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

Panagiota Fatourou, Marios Mavronicolas, Paul Spirakis

2004
Efficiency of Oblivious Versus Non-Oblivious Schedulers for Optimistic, Rate-Based Flow Control*

Panagiota Fatourou† Marios Mavronicolas‡ Paul Spirakis§

(December 16, 2004)

*A preliminary version of this work appears in the Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing (Santa Barbara, California, August 1997), pp. 139–148, ACM, New York, 1997. This work has been partially supported by the ESPRIT Program of the European Union under the Long Term Research Project ALCOM-IT (contract number 20244), by the IST Program of the European Union under Projects ALCOM-FT (contract number IST-1999-14186), FLAGS (contract number IST-2001-33116) and DELIS (contract number 001907), by funds from the Program Ε.Π.Ε.Α.Ε.Κ. II of the Greek Ministry of Education, by research funds at University of Cyprus, and by AT&T/NCR research funds.

†Department of Computer Science, University of Ioannina, P. O. Box-1186, Ioannina GR-45110, Greece. Part of the work of this author was performed while at Department of Computer Engineering and Informatics, University of Patras and at Computer Technology Institute, while at Max-Planck Institute für Informatik, and while visiting Department of Computer Science, University of Cyprus. Email: faturu@cs.uoi.gr

‡Department of Computer Science, University of Cyprus, P. O. Box 20537, Nicosia CY-1678, Cyprus. Part of the work of this author was done while at Department of Computer Science and Engineering, University of Connecticut, while visiting University of Patras and Computer Technology Institute, and while at AT&T Labs – Research, as a visitor to the DIMACS Special Year on Networks. Email: mavronic@ucy.ac.cy

§Department of Computer Engineering and Informatics, University of Patras, Rion, Patras GR-26500, Greece and Computer Technology Institute, P. O. Box 1122, Patras GR-26110, Greece. Email: spirakis@cti.gr
Abstract. Two important performance parameters of distributed, rate-based flow control algorithms are their locality and convergence complexity. The former is characterized by the amount of global knowledge that is available to their scheduling mechanisms, while the latter is defined as the number of update operations performed on rates of individual sessions until max-min fairness is reached. Optimistic algorithms allow any session to intermediately receive a rate larger than its max-min fair rate; bottleneck algorithms finalize the rate of a session only if it is restricted by a certain, highly congested link of the network. In this work, we present a comprehensive collection of lower and upper bounds on convergence complexity, under varying degrees of locality, for optimistic, bottleneck, rate-based flow control algorithms.

Say that an algorithm is oblivious if its scheduling mechanism uses no information of either the session rates or the network topology. We present a novel, combinatorial construction of a capacitated network, which we use to establish a fundamental lower bound of \( \frac{dn}{4} + \frac{n}{2} \) on the convergence complexity of any oblivious algorithm, where \( n \) is the number of sessions laid out on a network, and \( d \), the session dependency, is a measure of topological dependencies among sessions. Moreover, we devise a novel simulation proof to establish that, perhaps surprisingly, the lower bound of \( \frac{dn}{4} + \frac{n}{2} \) on convergence complexity still holds for any partially oblivious algorithm, in which the scheduling mechanism is allowed to use information about session rates, but is otherwise unaware of network topology.

On the positive side, we prove that the lower bounds for oblivious and partially oblivious algorithms are both tight. We do so by presenting optimal oblivious algorithms, which converge after \( \frac{dn}{4} + \frac{n}{2} \) update operations are performed in the worst case. To complete the picture, we show that linear convergence complexity can indeed be achieved if information about both session rates and network topology is available to schedulers. We present a counter-example, non-oblivious algorithm, which converges within an optimal number of \( n \) update operations.

Our results imply a surprising convergence complexity collapse of oblivious and partially oblivious algorithms, and a convergence complexity separation between (partially) oblivious and non-oblivious algorithms for optimistic, bottleneck rate-based flow control.

Key Words. distributed algorithms, lower bounds, rate-based flow control, max-min fairness, convergence complexity, oblivious, partially oblivious, and non-oblivious algorithms, bottleneck algorithms.

1 Introduction

In many modern communication networks, a connection between different users is established by a session, a virtual circuit of infinite duration that involves a fixed path between a source and a destination. The role of flow control algorithms is to alleviate throughput degradation, unfairness, deadlocks, and failures by preventing buffer overflow from arising in situations where the externally injected load is larger than what can be accommodated even with optimal routing (see, e.g., [2, Chapter 6] or [3, 6, 11, 14, 15, 16, 21]). 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 still maintaining fairness between different sessions (see, e.g., [1, 2, 4, 5, 10, 12, 15, 17]).

For a range of settings including both high-speed networks and Internet applications, max-min fairness [1, 2, 5, 6, 7, 12, 13, 14, 15, 16, 18] has emerged as a widely accepted formulation of fairness for flow control: roughly speaking, max-min fairness requires that it be impossible for any session to receive an infinitesimally larger rate on the account of a session with a smaller or equal rate [14, 15, 16, 20]. Call max-min fair rates those achieving max-min fairness.

Any rate-based flow control algorithm can be classified into one of two broad classes, conservative and optimistic, according to the way in which rates of sessions are adjusted. Conservative algorithms (see, e.g., [2, Chapter 6]) do not provide for decreases to the rates of sessions; in contrast, optimistic algorithms allow decreases, so that rates may go up and down and a session can intermediately receive a rate larger than its final. Afek et al. [1] introduced optimistic algorithms and advocated them to fit better than the conservative ones into real dynamic networks; indeed, in such networks, new sessions may join in, so that it is desirable to incrementally adjust rates by decreasing the rates of some of them.

A crucial component of a rate-based flow control algorithm is its scheduler, the abstract mechanism it uses to decide which session’s rate to adjust next. Apparently, it is desirable that the scheduler does not require global knowledge of either the session rates or the network topology. Clearly, no-knowledge schedulers are not only more efficient in terms of communication and computation, but they also adjust more easily to dynamic changes in network topology. So, one important performance parameter of a rate-based flow control algorithm is its locality, measured by the amount of global knowledge required by the scheduler.

Call a scheduler that uses no information of either the session rates or the network topology an oblivious scheduler. On the opposite extreme, a non-oblivious scheduler requires full knowledge of both session rates and network topology. There is, in addition, an important 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*. Clearly, 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. Afek et al. [1, Sections 4 & 5] present two interesting, partially oblivious schedulers called *GlobalMinSched* and *LocalMinSched*, respectively; these schedulers always choose for an increase the session whose rate is the *globally* and *locally* smallest, respectively.

A second crucial component of a rate-based flow control algorithm is its *terminator*, the abstract mechanism it uses to decide which sessions to terminate at each specific instant.\(^*\) *Bottleneck* terminators [15] finalize the rate of a session, thereby terminating the session, only if its rate is restricted on some particular network link that allows for the least possible share of bandwidth to each session traversing it; call such a link a *bottleneck edge* [15].

When a session is scheduled for an increase, its rate is increased by the minimum possible share of bandwidth along its path; this may involve possible decreases to the rates of crossing sessions (see, e.g., [5, 6, 7, 17, 18]). The *convergence complexity* of a rate-based flow control algorithm is the number of individual rate adjustments performed in the worst case until the algorithm quiesces and rates have reached max-min fairness (so that they do not change any further). Thus, convergence complexity models the number of rounds of communication and local computation needed for convergence to max-min fairness, so that it is another significant performance parameter of such distributed algorithms.

We measure convergence complexity in terms of a particular, simple abstraction of rate adjustments, called *update operation*, which was introduced in [1, Section 2.1]. In essence, an *update* operation atomically adjusts the rates of a group of neighboring sessions in a fair and optimistic way.\(^1\) We note that some of the essential intricacies encountered when designing practical, distributed flow-control algorithms [4, 5, 6, 7, 17, 18, 21], include scheduling the rate adjustments, minimizing the communication and converging to fairness fast. Although we do not address in this work implementation issues for the model used, we feel that it captures most of these intricacies. (For a discussion of such issues for this model, see [8].)

---

\(^*\)In all of our discussion, a terminated session is meant to be one that has reached its final (max-min fair) rate, so that its rate value will not change any further.

\(^1\)In more detail, an *update* operation increases, if possible, the rate of one session, so that it becomes the session with the maximum rate on some particular link that allows the minimum possible increase. The new value is a function of the link capacity and the rates of other sessions traversing the link. Conflicting sessions whose rates exceed the new value have their rates decreased to the new value, while rates of other sessions remain unchanged. These adjustments saturate that particular link.
This work presents a comprehensive collection of bounds on convergence complexity under varying degrees of locality for optimistic, bottleneck, rate-based flow control algorithms; more specifically, we show upper and lower bounds (Sections 5 and 7, respectively) on convergence complexity for oblivious, partially oblivious, and non-oblivious such algorithms. The lower bounds demonstrate that achieving oblivious, or even partially oblivious, scheduling, and therefore locality, necessarily imposes a non-constant, multiplicative overhead on convergence complexity. These are the first general lower bounds ever shown on the convergence of rate-based flow control algorithms. In addition, our algorithms span a wide spectrum of convergence complexity bounds and locality properties, while they improve significantly upon all previous optimistic algorithms with respect to the combination of these two performance measures. To establish these upper bounds, we offer several new basic properties and tools for the design and analysis of optimistic, bottleneck, rate-based flow control algorithms (Section 4); these properties significantly extend and strengthen the corresponding ones shown in [1].

Our bounds are expressed in terms of \( n \), the total number of sessions laid out on the network, and a new parameter \( d \), called session dependency, through which we derive more accurate bounds on convergence complexity. Specifically, \( d \) is the maximum number of sessions that share an edge either directly or indirectly. In the rest of this paper, we will focus on optimistic and bottleneck algorithms; we shall sometimes refer to them simply as algorithms.

Our first major result is a fundamental lower bound on the convergence complexity of oblivious algorithms. Its proof relies on constructing, given any arbitrary oblivious algorithm, a specific network, as a function of the algorithm’s scheduler, so that if sessions are scheduled on this network according to the scheduler and the algorithm computes the max-min fair rates, then \( \Omega(dn) \) update operations are required before convergence (Section 6). The construction is novel, purely combinatorial and of independent interest. We rely on the algorithm being optimistic and bottleneck for showing that convergence is slow.

Although intuition may suggest that knowledge of session rates can be crucial to performance, we surprisingly establish that the lower bound of \( \Omega(dn) \) we show on the convergence complexity of oblivious algorithms applies also to partially oblivious algorithms. We use a powerful simulation proof to simulate any partially oblivious scheduler on some network by an oblivious scheduler on the same network. The simulation inherits the \( \Omega(dn) \) lower bound shown for oblivious algorithms down to partially oblivious algorithms.

We show a matching upper bound on the convergence complexity of oblivious and partially oblivious algorithms. We present a class of oblivious algorithms, called \( d \)-Epoch, with convergence complexity \( \Theta(dn) \); an example is algorithm RoundRobin, which uses the simple and
natural idea of scheduling sessions in a round-robin order. We note that the partially oblivious algorithms \texttt{GlobalMin} and \texttt{LocalMin} from [1, Sections 4 & 5] also achieve convergence complexity $\Theta(dn)$, which implies the tightness of our lower bound for partially oblivious algorithms; however, any \texttt{d-Epoch} algorithm improves over these two algorithms since it is oblivious.

At this point, it is only natural to ask whether it is possible to overcome the $\Theta(dn)$ barrier on convergence complexity achievable by oblivious or partially oblivious algorithms, possibly at the cost of sacrificing locality. Perhaps not very 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 present a counter-example, non-oblivious and optimistic algorithm, called \texttt{Linear}, that achieves an exact bound of $n$ on convergence complexity; \texttt{Linear} follows the earlier idea of selecting and conservatively increasing the rate of any session that traverses the most highly congested link in the network (see, e.g., [2, Section 6.4.2] or [14]). We emphasize that \texttt{Linear}, although theoretically efficient, is clearly impractical.

Our work differs at first from earlier work on rate-based flow control carried out in the data networks community (see, e.g., [2, 3, 10, 11, 14, 15, 16, 21]) in adopting the optimistic approach introduced in [1]. We have been able to show that certain classical scheduling policies, such as round-robin [12, 13] or scheduling a session that traverses the most congested link [14], are correctly applicable in the optimistic framework. Most interestingly, we have shown the first general lower bounds on the convergence complexity of rate-based flow control algorithms; these imply that some scheduling policies are provably superior over some other policies, in terms of either convergence complexity or locality (or both), and even optimal.

2 Model

Many of our definitions formalize and refine corresponding ones from [1] and [2, Chapter 6]. A \textit{communication network} is a graph $G = (V, E)$, where vertices and edges represent switches and links, respectively. A set $S$ of $n \geq 1$ \textit{sessions} is laid out on $G$. Each session $S_i$ (also denoted as $i$) passes through a sequence of edges and has a \textit{rate} $r(S_i)$. The vector $r = (r(S_1), \ldots, r(S_n))$ is the \textit{rate vector}. Each edge has a \textit{capacity} $c(e) > 0$, which the sum of the rates of the sessions traversing it cannot exceed; when this sum equals the capacity, the edge is \textit{saturated}. In a \textit{max-min fair rate vector} [14, 15, 16, 20], an increase to the rate of any session requires either exceeding an edge capacity or decreasing the rate of another session with equal or lower rate.

The communication network is abstracted as a state machine. Each state $Q = (r_Q, A_Q)$ consists of a \textit{rate vector} $r_Q$ and a set $A_Q \subseteq S$ of \textit{active sessions}. Intuitively, an active session
is one that has not yet reached its max-min fair rate. Denote $D_Q = S \setminus A_Q$ the set of done sessions in state $Q$. In the initial state $Q^0$, all sessions are active and have zero rates. A state is final if all sessions have reached their max-min fair rates.

An operation defines a procedure to compute new rates for a set of sessions on the basis of their old rates. Formally, an operation is a function $\text{operation}$ that takes as input a session $S_i$ and a state $Q$, and outputs new rates for $S_i$ and all sessions $S_j$ sharing edges with $S_i$. Say that operation is conservative if it decreases no individual rate; else operation is optimistic.

We will consider a specific optimistic operation, called $\text{update}$, which we describe below. For a state $Q$ and each edge $e$ traversed by $S_i$, $\Delta_Q(i,e)$ is the maximum amount by which $r_Q(S_i)$ can be increased (without exceeding the capacity of $e$) while decreasing down to the increased rate of $S_i$, the rates of other active sessions passing through $e$ and exceeding the increased rate. Notice that these rate adjustments saturate $e$. Intuitively, $\Delta_Q(i,e)$ is the maximum amount by which $r_Q(S_i)$ can be increased in a fair manner if edge $e$ were the only edge constraining $S_i$.

Finally, $S_i$’s rate is increased by $\Delta_Q(i,e)$, called the increase for $S_i$ in $Q$, which is the minimum of these maximum amounts over all edges that $S_i$ passes through. In addition, each (active) session $S_j$ sharing an edge with $S_i$ has its rate decreased down to the new rate $r(S_i)$ (unless it is already less). We remark that the rates computed by $\text{update}$ saturate the edge(s) realizing $\Delta_Q(i,e)$, but no other edges. Notice also that the $\text{update}$ operation uses only local information with respect to session $S_i$ [16, Section IV]. From now on, we will only consider optimistic algorithms that employ the $\text{update}$ operation.

A scheduler decides the order of sessions on which to perform the $\text{update}$ operation. Formally, a scheduler (cf. [1]) is a function $\text{Sched}$ that maps a pair $(G,S)$, a state $Q$ and a state index $l \geq 1$ to a session $i = \text{Sched}((G,S),Q,l)$. A terminator is a function $\text{Term}$ that maps a pair $(G,S)$ and a state $Q$ to a set of sessions $T = \text{Term}((G,S),Q) \subseteq A_Q$; intuitively, $\text{Term}$ decides which active sessions will be terminated (and their rates will not change any further). An algorithm is a pair $\text{Alg} = (\text{Sched}, \text{Term})$. The classification of operations into conservative and optimistic leads to the corresponding classification of algorithms in a natural way.

An oblivious scheduler uses no knowledge of either the topology of the network or the rates and statuses (active or done) of sessions. Thus, an oblivious scheduler is a (finite or infinite) sequence $\text{Sched} = i_1, i_2, \ldots$, where for each $l \geq 1$, $i_l \in [n]$. A partially oblivious scheduler uses no knowledge of the topology of the network, while it may use knowledge of the rates (and statuses) of sessions. Notice that any oblivious scheduler is also partially oblivious, but not vice versa. A non-oblivious scheduler is a scheduler that is not partially oblivious. The classification of schedulers into oblivious, partially oblivious and non-oblivious leads to the
corresponding classification of algorithms in a natural way.

For any edge $e \in E$ and state $Q$, the allotted capacity of $e$ in $Q$ [1, Section 2.1], denoted $\text{allot}_Q(e)$, is the total rate already allocated to done sessions passing through the edge. Clearly, $\sum_{S \in A_Q[e]} r_Q(S) \leq c(e) - \text{allot}_Q(e)$, where $A_Q | e$ denotes the set of active sessions traversing $e$ in $Q$. For any state $Q$ and for any edge $e$ with $|A_Q | e| > 0$ active sessions, the fair share of $e$ in state $Q$ [1, Section 2.1], denoted $FS_Q(e)$, is defined to be $FS_Q(e) = \frac{c(e) - \text{allot}_Q(e)}{|A_Q | e|}$; intuitively, $FS_Q(e)$ is the per session share of the portion of the capacity of edge $e$ which has not yet been allocated to sessions done in state $Q$ that traverse the edge. For any state $Q$, a bottleneck edge for $Q$ [15] is an edge $e$ such that for each active session passing through $e$, $FS_Q(e)$ is the minimum fair share over all edges traversed by the session.

A terminator $\text{Term}$ is bottleneck [15] if for any state $Q$ and session $S$, $S \in \text{Term} ((G,S),Q)$ if (and only if) there exists an edge $e$ traversed by $S$ such that (1) $e$ is a bottleneck edge for $Q$, and (2) $r_Q(S) = FS_Q(e)$. Whenever such an edge $e$ exists for state $Q$ and $\text{Term}$ is bottleneck, we will say that $e$ causes the termination of session $S$ in $Q$. Say that an algorithm $\text{Alg}$ is bottleneck if $\text{Term}$ is.

The execution $\alpha$ of $\text{Alg}$ on $(G,S)$ is an infinite sequence of alternating states and session indices $\alpha = Q_0, i_1, Q_1, \ldots, i_t, Q_t, \ldots$, satisfying the following conditions:

1. $Q_0 = Q^\text{in}$ and $A_{Q_0} = A_{Q_1} = S$;
2. for each $l \geq 1$, $i_l = \text{Sched} ((G, S), Q_{l-1}, l-1)$;
3. for each $l \geq 1$, if $i_l \in D_{Q_l}$ or $\Delta_{Q_{l-1}}(i_i) = 0$, then $r_{Q_l} = r_{Q_{l-1}}$ and $A_{Q_{l+1}} = A_{Q_l}$, else
   - (a) $r_{Q_l}(i_l) := r_{Q_{l-1}}(i_l) + \Delta_{Q_{l-1}}(i_l)$;
   - (ii) for each session $i \in A_{Q_l}, i \neq i_l, $ such that $S_i \cap S_{i_l} \neq \emptyset$,
     $r_{Q_l}(i) := \min\{r_{Q_{l-1}}(i), r_{Q_{l-1}}(i_l) + \Delta_{Q_{l-1}}(i_l)\}$;
4. $A_{Q_{l+1}} = A_{Q_l} \setminus \text{Term} ((G,S), Q_l)$.

Call each state $Q_l$ in an execution $\alpha$, where $l \geq 1$, a reachable state. Say that $\text{Alg}$ computes the max-min fair rate vector if for each execution of $\text{Alg}$ a final state $Q$ is reachable such that $r_Q$ is a max-min fair rate vector.

The number of update operations in any execution $\alpha$ of $\text{Alg}$ is the number of state indices $l \geq 1$ such that $i_l \in A_{Q_{l-1}}$. The convergence complexity of $\text{Alg}$ on network $G$ with session set $S$, denoted $U_{\text{Alg}}((G,S))$, is the number of operations in the execution of $\text{Alg}$ on $G$ with $S$. The convergence complexity of $\text{Alg}$, denoted $U_{\text{Alg}}$, is the maximum, over all pairs $(G,S)$, of the convergence complexity of $\text{Alg}$ on $G$ with $S$. 

8
3 Notation

We collect here some notation that will be used in most of the following sections. For each edge \( e \) and for any set of sessions \( S' \subseteq S \), denote \( S' \mid e \) the set of sessions in \( S' \) traversing \( e \). We will sometimes treat a session \( S_i \) as the set of its links; so, for any edge \( e \) traversed by \( S_i \), we will write \( e \in S_i \). For an edge \( e \) and for a rate vector \( r \), denote \( r \mid e \) the restriction of \( r \) to sessions traversing \( e \). We will sometimes abuse notation by writing \( r_{AQ} \) and \( r_{DQ} \) to denote the restriction of \( r_Q \) to active and done sessions, respectively, in \( Q \).

Define the share-an-edge relation on \( S \), denoted \( \| S \), as follows. For any pair of sessions \( S_i, S_j \in S \), \( S_i \| S S_j \) if \( S_i \) and \( S_j \) traverse a common edge. The transitive closure of \( \| S \) is an equivalence relation on \( S \), which partitions \( S \) into equivalence classes \( S_1, \ldots, S_c \), called clusters, where \( 1 \leq c \leq n \). The session dependency \( d \) is the maximum size of a cluster. Call a cluster \( S_j \) an active cluster in state \( Q \) if it contains at least one session that remains active in \( Q \).

The set of active edges of the cluster \( S_j \) in state \( Q \), denoted \( AE_Q(S_j) \), contains all edges of the network traversed by at least one active session in \( Q \) that belongs to \( S_j \). The set of active edges of the network in state \( Q \), denoted \( AE_Q \), contains all edges of the network traversed by at least one active session in \( Q \). The residual capacity of edge \( e \) in state \( Q \), denoted \( \text{resid}_Q(e) \), is the difference between the capacity of \( e \) and the total rate of sessions traversing \( e \) in \( Q \).

The set of edges with minimum fair share for cluster \( S_j \) in state \( Q \) is defined as \( MFSE_Q(S_j) = \{ e \in AE_Q(S_j) \mid FS_Q(e) \leq FS_Q(e') \text{ for each } e' \in AE_Q(S_j) \} \). The set of edges with minimum fair share in state \( Q \), denoted \( MFSE_Q \), is defined to be \( MFSE_Q = \bigcup_{1 \leq j \leq c} MFSE_Q(S_j) \).

An execution fragment \( \alpha \) of \( \text{Alg} \) is a contiguous subsequence of some execution of \( \text{Alg} \) starting with the state \( \text{first}(\alpha) \); if \( \alpha \) is finite, it ends with the state \( \text{last}(\alpha) \). For each execution (resp., execution fragment) \( \alpha \) of \( \text{Alg} \), the schedule \( \sigma(\alpha) \) is the sequence of session indices in \( \alpha \). If \( \alpha \) is a finite execution fragment and \( \alpha' \) is any execution fragment such that \( \text{first}(\alpha') = \text{last}(\alpha) \), then \( \alpha \cdot \alpha' \) is the concatenation of \( \alpha \) and \( \alpha' \), eliminating the duplicate occurrence of \( \text{last}(\alpha) = \text{first}(\alpha') \).

For any index \( l \geq 1 \), the preceding state of \( Q_l \) in execution \( \alpha \), denoted \( \overline{Q}_{l \mid} \), is the state \( Q_{l-1} \); for any index \( l \geq 0 \), the successor state of \( Q_l \) in \( \alpha \), denoted \( Q_{l \mid} \), is the state \( Q_{l+1} \). For any indices \( l \) and \( l' > l \), write \( Q_l \xrightarrow{\alpha} Q_{l'} \) to denote that \( Q_l \) precedes \( Q_{l'} \) in \( \alpha \); moreover, write \( Q_l \xrightarrow{\alpha} Q_{l'} \) if additionally \( Q_l \) and \( Q_{l'} \) may coincide. For any index \( l \geq 1 \), we will sometimes abuse language and say that session \( l \) is scheduled in front of state \( Q_l \). For any state \( Q \), denote \( i_Q \) the session scheduled in front of \( Q \). The least schedule for \( Q_l \) in \( \alpha \), denoted \( \hat{l} \), is the index of the earliest state following \( Q_l \) by which all sessions that remain active in \( Q_l \) have been scheduled at least once, or infinite if no such state exists. Define \( \hat{l} \mid e \) in the natural way.
4 Bottleneck Algorithms

In this section, we present basic properties of bottleneck algorithms, which will be useful in the sequel. These properties refer to an execution \( \alpha = Q_0, i_1, Q_1, \ldots, i_l, Q_l, \ldots \) of any bottleneck algorithm. To prove these properties, some more general properties are also proved in Section 4.1.

4.1 Preliminaries

We study how several quantities of interest change during an execution. We first state an immediate consequence of the definitions of allotted capacity and execution; we then prove a similar simple fact that will be helpful in later proofs.

**Lemma 4.1** For each integer \( l \geq 1 \), and for any edge \( e \),

\[
\text{allot}_{Q_l}(e) = \text{allot}_{Q_{l-1}}(e) + \sum_{i \in (A_{Q_{l-1}} \setminus A_{Q_l}) \mid e} r_{Q_{l-1}}(i) .
\]

**Lemma 4.2** For any integers \( l_0 \) and \( l \), \( 0 \leq l_0 < l \), and for any edge \( e \),

\[
\sum_{i \in (A_{Q_{l_0}} \setminus A_{Q_l}) \mid e} r_{Q_{l_0}}(i) = \text{allot}_{Q_l}(e) - \text{allot}_{Q_{l_0}}(e) .
\]

**Proof:** Clearly, \( i \in (A_{Q_{l_0}} \setminus A_{Q_l}) \mid e \) if and only if \( i \in (D_{Q_l} \setminus D_{Q_{l_0}}) \mid e \). Hence, it follows that

\[
\sum_{i \in (A_{Q_{l_0}} \setminus A_{Q_l}) \mid e} r_{Q_l}(i) = \sum_{i \in (D_{Q_l} \setminus D_{Q_{l_0}}) \mid e} r_{Q_l}(i) = \sum_{i \in D_{Q_l} \mid e} r_{Q_l}(i) - \sum_{i \in D_{Q_{l_0}} \mid e} r_{Q_l}(i) \quad \text{(since } D_{Q_{l_0}} \subseteq D_{Q_l})
\]

\[
= \text{allot}_{Q_l}(e) - \text{allot}_{Q_{l_0}}(e),
\]
as needed.

We continue to prove that the saturation of an edge depends in a critical way on how rates of sessions traversing the edge compare to each other.

**Lemma 4.3** For any integer \( l_0 \geq 0 \), assume that edge \( e \) is active in state \( Q_{l_0} \). Then, for each integer \( l \geq l_0 \), the following hold:
Proof: We start by proving (1). Clearly,
\[
\sum_{i \in A_{Q_0} | e} r_{Q_i}(i) = \sum_{i \in A_{Q_0} | e} r_{Q_i}(i) - \sum_{i \in (A_{Q_0} \setminus A_{Q_0}) | e} r_{Q_i}(i) = \sum_{i \in A_{Q_0} | e} FS_{Q_0}(e) - \left( allot_{Q_i}(e) - allot_{Q_0}(e) \right) \quad \text{(by assumption and Lemma 4.2)}
\]

Condition (2) is proved in an almost identical way. (The only difference is that now the assumption for (2) and Lemma 4.2 imply that \( \sum_{i \in A_{Q_0} | e} r_{Q_i}(i) \).)

We finally prove (3). Assume otherwise; so, there exists some session \( k \in A_{Q_0} | e \) such that \( r_{Q_i}(k) > FS_{Q_0}(e) \), while for each \( i \in A_{Q_0} | e, i \neq k \), \( r_{Q_i}(i) \geq FS_{Q_0}(e) \). Then,
\[
\sum_{i \in A_{Q_0} | e} r_{Q_i}(i) > \sum_{i \in A_{Q_0} | e} FS_{Q_0}(e) + \sum_{i \in A_{Q_0} | e, i \neq k} allot_{Q_i}(e) + allot_{Q_0}(e) \quad \text{(by assumption)}
\]

so that \( \sum_{i \in A_{Q_0} | e} r_{Q_i}(i) > c(e) - allot_{Q_i}(e) \). A contradiction. \( \blacksquare \)
The next claim follows directly from the definitions of bottleneck edge and fair share.

**Lemma 4.4** Let $e$ and $e'$ be bottleneck edges for state $Q$ such that $A_Q \cap e \cap A_Q \cap e' \neq \emptyset$. Then, $FS_Q(e) = FS_Q(e')$.

The following (easy to prove) claim is a direct consequence of the definitions of a bottleneck edge and a minimum fair share edge for some particular cluster.

**Lemma 4.5** For any state $Q$ and cluster $S_j$, consider an edge $e \in MFSE_Q(S_j)$. Then, $e$ is a bottleneck edge for $Q$.

We are interested in algorithms for which a final state is reachable for each possible execution. Bottleneck algorithms (whether conservative or optimistic) enjoy a related, interesting property [14].

**Proposition 4.6 (Hayden [14])** Assume that $Alg$ is a bottleneck algorithm. Then, for any reachable final state $Q$ of $Alg$, $r_Q$ is a max-min fair rate vector.

Proposition 4.6 implies that in order to show that any given bottleneck algorithm computes the max-min fair rate vector, it suffices to prove that it reaches a final state. We continue with an interesting monotonicity property of fair share.

**Lemma 4.7 (Afek et. al. [1])** Assume that $Alg$ is a bottleneck algorithm. Then, for each integer $l \geq 1$ and for any edge $e \in AE_{Q_l}$, $FS_{Q_l}(e) \geq FS_{Q_{l-1}}(e)$.

### 4.2 Properties of Bottleneck Edges

We strengthen Lemma 4.7 for the special case of bottleneck edges. We present a collection of invariant properties for any edge that becomes bottleneck in the course of an execution of a bottleneck algorithm (whether conservative or optimistic).

**Proposition 4.8 (Invariants of bottleneck edge)** Assume that $Alg$ is bottleneck. For any integer $l_0 \geq 0$, fix any edge $e$ that is a bottleneck edge for $Q_{l_0}$. Then, for any integer $l \geq l_0$ such that $e \in AE_{Q_l}$, the following hold:
(1) \( FS_{Q_l}(e) = FS_{Q_{l_0}}(e) \);

(2) \( e \) is a bottleneck edge for \( Q_l \);

(3) for any session \( i \in (A_{Q_{l_0}} \setminus A_{Q_{l+1}}) \mid e, r_{Q_l}(i) = FS_{Q_{l_0}}(e) \).

Roughly speaking, Proposition 4.8 establishes that no change in the fair share of an active edge occurs once the edge has become bottleneck, so that the edge remains bottleneck; moreover, the final rate of any active session traversing it is equal to this constant fair share.

**Proof:** The proof is by induction on \( l \). For the basis case where \( l = l_0 \), condition (1) holds trivially, condition (2) holds by assumption, and condition (3) holds by definition of a bottleneck algorithm. Assume inductively that for some integer \( l \geq l_0 \), the claims hold for any integer \( l' \) where \( l_0 \leq l' < l \). For the induction step, we show that the claims hold for integer \( l \).

We start by proving condition (1). Clearly,

\[
FS_{Q_l}(e) = \frac{c(e) - allot_{Q_l}(e)}{|A_{Q_l}|e} = \frac{c(e) - allot_{Q_{l-1}}(e) - \sum_{i \in (A_{Q_{l-1}} \setminus A_{Q_l}) \mid e} r_{Q_{l-1}}(i)}{|A_{Q_l}|e} \quad \text{(by Lemma 4.1)}
\]

\[
= \frac{|A_{Q_{l-1}}|e FS_{Q_{l-1}}(e) - \sum_{i \in (A_{Q_{l-1}} \setminus A_{Q_l}) \mid e} r_{Q_{l-1}}(i)}{|A_{Q_l}|e} \quad \text{(by definition of fair share)}.
\]

Consider any session \( i \in (A_{Q_{l-1}} \setminus A_{Q_l}) \mid e \). Since \( A_{Q_{l-1}} \subseteq A_{Q_{l_0}} \), this implies that \( i \in (A_{Q_{l_0}} \setminus A_{Q_l}) \mid e \), so that by the induction hypothesis (condition (3)), \( r_{Q_{l-1}}(i) = FS_{Q_{l_0}}(e) \). So,

\[
FS_{Q_l}(e) = \frac{|A_{Q_{l-1}}|e FS_{Q_{l-1}}(e) - \sum_{i \in (A_{Q_{l-1}} \setminus A_{Q_l}) \mid e} FS_{Q_{l_0}}(e)}{|A_{Q_l}|e} = \frac{|A_{Q_{l-1}}|e FS_{Q_{l-1}}(e) - |(A_{Q_{l-1}} \setminus A_{Q_l})|e FS_{Q_{l_0}}(e)}{|A_{Q_l}|e} = \frac{|A_{Q_{l-1}}|e FS_{Q_{l-1}}(e) - |(A_{Q_{l-1}} \setminus A_{Q_l})|e FS_{Q_{l_0}}(e)}{|A_{Q_l}|e} = FS_{Q_{l_0}}(e),
\]
Proposition 4.9 (Invariant of minimum fair share edge)

We establish that the edge remains a minimum fair share edge (as long as it is active).

Proof: We start by proving a simple invariant property for any edge that becomes a minimum fair share edge. For any integer \( l \geq 0 \), consider any edge \( e \in MFSE_{Q_i_0}(S_j) \) for some active cluster \( S_j \) in \( Q_{i_0} \). Then, for any integer \( l \geq l_0 \) such that \( e \in AE_{Q_i} \), \( e \in MFSE_{Q_i}(S_j) \).

Proof: Consider any edge \( e' \in AE_{Q_i}(S_j) \); clearly, \( e' \in AE_{Q_{i_0}}(S_j) \). Since \( e \in MFSE_{Q_{i_0}}(S_j) \), it follows that \( FS_{Q_{i_0}}(e) \leq FS_{Q_{i_0}}(e') \). By Lemma 4.5, \( e \) is a bottleneck edge for \( Q_{i_0} \); thus, by

4.3 Properties of Minimum Fair Share Edges

We start by proving a simple invariant property for any edge that becomes a minimum fair share edge for any particular cluster in the course of an execution of a bottleneck algorithm. We establish that the edge remains a minimum fair share edge (as long as it is active).

Proposition 4.9 (Invariant of minimum fair share edge) Assume that \( Alg \) is bottleneck. For any integer \( l_0 \geq 0 \), fix any edge \( e \in MFSE_{Q_{i_0}}(S_j) \) for some active cluster \( S_j \) in \( Q_{i_0} \). Then, for any integer \( l \geq l_0 \) such that \( e \in AE_{Q_i}, e \in MFSE_{Q_i}(S_j) \).

Proof: Consider any edge \( e' \in AE_{Q_i}(S_j) \); clearly, \( e' \in AE_{Q_{i_0}}(S_j) \). Since \( e \in MFSE_{Q_{i_0}}(S_j) \), it follows that \( FS_{Q_{i_0}}(e) \leq FS_{Q_{i_0}}(e') \). By Lemma 4.5, \( e \) is a bottleneck edge for \( Q_{i_0} \); thus, by
Proposition 4.8 (condition (2)), \( FS_{Q_1}(e) = FS_{Q_{l_0}}(e) \). By Lemma 4.7, \( FS_{Q_{l_0}}(e') \leq FS_{Q_1}(e') \). So, \( FS_{Q_1}(e) \leq FS_{Q_1}(e') \). Since \( e' \) is arbitrary, it follows that \( e \in MFSE_{Q_1}(S_j) \)

Similarly to Proposition 4.8, Proposition 4.9 holds for any bottleneck algorithm (whether conservative or optimistic) as well. However, the rest of the properties established in this section require the assumption of optimistic, bottleneck algorithms. We first prove a safety property for any edge that becomes a minimum fair share edge for any particular cluster, during the execution of an optimistic, bottleneck algorithm.

**Proposition 4.10 (Safety property of minimum fair share edge)** Assume that \( Alg \) is optimistic and bottleneck. For any integer \( l_0 \geq 0 \), fix any edge \( e \in MFSE_{Q_{l_0}}(S_j) \) for some active cluster \( S_j \) in \( Q_{l_0} \). Consider any session \( i \in A_{Q_{l_0}} \mid e \) such that \( r_{Q_{l_0}}(i) \geq FS_{Q_{l_0}}(e) \). Then, for any integer \( l \geq l_0 \), \( r_{Q_l}(i) \geq FS_{Q_{l_0}}(e) \).

Proposition 4.10 considers any (active) session traversing a minimum fair share edge; roughly speaking, it establishes that no decrease to its rate below this particular minimum fair share is possible if the rate is initially no less than the minimum fair share.

**Proof:** The proof is by induction on \( l \). For the basis case where \( l = l_0 \), the claim holds by our assumption. Assume inductively that for some integer \( l > l_0 \), \( r_{Q_{l-1}}(i) \geq FS_{Q_{l_0}}(e) \). For the induction step, we show that \( r_{Q_l}(i) \geq FS_{Q_{l_0}}(e) \).

Assume first that \( r_{Q_l}(i) \geq r_{Q_{l-1}}(i) \). By the induction hypothesis, this implies that \( r_{Q_l}(i) \geq FS_{Q_{l_0}}(e) \), as needed. So assume that \( r_{Q_l}(i) < r_{Q_{l-1}}(i) \). By definition of execution and update operation, this implies that \( S_i \) intersects the session \( S_{i_l} \), scheduled in front of state \( Q_l \); moreover, \( r_{Q_l}(i) = r_{Q_l}(i_l) \). Let \( e' \) be an edge such that \( \Delta_{Q_{l-1}}(i_l) = \Delta_{Q_{l-1}}(i_l, e') \). We prove:

**Lemma 4.11** \( r_{Q_l}(i_l) \geq FS_{Q_l}(e') \).

**Proof:** Assume otherwise; so, \( r_{Q_l}(i_l) < FS_{Q_l}(e') \). By definition of update operation, \( e' \) is saturated in \( Q_l \); moreover, for any session \( k \in A_{Q_{l_0}} \mid e' \), \( r_{Q_l}(i_l) \geq r_{Q_l}(k) \). Thus, \( FS_{Q_l}(e') \geq r_{Q_l}(k) \). Lemma 4.3 (condition (2)) implies that \( e' \) is not saturated in \( Q_l \). A contradiction.

Since \( r_{Q_l}(i) = r_{Q_l}(i_l) \), Lemma 4.11 implies that \( r_{Q_l}(i) \geq FS_{Q_l}(e') \). By Lemma 4.7, \( FS_{Q_l}(e') \geq FS_{Q_{l_0}}(e') \), so that \( r_{Q_l}(i) \geq FS_{Q_{l_0}}(e') \). Since \( e \in MFSE_{Q_{l_0}}(S_j) \), \( FS_{Q_{l_0}}(e') \geq FS_{Q_{l_0}}(e) \). It follows that \( r_{Q_l}(i) \geq FS_{Q_{l_0}}(e) \), as needed.
The next claim complements Proposition 4.10 by giving a corresponding liveness property.

**Proposition 4.12 (Liveness property of minimum fair share edge)** Assume that Alg is optimistic and bottleneck. For any integer \( l_0 \geq 0 \), fix any edge \( e \in MFSE_{Q_{l_0}}(S_j) \) for some active cluster \( S_j \) in \( Q_{l_0} \), such that \( \hat{l}_0 | e < \infty \). Consider any session \( i \in A_{Q_{l_0}} | e \). Then, for any integer \( l \geq \hat{l}_0 | e \), \( r_{Q, l}(i) \geq FS_{Q_{l_0}}(e) \).

Proposition 4.12 considers any active session traversing a minimum fair share edge; roughly speaking, it establishes that eventually, once all active sessions traversing this minimum fair share edge have been scheduled at least once, the rate of the session will be no less than this particular minimum fair share.

**Proof:** We start with an informal outline of the proof. We consider the point of the execution following state \( Q_{l_0} \) where session \( i \) is scheduled; clearly, that point comes no later than when all sessions have been scheduled at least once. We establish that at this point, the rate of \( S_i \) is no less than the fair share of edge \( e \) in state \( Q_{l_0} \). We also argue that \( e \) remains a minimum fair share edge beyond state \( Q_{l_0} \); this allows us to exploit the safety property of minimum fair share edges in order to argue that the rate of \( S_i \) will subsequently remain no less than the fair share of \( e \) in state \( Q_{l_0} \). We now present the details of the formal proof.

Since \( e \in MFSE_{Q_{l_0}}(S_j) \), Lemma 4.5 implies that \( e \) is a bottleneck edge for state \( Q_{l_0} \). Since \( i \in A_{Q_{l_0}} | e \), it follows by definition of \( \hat{l}_0 | e \) that there exists a least index \( l' \), \( l_0 < l' \leq \hat{l}_0 | e \) such that \( i \) is scheduled in front of state \( Q_{l'} \). We proceed by case analysis.

1. Assume first that \( i \) is not active in state \( Q_{l'} \). Since \( i \in A_{Q_{l_0}} \), there exists an index \( l'' \), \( l_0 \leq l'' < l' \) such that \( i \in \text{Term}(Q_{l''}) \). Since \( \text{Term} \) is bottleneck, it follows that there exists some edge \( e' \) traversed by session \( i \) that is a bottleneck edge for state \( Q_{l''} \), and \( r_{Q_{l''}}(i) = FS_{Q_{l''}}(e') \). Since \( i \in A_{Q_{l''}} \) and \( i \) traverses \( e \), \( e \) is an active edge at \( Q_{l''} \). By Proposition 4.8 (conditions (1) and (2)), \( FS_{Q_{l''}}(e) = FS_{Q_{l_0}}(e) \), and \( e \) is a bottleneck edge for \( Q_{l''} \). By Lemma 4.4, \( FS_{Q_{l''}}(e') = FS_{Q_{l_0}}(e) \), so that \( FS_{Q_{l''}}(e') = FS_{Q_{l_0}}(e) \). It follows that \( r_{Q_{l''}}(i) = FS_{Q_{l_0}}(e) \). Take now any integer \( l \geq \hat{l}_0 | e \). Clearly, \( l \geq l'' \). Since \( i \in \text{Term}(Q_{l''}) \), \( r_{Q_{l''}}(i) = r_{Q_{l''}}(i) = FS_{Q_{l_0}}(e) \), which establishes the claim in this case.

2. Assume now that \( i \) is active in state \( Q_{l'} \). Since \( i \) traverses edge \( e \), it follows that \( e \) is active in state \( Q_{l'} \). By Proposition 4.8 (conditions (1) and (2)), \( FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e) \), and \( e \) is a bottleneck edge for \( Q_{l'} \). We continue to prove:
Lemma 4.13 \( r_{Q_{l'}}(i) \geq FS_{Q_{l_0}}(e) \).

Proof: Assume, by way of contradiction, that \( r_{Q_{l'}}(i) < FS_{Q_{l_0}}(e) \). Let \( e' \) be an edge such that \( \Delta_{Q_{l'}}(i) = \Delta_{Q_{l'}}(i, e') \). By definition of update operation, \( e' \) is saturated in state \( Q_{l'} \). Since \( e \) is a bottleneck edge for state \( Q_{l'} \), and \( i \) traverses both \( e \) and \( e' \), \( FS_{Q_{l'}}(e) \leq FS_{Q_{l'}}(e') \). Since \( FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e) \), this implies that \( FS_{Q_{l_0}}(e) \leq FS_{Q_{l'}}(e') \). Since \( r_{Q_{l'}}(i) < FS_{Q_{l_0}}(e) \), it follows that \( r_{Q_{l'}}(i) < FS_{Q_{l'}}(e') \). By definition of update operation, for any session \( k \in A_{Q_{l'}} \mid e \), \( r_{Q_{l'}}(k) \leq r_{Q_{l'}}(i) \), so that \( r_{Q_{l'}}(k) < FS_{Q_{l'}}(e') \). It follows by Lemma 4.3 (condition (2)) that \( e' \) is not saturated in state \( Q_{l'} \). A contradiction.

Take now any integer \( l \geq \hat{l}_0 \mid e \). Clearly, \( l \geq l' \). Since \( e \) is a minimum fair share edge for \( Q_{l_0} \), Proposition 4.9 implies that \( e \) is a minimum fair share edge for \( Q_{l'} \) as well. Moreover, by Lemma 4.13, \( r_{Q_{l'}}(i) \geq FS_{Q_{l_0}}(e) \). Since \( FS_{Q_{l'}}(e) = FS_{Q_{l_0}}(e) \), this implies that \( r_{Q_{l'}}(i) \geq FS_{Q_{l'}}(e) \). It follows by Proposition 4.10 (taking \( l' \) for \( l_0 \)) that \( r_{Q_{l'}}(i) \geq FS_{Q_{l_0}}(e) \), which establishes the claim in this case.

The proof of Proposition 4.12 is now complete.

4.4 Termination Properties

The first property considers active sessions in any particular cluster that traverse a minimum fair share edge; it is established that once each such session has been scheduled at least once, all of these sessions must have become done.

Proposition 4.14 (Termination of all sessions) Assume that Alg is optimistic and bottleneck. For any integer \( l_0 \geq 0 \), fix any edge \( e \in MFSE_{Q_{l_0}}(S_j) \), for some active cluster \( S_j \) in \( Q_{l_0} \), such that \( \hat{l}_0 \mid e < \infty \). Then, for any session \( i \in A_{Q_{l_0}} \mid e \), \( i \in D_{Q_{l_0} \mid e} \).

Proof: We start with an informal outline of the proof. We consider any session active in state \( Q_{l_0} \), and we argue that after all sessions have been scheduled at least once, the session will receive rate equal to the fair share of \( e \) in \( Q_{l_0} \). We will exploit the fact that \( e \) is a bottleneck edge for \( Q_{l_0} \) in order to argue that \( e \) remains bottleneck subsequently, and that its fair share does not change. Since Alg is a bottleneck algorithm, this will be sufficient for deducing that the session has reached its final rate. We now present the details of the formal proof.

Fix any session \( i \in A_{Q_{l_0}} \mid e \). We start by proving:
Lemma 4.15  \( r_{Q_{t0}}(i) = FS_{Q_{t0}}(e) \)

**Proof:** Assume, by way of contradiction, that \( r_{Q_{t0}}(i) \neq FS_{Q_{t0}}(e) \). By Proposition 4.12, \( r_{Q_{t0}}(i) \geq FS_{Q_{t0}}(e) \). It follows that \( r_{Q_{t0}}(i) > FS_{Q_{t0}}(e) \). We proceed by case analysis.

Assume first that there exists no session \( k \in A_{Q_{t0}} \mid e \) with \( k \neq i \); thus, \( A_{Q_{t0}} \mid e = \{i\} \). Denote \( Q_l \) the latest state in execution \( \alpha \), such that \( Q_{t0} \overset{\alpha}{\rightarrow} Q_l \overset{\alpha}{\rightarrow} Q_{t0}[e] \), and \( A_{Q_l} \neq \emptyset \). Thus, \( A_{Q_l} \mid e = A_{Q_{t0}} \mid e \) and \( allot_{Q_l}(e) = allot_{Q_{t0}}(e) \). Clearly,

\[
\sum_{k \in A_{Q_{t0}}[e]} r_{Q_{t0}}(k) = r_{Q_l}(i) \quad \text{(since } A_{Q_{t0}} \mid e = \{i\})
\]

\[
= r_{Q_{t0}}(i) \quad \text{(by definition of state } Q_l \text{)}
\]

\[
> FS_{Q_{t0}}(e) \quad \text{(by assumption)}
\]

\[
= c(e) - allot_{Q_{t0}}(e) \quad \text{(by definition of fair share and since } \left| A_{Q_{t0}} \mid e \right| = 1)
\]

\[
= c(e) - allot_{Q_l}(e) .
\]

A contradiction.

Assume now that there exists some session \( k \in A_{Q_{t0}} \mid e \) with \( k \neq i \). By Proposition 4.12, \( r_{Q_{t0}}(k) \geq FS_{Q_{t0}}(e) \). Together with \( r_{Q_{t0}}(i) > FS_{Q_{t0}}(e) \), this implies a contradiction to Lemma 4.3 (condition (3)), and the proof is now complete. \( \blacksquare \)

We continue with the proof of Proposition 4.14. In case \( i \in D_{Q_l} \) for some state \( Q_l \) in \( \alpha \) such that \( Q_{t0} \overset{\alpha}{\rightarrow} Q_l \overset{\alpha}{\rightarrow} Q_{t0}[e] \), the definition of execution implies that \( i \in D_{Q_{t0}[e]} \). So, assume that for each state \( Q_l \) in execution \( \alpha \) such that \( Q_{t0} \overset{\alpha}{\rightarrow} Q_l \overset{\alpha}{\rightarrow} Q_{t0}[e] \), \( i \in A_{Q_l} \). Denote \( Q_l \) the latest state in execution \( \alpha \) such that both \( Q_{t0} \overset{\alpha}{\rightarrow} Q_l \overset{\alpha}{\rightarrow} Q_{t0}[e] \) and \( r_{Q_l}(i) = r_{Q_{t0}}(i) \).

Since \( e \in MFSE_{Q_{t0}} \), Lemma 4.5 implies that \( e \) is a bottleneck edge for \( Q_{t0} \). Since \( i \in A_{Q_l} \) and \( i \) traverses edge \( e \), it follows that \( A_{Q_l} \mid e \neq \emptyset \). Hence, Proposition 4.8 (conditions (1) and (2)) implies that \( FS_{Q_l}(e) = FS_{Q_{t0}}(e) \), and \( e \) is a bottleneck edge for \( Q_l \). By Lemma 4.15, \( r_{Q_{t0}}(i) = FS_{Q_l}(e) \). It follows that \( r_{Q_l}(i) = FS_{Q_l}(e) \).

In total, \( e \) is a bottleneck edge for state \( Q_l \), traversed by session \( i \) for which \( r_{Q_l}(i) = FS_{Q_l}(e) \). Since Alg is bottleneck, it follows that \( i \in D_{Q_l} \). Since \( Q_l \overset{\alpha}{\rightarrow} Q_{t0}[e] \), it follows that \( i \in D_{Q_{t0}[e]} \). \( \blacksquare \)

The final termination property is a direct consequence of Proposition 4.14. Roughly speaking, we establish that scheduling any sequence of sessions that includes all currently active ones must result in finalizing the rate of at least one active session per cluster.
Proposition 4.16 (Termination of at least one session per cluster) Assume that Alg is optimistic and bottleneck. For any integer \( l_0 \geq 0 \) such that \( A_{Q_{l_0}} \neq \emptyset \) and \( l_0 < \infty \), fix any active cluster \( S_j \) in \( Q_{l_0} \). Then, there exists some session \( S_i \in S_j \cap A_{Q_{l_0}} \) such that \( S_i \in D_{Q_{l_0}} \).

Proof: Since \( S_j \) is active in \( Q_{A_{Q_{l_0}}} \), it follows that \( MFSE_{Q_{l_0}}(S_j) \neq \emptyset \). Fix any edge \( e \in MFSE_{Q_{l_0}}(S_j) \), and consider any session \( i \in A_{Q_{l_0}} | e \). By Proposition 4.14, \( i \in D_{Q_{l_0}} \).

5 Upper Bounds

5.1 Oblivious Algorithms

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

Theorem 5.1 (Upper bound for oblivious algorithms) RoundRobin computes the max-min fair rate vector within \( \frac{dn}{2} + \frac{n}{2} \) update operations.

The scheduler of RoundRobin conducts scheduling rounds. In each round, each of the \( n \) sessions is scheduled in round-robin order. Moreover, RoundRobin is bottleneck. By definition of RoundRobin, each session is scheduled once in each round. Thus, Proposition 4.16 implies that at least one session per cluster becomes done in each round. Since each cluster contains at most \( d \) sessions, all sessions are done after \( d \) rounds, whence the network enters a final state. So, Proposition 4.6 immediately implies that RoundRobin computes the max-min fair rate vector.

We now establish an upper bound on the convergence complexity of RoundRobin. Since at least one session per cluster becomes done in each round, at most \( |S_j| - l + 1 \) update operations are executed in round \( l, 1 \leq l \leq |S_j| \), on sessions in any cluster \( S_j \). Summing up over all clusters and rounds yields that \( U_{RoundRobin} \leq \sum_{j \geq 1} \sum_{1 \leq l \leq d} \min \{0, (|S_j| - l + 1)\} \leq \sum_{j \geq 1} \sum_{1 \leq l \leq |S_j|} (|S_j| - l + 1) = \sum_{j \geq 1} \frac{|S_j||S_j|+1}{2} \leq \frac{dn}{2} + \frac{n}{2} \).

The algorithm RoundRobin extends naturally to a class of bottleneck algorithms \( d \)-Epoch. The scheduler \( d\text{-EpochSched} = d\text{-EpochSched}_1 \ldots d\text{-EpochSched}_d d\text{-EpochSched}' \), of \( d \)-Epoch is a sequence such that for each index \( r \), \( 1 \leq r \leq d \), all sessions are included in \( d\text{-EpochSched}_r \). An identical argument to the one applied to RoundRobin immediately implies:

Theorem 5.2 (Upper bound for oblivious algorithms) \( d \)-Epoch computes the max-min fair rate vector within \( \frac{dn}{2} + \frac{n}{2} \) update operations.
5.2 Non-Oblivious Algorithms

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

**Theorem 5.3 (Upper bound for non-oblivious algorithms)** Linear computes the max-min fair rate vector within exactly \( n \) update operations.

The scheduler of Linear maintains an active edge of minimum fair share and schedules all active sessions traversing it in any order. Once it finishes, it chooses any other (active) edge of minimum fair share, and so on. Moreover, Linear is bottleneck.

Consider any state \( Q_{l_0} \) and an arbitrary edge \( e \in \text{MFSE}_{Q_{l_0}} \). Clearly, \( e \in \text{MFSE}_{Q_{l_0}}(S_j) \), for some cluster \( S_j \). By definition of Linear, each session traversing \( e \) is scheduled exactly once, so that the state \( Q_{l_0}^{e} \) is reached; by Proposition 4.14, each such session is done in state \( Q_{l_0}^{e} \). It follows that all sessions eventually become done and a final state is reached. Hence, Proposition 4.6 immediately implies that Linear computes the max-min fair rate vector.

Linear incurs \( n \) update operations. Recall that all rates are initially zero. Since all capacities exceed zero, all rates in a max-min fair rate vector exceed zero as well. Since each update increases the rate of exactly one session, it follows that at least \( n \) update operations are needed, so that \( U_{\text{Alg}} \geq n \), for every \( \text{Alg} \). Thus, Linear is optimal.

6 Network Construction

We present a generic, combinatorial construction of a network associated with any sequence \( \text{Seq} = i_1, i_2, \ldots \), where for each \( l \geq 1 \), \( i_l \in [n] \). For any sequence \( \text{Seq} \), denote \( |\text{Seq}| \) the length of \( \text{Seq} \); an infinite sequence has infinite length. For a sequence \( \text{Seq} \) of session indices, and for any set of sessions \( S' \subseteq S \), denote \( \text{Seq} \mid S' \) the restriction of \( \text{Seq} \) to indices of sessions in \( S' \). Denote \( \text{Seq} \mid S' \) the shortest prefix of \( \text{Seq} \) that includes the indices of all sessions in \( S' \), in the order they appear in this prefix of \( \text{Seq} \mid S' \) and with no repetitions; in case no such prefix exists, \( \text{Seq} \mid S' \) results from \( \text{Seq} \mid S' \) by removing repetitions, while, however, preserving the order of the indices. Denote \( \text{Seq} \downarrow S' \) the remaining suffix of \( \text{Seq} \mid S' \). For example, if \( \text{Seq} = 1, 5, 4, 2, 1, 3, 3, 5, 4 \) and \( S' = \{1, 3, 4\} \subseteq \{1, 2, 3, 4, 5\} \), then \( \text{Seq} \mid S' = 1, 4, 1, 3, 3, 3, 4 \), \( \text{Seq} \downarrow S' = 1, 4, 3 \) and \( \text{Seq} \downarrow S' = 3, 3, 4 \).

Fix any even integer \( d \), and choose any integer \( n \) that is a multiple of \( d \). We construct a network \( G = G(\text{Seq}) = (V(\text{Seq}), E(\text{Seq})) \), as a function of \( \text{Seq} \), with a set of sessions

\( ^* \)Standard “padding” techniques can be used to handle the case where \( n \) is not a multiple of \( d \).
$S = \{S_1, S_2, \ldots, S_n\}$ laid out on $G$. We will use, for assigning capacities to network edges, two (finite) sequences of real numbers, $b$ and $p$ (for bottom and potential, respectively), each of length $\frac{n}{2}$, defined recursively as follows.

- $b_0 = p_0 = 0$, $b_1 = 0$ and $p_1 = 2^p$ for some integer $p \geq d$;
- for each index $r$, $1 < r \leq \frac{n}{2}$, $b_r = b_{r-1} + \frac{p_{r-1}}{4}$ and $p_r = \frac{p_{r-1}}{4}$.

Partition $S$ into $\frac{n}{2}$ clusters $S_1, S_2, \ldots, S_{n/d}$ so that for each $j$, $1 \leq j \leq \frac{n}{2}$, $S_j$ contains sessions $S_{(j-1)d+1}, S_{(j-1)d+2}, \ldots, S_{(j-1)d+d}$; notice that $|S_j| = d$. For each cluster $S_j$, we construct a network $G_j = G_j (\text{Seq} | S_j) = (V_j (\text{Seq} | S_j), E_j (\text{Seq} | S_j))$, with $S_j$ laid out on $G_j (\text{Seq} | S_j)$, so that $G (\text{Seq}) = (\bigcup_{1 \leq j \leq n/d} V_j (\text{Seq} | S_j), \bigcup_{1 \leq j \leq n/d} E_j (\text{Seq} | S_j))$: thus, each individual network $G_j$ is a function of the sequence $\text{Seq} | S_j$, and the network $G$ is the resulting composition.

The construction of the network $G_j$ proceeds in a sequence of $\frac{n}{2}$ epochs; in epoch $r$, $1 \leq r \leq \frac{n}{2}$, the network $G_j^{(r)} = (V_j^{(r)}, E_j^{(r)})$ is constructed, so that $G_j = (\bigcup_{1 \leq r \leq d/2} V_j^{(r)}, \bigcup_{1 \leq r \leq d/2} E_j^{(r)})$: thus, the network $G_j$ is the composition of the individual networks $G_j^{(r)}$.

For each $r$, $1 \leq r \leq \frac{n}{2}$, the construction of $G_j^{(r)}$ uses $b_r$ and $p_r$ as parameters. It also uses the following sets and sequences:

- a set of indices $I_j^{(r)} \subseteq S_j$ such that $|I_j^{(r)}| = d - 2(r - 1)$;
- a sequence $\text{Seq}_j^{(r)}$, which is a suffix of $\text{Seq} | S_j$;
- a set $\{i_j^{(r)}, i_l^{(r)}\} \subseteq I_j^{(r)}$: roughly speaking, $i_j^{(r)}$ and $i_l^{(r)}$ will be defined to be the first and last, respectively, indices of $\text{Seq}_j^{(r)} \uparrow I_j^{(r)}$, or some of them will be set to arbitrary indices from $I_j^{(r)}$ in case $|\text{Seq}_j^{(r)} \uparrow I_j^{(r)}| < 2$.

These sets and sequences are inductively defined as follows. For the basis case where $r = 1$, $I_j^{(1)} := S_j$, $\text{Seq}_j^{(1)} := \text{Seq} | S_j$ and the set $\{i_j^{(1)}, i_l^{(1)}\}$ is defined as follows:

1. Assume first that $\text{Seq}_j^{(1)} = \lambda$, the empty sequence, so that $\text{Seq}_j^{(1)} \uparrow I_j^{(1)} = \lambda$; then, fix $i_j^{(1)}$ and $i_l^{(1)}$ to be any arbitrary indices in $I_j^{(1)}$.

2. Assume now that $\text{Seq}_j^{(1)} \neq \lambda$, so that $\text{Seq}_j^{(1)} \uparrow I_j^{(1)} \neq \lambda$; there are two cases to consider.

   a. Take first $|\text{Seq}_j^{(1)} \uparrow I_j^{(1)}| = 1$; then, $i_j^{(1)} := \text{Seq}_j^{(1)} \uparrow I_j^{(1)}$ and fix $i_l^{(1)}$ to be any arbitrary index in $I_j^{(1)} \setminus \{i_j^{(1)}\}$.
b. Take finally \( |\text{Seq}_j^{(1)} \upharpoonright I_j^{(1)}| > 1 \), so that \( \text{Seq}_j^{(1)} \upharpoonright I_j^{(1)} = i_f, \ldots, i_l \); then, \( i_f^{(1)} := i_f \) and \( i_l^{(1)} := i_l \).

Informally, \( I_j^{(1)} \) contains indices of all sessions in cluster \( S_j \), while \( \text{Seq}_j^{(1)} \) is the restriction of \( \text{Seq} \) to indices of sessions in cluster \( S_j \); moreover, \( i_f^{(1)} \) and \( i_l^{(1)} \) are the indices of sessions in cluster \( S_j \) appearing first and last, respectively, in \( \text{Seq}_j^{(1)} \upharpoonright I_j^{(1)} \) or some of them will be set to arbitrary indices if \( \text{Seq}_j^{(1)} \upharpoonright I_j^{(1)} \) misses any of such indices.

Assume inductively that we have defined \( I_j^{(r-1)} \), \( \text{Seq}_j^{(r-1)} \) and \( \{i_f^{(r-1)}, i_l^{(r-1)}\} \), for some integer \( r \), where \( 2 \leq r \leq \frac{d}{2} \). For the induction step, we show how to construct \( I_j^{(r)} \), \( \text{Seq}_j^{(r)} \) and \( \{i_f^{(r)}, i_l^{(r)}\} \). Define \( I_j^{(r)} := I_j^{(r-1)} \setminus \{i_f^{(r-1)}, i_l^{(r-1)}\} \), \( \text{Seq}_j^{(r)} := \left( \text{Seq}_j^{(r-1)} \upharpoonright I_j^{(r-1)} \right) \upharpoonright I_j^{(r)} \), and the set \( \{i_f^{(r)}, i_l^{(r)}\} \) is defined through a case analysis identical to the one for the basis case.

1. Assume first that \( \text{Seq}_j^{(r)} = \lambda \), so that \( \text{Seq}_j^{(r)} \upharpoonright I_j^{(r)} = \lambda \); then, fix \( i_f^{(r)}, i_l^{(r)} \) to be any arbitrary indices in \( I_j^{(r)} \).

2. Assume now that \( \text{Seq}_j^{(r)} \neq \lambda \), so that \( \text{Seq}_j^{(r)} \upharpoonright I_j^{(r)} \neq \lambda \); there are two cases to consider.

   a. Take first \( |\text{Seq}_j^{(r)} \upharpoonright I_j^{(r)}| = 1 \); then, \( i_f^{(r)} := \text{Seq}_j^{(r)} \upharpoonright I_j^{(r)} \) and fix \( i_l^{(r)} \) to be any arbitrary index in \( I_j^{(r)} \);

   b. Take finally \( |\text{Seq}_j^{(r)} \upharpoonright I_j^{(r)}| > 1 \), so that \( \text{Seq}_j^{(r)} \upharpoonright I_j^{(r)} = i_f, \ldots, i_l \); then, \( i_f^{(r)} := i_f \) and \( i_l^{(r)} := i_l \).

Informally, \( I_j^{(r)} \) is obtained by removing \( i_f^{(r-1)} \) and \( i_l^{(r-1)} \) from \( I_j^{(r-1)} \), while \( \text{Seq}_j^{(r)} \) results from \( \text{Seq}_j^{(r-1)} \) by chopping off its shortest prefix that includes all indices in \( I_j^{(r-1)} \), and restring the remaining suffix to indices in \( I_j^{(r)} \); moreover, \( i_f^{(r)} \) and \( i_l^{(r)} \) are the indices of sessions in \( I_j^{(r)} \) that appear first and last, respectively, in this suffix, or some of them will be set to arbitrary indices in case this suffix misses any such indices.

Since two different sessions are extracted from \( S_j \) in each of the \( \frac{d}{2} \) epochs, all \( d \) sessions in \( S_j \) will be eventually extracted. We now describe the construction of \( O_j^{(r)} \), \( 1 \leq r \leq \frac{d}{2} \):

- sessions \( i_f^{(r)} \) and \( i_l^{(r)} \) traverse some edge \( e_{i_l^{(r)}}^{(r)} \) with \( c (e_{i_l^{(r)}}^{(r)}) = 2b_r + \frac{d}{2} \);
- for each \( i \in I_j^{(r)} \setminus \{i_f^{(r)}, i_l^{(r)}\} \), sessions \( i_f^{(r)} \) and \( i \) traverse some edge \( e_{i_l}^{(r)} \) with \( c (e_{i_l}^{(r)}) = 2b_r + p_r \).
Informally, $i_j^{(r)}$ shares an edge with every other session in $I_j^{(r)}$; the capacity of the edge shared with $i_l^{(r)}$ is the smallest, while all other capacities are equal. All other sessions traverse only the edge shared with $i_j^{(r)}$. We finally state an easy to prove property of the construction.

**Lemma 6.1** For each integer $r$ where $1 < r \leq \frac{d}{2}$, for each index $r'$ where $1 \leq r' < r$, let $e^{(r)}$ and $e^{(r')}$ be any edges in $E_j^{(r)}$ and $E_j^{(r')}$, respectively. Then, $c\left(e^{(r)}\right) > c\left(e^{(r')}\right)$.

**Example 6.1** Fix $d = 6$ and choose $n = 12$. Consider the (infinite) elevator sequence

$$\text{ElevSched} = 1, 2, \ldots, 11, 12, 12, 11, \ldots, 2, 1, \ldots, 1, 2, \ldots, 11, 12, 12, 11, \ldots, 2, 1, \ldots$$

We construct the network $G = G(\text{ElevSched}) = (V(\text{ElevSched}), E(\text{ElevSched}))$, as a function of ElevSched, with a set of sessions $\mathcal{S} = \{S_1, \ldots, S_{12}\}$ laid out on $G$. Partition $\mathcal{S}$ into $\frac{\mathcal{S}}{6} = 2$ clusters $\mathcal{S}_1$ and $\mathcal{S}_2$, each containing six sessions, so that $\mathcal{S}_1 = \{S_1, \ldots, S_6\}$ and $\mathcal{S}_2 = \{S_7, \ldots, S_{12}\}$. Fix $p = 10$, so that $b_1 = 0$ and $p_1 = 2^{10} = 1024$.

For the basis case where $r = 1$, which corresponds to the first epoch, $I_1^{(1)} = \mathcal{S}_1$, and

$$\text{ElevSched}_1^{(1)} = \text{ElevSched} \mid \mathcal{S}_1 = 1, 2, \ldots, 6, 6, 5, \ldots, 1, \ldots, 1, 2, \ldots, 6, 5, \ldots, 1, \ldots$$

Thus, $\text{ElevSched}_1^{(1)} \upharpoonright \mathcal{S}_1^{(1)} = 1, 2, \ldots, 6$, so that $i_j^{(1)} = 1$ and $i_l^{(1)} = 6$. We continue to describe the construction of the network $G_1^{(1)}$:

- sessions 1 and 6 traverse edge $e_6^{(1)}$ with $c\left(e_6^{(1)}\right) = 2b_1 + \frac{p_1}{2} = 512$.
- for each $i \in \{2, 3, 4, 5\}$, sessions 1 and $i$ traverse $e_i^{(1)}$ with $c\left(e_i^{(1)}\right) = 2b_1 + p_1 = 1024$.

The construction of the network $G_2^{(1)}$ is similar; it can be found in Figure 1. We proceed to the case $r = 2$, corresponding to the second epoch, where $I_1^{(2)} = I_1^{(1)} \setminus \{1, 6\} = \{2, 3, 4, 5\}$ and

$$\text{ElevSched}_1^{(2)} = \left(\text{ElevSched}_1^{(1)} \upharpoonright I_1^{(1)}\right) \mid I_1^{(2)} = 5, 4, 3, 2, 2, 3, 4, 5, \ldots, 5, 4, 3, 2, 2, 3, 4, 5, \ldots$$

Thus, $\text{ElevSched}_1^{(2)} \upharpoonright I_1^{(2)} = 5, 4, 3, 2$, so that $i_j^{(2)} = 5$ and $i_l^{(2)} = 2$. We continue to describe the construction of $G_1^{(2)}$:

- sessions 5 and 2 traverse edge $e_2^{(2)}$ with $c\left(e_2^{(2)}\right) = 2b_2 + \frac{p_2}{2} = 1152$.
- for each $i \in \{3, 4\}$, sessions 5 and $i$ traverse edge $e_i^{(2)}$ with $c\left(e_i^{(2)}\right) = 2b_2 + p_2 = 1280$.
The construction of the network $G_2^{(2)}$ is similar; it can be found in Figure 1. We proceed to the case $r = 3$, corresponding to the third epoch, where $\mathcal{I}_1^{(3)} = \mathcal{I}_1^{(2)} \setminus \{2, 5\} = \{3, 4\}$ and

$$\text{ElevSched}_1^{(3)} = \left( \text{ElevSched}_1^{(2)} \upharpoonright \mathcal{I}_1^{(2)} \right) \mid \mathcal{I}_1^{(3)}$$

$$= 3, 4, 4, 3, \ldots, 3, 4, 4, 3, \ldots.$$ 

Thus, $\text{ElevSched}_1^{(3)} \upharpoonright \mathcal{I}_1^{(3)} = 3, 4$, so that $i_j^{(3)} = 3$ and $i_l^{(3)} = 4$. We continue to describe the construction of $G_1^{(3)}$:

- sessions 3 and 4 traverse edge $e_4^{(3)}$ with $c(e_4^{(3)}) = 2b_3 + \frac{r_3}{2} = 2 \cdot 640 + \frac{64}{2} = 1312$.

The construction of the network $G_2^{(3)}$ is similar; it can be found in Figure 1, which also depicts the complete network $G(\text{ElevSched})$.

\section{Lower Bounds}

\subsection{Oblivious Algorithms}

We present a lower bound of $\Omega(dn)$ on the convergence complexity of any optimistic, oblivious and bottleneck algorithm $\text{Alg} = \langle \text{Sched}, \text{Term} \rangle$ that computes the max-min fair rate vector.

The proof uses the network $G(\text{Sched})$ constructed in Section 6. We start with two immediate technical claims that quantify $\Delta_Q(i,e)$ in case edge $e$ is traversed by only two sessions that remain active in state $Q$. (These claims will be used for Proposition 7.3.)

\begin{claim}
For an edge $e$ traversed only by sessions $i, i' \in \mathcal{A}_Q \mid e$, $\Delta_Q(i,e) = c(e) - r_Q(i) - \min \left\{ \frac{c(e)}{2}, r_Q(i') \right\}$.
\end{claim}

Since in the setting of Claim 7.1, $c(e) - r_Q(i) - r_Q(i') = \text{resid}_Q(e)$, it follows:

\begin{claim}
For an edge $e$ traversed only by sessions $i, i' \in \mathcal{A}_Q \mid e$, $\Delta_Q(i,e) \geq \text{resid}_Q(e)$.
\end{claim}

We restrict attention to the execution $\alpha$ of $\text{Alg}$ on the network $G(\text{Sched} \mid S_j)$ with any particular cluster $S_j$; for notational simplicity, we shall abuse notation and use $G(\text{Sched})$ to denote $G(\text{Sched} \mid S_j)$ and $S$ to denote $S_j$.

For an execution $\alpha$, for any indices $l_1$ and $l_2$, $0 \leq l_1 \leq l_2$, define the set $S \mid_\alpha (Q_{l_1}, Q_{l_2}) \subseteq S$ to be $S \mid_\alpha (Q_{l_1}, Q_{l_2}) = \{ i_l \mid l_1 < l \leq l_2 \text{ and } i_l \in \mathcal{A}_{Q_{l_1}} \}$; roughly speaking, $S \mid_\alpha (Q_{l_1}, Q_{l_2})$
Figure 1: The network $G(ElevSched)$
contains indices of all sessions active in \( Q_{l_1} \) that are scheduled in front any state following \( Q_{l_1} \) and up to and including \( Q_{l_2} \). Notice that if \( l_1 = l_2, S \mid_\alpha (Q_{l_1}, Q_{l_2}] = \emptyset \). For an execution \( \alpha \), for any index \( l_1 \geq 0 \), define the set \( S \mid_\alpha (Q_{l_1}, \infty) \leq S \) to be \( S \mid_\alpha (Q_{l_1}, \infty) = \{ i_l \mid l_1 < l \text{ and } i_l \in A_{Q_{l_1}} \} \); roughly speaking, \( S \mid_\alpha (Q_{l_1}, \infty) \) contains indices of all sessions active in \( Q_{l_1} \) that are scheduled in front of any state following \( Q_{l_1} \). Define \( S \mid_\alpha (Q_{l_1}, Q_{l_2}] \mid e \) and \( S \mid_\alpha (Q_{l_1}, \infty) \mid e \) in the natural way. Recall that for any state \( Q_l \), \( \hat{l} \) is the least integer \( l' \geq l \) such that \( S \mid_\alpha (Q_l, Q_{l'}) = A_{Q_l} \), or infinite if no such integer exists.

Define inductively the index sequence \( l_0, l_1, \ldots \) as follows. For the basis case, \( l_0 = 0 \). Assume inductively that for any integer \( r \geq 1 \), we have defined \( l_1, \ldots, l_{r-1} \). For the induction step, define \( l_r = l_{r-1} \). Define also the execution fragments \( \alpha^{(1)}, \alpha^{(2)}, \ldots \) where for any integer \( r \geq 1 \), \( \alpha^{(r)} = Q_{l_{r-1}}, \ldots, i_{r_r}, Q_{l_r} \). Call each \( \alpha^{(r)}, r \geq 1 \), an execution epoch in \( \alpha \). Note that in case \( l_r = \infty \), for any integer \( r \geq 1 \), \( \alpha^{(r)} \) is the infinite sufix of \( \alpha \) following state \( Q_{l_{r-1}} \).

Thus, write \( \alpha = \alpha^{(1)} \cdot \alpha^{(2)} \ldots \). We remark that in case \( \alpha \) is infinite, the number of execution epochs in \( \alpha \) can still be finite if (and only if) there exists some integer \( r \geq 0 \) such that \( l_r = \infty \).

To simplify notation, denote each state \( Q_{l_r}, r \geq 0 \), as \( Q^{(r)} \). Thus, \( Q^{(r)} \) is the latest state in execution epoch \( \alpha^{(r)} \) of \( \alpha \). (Note that \( Q^{(r)} \) exists if and only if \( l_r < \infty \).

The backbone of our analysis is a technical proposition (Proposition 7.3) that describes the states of execution \( \alpha \). The first part of Proposition 7.3 (Part (A)) deals with each state starting from \( Q^{(r-1)} \), \( 1 \leq r \leq \frac{d}{2} \), such that not all active sessions in \( Q^{(r-1)} \) have yet been scheduled until this state. Thus, \( Q^{(r)} \) would be the state immediately following this sequence of states; we will later show that \( Q^{(r)} \) is well defined. Observe that the states considered in Part (A) for any particular integer \( r, 1 \leq r \leq \frac{d}{2} \), are precisely the states in execution epoch \( \alpha^{(r)} \) excluding state \( Q^{(r)} \). Part (B) explores properties of state \( Q^{(r)} \).

**Proposition 7.3 (Properties of execution \( \alpha \))** For each integer \( r, 1 \leq r \leq \frac{d}{2} \), the following hold for states in execution epoch \( \alpha^{(r)} \):

1. (Properties of states from \( Q^{(r-1)} \) to \( Q^{(r)} \)) Consider any state \( Q \) such that \( Q^{(r-1)} \) \( \xrightarrow{\alpha} Q \) and \( A_{Q^{(r-1)}} \not\subseteq S \mid_\alpha (Q^{(r-1)}, Q) \). Then, the following conditions hold:

   \[ r_Q(i) = \begin{cases} b_r + \frac{p_r}{2}, & \text{if } i \in S \mid_\alpha (Q^{(r-1)}, Q) \\ b_{r-1} + \frac{p_{r-1}}{2}, & \text{otherwise}; \end{cases} \]
(2) for each session \( i \in \mathcal{A}_{Q(r-1)} \),

\[
FSQ \left( e_i^{(r)} \right) = \begin{cases} 
   b_r + \frac{s_r}{T}, & \text{if } i \not\in \{ i_f^{(r)}, i_l^{(r)} \} \\
   b_r + \frac{s_r}{T}, & \text{if } i = i_l^{(r)} 
\end{cases}
\]

(3) for the edges of \( G(\text{Sched}) \), it holds, for \( Q \neq Q^{(r-1)} \), that:

(a) edge \( e_i^{(r)} \) is a bottleneck edge for state \( Q \);

(b) for each session \( i \in \mathcal{A}_{Q(r-1)} \setminus \{ i_f^{(r)}, i_l^{(r)} \}, FSQ \left( e_i^{(r)} \right) \neq FSQ \left( i_f^{(r)} \right) \), so that edge \( e_i^{(r)} \) is not a bottleneck edge for state \( Q \);

(c) for any integer \( r' \), \( 1 \leq r' < r \), and for any edge \( e^{(r')} \in E^{(r')}(\text{Sched}) \) traversed by session \( i \in \mathcal{A}_{Q(r-1)} \), FSQ \( e^{(r')} \) \( \neq \) FSQ \( i_f^{(r')} \), so that edge \( e^{(r')} \) is not a bottleneck edge for state \( Q \);

(d) for any integer \( r' \), \( r < r' \leq \frac{4}{27} \), and for any edge \( e^{(r')} \in E^{(r')}(\text{Sched}) \) traversed by session \( i \in \mathcal{A}_{Q(r-1)} \setminus \{ i_f^{(r)}, i_l^{(r)} \}, FSQ \left( e^{(r')} \right) \neq FSQ(i), \) so that edge \( e^{(r')} \) is not a bottleneck edge for state \( Q \);

(4) for each session \( i \in \mathcal{A}_{Q(r-1)}, i \not\in \text{Term}(Q) \):

(5) for each session \( i \in \mathcal{A}_{Q(r-1)} \setminus \mathcal{S}_\alpha \left( Q^{(r-1)}, Q \right) \) scheduled in front of state \( \overline{Q} \),

\[
\Delta Q(i) = \begin{cases} 
   \frac{s_r}{T}, & \text{if } i \neq i_l^{(r)} \\
   \frac{s_r}{T}, & \text{otherwise}.
\end{cases}
\]

(B) (Properties of state \( Q^{(r)} \)) The following conditions hold for state \( Q^{(r)} \):

(1) \( l_r < \infty \);

(2) for each session \( i \in \mathcal{A}_{Q(r-1)} \),

\[
r_{Q(r)}(i) = \begin{cases} 
   b_r + \frac{s_r}{T}, & \text{if } i \in \{ i_f^{(r)}, i_l^{(r)} \} \\
   b_r + \frac{s_r}{T}, & \text{if } i \in \mathcal{A}_{Q(r-1)} \setminus \{ i_f^{(r)}, i_l^{(r)} \} 
\end{cases}
\]

(3) for each session \( i \in \mathcal{A}_{Q(r-1)} \), \( i \in \{ i_f^{(r)}, i_l^{(r)} \} \) if and only if \( i \in \text{Term}(Q^{(r)}) \);

(4) for each session \( i \in \mathcal{A}_{Q(r-1)} \setminus \{ i_f^{(r)}, i_l^{(r)} \} \) and for any integer \( r' \), \( 1 \leq r' \leq r \), for any edge \( e_i^{(r')} \),
Proposition 7.3 deals mainly with sessions active in state $Q^{(r-1)}$ for any index $r$, $1 \leq r \leq \frac{d}{2}$.

We start with an informal description of the conditions in Part (A). Condition (A/1) determines rates of active sessions in state $Q$. Condition (A/2) determines the fair shares of all edges in epoch $r$; condition (A/3) establishes that edge $e^{(r)}_{i \rightarrow j}$ is the only bottleneck edge for state $Q$. Condition (A/4) guarantees that no session is terminated in state $Q$. Finally, condition (A/5) determines the increase in state $Q$ for sessions active in $Q^{(r-1)}$ that are not yet scheduled. We now continue with the conditions in Part (B). Condition (B/1) asserts that all active sessions in state $Q^{(r-1)}$ must be scheduled in execution epoch $\alpha^{(r)}$; condition (B/2) specifies their rates upon completion of $\alpha^{(r)}$. Moreover, condition (B/3) determines which of these sessions are terminated upon completion of $\alpha^{(r)}$. Condition (B/4) provides lower bounds on the residual capacity and fair share of some edges from previous epochs upon completion of $\alpha^{(r)}$.

We note that conditions (B/1) and (B/3) will suffice by themselves to imply the lower bound. However, the rather technical rest of conditions are necessary to assume inductively in the proof of conditions (B/1) and (B/3). To simplify notation, we will denote $e^{(r)}_{i \rightarrow j}$ as $e^{(r)}_{i \rightarrow j}$.

**Proof:** The proof is by induction on $r$. For the sake of shortening the proof, we merge the proof for the basis case (where $r = 1$) and the proof for the induction step; thus, the case $r = 1$ will be treated separately (where needed) along the proof of the induction step.

We assume, as our induction hypothesis, that the claims hold for all integers less than some fixed integer $r$, $1 \leq r \leq \frac{d}{2}$. Notice that if $r = 1$, the induction hypothesis is empty. We proceed to the induction step, where we prove the claims for $r$.

**Proof of Part (A):**

The proof is by induction on $Q$. For the sake of shortening the proof, we merge again the proof for the basis case (where $Q = Q^{(r-1)}$) and the proof for the induction step; thus, the case $Q = Q^{(r-1)}$ will be treated separately (where needed) along the proof for the induction step (on states).

Fix any state $Q$ such that $Q^{(r-1)} \xrightarrow{\alpha} Q$ and $A \xrightarrow{\alpha} Q^{(r-1)} \notin S \mid \alpha \left( Q^{(r-1)}, Q \right)$, and assume that for each state $Q'$ such that $Q^{(r-1)} \xrightarrow{\alpha} Q' \xrightarrow{\alpha} Q$, the claims of Part (A) hold for $Q'$; thus, we assume, as our induction hypothesis, that the claims of Part (A) hold for all states from $Q^{(r-1)}$. The proof proceeds by induction on $r$. For the sake of shortening the proof, we merge the proof for the basis case (where $r = 1$) and the proof for the induction step; thus, the case $r = 1$ will be treated separately (where needed) along the proof of the induction step (on states).
$Q^{(r-1)}$ through but not including state $Q$. Notice that if $Q = Q^{(r-1)}$, the induction hypothesis is empty. We now proceed with the induction step, where we prove the claims for $Q$.

**Proof of (A/1):**

There are two cases to consider.

1. Assume first that $Q = Q^{(r-1)}$. Then, $S \mid_\alpha (Q^{(r-1)}, Q) = \emptyset$. In case $r = 1$, $Q = Q_0$, and condition (A/1) holds trivially since all session rates are initially zero and $b_0 = p_0 = 0$. So assume $r > 1$. By the induction hypothesis of induction on $r$ (condition (B/3)),

$$A \xrightarrow{Q^{(r-2)}} A \xrightarrow{Q^{(r-1)}} \{i_{f_j}^{(r-1)}, i_{l_j}^{(r-1)}\}.$$  

Hence, condition (A/1) follows from the induction hypothesis of induction on $r$ (condition (B/2)).

2. Assume now that $Q \neq Q^{(r-1)}$. We proceed by case analysis on $i_Q$ (the session scheduled in front of state $Q$).

(a) Assume first that $i_Q \notin A \xrightarrow{Q^{(r-1)}}$. (Notice that this case need not be considered when $r = 1$, since all sessions are active in $Q^{(0)} = Q_1$.) Then, $S \mid_\alpha (Q^{(r-1)}, Q) = S \mid_\alpha (Q^{(r-1)}, Q) = \emptyset$. Since, in addition, no session rates change from $Q$ to $Q$, the claim follows inductively. So we proceed to the cases where $i_Q \in A \xrightarrow{Q^{(r-1)}}$.

(b) Assume next that $i_Q = i_{f_j}^{(r)}$. There are two subcases to consider.

i. Take first $i_{f_j}^{(r)} \notin S \mid_\alpha (Q^{(r-1)}, Q)$; that is, $i_{f_j}^{(r)}$ has not been scheduled in front of any state between $Q^{(r-1)}$ and $Q$. Recall that $i_{f_j}^{(r)}$ is the session scheduled first (following state $Q^{(r-1)}$) among active sessions in $Q^{(r-1)}$; so, for each session $i \in A \xrightarrow{Q^{(r-1)}}$, $i \notin S \mid_\alpha (Q^{(r-1)}, Q)$. Thus, by the induction hypothesis of induction on states (condition (A/1)), $r_{Q}^{-} (i) = b_{r-1} + \frac{r_{Q}^{-} (i)}{2} = b_r$ (by recursive definition of $b_r$). In particular, $r_{Q}^{-} (i_{f_j}^{(r)}) = b_r$. By the induction hypothesis of induction on states (condition (A/5)), $\Delta_{Q}^{-} (i_{f_j}^{(r)}) = b_r$. Thus, by update operation, $r_{Q} (i_{f_j}^{(r)}) = r_{Q}^{-} (i_{f_j}^{(r)}) + \Delta_{Q}^{-} (i_{f_j}^{(r)}) = b_r + \frac{b_r}{2}$. Since $p_r \neq 0$, it follows that for each session $i \in A \xrightarrow{Q^{(r-1)}} \{i_{f_j}^{(r)}\}$, $r_{Q}^{-} (i) < r_{Q} (i_{f_j}^{(r)})$. Thus, by update operation, $r_{Q} (i) = r_{Q}^{-} (i) = b_{r-1} + \frac{r_{Q}^{-} (i)}{2}$. Since
$S|_a \left( Q^{(r-1)}, Q \right) = \left\{ i_f^{(r)} \right\}$, it follows that for each session $i \in A_{\overline{Q^{(r-1)}}}$,

\[
r_Q(i) = \begin{cases} 
  b_r + \frac{p_r}{2}, & \text{if } i \in S|_a \left( Q^{(r-1)}, Q \right) \\
  b_{r-1} + \frac{p_{r-1}}{2}, & \text{otherwise}.
\end{cases}
\]

ii. Take now $i_f^{(r)} \in S|_a \left( Q^{(r-1)}, Q \right)$. Then, $S|_a \left( Q^{(r-1)}, Q \right) = S|_a \left( Q^{(r-1)}, \overline{Q} \right)$. Clearly, $i_f^{(r)} \not\in S|_a \left( Q^{(r-1)}, \overline{Q} \right)$, while $i_f^{(r)} \in S|_a \left( Q^{(r-1)}, \overline{Q} \right)$. By construction of $G(Sched)$, session $i_f^{(r)}$ traverses edge $e_i^{(r)}$ with $c(e_i^{(r)}) = 2b_r + \frac{p_r}{2}$, as does session $i_f^{(r)}$. By the induction hypothesis of induction on states (condition (A/1)), $r_Q \left( i_f^{(r)} \right) = b_r + \frac{p_r}{2}$, while $r_Q \left( i_f^{(r)} \right) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$. By Claim 7.1,

\[
\Delta_Q \left( i_f^{(r)}, e_i^{(r)} \right) = c(e_i^{(r)}) - r_Q \left( i_f^{(r)} \right) - \min \left\{ \frac{c(e_i^{(r)})}{2}, r_Q \left( i_f^{(r)} \right) \right\} = 2b_r + \frac{p_r}{2} - \left( b_r + \frac{p_r}{2} \right) - \min \left\{ b_r + \frac{p_r}{2}, b_r \right\} = 0,
\]

so that $\Delta_Q \left( i_f^{(r)} \right) = 0$. Thus, by update operation, for each session $i \in A_{\overline{Q^{(r-1)}}}$, $r_Q(i) = r_Q(i)$. By the induction hypothesis of induction on states (condition (A/4)), it follows that $A_{\overline{Q^{(r-1)}}} = A_{\overline{Q^{(r-1)}}}$. Hence, the induction hypothesis of induction on states (condition (A/1)) implies that for each session $i \in A_{\overline{Q^{(r-1)}}}$,

\[
r_Q(i) = \begin{cases} 
  b_r + \frac{p_r}{2}, & \text{if } i \in S|_a \left( Q^{(r-1)}, Q \right) \\
  b_{r-1} + \frac{p_{r-1}}{2}, & \text{otherwise}.
\end{cases}
\]

(c) Assume now that $i_Q \neq i_f^{(r)}$. There are two subcases to consider.

i. Consider first the case where $i_Q \not\in S|_a \left( Q^{(r-1)}, \overline{Q} \right)$. By the induction hypothesis of induction on states (condition (A/1)), it follows that $r_Q \left( i_Q \right) = b_{r-1} + \frac{p_{r-1}}{2} = b_r$; moreover, by condition (A/5), $\Delta_Q \left( i_Q \right) = \frac{p_r}{2}$. Thus, by update operation, $r_Q \left( i_Q \right) = r_Q \left( i_Q \right) + \Delta_Q \left( i_Q \right) = b_r + \frac{p_r}{2}$. By the induction hypothesis of induction on states (condition (A/1)), it follows that for each session $i \in A_{\overline{Q^{(r-1)}}} \setminus \{ i_Q \}$, $r_Q \left( i \right) \leq b_r + \frac{p_r}{2}$; hence, $r_Q \left( i \right) \leq r_Q \left( i_Q \right)$.

30
Thus, by \texttt{update} operation, \( r_Q(i) = r_{-Q}(i) \). Since \( S \mid_\alpha (Q^{(r-1)}, Q) = S \mid_\alpha (Q^{(r-1)}, Q) \cup \{i_Q\} \), the induction hypothesis of induction on states (condition (A/1)) implies now that for each session \( i \in A \xrightarrow{Q^{(r-1)}} \),

\[
r_Q(i) = \begin{cases} b_r + \frac{p_r}{2}, & \text{if } i \in S \mid_\alpha (Q^{(r-1)}, Q) \\ b_{r-1} + \frac{p_{r-1}}{2}, & \text{otherwise}. \end{cases}
\]

ii. Consider finally the case where \( i_Q \in S \mid_\alpha (Q^{(r-1)}, Q) \setminus \{i_Q^{(r)}\} \). Then, \( S \mid_\alpha (Q^{(r-1)}, Q) = S \mid_\alpha (Q^{(r-1)}, Q) \). Clearly, both \( i_Q, i_Q^{(r)} \in S \mid_\alpha (Q^{(r-1)}, Q) \). Thus, by the induction hypothesis of induction on states (condition (A/1)),

\[
r_{Q^{-}}(i_Q) = r_{Q^{-}}(i_Q^{(r)}) = b_r + \frac{p_r}{2}. \]

By construction of \( G(\text{Sched}) \), \( c(i_Q^{(r)}) = 2b_r + p_r \). It follows that \( r_{Q^{-}}(i_Q) = r_{Q^{-}}(i_Q^{(r)}) = \frac{c(i_Q^{(r)})}{2} \). Hence, Claim 7.1 implies that \( \Delta_{Q^{-}}(i_Q, i_Q^{(r)}) = 0 \), so that \( \Delta_{Q^{-}}(i_Q) = 0 \). Thus, by \texttt{update} operation, for each session \( i \in A_{Q^{-}} \), \( r_Q(i) = r_{Q^{-}}(i) \). By the induction hypothesis of induction on states (condition (A/4)), it follows that \( A_{Q^{-}} = A \xrightarrow{Q^{(r-1)}} \). Hence, the induction hypothesis of induction on states (condition (A/1)) implies that for each session \( i \in A \xrightarrow{Q^{(r-1)}} \),

\[
r_Q(i) = \begin{cases} b_r + \frac{p_r}{2}, & \text{if } i \in S \mid_\alpha (Q^{(r-1)}, Q) \\ b_{r-1} + \frac{p_{r-1}}{2}, & \text{otherwise}. \end{cases}
\]

\[\Box\]

\textbf{Proof of (A/2):}

Consider first edge \( e_i^{(r)} \), for any session \( i \in A \xrightarrow{Q^{(r-1)}} \setminus \{i_j^{(r)}, i_k^{(r)}\} \), with capacity \( c(e_i^{(r)}) = 2b_r + p_r \), which is traversed by sessions \( i_j^{(r)} \) and \( i \). Either by definition of \( Q_0 \) in case \( Q = Q^{(r-1)} \) and \( r = 1 \), or by the induction hypothesis of induction on \( r \) (condition (B/3)) in case \( Q = Q^{(r-1)} \) and \( r \neq 1 \), or by the induction hypothesis of induction on states (condition (A/4)) in case \( Q \neq Q^{(r-1)} \), it follows that both \( i_j^{(r)}, i \in A_Q \), so that \( A_Q \mid e_i^{(r)} = \{i_j^{(r)}, i\} \). Hence,

\[
FS_Q(e_i^{(r)}) = \frac{c(e_i^{(r)}) - \text{allot}_{Q}(e_i^{(r)})}{|A_Q \mid e_i^{(r)}}
\]
Consider now edge $e_i^{(r)}$ with capacity $c\left( e_i^{(r)} \right) = 2b_r + \frac{p_r}{2}$, which is traversed by sessions $i_f^{(r)}$ and $i_l^{(r)}$. Either by definition of $Q_0$ in case $Q = Q^{(r-1)}$ and $r = 1$, or by the induction hypothesis of induction on $r$ (condition (B/3)) in case $Q = Q^{(r-1)}$ and $r \neq 1$, or by the induction hypothesis of induction on states (condition (A/4)) in case $Q \neq Q^{(r-1)}$, it follows that both $i_f^{(r)}, i_l^{(r)} \in A_Q$, so that $A_Q \mid e_i^{(r)} = \{ i_f^{(r)}, i_l^{(r)} \}$. Hence,

$$FS_Q \left( e_i^{(r)} \right) = \frac{c\left( e_i^{(r)} \right) - allot_Q \left( e_i^{(r)} \right)}{|A_Q \mid e_i^{(r)}|}
= \frac{c\left( e_i^{(r)} \right)}{2}
= b_r + \frac{p_r}{2}.$$  

Proof of (A/3/a):

By construction of $G(\text{Sched})$, edge $e_i^{(r)}$ is traversed by sessions $i_f^{(r)}$ and $i_l^{(r)}$. We prove that each of them receives its minimum fair share on edge $e_i^{(r)}$.

1. Take first session $i_f^{(r)}$, which traverses:
   - edge $e_i^{(r)}$; by condition (A/2) shown above, $FS_Q \left( e_i^{(r)} \right) = b_r + \frac{p_r}{2}$;
   - edge $e_i^{(r)}$, for each session $i \in A \rightarrow Q^{(r-1)} \setminus \{ i_f^{(r)}, i_l^{(r)} \}$; by condition (A/2) shown above, $FS_Q \left( e_i^{(r)} \right) = b_r + \frac{p_r}{2}$;
   - edge $e^{(r')} \in E^{(r')} (\text{Sched})$, for each integer $r'$, $1 \leq r' < r$ in case $r > 1$.

By the induction hypothesis of induction on $r$ (condition (B/4/b)), $FS_{Q^{(r-1)}} \left( e^{(r')} \right) \geq b_{r-1} + \frac{3p_{r-1}}{4} = b_r + p_r$ (by recursive definition of $b_r$ and $p_r$). Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) is non-empty and implies that $A_Q = A \rightarrow Q^{(r-1)}$. It follows that $FS_Q \left( e^{(r')} \right) = FS_{Q^{(r-1)}} \left( e^{(r')} \right)$, so that $FS_Q \left( e^{(r')} \right) \geq b_r + p_r$.  

32
Hence, $MFS_Q \left( i_f^{(r)} \right) = b_r + \frac{p_r}{4}$, so that $MFS_Q \left( i_f^{(r)} \right) = FS_Q \left( e_i^{(r)} \right)$.

2. Take now session $i_l^{(r)}$, which traverses:

- edge $e_i^{(r)}$; by condition (A/2) shown above, $FS_Q \left( e_i^{(r)} \right) = b_r + \frac{p_r}{4}$;
- edge $e^{(r')} \in E^{(r')} (Sched)$, for each integer $r'$, $1 \leq r' < r$, in case $r > 1$.

By the induction hypothesis of induction on $r$ (condition (B/4/b)), $FS_{Q^{(r-1)}} \left( e^{(r')} \right) \geq b_{r-1} + \frac{3p_{r-1}}{4} = b_r + p_r$. Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) implies that $A_Q = A_{Q^{(r-1)}}$. It follows that $FS_Q \left( e^{(r')} \right) \geq FS_{Q^{(r-1)}} \left( e^{(r')} \right)$, so that $FS_Q \left( e^{(r')} \right) \geq b_r + p_r$.

These imply that $MFS_Q \left( i_l^{(r)} \right) = b_r + \frac{p_r}{4}$, so that $MFS_Q \left( i_l^{(r)} \right) = FS_Q \left( e_i^{(r)} \right)$. Hence $MFS_Q \left( i_f^{(r)} \right) = MFS_Q \left( i_l^{(r)} \right) = FS_Q \left( e_i^{(r)} \right)$, so that $e_i^{(r)}$ is a bottleneck edge for $Q$.  

**Proof of (A/3/b):**

Consider edge $e_i^{(r)}$, for any session $i \in A_{Q^{(r-1)}} \setminus \{ i_f^{(r)}, i_l^{(r)} \}$, which is traversed by sessions $i_f^{(r)}$ and $i$. By condition (A/3/a) shown above, edge $e_i^{(r)}$, traversed by session $i_f^{(r)}$, is a bottleneck edge for state $Q$, so that $MFS_Q \left( i_f^{(r)} \right) = FS_Q \left( e_i^{(r)} \right)$. By condition (A/2) shown above, $FS_Q \left( e_i^{(r)} \right) = b_r + \frac{p_r}{4}$ and $FS_Q \left( e_i^{(r)} \right) = b_r + \frac{p_r}{4}$. Since $p_r \neq 0$, it follows that $FS_Q \left( e_i^{(r)} \right) < FS_Q \left( e_i^{(r)} \right)$, so that $MFS_Q \left( i_f^{(r)} \right) < FS_Q \left( e_i^{(r)} \right)$. It follows that $e_i^{(r)}$ is not a bottleneck edge for $Q$.

**Proof of (A/3/c):**

The claim holds vacuously in case $r = 1$. So assume $r > 1$. Consider any edge $e^{(r')} \in E^{(r')} (Sched)$, for any integer $r'$, $1 \leq r' < r$, traversed by some session $i \in A_{Q^{(r-1)}}$. By the induction hypothesis of induction on $r$ (condition (B/4/b)), $FS_{Q^{(r-1)}} \left( e^{(r')} \right) \geq b_{r-1} + \frac{3p_{r-1}}{4}$. Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) is non-empty and implies that $A_Q = A_{Q^{(r-1)}}$. It follows that $FS_Q \left( e^{(r')} \right) = FS_{Q^{(r-1)}} \left( e^{(r')} \right)$, so that $FS_Q \left( e^{(r')} \right) = p_{r-1} + \frac{3p_{r-1}}{4} = b_r + p_r$. Session $i$ traverses also edge $e_i^{(r)}$. By condition (A/2) shown above, $FS_Q \left( e_i^{(r)} \right) \leq b_r + \frac{p_r}{4}$.

Since $p_r \neq 0$, it follows that $FS_Q \left( e_i^{(r)} \right) > FS_Q \left( e_i^{(r)} \right)$, so that $FS_Q \left( e^{(r')} \right) > MFS_Q(i)$. It follows that $e^{(r')} \neq 0$.  

**Proof of (A/3/d):**
Consider any edge $e^{(r')} \in E^{(r')}(Sched)$, for any integer $r', r < r' \leq \frac{d}{2}$. Take any session $i$ traversing $e^{(r')}$. By construction of $G(Sched)$, $i \in A \rightarrow_{Q^{(r-1)}} \{i^{(r)}, i^{(r')}_l\}$, and $i$ traverses also $e^{(r)}_i$ with $c\left(e^{(r)}_i\right) = 2b_r + p_r$. By condition (A/2) shown above, $FS_Q\left(e^{(r)}_i\right) = b_r + \frac{p_r}{2} = \frac{c\left(e^{(r)}_i\right)}{2}$. Since $Q \neq Q^{(r-1)}$, the induction hypothesis of induction on states (condition (A/4)) is non-empty and implies that both sessions traversing $e^{(r')}$ are active in $Q$. So, $FS_Q\left(e^{(r')}\right) = \frac{c\left(e^{(r')}\right)}{2}$.

By Lemma 6.1, $c\left(e^{(r)}_i\right) < c\left(e^{(r')}\right)$. So, $FS_Q\left(e^{(r)}_i\right) < FS_Q\left(e^{(r')}\right)$. By definition of minimum fair share, $MFS_Q(i) < FS_Q\left(e^{(r')}\right)$. Hence, $e^{(r')}$ is not a bottleneck edge for $Q$.

Proof of (A/4):

If $Q = Q^{(r-1)}$, the claim holds trivially. So assume $Q \neq Q^{(r-1)}$. By condition (A/3/b) shown above, any edge $e^{(r)}_i$ with $i \in A \rightarrow_{Q^{(r-1)}} \{i^{(r)}_l, i^{(r)}_l\}$ is not a bottleneck edge for $Q$. Moreover, by conditions (A/3/c) and (A/3/d) shown above, neither is any edge $e^{(r')} \in E^{(r')}(Sched)$ with $1 \leq r' < r$ or $r < r' \leq \frac{d}{2}$. Since Alg is bottleneck, none of these edges causes the termination of a session in $Q$. Thus, it only remains to prove that edge $e^{(r)}_i$ does not cause either the termination of any of the sessions $i^{(r)}_l$ and $i^{(r)}_r$ traversing it.

Since $p_r \neq 0$, conditions (A/1) and (A/2) shown above imply that $r_Q\left(i^{(r)}_l\right) \neq FS_Q\left(e^{(r)}_i\right)$ and $r_Q\left(i^{(r)}_r\right) \neq FS_Q\left(e^{(r)}_i\right)$. Since Alg is bottleneck, it follows that $i^{(r)}_l, i^{(r)}_r \notin \text{Term}(Q)$.

Proof of (A/5):

By case analysis on $i$.

1. Assume first that $i = i^{(r)}_l$. We first prove by case analysis on $Q$ that, in this case, for each session $k \in A \rightarrow_{Q^{(r-1)}}$, $r_Q(k) = b_r$. (This will be useful for some later calculations.)

   - Take first $Q = Q^{(r-1)}$. In case $r = 1$ where $Q = Q_0$, all session rates are zero in $Q_0$. Since $b_0 = p_0 = 0$, it follows that $r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2}$. Assume now $r > 1$. Since $k \in A \rightarrow_{Q^{(r-1)}}$, by construction of $G(Sched)$ and by the induction hypothesis of induction on $r$ (condition (B/3)), $k \notin \{i^{(r-1)}_l, i^{(r-1)}_r\}$. Thus, the induction hypothesis of induction on $r$ (condition (B/2)) implies that $r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2}$.

   - Take now $Q \neq Q^{(r-1)}$. Since $i^{(r)}_l \in A \rightarrow_{Q^{(r-1)}} \setminus S |_\alpha \left(Q^{(r-1)}, Q \right)$, $k \in A \rightarrow_{Q^{(r-1)}} \setminus S |_\alpha \left(Q^{(r-1)}, Q \right)$ as well. So, by condition (A/1) shown above, $r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2}$.
So, in all cases, \( r_Q(k) = b_{r-1} + \frac{p_{r-1}}{2} = b_r \). By construction of \( G(\text{Sched}) \), \( i_{f'}^{(r)} \) traverses:

- edge \( e_i^{(r)} \) with \( c(e_i^{(r)}) = 2b_r + \frac{p_r}{2} \);
- edge \( e_k^{(r)} \), for each session \( k \in A \xrightarrow{Q(r-1)} \{i_{f'}^{(r)}, i_i^{(r)}\} \), with \( c(e_k^{(r)}) = 2b_r + p_r \);
- edge \( e^{(r')} \in E^{(r')} (\text{Sched}) \) for each index \( r' \), \( 1 \leq r' < r \), in case \( r > 1 \).

We next calculate separately the increases for \( i_{f'}^{(r)} \) in \( Q \) allowed by these edges.

(a) Consider first edge \( e_i^{(r)} \), which is also traversed by \( i_i^{(r)} \in A \xrightarrow{Q^{(r-1)}} \). By Claim 7.1,

\[
\Delta_Q \left(i_{f'}^{(r)}, e_i^{(r)}\right) = c(e_i^{(r)}) - r_Q(i_{f'}^{(r)}) - \min \left\{ \frac{c(e_i^{(r)})}{2}, r_Q(i_i^{(r)}) \right\} = 2b_r + \frac{p_r}{2} - b_r - \min \left\{ b_r + \frac{p_r}{4}, b_r \right\} = \frac{p_r}{2}.
\]

(b) Consider next any edge \( e_k^{(r)} \), for any session \( k \in A \xrightarrow{Q^{(r-1)}} \{i_{f'}^{(r)}, i_i^{(r)}\} \), which is also traversed by session \( k \). By Claim 7.1,

\[
\Delta_Q \left(i_{f'}^{(r)}, e_k^{(r)}\right) = c(e_k^{(r)}) - r_Q(i_{f'}^{(r)}) - \min \left\{ c(e_k^{(r)}), r_Q(k) \right\} = 2b_r + p_r - b_r - \min \left\{ b_r + \frac{p_r}{4}, b_r \right\} = p_r.
\]

(c) Consider finally edge \( e^{(r')} \) for any integer \( r' \), \( 1 \leq r' < r \), in case \( r > 1 \). By the induction hypothesis of induction on \( r \) (condition \((B/4/a))\), \( \text{resid}_{Q^{(r-1)}}(e^{(r')}) \geq \frac{p_{r-1}}{2} \). Recall that \( i_{f'}^{(r)} \) is the session scheduled first (following state \( Q^{(r-1)} \)) among active sessions in \( Q^{(r-1)} \). Since \( i_{f'}^{(r)} \in A \xrightarrow{Q^{(r-1)}} S \mid_a (Q^{(r-1)}, Q) \), it follows that for each session \( i \in A \xrightarrow{Q^{(r-1)}} S \mid_a (Q^{(r-1)}, Q) \), it follows that for each session \( i \in A \xrightarrow{Q^{(r-1)}} S \mid_a (Q^{(r-1)}, Q) \). Thus, rates of all sessions are preserved from state \( Q^{(r-1)} \) to \( Q \). Hence, \( \text{resid}_Q(e^{(r')}) = \text{resid}_{Q^{(r-1)}}(e^{(r')}) \geq \frac{p_{r-1}}{2} = p_r \). By Claim 7.2, it follows that \( \Delta_Q \left(i_{f'}^{(r)}, e^{(r')}\right) \geq \text{resid}_Q(e^{(r')}) \geq p_r \).

Thus, by definition of increase, it follows that \( \Delta_Q \left(i_{f'}^{(r)}\right) = \frac{p_r}{2} \).

2. Assume now that \( i \notin \{i_{f'}^{(r)}, i_i^{(r)}\} \). Since \( i \in A \xrightarrow{Q^{(r-1)}} S \mid_a (Q^{(r-1)}, Q) \), by condition \((A/1)\) shown above, we have that \( r_Q(i) = b_{r-1} + \frac{p_{r-1}}{2} \). By construction of \( G(\text{Sched}) \), \( i \) traverses:
• edge \( e_i^{(r)} \) with \( c\left(e_i^{(r)}\right) = 2b_r + p_r; \\
• edge \( e^{(r')}(Sched) \), for each integer \( r' \), \( 1 \leq r' < r \), in case \( r > 1; \\
• edge \( e^{(r'')} \in E^{(r'')} (Sched) \), for any integer \( r'' \), \( r < r'' \leq \frac{d}{2}. \\

We next calculate separately the increases for \( i \in Q \) allowed by these edges.

(a) Consider first edge \( e_i^{(r)} \), which is also traversed by \( i_j^{(r)} \). Since \( i \) is scheduled in front of state \( Q \), it follows that \( i_j^{(r)} \in S |_\alpha (Q^{(r-1)}, Q) \). Thus, condition (A/1) shown above implies that \( \rho_Q(i) = b_r + \frac{p}{2} = \frac{c_i^{(r)}}{2} \). By Claim 7.1,

\[
\Delta_Q(i, e_i^{(r)}) = c(e_i^{(r)}) - \rho_Q(i) - \min \left\{ \frac{c(e_i^{(r)})}{2}, \rho_Q(i_j^{(r)}) \right\} \\
= c(e_i^{(r)}) - \rho_Q(i) - \frac{c(e_i^{(r)})}{2} \\
= \frac{c(e_i^{(r)})}{2} - \left(b_r \frac{p}{2} \right). 
\]

Alternatively, we derive that \( \Delta_Q(i, e_i^{(r)}) = b_r + \frac{p}{2} = \frac{c(e_i^{(r)})}{2}. \)

(b) Consider next edge \( e_i^{(r')} \), for any integer \( r' \), \( 1 \leq r' < r \), which is also traversed by \( i_j^{(r')} \), \( i_j^{(r')} \notin \{i_f^{(r')}, i_i^{(r')}\} \). Induction hypothesis of induction on \( r \) (condition (B/3)) implies that \( i_j^{(r')} \notin A_{Q^{(r-1)}} \); hence, \( i_j^{(r')} \notin A_Q \). So, \( \rho_Q(i) = \rho_Q(i_j^{(r')}) \). Recall that \( \rho_Q(i) = b_r + \frac{p}{2} \). On the other hand, by the induction hypothesis of induction on \( r \) (condition (B/2)), \( \rho_Q(i) = b_r + \frac{p_r}{4} \). Hence, \( \rho_Q(i) = \rho_Q(i) \).

(c) Consider finally edge \( e^{(r'')} \), for any integer \( r'' \), \( r < r'' \leq \frac{d}{2} \), which is also traversed by some other session \( k \in A_{Q^{(r-1)}} \setminus \{i_f^{(r')}, i_i^{(r')}\} \). By condition (A/1) shown above and by recursive definition of \( b_r \), \( \rho_Q(k) \leq \frac{c_i^{(r)}}{2} \). By Lemma 6.1,
\[ c\left(e^{(r''r)}\right) > c\left(e^{(r'i)}\right). \] By Claim 7.1,

\[
\Delta Q \left(i, e^{(r''r)}\right) = c\left(e^{(r''r)}\right) - r_Q(i) - \min\left\{\frac{c\left(e^{(r''r)}\right)}{2}, r_Q(k)\right\}
\]
\[
\geq c\left(e^{(r''r)}\right) - r_Q(i) - \min\left\{\frac{c\left(e^{(r''r)}\right)}{2}, \frac{c\left(e^{(r'i)}\right)}{2}\right\}
\]
\[
\geq c\left(e^{(r''r)}\right) - r_Q(i) - \frac{c\left(e^{(r''r)}\right)}{2}
\]
\[
= \frac{c\left(e^{(r''r)}\right)}{2} - \left(b_{r-1} + \frac{p_{r-1}}{2}\right)
\]

Since \( c\left(e^{(r''r)}\right) > c\left(e^{(r'i)}\right) \), a comparison of the first expression for \( \Delta Q \left(i, e^{(r''r)}\right) \) to \( \Delta Q \left(i, e^{(r'i)}\right) \) implies that \( \Delta Q \left(i, e^{(r''r)}\right) > \Delta Q \left(i, e^{(r'i)}\right) \).

Thus, by definition of increase, it follows that \( \Delta Q(i) = \frac{p_r}{2} \).

3. Assume finally that \( i = i^{(r'i)}_l \). Since \( i^{(r'i)}_l \in A_{Q^{(r'-1)}} \setminus S_{\alpha} \left(Q^{(r-1)}, Q\right) \), by condition (A/1) shown above, we have that \( r_Q\left(i^{(r'i)}_l\right) = b_{r-1} + \frac{p_{r-1}}{2} = b_r \). By construction of \( G(Sched) \), \( i^{(r'i)}_l \) traverses:

- edge \( e^{(r'i)}_l \) with capacity \( c\left(e^{(r'i)}_l\right) = 2b_r + \frac{b_r}{2} \);
- edge \( e^{(r'i)} \in E^{(r')} \) (Sched), for each integer \( r' \), \( 1 \leq r' < r \), in case \( r > 1 \).

We next calculate separately the increases for \( i^{(r'i)}_l \) in \( Q \) allowed by these edges.

(a) Consider first edge \( e^{(r'i)}_l \), which is also traversed by session \( i^{(r'i)}_f \). Since \( i^{(r'i)}_l \) is scheduled in front of \( Q \), it follows that \( i^{(r'i)}_f \in S_{\alpha} \left(Q^{(r-1)}, Q\right) \). Thus, condition (A/1) shown above implies that \( r_Q\left(i^{(r'i)}_f\right) = b_r + \frac{p_r}{2} \). By Claim 7.1,

\[
\Delta Q \left(i^{(r'i)}_l, e^{(r'i)}_l\right) = c\left(e^{(r'i)}_l\right) - r_Q\left(i^{(r'i)}_l\right) - \min\left\{\frac{c\left(e^{(r'i)}_l\right)}{2}, r_Q\left(i^{(r'i)}_f\right)\right\}
\]
\[
= 2b_r + \frac{p_r}{2} - b_r - \min\left\{b_r + \frac{p_r}{4}, b_r + \frac{p_r}{2}\right\}
\]
\[
= \frac{p_r}{4}.
\]
(b) Consider finally any edge $e^{(r')}$, for some integer $r'$, $1 \leq r' < r$, in case $r > 1$, which is also traversed by session $i_f^{(r')} \notin \{i_f^{(r)}, i_i^{(r)}\}$. Induction hypothesis of induction on $r$ (condition (B/3)) implies that $i_f^{(r')} \notin \mathcal{A}$. Hence, it follows that for any state $i_k = b$.

Consider now any session $r$. By Claim 7.2, it follows that state $i_k = b$. By the induction hypothesis of induction on $r$ (condition (B/2)), we have that $i_k^{(r)} = b + \frac{p_r - 1}{2}$. Hence, $r_Q^i(i_k^{(r)}) = r_Q^i(b)$.

It follows that $r_Q^i(i_k^{(r)}) = r_Q^i(b)$. Recall that $r_Q^i(b) = b_r + \frac{p_r - 1}{2}$. On the other hand, by the induction hypothesis of induction on $r$ (condition (B/2)), we have that $r_Q^i(i_k^{(r)}) = b_r + \frac{p_r - 1}{2}$. Thus, by definition of increase, it follows that $\Delta_i^Q(i_k^{(r)}, e^{(r')}) = \frac{p_r - 1}{2} = p_r$.

Thus, by definition of increase, it follows that $\Delta_Q(i_k^{(r)}) = \frac{p_r}{2}$. □

The proof of Part (A) is now complete. We continue with the proof of Part (B).

Proof of Part (B):

Proof of (B/1):

Assume, by way of contradiction, that $l_r = \infty$. Then, there exists some session $i \in \mathcal{A} \backslash \mathcal{A}$ such that $i$ is not scheduled in front of any state following $Q^{(r)}$. By condition (A/5) shown above, it follows that for any state $Q$ such that $Q^{(r)} = Q$, $\Delta_Q(i) = \frac{p_r}{2} > 0$. Since $p_r > 0$. Thus, by the update operation, the rate of session $i$ can be increased without causing any decrease to the rate of any other session with smaller rate. A contradiction to max-min fairness. □

Condition (B/1) establishes that state $Q^{(r)}$ is well defined. Recall that in Part (A) we considered all states from $Q^{(r-1)}$ through $Q^{(r)}$. We now explore further properties of $Q^{(r)}$.

Proof of (B/2):

Recall that session $i_f^{(r)}$ is scheduled in front of $Q^{(r)}$. By condition (A/1) shown above, $r \xrightarrow{Q^{(r)}} i_f^{(r)} = b_r + \frac{p_r}{2}$, while $r \xrightarrow{Q^{(r)}} i_i^{(r)} = b_r + \frac{p_r - 1}{2} = b_r$. Moreover, by condition (A/5) shown above, $\Delta \xrightarrow{Q^{(r)}} i_f^{(r)} = \frac{p_r}{2}$. Thus, by update operation, $r_Q^{(r)}(i_f^{(r)}) = r_Q^{(r)}(i_i^{(r)}) + \Delta \xrightarrow{Q^{(r)}} i_i^{(r)} = b_r + \frac{p_r}{2}$.

Consider now any session $i \in \mathcal{A} \backslash \mathcal{A}$. By definition of $Q^{(r)}$, $i \in \mathcal{S}_{\alpha} \left(Q^{(r-1)} \backslash Q^{(r)} \right)$. Thus, condition (A/1) shown above implies that $r \xrightarrow{Q^{(r)}} i = b_r + \frac{p_r}{2}$. Since, by construction of $G(Sched)$, $i \cap i_i^{(r)} = \emptyset$, the update operation implies that $r_Q^{(r)}(i) = r_Q^{(r)}(i) = b_r + \frac{p_r}{2}$. □
Proof of (B/3):

By condition (A/4) shown above, it follows that fair shares of all active edges are preserved from $Q_{(r)}$ to $Q_{(r)}$. Since $\text{Alg}$ is bottleneck and the termination condition for bottleneck algorithms is a predicate on session rates and fair shares, it follows that the only sessions that are candidates to be terminated in state $Q_{(r)}$ are those whose rates are changed in $Q_{(r)}$; by conditions (A/1) and (B/1) shown above, these are sessions $i_f^{(r)}$ and $i_l^{(r)}$.

By condition (A/3/a) shown above, edge $e_i^{(r)}$ is a bottleneck edge for $Q_{(r)}$. Condition (A/4) shown above implies that both $i_f^{(r)}, i_l^{(r)} \in A_{Q_{(r)}}$, so that $e_i^{(r)} \in AE_{Q_{(r)}}$. Thus, Proposition 4.8 (condition (2)) implies that $e_i^{(r)}$ is a bottleneck edge for $Q_{(r)}$. Recall that $FS_{Q_{(r)}}(e_i^{(r)}) = FS_{Q_{(r)}}^{-1}(e_i^{(r)})$. Thus, by condition (A/2) shown above, it follows that $FS_{Q_{(r)}}(e_i^{(r)}) = b_r + \frac{p_r}{4}$.

By condition (B/2) shown above, this implies that $r_{Q_{(r)}}(i_f^{(r)}) = r_{Q_{(r)}}(i_l^{(r)}) = FS_{Q_{(r)}}(e_i^{(r)})$. Since $\text{Alg}$ is bottleneck, it follows that $i_f^{(r)}, i_l^{(r)} \in \text{Term}(Q_{(r)})$. \hfill $\square$

Proof of (B/4):

Take any session $i \in A_{\rightarrow Q_{(r-1)}} \setminus \{i_f^{(r)}, i_l^{(r)}\}$, and fix any integer $r', 1 \leq r' \leq r$. The proof is by case analysis on $r'$.

1. Consider first the case where $r' = r$. By construction of $G(\text{Sched})$, edge $e_i^{(r)}$ with capacity $c(e_i^{(r)}) = 2b_r + p_r$ is traversed by sessions $i_f^{(r)}$ and $i$. So,

$$\text{resid}_{Q_{(r)}}(e_i^{(r)}) = c(e_i^{(r)}) - r_{Q_{(r)}}(i_f^{(r)}) - r_{Q_{(r)}}(i) = 2b_r + p_r - (b_r + \frac{p_r}{4}) = \frac{p_r}{4},$$

which establishes condition (B/4/a) in this case.

We continue with condition (B/4/b). By condition (B/3) shown above, $i_f^{(r)} \notin A_{\rightarrow Q_{(r)}}$ while $i \in A_{\rightarrow Q_{(r)}}$, so that $|A_{\rightarrow Q_{(r)}}| = 1$. Thus,

$$FS_{Q_{(r)}}^{-1}(e_i^{(r)}) = \begin{array}{c}
\text{allof}_{Q_{(r)}}(e_i^{(r)}) \\
|A_{\rightarrow Q_{(r-1)}}| e_i^{(r)}
\end{array}$$

39
The proof of Proposition 7.3 is now complete.

We are now ready to prove:

2. Consider now the case where \( r' < r \). Clearly, \( r > 1 \) in this case, so that the induction hypothesis of induction on \( r \) is non-empty. Fix any edge \( e_i^{(r')} \in E^{(r')} \) (Sched). Condition (B/3) shown above implies that \( i_f^{(r')} \notin A \xrightarrow{\text{Q}^{(r-1)}} \), so that \( r_{Q^{(r)}}(i_f^{(r')}) = r_{Q^{(r-1)}}(i_f^{(r')}) \).

By condition (B/2) shown above, \( r_{Q^{(r)}}(i) = b_r + \frac{p_r}{2} \), while by the induction hypothesis of induction on \( r \) (condition (B/2)), \( r_{Q^{(r-1)}}(i) = b_{r-1} + \frac{p_{r-1}}{2} = b_r \). Thus,

\[
\text{resid}_{Q^{(r-1)}}(e_i^{(r')}) - \text{resid}_{Q^{(r)}}(e_i^{(r')}) = \left( c e_i^{(r')} - r_{Q^{(r-1)}}(i_f^{(r')}) - r_{Q^{(r-1)}}(i) \right) - \left( c e_i^{(r')} - r_{Q^{(r)}}(i_f^{(r')}) - r_{Q^{(r)}}(i) \right) = \left( b_r + \frac{p_r}{2} \right) - b_r - 0 = \frac{p_r}{2}.
\]

By the induction hypothesis of induction on \( r \) (condition (B/4/a)), \( \text{resid}_{Q^{(r-1)}}(e_i^{(r')}) \geq \frac{p_{r-1}}{2} = p_r \). It follows that \( \text{resid}_{Q^{(r)}}(e_i^{(r')}) \geq p_r - \frac{p_r}{2} > \frac{p_r}{2} \), which establishes condition (B/4/a) in this case.

We continue with condition (B/4/b). Recall that \( i_f^{(r')} \notin A \xrightarrow{\text{Q}^{(r-1)}} \), so that \( i_f^{(r')} \notin A \xrightarrow{\text{Q}^{(r)}} \), and \( r_{Q^{(r-1)}}(i_f^{(r')}) = r_{Q^{(r)}}(i_f^{(r')}) \); on the other hand, \( i \in A \xrightarrow{\text{Q}^{(r-1)}} \), so that \( i \in A \xrightarrow{\text{Q}^{(r)}} \). It follows that \( FS_{Q^{(r-1)}}(e_i^{(r')}) = FS_{Q^{(r)}}(e_i^{(r')}) \). By the induction hypothesis of induction on \( r \) (condition (B/4/b)), \( FS_{Q^{(r-1)}}(e_i^{(r')}) \geq b_{r-1} + \frac{3p_{r-1}}{4} = b_r + p_r \). Thus, \( FS_{Q^{(r)}}(e_i^{(r')}) \geq b_r + p_r > b_r + \frac{3p_r}{4} \), which establishes condition (B/4/b) in this case.

The proof of Proposition 7.3 is now complete.

We are now ready to prove:

**Theorem 7.4 (Lower bound for oblivious algorithms)** Assume that Alg is optimistic, oblivious and bottleneck, and it computes the max-min fair rate vector. Then, \( U_{\text{Alg}} \geq \frac{dn}{n} + \frac{n}{2} \).
Proof: We will show that \( U_{Alg}(G(Sched), S) = \frac{dn^4}{4} + \frac{n^2}{2} \), where \( Alg = (Sched, Term) \) and \( G(Sched) \) is the network constructed in Section 6. To do so, we calculate \( U_{Alg}(G(Sched), S_j) \) for any particular cluster \( S_j, j \geq 1 \), and we add up over all \( \frac{n}{d} \) clusters. By Proposition 7.3 (condition (B/1)), the execution of \( Alg \) on \( G(Sched | S_j) \) is divided into \( \frac{d}{2} \) execution epochs \( \alpha^{(1)}, \ldots, \alpha^{(d/2)} \) such that all active sessions are scheduled at least once in each epoch; condition (B/3) implies that only two such sessions are terminated in each epoch. Thus, at least \( d - 2(r - 1) \) update operations are executed in \( \alpha^{(r)}, 1 \leq r \leq \frac{d}{2} \). Summing up over all clusters and execution epochs yields that \( U_{Alg}(G, S) \geq \sum_{j=1}^{n/d} \sum_{r=1}^{d/2} (d - 2(r - 1)) = \frac{dn^4}{4} + \frac{n^2}{2} \), as needed. 

7.2 Partially Oblivious Algorithms

In this section, we present a lower bound of \( \Omega(dn) \) on the convergence complexity of any optimistic, partially oblivious and bottleneck algorithm that computes the max-min rate vector.

For the sake of clarity of presentation, we first consider the special case where \( d = n \). (We remark that this is the case where the lower bound to be shown is maximum over all possible values of \( d, 1 \leq d \leq n \).) The backbone of our proof for this case will be a Simulation Lemma that establishes a correspondence between the executions of any partially oblivious algorithm and a suitable oblivious algorithm on the network constructed from the latter as in Section 6 (assuming \( d = n \)). We will then use the Simulation Lemma to prove a lower bound of \( \Omega(n^2) \) for this case. We finally consider the general case of arbitrary \( d \); we state the \( \Omega(dn) \) lower bound for it and we discuss its proof, which relies on a generalization of the Simulation Lemma to general \( d \). We start with the Simulation Lemma for the case \( d = n \).

Proposition 7.5 (Simulation Lemma) Assume that \( Alg \) is optimistic, partially oblivious and bottleneck, and it computes the max-min fair rate vector. Then, there exists some optimistic, oblivious and bottleneck algorithm \( O Alg = (OSched, OTerm) \) such that there exist execution prefixes \( \alpha = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_{d/2} \) of \( O Alg \) and \( \beta = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_{d/2} \) of \( Alg \), on network \( G(OSched) \), such that for each integer \( r, 1 \leq r \leq \frac{d}{2} \), the following conditions hold:

1. \( \alpha_r \) and \( \beta_r \) are identical;
2. all sessions active in first \( (\beta_r) \) are scheduled at least once in \( \beta_r \) and only two of them terminate in \( \beta_r \).

We first provide an informal, high-level outline of our proof. Given any partially oblivious algorithm, we derive some oblivious algorithm such that the executions of the two algorithms
on the network constructed from the oblivious one in Section 6 are identical. The execution of the oblivious algorithm is described by Proposition 7.3; since the two executions are shown to be identical, this will eventually imply the claimed lower bound for the partially oblivious algorithm. In order to derive the oblivious algorithm with the required properties, we start with the set of all \( n \)-epoch, oblivious algorithms; we inductively restrict this set until it contains only algorithms that are compatible with the partially oblivious algorithm (with respect to their schedulers) and perform sufficiently many update operations. All intermediate sets of oblivious algorithms contain only algorithms that induce executions identical to the partially oblivious algorithm up to each intermediate point. In more detail, the partial execution of each oblivious algorithm in any intermediate set on the network constructed from the algorithm in Section 6 is identical to the partial execution of the partially oblivious algorithm on the same network. Each individual intermediate step is extended progressively by considering the session scheduled next by the partially oblivious algorithm and restricting the intermediate set of oblivious algorithms to those that schedule the same session next. (The construction follows an outer induction on execution epochs and an inner induction on states within each epoch.)

**Proof:** For each index \( r \), \( 1 \leq r \leq \frac{d}{2} \), we will construct a set \( A^{(r)} \) of optimistic, oblivious and bottleneck algorithms that compute the max-min fair rate vector such that for each algorithm \( \text{OAlg} \in A^{(r)} \), there exist execution prefixes \( \alpha (\text{OAlg}) = \alpha_1 (\text{OAlg}) \ldots \alpha_r (\text{OAlg}) \) of \( \text{OAlg} \) and \( \beta (\text{OAlg}) = \beta_1 (\text{OAlg}) \ldots \beta_r (\text{OAlg}) \) of \( \text{Alg} \) on network \( G (\text{OSched}) \) so that:

1. for each algorithm \( \text{OAlg} \in A^{(r)} \), for each integer \( r' \), \( 1 \leq r' \leq r \), \( \alpha_{r'} (\text{OAlg}) \) and \( \beta_{r'} (\text{OAlg}) \) are identical;
2. \( \alpha (\text{OAlg}) \) is identical over all algorithms \( \text{OAlg} \in A^{(r)} \);
3. \( \beta (\text{OAlg}) \) is identical over all algorithms \( \text{OAlg} \in A^{(r)} \);
4. for each algorithm \( \text{OAlg} \in A^{(r)} \), for each integer \( r' \), \( 1 \leq r' \leq r \), all sessions active in state \( \text{first} (\beta_{r'} (\text{OAlg})) \) are scheduled at least once in \( \beta_{r'} (\text{OAlg}) \) but only two of them terminate in \( \beta_{r'} (\text{OAlg}) \).

Any algorithm \( \text{OAlg} \) in the final set \( A^{(d/2)} \) will satisfy the claim due to conditions 1 and 4 guaranteed by the construction.

The construction will be by induction on \( r \), \( 1 \leq r \leq \frac{d}{2} \). The construction employs the set of all optimistic, oblivious, bottleneck, \( n \)-epoch algorithms, which we denote as \( A^{(0)} \). By Theorem 5.2, any algorithm from \( A^{(0)} \) computes the max-min fair rate vector. Our construction
will progressively restrict the set $A^{(0)}$ so that $A^{(d/2)} \subseteq \ldots \subseteq A^{(r)} \subseteq \ldots \subseteq A^{(0)}$. Hence, for each integer $r, 1 \leq r \leq \frac{d}{2}$, any algorithm from $A^{(r)}$ computes the max-min fair rate vector. This property will be needed later, whenever we use Proposition 7.3 that assumes it. For each algorithm $OAlg \in A^{(0)}$, consider the initial state $Q_0 (OAlg)$ of network $G (OSched)$ (with all rates zero and all sessions active). Set $\alpha_0 (OAlg) := Q_0 (OAlg)$ and $\beta_0 (OAlg) := Q_0 (OAlg)$. For the sake of shortening the construction, we merge the construction for the basis case (where $r = 1$) and the construction for the induction step. Thus, the case $A^{(r)}$ with the required properties. Notice that if $\alpha_0 (OAlg)$ implies now that the set of active sessions in state $A^{(r)}$ has not been scheduled at least once in any such active session $i (OAlg)$ is complete and conditions (1), (2) and (3) are preserved along the inductive construction on states, while (4) holds when the construction of $A^{(r)}$ is complete.

For the basis case (of induction on states), take $A^{(r)} := A^{(r-1)}$; for any algorithm $OAlg = (OSched, OTerm) \in A^{(r-1)}$, set $\alpha_r (OAlg) = last (\alpha_{r-1} (OAlg))$ and $\beta_r (OAlg) = last (\beta_{r-1} (OAlg))$. In case $r = 1$, by definition of initial state $(Q_0 (OAlg))$, $\alpha_1 (OAlg)$ (resp., $\beta_1 (OAlg)$) is identical for all algorithms $OAlg \in A^{(1)}$, implying (2) and (3). By construction, for each algorithm $OAlg \in A^{(1)}$, it trivially holds that $\alpha_1 (OAlg)$ and $\beta_1 (OAlg)$ are identical, which implies (1). In case $r > 1$, by the induction hypothesis of induction on $r$, conditions (1), (2) and (3) hold.

We now continue with the induction step (induction on states) of the inductive construction of $A^{(r)}$. Take any algorithm $OAlg \in A^{(r)}$ and consider $\alpha_r (OAlg)$. (By the induction hypothesis of induction on states (condition (2)), the choice of $OAlg$ does not matter.) In case $r = 1$, all sessions are active in state $last (\beta_{r-1} (OAlg)) = Q_0 (OAlg)$. In case $r > 1$, since $r - 1 \leq \frac{d}{2}$. Proposition 7.3 (condition (B/3)) implies that the set of active sessions in state $last (\alpha_{r-1} (OAlg))$ is non-empty; the induction hypothesis of induction on states (condition (1)) implies now that the set of active sessions in state $last (\beta_{r-1} (OAlg))$ is non-empty as well. If all such sessions have been scheduled at least once in $\alpha_r (OAlg)$, then the inductive construction of $A^{(r)}$ is complete and conditions (1), (2) and (3) hold by the induction hypothesis of induction on states. (Condition (4) will be shown later.) So assume that there exists at least one such active session $i (OAlg)$ that has not been scheduled in $\beta_r (OAlg)$. By the induction hypothesis of induction on states (condition (1)), $\alpha_r (OAlg)$ and $\beta_r (OAlg)$ are identical. It follows that $i (OAlg)$ has not been scheduled in $\alpha_r (OAlg)$ either. Proposition 7.3 (condition
(A/5)) implies that the increase for session \(i(OAlg)\) in state last \((\alpha_r(OAlg))\) is non-zero. By the induction hypothesis of induction on states (condition (1)), it follows that the increase for session \(i(OAlg)\) in state last \((\beta_r(OAlg))\) is non-zero as well. By update operation, it follows that an increase to the rate of \(i(OAlg)\) in last \((\beta_r(OAlg))\) is possible without decreasing the rate of any other session. So, max-min fairness has not been reached yet and some session rate must change. However, a session rate changes only when an active session is scheduled. Since Alg computes the max-min fair rate vector, it follows that at least one active session in last \((\beta_r(OAlg))\) is scheduled after last \((\beta_r(OAlg))\). An inductive application of this argument implies that all sessions active in first \((\beta_r(OAlg))\) are scheduled at least once in \(\beta_r(OAlg)\). It follows that the inductive construction of \(A^{(r)}\) eventually terminates.

Denote \(i'(OAlg)\) the session scheduled by Alg immediately after last \((\beta_r(OAlg))\) on network \(G(OSched)\). By the induction hypothesis of induction on states (condition (3)), it follows that last \((\beta_r(OAlg))\) is identical for all \(OAlg \in A^{(r)}\). Since Alg is partially oblivious, this implies that \(i'(OAlg)\) is identical for all \(OAlg \in A^{(r)}\) as well; so, denote it \(i'\). Fix any algorithm \(OAlg \in A^{(r)}\) and restrict \(A^{(r)}\) to the set of all optimistic, oblivious, bottleneck, \(n\)-epoch algorithms whose schedulers have prefix \(\sigma(\alpha_r(OAlg))\) \(i'\).

We argue that \(A^{(r)}\) is non-empty and unique. Any sequence of sessions \(\sigma(\beta_r(OAlg))\) \(i'\) can be extended to an \(n\)-epoch scheduler (by appropriate padding). This implies that \(A^{(r)}\) is non-empty. By the induction hypothesis of induction on states (condition (3)), \(\beta_r(OAlg)\) is identical over all algorithms \(OAlg \in A^{(r)}\), so that \(\sigma(\beta_r(OAlg))\) \(i'\) is also identical over all \(OAlg \in A^{(r)}\). It follows that the constructed \(A^{(r)}\) is unique.

Take any algorithm \(OAlg = (OSched, OTerm) \in A^{(r)}\). Define \(Q(OAlg)\) to be the state that results when Alg, starting from state last \((\beta_r(OAlg))\) of the network \(G(OSched)\), schedules session \(i'\) on \(G(OSched)\) and set \(\beta_r(OAlg) := \beta_r(OAlg)\), \(i'\), \(Q(OAlg)\). Similarly, define \(Q'(OAlg)\) to be the state that results when \(OAlg\), starting from last \((\alpha_r(OAlg))\) of the network \(G(OSched)\), schedules session \(i'\) on \(G(OSched)\) and set \(\alpha_r(OAlg) := \alpha_r(OAlg)\), \(i'\), \(Q'(OAlg)\). We now prove the required properties for the sets \(\alpha(OAlg) = \{\alpha_1(OAlg) \cdots \alpha_r(OAlg) \mid OAlg \in A^{(r)}\}\) and \(\beta(OAlg) = \{\beta_1(OAlg) \cdots \beta_r(OAlg) \mid OAlg \in A^{(r)}\}\) of execution prefixes.

We start to prove (1). Take any algorithm \(OAlg = (OSched, OTerm) \in A^{(r)}\). By the induction hypothesis of induction on states (condition (1)), Alg and OAlg start from identical states of the network \(G(OSched)\); hence, scheduling the same session \(i'\) on the network \(G(OSched)\) results in identical states, which inductively extends the claim.

We now treat (2) and (3). Since \(i'(OAlg)\) is identical over all \(OAlg \in A^{(r)}\), the induction hypothesis of induction on states (condition (2)) implies that the set of sessions scheduled in
\(\alpha_r(O\text{Alg})\) is identical over all \(O\text{Alg} \in A^{(r)}\). So, Proposition 7.3 (conditions (A/1) and (A/4)) implies that \(Q'(O\text{Alg})\) is identical over all \(O\text{Alg} \in A^{(r)}\). Induction hypothesis of induction on states (condition (2)) and the fact that \(i'(O\text{Alg})\) is identical over all \(O\text{Alg} \in A^{(1)}\) imply that \(\alpha_r(O\text{Alg})\) is identical over all \(O\text{Alg} \in A^{(r)}\), which proves (2). Now (3) follows from (1) and (2).

We finally prove (4) for the set \(A^{(r)}\) constructed when the induction on states is complete. Fix any algorithm \(O\text{Alg} \in A^{(r)}\). Recall that all sessions active in first \((\beta_r(O\text{Alg}))\) are scheduled at least once in \(\beta_r(O\text{Alg})\); moreover, the termination condition used in the construction of \(A^{(r)}\) implies that \(\beta_r(O\text{Alg})\) is the shortest such fragment. By condition (1) shown above, this implies that \(\alpha_r(O\text{Alg})\) is the shortest execution fragment such that all sessions active in first \((\alpha_r(O\text{Alg}))\) are scheduled at least once in \(\alpha_r(O\text{Alg})\). So, by Proposition 7.3 (condition (B/3)), only two sessions terminate in \(\alpha_r(O\text{Alg})\). Hence, the induction hypothesis of induction on states (condition (1)) implies that only two sessions terminate in \(\beta_r(O\text{Alg})\), as needed.

The proof of the Simulation Lemma is now complete.

We are now ready to prove the lower bound for the case where \(d = n\).

**Theorem 7.6 (Lower bound for partially oblivious algorithms)** Assume that \(\text{Alg}\) is optimistic, partially oblivious and bottleneck, and it computes the max-min rate vector. Then,

\[ U_{\text{Alg}} \geq \frac{n^2}{4} + \frac{n}{2}. \]

**Proof:** Proposition 7.5 implies that there exists some optimistic, oblivious and bottleneck algorithm \(O\text{Alg} = (O\text{Sched}, \text{OTerm})\) such that there exist an execution prefix \(\beta = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_{d/2}\) of \(\text{Alg}\) on network \(G(O\text{Sched})\) such that for each integer \(r, 1 \leq r \leq \frac{n}{2}\), all sessions active in first \((\beta_r)\) are scheduled at least once in \(\beta_r\) and only two of them terminate in \(\beta_r\). Hence, at least \(n - 2(r - 1)\) update operations are executed in \(\beta_r, 1 \leq r \leq \frac{n}{2}\). Summing up over all \(r\) epochs, \(1 \leq r \leq \frac{n}{2}\), yields that \(U_{\text{Alg}}(G(O\text{Sched}), S) \geq \sum_{r=1}^{n/2} (n - 2(r - 1)) = \frac{n^2}{4} + \frac{n}{2}. \)

As a direct generalization of the Simulation Lemma for the case \(d = n\), we obtain:

**Proposition 7.7 (Simulation Lemma)** Assume that \(\text{Alg}\) is optimistic, partially oblivious and bottleneck, and it computes the max-min fair rate vector. Partition \(S\) into disjoint clusters \(S_1, \ldots, S_{n/d}\) with \(d\) sessions each. Then, there exists some optimistic, oblivious and bottleneck algorithm \(O\text{Alg} = (O\text{Sched}, \text{OTerm})\) such that there exist execution prefixes \(\alpha\) of \(O\text{Alg}\) and \(\beta\) of \(\text{Alg}\), on network \(G(O\text{Sched})\), such that for each cluster \(S_j, 1 \leq j \leq \frac{n}{d}\), \(\alpha\) and \(\beta\) can be written as \(\alpha = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_{\frac{n}{d}}\) and \(\beta = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_{\frac{n}{d}}\) so that for each integer \(r, 1 \leq r \leq \frac{n}{d}\), the following conditions hold:
(1) \(\alpha_r\) and \(\beta_r\) are identical;

(2) all sessions from \(S_j\) active in first \((\beta_r)\) are scheduled at least once in \(\beta_r\) and only two of them terminate in \(\beta_r\).

The execution prefixes \(\alpha\) and \(\beta\) in Proposition 7.7 are constructed in a similar inductive manner as the corresponding prefixes in Proposition 7.5. The additional complications stem from the fact that the execution fragments \(\alpha_r\) and \(\beta_r\), \(1 \leq r \leq \frac{d}{2}\), may now be different for each cluster; thus, the inductive construction in the proof of Proposition 7.5 needs to be adjusted in order to accommodate chopping off \(\alpha\) and \(\beta\) into the suitable execution fragments \(\alpha_r\) and \(\beta_r\), \(1 \leq r \leq \frac{d}{2}\), for each particular cluster. We finally use Proposition 7.7 to show our final lower bound; its proof is similar to the one of Theorem 7.6 that used Proposition 7.5.

**Theorem 7.8 (Lower bound for partially oblivious algorithms)** Assume that \(\text{Alg}\) is optimistic, partially oblivious and bottleneck, and it computes the max-min rate vector. Then, 
\[
U_{\text{Alg}} \geq \frac{dn}{4} + \frac{n}{2}.
\]

Consider the partially oblivious algorithm \(\text{GlobalMinSched}\) introduced by Afek et al. [1, Section 4], whose scheduler chooses, for each state \(Q\), the active session with the minimum rate. We note that [1, Theorem 4.3] implies an upper bound \(\frac{|S_j||S_j|+1}{2}\) on the number of update operations executed by \(\text{GlobalMin}\) on network \(G\) with session set \(S_j\), for any particular cluster \(S_j\). So, 
\[
U_{\text{GlobalMin}} \leq \sum_{j \geq 1} \frac{|S_j||S_j|+1}{2} \leq \sum_{j \geq 1} |S_j| \left(\frac{|S_j|+1}{2}\right) \leq \frac{dn}{2} + \frac{n}{2}. 
\]
This implies that the lower bound established in Theorem 7.8 is tight (within a factor of 2).

8 Discussion and Directions for Further Research

We have presented a comprehensive collection of lower and upper bounds on the convergence complexity of optimistic, bottleneck, rate-based flow control algorithms, under varying degrees of the knowledge used by the scheduling component of the algorithms. In particular, we have defined and studied oblivious, partially oblivious, and non-oblivious algorithms. We have shown that, perhaps surprisingly, the classes of oblivious algorithms and partially oblivious algorithms collapse with respect to convergence complexity; we have also shown a convergence complexity separation between (partially) oblivious algorithms and non-oblivious algorithms.

For the sake of completeness and comparison, we summarize in Table 2 all known lower and upper bounds on the convergence complexity of optimistic, bottleneck algorithms for rate-based flow control, established in this work and in the preceding work by Afek et al. [1].
Figure 2: Summary of known lower and upper bounds on convergence complexity for optimistic, bottleneck rate-based flow control algorithms

remark that RoundRobin represents an exponential improvement over the previous algorithm Arbitrary [1, Section 6] for the class of oblivious algorithms we introduced. (The algorithm Arbitrary schedules sessions in any arbitrary way.)

Our work leaves open several important questions. The most obvious open question would be to close the gap between the lower bound of $dn^4 + n^2$ and the upper bound of $dn^2 + n^2$ we have shown on the convergence complexity of oblivious algorithms.

The model considered in this work is simple and elegant, yet structured enough to capture several significant ingredients of distributed rate-based flow control; there remain, however, a number of significant practical issues untouched by our model and analysis. In the first place, we feel that the max-min fairness criterion may be undue in some realistic situations, where sessions have different demands. (Some results in this direction have already been obtained in [9].) Second, the limitation to static sets of sessions is somehow restrictive; it would be significant to extend our model and techniques to handle set-up and take-down of sessions. Third, practical considerations may demand that rate-based flow control algorithms avoid too small or too large adjustments to session rates. Encompassing such practical considerations, and analyzing their impact on convergence complexity, into the framework of rate-based flow control algorithms is an interesting research problem.

Kleinberg et al. [19] formulate some natural approximations to max-min fairness and advocate them as suitable fairness conditions for certain routing and load balancing applications. It would be interesting to study the convergence complexity of such approximations within the framework of rate-based flow control algorithms.

Acknowledgments: Our thanks go to the anonymous PODC 1997 and SIAM Journal on Computing reviewers for their helpful comments.
References


