

Media Streaming in Peer-to-Peer Overlay Networks

Adrian-Daniel Popescu
 Department of Electrical and Computer Engineering
 University of Toronto
 adrian@eecg.toronto.edu

Abstract—This paper presents an overview in the area of peer-to-peer media streaming. As a single traditional server node for a media stream can easily become saturated for a fairly low number of clients, the idea is to divide the server load among all client nodes within the system by putting them into both client and server roles at the same time (client role for their own server, and server role for their own clients). In this way, a better scalability of the media stream is achieved. Several approaches solve the problem of media streaming. Some of them focus on distributing the media stream by using a tree-based overlay rooted at the server (origin source). On the other hand, other approaches are receiver-driven oriented. For these a strict hierarchy between peer nodes is not specified and the receiver node has an active role in finding the source nodes and managing connections with them. By not maintaining a hierarchy and by using multiple sources for the media stream, these approaches are usually more robust to failures and have lower overheads.

I. INTRODUCTION

Media streaming received lot of attention in the past few years. As a consequence, live and on-demand media streaming is today widely used to stream tv & radio channels, tv shows, or arbitrary audio & video media. During this time several approaches have been devised to tackle the media streaming problem. The first one is to use a client-server model, where a single server is the media provider and multiple clients are the media consumers. The second one is to use a peer-to-peer approach where the clients help the server in delivering the media content by having the roles of consumers and providers at the same time.

Both schemes have their advantages and disadvantages. The client-server approach has the advantage that the client receives the content directly from the server with the minimum delay but at the cost of overwhelming the server in particular situations (for instance at high rate hours: e.g. football / basketball games etc). As a result, the server's bandwidth can quickly become a bottleneck in the system due to the large number of client requests. On the other hand, in the peer-to-peer approach algorithms are devised to multicast the content

between clients. In this case the clients have an active role in distributing the media content to other clients and thus remove the pressure from the server node. In this way, scaling the system functionality to a large number of consumers becomes a reality. However, this solution has its drawbacks too. Specifically, these algorithms have to tackle a high dynamic system, where clients can come and leave suddenly without any prior knowledge or guarantees.

This paper surveys several papers in the research of media streaming by using the peer-to-peer content delivery approach. In this particular area, a lot of works focus on distributing media content by using a tree-based overlay rooted at the source node (origin server) [1], [2], [3], [4]. Moreover, some of these solutions are using a single source streaming model, where a single source is delivering the media to the requesting node, while others solutions adopt the multiple source streaming model, where multiple sources are delivering media fragments to the requesting node. In this way, the required bandwidth is distributed between the source nodes according to nodes' individual capacity and the system is more resilient to peer departures. On the other hand, other works [5], [6], [7], [8], [9] focused on a receiver-driven media streaming system, where the receiver has the active role in locating the source nodes and configuring the media rates on each of the connection. Moreover, in a case of a node departure, the system is easily reconfigured locally, without affecting a large number of peer nodes.

The rest of this paper is organized as follows. Section II introduces the background in media-streaming and presents several definitions and metrics used in the rest of the paper. Section III presents the tree-based overlays while Section IV presents the receiver-driven approaches for a peer-to-peer media streaming system. Section V presents the lessons learned while studying these streaming schemes and finally, Section VI concludes the paper.

II. BACKGROUND

In this section we present the motivation for using a peer-to-peer (P2P) approach for online media streaming. Then we present the differences between a traditional

P2P data sharing system and P2P media streaming system. Finally, a set of definitions and notations, used in the rest of the paper, are introduced at the end of the section.

A. Motivation

As already presented, in a traditional client-server media streaming system a relative small number of clients can easily saturate the server's bandwidth. Thus a set of solutions which try to maintain server functionality or to improve server's scalability have been proposed (as presented in [1]):

- **Limit the number of clients:** This limits the number of clients to a well known maximum value. However, this does not solve the problem, as many clients will get frustrated by not getting the desired media.
- **Degrade QoS:** Another solution is to degrade the quality of the stream and thus reduce the bandwidth allocated per client. As a consequence, the server can increase the number of clients supported. However, this solution generates a continuous degradation of the stream as clients continue to join and certainly determines client frustration when the quality of the stream becomes unacceptable.
- **Content Distribution Networks:** The traditional client-server approach can be improved by using Content Distribution Networks (CDN). These scale the system functionality to a larger number of clients by using replicated servers which can deliver the same content to clients. Akamai is such a system which employ a dedicated network of thousands of machines in distributing locations to serve content on behalf of the server. However, this solution implies significant costs, affordable only by large commercial web sites. Moreover, there is still a limitation of the scalability as more and more clients join the system.
- **IP Multicast:** IP Multicast would be one of the desirable approaches in solving the problem of multicast, as the server has to send the same message only once, leaving to the routers the responsibility to multicast the content to all the subscribers. However, due to level-3 support required in routers and to other group and network management issues, the practicability of IP Multicast is still limited.
- **Application Layer Multicast (ALM):** The ALM tries to achieve some of the benefits of IP Multicast, by implementing the routing protocol at the application layer. It also uses end-host rather than routers to perform the routing and imply more overheads.

The difference between IP Multicast and ALM can be also seen in Figure 1. Some of the approaches use dedicated hosts into the network as multicast nodes. However, due to cost issues this solution is replaced by implementing ALM protocols on top of P2P networks which seem to solve the problem in a much cheaper way. All the rest of this paper is focusing on P2P media streaming by implementing different types of ALM on top of a P2P network.

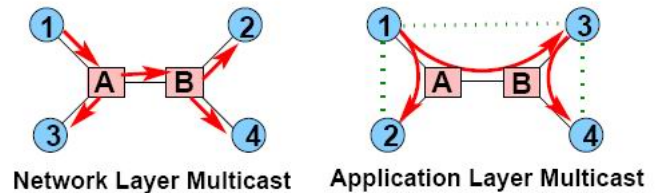


Fig. 1. Difference between network layer and application layer multicast. Square nodes are routers and circular nodes are end-hosts. [3]

B. Differences between P2P data sharing and P2P media streaming

A peer-to-peer (P2P) approach for live media streaming imply some differences with a traditional peer-to-peer data sharing system. Firstly, there is a server node which is the source of the media and there are multiple client nodes which take the role of both client and server. That is, the system is often a combination of a traditional client-server system (we have a server node) with a P2P system (the clients). The client nodes, called also peers or just nodes, connect to other nodes closer to the source node in order to get the media and usually form a tree hierarchy, rooted at the server, with the peers located at the tree's nodes. Secondly, in a traditional P2P network there is the concept to download all the data before opening the file or "open-after-downloading" [7]. In contrast, in media streaming the concept was changed into "play-while-downloading" [7] due to the large file sizes. Thirdly, studies showed that in a P2P system a particular peer is in average part of the system for at least an interval of one hour. That is in contrast with the online media streaming problem, where client nodes can join or leave very quickly, within intervals of minutes or even seconds.

C. Definitions

In the following we present the dimensions of interest in P2P media streaming, as defined in [3]:

- **Quality of the data-delivery path:** This is characterized by metrics such as: *stress*, *stretch* and *node degree*. The first two were originally defined in [10] and recalled in [3]. The stress metric is characterizing each link of the network by counting the number of identical packets sent by a protocol over that particular link. The stretch metric is defined per-node and represents the ratio of the path-length from the source node to the node along the overlay network to the length of the direct unicast path. The degree of a node represents the number of nodes fed by that particular node.
- **Robustness of the overlay:** Because the nodes of the overlay are autonomous (can join or leave at arbitrary moments of time) it is necessary for the application layer multicast protocols to solve the problem of failures. In this context, the robustness of the overlay is quantified by measuring the disruption in data delivery when different nodes fail and by measuring the time it takes to restore the data delivery to the affected nodes.
- **Control overhead:** This metric quantify the scalability of the multicast scheme to large overlays. Thus it is important to maintain the control overhead as low as possible.

D. Notations

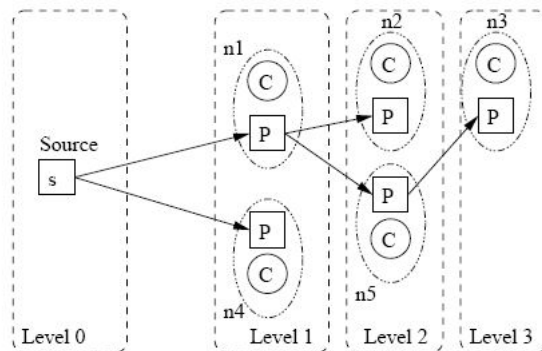
The original source of the media is called the *server*. A *peer* in the system is a host which does not have server characteristics (large bandwidth or large processing power) and can have both roles of client and server. When it has the role of client is a *receiver* and when it has the role of server is a *source*. The term of *node* and *peer* have the same meaning in this paper and are used interchangeably.

III. TREE BASED OVERLAYS

In this section we describe several tree-based approaches for media streaming. We start by presenting the single source source approaches, where each receiver peer receives the media from a single source node and continue with multiple source approaches, where each receiver peer receives the media stream from multiple source peers. We notice that the multiple source solutions are more resilient to peer departures and failures and also limit the client bandwidth usage to a minimum. It is worth also mentioning that all of the following solutions are designed for live media streaming.

A. Single Source Media Streaming

1) **SpreadIt:** SpreadIt [1] is one of the first solutions that distributes bandwidth load over a peer-to-peer network by implementing a multicast tree at the application layer. As it can be seen in Figure 2 the multicast tree rooted at the node s is built over a set of nodes N , which are organized into multiple levels. For each individual node n , located at level $l+1$, there is a parent node p , located at the level l ; n is named the child of p and at its turn, n can be the parent for other nodes situated at level $l+2$ (with $l = 0, 1, 2, \dots$). Building the multicast tree is done incrementally, as nodes adhere to the stream. A data session is created between each node and its children nodes. Specifically, the server s forwards the stream to all of its children which are situated at the level 1 of the tree. Further each node n , located at level l forwards the stream to all of its children located at the level $l+1$. In this way, each node from the system (with a single exception, the server) has two roles in the system: client, for its parent node and server, for its children nodes. The application layer multicast used by SpreadIt helps to balance the load across the network nodes and has the main advantages that is a cheap alternative for media streaming and is scalable, easily supporting thousands of number of clients without implying high performance penalties. The benefits of SpreadIt come at the cost of managing and maintaining the multicast tree. Moreover, the service delays increase with the increase of the levels in the multicast tree. Finally, SpreadIt does not provide the same reliability guarantees as offered by the traditional client-server approaches because the nodes are autonomous and unpredictable.



$N = \{n1, n2, n3, n4, n5\}$, P: Peering Layer, C: Application Client

Fig. 2. Application Level Multicast tree built on the peers. [1]

Another novelty in SpreadIt is the introduction of the *peering layer* between the application and transport layers. This is responsible to maintain connections with the parent and the children nodes. Thus the media appli-

cations are transparent to node failures and are capable in supporting the new system architecture without requiring any additional changes. However, new aspects have to be considered to make use of the novel architecture:

- **Discover the Node's Server:** In SpreadIt the multicast tree is built incrementally, as new nodes subscribe to the stream. The new nodes are fed by already existing nodes in the system. A redirect mechanism is used to find an unsaturated node capable to feed the stream to the new node. Basically, the new node n contacts the server s for that particular stream at the universally known URI (Uniform Resource Identifier) address. The server checks to see if itself is unsaturated. If it is unsaturated it accepts n as its node, otherwise it responds to n with a redirect message with one of its immediate children c as the target peer. The process continues iteratively until n finds an unsaturated node capable to support it.
- **Managing Transience:** Recall that the nodes in the multicast tree are autonomous and unpredictable. Thus SpreadIt has to manage the transience of nodes from the multicast tree. Two type of events which are affecting the transience of nodes are possible: node deletions and node failures. The difference between them is whether or not the node informs the parent and the children nodes of this change. In a normal deletion process a node informs its own parent of its action and also redirects the children with a target node (its parent node or the source node) in order to maintain the multicast tree. However, if a node fails a mechanism should exist in order to locate and replace the faulted node. To solve this problem, SpreadIt uses a heart-beat mechanism at the peering layer which sends alive messages between the node and the corresponding parent and children.
- **Managing Load:** The multicast tree is built over nodes whose primary concern is to act as individual clients. Thus each node should consider its available resources before accepting new children. Basically, the considered resources are the network bandwidth, the processing capacity and the available memory. In this context, the primary concern is the network bandwidth which is usually the top priority resource that limits the number of descendants. As the nodes are heterogeneous within the system, each individual node will limit the number of nodes it supports based on its own resources.

Performance evaluation showed that SpreadIt scales pretty well up to thousands numbers of nodes without

high performance penalties on the streaming quality. Moreover, even the rate of failures is higher than in a traditional client-server approach it has a limited impact on the streaming if the depth of the multicast tree is not large. The depth of the multicast tree is exponentially smaller than the number of clients served, thus the performance degrades "gracefully" with the increase in the number of clients.

2) **NICE:** The goal of NICE (Internet Cooperative Environment) [3] is to implement an application layer multicast tree capable to support large receiver sets while minimizing the latency across the distribution tree and the control overheads. The hierarchy of NICE can be seen in Figure 3. All the host nodes are members of Layer 0. Each layer is partitioned into a set of clusters of sizes between k and $(3k - 1)$, where k is a constant. At its turn, each cluster has a leader node which has the property that has the minimum maximum distance to all other hosts within the cluster. The leader node of a cluster located at the Layer i is also part of a cluster situated at the Layer $(i+1)$.

In NICE two types of paths exist: control paths and data paths. The control paths are concerned with maintaining the hierarchical structure in the case of node joins or node departures while the data paths are the actual paths used for data delivery (streaming). Figure 4 shows two panels corresponding to the control topology (Panel 0) and to the data delivery path when $A0$ is the source node of the data (Panel 1). As it can be seen in the figure, for a particular host X , the peers from its control topology are the other hosts within the same clusters to which node X belongs to. For instance for $A0$ the corresponding peers are: $A1$, $A2$ and $B0$. On the other hand, the delivery path for multicast data distribution needs to be loop free. Therefore NICE adopts to use tree based data delivery path. Specifically, the source node forwards the data to all nodes within the clusters it belongs to. Then, when a node h receives a data from a node p it forwards the packets received to all the clusters h belongs to with the exception of p .

As SpreadIt[1], NICE implements a protocol to maintain the system hierarchy. When a new node joins the system it has to map to some cluster at Layer 0. To achieve this it contacts an universal known node RN, called also the rendezvous point which forwards the request to nodes of the top level layer. The joining node then contacts all the nodes in the higher level in order to locate the node closest to itself. After finding the closest node, the process continues iteratively with the member nodes of the corresponding cluster of the selected node at the lower level, until the new node joins a cluster at the lowest layer (layer 0). Regarding

node transience, two kind of node departures exist in the system: graceful and ungraceful departures. For the second case, a similar heart-beat messages mechanism is used between the nodes. In contrast to SpreadIt, NICE needs to manage cluster maintenance and refinement. Specifically, new node leaders are selected whenever the cluster optimal node is changed and clusters are splitted or merged whenever the sizes of the clusters fall or increase the defined boundaries.

Complexity analysis for NICE showed encouraging results. Specifically, the highest control overhead for a cluster leader of the highest layer cluster is $O(k * \log_k N)$, while the average is only $O(k)$. Moreover, the data delivery path (number of hops) is varying logarithmically with the number of nodes in the system: $O(\log_k N)$. Performance evaluation proved that a low overhead hierarchical control structure can be built. They compared NICE with Narada [10], another application layer multicast protocol and showed slight improvements in the router and link stress and significant improvements in the bandwidth overheads.

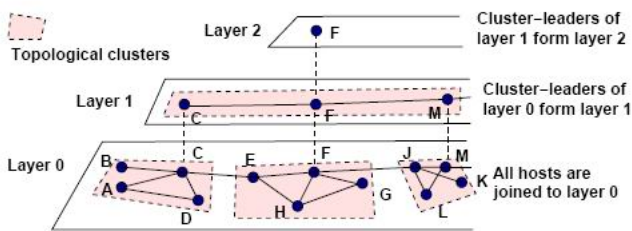


Fig. 3. Hierarchical structure in NICE. [3]

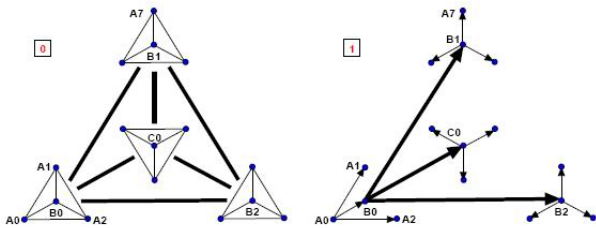


Fig. 4. Control and data delivery paths in NICE. [3]

3) **ZIGZAG**: ZIGZAG [2] is another approach for streaming live media from a single source to many receivers. Although it has some similarities with NICE [3], ZIGZAG aims to distribute high-bandwidth media streaming by using an optimized multicast tree topology. To achieve the goal they take into consideration the following issues: limit the end-to-end delay between the source node and the receiver node by maintaining the

node degree bounded by a constant, implement a quick and graceful recovery in the case of a failure and keep a small control overhead. The administrative organization of peer nodes is almost similar with that specified for NICE [3] with the observations that now we have a server node, which always is the head (has the same meaning with the leader, as specified in [3]) of the clusters it belongs to, the cluster size is bounded by $3k$ (not by $3k-1$) and the top layer contains only a single cluster with the size in the interval $[2, 3k]$. On the other hand the multicast tree is constructed in a quite different way. Figure 5 shows the administrative organization of peers and the corresponding multicast tree. The basic idea is to maintain the degree of a node bounded, this is why a node when is not situated as its highest layer does not have any links to or from any other peer (e.g. peer 4 at layer 1). Moreover, as the head of a cluster is not feeding its cluster mates it received the role of managing their connections with the source node. Further details regarding the construction of the tree structure can be found in [2].

As probably noticed, one of the first differences, as compared with NICE [3] is that the head node of a cluster is not used as the parent for its subordinates. As already stated, that is to limit the node degree and to avoid bottlenecks in the delivery path. ZIGZAG approach has a worst case node degree of $O(k^2)$, in contrast to NICE which has a worst node degree of $O(\log_k N)$. The authors describe also improvements in the control protocol. A node X from the layer- j communicates with its parent, its children and also with its layer- j clustermates. In this case the worst case control overhead is $O(k * \log_k N)$, but the amortized overhead is only $O(k)$. In terms of joining and leaving the multicast tree, ZIGZAG performs as follows. The client join mechanism starts by sending the join request to the server, which forwards the request along the multicast tree until the right peer is found. The algorithm also tries to minimize the distance from the server to the joining peer (in order to minimize delays). The join overhead is $O(\log_k N)$ in terms of the number of nodes the new node has to contact. On the other hand, the node departure mechanism is more lightweight and mostly does not burden the server; in the worst case the number of peers that need to reconnect are $O(k^2)$. In addition, the topology maintenance algorithms required to maintain the cluster bounds at each layer: the merge and split algorithms are both lightweight and bounded by a worst case overhead of $O(k^2)$. All these improvements make ZIGZAG better than NICE when looking at the recovery overhead, the average peer stretch and the average link stress.

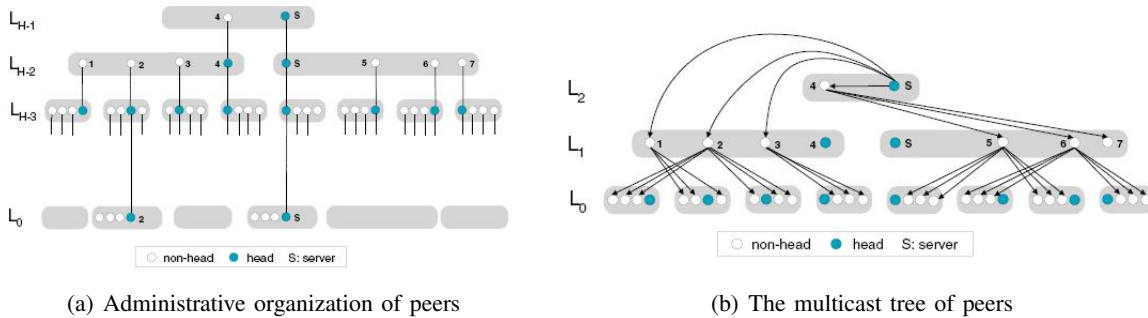


Fig. 5. Administrative organization of peers (left) and the corresponding multicast tree (right). [2]

B. Multiple Source Media Streaming

1) *CoopNet*: CoopNet [4] introduces a new approach in distributing streaming media content by using Multiple Description Coding (MDC) and multiple multicast trees to limit the disruption caused by node departures. It is a combination of an infrastructure system (the traditional client-server approach) with a peer-to-peer system, as it seeks to complement rather than replace the traditional client-server framework. CoopNet also supports both online and on-demand streaming.

One of the novelty of this approach is the usage of MDC which is a method of encoding the audio & video signal into M separate descriptions such that any of these descriptions can be received and decoded into a signal with distortion as compared with the original signal. The distortion decreases with the number of descriptions received, or in other words, the more descriptions received the higher the quality of the reconstructed signal. In order to implement MDC, CoopNet employs a layered audio and video coding model where the audio/video signal is partitioned into groups of frames or GOFs where each group has a specific duration T (e.g. 1 sec). Each GOF is independently encoded and packetized into M packets as it is shown in Figure 6. If any of $m \leq M$ packets are received, the initial R_m bits of the bit stream can be recovered resulting in a distortion $D(R_m)$, with $0=R_0 \leq R_1 \leq \dots \leq R_M$ and $D(R_0) \geq D(R_1) \geq \dots \geq D(R_M)$. As the number of packets received increases the distortion decreases. Every packet is transmitted by a different distribution tree, improving in this way the robustness of CoopNet to node departures or node failures.

What is also worth mentioning is that CoopNet adopts a centralized approach for the tree management protocol. The server has the knowledge of all the distribution trees and the nodes have to contact the server whenever they want to join or leave the multicast trees. As usual two kind of departures are possible: graceful and ungraceful departures. For the second ones, the system determines

a node failure on its delivery path due to quality degradation of the stream. Specifically, each node monitors the packet loss rate it is experiencing in each distribution tree. Whenever the packet loss reaches a threshold value, the node contacts its parent to check if the parent has the same problem. If so, the problem is on parent upstream path. If the parent is not having a problem or is not responding, the affected node will contact the server in order to re-join to a new location in the distribution tree.

Performance evaluation shows that by using MDC in a system with a high number of node departures and with an M value of 8 or 16 all the nodes receives at least one description and more than 98% of nodes receive the more than 87.5% of descriptions. This proves the usage of this technique in the face of node failures and departures. Moreover, if the average repair times is kept to a minimum (up to 1 second) the average fraction of description received is maintained high.

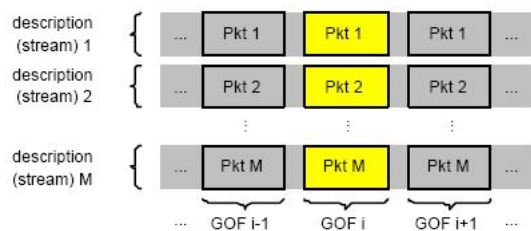


Fig. 6. Construction of MDC streams from packetized GOFs. [4]

IV. RECEIVER DRIVEN P2P MEDIA STREAMING SYSTEM

In this section we present several approaches that do not maintain a strict tree overlay structure as we have seen in the previous section. These are centered around the receiver node, which is now responsible to locate and to select by its own its media sources. All the presented solutions are using multiple sources in order to balance the bandwidth requirements on the source nodes and

to improve the robustness of applications. Most of the solutions can be applied for live streaming, however they seem more appropriate for on-demand streaming.

A. PROMISE & CollectCast

PROMISE is a receiver-driven P2P media streaming system which improves robustness of streaming by using multiple media sources and improves performance by using a topology based selection scheme which indicates the best source peers. The PROMISE architecture can be seen in Figure 7 and basically consists of a set of peers interconnected through a P2P substrate. What is worth mentioning is that PROMISE is independent of the P2P substrate and thus can be deployed on different underlying P2P infrastructures such as Chord [11], Pastry etc. PROMISE is using CollectCast[6], an application level P2P service which is responsible in finding the sending peers and in managing the network level interactions with the other nodes from the system. Specifically, the roles of CollectCast include the following: selecting the best sending peers, monitoring the P2P network, assigning streaming rates and data segments to sending peers and deciding when a change in the sending peers are required.

The receiver node plays a central role in the system whenever it establish a streaming session for a particular media stream. First it issues a look-up operation to find all the sources for that media. Then the protocol constructs an annotated topology by connecting the candidate peers with the receiver. An active set of peers is selected by choosing the best candidate peers, while the rest peers form the stand-by sender set. At this point the receiver establish connections with each peer from the active set. Two connections are used with each peer: a control connection (on top of TCP) to send control packets and a data connection (on top of UDP) to send the actual media. The receiver is the one that assigns a sending rate for each sender, based on their sender's offered rate and on quality of data path from the sender to the receiver. As far as no switches are required the streaming continues. When a switch is required due to a node failure or a congested path, the protocol updates the active set with new nodes taken from the passive set. What also worths mentioning is that PROMISE uses three techniques to find the best source peers: random, end-to-end and topology aware. The random technique, randomly chooses the the number of peers that can fulfill the aggregate rate requirement. The end-to-end technique estimates the "goodness" of the path from each candidate peer to the receiver. However, it does not consider shared segments among the paths, which

may become a bottleneck in particular situations. Finally, the topology-aware technique considers the underlying topology and its characteristics and thus can make a better selection. Performance evaluation showed that the topology-aware technique is far better than the other two. However, the other ones have still an acceptable performance for streaming the media.

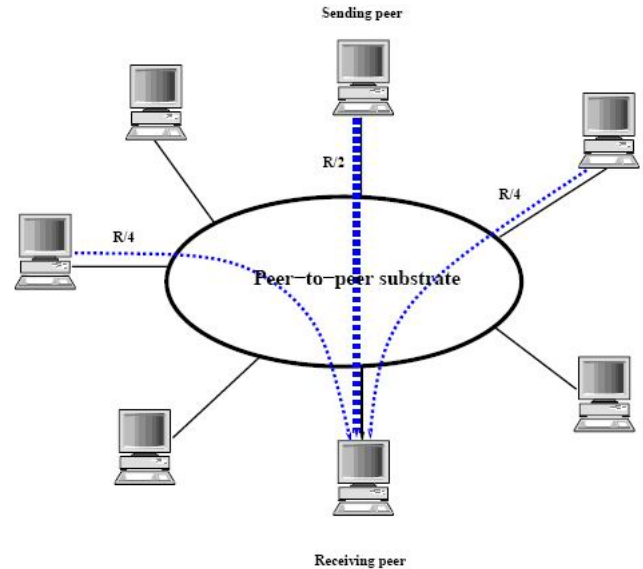


Fig. 7. PROMISE Architecture. [5]

B. Peer-to-Peer Media Streaming

The particularity of Peer-to-Peer Media Streaming [7] approach is that is considering a pure peer-to-peer approach in which server-like source nodes do not exist and peer nodes have heterogeneous characteristics. They also classify peers into classes, according to total number of possible values for their out-bound bandwidth offer. As the peers have limited resources, the media stream is taken from multiple sources. Specifically, two algorithms are used: OTS_{p2p} , that computes the optimal media assignment for each source node for the streaming session, and DAC_{p2p} , which is a distributed differentiated admission control protocol for nodes that attempt to get the media. The first one ensures that the node assignment will result in the minimum buffering delay requested by applications while the second one achieves faster amplification of peer-to-peer capacity and an overall higher admission rate and fewer rejections. Moreover the protocol differentiate between requesting peers with different out-bound promises and thus it stimulates peer nodes to offer their entire bandwidth that they have. DAC protocol is executed by both the supplying peers and by the requesting peers and has the following particularities:

- **DAC_{p2p} on Supplying Peers** The basic idea is that the supplying peers favour requesting peers within the same class or higher by always granting those requests (with a probability of 1). They also serve peers from a lower class but with a less probability. That is, the probability for a lower class decreases proportionally with the level of class below the supplier's node class.
- **DAC_{p2p} on Requesting Peers** The idea here is that each requesting peer P_r first obtains the list of candidate supplying peers for the requested media via a P2P lookup mechanism. Immediately after, the requesting node contacts the candidate supplying peers starting from high to low classes. P_r will be admitted to receive the stream if it will receive enough supplying peers which are not busy and which are willing to provide the stream. Moreover, the aggregated out-bound bandwidth of the supplying nodes has to be at least the minimum bandwidth required to stream the media.

C. GnuStream

GnuStream [8] is a P2P receiver-driven media streaming system built on top of Gnutella. As the previous approaches it is using a lookup mechanism in order to detect nodes that host the desired media. It also requiring multiple sources for the media stream. Other features that worth mentioning are:

- **Receiver data coordination:** The receiver has the responsibility to reconstruct the media data received from multiple sources in their original form before feeding the data to the media player.
- **Detection of peer status change:** GnuStream probes the sender nodes periodically in order to detect any changes in their status, specifically node disconnection, failure, bandwidth degradation.
- **Recovery from failure:** As in PROMISE [5], GnuStream has a set of standby sender nodes which can be used whenever some of the active senders fail or encounter a bandwidth degradation.
- **Buffer control:** GnuStream uses a set of buffers in order to accommodate a better level robustness for the streaming application.

D. CoolStreaming & DONet

The main characteristic of DONet [9] is that adopts a data centric design for the streaming overlay where a node forwards the data to other nodes that are expecting that data without any prescribed roles (for instance the father/child roles from the tree overlays). In other words, data availability determines the flow direction, as

compared with static structure overlays which impose flow restrictions. The main idea in DONet is to maintain a set of partners for each node, from which the node can potentially receive the unavailable data or to which it can send the available data. Each node has a unique identifier in the system and has a membership cache with a set of active nodes from DONet overlay. When a new node wants to join to the system it contacts the origin node (the initial source of the stream) which redirects the request to a deputy node. The node can then select a list of partner candidates from the deputy, and then contact these candidates directly in order to establish its partners. Figure 8 shows an illustration for a potential partnership between nodes.

Regarding media distribution, in DONet a media stream is divided into segments of the same size, and each node from the system can mark the availability of a particular segment by using a buffer map. As each node exchanges its own buffer map with its partners it can mark which segment to fetch from which partner. A scheduling algorithm is also devised to get the expected segments from the partners and basically it tries to satisfy two constraints: i) the playback deadline for each segment and ii) the heterogeneous bandwidth from the partners.

Performance evaluation on PlanetLab showed that CoolStreaming (an live streaming application built on top of DONet overlay) achieves high streaming quality in terms of streaming rate and playback continuity. Comparisons with tree based approaches show that playback continuity in DONet is much better, and thus the percentage of affected nodes in the case of node failures/departures is much smaller. Figure 9 shows the variation of continuity index with the time. As we can see, a traditional tree approach has many spikes during the time interval while the DONet ensures a much better and stable continuity. These are promising results which are showing again the main drawbacks of the tree based infrastructures: the overheads implied in recovering from node failures and the high influences of a node from the top of the tree hierarchy on its children nodes.

V. DISCUSSION

In the following we devise a short analysis of the works that we presented in this survey. We have seen two categories of approaches for the P2P media streaming problem, one that maintain an enforced hierarchical structure on top of P2P overlay (tree or multiple tree), and the second that don't impose any hierarchical restrictions and are receiver-driven oriented. Although is difficult to say which is better in the general case, we try to perform a comparison between them.

TABLE I
TREE BASED OVERLAY STREAMING VERSUS RECEIVER-DRIVEN STREAMING

Attribute	Tree-based	Receiver-driven
Hierarchy	tree maintenance	none; work on top of P2P
No of Source Nodes	one or many	many
Locate source(s)	contact parent node	lookup
Node management	contact root node, iterative process	localized, less overhead
Node heterogeneity	not always considered	considered
Robustness	fragile for single source	in general better
Media continuity	can fluctuate	more stable

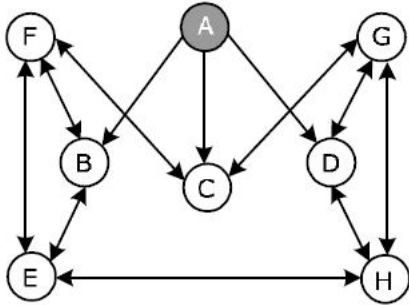


Fig. 8. Partnership in DONet (A is the origin node). [9]

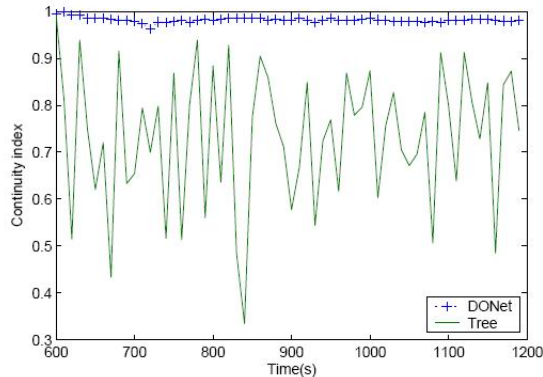


Fig. 9. Samples of continuity indices for DONet and a tree-based overlay. [9]

In our opinion the tree-based approach is more appropriate in live media streaming while the receiver-driven approach is more appropriate for the on-demand streaming. For the first case, it is clear that all the nodes from the tree hierarchy are interested in the same fragment of the media (because of synchronous media) and thus a hierarchical structure makes sense here (a child node receives the media from its parent node once the parent buffered that fragment of the media). On the other hand, for on-demand streaming this may not be the case. For instance, nodes can stream different fragments of the media file (even at the same moment of time), and

thus it makes no obvious sense to build and manage a tree structure where only parents node can feed the child nodes. In this case it would be more desirable to get the media from multiple sources (which have the required fragments of media), in a receiver-driven manner.

Table I shows some of the characteristics of both of the approaches. The main drawbacks of the tree approaches is that require tree maintenance and thus it implies more overhead. Moreover, in the case that a node from the top of the hierarchy leaves the system many child nodes get affected. On the other hand, the receiver-driven approaches don't have such a hierarchy. The difference is that the receiver has to lookup for source nodes and setup connections with each of them. They also consider heterogeneity of peer nodes. An advantage of some of the receiver-driven approaches which implement admission control mechanisms is that these encourage nodes to offer larger out-bound bandwidth by separating peer nodes into different classes (according to the offered bandwidth) and prioritizing those from higher classes to get first into the system. This determines a faster amplification of the peer-to-peer capacity and an overall higher admission rate. In the tree based approaches this idea wasn't really taken into consideration. Finally, we can also say that in general case the robustness and media continuity are better for the receiver-driven approaches, as showed in the experiments presented in DONet paper [9].

Finally, in the following we highlight some of the lessons learned while studying these approaches:

- ZIGZAG [2] showed that the tree based approaches can achieve fairly good performance if some designing issues are taken into consideration, specifically: i) maintain the degree of a node bounded, ii) implement quick recovery in the case of a failure and iii) keep the control overhead to a minimum.
- CoopNet [4] showed that robustness of tree-based approaches can be improved considerable by combining multiple-trees with the MDC mechanism.
- PROMISE [5] showed that the topology-aware se-

lection is the best technique in the process of selecting the best source nodes.

- DONet [9] showed that a data-centric design of the streaming overlay may be a better approach than a strict hierarchical structure (e.g. tree) that enforce the direction of flow. The results were very promising: considerable less fluctuations for the media continuity index and better robustness.

VI. CONCLUSIONS

This paper offers an introduction in P2P media streaming. It starts by presenting tree-based overlay approaches which are using a tree structure rooted at the server node for streaming the media. With a single exception, the origin server, all the other nodes in the tree are performing both roles of clients and servers at the same time. These solutions are highly applicable for live media streaming. However, they suffer in the case of node failures or unexpected node departures in the tree hierarchy. In order to improve robustness of the tree structure, multiple-tree approaches have also been introduced. One of these approaches is encoding the original media stream into multiple descriptions which are associated to different trees. As a consequence, when a node from a tree fails it doesn't affect the streaming process significantly, it affects only the playback quality.

The receiver-driven approaches, studied in the second part of survey don't maintain a strict hierarchical structure among the peer nodes. Here the receiver plays a central role: it searches the media fragments through the P2P substrate, it choses the senders and it selects the bandwidth allocations for each sender. These approaches are always using multiple source nodes. However, the approach is different here. The stream is fragmented into *distinct* parts which are then taken from different source nodes. From our point of view these solutions are more applicable for on-demand streaming. Moreover, by using admission control mechanisms in the streaming process they can determine a faster amplification of the peer-to-peer streaming capacity and an overall higher admission rate.

REFERENCES

- [1] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over a peer-to-peer network," *Technical Report*, August 2001.
- [2] D. A. Tran, K. A. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, 2003, pp. 1283–1292.
- [3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," *Technical report, UMIACS TR-2002*, 2002.
- [4] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2002, pp. 177–186.
- [5] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: peer-to-peer media streaming using collectcast," in *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*. New York, NY, USA: ACM, 2003, pp. 45–54.
- [6] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev, "Collectcast: A peer-to-peer service for media streaming," *ACM/Springer Multimedia Systems Journal*, October 2003.
- [7] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," *Purdue Computer Science Technical Report*, April 2002.
- [8] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "Gnustream: a p2p media streaming system prototype," in *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 325–328.
- [9] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming," in *IEEE INFOCOM 2005*, March 2005, pp. 2102–2111.
- [10] Y.-H. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *ACM SIGMETRICS*, June 2000.
- [11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, 2001.