Universitat Autònoma
de Barcelona

Departament d'Arquitectura de Computadors i

Sistemes Operatius

Màster en Computació d'Altes Prestacions

System Analysis of a Peer-to-Peer Video-on-Demand Architecture: Kangaroo

Memoria del trabajo de investigación del "Máster en Computación de Altas Prestaciones", realizada por ALVARO CHALAR ZÚÑIGA, bajo la dirección de ANA RIPOLL Presentada en la Escuela Técnica Superior de Ingeniería (Departamento de Arquitectura de Computadores y Sistemas Operativos)

2008

Trabajo de investigación

Máster en Computación de Altas Prestaciones

Curso 2007-08

Título System Analysis of a Peer-to-Peer Video-on-Demand Architecture: Kangaroo

Autor: Alvaro Chalar Zúñiga

Director: Ana Ripoll

Departamento Arquitectura de Computadores y Sistemas Operativos

Escuela Técnica Superior de Ingeniería (ETSE)

Universidad Autónoma de Barcelona

Firmado

Autor Director

A Zeynep…

…por aguantar todas mis manías

Çok teşekkür ederim!

Acknowledgments

First and foremost I want to thank to Emilio Luque and Dolores Rexachs (*a.k.a Lola*) for opening the ETSE doors for me and receiving me with their arms open.

I also want to show my appreciation to my tutor Ana Ripoll and Porfidio Hernández for their valuable help and guidance during the development of this dissertation.

Special thanks to Xiaoyuan Yang at Telefónica for making me see what brain-storming really is and for helping me with the conclusion of this work.

To all my friends at the CAOS Department, thank you for making me feel like home from day one, or maybe two…well you got the point!

Last but not least, to my wonderful family and friends spread around the world. I am grateful for your endless love and support; and for making distance……"*transparent to the user"*.

Muchas gracias,

Alvaro

Abstract

Architectural design and deployment of Peer-to-Peer Video-on-Demand (P2P-VoD) systems which support VCR functionalities is attracting the interest of an increasing number of research groups within the scientific community; especially due to the intrinsic characteristics of such systems and the benefits that peers could provide at reducing the server load. This work focuses on the performance analysis of a P2P-VoD system considering user behaviors obtained from real traces together with other synthetic user patterns. The experiments performed show that it is feasible to achieve a performance close to the best possible. Future work will consider monitoring the physical characteristics of the network in order to improve the design of different aspects of a VoD system.

Resum

El diseny arquitectònic i el desplegament de sistemes de Video Sota Demanda "Peer-to-Peer" que soporten funcionalitats VCR està captant l'interès d'un nombre creixent de grups de recerca a la comunitat científica, degut especialment a les caracteristiques intrinsiques dels mencionats sistemes i als beneficis que els peers podrien proporcionar a la reducció de la càrrega en el servidor. Aquest treball tracta l'analisis del rendiment d'un sistema P2P-VoD considerantel comportament d'usuaris obtingut amb traçes reals i amb patrons sintètics. Els experiments realitzats mostren que es viable asolir un rendiment proper al cas més óptim. Com treball futur es considerarà la monitorització de les caracteristiques físiques de la xarxa per a poder millorar el diseny dels diferents aspectes que formen un sistema de VoD.

Resumen

El diseño arquitectónico y el despliegue de sistemas de Video bajo Demanda “Peer-to-Peer” que soportan funcionalidades VCR está captando el interés de un número creciente de grupos de investigación dentro de la comunidad científica; especialmente debido a las características intrínsecas de tales sistemas y a los beneficios que los *peers* podrían proporcionar en la reducción de la carga en el servidor. Este trabajo se enfoca en el análisis de rendimiento de un sistema P2P-VoD considerando el comportamiento de usuarios obtenido de trazas reales, junto a otros patrones sintéticos. Los experimentos realizados muestran que es viable lograr un rendimiento cercano al caso más óptimo. El trabajo futuro considerará la monitorización de las características físicas de la red para poder mejorar el diseño de los diferentes aspectos que conforman un sistema de VoD.

Contents

List of Figures

Chapter 1

Peer-to-Peer Video-on-Demand Systems

Abstract

This chapter introduces the Peer-to-Peer (P2P) paradigm in Video-on-Demand (VoD) systems. It describes existing P2P media streaming systems, more specifically P2P Live Streaming and Video-on-Demand systems, providing a comparison of design requirements that influence their system architecture. A study of the most important challenges, current approaches for providing P2P-VoD services and how these systems manage to implement VCR functionalities is also presented along with their pros and cons. The chapter ends presenting the objectives of this dissertation.

1.1 Introduction

Peer-to-peer (P2P) systems have emerged as a new paradigm and have become very successful in distributing content by encouraging an important number of users to behave as both clients and servers, commonly known as *peers*. A peer is considered to be a provider and a consumer of resources, bringing along serving capacity into the system. These systems significantly reduce the load in the server, proving to be highly scalable when various peers have common media interests.

Unlike client-server based schemes, P2P systems are distributed systems consisting of nodes interconnected with each other, able to self organize into network topologies in order to share resources like content, bandwidth, CPU cycles and storage.

P2P technologies were designed initially for generic file distribution, such as the popular BitTorrent [1] and Emule [2] applications. In these file sharing applications, users need to wait until the complete video is downloaded in order to start watching it, causing long delays in most cases due to the intrinsic characteristics of the file, network bandwidth, and so on.

However, several P2P streaming systems have been deployed lately to offer live and on-demand video streaming services over the Internet. Systems like Coolstreaming [23] and PPLive [7] have been very successful delivering live media content over the Internet to a large number of users.

As a result, P2P-based VoD (P2P-VoD) emerged as a new challenge for researchers and P2P technology itself. These systems are much more challenging to design and deploy than live streaming approaches because a VoD service should allow users to arrive at arbitrary times to watch media content or

to use VCR-like functionalities, for example, while guaranteeing good performance at all times.

"*In general, the main challenge resides in designing systems that ensure that users can start watching a movie at any point in time, with small start-up times and sustainable playback rates*" [8]. Users are given the control of what to watch and when to watch it, without worrying about long start-up delays or low quality of service (QoS) issues.

Current studies focus on P2P-VoD systems that support VCR functionalities using different techniques, and considering the Internet always as a *cloud*. Our proposal comes from the idea that in order to provide more efficient system architectures, it is necessary to take into consideration the physical characteristics of the real network. Having precise information of the real network topology (i.e. what is behind the cloud), will help us determine new bottleneck locations, something that has not been considered in recent approaches.

In order to support VCR operations more efficiently, we believe that content-aware models which consider user-behavior according to the content being watched are necessary. Designing a P2P-VoD system with this kind of characteristics and information will improve better system performance allowing it to predict, more accurately, a jump operation before it even takes place.

1.2 P2P Live Streaming

As mentioned before, video streaming services can be divided into two groups: live streaming and video-on-demand streaming. Live streaming allows video content to be transmitted in real time to all requesting users. One or more users have their playbacks synchronized to provide their stored content to other peers. On the other hand, video-on-demand users have the flexibility to watch any

video at any moment in time, meaning that they do not need to synchronize their playback times. Moreover, they are capable to perform operations such as forward or backward on the media file.

This section gives a brief overview of the existing overlay network structures for P2P live streaming systems.

1.2.1 Tree-based systems

The tree-based P2P delivery has its origins in IP multicast [4], where a single block is transmitted from the source and replicated by the routers along a distribution tree rooted at the source node. In *Cooperative Networking (CoopNet)* [18], for example, each node in the tree forwards the received stream to each of its children. This way, the load on the server is alleviated by distributing content between cooperative users. The presence of a source node simplifies the task of locating content, since the root has all the essential information for constructing and maintaining the tree structure.

Although an efficient solution for streaming audio and video, the use of computation and bandwidth resources along with the complexity of transport control for multicast sessions resulted in router overhead, causing IP level multicast to never materialize entirely over the Internet. Consequently, an implementation was done at the application layer, commonly known as Application-Level Multicast or simply ALM.

- **Single-tree streaming**

Analogous to the scheme described above, users taking part of a streaming session can get together to form an ALM tree having the source video server as the root. Figure 1.1 shows the tree structure with peers distributed at different levels receiving and forwarding information in a top-bottom direction.

Overcast [15] and ESM [11] are considered pioneer works of single-tree based streaming. Both of them use a single ALM tree to distribute content; where each non-leaf peer in the tree has the function to retrieve and forward media content upon its arrival.

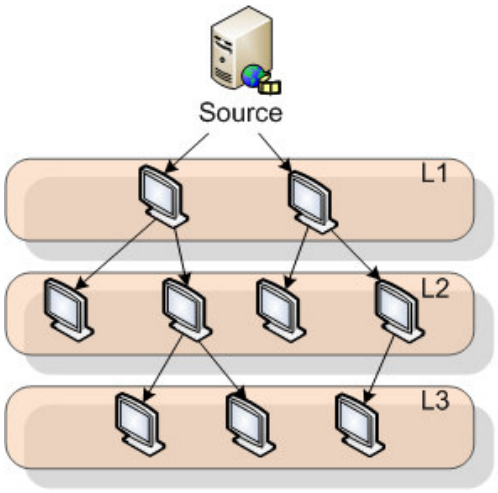


Figure 1.1 – Application Layer Multicast P2P video streaming

The figure above shows an ALM streaming tree with nine participating peers rooted at the source server node. The top level (L1), immediately below the root, is formed by only two peers receiving media content directly from the source server. At the same time, these two peers push the content one level below, towards L2, that includes four peers which receive the data, but only two of them forward it to end-leaf nodes located at the bottom level of the tree. Because leaf nodes usually account for a large portion of peers in the system, and they do not make any upload bandwidth contribution to it, the peer bandwidth utilization efficiency is affected to a great extent.

Considering that peers at lower levels are the last ones to receive video content, the construction of a streaming tree with fewest levels possible is preferred;

meaning that the tree topology should spread as much as possible at every level. Nonetheless, it is important to consider that a peer forming part of an internal node has uploading constraints which will determine its maximum fan-out degree.

Maintaining this type of tree structures is also very important since users forming part of a P2P streaming session are known to be very dynamic. Users can join and leave the session at any moment. Therefore, the sudden departure of one of these peers, either gracefully or unexpectedly (e.g. machine crashes), has a great impact in all its descendants since there is only one available path of streaming flow coming from the source. To handle this kind of disruption, the streaming tree needs to be recovered and recalculated by reassigning the affected nodes to the source server or other available peers.

Unfortunately, for a large streaming system, the source server becomes the performance bottleneck and single point of failure. To address this problem, distributed algorithms such as ZIGZAG [19] have been developed for constructing and maintaining streaming trees in a distributed manner. Despite of all the efforts, tree-based streaming has shown to be unable to recover fast enough to deal with frequent peer churn.

- **Multi-tree streaming**

Multi-tree based techniques such as Bullet [16] have been proposed to handle problems related to leaf nodes. In this approach, the source server divides the stream into multiple sub-streams, as shown in Figure 1.2. A single streaming tree is split into various sub-trees, one for each sub-stream. Each peer joins all sub-trees available to retrieve sub-streams and has different positions in each one of them.

The upload bandwidth of a peer is utilized to upload a sub-stream whenever it is considered to be an internal node in some sub-tree. In order to have a fully balanced multi-tree streaming, a single peer is positioned on an internal node (L2) in only one sub-tree (uploading one sub-stream from the level below), while positioned on a leaf node on the remaining sub-trees (downloading a sub-stream from the level above).

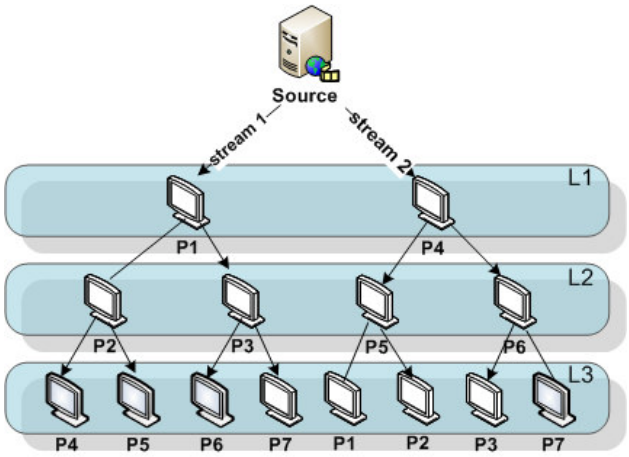


Figure 1.2 – Multi-tree based streaming

1.2.2 Mesh-based systems

Tree-based systems only allow peers to have one parent in a single streaming tree to download from; introducing a single point of failure. If the peer's parent leaves, the peer and all its children stop retrieving information until the peer connects to a different parent over again.

To deal with this problem, many P2P streaming systems adopt a mesh-based approach, such as DONet/Coolstreaming [23]. The main characteristic of mesh-

based streaming system is that there is no static streaming topology. At any given time, a peer collaborates with multiple neighboring peers, uploading/downloading video to/from multiple peers concurrently. Finding new peers to keep a desired level of connectivity is possible, and it is exactly what makes them particularly robust against peer churn.

Analogous to P2P file sharing systems like BitTorrent [1], a mesh streaming system includes a tracker to keep track of peer activities during the video session. Every new peer wanting to join the session needs to contact the tracker and report its own information (i.e. IP address, port number). Afterward, the tracker provides a list of peers containing the information of a random subset of active peers available. Using this list, the peer attempts to initiate peering connections; and if successful, it starts exchanging video content with its neighbors. To handle unexpected peer departures, peers regularly exchange keep-alive messages. At the same time, depending on system's peering strategies, a peer does not only connect to new neighbors in response to peer departures, but also when better streaming performance can be achieved.

In mesh-based systems, the concept of video stream becomes invalid due to the mesh topology. The basic data unit in this kind of systems is video chunk. The origin server divides the media content into small media chunks of a small time interval, each of them with a unique sequence number that serves as a sequence identifier. Later, each chunk is disseminated to all peers through the mesh.

Since chunks may take different paths in order to reach a peer, they may arrive to destination in a non-sequential order. To deal with this matter, received chunks are normally buffered into memory and sequentially rearranged before delivering them to its media player, ensuring continuous playback.

There are two different ways to exchange data in mesh-based systems: push and pull. In a mesh-push system, a peer constantly pushes a received chunk to the rest of neighbors who need it. However, due to system topology, redundant pushes could be responsible for wasting peer uploading bandwidth.

As a result, a mesh-pull system is used to avoid the situation described earlier. This technique allows peers to periodically exchange chunk availability according to buffer maps. A buffer map holds the sequence number of chunks currently available in a peer´s buffer; this way a peer can decide from which peers to download which chunks, avoiding redundant chunk transmissions that were present using push.

A disadvantage of the pull technique is that both frequent buffer map exchanges and pull requests produce more signaling overhead and introduce additional delays while retrieving a chunk.

1.3 P2P Video-on-Demand

Video-on-demand (VoD) systems provide multimedia services offering more flexibility and convenience to users by allowing them to watch any kind of video at any point in time. Such systems are capable of delivering the requested information and responsible for providing continuous multimedia visualization.

Contrary to live streaming, in VoD systems the user has complete control over the media by making use of VCR operations such as pause, forward and

backward functionalities (also known as jump operations); the same way as if the functionalities were used in a VHS or a DVD player.

VoD systems need to accommodate a large number of users watching the same video asynchronously, watching different parts of the same video at any given time. This is a very challenging design situation for tree-based P2P systems because users using this kind of overlay are synchronized and receive the content directly from the source server, and exactly in the same order it left the root node.

Mesh-based P2P systems were successfully introduced to distribute large files and later applied to live streaming. In this kind of systems a large file is usually broken into many small blocks. Both the system throughput and the rate, at which the content can be distributed to users, greatly depend on the diversity of the blocks contained at different peers. The challenge of providing VoD services using mesh-based P2P networks lies in the fact that the blocks have to be received at the peer-side in a sequential order, and time constraints have to be considered at all times to guarantee continuous visualization. Therefore, supporting VoD services using mesh-based P2P is also a great endeavor.

As shown earlier, in Section 1.2, tree-based and mesh-based P2P systems for live streaming have their own advantages and disadvantages. In this section, the different ways of how to adapt these two approaches to providing VoD services are presented.

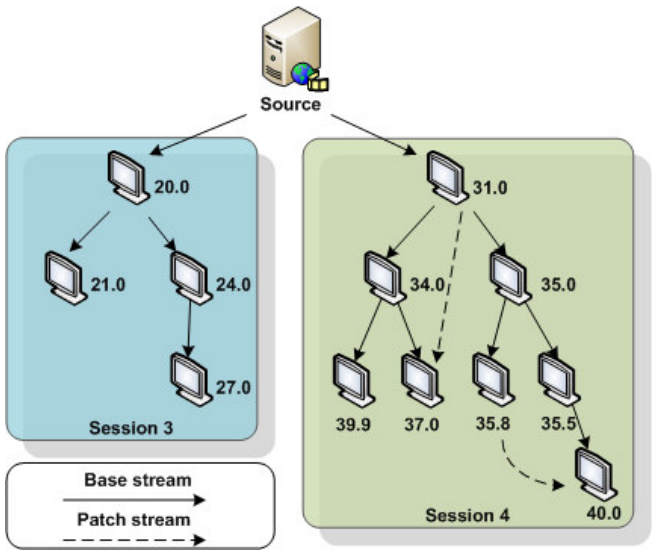


Figure 1.3 – Application Layer Multicast - P2Cast policy

1.3.1 Tree-based P2P VoD

Patching [13] is one of the first IP multicast policies introduced for supporting VoD services. Inspired on this scheme, P2Cast [12] was designed for distributing video content among asynchronous users, where every user behaves as a server while retrieving media content.

As shown in Figure 1.3, users arriving within a threshold form part of a session. Along with the source server, users belonging to the same session form an ALM tree, known as the base tree. The entire video is then streamed from the source server using this base tree; just the same way as in tree-based P2P live streaming. Users joining the session join the base tree and retrieve the base stream from it. In contrast, new clients

who missed the initial part of the video must obtain a patch directly from the server or other users who have already cached the required content.

Users behave just like peers in a P2P network, and provide two main functions: *(i) base stream forwarding*, where users participating in the tree-based overlay should be capable of forwarding the received base stream to others; and *(ii) patch stream serving,* where users cache the initial part of the video and serve the patch to latecomers.

1.3.2 Mesh-based P2P-VoD

In mesh-based P2P file sharing networks, a file is typically divided into a set of small size data blocks. The server, or better known as seeder, is in charge of distributing the blocks to different peers. Later, peers gather information of other users sharing the same content interests and form neighborhoods that allows them to exchange the blocks they are missing or willing to share. To maximize users upload bandwidth and consequently achieving the highest downloading throughput possible, block diversity needs to be taken into account at each one of the peers.

Block diversity improves the systems throughput, but could become a problem during playback time since video blocks have to provide continuity and be played in sequential order. Users need to receive blocks sequentially and not in a random order to watch the movie while downloading [8]. Additionally, the nature of VoD systems demands the availability of different media blocks at any given time, especially if users decide to perform VCR operations during playback and expect high level of service with low startup delays in return.

1.4 Design issues in P2P-VoD systems

P2P-VoD systems also deliver content by streaming, but unlike live streaming, peers are able to watch different parts of the video at the same time while collaborating with each other and offloading the server. To make this collaboration possible, users need to contribute with a small amount of storage instead of just the playback buffer in memory.

This section presents a general architecture and taxonomy for P2P-VoD systems, considering issues like service scheduling, replication strategies, and so on.

1.4.1 Major System Components

A P2P-VoD system has the following major components: a set of servers which act as source of content; a set of trackers in charge of helping peers connect to other peers to share the same content; a bootstrap server to help peers to find a suitable tracker (e.g. based on the geographical location), and other bootstrapping functions; other servers such as log servers used for data measurement purposes, and transit servers for helping peers located behind NAT boxes; finally, a set of peers running software downloaded from the P2P-VoD operator.

1.4.2 Segment sizes

Segmentation of content is fundamental in the design of P2P-VoD systems. There are some considerations to have in mind for making this decision. From the *scheduling point of view*, content should be divided into as many pieces as possible for flexibility reasons at the time of finding which piece to upload from which neighbor. From the *overhead point of view*, the larger the segment the better since it helps to minimize overheads (e.g. content headers, bitmaps, etc.). Another consideration is related to the real-time nature of streaming. The video player or *set-top-box* at the user-side expects a certain minimum size for a piece

of content to be viewable and delivered according to some deadline, namely a *chunk*. If units are too large, the chances of not fulfilling this deadline increase.

1.4.3 Replication Strategy

Assuming each peer contributes with some amount of hard disc storage, a distributed P2P storage system is formed by the entire viewer population, each of them containing chunks. Once all pieces in a chunk are available locally, the chunk is advertised to other peers. The aim of replication strategy is to make chunks available to every user in the shortest time possible in order to meet with viewing demands. Design issues regarding replication strategies contemplate: *(i)* allowing multiple movies to be cached if there is room on the hard disc. This is referred as *multiple movie cache (MVC);* and lets a peer watching a movie upload a different movie at the same time. *(ii)* to pre-fetch or not to pre-fetch; while pre-fetching could improve performance, it could also waste uplink bandwidth resources of the peer. *(iii)* selecting which chunk or movie to remove when the disc cache is full; preferred choices for many caching algorithms are least recently used (LRU) or least frequent used (LFU).

1.4.4 Content Discovery

Together with a good replication strategy, peers must also be able to learn who is holding the content they need without introducing too much overhead into the system. P2P systems depend on the following methods for content discovery: *(i)* a tracker; to keep track of which peers are replicating what part of the movie; *(ii)* DHT; used to assign movies to trackers for load balancing purposes; *(iii)* gossiping method; used for chunk discovery even if the tracker is not available, a peer asks its neighbors for their *chunk bitmaps* and with this information, it selects which neighbor to download from.

1.4.5 Piece selection

In order to download chunks from other collaborators, a peer uses a *pull* method and takes into account three important considerations for selecting which piece to download first: *(i) sequential*, selects the piece that is closest and necessary for playback; *(ii) rarest first*, selects the rarest piece in the system which in turn helps speeding up the proliferation of pieces; and *(iii) anchor-based,* fixed points defined to support VCR features (e.g. forward jumps), commonly found in VoD.

Some approaches implant a hybrid strategy for piece selection, combining sequential and rarest first. Such is the case of Kangaroo, for example.

1.5.6 Transmission Strategy

Once a particular chunk has been selected for download, what happens if more than one neighbor has a copy of it? To answer this question, P2P-VoD systems rely on transmission strategy algorithms. This kind of algorithms are designed based on two objectives: *(i)* maximize download rate; and *(ii)* minimize the overhead caused by duplicate requests and transmissions.

1.5 Current Approaches to P2P-VoD systems

As mentioned previously, the P2P paradigm has been successfully used for providing content distribution as well as live streaming [1, 2, 7, 19x]. Recent interest towards P2P-VoD systems had led to research groups such as Microsoft to elaborate detailed analysis of a current *client-server* VoD system.

In [14], the authors collect traces for a period of nine months and conclude that there are many potential benefits of peer-assisted video-on-demand, such as significant cost savings in server loading. Moreover, issues related to data

prefetching and ISP-friendly considerations should also be taken into account for a successful deployment of a Large- VoD system.

Sharing the same ideas, P2P-VoD developers have been deploying their systems in the last year [3, 5, 6] obtaining a growing number of viewer population in a short period of time. Below, we focus on approaches found within the research community and briefly describe them.

- **BitTorrent Streaming (BiToS)**

Being one of the most successful P2P mechanisms for content distribution in the last years, BitTorrent [1] has the particularity of distributing time insensitive content in a fast and efficient way by using swarming techniques, and by applying incentives to peers to contribute with the community preventing the appearance of *free-riders*. The media distributed in BitTorrent is referred as a torrent file, which is split in pieces of 256KB each, usually known as chunks. While a peer is downloading chunks, it is also uploading other acquired pieces to its neighbors.

Authors in BiToS [20] concentrated their efforts in making the original BitTorrent protocol, a support streaming protocol. During their work, they identified that a piece selection mechanism is the only thing that needs modification in order to achieve that goal. BiToS considers the streaming order of pieces as the more important factor, thus preferring pieces that are closer to the current playback point. BiToS shows to be a feasible approach for enabling BitTorrent to support time sensitive content.

This study does not contemplate VCR operations, but it is considered to be the first attempt to design a mesh-based P2P video-on-demand system.

- **BulletMedia**

BulletMedia [21] uses proactive caching to provide features such as back or forward operations that other deployed peer-assisted VoD systems are able to offer only with the help of very well provisioned source servers.

In this approach, blocks are also decomposed into a set of equal size pieces and grouped into chunks (e.g. 100 blocks in a chunk), which are unselfishly replicated by peers with the solely purpose of increasing the number of replicas in the system. This way, content replication helps ensure block availability in-overlay, thus reducing source dependency.

By using proactive caching, BulletMedia combines traditional overlay mesh approach, used to fetch blocks at a high rate under normal playback; with a structured overlay, responsible to enable efficient block discovery and control block replication.

A DHT is used to store meta-data about the content location within the peers of the mesh overlay. Whenever a peer performs a jump operation, it needs to rapidly discover a sender peer and fetch the required blocks for continuous visualization. If unsuccessful, the peer fetches directly from the source server.

Immediate playback of the media is not taken as a priority; instead this study actively attempts to replicate all blocks from the media file in the overlay.

- **GridCast**

The authors in [10] found that the popularity of channels and a reasonable number of concurrent users can derive in satisfactory user experience. Their experiment was done through the study of logs during a two-month period, using a deployed experimental P2P VoD system over the China Education Network (CERNET).

GridCast uses a set of source servers (or *server farms*) to distribute media files to participating peers, who asynchronously play the files while data is exchanged among each other. To solve the problem of peer seeking operations (i.e. forward jumps), GridCast peer maintains a routing table consisting of some peers placed in a set of concentric rings with power law radii [22], distanced using relative playback positions, and uses gossips to keep the routing table up-to-date. There is a tracker or stationary peer whose playback position does not change and remains fixed at time zero; in charge of bootstrapping new arriving peers.

The peer caches played content onto its disk, data that will be served to other peers or used in case of backward jump operations. Media is divided into chunks of one second play time. Chunks are exchanged between peers from the innermost ring and outwards, or from the origin server if necessary. If possible, the peer also tries to fetch anchors which are segments consisting of ten continuous seconds each, and distributed through the video file at fixed intervals. When a seek operation occurs, the playback position is automatically adjusted to the closest anchor if the anchor has been already downloaded.

Good network bandwidth at peers and sufficient server provisioning are considered critical. The slowest peer determines the startup latency.

1.6 Objectives of the dissertation

Architectural design and deployment of P2P-VoD systems which support VCR functionalities is attracting the interest of an increasing number of research groups within the scientific community; especially due to the intrinsic characteristics of such systems and the benefits that peers could provide at reducing the server load.

In spite of all the efforts, research groups have not been able to deploy a large P2P-VoD system that provides VCR operations optimally. Some existing

approaches rely on highly over-provisioned servers and peers to deal with such jumping operations; others focus their efforts on structured overlays to quickly find the segments needed by a peer. Additionally, these state-of-the-art approaches do not take into account system performance using real network topology; they see the network as a *cloud*.

The objective of this dissertation work is to analyze a P2P-VoD system performance according with the user behavior using real traces as well as synthetic user patterns.

Imagenio traces are studied using a P2P-VoD system developed at Telefónica R&D, named Kangaroo.

In order to perform these analyses, 3 main points are taken into account:

1) Evaluate the impact that VCR functionalities can have on a real P2P-VoD system for various arrival and viewing patterns such as flash-crowd, batched-join and random jumps.
2) Determine how these viewing patterns affect the performance of the system.
3) Use real video traces captured from a live VoD system to characterize the performance of the P2P system under realistic workloads and events.

The fact that there is a growing interest for video-on-demand based on the P2P paradigm within research groups is a motivating factor for the development of the current work.

1.7 Dissertation Organization

The rest of this dissertation is composed of four chapters and organized as follows:

Chapter 2 presents Kangaroo, a P2P-VoD system developed at Telefónica R&D. We describe the way Kangaroo achieves to build good collaboration neighborhoods by using a mesh-based overlay structure and its approach to provide high quality of service at all times.

In Chapter 3 we analyze user behavior in Imagenio. A brief introduction of Imagenio's network is provided and the estimation of an optimal P2P model (i.e. *BestP2P*) is studied.

In Chapter 4 we study perform some experiments in order to compare the optimal model presented in Chapter 3 against the implementation of Kangaroo using real Imagenio traces. Different kinds of workloads and user patterns are used to evaluate the P2P-VoD system performance.

To conclude, Chapter 5 summarizes the results and contributions of this work and states future lines of research.

Chapter 2

Kangaroo

Abstract

This chapter introduces Kangaroo, a P2P-VoD system capable of providing VCR functionalities. A description of Kangaroo application is given along with its most important characteristics.

KANGAROO

2.1 Introduction

Kangaroo is a system focused on providing both P2P video-on-demand services as well as live streaming content, allowing users to have high quality experience along with interactive functionalities, such as forward or backward jump operations. Throughout this work, Kangaroo is studied and used as a P2P-VoD system only.

The design of Kangaroo comprises two main modules: *tracker* and *peer* modules. The tracker is responsible for providing a centralized mechanism; a central point where peers gather information regarding others peers with the solely purpose of establishing future communication relations.

On the other hand, peers manage all the logic necessary to establish collaborations that allows them to exchange the required data in a timely manner. Furthermore, within these peers there are peers that stand out for holding all the segments of the video and are known as *source peers* or *seeds*. Figure 2.1 show a typical collaboration neighborhood in the Kangaroo system.

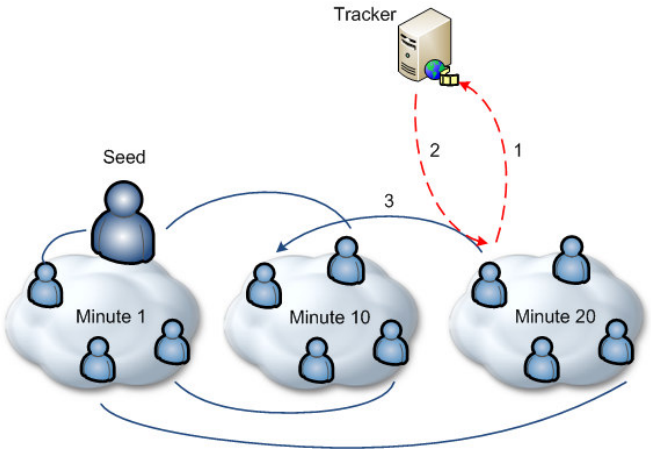


Figure 2.1 – Collaboration neighborhoods in Kangaroo

2.1.1 Tracker Modules

In order to establish collaboration neighborhoods, every peer contacts the tracker whenever it needs to visualize a video or needs to perform a jump operation during playback. The aim of the tracker is to establish relationships between peers interested in the same position of the media.

For example, the tracker retrieves the information provided by peer *A* and generates a list of peers that considers adequate to become part of its neighborhood. Based on the position of the video requested by peer *A* as well as the video position other peers are visualizing at that moment, the tracker selects the neighbors.

The most important tracker modules are as follows:

- CCollaboratorManager: this is the main class of the tracker, contains all the information related to the logic of the tracker. Includes the contact information, IP address and port number of peers for establishing communication. Information related to the media that every peer is willing to share is also given.
- CTrackerCollDescrip: this class represents a collaborator or peer.
- CTrackerMediaDescrip: contains the description of the media published by the peers.
- CTrackerListener: class that contains all the different services coming from the outside.
- CUpkeepScheduler: contains the logic in charge of implementing the timeout process for both published media as well as peers.
- CNewContribution: contains the code for decoding the information in an announcement of new peer contribution.

- CCollaboratorFinder: contains the code for decoding the parameters of collaborators search request.
- CTrackerCollSelector: classes that implement the different collaborator selection algorithms.
- Other support classes.

2.1.2 Peer Core Modules

The core of a peer can be subdivided in four primary blocks: (i) creation of relationships between peers, (ii) neighboring peer requests planning, (iii) request admission policy, and (iv) external interface.

The aim of the first block is to create a proper environment for data exchange and relationship formation amongst peers. Once the environment is set, the second block is responsible for deciding the order of segment requests. Next, the requests are accepted or rejected by the peers according with the admission policy. The design of these three blocks has a great effect on the system's performance in terms of transmitted data between peers. The interface block allows the development of applications that take advantage of Kangaroo's capacity for delivering multimedia content.

These four primary blocks depend on a content scheduler that contains all the metadata related to the media available at a peer. The classes managing the media are:

- CPublicationManager: the main class of the peer which holds the media list. For each media, it contains information regarding the segments, the tracker, the bit rate and the interest a peer has over every media available.

- CPublisherScheduler: the publication logic used for periodically contacting the tracker and announce the media available for collaboration.
- CSwarmManager: for every media that a peer is interested in retrieving, an instance of this class is created. The class is in charge of creating all the necessary handlers for the collaboration process.
- CPeerMediaDescrip: this class represents a media.
- CPeerSegDescrip: contains the state of the media segments and the techniques for handling them.

2.1.2.1 Modules for creation of relationships between peers

Kangaroo organizes the peers on a mesh-based overlay network where each peer establishes neighborhood relationships with 10 or 15 other peers. Additionally, the information of 10 spare peers is saved for the creation of possible future collaboration relationships. All the peers on the mesh are treated equally, having no discriminating features; apart from content availability and transmission capacity. The homogeneity of the peers is critical for the successful design of a distributed system such as Kangaroo.

The modules that implement the neighborhood management are:

- CNeighbourhoodManager: this is the class that centralizes all the metadata related with the neighbors. It also provides techniques to calculate the segment popularity of videos in the neighborhood.
- CNeighbourProcessor: class that manages all the communications with the neighbors.
- CNeighbourSendingProcess: class that sends the messages to the neighbors.

- CNatTraversalReq, CNatTraversalProcess, CNatTraversalReqMsg: classes that perform the Nat traversal requests from a neighbor.
- CPeerCollDescrip: class that describes a neighbor.
- CTrackerDescription: description of the tracker.

2.1.2.2 Modules for Request Planning

The request planning policy decides the order in which each peer performs video segment requests from its neighbors, as well as to which one of them to present such request.

Kangaroo has a hybrid planning policy made of sequential and local-rarest segments. The former always searches for immediate necessary segments to achieve continuous playback and achieve the deadline. In contrast, the local-rarest policy comes more as an altruistic policy where the peer gives more preference to the least replicated segments in the neighborhood.

Kangaroo considers segments replicated in less than 25% of the neighbors as "rare" segments and orders them with respect to the current playback point of the peer. Amongst all rare segments, the one located closer to the playback point is preferred and chosen. Together with it, four other sequential segments are added, making a total of five segments the peer tries to receive from the neighbors at one time.

In order to decide which neighbor will collaborate with media requests, the policy establishes two rules: *(i)* give more preference to the neighbors having less interesting information for the playback of the own peer; *(ii)* give more preference to peers with a greater number of free connections. The first rule has the aim of not saturating the source peer; and the second rule is conceived to avoid possible rejections due to neighbor connection occupation.

The modules implementing request planning are:

CDownloadSchedulerPolicy: the central class that performs periodical planning.

CDownloadSchedulerPolicy
CDownloadSchedulerLRPolicy
CDownloadSchedulerRarestFirstPolicy
CDownloadSchedulerGreedyPolicy
CDownloadSchedulerHybridPolicy
CDownloadSchedulerHybridRandomPolicy

} classes that implement different planning policies

CDownloader: class in charge of communicating with a neighbor to make segment petitions.

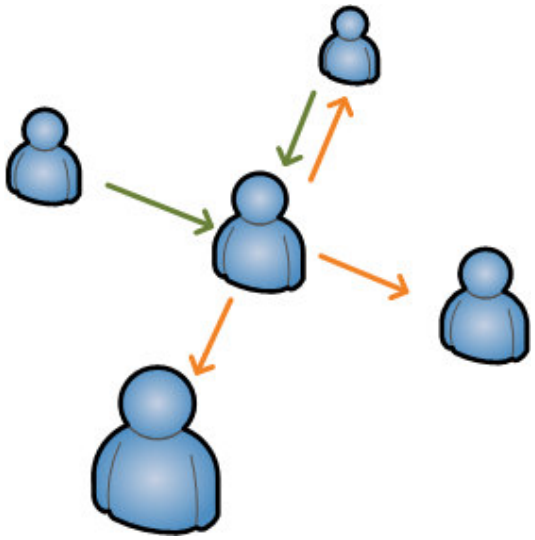


Figure 2.2 – Segment scheduling

2.1.2.3 Request Admission Modules

The aim of the admission policy is to improve the dissemination speed of segments between peers, while achieving a continuous reproduction of the media. This policy is particularly important for the source peer; from the point of view of this peer, the goal is to minimize the number of copies that must be

sent per segment. This way, maximum efficiency is guaranteed by contributing new data to the system, thus minimizing the use of upload bandwidth.

However, there are situations in which the added capacity provided by peers is not enough to guarantee all requests. Two main reasons could lead to such situations: *a)* the asymmetry of sending and receiving peer capacities (i.e. ADSL technology); or *b)* the information requested is located only in a few number of peers. The later condition is quite possible in a system like Kangaroo because users have the possibility of using the VCR operations to perform random jumps during playback.

Kangaroo presents a solution by incorporating a time-marker in every segment's petition, indicating the moment in which the peer needs the data. Moreover, the system defines a tolerance threshold for segment delivery delays. Based on these two parameters, the source peer executes the admission policy as follows:

1) Accepts the petition if the time-mark is out of the tolerance threshold; if not,
2) Accepts the petition if there is a free connection available and if the segment is one of the least replicated segments of the neighborhood; otherwise,
3) Accepts the petition if the bandwidth consumption is below 75%; or else,
4) Rejects, sending the contact information (i.e. IP address and port number) of a neighbor that holds the requested segment.

When a peer is not a source peer, the admission policy is a bit different and much simpler. The number of free connections and uplink bandwidth available are the only parameters taken into account. The classes implementing the admission policy are:

CUploaderScheduler: there is one instance per media that a peer is willing to contribute with. This class is the center of the admission policy.

CUploader: once the request has been admitted, this class is in charge of transmitting the data to the neighbors.

2.1.2.4 Interface modules with the peer core

The core of Kangaroo is a P2P independent system designed to provide multimedia content transmission. The communication with the exterior is done through TCP communications using an interface. The functions offered by the interface are described below.

1) Monitoring of P2P system status. An example would be the number of neighbors that a peer is connected to, or the speed for *receiving/sending* segments *from/to* the neighbors.
2) Interactive operation acknowledgments. As mentioned earlier, Kangaroo is oriented to be a video-on-demand system in which users have the capability of changing media visualization points at any time during playback. After a jump operation is performed, the interface reconfigures the P2P system to provide the requested service with a high level of QoS.
3) Content recomposition. In Kangaroo, the videos are divided in many segments of 64 Kbytes each. The interface provides mechanisms for recomposing these segments into a whole video, making the whole segmentation process completely transparent for the outside.

Classes implementing these functionalities are the following:

CPlaybackInterfaseReq: this is the class that manages all the interactive operations. Also, provides information for monitoring purposes (e.g. system's status).

CPlaybackReq: the class managing the content (or media) recomposition.

2.1.3 Metadata Generation

In order to playback a media, Kangaroo needs to generate a descriptive file associated with the media. To achieve this issue, a module automates this process using a number of input parameters such as: name of the media file, duration time (in seconds) and IP address of the tracker. Optionally, also accepts parameters like: a descriptive name for the video, the output file description name and the segment size. From all the input parameters, the module is capable of generating all the necessary fields for describing the media.

2.2 Communication Protocol

Different types of messages define the way entities communicate within Kangaroo. This section describes the network communication protocol between tracker and peer, and amongst peers.

2.2.1 Tracker - Peer communication protocol

There are three types of messages that are exchanged by the tracker and the peers, namely content announcement, peer search and statistical information.

- **Content announcement.** Every time a peer has content that can be shared with other peers, it sends an announcement to the tracker publishing this information. The format of the message is specified in Figure 2.3.

  NewC identifies the type of message; followed by a set of fields grouped in two blocks: peer and media. The first block contains contact information of the peer, such as: identification number (SSRC), IP address, port number, and software version.

```
NewC
<NewCMessage>
       <Peer>
              <SSRC>%lu</SSRC>
              <Port>%i</Port>
              <IP>%s</IP>
              <Version>%s</Version>
       </Peer>
       <Media>
              <SHA1>%s</SHA1>
              <Path>%s</Path>
              <Name>%s</Name>
              <CompleteNess>%f</CompleteNess>

       <FirstSegmentId>%li</FirstSegmentId>
              <LastSegmentId>%li</LastSegmentId>

<CurrentSegmentId>%li</CurrentSegmentId>
              <SwitchOff>%i</SwitchOff>
              <SuperSeed>%i</SuperSeed>
               <MediaLength>%lli</MediaLength>
```

Figure 2.3 – Content announcement message format

The second block includes a hash message (SHA1) identifying the content; a *Path* used for locating the content; *Name* is the describing name of the media; *CompleteNess* indicates the number of segments a peer has; *FirstSegmentId* and *LastSegmentId* hold the smallest and largest segment number, respectively; *CurrentSegmentId* is the present segment the peer is interested in; *SwitchOff* is set to 1 if the peer wants to stop collaborating with the media; *SuperSeed* takes the value of 1 if it is the server; *MediaLength* describes the size of the media; and *MediaByteRate* indicates the media rate in bytes per second.

- **Peer Search.** Each time a peer needs collaborators to receive media content, it contacts the tracker by sending the message shown in Figure 2.4. The message indicates the information about its own identity (i.e. SSRC, Port number, SHA1) along with the identification number of the segment of interest. Also, the media identification is provided.

```
Find
<FindMessage>
       <SSRC>%lu</SSRC>
       <PORT>%i</PORT>
       <SHA1>%s</SHA1>
       <Segment>%li</Segment>
</FindMessage>\n
```

Figure 2.4 – Peer search message format

The tracker generates a peer list and returns a message of the format presented in Figure 2.5. *FirstSegmentId* and *LastSegmentId* fields, immediately following *FindAnswer,* indicate the first and last segment of the media. The peer list shows the contact information and other statistics. *AnunceTime* specifies user activation time. The meaning of *FirstSegmentId*, *LastSegmentId* and *CurrentSegmentId* is self-explanatory, but this time for the own peer.

```
<FindAnswer>
<FirstSegmentId>%li</FirstSegmentId>
<LastSegmentId>%li</LastSegmentId>
<Peer p="0.9">
       <SSRC>%li</SSRC>
       <Address>%s</Address>
       <Port>%i</Port>
       <Path>%s</Path>
       <CompleteNess>%i</CompleteNess>
       <AnunceTime>%li</AnunceTime>
       <FirstSegmentId>%li</FirstSegmentId>
       <LastSegmentId>%li</LastSegmentId>
       <CurrentSegmentId>%li</CurrentSegmentId>
</Peer>
```

Figure 2.5 – Reply message format to a peer search

- **Statistical information.** A peer can obtain statistical information about a video sending a message containing the media identification number (SHA1), as shown in Figure 2.6.

```
MSTA
<MSTAMessage>
        <SHA1>%s</SHA1>
</MSTAMessage>\n
```

Figure 2.6 –Message format for statistical information request

Once the request is received by the tracker, a message containing the statistical information of the media is sent back. The message has the format shown in Figure 2.7, and includes the first and last segment id, the number of peers watching this video, the number of seeds holding this video and the byte rate of the media.

```
<MSTAAnswer>
        <FirstSegmentId>%li</FirstSegmentId>
        <LastSegmentId>%li</LastSegmentId>
        <NumPeer>%li</NumPeer>
        <NumSeed>%li</NumSeed>
        <ByteRate>%li</ByteRate>
```

Figure 2.7 – Content announcement message format

2.2.2 Communication protocol amongst peers

There are two types of communication amongst peers. In the first one, all messages are associated with the control; the way relationships are established between peers and the exchange of information according to the availability of segments. The second type of communication is traffic associated with data

delivery. Kangaroo uses a standard HTTP/1.1 protocol for data transmission. The reason of using this standard protocol is to make things easier when integrating the Kangaroo system with a consolidated Web proxy environment.

- **Control protocol**

The first step in the collaboration process between peers is a handshake message, *Hand,* as shown in Figure 2.8. After receiving the *hand* message, the peer needs to decide if it is interested in establishing the relationship request. If interested, an acknowledgement message is returned including its identification number together with the media identification number. These two fields are used for validating the peer's identity.

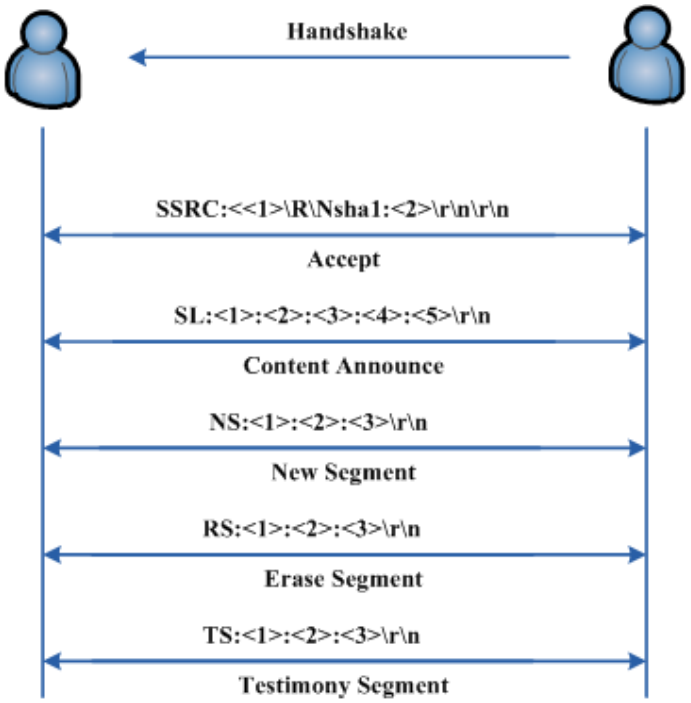


Figure 2.8 – Control protocol between peers

Once there is a relationship established between both peers, messages indicating the availability of data are exchanged. The previous figure shows all the different messages exchanged by the peers. Every message includes different fields that will be described as follows:

Accept: the message is accepted to establish a neighborhood relationship. (includes peer ID number and media ID number.)

Content Announce: announce message of availability of media segments,

1. Percentage of free output connections
2. Segment ID of current interest
3. value = 1, if interested in receiving data
4. The smallest segment number available
5. *Bitmap* of segment availability. A value of 1 means that the segment is available.

New segment: fields 1 and 2 are identical to the content announce message; field 3 is for new segment ID.

Erase segment: message to eliminate or erase a segment that has been announced as available. All the fields are identical to the new segment message.

Testimony message: the message of testimony.

- **Data transmission protocol**

In Kangaroo, every segment is an independent file and in order to transmit every one of them, an HTTP protocol GET operation is initialized by the destination peer. Additionally to the common parameter used by a GET operation, a

TimeStamp parameter is attached to indicate the moment the segment will be needed. The TimeStamp value is used by the admission policy.

However, the source peer could reply to such a request either positively (accepted) or a negatively (rejected). If accepted, it means there are no differences regarding the HTTP standard. Otherwise, if the petition is rejected Kangaroo includes some suggestion information to improve content location like: adding the *SuggestPeerAddr, SuggestPeerPort, SuggestPeerSSRC, SuggestPeerPath*. These fields are necessary for establishing future neighborhood relations. This data exchange process is shown in Figure 2.9.

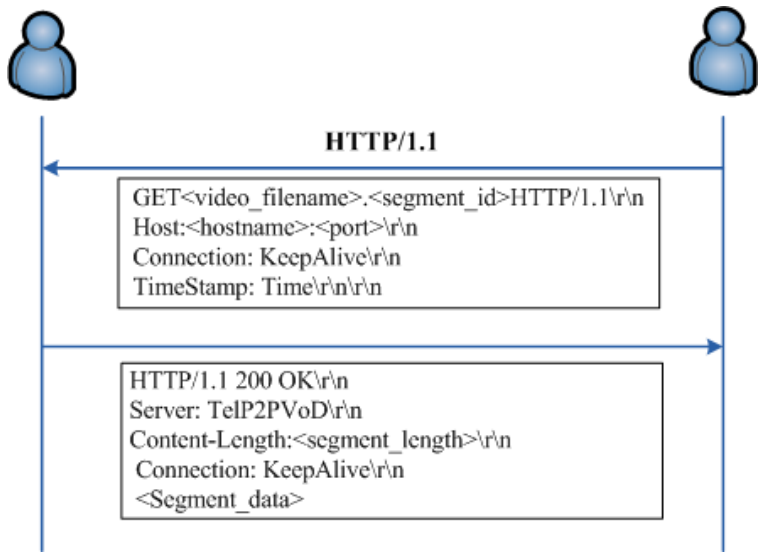


Figure 2.9 – Data exchange protocol

2.3 Block Diagram

All Kangaroo classes are grouped in three different namespaces as shown in Figure 2.10.

The *TelP2PVoDTCP* space gathers all the functions related to TCP communication. All the functions are codified to be compatible with the socket API of both Linux and Microsoft Windows.

The *TelP2PVoDBase* contains functions and base classes shared between all other spaces. Additionally, this space offers two important mechanisms: thread pool and service definitions. *Thread-pool* is implemented in three classes and provides an efficient mechanism of creation and termination of threads, in an environment where the number of threads is unpredictable beforehand. The service definition is done through three classes, including the base class of parameter decoding of a service request.

The spaces on top, *TelP2PVoDPeer* and *TelP2PVoDTracker* include the peer and tracker logic, respectively.

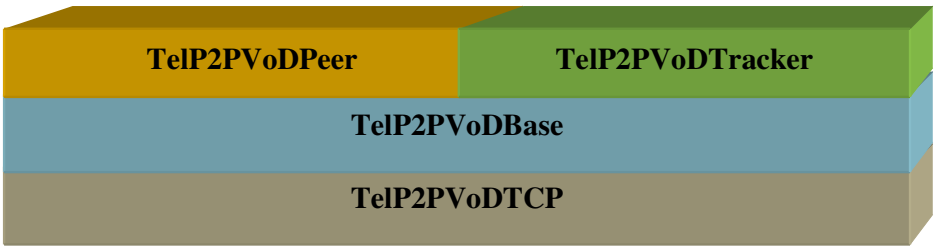


Figure 2.10 – Block diagram of Kangaroo

Chapter 3

User Behavior Analysis in Imagenio

Abstract

This chapter makes a brief introduction of the Imagenio architecture. It also describes the model and the Imagenio traces used in trying to estimate an optimal P2P model that will serve us in our experimental phase.

3.1 Introduction

Imagenio is a global solution for audiovisual services over ADSL in a TV environment; providing a complete range of services such as video-on-demand, digital TV and audio, broadband access to Internet, from both PC and TV and interactive applications that can be used directly from the television.

The Imagenio design uses standard protocols in order to guarantee interoperability, evolution, less dependencies and better development time. Therefore, standards such as MPEG-2, IGMP, HTML and RTSP have been adopted. This approach allows Imagenio to have great flexibility for supporting other networks and access technologies different from ADSL. Being entirely based on the IP protocol, Imagenio architecture is highly compatible with access types like FTTH (*Fiber to the Home)* or VDSL *(Very high bit-rate Digital Subscriber Line)*.

Video-on-demand services in Imagenio allow clients to have TV-quality like media content experience and absolute control over it at any time, allowing clients to perform VCR operations like: fast forward, rewind and pause. Unlike television channel delivery which is broadcast throughout the complete platform, on-demand content is issued by a video server, having the decoder requesting the video as the only destination point. ADSL networks are well adapted to provide this kind of service since every client owns a dedicated channel.

Two logic channels are involved in this type of service: a unidirectional channel in charge of retrieving the video, and a bidirectional channel responsible for content control where the client transmits control commands.

Imagenio network architecture consists of different areas, as shown in Figure 3.1. A brief description of the different areas is given as follows:

- Local services area (ASL): includes video servers of type CCOR (former nCube).

- Central services center (CSC)
- TV Head end: used for unidirectional distribution of content (unicast to multicast). Consists of a MPEG-2 video channel and one or more stereo audio channels of MPEG-1 Layer II format.
- Management center (CGMM): in charge of management and control of the different elements within the platform.
- Customer premise: includes a Set-Top-Box capable of supporting current and future Imagenio services.

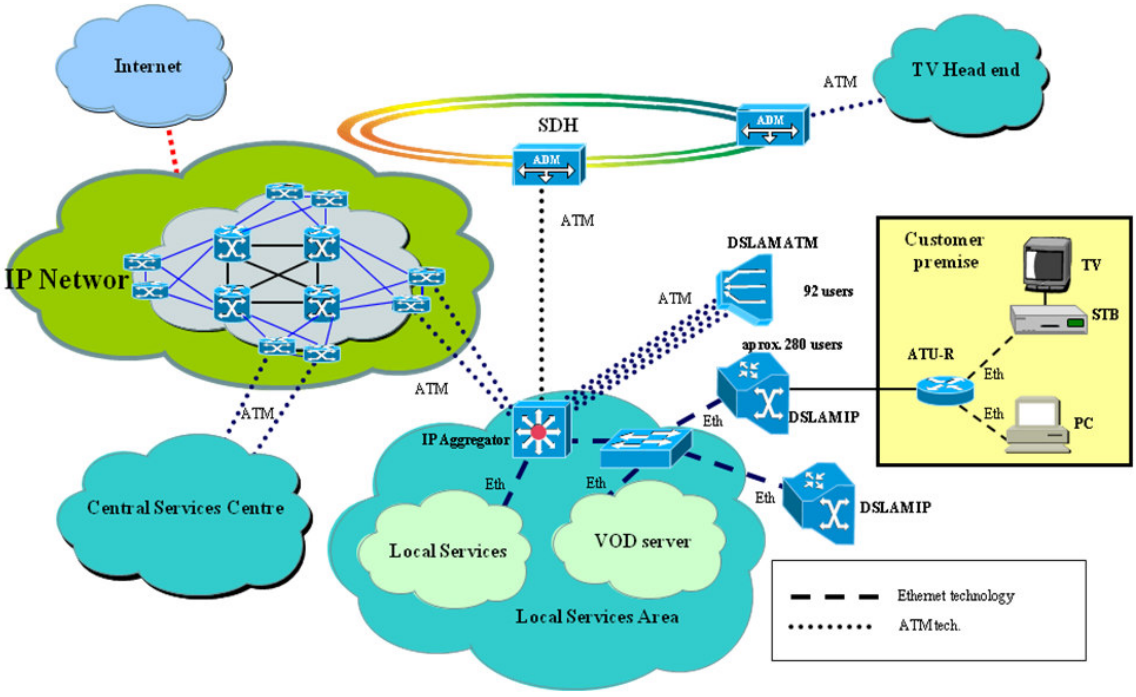


Figure 3.1 – Imagenio Network Architecture

These centers are interconnected by the following transport networks:

- SDH network: includes TV channels distribution and content on demand distribution.
- IP network (Telefónica's main IP network): Data transport (service and management), authorization data.
- Backup of management network.

In order to provide video content, a central server, which is not a video server, gathers the requested files from an external storage device, indexes them and adds a DRM encryption before sending them via multicast to the servers located at the local areas. These local area servers receive the files and insert them into video servers type CCOR (former nCube). In turn, the output of these video servers (in a local area) is aggregated by IP switches which later connect with the level 2 network DSLAMs using a link capacity of 1 Gbps. Every DSLAM is capable of providing service to thousands of users.

The aim of this work is not to study the Imagenio network architecture in detail. Instead, we will focus on the Imagenio VoD traces analyzed and presented in the following section.

3.2 Imagenio VoD Traces

For the analysis of user behavior in the Imagenio network, some important considerations are taken into account.

First, from the traces collected only a number of sessions containing relevant information are selected. Second, the VCR operations we are concerned about are: *jump forward, jump backward, play and pause.* Third, we define different sets of workloads that will help us measure the impact on the capacity of the

source server. These workloads can be of three types: *flash crowds,* where a set of peers make requests all at the same time; *batched joins,* a set of successive peers making requests all at once, and *random jumps*.

- **Environment**

Imagenio traces from 109 days with over 65,500 sessions are available. As Figure 3.2 shows, media content of less than 3 minutes in length is abundant; but not very useful since in such a short period of time is very unlikely a user will perform many seeking operations. As a result, we focus in sessions longer than 10 minutes (around 12,100 sessions) as they contain interesting jumping patterns from which we can extract valuable information.

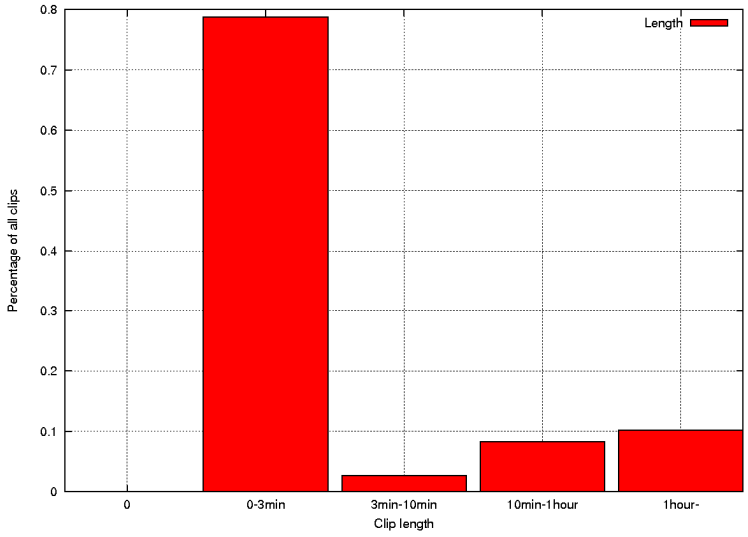


Figure 3.2 – Imagenio VoD traces video length

- **Markov Model**

Inside the trace file, each session corresponds to a user watching a certain movie at a certain time and includes a number of events triggered by the user. Based on

the Markov model, at each step the system could change its state (i.e. playback position) from the current state to a new one, according to a certain probability distribution. According to the Imagenio traces, a user can be in one of four stages: play, pause, forward or backward, with probability of *0.54*, *0.09*, *0.31* and *0.04* respectively. The Markov model, shown in Figure 3.3, is more or less a set of vertices of a graph, including transition steps which involve moving to any of the neighbors of the current vertex with a certain probability.

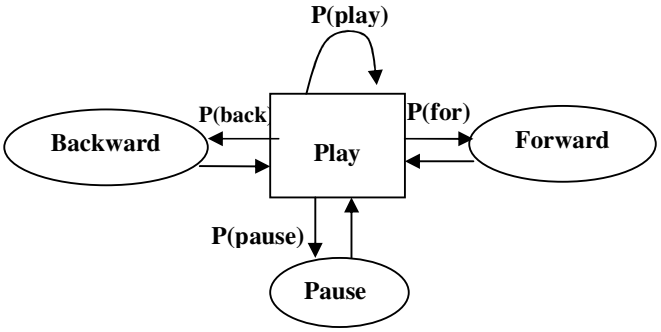


Figure 3.3 – Markov model of expected user behavior

- **The Trace**

Although the Imagenio trace file contains plenty of information regarding user actions during the visualization of a video; it is worth mentioning that all the information included here is, in fact, not relevant for our analysis. The original format of the trace file is shown in Figure 3.4.

20080405\000216.171\mmsvmuvi1\PLAY\10.63.68.\|8413251420390209411\/ /172.26.23.12:554/vod1_trl_ yesterday.mpi\1.0\0.000-\\\\200

Figure 3.4 – Original entry of traces

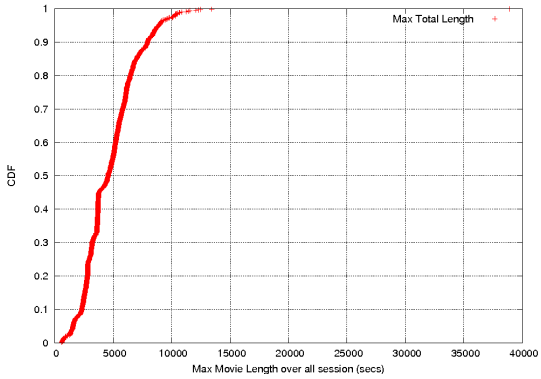
As a result, the original entry file is parsed using a couple of python scripts which help us obtain a final subset of useful values that we can work with. These values of the *new* parsed file are presented in Figure 3.5 and described as follows:

- A date stamp (e.g. *20080405*)
- A time stamp: (e.g. *17:01:49.313*)
- The operation (e.g. PLAY)
- The movie title: *vod1_yesterday.mpi*
- The play speed that could take the values of -8.0 for backward jumps, 1.0 for normal playback, and 8.0 for forward jumps
- Frame position, 0.000- meaning the beginning of the movie
- RTSP response: 200 (Real-Time Streaming Protocol used for VoD requests)

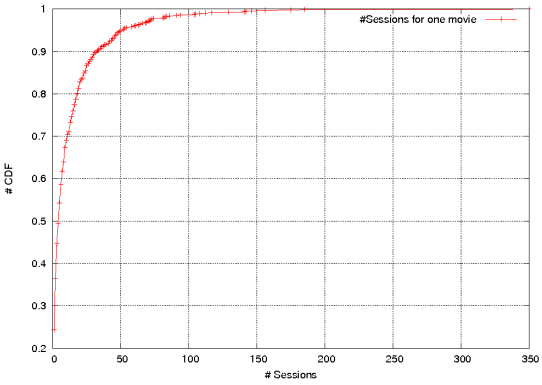
```
20080405 170149.313 PLAY vod1_yesterday.mpi 1.0 0.000 200
20080405 170154.975 PLAY vod1_ yesterday.mpi 8.0 4.800 200
20080405 170215.230 PLAY vod1_ yesterday.mpi 1.0 162.960 200
20080405 170224.541 PLAY vod1_ yesterday.mpi 8.0 171.600 200
20080405 170232.453 PLAY vod1_ yesterday.mpi 1.0 232.560 200
20080405 170318.148 PLAY vod1_ yesterday.mpi 8.0 277.680 200
20080405 170358.477 PLAY vod1_ yesterday.mpi -8.0 594.960 200
20080405 170409.232 PLAY vod1_ yesterday.mpi 1.0 511.920 200
20080405 170453.503 PLAY vod1_ yesterday.mpi 8.0 555.600 200
.......
20080405 171708.531 PLAY vod1_ yesterday.mpi 1.0 4983.120 200
20080405 171713.257 PLAY vod1_ yesterday.mpi 8.0 4986.960 200
20080405 171728.143 PLAY vod1_ yesterday.mpi 1.0 5104.080 200
20080405 171848.885 TEARDOWN vod1_ yesterday.mpi   200
```

Figure 3.5 – Parsed traces containing a subset of interest

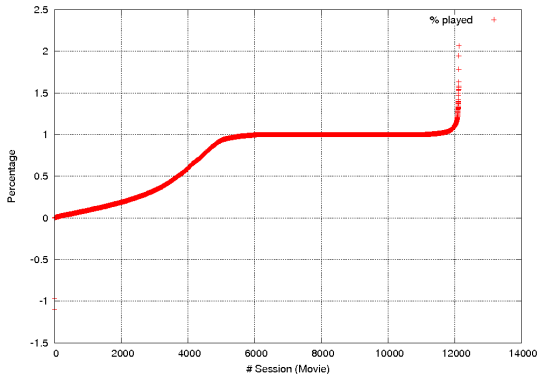
Some of the drawbacks found after parsing the Imagenio traces are described as follows: *(i)* user identity is hidden because the last octet of its IP address is not included in the original file. As a result, sessions cannot be grouped to observe the individual behavior of users. *(ii)* The total movie length is missing but can be inferred (but not very precise). *(iii)* Fast forward and fast backward operations are translated into jumps since we are only considering 4 states based on the Markov model described earlier.



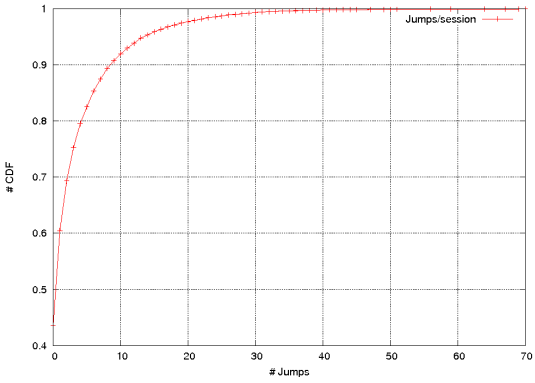
(a) Inferred total length



(b) Clip popularity



(c) Percentage of time played



(d) Number of jumps per session

Figure 3.6 – Data for movies larger than 10 minutes

Figure 3.6 shows results obtained for movies longer than ten minutes considering the drawbacks found after parsing the original file. Figure 3.6 (a) shows that the inferred maximum movie length is around 10,000 seconds. Figure 3.6 (b) shows movie (*clip*) popularity according to the number of sessions a movie has. Figure 3.6 (c) presents the percentage of time a movie was played. Finally, Fig. 3.6 (d) includes the number of jump operations per session.

3.3 Workload Testing

Once we have obtained results using the Imagenio traces, and in order to study how the server capacity is affected by different workloads, an optimal P2P model, called *BestP2P*, is considered.

With this optimal model, we focus on the workloads and assume that peers have infinite upload bandwidth and upload connections used to disseminate segments within the neighborhood, once these segments are provided by the source peer. In every turn, peers request different segments from the source. For this reason, the source must have enough capacity to provide all the necessary segments that have not been requested yet. In case of segments that have been served already, the source relies on the P2P swarm to take care of the distribution.

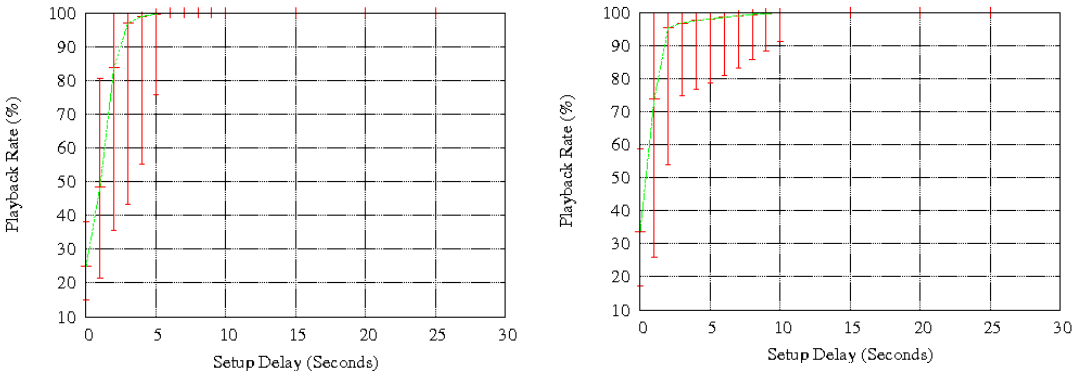
BestP2P is a tracker-based model. Peers contact the tracker and send current playback points. The tracker replies with at most 8 peers, 4 randomly chosen and 4 located as close as possible to that playback position. Minimum state is kept at the tracker. Moreover, peers calculate a healthiness factor h of their neighborhood. If this healthiness factor falls below a certain threshold, then the neighborhood reshuffles in order to obtain better collaborating peers.

As mentioned before, we are interested in understanding the way workload affects the capacity at the server. For this reason, we consider flash crowd and batched join arrival patterns during the tests. For both scenarios we use a movie

clip of 128 seconds of duration. The playback rate is set to 1Mbps and a segment size of 0.5 Mbits. The number of neighbors may vary between 15 and 20, while the number of active neighbors, the ones participating as collaborating peers, varies between 10 and 15. During these experiments only the source peer is connected to everyone else. Figure 3.7 (a) and 3.7 (b) show the scenarios of flash-crowd and batched join, respectively.

Scenario 1 – Flash-Crowd (130 peers)

For the first scenario, 130 peers arrive in a flash crowd all at the same time. Their upload and download rate is the same and equal to 1.2Mbps or "α = 1.2". The source peer has α = 2. As soon as the peers finish their playback they exit the system. Figure 3.7 (a) shows that after waiting for 5seconds of setup delay 100% of the peers arriving in a flash-crowd are able to obtain the content.



(a) (b)

Figure 3.7 – Setup delay time for user arrival patterns:
(a) flash-crowd and (b) batched-join

Scenario 2 – Batched-join

During the second scenario, 75 peers are distributed in groups of 7 each. At time *t=0*, one peer joins the system. At times *t=30, 60, 90, 120, 140, 160* groups of peers join the swarm. There is also a homogeneous crowd of $\alpha = 1.5$, while the source peer has a value of $\alpha = 2$. In this case, peers stay for 10 more seconds after they finish playing. For this case, around 10 seconds are required in order to have all the peers visualizing the media, as shown in Figure 3.7 (b).

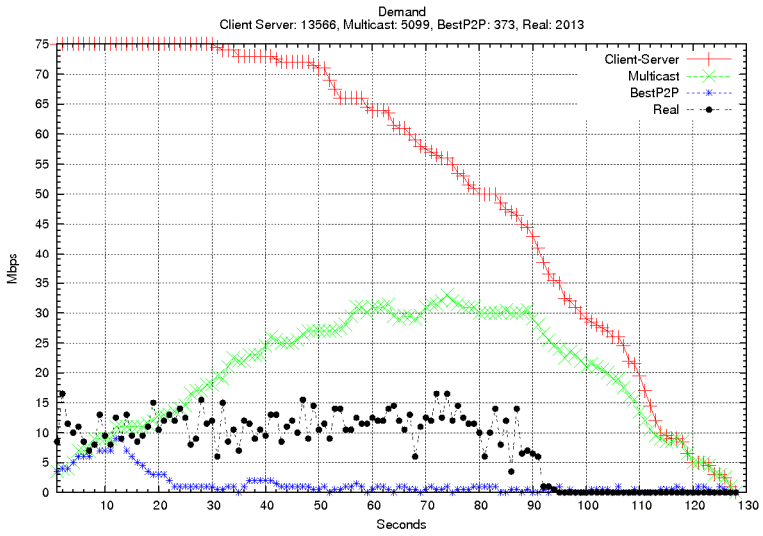


Figure 3.8 – Load expected at the source

Scenario 3 – Random jumps (75 peers)

The third scenario includes random jumps generated with values of 0.99 for play operations and 0.01 for forward jumps. Jump distances follow a normal distribution with mean of 25 seconds and variance of 10 seconds. This is a

homogeneous crowd with $\alpha = 1.6$, while the source peer has unlimited capacity. Each peer follows different assigned schedule and can start jumping at any moment. In Figure 3.8, the values of load expected on the server are shown. As it can be seen, the optimal P2P model (i.e. *BestP2P*) demands less server capacity compared to the other approaches.

In the following chapter, we will use the results obtained from BestP2P and will compare them with a real P2P-VoD system.

Chapter 4

Experimentation

Abstract

In this chapter we compare the optimal P2P model introduced in Chapter 3 to the Kangaroo system applying different workloads and studying how these affect user performance and system provisioning.

4.1 Introduction

Due to the different performance problems that jump operations (user behavior) generate in Large P2P-VoD systems, we decided to evaluate a real P2P-VoD system's behavior as the first step in the analysis of this kind of architectures.

In order to evaluate the effect of jump operations via experiments on a real system; Kangaroo, a mesh-based P2P-VoD system, was used. Kangaroo achieves dealing with jumps by making careful design choices in several aspects of the system as described in Chapter 2.

Particularly, Kangaroo implements a scheduling policy that combines a greedy and an altruistic behavior to accomplish continuous playback and also to improve block diversity. A topology manager is implemented for helping peers at similar playback points to mesh with each other and to quickly find peers with the desired data segments during jump operations.

The effect produced by various workloads on the system performance are presented hereafter, showing through a variety of experiments that Kangaroo can perform well in terms of user delay and server capacity values.

4.2 Experiments

For the experimental setup, a cluster of 10 computers interconnected using a switch of 100Mbps, were used. In every machine multiple copies of a peer are executed as well as a total of 300 peers used during the experiments. Also, videos of approximately 1024 seconds with a bit rate of 1Mbps are used.

In order to measure the performance, a couple of metrics are used. The first one, to ensure that all segments of a video achieve their deadline and arrive on time. The delay of a single segment will imply the reduction of this metric. The second one is related to the upload bandwidth required by the source peer.

During the experiments Kangaroo performance has been analyzed according to different request arrival patterns, such as: flash-crowd, batched-join and Poisson. At the same time, two types of jumps have been considered: random jumps and anchor jumps. Anchor jumps, as mentioned in [10], are a set of well-defined points along the playback where the user is allow and forced to jump.

Imagenio traces were also used considering the user behavior of more than 60,000 users. The analysis of these traces with Kangaroo pretend to show the performance of a real system.

Throughout the experiments, we consider this to be a homogeneous system, meaning that all the peers will have the same upload and download bandwidth.

4.2.1 Sequential Playback

It is considered as the simplest viewing pattern because peers play the media sequentially from beginning to end. Here, we combine sequential playback with two challenging arrival patterns. First, flash-crowd with 172 peers joining the swarm all at the same time. Second, a batched-join scenario consisting of 7 groups of 25 peers each joining the system every 30 seconds. The upload rate α at the source varies from 1,25 - 2Mbps.

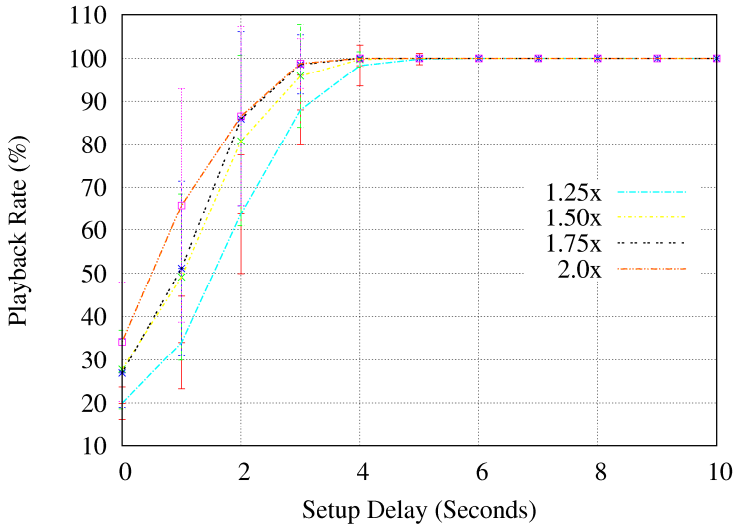


Figure 4.1 – Playback rate obtained for flash-crowd arrival pattern

Figure 4.1 shows the playback rate that can be achieved per user as a function of the setup delay time caused by the flash-crowd arrival pattern. Using Kangaroo, only 6 seconds of waiting are necessary for all the peers to achieve 100% of bitrate. This waiting time is inversely proportional to the network capacity. With α = 2,0, only 4 waiting seconds are necessary to achieve maximum performance.

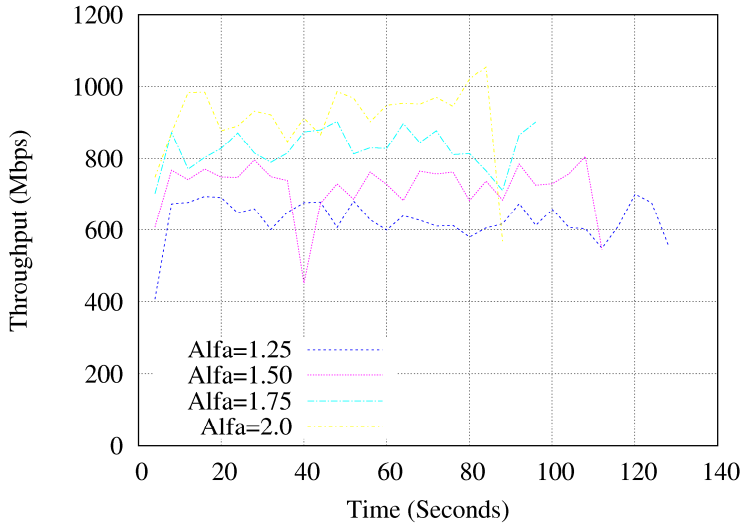


Figure 4.2 – System throughput during flash-crowd arrival pattern

Figure 4.2 shows the system throughput for different values of α (*Alfa*). To higher values of α, the system throughput also increases.

The results obtained for the second scenario, batched-join, are presented in Figure 4.3. The video download times of the peers are very similar; all of them are able to achieve continuous playback after waiting for only 5 seconds. The system throughput results are almost identical with the one shown in Figure 4.2, so we will omit it.

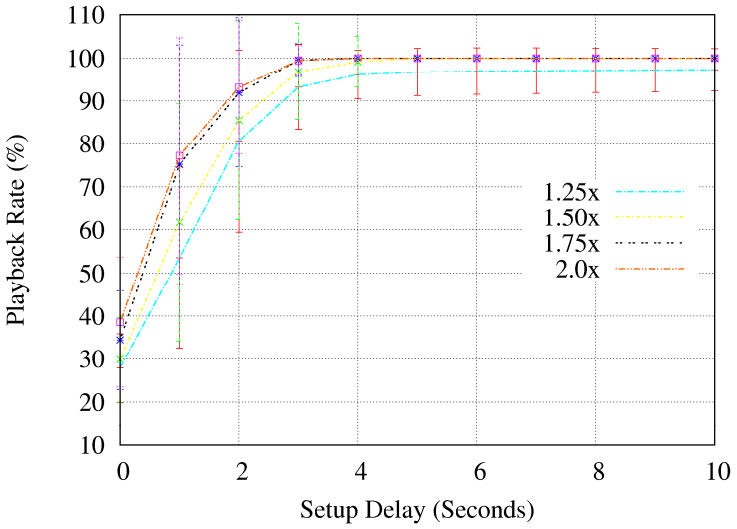


Figure 4.3 – Playback rate obtained for batched-join arrival pattern

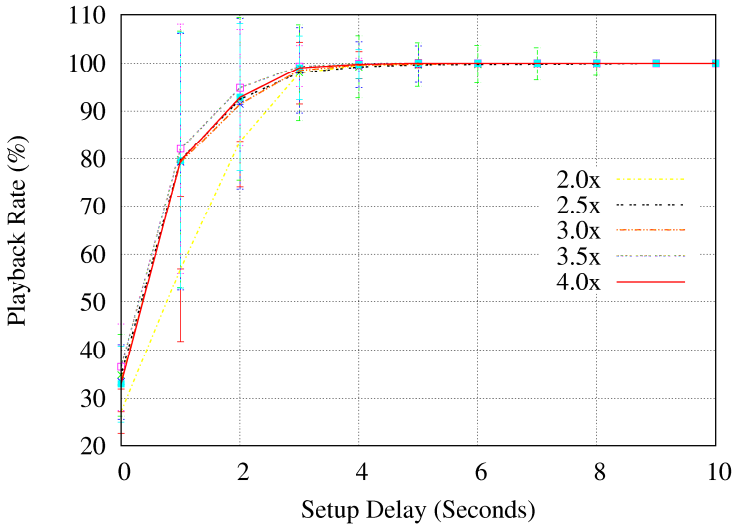


Figure 4.4 – Kangaroo's performance related to source capacity

The performance of Kangaroo as a function of the server network capacity was taken. In this case, the upload and download capacity of peers was set to 1,75x while the source bandwidth capacity varied from 2,0x to 4,0x. The results obtained show that the source has a maximum bandwidth. Once that point is reached, no more network resources are needed because these won't provide any increase in the system's performance, in terms of response time. Figure 4.4 shows that with α = 1,75x, the maximum bandwidth needed by the server is 2,5x.

4.2.2 Random and Anchor Jumps

The second pattern is related to random jumps and the impact they have on system resource requirements. Figure 4.5 shows the percentage of jumps that can achieve 100% of bit rate in function of the waiting time and the bandwidth value. In this experiment 172 peers arrive in a flash-crowd making 3 jump operations each. It can be observed that the anchor jumps achieve better results than random jumps. Within 4 waiting seconds and α=1,75, only 80% of random jumps can achieve the maximum ratio, compared to 92% of anchor jumps.

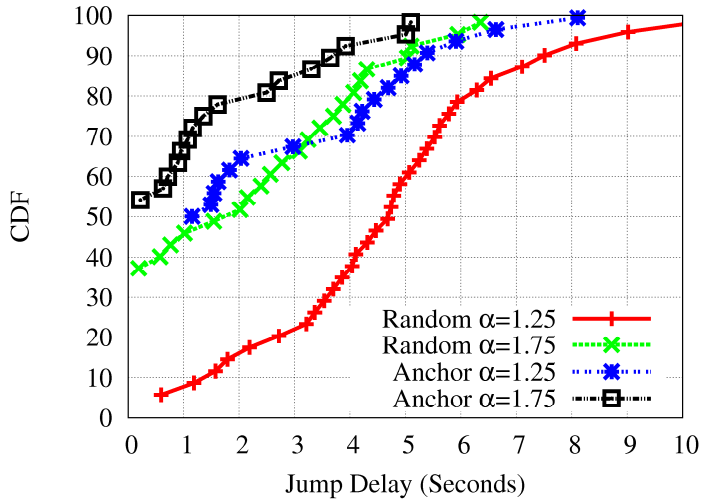


Figure 4.5 – Kangaroo's performance with jump operations

4.2.3 Imagenio Traces

For this evaluation, 172 Imagenio users were selected randomly. Every user performs jump operations forward, backward or pauses the playback. The movie is normalized to 1024 seconds for simplicity. Figure 4.6 (a) and (b) show the percentage of jumps achieving 100% of bit rate in function of the waiting time and the load on the source. It can be seen that compared with the optimal P2P, the load on the source in Kangaroo is very similar if α=1,75. Data shows a better performance of Kangaroo using real user behaviors, but we have to consider that the actual Imagenio system presents a very reduced number of jumps.

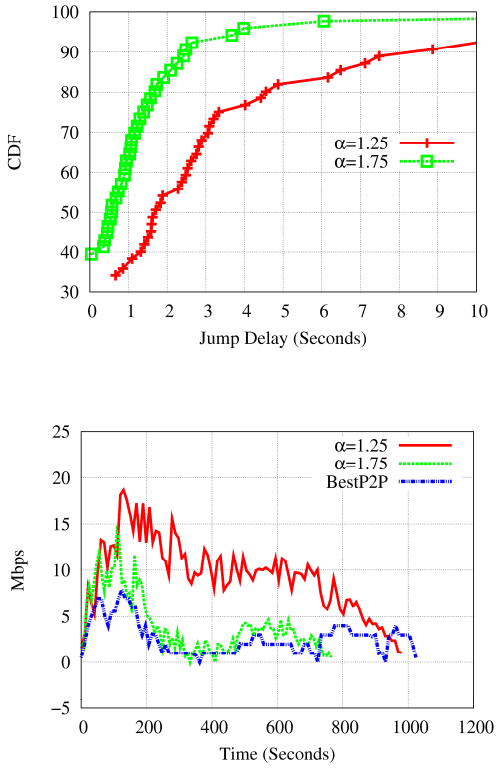


Figure 4.6 (a) y (b) – Kangaroo's performance using Imagenio traces

4.2.4 Tracker Load

How many requests are received by the tracker should be considered since it is critical for the scalability of the system. With the purpose of updating their neighborhood, peers get in contact with the tracker in two cases: *(i)* at every jump operation, and *(ii)* when triggered by neighborhood health evaluation.

The former condition depends on the workload established while the latter condition relies on the value of the threshold t of acceptable health factor. Choosing a value of t involves a tradeoff: higher values of t lead to better neighborhoods but also increase the number of messages sent to the tracker, which in turn can increase the response time of the tracker and eventually the delay experienced by peers at each jump.

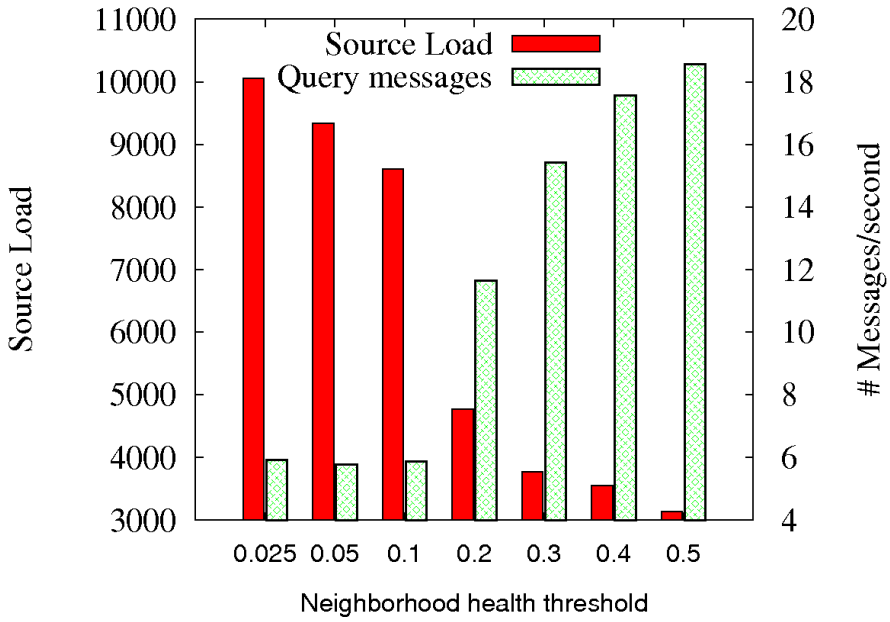


Figure 4.7 – Threshold values for the neighborhood health factor

Figure 4.7 shows the number of triggered update messages and the source load as functions of threshold *t*, for 172 peers arriving in a flash crowd and doing 2 random jumps forward. For a value of $t = 0.2$, the best tradeoff between the tracker response time and topology connectivity is given. A higher value implies more load on the tracker, what could lead to a bottleneck. In other words, *t* should be chosen so as to combine a low number of tracker updates and high performance.

Finally, Figure 4.8 shows a comparison of the demand at the source peer for different workloads and Poisson arrivals ($\lambda = 1$ peer/sec). Naturally, sequential playback requires less capacity at the source for its simplicity. Random jumps at uniformly distributed points of the remaining sequence are the most demanding workload. Random jumps at uniformly distributed points in the entire sequence provide an easier workload since backward jumps can be served by the swarm. Imagenio traces are non-demanding workloads also, requiring server capacity of $\alpha \leq 10$ for 95% of the time.

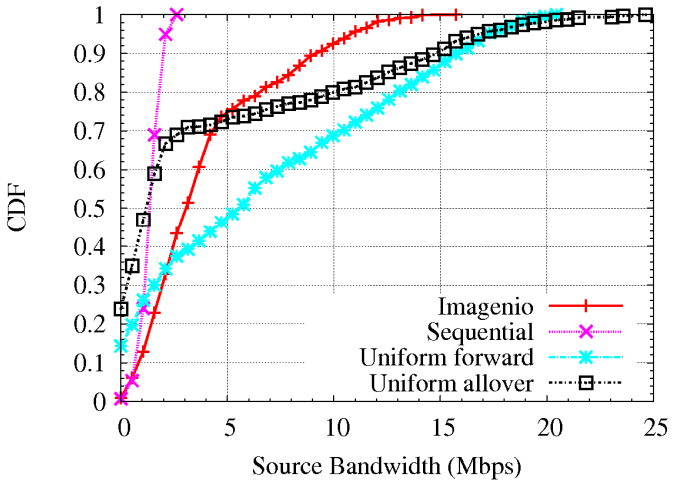


Figure 4.8 – Demand at the source for different workloads and user arrivals

EXPERIMENTATION

Chapter 5

Conclusions

Abstract

This chapter describes the conclusions obtained from this dissertation work as well as the future lines of research.

CONCLUSIONS

5.1 Conclusions

The success of the peer-to-peer paradigm in both file distribution and live streaming application derived in the adoption of this technology for the delivery of video-on-demand content.

However, providing on-demand services using P2P is a very challenging task; and has attracted the attention of different research groups and application developers who are looking for feasible solutions that will consider real-time constraints, VCR functionalities and high quality of service (QoS).

Some current approaches rely on over-provisioning servers or on structured overlays to quickly find segments requested by users to provide VoD services. But regardless of how well the P2P protocol is designed, we believe there are certain user behaviors as well as jumping patterns that will have an effect on the server load. For example, if peers in the system decide to jump to different positions all at the same time, most probably the system will be incapable of dealing with the load and will crash.

During the realization of this work, we evaluated the effect of different workloads and user behavior patterns on a real P2P-VoD system performance. From the results obtained, we can conclude that Kangaroo performs pretty well, achieving low user delay with small server capacity for all considered

workloads: *flash-crowd, batched-join and random jump*; and user patterns determined from the Imagenio traces.

5.2 Future Lines of Research

Having analyzed the feasibility of a real P2P-VoD system, such as Kangaroo, to deal with different workloads and user interactivity operations, the following points are considered as future lines of research:

- Consider the classification of users according to the content being watched, and study if there is a direct relation between any particular user and a specific content.
- Take into account ISP-friendly considerations and study how cross-ISP traffic can be reduced while maintaining P2P performance.
- Study the effect of NAT traversal problems in P2P systems. To solve this some questions need to be answered : What kind of incentives are feasible? Under what kind of conditions are peers willing to provide more resources?

References

[1] BT. BitTorrent Homepage. http://www.bittorrent.com/.

[2] EMULE. Emule Homepage. http://www.emule-project.net/.

[3] GridCast. http://www.gridcast.cn/.

[4] Internet Protocol Multicast, ch 43 of Internetworking Technology Handbook, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.pdf

[5] Joost. http://www.joost.com/.

[6] PFSVOD. http://www.pplive.com/subject/20070808pfsvod/.

[7] PPLive. http://www.pplive.com/.

[8] Annapureddy, S., Guha, S., Gkantsidis, C., Gunawardena, D., & Rodriguez, P. R. (2007). Is high-quality vod feasible using P2P swarming? *Proceedings of the 16th International Conference on World Wide Web,* , 903-912.

[9] Cheng, B., Jin, H., & Liao, X. (2007). Supporting VCR functions in P2P VoD services using ring-assisted overlays. *Communications, 2007.ICC'07.IEEE International Conference on*, , 1698-1703.

[10] Cheng, B., Liu, X., Zhang, Z., & Jin, H. (2007). A measurement study of a peer-to-peer video-on-demand system. *IPTPS, Bellevue, WA, Feb,*

[11] Chu, Y., Rao, S., Seshan, S., & Zhang, H. (2002). A case for end system multicast. *Selected Areas in Communications, IEEE Journal on, 20*(8), 1456-1471.

[12] Guo, Y., Suh, K., Kurose, J., & Towsley, D. (2003). P2Cast: Peer-to-peer patching scheme for VoD service. *Proceedings of the 12th International Conference on World Wide Web*, , 301-309.

[13] Hua, K. A., Cai, Y., & Sheu, S. (1998). Patching: A multicast technique for true video-on-demand services. *Proceedings of the Sixth ACM International Conference on Multimedia,* , 191-200.

REFERENCES

[14] Huang, C., Li, J., & Ross, K. W. (2007). Can internet video-on-demand be profitable? *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, , 133-144.

[15] Jannotti, J., Gifford, D. K., Johnson, K. L., Kaashoek, M. F., & O'Toole Jr, J. W. (2000). Overcast: Reliable multicasting with on overlay network. *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation-Volume 4 Table of Contents,* , 14-14.

[16] Kostić, D., Rodriguez, A., Albrecht, J., & Vahdat, A. (2003). Bullet: High bandwidth data dissemination using an overlay mesh. *ACM SIGOPS Operating Systems Review, 37*(5), 282-297.

[17] Li, J. (2008). On peer-to-peer (P2P) content delivery. *Peer-to-Peer Networking and Applications, Springer Journal on,* 45-63.

[18] Padmanabhan, V. N., Wang, H. J., Chou, P. A., & Sripanidkulchai, K. (2002). Distributing streaming media content using cooperative networking. *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video,* , 177-186.

[19] Tran, D., Hua, K., & Do, T. ZIGZAG: An efficient peer-to-peer scheme for media streaming. *INFOCOM 2003.Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.IEEE, 2*

[20]Vlavianos, A., Iliofotou, M., & Faloutsos, M. (2006). BiToS: Enhancing BitTorrent for supporting streaming applications. *IEEE Global Internet,*

[21]Vratonjić, N., Knežević, N., & Rowstron, A. (2007). Enabling DVD-like features in P2P video-on-demand systems. *Proceedings of the 2007 Workshop on Peer-to-Peer Streaming and IP-TV,* , 329-334.

[22]Wong, B., Slivkins, A., & Sirer, E. G. (2005). Meridian: A lightweight network location service without virtual coordinates. *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,* , 85-96.

[23] Zhang, X., Liu, J., Li, B., & Yum, T. S. P. (2005). CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. *Proceedings of IEEE INFOCOM, 3*, 13-17.