



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Master Thesis

Tran, Thanh Minh Tu

Mobile P2P Audio Network for the iPhone and iPod Touch

Tran, Thanh Minh Tu
Mobile P2P Audio Network for the iPhone and iPod Touch

Master thesis based on the examination and study regulations for the
Master of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Thomas C. Schmidt
Second examiner: Prof. Dr. Hans-Jürgen Hotop

Day of delivery December 7th, 2009

Tran, Thanh Minh Tu

Thema der Master Thesis

Mobile P2P Audio Network for the iPhone and iPod Touch

Stichworte

Peer-to-Peer Systems, Overlay Network, Application Layer Multicast Systems, Audio Live Streaming Technologies, iPhone Audio Processing

Zusammenfassung

Gruppenkommunikation ermöglicht eine große Anzahl von Anwendungen, z.B. Sprach- und Videokonferenzen, Radio- und TV-Broadcasts, verteilte Gruppenspiele etc., und wird aus diesem Grund sowohl in kommerziellen Entwicklungen wie Forschungsprojekten häufig zum Untersuchungsgegenstand. Während IP Multicast die Verkehrslasten im Netzwerk minimiert, ist eine globale Verbreitung bisher hinter den Erwartungen zurückgeblieben. Application Layer Multicast bietet hier eine alternative Lösungsmöglichkeit zur Content-Verteilung, da es ausschliesslich auf der Basis einer Peer-to-Peer Overlay-Schicht arbeitet. Viele Kommunikations- und Unterhaltungsdienste (z.B. Skype, Zattoo) verwenden ALM in der Realisierung. Mit den gestiegenen Hardware- und Netzwerkressourcen der kleinen elektronischen Geräte wie Smartphones und PDAs ist es nun möglich, ALM Lösungen auch auf Handhelds zu realisieren. Diese Arbeit entwickelt exemplarisch eine solche Anwendung für live Audio-Streaming auf dem iPhone und iPod Touch. Die Software erlaubt es dem Anwender, ein selbstorganisierendes P2P Overlay-Netzwerk auf den Geräten zu instantiiieren und dabei live verteilte Gruppen zu abonnieren oder zu beenden. Jeder Teilnehmer kann eine persönliche Gruppe zum Audio-Streaming erzeugen und darin z.B. in Echtzeit Karaoke Musik aufführen. Jeder andere Nutzer kann diesem Musikereignis beiwohnen, sobald er den zugehörigen Musikstrom abonniert.

Tran, Thanh Minh Tu

Title of the paper

Mobile P2P Audio Network for the iPhone and iPod Touch

Keywords

Peer-to-Peer Systems, Overlay Network, Application Layer Multicast Systems, Audio Live Streaming Technologies, iPhone Audio Processing

Abstract

Group communication offers a large application domain, e.g., voice and video conferencing, radio and television broadcasting, multiplayer gaming, etc. and has attracted much attention in commercial deployment, as well as in academic research projects. While IP Multicast minimizes the network traffic, but suffers from limited deployment, Application Layer Multicast offers an alternative solution for content distribution, and is operated on an overlay Peer-to-Peer network. Many communication and entertainment platforms (e.g., Skype,

Zattoo) use ALM to offer their services. With the increase of hardware resource, decrease in size, and the WLAN support in small electronic devices (e.g., SmartPhones, PDAs), it is now possible to deploy ALM on handheld devices. This work is an attempt to bring ALM for audio live streaming onto the iPhone and iPod Touch. The software allows users to self-organize an overlay P2P network, to subscribe or unsubscribe to a streaming group, to establish any personal streaming group for multicasting music or live performing karaoke music. Any user - when subscribing to this group - will be able to receive and listen to live stream karaoke music.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	General P2P Application Introduction	2
1.3	The PAN4i Application Introduction	3
1.4	Organization of the Report	3
2	Streaming Technologies on the Overlay	5
2.1	General Background	5
2.2	Overlay P2P Network	7
2.2.1	Unstructured P2P	8
2.2.2	Structured P2P - DHT-based approach	10
2.2.3	Pastry - a DHT-based P2P approach	14
2.3	The Dabek model for Structured P2P Overlays	20
2.4	Application Layer Multicast	22
2.5	P2P audio/video streaming	26
2.5.1	P2P live streaming	26
2.5.2	P2P video-on-demand	36
2.5.3	Base technologies	38
2.5.4	Scribe - a DHT-based ALM approach	40
2.6	P2P Streaming Systems	49
2.6.1	Skype	49
2.6.2	Zattoo	50
2.7	Conclusion and discussion	52
3	Audio Processing on the iPhone and iPod Touch	58
3.1	iPhone OS Technologies	58
3.2	Core Audio Overview	59
3.2.1	Audio Queue Services	60
3.2.2	Audio File Stream Services	61
3.2.3	Audio Unit Services	64
3.2.4	Audio Processing Graph Services	65
3.3	Audio codecs	66
4	PAN4i Concepts and Design	69
4.1	User requirements	69
4.2	Functional requirements	72
4.3	Software concepts	74
4.4	Application design	79

5 PAN4i Implementation	86
5.1 Pastry - KBR	86
5.2 Scribe - ALM	87
5.3 Application Layer	92
6 Testing	95
6.1 Test setup and running	95
6.2 PAN4i in the Nacht des Wissens 2009	97
7 Conclusion and Outlook	98
List of Figures	107
List of Tables	109
List of Source Code Snippets	110

1 Introduction

1.1 Motivation

Recently, people quickly adopt infotainment which is a combination between information-based media content and entertainment content. The purpose of this combination is to make a service more attractive in order to gain more customers or consumers. Based on this idea, there have been many software and hardware products or projects underdeveloped. There are examples for such applications, e.g., a car driving navigation system device having multimedia functions for radio and music, or the ongoing Mindstone project [1] for developing an e-learning application on the basis of social network for the mobile platform.

The Internet has been showing its powerful usage and popularity in many activities in daily life. Especially, the Internet plays an important role in the business, in communication and also in entertainment. With the increment of the Internet bandwidth, and supported hardware devices for wireless connection (Wireless LAN, Wimax), the Internet offers its users not only connectivity but also mobility. There are many success stories on the usage of Internet and especially in entertainment. Youtube [2], Spotify [3], Zattoo [4] are good examples.

While the traditional client-server model for large scale application requires much investment on the server establishment and maintenance, overlay Peer-to-Peer (P2P) network has proved that it is suitable for implementing large scale, stable application at low cost. Many overlay P2P approaches have been proposed and tested. Among them, Chord [5], Pastry [6], CAN [7] and Tapestry [8] approaches have been implemented as open source projects. This makes overlay P2P ready to be deployed for new networking applications.

Since Apple [9] has released the iPhone [10] and iPod Touch [11], its products have gained a huge success. Since the iPhone and iPod Touch share the same operating system and API, in the rest of this writing, the term iPhone means for both iPhone and iPod Touch. Starting from the iPhone first release day on June 29, 2007, it is said that about 10 million devices have been sold out to the market. Over one billion applications have been downloaded within 9 months [12]. The iPhone supports W-LAN and is not only a communication device, it is also an entertainment and information organizer. These abilities together with its nice design make it a good choice for those who would like to own a cell-phone. Besides, Apple Inc. opens the iPhone's API to third party developers. Applications developed by the third parties can be put on the iTunes store and developers will get 70% of sales revenue from their sold applications. The iPhone and its business model have been attracting a large number of third party companies as well as freelancer developers to create a huge amount and rich content for the iPhone.

Inspired from what the iPhone can do and from the wonderful entertainment arts that YouTube, Zattoo and Spotify bring, a group of students from the Information Engineering course at the HAW Hamburg University have initially started the Mobiles Video-Network Hamburg project [13]. The idea was to develop an overlay P2P network for sharing live

video streams on the iPhone and iPod Touch via the W-LAN. This idea has won a support from the Ditze Foundation. Later on, the project has not aimed at sharing live video stream content since there was not an adequate support for live video stream on the iPhone at that time. And the project was continuing as a final thesis project with the title Peer To Peer Audio Streaming For The iPhone And iPod Touch. It is to develop a P2P audio streaming application to form a communication and entertainment platform via the W-LAN for the iPhone and iPod Touch. And this report is done on the basic work of this thesis project.

1.2 General P2P Application Introduction

The very beginning concept of the P2P network could be found in IP Router or Domain Name Server in which these devices exchange routing information. This collaboration between devices forms a distributed system in which each device contributes its knowledge (routing information) and resource (CPU, memory, network bandwidth) to let the network routing functioning correctly and effectively. But until the birth of Napster P2P application for music sharing between its users in 1999, P2P architecture has been much considered in academic research projects as well as development in commercial business model.

At first, P2P applications were developed for desktop computers or laptops, those having enough resource for a P2P application to run on. These applications have been used mostly for file sharing such as Napster, Gnutella, FastTrack, BitTorrent and Transmission. Since the Internet connection has been increasing its availability with higher bandwidth, P2P architecture is considered for audio and video streaming such as Skype, PPLive and Zattoo. Furthermore, the increase of hardware resource in combination with decrease in size, and more widespread wireless network accessibility have allowed small electronic devices to provide enough resources to run such softwares.

Besides its traditional software for Desktop, Skype [14] (a Voice over Internet Protocol (VoIP) software) has recently released Skype for iPhone, Windows Mobile, Android, Symbian and claimed to support Blackberry soon. Another example is the Moviecast project from the HAW Hamburg. And as stated on its project page [15], "*the aim of the project Moviecast is the design and development of an Internet based videoconference solution for mobile devices*".

Other attempts try to deploy P2P file sharing application onto the iPhone. One of the first attempt which is claimed to be the first P2P torrent client on the iPhone was the migration of the native open source P2P torrent Transmission onto the iPhone [16]. Unfortunately, the original project page could not be accessed anymore at the time of this report writing. Another one is the iSlisk - Native Souseek P2P sharing client for iPhone which is able to access the P2P Souseek file sharing network [17]. Although there are several implementations of P2P application on small electronic devices, the number of such implementations is very small as compared to the traditional desktop P2P application. And this project is another attempt to push the P2P audio streaming application on the iPhone, iPod Touch and later may also be extended to other different mobile platforms.

1.3 The PAN4i Application Introduction

Mobile P2P Audio Network for the iPhone and iPod Touch ([PAN4i](#)) is a networking application which was initially planned to have two main functions. It can be used for voice chatting and this serves as a communicating function. The other one is serves as an entertaining function by letting its users to receive a karaoke music stream. Any user can be a singer for this karaoke background music and be able to send his or her beautiful voice to the overlay network created by PAN4i users.

At first, a new user would like to join the PAN4i overlay P2P network. This joining node is assigned a key. This unique key is used to distinguish each different node on the overlay network. It joins the network by sending a JOIN request to any known node already existing on the network. The new coming node will then receive information about other nodes in the network whose keys are closest to the new node's key. These information is used for routing messages within the overlay network. When a node is in the overlay network, it can do the following things. It can create a new overlay multicast group (later in this report, the term group represents an overlay multicast group). It can join an existing group. It can send messages, voice audio packets to an existing group. When a node is in a group, it will receive all message sent to this group. By this way, nodes in one group can send voice audio packets to each others to form a voice chatting group. Also, a node can send music audio packets to one group i.e. sending an MP3 background music stream. The receiver can sing on this background music stream and send this new mixed karaoke stream to another group. The member of this "another" group can listen to the live performing karaoke stream. The rest of this report will discuss on the different techniques that help to implement the PAN4i, e.g., how to build an overlay P2P network, how to create, join or leave a group, how to send, receive and playback the network audio stream, how to manage the multicast tree when nodes leave their group gracefully or unexpectedly. Other solution choices, challenges, possible improvement and future work of this application will also be discussed.

1.4 Organization of the Report

The remainder of this work is organized as follows:

Chapter 2 discusses different technologies that can be used to develop an audio/video streaming Application Layer Multicast ([ALM](#)) software built on an overlay P2P network. The key concepts of these technologies are generally discussed, in which the technologies used in developing the PAN4i (i.e., the Pastry overlay and Scribe Application Layer Multicast approaches) are discussed in more detail. The discussion ranges from a set of different approaches for creating an overlay P2P network, to a set of different streaming topologies built on an overlay that are classified into two different streaming application domains namely the live streaming and the video-on-demand streaming. Besides, other based technologies for media streaming, and the Dabek model for implementing P2P systems on the Structured

P2P overlays are also mentioned. Chapter 2 is closed with an introduction to two different commercial streaming systems (Skype and Zattoo), and a conclusion and discussion section on the discussed technologies.

Chapter 3 is an introduction to the Core Audio Application Programming Interface (API) of the iPhone Operating System (OS) which is used to program the audio processing functional unit of the PAN4i. Core Audio API provides many interfaces for different audio processing purposes. This chapter does only focus on the functions and technical usage of interfaces that are needed to implement the required audio processes of PAN4i, i.e., playback, recording, mixing, streaming, and encoding.

Chapter 4 provides information on the software concepts and design strategies of the PAN4i. It starts with the user requirement analysis. Coming up from the user requirements, the functional requirements of the software are then figured out. From these requirements combining with the related technology know-hows, the software concepts describing the selected design model, all the functional units with their dependences, and communication protocol for sender and receiver are defined. Lastly, the application design, which is about the design strategies including application logic and Graphical User Interface (GUI) design, that can bring all the defined functional units into a working mechanism. And so that the software can offer the services as described in the application use cases.

Chapter 5 is a report on the current implementing state of PAN4i. The implementation exactly follows the described software concept and application design strategies in chapter 4. This chapter reports on the functional units that have been implemented and those that are left for further implementation. It also introduces to the open source projects (i.e., Chimera [18], oRTP [19]) that are employed in the software. The already implemented units are demonstrated with source code snippets and explanation. The other units for future work, together with the implementing approaches are also discussed.

Although the PAN4i has not been fully implemented that it can provide full functionalities as described in the software concepts and design, the current state of PAN4i can provide key functionalities for music, karaoke streaming and receiving. These key functionalities of PAN4i have been displayed to the visitors of the Nacht des Wissens 2009 (i.e., Night of Sciences 2009) at the Hamburg University of Applied Sciences. In this chapter 6, we would like to describe the procedure that we setup the PAN4i system for running a test as we did and shown to the guest on that event.

Lastly, chapter 7 concludes the work of this thesis. This chapter describes our target to run this PAN4i project, about what we want to do, what we have achieved, how we have achieved, the current state of the implementation, what can be done in the future for improvement, how would the application look like and used for when all features would have been implemented.

2 Streaming Technologies on the Overlay

2.1 General Background

Streaming technologies involve two main techniques. The first one is the building of streaming network topologies namely overlay P2P network described in section 2.2 (later on in this report, the term P2P means overlay P2P). The second one is the multicast streaming techniques implemented upon their network topologies namely ALM described in section 2.4. But first of all, we would like to draw out the general picture of different types of streaming network topology which have been proposed. Some have been out of date, some are widely implemented and some are recently proposed for new implementations.

Since there is a need for group communication, such as voice and video conference, radio or television broadcasting or multiplayer gaming etc., there is a need to have solutions for this. A solution can be a sender is transmitting each content message to each receiver (unicast) which shown in figure 1. This method is not scalable for processing time at sender, more delay at receivers, and consumes more network bandwidth. Another choice is to use broadcast method at sender (figure 2). This method is also not sufficient, because unwanted node will also receive the content message from the sender and it is a waste of network resource. This makes multicast (figure 3) a choice and led to the proposal of IP Multicast described in RFC 1112 [20] from the Internet Engineering Task Force.

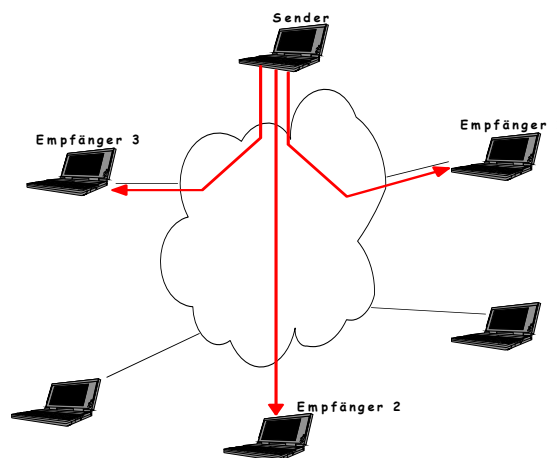


Figure 1: Unicast (source [21])

IP Multicast allows one or more senders to send data to a group of receivers, whereby the sender sends only one copy of data out to the network. When this data is passed through any routers which support IP Multicast, these routers will replicate this data to each receivers belonging in its domain network. Although this method has its advantages such as effec-

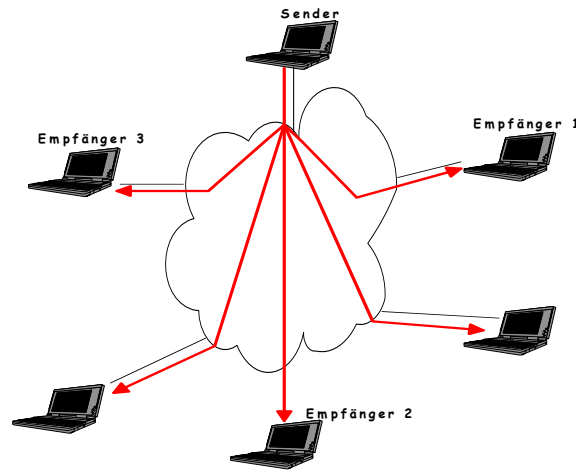


Figure 2: Broadcast (source [21])

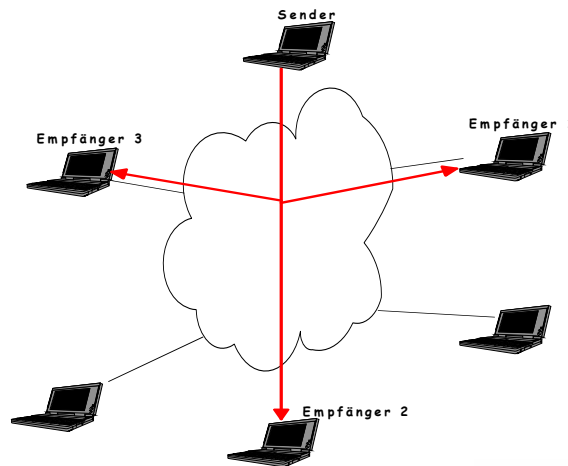


Figure 3: Multicast (source [21])

tively minimizing the network traffic, reducing loading burden for the network and server, but its deployment issues make it not everywhere feasible. In order to setup an infrastructure for IP Multicast, it requires router capabilities, maintenance and solution for inter-domain multicast routing problem. Firstly, these requirements mean investment cost for ISPs or carriers. Secondly, there is a missing of a charging model for the large amount of traffic replicated within their infrastructure. These reasons are not incentive for ISPs or carriers to deploy IP Multicast. The first global deployment of this method was an experimental backbone for IP Multicast traffic across the Internet name Mbone (short for "multicast backbone"). Mbone is used mostly for the creation, exchange and viewing of multimedia (radio, television or video-conferencing).

As an alternative solution for IP Multicast, ALM was proposed. ALM is built upon an overlay P2P network topology. For each different overlay topology, there are different ALM approaches. Unlike IP Multicast which requires underlay network support, both ALM and P2P do not require support of routers in the underlay network, and can be entirely implemented on the application layer. This characteristic attracts research community as well as commercial implementers.

P2P models are not only used in multicast applications e.g. multiplayer gaming, IPTV etc., but also in distributed file sharing applications. Starting from the first P2P network model deployed in the Napster file sharing application, over the time, many other P2P models have been proposed. These models are divided into two categories, Unstructured P2P and Structured P2P. The Unstructured P2P has three variants namely Centralized P2P, Pure P2P and Hybrid P2P. Structured P2P has one variant, that is the Distributed Hash Table (DHT)-based approach. For the DHT-based model, there are several proposed protocols such as Chord, Pastry, Tapestry or CAN. The DHT-based P2P model will be discussed in section 2.2.2. Since Structured DHT Based - Pastry protocol is the selected overlay network topology for the PAN4i application, this model will be discussed in section 2.2.3.

ALM is built upon P2P network. There are many proposals for the ALM protocol. They are different from each one in their characteristics for both application requirements and networking point of view. These characteristics include application domain (i.e. live streaming, archive streaming or video on demand), overlay routing protocols, design choices such as mesh-base or tree-base approach, single source sender or multi-source senders, consideration in the tree depth versus the tree fan-out degree, etc. These properties are discussed in section 2.5. Among different proposals for the ALM protocols, Scribe [22] provides an ALM protocol for the Structured P2P - Pastry protocol and is used in the PAN4i application. Scribe will be discussed in more details in section 2.5.4.

2.2 Overlay P2P Network

The conceptual differences between the Client-Server model (Figure 4) and the P2P model are that Client-Server model runs the application service on one normally powerful machine called server and many machines requesting the services from the server called clients. The server is the central entity which only provides services. Different from this concept, each machine (called a peer or a node) in a P2P network can act as both, service provider and requester. Resources are shared and can be accessed directly between peers. Further more an overlay P2P network is a virtual signaling network built on top of the basic of the routing function in a P2P application at each peer. These are the features of the overlay P2P network.

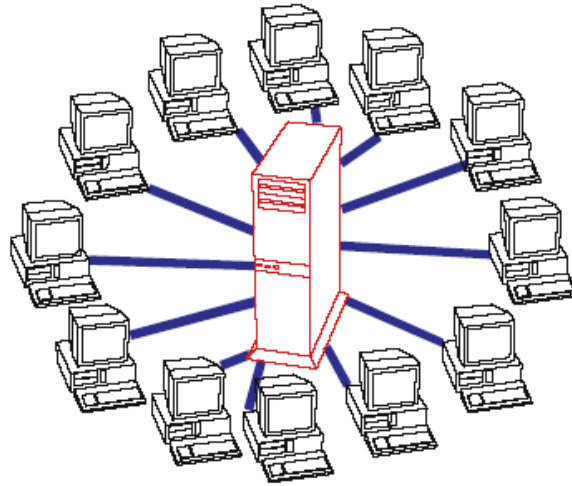


Figure 4: Client-Server Topology (source [23])

2.2.1 Unstructured P2P

Unstructured P2P has three variants: Centralized P2P, Pure P2P and Hybrid P2P

Centralized P2P Starting from the Centralized P2P (Figure 5) which was deployed for Napster in May 1999, this model has all features of P2P. It uses a central entity to provide file list indexing and signaling service i.e. registration, logging in, searching content etc. Peers upload their file list to Napster Server, and can query from here a provider list for requesting file. When a peer receives the provider list, it will communicate directly with the providing peers and get the file. This single central lookup server is technically a single point for failure and was a target for a lawsuit filed by the Recording Industry Association of America (RIAA) against Napster Inc. In July 2001, due to court decision, Napster had to close its central server and thereby its file sharing service.

Pure P2P Different from Centralized P2P, Pure P2P (Figure 6) which was used for Gnutella version 0.4, does not use any central entity. It has all features of P2P. A node connects to at least one active node in the P2P network. In order to look up for a file, it sends a search query to its neighbor nodes. These neighbor nodes forward the query in the same way. A Time To Live (TTL) field in the query message determines when the forwarding in the network stops (how many hops a message may be forwarded). When there is another node which has the queried file, it responds a message, and the message is routed back on the same way to the requester. The requester may receive different response messages, it will select the best answers and connect to the providing peers. Query forwarding with TTL constraint floods the

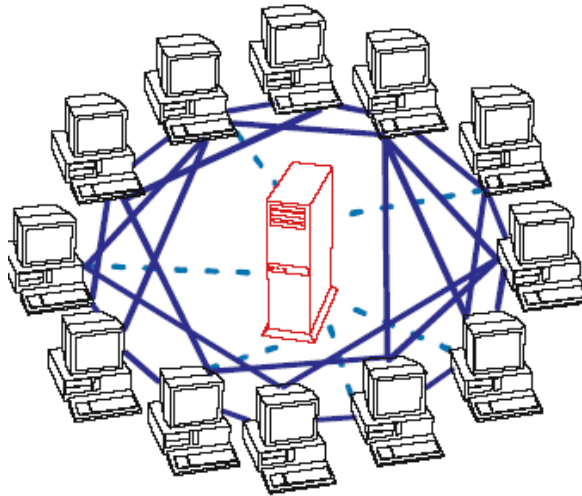


Figure 5: Centralized P2P Topology (source [23])

network and is restricted to a limited number of hops introduces scaling problems to this P2P model.

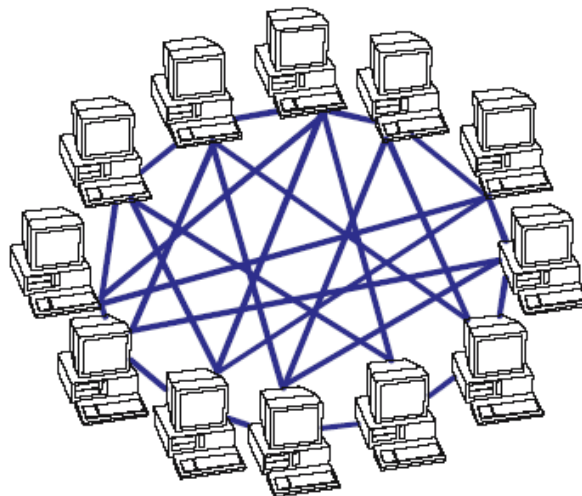


Figure 6: Pure P2P Topology (source [23])

Hybrid P2P To solve the scalability drawback of Pure P2P, Gnutella version 0.6 was deployed on the Hybrid P2P (Figure 7). This P2P model is a two tiers model - normal nodes and super nodes. Each normal node is connected to one of any super nodes and is called leaf node. This two tier model helps to improve scalability via signaling (routing, file list indexing

etc.) reduced to super nodes. A super node is elected among leaf nodes, the one which has more resources (memory, storage, processing power), high bandwidth connection, the uptime of a node etc. Leaf nodes announce their shared content to the super node that they are connected to. When a leaf node wants to look for a file, it sends a request to its super node. This super node forwards this request to other super nodes. If a super node has information for the requesting file shared by anyone of its leaf nodes, a response message is routed back to the requester. The requester will select the any best matched provider peer and get the file from there directly. Skype is also deployed on the Hybrid P2P model.

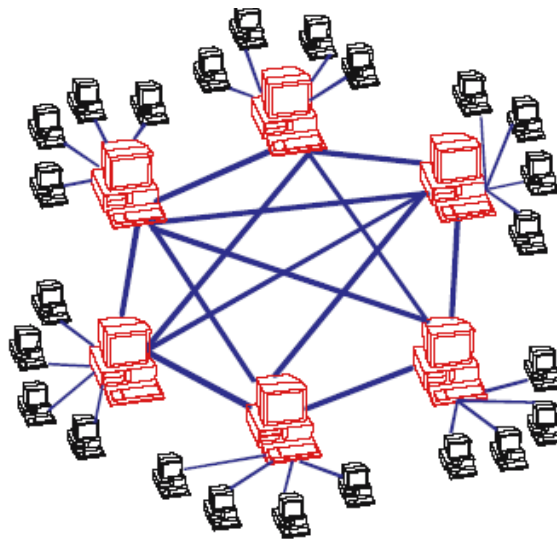


Figure 7: Hybrid P2P Topology (source [23])

Although Hybrid P2P has improved the Pure P2P much and is gained consideration in commercial implementation, it is a Pure P2P between islands of Centralized P2P systems. It enhances the scalability of both, however bears the limitations of each. Beside Hybrid P2P, research community has proposed a new approach which can bring its own, definite and predictable routing structure that has less signaling overhead, and also reduces the flexibility in choosing routes. It is Structured P2P - DHT-based P2P.

2.2.2 Structured P2P - DHT-based approach

A picture of a P2P network is that resources are located at different nodes. Any nodes can join or leave the network at anytime and the network can serve a large number of nodes. So that the challenges for a successful P2P network is to have an effective resource locating method. It is able to limit the complexity for communication and storage. It is robust, resilient over frequent arrivals and departure of nodes. For these requirements, the central server approach does not answer the robustness requirement for an increasing big number of nodes

since all signaling communications are assigned to a single point of failure. Centralized approach results in $O(1)$ flexible searches and $O(N)$ node states at server. Whereby flooding-based approaches have $O(1)$ nodes states but resulting not only in communication overhead $\geq O(N^2)$ but also limiting in search results.

For these challenges, a better P2P solution would have a compromise between both approaches. It is the balance of $O(\log N)$ communication overhead and $O(\log N)$ node states (Figure 8). This requirement turns out that Distributed Indexing in the form of **DHT** is one of the most suitable method. This approach is also called Structured P2P because of its definitive and predictable routing structure (proactive procedure).

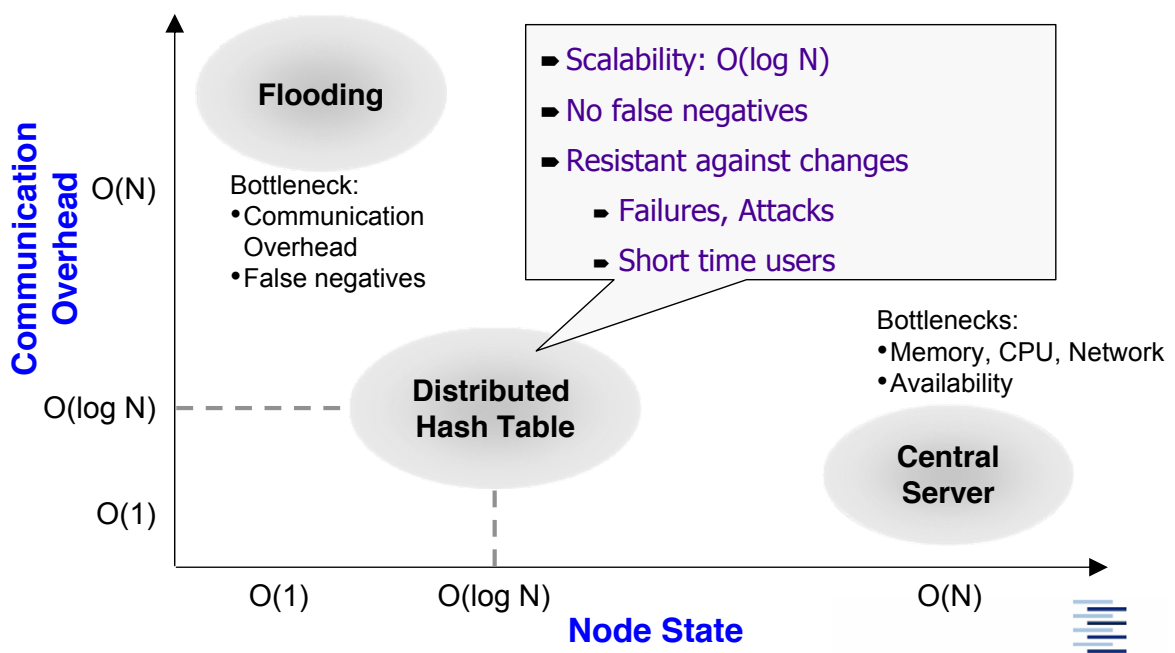
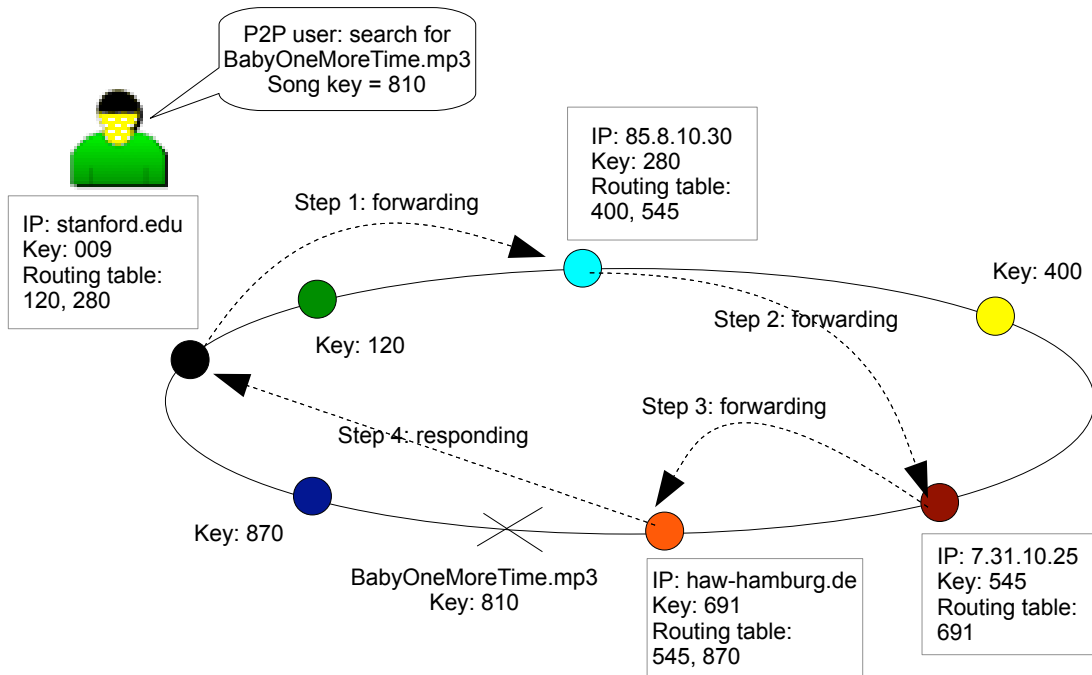


Figure 8: Complexity of different P2P models (source: [24])

Figure 9 demonstrates a resource locating scenario in a DHT-based overlay P2P network. Each node or each data item (stored on any node) has a unique identifier ID (key) value in the address space i.e. the range from 0 to $2^{160}-1$ of a P2P network. The calculation for this key can be done via a collision-resistant hash function such as SHA-1 [25]. The input parameter of this function for a peer node can be its IP:Port address and for a data item (i.e. a file) can be its string name. Each node manages one DHT having key:value pairs in form of identifier ID for key and IP:port for value. This DHT, called the routing table, is the knowledge information of the local node about other neighboring nodes. If the DHT-based P2P system is used for file sharing, each node may also manages another DHT having key:value pairs of all data items whose keys in the responded range of this node. This range can be, for example, defined as it is greater than the local node key and less than the next adjacent node key, e.g., the data item key range of node 009 is from 010 to 119 and key range of node 120 is from 121 to 279 and so on.

A key range of a node may be changed during its lifetime. Supposing that node 120 would have not joined in the P2P network, then the key range of node 009 would cover from 010 to 279. When a new node 120 joins the P2P network, then node 009 would have to update its key range from 010 to 119, and node 120 would take the range from 121 to 279. In contrast, a node 009 would enlarge its key range from 010 to 279 again if node 120 would leave the P2P network via departure or node failure. The joining and leaving (called the dynamic of DHT or peer churn) of nodes may happen frequently to an overlay P2P at any time. Structured P2P software has to deal with this problem to ensure its robustness, reliability and scalability.

The functionality of DHTs can be interpreted as routing systems or as storage systems. For the given destination keys, the first one focuses on the use of DHTs to route packets and deliver them to the destination nodes like for a multicast streaming system. The second one uses DHTs as distributed storage system like a P2P file sharing application. Routing is the core functionality of DHTs. There are different proposals for the DHT routing protocol. But the fundamental principle of this routing can be described in a scenario, where a user on node 009 wants to search for a data item, say, a MP3 song Baby One More Time from singer Britney Spears. At first, node 009 converts the song title to a key value (suppose that the result is 810). Then node 009 will send a lookup request to a node in its routing table whose key is numerically closest to the song key 810. And node 280 happens a closer one than node 120 for the lookup key 810. The request message is sent to node 280. Node 280 checks if key 810 belongs to its key range (from 281 to 399), if this is not the case, it will forward to the next closest key node in its routing table. This procedure is repeated on each forwarding node. Until the request message is forwarded to node 691 whose key range is from 692 to 869. Node 691 realizes that the song key 810 stays in its range. It then checks for the existence of this data item in its data item DHT, if there is, node 691 is the correct destination and it sends a response message to the requester-node 009. Data item 810 on node 691 can be a MP3 file or a pointer to this file stored in another node on the overlay.



Node locations on the overlay P2P network



Node locations on the underlay IP network

Figure 9: Use case scenario in DHT-based P2P

In conclusion, the above scenario shows the basic concepts of the DHT-based P2P approach. Besides, there are other proposals for improvement in dealing with peer churn¹ condition such as copying redundant key-value pairs to different nodes so that when one node fails, others can repair the content and the routing information held by the failing node. Another proposal is that each node holds locality and proximity² information about other nodes to improve the routing effort. Routing efficiency, communication overhead and dynamic of DHT are challenges for designing a DHT-based P2P application. Based on these fundamental concepts of the DHT-based P2P, there are a number of proposed protocols. Among of them are Chord, Pastry (described in section 2.2.3), CAN or Tapestry, those are widely supported, implemented and tested. Since properties for DHT-based P2P may conflict one another (like communication overhead vs. node states, complexity), each system has its own advantages and disadvantages. Each one targets its specific functionality goals.

Table 1 is to compare different properties (i.e. per node state, communication overhead, fuzzy queries³ and robustness) between different overlay P2P approaches. And figure 10 is to summarize the main features the different Overlay P2P approaches, those have been discussed up to this point.

System	Per Node State	Communication Overhead	Fuzzy Queries	Robustness
Central Server	$O(N)$	$O(1)$	yes	no
Flooding Search	$O(1)$	$\geq O(N^2)$	yes	yes
Distributed Hash Table	$O(\log N)$	$O(\log N)$	no	yes

Table 1: Comparison of central server, flooding search and distributed indexing (source: [25])

2.2.3 Pastry - a DHT-based P2P approach

In the rest of this section, we would like to introduce Pastry - a decentralized message routing and object locating protocol. Pastry is one of the proposals for the Structured (DHT-based) P2P approach, which is supported by a number of open source implementations, e.g., the Bamboo-DHT [26], the FreePastry [27], and the Chimera [18] projects fully or partly implement the Pastry protocol, whereby the OverSim project [28] offering the overlay simulation

¹Churn in the context of P2P network means the rate of node continuously join or leave the P2P network which is also called the dynamic behavior of the P2P network

²Proximity concerns the distance between the two peers e.g. the latency, the throughput and the ISP locality

³Fuzzy queries or fuzzy searches: The results of these searches do not need to be exactly match with the search arguments. But searches will return a result list in which the most argument matched results are on top of the list and then other likely matched results go after e.g. Google searching with keywords "Britney Spears" (with double quotes) will result in items having exactly the term keywords, but when searching with keywords Britney Spears (not within double quotes) will return a fuzzy searched results

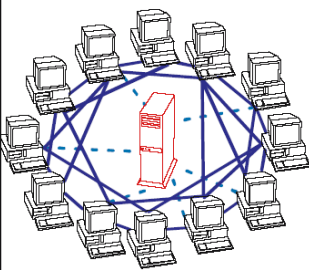
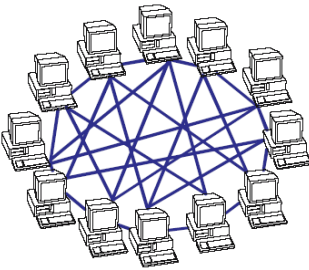
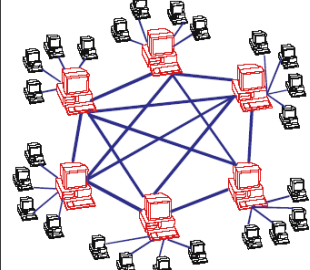
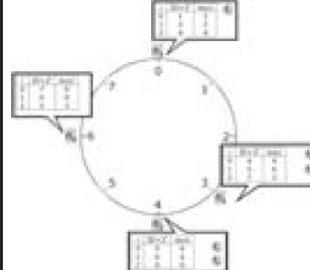
Peer-to-Peer			
1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept)			
Unstructured P2P			Structured P2P
1st Generation		2nd Generation	
<i>Centralized P2P</i>	<i>Pure P2P</i>	<i>Hybrid P2P</i>	<i>DHT-Based</i>
1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Example: Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities Examples: Gnutella 0.4, Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Example: Gnutella 0.6, JXTA	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" Examples: Chord, CAN
			

Figure 10: Comparison Table for Client-Server and different P2P models (source: [23])

framework for Pastry and others. We have used the routing scenario in figure 9 to describe the general concepts of the DHT-based approach. We will also use this scenario to describe the working concepts of the Pastry protocol in detail. A full description and evaluation of Pastry can be found in [6]. The Chimera project, which is a light-weight C implementation of the structured overlay that provides similar functionality as prefix-routing protocol of Pastry, is used in the PAN4i application.

According to Rowstron et al. [6], "Each node in the Pastry network has a unique identifier (*nodeId*). When presented with a message and a key, a Pastry node efficiently routes the message to the node with a *nodeId* that is numerically closest to the key, among all currently

live Pastry nodes. Each Pastry node keeps track of its immediate neighbors in the node ID space, and notifies applications of new node arrivals, node failures and recoveries. Pastry takes into account network locality; it seeks to minimize the distance messages travel, according to a scalar proximity metric like the number of IP routing hops. Pastry is completely decentralized, scalable, and self-organizing; it automatically adapts to the arrival, departure and failure of nodes."

A Pastry system is a self-organizing overlay network of nodes. In Pastry, each node or data item is uniquely assigned a *len-bit* identifier or a node ID or a key (*len* is normally selected to be 128). The node ID is used to indicate a node position in a circular node ID space, which ranges from 0 to $2^{128} - 1$. An ID is a string of digits to base 2^b where "*b*" is a configuration parameter with typically value 4. A key is located on a node to whose ID it is numerically closest. This key space concept is different to the key range of a node described in the scenario in figure 9 where all keys within the range of the local node ID to the next adjacent node ID are located on this local node. Figure 11 illustrates the identifier space concept in Pastry. This concept works in the way that key K03 is between nodes N01 and N10. But because key K03 is closer to node N10 than to node N01, so that key K03 is located on node N10. This principle is applies to the rest of keys shown on this figure.

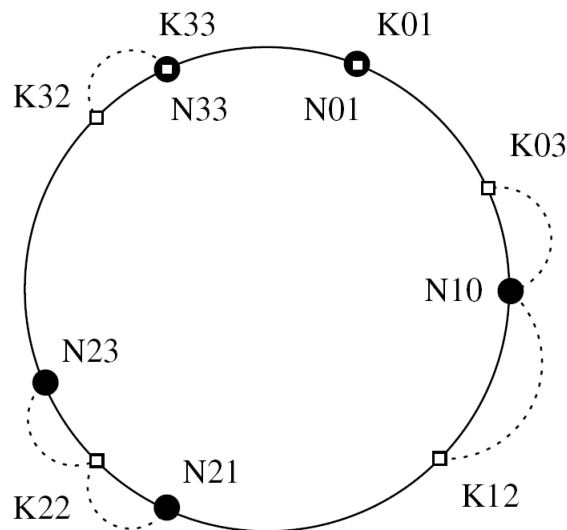


Figure 11: A 4-bit Pastry identifier space with six keys mapped onto five nodes. Numeric closeness is an ambiguous metric for assigning keys to nodes as illustrated for key K03 (source: [29])

The routing information (or node state) of a Pastry node is of type hash table having key (node or data item identifier) - value (IP-address:Port or pointer to data item) pair and is contained in three elements: the routing table, the leaf set and the neighborhood set. Figure

12 shows an example of the node state of a Pastry node. In these tables, only the keys are shown, the values (IP-address:Port or data item pointer) corresponding to the keys are not shown.

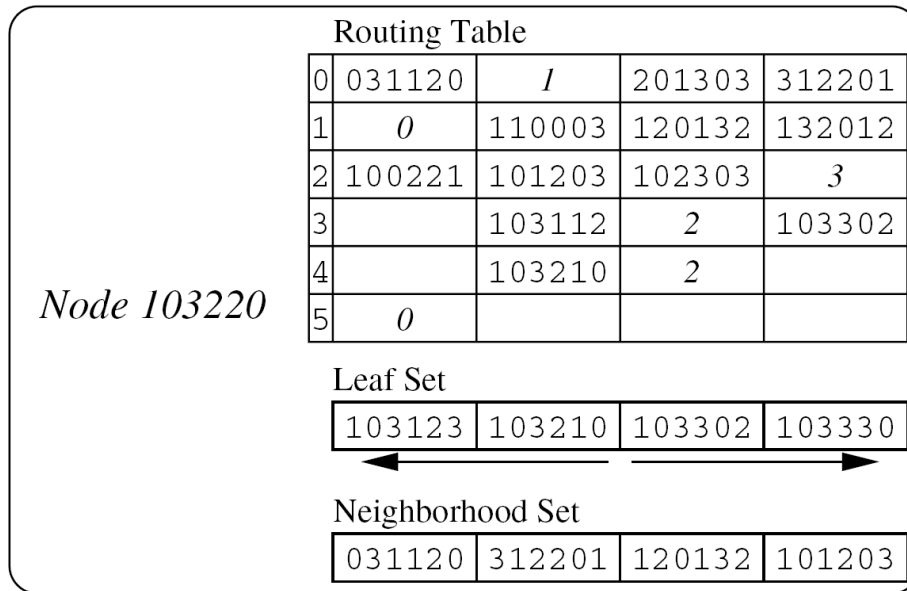


Figure 12: Pastry node state for node 103220 in a 12-bit identifier space and a base of 4 ($k = 12$, $b = 2$). The routing table lists nodes with the length of the common node identifier prefix corresponding to the row index. (source: [29])

Routing Table The routing table sorts node IDs by prefix. It is made up of $\frac{len}{b}$ rows and $2^b - 1$ entries for each row. Each entries in row i^{th} and the local node have the first i -digits prefix in common. For example, in the routing table in figure 12, the first column show the i^{th} row order. In the first row ($i = 0$), all entries have no common prefix with the local node 103220. In the second row ($i = 1$), they have one digit (the first digit) common prefix.

Leaf Set The leaf set holds L nodes whose IDs numerically closest to the local node ID. The use of the leaf set helps to enhance the routing efficiency, such that, if a message with a key is within the leaf set, then the closest key to the message key in the leaf set is the destination for this routing message. In this case, there is no need to perform a routing lookup in the routing table to forward the message to a next forwarding node.

Neighborhood Set Neighborhood set holds N nodes metrically closest to the local node with regard to the network proximity metric. The proximity metric reflects the distance

between any pair of nodes, e.g., the round trip time, number of hops etc. For this, a function is needed which allows each Pastry node to determine the distance between itself and another remote node. For more information on this, please refer to the locality property of Pastry from [6].

Routing Procedure Using the MP3 music lookup scenario in figure 9, the lookup query is routed from the sender to the destination node via forwarding nodes on the Pastry overlay. The routing decision at each node is made by the following two steps. Step 1: a node first checks whether the lookup key k belongs to its leaf set range (i.e., $\text{far-left-leaf-node-key} \leq \text{lookup-key-}k \leq \text{far-right-leaf-node-key}$). If it is the case, the destination node is the node in this leaf set whose key is numerically closest to the lookup key k . If the local node key turns out to be the closest one, this local node is the destination node which hosts the lookup key, and the lookup query finishes. Step 2: if key k does not belong to the leaf set range, the local node will find the next forwarding node in the routing table whose key is numerically closest to the key k . This lookup is done as follows. The local node check the number of digits in common prefix between its key and the lookup key k (supposed this number is c). The local node will then go to row c^{th} of the routing table, and try to pick a key on this row which has $c+1$ common prefix with this key k . This key stays at the column whose order is equal to the digit at the c position of key k . If there is no entry at this position on row c^{th} , the selected key is the right or left adjacent key to this empty entry which is numerically closest to key k .

During the life time of a Pastry node, each node will dynamically update its routing information via exchanging the routing information with its neighborhood nodes or on arrival or departure of nodes. The routing information is used for the routing procedure and for self-organizing (node arrival and departure or failure) of the overlay network which will be discussed in the following. With this routing fashion, for the Pastry overlay network of N nodes, Pastry can route to any node in less than $\lceil \log_2 b \rceil$.

Node Arrival When a new node having node ID X arrives in the Pastry overlay network, it needs to initiate its state tables. It does it by sending a JOIN message to a known node A which is nearby node X according to the proximity metric. The JOIN message with the key of X is then routed from node A to a destination node Z . In response to the receiving of this JOIN request, node X , node A , destination node Z and other forwarding nodes (e.g. F_1, F_2, \dots, F_n) on the routing path from A to Z send their state tables to the new node X . From these information, node X can construct its own state tables.

Specifically, node A sends to node X its neighborhood set. This is because A and X are proximity metrically nearby so that nodes in the neighborhood set of A are also nearby to node X . Node Z sends to X its leaf set. This is because the node ID of Z is numerically closest to node ID of X , so that the numerically closest keys to Z in node Z leaf set are also numerically closest to X . According to Rowstron et al. from [6], in order to construct

the routing table of node X , X takes the first row (row zero) of A to be its first row. (The explanation for this is because the entries in row zero of the routing table are independent of a node's node ID. So that row zero of A contains appropriate values for row zero of X)⁴. Each time the JOIN message is forwarded to the next forwarding node (F_1, F_2, \dots, F_n), the length of the common prefix of the key X and the next forwarding node also increases. So that row 1 of F_1 can be used for row 1 of X , row 2 of F_2 can be used for row 2 of X and so on. From these receiving information, the routing table can be constructed for node X .

Finally, node X sends its node state to all nodes in its state tables. These nodes can also update their own routing information accordingly.

Node Failure Failures of nodes in the routing table or in the leaf set are lazily detected. This means that during the routing, communication attempts of the local node with these nodes fail. Whereby, the local node periodically test the liveness of the nodes in its neighborhood set (active detection), because these nodes do not contribute into the routing process.

For replacing a failed node at entry i in row j of the routing table (R_j^i), the local node contacts a remote node whose key is on row i of its routing table. Since entries in the same row j of the remote node are valid to the local node, the entry (R_j^i) from the remote node is used to replace the failed entry of the local node. If this (R_j^i) is also failed, the local node will do the same steps with another remote node whose key is in row j . If it is still not valid, the attempts will continue with nodes on the preceding row R_{j-1} until a valid entry is found for replacing the failed entry in the routing table of the local node.

For replacing a failed node in the leaf set L , the local node contacts another remote node whose entry in the leaf set having the largest index (far left or far right) on the side of the failed node. The local node will then retrieve the remote leaf set L' . If this remote node is also failed, it contacts another entry with a smaller index in the leaf set. Since the entries in the local leaf set L and the remote L' are close to each other in the identifier space and overlap, the local node can select an appropriate entry for the replacement.

For replacing a failed node in the neighborhood set, the replacing procedure is similar to the one for the leaf set. But as mentioned, for node failure detection, the local node has to check the liveness of its entries in the neighborhood set periodically.

Node Departure The decision for finding an entry to replace a departed node is similar to the node failure case. The benefit of a graceful departure may help to prevent data loss, reduce communication overhead than in the node failure recovery case.

⁴Our comment: from the technical point of view, taking row zero of A for X is not always correct. To prove that it is not a perfect solution, let us consider node X has key 020231, and node A has key 103220 with its routing tables shown in figure 12. When X takes row zero of A to be its row zero, then the first item in this row has one common digit (0) with X key which must not stay in row zero of X . Secondly, the second item or this row is missing. In order to solve this row zero problem, we suggest that A can send X its row zero if A and X has common prefix, otherwise A will send X its row one if they have no common prefix

Pastry API In a simplified manner, Pastry exports the following operations:

- `nodeId = pastryInit(Credentials, Application)` causes the local node to join an existing Pastry network (or start a new one) and to initialize all relevant states
- `route(msg, key)` causes Pastry to route a given message to the node whose `nodeId` is numerically closest to the parameter `key`, among all live Pastry nodes.
- `send(msg, IP-addr)` causes Pastry to send a given message to the node with the specified IP address, if that node is live. The message is received by that node through the `deliver` method (described next)

Applications layered on top of Pastry have to export the following operations:

- `deliver(msg, key)` called by Pastry when a message arrives at the local node and the local node's `nodeId` is numerically closest to the message `key`, among all live nodes.
- `forward(msg, key, nextId)` called by Pastry just before a message is forwarded to the next node with `nodeId` is the `nextId` parameter. The application may change the contents of the message or the value of `nextId`. Setting the `nextId` to NULL terminates the message at the local node.
- `newLeafs(leafSet)` called by Pastry whenever there is a change in the local node's leaf set. This provides the application with an opportunity to adjust application-specific invariants based on the leaf set.

Pastry can be used to build P2P file sharing applications, or multicast streaming applications (e.g. Scribe) etc. Since Pastry and Scribe are used in the PAN4i application, Scribe is discussed in section [2.5.4](#).

2.3 The Dabek model for Structured P2P Overlays

Structured overlays can be used to construct services such as distributed hash tables (storage systems), routing systems (group communication multicast / anycast) and decentralized object locations. Since each structured overlay protocol might use different API and provides services with subtly different semantics, the result would be that applications would be locked into one system and unable to leverage innovations in other protocols. The work from Dabek et al. [30] was an attempt to identify the fundamental abstractions provided by structured overlays and to define API for the common services they provide, in particular the Key-based

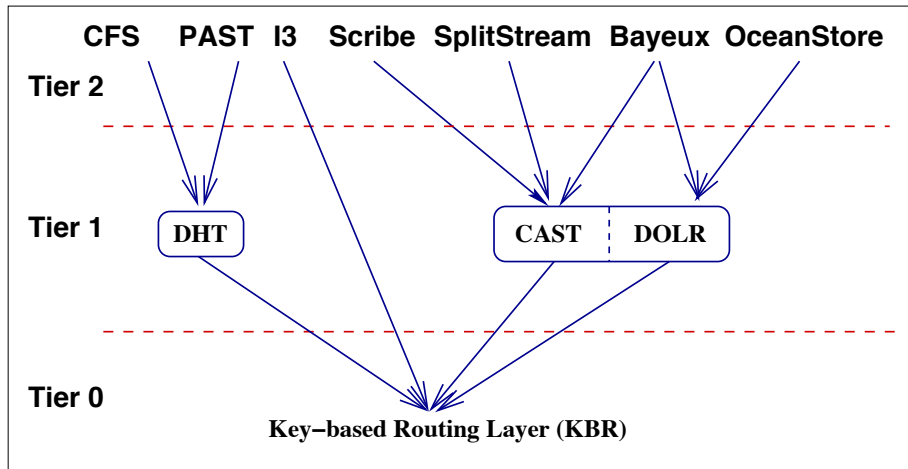


Figure 13: Basic abstractions and APIs, including Tier 1 interfaces: distributed hash tables (DHT), decentralized object location and routing (DOLR), and group anycast and multicast (CAST) - (source from [30])

Routing (**KBR**) functionality. This work resulted in a so called Dabek model. The key purpose of this model is to facilitate independent innovation in overlay protocols, services, and applications to allow direct experimental comparisons, and to encourage application development by third parties. The API also helps to accelerate the adoption of structured overlays and the implementation of any higher service layer. This concept is visualized in figure 13.

According to figure 13, KBR is the common service provided by all systems at tier 0. Tier 1 provides other higher level abstractions, those are specific for different applications, and higher level services stay on tier 2. In the Dabek model, only the KBR API at tier 0 is defined in detail as follows.

The common KBR API proposes two group of functions, the routing message and the routing state access. The following API⁵ and their explanation are taken from [30]. **Routing message functions** are used to route a message (or data packet) to destination nodes. They are

- `void route(key ->K, msg ->M, nodehandle ->hint)` *"This operation forwards a message M, towards the destination node of key K. The optional hint argument specifies a node that should be used as a first hop in routing the message."*
- `void forward(key <->K, msg <->M, nodehandle <->nextHopNode)` *"This upcall is invoked at each node that forwards message M, including the source"*

⁵In the following interfaces, a parameter p is denoted as ->p if it is a read-only parameter and <->p if it is a read-write parameter. And an ordered set p of objects of type T as T[] p.

node, and the destination node-owner key K (before deliver (described below) is invoked). The upcall informs the application that message M with key K is about to be forwarded to nextHopNode. The application may modify the M, K, or nextHopNode parameters or terminate the message by setting nextHopNode to NULL."

- `void deliver(key ->K, msg ->M)` *"This function is invoked on the the destination node owning key K upon the arrival of message M."*

Routing state access functions are used to access the routing state of a node in order to look up for instant a suitable next forwarding node (i.e. whose key is numerical closest to destination key). They are

- `nodehandle[] local_lookup(key ->K, int ->num, boolean ->safe)`
This call returns a number (num value) of nodes (i.e. whose keys are numerical closest to destination key K) that can be used as next forwarding node on a route towards destination key K.
- `nodehandle[] neighborSet(int ->num)` This call returns a number (num value) of nodes in the DHT routing table whose keys are numerical closest to the local node key and called neighbor nodes. It is to improve the routing efficiency (in Pastry, it is the leaf set).
- `nodehandle[] replicaSet(key ->k, int ->maxRank)` This call returns a number (maxRank value) nodes whose keys numerical closest to the data item k. This data item can be replicated and stored on these returned nodes. It is to improve scalability in case the local node fails.
- `update(nodehandle ->n, bool ->joined)` *"This up-call is invoked to inform the application node that node n has either joined or left the neighbor set of the local node."*
- `boolean range(nodehandle ->N, rank ->r, key <->lkey, key <->rkey)`
This call returns information about the responsible key range of node N which is a node in the neighbor set in the local node.

2.4 Application Layer Multicast

In section 2.1, we have discussed the problems of IP Multicast mainly in lacking of deployment support. As an attempt to overcome this drawback, ALM has been proposed. The concept of ALM is that the multicast functionality is implemented on an application service instead of using a network service as for IP Multicast. ALM systems can be deployed on any overlay topology which can support multicast packet routing mechanisms. As shown in

Issues	IP Multicast	ALM
Multicast efficiency in terms of delay/bandwidth	High	Low – Medium
Complexity or Overhead	Low	Medium – High
Ease of deployment	Low	Medium – High
The layer where the multicast protocol works	Network layer	Application layer

Table 2: Conceptual comparison of IP multicast and ALM (source [31])

figure 14 a) for a scenario of IP multicast, all packets from sender S are routed to destinations via underlay network routers R. Whereby in scenario b) for ALM, packets from sender S are routed on the overlay network to destination D4 via intermedia node D1. This requires each node to have knowledge about other neighboring nodes. These end-nodes with their knowledge form an overlay network which can be used for routing packets from sender S to any nodes in the overlay network. An application built upon this overlay network, that provides multicast functionality, is called ALM or End-system Multicast, or Overlay Multicast (i.e. overlay network is the topology for ALM). But ALM also has its drawbacks as compared to IP multicast such as multiple copies of the same packet may occur on the same links, routing paths are non-optimal resulting in longer jitter delay. Table 2 is a comparison between IP multicast and ALM based on different issues.

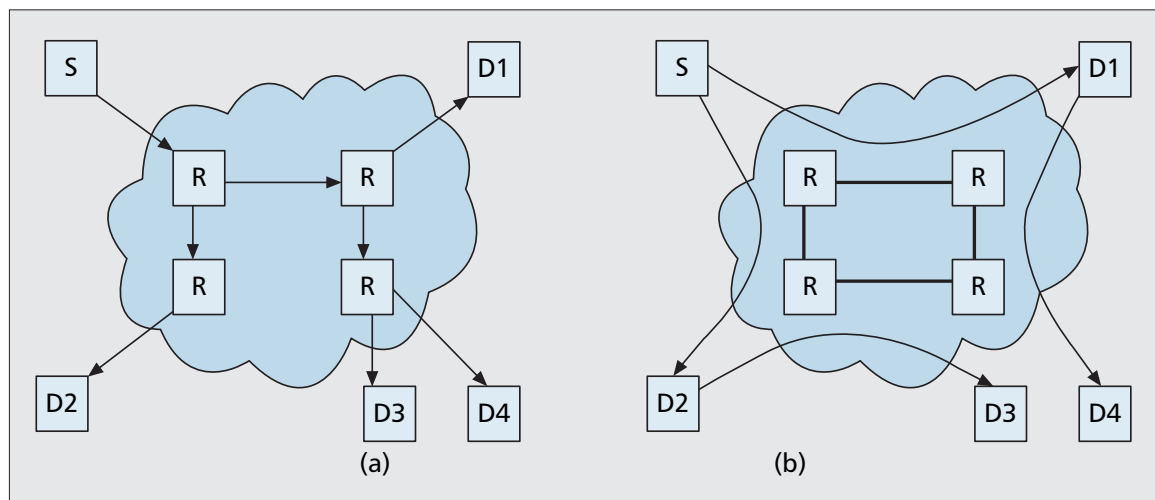


Figure 14: a) IP multicasting scenario and b) an overlay multicast tree (sender S, router R, destination D) (source [31])

There are a number of protocols designed for ALM. Each one is designed for each different application target. According to Hosseini et al. [31], things that have to be taken into account for designing an ALM protocol are the deployment level (proxy-based ALM or

end-system ALM), application domain (number of user, application functionalities, processing power, network resource, latency constraint, quality of services, nodes behind NAT etc.), the requirement for multicast group management and also the selected routing mechanism.

For the deployment level, the proxy-based approach requires servers/proxies on the Internet in a way they can form an overlay network and provide multicast service to end-user. This topology is somehow similar to the Hybrid P2P described in 2.2.1 where servers/proxies play the role of super-nodes. On the other hand, the end-system approach lets end-users organize themselves to form an overlay network for multicast service without the need of any server or proxy, e.g., solely using the P2P approach. Figure 15 shows a taxonomy of architectures for ALM.

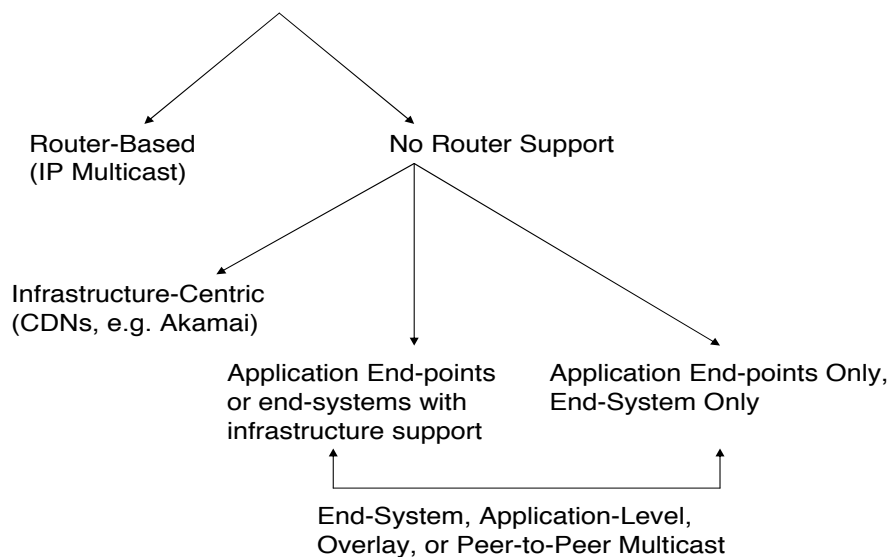


Figure 15: Taxonomy of architectures for ALM (source [32])

According to Diot et al. [33], application domains for multicast deployment are categorized as follows.

- *Audio/video streaming*: involves one source sending live or recorded audio/video such as streaming live sport event or video on demands. The typical consideration in this application domain are bandwidth and latency.
- *Push applications (information delivery)*: allows users to subscribe to any information channels. This information is then automatically pushed to them at regular time intervals. Bandwidth is in consideration.
- *Audio/video conferencing*: is similar to audio/video streaming but with smaller size

groups and higher interaction (multi-parties, multi-senders). The real-time effect, i.e., small latency is strictly required.

- *File transfer*: involves sending data (typically large amounts of data) (e.g. distributed object or file sharing). Bandwidth is in consideration.

Multicast group management involves the questions of how users find out about a multicast session, how they join or leave a session, and how they individually can create a multicast session etc. Furthermore, the question of whether multicast group is managed in a distributed or centralized manner. Nodes in a multicast group can be organized in a tree or a mesh topology. And when nodes join or leave the tree frequently, how to maintain the tree.

Different routing approaches between nodes on an overlay network satisfy different requirements such as minimum overlay delay or minimum worst case delay. The Shortest Path approach (figure 16b) aims to construct a minimum path from a source node to each receiver applying the Dijkstra's algorithm. This approach is for the best delivery effort strategy. The Minimum Spanning Tree approach (Figure 16c) aims to construct a tree with minimum total cost spanning all members which saves the cost of network resource. The Clustering Structure approach (Figure 17) aims to construct a hierarchical cluster of nodes. Each cluster has a head node and some sub-nodes. This helps to reduce control signaling overhead, and to manage the tree effectively.

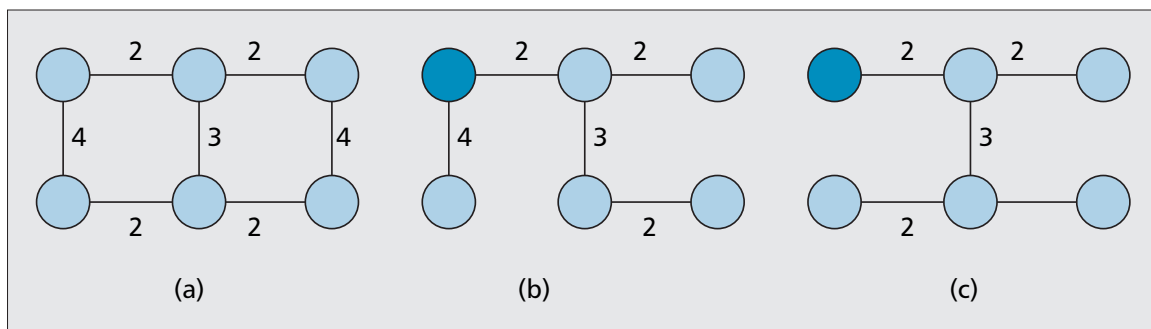


Figure 16: a) A graph with link costs; b) shortest path tree; c) minimum spanning tree (source [31])

Each approach listed in the above has its own advantages and disadvantages. The answer for which one to use depends on the objective and the application target. There have been many publications to propose solutions and improvements on any feature of the ALM systems. Therefore, in the scope of this report, not every approach will be discussed. Common features which are mainly considered in designing an ALM application will be discussed in the next section.

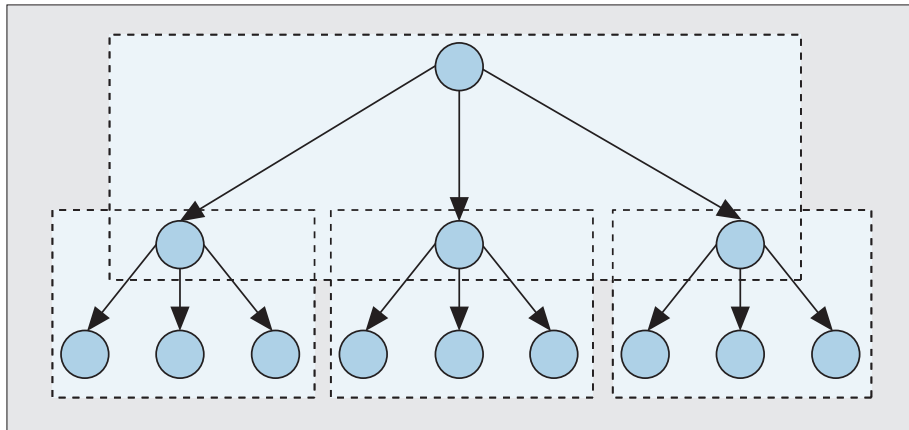


Figure 17: A hierarchical cluster of nodes with cluster size 4 (source [31])

2.5 P2P audio/video streaming

Audio/video streaming can be classified into two categories namely live streaming systems (e.g. sending live sport event) and video-on-demand systems (e.g. cinema at home service). All subscribers of a live streaming system receive the same stream at almost the same time (with a very small delay). Like watching a television channel, any subscriber can not go forward or backward to get the past streamed content. Whereby video-on-demand service allows users to watch any point of video at any time. Users can go backward or forward the video content at any time. Both multicast streaming systems can be built on a Tree-first or a Mesh-first approach. The main difference of the on-demand streaming from the live streaming is the caching technique used in the on-demand streaming system. In this context, the live streaming technology which is used to deploy in the PAN4i application is discussed in detail, on-demand streaming will only be discussed in brief.

2.5.1 P2P live streaming

In a live streaming session such as a live sport event, live audio/video content is distributed to all users synchronically and in realtime. ALM technologies can be used to provide live streaming service on an overlay P2P topology. In a P2P live streaming system, subscribers will first join an existing overlay P2P network on which the streaming service is based. There are mainly two approaches for building a P2P streaming multicast network namely Tree-first and Mesh-first approaches. Each one has its own advantages and disadvantages. They are discussed in the following.

A) Tree-first approach ALM tree-first approach organizes all participating peers into a hierarchical structure with the peering relationship of parent node (header node) and children

node (sub node). A parent node has one or many children nodes. A child node of a parent node can have other children nodes or have no child node. A node has no child node is called the leaf node. This hierarchical structure forms a tree structure and is used for delivering media stream from the top header node (also called the root of the tree) downward to all nodes connected on the tree. The construction and maintenance of a tree can be done in a centralized or decentralized approach. The Tree-based approach has two variations namely the Single-tree and Multi-tree.

A.1) Single-tree streaming In this approach, there is only one unique stream flowing from the top down to the bottom. All peers at all levels should continuously receive the same stream content one after another. When a new node joins a multicast tree, it is added to an existing node at a certain level of the tree and becomes a child node of this parent node. A sender can send media stream to this multicast tree by sending packets to the top header node on top of the tree (the root node). From this root node, the media stream is pushed to the rest of the multicast tree. Tree-based structure is easy to construct but the largest disadvantage of this approach is the scenarios of inner node failure. When a node fails or leaves the multicast tree, especially those higher in the tree, it will temporarily disrupt the video stream to all nodes in the sub-tree rooted at this departed node. The term temporarily here means that the tree has to be repaired and the disrupted sub-tree can continue to receive the stream as soon as possible. Other disadvantages are the delay in replication load if the tree has many levels. For reducing the delay in lower level nodes, each parent can host more children nodes (high fan-out degree). And in a tree with high fan-out degree, there is a large number of node in the leaf level that do not contribute their uploading bandwidth to the overall streaming system. Figure 18 shows an example of nodes on the Single-tree multicast streaming network with the corresponding locations of nodes in the underlay network. For a given sender, a stream is sent to the root of the tree. The root forwards each receiving packet to its children (nodes 1 and 2), nodes 1 and 2 continue to forward to their children nodes 3, 4, 5 and 6. Nodes 3, 4 and 6 forward to nodes 7, 8, 9 and 10. Since nodes 5, 7, 8, 9, 10 have no child to forward to, they are called the leaf nodes of the tree.

For the tree construction, there are two major considerations. They are the depth degree of the tree and the fan-out degree of the internal nodes (the possible number of supported children of a node). A node in a lower level receives the stream after the node at its upper level. The more level a tree has, the longer delay of nodes at bottom level have. To reduce this delay, a streaming tree would have fewest levels possible. This means that the tree should grow up horizontally and not vertically. For doing this, each node should be able to support as many children nodes as possible. The uploading bandwidth of a node will define the number of children it can support.

Another important issue in the Tree-based streaming approach is the tree maintenance. The tree needs to be repaired in case of any intermediate node leaves the tree either grace-

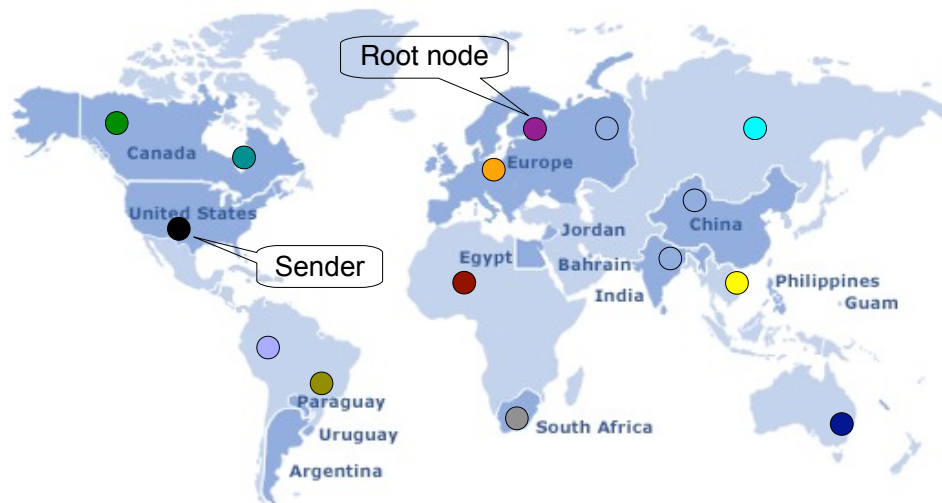
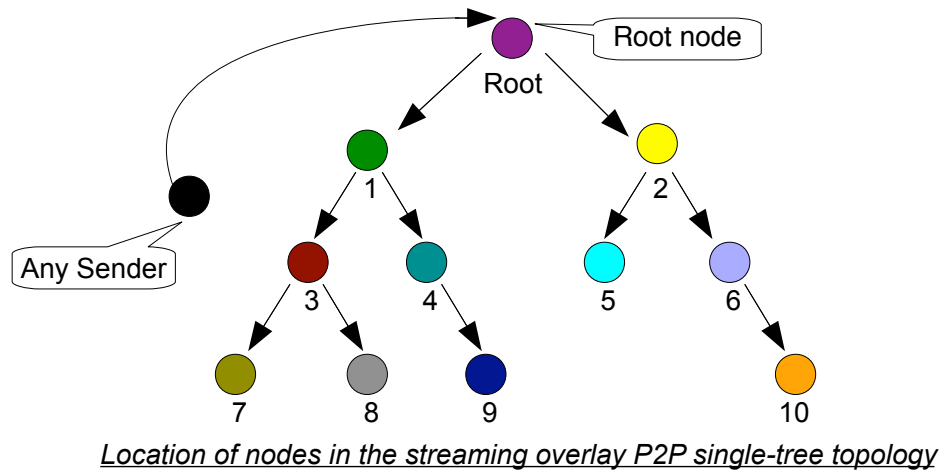


Figure 18: Nodes of a multicast streaming group in a single-tree

fully or unexpectedly e.g. machine crashes. When a parent node leaves the tree, all descendant nodes are disconnected to the streaming tree and could not receive the stream anymore. This would result in disruption during playback. The tree needs to be recovered as soon as possible. Figure 19 shows a scenario where node 1 leaves the tree. All descendant nodes 3, 4, 7, 8 and 9 are disconnected from the tree. The tree could possibly be repaired as node 3 is connected to the root node, and nodes 4 and 9 are connected to node 5 (Figure 20).

Tree management (i.e. construction and maintenance) can be done either in a centralized fashion or de-centralized fashion. According to Liu et al. [34], in the centralized fashion, all

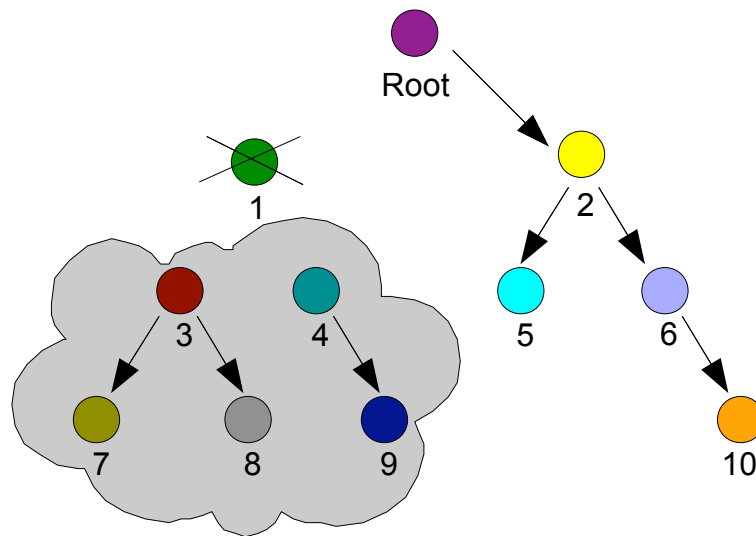


Figure 19: Scenario when a node leaves the multicast tree

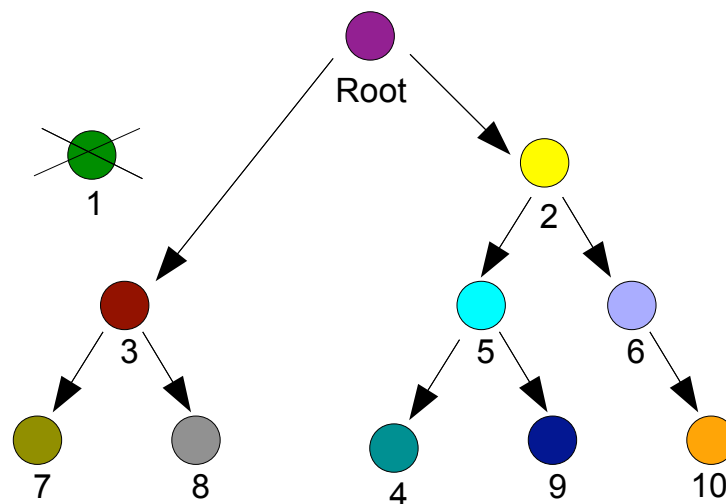


Figure 20: Repaired tree after a node leaves

signaling communication of peers on the tree is done at a central server. The server has full knowledge about the tree structure and the status of all nodes. When a new peer wants to join a multicast group, it contacts the server. With the knowledge (e.g. bandwidth, number of children, locality etc.) about each node on the tree and based on the characteristics of the new coming node, such as network access and location, the server will look for a suitable parent node for this new node. The new node would then be asked to connect to the suggested parent node directly. An example is that a selected parent node is close to the

new node in locality e.g. in the same sub network or the parent node has more bandwidth available, or delay between new node and suggested parent is small etc. On the other hand, the server can detect a departure of a node through either a graceful LEAVE message or after a time-out period without receiving any heart-beat message from this node. In both cases, the server will calculate a new tree structure for the remaining peers and instruct them to form a new tree structure. For a large scale streaming system, this centralized approach with its central server is the single point of failure.

To overcome this problem, distributed tree management has been proposed. Distributed tree management means that no central server is needed. All peers in the tree take part into the management. Depending on different schemes, each one has a different approach. An approach can be that each peer maintains relationships with some other peers called neighbor peers. These peering relationships form an overlay network for routing the signaling messages to any peers on the overlay. Based on the knowledge (e.g. bandwidth of each node, locality, delay between nodes etc.) of the existing overlay network, another streaming topology (e.g. shorted path tree or minimum spanning tree etc.) for content delivery (e.g. media streaming) is built on the existing overlay network. Scribe (discussed in section 2.5.4) is an example of this approach.

To solve the non-contribution of uploading bandwidth of leaf nodes in a Single-tree problem (especially for a high fan-out degree tree) which would degrade the utilization of bandwidth efficiency in the system, a Multi-tree approach is proposed.

A.2) Multi-tree streaming The Multi-tree approach is designed to make use of the uploading bandwidth of all leaf nodes for the streaming system. In order to do so, the sender will divide the stream into a number of sub-streams. For each sub-stream, there is a corresponding sub-tree. All nodes of a multicast group will have to join in all different sub-trees at different levels (intermediate or leaf level) in order to receive all the sub-streams and then re-order them into a full stream for playing back. When a node stays in an intermedia level of the tree, it receives the stream from its parent and then upload the received stream to its children. If a node has more bandwidth capacity, it could be placed more often at the intermedia level on different sub-trees than other nodes those having less bandwidth capacity. This is to utilize the available bandwidth of all peer in a tree to best effort. According to Liu et al. [34], given a fully balanced Multi-tree with m sub-streams, the node degree⁶ of each sub-tree is m . In the m sub-trees, a single peer stays at an intermediate level in only one sub-tree and upload the received sub-stream to only m children connected to it in this sub-tree. In the rest of $m-1$ sub-trees, this peer stays only at the leaf levels and receives other sub-streams from its parents. Figure 21 illustrates an example of a multi-tree topology. In this tree, there are one sender and seven receiver peers. The sender splits the stream into two sub-streams, each one is sent to its corresponding sub-tree. Each of the seven peers stays in both sub-

⁶Node degree: the number of children connected directly to this node

trees. Except peer 7, which stays at the leaf level on both sub-trees, all other peers stay at the intermedia levels on one sub-tree and at the leaf levels on another. Peer 7 does not contribute its bandwidth for uploading since the bandwidth of the sender is used instead.

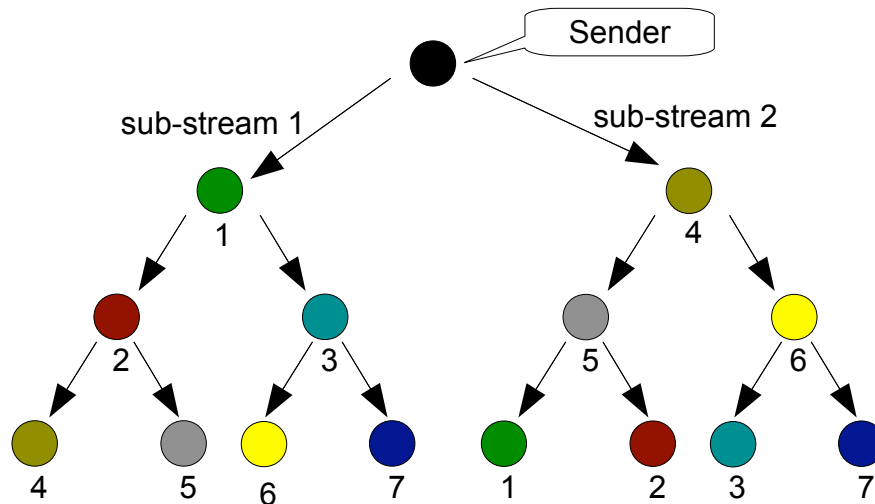


Figure 21: An example of multi-tree streaming topology

The Multi-tree approach solves the issue of bandwidth efficiency for the Single-tree approach. But it also introduces more complexity and communication overhead for the tree management. In general, the Tree-based approach gives direct control over the tree and the functionalities of each peer are well defined. This makes the Tree-based approach fairly straight forward to be implemented. Although multicast on a tree topology is claimed to have an efficient delivery effort, but jitter delay is also an issued to be considered. Under peer churn, both of the Tree-based approaches still do not solve the question of robustness for minimum disruption. Therefore, another Mesh-based approach was proposed and is discussed in the next section.

B) Mesh-based approach In the Mesh-based system, each peer maintains relationships with neighboring peers to form a connected overlay called a mesh. There are several proposals for a streaming topology established upon a mesh which mainly focus on the following two main approaches. A streaming tree topology is built upon an existing mesh overlay e.g. the shortest path spanning tree known as the Narada's protocol [35]. Another one uses the mesh overlay as for the streaming topology which means that the stream can arrive at a destination node from different neighbor nodes of this destination node [34].

B.1) Delivery tree in Mesh-based approach The Narada protocol [35] constructs and maintains a source rooted (reverse) shortest path spanning trees on a mesh. Narada al-

lows more trees to be built on the same mesh. Each one is used for a multicast session and is rooted at the corresponding source. Narada design involves three main components. Mesh management is the first component to ensure the mesh remaining connected in face of churn. The second one is the mesh optimization which uses distributed heuristics for ensuring shortest path delay between peers in the mesh is small. For a given poor mesh overlay topology in figure 22a, the paths between each peer pair in the mesh are dynamically added or removed according to the following optimization rules. Each peer periodically probes other peers at random and monitor existing links. An existing link is dropped if the Cost of dropping a link⁷ is less than a Drop Threshold⁸ value (Figure 22b). A new link between two peers is added if the Utility Gain⁹ of adding link is greater than an Add Threshold value (Figure 22c and d). The last component is about the spanning tree construction. The tree is built on the improved mesh (Figure 22e and f) and its construction's decision is based on the distance vector routing and reverse path forwarding algorithms¹⁰.

B.2) Multiple delivery paths in Mesh-based approach Unlike the multicast tree in the Narada Mesh-based protocol where the mesh is considered as an overlay for another streaming tree topology to be built on, different approaches allow the stream delivery via multiple paths to a destination node e.g the ChunkySpread [36] approach which forms a multi-tree over a mesh. In the paper [34], the mesh of this type forms the multicast streaming topology. In general, each peer at any given time maintains relationships with multiple selected peers. In case of peer churn, a peer can still be connected with the streaming session via connections with other neighbors. At the same time, a peer can learn about new peers and would change its neighbors to better suitable peers for improving its playback performance and to ensure that it always has a certain number of connected neighbors. With these relationships, a peer can upload to or download media data from its neighbors. For such a dense peering relationship, this Mesh-based approach is extremely robust against peer churn. According to Liu et al. [34], there are two main components for dealing with the Multi-tree Mesh-based system design namely the mesh formation and maintenance, and the data exchange.

B.2.1) Mesh formation and maintenance The Mesh-based streaming system uses a tracker¹¹ to manage all active peers in the streaming session. Upon arrival, a new node con-

⁷Cost of dropping a link is based on the number of peers to which routing delay increases for either neighbor [35]

⁸Add/Drop Thresholds are functions of the peer's estimation of group size and the current and maximum degree of member in the mesh [35]

⁹Utility Gain of adding a link is based on the number of peers to which routing delay improves and how significant the improvement in delay to each peer is [35]

¹⁰Distance vector routing and reverse path forwarding protocols are both used in IP Multicast

¹¹The use of a tracker follows the idea of using tracker in the file sharing systems like BitTorrent

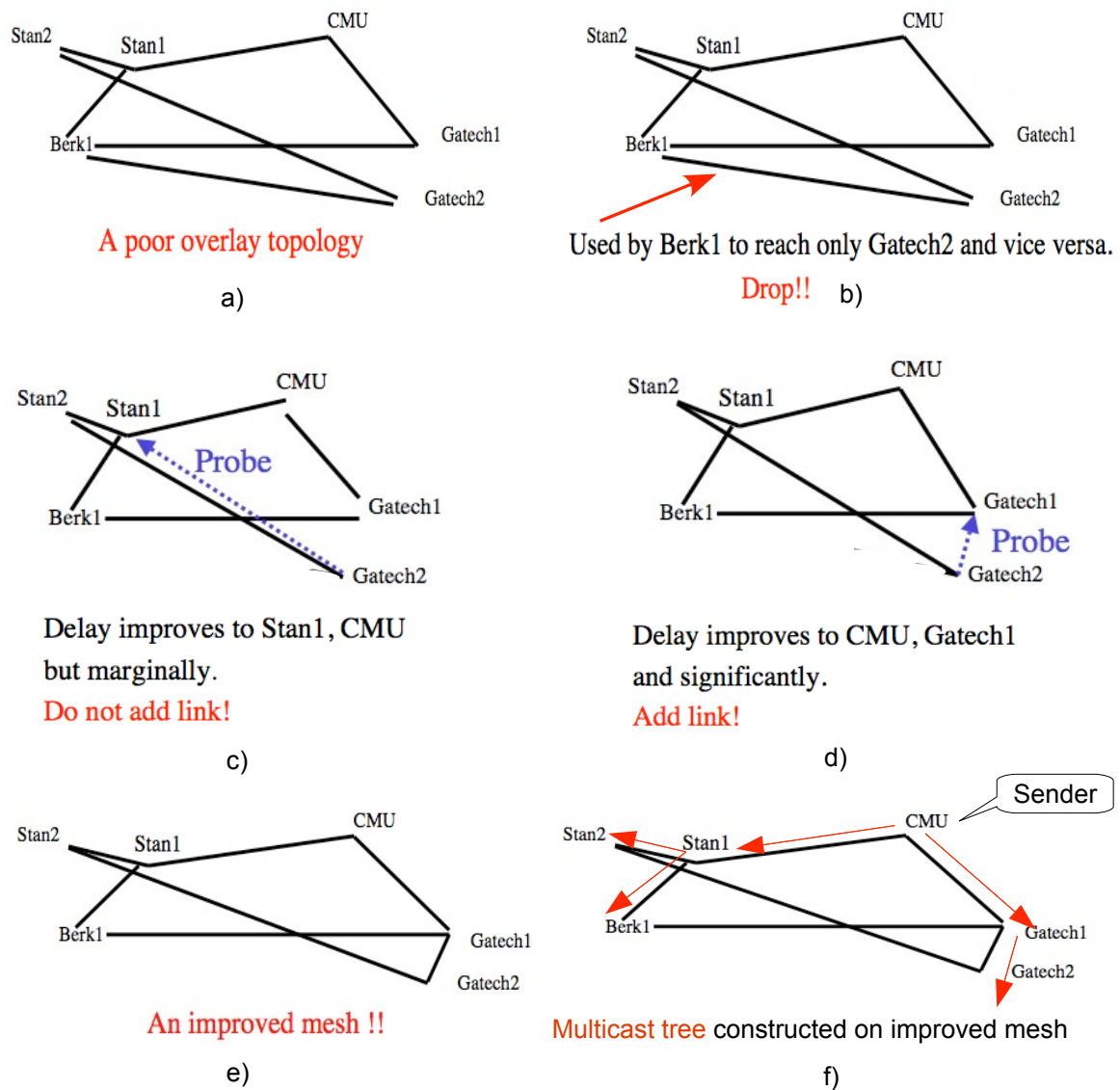


Figure 22: Narada - Optimizing Mesh Quality (source [35])

tacts the tracker to provide the tracker about its information e.g. IP address, port number etc. and to receive a suggested list of peers being in the streaming session. The new node will then try to connect with the remote nodes in the suggested list. If a remote node accepts the connection request, this remote node is added to the neighbor list of the new node. Figure 23 illustrates a scenario for the arrival of a new node to an existing streaming mesh. After having enough neighbors, the new node can start to exchange media content with its neighbors. During a streaming session, a node dynamically updates its neighbors by adding new nodes to replace departed nodes, changing its neighbors to get better neighbors for achieving bet-

ter performance. At any time, a node in the mesh should always maintain a certain number of neighbors keeping it robust against peer churn. The mechanism for updating neighbors can be done in two ways. A node can ask the tracker for a list of new suggested neighbors. Or it can also find new neighbors by exchanging neighbor information between nodes in the mesh. In case of graceful departure, a node will send a LEAVE message to the tracker and its neighbors so that its information is removed from these receivers immediately. Since each node periodically sends heart-beat messages to all its neighbors, in case of its unexpected departure e.g., computer crash, neighbor nodes will detect and remove the crashed node after a certain time without receiving any heart-beat message from this crashing node.

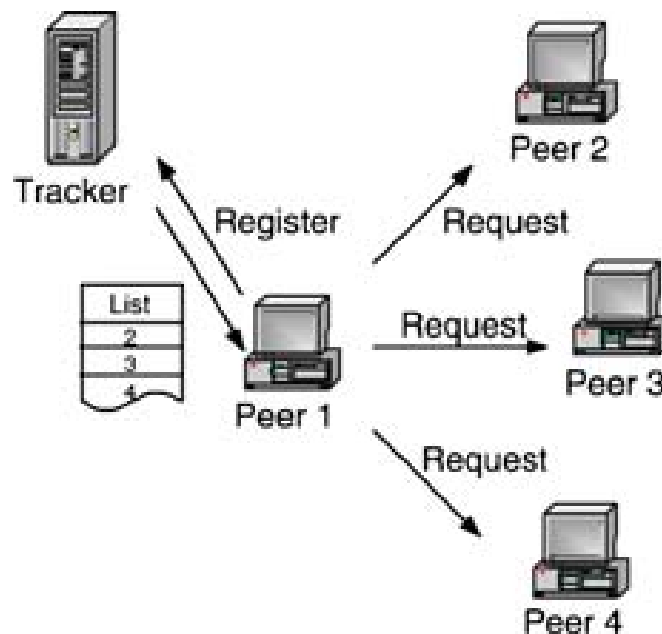


Figure 23: Peer list retrieval from the tracker server (source [34])

According to Liu et al. [34], a peering connection is established based on the mutual agreement between two peers at both ends. The decision on which peers to connect to, when and how often to exchange neighbors etc. is different for different protocols. Basically, a decision is based on the following considerations.

- The resource availability e.g. CPU, memory, uploading and downloading bandwidth etc. on both ends.
- The current condition of the network e.g. packet delay, packet loss characteristics on the network path between both ends.
- The content availability i.e. how often a remote peer having content needed by the local peer.

B.2.2) Data exchange This mesh-based approach has the ability to effectively utilize the uploading bandwidth of all participating peers as the group size grows. The data unit for the streaming is media chunk. Each chunk contains media content for a small time interval e.g. 0.1 second. Each chunk has unique time-stamp and sequence number. Since chunks may arrive at receivers out of order, these values are used to re-order chunks for playback. Unlike the Tree-based approach where content is pushed from the sender and flows down to all peers in the tree, Mesh-based approach normally couples push content reporting with pull content requesting. Each node has a storage buffer for the latest received trunks and a buffer map of available trunks in the buffer. Pushing content reporting means that a node will send its buffer map to its neighbors when there are new chunks available. And pulling content requesting means that when a node receives these buffer map advertisements, depending on its packet scheduling algorithm, it will pull the needed chunks from its neighbors to perform its playback. Figure 24 illustrates a scenario where buffer maps are exchanged and needed chunks are pulled between peers.

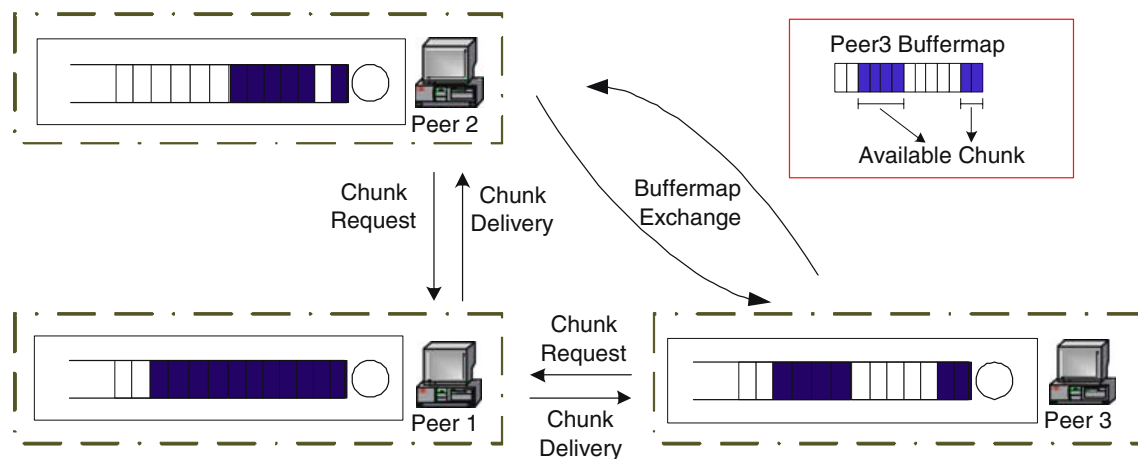


Figure 24: Buffer map exchange and data pull among peers (source [34])

According to Magharei et al. [37] *"The packet scheduling algorithm is the key component of a Mesh-based P2P streaming mechanism that would achieve the following three design goals: (i) effectively utilizing the available bandwidth from all parents peers, (ii) pulling a proper number of description (i.e., desired quality) from all parent peers, and (iii) ensuring in-time delivery of requested packets"*.

The dynamic peering relationship characteristic of the Mesh-based approach helps to solve the question of robustness against peer churn problem of the Tree-based approach. But the Mesh-based approach has to pay a price for its communication overhead, i.e., communication for maintaining relationship with neighbors and buffer map advertising. Another drawback feature of the Mesh-based approach is that data chunks are pulled in-orderly from different neighbors to a peer. And this makes the video distribution efficiency unpredictable

which would result in playback quality degradation such as low video bit rates or frequent playback freezes.

2.5.2 P2P video-on-demand

Besides the live streaming system, P2P video-on-demand (VoD) is another streaming system which uses different techniques. VoD allows users to watch video content at any point and at any time. This section provides an introduction with key components to the streaming techniques for P2P VoD services. It does not provide a deep discussion to the techniques as in the P2P live streaming section.

Similar to the P2P live streaming system, the purpose of the P2P VoD streaming system is to organize peers into a multicast topology for making use of the uploading bandwidth of all peers with best effort. Unlike the live streaming system where all peers playing the media stream synchronously, peers in the VoD system play the media stream asynchronously (i.e. different users may watch different portions of the same video stream at any given time). This results in different streaming topologies and strategies as compared to the live streaming system. According to Liu et al. [34], the following three solutions have been developed to support VoD using tree-based and mesh-based P2P system.

Tree-based P2P VoD The basic idea is that users are grouped into a session tree, based on their arrival time. A session is defined for a time period. If the arrival time of users is in this session time period, these users together with the streaming server are grouped into an ALM tree. The server forwards the base stream¹² continuously during time to each of the session tree connected to it. Each peer in the tree receives the stream from its parent and stores the past received content for a time length equals to the session time period. On arrival of a new node to an existing session tree, the new node is connected to an existing node and receives the based stream (synchronously flowed from the server) from this parent node. And at the same time, this new node is also connected to another existing node on this streaming session tree and receive the asynchronous patch stream from it. Patch stream is the archived media content from the beginning time of this session to the new node's arrival time. Figure 25 visualizes the concepts of this streaming tree approach.

Cache-and-relay P2P VoD This approach also employs a Tree-based topology for distributing VoD content. Each node keeps the past received content for a length of some time e.g., 10 minutes. But the difference is that this approach does not organize peers into ALM trees based on per timing session. Instead, new arrival nodes would join an existing tree at a leaf node if the 10 minutes buffer length of this leaf node has the archived video content

¹²Base stream is the stream continuously flow from the streaming server

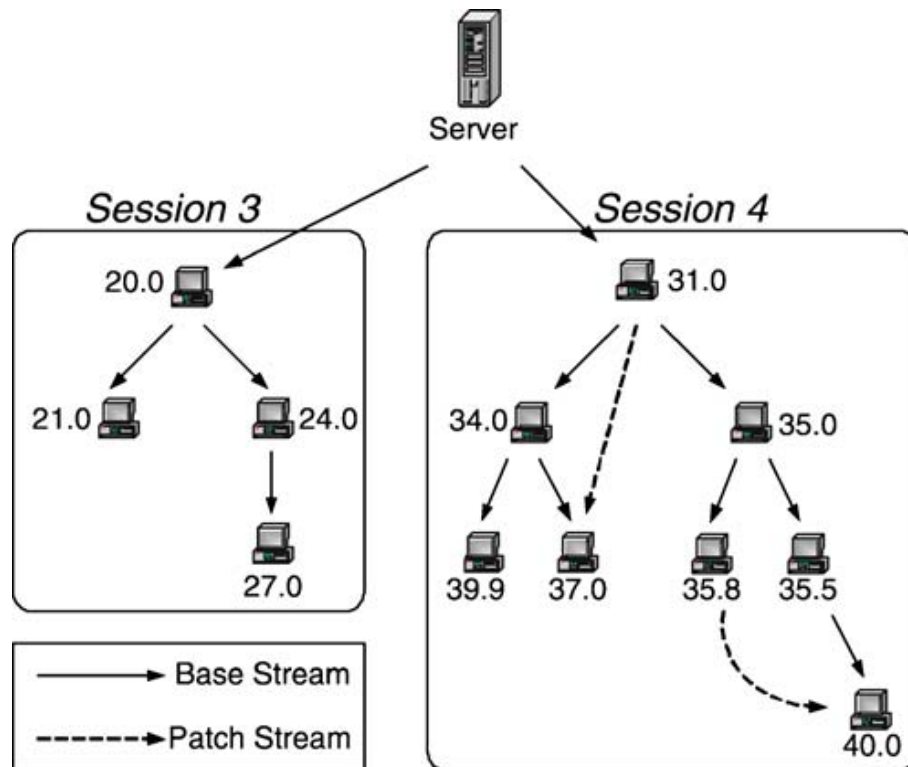


Figure 25: A snapshot of the scheme at time 40. Users belonging to the same session form an application-level multicast tree together with the server. Users in session 3 have finished patch retrieval; while 3 clients in session 4 are still receiving the patch stream from their parent patch servers (source: [34])

starting from the beginning of the video session. Otherwise, the new node will have to connect to the server directly to get the video stream from the beginning. Figure 26a shows a scenario where peer A arrives at time 1 and receives the stream directly from the server. At time 3 and 8, peers B and C arrive and are asked to connect to A. At time 3, A has archived video buffer for the last 2 minutes (because it arrived at time 1) starting from the beginning of the video session. A can then send to B the archived buffer starting from the beginning of the video stream, while still receiving continuing stream from the server. This concept is repeated so that peers C, D, E can be connected to the same tree. When F arrives at time 50, the oldest archived video buffer is in E (starting from time 24 to 34). No archived video starting from the beginning at time 0 is available in the existing tree. Then node F has to connect to the server directly to get the the stream from the beginning. Figure 26b shows that A has received to the end of the stream from the server and also has sent all its archived video in the buffer to its children B and C. A now plays no role in the streaming tree while B and C still serve to stream all content in their buffers to their children.

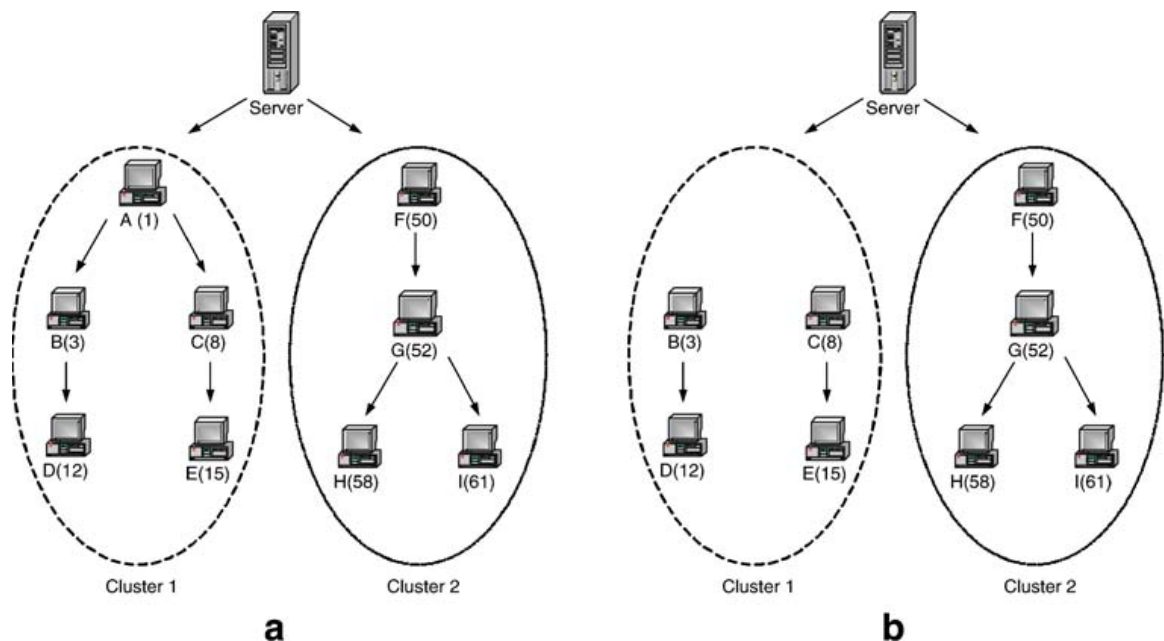


Figure 26: DirectStream system. a) DirectStream system with two clusters – one headed by client A and the other headed by client F. b) DirectStream system after the departure of client A. No service from the server is required from now on (source: [34])

Mesh-based P2P VoD Very similar to the concept of the Mesh-based approach for the live streaming system, the Mesh-based for VoD also works on the concept of advertising buffer maps between neighbor nodes and pulling missing packets in the local node from its neighbor nodes. The only difference is that VoD users are watching different part of video so that they need to pull packets lying at different part of the stream.

In both Tree-based and Cache-and-relay approaches, peer churn problem remains to be the key issue. In Mesh-based approach, communication overhead for managing the mesh overlay, and how to optimally allocate resource across different parts of the video are the key issues.

2.5.3 Base technologies

Besides the streaming topologies, other base technologies are considered in development of a P2P streaming system.

RTP and RTCP In order to re-construct all the receiving audio/video packets for playback, a receiver needs to get not only the media content of the packets, but also other information about the packets for real-time processing at receivers. These information is about the

sequence number of a packet used for packet re-ordering, session identification used for distinguishing from which sender, media encoding e.g. frame type, layers etc. These attributes are not available in a UDP packet profile. This has led to the proposal of the Real-time Transport Protocol (RTP) from H. Schulzrinne et al. [38]. RTP is used for sending audio/video stream. A media content packet can be encapsulated together with other real-time properties within an RTP packet. This RTP packet is then encapsulated within a UDP and IP packets for sending out to the network. Normally RTP is used together with RTP Control Protocol (RTCP) which is defined in RFC-3550 [38]. RTCP provides feedback by a periodic transmission of control packets to all members of a RTP session. The feedback reports on receivers about reception quality, packet loss, delay, jitter or faults to diagnose network problems. RTCP facilitates flow control and multicast session surveillance. These reports help to adapt the streaming to network conditions and session members. As an example, the playback speed should be equal to the packet arrival rate at a receiver. When a media player consumes more packets than arriving packets that will result in playback interruption. Vice versa, when there are more arriving packets than the amount of packet that a player consumes, those overloading packets will get lost if the buffer has not enough space. In this case, the receiver would like to send to the sender RTCP packets and reports that the sender should slow down the transmission speed to its playback rate value. For privacy issues, there is secure real-time protocol profile (SRTP) which is defined in RFC 3711 [39] and provides encryption for RTP packets.

Codec Raw video and audio streams are expensive for the network's bandwidth. Before sending to the network, the raw media stream is normally compressed to reduce its data size at sender. The received compressed stream at receiver can not be play immediately, it has to be decompressed to the raw data format for playing back. To do this, an encoder is needed at the sender and a decoder at the receiver. Not only used for compression, encoder can also be used for encryption, e.g., Skype provides encryption service for its audio stream to ensure secure and privacy for its media content. There are two compression types namely lossy and lossless compressions. A multimedia player e.g. QuickTime Player, Windows Media Player, VLC Player etc. can have one or more codecs to play different audio/video formats. For audio compression format e.g. MP3[40], Speex[41] etc., properties of an audio format include the sample rate, bit rate (i.e. sample rate times bits per sample), constant bit rate (CBR)¹³ or variable bit rate (VBR)¹⁴, number of channels etc.

Table 3 shows that different codecs are implemented for different application targets. The audio Wav format is used for high quality CD music, and has less data reduction. MP3 format for music can save much space as compared to Wav. The principle concept of MP3 codec works in a way that *"Parts of the music which are well-perceived are represented very*

¹³in CBR, all packets have the same size including silent packets

¹⁴in VBR format, size of audio packets are different, e.g., no audio packet for silence

precisely, while other parts that are not very audible can be represented with lower accuracy. Inaudible information will be discarded" [40]. For VoIP application, the requirements for low network bandwidth transmission and hearable quality for human voice lead to higher compression formats such as Speex or iLBC [42].

Audio Compression Format	Algorithm	Sample Rate	Bit Rate	CBR or VBR	Application
Wav	no compression	44.1 kHz et al.	1411.2 kbps et al.	CBR	CD music
MP3	Lossy	44.1 kHz et al.	128 kbps et al.	VBR	music
Speex	Lossy	8 to 32 kHz	2.15 to 24.6 kbps (NB ¹⁵) 4 to 44.2 kbps (WB ¹⁶)	VBR	VoIP

Table 3: Audio codec comparison

2.5.4 Scribe - a DHT-based ALM approach

Scribe is a scalable Structured DHT-based ALM protocol built on top of Pastry (discussed in section 2.2.3). Scribe organizes nodes on the Pastry overlay to form a streaming Single-tree topology. Scribe offers a de-centralized approach for the tree management. Figure 27 illustrates a Scribe streaming tree built on the Pastry overlay network. Any Scribe node may create a streaming group; other nodes can join the group, or multicast messages (e.g. media content) to all members of the group. Any node can create, send messages to, and join many groups. A group may have multiple multicast source senders and many members. Scribe claims to provide best-effort delivery of multicast messages, and can support simultaneously a large number of group with a wide range of group size, and a high rate of membership turnover. Scribe is the selected ALM approach used in the PAN4i application.

Scribe offers a simple API to its applications:

- `create(credentials, groupId)` creates a streaming group with identifier `groupId`; the `credentials` are used for access control.
- `join(credentials, groupId, messageHandler)` causes the local node to join the group `groupId`; all subsequently received multicast messages for that group are passed to `messageHandler` which is normally a callback function.
- `leave(credentials, groupId)` causes the local node to leave the group with `groupId`.

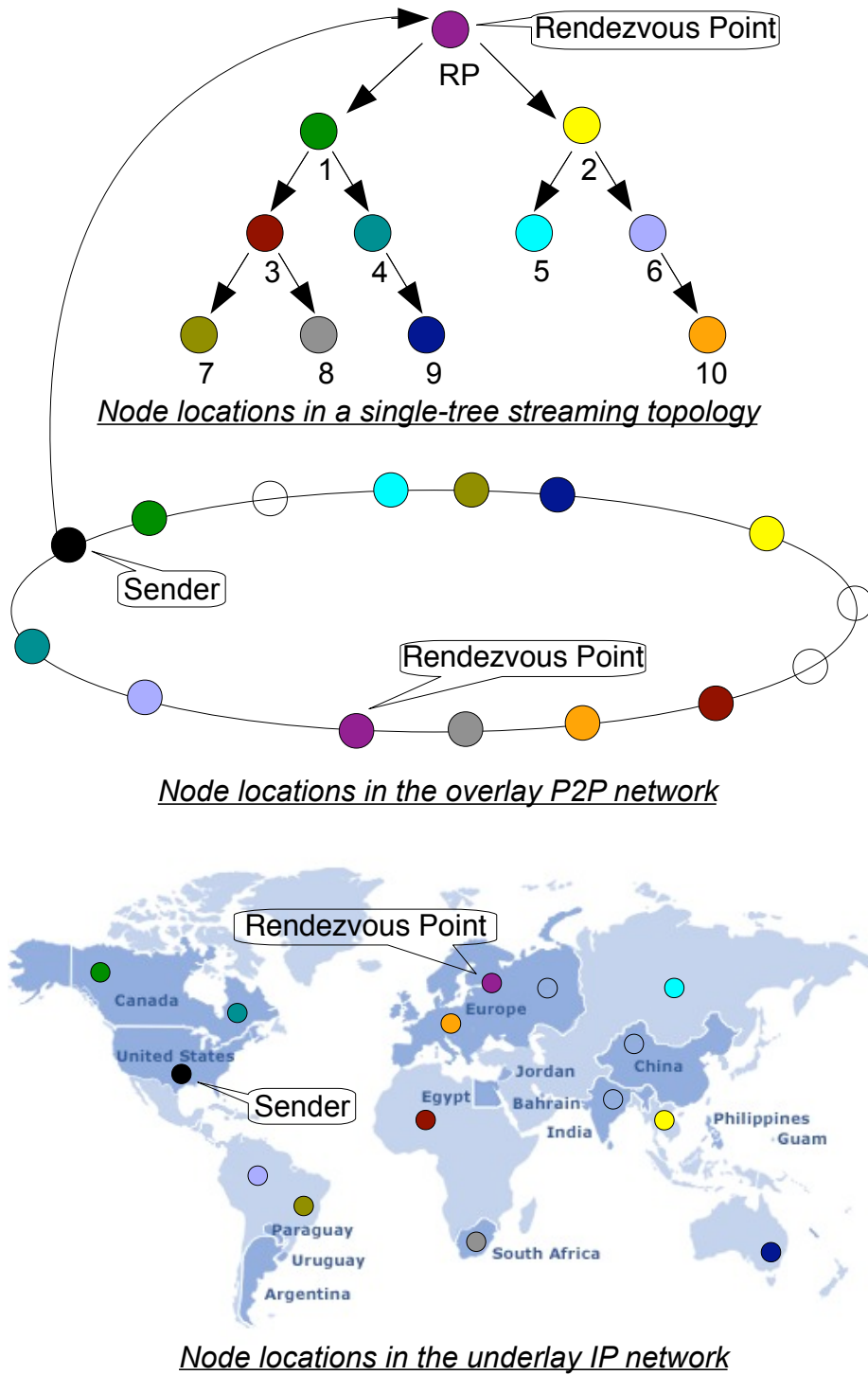


Figure 27: Scribe streaming tree built on the Pastry overlay P2P network

- `multicast(credentials, groupId, message)` causes the local node to send the multicast message (e.g. media packet) to the group with `groupId`.

A Scribe system operates on the DHT-based overlay network consisting of Pastry nodes, where each node runs the Scribe application software. Scribe uses Pastry for the routing mechanism especially in managing group creation, group joining, and multicast tree management (construction and maintenance). All operations of Pastry and Scribe are fully decentralized without any server interaction, where all decisions are based on local information. Besides the above API, two other methods i.e. "*forward*" and "*deliver*" need to be implemented on the Scribe layer, and are invoked in the Pastry layer whenever a message arrivals at a node. Specifically, *forward* is called whenever a Scribe message is arrived at a forwarding node. And *deliver* is called whenever a Scribe message is arrived at a destination node (i.e. no more forwarding, since the `nodeId` of the destination node is numerically closest to the key of the message), or when a message that was transmitted via "*send(msg, node)*"¹⁷ is received. There are four message types "CREATE, JOIN, LEAVE and MULTICAST" used by Scribe, those are used respectively with the invocation of the four methods (i.e. create, join, leave and multicast) in its API. Figure 28 and figure 29 show the pseudo-code for the forward method and deliver method respectively.

In the pseudocode, *groups* is the set of streaming groups that the local node is aware of, *msg.source* is the `nodeId` of the message source node, *msg.group* is the `groupId`¹⁸ of the group, and *msg.type* is the message type.

```

(1) forward(msg, key, nextId)
(2)   switch msg.type is
(3)     JOIN :       if !(msg.group ∈ groups)
(4)                   groups = groups ∪ msg.group
(5)                   route(msg, msg.group)
(6)                   groups[msg.group].children ∪ msg.source
(7)                   nextId = null    // Stop routing the original message

```

Figure 28: Scribe implementation of forward (source: [43])

Group management Each group has a unique `groupId` (a key in the Pastry overlay network). A Scribe node with a `nodeId` numerically closest to the `groupId` acts as the root (rendezvous point) of the multicast tree of this group.

¹⁷*send(msg, node)* is implemented on Pastry which is used to send a message directly to a given node with its IP address and port number, this method is described in the Pastry API section 2.2.3

¹⁸`groupId`: is a unique key (converted from its group name) used in a multicast streaming system which is similar to a data item key (converted from a file name) used in a DHT file sharing system discussed the DHT-based P2P section 2.2.2


```

(1) deliver(msg,key)
(2)   switch msg.type is
(3)     CREATE :      groups = groups  $\cup$  msg.group
(4)     JOIN  :      groups[msg.group].children  $\cup$  msg.source
(5)     MULTICAST :   $\forall$  node in groups[msg.group].children
(6)                       send(msg,node)
(7)                       if memberOf(msg.group)
(8)                             invokeMessageHandler(msg.group, msg)
(9)     LEAVE  :      groups[msg.group].children = groups[msg.group].children - msg.source
(10)                       if (|groups[msg.group].children| = 0)
(11)                             send(msg,groups[msg.group].parent)

```

Figure 29: Scribe implementation of deliver (source: [43])

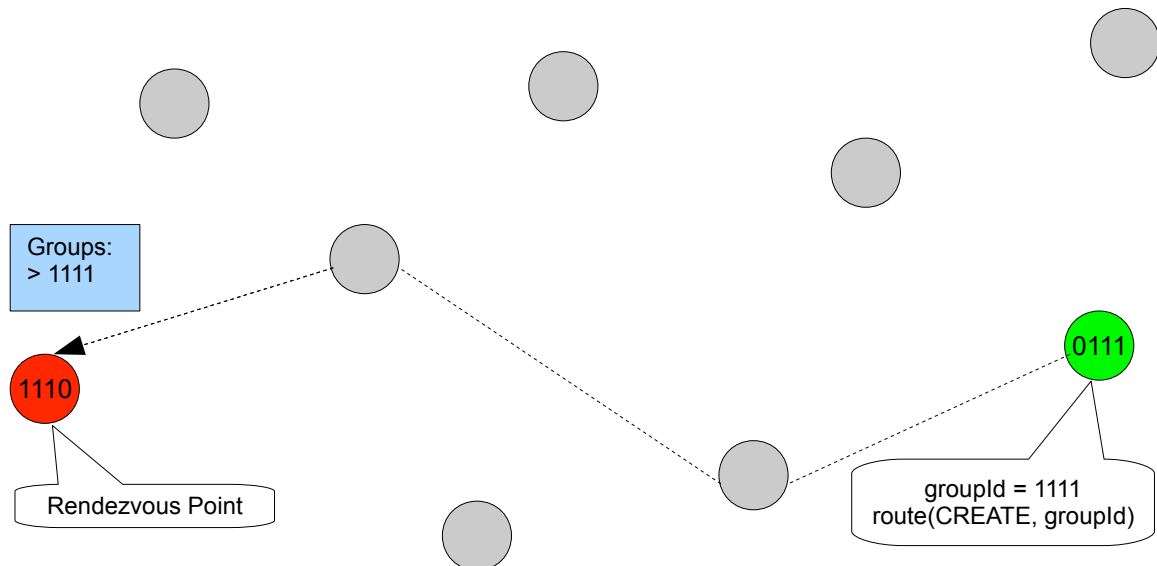


Figure 30: Group management (creation) in Scribe

For creating a group, a Scribe node asks Pastry to route a CREATE message using the groupId as the key (e.g. invoking the Pastry `route(CREATE, groupId)` method). This message is delivered to the node having nodeId numerically closest to groupId. The deliver method at this destination node is invoked and it adds the group to the list of groups it already knows (line 3 of figure 29). This destination node is now the Rendezvous Point (RP) of the group.

Figure 30 visualizes a scenario for a group creation. Node 0111 routes a CREATE message for groupId 1111 through two forwarders, and the message is delivered at node 1110 whose nodeId is numerically closest to groupId 1111.

Membership management The multicast tree is constructed via routing JOIN messages from subscriber nodes. Each streaming group has its own constructed multicast tree. On a multicast tree, Scribe nodes, those are not the RP and not the leaf nodes, are forwarding nodes (e.g. forwarders). RP and forwarders may or may not be members of the multicast group. Each forwarder maintains a children table of the group containing IP addresses, port numbers and nodelds of its children in the multicast tree. If a forwarder is a member of the group, it will receive multicast messages, use them for its application logic, and forward them to its children on this group. If a forwarder is not a member of the group, it will just forward the messages to its children without using the messages for its own application logic.

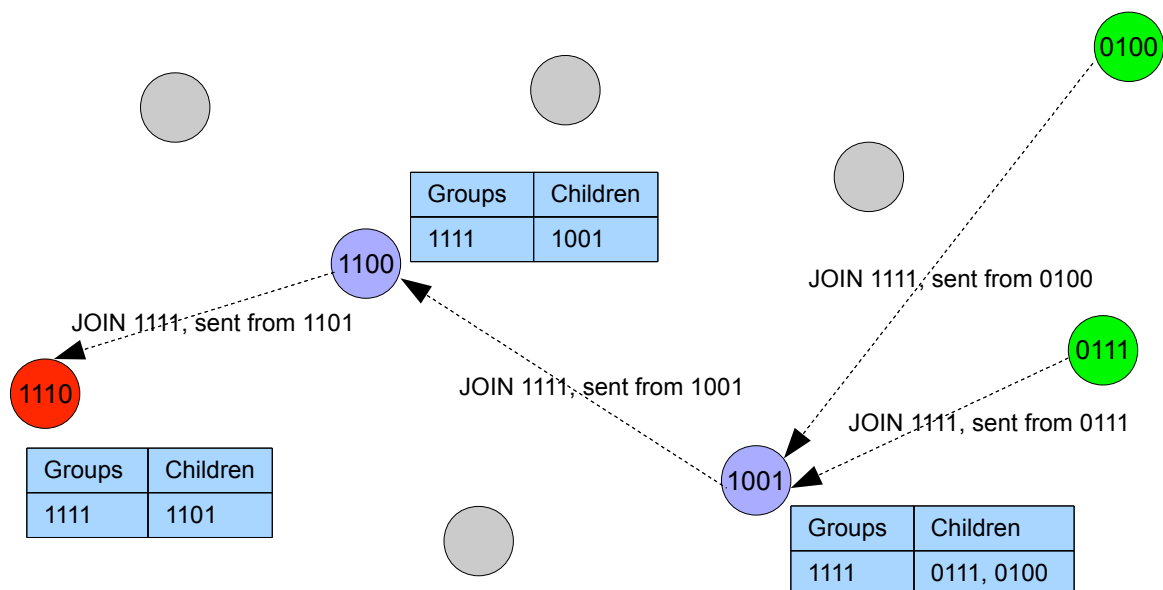


Figure 31: Joining a multicast group in Scribe

To join a group A Scribe node asks Pastry to route a JOIN message with groupId as the key. This means the routing path of this message is the path towards the RP. At each node along the route, Pastry invokes Scribe forward method (lines 3 to 7 in figure 28) to check if it is currently a forwarder for any other subscriber node of this group. If yes, it adds the joining sender node to its children list of this group and terminates this message routing. If not, it will become the forwarder - an intermediate level node in the streaming tree. It creates an entry of this group, then adds the sender node as the first child in the children list of this group, it modifies the JOIN message with its key the sender key of this message, and then sends the message to the next node along the routing path to the RP. This procedure is repeated at all forwarders during the JOIN routing.

Figure 31 visualizes a scenario for two nodes joining the existing group 1111 rooted at

the RP 1110. At first, node 0111 routes a JOIN message using groupId 1111 as key. Pastry routes this message to node 1001. Node 1001 checks to see that it has no group 1111 before. It then adds group 1111 and node 0111 to the children list of this group. Node 1001 becomes the forwarder and continues to route the JOIN message with itself the source sender to the next node 1100. This procedure is repeated until the JOIN message is delivered at the RP. After group 1111 is available on the forwarder node 1001, if node 0100 routes a JOIN message to node 1001, node 1001 would simply add node 0100 to the children list of group 1111 and stop forwarding the JOIN message to any further next node.

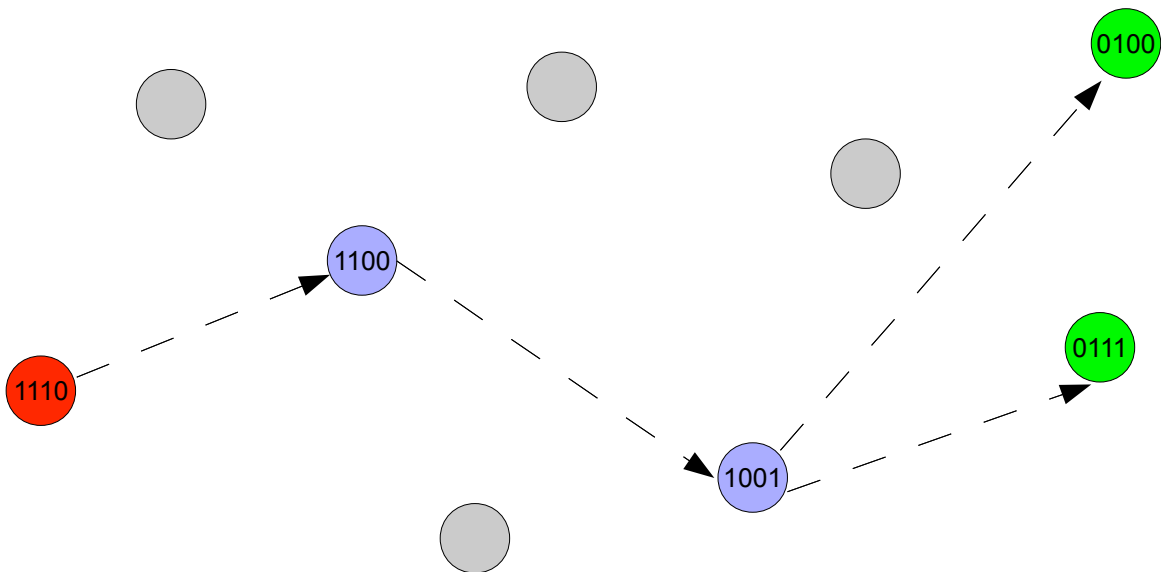


Figure 32: A case for a constructed Scribe multiset tree

To leave a group A Scribe node removes itself from the children list of the group. If the children list of the local node is not empty, this local node is still the forwarder but not a member of the multicast tree. If there is no node in the children list, it will send a LEAVE message to its direct connected parent (lines 9 to 11 in figure 29). The parent node will then remove this local node out of its children list, and the local node will not receive multicast message from the parent anymore. The message proceeds recursively up the multicast tree, until a parent node is reached that still has nodes in its children list after removing the departing child.

Figure 32 illustrates a case for a constructed Scribe multiset tree. Supposing that, node 0100 sends a LEAVE message for group 1111 to its parent node 1001. The entry 0100 is then removed from the children list of group 1111 on the parent node 1001. After that, if node 0111 sends another LEAVE message for group 1111, the parent node 1001 will remove the last entry in the children list of group 1111, and also sends another LEAVE message to its

parent 1100, since there is no child node that needs to receive multicast messages. And this process is again repeated at node 1100.

Multicast message dissemination Multicast source senders use Pastry to locate the RP of a group by calling the `route(MULTICAST, groupId)` method of Pastry, and ask the RP to return its IP address and port number. The source senders cache this IP address and port number for direct sending the subsequent multicast messages to the RP without repeating the routing through Pastry overlay network. At the RP, multicast messages are disseminated to its children, and the children then forward to their children. This procedure (lines 5 and 6 of figure 29) is done along the multicast tree repeatedly.

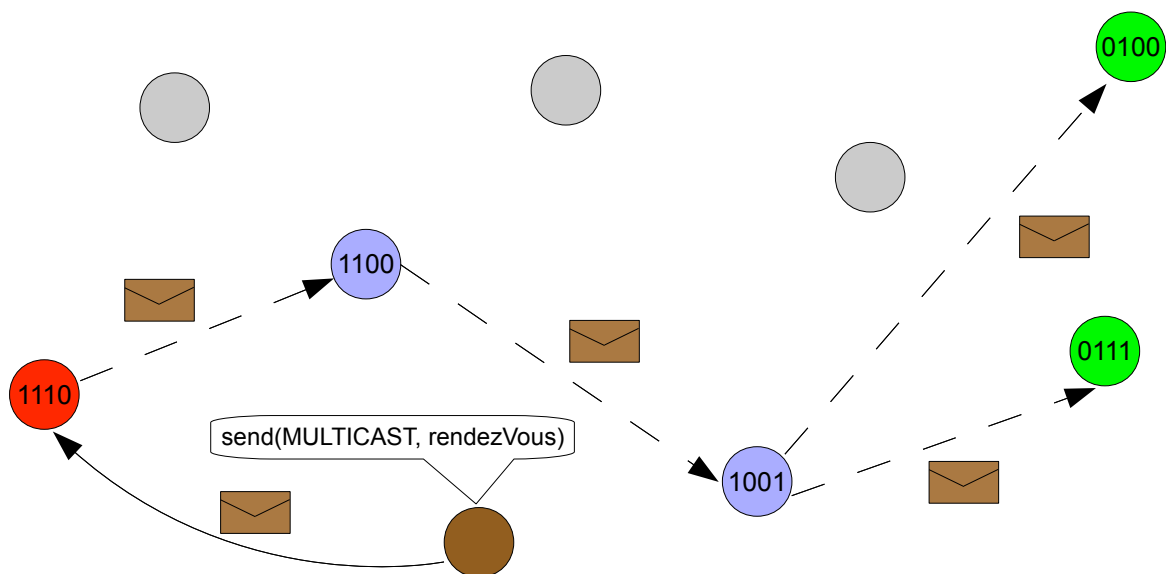


Figure 33: Sending multicast message in the Scribe multicast tree

Figure 33 illustrates that any node can send multicast messages to the RP node 1110. These messages are disseminated through forwarders (1100, 1001) to all subscriber nodes (0100, 0111) for group 1111 on the multicast tree.

Repairing the multicast tree Scribe uses Pastry to repair the multicast tree when a forwarder fails. Periodically, each non-leaf node in the tree sends heart-beat messages to its children. Within a heartbeating period of time, if there is already any multicast message sent to the children, the parent does not need to send the heartbeat messages to its children anymore. A child considers that its parent is faulty when it fails to receive heartbeat or multicast messages. In this case, the child will use Pastry to route a JOIN message to the groupId. Pastry will route this message to another new parent, and thus repair the multicast tree.

Entries in the children tables are discarded unless they are periodically refreshed by an explicit message from the child.

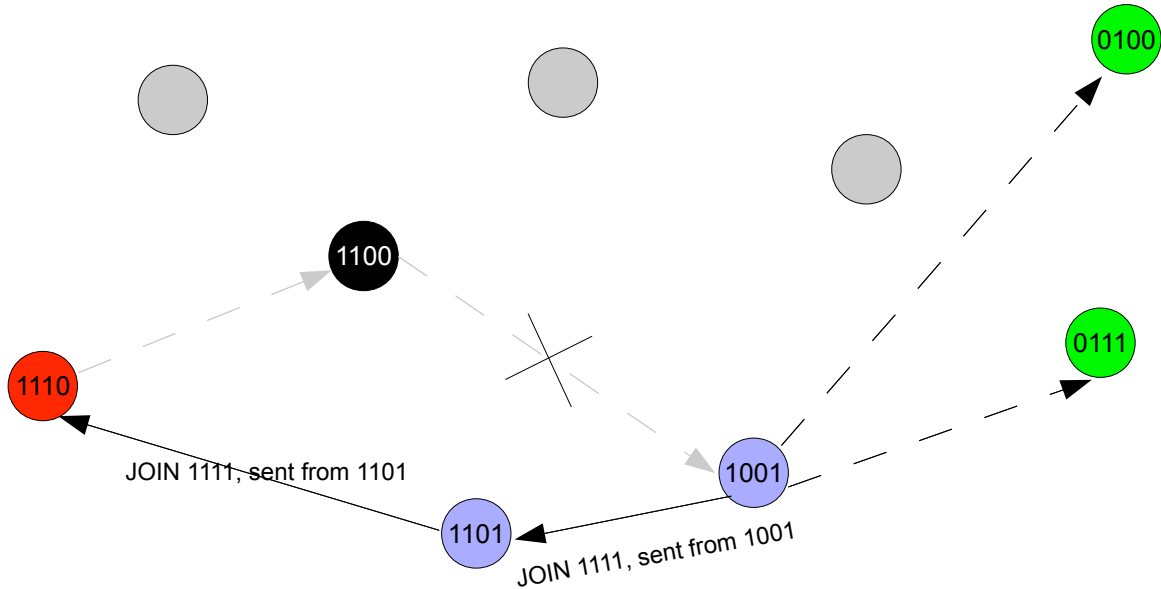


Figure 34: Repairing the Scribe multicast tree

Figure 34 illustrates a scenario where forwarder node 1100 departs from the multicast tree. The direct connected child 1001 of the parent node 1100 will have to route a JOIN message for group 1111 to another parent node 1101. Node 1101 again joins this group on the RP and the multicast tree can be repaired. After sometime, the RP receives no refreshed message from its leaving child node 1100 and the RP will remove the entry 1100 from the children list of group 1111.

Scribe replicates the state of the RP e.g. group creator, access control etc. across the k numerically closest nodes to the RP nodeId (typically $k = 5$). These closest nodes should be the nodes in the RP leaf set. Upon a failure detection of the RP, its direct connected children will use Pastry to route JOIN messages to the new RP (which is one of the numerically closest nodes in the leaf set of the old RP). Multicast senders likewise discover the new RP by routing via Pastry.

A RP state of a group on node A may have to be shifted to another new joining node. This situation may happen when node A is the RP of a group C, but after sometime, a new node B join the overlay and the key of B is numerically closest to group C as compared to current RP node A. Now, B will become the new RP of group C. And A has to send the RP state of group C to B, and discard its RP group state. For doing this, a updateHandler callback can be defined in Scribe which is invoked whenever there is an update on the leaf set table of a node. Since the keys of A and B are close, Pastry on A will update B to its leaf set table. And

this causes the `updateHandler` callback to be invoked. This handler function can check if the local peer is a RP of any group, and if any RP group state has to be copied to node B.

Providing additional guarantees Scribe also allows applications to implement stronger reliability guarantees via providing three up-call functions in three events, i.e., before Scribe is about to send a multicast message (`forwardHandler(msg)`), or when Scribe adds a new child to a group (`joinHandler(msg)`), or when Scribe detects the failure of a parent node of a group and is about to send the JOIN message to a new parent node (`faultHandler(msg)`). Using these functions, Scribe allows applications on its upper layer to modify the messages (`msg`) passed on each up-call function. These functions are implemented in the application layer and are invoked by Scribe. What an application would modify these messages is the strategy of this application. According to [43], an example for the use of these functions is that the `forwardHandler` at the root assigns a sequence number to each multicast message. Recent multicast messages (having greater sequence numbers) are buffered by all nodes in the tree. When a node detects the failure of a parent node, its `faultHandler` adds the last received multicast message sequence number `n` to the JOIN message. The receiver of this JOIN message uses the `joinHandler` to retransmit the buffered messages starting from the received sequence number `n`. This mechanism helps to implement an ordered, reliable delivery streaming system.

For more information on the experimental evaluation of Scribe, one can refer to [43].

According to Bharambe et al. [44], there are two principal reasons for advocating a DHT-based approach. *"First, DHTs provide a generic primitive that can benefit a wide range of applications, among them overlay multicast. Second, the same DHT-based overlay can be used to simultaneously support and maintain a large number of overlay applications and multicast trees. This could help achieve lower overheads as compared to constructing and maintaining several separate overlays."* One of the main differences of this structured DHT-based approach as compared to the unstructured Tree-based and Mesh-based approaches is that the streaming links of the unstructured approaches are established based on the resource and performance of each peer. These approaches are also called the performance centric approaches. Whereby, the peering relationship of the DHT-based approach is originally established based only on the numerically closest key routing mechanism. This policy cannot account for bandwidth heterogeneity in streaming systems and cannot ensure the requirement of growing tree more in fan-out degree than in depth. These are big drawbacks and may result in the unknown of the application performance. There are proposals to fix this bandwidth heterogeneity and the tree's high depth degree issues by adopting performance-based techniques which result in high fraction of non-DHT links (i.e. links exist in the streaming tree but are not part of the DHT overlay network) and application-specific monitoring overhead. The result of these proposals again conflict with basic DHT concept (i.e. a parent node is chosen via the closest key and not via the bandwidth).

Besides Scribe, SplitStream [45] is a Multi-tree multicast streaming approach built on top of the Pastry overlay.

2.6 P2P Streaming Systems

There are success stories on commercial or academic research ALM streaming deployments e.g. Skype, Zattoo and ESM. Skype [14] is a P2P VoIP, instant messaging and videoconferencing client application. It offers some free and paid services. It is available in many different desktop and also mobile platforms. Skype was founded in 2003 and was purchased by eBay in 2005 with an approximate price of \$ 2.5 billion. In 2009, eBay sold 65% of Skype in deal valuing it at \$2.75 billion. According to a statistic made by [46] on 29/10/09 at 11:30, there were 16.836.599 currently online users of 48.190.335 Skype registered users. Another commercial streaming system is Zattoo which was founded in 2005. Zattoo is a P2P live streaming system for realtime TV on the Personal Computer (PC)/Laptop. It offers free service for receiving normal quality TV stream and paid services for advertisements and for receiving high quality stream. In a report [47] from the Terena Networking Conference 2008, Zattoo claimed to have more than 2.3 million users at that time. In Switzerland in 2006, it is said that about 28% of broadband users was for Zattoo with 4 channels. While Skype and Zattoo are commercial systems, ESM (End System Multicast) [48] was a research project from the Carnegie Mellon University. ESM lets its users view and broadcast live streaming media. The Mesh-based streaming approach Narada (discussed in section 2.5.1) was designed for this ESM implementation. In this section, we would like to have a look on the key components of the two commercial Skype and Zattoo P2P streaming systems in term of their system architectures and functionalities.

2.6.1 Skype

Skype is the commercial system and its system internal design and implementation is not shown to the public. On its homepage, Skype only mentions that it uses overlay P2P topology, decentralized approach. It also claims that its call-completion rates are high for connections under firewalls and NAT. This makes Skype to be the successful VoIP system as compared to others, those having low all-completion rate which is said to be about 50% of residential computers unable to communicate with traditional VoIP software. The following presented technical information of the Skype system is based on the research work for analyzing the Skype protocol from [49].

Skype runs on a Hybrid overlay P2P network which is a two tier network including sub nodes and super nodes. A super node in the Skype network is elected among those sub nodes which has powerful resource, high bandwidth and has to own a public IP address. One of the main task of a super node is to relay media traffic to nodes behind NATs. Each sub node has to connect to one super node and must authenticate itself to a login server.

Figure 35 illustrates the relationship between sub nodes, super nodes and the login server. Each sub node also maintains a table of reachable nodes on the overlay network. A Skype sub node listens on particular ports for different incoming calls. Skype uses TCP for signaling and UDP or TCP for media streaming. Skype uses UDP for media packets where useable. But in case of both caller and callee are behind port-restricted NAT and UDP-restricted firewall, the voice traffic from both directions are forwarded over TCP via another online Skype node. Skype encrypts messages end-to-end. Generally, its codec maintains a call quality at an available bandwidth of 32 kbps of frequency between 50-8000 Hz, and a voice packet is of size between 40 and 120 bytes. According to [49], the key functions of Skype are startup, login, user search, call establishment and tear down, media transfer, and keep-alive messages.

2.6.2 Zattoo

Zattoo is also a commercial TV live streaming system. There is limited information about its detail protocol design. From a talk on Zattoo [47], one can learn that Zattoo uses the H.264/AAC codec for video and audio stream. All of its contents are encrypted. It is a P2P live streaming system which has those live streaming features as discussed in section 2.5.1 such as stream buffering, time bounced constraints and effect of the uploading bandwidth and session duration on the playback performance etc.

Zattoo organizes peers into a source rooted Directed Acyclic Graph (DAG¹⁹). Each TV channel is rooted at one root server on a particular overlay network. There are three key issues in constructing the streaming DAG. Firstly, Zattoo is topology-aware. For the efficient use of the streaming traffic, peers within the same subnet have direct links. Secondly, peers with high resource (e.g. bandwidth, processing power) are referred to be placed near to the source sender and to have more outgoing (uploading) links. Figure 36 shows the scenario where the first two issues are applied to form an effective streaming topology. Node A and B are in the same subnet and have a direct link. Considering that D has high bandwidth, it should be placed near to the source C and have two outgoing links. Thirdly, for nodes behind different NAT types, Zattoo considers Port-restricted NAT hosts cannot share with Symmetric NAT hosts and Symmetric NAT hosts cannot share with each other. These hosts have restricted download and upload capabilities. Zattoo considers these limited hosts (behind NATs) in its graph construction i.e. decision on establishing links between hosts. As compared to figure 36, figure 37 shows another scenario where node D and E are behind NATs. Zattoo now updates its topology for the best delivery effort.

Figure 38 shows the GUI of Zattoo. The left window is the live streaming player and a list of available channels is in the right window. Figure 39 shows the setup steps that work

¹⁹DAG is a directed graph with no directed cycles. That is, each node in the graph are connected to other nodes with links (incoming and/or outgoing links). And for any given node, if there are outgoing links, there is no way from these outgoing links to route back to this given node

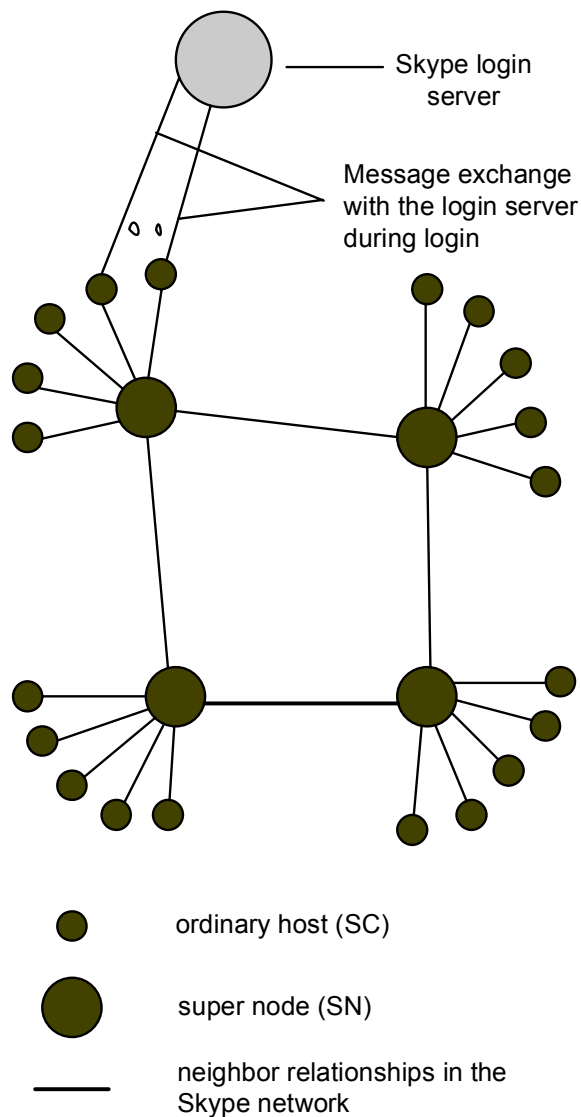


Figure 35: Skype Network. There are three main entities: supernodes, ordinary nodes, and the login server (source: [49])

for Zattoo. When a user selects a channel, the user has to authenticate its valid account with an authentication server (steps 1 and 2). Since Zattoo only allows a particular user to watch some particular channels (e.g., users from Germany can only select and watch the german TV channels), on success authentication, the selected channel is checked with the Rendezvous Host for the user right to watch his/her selected one (steps 3 and 4). When this test is approved, this user will send JOIN requests to the broadcast server and its known peers (step 5). A suitable peer will establish an outgoing link to this user (step 6). This

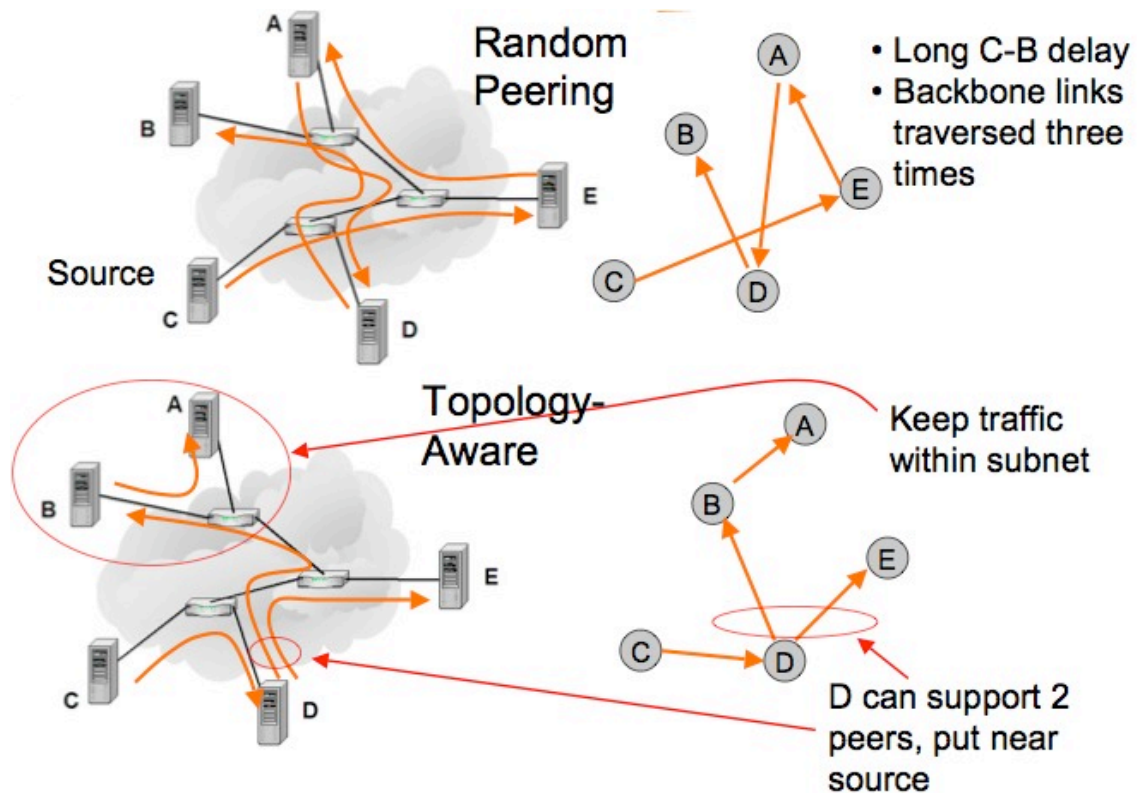


Figure 36: Zattoo's overlay topology-aware characteristic (source: [47])

suitable peer is selected according to the above discussed topology strategies. This user is then connected to the streaming DAG network of Zattoo and can be able to receive the live stream.

In its presentation for the Terena Networking Conference 2008, for running its service in Europe, about 500 servers were deployed in 15 locations across Europe. Zattoo expects its users to provide broadband Internet connection (>500 kbps). In conclusion, Zattoo claimed that the P2P network can help alleviating the bandwidth requirement, but still it is not sufficient to scale at present uploading capacity. This needs to further study on the performance and scalability of the Zattoo network. Figure 40 illustrates the Zattoo Infrastructure Architecture in the underlay network.

2.7 Conclusion and discussion

In this Streaming Technologies on the Overlay section 2.5, we have discussed the key concepts from the low to high layers for building up an ALM streaming system. We have shown that the advantages in deployment cost for large scale systems that P2P technologies brings

- Hosts behind NATs and firewalls have restricted download/upload capabilities

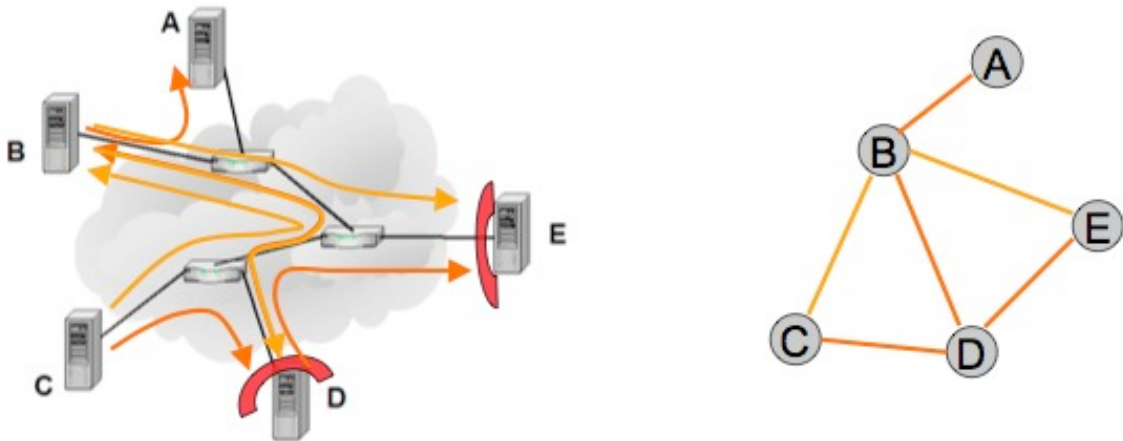


Figure 37: Zattoo's overlay topology-aware characteristic in case there are nodes behind NATs or Firewalls (source: [47])

over the traditional Client-Server model. We have discussed the reasons why IP Multicast is frequently not feasible and the choice of using ALM. In section 2.2, we have discussed different approaches and techniques for building an overlay P2P network, together with the discussion on the advantages and disadvantages of each approach. We have mentioned that the choice for which approach to use depends on the application targets.

Overlay P2P network is the communication layer for providing peering relationships and routing services to its upper application layer. Applications on this layer can be any file sharing systems, unicast or multicast systems etc. For the messaging protocol between these layers, we have discussed the so called three-tiers Dabek abstract model for the Structured P2P overlay including its suggested API for the common KBR layer. Although IP Multicast has not been widely used, IP Multicast does have advantages, e.g., its routing and bandwidth efficiency as compared to the ALM. For this reason, there have been proposals where both the use of IP Multicast and ALM are considered on one same multicast protocol. The Dabek model has to be extended for this hybrid multicast protocol. The work [50] from authors Matthias Wählisch and Thomas C. Schmidt was for this hybrid multicast model.

In the application layer, we have limited our discussion in the ALM for media streaming systems. There are two considerations in this discussion. The first one is about the characteristics and used technologies for a general streaming system (i.e. RTP, RTCP and Video/Audio codec). The second consideration is about different overlay P2P topologies for

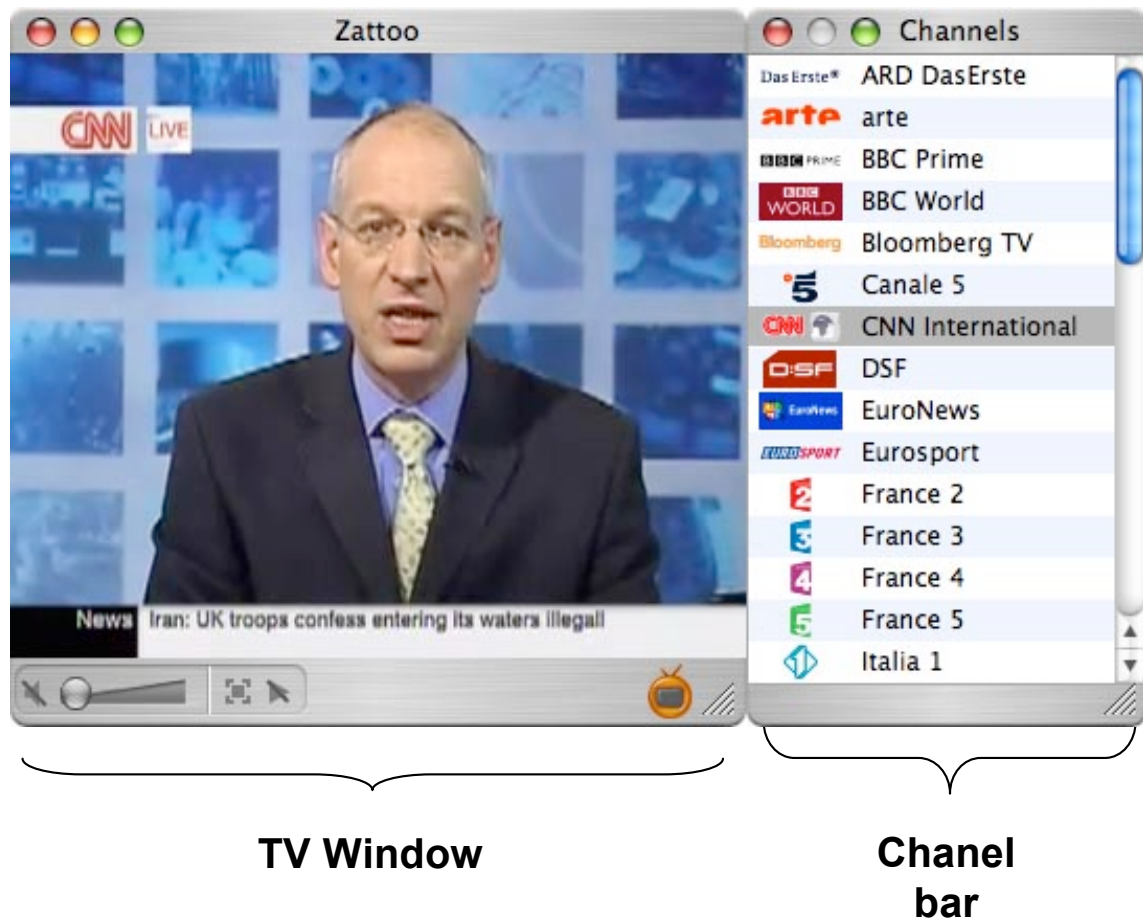


Figure 38: The Zattoo Player (source: [47])

the media streaming multicast systems. Basically, the Tree-first approaches provides the best delivery effort, but it deals with the peer churn problem. Whereby the Mesh-first approaches can solve the peer churn problem, but result in communication overhead and high degree of packets that arrive in-orderly. While in the Tree-first approach, the construction algorithm is the major issue, and in the Mesh-first approach, the packet scheduling is the most important design part. We have mentioned that the topologies and protocols depend on the application domains of multicast streaming system (i.e. whether it is a live, archive or on demand streaming system). We have discussed the characteristics, advantages, disadvantages and have compared on different possible streaming topologies for the live streaming and VoD systems.

Besides the unstructured Tree-first and Mesh-first approaches, there is DHT-based approach. In this approach, normally a streaming Single-tree (e.g., Scribe) or Multi-tree is built on a Structured or DHT-based overlay P2P network. This overlay provides the mes-

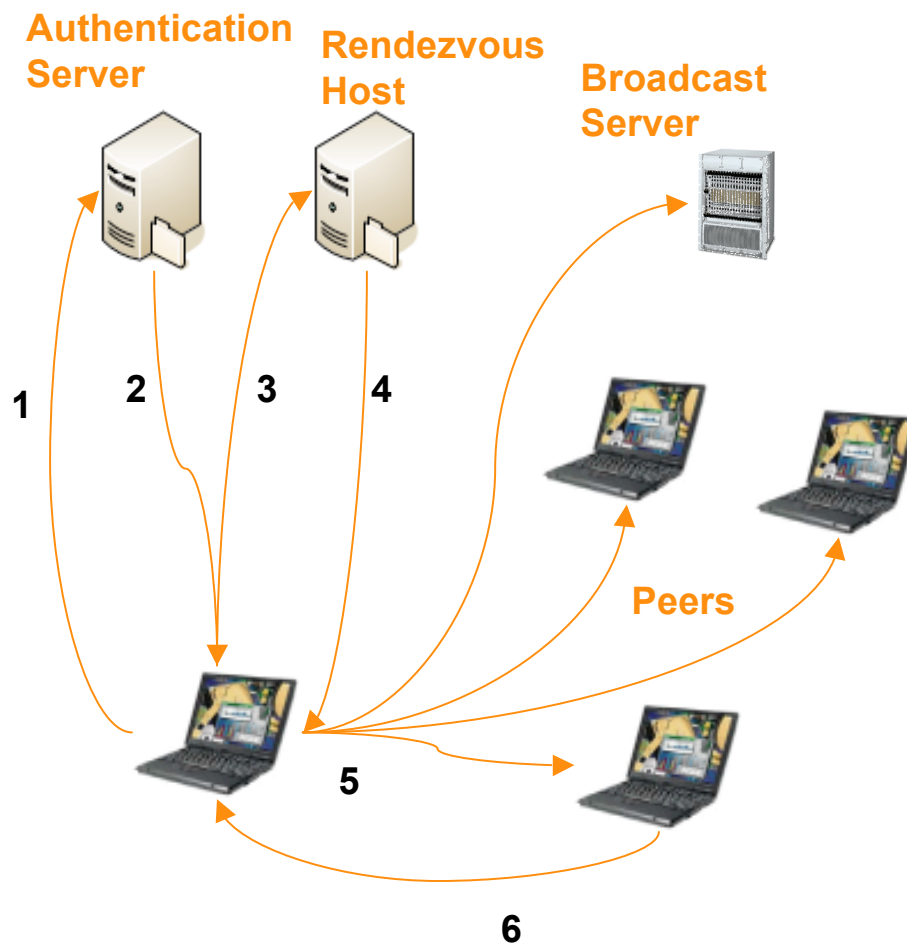


Figure 39: Steps to watch a channel in Zattoo (source: [47])

sage routing services for the construction and maintenance of the streaming tree. While the decision on the tree management of the unstructured Tree-first approach, and the peering relationship management together with the packet scheduling strategies of the unstructured Mesh-first approach are based on the resource and performance of the peers, the basic tree management concepts of the DHT-based approach are based on the closest key selection. This management does not account on the bandwidth heterogeneity which is very important for designing an effective streaming system. To deal with this problem, there are proposals to adopt performance-based techniques which result in high fraction of non-DHT links. This solution has its own drawback, because non-DHT links can not participate to the DHT-based overlay routing network and conflict with the basic of the KBR concept. Other proposals were trying to consider the bandwidth heterogeneity issue into the DHT-based approach and not considering for non-DHT links. The authors from the work [44] proposed solutions to this is-

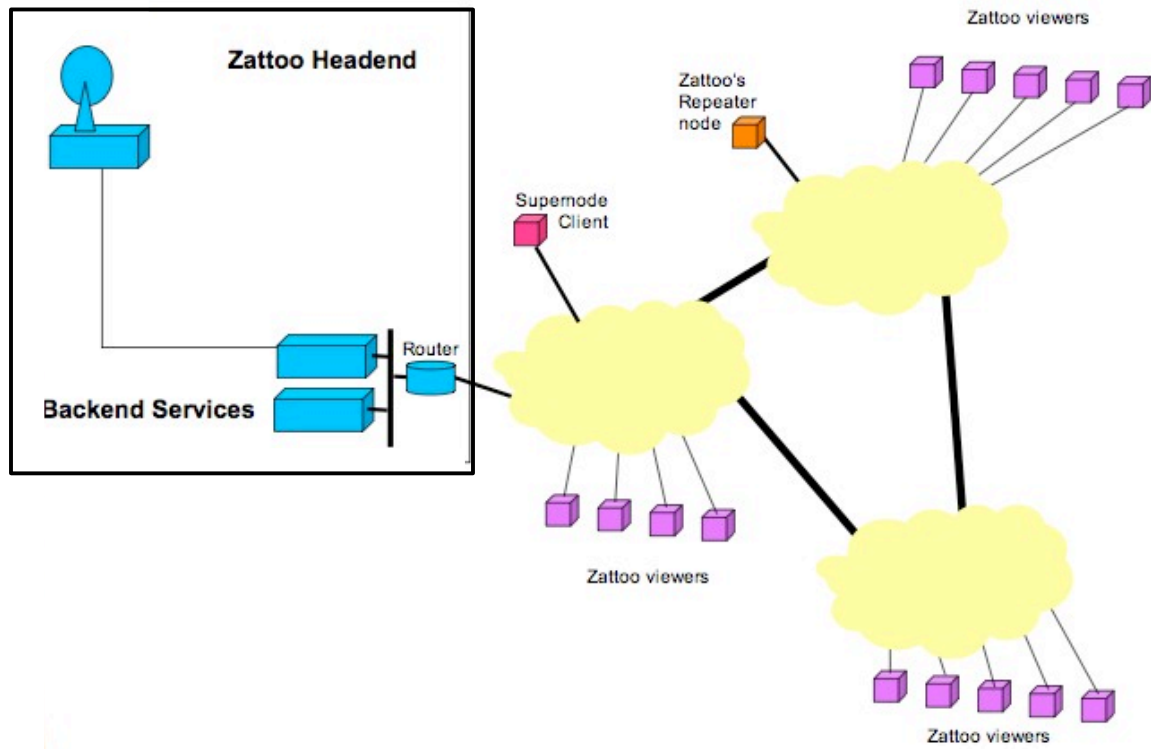


Figure 40: Zattoo's underlay infrastructure architecture (source: [47])

sue such as changing the random key assignment (e.g. via the SHA-1 function) and instead employing an assignment where node keys are correlated to node bandwidth constraints.

Our discussion went further briefly on the VoD streaming system. We have shown that main difference of the VoD streaming system as compared to the live streaming system is that VoD uses caching technique at each peers to cache the archive video stream and relay this stream to the new connecting nodes.

In general, the key issues when developing an overlay multicast application for media streaming are to ensure the system scalability for supporting a big number of user, the robustness under peer churn, the proximity and heterogeneity²⁰ considerations, and to ensure the working transparency for nodes behind NATs or Firewalls in a streaming system. The streaming system can also employ block coding and network coding techniques [51] which can be used to improve the content delivery throughput and eliminate the potential bottleneck at the source node.

Overlay P2P network and ALM have been developing for a decade. Many proposals have been published. Each one has its own advantages and disadvantages which aims to

²⁰Heterogeneity concerns the difference of the resource available to a peer e.g. the uploading/downloading bandwidth, the CPU and memory resource

solve a specific problem or to work for an application target. There are many open questions in improving the scalability, robustness and performance (e.g. long jitter delay in buffering, freezing or disruption during playback), those are still not fully answered. Zattoo claims that the uploading bandwidth of its users still can not be sufficient enough and needs to deploy more repeater nodes in the backbone network. ALM systems are said to consume a large portion of the Internet bandwidth. So that the future of the overlay P2P and ALM may need any further study in order to make ALM a friendly model to ISP and with better QoS to compete with the traditional broadcasting methods e.g. radio or television.

3 Audio Processing on the iPhone and iPod Touch

3.1 iPhone OS Technologies

The iPhone OS technologies are categorized into four layers. Figure 41 shows these four different layers. The lower-level layers provide the fundamental services for the higher-level layers and applications. The higher-level layers provide more specific sophisticated services and technologies for applications. The following information is taken from [52].

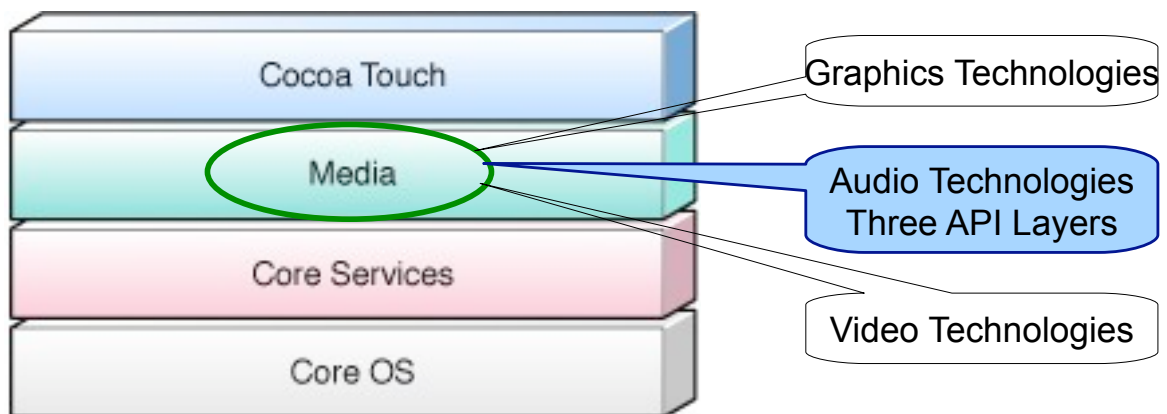


Figure 41: Layers of iPhone OS (source [52])

Cocoa Touch Layer This layer contains the key frameworks for developing applications. Developers should start developing with this layer and go down to lower-level frameworks when as needed. It provides the Apple Push Notification Service to alert users when something come up. It provides support for working with address book, email, map, P2P (built and operates on top of Bonjour²¹), interfaces for implementing graphical.

Media Layer This layer contains the graphics, audio, and video services. Each service is divided into higher- or lower-layer sub-services. Using sub-services in higher-layers makes it easy to create advanced graphics, animations, audio processing, etc. quickly. Whereby, the lower-layer sub-services provide more access control to the tools for doing exactly what a developer wants.

Core Services Layer All applications use the fundamental services provided by this layer. Even if these services are not used directly by applications, many parts of the system are

²¹ "Bonjour, also known as zero-configuration networking, enables automatic discovery of computers, devices, and services on IP networks" [9]. Bonjour is a service in the Core OS Layer

built on top of them. The services that this layer provides are address book access interfaces, core data for managing the data model of an application, core foundation (C-based interfaces) for data management and service features (e.g., data types (arrays, sets etc.), string management, date time, threads, ports and sockets communication etc.), foundation framework which provides similar services (Objective-C²² wrappers) as the core foundation does etc.

Core OS Layer This layer provides security, network and system supports, e.g., low-level UNIX interfaces for threading, networking, file-system access, Bonjour and DNS services, memory allocation etc.

For audio processing technologies used in the PAN4i application, some parts of the audio technologies in the media layer (highlighted in figure 41) are used and discussed in the following.

3.2 Core Audio Overview

Core Audio provides software interfaces for audio processing on the iPhone and on the Mac Laptop or Desktop. Core Audio includes a set of frameworks, those are for recording, playback, sound effects, positioning, format conversion, file streaming parsing, audio mixing, accessing to audio input and output hardware etc. As compared to Mac OS X, in iPhone OS, Core Audio is optimized for the limited computing resources, e.g., processing power, battery of the mobile device. Core Audio uses the notion of proxy objects to represent files, streams, audio players, etc. Most Core Audio interfaces use properties for managing object states or refining object behaviors. A property has a key-value pair. Many Core Audio interfaces use callback functions to communicate with applications. The use of callback functions for such things:

- To deliver new audio data to applications (e.g., for recording; the callback writes the recorded data to disk or sends to the network)
- To request new audio data from applications (e.g., for playback; the callback reads from disk to provide the requested data)
- To implement an event handler function (on an event occurrence, the callback takes appropriate actions)

According to figure 42, the programming interfaces of Core Audio are divided into different services, those are arranged into three layers. In the following sections, only the services

²²Objective-C is a object-oriented programming language which is used on Apple's Mac OS X and iPhone OS

used in the PAN4i application are discussed. For a complete information on other services, the Core Audio Overview homepage [53] and its related links are good sources.

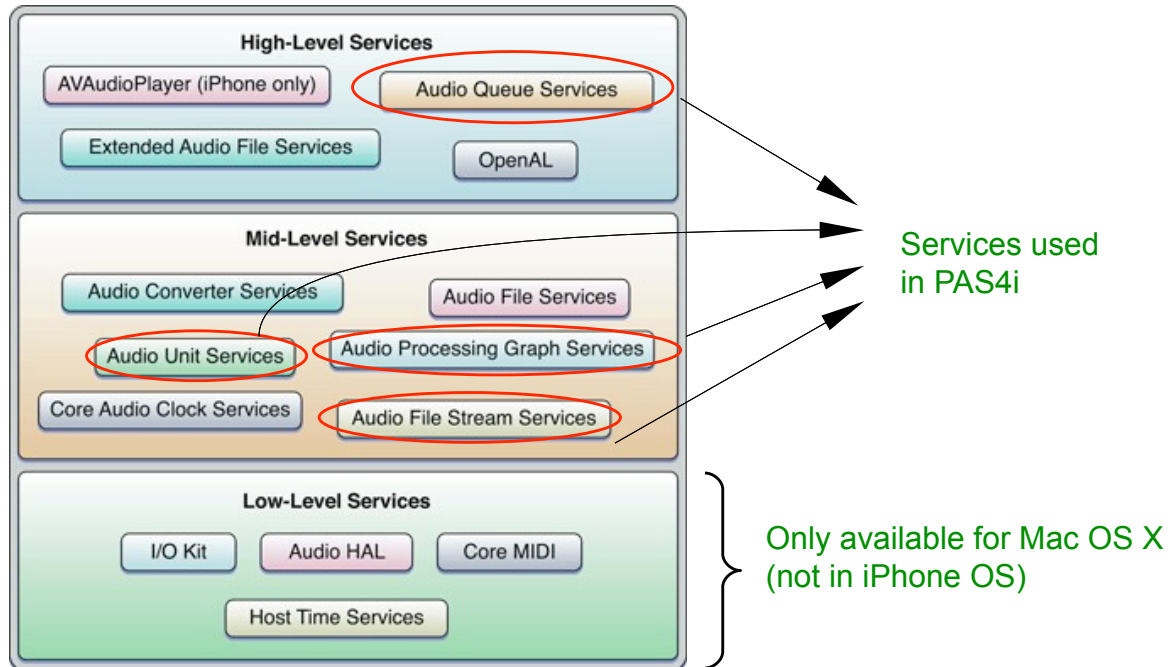


Figure 42: The three API layers of Core Audio (source [53])

3.2.1 Audio Queue Services

Audio Queue Services (AQS) API provides a straightforward and low overhead way to record and play audio. The audio recording and playing formats can be linear PCM, or any compressed format supported natively in the iPhone OS, or any format from third party installed codec. In figure 42, AQS stays in the high level which means that *"It lets your application use hardware recording and playback devices (such as microphone and loudspeakers) without knowledge of the hardware interface. It also lets us using sophisticated codecs without knowledge of how the codec work"* [53]. AQS and the AVAudioPlayer class are the only way to play compressed audio e.g., MP3, AAC etc. Audio File Services or Audio File Stream Services can be used together with AQS in order to provide audio input data to AQS e.g., for playback.

AQS uses `AudioStreamBasicDescription` structure to set the audio format which is going to be played or recorded which can be stored into an audio file.

```
struct AudioStreamBasicDescription
{
    Float64 mSampleRate;
    UInt32 mFormatID;
    UInt32 mFormatFlags;
    UInt32 mBytesPerPacket;
    UInt32 mFramesPerPacket;
    UInt32 mBytesPerFrame;
    UInt32 mChannelsPerFrame;
    UInt32 mBitsPerChannel;
    UInt32 mReserved;
};
```

In this structure, the `mFormatID` field represents the audio codec that AQS will employ for playback or recording. `mFormatID` can be one of the system defined ID for the supported codecs on the iPhone OS or any third party installed codecs. Some of the supported codec IDs on the iPhone are:

```
kAudioFormatLinearPCM = LPCM
kAudioFormatMPEG4AAC = AAC
kAudioFormatMPEGLayer3 = MP3
kAudioFormatiLBC = iLBC
etc.
```

An audio queue has three components: a set of audio queue buffers for storing temporary audio data, a buffer queue for an ordered list of audio data that is used for playback or for storing onto disk, and an audio queue callback function that provides audio data to the audio queue buffers on each invocation.

Figure 43 visualizes the playback process of the audio queue. In figure 43 a, the audio queue buffers are filled with audio data from the callback function. This data can be retrieved from the local hard disk or from the received network packets. When all the audio queue buffers are orderly filled, the speaker can start playing from the buffer index 1 (shown in figure 43 b). After finishing playing audio data in buffer 1, the audio queue returns buffer 1 for re-filling at callback function and continues to play the next buffer (shown in figure 43 b). This process is repeatedly to the last buffer 3 and starts again with the first buffer 1. For recording with AQS, the audio callback has a converse role.

3.2.2 Audio File Stream Services

According to [55], Audio File Stream Services API is used for parsing audio stream (from local file or from network stream) and returning packets of audio data or audio property

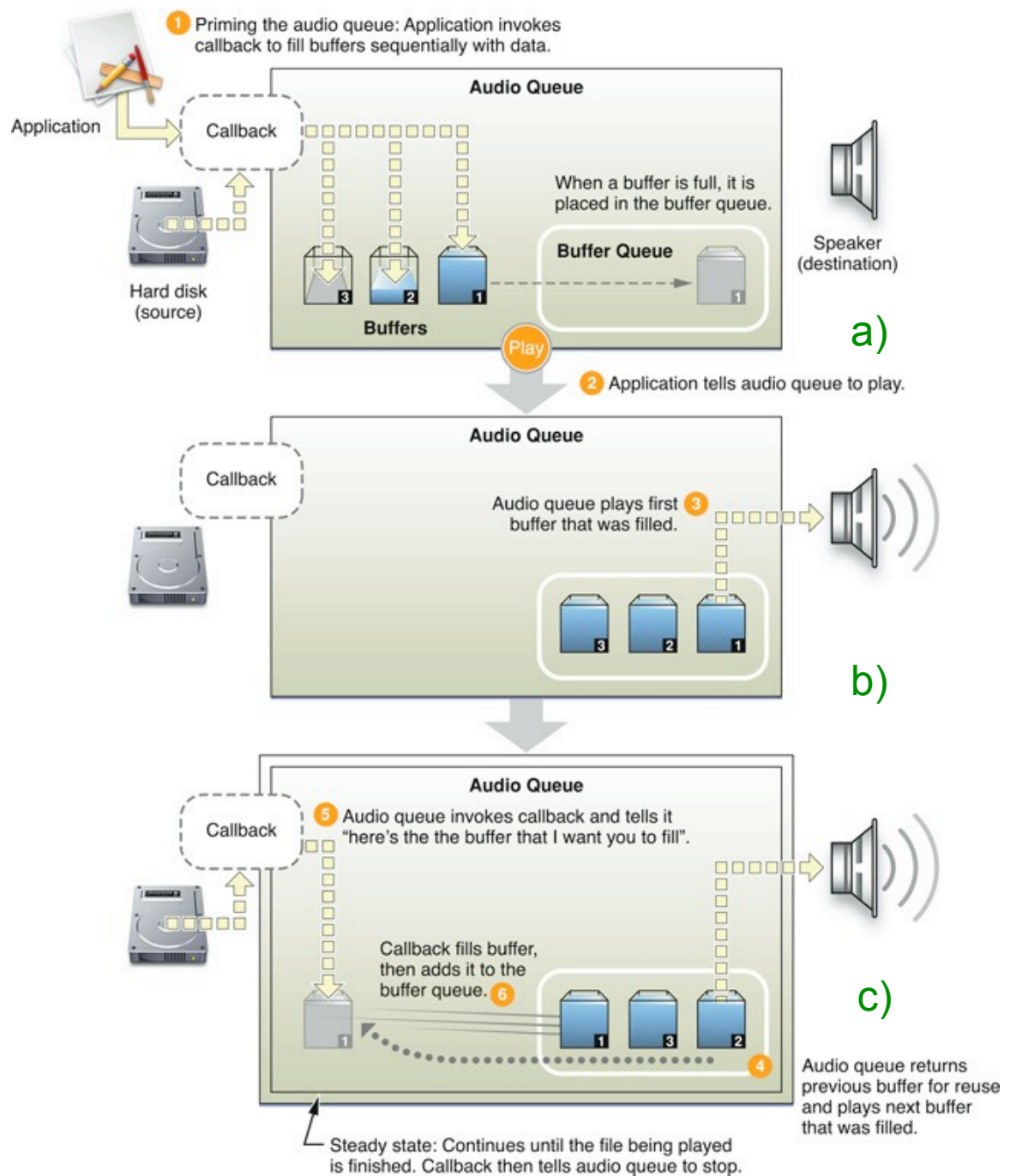


Figure 43: The playback process of AQS (source [54])

data. Two callback functions have to be provided, one for audio data processing, the other for property data . The audio stream is fed to the Audio File Stream Services parser. The

callbacks are invoked, when the parser has either a complete property value or a complete audio packet. To implement an Audio File Stream Services parser, one has to do the following steps.

1. Implementing callback functions for processing the audio property and the audio data. The callbacks are invoked and provided the parsed audio packet or with the audio property (type and value) in the callback parameters. The types of these two callbacks are

```
AudioFileStream_PropertyListenerProc
```

```
AudioFileStream_PacketsProc
```

2. Opening an audio file stream via calling the function

```
AudioFileStreamOpen (  
    void * inClientData,  
    AudioFileStream_PropertyListenerProc inPropertyListenerProc,  
    AudioFileStream_PacketsProc inPacketsProc,  
    AudioFileTypeID inFileTypeHint,  
    AudioFileStreamID *outAudioFileStream  
)
```

The `inClientData` parameter is a constant that will be passed to the callbacks every time they are invoked. The `inPropertyListenerProc` parameter is the pointer to the audio property callback function. The `inPacketsProc` parameter is the pointer to the audio data callback function. The `inFileTypeHint` parameter indicates the file type of the stream. And the `outAudioFileStream` parameter is the new file stream ID for use in other `AudioFileStream` API calls.

3. Passing the stream to the parser via calling the below function with parameters for the file stream ID, the data passed to be parsed, the number of bytes for parsing and the flag to indicate whether the data is discontinuity.

```
AudioFileStreamParseBytes(  
    AudioFileStreamID inAudioFileStream,  
    UInt32 inDataByteSize,  
    const void* inData,  
    UInt32 inFlags  
)
```

3.2.3 Audio Unit Services

Audio Unit Services provides the mechanism for applications to work on audio input and output with low-latency, e.g., VoIP. It also provides some digital signal processing features. iPhone OS audio units use linear PCM audio data format for input and output. For describing the information about capabilities and configuration, audio units use properties. One can set and get values for properties of an audio unit via using `AudioUnitSetProperty` and `AudioUnitGetProperty` functions. Further more, the setting values of audio units can be adjustably changed in real time. Callback functions can also be used for event handler, input requesting and output rendering. Audio units have a number of units, and some of them are used in the PAN4i, i.e., Converter unit, Multichannel mixer unit and RemoteIO unit. These units are discussed as follows:

Converter unit This unit, of property type `kAudioUnitSubType_AUConverter`, allows to convert the sample rate, bit depth, and bit format (linear or fixed-point) between different audio PCM formats. This unit does not convert between PCM and other compressed audio formats. For audio format compression, the Audio Converter Services (which does not belong to the Audio Unit Services) is used instead. Figure 44 illustrates the Converter unit where PCM audio data goes into the unit on input bus 0, and the converted PCM* data can be retrieved on output bus 0.

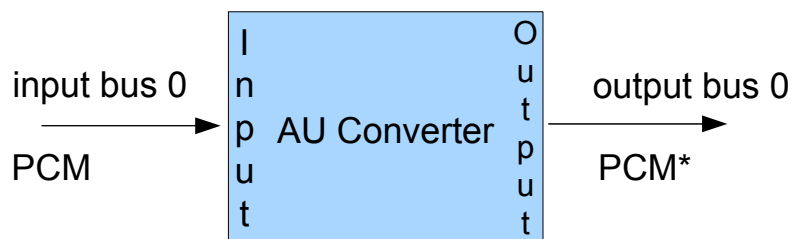


Figure 44: Converter unit

Multichannel mixer unit This unit allows to mix multiple audio input PCM streams to a single output PCM stream. It also allows applications to mute or unmute each input channel and to control its volume. Figure 45 illustrates the Multichannel mixer unit where there are more input channels, each one on each different input bus to the unit and the mixed stream is produced at the output bus 0. The property type of this unit is `kAudioUnitSubType_MultiChannelMixer`.

RemoteIO unit This unit, of property type `kAudioUnitSubType_RemoteIO`, allows connections to audio input hardware (microphone) and output hardware (loudspeaker), and

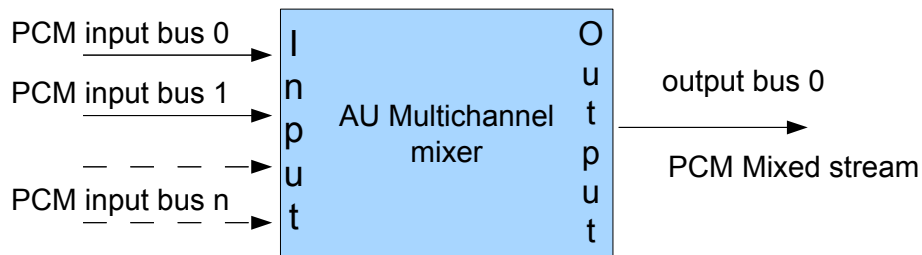


Figure 45: Multichannel mixer unit

supports realtime I/O. By using this unit, one can retrieve the recorded voice from the mic for use in application, and can send any audio data to the loudspeaker for playback. Input and output audio data formats of this unit are also PCM. Figure 46 illustrates the use of this RemoteIO unit. Any input audio data from application provided to this unit has to take input bus 0. And recorded voice from microphone takes input bus 1. Whereby, output to the loudspeaker takes output bus 0, and applications can retrieve output audio data (e.g., the recorded voice audio data) on output bus 1.

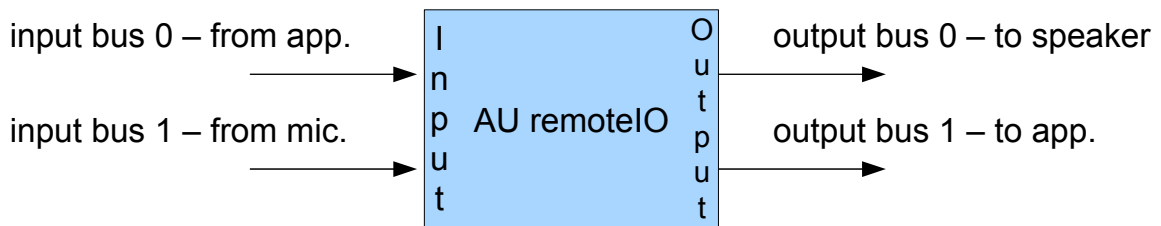


Figure 46: I/O unit (or called RemoteIO unit)

3.2.4 Audio Processing Graph Services

"Audio Unit Processing Graph Services provide interfaces for representing a set of audio units, connections between their inputs and outputs, and callbacks used to provide inputs. It also enables the embedding of sub (or child) processing graphs within parent graphs to allow for a logical organization of parts of an overall signal chain." [56].

Figure 47 shows the use of this Audio Unit Processing Graph Services for connecting inputs and outputs of different audio units to form a processing signal chain. This signal chain is also used in the PAN4i application. Starting from the input bus 1 for the microphone of the remotelIO unit, the high quality PCM recorded voice goes through the remotelIO unit, and out at bus 1 to the application. This output bus 1 of remotelIO is connected to the input bus 1 (considered for voice input signal) of the Multichannel mixer unit. Music audio data got from the callback goes to the input bus 0 of the mixer. The high quality PCM mixed audio

stream is produced at output 0 of the mixer is connected to the input bus 0 of the Converter unit. This unit converts the high quality input PCM to a lower quality PCM at output bus 0 of the Converter unit. This converted audio can now be sent to the network, and/or for playback. Audio data goes to input bus 0 for the remoteIO unit which will directly go to the loudspeaker for playback.

By connecting these audio units in a processing graph and setting the property values of each unit, the audio signal is automatically processed without the need to write any extra code for controlling the process of each unit. The callback function has to be implemented for providing input audio data to the mixer. Audio Unit Processing Graph allows to define any callback function for any input bus of any unit. If a callback for providing input data is registered with an input bus of an audio unit. This input bus cannot be connected with any output bus of any audio unit.

In order to use this graph service, one has first to declare some audio units of type `AUNode` (e.g., `ioNode`, `converterNode`, `mixerNode`), those are connected via the graph service API for audio processing. After this step, an audio processing graph can be setup via the following functions.

`NewAUGraph` to create a new audio unit processing graph

`AUGraphAddNode` to add each declared `AUNode` to the graph

`AUGraphConnectNodeInput` to connect any pair of node (e.g., output bus of the `mixerNode` is connected to the input bus of the `converterNode`)

`AUGraphOpen` to open the graph and so the audio units in the graph are also opened for further initialization, e.g., for initializing property values of any node or defining any callback function to provide input to any node in the graph

`AUGraphNodeInfo` to return information about a particular node

`AUGraphSetNodeInputCallback` to register any callback function for providing data to any input bus of any node

`AUGraphInitialize` to initialize the graph containing already configured nodes

`AUGraphStart` to start the graph and let it running

From figure 47, in order to get the output audio at bus 0 of the AU Converter for sending to the Internet, one can use the `AudioUnitRender` interface. This interface is used to get the audio data at the output bus of any audio unit, e.g., to get the recorded audio from the microphone.

3.3 Audio codecs

For recording and playback, iPhone OS provides two types of audio codecs, i.e., hardware-assisted codecs and software codecs.

The hardware-assisted codecs include the audio formats listed in table 4. The advantage

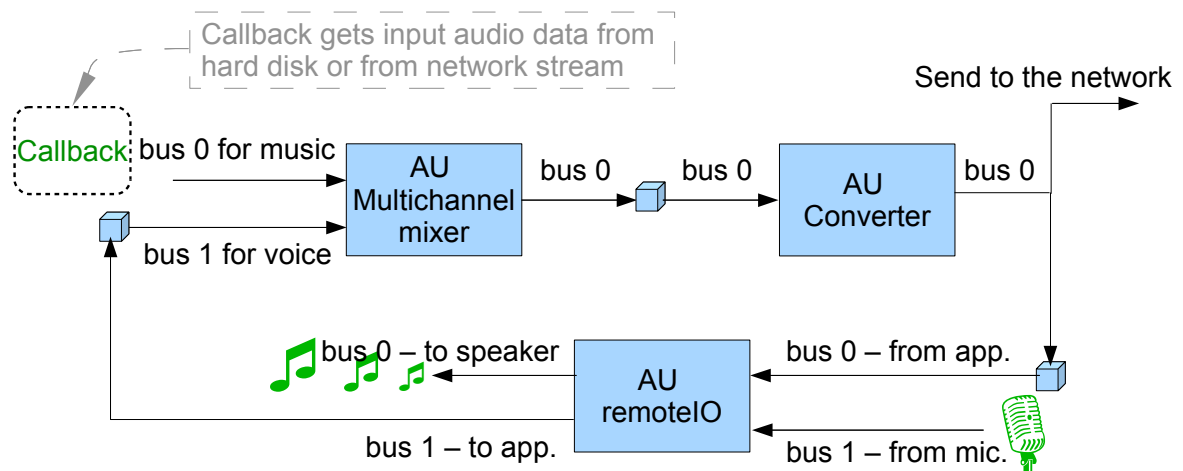


Figure 47: Audio processing chain using Audio Unit Processing Graph interfaces

of using the hardware-assisted codecs is to save the processing resource on the iPhone (e.g. battery). Since all the hardware-assisted codecs share a single hardware path, there is only one single instance these three supported formats that can be played at a time. According to [57], "Offline rendering with an audio queue output allows applications to render audio to a buffer instead of directly to an output device". The main interface used for offline rendering (i.e., using the hardware-assisted codecs) is `AudioQueueOfflineRender`.

AAC
Apple Lossless
MP3

Table 4: iPhone OS audio formats supported by hardware-assisted codecs (source: [53])

The supported software codecs are listed in table 5. By using the software codecs, one can concurrently play more than one instance of each of these formats.

AMR (Adaptive Multi-Rate, a speech codec)
iLBC (internet Low Bitrate Codec, also a speech codec)
IMA/ADPCM (also known as IMA-4)
Linear PCM
μ Law and aLaw

Table 5: iPhone OS audio formats supported by software codecs (source: [53])

For voice recording²³, iPhone OS contains the supported recording codecs listed in table 6.

Apple Lossless
iLBC (internet Low Bitrate Codec, also a speech codec)
IMA/ADPCM (also known as IMA-4)
Linear PCM
μ Law and aLaw

Table 6: iPhone OS: recording audio formats (source: [53])

²³If using Audio Unit Services to implement a voice recording function, the audio data got from the microphone is Linear PCM. By using the Audio Converter Services, the PCM recorded voice data can be converted into other different compressed formats

4 PAN4i Concepts and Design

4.1 User requirements

The idea to develop the PAN4i application is to create a P2P networking entertainment application in the form of karaoke singing and in combination with group communication on the iPhone and iPod Touch devices. Any iPhone or iPod Touch (called peer or node in the overlay network) having the PAN4i installed and network (W-LAN) connectivity can use this application. Figure 48 illustrates the use cases of this application. Generally, the application would allow users to self organize (i.e., create or join) an overlay P2P network without any server interaction. Any user can receive (i.e., joining a streaming group) and contribute (i.e., sending multicast) audio streams to a shared, multi-channel application that we coined for karaoke. Users may request music from a distributed jukebox or a local music file and sing karaoke music on an iPod Touch microphone. Simultaneously, any user may stream the live karaoke music to his or her own, personally created channel on the overlay network. All other users, when subscribing to this channel, will be able to receive and listen to live stream karaoke music. According to the above description, the following user requirements are defined:

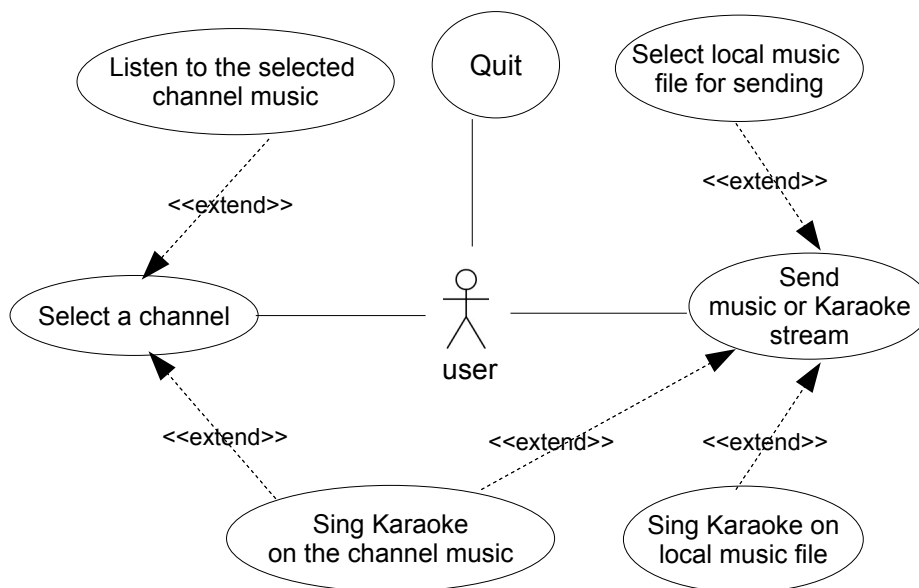


Figure 48: Usecase diagram of the PAN4i application

Requirement 1: Get informed about bootstrapping information
 Precondition: no
 Postcondition: the local peer has information about a nearby peer on an existing overlay or gets informed that there is no existing overlay network
 Basic actions (dependent on the bootstrapping strategy of the system): 1. If using IP Multicast for providing bootstrapping information

in a local network, then when a node is connected to the local network, it can get this information via the IP Multicast in the local network. The bootstrapping information can also be retrieved from some permanent active peers on the overlay (the software can store a list of these peers). Or it can be retrieved from a server not belonging to the overlay. This server will need to be informed about peers in the group. The bootstrapping peers can be updated with the newest requesting peers.

In the rest of these requirements, the bootstrapping information is considered to be available for any peer that may need in order to carry out other actions in each user requirement.

Requirement 2: Start up a new overlay network

Precondition: no

Postcondition: A new overlay network is created with only one peer

Basic actions:

1. The user creates a new overlay network
3. The user is acknowledged on success or failure

Requirement 3: Join in an existing overlay network

Precondition: There is an existing overlay network, a new peer has to know any existing peer on this overlay network

Postcondition: This new peer is connected to this overlay network

Basic actions:

1. The user gets informed of any existing peer on the overlay network
2. The user sends an overlay JOIN request to this existing peer
3. The user request is acknowledged on success or failure

Requirement 4: Create a new streaming group

Precondition: The user peer is on the overlay network

Postcondition: A new streaming group is created and available on the overlay

Basic actions:

1. The user gives a new group name, and sends a CREATE request to the overlay network
2. The user request is acknowledged on success or failure

Requirement 5: Multicast a music stream from a local file to a group

Precondition: The user peer is on the overlay network, the streaming group exists on the overlay network

Postcondition: no

Basic actions:

1. The user gives a group name, and sends MULTICAST messages of the music stream from a local music file to this group

2. On failure, the user is informed that the multicast attempt was fail

Requirement 6: Multicast mixed Karaoke stream of the live singing voice and the background music stream the a local file

Precondition: The user peer is on the overlay network, the streaming group exists on the overlay network (if the iPod Touch is used, an external microphone has to be connected)

Postcondition: no

Basic actions:

1. The user gives a group name, and sends MULTICAST messages of the mixed Karaoke stream to this group
2. On failure, the user is informed that the multicast attempt was fail

Requirement 7: Join a group to listen to a music or Karaoke stream

Precondition: The user peer is on the overlay network, the streaming group exists on the overlay network

Postcondition: no

Basic actions:

1. The user gives a group name, and sends a group JOIN request to the overlay
2. The user listen to the receiving music or Karaoke stream
3. The user is informed if after some time no stream is available or end of the streaming session

Requirement 8: Join a group for receiving a background music stream and sing Karaoke on this music stream

Precondition: The user peer is on the overlay network, both the streaming groups for receiving the background music stream and for sending the Karaoke stream to another group exist on the overlay network (if the iPod Touch is used, an external micro phone has to be connected)

Postcondition: no

Basic actions:

1. The user gives a group name, and sends a group JOIN request to the overlay (for receiving the background music stream)
2. The user sings on the receiving background music, and listen to his or her produced Karaoke music and send MULTICAST messages of this Karaoke stream to another group (his or her own Karaoke group)
3. The user is informed if after some time no receiving stream is available or end of the streaming session

Requirement 9: Leave a group

Precondition: The user peer is on the overlay network, the streaming group exists on the

overlay network, the user is a subscriber of this group

Postcondition: music playback of the receiving stream stops

Basic actions:

1. The user gives a group name, and sends a group LEAVE request to the overlay
2. The user hear no music playback anymore
3. The user request is acknowledged on success or failure

Requirement 10: Quit the PAN4i application gracefully while being as a subscriber of a streaming group

Precondition: The user peer is on the overlay network, the streaming group exists on the overlay network, the user is a subscriber of this group

Postcondition: music playback of the receiving stream stops, PAN4i stops running

Basic actions:

1. The user press on the quit button of the iPhone or iPod Touch device

Requirement 11: Quit the PAN4i application unexpectedly while being as a subscriber of a streaming group

Precondition: The user peer is on the overlay network, the streaming group exists on the overlay network, the user is a subscriber of this group

Postcondition: music playback of the receiving stream stops, PAN4i stops running

Basic actions:

1. The user does not press the quit button on the device, but the PAN4i application turned off, e.g., device crashes

4.2 Functional requirements

This section defines the required functionalities and the related technologies that are needed to develop the application and to fulfill the user requirements as described above.

The application allows peers to organize themselves into an overlay network. It allows a new node to create or join an overlay network. It also allows any peer to send signaling messages to the overlay network. These are the requirements for an implementation of a structured overlay P2P network. The implementation has to have basic functions for creating and joining an overlay network, updating its routing state tables, routing messages to any peer in this network. For the message routing, it must have the functions for selecting a next routing node, delivering the message at a node and stopping the routing, and providing the updating ability to the routing information on upper layer application. It should be scalable and robust and able to host a large number of peers. These requirement can be fulfilled with the use of Pastry overlay network (described in section [2.2.3](#)). Pastry provides a set of API calls that allows it to do exactly what the application requires for setting up such an overlay network.

When peers are in the overlay network, the application allows the peers to create any personal streaming group. Any peer can also send audio (music or karaoke) stream to a group where all subscriber peers of this group will be able to receive the stream for playback. Any peer can also leave a streaming group or change to another group. These are the requirements for an implementation of an ALM system. The implementation has to provide the basic functions for creating, joining, leaving and multicasting to a streaming group. The Scribe ALM approach (described in section 2.5.4) can be used for this purpose. Firstly, Scribe operates on top of Pastry. Secondly, Scribe organizes peers of a streaming group into a streaming tree hierarchy which provides the best delivery effort. Scribe also provides a set of API calls for any peer in the overlay that can create, join, leave and multicast to a streaming group. Scribe also has methods for repairing the multicast tree in case any node departs from the tree gracefully or unexpectedly, e.g., by computer crashes. For doing this, all Scribe nodes on intermediate tree levels have to send heart-beat messages to their children periodically. If after sometime, a child does not receive any multicast or heart-beat message from its parent, it will look for another parent. Since a RP node may fail or quit the system, Scribe replicates the RP group state information on the RP node to the nodes in its leaf set table. The RP group state of a node A may have to exported to another new coming B if the new node B key is closest to the RP group hosted on node A as compared to the key of A. The replication of the RP group state, or copying from one node to another can be done in an handler function which can be invoked on the leaf set updating event in Pastry.

The application must allow any user to playback a receiving audio stream, sing Karaoke on a playing music stream from the network or from a local file. These are the requirement for implementing a real-time audio processing functions. The implementation has to provide the basic function for live playing an audio stream from the network, mixing the local or network stream with the recorded singing voice and sending the live mixed stream to the network. Further more, the implementation should be able to provide appropriate audio codecs that may take the network bandwidth into consideration. The codecs may be used to do down or up-sampling, convert to another audio format. The Core Audio API (described in section 3.2), apart of the iPhone OS, can help to fulfill these requirements. It provides interfaces for getting audio packets from the microphone buffer and sending any audio packet to the loudspeaker for playback. It provides a multichannel mixer unit for mixing music stream and recorded voice, and procedure the mixed Karaoke stream. And it also provides different codecs for different audio processing purposes, e.g., iLBC for VoIP, MP3 for music etc.

For delivering audio stream in real-time, TCP is not used because of its long delay in re-transmission for lost and error packets. UDP is used with the consideration that the packets may arrive out of order. An adaptive jitter buffer is needed to re-order the packets before playback. This requirement can be answered with the use of RTP which is discussed in section 2.5.3.

From the defined functions and technologies applied for each requirement, the PAN4i system can be described as follows. Generally, each node has to join an existing Pastry

overlay network. If no overlay network exists, a node can start a new overlay network. Each node has a key value derived from its IP address and port number. A streaming group channel also has its own key value which is derived from its channel name. Not only a node is the destination for the key of itself, it is also the destination node of any channel whose key is numerically closest to its node key. In Scribe, a destination node of a channel is called the RP of this channel. Pastry uses the keys (of nodes and channels) to organize nodes and distribute channels on nodes in its overlay network in a way that for a given message with a key, Pastry can route this message to the correct destination node of this key value. An example is that a new channel of key A (created by any node) is routed to the destination node of key B where A and B are numerically closest. B becomes the RP of channel A. Any message with key A or B is routed to node B. The routing mechanism of the system follows the Pastry protocol. When a message arrives at a node (or starting from the sender), this local node will lookup a numerically closest key in its routing tables to the message key. This message is then forwarded to the next node of the returned closest key. If the local node key turns out to be the closest key, this message is delivered at this local node and the routing finishes here.

Based on these principles, the following section will discuss on the software concepts and application design of the PAN4i application.

4.3 Software concepts

Derived from the functional requirements, the main functional units and their dependence relationships of the PAN4i software can be defined and illustrated in figure 49. This figure shows that the group creating, joining and multicasting units (provided by Scribe for group management and multicasting) are derived from the overlay starting or joining unit. This is explained as these functions can only be invoked on a peer after the peer is already in an overlay network which means that the overlay starting or joining function (provided by Pastry) would have been invoked before. The group leaving, and heart-beat messaging functions (Scribe) can only be invoked after the group joining would already have been invoked. Furthermore, when the overlay starting or joining function is invoked, the including bootstrapping function is also invoked to provide bootstrapping information needed by a peer for joining to the overlay network. And the same for multicast sending unit, when it is invoked, it uses the request IP of RP function to request for the IP address and port number of the RP so that the subsequent multicast messages can be directly sent to the RP without going through the routing procedure. Moreover, the multicast sending unit includes the RTP/RTCP unit, so that any audio packets sent out to the network via using this unit are encapsulated in RTP packets. Similarly, when a node joins a group, the receiving stream will go to the multicast receiving unit. This unit also includes the RTP/RTCP unit, so that the receiving RTP packets are decapsulated to audio packets for further processing. Finally, the overlay starting or joining unit, the group creating and joining units and the IP of RP requesting unit use the KBR

unit of Pastry to route their messages to destinations. KBR unit will invoke its extending forward, deliver and update handler functions for routing a receiving message to the next node, delivering the receiving message to the local node, or to perform any appropriate action when the local leaf set table is updated.

In order to realize the use cases shown in figure 48, for multicasting audio (music or karaoke), the three extended units of the multicast sending unit can be used. They are the "sing on a local music file", "select a local music file" and "sing on receiving stream" units. Whereby, for listening to a multicast stream (music or karaoke), the derived "listen to receiving stream" and "sing on receiving stream" units of the multicast receiving unit can be used. The "sing on receiving stream" unit is a special unit. It is the extended unit of the multicast sending unit and also the derived unit of the multicast receiving unit. For listening music and singing karaoke from network stream or local music file, these units have to use the audio processing unit which is the Core Audio API and described in section 3.2.

The PAN4i is developed based on the Three Tier Dabek model (section 2.3) as shown in figure 50, which means that all the functional units are placed in these three tiers. From this model, the Pastry on tier 0 provides the overlay start, join, bootstrapping and KBR services. The Scribe ALM approach is built on top of Pastry and is placed at tier-1 and provides group create, join, leave, multicast and request IP of RP functional units. Tier-2 is for the implementation of the application GUI, application logic and audio processing, i.e., the music listening, Karaoke singing functional units.

The PAN4i software concepts can be more clearly described with the two figures 51 and 52. Figure 51 illustrates the communication concepts that PAN4i uses for sending signaling and multicast messages to peers. Correspondently, figure 52 illustrates the communication concepts that are used for the arrivals of signaling and multicast messages at a peer on the overlay.

The general concepts shown on both figures are that the application layer includes the GUI, application logic for using the functional units, and the audio processing unit. There are two types of packet transmitted in the system, i.e., the messaging packets for signaling and the audio packets for multicasting. The audio packets can be part of a MP3 music or a Karaoke stream. The signaling messages are divided into a number of message types, some are used by the KBR service - Pastry, and others are for the ALM - Scribe and any of the user defined message types needed for the application logic. These messages are used for managing or updating the overlay network, the streaming trees or any application specific purpose, e.g., exchanging leaf set information between nodes, adding or moving nodes from a tree, etc. The main Pastry message types can be PASTRY_JOIN (for joining an overlay network), PASTRY_UPDATE (for updating the leaf set). Whereby, the main Scribe message types can be SCRIBE_CREATE (for creating a group channel), SCRIBE_JOIN (for joining an existing group), SCRIBE_LEAVE (for leaving a group), and SCRIBE_MULTICAST (for sending multicast messages). Each Scribe message corresponds to a function defined in the Scribe API (e.g., Create, Join, Leave, Multicast) Any new message type can be defined

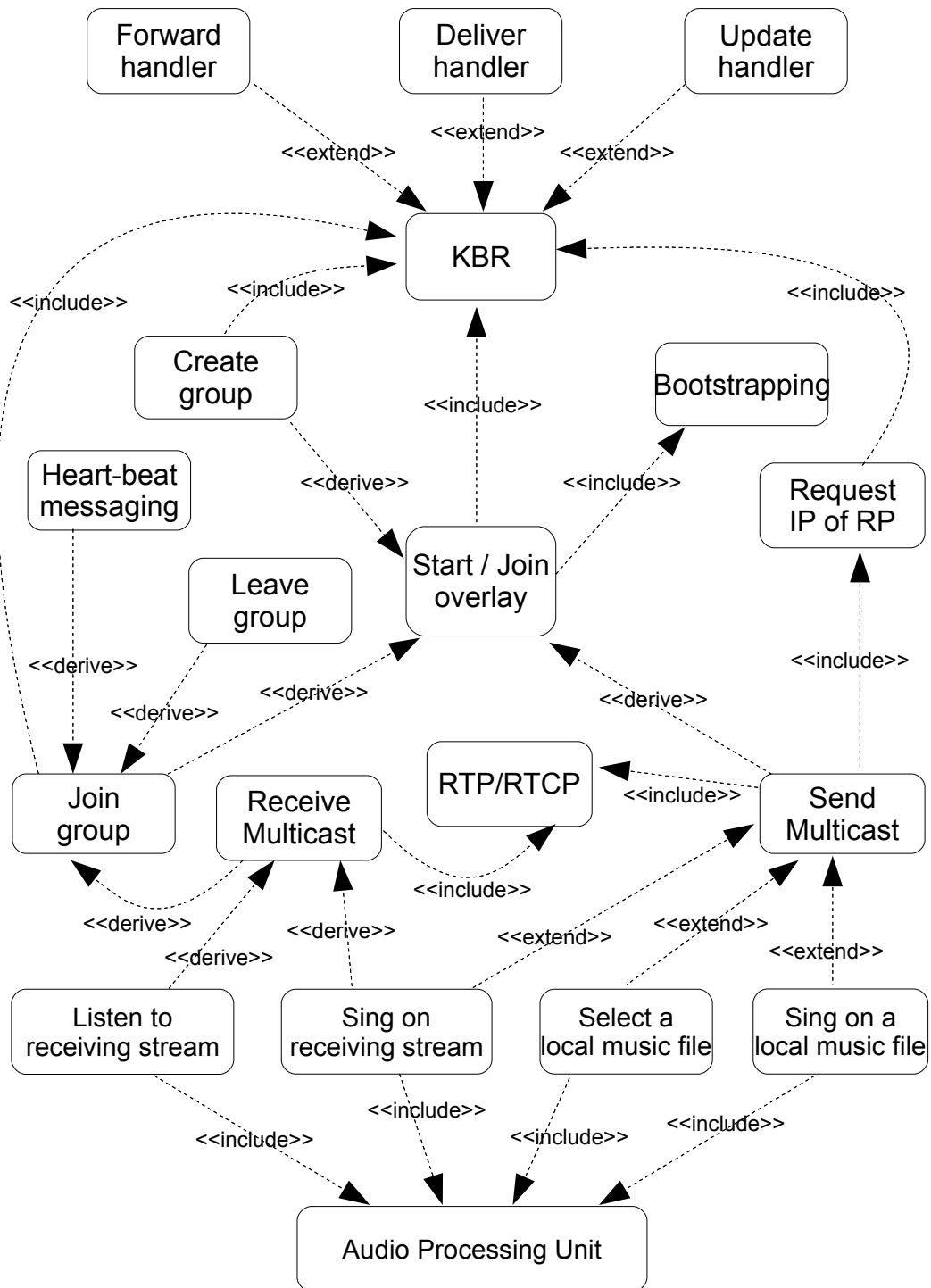


Figure 49: The main functional units and their dependences of PAN4i

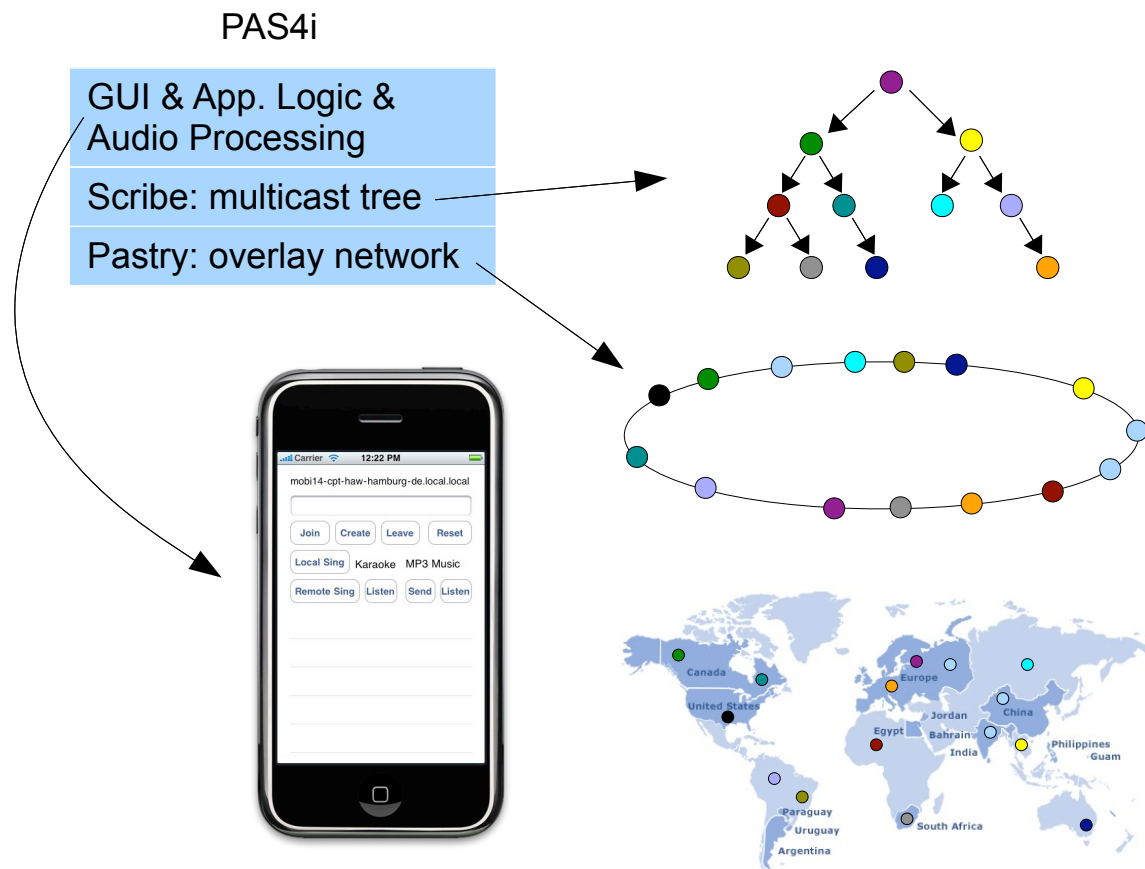


Figure 50: The PAN4i Three Tiers Dabek model

for the application logic as needed (e.g., REQ_IP, RESP_IP for requesting and responding the IP address and port number of a RP). The Scribe and user defined message types can be used in two different ways. A message with a given key can be sent via the KBR service (forwarding at forwarders and delivering at a destination receiver), or a message with a given IP address and port number can be sent directly to the destination receiver.

For the smoothing playback experience at receivers in a streaming network application, the PAN4i uses the RTP to encapsulate its sending audio packets (figure 52). At the receiver side, packets can be re-ordered in the jitter buffer for playback. RTP is discussed in section 2.5.3.

For the highest possible real-time streaming experience, the audio packets have higher priority to the signaling messages for processing. This can be done via using the main thread for audio processing and creating new threads for message processing. Mover over, by using two different sockets for sending messages and audio packets, at the receivers, messages and audio packets are de-multiplexed and go to their audio streaming or messaging socket

interface. This helps to determine the type of the packets for performing appropriate processing quickly without the need to read into the packets, and would improve the performance of the application.

To be concrete, in figure 51, the PAN4i starts up via calling the join (overlay) function on the application layer to join the overlay. This call sends the PASTRY_JOIN message with the node key to the KBR service for routing the message on the overlay. The routing procedure of Pastry is clearly described in section 2.2.3. During the routing to a destination peer, the local peer receives routing information from a bootstrapping node, forwarding nodes and a destination node to build up its own routing tables. On success of the routing, this local node with its own routing tables is available on the overlay network. This overlay joining procedure of Pastry is also described in section 2.2.3.

After joining to the overlay network, the local peer can send messages with keys via the KBR service (i), or it can send messages with IP address and port number of a node directly to this node (ii), or it can send multicast audio packets directly to a RP of a group (iii) (shown in figure 51).

An example for the (i) sending method is that, a peer can send Scribe messages with a group key for creating a group or joining an existing group. The KBR service is used to forward the messages to a next forwarding node or to deliver the message to the application layer of at a destination node (i.e., the node key and message key are numerically closest). The forwarding and delivering of messages at a receiver can be illustrated in figure 52. In this figure, the arriving messages go through the messaging socket to the KBR service. Here, Pastry performs the routing lookup procedure. If this local node is the destination node, the deliver handler function is invoked. If the message is of type SCRIBE_CREATE, then a new group of the message key is created and the local node becomes the RP of this group. If it is of type SCRIBE_JOIN, then the local node will add the source sender node key to the children list of this message group key. The deliver handler function will return the control to the application layer for performing any other processing. If this local node is not the destination node, the forward handler function with a returned next forwarding node is invoked. This handler may perform any extra processing before the message is sent to the next node. The update handler is invoked whenever there is an update in the leaf set of the local node. An application may use this handler to perform any specific action on this event.

For direct messaging in (ii), Pastry also provides an API call for this purpose. By providing a message with an IP address and a port number, the message can be sent directly to the receiver. An example is that a Scribe node on an intermediate level wants to leave the streaming tree. Since this local node contains IP addresses and port numbers of its parent node and children nodes, it will just send the SCRIBE_LEAVE messages directly to these node without using the KBR service. At the receiver (figure 52), the direct message handler function is invoked, e.g., to remove the source sender node key from the children list of a group. To destroy a streaming tree, the RP of a group can send direct messages (e.g., user defined message type TREE_DESTROY) to its children to inform them that the streaming

group is no more available. The children will remove the group state and repeatedly inform their children of this tree group. This situation may be happened, if after sometime, the RP receives no multicast message from any source sender on this group. A Scribe intermediate node on a streaming tree uses the heart-beat function for sending hear-beat messages periodically directly to its children to inform its availability and connecting state.

When a Scribe streaming tree is constructed, and after receiving the IP address and port number of the tree RP, the peer can send multicast audio packets to this RP directly. Depending on different audio multicasting purposes (e.g., sending a normal MP3 stream, or Karaoke stream encoded in MP3 format), different audio processing methods are applied. Audio processing methods for MP3 music or mixed Karaoke stream are described in section 3. At sender, audio packets are encapsulated in RTP packets and are sent through the streaming socket to a receiver (e.g., the RP). At the receiver (figure 52), RTP packets go through the streaming socket to the media packet handler function. Here, the RTP packets are forwarded to the children list of the current streaming group directly. Then the local node will check if it is also a subscriber of this group. If yes, then RTP packets are decapsulated and re-ordered (using jitter buffer), and sent to an audio processing unit. At the application layer, the processed audio packets can be sent to the loudspeaker for playback, or the application may perform mixing and send the mixed stream again to another group (i.e., the case for singing Karaoke on a receiving music network stream).

4.4 Application design

The above described software concepts are the functional units and footprints that the PAN4i needs to have and follow in order to provide services as describe in the use case figure 48. In this section, application design is about the selecting of different approaches for implementing the application logic as well as GUI, that can bring all these software concepts into a well cooperating mechanism to let the application running like what is described in the use cases.

Figure 53 demonstrates an exemplar that the PAN4i GUI can be designed. Basically, there are four main windows in the application: a home window (figure 53 a), a window for selecting a MP3 local file for multicasting (figure 53 b), a window for singing Karaoke on a selected local file or on a receiving music stream and for multicasting the live Karaoke stream (figure 53 c), and a window for listening to a music stream (figure 53 d). The layout designs of these window are similar, except the input forms in the middle of the windows. The common layout components are the home button for going to the home window, the stop button for stopping playing music, the status message on the top for informing the current usage mode (idle, sending MP3, sending Karaoke, or listening to a music stream, and the current receiving or sending channel name), the up-to-date channel list.

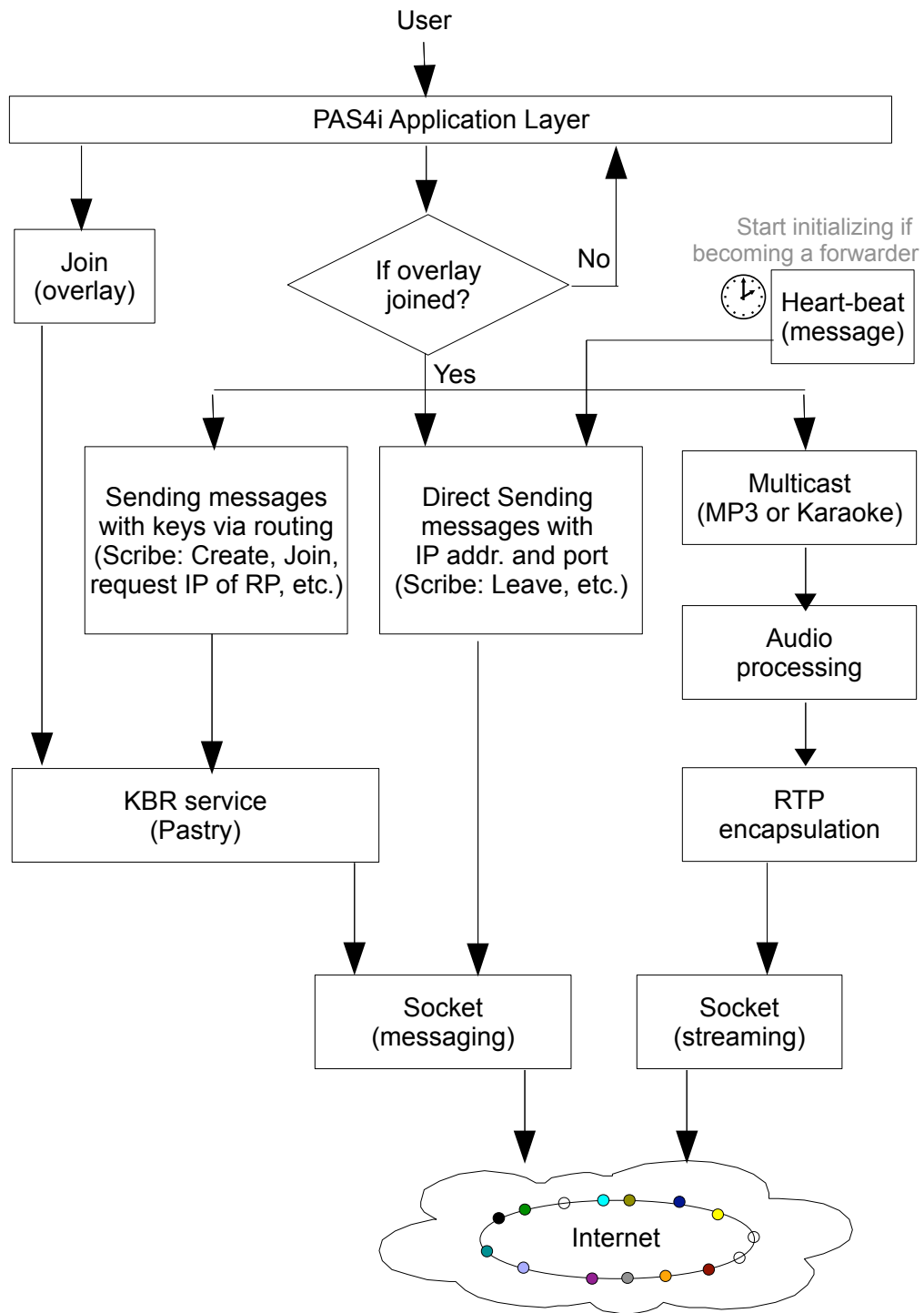


Figure 51: Communication protocol at a source sending node

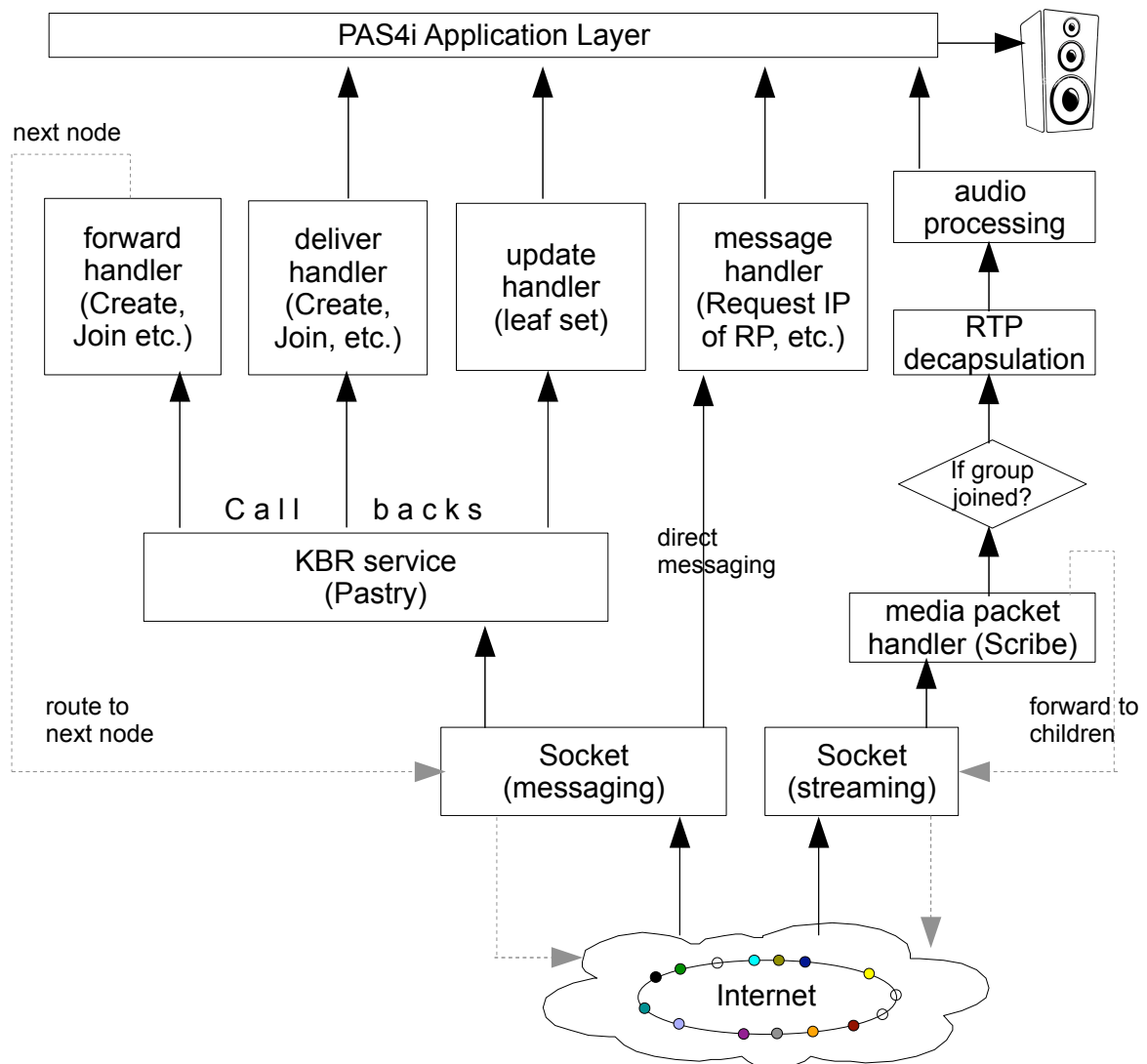
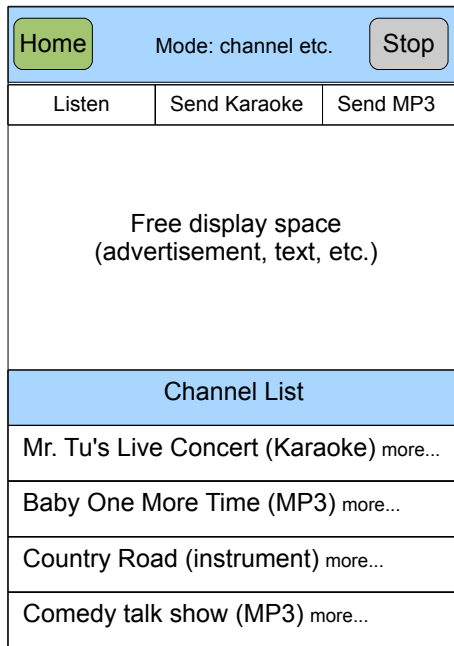
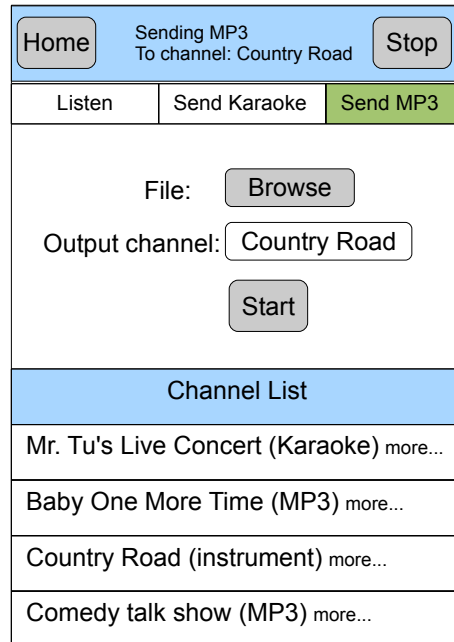


Figure 52: Communication protocol at a forwarding node or a destination node

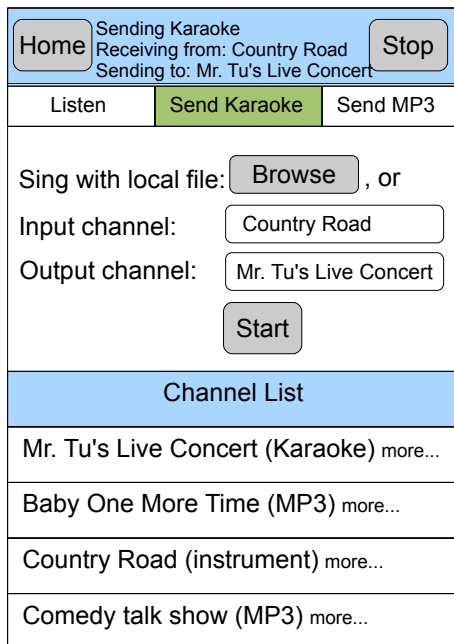
Bootstrapping - overlay joining On the application startup, not only does the home window (figure 53 a) appear, but also the overlay joining process is executed. At first, the KBR service of Pastry including the network socket interfaces, and other related components are initialized. The PAN4i can store the addresses and port numbers of some permanent bootstrapping peers on the overlay in a Extensible Markup Language (XML) file. For the overlay joining process, the local peer can send the PASTRY_JOIN message to any of these permanent bootstrapping peers. These permanent bootstrapping peers can be updated after each time the user runs the application.



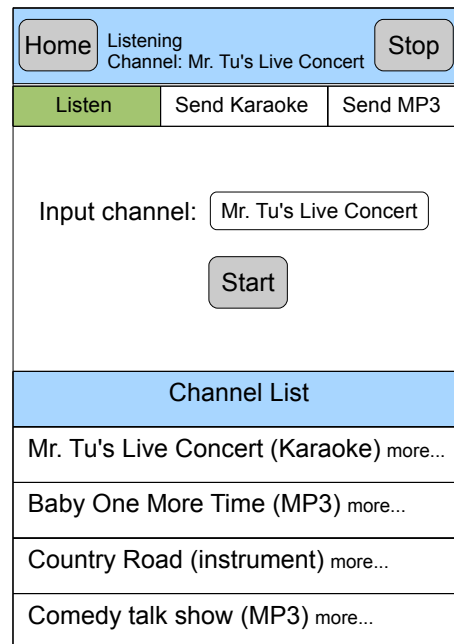
a) Home window



b) Window for sending MP3



c) Window for sending Karaoke



d) Window for listening MP3 or Karaoke

Figure 53: PAN4i GUI

Initializing and updating the channel list The PAN4i system may have many user channels and one system default channel (e.g., a startup channel). Right after joining to the overlay on startup, all peers will have to send the SCRIBE_JOIN message to this system default channel. This channel is used to inform all peers in the system information about all the available channels currently operating on the system. When a new channel is created or a channel is discarded, it is informed to all peers in the system via sending a multicast message to the RP of this system default channel. All peer in the system will cache these up-to-date channel list information. When a new peer arrives and is connected to an existing peer, this existing peer will send to the new peer its cached channel list, and the new peer can initialize the list the available channels in the system.

Multicasting a MP3 stream read from a local file After joining the overlay and initializing the channel list, a user can select the "Send MP3" tab (green highlight in figure 53 b) to open the window for sending MP3. The user will touch on the Browse button to select a MP3 file in the music library on the local file system. Then the user needs to select a channel name (e.g., Country Road via selecting the item on the channel list or typing the name to the text field manually) to that the MP3 is streamed. This channel can be a new or an existing one. When the start button is selected, the key of this channel name is calculated. If this channel is a new one, the SCRIBE_CREATE message with the channel key is routed to a RP. If it is an existing one, the REQ_IP is routed to the RP. Until, the local peer receives the responded IP address and port number of the RP, this local peer can now call the multicast function to read and encapsulate the music stream into RTP packets, and send out directly to the RP.

Multicasting a mixed Karaoke stream The Karaoke stream is the mixed stream of the recorded singing voice and a local music stream or a network receiving music stream. After joining the overlay and initializing the channel list, a user can select the "Send Karaoke" tab (green highlight in figure 53 c) to open the window for sending Karaoke. For doing this, there are two options to get the background music. The user can produce the Karaoke stream from a local music file via touching on the browse button and selecting a MP3 music file for Karaoke. Or the user can join to any channel that streams music for singing Karaoke via entering the input channel name in the text field "Input channel". Before multicasting the Karaoke stream, the "Output channel" text field has to be filled with another channel name. When the start button is selected, the keys of the two channel are calculated. The SCRIBE_JOIN message with the input channel key is routed to join to an existing streaming channel. The procedure for the output channel routing is similar to the above described procedure for multicasting a MP3 stream. Then the audio processing unit for Karaoke singing is initialized (if it has not yet been done). The receiving MP3 stream has to be converted into the PCM format for mixing with the PCM recorded singing voice. The mixed PCM Karaoke stream is then converted to MP3 format, and encapsulated in RTP packets for multicasting

to another channel. Figure 54 illustrates the audio processing graph for producing the mixed Karaoke stream.

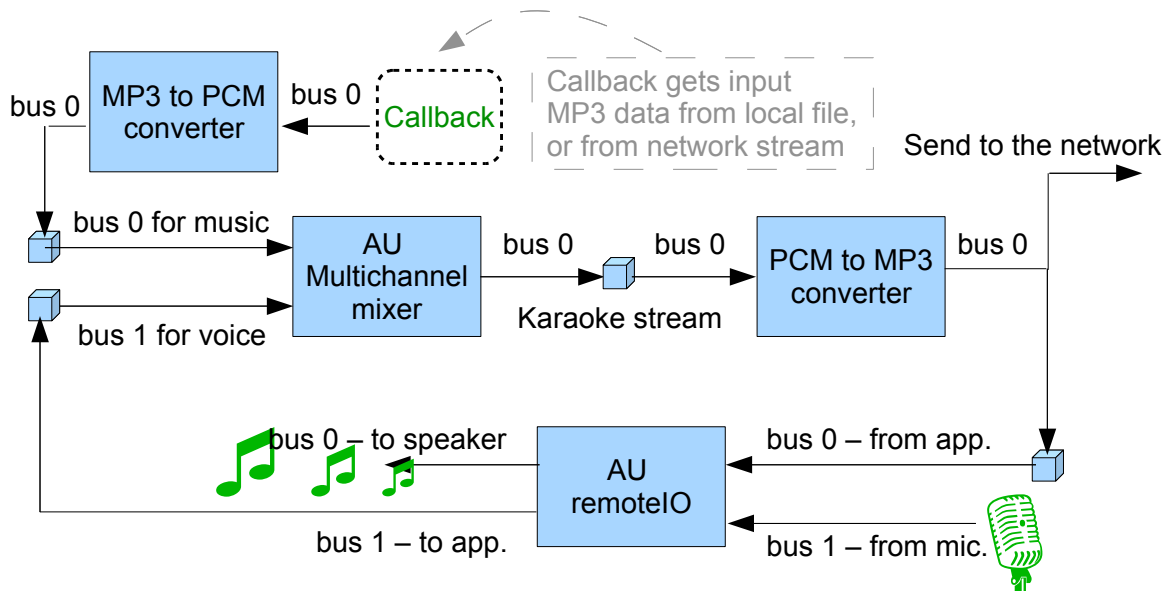


Figure 54: Communication protocol at a source sending node

Listening to a music or Karaoke stream After joining the overlay and initializing the channel list, the user can select the "Listen" tab (green highlight in figure 53 d) to listen on a selected channel. To do this, the user will type the channel name in the text field or touch on an item in the channel list so that the channel name is filled in the text field automatically. By selecting the start button, the channel key is calculated, the SCRIBE_JOIN message with this key is routed. And this peer is connected to the streaming group of this channel, and then the user can initialize the audio processing unit (if it has not been done yet) for playback the receiving network stream.

Changing to another channel during playback During listening on the selected channel, the user may select another channel in the list. With this action, in the background process, the local peer will check if it stays on the leaf level of the tree or on an intermediate level. If it is on the leaf level, it will directly send a SCRIBE_LEAVE message to its parent. If it is on an intermediate level (i.e., it has a children list of this group), it will not send this leave message. It still has to receive the stream and forward it to the children, but it will stop processing the audio stream for playback. Then the SCRIBE_JOIN message with the new channel key is routed to connect this local peer to another streaming group tree. On success, the peer can playback the new channel stream.

Stopping the currently Listen, or Send Karaoke, or Send MP3 play mode When the user is in one of the three above described play modes, if the stop button is selected, the application will stop running on this mode, de-allocate the initialized related audio units, free unused memories, and go to the home window. If the local node is the subscriber of a streaming group, the leaving procedure is carried out as described in the above procedure for changing to another channel. If it is a source sender of a streaming group, it will send a group destroy message to the RP, this RP will then forward this message to its children. And each child node repeatedly forwards this message to its children. In this way the streaming tree can be destroyed.

Other background running processes These processes are run on different background threads infinitely and independently from an user interaction on the GUI. They are the process for Pastry KBR routing service, a process for periodically sending heart-beat messages to children nodes of a streaming group which is used for detecting node failures or for discarding a streaming group in a RP and destroying the streaming tree when after sometime it receives no multicast message from any source sender. Another process waits for if there is any message or audio packet arriving at the incoming sockets.

Turning off the PAN4i application When the user is in one of the three above described playing modes, if the quit button of the iPhone or iPod Touch device is pressed, the peer will send SCRIBE_LEAVE messages to its parent node, and children nodes. The children nodes in this case have to look for their new parent node. All the allocated memories are de-allocated. Any operating state (e.g., permanent bootstrapping peers, history of visited channels, favorite channels etc.) may be stored for the next application startup.

5 PAN4i Implementation

Since PAN4i employs two open source projects Chimera [18] and oRTP [19] which are written in C, its implementation is mainly in C, and partly in Objective-C for the GUI implementation.

PAN4i implementation follows the Dabek model, and the functional units of this software (described in the software concepts section 4.3) are categorized and implemented on each tier. This chapter is a report on the current implementing state of PAN4i. On each tier, the already implemented functional units and others that are for future work will be presented.

5.1 Pastry - KBR

The software employs the Chimera open source project for providing its KBR service. According to the Chimera project page [18], "*Chimera is a light-weight C implementation of a "next-generation" structured overlay that provides similar functionality as prefix-routing protocols Tapestry and Pastry*". Basically, Chimera provides full features that described in the Pastry-KBR protocol (section 2.2.3), except that it does not provide the proximity feature of Pastry. Chimera communicates using UDP messages sent over BSD sockets. Chimera API follows the syntax defined in the Dabek model section 2.3, and is divided into three parts: the basic interface, the message Up-call interface, and the routing interface. Figure 55 shows the system design architecture of Chimera.

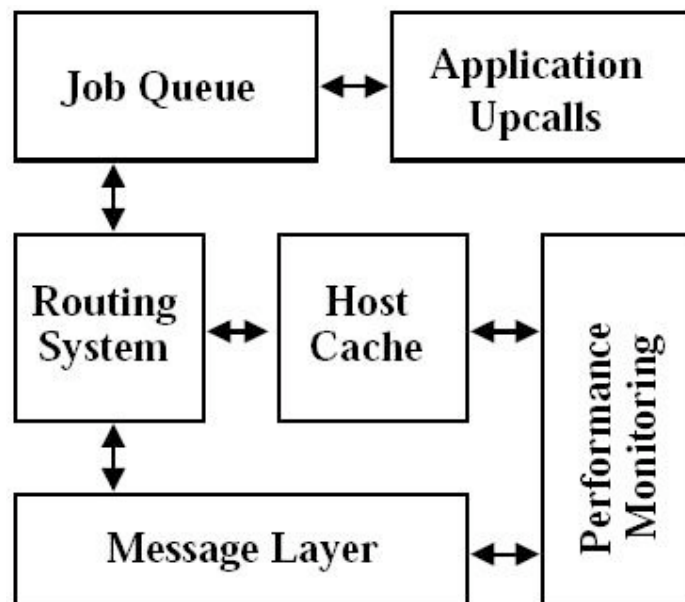


Figure 55: Chimera system design architecture (source [18])

The basic interface provides functions for initializing the Chimera overlay (`chimera_init`), joining to an existing Chimera overlay (`chimera_join`), assigning key to a node (`chimera_setkey`), etc., and especially Chimera allows developers to define message types that the KBR service is used for sending and receiving their messages. To register message types that will use the KBR service (i.e., on an arrival of a message, KBR will up-call the forward, deliver, or update callback handler function), the `chimera_register` function can be used. Chimera also allows any callback functions and message types that are registered for each of these callback functions to be defined. On the arrival of these messages, the messages are passed to their registered callback functions and not to the KBR service. The `message_handler` function is used to register a message type to any user defined callback function.

The message Up-call interface Supposing that the three callback functions `forward`, `deliver`, and `update` have been implemented on Scribe, this interface provides the functions for registering these three callbacks for up-calling to the KBR service. An example for the implementation of the `deliver` callback on Scribe in PAN4i is in listing 1.

The routing interface allows the application to access the routing state and pass down the routing preferences to the KBR. By invoking `route_lookup`, a list of possible next nodes on toward a given key is returned. By calling `chimera_route` with a given message and a key, the KBR will forward the message to the destination key. This call will cause either the `forward`, or `deliver` up-call at each receiver node. Calling `route_neighbors` will return the closest nodes in the leaf set of the local node.

Future work The proximity functionality may be implemented for Chimera. Besides, Chimera claims that its current implementation needs further improvement, e.g., Chimera processes the incoming messages in the order of first-in-first-out, and uses a simple algorithms for calculating latency with inaccurate prediction. It suggests that the improvement can be done in assigning different priorities for the message processing, and in using more effective and complex methods for predicting network performance. The bootstrapping mechanism of Chimera overlay also needs to be implemented.

5.2 Scribe - ALM

The current Scribe implementation state in PAN4i is that the group management and multi-cast messages dissemination functions have been implemented. Whereby, the tree repairing functions have not yet implemented and are considered for future work.

```
1  /*
2  * Invoked by the KBR on arrival of the KBR registered message types
3  * Specific handler function of each message type is further invoked
4  *
5  * @param *state: the overlay Chimera operating state of this peer
6  * @param *msg: the delivered message
7  */
8  void delivery_handler(ChimeraState *state, Message *msg) {
9      switch (msg->type) {
10         case SCRIBE_CREATE:
11             create_deliver(state, msg);
12             break;
13         case SCRIBE_JOIN:
14             join_deliver(state, msg);
15             break;
16         case SCRIBE_LEAVE:
17             leave_deliver(state, msg);
18             break;
19         case SCRIBE_REQUEST_RP_IP:
20             respondIPAddress(state, msg);
21             break;
22         default:
23             break;
24     }
25 }
```

Listing 1: Deliver callback invoked by the KBR

```

1  /*
2  * Invoked at sender to join a streaming group
3  *
4  * @param *state: the routing state of the local node
5  * @param groupID: the ID of the joining group
6  */
7  void join(ChimeraState *state, Key groupID) {
8      ChimeraGlobal *chglo = (ChimeraGlobal *)state->chimera;
9      char s[256];
10     int len;
11     //current host encode string "key:hostname:port"
12     host_encode(s, 256, chglo->me);
13     len = strlen(s) + 1;
14     //create a SCRIBE_JOIN message and send it using the KBR
15     chimera_send(state, groupID, SCRIBE_JOIN, len, s, SEQNUM);
16 }

```

Listing 2: Join function for sending SCRIBE_JOIN message

Group management implementation includes `create`, and `create_deliver` for sending and deliver handling the SCRIBE_CREATE messages, `join`, `join_forward`, and `join_deliver` for sending, forward handling and deliver handling the SCRIBE_JOIN messages. Moreover, `leave`, and `leave_deliver` are used for sending and deliver handling the SCRIBE_LEAVE messages. Listing 2, 3, and 4 demonstrate implementation for the SCRIBE_JOIN messages.

On line 15 of listing 2, the `chimera_send` function of the Chimera KBR is invoked to create a SCRIBE_JOIN message with the providing parameters, and then perform a lookup routing via the `chimera_route` sub-call.

The if, else condition on line 17 and 23 of listing 3 checks if the joining group is known by this local node. If not (`group == NULL`), this new group is added with its first child (the message sender) is added to this local node. If it is true (executing the else block), the message sender is simply added to the children list of this group on the local node, and the SCRIBE_JOIN message routing terminates here.

Listing 4 is the group joining handler function of the deliver callback invoked by the KBR. Since the key of this message is the group ID, this function should only be invoked at the RP of this group ID. The `addNewHostToList` function call in this listing adds the message sender to the children list of the requested joining group.

This current version does not support the initializing and updating the channel list (new channel group announcement). This function will be supported in the next version of PAN4i.

```

1  /*
2  * This function is the handler function for the forward up-call
3  * which is invoked by the KBR before sending the msg to the
4  * next forwarding node
5  *
6  * @param *state: the routing state of the local node
7  * @param **message: the SCRIBE_JOIN message
8  * @param **nextHost: the intended next forwarding node
9  */
10 void join_forward(ChimeraState *state, Message **message, ChimeraHost ←
    **nextHost) {
11     ChimeraGlobal *chglo = (ChimeraGlobal *) state->chimera;
12     Message *msg = *message;
13     ChimeraHost *nextNode = *nextHost;
14     ScribeGroup *group = NULL;
15     ChimeraHost *host = host_decode(state, msg->payload);
16     group = getGroupFromList(state, msg->dest);
17     if(group == NULL){ //this group does not exist, create it here
18         group = addNewGroupToList(state, msg->dest, nextNode, host);
19         //update this node is now the sender of the SCRIBE_JOIN msg to the ←
           nextNode
20         key_assign(&(msg->source), chglo->me->key);
21         host_encode(msg->payload, 256, chglo->me);
22     }
23     else{ //if the group is known, then add the message sender to its ←
           children list
24         addNewHostToList(state, host, msg->dest);
25         *nextHost = NULL;
26     }
27 }

```

Listing 3: Join forward handler invoked by the forward callback


```

1  /*
2  * Invoked when the join message is delivered at the RP of the group
3  * The direct sender of this msg is added to the children list of this ←
4  *   group
5  *
6  * @param *state: the routing state of the local node
7  * @param *msg: SCRIBE_JOIN message its payload is of form "keystring: ←
8  *   hostname:port"
9  */
10 void join_deliver(ChimeraState *state, Message *msg){
11     ChimeraGlobal *chglo = (ChimeraGlobal *) state->chimera;
12     if(key_equal(msg->source, chglo->me->key)){
13         addNewHostToList(state, chglo->me, msg->dest);
14     }
15     else{
16         ChimeraHost *host = host_decode(state, msg->payload);
17         addNewHostToList(state, host, msg->dest);
18     }
19 }

```

Listing 4: Join deliver handler invoked by the deliver callback

Multicast messages dissemination includes `requestIPAddress`, and `respondIPAddress` for requesting and responding the IP address and port number of a RP of a given group ID. After receiving the address of the RP, the requester can multicast a MP3 stream from a local file to this RP via executing the multicast function in a separate thread.

```

pthread_t tid;
pthread_create (&tid, NULL, multicast, (void *) state);

```

The `multicast` parameter in the `pthread_create` function is the function pointer where this `multicast` implementation is executed. Audio packets are sent to the RP via using the `rtp_session_send_with_ts` function. This function encapsulates the audio packets under RTP packets and send them to the RP. Furthermore, for sending live karaoke stream, every time when the mixed karaoke packets are rendered, they are directly sent to the streaming group. For handling the receiving stream, the `rtp_mcast_handler` is invoked on the arrival of the RTP audio packets. This handler will forward these packets to the children of the streaming group. If the local node is also a subscriber of the group, these packets are sent to the `rtp_session_rtp_parse` function for re-ordering the packets using the adaptive jitter buffer implemented in this function. Then these re-ordered RTP are sent to the `rtp_session_rtp_parse` function for decapsulating the RTP packets to audio packets

(i.e., MP3 packets) for further processing. The mentioned interfaces for dealing with RTP and jitter buffer are provided by the open source oRTP library [19]. For more detail about these implementation, please refer to the project source code on the CD that is included on the back of this thesis.

Repairing the multicast tree This functional unit has not yet been implemented, and is considered for the future work of PAN4i. Fundamentally, it includes a function for sending LEAVE messages on node graceful departures, and a function for sending heart-beat messages to children and parent of a all the streaming group available on this departing node. In a period of time, if the local node does not forward any multicast message to its children, it will send the heart-beat messages to them. This function is used to detect node unexpected departures or failures. That is, if the parent node does not receive and heart-beat message from a child node, it will remove this child from its streaming group children list. And if a child does not receive any multicast or heart-beat message from its parent, it will look for another parent. RP nodes also need to have a function to detect if there is no multicast sender for their hosting group. If this is the case, the RP will send tree-destroy messages to its children, so that the non-operating streaming tree will be discarded. In case of the RP node failures, or a new node becomes the new RP of an existing group, the updateHandler callback (invoked by the KBR if the leaf set of a node is updated) have to be implemented. This callback will provide appropriate actions in order to copy the RP group state of a RP node to the nodes in its leaf set. Further implementation would consider the fan-out degree that a node with its limited bandwidth can support, etc.

5.3 Application Layer

Audio processing The concepts and related interfaces for the iPhone audio processing used in PAN4i have been described in section 3. All the discussed audio units in this section are employed in this software, except that the hardware-assisted codecs and software-codecs. Listing 5 shows the playback callback implementation that can playback the receiving karaoke stream (from line 9 to 18), or render the mixed karaoke packets from the down sampling converter output (line 22). These live performing mixed karaoke packets are sent to the playback buffer to the loudspeaker, and also to the RP via using the `rtp_session_send_with_ts` function of the oRTP library (line 37).

For the future work, the PAN4i does want to implement a suitable codec for its karaoke stream. MP3 is a suitable codec for music streaming. But since iPhone OS provides only the MP3 decoder but not its encoder, and Apple claims that encoding MP3 on the iPhone would be an expensive process for its resource (e.g., battery), therefore another codec for the karaoke stream may be considered (e.g., Ogg Vorbis [58]). Currently, PAN4i supports karaoke streaming via playing the background music for singing on a local file, and not on

```

1  static OSStatus playbackCallback(void *inRefCon,
2      AudioUnitRenderActionFlags *ioActionFlags,
3      const AudioTimeStamp *inTimeStamp, UInt32 inBusNumber,
4      UInt32 inNumberFrames, AudioBufferList *ioData) {
5      OSStatus err = noErr;
6      ChimeraState *state = (ChimeraState *)inRefCon;
7      char *frameBuffer = ioData->mBuffers[0].mData;
8      memset(frameBuffer, 0, ioData->mBuffers[0].mDataByteSize);
9      if (state->myAudio->runMode == 1) { //karaoke listening mode
10         if (state->myAudio->readStart) {
11             memcpy(frameBuffer, state->myAudio->receiveBuffer[state->myAudio ←
12                 →readInd],
13                 ioData->mBuffers[0].mDataByteSize);
14             state->myAudio->readInd++;
15             if (state->myAudio->readInd == kNumAQBufs){
16                 state->myAudio->readInd = 0;
17             }
18         }
19     }
20     if (state->myAudio->runMode == 2) { //karaoke sending mode
21         //render karaoke packet from the down sampling converter
22         //and copy it the playback buffer (ioData)
23         err = AudioUnitRender(state->myAudio->downConverterUnit, ←
24             ioActionFlags, inTimeStamp, 0, inNumberFrames, ioData);
25         ChimeraGlobal *chglo = (ChimeraGlobal *)state->chimera;
26         int gKeyLen = strlen(chglo->me->sentGroupID) + 1;
27         size_t bytesRead = ioData->mBuffers[0].mDataByteSize + gKeyLen;
28         //initialize sendBuffer the first time, or the size is changed
29         if (state->myAudio->sendBuffer == NULL || state->myAudio-> ←
30             sendBufferLen != bytesRead) {
31             if (state->myAudio->sendBuffer != NULL) {
32                 free(state->myAudio->sendBuffer);
33             }
34             state->myAudio->sendBufferLen = bytesRead;
35             state->myAudio->sendBuffer = calloc(1, bytesRead);
36             memcpy(state->myAudio->sendBuffer + ioData->mBuffers[0]. ←
37                 mDataByteSize, chglo->me->sentGroupID, gKeyLen);
38         }
39         //sending karaoke packet to the RP using oRTP interface
40         memcpy(state->myAudio->sendBuffer, ioData->mBuffers[0].mData, ←
41             ioData->mBuffers[0].mDataByteSize);
42         rtp_session_send_with_ts(state->session, state->myAudio-> ←
43             sendBuffer, bytesRead, state->myAudio->user_ts);
44         state->myAudio->user_ts += ioData->mBuffers[0].mDataByteSize;
45     }
46     return err;
47 }

```

Listing 5: Implementation of the remotelO playback callback function

the remote receiving stream for singing. Singing on a remote receiving stream would be supported in the next version of PAN4i.

Graphical User Interface For our testing purpose, a simple GUI is created for PAN4i. The GUI is programmed in Objective-C language. In the future work, this current GUI needs to be replaced by another better user friendly GUI. One possible GUI design is described in figure [53](#). For the description of how to use the current GUI to test the application, please refer to the next testing chapter [6](#).

6 Testing

6.1 Test setup and running

Although all features of the PAN4i described in the software concepts and design section have not been fully implemented, its key functionalities for the streaming group management and the multicast function for live audio streaming on the overlay have been implemented (section 5), and shown to the visitors of the "Nacht des Wissens 2009" (i.e., Night of Sciences 2009) event at the Hamburg University of Applied Sciences. In this chapter, the test setup and running on the current version of PAN4i, that was performed on this event, will be described.

Figure 56 illustrates the currently used PAN4i GUI for testing. For our testing, there are four iPod Touches connecting to the Internet via the W-LAN, and each one owns one public IP address and a port number. In this writing, we consider that the four devices have the following domain name and port number pairs: iPodA:portA, iPodB:portB, iPodC:portC, and iPodD:portD.

Step 1: Self-organizing an overlay network On application startup, the application GUI is displayed with its domain name on top. Supposing that, at the beginning, there is no existing overlay network. Peer A starts the overlay first by entering a selected port number, say portA, into the text field and pressing the Join button. This Join button is used for two purposes, i.e., to join to the overlay network and after that is used for joining to a streaming group. Since peer B wants to join to the overlay network created by A, supposing that peer B knows the address and the port number of peer A (e.g., via a bootstrapping mechanism), peer B user needs to type the "iPodA:portA portB" string into the text field and press the Join button. The meaning of this string is that peer B on portB sends an overlay JOIN message to the address iPodA:portA of the bootstrapping peer A. On success, peer B is now in the overlay with peer A. Later, both peers C and D come to join to the overlay using the the same procedure that described for peer B. Peers C and D can use peer A or B, those are already in the overlay, for their bootstrapping node. When all the four peers are in the overlay, they can create any streaming group, subscribing to any group, multicasting to any group, and leaving from any subscribed group as described in the next step.

Step 2: Streaming group management and multicast message dissemination Supposing that, peer B creates a streaming group (groupB) by entering "groupB" string into the text field and pressing the Create button. Now that, any peer can send multicast messages to this groupB. For example, peer B wants to send a MP3 stream to its created groupB, it enters "groupB" into the text field and presses the "MP3 Send" button (for the testing purpose, the MP3 file is selected by default). From now on, if any of peers A, C and D joins groupB (via entering "groupB" in the text field and pressing the Join button), they will receive and be able to listen to the MP3 stream. For sending or listening to a karaoke stream, one

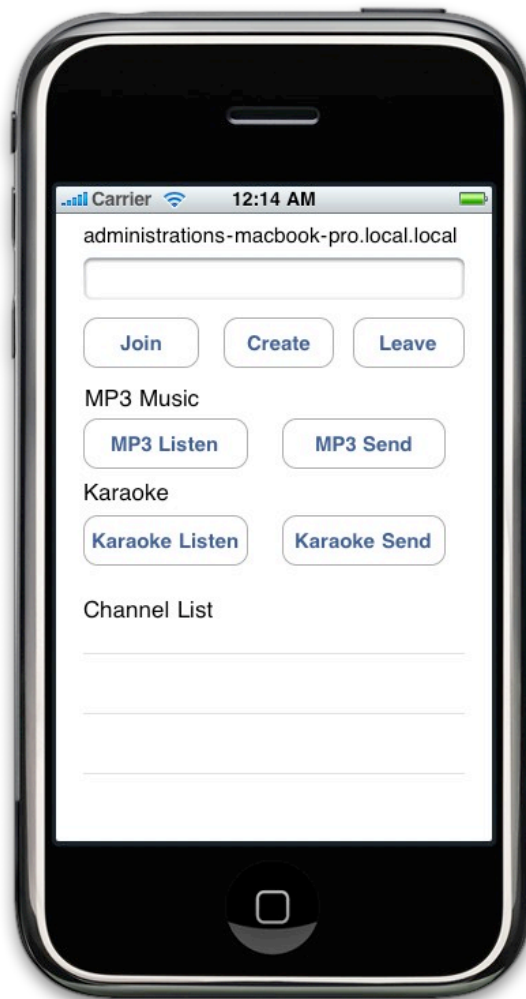


Figure 56: The PAN4i GUI for testing (source [32])

can do the same procedure as for MP3 music. To send karaoke, if using the iPod Touch, an external microphone has to be connected to the device, and the user has to sing on the playing background music of a local file (this local file is selected by default for the testing purpose). While being in groupB, any subscriber peer A, C, or D may not want to listen to the music stream and want to leave the group via entering "groupB" in the text field and pressing the Leave button. At the same time, on the overlay, many groups can exist (groupA, groupB, etc.). And for each group, it may have one or more source senders (i.e., for music streaming, it should have only one, but for a voice chat group, it has more than one sender).

6.2 PAN4i in the Nacht des Wissens 2009



Figure 57: PAN4i introduction slice



Figure 58: Establishing the PAN4i overlay and a streaming group

7 Conclusion and Outlook

The aim of this project is to create an entertainment platform for the iPhones and iPod Touches which is based on the ALM and overlay P2P network technologies. The entertainment art is in the form of listening and sending music or live performing karaoke in real-time. It also allows users to create any personal streaming channel where he or she can distribute his or her music to a group of audiences. The audiences would also be able give feedback to the Karaoke singers. Furthermore, the users could also be able to create chat or conferencing group. This system is designed to host a large number of users, and operating on the self-organizing of user nodes without installing a single server, so that the cost for running this system is zero.

To realize such a system, there are challenges that one will meet in the development process. Firstly, one will face the challenges in designing and implementing the overlay P2P network. It has to guaranty that the overlay works dynamically and precisely in case of node arrivals and departures. Secondly, the challenges are in the implementation of an ALM for audio streaming built on top of the overlay. The ALM for audio streaming has to be robust



Figure 59: Performing the PAN4i live karaoke streaming

under peer churn which happens very frequently in any P2P system. For the best delivery effort, proximity and heterogeneity properties of each node are considered as design strategies for the implementation. The challenges are also for using the right interfaces in order to have a wanted audio effect, especially those interfaces in the lower layer services of the iPhone Core Audio API. When doing audio processing, looking for the right documentation and example from Apple is sometime a challenge (in the time of this writing). From our experience, there is at the moment many books on the iPhone programming which cover many different interfaces for doing different tasks, but there is no or very limited information on the low layer audio processing techniques. Lastly, there are challenges for enhancing the QoS in the playback experience.

With the result from our work, we have shown the feasibility in realizing a large scale audio streaming application without any server deployment. PAN4i allows users to self-organize an overlay P2P network without any server interaction. Any user can receive and contribute music or live performed Karaoke stream to a shared, multi-channel application. Any user may select a song on the local file system, or play the song and mix with the singing voice to produce a live performed karaoke music for streaming to his or her own,



Figure 60: Explaining the PAN4i concepts to the visitors

personally created channel on the overlay network. All other users - when subscribing to this channel - will be able to receive and listen to live stream karaoke music.

PAN4i implementation follows the Three-tier Dabek model. On tier-0, the open source project Chimera is used to provide the KBR service. Chimera protocol and its KBR service are similar to Pastry, but it does not offer the Pastry proximity property. Upon Chimera, Scribe ALM for audio streaming is implemented in tier-1. The create, join, leave and multicast functions of Scribe are implemented. On the application layer (tier-2), via using the iPhone Core Audio API, the audio processing units for mixing Karaoke and playback are implemented. Furthermore, the open source library oRTP is used for integrating the adaptive jitter buffer into the application, and a simple GUI is created for application testing.

For the future work on this project, the following improvement can be considered. Firstly, the tree repairing function of Scribe including the replication of group state information on RP nodes to their leaf set nodes needs to be implemented. Having this, the streaming tree can operate and be robust under peer churn. Secondly, a suitable codec should be used for the

Karaoke stream (currently in PCM). Since the iPhone OS provides only MP3 decoder and not encoder, Apple also warns that the MP3 encoding process would be expensive for the battery resource, and MP3 codec is not free, so that other codecs may be in consideration (e.g., Ogg Vorbis [58]). To make the PAN4i more interesting and an interacting entertainment platform, one could add more functionalities to it. Sophisticated audio processing can be applied to produce good music, e.g., changing echo, key tone, etc. It may allow that the source senders send meta information about the singer (e.g., name, location, foto etc.), lyrics of the streaming song, and also allow audience to give feedbacks (e.g., grading, etc.) to the artists. A group communication function can be implemented for different purposes (e.g., voice chat, conferencing). The GUI needs to be improved. Since Skype claims that over 50% of residential computers are behind NATs or firewalls, in order to enable the PAN4i usage ability to a larger group of users, an approach for traversing NATs and firewalls should be implemented. To improve the streaming efficiency on which the QoS of playback partly depends, PAN4i should consider the proximity and heterogeneity properties of each peer on the overlay to construct efficient streaming trees, e.g., the fan-out degree of a peer, or the peer selection not only depends on the key, but also considers the power resource and bandwidth properties, etc.

When having all these features, PAN4i would be a very promising entertainment platform that would attract a huge number of users worldwide. The idea of having a live or archive audio/video streaming entertainment or communication platform, that it allows each user to create his or her own streaming channel, has been implemented in several existing systems, e.g., Youtube, Skype, USTREAM [59], Justin [60], etc., but the idea that PAN4i would like to bring to its users is the ability not only to be entertained, but also to entertain, to interact between audience and artists, to create personal styles and taste that are all carried out in real-time and mobility. For an usage scenario example that PAN4i can bring, one can create some kind like a "Germany Next Super Karaoke Star" contest on the PAN4i network, where there are candidates, examiners, and audience around the world. The candidate singing, examiner discussion and comments can be streamed to all audience live. And the statistical votes and feedbacks for each candidate from the audience can be soon resulted and available to all participants. PAN4i would be successful without or with very low deployment cost.

References

- [1] Thomas C. Schmidt, “The Mindstone projectpage,” <http://inet.cpt.haw-hamburg.de/projects/mindstone>, 11 2009.
- [2] YouTube, “The Youtube homepage,” <http://www.youtube.com>, 11 2009.
- [3] Spotify Ltd, “The Spotify homepage,” <http://www.spotify.com/>, 11 2009.
- [4] Zattoo, “The Zattoo homepage,” <http://zattoo.com/>, 11 2009.
- [5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2001, pp. 149–160.
- [6] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, ser. LNCS, vol. 2218. Berlin Heidelberg: Springer-Verlag, Nov. 2001, pp. 329–350.
- [7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, “A Scalable Content-Addressable Network,” in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 161–172.
- [8] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, “Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and,” University of California at Berkeley, Technical Report UCB/CSD-01-1141, April 2001.
- [9] Apple Inc., “The Apple Inc. homepage,” <http://www.apple.com/>, 11 2009.
- [10] —, “The iPhone homepage,” <http://www.apple.com/iphone/>, 11 2009.
- [11] —, “The iPod Touch homepage,” <http://www.apple.com/ipodtouch/>, 11 2009.
- [12] —, “iPhone, iPod Touch billion applications countdown,” <http://www.apple.com/itunes/billion-app-countdown/>, 11 2009.
- [13] Thomas C. Schmidt, “The Mobile Audio Video Network Hamburg projectpage,” <http://inet.cpt.haw-hamburg.de/projects/mobinet/>, 11 2009.
- [14] “The Skype homepage,” <http://www.skype.com>, 11 2009.

-
- [15] Thomas C. Schmidt, "The Moviecast projectpage," <http://www.realmv6.org/moviecast.html>, 11 2009.
- [16] The Unofficial Apple Weblog (TUAW), "iPhone gets native P2P torrent software," <http://www.tuaw.com/2008/03/02/iphone-gets-native-p2p-torrent-software/>, 11 2009.
- [17] iphonefreakz.com, "iSlk P2P music sharing on your iPhone or iPod Touch," <http://iphonefreakz.com/2008/05/14/islisk-p2p-music-sharing-on-your-iphone-or-ipod-touch/>, 11 2009.
- [18] K. Puttaswamy *et al.*, "Chimera - a light-weight C implementation of a "next-generation" structured overlay that provides similar functionality as prefix-routing protocols Tapestry and Pastry," <http://current.cs.ucsb.edu/projects/chimera/>, 11 2009.
- [19] Linphone.org, "Open source package for RTP/RTCP," http://www.linphone.org/index.php/eng/code_review/ortp, 12 2009.
- [20] S. Deering, "Host extensions for IP multicasting," IETF, RFC 1112, Aug. 1989, updated by RFC 2236.
- [21] Thomas C. Schmidt, "Lecture slides on group communication in Internet," <http://inet.cpt.haw-hamburg.de/teaching/ws-2009-10/technik-technologie/mcast.pdf>, 11 2009.
- [22] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "Scribe: The Design of a Large-Scale and Event Notification Infrastructure," in *Networked Group Communication. Third International COST264 Workshop, NGC 2001. Proceedings*, ser. LNCS, J. Crowcroft and M. Hofmann, Eds., vol. 2233. Berlin Heidelberg: Springer-Verlag, 2001, pp. 30–43.
- [23] Thomas C. Schmidt, "Lecture slides on Introduction to Peer-to-Peer Systems," <http://inet.cpt.haw-hamburg.de/teaching/ws-2009-10/internet-technologies/intro-p2p.pdf>, 11 2009.
- [24] —, "Lecture slides on Structured Peer-to-Peer Networks," <http://inet.cpt.haw-hamburg.de/teaching/ws-2009-10/internet-technologies/structured-p2p-p.pdf>, 11 2009.
- [25] K. Katrinis and M. May, "Application-Layer Multicast," in *Peer-to-Peer Systems and Applications*, ser. LNCS, R. Steinmetz and K. Wehrle, Eds. Berlin Heidelberg: Springer-Verlag, 2005, vol. 3485, ch. 11, pp. 157–170.
- [26] S. Rhea *et al.*, "Bamboo-DHT," <http://www.bamboo-dht.org/>, 11 2009.

- [27] P. Druschel *et al.*, “FreePastry,” <http://www.freepastry.org/>, 11 2009.
- [28] I. Baumgart, B. Heep, S. Krause, and S. Mies, “The OverSim P2P Simulator,” <http://www.oversim.org>, 11 2008.
- [29] S. Götz, S. Rieche, and K. Wehrle, “Selected DHT Algorithms,” in *Peer-to-Peer Systems and Applications*, ser. LNCS, R. Steinmetz and K. Wehrle, Eds. Berlin Heidelberg: Springer-Verlag, 2005, vol. 3485, ch. 8, pp. 95–117.
- [30] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, “Towards a Common API for Structured Peer-to-Peer Overlays,” in *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003, Revised Papers*, ser. LNCS, M. F. Kaashoek and I. Stoica, Eds., vol. 2735. Berlin Heidelberg: Springer-Verlag, 2003, pp. 33–44.
- [31] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, “A Survey of Application-Layer Multicast Protocols,” *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 58–74, 2007.
- [32] J. Liu, S. Rao, B. Li, and H. Zhang, “Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast,” in *Proceedings of the IEEE*, 2008, pp. 11–24.
- [33] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, “Deployment Issues for the IP Multicast Service and Architecture,” *IEEE Network Magazine*, vol. 14, no. 1, pp. 78–88, 2000.
- [34] Y. Liu, Y. Guo, and C. Liang, “A survey on peer-to-peer video streaming systems,” *Peer-to-Peer Networking and Applications*, vol. 1, no. 1, pp. 18–28, Mar. 2008.
- [35] Y.-H. Chu, S. G. Rao, and H. Zhang, “A case for end system multicast,” in *Joint International Conference on Measurement and Modeling of Computer Systems*. New York, NY, USA: ACM, 2000, pp. 1–12.
- [36] V. Venkataraman and P. Francis, “Chunkyspread: Multi-tree Unstructured Peer-to-Peer Multicast,” in *Proceedings of the V. International Workshop on Peer-to-Peer Systems (IPTPS’06)*, 2006.
- [37] N. Magharei, R. Rejaie, and Y. Guo, “Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches,” in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1424–1432.
- [38] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” IETF, RFC 3550, Jul. 2003.

- [39] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)," IETF, RFC 3711, Mar. 2004.
- [40] Fraunhofer-Gesellschaft, "MP3 working principle," <http://www.iis.fraunhofer.de/EN/bf/amm/products/mp3/mp3workprinc.jsp/>, 11 2009.
- [41] "The Speex projectpage," <http://www.speex.org/>, 11 2009.
- [42] S. Andersen, A. Duric, H. Astrom, R. Hagen, W. Kleijn, and J. Linden, "Internet Low Bit Rate Codec (iLBC)," IETF, RFC 3951, Dec. 2004.
- [43] 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*, vol. 20, no. 8, pp. 100–110, 2002.
- [44] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan, and H. Zhang, "The Impact of Heterogeneous Bandwidth Constraints on DHT-Based Multicast Protocols," in *Peer-to-Peer Systems IV. 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005, Revised Selected Papers*, ser. LNCS, M. Castro and R. van Renesse, Eds., vol. 3640. Berlin Heidelberg: Springer-Verlag, 2005, pp. 115–126.
- [45] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Content Distribution in Cooperative Environments," in *Peer-to-Peer Systems II. Second International Workshop, IPTPS 2003 Berkeley, CA, USA, February 21-22, 2003 Revised Papers*, ser. LNCS, M. F. Kaashoek and I. Stoica, Eds., vol. 2735. Berlin Heidelberg: Springer-Verlag, 2003, pp. 292–303.
- [46] "Skype Statistical Analysis," <https://dl.getdropbox.com/u/223873/OnlineData.htm>, 11 2009.
- [47] Sugih Jamin, "Zattoo presentation slides for the Terena Networking Conference 2008," http://tnc2008.terena.org/core/getfile.php?file_id=459, 11 2009.
- [48] Hui Zhang, "The End System Multicast projectpage," <http://esm.cs.cmu.edu/>, 11 2009.
- [49] S. A. Baset and H. G. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006, pp. 1–11.
- [50] M. Wählisch and T. C. Schmidt, "Multicast Routing in Structured Overlays and Hybrid Networks," in *Handbook of Peer-to-Peer Networking*, X. Shen, H. Yu, J. Buford, and M. Akon, Eds. Berlin Heidelberg: Springer Verlag, January 2010, to appear. [Online]. Available: <http://www.springer.com/computer/communications/book/978-0-387-09750-3>

-
- [51] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.
- [52] Apple Inc., "iPhone OS Technology Overview," <http://developer.apple.com/iphone/library/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview>, 11 2009.
- [53] —, "Core Audio Overview," <http://developer.apple.com/iphone/library/documentation/MusicAudio/Conceptual/CoreAudioOverview/Introduction/>, 11 2009.
- [54] —, "Audio Queue Programming Guide," <http://developer.apple.com/mac/library/documentation/MusicAudio/Conceptual/AudioQueueProgrammingGuide/>, 11 2009.
- [55] —, "Audio File Stream Services Reference," <http://developer.apple.com/mac/library/documentation/MusicAudio/Reference/AudioStreamReference/>, 11 2009.
- [56] —, "Audio Unit Processing Graph Services Reference," <http://developer.apple.com/iphone/library/documentation/AudioToolbox/Reference/AUGraphServicesReference/>, 11 2009.
- [57] —, "Audio Queue - Offline Rendering (Technical Q&A QA1562)," <http://developer.apple.com/iphone/library/qa/qa2009/qa1562.html>, 11 2009.
- [58] Xiph.Org, "The audio Ogg Vorbis projectpage," <http://www.vorbis.com/>, 12 2009.
- [59] Ustream.TV, "USTREAM, Free Live Video Streaming, Online Broadcasts," <http://www.ustream.tv/>, 12 2009.
- [60] Justin.tv, "Live Video and Chat for Everyone," <http://www.justin.tv/>, 12 2009.

List of Figures

1	Unicast (source [21])	5
2	Broadcast (source [21])	6
3	Multicast (source [21])	6
4	Client-Server Topology (source [23])	8
5	Centralized P2P Topology (source [23])	9
6	Pure P2P Topology (source [23])	9
7	Hybrid P2P Topology (source [23])	10
8	Complexity of different P2P models (source: [24])	11
9	Use case scenario in DHT-based P2P	13
10	Comparison Table for Client-Server and different P2P models (source: [23])	15
11	A 4-bit Pastry identifier space with six keys mapped onto five nodes. Numeric closeness is an ambiguous metric for assigning keys to nodes as illustrated for key K03 (source: [29])	16
12	Pastry node state for node 103220 in a 12-bit identifier space and a base of 4 ($k = 12$, $b = 2$). The routing table lists nodes with the length of the common node identifier prefix corresponding to the row index. (source: [29])	17
13	Basic abstractions and APIs, including Tier 1 interfaces: distributed hash tables (DHT), decentralized object location and routing (DOLR), and group any-cast and multicast (CAST) - (source from [30])	21
14	a) IP multicasting scenario and b) an overlay multicast tree (sender S, router R, destination D) (source [31])	23
15	Taxonomy of architectures for ALM (source [32])	24
16	a) A graph with link costs; b) shortest path tree; c) minimum spanning tree (source [31])	25
17	A hierarchical cluster of nodes with cluster size 4 (source [31])	26
18	Nodes of a multicast streaming group in a single-tree	28
19	Scenario when a node leaves the multicast tree	29
20	Repaired tree after a node leaves	29
21	An example of multi-tree streaming topology	31
22	Narada - Optimizing Mesh Quality (source [35])	33
23	Peer list retrieval from the tracker server (source [34])	34
24	Buffer map exchange and data pull among peers (source [34])	35
25	A snapshot of the scheme at time 40. Users belonging to the same session form an application-level multicast tree together with the server. Users in session 3 have finished patch retrieval; while 3 clients in session 4 are still receiving the patch stream from their parent patch servers (source: [34])	37

26	DirectStream system. a) DirectStream system with two clusters – one headed by client A and the other headed by client F . b) DirectStream system after the departure of client A. No service from the server is required from now on (source: [34])	38
27	Scribe streaming tree built on the Pastry overlay P2P network	41
28	Scribe implementation of forward (source: [43])	42
29	Scribe implementation of deliver (source: [43])	43
30	Group management (creation) in Scribe	43
31	Joining a multicast group in Scribe	44
32	A case for a constructed Scribe multicaset tree	45
33	Sending multicast message in the Scribe multicast tree	46
34	Repairing the Scribe multicast tree	47
35	Skype Network. There are three main entities: supernodes, ordinary nodes, and the login server (source: [49])	51
36	Zattoo’s overlay topology-aware characteristic (source: [47])	52
37	Zattoo’s overlay topology-aware characteristic in case there are nodes behind NATs or Firewalls (source: [47])	53
38	The Zattoo Player (source: [47])	54
39	Steps to watch a channel in Zattoo (source: [47])	55
40	Zattoo’s underlay infrastructure architecture (source: [47])	56
41	Layers of iPhone OS (source [52])	58
42	The three API layers of Core Audio (source [53])	60
43	The playback process of AQS (source [54])	62
44	Converter unit	64
45	Multichannel mixer unit	65
46	I/O unit (or called RemotelIO unit)	65
47	Audio processing chain using Audio Unit Processing Graph interfaces	67
48	Usecase diagram of the PAN4i application	69
49	The main functional units and their dependences of PAN4i	76
50	The PAN4i Three Tiers Dabek model	77
51	Communication protocol at a source sending node	80
52	Communication protocol at a forwarding node or a destination node	81
53	PAN4i GUI	82
54	Communication protocol at a source sending node	84
55	Chimera system design architecture (source [18])	86
56	The PAN4i GUI for testing (source [32])	96
57	PAN4i introduction slice	97
58	Establishing the PAN4i overlay and a streaming group	98
59	Performing the PAN4i live karaoke streaming	99
60	Explaining the PAN4i concepts to the visitors	100

List of Tables

1	Comparison of central server, flooding search and distributed indexing (source: [25])	14
2	Conceptual comparison of IP multicast and ALM (source [31])	23
3	Audio codec comparison	40
4	iPhone OS audio formats supported by hardware-assisted codecs (source: [53])	67
5	iPhone OS audio formats supported by software codecs (source: [53])	67
6	iPhone OS: recording audio formats (source: [53])	68

Listings

1	Deliver callback invoked by the KBR	88
2	Join function for sending SCRIBE_JOIN message	89
3	Join forward handler invoked by the forward callback	90
4	Join deliver handler invoked by the deliver callback	91
5	Implementation of the remotelO playback callback function	93

Acronyms

PAN4i	Mobile P2P Audio Network for the iPhone and iPod Touch
OS	Operating System
GUI	Graphical User Interface
PC	Personal Computer
AQS	Audio Queue Services
TV	Television
P2P	Peer-to-Peer
ALM	Application Layer Multicast
DHT	Distributed Hash Table
RIAA	Recording Industry Association of America
TTL	Time To Live
API	Application Programming Interface
KBR	Key-based Routing
XML	Extensible Markup Language
VoIP	Voice over Internet Protocol
RP	Rendezvous Point

Declaration

I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Master report has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, December 7th, 2009

Tran, Thanh Minh Tu