Multi-Tree Multicast Backpressure Algorithm for Content Distribution

Chunglae Cho and Ye Xia

Abstract

This paper addresses the problem of utility maximization of multicast sessions with multiple trees for content distribution over static infrastructure networks. The utility functions are general concave functions. Inspired by the derivation of the queue-length-based backpressure algorithms, we introduce a problem formulation with *tree-flow conservation* constraints and derive a multicast backpressure algorithm. The algorithm is much more distributed and local than previous tree-based algorithms for content distribution. We provide a rigorous analysis of the performance including the primal optimality and the bound on real queue sizes using the Lyapunov optimization technique and the convex optimization techniques together. To apply the Lyapunov optimization technique, we need to construct some feasible solution for comparison by defining an ϵ-modified problem. We show that there are alternative ways of doing this by modifying different constraints and they give different performance bounds. We also provide simulation results to evaluate the algorithm performance.

Index Terms

Content Distribution, Multicast, Multiple Trees, Distributed and Local Algorithm, Backpressure, Optimal Rate Allocation.

I. Introduction

Massive content distribution has become one of the most important applications on the Internet. Some of the examples are the delivery systems of television programs or video content, file-sharing systems, and e-science networks. One important class of content distribution technique is *swarming*. In a swarming session, a file is broken into many chunks at the source node, which are then distributed to the receivers through various paths.

Despite its origin in the end-system-based peer-to-peer (P2P) community, swarming is also attractive for content distribution services over infrastructure networks provided by network

C. Cho is with Electronics and Telecommunications Research Institute, Daejeon, 305-700, Korea. This work was done when he was with the Department of Computer and Information Science and Engineering, University of Florida. E-mail: clcho@etri.re.kr.

Y. Xia is with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611. E-mail: yx1@cise.ufl.edu.

or service providers. For those services, infrastructure nodes are strategically placed by the providers and are well managed and relatively static. In this setting, it has been shown that it is beneficial to view swarming as distribution over *multiple* multicast trees [1], [2]. This view allows us to focus on a formal approach based on optimization theory to develop optimal solutions systematically. Furthermore, it is often easier to first develop sophisticated algorithms under the static assumption, and then, adapt them to practical situations.

In this paper, we study how to distribute content over multiple multicast trees optimally. We suppose that each multicast session is associated with a single source and a set of receivers. Each session has infinite data backlog at its source and is given a small number of trees. Data will be divided and then distributed over the trees to the receivers. The optimization objective is to maximize the sum of the session utilities, each of which is function of the admitted traffic rate. We assume that the utility functions are general concave functions, and the limitation in the link capacities poses the major constraints. Then, our problem is to find a rate control and link scheduling algorithm that leads to an optimal tree rate allocation while maintaing the boundedness of the network queues.

We are interested in developing distributed and local algorithms where rate control and link scheduling are done locally at each node. To achieve the goal, we resort to the class of queue-length-based, backpressure algorithms, which have been studied previously mostly for unicast network flow problems [3]–[13] and occasionally for multicast problems [14]–[19]. In a backpressure algorithm, each node makes a transmission decision based on the queue size information at the node itself and its neighboring nodes. Such an algorithm can be not only decentralized but also local: There is no global exchange of control messages.

Our contributions are as follows. First, we develop a backpressure-based adaptive-control algorithm for the utility maximization problem of multi-tree multicast, which is amenable to implementation in large network systems. While it is relatively straightforward to develop such an algorithm for unicast problems, it is less so in the multicast case. The key to deriving our backpressure algorithm is a novel problem formulation that specifically incorporates the *tree-flow conservation* constraints: The amount of flow on a link on a tree should be no less than the amount of flow on its parent link with respect to the tree. We then relax the tree-flow conservation constraints and write down a subgradient algorithm for the corresponding static optimization problem, which then leads to the backpressure algorithm. Thus, we have a systematic method for developing backpressure algorithms for multi-tree multicast. As shown in the literature review later, few of the existing algorithms for multicast utility optimization are of the backpressure type. Much of the subsequent discussion focuses on how our work is compared to the few exceptions.

Our second contribution is technical, and it also marks a more substantial distinction of this

work from the closest related work (e.g., [14] [15]). We have obtained more complete results about the algorithm optimality and network stability, and we have been able to do so by using a combination of two sets of analytical techniques. The results include both primal and dual optimality and the boundedness of both the virtual queues and the real queues, under general (non-strictly) concave utility functions.

We will briefly discuss the two sets of techniques and what they can accomplish for us. It is difficult to apply the conventional convex optimization theory and techniques for subgradient algorithms to show the primal optimality of our algorithm. This is mainly due to the assumption of general concavity rather than strict concavity on the utility functions. The consequence is the lack of a continuous map from the dual to the primal variables, which is crucial to the proof of primal convergence as described in [20], [21]. Even if the dual variables converge, the primal variables may oscillate. Instead, we show the primal optimality of the algorithm in the long-time average sense using the Lyapunov optimization technique (see [7], [9], [22]). For network stability, however, the Lyapunov optimization technique can only show the boundedness of the long-time average virtual queue sizes[1], which is not enough to conclude network stability. We use the convex optimization techniques to prove the dual optimality, and then, we use that result to show that both the virtual and real queues are bounded *for all time slots*. The two sets of analytical techniques complement each other in our analysis.

The third contribution is a technical point with respect to the application of the Lyapunov optimization technique. To apply the Lyapunov optimization technique, one of the most important steps is to construct some feasible solution for the purpose of comparison with the proposed algorithm. This usually involves defining an ϵ-tightened (or ϵ-modified) problem. We show two ways of doing this by modifying different constraints. They give different performance bounds. Explorations of this kind can be useful for finding tighter performance bounds.

Prior work either has substantially different problem formulations, or uses different classes of algorithms, or is not as comprehensive in performance analysis. We first contrast our results with the work in [14]–[19], which have introduced similar problems and backpressure-based solutions. In [14], the authors present a backpressure algorithm for a stability problem with random arrivals to the sources whereas we consider an optimization problem with infinite backlogs at the sources. The analytical technique of [14] is Lyapunov drift analysis for stability as in [3], which is not the same as the Lyapunov optimization technique. Neither are there any convex optimization results or analysis. The authors in [15] consider a multicast backpressure algorithm, but for a wireless network[2]. The main similarity between [15] and this study is that the backpressure algorithms

[1]Our algorithm uses the virtual queue sizes for control.

[2]The discussion on [15] is restricted to the part about single-rate multicast, which is more relevant to our problem. The focus of [15] is in fact on multi-rate multicast.

are derived from subgradient algorithms. Other than that, there are substantial differences in the optimization problems themselves, the details of the algorithms, and the analytical results and techniques. The difference in the problem formulation is that they restrict to the case of a single tree per session, and, instead of the wireline link capacity constraints, they have the wireless network capacity constraint. The problems and algorithms are different enough that new proofs are needed even for the part of the results that are derived from the Lyapunov optimization technique. Furthermore, [15] contains no convex optimization results or analysis. Another major difference is that real queues are used for control in [14] and [15] whereas we use virtual queues. Because of this difference, the performance issues we investigate have a new twist: Although our algorithm is based on the virtual queues, ultimately, we wish to prove the boundedness of the real queues. As a result, the proof techniques used in [14], [15] are not sufficient, and new ones are needed. We mention in passing that, while using real queues may be associated with ease of implementation under some circumstances, using virtual queues may also have certain advantages over using real queues. For instance, it may lead to significantly smaller real queue sizes, and hence, lower delay, as shown in [11].

In [16]–[19], the authors consider multi-rate multicast problems with a single multicast tree for each session. Their objectives are with respect to the receivers' utilities rather than the sessions. Hence, those problems are different from ours. Moreover, these studies are not concerned with the boundedness of the real queues. In [18], the performance objective is maxmin fairness, and it is not clear how to extend its results to other types of fairness. In [16], [17], [19], the authors assume that the utility functions are strictly concave, which is a key condition used in their proofs. Therefore, their proof techniques cannot be applied to our case with non-strict concave utility functions.

We next discuss some less related work. The use of the optimization approach on multicast problems has been reported in [1], [2], [23]–[26]. However, the resulting algorithms are not of the backpressure type and all of them require global exchange of control messages, i.e., the source nodes and links belonging to the same session exchange information about the session rates and link prices. In [1], [23], special objectives are considered such as maximizing throughput or minimizing network congestion. In [24]–[26], the authors assume that the network bottlenecks are at the nodes' uplink capacities whereas we assume that the bottlenecks can be anywhere in the network. In [1], [2], although an optimal set of trees with an optimal rate allocation is found among all possible trees, the algorithms are less local and the computation requirement at each iteration is much higher than the algorithm proposed in this paper.

The unicast multi-path utility maximization problems have been studied in the literature [6], [20], [27]–[29]. In [20], [28], the authors attempt to solve the oscillation problem of the primal variables due to the lack of strict concavity by using the proximal optimization method [30]. The

algorithms are distributed but less local than ours. In [6], a backpressure subgradient algorithm is presented for unicast. Since the authors assume strictly concave utility functions, they can show the primal optimality using the convex optimization techniques.

The remaining paper is organized as follows. Section II describes the network model and problem formulation. In Section III, we present the backpressure algorithm. In Section IV, we show that our algorithm achieves an optimal solution and the virtual queues are bounded in the long-time average sense. We provide stronger results on queue boundedness in Section V. We give simulation results in Section VI. The concluding remarks are in Section VII.

II. Problem Description

A. Network Model

Consider a network which is represented by a directed, edge-capacitated graph $G = (V, E)$, where V is the set of nodes and E is the set of directed links. Each link e in E has a finite capacity $c_e > 0$. Let $c_{max} \triangleq \max_{e \in E} c_e$ be the maximum link capacity over all links. Let S be the set of all multicast sessions in the network and $|S|$ be the number of sessions. Each session is has one source and a set of destinations (receivers), which are all members of V. Each session is given a set of trees for distributing the session's data. The root node of each tree is the source and the leaf nodes are all the receivers of the session.

For each session s, let T_s denote the set of its distribution trees and let $|T_s|$ be the number of trees in T_s[3]. Let T be the union of T_s. Note that if $t_1 \in T_{s_1}$ and $t_2 \in T_{s_2}$ have the same topology, where s_1 and s_2 are two different sessions, we regard them as different trees. Let r_e^t be the transmission rate assigned for tree t on link e. Let E_t be the set of links on tree t. Denote $b(e)$ and $d(e)$ to be the transmitting (tail) node and the receiving (head) node of link e, respectively. Let V_t be the set of transmitting nodes $b(e)$ for all e in E_t. V_t represents the set of nodes on tree t which are not leaves. Let $o(t)$ be the root node of tree t. Note that, for every $t \in T_s$, $o(t)$ is the source of session s.

Let $p(t, e)$ be the parent link of link e on tree t. Let $\Omega(t, n)$ be the set of outgoing links from node n on tree t. Let x_s be the admitted rate for session s, and y_n^t be the admitted rate at node n on tree t. We assume that each source has an infinite backlog of data. Let Q_e^t be the real queue size at link e for tree t. When without confusion, we may also use Q_e^t to refer to the queue instead of the queue size.

Fig. 1 illustrates the network model and the notations by showing a part of a network consisting of some links used by two trees, t_1 and t_2, of session s. The root of both trees is the node labeled

[3]We assume that each session is given a small number of trees. One may use the results of previous work [2], [19], [23], [31] to find a good candidate set of multicast trees. Our algorithm does not scale well when all possible trees are allowed for each multicast session.

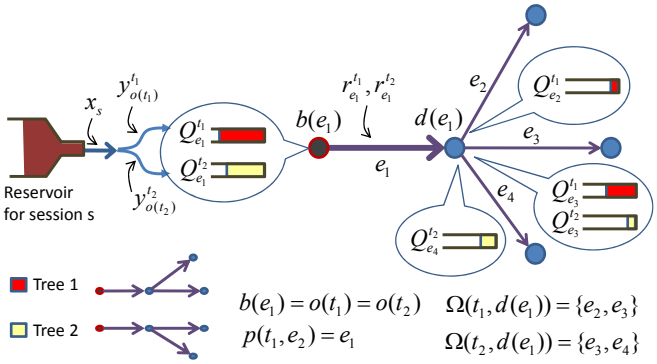


Fig. 1. An example network.

$b(e_1)$, which is also the source of session s. There is a reservoir of data for session s at the source node $b(e_1)$. The source pushes the data from the reservoir into the network at the admitted rate x_s. The admitted data is distributed over the two trees at rates $y^{t_1}_{o(t_1)}$ and $y^{t_2}_{o(t_2)}$, respectively. To do that, some of the admitted data is pushed into the queue $Q^{t_1}_{e_1}$ and the rest into the queue $Q^{t_2}_{e_1}$ at the ratio $y^{t_1}_{o(t_1)} : y^{t_2}_{o(t_2)}$.

Note that the queues associated with a link are maintained at the transmitting node (tail node) of the link. For example, the queues associated with link e_1 are actually at node $b(e_1)$. On link e_1, the data for the two trees are transmitted at the allocated rates $r^{t_1}_{e_1}$ and $r^{t_2}_{e_1}$, respectively, assuming there are sufficient data in the queues $Q^{t_1}_{e_1}$ and $Q^{t_2}_{e_1}$. After being transmitted over link e_1, the part of the data that is distributed over tree t_1 arrives at node $d(e_1)$ and is replicated into the queues $Q^{t_1}_{e_2}$ and $Q^{t_1}_{e_3}$, which are associated with the outgoing links $\Omega(t_1, d(e_1)) = \{e_2, e_3\}$ from node $d(e_1)$ on tree t_1. Similarly, for tree t_2, the data transmitted on e_1 is replicated into the queues associated with $\Omega(t_2, d(e_1)) = \{e_3, e_4\}$. Link e_1 is the parent link of links e_2 and e_3 on tree t_1, and the parent link of links e_3 and e_4 on tree t_2.

B. Problem Formulation

Denote (a_i) to be the vector with entries a_i. Let x, y and r be the vectors $(x_s)_{s\in S}$, $(y^t_n)_{t\in T, n\in V_t}$, and $(r^t_e)_{t\in T, e\in E_t}$, respectively. The notation such as $(a)_{t\in T}$ means repeating the value a in all entries of the vector. Denote $\mathbf{1}$ to be the vector whose entries are all 1's.

Let $\bar{x}_s(k)$ be the time average of the admitted traffic rate at the source of session s up to time k, i.e.,

$$\bar{x}_s(k) \triangleq \frac{1}{k}\sum_{\kappa=0}^{k-1} x_s(\kappa).$$

Let $\bar{x}_s$ be the limit of $\bar{x}_s(k)$[4],

$$\bar{x}_s \triangleq \lim_{k\to\infty} \bar{x}_s(k).$$

We define $\bar{y}_n^t(k)$, $\bar{y}_n^t$, $\bar{r}_e^t(k)$ and $\bar{r}_e^t$ similarly.

The optimal control problem of the paper is to find an algorithm that allocates the rates, $x(k), y(k), r(k)$, on each time slot k so that the long-time average rates optimize the following static optimization problem $\mathbf{P}$.

$$\begin{aligned}
\mathbf{P}: \ \max \ & \sum_{s\in S} U_s(\bar{x}_s) \\
\text{s.t.} \ & \bar{r}_e^t - \bar{r}_{p(t,e)}^t - \bar{y}_{b(e)}^t \geq 0, \quad \forall t \in T, \forall e \in E_t, && (1)\\
& \sum_{t\in T_s} \bar{y}_{o(t)}^t = \bar{x}_s, \quad \forall s \in S, && (2)\\
& \sum_{t:e\in E_t} \bar{r}_e^t \leq c_e, \quad \forall e \in E, && (3)\\
& 0 \leq \bar{x}_s \leq X_{max}, \quad \forall s \in S, && (4)\\
& \bar{y}_n^t \geq 0, \quad \forall t \in T, \forall n \in V_t, && (5)\\
& \bar{r}_e^t \geq 0, \quad \forall t \in T, \forall e \in E_t. && (6)
\end{aligned}$$

The constraints in (1) imply that for every tree t, the allocated link rate on a link e on the tree should be no less than the sum of the allocated link rate at link e's parent link, which is $p(t,e)$, and the rate of exogenous arrivals to the tail node $b(e)$, which is $\bar{y}_{b(e)}^t$. These constraints can be considered as a relaxed form of the flow conservation constraints for trees. We assume that $\bar{r}_{p(t,e)}^t = 0$ if $p(t,e)$ is null. Note that, for each tree t, the tree rate variables $\bar{y}_n^t$ are defined for all non-leaf nodes $n \in V_t$. The variables $\bar{y}_n^t$ for $n \neq o(t)$ appear unnecessary because there is only one tree rate for each tree and it is sufficient to associate the tree rate with the root node of the tree. In fact, our algorithm always assigns 0 to $\bar{y}_n^t$ for $n \neq o(t)$. Nevertheless, we define $\bar{y}_n^t$ for all non-leaf nodes, which will be convenient in later performance analysis (see Section IV).

The constraints in (2) mean that the aggregate rate transmitted by the trees for a session must be equal to the session rate. The constraints in (3) are the link capacity constraints that the sum of the transmission rates on a link cannot exceed the link capacity. In (4), we assume the session rates must be bounded by X_{max} from above, where X_{max} is a constant.

[4]We temporarily assume that the limit exists. We shall replace $\lim$ with $\liminf$ or $\limsup$ in case that the limit does not exist.

Remark. When viewed as a static optimization problem, we can rewrite the decision variables of problem $\mathbf{P}$ without the bars to simplify the notations.

Let Λ be

$$\Lambda \triangleq \{(x, y, r)|(x, y, r) \text{ is feasible to problem } \mathbf{P}.\}.$$

We make the following assumption on Λ.

AS 1. *There exists a feasible solution of* $\mathbf{P}$ *such that the constraints in (1) hold with strict inequality.*

AS 1 holds as long as there exists a multicast tree for each session and all the link capacities are strictly greater than 0. Under such conditions, it is easy to find feasible solutions described in AS 1[5]. The key reason is that the flow-conservation constraints in (1) are written as inequalities instead of equalities.

We also define the sets $\Lambda_{(x,y)}$ and Λ_r as follows, respectively.

$$\Lambda_{(x,y)} \triangleq \{(x, y)|(x, y) \text{ satisfies the constraints in (2), (4) and (5).}\},$$

$$\Lambda_r \triangleq \{r|r \text{ satisfies the constraints in (3) and (6).}\}.$$

These notations will be used to simplify later expressions.

We have the following assumptions on the utility function $U_s(x_s)$.

AS 2. *U_s is a concave function. U_s is continuous and differentiable. The derivative of U_s is bounded on $[0, X_{max}]$.*

Note that U_s is a somewhat general concave function. It could be linear or non-strictly concave. We do not even assume that it is monotonically increasing. Under this assumption, which is more general than that in [6], [21], we do not have a continuous map from the dual variables to the primal variables in the subgradient algorithm. Such continuity is available in [6], [21] and is used to prove the primal optimality. The assumption also implies that U_s is bounded on $[0, X_{max}]$.

Different utility functions can reflect various objectives of network or service providers operating the content distribution service. For example, if they want to get proportional fairness among sessions, they can use $U_s(x_s) = \log(x_s + 1)$ as the utility functions for sessions[6]. On

[5]In particular, we can set all the $\bar{y}$ variables to 0. For each tree t, we let $\bar{r}^t_e - \bar{r}^t_{p(t,e)} = \delta$, where $\delta > 0$ is a small constant. That is, for each tree t, the rate allocation on a link e is slightly greater than the rate allocation on the parent link of e on tree t. We also set $\bar{x}_s = 0$ for all s. When δ is sufficiently small, the link capacity constraints in (3) are all satisfied. Thus, we have found a feasible solution to problem $\mathbf{P}$.

[6]Strictly speaking, we have to use $\log x_s$ to achieve exact proportional fairness. We use $\log(x_s + 1)$ to have the derivative of the utility bounded.

the other hand, if they want to maximize the weighted throughput, they can use $U_s(x_s) = w_s x_s$ where w_s is the weight for session s. More examples can be found in [32], [33].

III. Backpressure Algorithm

Let $[\cdot]^+$ and $[\cdot]_a^b$ denote the projection onto the non-negative domain and the interval of $[a, b]$, respectively. Let γ_e^t be the non-negative Lagrange multipliers associated with the constraints in (1). Let γ be the vector $(\gamma_e^t)_{t\in T, e\in E_t}$. From problem $\mathbf{P}$, we relax the constraints in (1) and write the Lagrangian as follows.

$$\begin{aligned}
&L(x, y, r; \gamma)\\
=&\sum_{s\in S} U_s(x_s) + \sum_{t\in T}\sum_{e\in E_t} \gamma_e^t (r_e^t - r_{p(t,e)}^t - y_{b(e)}^t)\\
=&\sum_{s\in S}\Big(U_s(x_s) - \sum_{t\in T_s} y_{o(t)}^t \sum_{e\in\Omega(t,o(t))} \gamma_e^t\Big)\\
&-\sum_{s\in S}\sum_{t\in T_s}\sum_{e\in E_t: e\notin\Omega(t,o(t))} y_{b(e)}^t \gamma_e^t\\
&+\sum_{e\in E}\sum_{s\in S}\sum_{t\in T_s: e\in E_t} r_e^t(\gamma_e^t - \sum_{e'\in\Omega(t,d(e))} \gamma_{e'}^t).
\end{aligned}$$

The last equality holds because, for all t,

$$\sum_{e\in E_t} y_{b(e)}^t \gamma_e^t = \sum_{e\in\Omega(t,o(t))} y_{b(e)}^t \gamma_e^t + \sum_{e\in E_t: e\notin\Omega(t,o(t))} y_{b(e)}^t \gamma_e^t,$$

where $b(e) = o(t)$ for $e \in \Omega(t, o(t))$.

Then, the dual function is given by

$$\begin{aligned}
&D(\gamma)\\
=&\max_{(x,y)\in\Lambda_{(x,y)}, r\in\Lambda_r} L(x, y, r; \gamma)\\
=&\max_{(x,y)\in\Lambda_{(x,y)}} \sum_{s\in S}\Big(U_s(x_s) - \sum_{t\in T_s} y_{o(t)}^t \sum_{e\in\Omega(t,o(t))} \gamma_e^t\Big)\\
&+\max_{(x,y)\in\Lambda_{(x,y)}} \sum_{s\in S}\Big(-\sum_{t\in T_s}\sum_{e\in E_t: e\notin\Omega(t,o(t))} y_{b(e)}^t \gamma_e^t\Big)\\
&+\max_{r\in\Lambda_r} \sum_{e\in E}\sum_{s\in S}\sum_{t\in T_s: e\in E_t} r_e^t(\gamma_e^t - \sum_{e'\in\Omega(t,d(e))} \gamma_{e'}^t) \qquad (7)\\
=&\max_{(x,y)\in\Lambda_{(x,y)}} \sum_{s\in S}\Big(U_s(x_s) - \sum_{t\in T_s} y_{o(t)}^t \sum_{e\in\Omega(t,o(t))} \gamma_e^t\Big)\\
&+\max_{r\in\Lambda_r} \sum_{e\in E}\sum_{s\in S}\sum_{t\in T_s: e\in E_t} r_e^t(\gamma_e^t - \sum_{e'\in\Omega(t,d(e))} \gamma_{e'}^t),
\end{aligned}$$

where the second equality holds because the constraints of the maximization can be separated over the three terms, and the last equality holds because the maximum of the second term in (7) is always zero. The dual problem is as follows.

$$\begin{aligned} &\min\ D(\gamma) \\ &\text{s.t.}\ \ \gamma_e^t \geq 0, \quad \forall t \in T, \forall e \in E_t. \end{aligned}$$

Following the standard subgradient method [34], we have a subgradient algorithm as follows.

$$(x(k), y(k), r(k)) = \arg \max_{(x,y)\in\Lambda_{(x,y)}, r\in\Lambda_r} L(x, y, r; \gamma(k)), \tag{8}$$

$$\gamma_e^t(k+1) = [\gamma_e^t(k) - \delta(r_e^t(k) - r_{p(t,e)}^t(k) - y_{b(e)}^t(k))]^+, \quad \forall t \in T, \forall e \in E_t, \tag{9}$$

where $\delta > 0$ is a step size.

Let $q_e^t(k) = (1/\delta)\gamma_e^t(k)$, which represents a *virtual* queue size at link e for tree t at time slot k. Let $q(k)$ be the vector $(q_e^t(k))_{t\in T, e\in E_t}$. Next, we describe the distributed algorithm in details.

A. Algorithm

The algorithm has three conceptual components. At each time slot k, the source of each multicast session runs the *session and tree rate control* component; the tail node of each link runs the *link rate allocation* component and the *virtual queue update* component on behalf of the link. The session and tree rate control component is also known as the source part of the algorithm. The rest two components are the link part of the algorithm.

Source Part of the Algorithm (Session and Tree Rate Control):

- At each time slot k, the source of each session s observes the queue sizes $q_e^t(k)$ for every link $e \in \Omega(t, o(t))$ for every multicast tree of the session, i.e., $t \in T_s$.
- It decides the admitted rate $x_s(k)$ and the tree rate $y_{o(t)}^t(k)$ for $t \in T_s$ by solving the following optimization problem using the steps described in Section III-B.

$$\begin{aligned} \max\ & \frac{1}{\delta}U_s(x_s) - \sum_{t\in T_s} y_{o(t)}^t \sum_{e\in\Omega(t,o(t))} q_e^t(k) \\ \text{s.t.}\ & \sum_{t\in T_s} y_{o(t)}^t = x_s, \\ & 0 \leq x_s \leq X_{max}, \\ & y_{o(t)}^t \geq 0, \quad \forall t \in T_s. \end{aligned} \tag{10}$$

Link Part of the Algorithm

(Link Rate Allocation):

- At each time slot k, the tail node of each link e observes the queue size $q_e^t(k)$ for every tree passing through link e, i.e., t such that $e \in E_t$; it also collects the queue size $q_{e'}^t(k)$ for every child link of e on tree t, where, again, t is a tree passing through e.
- The tail node of link e decides the allocated link rates, $r_e^t(k)$, for every tree t passing through e by solving the following optimization problem using the steps described in Section III-C. Here, the decision variables are $\{r_e^t\}_{t:e\in E_t}$.

$$\begin{aligned} \max \quad & \sum_{t:e\in E_t} r_e^t(q_e^t(k) - \sum_{e'\in\Omega(t,d(e))} q_{e'}^t(k)) \\ \text{s.t.} \quad & \sum_{t:e\in E_t} r_e^t \le c_e, \\ & r_e^t \ge 0, \ \forall t : e \in E_t. \end{aligned} \tag{11}$$

(Virtual Queue Update):

- At each time slot k, the tail node of each link e updates the virtual queues for every tree t such that $e \in E_t$ as follows:

$$q_e^t(k+1) = [q_e^t(k) - r_e^t(k) + r_{p(t,e)}^t(k) + y_{b(e)}^t(k)]^+. \tag{12}$$

It is not difficult to check that at each time slot k, the first two components - session and tree rate control and link rate allocation - together find a solution that maximizes the Lagrangian L given the dual variables $\delta q(k)$ (see (7) and (8)). This property is important in the later performance analysis. The virtual queue update component corresponds to (9).

We will show later that the parameter δ can be used to adjust the performance bounds (see Theorem 4). Rewriting the algorithm from (8)-(9) into (10)-(12) has the benefit that only the sources need to adjust δ if the performance bound needs to be changed.

With respect to the communication requirement of the algorithm, we will show that only local information exchange is needed, i.e, the exchange between a node and its neighbors. First, consider the source algorithm. Recall that $o(t)$ is the root node of tree t and $\Omega(t, o(t))$ is the set of outgoing links from the root node on tree t. Since the source of each session s is in fact the common root node of the session's multicast trees, the queue sizes $q_e^t(k)$, where $e \in \Omega(t, o(t))$ and $t \in T_s$, are observed locally at the source/root node. Second, in the link rate allocation component, each required $q_e^t(k)$ in (11) is available at the tail node of link e itself and each required $q_{e'}^t(k)$ can be transmitted from the head node of link e to the tail node. Third, in the virtual queue update component, the required $r_e^t(k)$, $r_{p(t,e)}^t(k)$ and $y_{b(e)}^t(k)$ in (12), which are decided by either the source algorithm or the link rate allocation component, are either available locally or can be obtained by local information exchange. Specifically, $r_e^t(k)$ is known locally at the tail node of link e. Link $p(t, e)$ is the parent link of e on tree t. If the parent link exists,

which happens when the tail node of link e is not the source of the session, $r^t_{p(t,e)}(k)$ can be transmitted from the tail node of link $p(t,e)$ to the tail node of link e. If the tail node of link e is the source of the session, $y^t_{b(e)}(k)$ can be obtained locally at the source; otherwise, $y^t_{b(e)}(k)$ is known to be 0.

Hence, the algorithm is well distributed and local because all the necessary information can be obtained either locally or from the neighboring nodes. Furthermore, since the total number of trees is reasonably small, each node can keep in its routing tables the local topology (i.e., the relevant outgoing links) of the trees that pass through node. With that, each data packet only needs to carry the identifier of the tree on which it is distributed.

B. Solving Subproblem (10)

Let $t^*_s(k)$ be a tree with the minimum total virtual queue backlog at the root of the tree. By total virtual queue backlog at the root, we mean the sum of the virtual queue sizes associated with those links on the tree coming out of the root of the tree. Mathematically,

$$t^*_s(k) \in \arg\min_{t\in T_s} \sum_{e\in\Omega(t,o(t))} q^t_e(k).$$

If there are more than one such trees, we pick one of them arbitrarily but deterministically (for ease of presentation, the algorithm does not involve randomization).

Then, we consider the following expression:

$$\zeta_s(x_s) \triangleq \frac{1}{\delta}U'_s(x_s) - \sum_{e\in\Omega(t^*_s(k),o(t^*_s(k)))} q^{t^*_s(k)}_e(k).$$

We set $x_s(k)$ as follows:

$$x_s(k) = \begin{cases} x_s, & \text{if } \zeta_s(x_s) = 0 \text{ where } x_s \in [0, X_{max}], \\ X_{max}, & \text{if } \zeta_s(x_s) > 0 \text{ for all } x_s \in [0, X_{max}], \\ 0, & \text{otherwise.} \end{cases}$$

Note that since U'_s may not be one-to-one, $x_s(k)$ can oscillate over time even if $q^t_e(k)$ stabilizes.

Next, for each $t \in T_s$, we set $y^t_{o(t)}$ to be

$$y^t_{o(t)}(k) = \begin{cases} x_s(k), & \text{if } t = t^*_s(k), \\ 0, & \text{otherwise.} \end{cases}$$

Hence, the source of the session selects only one tree that has the minimum total virtual queue backlog at the root of the tree, and all the admitted virtual traffic is dispatched onto the selected tree.

C. Solving Subproblem (11)

First, each link e finds the tree with the maximum differential backlog (with respect to the virtual queues); this tree is denoted by $\tau_e^*(k)$. Specifically, let $\beta_e^*(k)$ be the maximum different backlog, defined by

$$\beta_e^*(k) \triangleq \max_{t:e\in E_t}\{q_e^t(k) - \sum_{e'\in\Omega(t,d(e))} q_{e'}^t(k)\}. \tag{13}$$

Let $\tau_e^*(k)$ be the tree that solves (13) with tie broken deterministically.

Next, link e assigns $r_e^t(k)$ for each tree t on link e as follows:

$$r_e^t(k) = \begin{cases} c_e, & \text{if } t = \tau_e^*(k) \text{ and } \beta_e^*(k) \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

D. Moving Real Data

The algorithm given in Section III-A solves problem $\mathbf{P}$, a fact to be shown in Section IV. The algorithm maintains the virtual queues and computes the admitted rates, the tree rates, and the allocated link rates on each time slot. However, it says nothing about how to move the real data, which is the subject of this sub-section.

There are many ways of applying the algorithm in Section III-A to the distribution of the real data, which are originally stored in the reservoirs at the sources. One possibility is to run the algorithm until the various long-time average rates converge to the optimum, and only after that, use those optimal values to control the transmission/distribution of the real data. This approach incurs an initial delay before the actual distribution of the data.

We next describe an approach that does not incur the initial delay.

Source Part: Recall that, on time slot k, the source of each session s determines its session rate $x_s(k)$ and selected tree $t_s^*(k)$ as described in Section III-B.

- On time slot k, the source of session s pushes $x_s(k)$ amount of data from its reservoir to the real queues associated with the outgoing links of the selected tree $t_s^*(k)$ at the root node (the data is duplicated if there are multiple outgoing links).

Link Part: Recall that, on time slot k, the tail node of each link e determines the tree $\tau_e^*(k)$, which has the maximum differential backlog in the virtual queues, as described in Section III-C.

- If the differential backlog on tree $\tau_e^*(k)$ is negative, do nothing and return.
- The tail node of link e transmits c_e amount of data if it has sufficient data in the real queue associated with tree $\tau_e^*(k)$ and link e. Otherwise, it transmits all the data in that real queue and does not use the remaining link capacity[7].

[7]In the analysis of real queue boundedness, we need to assume that the remaining link capacity is not used.

- The receiving node moves (and duplicates if needed) the received data into the real queues associated with the child links of link e on tree $\tau_e^*(k)$.

IV. Long-time Average Performance Analysis

We show in this section that with the algorithm (10)-(12), the achieved utility can be arbitrarily close to the optimum and the virtual queue sizes are bounded in the long-time average sense. This result is given in Theorem 4. Lemmas 2 and 3 are intermediate steps to reach Theorem 4. Lemma 1 is used in the proof of Theorem 4 together with Lemma 3. The analysis follows the Lyapunov optimization technique [7], [9], [22].

Define $\mathcal{Y}$ to be

$$\mathcal{Y} \triangleq \{y | \exists (x, r) \text{ such that } (x, y, r) \text{ is feasible to } \mathbf{P}\}.$$

Let Y be the largest rate such that the vector $y_{sym} \triangleq (Y)_{t\in T, n\in V_t}$, whose entries are all equal to Y, is in $\mathcal{Y}$.

We consider the following ϵ-tightened problem $\mathbf{P}(\epsilon)$.

$$\begin{aligned} \mathbf{P}(\epsilon): \ \max \ & \sum_{s\in S} U_s(x_s) \\ \text{s.t.} \ & r_e^t - r_{p(t,e)}^t - y_{b(e)}^t \geq \epsilon, \quad \forall t \in T, \forall e \in E_t, \\ & \text{and the constraints (2) - (6)}, \end{aligned} \tag{14}$$

where $0 < \epsilon < Y$.[8] Problem $\mathbf{P}(\epsilon)$ is well defined for all ϵ such that $0 < \epsilon < Y$. Since y_{sym} is in $\mathcal{Y}$, $(Y - \epsilon)_{t\in T, n\in V_t}$ is a part of a feasible solution to problem $\mathbf{P}(\epsilon)$. Therefore, we can always find a feasible solution for all ϵ such that $0 < \epsilon < Y$.

The above problem formulation is obtained by tightening the constraints in (1) of problem $\mathbf{P}$. Such tightening has the following interpretation: We start with the situation described by problem $\mathbf{P}$. On top of that, for each tree and each non-leaf node of the tree, there is an additional traffic source that generates traffic at the rate ϵ. The vector y_{sym} is obtained by pushing the same traffic rate as much as possible not only into the root nodes of the trees but also into all other non-leaf nodes of the trees.

Define $\Lambda(\epsilon)$ and $\mathcal{Y}(\epsilon)$ to be

$$\Lambda(\epsilon) \triangleq \{(x, y, r) | (x, y, r) \text{ is feasible to } \mathbf{P}(\epsilon).\},$$
$$\mathcal{Y}(\epsilon) \triangleq \{y | \exists (x, r) \text{ such that } (x, y, r) \text{ is feasible to } \mathbf{P}(\epsilon).\},$$

[8]More precisely, by saying the variables (x, y, r) satisfy the constraints (2) - (6), we mean they satisfy (2) - (6) with (x, y, r) replacing $(\bar{x}, \bar{y}, \bar{r})$.

respectively, where $0 < \epsilon < Y$. It is easy to see that $\Lambda(\epsilon)$ and $\mathcal{Y}(\epsilon)$ are subsets of Λ and $\mathcal{Y}$, respectively. That is, any feasible solution of problem $\mathbf{P}(\epsilon)$ is feasible to problem $\mathbf{P}$.

Let (x^*, y^*, r^*) and $(x^*(\epsilon), y^*(\epsilon), r^*(\epsilon))$ be some optimal solutions of problem $\mathbf{P}$ and $\mathbf{P}(\epsilon)$, respectively. Note that $(x^*(\epsilon), y^*(\epsilon), r^*(\epsilon))$ is feasible to problem $\mathbf{P}$.

Remark: The optimal solution of the ϵ-tightened problem is used in the analysis. To get a proper performance bound, it is important to decide which constraints are tightened. Here, we tighten the constraints that were relaxed when we derived the algorithm. It allows us to easily associate ϵ with each of the virtual queues, which is crucial to get the virtual queue size bound. The point will become clearer in the proof of Theorem 4. In subsection IV-A, we will show other possibilities.

Define U_{max} to be

$$U_{max} \triangleq \sum_{s\in S} \max_{0\le x_s \le X_{max}} U_s(x_s).$$

Note that U_{max} is well defined. Let f^* and $f^*(\epsilon)$ be the optimal objective values of problem $\mathbf{P}$ and $\mathbf{P}(\epsilon)$, respectively. Then, we have the following lemma.

Lemma 1.

$$f^*(\epsilon) \to f^*, \ \textit{as} \ \epsilon \to 0.$$

We omit the proof of Lemma 1. Interested readers may refer to Chapter 5 in [35].

Let $\sum_{s,t,e}(\cdot)$ be the abbreviated notation of $\sum_{s\in S}\sum_{t\in T_s}\sum_{e\in E_t}(\cdot)$. Define the Lyapunov function $V(k)$ and the Lyapunov drift $\Delta(k)$ as follows:

$$V(k) \triangleq \sum_{s,t,e} (q_e^t(k))^2,$$

$$\Delta(k) \triangleq V(k+1) - V(k). \tag{15}$$

Then, we can get a bound for the Lyapunov drift as follows.

Lemma 2. *For any finite $\delta > 0$ and $0 < \epsilon < Y$, there exists a constant B such that for all time slots t and all virtual queue sizes $q(k)$, the Lyapunov drift satisfies*

$$\Delta(k) - \frac{2}{\delta}\sum_{s\in S} U_s(x_s(k)) \le B - \frac{2}{\delta} f^*(\epsilon) - 2\epsilon \sum_{s,t,e} q_e^t(k). \tag{16}$$

Proof: We will use the following inequalities later in the proof.

$$\sum_{s,t,e} \left(r_e^t(k) - r_{p(t,e)}^t(k) - y_{b(e)}^t(k) \right)^2$$

$$\le \sum_{s,t,e} (c_{max} + X_{max})^2. \tag{17}$$

Define the constant B as follows:

$$B \triangleq \sum_{s,t,e} (c_{max} + X_{max})^2.$$

By squaring the virtual queue evolution equation in (12) and followed by simple manipulation, we have

$$\begin{aligned}
&(q_e^t)^2(k+1) - (q_e^t)^2(k) \\
\leq\ & (r_e^t(k) - r_{p(t,e)}^t(k) - y_{b(e)}^t(k))^2 \\
&- 2q_e^t(k)(r_e^t(k) - r_{p(t,e)}^t(k) - y_{b(e)}^t(k)).
\end{aligned}$$

Summing over all $t \in T$ and all $e \in E_t$ and using the inequality in (17), we have

$$\begin{aligned}
&\Delta(k) \\
\leq\ & B - 2\sum_{s,t,e} q_e^t(k)\Big(r_e^t(k) - r_{p(t,e)}^t(k) - y_{b(e)}^t(k)\Big).
\end{aligned}$$

By adding the term $-(2/\delta)\sum_{s\in S} U_s(x_s(k))$ to the both sides of the above inequality, we have

$$\begin{aligned}
&\Delta(k) - \frac{2}{\delta}\sum_{s\in S} U_s(x_s(k)) \\
\leq\ & B - 2\sum_{s\in S}\Big(\frac{1}{\delta}U_s(x_s(k)) - \sum_{t\in T_s}\sum_{e\in E_t} y_{b(e)}^t(k) q_e^t(k)\Big) \\
&- 2\sum_{s,t,e} q_e^t(k)\Big(r_e^t(k) - r_{p(t,e)}^t(k)\Big). \qquad (18)
\end{aligned}$$

Define functions $\Psi(k)$ and $\Phi(k)$ as follows:

$$\begin{aligned}
\Psi(k) \triangleq\ & \sum_{s\in S}\Big(\frac{1}{\delta}U_s(x_s(k)) - \sum_{t\in T_s}\sum_{e\in E_t} y_{b(e)}^t(k) q_e^t(k)\Big), \\
\Phi(k) \triangleq\ & \sum_{s,t,e} q_e^t(k)\Big(r_e^t(k) - r_{p(t,e)}^t(k)\Big) \\
=\ & \sum_{e\in E}\sum_{s\in S}\sum_{t\in T_s: e\in E_t} r_e^t(k)\Big(q_e^t(k) - \sum_{e'\in\Omega(t,d(e))} q_{e'}^t(k)\Big).
\end{aligned}$$

Since the algorithm (10)-(11) greedily maximizes $\Psi(k)$ and $\Phi(k)$, we have the following inequalities for our algorithm.

$$\begin{aligned}
\Psi(k) \geq & \sum_{s\in S}\Big(\frac{1}{\delta}U_s(\tilde{x}^s) - \sum_{t\in T_s}\sum_{e\in E_t} \tilde{y}_{b(e)}^t q_e^t(k)\Big), \\
\Phi(k) \geq & \sum_{s,t,e} q_e^t(k)\Big(\tilde{r}_e^t - \tilde{r}_{p(t,e)}^t\Big),
\end{aligned}$$

where $\tilde{x}$, $\tilde{y}$ and $\tilde{r}$ are any nonnegative vectors such that $(\tilde{x},\tilde{y}) \in \Lambda_{(x,y)}$ and $\tilde{r} \in \Lambda_r$.

Since $(x^*(\epsilon), y^*(\epsilon), r^*(\epsilon))$ is feasible to problem $\mathbf{P}$, we have

$$\Psi(k) \geq \sum_{s\in S}\Big(\frac{1}{\delta}U_s(x_s^*(\epsilon)) - \sum_{t\in T_s}\sum_{e\in E_t} y_{b(e)}^{*t}(\epsilon)q_e^t(k)\Big). \tag{19}$$

Moreover, since $(x^*(\epsilon), y^*(\epsilon), r^*(\epsilon))$ is feasible to problem $\mathbf{P}(\epsilon)$, we have

$$r_e^{*t}(\epsilon) - r_{p(t,e)}^{*t}(\epsilon) - y_{b(e)}^{*t}(\epsilon) \geq \epsilon, \quad \forall t \in T, \forall e \in E_t.$$

Then, we have

$$\begin{aligned}\Phi(k) &\geq \sum_{s,t,e} q_e^t(k)\Big(r_e^{*t}(\epsilon) - r_{p(t,e)}^{*t}(\epsilon)\Big)\\ &\geq \sum_{s,t,e} q_e^t(k)(y_{b(e)}^{*t}(\epsilon) + \epsilon).\end{aligned} \tag{20}$$

Applying the above inequalities in (19) and (20) to the drift expression in (18) and canceling the common terms, we get the inequality in (16). ■

Next, we will show the long-time average of the achieved utility is lower-bounded by the optimal solution to the ϵ-tightened problem $\mathbf{P}(\epsilon)$. We will also show that the long-time average of the aggregate virtual queue size is bounded.

Lemma 3. *For any finite $\delta > 0$ and $0 < \epsilon < Y$, if there exists a constant B such that for all time slots k and all virtual queues $q(k)$, the Lyapunov drift satisfies the drift bound in (16), then we have the following long-time average performance and virtual queue size bounds.*

$$\liminf_{k\to\infty} \sum_{s\in S} U_s(\bar{x}_s(k)) \geq f^*(\epsilon) - \frac{\delta B}{2}, \tag{21}$$

$$\limsup_{k\to\infty} \frac{1}{k}\sum_{\kappa=0}^{k-1}\sum_{s,t,e} q_e^t(\kappa) \leq \frac{B + 2U_{max}/\delta}{2\epsilon}. \tag{22}$$

Proof: From (16), summing the inequality over $k \in \{0, 1, ..., K-1\}$, we get

$$\begin{aligned}&V(K) - V(0) - \frac{2}{\delta}\sum_{\kappa=0}^{K-1}\sum_{s\in S} U_s(x_s(\kappa))\\ &\leq BK - \frac{2K}{\delta}f^*(\epsilon) - 2\epsilon\sum_{\kappa=0}^{K-1}\sum_{s,t,e} q_e^t(\kappa).\end{aligned} \tag{23}$$

Using the fact that removing the terms $V(K)$ and $-2\epsilon\sum_{\kappa=0}^{K-1}\sum_{s,t,e} q_e^t(\kappa)$ preserves the inequality and rearranging the inequality in (23), we get

$$\frac{1}{K}\sum_{\kappa=0}^{K-1}\sum_{s\in S} U_s(x_s(\kappa)) \geq f^*(\epsilon) - \frac{\delta(B + V(0)/K)}{2}.$$

Taking the $\liminf$ as $K \to \infty$ yields the bound on the long-time average aggregate utility.

$$\liminf_{K\to\infty} \frac{1}{K} \sum_{\kappa=0}^{K-1} \sum_{s\in S} U_s(x_s(\kappa)) \geq f^*(\epsilon) - \frac{\delta B}{2}.$$

Using Jensen's inequality, we get (21).

On the other hand, using the property of $\sum_{s\in S} U_s(x_s) \leq U_{max}$ over $0 \leq x_s \leq X_{max}$, and the fact that removing the terms $V(K)$ and $-2Kf^*(\epsilon)/\delta$ preserves the inequality and rearranging the above inequality in (23), we get

$$\frac{1}{K} \sum_{\kappa=0}^{K-1} \sum_{s,t,e} q_e^t(\kappa) - \frac{V(0)}{2K\epsilon} \leq \frac{B + 2U_{max}/\delta}{2\epsilon}.$$

Taking the $\limsup$ as $K \to \infty$ yields the bound on the long-time average virtual queue size as in (22). ■

Using the above lemmas, we can derive the following theorem.

Theorem 4. *For any parameter $\delta > 0$, the algorithm (10)-(12) satisfies the following performance bounds.*

$$\liminf_{k\to\infty} \sum_{s\in S} U_s(\bar{x}_s(k)) \geq f^* - \frac{B\delta}{2}, \tag{24}$$

$$\limsup_{k\to\infty} \frac{1}{k} \sum_{\kappa=0}^{k-1} \sum_{s,t,e} q_e^t(\kappa) \leq \frac{B + 2U_{max}/\delta}{2Y}. \tag{25}$$

Proof: The inequalities in (21) and (22) hold for any ϵ such that $0 < \epsilon < Y$. Note that ϵ does not affect our algorithm. From (21), letting $\epsilon \to 0$, we get (24). From (22), letting $\epsilon \to Y$, we get (25). ■

Remark: The proofs for Lemmas 2, 3 and Theorem 4 are similar to some of the proofs in [9]. However, substantial differences exist due to the different problem formulations.

Theorem 4 implies that the long-time average of the solution given by the algorithm can be arbitrarily close to the optimal solution of problem $\mathbf{P}$ by choosing δ to be arbitrarily small. However, the bound on the long-time average of the virtual queue sizes increases as δ decreases. Therefore, the parameter δ can be used as a knob to tradeoff the optimization performance and virtual queue size bounds. It is not hard to derive an expression for Y. From the expression, one can see that the value of Y depends on the link capacities and the trees. Assuming the link capacities are held unchanged, the value of Y tends to be small if many deep links on different trees share a link. Hence, by Theorem 4, to make the queue size bound small, it would be preferable to construct trees such that each link is not shared by too many deep links on different trees.

In the parts of the algorithm that solve the subproblems (10) and (11), we break ties deterministically. If there is randomness from tie-breaking and possibly other sources, then one can add expectations to the relevant variables and expressions as in [9], and everything will go through with minor modifications.

The boundedness of the virtual queues does not directly imply the boundedness of the real queues, because the evolution of the two types of queues may become quite different. For example, in the link rate allocation step, all the link capacity is allocated to the currently selected tree. This allocated rate is used to update the virtual queues involved. But, any real queue is updated with the actual amount of real packets that enter or leave the queue. There is a possibility that the virtual queues and the corresponding real queues grow completely out of sync. Moreover, the algorithm only uses the virtual queue sizes for rate control. There is a possibility that the virtual queue sizes do not reflect the actual network congestion correctly. Therefore, in order to claim that the algorithm using the virtual queue sizes stabilizes the network, we must explicitly prove the real queue boundedness, which we will do in Section V.

A. Alternative Way to Prove the Algorithm Performance

In this subsection, we consider an alternative way to prove the algorithm performance by modifying different constraints. Although the alternative method does not necessarily lead to improved performance bounds, it is interesting to see that tightening the constraints in (1) is not the only way to show the algorithm performance.

We consider the first alternative way to proving the algorithm performance by reducing session flow rates by ϵ amount. Let X_{sym} be the largest admitted session rate x_s such that the vector $(X_{sym})_{s\in S}$ is a part of a feasible solution to $\mathbf{P}$. We may consider the following ϵ-modified problem $\mathbf{P_{A1}}(\epsilon)$.

$$\begin{aligned}\mathbf{P_{A1}}(\epsilon):\ \max\ & \sum_{s\in S} U_s(x_s)\\ \text{s.t.}\ & \sum_{t\in T_s} y^t_{o(t)} = x_s + \epsilon, \quad \forall s \in S,\\ & \text{and the constraints (1), (3) - (6),}\end{aligned}$$

where $0 < \epsilon < X_{sym}$.

Suppose that $(x^*(\epsilon), y^*(\epsilon), r^*(\epsilon))$ is an optimal solution of problem $\mathbf{P_{A1}}(\epsilon)$. Since $(x^*(\epsilon), y^*(\epsilon), r^*(\epsilon))$ is not feasible to problem $\mathbf{P}$, we cannot use it for the comparison with our solution. We can see that $(x^*(\epsilon), (y^{*t}_n(\epsilon) - \epsilon/|T_s|), r^*(\epsilon))$ and $(x^*(\epsilon) + \epsilon\mathbf{1}, y^*(\epsilon), r^*(\epsilon))$ are feasible to problem $\mathbf{P}$. Both feasible solutions are needed for the comparison in the proof. Then, we can derive the following performance bound.

Lemma 5. *For any finite $\delta > 0$, the algorithm (10)-(12) satisfies the following performance bounds.*

$$\liminf_{k\to\infty} \sum_{s\in S} U_s(\bar{x}_s(k)) \geq f^* - \frac{B\delta}{2}, \tag{26}$$

$$\limsup_{k\to\infty} \frac{1}{k} \sum_{\kappa=0}^{k-1} \sum_{s,t,e} q_e^t(\kappa) \leq \frac{(B + 2U_{max}/\delta) \max_{s\in S} |T_s|}{2X_{sym}}, \tag{27}$$

where B is the same constant as in Lemma 2.

We omit the proof of Lemma 5 since it is similar to those of Lemmas 2 and 3, and Theorem 4.

Note that, compared with Theorem 4, the term $\max_{s\in S} |T_s|$ may loosen the bound on the long-time average of the virtual queue sizes, whereas the term X_{sym} may tighten it.

Remark: In constructing the ϵ-tightened (or ϵ-modified) problem, the usual procedure is to modify the link capacity constraints as follows. From the problem formulation of $\mathbf{P}$, consider tightening the link capacity constraints as:

$$\sum_{t:e\in E_t} r_e^t \leq c_e - \epsilon, \quad \forall e \in E,$$

for ϵ satisfying $0 < \epsilon < c_{min}$, where $c_{min} \triangleq \min_{e\in E} c_e$. However, there seem to be difficulties in this approach and we were unable to find the right feasible solutions for the proof.

V. Further Results on Queue Boundedness

In this section, we show stronger results on the boundedness of the queues. In Lemma 10, we show that the virtual queue sizes at every time slot are bounded above uniformly. More importantly, in Theorem 11, we show that the real queue sizes at every time slot are bounded above uniformly. Lemmas 6, 7, and 8 are used in the proof of Lemma 9, which shows the convergence of the virtual queue vector.

A. Dual Optimality and Boundedness of Virtual Queues

The boundedness of the virtual queues can be proven from the dual optimality of our algorithm. In the following lemmas, we show that the scaled virtual queue vectors given by our algorithm approach the set of dual optimal solutions using the convex optimization techniques in [21], [34].

Let $J_e^t(y, r) \triangleq r_e^t - r_{p(t,e)}^t - y_{b(e)}^t$ and $J(y, r)$ be the vector $(J_e^t(y, r))_{t\in T, e\in E_t}$. Since x, y and r are bounded, there exists a constant $M_J < \infty$ such that

$$M_J \geq \max_{(x,y)\in\Lambda_{(x,y)}, r\in\Lambda_r} \|J(y, r)\|^2.$$

Let $\Gamma^* \triangleq \{\gamma^* \geq 0 | D(\gamma^*) = \min_{\gamma \geq 0} D(\gamma)\}$ be the set of optimal dual solutions of problem $\mathbf{P}$.

Lemma 6. *Γ^* is non-empty, closed and bounded.*

For finite $\eta > 0$ and $\gamma^* \in \Gamma^*$, let $\Gamma(\eta) \triangleq \{\gamma \geq 0 | D(\gamma) \leq D(\gamma^*) + \eta\}$. We have the following property of $\Gamma(\eta)$.

Lemma 7. *$\Gamma(\eta)$ is bounded.*

For brevity, we omit the proofs for Lemmas 6 and 7. We also have the following property of vector $J(y(k), r(k))$.

Lemma 8. *The vector $J(y(k), r(k))$ is a subgradient of D at $\delta q(k)$.*

We omit the proof for Lemma 8 since it is a corollary of known results (see [34], page 731).

Now, we are ready to show that the scaled virtual queue vectors $\delta q(k)$ can be arbitrarily close to Γ^* by choosing small enough $\delta > 0$. Let $d(\gamma, \Gamma^*) \triangleq \min_{\gamma^* \in \Gamma^*} \|\gamma - \gamma^*\|$ be the distance between γ and the nearest optimal solution of the dual. For finite $\eta > 0$, let $\xi(\eta) \triangleq \max_{\gamma \in \Gamma(\eta)} d(\gamma, \Gamma^*) + \eta$. The following lemma is a corollary of known results about the subgradient method [34]. We will sketch the proof for the lemma because the argument is also used to prove Lemma 10.

Lemma 9. *For any $\epsilon > 0$, there exist $\delta > 0$ and a sufficiently large $K_0 < \infty$ such that, with any finite initial feasible $q(0)$, for all $k \geq K_0$, $d(\delta q(k), \Gamma^*) < \epsilon$.*

Proof: For any $\gamma^* \in \Gamma^*$, we have

$$\|\delta q(k+1) - \gamma^*\|^2 \leq \|\delta q(k) - \gamma^*\|^2 + \delta^2 \|J(y(k), r(k))\|^2 + 2\delta(D(\gamma^*) - D(\delta q(k))). \tag{28}$$

Fix $\eta > 0$ and consider $\Gamma(\eta)$. Pick $\delta \leq \eta / M_J$. Then, as long as $\delta q(k) \notin \Gamma(\eta)$, i.e., $D(\delta q(k)) - D(\gamma^*) > \eta$, from (28), we get

$$\|\delta q(k+1) - \gamma^*\|^2 \leq \|\delta q(k) - \gamma^*\|^2 - \delta\eta.$$

The above inequality means that $\delta q(k)$ eventually enters the set $\Gamma(\eta)$.

On the other hand, if we pick $\delta \leq \eta / \sqrt{M_J}$, then once $\delta q(k) \in \Gamma(\eta)$, we have

$$\|\delta q(k+1) - \gamma^*\| \leq \|\delta q(k) - \gamma^*\| + \eta.$$

Since the above inequalities hold for any $\gamma^* \in \Gamma^*$, we get

$$d(\delta q(k+1), \Gamma^*) \leq d(\delta q(k), \Gamma^*) + \eta.$$

If we choose δ such that $\delta \leq \min\{\eta/M_J, \eta/\sqrt{M_J}\}$, then there exists a time K_0 such that $d(\delta q(k), \Gamma^*) \leq \xi(\eta)$ for all $k \geq K_0$.

Note that $\xi(\eta) \to 0$ as $\eta \to 0$. Since the dual function is continuous, $\max_{\gamma \in \Gamma(\eta)} d(\gamma, \Gamma^*) \to 0$, as $\eta \to 0$.

Then, for any $\epsilon > 0$, by picking $\eta > 0$ sufficiently small, we have $\xi(\eta) < \epsilon$. Therefore, there exists $\delta_0 = \min\{\eta/M_J, \eta/\sqrt{M_J}\} > 0$, such that for any $\delta \leq \delta_0$ and any initial feasible $q(0)$, there exists a time K_0 such that $d(\delta q(k), \Gamma^*) < \epsilon$ for all $k > K_0$. ■

In the next lemma, we show that the virtual queue backlogs at every time slot are bounded above in our algorithm.

Lemma 10. *For any finite $\delta > 0$, there exists a finite $M_q > 0$ such that, for every link e and tree t, the virtual queue size*

$$q_e^t(k) < M_q,$$

for all time slots k.

Proof: We will show $\sup_k \|q(k)\| < \infty$. Pick any $\gamma^* \in \Gamma^*$. Given a finite $\delta > 0$, we choose η such that $\eta = \max\{\delta M_J, \delta\sqrt{M_J}\}$. Since $\delta \leq \min\{\eta/M_J, \eta/\sqrt{M_J}\}$, by the same argument used in Lemma 9, there exists K_0 such that $d(\delta q(k), \gamma^*) \leq \xi(\eta)$. Then, by the boundedness of Γ^*, $\sup_k \|\delta q(k)\| < \infty$. Since $\delta > 0$, $\sup_k \|q(k)\| < \infty$. ■

B. Boundedness of Real Queues

To argue that the network is stabilized with the algorithm, we need to show that the real queue sizes are bounded. In [9], [15], the authors show the stability of their algorithms by claiming that the long-time average of the real queue sizes are bounded. In [6], the authors only show that the scaled virtual queue sizes are bounded. In this section, we show that with our algorithm, the real queue sizes at every time slot are bounded, which is a stronger result.

Let $Q_e^t(k)$ be the real queue size at link e for tree t at time slot k and $Q(k)$ be the vector $(Q_e^t(k))_{t \in T, e \in E_t}$. The algorithm (10)-(12) satisfies the following real queue bound.

Theorem 11. *There exists a finite $M_q > 0$ such that, for every link e and tree t, the real queue size*

$$Q_e^t(k) < M_q,$$

for all time slots k.

Proof: Let $R_e^t(k)$ be the amount of real traffic transmitted at link e on tree t at time slot k and $R(k)$ be the vector $(R_e^t(k))_{t \in T, e \in E_t}$.

Since $R_e^t(k) \leq r_e^t(k)$ for all time slots k, we have

$$\begin{aligned} & \sum_{u=k_0}^{k} \Big(R^t_{p(t,e)}(u) + y^t_{b(e)}(u) - r^t_e(u) \Big) \\ \leq & \sum_{u=k_0}^{k} \Big(r^t_{p(t,e)}(u) + y^t_{b(e)}(u) - r^t_e(u) \Big), \end{aligned} \tag{29}$$

for all k_0 and k such that $0 \leq k_0 \leq k$.

Assume that all the real and virtual queues are empty at time $k = 0$. Applying Loynes' formula and using the above inequality in (29), we have

$$\begin{aligned} Q^t_e(k) = & \max_{0 \leq k' \leq k} \sum_{u=k'}^{k} \Big(R^t_{p(t,e)}(u) + y^t_{b(e)}(u) - r^t_e(u) \Big) \\ \leq & \max_{0 \leq k' \leq k} \sum_{u=k'}^{k} \Big(r^t_{p(t,e)}(u) + y^t_{b(e)}(u) - r^t_e(u) \Big) \\ = & \, q^t_e(k+1). \end{aligned} \tag{30}$$

Since by Lemma 10, $q_e^t(k) < M_q$ for all k, we can conclude that $Q_e^t(k)$ is bounded above by M_q for every k. ■

VI. Simulation Results

In this section, we verify the correctness and evaluate the performance of the backpressure algorithm with simulation. We examine how the scaling parameter δ and the number of trees in a session affect the performance of the algorithm.

Given the focus and the scope of the paper, which is to develop and show the correctness of the algorithm, the objectives for conducting the simulation are limited to what is typical in the evaluation of an optimization algorithm, i.e., correctness and convergence speed. We have left out the effects of network imperfection because the simulator is implemented faithfully according to the description of the algorithm in the paper. In particular, the control information is never lost and it arrives at the intended destinations within one time slot. It is not our intention to report a more extensive evaluation of the algorithm under longer delays, non-zero losses and other network variability, since doing so first requires modification of the algorithm (see Section VII for a brief discussion on possible modifications). We leave it as future work to explore such modifications and conduct fuller evaluations of the modified algorithm in a general network setting.

A. Simulation Setup

We suppose that we run the backpressure algorithm over an ISP network where the data servers are strategically placed by the network operator and link capacities are exclusively allocated to the content distribution service. To make the setup realistic, we use the Sprintlink's ISP network topology obtained from the Rocketfuel project [36], which consists of 315 backbone nodes and 1944 links. Then, we attach 100 data servers randomly to some backbone nodes, with at most one server per backbone node. We assign link capacity 1000 to most of the backbone links except some critical links[9]. We assign relatively large link capacity 4000 to those critical links so that they will not become bottlenecks. We assign link capacity 3000 to the links between a data server and its attached backbone node.

In the experimental evaluations, the various quantities are kept unit-less. The algorithm iterates once every time slot. The convergence speed is measured in terms of the number of iterations, which is the same as the number of time slots. A link capacity of 1000 means 1000 data units per time slot, where the data unit can be chosen as needed, e.g., megabits, gigabits or bits. The unit of any rate is also in the number of data units per time slot. There are benefits in the unit-less approach, since we can interpret the quantities in various ways and one set of simulation results is applicable to networks of different parameters. For instance, suppose the time slot size is 1 second and the data unit is 1 megabits. Then, the unit for the link capacities and various rates is Mbps. The queue sizes are in megabits (Mb). In another example, suppose the time slot size is 0.1 seconds and the data unit is 10^5 bits. Then, the unit for the link capacities and various rates is again Mbps. However, the unit for the queue sizes are 10^5 bits.

We use the following heuristic to obtain the set of multicast trees for each session. We run the algorithm in [1], which computes the optimal set of trees (approximately), as well as their rate allocation and costs. The number of trees returned by that algorithm is usually very large when the network size is large. We choose to use a few of them in the increasing order of the tree cost.

Unless mentioned otherwise, the real data is injected from the source nodes into the network without an initial delay, as explained in Section III-D. In the discussion about the instantaneous queue sizes, we will see how the queue sizes are affected by various initial delays.

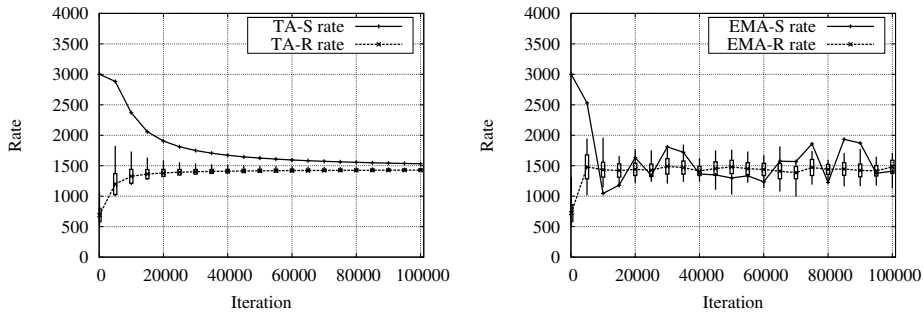
B. Performance Evaluation

1) Rates: We examine how the algorithm works for a single session[10]. The session consists of one source and 99 receivers. We choose $U_s(x_s) = x_s$ as the utility function. Therefore, the

[9]By critical links, we mean roughly the links that easily become bottleneck if they do not have sufficient capacity.

[10]We have also conducted experiments with multiple sessions. For brevity, those results are not shown.

objective is to maximize the throughput. The scaling parameter δ is set to be 10^{-5} and the number of trees is 10. For comparison purpose, if there is no restriction on the number of trees, the algorithm in [1] has eventually found 57 trees which form the optimal tree set and the optimal session rate is 1846, which is the maximum rate achievable by the backpressure algorithm.



(a) (b)

Fig. 2. Session rate and receiving rates; (a) time average; (b) exponential moving average.

The solid line in Fig. 2(a) shows the convergence of the time average session rate (TA-S). We also plot the time average of the receiving rate with a dotted line. Before reaching the steady state, the session rate can be different from each of the receiving rates because some packets are temporarily queued in the network. We show the time average rates at every 5000 iterations. Each point on the dotted line represents the average across the receivers of the time average receiving rates (TA-R) at the corresponding iteration. The box centered around each point marks one standard deviation above or below the average rate, and the end points of the vertical line on each point represent the maximum and the minimum (with respect to the rate samples collected across the receivers at the corresponding iteration). One can see that the average of the time average receiving rates eventually approaches the time average session rate. This suggests that the queues in the network will not grow indefinitely. Moreover, the standard deviation of the time average receiving rates also decreases as convergence takes place.

Even though the convergence of the long-time average rates appears slow, the short-time average rates experienced by the source and the receivers approach more quickly to the convergent values. We verify this by examining the exponential moving average of the rates, which is computed as

$$\tilde{x}_s(k) = (1-\nu)\tilde{x}_s(k-1) + \nu x_s(k),$$

where $\tilde{x}_s(k)$ is the exponential moving average of the session rate for session s at time k, and ν is a constant on $(0, 1)$. In our simulation experiments, ν is set at 0.1. We compute the exponential

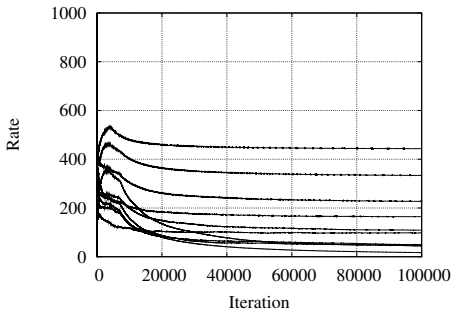


Fig. 3. Time average tree rates.

moving average of the receiving rates similarly. Although such an averaging operation retains the effect of the samples back to the very beginning, the contribution from the past samples diminishes very rapidly for $\nu = 0.1$. The resulting average reflects the short-time average, i.e., the average of the current and recent samples.

Fig. 2(b) shows the exponential moving average session rate (EMA-S) and the average of the exponential moving average receiving rates (EMA-R). It shows that the short-time average rates approach more quickly to the steady state values than the long-time average rates. The reason is that the rates assigned at the initial iteration are usually far from the optima and their effect is slow to dissipate in the long-time average.

Fig. 3 shows that the time average tree rates also converge. Some of the trees have tiny rates, which implies that we may use fewer trees without significant loss in the session rate. We will show the impact of the number of trees on the performance later.

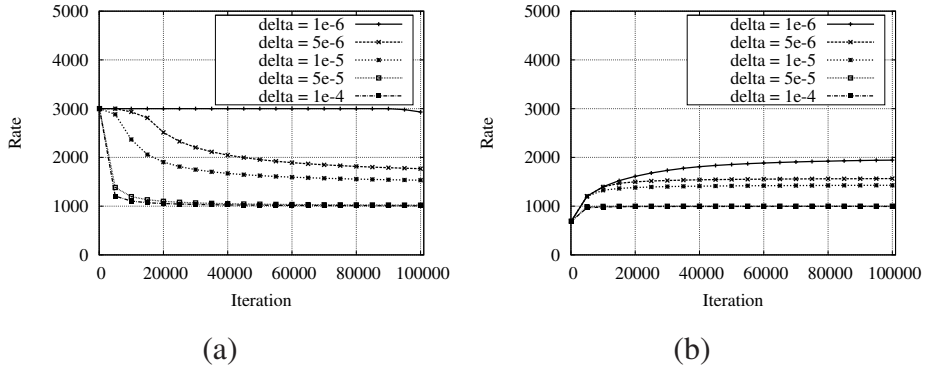


Fig. 4. Effect of δ on achievable rates; (a) time average session rates; (b) time average receiving rates.

We next show how the scaling parameter δ affects the performance of the algorithm. Fig. 4(a) shows that the achieved time average session rate tends to increase as δ decreases. Of course,

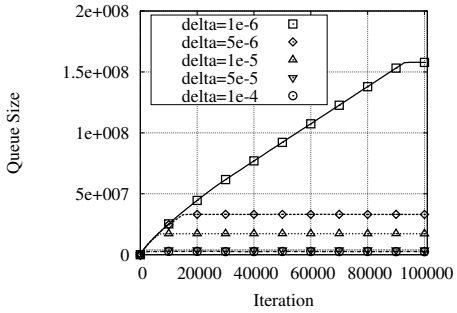


Fig. 5. Effect of δ on the queue sizes.

the amount of increase eventually becomes negligible as δ becomes sufficiently small[11]. This relationship can be seen better when the receiving rates are also examined, as shown in Fig. 4(b).

The scaling parameter δ affects the convergence speed of the session rate. As δ decreases, it takes longer for the rate to converge. For example, for $\delta = 5 \times 10^{-5}$, the time average session rate nearly stabilizes at around the 2×10^4th iteration, whereas for $\delta = 10^{-5}$, it does so at around the 8×10^4th iteration. Similar results can also be found for the time average receiving rates.

Through its effect on the convergence speed, the scaling parameter δ also affects the queue sizes in the network. In Fig. 5, we plot the trajectory of the total queue size in the network at every 1000 iterations for each δ. As δ decreases, the time it takes for the total queue size to stabilize increases and the total queue size increases as well.

Next, we show how the number of trees in the session affects the performance of the algorithm, in terms of the achievable rates. In this experiment, we set δ to be 10^{-5}. Fig. 6 shows that the achieved time average session rate tends to increase as the number of trees increases. However, beyond 5 trees, further improvement in performance is slight. On the other hand, with only 2 trees, the resulting session rate is only about 60 percent of what is achievable with 5 trees. Nevertheless, using 2 trees is still much better than using a single tree.

2) Queue Sizes and Buffer Requirements: Although they are not a focus of the paper, we will make some observations and comments about the queue size and buffer requirements. First, note that the simulation experiments are done under the condition that the buffers are infinite. As shown in Fig. 5 and Fig. 7, the total queue size grows rapidly initially during the transient phase of the algorithm, and later becomes stabilized during the steady state. The queues build up during the transient phase because the algorithm hasn't found the right rate allocation yet. Once

[11]With $\delta = 10^{-6}$, although the curve is becoming quite flat at the 10^5th iteration, the time average session rate has not fully converged. This is made clear by the corresponding receiving rate at that time, which is much less than the session rate.

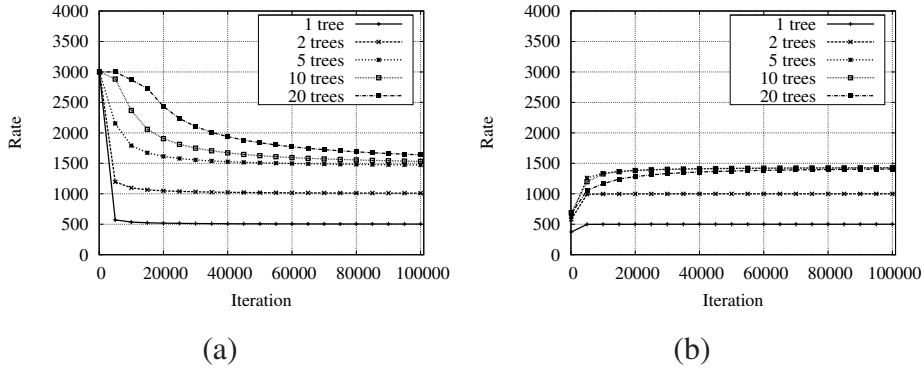


Fig. 6. Effect of the number of trees; (a) time average session rates; (b) time average receiving rates.

the time average rates approach the optimum (which is also feasible), the queues stop growing and their dynamics reaches the steady state. When the multicast sessions are long-lasting and the network topology and link capacities are unchanging, it is enough to decide the buffer sizes based on the steady-state queue behavior. Nevertheless, we will comment on both the transient and steady-state queues.

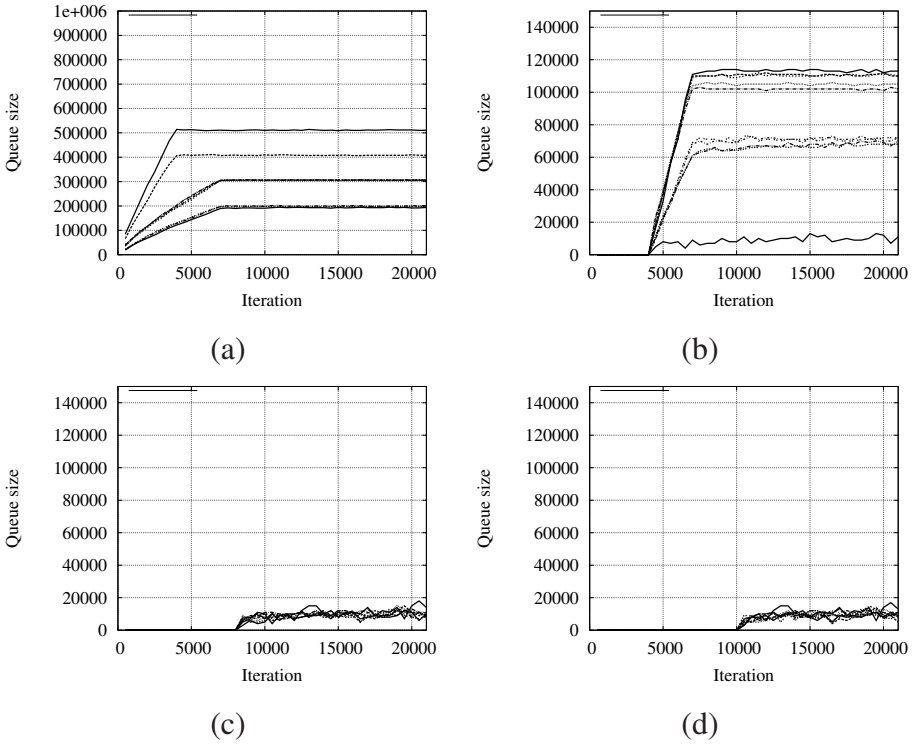


Fig. 7. Instantaneous queue sizes of the congested links; (a) RTIT = 1; (b) RTIT = 4000; (c) RTIT = 8000; (d) RTIT = 10000.

a) Transient Queue Sizes: In the basic algorithm, the transient queue sizes can be quite large. If that is a concern, one can modify the algorithm slightly by introducing an initial delay before the real data is injected into the network. The queue size reduction can be very significant. Fig. 7 shows the queue size trajectories of the ten links with the highest maximum congestion degrees. The *maximum congestion degree* of a link is defined as the maximum queue size of the link, observed over the duration of the simulation, divided by the amount of data the link can transmit per time slot[12]. We also define the *time average (TA) congestion degree* of a link to be the time average queue size of the link divided by the amount of data the link can transmit per time slot. The time slot at which the real data starts to be injected into the network is called the *real traffic injection time* (RTIT). In this set of experiments, δ is 10^{-5}. In Fig. 7 (a) through (d), the RTIT goes from 1, 4000, 8000 to 10000 time slots. All the ten most congested links in each case are found to have link capacity 1000 each.

With RTIT = 1, the queues of those congested links build up to large values within the first 4000 to 7000 iterations, after which they become stabilized. We have found that most of these congested links are in the neighborhood of the source node. The main reason for the queue buildup is that the source injects excessive data to the network during the transient phase before the algorithm finds the correct (optimal, and hence, feasible) time average rate. With an initial delay, the algorithm is given some time to get close to the correct rate so that the queues will be smaller compared to the case without a delay. Fig. 7 shows that, as the RTIT is increased from 1 to 8000, the queue sizes of the congested links decrease drastically. Of course, the improvement in the queue sizes is at the expense of extra delay in receiving the data. However, for long-lasting sessions in massive content distribution, the trade-off can be quite worthwhile. As Fig. 8 shows, the time average receiving rates jump up very quickly to the optimum after the initial delay.

b) Steady-State Queue Sizes: In the steady state, queues can still exist, although they are much smaller than the transient queues. The steady-state queues are caused by rate oscillations, which are part of the algorithm behavior. First, a multicast session hops among different multicast trees even in the steady state. As a result, some of the links can be temporarily overloaded. Second, the source rates may also change abruptly, rather than vary smoothly. The solid line in Fig. 9 (a) shows the maximum congestion degrees of all the links used by the session when the RTIT is 10000. The largest maximum congestion degree is 23 and the average degree across the links is 4.62. When the real data starts to be injected into the network at the 10000th time slot, the algorithm has already obtained a near-optimal rates. In some sense, the results shown as the solid line in Fig. 9 (a) correspond to the worst-case steady-state behavior (see more discussions later).

[12]This amount can be interpreted as the bandwidth-delay product.

From Fig. 9 (a), one may get the impression that the steady-state queue sizes are not sufficiently small, since they are larger than the bandwidth-delay product (which corresponds to a congestion degree 1). However, this isn't the complete picture. First, since the maximum congestion degree is proportional to the maximum queue size of the link over the observation period, it measures the worst-case congestion of the link during that time period. The time average congestion degree of the link is usually much smaller. The dashed line in Fig. 9 (a) represents the time average congestion degrees of the links. The worst time average congestion degree is 14.65 and the average of them across all the links is 2.36. Second and more importantly, in these experiments, the optimal rates found by the algorithm are binding with respect to the link capacity constraints at the critical (congested) links, achieving 100% link utilization there. Hence, the queue sizes accumulated due to the aforemetioned rate oscillations will stay steady at those links, as shown in Fig. 7 (d). In practice, rate-allocation algorithms rarely seek 100% link utilization but leave spare capacities in anticipation of traffic fluctuations and other contingencies. For instance, 80% link utilization is considered quite good. In that case, the capacity of a link exceeds the steady-state incoming traffic rate for the link and the steady-state queue will be small. Fig. 9 (b) shows the maximum and the time average congestion degrees when our algorithm uses only 80% of the link capacities in the rate computation. In this case, the average of the time average congestion degrees across all the links drops down to 1.31, which is nearly a half of the 100% link utilization case[13].

Finally, rate oscillations are non-essential features belonging only to the basic version of the algorithm. A modified, more sophisticated version can have much smoother dynamics, and hence, smaller steady-state queues. For instance, each session can split its traffic simultaneously over multiple multicast trees on each time slot and the source can ramp up its rate more smoothly (e.g., incrementally) in the absence of congestion.

VII. Conclusion and Discussion

In this paper, we have presented a backpressure algorithm for the utility maximization problem for multi-tree multicast. Compared with previous algorithms, our algorithm is not only distributed but local. It is not straightforward to show that our algorithm leads to both primal optimality and network stability due to the assumption of the general concave utility functions and the fact that the algorithm relies on virtual queue updates. We have shown that the two sets of analytical tools, the Lyapunov optimization technique and the convex optimization techniques, can complement each other and circumvent the difficulty. It is also interesting that there are alternative ways to define an ϵ-tightened (or ϵ-modified) problem in the proof of the algorithm performance using the Lyapunov optimization technique. Further exploration of the alternatives may be fruitful.

[13]Of course, this improvement is at the expense of about 20% down of the achieved rate.

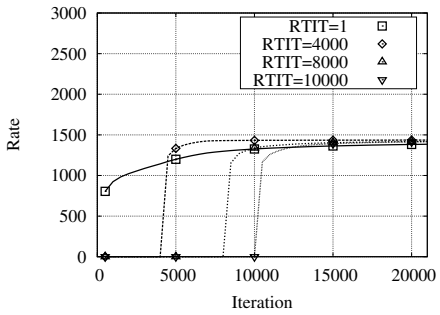


Fig. 8. Time average receiving rates with different RTITs.

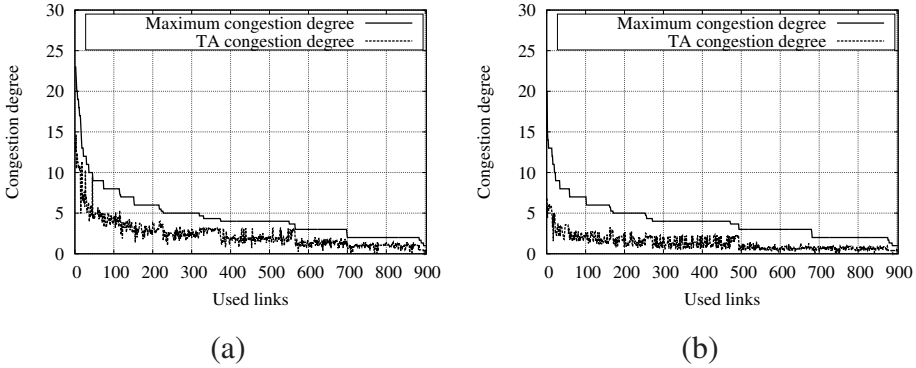


Fig. 9. Congestion degrees of the links used by the session; (a) 100% link utilization case; (b) 80% link utilization case.

The main proofs in the paper are done under the assumptions that the information needed by the algorithm is not lost or delayed. The purpose of having such assumptions is to make the proofs easy. They are not crucial for either practice or theory. In practice, we can make the following natural and minor modification to the algorithm. For each piece of needed information, the algorithm uses the most recent update. In particular, if a control packet that carry some needed information is lost or delayed for more than one time slot, the intended recipient will use the information contained in the last control packet of the same type that it has received. Furthermore, the algorithm can operate asynchronously; i.e., different network entities do not need to synchronize the time slots. The modified algorithm is expected to work well for reasons discussed next.

On the theory side, there are strong reasons to believe that the modified (and asynchronous) algorithm can still achieve optimality and queue boundedness under the following mild conditions: (i) The network delays experienced by the control packets are bounded; and (ii) the number of consecutive control-packet losses is bounded. The boundedness assumptions mean that, although the information used by the algorithm can be outdated and inaccurate for the present time, the

inaccuracy is no more than a bounded amount. For asymptotic results about long-time averages, the bounded inaccuracy usually does not matter. Similar conditions and arguments have been used in [9] [37] to show the optimality of distributed, asynchronous algorithms with delayed or inexact control information.

The control packet loss and delay can be mitigated by implementing the following two mechanisms when possible. First, at each link, the control packets are given higher priority for transmissions than the data packets so that they experience minimal delay and low loss probability. Second, the time slot size is chosen to be greater than the worst-case round-trip propagation time between any neighboring node pair. With these, the control packets will rarely be lost or delayed for more than one time slot.

There can be at least two different implementations of our algorithm. To apply the algorithm to an overlay content distribution network where the nodes are content servers, the algorithm can be implemented in an application-level protocol running over UDP. To apply the algorithm to a router network, new transport-level and network-level protocols are needed to replace aspects of the current TCP/IP and UDP/IP protocols. The source part of algorithm will replace TCP and the link part of the algorithm will be executed by the routers. The latter implementation involves complex changes to the network systems, but is possible today in specialty networks. In particular, some of the current-generation routers/switches employ flexible and open interfaces to enhance programmability (e.g., by virtualization or open-flow architecture). One may use these new network equipments for specialty networks and have a customized implementation of the router algorithms.

REFERENCES

[1] X. Zheng, C. Cho, and Y. Xia, "Optimal peer-to-peer technique for massive content distribution," in *Proc. IEEE INFOCOM*, vol. 8, 2008, pp. 151–155.

[2] ——, "Content distribution by multiple multicast trees and intersession cooperation: optimal algorithms and approximations," in *Proc. IEEE Conference on Decision and Control*, 2009, pp. 5857–5862.

[3] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[4] B. Awerbuch and T. Leighton, "A simple local-control approximation algorithm for multicommodity flow," in *Annual Symposium on Foundations of Computer Science*, vol. 34, 1993, pp. 459–468.

[5] ——, "Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks," in *Proc. ACM Symposium on Theory of Computing*, 1994, pp. 487–496.

[6] X. Lin and N. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *Proc. IEEE Conference on Decision and Control*, 2004, pp. 1484–1489.

[7] M. Neely, E. Modiano, and C. Rohrs, "Dynamic power allocation and routing for time-varying wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 1, pp. 89–103, 2005.

[8] A. Eryilmaz and R. Srikant, "Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1333–1344, 2007.

[9] M. Neely, E. Modiano, and C. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 396–409, 2008.

[10] L. Ying, R. Srikant, and D. Towsley, "Cluster-based back-pressure routing algorithm," in *Proc. IEEE INFOCOM*, 2008, pp. 484–492.

[11] L. Bui, R. Srikant, and A. Stolyar, "Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing," in *Proceedings of IEEE INFOCOM*, 2009, pp. 2936–2940.

[12] L. Ying, S. Shakkottai, and A. Reddy, "On combining shortest-path and back-pressure routing over multihop wireless networks," in *Proc. IEEE INFOCOM*, 2009, pp. 1674–1682.

[13] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: The backpressure collection protocol," in *Proc. ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2010, pp. 279–290.

[14] S. Sarkar and L. Tassiulas, "A framework for routing and congestion control for multicast information flows," *IEEE Transactions on Information Theory*, vol. 48, no. 10, pp. 2690–2708, 2002.

[15] L. Bui, R. Srikant, and A. Stolyar, "Optimal resource allocation for multicast sessions in multi-hop wireless networks," *Philosophical Ttransactions of the Royal Society A*, vol. 366, no. 1872, pp. 2059–2074, 2008.

[16] K. Kar, S. Sarkar, and L. Tassiulas, "Optimization based rate control for multirate multicast sessions," in *Proc. IEEE INFOCOM*, 2001, pp. 123–132.

[17] ——, "A scalable low-overhead rate control algorithm for multirate multicast sessions," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1541–1557, 2002.

[18] S. Sarkar and L. Tassiulas, "Back pressure based multicast scheduling for fair bandwidth allocation," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 1123–1132.

[19] Y. Cui, Y. Xue, and K. Nahrstedt, "Optimal resource allocation in overlay multicast," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 8, pp. 808–823, 2006.

[20] X. Lin and N. Shroff, "Utility maximization for communication networks with multipath routing," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 766–781, 2006.

[21] ——, "The impact of imperfect scheduling on cross-layer congestion control in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 302–315, 2006.

[22] L. Georgiadis, M. Neely, and L. Tassiulas, *Resource allocation and cross layer control in wireless networks*. Now Publishers Inc., 2006.

[23] Y. Cui, B. Li, and K. Nahrstedt, "On achieving optimized capacity utilization in application overlay networks with multiple competing sessions," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, 2004, pp. 160–169.

[24] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. Chou, "Utility maximization in peer-to-peer systems," in *Proc. ACM SIGMETRICS*, 2008, pp. 169–180.

[25] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," in *Proc. ACM SIGMETRICS*, 2008, pp. 313–324.

[26] S. Liu, M. Chiang, S. Sengupta, J. Li, and P. Chou, "P2P streaming capacity for heterogeneous users with degree bounds," in *Proc. Annual Allerton Conference on Communication, Control, and Computing*, 2008, pp. 968–976.

[27] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *The Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[28] W. Wang, M. Palaniswami, and S. Low, "Optimal flow control and routing in multi-path networks," *Performance Evaluation*, vol. 52, no. 2-3, pp. 119–132, 2003.

[29] H. Han, S. Shakkottai, C. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the Internet," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1260–1271, 2006.

[30] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Athena Scientific, 1997.

[31] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing Steiner trees," in *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2003, pp. 266–274.

[32] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, 2000.

[33] R. Srikant, *The mathematics of Internet congestion control*. Birkhauser, 2004.

[34] D. Bertsekas, *Nonlinear programming*, 2nd ed. Athena Scientific, 1999.

[35] M. Neely, "Dynamic power allocation and routing for satellite and wireless networks with time varying channels," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, 2003.

[36] *Rocketfuel: an ISP topology mapping engine*, University of Washington, http://www.cs.washington.edu/research/networking/rocketfuel/.

[37] J. N. Tsitsiklis and D. P. Bertsekas, "Distributed asynchronous optimal routing in data networks," *IEEE Transactions On Automatic Control*, vol. AC-31, pp. 325–332, 1986.

Chunglae Cho is a senior researcher at Electronics and Telecommunications Research Institute, Daejeon, Korea. He received his B.S. and M.S. degrees in computer science from Pusan National University, Korea, in 1994 and 1996, respectively. He received his Ph.D. in computer engineering from the University of Florida in 2011. His research interests are in resource allocation, load balancing, congestion control and optimization in communication networks, peer-to-peer networks, content distribution networks, wireless networks and sensor networks.

Ye Xia is an associate professor at the Computer and Information Science and Engineering department at the University of Florida. He has a PhD degree from the University of California, Berkeley, in 2003, an MS degree in 1995 from Columbia University, and a BA degree in 1993 from Harvard University, all in Electrical Engineering. Between June 1994 and August 1996, he was a member of the technical staff at Bell Laboratories, Lucent Technologies in New Jersey. His main research area is computer networking, including performance evaluation of network protocols and algorithms, resource allocation, wireless network scheduling, network optimization, and load balancing on peer-to-peer networks. He also works on cache organization and performance evaluation for chip multiprocessors. He is interested in applying probabilistic models to the study of computer systems.