Live Streaming in Overlay Networks

Mário Rui Vazão Vasco Ferreira
mvvf@gsd.inesc-id.pt

Instituto Superior Técnico

(Advisor: Professor Luís Rodrigues)

Abstract. Overlay networks have emerged as a promising technology to support the dissemination of information on the Internet, including multimedia streams with quality of service requirements. This report presents a survey of the main strategies to implement live-streaming in large-scale overlays and identifies some possible lines of research to improve the current state of art.

1 Introduction

Live streaming applications are emerging in the Internet. Due to the lack of widespread of IP Multicast support, the Peer-to-Peer (P2P) paradigm became an attractive alternative to build this kind of applications, supporting a large number of participants connected by self-organizing overlays. In P2P systems, the available resources such as upload bandwidth, CPU cycles, or data storage capacity, scale with the demand, because interested users are also participants in the system and, increasing the system participants also increases the available resources.

The key mechanism to perform live streaming to a large number of participants is multicast. Several approaches have been proposed to support multicast in P2P systems. Some approaches focus on building specialized structures to support data dissemination while others employ different techniques, such as flooding or distance vector routing, to perform the multicast operation. While the former adds complexity in overlay management to maintain the structure, the latter offers potential to better cope with the underlying topology allowing the overlay to explore more efficient routing paths.

In an Internet environment, there is a huge heterogeneity in the resources available at each node. These differences affect live streaming systems in the sense that nodes cannot contribute equally to support the system load. In consequence, some nodes may not be able to deliver stream data on time. Thus, recent research efforts have attempted to develop bandwidth-aware multicast mechanisms that can account for network heterogeneity. While some systems focus on efficiency, by providing low delay dissemination mechanisms, others focus on balancing the system load through all participants in a bandwidth-aware fashion. However, few systems try to combine both features.

This work addresses the problem of managing bandwidth heterogeneous capacity in P2P live streaming systems, focusing in combining load balancing mechanisms, distributing the system load according to the resources available to each participant, simultaneously maintaining efficient data dissemination mechanisms.

The remaining of this report is organized as follows. Section 2 describes the goals and motivation for this work. Section 3 surveys previous research, covering different information dissemination algorithms for P2P systems. Section 4 presents the proposed architecture. Section 5 describes how we plan to evaluate the proposed techniques. A schedule of future activities in provided in Section 6. Finally, Section 7 concludes the report.

2 Goals

This work addresses the problem of disseminating multimedia streams with quality of service requirements in P2P systems. More precisely:

> *Goals:* This works aims at analyzing, designing, and evaluating algorithms to efficiently support live streaming in large-scale overlays supporting heterogeneous bandwidth capacity participants.

Our approach to tackle this problem is based on from the observation that most bandwidth-aware systems sacrifice efficiency in order to balance the system load. Our proposal combines bandwidth-aware mechanisms with efficient tree-based broadcast algorithms, balancing system load while maintaining the efficiency of the data dissemination solutions that are employed in other systems. In summary, we aim at achieving the following results:

> *Expected results:* This work will: i) design and implement live-streaming algorithms and; ii) provide an evaluation of these algorithms based on PeerSim[1] simulations.

We will also try to perform some experiments on the PlanetLab[2] platform.

3 P2P Streaming

This section presents an overview of the existing related work on supporting the dissemination of information with quality-of-service constraints in large-scale peer-to-peer systems. Although our goal is to support live-streaming, for completeness, we also describe some solutions that are focused on the video-on-demand and multicast support with soft real-time constraints.

The remaining of this section is organized as follows. Section 3.1 presents three representative classes of bandwidth-consuming applications. Section 3.2 compares the two main approaches to perform multicast. Section 3.3 identifies the main challenges and performance metrics and Section 3.4 presents the main design choices concerning live streaming. Section 3.5 presents the selected systems and, finally, Section 3.6 concludes the section.

3.1 Types of Bandwidth-Consuming Applications

Some applications depend on remote data from the network to fulfill its operation. This applications have bandwidth requirements and are called bandwidth-consuming applications. They can be divided into three representative classes[3], namely: *Fixed Rate*, *Required Rate* and *Elastic Rate*. These classes of applications differ on how content is generated and consumed, and on the quality of service metrics used to assess their performance.

– In *Fixed Rate* applications, data content is generated at a fixed rate r and distributed on the fly. There are strict time requirements for content distribution, because peers should receive the streamed data at the same rate r. A quality of service metric for this applications is the download continuity, which means that the download rate should always be equal to r. An example of such application is live-streaming[4, 5].
– In *Required Rate* applications, the content is available at a subset of nodes and it should be downloaded by the other nodes at a minimum rate r. Similarly to *Fixed Rate* applications, this restriction is imposed to ensure continuity, but unlike in the previous class, content is already completely available a priori, and can be cached or stored until it is consumed. Therefore, the quality of service metric can be expressed as: a node must download content at a rate greater or equal than r. A typical example of this class of applications is Video-on-Demand (VoD)[6, 7].
– In *Elastic Rate* applications the content is also initially available at a subset of nodes, but there are no constraints on the minimum download time or rate. The quality of service metric is the download time, which should be minimized. This kind of applications includes file-sharing applications [8–10].

3.2 IP Multicast vs Application-Level Multicast

There are two main approaches to multicast data to a potentially large set of recipients in the Internet, which main difference is the layer at which they are implemented. IP Multicast[11] operates at the IP-layer while Application-Level Multicast (ALM) [12, 13] operates at the application-layer.

IP Multicast Since 1990, multicast mechanisms at the IP-layer have been proposed[11]. By supporting multicast at IP level it is possible to prevent the same data from crossing any given link twice. It it also possible to ensure that packets follow almost IP-optimal paths. Therefore, IP multicast has the potential to achieve good performance and efficient network consumption. However, the deployment of IP multicast has been limited due to a number of drawbacks of the technology. First, it requires routers to keep per group state which, in addition to violating the "stateless" principle of the original IP design, leads to scalability limitations. Moreover, IP Multicast, similarly to IP unicast, is a best effort service. Unfortunately, providing high level guarantees such as reliability, congestion control, flow control, or security in multicast systems, is much more challenging than providing such guarantees for unicast communication. Additionally, IP Multicast imposes infrastructural changes, slowing down the deployment phase. Recently, some effort has been made to overcome this difficulties; in [14], the authors attempt to provide a new approach for IP Multicast deployment which uses routing protocols that already exist but, with a loss of efficiency because the proposed solution increases bandwidth requirements.

Application-Level Multicast Because of the aforementioned deployment issues of IP Multicast, P2P systems have emerged as an alternative to support multicast service using decentralized and scalable solutions. These solutions typically create an *overlay network* among the participants in the system, namely among the content providers and all the recipients, which is later used for media/data dissemination. Given that the overlay supports the communication among application level entities directly, it facilitates the development of high level mechanisms to enforce reliability, congestion control, etc, that are difficult to introduce at the IP Multicast level.

3.3 Challenges and Performance Metrics

Building P2P streaming solutions and applications is not an easy task. Internet is an heterogeneous environment, where most nodes and links have different characteristics. To build efficient and scalable P2P streaming solutions these differences need to be considered in order to meet applications requirements. We now present the main challenges and performance metrics that need to be considered when designing these solutions.

Latency The latency of the multicast is a function of the sum of the latencies introduced by each link in the path from the source to the recipient. Ideally, the multicast service should forward messages using the links with the lowest latency, if possible. In turn, the latency of each overlay link depends on the properties of the underlying connection (including the maximum throughput, the physical distance among the endpoints, and the network congestion). Latency can be estimated in runtime using multiple Round-Trip Time (RTT) measurements[15].

Bandwidth Bandwidth is typically measured in bits per second, and captures the capacity of a system to transmit data from the source to the destination. In most P2P systems, where many nodes are connected to the Internet via ADSL or cable connections, the participants bandwidth is typically limited and asymmetric: nodes have much larger download bandwidth than upload bandwidth. Limited upload bandwidth needs to be managed to efficiently disseminate data while preserving application-level requirements.

Link loss Streaming systems using UDP as the transport protocol need to deal with packet losses. TCP can mask omissions at the network level but at the cost of increasing the communication overhead which limits the resource management at the streaming protocol level. Furthermore, in live-streaming applications, it is often preferable to deal with the loss of information than to introduce a lag in the delivery flow to recover a packet[4, 16]. Link losses may also add complexity to the maintenance of the overlay network. For instance, failure detectors [17, 18] may report a high number of false positives if a large number of control messages (such as heartbeat messages) are dropped.

Churn Any distributed application needs to deal with the failure or abrupt departure of nodes. This is also the case in P2P streaming applications. For instance, when a node fails, several other nodes will be affected as the failing node could be in charge of forwarding data to them. In large-scale P2P systems, failures may be common, not only due to the size of the system but also because participants may leave shortly after joining the system. A long sequence of join and leave / failures at very high pace in P2P system is a phenomena that is known as *churn*[19].

Free-Riding P2P systems are designed to exhibit the best performance when all participants cooperate in the execution of the distributed algorithm. However, it is possible that a selfish user deviates itself from the algorithm, such that it is able to receive content without contributing to its dissemination. Nodes that avoid to contribute to the data dissemination are known as free-riders. Therefore, a P2P streaming system must incorporate appropriate incentive and punishment mechanisms to mitigate or eliminate this behavior[20, 21].

It is worth noting that is often impossible to optimize the performance of the system with regard to all of the challenges and performance metrics listed above. For instance, there is usually a trade off between the latency and bandwidth consumption.

3.4 Main Design Choices

This section discusses the main design choices in the implementation of live-streaming P2P systems. It identifies relevant design choices analyzing the advantages and disadvantages of existing design options.

3.4.1 Single vs Multiple View Many algorithms are designed to support the dissemination of a single multimedia stream. These systems are called *single view* [4, 22, 16, 23, 24, 5] systems. Naturally, it is possible to use such systems to disseminate multiple streams, by running multiple *independent* instances of the algorithm, one for each multimedia stream. This approach, however, may suffer from scalability issues.

On the other hand, it is possible to design a system in such a way that it is prepared to disseminate multiple multimedia streams. These systems are called *multi-view* systems[25, 26]. The advantage of having an integrated multi-view system is that it has the potential to achieve a better resource usage, for instance, by implementing schemes that promote resource sharing and load balancing. Multi-view systems are of particular importance to support IP-TV, because a user usually watches multiple TV channels[27].

3.4.2 Centralized vs Decentralized Approaches A centralized streaming system is a system where a unique node is responsible for transmitting the stream directly to all recipients. These solutions have the appeal of simplicity and may work in small-scale scenarios while suffering from obvious scalability problems.

In fully decentralized approaches[28, 12, 13], the source is not required to interact directly with all recipients, as the recipients coordinate to distribute the load of disseminating the data among themselves. Such solutions have the potential to overcome the scalability issues of centralized solutions.

For scalability, solutions only requiring local knowledge are favored, i.e., each individual participant should not be required to have global knowledge of the system. For instance, it may not know the total number of participants, their location, the bandwidth of all links, etc. Instead, it only has a partial view of the system, in particular, of its neighbors in the overlay[29–31].

Naturally, there are optimizations that cannot be achieved only with local information. Due to this fact, some systems[25, 4] adopt an intermediate form that combines both approaches, using a decentralized structure for data dissemination, but relying on a centralized component to facilitate or perform control operations, such as coordinating the join and leave of participants.

3.4.3 Structured vs Unstructured Approaches There are two main approaches to build an overlay to support message dissemination: *structured* and *unstructured*.

- Structured overlays assume a regular topology, imposed by construction. They were initially designed to support distributed lookup services, but have also been used to disseminate data in an efficient manner. Example topologies include rings[32], hypercubes[33] and Plaxton Meshes[34, 35]. The disadvantage of these overlays is the overhead required to maintain the structure during join and leave operations. This can lead to a poor performance when experiencing high levels of churn.
- On the other hand, unstructured overlays, typically named meshes, impose little or no specific topology to the system. Therefore, there are not as many constraints as in a structured overlay, enabling these overlays to better cope with network dynamics being more resilient to churn. Unfortunately, it is harder to design efficient dissemination schemes over unstructured overlays. While some systems[12] use unstructured overlays to run a distance-vector routing protocol[11], the simplest approach consists in flooding the overlay, which in theory allows to reach all participants with a high cost due to the high communication redundancy imposed by this approach. In practice, the redundancy of the flood procedure may congest the network causing additional packet losses that may impair the reliability of the streaming protocol.

As the name implies, tree-based models forward the multicast messages along a spanning tree. This tree defines a single parent/father and a set of children to every node. To disseminate a message, a node receiving a message from its father simply has to forward it to all its children.

Tree-based approaches are not easy to include in a single classification mentioned above. In fact, they can be classified as both structured or unstructured approaches depending on how the tree is constructed.

In [4], tree management is held by the stream source which also manages all filiation operations (nodes joining and leaving), imposing a specific structure to the overlay topology. Thus, the approach is structured because it forces the overlay to assume a regular structure by construction.

Other approaches[28, 36] store in each node routing information to eliminate unnecessary redundant messages and, at the same time, choose the more efficient path to route data streams. These approaches do not impose a specific organization by construction; instead they implicitly allow a loose structure to emerge in the overlay[37], which can adapt easily to changes in the membership and, at the same time, provide efficient resource usage.

Tree-based approaches are perform better when there is a unique stream source, for which the tree is optimized. Multiple sources can be addressed by using a shared tree, which will result in sub-optimal efficiency. Another alternative is to create multiple trees, having each tree optimized for each different source. However, there are scalability limits to the number of sources that can be supported simultaneously as the tree maintenance cost will grow linearly with the number of sources. Multiple trees can also be used to overcome network packet losses, sending the stream or a sub-stream through each tree. This increases the probability of the stream being received at all destinations, even in the presence of packet losses or node failures. When experiencing node leaves or failures, the tree-based models must recover the tree ensuring that none of the receivers becomes disconnected.

3.4.4 Single vs Multiple Overlay Another choice in the design of P2P streaming systems is to use an unique or multiple overlays to disseminate data. Although the use of multiple overlays is more intuitive in the case of multi-view systems[25, 26], it is possible to use a single overlay to disseminate multiple streams and also to use multiple overlays to disseminate a single stream. The latter approach may be useful, for instance, if the the stream is decomposed in multiple sub-streams, so that the quality of the stream delivered to the application is dictated by the number of received sub-streams. In this case, each sub-stream may be disseminated using a different overlay, and participants with limited download bandwidth could join only a few overlays and still receive the required data.

3.4.5 Push vs Pull-Based Forwarding There are two main strategies to control the data flow in an overlay topology. If the control is held by the node sending the data, i.e., the node that decides when and where it forwards the data to, the strategy is called *push-based.* On the other hand, if the node that receives the data has the control, i.e., it decides from when and where it requests the data from, the strategy is called *pull-based.*

Eager push-based schemes forward messages to their neighbors as soon as they are received. Thus, data is propagated as fast as the network allows. The drawback of this approach is that, without the proper coordination, two or more nodes may forward the same message concurrently to the same node, which creates unnecessary (and, most of the time, undesirable) redundancy.

There is a variant of the push-based strategy that circumvents this effect named *lazy-push.* In lazy-push, the control is still held by the node that has the data. However, instead of sending the message payload immediately, it sends a control message to notify the potential recipient, which, in turn, indicates if the message is new or redundant. Doing so, a node will only request messages it has not received yet, preventing duplicates. Push-based schemes are suitable for systems where upload bandwidth is limited, because the sender has a better control of the number of messages it forwards.

In a pull-based strategy, the recipient has to contact potential senders, asking for new content. Therefore, it is possible that a node requests a message from one of its neighbors and fails to obtain the required message (because this neighbor has also not received it yet). This model can be effective when the download bandwidth is limited because receivers can control which packets to download and avoid message collisions.

3.4.6 Bandwidth-aware vs Bandwidth-oblivious Many P2P streaming systems assume that all nodes have similar capacity and, therefore, try to distribute the load evenly among all participants, i.e., bandwidth-oblivious systems. However, in more realistic scenarios, is more common that different participants may have very distinct capacities, including available CPU and bandwidth. A system that is able to take the capacity of individual nodes into account when distributing the load is named *bandwidth-aware*[36, 22]. For instance, nodes with faster network connections can contribute more for message dissemination than low bandwidth nodes.

Bandwidth-aware systems must also address the problem of managing the competition for the available bandwidth.In a scenario where downstream peers need to be supplied from multiple upstream peers, bandwidth competition, which consists in managing the available upload bandwidth to supply all the receivers, needs to be solved in a fair way for all participants. One interesting approach to solve this problem is through the use of game theory[38, 39], by modeling the bandwidth competition problem as an auction game.

Finally, a bandwidth-aware system should strive to avoid congesting links in the overlay, as a congested link presents higher delays and packet drop rates which can lead to scenarios with decreased streaming quality.

3.4.7 FIFO vs Topology-Aware Chunk Scheduling Chunk scheduling is related to the policy that decides which packets are exchanged in each communication step. In a system where there is a single path from the source to the recipient, the most used chuck scheduling policy is FIFO; i.e., packets are forwarded by the order they have been received (which is related to their production order in the case of live streaming). However, when there are multiple paths between a node and the source, more sophisticated scheduling schemes may be used to improve load balancing among links while reducing message redundancy. For instance, a node with two paths to the source could receive half of the packets from one path and the other half from the other path by employing a suitable scheduling policy.

3.4.8 UDP vs TCP Another important decision when building a P2P streaming system is the choice of the transport protocol, in particular between TCP or UDP.

TCP is an attractive option because of the reliability mechanisms it offers. Besides, TCP can also be used to detect neighbor failures and lower the time the system takes to reconfigure itself to recover from these failures. Finally, TCP embeds congestion control mechanisms and is favored over UDP by Internet Service Providers and carriers. On the other hand, TCP is more expensive, in terms of bandwidth, than UDP and, additionally, the occurrence of omissions results in additional delays in data delivery by the receiver, due to TCP recovery mechanisms design, which may introduce undesirable glitches in the stream. Furthermore, UDP is more lightweight than TCP, however it lacks reliability and is often penalized by Internet providers.

An interesting approach explored in [16] is to used both protocols. This system uses UDP for fast data dissemination, and TCP for control message exchange and failure detection.

3.4.9 Stream Spliting vs Network Coding When performing data streaming, it may be desirable to split a data stream in several sub-streams and send each through a different path. This approach is useful to improve the resiliency to packet losses, adding data redundancy in the sub-streams, and also to perform load balancing by spreading the forwarding effort through different nodes.

The most simple approach to obtain such sub-streams is to divide the main data stream by distributing the packets that form the stream using a specific policy. A simple policy to create two sub-streams could simply be to assign half the packets to one sub-stream and half to another.

Another way to obtain the sub-streams is by using network coding techniques to encode the stream data, adding redundancy and allowing the receiver to decode a stream with only a subset of the encoded packets. This is mostly used to improve the resilience of the system to failures, by allowing nodes to, partially or completely, rebuild the data stream using only a fraction of the disseminated data. Multiple Description Coding[4] is a form of network coding that encodes the source stream into layers. A peer receiving a single layer is able to decode the original stream but the quality increases with the number of layers used in the decoding. Such mechanisms are useful to provide robustness in a P2P streaming scenario with unreliable network links and nodes.

3.5 Selected Systems

In this section, we describe a selection of relevant systems, from simple application-level multicast approaches to multi-view, multi-overlay scalable streaming systems.

3.5.1 Narada Narada[12] is an unstructured approach for application-level multicast. In this protocol, a mesh-overlay is built among participating end systems in a self-organizing and fully-distributed manner.

In Narada, every node needs to maintain knowledge about the whole group membership. To ensure this, each node uses the mesh to periodically propagate refresh messages to its neighbors with increasing sequence number. To reduce the communication overhead of probing the whole membership, nodes only exchange refresh messages with their neighbors. Each refresh message from node i contains the address and the last sequence number i has received in a refresh message from any other node.

When a new participant wishes to join the overlay, it contacts a list of existent nodes and connects to them. During a leave event, a node informs all its neighbors with a message that is propagated through all the group members. Failures are detected locally and propagated to the rest of the members. Failure detection consists of a node watching if a certain neighbor refresh message was not issued during a certain time. If a timeout is exceeded, the node assumes that its neighbor has failed.

The mesh is periodically optimized, dropping some links to add some others according to a utility function and the expected cost of dropping a link. The utility function measures the performance gain of adding a new link, while the cost, which depends on the number of paths that are routed through the respective link, evaluates the utility of the link to be dropped. For the sake of stability, an exchange is only performed if the associated gain is above a given threshold.

Data delivery is performed with the support of multicast trees. These trees are built over the mesh overlay using a reverse shortest path mechanism between each recipient and the source. A member M that receives a packet from source S through a neighbor N forwards the packet only if N is in the shortest path from M to S. This is verified based on the information exchanged in the refresh messages.

The multicast approach of Narada provides efficient data dissemination thanks to the optimized trees which are constructed over the mesh-based overlay. In the presence of multiple sources, several trees must be constructed whereas each tree is optimized to a single source which offers an adequate substrate for multi-source multicast. One limitation of Narada is the need to maintain the information of the whole group membership in each node, limiting its scalability.

3.5.2 Scribe Scribe[13] uses a structured approach to Application-Level Multicast. Scribe is built on top of Pastry[34], a generic P2P object location and routing substrate overlay for the Internet. Pastry imposes an overlay organization among the participants, according to an unique identifier generated for each node.

Pastry is fully decentralized and can locate a node in the structure with $log(N)$ complexity, where N is the total number of participants. Each node also needs to maintain routing information about $log(N)$ nodes only. Scribe uses Pastry to manage group creation and to build a per-group multicast tree used to disseminate multicast messages in the group. All decisions are based on local information, and each node has identical responsibilities. As a consequence, both Scribe and Pastry are scalable.

Scribe allows distinct multicast groups to co-exist. This groups are associated to different interest topics that user can subscribe. Each group, similarly to the Pastry participants, has an unique *groupId* and the node with *nodeId* closest to the *groupId* acts as the rendez-vous point, which means it is the root of the group multicast tree. Alternatively, Scribe allows the creator of the group to become the rendez-vous point by setting the *groupId* according to the respective *nodeId*. This alternative can be useful to optimize performance in situations where the creator will send often to the group. Both alternatives use only group and node names to generate the identifiers and do not require any name service.

The multicast tree is set during group join operations using a scheme similar to reverse path forwarding. A node sends a $JOIN$ message to the rendez-vous point, and all the nodes which forward the messages update the set of children for the respective group with the message source identifier. The $JOIN$ message is forwarded until it reaches a node which is already a forwarder for that group or to the rendez-vous point of the group. When a node performs a multicast, it needs to locate the rendez-vous point which will perform the data dissemination process. However, the rendez-vous address is cached to enhance the performance of subsequent multicasts.

Scribe provides a structured overlay to efficiently disseminate multicast data. It divides the load of the nodes through all participants using the Pastry substrate. However, it assumes all nodes have the same characteristics. On the other hand, Scribe forces nodes which are not interested in a given group to be forwarders if they are part of the routing path between any receiver and the rendez-vous associated with the group.

3.5.3 PlumTree In[28], the authors propose PlumTree, an integrated approach combining epidemic and deterministic tree-based broadcast strategies. PlumTree is a tree-based approach but does not impose a specific structure to its participants. Instead, the protocol embeds a minimum delay broadcast tree in an unstructured gossip-based overlay, maintaining it with a low cost algorithm. The tree emerges from the combined use of eager and lazy push gossip strategies.

The protocol selects eager push links so that their closure effectively builds a broadcast tree embedded in the overlay. Lazy push links are used to ensure reliability, when nodes fail, and to quickly heal the broadcast tree. The membership protocol used is symmetric, simplifying the task of creating bi-directional trees. Initially, all links use eager push, resulting in a flood-like broadcast. When a node receives a duplicated message, it marks the source link as lazy push and informs the correspondent node to perform the same operation. In the presence of failures, lazy push links are used both to recover missing messages and to heal the tree. If a neighbor fails, a fact detected by the membership protocol using TCP connections, the node will change one of the lazy push links to eager push. Upon the reception of a message, there is an optimization step that verifies if there is a lazy link that is performing better than the eager push link through which the message was sent. In this case, the roles of both links are switched.

Combining eager and lazy push strategies allows the system to recover quickly from failures, and prevents messages from flooding the network when the system is stabilized. Compared to Scribe[13], PlumTree does not balance the load between nodes. In the absence of failures, the forwarders of the multicast messages are always the same. Also, opposed to Narada[12], with a single multicast tree data dissemination is optimized for a single source. If consecutive multi-source multicasts were performed, a single tree would be optimized to none of them. The protocol allows the usage of multiple trees, each optimized for one source, but the complexity of managing more than one tree limits the number of possible multicast sources in the system.

3.5.4 Adaptive Chunk Selection The authors of [40] propose a distributed adaptive algorithm to enable a server to notify all peers about an optimal chunk policy. In opposition to the previous presented approaches, the policies referred in [40] use a pull-based model, i.e., the receiver makes the decision of which chunk will be exchanged. The policies manage a limited size buffer in each peer. In more detail, a chunk policy determines which chunk will be downloaded to the buffer by the next pull request of each peer. The protocol also assumes that a peer can download chunks directly from the main server or from another peer.

To evaluate the best policy, a priority-based chunk selection model is used to represent a large policy family. This model assumes a priority value associated to each buffer position. The highest priority can be assigned, for instance, to *chunks with lower sequence number*, downloading the chunks immediately needed to play the video (*greedy strategy*), or to *chunks with higher sequence number*, downloading the rarest chunks (the last reaching the peer buffer). The use of similar priorities for all chunks is equivalent to the use of a random policy. This priority model allows the definition of $(n-2)!$ different policies, where n is the size of the buffer (see Fig. 1), because the positions 1 and n are reserved for the next chunk downloaded from the main server and the next chunk to be consumed by the video player, respectively.

Evaluation results show that the optimal chunk policies are $\vee$-shaped, this means that, considering $B(k)$ the buffer cell with lowest priority value, priority increases as the position moves away from k. On the other hand, worst chunk policies present the inverse behavior named $\wedge$-shaped, meaning that, considering $B(k)$ the buffer cell with highest priority value, priority decreases as the position moves away from k. Another important observation is the fact that the optimal

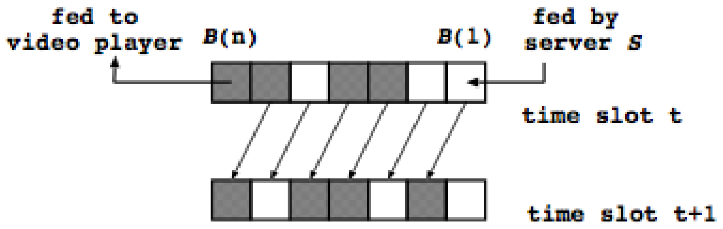

Fig. 1. Buffer structure B with n = 7, server S fills B(1) at time slot t.

policies vary according to the fraction of downloads a peer accomplishes using the main server, f(opposed to the fraction of downloads accomplished through other peers, $1 - f$). f depends on the main server upload capacity (C) and the number of receivers (M). The experiments described in [40] point to the conclusion that the optimal policies are greedier when the value of f increases. The rationale that explains this result is that with higher values of f the server has additional available bandwidth capacity to serve the peers, resulting in low chunk scarcity. Thus, peers should download those chunks which have earliest playback deadlines.

The distributed adaptive algorithm proposed in [40], decides which policy to be used based on f, which can be calculated by the quotient between the main server known capacity C and the number of receivers M. To estimate M, all join and leave operations are performed through a *Tracker* according to a DHT structure (similar to Chord[32]). The *Tracker* is responsible for maintaining M updated, for generating the IDs, and determining the neighborhood for new nodes. Peers exchange *keep-alive* messages with their neighbors to detect failures.

When M changes, the new policy is determined accordingly and peers are informed using a gossip protocol.

Under high churn rates, this approach may result in many policy changes, leading to a high message overhead due to the gossip mechanism used to update the peers with the new policy. Despite this, the authors present an evaluation of the existent chunk selection policies for pull-based communication model.

3.5.5 Most Deprived Peer, Latest Useful Chunk

Tree-based designs are able to achieve close-to-optimal rates in real-life conditions. The authors of [23] propose a mesh-based communication strategy called DP/LU, which intends to prove its competitive quality against tree-based solutions. This approach differs from the one employed in Narada[12] because it does not build a dissemination tree over the mesh. Instead, the data is pushed to the node's neighbors according to a chunk scheduling strategy.

DP/LU intends to overcome the deficiencies presented by another chunk policy called DP/RU. DP/RU stands for *Most deprived peer, random useful chunk* and selects the neighbor which has received less chunks from the current node and sends it a random *useful* chunk. A *useful* chunk is a chunk which has not been sent to that peer yet. Despite being analytically shown that this policy is rate-optimal, experiments using PlanetLab[24] proved that it could not achieve high rates in real world deployment.

DP/LU, which stands for *Most deprived peer, latest useful chunk* uses the same mechanism as DP/RU to choose the neighbor but sends the useful chunk with higher sequence number. Simulations show that this strategy provides better results than the previous approach and supports the usage of mesh-based overlays to perform live streaming.

3.5.6 Bandwidth-aware Clustering In [5], an approach to minimize the maximum end-to-end streaming delay in *mesh-based overlay networks* is described. Previous presented systems and algorithms assume an uniform node degree and a single-commodity flow for each receiver. These assumptions are not appropriate for P2P streaming, where peers have heterogeneous bandwidth capacities and need to aggregate multiple flows for a smooth playback.

The approximation algorithm proposed by the authors of [5] clusters nodes according to a parameter λ, based on the optimal delay value. Nodes at range λ from the source form the first cluster. Then, a *representative node* is selected from these nodes. The selected node is the one with most residual bandwidth capacity. It is used to form another cluster with some of the unclustered nodes, following the same process. This process is repeated until all nodes are clustered.

The inter-cluster links resulting from the process described above are virtual. They represent the aggregated inter-cluster stream instead of the actual flows to the representative nodes. The real links are decided depending on the bandwidth capacities of the nodes in the cluster and can be divided across several participants.

The distributed version of this algorithm relies on a rendezvous point to maintain the list of the *representative nodes* to manage join operations and to support stream coordination.

This system presents an interesting approach to define clusters in a streaming overlay using bandwidth-aware mechanisms.

3.5.7 Unstructured Multi-source Multicast In [36], the authors propose UMM, which stands for Unstructured Multi-source Multicast. This system is an attempt to use unstructured overlays to provide efficient multi-source multicast. As in [5], the system is bandwidth and heterogeneity aware.

Similarly to Narada[12], this approach builds a base dense overlay where efficient distribution trees are embebbed using a flood mechanism. During this process, a node which receives a duplicated message from a certain source informs the respective forwarder not to send more messages from that source through that link. This link filtering mechanism has the ability to eliminate the least efficient links. To detect network performance changes, this information is treated as soft-state and, after a certain timeout, links need to be filtered again. This link filtering mechanism has also similarities with the approach taken in

PlumTree[28]. The main difference is that, in PlumTree, the optimization step uses a reactive strategy to accommodate link performance changes, responding quicker than the cyclic strategy employed by this system. On the other hand, the filtering mechanism used in UMM filters messages per-source, allowing the system to optimize data dissemination for different sources, simultaneously.

Each node in UMM has half of the links labelled as short and half labelled as long. The definition of short and long depends on the latency of the link and a threshold value. During overlay optimization rounds, a node selects a small subset of peers which it knows and to whom it is not directly connected, and evaluates the potential link it may establish with these nodes. Short links are optimized for latency while long links are optimized for bandwidth. Additionally, the number of neighbors a node supports, and thus the node degree, which affects the load imposed to each participant, is proportional to its upload bandwidth capacity.

To avoid node disconnection from the overlay, heartbeat messages are broadcasted from multiple sources for failure detection. A node determines that it is disconnected from the overlay if it receives less than half of the heartbeat messages from all sources.

This system proposes a simple unstructured approach for application-level multicast and a mechanism for supporting source optimization for multi-source multicast. The link filtering mechanism information expires periodically, forcing nodes to return to a flooding strategy and stabilize again.

3.5.8 ChunkySpread ChunkySpread[41] is a tree-based approach for streaming multicast. It is an approach to distribute load over a tree-based multicast approach using multiple distribution trees. It divides the stream into M stripes and sends each one through a different multicast tree. Each node has a target and a maximum load, both expressed as a number of stripes equal or greater to zero.

This system sets the node degree proportional to its target load. The source of a multicast has M neighbors (M being the number of slices of a stream) and sends a sub-stream to each neighbor. Each of these neighbors will be the root of a multicast tree.

The propagation graph is constructed using weighted random walks with support for statistical control over nodes degree. This means that each node can control the probability with which it is chosen by the random walks.

In this system, nodes try to replace overloaded parents with underloaded nodes. Once a node is satisfied with its neighbors concerning their load, it switches its behavior to optimize latency. Nodes identify the targets for a possible switch to improve latency using the relative time it takes for them to receive the slices of the stream.

ChunkySpread focus, primarily, in optimizing its trees concerning peers load leaving link latency improvement in second plane. Therefore, this approach may lead to sub-optimal paths.

3.5.9 CoopNet The authors of [4] address the problem of distributed live streaming media content to a large number of hosts. To address this problem, they designed CoopNet, a system which attempts to build multiple short trees, providing a node with as many neighbors as its bandwidth can support. The multiple tree approach allows the system to balance the load of forwarding stream packets through all participants.

As in ChunkySpread[41], data dissemination is performed using multiple trees. The difference is that tree management is centralized at the root node. Trees can be built using one of two methods: *Randomized Tree Construction*, which picks one leaf node with spare bandwidth to support a child, randomly; and *Deterministic Tree Construction*, builds diverse trees where a node is only an interior node in one tree. In the *Deterministic Tree Construction*, new nodes are inserted in the trees with less interior nodes.

One novelty of CoopNet is the use of *Multiple Description Coding(MDC)* to provide data redundancy needed for robust P2P media streaming. MDC separates a stream in sub-streams such that the quality of the received data depends on the number of the received sub-streams. Each sub-stream is sent using a different tree. A participant receiving a single sub-stream can display the media content with less quality.

One drawback of this approach is the assumption that only the root can send data and that it holds all the tree management information. However, the tree diversity concept can help the system to distribute the multicast load among all nodes more fairly.

3.5.10 Bandwidth-Aware In [22] a Bandwidth-Aware(BA) system is proposed. The fundamental idea behind the design of this system is to deliver chunks of a stream first to peers that can contribute more to the chunk dissemination, based on their upload capacity.

The authors of [23] proposed a similar strategy that operates only at the source level. In BA peers with high upload capacity have higher probability of being chosen to deliver a chunk, not only by the source but also by any node that forwards a chunk. To make this decision, peers need to know their neighbors capacities. To support this, a signaling mechanism is employed.

BA also adjusts the neighborhood cardinality according to each node upload capacity. Notice that is an inherent trade-off between the number of neighbors and the signaling overhead.

The algorithms proposed in [22] describe an interesting approach to allocate the heterogeneous bandwidth capacity of a P2P system. The use of probabilistic approach ensures that peers with greater upload capacity will often receive data first but does not undervalue completely nodes with lower capacities. However, the use of a push-based probabilistic model allows low capacity nodes to become starved. Combining the two push and pull-based as in PlumTree[28] is a possible strategy to mitigate this issue.

3.5.11 Divide-and-Conquer

Multi-view P2P live streaming systems have recently emerged. This type of systems enables users to simultaneously watch different channels. In [25], the authors propose a protocol for multi-view P2P streaming that can efficiently solve the inter-channel bandwidth competition problem. The proposed solution is flexible enough to incorporate any streaming protocol. The main design goals of this system were: flexibility, efficiency and scalability.

The Divide-and-Conquer (DAC) system ensures the scalability goal by dividing overlapped overlays in different independent logical overlays, one for each channel. Assigning each channel a different overlay, allows the system to solve the inter-channel competition at the channel-level opposed to approaches which solve the problem at the peer level. DAC does not have intra-channel requirements allowing any streaming protocol to be used which allows to achieve the flexibility goal.

To provide efficient bandwidth allocation, a mathematical model, based on an utility function, is employed to determine each channel's bandwidth allocation over each overlay link. Executing this model requires the measurement of system information, which is supported by a few sampling servers that are also part of the system and are responsible to statistically capture relevant information, and periodically report it.

The system implements a solution to handle multi-channel streaming. It claims to meet the scalability requirement using a different overlay for each subscribed channel. This means that a peer will have to manage as many overlays as the number of channels it is receiving. It could become difficult to handle this configuration with many channels.

3.5.12 Channel-Aware Peer Selection

In [26], the authors propose a Channel-Aware Peer Selection (CAPS) algorithm for P2P MultiMedia Streaming (P2PMMS) applications. This algorithm goal is to optimize peer resource allocation in multi-view streaming systems.

In single-view systems there are two main bottlenecks. First, the bandwidth bottleneck, which occurs when a neighbors have limited upload bandwidth to supply that peer. This issue can be solved by constructing an efficient overlay. Secondly, the content bottleneck, which occurs when neighbors have sufficient upload bandwidth but they do not have the requested content. The content bottleneck can be mitigated with adequate and efficient chunk scheduling policies.

A multi-view peer is a peer that belongs to more than one overlay and, consequently, receives more that one channel stream, increasing the flexibility of its upload bandwidth allocation. So, joining peers would benefit from setting all their connections with multi-view nodes, but this would result in two new problems. The first one is unfairness, due to the fact that early joining peers would consume most of the available multi-view peers. The second problem concerns the fact that nodes would cluster around multi-view peers building an overlay that could be quite fragile to multi-view peers leaves or failures. To solve these problems, the authors propose a new channel-aware peer selection algorithm for multi-view systems.

CAPS distinguishes peers with different types, in relation to the number of watched channels, in different groups. A joining peer chooses its neighbors based on a matrix which sets a higher probability of choosing neighbors from groups with more spare bandwidth. To do this it is required to estimate the peers total upload bandwidth capacity.

The probability mechanism used to select peers ensures the neighbor diversity, providing a good mechanism to obtain a balanced overlay.

3.5.13 Strategies of Conflict in Coexisting Streaming Overlays In [38], the authors propose game theory based algorithms to solve bandwidth competition in coexisting overlays. Coexisting overlays are used in IP TV systems with multiple channels where, usually, each channel is transmitted through a distinct overlay. The authors of [38] propose decentralized strategies to resolve conflicts among coexisting streaming overlays.

The proposed approach uses game theory concepts, and models the upload bandwidth sharing problem of a node, called the seller, between its peers, called the players, as an auction game. In each round, players place their bids to the sellers they know about. A bid consists of a 2-tuple with the values of the requested bandwidth and the price which the player is willing to pay for it. The sellers job is to analyze the bids and choose the ones that carry out the most profit. To do this, in an auction game, the seller chooses the bid with highest price and allocates the required bandwidth for that player limited by the available bandwidth it still owns. If it still has available bandwidth after this allocation it repeats the process for the remaining bids.

A player is allowed to bid multiple sellers. Its objective is to allocate the required bandwidth it needs. When a player receives a stream from a seller, there are two kinds of costs associated. First, the streaming cost, which is expressed as a function of the requested bandwidth and represents the streaming latency actually experienced by the player. Second, the bidding cost, which is expressed by the bid made to seller and represents the degree of competition and demand for bandwidth in the auction games. A player should place its bids in order to obtain the required bandwidth, minimizing the cost function which is expressed by the sum of both the referred costs.

A player also adjusts its bidding price according to the previous bidding round. It starts with a bidding pricing of 0. When the upstream peer allocates its bandwidth, it sends the results to the players and, if the player has not been fully supplied it increases the bid prices, otherwise decreases it.

In order to achieve a prioritization mechanism, the authors set a limited budget and a player bidding cost cannot exceed this budget. With this mechanism, overlays with higher budgets will perform better achieving higher streaming rates. This approach is useful on IPTV systems to enhance, for instance, the quality of premium channels.

This approach offers a set of strategies to share an upstream peer bandwidth between a set of downstream peers in a fully decentralized way. The proposed algorithms are targeted for multiple overlay bandwidth competition scenarios, and they provide an interesting solution for multi-view systems which face such competition issues.

The proposed strategies tackle the problem of bandwidth sharing in multi-overlay scenarios but do not account for lossy or congestioned links. If these parameters were incorporated within these strategies, auctions could make a better use of available upload bandwidth.

3.5.14 CollectCast In [16], the authors propose CollectCast, a P2P streaming system that assumes a multiple-to-one streaming model. In CollectCast each receiver of a stream gets data from multiple sources in order to achieve the required download rate needed for continuous video playing.

CollectCast service is layered on top of a P2P lookup substrate, and it is independent of the chosen substrate. The lookup service is required to return a candidate set of sender peers for a certain media file. The authors present tomography-based techniques[42] that use inferred network topology information and select peers from the candidate set accordingly. This approach can detect situations where end-to-end measures are not accurate. For instance, if paths from two senders to the receiver overlap in the same physical link, it is not possible to receive at a rate equal to the sum of both paths due to the common physical link which is a bottleneck. This approach has considerable overhead. The peer selection method chooses the most appropriate peers for the streaming session and the remaining candidate set is simply left in stand by mode.

In CollectCast a receiver uses lightweight UDP channels to receive stream data but it also owns a reliable TCP channels for exchanging control messages. The control channels are used to detect node failures and stream flow degradations. When both situations occur, the respective peer is replaced with the best candidate in standby, according to the peer selection method.

CollectCast makes an effort to maintain a high quality level of the received stream, by setting multiple sources to feed a single receiver. The approach taken is not applicable for a system where every peer is both a receiver and a sender at the same time, as in P2P live streaming applications. However, some of the choices made by the authors could enhance the quality of the received streams in those scenarios as well. The multiple-to-one communication models used by CollectCast also fits nicely on mesh-based overlays.

3.5.15 Dynamic Video Rate Control In [43], the authors propose Dynamic Video Rate Control (DVRC), a receiver-centric protocol resilient to congestion and adaptive to network packets losses. The protocol attempts to maintain the video streaming desired smoothness, avoiding at the same time the significant damage usually caused due to network congestion.

DVRC is an extension to the UDP protocol. It employs acknowledgments not to perform retransmissions of lost packet but to estimate the RTT and loss rate values. There are two different strategies to adjust a stream rate: additive and multiplicative. Additive adaptation has the advantage of performing smoother changes while multiplicative adaptation responds faster to overloading situations.

The authors discuss two approaches to adjust the received stream rate according to experienced load and both employ an additive increase to perform a smooth increase on the received rate when the experienced load is under the target value. The difference remains on the decrease strategy:

Additive Increase Additive Decrease (AIAD): When the receiver is overloaded AIAD reacts slowly, using an additive strategy, resulting in a poor adaptation to critical situations.

Additive Increase Multiplicative Decrease (AIMD): AIMD approach employs a multiplicative decrease, reacting faster, but it can cause some unnecessary instability when random losses occur, because of its aggressive adaptation strategy.

To overcome these problems exhibited by both these approaches and to and get the best of both worlds, DVRC employs an hybrid strategy called Additive Increase Additive Multiplicative Decrease (AIAMD). This approach uses the receiving rate history to decide when to use an additive or multiplicative rate adaptation. To maintain the receiving rate history, the protocol updates a moving average value at each received rate measure. The authors divide the elapsed time in epochs. An epoch is the elapsed time between two lost packet events. At the end of each epoch the experienced received rate is measured and the rate moving average is updated. If the difference between both values exceeds a certain threshold, multiplicative adaptation is performed to quickly recover from congestion. Otherwise, the additive strategy is used in order to provide a smooth adaptation.

The proposed protocol is aimed at client-server systems. However, the approach could be extended to perform congestion control in P2P streaming applications. Assuming multiple sources, the main issue would be to define a strategy for choosing which peers to increase or decrease the exchanged rates.

The protocol's goal is to maintain the receiver load under a target value, but at the cost of decreasing the received rate, resulting in a lower stream quality. Again, in P2P systems, this issue could be solved by taking advantage of the flexibility in changing the stream sources.

3.6 Summary

In this Section, we presented some of the most significative approaches to support P2P multicast operation discussing the impact of their design choices. An interesting observation that steams from this overview of related work is that unstructured approaches such as PlumTree, which deduces dissemination trees from the data flows, are able to combine the natural resiliency of such approaches with some of the efficiency of structured approaches.

Systems which deliver content in a bandwidth-aware fashion typically organize the participants according to their bandwidth capacities which, usually, results in situations where data is delivered through non-optimized paths. Tree-based approaches are able to disseminate data using optimized and efficient paths, however they allow situations where some nodes become overloaded in the dissemination tree. To address this problem many systems adopted the multiple tree design. This design requires a mechanism to split the original streams in sub-streams which are disseminated using different trees.

We also presented some streaming systems intended for single-receiver scenarios, which do not address live streaming issues, but identify some details which may enhance data communication of those systems. These systems address issues of the used communication protocols and adaptation of the stream download rates during overloading situations that may be useful in live streaming scenarios.

4 Proposed Architecture

The proposed architecture aims at building a live streaming system based on PlumTree[28]. PlumTree was not designed to support live streaming but, as stated before, it constructs a minimum delay dissemination tree that emerges from the experienced data flows, according to the properties of the links that form the underlying unstructured overlay network that supports PlumTree.

In Section 4.1, the relevant properties of PlumTree that justify its choice as the basis of this work are discussed. Section 4.2 identifies the main challenges that must be overcome to enable the proposed architecture to support live streaming and Section 4.3 presents the mechanisms that can be employed to mitigate the identified challenges.

4.1 PlumTree Properties

Analyzing the PlumTree design, three key qualities can be identified:

Quick Failure Detection PlumTree is built on top of a membership protocol that uses TCP as the transport protocol (HyParView[29]). On the presence of participant crashes, TCP provides a fast failure detection mechanism that permits the dissemination tree to be quickly repaired.

Efficient data dissemination The dissemination tree constructed by PlumTree contains the lowest delay paths to reach all participants. The tree links are selected from the experienced overlay traffic. Therefore, PlumTree provides the best dissemination paths possible from the underlying unstructured overlay network.

Tree adaptation and healing PlumTree eager push/lazy push hybrid strategy allows the system to easily adapt to link performance changes and participant crashes. The system maintains a set of lazy push peers as backup to serve as data providers when eager push peers cannot deliver data on time.

This strategy also enables the protocol to detect and recover from tree partitions and eager push peers failure as the mechanism to recover lost messages implicitly promotes a lazy push peer to a eager push peer.

4.2 PlumTree Challenges

Despite the stated properties, PlumTree also presents several challenges in adapting itself to support live streaming:

Load Distribution When the system is in steady state, traffic pattern is deterministic always following the same paths. This approach may overload nodes that support a large number of the links that form the dissemination tree, while the remaining ones may have available resources and do not contribute to support the system load of forwarding packets. Besides, it is not likely that a single peer has the required capacity to supply its children with a constant streaming rate[16].

Congestion The communication model of PlumTree does not control the streaming rates forwarded by a peer. A peer can suffer from congestion if the forwarding load required exceeds its capacity. This can slow down or even disrupt the overall data dissemination.

Communication Protocol The use of TCP to send packets has substantial overheads that might impair data dissemination. This fact complicates the usage of PlumTree to perform live streaming.

4.3 Proposal

Our proposal consists in rebuilding the algorithms that compose the PlumTree protocol to provide low delay and constant rate streaming. To accomplish this we divide the original stream in several *slices* and forward each slice through a different dissemination tree as in [4].

The multiple trees are extracted from a random overlay network using the PlumTree eager/lazy push strategy. For instance, a possible way to build such trees would be to use a larger partial view (i.e. degree) at the unstructured overlay network level and use a gossip mechanism which uses a lower fanout f when disseminating the messages used to build each tree. This approach avoids the flood used by PlumTree while promoting the diversity of the extracted trees, increasing the probability of obtaining variable per *slice* trees. In this way, each tree only explores the lowest delay paths within the f neighbors. To mitigate this limitation, an optimization step could be used that, periodically, attempts to replace one of the f current neighbors with a more efficient one from its partial view.

The dissemination trees are maintained using PlumTree's eager/lazy push strategy. With this configuration, it is possible to limit the number of *slices* forwarded by a node avoiding overloading situations. When a node refuses to forward a *slice* through the lowest delay path, PlumTree protocol will use its eager/lazy push strategy to forward it using the next lowest delay path.

To accomplish the described solution we propose the following mechanisms:

Stream Splitting This mechanism is responsible, at the stream source, for the partition of the incoming stream in sub-streams packets by labeling each packet with an identifier of the respective sub-stream and, at the receiver level, by joining the stream packets from distinct sub-streams together to reconstruct the incoming stream. Hence, since there are several approaches to split the stream, we propose a Streaming Layer which intends to abstract the mechanism employed allowing the integration and evaluation of different mechanisms for this purpose.

Load Limit Each node has a maximum capacity value expressed as the number of slices multiplied by node degree and the upload capacity it can afford. When the node reaches its maximum load (i.e. it is forward a number of slices equal to its maximum load value) it stops forwarding new incoming slices to its children. This will force PlumTree eager /lazy push strategy for forward packets through the next lowest delay path.
This mechanism prevents a node to become overloaded, forcing data to be forwarded through different paths distributing more fairly the forwarding load across participants. In addition, PlumTree eager/lazy push communication strategy will always try to forward data using the lowest delay paths.

Congestion Avoidance To avoid node congestion, we propose a mechanism that allows a node to refuse the reception of a *slice*. This is useful, for instance, when a node has not the available download capacity to receive the total stream. In that case, the reception of all stream *slices*

Hybrid Communication Protocol To overcome additional delays and overhead imposed by the use of TCP in streaming protocols, we plan to combine the use of both UDP and TCP, as in [16]. UDP will be primarily used for transmitting the streaming data while TCP will be employed to support the transmission of control messages and also to operate as an unreliable failure detector.

5 Evaluation

The evaluation of the proposed architecture and communication strategies employed will be performed experimentally, building a prototype. It will be mainly tested through simulation experiments. If time permits, it will be also tested in a close to real scenario using PlanetLab[2] nodes.

5.1 Metrics

The main properties that will be measured in the evaluation are performance, reliability and resiliency of data dissemination, and the load balancing strategy.

To measure these properties, the following metrics will be used:

Last Delivery Hop (LDH) This metric was used in PlumTree[28] and measures the number of forwarding steps required to deliver a broadcasted stream packet to all its recipients. The LDH is the maximum number of hops a message must be forwarded in the system in order to be delivered.

Latency The latency measures the elapsed time to deliver a multicast message to a receiver and expresses the playback delay experienced by that peer.

Load The load metric is defined by a node allocated upload bandwidth expressed as a number of stream *slices*. This metric also shows the contribution of the participant to the overall system.

Reliability The reliability is expressed as the percentage of active participants which receive a certain stream packet.

Healing Time This metric measures the time needed for the system to heal the dissemination structure after failure events.

5.2 Methodology

The evaluation phase will be divided in several steps:

- Evaluation of the impact of the bandwidth-aware strategy. This will be achieved by comparing the performance metrics of the prototype with the original baseline, PlumTree.
- Evaluation of the benefits of the bandwidth-aware approach by overloading the overall system and measuring the individual load at each participant.
- The reliability property of the system will also be evaluated, measuring the reliability metric experienced when a percentage of participants fails.
- Using the same scenarios of the reliability measurements, the resiliency of the system will be evaluated by determining the healing time metric.

6 Schedule

Future work is scheduled as follows:

- January 9 - March 29: Detailed design and implementation of the proposed architecture, including preliminary tests.
- March 29 - May 2: Perform the complete experimental evaluation of the results.
- May 3- May 23, 2009: Write a paper describing the project.
- May 24 - June 13: Finish the writing of the dissertation.
- June 14, 2009: Deliver the MSc dissertation.

7 Conclusion

In this report we have surveyed the most representative approaches to support live-streaming in large-scale P2P systems.

We analyzed the main design choices which need to be considered when building P2P streaming systems and their trade-offs. Several systems have been presented to illustrate how the state of the art solutions address these choices. We have also identified which metrics that are relevant to evaluate these kind of systems.

This report also proposes an architecture to develop a streaming system combining efficient data dissemination with bandwidth-aware load balancing.

We concluded the report with a description of the methodology to be applied in the evaluation of the proposed solution.

Acknowledgments I am are grateful to João Leitão, Maria Couceiro and João Alveirinho for the fruitful discussions and comments during the preparation of this report. This work was partially supported by project "Redico" (PTDC/EIA/71752/2006).

References

1. Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: The Peersim simulator `http://peersim.sf.net`.
2. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. **33**(3) (2003) 3–12
3. Benbadis, F., Mathieu, F., Hegde, N., Perino, D.: Playing with the Bandwidth Conservation Law. 2008 Eighth International Conference on Peer-to-Peer Computing (2008) 140–149
4. Padmanabhan, V., Wang, H., Chou, P.: Resilient peer-to-peer streaming. In: Network Protocols, 2003. Proceedings. 11th IEEE International Conference on. (Nov. 2003) 16–27
5. Huang, F., Ravindran, B., Kumar, V.: An approximation algorithm for minimum-delay peer-to-peer streaming. In: Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on. (Sept. 2009) 71–80
6. Annapureddy, S., Guha, S., Gkantsidis, C., Gunawardena, D., Rodriguez, P.: Exploring VoD in P2P Swarming Systems. IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications (2007) 2571–2575
7. Huang, Y., Fu, T.Z., Chiu, D.M., Lui, J.C., Huang, C.: Challenges, design and analysis of a large-scale p2p-vod system. ACM SIGCOMM Computer Communication Review **38**(4) (2008) 13
8. Bittorrent: `http://bittorrent.org`
9. Emule: `http://www.emule.com`
10. Kazaa: `http://www.kazaa.com`
11. Deering, S.E., Cheriton, D.R.: Multicast routing in datagram internetworks and extended LANs. ACM Transactions on Computer Systems (TOCS) **8**(2) (1990)
12. Chu, Y.H., Rao, S., Seshan, S., Zhang, H.: A case for end system multicast. IEEE Journal on Selected Areas in Communications **20**(8) (Oct 2002) 1456–1471
13. Castro, M., Druschel, P., Kermarrec, A.M., Rowstron, A.: Scribe: A large-scale and decentralised application-level multicast infrastructure. In: IEEE Journal on Selected Areas in Communication. (October 2002)
14. Ratnasamy, S., Ermolinskiy, A., Shenker, S.: Revisiting IP multicast. In: ACM SIGCOMM 2006. (2006)
15. Comer, D.E., Lin, J.C.: Probing TCP implementations. Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1 (1994)

16. Hefeeda, M., Habib, A., Xu, D., Bhargava, B., Botev, B.: Collectcast: A peer-to-peer service for media streaming. In: ACM/Springer Multimedia Systems Journal. (November 2005) 68 – 81
17. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM (JACM) **43**(2) (1996)
18. Renesse, R.V., Minsky, Y., Hayden, M.: A gossip-style failure detection service. Technical Report TR98-1687, Dept. of Computer Science, Cornell University (28, 1998)
19. Godfrey, P.B., Shenker, S., Stoica, I.: Minimizing churn in distributed systems. Applications, Technologies, Architectures, and Protocols for Computer Communication **36**(4) (2006)
20. Haridasan, M., Vanrenesse, R.: SecureStream: An intrusion-tolerant protocol for live-streaming dissemination. Computer Communications **31**(3) (Feb 2008) 563–575
21. Li, H., Clement, A., Marchetti, M., Kapritsos, M., Robison, L., Alvisi, L., Dahlin, M.: FlightPath: Obedience vs. Choice in Cooperative Services. OSDI (2008) 355–368
22. da Silva, A.P.C., Leonardi, E., Mellia, M., Meo, M.: A bandwidth-aware scheduling strategy for p2p-tv systems. In: P2P. (2008)
23. Picconi, F., Massoulie, L.: Is there a future for mesh-based live video streaming? In: P2P. (2008)
24. Chao, L., Yang, G., Yong, L.: Is Random Scheduling Sufficient in P2P Video Streaming? 2008 The 28th International Conference on Distributed Computing Systems (2008) 53–60
25. Wang, M., Xu, L., Ramamurthy, B.: A flexible divide-and-conquer protocol for multi-view peer-to-peer live streaming. In: P2P. (2009)
26. Wang, M., Xu, L., Ramamurthy, B.: Channel-aware peer selection in multi-view peer-to-peer multimedia streaming. In: Proceedings of 17th International Conference on Computer Communications and Networks (ICCCN '08). (2008) 1 – 6
27. Oh, K.J., Kim, M., Yoon, J.S., Kim, J., Park, I., Lee, S., Lee, C., Heo, J., Lee, S.B., Park, P.K., Na, S.T., Hyun, M.H., Kim, J., Byun, H., Kim, H.K., Ho, Y.S.: Multi-View Video and Multi-Channel Audio Broadcasting System. In: 3DTV Conference, IEEE Comput. Soc (Maio 2007) 14
28. Leitão, J., Pereira, J., Rodrigues, L.: Epidemic broadcast trees. In: Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, Beijing, China (October 2007) 301 – 310
29. Leitão, J., Pereira, J., Rodrigues, L.: Hyparview: a membership protocol for reliable gossip-based broadcast. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Edinburgh, UK (June 2007) 419–429
30. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.: Peer-to-Peer Membership Management for Gossip-Based Protocols. IEEE Transactions on Computers **52**(2) (2003)
31. Voulgaris, S., Gavidia, D., van Steen, M.: CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. Journal of Network and Systems Management **13**(2) (Jun 2005) 197–217
32. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking (TON) **11**(1) (2003)
33. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A Scalable Content-Addressable Network. In: Proceedings of the 2001 conference on Appli-

cations, technologies, architectures, and protocols for computer communications, ACM (2001) 172

34. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, London, UK, Springer-Verlag (2001) 329–350
35. Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley (April 2001)
36. Ripeanu, M., Foster, I., Iamnitchi, A., Rogers, A.: A dynamically adaptive, unstructured multicast overlay. In: Service Management and Self-Organization in IP-based Networks. Dagstuhl Seminar Proceedings, Dagstuhl, Germany (2005)
37. Carvalho, N., Pereira, J., Oliveira, R., Rodrigues, L.: Emergent structure in unstructured epidemic multicast. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Edinburgh, UK (June 2007)
38. Wu, C., Li, B.: Strategies of conflict in coexisting streaming overlays. Proceedings of IEEE INFOCOM (2007)
39. Wu, C., Li, B., Li, Z.: Dynamic bandwidth auctions in multi-overlay p2p streaming with network coding. IEEE Transactions on Parallel and Distributed Systems **19** (2008) 806 – 820
40. Zhao, B.Q., Lui, J.C., Chiu, D.M.: Exploring the optimal chunk selection policy for data-driven p2p streaming systems. In: P2P. (2009)
41. Venkataraman, V., Francis, P., Calandrino, J.: Chunkyspread: Multitree unstructured peer to peer multicast. In: 5th International Workshop on Peer-to-Peer Systems. (February 2006)
42. Coates, M., Iii, A.O.H., Nowak, R., Yu, B.: Internet Tomography. IEEE Signal Processing Magazine (2002)
43. Papadimitriou, P., Tsaoussidis, V., Mamatas, L.: A receiver-centric rate control scheme for layered video streams in the internet. Journal of Systems and Software (2008) 2396–2412