MASTER IN COMPUTER SCIENCE

Video-on-Demand over Internet: a survey of existing systems and solutions

DONY, Yves

Award date:
2008

Facultés Universitaires Notre-Dame de la Paix,
Namur
Faculty of Computer Science
Academic year 2007 - 2008

Video-on-Demand over Internet

A Survey of Existing Systems and Solutions

Yves Dony

Thesis submitted in partial fulfillment of the requirements for the Master degree in Computer Science.

Abstract

Video-on-Demand is a service where movies are delivered to distributed users with low delay and free interactivity. The traditional client/server architecture experiences scalability issues to provide video streaming services, so there have been many proposals of systems, mostly based on a peer-to-peer or on a hybrid server/peer-to-peer solution, to solve this issue. This work presents a survey of the currently existing or proposed systems and solutions, based upon a subset of representative systems, and defines selection criteria allowing to classify these systems. These criteria are based on common questions such as, for example, is it video-on-demand or live streaming, is the architecture based on content delivery network, peer-to-peer or both, is the delivery overlay tree-based or mesh-based, is the system push-based or pull-based, single-stream or multi-streams, does it use data coding, and how do the clients choose their peers. Representative systems are briefly described to give a summarized overview of the proposed solutions, and four ones are analyzed in details. Finally, it is attempted to evaluate the most promising solutions for future experiments.

Résumé

La vidéo à la demande est un service où des films sont fournis à distance aux utilisateurs avec un court délai de démarrage et des possibilités d'interaction. L'architecture traditionnelle client/serveur éprouve des problème à s'adapter à la fourniture de streaming video à grande échelle, de telle sorte que de nombreux systèmes ont été proposés pour résoudre ce problème, souvent sur base d'une architecture peer-to-peer ou d'une solution hybride serveur/peer-to-peer. Ce travail présente un aperçu des systèmes et des solutions actuellement proposés ou existants sur base d'un sous-ensemble de systèmes représentatifs, et définit des critères de sélection permettant de classifier ces systèmes. Ces critères sont basés sur des questions usuelles telles que, par exemple, s'agit-il de vidéo à la demande ou de live streaming, l'architecture est-elle basée sur un content delivery network, du peer-to-peer ou bien les deux, l'overlay est-il tree-based ou mesh-based, le système est-il push-based ou pull-based, single-stream ou multi-streams, emploie-t-il le codage de données, et comment font les clients pour choisir leurs pairs. Les systèmes représentatifs sont brièvement décrits pour donner une vue d'ensemble récapitulative des solutions proposées, et quatre systèmes sont analysés en détail. Finalement, les différentes options sont analysées afin d'essayer d'évaluer les plus prometteuses pour les développements futurs.

Keywords

Video-on-demand service, live streaming video streaming, peer-to-peer networks, overlays, selection criteria.

I would like to thank Mr Laurent Schumacher, my promoter, for his active support throughout the preparation and the writing of this thesis.

Table of Contents

Glossary of used acronyms

CDN	Content Delivery Network
DHT	Distributed Hash-Table
DSL	Dynamic Skip List
FGS	Fine Grain Scalable
GNP	Global Network Positioning
GOP	Group of Pictures
MD-FEC	Multiple Description - Forward Error Correction
MDC	Multiple Description Coding
MPEG	Moving Picture Experts Group
P2P	Peer-to-Peer
PIC	Practical Internet Coordinates
QoS	Quality of Service
RFMD	Redundancy-Free Multiple Description
RMVB	RealMedia Variable Bitrate
RS	Reed-Solomon
SFC	Space Filling Curve
SVC	Scalable Video Coding
WMV	Window Media Video

1. Introduction

With the widespread deployment of broadband residential Internet access, there has been an increasing interest in developing media streaming services and in Video-on-Demand (VoD) in particular during the last ten years. VoD is a service where movies are delivered to distributed users with low startup delay and full interactivity. However, providing such a service over the Internet with the traditional client/server architecture leads to lack of scalability, long startup delays, latency problems, and streaming continuity issues.

As a result, there have been many proposals to provide scalable video streaming services. Though current popular video services mainly rely on content delivery networks (CDNs) and local streaming proxies to increase system scalability, there have been some successful deployments for live streaming, such as CoolStreaming and PPLive, and VoD streaming, such as Joost, using a peer-to-peer overlay architecture. The purpose of this work is to carry out a survey of the currently existing or proposed systems and solutions for providing VoD services over the Internet.

This work is related to a project which researchers of the Faculty of Computer Science are part of, with the objective of proposing an architecture for VoD streaming using a peer-to-peer network to provide a reliable and scalable solution for the simultaneous delivery of video streams to multiple sets of distributed clients.

The first step of this work consisted in collecting and reading scientific papers about video streaming in order to draw up a list of the proposed or existing systems for which the necessary information to carry out an analysis was available. It proved quickly to be more difficult than foreseen, because it was actually hardly possible to set up a comprehensive list of them all, within a limited period of time, given the outstanding number of ideas on the matter and the multitude of proposed systems. So it was decided to limit the survey to a subset of systems being representative of the architectural options.

Then, the selection criteria allowing to classify the selected systems into different categories were defined based on, first the main architectural options, and second the questions considered to be relevant for the above mentioned project. Those questions were: Does the system use a push- or pull-based scheme? Do peers get the video from a single source, or combine from several sources? How to identify the peers to stream from? How do we collect P2P status? How do you know where the content you are interested in is to be found? What content does the peer serve? Where is the content stored? How much memory does the peer dedicate for serving other peers? Are peers homogeneous or not? Does the system use incentives?

Finally, a further reduced selection of systems has been chosen for a detailed analysis, with a balance between VoD systems and live streaming systems, and a representativeness of the different overlay architectures and the currently implemented systems. The remaining selected systems have been briefly described on basis of the selection criteria.

Most of the information used in this work has been found in scientific papers related mainly to video streaming and to peer-to-peer networking. Little information has also been found on the web sites of systems released for public viewing. In this document, the choice has been made to explicitly cite or paraphrase those sources which important parts have been borrowed from. The originality of the work lies therefore in the analysis of the systems, presented from chapter 3 on.

This work is organized as follows. Video-over-IP applications and Video-on-Demand in particular are presented in chapter 2. In chapter 3, a list of selection criteria used to classify the different systems is presented with a detailed description of the various proposed options. In chapter 4, the proposed or existing systems are described and a selection of representative ones are analyzed in detail. Next, an analysis is presented in chapter 5. Finally, this work is concluded in chapter 6.

2. Video-over-IP applications and Video-on-Demand (VoD)

Video-over-IP applications have recently attracted a large number of users over the Internet. On YouTube, the world's most popular online video community, people are watching hundreds of millions of videos a day, and uploading hundreds of thousands of videos daily. In fact, every minute, ten hours of video is uploaded to YouTube [YouTube 2008]. In the same time, Internet live video streaming systems, also known as Internet Protocol Television (IPTV*),* such as PPLive [PPLive 2008], CoolStreaming [CoolStreaming 2008] and Zattoo [Zattoo 2008] have been developed and deployed, and have been watched by thousands of users. With the fast deployment of high speed residential access, video traffic is expected to be the dominating traffic on the Internet in the near future [Liu et al, 2008].

As a result, "video streaming over the Internet has been one of the most prolific research areas for over a decade. Most related to this work are the research efforts for the designing of video distribution systems that can support a large number of users" [Annapureddy et al, 2007]. The particular problem of designing a Video-on-Demand (VoD) service has also received extensive attention in the past [Annapureddy et al, 2006].

"VoD is an interactive multimedia service, which delivers video content to subscribers (or users)" [He et al, 2007]. It "proposes to provide subscribers with the possibility of watching the video of their choice at the time of their choice, as if they were watching a rented video cassette" [Hu 2001] or a DVD. "Due to the frequent VCR controls from users, such as play, pause, fast forward, fast search, reverse search and rewind, existing approaches either present long latencies on the user side, or incur excessive stress on the server side" [He et al, 2007].

The VoD user behavior could be described as follows. "First, VoD users rarely view the movie continuously from the beginning to the end. Second, accesses to different periods of a video are not uniformly distributed. Some parts always attract more accesses than the others. Hot scenes are always appreciated by most audience and lead to a consensus on the popularity of a video. Third, VCR interactivities occur frequently. Many VCR controls are forward-looking, due to the users' skips of the video, ignoring the uninteresting periods" [He et al, 2007].

"VoD allows users to watch any point of video at any time. Compared with live streaming, VoD offers more flexibility and convenience to users and truly realizes the goal of watch whatever you want whenever you want. VoD has been identified as the key feature to attract consumers to IPTV service" [Liu et al, 2008].

"Video-on-demand (VoD) dedicates a single channel to each user and enables the video to be started at any time with VCR-like controls (pause, rewind, fast-forward, etc.). Today, IP network based deployments for services such as VoD are very limited in scope. The largest deployed libraries achieve a mere 0.7 per- cent (5,000 hours) of the global movie and TV-series catalog (DVD by mail services, such as Netflix, offer up to 10 per- cent), and peak utilization reaches only 10–15 percent of broadcast TV. There is a strong belief among telecommunication companies that this market will expand exponentially in the next few years. Service providers are intensely interested in scalable design methodologies for the deployment of large-scale IP content delivery networks that provide content propagation, storage, streaming, and transport" [Thouin and Coates 2007].

"Extensive VoD systems also have the potential to consume enormous bandwidth. The interactivity requirement and lack of network support restrict the widespread application of multicast streaming. Therefore, many concurrent unicast streams must be supported. Even when state-of-the-art compression methods such as MPEG-4 are employed, hundreds of gigabits of streaming capacity are required. The high bandwidth requirements encourage distributed architectures with replication of content and localization of network traffic, but such architectures imply a substantial increase in

storage requirements, which is not a negligible factor, given the large size of video files" [Thouin and Coates 2007].

"The basic solution for streaming video over the Internet is the client-server service model. A client sets up a connection with a video source server and video content is streamed to the client directly from the server. One variation of client-server service model is the Content Delivery Network (CDN) based video streaming. In CDN based solution, the video source server first push video content to a set of content delivery servers placed strategically at the network edges.

Instead of downloading from the video source server, a client is normally directed to a nearby content delivery server to download the video. CDN effectively shortens the users' startup delays, reduces the traffic imposed on the network, and serves more users as a whole.

YouTube employs CDN to stream video to end users" [Liu et al, 2008]. "YouTube videos today are typically less than 10 minutes in length and have a bit rate under 200 kbps" [Liu et al, 2007].

"The major challenge for server based video streaming solutions, though, is its scalability. A video session with good quality requires high bandwidth. With the current video compression technology, the streaming rate for a TV quality video is more than 400 kilo-bits-per-second. The bandwidth provision, at video source servers or in CDNs, must grow proportionally with the client population. This makes the server based video streaming solutions expensive" [Liu et al, 2008].

IP Multicast is probably the most efficient vehicle for video streaming services; but its deployment however remains confined due to many practical and political issues, such as the lack of incentives to install multicast-capable routers and to carry multicast traffic. Researchers thus have resorted to application-level solutions using Peer-to-Peer (P2P) systems, which build an overlay network out of unicast tunnels across cooperative participating users, called overlay nodes, and multicast is then achieved through data relaying among these nodes [Zhang et al, 2005].

"P2P systems have been immensely successful for large scale content distribution. Current peer-to-peer applications generate a large percentage of the traffic over the Internet and a large fraction of that traffic relates to distributing video content. However, with current systems, the users need to download the complete file, and as a result suffer long delays before they can watch the video. Recently systems such as CoolStreaming [Zhang et al, 2005] have been very successful in delivering live media content to a large number of users using mesh peer-to-peer technology. However, it has been an open question whether similar P2P technologies can be used to provide a VoD service. A P2P VoD service is more challenging to design than a P2P live streaming system because the system should allow users arriving at arbitrary times to watch (arbitrary parts of) the video, in addition to providing low start up delays. The fact that different users may be watching different parts of the video at any time can greatly impact the efficiency of a swarming protocol" in which a collection of peers actively share a file, exchanging pieces of it and being simultaneously client and seeder. "The lack of synchronization among users reduces the block sharing opportunities and increases the complexity of the block transmission algorithms" [Annapureddy et al, 2006].

"Current P2P systems have inherent limitations that do not allow them to support a play-as-you-download experience. In these systems, the peers have partial content which they exchange with each other in order to download the complete file. The system achieves high throughput when the peers can exchange content with each other, and this happens only when they have non-overlapping pieces of the file. Hence, the peers download pieces of the file in random order to minimize the overlap. However, in order to support a play-as-you-download experience, the peers require blocks in sequential order from the beginning. However, if all the peers were to download content in sequential order, they would have overlapping pieces of the file, and the utilization (throughput) of the system would be low. The goal then is to design a P2P system which meets the VoD requirement

of play-as-you-download experience, while maintaining a high utilization of the system resources" [Annapureddy et al, 2006].

So it appears that the video streaming systems are not mature yet. Nevertheless, a lot of work has already been done upon such systems. The purpose of this work is to give a synthetic outline of the state of the art on the subject, and to analyze the various techniques and strategies designed or used in the different video streaming systems.

The focus of this work is set on Video-on-Demand (VoD) but, according to the similarities in both the systems constraints, like huge need of bandwidth, and used techniques, like overlay topologies, we will also pay attention to the live streaming systems and to the way the developments designed for those systems could contribute to the VoD systems.

3. Selection criteria

In this chapter, we describe a series of selection criteria used to categorize existing and proposed streaming systems. Those criteria are video streaming type, system architecture, delivery overlay topology, data exchange design, single or multiple sources, peer choice protocol, content repository, peer upload source, homogeneity of peers, locality of peers, incentives, and performances evaluation.

3.1. Streaming type

Video streaming applications can be divided into two main categories: Video-on-Demand and live streaming. Though those categories share several common principles, they both have their own particular issues which we describe in this section.

3.1.1. Video-on Demand (VoD)

Video-on-Demand (VoD) is an interactive multimedia service which should enable users to start watching the video of their choice at the time of their choice, after waiting for a small startup delay, and perform VCR-like control operations such as play, pause, fast forward, fast search, reverse search and rewind, while downloading the video in parallel.

The first point to address in the implementation of such a service is the amount of available media content; the amount of titles should at least compare with that available in an average video store. This should no longer be an issue considering the huge storage capacity commonly available on much of the Internet current servers.

The second point and first critical problem is the bandwidth required for providing such a service. While providing low quality short videos to a limited number of users could easily be implemented using a server cluster and a classical client/server architecture, providing full length movies with a DVD-like quality to several thousands of users at the same time poses a significant challenge in terms of scalability.

The third point is that users should be able to watch the video at an arbitrary time. Although a large number of users may be watching the same video, they are asynchronous to each other and different users are watching different portions of the same video at any given moment. This particularity causes an additional problem in the use of P2P networks in order to meet the scalability issue mentioned in the previous point.

The fourth point is the tough issue of supporting interactive VoD. While pause and rewind can be supported by introducing more buffer space, fast search and fast forwarding are the most difficult features to implement in overlay protocols. Fast forward requires the video data transmitted in shorter time than normal, while fast research generally causes important latency delays that dramatically drop the Quality of Service (QoS).

Many existing designs for VoD streaming address one or more of those points, but none of them really addresses all of them; in particular, most of the known systems do not support interactive functions, and so provide only near VoD service. For this reason, significant advances are still needed to bring the concept of media overlays for VoD streaming to the required level of maturity, in order to develop cost efficient, scalable and reliable solutions with effective QoS.

3.1.2. Live streaming

Live video streaming, or Internet Protocol Television (IPTV), is a service which enables users to watch several TV channels through the Internet. Unlike in VoD streaming, all the users watching the same channel at the same time would have their viewing times synchronized. The only operation that the users should be able to perform is to switch channels.

The first point to address for distributing live video streaming, is also related to bandwidth requirements since the streaming rate has to be maintained for smooth playback. Since live content becomes available as time progresses, it is delivered to each peer in a roughly sequential order. The challenge of streaming is that the demand for bandwidth at the streaming rate must be satisfied at all peers, while additional bandwidth is, in general, not required.

The second point to address is the reception timeliness which is defined as the delay (also known as play-out delay) between the generation at the source of a stream of frames and its reproduction at the receiver. Live streaming requires the maximum play-out delay to be reasonably low, ranging from few seconds to few tens of seconds. The constraint on play-out delay is more bound to the perception of the user that s/he is receiving fresh media data: live streaming deals with information whose interest to the user would rapidly decay if it was delayed too much.

The third point is the media quality which depends on the completeness of the data received before its playback deadline. The qualitative effect of incomplete stream data on the user's playback experience depends on several factors, including the media coding format, the presence of redundant encoding to protect the stream, the ability of the player application to hide discontinuities in the media, and the subjective user sensitivity to visual artifacts.

A lot of live streaming application protocols have been designed during the last few years, but the implicit constraints of this application, both on media quality and timeliness of stream reception, are quite challenging when faced together: different approaches have been proposed so far, striking in all cases a trade-off between these two strongly-correlated factors.

3.2. System architecture

In the construction of interactive VoD systems, the most promising idea is to use media overlay networks on top of the existing infrastructures. These overlays have been evolving from the original client-server model through CDN technologies, on to the P2P networks that permit to deploy media and network adaptive video streaming techniques.

While traditional CDN systems offer a means for media delivery and streaming, they are rather expensive to build and to maintain, and they also pose a significant performance challenge in terms of scalability and service delay as the number of clients increases. To solve this issue, P2P technologies have been applied to support the VoD systems, which is highly cost-effective but do not guarantee high-quality streaming service because of the quality of available data and because the capacities of peers can be heterogeneous and their availabilities can be transient [Guo et al, 2006].

So, since the use of single CDN or P2P systems do not satisfy the true VoD requirements, mostly due to fundamental scalability and complexity limitations of the existing solutions, some authors proposed hybrid CDN/P2P solutions [Guo et al, 2006][Xu et al, 2006] in order to address the scalability problem of CDN-based techniques, and to ensure the high-quality of the delivered media content.

3.2.1. Content Delivery Network (CDN)

The first approach to deal with the scalability issue of video streaming systems is to use a CDN. A CDN is a network infrastructure with several network elements for scaling and enhancing the delivery of content from providers to many end-users over the Internet. CDNs traditionally carry static content, such as HTML pages, images, documents, software patches, audio, and video files. Recently, CDNs have been used to deliver live and on-demand streaming media, for example on video sharing sites such as YouTube [YouTube 2008] and Google Video [Google Video 2008].

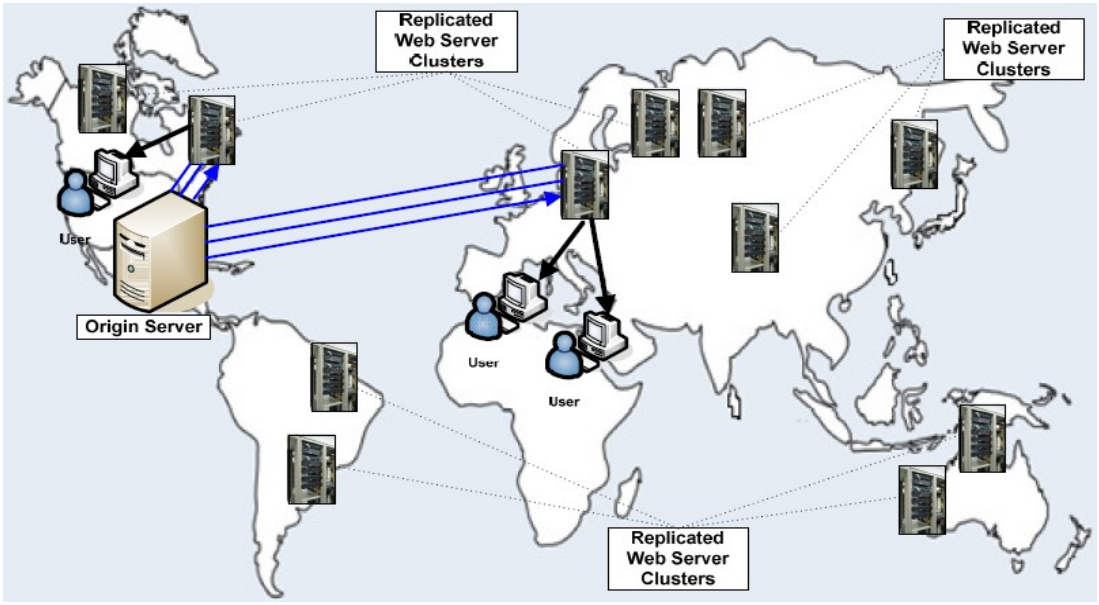

Figure 3.1 Abstract architecture of a Content Delivery Network (CDN) [Pathan et al, 2007]

"CDNs provide services that improve network performance by maximizing bandwidth, improving accessibility and maintaining correctness through content replication. They offer fast and reliable applications and services by distributing content to cache or edge servers located close to users. A CDN has some combination of content-delivery, request-routing, distribution and accounting infrastructure" [Pathan et al, 2007].

The content-delivery infrastructure consists of a set of edge servers (also called surrogates) that deliver copies of content to end-users (Figure 3.1). Edge servers are usually clusters of servers and are located at network operator Points-of-Presence (PoPs). Edge server placement affects the viability of CDNs as it tries to optimize both QoS for the end-user and cost for the CDN provider.

As illustrated on Figure 3.2, "the request-routing infrastructure is responsible for directing client requests (4) to appropriate edge servers (6). It also interacts with the distribution infrastructure (3) to keep an up-to-date view of the content stored in the CDN caches" [Pathan et al, 2007]. Different techniques have been used to guide clients to use a particular replica server, such as HTTP redirection, client multiplexing, DNS indirection, and anycasting. HTTP redirection is the simplest and probably the least efficient means of redirecting requests. The origin server (or server cluster) re-directs the browser to a new URL at the HTTP protocol level. Because it is the only point responsible for redirecting requests, it could become a bottleneck. In client multiplexing, the client receives the addresses of a set of candidate replica servers and chooses one to send the request. This imposes additional overhead in sending the set of candidate replica servers to the client, and, due to lack of overall information, the client may choose a server with high load, which could result in overloading servers and hence larger access latency. DNS indirection uses Domain Name System (DNS) modifications to return the IP address of a replica server when the DNS server is queried by the client. The quality of server selection can be improved by taking into account the server

performance. In Anycasting, an anycast address/domain name, which can be an IP anycast address or a URL of content, is used to define a group of servers that provide the same service. The client sends packets with the anycast address in the destination address field. The packet is then routed via anycast-aware routers to at least one of the servers identified by the anycast address [Peng 2003].

“The distribution infrastructure moves content from the origin server to the CDN edge servers and ensures consistency of content in the caches (2-3)” [Pathan et al, 2007]. The system can use a full replication or a partial replication technique. In the simplistic approach of full replication, all of the origin server's content is copied to each edge server. In the more complex approach of partial replication, only some of the origin server's content is copied to the edge servers. Selection of content in partial replication can be on the basis of heuristics, popularity [Chen et al, 2003], or sensitivity to QoS [Wu et al, 2006]. Distributing content to replica servers can be done using the Internet or using broadcast satellite. Internet distribution of content is simpler but might suffer from the unpredictability and problematic performance of the Internet itself. Data satellite broadcast has the potential for remarkable cost savings, and it provides a high-quality, predictable performance path for sending critical content such as real-time streaming media [Peng 2003]. “Limelight Networks uses a dedicated high-speed network to span the globe, interconnecting all of its regional data centers over a fiber-optic backbone. This allows Limelight Networks to deliver the most massive of files” [Limelight 2008].

“The accounting infrastructure maintains logs of client accesses and records the usage of the CDN servers (7). This information is used for traffic reporting and usage-based billing (8)” [Pathan et al, 2007]. Typical metrics are: cache hits, origin server upload bandwidth, response latency, edge server utilization, packet loss, and proximity.

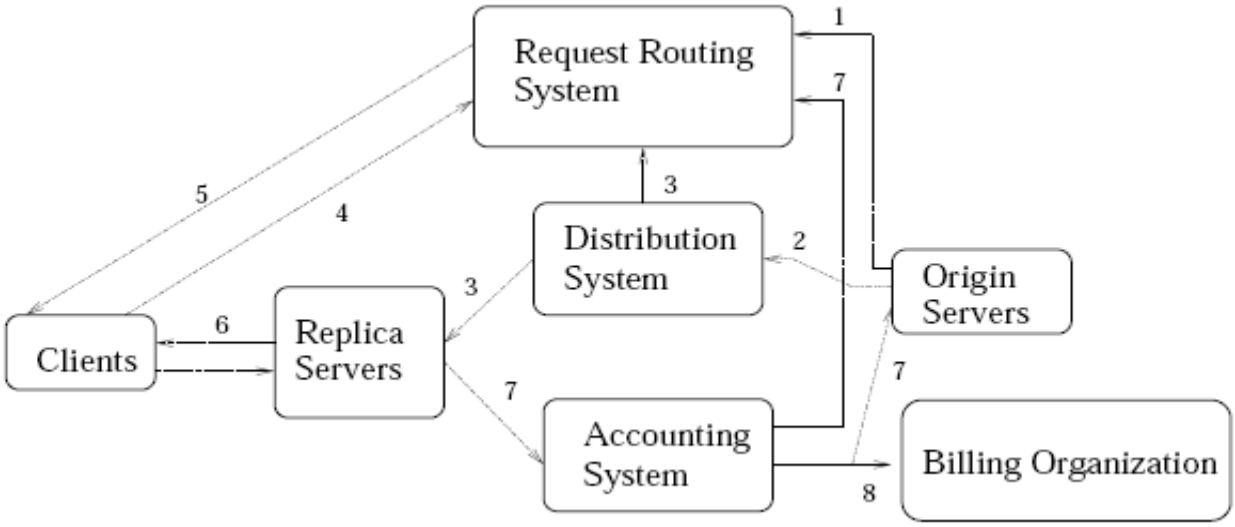

Figure 3.2 Components of a CDN [Peng 2003]

The most important CDNs deployed over the Internet are commercial ones, such as Akamai Technologies [Akamai 2008], Limelight Networks [Limelight 2008], and Mirror Image [Mirror Image 2008]. However some academic CDNs, such as CoDeeN [Pai et al, 2003] and COMODIN [Esteve et al, 2007] have also been deployed.

Akamai is the market leader in providing content delivery services [Vakali et al, 2003] [Pallis et al, 2006]. It owns about 34,000 servers over more than 1,000 networks in 70 countries. LimeLight Network provides a suite of services (including music download and subscription services, video game developers and distributors, movie/video download services, and so forth) and supports surrogate servers located in 72 locations around the world (Asia, the U.S., and Europe). Mirror

Image supports surrogate servers located in 22 cities around the world (North America, Europe, and Asia), which provide a range of value-added services, from content distribution to media streaming and managed caching. Akamai uses a combination of the overlay and network approaches to deliver static and dynamic web content, streaming media, and services. Content is selected using full and partial replication. Limelight Networks follows the overlay approach to deliver static and dynamic web content, as well as streaming media and uses partial replication for content selection. Both use non-cooperative pull for content replication and on-demand content update, and DNS-based request-routing for redirection.

CoDeeN was developed at Princeton University and deployed on 496 PlanetLab nodes. It uses the overlay approach to deliver only static web content. CoDeeN selects content with partial replication and copies content with cooperative pull, but it does not support content update. Redirection is through HTTP and accounting performs internal measurements of traffic and system parameters using a companion system (CoTop) [Pai et al, 2003]. Finally, COMODIN was developed jointly by the Polytechnic University of Valencia and the University of Calabria and deployed on three testbeds connected through the Internet in one domain. It uses the network approach and focuses on delivery and control of on-demand streaming. Content is fully replicated with cooperative pull and periodic updates. COMODIN uses DNS-based request-routing and employs internal network probing and server feedback for accounting [Esteve et al, 2007].

Though rather expensive, CDNs are currently able to provide video streaming services with a satisfying video quality. Nevertheless, they may still raise a scalability issue in case of increasing number of users and very high quality videos encoded at higher bit rates.

3.2.2. Peer-to-peer (P2P) overlays

The second approach to deal with the scalability issue of video streaming systems is to use P2P overlays (Figure 3.3). “P2P networking architectures receive a lot of attention nowadays, as they enable a variety of new applications that can take advantage of the distributed storage and increased computing resources offered by such networks” [Jurca et al, 2007]. During the last years, systems based on BitTorrent [BitTorrent 2008], a second generation P2P file sharing protocol, designed by Bram Cohen [Cohen 2003], have been proved to be a very effective mechanism for content distribution. “P2P systems represent a scalable and cost effective alternative to classic media delivery services, which allows for extended network coverage in the absence of IP multicast or expensive CDNs. Their advantage resides in their ability for self organization, bandwidth scalability, and network path redundancy, which are all very attractive features for effective delivery of media streams over networks” [Jurca et al, 2007].

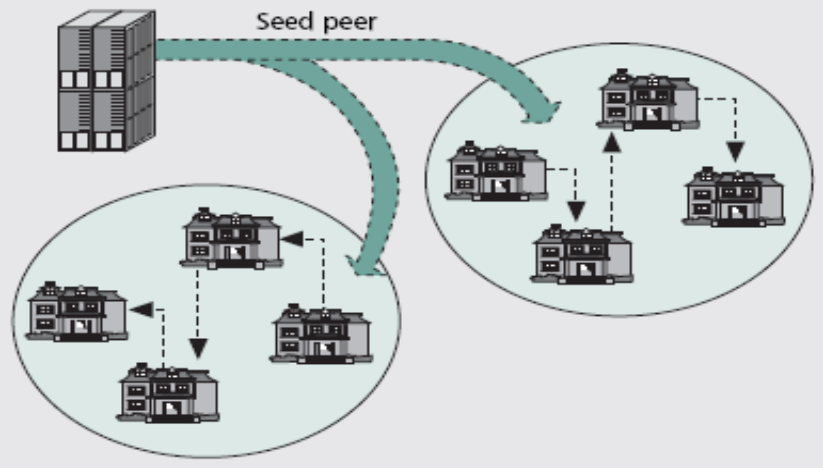

Figure 3.3 A peer-to-peer overlay [Thouin and Coates 2007]

This approach has recently become very popular, and some P2P video streaming systems have been deployed on the Internet, like PPLive [PPLive 2008] and Coolstreaming [CoolStreaming 2008]. The basic principle of P2P video streaming is as follows: a server storing a video that will be viewed by many users chooses to cut the video into several chunks, then send these chunks separately to some users and let them exchange data so that all users receive all chunks of the stream before they are played.

"However, specificities of media applications in terms of bandwidth, delay, and reliability are not completely addressed by the characteristics of unstructured P2P systems. The lack of coordination of such systems, the limited peer capabilities, and the low system stability over time represent a great challenge for the deployment of high quality P2P streaming applications. The replacement or extension of conventional media delivery infrastructures with P2P systems clearly necessitates adaptation of existing coding, routing, and scheduling algorithms to unreliable network environments" [Jurca et al, 2007]. The existing P2P systems suffer from high startup latency, which makes them unsuitable for interactive applications. Indeed, since the download rate depends on the upload rate, a user may have to wait a long time before the received video can be played back continuously.

3.2.3. Hybrid CDN/P2P overlays

Infrastructure-based CDNs with dedicated network bandwidths and hardware supports can provide high-quality streaming services, but at a high cost. Client-based P2P networks are scalable, but do not guarantee high-quality streaming service due to the transient nature of peers. In order to address the limitations of both CDN and P2P approaches, there have been attempts to combine CDN and P2P overlays. Such hybrid solutions have been proposed in [Xu et al, 2006] and [Guo et al, 2006]. Also, though the mechanisms behind it are still unrevealed, based on extensive experiments, Joost [Joost 2008] appears to be a server-assisted P2P VoD system.

In [Xu et al, 2006], the authors study how the upload capacity of peer nodes can be exploited in such architectures. The paper investigates different contribution policies for the peer population and the dynamics of the instance when no CDN assistance will be needed to serve clients that connect later in the streaming process. In their architecture, the CDN server plays the roles of the actual media streaming server and of the P2P index server. The media file is first distributed by the CDN server; while fulfilling requests for the first clients, the server dynamically creates supplying peers that can then serve subsequent requests with higher capacity than the server itself; finally, when the P2P capacity reaches a certain level, the CDN server stops serving streaming sessions and lets the peers take over the task (this transition is called "CDN-to-P2P" handoff).

In [Guo et al, 2006], the authors propose a system called PROP (for "collaborating and coordinating PROxy and its P2P clients") which attempts to address both the scalability and the reliability issues of streaming media delivery in a cost-effective way. The server provides a dedicated storage and reliable streaming services when peers are not available or not capable of doing so. They also propose a model to analyze the cache redundancy, to give the optimal replica distribution in such a system. Their objective is to keep popular media segments in the server for global sharing, and leave a certain space in each peer to cache the relatively unpopular segments.

Although such a hybrid CDN/P2P architecture seems to be very promising, some further aspects should be taken into account, including for example the heterogeneity of users' download and upload capacity, and the dynamics of interaction between servers and clients.

3.3. Delivery Overlay Topology

In P2P streaming systems, "two main types of architectures are generally considered for providing the organization necessary to streaming applications: tree-based overlay for streaming sessions from media sources to a pool of client peers, and mesh overlay for massive parallel content distribution among peers" [Jurca et al, 2007]. CDN overlays are likely to be mostly built upon a tree-based scheme. So the use of a tree-based topology in P2P overlays appears as an extent of the existing topology to the new available resources provided by the cooperating peers.

3.3.1. Tree-based overlays

"Tree-based overlays organize the peers as a single, or multiple tree overlay that connects the source of the media content to the clients. Clients are leaf nodes in the distribution tree, while intermediate peers push the content from the source (Figure 3.4). Single tree architectures are easy to implement and maintain, either in a distributed or centralized way by the source. However, they are fundamentally limited by two factors:

i) due to the high rate of peers joining/leaving the system (the so called churn rate), the architecture suffers from high instability;

ii) the received media quality is limited by the minimum upload bandwidth of the intermediate peers in the branch, since each client is connected to the source through a single tree branch.

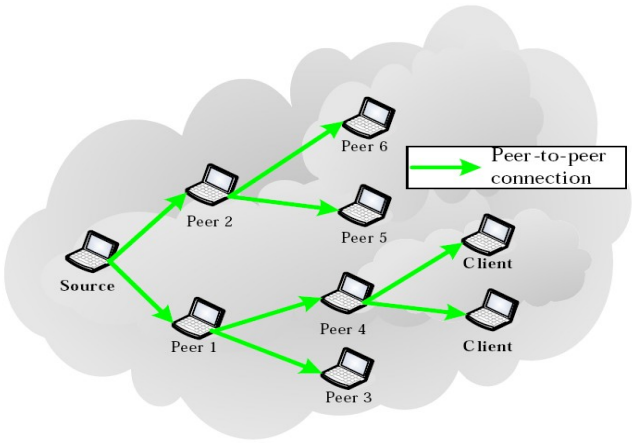

Figure 3.4 A tree-based overlay [Jurca et al, 2007]

Multiple tree architectures address the aforementioned problems, by providing redundancy in network paths. However, designing and maintaining such systems becomes less trivial" [Jurca et al, 2007].

3.3.2. Mesh-based overlays

Mesh-based overlays are (Figure 3.5) "based on self organization of nodes in a directed mesh that is used for media delivery to clients. The original media content from a source is distributed among different peers. A peer is connected to the mesh through one or more parent peers, where it retrieves media information, and to a set of child peers to which it serves media packets. The advantages of such an architecture reside in the low cost and simplicity of structural maintenance, and in the resilience of the topology to node failure or departure, due to the increased probability of available distinct network paths.

However, streaming applications over such architectures face important challenges. First, due to the inherent sequential media encoding and play-out, packet dissemination and data requests must

follow closely the temporal ordering of the content at the source. This constraint may be slightly reduced by the implementation of play-back buffers, when delays permit it. Second, the limited look-ahead content availability, especially in the case of live streaming scenarios, limits greatly the flexibility in terms of content download/upload through such an architecture" [Jurca et al, 2007].

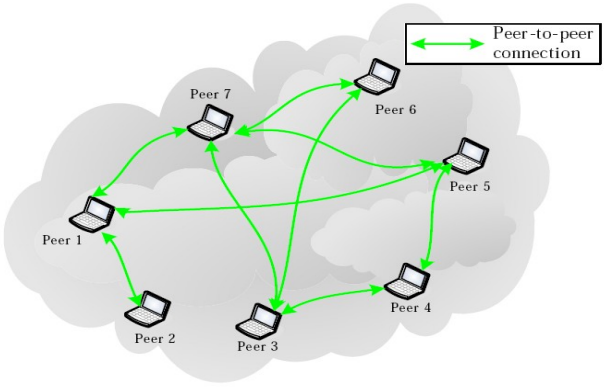

Figure 3.5 A mesh-based overlay [Jurca et al, 2007]

In [Carra et al, 2007], the authors study fundamental properties of stream-based content distribution services in the presence of an overlay network such as those built by P2P systems with limited degree of connectivity, and develop a mathematical model that captures the essential features of overlay-based streaming protocols and systems. Their methodology is based on stochastic graph theory. They model the streaming system as a stochastic process, whose characteristics are related to the streaming protocol. Results show that mesh based architectures are able to provide bounds on the receiving delay and maintain rate fluctuations due to very slow system dynamics.

3.4. Data exchange designs

There are two major approaches of data exchange designs in P2P-based systems: push-based systems where the data are pushed from data owners to other peers, and pull-based systems where a peer has to explicitly request a missing chunk.

3.4.1. Push-based systems

In a push-based system, a peer actively pushes a received chunk to its neighbors who have not obtained the chunk yet. In a tree-based system, a chunk should always be pushed from a peer to all its children peers in the streaming tree. In a mesh-based system, as there is no clearly defined parent-child relationship between peers, a peer might blindly push a chunk to a peer already having it. It might also happen that two different peers push the same chunk to one same peer. Peer uploading bandwidth would then be wasted in redundant pushes. To address this problem, chunk push schedules need to be carefully planned between neighbors, and the schedules need to be reconstructed upon neighbor arrivals and departures [Liu et al, 2008].

3.4.2. Pull-based systems

In a pull-based system, peers exchange chunk availability using buffer maps periodically. A buffer map contains the sequence numbers of the chunks currently available in a peer's buffer. After obtaining buffer maps from its neighbors, a peer can decide a chunk pull schedule that specifies from which peers to download which chunks. Then it will send requests to its neighbors to pull missing chunks. Redundant chunk transmissions can be avoided since a peer will request to download a missing chunk from one neighbor only. However, frequent buffer map exchanges and

pull requests incur more signaling overhead and might introduce additional delays in chunk retrieval [Liu et al, 2008].

3.5. Video Data Coding techniques

Video coding deals with the representation of video data for storage and transmission. The goals are to accurately and compactly represent the video data, to provide means to navigate the video, and to possibly add other additional author and content benefits such as text, meta information, and digital rights management. A big challenge for video coding is to reduce the size of the video data using video compression. The video stream is encoded in one or more substreams which themselves can be decoded to reconstruct the original stream.

In P2P streaming systems, a node can receive the video data either from one particular peer or from multiple peers in parallel. In case downloading from multiple peers is combined with data coding, a client can receive the different substreams of a same video block from different parent sources. It should then reassemble and decode the substreams before playing back the video.

Different data coding techniques exist, including single layer coding, layered coding, and Multiple Description Coding (MDC). Redundancy-Free Multiple Description (RFMD) coding has also been proposed as an enhancement for multiple description coding.

3.5.1. Single Layer Coding

In a single layer scheme, the video is encoded into a single layer, and each Group of Pictures (GOP) is divided sequentially into several blocks, so that earlier blocks contain data from earlier frames in the group. Each peer holds one block from each group. In case one block is not available, the frames contained in the following blocks of the group are not decodable either (assuming the video is coded using temporal prediction), and only the frames contained in the previous blocks can be decoded and frozen until the next group. For example, if the second block is not available, only the first block can be decoded [Liu et al, 2007].

3.5.2. Layered Coding

“Layered coding generates multiple layers with recursive dependency. Specifically, layer k + 1 can only be decoded if layers 1 through k are available. MPEG-4 Fine Grain Scalable (FGS) is a popular scheme for creating layered coding. An FGS encoder encodes the video into a base layer and a scalable enhancement layer. The enhancement layer is then sliced into M −1 substreams, creating a total of M substreams. The latest scalable coding standard, known as Scalable Video Coding (SVC), also has a base-layer and a successfully refinable enhancement layer. With FGS or SVC, the rate of the base layer must be sufficiently high so that an acceptable quality can be recovered from the base layer only. Compared to a single-layer coder, the distortion achievable by a scalable coder at the same rate (above the base layer) is typically higher” [Liu et al, 2007].

3.5.3. Multiple Description Coding

“MDC consists in constructing several independent descriptions of the same signal. In this approach, a controlled level of redundancy is left in the media content at compression, so that the received video quality is proportional to the number of descriptions that are received. MDC represents a natural solution for multi-path streaming scenarios, where independent descriptions can be sent on disjoint paths. It is generally less efficient than scalable encoding in terms of compression; however, it exhibits stronger resilience to packet loss” [Jurca et al, 2007]. A popular

scheme for multiple description encoding with many descriptions is Multiple Description source coding through Forward Error Correction codes (MD-FEC) [Liu et al, 2007].

"The first step of MD-FEC is to encode each GOP into M layers. This can be performed with a scalable video coder such as MPEG4 FGS or SVC. This is shown in Figure 3.6 (a) for the case of 4 layers. Denote by $L1$, $L2$, $L3$, and $L4$ for the bits in these 4 layers. The kth layer is then further divided into k equal-length groups. Thus, as shown in Figure 3.6 (b), layer two is broken into two equal-size groups $L21$ and $L22$; layer three is broken into three equal-size groups $L31$, $L32$ and $L33$; and layer four is broken into four equal-size groups $L41$, $L42$, $L43$ and $L44$. Then a (M, k) Reed-Solomon (RS) code is applied to the k groups from layer k to yield M groups. The M^2 groups are then arranged as in Figure 3.6 (c). For layer 1, the RS step creates three redundant groups $R11$, $R12$, and $R13$; for layer 2 it creates two redundant groups $R21$ and $R22$; and for layer 3 it creates 1 redundant group $R31$. Thus, due to the RS code, if any k of the M groups is received for layer k, then layer k can be decoded" [Liu et al, 2007].

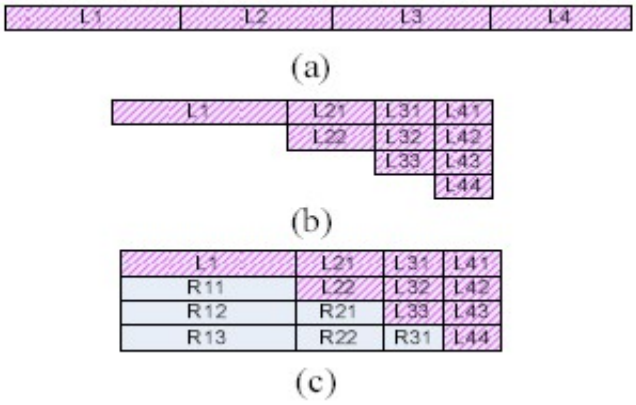

Figure 3.6 MD-FEC encoding procedure [Liu et al, 2007]

"After creating the M^2 groups, the substreams are generated by combining the groups across the rows in Figure 3.6 (c). For example, the first substream is created by combining $L1$, $L21$, $L31$ and $L41$; the fourth substream is created by combining $R13$, $R22$, $R31$ and $L44$. From this construction, it is easy to see that the substreams have the following desirable properties:

- Each substream has the same bit-rate;
- In order to recover k layers from the original layer encoded video, the receiver needs to receive any k of the M substreams. Thus each stream is of equal importance" [Liu et al, 2007].

One nice feature of MD-FEC as a multiple description technique is that it can generate any number of substreams (any M) from a scalable stream generated by any scalable coder, which is desirable for P2P VoD. When a supplying peer disconnects, the video quality is not severely degraded while waiting for a substitute peer to supply the substream. Note that when a receiver receives k of the M substreams, only a fraction of bits are used (for all values of k). what is inefficient in that it wastes significant upload bandwidth resources [Liu et al, 2007].

3.5.4. Redundancy-Free Multiple Description Coding

RFMD is a new multi-stream coding scheme proposed in [Liu et al, 2007]. This proposal is based on the fact that in P2P VoD, which does not have stringent delay constraints, it is not necessary to always transmit all the encoded bits; the amount of redundancy and unhelpful bits that are transmitted should adapt to the number of available supplying peers. This new coding scheme is derived from traditional MD-FEC coding.

The coding procedure proceeds as follows:

- Data is first encoded using MD-FEC to generate M descriptions;
- If m supplying peers are available, then each supplying peer only transmits a fraction k/m portion of the data for layer k, where $k = 1, \ldots, m$; each supplying peer transmits different portion of layer k data;
- The receiving peer combines the received m substreams and obtains layer k ($k = 1, \ldots, m$) by (M, k) FEC decoding; hence, the lowest m layers are recovered.

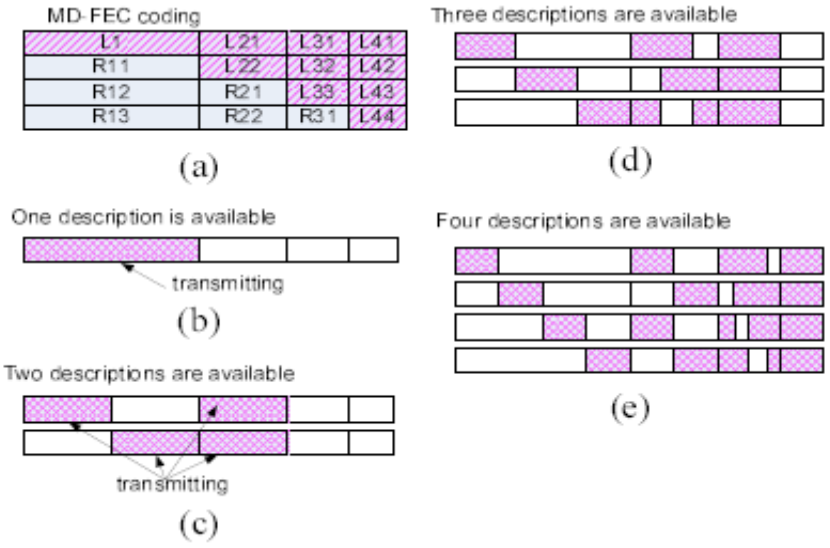

Figure 3.7 Redundancy-Free Multiple Description Coding (RFMD) [Liu et al, 2007]

"Figure 3.7 shows the RFMD transmission of MD-FEC data (M=4): (a) shows the stored data in each description, with purple color representing the source bits and gray color the redundancy bits; (b-e) shows the portion of the data (purple) delivered by each supplying node" [Liu et al, 2007].

Because the portion of data transmitted at each supplying peer depends on how many peers are available, generally, the transmission rate at each peer is not constant. However, constant bit rate can be achieved with the appropriate rate partitioning for the different layers before MD-FEC encoding [Liu et al, 2007].

3.6. Peer choice and data scheduling protocol

In a P2P architecture, a peer may implement various strategies for selecting which peer(s) to collaborate with, and which data to exchange with the selected peer(s). Among these strategies, gossip protocols are the most popular ones, but other protocols have also been proposed or implemented.

3.6.1. Gossip protocols

Gossip algorithms have become popular solutions to multicast message dissemination in P2P systems. According to [Kermarrec et al, 2007], gossiping could be described as a system where a node which has just been updated tells a number of other replicas about its update. The updated node first contacts another one; if the second node already knows the update, it stops; otherwise it contacts another randomly selected node.

According to [Montresor 2008], the protocol can be modeled by means of two separate threads executed at each node: an active one that takes the initiative to communicate, and a passive one accepting incoming exchange requests. In the active thread, a node periodically selects a peer node

p from the system population; it extracts a summary of the local state and finally, sends this summary to node p. This set of operations is repeated forever. The other thread passively waits for incoming messages, replies in case of active requests, and modifies the local state. This scheme being too generic must also be associated with the following rules to distinguish gossip from non-gossip protocols; peer selection must be random, or at least guarantee enough peer diversity; only local information is available at all nodes; communication is round-based (periodic); transmission and processing capacity per round is limited; and all nodes run the same protocol.

"The main reason that make gossip protocols popular is robustness: node failures do not cause any major havoc to the system, and can be tolerated in large quantity; message losses often cause just a speed reduction rather than safety issues. Low-cost is another plus: load is equally distributed among all nodes, in a way such that overhead may be reduced to few bytes per second per node. The cause of such robustness and efficiency can be traced back to the inherently probabilistic nature of gossip protocols. They represent a certain "laid-back" approach, where individual nodes do not take much responsibility for the outcome. Nodes perform a simple set of operations periodically, they are not aware of the state of the entire system, only a very small (constant) proportion of it, and act based on completely local knowledge. Yet, in a probabilistic sense, the system as a whole achieves very high levels of robustness to benign failures and a favorable (typically logarithmic) convergence time" [Montresor 2008].

3.6.2. Push-based protocols

In [Bonald et al, 2008], the authors focus on push-based diffusion schemes where the transmission of a chunk between two peers is initiated by the sender. Each peer has only a partial knowledge of the overall system and can only send chunks to one of its neighbors. Push-based schemes may be broadly categorized into two classes depending on whether the destination peer or the chunk is selected first. The authors analyze several peer and chunk selection schemes:

Random peer: The destination peer is chosen uniformly at random among the neighbors of the sender;

Random useful peer: The destination peer is chosen uniformly at random among those neighbors that did not yet receive all the chunks owned by the sender. When the chunk is selected first, the choice of the destination peer is restricted to those neighbors that do not own the selected chunk;

Most deprived peer: The destination peer is chosen uniformly at random among those neighbors that did not yet receive a maximum of the chunks owned by the sender. When the chunk is selected first, the choice of the destination peer is restricted to those neighbors that do not own the selected chunk;

Latest blind chunk: The sender peer chooses the most recent chunk in its collection;

Latest useful chunk: The sender peer chooses the most recent chunk in its collection that has not yet been received by at least one of its neighbors. When the destination peer is selected first, it chooses the most recent chunk in the set of chunks not yet received by the selected peer.

Random useful chunk: The sender peer chooses uniformly at random a chunk in its collection that has not yet been received by at least one of its neighbors. When the destination peer is selected first, the chunk is chosen uniformly at random in the set of chunks not yet received by the selected peer.

A rich class of push-based schemes can be built from the combination of these peer/chunk selection algorithms. The authors prove that the so called random peer, latest useful chunk algorithm can achieve dissemination at an optimal rate and within an optimal delay, up to an additive constant term. This qualitative result suggests that live streaming algorithms where peers are responsible for disseminating the data can achieve near-unbeatable rates and delays [Bonald et al, 2008].

3.6.3. Topology aware protocol

In [Hefeeda et al, 2003], the authors compare three possible techniques for selecting peers from a set of candidates: on one hand a generic random technique, and on the other hand an end-to-end technique and a topology-aware technique they further describe. A random technique randomly chooses a number of peers that can fulfill the aggregate rate requirement, even though these peers may have low availability and share a congested path. The end-to-end technique chooses the active set based on the quality of the individual paths and on the availability of each peer. It does not consider shared segments among paths, which may become bottlenecks if peers sharing a tight segment are chosen in the active set. In contrast to the end-to-end technique, the topology-aware technique infers the underlying topology and its characteristics and considers the goodness of each segment of the path. Thus, it can make a judicious selection by avoiding peers whose paths are sharing a tight segment.

The topology-aware selection technique is based on goodness topology, a directed graph that interconnects the candidate peers and the receiving peer. Each edge (or segment) is annotated with a goodness random variable. Each leaf node represents a peer from the set of candidate peers and has two attributes: a fixed offered rate and a random variable that describes its availability for streaming [Hefeeda et al, 2003].

"The goodness topology is built in two steps. In the first step, network tomography techniques are used to infer the approximate topology and annotate its edges with the metrics of interest, (e.g., loss rate, delay, and available bandwidth), what is called the inferred topology. A segment in the inferred topology may represent a sequence of links with no branching points in the physical topology. The inferred topology is a tree-structured graph rooted at the receiver. The routes from candidates peers to the receiver are assumed not to change during the course of the streaming session. The second step transforms the inferred topology to the goodness topology. The transformation process is basically computing a "logical" goodness metric for each segment from its properties.

Segment goodness is a function of one or more properties of the segment, depending on the feasibility and ease of measuring these properties segment-wise. Segment properties may include loss rate, delay, jitter, and available bandwidth. In the proposed scheme, the segment goodness is represented as a function of the loss rate and available bandwidth because these two metrics can be measured segment-wise, and are the most influential on the receiving rate, and hence on the quality. A segment with high available bandwidth and low loss is unlikely to introduce high jitter or long queuing delay.

Peer goodness is a function of the availability of a peer, and of the goodness of all segments comprising the path to the receiver. Peers with high expected goodness values indicate that these peers are likely to provide good and sustained sending rate. This is because they are unlikely to stop sending packets and these packets will be transmitted through network paths of low dropping probability. The best active peers set is the subset of peers that are likely to provide the "best" quality to the receiver. The perceived quality is quantified by the aggregated receiving rate.

Once a node has built this goodness topology, then the peers selection problem can be stated as finding the set of active peers that, given the annotated goodness topology, maximizes the expected aggregated rate at the receiver, provided that the receiver inbound bandwidth is not exceeded. The system then determines the expected aggregated rate for all possible active sets and selects the one with the highest rate" [Hefeeda et al, 2003].

Instead of building the underlying topology, the end-to-end technique uses the end-to-end path bandwidth and loss rate in addition to peer availability. It exploits no information about the path segments shared among peers and therefore imposes less overhead than the topology-aware

selection, but while better than random selection, it does not perform as well as the topology-aware one does [Hefeeda et al, 2003].

3.6.4. Streaming Scheduling Algorithm

In [Zhang et al, 2006], the authors proposed a streaming scheduling algorithm for data-driven overlay network, where data exchange is dynamically determined according to data availability and not restricted in specific directions. In data-driven overlay network based streaming applications, the protocol for data exchange between peers contains two steps: the first step is overlay construction, in which each node independently selects its neighbors so as to form an unstructured overlay network, and notifies its neighbors what streaming data it has; the second step is the streaming scheduling, in which each node retrieves the missing data from its neighbors according to the notification of data availability, while it sends the available data to its neighbors if requested. The peers then go on notifying data availability, and requesting and sending data.

Since all nodes are equivalent and independent in data-driven overlay networks, this two-step scheme is robust with minimum maintenance cost for the entire system, and its performance relies on the algorithms in these two steps. In their work, the authors focus on the second step and present an analytical model and the corresponding solutions to tackle the streaming scheduling problem in data-driven overlay networks.

Each node should optimally decide from which neighbor it asks for each block and optimally allocate its limited outbound bandwidth to every neighbor, in order to maximize the throughput with heterogeneous bandwidth constraints in such an unstructured overlay network. To maximize the throughput of the system, the approach is to increase the number of blocks that is requested successfully under bandwidth constraints as much as possible within every scheduling period.

The priority of the blocks is defined using two factors: as “rarest-first” as been shown as one of the most efficient strategies in data dissemination, rarity factor is used as the first factor. When a streaming application has real-time constraint, emergency is considered as the second factor: a block that is in danger of being delayed beyond the deadline should be with more priority than the one that is just entering the exchanging window.

Then the target is to maximize the average priority sum of each node, while ensuring that

- the block scheduling should satisfy the inbound, outbound, and end-to-end available bandwidth constraints,
- each block should be fetched from at most one neighbor so that no duplicate blocks arrive.

This formulation can be transformed into an equivalent minimum cost network flow problem [Zhang et al, 2006].

However, the solution of this optimization problem is centralized and requires global knowledge, making it impractical in real systems. Based on its basic idea, a heuristic algorithm which is fully distributed and asynchronous is proposed with only local information exchange. Simulation results indicate that this distributed algorithm outperforms other different existing methods by about 10%~80% gains in high streaming rate [Zhang et al, 2006].

3.7. Content repository

In CDN-based video streaming systems, the video data is provided to the client node using a client-server scheme; all the blocks are transmitted in a sequential order according to the current playback location. In a push-based P2P streaming system, the client node receives the video blocks from its parent(s) and forwards/pushes them to its child(ren); it does not have to find each block of data, but

needs to find (a) parent(s) which will be able to send it the appropriate data. In a pull-based P2P streaming system, the client node needs to be provided a list of potential supplying peers for each block of data it is interested in. So P2P systems must have a content repository in order to enable the clients to locate the peers that own the appropriate video content. The basic solution is the use of a kind of central database, owned and managed by the server. But other solutions, based on decentralized architectures like Distributed Hash-Tables (DHT) or Dynamic Skip Lists (DSL) also exist to address this problematic.

3.7.1. Centralized database

A central database at the server side is the basic approach for managing content repository in a P2P streaming system for VoD. The server manages an index table in a convenient way, possibly a hash-table, based on a key such as video and segment IDs, with the list of peers owning each particular segment. When a client node needs to find a segment, it just asks the server which provides a list of possible seeds. Peers should also warn the server about all changes in the data they can share to other peers. This is not mostly efficient because all the transactions regarding data location must go through the server which could become a bottleneck.

3.7.2. Distributed Hash-Table

“A DHT is a structured overlay constructed among peers. It works like a traditional hash table, that is, given a hashed key, it returns the corresponding object or its location. The difference is that the table entries are not located in the same place but distributed among the peers in the network. With a proper routing mechanism, the DHT supports primitive functions such as: *put(id, object)*, *get(id)* and *delete(id)*, where *id* is the object’s identifier, as in a traditional hash table” [Yiu et al, 2007].

“Basically, a DHT is designed as a circular, double-linked list. Each node keeps a reference to the next and previous nodes in the list. For each node in the list, the next node is the node whose ID is closest to but still greater than the current node's ID, except for the node with the greatest ID whose successor is the node with the smallest ID.

Each node is itself a standard hash table. To store or retrieve a value from the hash table, one just needs to find the appropriate node in the network, then do a normal hash table store or lookup there. To determine which node is appropriate for a particular key, one should take the key, hash it to generate a node ID-like key, and find the node whose ID is closest to but still greater than the hashed key. This node is the one responsible for storage and lookup for that particular key.

For a new joining node, the first step is to look up the successor of the new node's ID using the normal lookup protocol. The new node should be inserted between the found successor node and that node's predecessor. The new node is responsible for some portion of the keys for which the predecessor node was responsible. Leaves are very simple; the leaving node copies all of its stored information to its predecessor. The predecessor then changes its next node pointer to point to the leaving node's successor. In a DHT, the insertion and removal of nodes is independent of the insertion and removal of data.

However, with only one link to the previous and next node, the performance is $O(n)$ with an expected performance of $n/2$. In order to achieve better performance, instead of storing a pointer to the next node, each node can stores a “finger table” containing the addresses of k nodes. The distance between the current node's ID and the IDs of the nodes in the finger table increases exponentially. Each traversed node on the path to a particular key is closer logarithmically than the last, with $O(\log n)$ nodes being traversed overall. When doing lookups, one now has k nodes to choose from at each hop, instead of only one at each. For each node visited from the starting node, one follows the entry in the finger table that has the shortest distance to the key.

Despite the apparent chaos of periodic random changes to the membership of the network, DHTs make provable guarantees about performance" [Stoica 2003].

3.7.3. Dynamic Skip List

In [Wang and Liu 2008], the authors propose the use of a Dynamic Skip List (DSL), a randomized structure consisting of a set of layers. Each new node, with its playback offset as a key, first joins a base layer, and then randomly and independently promotes itself to upper layers. Logical links to its neighbors in each layer are set up during this promotion process, which can then be used to quickly locate nodes with the expected keys through a fast skipping operation.

"The salient features of a DSL are the following:

1) Its probabilistic nature eliminates costly re-balancing operations after nodes join or leave, making it a highly efficient and adaptive structure;
2) Its parallel logical links have inherent power to support multi-peer collaboration in an overlay network;
3) Both the search cost and the state information kept at a node are sub-linear (constant or logarithmic) to the DSL size, suggesting good scalability" [Wang and Liu 2008].

"A skip list is an ordered list with additional, parallel links. Assume there are N keys in the list, indexed from 1 to N . The $i \times 2^l$-th key, $i = 1, 2, \cdots$, $l = 0, 1, \cdots$ will have links to the $(i - 1) \times 2^l$-th and the $(i + 1) \times 2^l$-th keys respectively. This translates to a layered structure, where the link distance between two neighboring nodes in layer l is 2^l" [Wang and Liu 2008].

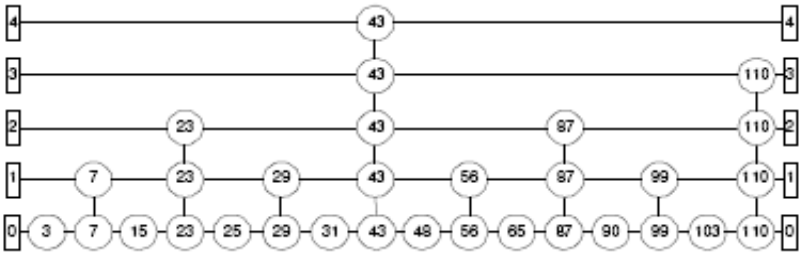

Figure 3.8 A regular skip list with 16 nodes [Wang and Liu 2008]

Figure 3.8 shows a regular skip list with 16 nodes in 5 layers ($l = 0, 1, 2, 3, 4$). Each $i \times 2^l$-th node has been inserted in layer l, and has links to the $(i - 1) \times 2^l$-th node and to the $(i + 1) \times 2^l$-th node. For example, node 23, which is the 4th one (= 2 x 2^1), has on layer 1 links to node 7, which is the 2nd one (= 1 x 2^1), and to node 29, which is the 6th one (= 3 x 2^1).

"In this layered representation, a single key in the list is mapped into multiple logical nodes along the same column. Since the parallel links in higher layers skip geometrically more than those in lower layers, a key search can be started from the highest layer, so as to quickly skip unnecessary parts, and then progressively move to lower layers until it achieves a hit. The complexity of this top-down search is bounded by $O(\log N)$.

A skip list can also be constructed in a random fashion: each key is first inserted into the base layer, and then randomly promotes itself to the upper layer with probability 1/2. If successful, the key will leave a node copy in the previous layer, and try to promote itself again in the new layer until it fails or a *MaxLayer* is met. Assuming it stops at layer l, it will then connect to all the neighbors from layer 0 through layer l. This randomized version achieves the same search performance as the deterministic version when *MaxLayer* is set to $\log(N)$" [Wang and Liu 2008].

A randomized skip list is significantly easier to implement and generally faster than other typical indexing structures, and its probabilistic nature eliminates the need for costly re-balancing operations after each key insertion, making it an attractive solution for distributed applications. Nevertheless, there are some issues for such an application: first, the number of keys in a skip list has to be predefined; second, higher layer nodes often encounter significantly more hits than lower ones and so are vulnerable; and third, the random promotion could generate unbalanced layers, which greatly reduces the maintenance and search efficiency.

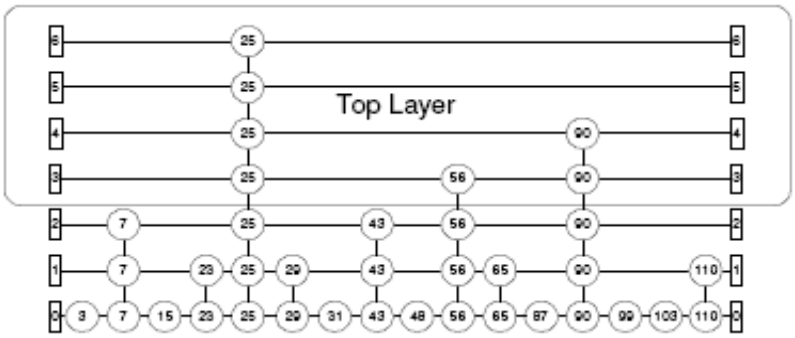

Figure 3.9 A dynamic skip list with 16 nodes [Wang and Liu 2008]

"The DSL addresses those limitations by allowing an adaptive setting for the list size, and effectively compressing unbalanced layers. It is built in a similar way to a regular skip list, but there is no MaxLayer limit: a newly inserted key will stop promoting itself only when it fails. Since the layers above $L = \log (N/\log N)$ might be unbalanced owing to excessive promotions, what means unnecessary neighbor information has to be maintained making the search efficiency potentially quite low across these layers, all the layers above L shall be compressed into a single top layer to solve this problem (Figure 3.9). The number of logical nodes in the top layer being bounded by $O(\log N)$, this small set of nodes can be easily monitored by a single entity, such as the content server in an overlay network. As N is neither fixed nor known a priori in this dynamic list, it should be kept track of the number of the keys to determine which of the nodes belong to the top layer" [Wang and Liu 2008].

In an asynchronous overlay, "the playback offset of a client serves as its key in the DSL, and the set of logical nodes associated with this key all map to the client node in the overlay. The key is updated over time according to the playback progress. Since the playing speed is identical for all the normal clients, their relative distances will not change, unless VCR operations are invoked. The client also maintains the logical links in the DSL, and the content server needs to keep track of all the logical nodes in top layer.

When a new client is to join the overlay, it first contacts the content server, which redirects the client to a top-layer node with the closest key. The new client then performs a top-down search to insert itself into the base layer, and chooses the left neighbor (with earlier playback time) as its supplier in the overlay. It goes on to conduct the bottom-up random promotion, and set up its links to the corresponding neighbors in each layer.

A client that is scheduled to leave the overlay should first notify its neighbors in the DSL such that that they can re-connect with each other to form new neighboring relations. Every client also periodically exchanges echo messages with its neighbors in the DSL, enabling an abrupt client failure to be easily detected. The parallel links in the DSL then enable the affected neighbors to perform local repairs." [Wang and Liu 2008].

"The DSL provides effective support for most of the VCR-like interactions, including fast-forward and rewind, through its horizontal links, which enable a client to skip unnecessary nodes at a fairly stable speed and hence segments" [Wang and Liu 2008]. "The cost for a jump operation is

independent of the overlay size, nor the fast-forwarding/rewinding speed. Such a performance can be difficult to achieve with linear or tree structures, because there are no logical links for efficient skipping nodes with regular distances" [Wang and Liu 2008]. "The maximum speed that DSL can provide is bounded by its highest layer. Nevertheless, a speed of no more than 32x can be easily achieved even in a small DSL overlay, and a higher speed is rarely perceived as useful by users, nor is it supported in most commercial VHS or DVD players" [Wang and Liu 2008].

3.8. Content Upload source

In all the P2P video streaming systems, peers share the video content with each others but the content peers serve vary from one system to one other. Peers can upload from what they see or have just seen, from what they saw during their last session, from something they have no immediate interest in, or from something they have no interest in at all.

The first and most common source of content that the peers are using is their current playing buffer; in other words, they share the data they downloaded for immediate use or the data they used from a few seconds up to some minutes ago, depending on the size of their buffer. This is actually the immediate solution for live streaming [Li et al, 2007] [Wang and Li 2007] [Zhang et al, 2007] where, with regards to one particular channel, all the users are watching (or would like to watch) the same video content at the same time. Data segments are then generally discarded after a few seconds because no more users are likely to be interested in those ones again. The same scheme is also often used for VoD systems [Do et al, 2004] [Zhou and Liu 2005] [Vlavianos et al, 2006] [Guo et al, 2008], but the data segments may remain available much longer, depending on the buffer size. The drawback of using this scheme in VoD system is that, "if a parent jumps to another playpoint in the video, it starts to receive media data which is not interested by its children" [Yiu et al, 2007].

A possible scheme would be to use the content downloaded by the peer during its latest session, since it even requires no additional download and the content is immediately available, but this solution was not used in any of the studied systems.

Besides the buffer or already seen option, the system can also be designed to use the spare bandwidth of the peers for prefetching purpose and forced them to download segments that are of no immediate use to them [Yiu et al, 2007]. Depending on its storage capacity, each peer stores a number of segments of the video. It is not certain that the client will use those stored segments, unless s/he watches the whole video. If the nodes randomly pick the segments to store, the different segments of the video will be uniformly distributed over the network, but the system also has the opportunity to balance the distribution of the segments, for example by the use of a popularity based policy, so that there will be more occurrences of the most requested segments.

Peers can also be forced to download data in which they are no interest at all; for example data from another live streaming channel or from another video file. This scheme is used for VoD in Push-to-Peer [Suh et al, 2006] and is likely to be used for live streaming in PPLive [Vu et al, 2006]. The purpose of such a policy is to balance the bandwidth and storage load needed for all the available content using the bandwidth and storage capacity provided by the whole P2P overlay.

3.9. Peers heterogeneity

"In P2P multicast streaming, it is generally assumed that a peer, acting as a relay, contributes an outbound streaming rate that is at least equal to the full streaming rate. Less effort has been devoted to P2P streaming to an individual requesting peer, under the conditions that supplying peers are heterogeneous and each willing to contribute only a fraction of the streaming rate" [Xu et al, 2006].

Actually, the capacities of peers in terms of access bandwidth, storage and processing power can be heterogeneous and their availabilities can be transient. Systems must handle heterogeneous networks with on one hand nodes located behind residential broadband connections (including cable modem and DSL) having asymmetric upload and download capacities (their upstream rate is significantly lower than their downstream rate), and on the other hand hosts located on high bandwidth university and corporate LANs.

In heterogeneous bandwidth environments that may involve hosts behind DSL and Ethernet, P2P video streaming systems should try to match up nodes with similar bandwidth capabilities, and allow a high-bandwidth node to receive the best download rate from other high-bandwidth nodes, while a low capacity node would receive data from comparatively low capacity nodes only. Otherwise, high capacity nodes might waste bandwidth sending data to low capacity nodes whose download capacity could become the bottleneck. This could lead to decrease in uplink utilization, so clustering similar nodes using matching bandwidth is important to ensure optimal bandwidth utilization [Pianese et al, 2007].

"The bandwidth utilization and thus the delivered quality to individual peers in the mesh-based approach depends on the aggregate quality of available content among their parents. Therefore, as the percentage of high bandwidth peers increases, the performance of the mesh-based approach gradually improves. In the context of tree-based approach, the main determining factor for both utilization and quality is the average depth across different trees. Increasing the percentage of high bandwidth peers rapidly drops depth of all trees which in turn improves both utilization and delivered quality" [Magharei and Rejaie 2007b].

The delivered quality and buffer requirements of high bandwidth peers are affected by the degree of bandwidth heterogeneity and the percentage of low bandwidth peers. When the percentage of high bandwidth peers is small, a larger fraction of their swarming parents consists of low bandwidth peers. This in turn reduces the aggregate available quality among their swarming parents and increases the probability of content bottleneck [Magharei and Rejaie 2007a].

Peers heterogeneity is also an important issue for systems including policies for video segments distribution among peers; if a segment exists at a large number of nodes, the system could consider this segment as being well represented, but actually, if many of the nodes sharing the segment have limited upload capacities, the aggregate delivery capacity for this segment could be rather low. To avoid this problem, segment distribution policies should be adjusted to include the current seeding capacity of the peers [Annapureddy et al, 2007].

3.10. Network coordinates

Network coordinates (or network location) embed inter-node latency in a low-dimensional geometric space, providing an alternative approach to building a latency service. As network conditions change, each node maintains its location in the coordinate space, keeping distances in virtual space an estimate of inter-node latencies.

"There exist two main classes of algorithms for calculating coordinates: landmark-based schemes, in which overlay nodes use a fixed number of landmark nodes to calculate their coordinates, and simulation-based schemes, which are decentralized and calculate coordinates by modeling nodes as entities in a physical system.

In Global Network Positioning (GNP) [Ng and Zhang 2002], nodes contact multiple landmark nodes to triangulate their coordinates. The drawbacks of this landmark-based approach are that the accuracy of the coordinates depends on the choice of landmark nodes and landmark nodes may become a bottleneck. Lighthouses [Pias et al, 2003] addresses this by supporting multiple independent sets of landmarks with their own coordinate systems. These local coordinates map into

a global coordinate system. Practical Internet Coordinates (PIC) [Costa et al, 2004] does not use explicit landmarks, incorporating measurements to any node using a simplex optimization algorithm to obtain an up-to-date coordinate. These landmark-based schemes require a reasonably stable infrastructure.

On the other hand, Vivaldi [Dabek et al, 2004] and Big Bang Simulation [Shavitt and Tankel 2003] determine coordinates using spring-relaxation and force-field simulation, respectively. In both, nodes attract and repel each other according to network distance measurements. The low-energy state of the physical system corresponds to the coordinates with minimum error. A different method for stabilizing coordinates is proposed in [de Launois et al, 2004]: asymptotically dampening the effect of each new Vivaldi measurement. While this factor does mitigate oscillations in a fixed network, it prevents the algorithm from adapting to changing network conditions" [Ledlie et al, 2007].

Network coordinates may be useful in a P2P system where performance relies on accurate latency estimation between peers. In a VoD streaming system, when choosing among a list of potential seeders, a node may want to choose the closest ones because they should be able to deliver the requested data faster and so increase the efficiency of peer-to-peer communications and reduce the system latency.

3.11. Incentives

Collaborative peer-to-peer networks are exposed to abuse of selfish peers which would use the collaborative network without contributing back to it. So some distributed systems, like BitTorrent [Cohen 2003] use incentives to prevent abuse of the overlay network to give preferential treatment to nodes that do contribute. An important source for incentives in cooperative systems is the game theory, and the effective strategies for the iterated prisoner's dilemma.

3.11.1. The Stanford Game Theory

"Game theory is the study of the ways in which strategic interactions among rational players produce outcomes with respect to the preferences (or utilities) of those players, none of which might have been intended by any of them" [Ross 2006].

"Game theory is the most important and useful tool in the analyst's kit whenever she confronts situations in which what counts as one agent's best action (for her) depends on expectations about what one or more other agents will do, and what counts as their best actions (for them) similarly depend on expectations about her" [Ross 2006].

"All situations in which at least one agent can only act to maximize his utility through anticipating (either consciously, or just implicitly in his behavior) the responses to his actions by one or more other agents is called a game. Agents involved in games are referred to as players. If all agents have optimal actions regardless of what the others do, as in purely parametric situations or conditions of monopoly or perfect competition, we can model this without appeal to game theory; otherwise, we need it" [Ross 2006].

"Each player in a game faces a choice among two or more possible strategies. A strategy is a predetermined 'programme of play' that tells her what actions to take in response to every possible strategy other players might use" [Ross 2006].

The Prisoner's Dilemma is a game having its name derived from the following situation typically used to exemplify it. "Suppose that the police have arrested two people whom they know have committed an armed robbery together. Unfortunately, they lack enough admissible evidence to get a jury to convict. They do, however, have enough evidence to send each prisoner away for two years

for theft of the getaway car. The chief inspector now makes the following offer to each prisoner: If you will confess to the robbery, implicating your partner, and she does not also confess, then you'll go free and she'll get 10 years. If you both confess, you'll each get 5 years. If neither of you confess, then you'll each get two years for the auto theft" [Ross 2006].

	Prisoner B Stays Silent	Prisoner B Confess
Prisoner A Stays Silent	Each serves 2 years	Prisoner A: 10 years Prisoner B: goes free
Prisoner A Confess	Prisoner A: goes free Prisoner B: 10 years	Each serves 5 years

Table 3.1 The prisoner's dilemma

The first prisoner evaluates the two possible actions by comparing his payoffs to see which one is preferable for each possible action taken by the second one. If the second prisoner confesses, the first one will go in prison for 5 years by confessing and for 10 years by refusing. If the second prisoner refuses, the first one will go free by confessing and in prison for 2 years by refusing. Therefore, the first prisoner is better off confessing regardless of what the second one does. The second one, meanwhile, evaluates his actions by comparing his payoffs, and he comes to exactly the same conclusion. Thus both players will confess, and both will go to prison for 5 years.

Something disturbing about the outcome of the Prisoner's Dilemma is that if both had refused to confess, they would each have gone to prison for only 2 years, thereby both earning higher utility than they receive when they confess. This is the most important fact about the Prisoner's Dilemma, and its significance for game theory is quite general.

However, if in one-shot games, that is, games in which players' strategic concerns extend no further than the terminal nodes of their single interaction, the only possible outcome is mutual defection, this may no longer hold, in repeated games, that is, games in which sets of players expect to face each other in similar situations on multiple occasions. Actually games are often played with future games in mind, and this can significantly alter their outcomes and equilibrium strategies. Rational players repeatedly interacting for indefinitely long games, repeated Prisoner's Dilemma for example, having memory of at least one previous game, can sustain the cooperative outcome [Ross 2006].

3.11.2. Tit-for-Tat

"Tit for Tat" is one simple and famous, but not necessarily optimal, deterministic strategy for preserving cooperation in repeated prisoner's dilemma. This strategy tells each player to behave as follows:

1. Always cooperate in the first round;
2. Thereafter, take whatever action your opponent took in the previous round.

A group of players all playing tit-for-tat will never see any defections. Since, in a population where others play tit-for-tat, tit-for-tat is the rational response for each player.

But there are two complications. Firstly, the players must be uncertain as to when their interaction ends, otherwise it will be rational for players to defect in the last round since no punishment will be possible. Considering then the second-last round, players also face no punishment for defection, since they know they will defect in the last round anyway. So they defect in the second-last round too, and do so on, until they reach the first round. Therefore, cooperation is only possible in repeated prisoner's dilemmas where the expected number of repetitions is indeterminate. Secondly,

players' ability to distinguish defection from cooperation should be perfect. Otherwise tit-for-tat players could mistake the second case for the first and defect, thereby setting off a chain-reaction of mutual defections from which they can never recover [Ross 2006].

A slightly better strategy can be "Tit for Tat with forgiveness." When the opponent defects, on the next move, the player sometimes cooperates anyway, with a small probability (between 1% and 5%). This allows for occasional recovery from getting trapped in a cycle of defections.

3.11.3. Excess-based

Excess-Based is an economic model to perform resource allocation in P2P networks and defend against malicious participants. This model has been described in [Grothoff 2003] as follows.

Instead of money, the model is based on trust. Each node keeps track of how much it trusts each of the other nodes it has had contact with in the network. The level of trust is measured as a non-negative integer.

“Nodes form their opinions upon requests, and replies. A request is considered network usage, and a reply is considered to be a contribution to the network, so nodes that send large numbers of requests will lose trust, and nodes that send replies earn trust. Each request comes with a priority which defines the amount of trust that the sender node is willing to risk for this request. The receiver of the request can reduce the amount of trust it has in the sender by that amount, and if the receiver can answer the request, the sender will increase its trust in it by the same amount. Note that the receiver will give the request an effective priority which is the minimum between the priority given by the sender and its trust in it [Grothoff 2003].

The main motivation for the introduction of an economic model is resource allocation. If a node is too busy to process all requests, it will decide to drop first the requests with the lowest priority, which are going to be less valuable. This scheme has the desirable property that nodes that contributed to the network will receive better service than nodes that did not contribute. On the other hand, if a node is idle, or its current load is under a certain threshold, it may decide not to charge the sender for the request. However, the sender of the request will still credit the replier for its answer. This is essential for the economic model to work as it makes possible to infuse the network with trust [Grothoff 2003].

It is in the nodes' best interest to have knowledge about their peers’ performance in the past because it helps making good decisions. If they were to lose that knowledge, their decisions would be less informed and thus potentially harmful for them. Note that it does not matter whether a node forgets, ignores or disregards its knowledge about its peers but what it bases its decision on. The only important decision a node makes that depends upon its trust in its peers is which one of two peers it should drop the request from if it is so busy that it can only answer one of them. If a node has no proper records of its peers’ past, it might drop a request from a good host that has answered thousands of its requests in the past in order to serve a less good one; it would so miss a great opportunity to increase its own standing with the good peer which may then decide it is a malicious node and later prefer answering requests from other ones [Grothoff 2003].

4. Existing or proposed systems

A lot of different systems have been proposed for both VoD and live streaming during the last years, so that it is even difficult to draw up a comprehensive list with all of them. In order to give an overview of the proposed solutions, a representative panel comprising a dozen of systems has been selected. Section 4.1 presents a brief description of those systems, based on the selection criteria defined in chapter 3, while section 4.2 presents a detailed analysis of the most representative ones. The systems analyzed in section 4.2 are only listed in section 4.1 without any further description. Section 4.3 presents a global overview of the options chosen in the different systems.

4.1. Description of existing or proposed systems

P2VoD (Peer-to-Peer approach for VoD streaming) is a P2P architecture for VoD streaming proposed in [Do et al, 2004]. Peers are organized in a tree-based overlay, grouped by generations where a generation is a group of peers having always the same smallest numbered block in their cache. Children receive data using a push-based scheme from a single parent, which uploads data from its own playing cache. Parents are proposed to be chosen using a round robin, smallest delay, or smallest distance selection algorithm, from a list obtained from the server. The performance of the system has been evaluated using a simulation method. The system does not support VCR-like operations.

PPLive : this system has been described with more details in section 4.2.3.

CoolStreaming/DONet : this system has been described with more details in section 4.2.1.

HON (Hybrid Overlay Network) is a P2P protocol for VoD streaming proposed in [Zhou and Liu 2005]. This system constructs both a tree overlay and a mesh overlay which collectively deliver the video data to the clients; much of the data is delivered through the mesh overlay while a node will only resort to the tree overlay if it fails to fetch some segment after a certain deadline. Both overlays use a pull-based scheme for data delivery. Nodes are assigned their parent in the tree overlay by a managing node, responsible for constructing and maintaining the overlay. Multiple parent peers in the mesh overlay are chosen using a gossip selection algorithm. The performance of the system has been evaluated using a simulation method. It is just proposed to simulate VCR functionalities by initiating a new subscription request.

BiToS (BitTorrent Streaming) is P2P streaming protocol for VoD proposed in [Vlavianos et al, 2006]. Peers are organized in a mesh-based overlay and data distribution follows a pull-based scheme. Parents are chosen using a tracker program running on a server. Pieces requests and exchanges among peers follow both rarest-piece-first and tit-for-tat policies. Pieces from a particular video file are contained in three components called Received Pieces, High Priority set and Remaining Pieces Set. Pieces from the Received Pieces set can be shared with peers. The performance of the system has been evaluated using a simulation method. The system does not support VCR-like operations.

PROP (collaborating and coordinating PROxy and its P2P clients) is a hybrid proxy/P2P system for VoD streaming proposed in [Guo et al, 2006]. The proxy serves as a persistent cache site, and takes over the streaming whenever the media cannot be served by any peer. Peers are self-organized into a structured overlay and data distribution follows a pull-based scheme. They share the cached media data they are currently playing. The system uses a DHT for content location, a popularity-based proxy replacement policy and a utility-based peer replacement policy. The performance of the system has been evaluated using a simulation method. VCR-like operations have not been studied in the system.

Push-to-Peer is a VoD system for long-lived peers proposed in [Suh et al, 2006]. The authors consider a controlled environment with always-on peers, constant bandwith, and possible centralized control. The content distribution proceeds in two phases: first the content server pushes content to the nodes, then the nodes will pull content of interest from other peers. A peer may store and share content in which it has no interest. The system assumes homogeneous peers. Clients download media data from a small number of peers. The performance of the system has been evaluated using theoretical models. Startup delay is around 22 sec. VCR-like operations have not been studied in the system.

Joost : this system has been described with more details in section 4.2.2.

PRIME is a P2P live streaming system proposed in [Magharei and Rejaie 2007a]. The peers are organized into both a tree and a random mesh overlay and data distribution follows a pull-based scheme. Connections between peers in level n and their children in level n-1 of the tree-based overlay are only used for the diffusion of data units received from level n+1. Parent peers then swarm data units with child peers in the same or higher level. Peers receive media content from multiple sources. The performance of the system has been evaluated using a simulation method.

BulletMedia is a P2P system for VoD proposed in [Vratonjic et al, 2007]. Peers are organized in a mesh-based overlay and data distribution follows a pull-based scheme. Peers are selected uniformly at random. The blocks are selected using the rarest random strategy. The system uses a DHT to store content location within peers. Peers upload from the entire set of chunks associated with the media being played; they use their spare bandwidth to prefetch blocks chosen to ensure good diversity across the overlay. The performance of the system has been evaluated using a prototype implementation. Startup delay: about 20 seconds to cache the initial blocks. The system supports VCR-like operations.

PULSE is a P2P live streaming system proposed in [Pianese et al, 2007]. Peers are organized in a mesh-based overlay and data distribution follows a pull-based scheme. The system uses a primary optimistic tit-for-tat peer selection policy, and an additional excess-based altruistic incentive. Chunks are selected using a rarest first policy by the requester and ordered using a least sent first, random strategy by the sender. The performance of the system has been evaluated using a simulation method and limited deployment of a node prototype.

R^2 is P2P live streaming system proposed in [Wang and Li 2007]. Peers are organized in a mesh-based overlay and data distribution follows a push-based scheme. The system is based on Random Push with Random Network coding. The chunks to be sent are chosen randomly with a preference to the ones from a priority region. Seeds also randomly choose downstream peers for each segment. Unlike other live streaming systems, playback is synchronized for all peers. The performance of the system has been evaluated using a prototype implementation. Startup delay: fill the priority region is less than 6 seconds with initial buffering delay set to 16 seconds.

VMesh : this system has been described with more details in section 4.2.4.

GridMedia is a P2P live streaming system fully implemented to broadcast live TV programs since January 2005 and analyzed in [Zhang et al, 2007]. Peers self-organize into an unstructured random mesh overlay and data distribution follows a pull-push hybrid protocol: a pull-based protocol is first used to form trees along which packets are then pushed. A partial list of current online nodes is given by the server. Packets are requested to peers randomly with the same probability. The performance of the system has been evaluated using a simulation method and a full implementation. The average playback delay is around 25 seconds.

DirectStream is a P2P streaming system for VoD proposed in [Guo et al, 2008]. Peers are organized in a tree-based overlay and data distribution follows a push-based scheme. Parent peer is selected using a distance-bandwidth ratio. Content location use an application-level multicast based directory service called AMDirectory, built on the top of Scribe [Rowstron et al, 2002], which is a decentralized multicast infrastructure. The performance of the system has been evaluated using a simulation method. VCR functionalities are simulated by initiating a new joining process.

4.2. Detailed analysis of representative systems

In this section, we describe a selection of existing or proposed streaming systems. As already mentioned in this chapter, given the significant amount of systems, it was impossible to describe them all in detail as part of the present thesis. So it was decided to limit this section to the analysis of four representative systems: two VoD systems and two live streaming systems. For the VoD part, Joost and VMesh have been chosen: the former for having an hybrid server/P2P architecture, the latter for being P2P only. For the live streaming part, Cool-Streaming and PPLive were selected.

4.2.1. CoolStreaming

CoolStreaming is a P2P live video streaming system presented in [Zhang et al, 2005] as DONet, which stands for Data-driven Overlay Network. “Since the first release (CoolStreaming v0.9) as a public Internet-based implementation in March 2004, it has attracted millions of downloads worldwide. The peak concurrent users reached over 80,000 with an average bit rate of 400 Kbps, with users from 24 different countries” [Li et al, 2007]. It is currently deployed on [CoolStreaming 2008], and its implementation is platform independent and supports Window Media Video (WMV) or RealMedia Variable Bitrate (RMVB) formats. Its internal architecture, dynamics and performance have further been studied in [Li et al, 2007].

“CoolStreaming represented one of the earliest large- scale P2P video streaming experiments, which was built on the notion of data-driven, somewhat similar to the technique used in BitTorrent [Cohen 2003] but with much more stringent timing and rate constraints” [Li et al, 2007].

“The system consists of five basic components:

1) the Membership manager, which maintains the partial view of the overlay;
2) the Partnership manager, which establishes and maintains partnership with other nodes;
3) the Scheduler, which is responsible to schedule data transmission across streams;
4) the Buffer, which stores video data before playback;
5) the Buffer Map, which represents the current status of the buffer and data requests” [Li et al, 2007].

DONet is a mesh-based system which has been designed as “data-centric”: a node always forwards data to others that are expecting the data, with no prescribed roles like father/child, internal/ external, and upstreaming/downstreaming; the availability of data that guides the flow directions, while not a specific overlay structure that restricts the flow directions [Zhang et al, 2005].

“For each segment of a video stream, a DONet node can be either a receiver or a supplier, or both, depending dynamically on this segment’s availability information, which is periodically exchanged between the node and its partners. An exception is the source node, which is always a supplier, and is referred to as the origin node. It could be a dedicated video server, or simply an overlay node that has a live video program to distribute” [Zhang et al, 2005].

Node membership is managed as follows: the users first contact a web server to select the program that they intend to watch [Li et al, 2007]. After selecting the program, the newly joining node

contacts the origin node (or boot-strap node), which randomly selects a deputy node (from a set of 24 dedicated servers) and redirects the new node to the deputy, as shown on figure 4.1. The new node then obtains a list of partner candidates from the deputy, and contacts them to establish its partners in the overlay. To accommodate overlay dynamics, each node periodically generates a membership message to announce it existence. A gracefully departing node should issue a departure message. When a node fails, the partner that detects the failure will issue the departure message on behalf the failed node [Zhang et al, 2005]. “An ActiveX component in JavaScript code is used to collect the peer activities as well as status information and reports back to a log server” [Li et al, 2007].

The initial system adopted a simple pull-based scheme for content delivery based on content availability information, what incurred per block overhead and resulted in a longer delay in retrieving the video content. The present one implements a hybrid push-pull mechanism, in which the video content is pushed by a parent node to a child node except for the first block. Each peer only sends a single request for a sub-stream; once the request is accepted, the parent node will push all subsequent data blocks from this sub-stream to the requesting child peer. It remarkably lowers the overhead associated with each video block transmission, reduces the initial delay and increases the video playback quality [Li et al, 2007].

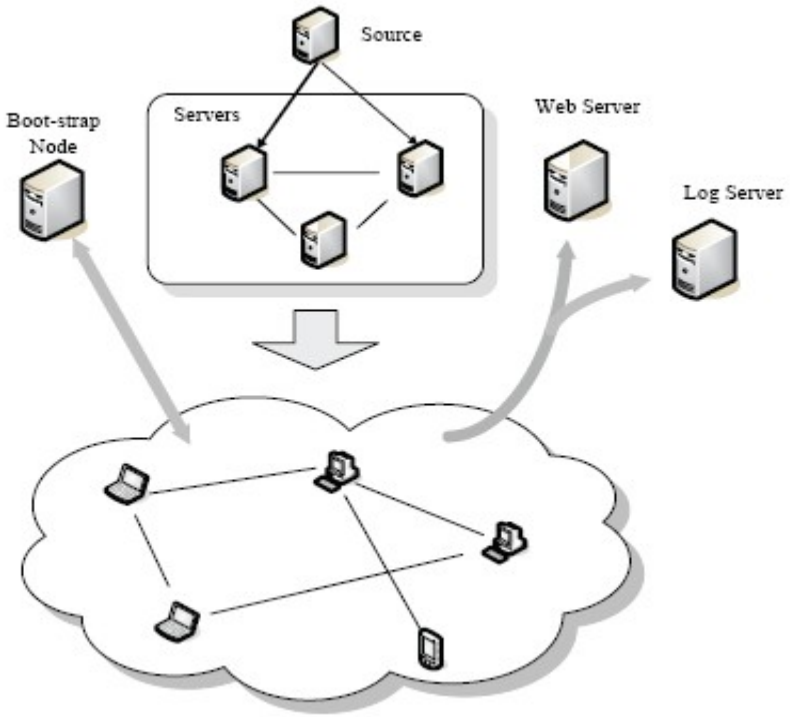

Figure 4.1 CoolStreaming system configuration [Li et al, 2007]

Originally, the system was based on a single-stream scheme. In the current version, a video stream is divided into multiple sub-streams, which essentially enables multi-source and multi-path delivery. It means that a node can subscribe for sub-streams from different partners. This enhances the video playback quality and also improves the effectiveness against system dynamics. A sub-stream is further divided into blocks with equal size, in which each block is assigned a sequence number representing its playback order [Li et al, 2007].

Each node maintains an internal buffer. A video stream is divided into segments of uniform length, and the availability of the segments in the buffer of a node is represented by a Buffer Map. The playback progresses of the nodes are semi-synchronized. Analytical results demonstrate that the average segment delivery latency is bounded and experimental results further suggest that the time

lags between nodes are unlikely higher than 1 minute. If each segment contains 1-second video, a sliding window of 120 segments can effectively represent the buffer a node uses to upload to partners, because a partner should not be interested in the segments that are outside of the window [Zhang et al, 2005].

CoolStreaming uses a gossiping protocol for membership and management based on a basic random partner selection. The newly joined node obtains a list of partner candidates from the deputy node, and contacts them to establish its partners in the overlay. It maintains a membership cache that contains a partial list of the active nodes.

Each node continuously exchange its buffer map with its partners, and then schedules which segment is to be fetched from which partner; the scheduling algorithm strikes to meet two constraints: the playback deadline for each segment, and the heterogeneous streaming bandwidth from the partners. If the first constraint cannot be satisfied, then the number of segments missing deadlines should be kept minimal. The algorithm first calculates the number of potential suppliers for each segment. Since it could be more difficult to obtain a segment with less potential suppliers within the deadline constraints, the algorithm determines the supplier of each segment starting from those with only one potential supplier, then those with two, and so forth. Among the multiple potential suppliers, the one with the highest bandwidth and enough available time is selected. The segments to be fetched from the same supplier are marked in a buffer map-like bit sequence, which is sent to that supplier, and these segments are then delivered in order through a real-time transport protocol [Zhang et al, 2005].

Finally, each node periodically establishes new partnerships with nodes randomly selected from its membership cache. This operation first helps each node maintain a stable number of partners in the presence of node departures; second, it helps each node explore partners of better quality [Zhang et al, 2005].

[Zhang et al, 2005] first present the performance results of DONet experimented on a distributed testbed. Under a stable environment where all the nodes join in an initialization period, the control overhead increases with the number of partners but is fairly limited to less than 2% of the total traffic. The playback continuity improves with an increasing number of partners because each node has more choice for suppliers. It reaches 98% when using 4 partners and improvements with more partners are marginal. The system appears to be scalable in terms of both overlay and streaming rate. Larger overlays obviously lead to better playback continuity due to the increasing degree of cooperation. Under a dynamic environment the control overhead is slightly higher with a more dynamic node behavior; it also leads to poorer continuity but the drop is insignificant.

Then the authors present the results of the first implemented public version of the system called CoolStreaming. Those results reveal that the Internet has enough bandwidth to support TV-quality streaming, and that the larger the data-driven overlay is, the better quality it delivers.

In [Li et al, 2007], the authors present the results of the performance measurement studied using a large set of live streaming traces obtained from the current CoolStreaming system. They showed that: the excessive start-up time and high join failure rates are the critical performance issue in P2P streaming systems. The system dynamics, in particular the peers churn, appeared to affect the overall performance. Finally, they noticed a highly unbalanced distribution in term of uploading contributions from nodes. So the authors conclude that a certain server deployment is of necessity, but as pure server-based approaches like CDN can be costly and do not scale well, a large-scale commercial Internet streaming system should be a hybrid one, using P2P with assistance from distributed servers.

4.2.2. Joost

Joost is a P2P VoD streaming system created by Janus Friis and Niklas Zennström, co-founders of Skype and Kazaa. It was commercially launched on May 1, 2007. The current Joost Beta version (Joost Beta 1.1.8. Release date: July 21, 2008) runs on Windows XP Service Pack 2, Windows Vista, and on any Intel-based Mac running OS X 10.4. It supports more than 28,000 TV shows through more than 480 channels [Joost 2008].

Joost is one of the earliest and best-known commercial P2P VoD products and has the potential to become very popular. It offers high-quality and comprehensive VoD services, and the current version supports an instant on-demand video without any need for additional set top box. However, the mechanisms behind Joost are still unrevealed. The underlying Joost architecture and its key components have been studied, and its media streaming behaviors and peer management mechanisms analyzed in [Lei et al, 2007] through close investigations on its network traffic. Their work has been the source for the information reported below.

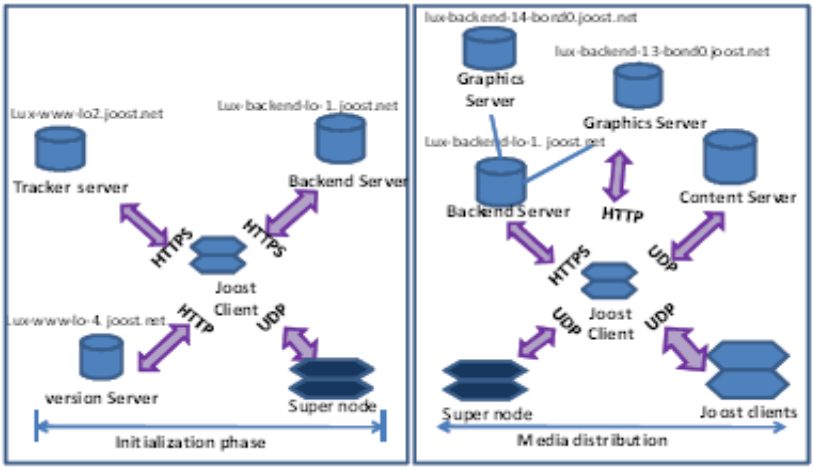

Figure 4.2 Joost architecture [Lei et al, 2007]

Joost is a server-assisted peer-to-peer VoD system that mainly relies on plenty of dedicated infrastructure nodes (e.g. content servers) to distribute video. P2P technologies are used to help distributing video and to extend the system's scalability. Five types of servers are participating in the Joost architecture (Figure 4.2):

1) the version server is responsible for checking the current version of the software; when the Joost client crashes, it is also responsible for the error reports;
2) the tracker server keeps track of membership and helps bootstrapping new peers by sending the initial peer list that includes some of the super nodes and content servers; after that, the tracker server will not appear in any of the other stages;
3) the backend (or control) server performs channel list management and load balancing within the server cluster it is part of with the graphics servers; it also communicates every one minute with the client;
4) the channel graphics servers are responsible for tasks like channel list updating and channel graph downloading actions, especially when the control server is overloaded;
5) the content servers, of which a significant number have been spread over the network, on sites owned by different network solutions and service providers.

Joost uses super nodes to perform the following three basic functions:

1) they direct new joined clients to available peers which are either clients or content servers;

2) for on-demand video functions, they periodically exchange with clients some small UDP packets which are believed to be used for peer management, such as keep-alive probing;

3) channel switching requires the client to talk to them, most likely to help it finding available peers to fetch the new media data.

Super nodes are not responsible for relaying or forwarding media data to other peers.

To connect to the system, a client first communicates with the tracker server from which it will receive some available super node addresses and possibly some content server addresses. Then it requests the version server for getting the latest software version. Finally, it starts contacting some of Joost super nodes to obtain the list of other available clients and begin exchanging video content; before long, it starts communicating with other peers besides Joost servers. After a short period, the backend server appears, and continuously sends a large amount of data to the client in order to update the channel list. When restarted the client proceeds identically to reconnect, except that the version server might not appear if the interval was short, and that it attempts to communicate with peers from which it has downloaded content previously.

Peer selection mechanisms in Joost seem to occur as follows. First, the popularity is considered during the peer selection. Most likely, the client receives a peers list with more available clients if a most popular channel is chosen. Second, during the peer selection, low capacity peers probably connect mostly with low capacity peers except for most popular programs. It is quite similar to the swarm mechanisms used in BitTorrent [Cohen 2003], which allow low capacity nodes to receive data from comparatively low capacity nodes only since otherwise, high capacity nodes might waste resources for sending data to low capacity nodes instead of high capacity nodes. But for the most popular channels, there are enough peer resources so that it would be no problem for low capacity nodes to receive data from some high capacity nodes. Third, if the client requests a seldom channel or it is the only one in that channel, most of the data will be sent from the Joost content server since there are not enough peers which can contribute to the client. Fourth, the geographical locality may have been considered in Joost. Topological locality has however not been realized in the current version.

A client stores the media data in its local cache as “anthill cache” on its system disk. Joost claims the system runs on a media streaming library the company has nicknamed “Anthill”. Here, Anthill [Babaoglu et al, 2002] is an agent-based P2P system to support the media distribution services. The cache size depends on which and how long programs have been played. Each time a new program is chosen, the size of the cache will automatically increase. It sometimes reaches more than 2 GB. Therefore, the user’s system resources could be significantly occupied if the client continues to watch different channels.

The VoD aspect of Joost is particularly interesting. Unlike file sharing or live media streaming, each client only cares about contents after its current playing position, which is often different from other peers. The peer can only download from those whose playback positions are ahead, or from whom have already watched the program. Instead, itself can help peers which join later. However, as each Joost client can change its playback position at any time, which differs from many other P2P streaming systems, it becomes difficult to optimize the overall VoD system. For example, the “rarest-first” strategy [Cohen 2003] in BitTorrent is not applicable here.

In Joost, each media file is broken down into fixed-time chunks. Each chunk is encrypted, and includes an anchor which is a dedicated marker for the encrypted media data. When a seek is triggered in a client, it will always search for the closest anchor in the local video cache if it is already downloaded. Otherwise, it first sets a new anchor and requests new data from other peers. If the fast forward interval is smaller than 5 seconds the client may continuously play without waiting. However, if the interval is large, it takes $5 - 10$ seconds to start playing. So, each media file is

supposed to be divided into multiple 10-second play time chunks, but the exact size of the chunk is unknown.

When the client drags the control bar into any specific position, it communicates with one of the super nodes, supposedly in order to support the VoD functionalities. During the actions of "fast forward" within the same program, there is a large amount of traffic sent from the super node. Otherwise, the traffic from the super node is quite low compared to the "fast forward period"; UDP is used to carry the traffic and the size of both received and sent packets is always below 150 bytes. These packets are supposedly only used for control, not for media transmission. Probably the updated lists, which contain information about peers having already received the on-demand contents, are encoded in these packets.

The channel switching aspect is also interesting. If the program is completely new to the client, it may take up to 20 seconds to really start the program. Most likely, the time is required for requesting contents from other peers and preparing downloading. Otherwise, the program will start immediately after selection since content can be directly fetched from the local video cache. Moreover, the Joost client can browse the channel list and add selected channels into a favorite channel list for client's convenience. It usually takes $7 - 9$ seconds for switching between those channels.

The performance of Joost have been measured on the real network using three test machines located in Germany, following different scenarios [Lei et al, 2007]. The authors demonstrate that with some dedicated infrastructure the current Internet is capable of providing performance requirements of high quality VoD services. However, the performance remains to be improved in the following branches:

- Such an architecture heavily relying on a set of centralized content servers may still raise a scalability issue.
- Joost does not efficiently use the peers' resources, especially when high capacity peers are available. For example, it takes a long time to browse the channel list since it is dynamically downloaded from the server, which is highly overloaded in case of a crowded browsing. High capacity peers could be used for providing such a service.
- The authors noticed program unavailability over five times during experiments; at those times, all programs were unavailable for up to 30mins. So it seems that the current Joost P2P technology is not always reliable.
- Joost currently provides each client with the same quality of video, which may result in an inefficient resource utilization if some clients are unable to support the desired video quality.

4.2.3. PPLive

PPLive is a free P2P-based live streaming (IPTV) and VoD application that has been released for the first time in January 2005. According to the PPLive web site [PPLive 2008], as of May 2006, the PPLive network provided more than 200 channels with a daily average of 400,000 aggregated users, and most of its channels had several thousands of users at their peaks [Vu et al, 2006]. With typically over 100,000 simultaneous users PPLive is the most popular IPTV application today. The system is increasing in popularity, especially in China and Asia [Hei et al, 2006]. The number of subscribers is predicted to increase to 36.9 millions by 2009 [Vu et al, 2006]. The current client version (PPLive 1.9.15) runs on Windows operating systems.

"The bit rates of video programs mainly range from 250 Kbps to 400 Kbps with a few channels as high as 800 Kbps. The PPLive network does not own video content. The video content is mostly feeds from TV channels in Mandarin. The channels are encoded in two video formats: WMV and

RMVB. The encoded video content is divided into chunks and distributed to users through the PPLive P2P network" [Hei et al, 2006].

The PPLive web site provides limited information about its video content distribution mechanism. Although PPLive is paving the way for an important new class of bandwidth intensive applications, as it employs proprietary signaling and video delivery protocols, details about its performance, streaming workload and overlay characteristics are still largely unknown. In [Hei et al, 2006], the authors undertake a preliminary measurement study of PPLive, reporting results from passive packet sniffing of residential and campus peers for streaming performance, workload characteristics, and overlay properties. In [Vu et al, 2006], the authors studied the overlay characteristics of the proprietary PPLive protocol by undertaking a crawler-based investigation of the system.

PPLive streams live TV and video data through overlays of cooperative peers, in which peers download and redistribute live television content from and to other peers. The PPLive system has multiple channels, each of which forms its own overlay. Each channel streams either live audio-video feeds, or movies according to a preset schedule. A user can join at most one channel [Vu et al, 2006].

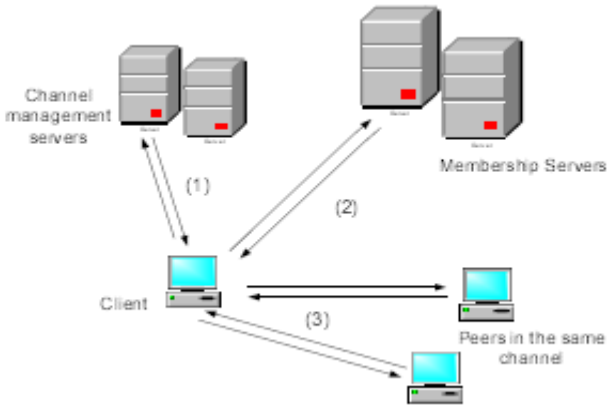

Figure 4.3 PPLive membership and partnership protocols [Vu et al, 2006]

The PPLive software implements a gossip-based protocol for peer management and channel discovery, see Figure 4.3. When an end-user joins the PPLive network, it sends out a query message to the PPLive channel server to obtain an updated channel list. Before a peer actually starts to watch a channel, it does not exchange data with other PPLive peers. After a peer selects a channel to watch, it sends out multiple query messages to some root servers to retrieve an online peer list for this channel. Upon receiving this list, the PPLive client sends out probes to peers on the list to find active peers for the channel. Some active peers may also return their own peer lists, helping the initial peer to find more peers. [Hei et al, 2006].

Although PPLive is not open-source, a little of its internal design decisions are known. Each PPLive node executes two protocols, one for registration and harvesting of partners, and the second for P2P video distribution. A PPLive node maintains two kinds of partners: "candidates" and "real" partners. Partners of the latter type are used for exchanging video streams, while the former is used to replace real partners that have become unresponsive [Vu et al, 2006]. During a peer's lifetime, it constantly changes its upload and download neighbors [Hei et al, 2006].

PPLive divides video streams into chunks and distributes them via overlays of cooperative peers. The PPLive system consists of multiple overlays, with one overlay per channel. Each channel streams either live content or a repeating prefixed program, and the feed from the channel originates

from a server. The structure of the PPLive overlays is close to that of random graphs [Vu et al, 2006].

The major software component of PPLive is its TV engine. It takes care of downloading video chunks from the PPLive network and streaming the downloaded video to a local media player. The streaming process in the PPLive traverses two buffers in local memory: the PPLive TV engine buffer and the media player buffer. This double buffering mechanism is designed to pre-cache media content to combat download rate variations from the PPLive network; and to ensure efficient content distribution between peers. The peer client contacts multiple active peers to download media content of the channel it selected. Received video chunks are reassembled in order and buffered in the queue of the PPLive TV engine, forming a local streaming file in memory [Hei et al, 2006].

When PPLive starts, the PPLive TV engine downloads media content from peers aggressively to minimize the playback start-up delay. When the streaming file length crosses a predefined threshold, the PPLive TV engine launches a media player, which downloads video content from the local HTTP streaming server. After the buffer of the media player fills up to the required level, the actual video playback starts. When the media player has received enough content and has started to play the media, the streaming process gradually stabilizes. The PPLive TV engine streams data to the media player at the media playback rate [Hei et al, 2006].

Due to the distributed nature of PPLive streaming, it is possible that a PPLive peer downloads duplicate media content from multiple peers. The transmission of redundant video chunks wastes network bandwidth. Hence, it is interesting to have a measurement of the redundancy of the PPLive video traffic. In [Hei et al, 2006], the authors computed the redundant traffic by the difference between the total received video traffic and the estimated media segment size, the redundancy ratio being define as the ratio between the redundant traffic and the estimated media segment size. They observed that the traffic redundancy in PPLive is limited. This is partially due to the long buffer time period so that PPLive peers have enough time to locate peers in the same streaming channel and exchange content availability information between themselves.

The PPLive TV engine is responsible for downloading video chunks from the PPLive network and streaming the downloaded video to a local media player. The cached contents can be uploaded to other peers that are watching the same channel. While the peer client contacts multiple active peers to download media content of the channel, at the same time, this peer client may also upload cached video chunks to multiple peers [Hei et al, 2006].

The proprietary PPLive system is rumored to use the idea of inter-overlay optimizations. As a result, a client machine may appear as a participant in multiple overlays, including ones that the user is not subscribed to. A user may join any channel via her/his client machine, but the client machine could also be chosen by the protocol to act as a relay for feeds from channels other than the subscribed ones [Vu et al, 2006].

There is distinct peer connectivity behavior for campus peers and for residential peers. A campus peer has many more active video peer neighbors than a residential peer, and utilizes its high-bandwidth connectivity, maintaining a steady number of active TCP connections for video traffic exchange [Hei et al, 2006].

Content popularity has a significant impact on the number of active peer neighbors for a residential peer. A residential peer with a less popular channel seems to have difficulty in finding enough peers for streaming the media. This reduction in video neighbors impacts the download rate significantly. The PPLive client detects such a rate reduction and starts to search for new peers for additional video download. When new peers are found and fresh streaming flows are established, the video download rate recovers quickly as a result. [Hei et al, 2006].

The PPLive system seems to take the geographic distribution of peers into account. Using the first prefix of the peer's IP addresses to estimate their geographic distribution, the authors in [Hei et al, 2006] observed that a large number of peers located in Asia contributed the majority of the download traffic for the peers located in New York they traced in their study, probably because many PPLive users are located in Asia. But, on the other hand, the majority of the video traffic uploaded by the same traced peers, was to peers in North America.

Overall, PPLive exhibits reasonably good start-up user experiences. When PPLive first starts, it requires some time to search for peers and then tries to download data from active peers. The delay from when one channel is selected until the streaming player pops up is in general 10 to 15 seconds and, the delay from when the player pops up until the playback actually starts is around 10 to 15 seconds; therefore, the total start-up delay is around 20 to 30 seconds. Nevertheless, some less popular channels could have a total start-up delay of up to 2 minutes [Hei et al, 2006].

4.2.4. VMesh

VMesh is a distributed P2P VoD streaming scheme proposed in [Yiu et al, 2007], and designed to efficiently support random seeking functionality. It builds an overlay mesh upon peers, with a pull-based approach, to support random forward/backward seek, pause and restart during playback.

Most of the existing works on P2P-based media streaming systems have made an implicit assumption that a user who joins a streaming session would play the media from the beginning to the end. On the contrary, VMesh has been based on observations showing that most users performed random seeking frequently, and that the jump distances are usually small, what is found reasonable because users would usually skip boring scenes by jumping a bit forward or review some exciting scenes by jumping a bit backward [Yiu et al, 2007].

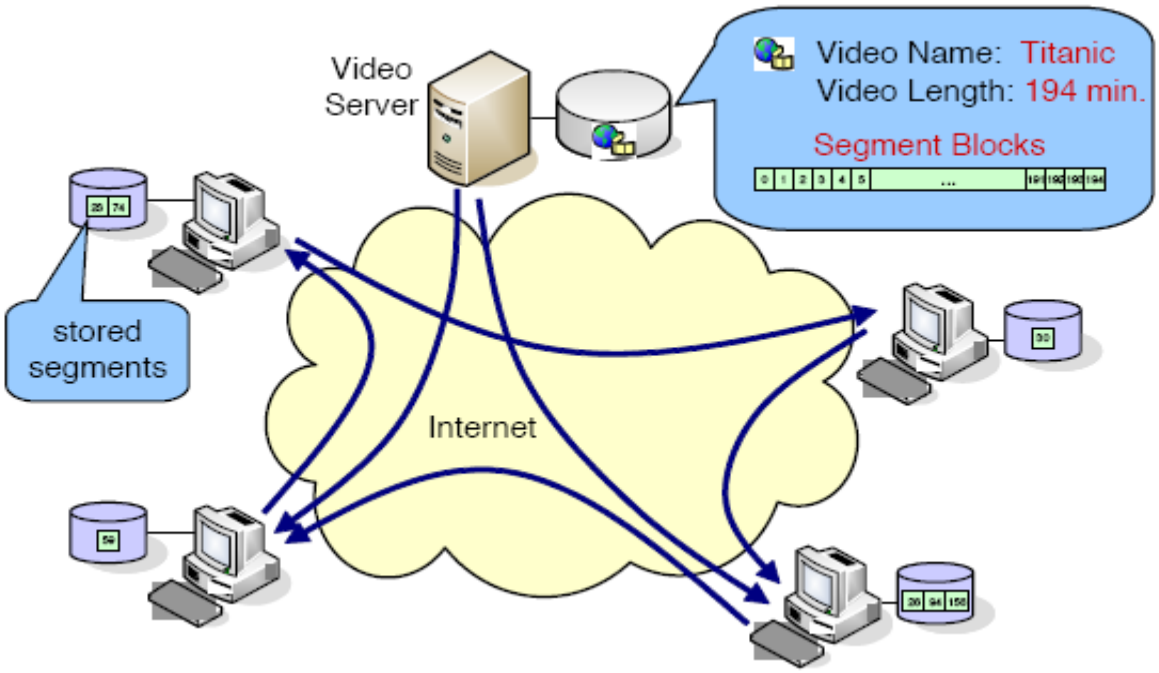

Figure 4.4 Overview of VMesh architecture with static local storage [Yiu et al, 2007]

In the proposed scheme, a video server stores the videos for user access. Each video is divided into N segments, each of them being identifiable by its video ID and segment ID. VMesh utilizes the large aggregate storage capacity of peers to amplify the supply of video segments to achieve user scalability. Depending on the capacity of its local storage, each peer stores a number of segments randomly chosen from the N segments of the video. These peers are referred as storage peers. In

this way, there are multiple copies of each video segment in the network. Figure 4.4 gives an overview of the VMesh architecture with a video file divided into $N = 194$ segments and each peer storing some segments for serving others. When a client wants to play a segment, it first looks for the supplying peers of that segment in the P2P network, then sends requests to those peers for the service. Those supplying peers with enough outgoing bandwidth would serve the requesting peer. If there is no supplying peer, the requesting peer requests the media server for the target segment as the last resort [Yiu et al, 2007].

In the traditional "cache-and-relay" paradigm, a VoD client commonly relies on the content that resides in its parents' buffers. If a parent jumps to another play point in the video, it starts to receive media data which is of no interest for its children, and those need to search for a new parent. In contrast, VMesh uses static local storage instead of sliding window buffering to help handle user interactivity efficiently and to reduce the complexity. The segments stored by a peer in VMesh would not be changed by any user interactive actions. The advantage of this scheme is that interactive action (e.g. random seeking) of a peer does not stop its children from continuing to receive its stored data. The peers, on the other hand, will connect to new parents for each segment. It also means that, during normal playback, the time for a peer to swap parents is predictable. Therefore, the peer can start caching the next segment when it nearly finishes playing out the current segment [Yiu et al, 2007].

The size of the segments is not fixed. Having very small segments would require each peer to swap parents very often, which would possibly introduce more control overhead. However, each peer has only a limited storage space for caching video data, and one cannot assume that a low-profile set-top box (STB) is capable of storing a whole movie in its local storage. The size should be chosen properly as a trade off between the requirement on the peers' local storage size and the messaging overhead caused by swapping parents. In [Yiu et al, 2007], the authors just set the segment size to be 5 minutes long of a video stream with bit rate of 1 Mbps, which is around 36 megabytes, considering it is reasonable for both set-top boxes and personal computers to have such a storage space.

VMesh utilizes a DHT to locate the video segments. A new arriving peer uses the DHT built among existing peers to bootstrap a new video streaming session, searches for its first segment of interest in the DHT network, and starts playing the video when the requested data arrives and fills up its buffer. Using its residual bandwidth, it randomly downloads and stores some video segments in its local storage. These segments would then be used to stream to another peer of interest. After a video segment is completely downloaded, the client registers it in the DHT, to allow other peers to locate it [Yiu et al, 2007].

It is possible for a client to continuously search for the next required segment using the DHT network. However, a client who is playing a particular segment is likely to play the next segment too, because the access probabilities of two adjacent segments are not independent but highly correlated. Therefore, to further reduce the latency as well as the routing message overhead introduced by the DHT search process, VMesh additionally builds an overlay mesh among the peers, which links up the contiguous media segments with bi-directional pointers. Each storage peer keeps a list of the storage peers storing the previous and next segments, so its children can quickly find these peers without having to search over the DHT network. Furthermore, a peer also keeps a list of pointers to some peers which are storing the same segment for load balancing purposes, so when a storage peer is overloaded, it can redirect some of its children to other storage peers which are also able to serve them [Yiu et al, 2007].

In case of random seeking a favorite scene in a movie, a user jumps back and forth in the video. Short-distance jumps, depending on the segments size, can be satisfied by either the next- or

previous-segment-list from its current parents. In case of longer distance jumps, the client searches for new parents using the DHT [Yiu et al, 2007].

Keeping all the pointers in the lists up-to-date by frequent updates would cause a kind of messaging overhead that can be eliminated by making children responsible for checking the validity of the pointers sent by their parents. If the percentage of invalid or failed pointers reaches a certain threshold, the child reports the situation to its parent, which shall then update the pointers in its lists by using the DHT [Yiu et al, 2007].

In order to provide failure-tolerant streaming service, a client connects to multiple parents who have stored the segment of interest so that they stream the video in parallel and collaboratively. VMesh consider simple round-robin scheduling for the client to decide, among the multiple possible parents, which blocks to be delivered by each parent [Yiu et al, 2007].

The mutual network distance between parent and children is a crucial factor for efficient streaming. If the parent-child relationships are casually formed, much network resources would be wasted due to inefficient routing and increase in link stress. To address this problem, VMesh takes the network locations of the peers into consideration while locating supplying peers for a client [Yiu et al, 2007].

In order to search for parents with close network locations, we put this locality information into the DHT search keys of the segments registered by the peers using a Space Filling Curve (SFC). Each peer registers its own key for its stored video segments in the DHT. At the same time, it searches for its parents using DHT search keys constructed by segment ID and its own mapped coordinates. Most DHTs can be modified to reply queries with multiple peers whose keys are numerically closest to the search key. Since the peer's own mapped coordinates are used to construct the search key, multiple parents closest to the requesting peer are returned. The peer can then connect to them and request for the segment [Yiu et al, 2007].

Additionally, due to the design of the DHT search key, the lists of bi-directional pointers to contiguous media segments should contain the qualified parents whose locations are close to the client. Therefore, children of a peer are likely redirected to close parents during playback or jumping [Yiu et al, 2007].

There are a number of approaches to extend VMesh for supporting fast-forward operation. One may consider performing fast-forward operation solely by playing out the video stream at a faster speed by the client peer. But this approach would definitely increase the download bandwidth required and costs 100% overhead for the transmission of skipped frames. Therefore, two approaches have been proposed to support FF in VMesh:

- Encode-on-demand at peers: when a peer performs a FF operation, it requests its parents to encode and deliver a video stream of the segment at a faster frame rate dynamically. This approach requires not only processing power from the peers for dynamic encoding, but also the synchronization of all encoding parameters. This would also complicate the client software implementation, and moreover, the client peer needs to swap its parents more frequently while it performs FF operation.
- Distribution of pre-encoded frame-skipped version: the source provides encoded streams of the original version as well as the frame-skipped versions for various speeds. The frame-skipped versions are then distributed to the peers in the same way as the original, the speed of a version being embedded into the media information part of the DHT key. This approach requires to pre-encode all versions, and the peers to store segments of versions other than the original, what requires more global storage capacity [Yiu et al, 2007].

Given a streaming media, the popularity of the segments are different if user interactivity like jumping is allowed, what makes the access rate of the segments non-uniform. Therefore, if storage

nodes uniformly pick random segments to store, some nodes would have heavier load than others since some segments are accessed more frequently. Intuitively, the load-balancing in the system can be improved if the supply of each segment matches the demand of that segment [Yiu et al, 2007].

So in [Yiu et al, 2007], the authors also model the popularities of segments by considering user interactivity. They propose a mechanism to estimate the segment popularities in a distributed manner, and describe a distributed algorithm to determine which segments to be cached by the storage peers and how to perform cache replacement to adapt the supply of segments which meets the changing demand.

The performance of VMesh has been evaluated in [Yiu et al, 2007] using packet-level event-driven simulation, with three variants of the system:

1. the first one in which peers choose parents randomly regardless of their network locations and random selection is applied for segment storage;
2. the second one applies locality-aware segment location;
3. the third and last one applies both locality-aware segment location and popularity-based segment storage schemes.

As the average user population increases exponentially, the server stress for VMesh systems increases very slowly. The system appears to be highly scalable for a large user population, and maintains a relatively constant server stress at a low level. The use of locality-aware segment location reduces server stress because clients search for close peers to become one's parents and those close peers can provide efficient streaming in terms of error rate and delay, while segments requested from distant parents may experience higher packet loss rates and higher delays. The additional use of popularity-based segment storage can further reduce the server stress when the user population is large enough for the scheme to distribute enough segments among peers and match their supplies and demands, what relieves the server from handling requests for segments whose demands exceed their supplies. Furthermore, as VMesh peers store segments statically and a peer’s random seeking does not stop its children from continuing to receive data from its storage, it can keep server stress at a low level under user interactivity [Yiu et al, 2007].

All three VMesh schemes can achieve high playback continuity under dynamic peer join/leave in a lossy network. Since VMesh employs multiple parents, if one of the parents leaves the system ungracefully, the child can request the remaining parents to share the load of the departed parent temporarily. Additionally, locality-aware segment location achieves better continuity because searching for close parents leads to less packet losses. Close parents are also more likely to retransmit lost packets quickly enough before the playback deadline. On the other hand, as popularity-based segment storage only helps in load-balancing among storage peers, it does not help improving the playback continuity [Yiu et al, 2007].

All schemes suffer from segment location and buffering latencies. The use of a DHT for searching parents makes segment location latency at startup more or less the same regardless of the user population size. However, thanks to the overlay links to the peers who store the next and the previous segments, segment location latency at random seeking can be lower than at startup, because a peer could follow the mesh links to locate the targeted segments. Regarding buffering latency, as there are more users in the system, it is more likely for users using both locality-aware segment location and popularity-based segment storage to find close parents and hence fill up their buffers more quickly [Yiu et al, 2007].

Finally, control messages for operations like joining, data scheduling, and mesh construction cause control traffic overhead. When using popularity-based segment storage, distributed consensus also requires exchange of messages. VMesh consumes very low control overhead and is scalable when

the user population increases, ranging from 100 to 400 bytes per second. For a 1 Mbps video stream, the control traffic is only 0.1% to 0.3%. Distributed consensus algorithm in the popularity-based segment storage scheme adds extra control messages, and hence increases the control traffic overhead by around 40 bytes per second [Yiu et al, 2007].

	Streaming type	*System architecture*	*Overlay topology*	*Data exchange design*	*Data coding*	*Peer choice*	*Content repository*	*upload source*	*Heterogeneity*	*Locality*	*Incentives*	*Performance measurement*
CoolStreaming	Live	P2P	Mesh	Hybrid Push-Pull	Multi-streans	Gossip	Centralized	Buffer	Select highest bandwidth peer	*not used*	*not used*	Simulation + Implementation
GridMedia	Live	P2P	Mesh	Hybrid Push-Pull	Multi-streans	Randomly	Centralized	Buffer	capacity supply ratio	*not used*	*not used*	Simulation
PPLive	Live	P2P	Mesh	*unknown*	Multi-streans	Gossip	Centralized	Buffer + other channels ?	*unknown*	*unknown*	*unknown*	Full implementation
PRIME	Live	P2P	Tree +Mesh	Pull	Multi-streans (MDC)	Randomly	Centralized	Buffer	taken into account	*not used*	*not used*	Simulation
PULSE	Live	P2P	Mesh	Pull	Multiple sources	Gossip	*unknown*	Buffer	supports bandwidth heterogeneity	taken into account	tit-for-tat + additional excess	Simulation + prototype
R2	Live	P2P	Mesh	Push	Random network coding	Gossip	Centralized	Buffer	*not used*	*not used*	*not used*	Prototype implementation
BiToS	VoD	P2P	Mesh	Pull	Multiple sources	Tracker	Centralized	Received Pieces Set	*not used*	*not used*	Tit-for-tat	Simulation
BulletMedia	VoD	P2P	Mesh	Pull	Multiple sources	uniformly at random	DHT	Proactive caching	maximize bandwidth utilization	*not used*	*not used*	Prototype implementation
DirectStream	VoD	P2P	Tree	Push	Single-stream	distance-bandwidth ratio	Distributed	Buffer	distance-bandwidth ratio	proximity information	*not used*	Simulation
HON	VoD	P2P	Tree +Mesh	Pull	Multi-streans	Gossip	Centralized	Buffer	available bandwidth	*not used*	*not used*	Simulation
Joost	VoD	Hybrid Server/P2P	Mesh	*unknown*	Multi-streans	*unknown*	Centralized	"Anthill" cache	taken into account	geographical distance	*unknown*	Implementation
P2VoD	VoD	P2P	Tree	Push	Single-stream	Round robin	Centralized	Buffer	*not used*	*not used*	*not used*	Simulation
PROP	VoD	Hybrid Proxy/P2P	Mesh	Pull	Single-stream	"maximal available bandwith"	DHT	Currently playing	"maximal available bandwith"	data locality exploited	*not used*	Simulation
Push-to-Peer	VoD	Long-lived peers	Controlled overlay	Pull	Multi-streans	*unknown*	*unknown*	Pushed content	Homogeneous peers	*not used*	*not used*	Modelisation
Vmesh	VoD	P2P	Mesh	Pull	Multi-streans	Round robin	DHT + pointers	Static local storage	"enough upload bandwidth"	taken into account	*not used*	Simulation

Table 4.1 Summary of existing or proposed systems

4.3. Synthesis of the chosen options

Table 4.1 presents a global overview of the chosen options in the shape of a two-entry table with system and selection criteria. It comprises 9 VoD systems and 6 live streaming systems. Most of the listed systems are based on a P2P architecture, and 3 of them only use a hybrid architecture. There are probably two reasons for this:

1. the P2P architecture is largely considered as being the solution to the scalability problem faced when implementing video streaming systems;
2. no or few improvements can be brought to a system based on a CDN architecture, this one being only an extension of the traditional client/server scheme.

The use of swarming protocols is preponderant, with mesh-based overlays and pull-based exchange, very often combined with the use of gossip protocols. That probably comes owing to the fact that swarming appears as a very efficient solution, requiring in addition few centralized management, peers self-arranging to form the overlay and choosing parents to send the appropriate video data. The use of multiple streams and data coding is also largely widespread.

Many systems use a centralized content repository although the server is then a central point of failure and could become a bottleneck. However there is a trend to use distributed data structures within the most recently proposed systems. Regarding the data segments shared by the peers, the buffer remains the upload source for almost all of the systems. In addition, very few systems take factors such as locality or heterogeneity of peers into account, and little use incentives to support collaboration between peers.

5. Analysis

In the previous chapters we presented an outline of the proposed or implemented solutions for video streaming services, mainly for VoD streaming but also for live streaming. In order to complete this work, we now review once again the different technical aspects of video streaming by selecting among the several proposed options those which appear to be the most interesting or the most promising for the future.

The first choice to be done is the system architecture. Some very popular video sharing systems, such as YouTube [YouTube 2008] and Google Video [Google Video 2008], are built on a CDN architecture. Joost [Joost 2008] uses a P2P architecture but the whole system relies on a highly centralized management infrastructure of servers or server clusters. On the other hand, most of the systems the different authors recently proposed, such as BulletMedia [Vratonjic et al, 2007], VMesh [Yiu et al, 2007] and DirectStream [Guo et al, 2008], are based on a P2P architecture. Some, such as PROP [Guo et al, 2006], also proposed a hybrid CDN/P2P architecture.

CDNs are effective infrastructures. They are built and daily used to deliver a huge amount of data over the Internet. They have enough storage capacity for a lot of video data files, making it possible to host a comprehensive library of high-quality full length movies. However, CDN infrastructures are really expensive to deploy and maintain, especially if they were scaled with the appropriate bandwidth capacity to face flash crowd. On the other hand, P2P networks showed their effectiveness in scalability to deliver data to a large amount of users. Their bandwidth capacity is directly proportional to the number of connected users. However, a commercial VoD service could not rely on a P2P overlay only, as some file-sharing networks do. They must comprise a reliable and effective data source seed in order to ensure a high quality of service, even with a limited number of connected users or when delivering less popular content.

So a hybrid solution which combines CDN and P2P overlays seems to be promising in order to address both the scalability and the reliability issues of streaming media delivery in a cost-effective way. In such a solution, the CDN provides dedicated storage and reliable streaming services when peers are not available or not capable of doing so. It fulfills the requests from the first clients until the P2P capacity reaches a sufficient level. The P2P overlay can then serve subsequent requests with higher bandwidth capacity than the CDN itself. When the demand for a particular video file decreases, the system must rely on the CDN again. In order to remain cost-effective, the CDN architecture should be carefully dimensioned and remain fairly limited.

Regarding the data delivery overlay topology, tree-based overlays and mesh overlays have been reported as the two main types of architectures. A tree-based overlay is a fairly natural way to arrange collaborating peers, as an extent of the topology probably applied in several CDN infrastructures, and easily matching with the idea of video data streaming from the source seed to the different peers. It appears also particularly suited to live streaming where clients just want to watch at a same TV program. However, a big issue in the tree-based approach is its lack of ability to cope with churn, as each node can be a single point of failure for a whole subtree. This problem is probably worsened because the first joined nodes, which are consequently placed at a higher level closer to the source, are also most likely to leave first, breaking the streaming flow for sub-trees containing a lot of nodes. Another issue occurs on symmetric trees that do not take the bandwidth heterogeneity of peers into account. Furthermore trees are not naturally suited for a VoD service where users can watch anything at any time. On the other hand, in a mesh-based approach, nodes are self-arranged and randomly connected to form the overlay. None of the nodes has a particular position in the overlay, so mesh-based networks appear to be resilient to peers churn. They have proved to be very efficient delivering appropriate data in a timely manner, also for live streaming service [Wang and Li 2007]. The dynamic formation of the delivery mesh enables to effectively

utilize the available bandwidth. So the mesh-based approach seems to be more promising for future developments.

Choosing between push-based and pull-based data exchange designs in a P2P streaming system is a tough issue. Both have proved to be efficient for delivering video data in VoD [Guo et al, 2008] [Yiu et al, 2007] and live streaming services [Wang and Li 2007] [Zhang et al, 2007]. A drawback of a push-based protocol is that two different peers may push the same data to one same peer, or in a VoD system that a peer may be pushed data it is not interested in anymore because it has jumped to another playback location, both examples leading to a waste in uploading bandwidth. So data push scheduling is not a trivial issue. On the other hand, a drawback of a pull-based protocol is that frequent storage map exchanges and pull requests incur more signaling overhead and might introduce additional delays in data retrieval. To solve this particular issue, some systems based upon a hybrid push-pull scheme have also been proposed [Li et al, 2007] [Zhang et al, 2007]. However, though a push-based scheme may be really efficient in a live streaming system, where new created video segments should be forwarded to peers as soon as possible to minimize the playback delay, it seems more promising to use a pull-based scheme in a VoD streaming service designed to support frequent VCR-like interactions so that storage map exchanges must occur anyway each time the playing point of a node or one of its parent seeds changes.

The combined use of data coding and streaming from multiple sources has been an important improvement in P2P video streaming systems. The latest proposed tree-based systems also use data coding combined with multiple-tree overlays or forests [Magharei and Rejaie 2007b]. The use of multiple streams makes the system more resilient to peers churn. While some systems use Multiple Description Coding (MDC) [Padmanabhan et al, 2002] [Magharei and Rejaie 2007a], some other coding schemes have also been proposed [Chi and Zhang 2006] [Liu et al, 2007]. An important point about downloading from multiple sources is that a client must be able to decode the data segment even if s/he only received one of the substreams for this segment.

Another issue in P2P overlays is the choice of the peers from which to download the data. In a tree-based system, this is a trivial issue since the data flow is closely linked to the tree structure and a node receives the data from the node(s) located at the upper level in the tree(s). In a mesh overlay with a pull-based design, the client must select its providing peers among a list of potential seeds given by the system. This selection is usually done using gossip protocols, which have proved to be efficient, according to the availability of the peers and their willingness to serve data. But other parameters can also be taken into account such as the distance and bandwidth [Guo et al, 2008] or the network topology [Hefeeda et al, 2003]. VMesh takes the network location of peers into consideration while locating supplying peers for a client [Yiu et al, 2007]: the content repository (a DHT) includes the network coordinates of the peer with the video information and the segment ID in its key, so that the closest peers are automatically selected. This kind of information appears to be important when designing a peer choice protocol since the simulation results show that VMesh with the locality-aware peer location outperforms in both startup and seeking latencies the system built without this feature. A related issue is the choice of the segment to be downloaded: the rarest first policy is often used because a less represented segment is more likely to become unavailable. Note that in a system with a push-based design, the problem of peer choice is opposite because the transmission of data between two peers is initiated by the sender, but the problem of the choice of the segment to be sent also exists. According to [Bonald et al, 2008], in such a scheme the random peer, latest useful chunk algorithm is the optimal one.

In order to provide a list of potential seeds for a particular segment, as mentioned in the previous paragraph, a P2P streaming system using a pull-based scheme must have a content repository with a comprehensive catalog of all the peers hosting each particular segment, and even each substream of these ones for systems using data coding. In many systems, like P2VoD [Do et al, 2004], BiToS [Vlavianos et al, 2006] and Joost [Lei et al, 2007], this repository is a centralized one located at the

server side. This is a rather reliable solution except that all the transactions regarding data location must go through the server which could become a bottleneck. That is why more recent systems, like BulletMedia [Vratonjic et al, 2007], VMesh [Yiu et al, 2007] and DirectStream [Guo et al, 2008], prefer using a distributed solution, which is often a DHT. A DHT works like a traditional hash table but the table entries are distributed among the peers in the network. So as the number of peers hosting the table grows with the number of entries in the table, the transactions are fairly distributed among the peers and the number of transactions per peer remains stable. Such a distributed solution appears to be better than a centralized one. Additional solutions, such as bi-directional pointers [Yiu et al, 2007] and Dynamic Skip List [Wang and Liu 2008], have also been proposed. Those solutions are very interesting because they allow the clients nodes to find peers hosting segments related to the current one without having to query the system repository. Nevertheless, it seems those solutions must be combined with another repository structure.

The media content that the peers are serving is usually the video segments located in their playing buffer, so the data they are able to serve change frequently, depending on the size of their buffer. This also means that the content repository must be more frequently updated to reflect those changes, and that a node shall have to search for a new providing peer if a parent jumps to another playing point in the video. In order to address this issue, VMesh uses a static local storage as upload source [Yiu et al, 2007]. The peers use their spare bandwidth to download some segments from the video file they are currently playing but which are not immediately interesting for them. The downloaded segments can then remain available during their whole video session even if they frequently jump forward and backward. Such a solution appears to be very promising, especially because it also gives the system the opportunity to balance the distribution of the segments, based for example on their popularity as proposed in VMesh.

Few systems seem to pay attention to peers heterogeneity. However bandwidth capacities of nodes located behind residential broadband connections can strongly differ from those of nodes located on high bandwidth corporate or university LANs. High-bandwidth nodes are very useful for P2P networks because they provide most of the upload capacity of the overlay, but this capacity should first be used to provide the best download rate to other high-bandwidth nodes, and the system must avoid wasting this bandwidth sending data to low capacity nodes whose download capacity could become the bottleneck. Nevertheless, peer heterogeneity is indirectly taken into account by several systems because their peer choice protocol is based on the availability of peers and this one is depending on their bandwidth.

Only some systems, like BASS [Dana et al, 2005], BiToS [Vlavianos et al, 2006] and PULSE [Pianese et al, 2007], use incentives to enhance collaboration between peers, and to prevent abuse of selfish peers which would use the collaborative network without contributing back to it, though the success of BitTorrent [Cohen 2003] seems to be related to the use of Tit-for-Tat.

To conclude this chapter, it appears interesting to point out that among all the surveyed VoD streaming systems, two of them only, BulletMedia [Vratonjic et al, 2007] and VMesh [Yiu et al, 2007], have been explicitly designed to support VCR-like control operations such as fast forward, fast search, reverse search and rewind. In the other systems, either some of these functions are simulated by leaving and joining again at another playing point, or they are not supported at all. That seems well to be a sign that many scientific and technological challenges are still to be faced in this research field. There is still so much work to do in order to provide a service enabling at least to view a movie in DVD quality, from the very beginning to the final end, without experiencing continuity issue or any other disturbing event, that VCR interactions are still very often viewed as only a nice-to-have.

6. Conclusion

In this work, a survey and a synthesis of the currently existing or proposed systems and solutions for providing VoD and live streaming services over the Internet has been presented. This survey was based upon a subset of systems being representative of the architectural options, and for which the necessary information to carry out an analysis was available.

A list of selection criteria allowing to classify the video streaming systems into different categories was defined. Those selection criteria are the streaming type, the system architecture, the delivery overlay topology, the data exchange design, the use of data coding techniques, the peer choice and data scheduling protocols, the structure used for the content repository, the source of data used to upload content to peers, the peer locality and heterogeneity factors, and the incentives.

A detailed analysis of the CoolStreaming, Joost, PPLive and VMesh systems was presented. Those four systems have been chosen to keep a balance between VoD systems and live streaming systems, and a representativeness of the different overlay architectures and the currently implemented systems.

Finally, an overview of the proposed solutions was given in a brief description, based on the defined selection criteria, of the selected representative systems, and the different technical aspects of video streaming were reviewed and analyzed, with an attempt to evaluate the options which appear to be the most interesting or the most promising for future experiments.

Based on this survey, it appears that, although some recent improvements, there is still work much to do in order to provide a VoD service where movies are delivered to distributed users with low delay and free interactivity.

References

Akamai Technologies, Inc., Akamai: The Leader in Web Application Acceleration and Performance Management, Streaming Media Services and Content Delivery [online], http://www.akamai.com, last visited : August 12, 2008

Annapureddy S., Gkantsidis C., and Rodriguez P., "Providing Video-on-Demand using Peer-to-Peer networks.", in *Proceedings of Internet Protocol TeleVision (IPTV) workshop, 15th International World Wide Web Conference 2006 (WWW '06)*, Edinburgh, Scotland, United Kingdom, 23 May 2006 .

Annapureddy S., Guha S., Gkantsidis Ch., Gunawardena D. and Rodriguez P., "Is High-Quality VoD Feasible using P2P Swarming?", in *Proceedings of the 16th international conference on World Wide Web (WWW '07)*, Banff, Alberta, Canada, pages 903-912, 8-12 May 2007.

Babaoglu O., Meling H. and Montresor A., "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems," in *Proceedings of 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

BitTorrent, Inc., BitTorrent [online], http://www.bittorrent.com, last visited : August 17, 2008.

Bonald T., Massoulié L., Mathieu F., Perino D., and Twigg A., "Epidemic Live Streaming: Optimal Performance Trade-Offs", in *ACM SIGMETRICS'08*, Annapolis, Maryland, USA, June 2–6, 2008.

Carra D., Lo Cigno R. and Biersack E.W., "Graph Based Analysis of Mesh Overlay Streaming Systems" , in *IEEE Journal on Selected Areas in Communications*, Volume 25, Issue 9, Pages 1667-1677 , December 2007.

Chen Y., Qiu L., Chen W., Nguyen L., and Katz R. H., "Efficient and Adaptive Web Replication using Content Clustering", in *IEEE Journal on Selected Areas in Communications*, Volume 21, Issue 6, pages 979-994, August 2003.

Cheng B., Liu X., Zhang Z., and Jin H., "A Measurement Study of a Peer-to-Peer Video-on-Demand System", in *Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS '07)*, February 2007.

Chi H.-Ch., Zhang Q., "Deadline-aware network coding for video on demand service over P2P networks", in *Journal of Zhejiang University SCIENCE A,* Volume 7, Number 5, pages 755-763, 2006.

Cohen B., "Incentives Build Robustness in BitTorrent.", in *1st Workshop on the Economics of Peer-2-Peer Systems*, Berkley, CA, June 5-6 2003.

CoolStreaming Platform IPTv // P2P Tv // Net Tv [online], http://www.coolstreaming.us, last visited: August 17, 2008.

Costa M., Castro M., Rowstron A. and Key P., "PIC: Practical Internet coordinates for distance estimation", in *International Conference on Distributed Systems*, pages 178-187, March 2004.

Dabek F., Cox R., Kaashoek F. and Morris R., "Vivaldi: A Decentralized Network Coordinate System", in *SIGCOMM*, pages 15-26, August 2004.

Dana Ch., Li D., Harrison D., and Chuah Ch.-N., "BASS: BitTorrent Assisted Streaming System for Video-on-Demand", in *Proceedings of IEEE 7th Workshop on Multimedia Signal Processing, 2005*, Shanghai, pages 1-4, October 2005.

Do T.T., Hua K.A., and Tantaoui M.A., “P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment”, in *Proceedings of IEEE International Conference on Communications, 2004*, Volume 3, pages 1467- 1472, 20-24 June 2004.

Esteve M., Fortino G., Mastroianni C., Palau C. E., and Russo W., “CDN-supported Collaborative Media Streaming Control”, in *IEEE MultiMedia*, volume 14, number 2, pages 60-71, April-June 2007.

Google Video [online], http://video.google.com, last visited: August 15, 2008.

Grothoff Ch., “An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks”, in *Wirtschaftsinformatik*, June 2003.

Guo L., Chen S., and Zhang X., “Design and evaluation of a scalable and reliable P2P assisted proxy for on-demand streaming media delivery”, in *IEEE Transactions on Knowledge and Data Engineering*, Volume 18, Number 5 , pages 669-682, May 2006.

Guo Y., Suh K., Kurose J., and Towsley D., “P2Cast: Peer-to-peer Patching Scheme for VoD Service,” in *Proceedings of the twelfth international conference on World Wide Web (WWW’03)*, Budapest, Hungary, pages 301-309, May 2003.

Guo Y., Suh K., Kurose J., and Towsley D., "DirectStream: A directory-based peer-to-peer video streaming service", in *Journal of Computer communications (COMCOM),* Elsevier, Volume 31, Number 3, pages 520-536, 25 February 2008.

He Y. and Liu Y., "Supporting VCR in Peer-to-Peer Video-On-Demand", in *Proceedings of the IEEE International Conference on Network Protocols (ICNP 2007)*, Beijing, China, pages 328-329, 16-19 October 2007.

Hefeeda M., Habib A., Botev B., Xu D., and Bhargava B., “PROMISE: peer-to-peer media streaming using CollectCast”, in *Proceedings of ACM Multimedia 2003 (MM’03)*, Berkeley, CA, November 2003.

Hei X., Liang C., Liang J., Liu Y. and Ross K. W., “Insights into PPLive: A Measurement Study of a Large-scale P2P IPTV System”, in *Proceedings of Internet Protocol TeleVision (IPTV) workshop, 15th International World Wide Web Conference 2006 (WWW ’06)*, Edinburgh, Scotland, United Kingdom, 23 May 2006.

Hu A., “Video-on-demand Broadcasting Protocols: A Comprehensive Study”, in *Proceedings of IEEE INFOCOM*, pages 508-517, 2001.

Joost - Free online TV - Comedy, cartoons, sports, music and more [online], http://www.joost.com, last visited: August 17, 2008.

Jurca D., Chakareski J., Wagner J.-P., Frossard P., "Enabling adaptive video streaming in P2P systems", in *IEEE Communications Magazine*, Volume 45, Issue 6, pages 108-114, June 2007.

Kermarrec A.-M. and van Steen M., "Gossiping in distributed systems", in *ACM SIGOPS Operating Systems Review*, Volume 41, Issue 5, pages 2-7, October 2007.

de Launois C., Uhlig S., and Bonaventure O., "A Stable and Distributed Network Coordinate System", Technical report, Universite Catholique de Louvain, December 2004.

Lazar I. and Terrill W., "Exploring Content Delivery Networking", in *IT Professional*, Volume 3, Number 3, pages 47-49, 2001.

Ledlie J., Gardner P., and Seltzer M., "Network Coordinates in the Wild", in *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation*, Cambridge, MA, p**ages 299-311**, 11-13 April 2007.

Lei J., Shi L., and Fu X., “An Experimental Analysis of Joost Peer-to-Peer VoD Service”, Technische Berichte des Instituts für Informatik an der Georg-August-Universität Göttingen , Number IFI-TB-2007-03, 15 October 2007.

Li B., Xie S., Keung G.Y., Liu J., Stoica I., Zhang H., and Zhang X., “An Empirical Study of the Coolstreaming+ System”, in *IEEE Journal on Selected Areas in Communications*, Volume 25, Number 9, pages 1627-1639, December 2007.

Limelight Networks [online], http://www.limelightnetworks.com, last visited: August 31, 2008.

Liu Y., Guo Y. and Liang Ch., "A survey on peer-to-peer video streaming systems", in *Journal of Peer-to-Peer Networking and Applications,* Springer, New York, February 2008.

Liu Z., Shen Y., Panwar S., Ross K. W., and Wang Y., "Efficient Substream Encoding for P2P Video on Demand", in *Packet Video 2007*, pages 143-152, 12-13 November 2007.

Magharei N. and Rejaie R., “PRIME: Peer-to-peer Receiver drIven MEsh-based streaming”, in *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM 2007)*, Anchorage, Alaska, USA, pages 1415-1423, 6-12 May 2007.

Magharei N. and Rejaie R., "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches", in *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM 2007)*, Anchorage, Alaska, pages 1424-1432, 6-12 May 2007.

Mirror Image, powering your online business with digital media delivery [online], http://www.mirror-image.com, last visited: August 12, 2008.

Montresor A., "Intelligent Gossip", in *2nd International Symposium on Intelligent Distributed Computing (IDC 2008)*, Catania, Italy, September 2008.

Ng T. S. E. and Zhang H., "Predicting Internet network distance with coordinates-based approaches", in *INFOCOM*, pages 170-179, June 2002.

Padmanabhan V., Wang H., Chou P., and Sripanidkulchai K., “Distributing streaming media content using cooperative networking”, in *Proceedings of the 12th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV ’02)*, Miami Beach, FL, USA, May 2002.

Pai V. S., Wang L., Park K. S., Pang R., and Peterson L., “The Dark Side of the Web: An Open Proxy's View”, in *Proceedings of the Second Workshop on Hot Topics in Networking (HotNets-II)*, Cambridge, MA, USA, November 2003.

Pallis G. and Vakali A., "Insight and Perspectives for Content Delivery Networks", in *Communications of the ACM*, Volume 49, Number. 1, ACM Press, NY, USA, pages 101-106, January 2006.

Pathan A. M. K. and Buyya R., "A Taxonomy and Survey of Content Delivery Networks", Grid Computing and Distributed Systems (GRIDS), Technical Report, GRIDS-TR-2007-4, The University of Melbourne, Australia, February 2007.

Peng G., "CDN: Content Distribution Network", Technical Report TR-125, Experimental Computer Systems Lab, Department of Computer Science, State University of New York, Stony Brook, NY, 2003.

Pianese F., Perino D., Keller J., and Biersack E.W., “PULSE: An Adaptive, Incentive-Based, Unstructured P2P Live Streaming System ”, in *IEEE Transactions on Multimedia*, Volume 9, Number 8, pages 1645-1660, December 2007.

Pias M., Crowcroft J., Wilbur S., Harris T., and Bhatti S., “Lighthouses for Scalable Distributed Location”, in *IPTPS*, February 2003

Pinho L. B., Ishikawa E., and Amorim C. L., “GloVE: A Distributed Environment for Low Cost Scalable VoD Systems”, in *Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Vitoria, ES, Brazil, pages 117–124, October 2002.

PPLive - The Largest World Wide Internet TV Network [online], http://www.pplive.com/en, last visited: August 17, 2008.

Ross D., “Game Theory”, in *Stanford Encyclopedia of Philosophy*, http://plato.stanford.edu/entries/game-theory/, First published Sat Jan 25, 1997; substantive revision Fri Mar 10, 2006.

Rowstron A., Kermarrec A.-M., Castro, M. and Druschel P., “Scribe: A large-scale and decentralised application-level multicast infrastructure,” in *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.

Shavitt Y. and Tankel T., "Big-bang simulation for embedding network distances in Euclidean space", in *Proceedings of IEEE INFOCOM*, June 2003.

Stoica I., Morris R., Liben-Nowell D., Karger D. R., Kaashoek M. F., Dabek F. and Balakrishnan H., “Chord: A scalable peer-to-peer lookup protocol for internet applications,” in *IEEE/ACM Transactions on Networking*, volume 11, number 1, pages 17–32, February 2003.

Suh K., Diot C., Kurose J., Massoulie L., Neumann C., Towsley D., and Varvello M., “Push-to-Peer Video-on-Demand system: design and evaluation”, *Thomson Technical Report*, Number CR-PRL-2006-11-0001, 29 November 2006.

Thouin F. and Coates M., “Video-on-Demand Networks: Design Approaches and Future Challenges”, in *IEEE Network*, Volume 21, Issue 2, pages 42-48, March-April 2007.

Vakali A. and Pallis G., "Content Delivery Networks: Status and Trends", in *IEEE Internet Computing, IEEE Computer Society*, pages 68-74, November-December 2003.

Vlavianos A., Iliofotou M., and Faloutsos M., “BiToS: Enhancing BitTorrent for supporting streaming applications”, in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006),* Barcelona, Spain, pages 1–6, April 2006.

Vratonjic N., Gupta P., Knezevic N., Kostic D., and Rowstron A., “Enabling DVD-like features in P2P Video-on-demand Systems,” in *ACM SIGCOMM 2007 Data Communications Festival, Peer-to-Peer Streaming and IP-TV Workshop (P2P-TV’07)*, Kyoto, Japan, 31 August 2007.

Vu L., Gupta I., Liang J., and Nahrstedt K., “Mapping the PPLive network: Studying the impacts of media streaming on P2P overlays”, Department of Computer Science, University of Illinois at Urbana-Champaign, Technical Report. UIUCDCS-R-2006-275, August 2006.

Wang M. and Li B., “R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming”, in *IEEE Journal on Selected Areas in Communications*, Volume 25, Number 9, pages 1655-1666, December 2007.

Wang D. and Liu J., “A Dynamic Skip List-Based Overlay for On-Demand Media Streaming with VCR Interactions”, in *IEEE Transactions on Parallel and Distributed Systems*, Volume 19, Number 4, pages 503-514, April 2008

Wu B. and Kshemkalyani A. D., “Objective-optimal algorithms for long-term Web prefetching”, in *IEEE Transactions on Computers*, Volume 55, Number 1, pages 2-17, 2006.

Xu D., Kulkarni S. S., Rosenberg C. and Chai H. K., “Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution”, in *Multimedia System*, volume 11, number 4, pages 383-399, 2006.

YouTube - Broadcast Yourself [online], http://www.youtube.com, last visited: August 27, 2008.

Yiu W.-P.K., Jin X., and Chan S.-H.G., “VMesh: Distributed Segment Storage for Peer-to-Peer Interactive Video Streaming”, in *IEEE Journal on Selected Areas in Communications*, Volume 25, Number 9, pages 1717-1731, December 2007.

Zattoo | TV meets PC [online], http://zattoo.com, last visited: August 15, 2008.

Zhang X., Liu J., Li B., and Yum T.-S. P., “DONet/CoolStreaming: A data-driven overlay network for peer-to-peer live media streaming”, in *Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2005)*, Miami, FL, USA, Volume 3, pages 2102–2111, 13-17 March 2005.

Zhang M., Xiong Y., Zhang Q., Yang S., “On the Optimal Scheduling for Media Streaming in Data-driven Overlay Networks”, in *Proceedings of IEEE GLOBECOM*, San Francisco, CA, USA, November 2006.

Zhang M., Zhang Q., Sun L., and Yang Sh., “Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?”, in *IEEE Journal on Selected Areas in Communications*, Volume 25, Number 9, pages 1678-1694, December 2007.

Zhou M. and Liu J., “A Hybrid Overlay Network for Video-on-Demand”, in *Proceedings of IEEE International Conference on Communications (ICC 2005)*, Seoul, Korea, Volume 2, pages 1309-1313, 16-20 May 2005.