Live Streaming / Video-on-Demand

An Integration

SARA HAGHIGHI FARD

Master of Science Thesis Project
conducted at

SICS
Swedish Institute of Computer Science
Computer Systems Laboratory

Supervisor: Dr. Jim Dowling
dowling@sics.se
Examiner: Prof. Seif Haridi
sharidi@sics.se

TRITA-ICT-EX-2012:105
Abstract

Video delivery over the Internet is becoming increasingly popular and comes in many flavors, such as Live Streaming and Video-on-Demand. In the recent years, many peer to peer solutions for Live Streaming and VoD have been proposed as opposed to the centralized solutions that are not scalable due to the high bandwidth requirements. In all existing solutions, Live Streaming and VoD have traditionally and artificially been considered as separate technical problems. We propose an integrated Live Streaming with VoD system that offers the potential for users to watch live TV with short delays.

In Live Streaming, peers are interested in the content that is being generated live by the streaming source, unlike VoD in which peers are interested in the content that has been generated from the beginning of the streaming. In this manner, Live nodes can contribute to VoD nodes and send them the pieces they have downloaded since their joining time. In this work, we demonstrate that our system, called Live-VoD, brings out the possibility of having both types of nodes in one system, each being served based on their interest. We propose a P2P Live-VoD protocol for overlay construction based on peer’s upload bandwidth that is built on top of the Gradient topology and an innovative algorithm based on the number of pieces peers can contribute to each other. We also propose an innovative stochastic algorithm for data dissemination based on the rareness of the piece and the requesting node’s download position. Our experiments show that Live-VoD is decentralized, scalable and self-organizing. We also show that even when most of the nodes in the system are VoDs, all VoD nodes regardless of their joining time, manage to download the whole movie with no assistance from the streaming source.
Acknowledgements

I would like to give my enormous appreciation to my supervisor, Dr. Jim Dowling.....for the great supervisor he is: the time he spent on sharing his knowledge and experience, his advices, his support and his understanding

I also give my many thanks to my amazing partner.....for being always there for me
Contents

Acknowledgements .................................................. ii

Contents .................................................................. iii

List of Figures ........................................................ iv

List of Tables .......................................................... v

1 Introduction .......................................................... 1
  1.1 Motivation ......................................................... 2
  1.2 Contributions ...................................................... 3
  1.3 Structure of the Report .......................................... 3

2 Background and Related Work .................................. 5
  2.1 Overlay Construction ........................................... 5
     2.1.1 Gossiping ..................................................... 5
  2.2 Data Dissemination ............................................. 7
  2.3 Sepidar .............................................................. 7
  2.4 GVoD ................................................................. 10

3 Live Streaming and Video on Demand ...................... 13
  3.1 Live Streaming ................................................... 13
     3.1.1 Live Streaming peer properties ......................... 13
  3.2 Video-on-Demand .............................................. 15
     3.2.1 Video on Demand peer properties .................... 15

4 Peer Organization: Gradient Topology ..................... 17
  4.1 Gradient Topology ............................................... 18
  4.2 Peer Utility ....................................................... 18
     4.2.1 Dynamic Utility ........................................... 19
     4.2.2 Utility Uniqueness ......................................... 19
     4.2.3 Composite Utility ......................................... 20
  4.3 Neighbor Selection ............................................ 20
     4.3.1 Similar View ............................................... 20
## List of Figures

2.1 Peers organization into different market levels; The similar view and fingers of node $P$.

2.2 Three-layered Gossiping Architecture of GVoD.
5.1 The Gradient topology for GVoD

5.2 Gradient Overlay with Gradient Search.

5.3 Live-VoD System components in Kompics Simulation Mode.

5.4 Live-VoD Peer components in Kompics.

5.5 The Gradient overlay network is bootstrapped using samples from the Random overlay network, the P2P-Live overlay network is built and maintained using samples from the Gradient overlay network and the P2P-VoD is constructed using samples both from the Random overlay and the Live overlay.

5.6 Live-VoD Overlay.

5.7 The effect of $DP(q)$ and availability of the time index $t(A(t))$ on the usefulness of $p$ for $q$. The plot shows the hypothetical contribution of different time indices $t$ in calculating usefulness of $p$ for $q$ with $DP(q) = 10$, for different values of $t$ and $A(t)$.

5.8 Each file is divided into chunks and chunks are divided further into pieces. Both chunks and pieces are timestamped so that they can be re-ordered sequentially once delivered.

6.1 The probability of sending time index $t$ from node $p$ to node $q$, for different values of $t$ and $A(t)$ when $DP(q) = 10$, $\sigma_A = 2$ and $\sigma_{DP} = 2$.

6.2 The average clustering coefficient against the number of nodes.

6.3 Time taken to download the video against the number of nodes.

6.4 Time taken to download the video against the number of nodes.

6.5 Time taken to download the video against the number of nodes.

6.6 Time taken to download the video against the number of nodes.

6.7 Time taken to download the video against the number of nodes.

---

List of Tables

2.1 Data Dissemination Topology in different Systems

5.1 Notations used throughout the report

6.1 Variables in Live-VoD Experiments
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Bandwidth Distribution used in Live-VoD Simulation</td>
<td>44</td>
</tr>
<tr>
<td>6.3</td>
<td>Variables Value in Experiment 1</td>
<td>44</td>
</tr>
<tr>
<td>6.4</td>
<td>Variables Value in Experiment 2</td>
<td>44</td>
</tr>
<tr>
<td>6.5</td>
<td>Variables Value in Experiment 3</td>
<td>45</td>
</tr>
<tr>
<td>6.6</td>
<td>Variables Value in Experiment 4</td>
<td>45</td>
</tr>
<tr>
<td>6.7</td>
<td>Variables Value in Experiment 5</td>
<td>45</td>
</tr>
</tbody>
</table>
# List of Algorithms

1. Update Similar View Using Random View after periodic shuffling . . 32
2. Connect a Live node to another Live node . . . . . . . . . . . . . . . 32
3. Connect a VoD node to another VoD node . . . . . . . . . . . . . . . 33
4. Connect a VoD node to a Live node . . . . . . . . . . . . . . . . . . 34
5. Time Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
Chapter 1

Introduction

Peer to peer systems are systems in which peers share their resources and contribute to one another to fulfill the goal of the system. Each peer is considered both as a client and a server in opposed to Client/Server systems in which there is a centralized server which handles the requests from clients and keeps serving them. This decentralization property along with robustness and self-organization are some major characteristics of peer to peer systems that make them more scalable and hence cheaper compared to Client/Server systems.

The term Media Streaming means sending media content, either video or audio, over a network so that it is received and played by end users who do no longer need to wait for all the media content to be downloaded. In this fashion, they can play it while the media is being delivered. There are two types of media streaming, Live Streaming and Video-on-Demand. In Live Streaming, the media stream is being streamed at a streaming source and it is available at one particular time, whereas in VoD, the media stream can be available at any time at the user’s request. VoD also gives more flexibility to the user by providing a set of VCR functionalities such as Pause, Fast Forward, Rewind and etc. In both Live Streaming and VoD, peers require a minimal download speed to sustain playback continuity and playback quality.

In the recent years, the use of peer to peer systems has been dramatically increased. Today many users and companies utilize the advantages the peer to peer systems have to offer. And now, after the successes peer to peer file sharing systems have experienced, it is the peer to peer media streaming systems’ turn to present their capabilities and draw the attention of millions of users all around the world. With the increasing interests of users to media streaming websites like YouTube that attract many users per day, the need for such systems with less costs becomes more apparent. One major problem with the centralized systems is that they require highly capable servers both in terms of bandwidth and memory which are pretty costly. P2P media streaming systems reduce the bandwidth requirements of the media source by making every client (called peer) in the system to contribute a part of the stream to other clients. Having the peers serve one another when down-
CHAPTER 1. INTRODUCTION

loading the media content reduces the costs dramatically. This is because there is no need for dedicated servers to be up and running constantly and waiting for clients to send them requests.

The goal of Peer-to-peer streaming systems is to reduce server loading while keeping the same streaming performance (High video quality (high playback continuity and low playback latency) and minimum playback delay). However, for achieving this assumably easy goal, several factors have to be taken into account. The fact that peers are heterogenous in terms of bandwidth, memory, session time,... makes it extremely challenging for P2P systems to find an optimal way to connect peers together. In such asymmetric environment, some nodes use the service of other nodes and do not contribute back, which leaves the system imbalanced. Unexpected failure either unintentionally or on purpose is also one big challenge for particularly P2P streaming systems to deal with. These challenges bring up problems such as neighboring or partnership, data delivery and data scheduling to solve.

Considering the same challenges that both Live Streaming and Video-on-Demand streaming systems should struggle with, we thought of integrating the two, hence our system Live-VoD. By having nodes that want to watch a video live and nodes that want to watch the same video from the beginning, the two systems can be united. Live-VoD has addressed this problem by utilizing the same characteristics both P2P Live Streaming and P2P VoD have. We have found the overlapping areas, taken into consideration the differences and created Live-VoD. Our system has focused both on overlay construction and data dissemination and in both areas we have proposed innovative solutions and showed in simulation that Live-VoD integration is plausible. In this report, different characteristics and key properties of P2P-Live Streaming systems and P2P-VoD systems are described. The main challenges for each system and their differences are gone through and finally, a model for integrating the two is suggested. We demonstrate the performance of Live-VoD by evaluating the average clustering coefficient on overlay construction and average downloading time on both the overlay maintenance and data dissemination.

1.1 Motivation

Despite the fact that P2P Live Streaming and P2P VoD systems have so many properties in common and deal with some similar issues, they have always been treated separately and the approaches used to solve each of these problems have not considered the integration of the two. When the possibility of being able to watch a movie that is being streamed from the beginning is needed, P2P Live Streaming and P2P VoD systems should be integrated, offering the capabilities of the two to the users.
1.2 Contributions

In this thesis work, we present the design and implementation of a system that integrates the P2P Live Streaming and P2P VoD systems to the best of our knowledge for the first time, based on, on top of the Gradient Overlay. We have evaluated the performance of our algorithms in different scenarios and shown that the two can work together and one can improve the experience for the other. The main contributions of this work are:

- We propose a model called Live-VoD that shows the possibility of integrating Live Streaming and Video-on-Demand systems using peer-to-peer (P2P) technology
- We show how our model enhances the streaming experience for VoD peers through Simulation
- We consider the heterogeneity characteristics of the peers in system and show how they can be dealt with using Gradient Topology
- We propose an innovative algorithm for overlay construction and maintenance based on peer’s upload bandwidth or peer usefulness depending on peer’s type
- We propose an innovative stochastic algorithm for data dissemination based on rareness and the target’s download position
- We evaluate our model using average clustering coefficient and average downloading time metrics to demonstrate the performance of Live-VoD

1.3 Structure of the Report

The remainder of this report is organized as follows. The next chapter, gives a brief description of the background of P2P Live Streaming and P2P VoD systems. We then introduce the related work for each of these systems. Chapter 3 talks about Live Streaming and Video on Demand properties and gives more insights into these systems. Next, we present Gradient topology and its characteristics in chapter 4. In chapter 5 we present our Live-VoD model, system architecture and finally, we introduce the algorithms we have designed and implemented. In chapter 6 we present our experiments and their results. Finally we conclude our work and discuss future work in chapter 7.
Chapter 2

Background and Related Work

Every P2P media streaming system focuses on two key problems for building and maintaining an overlay network: i) how to discover peers that have the media stream and, ii) how to disseminate the media content to other peers. For solving each of these individually challenging problems, different approaches have been proposed over the years. In the next sections we will introduce these approaches and some of the related works regarding each.

2.1 Overlay Construction

Discovering other peers with the media data that one peer is interested in has been addressed and solved differently in different systems. The existing approaches are i) Centralized, ii) DHT iii) Flooding and iv) Gossiping. In the next section, we will describe gossiping that has been also used in our work, Live-VoD.

2.1.1 Gossiping

Gossiping is a communication/dissemination protocol which are designed to infect the whole network very quickly. Gossiping is highly resilient to churn (the frequent join, leave and failure of nodes) and the overlays constructed using gossiping are easier to maintain. Gossip protocols are very effective in large scale dynamic peer to peer systems. Despite the high overhead and giving no guarantee to find a supplier peer on time, their simplicity and easy implementation, makes them a desirable solution for overlay construction, as gossiping after all gives close to optimal results. We now describe two works that have improved basic gossiping.

Gossip++: Three-Phase Gossip

The three phases of the basic gossip protocol are as follows:

- Propose phase. Periodically, i.e., every gossip period, each node picks a new set of $f$ (fanout) other nodes. This is usually achieved using a peer sampling
protocol. It then proposes, to each of them, the identifiers of the chunks it received since its last propose phase.

- **Request phase.** Upon receipt of a proposal for a set of chunk identifiers, a node determines the subsets of chunks it needs, and requests them from the sender. Clearly, the needed chunks are those that the node has not yet received.

- **Serving phase.** When a proposing node receives a request message, it replies with the corresponding chunks, that is by sending their actual payloads. Nodes only serve chunks that they previously proposed.

Gossip++ has augmented the three-phase gossip protocol with two components: **Codec** and **Claim** in order to provide reliable dissemination of streaming content \[7\]. Codec is an FEC (Forward Error Correction) mechanism, which gives back information to the three-phase gossip protocol to increase the efficiency of the content dissemination. As the chunks are sent with a high probability to all nodes, there is a possibility that some nodes do not receive all chunks. FEC allows these nodes to recover the missing chunks if they cannot actually be requested from any of the other nodes. While Codec has the responsibility of reconstructing the missing chunks, Claim uses retransmission to recover at least $k$ chunks needed by Codec, even when message loss affects more than $c$ chunks.

**Heterogenous Gossip**

HEAP addresses the limitations of standard gossip by preventing congestion at low-capability nodes through the adaptation of each node’s workload. HEAP improves the standard homogeneous gossip with respect to stream quality, bandwidth usage and resilience to churn. When the stream rate is close to the average available bandwidth, the improvement is even more significant. A natural way to further improve the quality of gossiping is to bias the neighbor selection towards rich nodes in the early steps of dissemination. For this reason, we use the gradient topology where nodes are connected to those nodes that have the same or higher utility values(upload bandwidth).

The reliability of gossip dissemination is actually preserved as long as a fanout value of $f = \ln(n) + c$, $n$ being the size of the network, is ensured on average, regardless of the actual fanout distribution across nodes. To achieve this, HEAP exploits a simple gossip-based aggregation protocol which provides an estimate of the average upload capacity $\bar{b}$ of network nodes. The aggregation protocol works by having each node periodically gossip its own capability and the freshest received capabilities. Each node aggregates the received values and computes an estimate of the overall average capability. Based on this estimate, each node, $p_i$ regulates its fanout, $f_{pi}$ according to the ratio between its own and the average capability, i.e. $f_{pi} = \frac{f_{pi} \cdot \bar{b}}{\bar{b}}$, \[8\].


2.2 Data Dissemination

For solving the data delivery problem, early systems have originally used tree-based mechanisms, where the media source is at the root of the tree and media stream is pushed from the root down to the interior nodes and further down to the leaves. Some of the early systems that have used tree structure for data dissemination are: Climber [18], ZigZag [28] and NICE [2]. Despite of having short latency as an advantage, tree-based systems are fragile upon failure of nodes in the upper layers and the amount of traffic the interior nodes experience. The downside of the tree structure has been improved by dividing up the tree into multiple trees and the stream into a subset of sub-streams. Each of trees then handles one of the sub-streams. In this manner, the system becomes more resilient to node failure and load balancing also improves. However, these systems are harder to implement than single tree structured systems. One such system is SplitStream [4]. Coopnet [17] and ChunkySpread [29] are also other examples of such systems. As a third alternative, mesh-based structure got introduced. In Mesh-based systems, the media stream is divided into small blocks and nodes shape into a mesh topology and start pulling the blocks explicitly from other nodes in the system. The main disadvantage of Mesh-based systems is the unpredictable latencies nodes experience in order to receive the desired block due to the frequent requests and notifications they exchange. Examples of mesh-based systems are CoolStreaming [32], GnuStream [12] and Promise [10]. Finally, Mesh-Tree structure comes to the game to cover the disadvantages of each of the previous methods. Mesh-Tree structure as the name suggests, combines mesh and tree structures to construct a data delivery overlay. This means that the data is usually pushed through the tree, while the missing blocks can be pulled from the mesh neighbors. Some of the systems that use this mechanism are Sepidar [22], GradienTv [20], NewCoolStreaming [14] and Mtreebone [31].

Different systems and their methods in data dissemination have been summarized in table 2.1. In the next sections we will discuss some of the systems both from Live Streaming and Video-on-Demand that have inspired our model and have some similarities to our system.

2.3 Sepidar

Sepidar [22] is a peer to peer market-based model that constructs and maintains overlay trees. Sepidar constructs a minimal height streaming tree for a number of substreams, called stripe, for delivering live media content. This mechanism allows more nodes to contribute bandwidth and the constructed trees will be more robust. Nodes with higher upload bandwidths are closer to the media source, which resides at the root of the tree. Sepidar is run against a similar set of nodes in terms of upload bandwidth that are samples generated by the Gradient overlay network.
CHAPTER 2. BACKGROUND AND RELATED WORK

<table>
<thead>
<tr>
<th>Single Tree</th>
<th>Multi Tree</th>
<th>Mesh</th>
<th>Mesh-Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectedStream</td>
<td>Coopnet</td>
<td>BulkTree</td>
<td>Sepidar</td>
</tr>
<tr>
<td>HyMoNet</td>
<td>ForestCast</td>
<td>CollectCast</td>
<td>GradienTv</td>
</tr>
<tr>
<td>Yoid</td>
<td>Zebra</td>
<td>Promise</td>
<td>Prime</td>
</tr>
<tr>
<td>Nice</td>
<td>SpliStream</td>
<td>SAAR</td>
<td>Pulsar</td>
</tr>
<tr>
<td>ZigZag</td>
<td>SAAR</td>
<td>GnuStream</td>
<td>CliqueStream</td>
</tr>
<tr>
<td>Climber</td>
<td>Orchard</td>
<td>CoolStreaming</td>
<td>Bullet</td>
</tr>
<tr>
<td>SAAR</td>
<td>ChunkySpread</td>
<td>Pulse</td>
<td>NewCoolStreaming</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chainsaw</td>
<td>Mtreebone</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MeshCast</td>
<td>GridMedia</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tribler</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dagstream</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1. Data Dissemination Topology in different Systems

Nodes at each level continuously compete with one another to become children of nodes that provide stripes closer to the media source. On the other hand, a parent node would prefer to have a child node who offers to forward the highest number of copies of stripes. Therefore children proactively change their parents, when the market-modeled benefit of switching is greater than the cost of switching, until the trees stabilize. Sepidar also addresses the problem of freeriding by having the grandchildren of each node to report back regarding their parents behavior. This way parents get to audit their children periodically.

Gradient overlay [26] in Sepidar is an underlying layer that organizes the peers into a gradient structure based on any specific system characteristic called peer utility by gossiping. Nodes in Gradient for Sepidar are organized into different levels based on their upload bandwidth. When nodes sample their neighbors, they receive nodes that have similar upload bandwidth to theirs. Therefore, using Gradient improves the speed of convergence of the trees in Sepidar. We will discuss the details of Gradient in the next chapter.

As for data dissemination, the video file is divided into equal sized blocks which are delivered to nodes over multiple sub-streams, called stripes. The number of stripes any node is willing and able to forward to other nodes is called its upload slots. This number is proportional to the amount of upload bandwidth capacity at that node. On the other hand, nodes all have sufficient download bandwidth capacity to receive all the stripes.

Sepidar System has introduced three properties, namely: Currency, the total number of upload slots at a node, which is used for bidding for a connection to another node’s upload slot. Price, the minimum currency that is needed to bid for a connection. Cost, is calculated for each stripe and it is the distance from one node to the root of a particular stripe. This means that nodes can have different prices for different stripes they have. Working quite similarly like other auction algorithms, in Sepidar nodes iteratively bid for upload slots. A child node bids its
2.3. SEPIDAR

Figure 2.1. Peers organization into different market levels; The similar view and fingers of node P.

entire currency for each stripe it is interested in on upload slots at parent nodes with lowest cost. If a parent node has at least one free upload slot or one free-riding child the price for that upload slot will be zero, otherwise the price for an upload slot will be equal to the currency of one of its children with the lowest number of upload slots. Parent nodes will switch their lowest currency children with those nodes that have more currency than the current price for an upload slot. This phase is called the continuous auction. This model in contrast to auction algorithms is decentralized; meaning that each node only has a partial view from all nodes in the system it can bid with. This is where the Gradient overlay plays its role by connecting peers that have similar upload slots with each other.

There are five market levels based on 5 different range of upload bandwidths that along with node’s upload bandwidth define the node’s utility value. Each node calculates this value locally and will communicate it to other nodes once it received a utility share request message. A node prefers to fill its similar-view (the partial view kept locally) with nodes from the same or slightly higher utility value. In order for the low-bandwidth nodes to be able to utilize the higher bandwidth nodes’ spare upload slots, each node also keeps one finger list which refers to higher utility nodes. Nodes organization into different market levels and fingers are shown in Figure 2.1.

This new market model has been compared with NewCoolstreaming [14,32], and different experimental scenarios show that Sepidar improves the playback continuity and decreases the average playback latency at clients compared to NewCoolstream-
2.4 GVoD

GVoD is a P2P VoD protocol that is built on top of Gradient overlay network. The Gradient overlay network is also built on top of a Random overlay network. Each protocol at each layer samples nodes from the underlying running protocol. Nodes at each layer construct their neighbor sets by gossiping. (Gossiping is discussed in the next chapter, where Gradient is also described in more details.) This three-layered gossiping architecture is shown in Figure 2.2.

At the lowest level, nodes are connected to one another randomly. At the second level, the gradient topology for GVoD is built based on each node’s download position. Download position in the video file is calculated by each node and saved as its local utility. In this manner, nodes are organized based on their Download Position, meaning that nodes with higher download positions are located at the center of the gradient topology and nodes with lower download positions are located at the edges of the topology. Gradient topology for GVoD is shown in Figure 2.3, where the color of the nodes indicate their utilities. The darker the node color, the higher the utility for that node. Note how the similar utility nodes are connected to one another constructing the gradient topology. Nodes keep neighbors with similar utility values in their similar views.

Each video file is divided into chunks of 2MB size and chunks are further divided into pieces of 16KB size which is the minimum unit of playable file content and pieces also consist of sub-pieces of 1KB size which are the minimum unit of network
2.4. GVOD

Fig. 2.3. The Gradient Topology for GVoD

transfer. In GVoD, each node has three sets of neighbors: a utility set, an upper set and a lower set. The utility set for node $p$ consists of nodes that have similar utility values to $p$, i.e. nodes whose download positions are either in the same chunk or one chunk higher. The utility set is symmetric; if node $p$ is in $q$’s utility set, then node $q$ is also in node $p$’s utility set. The upper set consists of nodes with higher download positions, at least 2 chunks ahead. When node $p$ adds node $q$ to its upper set, node $q$ adds node $p$ to its lower set. Therefore, both upper set and lower set are asymmetric.

GVoD uses rarest first policy of BitTorrent protocol for piece sharing. Nodes ask for pieces they need from their neighbors in their utility set and if that piece is not available in their utility set, they request it from their neighbors in their upper set using an in-order policy. The final constructed topology for GVoD is a tree like structure.

The authors have evaluated the performance of GVoD by measuring the total time needed to download a complete video for different bandwidths. They show that on average the time needed to download the whole file is less than the time needed to watch the movie. They have also shown that the startup buffer time before the movie playback without interruption is approximately 20 seconds. Finally, they conclude that their approach has good results for nodes with enough upload bandwidths.
Chapter 3

Live Streaming and Video on Demand

Every P2P Live Streaming and Video on Demand system consists of two key architectural components: the overlay topology and the data dissemination algorithm. The former deals with the construction and the maintenance of an overlay topology and the latter focuses on chunk scheduling and peer selection algorithms. The issues both systems deal with are: system dynamics, heterogeneity, churn, network congestion, system synchronization, multipath routing and topology stability and convergence. In this chapter we describe the properties of a peer in each system.

3.1 Live Streaming

In this section, we describe the most important peer properties for P2P live streaming systems.

3.1.1 Live Streaming peer properties

- **Upload Bandwidth**: Mostly, nodes have enough download bandwidth to be able to get the contents and watch the video with a high quality. However, nodes normally do not have enough upload bandwidth (A node’s bandwidth is typically asymmetric, i.e., upload bandwidth is always less than download bandwidth) to be able to contribute to the network and for that reason even if a node has enough download bandwidth to get the contents at a fast rate, some of its download bandwidth will be wasted as there might not be enough upload bandwidth available. Therefore, it is important to find those nodes that have higher upload bandwidths to have them serve more nodes than those who have less upload bandwidths.

To measure the upload bandwidth of a peer, a dedicated server (bootstrap server for instance) can periodically (by the time the peer joins the system and after that every 1 minute) send a packet to it (The packet can be the size of a piece) and wait to get the packet back (An established TCP connection is needed). To further relax this feature, the upload bandwidth can be measured
only at the time the peer joins the system. Multiple threads, sending multiple packets at a time.

There is also the possibility of having the user to enter its upload and download bandwidth at the beginning before the streaming starts. Also some approaches have been offered by [27] and [5].

- **Joining Time:** The joining time in live streaming matters, as each node has only the content from the time it joins until it leaves. (For instance from time 9 to time 20). This will be fine for a solo live streaming system, as later when another node joins the system, it will watch from the time the source is currently at, therefore other nodes already in the system can be the newly joined node’s neighbors and send the contents to it. However, in an integrated Live-VoD system, if a VoD streamer joins at time 20 and it wants to watch the video from time 0, only those live streamers can be the newly joined node’s neighbor who have watched the video from time 0. As the ones who have joined later will not have all the chunks needed. (For instance chunks from time 0-9)

- **Source Rate:** The rate the source generates the content and pass them on to the directly connected nodes matters, as those nodes should have enough time to decode the data received. When the source is generating the content at a very fast rate, nodes might not have enough time to decode the content and hence some content might be delayed to get to the player and even if they arrive later, they will be discarded and ignored. Therefore, the video quality will be reduced or the playback delay will be increased (Due to buffering). This leads us to the fact that nodes closer to the source should be nodes with higher upload bandwidth. As suggested by [24], the maximum achievable streaming rate is:

$$r_{max} = \min\left(u_s, \frac{u_s + \sum_{i=1}^n u_i}{n}\right)$$

(3.1)

where $u_s$ is the server’s uplink capacity, $u_i$ the capacity of peer $i$, and $n$ the number of peers in the system. Following that the maximal delay $D$ verifies:

$$D \geq \min(d \geq 0 - G(d) - G(d - 1/r) \geq n)$$

(3.2)

where $G(t)$ denotes the maximal number of blocks that can be spread in the system, $r$ is the streaming rate and $n$ the number of peers in the system.

One insight that can be learnt from this argument is that, to spread blocks at optimal speed $G$, one must target fast peers preferentially to slower ones. (Which is what we intend to do as the nodes with higher upload bandwidth are going to be closer to the media source and will have more fanouts accordingly and will serve more peers)

- **NAT/Firewall:** Most of the users of streaming services (VoD or live streaming) are home users that are behind NATs or have their firewalls enabled.
3.2. VIDEO-ON-DEMAND

These nodes are called private nodes as public nodes (nodes that are not behind NATs) cannot establish a direct connection to them. For more details on NAT/Firewall refer to the work Payberah et al. [19] and Roverso et al. [25] have done on NAT traversal on a peer sampling service.

- **Locality (Geographical, AS-Level, Topological):** Preferably nodes should be connected to nodes that are locally close to them. There are three different levels of locality: 1) Geographical (the same continent area) 2) AS-level (same AS) 3) Topological (same access network). More details on network locality can be found in [13].

- **Playback Buffering:** Peers have to contribute a playback buffer in memory. The playback buffering can be used to not only contribute to live streamers, but also to those VoD streamers who choose to watch the same video as the live streamers, but want to start from time 0. In that case if a live streamer goes really ahead in time (for instance to time 13) and if it only has the content that it has cached, then it only has for instance the last 5 minutes and not the first 25 minutes, therefore it seems necessary for the live streamers to also contribute some storage area.

- **Chunk/Piece Loss:** Due to the fact that in live streaming nodes are interested in watching the video live, some chunks/pieces that are delayed or lost should be ignored. Some approaches such as Network coding in [9] and [15] have been suggested, where some redundancy is added to the original data to let the receivers recover from chunk losses or delivery beyond the playback delay.

### 3.2 Video-on-Demand

It is more challenging to design a P2P VoD system than a P2P Live Streaming system, because peers in VoD can arrive at arbitrary times to watch the video at the time of their request. Peers may not be synchronized and therefore the opportunity of data sharing reduces. In this section, we describe the most important peer properties for P2P VoD systems.

#### 3.2.1 Video on Demand peer properties

- **Download Position:** In a P2P VoD system, a peer p can choose only those peer qs whose download position is ahead of peer p’s and not the other way around. Therefore, q is p’s neighbor, but p is not q’s neighbor (this represents an asymmetric model). Although, this can change if for instance peer p chooses to jump to a download position ahead of peer q’s download position. In that case, peer p will not be interested in having peer q as its neighbor and it should choose some other peers whose download positions are again ahead of peer p’s or simply have the chunks peer p is interested in. Interestingly, peer q will
later need peer \( p \) and choose it as its neighbor. Also, depending on when the peer has started watching the video and which parts of the video the peer was interested in and jumped to and also its available bandwidth, its download position varies and might be different with its neighbors.

- **Upload Bandwidth**: Same as in Live streaming.
- **NAT/Firewall**: Same as in Live streaming.
- **Locality (Geographical, AS-Level, Topological)**: Same as in Live Streaming.
- **Playback Buffering/Disk Storage**: In VoD every peer needs to contribute a small amount of storage as well as the playback buffer in memory.
- **VCR Functionalities**: VCR functionalities include rewind, fast forward, pause, stop and etc. VCR functionalities are out of the scope of this thesis work. Huang et al. \cite{Huang2011} have covered VCR functionalities in more detail.
Chapter 4

Peer Organization: Gradient Topology

Unlike what the name suggests, peer to peer systems do not consist of identically capable peers who play the same role or behave exactly the same in the system. In fact peers have heterogeneous characteristics, such as different storage space, bandwidth, session duration, CPU power, which result in a non-uniform distribution in P2P topologies. This variation in peer to peer systems resources is one property that can be exploited to achieve a better performance and a higher reliability for topology construction and maintenance. By organizing peers into different categories, the system can distinguish between super peers (peers with higher capabilities and stability) with ordinary peers (peers with lower capabilities). Super peers then can be assigned to more responsibilities and handle the core system functionalities and serve other peers.

In Gradient topologies, a utility metric reflects a peer’s capability to contribute resources to the system. This utility metric is a function that is evaluated at every peer locally depending on particular system’s requirements. Gradient topologies have one major property that differentiates them from traditional super peer topologies which is called the utility threshold. Any peer that has a utility value higher than this threshold can become a super peer and any peer with a utility value below this threshold will be considered an ordinary peer. This threshold can be increased or decreased to adjust and balance the number of super peers to ordinary peers according to the system’s requirements. Ordinary peers can search for and discover super peers in the system using a heuristic called Gradient Search.

As mentioned earlier, Gradient topologies are system-specific and can be exploited in different applications with different system properties. Some of the applications of gradient topologies include: i) Storage Systems ii) Name Services iii) File Sharing and last but not least iv) Media Streaming. In this chapter, we will explain the Gradient Topology. However, we will mainly focus on peer utility function and neighbor selection algorithms that are mostly related to our work.
4.1 Gradient Topology

A Gradient topology is a P2P topology that organizes the peers using a local utility function that is run at each node in the network. At the core, the highest utility peers reside and nodes with lower utility values are located further from the core in a descending order. In other words, the utility value of each peer determines its position in the topology. This peers organization has been shown in Figure 4.1, where nodes with similar utilities (in the next section we will define what we mean by similar utilities) are shown with the same color and the darker the color, the higher the node’s utility value. As it can be seen, nodes with similar utilities are connected to each other. Nodes with the highest utility value, i.e. the black nodes, are at the center of the overlay network and as the nodes’ utility value decreases their distance to the core increases [26].

4.2 Peer Utility

In order for the peers to be evaluated in a Gradient topology and find their similar neighbors, they are associated with a utility value that is a system-specific property. For instance, in a P2P storage system, the peer utility can be a peer’s available storage space or its available bandwidth and in a Multi Media streaming system, it might be a peer’s bandwidth, its latency or both. This value is calculated by each peer locally, therefore, peers at the beginning know only their own utilities and in order for them to get other peers’ utility values, they need to communicate this information with each other. This information sharing is done by gossiping.

Gossiping is a cyclic/periodic pair-wise interaction between peers that solves the
4.2. PEER UTILITY

problems in dynamic large scale systems. During each cycle/period, peers exchange information with one another that results in changing the state of one or both peers in a way that reflects the state of the other peer. It is a scalable, simple technique that is also robust to node failures, message loss and network partitions. Gossiping protocols have been used in bootstrapping and membership protocols, aggregation protocols, topology construction protocols, global failure detection protocols and many others.

Every peer that wants to disseminate some message does the following:

- It caches every message it receives up to a certain capacity (cache size)
- It forwards the message a limited number of times $t$
- It forwards the message each time to $f$ (fanout) randomly selected set of peers

Gossiping in gradient is used to i) Generate a Random Set of neighbors ii) Generate a Similar Set of neighbors iii) Execute an aggregation algorithm to estimate the Utility Distribution of the System.

4.2.1 Dynamic Utility

Every peer in the system has a utility value here after called, $U(p)$. $U(p)$ of peer $p$ is a number that reflects the appropriateness of peer $p$ to act as a super-peer and to determine its position in the topology. If this utility value is a peer property that constantly changes, (such as storage space, bandwidth, ..), peers need to re-compute their utility values over time. To reduce the overhead, one possibility is for the peer to cache its latest utility and re-evaluate the cached values older than a predefined parameter $t_{max}$. Another possibility is to calculate the peer utility periodically. For the same reason, peers need to maintain a cache that contains the most recent utility value, $U_p(q)$ for each neighbor $q$. Each utility value is associated with a timestamp. Utility of peer $p$ at time $t$ is defined as:

$$ U_{p,t}(p) = \frac{1}{T}(S_{p,t} + S_{p,t-1} + S_{p,t-2} + ... + S_{p,t-T+1}) $$  \hspace{1cm} (4.1)

4.2.2 Utility Uniqueness

If the peer utility is not unique for each peer, then construction of a gradient topology may become impossible. To address this problem, each peer can add a relatively small random number to its utility value to break the symmetry with other peers. Then the utility of peer $p$ is defined as:

$$ U_p(P) = \dot{U}_p(P) + \epsilon_p $$  \hspace{1cm} (4.2)
4.2.3 Composite Utility

If in a system a multiple peer properties are to be used as the peer utility, then these properties can be combined into one utility function. In that case peer utility can be defined as:

\[ U(p) = \varphi_1(p) \cdot \varphi_2(p) \cdots \varphi_m(p) \]  \hspace{1cm} (4.3)

where \( \varphi_i(p) \) is a peer property. And in order to assign a weight on each of the peer properties, the composite utility can be calculated as:

\[ U(p) = \alpha_1\varphi_1(p) \cdot \alpha_2\varphi_2(p) \cdots \alpha_m\varphi_m(p) \]  \hspace{1cm} (4.4)

4.3 Neighbor Selection

The neighbor selection algorithm in a completely decentralized P2P environment, generates and maintains the gradient topology using gossiping. Each node has two subsets of neighbors and hence keeps two different views, a similar view and a random view. The similar view consists of the nodes that have similar or slightly higher utility values than the node and the random view consists of a set of uniformly random sample of nodes. As said earlier the utility value of each node is not only used locally for each peer to decide which nodes to connect to but also it is the utility value that reflects whether a node is a good candidate for becoming a super peer or not. The random view is used both to discover new nodes for the similar view and to prevent partitioning of the similar view and hence avoiding clustering.

Nodes periodically gossip to exchange and update their similar-views. Each node keeps a descriptor for each of its neighbors in its similar view that contains their utility value and other required information such as the node’s IP address. When a node receives a set from a neighbor, it selects a peer that has closest utility to its own and replaces one entry (based on a criteria, for instance the oldest peer) in its similar view. This is how the gradient topology gets generated. Peers also select one random peer from a received random view from a neighbor and replace one entry in their random view with the newly selected one. This is how peers get the opportunity to explore the system further in order to find potentially similar neighbors.

4.3.1 Similar View

Neighbor selection in Gradient is based on a preference function, meaning that peers aim to connect to other peers that have similar and slightly higher utility to them. This preference functions is defined as: Peer \( p \) prefers \( q \) over \( r \) for its similar view \( (q \succ_p r) \) if and only if

\[ U_p(q) > U_p(p) \text{ and } U_p(r) < U_p(p) \]  \hspace{1cm} (4.5)
4.3. NEIGHBOR SELECTION

or

\[ |U_p(q) - U_p(p)| < U_p(r) - U_p(p) | \]

where \( U_p(q) \) and \( U_p(r) \) are the \( p \)'s cached utility values for neighbors \( q \) and \( r \). After selecting peer \( q \) using the preference function, peer \( p \) retrieves similar view from \( q \) and merges received view by preserving most recent values in cache by their timestamps. Finally peer \( p \) and peer \( q \) perform a symmetric add. This is because neighbor relation in gradient is symmetric, meaning that if \( q \) is in \( p \)'s similar view, \( p \) is also in \( q \)'s similar view.
Chapter 5

The Model

Our model called Live-VoD has covered both problems of P2P Media Streaming systems, Node Discovery and Data Dissemination. In this chapter, we describe our model in more details. We break down the components of Live-VoD and other components Live-VoD interacts with. Before getting into any details, for the sake of clarity, we define the variables and notations used in the rest of the report in Table 5.1.

5.1 Live-VoD Model

We have assumed a network of nodes that can join at any time to watch a video that is being streamed on a Streaming Server or has already been streamed. Nodes can either choose to watch the video live from the time they join the network or from the beginning. We call the nodes that choose to watch the video live, Live streamers and the nodes that choose to watch it from the beginning, the VoD streamers. The first nodes that join Live-VoD are Live streamers and in order for the first VoD streamers to be able to join, we assume that there exists a positive number of Live streamers in the system. Once the video comes to its end, the streaming server stops streaming and no more Live streamers can join the system, however, VoD streamers can still join the system and watch the video from the beginning by contacting Live and VoD seeders.

The video file is divided into $|C|$ number of chunks and each chunk is further divided into $|P|$ number of pieces, without any coding. Every $c \in C$ has a sequence number to represent its playback order in the stream and every $p \in P$ has a sequence number to represent its position in a $c \in C$. The video file is initially streamed at a streaming sever which has a positive number of children that are only Live nodes. Each node has a buffer map, called $BM(p)$, where it stores all the chunks and the pieces it has. Nodes periodically ask for data from their neighbors and send a portion of their buffer map, which is called buffer window denoted as $BW(p)$, to them to inform them of the last available data they have received and exists in their Buffer Map.
CHAPTER 5. THE MODEL

<table>
<thead>
<tr>
<th>Notation</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>node $p$</td>
<td>-</td>
</tr>
<tr>
<td>$\mathcal{R}(p)$</td>
<td>Random View of $p$</td>
<td>5.3.3</td>
</tr>
<tr>
<td>$\mathcal{I}(p)$</td>
<td>Similar View Incoming of $p$</td>
<td>5.3.1</td>
</tr>
<tr>
<td>$\mathcal{O}(p)$</td>
<td>Similar View Outgoing of $p$</td>
<td>5.3.1</td>
</tr>
<tr>
<td>$\mathcal{S}(p)$</td>
<td>Similar View of $p$</td>
<td>$\mathcal{S}(p) = \mathcal{I}(p) \cup \mathcal{O}(p)$ 5.3.1</td>
</tr>
<tr>
<td>$\mathcal{L}(p)$</td>
<td>Live View of $p$</td>
<td>5.3.1</td>
</tr>
<tr>
<td>$\mathcal{V}(p)$</td>
<td>VoD View of $p$</td>
<td>5.3.1</td>
</tr>
<tr>
<td>$\mathcal{A}(p)$</td>
<td>Set of all neighbors of $p$</td>
<td>$\mathcal{A}(p) = \mathcal{S}(p) \cup \mathcal{V}(p) \cup \mathcal{L}(p)$</td>
</tr>
<tr>
<td>$UBL(p)$</td>
<td>Upload Bandwidth Level of $p$</td>
<td>Algorithm 1</td>
</tr>
<tr>
<td>$UBL^U(p)$</td>
<td>Upload Bandwidth Level Upper Boundary of $p$</td>
<td>Algorithm 1</td>
</tr>
<tr>
<td>$UBL^L(p)$</td>
<td>Upload Bandwidth Level Lower Boundary of $p$</td>
<td>-</td>
</tr>
<tr>
<td>$n_R$</td>
<td>Max Random View Size</td>
<td>-</td>
</tr>
<tr>
<td>$n_I$</td>
<td>Max Similar View Incoming Size</td>
<td>Algorithm 1</td>
</tr>
<tr>
<td>$n_O$</td>
<td>Max Similar View Outgoing Size</td>
<td>Algorithm 2</td>
</tr>
<tr>
<td>$n_L$</td>
<td>Max Live View Size</td>
<td>-</td>
</tr>
<tr>
<td>$n_V$</td>
<td>Max VoD View Size</td>
<td>Algorithm 4</td>
</tr>
<tr>
<td>$u(p \rightarrow q)$</td>
<td>Usefulness of $p$ for $q$</td>
<td>5.6</td>
</tr>
<tr>
<td>$\mathcal{C}$</td>
<td>Set of all chunks</td>
<td>5.3.7</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>Set of all pieces</td>
<td>5.3.7</td>
</tr>
<tr>
<td>$\mathcal{T}$</td>
<td>Set of all time indices</td>
<td>5.3.7</td>
</tr>
<tr>
<td>$BM(p)$</td>
<td>Buffer Map of $p$</td>
<td>5.4.3</td>
</tr>
<tr>
<td>$BW(p)$</td>
<td>Buffer Window of $p$ at a specific time</td>
<td>$\mathcal{I}(p)$</td>
</tr>
<tr>
<td>$BW_{\mathcal{A}(p)}(p)$</td>
<td>Buffer Windows of all neighbors of $p$</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Table 5.1. Notations used throughout the report

When $p$, either Live or VoD, joins the system, it will contact the Bootstrap Server to be introduced to some peers that are already in the system. The Bootstrap Server responds to $p$’s request by sending a set of random insiders already in the system. $p$ starts contacting those random insiders to construct its partial views to be able send data requests to download chunks and pieces of the video file and store them in $BM(p)$ and $BW(p)$. Once, $p$ has enough data in its $BM(p)$, it will start playback.

We have assumed that all nodes are willing to contribute and none of them are malicious or have intentions to intentionally break down the system. Issues such as free-riding and providing incentives for peers to contribute to each other are out of the scope of this thesis work and are not covered in our model. The design principles and key characteristics of Live-VoD are enumerated below:

1. Live-VoD is a peer-to-peer Media Streaming protocol
5.2. LIVE-VOD ARCHITECTURE

2. Live-VoD is Mesh-Tree based

3. Live-VoD is push-pull based

4. Live-VoD is Unstructured: Nodes in Live-VoD gossip for both node discovery and data dissemination

5. Live-VoD reduces the amount of message passing required for overlay construction and maintenance and data dissemination between peers

6. Nodes have some knowledge of other peers in the system, i.e., they have partial views that is taken from the overlay construction and maintenance protocol, in which the construction protocol constructs the overlay for the first time and the maintenance protocol acts like a peer sampling service and periodically updates the constructed overlay.

7. Nodes can determine their download position, called $DP(p)$, in the stream and have a $BM(p)$ that saves the chunks and pieces they have downloaded from their neighbors. These will be taken care of by the content dissemination protocol.

8. Nodes know or can estimate their upload bandwidth, called $UB(p)$, which is used by $p$ to determine the number of upload slots it can have if it is either Live or VoD and to determine its position in the overlay if it is Live.

5.2 Live-VoD Architecture

We have used Kompics \[1\], a component model that can be used for building distributed systems and peer to peer systems by composing protocols programmed as event-driven components in both simulation mode and execution mode. In this section, we will demonstrate what the components of Live-VoD in Kompics are and how they are connected to one another.

5.2.1 Live-VoD System Architecture

In Figure 5.1 we have shown all the components of our system designed in Kompics simulation mode. The model consists of 7 components, of which one of them, called Live-VoDPeer, is where Live-VoD protocol runs. We have shown what components Live-VoDPeer consists of in Figure 5.2. Apart from Live-VoD, Live-VoD peer has two other components i.e. Monitor Client and Bootstrap Client. Bootstrap Client component provides a bootstrap service to the peer, meaning that when the peer joins the system for the first time it sends a Bootstrap Request to the client which upon receiving a list of random peers from the Bootstrap Server, sends back a Bootstrap Response to the peer. The Monitor Client component runs a monitoring protocol on each peer to inspect the status of different components running on the
peer component or interacting with it. It gathers all the information and reports 
them to the Monitor Server.

5.2.2 Servers Architecture

As it can be observed in Figure 5.1, there exists three server components in our 
model. These servers are: i) The bootstrap server ii) The monitor server and iii) 
The streaming server (the media source). Each of these servers consists of three 
components. The main component is where the algorithm of that server runs. The 
other two components are Timer and Network. Timer component gives the possi-
bility of simulating the time for scheduling tasks/events and Network component 
simulates the transport protocol for sending and receiving the messages that are 
exchanged between each of components at either the same machine or with other 
components at other machines.

Bootstrap Server

The bootstrap server has the responsibility of introducing random peers that are 
already in the system to any node that joins the system. The random peers the 
bootstrap server introduces to the newly joined peer \( p \) become its initial neighbors 
in its Random View, called \( R(p) \). Once the peer connects to each of these neighbors, 
it can start gossiping to create its other views.
5.2. LIVE-VOD ARCHITECTURE

![Diagram of Live-VoD Peer components in Kompics.]

**Monitor Server**

The monitor server keeps track of the peers and their connections. The functionality of the monitor server is to take snapshots at any point in time from the constructed overlay. It receives reports from the Monitor Client component of each Live-VoD peer in the system, aggregates the status of all peers and presents a global view on a web page or an application GUI. In Live-VoD, we have used the monitor server to measure the performance of our protocol and report the global view to output evaluation files. We also have defined a GUI, where the nodes and their connections can be viewed if desired.

**Streaming Server**

The streaming server or the media source is the server where the media file resides. The streaming server generates the stream at a fixed bit rate and disseminates the data chunks to its immediate children, which are only Live nodes. At the time any Live node joins the system, it sends a Download Position Request to the Streaming Server which gets back to it with the last time index $t$ it has generated. The Streaming Server can also add a positive number of Live nodes to its children list to push data to them.
CHAPTER 5. THE MODEL

5.3 Live-VoD: Overlay Construction

Live-VoD consists of three different layers for overlay construction, of which the highest layer is where the Live-VoD overlay gets constructed. Live-VoD overlay uses two underlying layers i.e. Gradient Overlay and Random Overlay for node discovery. Figure 5.3 shows how overlays get constructed by sampling from one another. The lowest layer is the Random Overlay, which is built using Random Peer Sampling Cyclon \[30\]. The layer above it is Gradient Overlay \[26\] which is the basis of our node discovery algorithm for Live-VoD. As we mentioned in \[4\] a gradient topology gets constructed according to peer utility values that each peer calculates locally using a utility function by sampling peers from the underlying random overlay. Live overlay samples from Gradient Overlay, while VoD overlay get constructed by sampling from Random and Live Overlay. In the sections below we explain in more details how each of these overlays get constructed and how they sample from one another.

5.3.1 Live-VoD Protocol

In this section, we describe how our Live-VoD protocol works in details. In Live-VoD, each node has three neighbor views, Random View denoted by \(R(p)\), Similar View denoted by \(S(p)\) and depending on being a Live node or a VoD node, a Live
5.3. LIVE-VOD: OVERLAY CONSTRUCTION

View referred to with $L(p)$ or a VoD View referred to with $V(p)$. Each of these views will be populated at a specific overlay layer mentioned earlier. Nodes populate their views by gossiping and based on certain criteria.

At the time of joining if a node is Live, it will be introduced only to Live nodes, on the other hand, if a node is VoD, it will be given a list of both Live and VoD nodes. This is because a Live node might be able to contribute to a VoD node in two situations: i) If there is no VoD node already in the system ii) If it has some parts of the data that the VoD node is interested in.

We have summarized the overlay construction protocol in Live-VoD below:

- Both Live and VoD nodes connect to some random nodes to prevent the overlay network from clustering. Live nodes have only random Live nodes in their Random Views, VoD nodes have both Live and VoD random nodes in their Random View.

- Live nodes, assuming they all have similar download positions, are connected to each other based on their upload bandwidths in the center of the overlay. Live nodes keep their similar and higher upload bandwidth neighbors in their Similar View Incoming, $I(p)$ and the lower upload bandwidth neighbors in their Similar View Outgoing, $O(p)$.

- VoD nodes connect to nodes that are more useful for them i.e. nodes that can contribute more pieces to them. They keep the VoD nodes that are useful for them in their Similar View Incoming, called $I(p)$ and the VoD nodes they are useful for in their Similar View Outgoing, denoted by $O(p)$.

- As VoD nodes are behind the the Live nodes, they reside further away from the center (where the Media Source and Live nodes reside).

- Live nodes also accept connect requests from the VoD nodes they are more useful for(have more pieces to contribute to). Live nodes keep these VoD nodes in their VoD View, and VoD nodes that receive the accept request from Live nodes, keep them in their Live View.

In Figure 5.4 Live-VoD overlay has been shown. The color of the nodes indicate their upload bandwidth (Blue Spectrum for Live nodes and Orange Spectrum for VoD nodes). As it can be observed, the Live nodes are all gathered at the center, as they all have the same download position. VoD nodes, however, are further away from the center because they are either slightly or highly behind Live nodes.

5.3.2 Live-VoD: Peer Utility and Partnering

Each node $p$ regardless of being a Live or a VoD node has a utility value which is the combination of Download Position, $DP(p)$, and Upload Bandwidth, $UB(p)$. Nodes constantly receive a utility value from other nodes and update their views accordingly.
When a new peer wants to join the Live-VoD system, it contacts the bootstrap server to determine whether it is Live or VoD and to specify its upload bandwidth. Then the bootstrap server introduces a subset of nodes currently active in the system; The bootstrap server chooses these nodes uniform randomly among all nodes already in the system. Then the newly joined node adds the node descriptor of each of these nodes to its local Random View. The node descriptor for each node consists of their type, their utility and their partners with their IP addresses. These nodes will be the newly joined node’s partners or so called neighbors. "The neighborhood relation over the nodes defines the topology of the overlay network. As such, nodes adapt the overlay topology by constantly updating their neighborhood using a preferential neighbor selection policy." 

Nodes periodically run a peer sampling service and will exchange their subset of neighbors with their neighbors using a preferential neighbor selection policy, always updating their views but never exceed the maximum view size. This is to ensure that nodes always have the best neighbors according to their needs and to allow resilience to churn.

### 5.3.3 Random Overlay

The random overlay in Live-VoD has been implemented using Cyclon\textsuperscript{30}. Cyclon is a gossip-based membership management protocol that constructs random graphs that are highly resilient to node failures, have low diameter, low clustering and highly symmetric node degrees. The random overlay network is the first layer of
5.3. LIVE-VOD: OVERLAY CONSTRUCTION

our Live-VoD overlay network that Gradient overlay samples from it. Nodes keep
their random neighbors in their Random View. Nodes periodically shuffle with their
neighbors in their Random View in order to discover better nodes and add them
to their Similar View. Therefore, maintaining Random View prevents the network
from partitioning and contributes in finding better nodes.

The first random nodes are bootstrapped from the bootstrap server at the time
of joining for each peer. Live nodes get introduced to only random Live nodes
whereas VoD nodes are handed over both random Live and VoD nodes. The random
nodes received from the bootstrap server are added into the node’s Random View.
Nodes periodically shuffle with their neighbors in their Random View executing the
following steps:

1. Increase by one the age of all neighbors.
2. Select neighbor \( q \) with the highest age among all neighbors, and \( l - 1 \) other
random neighbors.
3. Replace \( q \)’s entry with a new entry of age 0 and with \( p \)’s address.
4. Send the updated subset to peer \( q \).
5. Receive from \( q \) a subset of no more than \( l \) of its own entries.
6. Discard entries pointing at \( p \) and entries already contained in \( p \)’s cache.
7. Update \( p \)’s cache to include all remaining entries, by firstly using empty cache
slots (if any), and secondly replacing entries among the ones sent to \( q \).

Having done the shuffling, the newly merged Random View that includes the
newly received Random nodes will be used to update the Similar View. Algorithm
1 describes how the Similar View will be updated using the Random View.

The worst UB neighbor is a node with the least UB in a Live’s Similar View.
The least useful neighbor is a node that is least useful for a VoD node in terms of
being able to contribute the missing pieces of the media file to it. We will explain
how the usefulness is calculated in more details in [5.3.5]

5.3.4 Gradient Overlay for Live

The gradient overlay for Live streaming gets constructed according to upload band-
width of the nodes. Therefore, nodes with higher upload bandwidth are closer to
the core and nodes with lower upload bandwidth are on the edges of the overlay.
This allows the nodes with a higher upload bandwidth to be closer to the source
and hence be able to receive the stream at the rate the source is generating the
stream. Each Live node keeps its upload bandwidth as its utility to be able to find
similar and slightly higher UB neighbors. Live nodes keep their similar and higher
UB neighbors in their Incoming Similar View and their lower UB neighbors in their
Outgoing Similar View.
CHAPTER 5. THE MODEL

Algorithm 1 Update Similar View Using Random View after periodic shuffling
1: for all \( q \in R(p) \setminus S(p) \) do
2: if \( p \) is Live then
3: if \( UBL(p) \leq UBL(q) \leq UBL_U(p) \) then
4: \( \mathcal{I}(p) = \mathcal{I}(p) \cup \{q\} \)
5: if \( |\mathcal{I}(p)| > n_\mathcal{I} \) then
6: \( q_u = \arg\min_{q \in \mathcal{I}(p)} UB(q) \)
7: \( \mathcal{I}(p) = \mathcal{I}(p) \setminus \{q_u\} \)
8: end if
9: end if
10: else
11: if \( u(q \rightarrow p) > 0 \) then
12: \( \mathcal{I}(p) = \mathcal{I}(p) \cup \{q\} \)
13: if \( |\mathcal{I}(p)| > n_\mathcal{I} \) then
14: \( q_{pw} = \arg\min_{q \in \mathcal{O}(p)} UB(q) \)
15: \( \mathcal{I}(p) = \mathcal{I}(p) \setminus \{q_{pw}\} \)
16: end if
17: end if
18: end if
19: end for

Algorithm 2 describes how a Live node \( p \) accepts a connect request from a Live node \( q \).

Algorithm 2 Connect a Live node to another Live node
1: if \( q \notin S(p) \) & \( UBL(q) \leq UBL(p) \leq UBL_U(q) \) then
2: if \( |\mathcal{O}(p)| < n_\mathcal{O} \) then
3: \( \mathcal{O}(p) = \mathcal{O}(p) \cup \{q\} \)
4: else
5: \( q_{pw} = \arg\min_{q \in \mathcal{O}(p)} UB(q) \)
6: if \( UBL(q_{pw}) < UBL(q) \) then
7: \( \mathcal{O}(p) = \mathcal{O}(p) \setminus \{q_{pw}\} \)
8: \( \mathcal{O}(p) = \mathcal{O}(p) \cup \{q\} \)
9: end if
10: end if

5.3.5 VoD Overlay

To construct an overlay for VoD streamers, each node calculates its local utility as its download position \( DP \) in the video file. Nodes with higher download position...
5.3. LIVE-VoD: OVERLAY CONSTRUCTION

(more ahead in time of total video length) are closer to the core and nodes with lower download positions should be located at the edges of the VoD topology. As the time passes by nodes on edges move towards the core. Once nodes at the core finish the whole playback, they start seeding other nodes who still have not. In order for each VoD node to connect to another VoD node, nodes keep their neighbors buffer window and based on that, calculate their usefulness for the received connect requests.

Algorithm 3 describes how a VoD node $p$ accepts a connect request from a VoD node $q$.

Algorithm 3: Connect a VoD node to another VoD node

```
if $q \notin S(p)$ then
    if $|\mathcal{O}(p)| < n_{\mathcal{O}}$ then
        if $u(p \rightarrow q) > 0$ then
            $\mathcal{O}(p) = \mathcal{O}(p) \cup \{q\}$
        end if
    3:
    else
        $q_u = \arg \min_{q \in \mathcal{O}(p)} u(p \rightarrow q)$
        if $u(p \rightarrow q) - u(p \rightarrow q_u) > \theta_u$ then
            $\mathcal{O}(p) = \mathcal{O}(p) \setminus \{q_u\}$
            $\mathcal{O}(p) = \mathcal{O}(p) \cup \{q\}$
        end if
    9:
end if
12:
```

$\theta_u \geq 0$ is a threshold used to prevent a frequent change of neighbors. If the usefulness of node $p$ for the newly requesting node $q$ is not at least $\theta_u$ units higher than node $p$’s usefulness for its unhappiest neighbor in its Similar View, node $p$ will not add $q$ to its Similar View. $\theta_u$ is a system parameter and the lower it is the more frequent the neighbors will change. $\theta_u$ has been used to decrease the number of messages exchanged between nodes and hence it will lower the connect/disconnect messaging overhead between nodes.

5.3.6 Live-VoD Overlay

VoD nodes can also connect to Live nodes in our system. In fact, the first VoD nodes that join the system can only download data from Live nodes and it is only after a while that VoD nodes can effectively contribute to each other. When a VoD node joins the system, it receive a number of random Live nodes which keeps them in its Random View. It then sends a connect message to each of them. When a Live node receives the connect message request, it calculates its usefulness in terms of the amount of data it can contribute to it. We define the usefulness function in Algorithm 4, describes how a Live node $p$ accepts a connect request from a VoD node $q$.
Algorithm 4 Connect a VoD node to a Live node

if $q \not\in \mathcal{V}(p)$ then
  if $|\mathcal{V}(p)| < n_V$ then
    if $u(p \rightarrow q) > 0$ then
      $\mathcal{V}(p) = \mathcal{V}(p) \cup \{q\}$
    end if
  else
    $q_u = \arg\min_{q \in \mathcal{V}(p)} u(p \rightarrow q)$
  end if
end if

if $u(p \rightarrow q) - u(p \rightarrow q_u) > \theta_u$ then
  $\mathcal{V}(p) = \mathcal{V}(p) \setminus \{q_u\}$
  $\mathcal{V}(p) = \mathcal{V}(p) \cup \{q\}$
end if

5.3.7 Usefulness Estimate

Each node $p$ keeps a $BW(p)$ for all its neighbors in its $S(p)$ and depending on whether it is Live or VoD in its $\mathcal{V}(p)$ or its $L(p)$. $BW(p)$ contains the time indices peer $p$ has downloaded, regardless of its playback time.

A media file as said earlier is divided into a number of chunks and further down to a number of pieces. If a media file contains $\mathcal{C}$ as a set of chunks, $\mathcal{C} = c_0, c_1, c_2...c_n$, and $\mathcal{P}$ as a set of pieces, $\mathcal{P} = p_0, p_1, p_2...p_m$, the number of total times, $T$, in that media file will be calculated as:

$$ |T| = |\mathcal{C}|.|\mathcal{P}| $$

and each time $t \in T$ is calculated as:

$$ t = c.|\mathcal{P}| + p $$

given $t$, chunk $c$ is calculated as:

$$ c = t/|\mathcal{P}| $$

and piece $p$ is calculated as:

$$ p = t \mod |\mathcal{P}| $$

Every node after receiving a piece of data from a particular chunk, calculates the time using equation 5.2 and keeps that time in its Buffer Window. Nodes send their Buffer Window periodically to their neighbors when they request for data. Buffer Window is used for two purposes: i) calculating an estimation of availability of $t$ in the network among a node’s neighbors for Live-VoD overlay construction. ii)
5.3. LIVE-VOD: OVERLAY CONSTRUCTION

Figure 5.5. The effect of \(DP(q)\) and availability of the time index \(t\) \((A(t))\) on the usefulness of \(p\) for \(q\). The plot shows the hypothetical contribution of different time indices \(t\) in calculating usefulness of \(p\) for \(q\) with \(DP(q) = 10\), for different values of \(t\) and \(A(t)\).

Time selection for data dissemination. The set of buffer windows for all neighbors of node \(p\) is formally defined as:

\[
BW_{A(p)}(p) = ((q, T_q) | q \in A(p))
\]  
(5.5)

Each node \(p\) periodically updates an estimation of availability of each time \(t \in \mathcal{T}(p)\) among its neighbors using their Buffer Window. The \(A(t)\) is the availability of time index \(t\) which is defined as the number of \(t\) available in all of node \(p\)'s neighbors’ Buffer Window and itself. We use \(A(t)\) to find rare time indices among all nodes of a neighborhood. The \(A(t)\) is used in calculating the usefulness of node \(p\) for node \(q\), denoted by \(u(p \rightarrow q)\) which is calculated as follows:

\[
u(p \rightarrow q) = \sum_{t \in \mathcal{T}(p) \setminus \mathcal{T}(q)} \frac{1}{A(t)(1 + |t - DP(q)|)}
\]  
(5.6)

Using equation 5.6, an estimate of the number of time indices node \(p\) can contribute to node \(q\) is estimated, where each time index is weighted by an estimate of its rareness and immediate usefulness for \(q\). The rareness of the time index \(t\) is estimated using \(\frac{1}{A(t)}\) and its immediate usefulness is estimated using \(\frac{1}{1+|t-DP(q)|}\). The smaller \(|t-\text{DP}(q)|\), the closer that time index is to \(\text{DP}(q)\) and the higher the weight will be.

In Figure 5.5, the effect of \(DP(q)\) and availability of the time index \(t\), \(A(t)\), on the usefulness of \(p\) for \(q\) are depicted. It can be observed that the contribution of each time index \(t\) (provided that \(t \in \mathcal{T}(p) \setminus \mathcal{T}(q)\)), depends on \(DP(q)\) and \(A(t)\). In this example, time indices that have smaller offset to \(\text{DP}(q) = 10\) and are least available will contribute more to the usefulness of \(p\) for \(q\).
5.3.8 Neighbor Selection by messaging in Live-VoD

In order to keep the neighborhood views up to date, peers should frequently exchange messages with their neighbors. The messages nodes send to each other are for overlay construction and maintenance are: 

- **i) Connect Message**
- **ii) Disconnect Message**
- **iii) Shuffling Message**. Below we will sum up exactly when each of these messages are sent and received from node \( p \) to node \( q \):

- **Connect Message:**
  - At the joining time, node \( p \) sends a Connect Message to node \( q \) which is among the random peers received from the Bootstrap Server
  - After shuffling with a peer in \( S(p) \), upon receiving a random set from that node, node \( p \) sends a Connect Message to any chosen node \( q \)

- **Disconnect Message:**
  - Due to Shuffling, a better node has been chosen over node \( q \), so node \( p \) sends a Disconnect Message to \( q \)
  - A newly joined peer has sent a Connect Message to \( p \) and it is chosen over node \( q \), so \( p \) sends a Disconnect Message to \( q \)

- **Shuffling Message:**
  - At the joining time, node \( p \) sends a Shuffling Message to a randomly chosen node \( q \) which is among the random peers received from the Bootstrap Server
  - Node \( p \) periodically sends Shuffling Messages to the the oldest random node \( q \) in \( R(p) \)

5.4 Live-VoD: Data Dissemination

The main goal in Live Streaming and VoD is to deliver the stream data to all peers before their playback deadline. This is due to the fact that in P2P streaming systems the video content is generated in real time and at a limited bit rate. In order for the users to watch the movie while downloading its pieces, the pieces need to be received sequentially and not in a random order.

There are two main approaches for data dissemination: 

- **i) Tree-based (Push)**
- **ii) Mesh-based (Pull)**

In the next sections, we explain each of these approaches more thoroughly and then we introduce our method for data dissemination in Live-VoD.
5.4. LIVE-VOD: DATA DISSEMINATION

5.4.1 Tree-Based/Push-based Systems

In tree-based systems nodes get arranged in form of a tree with the media source at the root of the tree and the rest of the nodes will be arranged having parent, children relationship. In a tree structure media is pushed from the root to interior nodes to leaf nodes. The main advantage of the tree structure is the short latency of data delivery. However, tree structures are very fragile upon the failure of the nodes close to the root of the media streaming. Another main problem of the push-based systems is message duplication(also called collision), when two nodes push the same chunk to the same node, this will cause the nodes to waste some of their bandwidth. Also, all of the traffic is forwarded through the interior nodes whereas the leaves experience no traffic at all.

5.4.2 Mesh-Based/Pull-Based Systems

The main challenge in mesh-based systems is how to devise an optimal scheduling algorithm, i.e. picking which chunks to send to whom in a node’s neighborhood [7]. In peer to peer streaming systems, the video file encoded at the source is chopped into small pieces called chunks. These chunks then will be distributed over a meshed overlay network in a timely manner in order to meet a deadline, for a good playback continuity and a low playback latency experience. Therefore, delivery delay is one important key factor of mesh-based systems.

The advantages of Mesh-based systems are more resilience to high rates of churn, providing many possible paths that data can traverse to get to its destination and nodes can easily change position and timely react to both membership changes and bandwidth capacity fluctuations [23]. The main disadvantage of the mesh-based systems is the higher overhead due to the frequent message passing between the peers.

5.4.3 Live-VoD: Tree-Mesh

We use a hybrid Tree-Mesh structure in Live-VoD. We have shown the structure at which the data gets disseminated through Live-VoD in Figure [5.6] Live nodes are organized into a tree structure based on their upload bandwidth, where at the root of the tree, the streaming source resides and nodes get arranged at each level of the tree based on their upload bandwidth. Nodes with higher upload bandwidth become the parent of nodes with lower upload bandwidths and push the data to them. On the other hand, nodes with lower upload bandwidth pull the data from their similar or higher upload bandwidth neighbors. VoD nodes are connected to each other in a Mesh-based structure and they pull the data from both their neighbors in their Similar View and their neighbors in their Live View.
CHAPTER 5. THE MODEL

Figure 5.6. Data Dissemination between Lives and Lives, VoDs and VoDs and Lives and VoDs.

Streaming Source

To disseminate the video content more efficiently each video file is divided into smaller fixed size video data which are called chunks. These chunks are further divided into pieces of equal size with no encoding. Every chunk and piece has a sequence number which determines its place in the data file, which allows the nodes to correctly rebuild the received data and estimate their playback delay. In Figure 5.7 we have shown how the media file is divided into chunks and chopped further into pieces. Initially, at the beginning of the movie, the media source starts generating these chunks and their containing pieces at a fixed rate. We are aware that dividing the video content into fixed size chunks that are generated at a constant rate at source is quite unrealistic. Actual video streams carry highly organized information, part of which is more important than other, and with high variability in the generated bitrate [6]. One recommend method by [7] is FEC code (Fixed-rate Error Correction) that can be applied to the data to achieve a better resilience to chunk loss. However, for the purposes of our work, this method should suffice.

Every time a Live node enters the system it sends a Request to the source/initial seed to receive the time the movie is being streamed at. The streaming source will send its current $DP$ to the requesting Live node so that it knows from where it will be downloading the media. The streaming source has a limited number of children, which consist of only Live nodes. The source pushes the data to its immediate
5.4. LIVE-VOD: DATA DISSEMINATION

Figure 5.7. Each file is divided into chunks and chunks are divided further into pieces. Both chunks and pieces are timestamped so that they can be re-ordered sequentially once delivered.

The number of children for the source is defined as follows:

\[ \text{NumberOfChildren} = \frac{DB(S)}{\text{Size of } p} \] \tag{5.7}

where \( DB(S) \) is download bandwidth of the streaming source.

The reason for only connecting the Live nodes to the source is that Live nodes need to get the data faster than VoD nodes due to the little amount of time they have for startup buffering. Once the source reaches its limitation, any node \( p \) that joins afterwards will be treated in this manner:

- If \( p \) has higher upload bandwidth than any of the nodes in source’s children list, the source will remove the worst child and add \( p \)
- If there is no child that has lower upload bandwidth than \( p \), source ignores \( p \)

Buffer Map

The set of chunks that a peer possesses is called \( BM(p) \), buffer map of node \( p \). Each node keeps a buffer map from which the useful chunks will be selected to be transmitted to its neighbors and sent to the media player for displaying. In Live-VoD, \( BM(p) \) includes the following data:

- \( DP(p) \): Download position of node \( p \)
- \( B(p) \): Buffer of node \( p \) (Set of all chunks and pieces received at node \( p \))
- \( BW(p) \): Buffer Window of node \( p \) (Set of all time indices \( p \) has)
- \( A(t \in T) \): Availability estimate of each time index \( t \) in \( T \)

These information are useful for
(i) peer discovery at the time of joining
(ii) overlay maintenance, peer sampling service
(iii) data dissemination (time selection).
Node’s Capacity

Each node can only ask for the amount of data it has enough capacity for. The available capacity of a node is calculated as follows:

\[
Capacity = \frac{DB(p)}{N} \quad (5.8)
\]

where \( N \) when \( p \) is Live is: \(|\mathcal{I}(p)|\) and when \( p \) is VoD is \(|\mathcal{I}(p)| + |\mathcal{L}(p)|\)

Time Selection

Time index \( t \in \mathcal{T} \) represents \((p_i \in \mathcal{P}\) in \( c_i \in \mathcal{C}\)). In order for a peer to pick which pieces to send to the requesting neighbor, we have designed a time selection algorithm. In Live-VoD, the policy for time selection is a stochastic process, which we will describe in more details below:

Each Live and VoD node keeps a buffer window for all its similar neighbors which consists of the past \( t \) time indices it has sent to that neighbor. Live nodes also keep a buffer window for each of their VoD neighbors in their VoD View. Therefore, once a node \( q \) sends a request to one of its neighbors \( p \) to ask for data, \( p \) will consider:

- The time indices it has: \( \mathcal{T}(p) \)
- The time indices \( q \) has: \( \mathcal{T}(q) \)
- The availability of each time index \( t \): \( A(t) \)
- Download Position of node \( q \)
- Node \( q \)’s capacity

The stochastic process consists of two steps: \( i) \) calculating a probability for sending each time index \( t \) and \( ii) \) stochastically sampling from the joint distribution of the mentioned probabilities. The probability of sending time index \( t \), from \( p \) to \( q \) is referred to with \( P(s_{p \rightarrow q}^{(t)} = 1) \) and it is defined by:

\[
P(s_{p \rightarrow q}^{(t)} = 1|A(t),DP(q)) \propto \exp \left\{ -\frac{1}{2} \left( \frac{A(t) - 1}{\sigma_A} \right)^2 - \frac{1}{2} \left( \frac{t - DP(q)}{\sigma_{DP}} \right)^2 \right\} \quad (5.9)
\]

where

- \( s_{p \rightarrow q}^{(t)} \) refers to the event of sending the time index \( t \) from node \( p \) to node \( q \) i.e. \( s_{p \rightarrow q}^{(t)} = 1 \) is the decision of sending the time index \( t \) while a value of 0 for it is the decision of not sending that time index.
- \( \sigma_A \) and \( \sigma_{DP} \) are two parameters of the model which determine the relative importance of \( A(t) \) - the availability of time index \( t \) - and the importance of the offset of time index \( t \) from node \( q \)’s Download Position - \((t - DP(q))\).
5.4. LIVE-VOD: DATA DISSEMINATION

Figure 5.8. The probability of sending time index $t$ from node $p$ to node $q$, for different values of $t$ and $A(t)$ when $DP(q) = 10$, $\sigma_A = 2$ and $\sigma_{DP} = 2$.

In Figure 5.8 the probability of sending time index $t$ from node $p$ to node $q$, for different values of $t$ and $A(t)$ where $DP(q) = 10$ and $\sigma_A = 2, \sigma_{DP} = 2$ is shown.

Once the probabilities are calculated for all $t \in T(p) \setminus T(q)$, the time indices to be sent to $q$ are picked via an iterative stochastic sampling of $P\left(\{s_{p\rightarrow q}^t|t \in T(p) \setminus T(q)\}|A,DP(q))$ according to:

**Algorithm 5** Time Selection

\begin{align*}
P &= D_{p\rightarrow q} = \emptyset \\
&\text{for all } t \in T(p) \setminus T(q) \text{ do} \\
&P = P \cup \{t, P(s_{p\rightarrow q}^t = 1|A(t), DP(q))\} \\
&\text{end for} \\
&\text{for } i = 1 \rightarrow \text{Capacity do} \\
&\text{normalize } P \text{ to sum to 1} \\
&\text{sample } t \sim P \\
&P = P \setminus \{t, \} \\
&D_{p\rightarrow q} = D_{p\rightarrow q} \cup \{t\} \\
&\text{end for}
\end{align*}

In algorithm 5, $P$ is the discrete probability distribution, constructed from $P(s_{p\rightarrow q}^t = 1)$ of the remaining time indices, from which the time indices are sampled without repetition.
Chapter 6

Results

As stated in the previous chapter, we have used Kompics for designing our model in Simulation mode of Kompics [1]. As Kompics also gives the possibility to write different experiment scenarios for system evaluations very easily, we have used it to do our experiments and evaluate our work too.

6.1 Experimental Setup

In the experiments we have run in Kompics, there are different variables that we set their values differently for each experiment, these variables are: Number of peers, Joining Order, Inter Arrival Time, Delay, Type of Peers and Bandwidth Distribution. Table [6.1.1] shows the notations we have used for each of these variables. We are going to use these notations later in this chapter to specify the values we have set in each of our experiments.

6.1.1 Bandwidth Distribution

In Table [6.2] we have specified the bandwidth distribution we have used in our experiments. Kbps is short for Kilo bit per second and is equal to 10³ bits per second.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of peers</td>
</tr>
<tr>
<td>jo</td>
<td>Joining Order</td>
</tr>
<tr>
<td>iat</td>
<td>Inter Arrival Time in ms</td>
</tr>
<tr>
<td>d</td>
<td>Delay in ms</td>
</tr>
<tr>
<td>t(p)</td>
<td>Type of Peer</td>
</tr>
<tr>
<td>D(BW)</td>
<td>Bandwidth Distribution</td>
</tr>
</tbody>
</table>

Table 6.1. Variables in Live-VoD Experiments
CHAPTER 6. RESULTS

<table>
<thead>
<tr>
<th>Upload Bandwidth</th>
<th>Download Bandwidth</th>
<th>Level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 * Kbps (1 Mbps)</td>
<td>1024 * Kbps * 10 (10 Mbps)</td>
<td>6</td>
<td>5%</td>
</tr>
<tr>
<td>512 * Kbps (512 Kbps)</td>
<td>512 * Kbps * 4 (2 Mbps)</td>
<td>5</td>
<td>15%</td>
</tr>
<tr>
<td>320 * Kbps (320 Kbps)</td>
<td>320 * Kbps * 4 (1.25 Mbps)</td>
<td>4</td>
<td>20%</td>
</tr>
<tr>
<td>256 * Kbps (256 Kbps)</td>
<td>256 * Kbps * 4 (1 Mbps)</td>
<td>3</td>
<td>30%</td>
</tr>
<tr>
<td>128 * Kbps (128 Kbps)</td>
<td>128 * Kbps * 4 (512 Kbps)</td>
<td>2</td>
<td>15%</td>
</tr>
<tr>
<td>96 * Kbps (96 Kbps)</td>
<td>96 * Kbps * 6 (512 Kbps)</td>
<td>1</td>
<td>15%</td>
</tr>
</tbody>
</table>

Table 6.2. Bandwidth Distribution used in Live-VoD Simulation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>200</td>
</tr>
<tr>
<td>$jo$</td>
<td>50-50-100</td>
</tr>
<tr>
<td>$iat$</td>
<td>50ms</td>
</tr>
<tr>
<td>$d$</td>
<td>100ms-200ms-200ms</td>
</tr>
<tr>
<td>$t(p)$</td>
<td></td>
</tr>
<tr>
<td>$D(BW)$</td>
<td>14% level 1, 10% level 2, 35% level 3, 25% level 4, 14% level 5, 2% level 6</td>
</tr>
</tbody>
</table>

Table 6.3. Variables Value in Experiment 1

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>300</td>
</tr>
<tr>
<td>$jo$</td>
<td>300</td>
</tr>
<tr>
<td>$iat$</td>
<td>50ms</td>
</tr>
<tr>
<td>$d$</td>
<td>200ms</td>
</tr>
<tr>
<td>$t(p)$</td>
<td></td>
</tr>
<tr>
<td>$D(BW)$</td>
<td>15% level 1, 18% level 2, 27% level 3, 17% level 4, 18% level 5, 5% level 6</td>
</tr>
</tbody>
</table>

Table 6.4. Variables Value in Experiment 2

6.2 Live-VoD Experiments

We have done the following experiments for one scenario i.e. Join Only:

1. **Join Only:** In every scenario we have run for our experiments, at first $n_{init}^{live}$ Live nodes ($n_{init}^{live} = 5$ in our experiments) join the network and then an arbitrary number of Live or VoD nodes with equal probability will join the system, once the live streaming is finished, only VoD nodes join the system. A video of length 4 MB is cut into 256 KB chunks and 16 KB pieces and is distributed from the initial seeder with a 1 Mb/s upload link.

   - **Experiment 1:** Table 6.3
   - **Experiment 2:** Table 6.4
   - **Experiment 3:** Table 6.5
6.3. EVALUATION

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>400</td>
</tr>
<tr>
<td>$jo$</td>
<td>50-150-200</td>
</tr>
<tr>
<td>$iat$</td>
<td>50ms</td>
</tr>
<tr>
<td>$d$</td>
<td>100ms-200ms-200ms</td>
</tr>
<tr>
<td>$t(p)$</td>
<td>30% Lives and 70% VoDs</td>
</tr>
<tr>
<td>$D(BW)$</td>
<td>11% level 1, 16% level 2, 28% level 3, 27% level 4, 14% level 5, 4% level 6</td>
</tr>
</tbody>
</table>

Table 6.5. Variables Value in Experiment 3

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>500</td>
</tr>
<tr>
<td>$jo$</td>
<td>500</td>
</tr>
<tr>
<td>$iat$</td>
<td>50ms</td>
</tr>
<tr>
<td>$d$</td>
<td>200ms</td>
</tr>
<tr>
<td>$t(p)$</td>
<td>34% Lives and 66% VoDs</td>
</tr>
<tr>
<td>$D(BW)$</td>
<td>13% level 1, 15% level 2, 31% level 3, 18% level 4, 20% level 5, 4% level 6</td>
</tr>
</tbody>
</table>

Table 6.6. Variables Value in Experiment 4

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>600</td>
</tr>
<tr>
<td>$jo$</td>
<td>600</td>
</tr>
<tr>
<td>$iat$</td>
<td>50ms</td>
</tr>
<tr>
<td>$d$</td>
<td>200ms</td>
</tr>
<tr>
<td>$t(p)$</td>
<td>28% Lives and 72% VoDs</td>
</tr>
<tr>
<td>$D(BW)$</td>
<td>14% level 1, 16% level 2, 33% level 3, 22% level 4, 9% level 5, 6% level 6</td>
</tr>
</tbody>
</table>

Table 6.7. Variables Value in Experiment 5

- Experiment 4: Table 6.6
- Experiment 5: Table 6.7

### 6.3 Evaluation

In this section, we evaluate the performance of Live-VoD for different system settings. In each of the above experiments, we have measured the average clustering coefficient and the average, min and max time VoD nodes needed to download the complete video for systems of different sizes from 200 to 600 nodes. In this section, we describe each of these metrics in detail and present the performance of our protocol using them.
6.3.1 Connectivity

One fundamental property of overlay networks is that given a fail-free environment, connectivity of all nodes in the overlay should be achieved. During the overlay construction, when Live-VoD is running, it is clear that no neighbor selection or shuffling operation should result in any individual node to get disconnected from the overlay network. One method to evaluate connectivity is by measuring the clustering coefficient. "The clustering coefficient of a node is defined as the ratio of the existing links among the node’s neighbors over the total number of possible links among them. It basically shows to what percentage the neighbors of a node are also neighbors among themselves. The average clustering coefficient is the clustering coefficient averaged across all nodes in the network. It is generally undesirable for an overlay to have a high average clustering coefficient for two reasons. First, it weakens the connectivity of a cluster to the rest of the network, therefore increasing the chances of partitioning. Second, it is not optimal for information dissemination applications due to the high number of redundant message deliveries within highly clustered node communities." [30]. Roughly speaking clustering coefficient determines how well connected the neighborhood of the node is and it is calculated as the ratio of number of connections in the neighborhood of a node and the number of connections if the neighborhood was fully connected. In other words, clustering coefficient means to what extent the neighbors of a node are also neighbors among themselves.

A fully connected group of N nodes in the network should have: \(N \times (N - 1)/2\) connections. A strongly connected directed graph has a path from each node to every other node and vice versa (e.g. AB path and BA path). A weakly connected directed graph is connected if we disregard the edge directions. The clustering coefficient for node \(p\) is calculated as below:

\[
C(p) = \frac{e_p}{deg(p)(deg(p) - 1)/2}
\]

(6.1)

Where \(e_p\) is the number of edges between the neighbors of node \(p\) and \(deg(p)(deg(p) - 1)/2\) determines the number of possible edges for node \(p\). We have measured the average clustering coefficient for a Live-VoD node by:

\[
C = \frac{1}{N} \sum_{i=1}^{N} C_i
\]

(6.2)

Figure 6.1 shows the average clustering coefficient for up to 600 nodes in a Live-VoD network. The drop in the value of the average clustering coefficient for more number of nodes is because of the max view sizes we have. When the number of nodes increases in the system, there is little chance for all neighbors to be neighbors within each other, because the maximum view size causes the nodes to always have some limited number of connections. However, this value has never reached 0 in our experiments and slightly converges when number of nodes exceeds 300.
6.3. EVALUATION

6.3.2 Live-VoD Neighbor Selection/Data Dissemination Performance

In this section, we evaluate our Live-VoD neighbor selection and data dissemination algorithms performance. We have measured the average, min and max time VoD nodes needed to download the complete video for systems of different sizes from 200 to 600 nodes. Our experiments show that Live-VoD is decentralized, scalable and self-organizing and even when more than half of the nodes in the system are VoDs (experiments 6.5, 6.6 and 6.7), all VoD nodes, regardless of their joining time, still manage to download the whole movie with no assistance from the streaming source. Figures 6.2, 6.3, 6.4, 6.5 and 6.6 show the average downloading time, the maximum and the minimum downloading time for VoD nodes in a Live-VoD network. Figure 6.7 summarizes all the results.

Figure 6.1. The average clustering coefficient against the number of nodes
Figure 6.2. Time taken to download the video against the number of nodes.

Figure 6.3. Time taken to download the video against the number of nodes.
6.3. EVALUATION

Figure 6.4. Time taken to download the video against the number of nodes

Figure 6.5. Time taken to download the video against the number of nodes
Figure 6.6. Time taken to download the video against the number of nodes

Figure 6.7. Time taken to download the video against the number of nodes
Chapter 7

Conclusions

7.1 Summary

In this report, we presented our system "Live-VoD" which is an integration of peer to peer Live streaming and peer to peer Video-on-Demand systems. We proposed a solution built on top of Gradient for Live-VoD overlay network construction which is an integration of Live overlay and VoD overlay resulting in Live-VoD overlay as Lives contribute to VoDs for downloading the data they need. We proposed four different algorithms for overlay construction and one algorithm for data dissemination. In our algorithms, we have used innovative and novel ideas, implemented them and through simulation showed how they make the integration of Live-VoD possible. Through our evaluations we showed that Live-VoD is scalable and completely decentralized. We measured the performance of Live-VoD using average clustering coefficient and average, min and max downloading time for both our overlay construction and data dissemination algorithms.

7.2 Future Work

One obstacle the P2P systems encounter is the partnership disconnection which is either caused by node departure/failure or occur as an impact of NAT or Firewall. Free-riding is also another well known problem which is the act of consuming more resources than contributing them, with some proposed solutions in \[16\] and \[22\] and \[21\]. In future, we plan to take free-riding, NAT and Firewall restrictions and network proximity into account.

Also, in our future work, we plan to optimize and test our algorithms for failure and churn scenarios as well. We plan to make modification and further optimizations on our algorithms to be able to measure playback continuity, playback latency and playback quality. This may need additional research on a suitable progressive downloading video format that is suitable for "Live-VoD".
Bibliography


BIBLIOGRAPHY


