Efficiency of Oblivious Versus Non-Oblivious Schedulers for Optimistic, Rate-Based Flow Control*

(Extended Abstract)

*Panagiota Fatourou** *Marios Mavronicolas*† *Paul Spirakis*‡

Abstract

Two important performance parameters of *rate-based flow control algorithms* are their *locality,* measured by the amount of global knowledge required for the distributed implementation of their scheduling mechanisms, and their *convergence complexity,* defined as the number of *update operations* performed on individual sessions till *max-min fairness* is reached. *Optimistic* algorithms may go through transient states in which one or more sessions receive more than their fair share of bandwidth. In this work, we establish lower and upper bounds on convergence complexity, under varying degrees of locality, for optimistic, rate-based flow control algorithms. We assume a collection of n sessions laid out on a network of *session dependency* d; thus, an update operation on one session may influence at most d sessions.

Say that an algorithm is *oblivious* if its scheduling mechanism uses no information of either the session rates or the network topology. We show a lower bound of $\Omega(dn)$ on the convergence complexity of any oblivious algorithm. We establish that this lower bound is tight by presenting an *optimal,* deterministic, oblivious algorithm which converges after $\Theta(dn)$ update operations are performed in the *worst* case; this algorithm is simple and schedules sessions for an increase in a *round-robin* fashion.

We proceed to show that randomness can be exploited to yield an even simpler oblivious algorithm at the prize of a small increase in convergence complexity. We present a *randomized,* oblivious algorithm, where each session S_i is scheduled for an increase with probability p_i. Denote $p_{min} = \min_{1 \leq i \leq n} p_i$. This algorithm converges to the max-min rates after an *expected* number of $O(d \ln n / p_{min})$ update operations. The proof of this upper bound relies on an analysis of the *generalized coupon collector's problem,* an interesting variant of the classical *coupon collector's problem.*

We next turn to algorithms that are *partially oblivious,* in that their scheduling mechanisms "know" and may use session rates, but are unaware of network topology. We show that, perhaps surprisingly, the lower bound of $\Omega(dn)$ on convergence complexity still holds for partially oblivious algorithms; this lower bound is matched by the convergence complexities of previously proposed algorithms which must use session rates. Our results for partially oblivious algorithms imply that knowledge of session rates cannot merely suffice to reduce convergence complexity.

We finally show that *linear* convergence complexity can be achieved if knowledge of both session rates and network topology is available to schedulers. We present a "counter-example", non-oblivious algorithm, which schedules a session for an increase only if it uses the most congested link in the network. We show that this algorithm converges after performing an *optimal* number of n update operations in *every* case. Our results imply a convergence complexity separation between partially oblivious (in particular, oblivious) and non-oblivious algorithms for optimistic, rate-based flow control.

1 Introduction

In most commercial *communication networks,* a connection between different users is established by a *session,* a virtual circuit involving a fixed path between a source and a destination.[1] In such networks, situations often arise where the externally offered load is larger than what can be handled even with optimal routing; then, queue sizes at bottleneck links may grow indefinitely and eventually exceed the buffer space at the corresponding nodes, and packets arriving at nodes with no available buffer space will have to be discarded and later retransmitted, thereby wasting communication resources. The function of *flow control algorithms* is to prevent such situations from arising (see, e.g., [3, Chapter 6] or [9, 11, 13, 14, 20]).

*Department of Computer Engineering and Informatics, University of Patras, Patras, Greece & Computer Technology Institute, Patras, Greece. Partially supported by the EU ESPRIT Long Term Research Projects ALCOM-IT (Proj. # 20244) and GEPPCOM (Proj. # 9072) and the Greek Ministry of Education. E-mail: faturu@cti.gr

†Department of Computer Science, University of Cyprus, Nicosia, Cyprus. Supported by funds for the promotion of research at University of Cyprus (research projects "Distributed, Parallel, and Concurrent Computations" and "Distributed Processing: Multiprocessors, Networks and Verification"), and by NCR research funds. E-mail: mavronic@turing.cs.ucy.ac.cy

‡Department of Computer Engineering and Informatics, University of Patras, Patras, Greece & Computer Technology Institute, Patras, Greece. Partially supported by the EU ESPRIT Long Term Research Projects ALCOM-IT (Proj. # 20244) and GEPPCOM (Proj. # 9072), and the Greek Ministry of Education. E-mail: spirakis@cti.gr

[1] In the model we consider, it is assumed that each session lifetime is infinite; this assumption is suitable for applications such as ftp, batch jobs and Mosaic.

In particular, *rate-based* flow control algorithms adjust transmission rates of different sessions in an end-to-end manner, with the objective to optimize network utilization while maintaining fairness between different sessions (see, e.g., [1, 3, 6, 10, 12, 14]). The rate-based approach has been particularly attractive due to its simplicity and its modest hardware requirements per virtual circuit (as opposed to those of *credit-based* [18] or *virtual-channel-based* [7] approaches). Indeed, the Asynchronous Transfer Mode (ATM) Forum on Traffic Management has adopted rate-based flow control as the prime mechanism for flow control of Available Bit Rate (ABR) traffic in its networks (see, e.g., [4, 8, 17, 19, 23]). A widely accepted fairness criterion for rate-based flow control is *max-min fairness* [1, 3, 6, 12, 13, 14], requiring that it be impossible to infinitesimally increase the rate of any session without decreasing the rate of a session whose rate is equal or smaller.

Any rate-based flow control algorithm falls into one of two broad classes, *conservative* and *optimistic*, according to the way in which rates of sessions are adjusted. Conservative algorithms converge without ever assigning a session a rate that is larger than its final rate; in contrast, optimistic algorithms are more "aggressive" in that they allow a session to intermediately receive a rate larger than its final. Optimistic algorithms fit better than conservative ones into "real" dynamic networks that need to be able to decrease the rates of some sessions in order to accomodate new entering sessions; moreover, the optimistic approach appears closer to rate-based flow control algorithms that are suggested in the ATM Forum on Traffic Management (see, e.g., [15, 24]). Surprisingly, however, it has been only very recently that optimistic algorithms were introduced in a pioneering work by Afek *et al.* [1].

A significant component of any rate-based flow control algorithm, whether conservative or optimistic, is its *scheduler,* the mechanism it uses to decide which session to adjust next by an increase. In favor of robust and efficient distributed implementations, it is desirable that the scheduler does not require *global* knowledge of network topology and session rates. Clearly, "non-centralized" schedulers adjust easier to dynamic changes in network topology, and they are more efficient in terms of both communication and computation. So, one important parameter of a rate-based flow control algorithm, from the point of view of network performance, is its *locality,* measured by the amount of global knowledge required by the scheduler.

Call a scheduler that uses no information of either the network topology or the session rates an *oblivious* scheduler; apparently, an oblivious scheduler enjoys nice properties of robustness, efficiency, and portability. In particular, oblivious schedulers avoid both the problem of communication in a distributed algorithm, and the problem of the ambiguity in the meaning of rate for interactive traffic (cf. [3, Section 6.4.2]). At the opposite extreme, a *non-oblivious* scheduler requires full knowledge of both network topology and (all) session rates; clearly, a non-oblivious scheduler is more amenable to "centralized", rather than distributed, implementations. There is, however, a middle ground between oblivious and non-oblivious schedulers: schedulers which, although unaware of network topology, do have access to session rates; call these schedulers *partially oblivious.* A partially oblivious scheduler is superior to a non-oblivious scheduler in terms of robustness to dynamic changes in network topology, while it is surpassed by an oblivious scheduler in being ambiguous with respect to the meaning of rate for interactive traffic. Afek *et al.* [1, Sections 4 & 5] present two interesting, partially oblivious schedulers called GlobalMin and LocalMin, respectively.

Once a session has been scheduled, a control message loops around its path and calculates on its way the minimum "share" of bandwidth the session may take from the excess capacities of links along the path; this requires possible adjustments to the rates of conflicting sessions. Roughly speaking, the *convergence complexity* of a rate-based flow control algorithm captures the number of rate adjustments performed in the worst-case till max-min fairness is reached. Since reaching max-min fairness fast is essential for efficient utilization of the virtual circuits, convergence complexity is another significant performance parameter of rate-based flow control algorithms. We measure convergence complexity in terms of a simple abstraction of update operations introduced by Afek *et al.* [1, Section 2.1]; an *update* operation adjusts the rates of sessions in a fair and optimistic way.

The convergence complexities of optimistic algorithms proposed in [1] have been expressed in terms of the total number n of sessions laid out on the network. A significant contribution of our work is the identification of a parameter other than n and the derivation of more meaningful bounds on convergence complexity in terms of this parameter. Specifically, let d be the maximum number of sessions that share an edge either directly or indirectly; call d the *session dependency.* Thus, d is the maximum number of sessions "influenced" by a single update operation on any session; clearly, $1 \leq d \leq n$. It turns out that the session dependency d, together with the number of sessions n, precisely determines the optimal convergence complexity achievable by oblivious or partially oblivious algorithms.

Our first major result is a fundamental and generic lower bound on the convergence complexity of any oblivious algorithm that computes the max-min fairness rates. We provide a general and novel methodology for constructing, given any fixed but arbitrary oblivious algorithm that computes the max-min rates, a specific network, as a function of the algorithm's scheduler, so that if sessions are scheduled for increase on this network according to the scheduler, $dn/4+n/2$ update operations are required before converging to the max-min rates. The construction uses the sequence of sessions in order to appropriately define the edges and assign capacities to them, in a way that "retards" sessions from reaching their final (max-min) rates.

We show a corresponding, matching upper bound on the convergence complexity of oblivious algorithms. We present an oblivious, deterministic algorithm, RoundRobin, with convergence complexity $dn/2 + n/2$. This algorithm borrows the very simple idea of scheduling different sessions on a "round-robin" basis, originally introduced into rate-based flow control by Hahne and Gallager [12]; the scheduler conducts n "rounds," in each of which all sessions are scheduled in a "round-robin" fashion.

We show non-trivial properties of bottleneck algorithms, which we use to show an upper bound of dn on the convergence complexity of RoundRobin. Afek *et al.* [1, Sections 4 & 5] establish upper bounds of $\Theta(n^2)$ on the convergence complexities of both schedulers GlobalMin and LocalMin; moreover, Afek *et al.* present a specific network construction to show that these upper bounds are *tight* in the case where $d = n$ (that is, all sessions fall in a single cluster). However, since there are cases where $d \in o(n)$, the upper bound of $\Theta(dn)$ we have shown for the convergence complexity of RoundRobin is the first subquadratic upper bound on convergence complexity shown for optimistic, rate-based, flow control algorithms. Even more so, both schedulers GlobalMin and LocalMin are only partially oblivious, while RoundRobin is oblivious, hence more "local" and more "distributed".

Is it possible to use randomization for obtaining optimistic, rate-based flow control algorithms that are even simpler than RoundRobin, yet still oblivious? We answer this question in the affirmative by presenting an oblivious, randomized algorithm, called Random; indeed, Random is the first randomized algorithm ever proposed for rate-based flow control (either conservative or optimistic). A probability p_i is associated with each session S_i, $1 \leq i \leq n$, which is the probability that the scheduler selects the session.

To determine the convergence complexity of Random, we consider and analyze an interesting generalization of the classical *coupon collector's problem* [22], where each coupon is drawn with a certain probability (not necessarily uniform); call it the *generalized coupon collector's problem.* We use properties of bottleneck algorithms to observe an apparent correspondence between the analysis of the *generalized coupon collector's problem* and that of the convergence complexity of Random. This correspondence implies that the expectation and standard deviation of the number of update operations performed in any run of Random are $O(d \ln n / p_{min})$ and $O(\sqrt{d}/p_{min})$, respectively, where $p_{min} = \min_{1 \leq i \leq n} p_i$. It is remarkable that the bound on the expectation of the convergence complexity of Random becomes $O(dn \ln n)$ in the *uniform* case, where all probabilities are equal to $1/n$; this bound is only slightly inferior to the corresponding bound shown for the still oblivious but somehow less simple algorithm RoundRobin.

We next turn to partially oblivious algorithms. We extend the generic lower bound of $\Omega(dn)$ we have shown on the convergence complexity of oblivious algorithms to partially oblivious ones. We do so by simultaneously "hand-crafting," in a step-by-step fashion, both the sequence of sessions produced by *any* partially oblivious algorithm and the network for which $\Omega(dn)$ update operations are required for convergence to the max-min vector. It turns out that this lower bound is, indeed, tight for partially oblivious algorithms. We observe that the upper bound of $\Theta(n^2)$ on convergence complexity shown by Afek *et al.* [1] for the partially oblivious algorithms GlobalMin and LocalMin, specialized for the case where $d = n$, implies an upper bound of $\Theta(dn)$ for these algorithms in the case of general d. Thus, although intuition may suggest that knowledge of session rates can be crucial to performance, our results imply that this is *not* the case.

At this point, it is natural to ask whether it is possible to beat the $\Theta(dn)$ bound on convergence complexity achievable by oblivious or partially oblivious algorithms, possibly at the prize of sacrificing locality. Perhaps not too surprisingly, it turns out that the locality enjoyed by oblivious and partially oblivious algorithms comes at a multiplicative in d overhead on convergence complexity. We discover a "counter-example" (deterministic) algorithm Linear that achieves an *exact* bound of n on convergence complexity; however, Linear is neither oblivious nor partially oblivious. In a nutshell, Linear follows the natural idea of selecting for an increase a session on the most congested link in the network (see, e.g., [3, Section 6.4.2] or [13]).[2] Clearly, the algorithm Linear is *optimal* with respect to convergence complexity, since each of the n sessions must be updated at least once by any algorithm (whether oblivious or not).

Our lower and upper bounds exhibit interesting trade-offs between convergence complexity and locality for optimistic, rate-based flow control algorithms. Among our three proposed algorithms, Random is the "simplest", hence the most "distributed", but attains the highest (expected) convergence complexity of the three. Alternatively, RoundRobin, though less "simple" than Random, still enjoys locality but yet manages to achieve convergence complexity which is optimal for the class of oblivious schedulers. On the other hand, the "centralized" algorithm Linear achieves optimal convergence complexity, but does not enjoy locality.

Both RoundRobin and Random admit simple distributed implementations. The implementation of RoundRobin makes use of a set of underlying "logical rings," one per session cluster, built on top of the communication network. Each such ring is designed to traverse at least one edge, called *logical edge,* of each session in the cluster, and be as short as possible. A "token" is circulated on each ring to simulate the round-robin scheduling of sessions in the cluster; the ordering of the scheduling is determined by the sequence of logical edges in the ring. Each time the token traverses a logical edge, an update operation on the corresponding session is initiated; once the operation is completed, the token advances to the next logical edge, and so on. The implementation of Random relies on a randomized, distributed protocol to toss a multiple-sided, biased coin among sessions in each cluster; the communication between different sessions uses shortest paths on the underlying communication network.

In conclusion, our work deviates from previous work on rate-based flow control (e.g., [3, 9, 10, 11, 12, 13, 14, 20]) carried out in the data networks community in that it adopts the optimistic approach put forward in the pioneering work by Afek *et al.* [1]; this approach emerges around the update operation, which allows sessions to intermediately receive rates that are lower than their final (fair) rates. We have managed to show that certain "classical" scheduling policies, such as round-robin scheduling [12] or scheduling a session using the most congested link [3, 13], still work in the optimistic framework; moreover, we have shown that some of these policies are superior, in terms of either convergence complexity or locality (or both), to some other "classical" policies, such as scheduling the session with the globally smallest rate [3], that have been already studied within the optimistic framework by Afek *et al.* [1].

The rest of this paper is organized as follows. Section 2 includes definitions for the formal model and some preliminary facts, while Section 3 exhibits key properties of bottleneck algorithms. Results on deterministic and randomized oblivious schedulers appear in Sections 4 and 5, respectively. Deterministic, partially oblivious schedulers are studied in Section 6, while Section 7 considers non-oblivious schedulers. We conclude, in Section 8, with a discussion of our results and some open problems.

2 Definitions and Preliminaries

In this section, we present formal definitions and some preliminary facts. Many of our definitions formalize and refine corresponding ones in [1, 3]. This section is organized as follows. Section 2.1 introduces the update relation, while Section 2.2 summarizes material on a generalization of the coupon collector's problem. Network, sessions and allocation vectors are introduced in Section 2.3. A network model appears in Section 2.4. Section 2.5 formally defines the up-

[2] This idea forms the basis of a simple and classical iterative algorithm to compute the max-min fairness rates [3, Section 6.4.2]. In each iteration, the rates of sessions that use the "most congested" link are fixed. In the next iteration, the fixed rates are subtracted from the corresponding link capacities, and the procedure repeats itself. We note that this algorithm is not optimistic.

date operation. Schedulers and terminators are introduced in Section 2.6, while Section 2.7 defines executions and convergence complexity.

2.1 The Update Relation

The following is a restatement of [1, Appendix A, Definition A.1] in the form of a relation. Fix any integer $m \geq 2$. The *update relation* is a relation $update \subseteq (\Re^m \times [m] \times \Re) \times \Re^m$ such that $\langle\langle \mathbf{R}, j, c\rangle, \mathbf{R}'\rangle \in update$ if and only if there exists a number $\Delta > 0$ for which the following conditions are satisfied: (1) $\|\mathbf{R}'\|_1 = c$; (2) $r'_j = r_j + \Delta$; (3) for each $i \in [m]$, $i \neq j$, if $r_i \leq r_j + \Delta$, then $r'_i = r_i$; (4) for each $i \in [m]$, $i \neq j$, if $r_i > r_j + \Delta$, then $r'_i = r_j + \Delta$. Whenever $\langle\langle \mathbf{R}, j, c\rangle, \mathbf{R}'\rangle \in update$, call $\mathbf{R}'$ a *reform* of $\mathbf{R}$ (cf. [1, Appendix A, Definition A.1]). Intuitively, a reform is such that c is "saturated" (Condition 1) by increasing entry r_j by Δ (Condition 2), while smaller entries are not affected (Condition 3); moreover, the reform "preserves" fairness since there can be left no entries larger than the increased entry (Condition 4). A *maximum reform* of $\mathbf{R}$ is a reform of $\mathbf{R}$ that corresponds to the maximum possible Δ.

2.2 The Generalized Coupon Collector's Problem

In the *generalized coupon collector's problem,* there are m coupon types denoted $1, \ldots, m$; at each trial, a coupon is chosen at random. Each randomly chosen coupon is likely to be of type i, $1 \leq i \leq m$, with probability p_i, where $\sum_{i=1}^{m} p_i = 1$; denote $p_{min} = \min_{1 \leq i \leq m} p_i$. Notice that $p_{min} \leq 1/m$, and assume throughout that $p_{min} > 0$. The random choices of coupons are mutually independent. In the "standard" *coupon collector's problem* [22], $p_1 = \ldots = p_m = 1/m$. Let X be the random variable defined as the number of trials required to collect at least one of each type of coupon. Denote $\mathcal{E}(X)$ and $\sigma(X)$ the expectation and standard deviation of X, respectively. We show:

Proposition 2.1 *The following hold on the probability distribution of X:*

(1) $\mathcal{E}(X) \leq \ln m/p_{min} + r(m)$, *where* $r(m) \in \Theta(1/p_{min})$;

(2) $\sigma(X) \in O(1/p_{min})$.

2.3 Network, Sessions, and Allocation Vectors

A *communication network* is a directed graph $G = (V, E)$, where each vertex $v \in V$ represents a network node, and each edge $e \in E$ represents a link in the network. Associated with each edge $e \in E$ is a *capacity* $cap(e)$, which is a real number greater than zero. A *session* is a sequence of links which is a simple path in G between a *source* and a *destination*; a set of n sessions $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ is laid out on G. For each edge $e \in E$, denote $sessions(e)$ the set of sessions passing through e.

Define the *share-an-edge* relation on $\mathcal{S}$, denoted $\|_{\mathcal{S}}$, as follows. For any pair of sessions $S_i, S_j \in \mathcal{S}$, $S_i \|_{\mathcal{S}} S_j$ if there exists an edge $e \in E$ such that both $S_i \in sessions(e)$ and $S_j \in sessions(e)$; that is, $S_i \|_{\mathcal{S}} S_j$ if S_i and S_j share an edge "in parallel". Notice that $\|_{\mathcal{S}}$ is reflexive and symmetric. The transitive closure of $\|_{\mathcal{S}}$, denoted $\|^{\star}_{\mathcal{S}}$, is an equivalence relation on $\mathcal{S}$, which partitions $\mathcal{S}$ into equivalence classes $\mathcal{S}_1, \ldots, \mathcal{S}_c$, called *session clusters,* or *clusters* for short, where $1 \leq c \leq n$, $|\mathcal{S}_j| = n_j$ and $n_1 + \ldots + n_c = n$; note that for all indices j and k, $1 \leq j, k \leq c$ and $j \neq k$, $\mathcal{S}_j \cap \mathcal{S}_k = \emptyset$, while $\cup_{1 \leq j \leq c} \mathcal{S}_j = \mathcal{S}$. Denote $d = \max\{n_1, \ldots, n_c\}$; that is, d is the maximum size of a session cluster. Call d the *session dependency.* Clearly, $1 \leq d \leq n$.

For a sequence ς of session indices, denote $\varsigma \mid \mathcal{S}_j$ the *restriction* of ς to indices of sessions in cluster $\mathcal{S}_j$. Denote $\varsigma \uparrow \mathcal{S}_j$ the shortest prefix of $\varsigma \mid \mathcal{S}_j$ that includes the indices of all sessions in cluster $\mathcal{S}_j$ in the order they appear in this prefix of $\varsigma \mid \mathcal{S}_j$ and no repetitions, or $\varsigma \mid \mathcal{S}_j$ if no such prefix exists. Denote $\varsigma \downarrow \mathcal{S}_j$ the remaining suffix of $\varsigma \mid \mathcal{S}_j$.

Allocated to each session S is a *bandwidth* or *rate,* denoted $rate(S)$, which is a real number. An *allocation vector* is a vector $\mathbf{R} = \langle rate(S_1), rate(S_2), \ldots, rate(S_n)\rangle$.

The capacity constraint requires that for each edge, the sum of rates of sessions sharing the edge does not exceed the edge capacity. A *feasible allocation vector,* or *feasible allocation* for short, is an allocation that satisfies all capacity constraints. A *maximal allocation vector,* or *maximal allocation* for short, is a feasible allocation in which it is not possible to increase the rate of any session without decreasing the rate of some other session. A *max-min allocation vector,* or *max-min allocation* for short, is a maximal allocation such that for each session S_i, $rate(S_i)$ cannot be increased (while still maintaining feasibility) without decreasing $rate(S_j)$ for some session S_j such that $rate(S_j) \leq rate(S_i)$; that is, an increase in the rate of any session requires a decrease in the rate of a session with equal or lower rate.

2.4 Modeling the Network

The communication network is modeled as a (possibly infinite) state machine. Each state Q of the network consists of two components: a *feasible allocation vector* $\mathbf{R} \in \Re^n$, and an *active set* $\mathcal{A} \subseteq \{S_1, S_2, \ldots, S_n\}$, which is a set of "active" sessions; that is, $Q = \langle \mathbf{R}, \mathcal{A}\rangle$. Intuitively, an active session is one that has not yet reached its final rate. In the *initial state* $Q^{in} = \langle \mathbf{R}^{in}, \mathcal{A}^{in}\rangle$, $\mathbf{R}^{in} = \langle 0, 0, \ldots, 0\rangle$ and $\mathcal{A}^{in} = \langle S_1, S_2, \ldots, S_n\rangle$. A state $Q = \langle \mathbf{R}, \mathcal{A}\rangle$ is *final* if $\mathcal{A} = \emptyset$; that is, in a final state, all sessions have reached their final rates. Denote $\mathcal{D} = \{S_1, S_2, \ldots, S_n\} \setminus \mathcal{A}$; that is, $\mathcal{D}$ is the set of *done* sessions in state Q, which are sessions that have reached their final rates in state Q. Call a session cluster $\mathcal{S}_j$ an *active cluster* if it contains at least one active session; otherwise, $\mathcal{S}_j$ is called a *done cluster.*

For each edge $e \in E$ and state $Q = \langle \mathbf{R}, \mathcal{A}\rangle$, denote $\#_Q(e)$ the number of sessions that are active in state Q and pass through edge e; that is $\#_Q(e) = |\mathcal{A} \cap sessions(e)|$. The set of *active edges for a session cluster $\mathcal{S}_i$ in state Q,* denoted $AE_Q(\mathcal{S}_i)$, is the set of all edges of the network traversed by at least one active session in state Q, that belongs to cluster $\mathcal{S}_i$. Correspondingly, the *set of active edges of the network in state Q,* denoted AE_Q, is the set of all edges of the network traversed by at least one session that is active in state Q.

For each edge $e \in E$ and state $Q = \langle \mathbf{R}, \mathcal{A}\rangle$, the *allotted capacity of e in Q* [1], denoted $allotted_Q(e)$, is defined to be the total rate already allocated to *done* sessions passing through e. For any edge $e \in E$ and any state Q such that $\#_Q(e) > 0$, the *Fair Share of e in Q* [1], denoted $FS_Q(e)$, is defined as $FS_Q(e) = (cap(e) - allotted_Q(e))/\#_Q(e)$; roughly speaking, $FS_Q(e)$ is the fair share of the "available" capacity of edge e in state Q for each of the sessions passing through e that are still active. For each state $Q = \langle \mathbf{R}, \mathcal{A}\rangle$, a *bottleneck edge for Q* is an edge $e \in E$ such that each active session passing through e receives its smallest fair share, among all of its edges, on edge e. For each $j \in [n]$, the *Minimum Fair Share of session S_j in state Q* [1, Definition 3.6], denoted $MFS_Q(S_j)$, is the smallest among all fair shares that S_j

receives on any of its edges in state Q.

For each cluster $\mathcal{S}_j$, the set of *edges with minimum fair share for cluster* $\mathcal{S}_j$ *in state* Q, denoted $MFSE_Q(\mathcal{S}_j)$, is the set of all edges traversed by a session in $\mathcal{S}_j$ that is active in state Q, whose fair share is the least among all such edges. The set of *edges with minimum fair share in state* Q, denoted $MFSE_Q$, includes an edge e only if there exists an active session passing through e whose minimum fair share is equal to the fair share of e. Of particular interest is the subset $LMFSE_Q$ of $MFSE_Q$ including edges whose fair share is the *least* among those of all edges in $MFSE_Q$.

2.5 The Update Operation

For any session S_j, the *maximum increase for* S_j *in state* Q *imposed by edge* $e \in S_j$, denoted $\Delta_Q(j,e)$, is the maximum reform increase for $\langle \mathbf{R}, j, cap(e)\rangle$; that is, $\Delta_Q(j,e)$ is the maximum $\Delta > 0$ for which there exists a vector $\mathbf{R}'$ such that $\langle\langle \mathbf{R}, j, cap(e)\rangle, \langle \mathbf{R}', \Delta\rangle\rangle \in update$; roughly speaking, $\Delta_Q(j,e)$ is the maximum possible increase to the rate of session S_j that "saturates" edge e, while possibly decreasing in a fair manner other sessions passing through e.

The *maximum increase for* S_j *in state* Q, denoted $\Delta_Q(j)$, is the minimum, over all edges $e \in S_j$, of the maximum increase for S_j in state Q imposed by edge e; that is, $\Delta_Q(j)$ is the minimum among all maximum increases of S_j which "saturate" edges traversed by S_j and possibly decrease sessions "parallel" to S_j in a fair manner; $\Delta_Q(j)$ captures the fact that S_j can be increased by a certain amount only if it can be increased by this amount on each of its edges.

An *update operation* is a specific instantiation of the update relation. Intuitively, in an update operation, $\mathbf{R}$ corresponds to the rates of a set of sessions sharing an edge $e \in E$ such that $cap(e) = c$, and j is a session index; $\mathbf{R}'$ corresponds to the rates of these sessions after performing an update operation. The operation "saturates" e (Condition 1) by increasing S_j by Δ (Condition 2), while sessions with lower or equal rates are not affected (Condition 3); moreover, the operation "preserves" fairness since, in addition, there can be left no sessions with rate higher than that of the increased session (Condition 4).

2.6 Schedulers and Terminators

A *deterministic scheduler* is a function Scheduler that maps a pair $(G, \mathcal{S})$ of a network G and a set of sessions $\mathcal{S}$ laid out on G, a network state Q, and an integer $l \geq 1$ to an index $i = \mathsf{Scheduler}((G,\mathcal{S}), Q, l)$ of some session from $\mathcal{S}$.

An oblivious scheduler "knows" neither the topology of the network nor the status and rates of the sessions in choosing the session to schedule next. Thus, for a session set $\mathcal{S} = \{S_1, \ldots, S_n\}$, an oblivious scheduler may be identified with a (finite or infinite) sequence $\mathsf{Scheduler} = i_1, i_2, \ldots$, where for each $l \geq 1$, $i_l \in [n]$. A *partially oblivious scheduler* does not "know" the topology of the network in choosing the session to schedule next, but does know the rates of (all) sessions. Thus, for a session set $\mathcal{S} = \{S_1, \ldots, S_n\}$, a partially oblivious scheduler Scheduler may be identified with a (finite or infinite) sequence of functions $\mathsf{Scheduler}_l$, one for each integer $l \geq 1$, where each function maps a network state Q to an index $i = \mathsf{Scheduler}_l(Q)$ of a session from $\mathcal{S}$.

A *randomized scheduler* is a function Scheduler that maps a pair $(G, \mathcal{S})$ of a network G and a set of sessions $\mathcal{S}$ laid out on G, a network state Q, and an integer $l \geq 1$ to a (discrete) probability distribution $f = \mathsf{Scheduler}((G,\mathcal{S}), Q, l)$ on the session set $\mathcal{S}$. An oblivious randomized scheduler is defined in the natural way.

A *terminator* is a function Terminator that maps a network state $Q = \langle \mathbf{R}, \mathcal{A}\rangle$ to a session set $\mathcal{T} = \mathsf{Terminator}(Q)$ such that $\mathcal{T} \subseteq \mathcal{A}$; intuitively, Terminator decides which sessions among those still active should be marked as *done*. Say that Terminator is *bottleneck* (see, e.g., [3, Chapter 6]) if for any state $Q = \langle \mathbf{R}, \mathcal{A}\rangle$ and session S, $S \in \mathsf{Terminator}(Q)$ if (and only if) there exists an edge $e \in S$ such that (1) e is a bottleneck edge for Q, and (2) $rate(S) = FS_Q(e)$.

An *algorithm* is a pair $\mathsf{Alg} = \langle \mathsf{Scheduler}, \mathsf{Terminator}\rangle$. Say that Alg is *oblivious* if Scheduler is; say that Alg is *bottleneck* if Terminator is.

2.7 Executions and Convergence Complexity

We model computations on the network as sequences of update operations, each of which increases the rate of some session in the network and decreases the rates of other sessions that are "parallel" to the one being increased. Formally, an *execution of* Alg *on network* G *with session set* $\mathcal{S}$ is an infinite sequence of alternating states and session indices $\alpha = Q_0, i_1, Q_1, \ldots, i_l, Q_l, \ldots$, satisfying the following conditions: (1) $Q_0 = Q^{in}$; (2) for each $l \geq 1$, $i_l = \mathsf{Scheduler}(Q_{l-1}, i_{l-1})$; (3) for each $l \geq 1$, $\mathcal{A}_l = \mathcal{A}_{l-1} \setminus \mathsf{Terminator}(Q_{l-1})$; (4) for each $l \geq 1$, for each edge $e \in S_{i_l}$,

$$\langle\langle \mathbf{R}_{l-1}, i_l, cap(e)\rangle, \langle \mathbf{R}_l, \Delta_{Q_{l-1}}(i_l, e)\rangle\rangle \in update.$$

For any integers l_1 and l_2, $1 \leq l_1 \leq l_2$, the (l_1, l_2)-*suffix* denoted *suffix*(l_1, l_2), is defined to be the set of indices of all sessions that were updated by at least one of the update operations performed strictly after the network entered state Q_{l_1} and no later than when the network enters state Q_{l_2}. For any state Q_i, let $|\mathcal{A}_{Q_i}| = k \geq 1$. We denote by $F(k)$ the minimum number of required update operations, over all bottleneck algorithms, so that *suffix*$(i, i + F(k))$ contains all sessions active in Q_i; clearly, *suffix*$(i, i + F(k)) = \mathcal{A}_{Q_i}$.

The *convergence complexity of* Alg *on network* G *for the cluster* $\mathcal{S}_j$, denoted $\mathcal{U}_{\mathsf{Alg}}(G, \mathcal{S}_j)$, is defined to be the maximum, over all executions of Alg on network G of the minimum number of update operations that need to be performed in an execution in order for all sessions in $\mathcal{S}_j$ to be marked as *done*. The *convergence complexity of* Alg *on network* G *with a set of sessions* $\mathcal{S}$, denoted $\mathcal{U}_{\mathsf{Alg}}(G, \mathcal{S})$, is the sum, over all session clusters, of $\mathcal{U}_{\mathsf{Alg}}(G, \mathcal{S}_j)$. The *convergence complexity of* Alg, denoted $\mathcal{U}_{\mathsf{Alg}}$, is defined to be the maximum, over all pairs of a network G and a session set $\mathcal{S}$ laid out on G, of the convergence complexity of Alg on network G with session set $\mathcal{S}$.

Consider any algorithm Alg such that $\mathcal{U}_{\mathsf{Alg}}(\mathcal{S}_j) \leq n_j^2$. Since $\sum_{j \geq 1} n_j^2 \leq dn$ and $\mathcal{U}_{\mathsf{Alg}} = \sum_{j=1}^{c} \mathcal{U}_{\mathsf{Alg}}((\mathcal{S}_j)$, it follows that $\mathcal{U}_{\mathsf{Alg}} \leq dn$. In particular, for the partially oblivious schedulers LocalMin and GlobalMin [1, Sections 4 & 5], it holds that:

Proposition 2.2 $\mathcal{U}_{\mathsf{GlobalMin}} \leq dn$ *and* $\mathcal{U}_{\mathsf{LocalMin}} \leq dn$.

3 Bottleneck Algorithms

In this section, we exhibit basic properties of bottleneck algorithms. Some of these properties have been known previously, while most of them are new. The first property states that the output allocation vector of a bottleneck algorithm is a max-min allocation vector.

Proposition 3.1 (Bertsekas & Gallager [3]) *Let* Alg *be a bottleneck algorithm. Then, for any final state* $Q = \langle \mathbf{R}, \emptyset\rangle$, $\mathbf{R}$ *is a max-min allocation vector.*

The next result states that the fair share of an edge is monotonically non-decreasing for any execution of a bottleneck algorithm.

Proposition 3.2 (Afek *et al.* **[1])** *Let* Alg *be a bottleneck algorithm . Then, for each edge $e \in E$ and $l \geq 1$ such that $\#_{Q_l}(e) > 0$,*

$$FS_{Q_l}(e) \;\geq\; FS_{Q_{l-1}}(e).$$

Roughly speaking, Proposition 3.2 implies that the fair share of an edge may not decrease, as long as there are still active sessions using the edge, while a bottleneck algorithm is running. We continue to show:

Proposition 3.3 *Assume that* Alg *is a bottleneck algorithm. Consider a cluster $\mathcal{S}_j$ that is active in state Q_k, and fix any edge $e \in MFSE_{Q_k}(\mathcal{S}_j)$. Then, for any index $l \geq k$ such that $\mathcal{S}_j$ is active in Q_l, for each edge $e' \in AE_{Q_l}(\mathcal{S}_j)$,*

$$FS_{Q_l}(e') \;\geq\; FS_{Q_k}(e).$$

Proof: Take any edge $e' \in AE_{Q_l}(\mathcal{S}_j)$. Obviously, $e' \in AE_{Q_k}(\mathcal{S}_j)$. Hence, by definition of $MFSE_{Q_k}(\mathcal{S}_j)$, $FS_{Q_k}(e) \leq FS_{Q_k}(e')$. By Proposition 3.2, $FS_{Q_l}(e') \geq FS_{Q_k}(e')$. It follows that $FS_{Q_l}(e') \geq FS_{Q_k}(e)$, as needed. ■

The next result presents another property of bottleneck algorithms.

Proposition 3.4 (Afek *et al.* **[1])** *Let* Alg *be a bottleneck algorithm. Consider any state Q_k such that either $i_k = j$ or $rate_{Q_{k+1}}(S_j) < rate_{Q_k}(S_j)$. Then for all $l \geq k$,*

$$rate_{Q_l}(S_j) \;\geq\; MFS_{Q_k}(S_j).$$

We continue to show some additional properties of bottleneck algorithms.

Proposition 3.5 *Assume that* Alg *is a bottleneck algorithm. Consider a cluster $\mathcal{S}_j$ that is active in state Q_i, and fix any edge $e \in MFSE_{Q_i}(\mathcal{S}_j)$. For every session $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ the following hold:*

(1) *There is no edge that may cause a decrease of the rate of S_r, to a value less than $FS_{Q_i}(e)$.*

(2) *S_r can not have final rate greater than $FS_{Q_i}(e)$.*

Proof: We first prove (1). Since $e \in MFSE_{Q_i}(\mathcal{S}_j)$ for each session $S_l \in sessions(e) \cap \mathcal{A}_{Q_i}$ we have, $FS_{Q_i}(e) = \min_{e' \in S_l} FS_{Q_i}(e')$, which implies that e is a bottleneck edge. Hence, each of the sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ receives on edge e the minimum among all fair shares on any edge $e' \in AE_{Q_i}(\mathcal{S}_j)$. Thus, there is no edge that may cause a decrease of the rate of any $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$, to a value less than $FS_{Q_i}(e)$.

The second property is an immediate consequence of fairness and the capacity constraint. ■

An immediate consequence of Proposition 3.5 follows.

Corollary 3.6 *Assume that* Alg *is a bottlenek algorithm. Consider a cluster $\mathcal{S}_j$ that is active in state Q, and fix any edge $e \in MFSE_Q(\mathcal{S}_j)$. Then, the final rate of any session traversing e that is active in state Q is $FS_Q(e)$.*

The next property implies that that the edge with the minimum fair share among all edges that are active for some particular cluster is "stable"; that is, this edges retains its property as long as it remains active.

Proposition 3.7 *Assume that* Alg *is a bottleneck algorithm. Consider a cluster $\mathcal{S}_j$ that is active in state Q_i, and fix any edge $e \in MFSE_{Q_i}(\mathcal{S}_j)$. Take any index $l \geq i$ such that $e \in AE_{Q_l}$. Then, $e \in MFSE_{Q_l}(\mathcal{S}_j)$.*

Sketch of proof: By Proposition 3.3, the following holds: $FS_{Q_l}(e') \geq FS_{Q_i}(e)$, $\forall e' \in AE_{Q_l}(\mathcal{S}_j)$ and $l \geq i$. We will prove that $FS_{Q_l}(e) = FS_{Q_i}(e)$, which implies that $FS_{Q_l}(e') \geq FS_{Q_l}(e)$, that is e continues to be the edge with the smallest fair share. For any state Q_l, $l \geq i$, if none of the $S_k \in sessions(e) \cap \mathcal{A}_{Q_i}$ has been marked *done*, then $FS_{Q_l}(e) = FS_{Q_i}(e)$. This and the fact that fair share is non decreasing imply that for every $l \geq i$ such that none of the $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ has been transfered to *done*, $e \in MFSE_{Q_l}(\mathcal{S}_j)$.

Let $S_i \in sessions(e) \cap \mathcal{A}_{Q_i}$ be the first session among all sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$, that becomes *done* in some state Q_k. By Corollary 3.6 the final rate of S_i equals $FS_{Q_i}(e)$.

Since Q_k is the state in which the rate of session S_i receives its final value and S_i is the first session that is marked *done* among all sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$, it holds that $allotted_{Q_k}(e) = allotted_{Q_i}(e) + FS_{Q_i}(e)$, which implies that $FS_{Q_k}(e) = FS_{Q_i}(e)$. Thus, e remains the edge with the minimum fair share among all $e' \in AE_{Q_i}(\mathcal{S}_j)$, even after any session becomes *done*.

Applying repeatedly the preceding arguments, we get that edge e remains the edge with the minimum fair share among all edges $e' \in AE_{Q_i}(\mathcal{S}_j)$ in every state Q_l such that $sessions(e) \cap \mathcal{A}_{Q_l} \neq \emptyset$. ■

We continue to show that, for any bottleneck algorithm, after the rates of all sessions traversing the minimum fair share edge for some particular cluster, have been increased at least once, these sessions must have all become *done*.

Proposition 3.8 *Assume that* Alg *is a bottleneck algorithm. Consider a cluster $\mathcal{S}_j$ that is active in state Q_i. Fix any edge $e \in MFSE_{Q_i}(\mathcal{S}_j)$. After all sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ have been updated once, all these sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ can not further increase their rate (that is, for every $l > k$, $rate_{Q_l}(S_r) = rate_{Q_k}(S_r)$, where Q_k is the network state where the last update occured).*

Proof: Since the initial rate of every session equals 0 and increases of rates occur only by session updating, in order all sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ to become *done*, each of them must be updated at least once. We will prove next, that if all these sessions are updated at least once, then all becomes *done*.

By Proposition 3.7, edge $e \in MFSE_{Q_l}(\mathcal{S}_j)$, for any state Q_l such that $sessions(e) \cap \mathcal{A}_{Q_l} \neq \emptyset$. By Proposition 3.5, for every session $S_r \in sessions(e) \cap \mathcal{A}_{Q_l}$, there is no edge that may cause a decrease of the rate of S_r, while simultaneously the reate of S_r can not be greater than $FS_{Q_l}(e) = FS_{Q_i}(e)$. However, after all these sessions have been updated at least once all of them have rate equal to $FS_{Q_i}(e)$ and they can not increase their rate any more. ■

Finally, we prove that whenever all active sessions have been updated at least once, at least one session in each cluster must have become *done*.

Proposition 3.9 *For any state Q_i, let Q_j, $j > i$, be the first state of the network such that $suffix(i,j) = \mathcal{A}_{Q_i}$. Then, at least one session in each cluster can not increase its rate any more after the state Q_j.*

Proof: Consider a cluster $\mathcal{S}_k$ such that $e \in MFSE_{Q_i}(\mathcal{S}_k)$. Since $suffix(i,j) = \mathcal{A}_{Q_i}$, in state Q_j all active sessions in state Q_i have been updated at least once. Thus, every session $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ has also been updated at least once and by Proposition 3.8 all these sessions can not increase their rate any more. Thus, in any cluster, at least one session can not increase its rate any more after state Q_j, as needed. ∎

4 Deterministic, Oblivious Schedulers

Sections 4.1 and 4.2 include our lower and upper bound, respectively, for oblivious schedulers.

4.1 Lower Bound

We present a lower bound of $\Omega(dn)$ on the convergence complexity of any oblivious algorithm that computes the max-min vector.

Theorem 4.1 *Assume that* Alg *is a deterministic oblivious algorithm that computes the max-min vector. Then,*

$$\mathcal{U}_{\mathsf{Alg}} \geq \frac{dn}{4} + \frac{n}{2}.$$

Sketch of proof: Fix any even integer d, and choose any integer n that is a multiple of d. We construct a network $G = (V, E)$ with a set of sessions $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ laid out on G so that $\mathcal{U}_{\mathsf{Alg}}(G, \mathcal{S}) = dn/4 + n/2$.

The construction uses two sequences of real numbers, b and p (for "bottom" and "potential," respectively), defined as follows: $b_1 = 0$ and $p_1 = 2^p$ for any integer $p \geq 2d$; for each index r, $1 < r \leq n/d$, $b_r = b_{r-1} + p_{r-1}/2$, and $p_r = p_{r-1}/4$.

Partition $\mathcal{S}$ into n/d session clusters $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n/d}$ so that $\mathcal{S}_j = \{S_{d(j-1)+1}, S_{d(j-1)+2}, \ldots S_{d(j-1)+d}\}$, where $1 \leq j \leq n/d$; notice that for each j, $1 \leq j \leq n/d$, $|\mathcal{S}_j| = d$. For each cluster $\mathcal{S}_j$, we construct a network $G_j = (V_j, E_j)$ so that $G = (\cup_{1 \leq j \leq n/d} V_j, \cup_{1 \leq j \leq n/d} E_j)$. For each j, $1 \leq j \leq n/d$, denote $\mathcal{U}_{\mathsf{Alg}}(G_j, \mathcal{S}_j)$ the number of update operations performed by Alg on sessions in cluster $\mathcal{S}_j$ laid out on the network G_j. Clearly, $\mathcal{U}_{\mathsf{Alg}}(G, \mathcal{S}) = \sum_{j=1}^{n/d} \mathcal{U}_{\mathsf{Alg}}(G_j, \mathcal{S}_j)$. The construction of the network G_j proceeds in a sequence of $d/2$ *epochs*; the network $G_j^{(r)} = (V_j^{(r)}, E_j^{(r)})$ is constructed in epoch r, where $G_j = (\cup_{1 \leq r \leq d/2} V_j^{(r)}, \cup_{1 \leq r \leq d/2} E_j^{(r)})$.

For each r, $1 \leq r \leq d/2$, the construction of the network $G_j^{(r)}$ uses b_r and p_r as parameters. The construction is inductive. For the basis case, where $r = 1$, we describe the construction of the network $G_j^{(1)}$. Assume that $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j = i_1, i_2, \ldots, i_d$. Clearly, it must be that $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$ is not empty (that is, each session in $\mathcal{S}_j$ is eventually scheduled by Scheduler), since otherwise at least one session in $\mathcal{S}_j$ would never be updated and thus keep zero rate, which contradicts the fact that Alg computes the max-min vector. By our notation, S_{i_1} and S_{i_d} are the sessions in cluster $\mathcal{S}_j$ which are updated first and last, respectively, in the prefix $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$. We construct $G_j^{(r)}$ as follows: sessions i_1 and i_d share an edge $e_d^{(1)}$ with $cap(e_d^{(1)}) = 2b_1 + p_1/2$; for each l, $2 \leq l \leq d-1$, sessions i_1 and i_l share an edge $e_l^{(1)}$ with $cap(e_l^{(1)}) = 2b_1 + p_1$.

Note that, by the construction of the network $G_j^{(1)}$, each of the sessions $i_1, i_2, \ldots, i_d$ is increased only once during $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$, even if there were more than one occurrences of any particular session in this prefix before removing repetitions. This is so because, on the first update of any particular session other than i_d, the edge traversed by the session becomes saturated; so this session cannot increase its rate any further before session i_d is updated, which is the last session to be updated in $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$. This update on i_d decreases the rate of i_1 by $p_1/4$, thereby increasing the "free" capacity on each other edge by $p_1/4$. Clearly, the execution of the prefix $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$ on $G_j^{(1)}$ results on the marking of i_1 and i_d as *done*, while the remaining $d-2$ sessions in cluster $\mathcal{S}_j$ remain active. Since Alg computes the max-min vector, it follows that the sequence $\mathsf{Scheduler} \downharpoonright \mathcal{S}_j$ is not empty.

Assume inductively that we have constructed $G_j^{(r-1)}$. Let $\mathsf{Scheduler} := \mathsf{Scheduler} \downharpoonright \mathcal{S}_j$. Clearly, the number of sessions in $\mathcal{S}$ that are still active equals now $d - 2(r-1)$, since, inductively, two sessions are marked as *done* in each of the previous $r-1$ epochs.

We prove that Scheduler must contain all indices of sessions that have not yet become *done*; that is, we prove that $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$ is not empty. Assume, by way of contradiction, that Scheduler does not contain all indices of active sessions; let i_k be any index of an active session that is not included in Scheduler. Denote i_1 the index of the session updated first. We construct $G_j^{(r)}$ as follows: sessions i_1 and i_k share an edge $e_k^{(r)}$ with $cap(e_k^{(r)}) = 2\,b_r + p_r/2$; for each l, $2 \leq l \leq d - 2(r-1)$, $l \neq k$, sessions i_1 and i_l share an edge $e_l^{(r)}$ with $cap(e_l^{(r)}) = 2\,b_r + p_r$. In this construction, if session i_k is not updated, none of the sessions i_l, $2 \leq l \leq d - 2(r-1)$ and $l \neq k$, can become *done*, which contradicts the fact that Alg computes the max-min vector. It follows that $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$ is not empty.

Assume, without loss of generality, that $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j = i_1, i_2, \ldots, i_{d-2(r-1)}$. Clearly, S_{i_1} and $S_{i_{d-2(r-1)}}$ are the sessions in cluster $\mathcal{S}_j$ which are updated first and last, respectively, in the prefix $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$. We construct $G_j^{(r)}$ as follows: sessions i_1 and $i_{d-2(r-1)}$ share an edge $e_{d-2(r-1)}^{(r)}$ with $cap(e_{d-2(r-1)}^{(r)}) = 2\,b_r + p_r/2$; for each l, $2 \leq l \leq d - 2(r-1) - 1$, sessions i_1 and i_l share an edge $e_l^{(r)}$ with $cap(e_l^{(r)}) = 2\,b_r + p_r$.

Note that, by the construction of the network $G_j^{(r)}$, each of the sessions $i_1, i_2, \ldots, i_{d-2(r-1)}$ is increased only once during $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$, even if there were more than one occurrences of any particular session in this prefix before removing repetitions. This is so because, on the first update of any particular session other than $i_{d-2(r-1)}$, the edge traversed by the session becomes saturated; so this session cannot increase its rate any further before session $i_{d-2(r-1)}$ is updated, which is the last session to be updated in $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$. This update on $i_{d-2(r-1)}$ decreases the rate of i_1 by $p_r/4$, thereby increasing the "free" capacity on each other edge by $p_r/4$. Clearly, the execution of the prefix $\mathsf{Scheduler} \upharpoonright \mathcal{S}_j$ on $G_j^{(r)}$ results on the marking of i_1 and $i_{d-2(r-1)}$ as *done*, while the remaining $d - 2(r-1) - 2 = d - 2r$ sessions in cluster $\mathcal{S}_j$ remain active.

We prove that, by the way capacities of edges in $E_j^{(r)}$ are chosen, there is no effect of these edges during epoch $r-1$ of the execution on the network $G_j^{(r-1)}$. We do so by showing that the capacity of any edge in $E_j^{(r)}$ exceeds the capacity of every edge in $E_j^{(r-1)}$; this is shown using the definitions of the edge capacities in the construction, and those of the sequences b and p. Roughly speaking, this implies an appropriate ordering of the fair shares of edges in consecutive epochs, which determines that the edges restricting the allocated capacity in each session during a particular epoch are those "constructed" during the epoch.

For each j, $1 \leq j \leq n/d$ and r, $1 \leq r \leq d/2$, denote $\mathcal{U}^{(r)}_{\mathsf{Alg}}(G_j, \mathcal{S}_j) = d - 2(r-1)$ the number of update operations performed on sessions in cluster $\mathcal{S}_j$ during epoch r. Summing up $\mathcal{U}^{(r)}_{\mathsf{Alg}}(G^{(r)}_j, \mathcal{S}_j)$ over all clusters and epochs, we obtain that $\mathcal{U}_{\mathsf{Alg}}(G, \mathcal{S}) = dn/4 + n/2$, as needed. ■

4.2 Upper Bound

We present the algorithm RoundRobin and show:

Theorem 4.2 RoundRobin *computes the max-min allocation vector within $dn/2 + n/2$ update operations.*

The RoundRobin scheduler conducts n scheduling rounds. In each round, each of the n sessions is scheduled in "round-robin" order; The terminator of RoundRobin works as follows. The session currently scheduled for an increase is marked as done if it cannot be increased, while some of its edges is a bottleneck edge such that each of its incident sessions receives a rate equal to the fair share of the edge. Thus, RoundRobin is a bottleneck algorithm, and computes the max-min vector. We continue to prove:

Proposition 4.3 $\mathcal{U}_{\mathsf{RoundRobin}} \leq dn/2 + n/2$

Sketch of proof: All active sessions are updated once in each round, so Proposition 3.9 implies that at least one session of each active cluster is marked as *done* in each round. Consider any cluster $\mathcal{S}_j$. Then, $\mathcal{U}_{\mathsf{RoundRobin}}(\mathcal{S}_j) \leq n_j + n_j - 1 + \ldots + 1 = n_j^2/2 + n_j/2$, and $\mathcal{U}_{\mathsf{RoundRobin}} \leq \sum_{j\geq 1}(n_j^2/2+n_j/2)$. However, $\sum_{j\geq 1} n_j^2 \leq dn$, which implies that $\mathcal{U}_{\mathsf{RoundRobin}} \leq dn/2 + n/2$, as needed. ■

We proceed to establish a lower bound on $\mathcal{U}_{\mathsf{RoundRobin}}$.

Proposition 4.4 $\mathcal{U}_{\mathsf{RoundRobin}} \geq dn/4 + n/2$

Sketch of proof: Fix any integer d, and choose any integer n that is a multiple of d. We construct a network $G = (V, E)$ with a set of n sessions $\mathcal{S} = \{S_1, \ldots, S_n\}$ laid out on G, so that G has session dependency d and $\mathcal{U}_{\mathsf{RoundRobin}}(G) = dn/4 + n/2$.

For each $j \in [n/d]$, fix any even integer $c_j > 10(d+1)$. For any pair of integers j and k, $j \in [n/d]$ and $k \in [d-1]$, sessions $S_{d(j-1)+k}$ and $S_{d(j-1)+k+1}$ share an edge $e_{d(j-1)+k}$ of capacity $c_j(d-k+2)$. Clearly, $\|^*_{\mathcal{S}}$ partitions $\mathcal{S}$ into clusters $\mathcal{S}_1, \ldots, \mathcal{S}_{n/d}$, where for each j, $j \in [n/d]$, $\mathcal{S}_j = \{S_{d(j-1)+1}, \ldots, S_{d(j-1)+d}\}$. It follows that the session dependency of G is d.

During the lth round, session $S_{d(i-1)+j}$ is increased from $c_i(d-j+2)/2$ to $(c_i(d-j+3)/2$ because capacity $d/2$ is available. Afterwards, session $S_{d(i-1)+j+1}$ is increased from $c_i(d-j+1)/2$ to $c_i(d-j+2)/2$, which causes the rate of session $S_{d(i-1)+j}$ to be decreased to $c_i(d-j+2)/2$, $1 \leq i \leq k$ and $1 \leq j \leq d-1$. Furthermore, sessions $S_{di-2(l-2)}$ and $S_{di-2(l-2)-1}$, $i = 1, \ldots, k$, are marked *done*. The number of update operations performed in each cluster $\mathcal{S}_i$ is equal to $d - 2(l-1)$. For each cluster index j and r, $1 \leq r \leq d/2$, denote $\mathcal{U}^{(r)}_{\mathsf{RoundRobin}}(G, \mathcal{S}_j) = d - 2(r-1)$, the number of update operations performed on sessions in cluster $\mathcal{S}_j$ during round r. Summing up $\mathcal{U}^{(r)}_{\mathsf{RoundRobin}}(G, \mathcal{S}_j)$ over all clusters and rounds, we obtain that $\mathcal{U}_{\mathsf{RoundRobin}}(G, \mathcal{S}) = dn/4 + n/2$, as needed. ■

5 A Randomized, Oblivious Scheduler

We present the algorithm Random and show:

Theorem 5.1 Random *computes the max-min allocation vector within an expected number of $O(d \ln n/p_{min})$ update operations.*

Fix probabilities $p_1, p_2, \ldots, p_n$, such that $p_{min} = \min_{1\leq j\leq n} p_j > 0$. The scheduler of Random is randomized; for each j, $1 \leq j \leq n$, session S_j is scheduled for an increase with probability $p_j > 0$. Thus, the scheduler of Random is oblivious. The terminator of Random works as follows. The session currently being scheduled for an increase is marked as *done* if it cannot be increased, while some of its edges is a bottleneck edge such that each of its incident active sessions is receiving a rate equal to the fair share of the edge. Thus, the terminator of Random is bottleneck. Hence, Random computes the max-min allocation vector.

We continue to show upper bounds on the expectation and standard deviation of the number of update operations required for the convergence of Random.

Proposition 5.2 *The following hold on the probability distribution of $\mathcal{U}_{\mathsf{Random}}$:*

(1) $\mathcal{E}(\mathcal{U}_{\mathsf{Random}}) \leq d \ln n/p_{min} + \Theta(d/p_{min})$;

(2) $\sigma(\mathcal{U}_{\mathsf{Random}}) \in O(\sqrt{d}/p_{min})$.

Proof: We start by showing (1). Let $|\mathcal{A}_{Q_i}| = l$ in some state Q_i. By Proposition 3.9, after $F(l)$ update operations, at least one session in each active cluster will become *done*. Let $Q_{j_1}, Q_{j_2}, \ldots, Q_{j_d}$ be states such that $\mathit{suffix}(0, j_1) = \mathcal{A}_{Q_0}$, $\ldots$, $\mathit{suffix}(j_{d-1}, j_d) = \mathcal{A}_{Q_{j_d}}$, correspondingly. Suppose further that $|\mathcal{A}_{Q_{j_1}}| = l_1, |\mathcal{A}_{Q_{j_2}}| = l_2, \ldots, |\mathcal{A}_{Q_{j_d}}| = l_d$. Then, $\mathcal{U}_{\mathsf{Random}} \leq F(l_1) + F(l_2) + \ldots + F(l_d)$.

Bounding $F(l_i)$ corresponds to bounding the random variable X in the generalized coupon collector's problem. Let $S_{r_1}, S_{r_2}, \ldots, S_{r_{l_i}}$ be the active sets in state Q_{j_i}, and let $p_{r_1}, p_{r_2}, \ldots, p_{r_{l_i}}$ be the corresponding scheduling probabilities. Let $p_{l_i,min} = min\{p_{r_1}, p_{r_2}, \ldots, p_{r_{l_i}}\}$. Then, $\mathcal{E}(F(l_i)) \leq \log l_i/p_{l_i,min} + O(1/p_{l_i,min}) \leq \log n/p_{min} + O(1/p_{min})$, since $p_{min} \leq p_{l_i,min}$ and $l_i \leq n$ for each $i \in [d]$. By linearity of expectation, the previous imply:

$$\mathcal{E}(\mathcal{U}_{\mathsf{Random}}) \leq d \log n/p_{min} + O(d/p_{min}) \in O(d \log n/p_{min}).$$

We continue to show (2). For each pair of i and j, corresponding to $F(l_i)$ and $F(l_j)$, the appearence of one of them only "complicates" the other; thus, the random variables are negatively correlated, so that by the FKG inequality [2, Chapter 6], all covariances are negative. Hence, $\mathbf{var}(\mathcal{U}_{\mathsf{Random}}) \leq \mathbf{var}(F(l_1)) + \ldots + \mathbf{var}(F(l_d))$.

By the correspondence between our problem and the generalized coupon collector's problem, we have $\mathbf{var}(F(l_i)) \leq 1/p^2_{l_i,min} \leq 1/p^2_{min}$. Thus, $\mathbf{var}(\mathcal{U}_{\mathsf{Random}}) \leq d/p^2_{min}$. It follows that $\sigma(\mathcal{U}_{\mathsf{Random}}) \in O(\sqrt{d}/p_{min})$, as needed. ■

A significant special case occurs when p_{min} is bounded below by an inverse power n^{-k}; since $p_{min} \leq 1/n$, it must be that $k \geq 1$. In this case, Proposition 5.2 immediately implies:

Corollary 5.3 *Assume that $p_{min} \geq 1/n^k$ for some constant $k \geq 1$. Then, the following hold on the probability distribution of $\mathcal{U}_{\mathsf{Random}}$:*

(1) $\mathcal{E}(\mathcal{U}_{\mathsf{Random}}) \leq n^k|C|\ln n + r(n)$, *where* $r(n) \in O(n^k|C|)$*;*

(2) $\sigma(\mathcal{U}_{\mathsf{Random}}) \in O(n^k\sqrt{|C|})$.

Clearly, the upper bounds on $\mathcal{E}(\mathcal{U}_{\mathsf{Random}})$ and $\sigma(\mathcal{U}_{\mathsf{Random}})$ attain their lowest values when $k = 1$; in this case, for each $i \in [n]$, $p_i \geq p_{min} \geq 1/n$, which implies that $p_i = 1/n$ for each $i \in [n]$, so that:

Corollary 5.4 *Assume that* $p_i = 1/n$ *for each* $i \in [n]$*. Then, the following hold on the probability distribution of* $\mathcal{U}_{\mathsf{Random}}$*:*

(1) $\mathcal{E}(\mathcal{U}_{\mathsf{Random}}) \leq dn\ln n + r(n)$, *where* $r(n) \in O(dn)$*;*

(2) $\sigma(\mathcal{U}_{\mathsf{Random}}) \in O(n\sqrt{d})$.

6 Deterministic, Partially Oblivious Schedulers

We show a lower bound of $\Omega(dn)$ on the convergence complexity of any partially oblivious algorithm that computes the max-min vector.

Theorem 6.1 *Let* Alg *be a deterministic partially oblivious algorithm that computes the max-min vector. Then,*

$$\mathcal{U}_{\mathsf{Alg}} \geq \frac{dn}{4} + \frac{n}{2}.$$

Sketch of proof: Our proof is similar to that of Theorem 4.1; the main complication is due to the fact that the "complete" sequence of sessions produced by *any* partially oblivious scheduler is not known a priori, as has been the case for oblivious schedulers. Hence, our modified proof simultaneously "hand-crafts" in a step-by-step fashion both the sequence of sessions produced by any arbitrary but fixed partially oblivious scheduler (by using all intermediate rates) and the network for which $\Omega(dn)$ update operations are required for convergence to the max-min vector (whose capacities determine the intermediate rates). For purpose of exposition, we sketch the construction of the session sequence $i_1, i_2, \ldots, i_d$ for the first epoch. This sequence is defined indutively. For the basis case, where $l = 1$, set $i_1 = \mathsf{Scheduler}_1(\mathbf{0})$; that is, S_{i_1} is the session scheduled first by Scheduler. Assume inductively that we have defined $i_1, i_2, \ldots, i_{l-1}$; let $Q_1, Q_2, \ldots, Q_{l-1}$ be the sequence of network states right after the scheduling of each of these sessions. Define $i_l = \mathsf{Scheduler}_l(Q_{l-1})$ such that i_l is different from each of $i_1, i_2, \ldots, i_{l-1}$ (we continue applying Scheduler till we reach an index different than those seen so far). Denote $i_1, i_2, \ldots, i_d = \mathsf{Scheduler} \uparrow \mathcal{S}_j$. The rest of the proof closely follows that of Theorem 4.1. ■

Proposition 2.2 implies that the lower bound on the convergence complexity of any partially oblivious algorithm established in Theorem 6.1 is tight (within a constant factor).

7 A Non-Oblivious Scheduler

In this section, we present the algorithm Linear and show:

Theorem 7.1 Linear *computes the max-min allocation vector within exactly n update operations in every case.*

The scheduler of Linear selects any arbitrary edge e from $LMFSE_Q$ and increases one by one all the sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$. Note that the edge e is the one of the smallest fair share among all edges $e' \in AE_{Q_i}$ (by the definition of $LMFSE_{Q_i}$).

Proposition 7.2 Linear *is a bottleneck algorithm.*

Proof: Let $\alpha = Q_0, i_1, Q_1, \ldots, i_l, Q_l, \ldots,$ be any execution of Linear on some network G with session set $\mathcal{S}$. We argue that there is some index $m \geq 1$ such that $Q_m = (\mathbf{R}_{Q_m}, \mathcal{A}_{Q_m}) \in \alpha$ and $\mathcal{A}_{Q_m} = \emptyset$ (i.e., Linear "terminates"). Let i be any step of Linear and e be any edge in $LMFSE_{Q_i}$. Note that all sessions $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ will increase their rate. Since e is the edge with the globally smallest fair share and all $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$ were increased once, Proposition 3.8 implies that all these sessions can not increase their rate any more. Thus, all such sessions will be marked *done*. However, when this happens, e will not belong to $LMFSE_{Q_i}$ any more, since when $sessions(e) \cap \mathcal{A}_{Q_i} = \emptyset$, $e \notin AE_{Q_i}$, and another (new) edge will be examined in subsequent steps of the algorithm. It follows that Linear terminates.

We continue to show that Linear is a bottleneck algorithm. We need to show that when a session S is marked *done*, there is an edge $e \in S$ that is a bottleneck edge in Q_i, and $rate_{Q_i}(S) = FS_{Q_i}(e)$. This holds, indeed, for the edge e with the globally smallest fair share, which is selected by Linear. Clearly, this edge is a bottleneck edge. Furthermore, by Corollary 3.6, for every session $S_r \in sessions(e) \cap \mathcal{A}_{Q_i}$, $rate_{Q_{final}}(S_r) = FS_{Q_i}(e)$. It follows that Linear is a bottleneck algorithm, as needed. ■

By Proposition 7.2, Proposition 3.1 implies that Linear computes the max-min allocation vector. We continue to show a *tight* bound on the convergence complexity of Linear.

Proposition 7.3 $\mathcal{U}_{\mathsf{Linear}} = n$.

Sketch of proof: The upper bound of n on $\mathcal{U}_{\mathsf{Linear}}$ is an immediate consequence of Proposition 3.8: if all active sessions traversing an edge with the smallest fair share among all active edges have been updated, then all of them will become *done*, without any further update operation to be performed on them. The lower bound of n is obvious. ■

8 Conclusion

We have presented a collection of lower and upper bounds on the convergence complexity of optimistic, rate-based flow control algorithms, under varying degrees of the "knowledge" used by the scheduling component of the algorithms. We have shown that the classes of oblivious algorithms and partially oblivious algorithms "coincide" with respect to convergence complexity; moreover, our results imply a convergence complexity separation between partially oblivious algorithms (in particular, oblivious) and non-oblivious algorithms for optimistic, rate-based flow control. Our algorithms demonstrate remarkable combinations of moderate convergence complexity with interesting locality properties; these algorithms improve on the ones in [1]. On the other hand, our lower bounds identify fundamental limitations on the performance of oblivious or even partially oblivious algorithms. Both lower and upper bounds on convergence complexity we have shown have been expressed in terms of session dependency, a critical network parameter which has been introduced by our research as a measure of "locality" into complexity considerations.

Our work has envisioned the problem of optimistic, rate-based flow control as one of *distributed decision-making with incomplete information* (see,e.g. [16, 21]), and has provided yet another demonstration of the economic value of information as a key computational resource in a distributed system.

There remain a number of practical issues still completely untouched by our work. In the first place, we feel that the max-min fairness criterion may be undue in some realistic situations, where there are widely varying demands on different sessions. Second, the limitation to "static" sets of sessions is overly restrictive; it would be extremely significant to extend our model and techniques to handle set-up/take-down of sessions.

Acknowledgements:

We would like to thank the anonymous PODC'97 reviewers for their helpful comments.

References

[1] Y. Afek, Y. Mansour and Z. Ostfeld, "Convergence Complexity of Optimistic Rate Based Flow Control Algorithms," *Proceedings of the 28th Annual ACM Symposium on Theory of Computing,* pp. 89–98, May 1996.

[2] N. Alon, J. H. Spencer and P. Erdos, *The Probabilistic Method,* Wiley-Interscience Series in Discrete Mathematics and Optimization, 1992.

[3] D. P. Bertsekas and R. G. Gallager, *Data Networks,* Prentice Hall, 1987.

[4] F. Bonomi and K. Fendick, "The Rate-Based Flow Control Framework for Available Bit Rate ATM Service," *IEEE/ACM Transactions on Networking,* pp. 24–39, March/April 1995.

[5] L. S. Brakmo and L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communication,* Vol. 13, No. 8, pp. 1465–1480, October 1995.

[6] A. Charny, "An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback," Technical Report MIT/LCS/TR-601, Laboratory for Computer Science, Massachusetts Institute of Technology, April 1994.

[7] W. Dally, "Virtual-Channel Flow Control," *Proceedings of the 17th International Symposium on Computer Architecture,* pp. 60–68, May 1990.

[8] Z. Dziong, M. Juda and L. G. Mason, "A Framework for Bandwidth Management in ATM Networks – Aggressive Equivalent Bandwidth Estimation Approach," *IEEE/ACM Transactions on Networking,* Vol. 5, No. 1, pp. 134–147, February 1997.

[9] E. Gafni, "The Integration of Routing and Flow Control for Voice and Data in Integrated Packet Networks," Ph. D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1982.

[10] E. Gafni and D. Bertsekas, "Dynamic Control of Session Input Rates in Communication Networks," *IEEE Transactions on Automatic Control,* Vol. 29, pp. 1009–1016, 1984.

[11] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications,* Vol. 28, pp. 553–574, 1980.

[12] E. Hahne, "Round-Robin Scheduling for MaxMin Fairness in Data Networks," *IEEE Journal on Selected Areas in Communications,* Vol. 9, No. 7, September 1991.

[13] H. Hayden, "Voice Flow Control in Integrated Packet Networks," Technical Report MIT/LIDS/TH/TR-601, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1981.

[14] J. M. Jaffe, "Bottleneck Flow Control," *IEEE Transactions on Communications,* Vol. 29, No. 7, pp. 954–962, July 1981.

[15] R. Jain, S. Kalyanaraman and R. Viswanathan, "The OSU Scheme for Congestion Avoidance Using Explicit Rate Indication," Technical Report ATM-FORUM/95-0883, The Ohio State University, September 1994.

[16] P. C. Kanellakis and C. H. Papadimitriou, "The Complexity of Distributed Concurrency Control," *SIAM Journal on Computing,* Vol. 14, No. 1, pp. 52–75, February 1985.

[17] G. Kesidis, *ATM Network Performance,* Kluwer Academic Publishers, 1996.

[18] H. T. Kung and R. Morris, "Credit-Based Flow Control for ATM Networks," *IEEE/ACM Transactions on Networking,* pp. 40–48, March/April 1995.

[19] J. Le Boudec, "The Asynchronous Transfer Mode: A Tutorial," *Computer Networks and ISDN Systems,* Vol. 24, No. 4, pp. 279–309, 1992.

[20] J. Mosley, "Asynchronous Distributed Flow Control Algorithms," Ph. D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1984.

[21] C. H. Papadimitriou and M. Yannakakis, "On the Value of Information in Distributed Decision Making," *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing,* pp. 61–64, August 1991.

[22] G. Polya, "Eine Wahrscheinlichkeitsaufgabe zur Kundenwerbung," *Zeitschrift Fur Angewandte Mathematik und Mechanik,* Vol. 10, pp. 96-97, 1930.

[23] M. D. Prycher, *Asynchronous Transfer Mode, Solution for Broadband ISDN,* Ellis Horwood, 1992.

[24] K. S. Sin and H. H. Tzeng, "Adaptive Proportional Rate Control (APRC) with Intelligent Congestion Indication," Technical Report ATM-FORUM/94-0888, University of California at Irvine, September 1994.