Resilience in Live Peer-to-Peer Streaming

Viktoria Fodor and György Dán, KTH Royal Institute of Technology

ABSTRACT

The success of peer-to-peer overlays for live multicast streaming depends on their ability to maintain low delays and a low ratio of information loss end-to-end. However, data distribution over an overlay consisting of unreliable peers is inherently subject to disturbances. Resilience is thus inevitably a key requirement for peer-to-peer live-streaming architectures. In this article, we present a survey of the media distribution methods, overlay structures, and error-control solutions proposed for peer-to-peer live streaming. We discuss the trade off between resilience and overhead and argue that efficient architectures can be defined only through thorough performance analysis.

INTRODUCTION

Live multicast of streaming media over the Internet presents several challenges. First of all, high bit rate streams must be delivered to a potentially large population of users. This requires either high transmission capacity at the streaming server, if using the traditional client-server approach, or multicast support in the network. The peer-to-peer approach aims to alleviate these demands by utilizing the upload bandwidth of the participating peers to distribute the media stream. However, peer-to-peer streaming faces several challenges. The number of peers participating in the overlay may change rapidly; the streams must be transmitted with end-to-end delays that are acceptable for live applications, in the range of a couple of tens of seconds; and to keep the perceived media quality acceptable, the packet loss rate must be low.

In this article, we present an overview of the methods proposed to achieve robust data transmission in peer-to-peer live media streaming. We argue that the proposed ideas must be evaluated thoroughly — both analytically and in real-life settings — to define the building blocks of an efficient architecture.

We present the main system requirements and the two media distribution methods proposed for live streaming in peer-to-peer networks. We discuss the building blocks and the performance of streaming with the push method and the pull method, respectively. We discuss how to provide resilient transmission for peer-to-peer streaming. We give an overview of the ideas to optimize streaming performance and then conclude the article.

SYSTEM REQUIREMENTS AND DATA TRANSMISSION METHODS

For peer-to-peer streaming to be successful, it must provide low end-to-end packet loss rate, delay, and delay jitter, similarly to point-to-point live streaming. To satisfy these requirements, the transmission bandwidths must be utilized effectively in bandwidth-scarce environments, and the system must adapt to changes in network conditions, for example, to increasing congestion on some of the peer-to-peer transmission paths. The solution must be resilient to churn, that is, when peer nodes join and leave the overlay during the streaming session, the cohesion of the overlay structure must be maintained, and the information loss due to node departures [1] must be minimized.

The architecture proposed for peer-to-peer streaming generally falls into one of two categories: push-based or pull-based. Solutions in both categories utilize multi-path transmission to ensure graceful quality degradation in dynamic overlays. With multi-path transmission, parts of the stream reach the peers through independent overlay paths, and consequently a large part of the streaming data can be received, even if some of the peers stop forwarding.

The push method [2–4] follows the traditional approach of IP multicast. The streaming data is forwarded along multiple disjoint transmission trees with the streaming server as the root. The peers are the nodes of the trees and relay streaming data with low processing delay. The streaming data is divided into packets, and packets are allocated in a round robin manner to as many slices as there are trees. Packets of a slice are transmitted through one of the transmission trees, providing path diversity for subsequent packets in this way. The transmission trees are constructed at the beginning of the streaming session and maintained throughout the session by a centralized or a distributed protocol.

The pull method (also called swarming) [1, 5, 6] follows the approach of off-line peer-to-peer content distribution and aims at efficient streaming in highly dynamic environments. There is no global structure maintained, and parent-child relations are determined locally on the fly as the overlay membership changes. Peer nodes consider the reception of a block of consecutive packets at a time (or larger segments)and fetch those packets from some of their parents. A new set of parents can be selected for the transmission of each block of packets.

Streaming with the Push Method

Multiple-Tree-based Overlays

If the push method is used for peer-to-peer streaming, the structure of the transmission trees and their maintenance have a dominant effect on the streaming performance.

Overlay Structure — The structure of the overlay depends on the download and upload capacities of the peers and the root node and on a number of design choices.

The Download and Upload Bandwidth of the Peers — The download bandwidth of a peer determines whether the peer can receive all slices belonging to the stream and consequently, the number of slices it can relay. If all the peers have a download bandwidth larger then the streaming bandwidth, then all peers are part of all transmission trees.

The Multiplicity of the Root Node — Often, the streaming server itself has enough upload bandwidth to support the transmission of more than one copy of the media stream. The upload capacity of the streaming server — which is the root of the transmission trees — is called root multiplicity. Figure 1 shows a scenario where three peer nodes are directly connected to the root in each tree, thus the root multiplicity is three. The higher the root multiplicity, the shorter the transmission paths are in the overlay.

The Number of Transmission Trees — In the first place, the overlay is based on multiple transmission trees to provide path diversity. Figure 1 shows a case with three transmission trees for an overlay of nine nodes. The figure shows how node 7 receives the slices through other overlay nodes. If node 4 leaves the overlay, node 7 still receives two out of the three slices. By increasing the number of trees, the amount of data lost at peer departures is decreased, which in turn improves the robustness of the data transmission.

The Number of Slices — A slice is the smallest data unit a peer can transmit. The number of transmission trees defines the number of slices that the streaming data is divided into and consequently, the transmission rates of the slices. Thus, the effective utilization of the upload bandwidth of the peers may require a high number of transmission trees.

Upload Bandwidth Allocation Policy — The two extremes of bandwidth allocation policy are to divide the upload bandwidth of a peer evenly among the transmission trees or to allocate all the upload bandwidth to one of the trees. Figure 1 shows an example of the first case. The upload

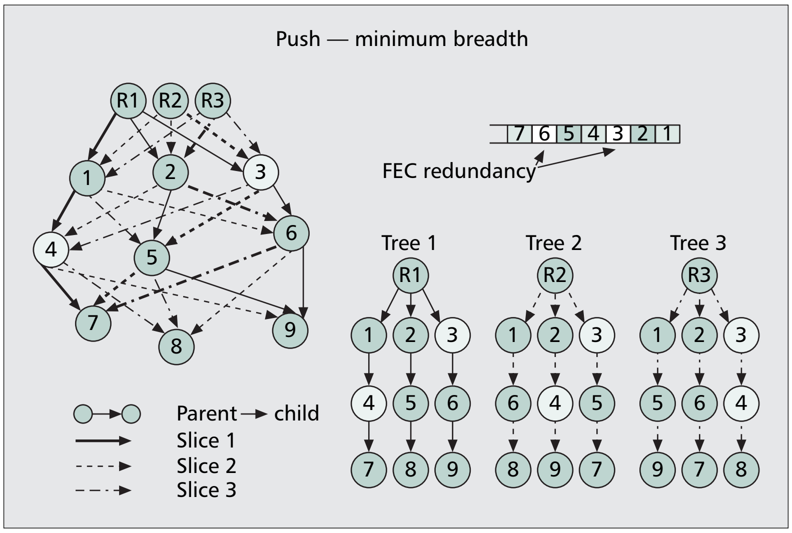


Figure 1. *Push method. Traffic flow in multiple transmission trees with FEC(3,2). Minimum breadth transmission trees for nine nodes, three trees, and a root node with multiplicity 3.*

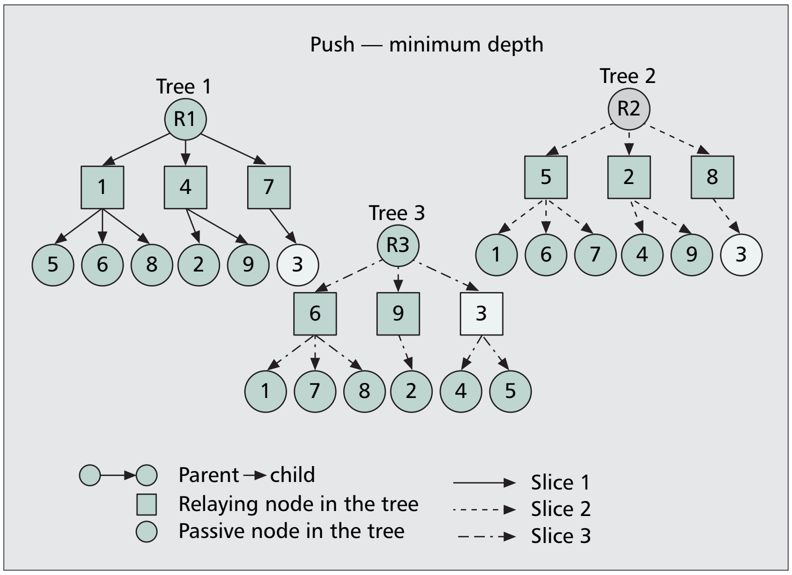


Figure 2. *Push method. Minimum depth transmission trees for nine nodes, three trees, and a root node with multiplicity 3. If node 3 leaves, tree 3 becomes disconnected.*

bandwidths of the peers are considered to be equal to their download bandwidths, and each peer relays data in all transmission trees. Figure 2 gives an example of the second strategy. Here a peer relays data in one tree only, but to several child peers. The second bandwidth allocation strategy has the clear advantage of generating maximum breadth — minimum-depth trees — which in turn minimizes the maximum transmission delays in terms of overlay hops and the effect of churn [7]. More specifically, with the first strategy, the number of overlay hops grows linearly with the number of peers; with the second strategy, the growth is only logarithmic.

We must note, however, that the minimum-depth tree allocation is very rigid, and it can be difficult to maintain the trees in a dynamic scenario. For example, consider the case when node

Control in tree-based overlays is required to connect and reconnect the peers and to adapt the overlay to the varying network conditions. Consequently, the control overhead increases with the number of trees, with the level of peer dynamism, and with the targeted level of adaptation.

3 leaves from the minimum-depth trees shown in Fig. 2. Node 3 leaves two disconnected children, nodes 4 and 5. One of the children can take the position of node 3, but the other one fails to reconnect — the tree runs out of free capacity.

The probability of reconnection failure can be decreased in a number of ways. First, the reconnection failure probability is low if the root multiplicity, and thus the free capacity, in the overlay is high. However, it is not always possible to increase the multiplicity of the root node due to bandwidth constraints of the streaming server. Second, the constraint on allocating all upload bandwidth to only one tree can be relaxed, allowing peers to transmit in several trees, if necessary [8]. This in turn deepens the transmission trees but decreases the reconnection failure probability due to the increased flexibility. Figure 4 shows the reconnection failure probabilities in an overlay with 10,000 nodes, as a function of the number of transmission trees. If the root multiplicity is the same as the number of trees, then the reconnection failure probability is over 10 percent. By doubling the root multiplicity, the reconnection failure probability is decreased to the one percent range. However, this is a solution that is rather demanding of bandwidth. An even greater gain can be achieved by allowing peers to transmit in two of the trees — at the price of increasing the depth of the overlay.

The Placement of Peers in the Overlay — This question is relevant if the peers are heterogeneous in terms of upload bandwidth. Placing peers that allocate high upload bandwidth for a tree — and thus can support many children — close to the root node will give the trees more breadth, further decreasing the depth of the overlay.

Based on this reasoning, we conclude that peers should allocate a large part of their upload bandwidth to a few of the transmission trees to provide shallow but robust overlays. In addition, high contributor peers should be placed close to the root.

Overlay Management — The construction and the maintenance of the trees can be performed either by a distributed protocol [2, 4] or by a central entity [3]. The tree maintenance affects the performance of the streaming in the overlay through factors such as the introduced control traffic, the probability of incorrect peer availability information, the time to detect a peer departure, and the time to reconnect the disconnected children.

The overlay management is responsible for maintaining the overlay under varying network conditions, such as congestion levels between the peers and in the case of churn. The tree construction algorithm may aim for building structured [2, 3] or unstructured [4] trees. In the first case, the structure is predefined to ensure an overlay that is optimal in some sense, for example, has minimum depth. However, maintaining the optimal structure may lead to low performance in the case of varying network conditions [1, 4], may require significant management overhead in the case of churn [8], and gives little possibility to prioritize peers. In the unstructured case, the overlay may be improved continuously under the streaming session according to some measure, for example, by pushing reliable peers close to the root and by re-selecting parents if the transmission performance is not adequate.

Control in tree-based overlays is required to connect and reconnect the peers and to adapt the overlay to the varying network conditions. Consequently, the control overhead increases with the number of trees, with the level of peer dynamism, and with the targeted level of adaptation.

Data Transmission

In push-based peer-to-peer streaming applications, one of the principal sources of impairment of streaming quality is peers leaving the overlay. When a peer leaves, the subtrees of its children become temporarily disconnected and do not receive the slices transmitted in the particular trees. In addition, transmission paths between the peers may become congested, leading to packet losses due to buffer overflow or to late arrival to the playout buffer.

Because peers relay the stream, information loss between a parent-child pair may further propagate in the transmission tree, and losses accumulate as the distance from the root increases. Therefore, peer-to-peer streaming requires error-control solutions that enable information reconstruction at the peers, also with low delay, considering the delay requirements of live streaming. We discuss the error-control solutions considered for push-based peer-to-peer streaming later.

Streaming with the Pull Method

Mesh-based Overlays

The overlay constructed for pull-based streaming is often referred to as directed mesh, where directed edges represent potential parent-child relations. The mesh is reconstructed periodically as peers search for new parents after the transmission of each data block. Overlay membership information is managed by a central entity or spread by gossiping with message exchange between child and parent peers.

As shown in Fig. 3, peers select a set of parent nodes independently from each other, receive a buffer map, receive the list of available packets from the parents, construct a transmission schedule, and fetch the block of packets. A peer node can be selected as a parent if it possesses some of the packets in the required data block and has upload capacity not allocated to other children.

Overlays employing the pull method are dynamic and follow the changes of membership and network conditions in the overlay easily, compared to the push-based approach, where a tree structure must be maintained [1]. The cost of this simplified overlay management is twofold. First, the data transmission requires the frequent exchange of control messages. Second, the pulling process introduces additional delay at each overlay hop: first the list of available packets is transmitted from the parent to the child node, then the child node sends a request message to the parent, and finally, the packets are transmitted.

The performance of pull-based streaming is determined by the overlay structure, the packet scheduling, and the overlay management.

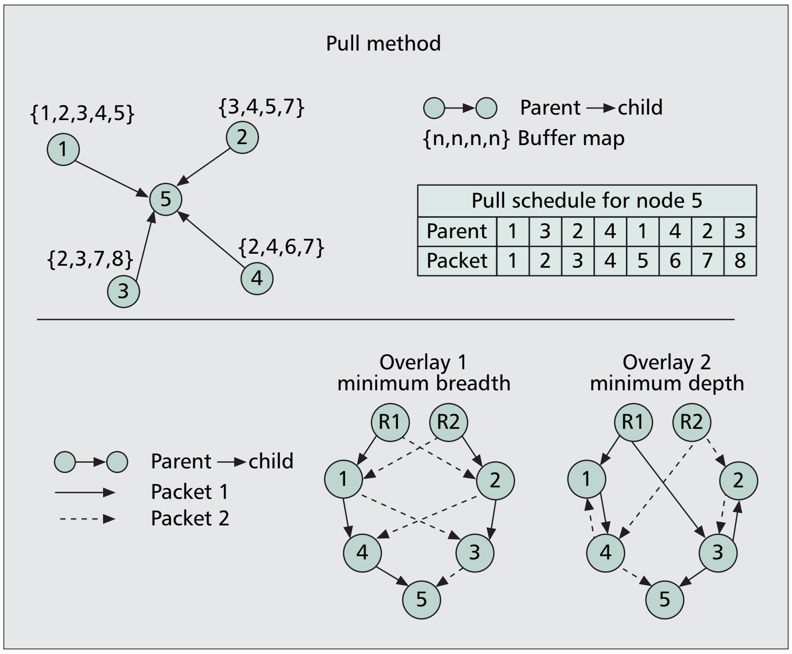
Overlay Structure — The structure of the overlay depends on the download and upload capacities of the peers and the root node, as in the case of the push method.

The Number of Parents for Each Peer — The number of parents for each peer determines the number of potential peers that can upload parts of the stream. Increasing the number of parents increases path diversity and resilience to churn. This parameter relates to the number of transmission trees in the case of push-based streaming.

The Number of Packets in a Block — One packet per block gives the smallest transmission rate at which a peer can transmit. Consequently, the choice of this parameter determines the utilization of peer upload bandwidths in a bandwidth heterogeneous environment. Note that in contrast to the push method, the minimum transmission rate is now decoupled from the number of parents. This flexibility enables better utilization of transmission bandwidths without significantly increasing the control overhead.

The Selection of the Parents and the Scheduling of Packet Transmission in a Block — In most of the proposed solutions, [5, 6] children try to fetch data at the earliest from parents who possess the packets, aiming to minimize the number of overlay hops. As Fig. 3 shows, however, more careful parent selection is required to obtain shallow overlays. The figure shows how packets of an arbitrary block of length two can be spread in the overlay. All peers have two parents, and the root has a multiplicity of two. In the figure, two possible cases are shown. In the first case, packets travel along transmission paths that are analogous to the minimum-breadth trees with the push method; in the second case, the transmission paths are analogous to the minimum-depth trees with the push method. For a small network such as this, the maximum length of the transmission paths already is larger for the first case. Consequently, the parent selection, and also the scheduling scheme — the way parents allocate bandwidth to the children — affect the length of the transmission paths. Since packets travel along transmission trees even in pull-based streaming, we can follow the earlier reasoning and conclude that short transmission paths are achieved if each peer forwards only a small part of the streaming data, but to a high number of children. In the pull method, too, peers with high upload bandwidth should be placed close to the root to minimize the number of overlay hops.

Packet Scheduling — For each block of packets, the child node decides from which parent the individual packets should be pulled. Due to this block-by-block decision, pull-based overlays can quickly react to changes in the network conditions and to churn. A node can immediately utilize the free upload bandwidth of some of its parents when others get congested or leave the overlay. However, the flexibility of packet scheduling comes at a price. Because schedules at the peers are not coordinated, the scheduling may be infeasible so that child peers may experience discontinuity in the stream, as we discuss later.



Pull schedule for node 5								
Parent	1	3	2	4	1	4	2	3
Packet	1	2	3	4	5	6	7	8

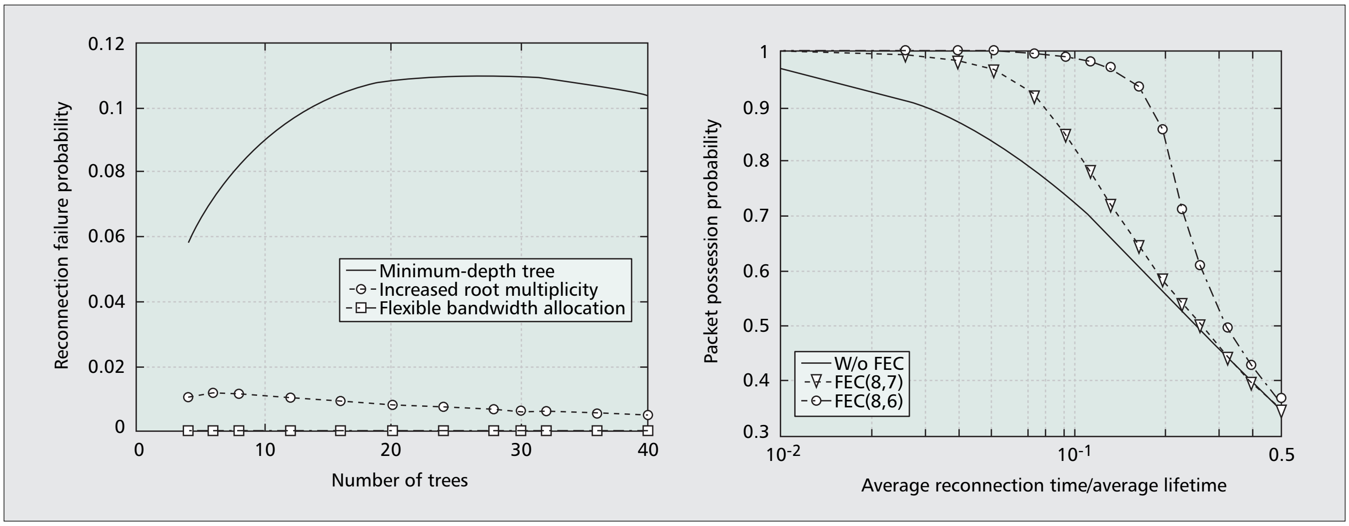
■ **Figure 3.** *Pull method. Example for buffer maps and pull schedule and packet transmission paths with different parent selection.*

Overlay Management — The overlay management in the case of the pull method deals with maintaining the mesh of parent-child relations. Parent-child relations are changed if a parent leaves the overlay, or if the current set of parents cannot provide the required streaming quality.

Figure 5 shows the control overhead — the ratio of control and data traffic at each peer — at a 500 kb/s streaming rate in the case of distributed overlay management. The graphs show approximate values from [5]. The control overhead depends on the number of parents a peer maintains, because the overlay is managed and the data transmission is controlled through message exchanges between parent-child pairs and seems to be independent of the size of the overlay. The control overhead increases with the level of network dynamism — here reflected by the length of on and off periods of each peer in the overlay — as the overlay membership information must be refreshed more frequently.

DATA TRANSMISSION

Discontinuity in streaming can occur in a dynamic overlay if the partnership information is incorrect, or if a parent leaves before transmitting the pulled packets.



■ **Figure 4.** *Push method. Reconnection failure probability and data distribution performance in dynamic overlays.*

Discontinuity happens even in a static overlay when a packet is lost due to congestion or can not be scheduled within the transmission deadline. This in turn means that the data transmission is not feasible due to one of the following reasons: the overlay ran out of capacity, the potential parents have the packet available but no available upload bandwidth, there are too few peers possessing the packet, or the packet "disappeared" from the network due to network failures.

Although analytic results on the data distribution performance are not available in the literature, experimental results are reported in [1, 5]. As shown in Fig. 5, the ratio of packets not received is around 2 to 5 percent in a stable environment, depending on the number of parents and 5-15 percent in a dynamic environment, depending on the level of network dynamism, with the distributed overlay management described in [5]. Further analysis is required to see if discontinuity is caused by unfeasible schedules at the peer nodes or by error propagation similar to the push-based case and to evaluate the performance of very large overlays.

STREAMING WITH THE PUSH-PULL METHOD

While pull-based solutions provide high bandwidth utilization in heterogeneous networks and react quickly to churn and varying network conditions, they introduce significant latency by pulling small amounts of data at a time. To decrease latency while maintaining the desirable properties of pull-based streaming, combining push and pull mechanisms may offer a good compromise [9].

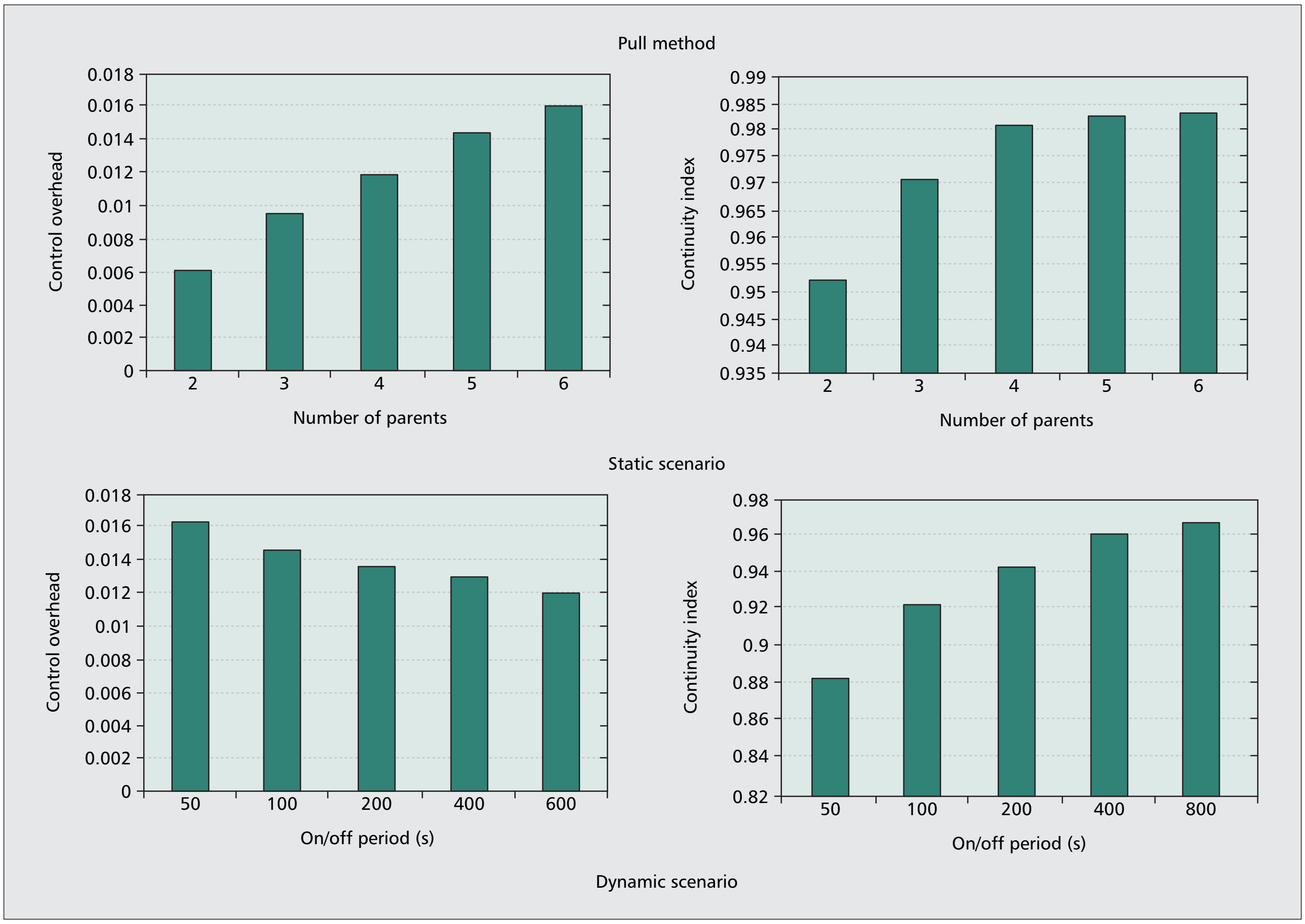
With the push-pull methods, a schedule of packet transmission is defined through a pulling phase, based on the buffer maps of the parent peers. After the packet schedule is defined for one block, the same schedule is applied for the subsequent blocks without testing the actual buffer content of the parents, that is, the overlay works in push mode for some time, providing low end-to-end delays and lower control load. When a new packet schedule is defined, the push phase again is followed by a pull phase.

The length of the push phase determines how quickly the overlay reacts to churn and to changes in the network conditions and can be tuned adaptively. Consequently, the push-pull method trades flexibility for lower end-to-end delay and control load.

RESILIENT TRANSMISSION FOR PEER-TO-PEER STREAMING

In peer-to-peer networks, packets may not be delivered to the peers due to churn or due to congestion or network failures. To provide the desirable low packet-loss rate for the streaming application, some form of error control is required. Traditionally, there are two methods of error control in communication networks. Lost information can be retransmitted or regenerated based on the redundancy present in the stream. In the following sections, we discuss how these two approaches can be used in the case of peer-to-peer streaming. In the case of pull-based streaming, discontinuity also occurs if the parents have the same subset of packets available. We discuss how network coding helps in this case.

Error Control with Redundancy — Redundancy can be added to the media stream through source coding or by applying algebraic codes over the coded stream. MDC (multiple description coding) falls in the first category and has been proposed for point-to-point streaming applications. MDC encodes the stream into multiple sub-streams called descriptions. If only one description is received, the stream can be decoded with certain accuracy. If more than one description is received, the information from the other descriptions can be used to enhance the accuracy. MDC can be used in peer-to-peer streaming applications to accommodate peers with heterogeneous download bandwidth. However, for error control, MDC is of limited use because peers receiving a subset of the descriptions cannot regenerate the missing ones, which in turn leads to propagation of losses in the overlay.



■ **Figure 5.** *Pull method. Control overhead and continuity index in static and dynamic overlays.*

Forward Error Correction (FEC) based on algebraic codes — like Reed-Solomon codes, works as follows. The source generates the coded media stream and constructs the data packets to transmit. Then, it takes a block of k consecutive packets, applies an error-coding scheme to all the bits in the same position across the packets, and generates the redundant bits, which will compose c redundant packets. This procedure gives an FEC(k + c,k) coded stream. It is enough to receive any k out of $k + c$ packets; the rest of the packets can be reconstructed. The advantage of FEC is that peers can reconstruct the missing packets and forward them to their children; thus with proper redundancy, errors do not propagate in the overlay.

In a peer-to-peer network with push-based transmission, packets of a FEC block are transmitted in the trees in a round-robin manner. The multi-path transmission introduces randomization in the packet-loss process within a block and makes FEC efficient. An example with FEC(3,2) is shown in Fig. 1. Slices with original packets are transmitted in trees 1 and 2, while the slice of the redundant packets is transmitted in tree 3. We follow how node 7 receives the stream. If node 3 leaves, node 4, 5, and 6 receive only two out of the three slices but can reconstruct and then relay the packets of the third one. Then, even if node 4 also leaves, node 7 receives two slices and can reconstruct all data. FEC works similarly for the pull method when applied over the set of packets in a pull block.

The level of redundancy must be selected carefully, so that it really compensates for the information loss. Figure 4 shows analytical results [8] on the average packet-possession probability of the peers in an overlay of 10,000 nodes depending on the node dynamism — reflected by decreasing peer lifetime — and on the added redundancy. The average packet-possession probability already decreases at very low node dynamism if no error coding is applied. Adding redundancy, the packet-possession probability can be kept high, up to a higher level of node dynamism. If the redundancy is not adequate, the packet-possession probability and thus, the perceived media quality, drops to zero in large overlays.

Although redundancy must be adjusted to cope with the losses, it can be increased only by decreasing the rate of the media stream at the same time to keep the transmission rate unchanged. Otherwise, sources might increase their transmission rates as a reaction to congestion, further increasing the congestion in this way. Consequently, adaptive control must be adopted to tune the redundancy level according to the level of dynamism and congestion in the overlay.

Although FEC provides a solution to stop error propagation in the overlay, it has several

The success of live peer-to-peer multimedia streaming depends on the ability of the overlays to meet the expectations of the end users. Overlays should distribute streaming data with acceptable delays and with acceptable perceived media quality in spite of the often highly dynamic overlay membership.

shortcomings. It does not accommodate peers with different download bandwidths, and it is inefficient if the packet-loss probability varies across the peers and if burst of losses happen in short time intervals. To deal with overlays that are heterogeneous in terms of download bandwidth and loss probability, the combined use of FEC and *layered coding* is proposed [3]. Layered coding sorts the bits of the streaming data (bits of a group of frames in the case of video streaming) in decreasing order of importance. Then, FEC is used with different levels of redundancy to protect the data units to a different extent. Peers experiencing high loss rate are able to decode important data with high probability. Peers with low download bandwidth can join to trees pushing the important data in push-based overlays and pull selectively the important packets in pull-based overlays.

Error Control with Retransmission — Redundancy added at the source is an inefficient solution if loss rates fluctuate quickly. Highly varying loss can be handled by the retransmission of lost information or by the combination of added redundancy and retransmission.

Pull-based streaming incorporates retransmission of missing data in the basic operation. Peers can pull a packet again from another parent if it was not received when first pulled but still could be received before the playout time. With push-based streaming, the basic protocol must be extended to incorporate retransmission. If a packet is not received, the peer can try to pull it from one of its parents in the other transmission trees. Both of these solutions utilize the free upload bandwidth of the parent peers. Alternatively, a set of potential parents can be maintained and used for retransmission.

The opportunities for information retransmission might appear to be limited due to the required low end-to-end delays, but efficiency can be improved. If retransmission is combined with error coding, only the packets that improve the error coding capability must be retransmitted. For example, if block coding with k information and c redundant packets is applied, the retransmission must ensure that k out of $k + c$ packets are received, so that the peer node can reconstruct the missing c packets. In the case of coded video, packets retransmitted after their playout time can be used to decode predicted frames and thus stop temporal error propagation in the video stream [10].

Network Coding for Peer-to-Peer Streaming — Originally, network coding was proposed to decrease the required transmission bandwidth if information is transmitted from multiple sources to multiple destinations. Then, the method was extended for the case of large-scale content distribution and recently, for pull-based peer-to-peer streaming [11]. The main point of network coding is to transmit the linear combination of data units instead of the data units themselves. Then, the original data units can be decoded after receiving the right number of linearly independent combinations. Network coding does not help regenerating lost information but increases the possibility of collecting the right amount of information in a distributed environment. Consequently, it can resolve scheduling conflicts in pull-based overlays. Network coding introduces some overhead because the coefficients of the linear combinations must be transmitted but introduces little coding and decoding delay and is computationally simple. According to the first evaluations, network coding can improve streaming quality in dynamic and bandwidth scarce environments.

Optimizing Performance

The performance of the data transmission can be improved by decreasing the end-to-end delays and the effects of peer departures. A number of prioritization schemes, which move peers in the overlay or reallocate parents according to some metrics, were proposed and evaluated [6, 7, 12].

End-to-end delays are affected by the number of peer-to-peer hops that data traverses in the overlay and by the propagation delays between the parent-child pairs. The first one can be minimized with the correct tree structure and by moving peers with high upload bandwidth close to the root [12]. Propagation delays can be minimized by selecting parents based on geographic locations [6]. Since both the upload bandwidths and the geographic location of the nodes are static, predictable metrics that prioritize according to upload bandwidth and geographic location, prove to be efficient methods to improve the quality of the data transmission.

It is more difficult to minimize the effects of congestion and dynamic overlay membership. The disrupting effect of congested and departing peers can, in principle, be minimized by moving reliable peers close to the root node. However, experiments with real-life traces showed that predictions on congestion and peer lifetime are not accurate, and prioritizing according to the predicted reliability does not improve and might even degrade the performance [4, 7].

Discussion

The success of live peer-to-peer multimedia streaming depends on the ability of the overlays to meet the expectations of the end users. Overlays should distribute streaming data with acceptable delays and with acceptable perceived media quality in spite of the often highly dynamic overlay membership. In this article, we discussed the performance of the two methods underlying most of the proposed mechanisms for live peer-to-peer streaming: the push method, where data is distributed along transmission trees, and the pull method, where data is pulled from frequently reselected parent peers. Table 1 summarizes the main properties of the pull and the push methods.

We found that the performance of the data distribution is affected by the ability of the overlays to provide transmission paths with a low number of overlay hops, which in turn depends on the strategy of the peers to allocate their upload bandwidths. Although upload bandwidth allocation strategies are easy to incorporate in the push-based methods, they may increase the complexity of parent selection significantly in the now emerging pull-based solutions. In contrast, pull-based

	Push	Pull
Overlay	Maintains multiple transmission trees	Defines partnership mesh and for the whole streaming session Schedules block of packets
Sign of infeasibility	Reconnection failure	Infeasible transmission schedule
Delay control	Tree structure	Parent selection and scheduling
Loss control	Redundancy and retransmission	Redundancy, scheduling, including retransmission and network coding
Bandwidth utilization	Tree construction and maintenance	Scheduling
Performance optimization	Tree maintenance	Scheduling and parent reselection
Resilience to churn	Tree construction and loss control	Mesh maintenance, scheduling, and loss control
Control cost	Tree maintenance	Mesh maintenance and packet pulling
Trades resilience for	Redundancy and control	Delay and control

Table 1. *Comparing the push and the pull methods for peer-to-peer streaming.*

solutions can better utilize the transmission bandwidth in bandwidth-scarce overlays and adapt better to temporal changes of network conditions.

We showed how packet loss due to peer departure can propagate in the push-based overlays and discussed the challenges of efficient error control. We also argued that a thorough analytical evaluation of the push method is necessary to see how the overlay size affects the data distribution performance. Only after we understand the behavior of the push and the pull methods can we define a hybrid architecture that achieves both resilience against churn and good data distribution performance in terms of end-to-end delay and loss.

ACKNOWLEDGMENTS

This work was supported in part by the Swedish Foundation for Strategic Research through the projects Winternet and AWSI and by Wireless@KTH.

REFERENCES

[1] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," *Proc. IEEE INFOCOM*, May 2007.

[2] M. Castro *et al.*, "SplitStream: High-Bandwidth Multicast in a Cooperative Environment," *Proc. ACM Symp. Op. Sys. Principles*, Oct. 2003.

[3] V. N. Padmanabhan, H. Wang, and P. Chou, "Resilient Peer-to-Peer Streaming," *Proc. IEEE Int'l. Conf. Network Protocols*, Nov. 2003.

[4] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured End System Multicast," *Proc. IEEE Int'l. Conf. Network Protocols*, Nov. 2006.

[5] X. Zhang *et al.*, "CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming," *Proc. IEEE INFOCOM*, Mar. 2005.

[6] J. Liang and K. Nahrstedt, "DagStream: Locality Aware and Failure Resilient Peer-to-Peer Streaming," *Proc. Multimedia Comp. and Networking*, Jan. 2006.

[7] P. B. Godfrey, S. Shenker, and I. Stoica, "Minimizing Churn in Distributed Systems," *Proc. ACM SIGCOMM*, Sept. 2006.

[8] G. Dán, V. Fodor, and I. Chatzidrossos, "On the Performance of Multiple-tree-based Peer-to-Peer Live Streaming," *Proc. IEEE INFOCOM*, May 2007.

[9] M. Zhang *et al.*, "Large-Scale Live Media Streaming over Peer-to-Peer Networks through the Global Internet," *Proc. ACM Wksp. Advances in Peer-to-Peer Multimedia Streaming*, Nov. 2005.

[10] E. Setton, J. Noh, and B. Girod, "Rate-Distortion Optimized Video Peer-to-Peer Multicast Streaming," *Proc. ACM Wksp. Advances in Peer-to-Peer Multimedia Streaming*, Nov. 2005.

[11] M. Wang and B. Li, "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming," *Proc. IEEE INFOCOM*, May 2007.

[12] T. Small, B. Liang, and B. Li, "Scaling Laws and Trade-Offs in Peer-to-Peer Live Multimedia Streaming," *Proc. ACM Multimedia*, Oct. 2006.

BIOGRAPHIES

VIKTORIA FODOR (vfodor@ee.kth.se) received her M.Sc. and Ph.D. degrees in informatics from Budapest University of Technology and Economics, Hungary, in 1993 and 1998, respectively. She is an associate professor at the Laboratory for Communication Networks at KTH, Royal Institute of Technology, Stockholm, Sweden. Her current research interests include protocol design and performance evaluation of multimedia content distribution systems. She has conducted research in the areas of all-optical networking, end-to-end traffic control, and multimedia communication. She worked as a senior research engineer at the Hungarian Telecommunications Company in 1997–1998 and joined KTH in 1999.

GYÖRGY DAN (gyuri@ee.kth.se) received an M.Sc. degree in informatics from Budapest University of Technology and Economics in 1999 and an M.Sc. degree in business administration from Corvinus University of Budapest in 2003. He received his Ph.D. in telecommunications in 2006 from KTH. Currently, he is a post-doctoral researcher at KTH. He worked as consultant in the field of access networks, streaming media, and videoconferencing from 1999 to 2001. His research interests include traffic control and performance evaluation of point-to-point and peer-to-peer multimedia communications.