SIGCOMM) conference, San Diego, California, USA*, vol. 31, no. 4, pp. 149–160, August 2001.

[33] S. Birrer and F. E. Bustamante, “A comparison of resilient overlay multicast approaches,” *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1695–1705, December 2007.

[34] F. E. Bustamante and S. Birrer, "Resilient peer-to-peer multicast with out the cost," *In proceedings of 12th Annual Multimedia Computing and Networking Conference (MMCN), San Jose, CA, USA, January, 2005.*

[35] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," *In proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP), Bolton Landing, NY, USA*, pp. 298–313, October 2003.

[36] A. Singh and Y. N. Singh, "Multipath approach for reliability in query network based overlaid multicasting," *http://arxiv.org/abs/1309.3628*, September 2013.

[37] B. Li and J. Liu, "Multirate video multicast over the internet: An overview," *IEEE Network*, vol. 17, no. 1, pp. 24–29, 2013.

[38] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," *In proceedings of the 11th IEEE International Conference on Network Protocols (ICNP), Atlanta, Georgia, USA*, November 2003.

[39] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," *In proceedings of 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'02), Florida, NY, USA*, pp. 177–186, May 2002.

[40] V. Goyal, "Multiple description coding: Compression meets the network," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 74–93, 2001.

[41] Wikipedia, "Web acts as hub for info on attacks," *http://news.cnet.com/news/0-1005-200-7129241.html?tag=rltdnws*, 11 September 2001.

[42] "Akamai," *http://www.akamai.com/*.

[43] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the i p multicast service and architecture," *IEEE Network*, vol. 14, no. 1, January 2000.

[44] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," *In proceedings of 29th Annual ACM symposium on Theory of Computing (P2P'09), El Paso, TX*, pp. 654–663, May 1997.

[45] S. Y. Shi and J. S. Turner, "Multicast routing and bandwidth dimensioning in overlay networks," *IEEE journal on selected areas in communications*, vol. 20, no. 8, pp. 1444–1455, October 2002.

[46] X. Jin, W. P. Ken, and S. H. Gary Chan, "Loss recovery in application-layer multicast," *IEEE Multimedia, published by IEEE computer society*, vol. 15, pp. 18–27, 2008.

[47] A. Montresor and J. Mark, "Peersim: A scalable p2p simulator," *In proceedings of 9th International Conference on Peer-to-peer (P2P'09), Seattle, WA*, pp. 99–100, September 2009.

[48] "Peersim," *http://peersim.sourceforge.net*.

[49] "Brihaspati-3," *http://brihaspati.nmeict.in/brihaspati/servlet/brihaspati*.

[50] Y. Chen, B. Zhang, and C. Changjia, "Modelling and performance analysis of peer-to-peer live streaming systems under flash crowds," *In proceedings of IEEE ICC 2011, Kyoto*, pp. 1–5, June 2011.

[51] Z. Chen, B. Li, G. Keung, H. Yin, C. Lin, and Y. Wang, "How scalable could p2p live media streaming system be with the stringent time constraint," *In proceedings of IEEE ICC 2009, Dresden*, pp. 1–5, June 2009.

[52] A. Dwivedi, S. Awasthi, A. Singh, and Y. N. Singh, "Flash crowd handling in peer-to-peer live video streaming systems," *In proceedings of ICEIT MCNC 2015, New Delhi*, pp. 1–5, April 2015.

List of Publications

1. Ashutosh Singh, Anurag Dwivedi, and Yatindra Nath Singh,“Algorithms for Faster Overlay Creation under High Growth Rate in Query Network based Overlaid Multicasting”, National Communication Conference (NCC), IIT Guwahati, 4-6 March 2016, Available at IEEE Xplore Digital Library
2. Ashutosh Singh, and Yatindra Nath Singh,“Approaches toward Maintaining Biconnectivity for resilience in Overlaid Multicasting”, Networking and Internet Architecture, Computer Science, Cornell University Library, 2013, Arxiv.org/abs/1310.4291
3. Ashutosh Singh, and Yatindra Nath Singh,“Multipath Approach for reliability in Query Network based Overlaid Multicasting”, Networking and Internet Architecture, Computer Science, Cornell University Library, 2013, Arxiv.org/abs/1309.3628
4. Anurag Dwivedi, Sateesh Awasthi, Ashutosh Singh, Yatindra Nath Singh, “Flash Crowd Handling in P2P Live Video Streaming Systems”, ICEIT MCNC 2015, 16-17 April, 2015, New Delhi.