On Meeting P2P Streaming Bandwidth Demand with Limited Supplies

Chuan Wu and Baochun Li

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada M5S 3G4

ABSTRACT

As a basic requirement of live peer-to-peer multimedia streaming sessions, the streaming playback rate needs to be strictly enforced at each of the peers. In real-world peer-to-peer streaming sessions with very large scales, the number of streaming servers for each session may not be easily increased, leading to a limited supply of bandwidth. To scale to a large number of peers, one prefers to regulate the bandwidth usage on each of the overlay links in an optimal fashion, such that limited supplies of bandwidth may be maximally utilized. In this paper, we propose a decentralized bandwidth allocation algorithm that can be practically implemented in peer-to-peer streaming sessions. Given a mesh P2P topology, our algorithm explicitly *reorganizes* the bandwidth of data transmission on each overlay link, such that the streaming bandwidth demand is always guaranteed to be met at any peer in the session, without depending on any *a priori* knowledge of available peer upload or overlay link bandwidth. Our algorithm is especially useful when there exists no or little surplus bandwidth supply from servers or other peers. It adapts well to time-varying availability of bandwidth, and guarantees bandwidth supply for the existing peers during volatile peer dynamics. We demonstrate the effectiveness of our algorithm with in-depth simulation studies.

Keywords: Peer-to-peer streaming, bandwidth allocation, scalability, decentralized algorithm

1. INTRODUCTION

The peer-to-peer communication paradigm has been successfully used in live media streaming applications over the Internet.[1,2] As participating peers contribute their upload bandwidth capacities to serve other peers in the same streaming session, the load on dedicated streaming servers is significantly mitigated. Therefore, as one of the most significant benefits, peer-to-peer streaming enjoys the salient advantage of *scalability*, as compared to traditional streaming using multiple unicast sessions.

In current-generation mesh-based peer-to-peer streaming applications,[1–3] it is critical to achieve and maintain a specific streaming playback rate at each participating peer, in order to guarantee the smooth playback of the media. For example, with the H.264 codec, a Standard-Definition stream demands approximately 800 Kbps, while 480p (848×480 pixels) High-Definition media stream using the H.264 codec requires 1700 Kbps. In this paper, such streaming playback rates are collectively referred to as the *demand* of P2P streaming bandwidth, and we wish to make sure that such demands are satisfied on all participating peers in the streaming session.

On a participating peer in the streaming session, whether or not the streaming bandwidth demand can be achieved depends on three constraints. First, the last-mile download capacity must exceed the streaming rate. We typically assume that this constraint is always satisfied, as otherwise the required streaming rate cannot be achieved with any solution. It is most likely the case in reality, as peers without sufficient last-mile download capacities would soon leave the session, and join another session to download the media encoded at lower bit rates. Second, assuming that the peer in question is served by multiple upstream peers, the last-mile upload capacities of these upstream peers are limited. Finally, the available bandwidth on each overlay link between two peers is limited, subject to link capacity and cross traffic in the Internet core.

In essence, the decisive factor of meeting the P2P streaming bandwidth demand in the session is the *bandwidth supply* from either dedicated streaming servers or uploading peers. Ideally, when the total bandwidth supply is abundant, a peer can easily contact new streaming servers or peers when its demand cannot be met at any time. However, such a simple solution does not work effectively at all when there exist very limited bandwidth supplies to meet the demand.

E-mail: {chuanwu, bli}@eecg.toronto.edu

Such a microeconomic phenomenon of tight supply-demand relationships occurs in real-world scenarios, when a pool of streaming servers scrambles to meet the demand of a P2P session that scales up in a short period of time — a typical flash crowd scenario. As adding streaming servers (*i.e.,* adding bandwidth supplies) is by no means straightforward, one needs to take advantage of existing bandwidth supplies in the most efficient manner in scenarios with tight supply-demand relationships.

In this paper, we seek to propose a new bandwidth allocation algorithm that dynamically adjusts the bandwidth utilization on each overlay link, so that the streaming bandwidth demand is achieved on each participating peer in the session, even with very limited bandwidth supplies. Given a mesh peer-to-peer topology, our new algorithm effectively *"reorganizes"* the bandwidth allocated to each link in the topology, with full awareness of the streaming rate demand of the peer-to-peer session. The new algorithm we propose enjoys the following salient advantages. *First*, although the rates allocated are subject to the capacity constraints at the edge and on the overlay links, our new algorithm does not need knowledge of these capacity constraints. *Second*, our algorithm is fully decentralized, and can thus be realistically implemented. To design such an algorithm, we formulate the bandwidth allocation problem in peer-to-peer streaming as a *feasibility* problem, and propose a simple algorithm to find its solution. We discuss the implementation and prove the convergence of the algorithm in synchronous and asynchronous environments. *Third*, we show that even in cases of persistent peer dynamics and network changes, the algorithm is still guaranteed to achieve the streaming playback rate at any participating peer and at any time during the streaming session. *Finally*, our solution is across-the-board and not specific to any particular mesh-based topology construction mechanism, P2P streaming protocol, or media codec, as the problem of bandwidth allocation to guarantee smooth streaming playback is fundamental and widely applicable with any P2P topology construction algorithm. Bandwidth allocation may also be realistically implemented in any P2P streaming protocol, by using per-link bandwidth shaping mechanisms at peers (usually based on UDP as the transport protocol).

The organization of this paper is as follows. In Sec. 2, we present our system model and motivate the feasibility problem formulation of the bandwidth allocation problem. In Sec. 3, we discuss the distributed algorithm to solve the problem, and its convergence to the feasible solution in an ideal synchronous environment. In Sec. 4, a practical bandwidth allocation protocol is designed, and its convergence in asynchronous and dynamic environments is discussed. Simulation results are presented in Sec. 5. We discuss related work and conclude the paper in Sec. 6 and Sec. 7, respectively.

2. MOTIVATION AND PROBLEM FORMULATION

Consider a *mesh* peer-to-peer topology $\mathcal{G} = (\mathcal{S}, \mathcal{N}, \mathcal{A})$, where $\mathcal{S}$ is the set of streaming servers, $\mathcal{N}$ is the set of participating peers, and $\mathcal{A}$ is the set of directed overlay links. We assume such a mesh topology is constructed and maintained with a certain topology construction protocol, *e.g.,* the random mesh construction by randomly assigning neighbors to the participating peers using central tracking servers, as employed by most current-generation P2P streaming applications.[4,5] The overlay links among peers in the topology are established based on their media content availability during streaming, *i.e.,* a peer streams from one or more *upstream* peers, which can provide it with media blocks it requires, and further serves one or more *downstream* peers with the media streams. Each directed overlay link in set $\mathcal{A}$ is represented as (i, j), indicating upstream peer i serves a flow of required media blocks to the downstream peer j.

In this paper, we focus on the availability of *bandwidth* in the peer-to-peer topology, arguably the most critical resource in peer-to-peer streaming sessions. We study both the "demand" and the "supply" of bandwidth based on the given topology. On the side of *demand* of bandwidth, a streaming playback rate $\mathcal{R}$ is strictly required at each participating peer to guarantee smooth playback, *i.e.,* media content should be downloaded at an aggregated bandwidth of no lower than $\mathcal{R}$ at the peer. On the side of *supply* of bandwidth, however, we need to consider both peer upload bandwidth and overlay link bandwidth constraints, without *a priori* knowledge on either. How do we meticulously allocate the supply of bandwidth so that the streaming playback rate $\mathcal{R}$ — the "demand" in each session — can be satisfied at all times? This problem is henceforth referred to as the *bandwidth allocation* problem, as we seek to design a practical and decentralized algorithm to address such a challenge. While our focus in the paper is not on the specific peer-to-peer topology construction protocol nor media scheduling protocol, we will discuss the interactive play between our bandwidth allocation and topology construction in peer-to-peer streaming in Sec. 4.

Since we do not need to consume *more* bandwidth than the required streaming playback rate at each peer, we believe it is practical to formulate such a bandwidth allocation problem as a *feasibility* problem, and aim to find a *feasible* bandwidth allocation solution that guarantees a streaming rate of no lower than $\mathcal{R}$ at each peer.

One may wonder why a naive bandwidth allocation may fail to satisfy the streaming bandwidth demand at all peers. We show this with an example in Fig. 1, which implies that an explicit bandwidth allocation algorithm is required.

In this example peer-to-peer streaming network, the required streaming rate is $\mathcal{R} = 1$ Mbps. Peers a_1, a_2, and a_3 directly stream at this rate from server s, which has 3 Mbps of upload capacity, and then serve b_1, b_2, and b_3. Assume that bandwidth bottlenecks occur at the upload links of the three a peers, with upload capacities of 0.8, 1.0 and 1.2 (in Mbps), respectively. With a naive even bandwidth allocation method, their upload capacities are evenly shared among their respective downstream peers, and the allocated bandwidths are labeled on the links (numbers outside the brackets). Such a bandwidth allocation outcome is infeasible (*i.e.*, streaming bandwidth demand is not satisfied at all peers), as peer b_1 only obtains an aggregate bandwidth of 0.9 Mbps, while b_3 is allocated more than 1 Mbps.

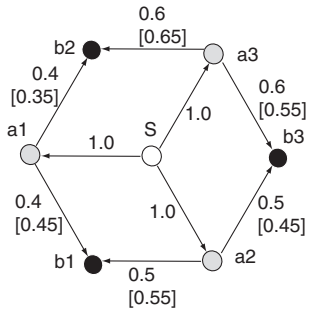

Figure 1. Infeasible bandwidth allocation with a naive protocol: an example.

When the required streaming bandwidth is not successfully achieved, the common practice with existing peer-to-peer protocols is to find new upstream peers, or to start a new connection from the streaming server, which may well fail to locate available bandwidth when there is a tight supply-demand relationship of bandwidth in the network. It is important to note, however, that if bandwidths are explicitly *reallocated* based on the current topology, a feasible solution that satisfies all the peers can be achieved, as shown in brackets in our example. After such a "reorganizing" process, the "supply" of bandwidth is maximally utilized, the need to find new upstream peers is eliminated, and server bandwidth costs are minimized.

In order to design a practical algorithm to achieve such feasible rate allocation solutions in a particular P2P mesh topology, we first formally formulate the problem. Let x_{ij} denote the allocated transmission rate on the overlay link (i, j), $\forall (i, j) \in \mathcal{A}$. In practical peer-to-peer systems, such overlay link rates are restricted by the capacities of last-mile access links of the peers, and affected by the cross traffic sharing the same underlying IP network. Let $\mathcal{C}_{ij}$ denote the currently available bandwidth along overlay link (i, j), subject to the cross traffic. Let $\mathcal{O}_i$ denote the upload capacity at peer i, $\forall i \in \mathcal{S} \cup \mathcal{N}$. The feasibility problem is formally defined by the following set of linear rate and capacity constraints:

LC:

$$\sum_{i:(i,j)\in\mathcal{A}} x_{ij} \geq \mathcal{R}, \qquad \forall j \in \mathcal{N}, \tag{1}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{ij} \leq \mathcal{O}_i, \quad \forall i \in \mathcal{S} \cup \mathcal{N}, \tag{2}$$

$$x_{ij} \leq \mathcal{C}_{ij}, \quad \forall (i, j) \in \mathcal{A}. \tag{3}$$

Let $x = (x_{ij}, (i, j) \in \mathcal{A})$ be the $|\mathcal{A}|$-dimensional vector of allocated link bandwidths. Let X_R be the region defined by the streaming rate constraints in (1), *i.e.*, $X_R = \{x : \sum_{i:(i,j)\in\mathcal{A}} x_{ij} \geq \mathcal{R}, \forall j \in \mathcal{N}\}$. Let X_C be the region defined by the capacity constraints in (2) and (3), *i.e.*, $X_C = \{x : \sum_{j:(i,j)\in\mathcal{A}} x_{ij} \leq \mathcal{O}_i, \forall i \in \mathcal{S} \cup \mathcal{N}, x_{ij} \leq \mathcal{C}_{ij}, \forall (i, j) \in \mathcal{A}\}$. A solution to this feasibility problem represents a feasible bandwidth allocation scheme, expressed as $x \in X_R \cap X_C$.

3. DECENTRALIZED BANDWIDTH ALLOCATION: THE SYNCHRONOUS CASE

We are now ready to propose our iterative algorithm to solve the feasibility problem **LC**. We show that it can be readily implemented in a fully decentralized fashion, and analyze its convergence in the synchronous case.

3.1 Feasible bandwidth allocation: an iterative algorithm

Inspired by the iterative optimization algorithm proposed by Kar *et al.*,[6] we design a simple iterative algorithm to derive a feasible solution satisfying all the constraints in the problem **LC**.

Let $x_{ij}^{(n)}$ be the allocated rate on link (i,j), $\forall (i,j) \in \mathcal{A}$, at the n^{th} step. Let

$$\lambda_j^{(n)} = \max(0, \mathcal{R} - \sum_{i:(i,j)\in\mathcal{A}} x_{ij}^{(n)}),$$

and

$$e_{ij}^{(n)} = \begin{cases} 1 & \text{if } \sum_{j:(i,j)\in\mathcal{A}} x_{ij}^{(n)} > \mathcal{O}_i, \\ & \text{or } x_{ij}^{(n)} > \mathcal{C}_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$

We update $x_{ij}^{(n)}$ by:

$$x_{ij}^{(n+1)} = \begin{cases} x_{ij}^{(n)} & \text{if } \lambda_j^{(n)} = 0, e_{ij}^{(n)} = 0, \\ x_{ij}^{(n)} + \alpha_n \lambda_j^{(n)} & \text{if } \lambda_j^{(n)} > 0, e_{ij}^{(n)} = 0, \\ x_{ij}^{(n)} - \beta_n e_{ij}^{(n)} & \text{if } \lambda_j^{(n)} = 0, e_{ij}^{(n)} > 0, \\ x_{ij}^{(n)} + \alpha_n \lambda_j^{(n)} - \beta_n e_{ij}^{(n)} & \text{if } \lambda_j^{(n)} > 0, e_{ij}^{(n)} > 0, \end{cases} \tag{4}$$

where α_n and β_n are two sequences with the following properties:

$$\lim_{n\to\infty} \alpha_n = 0, \sum_{n=1}^{\infty} \alpha_n = \infty, \lim_{n\to\infty} \beta_n = 0, \sum_{n=1}^{\infty} \beta_n = \infty, \lim_{n\to\infty} \frac{\alpha_n}{\beta_n} = 0. \tag{5}$$

For example, the sequences $\alpha_n = \frac{a}{n}$ and $\beta_n = \frac{b}{\sqrt{n}}$, where a, b are positive constants, satisfy the above properties.

In each step n, $\lambda_j^{(n)}$ represents how much more bandwidth peer j needs to acquire in order to achieve the required streaming rate $\mathcal{R}$. $e_{ij}^{(n)}$ can be understood as a *binary* indicator of insufficient bandwidth, showing whether available bandwidth is exceeded along overlay link (i,j): either the upload capacity of peer i is exceeded, or $x_{ij}^{(n)}$ goes beyond the available bandwidth on overlay link (i,j).

The intuition behind the updates in (4) is as follows: Whenever an overlay link does not have sufficient bandwidth, the allocated rate along the link is reduced; whenever the aggregate rate on the download links of a peer falls below $\mathcal{R}$, the allocated link bandwidths are increased, according to how much the aggregate rate deviates from $\mathcal{R}$. α_n and β_n denote the step lengths of the updates. The increment step length α_n is much smaller than the decrement step length β_n for sufficiently large n, and both of them are diminishing. These are important properties to guarantee the convergence of the algorithm to a feasible solution of **LC**, as will be used in our convergence analysis.

3.2 Practical implementation in the synchronous case

Our iterative algorithm can be readily implemented in a fully distributed fashion. We first study its decentralized implementation in the synchronous case, where updates are synchronized to occur at times $n = 1, 2, \ldots$. In the subsequent section, we will show that, with minor modifications, the implementation can also achieve feasible bandwidth allocation in asynchronous and dynamic environments.

In the synchronous case, the allocated bandwidth on a link (i,j) is adjusted at the downstream peer j during the actual streaming process. The media streams are transmitted from upstream peers at the allocated transmission rates, x_{ij}'s, using a bandwidth shaping mechanism.

In our implementation, at times $n = 1, 2, \ldots$, peer j calculates $\lambda_j^{(n)}$ based on the discrepancy between $\mathcal{R}$ and the currently allocated rates on its download links, *i.e.*, $\lambda_j^{(n)} = \max(0, \mathcal{R} - \sum_{i:(i,j)\in\mathcal{A}} x_{ij}^{(n)})$. If $\lambda_j^{(n)} > 0$, it increases the

allocated rates by $x_{ij}^{(n)'} = x_{ij}^{(n)} + \alpha_n \lambda_j^{(n)}$, $\forall i : (i,j) \in \mathcal{A}$; otherwise, it sets $x_{ij}^{(n)'} = x_{ij}^{(n)}$. Meanwhile, peer j estimates the actually achieved receiving rate $y_{ij}^{(n)}$ from each of its upstream peers, by dividing the number of bytes received on each link in a time interval by the interval length. It sets $e_{ij}^{(n)} = 1$ if the actual receiving rate is lower than the allocated rate on the link, *i.e.*, $y_{ij}^{(n)} < x_{ij}^{(n)}$, or sets $e_{ij}^{(n)} = 0$ otherwise. It then proceeds to update the allocated rates again by $x_{ij}^{(n+1)} = x_{ij}^{(n)'} - \beta_n e_{ij}^{(n)}$, $\forall i : (i,j) \in \mathcal{A}$, and requests these new transmission rates from its respective upstream peers. After an upstream peer receives the new requested rates $x_{ij}^{(n+1)}$ from all its downstream peers, it adjusts its sending rates to the new values.

We note that our implementation does *not* depend on any *a priori* knowledge of peer upload and overlay link bandwidth, nor any feedback from IP-layer routers. In our implementation, the value of *insufficient bandwidth indicator* $e_{ij}^{(n)}$ on each link (i,j) is inferred by comparing the allocated transmission rate from upstream peer i with the achieved receiving rate at downstream peer j during the streaming process. The rationale behind this is that when an allocated transmission rate is larger than a respective receiving rate, bandwidth insufficiency is implied either at the upstream peer or on the overlay link, *i.e.*, $e_{ij}^{(n)} = 1$; otherwise, no bandwidth limit is exceeded along link (i,j), and $e_{ij}^{(n)} = 0$. In this way, although we formally formulate the problem **LC** with $\mathcal{O}_i$ and $\mathcal{C}_{ij}$ in (2) and (3), respectively, we do not actually need to perform any bandwidth probing to explicitly derive the values of these bandwidth limits.

3.3 Convergence analysis

We now analyze the convergence of our decentralized implementation of the iterative algorithm, in the synchronous case. To facilitate our analysis, we consider the realistic scenario that, if the aggregate requested (sending) rate at an upstream peer i is higher than its upload capacity, the receiving rates at all its downstream peers are lower than their respective requested rate, *i.e.*, each of them is able to detect the bandwidth insufficiency. Our discussion is divided into two cases: (1) a feasible solution exists for **LC** , *i.e.*, $X_R \cap X_C \neq \phi$; and (2) a feasible solution does not exist for **LC**, *i.e.*, $X_R \cap X_C = \phi$.

Theorem 1 shows that, when a feasible solution exists, *i.e.*, there is sufficient bandwidth in the overlay to support all peers at the required streaming rate $\mathcal{R}$, the decentralized implementation of our iterative algorithm converges to such a solution.

Theorem 1. If $X_R \cap X_C \neq \phi$, with iterative updates in (4) and diminishing step lengths in (5), the sequence $\{x^{(n)}\}$ converges to $\tilde{x}$, a feasible solution of problem **LC**, *i.e.*, $\tilde{x} \in X_R \cap X_C$.

Theorem 2 addresses the second case, when a feasible bandwidth allocation solution does not exist, *i.e.*, the overlay cannot accommodate all the peers at $\mathcal{R}$. Theorem 2 states that, at all the peers, our implementation is able to achieve the maximum throughput supported by the overlay.

Let $\mathcal{R}_{\max}$ be the maximum throughput at the peers that the network can support, *i.e.*, the maximum aggregate streaming bandwidth that each peer can acquire. It implies that there exist feasible solutions to the following problem:

LC':

$$\begin{aligned} \sum_{i:(i,j)\in\mathcal{A}} x_{ij} &\geq \mathcal{R}_{\max}, \qquad \forall j \in \mathcal{N}, \\ \sum_{j:(i,j)\in\mathcal{A}} x_{ij} &\leq \mathcal{O}_i, \quad \forall i \in \mathcal{S} \cup \mathcal{N}, \\ x_{ij} &\leq \mathcal{C}_{ij}, \quad \forall (i,j) \in \mathcal{A}. \end{aligned} \tag{6}$$

Theorem 2. If $X_R \cap X_C = \phi$, with iterative updates in (4) and diminishing step lengths in (5), the sequence $\{x^{(n)}\}$ converges to the feasible region of problem **LC'**, *i.e.*, $\lim_{n\to\infty} \rho(x^{(n)}, X') = 0$, where X' is the region defined by the constraints in **LC'**.

Due to space constraints, interested readers are referred to our technical report[7] for complete proofs of both theorems. The key intuition behind the proofs is to show that, based on the diminishing step lengths, in each step of the iterative algorithm, the current bandwidth allocation improves towards a feasible solution of **LC** or approaches the feasible region of **LC'**. Based on these theorems, a corollary follows:

Corollary 1. During the convergence of $\{x^{(n)}\}$, the actually achieved streaming rate at each peer j, i.e., $\sum_{i:(i,j)\in\mathcal{A}} y_{ij}^{(n)}$, is asymptotically increasing and converges to $\mathcal{R}$ if the network can support such rate at each peer, and $\mathcal{R}_{\max}$ otherwise.

During the dynamic process of bandwidth allocation, a peer's achieved streaming rate may temporarily decrease when the allocated rates on its download links decrease due to lack of available bandwidth. However, over time, this achieved streaming rate is asymptotically increasing until it reaches $\min(\mathcal{R}, \mathcal{R}_{\max})$.

4. DECENTRALIZED BANDWIDTH ALLOCATION: THE ASYNCHRONOUS CASE

Granted, while important as a first step in our study, the synchronous case that we have considered is an idealistic view of practical peer-to-peer networks. Peers are inherently *asynchronous*, with different processing times and messaging latencies. Fortunately, with minor modifications, we can extend our decentralized implementation to the asynchronous case, with the ability to handle peer and network dynamics.

In an asynchronous overlay, if we execute our decentralized implementation previously proposed for the synchronous case, the step lengths at a certain time t, $\alpha(t)$ and $\beta(t)$, are not identical at all the peers, as peers update the allocated rates at their own paces. However, it is the key to guarantee algorithm convergence by updating bandwidth allocation synchronously with the same step lengths across the network, as used in proofs of Theorem 1 and 2. Thus the iterative synchronous implementation may fail to converge in the asynchronous case.

Fortunately, we are able to show that, the update process can still be proven to converge to a feasible solution of **LC** in an asynchronous environment, if each peer follows the *synchronous update rule* across its own download and upload links. More rigorously, on a downstream peer j, all increments of allocated rates on all its download links are performed at the same time t, and use a same diminishing step length $\alpha_j(t)$, *i.e.*, $x_{ij}(t+1) = x_{ij}(t) + \alpha_j(t)\lambda_j(t), \forall i : (i,j) \in \mathcal{A}$. On the other hand, on an upstream peer i, all decrements of allocated rates on all its upload links are performed at the same time t, and use a same diminishing step length $\beta_i(t)$, *i.e.*, $x_{ij}(t+1) = x_{ij}(t) - \beta_i(t)e_{ij}(t), \forall j : (i,j) \in \mathcal{A}$.

We are now ready to present our decentralized implementation in the asynchronous case, and show its convergence to a feasible solution. In the asynchronous implementation, the allocated rate on a link (i,j) is adjusted with the cooperation of both upstream peer i and downstream peer j, *i.e.*, increment at peer j and decrement at peer i. To implement this, rate updates and inferred values of the *insufficient bandwidth indicators* need to be passed between upstream and downstream peers in special protocol messages, referred to as *Rate Update (RU)* messages. These protocol messages are delivered using reliable transport protocols such as TCP.

Our decentralized asynchronous implementation executed at each peer i proceeds as follows.

Initialization:

1. Initialize the set of current upstream peers $\mathcal{U}_i$ and downstream peers $\mathcal{D}_i$, as well as the step counters $n_i = 1$, and $m_i = 1$.

2. For every upstream peer u in $\mathcal{U}_i$:

(2.1) Set $x_{ui} = \mathcal{R}/|\mathcal{U}_i|$ and $e_{ui} = 0$.

(2.2) Send x_{ui} and e_{ui} to peer u with a *RU* message.

Next, peer i executes the following steps in its dual roles as a downstream peer and an upstream peer, using step counters n_i and m_i, respectively.

As a downstream peer:

1. Receive *RU* messages from its upstream peers, and estimate the actually achieved receiving rate y_{ui} from each of them. Adjust $\mathcal{U}_i$ if it detects any upstream peer failures.

2. After it has received *RU* messages from all its existing upstream peers, do the following:

(2.1) Retrieve allocated rates $x_{ui}(t)$, $\forall u \in \mathcal{U}_i$, from the received *RU* messages.

(2.2) Compute $\lambda_i(t) = \max(0, \mathcal{R} - \sum_{u:(u,i)\in\mathcal{A}} x_{ui}(t))$.

(2.3) For each upstream peer u:

(2.3.1) If $\lambda_i(t) > 0$, increase the allocated rate by $x_{ui}(t+1) = x_{ui}(t) + \alpha_{n_i}\lambda_i(t)$; otherwise, set $x_{ui}(t+1) = x_{ui}(t)$.

(2.3.2) If $y_{ui} < x_{ui}(t)$, set $e_{ui}(t+1) = 1$; otherwise, set $e_{ui}(t+1) = 0$.

(2.3.3) Send $x_{ui}(t+1)$ and $e_{ui}(t+1)$ to peer u with a *RU* message.

3. Increment step counter: $n_i = n_i + 1$.

As an upstream peer:

1. Receive *RU* messages from its downstream peers. Adjust $\mathcal{D}_i$ if it detects any downstream peer failures, or receives *RU* messages from new downstream peers.

2. After it has received *RU* messages from all its existing downstream peers, do the following:

For each downstream peer j:

(2.1) Retrieve $e_{ij}(t)$ and $x_{ij}(t)$ from the *RU* message from peer j.

(2.2) If $e_{ij}(t) = 1$, decrease the allocated rate by $x_{ij}(t+1) = x_{ij}(t) - \beta_{m_i}e_{ij}(t)$; otherwise, set $x_{ij}(t+1) = x_{ij}(t)$.

(2.2) Adjust the sending rate to peer j to $x_{ij}(t+1)$, and send $x_{ij}(t+1)$ in a *RU* message to peer j.

3. Increment the step counter: $m_i = m_i + 1$.

Theorem 3 shows the convergence of our decentralized asynchronous implementation of the iterative algorithm.

Theorem 3. With our decentralized asynchronous implementation, and under the assumption that both the message passing delay and the time between consecutive updates are finite, the sequence $\{x(t)\}$ (the rate vector at time t) converges to a feasible solution of **LC** if $X_R \cap X_C \neq \phi$, or to the feasible region of **LC'** otherwise.

Again, interested readers are referred to our technical report[7] for the proof of Theorem 3. The key to guarantee the convergence is, as pointed out earlier, when a downstream peer i updates the allocated rates on its download links, it increases them altogether with the same diminishing step length α_{n_i}; when an upstream peer i updates the allocated rates on its upload links, it deducts them altogether with the same diminishing step length β_{m_i}. In this case, the bandwidth allocation still improves towards feasibility in each step.

We conclude our discussion in the asynchronous case with an important note that, our asynchronous implementation can maximally guarantee the required streaming bandwidth at each peer, in cases of both peer and network dynamics. To understand such robustness against dynamics, we consider the influence of dynamics *during* and *after* the convergence process.

Dynamics during convergence. Our asynchronous implementation of the algorithm can readily adapt to dynamics introduced before the allocated rates converge. *First,* when a new peer i *joins* the streaming session, it is assigned an initial set of upstream peers, which is decided by the topology construction protocol based on media content availability, and executes the *initialization* phase. After its selected upstream peers receive its *RU* messages, they include peer i in their respective sets of downstream peers. Thus, peer i can immediately participate in the asynchronous implementation as a downstream peer from the initial step counter $n_i = 1$, while its upstream peers continue their own execution with their current step counter values. *Second*, in the case of peer failures or departures, after the downstream and upstream peers detect peer i's failure or departure, they simply remove i from their respective sets of upstream or downstream peers and continue with their execution, effectively excluding i from later message exchanges and computation. *Finally*, our implementation naturally adapts to fluctuating overlay link bandwidth due to the appearing and vanishing of congestion along the links, since our implementation uses binary indicators of insufficient bandwidth e_{ij}, rather than explicit *a priori* knowledge of link bandwidth.

To derive the convergence properties of the asynchronous implementation in dynamic networks, the analysis in Theorem 3 still applies, *i.e.*, we can still show the bandwidth allocation dynamically improves towards feasibility in the current network, and converges to one feasible solution if there exists one in the dynamic network. Formally, we present it as Corollary 2, which can be obtained directly from Theorem 3:

Corollary 2. If dynamics occur during the convergence of our asynchronous implementation of the iterative algorithm, the rate vector $x(t)$ improves towards feasible bandwidth allocation of the current overlay, and converges to a feasible solution whenever there exists one, or maximizes the peer throughput in the dynamic overlay.

Dynamics after convergence. If dynamics occur after allocated rates have converged, the affected peers initiate a new round of protocol execution with reset step counters, in which the bandwidth allocation continues to improve towards feasibility or throughput maximization.

The handling of the case that a new peer joins the streaming session is similar to that discussed when dynamics occur during convergence, except that all peers now execute the protocol from $n_i = 1$ or $m_i = 1$. In the cases of peer failure or departure, or overlay link bandwidth fluctuations, which have caused the loss of streaming rate at a particular downstream peer, the affected downstream peer will reallocate its bandwidth requests towards its remaining upstream peers, and send out the new *RU* messages to each of them. In this way, a new round of protocol execution is invoked, and the involved peers reset their step counters and cooperate in the protocol execution from their current bandwidth allocation.

To conclude our analytical discussions of bandwidth allocations in peer-to-peer streaming sessions, we reiterate our motivation for this work. *Why* do we need to perform bandwidth allocation in a particular P2P topology, when P2P topologies are highly dynamic? We believe that our bandwidth allocation algorithms are *complementary* to existing topology construction protocols (*e.g.*, random peer selection), and may help these protocols so that they are more effective, especially in the case of tight supply-demand relationships of bandwidth. Once a P2P topology is first constructed using a particular protocol, our bandwidth allocation algorithm can be used to maximally utilize the existing supply of bandwidth in such a topology. If the required streaming rate is not satisfied on all the peers in the topology, the topology construction protocol can be reactivated to construct better topologies. After the topology change, our bandwidth allocation algorithm can come into play again, which achieves the highest level of peer streaming rate satisfaction in the new topology. With our bandwidth allocation algorithm, the need for topology reconstruction, *i.e.*, to find new upstream peers and adjust peer connectivity, is minimized, and the feasible streaming rate is achieved quickly at the peers to counter the effects of network dynamics.

5. PERFORMANCE EVALUATION

To evaluate the effectiveness, in particular the dynamic behavior of convergence, we have conducted an in-depth empirical study on our decentralized asynchronous implementation proposed in Sec. 4 with C++-based simulations. We choose to simulate our implementation in the asynchronous case due to its practicality in realistic P2P topologies. The first several proof-of-concept experiments use small example topologies, akin to the concept of "unit testing" in software development.

Experiment A (behavior of convergence): We first illustrate how the implementation converges in an example topology in Fig. 2. We simulate a streaming session with $\mathcal{R} = 800$ Kbps. In this network, peers p_1 and p_2 directly stream from the server (not shown in Fig. 2), and serve as "mini-sources" for p_3, p_4 and p_5. The rate vector is $x = (x_{13}, x_{23}, x_{14}, x_{34}, x_{25}, x_{35})$. End-to-end delays on overlay links are $(30, 50, 60, 100, 80, 120)$ (in milliseconds), respectively. We investigate both feasible and infeasible cases.

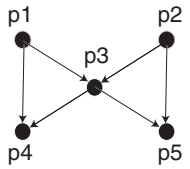

Figure 2. An example peer-to-peer streaming topology.

The feasible case: The overlay topology is able to support p_3, p_4 and p_5 at $\mathcal{R}$, with upload capacities of $(0.7, 0.8, 1.0)$ (in Mbps) at p_1, p_2, p_3, respectively, and available link capacities of $(0.4, 0.5, 0.5, 0.8, 0.6, 0.9)$ (in Mbps) on the six links.

The infeasible case: The overlay topology is unable to support p_3, p_4 and p_5 at $\mathcal{R}$, with upload capacities of $(0.7, 0.8, 0.6)$ (in Mbps) at p_1, p_2 and p_3. Available link capacities are the same as in the feasible case.

The step length sequences used in our experiments are $\alpha_n = 1/n$ and $\beta_n = 1/(10\sqrt{n})$. Results for the two experiments are illustrated in Fig. 3 and Fig. 4, respectively.

Fig. 3(a) depicts the convergence of allocated link rates, x_{ij}'s, in the feasible case, which are all initiated to 400 Kbps. In the first iteration, p_3 and p_4 find that there is insufficient bandwidth on link $(1, 3)$ and $(1, 4)$, respectively, as p_1's aggregate sending rate exceeds its upload capacity. Based on the feedbacks from its downstream peers, p_1 decreases x_{13} and x_{14}. In the next iteration, as the aggregate download link rates at p_3 and p_4 fall below $\mathcal{R}$, they increase the allocated rates on their download links. During the convergence, x_{34} and x_{35} keep increasing as p_3's spare upload capacity is being utilized, and

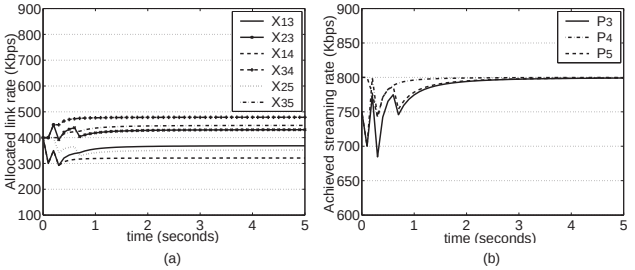

Figure 3. Convergence in the example topology: the feasible case.

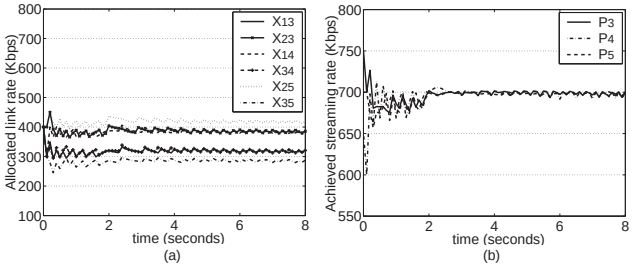

Figure 4. Convergence in the example topology: the infeasible case.

the rate vector quickly converges to a feasible solution, $(370, 430, 322, 478, 352, 448)$. Correspondingly, Fig. 3(b) shows the convergence of the actually achieved streaming rates at the peers during the bandwidth allocation process. Though there is a temporary decrease initially, these rates steadily increase to reach the required rate.

Fig. 4 illustrates the convergence in the infeasible case, in which the maximum throughput achieved at p_3, p_4 and p_5 is 700 Kbps. We observe that while their initial streaming rates are $750, 650, 700$ Kbps, respectively, the rates quickly converge to 700 Kbps at all three peers. This reveals that our implementation is able to achieve fair allocation among the peers, when there is insufficient "supply" of bandwidth in the overlay. The observations we have had in both feasible and infeasible cases have supported Corollary 1 in Sec. 3.

Experiment B (effects of dynamics): We next investigate how our implementation converges in dynamic environments, again with the example topology in Fig. 2. In this experiment, upload capacities at p_1, p_2 and p_3 are $(0.7, 1.0, 1.5)$ (in Mbps) and available link capacities on the six links are $(0.4, 0.8, 0.5, 0.8, 0.35, 0.9)$ (in Mbps). We study two cases, and show our results in Fig. 5(a) and (b), respectively.

Dynamics occur during convergence: p_1 and p_2 already exist in the overlay session. p_3 joins the session at time 0, p_4 joins at 1 second, p_5 joins at 2 seconds, and then p_1 leaves the network at 3 seconds.

Dynamics occur after convergence: p_1 and p_2 already exist in the overlay session. p_3 joins the session at time 0, p_4 joins at 1 seconds, and p_5 joins at 2 seconds. The available bandwidth on link $(1, 3)$ is then decreased to 0.1 Mbps at 6 seconds. Finally, p_1 leaves the session at 12 seconds.

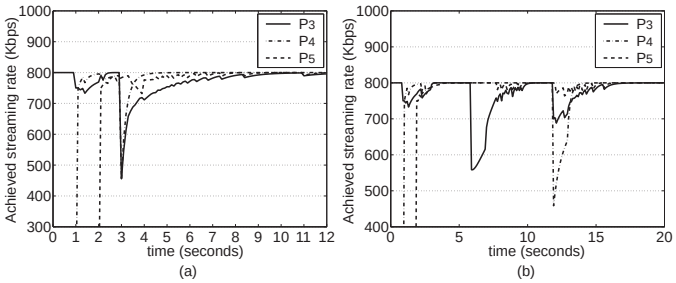

Figure 5. Convergence in the example topology: the dynamic case.

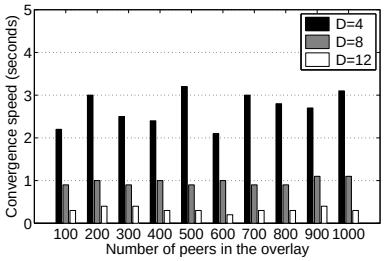

Figure 6. Converge speed in large random networks.

In the first case, comparing its results in Fig. 5(a) with Fig. 3(b), we can see that the convergence process is slightly prolonged due to peer dynamics, but is still performed rapidly. In the second case, after all three peers have joined the session and their rates stabilize, at 6 seconds, the decrease of available bandwidth on link $(1, 3)$ causes the decrease of p_3's streaming rate. As p_3 attempts to acquire additional bandwidth allocations from p_2, it further affects p_5. p_3 and p_5 then further adjust their bandwidth allocation, while p_4 is not affected. After the rates converge again, at 12 seconds, p_1's departure causes all three peers to cooperate in another round of rate adjustment, which quickly converges to a new feasible bandwidth allocation.

Experiment C (large networks): We are now ready to investigate how the asynchronous implementation of our iterative algorithm converges in more realistic and larger networks, generated with the BRITE topology generator.[8] In this set of experiments, peers are classified into two classes: 30% of them are Ethernet peers, with 10 Mbps upload capacities;

the remainder are ADSL peers, with $0.4 - 0.6$ Mbps upload capacities. The streaming server is an Ethernet host. The message passing delays on overlay links are sampled from the distribution of pairwise ping times between PlanetLab nodes.[9] Available link bandwidths are chosen from the distribution of measured capacities between PlanetLab nodes as well.[10] As our bandwidth allocation algorithm is orthogonal to peer selection strategies used by the streaming applications, we apply random peer selection as our topology construction protocol. The experiment is further divided into two parts.

Exp. C. 1: To investigate the scalability of the protocol, we examine its convergence speed in networks of different sizes and various numbers of upstream peers for each peer (D) in a static setting, without peer dynamics.

The results in Fig. 6 show that our algorithm scales very well with the increase of network sizes. This reveals that, in realistic large networks, by adjusting its bandwidth allocation with peers in its neighborhood, each peer can quickly obtain the required streaming bandwidth. The convergence is faster when a peer has more upstream peers.

Exp. C. 2: We then simulate a practical dynamic streaming session with $\mathcal{R} = 800$ Kbps, and monitor the achieved streaming rates at the peers during a 10-minute period. In the session, 200 peers join and depart following an On/Off model, with On/Off periods both following an exponential distribution with an expected length of T seconds. Each peer executes the following: Upon joining, it randomly selects D upstream peers and executes the bandwidth allocation algorithm. During such execution, if its achieved streaming rate is below $\mathcal{R}$ for 2 seconds, it randomly adds a new upstream peer if it currently has fewer than D upstream peers (due to peer failures), or randomly switches to new upstream peers otherwise.

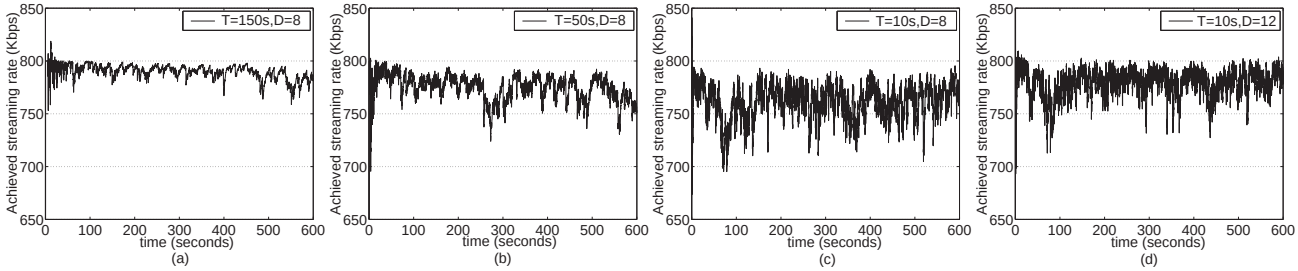

Figure 7. Average achieved streaming rate in a dynamic streaming session with 200 peers.

The results in Fig. 7 demonstrate that our algorithm can provide the peers with steady streaming rates under high peer churn rates. In the case that each peer joins/leaves every 10 seconds, with 200 peers, every second there are 20 peer joins/departures on average. Even in such an extremely dynamic environment, the streaming rates at existing peers are rather satisfactory any time during the streaming session. Comparing Fig. 7(c) with (d), we can see that the streaming rates are better if each peer has more upstream peers.

We have also experimented with varying available overlay link bandwidths. However, as the available link capacities sampled from those between PlanetLab nodes are generally much larger than $\mathcal{R}$, such variations do not materially affect our results. We choose not to present further details of these experiments.

6. RELATED WORK

When it comes to the problem of streaming bandwidth allocation in *mesh* overlay topologies, existing peer-to-peer streaming systems either use TCP, TFRC[11] or UDP with heuristic traffic shapers.[1,2,12,13] For those using TCP or TFRC,[1,12] transmission rates are adjusted on a per-link basis, and there is no guarantee that the required streaming rate can be provided to each participating peer. In our bandwidth allocation, we explicitly take the streaming bandwidth demand into consideration in the rate adjustment at each peer across all its downloading links. Delivering media packets over UDP, GridMedia[2] applies traffic shapers at a sending peer for each of its downstream peers. While the main purpose of such traffic shapers is to guarantee smooth delivery and avoid packet bursts, it is not discussed how to allocate the upload capacity when it is lower than the total requested rate from the downstream peers. In Promise,[13] each downstream peer assigns a sending rate to each of its upstream peers based on their advertised rates. The problem is not addressed with respect to how the sender should carefully allocate upload capacity, if multiple peers wish to stream from it at high rates. Similarly, in another well-known pull-based peer-to-peer mesh streaming system, Chainsaw,[3] the peers are sending packets as their bandwidths allow, but it is not specified how the sending rates towards different neighbors are to be regulated. In PRIME,[14] Magharei *et al.* suggest that each P2P connection in the mesh streaming overlay should have roughly the same bandwidth

in order to maximize the utilization of peer access link bandwidth, but have not emphasized on the achievement of this in practice.

With respect to resource allocation in overlay multicast applications or for general network flow problem, there exist studies that establish optimization models and propose distributed solution algorithms.[15–17] A fundamental requirement of these algorithms is that the available bandwidths in the network are known *a priori*, and the network topology is static during algorithm convergence. This is generally not realistic in practice. In contrast, our algorithm only utilizes end-to-end feedbacks, without depending on any *a priori* knowledge of capacities, and also adapts well to peer dynamics.

Optimization based approaches have been applied in Internet congestion control among multiple unicast flows.[6,18–20] While earlier work mainly deals with elastic traffic and aims to maximize concave user utility functions, there have been recent discussions on congestion control with sigmoidal-like utility functions,[21,22] which are closer to the streaming scenario we consider. Nevertheless, fundamental differences lie between our work and these existing work. First, all of the existing work discuss rate control among point-to-point connections, while we consider many-to-many mesh topologies in peer-to-peer streaming. Second, previous work requires feedback from IP-layer routers along the paths of the flows, while only end-to-end peer feedback is required in this paper. Finally, we model the bandwidth allocation problem as a feasibility problem, which represents simpler solution algorithm and faster convergence, as opposed to an optimization problem, and also well addresses the practical consideration that being able to guarantee a feasible streaming bandwidth for all peers is sufficient in realistic systems.

As mentioned earlier, our iterative algorithm is inspired by the work of Kar *et al.*[6] However, their iterative algorithm is used to solve the same user utility maximization problem, as discussed by all the previous work on congestion control cited above. Our proposed algorithm is significantly different, as we seek to address a completely different problem of practical bandwidth allocation in peer-to-peer streaming.

7. CONCLUDING REMARKS

In this paper, we have proposed a practical algorithm that allocates the limited "supply" of bandwidth in peer-to-peer streaming sessions, so that the "demand" of streaming playback rates can be satisfied on all the peers. We model the problem of bandwidth allocation as a feasibility problem, defined as a set of linear constraints. We further propose an iterative algorithm, which converges to a feasible solution if it exists, adapts well to dynamics, and can be implemented in a fully decentralized fashion, in both the synchronous and asynchronous cases. As a salient advantage of our algorithm, we do not rely on *a priori* knowledge of available upload and link bandwidth. As future work, we are interested in devising practical schemes to achieve prioritized bandwidth allocation in the case of multiple streaming sessions.

REFERENCES

1. X. Zhang, J. Liu, B. Li, and T. P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Live Media Streaming," in *Proc. of IEEE INFOCOM 2005*,
2. M. Zhang, L. Zhao, Y. Tang, J. Luo, and S. Yang, "Large-Scale Live Media Streaming over Peer-to-Peer Networks through Global Internet," in *Proc. of ACM Multimedia 2005*,
3. V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *Proc. of the Fourth International Workshop on Peer-to-Peer Systems (IPTPS'05)*, 2005.
4. X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Trans. on Multimedia (to appear)* , November 2007.
5. C. Wu, B. Li, and S. Zhao, "Magellan: Charting Large-Scale Peer-to-Peer Live Streaming Topologies," in *Proc. of the 27th International Conference on Distributed Computing Systems (ICDCS 2007)*, June 2007.
6. K. Kar, S. Sarkar, and L. Tassiulas, "A Simple Rate Control Algorithm for Maximizing Total User Utility," in *Proc. of IEEE INFOCOM 2001*,
7. C. Wu and B. Li, "On Meeting P2P Streaming Bandwidth Demand with Limited Supplies," tech. rep., http://iqua.ece.toronto.edu/papers/meetbwdemand.pdf, ECE, University of Toronto, June 2007.
8. A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Boston University Representative Internet Topology Generator," tech. rep., http://www.cs.bu.edu/brite, 2000.
9. "All-Sites-Pings for PlanetLab," (http://ping.ececs.uc.edu/ping/).
10. "PlanetLab IPerf," (http://jabber.services.planet-lab.org/php/iperf/).

11. M. Handley, S. Floyd, J. Pahdye, and J. Widmer, “TCP Friendly Rate Control (TFRC) : Protocol Specification,” (RFC 3448), Jan 2003.
12. D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, “Bullet: High bandwidth data dissemination using an overlay mesh,” in *Proc. of ACM SOSP 2003*, 2003.
13. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, “PROMISE: Peer-to-Peer Media Streaming Using Collect-Cast,” in *Proc. of ACM Multimedia 2003*,
14. N. Magharei and R. Rejaie, “PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming,” in *Proc. of IEEE INFOCOM 2007*, May 2007.
15. Y. Cui, Y. Xue, and K. Nahrstedt, “Optimal Resource Allocation in Overlay Multicast,” in *Proc. of 11th International Conference on Network Protocols (ICNP 2003)*, November 2003.
16. D. S. Lun, N. Ratnakar, R. Koetter, M. Medard, E. Ahmed, and H. Lee, “Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding,” in *Proc. of IEEE INFOCOM 2005*,
17. D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, 1989.
18. F. P. Kelly, A. Maulloo, and D. Tan, “Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability,” *Journal of the Operational Research Society* **49**, pp. 237–252, March 1998.
19. S. H. Low and D. E. Lapsley, “Optimization Flow Control, I: Basic Algorithm and Convergence,” *IEEE/ACM Transactions on Networking* **7**, pp. 861–875, December 1999.
20. R. Srikant, *The Mathematics of Internet Congestion Control*, Birkhauser, 2004.
21. J. Lee, R. R. Mazumdar, and N. B. Shroff, “Non-convexity Issues for Internet Rate Control with Multi-class Services: Stability and Optimality,” in *Proc. of IEEE INFOCOM 2004*, 2004.
22. M. Chiang, S. Zhang, and P. Hande, “Distributed Rate Allocation for Inelastic Flows: Optimization Frameworks, Optimality Conditions, and Optimal Algorithms,” in *Proc. of IEEE INFOCOM 2005*, 2005.