Design and Implementation of Centrally-Coordinated Peer-to-Peer Live-streaming

ROBERTO ROVERSO

Licentiate Thesis
Stockholm, Sweden 2011
Abstract

In this thesis, we explore the use of a centrally-coordinated peer-to-peer overlay as a possible solution to the live streaming problem. Our contribution lies in showing that such approach is indeed feasible given that a number of key challenges are met.

The motivation behind exploring an alternative design is that, although a number of approaches have been investigated in the past, e.g. mesh-pull and tree-push, hybrids and best-of-both-worlds mesh-push, no consensus has been reached on the best solution for the problem of peer-to-peer live streaming, despite current deployments and reported successes.

In the proposed system, we model sender/receiver peer assignments as an optimization problem. Optimized peer selection based on multiple utility factors, such as bandwidth availability, delays and connectivity compatibility, make it possible to achieve large source bandwidth savings and provide high quality of user experience. Clear benefits of our approach are observed when Network Address Translation constraints are present on the network.

We have addressed key scalability issues of our platform by parallelizing the heuristic which is the core of our optimization engine and by implementing the resulting algorithm on commodity Graphic Processing Units (GPUs). The outcome is a Linear Sum Assignment Problem (LSAP) solver for time-constrained systems which produces near-optimal results and can be used for any instance of LSAP, i.e. not only in our system.

As part of this work, we also present our experience in working with Network Address Translators (NATs) traversal in peer-to-peer systems. Our contribution in this context is threefold. First, we provide a semi-formal model of state of the art NAT behaviors. Second, we use our model to show which NAT combinations can be theoretically traversed and which not. Last, for each of the combinations, we state which traversal technique should be used. Our findings are confirmed by experimental results on a real network.

Finally, we address the problem of reproducibility in testing, debugging and evaluation of our peer-to-peer application. We achieve this by providing a software framework which can be transparently integrated with any already-existing software and which is able to handle concurrency, system time and network events in a reproducible manner.
Acknowledgements

I am extremely grateful to Sameh El-Ansary for the way he supported me during this work. This thesis would not have possible without his patient supervision and constant encouragement. His clear way of thinking, methodical approach to problems and enthusiasm are of great inspiration to me. He also taught me how to do research properly given extremely complex issues and, most importantly, how to make findings clear for others to understand. In particular, I admire his practical approach in solving problems in an efficient and timely fashion given the very demanding goals and strict deadlines imposed by the industrial setting we have been working in. In this years, he has treated me more as a friend than a colleague/student and I feel very privileged to have worked with him and hope to continue doing so in the future.

I would like to acknowledge my supervisor Seif Haridi for giving me the opportunity to work under his supervision. His vast knowledge of the field and experience was of much help to me. In particular, I take this opportunity to thank him for his understanding and guidance for the complex task which was combining the work as a student and as employee of Peerialism.

I am grateful to Peerialism AB for funding my studies and to all the company’s very talented members for their help: Johan Ljungberg, Andreas Dahlström, Mohammed El-Beltagy, Nils Franzen, Magnus Hedbeck, Amgad Naiem, Mohammed Reda, Jonas Vasell, Christer Wik, Riccardo Reale and Alexandros Gkogkas. Each one of them has contributed to this work in his own way and has made my stay at the company very enjoyable.

I would also like to acknowledge all my colleagues at the Swedish Institute of Computer Science: Cosmin Arad, Joel Höglund, Tallat Mahmood Shaffat, Ahmad Al-Shishtawy, Amir Payberah and Fatemeh Rahimian. In particular, I would like to express my gratitude to Jim Dowling for providing me with valuable feedback on my work.

Special thanks go to all the people that have supported me in the last years and have made my life exciting and cherishable: Stefano Bonetti, Tiziano Piccardo, Christian Calgaro, Jonathan Daphy, Tatjana Schreiber to name a few and, in particular, Maren Reinecke for for her love.

Finally, I would like to dedicate this work to my parents Segio and Odetta and close relatives, Paolo and Ornella, who have at all times motivated and helped me in every possible way.
To my family
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>vii</td>
</tr>
<tr>
<td>I Thesis Overview</td>
<td>1</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Content Streaming</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Thesis Organization</td>
<td>6</td>
</tr>
<tr>
<td>2 Peer-To-Peer Live Streaming</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Tree-based</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1 Overlay maintenance and construction</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Mesh-based</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Hybrid Approaches</td>
<td>13</td>
</tr>
<tr>
<td>3 Problem Definition</td>
<td>15</td>
</tr>
<tr>
<td>4 Thesis Contribution</td>
<td>17</td>
</tr>
<tr>
<td>4.1 List of publications</td>
<td>17</td>
</tr>
<tr>
<td>4.2 Design and implementation of a centrally-managed peer-to-peer live streaming platform</td>
<td>18</td>
</tr>
<tr>
<td>4.2.1 Solving Linear Sum Assignment Problems in a time-constrained environment</td>
<td>19</td>
</tr>
<tr>
<td>4.3 NAT Traversal</td>
<td>19</td>
</tr>
<tr>
<td>4.4 Highly reproducible emulation of P2P systems</td>
<td>20</td>
</tr>
<tr>
<td>5 Conclusion and Future Work</td>
<td>21</td>
</tr>
<tr>
<td>Bibliography</td>
<td>25</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>II</td>
<td>Research Papers</td>
</tr>
<tr>
<td>6</td>
<td>On The Feasibility Of Centrally-Coordinated Peer-To-Peer Live Streaming</td>
</tr>
<tr>
<td>7</td>
<td>NATCracker: NAT Combinations Matter</td>
</tr>
<tr>
<td>8</td>
<td>GPU-Based Heuristic Solver for Linear Sum Assignment Problems Under Real-time Constraints</td>
</tr>
<tr>
<td>9</td>
<td>MyP2PWorld: Highly Reproducible Application-level Emulation of P2P Systems</td>
</tr>
</tbody>
</table>
Part I

Thesis Overview
Chapter 1

Introduction

Peer-to-peer (P2P) systems have shown a significant evolution since first introduced to the world by Gnutella [41] and Kazaa [27]. Nowadays, applications which use the peer-to-peer approach vary from illegal file sharing to distributing games updates. It is safe to state that Bittorrent in particular has been a major force in driving the bandwidth demands of most consumer networks throughout the last 5 years. It is estimated that in 2009 one fourth to one third of the Internet traffic was somehow related to P2P applications [20]. The consequences of an increased popularity of P2P platforms has coincided with efforts from the academia to try to understand how such an important amount of traffic influences the Internet and what can be done to reduce its congesting impact, in particular regarding Bittorrent [39][44].

The industry as well has applied the peer-to-peer approach to a number of areas, including VoIP, with Skype [18], distributed storage, with Wuala [19], and on-demand audio streaming, with Spotify [25]. All the aforementioned are attempts to provide services which do not involve significant costs from the point of view of bandwidth consumption and infrastructure.

P2P-based software amounts to only a tiny part of all Internet-based services. The bulk of the industry instead relies on expensive but reliable solutions, that are content delivery networks (CDNs) and Clouds. In particular, when considering video streaming, no commercial peer-to-peer software has been widely deployed.

However, a number of free applications such as SopCast [2] and PPLive [14] have proven very effective for large-scale live streaming, mostly because of their limited requirements in terms of bandwidth at the distribution site. In fact, most of the source of the streams in those systems are users which broadcast live content for thousands of others with limited upload bandwidth. On the other hand, the aforementioned applications provide a low quality of service which would not be acceptable in a commercial system.

Many solutions and different approaches have been proposed by the research community to the problem of streaming live video and audio content over the Internet using overlay networks. However, no consensus has been reached on which
1.1 Content Streaming

Streaming services can be classified in two main classes: video-on-demand and live.

**Video-on-Demand (VOD).** VOD allows users to select and watch pre-recorded video/audio material at any point in time. Users are usually presented with a catalog of streams to choose from; once a decision has been made, the stream is sent to the player as a flow of video/audio chunks. VOD allows for the delivery to start at any point of the stream. Seeking operations are also allowed.

VOD has an inherently sparse content popularity distribution. It is widely recognized that content follows a long tail, where the majority of the videos are not accessed very often, while few popular others are requested very frequently [8]. The complexity of VOD lies in guaranteeing the same quality of experience for popular and non-popular content items.

**Live Streaming.** The main difference of Live streaming compared to VOD is that the content is not pre-recorded. The stream instead is being created and broadcasted in real-time. As a requirement, every client receiving the live content should have the minimum possible delay from the moment the content becomes available at the distribution point, i.e. the streaming server, to the point when it gets played at the receiver’s end. A desirable feature is also to minimize the inter-client delay, i.e. the playback point should be synchronized or within a reasonable time window across all clients.

Live streaming is usually implemented using stateful network control protocols, such as RTSP [52], where clients establish and manage media sessions towards the streaming server by issuing VCR-like commands, e.g. play, stop and pause.

The media delivery is carried out using a transport protocol such as RTP [51], however proprietary alternatives are also common, e.g. RDT [1]. In standard live streaming, it is the server that pushes content fragments at a specific rate to the client following a single play request. At the transport level, standard live streams are delivered through UDP, while TCP is used for control messages.
CHAPTER 1. INTRODUCTION

Recently, the industry has introduced a new technology for live streaming, called HTTP-live. In HTTP-live, the stream is split into a number of small HTTP files, i.e. fragments. The streaming server appears to the clients as a standard HTTP server.

When a client first contacts the streaming server, it is presented with a descriptor file, called Manifest, which outlines the characteristics of the stream. It contains the stream’s fragments path on the HTTP server and the bitrate characteristics. As the content becomes available from an encoder or a capturing device, the streaming server creates new fragments and regenerates the Manifest accordingly. The player periodically requests a new copy of the Manifest to be aware of which fragments are available at the current time.

Reasons behind the development of HTTP live protocol are the simplicity of management at server side and the use of HTTP as a transport protocol, which makes it easier to achieve good connectivity in restrictive environments such as corporate networks. Examples of HTTP live protocols are Apple’s Live streaming [16] and Microsoft’s Smooth Streaming [17].

### Bandwidth Requirements.

Media streams are usually quite demanding from the point of view of bandwidth consumption. You Tube for example requires a minimum bitrate of 256Kbit/s for normal quality videos encoded with the H264 codec. With the same video compression format, it is possible to achieve a quality comparable to Digital Satellite TV at 1.5Mbit/s, whereas for an HD quality stream a minimum bitrate of 4Mbit/s is required.

The high bitrate requirements of video streaming raise obvious challenges. First, from the point of view of server infrastructure, since a single streaming server is typically able of handling just a few thousands of clients. And second, from the point of view of bandwidth consumption, because streaming requires a capacity of many Gbit/s towards the distribution site. Bandwidth capacity is by far the most expensive of the two aspects. Pricing as of Q4 2010 for streaming from a CDN is shown in Table 1.1.

<table>
<thead>
<tr>
<th>Volume</th>
<th>Max Price ($)</th>
<th>Min Price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50TB</td>
<td>0.45</td>
<td>0.40</td>
</tr>
<tr>
<td>100TB</td>
<td>0.25</td>
<td>0.20</td>
</tr>
<tr>
<td>250TB</td>
<td>0.10</td>
<td>0.06</td>
</tr>
<tr>
<td>500TB</td>
<td>0.06</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 1.1: Table showing the highest and lowest prices per acquired volume in the CDN market as of Q4 2010. Data taken from [40].

### Distribution Infrastructure.

Live and VOD streams are mostly distributed using unicast towards a single content source or a CDN. Multicast is also exploited.
Given that providers of steaming services are usually ISPs, the quality of the service is guaranteed by means of network resources reservation. Alternatives to the ISP approach include proprietary application-level solutions, such as Voddler, Netflix and Hulu. The delivery of audio and video streams in this case happens without any guarantee of quality of service or prevention of service interruption. Despite their best effort nature, these solutions have known a large amount of success in the last years.

Internet-based services use different delivery strategies for streaming:

- **Unicast**: Single End-to-End connectivity through either TCP or best effort UDP is implemented in this case. The load of multiple clients is usually shared among multiple locations of a Content Delivery Network. Server farms are placed in strategical geographical locations. Proximity to clients allow for lower distribution delays. CDNs are usually organized in a way to lower peering costs by placing servers with copies of the same content inside ASs and ISPs.

- **IP Multicast**: support for efficient broadcasting in this case is implemented at Network layer. Multicast is the most efficient way to deliver streams of content, since the distribution happens using a single stream of data along a tree-like network path. However, IP Multicast setup and configuration is cumbersome and requires expensive hardware.

- **Peer-To-Peer**: As opposed to the aforementioned strategies, the peer-to-peer approach allows to utilize hosts as relays for content streams. A client plays a double role: it receives the content data delivering it to the player, and it makes the data available to other peers. This approach allows for sharing the load of distribution among all involved hosts. Only few peers need to be receiving content from the source, whilst the others can retrieve it from them. Obviously, if the peer-to-peer delivery is organized in the right way, this can lead to significant savings in terms of bandwidth utilization at the distribution site and to improved scalability of service.

### 1.2 Thesis Organization

This chapter provides a general introduction to the thesis and the problem of content streaming. Chapter 2 presents the state of the art of peer-to-peer live streaming. A definition of the problems addressed in this work is presented in Chapter 3. The contribution to the defined issues is explained in Chapter 4. Section 4.1 provides the list of publications related to this thesis. Finally, Chapter 5 concludes the thesis and gives an insight of future directions of this work.
Chapter 2

Peer-To-Peer Live Streaming

Peer-to-peer content streaming can be viable solution to provide scalability and distribution savings. However, P2P live streaming is subject to a number of challenges. Typically, in any live streaming architecture, the source streamlines the content into a number of data chunks of fixed or variable size which need to be delivered to the player at fixed rate and within a strict deadline. Missing a deadline might cause loss of quality, temporary interruption or full termination of the playback. The behaviour of the content players, when such problematic events happen, can vary significantly according to the type of transport protocol, the codec used for encryption/decryption and quality of the stream.

Challenges

The main challenge for a peer-to-peer live streaming system is to meet real-time constraints while coping with dynamicity, e.g. churn, network congestion and connectivity limitations. On top of that, the application should strive for efficient bandwidth utilization. The peers in the overlay network should in fact contribute with their upload bandwidth in order to offload as much as possible the source of the content.

Churn, defined as the process and rate of peer joining and leaving the peer-to-peer network, is an important issue in peer-to-peer systems. This because nodes have very limited time to adapt to overlay network changes as deadlines for video/audio fragments are in the order of seconds rather than minutes.

Studies on peer-to-peer file-sharing systems have shown that node lifetimes follow an heavy-tailed Pareto distribution [6][50], where few peers have very long lifetime while all others have a very short one. In addition to that, the population size tends to remain stable over time.

In live streaming instead, churn behaviour is significantly different. In Figure 2.1(a) and Figure 2.1(b), we show the arrival and departure rates of a movie channel
broadcasted using the PPLive streaming platform\(^1\) during a 24 hours time span. As we can see, the join rate varies significantly during the day and peaks out at noon and late evening. Being this a movie channel, we observe batch peer departures at the end of each movie, i.e. every second hour. Figure 2.1(c) and 2.1(d) instead show the same type of statistics but for a TV channel. The arrival rate is very similar to the previous case, however, the departures trend is totally different. Figure 2.2 shows the lifetime distribution of the same system. As we can see, most of the users, both for the movie and TV channel, stay online for less than 1.5 hours. It is therefore clear that a steady amount of dynamicity should be expected in a peer-to-peer live streaming system at all times. In addition, flashcrowds and sudden departure of large number of peers are also observed.

Further source of disruption in a peer-to-peer network is congestion. Although it is believed that the last mile segment is usually the bottleneck along a network route, congestion can be experienced at different levels in the network. This is caused mainly by the fact that ISPs and ASs dimension their internal network and border links considering the average traffic, rather than the peak. This leaves room to possible congestion scenarios when most of the users are downloading significant amount of data, e.g. when accessing high definition live streaming events

\(^{1}\)Figures taken from "A Measurement Study of a Large-Scale P2P IPTV System" [15]
of particular interest. Effects of network congestion are: longer transmission delay, packet loss and blocking of new connections.

An additional requirement of a live streaming platform is that the stream must be received with a small delay from the source of the stream. Furthermore, users in the same geographical location should not experience significant difference in playback point with respect to their neighbours. These requirements tie directly into the meaning of live broadcasting. Guaranteeing low delay from the source is particularly cumbersome in peer-to-peer networks since, in most cases, the stream has to traverse multiple hops. In order to solve this issue, it is often necessary to introduce structure in the overlay such that the amount of hops can be kept under control.

Finally, a peer-to-peer live streaming application should strive for keeping the traffic inside a certain network segment, i.e. an ISP or an Autonomous System (AS) network. In fact, hosts which belong to a certain segment are likely to have low communication delay between them, whereas promoting intra-segment communication also lowers incurred peering costs for network operators.

There exists three main approaches to peer-to-peer live streaming, we detail them in the next sections.

### 2.1 Tree-based

The Tree-based approach aims to recreate the same network structure of IP multicast but with using an overlay network. The peers organize themselves in a tree whose root is the broadcast provider. The content is then pushed from the root along the tree by letting peers receive the video from their parents and then forward it to their children.

The typical structure of a system based on the tree-based approach is shown in Figure 2.3. Peers with highest upload bandwidth capacity are usually placed near the the source of the stream, in this case peers number two and three, and the others are positioned into new rows according to their available capacity in a decreasing order. Examples of tree-based systems are Overcast [22], Climber [34]
The main advantage of a tree-based overlay is that the distribution delay is predictable and corresponds to the sum of the delays along the overlay path. There exist a number of ways to construct a peer-to-peer tree given a set of available peers. Two major aspects must be considered when building such a structure: the depth of the tree and the fan-out degree of the internal nodes. For instance, as peers from lower levels in the tree receive content from others which are closer to the source, it is necessary to keep the number of rows in the tree to a bare minimum in order to avoid delays. As a consequence, the fan out degree of the internal nodes in the tree should be as large as possible. However, the fan out degree of a node is constrained by its upload capacity and it is therefore able to provide only to a limited number of peers.

In order to improve bandwidth utilization, a multi-tree structure can be used. In a single tree structure, the leaves of the tree do not contribute to the delivery. In a multi-tree configuration instead, the content server splits the stream in multiple sub-streams which are then broadcasted over separate overlay trees. As a consequence, a peer might be a provider of a certain sub-stream but only a receiver for another. Solutions using this approach are, for instance, SplitStream [7] and Orchard [30]. A study on a widely deployed system, i.e. GridMedia [61], has shown that using multi-tree based systems leads to near-optimal bandwidth utilization within a certain degree of churn. Figure 2.4 shows a multi-tree overlay network structure. In the example, two sub-streams are broadcasted by the streaming server along two trees, starting at peer 2 and 1.

Maintenance of the streaming tree is essential given the high dynamicity of peer-to-peer networks. When a peer abruptly leaves the overlay, all of its descendants get disconnected from the stream. In order to create room for the system to recover from a failure, a peer typically keeps a buffer. The buffer provides a way to compensate for disruptions in the delivery. Since the buffer itself introduces a delay from the point the data is received to the one it is provided to the player, its size is usually kept small. It is therefore very important for the system to be able to recover quickly from failures in order to avoid playback issues.
2.1.1 Overlay maintenance and construction

A tree-based overlay construction and maintenance can be carried out either in a centralized or decentralized fashion. We describe both of the methods in the next paragraphs.

Centrally-coordinated tree construction. In this case, peers rely on a central coordinator for instructions on which is the peer they should get the stream from. The central server keeps an overview of the system which includes characteristics of the peers and configuration of the tree at a certain point in time. Using this information, the central server makes decisions upon the join of a peer or a failure. For the purpose of load balancing, the central entity might enforce an explicit reconfiguration of the overlay based on some criteria. Load balancing operations may be carried out, for instance, to lower the number of rows in the overlay three or to improve efficiency.

The coordinator might become the performance bottleneck of the system, limiting scalability. This considering the challenging task of both providing quick reaction in case of churn and coping with overlay management complexity.

To the best of our knowledge, central coordination has been used exclusively for content distribution, e.g. in Antfarm [36], but never for live streaming.

In this thesis, we will explore this approach and report our experience and findings.

Decentralized tree construction. A number of distributed algorithms have been designed to construct and maintain a tree-based live streaming overlay. In this approach, peers negotiate by means of gossiping their placement in the tree using their upload bandwidth as a metric. Examples of systems using decentralized tree construction are: SplitStream [7], Orchard [30], ChunkySpread [57] and CoopNet [32].

2.2 Mesh-based

In mesh-based overlay networks no static overlay structure is enforced. Instead, peers create and lose peering relationships dynamically. Examples of systems using this kind of approach are: SopCast [2], DONet/Coolstreaming [62], Chainsaw [33], BiToS [58] and PULSE [37]. A typical structure of a mesh-based system is shown in Figure 2.5. In the pictured case, only few peers receive the stream from the source while the majority of them exchanges content chunks through overlay links, i.e. the black arrows in the picture. A mesh-based system is usually composed by three main parts: membership, partnership and chunk scheduling. The membership mechanism allows peers to discover others in the network receiving the same stream. This is usually achieved by means of a central discovery service where all peers report their address, e.g. the Bittorrent Tracker. Another way of discovering peers
is using gossip. Gossiping algorithms in this case are usually biased towards finding neighbours that have interesting characteristics, e.g. peers with similar playback point and available upload capacity [35] or that are geographically closer to the requester [23]. Upon the discovery of peers, the partnership service is used to establish temporary peering connections with a subset of peers which is considered suitable for the receipt of the stream.

A partnership between two peers is commonly established according to the following considerations:

- The load on the peer and resource availability at both ends. Possible load metrics include: available upload and download bandwidth capacity and CPU/memory usage.
- Network Connectivity. The potential quality of the link between the two peers in terms of delay, packet loss characteristics, network proximity and firewall/NAT configurations, as in [24].
- Content Availability. The available chunks at both ends, i.e. the parts of the stream which have been already downloaded and are available locally. Availability of chunks is advertised periodically.

Being the peer-to-peer network very dynamic, the partnership service continuously creates and terminates peering relationships. Control traffic generated by the partnership service is usually significant given that peers need frequent exchange of status information. On the other hand, less frequent updates might lead to increased latencies since pieces become available later. On top of that, a larger degree of partners gives more flexibility when requesting content, as peers have more choice to quickly change from a partner to the other if complications arise during the delivery. It is therefore important to find the right trade-off between partner set size and frequencies of updates.

The chunk scheduling service is entitled with the task of requesting content chunks as the delivery progresses, making use of information about the available peers collected by the membership service. Differently from the tree-based case, content chunks are not delivered through an already established overlay network path. In fact, a peer might be downloading different chunks from different peers in parallel. Chunks may then take different paths to reach a peer and consequently be delivered in an out-of-order fashion. In order to guarantee continuous playback, a peer keeps a buffer of received chunks and re-orders them before delivering them out to the player. The content of the buffer is usually what is made available to other peers. For this purpose, a peer might keep chunks available for a longer time.

Given their high peering degree and randomness, mesh-based systems are extremely robust both against churn and network-related disruptions, such as congestion. Since no static structure is enforced on the overlay, a peer can quickly switch between different providers if a failure occurs or the the necessary streaming rate cannot be sustained. That said, since every chunk is treated as a separate delivery unit, per-packet distribution paths and delivery times are not predictable.
and highly variable. Consequently, it is very challenging to design a mesh-based streaming platform able to guarantee playback deadlines.

Another drawback of the mesh-based approach is sub-optimal bandwidth utilization of provider peers since chunk requests are mostly unpredictable.

2.3 Hybrid Approaches

A tree-based approach can be combined with a mesh-based to obtain better bandwidth utilization and lower delays. mTreebone[59] elects a subset of nodes in the system as stable and uses them to form a tree structure. The content is broadcasted from the source node along the tree structure. A second mesh overlay is then built comprising both of the peers in the tree and of the rest of the peers in the system. For content delivery, the peers rely on the few elected stable nodes but default to the auxiliary mesh nodes in case of a stable node failure. The drawback of this approach is that a few stable peers might become congested while the others are not contributing with their upload bandwidth. Thus, this solution clearly ignores the aspect of efficient bandwidth utilization.

As an alternative approach, CliqueStream [3] creates clusters of peers using both delay and locality metrics. One or more peers are then elected in each cluster to form a tree-like structure to interconnect the clusters. Both in PRIME [29] and NewCoolsteaming [26], peers establish a semi-stable parent-to-child relationship to combine the best of push and pull mesh approach. Typically a child subscribes to a certain stream of chunks from the parent and the parent pushes data to the child as soon as it becomes available. It has been shown that this hybrid approach can achieve near-optimal streaming rates [38] but, in this case as well, no consideration has been given to efficient bandwidth utilization.
Chapter 3

Problem Definition

In this work, we target the design and implementation of a live streaming platform which allows for large amount of bandwidth savings at the source of the stream and high quality of user experience.

In order to achieve the aforementioned goal, we argue that a number of important challenges must be addressed:

- **Efficient Overlay Management.**
  The system must be carefully crafted to enable management of peers in a way to enable high upload bandwidth utilization while coping with real-time streaming deadlines. User quality of experience must be guaranteed at all times, even if bandwidth savings need to be sacrificed for that purpose. As a consequence, low initial buffering time, small playback and distribution delays should be maintained throughout a streaming session. A further requirement on the platform is to preserve network locality of streams in order to limit peering costs for operators.

- **Scalability.**
  The proposed system should scale to thousands of users. In order to achieve this, obvious bottlenecks should be resolved with targeted solutions or by providing an alternative design of the platform.

- **Circumventing NAT limitations.**
  In peer-to-peer systems deployed on consumer network, it is common to observe a large amount of participating hosts behind NAT, typically up to 70% [47]. Obviously, limited accessibility to peers translates in limited efficiency of the system. In order to overcome this limitation, a number of NAT traversal techniques exist in order to enable connectivity even when peers are behind two different NAT boxes [10][53][42][43]. However, in our experience, their effectiveness is limited in the case direct connections must be achieved between peers behind different NAT boxes. When a direct connection fails to be established, content is simply relayed through other peers [21]. This is not feasible
when considering streaming systems where the amount of data to be relayed is extremely large. Research on NAT limitations has not known significant advancements in the last years. This, even though NAT constraints are one of the most relevant issues in peer-to-peer systems, for the simple fact that good connectivity is a precondition for any distributed algorithm to function correctly.

- **Deterministic Testing/Debugging/Evaluation Environment.** Arguably, when designing large-scale systems, it is infeasible to cover all possible runtime scenarios with pure reasoning. For that reason, prototyping of peer-to-peer systems is often conducted in a controlled environment, such as Discrete Event Simulator. One of the main advantages in using a controlled environment is that, depending on the platform, it is possible to achieve partial reproducibility of executions, i.e. the ability to execute the same experiment many times while preserving the exact same sequence of events, at least network-related ones. This brings obvious advantages when it comes to debugging of code and detailed inspection of the application’s execution. A number of platforms have been developed to allow reproducibility of network events, both at kernel, e.g. [55], and application level, e.g. WiDS [28], ProtoPeer [11] and SSFNet [60]. However, to the best of our knowledge, no solution has been developed to manipulate operating system concurrency and thus enabling a fully deterministic application execution.
Chapter 4

Thesis Contribution

In this chapter, we summarize the contributions of this thesis. First, we list the publications that were produced as a result of this work. After that, we provide a small summary of each publication’s contribution.

4.1 List of publications


4.2. DESIGN AND IMPLEMENTATION OF A CENTRALLY-MANAGED PEER-TO-PEER LIVE STREAMING PLATFORM

List of publications of the same author but not related to this work


4.2 Design and implementation of a centrally-managed peer-to-peer live streaming platform

The work describing the design and implementation of our live streaming platform has been published in a conference paper [49]. The paper appears as Chapter 6 in this thesis.

In our work, we adopt the approach of a centrally coordinated overlay in order to provide significant amount of savings while coping with churn and limitations on connectivity of the underlying network. Our contribution lies in showing that the realization of a system based on the central coordination approach is indeed feasible.

In this work we present a design where a central coordinator organizes the overlay network by issuing direct instructions to peers. Our results show that this approach leads to efficient delivery of streams by letting the central entity help peers in their provider’s choice, thus avoiding the time consuming trial-and-error process typical of fully decentralized solutions.

The main challenge we faced was to design an efficient decision engine which can provide directions on how to organize the overlay in a very short period of time. If the population of the overlay network varies considerably due to churn while the decision engine is running, then its decisions might be of no use or even detrimental to the system performance. A decision is based on a snapshot of the overlay network status based on the information periodically sent by the peers to the central coordinator.

The decision process is composed of three steps:

I Row construction. Peers are placed in subsequent rows according to their available upload bandwidth. Peers with highest bandwidth capacity are placed in the row closest to the source of the stream so that they can provide to others. The rest of the rows are filled with peers with decreasing amount of upload bandwidth. During this process we make use of an heuristic based on the max-flow approach [13] in order to guarantee connectivity compatibility between peers in consecutive rows, in particular due to the presence of NAT.

II Input definition. Every assignment between a peer $a$ in an upper row and another $b$ in a lower row is given a certain score and placed in an assignment matrix. The score depends on the following metrics: inter-peer delay, stickiness (connections that already exists are favoured), playback buffer matching, NAT compatibility probability and ISP friendliness.
III Assignment problem solving. In an effort to provide the best possible set of interconnections between peers in the overlay, we model peer assignment between consecutive rows as an optimization problem, namely as a Linear Sum Assignment Problem. We make use of an heuristic called Deep Greedy Switching (DGS) which provides fast outcomes while attaining near-optimal solutions. In practice, we have seen it deviate by less than 0.6%. The input to the optimization engine is the assignment matrix produced in step II.

Once an outcome of the decision has been achieved, orders are issued to the peers and the overlay re-organized accordingly. The process is repeated periodically or as need arises, i.e. when disruptions are observed on the overlay.

4.2.1 Solving Linear Sum Assignment Problems in a time-constrained environment

Another contribution of this thesis is the parallelization of the Deep Greedy Switching heuristic algorithm and its implementation on Graphic Processing Units (GPUs). This work is motivated by the fact that we identified the optimization engine to be the bottleneck of our peer-to-peer live streaming system. In fact, large instances of the Linear Sum Assignment problem need to be solved in order to couple peers of consecutive rows to each others. Sequential implementations of the DGS solver, although much faster than any other similar software using algorithms with optimal outcome, such as for the Auction algorithm [5][56][31], fell short of our needs in terms of time complexity. Given the time-constrained nature of the application, the solver must provide an outcome in a matter of seconds rather than minutes. This work was published in a conference [48] and can be found in Chapter 8 of this thesis. The conference paper details the parallelization of the original Deep Greedy Switching algorithm and the evaluation of the new heuristic’s implementation on GPUs using the NVIDIA CUDA language [12].

The solver is very generic and can be used outside of our streaming platform’s context for solving any instance of the LSAP problem.

4.3 NAT Traversal

The work on NAT traversal has been published in a conference paper [47] and appears in Chapter 7 of this thesis. In this work, we classify behaviours of Network Address Translators recently discovered by the community in 27 different types. Given this set of behaviors, we tackle two fundamental issues that were not previously solved in the state of the art. We provide a study which comprehensively states, for each of the possible 378 combinations, whether direct connectivity, without relaying of traffic, is possible or not between NAT types. The outcome is the first contribution of this work. In addition, we define a formal model of NAT which helps us describe how each type behaves when allocating new ports. This is the second outcome of our work. As a third contribution, we define through
an augmentation of currently-known traversal methods which techniques should be used to traverse all pairwise combinations of NAT types. This scheme is validated by reasoning using our formalism, simulation and implementation on a real test network.

4.4 Highly reproducible emulation of P2P systems

The work on application-level emulation has been published in a workshop [45] and appears as Chapter 9 of this thesis. In this work we describe our application-level emulator which has been designed to tackle the problem of reproducibility for development and testing of an existing Peer-To-Peer application. The contribution in this case is a novel platform which can provide highly transparent emulation which can be used for reproducible debugging, testing and evaluation of an already-developed peer-to-peer software.

We achieve transparency by injecting custom implementations of standard/widely-used APIs, such as Apache MINA, for networking, and Java Executors, for concurrency. Calls made to those APIs are redirected to a Discrete Event Simulator adapted for the purpose of emulation.

A further contribution of this work is an implementation of an accurate flow-level bandwidth model, based on the progressive filling algorithm [4], which is used to mimic transfers duration between peers according to the configured upload/download capacity of the emulated nodes.
Chapter 5

Conclusion and Future Work

In this Chapter, we present conclusions for this thesis work. Conclusion are described separately for each problem area that we addressed. Finally, we present some of the future work related to this thesis.

Design and implementation of a centrally-managed peer-to-peer live streaming platform

In this thesis, we have reported our efforts with addressing the issue of bandwidth savings and user experience in live streaming using a central coordination approach. We employed a design which models the issue of assigning peers to each other as a linear sum optimization problem (LSAP). Our main concern with scalability of the solution has been addressed using a fast LSAP heuristic called Deep Greedy Switching.

We further improved on the performance of DGS by parallelizing the heuristic’s algorithm and implementing the resulting algorithm on commodity GPUs using the CUDA language. We show in our results, how the solver is able to handle 10,000 problem instances in less than 3 seconds. The solver is generic and can be used to solve any instance of LSAP given that a tolerance of 0.6%, or less than, on the outcome optimality is tolerated.

The merit of our live streaming approach is that it provides high savings at the distribution site in real-world scenarios where realistic delay, packet-loss, bandwidth emulation and NAT constraints are modeled in the experiments. Our results also show that, in such difficult scenarios, the system is able to maintain low initial buffering time and playback delays.

Concerns on scalability for overlays which require the assignment engine to scale over 10,000 can be addressed by either providing specialized GPU processing hardware, e.g., GPU racks, or by partitioning the overlay into multiple decision engines. An alternative approach is to manage a backbone of nodes with central coordination and let swarms of peers create along that backbone.
NAT Traversal

In this work, we proposed a comprehensive analysis of what combinations of NAT types are traversable using direct connectivity without relaying connections. We have defined a formal model for describing NAT behaviors. Using that model, we covered all possible NAT combinations describing which traversal techniques are applicable for each combination.

With formal reasoning, we have shown that about 80% of all possible combinations are traversable. Our experimental results however, show that only 50% of all possible combinations are encountered in reality, or at least in our test network. Of those, 79.4% are shown to be traversable with various degrees of success, which are also presented in our study. Data on the time needed for the carrying out traversal strategies is also reported in our findings.

It is the first time that such a comprehensive study, including all new findings in Network Address Translation behaviors, has been conducted to provide a clear understanding on NAT traversal both from a formal point of view and from experimental results on a real deployed network.

Highly reproducible emulation of P2P systems

We have provided our experience in developing an emulation framework which can be used for debugging, testing and evaluation of an already-existing peer-to-peer software. By inspecting the state of the art, we found that we could not find a tool which simultaneously satisfies the following requirements: minimal changes to the software under test, ease of deployment and high reproducibility.

Our approach in developing the platform was to adopt application-level emulation but with ensuring that high reproducibility is met. We achieved this by controlling concurrency, system time and all network related events.

The resulting framework called “MyP2PWorld” has been used for a number of months for testing our live streaming peer-to-peer application. It has resulted in huge improvements in software quality and bug discovery rate. It became an integral part of our development process.

While we have initially developed MyP2PWorld to complement the implementation of a specific software, we are now trying to make it available for other peer-to-peer software developers to use.

Future Work

As future work, we would like to investigate using central coordination or other types of peer-to-peer approaches with HTTP live streaming.

HTTP-live streaming is significantly different from other streaming protocol in the way that requests for chunks of the content are completely unrelated between each others.
The streaming player at the end of the distribution chain decides which segment to request and at which bitrate. The time of the request is also not deterministic. This approach greatly simplifies the complexity on a unicast system, where the distribution site is composed of one or more standard HTTP servers. However, it poses significant issues when trying to efficiently organize the delivery of the content over a peer-to-peer overlay, since no assumptions can be made on which fragment of the content will be requested next.

As far as we know, there exist no current peer-to-peer system which provides support for HTTP-live.

Related to the work conducted in this thesis, we would like to extend our NAT traversal findings by deploying our solution on a global scale to conduct further tests and, if possible, improve our approach. We also would like to implement a connectivity library to be used in any peer-to-peer application and which transparently handles traversing of NAT boxes.

As an additional feature of the connectivity library, we would like to include support of different traffic prioritization policies. This would allow peer-to-peer applications to tweak the desired level of QoS. In the case of live streaming for instance, traffic generated by the peer-to-peer application should have priority over all other transfers happening at the same time on the host. We think this is achievable using dynamic congestion control mechanism, such as MulTFRC [9] or LEDBAT [44], over UDP. We are not aware of any other attempt of including general traffic prioritization for peer-to-peer applications at application-level.

We are currently in the process of transforming the platform called MyP2PWorld to add support for a component model and event driven runtime. This would allow for faster prototyping and easier testing of single part of the application. The event runtime should also provide a greater degree of scalability and better execution semantics when it comes to concurrency.

We are also considering improving scalability of the bandwidth model’s implementation used in MyP2PWorld.
Bibliography


Part II

Research Papers