

Algorithms and Stability Analysis for Content Distribution over Multiple Multicast Trees

Xiaoying Zheng, Chunglae Cho, and Ye Xia

Abstract

The paper investigates theoretical issues in applying the universal swarming technique to efficient content distribution. In a swarming session, a file is distributed to all the receivers by having all the nodes in the session exchange file chunks. By universal swarming, not only all the nodes in the session, but also some nodes outside the session may participate in the chunk exchange to improve the distribution performance. We present a universal swarming model where the chunks are distributed along different Steiner trees rooted at the source and covering all the receivers. We assume chunks arrive dynamically at the sources and focus on finding stable universal swarming algorithms. To achieve the throughput region, universal swarming usually involves a tree-selection subproblem of finding a min-cost Steiner tree, which is NP-hard. We propose a universal swarming scheme that employs an approximate tree-selection algorithm. We show that it achieves network stability for a reduced throughput region, where the reduction ratio is no more than the approximation ratio of the tree-selection algorithm. We propose a second universal swarming scheme that employs a randomized tree-selection algorithm. It achieves the throughput region, but with a weaker stability result. The proposed schemes and their variants are expected to be useful for infrastructure-based content distribution networks with massive content and relatively stable network environment.

Index Terms

Communication Networks, Multicast, Stability, Queueing, Randomized Algorithm, Content Distribution

I. INTRODUCTION

The Internet is being used to transfer content on a more and more massive scale. A recent innovation for efficient content distribution is a technique known as *swarming*. In a swarming

X. Zheng is with Shanghai Advanced Research Institute, Chinese Academy of Sciences and Shanghai Research Center for Wireless Communications, China. E-mail: zhengxy@sari.ac.cn.

C. Cho and Y. Xia are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611. E-mail: {ccho, yx1}@cise.ufl.edu.

session, the file to be distributed is broken into many chunks at the original source, which are then spread out across the receivers. Subsequently, the receivers exchange the chunks with each other to speed up the distribution process. Many different ways of swarming have been proposed, such as FastReplica [1], Bullet [2], Chunkcast [3], BitTorrent [4], and CoBlitz [5].

The swarming technique was originally introduced by the end-user communities for peer-to-peer (P2P) file sharing. The subject of this paper is how to apply swarming to infrastructure-based content distribution, where files are to be distributed among content servers in a content delivery network. The content servers are usually connected with leased high-speed links and, as a result, the bandwidth bottleneck may no longer be at the access links. It has been shown that content delivery traffic has made capacity shortage in the backbone networks a genuine possibility [6]. The size of such a content distribution network is usually small, consisting of up to hundreds of network nodes and up to thousands of servers. Unlike the dynamic end-user file-sharing situations, infrastructure networks and content servers are usually centrally managed, generally well-behaved and relatively static (however, the traffic can still be dynamic). In this setting, it is beneficial to view swarming as distribution over multiple multicast trees, each spanning the content servers. This view allows us to pose the question of how to optimally distribute the content (see [7]). Furthermore, it is often easier to first develop sophisticated algorithms under this tree-based view, and then, adapt them to practical situations where the tree-based view is only partially adequate. Hence, in this paper, swarming is synonymous to distribution over multiple multicast trees.

This paper concerns a class of improved swarming techniques, known as *universal swarming*. We associate with each file to be distributed a *session*, which consists of the source of a file and the receivers who are interested in downloading the file. In traditional swarming, chunk exchange is restricted to the nodes of the session. However, in *universal swarming*, multiple sessions are combined into a single “super session” on a shared overlay network. Universal swarming takes advantages of the heterogenous resource capacities of different sessions, such as the source upload bandwidth, receiver download bandwidth, or aggregate upload bandwidth, and allows the sessions to share each other’s resources. The result is that the distribution efficiency of the resource-poor sessions can improve greatly with negligible impact on the resource-rich sessions (see [8]).

In universal swarming, if we focus on a particular file, not only the source and all the receivers participate in the chunk exchange process, some other nodes who are not interested in the file may also participate. We call the latter out-of-session nodes. To illustrate the essence of universal

swarming, as well as the main issues, consider the toy example in Fig. 1. The numbers associated with the links are their capacities. Let us consider a particular file for which the source is node 1 and the receivers are nodes 2 and 3. Node 4 is out of the session. Let us focus on a fixed chunk and consider how it can be distributed to the receivers. With some thoughts, it can be seen that the chunk propagates on a tree rooted at the source and covering both receivers. All possible distribution trees are shown in Fig. 2. We notice that a distribution tree may or may not include the out-of-session node, 4. Thus, a distribution tree in general is a *Steiner tree* rooted at the source covering all the receivers, where the out-of-session nodes (e.g., node 4) are the Steiner nodes.

With this model of multi-tree multicast, one of the main questions is how to assign the chunks to different distribution trees so as to optimize certain performance objective, such as maximizing the sum of the utility functions of the sessions, or minimizing the distribution time of the slowest session. This is a *rate allocation problem* on the multicast trees. One such question was addressed in [7] in the context of non-universal swarming, where each session’s multicast trees are spanning trees instead of Steiner trees. For universal swarming, the question was addressed in [8].

This paper addresses the *stability problem*. The main question is: Given a set of data rates from the sources (which are possibly the solutions to the aforementioned rate allocation problem), how do we get a universal swarming algorithm so that the network queues will be stable? For the example in Fig. 1, a source rate of 2 is the largest distribution rate that can be supported by the network if everything is deterministic. To achieve stability under random arrivals, it usually requires that the data arrival rate is strictly less than 2 (for a justification, consider a single-queue system). Hence, when the file chunks arrive at (or generated by) the source node 1 at a mean rate $2 - 2\epsilon$, where $0 < \epsilon < 1$, we can place chunks on the first and the second tree in Fig. 2 at a mean rate $1 - \epsilon$ each. For this example, the solution actually stabilizes the network. But, this conclusion requires technical conditions and is not generally true for more complicated situations.

In this paper, we develop a universal swarming scheme that employs an approximation algorithm to the tree selection (*a.k.a.* scheduling) problem, which achieves a rate region¹ equal to the throughput region reduced by a constant factor γ , $\gamma \geq 1$. We show that γ is no more than the approximation ratio of the tree scheduling algorithm. The scheme requires network signaling and source traffic regulation. We propose a second universal swarming scheme that utilizes a

¹Subsequently, when we say an algorithm achieves or stabilizes a region, we mean the interior of the region.

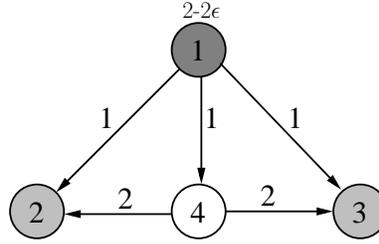


Fig. 1. Node 1 sends the file to nodes 2 and 3.

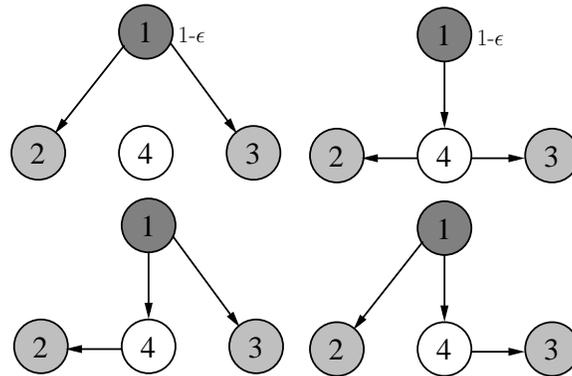


Fig. 2. All possible distribution trees for the example in Fig. 1.

randomized tree selection algorithm, which achieves the entire throughput region, but with a weaker stability property.

The difference between our problem and the wireless scheduling problems is substantial. Most previous papers that consider multi-hop traffic are either in the unicast setting or in a multicast setting with a few fixed multicast trees per multicast session. We must consider multi-hop, multicast communications, and, to have the largest possible throughput region, we allow each multicast session to use *any* multicast trees for the session. The combination of these three features makes our problem both unique and very hard. One of the main challenges is that there is no obvious way to specify the packet forwarding behavior without per-tree queueing (i.e., having a separate queue for each multicast tree) whereas, on the other hand, per-tree queueing is impractical due to the exceedingly large number of trees. Our solution to resolve this difficulty, on the algorithm side, is to use the techniques of signaling and virtual queueing. The techniques allow us to have very small numbers of queues and at the same time show stability guarantee.

Having avoided per-tree queueing, however, the performance analysis is still difficult. In

particular, there is no easy way to prove the stability of the real queues. Our analytical approach is to first prove the stability of the virtual queues. The proofs in this step are relatively conventional, using the techniques of Lyapunov drift analysis. The second step is to make connections between the virtual queues and the real queues, and use the stability results for the virtual queues to prove that the real queues are also stable.

The tree selection subproblem is inherent to the problem formulated in the paper and it remains difficult. As a remedy, our algorithms can work with low-cost trees as opposed to the min-cost trees; finding low-cost trees can be much easier. The stability results are applicable to classes of algorithms by allowing different tree selection sub-algorithms. Hence, the research for finding simpler, more practical algorithms can continue.

The rest of the paper is organized as follows. The models and the problem description are given in Section II. The first universal swarming scheme and the analysis are presented in Section III. The second universal swarming scheme and the analysis are presented in Section IV. In Section VI, we discuss additional related work. The conclusion is in Section VII.

II. PROBLEM DESCRIPTION

We consider a time-slotted system where each time slot has a duration of one time unit. Let the network be represented by a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. For each link $e \in E$, let c_e denote its capacity (e.g., in the number of file chunks it can transmit per time slot), where $c_e > 0$. We assume that each session, which distributes a distinct file, has one source, and hence, there is a one-to-one mapping between a session and a source.² Let S denote the set of sources (sessions). For each $s \in S$, let $V_s \subseteq V$ be the set of receivers associated with the source (session) s .

For each source $s \in S$, suppose constant-sized data packets (i.e., file chunks)³ arrive at the

²The case of multiple sources makes the most sense when the sources each possess a copy of a common file, which will be distributed to a common group of receivers. We can extend the network graph by adding a virtual node and virtual edges. Each virtual edge connects the virtual node to one of the sources, and it is given an infinite capacity. In the expanded graph, the virtual node will be considered the source of the multicast session. In the dynamic case where chunks of a common file arrive at the sources following stochastic processes, it is difficult to parsimoniously specify the relationship among the file chunks to different sources. Without that information, we will identify each source as a separate session (which has the same set of receivers).

³Since our algorithms can be deployed more easily at the application level, in this paper, we use the term *data packet* (or *real packet*) to mean an application-level data unit and we regard it as being synonymous to a chunk. When the context is clear, we usually call a data/real packet a packet. A data packet can be fairly large, such as 256 KB, and may need to be carried by multiple network-level packets. Our algorithms also use signaling/control packets, which are much smaller, e.g., under 400 bytes.

source according to a random process, which will be distributed over the network to all the receivers, V_s . The motivation for using a source model with dynamic arrivals is to account for the end-system bottleneck and timing variations in reading and transmitting locally stored data. In some cases, the content may not be a static file or stored locally. The model is general enough to cover realtime content, streaming video with time-varying rate, or non-locally stored static data. Even if the entire file is static and stored at the source, this source model can still be useful. For instance, the data packets can be injected into the source node at a constant rate, which corresponds to a deterministic arrival process with a constant arrival rate. Let $A_s(k)$ be the number of packet arrivals on time slot k . Let us make the following assumption on the arrival processes $\{A(k)\}$ throughout the paper, unless mentioned otherwise. Additional assumptions may be added as needed.

AS 1. For each source $s \in S$, $E[A_s(k)|x(k)] = \lambda_s$, and $E[(A_s(k))^2|x(k)] < K_1$ for some $0 < K_1 < \infty$, for all time k , where $x(k)$ represents the system state at time k .

The following are some remarks about Assumption AS 1.

- The system state, $x(k)$, will depend on the specific settings of the two algorithms considered in this paper and will become clear later. It usually includes all the queue sizes at time k , and possibly some additional auxiliary variables. If the arrival process is independent of the past for each source s , assumption AS 1 can be stated without conditioning on $x(k)$. However, some non-IID arrival processes can lead to dependence between the current arrivals and the current queue sizes, in which case the statement with conditioning is more general than one without conditioning.
- If the arrival process is independent of the past for each source s , assumption AS 1 can be stated without the conditioning.
- $(\lambda_s)_{s \in S}$ could be the solution of a rate allocation problem (see the discussion about the rate allocation and stability problems in Section I).

In this paper, we will present stable universal swarming algorithms to distribute the packets to all the receivers. For each $s \in S$, the packets will be transmitted along various multicast distribution trees rooted at s to the receivers in V_s . A multicast tree in the multicast case corresponds to a path between a sender and a receiver in the unicast case. Hence, using multiple multicast trees for a multicast session is analogous to data delivery using multiple paths between a sender and a receiver in the unicast case.

We will take Neely's definition of stability ([9]; [10], chapter 2) unless mentioned otherwise.

For a single-queue process $\{q(k)\}$, let us define the overflow function:

$$g(M) = \limsup_{K \rightarrow \infty} \frac{1}{K} \sum_{k=1}^K P\{q(k) > M\}. \quad (1)$$

Roughly speaking, the overflow function $g(M)$ in (1) defines the long-time average fraction of time when the queue size $q(k)$ is more than a chosen threshold M . In the stationary and ergodic case, it coincides with the stationary (and limiting) probability that the queue size exceeds M .

Definition 1. *The single-queue process $\{q(k)\}$ is stable if $g(M) \rightarrow 0$ as $M \rightarrow \infty$. A network of queues is stable if every queue is stable.*

With this definition of network stability, a sufficient condition for network stability is: Some Lyapunov function of the queues has a negative drift when any of the queues becomes large enough [9] [10] [11] [12]. If with additional assumptions, the network queues form an ergodic Markov chain, the same drift condition implies the chain is positive recurrent, or equivalently, has a stationary distribution.

A. Throughput Region

For each source $s \in S$, let the set of candidate distribution trees be denoted by T_s . Throughout this paper, T_s contains all possible distribution trees rooted at the source s unless specified otherwise. Let $T = \cup_{s \in S} T_s$. The trees can be enumerated in an arbitrary order as $t_1, t_2, \dots, t_{|T|}$, where $|\cdot|$ denote the cardinality of a set. Although $|T|$ is finite, it might be very large.

The *throughput region* is defined as

$$\Lambda = \{\lambda \geq 0 : \exists \alpha \geq 0 \text{ such that } \sum_{t \in T_s} \alpha_t = 1, \forall s \in S \text{ and } \sum_{s \in S} \sum_{\{t \in T_s | e \in t\}} \alpha_t \lambda_s \leq c_e, \forall e \in E\}. \quad (2)$$

Here, α represents how the traffic from the sources is split among the distribution trees. The definition of Λ says that a source rate vector λ is in Λ if there exists a set of tree rates for each multicast session such that the resulting total link data rate is no more than the link capacity for any link. Obviously, Λ contains the stability region, i.e., all λ that can be stabilized by some algorithms. This is so because, for any non-negative mean rate vector $\lambda \notin \Lambda$, no matter how the traffic is split among the distribution trees, there exists a link e such that the total arrival rate to e is strictly greater than its service rate. Furthermore, this definition of the throughput region allows the bandwidth bottleneck to be anywhere in the network, at the access links or at the core.

We also define a γ -reduced throughput region as $\frac{1}{\gamma}\Lambda$, where $\gamma \geq 1$. By saying that the arrival rate vector λ is *strictly inside the region* $\frac{1}{\gamma}\Lambda$, we mean that there exist some $\epsilon_0 > 0$ and a vector $\alpha \geq 0$ such that $\sum_{t \in T_s} \alpha_t = 1, \forall s \in S$ and $\sum_{s \in S} \sum_{\{t \in T_s | e \in t\}} \alpha_t \lambda_s \leq \frac{1}{\gamma} c_e - \epsilon_0, \forall e \in E$. This is equivalent to

$$c_e \geq \gamma(\epsilon_0 + \sum_{s \in S} \sum_{\{t \in T_s | e \in t\}} \alpha_t \lambda_s), \quad \forall e \in E. \quad (3)$$

Note that the region of rate vectors that are strictly inside the region $\frac{1}{\gamma}\Lambda$ contains the interior of $\frac{1}{\gamma}\Lambda$. In Section III, we will show that the interior of Λ is stabilizable. That is, for any rate vector λ strictly inside the region Λ , there exists a scheduling algorithm such that the queues in the network are stable under the algorithm.

B. The Class of Algorithms: Time Sharing of Trees

Each source has at least two possible approaches to use the multicast trees. In one approach, the traffic from each source s may be split according to some weights $(\alpha_t)_{t \in T_s}$ and transmitted simultaneously over the trees on every time slot. Alternatively, the distribution can be done by time-sharing of the trees. The algorithms in this paper follow the time-sharing approach. On each time slot k , the source s selects one distribution tree from the set T_s , denoted by $t_s(k)$, according to some tree-scheduling (tree-selection) scheme, and transmits packets only over this tree on time slot k . The time-sharing approach can emulate the first approach in the sense that, when done properly, the fraction of time each distribution tree is used over a long period of time can approximate any weight vector $(\alpha_t)_{t \in T_s}$.

In addition to selecting the distribution tree $t_s(k)$ at each time slot, an algorithm also needs to decide how many packets are released to the tree. We will present two algorithms in the following sections. The key question is what portion of the rate region is stabilizable by each algorithm.

III. SIGNALING, SOURCE TRAFFIC REGULATION AND γ -APPROXIMATION MIN-COST TREE SCHEDULING

A. Signaling Approach

Stability analysis of a multi-hop network is often difficult because the packets travel through the network hop-by-hop, instead of being imposed directly to all links that they will traverse. As a result, the arrival process to each internal link can be difficult to describe. The frequently-used technique of network signaling can be helpful. In our case, on each time slot k , each source

s sends one signaling packet to each node on the currently selected tree $t_s(k)$. A signaling packet is one of the two types of control packets in the paper. It has two main functions. The first is to set up the multicast tree $t_s(k)$. A signaling packet contains a list of link IDs, which describe to the receiving node which of its outgoing links are part of the multicast tree. The second function is to inform the receiving node the intended source transmission rate on time slot k , i.e., the number of packets to be transmitted on time slot k . To make the proofs for the main results easier, we make the following assumptions about all control packets, including the forward signaling packets and the feedback packets: The control packets are never lost and they arrive at their intended destinations within the same time slot on which they are first transmitted. Note that these assumptions are not crucial for either the theory or practice⁴. We do not make any assumptions on the data packets.

We will see that the rate information contained in a signaling packet is a very tight upper bound on the number of real packets released by the source on that time slot. To mark the slight discrepancy, we use the term *virtual packets* and call the rate contained in the signaling packet the *rate of virtual packets* or the *virtual source rate*. Consider a particular time slot k and a particular internal link e on the selected distribution tree. The real packets issued by the source on time slot k will in general be delayed or buffered at upstream hops and will not arrive at link e until later. However, via signaling, link e knows how many virtual packets are injected by the source and arrive at link e on time slot k . The cumulative number of arrived real packets at link e must be no more than the cumulative number of arrived virtual packets (via signaling packets).

One question is how many real packets are to be released to the network on a time slot. One possibility is that each source s releases all the packets that arrive during time slot k , i.e., $A_s(k)$.

⁴ In actual operation, the two algorithms in the paper need not enforce these assumptions. The control packets can be lost or delayed. With straightforward minor modifications, such as using old information or postponing the algorithm execution until new information is available, the algorithms can cope with these conditions and are expected to be robust. They can achieve stability and throughput optimality as indicated by the theory. For better performance with respect to other metrics (e.g., data queue size, data delay, convergence speed), the algorithms can implement the following policy: The control packets are given higher priority at all the nodes than the data packets so that they experience minimal delay; the time slot size is chosen to be greater than the worst-case round-trip propagation time. With such a policy, the assumptions are expected to be satisfied most of the time. In our simulation experiments (see Section V), the control packet delays are included. The algorithms converge fast and to the right values. On the theory side, there are good reasons to believe that the stability results in this paper still hold under the significantly relaxed assumptions: The network delays of the control packets are bounded and the number of consecutive control-packet losses is bounded. The boundedness assumptions make the conditions of Corollary 1 in [11] satisfied. The stability results would follow.

However, the uncontrolled randomness of $A_s(k)$ causes difficulty in the stability analysis, as we will see later. In our algorithm, each source s sets the number of real packets to be released at the constant value $\lambda_s + \epsilon_1$ on every time slot k , if that number of real packets is available. Every signaling packet from source s contains the constant virtual packet rate $\lambda_s + \epsilon_1$. Here, ϵ_1 is a sufficiently small constant such that $0 < |S|\epsilon_1 < \epsilon_0$. This guarantees the stability of the source regulators, as we will see.

In the algorithm, each link e updates a virtual queue, denoted by $q_e(k)$.

$$q_e(k+1) = [q_e(k) + \sum_{s \in S: e \in t_s(k)} (\lambda_s + \epsilon_1) - c_e]_+. \quad (4)$$

$[\cdot]_+$ is the projection operation onto the non-negative domain. Note that the second term on the right hand side of (4) is the aggregate virtual data arrival rate from all the trees containing link e , which means the link capacities are shared by different trees. Tree scheduling is based on the virtual queues instead of the real queues.

B. Source Traffic Regulation

A regulator is placed at each source s to ensure that on each time slot, source s transmits no more than $\lambda_s + \epsilon_1$ real packets. A regulator is a traffic shaping device. All the packets arriving at source s first enter a regulator queue. They will be released to the network later in a controlled fashion. On each time slot k , let $D_s(k)$ denote the number of real packets released from the regulator to the distribution tree $t_s(k)$, and let $p_s(k)$ be the regulator queue size at source s . The evolution of the regulator queue is given by

$$p_s(k+1) = p_s(k) + A_s(k) - D_s(k), \quad (5)$$

where

$$D_s(k) = \begin{cases} \lambda_s + \epsilon_1 & \text{if } p_s(k) \geq \lambda_s + \epsilon_1; \\ p_s(k) & \text{otherwise.} \end{cases} \quad (6)$$

Expressions (6) ensures that at most $\lambda_s + \epsilon_1$ real packets are released on each time slot. Since this departure rate is higher than the mean packet arrival rate, stability of the regulator is guaranteed⁵. We will provide more details in the stability analysis. Note that the traffic regulators are required only at the sources and they can be implemented at the end-systems, i.e., the content servers.

⁵ A packet may experience some delay at the regulator before it is released to the network. The performance analysis shows that this delay is bounded (in expectation) since all regulator queues are bounded. Our simulation results show that the delay and queue sizes can be made small even for very small ϵ_1 .

C. γ -Approximation Min-Cost Tree Scheduling

We can interpret the virtual queue size q_e as the cost of link e . Then, the cost of a tree t is $\sum_{e \in t} q_e$. We propose the γ -approximation min-cost tree scheduling scheme: On each time slot k and for each source s , the selected tree $t_s(k)$ satisfies

$$\sum_{e \in t_s(k)} q_e(k) \leq \gamma \min_{t \in T_s} \sum_{e \in t} q_e(k), \quad (7)$$

where $\gamma \geq 1$. If there are multiple trees satisfying (7), the tie is broken arbitrarily.

The rationale for this tree-scheduling scheme is straightforward. When $\gamma = 1$, the tree-scheduling scheme solves the min-cost Steiner tree problem, which is NP-hard. But, the min-cost Steiner tree problem has approximation solutions, which we can use. In [13], a family of approximation algorithms for the directed Steiner tree problem is proposed, which achieves an $O(\log^2 N)$ approximation ratio in quasi-polynomial time, where N is the number of receivers. It will be proven in the following stability analysis that, if we are able to find the minimum-cost Steiner tree on each time slot, we can stabilize the network for the interior of the entire throughput region, Λ ; if we adopt the γ -approximated min-cost tree scheduling, we can stabilize the network for the interior of $\frac{1}{\gamma}\Lambda$.

The link costs (i.e., the virtual queue sizes) are carried to the multicast sources by the second type of control packets - the feedback packets. On each time slot, a network node sends to each source one feedback packet, which contains the costs of the node's outgoing links.

D. Stability Analysis

The stability analysis is based on the drift analysis of Lyapunov functions.

1) *Stability of the Regulators:* Define a Lyapunov function of the regulator queues p as

$$L_1(p) = \sum_{s \in S} p_s^2. \quad (8)$$

Lemma 1. *There exists some positive constant $0 < M_1 < \infty$ such that for every time slot k and the regulator backlog vector $p(k)$, the Lyapunov drift satisfies*

$$E[L_1(p(k+1)) - L_1(p(k)) | p(k)] \leq -\epsilon_1 \sum_{s \in S} p_s(k), \quad (9)$$

if $\sum_{s \in S} p_s(k) \geq \frac{M_1}{\epsilon_1}$.

Proof: This is because the mean arrival rate is strictly less than the mean service rate provided the regulator has sufficient packets. The proof is standard and we omit the details.

2) *Stability of the Virtual Queues:* Define a Lyapunov function of the virtual queue backlog vector q as

$$L_2(q) = \sum_{e \in E} q_e^2. \quad (10)$$

Let $t(k) = (t_s(k))_{s \in S}$ be the vector of the chosen distribution trees at time k . We allow randomness in determining $t(k)$. For instance, when there are multiple trees satisfying (7), the tie can be broken randomly.

Lemma 2. *If the mean arrival rate vector λ is strictly inside the region $\frac{1}{\gamma}\Lambda$, then, there exist some positive constants $0 < M_2 < \infty$ and ϵ for all sample paths of $\{t(k)\}_k$ such that, for every time slot k and virtual queue backlog vector $q(k)$, the Lyapunov drift satisfies*

$$L_2(q(k+1)) - L_2(q(k)) \leq M_2 - 2\epsilon \sum_{e \in E} q_e(k), \quad (11)$$

where $\epsilon = \gamma\epsilon_0 - |S|\epsilon_1 > 0$.

Proof: Under any sample path of the process $\{t(k)\}_k$, (11) holds due to the tree-selection algorithm (7) and the definition of the γ -reduced throughput region (3). The detailed proof is omitted for brevity.

Hence, when $\sum_{e \in E} q_e(k) \geq \frac{M_2}{\epsilon}$, the Lyapunov function has a negative drift under all sample paths of $\{t(k)\}_k$.

$$L_2(q(k+1)) - L_2(q(k)) \leq -\epsilon \sum_{e \in E} q_e(k). \quad (12)$$

Corollary 3. *For each link e , there exists a sufficiently large constant $M_e < \infty$ such that $q_e(k) \leq M_e$.*

Proof: From Lemma 2, under any sample path of $\{t(k)\}_k$, $q_e(k)$ is uniformly bounded from above. Hence, there exists a sufficiently large constant $M_e < \infty$ such that $q_e(k) \leq M_e$.

Remark: The chosen deterministic release rates of the virtual packets guarantee that the virtual queues are bounded. This is an important fact for proving the stability of the real queues. If the sources signal the actual numbers of real packet arrivals on each time slot, which are random, the virtual queues may be stable but are not guaranteed to be bounded.

3) *Stability of the Real Queues:* For convenience, let us assume each real packet remembers its distribution tree. This way, the nodes on the tree know when to duplicate the packet. Moreover, each packet at any link also has an unambiguous *hop count*, which is the hop count on its tree

path from the source to the current link. With this setup, we can assume to use the following queueing discipline for the real queues.

AS 2. *At each link e , a packet with a smaller hop count has priority over any packet with a larger hop count.*

First, we will show some properties of the real packet arrival rates to the intermediate links. Define an indicator function $I(e, t)$, where e is a link and t is a tree.

$$I(e, t) = \begin{cases} 1 & \text{if } e \in t; \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 4. *For any link $e \in E$, there exists a constant $0 < M_e < \infty$ such that for any k_0 and k with $k_0 \leq k$,*

$$\sum_{u=k_0}^k \sum_{s \in S} D_s(u) I(e, t_s(u)) \leq (k - k_0 + 1)c_e + M_e. \quad (13)$$

Proof: According to (4),

$$q_e(k+1) \geq q_e(k) + \sum_{s \in S} (\lambda_s + \epsilon_1) I(e, t_s(k)) - c_e, \forall e \in E. \quad (14)$$

By summing (14) over time slots, the proof is straightforward and is omitted.

Let $Q_e(k)$ denote the real queue backlog of link e at time slot k . We can show by induction that under the prioritized queueing strategy in AS 2, the real queue backlogs are bounded. The proof is adapted from [14].

Theorem 5. *With the additional assumption AS 2, if the mean arrival rate vector λ is strictly inside the region $\frac{1}{\gamma}\Lambda$, the real queue backlogs are bounded. I.e., there exists some constant $0 < M' < \infty$ such that*

$$Q_e(k) \leq M', \quad \forall k, \forall e \in E. \quad (15)$$

Proof: See Appendix A.

Theorem 6. *With the additional assumption AS 2, if the mean arrival rate vector λ is strictly inside the region $\frac{1}{\gamma}\Lambda$, $\gamma \geq 1$, the γ -approximation min-cost tree scheduling scheme stabilizes the network.*

Proof: From Lemma 1, Lemma 2 (or Corollary 3) and Theorem 5, the regulator queues have negative drifts, the virtual queues and the real queues in the network are all bounded.

The reduction factor $1/\gamma$ in Theorem 6 is a lower bound for the worst case. In practice, the actual reduction factor for a given network and a specific tree-scheduling scheme may be much larger. For instance, we use a sub-optimal tree-scheduling scheme and some ISP networks for the simulation results in Section V, and the algorithm can achieve nearly the full throughput region.

IV. RANDOMIZED TREE SCHEDULING

Theorem 6 implies that the interior of the throughput region Λ can be stabilized, provided one can solve the hard min-cost Steiner tree problem. If approximation algorithms are used for the Steiner tree problem, Theorem 6 says that a reduced rate region is stabilizable. In this section, we will continue to cope with the hard Steiner tree problem. Instead of approximation algorithms, we will consider an algorithm that randomly samples the trees at each time slot. Selecting trees by random sampling is attractive in practice since the algorithms for doing this tend to be simple and fast. Some practical systems such as BitTorrent [4] already use variants of random sampling.

Our main concern is whether the tree-sampling approach has any performance guarantee with respect to stability. We conjecture it does. We will show important steps that may eventually lead to the conclusion that, in contrast to the case with approximation algorithms, the entire interior of Λ is stabilizable. The development and analysis of the algorithm are in part based on [15].

A. Signaling

In this algorithm, the sources still signal the links about the incoming traffic, but they are not regulated. Specifically, the number of virtual packets signaled by source s on every time slot k is $A_s(k)$ instead of $\lambda_s + \epsilon_1$. For each $e \in E$, the evolution of the virtual queue, $q_e(k)$, is

$$q_e(k+1) = [q_e(k) + \sum_{s \in S: e \in t_s(k)} A_s(k) - (c_e - \epsilon_2)]_+, \quad (16)$$

where $0 < \epsilon_2 < \epsilon_0$. From (16), the virtual queue is serviced at less than the full service capacity. We will see the reason in the stability analysis (see Corollary 14 and the remark after it).

B. Randomized Tree Scheduling

Let $\tau_s(q)$ denote the min-cost tree for source s with respect to the link cost vector q . We have,

$$\sum_{e \in \tau_s(q)} q_e = \min_{t \in T_s} \sum_{e \in t} q_e. \quad (17)$$

If multiple min-cost trees exist, an arbitrary one is chosen.

The algorithm has two stages: *pick* and *compare*. In the pick stage, each source uses some randomized algorithm to pick a tree, with the requirement that there is a positive probability to pick a min-cost tree. More specifically, let $\hat{t}(k) = (\hat{t}_s(k))_{s \in S}$ be the trees picked by the randomized algorithm on time slot k . The following condition is satisfied for some $\delta > 0$,

$$P\left\{ \sum_{e \in \hat{t}_s(k)} q_e(k) = \sum_{e \in \tau_s(q(k))} q_e(k), \forall s \in S \right\} \geq \delta. \quad (18)$$

In the compare stage, the cost of the tree just picked is compared with the cost of the selected tree on the previous time slot, with respect to the current link cost vector. The picked tree is selected only if it has a lower cost. This ensures that the tree that ends up being selected is better than the previously selected tree. Recall that $t_s(k)$ is the scheduled tree at time k . The compare stage yields a selected tree that satisfies the following: For any source $s \in S$,

$$t_s(k) = \begin{cases} \hat{t}_s(k) & \text{if } \sum_{e \in \hat{t}_s(k)} q_e(k) \leq \sum_{e \in t_s(k-1)} q_e(k); \\ t_s(k-1) & \text{otherwise.} \end{cases} \quad (19)$$

There are many possible randomized selection algorithms that satisfy (18) and (19). For instance, one algorithm might be to modify the current tree by randomly adding or deleting edges until a new multicast tree is found. The selection of the edges can be biased toward lower-cost ones for addition and higher-cost ones for deletion. In this paper, we will not dwell on finding specific algorithms but will focus on the stability issue of the whole algorithm class.

C. Stability Analysis

We will show that, if the mean arrival rate vector λ is strictly inside the throughput region Λ , the randomized tree-scheduling scheme is able to stabilize all the virtual queues. With additional assumptions, the cumulative arrival of the real packets by any time slot is strictly less than the accumulation of the link service rate for every link.

1) *Stability of the Virtual Queues:* The virtual queue sizes $q(k)$ are considered as the link costs. Let $t(k)$ be the vector of chosen trees. Define a Lyapunov function of $x = (q, t)$:

$$L(x) = L_1(x) + L_2(x),$$

where

$$L_1(x) = \sum_{e \in E} q_e^2, \quad L_2(x) = \left(\sum_{s \in S} \lambda_s \left(\sum_{e \in t_s} q_e - \sum_{e \in \tau_s(q)} q_e \right) \right)^2.$$

The proofs for the following three main lemmas parallel the development in [15], although the details are different and technical. We omit them for brevity.

Lemma 7. *If the mean arrival rate vector λ is strictly inside the throughput region Λ , there exist some positive constants M_1 and ϵ such that*

$$E[L_1(x(k+1)) - L_1(x(k)) | x(k)] \leq M_1 + 2\sqrt{L_2(x(k))} - 2\epsilon \sum_{e \in E} q_e(k). \quad (20)$$

Lemma 8. *If the arrival rate vector λ is strictly inside the throughput region Λ , there exist some positive constants M_2 and M_3 such that*

$$E[L_2(x(k+1)) - L_2(x(k)) | x(k)] \leq M_2 + M_3\sqrt{L_2(x(k))} - \delta L_2(x(k)). \quad (21)$$

Lemma 9. *If the arrival rate vector λ is strictly inside the throughput region Λ , there exist some positive constants $M < \infty$ and ϵ such that, if $L(x(k)) \geq M$,*

$$E[L(x(k+1)) - L(x(k)) | x(k)] \leq -\epsilon\sqrt{L_1(x(k))}. \quad (22)$$

Theorem 10. *If the mean arrival rate vector λ is strictly inside the throughput region Λ , the randomized tree scheduling scheme stabilizes the virtual queues.*

Proof: This is a corollary from Lemma 9.

2) *Stability of the Real Queues:* We have partial results about the stability of the real queues under additional conditions. We assume the following in this subsection.

AS 3. *The processes $\{A_s(k)\}_k$ for different s are independent from each other. For each $s \in S$, $\{A_s(k)\}_k$ is IID. At every time k , there is a nonzero probability that no packet arrives at the sources, i.e., $P\{A_s(k) = 0, \forall s \in S\} > 0$.*

We will show that for any link e , its average traffic intensity (load), ρ_e , satisfies $\rho_e < 1$, where ρ_e is the ratio of the average packet arrival rate and the link rate. First, stronger stability conclusions can be said about the virtual queues.

Theorem 11. *Suppose the mean arrival rate vector λ is strictly inside the throughput region Λ , and assumptions AS 1 and AS 3 hold.*

- *The process $\{q(k), t(k)\}_{k=0}^{\infty}$ is an aperiodic and irreducible Markov chain with a stationary distribution. Moreover, let \hat{q} be the virtual queues under the stationary distribution. Then, $E[\hat{q}_e] < \infty$.*
- *The strong law of large numbers holds: For each initial condition, and for all $e \in E$,*

$$\lim_{k \rightarrow \infty} \frac{\sum_{u=0}^k q_e(u)}{k+1} = E[\hat{q}_e], \text{ almost surely.} \quad (23)$$

- *The mean ergodic theorem holds: For each initial condition, and for all $e \in E$,*

$$\lim_{k \rightarrow \infty} E[q_e(k)] = E[\hat{q}_e]. \quad (24)$$

Proof: See Appendix A.

Theorem 12. *For any link $e \in E$,*

$$\limsup_{k \rightarrow \infty} \frac{1}{k+1} \sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u)) \leq c_e - \epsilon_2, \quad (25)$$

$$\limsup_{k \rightarrow \infty} E\left[\frac{1}{k+1} \sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u))\right] \leq c_e - \epsilon_2. \quad (26)$$

Proof: See Appendix A.

Recall that $Q_e(k)$ denotes the real queue backlog of link e at time slot k . Next, we show that the process $\{Q_e(k)\}$ is *rate stable* for all links, where the definition of rate stability is given as in [12].

Corollary 13. *Suppose the mean arrival rate vector λ is strictly inside the throughput region Λ , and assumptions AS 1 and AS 3 hold. For any link $e \in E$, the process $\{Q_e(k)\}$ is rate stable, i.e.,*

$$\lim_{k \rightarrow \infty} \frac{Q_e(k)}{k} = 0 \quad \text{with probability 1.}$$

Proof: See Appendix A.

Rate stability implies that the long-term average rates of arrivals and departures are identical for each queue, and is weaker than the stability definition of the queue backlog being bounded.

Corollary 14. *For any link $e \in E$, the average traffic intensity (or load) $\rho_e < 1$, where ρ_e is defined as*

$$\rho_e = \limsup_{k \rightarrow \infty} \frac{\sum_{u=0}^k a_e(u)}{(k+1)c_e},$$

where $a_e(u)$ is the number of real packets arriving at link e at time u .

Proof: For any time slot k , the number of cumulative arrivals at the real queue is no more than the number of cumulative arrivals at the virtual queue, i.e.,

$$\sum_{u=0}^k a_e(u) \leq \sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u)).$$

Then, $\rho_e < 1$ follows from Theorem 12.

Remark: The service rate of the virtual queue of link e , which is $c_e - \epsilon_2$, guarantees $\rho_e < 1$.

Under the randomized tree scheduling scheme, the virtual queues are stable, the real queue processes $\{Q_e(k)\}$ are rate stable, and the real traffic intensity satisfies $\rho_e < 1$ for every link e . But, we do not know whether the real queues are stable in the sense of Definition 1. We expect that in practice, they are almost always stable. We suspect that under more assumptions on the traffic arrival processes and the queueing discipline, the real queues can be proven to be stable.

V. SIMULATION RESULTS AND EVALUATIONS

In this section, we present illustrative examples from simulation experiments that support the stability analysis of the algorithms. We will also evaluate the control overhead of the algorithms.

Since the algorithms are based exclusively on the information contained in the control packets (including the forward signaling and reverse feedback packets), we trace the behavior of the control packets carefully with event-driven simulation at the packet level. Link propagation delays and transmission delays for the control packets are included in the simulation. The control packets are routed on the shortest path, measured by the hop count. At each link, the control packets are transmitted at a higher priority than the data packets and, if needed, are stored in a high-priority queue. When a control packet arrives at its intended destination, an event will be triggered to update the virtual queues or the network costs, depending on whether it is a forward signaling or a feedback packet. The rest of the algorithm operations take place on time slot boundaries. On each time slot, a source sends one signaling packet to each node on the currently selected multicast tree to set up the tree and to inform the virtual source rate; it transmits data to the tree in the amount decided by the algorithms; it also computes a new tree according to the network costs that it currently knows, and the new tree will be set up and used on the next time slot. On each time slot, a network node sends at most one set of link costs (of its outgoing links) to each of the multicast sources.

To evaluate the stability of the real queues, we also need to track the sizes of the real data queues. In the interest of reducing simulation time, we trace the real data at the burst level instead of the packet level. Specifically, for each link, the simulator computes the amount of data it can transmit in the time slot, which is the difference of the link capacity and the amount of control packets transmitted during that time slot. Then, the burst of data is pushed to the next hop. Although there is a slight degree of inaccuracy in the simulated queue sizes, the outcome of whether or not the queues are stable is not altered by the burst-level simulation for data.

We simulate our algorithms over two commercial ISP network topologies obtained from the Rocketfuel project [16]. The first one consists of 41 nodes and 136 links; the second one consists

of 295 nodes and 1086 links. For each network, we assume the link capacities are exclusively allocated to the content distribution service. We assume there is a single distribution session.

On the smaller network with 41 nodes and 136 links, we select one node as the source and 20 nodes as the receivers. All other out-of-session nodes may be used as helper nodes. We assign 1 Gbps link capacity to all the links except some critical links. By critical links, we mean the links that become bottleneck easily if they do not have sufficient capacity. We assign 5 Gbps link capacity to each of the critical links. There are exogenous random arrivals of packets to the source. We assume that the number of packet arrivals on each time slot is a Poisson random variable and that the arrivals on different time slots are IID. The size of each data packet (chunk) is chosen to be 256 KB. Since the time slot size is equal to 1 second, the mean of the Poisson distribution is equal to $\lambda_s / (256 \times 8 \times 1000)$ and the unit is in packets. As an example, for the source arrival rate $\lambda_s = 1990$ Mbps, the mean number of arrivals is about 972 packets. The standard deviation is about 31.2 packets. The Poisson distribution is widely used to capture the total effect of many small disturbances when the outcome is non-negative and integer-valued. The maximum achievable session rate is 2 Gbps, which is obtained by running the subgradient algorithm introduced in [8]. The control packet size is under 400 bytes for our experiments. The time slot duration is 1 second. We have done experiments with different link propagation delays: 20 ms, 50 ms, 80 ms, and 100 ms. These cases have similar performance results. The 1-second time slot size is the relevant delay that determines the algorithm performance. Hence, we will only present the results for the case of 100 ms propagation delay at each link. We vary the mean arrival rate λ_s to see whether the algorithms can achieve network stability if the rate is below the maximum achievable session rate.

We also conducted experiments with other traffic models, such as truncated Pareto distributions, which have very large variances, and other distributions with very small variances. In addition, we conducted experiments where multiple multicast sessions exist simultaneously in the network. The results for these cases do not show much more insight than what we will subsequently present, and for brevity, they are not reported in the paper.

A. Algorithm Using Source Traffic Regulation and Approximate Min-Cost Tree Scheduling

In this subsection, we show the performance of the algorithm introduced in Section III. For the approximate tree selection algorithm, we use the algorithm by Charikar *et. al* with tree level 2, as proposed in [13]. A regulator queue is maintained only at the source. In the simulation, we set $\epsilon_1 = 1$ packet per second or 2.048 Mbps. Our main concern is whether the regulator and

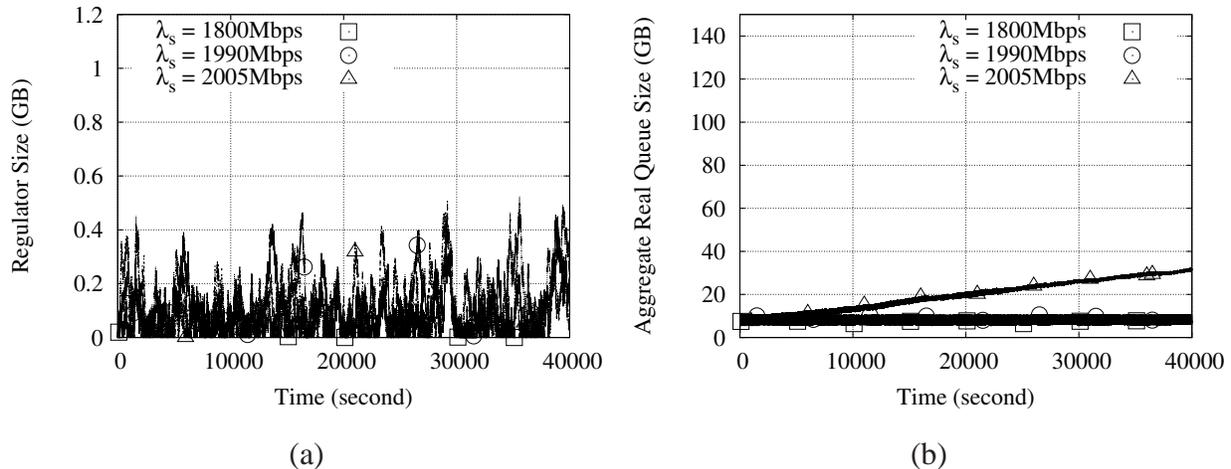


Fig. 3. Performance of the first algorithm; (a) regulator queue size when $\epsilon_1 = 1$ packet per second (or 2.048 Mbps); (b) aggregate real queue sizes.

real queues are bounded if the mean arrival rate to the source is below the maximum achievable session rate.

Fig. 3 shows that the network queues remain stable if λ_s is below the maximum achievable session rate (2 Gbps) even when λ_s is quite close to the maximum session rate. when λ_s exceeds the maximum achievable session rate, the total network queue size grows without a bound, which means some of the queues are unstable.

Fig. 3 (a) shows that the regulator queue size is bounded and is less than 500 MB even when λ_s is greater than the maximum achievable session rate. The reason is that the regulator queue is a simple single-server queue with a deterministic service rate and the algorithm sets the service rate to be slightly greater than λ_s , by ϵ_1 as given in (6). In the experiments here, ϵ_1 is very small relative to the traffic arrival rate and the traffic load is extremely heavy. For instance, when $\lambda_s = 1990$ Mbps, the traffic load (intensity) to the regulator queue is 0.999. Even under such heavy load, the regulator queue size is not very large. It can be made much smaller when ϵ_1 is increased. Fig. 3 (b) shows that, eventually, the aggregate real queue size over all the queues in the network is under 11 GB when the arrival rate ($\lambda_s = 1990$ Mbps) is slightly below the maximum achievable rate; that yields an average queue size of 80.9 MB at each link. The largest queue size at a link is about 625 MB under that arrival rate⁶. The queue sizes can be much smaller when the arrival rate is lower. These queue size values can be compared with

⁶In this section, the maximum queue sizes are what we observed during a long simulation run.

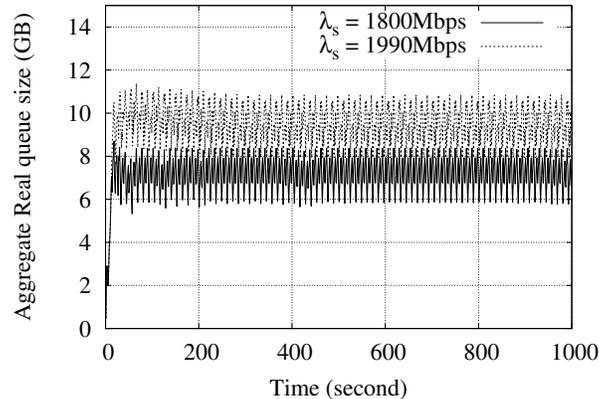


Fig. 4. Aggregate real queue sizes of the first algorithm on time slots 0 to 1000.

the bandwidth-delay product, which is 625 MB for a critical link and 125 MB for a non-critical link. Hence, using a 1 GB buffer at each link is more than sufficient for this test case. Since the source sends data packets at a rate of 1990 Mbps and the packets need to be duplicated to 20 receivers, about 5 GB data must flow through the network every second. The 11 GB data stored in all the queues is about twice of that amount. Hence, the amount of queued data is reasonably small.

It is important to point out that the eventual queue sizes shown in Fig. 3 are mostly determined by the transient phase of the algorithm, which is the phase at the beginning of the algorithm operation before the time-average rates approach the optimum. The queues build up at this phase because the algorithm hasn't found the right transmission rates yet. Once the time-average rates approach the optimum, the queues stop growing but oscillate around some values (see Fig. 4, which shows the aggregate queue sizes on time slots 0 to 1000). The oscillation is due to a feature of the algorithm, which is that the multicast session hops among different multicast trees even in the steady state. A consequence is that some of the links can be temporarily overloaded. From the simulation results, we see that the magnitude of the oscillation can be much smaller than the queue size itself. The oscillation of the aggregate queue size is less than 3 GB when $\lambda_s = 1990$ Mbps, which yields an average of 22.1 MB per link. When the multicast sessions are long-lasting and the network topology and link bandwidth are unchanging, it is enough to decide the buffer sizes based on the steady-state queue behavior. In the cases of Fig. 3, we see that the buffer requirement is much smaller than the bandwidth-delay products.

The good queue-size performance can be explained by two observations. First, by using

multiple multicast trees, the excess packets are spread out and queued all over the networks. Second, the algorithm converges reasonably fast. Fig. 6 (a) shows that, for each experiment where the arrival rate is below the maximum achievable session rate, the average data receiving rate per receiver reaches more than 90% of the arrival rate at around 100 to 200 seconds, and it reaches nearly 100% of the arrival rate after 1000 seconds. The average receiving rate is also a time average, which has long-time memory. Judging by Fig. 4, the queues of the system approach the steady state very quickly, within 50 time slots. The instantaneous receiving rates must have reached the arrival rate within the same time frame.

B. Algorithm Using Randomized Tree Scheduling

In this subsection, we show the stability of the algorithm introduced in Section IV. For the randomized tree selection algorithm, we let each link be selected as an edge on the random tree with a probability inversely proportional to its virtual queue size. The idea is to reduce the chance of selecting links with large virtual queues. Once a link is selected to be on the tree, all links that will lead to a loop with the selected links are removed from the candidate list. The candidate links are scanned repeatedly in a breadth-first order, starting from the source, until all the receivers are connected. How the random tree is selected will not affect the stability result as long as the condition in (18) is satisfied. However, the choice of the tree affects other aspects of performance, such as the queue sizes. When the queue sizes are considered in addition to throughput, what can be considered as good choices for the random tree remains an open question. Interested readers may refer to related literature in randomized link scheduling algorithms for wireless networks [17]–[20].

Fig. 5 shows that both the network virtual queues and the real queues remain stable if λ_s is below the maximum achievable session rate. When $\lambda_s = 1990$ Mbps, the aggregate real queue size of all the network queues is under 25 GB, which is about the amount of data flowing through the network in a 5-second interval. Under the same arrival rate, the average queue size per link is 183.8 MB and the largest real queue size at a link is 1.8 GB, which is 2.88 times of the bandwidth-delay product of a critical link. For $\lambda_s = 1800$ Mbps, the aggregate queue size is under 15 GB, the average queue size per link is under 110.3 MB, and the maximum queue size at a link is under 1 GB. Hence, the queue sizes can be made much smaller with a slightly reduced arrival rate. Compared with the first algorithm, a larger buffer is required at each link. When λ_s exceeds the maximum achievable session rate, both the aggregate virtual queue size and the aggregate real queue size grow indefinitely; the network is unstable. After reaching the

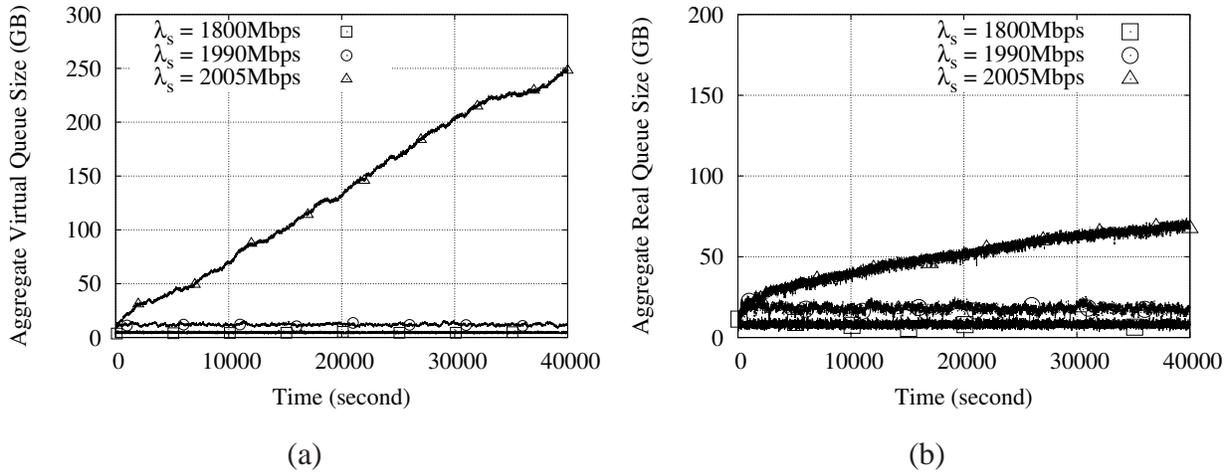


Fig. 5. The stability of the second algorithm; (a) aggregate virtual queue sizes; (b) aggregate real queue sizes.

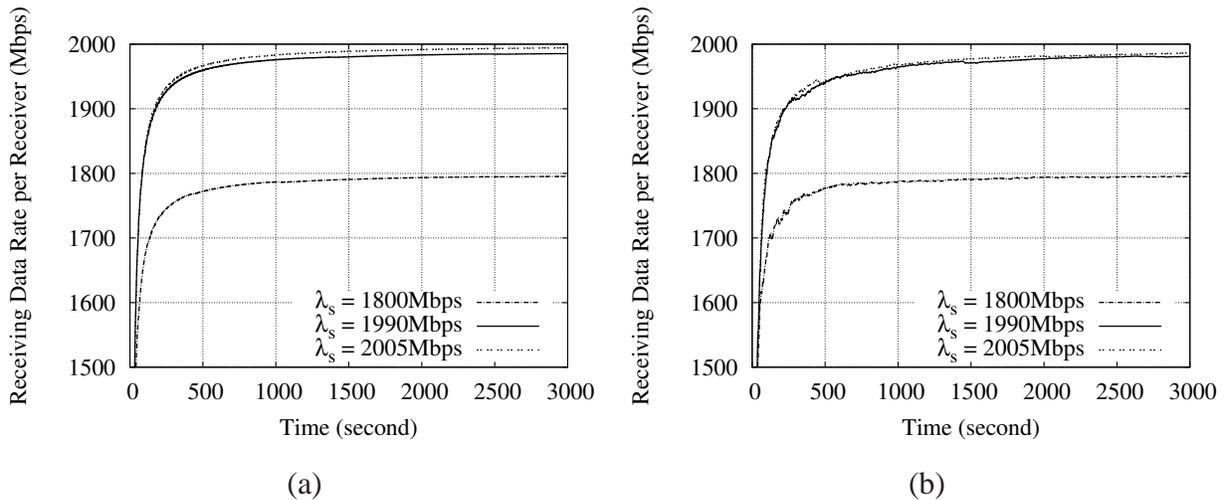


Fig. 6. Convergence of the two algorithms; (a) the first algorithm; (b) the second algorithm.

steady state, the real queue process exhibits more oscillation than that in the first algorithm. This is in part because the randomly selected tree on each time slot is not necessarily the min-cost tree. The aggregate queue size oscillates within a 10-GB range and the average oscillation is 73.5 MB at each link. Fig. 6 (b) shows that the average receiving rate converges reasonably fast to the arrival rate when the arrival rate is below the maximum achievable session rate.

C. Results for the Larger Network

We have conducted more experiments on the larger ISP network with 295 nodes and 1086 links. We assign 5 Gbps link capacity to each of the critical links and 1 Gbps to each of the other links, and we set up one multicast session with 39 receivers. The experiments suggest that the maximum session rate is around 2 Gbps, although we do not know the exact value. Fig. 7 (a) shows that the network is stable when the source rate λ_s is 2 Gbps and the real queues build up indefinitely when the source rate is 2.1 Gbps. For the case of $\lambda_s = 2$ Gbps, the aggregate queue size is under 60 GB or 80 GB for the first or second algorithms, respectively; the average queue size per link is under 55.2 MB or 73.7 MB, respectively. The oscillation of the aggregate queue size in the steady state is under 15 GB for both algorithms; after divided by the number of links, the average is under 13.8 MB per link. The largest queue size observed at a link is 3 GB for the first algorithm and 21 GB for the second algorithm. Evidently, tree selection in the first algorithm is better at avoiding congested links. In the second algorithm, a link coming out of the source has the largest queue size for much of the simulation duration. Better tree selection should help the second algorithm to reduce the largest queue size. Overall, these are promising results for buffer requirements, particularly considering the fact there are more receivers in this set of experiments. Fig. 7 (b) shows that, in each stable case where $\lambda_s = 2$ Gbps, the average receiving rate per receiver ramps up to 1.8 Gbps in under 300 time slots, which is 90% of the arrival rate. Again, this receiving rate is a time average as well. The convergence speed of the instantaneous receiving rate can be inferred from Fig. 7 (a). There, we see that the queues become steady in less than 500 or 1000 time slots for the first and second algorithm, respectively. We can deduce that convergence to greater than 99% of the final value has been achieved within those time slots. We conclude that the algorithms converge fairly fast.

D. Control Overhead

We will briefly describe a simple design of the signaling/control protocol and show that the control overhead is small. On each time slot, a source sends one signaling packet to each node on the multicast tree selected by the algorithm. The signaling packet directed toward a node contains a list of link IDs (32 bits each), which designate the node's outgoing links on the multicast tree. The packet also contains a 32-bit virtual source rate and a 32-bit multicast session ID (which can use the source ID). The second type of control packets has the function of carrying the costs of the links (i.e., links' virtual queue sizes) back to the sources of the multicast sessions. On each time slot, a node sends one feedback packet to each source. The packet contains the

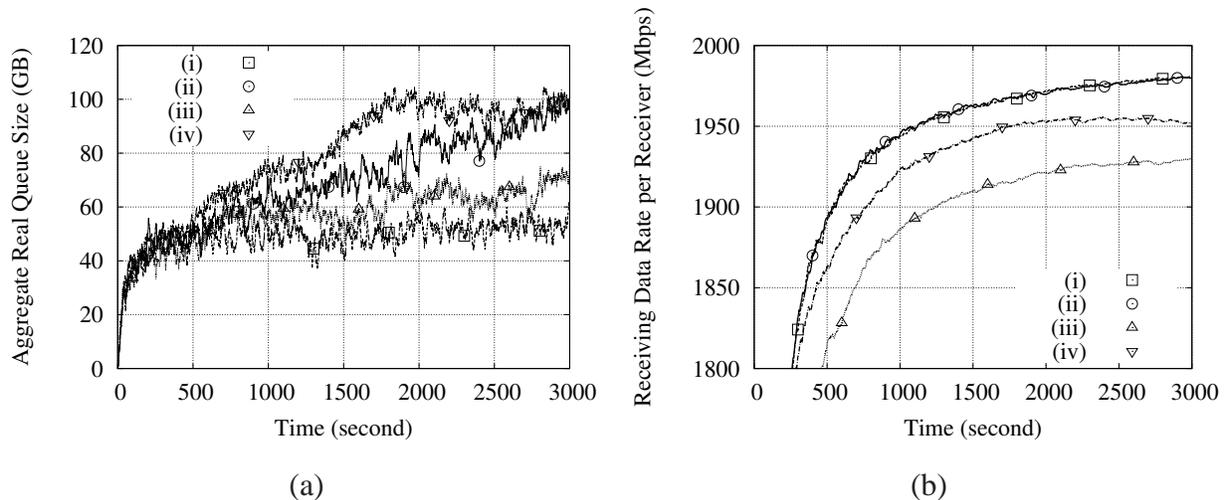


Fig. 7. The experiments on a network with 295 nodes and 1086 links; (a) aggregate real queue sizes; (b) average receiving rate per receiver. Legends: (i) the first algorithm with $\lambda_s = 2$ Gbps, (ii) the first algorithm with $\lambda_s = 2.1$ Gbps, (iii) the second algorithm with $\lambda_s = 2$ Gbps, (iv) the second algorithm with $\lambda_s = 2.1$ Gbps.

IDs and costs (32 bits each) of the node's outgoing links and the 32-bit node ID. Every control packet has an additional 20-byte header containing miscellaneous information. The size of each control packet is well under 1000 bytes in typical cases.

For a network with n nodes and m links, the total size of all the signaling packets from a source is at most $(20 \times 8 + 32 + 32)n + 32m$ bits on each time slot. Consider a fairly aggressive example where the network has 300 nodes and 3000 links and the time slot size is 0.5 seconds. On each time slot, the total size of the forward signaling packets is 163,200 bits, which means the control traffic rate from the source is 326.4 Kbps. If the source data rate is 100 Mbps, the forward signaling overhead is only 0.3264%. Furthermore, the control traffic rate is independent of the source data rate. As technology progresses to 1 Gbps or 10 Gbps source rates, the overhead becomes 0.03264% or 0.003264%, respectively. It is also sensible to define the signaling overhead with respect to the *total* data traffic rate for the multicast session. For a multicast session with r receivers, the total data traffic rate entering the receivers is equal to r times of the source rate. If a multicast session has 100 receivers and the source rate is 100 Mbps, the total data rate is 10 Gbps. The signaling traffic is 0.003264% of 10 Gbps.

The second type of control packets has the function of carrying the costs of the links (i.e., links' virtual queue sizes) back to the sources of the multicast sessions. On each time slot, each node sends one feedback packet to every source. The packet contains the IDs and costs of the node's outgoing links, the node ID and a 20-byte header. Then, the total size of all the feedback

packets received by a source on each time slot is $(20 \times 8 + 32)n + (32 + 32)m$ bits. For a network with 300 nodes and 3000 links, the total size is 249,600 bits, which corresponds to a traffic rate of 499.2 Kbps when the time slot size is 0.5 seconds. For 100 Mbps, 1 Gbps or 10 Gbps source rates, the overhead is 0.4992%, 0.04992% or 0.004992%, respectively, when compared with the source rate; it is 0.004992%, 0.0004992% or 0.00004992%, respectively, when compared with the total data rate for a session with 100 receivers.

VI. RELATED WORK

Research on similar stability questions has been very active, but generally, in the context of unicast (e.g. [9], [14], [21]–[26]), possibly with multiple paths per connection. The presence of multicast puts our problem in a class of its own in that many earlier stable control algorithms, such as the maximum backpressure-based algorithm [22], and techniques for stability analysis are not directly applicable. The main reason is that, unlike unicast, the flow conservation condition no longer holds under multicast.

There are several papers on the stability analysis of multicast/broadcast [27]–[31]. Except [27], most of these assume an access constrained network. In [28]–[30], various Bittorrent-like algorithms are proposed and are proved to achieve the optimal performance in terms of the distribution rate and/or delay, where the bandwidth bottleneck is at the upload links. In [27], Massoulié *et al.* present a simple local-control algorithm for broadcast in a general network, which provably achieves the optimal broadcast rate. The algorithm only allows broadcast from a single source and requires all nodes in the network to receive a complete copy of the data. In [31], the stability of multiple-tree-based peer-to-peer live streaming is analyzed, where stability is defined as the availability of data in the presence of node dynamics.

Another salient aspect of the universal swarming problem is most related to the problem of link scheduling in wireless networks subject to link interference constraints, which has attracted much attention recently [9], [14], [15], [21], [26], [32]–[36]. In [21], Tassiulas *et al.* showed that the maximum-weight link schedule achieves (i.e., stabilizes) the interior of the throughput region, where the weights are the queue size differences, or the backpressure. However, finding such a schedule is in general an NP-hard problem. The universal swarming problem usually involves an NP-hard subproblem in order to achieve the entire throughput region, which is to find a minimum-cost Steiner tree. This similarity makes many of the concerns and investigative approaches in the wireless link scheduling problem relevant to the universal swarming problem. In [14], [37], Lin *et al.* showed that approximation algorithms for the maximum-weight scheduling

problem can be used to stabilize a portion of the throughput region. Some researchers considered maximal scheduling algorithms and studied what their stability regions are [25], [26], [34]–[36]. Other authors proposed randomized scheduling algorithms that achieve the entire throughput region [15] [18].

VII. CONCLUSIONS AND DISCUSSIONS

In universal swarming, packets are distributed to the receivers along multiple multicast trees. The paper focuses on analyzing the stability of the algorithms for sending dynamically arriving packets onto the trees. To achieve the throughput region, we encounter a min-cost Steiner tree problem, which is NP-hard. Multi-hop traffic is another difficulty for finding stable universal swarming algorithms. We propose a γ -approximation min-cost tree scheduling algorithm with network signaling and source regulators. It guarantees network stability in a reduced throughput region, where the reduction ratio is no more than the approximation ratio of the algorithm for the min-cost tree problem. We further develop a randomized tree-scheduling algorithm with network signaling. It achieves the throughput region and stabilizes the virtual queues. Moreover, the real queue processes are rate stable and the average traffic intensity at each link is strictly less than one. However, whether or not the real queues are stable remains an open question.

In the worst case, even finding an approximate min-cost tree can be very time consuming. However, the algorithms and results in this paper can still be practically relevant. First, our algorithms do not require to find the true min-cost tree. In practice, there is a time budget (e.g., the time slot size) in the tree computation step. The tree that is found to have the least cost during the budgeted time will be used by our algorithms. Second, there are several possibilities that, in practice, the tree-computation time may not be prohibitively long. (i) In small networks (less than 100 nodes and links), finding the min-cost tree can be quite fast. (ii) For the intended application, the network topology is a fixed one, not an arbitrary one. One may be able to find specialized, fast algorithms for that particular topology. (iii) For a fixed topology, one may be able to find a heuristic algorithm that has near optimal performance most of the time. (iv) In many practical networks, such as the ones used in our simulation, it is sometimes quite clear that a small number of links are the critical ones and they almost always experience large queue sizes. It is not necessarily hard to find a min-cost (or nearly min-cost) tree, if only these critical links need to be inspected. Finally, the tree computation time can be reduced drastically at the expense of a small reduction in the throughput. If the time budget for tree computation is chosen to be very short, e.g., close to the round-trip propagation time, we may restrict each multicast

session to use a small number of precomputed trees (e.g., 4 – 100 trees); finding the min-cost tree among them is trivial. Experiences have shown that the loss in throughput due to the restriction on the trees is likely to be small (see [38] for the unicast case). The performance guarantees found in the paper still apply if we modify the optimization problem by adding the constraints about the allowed trees.

We now briefly discuss how to adapt the algorithms to the case of small buffer sizes. This is possible because, fundamentally, the algorithms use the virtual queue sizes for rate computation and they will find the optimal rates regardless of the actual buffer sizes. The first idea is again to use a small number of pre-computed trees for each multicast session, say L trees. Since finding the least-cost tree among the L trees is extremely easy, we can reduce the time slot size to near the round-trip propagation time, and hence, reduce the bandwidth-delay product. Next, we wish to reduce temporary overload in the steady state due to tree-hopping. The idea is to use the L multicast trees *simultaneously* with the correct tree rates. In the modified algorithm, every source computes the time average of the virtual source rates (for releasing virtual packets to the network) that each of its L trees sees; this yields the time averages of the virtual tree rates and these time-average rates converge to the optimal tree rates regardless of the buffer size⁷. The computed average rates are used as targets for what the real tree rates (and hence, the real source rate) should be. In a static network environment (i.e., one with fixed topology, constant link bandwidth and long-lasting multicast sessions), the real tree rates can be set directly at the time-average virtual tree rates when the latter have stabilized⁸. Then, the only remaining possible cause for queue buildup is the randomness in the arrival process. A very small buffer size is sufficient to absorb such traffic fluctuation and prevent packet losses (e.g., 10-100 packets). Finally, we can further enhance the algorithms to reduce the transient buildup of the queues so that the algorithms can cope with macro-level dynamics, by which we mean changes in the network topology and link bandwidth or the arrivals/departures of multicast sessions. The idea is to have another level of adaptation by the real tree rates. For instance, a real tree rate can rapidly increase to 80% of the time average of the corresponding virtual tree rate; after that, it increases gradually until packet losses occur, at which point, it drops to half of the time-average virtual tree rate. This way, the real rates will not overshoot too much. Note that the adaptation of the real rates and the computation of the time averages are done concurrently; there is no

⁷Here, the optimal rate-allocation problem is a modified one. Compared with the original optimization problem, the only modification is that it assumes L fixed trees per multicast session.

⁸The initial waiting time for the convergence of the time-average virtual rates is not a concern under the static assumption.

waiting for the convergence of the time-average rates.

APPENDIX A PROOFS

A. Proof of Theorem 5

Recall that every packet at any link has a hop count from the source, which is the hop count on its tree path to the link. If a packet has a hop count h when it arrives at a link, we say it belongs to the h^{th} -hop traffic. Let $Q_e(k, h)$ denote the queue backlog at link e at time k contributed by all first-hop through h^{th} -hop traffic. We assume $Q_e(k, 0) = 0$ for ease of presentation.

For each $e \in E$, let $\bar{h}_e \leq |V| - 1$ be the largest hop count of any packet in e . Let $\bar{h} = \max_{e \in E} \bar{h}_e$. We claim that for all $h = 1, \dots, \bar{h}$,

$$Q_e(k, h) \leq M_h, \quad \forall k, \forall e \in E, \quad (27)$$

where the constants satisfy $M_1 \leq M_2 \leq \dots \leq M_{\bar{h}} < \infty$. We prove this claim by induction.

Base: When $h = 1$, note that $Q_e(k, 1) \leq q_e(k)$ and $q_e(k) \leq M_e$ by Corollary 3. Let $M_1 = \max_{e \in E} M_e$. Then, (27) holds for $h = 1$ and for all k .

Assume (27) holds for $1, \dots, h - 1$ and for all k .

Induction on h : Let $x_e(k, h)$ denote the number of packets arriving at link e on time slot k that belong to the first-hop through h^{th} -hop traffic. Then, during any interval of time, k_0 through k , the total number of arrivals is no more than the sum of the number of packets released by the sources during this interval that travel through link e and all the backlogged, first-through- $(h - 1)^{\text{th}}$ -hop packets in the network at time k_0 , i.e.,

$$\begin{aligned} \sum_{u=k_0}^k x_e(u, h) &\leq \sum_{u=k_0}^k \sum_{s \in S} D_s(u) I(e, t_s(u)) + \sum_{e' \in E} Q_{e'}(k_0, h - 1) \\ &\leq \sum_{u=k_0}^k \sum_{s \in S} D_s(u) I(e, t_s(u)) + |E| \cdot M_{h-1}. \end{aligned} \quad (28)$$

(28) holds due to the induction hypothesis.

Assume all the queues are empty at time 0. For all k ,

$$Q_e(k, h) = \max_{0 \leq k_0 \leq k} \left\{ \sum_{u=k_0}^k x_e(u, h) - c_e(k - k_0 + 1) \right\} \quad (29)$$

$$\leq \max_{0 \leq k_0 \leq k} \left\{ \sum_{u=k_0}^k \sum_{s \in S} D_s(u) I(e, t_s(u)) + |E| \cdot M_{h-1} - c_e(k - k_0 + 1) \right\} \quad (30)$$

$$\leq M_e + |E| \cdot M_{h-1}. \quad (31)$$

(29) is by an elementary fact about the queue size. It is derived by expanding the one-step queue update backward in time. It says that the queue size is the largest difference between the total arrivals and the accumulated link capacity on any sub-interval ending at the current time k . (30) is by using (28) in (29). (31) is by Lemma 4. We can define $M_h = M_e + |E| \cdot M_{h-1}$.

Finally, note that the overall queue backlog $Q_e(k)$ at link e is equal to $Q_e(k, \bar{h}_e)$, where $\bar{h}_e \leq \bar{h}$. Hence, $Q_e(k) \leq M_{\bar{h}_e} \leq M_{\bar{h}}$ for all k .

B. Proof of Theorem 11

The proof follows from Theorem 8.0.3 in [39]. Denote by \mathcal{X} the state space of the Markov process $\{x(k)\} = \{q(k), t(k)\}$. Define a finite subset of the state space $\hat{\mathcal{X}} = \{x \in \mathcal{X} : \sum_{e \in E} q_e \leq \frac{M}{\epsilon}\}$, for M and ϵ as specified in Lemma 9. Note that by assumption AS 3, the process $\{x(k)\}$ is an x^* -irreducible Markov chain with $x^* = (q^*, t^*)$, where $q^* = \vec{0}$ and t^* is a fixed set of minimum-cost trees under the link cost vector $q^* = \vec{0}$;⁹ and it is also aperiodic on the countable state space \mathcal{X} . Assumption AS 3 says that on every time slot, there is some nonzero probability that no new arrivals enter the sources. The system will empty after a finite number of such successive “no arrival” slots, an event that has a positive probability. This implies the process $\{x(k)\}$ is an x^* -irreducible Markov chain (i.e., $\sum_{k=0}^{\infty} P^k(x, x^*) > 0, x \in \hat{\mathcal{X}}$). Aperiodicity follows from $P(x^*, x^*) = P\{A(k) = 0\} \cdot P\{t^* \text{ is chosen}\} > 0$.

By Lemma 9, the chain $\{x(k)\}$ satisfies the Foster-Lyapunov drift conditions [39]. Now all the conditions of Theorem 8.0.3 in [39] are met. Hence, Theorem 11 holds.

C. Proof of Theorem 12

According to (16),

$$q_e(k+1) \geq q_e(k) + \sum_{s \in S} A_s(k) I(e, t_s(k)) - (c_e - \epsilon_2), \forall e \in E.$$

Summing the above inequality from time slots 0 to k , we have

$$\begin{aligned} \sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u)) &\leq (c_e - \epsilon_2)(k+1) + q_e(k+1) - q_e(0) \\ &\leq (c_e - \epsilon_2)(k+1) + q_e(k+1). \end{aligned} \tag{32}$$

⁹There might be multiple sets of the minimum-cost trees. Let t^* be an arbitrary one of them. Since the tie is broken uniformly at random, there is a non-zero probability that we choose the set t^* .

Dividing both sides of (32) by $k + 1$ yields

$$\begin{aligned} \frac{\sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u))}{k+1} &\leq c_e - \epsilon_2 + \frac{q_e(k+1)}{k+1} \\ &= c_e - \epsilon_2 + \frac{\sum_{u=0}^{k+1} q_e(u) - \sum_{u=0}^k q_e(u)}{k+1} \\ &= c_e - \epsilon_2 + \frac{\sum_{u=0}^{k+1} q_e(u)}{k+2} \cdot \frac{k+2}{k+1} - \frac{\sum_{u=0}^k q_e(u)}{k+1}. \end{aligned}$$

Taking the limit on both sides of the above inequality as k goes to infinity, we get,

$$\begin{aligned} \limsup_{k \rightarrow \infty} \frac{\sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u))}{k+1} &\leq c_e - \epsilon_2 + \lim_{k \rightarrow \infty} \frac{\sum_{u=0}^{k+1} q_e(u)}{k+2} \cdot \lim_{k \rightarrow \infty} \frac{k+2}{k+1} \\ &\quad - \lim_{k \rightarrow \infty} \frac{\sum_{u=0}^k q_e(u)}{k+1} \\ &= c_e - \epsilon_2. \end{aligned}$$

The last equality holds because $\lim_{k \rightarrow \infty} \frac{\sum_{u=0}^{k+1} q_e(u)}{k+2} = \lim_{k \rightarrow \infty} \frac{\sum_{u=0}^k q_e(u)}{k+1} = E[\hat{q}_e]$ by Theorem 11. Hence, (25) holds.

Now taking the expectation on the both sides of (32) yields

$$\begin{aligned} E\left[\sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u))\right] &\leq (c_e - \epsilon_2)(k+1) + E[q_e(k+1)] \\ &\leq (c_e - \epsilon_2)(k+1) + M_e, \end{aligned}$$

where $E[q_e(k+1)] \leq M_e < \infty$. Because Theorem 11 says, $\lim_{k \rightarrow \infty} E[q_e(k)] \rightarrow E[\hat{q}_e]$ and $E[\hat{q}_e] < \infty$, such M_e exists. Dividing both sides of the above inequality by $k+1$ and taking the limit, we have,

$$\limsup_{k \rightarrow \infty} E\left[\frac{1}{k+1} \sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u))\right] \leq c_e - \epsilon_2 + \lim_{k \rightarrow \infty} \frac{M_e}{k+1} = c_e - \epsilon_2.$$

D. Proof of Corollary 13

For any time slot k , let $\{a_e(k)\}$ denote the real packet arrival process at link e . The real queue dynamic equation can be written as

$$Q_e(k+1) = \max[Q_e(k) - c_e, 0] + a_e(k). \quad (33)$$

Note that, for any time slot k , the number of cumulative arrivals at the real queue is no more than the number of cumulative arrivals at the virtual queue, i.e.,

$$\sum_{u=0}^k a_e(u) \leq \sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u)).$$

Then, by Theorem 12,

$$\limsup_{k \rightarrow \infty} \frac{\sum_{u=0}^k a_e(u)}{k} \leq \limsup_{k \rightarrow \infty} \frac{\sum_{u=0}^k \sum_{s \in S} A_s(u) I(e, t_s(u))}{k} \leq c_e - \epsilon_2 \leq c_e.$$

According to the Rate Stability Theorem in [12] (Theorem 2.4), $Q_e(k)$ is rate stable, i.e.,

$$\lim_{k \rightarrow \infty} \frac{Q_e(k)}{k} = 0 \quad \text{with probability 1.}$$

REFERENCES

- [1] J. Lee and G. de Veciana, "On application-level load balancing in FastReplica," *Computer Communications*, vol. 30, no. 17, pp. 3218–3231, November 2007.
- [2] D. Kostić, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat, "Maintaining high bandwidth under dynamic network conditions," in *Proceedings of USENIX Annual Technical Conference*, 2005.
- [3] B.-G. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz, "ChunkCast: An anycast service for large content distribution," in *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.
- [4] BitTorrent Website, <http://www.bittorrent.com/>.
- [5] K. Park and V. S. Pai, "Scale and performance in the CoBlitz large-file distribution service," in *Proceedings of the 3rd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.
- [6] K. Cho, K. Fukuda, H. Esaki, and A. Kato, "The impact and implications of the growth in residential user-to-user traffic," in *Proceedings of the ACM SIGCOMM*, 2006.
- [7] X. Zheng, C. Cho, and Y. Xia, "Optimal peer-to-peer technique for massive content distribution," in *Proceedings of the IEEE INFOCOM*, 2008.
- [8] —, "Content distribution by multiple multicast trees and intersession cooperation: Optimal algorithms and approximations," in *Proceedings of the 48th IEEE Conference on Decision and Control (CDC)*, 2009.
- [9] M. J. Neely, "Energy optimal control for time varying wireless networks," *IEEE Transactions on Information Theory*, vol. 52, no. 7, pp. 2915–2934, July 2006.
- [10] —, "Dynamic power allocation and routing for satellite and wireless networks with time varying channels," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.
- [11] M. J. Neely, E. Modiano, and C. P. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 396–409, 2008.
- [12] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool, 2010.
- [13] M. Charikar and C. Chekuri, "Approximation algorithms for directed Steiner problems," *Journal of Algorithms*, vol. 33, no. 1, pp. 73–91, 1999.
- [14] X. Lin and N. B. Shroff, "The impact of imperfect scheduling on cross-layer rate control in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 302–315, April 2006.
- [15] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proceedings of the IEEE INFOCOM*, 1998.
- [16] *Rocketfuel: An ISP Topology Mapping Engine*, University of Washington, <http://www.cs.washington.edu/research/networking/rocketfuel/>.
- [17] X. Lin and S. B. Rasool, "Constant-time distributed scheduling policies for ad hoc wireless networks," in *Proceedings of the IEEE CDC*, 2006.
- [18] S. Rajagopalan, D. Shah, and J. Shin, "Network adiabatic theorem: an efficient randomized protocol for contention resolution," in *Proceedings of SIGMETRICS*, 2009.

- [19] L. Jiang, M. Leconte, J. Ni, R. Srikant, and J. Walrand, "Fast mixing of parallel Glauber dynamics and low-delay CSMA scheduling," *IEEE Transactions on Information Theory*, vol. 58, no. 10, pp. 6541–6555, October 2012.
- [20] J. Ni, B. Tan, and R. Srikant, "Q-CSMA: Queue-length based CSMA/CA algorithms for achieving maximum throughput and low delay in wireless networks," in *Proceedings of the IEEE INFOCOM*, 2010.
- [21] 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, December 1992.
- [22] L. Tassiulas, "Adaptive back-pressure congestion control based on local information," *IEEE Transactions on Automatic Control*, vol. 40, pp. 236–250, 1995.
- [23] A. Eryilmaz and R. Srikant, "Fair resource allocation in wireless networks using queue-length based scheduling and congestion control," in *Proceedings of the IEEE INFOCOM*, 2005.
- [24] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends in Networking*, vol. 1, no. 1, pp. 1–144, 2006.
- [25] X. Wu, R. Srikant, and J. R. Perkins, "Scheduling efficiency of distributed greedy scheduling algorithms in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 6, pp. 595–605, 2007.
- [26] C. Joo, X. Lin, and N. B. Shroff, "Understanding the capacity region of the greedy maximal scheduling algorithm in multi-hop wireless networks," in *Proceedings of the IEEE INFOCOM*, 2008.
- [27] L. Massoulié, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proceedings of the IEEE INFOCOM*, 2007.
- [28] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs," in *Proceedings of SIGMETRICS*, 2008.
- [29] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for P2P streaming systems," in *Proceedings of the IEEE INFOCOM*, 2007.
- [30] S. Sanghavi, B. Hajek, and L. Massoulié, "Gossiping with multiple messages," in *Proceedings of the IEEE INFOCOM*, 2007.
- [31] G. Dan, V. Fodor, and I. Chatzidrossos, "On the performance of multiple-tree-based peer-to-peer live streaming," in *Proceedings of the IEEE INFOCOM*, 2007.
- [32] M. Johansson and L. Xiao, "Cross-layer optimization of wireless networks using nonlinear column generation," *IEEE Transactions on Wireless Communications*, vol. 5, no. 2, pp. 435–445, Feb. 2006.
- [33] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle, "Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks," in *Proceedings of the IEEE INFOCOM*, 2006.
- [34] A. Dimakis and J. Walrand, "Sufficient conditions for stability of longest-queue-first scheduling: second-order properties using fluid limits," *Advances in Applied Probability*, vol. 38, no. 2, pp. 505–521, 2006.
- [35] G. Sharma, N. B. Shroff, and R. R. Mazumdar, "Joint congestion control and distributed scheduling for throughput guarantees in wireless networks," in *Proceedings of the IEEE INFOCOM*, 2007.
- [36] A. Gupta, X. Lin, and R. Srikant, "Low-complexity distributed scheduling algorithms for wireless networks," in *Proceedings of IEEE INFOCOM*, 2007.
- [37] X. Lin, N. B. Shroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1452–1463, Aug. 2006.
- [38] K. Rajah, S. Ranka, and Y. Xia, "Scheduling bulk file transfers with start and end times," *Computer Networks*, vol. 52, no. 5, pp. 1105–1122, April 2008.
- [39] S. Meyn, *Control Techniques For Complex Networks*. Cambridge University Press, 2008.

PLACE
PHOTO
HERE

Xiaoying Zheng received the bachelor's and master's degrees in computer science and engineering from Zhejiang University, P.R. China, in 2000 and 2003, respectively, and the PhD degree in computer engineering from the University of Florida, Gainesville, in 2008. She is an assistant professor at Shanghai Research Center of Wireless Communications, from March 2009 to October 2011. She then joined Shanghai Advanced Research Institute, Chinese Academy of Sciences in 2012 as an associate professor. Her research interests include applications of optimization theory in networks, distributed systems, performance evaluation of network protocols and algorithms, peer-to-peer overlay networks, content distribution, and congestion control. She was an assistant professor in Shanghai Research Center for Wireless Communications and Key Lab of Wireless Sensor Network and Communications, Chinese Academy of Sciences (CAS), China. Her research interests include applications of optimization theory in networks, performance evaluation of network protocols and algorithms, peer-to-peer overlay networks, wireless networks, content distribution and congestion control.

PLACE
PHOTO
HERE

Chunglae Cho received the B.S. and M.S. degrees in computer science from Pusan National University, Korea, in 1994 and 1996, respectively. He worked as a research staff member at Electronics and Telecommunications Research Institute, Korea, between 2000 and 2005. He is currently working toward a Ph.D. degree at the Computer and Information Science and Engineering department at the University of Florida, Gainesville, FL. His research interests are in computer networks, including resource allocation and optimization on peer-to-peer networks and wireless networks.

PLACE
PHOTO
HERE

Ye Xia is an associate professor at the Computer and Information Science and Engineering department at the University of Florida, starting in August 2003. 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.