

# A Distributed Algorithm for Max-min Fair Bandwidth Sharing

Xinran (Ryan) Wu and Andrew A. Chien  
 Department of Computer Science and Engineering and  
 Center for Networked Systems  
 University of California, San Diego  
 La Jolla, CA, 92093-0114, USA

**Abstract**—Dense Wavelength Division Multiplexing (DWDM), dedicated optical paths, high speed switches, and high speed routers are giving rise to networks with plentiful bandwidth in the core. In such networks, bottlenecks and congestion are concentrated more at the edge and at end nodes. We consider how to efficiently and fairly share source and sink capacities in such network environments.

As a foundation, we first describe a global algorithm which calculates max-min fair rate allocation for network sessions with known desired peak rates. We then present a novel distributed rate allocation algorithm which controls each source and sink independently with local information, adaptively allocating its capacity to candidate sessions. This algorithm is given no knowledge of the desired session rates, but rather discovers them. This allows it to be used in general network settings. We prove that our distributed algorithm converges rapidly to the max-min fair rate allocation in a discrete time system. Simulations confirm this behavior, and further show that convergence is in practice fast in networks of 64 to 1024 nodes. Simulations also prove that the stability and convergence results still hold in asynchronous distributed settings with a range of network delays.

## I. INTRODUCTION

Continuing advances in optical transmission are driving rapid increases in feasible network bandwidths at densities of over a terabit per optical fiber for both metro and long haul networks. A key driver is dense wavelength division multiplexing (DWDM) which allows each optical fiber to carry hundreds of lambdas (i.e. wavelengths) of 10 or 40 Gbps. These capabilities are being used to build high-speed traditional shared, routed internet networks, private dark fiber networks [39], [23], and based on new technologies for dynamic configuration, dynamic private lambda networks. Examples of these dynamic private networks include LambdaGrid [18], OptIPuter [45], CANARIE [3], Netherlight [8], NAREGI [7], etc.).

Our research focuses in particular on networks where sets of high-speed private optical paths are constructed dynamically, forming a high-performance private network. In such networks, the network bandwidth in the core of lambdas is high, matching or exceeding the speeds at which endpoints can source or sink network traffic. In short, the network of lambdas provides high bandwidth, very low loss packet communication. We call these high speed, private networks “lambda networks.”

However, lambda-networks differ from traditional shared IP networks in several key ways. First, shared IP networks were designed to manage internal resource bottlenecks (links,

routers or switches), and include mechanisms for efficient link multiplexing and management of network congestion. In contrast, lambda-networks have abundant network bandwidth and switching, and thus have little network congestion and little packet loss. However, congestion does arise at end nodes due to their limited capacity, and give rise to packet loss there. Second, because of application trends in WWW technologies and other large-scale data sharing, there is increasing use of multipoint-to-point traffic patterns (as evidenced in the worldwide web, P2P networks [6], [2], and large-scale scientific data sharing [4], [5], [1]). High-speed core networks enable many fast flows to converge on an endpoint, raising challenges in efficient and fair sharing of source and sink capacity.

As a result, we study how end nodes should manage their capacity across multiple sessions in networks with plentiful bandwidth. In any distributed networks, end nodes must allocate their capacity based on incomplete information: local session status and approximations of desired rates. This problem differs in three important ways from that addressed by TCP and its variants [31], [21], [22], [49]. First, TCP and variants all manage single flows, providing rate and congestion control. They do not consider interactions amongst flows. Second, TCP and variants generally assume shared networks with internal congestion, so they focus on managing congestive packet loss within the network, not at the endpoints. Third, when TCP flows share a link with different round-trip delays (RTT), the delivered bandwidth is inversely proportional to RTT, so the sharing is unfair, and this is not considered a major problem. In contrast, achieving fair allocation with varied RTT’s is our primary goal.

Our distributed rate allocation problem is distinct from that explored in ATM networks [27], [14] and IP networks [35], which focus primarily on allocating router and switch capacity within the network. We solve the inverse problem. With plentiful bandwidth in the core, we focus on resource allocation for edge nodes.

Our work on distributed rate allocation is motivated by and provides an algorithmic basis for an increasing number of new protocols [11], [38], [48]. The analytical and dynamic behavior modeling presented in this paper complement the wealth of empirical work on these protocols. For example, consider the Group Transport Protocol (GTP) [48] which receiver-based rate control and coordinates amongst active sessions to achieve high performance in lambda networks. Our previous

work involved a range of empirical studies, but the distributed rate allocation framework described in this paper significantly advances our previous GTP work, extending management and control to source and sinks and providing strong properties of efficiency, fairness, and convergence.

The critical objectives for distributed rate allocation in networks are:

- Efficiency: Maximize utilization of source and sink capacity;
- Fairness: Allocate a fair share of source and sink capacity to each session (eventually);
- Stability & Convergence: Converge to a unique, optimal, and fair rate allocation from any initial state.

We describe a discrete-time distributed rate allocation algorithm. The distributed algorithm uses local information only to allocate capacity at each source and sink, assigning an expected rate to each session. So for each session, its actual rate is determined by the minimum of the expected rate at its source and sink, and its desired rate (which is not generally known to the network). Our distributed algorithm converges rapidly to a max-min fair rate allocation [12] [34], [27], [42], and also adapts quickly when desired rates change.

Specific contributions are as follows:

- 1) a mathematical model of lambda-networks, and a formulation of the bandwidth sharing problem in lambda-networks,
- 2) a global rate allocation algorithm which calculates the optimum rate allocation given the constraints of a set of finite desired session rates. We prove that the algorithm converges to the unique max-min fair rate allocation,
- 3) a distributed rate allocation algorithm which allocates source and sink bandwidth. We prove that the distributed algorithm converges to the unique max-min fair rate allocation,
- 4) an evaluation of the distributed rate allocation algorithm which shows that it achieves efficient and fair allocations. And further that it converges rapidly for a range of network configurations,
- 5) an evaluation of the distributed rate allocation algorithm in an asynchronous, distributed network (varying size, latency, and synchrony). Our simulation results demonstrate the strong convergence and stability properties also hold in these more practical circumstances.

This paper describes work that is part of the OptIPuter project [45], [46], [47], [48], which explores the use of configurable optical networks to provide high bandwidth and tighter coupling amongst computational and storage resource.

The rest of the paper is organized as follows. In Section II, we introduce the model of the lambda-networks, and formulate the bandwidth sharing and rate adaptation problem in the lambda-networks. In Section III, we present a global algorithm which calculates the max-min fair rate allocation in the lambda-networks. We prove that such max-min fair allocation is unique. In Section IV, we propose a distributed algorithm with capacity allocation and rate adaptation at each source and sink. We prove the stability and convergence properties of the proposed algorithm. In Section V and VI,

we conduct numerical case studies to illustrate the convergence properties of the distributed algorithm in both synchronous and asynchronous distributed settings. Related works are briefly described in Section VII, and we conclude in Section VIII.

## II. PROBLEM FORMULATION

### A. Modeling Lambda-Networks

In the lambda-networks, high-speed private optical paths can be constructed on-demand to provide plentiful dedicated end-to-end bandwidth. Such networks differ from the traditional shared Internet in that there is essentially no packet loss due to congestion within the network, and any contention and congestion loss happens at end nodes (sources and sinks).

We make the following assumptions when modeling lambda-networks.

- 1) With plenty of network bandwidth is provided by private, dynamically configured lambdas, there is no traffic congestion within the network. Because the capacity of lambda-network is higher than the desired rate of each session, no modeling of network internals is required. The rate control and allocation problem is reduced to the problem of how to efficiently and fairly share the source and sink capacities amongst the traffic sessions.
- 2) Each each source and sink node has explicit knowledge regarding its capacity, usually its network interface speed, local packet processing capacity or a shared link nearby. With capacity information, each node knows how much capacity it has to allocate across its sessions.
- 3) each session has its own desired rate – the peak rate it can reach. The desired rate can be also interpreted as the maximum data handling speed of the applications using the network. The desired rate of each session is unknown to its source and sink, and can vary over time.

### B. Notation

Our notation for modeling lambda-networks follows.

Consider a lambda-Network  $\mathcal{G}$  with a finite set  $\mathcal{V}$  of nodes and a set  $\mathcal{K}$  of  $K$  active sessions. Let  $\mathcal{V}^S$  and  $\mathcal{V}^R$  denote the set of source and sink nodes, respectively. Note that multiple sessions may start or terminate at the same source or sink node.

Consider a discrete-time model, where time is divided into slots of equal length. Let each session  $k \in \mathcal{K}$  be associated with a desired (peak) rate  $M_k$ . Let  $x_k(t)$  be the instant rate of session  $k$  in time slot  $t$ . Let  $\mathbf{x}(t) = (x_1(t), x_2(t), \dots, x_K(t))$  be the rate vector of all active sessions in time slot  $t$ .

Let each source and sink  $v \in \mathcal{V}$  have a capacity  $C^v$ . It is either the sending or receiving capacity of node  $v$ , when  $v \in \mathcal{V}^S$  or  $v \in \mathcal{V}^R$ , respectively. Let  $\mathcal{K}^v$  denote the set of sessions that node  $v$  participates in. Note that in our model each node  $v$  is either a sink or a source node, but not both.

Therefore  $\mathcal{G}(\mathcal{V}, \mathcal{C}, \mathcal{K}, \mathcal{M})$  characterizes an instance of the bandwidth sharing problem in the lambda-networks.

### C. Rate Allocation Problem

The challenge of bandwidth sharing and rate allocation in a lambda network is how to *efficiently* and *fairly* share the capacity of each source and sink among active sessions. Of course, the allocation algorithm should also be *stable*.

We describe the main bandwidth sharing objectives as follows.

First, the rate allocation (bandwidth sharing) algorithm should efficiently utilize of the capacity of each source and sink while maintaining *feasibility*. Feasibility, simply put, requires that aggregate rate of the sessions for each source and sink does not exceed its capacity, and each session's rate does not exceed its desired rate. Formally, *feasibility* can be defined as follows.

**Definition 1: Feasibility.** A rate vector  $\mathbf{x}$  is *feasible*, when for each node  $v \in \mathcal{V}$ ,

$$\sum_{k \in \mathcal{K}^v} x_k \leq C^v,$$

and for each session  $k \in \mathcal{K}$  we have that  $x_k \leq M_k$ .

Second, the a rate allocation algorithm should treat all active sessions fairly. We adopt *Max-min Fairness* [12] as the criteria, as it is widely-used in wired and wireless networks [34], [42], [41]. Max-min fairness maximizes the rates of the sessions with lower rates, and shares the remaining bandwidth until the network is fully utilizes. Formally, max-min fairness is defined as follows.

**Definition 2: Max-min Fairness.** A feasible rate vector  $\mathbf{x}$  is *max-min fair* if it is not possible to increase the rate of a session  $x_p$  while maintaining feasibility, without reducing the rate of some session  $x_q$  ( $p \neq q$ ) with  $x_p \leq x_q$ . i.e. for any other feasible vector  $y$ ,

$$(\exists p \in \mathcal{K}) y_p > x_p \implies (\exists q \in \mathcal{K}) y_q < x_q < x_p.$$

Third, the distributed rate allocation algorithm should converge rapidly, reaching a unique rate allocation which is max-min fair. Consider under a discrete-time system, in time slot  $t$ , the rate vector  $\mathbf{x}(t)$  is updated as

$$\mathbf{x}(t+1) = f(\mathbf{x}(t)),$$

where  $f(\cdot)$  is the update function. The function  $f(\cdot)$  may depend on session desired rates, the status of each source and sink, etc. For good network behavior, the rate allocation algorithm should be *stable* and *converge*.

**Definition 3: Stability and Convergence.** An algorithm is *stable* if there is a unique equilibrium rate vector  $\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_K^*)$ , s.t.

$$\mathbf{x}^* = f(\mathbf{x}^*).$$

An algorithm *converges*, if when all  $K$  sessions are long-lived, the rate vector  $(x_1, x_2, \dots, x_K)$  over time approaches and achieves the unique equilibrium rate allocation  $\mathbf{x}^*$ , regardless of its initial state, session starting sequence, or other temporal details.

The definition of stability is straightforward. The definition of convergence not only encompasses arbitrary initial state, but also changes in session desired rates during the convergence process. In the following section, we study several global and distributed bandwidth allocation algorithms.

## III. GLOBAL ALGORITHM

A global algorithm is presented to calculate the max-min fair allocation. We show that the result rate vector is the unique max-min fair allocation for the bandwidth sharing problem we defined in the previous section.

### A. Global Algorithm

With global knowledge, max-min fair rates can be calculated in straightforward fashion [12]. At each step the algorithm assigns equal rate shares to sessions at the current bottleneck node. The bottleneck node is the one with with the minimum average rate after its remaining capacity is assigned to the remaining sessions (also known as undetermined sessions). Formally,

**Definition 4: Bottleneck Node.** Given a partially determined rate vector  $\mathbf{x}$ , node  $v \in V$  is called the bottleneck node, if we have *undetermined sessions*  $k \in \{j | x_j = 0, j \in \mathcal{K}\}$ , and remaining available capacity at node  $v$  divided by the number of undetermined sessions,  $x_f$  (also known as the bottleneck rate), satisfies

$$x_f = \frac{C^v - \sum_{i \in \mathcal{K}^v} x_i}{|\{j | x_j = 0, j \in \mathcal{K}^v\}|} = \min_{v' \in V} \frac{C^{v'} - \sum_{i \in \mathcal{K}^{v'}} x_i}{|\{j | x_j = 0, j \in \mathcal{K}^{v'}\}|}, \quad (1)$$

where node  $v' \in V$  such that  $|\{i | x_i = 0, i \in \mathcal{K}^{v'}\}| > 0$ .

Our algorithm extends the classic max-min algorithm [12] by considering the session's desired rate.

Sessions whose desired rate has already been satisfied are considered determined, and should thereby be excluded from the undetermined sessions.

The global algorithm is formalized as follows.

#### Algorithm 1: Global Algorithm

Notations:

$\mathbf{x}^*$ : The result rate vector, and  $\mathbf{x}^* = (x_1^*, \dots, x_K^*)$ .

$b(v)$ : Computes the average rate of undetermined sessions at node  $v \in V$ , denoted by

$$b(v) = \frac{C^v - \sum_{i \in \mathcal{K}^v} x_i^*}{|\{j | x_j^* = 0, j \in \mathcal{K}^v\}|}. \quad (2)$$

If there are no undetermined sessions at node  $v \in V$ , set  $b(v) = +\infty$ .

1. set  $\mathbf{x}^* = \mathbf{0}$ ;

2. **repeat**

3   **if**  $\min_{v \in \mathcal{V}} b(v) < \min_{k \in \mathcal{K}, x_k^* = 0} M_k$

4      $v = \arg \min_{v' \in \mathcal{V}} b(v')$ ;

5      $\forall k \in \mathcal{K}^v$ , if  $x_k^* = 0$  then let  $x_k^* = b(v)$ ;

6   **else**

7      $k = \arg \min_{k' \in \mathcal{K}, x_{k'}^* = 0} M_{k'}$ ;

8      $x_k^* = M_k$ ;

9   **endif**

10 **until**  $|\{j | x_j^* = 0, j \in \mathcal{K}\}| = 0$ .

Note that there may be multiple bottleneck nodes with the same remaining capacity for any given iteration. In that case, we pick an arbitrary node.

## B. Properties of Global Algorithm

We show that the global algorithm does in fact compute the max-min fair rate allocation.

We show in Proposition 1 that the rate vector  $\mathbf{x}^*$  calculated by Algorithm 1 is the unique max-min fair rate allocation.

*Proposition 1:* 1. Algorithm 1 (Global Algorithm) terminates within  $K$  iterations.

2. The rate allocation  $\mathbf{x}^*$  computed by Algorithm 1 is max-min fair and unique.

*The proofs of the Proposition 1 are straightforward and included in Appendix A.*

While the global algorithm is simple and has numerous desirable properties, it has a number of important limitations which we relax in subsequent sections. First, the global algorithm requires global information for the entire network to set the allocation for each node and session. Second, the global algorithm requires desired session rates, which is typically difficult to extract from applications. We consider a distributed algorithm which overcomes these limitations in the following section.

## IV. DISTRIBUTED ALGORITHM

We present a distributed algorithm which overcomes the major limitations of the global algorithm. The distributed algorithm requires only local node and actual session behavior to produce max-min fair rate allocation for arbitrary workloads.

### A. Distributed Rate Allocation and Adaptation

Consider a discrete-time view where at time  $t$  each node  $v \in V$  knows the current sending rate  $x_k(t)$  of each session  $k \in \mathcal{K}^v$ , but no knowledge about its desired rate  $M_k$ . As a result, a source and sink cannot tell whether a session with small rate in time slot  $t$  is constrained by the session's desired rate or the capacity of the peer node (complementary source or sink). This means that a static rate allocation scheme (without rate adaptation) has no means to determine or adapt to changes in the desired session rate.

To meet these needs, we formulate the rate allocation problem as a feedback-based rate adaptation problem with coordinated rate allocation at each source and sink node. Specifically, in each time slot, all source and sink nodes operate autonomously. In time slot  $t$  each sink node allocates its capacity and sets expected rates  $\hat{x}_k^r$  for each of its sessions, using the status of all sessions terminating at the same sink:

$$\hat{\mathbf{x}}^r(t+1) = g(\mathbf{x}(t)).$$

The expected rate for each session  $k$ ,  $\hat{x}_k^r(t+1)$  at the sink is then fed back to the source. With this information, each source node allocates its capacity to each session in similar fashion. Each source calculates the expected rate  $\hat{x}_k^s$  for each of its sessions  $k \in \mathcal{K}^v$ :

$$\hat{\mathbf{x}}^s(t+1) = h(\mathbf{x}(t)).$$

As the network operates, the actual rate of session  $k \in \mathcal{K}$  in time slot  $t+1$  is then determined by the minimum of expected

source rate  $\hat{x}^s(t+1)$ , expected sink rate  $\hat{x}^r(t+1)$ , and the session desired rate  $M_k$ , formally

$$x_k(t+1) = \min\{\hat{x}^s(t+1), \hat{x}^r(t+1), M_k\}. \quad (3)$$

### B. Source and Sink Algorithms

The idea in the distributed algorithm is to adapt each session's current rate towards the max-min fair rate for the session. More specifically, at each step, each node picks the session with the minimum rate among its undetermined sessions and assigns expected rates based on current expected rate and the *target rate*. Intuitively, the *target rate* approximate the max-min fair allocation of the remaining capacity over the undetermined sessions.

Because expected session rates are calculated one at a time, target rates may vary across even similar sessions. If the current session rate is less than the target rate  $x_f$ , we adapt it toward the target rate, as

$$\hat{x}_k(t+1) = x_k(t) + \alpha(x_f - x_k(t)),$$

where  $\alpha \in (0, 1)$  is the adaptation step size ( $0 < \alpha < 1$ ), and  $x_f$  is the current remaining capacity over the number of undetermined sessions:

$$x_f = \frac{C^v - \sum_{i \in \mathcal{K}^v, \hat{x}_i(t+1) \neq 0} x_i(t)}{|\{j | \hat{x}_j(t+1) = 0, j \in \mathcal{K}^v\}|}. \quad (4)$$

We update  $x_f$  each iteration to reflect the changes in remaining capacity. When the minimum session rate of all undetermined sessions is greater than the *target rate*  $x_f$ , we distribute the remaining bandwidth evenly to the sessions, assigning each of them the *target rate*, as

$$\hat{x}_k(t+1) = x_f, \text{ if } x_k(t) \leq x_f.$$

In general, we try to maximize the session rates for sessions with lower desired rates by considering them earlier. Sessions with minimum rates are considered first, and their rates continue to increase as long as their rate is still minimum. It only stops increasing when it is restricted by its peer node or desired rate, or reaches the *target rate*. In the last case, the target rate is actually  $C^v / |\mathcal{K}^v|$ , a fair share of bandwidth when no sessions are limited by their desired rates or the peer nodes.

More formal descriptions of the sink and source node algorithms is as follows.

#### Algorithm 2: (Sink Algorithm)

**Input:**  $\mathbf{x}(t)$ ,  $\mathcal{K}$ ,  $V^R$ . **Output:**  $\hat{\mathbf{x}}^r(t+1)$ .

1. **foreach**  $v$ ,  $v \in V^R$ ,
2. Let  $\hat{x}_k^r(t+1) = 0$ ,  $\forall k \in \mathcal{K}^v$ .
3. **repeat**
4.  $x_f = \frac{C^v - \sum_{k, k \in \mathcal{K}^v, \hat{x}_k^r(t+1) \neq 0} x_k(t)}{|\{k | \hat{x}_k^r(t+1) = 0, k \in \mathcal{K}^v\}|}$
5.  $k = \arg \min_j x_j(t)$ , for all  $j \in \mathcal{K}^v$  and  $\hat{x}_j^r(t+1) \neq 0$ ;
6. **if**  $x_k(t) < x_f$
7.  $\Delta x = \alpha \times (x_f - x_k(t))$ ;
8.  $\Delta x = \min(\Delta x, \alpha \cdot x_f)$ ;
9.  $\Delta x = \max(\Delta x, \beta \cdot \cdot \times x_f)$ ;

10.  $\hat{x}_k^r(t+1) = x_k(t) + \Delta x;$
11. **endif**
12. **until**  $\{j | \hat{x}_j^r(t+1) = 0, j \in \mathcal{K}^v\} = 0$  **or**  $x_k(t) \geq x_f$
13. **for all**  $k \in \mathcal{K}^v$
14. **if**  $\hat{x}_k^r(t+1) = 0$
15.  $\hat{x}_k^r(t+1) = x_f;$
16. **endif**
17. **endfor**

The sink algorithm includes upper and lower bounds on the adaptation steps ( $\alpha, \beta$  between 0 and 1). An upper bound ensures that expected rate does not change too quickly, and limits overshoot. A lower bound supports faster convergence when current rates are close to the *target rate*.

**Algorithm 3: Source Algorithm** The source algorithm calculates source expected rates in fashion similar to the sink algorithm. We use a different notation  $\hat{x}_k^s$  is used to denote the expected rate at the source.

Therefore the actual sending rate of a session  $k \in \mathcal{K}$  in time slot  $t+1$  is the minimum among session  $k$ 's desired rate, sink expected rate, and source expected rate, defined as

$$x_k(t+1) = \min\{\hat{x}_k^s(t+1), \hat{x}_k^r(t+1), M_k\}. \quad (5)$$

Intuitively, for flows with low session rates we have  $x_f > x_k$ , meaning that the *target rate* is higher than the current session, giving  $\hat{x} > x_k$ . As a result, sessions with low rates have room to increase if so desired by the application. If a session's  $M_k$  increases beyond the target rate, and there is available capacity, it will be allocated more rate.

### C. Stability and Convergence

We study the stability and convergence properties of the distributed rate allocation algorithm. In particular, we show that (a) the distributed algorithm has the max-min fair allocation  $\mathbf{x}^*$  in the global algorithm as an equilibrium point and (b) the distributed algorithm causes the session rates to converge, regardless of their initial states. Formally,

**Proposition 2: Stability.** The max-min fair rate vector  $x^*$  calculated by the global algorithm is an equilibrium point for the distributed algorithm, i.e. if in time slot  $t$ ,

$$\mathbf{x}(t) = \mathbf{x}^*,$$

then we have that

$$\mathbf{x}(t+1) = \mathbf{x}^*.$$

The proof for Proposition 2 is included in Appendix B. Proposition 2 essentially says that the set of equilibrium rates for the distributed algorithm are the same as those calculated by the global algorithm. Next, in Proposition 3, we show that under the distributed rate allocation algorithm the network converges to this equilibrium rate allocation.

**Proposition 3: (Convergence)** For any initial rate vector  $\mathbf{x}(0)$  (at time  $t = 0$ ), under the distributed rate allocation algorithm the rate vector  $\mathbf{x}$  converges to the final rate vector  $\mathbf{x}^*$  computed by the global algorithm in a finite number of steps, i.e. there exists time  $t_0 > 0$  and for any time slot  $t > t_0$ , we have that

$$\mathbf{x}(t) = \mathbf{x}^*.$$

We provide a proof of Proposition 3 in Appendix C. The key insight is that the leximin ordering (as defined in Appendix A) of the rates can be proven to move toward the equilibrium rate vector.

Combining Proposition 2 and 3 shows directly that the rate vector  $\mathbf{x}^*$  is the unique equilibrium point for the distributed algorithm. And, this rate vector is the max-min rate allocation computed by the global algorithm.

**Corollary 1: (Capacity Utilization)** Except for the initial state ( $t = 0$ ), the maximum aggregate rate at any node  $v \in \mathcal{V}$  in time slot  $t \geq 1$  is limited by  $(1 + \alpha)C^v$ , i.e.

$$\sum_{k \in \mathcal{K}^v} x_k(t) \leq (1 + \alpha)C^v.$$

*Proof:* This can be directly got from the following two facts:

- (a) For sessions with lower rates than the *fair rate*, the rate increase is no more than  $\alpha \cdot C^v / |\mathcal{K}^v|$ .
- (b) For sessions with higher rates than the *fair rate*, they will be assigned with fair rate, with leads to the aggregate rate when not counting the increment in (a) equal to the node capacity  $C^v$ . ■

In summary, we have shown that there exists a single max-min rate allocation for a network, and that rate allocation is computed by both the global algorithm and the distributed rate allocation algorithm. Thus, the distributed rate allocation algorithm is fair. Further, we have shown that the distributed rate allocation algorithm converges within a finite number of steps, so the distributed rate allocation mechanism is stable.

### D. Extension to the Asynchronous Case

We extend the proposed distributed rate allocation algorithm from synchronous, discrete time asynchronous, continuous time. Each session can have variable round-trip times, and each source and sink node can make rate allocation decisions asynchronously. More specifically we make the following assumptions:

- Time is continuous, and each node runs the source/sink algorithm at each control interval,
- Sessions have varied round-trip delay and information propagates from sink to source with latency of one-half RTT,
- Each source or sink performs its local rate allocation algorithm asynchronously, and we explore various, fixed control interval;
- The step-size parameters,  $\alpha$  and  $\beta$ , for adjusting rate remain fixed and are the same for each source and sink;

The major part of the algorithm remains unchanged. However each node now make its own rate allocation decision asynchronously based on information which is delayed by round-trip latencies. We will show through simulations that the same stability and convergence results still hold in such asynchronous settings.

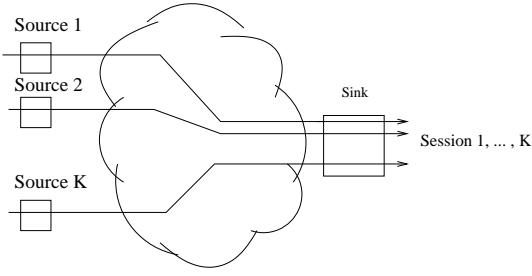


Fig. 1. A single sink, multiple source Topology

## V. EMPIRICAL STUDIES: THE SYNCHRONIZED CASE

In this section, we illustrate the stability and convergence properties of our distributed algorithm, exploring small networks and large with a variety of desired rates, traffic patterns, and dynamic desired rate changes.

For session rate behavior, we plot session rates for particular flows versus time. However, this visualization quickly becomes ineffective for large numbers of flows, as the plots become unreadable. To show the algorithms convergence properties for large networks, we define a distance metric between the current rate vector  $\mathbf{x}(t)$  and the max-min fair equilibrium  $\mathbf{x}^*$  (calculated for example by the global algorithm). Experimenting with various distance metrics, we find one that works well is the 2-norm distance, defined as:

$$D(t) = \left[ \sum_{i=1}^K (x_i(t) - x_i^*)^2 \right]^{1/2}.$$

Therefore in the equilibrium state, we have  $D = 0$ .

### A. Single Sink, Multiple Source

We first explore the dynamics of our distributed algorithm and its transient behavior with a case study of 5 sources and single sink. As shown in Figure 1, there are 5 sources, each with a session terminating at a single sink. The five sessions are assigned rates 0.1, 0.2, 0.3, 0.4 and 0.5, respectively, and the capacity of each source and sink is 1. We use distributed algorithm parameters of  $\alpha = 0.1$  and  $\beta = 0.1$ . In this case, the session desired rates and the receiver are the bottlenecks.

For the experiment, all sessions start at  $t = 0$  with initial rates  $(0, 0, 0, 0, 0)$ , and different desired rates. The trajectories of all sessions and the aggregate rate at the sink are depicted in Figure 2. Within 20 time periods, all sessions have converged to the equilibrium rates. To explore dynamic adaptation, at time  $t = 50$ , session 1's desired rate is increased from 0.1 to 1. The distributed algorithm reacts to this change promptly, and the session rates converge to a new equilibrium rate allocation within a short period with each session gets the same rate 0.2. To explore session termination, we terminate session 5, 4, 3, 2 at times  $t = 100, 150, 200, 250$ , respectively. Again, the simulation results show that the rates of remaining sessions are able to adapt quickly to the changes, filling up the vacated capacity, and converging to a new equilibrium rate allocation. We list the equilibrium rate allocations in various time slots in Table I.

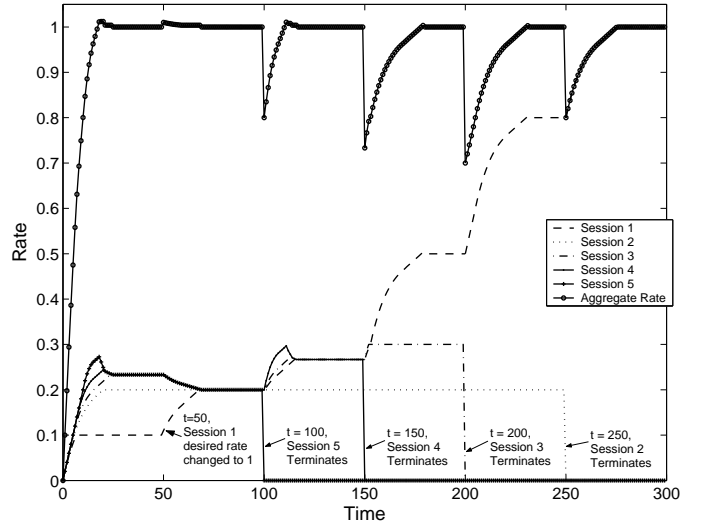


Fig. 2. Session rate trajectories for each of five sessions in a scenario with single sink and five sources. The five sessions are assigned rates 0.1, 0.2, 0.3, 0.4 and 0.5. They terminate in different time slots.

TABLE I  
EQUILIBRIUM RATES OF FIVE SESSIONS AND THE RATE AT THE SINK

Time	0-50	51-100	101-150	151-200	201-250	251-
Session 1	0.1	0.2	0.267	0.5	0.8	1
Session 2	0.2	0.2	0.2	0.2	0.2	0
Session 3	0.233	0.2	0.267	0.3	0	0
Session 4	0.233	0.2	0.267	0	0	0
Session 5	0.233	0.2	0	0	0	0
Sink Rate	1	1	1	1	1	1

Next, we study the importance of distributed algorithm parameters  $\alpha$  and  $\beta$  on convergence. For the same 5-to-1 scenario, we plot the distance  $D$  of the current rate vector from the equilibrium over time, varying  $\alpha$  from 0.05 to 0.2, with fixed  $\beta = 0.1$ . (see Figure 3) Our results show that larger  $\alpha$  values can enable more rapid convergence. However as we discuss in Corollary 1 that a large  $\alpha$  may lead to higher overshoot in some cases. For the same 5-to-1 scenario, we plot the distance  $D$  for various values of  $\beta$  (see Figure 4). Our results show that larger  $\beta$  value help speed convergence when the system is close to the equilibrium point.

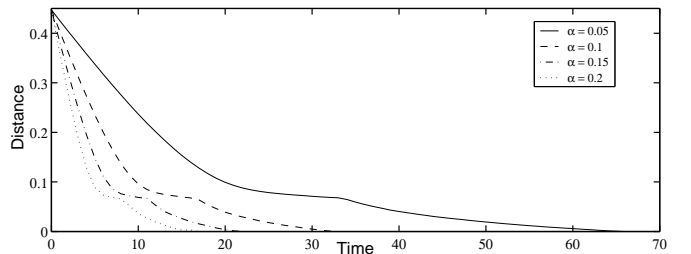


Fig. 3. The five-source, single-sink scenario: the 2-norm distance between current rates and equilibrium rates in the cases of different parameter  $\alpha$ .

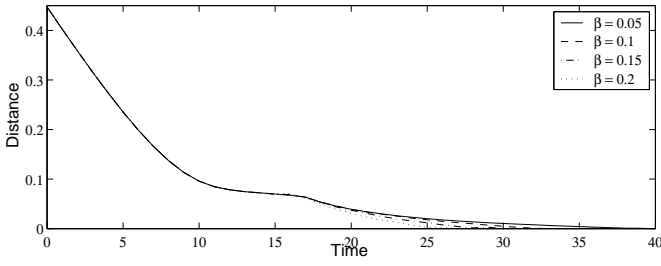


Fig. 4. The five-source, single-sink scenario: the 2-norm distance between current rates and equilibrium rates in the cases of different parameter  $\beta$ .

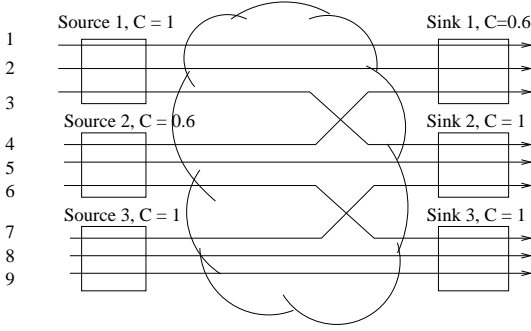


Fig. 5. A topology with three sources and three sinks.

### B. Multiple Sink, Multiple Source

We now consider a scenario with three sources and three sinks. Each source and sink participates in three sessions with connections and node capacities as shown in Figure 5. Note that source 2 and sink 1 have lower capacities than the rest of end nodes. All sessions have the same desired rate of 1. All sessions start at time  $t = 0$  with different initial rates (0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18). We use distributed algorithm parameters  $\alpha = 0.1$  and  $\beta = 0.1$ . We show the trajectories of each session and the aggregate rate at each source node in Figure 6.

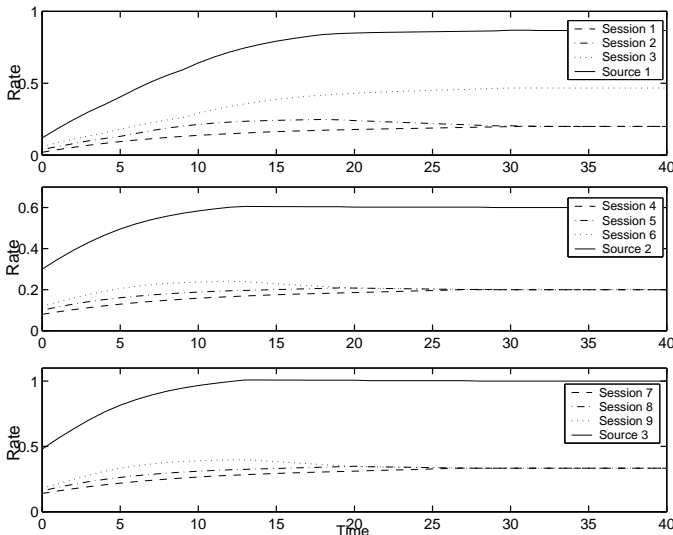


Fig. 6. The three-source, three-sink scenario: the trajectories of each session and the aggregate rate at three source nodes.

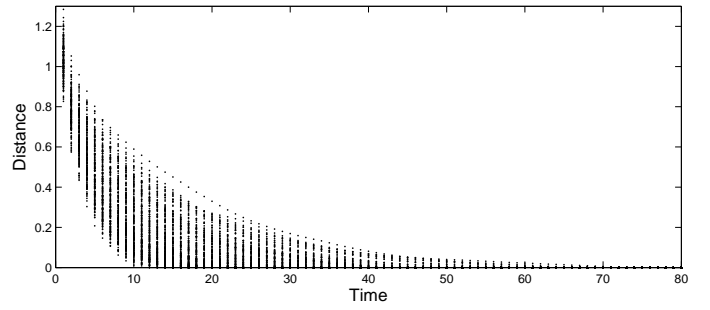


Fig. 7. The 32-node network scenario: network 2-norm distance for 200 test cases

We observe from Figure 6 that the rate of each session as well as the aggregate rate at each source converges to the equilibrium point within 30 time slots.

### C. Larger Networks

While lambda networks are likely to be much smaller than the internet, the scalability properties of our distributed rate allocation algorithm are still of interest. We evaluate the convergence properties of the distributed algorithm over a range of network sizes with random traffic patterns as described below. Networks of 32, 128, and 1024 end nodes are used. Each end node has unit capacity. In each scenario, half of the nodes are sources and the other half are sinks. Let each source participate in 4 sessions with randomly picked sinks. We generate up to 200 test cases for each of the three scenarios with randomly picked parameters following uniform distributions:

- Step size  $\alpha$ : between 0.05 and 0.25;
- Step size  $\beta$ : between 0.05 and 0.25;
- Desired rates  $M_k$ : between 0.1 and 0.5;
- Initial rates  $x_k(0)$ : between 0 and  $M_k$ .

Figure 7, 8, and 9 show our simulation results, plotting the 2-norm distance between current rates and the equilibrium rates over time of the 50 testing cases for the three scenarios of 32, 128 and 1024 nodes. We see that although the initial distance varies due to different number of active sessions and initial state, the distributed algorithm causes the distance to decrease quickly, reaching the equilibrium in no more than 80 time slots. Thus our results are promising, illustrating that our distributed algorithm is able to converge in a significant range of networks and conditions.

## VI. EMPIRICAL STUDIES: THE ASYNCHRONOUS CASE

In this section, we study the distributed rate allocation algorithm in asynchronous scenarios. We consider various round-trip delay between each source and sink node, and let each node make rate allocation decisions asynchronously. We conduct ns2 simulations to prove that the stability and convergence properties of our distributed algorithm still hold under such fully distributed settings. In all the experiments, the node capacity is set as 500Mbps and normalized to 1. The control interval at each node and the RTT delay between each source and sink are randomly generated. The ns-2 simulation results are presented in the following subsections.

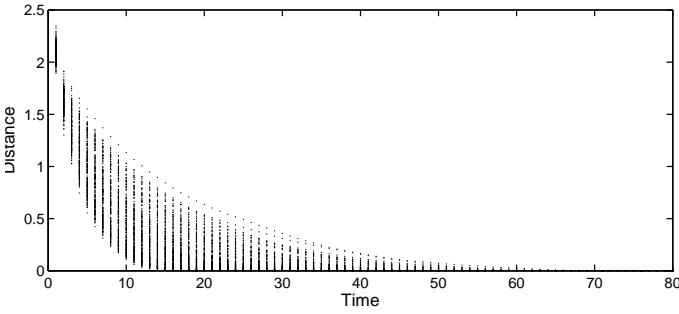


Fig. 8. The 128-node network scenario: 2-norm distance for 200 test cases

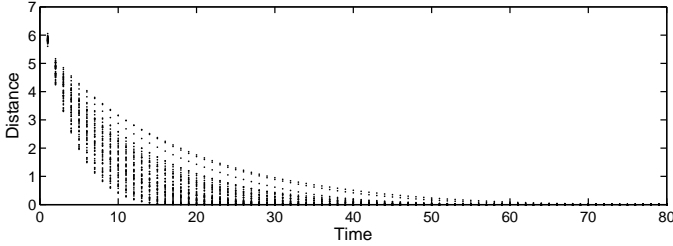


Fig. 9. The 1024-node network scenario: 2-norm distance for 50 test cases

#### A. A Multipoint-to-Point Case

We let five sessions start at time  $t = 0, 8, 16, 24$  and  $32$ , respectively. The RTT is generated between 1ms and 100ms (100ms, 75ms, 50ms, 25ms and 1ms); The control interval of each node is randomly set between 10ms and 100ms. Figure 10 depicts the rate trajectories of each session. We observe that after each new session joins, the system converges to a new equilibrium state, in which all sessions get the same share of the sink bandwidth.

#### B. A Four-to-four Case

We now show the rate trajectories and convergence results when there are four sources and four sinks. We setup one session between each source and sink nodes, thus there are

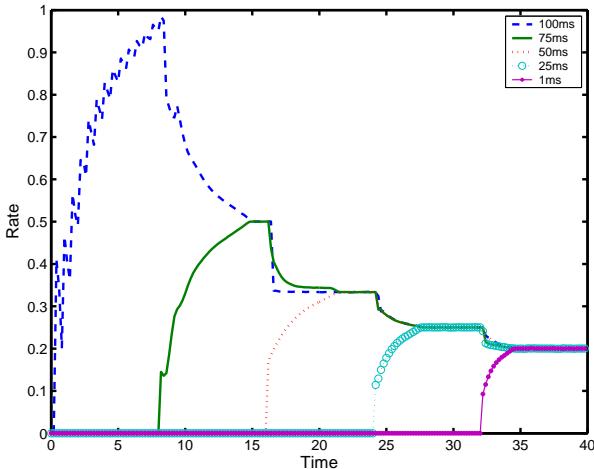


Fig. 10. The trajectories of 5 sessions between five source nodes and one sink node. The 5 sessions start at various times.

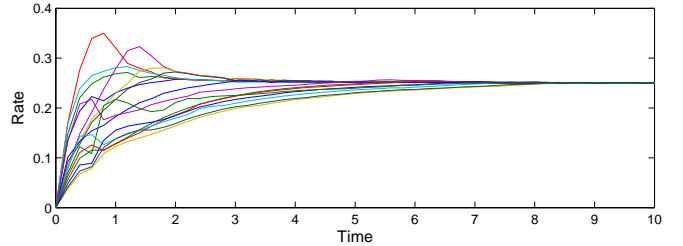
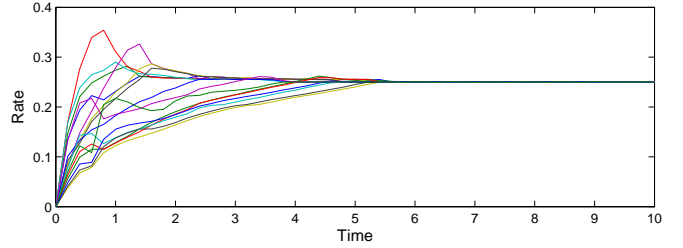


Fig. 11. The trajectories of 16 sessions in a 4-to-4 case. Top: Step size  $\beta = 0.2$ ; Bottom: Step size  $\beta = 0.05$ .

16 sessions in total. The sources and sinks have the same capacity, which is 500Mbps, and normalized to 1. The initial rate of each session is 0. The RTT is randomly generated with uniform distribution between 1ms and 100ms. The control interval of each node varies between 10ms and 100ms. Step size parameters  $\alpha$  is equal to 0.1. The trajectories of each sessions are depicted in Figure 11, with two different values of parameter  $\beta = 0.2$  and 0.05. We see that the rates of all 16 sessions converge to 0.25, and a smaller parameter  $\beta$  leads to longer time for the system to converge.

#### C. Larger Networks

Network sizes considered are 32, 128, and 1024 end nodes. Each end node has the same capacity, which is normalized to 1. In each scenario, half of the nodes are sources and the other half are sinks. We let each source participate in 4 sessions with randomly picked sinks. We generate 30 test cases for each of the three scenarios with randomly picked parameters following uniform distributions:

- Step size  $\alpha$ : between 0.05 and 0.15;
- Step size  $\beta$ : between 0.05 and 0.15;
- RTT: between 1 ms and 100 ms;
- Control interval: between 10 ms and 100 ms.

To verify the convergence properties, we still use the 2-norm distance defined in the previous section. Figure 12, 13 and 14 plot the 2-norm distance between current rates and the equilibrium rates over time of the 30 testing cases for the three scenarios of 32, 128 and 1024 nodes. To compare with the results from the synchronous case, in each figure we also plot three 2-norm distance trajectories in synchronous cases by setting each time slot to be 1ms, 50ms, and 100ms.

From all three figures we see that the distributed algorithm causes the 2-norm distance to decrease quickly from initial states, reaching the equilibrium in no more than 6 seconds. The 2-norm distance is also bounded between two synchronous cases with minimum (1ms) RTT and maximum (100ms) RTT.

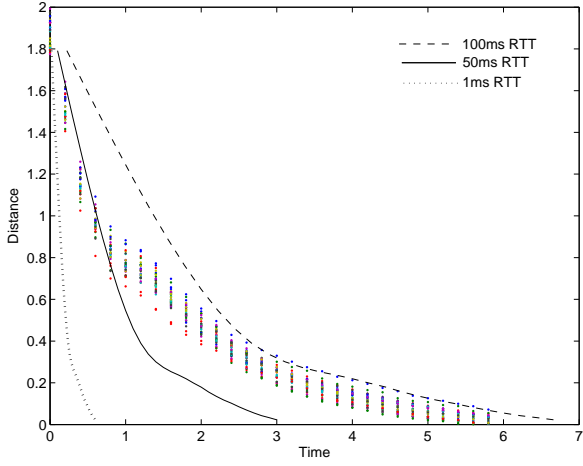


Fig. 12. The 32-node network case: a comparison between the 2-norm distances of 30 test cases in the asynchronous case and the 2-norm distance trajectories in the synchronous case with various time slot size ( 1ms, 50ms and 100ms).

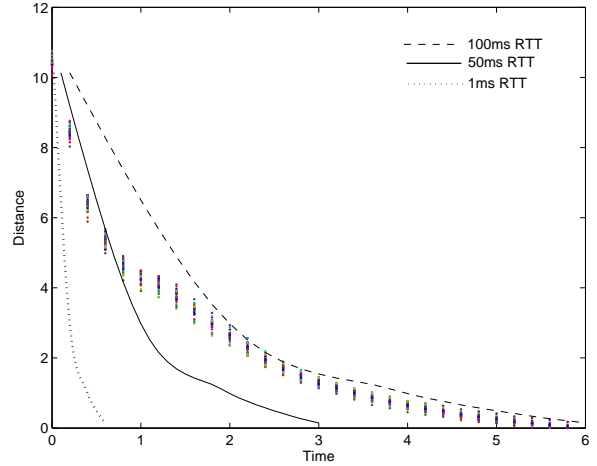


Fig. 14. The 1024-node network case: a comparison between the 2-norm distances of 30 test cases in the asynchronous case and the 2-norm distance trajectories in the synchronous case with various time slot size ( 1ms, 50ms and 100ms)

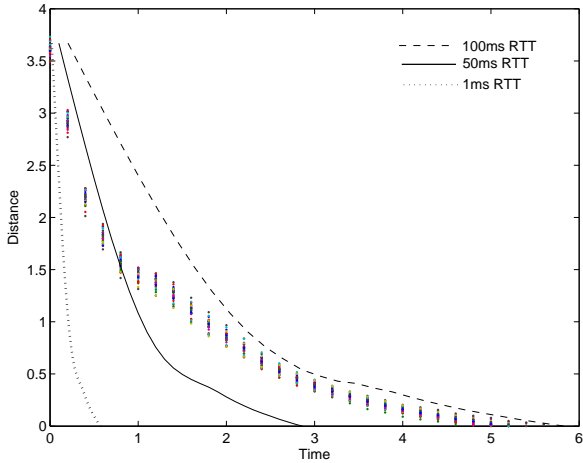


Fig. 13. The 128-node network case: a comparison between the 2-norm distances of 30 test cases in the asynchronous case and the 2-norm distance trajectories in the synchronous case with various time slot size ( 1ms, 50ms and 100ms)

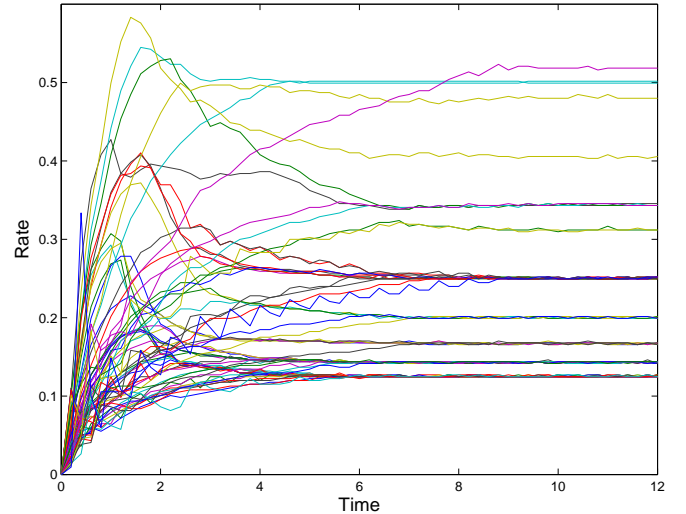


Fig. 15. The trajectories of 64 sessions in an asynchronous 32-node network case

Figure 12 shows the trajectories of all 64 sessions in the case with 32 nodes.

In conclusion, the results from ns-2 simulations verify that the stability and convergence properties of the distributed algorithm we proposed in the previous section still hold under asynchronous distributed settings. As the asynchronous distributed settings we consider share similar scale and round-trip time variance with to the real lambda-networks, the distributed rate allocation algorithm is proved to be a promising approach for such lambda-network environments.

## VII. RELATED WORK

Delivering end-to-end high-performance communication in high bandwidth-delay product networks is a long-standing research challenge in wired, wireless and satellite networks [19], [40]. The commonly used transport protocol, standard TCP (TCP Reno) [10], was designed for shared low-bandwidth

networks. As a result the protocol is inefficient in providing satisfactory performance in high-speed environments. The protocol reacts adversely to the increases in bandwidth and delay [30], and spends too long to recover from packet loss and reach high bandwidth due to the *Additive Increase and Multiplicative Decrease* (AIMD) control law [16].

Different approaches have been used to improve standard TCP and achieve a high performance transmission in fast long-delay networks. First, several approaches preserve the TCP control scheme, but improve the implementation of TCP stack [15], [20], place performance enhancing proxies along long-delay links [13], or use parallel connections [25]. Second, a number of high performance TCP control alternatives have been proposed to improve TCP performance in shared, packet-switched networks (e.g. High Speed TCP [21], Scalable TCP [36], BIC-TCP [49]). Third, a new set of approaches exclude the loss-based schemes used in TCP, thus exploring

the delay-based (e.g. FAST-TCP [33]), and rate-based (e.g. NETBLT [17], RBU DP [26], Tsunami [9], and GTP [48]) data transport protocols.

Rate-based max-min fair bandwidth allocation has been drawn much research attention [27], [14], [28], [43], [32] in the context of available bit rate (ABR) service in the ATM networks. Such schemes allocate the capacities of switches to active sessions in a max-min fair manner. They usually assume infinite session demands, or explicit knowledge regarding the desired rate. Instead our work considers the allocation of endpoint's capacity, and we assume each endpoint has no knowledge about the desired session rate. Our rate allocation and adaptation scheme discovers each session's desired rates continuously.

In contrast to most of the max-min scheme in ATM networks which only allocate rates with limited rate adaptation, XCP [35] lets each router make rate feedback based on available capacity and queue size. This significantly improves the performance over flow transitions. XCP does not conduct rate allocation and adaptation at end nodes. Under XCP, sessions with finite desired rate may receive an arbitrarily small fraction of its max-min allocation in a networked case with multiple sources and multiple sinks [37].

Multipoint-to-point transmission has been proposed for web traffic [24] and content delivery network [44]. There are some research projects sharing the same idea of endpoint based management across flows. In [11], a receiver-side integrated congestion management architecture is proposed, which targets managing traffic across various protocols for real-time traffic. However detailed rate allocation schemes and fairness among flows are not considered. Examples of receiver centric approaches also include [29] for wireless networks. In nearly all cases, these studies focus on networks that are slow relative to the nodes attached to them. In contrast we study the problem of how to allocation the capacity of both sources and sinks among all sessions in the lambda-networks, which has high-speed low-loss core. Furthermore both efficiency and fairness are our objectives for bandwidth sharing.

## VIII. SUMMARY AND FUTURE WORK

We considered the problem of efficient and fair sharing of source and sink capacities among communication sessions networks with plentiful bandwidth in the core (e.g. lambda-networks). The key challenge in such systems can be formulated as a distributed rate allocation and adaptation problem in which the desired session communication rates are unknown. First, we presented a global algorithm which computed the max-min optimal rate allocation based on global information, including desired session rates. We then presented a distributed algorithm which achieves the same end, but without the benefit of global information or desired session rates. Proof that the distributed algorithm converges to the same max-min rate allocation as the global algorithm ensures that good allocations will be achieved. The distributed algorithm is also significantly more flexible because it does not require any "need" information from the applications, rather discovering it implicitly. Finally, we evaluated the distributed algorithm

empirically, demonstrating its stability and rapid convergence in a range of network scenarios in both synchronous and asynchronous settings.

We have identified the following future work. First, we plan to formally prove the stability and convergence properties of distributed algorithm in asynchronous settings, in which each sessions may have varied control decision intervals and variable round-trip delays. Second, we plan to integrate the distributed algorithm into a new version of our Group Transport Protocol [48], and demonstrate the stability and convergence properties on various lambda-network testbeds [45]. Third, as the techniques used in the distributed algorithm may also be applied to the network with internal bottlenecks, it may be interesting to see if the stability and convergence properties of the distributed algorithm we proposed can still hold in such general network environments.

## ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under awards NSF Cooperative Agreement ANI-0225642 (OptIPuter), NSF CCR-0331645 (VGrADS), NSF ACI-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from the UCSD Center for Networked Systems, BigBangwidth, and Fujitsu is also gratefully acknowledged.

## REFERENCES

- [1] Biomedical informatics research network (birn). <http://www.nbirn.net>.
- [2] Bittorrent. <http://bitconjurer.org/BitTorrent/>.
- [3] Canarie. <http://www.canarie.ca>.
- [4] Griphyn project. <http://www.griphyn.org>.
- [5] International virtual data grid laboratory (ivdgl). <http://www.ivdgl.org>.
- [6] Kazaa. <http://www.kazaa.com>.
- [7] Naregi. <http://www.naregi.org>.
- [8] Netherlight. <http://www.netherlight.net>.
- [9] Tsunami. Presented at PFLDnet2003, available from <http://steinbeck.ucs.indiana.edu/mmeiss/papers.html>.
- [10] M. Allman, V. Paxson, and W. Stevens. Rfc 2581: Tcp congestion control. April 1999.
- [11] H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion management architecture for internet hosts. *Proceedings of ACM SIGCOMM 1999*.
- [12] D. P. Bertsekas and R. Gallager. Data networks. Prentice-Hall, Englewood-Cliffs, New Jersey, 1992.
- [13] J. Border, M. Kojo, J. Griner, and G. Montenegro. Performance enhancing proxies. *RFC 3135, Nov 2000*.
- [14] A. Charny, D. D. Clark, and R. Jain. Congestion control with explicit rate indication. *Proc. IEEE International Conference on Communications (ICC'95)*, June 1995.
- [15] J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed TCP. *IEEE Communications Magazine*, 39(4):68–74, 2001.
- [16] D. M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.
- [17] D. D. Clark, M. Lambert, and L. Zhang. Netblt: a high throughput transport protocol. In *Proceedings of ACM SIGCOMM 1987*.
- [18] T. DeFanti, C. Laa, J. Mambretti, K. Neggers, and B. Arnaud. Translight: a global-scale lambda grid for e-science. *Communications of the ACM (CACM)*, 47(11), November 2003.
- [19] A. Falk, T. Faber, J. Bannister, A. Chien, R. Grossman, and J. Leigh. Transport protocols for high performance. *Communications of the ACM (CACM)*, 47(11), November 2003.
- [20] M. Fisk and W. Feng. Dynamic right-sizing in tcp. In *Proc. of the Los Alamos Computer Science Institute Symposium, October 2001*.
- [21] S. Floyd. Highspeed tcp for large congestion windows. *Internet draft*, August 2002.

- [22] S. Floyd, S. Ratnasamy, and S. Shenker. Modifying tcp's congestion control for high speeds. Available from URL <http://www.icir.org/floyd/hstcp.html>.
- [23] T. Greene. Dwdm is the right rx for new york presbyterian hospital. *Network World*, 05/16/05, available at <http://www.networkworld.com/news/2005/051605-presbyterian.html>.
- [24] R. Gupta, M. Chen, S. McCanne, and J. Walrand. Webtp: A receiver-driven web transport protocol. *University of California at Berkeley Technical Report*, 1998.
- [25] T. J. Hacker, B. D. Noble, and B. D. Athey. Improving throughput and maintaining fairness using parallel tcp. In *proceedings of IEEE INFOCOM 2004, Hongkong, March 2004*.
- [26] E. He, J. Leigh, O. Yu, and T. DeFanti. Reliable blast udp: predictable high performance bulk data transferrbudp. *IEEE Cluster Computing*, 2002, p. 317.
- [27] Y. Hou, B. Li, S. Panwar, and H.Tzeng. On network bandwidth allocation policies and feedback control algorithms for packet networks. *Computer Networks*, vol.34, pp.481-501, 2000.
- [28] Y. T. Hou, H. Tzeng, S. S. Panwar, and V. P. Kumar. A generalized max-min rate allocation policy and its distributed implementation using the abr flow control mechanism. *Proceedings of IEEE INFOCOM 1998*, pp.1366-1375, San Francisco, CA, 1998.
- [29] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. *Proceedings of the 9th annual international conference on Mobile computing and networking*, 2003: p. 1-15.
- [30] B. Jacobson. Tcp extensions for high performance. *RFC 1323*, May 1992.
- [31] V. Jacobson. Congestion avoidance and control. *Proceedings of ACM SIGCOMM 1988 pp.314-329*, August 1988.
- [32] R. Jain, S. Kalyanaraman, and R. Viswanathan. The osu scheme for congestion avoidance in atm networks: Lessons learnt and extensions. *Performance Evaluation Journal*, Vol. 31, Dec 1997.
- [33] C. Jin, D. Wei, and S. Low. Fast tcp: Motivation, architecture, algorithms, and performance. In *proceedings of IEEE INFOCOM 2004, Hongkong, March 2004*.
- [34] P. Karbhari, E. Zegura, and M. Ammar. Multipoint-to-point session fairness in the internet. *Proceedings of IEEE INFOCOM 2003*.
- [35] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for high bandwidth delay product network. *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, Aug 2002.
- [36] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks, Feb 2003.
- [37] S. H. Low, L. L. H. Andrew, and B. P. Wyrowski. Understanding xcp: Equilibrium and fairness. *Proc. IEEE Infocom 2005, Miami, FL, March 2005*.
- [38] P. Mehra, A. Zakhor, and C. Vleeschouwer. Receiver-driven bandwidth sharing for tcp. *Proceedings of IEEE INFOCOM 2003*.
- [39] C. Metz. The bright side of dark fiber optics. *PC Magazine*, available at <http://www.pcmag.com/article2/0,1759,1813381,00.asp>.
- [40] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. Long thin networks. *Jan 2000*.
- [41] B. Radunovic and J. L. Boudec. A unified framework for max-min and min-max fairness with applications. *Proceedings of 40th Annual Allerton Conference on Communication, Control, and Computing*, Allerton, IL, October 2002.
- [42] B. Radunovic and J.-Y. L. Boudec. Rate performance objectives of multi-hop wireless. *Proc. IEEE Infocom 2005, Miami, FL, March 2005*.
- [43] S. H. Rhee and T. Konstantopoulos. Achieving max-min fairness by decentralization for the abr traffic control in atm networks. *IEICE Trans. Commun.*, Vol. E84-B, No.8, August.
- [44] P. Rodriguez and E. Biersack. Dynamic parallel access to replicated content in the internet. *IEEE/ACM Transactions on Networking*, 2002. 10(4): p. 455-465.
- [45] L. Smarr, A. A. Chien, T. DeFanti, J. Leigh, and P. Papadopoulos. The optiputer. *Communications of the ACM (CACM)*, 47(11), November 2003.
- [46] N. Taesombut and A. A. Chien. Distributed virtual computer(dvc): Simplifying the development of high performance grid applications. In *Proceedings of the Workshop on Grids and Advanced Networks (GAN 04)*, April 2004, Chicago, Illinois.
- [47] E. Weigle and A. A. Chien. The composite endpoint protocol (cep): Scalable endpoints for terabit flows. *Proceedings of CCGrid 2005*.
- [48] X. Wu and A. Chien. Gtp: Group transport protocol for lambda-grid. *Proceedings of CCGrid 2004*.
- [49] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast long-distance networks. In *proceedings of IEEE INFOCOM 2004, Hongkong, March 2004*.

## APPENDIX

## A. Proof of Proposition 1

Before proving Proposition 1 we introduce two definitions: *leximin mapping* and *leximin ordering*.

**Definition 5: (Leximin Mapping).** The leximin order mapping  $\tau : R^N \rightarrow R^N$  is the mapping that sorts vector  $\mathbf{x}$  in non-decreasing order. We have that

$$\tau(\mathbf{x}) = (x_{(1)}, \dots, x_{(n)}),$$

where  $x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(n)}$ .

We let  $\tau_m(\mathbf{x})$  denote the vector of first  $m$  elements from  $\tau(\mathbf{x})$ , which is  $(x_{(1)}, \dots, x_{(m)})$ . We define the leximin reverse mapping function  $\phi(\cdot)$  as

$$\phi(x_{(j)}) = i,$$

where  $x_i$  corresponds to  $x_{(j)}$  after leximin mapping.

In the following, we refer  $x_{(i)}$  to the  $i$ th element of the leximin mapping of vector  $\mathbf{x}$ .

**Definition 6: (Leximin Ordering)** Given that  $(x_{(1)}, \dots, x_{(n)})$  and  $(y_{(1)}, \dots, y_{(n)})$  are the leximin mappings of  $\mathbf{x}$  and  $\mathbf{y}$ , we define the lexicographic ordering of  $\mathbf{x}$  and  $\mathbf{y}$  as  $\mathbf{x} \succ^L \mathbf{y}$ , if and only if  $(\exists i > 0)x_{(i)} > y_{(i)}$  and  $(\forall 0 < j < i)x_{(j)} = y_{(j)}$ . We also define  $\mathbf{x} \succeq^L \mathbf{y}$ , if and only if  $\mathbf{x} \succ^L \mathbf{y}$  or  $\mathbf{x} = \mathbf{y}$ ; and  $\mathbf{x} \prec^L \mathbf{y}$ , if and only if  $\mathbf{y} \succ^L \mathbf{x}$ .

We let  $\equiv_n^L$  denote a relation between two vectors  $\mathbf{x}$  and  $\mathbf{y}$ , such that  $\mathbf{x} \equiv_n^L \mathbf{y}$  if and only if  $\tau_n(\mathbf{x}) = \tau_n(\mathbf{y})$ , and  $x_{\phi(x_{(i)})} = y_{\phi(x_{(i)})}$ ,  $i = 1..n$ .

It is shown in [41] that if a max-min fair vector exists, then it is the unique leximin maximal vector. Thus the existence of a max-min fair vector implies the uniqueness of a leximin maximum. Now we provide a proof to Proposition 1.

*Proof:* (Proposition 1.) As in the global algorithm (Algorithm 1) at least one undetermined session is assigned with a rate in each iteration, the total number of iterations required for Algorithm 1 is then no more than  $K$  iterations, where  $K$  is the total number of active sessions. This proves the correctness of the first statement.

The Proposition 3 in [41] shows that if a max-min fair vector exists, it is the unique leximin maximal vector. As it is straight forward that the calculated rate vector  $\mathbf{x}^*$  is feasible, we are to prove by contradiction that if there is another feasible rate vector  $\mathbf{y}$ , and  $\mathbf{y} \succ^L \mathbf{x}^*$ , then  $\mathbf{y}$  is not feasible.

Suppose there exists  $m \in [1, \dots, K]$ , such that  $y_{(m)} > x_{(m)}^*$  and  $\tau_{m-1}(\mathbf{x}^*) = \tau_{m-1}(\mathbf{y})$ .

We firstly show by induction on  $p = 1..m - 1$  that

$$x_{\phi(x_{(p)}^*)}^* = y_{\phi(x_{(p)}^*)}.$$

(1) When  $p = 1$ , let  $i = x_{(1)}^*$ . As  $y_{(1)} = x_{(1)}^*$ , for all  $k = 1..K$ , we have

$$x_i = x_{(1)}^* \leq y_k.$$

Therefore either  $x_i^* = y_i$  or  $x_i^* < y_i$ .

We consider the case when  $x_i^* < y_i$ . From global algorithm we know that  $x_i^* = \min\{M_i, b(v)\}$ , where  $v$  is the corresponding node that has the smallest  $b(v)$  in the  $i$ th iteration.

If  $x_i^* = M_i$  then  $y_i > M_i$ , which violates the feasibility. Now we consider  $x_i^* = b(v)$ . The global algorithm implies that for any  $j$ ,  $0 < j < i$ ,  $x_j^* \geq x_i^*$ . Then we have

$$x_i^* = b(v) \geq C^v / |\mathcal{K}^v|.$$

And as for any  $j$  ( $1 \leq j \leq K$ ),  $y_j > x_1^*$ , then for the same node  $v$  we have that

$$\sum_{j,j \in \mathcal{K}^v} y_j \geq y_{(1)} \cdot |\mathcal{K}^v| > x_1^* \cdot |\mathcal{K}^v| \geq C^v,$$

which also violates the feasibility condition.

As it is impossible to have  $x_i^* < y_i$  for a feasible  $\mathbf{y}$ , we obtain that  $x_i^* = y_i$ .

(2) Suppose that  $\tau_p(\mathbf{x}) = \tau_p(\mathbf{y})$ , and for all  $j' \in [1, p-1]$ , we have

$$x_{\phi(x_{(j')})}^* = y_{\phi(x_{(j')})}. \quad (6)$$

We show that  $x_{\phi(x_{(p)})}^* = y_{\phi(x_{(p)})}$ .

Let  $i = \phi(x_{(p)})$ . We show by contradiction that both Case (2.a)  $y_i < x_i$  and Case (2.b)  $y_i > x_i$  are not possible.

Case (2.a): If  $y_i < x_i$  then  $y_i < y_{(p)}$ . This implies that there exists  $q > 0$  and  $q \neq p$ , s.t.  $i = \phi(y_{(q)})$ . Then we have  $q < p$ . However from Eq. 7 we obtain that  $y_{\phi(x_{(1)})} \dots y_{\phi(x_{(p-1)})}$  are  $p-1$  unique ones to  $y_{(q)}$  and are all less than  $y_{(p)}$ . As a consequence there are  $p$  elements that are less than  $y_{(p)}$ , which is impossible according to the definition of lexmin mapping.

Case (2.b): If  $y_i > x_i$ , then from the global algorithm we know that

$$x_i^* = \min\{M_i, b(v)\},$$

where  $v$  is the corresponding node that has the smallest  $b(v)$  in the  $i$ th iteration. If  $x_i^* = M_k$ , then we have  $y_i > M_i$  which violates the feasibility condition. If  $x_i^* = b(v)$ , then we have

$$\begin{aligned} \sum_{j,j \in \mathcal{K}^v} y_{(j)} &= \sum_{j,j \in \mathcal{K}^v, x_{(j)}^* \neq 0} y_{(j)} + \sum_{j,j \in \mathcal{K}^v, x_{(j)}^* = 0} y_{(j)} \\ &\geq \left( \sum_{j,j \in \mathcal{K}^v, x_{(j)}^* \neq 0} y_{(j)} \right) + y_{(j)} \cdot |\{j | x_{(j)}^* = 0, j \in \mathcal{K}^v\}| \\ &> \left( \sum_{j,j \in \mathcal{K}^v, x_{(j)}^* \neq 0} x_{(j)}^* \right) + x_p^* \cdot |\{j | x_{(j)}^* = 0, j \in \mathcal{K}^v\}| \geq C, \quad (7) \end{aligned}$$

which also violates the feasibility condition.

By induction from (1) and (2), for  $p = 1..m-1$ , we have

$$x_{\phi(x_{(p)})}^* = y_{\phi(x_{(p)})}.$$

It can be proved by applying similar arguments as above that if  $y_{(m)} > x_{(m)}^*$  then  $\mathbf{y}$  is not feasible. This proves the second statement.

The third statement directly follows from the Proposition 1 in [41] (If a max-min fair vector exists on a set, then it is unique). ■

## B. Proof of Proposition 2

*Proof:* We first to show that if  $\mathbf{x}(t)$  is feasible, then for any session  $k \in [1, ..K]$ , we have

$$\hat{x}_k^s(t+1) \geq x_k(t), \text{ and } \hat{x}_k^r(t+1) \geq x_k(t). \quad (8)$$

Recall that in the distributed algorithm the expected rate  $\hat{x}_k$  of session  $k$  depends on the *target rate*  $x_f$  and its current rate  $x_k(t)$ . And only when  $x_f$  is less than the current session rate  $x_k(t)$ , the expected rate will be set to  $x_f$ , thus smaller than the current rate:

$$x_k(t+1) = x_f < x_k(t).$$

Eq. 4 implies that this only happens when the remaining capacity is less than the aggregate current rate of all undetermined sessions. In this case we have

$$\sum x_k(t) > C.$$

However this also implies that the rate vector  $\mathbf{x}$  in time slot  $t$  is not feasible. Therefore given any feasible rate vector  $\mathbf{x}$  in time slot  $t$ , the expected rates of each session  $k$  in time slot  $t+1$  satisfies Eq. 8.

Recall that the rate of each session  $k$  in time slot  $t$  is determined by

$$x_k(t) = \min\{\hat{x}_k^s(t), \hat{x}_k^r(t), M_k\}.$$

We consider the following two separate cases: (1)  $x_k(t) = M_k$ , (2)  $x_k(t) < M_k$  and  $x_k(t) = \hat{x}_k^s(t)$  or  $x_k(t) = \hat{x}_k^r(t)$ .

Case (1):  $x_k(t) = M_k$ . We can get by Eq. 8 that

$$x_k(t+1) = \min\{\hat{x}_k^r(t+1), \hat{x}_k^s(t+1)\} \geq x_k(t).$$

As it is always true that  $x_k(t+1) \leq M_k = x_k(t)$ , it directly follows that  $x_k(t+1) = x_k(t)$ .

Case (2):  $x_k(t) < M_k$  and  $x_k(t) = \hat{x}_k^s(t)$  or  $x_k(t) = \hat{x}_k^r(t)$ . This implies that session  $k$  is constrained by its source/sink node but not its desired rate. Let  $v$  be the corresponding node on which  $x_k(t)$  equals to its expected rate in time slot  $t$ . This also implies that  $x_k(t) = x_k^* = x_f$ . We notice the following facts from the global algorithm: (a) As session  $k$  is not restricted by its desired rate, the aggregate rate at node  $v$  equals to  $C^v$ ; (b) both  $x_k^*$  and  $x_k(t)$  equal to the highest session rate among all sessions at node  $v$ . Then from the calculation of  $x_f$  (Eq. 4) in the distributed algorithm we can obtain in time slot  $t+1$ :

$$\begin{aligned} x_f &= \frac{C^v - \sum_{i \in \mathcal{K}^v, \hat{x}_i(t+1) \neq 0} x_i(t)}{|\{j | \hat{x}_j(t+1) = 0, j \in \mathcal{K}^v\}|} \\ &= \frac{C^v - \sum_{i \in \mathcal{K}^v, x_i(t) < x_k(t)} x_i(t)}{|\{j | x_j(t) \geq x_k(t), j \in \mathcal{K}^v\}|} \\ &= \frac{C^v - \sum_{i \in \mathcal{K}^v, x_i^* < x_k^*} x_i^*}{|\{j | x_j^* \geq x_k^*, j \in \mathcal{K}^v\}|} = x_k^*. \end{aligned}$$

Thus we have

$$\hat{x}_k(t+1) = x_k(t),$$

and  $x_k(t+1) = x_k(t)$ .

Combining Cases (1) and (2), we have shown that the rate vector calculated by the global algorithm is an equilibrium

point for the distributed algorithm. By Proposition 1, the equilibrium point is max-min fair and unique. ■

### C. Proof of Proposition 3

The proof of Proposition 3 uses the notion of leximin ordering as defined in Appendix A. We show that the rate vector  $\mathbf{x}$  converges to  $\mathbf{x}^*$  in leximin order, meaning that (1) among unconverged sessions the one with smallest equilibrium rate will converge next, and (2) once a set of sessions with smallest equilibrium rates converge, they will retain such rates afterwards. Intuitively this is due to the fact that the distributed algorithm gives higher priority to consider the sessions with small rates.

Before proving Proposition 3, we state several preliminary lemmas, and give an outline the proof for each of them.

*Lemma 1:* Let  $\mathbf{x}^*$  be the result of the global algorithm. Let  $\mathbf{x}(t)$  be the rate vector of the distributed algorithm in time slot  $t$ . If there exists  $t_1 \geq 0$  and there exists  $n \in [1..K]$ , s.t.

$$\mathbf{x}(t_1) \equiv_n^L \mathbf{x}^*, \quad (9)$$

then for any  $t' > t_1$ , it holds that

$$\mathbf{x}(t') \equiv_n^L \mathbf{x}^*. \quad (10)$$

*Proof:* Given Eq. 9, we are to prove the lemma by showing that the following two statements are always true.

If Eq. 9 is true for time slot  $t$ , then in time slot  $t + 1$  we have that

(a) For any  $j \in [1..n]$ , let  $i = \phi(x_{(j)})$ . Then it holds that

$$x_i(t+1) = x_i(t);$$

and (b) For any  $j \in [n+1..K]$ , it is true that

$$x_{(j)}(t+1) \geq x_{(n)}(t).$$

We prove (a) by induction on  $j = 1..n$ .

(1) When  $j = 1$ , let  $i = \phi(x_{(1)})$ . Then we have

$$x_i(t) = x_i^* = x_{(1)}.$$

We consider two cases: (1a):  $x_i(t) = M_i$  and (1b):  $x_k(t) < M_k$  and  $x_k(t) = \hat{x}_k^s(t)$  or  $x_k(t) = \hat{x}_k^r(t)$ . Similar arguments as used in the proof of Proposition 2 can be applied, and it can be proved that

$$x_i(t+1) = x_i(t) = x^*(t).$$

(2) Suppose (a) is true for  $1..j-1$  ( $j > 0$ ). We need to show that the same is true for  $j$ , i.e. if  $i = \phi(x_{(j)})$  then we have

$$x_i(t+1) = x_i(t).$$

Notice the fact that the calculation of  $\hat{x}_i^r(t+1)$  and  $\hat{x}_i^s(t+1)$  only depends on the source/sink capacity of session  $i$  and sessions with lower rate than  $x_i(t)$  at the same source or sink node. This is also true when calculating  $x_i^*$  in the global algorithm. Notice the similarity between  $b(v)$  and  $x_f$  when  $\mathbf{x}(t) \equiv_{j-1}^L \mathbf{x}^*$ , and by Eq. 5 and 2 it can be proved that

$$x_i(t+1) = x_i(t) = x_i^*.$$

Now we show the correctness of (b). Suppose that there exists  $j \in [n+1, K]$ , and let  $i = \phi(x_{(j)})$ , s.t.

$$x_i(t+1) < x_{(n)}(t) = x_{(n)}^*. \quad (11)$$

As by the definition of leximin ordering,  $x_i(t) \geq x_{(n)}(t)$ , we then have

$$x_i(t+1) < x_i(t). \quad (12)$$

As stated before, Eq. 12 is only valid when  $x_i(t) > x_f$ , where  $x_f$  is the *target rate* at the source or sink of session  $i$  when calculating its expected rate. However from the fact that

$$\mathbf{x}(t) \equiv_{j-1}^L \mathbf{x}^*,$$

we have

$$x_f = \frac{C - \sum_{k, k \in \mathcal{K}^v, \hat{x}_k^r \neq 0} x_k(t)}{|\{k | \hat{x}_k^r = 0, k \in \mathcal{K}^v\}|} \geq \frac{C^v - \sum_{i=1}^{n-1} x_{(i)}^*}{K - n + 1} \geq x_{(n)}^*.$$

Therefore  $x_i(t+1) > x_{(n)}^*$ , which contradicts with Eq. 11. ■

Lemma 1 implies that if a subset of rates which are the lowest ones of  $\mathbf{x}$  converges to the subset of lowest rates of  $\mathbf{x}^*$ , then these rates will stay unchanged afterward. The next lemma is to show that any rate vector that is lexicographically less than  $\mathbf{x}^*$  will monotonically increase in the leximin order in the next time slot.

*Lemma 2:* Let  $\mathbf{x}^*$  be the result of the global algorithm. Let  $\mathbf{x}(t)$  be the rate vector of the distributed algorithm in time slot  $t$ . If

$$\mathbf{x}(t) \prec^L \mathbf{x}^*,$$

then in time slot  $t + 1$  we have that

$$\mathbf{x}(t) \prec^L \mathbf{x}(t+1).$$

*Proof:* As  $\mathbf{x}(t) \prec^L \mathbf{x}^*$ , by definition there exists  $n \in (0, K)$ , s.t.  $\mathbf{x}(t) \equiv_{n-1}^L \mathbf{x}^*$  and (1)  $x_{(n)}(t) < x_{(n)}^*$  or (2)  $x_{(n)}(t) = x_{(n)}^*$ , and  $x_{\tau(x_{(n)})} < x_{\tau(x_{(n)})}^*$ .

For case (1), by applying Lemma 1 we obtain that in time slot  $t + 1$ ,

$$\mathbf{x}(t+1) \equiv_{n-1}^L \mathbf{x}^*.$$

Then it is sufficient to show that in time slot  $t + 1$ ,

$$x_{(n)}(t+1) > x_{(n)}(t). \quad (13)$$

and for any  $l \in (n+1, K]$

$$x_{(l)}(t+1) \geq x_{(n)}(t). \quad (14)$$

Let  $i = \phi(x_{(n)}(t))$ . As is shown in the proof of Lemma 2 that only when the remaining capacity over the number of undetermined sessions ( $x_f$ ) is less than the current minimum rate among undetermined sessions, we have  $\hat{x}_k(t+1) < x_k(t)$ . However as

$$x_i(t) = x_{(n)}(t) < x_{(n)}^* \leq x_i^* \leq x_f,$$

then we have that Eq. 13 holds.

For any  $l \in (n+1, K]$ , similarly as in case (1), when  $x_{(l)}(t) \geq x_{x_{(n)}}^*$ , we have

$$x_{(l)}(t+1) \geq x_{x_{(n)}}^* > x_{(n)}(t);$$

and when  $x_{(l)}(t) < x_{x_{(n)}}^*$ , we have

$$x_{(l)}(t+1) > x_{(l)}(t) \geq x_{(n)}(t).$$

Therefore Eq. 14 is valid.

Case (2) can be proved by using similar arguments as in the proof of case (1). ■

*Lemma 3:* Let  $\mathbf{x}^*$  be the result of the global algorithm. Let  $\mathbf{x}(t)$  be the rate vector of the distributed algorithm in time slot  $t$ . If in time slot  $t_1$ ,  $\mathbf{x}(t_1) \equiv_n^L \mathbf{x}^*$ , where  $0 \leq n < K$  and  $x_{(n+1)}(t_1) \neq x_{(n+1)}^*$ , then there exists  $t_2 > t_1$ , s.t. in any time slot  $t > t_2$  we have that

$$\mathbf{x}(t) \equiv_{n+1}^L \mathbf{x}^* \quad (15)$$

*Proof:* Given that in time slot  $t_1$ ,  $\mathbf{x}(t_1) \equiv_n^L \mathbf{x}^*$ , we consider the following two situations: (1)  $\mathbf{x}(t_1) \succeq_n^L \mathbf{x}^*$ , and (2)  $\mathbf{x}(t_1) \prec_n^L \mathbf{x}^*$ .

*Case (1):* The relations  $\mathbf{x}(t_1) \equiv_n^L \mathbf{x}^*$  and  $\mathbf{x}(t_1) \succeq_n^L \mathbf{x}^*$  imply that for any  $j \in [n+1, K]$ , it holds that

$$x_{(j)}(t_1) \geq x_{(n)}^*.$$

Therefore it can be easily proved that for any  $j \in [n+1, K]$ , let  $i = \phi(x_{(j)})$ , then we have

$$x_i(t+1) \geq x_{(n)}^*.$$

We can also show by using similar technique as in the proof of Lemma 1 that when  $i' = \phi(x_{(n+1)})$ , then we have

$$x_{i'}(t+1) = x_{(n+1)}^*.$$

Together with the fact that  $\mathbf{x}(t_1+1) \equiv_n^L \mathbf{x}^*$  (by Lemma 1), we have

$$x_{(n+1)}(t_1+1) = x_{(n+1)}^*,$$

and Eq. 15 is valid.

*Case (2):* By applying Lemma 2 we obtain that

$$\mathbf{x}(t_1) \prec_n^L \mathbf{x}(t_1+1).$$

Also from the fact  $\mathbf{x}(t_1+1) \equiv_n^L \mathbf{x}^*$  and the smallest increment in the distributed algorithm is  $\alpha \cdot \beta \cdot C^v / |\mathcal{K}^v|$ . Therefore within finite time slots, i.e. there exists time  $t_2 > 0$ , in time slot  $t_2$  there is no unconverged session with rate lower than  $x_{(n+1)}^*$ . We then have that

$$\mathbf{x}(t_2) \succeq_n^L \mathbf{x}^*.$$

This becomes the same case as case (1).

To conclude, in both cases (1) and (2), Eq. 15 becomes valid within finite steps. Lemma 1 ensures that Eq. 15 is always valid afterwards. ■

Lemma 3 shows that if the smallest  $n \in [0, \dots, K-1]$  elements in the rate vector converges, then the  $(n+1)$ th element will converge to its corresponding equilibrium rate within finite time slots. Based on these lemmas we now outline the proof for Proposition 3.

*Proof:* Proposition 3 can be proved by applying lemma 3 through induction on  $n$  from 0 to  $K-1$ , i.e. we show the following:

(1) It holds that there exists  $t_1 \geq 0$  and for any  $t \geq t_1$  we have that  $\mathbf{x}(t) \equiv_1^L \mathbf{x}^*$ . This can be directly obtained from Lemma 3 by setting  $n = 0$ .

(2) Suppose there exists  $t_{n-1} \geq 0$  and for any  $t \geq t_{n-1}$  we have that  $\mathbf{x}(t) \equiv_{n-1}^L \mathbf{x}^*$ . Then there exists  $t_n \geq t_{n-1}$  and for any  $t \geq t_n$  we have that  $\mathbf{x}(t) \equiv_n^L \mathbf{x}^*$ . This is true by applying Lemma 3 and Lemma 1.

Combining (1) and (2), we conclude that there exists  $t_K \geq 0$  and in any time slot  $t \geq t_K$  we have that

$$\mathbf{x}(t) \equiv_K^L \mathbf{x}^*.$$

This proves the proposition. ■