Optimal Peer Selection for Minimum-Delay Peer-to-Peer Streaming with Rateless Codes

Chuan Wu, Baochun Li
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, ON M5S 3G4, Canada
{chuanwu,bli}@eecg.toronto.edu

ABSTRACT

Due to the limitation of peer upload capacities and high bandwidth demand of multimedia applications, optimal peer selection to construct high-quality streaming topology represents a major challenge in peer-to-peer streaming. In this paper, we propose a fully distributed algorithm to achieve optimal peer selection and streaming rate allocation, which minimizes end-to-end latencies in the streaming sessions. We design this efficient distributed algorithm based on the solution to a linear optimization model, which optimizes towards a latency-related objective to decide the best streaming rates among peers. Combining this optimal peer selection algorithm with our coding scheme based on rateless codes, we obtain a complete, fully decentralized minimum-delay peer-to-peer streaming scheme. Our scheme is resilient to network dynamics that is characteristic in peer-to-peer networks. The validity and effectiveness of our approach are demonstrated in extensive simulations.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*

General Terms

Algorithms, Design, Performance

Keywords

Peer-To-Peer, Media Streaming, Peer Selection, Optimization, Rateless Codes

1. INTRODUCTION

The limited bandwidth capacities in peer-to-peer networks pose a significant technical challenge in peer-to-peer media streaming. As nodes in peer-to-peer networks reside at the edge of the Internet, they usually have limited availability of upload and download capacities. In addition, due to peer heterogeneity, the available per-node bandwidth may differ by at least an order of magnitude. For delay-sensitive media streaming applications, the typical streaming bit rates in modern streaming codecs must be accommodated for all the peers in a streaming session, in order to ensure their uninterrupted streaming playback. Therefore, it is typical for a peer node to parallelly download from multiple upstream peers, in order to improve the overall bandwidth availability. In this case, a critical question arises: What is the best way for the peer nodes to select the upstream peers and allocate the streaming rates among the selected peers, such that a specified aggregate streaming bit rate is satisfied and the end-to-end latencies are minimized at the receivers? It is a nontrivial problem to obtain a feasible *peer selection and streaming rate allocation* strategy which guarantees all requesting peers can acquire the streaming bit rate of the session, not to mention that which minimizes end-to-end latencies.

When the streaming rates from selected upstream peers have been optimally allocated, the next question to answer is how to assign the media contents to be delivered along each link. In the constructed mesh streaming topology featuring parallel retrievals, there are always risks that the same contents may be unnecessarily supplied by multiple upstream peers. Therefore, the peer nodes need to reconcile differences among the sets of content segments they hold. A *content assignment* scheme, which schedules which content segment to retrieve from which upstream peer, needs to be designed to minimize the delivery redundancy [7, 22].

This paper focuses on tackling the former problem: We first formulate the optimal peer selection problem as a linear optimization problem, which guarantees bandwidth availability and minimizes streaming latencies; We then design an efficient and decentralized algorithm to solve the problem, based on the Lagrangian relaxation technique and the subgradient algorithm. The main contribution of the paper is the derived optimal peer selection algorithm, which computes the optimal streaming rates on the peer-to-peer links in a fully decentralized and iterative fashion. Our algorithm is also reactive to network dynamics, including peer joins, departures and failures.

In our previous work [21], we have proposed an efficient coding scheme based on rateless codes to address the later problem of delivery redundancy and reconciliation. Based on the loss resilience and "ratelessness" properties of rateless codes, our scheme provides excellent resilience to network dynamics, and guarantees that no duplicated contents exist in the network. This completely eliminates the need for set reconciliation and content assignment on the links, which otherwise involves high computation and messaging overhead [7].

Combining the optimal peer selection algorithm with the rateless-code coding scheme, we are able to derive a complete resilient and optimal peer-to-peer streaming solution.

The remainder of this paper is organized as follows. In Sec. 2, we formulate the minimum-delay optimal peer selection problem as a linear optimization problem, and analyze its combination with the rateless-code coding scheme. In Sec. 3, we present the efficient distributed algorithm to solve the problem and compute the optimal streaming rates over the links. We show the adaptation of the algorithm to peer dynamics in Sec. 4. Simulation results are presented in Sec. 5. We discuss related work and conclude the paper in Sec. 6 and Sec. 7, respectively.

2. PROBLEM FORMULATION

In this paper, we consider a peer-to-peer streaming session with one streaming *source* and multiple participating *receivers* (Fig. 1). A subset of the receivers retrieve the media contents directly from the source, while the others stream from one or more receivers in the *upstream*. When a new peer joins the session, it is bootstrapped with a list of known peers in the session, who may serve as the initial set of upstream peers. This constructs the initial *mesh* overlay topology for the streaming session. Such a mesh topology can be modeled as a directed graph $G = (N, A)$, where N is the set of vertices (peers) and A is the set of directed arcs (directed overlay links). Let S be the streaming source, and let T be the set of receivers in the streaming session. We have $N = S \cup T$.

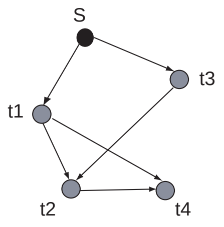


Figure 1: An example of the peer-to-peer streaming network model: S - the streaming source, $t1, t2, t3, t4$ - the receivers.

Our objective is to stream live multimedia contents, coded to a constant bit rate bitstream with a current generation codec such as H.264/AVC, H.263 or MPEG-4. For such live media streaming applications, a minimized end-to-end latency at each receiver is significant to guarantee the high liveness of the streaming media, as also emphasized by existing work [20, 3]. Therefore, it is desirable to design an optimal peer selection strategy that constructs an optimal streaming topology, on top of which the end-to-end latencies at all receivers are minimized.

In the following, we set up linear programming models to address the optimal peer selection problem. We formulate the objective functions to reflect the minimization of streaming latencies at the receivers, and the constraints to reflect the capacity limitations in the peer-to-peer network. We first formulate the linear program for the single session streaming case, and then give its extension to the multiple session streaming case. We motivate our LP formulations by analyzing a unicast streaming session from the streaming source to a receiver.

2.1 LP for unicast streaming

The unicast flow from the streaming source to a receiver is a standard network flow following the nice property of flow conservation at each intermediate node. Let r be the end-to-end streaming rate of this unicast flow, c_{ij} be the delay and f_{ij} be the streaming rate on the link (i, j). Fig. 2(a) depicts an example of a unit unicast flow from S to t_4 in the network as shown in Fig. 1, with $r = 1$, $c_{ij} = 1, \forall(i, j) \in A$ and the streaming rates f_{ij} labeled on the arcs. As shown in this example, a unicast flow may flow along multiple peer-to-peer paths from the source to the receiver. Thus it can be viewed as composing of multiple fractional flows, each going along a different path. Notice that different paths may share some peer-to-peer links and the streaming rate on each common link is the sum of the rates of all fractional flows that go through the link. Fig. 2(b) illustrates the decomposition of the unit unicast flow into three fractional flows, with the rates of 0.2, 0.3 and 0.5 respectively.

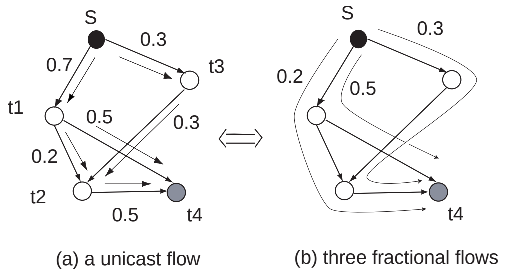


Figure 2: An example of a unicast flow from S to t_4 and its decomposition into three fractional flows.

For the receiver of such a unicast flow, we calculate its average end-to-end latency, as the weighted average of the end-to-end latencies of all the fractional flows. The weight for the delay of each fractional flow is the portion of its flow rate in the aggregate unicast flow rate. For the example shown in Fig. 2, the delays of the three paths are 3, 3 and 2 respectively, and thus the average end-to-end latency is $0.2 \times 3 + 0.3 \times 3 + 0.5 \times 2$. We further notice that

$$\begin{aligned}
& 0.2 \times (1+1+1) + 0.3 \times (1+1+1) + 0.5 \times (1+1) \\
= \; & 1 \times (0.2+0.5) + 1 \times 0.3 + 1 \times 0.2 + 1 \times 0.5 + 1 \times 0.3 \\
& +1 \times (0.2+0.3) \\
= \; & 1 \times 0.7 + 1 \times 0.3 + 1 \times 0.2 + 1 \times 0.5 + 1 \times 0.3 + 1 \times 0.5 \\
= \; & \sum_{(i,j)\in A} c_{ij} f_{ij}/r.
\end{aligned}$$

For the general case, we prove $\sum_{(i,j)\in A} c_{ij} f_{ij}/r$ also represents the average end-to-end delay of a unicast flow, as in the following proposition.

Proposition. *Let r be the end-to-end streaming rate of a unicast session, c_{ij} be the delay and f_{ij} be the streaming rate on the link (i, j), $\forall(i, j) \in A$. $\sum_{(i,j)\in A} c_{ij} f_{ij}/r$ represents the average end-to-end delay of this unicast flow.*

Proof: Let P be the set of paths from the streaming source to the receiver in the session. Let $f^{(p)}$ be the rate of the fractional flow going along path $p \in P$. The average end-to-end latency at the receiver is

$$\begin{aligned}
& \sum_{p\in P} \frac{f^{(p)}}{r} \Big(\sum_{(i,j):(i,j) \text{ on } p} c_{ij} \Big) \\
= \; & \frac{1}{r} \sum_{(i,j)\in A} c_{ij} \Big(\sum_{p:(i,j) \text{ on } p} f^{(p)} \Big) \\
= \; & \frac{1}{r} \sum_{(i,j)\in A} c_{ij} f_{ij}.
\end{aligned}$$

□

Based on this formulation of the average end-to-end latency, we formulate the peer selection and streaming rate allocation problem for the unicast streaming into the following linear program. Here

u_{ij} is the capacity of the link (i,j). Since r is a constant end-to-end streaming rate, we omit it and use $\sum_{(i,j)\in A} c_{ij} f_{ij}$ to reflect the average end-to-end latency.

$$\min \sum_{(i,j)\in A} c_{ij} f_{ij} \tag{1}$$

subject to

$$\begin{array}{ll} \sum_{j:(i,j)\in A} f_{ij} - \sum_{j:(j,i)\in A} f_{ji} = b_i, & \forall i \in N, \\ 0 \le f_{ij} \le u_{ij}, & \forall (i,j) \in A, \end{array}$$

where

$$b_i = \begin{cases} r & \text{if } i = S, \\ -r & \text{if } i = t, \\ 0 & \text{otherwise.} \end{cases}$$

We can see this is a standard minimum cost flow problem. Practically, by minimizing the average end-to-end delay, we allocate the streaming rates in such a way that high end-to-end latency links, such as satellite and transcontinental links, are avoided as much as possible. When c_{ij}'s ($\forall (i,j) \in A$) are of similar magnitude, we are minimizing the average hop count of the receiver from the streaming source in the network.

We call the optimal unicast flow decided by this linear program a *minimum-delay flow*. This minimum-delay flow is useful for minimum delay multicast streaming in the peer-to-peer network, in that the multicast streaming flow with minimum delays from the source to all receivers can be viewed as consisting of multiple minimum-delay flows. Here we make use of the concept of *conceptual flow* introduced in [12]. A multicast flow is conceptually composed of multiple unicast network flows from the sender to all receivers. These conceptual flows co-exist in the network without contending for link capacities, and the multicast flow rate on a link is the maximum of the rates of all the conceptual flows going along this link. For the peer-to-peer streaming example shown in Fig. 1, the multicast streaming flow from S to t_1, t_2, t_3 and t_4 can be understood as consisting of four conceptual flows from S to each of the receivers. When each conceptual flow is a minimum-delay flow, the end-to-end delays of the multicast session are minimized. Based on this notion, we formulate the linear optimization model for the optimal peer selection and rate allocation problem for our peer-to-peer streaming model.

2.2 Single session case

We first consider the case of single session peer-to-peer streaming. A linear program is formulated to obtain the optimal peer selection strategy, with the objective of minimizing the overall end-to-end streaming delays from the source to all receivers.

Let r be the end-to-end streaming rate of the session. In order to cope with packet loss and the inherent unreliability of peers in a peer-to-peer network, we stream the media at an end-to-end rate that is slightly higher than r. With a tolerance factor α ($\alpha \ge 1$ and the actual streaming rate is set to be αr), a receiver can still receive at a rate at least r for a continuous playback, even with packet loss and peer failures. The value of α can be dynamically adjusted based on the dynamics of the peer-to-peer network in practice, which we will discuss in Sec. 4.

In our linear program, we consider the upload and download capacity constraints at each peer, rather than link capacity constraints. This comes from the observations that overlay links in a peer-to-peer network are usually correlated for sharing certain underlying physical links, and thus individual overlay link capacities are subject to changes due to the traffic dynamics. Furthermore, bandwidth bottlenecks usually occur on the "last-mile" links, and by limiting the total upload and download capacities at each peer, we take this into consideration.

In the following linear program, f^t denotes the conceptual flow from S to a receiver t, $\forall t \in T$. f^t_{ij} denotes the rate of f^t flowing through link (i,j). x_{ij} is the actual multicast streaming rate on link (i,j) and c_{ij} is the delay on link (i,j), $\forall (i,j) \in A$. For node i, O_i is its upload capacity and I_i is its download capacity. We assume all these variables are non-negative.

$$\min \sum_{t\in T} \sum_{(i,j)\in A} c_{ij} f^t_{ij} \tag{2}$$

subject to

$$\sum_{j:(i,j)\in A} f^t_{ij} - \sum_{j:(j,i)\in A} f^t_{ji} = b^t_i, \quad \forall i \in N, \forall t \in T \tag{3}$$

$$f^t_{ij} \ge 0, \quad \forall (i,j) \in A, \forall t \in T \tag{4}$$

$$f^t_{ij} \le x_{ij}, \quad \forall (i,j) \in A, \forall t \in T \tag{5}$$

$$\sum_{j:(i,j)\in A} x_{ij} \le O_i, \quad \forall i \in N \tag{6}$$

$$\sum_{j:(j,i)\in A} x_{ji} \le I_i, \quad \forall i \in N \tag{7}$$

where

$$b^t_i = \begin{cases} \alpha r & \text{if } i = S, \\ -\alpha r & \text{if } i = t, \\ 0 & \text{otherwise.} \end{cases}$$

We can see that in this linear program, each conceptual flow f^t is a valid network flow, subject to constraints (3)(4)(5) similar to those in the minimum-delay problem in Eq. (1). The difference lies in that on link (i,j), f^t_{ij} is bounded by the multicast streaming rate x_{ij} in (5), which is further constrained by the upload and download capacities at the incident nodes in (6) and (7).

An optimal solution to this linear program provides an optimal rate ${f^t_{ij}}^*$ for the conceptual flow f^t on the link (i,j), $\forall (i,j) \in A$. Let z be the optimal multicast streaming flow in the network. We compute the optimal streaming rate on link (i,j), $\forall (i,j) \in A$, as follows:

$$z_{ij} = \max_{t\in T} f^t_{ij}. \tag{8}$$

Thus we obtain an optimal streaming rate allocation scheme for the minimum-delay peer-to-peer streaming. This also provides the optimal peer selection strategy. For each receiver $t \in T$, its set of selected upstream peers includes the ones with non-zero optimal streaming rate from them to t.

2.3 Multiple session case

In the general case, there may co-exist multiple streaming sessions in the same network. Based on a generalization of our network model, we extend the linear program in Eq. (2) to its multiple session form, with the similar objective of minimizing the overall end-to-end streaming delays of all the sessions. Let M be the set of sessions concurrently existing in the network. Each session m, $\forall m \in M$, has a streaming source S_m and a set of receivers T_m. Let f^{m_t} be the conceptual flow from S_m to the receiver t in session m . x^m_{ij} is the multicast streaming rate of session m on the link (i,j), $\forall (i,j) \in A$. r_m is the end-to-end streaming rate for session m and α_m is its tolerance factor. We obtain the following LP:

$$\min \sum_{m\in M}\sum_{t\in T_m}\sum_{(i,j)\in A} c_{ij} f_{ij}^{m_t} \tag{9}$$

subject to

$$\sum_{j:(i,j)\in A} f_{ij}^{m_t} - \sum_{j:(j,i)\in A} f_{ji}^{m_t} = b_i^{m_t}, \forall i \in N, \forall t \in T_m, \forall m \in M,$$

$$\begin{aligned} f_{ij}^{m_t} &\geq 0, & \forall (i,j) \in A, \forall t \in T_m, \forall m \in M,\\ f_{ij}^{m_t} &\leq x_{ij}^{m}, & \forall (i,j) \in A, \forall t \in T_m, \forall m \in M, \end{aligned} \tag{10}$$

$$\textstyle\sum_{j:(i,j)\in A}\sum_{m\in M} x_{ij}^{m} \leq O_i, \quad \forall i \in N, \tag{11}$$

$$\textstyle\sum_{j:(j,i)\in A}\sum_{m\in M} x_{ji}^{m} \leq I_i, \quad \forall i \in N, \tag{12}$$

where

$$b_i^{m_t} = \begin{cases} \alpha_m r_m & \text{if } i = S_m,\\ -\alpha_m r_m & \text{if } i = t,\\ 0 & \text{otherwise.} \end{cases}$$

The LP in Eq. (9) is the result of extending the LP in Eq. (2) to its multicommodity variant. Similar to the single session case, the conceptual flows in each session don't contend for the capacities, and the multicast streaming rate for the session on each link is the maximum of all the conceptual flow rates in the session, as shown in constraint group (10). However, the multicast streaming flows belonging to different sessions contend for the capacities, and thus the summation of the per-session multicast streaming rates on the incoming and outgoing links of a node should not exceed its download and upload capacities, which are the constraints in (11) and (12) respectively.

Similarly, based on the optimal solution to the LP in Eq. (9), we can obtain the optimal peer selection and streaming rate allocation strategy for the multiple sessions. Let z^m be the optimal multicast streaming flow of session m. We have $z_{ij}^m = \max_{t\in T_m} f_{ij}^{m_t}$ on the link (i,j), $\forall (i,j) \in A$.

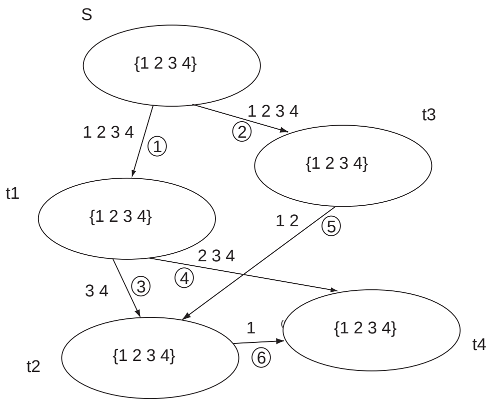
Because of the similarities between the linear programs in Eq. (2) and Eq. (9), they can be solved in a similar fashion. Due to space limitation, in the rest of the paper, we focus on the LP for the single session case.

2.4 Encoding and recoding with rateless codes

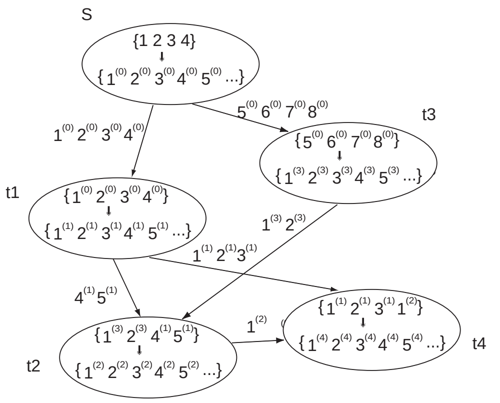
As the linear optimization model optimally decides the streaming rates on each individual overlay link, the media contents to be delivered along each link still need to be carefully assigned to eliminate the delivery redundancy. To address this problem, we employ our coding scheme based on rateless codes, as presented in [21].

We briefly introduce rateless codes and review the main idea of our coding scheme. *Rateless fountain codes*, are a recently developed category of erasure codes, including LT codes [13], Raptor codes [18] and online codes [14]. The "rate" of a traditional erasure code is usually defined as the number of original symbols divided by the number of different encoded symbols that can be generated from them. As for rateless codes, the number of encoded symbols that can be generated from the original symbols is potentially unlimited, which explains the name.

In our coding scheme with rateless codes, we treat the multimedia bit stream to be delivered from the source as a stream of symbols and partition it into consecutive segments. Each segment is further divided into a number of blocks, which are the input units to rateless-code encoders. At the streaming source, the blocks of each media segment are coded with rateless codes before they are transmitted over the peer-to-peer links. At each receiver, after receiving all the encoded blocks belong to one segment, we decode and obtain the original segment. When this segment is further requested by some other peers, we encode its decoded blocks again and deliver the freshly encoded blocks.



(a) Peer-to-peer streaming without rateless-code coding



(b) Peer-to-peer streaming with rateless-code encoding and recoding

Figure 3: A comparison of peer-to-peer streaming with or without the rateless-code coding scheme.

As rateless codes are naturally loss-resilient, our coding scheme provides excellent resilience to peer dynamics. Being *rateless*, there is potentially no limit with respect to the number of uniquely encoded "blocks" generated for each segment, and a sufficient number of encoded blocks from any set of upstream peers may be used to recover the original segment. Therefore, by recoding the received blocks at the receivers with rateless codes before their further transmission, our scheme guarantees the uniqueness and usefulness of all the delivered contents, and thus completely eliminates the needs for set reconciliation and content assignment. We have also shown that our coding scheme will not introduce much delay and computation overhead into the streaming session, due to the high efficiency of rateless-code encoding and decoding [21].

We illustrate our coding scheme with a simple example given in Fig. 3. In this example, S transmits a unit streaming flow to receivers t_1, t_2, t_3 and t_4. Assume the optimal streaming rates computed by the optimization model are 1, 1, 0.5, 0.75, 0.5 and 0.25 on link 1 to link 6 respectively. Each segment of the media stream is composed of 4 blocks. Based on the optimal streaming rates, t_1 and t_3 directly retrieve 4 blocks from S; t_2 parallelly retrieves 2 blocks from t_1 and 2 blocks from t_3, while t_4 retrieves 3

blocks from t_1 and 1 block from t_2. In the case that no rateless-code coding scheme is applied as in Fig. 3(a), both t_2 and t_4 need to carefully reconcile the block difference between their upstream peers and then decide which block to retrieve from which upstream peer. In Fig. 3(b) with our rateless-code coding scheme, S generates a potentially unlimited number of blocks $1^{(0)}$, $2^{(0)}$, ... from the 4 original blocks. After receiving $1^{(0)}$, $2^{(0)}$, $3^{(0)}$ and $4^{(0)}$ from S, t_1 decodes them to derive the original blocks 1, 2, 3 and 4. Then it encodes the original blocks again into $1^{(1)}$, $2^{(1)}$, $3^{(1)}$, $4^{(1)}$, ..., upon downloading requests from t_2 and t_4. The same recoding process is performed at the other receivers. As all the freshly encoded blocks are unique, t_2 and t_4 can safely retrieve any new blocks from their upstream peers without reconciliation.

We combine the optimal peer selection strategy decided by the linear optimization model with our coding scheme based on rateless codes. As they address the two fundamental peer-to-peer streaming problems of optimal peer selection and content assignment respectively, together we are able to achieve a complete minimum-delay peer-to-peer streaming scheme.

In the following section, we proceed to design a distributed algorithm to solve the linear program given in Eq. (2). Combined with our rateless-code coding scheme which is carried out in a distributed manner, this completes our design for the fully-decentralized minimum-delay peer-to-peer streaming.

3. DISTRIBUTED SOLUTION

We aim at designing an efficient distributed algorithm to solve the linear program in Eq. (2). General LP algorithms, such as the simplex, ellipsoid and interior point methods, are inherently centralized and costly, which are not appropriate for our purpose. Our solution is based on the technique of Lagrangian relaxation and subgradient algorithm, and can be naturally implemented in a distributed manner.

3.1 Lagrangian dualization

We start our solution by relaxing the constraint group (5) in Eq. (2) to obtain its Lagrangian dual. We choose to relax this set of constraints, since with such relaxation, the resulting Lagrangian subproblem can be decomposed into classical LP problems, for each of which efficient algorithms exist. We associate Lagrangian multipliers μ_{ij}^t with the constraints in (5) and modify the objective function in (2) as follows:

$$\begin{aligned} & \sum_{t\in T}\sum_{(i,j)\in A} c_{ij} f_{ij}^t + \sum_{t\in T}\sum_{(i,j)\in A} \mu_{ij}^t (f_{ij}^t - x_{ij}) \\ = \ & \sum_{t\in T}\sum_{(i,j)\in A} (c_{ij}+\mu_{ij}^t) f_{ij}^t - \sum_{t\in T}\sum_{(i,j)\in A} \mu_{ij}^t x_{ij}. \end{aligned}$$

We thus derive the Lagrangian dual of the LP in Eq. (2):

$$\max_{\mu\geq 0} L(\mu) \tag{13}$$

where

$$L(\mu) = \min_P \sum_{t\in T}\sum_{(i,j)\in A} (c_{ij}+\mu_{ij}^t) f_{ij}^t - \sum_{t\in T}\sum_{(i,j)\in A} \mu_{ij}^t x_{ij} \tag{14}$$

and the polytope P is defined by the following constraints:

$$\begin{array}{ll} \sum_{j:(i,j)\in A} f_{ij}^t - \sum_{j:(j,i)\in A} f_{ji}^t = b_i^t, & \forall i\in N, \forall t\in T, \\ f_{ij}^t \geq 0, & \forall (i,j)\in A, \forall t\in T, \\ \sum_{j:(i,j)\in A} x_{ij} \leq O_i, & \forall i\in N, \\ \sum_{j:(j,i)\in A} x_{ji} \leq I_i, & \forall i\in N. \end{array}$$

Here, the Lagrangian multiplier μ_{ij}^t can be understood as the link price on the link (i,j) for the conceptual flow from the source S to the receiver t. Such interpretation should be clear as we come to the adjustment of μ_{ij}^t in the subgradient algorithm.

We further observe that the Lagrangian subproblem in Eq. (14) can be decomposed into a maximization problem

$$\max \sum_{t\in T}\sum_{(i,j)\in A} \mu_{ij}^t x_{ij} \tag{15}$$

subject to

$$\begin{array}{ll} \sum_{j:(i,j)\in A} x_{ij} \leq O_i, & \forall i\in N, \\ \sum_{j:(j,i)\in A} x_{ji} \leq I_i, & \forall i\in N, \end{array}$$

and multiple minimization problems ($\forall t\in T$)

$$\min \sum_{(i,j)\in A} (c_{ij}+\mu_{ij}^t) f_{ij}^t \tag{16}$$

subject to

$$\begin{array}{ll} \sum_{j:(i,j)\in A} f_{ij}^t - \sum_{j:(j,i)\in A} f_{ji}^t = b_i^t, & \forall i\in N, \\ f_{ij}^t \geq 0, & \forall (i,j)\in A. \end{array}$$

We notice that the maximization problem in Eq. (15) is an inequality constrained transportation problem. For this class of transportation problems, there exist distributed algorithms, such as the auction algorithm [4], to solve them in polynomial time. For the minimization problem in Eq. (16), we make the substitution ${f_{ij}^t}' = f_{ij}^t/\alpha r$ and obtain the following shortest path problem:

$$\min \sum_{(i,j)\in A} \alpha r (c_{ij}+\mu_{ij}^t) {f_{ij}^t}' \tag{17}$$

subject to

$$\begin{array}{ll} \sum_{j:(i,j)\in A} {f_{ij}^t}' - \sum_{j:(j,i)\in A} {f_{ji}^t}' = b_i^{t\prime}, & \forall i\in N, \\ {f_{ij}^t}' \geq 0, & \forall (i,j)\in A, \end{array}$$

where

$$b_i^{t\prime} = \begin{cases} 1 & \text{if } i=S, \\ -1 & \text{if } i=t, \\ 0 & \text{otherwise.} \end{cases}$$

The shortest path problem is a classical combinatorial optimization problem, and efficient distributed algorithms exist to solve the problem in a directed network. The classical algorithm is the Bellman-Ford algorithm, and there are other algorithms, such as label-correcting algorithms [2] and relaxation algorithms [6], that are essentially the same as the Bellman-Ford algorithm. In our algorithm, we employ the distributed Bellman-Ford algorithm [5, 6] to obtain the shortest path from S to the receiver t with weight $\alpha r(c_{ij}+\mu_{ij}^t)$ on the link (i,j). Then, by letting ${f_{ij}^t}' = 1$ when the link (i,j) is on the shortest path and ${f_{ij}^t}' = 0$ otherwise, we obtain an optimal solution to the LP in Eq. (17). Therefore, by taking $f_{ij}^t = \alpha r {f_{ij}^t}'$, which can be understood as the delivery of a concept flow of rate αr along the shortest path, we further obtain an optimal solution to the LP in Eq. (16).

3.2 Subgradient algorithm

We now describe the subgradient algorithm, which we apply to solve the Lagrangian dual problem in Eq. (13). We start with a set of initial non-negative Lagrangian multipliers $\mu_{ij}^t[0]$. During each iteration k, given current Lagrangian multiplier values $\mu_{ij}^t[k]$, we solve the transportation problem in (15) and the shortest path problems in (16) to obtain new primal values $x_{ij}[k]$ and $f_{ij}^t[k]$. We then update the Lagrangian multipliers by

$$\mu_{ij}^t[k+1] = \max(0, \mu_{ij}^t[k] + \theta[k](f_{ij}^t[k] - x_{ij}[k])), \quad \forall (i,j) \in A, \forall t \in T, \qquad (18)$$

where θ is a prescribed sequence of step sizes that decides the convergence and the convergence speed of the subgradient algorithm. When θ satisfies the following conditions, the algorithm is guaranteed to converge:

$$\theta[k] > 0, \lim_{k\to\infty}\theta[k] = 0, \text{ and } \sum_{k=1}^{\infty}\theta[k] = \infty.$$

Eq. (18) shows the adjustment of link prices for each conceptual flow. If the rate of the conceptual flow exceeds the multicast flow rate on the link, constraint (5) is violated, so the link price is raised. Otherwise, the link price is reduced.

Since the primal values in the optimal solution of the Lagrangian dual in Eq. (13) are not necessarily an optimal solution to the primal LP in Eq. (2), and even not a feasible solution to it, we use the algorithm introduced by Sherali *et al.* [17] to recover the optimal primal values ${f_{ij}^t}^*$. At the k^{th} iteration of the subgradient algorithm, we also compose a primal iterate $\widehat{f_{ij}^t}[k]$ via

$$\widehat{f_{ij}^t}[k] = \sum_{h=1}^{k} \lambda_h^k f_{ij}^t[h], \qquad (19)$$

where $\sum_{h=1}^{k} \lambda_h^k = 1$ and $\lambda_h^k \geq 0$, for $h = 1, \ldots, k$. Thus, $\widehat{f_{ij}^t}[k]$ is a convex combination of the primal values obtained in the earlier iterations.

In our algorithm, we choose the step length sequence $\theta[k] = a/(b+ck), \forall k, a > 0, b \geq 0, c > 0$, and convex combination weights $\lambda_h^k = 1/k, \forall h = 1, \ldots, k, \forall k$. These guarantee the convergence of our subgradient algorithm; they also guarantee that any accumulation point $\widehat{f}^*$ of the sequence $\{\widehat{f}[k]\}$ generated via (19) is an optimal solution to the primal problem in Eq. (2) [17].

We can thus calculate $\widehat{f_{ij}^t}[k]$ by

$$\begin{aligned} \widehat{f_{ij}^t}[k] &= \sum_{h=1}^{k} \frac{1}{k} f_{ij}^t[h] \\ &= \frac{k-1}{k} \sum_{h=1}^{k-1} \frac{1}{k-1} f_{ij}^t[h] + \frac{1}{k} f_{ij}^t[k] \\ &= \frac{k-1}{k} \widehat{f_{ij}^t}[k-1] + \frac{1}{k} f_{ij}^t[k]. \end{aligned}$$

and obtain $\widehat{f_{ij}^t}[k]$ from the current primal value $f_{ij}^t[k]$ and the primal iterate $\widehat{f_{ij}^t}[k-1]$ of the last iteration. In this way, we do not need to keep all the primal values calculated since the very first iteration, which are needed by Eq. (19).

3.3 Distributed algorithm

Based on the subgradient algorithm and primal solution recovery algorithm, we design our distributed algorithm to solve the linear program in Eq. (2) and achieve the optimal streaming rates on the links by Eq. (8). The distributed algorithm to be executed by link (i,j) is summarized in Table 1. In practice, we have each link (i,j) in the peer-to-peer network delegated by the receiver j, and thus the computation tasks on all the incoming links of one peer is carried out by the peer.

Table 1: The distributed algorithm on link (i,j)

```
1. Choose initial Lagrangian multiplier values $\mu_{ij}^t[0]$, $\forall (i,j) \in A$, $\forall t \in T$.
2. Repeat the following iteration until the sequence $\{\mu[k]\}$ converges to $\mu^*$ and the sequence $\{\widehat{f}[k]\}$ converges to $\widehat{f}^*$: $\forall (i,j) \in A, \forall t \in T$
    1) Compute $x_{ij}[k]$ by the distributed auction algorithm;
    2) Compute $f_{ij}^t[k]$ by the distributed Bellman-Ford algorithm;
    3) Compute $\widehat{f_{ij}^t}[k] = \frac{k-1}{k}\widehat{f_{ij}^t}[k-1] + \frac{1}{k} f_{ij}^t[k]$;
    4) Update Lagrangian multiplier $\mu_{ij}^t[k+1] = \max(0, \mu_{ij}^t[k] + \theta[k](f_{ij}^t[k] - x_{ij}[k]))$, where $\theta[k] = a/(b+ck)$.
3. Compute the optimal multicast streaming rate $z_{ij} = \max_{t\in T} {\widehat{f_{ij}^t}}^*$.
```

By delivering the media contents at the computed optimal streaming rates on the links, we achieve minimum-delay peer-to-peer streaming. We further emphasize that this is actually achieved by applying our coding scheme with rateless codes, but without the complex set reconciliation and content assignment in the streaming session.

4. HANDLING PEER DYNAMICS

In peer-to-peer streaming, peers may arbitrarily join a streaming session at any time, and may depart or fail unexpectedly. By applying rateless codes and introducing the tolerance factor α into our linear optimization model, our minimum-delay peer-to-peer streaming scheme provides excellent resilience to peer dynamics.

4.1 Peer joins

In our system, a new peer is bootstrapped and admitted into a streaming session only if its download capacity can support the streaming rate r of the session. The new peer then starts streaming immediately with the available upload capacities acquired from its known upstream peers. At the same time, it sends a request to the streaming source, asking for re-computation of the new globally optimal streaming rates on the links. The source broadcasts such a request, such that all the peers in the session activate a new round of execution of the distributed algorithm in Table 1, while continuing with their own streaming at the original optimal rates. If the distributed algorithm converges, all the peers adjust their download rates from their upstream peers to the new optimal values. If the algorithm fails to converge, it is evident that the upload capacities provided by the peers in the session now are not able to support all the receivers at the session rate of r. In this case, the original streaming rate allocation is not to be adjusted.

4.2 Peer departures and failures

The departures and failures of peers may lead to interrupted playback at the remaining receivers. To compensate the rate loss, we trade off some of the optimality, and always deliver slightly more than the end-to-end rate of r, controlled by the tolerance factor α. Thus, when a peer detects the departure of its upstream peer(s), it first estimates whether its remaining aggregate streaming rate is still no less than r. If so, it is not affected and keeps its current streaming rates; otherwise, it attempts to acquire more upload capacities from its remaining upstream peers. Only when the peer still fails to receive at the adequate aggregate streaming rate, it sends a re-calculation request to the source for the new globally optimal rates. Similarly, the remaining peers adjust their streaming rates to the new values if the distributed algorithm converges. Otherwise, they do not do so in order to keep the streaming optimality of unaffected peers. In this way, we only invoke the distributed algorithm and adjust the rates when necessary, so as to minimize the computation and communication overhead.

The value of the tolerance factor α ($\alpha \geq 1$) in our optimization model is adjustable based on the dynamics of the peer-to-peer network. We start with an initial value of α based on an estimation of the peer failure probability in the network. The peer failure probability is continuously re-estimated by the streaming source, according to the frequency of re-calculation requests due to peer failures. When the source frequently receives requests to invoke the distributed algorithm and thus the estimated peer failure probability increases, α is also increased, as long as the network can support the end-to-end streaming rate of αr. When the estimated peer failure probability is low and it occurs that the network fails to accommodate more freshly joined peers at the rate of αr, we reduce α so that the network is able to support more receivers at an end-to-end rate of at least r. The relationship between α and the percentage of peer failures is to be shown in the simulations.

5. PERFORMANCE EVALUATION

We have carried out extensive simulations to investigate the performance of our optimal peer selection algorithm over realistic network topologies. To this end, we generate random networks with power-law degree distributions with the BRITE topology generator [15]. We simulate a live streaming session of a high-quality 300 Kbps multimedia bitstream from a streaming source with 10 Mbps of upload capacity. There are two classes of receivers: ADSL/cable modem peers and Ethernet peers. In our general setting, ADSL/cable modem peers take 70% of the total population with $1.5 - 4.5$ Mbps of download capacity and $0.6 - 0.9$ Mbps of upload capacity, and Ethernet peers take the other 30% with both upload and download capacities of $8 - 12$ Mbps.

5.1 Performance of the distributed algorithm

We first investigate the convergence speed of our distributed algorithm to obtain the optimal streaming topology. The result is shown in Fig. 4. We compare the convergence speed in networks of different *network sizes* (numbers of peers in the network) and different *edge densities* (the ratio of the number of edges to the number of peers in the network). We can see that it takes around 70 iterations to converge to optimality in a network of 50 peers, and this number increases slowly to about 170 for a network of 500 peers. However, the convergence speed remains approximately the same in a fixed-sized network with different edge densities. Therefore, the slow increase of iteration numbers with network sizes does not affect the scalability of our algorithm.

We further compare the convergence speeds of our algorithm to the first primal feasible solution, to the feasible solution which achieves 90% optimality as to the value of the objective function, and to the optimal solution. From Fig. 5, we observe that the convergence speed to the first primal feasible solution is usually much faster than the convergence to optimality. It can also be seen that the number of iterations needed to converge to feasibility drops quickly with the increase of the percentage of Ethernet peers in the network, which bring more abundant upload capacities. Furthermore, in order to converge to the feasible solution which achieves 90% optimality, the algorithm takes only 75% of the number of iterations required for convergence to the optimal solution. Therefore, in practice, we can obtain a feasible solution to a certain degree of the optimality in a much shorter time, when it is not always necessary to achieve the optimal solution in a realistic streaming system.

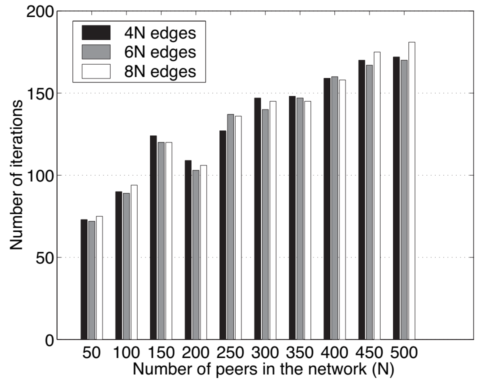


Figure 4: Convergence speed in random networks.

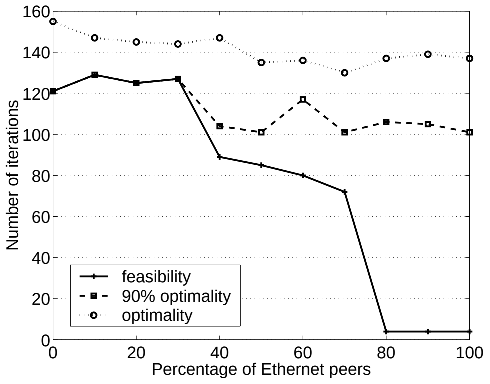


Figure 5: Convergence speed to feasibility, 90%-optimality and optimality in random networks of 300 peers and 2400 edges.

We next compare our optimal peer selection algorithm with a commonly used peer selection heuristic [11, 22]. In the heuristic, a receiver distributes the streaming rates among its upstream peers in proportion to their upload capacities. We compare the end-to-end latencies at receivers in the resulting streaming topologies. The end-to-end latency at each receiver is calculated as the weighted average of the delays of flows from all its upstream peers, and the weight for each flow is the portion of the assigned streaming rate from the upstream peer in the aggregate streaming rate.

The results illustrated in Fig. 6 meet our expectations. In networks of different network sizes and edge densities, our end-to-end latency minimization algorithm is able to achieve much lower latencies than the heuristic, which does not take link delay into consideration. We further notice that the denser the network is, the higher the average end-to-end latency is by the heuristic. In contrast, our optimal algorithm achieves lower latencies in denser networks. When the edge density is $4N$ in a network of N peers, the average end-to-end latency of the heuristic is about 1.5 times higher than that of our optimal algorithm, while this ratio becomes 2 in a network with $8N$ edges. For such an achievement of lower latencies in denser networks with our algorithm, we believe the reason is that there are more choices of upstream peers in a denser network and our algorithm can always find the best set of upstream peers on low delay paths. Thus, in realistic peer-to-peer streaming networks with high edge densities, the advantage of our algorithm is more evident over the commonly used heuristic.

The streaming topologies shown in Fig. 7(a) and Fig. 7(b) further illustrate the superiority of our optimal algorithm. In these graphs,

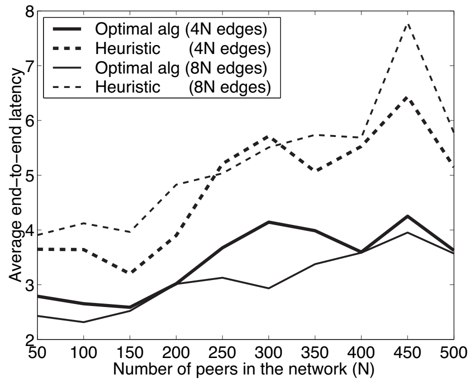
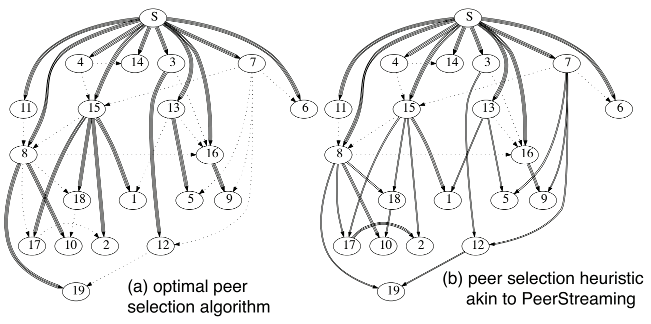


Figure 6: Average end-to-end latency: a comparison between optimal peer selection algorithm and a peer selection heuristic.



(a) optimal peer selection algorithm
(b) peer selection heuristic akin to PeerStreaming

Figure 7: Peer-to-peer streaming topologies of 20 peers: a comparison.

distances between pairs of peers represent latencies, and the widths of edges show the streaming rates along them. The dotted lines represent links that are not used in the resulting streaming topologies. It can be seen that by our optimal peer selection, receivers are streaming from the best upstream peers with minimal end-to-end latencies, while with the peer selection heuristic, peers simply distribute their download rates among the upstream peers, which may lead to large end-to-end latencies.

5.2 Resilience to peer failures

In this section, we examine the resilience of our complete peer-to-peer streaming scheme in case of peer failures. We focus on evaluating the effects of α, the failure tolerance factor in our optimization model, on the tolerance to failures in the resulting streaming topologies. For this purpose, we randomly choose different percentages of failed peers in the streaming session. At different values of α, we calculate the remaining aggregate streaming rates at the remaining peers. We seek to answer the question: at what percentage of peer failures can we still maintain an end-to-end streaming rate of 300 Kbps at all receivers?

From Fig. 8, besides the fact that higher failure percentages can be tolerated by larger values of α in each network, we also observe that this failure tolerance improves quickly with the increase of edge densities in the network, under each fixed value of α. A setting of $\alpha = 1.5$ can tolerate peer failure percentage of up to 60% in a dense network. The impact of edge densities is further illustrated in Fig. 9. In this simulation, when α is set to a relatively small value of 1.2, which means we stream at the end-to-end rate of 360 Kbps, the 300 Kbps required aggregate rate can still be guaranteed at the receivers in a dense network in case of 30% peer failures. We believe that dense networks are more akin to realistic peer-to-peer streaming networks, in which each peer knows quite a number of upstream peers. Such satisfactory failure tolerance in dense networks achieved by small values of α suggests that our scheme may deliver good performance in practice, with respect to tolerating peer departures and failures.

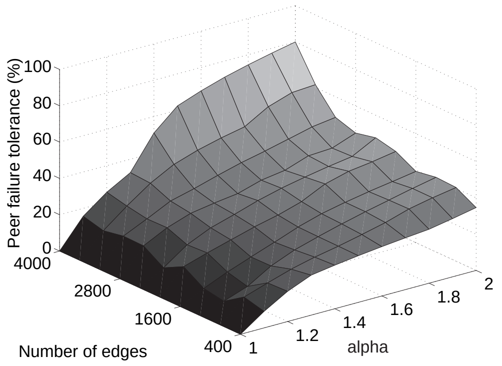


Figure 8: Tolerance of peer failures in random networks with 200 peers.

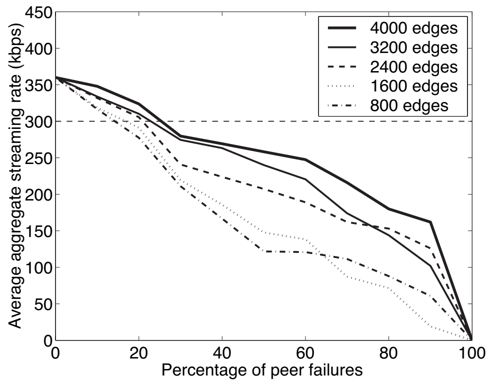


Figure 9: Average aggregate streaming rates in case of peer failures in random networks, network size = 200, α = 1.2.

6. RELATED WORK

Earlier work on peer-to-peer multimedia streaming has been based on a single multicast tree [9, 19], rooted at the streaming source, and constructed with a minimized height and a bounded node degree. The challenge, however, surfaces when interior peers in the tree do not have sufficient available capacities to upload to multiple children nodes, and when they depart or fail, which interrupts the streaming session and requires expensive repair processes.

Streaming based on multiple multicast trees has been proposed to address this problem, as in CoopNet [16] and SplitStream [8]. The media can be split into multiple sub-streams, each delivered along a different multicast tree. As a result, these systems accommodate peers with heterogeneous bandwidths by having each peer join different numbers of trees. It also is more robust to peer departures and failures, as an affected receiving peer may still be able to continuously display the media at a degraded quality, while waiting for the tree to be repaired. These advantages come with a cost, however, as all the trees need to be maintained in highly dynamic peer-to-peer networks. Combined with rateless codes, our optimal peer selection algorithm constructs mesh topologies to provide resilience and flexibility, but without the costs of explicit tree maintenance.

Similar to our peer-to-peer streaming model, CoolStreaming [22] and PeerStreaming [11] fall into the category of streaming over mesh topologies. In CoolStreaming, each peer periodically exchanges media availability with its neighbors, and the media segments to be retrieved from each neighbor are dependent upon the number of potential suppliers for each segment and the available upload capacities of neighbors. In PeerStreaming, the media downloading load is distributed among the set of supplying peers in proportion to their upload capacities. Compared to our peer selection algorithm, these heuristics fall short of achieving global optimality, and thus may starve the peers with high download demands. Our algorithm also considers the heterogeneity of link delays, which has not been taken into consideration in their work.

With respect to peer selection, like CoolStreaming and PeerStreaming, most existing work employs various heuristics without formulating the problem theoretically, with only one exception: Adler *et al.* [1] propose linear programming models that aim at minimizing general cost functions in peer-to-peer streaming. Our algorithm is tailored to the specific requirements of media streaming, by minimizing streaming latencies. In [1], i supplying peers are allowed to fail, by having constraints guaranteeing the aggregate rate from any subset composed of $n - i$ supplying peers out of the total n is larger than the streaming rate. Our algorithm handles peer failures by simply incorporating a tolerance factor, which can be flexibly adjusted according to the network dynamics. The problem of content reconciliation and assignment is not addressed in [1], which is the motivating factor towards the combination of rateless codes with our peer selection algorithm.

As for the content reconciliation problem that naturally arises in all mesh-based proposals, Byers *et al.* [7] provide algorithms for estimation and approximate reconciliation of sets of symbols between pairs of collaborating peers. These algorithms are very resource intensive with respect to both computation and messaging. Besides, although Byers *et al.* advocate the use of Tornado codes to provide reliability and flexibility, the sets of encoded symbols acquired by different peers are still likely to overlap, as Tornado codes are not rateless. PeerStreaming [11] also employs a high rate erasure code, which is a modified Reed-Solomon code on the Galois Field GF(2^{16}), and ensures with high probability that the serving peers hold parts of the media without conflicts. PROMISE [10] uses Tornado codes to tolerate packet losses and peer dynamics, and performs rate assignment of the encoded streams to a selected set of supplying peers. By applying these erasure codes with fixed rates, the need for content reconciliation is mitigated, but not eliminated. In comparison, by using rateless codes and recoding, our scheme completely excludes any necessity for content reconciliation and assignment among peers.

7. CONCLUSION

The problem of interest in this paper is to design an efficient distributed algorithm for optimal peer selection and streaming rate allocation in peer-to-peer streaming. For this purpose, we formulate the problem as a linear optimization problem, which optimizes bandwidth utilization towards minimized end-to-end latencies. Based on the efficient subgradient solution, we develop a fully decentralized algorithm to efficiently compute the optimal streaming rates over the peer-to-peer links. We believe combining this optimal peer selection algorithm with our rateless-code coding scheme, it provides a complete solution to battle on the fundamental challenges of peer-to-peer streaming: dynamics, reconciliation, and limited bandwidth. In the ongoing work, we are working towards a practical implementation of the minimum-delay peer-to-peer streaming system, in the real-life environment of the Internet.

8. REFERENCES

[1] M. Adler, R. Kumar, K. W. Ross, D. Rubenstein, T. Suel, and D. D. Yao. Optimal Peer Selection for P2P Downloading and Streaming. In *Proc. of IEEE INFOCOM 2005*, March 2005.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proc. of ACM SIGCOMM 2002*, August 2002.

[4] D. P. Bertsekas and D. A. Castanon. The Auction Algorithm for the Transportation Problem. *Annals of Operations Research*, 20:67–96, 1989.

[5] D. P. Bertsekas and R. Gallager. *Data Networks, 2nd Ed.* Prentice Hall, 1992.

[6] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[7] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *Proc. of ACM SIGCOMM 2002*, August 2002.

[8] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP) 2003*, October 2003.

[9] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media over a Peer-to-Peer Network. Technical report, Standford Database Group 2001-20, August 2001.

[10] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-Peer Media Streaming Using CollectCast. In *Proc. of ACM Multimedia 2003*, November 2003.

[11] J. Li. PeerStreaming: A Practical Receiver-Driven Peer-to-Peer Media Streaming System. Technical report, Microsoft Research MSR-TR-2004-101, September 2004.

[12] Z. Li, B. Li, D. Jiang, and L. C. Lau. On Achieving Optimal Throughput with Network Coding. In *Proc. of IEEE INFOCOM 2005*, March 2005.

[13] M. Luby. LT Codes. In *Proc. of the 43rd Symposium on Foundations of Computer Science*, November 2002.

[14] P. Maymounkov and D. Mazieres. Rateless Codes and Big Downloads. In *Proc. of the 2nd Int. Workshop Peer-to-Peer Systems (IPTPS)*, February 2003.

[15] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Boston University Representative Internet Topology Generator. Technical report, http://www.cs.bu.edu/brite, 2000.

[16] V. N. Padmanabhan, H. J. Wang, P. A. Chow, and K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. In *Proc. of NOSSDAV 2002*, May 2002.

[17] H. D. Sherali and G. Choi. Recovery of Primal Solutions when Using Subgradient Optimization Methods to Solve Lagrangian Duals of Linear Programs. *Operations Research Letter*, 19:105–113, 1996.

[18] A. Shokrollahi. Raptor Codes. In *Proc. of the IEEE International Symposium on Information Theory (ISIT) 2004*, June 2004.

[19] D. A. Tran, K. A. Hua, and T. Do. ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. In *Proc. of IEEE INFOCOM 2003*, March 2003.

[20] D. A. Tran, K. A. Hua, and T. T. Do. A Peer-to-Peer

architecture for Media Streaming. *IEEE Journal on Selected Areas in Communications*, 22:121–133, January 2004.

[21] C. Wu and B. Li. rStream: Resilient Peer-to-Peer Streaming with Rateless Codes. In *Proc. of ACM Multimedia 2005, to appear*, November 2005.

[22] X. Zhang, J. Liu, B. Li, and T. P. Yum. Data Driven Overlay Streaming: Design, Implementation, and Experience. In *Proc. of IEEE INFOCOM 2005*, March 2005.