Coordinated Media Streaming and Transcoding in Peer-to-Peer Systems

Fang Chen, Thomas Repantis, Vana Kalogeraki*
Dept. of Computer Science and Engineering
University of California, Riverside, CA 92521
{fchen, trep, vana}@cs.ucr.edu

Abstract

In this paper we study the problem of multimedia streaming and transcoding in P2P systems. We propose a multimedia streaming architecture in which transcoding services coordinate to transform the streaming data into different formats and adapt to both the QoS requirements of the applications and to the availability of the system resources. Our techniques are entirely distributed, use only local knowledge and scale well with the size of the system. Extensive simulation results validate the performance of our approach.

1. Introduction

Peer-to-peer (P2P) systems have gained much attention recently due to their attractive features of self-organization, scalability and decentralized control. In those systems, logical networks are composed of heterogeneous peers with different resource capabilities and variable computation and communication latencies. Peer-to-Peer (P2P) networks have found popular applications in file sharing and content distribution. Recent efforts have focused on providing multimedia streaming at the application level [3, 18, 24]. In these approaches, peers collaborate and share resources with each other in streaming sessions, thus eliminating the need for a dedicated server with a high bandwidth connection.

However, providing adequate support for media streaming on a peer-to-peer system remains a challenge: Firstly, peers are inherently heterogenous in terms of processor capacity, transmission loss rate, network inbound/outbound bandwidth, display resolution as well as decoding software. Secondly, peers are more dynamic than dedicated servers, they may fail or leave the system unexpectedly. Multimedia streaming on the other hand poses stringent resource requirements on processor cycles and available network bandwidth along streaming paths. Thus, since the resource availability in the system is largely affected by peer dynamics, the streaming quality may vary with time in an unpredictable way.

Current solutions generally focus on solving the bandwidth heterogeneity problem. Streaming data, for instance, might be divided into different flows or different layers and transmitted to receivers using different paths [4, 7, 10, 17]. However, these solutions fail to support multimedia applications that require both communication and processing (e.g., transcoding) at various nodes in the overlay network. For example, streaming data might need to be transcoded to a lower spatial and chroma resolution, before it is received by energy-constrained mobile devices [21]. Another example includes users with disabilities that may need data to be transformed in specific ways [2]. Transcoding operations are computation expensive for peers and therefore need careful management to optimize the resource usage. That is especially the case when resources are shared by multiple streaming sessions. Moreover, if multiple streaming paths are utilized for each streaming session [12], transcoding operations at different peers may need coordination in order not to violate the end users' QoS requirements.

In this paper, we present a novel peer-to-peer multimedia streaming system that: **i)** deploys a transcoding service, **ii)** constructs multiple streaming paths with collaborating transcoding peers, **iii)** enables each transcoding peer to adapt to changing resource conditions locally to maximize the transcoding quality, and **iv)** empowers the receiver to coordinate the operations of all the transcoding peers in a session to address quality fluctuations. This paper makes the following contributions:

1. We demonstrate a viable peer-to-peer system that performs an adaptive transcoding service instead of only transferring streaming data. The system uses both local and system-wide adaptation mechanisms to adapt to changes in the infrastructure, the current resource conditions and the user QoS requirements. This scheme

* This research has been supported by NSF Award 0330481.

could be further applied to other services such as watermarking, compression, encryption *etc.*

2. We carry out an extensive performance evaluation through simulations. The results demonstrate the scalability and performance of our approach and show that it can satisfy the QoS requirements of peer-to-peer media streaming.

2. System Model and Design

2.1. Peer-to-Peer Overlay Network

As illustrated in Figure 1 (A), a peer-to-peer overlay network consists of a set of interconnected nodes $S : \{p_0, ..., p_n\}$. Each node $p_i \in S$ is characterized by a tuple $(identification\ id_i,\ bandwidth\ b_i,\ processor\ capability\ c_i, shared\ objects\ O_i,\ service\ components\ S_i)$, in which S_i describes what services are shared by the peer (*e.g.*, transcoding service). Note, however, that O_i or S_i may be empty sets. To identify nodes in the overlay network, id_i is the pair $< IP_i,\ port_i >$. In order to support multi-path streaming of the same media object, each media object $ob_j \in O_i$ is uniquely identified with a hash value h_j. A p_i maintains a limited number of connections to direct peers in its neighbor list. Each node p_i in the overlay network S takes any one or a combination of the following three roles:

- *Media Source*: p_i shares its media objects O_i, responds to media search queries, and acts as a producer of media streams.
- *Media Receiver*: p_i initiates the search for a specific media object, constructs streaming paths, and acts as a consumer of media streams.
- *Media Transcoder*: p_i provides stream data transformation components S_i, shares its local processor cycles and network bandwidth, and responds to service search queries.

Our system can be built on top of both structured and unstructured overlays, or even in a centralized fashion, and take advantage of their existing routing and resource discovery mechanisms. For example, if coupled with DHT-based P2P networks (e.g. Chord, Pastry) , our system can locate media sources and transcoder peers within $log(N)$ steps, where N is the size of network. With unstructured P2P networks (e.g. Gnutella), our streaming system keeps minimal information but employs the underlying flooding technique for resource discovery. The latter approach results in unbounded resource discovery time and high message overhead, but peers can enter and leave the system without complications.

Considering the wide availability of Gnutella systems, in this paper we focus on unstructured P2P networks. Media receiver nodes utilize Gnutella's $Query$ and $QueryHit$ messages to initiate the streaming graph composition (section 3.1), discover media objects and transcoder components. Furthermore, media receiver nodes might need to send control messages to media source nodes. We leverage Gnutella's message mechanism with a $Control$ message type (section 3.3) to send feedback information from a receiver to the sources.

2.2. Service Graph

A service graph G_i (*i.e., streaming graph* for this particular service) for a streaming session $stream_i$ represents the streaming dependency between media producers, media transcoders and the media receiver, and the corresponding resource requirements, as shown in Figure 1 (B). In general, a streaming session might have several sources located at multiple streaming paths, and several transcoders might contribute to a specific streaming path.

We assume the system provides on-demand multimedia streaming, where each requesting peer $p_r \in S$ searches for a media object ob_r by name. The returned results contain the media objects' properties, $A : (bitrate\ b,\ resolution\ r,\ codec\ c,\ hash\ h)$, and are identified by their hash values. p_r selects one or multiple source nodes $s_0,\ s_1,\ ...,\ s_i$, where $s_i \in S$, to stream the same media object based on the source nodes' resource constraints. Specifically, multiple source nodes may be selected such that $\sum_i b_{s_i} > b$, where b is the required bandwidth for streaming the select object (i.e. the object's bitrate). $stream_i$ is characterized by a set of parameters about playback requirements, $B : (bitrate\ b_r,\ resolution\ r_r,\ codec\ c_r)$ that describes p_r's local resource constraints or the user's preferences. If p_r's streaming requirements B cannot be satisfied by the media object ob_r's properties A, p_r needs to recruit collaborative peers $T : t_0,\ t_1,\ ...,\ t_j$ that provide the necessary transcoding services. A service graph G_r is composed by the receiver p_r, which connects p_r, s_i and t_j and satisfies the streaming requirements B. Thus it's p_r's responsibility to allocate data segments and receive different sub-streams of the multimedia data from different s_i, and realize the service graph G_r by constructing all streaming paths on top of the underlying peer-to-peer substrate.

For example, in Figure 1 (B), the selected media object requires 400Kb streaming bit-rate and a display resolution of 600x480, which does not match the receiver's resource constraint of 150Kb and 320x240. Thus the receiver recruits two media source nodes that offer 300Kb and 100Kb bitrate, respectively. In addition, transcoders are also selected that provide the appropriate bandwidth and processor cycles for the necessary transcoding operations.

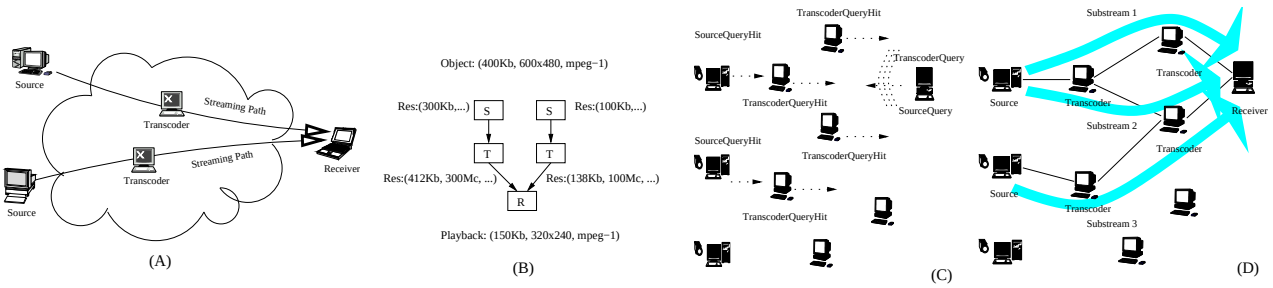

Figure 1. **(A) Illustration of a streaming session where streaming data take multiple paths and are transcoded before reaching the receiver. (B) The corresponding service graph of the session, connecting sources S, transcoders T and the receiver R. (C) Discovery of sources and transcoders. (D) Connection establishment and data flow.**

2.3. Transcoding Service

A transcoding service performs a limited number of data transformation operations on incoming independent media units (e.g. groups-of-pictures of MPEG streams), such as the change of encoding algorithms, streaming bit-rate, frame-rate and display resolution. Specifically, each media transcoding operation consists of a set of functions $F : \{f_1, ..., f_n\}$, where each f_i has a number of implementations with diverse output qualities and corresponding resource requirements. The complexity of the whole operation is controlled by selecting an implementation for each function $f_i \in F$, and thus the transcoding quality level and the corresponding resource budget are tunable. Our solution relies on the fact that media streaming data can be fragmented into independent units, then transformed on different streaming paths and merged at the end receiver. This holds at least for MPEG streams that consist of a sequence of independent media units, called group-of-pictures (GOPs).

For simplicity, we can assume one transcoding operation for each media unit (*e.g.*, bit-rate reduction or adjustment of resolution). When media units from different streams arrive at a transcoding node concurrently, they compete for CPU and bandwidth resources. We model them as a set of concurrent periodic tasks $J : \{j_1, ..., j_l\}$. In the transcoding process, a transcoder delivers the service according to local resource constraints (CPU and available bandwidth). We further assume that the QoS (e.g. PSNR) of the delivered service by each transcoder can be approximated using m discrete levels: $Q : \{q_1, ..., q_m\}$. Usually a higher QoS level means more consumption of CPU cycles in transformation. The resource requirements of a transcoding task j_j can be depicted as $\{c_j(q_i), b_j(q_i)\}$ where c_j and b_j are processor cycle and bandwidth requirements, respectively.

For each multimedia stream, a quality level, $q_i \in Q$, indicates a different service satisfaction, and could be associated with a utility value $u(q_i)$. We also assume that each node allocates a portion of its available bandwidth to media units that do not require any transcoding services (*e.g.*, media units being served in the upstream), we indicate that portion of bandwidth as g_k for streaming path k at a node. Then the available bandwidth for the transcoded multimedia streams is $b_k = t_k - g_k$, where t_k is the total available bandwidth along path k.

3. Adaptive Receiver-Coordinated Multimedia Streaming

The operation of our proposed coordinated multimedia streaming mechanism consists of the four following techniques: (a) *Streaming graph composition*, (b) *Local quality adaptation*, (c) *Feedback-based coordination* and (d) *Streaming graph restructuring*, which we discuss in the following subsections.

3.1. Streaming Graph Composition

The receiver initiates and realizes the streaming graph construction in four phases.

Discovery Phase. The receiver creates a query for a media object *SourceQuery(object)*, specifying the object's name. The sources offering the media object respond with a QueryHit message *SourceQueryHit(bitrate,resolution,codec,hash,bandwidth)*, indicating the bitrate, resolution, codec, and hash value of the offered object, as well as their available bandwidth (Figure 1 (C)). By comparing the offered bitrate, resolution, and codec to its needs, the receiver determines whether a transcoding service is needed. In this case, it creates a Query for transcoders that perform that particular service *TranscoderQuery(service)*. The query contains the name of the service (e.g. MPEG-2, 512, 800x600 to MPEG-4, 256, 640x480). Transcoders that offer the requested service respond with a QueryHit *TranscoderQueryHit(incoming*

bandwidth,outgoing bandwidth,available cycles) indicating their available incoming and outgoing bandwidth and their current available CPU cycles (Figure 1 (C)).

Selection Phase. Based on the responses of the peers offering the media object, the receiver determines which of them offer the same object (determined by the hash value), with the same bitrate, resolution, and codec, so that streaming from all the media sources can be combined. The receiver also determines how many sources are needed to satisfy the bandwidth requirement of the streaming session. According to the above the receiver selects the set of sources that will participate in the streaming graph.

Based on the responses of the peers offering the needed transcoding service, their available bandwidth and CPU cycles, the receiver determines how many transcoders are needed. It then selects a set of the less loaded transcoders so that the bandwidth requirements of the streaming session are satisfied.

Connection Establishment Phase. Having selected the sources and the transcoders that will participate in the streaming graph, the receiver initiates the connection establishment. It creates connections to all the sources. Those connections will be used as control channels to transfer the quality feedback messages. It also sends graph composition messages to the sources, specifying to which transcoders to connect to. The service graph is comprised of streaming paths, each one of them starting from a source, containing one or more transcoders and ending with the receiver. Each of the graph composition messages sent to a source contains all the transcoders that will participate in the particular streaming path. Hence, by propagating the graph composition message from the source to the transcoders, all the connections needed for each path are established consecutively. The streaming graph is thus created, as shown in Figure 1 (D).

Data Assignment Phase. The receiver assigns media units to the different streaming paths proportionally to their aggregated offered bandwidth. Thus, sources with more available bandwidth will serve more media units. Serving parts of the same stream from different sources is possible, since the stream is divided in independent media units. Each sub-stream has a period, allowing them to be scheduled in the transcoders. The receiver stores several seconds of the stream in a buffer, sorted according to their sequence number. This buffering makes it possible to absorb the jitter before playing back the stream.

3.2. Local Quality Adaptation

Media streaming is resource demanding in that it requires a large amount of bandwidth and processor cycles during a session. Without enough resources, media data will not be delivered to the receiver on time. There are two schemes to solve this problem: 1) resource reservation and 2) quality adaptation. However, only soft resource reservation is viable in dynamic peer-to-peer systems in which the reservation states might be purged periodically. Therefore adaptation that trades quality with resource usage is an essential component in media streaming over P2P. As described in section 2.3, the output quality of a transcoding operation is approximated with m discrete levels, $Q : \{q_1, ..., q_m\}$. All transcoders can share this knowledge, because it can be described using meta-data of the transcoding components, and installed locally together with them.

Stream Data Structures. We assume that each transcoder node maintains a transcoding task queue JQ that keeps incoming media units requiring transcoding operations, a resource budget table RB, and a stream table QT. RB describes the relationship between transcoding operations and corresponding estimated resource usage under a particular quality level. Such estimation could be obtained through pre-configured profiles, or on-line profiling. Each entry in the RB is of the following format, $< op_j,\ period_j,\ (c_j(q_1), b_j(q_1)), ..., (c_j(q_m), b_j(q_m)) >$, in which $(c_j(q_i), b_j(q_i))$ denotes the average processor cycles and bandwidth required for transcoding operation op_j with period $period_j$ on a media unit. Table QT contains streaming session information. Each entry of QT is denoted as $< stream\_id\ sid,\ sub\_stream\_id\ ss,\ lowest\_quality\ q_l,\ highest\_quality\ q_h,\ \ current\_quality\ q_r\ >$. Note that a stream session might possess multiple streaming paths identified by ss. The pair $<\ q_l, q_h\ >$ limits the preferred quality levels and initially is set to $< 1, m >$. The sizes of RB and QT are proportional to the number of concurrent streaming paths that need transcoding operations at this node, thus practically small. In addition, RB is updated only when the transcoder joins or leaves a streaming session; QT, on the other hand, is also updated when feedback coordination information is received (see section 3.3), or local resources change significantly. This scenario is discussed later in this subsection.

Utility Functions. To provide the best possible quality for all competing streaming sessions, transcoder nodes monitor the local resource conditions and try to optimize the output quality of each transcoding operation. To monitor the available bandwidth along each streaming path, we could use a simple technique based on the TCP congestion control mechanism, as discussed in [6]. To evaluate how much benefit is gained with a given output quality of an operation op_j for any stream, each transcoder maintains a discrete utility function $u = u_j(q_i)$, denoting the utility value of the transcoding media unit j with quality level i. The selection of the function u is operation specific (e.g. linear function), with a common characteristic being that the higher the quality level, the larger the benefit.

Utility Optimization. Several issues affect the selection of the output quality of a transcoder node t_i and trigger a new solution to an optimization problem: 1) The available bandwidth of a streaming path from t_i to its next hop, shared with extraneous traffic, 2) The CPU capacity of t_i shared with other applications with unpredictable behaviors, and 3) The number of streaming paths that traverse t_i, changing dynamically when new streaming sessions occur or existing sessions finish. To schedule the transcoding tasks, the transcoder node uses the Earliest Deadline First (EDF) scheduling algorithm. Thus, the local quality adaptation problem then is: Given all sub-streams ss described in RB and QT tables, find a set of quality levels $< q_r(ss_1), ..., q_r(ss_n) >$ such that the total utility value from all transcoding operations is maximized. This is mathematically depicted as follows:

$$Maximize \sum_j w_j * u_j(q_i) \quad (1)$$

$$Subject\ to \sum_j \frac{c_j(q_i)}{period_j} \leq f_p \quad (2)$$

$$b_j(q_i) \leq b_j, \quad j = 1, ..., n \quad (3)$$

$$q(ss_j) \in [q_l(ss_j), q_h(ss_j)],\ j = 1, ..., n \quad (4)$$

In the formulae, w_j is an importance weight associated with each streaming session. If all streaming sessions are considered equal, then $w_j = 1, \forall j$. In addition, b_j is the available bandwidth from the transcoder node to the next hop along the streaming path. f_p are the CPU cycles available in a second. Since other running applications are sharing CPU and bandwidth with the transcoding service, f_p and b_j could be just a portion of available resources. Equation 1 denotes the optimization objective to maximize the total utility value. Equations 2 and 3 indicate that the cycle requirements should be less than the available CPU capacity using the EDF algorithm, and that the bandwidth requirement of any particular streaming path should not exceed the available bandwidth along that path. Equation 4 ensures that the quality level selected for each stream should be within the preferred range. This problem can be solved using dynamic programming in time that is proportional to the number of streams and the utility values of the media units. The solution is afterwards used to update the current quality level fields (q_r) of the QT table. The overhead of the local optimization depends on peer dynamics as discussed above, but we can limit the frequency of this computation to an acceptable level.

Media Unit Drop. In the simplest case, where the utility value is linearly proportional to the selected quality level, several sets of solutions are viable by repeatedly reducing the quality levels of any stream. To eliminate the possibility that a single stream is punished, the algorithm reduces the quality level of a randomly picked stream in each iteration until all resource constraints are met. Also note that it is possible that no solution exists even if all minimum quality levels q_l are selected. In this case, the transcoder node estimates the transcoding queue length, and rejects any incoming media units when the queue is too long. Specifically, assuming the task queue consists of media units $\{(sid_1,\ num_1), ..., (sid_n,\ num_n)\}$ where num_i indicates the number of media units in the queue that belong to stream sid_i, then new media units are rejected when $\sum_i c_i(q) * num_i \geq f_p$.

3.3. Feedback-based Coordination

As transcoder nodes locally adapt the output quality levels of the streams being transformed, the user perceived streaming quality can fall out of acceptable limits. This is because the overall streaming quality is determined by resource conditions along all the streaming paths in a session, and could fluctuate rapidly when adjacent media units are coming from different streaming paths. Without global information, however, services offered by each transcoding node are based on locally monitored resource constraints, and thus are best-effort, in the sense that they may conflict with the quality preferred by end users and waste resources. When the quality of a sub-stream degrades because of resource overload, it might be better to let other sub-streams in the same session degrade qualities as well, even if these sub-streams have enough resources at their disposal. This way the quality level of the whole stream will be consistent. Thus we need a synchronization mechanism that coordinates the transcoding operations for a specific session in order to prevent frequent quality fluctuations.

Feedback coordination. In our approach we utilize the quality information observed by the stream receiver to coordinate the transcoding behaviors along each streaming path, and guide their resource utilization, in order to stabilize the quality of the whole stream. The basic idea is that the receiver R calculates a preferred quality interval $q_{interval} = (q_l, q_h)$ dynamically, indicating that the streaming quality should not exceed the bounds. This interval is based on two factors: 1) the media unit quality of the corresponding sequence 2) the media unit miss rate, indicating how many media units are dropped by the last hop transcoder, or have missed their play-back time. This quality interval $q_{interval}$ is encapsulated in a $Control$ message, and sent through a separate control channel to all media sources $s_i \in S$ (Figure 2 (A)). Upon reception of this feedback information, a source node attaches $q_{interval}$ to the next media unit that flows through each transcoder node along the streaming path (Figure 2 (B)). Consequently, a transcoder updates the corresponding entries in its QT table with the new $q_{interval}$ and uses it for the next local adaptation computation.

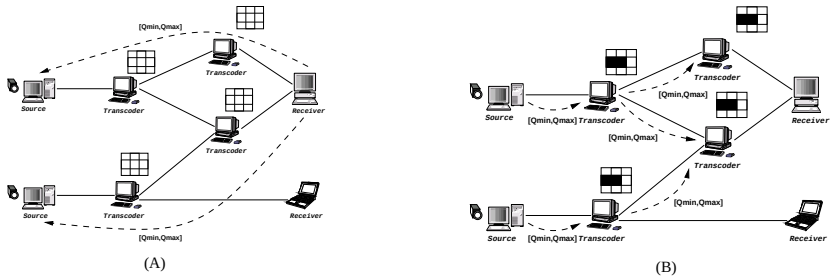

Figure 2. **Feedback based coordination. (A) The preferred quality interval is sent through control channels to the media sources. (B) That feedback information flows to the transcoder nodes, piggybacked on the media units to be transformed.**

Quality interval calculation. Each receiver node owns a playback buffer and keeps track of the sequence number seq_{next} of the next media unit to be played back. When a media unit arrives, its sequence number seq is compared with seq_{next} and is dropped if $seq < seq_{next}$. Otherwise, it is inserted into the buffer and sorted according to seq. During playback, the receiver always fetches the media unit with the smallest seq. To calculate the preferred quality intervals, the receiver maintains a vector V_{qos} that stores the quality level values for the l recently played-back media units. Note that, for media units that were dropped along the streaming path, or that missed their playback time, a quality value of 0 is inserted into V_{qos}. Periodically, the receiver node r calculates the quality interval $q_{interval}$ with lower bound $q_l = \sum_i \frac{V_{qos}(i)}{V.size()}$, $i = 1, ..., l$, and upper bound $q_h = q_l + q_{window}$, where q_{window} is a pre-configured threshold value. The calculation of q_l takes into account dropped or missed media units with zero values, which indicate that either a streaming path is congested, or that a transcoder is overloaded. The frequency of the calculation of the quality interval depends on the number and frequency of user requests and the QoS level fluctuations. Furthermore, q_{window} is selected to be a small constant to limit the fluctuation of a stream's quality within a short time range.

3.4. Streaming Graph Restructuring

Streaming Graph Destruction. When the streaming process has finished, or when the receiver decides to terminate it, it notifies the sources and transcoders associated with that stream. These are released and free the corresponding entries in the streaming parameters tables. The notification of the participating nodes is done in a way similar to the one used in the graph construction. Using the control connections, the receiver notifies the sources, which in turn notify the transcoders they are connected to and they notify the rest of the transcoders consecutively.

Streaming Graph Restructuring. The quality of the streaming is mainly affected by overloaded or failing peers. Once the receiver detects a degradation in quality that is too severe to be improved using feedback hints, it initiates a restructuring of the streaming path. Since the receiver keeps track of the streaming paths and the quality levels of the corresponding sub-streams in the service graph, it is able to identify which sub-stream lacks in quality. Hence, only the corresponding streaming path and not the whole graph needs to be restructured. Thus, the procedure described for the streaming graph composition is used for composing a streaming path with the characteristics of the one that is being substituted. Since our system does not maintain backup service graphs, the overhead of the service graph maintenance is low. A peer failure cannot be handled instantly, but the quality of the end result is only affected to the extent it depends on the peer that failed.

Thus, the high-overhead, infrequent restructuring loop is used to adapt to node failures or severe quality degradations, whereas the low-overhead, frequent feedback loop is used to adapt to small quality fluctuations.

4. Performance Evaluation

4.1. Simulation Setup

To evaluate the performance of our coordinated media streaming and transcoding, we implemented a simulator that utilizes an underlying physical topology of 1476 routers generated with GT-ITM, with a topology diameter of around 750ms. Table 1 summarizes our simulation parameters.

To simulate different roles of nodes in the system, we assume n_s represents how many nodes can offer media objects, which have connection bandwidth between 50Kb and 200Kb, and n_t denotes the number of transcoder nodes, with connection bandwidth between 400Kb and 2Mb and processor capability between 400M and 800M cycles per second. Note that any node can initiate a streaming session in the simulation. In addition, as the cross traffic has

an impact on the available bandwidth of each overlay link, we simulate how congested a link is at any given time using random probabilities. We simulate a transcoding service (bit-reduction), where the execution time of an operation is proportional to the size of a media unit, which varies with a small random number. We simulate 10 levels of output quality for all streams and utilize the same linear utility function that is proportional to the output quality level. Currently, transcoder nodes solve the optimization problem every second.

For comparison, we measured the performance of our *coordinated* scheme against both an *adaptive* and a *static* scheme. In the static scheme, media units are always transcoded with the maximum quality level, and resource conditions are not considered. The adaptive scheme, on the other hand, adapts the output quality of media units between the minimum and the maximum, but does not utilize coordination information. The coordinated scheme employs both local adaptation in the transcoders and a feedback loop coordinating them globally.

Simulation Time	200 seconds
Number of Streams	80
Receiver Buffer Time	4 seconds
Quality Interval Size	3
Media Object Bitrate	(200, 250, 300, 350, 400) Kb
Req. Streaming Bitrate	(50, 75, 100, 125, 150) Kb
# of Total Nodes	1000
# of Source Nodes	300
# of Transcoder Nodes	300

Table 1. Simulation parameters.

We used the following metrics to evaluate the performance of our system:

1. **Streaming quality** ϖ. This is represented by the sequence of quality levels when media units are played back at designated time.
2. **End-to-end delay** ξ. The end-to-end delay of a media unit is the sum of the queuing time, transcoding time and network transmission time.
3. **Streaming bit-rate** γ. The bit-rate represents how much data arrives at the receiver on time every second. This is affected by dropped units and units that missed their deadlines.

4.2. Results and Analysis

We compare the three different strategies (static, adaptive and coordinated streaming) and then we focus on the coordinated streaming, by investigating the effects of the buffer size, the quality interval size, and the number of streams in the system.

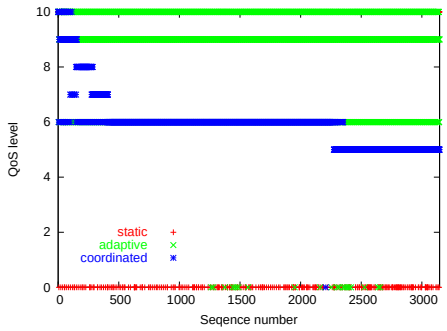

Figure 3. QoS level fluctuation of the media units of a stream.

4.2.1. Comparison of the different strategies. We first compare the behavior of the three different streaming strategies over time, for several metrics:

Quality fluctuation. As shown in Figure 3, static streaming always tries to offer the highest quality level (10). When the transcoders are overloaded they either drop media units, or they delay their transcoding and they miss their deadline. The result in those cases is a quality of 0. Hence the quality level of static streaming fluctuates between 10 and 0, which clearly is not acceptable. In adaptive streaming transcoders try locally to maximize the utility value of their operations. Since their efforts are not coordinated though, the overall result for a stream presents very frequent fluctuations of the quality level. This is also disturbing for the user, even though the fluctuation is between adjacent quality levels. Coordinated streaming uses the feedback hints to orchestrate the efforts of all the transcoders for local adaptation. This way the fluctuation of the quality level is avoided. When some of the transcoders are unable to deliver a certain quality level anymore, the coordination mechanism ensures that the rest of the transcoders also follow. The result is a *smooth* degradation in the overall quality.

End-to-end delay. The end-to-end delay for the three streaming strategies is presented in Figure 4. Since static streaming always tries to offer the highest quality level, several media units are dropped or missed and are depicted as having a 10000ms latency. Furthermore, even media units that reach the receiver, while still being useful, may have high delays. One reason for that is that transcoders are overloaded trying to offer always the highest quality. Adaptive streaming decreases the end-to-end delay by allowing the transcoders to change the quality level they are offering. The best end-to-end delay

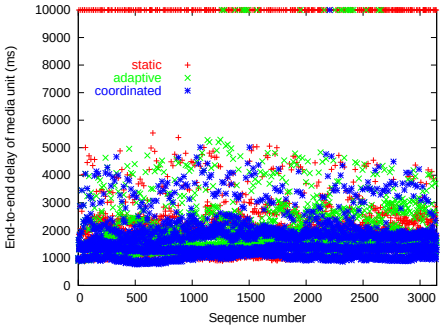

Figure 4. **End-to-end delay of the media units of a stream.**

however is achieved by coordinated streaming. The communication latency to transfer the feedback hints is compensated by the gains this information offers: Transcoders spend less time trying to offer high quality which is not going to make a difference in the end result and since they are less loaded have smaller queue latency too.

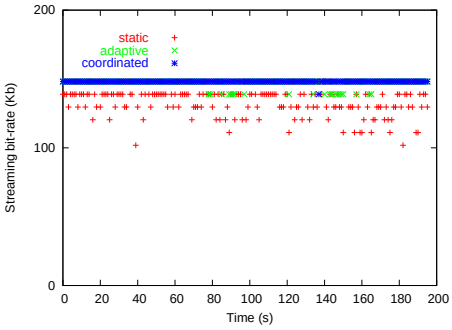

Figure 5. **Streaming bit-rate of a streaming session over time.**

Streaming bit-rate. The streaming bit-rate offered by the three strategies is displayed in Figure 5. Again, static streaming has the worst performance, due to the large number of dropped media units, or of media units that have missed their deadline. Adaptive streaming presents less fluctuation in the delivered streaming bit-rate. Yet it is again outperformed by coordinated streaming, which due to the global adaptation mechanism is able to deliver almost all media units in time, thus offering an almost constant bit-rate.

4.2.2. Comparison of different buffer sizes for coordinated streaming. Having shown that coordinated streaming outperforms the other strategies in every aspect, we now investigate the effect of the receiver buffer size on its performance.

Streaming bit-rate. The streaming bit-rate for four different buffer sizes is presented in Figure 6. As expected, the

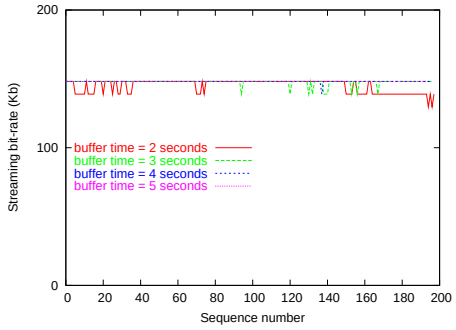

Figure 6. **Streaming bit-rate of a stream for various buffer sizes.**

largest buffer is able to offer a constant streaming rate, not letting media units to be dropped or delivered too late. Even the second largest buffer (4 seconds in our case) is absorbing the jitter almost all the time, even for a heterogeneous and dynamic system like the one presented here.

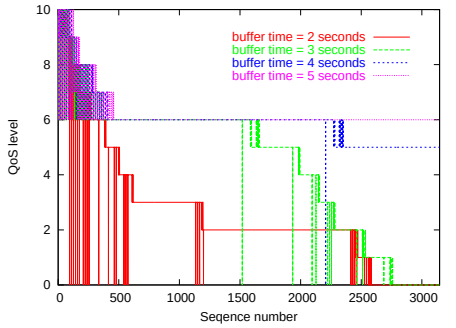

Figure 7. **QoS level fluctuation for various buffer sizes.**

Quality fluctuation. Apart from the streaming bit-rate, the buffer size also affects the QoS level, as is shown in Figure 7. A large buffer absorbs jitter, eliminating media units that are dropped or miss their deadlines. Hence, the quality level needs not to be adapted (degraded) often. However, even when using a large buffer, coordination is needed at least in the beginning of the streaming session, to determine a suitable quality level that will enable the system to avoid quality fluctuations.

4.2.3. Comparison of different quality interval sizes for coordinated streaming. Quality fluctuation. We examine the effect of the quality interval size on the quality level, as shown in Figure 8. Using a large interval size in the coordination mechanism results in larger quality fluctuations, since transcoders can choose from a variety of quality levels to offer. A small interval size on the other hand may result in more fluctuations before the optimal quality level for

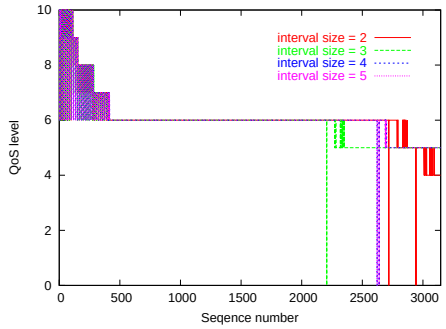

Figure 8. **QoS level fluctuation for various feedback interval sizes.**

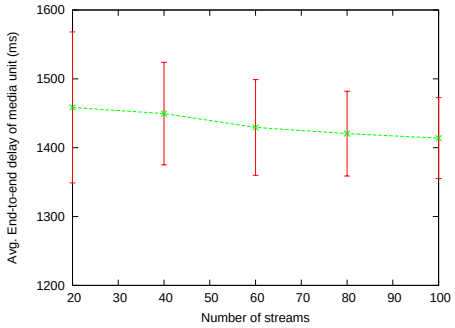

Figure 10. **Average end-to-end delay of the media units of all streams with changing system load.**

the current system state is reached. A modest interval size can help the system adapt quickly and yet not let the quality fluctuate a lot after the system has stabilized.

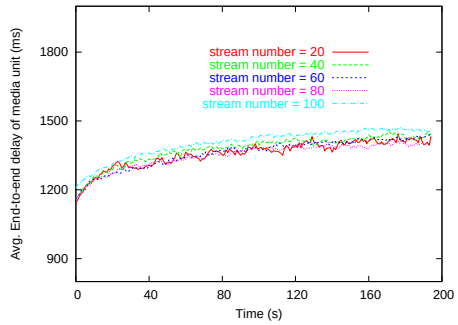

Figure 9. **Average end-to-end delay of the media units of all streams for various numbers of streams.**

4.2.4. Scalability of coordinated streaming. End-to-end delay. To measure the scalability of coordinated streaming, we vary the number of concurrent media streams in the system from 20 to 100, and calculate the average end-to-end delay as $\frac{\sum_{i=1}^{n} \xi_i}{n}$ for all the media units. Figure 9 shows that for a specific topology, the average end-to-end delay is not greatly affected by the number of streams.

To further investigate the effect of the number of streams on the system performance, we ran each experiment 10 times, using different topologies and we present the average results with 95% confidence interval in Figure 10. We observe that the average end-to-end delay slowly decreases when the system load increases. This happens when transcoders degrade their output quality levels, thus significantly reducing the transcoding time and the corresponding queuing time of the media units. This reduction compensates for the increased media unit arrival rate, which is a result of the larger number of streams. This shows that coordinated streaming can accommodate increasing numbers of streams offering acceptable streaming qualities.

5. Related Work

Existing work on structured [19, 20, 22] and unstructured [8] peer-to-peer overlays has focused on mechanisms for the efficient sharing of data objects. A few recent research efforts discuss the use of such overlays to provide other services, like media streaming.

The service graph construction has been the focus of works like PROMISE [12] and SpiderNet [9]. SpiderNet uses a probing protocol to setup the service graph. PROMISE focuses on a topology construction to support media streaming from multiple senders to one receiver. These mechanisms work well. Yet, we can further improve their effectiveness –even in cases of inaccurate bandwidth estimations– by using receiver feedback.

Systems like PROP [11] and MediaNet [13] coordinate service provision by making use of a central scheduler. Layered encoding and multiple description encoding of data streams have been proposed as techniques for cooperative playback from multiple senders [7, 1]. Under constrained bandwidth, the quality of the received stream is lowered by omitting layers. However, peers are heterogeneous in a lot more aspects, like their codec requirements or preferences. Those needs cannot be addressed by layered encoding, but are solved by our transcoding-based approach.

Multicast topologies [4, 23, 14, 10] have also been proposed as a mechanism for efficient content distribution in peer-to-peer networks. However, neither differences in user requirements nor the quality experienced by the users are taken into account by multicasting in itself. Hence, multicasting focuses more on content distribution, rather than

on-demand content delivery, which is the focus of our approach. Distributed video streaming has been studied in contexts other than peer-to-peer [16, 17]. They usually employ centralized approaches and do not handle dynamic streaming requirements.

Q-RAM [15] also uses discrete quality levels to address the local optimization problem. Our local adaptation mechanism builds on top of that approach and employs it in a peer-to-peer environment.

6. Conclusions

In this paper we have proposed a novel P2P streaming architecture that deploys transcoding services to satisfy users' preferences. To meet the QoS requirements of the streaming, transcoder nodes adapt to variable local resource conditions and optimize their resource usage. Receiver nodes calculate feedback information and coordinate transcoding operations in their own streaming sessions. Our mechanism is fully distributed, uses only local knowledge and scales well with the size of the system. Simulation results validate our mechanism and show that it provides real-time guarantees to media streaming in P2P systems.

References

[1] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. On multiple description streaming with content delivery networks. In *Proceedings of IEEE INFOCOM 2002*, June 2002.

[2] V. Balasubramanian and N. Venkatasubramanian. Server transcoding of multimedia information for cross disability access. In *Proceedings of the ACM/SPIE Conference on Multimedia Computing and Networking*, January 2003.

[3] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proceedings of IEEE INFOCOM 2003*, 2003.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proceedings of SOSP 2003*.

[5] S. Chandra, C. S. Ellis, and A. Vahdat. Differentiated multimedia web services using quality aware transcoding. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[6] F. Chen and V. Kalogeraki. RUBEN: A technique for scheduling multimedia applications in overlay networks. In *Proceedings of the IEEE Global Telecommunications Conference 2004 (Globecom'04)*, November 2004.

[7] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *Proceedings of the 13th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'03)*, June 2003.

[8] Gnutella Protocol Development . http://rfc-gnutella.sourceforge.net/, 2003.

[9] X. Gu and K. Nahrstedt. SpiderNet: An integrated peer-to-peer service composition framework. In *Proceedings of the 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*, June 2004.

[10] J. Guo, Y. Zhu, and B. Li. CodedStream: Live media streaming with overlay coded multicast. In *Proceedings of the ACM/SPIE Conference on Multimedia Computing and Networking 2004 (MMCN'04)*, January 2004.

[11] L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang. PROP: a scalable and reliable p2p assisted proxy streaming system. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, March 2004.

[12] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using collectcast. In *Proceedings of the ACM Multimedia 2003 (MM'03)*, November 2003.

[13] M. Hicks, A. Nagarajan, and R. Renesse. User-specified adaptive scheduling in a streaming media network. In *Proceedings of the 6th IEEE Conference on Open Architectures and Network Programming (OPENARCH'03)*, April 2003.

[14] X. Jiang, Y. Dong, D. Xu, and B. Bhargava. GnuStream: A P2P media streaming prototype. In *Proceedings of the 2003 IEEE International Conference on Multimedia and Expo (ICME'03)*, July 2003.

[15] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource QoS problem. In *In Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, December 1999.

[16] T. Nguyen and A. Zakhor. Distributed video streaming over the internet. In *Proceedings of the ACM/SPIE Conference on Multimedia Computing and Networking 2002*, January 2002.

[17] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of the 12th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, May 2002.

[18] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. An application level multicast infrastructure. In *USENIX Syposium on Internet Technologies and Systems*, 2001.

[19] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[21] P. Shenoy and P. Radkov. Proxy-assisted power-friendly streaming to mobile devices. In *Proceedings of the ACM/SPIE Conference on Multimedia Computing and Networking 2003 (MMCN'03)*, January 2003.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[23] D. Tran, K. Hua, and T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM 2003*, April 2003.

[24] Y.Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM*, 2001.