HPAM: Hybrid Protocol for Application Layer Multicast

Yeo Chai Kiat

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2006

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

17 November 2006
Date

Yeo Chai Kiat

Acknowledgements

I am extremely grateful to my supervisors, Professor Er Meng Hwa and Associate Professor Francis Lee Bu Sung. I would like to thank them for all these years of advice, guidance, encouragement, patience, understanding and support. I have benefited greatly from their creativity and vast knowledge.

My sincere appreciation to my husband, my two girls and my parents who have been a constant source of strength and comfort for me during all these years when I have to juggle my PhD work among my work and family.

I would also like to thank my bosses and colleagues in the School of Computer Engineering, Nanyang Technological University, for their warmth, support and understanding.

Last but not least, I would like to thank Singapore Advanced Research Networks (SingAREN) and acknowledge that this work is supported in part through SingAREN grant ICT/00/013/08.

Summary

In this dissertation, an application level multicast protocol for efficient multipoint live media streaming over the Internet without the need for native IP multicast support is presented. The proposed protocol is called Hybrid Protocol for Application Layer Multicast (HPAM). It is designed to be simple and scaleable and yet is topology-aware in order to respond to environmental changes. To disseminate media data to clients who have subscribed to the session, HPAM self-organizes these clients on the fly to form an efficient source-based overlay tree with no proxies used. The overlay comprises both unicast connections between the clients and native multicast islands which are within a client's site. Data distribution is based on a best effort model. The overlay tree is self-improving as it adapts readily to both group dynamics and network environments and it is also capable of self-recovery from a tree partition in the event of a client failure. The focus of HPAM overlay is to minimize the latency to root for each client while that of the enhanced version, HPAM-D, is to minimize two metrics, with latency to root as the primary metric and loss rate as the secondary metric.

HPAM has adopted a hybrid approach by exploiting the simplicity and optimality of a lightweight, centralized controller, *DS* (Directory Server), with the robustness and scaleability of distributed clients. *DS* facilitates peer discovery and also serves as a reliable backup should the distributed algorithms fails to perform.

Tree construction, refinement and recovery from partitions are carried out independently by the clients. The distributed tree refinement and self-recovery are accomplished through a systematic process of effective parent switching. Switching is facilitated by the novel use of the *Gossip* mechanism in conjunction with heuristics to determine who and when to switch to. In the gossip mechanism, clients maintain a cache of a select number of peers called gossipers which serves as a ready list of potential parents. Through the gossiping process, clients discover new gossipers. To improve tree quality, clients initiate proactive tree refinement algorithms to evaluate the potential QoS improvement they can derive from the gossipers if parent switching were to occur. Reactive tree refinement algorithm is further incorporated into the enhanced HPAM-D to complement the proactive approach so as to reduce the unacceptable loss rate experienced by a client. In addition, the gossip mechanism complements the spiral mechanism for orphaned clients to recover from tree partition.

Another unique feature of HPAM is the *JoinSource&Adopt* (*JSA*) algorithm, designed to prevent early-joining but latency-inefficient clients from hogging the level of the tree immediately below the root. Such hogging is a characteristic problem of overlay trees built on the fly without centralized control. *JSA* algorithm replaces these hogging ancestors with latency-efficient ones as new clients join the tree. By so doing, the overall latency of all clients downstream of the ancestors will be duly minimized.

Other innovations of HPAM include: the *Spiral* mechanism to provide client with a hot standby node for effective parent switching so as to achieve speedy partition repair; the incorporation of loss rate as the second metric in HPAM-D to boost the QoS delivered; the *Relative Loss Rate (RLR)* based heuristics for the detection of

possible local congestion that exists between a client and its parent, to reduce unnecessary parent switching; the study on the impact of cheating clients who fabricate distance measurements on HPAM performance; the cheat detection techniques to counter the cheats.

Extensive simulation analysis is performed to evaluate HPAM's transient and steady state performance in environments with stable group membership and network conditions as well as those with dynamic group membership and variable network characteristics. HPAM has been evaluated on metrics such as latency, link stress, protocol overhead, tree cost and its ability to adapt to group dynamics, to improve the tree quality, to repair and recover from partition. Comparative evaluations are also made against the centralized Host Based Multicast (HBM) and the distributed Host Multicast (HM). The main contribution of HPAM is that it can build and maintain application layer multicast trees with reasonable overheads and network stress and is able to deliver high QoS to its clients. This is largely due to the innovations incorporated in HPAM. HPAM's protocol overhead is lower than a fully centralized system and it is more responsive to group dynamics and network environment than a fully distributed system. It is certainly not as efficient as IP multicast but is far more efficient than naïve unicast in multipoint data distribution. HPAM therefore presents a simple and easily deployable solution towards solving the multipoint distribution problem. Simulation analysis concludes that cheating among clients is counterproductive and cheat detection increases the complexity of the protocols and overheads which undermine the very simplicity principle of HPAM's design.

Table of Contents

List of Figures

List of Tables

List of Abbreviations & Acronyms

ALM-CAN	Application-Level Multicast using Content-Addressable Networks
ALM-DT	Application-Level Multicast using Delaunay Triangulations
ALMI	Application Level Multicast Infrastructure
ALR	Absolute Loss Rate
BCFS	Bread-Crumb Forwarding Service
BGP	Border Gateway Protocol
BTP	Banana Tree Protocol
CBT	Core based Tree
CHK	Content Hash Key
CM	Core Member
DM	Designated Member
DS	Directory Server
DT	Detection Time
DVMRP	Distance Vector Multicast Routing Algorithm
DVTS	Digital Video Transport Service
HBM	Host based Multicast
HM	Host Multicast
HPAM	Hybrid Protocol for Application Layer Multicast
HPAM-D	Hybrid Protocol for Application Layer Multicast – Dual Metric
HPAM-L	Hybrid Protocol for Application Layer Multicast – Loss Metric
ICMP	Internet Control Message Protocol
IDMaps	Internet Distance Maps
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Providers
JSA	Join Source and Adopt
L1C	Level 1 Clients
MBGP	Multicast Border Gateway Protocol
MRT	Mean Recovery Time
MSDP	Multicast Source Discovery Protocol
MSN	Multicast Service Node
MST	Minimum Spanning Tree
NCM	Non Core Member
NICT	National Institute of Information and Communications Technology (Japan)
NONOG	North American Network Operator's Group
NRU	Normalized Resource Usage
NTP	Network Time Protocol
OMNI	Overlay Multicast Network Infrastructure
OSPF	Open Shortest Path First
P2P	Peer-to-peer
PGM	PraGmatic Multicast
PIM	Protocol Independent Multicast
PIM-SM	Protocol Independent Multicast – Sparse Mode

POR	Protocol Overhead Ratio
PTR	Proactive Tree Refinement
QoS	Quality of Service
RDP	Relative Delay Penalty
RepT	Repair Time
RIP	Routing Information Protocol
RLR	Relative Loss Rate
RP	Rendezvous Point
RTCP	Real-Time Control Protocol
RTP	Real-Time Protocol
RTR	Reactive Tree Refinement
RTT	Round Trip Time
RVL	Redundant Virtual Links
SAP	Session Announcement Protocol
SCX	Scattercast ProXies
SDP	Session Description Protocol
SHA	Secure Hash Algorithm
SN	Spiral Node
SPT	Shortest Path Tree
SRM	Scaleable Reliable Multicast
SSK	Signed Subspace Key
TBCP	Overlay Tree Building Control Protocol
TCP	Transmission Control Protocol
TCR	Tree Cost Ratio
UDP	User Datagram Protocol
URL	Uniform Resource Locator
vBNS	Very high-speed Backbone Network Service
VPN	Virtual Private Network
WIDE	Widely Integrated Distributed Environment

Chapter 1

Introduction

1.1 Motivation

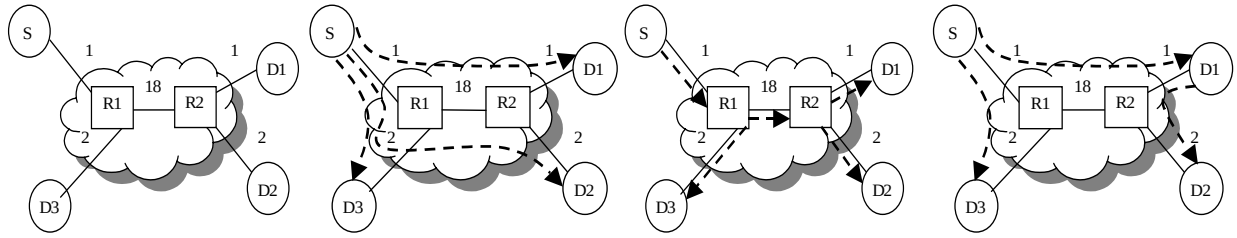
The Internet's ubiquity and connectivity have led to the globalisation of information and services. Internet's phenomenal impact and its subsequent growth, enhanced by advances in high-speed desktop computing, data processing, data storage and broadband network technologies, have resulted in unprecedented demand for distributed information services, an example of which is the world wide web (www) [1]. More and more people use the web for searching information, downloading content (e.g. software, music, video and images), shopping or entertainment. Information transmitted over the Internet is thus no longer confined to the traditional text-based medium but has become multimedia in nature.

Content distribution and information services over the Internet can be broadly classified by the mode of delivery, into two categories, namely, point-to-point delivery or group (multipoint) delivery. Point-to-point communication comprises only

two participants, one of which is typically a client (the requestor for content) and the other of which is a server (the content provider). Group or multipoint communication, on the other hand, involves more than two users in a one-to-many or many-to-many relationship where content is delivered to many clients simultaneously. Such services span a very wide spectrum from data streaming services such as Internet radio [2], Internet TV [3], video [4]-[6] and audio over the net [7], to multimedia collaboration such as audio, video, whiteboard conferencing [8]-[10], to distributed gaming [11], to real-time information dissemination such as stock quotes and updates [12] to non real-time content distribution such as software distribution.

Using conventional point-to-point unicast delivery protocol from the source to the multiple destinations in multipoint communication is inefficient and non-scaleable as this involves transmitting a duplicate copy of the datagram generated by the source to each and every client in the group. The source thus becomes a hot-spot as the number of destinations increases leading to network congestion and server overload. This is illustrated in Figure 1.1b which shows three copies of the data being duplicated on link S-R1 and two copies of the data being duplicated on link R1-R2. To alleviate traffic congestion and increase fault tolerance, many servers are replicated at several locations. The replicated servers (also called mirrors) are usually statically placed at some geographically important locations. Each replicated server will provide service to the clients in the local geographical regions. A client service will only be forwarded to another (replicated) server when its local server is not available due to whatever failures. The inefficiency of the unicast approach to multipoint communication as a result of its redundant use of network bandwidth and lack of scaleability is attested by the experience during the September 11 Terrorist Attacks on

USA when news web sites are inundated with requests: "*Demand on servers at the major news web sites was unprecedented – to such an extent that several of these systems were rendered inoperative for a period of hours*" [13]. The "melt-downs" of Internet news and other unicast streaming sites after September 11 were also reported in the October 2001 North American Network Operators' Group (NANOG) meeting [14].



(a) Network topology with routers R1, R2, end-clients S, D1, D2, D3 and link delays.

(b) Naïve unicast with significant data duplication with S as source. The expensive transcontinental link (R1-R2) is being traversed twice.

(c) IP multicast tree rooted at source S with no data duplication.

(d) Overlay tree rooted at source S with data redundancy reduced at the transcontinental link compared to naive unicast.

Figure 1.1: An example to illustrate naïve unicast, IP multicast and overlay with S as source and Dn are the destinations. The integers indicate link cost.

IP Multicast [15] - [17] has been regarded as an excellent solution to address the needs of multipoint communication as it is inherently bandwidth efficient and scaleable. It also has built-in robustness as it does not depend on any central sources such as reflectors or mirrors for data propagation. In fact, Eubanks in his presentation during the NANOG meeting [14], presented reports to show that multicast fared well after September 11 with no known outages, despite the sudden flood of news requests. IP Multicast reserves a set of addresses to identify groups of receivers. The source sends the datagram to the reserved address instead of directly to each of the receivers. In this way, the source only needs to send a single copy of its datagram and the

multicast-enabled routers within the network will only replicate the packets where necessary to forward them to all entities registered as members of a group. Data forwarding is effected along a distribution tree which spans all members of a group. The tree is rooted either at the source (e.g. DVMRP [18] which is illustrated in Figure 1.1c), a core router (e.g. Core-based tree [19]) or a rendezvous point (e.g. PIM-SM [20]) depending on the prevailing routing protocol used. A taxonomy of multicast transport mechanisms can be found in the survey paper by Obraczka [21].

However, IP Multicast is not without its drawbacks as cited by the authors in [22]. Today's IP multicast is limited to "islands" of network domains under single administrative control or local area networks, even though multicast has been implemented in many commercial routers. Most Internet Service Providers (ISP) are reluctant to enable it in their domains in order to reduce router load and to protect against unwanted traffic. This lack of ubiquitous multicast support more than a decade after its introduction thus limits the development of multicast applications, which in turn reduces the incentive for the network operators to enable multicast. It is therefore no surprise that the bulk of multicast traffic today occurs in the Mbone [23], an effort originated from the Internet Engineering Task Force (IETF) to link some of these multicast islands into a virtual network backbone. The problems of multicast can be summarized as follows:

- IP Multicast requires routers to maintain separate routing state for each multicast group which introduces high complexity and serious scaling constraints at the IP layer.

- The current IP multicast model allows for an arbitrary source to send data to an arbitrary group. There is a lack of access control and the groups entail complicated membership control as well as network management and provisioning.
- IP multicast requires every group to dynamically obtain a globally unique address from the limited multicast address space. It is therefore difficult to ensure scaleability and consistency with the distributed multicast address allocation.
- IP multicast is a best effort service and it has proven to be more challenging to provide higher level features such as reliability, flow and congestion control, security than in the unicast case.
- IP multicast entails substantial modifications at the infrastructural level since it requires fairly elaborate control support from the routers, in particular membership management and multicast routing protocols.
- Currently, there is an absence of a widely accepted, efficient, scaleable and deployable protocol for inter-domain multicast routing.

Simple Multicast [24], *Express* [25] and *Source Specific Multicast* [26] offer some alternate schemes to partially address the above problems by simplifying or improving multicast delivery in terms of address allocation and access control. Nevertheless, they lack an installed base as they all require substantial changes to the network infrastructure having the same dependency on routers as their traditional IP Multicast counterpart. The multicast deployment hurdles remain unresolved.

There is therefore a need to provide an efficient delivery solution for multipoint communication which is ubiquitous and does not possess the problems of the conventional unicast approach using servers/mirrors and entails neither IP multicast support in the routers nor modifications in the network infrastructure. To address this need, researchers have increasingly turned to solutions which employ application level overlays [27]-[48]. In fact, PlanetLab, a testbed for overlay networks seeded by Intel Research, believes that the next Internet will be created as an overlay in the current one [49].

An application level overlay refers to the approach whereby the end users of a multipoint communication group are connected via a virtual network. Each edge connecting a pair of end users corresponds to a unicast path between them in the physical network. The multicast functionality is moved from the routers to the end users, i.e. from the network layer to the application layer. End users route data along trees that are embedded in the virtual overlay network so that it reaches all members without any support from the routers beyond regular unicast transport service. Figure 1.1d illustrates an example of a smart overlay tree which spans the source and all the end-users. Note that although the overlay tree is not as efficient as IP multicast (Figure 1.1c) in its ability to minimize data replication, it is nevertheless more efficient than naïve unicast which causes high congestion in the link nearest to the source (Figure 1.1b). Since data is sent via unicast, flow control, congestion control and reliable delivery services that are available to unicast can be readily leveraged. There is also no need to allocate a global group identifier such as an IP multicast address.

However, overlay system incurs some performance penalty compared to a router-based system like IP multicast in terms of efficiency and latency. As overlay does not have the routing information available to routers, it must rely on end-to-end measurement mechanisms to infer the network metrics needed to build an efficient data delivery tree. Hence, the topology of an overlay will not be congruent to the underlying physical network. It is inevitable that multiple edges of the overlay are mapped to the same physical network link resulting in multiple copies of the same datagram traversing the same physical link. Moreover, data forwarding between end users incurs potentially higher latencies.

The critical issues to be addressed in overlay systems are therefore to devise algorithms for building efficient overlays, to track network metrics without incurring excessive overheads, to reduce the end-to-end latencies and to minimize the additional bandwidth requirements as compared to IP multicast.

1.2 Objectives

With the advent of broadband technologies, video and audio streams have become one of the most sought after content over the Internet. According to the latest report by AccuStream iMedia Research, the total video streams served/viewed rose 80.7% in 2004 to 14.2 billion where broadband streams constituted 79.3%. The report also projected that the total annual video streams will reach 28 billion in 2006 [50]. This testifies to the paramount importance of Internet as a complementary delivery channel to the traditional broadcast agents (e.g. TV). In fact, delivery of live content is a

capability that is unique to the Internet in the sense that users can access content and bypass the editorial controls and governmental stipulations necessary for broadcast purposes.

Live media streaming refers to the synchronized distribution of streaming media content. The content can be truly live (e.g. live soccer broadcast) or pre-recorded (e.g. a movie premier) while on-demand streaming distributes pre-recorded content to users on a demand-basis where the streams corresponding to each user are not synchronized (e.g. pay-per-view movie). From a user's perspective, their fundamental difference lies in that live content does not allow the user to revisit the content in future unless the content has been stored for future viewing. This has implications in the way content server performs admission control as well as the level of Quality of Service (QoS) acceptable to the user. For an overloaded server, the conventional approach is to reject new requests as user can revisit the stored content at a later point in time. This is however, unacceptable in live streaming as turning down a user's request amounts to denial of service since the value of its content is in its liveness. Moreover, latency in servicing user requests is less tolerable in live streaming as users would wish to partake in the session as soon as possible so as not to miss any live action such as live voting shows. Another consequence is the correlation between the user session length and the QoS of the playout. For live streaming, the correlation is weaker as users are more tolerant of a lower QoS threshold unlike the case of stored media where they will stop viewing a stream when the QoS degrades beyond the threshold since they can return later when the QoS has improved.

In brief, media streaming service is real-time in nature and typifies the one-to-many mode of delivery involving one source and many simultaneous clients. It is different from other types of content distribution by its tolerant for losses, its bandwidth-intensive nature and its critical time constraints. Its efficient delivery to multiple destinations therefore poses a unique challenge to network researchers especially.

For this research, our focus is on proposing a protocol for efficient and ubiquitous live media streaming over the Internet. The main objectives of this research are:

- to design an application layer protocol for group communication to support live media streaming applications ubiquitously;
- to minimize delay and loss in data delivery;
- to investigate the effects of different single network parameters and their combination on the data delivery efficiency of the protocol;
- to explore mechanisms to infer prevailing network conditions in the absence of interactions with network routers and knowledge on network topology;
- to investigate the impact of cheating clients on the efficiency of the protocol; and to propose detection techniques
- to compare the proposed architecture with existing schemes.

1.3 Major Contributions of the Thesis

The main contributions of this dissertation (not ranked in any order of importance) are as follows:

- The design of the Hybrid Protocol for Application Layer Multicast (HPAM)[1] for media streaming application which:
  - operates as an overlay on top of a dynamic, unpredictable and heterogeneous Internet;
  - allows accelerated application deployment;
  - is simple and scaleable yet topology-aware to adapt to changes in network dynamics and group membership;
  - does not rely on IP multicast support from the network infrastructure or any other specific network support or equipment;
  - does not require any modifications to the substrate network

- The development of a hybrid protocol which enlists distributed algorithms complemented with a light-weight centralized directory server to form efficient, loop-free, self-organizing and self-improving data distribution trees that adapt to network and group dynamics without incurring the overheads of using a mesh and maintaining a minimum of state information at each and every node of the distribution tree. Mathematical models have also been formulated to analyse the performance of HPAM and to derive the analytical upper and lower bounds as well as steady state values of its performance metrics where applicable.

- The incorporation of the *JoinSource&Adopt* (*JSA*) algorithm to specifically ensure that only clients with very small latency can occupy the level 1 (the

[1] This research is supported in part through grant ICT/00/013/08 from the Singapore Advanced Research and Education Networks (SingAREN).

level nearest to the root of the tree) slots of the data tree. This ensures the downstream clients do not accumulate excess latency due to inefficient level 1 ancestors.

- The incorporation of the *Gossip* and *Spiral* mechanisms to facilitate tree refinement and speedy repair. This innovation results in lower protocol overheads than a fully centralized system and a quicker response to group dynamics and network environment than a fully distributed system.

- The introduction of loss rate as a second performance metric in HPAM-D as opposed to the use of only latency in building and improving the data distribution tree. This helps most clients to maintain a better QoS. Moreover, the evaluation of performance of the data distribution tree based on single metric (latency only and loss rate only) versus that of dual metrics (latency and loss rate) is the first.

- The design of a *Relative Loss Rate (RLR)* heuristics for the detection of possible local congestion that exists between a client and its parent. This not only helps a client to reduce unnecessary parent switches thereby reducing protocol overheads but also contributes to the ability of the HPAM-D to maintain a low loss rate for its clients without compromising the RDP performance.

- The investigation into the impact of HPAM clients cheating in RTT measurements in order to move close to the root and to avoid fathering

children. To counter such cheating, innovative detection algorithms based on a combination of *DS* and community policing by the clients have been designed and incorporated into HPAM.

1.4 Organization of the Thesis

This dissertation comprises eight chapters. In Chapter 2, a comprehensive literature survey of the current application level solutions to overcome the deficiencies of IP Multicast in multipoint data delivery is presented. They are categorized based on their topology, architecture, service model and their merits and shortcomings are discussed. A qualitative performance comparison is also conducted to provide readers with a better insight into their contributions. Chapter 3 presents an overview of the proposed Hybrid Protocol for Application Layer Multicast (HPAM) complete with its design goals, architecture, topology used for data distribution, service model and group management operation. Based on its properties, its merits and shortcomings are discussed.

Chapter 4 presents a mathematical formulation of the problem which HPAM is trying to solve, namely to construct a tree with the aim of minimizing latency and to construct a tree with the aim of minimizing both latency and loss rate in the case of HPAM-D. Details of the four main components of HPAM, namely, *DS*, overlay tree construction, overlay dynamics which includes tree refinement and membership management as well as overlay robustness which comprises tree repair are presented.

Details of all algorithms and theoretical analysis of its performance are reported where possible.

Chapter 5 documents the results of the extensive simulation analysis of HPAM's performance in terms of parameters such as *Relative Delay Penalty (RDP), Network Stress, Tree Cost Ratio (TCR), Protocol Overhead Ratio (POR), Adaptation to Group Dynamics and Tree Improvement Ability* as well as *Message Load on DS.* HPAM's performance is also evaluated against other related application level multicast solutions, namely, a centralized protocol called Host Based Multicast (HBM) [42][43], a distributed protocol called Host Multicast (HM) [39] as well as against naïve unicast and IP multicast. HBM and HM are being evaluated against HPAM as they are most similar to HPAM in terms of the overlay topology since they all employ a tree-based approach and a flat topology in the overlay construction.

Chapter 6 focuses on two of the major enhancements made to HPAM. The first is the incorporation of loss rate as a second performance metric for the tree building algorithm in addition to the use of latency as the conventional performance metric which results in the HPAM-D protocol. The second is a new heuristics based on relative loss rate to make inferences of the prevailing network conditions to stabilize and improve the performance of the data delivery tree. Chapter 7 examines the effects of cheating clients on HPAM's performance. Application level multicast technique works on the premise that members in a session are willing and will honestly take part in the data forwarding process. This chapter examines the effects on HPAM when members are willing to take from others but are unwilling to give. Conclusions and recommendations for future work are given in Chapter 8.

1.5 Terminology

It is to be noted that the terms *members, clients* and *hosts* are used interchangeably throughout this dissertation.

The terms *source* and *root* are also used interchangeably.

As for the definition of the *level number* of a client, it is to be noted that the root of the tree is considered to be at Level 0. The root is thus sited at the top of the tree. Level 1 therefore refers to the level immediately below the root and Level 2, is below Level 1. Hence, the lower the level number of a client, the higher up the tree is the client, the nearer it is to the root. A high level number for a client means that the client is lower down the data distribution tree and is thus further away from the root and is nearer to the bottom of the tree.

Chapter 2

Literature Survey

Application level multicasting is a relatively new area which has started to generate interest in both the research community and the commercial world only in the last five years. Unlike other more matured research areas, there is a lack of comprehensive survey papers focusing on application level multicast until the efforts of E-Sayed et. al. [51] and Yeo et. al. [52]. In this chapter, we survey the various milestone research work related to application level multicast, classify them into different broad categories to facilitate better understanding of their contributions and discuss their merits and limitations. A comparative insight into their relative performance is also provided through the use of a set of evaluation metrics which are directly related to their performance and more importantly, whose data can be derived and inferred from their designs. Their properties and performance are tabulated for ease of reference. In addition, a vertical comparison of multicast protocols implemented on the different levels of the protocol stack, namely, IP multicast and application layer multicast is also provided.

2.1 Introduction

Application level multicast solution is emerging as a fundamental technique to circumvent the problems of non-ubiquitous deployment of multicast across heterogeneous networks to enhance wide-area service scaleability, performance and availability. It offers accelerated deployment, simplified configuration and better access control at the cost of additional (albeit small) traffic load in the network through an overlay-based approach. The general approach to building an application level multicast architecture involves tracking network characteristics and building appropriate topologies by having the end users to self-organize into logical overlay networks for efficient data delivery. Data delivery is accomplished via a data delivery tree which can either be the overlay itself or is embedded in the overlay. The overlay must be capable of failure detection and attempts to match the underlying network topology. Since the overlay abstracts away the details of multipoint message forwarding through the network and implements them at the end users, application level multicast is also popularly termed as end-system multicast or host multicast.

There are many ways to classify the myriad application level multicast (ALM) techniques. Here, we can broadly group them according to overlay topology design, service model, architecture and performance, to provide a systematic insight into the contributions by the different researchers. Different piece of work may appear in one or more categories.

2.2 Classification by Overlay Topology Design

Overlay is an integral part of application level multicast solution as it is the underlying mechanism effecting multipoint communication. The nodes in the overlays can be logically organized into two topologies, namely, the control topology and the data topology. Control topology carries control data such as heartbeat messages, refresh messages, network probes and probed data while data topology comprises the actual data delivery paths to the multipoint destinations. Nodes in the control topology may not necessarily be the members of the multicast group. Hence the control topology is a superset of the data topology. It is the norm for most techniques to adopt the tree structure for the data topology as it is simple to build and is efficient. Control topology may assume a separate physical structure in the form of a mesh where the nodes in the topology possess higher connectivity or it may share the same structure as the data topology. Depending on the approaches adopted, the resulting overlay topology can be classified into three distinct flavours: Tree, Mesh-Tree and Embedded Structure. Table 2.1 summarizes the topology design of the different proposals. Tree and Mesh-Tree design can be collectively grouped as topology-aware design while Embedded Structure is generally grouped under the topology-agnostic category with some exceptions as shown in Table 2.1. Network topology-aware design uses active measurements to infer network properties and to make an informed choice in constructing an efficient data topology to match the physical network topology as close as is practically possible. Topology-agnostic approach ignores network characteristics. Hence topology-aware algorithms have the advantage of minimizing the inefficiencies of overlays but do so at the cost of increased management overhead and potentially poor scaleability compared to its agnostic counterpart.

Table 2.1: Classification by overlay topology design.

Topology-Aware			*Topology-Agnostic*
Tree	*Mesh and Tree*	*Embedded Structure*	*Embedded Structure*
ALMI	Kudos	Bayeux	ALM-CAN
BTP	Narada	NICE	ALM-DT
HBM	Scattercast	SCRIBE	Freenet
HM	Yoid		
OMNI			
Overcast			
TBCP			

2.2.1 Tree

Group members self-organize themselves into a tree by explicitly picking a parent for each new group. Nodes on the tree may establish and maintain control links to one another in addition to the links provided by the data tree. As such, the tree, with these additional control links constitutes the control topology in a tree structure. This approach is simple and is capable of building efficient data delivery trees. However, the tree building algorithm must prevent loops and handle tree partition as the failure of a single node may cause a partition of the overlay topology.

Application Level Multicast Infrastructure (ALMI) [32] provides a multicast middleware which is implemented above the socket layer. ALMI is tailored towards

support of multicast groups of relatively small size (several tens) with many-to-many service mode. It employs a central controller to create 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 thereby optimizing the ALMI overlay for low latency data delivery. Control topology takes the form of unicast connections between each member and the controller. The central 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 prevent these problems, version number is assigned to each version of MST. Members maintain a cache of the different versions of the routing tables and only route packets with tree versions contained in the cache. If a packet with a newer tree version is received, it will re-register with the controller to receive the new MST. ALMI multicast trees have been shown [32] to be close to source-rooted multicast trees in efficiency with low performance tradeoff albeit with higher control overheads due to the maintenance of the different tree versions.

Banana Tree Protocol (BTP) [40] uses a receiver-based, self-organizing approach to build a shared group data tree. It is designed for distributed file sharing applications. The first host to join the group becomes the root of the tree. Subsequent new member learns of the root node and joins the tree. There is no special algorithm to prevent tree partition except for the affected node to rejoin the tree. The tree is incrementally optimized for low delay by allowing a node to switch

to a sibling if the latter is closer to the node than its current parent. The proximity metric used can be the round trip time (RTT) between the two nodes obtained from pinging each sibling. The siblings' information maintained in each node is updated by the node's parent. Figure 2.1 shows how sibling switching can lower tree cost. To prevent loops from forming during switching as a result of simultaneous switching in the case of Figure 2.2 and outdated information shown in Figure 2.3, two conditions must be fulfilled. First, a node will reject all attempts at switching if it is itself in the process of switching parents. Second, a node must include its current parent information in its switch request so that the potential parent can verify that they are indeed siblings.

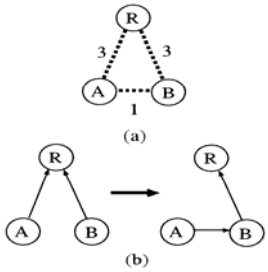


Figure 2.1: Sibling switching to lower tree cost in BTP[40]. Tree cost is lowered from 6 in (a) to 4 in (b) after node A switches to node B.

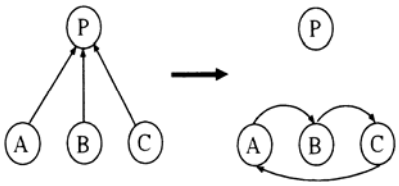


Figure 2.2: Simultaneous switching creates loop in BTP[40]. A tries to switch to B, while B is trying to switch to C and the latter is attempting to switch to A.

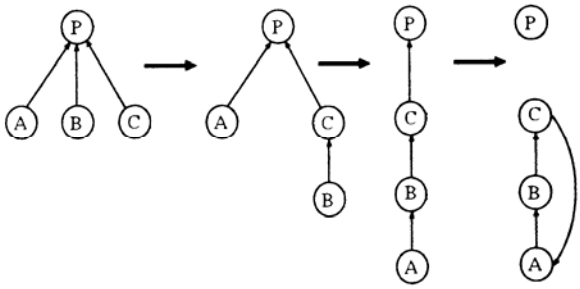


Figure 2.3: Switching with outdated information creates loop in BTP[40]. Suppose B switches to C and then A switches to B. If information has not been updated in C and A and C then switches to A, a loop is formed.

Host-based Multicast (HBM) [42][43] protocol creates a shared overlay tree among the various group members using point-to-point UDP tunnels. It epitomizes a full centralized system whereby a single host called the *Rendezvous Point (RP)* is responsible for the overlay topology calculation and its dissemination among group members as well as membership management. The RP has full knowledge of its membership and the distance between them. All members periodically evaluate their distances (RTT) and inform the *RP*. The *RP* periodically calculates a new topology and informs each member. *RP* distinguishes the client nodes as core members *(CM)* and non-core members *(NCM)*. The latter are nodes which are unstable such as a mobile node with a bad wireless connection. When the *RP* computes the tree topology, the *CM*, being stable, will form the transit nodes in the overlay tree topology and take part in the data distribution while the *NCM* are grafted as leaf nodes. As the stability of a node is unknown when it first joins a session, a default (conservative) value is first assigned to its *node_stability* variable and this is regularly updated as time goes on. The tree construction algorithm comprises *N-1* steps where *N* is the number of member nodes and is summarized as follows:

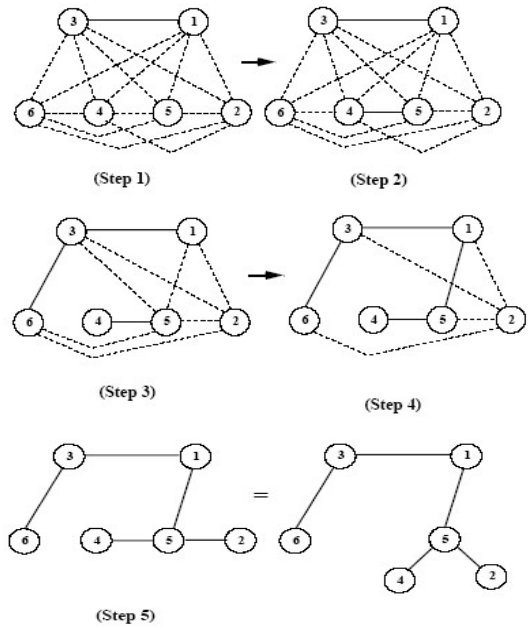
- At any stage of the algorithm, two sets of members are defined:
  - S_{in} contains the members already added to the current overlay topology. The members appear as subsets (C_i) of nodes which are already connected.
  - S_{out} contains the members not yet added to the overlay topology.
- First, the two closest members in S_{out} are chosen and creates the first subset C_0 in S_{in} (i.e. it creates a link between them in the overlay).
- Secondly, it chooses the next two closest members that are either:
  - two members in S_{out}, and in that case it creates a new subset for them,

- one in S_{in} and another one in S_{out}, and adds the member of S_{out} to the subset of the member of S_{in}, or
- two members in S_{in} but in different subsets, and merges these two subsets.

Note that when the member in S_{in} is a *NCM*, it will not be connected to any one in S_{out} or in another subset of S_{in} (i.e. when a *NCM* is connected to the overlay topology, it is deleted from the sets S_{in} and S_{out}).

- It repeats the previous process until there is no member in S_{out} with only a single subset in S_{in}.

Figure 2.4 shows the tree construction algorithm for HBM.



Given that *Node 4* is a *NCM* and the rest are *CM*.
Step 1: S_{in} = [NULL] , S_{out} = [1,2,3,4,5,6], Min ($S_{out} \rightleftharpoons S_{out}$) is 1 $\rightleftharpoons$3, so a new subset C_0 is created in S_{in}.
Step 2: S_{in} = [C_0={1,3}], S_{out} = [2,4,5,6], Min ($S_{in} \rightleftharpoons S_{out}$, $S_{out} \rightleftharpoons S_{out}$) is 4 $\rightleftharpoons$5, so a new subset C_1 is created in S_{in} and contains {5} because *Node 4* is *NCM*.
Step 3: S_{in} =[C_0 = {1,3}, C_1 = {5}], S_{out} = [2,6], Min ($S_{in} \rightleftharpoons S_{out}$, $S_{out} \rightleftharpoons S_{out}$, $C_0 \rightleftharpoons C_1$) is 3 $\rightleftharpoons$6, so 6 is added to C_0.
Step 4: S_{in} =[C_0 = {1,3,6}, C_1 = {5}], S_{out} = [2], Min ($S_{in} \rightleftharpoons S_{out}$, $S_{out} \rightleftharpoons S_{out}$, $C_0 \rightleftharpoons C_1$) is 1 $\rightleftharpoons$5, so C_0 and C_1 are merged into subset C_0.
Step 5: S_{in} =[C_0 = {1,3,5,6}, S_{out} = [2], Min ($S_{in} \rightleftharpoons S_{out}$, $S_{out} \rightleftharpoons S_{out}$) is 2 $\rightleftharpoons$5, so 2 is added to C_0. Algorithm is terminated as there is a single subset C_0 and S_{out} = [NULL].

Figure 2.4: Host-based Multicast Tree Construction Example[42].

HBM introduces some Redundant Virtual Links (RVL) into the tree topology until the probability of having a partitioned topology after a node failure falls below a predefined threshold [45]. Although this results in the creation of some loops, HBM deems it to be a simple partition suppression mechanism to incorporate.

Host Multicast (HM) [39] aims to provide best-effort multicast delivery service to applications and be compatible with IP Multicast to the furthest extent possible. It automates the interconnection of IP multicast enabled islands and provides multicast to multicast-incapable end hosts via unicast tunnels. Multicast islands are connected via UDP tunnels between the Designated Members (DM) where each island elects a DM. The architecture is shown in Figure 2.5. The data distribution tree is a shared tree where any member in the group can be a source. HM uses a distributed tree building protocol which scales to the number of group members (*O(N)* where *N* is the group size). As shown in Figure 2.6, a new member H discovers the root A of the shared tree through the Rendezvous Point (RP). H then sets A as a potential parent and requests for the list of A's children. Among A and A's list of children, H picks the closest one. The distance metric used is the member-to-member RTT and is collected via end-to-end measurements by HM. In this case, D is a new potential parent. H repeats the process down the tree and finds the next potential parent F. In the next iteration, if H finds that F remains the closest to itself, it issues a join request. If F has not breached its degree bound for the number of children, it will accept H's request. Otherwise, H will backtrack by one level and repeat its search. Each member in HM has to maintain information about all members on its path to the root. To improve the data tree, each member will periodically try to find a closer parent by reinitiating the join process from some random member on its root path. HM uses loop detection and resolution mechanism instead of loop avoidance. Loop detection is easily effected as members know its entire root path. To recover from tree partition problem, each member can rejoin anyone in its root path or in its

cache. The cache is built up during the initial process when the member 'walks' down the tree.

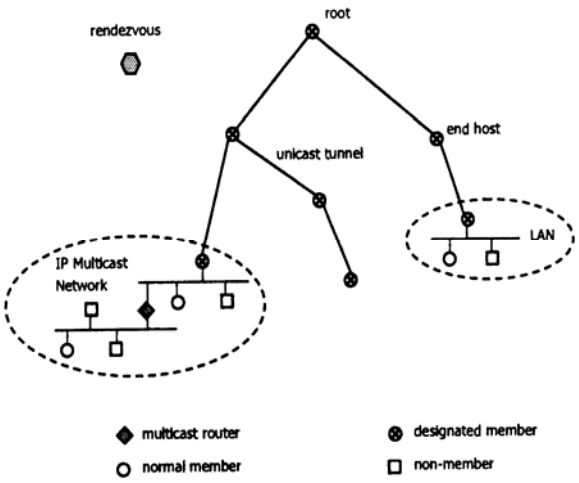


Figure 2.5: Host multicast architecture[39] .

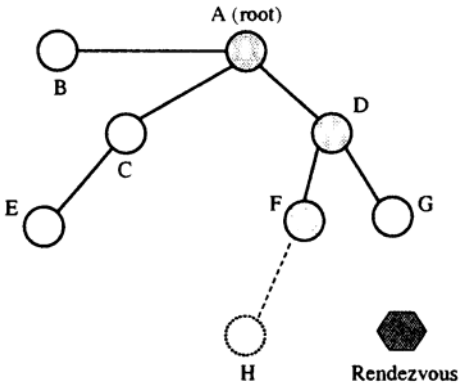


Figure 2.6: Join Process in host multicast [39] .

Overlay Multicast Network Infrastructure (OMNI) [41] constructs a single-source overlay tree from a set of service nodes called multicast service nodes (MSN) deployed in the network as shown in Figure 2.7. It is targeted at media-streaming applications. The root MSN is connected to the source. OMNI aims to minimize average latency of the entire overlay tree. Similar to HM, each MSN maintains state for all its tree neighbours, its ancestors and the overlay path from the root to itself. If the minimum out-degree of a MSN is two, and the number of MSNs in the group is *N*, then it maintains state for at most *O(degree + log N)* other MSNs. The root path is also used for loop detection during tree optimizations. The overlay tree building procedure comprises an off-line initialization phase before data delivery commences and the dynamic self-organizing process during data delivery. In the offline phase, the root MSN shown as A in Figure 2.7, measures and sorts the list of MSNs in

increasing order of unicast latencies from itself. It then builds a tree always choosing the nearest MSN to itself at each level of the tree. In other words, in Figure 2.7, D, E and F are further from A in latency terms than B and C. This centralized approach involves *O(N)* latency measurements. The key feature of OMNI's overlay tree is that it accords a dynamic priority to the different MSNs based on the size of its service set (i.e. the number of clients it serves) and iteratively optimizes the overlay tree. The dynamic self-organizing process is illustrated in Figure 2.8. At its initial configuration, the overlay latency from MSN 0 to MSN x is 59 ms (Panel 0). As the number of clients increases to 7 and then to 9, the importance of MSN x increases accordingly. It first changes its parent to MSN 6 (Panel 1) reducing its overlay latency to 54 ms and subsequently to the root MSN (Panel 2) with latency of 51 ms. Thereafter, the number of clients reduces and x migrates down the tree while other MSNs with larger client sizes move up. To adjust the overlay to changing latencies, each MSN performs periodic swapping among themselves if such swapping can reduce their current average latency. This swapping is strictly local in the sense that it is confined to within two levels of each other. OMNI also uses simulated annealing to probabilistically swap one MSN with a random member not within the two levels to allow the overlay to reach a global minima in terms of latency.

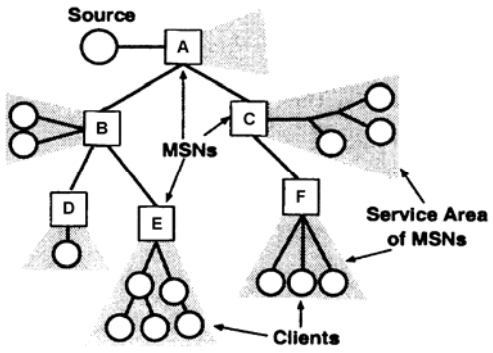


Figure 2.7: OMNI architecture[41].

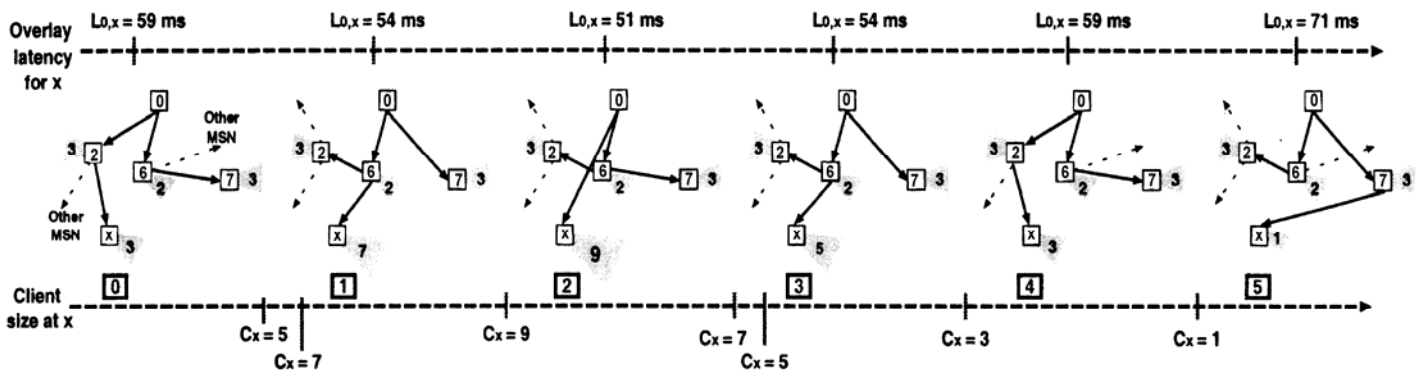


Figure 2.8: Dynamics of OMNI as the number of clients change at MSNs. MSN 0 is the root, MSNs 0, 2 and 6 had out-degree bound of 2 each and MSNs 7 and x had out-degree of 3 each. The number of cleints being served by MSNs is varied. The relevant unicast latencies (ms) between MSNs are as follows: $l_{0,2}$=29, $l_{0,6}$=25, $l_{0,7}$=42, $l_{0,x}$=51, $l_{2,x}$=30, $l_{6,2}$=4, $l_{6,7}$=18, $l_{6,x}$=29 and $l_{7,x}$=29. c_x indicates the number of clients at MSN x which changes with time. The time axis is not drawn to scale. [41]

Overcast's [29] objective is to maximize bandwidth to the source for all members potentially at the expense of delay increase. It aims to provide scaleable and reliable single-source multicast. It builds a source specific overlay tree which spans proxies nodes deployed across the network called overcast nodes. Overcast's tree building algorithm proceeds by placing a new node as far away from the root as possible without sacrificing bandwidth to the root. The new overcast node first contacts the root of the group. The root thereby becomes the current node. Similar to HM the node will obtain a list of children from the current node. However, unlike HM which measures the latency between itself and the list (including the root), it measures the bandwidth instead. The bandwidth is measured by observing the download time of 10 Kbytes of data. If the bandwidth through any of the children is about as high as the direct bandwidth to the current node, then one of these children becomes the current node and a new round of testing commences. In the case of multiple suitable

children, the child closest (in terms of network hops) to the joining member is chosen. If no child is suitable, the search for a parent ends with the current node.

Overlay Tree Building Control Protocol (TBCP) [54] aims to provide an efficient and distributed protocol to build control data delivery trees for application level multicasting. It builds overlay trees which are explicitly constrained in that the host fixes an upper-limit on the number of children it is willing to support. The root of the overlay tree is the main sender. It tries to build as good a tree as possible on the outset given the partial knowledge of group membership and restricted network topology information. A new member will be bootstrapped to the root via some well-known registry. Similar to HM and Overcast, the new member will be given a list of its children. The new node will measure the rtt between itself and the given list and the root and send the measured data to the root. The root will test all local configurations to select the one which is optimal. This will require testing of $O((N+1)!)$ configurations where N is the number of children belonging to the root. Figure 2.9 shows the possible configurations tested by the root (P) when new member M attempts to join the tree rooted at P with members C_1, C_2 and C_3. If the last configuration is the most optimal, M will be joined to P and C_3 will be redirected to start a join procedure with M assuming the earlier role of P. There is no special mechanism to recover from node failure.

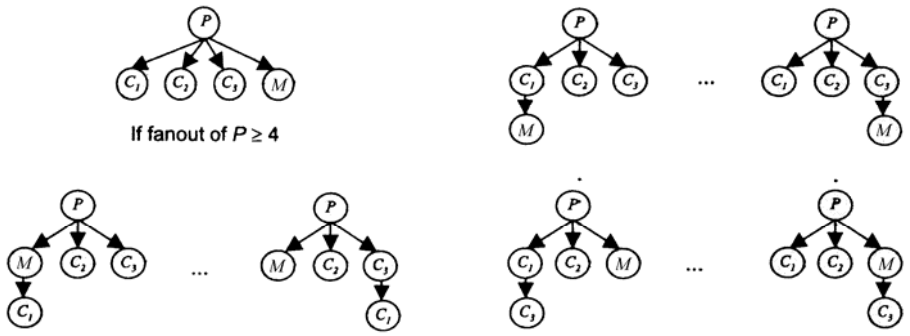


Figure 2.9: Local configurations tested in TBCP[54].

2.2.2 Mesh-Tree

The mesh-tree approach is a two-step design to the overlay topology. It is common for group members to first distributedly organize themselves into an overlay control topology called the mesh. A routing protocol runs across this control topology and defines a unique overlay path to each and every member. Data distribution trees rooted at any member is then built across this mesh based on some multicast routing protocols e.g. DVMRP. Compared to tree only design, mesh-tree approach is more complex. However, it has the advantages of avoiding replicating group management functions across multiple (per-source) trees, providing more resilience to failure of members, leveraging on standard routing algorithms thus simplifying overlay construction and maintenance as loop avoidance and detection are built-in mechanisms in routing algorithms.

Narada [27][28] is one of the first application layer multicast protocols that demonstrated the feasibility of implementing multicast functionality at the end hosts. It is targeted to support collaborative applications with small group size. Narada first

builds a mesh control topology across participating nodes and subsequently, the data topology is built on top of the mesh by having the group members self-organized into source-rooted multicast trees using DVMRP like routing protocol. This is shown in Panel 0 of Figure 2.10. New members obtain a list of group members in the mesh via a rendezvous point (RP) which maintains state about all members joined to the multicast group. The new member randomly selects some of these members as a neighbour under constraint of maximum degree (Refer to Panel 2). Each member keeps state about all other members in the group and the routing path and these states are updated via periodic refresh messages exchanges with one another. The aggregate control overhead resulting from the distribution of state messages is very high to the order of $O(N^2)$. This overhead for member discovery and the maintenance of membership in each member limit Narada to support only small to medium group multicast. However, the mesh control topology has the advantages of increased connectivity and is robust to overlay partition and node failure. This is illustrated in Panel 1. To adaptively refine the mesh which directly determines the quality of the data topology, every member periodically evaluates the utility of adding a link to another member of the control mesh and deleting existing links. An example is shown in Panel 3 where adding the edge <J,G> is useful as it results in a large number of shorter unicast paths being created on the mesh (e.g. between member sets {A,C,J} and {G,H}, and between member sets {C,J} and {F}). Edge <A,C> is removed since it has been made less useful after the addition of <J,G>. The utility is computed based on the following heuristics:

Adding Link: A member m computes the utility gained if a link is added to member n based on (i) the number of members to which n improves the routing delay of m (ii) how significant this improvement is in terms of delay.

Dropping Link: The cost of a link between m and n in m's perceptive is the number of group members for which m uses n as next hop and vice versa. These are computed and m will choose the higher of the two as the consensus cost of the link to each of its neighbours. The link with the lowest consensus cost will be dropped.

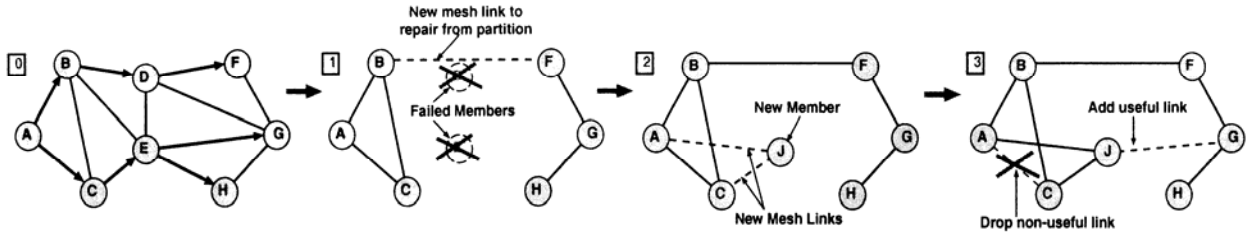


Figure 2.10: Control and data topologies in Narada. Neighbours on the control path (mesh) are connected by edges. In Panel 0, the arrowed edges indicate the multicast data path where A is the source. These edges also form part of the mesh. In Panel 1, the departure of D and E leads to a mesh partition. Remaining members, detect the departure and B adds a new mesh link to F hence repairing the partition. In Panel 2, a new member joins the mesh and sets up mesh-neighbour links with two randomly chosen existing members. Periodically members evaluate the utility of different links on the mesh. In Panel 3, a non-useful link is deleted from the mesh and a potentially useful link is added. [60]

Note that original Narada [28] only uses latency as the performance metric while the improved version [8] uses both latency and bandwidth to improve performance. The latter version prioritizes bandwidth over latency by incorporating these metrics into the distance vector protocol running on the mesh. The routing protocol uses a variant of the shortest widest path algorithm presented in [61]. Every member tries to pick the widest (highest bandwidth) path to every other member. If there are multiple paths with the same bandwidth, the member picks the shortest (lowest latency) path among these. The utility gain and the consensus cost in the algorithms

to add and drop links described are now computed based on the number of members to which performance improves (degrades) in bandwidth and latency if the mesh link were added (dropped) and the significance of the improvement (degradation).

Kudos [53] is an extension of Narada by adding hierarchy in the overlay topology to significantly increase its scaleability. Kudos partitions overlay nodes into clusters and each cluster has a unique representative node, called the cluster head. Figure 2.11 shows the two-level hierarchy. All non-head nodes in a cluster are referred to as children. At the bottom level, independent instances of Narada (the version using latency as the metric) is run within each cluster forming an overlay of children. At the top-level, an overlay spans across all cluster heads built using Narada too. In fact, any topology aware overlay building algorithm can be used. The data topology therefore comprises individual trees at each cluster in the bottom level and a tree at the top level. Likewise for the control topology. In an overlay of n nodes, Kudos form approximately $n^{1/2}$ clusters each containing $n^{1/2}$ nodes. The merits of the hierarchy lies in its superior scaleability and low management overhead since measure probes are run across smaller groups and effects of member failures are localized to smaller groups. This is achieved at the cost of efficiency as the children nodes in different clusters are unable to form overlay links to one another. The added complexity in Kudos is the clustering which involves migration, splitting and diffusion of clusters. A node joins any member using some bootstrapping mechanisms and learns of other cluster heads from its current head node. A node in a cluster can migrate to another if its periodic probes to other head nodes yield a significantly lower latency than its latency to its current head node. To reduce the load on head nodes due to such probes, a child will not probe a head node which is

more than twice the latency between the child and its current head node. Splitting occurs when a cluster is more than twice as large as $n^{1/2}$. The head node will remain as head for one of the new cluster and selects a new head node from the other new cluster. A head node is chosen to minimize the average latency to all other child nodes in its cluster. As Narada maintains state data of every other member in the overlay, pairwise latency data can be easier obtained for head selection. A cluster which has shrunk in size by more than a factor of two smaller than $n^{1/2}$ due to node migration, death or departure is disbanded. The remaining nodes migrate to other cluster.

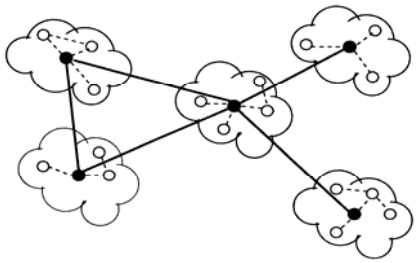


Figure 2.11: The two-level hierarchy of Kudos. Clouds represent cluster with the solid nodes as head nodes. The solid lines between the heads is the top level topology[53].

Scattercast [37] proposes an application level infrastructure to provide routing and forwarding services. It also provides a customizable transport framework on top of this infrastructure that leverages application defined semantic to tune the transport protocol for Internet broadcast distribution. It is similar to Narada except for the explicit use of infrastructure service agents, called ScatterCast proXies (SCXs). These proxies which are strategically deployed within the network infrastructure use a protocol called gossamer to self-organize into an application level mesh over which a global routing algorithm is used to construct source-rooted data distribution trees using latency as the routing metric. Figure 2.12 shows the scattercast architecture. Clients communicate with SCXs either via locally scoped multicast

groups or via unicast. Figure 2.13 shows the gossomer overlay topology and illustrates its resilience to failure and the mesh's reusability for multiple source-rooted trees. The initial mesh is randomly built as new SCXs bootstrap themselves via well-known list of rendezvous points and rely on gossip-style discovery algorithm (used initially in the Clearinghouse project [62] to maintain database consistency and have since been used to achieve fault tolerance and detection [63][64]) to locate other members and join them as neighbours. The gossip protocol works by having each member periodically and independently picks some random nodes and exchanges membership information thereby propagating information rapidly across the entire system. Optimization of the mesh is performed using latency as the metric based upon which the member decides to accept others as neighbours or to change neighbours according to a predefined cost function and threshold. The cost function of a member is the cost of routing to the various sources via its individual neighbours. A member X will be accepted as a neighbour to Y if its cost function is less than the maximum cost functions among all the existing neighbours of Y. Similar to Narada, every member must maintain a full state and routing tables to all the other members. Scattercast relies on centralized rendezvous points for repairing mesh partition. Although this assumption simplifies significantly the partition recovery mechanism, the partition problem will still arise in the event of failure of all rendezvous points.

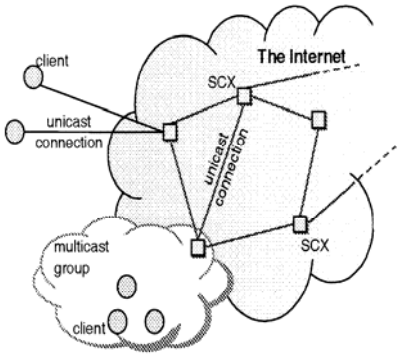


Figure 2.12: The Scattercast architecture[37].

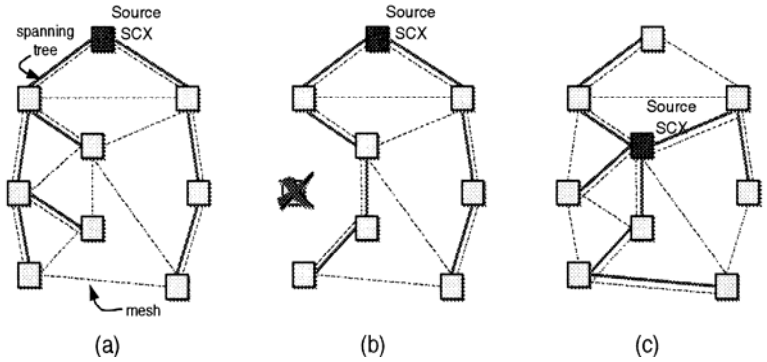


Figure 2.13: A typical Gossamer overlay topology. (a) Topology consists of a mesh with a source-rooted spanning tree superimposed atop. (b) The tree-mesh provides resilience to failures by automatically re-routing the tree around a failed SCX. (c) The mesh can be reused for multiple source-rooted trees without excessive additional computation[37].

Yoid [31][58] aims to extend the multicast architecture and defines a set of protocols for host-based content distribution either through tunneled unicast connections or IP multicast wherever available. It uses the tree-mesh structure differently from the other proposals categorized here in the sense that it builds a shared data tree directly and creates a mesh later to recover from tree partitions. Yoid's tree building algorithm is similar to BTP in that the first member to join a tree becomes its root and a new member queries a centralized RP which responds with a list of members (called candidate parents) which have already joined the tree. The new member then probes this list of members and decides on its parents based on a performance metric. The RP in Yoid takes on the additional roles of partition healing and group security. Yoid's objective of tree refinement is not so much to optimize performance but to avoid pathologies as its creators believe that the latter goes a long way towards acceptable level of performance. Tree refinement is accomplished through switching parents based on observed loss rates and latencies.

For latency measurements, Yoid members periodically query RP for updates on their initial lists of candidate parents. Each node operates tentative links to a subset of this candidate parents list. The average latency difference between the data frames sent via these tentative links is compared to those sent via the data tree. If the latter is higher than the former by a threshold, the node will switch to the candidate parent. Note that these tentative links also serve as ready potential parents for a node to switch to in the event that it is experiencing data losses which are higher than a threshold. Yoid member passively captures data losses from the data it receives and exchanges its loss print (average losses over a period) with its neighbours. Upstream nodes will gather loss prints from its loss print nodes to decide if a downstream node is to be removed to reduce fan-out and improve loss rate. Figure 2.14 illustrates such a process.

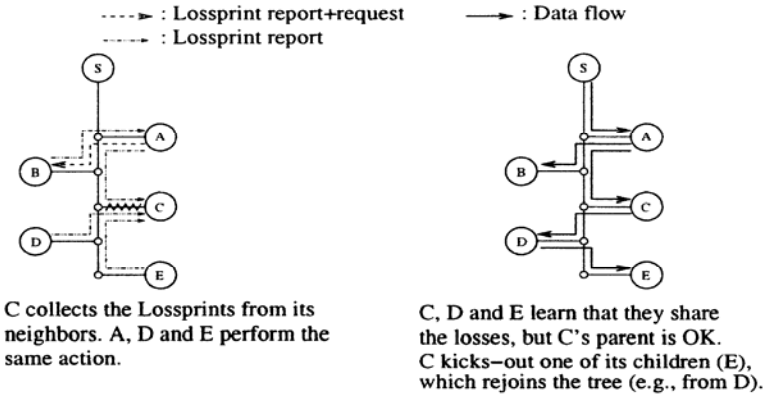


Figure 2.14: Loss-rate refinement algorithm for Yoid. C, D, and E reorganize to reduce the load on the congested bottleneck link[31].

Loop detection is effected through the presence of route paths in each node of the data tree as per HM. The mesh topology in Yoid is built by having each member maintains a small cache of members (mesh members) which are randomly selected using a frame delivery mode called “mesh anycast”. In anycast, a discovery message takes a random walk along the mesh and randomly selects a member. Mesh

members must be distinct and must not be tree neighbours. These mesh links ensure that partition resulting from node on the tree will be recovered through the additional connectivity.

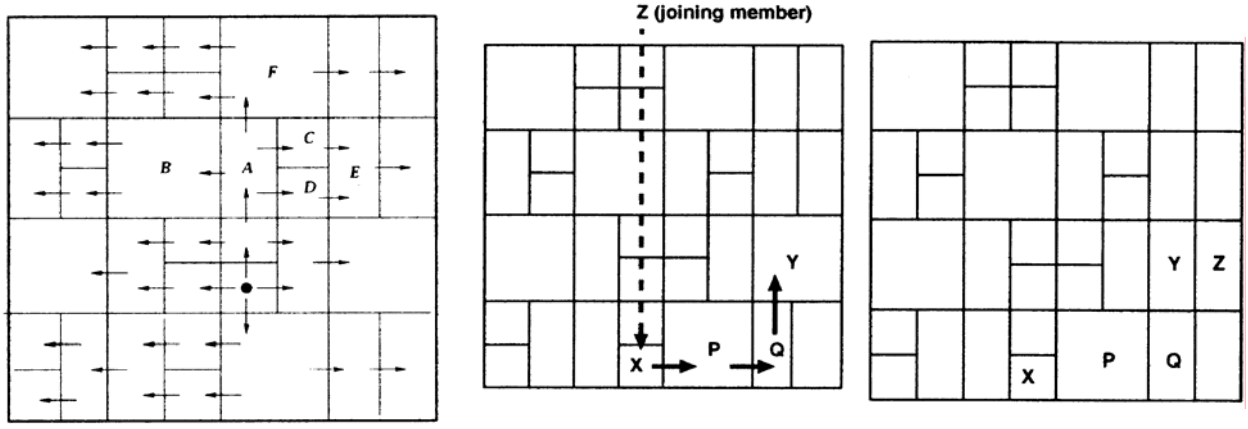
2.2.3 Embedded Structure

Embedded structure refers to proposals who assign to members of the overlay network logical addresses from some abstract coordinate space and builds an overlay with the help of these logical addresses. By embedding neighbour mappings in member addresses, next-hop routing of messages can be performed without the need for full fledged routing protocols such as DVMRP. Moreover, each member needs to maintain knowledge about only a small subset of members enabling the protocols to be highly scaleable. However, in contrast to tree and mesh based approaches, many of these protocols impose rules on neighbour relationships that are dictated by addresses assigned to hosts rather than performance. This may involve a performance penalty in constructed overlays as the overlays do not map well to the physical network topology. Hence a lot of overlay topologies built using embedded structure are also known as topology-agnostic designs with the exception of those which have implicit characteristics which enables them to be close to the substrate network. The following sections will first introduce the topology-agnostic solutions, namely ALM-CAN and ALM-DT and thereafter, proceed to detail the topology-aware proposals.

Application-Level Multicast using Content-Addressable Networks (ALM-CAN) [36] makes use of Content-Addressable Network (CAN) [65] architecture to provide an application level multicast solution. ALM-CAN is designed to scale to large group sizes without restricting the service model to a single source. CAN is an application-level structured peer-to-peer overlay network whose constituent nodes form a virtual d-dimensional Cartesian coordinate space and each member owns its individual distinct zone in this space. Figure 2.15 shows a 2-D coordinate space partitioned into zones by 34 CAN members of which six (A-F) are marked. The black dot represents a source node. Examples of the virtual coordinate zones of A, C, D are: A(2.0-2.5,2.0-3.0), C(2.5-3.0,2.5-3.0) D(2.5-3.0,2.0-2.5). The d-dimensional Cartesian coordinate space therefore comprises the control topology of ALM-CAN. The multicast data topology is implicitly defined by performing directed flooding on the control topology (Refer to Figure 2.15a). No explicit tree construction is required. Each CAN member maintains a routing table to its neighbours as well a packet cache to identify and discard any duplicate packets. Neighbours are defined as members whose zones abut each other. Hence for a d-dimensional CAN, a member node maintains state for 2d additional nodes (its abut neighbours) regardless of the number of source in the group. The data forwarding rule is summarized as follows:

The source forwards a data packet to all its neighbours in the control topology. The receiving neighbours in turn forward the data to all their neighbours except the neighbour from whom it receives the data. This is subject to the condition that the packet has not already traversed at least half-way across the space from the source coordinates along the dimension of data forward. This is to prevent data packets

from looping around the overlay. (Refer to the example in Figure 2.15a where D does not forward data upwards or downwards.)



(a)[36] (b)[60] (c)

Figure 2.15: Structure of a 2-D CAN and the corresponding control and data topologies.

A new member Z who wishes to join the multicast group queries a RP to find at least one existing member say X. Z then picks a random point in the 2-D coordinate space (say Y's space) and sends a join request through X to locate Y. This is done by routing through CAN as shown in Figure 2.15b. Upon locating Y, Y's zone is split into two with Z owing one of them (See Figure 2.15c). The split is done by assuming a certain order of dimension so that zones can be remerged when nodes leave. For a 2-D CAN, the split is along the X-dimension first, then the Y and so on. As ALM-CAN uses a topology-agnostic approach to build the control and data topologies without taking the relative distances between nodes into account, their data distribution paths can be very long compared to the physical network distance between them. Ratnasamy et al. [36] proposes to ameliorate this situation by using distributed binning by which members that are close to one another are assigned nearby zones in the coordinate space.

Application Layer Multicasting using Delaunay Triangulations (ALM-DT) [30] is aimed to support very large multicast group size. It assigns each member a pair of logical (x,y) coordinates in a plane. The coordinates can be assigned via some external mechanisms such as GPS or user input and can be selected to reflect the geographical locations of the nodes. Using these coordinates, the control topology which essentially comprises combinations of DTs [66], is built through angular calculations and comparisons based on the properties of DT. DT has been extensively studied in computational geometry [66]. A DT for a set of vertices A is a triangulation graph with the defining property that for each circumscribing circle of a triangle (See Figure 2.16) formed by three vertices in A, no vertex of A is in the interior of the circle. The underlying mechanism used in forming the DT control topology is the neighbour test. The neighbour test is based on the locally equiangular property. Details of this property is provided in [67]. Two nodes are neighbours in the overlay if their corresponding vertices are connected by an edge in the DT that comprises the vertices associated with all the nodes in the overlay. A new node N requests to join the group by contacting the DT server who bootstraps it to an existing node D. Besides managing group membership, the DT server also helps in repairing partition in the overlay. Node D will perform the neighbour's test on N and if successful, N will become D's neighbour. Otherwise, D route the join request to one of its neighbours whose coordinates are closer to those of N. The process repeats towards the coordinates of the new node N until it passes a neighbour's test. Hence the only state information to be maintained by each node is confined to that of its neighbours.

Once the control topology is built, the source-rooted multicast data tree is embedded in the DT without requiring a routing protocol as packet forwarding information is encoded in the coordinates of a node. Compass routing [68][69] is used to determine the multicast routing tree where nodes calculate their child nodes in the multicast routing tree in a distributed fashion. No loop detection is required as compass routing in DT does not result in loops [69]. Each node can locally determine its child nodes with respect to a given tree using its own coordinates, the coordinates of its neighbours and the coordinates of the sender. Hence average overhead of a node is a few kbps in steady state. Figure 2.17 shows an example of compass routing. TCP unicast connections are adopted.

Figure 2.16: A Delaunay Triangulation[30].

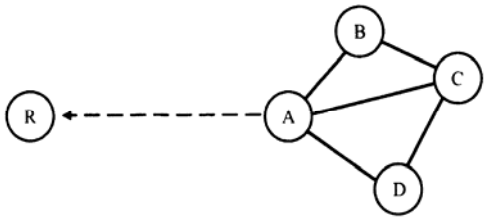


Figure 2.17: Compass Routing. A determines that it is the parent for C since ∠RCA < ∠RCD and ∠RCB. Likewise, B and D determines that they are not the parents of C since ∠RCA < ∠RCB and ∠RCA < ∠RCD[30].

DT's merits lie in that it generally has a set of alternate non-overlapping routes between any pair of vertices which can be readily exploited for building optimal data topology and to facilitate recovery when overlay nodes fail. DT can be established and maintained in a distributed fashion where no entities maintain knowledge of the entire group unlike Narada and Scattercast. It has been shown through experiments that it can scale up to 10,000 members [30]. Similar to ALM-CAN, its weakness lies in the suboptimal mapping of the overlay to the physical

network given the topology-agnostic nature of the logical coordinates of the members.

Bayeux [33] focuses on fault-tolerant packet delivery as a primary goal. It can be used for Internet content distribution and is designed to scale to arbitrarily large receiver groups. The control topology of Bayeux is the Tapestry overlay. Tapestry [59] is a wide area routing and location infrastructure which embeds nodes in a well-defined virtual address space. Nodes have names independent of their locations in the form of random fixed-length bit sequences represented by a common base (e.g. 40 Hex digits representing 160 bits). These can be generated by secure one-way hashing algorithms such as SHA-1 [70]. Tapestry uses similar mechanism to the hashed-suffix mesh introduced by Plaxton et. al. in [71]. It follows from the proof in [71] that the network distance traversed by a message during routing is linearly proportional to the real underlying network distance. Experiments performed in [59] verified that this proportionality is maintained with a small constant in real networks. Hence Bayeux's overlay is classified as topology-aware design although it is built via embedded structure approach. Tapestry uses local routing maps (neighbour map) stored at each node to incrementally route overlay packets to the destination ID digit by digit (e.g. ***8 => **98 => *598 => 4598 where * represents wildcards) as shown in Figure 2.18. Assuming consistent neighbour map, Tapestry's natural hierarchy ensures that the destination can be reached within at most $log_b N$ logical hops in a system with an N size namespace of base b. As each neighbour map at a node assumes that the preceding digits all match the current node's suffix, it only needs to keep a small constant size (b) entries at each route level, yielding a neighbour map of fixed constant size $blog_b N$.

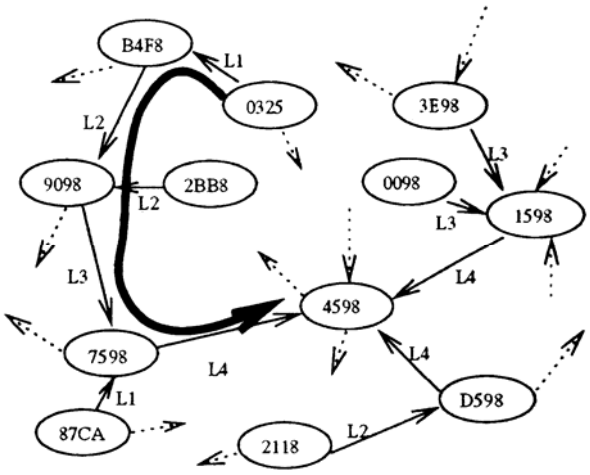


Figure 2.18: Tapestry routing example showing the path taken by a message originating from node 0325 and destined for node 4598 in a Tapestry network using hexadecimal digits of length 4 (65536 nodes in namespace)[33].

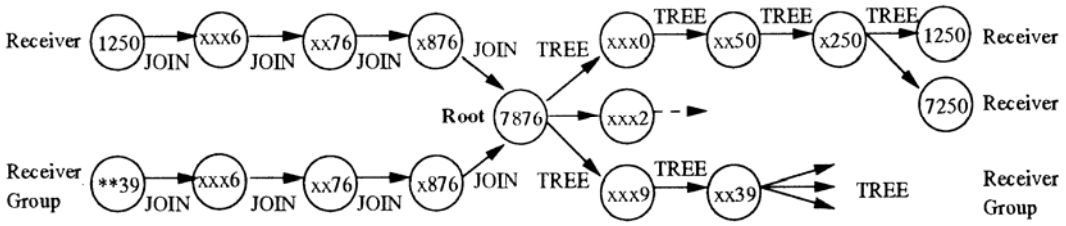


Figure 2.19: Bayeux Tree Maintenance[33].

Unlike ALM-CAN and ALM-DT, Bayeux has to explicitly build a source-based multicast tree on top of the Tapestry control overlay. Bayeux's data tree comprises both Tapestry nodes acting as software routers as well as the multicast receivers. A JOIN request by Receiver 1250 in Figure 2.19 to join a session is routed by Tapestry all the way to the root node (ID 7876 in Figure 2.19). Then the root records the identity of the new member and uses Tapestry to route a TREE message back to the new member. Every tapestry node along this route (xxx0, xx50, x250) adds the identity of the new member to the list of receiver node IDs that it is responsible for and updates its routing table. Requests to leave the group are handled similarly. Bayeux therefore requires nodes to maintain more group membership information

and it generates more traffic when handling group membership changes. In particular the root keeps a list of all group members and all group management traffic must go through the root which is not only a bottleneck but also a single point of failure. Bayeux proposes a multicast tree partitioning mechanism to ameliorate these problems by splitting the root into several replicas and partitioning members across them.

NICE [38] proposes a highly scaleable solution designed for low-bandwidth, data streaming applications with large receiver sets. It proposes an extremely low control overhead structure over which different low-latency data distribution paths can be built. The scaleability is achieved by organizing the members into a multi-level hierarchical control topology. Layers are numbered sequentially with the lowest layer being labelled Layer zero (denoted by L_0). Members in each layer are partitioned into a set of clusters. Each cluster is of size between k and $3k-1$ where k is a constant and comprises members that are close to each other. NICE uses latency between members as the distance metric used in organizing the overlay. Figure 2.20 shows NICE's hierarchy using $k = 3$. Note that all members are part of the lowest layer, L_0. A distributed clustering protocol at each layer partitions these members into a set of clusters with the specified size bounds. The protocol chooses a cluster leader which is the graph theoretic centre of the cluster in layer L_i to join layer L_{i+1}. The cluster leader therefore has the minimum maximum distance to all other members in the cluster. This allows new members to quickly locate its position in the hierarchy with minimum queries to other members. A new member is bootstrapped by a RP to the hosts in the highest layer (L_i) of the hierarchy. The joining host contacts these members and selects the one which is closest to it in

terms of latency. This selected node will inform the joining host about its peers in layer L_{i-1}. The joining host iteratively identifies the closest member in each layer until it locates its cluster in layer L_0. Hence NICE is a topology-aware design as it chooses overlay peers based on network locality although its data topology is implicit in the NICE hierarchy. The NICE members hierarchy is used to define both the control and the data topologies. Using cluster size of 4, the control topology is shown in Panel 0 of Figure 2.21. The edges indicate the peerings between group members on the overlay topology. Each set of 4 hosts arranged in a 4-clique in Panel 0 are the clusters in layer L_0. Hosts B_0, B_1, B_2 and C_0 are the cluster leaders of these four L_0 cluster and form the single cluster in layer L_1. Host C_0 is the leader of this cluster in layer L_1. In the control topology, all members of each cluster peer with each other and exchange periodic refresh messages incorporating the latencies between them. For example, the control path peers of A_0 in layer L_0 are all the members in its L_0 cluster (i.e. A_1, A_2 and B_0) while those of B_0 which belongs to layers L_0 and L_1 are all the other members of its L_0 cluster (i.e. A_0, A_1 and A_2) and L_1 cluster (i.e. B_1, B_2 and C_0). In addition, all members periodically probe members in its supercluster (defined as the cluster in the next higher level where its leader belongs to) to identify a cluster leader than its current one so that the members' hierarchy can be improved to adapt to changing network conditions. Cluster leaders are also responsible for splitting and merging clusters to satisfy the cluster size bound. NICE's worst case control overhead is $O(klogN)$ and the number of application level hops on the basic data path between any pair of members is $O(logN)$.

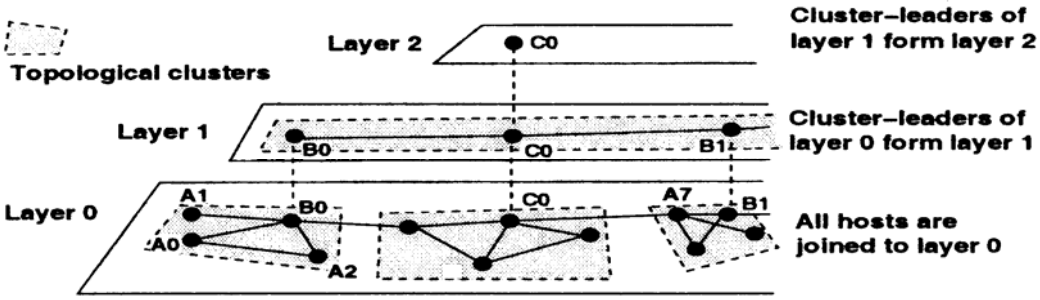
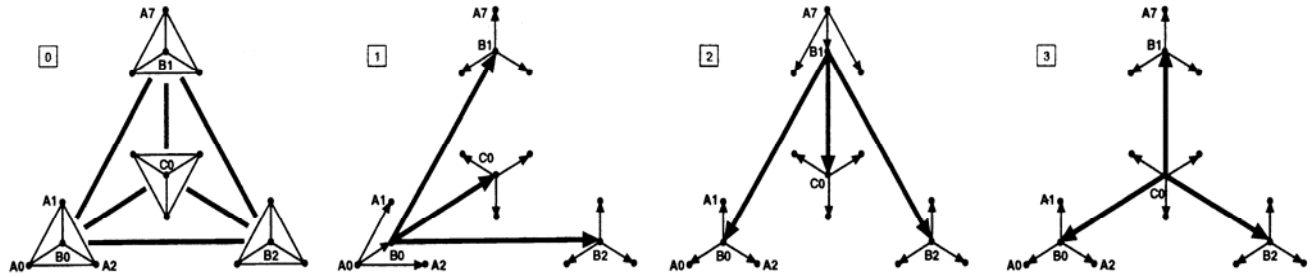


Figure 2.20: Hierarchical arrangement of hosts in NICE. The layers are logical entities overlaid on the same underlying physical network.

The data topology is an embedded source-specific tree defined by a forwarding rule on the control topology without the need to build it explicitly or use any complex routing algorithm. A source sends a data packet to all its peers in the control topology (See Panel 1 of Figure 2.21 where A_0 is the source and it forwards data to peers A_1, A_2 and B_0). A receiving host will only forward the data to its peers if and only if it is the cluster leader (See Panel 1 where cluster leaders B_0, B_1, B_2 and C_0 forward data to their respective peers). Panels 2 and 3 show different trees rooted at sources A7 and C0 respectively. Loops and partitioning of the data tree may occur due to stale cluster membership data and node failures respectively. No special mechanisms are introduced except for the members and cluster leaders to restore the hierarchical relationships and reconcile the cluster view for all members.



(a) Control topology (b) Data tree rooted at A_0 (c) Data tree rooted at A_7 (d) Data tree rooted at C_0

Figure 2.21: Control and data delivery paths for a 2-layer hierarchy. All A_i hosts are members of only L_0 clusters. All B_i hosts are members of both layers L_0 and L_1. The only C host is the leader of the L_1 cluster comprising itself and all the B hosts.[38]

SCRIBE [34]-[35] is a large-scale event notification system to disseminate data on topic-based publish-subscribe groups. It builds a source-based multicast shared tree on top of Pastry [72] which is a peer-to-peer location and routing overlay network in the Internet. Pastry uses a circular 128-bit namespace and assigns an ID based on this namespace by basing the nodeId on a secure hash of the node's public key or IP address. Given a message and a destination key, Pastry routes the message to the node whose node ID is numerically closer to the key. This is illustrated in Figure 2.22. Assuming a Pastry network of N nodes, the expected number of application level hops is $log_2^b N$ where b is a configuration parameter with a typical value of 4. With concurrent node failures, eventual delivery is guaranteed unless $l/2$ or more nodes with adjacent nodeIds fail simultaneously (l is an even integer parameter with typical value 16). NodeIds and keys can be interpreted as a sequence of digits in base 2^b. A node's routing table is organized into $128/b$ levels with 2^b entries in each level. The entry in column m at level n of a node p's routing table points to a node whose nodeId shares the first n digits with p's node Id and whose $(n+1)$th digit is m. The routing table entry is left blank if no such node is known. In addition, each node maintains IP addresses for the nodes in its leaf set. A leaf set is the set of l nodes that are numerically closest to the current node with $l/2$ being smaller and $l/2$ being bigger. It also maintains a neighbour set which have members that are close based on the distance metric. The uniform distribution of nodeIds ensures an even population of the nodeId space with only $log_2^b N$ levels in the routing table populated on average. Moreover, only (2^b-1) entries per level are populated as one of them is the local node. Hence the average state entries maintained by each node is only $(2^b-1)\ log_2^b N$ entries. Moreover, after a node failure or the joining of a new node, the

tables can be repaired by exchanging $O(log_2^b N)$ messages among the affected nodes. At each routing step, a node forwards the message to a node that shares a longer common prefix with the destination key than its own ID. When no such member is found in the routing table, the message is forwarded to a member in the leaf set that is numerically closer to the destination key than its own ID. Similar to Tapestry, Pastry exploits network locality to reduce routing delays by measuring the delay (RTT) to a small number of nodes when building the routing tables. For each routing table entry, it chooses the closest nodes whose nodeId satisfies the constraint for that entry. Since the constraints are stronger for the lower levels of the routing table, the average delay of each Pastry hop increases exponentially until it reaches the average delay between two nodes in the network.

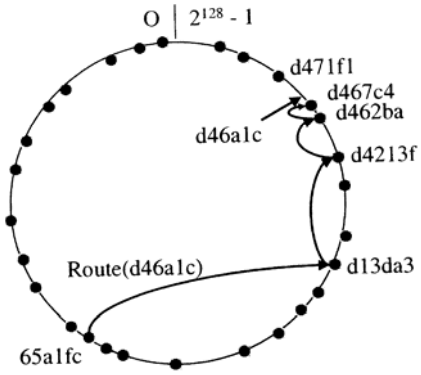


Figure 2.22: Routing a message from node 65a1fc with key d46a1c. The dots depict live nodes in Pastry's circular namespace[34].

The control topology of SCRIBE is essentially that of Pastry where the neighbours of any member on the control path include all in its routing table, neighbourhood set and leaf set entries. Neighbours exchange refresh messages with one another. A new node X, joining Pastry is bootstrapped to an existing node A to route a join message using X as the key. It is routed to the existing node Z with nodeId numerically closest to X. X obtains the leaf set from Z and the *ith* row of

the routing table from the *ith* node encountered along the route from A to Z thereby initializing its state and notifying neighbours of its arrival. In the event of a failed node, neighbours learn of its demise and update their leaf sets. The recovering node will contact the nodes in its last known leaf set, obtain their leaf sets and updates its own. Each multicast group in SCRIBE typically consists of a subset of the members that have already joined the Pastry control topology and has its own ID (called topic identifier). In other words, a member needs to be joined to the Pastry overlay in order to join a SCRIBE multicast group. SCRIBE creates a multicast tree, rooted at the RP as its data topology. The RP is the node with nodeId closest to the topic identifier and the tree is formed by joining the Pastry routes from each group member to the RP. A Pastry node who wishes to join a SCRIBE group simply routes a join message using the topic identifier as its destination key. All members on this path that are not already part of the multicast data delivery tree for the group add themselves to the tree. SCRIBE node therefore maintains a children table for the group containing an entry (IP address and nodeID) for each of its children in the multicast tree. Upon detection of a tree partition, the SCRIBE node will call Pastry to route a join request to its topic identifier. In the process, Pastry will route the message to a new parent thus repairing the multicast tree.

Freenet [76] is a distributed large-scale peer-to-peer (P2P) data store which provides information and strongly protects the anonymity of the authors and readers. It is considered a pioneer in P2P routing while Pastry, ALM-CAN and SCRIBE represent the second generation P2P overlay networks. The data and control topologies share the same distributed network of peer nodes. A Freenet node

maintains a dynamic routing table containing addresses of some number of nodes and the keys that they are likely to know and its own local datastore which it makes available to the network for reading and writing. Storing of data corresponding to these keys is optional. The basic operating model is such that a request for a key is routed through the P2P network by passing from neighbour to neighbour until it reaches its destination. A Freenet node neither cares nor knows whether the neighbour will forward the message to another node or whether the neighbour is the final destination or the original source of the message. Moreover, the contents of each file are encrypted and may be distributed piecemeal over many different nodes. It is therefore extremely difficult to know who actually hosts what file and whether a node is just forwarding a file or is the final destination. Herein lies the heuristic of Freenet to protect the anonymity of the users and publishers of the requested information.

Freenet adopts a key based routing protocol. Similar to Bayeux, files are identified by binary file keys obtained by applying the 160-bit SHA-1 hash function [70]. Unlike Bayeux and SCRIBE, the routing algorithm is heuristic with no special provision to guarantee successful content location. A request message will specify a key to the document to be located and a hops-to-live value. The latter arbitrary limits the maximum number of hops which means that there is no guarantee of locating the content. A node that receives a request for a key for which it does not know the exact location forwards the request to one of the nodes in its routing table that it thinks will know and whose keys are closer to the requested key. An illustration is shown in Figure 2.23 where a request from node *a* is passed to node *b* and then node *c*. At node *c*, a dead-end is encountered and the content is yet to be found. A failure

message is then sent by the node which fails to locate the desired content (node *c)* to its upstream node (node *b*) which will then try the alternate downstream node that is its next-best neighbour, node *e*, in its routing table. The protocol hence backtracks along the path the request has traversed (see step 3). In this way, a request operates as a steepest-ascent hill-climbing search with backtracking. The process repeats at each node until the content is found or the hop limit is exceeded. Loops are detected as can be seen from steps 5, 6, 7 and 8 in Figure 2.23. If the content is found, it will be backtracked to the originator. The intermediate nodes can choose to cache the document in their local data store. The same path finding process is used to insert a document by sending a request for the document. If the search fails, the document will be sent along the search path and get inserted into the network at the same place as the search path. If the initial search succeeds, the document has already exist and does not need to be added again. A new node who wishes to join the network will discover some existing nodes through out-of-band connection. As a new node has no information about the performance of other nodes, its routing of request is thus random. However, the routing tables will improve over time due to two reasons. Firstly, nodes should be specialized in locating sets of similar keys because a node listed in routing tables under a popular key will tend to receive mostly requests for similar keys. Moreover with backtracking, it will become better informed in its routing tables about which other nodes carry these keys. Secondly, nodes should become similarly specialized in storing clusters of files having similar keys. This is because forwarding a successful request will result in the node itself gaining a copy of the requested file, and since most requests will be for similar keys, the node will end up acquiring files with similar keys. Deletion of contents in the local data store and the routing table are carried out when the individual node runs out of space.

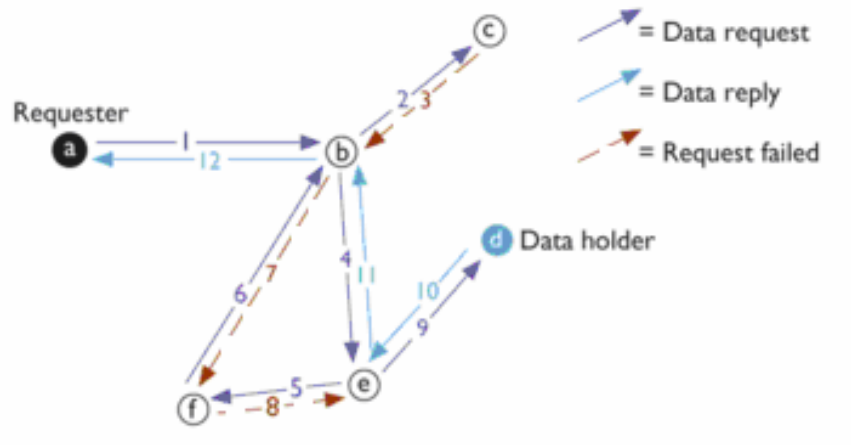


Figure 2.23: A typical request sequence in Freenet. The request moves through the network from node to node, backing out of a dead-end (step 3) and a loop (step 7) before locating the desired file[76].

As mentioned, keys are just hashes with no semantic relationship to the popularity of the content. Key closeness does not imply similar popularity of the respective contents indexed by the keys. Freenet uses Content Hash Key (CHK) and Signed Subspace Key (SSK). A CHK is a SHA-1 hash of a document and uniquely identifies the document. A node can check the authenticity of a document returned by hashing it and checking the digest key against the CHK. CHK also reduces data redundancy since the same data will share the same CHK. SSK is based on public-key cryptography and enables personal namespaces. A user creates a namespace by randomly generating a public/private key pair which serves to identify the chosen name space. To insert a file, another short, descriptive text string is used as before. The public namespace key and the descriptive string are hashed independently, XOR'ed together and then hashed again to yield the file key. The private half of the asymmetric key pair is used to sign the file and the file is encrypted by the descriptive text string. The digital signature of the document can be verified by any

Freenet node. To allow others to retrieve the file, the user publishes the descriptive string together with the subspace's public key. Storing data requires the private key and hence only the owner can add files to it.

2.2.4 Flat Topology versus Hierarchy Topology

A flat topology is one where there is only a single-layer overlay sitting atop the physical network. All nodes in the overlay are in the same logical level. A hierarchical design introduces hierarchy into the overlay forming a multi-layer overlay. Nodes form clusters at the lowest level where each cluster has a leader. The leaders in turn form clusters at the next level and so on, thereby creating an overlay at each level. Table 2.2 shows that Kudos and NICE are the only advocates of hierarchical topologies. The main advantage of a hierarchical overlay lies in its ability to scale and low management overhead since measure probes are run across smaller groups. This is achieved at the cost of efficiency as the children nodes in different clusters are unable to form overlay links to one another and there is additional overhead in maintaining the clusters. As explained above, Kudos builds a 2-level hierarchical overlay while NICE is a multi-level overlay. It is noted that TBCP introduces a form of hierarchy by organizing its members into different domains based on their domain identity so as to cluster members of the same domain into the same subtree to improve tree efficiency. However, as the domain roots do not form another layer of overlay among themselves, TBCP is thus still classified as a flat topology with a single-layer overlay.

Table 2.2: Flat versus Hierarchical topologies.

Flat			*Hierarchical*
ALMI	FreeNet	Overcast	Kudos
ALM-CAN	HBM	SCRIBE	NICE
ALM-DT	HM	Scattercast	
BTP	Narada	TBCP	
Bayeux	OMNI	Yoid	

2.3 Classification by Service Models

By service model, we refer to the manner in which data is being delivered to the multiple destinations, namely best-effort delivery versus reliable delivery and source-specific delivery model (one-to-many) versus any-source delivery model (many-to-many). The service model will determine the type of applications the various multicast techniques can support. For example, real-time broadcast streaming of multimedia data and certain collaborative applications such as audio and video conferencing are tolerant of loss and hence best-effort delivery will suffice while certain types of content distribution such as data replication services will require data integrity and therefore entails reliable transfer. Content distribution usually belongs to the source-specific model while collaborative applications use the any-source model.

2.3.1 Best Effort versus Reliable Transfer

Note that the definition of reliable transfer here is more rigorous and refers to end-to-end reliability. In other words, it includes more than just the use of TCP for data transfer. TCP only ensures hop-to-hop reliability which prevents losses due to transmission error and network congestion albeit at the expense of real-time delivery. To recover from error arising from delivery tree transitions and node failures, error recovery techniques such as data duplication and retransmissions must be incorporated. These techniques, as well as hop-to-hop TCP transfer impose a back pressure on the source as the source needs to handle the retransmission requests and rate-limit itself to avoid buffer overflow. In this respect, the receivers in application level multicast techniques can help to ease the pressure on the source if they possess buffering and/or rate-limiting capability as flow control and retransmissions can be handled locally.

It is noted without prejudice that all the proposals covered herein are capable of best effort delivery be it via TCP or UDP transfer with the former being deployed at the expense of real-time performance. Table 2.3 shows how the various application multicast solutions can be fitted into the different service models based on the above definitions. Only a handful of proposals, namely, ALMI, Scattercast, Bayeux and Overcast incorporated additional mechanisms and protocols to support end-to-end reliable transfer while the rest choose to focus on specific applications and the efficiencies of the overlays. Yoid has included sequence number in the packet header in the Yoid Delivery Protocol specifications for in-order delivery [58] but it has however not defined the protocols for lost packet recovery.

Table 2.3: Classification by data transfer reliability.

Best Effort Delivery				
Without Complete End-to-End Reliable Transfer Mechanisms				*Reliable Delivery*
ALMI	Freenet	Narada	TBCP	ALMI
ALM-DT	HBM	NICE	Yoid	Scattercast
ALM-CAN	HM	OMNI		Bayeux
BTP	Kudos	SCRIBE		Overcast

ALMI and Scattercast support data delivery via both UDP and TCP. In addition, ALMI proposes out-of-band connection direct to the source for re-transmission for error recovery and in cases where application has buffering capability, retransmission can happen locally. Scattercast uses SRM (Scaleable Reliable Multicast) [55] in addition to TCP for hop-to-hop reliability. To recover lost data, a receiver issues recovery requests which get forwarded from its local proxy (called SCX) through other SCXs towards the source until one of the SCX can recover the data either from itself or its local group. It uses a scheme based on PGM (PraGmatic Multicast) [56] and BCFS (Bread-Crumb Forwarding Service) [57] to limit the scope of retransmitted data so that data is only re-transmitted to those links which have requested for it. The links are kept as soft state in each SCX.

Bayeux leverages the redundant paths to every destination provided by Tapestry, an overlay location and routing layer presented by Zhao et. al. in [59] for end-to-end reliability. It proposes a suite of protocols which include 100% data duplication on

the first backup route to selective duplication based on the probability of successful delivery of each link. Other protocols do away with data duplication by either using heuristics to choose the best outgoing path for each packet or predicting the shortest and most reliable outgoing link to the next hop Tapestry router. Todate, Bayeux has adopted the latter approach to minimize overheads due to data duplication.

Overcast provides reliable data transfer to users via TCP over its overlay of proxy nodes (called Overcast nodes) that reside within the network infrastructure. Overcast node performs store-and-forward and failure recovery operations. It maintains a log of the data it has received and stored, and upon failure recovery, it resumes on-demand distribution from where it leaves off.

2.3.2 Source-Specific versus Any-Source Delivery

Table 2.4 shows the classification of the various techniques into source specific versus any-source models. The former builds data delivery tree rooted at a single source with one tree for each source while the latter builds a group-shared tree where any members in the tree can be a source. The shared tree is usually rooted at the dominant source though it can be rooted at any node as defined by the respective tree-building algorithms. Source-specific trees are simple and are known to have latency advantage over group-shared trees although routing states associated with source-specific tree grow linearly with the number of data sources as this approach needs to deal with the overhead of maintaining and optimizing multiple trees. This is however less of a concern on end hosts than on routers. On the other hand, group-

shared trees reduce routing inefficiency and incur less overhead in tree building and management but are not optimized for the individual source and are susceptible to a central point of failure.

Table 2.4: Classification by single-source versus any-source model.

Source Specific Model		*Any-Source Model*		
Bayeux	OMNI	ALMI	Freenet	TBCP
Kudos	Overcast	ALM-CAN	HBM	Yoid
Narada	Scattercast	ALM-DT	HM	
NICE		BTP	SCRIBE	

The service models adopted by the various proposals are largely influenced by the targeted primary applications. Bayeux, Narada, Kudos, NICE, OMNI, Overcast and Scattercast are aimed at media streaming and broadcast applications which are characterized by a single source. These proposals therefore fully exploit this characteristic to produce simple and efficient data delivery trees. Proposals such as ALMI are primarily targeted at collaborative applications where there can be more than one source per group while the rest are more generic in their service models. Freenet is classified under any-source model as its P2P overlay network functions like a shared tree where any peers can be the producer of the information requested. Hence, if the applications do not entail multiple sources, the proposals classified under the any-source model perform as per those in the source specific model. Any-source model is hence a superset of the single-source model.

2.4 Classification by Architecture

Architecturally, proposals for overlay based multicast can be distinguished by whether they assume a peer-to-peer architecture or an infrastructure (proxy) based architecture. A peer-to-peer (P2P) approach constructs the overlay across end-users with all functionalities being vested with these end-users. A proxy based approach makes explicit use of infrastructure service agents or proxies or dedicated servers strategically deployed within the network to provide efficient data distribution and value-added services to a set of end hosts. The overlay is built across these proxies and end hosts attach themselves to proxies with the proxies operating on their behalf.

The diverse application layer techniques can also differ in the approaches adopted for the overlay creation and maintenance. This will determine the presence and role of a central controller in their architectures. The functions of this controller include being a centralized tree construction server, a centralized directory of all members, managing group membership such as join and leave processes and performing partition recovery.

2.4.1 Peer-to-Peer versus Proxy Support

P2P architectures have the attractive property of scaleability. This is because being distributed, each peer needs only to keep state for a small number of peers. Its other advantages include the simplicity of set-up and deployment, its resource sharing capability as well as dynamicity. Their vast combined resources such as physical connectivity, computing resources may be heterogeneous but they can be individually harnessed according to their capacities. P2P systems thus provide redundancy as a single failure will not radically affect the big group. Peers can be dynamically deployed in large numbers and at hot spots quickly with minimum prior configuration.

The merits of using proxies as opposed to any end hosts lie mainly in their functional dedication. Being dedicated, homogeneous and better provisioned than individual hosts, proxies are thus more reliable and robust to failure. Proxies are persistent beyond the lifetime of individual hosts. They are more intelligent than end-hosts as they can provide value-added services such as being pre-configured with application specific components to render them application-aware. In addition, they can be positioned at strategic positions such as co-locating with IP routers or at hotspots to provide more efficient services. However, there may be problems with the acceptance and deployment of such proxy, and when compared to P2P systems, it is less responsive to changing environment conditions as proxy placement is usually static and has to be manually deployed. Table 2.5 distinguishes the P2P systems from their proxy based counterparts.

Table 2.5: Classification by the need for proxy support.

Peer-to-Peer Based		*Infrastructure Support*
ALMI	HM	Bayeux (uses Tapestry nodes)
ALM-DT	Narada	OMNI (uses Multicast Service Nodes, MSNs)
ALM-CAN	NICE	Overcast (uses Overcast nodes)
BTP	SCRIBE	Scattercast (uses ScatterCast proXies, SCXs)
Freenet	Yoid	
Kudos	TBCP	
HBM		

OMNI deploys proxies called Multicast Service Nodes (MSNs) across the network. It is aimed at large scale data distributions where there is a single source e.g. a webcast. Prior to the commencement of the webcast, the MSNs will organize themselves into an initial data delivery tree where clients join and leave the tree dynamically. The root MSN is the one that is connected to the source. This tree of MSN will transform itself to adapt to changing network conditions. Overcast organizes a set of proxies called overcast nodes into a distribution tree rooted at a central source for single source reliable multicast. A distinguishing feature of Overcast is the explicit provision of permanent storage capacities in the network fabric accomplished through its proxies to support applications such as broadcasting and reliable data-delivery applications where delay is not a concern. Scattercast uses its ScatterCast proxies (SCXs) to build an intelligent, application-aware proxy infrastructure to provide a customizable framework for efficient broadcasting. Application specific components such as rate adaptation are incorporated into these

SCXs to provide more application functionality. SCX also serves to repair partitions in the overlay network. Clients are either statically configured with the location of their closest SCX or locate their SCX via a registry or dynamically via DHCP. Bayeux has a unique feature in that the overlay built comprises both end hosts as well as infrastructure nodes called Tapestry nodes [59]. These reliable servers are installed across the network to act as software routers with multicast functionality.

A point to note is that for HM, only designated members (DM) are involved in the building of the shared tree. The DM can be a single host or a member which is elected to act on behalf of the members in a multicast island. The DM communicates with its island members via local multicast. Hence, some researchers may consider DMs as proxies and classify HM as a proxy-based system. However, in this thesis, we classify HM as a peer-to-peer system as the DMs are not dedicated server nodes but are specially designated end hosts. Similarly, HBM classifies clients into Core Members (CM) and Non-Core Members (NCM) where NCM are nodes which are deemed unstable due to poor connection. CM form the transit nodes on a HBM data distribution tree while NCM become the leaf nodes. The CM are therefore any ordinary peer nodes and are not dedicated proxies.

2.4.2 Centralized Controller, Distributed Approach or Hybrid

A centralized approach to the overlay tree creation problem refers to the vesting of all responsibilities for group management, overlay computation and optimization with a central controller. The controller maintains group information, handles group

membership, collects measurements from all members, computes optimal distribution tree and disseminates the routing tables to all members. The main advantage of a centralized controller is in the great simplification of the routing algorithms, the more efficient and simpler group management and the provision of a reliable mechanism to prevent tree partitions and routing loops. However, the centralized nature limits its scaleability and poses other reliability problem as it is more susceptible to a central point of failure while it helps to alleviate problems of tree partitions and looping.

As the name implies, distributed approach is receiver-based and it distributes the responsibilities of group membership and overlay topology computation to the individual nodes of the overlay network. It is therefore relatively more robust to failure as failure of an individual node will not impact the entire group. Having no central controller as a potential bottleneck, the distributed approach is thus more scaleable. However, a fully distributed approach causes excessive overheads and is not as optimal and efficient in building optimal overlay.

A hybrid approach is one where a lightweight controller is still used to facilitate some of the group management (e.g. join process), overlay construction and error recovery functions while the actual overlay construction is left to a distributed algorithm. Table 2.6 shows the classification of the techniques via their incorporation of central controller into the architecture.

ALMI and HBM are by far the only two techniques which advocate a completely centralized solution where a single centralized host has total knowledge

of group membership as well as of the characteristics of a mesh connecting the members. New member only needs to learn of the controller from either online or offline means such as URL, email, etc and simply issues a join request to the controller. Mesh characteristics are collected from periodic active probes sent by members to measure application level performance metric which in this case is round trip time. Based on this knowledge the centralized host called session controller in ALMI and *RP* in HBM, builds a minimum spanning tree and disseminates the routing tables to all the members. The use of a centralized approach simplifies the tree building process and enables the ALMI and HBM data distribution trees to be close to source-rooted IP multicast trees in efficiency with low performance tradeoff. Moreover, it makes for better reliability as tree partition and looping can be avoided and reduces overhead during a change of membership or recovery from node failure as these are more effectively managed by a central entity. However, the centralized nature of this architecture limits its scaleability and hence it is targeted at collaborative applications and scales to large number of sparse groups. The session controller in ALMI and the *RP* in HBM also constitutes a single point of failure.

Those proposals classified under distributed approach in Table 2.6 adopt a distributed algorithm for overlay construction and dispense with any centralized controller or server. However, they do require some form of bootstrap mechanisms for a host to learn of some nodes in the multicast group or to learn of the root of the data delivery tree. Bayeux advertises the root nodes of its data delivery trees in the network via Tapestry's location services and its root nodes handle all join and leave requests. BTP, Kudos, Narada and Freenet assume the availability of some out-of-

band bootstrap protocol. Overcast nodes (proxies) who wish to join the overlay is bootstrapped via a global well known registry. Clients who wish to join the overcast session discover the root via a web registry. The root node selects the best overcast proxy to serve the clients based on the status information in its database and performs the redirection. SCRIBE requires a *contact node* in the overlay to bootstrap the join process while NICE and TBCP need a rendezvous point to learn of the highest level nodes in NICE's hierarchy and to advertise the root of the TBCP's shared tree respectively. Scattercast provides a well-known list of rendezvous SCXs for a new SCX to bootstrap itself and then relies on gossip-style discovery to locate other SCXs. For clients joining a Scattercast session, they are statically configured with the location of its closest SCX. ALM-CAN requires a bootstrap node which maintains a partial list of members. HM employs a dedicated Host Multicast Rendezvous Point (HMRP) where new members can learn about the roots of the shared trees and bootstrap themselves. The roots stored in the HMRP are kept current through periodic refreshes.

ALM-DT and Yoid are examples of a hybrid approach whereby a light-weight controller is incorporated in the architecture while the overlay construction algorithms are distributed. ALM-DT and Yoid use a centralized server as a rendezvous point for new members to join the group and to recover from overlay partition when a member leaves the group. Membership status is maintained at these rendezvous points. OMNI operates quite differently from the other techniques. As described in Section 2.3.1, OMNI builds an overlay across its proxies called MSNs. Prior to a session, OMNI must execute a centralized initialized procedure. The source will first be linked to a MSN identified by some bootstrap mechanism. This

root MSN gathers the latency measurements between itself and the other MSNs and use this knowledge to build the initial data delivery tree and distributes the topology to the member MSNs. This centralized computation of the tree building algorithm does not impact the data delivery as it operates off-line before data delivery commences. The subsequent optimization process for the data delivery tree is performed in a distributed manner without central intervention.

Table 2.6: Classification by centralized, distributed or hybrid approach.

Centralized	*Distributed*		*Hybrid*
ALMI	ALM-CAN	Narada	ALM-DT
HBM	Bayeux	NICE	OMNI
	BTP	Overcast	Yoid
	Freenet	Scattercast	
	HM	SCRIBE	
	Kudos	TBCP	

2.5 Performance Comparison

As the application layer multicast techniques reviewed here vary widely in their goals, designs, performance evaluation metrics and evaluation strategies, it is not possible to evaluate all of them comparatively. Neither is it possible to examine all the performance evaluation metrics exhaustively. Instead, this thesis focuses on the few more common evaluation metrics adopted and will provide comparative data wherever available. The evaluation metric covered here includes scaleability

measured intuitively in terms of the size of the multicast receivers it can support, the protocol efficiency in terms of the quality of data paths measured, control overheads, amount of state information to be maintained at each member node and failure tolerance.

2.5.1 Scaleability

The scaleability of a solution is limited by the protocol efficiency which comprises factors such as control overheads, time and resources used to construct the application layer multicast overlay, the amount of state kept by each node to maintain the overlay. An intuitive but simple approach which can provide a quick insight into the scaleability of each technique is via the number of receivers each of the proposals can support. This data is readily provided in all the research proposals. Table 2.7 summarizes their scaleability. However, a point to note is that a system which scales to small group may not necessarily be viewed as inferior to one which scales to a large group as each can serve the different needs of different applications. The group size supported is banded into the following:

- Small (several tens of members)
- Medium (several hundreds of members)
- Large (several thousands of members)
- Very Large (tens of thousands of members)

Table 2.7: Scaleability comparison in terms of the group size supported.

Small	*Medium*	*Large*	*Very Large*
ALMI	BTP	ALM-CAN	ALM-DT
HBM	HM	Bayeux	Freenet
	Narada	Kudos	NICE
	TBCP	OMNI	
	Yoid	Overcast	
		Scattercast	
		SCRIBE	

2.5.2 Efficiency of Protocols

2.5.2.1 Performance Metrics

There are many different parameters for evaluating the efficiency of the various proposals. Here we are interested in the quality of data paths generated, the amount of overhead network traffic that is attributable to overlay maintenance, network probing and other control functions, amount of state maintained by each node and the robustness to failure.

The quality of data path is commonly evaluated using two metrics:

- *Stretch* also termed as *Relative Delay Penalty (RDP)* is a measure of the increase in latency that applications incur while using overlay routing. It is

the ratio of a protocol's unicast routing distances to IP unicast routing distances. Assuming symmetric routing, IP multicast and naïve unicast both have a *Stretch* or *RDP* of one.

- *Stress* is a measure of how effective a protocol is in distributing network load across different physical links. It is defined on a per link or per node basis and counts the number of identical packets sent by the protocol over that link or node. IP multicast has no redundant packet replication and hence has a *stress* of one while naïve unicast has a worst case stress equal to the number of receivers. *Stress* is also called *Normalized Resource Usage (NRU)* as it is a measure of the additional network resources consumed by application level multicast techniques compared to IP multicast.

In general, it is difficult to analytically compute and quantify the stretch or stress metrics for most protocols as most of these results are obtained either via simulation or empirically and they are all presented graphically. The performance also varies according to group sizes and the characteristics of the underlying topology. Nevertheless, by examining their plots, all of them have shown reasonable to good performance in either stretch or stress or both depending on their optimizing goals. To provide some feel on the quality of the data path, the *Path Length* measured in terms of the number of application-level hops and the *Outdegree* of a node on the data delivery tree are metrics which are indirectly related to the stress and stretch metrics and can be easily analysed for some protocols. The *Path Length* is an indicator of the stretch of the protocol while the *Outdegree* provides an upper bound

for the number of times that a data packet is forwarded at a node which contributes directly to the stress of the entire overlay.

Protocol Overhead refers to the total bytes of non-data traffic that enters the network. This overhead includes control and maintenance traffic required to maintain the overlay and state management as all application layer multicast solutions require nodes to exchange refresh messages with its neighbours, the probe traffic and other active measurements performed during the overlay self-organization and maintenance process. The average overhead per node is commonly used as an indicator of the scaleability of the protocol.

Another metric that is closely related to protocol overhead is the amount of information kept by each node on the overlay. Herein lies their relation. *State Information* maintained in each node has to be periodically updated to reflect dynamically changing network environment. This implies that a node maintaining more state information will generate more update messages, thereby contributing to additional protocol overhead. This metric therefore directly impacts the scaleability of a technique. Information maintained by each node includes routing tables and group state. Protocols which require the full set of member state to be stored in each node are thus not as scaleable as those which only maintain partial group member state.

Failure Tolerance is actually a multi-factorial function, being dependent on the various mechanisms put in place for error and failure recovery; whether the system is proxy-based or peer-to-peer based; whether the system relies on dedicated lower-

level infrastructure support; whether the protocols are prone to a single point of failure or the failure can be contained and localized. Section 2.2 has detailed the mechanisms put in place for error and failure recovery in the various overlay topology designs while Section 2.4.1 has addressed the facts that proxy-based system is inherently more robust than peer-to-peer based system. It is acknowledged that application level multicast techniques are certainly not as robust as one based on IP multicast with lower-level dedicated infrastructure support since the end-hosts are not as well provisioned, persistent and reliable as the infrastructure set up by the Internet Service providers. This fact, notwithstanding, to facilitate a single-dimension evaluation of application level multicast solutions without being intertwined into the convolutions of the different impacting factors listed above, *Failure Tolerance* is simply herein defined as whether the protocols are prone to a single point of failure or the failure can be contained and localized. It is important to highlight that the single point of failure in the context of application level multicast protocols is not as catastrophic as it sounds. This vulnerable point refers to the centralized node or dedicated controller used by the protocols to manage overlay construction and membership. It does not refer to the source of the multicast session as in the event that the source fails, there will be no session at all. Hence, the failure of this centralized point only prevents new members from joining the groups but does not incapacitate the entire system. Its impact on members who have already joined the groups is minimal or none depending on the respective algorithms. Hence all members and sessions in progress continue to function. Such protocols will generally strengthen this critical point with server clusters or replicated servers to minimize the probability of failure and to recover as soon as possible. Localize

failure refers to failure which only impacts the failed node and its immediate neighbours and does not impact any join processes.

2.5.2.2 Comparison and Evaluation

Table 2.8 shows a summary of the performance of the various protocols in terms of the performance evaluation metrics spelt out above.

Stretch: As discussed in section 2.5.2.1, *path length*, which is derived from the number of application level hops, provides a good indication of the relative maximum *Stretch* of the various techniques. From Table 2.8, it can be observed that the number of application level hops is only bounded for those techniques which use embedded structure to construct their overlay. The pathlength of Freenet is limited by the hops-to-live value specified in the information request message. However, this does not guarantee that Freenet will be able to locate the requested information within the specific number of hops. Tree and mesh-tree techniques possess unbounded hops and hence are likely to have longer maximum stretch than their embedded structure counterparts. The exception is OMNI which predetermines the number of nodes during its initialization phase prior to data delivery.

Stress: Outdegree provides an upper bound for the number of times that a data packet is forwarded at a node which contributes directly to the stress of the entire overlay. Similar to the performance for stretch, embedded structure has an intrinsic bound for *maximum outdegree per node* except for Freenet where the outdegree is unconstrained since each node can incrementally discover more nodes when it lies in

the routing path of a successful information retrieval process. On the other hand, tree and mesh-tree approaches have to impose a degree constraint to minimize stress and optimize bandwidth. This constraint can take the form of a static user defined outdegree (e.g. ALMI, HBM, HM, Yoid) or a more dynamic one where users (Narada) or proxies (OMNI, Scattercast) set the degree constraint according to its prevailing bandwidth. The exception is BTP and Overcast where there is no degree constraint imposed.

Protocol Overhead and State Information Maintained. The overhead introduced by the protocol in the form of non-data traffic entering the overlay network is normalized as the a*verage control overhead per node* in Table 2.8 to facilitate comparison among the different techniques. This is also directly related to the *number of state entries per node* as the more states a node keeps, the more updates are required to maintain its currency thereby impacting the amount of control information. needed to maintain, improve and repair the overlay. These two metrics actually affect the scaleability of the systems and as reflected in Table 2.8, the highly scaleable systems with embedded structures have a non-linear bound on these control and state overheads. In other words, the overheads do not increase proportionately with the increase in the number of nodes (e.g. Bayeux, SCRIBE). In fact, such overheads and state data are constant in the case of ALM-CAN, ALM-DT and NICE which is a primary reason for their superior scaleability. Mesh-tree systems such as Narada fare the worst as it has to maintain state for the entire group. Likewise for centralized algorithm based ALMI and HBM as well as unconstrained outdegree protocols such as BTP, Overcast and Freenet. Whereas, the other tree-

based algorithms with their imposed outdegree constraint, manage to cap these overheads to an upper bound.

Failure tolerance. The fully centralized ALMI and HBM as well as most hybrid systems, namely, ALM-DT and Yoid are considered less robust due to their reliance on some form of centralized entity for group management functions and recovery from data distribution tree partitioning. To alleviate the impact of a failure in the centralized entity, Yoid proposes the use of a cluster of servers while ALMI uses multiple hot standby back-up servers. These standbys receive periodic updates from the primary controller so that they are ready to phase into operation once the primary controller fails. These approaches however, often suffer from communication overheads and data consistency problems. OMNI, unlike its hybrid counterparts, does not face similar problems as its centralized entity is only involved in building the data distribution tree during the initialization phase. Its operation ceases once data distribution commences.

It is interesting to note that distributed techniques too have their Achilles heels such as the roots of the data distribution trees in the case of HM, Bayeux and Overcast. In these cases, the root handles all join and leave requests from session members. To ameliorate the problem of the root as a single point of failure, root replication is incorporated. Bayeux creates multiple root nodes and leveraging on Tapestry's location services, receivers will be routed to the closest local root in network distance. Receivers are therefore partitioned into disjoint membership sets transparently. The technique is similar to root replication used by many existing IP multicast protocols such as CBT [73] and PIM [74] [75]. Bayeux's root replication

does not incur additional overhead in the sense that these replicated roots do not need to send periodic advertisements to all receivers as they can be located via Tapestry location services. Overcast uses a technique called linear roots to replicate its root. Starting with the root, some number of nodes are configured linearly i.e. each node has only one child. All the other overcast nodes lie below this set of linear roots. These replicated roots are updated with all the information needed to handle the join operations in the event that the root node fails. The drawback of this technique is the increased latency of content distribution as data has to traverse all these extra nodes before reaching the rest of the overcast nodes as well as data consistency problems. HM proposes a cluster of servers to alleviate the problem of failure in a single-node HMRP.

2.6 IP Multicast vs Application Layer Multicast

This section provides a qualitative comparison of IP multicast versus application layer multicast. As this is a vertical comparison of multicast implementation across the protocol stack, a meaningful comparison can only be made from the basic architectures which characterized the two implementations. The similarities and differences will be highlighted. A qualitative performance evaluation of the implementation in terms of protocol efficiency, protocol overhead, failure tolerance, support for higher level functionality and ease of deployment is conducted.

Table 2.8: Summary of protocol performance.

Techniques	Path Length	Max. OutDegree	Avg. Control Overhead per Node	No. of state entries per Node	Failure Tolerance	Group Size
ALMI	Max: unbounded	User defined	*O(N)*	*O(N)*	Single Pt of Failure at central controller	Small
ALM-CAN	Max: [1]*O(d $N^{1/d}$)* Avg: *d/4* $N^{1/d}$	Constant of *2d*	Constant of *2d*	Constant of *2d*	Localized Failure	Large
ALM-DT	Avg: $\sqrt{N/4}$	Worst: [2]*O(N-1)* Avg: approx. *6*	Constant of *6*	Constant of *6*	Single Pt of Failure at DT server	Very Large
Bayeux	Max: *O($\log_b N$)* [3]	*O($\log_b N$)*	*O($\log_b N$)*	*O(b $\log_b N$)*	Single Pt of Failure at Root	Large
BTP	Max: unbounded	Unconstrained	*O(N)*	*O(N)*	Localized Failure	Medium
Freenet	Max: Hops-To-Live	Unconstrained	*O(N)*	*O(N)*	Localized Failure	Very Large
HBM	Max: unbounded	User defined	*O(N)*	*O(N)*	Single Pt of Failure at RP	Small
HM	Max: unbounded	User defined	*O(max. deg.)*	*O(max. deg.)*	Single Pt of Failure at RP	Medium
Kudos	Max: unbounded	*O($\sqrt{N}$)*	*O($\sqrt{N}$)*	*O($\sqrt{N}$)*	Localized Failure	Large
Narada	Max: unbounded	User defined to reflect bandwidth of user's outgoing link	*O(N)*	*O(N)*	Localized Failure	Medium
NICE	Max: *O(log N)*	[4]Between *k* and *3k*	Constant of *O(k)* Max: *O(klogN)*	Constant of *O(k)* *O(logN)* only for the highest hierarchical layer	Localized Failure	Very Large
OMNI	Max: *O(logN)*	MSN defined to reflect bandwidth of MSN's outgoing link	*O(max. deg+logN)*	*O(max. deg.+logN)*	Localized Failure	Large
Overcast	Max: unbounded	Unconstrained	*O(N)*	*O(N)*	Single Pt of Failure at Root	Large
Scattercast	Max: unbounded	User defined to reflect bandwidth of user's outgoing link	*O(max. degree)*	*O(max. degree)*	Localized Failure	Large
SCRIBE	Avg: *O($\log_2{}^b N$* [5]*)*	*O($\log_2{}^b N$)*	*O($\log_2{}^b N$)*	*O((2^b-1)$\log_2{}^b N$)*	Localized Failure	Large
TBCP	Max: unbounded	User defined	*O(max. degree)*	*O(max. degree)*	Localized Failure	Medium
Yoid	Max: unbounded	User defined	*O(max. degree)*	*O(max. degree)*	Central Pt of Failure at RP	Medium

[1] d is the dimension of the Cartesian coordinate space of a CAN overlay
[2] Theoretically, worse case is created when N-1 vertices form a circle and the nth vertex is in the center of the circle. However, the maximum outdegree is very small [30].
[3] *b* is the base used in the Bayeux node ID
[4] k is a constant
[5] *b* is a configuration parameter used in SCRIBE with a typical value of 4

NB: N is the number of nodes in the overlay

Table 2.9: Summary of the properties of the various milestone application level multicast techniques.

Techniques	Goals	Topology-Awareness	Topology Design	Topology Hierarchy	Delivery Type	Source Model	Proxy	Type of Algorithm
ALMI	Min. tree cost	Aware	Tree	Flat	Reliable	Any-Source	P2P	Centralized
ALM-CAN	Min. latency	Agnostic	Embedded	Flat	Best Effort	Any Source	P2P	Distributed
ALM-DT	Min. overhead, max. scaleability	Agnostic	Embedded	Flat	Best Effort	Any Source	P2P	Hybrid
Bayeux	Fault tolerant delivery, min. latency	Aware	Embedded	Flat	Reliable	Specific	Tapestry Nodes	Distributed
BTP	Min. tree cost	Aware	Tree	Flat	Best Effort	Any Source	P2P	Distributed
Freenet	Anonymity of authors and readers	Agnostic	Embedded	Flat	Best Effort	Any Source	P2P	Distributed
HBM	Min. tree cost	Aware	Tree	Flat	Best Effort	Any Source	P2P	Centralized
HM	Min. tree cost	Aware	Tree	Flat	Best Effort	Any Source	P2P	Distributed
Kudos	Max. scaleability	Aware	Mesh-Tree	2-level Hierarchy	Best Effort	Specific	P2P	Distributed
Narada	Min. latency, Max. bandwidth	Aware	Mesh-Tree	Flat	Best Effort	Specific	P2P	Distributed
NICE	Min overhead, max. scaleability	Aware	Embedded	Multi-level Hierarchy	Best Effort	Specific	P2P	Distributed
OMNI	Min. latency	Aware	Tree	Flat	Best Effort	Specific	MSNs	Hybrid
Overcast	Reliable delivery, Max. bandwidth	Aware	Tree	Flat	Reliable	Specific	Overcast Nodes	Distributed
Scattercast	Min. latency	Aware	Tree	Flat	Reliable	Specific	SCXs	Distributed
SCRIBE	Max. scaleability	Aware	Embedded	Flat	Best Effort	Any Source	P2P	Distributed
TBCP	Max. responsiveness	Aware	Tree	Flat	Best Effort	Any Source	P2P	Distributed
Yoid	Min. latency	Aware	Tree-Mesh	Flat	Best Effort	Any Source	P2P	Hybrid

2.6.1 Comparison of Basic Architecture

IP multicast utilizes a tree topology for its data distribution where routers constitute the nodes of the distribution tree. Hence, it has full knowledge of the network topology. When a host joins or leaves a group represented by a multicast group address, it informs its designated multicast router in its subnetwork. The participating multicast routers then exchange group membership information and use multicast routing protocols to build multicast trees to deliver data. Well-known IP multicast routing protocols include Distance Vector Multicast Routing Protocol (DVMRP) [18], Multicast Open Shortest Path First (MOSPF) [102], Core-Based Trees (CBT) [19], Protocol Independent Multicast (PIM) with Dense and Sparse Modes [17] and SSM [26]. Among these protocols, DVMRP, MOSPF, PIM-DM and SSM construct source based trees, whereas CBT and PIM-SM utilize a core or Rendezvous Point (RP) to organize any-source multicast trees. PIM-SM can also support per-source trees. The former group can be considered to be based on distributed algorithms where shortest paths are used to build the distribution tree while the latter group is based on centralized algorithms, where the special elected node has global visibility of the multicast tree, and runs a centralized algorithm to compute the best multicast tree. The data delivery model is best effort.

Application layer multicast techniques (refer to Summary of properties shown in Table 2.9), except for some of those using the embedded structures, also build data distribution trees. The difference lies in that the tree nodes are either strategically deployed proxies and/or mere end hosts instead of network routers. Since these data distribution trees are overlays, sitting on top of the underlying Internet, each branch

of the data distribution tree corresponds to the unicast path between two proxies/end hosts in the substrate network. Being overlays, application layer multicast does not have direct and full knowledge of the network topology. Instead, it relies on a host of measurements to glean partial knowledge of the underlying network while some such as ALM-CAN and ALM-DT are topology-agnostic. The data delivery model can be reliable or best effort. Overlay distribution trees can be source-specific or shared trees just like IP multicast depending on the routing protocols used. Similar to IP multicast, routing protocols can be distributed, centralized or hybrid.

Table 2.10 summarizes the similarities and differences between IP multicast and application layer multicast.

Table 2.10: Architectural comparison of IP multicast and application layer multicast.

Properties	IP Multicast	Application Layer Multicast
	Differences	
Topology-Awareness	Direct knowledge of underlying network topology.	Indirect knowledge of network acquired through probes and measurements.
Topology Design	Tree based with network routers as tree nodes.	Tree-based, tree-mesh or embedded structures with proxies and/or end hosts as nodes.
Delivery Type	Best effort.	Can be best effort or reliable.
	Similarities	
Topology Design	Tree	Mostly tree or tree-mesh.
Source Model	Can be source-specific or shared trees.	Can be source-specific or shared trees.
Type of Algorithm	Can be centralized or distributed.	Can be centralized or distributed or hybrid.

2.6.2 Performance Comparison

2.6.2.1 Protocol Efficiency

As mentioned in Section 1.1, IP multicast is considered the most bandwidth efficient for multipoint communication. By virtue of the tree topology used in IP multicast, only one copy of the data traverses all branches of the tree meaning that there are no redundant packets delivered over each physical link. Hence, there is only one source stream traversing the router network, regardless of the number of receiving clients in a multicast session. This not only greatly improves network bandwidth efficiency but also reduces the server load compared to naïve unicast, hence allowing IP multicast to scale to large group size. Moreover, the knowledge of the network topology allows IP multicast to build highly optimized tree which renders it much more efficient in terms of end-to-end delay compared to application layer multicast.

Application layer multicast which builds overlay trees comprising proxies and/or end hosts, is certainly not as bandwidth efficient as IP multicast. Although data delivery tree is also used, the tree branches on the overlay are basically unicast paths between the tree nodes. These tree branches are incongruent to the underlying physical network links unlike those in the IP multicast tree. A data stream traversing an overlay tree will hence result in multiple streams across the same physical network links. Although this is still very much better than naïve unicast in terms of bandwidth usage and server loading, it nevertheless increases the network load compared to IP multicast. In addition, the lack of knowledge about underlying

network topology usually results in less efficient overlay tree thereby incurring higher end-to-end latency.

2.6.2.2 Protocol Overhead

IP multicast requires routers to maintain separate routing state for each multicast group which introduces high complexity and serious scaling constraints at the IP layer especially in the presence of a large number of multicast groups [77]. These state tables consume router memory and slow down data forwarding as each packet forwarding involves a routing table lookup. Application layer multicast incurs similar protocol overhead in maintaining group membership and multicast trees among the end hosts. However, the latter is even more overhead-intensive as the lack of network topology knowledge requires it to carry out expensive measurements to acquire network data so as to be topology-aware.

2.6.2.3 Failure Tolerance

IP multicast relies purely on the network infrastructure, namely the routers, to perform group management and data delivery. Being infrastructure based, dedicated and better provisioned, the routers are much more reliable and robust to failure. The proxies used by application layer multicast can be equally dedicated and well provisioned but the same cannot be said of the end hosts. End hosts are less stable and they can leave or fail any time. Hence a robust mechanism has to be put in place to recover from partitions which have resulted from end host failure or departure.

2.6.2.4 Support for Higher Level Functionality

In this aspect, IP multicast is inferior to and much less versatile than application layer multicast. The former provides only best effort and unreliable service. Moreover, it allows any source to send data to an arbitrary group which renders the network vulnerable to flooding attacks by malicious sources. IP multicast requires every group to dynamically obtain a globally unique multicast address which makes it unscaleable and inconsistent especially with distributed multicast groups. To provide higher level functionalities, such as end-to-end reliability, congestion control, access control and security, to support the different applications poses significant challenges in IP multicast [78][79].

Application layer multicast, on the other hand, can easily take advantage of its proxies (in the case of distributed algorithm) or centralized servers (in the case of centralized and hybrid algorithms) to provide complex processing for access control, authentication, reliable transfer, error recovery, flow and congestion control, scheduling, differentiated message handling, data transcoding etc. on an application specific basis. As all packets are transmitted using unicast, solutions to provide some of these higher level functionalities can be simplified by leveraging well understood unicast solutions for these problems. Moreover, relaying proxies and end hosts on the overlay trees can provide additional resources such as computing power, data storage and data buffering capabilities, which facilitate streaming applications. Application layer multicast can be deployed easily, allows much flexibility in customizing the protocols to suit the different applications and does away with

consistent global naming of its multicast groups to better support application specific naming.

2.6.2.5 Ease of Deployment

Despite the bandwidth advantage, IP multicast is still not being widely deployed Internet-wide as discussed in Section 1.1. The bandwidth benefit is far outweighed by the lack of a scaleable inter-domain multicast routing protocol, the demanding requirement of global deployment of multicast-enabled routers, the reluctance of ISP to enable multicast for fear of security vulnerabilities and multicast traffic flooding, the control complexity and state scaleability associated with group setup and maintenance at the routers and the difficulty of member access control [22][78] [79][80].

However, IP multicast is successful in intra-domain deployment such as enterprise network environments, campus networks etc. to support multi-receiver applications which include video conferencing, executive briefings, product rollouts, e-learning, radio and video broadcast. This success is due to the fact that the intranets can be better controlled and administered and are simpler to manage given its relatively small group membership relative to the entire Internet. Companies such as Microsoft supplies Windows Media Series, an end-to-end platform to create, publish, manage and deliver secure digital media content for enterprise applications such as broadcast communications, e-learning, sales and marketing using IP multicast and unicast [81]. Cisco Systems also provides a full-suite of solutions to support IP multicast in enterprises. In addition, standardization efforts are underway to extend multicast to

virtual private network (VPN) technologies [82] so that multi-location enterprises can effectively use IP multicast.

Current inter-domain multicast deployment is seen mainly in Internet2 [83], an initiative led by more than 200 US universities, working with industry and government to develop and deploy advanced network applications and technologies for research and higher education so as to accelerate the creation of tomorrow's Internet. Internet2 multicast deployment follows the strict guideline set out by the Internet2 Multicast Working Group which stipulates that all multicast deployed in Internet2 must be native IP multicast and sparse mode [17] and no tunnels are to be used as all routers must support inter-domain multicast routing using MBGP [84] and MSDP [85]. MBGP is a protocol to advertise routes to multicast-enabled networks across different domains while MSDP disseminates information about active sources across domains. Successful multicast deployment includes two Internet2 backbone deployment in USA, namely, vBNS [86] [87] and Abilene Network [88] as well as connections to other high speed networks linking member institutions. Examples of streaming applications using IP multicast include the Digital Video Transport Service (DVTS) [89] maintained by the WIDE (Widely Integrated Distributed Environment) Project [90], an Internet2 international partner in Japan and C-Span [91] by Northwestern University. These IP multicast streaming applications can only run across the Internet2 backbone.

Comparatively, application layer multicast is much easier to deploy since it neither requires global multicast-capable IP routers nor depend on an inter-domain multicast routing protocol. Its use of unicast in data delivery as well as the use of

proxies and end hosts enable better control and management of the group membership as well as rendering the multicast to be less vulnerable to malicious attacks. The popularity of application layer multicast has already infiltrated the commercial sector as Microsoft has teamed up with Nine Systems to provide advanced streaming delivery services worldwide using Nine System's Network of Networks [92][93]. This content delivery network is essentially an application layer multicast architecture and does not need to rely on IP multicast. Its importance is further underscore as one of the focus of Microsoft Research Asia's peer-to-peer related research is on application layer multimedia streaming and delivery [94].

Table 2.11 summarizes the relative performance of IP multicast and application layer multicast. IP multicast is certainly efficient in terms of high protocol efficiency and medium protocol overhead. On the other hand, application layer multicast has compromised protocol efficiency but triumphs in terms of its easy deployment with a rich repertoire of high level functionality.

Table 2.11: Performance comparison of IP multicast and application layer multicast.

Properties	IP Multicast	Application Layer Multicast
Protocol Efficiency	High with efficient bandwidth usage and low end-to-end latency.	Medium with less efficient bandwidth usage and higher end-to-end latency.
Protocol Overhead	Medium as state data increases with large number of groups.	High as probes are needed to measure network conditions in addition to state data to be maintained.
Failure Tolerance	High as infrastructure routers are used.	High if proxies are used and low if only end hosts are used.
Higher Level Functionality Support	Poor	Excellent
Ease of Deployment	Difficult to deploy.	Easy to deploy.

2.7 Summary

This chapter surveys the milestone application level multicast techniques documented in the various literatures and classify them into different categories based on their topology design, service models and architecture. Table 2.9 summarizes their properties to facilitate a quick insight into and understanding of their contributions. In addition, a performance comparison of the various techniques based on factors such as scaleability, efficiency of the protocols in terms of stress on the underlying network, relative delay penalty, protocol overheads and failure tolerance has been conducted. The results are presented in Table 2.8. In summary, the different techniques have been designed to cater to their different goals with their respective strengths and weaknesses as discussed in this chapter. Techniques based on embedded structures tend to enjoy superior scaleability with the drawback that they may not be as network topology aware as those based on tree architecture. Tree-based techniques, being more topology aware and hence are better able to exploit underlying network topology for efficient data distribution except that they are less robust. Tree-mesh architectures serve to strengthen the robustness of the data distribution trees but at a cost of higher overheads.

In comparison with IP multicast, application layer multicast technique presents the best solution for immediate deployment albeit with a compromise in protocol efficiency compared to IP multicast until such times when the issues facing IP multicast have been resolved and IP multicast is widely enabled.

Chapter 3

Proposal Overview: Hybrid Protocol for Application Layer Multicast (HPAM)

Chapter 3 presents an overview of the Hybrid Protocol for Application Layer Multicast (HPAM) proposed. The chapter elaborates on the design goals of HPAM and discusses the basic components that constitute the proposed application multicast protocol. The characteristics of HPAM in terms of its overlay topology design, service model and architecture are also presented together with the corresponding merits and shortcomings. A prelude to the technical novelties, which are to be incorporated into the HPAM, is also provided.

3.1 Network Environment and Service Model

The essence of HPAM lies in the migration of the network-layer packet forwarding service up to the application layer, vesting the multipoint delivery functionality

entirely to the end-clients that participate in the multicast session without any global multicast support from the network layer. HPAM's architecture is built upon the ubiquitous Internet protocols and service models which are shown in Figure 3.1. The following are the key network concepts which form the bedrock for HPAM's design:

Internet Protocols. HPAM leverages on the well-understood and robust Internet protocols in the network and transport layers for multipoint delivery service at the application layer. The reliable delivery of the Transmission Control Protocol (TCP) [95] as well as the best-effort delivery of the User Datagram Protocol (UDP) [96] are both harnessed by HPAM in addition to the Real-Time Protocol (RTP) [97] which is a transport level protocol for real-time media such as audio and video, for efficient application layer multicasting. RTP is augmented by a control protocol, the Real-Time Control Protocol (RTCP) [97], which allows applications to monitor data delivery and provide feedback reports from the receivers to the sources.

Locally Scoped Multicast. Despite the drawbacks of IP multicast as highlighted in Section 1.1, it remains inevitably the most efficient vehicle for multipoint delivery in terms of bandwidth conservation and scaleability. Multicast islands have been set up using administrative scoping [98] [99] whereby the multicast traffic is confined to a certain configured administrative scope zone (e.g. an organization's intranet). Such locally scoped multicast traffic does not cross the configured administrative boundaries and hence can be locally enabled without router support from the Internet Service Providers (ISPs). These intra-domain multicast capabilities are tapped by HPAM for efficient data delivery wherever available. One member host in the multicast island will become a HPAM client. Data which it receives from the HPAM

overlay tree will then be multicast to all the hosts in the same multicast island. HPAM therefore exploits the efficiency of local area multicast and extends the multicast service to the wide area by relying on an overlay of ubiquitous unicast connections to provide a seamless, fully deployable and efficient architecture for multipoint media streaming in the current Internet.

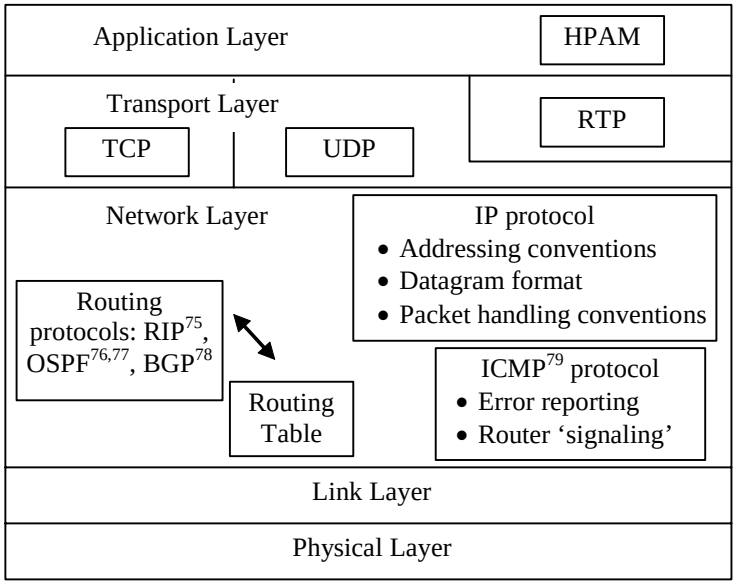


Figure 3.1: Internet Architecture and Protocols.

Topology of Network Substrate. The HPAM architecture is being abstracted from the underlying physical Internet topology as well as the IP routing infrastructure in order to be readily deployable. It does not require any modifications to the network topology, nor dictates the routing computation or packet forwarding mechanisms and policies adopted by the underlying infrastructure. Neither does HPAM require complete knowledge of the physical network structure. HPAM will instead discover the network topology incrementally by performing end-to-end measurements without disrupting the core network infrastructure. The only assumption made by HPAM is that the network provides a unicast communication service between any two IP end-

points and a locally scoped multicast distribution service as described previously. The lack of support from the network substrate will certainly impose a penalty on HPAM's ability to build optimal data distribution paths to fully exploit the network structure. However, the ability for HPAM to provide for efficient wide-area multicast delivery far outweighs the performance penalty on HPAM as discussed in Section 1.1.

End Host Overlay Network for Multicast. End host multicast, as the name implies, vests the multicast functionality at the end systems rather than within the network infrastructure to circumvent the pitfalls of IP multicast. End hosts taking part in a multipoint communication session essentially self-organize themselves, on the fly, into a dynamic overlay network of unicast connections. Multipoint data distribution to these end-clients is then effected through this unicast-based overlay network, providing seamless multicast functionality across the underlying network substrate without the need for ubiquitous multicast support at the network infrastructure level. This concept of end host multicast forms the very basis of HPAM's design.

3.2 Design Goals

HPAM's design is driven by three key goals:

Protocol Simplicity. The guidelines and philosophies set out in [105] for architects and designers of Internet backbone networks can best be distilled into the idea that simplicity itself can lead to some form of optimality. This idea is shared by Doyle in

[106] who stated that "*The evolution of protocols can lead to a robustness/complexity/fragility spiral where complexity added for robustness also adds new fragilities, which in turn leads to new and thus spiraling complexities*". Extending the simplicity principle up the protocol stack, HPAM protocol is hence designed to be simple yet efficient with reasonable robustness incorporated at little expense of increased complexity. Furthermore, the end-to-end design, described by Saltzer et. al. in [107] (as well as in RFC 1958 [108]), contends that end-to-end protocol design should not rely on the maintenance of state inside the network. Such state should be maintained only in the end points (e.g. hosts). This is augmented by Willinger [109] in his model of the Internet architecture which concludes that the complexity of the Internet belongs at the edges and the IP layer of the Internet should remain as simple as possible. In keeping with these principles, all of HPAM's functionalities are migrated to components entirely at the application level, leaving the core network layer technology untouched. This non-reliance on network layer and router support, further simplifies its design, implementation and deployment.

Overlay Efficiency. The data distribution tree constructed in HPAM must be efficient both from the network and the application perspective. On the network aspect, the constructed overlay must ensure that redundant transmission on physical links is kept minimal. For live media streaming application, techniques must be incorporated into the HPAM protocol to support minimization of single and dual metrics as well as to adapt to the dynamic environment. The former objective is to simultaneously ensure low latency delivery and low loss with low overheads incurred while the latter objective is to ensure that while the overlay adapts to the long-term variations in network dynamics, it should remain resilient to network noise and

inaccuracies inherent in the measurement of these quantities. Frequent changes to overlay topology can result in instability and transient performance degradation.

Self-Organising, Maintaining, Improving and Adapting. HPAM is designed to be self-managing and be responsive to changes in environmental conditions. It must be robust to dynamic changes in group membership and must incorporate mechanisms to gather network information and detect environmental changes. The protocol must allow for the overlay to self-tune and incrementally evolve for better performance as more information becomes available. New nodes are automatically integrated and unavailable nodes are disengaged whenever necessary to refine overall system performance. Furthermore, such integration and separation should occur with minimal disturbance to the rest of the system.

3.3 The HPAM Proposal

The fundamental building block of HPAM is the end host multicast overlay which sits on top of the network infrastructure. Upon this overlay of unicast connections, HPAM constructs source-specific data distribution trees on the fly. HPAM aims to provide best-effort multicast delivery service for real-time live data streaming with the goals of reducing latency and data loss. An example of the HPAM overlay is presented in Figure 3.2. It shows the HPAM multicast distribution tree rooted at the source with its branches drawn in thick solid lines and its nodes are the end-clients. Other components of HPAM include a lightweight centralized directory server (DS) which is the common rendezvous point for all members as well as the control message

communication channels. Control messages are shown in Figure 3.2 as thin dotted lines. Note that the control messages shown are not exhaustive so as not to clutter up the diagram. The alphabets shown beside the control messages indicate the sequence of steps involved when a new member joins a HPAM session. Local multicast network is shown in Figure 3.2 in thick dotted lines.

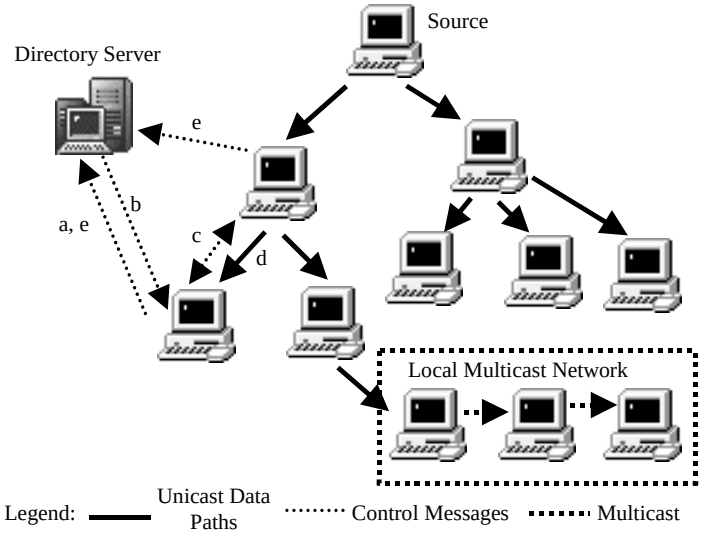


Figure 3.2: An Example of HPAM Overlay.

3.3.1 Basic Operation

A new client who wishes to participate in a HPAM session first contacts a global, well-known server to discover the available sessions. This can typically be a web server with a well-known URL. Sources advertise their sessions on this website. Both native IP multicast sessions and unicast sessions are supported. The former is made available to the server via session directory functionalities (e.g. SDP, SAP, address

allocation) [110] while the latter requires the unicast source to register itself with the server.

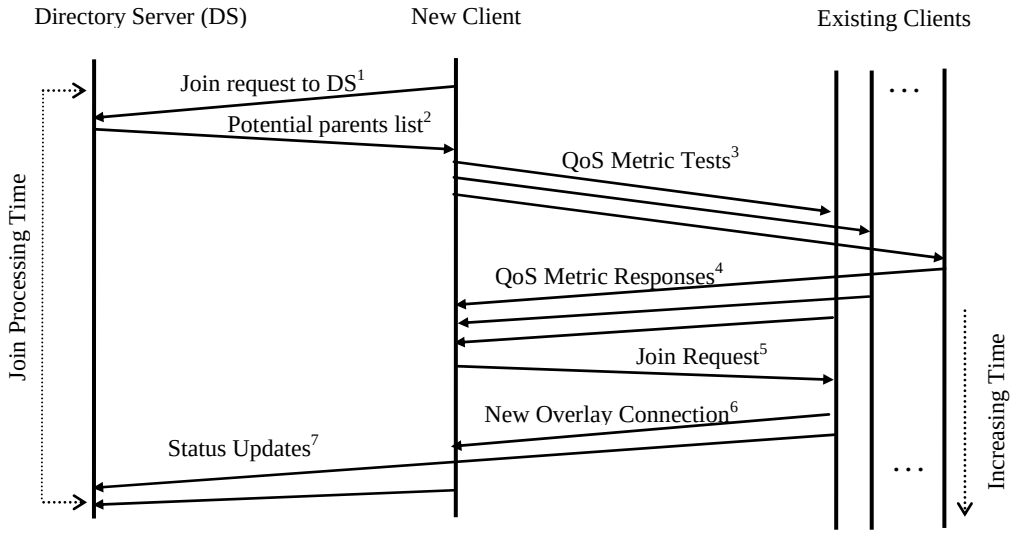


Figure 3.3: Communication Diagram of Join Process.

3.3.1.1 Join Process

Figure 3.3 shows the communication diagram of a join session. A new client selects a session it wishes to join and issues a join request to DS (Step a in Figure 3.2, Step 1 in Figure 3.3). DS searches its database and returns a list of potential parents (Step b, Step 2) based on a set of selection criteria. The client then contacts all the potential parents (Step c, Steps 3, 4) and conducts end-to-end QoS measurements to determine the QoS of the individual parent candidate. Optimally, the client will choose the parent who can provide the best QoS and issues a join request to this parent (Step 5). If this parent candidate has available out-degree to support the new client, the new client will be admitted as its child and it will start to distribute data to the new client via the new overlay connection (Step 6) and update the DS of their new status (Step e,

Step 7). In the event that the best parent is unable to accept the child, the new client will try the next best and so on. When no potential parent is found, the new client will issue a new join request to DS and the process is repeated. The tree construction algorithm, the parent selection criteria and the QoS metrics shall be covered in the following chapter on HPAM protocol. Unicast data packets are streamed to the various endhosts via unicast while IP multicast packets are distributed via UDP tunnels [111] [112].

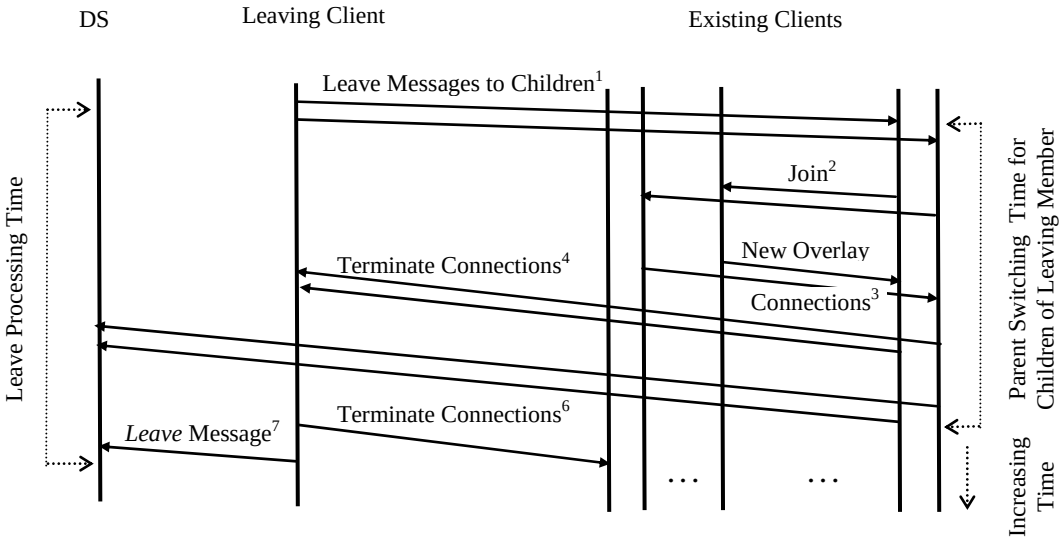


Figure 3.4: Communication Diagram of *Leave* Process.

3.3.1.2 Graceful Leave Process

Clients can join and leave the HPAM group at will. A client can leave the session gracefully by informing all parties of its intention to leave or it can leave abruptly without giving due notice to the other interacting parties. The former shall hereinafter be referred to as a *Leave* process while the latter is termed as *Sudden Failure.* Figure 3.4 shows the communication diagram of a leave session. To depart from a session, a

client first issues a *Leave* message to its children (refer to Step 1 of Figure 3.4). The affected children will perform a switch to new parents as indicated by the join requests to other existing clients (Step 2) in Figure 3.4. Upon successful connections to the respective new parents (Step 3), the children will terminate their connections to the leaving member (Step 4) and update DS of their new status (Step 5). Once all the children have switched to new parents or when the *Leave* timeout occurs, the leaving client will terminate the connection to its own parent (Step 6) and send the DS a *Leave* message (Step 7). This completes the *Leave* process. In the case where the leaving client is a leaf member of the HPAM overlay tree, it will skip Steps 1 through 5 since it has no children and proceed directly to terminate its connection with its own parent and inform DS of its departure.

3.3.1.3 Member Failure Management

So long as a client is part of the session, it sends periodic *Alive* messages to its parent (refer to Step c of Figure 3.2) and DS (refer to Step e of Figure 3.2). A prolonged absence of the *Alive* messages will be deemed as a sudden death of the client. In such event, the DS will delete the client's entry from its database while the affected children (if any) will switch to new parent(s).

At this juncture, it is to be noted that the ready list of new parents for the orphaned children to switch to is provided via the gossip and spiral mechanisms presented in the following section. As noticed from the communication diagram in Figure 3.4, there is no necessity for the children to approach DS to seek out new parents. This not only offloads DS considerably but more importantly, it facilitates and speeds up the

switching process. The latter ensures that there is minimum data disruption to the affected client.

3.3.2 Topology Design

HPAM is designed to be efficient, self-organising and self-improving. In keeping with the principle of a simple design, HPAM's data topology consists of self-configuring source specific trees spanning all members of the multicast session. The control topology comprises these data delivery trees, a lightweight centralized directory server (DS) which is the common rendezvous point for all members, control messages among a subset of members as well as between members and DS. DS does not take part in the actual data delivery and is only responsible for the group management functions. It should be noted that in the eventuality that DS fails, data distribution still functions normally across the data topology. The only setback is that new clients are unable to join the tree until DS recovers. Moreover, as DS does not take part in the actual streaming process, its load is vastly reduced compared to a centralised data server.

HPAM has adopted a tree-only approach as opposed to the two-step tree-mesh approach employed by some application level multicast systems for its simplicity of construction and lower overheads. A tree-mesh approach such as that adopted by Narada [27][28] has the advantage of being more resilient to failure of the clients compared to just the tree. However, it incurs much more complexity in its construction as it has to ensure connectivity in the mesh resulting in high overheads.

Two versions of HPAM have been proposed, namely, HPAM which is designed to minimize a single cost metric, i.e. latency in the data distribution tree [42][47], and HPAM-D [113] which is among the first to incorporate loss rate with latency as dual performance metrics to improve the data tree. For this research, latency is prioritized over loss rate. This fact notwithstanding, the HPAM protocol can be easily modified to prioritize loss rate over latency for applications which value perceived quality of the received data over the latency from the source.

In HPAM, the latency metric used in constructing the data distribution tree is being quantified as the Round Trip Time (RTT) or round trip application level delay between group members in the tree. RTT values between any two members are obtained via asynchronous end-to-end measurements. A new member will pick the node that renders it closest to the source in terms of network distance (i.e. RTT), to be the parent in the tree building process.

For HPAM-D, the potential parent is the one which will render the joining member to be closest to the source in terms of RTT and has a very low if not nil loss rate. Unlike the improved version of Narada which uses RTT and bandwidth as performance metrics, HPAM-D uses loss rate instead of bandwidth as the former can be implicitly estimated while data is being received by each member while the latter requires intrusive active end-to-end measurements. Such measurement involves transferring data for a period of time using the underlying transport protocol at a rate bounded by the source rate between members, thereby incurring high overhead and latency.

As a tree-only approach is less robust than the tree-mesh approach to tree partitioning problem and possible looping problem, gossip-style algorithm (used initially in the Clearinghouse project [62] to maintain database consistency and has since been used to achieve fault tolerance and detection [63][64]) and spiral mechanism have been incorporated to strengthen the data tree as well as to facilitate partition recovery when a member leaves the tree. Gossip-style algorithm uses a randomized mechanism for deciding when and with whom each member will gossip. Each member will randomly exchange soft states such as out-degree availability, loss rates and RTT with a dynamic cache of gossip candidates which are themselves also members of the multicast session. In addition, connections are also established with a member's ancestors or siblings forming a network of spiraling connections. The spiral mechanism basically provides a more robust overlay tree by its ability to withstand node failures in any of its overlay tree branches so long as these failures are not consecutive nodes of the same branch. Hence, the idea behind incorporating the gossip channels and the spiral network is to provide a form of communication redundancy among the members in the data distribution tree. This redundancy is a simple way to provide a quick recovery from failures in the data tree by providing a ready list of alternative parents to clients whose own parents have failed so that data distribution is minimally disrupted and the tree partition is repaired speedily. In addition, a client can also latch onto these gossip candidates to switch to a parent which can provide it with a better quality of service.

An example of the gossip and spiral mechanisms is shown in Figure 3.5. Member C10 is shown to be gossiping with C5, C6, C7 and C8. Member C9 spirals with C1

(C9's grandparent) which can withstand the failure of node C4 while C4 spirals with C2 instead of its grandparent which happens to be the source itself. C1 will spiral with its sibling C3 (the spiral is not shown in Figure 3.5) as it has no ancestor to spiral to. In the worst case scenario where no suitable alternative parent can be found, the node can turn to DS to request for a new list as a last recourse. The gossip and spiral mechanisms, besides providing a fast partition recovery, also serve to alleviate the load on DS when recovering from a tree partition.

State data maintained at each node is minimal as unlike HM [39] (another tree-only technique), information of all members on its path to the source is not maintained and neither does it need to maintain state for the entire membership (such as Narada). Hence HPAM incurs much less overhead where data maintenance and refreshing are concerned, keeping only a small subset of information pertaining to its parent, children and gossip cache. HPAM makes use of a level number for loop detection. Members update their level number each time they change a new parent to reflect their hierarchy from the source.

In summary, HPAM's topology is a flat single-layer overlay, sitting on top of the physical network, which leverages on the simplicity and efficiency of the tree structure. HPAM adopts a topology-aware approach to application layer multicast, building data delivery trees which are as congruent to the substrate network as is technically permissible at the application layer, relying on lightweight end-to-end measurements to learn about the underlying network topology.

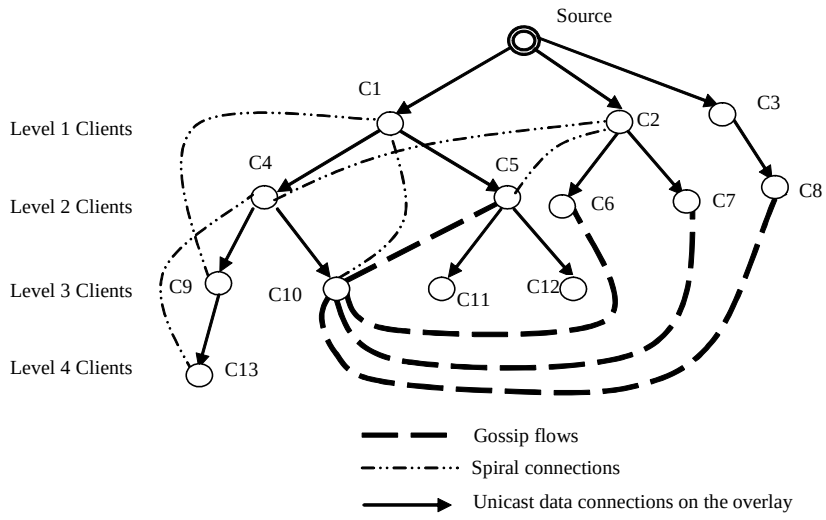


Figure 3.5: An example of a HPAM delivery tree showing the gossip and spiral connections among the members

3.3.3 Service Model

As HPAM is focused on real-time live media streaming applications, its service model is based on best-effort delivery, choosing to optimize delivery time rather than achieving total data integrity since video data are tolerant of some loss. A source-specific delivery model (one-to-many) is adopted by HPAM since media streaming operation typically involves a single-source and many receivers and these receivers are unlikely to subscribe to more than a single source simultaneously. This is verified by Beverly and Claffy in their work on wide area IP multicast traffic characterization where they observed that only a mere 1.1 percent of the groups monitored ever had multiple simultaneous sources [114]. Moreover, source-specific delivery model is simpler to conceptualize, implement and optimize. This is especially so in the absence of intimate knowledge of the underlying network topology where it is relatively simpler to optimize the data delivery tree with respect to a single source rather than

optimizing the shared tree for all sources. In fact, one of the factors impeding widespread adoption of multicast is the complexity incurred in supporting multiple sources. Beverly and Claffy actually recommended in [114] that source-specific multicast [26] is to be adopted as the de facto standard for multiple group multicast as a means to address the above problem.

3.3.4 Architectural Model

As illustrated in Figure 3.2, HPAM assumes a peer-to-peer architecture instead of an infrastructure (proxy) based architecture. The multicast and group management functionalities are vested with the end-users rather than proxies deployed explicitly at strategic locations across the Internet. The peer-to-peer architecture is adopted primarily due to the advantages derived from its distributed nature. Any member participating in an HPAM session can act as a receiver as well as a source for other receivers without the set-up and installation of dedicated proxies. This peering ability means that it can easily scale to larger number, be readily deployed across wide areas as well as bypass hot spots on the fly without being constrained by the number and locations of the proxies. Moreover, the end users need only to maintain state for those groups it is actually participating in rather than for all the active sessions as is the case of using proxies.

In terms of the type of algorithms adopted for overlay creation and maintenance, HPAM has adopted a hybrid approach which combines the simplicity and optimality of a centralized controller with the robustness and scaleability of a distributed receiver

based technique. In HPAM, a lightweight controller is used to facilitate some group management functions such as peer discovery, join process as well as to serve as a reliable back-up to support error recovery in the eventuality that the distributed algorithm fails. The actual tree construction, routing and recovery from partitions are left to the individual receivers. A totally distributed algorithm is not considered in HPAM as it causes excessive overheads and is not as optimal and efficient in building the data distribution tree.

It is important to note that the lightweight centralized controller (DS) does not perform routing computation and topology management unlike those controllers in centralized protocols such as ALMI, HBM. DS does not centrally compute the data distribution tree nor does it actively manage the tree membership. The only role that DS plays in routing and topology management is to catalyze the process by providing a new client with a list of possible parent candidates. This is made possible as the DS maintains minimal state information of all the members. Its role is actually very much similar to the rendezvous point (RP) or root adopted in many distributed protocols such as HM, TBCP, Narada etc. for the new client to discover an initial list of possible parents to trigger the subsequent distributed tree building process. Routing responsibility, tree construction and recovery from partition still vest solely in the individual client.

The benefit of simplicity offered by the resourceful, centralized lightweight server far outweighs any negative implication from the fault tolerance perspective and scaleability. The simplification of the tree construction protocol through the catalytic source discovery process makes for a speedy join without the need to complicate the

protocol by assigning a temporary foster parent until such time that the new client has finally located a suitable parent as exemplified in the distributed HM protocol. In addition, the DS serves as a reliable and fast back-up to tree partition recovery in the event that the distributed algorithms based on spiral and gossip mechanisms fail. It is to be noted that while the DS is centralized, it can still hold up well in situations of flash crowds as the burden of distributing media content is shared by all the peers. DS itself is not involved in the media streaming and in the unlikely event that DS fails, the data distribution tree continues to function. The only glitch is that new clients will be prevented from joining the groups.

3.4 Strengths and Weaknesses of HPAM

The strengths and weaknesses of HPAM are certainly a direct result of the design choices made. The rationales and the merits of its architecture have been discussed in the preceding sections and thus will only be briefly summarized in the following section while the weaknesses will be deliberated subsequently. Where applicable, comparison will be made with respect to IP multicast.

3.4.1 Merits

Efficiency in Simplicity. From the design perspective, the essence of HPAM's merits lies in the simplicity of its concept, design, protocol and implementation. Without delving into the specifics of HPAM protocol, a higher order definition of efficiency in the context of HPAM's architecture shall refer to the relative ease of refining the data

distribution tree and tracking of network metrics using lightweight end-to-end measurements without incurring excessive overheads. Hence, the choice of a flat, topology-aware, tree-only overlay topology contributes positively towards these ends compared to a two-stage tree-mesh approach or an embedded approach. The decision to settle for a source-specific service model for HPAM's data distribution tree further complements the design goal of deriving efficiency from plain simplicity. Optimization is also enhanced and overheads reduced by the joint operation of a centralized lightweight controller in conjunction with a distributed tree building algorithm at the end users. Moreover, a best-effort delivery model is adopted as it can adequately fulfill the application needs of real-time live media streaming without the complexity of designing more algorithms to ensure end-to-end reliable multicast.

Deployability. Adopting a peer-to-peer architecture speeds up the deployability of HPAM as there is no need for dedicated infrastructure support and neither is there a need to deploy HPAM universally across the entire Internet right at the outset. HPAM can also be deployed incrementally in the sense that as more nodes are added to an HPAM system, the system's resources, which can approximately be perceived as the aggregate resources of all its peers, are increased accordingly. These nodes can in turn support more nodes and extend the benefits of HPAM across the wide expanse of the Internet. This is unlike the proxy-based approach whereby new resources, which in this case refers to the proxy servers, have to be added to service the increasing number of member nodes.

Reducing Network Complexity. By migrating the media distribution service to the application layer, HPAM keeps the underlying network model intact and

straightforward by not having to perform modifications to the network substrate. Instead, HPAM leverages on the pervasive and robust Internet architecture and protocols and can therefore be readily deployed in the current Internet. By exploiting local multicast and adopting unicast protocols for wide area communication, HPAM eliminates the need for IP multicast across the Internet. The network routers need not worry about huge forwarding state problem resulting from managing the different multicast sessions and neither does the network need to worry about the absence of a universally accepted inter-domain multicast routing protocol. HPAM has migrated the state and protocol complexity of wide area multicast out of the network layer to the end users at the application layer where complexity can be more flexibly and efficiently handled.

3.4.2 Weaknesses

Notwithstanding the fact that HPAM does provide a number of advantages for efficient multipoint streaming of media data via application level multicast, HPAM too has its weaknesses.

Single Point of Failure. The use of a lightweight directory server in HPAM has implications on the reliability and fault tolerance of the system. As DS is the rendezvous point for new members to join the group, it constitutes a single point of failure for all operations pertaining to new members joining HPAM sessions. In such eventuality, new members cannot join the sessions although the functionality of the existing data streaming sessions will remain unaffected since DS does not participate

in the actual data distribution. The incorporation of mechanisms, namely, dynamic gossip and spiral mechanisms, has grossly reduced the reliance on DS to recover from tree partition thereby further minimizing the impact (if any) on the HPAM sessions in the event of a DS failure. From the fault tolereance perspective, the underlying design rationale is that the benefits of simplicity offered by the lightweight DS far outweigh the small negative implications of a transient DS failure.

One way to improve the fault tolerant capability of DS is to augment it with backup servers operating in standby mode, whose addresses are well-known. These standby servers are periodically refreshed with the states from the operating DS and can assume the function of DS once it fails.

Robustness. The robustness of HPAM is very much dictated by the design choices of its architecture. Besides the centralized nature of DS, the tree-only approach to overlay construction and the use of peering instead of proxy-based technique are perceived as potential weaknesses in HPAM's system reliability and fault tolerance. A lone data distribution tree is more susceptible to partition compared to a counterpart who is reinforced by a fully-connected mesh or other embedded structures discussed in Chapter 2. HPAM has in its design, sought a subtle balance of incorporating additional connectivity among its members through spiral and gossip mechanisms but not the full connectivity as in the case of a tree-mesh approach. With this moderated approach, HPAM saves on the complexity and excessive overheads that are associated with the construction of the mesh and other fault-tolerant structures. This huge savings is at the slight expense of a transient disruption in its service in the worst case.

The peering of members in HPAM is also considered less robust compared to infrastructure-based system using dedicated proxies set up and deployed by the service administrators. The proxies are expected to be better provisioned than individual hosts and are persistent beyond the lifetime of the end users. In this case, HPAM has settled for a quick recovery effected by the gossip and spiral mechanisms in the event of a peer failure in exchange for better deployability and scaleability.

Scaleability. HPAM's multicast overlay is a logical network of end users sitting atop the physical network infrastructure. Unlike multicast routers which typically execute the routing and forwarding algorithms in fast specialized hardware, HPAM's overlay nodes are peering software entities that can give rise to scaleability concerns. The ability of these end users to service the load offered by the various HPAM data streams may be a problem. Moreover, there are also the issues of multiple copies of the same HPAM data traversing the same physical network link compared to IP multicast where data always traverses a network link only once as well as the need for control data exchanges and end-to-end measurements which are not required in IP multicast. These essential overheads of application layer multicast further load the network infrastructure and impact the scaleability of the entire system. To arrest the problem of overloading the end users and their network resources, HPAM has incorporated into its design a constraint on the out-degree of each end user. In other words, each end host can only peer a certain number of nodes to ensure that the QoS of the data delivery is maintained. This will be discussed in the following chapter on HPAM protocol.

Another scaleability concern of HPAM is the centralized DS which manages member join and holds the performance data of the members in the HPAM sessions. These data has to be updated periodically by the HPAM members and DS is therefore a potential bottleneck. This issue can be addressed by replacing the DS with a workstation cluster to distribute its load and improves its response times under heavy load. As DS does not participate in the data streaming process, bandwidth should not pose a constraint to its scaleability.

3.5 Technical Novelties of HPAM

As per the goals set out in this chapter, HPAM is designed to be simple, scaleable and yet topology-aware of changes in network and group dynamics with the ultimate aim of providing best-effort multicast delivery service for real-time video stream at the application layer level without IP multicast support. Pivotal to HPAM's efficient performance is a host of innovations which are fundamental to HPAM and which distinguish it from other application level multicast techniques. This section shall highlight these technical novelties to serve as a prelude to the details of which will be covered in the subsequent chapters.

3.5.1 Hybrid Nature

Unique to HPAM is its hybrid approach to overlay construction, refinement and maintenance. It exploits the simplicity and optimality of a centralized controller (DS) with the robustness and scaleability of distributed receivers based technique. HPAM's

data topology is peer-to-peer in nature and comprises source-specific, loop-free overlay trees whose nodes are the individual clients of the multicast session with no proxies used. The individual clients are responsible for the actual tree construction, routing, refinement and recovery from partitions. The overlay trees are self-improving as they adapt readily to both group dynamics and network environments. Environmental data are gleaned from non-invasive RTT measurements and information exchange with a select group of clients. Information such as loss rate can be extracted from the received data stream itself without much additional protocol overhead. The overlay trees are also capable of self-recovery from tree partitions in the event of client failures through the novel use of the *Gossip* and *Spiral* mechanisms.

Its control topology comprises the overlay trees, DS which is the common rendezvous point for all members, refresh exchanges among a subset of members as well as updates to the DS. DS is used to facilitate some group management functions such as peer discovery, join process and to serve as a reliable back-up to support error recovery in the eventuality that the distributed self-organizing and self-improving algorithm fails. *DS* does not take part in the actual data delivery.

3.5.2 Gossip and Spiral Mechanisms

As mentioned in Section 3.4.2, one of the weaknesses faced by the HPAM overlay is the vulnerability of the overlay tree to partition as a result of client failure or voluntary departure. A partition occurs when a client failure or departure cuts off the data reception to its downstream clients, thereby isolating all clients belonging to this

branch from the HPAM data delivery tree. Two mechanisms, namely the *gossip* and the *spiral* mechanisms, are introduced to facilitate speedy partition repairs to ensure the robustness of the overlay.

The *spiral* mechanism is the first line of recovery from partition for an orphaned client. In the s*piral* mechanism, every client is assigned a spiral node as illustrated in Figure 3.5. A spiral node is a hot standby node and is initially the grandparent or an uncle (the siblings of its parent) of the client in the overlay tree. As a client discovers more and better peers during the tree refinement process, it can always replace its initial spiral node with a better one. An orphaned client is to graft itself to the spiral node in the event of a partition.

The second line of recovery is provided by the g*ossip* mechanism. Every client establishes a small random gossip network with a ready list of potential parent candidates. This gossip cache is first established when a new client joins the session and is given a list of potential parents by the DS. The gossip cache is replenished as clients discover new peers during their gossip process as they exchange both performance information as well as information of gossipers in their own gossip cache. Clients can also request DS for a fresh supply of gossipers. In the event that the spiral node is either too full to adopt the orphan or has itself failed, the orphan will then graft itself to the gossiper in its gossip cache who can yield the best QoS (RTT, loss rate) for the orphan.

3.5.3 Self-Refining Overlay Trees

The overlay trees in HPAM are self-improving in that the individual clients will carry out periodic refinement to improve the tree performance without any centralized intervention. The refining algorithm used in the single-metric HPAM is proactive in nature whereas the dual-metric HPAM-D makes use of both the proactive as well as the reactive approaches. Tree quality improvement is basically effected via a client switching to a parent which will provide it with a better QoS. The crux lies in discovering a 'better' parent and this is made possible through the gossip mechanism. The QoS that can be delivered by each gossiper in the gossip cache is dynamic as the QoS of the gossipers changes over time based on the prevailing network conditions. The changes are updated during the information exchange among the gossipers during the gossip process.

In HPAM, a client will periodically evaluate the improvement in the latency which it can derive if it were to switch its parent to each of the gossip cache candidates. If the improvement is substantial, it will perform a parent switch to the selected gossiper. This mechanism for improving the latency of the HPAM data tree is termed *Proactive Tree Refinement* (*PTR*) as the client takes the initiative to check for possible improvement to be made to the data tree.

In HPAM-D, tree quality improvement seeks to improve the latency as well as to reduce the loss rate. It is effected via both the enhanced *PTR* as well as via the *Reactive Tree Refinement (RTR)*. The enhanced *PTR* is periodic and self-initiated as per the HPAM's *PTR* except that the 'better' parent must also possess an acceptable

loss rate over and above the improved latency value. The *RTR* is named as such since the tree refinement is triggered in response to an unsatisfactory loss rate experienced by the client. A client will seek to reduce its loss rate through parent switching if it experiences a loss rate that is higher than a set threshold (L_{th}) for a period of time.

3.5.4 JoinSource&Adopt (JSA) Algorithm

A common problem faced by all tree construction protocols building data tree on the fly is that the resulting tree is entirely dependent on the order of join of the clients. It is perfectly plausible that the level 1 (the level closest to the root) slots are occupied by clients who are 'far away' from the root in terms of RTT but by virtue of their early join, they have 'locked' up the level 1 slots. The 'nearer' clients who join the tree later are thus relegated to levels which are further away from the root of the tree thereby increasing the latency of all clients downstream of that tree branch given the high latency of their level 1 ancestors.

Centralized tree construction algorithms which regularly re-compute the optimal tree inherently eradicate this problem (e.g. HBM [43]) since the entire tree after the root, is rebuilt. In the case of distributed algorithms, they face the same problem as HPAM since their refinement algorithms very often do not rebuild the entire tree. Moreover, distributed algorithms have knowledge of only a limited number of group members which is therefore inadequate for them to overcome this problem.

Herein lies the innovation in HPAM's *Join Source & Adopt (JSA)* algorithm. The *JSA* algorithm does not aim to eradicate this problem of latency-inefficient clients

hogging levels close to the root of the tree as such a solution will incur too much control overhead and disrupt the data distribution process. Instead, it focuses on just optimizing the level 1 clients. Once, this level is latency-efficient, the impact on the subsequent levels will be minimized. The *JSA* algorithm basically allows a new client, *m*, to replace an existing client to join the root of the tree. The existing client is then adopted by the new client as its child. The selection of which existing clients to replace is dependent on their respective latencies to root as compared to *m*'s latency to root.

3.5.5 Loss Rate as the Second Performance Metric

Among the application level multicast techniques published, HPAM is one of the first to incorporate loss rate as a second performance metric in addition to latency to build and optimize the overlay data tree in HPAM-D, the dual-metric version of HPAM. The incorporation of the second metric not only allows HPAM-D to minimize the latency to the root but also enables it to maintain a high percentage of clients with acceptable loss rate in a dynamic network environment where there are congestion and losses. All these are achieved with negligible protocol overhead over the single-metric latency-based counterpart as the loss rate experienced by a client can be passively measured from the data packets received.

3.5.6 RLR based Heuristics for Local Congestion Detection

Another novel feature in HPAM-D is the relative loss rate (*RLR)* heuristics incorporated for the detection of possible local congestion that exists between a client

and its parent which may therefore contribute to the data loss in the client. The *RLR* between a client and its parent is derived from the loss rate data maintained by all the clients. The *RLR*-heuristics works by only allowing a client to switch parent if the *RLR* is assessed to be poor and will otherwise delay its parent switch and wait for its parent to do the switching first. This ability to detect possible local congestion will enhance the performance of HPAM-D in terms of reducing latency and loss rate to the affected client as well as reducing the protocol overheads. The former is accomplished by clients switching to alternative parents to ease the congestion and reduce loss rate while the latter is facilitated through the reduction in the number of unnecessary parent switches.

3.5.7 Cheat Detection

The incorporation of detection algorithms against cheating HPAM clients is another distinctive feature of HPAM. Although HPAM is designed in the spirit of honest sharing and collaboration, there are incentives for clients to cheat to move close to the root and to do away with the burden of carrying children. HPAM uses a combination of *DS* and community policing by the clients to detect subtle cheats who fabricate their RTT measurements. The cheat detection algorithms require additional RTT tests to verify the reported readings by cheating clients as well as to invoke additional parent switching to isolate the cheats to the leaf nodes of the overlay tree. As a result, there are high protocol overheads and hence, these features are designed such that they can be turned off if their services are not deemed necessary.

3.6 Summary

In this chapter, an overview of HPAM's architecture, the design goals, its operation as well as the network environment underlying the architecture is presented. The pros and cons of HPAM which arisen as a result of its design are also deliberated. An insight into the novelties which will distinguish HPAM from other application level multicast techniques is also provided. Using the same categorization for the other application level multicast techniques surveyed in Chapter 2, HPAM's properties are discussed and they are summarized in Table 3.1 for ease of reference.

Table 3.1: Properties of HPAM.

Properties	HPAM
Goals	Minimize latency and data loss
Topology-Awareness	Topology-Aware
Topology Design	Tree
Topology Hierarchy	Flat
Delivery Type	Best Effort
Source Model	Source Specific
Proxy	Peer-to-Peer based
Type of Algorithm	Distributed with Lightweight Central Directory Server

Chapter 4

HPAM Protocol

Chapter 4 presents a description of the protocol design in HPAM to effect application layer multicasting over wide area heterogeneous network via a collection of end users. Details of the heuristic algorithms used to dynamically construct and maintain the robust and efficient data distribution trees are elaborated with theoretical analysis presented wherever possible.

4.1 Problem Formulation

HPAM's goal is to construct efficient source-rooted data distribution trees spanning the end hosts. The trees are constructed to optimize a cost function, be it the latency from the source in the case of HPAM or a cost combination of latency and loss rate for the case of HPAM-D. Since HPAM does not assume a lower level multicast service support, the data distribution trees are constructed at the application level based on data gathered from active network probes. These network probes provide HPAM with the necessary topology-awareness to build overlay distribution trees

which are as congruent to the physical network topology as possible to optimize latency.

IP multicast is most efficient in terms of latency and bandwidth since its spanning tree maps exactly with the physical network and a data packet only traverses a physical link once as shown in Figure 1.1. Naïve unicast, on the other hand, is a star-topology where N copies of the data are duplicated over N paths to N receivers, resulting in high bandwidth usage and losses especially near the source albeit at a lower latency compared to application level multicast. Furthermore, naïve unicast is non-scaleable. HPAM's application level multicast spanning tree is certainly less efficient than IP multicast tree since its data packets can traverse a link more than once resulting in undue network traffic. Nevertheless, it is much more efficient than naïve unicast as HPAM's protocols serve to limit the amount of duplicate data packets by restricting the out-degree of its node based on its bandwidth capabilities. However, a degree-constrained tree is deeper thereby resulting in longer latency for certain nodes than the corresponding delays in the IP multicast tree or naïve unicast star.

The objective of the HPAM protocol is formulated as follows:

To construct a directed degree-constrained spanning tree, rooted at source, across end hosts to:

i. minimize the overlay latency for all clients from the source
ii. minimize the loss rates of all clients (in the case of HPAM-D)

The problem formulation is as follows:

The underlying physical network is modeled by a complete directed graph denoted by $G = (V,E)$ with p vertices, $V = \{v_i\}$ where $i = 1 .. p$ and a set of directed links, $E = \{e_k\}$, where $k = 1 .. q$. A physical network path is defined as an alternating sequence of nodes and links in the physical network, i.e. $P(v_i,v_k) = \{v_i, e_i, v_{i+1}, e_{i+1}, \ldots, v_{k-1}, e_{k-1}, v_k\}$ such that $v_i \in V$, $e_i = \langle v_i,v_{i+1} \rangle \in E$, for $0 \leq i \leq k-1$. Each link, e_i, has a cost metric $\chi(e_i)$ which can represent hop count, delay, bandwidth capacity, loss rate or dollar cost of edge e_i. The physical network path distance or physical network path cost from vertices v_i to v_j based on the prevailing routing algorithm used is thus defined as

$$\chi\left[P\left(v_i, v_j\right)\right] = \sum_{e_k \in P\left(v_i, v_j\right)} \chi\left(e_k\right) \qquad \text{Eqn. 4.1}$$

Consider an overlay network whose nodes are end hosts participating in a HPAM session and whose set of connections are logical connections representing the unicast distances between two end hosts along the overlay path. Let N represent all the end hosts participating in a HPAM session with a source $n_s \in N$. These end hosts are connected to the physical network G at different vertices through access links. Hence, there exists a corresponding vertex v_i in the underlying physical network G for every n_i in the application layer. Each end host has an out-degree bound $k_i \geq 1$, $\forall n_i \in N$ on the HPAM overlay tree and L is the set of logical connections connecting these end hosts. The overlay data delivery path for HPAM is thus a directed spanning tree $T(n_s)$, rooted at source n_s, with the edges directed away from the root and spanning all end hosts, n_i such that δ_i, the degree of end host $n_i \in N$ in $T(n_s)$ is at most k_i.

Analogous to the definition of the physical network G, an overlay path on the HPAM data tree is defined as an alternating sequence of end hosts and logical

connections in the overlay network, i.e. $P_{HPAM}(n_i,n_k) = \{n_i, l_i, n_{i+1}, l_{i+1}, \ldots, n_{k-1}, l_{k-1}, n_k\}$ such that $n_i \in N$, $l_i = <n_i,n_{i+1}> \in L$, for $0 \le i \le k-1$ where the unicast latency for each overlay connection, l_i, is defined as $\lambda(l_i)$. The latency from end host n_i to end host n_k, defined as the summation of all the unicast distances or latencies along the overlay path from nodes n_i to n_k assuming shortest path symmetric Internet routing in the physical topology G, is given as

$$\lambda\left[P_{HPAM}(n_i,n_k)\right]_{i,k} = \sum_{l_j \in P_{HPAM}(n_i,n_k)} \lambda(l_j) \qquad \text{Eqn. 4.2}$$

Mapping the overlay paths onto the physical links of network G and defining the physical link cost metric $\chi(e_i)$ as the link response-time representing the total time including processing and transmitting by the edge, e_i, Eqn. 4.2 for latency between two end hosts can be expressed as:

$$\lambda\left[P_{HPAM}(n_i,n_k)\right] = \chi\left[P(v_i,v_k)\right] = \sum_{e_j \in P(v_i,v_k)} \chi(e_j) \qquad \text{Eqn. 4.3}$$

The tree cost $C_{T_{HPAM}}$ of $T(n_s)$ measured in terms of latency in HPAM is defined as

$$C_{T_{HPAM}} = \sum_{j=1, n_s \neq n_j}^{j=k_s} \lambda\left[P_{HPAM}(n_s,n_j)\right] + \sum_{\forall n_i \in N, n_i \neq n_j \neq n_s} \lambda\left[P_{HPAM}(n_i,n_j)\right] \qquad \text{Eqn. 4.4}$$

Substituting Eqn. 4.3 into Eqn. 4.4, $C_{T_{HPAM}}$ becomes

$$C_{T_{HPAM}} = \sum_{j=1, v_s \neq v_j}^{j=k_s} \lambda\left[P(v_s,v_j)\right] + \sum_{\forall v_i \in N, v_i \neq v_j \neq v_s} \lambda\left[P(v_i,v_j)\right] \qquad \text{Eqn. 4.5}$$

$$= \sum_{j=1, v_s \neq v_j}^{j=k_s} \sum_{e_i \in P(v_s,v_j)} \chi(e_i) + \sum_{\forall v_i \in N, v_i \neq v_j \neq v_s} \sum_{e_i \in P(v_i,v_j)} \chi(e_i)$$

Ideally, the objective of HPAM is therefore to find the distribution tree $T(n_s)$ whose total cost $C_{T_{HPAM}}$ is the minimum among all possible such trees. However, the minimum latency degree-bounded directed spanning tree problem in Eqn. 4.5 is known to be NP-complete [115][116]. Hence it is difficult to obtain the optimal solutions for the computationally hard problem defined above within a realistic period. Moreover, the problem is compounded by the lack of accessibility to the cost data of the physical link as HPAM does not assume that the routers will provide such information. The problem is exacerbated by the dynamicity of HPAM's group membership as it undermines the efficiency of the optimal HPAM spanning tree unless recomputation is done for every change in group dynamics. The computation of such ideal tree is prohibitively expensive and hence impractical. As such, a heuristic approach is proposed to provide reasonably good solutions in practical time to solve the HPAM spanning tree problem. In order to evaluate if the heuristic approach can construct a near optimal tree, it is imperative to use some form of benchmark.

A more practical benchmark to be used is the cost of the IP multicast tree, $T_{mcast}(n_s)$ which is rooted at n_s and spans all of HPAM's N end hosts. $C_{T_{mcast}}$ is computed as the sum of the cost of all the links in the IP multicast tree $T_{mcast}(n_s)$ and is given as follows:

$$C_{T_{mcast}} = \sum_{e_j \in T_{mcast}(v_s)} \chi(e_j) \qquad \text{Eqn. 4.6}$$

A point to note is that the theoretically optimum multicast tree involves solving the Steiner tree problem [118] which is again computationally intensive. A more feasible approach is to approximate such an ideal tree by algorithms such as the Prim's

algorithm [119] or to implement one of the many IP multicast protocols such as PIM [17], DVMRP [18], CBT [19] etc. In this work, the IP multicast tree is approximated by source-rooted shortest path tree to facilitate meaningful comparison with other application level multicast techniques [39][43].

As the IP multicast tree is built at the network layer with full router support and it is most efficient for multipoint communication, the total cost $C_{T_{mcast}}$ of the IP multicast tree thus serves as a practical lower bound for the optimal HPAM tree defined by Eqn. 4.5. Hence,

$$C_{T_{mcast}} \leq C_{T_{HPAM}} \qquad \textit{Eqn. 4.7}$$

To minimize the maximum latency of all clients from the source is akin to:

$$min \quad \lambda\left[P(v_s, v_i)\right] \qquad \forall v_i \in V, v_i \neq v_s \qquad \text{Eqn. 4.8}$$

$$= \quad min \quad \chi(e_i) \qquad \forall v_i \in V, v_i \neq v_s, e_i \in P(v_s, v_i)$$

Similar to Eqn. 4.5, Eqn 4.8 is also an NP-hard problem on the overlay tree. A practical lower bound to its solution is the shortest physical network path from source n_s to each and every client n_i.

The loss rate experienced by a client should ideally be zero. However, in practice, this is not achievable as it is dependent on the prevailing network conditions. Instead, a loss threshold, L_{th}, is defined, such that any loss experienced by a client is deemed acceptable so long as it is less than or equal to L_{th}:

$$min \quad loss(n_i) \qquad \forall n_i \in N, n_i \neq n_s \qquad \text{Eqn. 4.9}$$

$$= \quad loss(n_i) \leq L_{th} \qquad \forall n_i \in N, n_i \neq n_s$$

L_{th} can be adjusted according to the needs of the applications. For example, for live streaming as is the application targeted by HPAM, a relatively higher mitigating threshold may be set as users will continue to view the stream in spite of the loss since they do not have the option of revisiting the content. Whereas for stored media stream, L_{th} will need to be set lower as clients are less tolerant to losses as they can request to view the content at a later time when the network conditions improve.

4.2 Properties of HPAM Protocol

HPAM constructs overlay spanning trees in a single step process – that is, new members explicitly select their parents from among a list of members made known to them through the Directory Server (DS). The essence of HPAM's heuristics is to construct a good quality tree which possesses the following properties: (a) the shortest overlay path delay between a member and the root along the tree is at most K times the unicast latency between them, where K is a small constant and (b) each member has a limited number of neighbours in the tree which does not exceed the member's fanout bound, $k_{i,max}$, chosen to reflect the bandwidth of the member's connection to the Internet. Hence HPAM's heuristics are designed to fulfill the following requirements:

- To distribute the burden of overlay construction by sharing the task jointly among all members albeit with the assistance of a lightweight centralized directory server for better refinement of the overlay tree. A partially distributed approach is adopted so as not to overly rely on a single non-failing entity for overlay construction and control.
- To manage the dynamic membership as clients join and leave the session at any time. New members must be incorporated into the overlay tree as fast as possible and partitions caused by members leaving the group or by member failures are to be rapidly repaired.
- To gather network information as unintrusively as possible so as to adapt and improve the overlay tree into a better structure as more data becomes available over time.
- To reduce the amount of overhead traffic needed as well as state information maintained at each member in the construction and maintenance of the overlay tree.

4.3 HPAM Protocol Description

The HPAM protocol comprises four main components: directory server (DS), overlay tree construction, overlay dynamics which includes tree refinement and membership management as well as overlay robustness and repair to maintain the tree and to recover from client departure and failure. Note that the source, n_s shall hereinafter be referred to as the root, r, to facilitate easier referencing.

4.3.1 Directory Server

New clients join the multicast group by seeking the assistance of the directory server (DS). DS is a central repository which maintains the status of all members participating in the HPAM sessions. Unlike the conventional role of a centralized controller as exemplified by that in HBM [43] which includes group management (ie. client join and departure), tree construction, tree repair and maintenance, overlay topology computations, the role of DS is kept to the very minimal so that its failure will have minimal impact on HPAM's data distribution process. The primary responsibilities of DS in HPAM are for group management so as to bootstrap the parent discovery process for newly joined members as quickly as possible and to facilitate the construction of a good overlay tree at the outset. In terms of state data, as shown in Figure 4.1, DS only maintains a table of up-to-date information about all the nodes in the HPAM session. It does not maintain the current routing tables of each and every client unlike the centralized controllers of ALMI [32] and HBM or the root of distributed protocols such as Overcast [29]. Overcast root not only keeps track of the entire group membership but also logs all the changes in the tree topologies. The following presents a description of the state information maintained at DS:

- *Multicast session ID used to identify the session or sessions a client is participating in (s_i).*
- *Client ID to identify the client (n_i)*
- *Maximum fan-out of each client ($k_{,i,,max}$)*
- *Available fan-out of each client ($k_{i,}$)*

- *Client level number ($l_{i,}$) which is a monotonically increasing integer with respect to the root (source) of the HPAM overlay tree. The source has a level number of 0 and clients who are sourced directly by the root have level number of 1 and so forth.*
- *Absolute loss rate experienced by each client for the respective sessions (L_i).*
- *Unicast latency between the client and the root of the HPAM overlay tree for the respective sessions (ul_i).*
- *Parent ID to identify the parent who is sourcing data to the client for the respective sessions ($p(n_i)$).*
- *Last refresh time (r_i) to record the most recent time when update messages are received from the clients. This allows DS to monitor the aliveness of the clients and perform the necessary housekeeping functions to remove stale clients from its state tables.*

Session	**Client**	**Level**	**Unicast Latency**	**Loss Rate**	**Parent**
S_1	n_1	l_1,s_1	ul_1,s_1	L_1,s_1	$p(n_1)$
	n_3	l_3,s_1	ul_3,s_1	L_3,s_1	$p(n_3)$
	:				
S_2	n_2	l_2,s_1	ul_2,s_1	L_2,s_1	$p(n_2)$
	n_3	l_3,s_1	ul_3,s_1	L_3,s_1	$p(n_3)$
	:				

Client	**Fanout**		**Last Refresh Time**
	Max	**Available**	
n_1	$k_{,1,,max}$	$k_{,1}$	r_1
n_2	$k_{,2,,max}$	$k_{,2}$	r_2
n_3	$k_{,3,,max}$	$k_{,3}$	r_3
:			

Figure 4.1: State information maintained at DS.

As shown in Figure 4.1, the state information in DS is organized according to the different multicast sessions to reflect the respective group membership. Within each session, each row stores the details of a client participating in the session: for instance, clients n_1 and n_3 are members of session s_1 while client n_3 is also a member of session

s_2. Global data, namely the maximum and available fanouts of each client, which is relevant to all sessions, is stored separately rather than duplicated for each and every client entry in the respective sessions. The number of state entries in DS is proportional to *O(N)* where *N* is the number of members in all the HPAM trees.

The various operations of DS are listed as follows:

4.3.1.1 Bootstraps Source Discovery for New Clients

When a host joins a session, DS returns a list of potential parents to which the new member can initiate a connection. The length of the list is set at P_{max}. DS adopts a simple heuristic which serves to return a list of candidate parents which is as close to the source as possible and yet provide some diversity of candidates. The rationale is to fill up the top branches of the tree as intuitively, the higher up the tree a client is, the lower is the latency from the root. The deliberate introduction of some 'further' potential parents is to avoid the pitfall of local clustering where new clients are constantly being given the same set of potential parents. Moreover, as the tree is dynamic and network conditions change, these clients can thus move up and down the different tree branches and may become better over time while the initially 'closer' clients may deteriorate in performance. The diversity will also facilitate the expansion of the gossip network for the new clients which will in turn assist in tree improvement and partition repair.

The heuristic is shown in Figure 4.2. The existing clients in the DS table are sorted in ascending order of their unicast distance, *ul*, from the source, *r*, of the HPAM

overlay tree to form a sorted list. The source is thus the first element in the sorted list, *SortedP* since its unicast distance to itself is zero. The first $2*P_{max}$ candidates with available fanout (i.e. $k_i < k_{i,max}$) are cached. Within the cached candidates, the top $1/3*P_{max.}$ candidates will be included into the potential parent list, *P* with the other $2/3*P_{max.}$ candidates extracted from a random selection of the remaining cached candidates.

In addition to the potential parent list *P*, DS also returns a list, *L1C*, which contains all the Level 1 clients in the multicast session. This list is used in refining the data distribution tree as new members join the session. The details of its functions are documented in Section 4.3.2 Overlay Tree Construction.

```
Procedure: ProcessJoinSessionRequest (r, P,L1C)
{
    SortedP ← Existing clients in the session are sorted in increasing order of distance, ul,
              from r
    {Assert: SortedP[1] = r}

    j ← 1
    // Step 1: Cache 2*Pmax potential candidates
    while count(Cache)  < 2*Pmax do
        if SortedP[j]. kj < SortedP[j]. kj,max      // select clients with available slots
            Cache ← SortedP[j]
            inc(j)                                  // j = count(Cache)
            if  eof(DS)
               exit while loop
            end if
        end if
    end while

    // Step II: Fill Potential Parent list, P
    P ← Cache[1, 2, …, 1/3*Pmax ]                   // select 1/3*Pmax  clients closest to root
    while count(P)<Pmax
        P ← Random(Cache[1/3*Pmax+1, … , j])
    end while

    // Step III: Return list LIC which contains all the Level 1clients
    j ← 1
    while NOT eof(DS)
        if  level(SortedP[j]) = 1
```

```
                L1C ← SortedP[j]
                inc(j)
            end if
        end while
}
```

Figure 4.2: Heuristic for DS to return a list of potential parents, *P*, in response to a join request by a new client.

4.3.1.2 Handles Departure Requests from Leaving Hosts

Upon receipt of a *Leave* message from a leaving host, DS will delete its entry from the respective session's state table. If the host is involved in more than a session and it is leaving all the sessions, there will be multiple *Leave* messages sent to the DS. In addition, the fanout(s) of the respective parent(s) of the leaving host in the session(s) concerned will be incremented to reflect the free-up slot(s). Note that DS is not responsible for locating alternative parents for the children of the leaving host. This is done in a distributed manner by the affected children upon receipt of the *Leave* message from their leaving parent as shown in Figure 3.4.

For a host which fails suddenly, it will certainly not issue any *Leave* messages. Instead, the affected children will recognize that there is neither *Alive* message nor data from the parent and they will thus invoke their own switching process to locate a new parent without any interference from DS. As for DS, the prolonged absence of the *Alive* messages will be reflected in the staleness of the last refresh time, r_i, and it will delete the stale entry from its state table during its periodic housekeeping. It is acknowledged that during the transition period before the dead entry is removed, there is a possibility that DS may return the dead host as a potential parent to a new client.

This will, however, not impact the join process as the dead host will not be able to respond to a new client's QoS test (refer to Figure 3.3: Communication diagram of a join process) and hence will never become the parent to the new client. Its only impact to the new client is merely a potential parent less.

4.3.1.3 Updates Member States

All HPAM members send *Refresh* messages (also referred to as *Alive* messages) on a regular basis to DS to maintain its data up to date so as to allow it to recommend a good list of potential parents to a new member to build an efficient overlay tree. The *Refresh* message provides updates on a member's state information and contains the tuple <*available fanout, absolute loss rate, unicast latency to source, level number, parent*>. DS will update the last refresh time, r_i, upon receipt of a *Refresh* message. The heartbeat timer which determines the refresh rate directly affects the overhead bandwidth consumption as well as the overlay's response to changes in network topologies and metrics. A higher heartbeat will render the overlay more responsive to any changes albeit at the expense of added bandwidth consumption and vice versa. A higher heartbeat has been adopted by HPAM as the overhead bandwidth consumed is insignificant relative to the data bandwidth. However, in the event when there is a bandwidth crunch or when the session has stabilized, the heartbeat rate can be reduced accordingly.

4.3.1.4 Performs Miscellaneous Tasks

These tasks include periodic housekeeping to eradicate stale client data from the DS tables as well as responding to host requests to populate their gossip caches. Up-to-date DS tables not only enable new hosts to be grafted to the overlay tree as quickly as possible but also facilitates the construction of a good overlay tree. DS identifies stale host data by checking if the last refresh time, r_i, has exceeded a refresh threshold. DS also responds to host requests to top up their gossip caches when the number of gossip candidates run low. This is to ensure that there is always a ready supply of candidate parents for the hosts to quickly switch to in order to recover from a tree partition. DS selects the gossip candidates using the same criterion as that used in its response to a join request with the exception that there is no need to return the list of Level 1 clients. In the event that an orphaned host fails to find a parent, the host will issue a join request to DS. This join request will be processed as per the algorithm shown in Figure 4.2.

4.3.2 Overlay Tree Construction

HPAM protocol constructs source-based trees with different trees being built for the different sessions. The tree is by definition loop-free and is simple to construct entailing only a single-step process instead of the mesh-tree approach adopted by the like of Narada [27].

A common problem faced by all tree construction protocols building data tree on the fly is that the resulting tree is entirely dependent on the order of join of the clients. It is perfectly plausible that the level 1 (the level closest to the root) slots are occupied by clients who are 'far away' from the root in terms of RTT but by virtue of their early join, they have 'locked' up the level 1 slots. The 'nearer' clients who join the tree later are thus relegated to levels which are further away from the root of the tree thereby increasing the latency of all clients downstream of that tree branch given the high latency of their level 1 ancestors.

Centralized tree construction algorithms which regularly re-compute the optimal tree inherently eradicate this problem (e.g. HBM) since the entire tree after the root, is rebuilt. In the case of distributed algorithms, they face the same problem as HPAM since their refinement algorithms very often do not rebuild the entire tree. Moreover, distributed algorithms have knowledge of only a limited number of group members which is therefore inadequate for them to overcome this problem.

HPAM does not aim to eradicate this problem of latency-inefficient clients hogging levels close to the root of the tree as such a solution will incur too much control overhead and disrupt the data distribution process. Instead, HPAM focuses on just optimizing the level 1 clients. Once, this level is latency-efficient, the impact on the subsequent levels will be minimized. It is then left to the gossip mechanisms described in Section 4.3.3.2 to refine the tree. HPAM constructs the data tree using two algorithms, the execution of which is dependent on the proximity of the new client to the root as compared to the existing level 1 clients. If the new client is closer to the root than an existing level 1 clients, the *JoinSource&Adopt (JSA)* algorithm is

executed. Otherwise, the *Normal Join* algorithm is effected. This is summarized in Figure 4.3.

```
Procedure: RTTTest (n, P)
{
    RTT test packets from n → P
    n ← P reflects RTT test packets and provides tuple < L_P , ul_P,r >
    ul_n,P ← Average of test packets' RTT received from P
    ul_n,r (P) ← ul_n,P + ul_P,r . n estimates its latency to r via P
}

Procedure: JoinParent (n, P)
{
    JoinRequest from n → P
    n ← JoinSuccessfulAck from P, P decrements its available slot k_P by 1, adds m into its
        ChildTable, updates DS with tuple <l_P, ul_P,r , L_P, p(P), j_P, k_P >
    n ← P(data)
    p(n) ← P, n sets its parent to P
    l_n ← l_P +1, n updates its level number
    n → tuple <l_n, ul_,n,r , L_n, P, current time, k_n,max > to DS
}

Procedure: NormalJoin (n, P)
{
    SortedL ← Sort P in ascending order of latency from r, ul_n,r (P_j)
    for i = 1 to Size(SortedL)
        Potential Parent ← SortedL(i)
        if JoinParent (n, SortedL(i)) is successful
            exit forloop
        end if
    end for
    if JoinParent is unsuccessful for ∀P_j ∈ P
        Backoff and join session later
    end if
}

Procedure: JoinSource&Adopt (n, P, L1C)
{
    SortedLIC ← Sort LIC in descending order of (ul_,LIC(i),r - ul_,n,r )

    for i = 1 to Size(SortedLIC)
        if (ul_,LIC(i),r - ul_,n,r ) > C_hysteresis * ul_,LIC(i),r

            // requests to join r as parent
            JoinRequest from n → r
            n ← JoinSuccessfulAck from r
            n ← r(data)
            p(n) ← r, n sets its parent to r
            l_n ← l_r +1, n updates its level number
```

```
                //requests to adopt LIC(i) as child
                AdoptRequest from n → L1C(i)
                p(LIC(i)) ← n                    // L1C(i) sets its parent to n
                l_L1C(i) ← l_n +1                // L1C(i) updates its level number
                n ← AdoptSuccessfulAck from L1C(i), n decrements its available slot k_n by 1,
                      adds LIC(i) into its ChildTable
                LIC(i) ← n(data)                 // n streams data to L1C(i)
                n → tuple <l_n, ul_,n,r , L_n, r, current time, k_n,max > to DS
                 exit forloop
            end if
        end for
        if (JoinRequest to r) or (AdoptRequest) is unsuccessful for ∀LIC_j ∈ LIC
            NormalJoin(n,P)                      // proceed to normal join
        end if
}

Algorithm: Construct Tree (m)
{
        ProcessJoinSessionRequest (root, P)
        ∀P_j ∈ P do RTTTest(m, P_j)
        do RTTTest(m, r)
        JoinSource&Adopt (n, P, L1C)
}
```

Figure 4.3: HPAM Tree Construction Algorithm.

4.3.2.1 Normal Join Algorithm

A new host, *m*, issues a join session request to DS and DS returns a set of potential parents *P* as per Figure 4.2. The new host then measures the unicast latency $ul_{m,Pi}$ between itself and each of the candidates in *P* by exchanging packets between each other. Note that *ping* is not used for latency measurements as a lot of networks disable *ping* requests and this will impair the ubiquitous nature of the HPAM protocol. The unicast latency is determined as one-half of the round trip time (RTT) from *m* to P_i and back assuming network symmetry. In addition, the parent candidate also responds with its current loss rate L_i as well as its unicast latency to the tree root, $ul_{i,r}$. To optimize the latency of the overlay to the root of the tree, *m* attempts to join the parent candidate, *SortedL(1)* which gives *m* the lowest latency to the root.

A potential parent will admit a new child when the following conditions are satisfied:

a. The potential parent is not leaving the session

b. The potential parent is not in the process of switching

c. The potential parent has available child slots

Condition '*a*' ensures that the client is not joining a leaving parent which will orphan it soon and thus entails a parent switch subsequently. Condition '*b*' prevents a client from joining a parent while the latter is in a transient state as the potential parent's QoS has yet to stabilize while condition '*c*' ensures that the potential parent is not overloaded. Although DS has selected potential parents with available child slots, its data may be stale as the latest updates may not have arrived at DS. Hence, while admitting a new child, the potential parent will still check for child slot availability. If the initial join fails, the new client will request to join the next ranked candidate in the sorted potential parent list shown in Figure 4.3. In the event that the sorted list is exhausted with no successful join, the client will back off for a period and request to join the session again.

Upon a successful join, *m* will update its level number to one more than its parent. Note that it is the client that plays an active role in tree building with DS only kickstarting the source discovery. The tree construction algorithm is hence hybrid in nature where group membership is individually managed by the client with the assistance of the centralized DS.

4.3.2.2 Join Source & Adopt (JSA) Algorithm

The *Join Source & Adopt (JSA)* algorithm basically allows a new client, *m*, to replace an existing client to join the root of the tree. The existing client is then adopted by the new client as its child. The selection of which existing clients to replace is dependent on their respective latencies to root as compared to *m*'s latency to root. The former latencies are returned by the DS when it returns the list of level 1 clients in response to a new client's request to join a session (refer to Figure 4.2) while *m*'s latency is measured via RTT test to the root as shown in Figure 4.3.

The difference in the latencies between the existing level 1 clients and *m* are computed as per Eqn. 4.10 and an existing client is only replaced if the difference in latency satisfies Eqn. 4.11. The introduction of a hysteresis is to provide for tolerance against RTT measurement errors as well as to ensure that the improvement in latency is substantial enough to warrant the overhead incurred in disrupting an otherwise stable data tree. The latency differences are sorted in descending order to facilitate the replacement of the least 'latency-efficient' client. If none of the existing clients satisfies Eqn. 4.11, the new client, *m*, will invoke the *Normal Join* process described above.

$$\Delta Latency_{i,m} = ul_{i,r} - ul_{m,r} \qquad \text{where } i \in \text{level 1 clients} \qquad \text{Eqn. 4.10}$$

$$\Delta Latency_{i,m} > L_{hysteresis} * ul_{i,r} \qquad \text{where } L_{hysteresis} > 0 \qquad \text{Eqn. 4.11}$$

Otherwise, the new client, *m*, will issue a join request to the source as well as an adoption request to the *'to-be-displaced'* client, *q*. The source will add *m* to its children list without first deleting *q*, sends a successful join acknowledge message to *m* and commences data streaming to *m*. Hence, for a transient period, the source will have more children than its maximum slot until it receives a leave message from *q*. This is to ensure that data delivery to *q* continues to be uninterrupted until the seamless phase-in by the new parent, *m*. New client, *m*, will update the source as its parent, adopt *q* to its children list, commence data streaming, update DS. Displaced client *q*, will likewise, substitute its parent with *m*, send a successful adopt acknowledge message to *m*, send a leave message to ex-parent, *the source*, as well as update its children and DS of its new status.

The point to note is that the *JSA* algorithm is targeted at optimizing level 1 clients only. The fixed fanout supported by the root renders the resulting overhead manageable. It can be observed that this may be done at the expense of the latency to root of all the successive nodes in the affected branch as these nodes move a level down the tree with the adoption of *q* by *m*, unless, of course, the latency difference, $\Delta Latency_{q,m}$ is sufficiently huge. To iteratively apply *JSA* algorithm to the different tree levels will be akin to recomputing the entire data tree and the overhead will be in the order of *O(N)* for every new client. This is not feasible. Instead, HPAM relies on its gossip mechanism described in Section 4.3.3.2 to perform the rest of the tree improvement. To facilitate the process, the displaced client, *q*, will, after a random period, issue a request to DS to furnish it with a new list of gossip candidates. The random wait is to allow DS time to be updated with the latest status.

4.3.3 Overlay Dynamics

Overlay dynamics is a direct result of changes in tree membership and network conditions. The tree construction algorithm described in Figure 4.3 strives to build an overlay with latency which is as low as possible for each client on the fly with limited knowledge of global network and client information. As operating conditions changes, the information has to be made available to the clients so that they can adapt and improve the overlay tree and over time, the overlay can converge towards a minimum within the constraint of the respective client fan-outs. There must therefore be some means for the clients to gather the prevailing environmental conditions and thereafter to utilize such information to refine the overlay tree.

4.3.3.1 Environment Monitoring

Monitoring of the network environment is accomplished as unintrusively as possible through three means: periodic end-to-end measurements of RTT, refresh messages between pairs of clients and the determination of the loss rates experienced by the individual clients. RTT is used as a performance indicator of the prevailing network condition as it is easier to measure and has much less overhead compared to active bandwidth measurement techniques. The latter will require the sending of huge packet streams to determine the network bandwidth which is highly intrusive and incurs high network overheads. The refresh messages, besides serving as heartbeats for the clients, also serve the dual function of updating one another with their current status.

Finally, the loss rate is a moving average of the number of lost data packets experienced over a time period. This can be easily determined from the data packets a client received from its parent. Loss rate also serves to provide an indication of the prevailing QoS experienced by each client.

All clients periodically conduct RTT measurements to their respective parents by exchanging packets with one another as described in Section 4.3.2 as well as with the members in its gossip cache. Clients also receive updates on the latency to root, loss rate experienced by the parents and gossip cache members in addition to computing their own loss rate.

4.3.3.2 Tree Quality Improvement

Tree quality improvement is effected via a client switching to a parent which will provide it with a better QoS. The switching is made possible through the gossip mechanism introduced in Section 3.3.2, whereby each client maintains a ready list of potential parent candidates in its cache. Being built from the initial list of potential parents returned by the DS, the gossip cache is somewhat static as the gossip candidates can become full and can no longer be a potential parent. Moreover, as membership is dynamic, some of these gossip candidates may have failed or left the session. These incapacitated candidates are identified by the client from the refresh message exchanges or the lack of them in the case of a departure or failure and will be removed from the cache. There is thus a need to replenish the cache over time. Replenishment is performed via two means. The normal process is embedded in the

refresh message exchanges between the client and its gossipers. The gossiper will furnish the client with a random selection of gossipers from its own gossip cache to increase the client's gossip repertoire.

The second method is for the client, *n*, to request DS for assistance to supply a fresh list of potential parents. This is only used in the event that the gossip cache dwindles to below a third of its full capacity so as not to flood the DS.

The gossip cache is arranged in the order of priority of the client's latency to root if it were to use the respective gossiper as a parent. This order is dynamic as the QoS of the gossipers change over time based on the prevailing network conditions. A client will periodically evaluate the change in the latency which it can derive if it were to switch its parent to each of the gossip cache candidates (refer to Eqn. 4.12). A positive value implies that the latency via the new parent to the root is better than its current latency to the root. To damp any switching oscillations caused by marginal improvement to the unicast latency as well as to cushion any effects on the system due to fluctuations in the end-to-end RTT measurements, a hysteresis, $C_{hysteresis}$ is introduced into Eqn. 4.12 and a gain factor, *QoSGain* is computed as shown in Eqn. 4.13. *QoSGain* must be positive for a switch to be effected. A client will choose the parent which maximizes its *QoSGain* to switch to. This mechanism for improving the quality of the HPAM data tree is termed *Proactive Tree Refinement* as the client takes the initiative to check for possible improvement to be made to the data tree.

$$\Delta Latency_i = (ul_{i,r} + ul_{i,n}) - ul_{n,r} \qquad \text{where } i \in \text{gossip cache} \qquad \text{Eqn. 4.12}$$

$$QoSGain_i = \frac{\Delta Latency_i - C_{hysteresis} * ul_{n,r}}{C_{hysteresis} * ul_{n,r}} \qquad \text{Where } C_{hysteresis} > 0 \qquad \text{Eqn. 4.13}$$

Before a physical parent switch, the client will also check that the following conditions are satisfied to further avoid switching oscillations:

a. The client is not leaving the session
b. The client is not in the process of switching
c. The client has not received information that its current parent is also doing a switch

Conditions '*a*' ensures that the client is not leaving the session while condition '*b*' helps to prevent looping by disallowing simultaneous switching. Condition '*c*' prevents a client from switching when its current parent is in a transient state. In the transient state, the QoS (eg. RTT to root) delivered by the new and current parent have yet to stabilize and a switch during this period may prove futile as the previously computed *QoSGain* may be rendered invalid. Having satisfied itself that it is the opportune moment to switch, the client will inform its children about its switching and proceed to join the new parent as per procedure *JoinParent* in Figure 4.3. The potential parent will check that it has available slot and is neither leaving the session nor in the midst of a switch as per the *Normal Join algorithm* described in Section 4.3.2.1 before it admits the child during tree construction. The only additional step is to inform its current parent to stop data streaming. The terminated parent will remove *m* from its children table and update DS accordingly. For a potential parent to admit

an existing client as opposed to a new client, it is imperative that the potential parent checks that it is not in the process of switching. This helps to prevent looping by disallowing simultaneous switching.

Over time, as the client discovers new parent which improves its *QoSGain*, it will progressively switch its parent until an equilibrium is reached. The overlay tree is thus refined through the distributed efforts of the clients as each and every one strives to minimize its network latency to the root. Figure 4.5 summarizes the overlay tree refinement algorithm.

```
Algorithm: Refine Tree (n)
{
    { Assert: gossip cache of n ← adds gossip recommendation embedded in refresh
                                  messages from P if not already in cache}

    ∀Pj ∈ P
    {   if ((current time - last refresh time rj ) > Refresh_thres) or (available slot kj=0)
            remove Pj from gossip cache
        end if
        RTTTest(n, Pj)
        n computes QoSGain_j
    }
    if size(gossip cache) < 1/3*full capacity
        n sends RequestforParents → DS
        n deletes Pj ∈ ChildTable and p(Pj) ∈ ChildTable
                            // Deletes Pj which are children or grandchildren of n
    end if
    Potential Parent ← Pt where (QoSGain_t > 0) and (QoSGain_t = max(QoSGain_j ))
    if current parent pn(t) is not switching and Pt is not switching and Pt is not leaving
                                                                    session
        JoinParent (n, Pt)
    end if
    if JoinParent (n, Pt ) is successful
        LeaveRequest from n → OP, Old parent. OP discontinues data streaming,
                              deletes n from its ChildTable, increments available slot
                              k_OP by 1, updates DS with tuple <l_OP, ul_OP,r , L_OP, p(OP),
                              j_OP, k_OP >
    end if
}
```

Figure 4.4: Overlay Tree Refinement Algorithm.

4.3.3.3 Tree Membership Dynamics

Membership on the overlay tree is dynamic as clients join and leave the tree. Moreover, client can experience failures too. A leaf client who leaves the overlay tree or fails has no impact on the distribution tree. The leaving leaf client just sends its parent a *LeaveRequest* and informs DS about its departure while a failed node will be sensed by the parent when the parent does not receive the heartbeats from its child. However, an intermediate client leaving has an impact since it is streaming data to its children. A voluntary leaving client will inform its children and parent about its impending departure to preempt its children to locate and switch to new parent(s). In the meanwhile, it will continue to forward data until all the children have successfully switched to the new parent(s) or when *timeout* occurs whichever is earlier. Thereafter, the leaving client will inform DS and depart from the tree. The procedure for a graceful departure is shown in Figure 4.5. The *timeout* is necessary as the leaving client cannot wait indefinitely for all the children to switch successfully.

```
Procedure: Leave Tree (n)
{
    if size(ChildTable) > 0                          // for intermediate client nodes
        LeaveRequest from n → C  ∀C ∈ ChildTable_n
        wait for timeout or SwitchSuccess from  ∀C
    end if
    LeaveRequest from n →  p(n). Parent  discontinues data streaming, deletes n from
                                its ChildTable,  increments available slot k_P(n) by 1,
                                updates DS  with  <l_P(n), ul_P(n),r , L_P(n), p(p(n)), j_P(n), k_P(n) >
    LeaveRequest from n → DS. DS deletes n from its tables
}
```

Figure 4.5: Procedure for Graceful Client Departure.

An involuntary departure by a client causes an abrupt break in the data distribution tree. This failure needs to be detected locally and the partition repaired in a timely manner to minimize the disruption in data distribution to the downstream clients. Note that partition can also occur in the case of a voluntary departure. In the event that one or more of the children of a voluntary departing client have yet to switch successfully within the *timeout* duration, and the client has proceeded to depart from the session, partitions will occur as these children are left orphaned. As HPAM is based on a tree-only approach, the common approach of using a mesh to recover from abrupt failure cannot be adopted. Other techniques are thus needed to improve the robustness of the overlay and to speedily repair such partitions.

4.3.4 Overlay Robustness

A major challenge faced by the HPAM overlay is the vulnerability of the overlay tree to partition as a result of client failure or voluntary departure. A partition occurs when a client failure or departure cuts off the data reception to its downstream clients, thereby isolating all clients belonging to this branch from the HPAM data delivery tree. Two mechanisms, namely the gossip and the spiral mechanisms [43], are introduced to facilitate speedy partition repairs to ensure the robustness of the overlay. In addition, although a tree is by definition loop-free, the incorporation of a tree refinement algorithm to allow the switching of parents to improve the overlay's QoS may lead to looping. Partition recovery too may result in looping. Detection and

avoidance of loop is another issue that needs to be addressed to ensure the robustness of the overlay.

4.3.4.1 Partition Detection

A partition is detected when a client fails to receive heartbeats and/or data from its parent for a timeout duration. To the client, it implies that the parent has failed and it must switch to a new parent as soon as possible to minimize the data disruption to itself and its children. By virtue of the organization of the HPAM data distribution tree, the orphaned client will detect the failure first before its children. It will then broadcast a message to inform its children to backoff from switching until after a timeout duration as it is performing a switch operation. The purpose of the message is to prevent an influx of parent switching from its descendants while the affected client seeks to repair the partition. The message will be propagated downstream by its respective children. The downstream nodes will only perform their own independent parent switching if they fail to receive any data from their parent after the timeout.

4.3.4.2 Partition Repair

A three-layer strategy as summarized in Figure 4.6 has been set up in HPAM to facilitate the speedy repair of the occasional partition by the immediately affected clients in the following order of precedence:

- Layer 1: Affected client is to join the spiral node. A spiral node is a hot standby node which will adopt an orphaned client if a ‘distress’ join request is

received. It will adopt an orphaned child up to its maximum slot plus 1. Any more 'distress' join beyond the 1-slot overload will be rejected so as not to overly compromise the QoS delivered to the rest of its children.

- Layer 2: Affected client is to choose the gossip candidate with the maximum *QoSGain* in its gossip cache and issue a join request. The client may be rejected if the gossip candidate happens to be full or is leaving the group or if it fails. The orphaned client will repeat the procedure with the next *QoSGain*-ranked gossip candidate until it finds a parent or it exhausts its gossip cache whichever is earlier.

- Layer 3: Issue *JoinSessionRequest* to DS to 'rejoin' the session as per a new client.

```
Procedure: Repair Partition(n)
{   if spiralnode <> NULL              // attempt to join spiral node, SN
        JoinParent (n, SN)
    else if count(GossipTable) > 0     // attempt to join the gossiper with best QoSGain
        Remove G1 from GossipTable
        JoinParent (n, G1)
    else
        JoinSessionRequest (r, P)      // request DS to rejoin session
    end if
    wait for timeout or SwitchSuccess from ∀C
    if JoinParent is successful
        Updates DS
    else
        repeat Repair Partition(n)
    end if
}
```

Figure 4.6: Partition Repair.

Layer 1: Spiral Mechanism [42]. The spiral mechanism of a newly joined client is developed in two phases. The initial phase takes place at the point of joining and is aimed at providing additional robustness to the client at the earliest possible point in time. Here, the client maintains an additional connection with its grandparent or parents' siblings in the case of Level 2 clients (shown as *Initial spiral connections* in Figure 4.7). This spiral connection allows a client to withstand node failures in any of its overlay tree branches so long as these failures are not consecutive nodes of the same branch. The problems with the initial phase of establishing the spiral nodes lie in the limited number of potential candidates for spiraling as well as the potential overload of the grandparents or parents' siblings as each of them forms a one-to-many relationships with clients who are lower down the tree.

Phase 2 involves assigning new spiral node to a client after it has commenced the gossip process with the candidates in its gossip cache. As membership increases over time, a client has more choice for its spiral connections. Spiral node who has reached its maximum slot or who has not been sending heartbeats will be replaced. The substitute is the gossip candidate which returns the best root latency. As shown in Figure 4.7, *C4* has replaced its initial spiral connection to *C2* (Note that *C2* is being shared by *C4* and *C5* as spiral node) with a new spiral connection to *C7*. Likewise, *C10* has replaced its initial spiral connection to *C1* with a new spiral connection to *C11*. By spreading out the spiral nodes to clients rather than letting clients share the same spiral node helps to improve their resilience in the event of consecutive node failure. Then, they do not need to compete against one another to join the same spiral node.

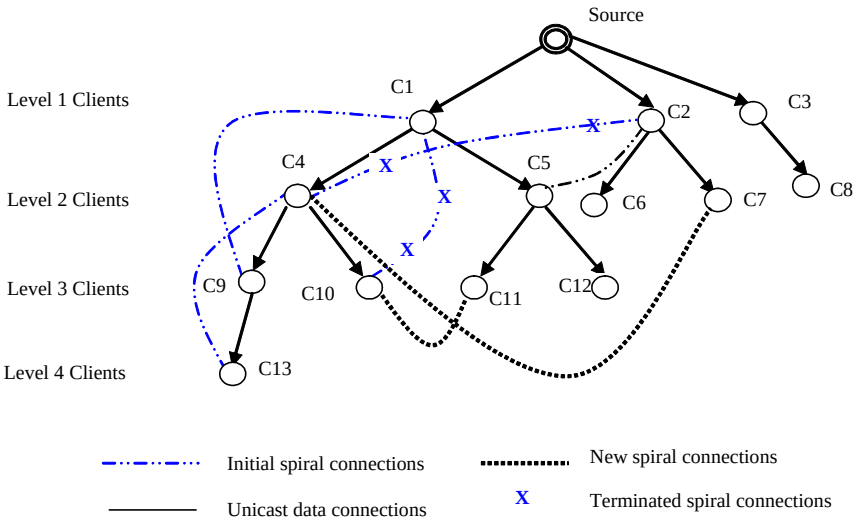


Figure 4.7: Phases 1 and 2 of spiral mechanisms

Layer 2: Gossip Mechanism. As described in Section 4.3.3.2, every client establishes a small random network with some other existing clients to not only facilitate the progressive refinement of the HPAM tree but also to recover from partition problem. The gossip mechanism is in no way as well-connected as the mesh network set up by Narada, but it serves the purpose of repairing a partition by enabling a 'partitioned' client to re-latch itself to the HPAM tree via one of its gossiper. Figure 4.8 illustrates the partition repair process. Given that the spiral nodes of clients *C10* and *C9* are unable to accommodate them (the spiral nodes are either already full or have failed), the gossip mechanism will kick in to repair the partition caused by the failure of *C4*. *C10* will choose *C5* to be the new parent as it is at the top of its gossip cache yielding the highest *QoSGain* for *C10* while *C9* will choose *C2* for the same reason.

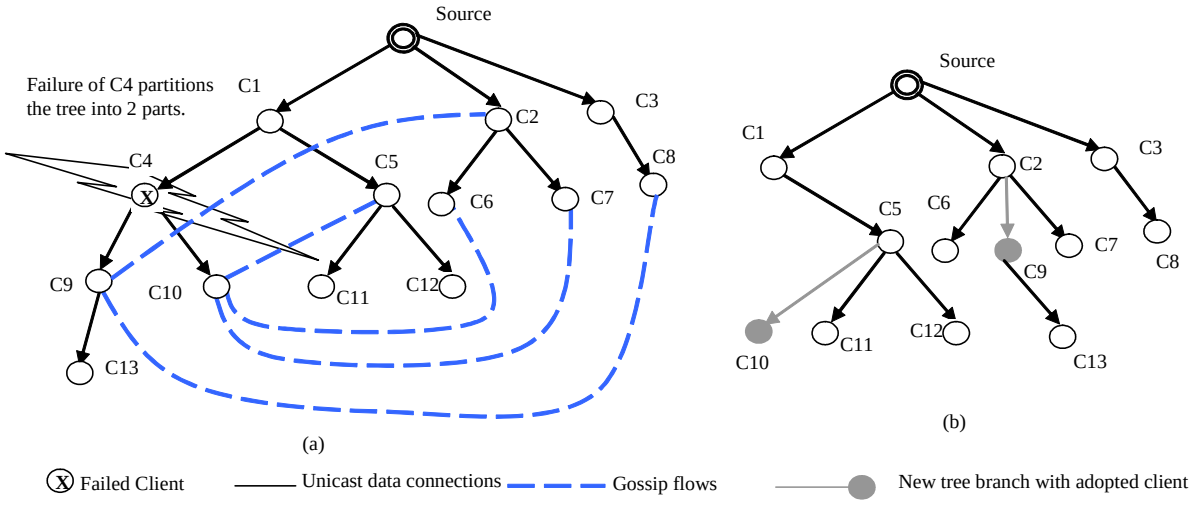


Figure 4.8: Partition Repair using Gossip Mechanism. Panel (a) shows the partitions formed when C4 fails. C9, C10 and C13 are detached from the HPAM tree. Panel (b) shows the re-attachment of the orphaned clients C9 and C10 to their new parents C2 and C5 respectively which are gossipers in their respective gossip caches.

Layer 3: Request DS to Rejoin Session. This is the last recourse for an orphaned client when it fails to join its spiral node or any of the gossipers in its gossip cache. DS will provide it with a fresh list of potential parents to join.

4.3.4.3 Loop Detection

HM [39] and OMNI [41] require each client to maintain its path to the root of the data distribution tree in order to detect for loops. Loop detection is hence simplified so long as a client does not choose any one in its return path to be its new parent. The complexity lies in that any change in a client's parent will require the root paths of all its descendants to be updated. Moreover, loops can still occur during this transition period of updating root paths. HPAM, on the other hand, does not need any special loop detection and resolution mechanism and does not require its clients to maintain

the root paths. Instead, HPAM incorporates a loop avoidance mechanism into its protocol. Nevertheless, the loop avoidance mechanism cannot guarantee loop freedom given the dynamic nature of application level multicast. However, it does minimize the occurrence of loops and their resultant effects on HPAM. Membership changes, parent switching by different clients to refine the HPAM tree and to repair partitions are constantly changing the HPAM tree dynamics. Clients and DS are kept abreast of the current status through refresh exchanges and updates. However, the inevitable intervals between the refresh and update periods may result in a delayed reflection of any changed status which is unknown at the point of evolution of the tree dynamics. This may in turn lead to loops being formed albeit only temporarily.

In the eventuality that a loop actually occurs in spite of the loop avoidance mechanism, the detection process is subsumed under HPAM's partition detection and repair algorithm. When a loop occurs, it results in a partition. The clients in the loop will experience poor or no data reception. This will trigger HPAM's self-improving protocol to kick in. The first client which detects the partition will institute a parent switch to seek a new parent and break out of the loop. The process follows that described in Section 4.3.4.1 for partition repair.

4.3.4.4 Loop Avoidance

Loop avoidance in HPAM is accomplished via the use of a monotonically increasing level number to reflect the hierarchical position of a client with respect to the root of the HPAM tree. As shown in Figure 4.6, the children to the source of the HPAM tree have level number being set to 1 while the grandchildren's level number is set to 2

and so forth. When a client latches on to a new parent, it will update its level number to that of its parent's plus 1. This mechanism will not cause a loop assuming steady state conditions and that multiple parent switches are not concurrent. For clarity of reference hereinafter, clients who are nearer to the root of the tree are said to have a lower level number or to reside higher up the tree. Conversely, clients who are further away from the root of the tree are said to have a higher level number or to reside lower down the tree.

Loop avoidance mechanism is activated under the following conditions:

a. During partition repair when an orphaned client has no spiral node or gossiper to latch to and has to revert to DS to rejoin the tree.
b. When a client requests DS to replenish its diminishing gossip cache.
c. During gossip exchanges to remove gossip candidates who may cause looping as their status change with time.

In instances '*a*' and '*b*', DS will return a list of potential candidates, *P*, as per the algorithm listed in Figure 4.2 for new clients seeking potential parents but *P* must satisfy the level number constraint set out in Eqn. 4.14 in order to ensure loop freedom. The level number, *l*, enables DS to distinguish an existing client from a newly-joined client since a new client will have *l* set at 0. Note that Eqn. 4.14 does not apply to new clients as they have yet to join the HPAM tree and therefore have no descendants to pose a potential loop problem.

$$l_{P_i} \leq (l_n + 2) \qquad \forall P_i \in P \qquad \text{Eqn. 4.14}$$

The inequality will return candidates up to 2 levels lower down the tree than client *n*'s current level. Ideally, DS should not return potential parents which are located lower down the tree than the client itself to avoid loop. However, noting that clients who are close to the root, in particular, levels 1 and 2 clients, would have a very limited number of candidates who are higher up the tree or equal in level to them, there is a need to provide these clients who are near to the root with a wider set of gossipers. Hence, Eqn. 4.14 in itself cannot fully avoid looping and this responsibility has to be shared between DS and the individual clients. In other words, after DS has returned the list *P*, client *n* will weed out those candidates which are its children or grandchildren. DS is actually capable of undertaking the entire responsibility given that it stores the parent of a client in its state tables as shown in Figure 4.1 and is therefore able to determine the parent-child relationship of any client. However, distributing part of the job to the clients helps to greatly simplify its task and avoid undue overloading on its part.

Notwithstanding the loop avoidance mechanism, when the HPAM tree membership is smaller than P_{max} (the maximum number of potential parents returned by DS), there is a possibility that DS may end up with a list comprising solely of the descendants of a client. This condition is only transient and it will certainly improve over time when membership increases as the heuristic adopted by DS in Figure 4.2 is predisposed to serving up potential parents with lower latency. A lower latency parent will usually have a lower level number although this is not necessarily always true. Nevertheless, in most cases, most part of the potential parents list will be filled with parents with lower level numbers (i.e. higher up the tree) than the client instead of its descendants.

In case '*c*', during gossip exchanges, the client will also check the level number of the candidates in its gossip cache and weed out candidates who fail to satisfy Eqn. 4.14 as well as those who are its descendants.

4.3.5 Packet Format

Basically, two types of packet formats are defined in HPAM, namely, the control message and the data packet formats. All HPAM protocol control messages have the same header as shown in Figure 4.9. The session and source IDs serve to uniquely identify the data delivery tree used for the session as HPAM builds source-specific overlay trees. The payload will differ depending on the message type. The various message types used in the HPAM protocol together with the corresponding control fields are tabulated in Table 4.1. Both TCP and UDP message packets are used.

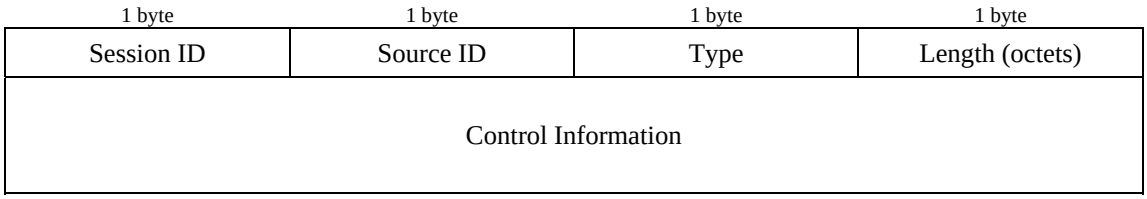


Figure 4.9: Control message format for TCP/IP and UDP/IP

The data packet format is shown in Figure 4.10 and the data packets are forwarded using UDP/IP. Besides the session and source IDs, a parent will embed its current unicast latency to source, its loss rate as well as its level number in the header of the data packets streamed to the children. This will ensure that the children are always being provided with the most up-to-date status of itself.

Table 4.1: Summary of the different types of HPAM protocol control messages

Type	Message	From	To	UDP/TCP	Fields
1	JoinRequestDS	New Client	DS	TCP	Level, fanout
2	JoinReplyDS	DS	New Client	TCP	IP addresses, unicast latencies to source, loss rates of potential parents and level 1 clients
3	GossipRequestDS	Client	DS	TCP	Level number
4	GossipReplyDS	DS	Client	TCP	IP addresses, Unicast latencies to source, loss rates, levels of potential parents
5	UpdateDS	Client	DS	UDP	Unicast latency to source, loss rate, children number, level
6	JoinRequestPeer	Client	Potential Parent	TCP	-
7	JoinConfirmPeer	Potential Parent	Client	TCP	Level, unicast latency to source, loss rate
8	LeaveNoticeDS	Leaving Client	DS	TCP	-
9	LeaveNoticeChild	Leaving Client	Children	TCP	-
10	LeaveNoticeParent	Leaving Client	Parent	TCP	-
11	HeartbeatParent	Children	Parent	UDP	-
12	PingRequest	Client	Client	UDP	Requestor's timestamp
13	PingReply	Client	Client	UDP	Level, unicast latency to source, loss rate
14	GossipPingRequest	Client	Gossiper	UDP	Client's timestamp, unicast latency to source, loss rate, children number, level
15	GossipPingReply	Gossiper	Client	UDP	Unicast latency to source, loss rate, children number, level of Gossiper and IP address of one randomly selected peer from Gossiper's gossip cache.
16	SwitchNoticeChild	Switching Client	Children	TCP	-
17	JSARequestSource	New Client	Source	TCP	Fanout
18	JSAConfirmSource	Source	New Client	TCP	-
19	JSAAdoptChild	New Client	Level 1 Clients	TCP	Level, unicast latency to source, loss rate

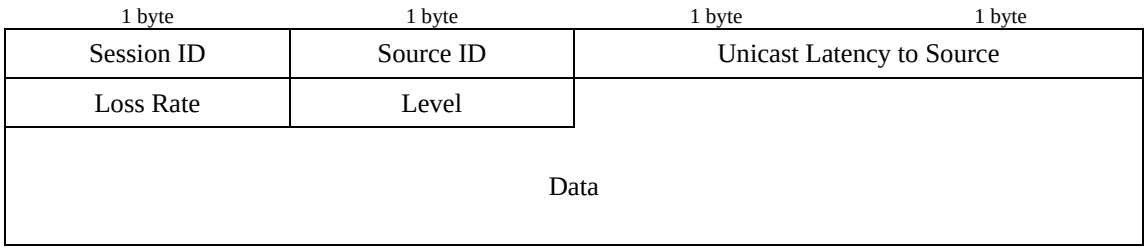


Figure 4.10: Data Packet Format for UDP/IP

4.4 Analysis

The lower bound to the solution to Eqn. 4.8, to minimize the maximum latency to the source, is the shortest physical path from a client to the source. As HPAM's protocols are based on heuristics and the HPAM tree is being built on the fly, the dynamic behaviour of HPAM protocol is difficult if not near impossible to model. As such, a quantitative analysis of its performance is not directly achievable. Instead, an upper bound of the overlay path length can be derived based on the condition that the HPAM membership is stable and known and an ideal HPAM tree is built in increasing order of unicast latency from the root. This upper bound serves as a barometer to evaluate the performance deviation of a typical HPAM tree from an ideal HPAM tree. The amount of state entries maintained by each client and the control overheads incurred per client are also analysed.

4.4.1 Upper Bound of Overlay Latency for Ideal HPAM Tree

An ideal HPAM tree is a tree in which the nodes are grafted onto the tree in increasing order of overlay latency to the source as shown in Figure 4.11.

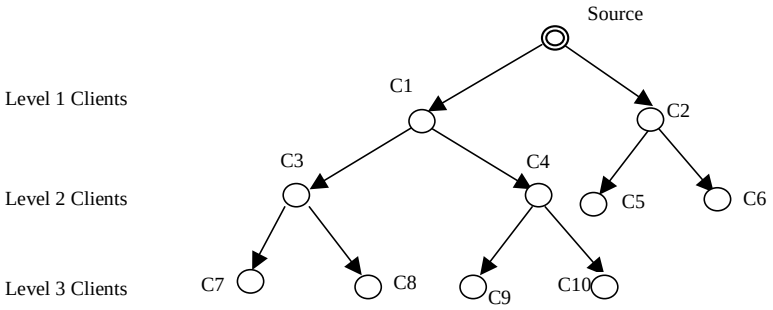


Figure 4.11: An ideal HPAM Tree where unicast latencies to the source for the clients obey $ul_{1,r} < ul_{2,r} < ul_{3,r}$ etc. along a path. The outbound degree for each client is 2 here.

To derive an upper bound on the overlay latency to source for the ideal HPAM tree, two assumptions are made. The first assumption is that the unicast path latencies are symmetric i.e. $ul_{i,j} = ul_{j,i}$. The second assumption is for any nodes on the HPAM tree, n_i and $n_j \in N$, their respective unicast latencies to the root, $ul_{i,r}$ and $ul_{j,r}$ follow the triangle inequality $ul_{i,j} \leq ul_{i,r} + ul_{r,j}$. Likewise, for their overlay paths to the root.

Consider a node n_i on the ideal HPAM tree and any node n_j that lies in the overlay path from r to n_i, $ul_{r,j} \leq ul_{r,i}$. For any 2 nodes n_j and n_k on the overlay path from r to n_i, using symmetry and the triangle inequality,

$$ul_{j,k} \leq ul_{j,r} + ul_{r,k} = ul_{r,j} + ul_{r,k} \leq 2ul_{r,i} \qquad \text{Eqn. 4.15}$$

Let $l_i \subseteq E$ be the set of logical connections in the overlay path from r to n_i. Given that the minimum out-degree of any node is d, it follows that $|l_i| \leq log_d N$. Thus, the overlay path $P_{HPAM}(r,n_i)$ from r to n_i is :

$$P_{\text{HPAM}}(r,n_i) = \sum_{(n_j,n_k)\in N} ul_{j,k} \leq 2ul_{r,i}|l_i| \leq 2ul_{r,i}\log_d N \qquad \text{Eqn. 4.16}$$

Eqn. 4.16 shows that the overlay path length of any node to the root on the ideal HPAM tree is bounded by $2ul_{r,i}log_d N$. Eqn. 4.16 can be rewritten as Eqn 4.17 to reflect the upper bound of $2log_d N$ on the *Stretch* or *Relative Delay Penalty (RDP)* which is defined as the ratio of overlay path to unicast latency in Section 2.5.2.1.

$$\frac{P_{\text{HPAM}}(r,n_i)}{ul_{r,i}} \leq 2\log_d N$$

$$RDP \leq 2\log_d N \qquad \text{Eqn. 4.17}$$

4.4.2 State Maintained by Each Client

Each client n_i maintains the following state information:

- its unicast latency to root
- its unicast latency to parent
- the absolute loss rate in terms of the data packets not received
- its level number
- the addresses of its children
- the addresses of its gossip candidates and its spiral node
- its unicast latencies to the respective gossip candidates and spiral node

The maximum state entries per client is at most $O(k_{i,max}$ + *number of gossip candidates)* entries where $k_{i,max}$ is the maximum fanout of a client n_i. State entries per client are being kept at a minimal level as HPAM does not require the client to maintain the overlay path from the root to itself and neither does it require each client to maintain state for the entire membership. The advantage of a small state information table lies in that it reduces the client's load of maintaining the table and processing refresh messages considerably.

4.4.3 Control Overhead Per Client

The control overheads incurred by a client include all HPAM protocol messages used in joining, maintaining, refining and repairing the HPAM data distribution tree and are summarized as follows:

- Periodic heartbeats from children
- Periodic heartbeat to DS
- Periodic RTT test messages to parent
- Periodic gossip exchange messages with gossip candidates
- Parent switching messages due to tree refinement or partition repair
- Occasional requests to DS to top up gossip cache
- Join request for initial join as well as for the occasional partition repair

Hence, the HPAM protocol involves an aggregate control overhead per client of $O(k_{i,max}$ + *number of gossip candidates + 2)* which is the same order of magnitude as

the average state maintained by the client. The constant '*2*' takes into account updates to DS and RTT test messages to the parent. This is a direct result of the lower amount of state information maintained at the client side. Unlike the mesh-based system such as Narada whose state information per client is *O(N)*, HPAM clients incur much less overheads in the refreshing the state entries. Hence, *O(N)* is the total control overhead of an HPAM tree while the Narada mesh incurs a total control overhead of $O(N^2)$.

The steady state *POR* per client per second can be computed as follows:

$$POR = \frac{(DSF + HbF + RTTF + m * GF) * ControlPacketSize}{Data\ Source\ Rate} \qquad \text{Eqn. 4.18}$$

where *DSF:* update frequency to DS

HbF: heartbeat frequency

RTTF: frequency of RTT test (2 messages per test since each test involves a roundtrip)

GF: frequency of gossip exchange messages per gossiper (2 messages per exchange)

M: maximum number of gossipers in the gossip cache

This steady state *POR* does not take into account other overheads such as queries to DS, join requests and join accepts, switch wait messages. These are more transient in nature and it will become negligible when the session has stabilized.

4.5 SUMMARY

In this chapter, the detailed algorithms used by HPAM to construct, maintain, improve and repair the data distribution trees are presented. Analysis is performed to determine the upper latency bound of a client in a HPAM tree, the amount of state maintained and the control overhead associated with each HPAM client. The novel heuristics adopted in the HPAM protocol include the *JoinSource&Adopt* algorithm to optimize the level 1 (level closest to root) of the data tree to reduce root latency of each clients, the *Gossip* mechanism to refine and repair the data tree, the hybrid use of a centralized albeit lightweight *DS* and the distributed *Gossip* and *Spiral* mechanisms to facilitate tree construction, refinement and repair with lower overheads than a fully distributed system and a quicker response to group dynamics and network environment than a centralized system.

Chapter 5

Performance Evaluation of HPAM

In Chapter 5, the HPAM protocol is evaluated through a comprehensive set of simulations complete with discussions on the efficacy and limitations of the protocol. The simulation setup and the performance metrics used in the evaluation are described. The impacts of the *JoinSource&Adopt (JSA)* algorithm and the gossip mechanism on the HPAM protocol are investigated. A comparison is also made between HPAM and Host Base Multicast (HBM) [43], a centralized protocol as well as between HPAM and Host Multicast (HM) [39], a distributed protocol.

5.1 HPAM Performance Evaluation

The properties and qualities of the overlay tree produced by HPAM and the overheads associated with HPAM protocols are evaluated through a series of detailed simulations and analysis.

5.1.1 Performance Metrics

HPAM's performance is evaluated along the following dimensions:

- *Stretch* or *Relative Delay Penalty (RDP)* [52] defined as the ratio of overlay delay over physical unicast delay. In HPAM the source *RDP* (which will be termed as *RDP* hereinafter) is used to quantify the quality of the HPAM data path with respect to the root of the tree.

- *Stress* [52] is another parameter used to quantify the quality of the data path and is defined as the number of duplicate copies of data traversing the physical network. It therefore represents the additional network resources consumed by HPAM compared to ideal multicast in the delivery of the data.

- *Tree Cost Ratio (TCR)* [32][39] defines the relative cost of a HPAM tree to those of source-rooted shortest path multicast trees. TCR provides an indication of how efficient HPAM uses the network resources compared to router-supported IP multicast. The tree cost is the sum of the delays on the tree links.

- *Control Traffic Overhead or Protocol Overhead* can be represented by the *Protocol Overhead Ratio (POR)* [52] which defines the ratio of the total control bandwidth incurred by the HPAM clients over the total bandwidth consumed by the data. The *POR* is presented as a percentage hereinafter to facilitate easier reading as its value is very small.

- *Adaptation to Group Dynamics and Tree Improvement Ability.* The responsiveness of the overlay to the changes in group membership and the time taken to recover from a partition will reflect the adaptation ability of HPAM protocol. The effectiveness of HPAM's tree refinement protocol is observed from its ability to improve the *RDP* of its clients over time.
- *Total Message Load on DS.* The amount of message traffic to the server DS provides an insight into its loading. This traffic comprises essentially of client status update messages and query messages. The latter includes requests to join or rejoin the session and requests to fill gossip caches.

Serving as a reference for evaluation are naïve unicast and IP multicast implemented as shortest path tree. Naïve unicast is implemented using Dijkstra's algorithm [117] and has a RDP of 1. This serves as a theoretical lower bound for HPAM's stretch performance (refer to Eqn. 4.8). Multicast, with a stress of 1, serves as a lower bound for HPAM's stress performance (refer to Eqn. 4.7).

5.1.2 Simulation Methodology

A packet-level, event-based simulator written in ModSim [120] is used to evaluate HPAM protocol. The simulator implemented the network routing protocol based on shortest path between any two members and models the propagation delay of physical links but does not model link loss, queueing delay and background traffic. This is firstly, because it is difficult to model Internet dynamics in a reasonable way in a

simulator and secondly, to render the simulations to be more scaleable. Thirdly, the interplay of link loss, queueing delay and background traffic will contribute too many random variables into the simulation which will lead to the masking of the actual characteristics of HPAM especially when the results which are presented in this chapter is the average results of numerous simulation runs. This is the primary reason that most simulations for application level multicast presented in the literature survey in Chapter 2 do not include such simulation. Instead data loss due to sudden failures of clients are simulated to enable a detailed study of the dynamics of HPAM protocol.

The network topologies used in the simulation are generated using the GT-ITM topology generator [121]. Two different models are used in generating the network topologies, namely, the Waxman random graph model [122] and the transit-stub model. All topologies have 600 routers with average node degree of <3,5> and approximate number of edges between 2400 to 3300. The link delays varied from 1 to 5 ms. Clients are attached to routers selected at random and in the case of the transit-stub model, they are randomly chosen from among the stub-domain nodes. Each member is assigned a random delay of up to 2 ms to simulate the heterogeneous nature of the clients. The outbound node degree for each member is fixed at 3 to facilitate comparison with analytical performance bounds. Five different topologies from each of the two models are used in the simulation experiments.

A single host is chosen at random to be the data source generating a constant bit rate and another chosen to be the DS. Experiments with group sizes ranging from 50 members to 300 members in steps of 50 members are run across these topologies. Each experiment is repeated 5 times on each of the 5 network topologies, each time

with a random selection of different members. In other words, each data point presented in the figures in this section is the average results of 25 simulation experiments.

A fixed number of members join the group in the first 100 seconds of simulation in random sequence. The simulation is run for 600 seconds. During simulation time <350s,450s>, a random selection totaling 15% of the group members will gracefully leave the group while 10% will depart suddenly to simulate client failures. This means a total of 25% of the original group members would leave the session within a duration of 100s.

5.1.3 Simulation Results

This section focuses on the characteristics of HPAM overlays. Such characteristics are invariant across different group sizes and network topologies. The figures shown here reflect a group size of 100 members running on the transit-stub network topology. Plots which do not indicate the simulation time are captured at t = 105s when the HPAM tree has stabilized but before the departure duration <350s,450s>. The heartbeat period is set to 1s to allow faster recovery from client failures albeit at the expense of increased protocol overhead.

5.1.3.1 Relative Delay Penalty

Figures 5.1 and 5.2 show the RDP and overlay delay of each client to the source versus its respective physical unicast latency. With reference to Eqn. 4.17, the upper bound on RDP based on an ideal HPAM tree, is derived by setting node degree *d* to *3* and group size *N* to *100* to reflect the parameters of the simulation experiment. This yields an upper bound of 8.384 as per Eqn. 5.1. It can be seen from Figure 5.1 that all of HPAM clients have a RDP less than 5 which is way below this upper bound. Moreover, this observation also testifies to the efficiency of HPAM protocol in approximating the solution to the NP hard optimization problem of Eqn. 4.8. Another point to note is the ability of HPAM to exploit clients with low unicast latencies to the root. As shown in Figure 5.1, the three clients closest to the root have a RDP of 1 since the *JoinSource&Adopt (JSA)* algorithm effectively places them in level 1 of the tree thereby reducing the RDPs of all clients. As *JSA* is only applied to level 1 of the tree, this explains the lack of correlation between RDP and physical latencies for clients which are not sited in level 1 of the tree. Thus, the resulting HPAM tree still deviates from the ideal tree which should exhibit a perfect RDP mapping to the physical latencies. Such is the weakness of application level multicast techniques when they do not have direct route and link information of the physical networks. The fully centralized protocols will fare much better as they have full knowledge of the end-to-end status of all the group full members.

$$RDP \leq 2\log_3 100 \qquad \text{Eqn. 5.1}$$

$$RDP \leq 8.384$$

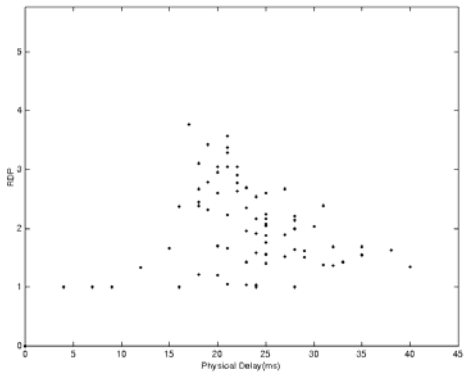


Figure 5.1: RDP vs unicast latency to source. Each point shows a client with a given RDP and unicast latency to source. Plot is captured at simulation time t = 105s after the last join at t = 100s.

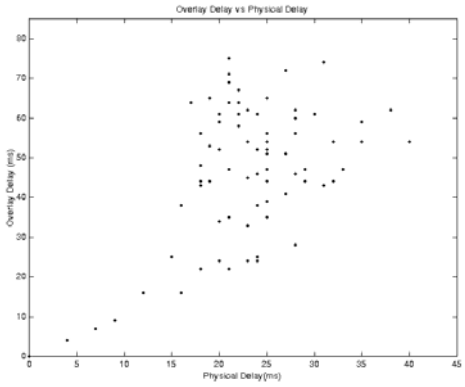


Figure 5.2: Overlay delay vs unicast latency to source. Each point shows a client with a given overlay delay and unicast latency to source. Plot is captured at simulation time t = 105s after the last join at t = 100s.

5.1.3.2 Adaptation to Group Dynamics and Tree Improvement Ability

Figure 5.3 plots the cumulative distribution of RDP at different simulation time snapshots. The vertical axis represents the percentage of group members for which the RDP is less than the value shown on the horizontal axis. The t=103s snapshot shows that all members have managed to join the HPAM tree 3s after the last join request before t=100s. Moreover, as time increases, plot t=105s shifts left and rises above plot t=103s (note that the group size remains at 100 for these two plots) before they merge at RDP = 2. This signifies the reduction in RDP as the quality of the HPAM tree improves through the gossip switch process. The same applies to plots t=455s, t=500s and t=600s which demonstrate the self-improvement capability of HPAM after the departure period <350s,450s>. The 3 plots shift progressively towards the left with time as the HPAM tree adapts itself and progressively improves on the RDP.

The t=455s snapshot shows the net group membership comprising 75 members with 25% having already left the group after the departure duration <350s,450s>. Plot t=500s shows the progressive refinement of the HPAM data tree as it adapts to the changes in membership dynamics. The RDP improves from 7 in t=455s to 5 in t=500s and then to 4 in t=600s. Plot t=600s shows the stabilized state. These 3 snapshots further illustrate the quick response time offered by HPAM protocols to changes in group membership as well as partition repair since 10 of the 25 departed members leave the tree through sudden failures.

Figure 5.4 shows the cumulative number of parent switches as a function of simulation time. This is related to the adaptation and self-improvement behaviour of HPAM as demonstrated in Figure 5.3. The time duration <350s,450s> and its aftermath see a progressively high concentration of switching activities which correspond to the period, <371s,505s> in Figure 5.3, where clients seek out new adoptive parents and repair any partitions arising out of the increasing number of departing parents. Moreover, after the tumultuous period of departure, clients will progressively have acquired up-to-date status of one another and the resulting tree refinement algorithm further contributes to increased switching activities as clients continue to discover better parents to switch to. The marked level of switch failures in this duration is mainly attributed to the potential parents which are leaving the group or have already failed and hence reject the adoption of the orphaned clients.

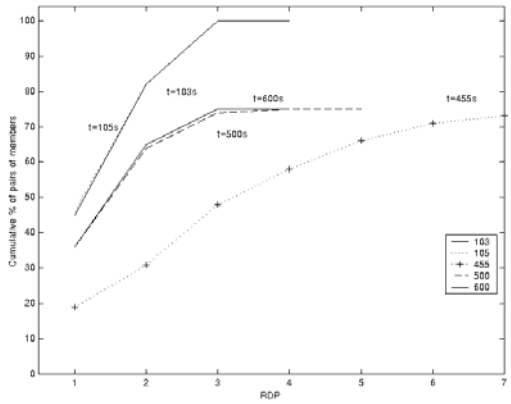


Figure 5.3: Cumulative distribution of RDP shown at various snapshots of simulation.

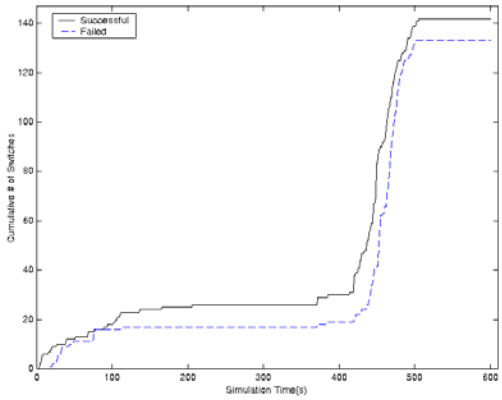


Figure 5.4: Cumulative number of parent switches.

The effects of spiral and gossip mechanisms on the adaptation of HPAM protocols will be analyzed and discussed in detail in Section 5.2.

5.1.3.3 Stress

The variation in link stress of HPAM is plotted against that of naïve unicast in Figure 5.5 at t=105s. The x and y axes represent the stress in log scale and the number of physical links with the given stress respectively. Both HPAM and unicast have most links with a small stress with unicast exhibiting a long tail with one link having a stress of 100 and quite a number of links with a stress greater than 16, the maximum stress experienced by HPAM. This is expected of naïve unicast as links near the source are likely to experience more stress. A point to note is that for HPAM, there is a noticeable kink at stress value of 4. This can be explained by the fact that the outdegree of each client has been set to 3 in the simulation. Taking into account the one incoming degree for each client, the total node degree of a fully-loaded client is 4. As such, the number of links with a stress of 4 is likely to be more typical than other

stress values which are greater than 1. These links are likely to be the last hop link to the clients.

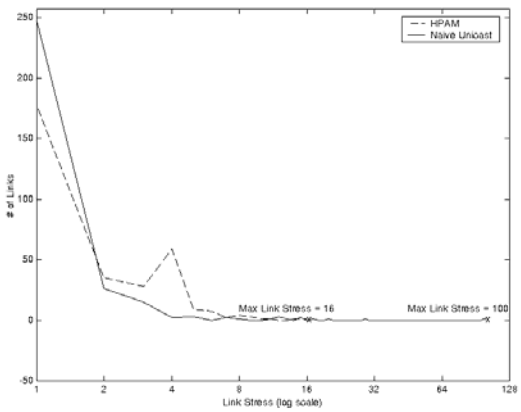


Figure 5.5: Number of physical links with a given stress vs stress for HPAM and naïve unicast.

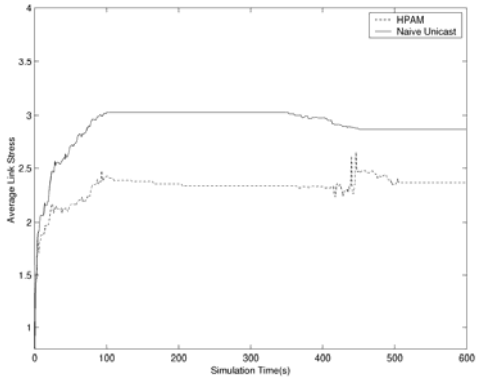


Figure 5.6: Average Link Stress vs time for HPAM and naïve unicast.

Figure 5.6 shows the average link stress vs simulation time. The average link stress is derived by dividing the total link stress collected over a recording duration of 1s by the number of links involved. Native unicast has an average link stress of about 1.3 times that of HPAM's. In fact, unicast's average link stress varies directly with group size as observed in the front and tail ends of the plot when the group size increases to 100 and drops to 75 respectively. HPAM, on the other hand, distributes the stress more evenly across the physical links and avoids loading the links near the source. HPAM's protocol of refining the tree through switching to better parents contribute to the drop in the average link stress seen after t=100s and after t=500s. The other fluctuations in the average links stress are due to the changes in membership during the initial join phase, the later departure phase as well as the tree refinement process. A point to note is the overhead imposed by the *JSA* algorithm on the link stress as observed at t=93s. There is a marked increase in link stress in this interval due to the

source simultaneously feeding a new client and a replaced level 1 client during the transient period when the new client has yet to adopt the replaced client as its child.

5.1.3.4 Protocol Overhead Ratio

Figure 5.7 plots the POR as a percentage of total data bandwidth. The protocol overhead is highest at the start of the HPAM session as the amount of data traffic relative to overhead traffic is high compared to the later time when more members join the tree. The mean control overhead measured in the stable duration <105,350s> stands at 0.27% of the total data bandwidth. Substituting the simulation parameters of Table 5.1 into Equation 4.18, the calculated stabilized mean POR is:

$$POR = \frac{(1+1+2+6*2/5)*64*8}{1.2*10^6}*100\%$$

$$= 0.273\ \%$$

This correlates well to the simulation mean of 0.27%. The bulk of these overheads are attributed to the refresh exchanges among members and with DS, while gossip exchanges constitute the little ripples seen in Figure 5.7.

Table 5.1: Simulation parameters for HPAM.

DSF: update frequency to DS	1 update per second
HbF: heartbeat frequency	1 heartbeat per second
RTTF: frequency of RTT test exchange	5 echo requests per 5 second
GF: frequency of gossip exchange messages per gossiper	2 gossip messages per 5 seconds
m: maximum number of gossipers in the gossip	6
ControlPacketSize:	64 byte
Data Source Rate:	1.2 Mbps

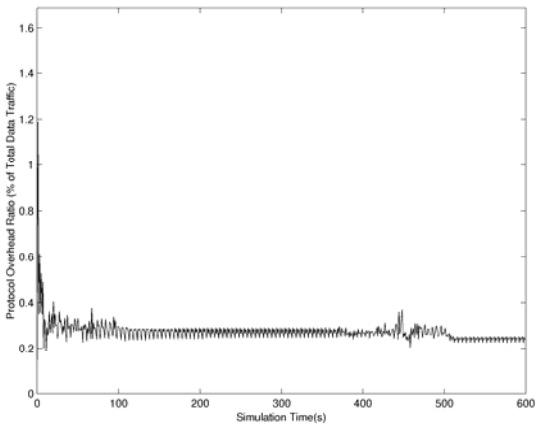


Figure 5.7: POR plotted as a % of total data traffic.

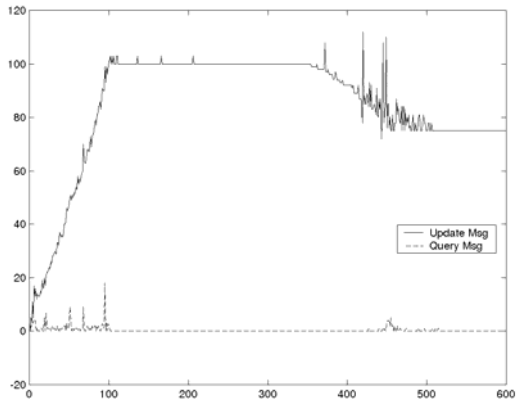


Figure 5.8: Message traffic to DS vs time.

5.1.3.5 Total Message Load on DS

Figure 5.8 plots the total message traffic to DS at 1s-intervals over simulation time. Each point shows the traffic accumulated over the past second up to its x-coordinate. As expected, the bulk of the message bandwidth to the DS, if not all, is consumed by the update messages sent by the clients with only a handful of query messages which appear during the periods of membership changes reflected during the initial join

period and around the departure interval <350s, 450s>. Otherwise, there is no query message to DS when the session is stabilized. During the join period, requests are made to DS by new clients seeking potential parents as well as clients requesting to fill their gossip caches at a time when group size is small. As the group size increases, the gossip cache contains more gossipers and the requests disappear. During the departure period, queries increase due mainly to clients requesting DS to replenish their depleting gossip caches. The upheaval in membership during the departure period writes off many gossip candidates due to non-availability of slots as well as departed or failed members. The subsequent detection and disposal of these stale gossip candidates deplete the gossip cache and hence replenishment is required.

As for the update messages, the steady state mean correlates with the number of members in the group as the refresh message heartbeat timer is set to 1s. To reduce the update overhead, the refresh period can be increased although there will certainly be some tradeoff in HPAM's responsiveness in adapting to the changing environment. The ripples seen in the vicinity of the duration <350s,450s> are attributed to the increased update activities as a result of departing clients and switching clients.

5.1.4 Impact of Group Size and Network Topology on HPAM

This section examines if the basic characteristics of HPAM, which are reflected in Section 5.1.3 still holds for different network topology models and different group sizes. The parameters being examined here are the RDP, the average stress and the TCR. The same simulation methodology as that described in Section 5.1.2 is adopted.

The results from both the Waxman topologies and the transit-stub topologies are presented.

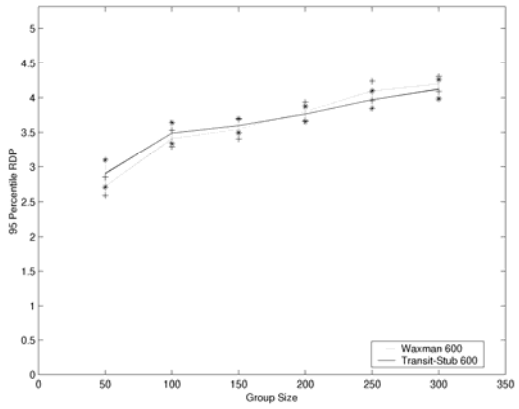


Figure 5.9: 95-percentile HPAM RDP vs group size for Waxman and transit-stub topologies.

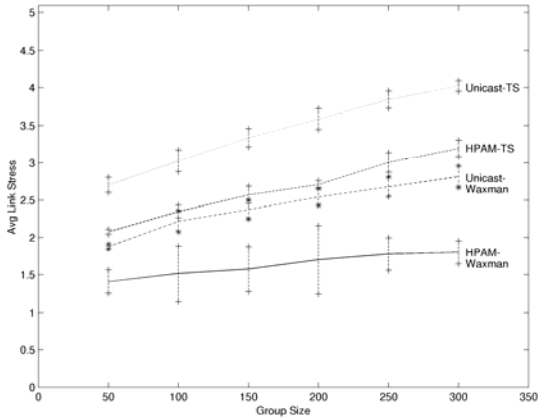


Figure 5.10: HPAM and unicast average physical link stress vs group size for Waxman and transit-stub topologies.

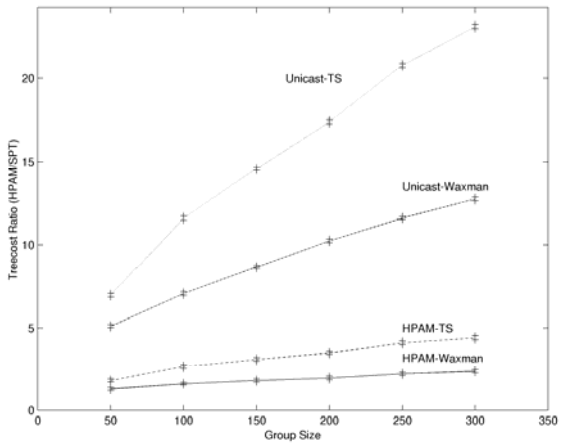


Figure 5.11: Treecost Ratio of HPAM vs group size for Waxman and transit-stub topologies.

5.1.4.1 RDP

Figure 5.9 plots the variation of the 95-percentile RDP with group size for two network topologies namely Waxman random topology and transit-stub topology. The two curves are close to each other therefore indicating that RDP of HPAM tree is insensitive to network topologies. RDP grows with group size as an increase in group size increases the HPAM tree depth resulting in a corresponding increase in RDP.

5.1.4.2 Average Stress

Figure 5.9 plots the variation of the average physical link stress with group size for the two different network topologies for both HPAM and unicast. The average stress increases with group sizes as the tree grows albeit in different directions. HPAM tree grows in depth while unicast tree grows in width. As expected, for both topologies, unicast stress is significantly higher than that of HPAM as physical links near the source are heavily loaded for unicast for all group sizes. This shows that HPAM is significantly more efficient in the use of network resources compared to unicast and this is consistent for all group sizes and all topologies. The larger average link stress in the transit-stub topology compared to that of the Waxman random topology can be explained by the fact that the former with its designated transit and stub nodes, provides relatively fewer direct node-to-node physical paths compared to the richer node-to-node connectivity of the Waxman random topology thereby resulting in a longer physical path between two nodes.

5.1.4.3 TCR

To further evaluate efficiency of HPAM in the use of network resources, HPAM's tree cost is compared to that of IP multicast (implemented as a shortest path tree) in the form of TCR in Figure 5.11. The TCR increases with group size as is consistent with the evaluation of RDP and average stress. With reference to Eqn. 4.7, for the Waxman random topology, HPAM tree ranges from 1.3 times that of IP multicast for a group size of 50 to 2.4 times for a group size of 300. For the transit-stub topology, HPAM tree's TCR ranges from 1.8 to 4.4 times. The transit-stub topology has a higher TCR than the Waxman random topology due to its less richly connected direct node-to-node paths. Nevertheless, HPAM's tree cost for both topologies compare very much more favourably against the unicast TCR of 5 to 13 for Waxman and 7 to 23 for transit-stub topologies. A point to note here is that HPAM tree is built based on the constraint that each node can only support an outbound degree of 3. The outbound degree therefore directly impacts the diameter of the tree. A larger outbound degree will result in a shallower HPAM tree which will in turn reduce HPAM's TCR further and brings it closer to the IP multicast's lower bound of 1 as per Eqn. 4.7.

5.2 Impact of Spiral and Gossip Mechanisms

To examine the contribution of the spiral and gossip mechanisms to HPAM's performance, a separate set of simulation is carried out on the transit-stub topologies. The same experimental setup for the five transit-stub topologies with varying group sizes and the same random members are used with one modification. The spiral and

gossip mechanisms are turned off just before the transition period <350s,450s> and they remain off thereafter. Without these mechanisms, when parents leave the HPAM tree or fail, the orphaned children will have to rejoin the tree via the DS. The purpose of such a setup is to maintain similar stabilized HPAM trees before the transition period. Only than can the impact of the spiral and gossip mechanisms on partition repair be examined in isolation and a fair comparison be made against the centralized partition recovery mechanism of approaching DS for a rejoin. The $C_{hysteresis}$ in Eqn. 4.12 is set at 10% for the simulation.

The efficiency of the partition recovery mechanism is measured as the Mean Recovery Time per affected client (*MRT*) and is defined in Eqn. 5.2. Detection Time *(DT)* captures the period of poor or no data reception before the affected client recognizes this as a possible partition. *DT* serves to measure the time scale at which the HPAM overlay has been designed to adapt to changing network conditions. For the simulation, the timeout for a partition detection is set to 1 s. In other words, if no data is received within the said timeout period, a partition is recognized. In the case when a client informs its children of its imminent departure, detection time is considered to be negligible as due notification has been served.

The effect of spiral and gossip mechanisms directly impacts the Repair Time (*RepT)*. *RepT* is the amount of time HPAM takes to connect to a new parent after a possible partition is detected. *RepT* therefore measures how fast HPAM responds once the decision to switch parent is made. *RepT* measures the time an orphaned client starts the process of switching until the time when a new parent is found. This may involve multiple attempts to switch parents. If the first attempt to switch to a gossiper

fails, the orphaned client will attempt to switch to the second gossiper and so on until the gossip list is exhausted. Its last recourse is to seek *DS* for assistance to perform a rejoin. Switching fails when the potential parent is full or when it is leaving the session or when it is dead. The last case will incur the highest delay since it will be only upon a timeout that the orphaned client realizes that the potential parent is not responding and that it has to proceed to switch to the next gossiper. For the simulation, the switch timeout is set at 2 s. As for centralized partition repair via *DS*, the orphaned client will contact *DS* directly for a rejoin. The main delay here is the RTT test, which for the simulation experiments, has been confined to 1 round trip test from the orphaned client to each of the potential parents performed simultaneously.

From Eqn. 4.18, the penalty of incorporating the spiral and gossip mechanisms in terms of *Control Overhead over Data Bandwidth (Gossip Overhead Ratio GOR)* is calculated to be around 0.124%.

$$MRT = \frac{\sum_{i \in M} [DT(n_i) + RepT(n_i)]}{\mathrm{M}} \qquad \text{Eqn. 5.2}$$

where *M is the total number of orphaned clients affected by the tree partition*
n_i is the orphaned client
$DT(n_i)$ is the duration from parent failure to detection of failure by client
$RepT(n_i)$ is the time taken to connect to a new parent after partition detection

Table 5.2 summarizes the 95-percentile and median values of *MRTs*, *DTs* and *RepTs* averaged over all the experiments for the spiral and gossip mechanisms versus those where only *DS* is used for partition recovery. To evaluate the impact of the spiral and gossip mechanisms versus the *DS*, only the *RepT* values should be

compared since the same partition detection mechanisms are used for both cases. This fact is reflected in the consistent *DT* values of Table 5.2 for both cases.

Table 5.2: Performance of spiral and gossip mechanisms versus rejoin via DS.

Recovery Mechanisms	Cause of Partitions	*MRT(s)*		*DT(s)*		*RepT(s)*	
		median	95-%tile	median	95-%tile	median	95-%tile
Spiral and Gossip Mechanisms	Node departure	0.089	2.213	-	-	0.089	2.213
	Node failure	1.308	2.924	1.135	2.060	0.067	2.050
Partition Recovery solely via DS	Node departure	0.173	3.370	-	-	0.173	3.370
	Node failure	1.348	2.050	1.121	2.059	0.189	1.635

Comparing the *RepT* values for node departure, it is observed that the spiral and gossip mechanisms take a much shorter time to repair the partition (1.94 times less for median value and 1.52 times less for 95-percentile value) compared to the case where *DS* is solely responsible for partition repair. The longer partition repair time is attributable to two main factors. Firstly, delay is incurred in performing RTT tests as partition repair is performed as per a join operation although this has been set to only 1 test per potential parent. Secondly, *DS* has a propensity to return a leaving client as a potential parent if the leaving client still has available slots. This is because, a leaving client who is not a leaf node is likely to have a better latency to root and hence will become a more probable potential parent candidate to joining clients. As such, more attempts are needed for an orphaned client to successfully locate a new parent as the leaving clients will reject its join request thereby increasing the *RepT*. This problem will become more acute for the *DS* case if there is a large number of clients leaving simultaneously.

Comparing the *RepT* for node failure, spiral and gossip mechanisms is again much faster than its *DS* counterpart in partition repair (2.69 times less for median value). The reasons are similar to the case of node departure explained above. However, for the 95-percentile value, the *DS* counterpart fares better being 0.8 times faster than the spiral and gossip mechanisms. This means that in the latter case, there are more partition repairs where the orphaned clients have attempted to switch to a dead parent and incurred the switching timeout delay compared to the *DS* case although the bulk of the switching does not involve a dead parent given the low median value of the spiral and gossip mechanisms. This phenomenon can be explained by the fact that clients are sending periodic updates to the *DS* at 1s-intervals while gossip exchange only takes place at 5s-intervals in the simulation to reduce control overheads. As such, *DS* maintains fresher client status than the gossip cache of a client and *DS* is therefore able to avoid returning dead nodes as potential parents to the requesting orphaned client. On the other hand, the gossip mechanism has a relatively more probable chance of having a dead gossiper in the gossip cache and this will cause switching timeouts to occur when the orphaned client tries to join a dead gossiper, thereby incurring a high switching delay. This phenomenon will occur when there is a large number of clients who die simultaneously.

In summary, the spiral and gossip mechanisms of HPAM provide much faster partition repair compared to the centralized approach. The slight disadvantage is the *POR* of 0.124% incurred as control overhead is incurred in updating the status of the gossipers. However, this control overhead is distributed among the many clients as opposed to all clients converging upon a centralized entity for partition repair.

5.3 Impact of JoinSource&Adopt (JSA) Algorithm

The *JoinSource&Adopt (JSA)* algorithm is introduced to overcome the problem of less latency-efficient clients hogging level 1 (the level closest to the root) slots of the HPAM tree because they join the tree earlier. *JSA* allows latecomers a chance to replace the less latency-efficient early birds. The performance metric which is affected most by *JSA* is the RDP. To illustrate the effects on RDP, another set of simulation is run as per those in Section 5.1.2 but without the incorporation of the *JSA*. The resulting 95-percentile RDP is presented in Figure 5.12 together with that of Figure 5.9 with *JSA* incorporated. The results clearly shown that applying *JSA* just at the level of the tree immediately following the root significantly improves the RDP of HPAM. This improvement is achieved at a slight increase of 0.01% in POR as HPAM-*JSA* registers a POR of 0.3% in the joining duration <0s,100s> while HPAM incurs a POR of 0.29% in the same time interval. This is the time interval where the *JSA* algorithm is active and the increase in POR is hence the direct result of the overheads incurred by the *JSA* algorithm.

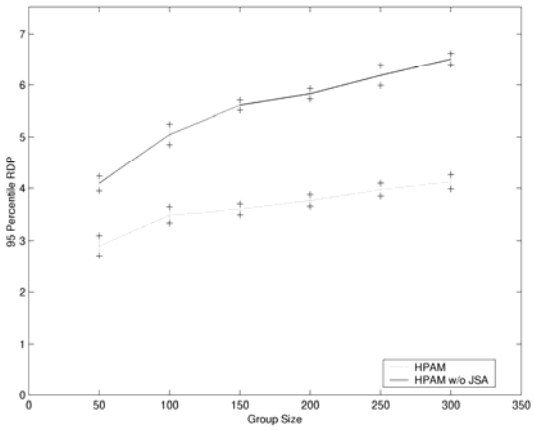


Figure 5.12: 95 percentile RDP vs group size for HPAM and HPAM without *JSA*.

5.4 Comparative Performance Evaluation

In this section, the performance of HPAM is compared against HBM [43], a centralized protocol and HM [39], a distributed protocol. These two protocols are chosen for comparative purposes as they are closest to HPAM in terms of data overlay topology and architecture. Similar to HPAM, both sport a flat, tree-based topology as well as a peer-to-peer (P2P) architecture which are unlike the hybrid protocols of Yoid [31], which is tree-mesh in topology or that of OMNI [41] which is proxy-based.

5.4.1 Simulation Setup

The same simulation setup as per that discussed in Section 5.1.1 is used with the implementation of HBM and HM according to the respective literature with the exception that the maximum outbound node degree for each member is now fixed at the HBM [43] value of 6, which is a middleground between the 3 used in HPAM above and the 8 used in HM [39]. HBM is implemented in its original version without the addition of virtual links called RVLs [46] to improve its robustness. The original version of HBM requires the central server, *RP*, to work out a sub-tree to link all the orphaned nodes together once the server is informed of a partition. RVLs reduce partitions by serving duplicate data to the clients via more than one link, thereby reducing the need to revert to the *RP* and hence improving the protocol response. The incorporation of RVLs does not impact the steady state values of RDP and treecost since HBM recomputes the tree topology periodically. However, RVLs addition will increase the HBM stress as RVLs carry duplicate copies of data. Overheads are also

incurred in the computation of the RVLs by the central server as well as clients who have to maintain additional connections to these RVLs. The original version of HBM is being compared here as it serves as a more stringent comparison for HPAM since it is the most ideal case of HBM in terms of stress and overheads.

5.4.2 Comparative Results

The comparison metrics used in this section are *RDP, average stress, TCR, POR* and the ability to adapt to tree dynamics. Figures 5.13, 5.14 and 5.15 respectively plot the RDP, average stress and TCR vs group size for HPAM, HBM and HM at the point when the protocols are at their steady state. *RDP* and *TCR* are obtained at t=350s while the average stress is obtained over the time period <105s,350s>. It can be seen that HBM enjoys the best performance in terms of *RDP*, average stress and *TCR* since it is a centralized system where the *RP* has complete knowledge of the group membership and their respective distance to one another. The *RP* is thus able to compute the optimum tree topology for data delivery and the resulting tree is close to the shortest path tree spanning all the members.

HPAM triumphs in terms of RDP to source as is evidenced in Figure 5.13 since it constructs its data tree with the assistance of *DS* to provide parents which are closest to the source. In terms of average stress (refer to Figure 5.14) and *TCR* (refer to Figure 5.15), HM has a slight edge though not significant, for smaller group size but this is eroded as the group size increases. The reason is that HM, being a distributed algorithm, does not have complete knowledge of all the member nodes. Its iterative algorithm which traverses the tree branch of a randomly selected node from its root

path in order to seek possible closer neighbours is unable to explore each and every members. As the membership increases, the limited group knowledge impedes its ability to exploit the full potential of the enlarged membership which reduces its efficiency. This is the same disadvantage faced by HPAM as it does not rely on its *DS* to centrally refine its tree but instead adopts the distributed, gossip approach. A point to note is that while HBM maintains relatively stationary *RDP*, *average stress* and *TCR* regardless of group size, HPAM and HM perform otherwise. These metrics for HPAM and HM increase with group size albeit at different rates. This is due to the fact that they do not recompute the entire tree to exploit the enlarged membership. Instead the tree is built on the fly with local refinements made using only a limited set of group members. Such refinements do not rebuild the entire tree and hence cannot efficiently map to the optimal shortest path spanning tree, unlike the case of HBM. Consequently, as group size increases, the TCR of HPAM and HM deviates more and more from the reference shortest path tree.

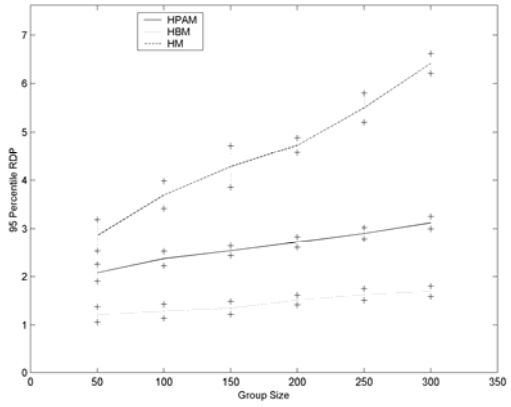


Figure 5.13: 95 percentile RDP vs group size for HPAM, HBM and HM.

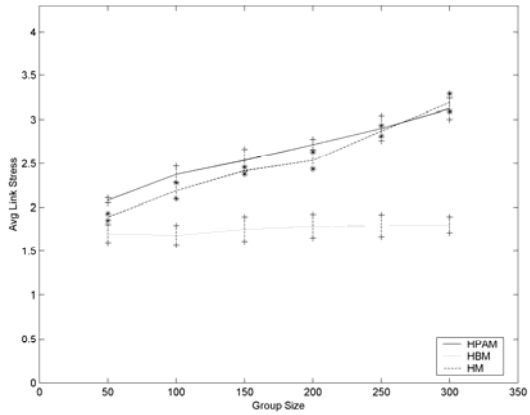


Figure 5.14: Average physical link stress vs group size for HPAM, HBM and HM.

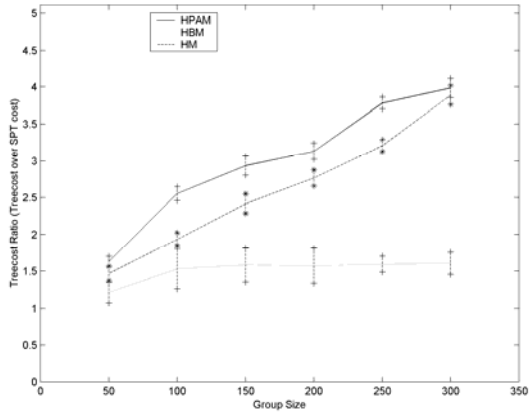


Figure 5.1.5: Treecost Ratio vs group size for HPAM, HBM and HM.

Table 5.3 shows the derived as well as the simulation average steady state POR for the three protocols. The POR derivation for HM and HBM are given in Appendices A and B respectively while that of HPAM value is obtained from Section 5.1.3.4. The simulation values are consistent with the derived values. HBM incurs the highest overhead as the centralized *RP* requires full knowledge of the client-to-client RTT in order to compute the optimal shortest path spanning tree. HM and HPAM are fairly close in terms of POR with HPAM incurring about 8% higher overhead than HM as the former strives to adopt a distributed tree refinement approach with the additional overhead of maintaining a lightweight centralized *DS*. Moreover, it should be noted that HM incurred a higher transient POR than HPAM during the client joining process. HM's iterative nature requires a long join latency for new clients. Hence, all new clients are randomly fostered out to existing parents until such time when they have located a permanent parent. The overhead incurred in the fostering process as well as the duplicate copies of data received by HM clients when they transitioned from foster parents to permanent parents have been ignored in the POR comparison as only the steady POR is being compared here.

Table 5.3: POR comparisons for HBM, HM and HPAM.

	HBM	HM	HPAM
Theoretical POR(%)	1.459	0.257	0.273
Experimental POR(%)	1.47	0.25	0.27

To provide an insight into the adaptivity of the various protocols to tree dynamics, the *MRT* values as defined in Eqn. 5.1 are compared. For HM, a partition is detected when there is an absence of root path messages from the parent [39]. In the simulation, the root path messages are sent by the parent every second. In the case of HBM [43], a partition is detected when there is an absence of data from the parents. The partition detection timeout for all three protocols are set at 1.5 s. Table 5.4 shows the *MRTs* for all three protocols. HPAM clearly triumphs over the other two protocols as it is able to recover from partition much faster due to a combination of speedy partition repair which is effected by the spiral and gossip mechanisms as well as a more sophisticated partition detection mechanism which will be covered in details in Section 6.1. HM is the slowest in terms of recovery from partition as it employs an iterative algorithm to find a parent which is closest to it. The process involves choosing a random node from the root path of the orphaned node and performing RTT tests until it finds the closest candidate in that branch of the tree. The RTT test is confined to one test per potential parent in the simulation. As for HBM, when a partition is detected, it will approach the centralized controller, *RP*, for a parent to be temporarily grafted until such time when the *RP* has re-computed the new optimized tree and sends it the most updated tree version. The advantage here is that the orphaned client does not perform any RTT test which accelerates the partition

recovery process albeit at the expense of having a less than efficient parent for the time being. A point to note is that the 95-percentile *MRT* for node failure of HBM is better than that of HPAM. As explained in Section 5.2, the RP of HBM is better able to detect the dead clients due to the more frequency updates to the RP compared to the gossipers in HPAM whose update frequency is one-fifth that of the RP update frequency. Hence chances of a HBM orphan encountering a dead foster parent is very much lower than a HPAM orphan trying to switch to a dead parent. The latter will thereby encounter a timeout which slows down the partition recovery process. For node departure, HPAM orphan does not encounter a timeout problem since the leaving gossiper will reject its join request straightaway and it can proceed to join the next gossiper without having to wait for a timeout.

Table 5.4: *MRT* comparison of HBM, HM, HPAM.

	MRT(s)					
Cause of Partitions	*HBM*		*HM*		*HPAM*	
	median	95-%tile	median	95-%tile	median	95-%tile
Node departure	0.102	2.575	8.814	20.475	0.089	2.213
Node failure	1.567	1.558	5.325	16.062	1.308	2.924

In summary, HPAM is efficient for reducing latency to source while HM is more effective in minimizing the overall tree cost which can translate to minimizing use of network resources but it does not strive to reduce the latency to the individual nodes. The hybrid HPAM is compatible to the distributed HM in terms of average stress, treecost and POR performance and delivers better latency to source for the clients than HM. In terms of partition recovery efficiency and adaptivity to dynamic

environment, HPAM is superior as HM's iterative algorithm involves a long delay to locate an efficient parent. HBM is optimal in all cases but the overheads incurred to maintain full knowledge of the membership renders it unsuitable for medium to large group deployment. HBM has explored a number of strategies in [45] to reduce the overhead by limiting the need to know the full membership metric. The different strategies serve in varying degrees to reduce its overhead to the order of *O(N)* but this is achieved at the expense of its average stress and TCR as the RP is no longer able to compute the optimal shortest path spanning tree without full knowledge of membership metric.

5.5 DISCUSSION

HPAM's very strength lies in the simplicity of its protocol. Extensive simulation results have shown that the HPAM data delivery tree is constructed at a tree cost of 1.5 to 4.0 times that of IP multicast and a RDP of between 2.0 to 3.2 times that of the unicast latency (shortest path) for group sizes ranging from 50 to 300. The efficiency of the HPAM tree is verified by the fact that more than 95 percent of the clients enjoy a RDP less than the upper bound RDP of an optimal HPAM data delivery tree as illustrated in Figure 4.9. All these are achieved at a protocol overhead of a mere 0.27% of the total data bandwidth. The DS load is kept relatively low with a message bandwidth in the order of *O(N)* where *N* is the number of group members. State overhead maintained at each client is independent of the group size and is a constant at *O(k)* where *k* is the sum of the outdegree of a client and the number of gossipers in the gossip cache.

Simulation also shows that the HPAM protocol is responsive to group dynamics and is able to improve the quality of the data delivery tree over time as well as to provide speedy repair of tree partitions.

There are shortcomings in HPAM. The primary weakness is the DS which is a single point of failure. A DS failure will prevent new members from joining existing HPAM sessions although it will not impact existing members. Existing members can continue to receive their data streams. Another concern pertaining to DS is its scaleability as it maintains the state of all HPAM members although it does not maintain detailed client-to-client connectivity status information unlike the case of HBM. As HPAM group size increases, the load on DS will increase correspondingly. A solution to this problem is to use a cluster of machines to host the DS instead of a single machine so as to distribute the load as well as to provide a hot standby in the event of a failure.

HPAM builds the data delivery tree on the fly as members join the group in consultation with DS. The parent discovery algorithm is basically to choose a parent which yields the closest proximity to the root. As such, the early members are the ones who will populate the lower levels of the tree compared to a late member, who may be physically closer to the root but are grafted to slots which are further away from the root of the HPAM tree as the lower slots are already full. This results in HPAM tree being less than optimal compared to the ideal HPAM tree where the nodes are grafted in ascending order of latency from the root as in Figure 4.9. However, it should be noted that such an ideal tree is only physically feasible if the

tree construction algorithm is fully centralized and all the members are known. Such an algorithm as epitomized in ALMI and HBM creates a minimum spanning tree rooted at a source and entails a re-computation when members join and leave the tree. Weighing the algorithmic complexity, HPAM decides on a less than ideal but more simplistic approach in ameliorating this 'earlybird inefficiency' problem by incorporating the *JSA* algorithm. *JSA* allows late members who are close to the root to replace existing less latency-efficient members sited immediately after the root. These replaced members are then adopted by the newcomer. Further refinement of the tree is then accomplished via the gossip mechanism.

5.6 Summary

Extensive simulation and analysis are performed to evaluate HPAM's performance in terms of RDP, stress, TCR, POR, message load on DS as well as HPAM's ability to adapt to group dynamics, to improve the tree quality and to repair and recover from partition. The main contribution of HPAM is that it can build and maintain application layer multicast trees with very little overheads and low performance tradeoffs as verified in the simulation results. The simplicity of HPAM's protocol with its self-organizing, self-improving and self-repairing characteristics enables HPAM to quickly construct efficient data distribution trees on top of a dynamic, unpredictable and heterogeneous Internet environment without relying on native multicast support. The hybrid nature of HPAM renders its performance to be comparable to distributed protocols which are tree-based such as HM in most aspects despite the simplicity of its protocols. In fact, it is better than HM in terms of RDP and its lightweight,

centralized DS with distributed tree refinement and partition recovery allows a faster join for new clients and swifter partition recovery compared to HM's iterative protocol. As such, the latter requires new clients to be joined to foster parents first until a permanent parent is located. Although HPAM is unable to achieve the level of optimality in terms of RDP and treecost compared to centralized system as personified by HBM, it is more responsive to changes in membership dynamics and more importantly, its performance is achieved at a very much lower overhead.

Chapter 6

Enhancements to HPAM Protocol

Chapter 6 focuses on enhancements made to the HPAM protocol. HPAM-D (HPAM-Dual Metric) constructs data distribution trees based on two performance metrics, namely, latency to source and loss rate experienced by the clients. The protocol is evaluated against the latency based single-metric HPAM presented in Chapter 4 and a modified HPAM-L which aims only to optimize loss rate regardless of the latency to source. HPAM-L acts as a control experiment in the evaluation. Another novel feature introduced in HPAM-D is the detection of likely local congestion via a heuristic based on the computation of a client's relative loss rate with respect to the data received by its parent as opposed to the commonly used absolute loss rate experienced by a client.

6.1 HPAM-D Protocol

As introduced in Section 4.1, HPAM-D is basically derived from HPAM by introducing a second parameter i.e. the loss rate experienced by a client in addition to the latency from the root in the construction, refining and maintenance of the data delivery tree. Unlike the improved version of Narada which uses latency and bandwidth as performance metrics, loss rate is used instead of bandwidth as loss rate can be implicitly estimated while data is being received by each member while the bandwidth requires intrusive active end-to-end measurements. This involves transferring data for a period of time using the underlying transport protocol at a rate bounded by the source rate between members, thereby incurring high overhead and latency. The loss rate is defined in Eqn. 6.1 and this is also known as the absolute loss rate (*ALR*) experienced by the client. It is to be noted that unless otherwise specified, the term *loss rate* shall refer to the *ALR* in this dissertation.

$$Loss\ Rate\ (ALR) = \frac{Number\ of\ lost\ data\ packets\ per\ second}{Total\ number\ of\ data\ packets\ sent\ by\ source\ per\ second} \qquad \text{Eqn. 6.1}$$

HPAM-D tree aims to provide an efficient compromised solution to satisfy a low latency root as defined in Equation 4.8 and an acceptable loss rate as defined in Equation 4.9. Since HPAM-D is derived from HPAM, it fully inherits the properties and characteristic features of HPAM as presented in Chapters 4 and 5. This section will therefore focus on the effects of the dual metrics on the performance of HPAM-D relative to HPAM and the modified HPAM-L which is based solely on loss rates. HPAM-L acts as a control experiment for HPAM and HPAM-D.

For HPAM-D, latency to root is prioritized over loss rate. The premise for this decision is to deliver the live streams to clients as near real-time as possible but at the same time, to provide a form of best-effort minimization of loss in the delivery stream. Nevertheless, the heuristic can be easily adapted to prioritize loss rate over latency if the application requires better perceptual quality with a tradeoff in the latency of data delivery. Among the application level multicast techniques published, HPAM-D is among the first to incorporate loss rate with RTT as dual performance metrics to optimize the data tree and to investigate the impacts of using these two metrics together.

6.2 Loss Rate as the Second Performance Metric

HPAM-D incorporates an additional check for the loss rate to ensure that it is less than the loss threshold, L_{th} as defined in Eqn. 4.9 in the following operations:

- *Section 4.3.1.1 Bootstraps Source Discovery for New Client*: When DS responds to requests for a list of potential parents as per the heuristic set out in Figure 4.2, it will also check that the absolute loss rates of all the potential parents satisfy Eqn. 4.9. Hence, the potential parent is one which is closest in latency to the root as well as having a very low if not nil loss rate in HPAM-D.
- *Section 4.3.2.1 Normal Join Algorithm:* Potential parent will reject a join request if it cannot satisfy Eqn. 4.9 in HPAM-D.
- *Section 4.3.3.2 Tree Quality Improvement:* A HPAM client uses the *Proactive Tree Refinement (PTR)* mechanism to periodically compute the quantum of *QoSGain* (defined in Eqn. 4.13) it can derive by switching its parent to a gossiper. As the

QoSGain is based on the latency to root, *PTR* therefore improves the latency to root metric of the data tree. In HPAM-D, the *PTR* mechanism has been enhanced to include the check that the gossiper's absolute loss rate is less than L_{th}. In other words, both Eqn. 4.13 and Eqn. 4.9 must be satisfied during *PTR* in HPAM-D as shown in Eqn. 6.2. The loss rate of the gossiper is updated as part of the gossip message exchanges.

$$Latency_{i,client} + Latency_{i,r} \leq Latency_{client,r} \textbf{ AND } Loss(i) \leq L_{th} \quad \text{Eqn. 6.2}$$

where $i \in$ gossip cache i

In HPAM-D, tree quality improvement seeks to improve the latency as well as to reduce the loss rate and it is effected via both the enhanced *PTR* as well as via the loss check. The latter mechanism is termed as *Reactive Tree Refinement (RTR)* since the tree refinement is triggered in response to an unsatisfactory loss rate experienced by the client over a period of time. A client will seek to reduce its loss rate through parent switching if it experiences a loss rate that is higher than L_{th} for a period of time but it is not a total loss which is a partition recovery problem already addressed in Section 4.3.4.1. The decision on whether the client concerned will start the *RTR* will be based on the heuristic described in Section 6.3. In this section, the focus is on how to seek a suitable candidate to switch to, i.e., the client has already decided to do a parent switch and invoked the *RTR.* The algorithm to select a suitable gossiper to switch to in *RTR* is therefore more complex than merely checking if Eqn. 6.2 is satisfied as per the enhanced *PTR* since the need to improve the unsatisfactory loss rate is more pressing that the *PTR's* proactive approach to improve a client who is currently enjoying an acceptable loss rate. The heuristic algorithm is set out in Figure 6.1 and it is designed to prioritize latency over loss rate. Basically, the

gossiper which satisfies Eqn. 6.2 will be selected as the potential parent. A gossiper with neither better latency nor satisfactory loss rate as depicted in Eqn. 6.3 will be removed from the gossip cache as this gossiper is deemed to be a poor potential parent. Should there be no gossipers who can satisfy Eqn. 6.2, the client will revert to *DS* to look for a new list of potential parents and switch to a new parent as per the *Normal Join Algorithm* detailed in Section 4.3.2.1. In this way, the gossip cache is also refreshed.

$$Latency_{i,client} + Latency_{i,r} > Latency_{client,r} \textbf{ AND } Loss(i) > L_{th} \qquad \text{Eqn. 6.3}$$

$\forall i \in$ gossip cache
{

Case 1: Eqn. 6.2 is satisfied
// The gossiper, i, will yield equal or better latency and will meet the satisfactory loss rate if a
// parent switch were to be made.

Proceed to switch parent
exit.

Case 2: Eqn. 6.3 is satisfed
// The gossiper, i, will yield worse latency and will not meet the satisfactory loss rate if a parent
// switch were to be made. The poor gossiper shall be removed from the gossip cache.

Remove this gossiper from the gossip cache
Proceed to test the next gossiper
Others:
// The gossiper, i, can either satisfy the latency or the loss requirements but not both.

Proceed to test the next gossiper.
}

// Approach DS to 'rejoin' the session as per a new client when there is no suitable gossiper
if *no suitable gossiper is found*
JoinSessionRequest *// Issue join request to DS*
NormalJoin (client, newParentList) *// Perform normal join as per Figure 4.3*
end if

Figure 6.1: Heuristic algorithm for the HPAM-D dual-metric *Reactive Tree Refinement.*

- *Section 4.3.4.2 Partition Repair:* An orphaned client will not switch to a gossiper with unsatisfactory loss rate in the event of a partition.

6.2.1 Simulation Methodology

The performance of HPAM-D is evaluated against HPAM and HPAM-L via simulation. Unlike the simulations conducted for HPAM as described in Chapter 5, the experiments conducted here include the simulation of queuing delays and buffer overflows in a random number of routers. Simulation of queueing delays will lead to buffer overflow in the congested physical networks, resulting in loss of data packets. This will better emulate the real-life Internet environment and more importantly to allow the showcase of HPAM-D's performance in a dynamic environment where losses do occur now and then even when there are no client failures. The maximum number of children per client is set to 3. The rest of the simulation setup is similar to that in Section 5.4.1 Simulation Setup. L_{th} is set to 5% in the simulation.

6.2.2 Performance Metric used in Evaluation

The comparative metric used here is the *average loss rate* which measures the mean loss rate experienced by each of the client per second. The *average loss rate* presents itself as a suitable candidate to compare performances here since loss rate check is incorporated as a second performance metric in HPAM-D and thus the loss rate experienced by a client is

reflective of the ability of the protocol to build, refine and maintain the quality of its data distribution tree in a lossy environment.

To investigate the effect of incorporating a loss metric in the HPAM protocol, the non-transient average loss rates of the individual clients are examined. By non-transient, it means that the periods <0s, 100s> and <350s, 450s> which correspond to new clients joins and existing clients' departures/deaths respectively, are excluded. These tumultuous periods involved the interplay of many different mechanisms and the resulting loss rates experienced by the clients cannot be directly attributed to the dual metrics used in HPAM-D or the absence of loss metric in HPAM. The average loss rates for each client are recorded in 1-second intervals. The number of records which satisfy the loss threshold, L_{th}, is determined from the total number of records which have registered a non-100% loss over the non-transient simulation periods and it is expressed as a percentage of the latter. The ability of the various protocols to maintain a satisfactory average loss rate for each client over time during the non-transient period will provide great insight into their respective performance. Only the non-100% loss records are considered here as recovery from 100% loss rate is addressed under partition repair in Section 4.3.4.2.

The other evaluation metric used is the 95-percentile RDP as it is interesting to investigate whether the incorporation of the loss metric does compromise the performance of the RDP metric. However, taking the mean values of the 95-percentile RDP over all the experiments in a dynamic network environment does not yield meaningful results as the physical network is now dynamic and different combinations of membership will result in different levels of congestion and different degrees of tree refinement. All these result in highly varying *RDP* values even when the random selection of members for the same group

size is run over the same network topologies. With different group sizes and member compositions running over the different network topologies, the variations are even more pronounced. To derive meaningful results, the RDP is compared on an experiment by experiment basis for the three protocols. Using HPAM as the reference, the ratio of 95-percentile RDP values of HPAM-D and HPAM-L experiments to that of the HPAM experiment is captured for a fixed group size and member composition. These ratios are then averaged over all the different experiments performed with different group sizes and member compositions to observe the characteristics of the protocols.

6.2.3 Comparative Results and Evaluation

Table 6.1 summarizes the mean (at 95% confidence intervals) of the 95-percentile RDP ratio. It is observed that HPAM-D and HPAM are said to be on par where *RDP* performance is concerned. Although the introduction of loss check in HPAM-D may sub-optimize the latency to root at certain times in order to maintain a satisfactory loss rate, the former metric is still given priority over the latter. On the average, HPAM-D's performance in *RDP* will therefore converge towards HPAM. The poor *RDP* value for HPAM-L is intuitively obvious since the HPAM-L protocol does not choose parents with the lowest root latency in the tree construction. Instead, the tree is constructed by choosing parents with the lowest loss rates. Where loss rates are similar, the potential parent who is closer to the joining candidate will be selected.

Table 6.2 summarizes the percentage of acceptable average loss rates recorded during the non-transient simulation periods over the total number of non-100% loss recorded. As

expected, HPAM-L maintains the highest percentage of acceptable loss rates as it is designed to optimize the average loss rate in its tree construction, refinement and maintenance. HPAM fares relatively worst albeit at a decent 74.5% acceptable average loss rate since it is optimized for latency to root metric while HPAM-D with its dual metrics is able to maintain a better average loss rate for its clients compared to the single-metric HPAM. In view of the 95-percentile *RDP* values presented in Table 6.1, HPAM-D is able to achieve a high percentage of acceptable average loss rates in its clients without compromising its RDP performance as the latter is comparable to HPAM.

Table 6.1: 95-percentile *RDP* ratios for HPAM-D, HPAM and HPAM-L.

	HPAM-D	HPAM	HPAM-L
Mean 95-%tile RDP Ratio	0.98 x HPAM	Reference	1.89 x HPAM

Table 6.2: Percentage of acceptable loss rates maintained by HPAM-D, HPAM and HPAM-L.

	HPAM-D	HPAM	HPAM-L
% Acceptable Average Loss Rate	87.5%	74.5%	95.9%

6.3 RLR based Heuristics for Local Congestion Detection

As HPAM-D uses both latency to root and loss rate in the construction, refinement and maintenance of its data tree, loss rate information is maintained by all the clients as well as *DS*. To further exploit the loss metric, HPAM-D has incorporated another variant of the loss rate into its protocol, namely, the relative loss rate (*RLR*) experienced by a client. *RLR* is defined in Eqn. 6.4 and it refers to the ratio of lost packets to the total number of data packets sent by a client's parent in a second. *RLR* is derived from the loss rates of the client as well as its parent. HPAM-D makes use of a client's *RLR* in conjunction with its own loss rate and that experienced by its parent to detect possible local congestion in the physical network linking itself and its parent on the HPAM-D's data tree which may therefore contribute to the data loss in the client.

$$RLR_{client} = \frac{LossRate_{client} - LossRate_{parent}}{1 - LossRate_{parent}} \qquad \text{Eqn. 6.4}$$

Local congestion can occur when a new node joins the overlay tree or a client switches parent since the HPAM-D does not have full knowledge of the underlying physical network. This may result in more data packets traversing certain routers and links than these infrastructures can manage. It therefore results in congestion and hence increased latency and loss rate. The ability to provide possible detection of local congestion will enhance the performance of HPAM-D in terms reducing latency and loss rate to the affected client as well as reducing the protocol overheads. The former is accomplished by clients switching to

alternative parents to ease the congestion while the latter is facilitated through the reduction in the number of unnecessary parent switches.

Table 6.3 illustrates the heuristics adopted by HPAM-D to detect for possible local congestion. The various possible loss rate scenarios experienced by a client are shown with the respective actions to be taken by HPAM-D after checking the client's *RLR* and both *loss rates.* In cases 2, 3 and 4, using a client's loss rate alone without the benefit of the *RLR* and the parent's loss rate, will not allow it to detect if there is a likely local congestion. In such cases, the client will simply invoke the *Reactive Tree Refinement (RTR)* to seek a suitable candidate for a parent switch. However, the check for parent's loss rate in conjunction with the *RLR* allows an inference to be made on the possibility of local congestion for cases 2, 3 and 4. This will delay switching in case 4 while it waits for the parent to activate *RTR* and improve the data delivery. The threshold set for *RLR* is the same as the satisfactory loss rate threshold used. In other words, if the *RLR* is greater than L_{th}, the local congestion is unacceptable and a parent switch is recommended.

Table 6.3: Heuristic algorithms for detection of likely local congestion for the different loss rate scenarios experienced by a HPAM-D client.

	Parent	Client		Action
	LossRate	*LossRate*	*RLR*	
Case 1	$\leq L_{th}$	$\leq L_{th}$	x*	None
Case 2	$\leq L_{th}$	$> L_{th}$	x	Confirmed local congestion. Switch parent.
Case 3	$> L_{th}$	$> L_{th}$	$> L_{th}$	Likely local congestion, switch parent.
Case 4	$> L_{th}$	$> L_{th}$	$\leq L_{th}$	Local congestion likely to be minimal. Wait for parent to switch first. If parent fails in switching, then switch parent.

*x refers to don't-cares

6.3.1 Impact of RLR based Heuristics on HPAM-D

To study the impact of the *RLR* based heuristics on HPAM-D, the heuristics is removed from HPAM-D. The modified HPAM-D will hence just test a client's loss rate and if the loss rate is higher than L_{th}, *RTR* kicks in and parent switch will take place. The modified HPAM-D is run under the same simulation methodology as HPAM-D in Section 6.2.1.

The performance metrics used in this evaluation are the average loss rate and the average number of parent switches made. This latter metric provides an insight how much the *RLR* based heuristics can reduce the number of unnecessary parent switches where loss is largely attributable to the parent and not to the local congestion. To prove that the reduction in the unnecessary parent switches does not compromise the quality of the HPAM-D overlay tree, besides the average loss rate, the mean 95-percentile RDP ratio obtained on an experiment by experiment basis as per Section 6.2.2 is also captured.

The simulation results are presented in Table 6.4. The number of switches is categorized into successful switches, failed switches and their sum. The switch data are captured under two simulation durations, namely, the steady state period and the dynamic period in which client departures and deaths are registered so as to evaluate the heuristics' performance under both steady and dynamic conditions.

Table 6.4: Performance comparison between HPAM-D and HPAM-D without RLR based heuristic.

	HPAM-D	HPAM-D without *RLR* based heuristics
% Acceptable Average Loss Rate	87.5%	76.7%
No. of Switches in Steady State Successful Failed Total	Reference	 2.52 x HPAM-D 1.29 x HPAM-D 1.71 x HPAM-D
No. of Switches in Dynamic State* Successful Failed Total	Reference	 1.40 x HPAM-D 1.12 x HPAM-D 1.21 x HPAM-D
Mean 95%-tile RDP Ratio	Reference	1.07 x HPAM-D

* Dynamic state refers to the simulation time period <350s,450s>. It excludes the initial join period <0s, 100s>.

From Table 6.4, it can be observed that RDP performance is not compromised as the mean 95-percentile RDP for both cases are very close. As for average loss rates, HPAM-D manages to maintain a lower loss rate compared to the modified HPAM-D without the *RLR* based heuristics. As for the number of parent switches, HPAM-D registers a lower switch count in all categories compared to its non-*RLR*-heuristic counterpart. A lower loss rate achieved at a reduction in switch count with no compromise in RDP performance in the HPAM-D clearly vindicates the effectiveness of the *RLR*-based heuristics. By differentiating between loss caused by local congestion and loss attributable to the current parent, clients can stick to the current parent which has good underlying network links to itself, thereby avoiding a potential bad switch who may land itself in a local congestion instead as well as reducing some unnecessary parent switches.

6.4 Summary

The incorporation of loss rate as a second performance metric after latency to root has enabled HPAM-D to not only maintain the level of optimality exhibited by HPAM in terms of RDP but it is also able to maintain a high percentage of acceptable loss rate in a dynamic network environment where there are congestion and losses as vindicated by the extensive simulation results. All these are achieved with negligible protocol overhead over HPAM as the loss rate experienced by a client can be gleaned from the data packets received. With the loss rate data maintained by the clients, HPAM-D is able to further exploit this data to derive the relative loss rate. Based on the *RLR*, HPAM-D is able to incorporate heuristics for the detection of possible local congestion that exists between itself and its parent. The heuristics not only helps a client to reduce unnecessary parent switches thereby reducing protocol overheads but also contributes to the ability of the HPAM-D to maintain a low loss rate for its clients without compromising the RDP performance.

Chapter 7

Impact & Detection of Cheats in HPAM Protocol

This chapter provides an overview of the incentives for clients to cheat as well as the cheating strategies used when operating under the HPAM protocol. The focus is on the impact on HPAM's performance when clients cheat in the RTT measurements during the tree construction and refinement process. Techniques to detect such subtle cheating are incorporated into the HPAM protocol as further enhancements and their efficacies are investigated.

7.1 Incentives for Cheating in HPAM

Application level multicast protocols strive on the fundamental spirit of honest collaboration and sharing among the different clients. Each client forms a node in the data distribution process with intermediate clients participating in both the data reception as well

as data distribution while leaf nodes do not carry the latter 'burden'. The protocols rely on the clients to take their own independent performance metric measurements, make their own independent decisions based on these measurements and advertise them to other clients. Since these protocols operate at the application level, the clients are thus 'empowered' to intercept and modify distributed contents as well as to transmit erroneous responses to control queries and requests. Hence, honesty from all clients has to form the bedrock of any successful deployment of application level multicast protocol and it is in fact the underlying assumption of application level multicast protocols.

However, clients do have the incentives to cheat. An intermediate client will be better off receiving data instead of carrying the additional burden of transmitting the received data downstream. Moreover, a client will enjoy lower data latency and less loss if it is positioned close to the root of the data distribution tree since latency and loss accumulate as one moves further away from the root of the tree.

7.2 Cheating in HPAM

With respect to HPAM (both HPAM and HPAM-D shall be collectively known as HPAM hereinafter unless otherwise stated) there are many ways a client can possibly cheat since it can manipulate both the data and control messages. Only independent cheating is examined here. This means the cheats operate independently from one another for their own selfish gains and are not malicious. They do not disrupt data flow nor cause denial-of-service to other members of the group but instead work towards their own benefit at the expense of others. The case of cooperating cheats with sophisticated and systematic cheating strategies

poses another level of research challenge altogether [123] and is not considered here. The simplest way for a client to avoid becoming an intermediate node and relay data to others is to declare that it can take zero child. Other flagrant cheating to avoid supporting children includes refusing to accept all join requests and reporting high loss rates and/or high RTTs so that DS and other clients will not select it as a potential parent. These cheating techniques can easily be detected by *DS*.

A more subtle means to cheat is to manipulate RTT test results whenever RTT measurements are requested from them. A cheat can purposely delay its response to RTT tests from other clients so as to inflate its RTT. The cheat will however, report the actual RTTs to *DS* as per any other clients so as to avoid arousing suspicions. Such cheats will continue to be selected by *DS* as potential parent candidates to other clients but chances of them becoming parents will be very slim due to the inflation of their RTTs during RTT tests.

To achieve proximity to the root in HPAM, a cheat can exploit the *JSA* algorithm described in Section 5.3. As *JSA* allows a new client with better latency-to-root than existing level 1 clients (level 1 is the level closest to the root of the tree) to join the source directly and adopt the replaced level 1 client, there exists a potential loophole for cheats to exploit. Cheats can fabricate their RTTs and request the root to accept them as children even though their RTTs to the root may be inferior to all the current level 1 clients. As long as they remain next to the root, they will update the understated RTT values to *DS* so as to maintain their positions and not be replaced by new honest clients or other cheats. Once they are displaced further from the root, they will start reporting the actual RTTs to DS as per normal. Reporting high RTTs to *DS* to discourage others from choosing cheats as

parents is actually detrimental to them maintaining their proximities to the root. This is because the high RTTs will actually push them further and further away from the root with each new client (both honest clients and cheats) joining the session since the newcomers enjoy a lower RTT than the inflated values stored in *DS*.

7.3 Detection of Obvious Cheating by DS

As HPAM is a hybrid protocol, the presence of the centralized lightweight server, *DS*, facilitates the detection of such cheating. For those cheats who have declared that they are unable to support children and are occupying slots near the root of the tree, *DS* can inform clients who are furthest from the root to adopt the cheats such that they are being relegated to the furthest leaf nodes possible. Their vacated 'prime' slots can then be opened to new and existing clients to render the data distribution tree more efficient. Likewise, the unusually high RTTs can be detected by comparing them with validated historical values of mean RTTs collected. These high-RTT cheats occupying slots near the root and with zero or low child count compared to long, children-laden branches of other clients are evidence of cheating and such cheats should be similarly banished to the faraway leaf nodes. Cheats who repeatedly refuse the join requests of clients will similarly be detected as their good latencies to root stand to contradict their lack of children. This is in stark contrast to their counterparts which possess relatively poorer latencies to root but are laden with children.

For cheats who report high loss rates to *DS* in HPAM-D, *DS* can check that despite the high loss rates reported, the cheat's parent is enjoying satisfactory loss rates. Such scenario will result in an honest client doing a parent switch as this is a case of local congestion as

per Case 2 of Table 6.3. Yet, the cheat still remains faithful to its parent. Cheats who repeatedly refuse the join requests of clients will similarly be detected as their good latencies to root stand to contradict their lack of children. Upon detection, *DS* will similarly relegate these cheats to the furthest leaf nodes possible.

7.4 Detection of Subtle Cheating

DS alone is unable to detect the two cases of subtle cheating, namely, inflating RTT to avoid getting children and understating RTT to the root to stay next to the root. Detection will involve the source as well as community policing by the HPAM clients. To prevent cheats from understating their RTTs to the root in order to exploit the *JSA* algorithm to move next to the root, the root will perform a RTT test to the client to verify that the client's RTT is genuinely better than all the existing L1 clients. The issue of understating the RTT will thus be detected when discrepancies arise in the two RTT readings. The cheat cannot reduce the response as the RTT test is made tamper-proof in the sense that it does not contain a timestamp for the cheat to manipulate. If the discrepancy between the cheat's claimed RTT and the source's measured RTT exceeds a tolerable threshold to take into account measurement errors, the source will reject the join and alert *DS* of the possible cheat. The cheat will have to join the data tree as per an honest client.

Detecting inflated RTT measurements is more involved as RTT tests are conducted independently by the different clients. Detection of suspicious RTT measurements will therefore involve community policing among the clients in the sense that clients will alert *DS* of suspicious cheats and the RTT result (*cprttp*) which it receives from the suspect.

Clients collect RTTs of various gossipers during the initial join process as well as during the gossip process. When a high RTT is noted with respect to the mean of the RTTs from the potential parents and gossipers (e.g. 100% more than the mean of others), the client will report it to DS. This is of course not foolproof if the client is overwhelmed with cheats. Nevertheless, the detection is the collective effort of many clients and even if some clients fail to detect them, there will be others who will. Note that the server would have been the first to lodge an alert report when it detects discrepancies in its measured RTT versus the cheat's claim although in the server's case, the *cprttp* is extremely low due to underreporting by the cheat.

DS maintains a blacklist of cheat suspects with information on IP address (*suspect*), average latency to others (*crttp*), number of alerts received (*alertcnt*), time of receiving first alert (*strtTimeAlert*), time of most recent report (*rntTimeAlert),* IP address of the most recent client (*clientpolice*) who reported on the cheat. Three more fields are included to store RTT test results, namely, suspect's parent to suspect's rtt (*parent2c),* suspect's parent to *clientpolice (parent2cp)* and *clientpolice to* suspect *(cp2c)*. As alert messages arrive, *DS* will update the blacklist accordingly. The detection algorithm on the part of *DS* is summarized in Figure 7.1. At regular intervals, *DS* will scan the list to weed out stale records and to direct the most recent host, *clientpolice*, as well as the parent of the suspect to conduct RTT tests to the suspect and report the results. The parent of the suspect also conducts a RTT test to the *clientpolice*. Using the 3-way RTT test results and the triangular inequality given in Eqn. 7.1, *DS* can estimate the RTT between the suspect and the *clientpolice* assuming shortest path unicast routing.

$$|parent2c - parent2cp| < cp2c < |parent2c + parent2cp| \qquad \text{Eqn. 7.1}$$

```
// This procedure processes  alert message sent by client who suspects that cheating has occurred.
Procedure: ProcessAlert(alert message from clientpolice)
{
For every alert received,
    if suspect∈BlackList
        crttp ← new average crttp computed from old crttp and cprttp sent by clientpolice
                        // the average excludes the source's crttp entry to avoid the undue
                        // reduction in the mean inflated crttp of the suspect
        inc(alertcnt)
        clientpolice ← IP of reporting client
        rntTimeAlert ← current time
    else
        insert new record of suspect into Blacklist
        crttp ← cprttp
        alertcnt ← 1
        clientpolice  ← Cnt_threshold IP of reporting client
        strtTimeAlert, rntTimeAlert ← current time
    end if
}

// This procedure is run at regular interval (Interval) by DS to clean up the cheats
Procedure:  CheckSuspect
{
For every record in BlackList,

    Case 1:  rntTimeAlert < Interval_n-2      // last alert is made2 intervals away.

             Delete entry of suspect          // considered as stale data.

    Case 2:  alertcnt >Cnt_threshold          // Cnt_threshold used to weed out sporadic reports

             clientpolice←TestRtt(clientpolice, suspect)      // perform 3-way RTT tests
             parent(suspect)←TestRtt(parent(suspect), suspect)
             parent(suspect)←TestRtt( parent(suspect),clientpolice)

             if cp2c≥ |parent2c + parent2cp|                  // Refer to Eqn. 7.1
                 Confirm suspect is a cheat
                 Get leaf node client who is faraway from the root to adopt suspect
              elseif  (childnum(suspect) ≤ 0) and  (rntTimeAlert-strtTimeAlert)≥2xInterval) and
                                                   (level(suspect))≤Level_threshold
                 Confirm suspect is a cheat
                 Get leaf node client who is faraway from the root to adopt suspect
              end
}
```

Figure 7.1: Algorithm incorporated into *DS* to detect subtle RTT cheats.

A subtle cheat will not cheat on its parent's RTT tests as this can be easily verified from the actual values stored in *DS*. Instead, it can recognize its parent's RTT test and respond with the correct value *(parent2c)* to be consistent. It will cheat on the *clientpolice*'s RTT test (*cp2c)*, thinking that it is part of the gossip process. No cheating is expected from an honest *clientpolice (parent2cp)* unless it is also a cheat. In which case, the RTT tests comparison will be unlikely to detect the cheat. If the reported *cp2c* is greater than the upper limit of Eqn. 7.1, it is confirmed that the suspect is a cheat since the other two values are actual data. *DS* will thus direct a leaf node further down the tree to adopt the cheat, releasing its slots for the honest clients.

If the RTT tests comparison fails, *DS* will make use of heuristic checks to infer if the suspect is a cheat. If a suspect fulfills the conditions that it has zero or one child; it has been reported suspicious for a prolonged period of time and it is occupying the levels which are closest to the roots, *DS* will classify it as a cheat and cast its position away from the root.

7.5 Impact of Subtle Cheating on HPAM

7.5.1 Simulation Methodology

To illustrate the impact of subtle cheating in HPAM, simulations are performed as per the setup detailed in Section 5.4.2 with infinite buffer size. The same member compositions used for group size of 100 members are run over the different transit-stub topologies. Subtle cheats are introduced on a random basis constituting 10%, 25%, 50% and 75% of the group members and these cheats inflate their RTTs by 5s in response to RTT tests.

7.5.2 Performance Metric used

The performance metrics used to evaluate the impact on HPAM is the Average Stress Ratio and the *RDP* (stretch) Ratio. The former illustrates the effect on network resources while the latter provides an insight into the impact on HPAM's *RDP* performance. Average Stress Ratio captures the ratio of the average link stress in the simulation runs where cheating occurs over that in the simulations of Section 5.4.2 where all the clients are completely honest. Similarly, the RDP ratio is defined as the RDP of each individual client in the simulation runs where cheats exist divided by the respective RDP of each individual client during the simulation runs featuring all honest clients in Section 5.4.2. The ratios are used to better reflect the relative penalties that are incurred as a result of the presence of cheats as opposed to fully honest clients in HPAM.

7.5.3 Simulation Results

The simulation results are presented in Table 7.1. The mean stress ratio of HPAM is seen to be statistically quite independent of the number of cheats. It is only slightly higher than HPAM without cheats. However, the maximum stress ratio increases in the presence of cheats. This is because cheats deprive honest clients who are in close proximity to choose them as parents. Instead these clients are forced to choose parents who are further away. This increases the probability of multiple copies of data being sent via the same physical network links since the data has to traverse more physical links to reach the honest clients.

When the number of cheats continues to increase, the honest clients will be overwhelmed and will have to turn to the cheats who are physically closest to them as parents. This helps to reduce the network stress and accounts for the drop in the mean and maximum stress ratio when the number of cheats hit 75% of the group size.

Table 7.1: Average stress and RDP ratios of HPAM clients in subtle cheating environments without the corresponding detection algorithms.

	Overall		Cheats		Honest Clients
	Average Stress Ratio		RDP Ratio		Mean RDP Ratio
	Mean	Maximum	Mean	Minimum	
10% Cheats	1.032	1.123	2.350	0.426	2.083
25% Cheats	1.109	1.485	2.772	0.560	3.000
50% Cheats	1.127	1.815	2.869	0.514	3.503
75% Cheats	1.113	1.581	2.768	0.541	4.900

The RDP ratios in Table 7.1 are segregated into two categories, one for the cheats themselves and the other for the honest clients. The idea is to illustrate how much penalty and/or gain the cheats have incurred versus the penalty the honest clients have suffered as a result of the cheating. The cheats under declare their RTTs to replace existing level 1 clients who may be honest clients or cheats too. These cheats may or may not be genuinely close to the root and by hogging the level 1 slots, they deprive genuinely close clients from occupying these slots resulting in a highly inefficient data distribution tree. As the number of cheats increases, competition intensifies among cheats and they too are being displaced away from the root. The process continues until most of the levels close to the root are overwhelmed with cheats. Hence, the honest clients are readily being pushed away from the

root by the cheats as they are being discouraged by the cheats' high RTTs to seek them as parents, resulting in a mean RDP ratio which increases steadily with the increase in the number of cheats. The impact is also seen in the cheats themselves with the mean RDP ratio increasing with the number of cheats as competition becomes fiercer. However, some cheats do get rewarded as the minimum RDP ratio is very much less than 1. These are the cheats who have extremely low fabricated RTTs to maintain their positions next to the root. In comparison with HPAM without cheats where only genuinely close clients are placed next to the root so that everyone gains from an efficient data tree, the presence of cheats only benefit some cheating clients and it degrades the performance for the majority of the clients.

7.6 Impact of Detection of Subtle Cheating on HPAM

The same sets of simulations with varying number of cheats as per Section 7.4 are performed but this time, the algorithms for subtle cheat detection described in Section 7.3 are incorporated into HPAM. The interval for *DS* to check for cheats is set at 10s while $Cnt_{threshold}$, $Level_{threshold}$ are set to 10 and 3 respectively. The RTT threshold used by clients to alert *DS* is set at twice the mean of other RTTs measured while the RTT discrepancy threshold used by the source to verify a cheat's RTT claim is set at 10%. The corresponding results of Table 7.1 after the incorporation of Subtle Cheating Detection are shown in Table 7.2.

The mean and maximum stress ratios do not change significantly from Table 7.1 for the same reason that clients who are in close proximity do not have cheats as parents although

the cause of such a phenomenon has changed here. In Table 7.1's case, the cheats avoid taking children even though they have a chance to do so while in Table 7.2's case, the cheats do not even have this chance as the detection algorithm pushes them to the leaf nodes. When the cheats do not take children even though they are in close proximity, the honest clients are forced to choose parents who are further away. This increases the probability of multiple copies of data being sent via the same physical network links since the data has to traverse more physical links to reach the honest clients thereby increasing the maximum stress ratio. Similar to Table 7.2, when the number of cheats continues to increase, the honest clients will be overwhelmed and will have to turn to the cheats who are physically closest to them as parents. This helps to reduce the network stress and accounts for the drop in the mean and maximum stress ratios when the number of cheats hit 75% of the group size.

Table 7.2: Average stress and RDP ratios of HPAM clients in subtle cheating environments with the corresponding detection incorporated.

	Overall		Cheats		Honest Clients
	Average Stress Ratio		Client's RDP Ratio		Mean Client's RDP Ratio
	Mean	Maximum	Mean	Minimum	
10% Cheats	1.019	1.207	2.659	1.026	1.127
25% Cheats	1.115	1.322	2.556	1.034	1.186
50% Cheats	1.134	1.611	2.460	1.028	1.113
75% Cheats	1.063	1.400	2.311	1.017	1.124

The minimum RDP ratio of cheats increases significantly over that in Table 7.1 as they can no longer bluff their way to be next to the root unless they are genuinely very close to

the root. In fact, the minimum RDP ratio is hovering about 1.0 which correlates with the case of HPAM without cheats where only genuinely close clients can be placed next to the root.

The mean RDP ratio for cheats exhibits an unexpected trend. When the cheat numbers are low, the mean RDP ratio increases compared to its counterpart in Table 7.1 as a result of the detection algorithm. However, as the number of cheats increases, the mean RDP ratio actually drops below that of Table 7.1 despite the cheats being pushed to random leaf nodes at the point of detection. This phenomenon can be explained by the fact that the data tree has been made more efficient due to the prevention of non-deserving cheats from hogging the slots close to the root. Hence, it is not surprising that the mean RDP ratio turns out to be superior to that in Table 7.1 when the number of cheats increases due to the law of averaging effect as there are now more cheats to enjoy the fruits of a more efficient data tree compared to the case where the number of cheats are smaller.

The mean RDP ratio for honest clients is markedly reduced although they are still not on par with HPAM without cheats. This is because by banishing the cheats to the leaf nodes, their potential in contributing to an efficient data tree cannot be fully exploited. Hence the data tree is still not as efficient as that without cheats.

It can be seen that for a subtle cheat to be detected, it involves a lot of overhead given the amount of message exchanges, RTT tests and checks performed. The 3-way RTT tests of the detection algorithm cannot detect composite cheating when two or more of the three parties involved are cheats. *DS* has to incorporate an additional heuristic to detect such cases. Cheating not only affects the QoS delivered by application level multicast protocols,

it also affects its scaleability given the higher protocol overheads involved. Cheating is certainly highly unproductive for application level multicast protocols. HPAM enjoys an edge in the ability to detect cheats as it has the assistance of a centralized entity, the *DS*. In distributed protocols with no centralized entities, detection against cheats will be a research challenge.

7.7 Summary

Although HPAM is designed in the spirit of honest sharing and collaboration, there are incentives for clients to cheat to move close to the root and to do away with the burden of carrying children. A combination of *DS* and community policing by the clients is used in the detection techniques incorporated into HPAM. No doubt, these techniques are successful in weeding out the cheats, they do take a toll on HPAM's performance in terms of the QoS delivered to the honest clients and the protocol overheads. These techniques can be switched off if their services are not required. Cheating is counterproductive and is expensive to detect and no application level multicast protocols can fully protect themselves against cheats. After all, cheating runs counter to the fundamental principle of application level multicast protocols. Cheat detection increases the complexity of the protocols and overheads which runs counter to the simplicity principle of HPAM's design.

Chapter 8

Conclusion & Future Directions

This chapter concludes the dissertation by reviewing the contributions of the research work and presenting the directions for future research in this area.

8.1. Conclusions

In this dissertation, a protocol, Hybrid Protocol for Application Layer Multicast (HPAM), for efficient and ubiquitous live media streaming over the Internet at the application layer level without multicast support from the underlying network infrastructure is proposed. HPAM operates as an overlay on top of a dynamic, unpredictable and heterogeneous Internet and allows accelerated application deployment. It does not rely on IP multicast support from the network infrastructure or any other support that is not already ubiquitous and hence does not require any modifications to the substrate network.

HPAM is designed to be simple and scaleable and yet is topology-aware to adapt to changes in network dynamics and group membership. The data distribution overlay is peer-to-peer in nature and comprises source-specific, loop-free, self-organizing, self-improving trees whose nodes are the individual clients of the multicast session with no proxies used. Data distribution is based on a best effort model. The data distribution tree is constructed to minimize latency to the root in HPAM; while the enhanced version, HPAM-D, minimizes latency as the primary goal and reduces loss rate as the secondary objective. Unless otherwise specified, the term HPAM shall refer to HPAM and HPAM-D collectively.

HPAM has adopted a hybrid approach to overlay construction, refinement and maintenance by exploiting the simplicity and optimality of a centralized controller with the robustness and scaleability of distributed receivers based technique. The lightweight controller is used to facilitate some group management functions such as peer discovery, join process and to serve as a reliable back-up to support error recovery in the eventuality that the distributed algorithm fails. The actual tree construction, routing, refinement and recovery from partitions are left to the individual clients.

HPAM has been evaluated on metrics such as latency, link stress, protocol overhead, tree cost, message load on DS as well as its ability to adapt to group dynamics, to improve the tree quality and to repair and recover from partition. The main contribution of HPAM is that it can build and maintain application layer multicast trees with very little overheads and low performance tradeoffs.

Comparative studies of HPAM's performance versus the tree-based, distributed protocol, HM, show that HPAM yields comparable performance in most aspects despite the simplicity of its protocols. In fact, it is better than HM in terms of latency and its lightweight, centralized server with distributed tree refinement and partition recovery allows a faster join for new clients and swifter partition recovery compared to HM's iterative protocol. As such, new HM clients need to be fostered first until a permanent parent is located. Comparative evaluation of HPAM to another tree-based but fully centralized protocol, HBM, shows that HPAM is unable to achieve the level of optimality in terms of latency and treecost. However, it is more responsive to changes in membership dynamics and more importantly, its performance is achieved at a very much lower protocol overhead. State overhead maintained at each HPAM client is independent of the group size and is a constant at $O(k)$ where k is the sum of the outdegree of a client and the number of gossipers in the gossip cache compared to $O(N)$ where N is the group size for the case of centralized protocol such as HBM.

Other major contributions in the HPAM protocol include the *JoinSource&Adopt* (*JSA*) algorithm to specifically ensure that only clients with very small latency can occupy the level 1 (the level nearest to the root of the tree) slots of the data tree. Since the latencies of level 1 clients will propagate to every client downstream and increase their latencies to the root, it is of paramount importance that clients with long latency to the root not hog these level 1 slots. Another contribution is the *Gossip* and *Spiral* mechanisms to facilitate tree refinement and speedy repair. These innovations result in lower protocol overheads than a fully centralized system and a quicker response to group dynamics and network environment than a fully distributed system.

Another major contribution of HPAM is the incorporation of loss rate as a second performance metric in HPAM-D as opposed to the use of only latency in building and improving the data distribution tree. This not only allows HPAM-D to maintain the level of optimality exhibited by its latency-based counterpart in terms of RDP but it is also able to maintain a high percentage of clients with acceptable loss rate in a dynamic network environment where there are congestion and losses. All these are achieved with negligible protocol overhead over its latency-based counterpart as the loss rate experienced by a client can be passively measured from the data packets received.

Adding to the major innovation list of HPAM-D is the *RLR* heuristics incorporated for the detection of possible local congestion that exists between a client and its parent. A client will only switch parent if the *RLR* is assessed to be poor and will otherwise delay its parent switch and wait for its parent to do the switching first. The *RLR* between a client and its parent is derived from the loss rate data maintained by all the clients. The *RLR*-heuristics not only helps a client to reduce unnecessary parent switches thereby reducing protocol overheads but also contributes to the ability of the HPAM-D to maintain a low loss rate for its clients without compromising the RDP performance.

Another major innovation of HPAM is the incorporation of detection algorithms against cheating HPAM clients. Although HPAM is designed in the spirit of honest sharing and collaboration, there are incentives for clients to cheat to move close to the root and to do away with the burden of carrying children. HPAM uses a combination of *DS* and community policing by the clients to detect subtle cheats who fabricate

their RTT measurements. These techniques have proven to be effective in identifying and isolating the cheats albeit at a cost to HPAM's performance in terms of the QoS delivered to the honest clients and the protocol overheads. It is proposed that these techniques be switched off if their services are not required. Cheating is counterproductive and is expensive to detect and no application level multicast protocols can fully protect themselves against cheats. After all, cheating runs counter to the fundamental principle of application level multicast protocols and cheat detection increases the complexity of the protocols and overheads which undermine the very simplicity principle of HPAM's design.

Last but not least, a comprehensive literature survey on the milestone application level multicast protocols is performed and the results summarized into two tables to facilitate a quick insight into and understanding of their contributions since the birth of application level multicast five years ago. Table 2.8 summarizes their performance based on factors such as scaleability, efficiency of the protocols in terms of stress on the underlying network, relative delay penalty, protocol overheads and failure tolerance. Table 2.9 summarizes their properties in terms of topology design, service models and architecture.

8.2. Directions for Future Work

This research has opened up a few interesting avenues for future work.

8.2.1 Replication of DS

The weakest link in HPAM is certainly the DS which constitutes a single point of failure. Although a DS failure will not impact existing members in their data distribution, it will prevent new members from joining the group. Another concern pertaining to DS is its scaleability as it maintains the state of all HPAM members although it does not maintain detailed client-to-client connectivity status information. As group membership increases, the load on DS will increase correspondingly.

A solution to the fault tolerant and scaleability problem is to use a cluster of servers to host the DS [124]. Server clustering allows each server to act on its own while using one common, redundant mass storage device. If one server goes down, the second one takes over while still maintaining its own workload. The server cluster therefore provides ready hot standbys as well as shares the processing load. Hence, the failover time is greatly reduced because each server is already attached to the database that clients are seeking. Moreover, there is less chance that data will be lost during a system failure.

Another approach is to use multiple mirrored DSs to complement the principal DS. The principal DS is the active production server while the mirrors receive duplicate

data. In the event the principal DS fails, one of the mirrors is ready to take over in a matter of time. A major benefit is that the mirrors can be in separate locations for disaster protection provided a powerful enough network connection is provided.

8.2.2 Reduction in Number of RTT Probes

HPAM uses RTT as the latency metric for its protocols and assumes that all paths are symmetric. RTT measures 2-way delay and such active RTT measurements constitute a major part of the HPAM protocol overheads. Using one-way delay will therefore reduce RTT-related overheads by 50%. One method used to determine one-way delay is to estimate it from cyclic-path delay measurements [125]. This method does not require any kind of synchronization among the HPAM nodes and takes into account the asymmetric nature of the network which is not uncommon in the Internet. Another way is to use Network Time Protocol (NTP) [126][127] to synchronize all the clients' clocks and perform one-way measurements as per [128], which can yield very accurate results if an NTP server is present in each measurement site. In fact, the National Institute of Information and Communications Technology (NICT) of Japan has taken the first step in January 2005 to provide an NTP service in a national effort to allow all computers who access the service to synchronize their internal clocks [129]. If all the clients' clocks are synchronized, one-way delay information can be extracted from the heartbeat and gossip exchange messages in the HPAM protocol without doing active one-way delay measurements although the accuracy is very much influenced by the clock resolution of the client's machine.

In fact, delay measurements can be reduced to a complementary role if Internet Distance Map Service (IDMaps) [130][131] information is available to the clients. IDMaps is a scaleable Internet-wide architecture which measures and disseminates distance information on the global Internet. Higher level services can collect such delay information to build a virtual distance map of the Internet and estimate the delay between any pair of IP addresses.

8.2.3 Proxies for HPAM

As HPAM is a peer-to-peer based protocol, a direction which is worth pursuing for future research is a proxy-based HPAM. Clients in the current HPAM are fragile as they can leave or fail any time and HPAM has no control over their robustness nor functionality. Proxies, on the other hand, are dedicated servers which are better provisioned with more stable network performance and are less likely to fail. Being better provisioned, they can perform more functionalities than normal clients and provide value-added services to suit the specific application needs such as end-to-end reliability, client authentication, data integrity, access control etc. In addition, proxies are persistent beyond the lifetime of the group sessions which thus enables them to exploit past history of network performances and reduce the number of active RTT measurements. All these can help to simplify the design of the self-organizing protocol to a certain extent. In a proxy-based system, clients joining the group will choose to join the nearest proxy instead of to fellow clients.

Another avenue to explore is the hybrid combination of proxies and normal clients. The number of proxies deployed can thus be reduced. These proxies can then assume active roles in partition recovery, network performance monitoring while the normal clients assume the data distribution role.

8.2.4 Impact of Collaborating Cheats on HPAM

Another direction for future work is to address the impact of collaborating cheats on HPAM. The cheat detection algorithms in HPAM are ineffective at detecting cheats who work in collaboration with other cheats. In the first place, the actual RTTs among them cannot be verified. To completely address this issue will require complete knowledge of the group members and their measured RTT matrix. This will render the HPAM to incur as high an overhead as the centralized protocol, HBM. To analyze the RTTs collected from only a handful of clients in order to detect the collaborating cheats is a research challenge.

The investigation into the impact of cheating in application level multicast protocols is still in its infancy and the impact of cheats in many protocols are not studied. In addition, the design of cheat detection and prevention techniques for the different types of metrics and for the various application level protocols are currently still open research problems.

8.3 Summary

In summary, this dissertation presents a comprehensive design of an application level multicast protocol for efficient and ubiquitous live media streaming over the Internet without IP multicast support at the infrastructure level. Extensive simulations and analysis have been performed to evaluate the protocol's transient and steady state performance in environments ranging from stable group membership and network environment to those with dynamic group membership and variable network characteristics. The impact on its performance in the presence of cheats is also investigated. Simulation results show that the protocol is able to deliver high QoS to its clients with reasonable protocol overheads and network stress. It is certainly not as efficient as IP multicast but it presents a simple and easily deployable solution towards solving the multipoint distribution problem.

Author's Publications

This section documents the list of related publications generated from this research.

Journal Papers

1. C. K. Yeo, B. S. Lee, M. H. Er, "A Survey of Application Level Multicast Techniques," *Computer Communications*, 27(15), pp. 1547-1568, September 2004.
2. C. K. Yeo, B. S. Lee, M. H. Er, "A Framework for Multicast Video Streaming over IP Networks", *Journal of Networks and Computer Applications*, 26(3), pp. 273-289, August 2003.
3. C. K. Yeo, B. S. Lee, M. H. Er, "A Peering Architecture for Ubiquitous IP Multicast Streaming", *ACM SIGOPS Operating Systems Review*, 36(3), pp. 82-95, July 2002.

Conference Papers

1. C. K. Yeo, B. S. Lee, M. H. Er, "Application Layer Multicast Architecture for Media Streaming", *Proc. 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, Hawaii, USA, pp. 455–460, August 2003.

2. C. K. Yeo, B. S. Lee, M. H. Er, "A Dynamic Peering Network for Multicast Streaming", *Proc. IEEE International Conference on Networking (ICN)*, Atlanta, pp. 139–149, August 2002.

3. C. K. Yeo, B. S. Lee, M. H. Er, "An Overlay for Ubiquitous Streaming over Internet", *Proc. 2nd International IFIP-TC6 Networking Conference*, Pisa, Italy, pp. 1239-1244, May 2002.

4. P. Gupta, C. K. Yeo, B. S. Lee, M. H. Er, "An Architecture for Multicast and Unicast Bridge", *Proc. 3rd International Conference on Information, Communications & Signal Processing (ICICS)*, Singapore, October 2001.

5. P. Gupta, C. K. Yeo, B. S. Lee, M. H. Er, "Dynamic Multicast Tunnelling Architecture for Multimedia Applications", *Proc. WSES/IEEE International Conference on Multimedia, Internet, Video Technologies (MIV) International Conference on CCS*, Singapore, pp. 2661-2666, September 2001.

Bibliography

[1] T. Berners-Lee, R. Cailliau, A. Loutonen, H. F. Nielsen and A. Secret, "The World Wide Web," *Communications of the Association for Computing Machinery*, 37(2) pp. 76-82, August 1998.

[2] NetRadio.com, http://www.netradio.com/

[3] World Wide Internet TV, wwiTV, http://wwitv.com

[4] C. K. Yeo, B. S. Lee and M. H. Er, "A Peering Architecture for Ubiquitous IP Multicast Streaming," *ACM SIGOPS Operating Systems Review*, 36(3), pp. 82-95, July 2002.

[5] M.H. Willebeek-LeMair, "Bamba-Audio & Video Streaming over Internet," *IBM J. Res. Develop*, 42(2), pp. 269-279, March 1998.

[6] S. McCanne and V. Jacobson, "Vic: A Flexible Framework for Packet Video," *Proc. ACM Multimedia*, pp. 51—522, November 1995.

[7] V. Jacobson and S. McCanne, *Visual Audio Tool*, Lawrence Berkeley Laboratory, ftp:ftp.ee.lbl.gov/conferencing/vat.

[8] Y. Chu, S. G. Rao, S. Seshan and H. Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," *Proc. ACM SIGCOMM*, August 2001.

[9] V. Jacobson and S. McCanne, *LBL Whiteboard*, Lawrence Berkeley Laboratory, ftp://ftp.ee.lbl.gov/conferencing/wb.

[10] P. Parnes, K. Synnes and D. Schefstrom, "mSTAR: Enabling Collaborative Applications on the Internet," *IEEE Internet Computing*, pp. 32-39, September-October 2000.

[11] OpenSkies, *Massive Multi-Player Networking for Games*, Cybernet systems Inc., http://www.openskies.net/system/system.html.

[12] CNET Networks, *Real-Time Streaming Stock Portfolio*, http://news.com.com/.

[13] Computer Science and Telecommunications Board (CSTB), National Research Council, *The Internet Under Crisis Conditions: Learning from September 11*, National Academies Press, Washington DC, pp. 61, 2003.

[14] T. M. Eubanks, *Multicast after 9/11/2001*, www.nanog.org/mtg-0110/ppt/eubanks/, North American Network Operators' Group 23 Meeting, October 2001 (NANOG).

[15] S. Deering, D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANS," *ACM Trans. On Comp. Syst.*, 8(2), pp. 85-100, May 1990.

[16] S. E. Deering, *Multicast Routing in a Datagram Internetwork*, Ph.D thesis, Stanford University, December 1991.

[17] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu and L. Wei, "The PIM Architecture for Wide-Area Multicast Routing," *IEEE/ACM Trans. On Networking*, pp. 784-803, December 1997.

[18] D. Waitzman, C. Partridge and S. Deering, *Distance Vector Multicast Routing Protocol*, RFC 1075, November 1988.

[19] T. Ballardie, P. Francis, J. Crowcroft, "Core Based Trees (CBT)," *Proc. ACM SIGCOMM*, pp. 85-95, September 1993.

[20] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, "An Architecture for Wide-area Multicast Routing," *IEEE/ACM Trans. on Networking*, 4(2), April 1996.

[21] K. Obraczka, "Multicast Transport Mechanisms: A Survey and Taxonomy", *IEEE Communications Magazine*, January 1998.

[22] C. Diot, B. N. Levine, B. Lyles, H. Kassan and D. Balensiefen, "Deployment Issues for the IP Multicast Service and Architecture," *IEEE Networks special issue on Multicasting*, 14(1) pp. 78-88, Jan/Feb 2000.

[23] H. Eriksson, "Mbone: The Multicast Backbone," *Communications of ACM*, vol. 37, pp. 54-60, August 1994.

[24] R. Perkman, C. Lee, A. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo and M. Green, *Simple Multicast: A Design for Simple, Low-Overhead Multicast*, IETF draft, draft-perlman-simple-multicast-03.txt, October 1999.

[25] H. Hoolbrook and D. Cheriton, "IP Multicast Channels: EXPRESS Support for Large-scale Single Source Applications," *Proc. ACM SIGCOMM*, September 1999.

[26] H. Hoolbrook and B. Cain, *Source Specific Multicast for IP*, IEFT draft, <draft-ietf-ssm-ssm-arch-03.txt>, 7 May 2003.

[27] Y. Chu, S. G. Rao, S. Seshan and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications*, 20(8), October 2002.

[28] Y. Chu, S. G. Rao and H. Zhang, "A Case for End System Multicast," *Proc. ACM SIGMETRICS*, pp. 1-12, June 2000.

[29] J. Jannotti, D. K. Gifford and K. L. Johnson, "Overcast: Reliable Multicasting with an Overlay Network," *Proc. Operating System Design and Implementation (OSDI)*, pp. 197-212, October 2000.

[30] J. Liebeherr, M. Nahas and W. Si, "Application-layer Multicast with Delaunay Triangulations," *IEEE Journal on Selected Areas in Communications*, 20(8), October 2002.

[31] P. Francis, *Yoid: Your Own Internet Distribution*, http://www.isi.edu/div7/yoid/, March 2001.

[32] D. Pendarakis, S. Shi, D. Verma and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," *Proc. 3rd Usenix Symposium on Internet Technologies & Systems (USITS)*, March 2001.

[33] S. Q. Zhang, B. Y. Zhao, A. D. Joseph, R. H. Jatz and J. D. Kubiatowicz, "Bayeux: An Architecture for Scaleable and Fault-tolerant Wide-Area Data Dissemination," *Proc. NOSSDAV*, April 2001.

[34] M. Castro, P. Druschel, A. M. Kermarrec and A. Rowstron, "SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure," *IEEE Journal on Selected Areas in Communications*, 20(8), October 2002.

[35] A. Rowstron, A. M. Kermarrec, M. Castro and P. Druschel, "SCRIBE: The Design of a Large-scale Event Modification Infrastructure," *Proc. 3rd Int. Workshop on Networked Group Communication (NGC)*, 2001.

[36] S. Ratnasamy, M. Handley, R. Karp and S. Shenkar, "Application-Level Multicast using Content Addressable Networks", *Proc. 3rd Int. Workshop on Networked Group Communication (NGC)*, pp. 14-29, 2001.

[37] Y. D. Chawathe, *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*, Ph.D thesis, Stanford University, September 2000.

[38] S. Banerjee, B. Bhattacharjee and C. Kommareddy, "Scaleable Application Layer Multicast," *Proc. ACM SIGCOMM*, August 2002.

[39] B. Zhang, S. Jamin and L. Zhang, "Host Multicast: A Framework for Delivering Multicast to End Users," *Proc. INFOCOM*, June 2002.

[40] D. A. Helder and Sugih Jamin, "End-host Multicast Communication using Switch-Trees Protocol," *Proc. 2nd IEEE/ACM Int. Sym. On Cluster Computing and the Grid*, May 2002.

[41] S. Banerjee, C, Kommareddy, K. Kar, B. Bhattacharjee and S. Khulle, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications", *Proc. INFOCOM*, April 2003.

[42] A. El-Sayed, *Application-level Multicast Transmission Techniques over the Internet*, PhD Thesis, Institut National Polytechhique de Grenoble, March 2004.

[43] V. Roca, A. El-Sayed, "A Host-based Multicast (HBM) Solution for Group Communications", *Proc. 1st IEEE Int. Conf. on Networking (ICN'01)*, pp. 610-619, July 2001.

[44] A. El-Sayed, V. Roca, "Improving the Scalability of an Application-level Group Communication Protocol", *Proc. 10th Int. Conf. on Telecommunications (ICT'03)*, February 2003.

[45] A. El-Sayed, V. Roca, "On robustness in Application-Level multicast: the case of HBM", *Proc. 9th IEEE Sym. On Computers and Communications (ISCC'04)*, July 2004.

[46] C. K. Yeo, B. S. Lee and M. H. Er, "A Framework for Multicast Video Streaming over IP Networks," *Journal of Networks and Computer Applications*, 26(3), 273-286, August 2003.

[47] C. K. Yeo, B. S. Lee, M. H. Er, "An Overlay for Ubiquitous Streaming over Internet," *Proc. 2nd Int. IFIP-TC6 Networking Conference*, pp. 1239-1244, May 2002.

[48] C. K. Yeo, B. S. Lee, M. H. Er, "A Dynamic Peering Network for Multicast Streaming", *Proc. IEEE International Conference on Networking (ICN)*, Atlanta, pp. 139–149, Aug 2002.

[49] L. Peterson, T. Anderson, D. Culler and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," *Proc. 1st ACM Workshop on Hot Topics in Networks (HotNets-I)*, October 2002

[50] Streaming Media 2004-2007: Market Development and User Data Analysis, *AccuStream iMedia Research*, http://www.accustreamresearch.com/products/streaming2007analysis.html, January 2005.

[51] A. El-Sayed, V. Roca, L. Mathy, "A Survey of Proposals for an Alternative Group Communication Service", *IEEE Network*, January/February 2003.

[52] C. K. Yeo, B. S. Lee, M. H. Er, "A Survey of Application Level Multicast Techniques," *Computer Communications*, 27(15), pp. 1547-1568, September 2004.

[53] S. Jain, R. Mahajan, D. Wetherall, G. Borriello and S. D. Gribble, *A Comparison of Large-Scale Overlay Management Techniques*, Technical Report UV-CSE 02-02-02, University of Washington, February 2002.

[54] L. Mathy, R. Canonico and D, Hutchison, "An Overlay Tree Building Control Protocol," *Proc. 3rd Int. Workshop on Networked Group Communication (NGC)*, pp. 76-87, 2001.

[55] S. Floyd, V. Jacobson, C. Liu, S. McCanne and L. Zhang, "A Reliable Multicast Framework for Light –Weight Session and Application Level Framing", *IEEE/ACM Trans. Networking*, 5(6), pp. 784-803, December 1997.

[56] T. Speakman, D. Farinacci, S. Lin and A. Tweedly, *Pragmatic General Multicast (PGM) Reliable Transport Protocol*, CISCO Systems, Internet Draft, 1998.

[57] K. Yano and S. McCanne, "The Breadcrumb Forwarding Service: A Synthesis of PGM and EXPRESS to Improve and Simplify Global IP Multicast," *ACM Computer Communication Review*, 30(2), April 2000.

[58] P. Francis, *Yoid: Extending the Internet Multicast Architecture*, http://www.isi.edu/div7/yoid/, April 2000.

[59] B. Y. Zhao, J. Kubiatowicz and A. Joseph, *Tapestry: An Infrastructure for Fault-Tolerant Wide-area Location and Routing*, Technical report, UCB/CSD-01-1141, University of California, Berkeley, April 2001.

[60] S. Banerjee, B. Bhattacharjee, *A Comparative Study of Application Layer Multicast Protocols*, Unpublished report, Department of Computer Science, University of Maryland, http://www.cs.umd.edu/suman/publications.html.

[61] Z. Wang and J. Crowcroft, "Bandwidth-delay based routing algorithms," *IEEE Globecom*, November 1995.

[62] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. E. Sturgis, D. C. Swinehart and D. B. Terry, "Epidemic Algorithms for Replicated Database Maintenance," *ACM Operating Systems Review*, pp. 22:8-32, January 1988.

[63] Q. Sun and D.C. Sturman, "A Gossip-based Reliable Multicast for Large-scale High-Throughput Applications," *Proc. IEEE Int. Conf. on Dependable Systems and Networks*, pp. 347-358, 2000.

[64] R. Tenesse, Y. Minsky and M. Hayden, "A Gossip-style Failure Detection Service," *Proc. IFIP Int. Conf. On Distributed Systems Platforms and Open Distributed Processing (Middleware)*, pp. 55-70, September 1998.

[65] S. Ratnasamy, P. Francis, M. Handley and R. Karp, "A Scaleable Content-Addressable Network", *Proc. ACM SIGCOMM*, August 2001.

[66] M. Berg, M. Krevekd, M. Overmas and O. Schwarzkopf, *Computational Geometry, Algorithms and Applications,* Springer Verlag, 1997.

[67] R. Sibson, "Locally Equiangular Triangulations," *The Computer Journal*, 21(3), pp. 243-245, 1977.

[68] B. N. Karp, *Geographical Routing or Wireless Networks*, PhD thesis, Harvard University, 2000.

[69] E. Kranakis, H. Singh and J. Urrutia, "Compass Routing on Geometric Networks," *Proc. Of 11th Canadian Conf. On Computational Geometry*, pp. 51-54, August 1999.

[70] M. J. B. Robshaw, "MD2, MD4, MD5, SHA and other hash functions," *Tech. Rep. TR-101 ver. 4.0*, RSA Labs, 1995.

[71] C. G. Plaxton, R. Rajaraman and A. W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Proc. 9th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, June 1997.

[72] A. Rowstron and P. Druschel, "Pastry: Scaleable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems," *Proc. IFIP/ACM Middleware*, November 2001.

[73] A. Ballardie, *Core Based Trees (CBT) Multicast Routing Architecture*, Internet Request for Comments RFC 2201, September 1997.

[74] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma and L. Wei, *Protocol Independent Multicast - Sparse Mode (pim-sm): Protocol Specification,* Internet Request for Comments RFC 2117, June 1997.

[75] D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma and A. Helmy, *A Protocol Independent Multicast – Dense Mode (pim-dm): Protocol Specification,* Internet Request for Comments RFC 2117, June 1997.

[76] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong, *"Freenet: A Distributed Anonymous Information Storage and Retrieval System"*, ICSI Workshop on Design Issues in Anonymity and Unobservability, July 2000.

[77] B. Zhang, H.T. Mouftah, *"Forwarding State Scalability for Multicast Provisioning in IP Networks"*, IEEE Communication Magazine, pp. 46-51, June 2003.

[78] H. Asaeda, S. Suzuki, K. Kobayashi, J. Murai, *"Architecture for IP Multicast Deployment: Challenges and Practice"*, IEICE Trans. Communications, 89(4), pp. 1 – 7, April 2006.

[79] K. Sarac, K.C. Almeroth, *"Monitoring IP Multicast in the Internet: Recent Advances and Ongoing Challenges"*, IEEE Communications Magazine, pp. 85 – 91, October 2005.

[80] K.C. Almeroth, *"The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment"*, IEEE Network, pp. 10 – 20, January/February 2000.

[81] Microsoft Windows Media 9 Series for Business, http://www.microsoft.com/windows/windowsmedia/forpros/enterprise.aspx

[82] T. Morin, et. al., *"Requirements for Multicast in L3 Provider-Provisioned VPNs"*, draft-ieft-l3vpn-ppvpn-mcast-reqts-08, 29 May 2006.

[83] Internet2, http://www.internet2.edu/.

[84] T. Bates, Y. Rekhter, R. Chandra, D. Katz, "*Multiprotocol Extensions for BGP-4*", RFC 2858, June 2000.

[85] D. Farinacci, Y. Rekhter, P. Lothberg, H. Kilmer, J. Hall, "*Multicast Source Discovery Protocol (MSDP)*", Internet Engineering Task Force (IETF), draft-farinaccimsdp -*.txt, June 1998.

[86] J. Jamison, R.Wilder, "vBNS: The Internet Fast Lane for Research and Education", IEEE Communications, January 1997.

[87] J. Jamison, R. Nicklas, G. Miller, K. Thompson, R. Wilder, L. Cunninghan, C. Song, "vBNS: Not Your Father's Internet", IEEE Spectrum, 35(7), pp. 38 – 46, 1998.

[88] Abilene Network, http://abilene.internet2.edu/.

[89] Internet2, *Digital Video Transport Service (DVTS)*, http://apps.internet2.edu/dvts.html.

[90] WIDE Project, http://www.wide.ad.jp/.

[91] Northwestern University, *C-SPAN*, http://www.i2-multicast.northwestern.edu.

[92] Broadcast with Windows Media, http://www.microsoft.com/windows/windowsmedia/forpros/broadcast.aspx.

[93] Nine Systems, http://www.ninesystems.com.

[94] Microsoft Research Asia, Peer-to-Peer Multimedia Networking, http://research.microsoft.com/wn/p2pmedia.aspx.

[95] J. Postel, *Transmission Control Protocol*, RFC 793, September 1981.

[96] J. Postel, *User Datagram Protocol*, RFC 768, August 1980.

[97] H. Schulzrinne, G. M. D. Fokus, S. Casner, R. Frederick and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889, January 1996.

[98] D. Meyer, *Administratively Scoped IP Multicast*, RFC 2365, July 1998.

[99] D. Thaler, M. Handley and D. Estrin, *The Internet Multicast Address Allocation Architecture*, RFC 2908, September 2000.

[100] C. Hedrick, *Routing Information Protocol*, RFC 1058, June 1988.

[101] J. M. McQuillan, I. Richer and E. C. Rosen, 'New Routing Algorithms for the ARPANET,', *IEEE Trans. on Communications*, 25(8), 1980.

[102] J. Moy, *Open Shortest Path First (OSPF) Version 2*, RFC 1247, July 1991.

[103] K. Lougheed and Y. Rekhter, *A Border Gateway Protocol (BGP)*, RFC 1247, June 1989.

[104] J. Postel, *Internet Control Message Protocol*, RFC 792, September 1981.

[105] R. Bush, T. Griffin. D. Meyer, *Some Internet Architectural Guidelines and Philosophy*, ietf draft-ymbk-arch-guidelines-05.txt, http://www.ietf.org/internet-drafts/draft-ymbk-arch-guidelines-05.txt, August 2002.

[106] J. Doyle, J. Carlson, S. Low, F. Paganini, G. Vinnicombe, W. Willinger, J. Hickey, P. Parrilo and L. Vandenberghe, "Robustness and the Internet: Theoretical Foundations," *Robust Design: A Repertoire form Biology, Ecology and Engineering*, E. Jen, ed. Oxford University Press, netlab.caltech.edu/pub/papers/RIPartII.pdf, July 2004.

[107] J. H. Saltzer, D. P. Reed and D. D. Clark, "End-to-End Arguments in System Design," *ACM Trans. On Computer Systems*, 2(4), pp. 277-288, November 1984.

[108] B. Carpenter (Editor), *Architectural Principles of the Internet*, RFC 1958, June 1996.

[109] W. Willinger and J. Doyle, "Robustness and the Internet: Design and Evolution," *Robust Design: A Repertoire form Biology, Ecology and Engineering*, E. Jen, ed. Oxford University Press, July 2004.

[110] M. Handley, "Session directories and scalable Internet multicast address allocation", *ACM Computer Communication Review*, 28(4), pp. 105-116, September 1998.

[111] P. Gupta, C. K. Yeo, B. S. Lee, M. H. Er, "An Architecture for Multicast and Unicast Bridge", *Proc. 3rd International Conference on Information, Communications & Signal Processing (ICICS)*, Singapore, October 2001.

[112] P. Gupta, C. K. Yeo, B. S. Lee, M. H. Er, "Dynamic Multicast Tunnelling Architecture for Multimedia Applications", *Proc. WSES/IEEE International Conference on Multimedia, Internet, Video Technologies (MIV) International Conference on CCS*, Singapore, pp. 2661-2666, September 2001.

[113] C. K. Yeo, B. S. Lee, M. H. Er, "Application Layer Multicast Architecture for Media Streaming", *Proc. 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, Hawaii, USA, pp. 455–460, August 2003.

[114] R. Beverly, K. Claffy, "Wide Area IP Multicast Traffic Characterization", *IEEE Network*, 17(1), pp. 8-15, January-February 2003.

[115] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[116] M. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan and M. Sudan, "The Minimum Latency Problem," *Proc. ACM Symposium on Theory of Computing*, pp. 86-97, May 1994.

[117] E. W. Dijkstra. A note on two problems in connection with graphs. *Nuerische Mathematik*, 1, pp. 269-271, 1959.

[118] F. Hwang, D. Richards, R. Winger, "The Steiner Tree Problem", *Annals of Discrete Mathematics*, 53, pp. 103, 1992.

[119] R. C. Prim, "Shortest Connection Networks and Some Generalizations," *Bell Sys. Tech Journal*, 36, pp. 1389-1401, 1957.

[120] Compuserve, ModSim III Release 2.2, www.compuserve.com, August 2000.

[121] E. W. Zegura, K. L. Calvert and S. Bhattacharjee, "How to Model an Internetwork", *Proc. INFOCOM*, pp. 40-52, March 1996.

[122] B. M. Waxman, "Routing of Multipoint Connections", *IEEE Journal on Selected Areas in Communications*, 6(9), pp. 1617-1622, 1988.

[123] L. Mathy, N. Blundell, V. Roca, A. El-Sayed, "Impact of Simple Cheating in Application-Level Multicast", *Proc. INFOCOM*, March 2004.

[124] Intel Corporation, "High Availability Server Clustering Solutions", *White Paper*, 251574-001, 2002.

[125] O. Gurewitz and M. Sidi, "Estimating One-Way Delays from Cyclic-Path Delay Measurements", *Proc. IEEE Infocom*, April 2001.

[126] D. L. Mills, "Network Time Protocol Specification", RFC-1305, IETF, March 1992.

[127] D. L. Mills, "Adaptive Hybrid Clock Discipline Algorithm for the Network Time Protocol", *IEEE/ACM Trans. Networking*, 5(6), October 1998.

[128] V. Smotlacha, "One-Way Delay Measurement using NTP", *Proc. TERENA Networking Conference*, Croatia, May 2003.

[129] Japan Standard Time Group, Time Dissemination Service, http://jjy.nict.go.jp/time/index-e.html.

[130] P. Francis, S. Jamin, V. Paxson, L. Zhang, F. Gryniewics and Y. Jin, "An Architecture for a Global Internet Host Distance Estimation Service", *Proc. IEEE Infocom*, March 1999.

[131] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt and L. Zhang, "A Global Internet Host Distance Estimation Service", *ACM/IEEE Trans. On Networking*, 9(5), October 2001.

Appendix A

STEADY STATE POR COMPUTATION FOR HBM

The steady state *POR* per second for HBM is computed as follows:

$$POR = \frac{[N^2 \text{ x } RPF + N \text{ x } HbF + N(N-1)2 \text{ x } 2 \text{ x } RTTF + 2(N-1)TF]}{N \text{ x } Data\ Source\ Rate} xControlPacketSize \quad \text{Eqn. A.1}$$

where *RPF:* update frequency to RP

HbF: heartbeat frequency

RTTF: frequency of RTT test (2 messages per test since each test involves a roundtrip)

TF: tree topology update frequency by RP

Substituting the following values used in the simulation into Eqn. A.1, the steady state POR of HBM with a group size $N = 100$ is 1.459%.

RPF: update frequency to RP	1 update per client per 15 seconds. 1 update comprises *N* messages.
HbF: heartbeat frequency	1 update per client per second.
RTTF: frequency of RTT test (2 messages per test since each test involves a roundtrip)	2 RTT tests per client per 15 seconds. Each client needs to test *(N-1)* other clients.
TF: tree topology update frequency by RP (link information is sent to both clients of an overlay link for all links)	1 update to both clients of each overlay link per 15 seconds.
ControlPacketSize:	64 bytes
Data Source Rate:	1.2 Mbps

Appendix B

STEADY STATE POR COMPUTATION FOR HM

Let the maximum number of children for each HM client be *C* and the total number of clients in a HM tree be *N*. Let the root (HMRP) of the tree be level 1 and assume the nodes all borne *C* children each, then, the last level (depth of the tree), *D*, in a full HM tree is:

$$D = log_C N(C-1) \qquad \text{Eqn. B.1}$$

Assuming that the clients at each level has a uniform probability of approaching a node in its root path to make a query, the expected number of queries issued by each client at the different levels of the HM tree is:

Level 2 $$\frac{C}{N}D \qquad \text{Eqn. B.2}$$

Level 3 $$\frac{C^2}{N}(D-\frac{1}{2}) \qquad \text{Eqn. B.3}$$

Level 4 $$\frac{C^3}{N}(D-1) \qquad \text{Eqn. B.4}$$

Level D $$\frac{(N-S_{D-1})}{N}(D-\frac{D-1}{2}+\frac{1}{2}) \qquad \text{Eqn. B.5}$$

where $$S_{D-1} = C + C^2 + .. + C^{D-2} = \frac{C(C^{D-2}-1)}{C-1}$$

The expected total number of queries in a HM tree with each level, *l*, being filled by C^l nodes except for level *D* is:

$$p = \sum_{i=1}^{D-2} \frac{C^i}{N}(D - \frac{i}{2} + \frac{1}{2}) + [\frac{(N - S_{D-1})}{N}(D - \frac{D-1}{2} + \frac{1}{2})] \quad \text{Eqn. B.6}$$

The expected total number of messages per client, *TRM*, involved in *p* queries for one iteration of tree refinement is:

$$TRM = 2[p + 2(p-1)C] \quad \text{Eqn. B.7}$$

This is due to HM's algorithm used in tree refinement where a client will query a random node in its root path for its children list, then perform 2 RTT tests to each child and thereafter, repeat the query to the child which is closest to it until it reaches a leaf node.

The steady state *POR* per client per second for HM is computed as follows:

$$POR = \frac{(PathF + HbF) + \frac{TRM}{TRP}}{Data\ Source\ Rate} xControlPacketSize \quad \text{Eqn. B.9}$$

where *PathF:* root path update frequency from parent to children

HbF: heartbeat frequency

TRP: period of tree refinement

TRM: expected total number of messages for all clients per tree refinement

Eqn. B.9 provides a upper bound to the POR estimate as it assumes that the tree is balanced and the total number of levels in a HM tree is at the optimum level of $log_C N(C-1)$. In reality, HM trees are not perfectly balanced with some branches being deeper than this optimum level while the others are shallower. Moreover, not all nodes in the higher levels are fully loaded with children. The deeper branches will increase the number of queries needed but this is offset by the shallower branches

need for reduced queries. In addition, fully loaded nodes will entail more RTT tests which incur more control messages than the query itself. Hence sparse upper levels will contribute towards a lower overhead given the reduced number of RTT tests compared to fully loaded upper levels. The other point to note is that the duplicate data packets received by a client in the HM are ignored. These duplicate data packets are not factored into the POR calculation.

Substituting the following values used in the simulation into Eqn. B.9, the steady state POR of HM with a group size $N = 100$ (i.e. $D = 4$) is 0.257%.

PathF: root path update frequency	1 update per second
HbF: heartbeat frequency	1 heartbeat per second
TRP: period of tree refinement	15 seconds
TRM: expected total number of messages per client per tree refinement	$p = 0.24 + 1.26 + 1.74 = 3.24$ $TRM = 2[3.24 + 2(3.24 - 1)6]$ $= 60.24$
C: maximum number of children per client	6
ControlPacketSize:	64 bytes
Data Source Rate:	1.2 Mbps