Max-Min Fair Flow Control Sensitive to Priorities*

Panagiota Fatourou[†] *Marios Mavronicolas*[‡] *Paul Spirakis*[§]

July 4, 1999

*A preliminary version of this work appears in the *Proceedings of the 2nd International Conference on Principles of Distributed Systems (OPODIS'98),* pp. 45–59, Amiens, France, December 1998.

[†]Department of Computer Engineering and Informatics, University of Patras, Rion, 265 00 Patras, Greece & Computer Technology Institute, P. O. Box 1122, 261 10 Patras, Greece. Partially supported by the EU ESPRIT Long Term Research Project ALCOM-IT (Proj. # 20244), and by the Greek Ministry of Education. Email: faturu@cti.gr

[‡]Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269-3155. Part of the work of this author was performed while at Department of Computer Science, University of Cyprus, Cyprus, and while at AT&T Labs – Research, NJ, as a visitor to the Special Year on Networks, DIMACS Center for Discrete Mathematics and Theoretical Computer Science, NJ. Email: mavronic@engr.uconn.edu

[§]Department of Computer Engineering and Informatics, University of Patras, Rion, 265 00 Patras, Greece, & Computer Technology Institute, P. O. Box 1122, 261 10 Patras, Greece. Partially supported by the EU ESPRIT Long Term Research Project ALCOM-IT (Proj. # 20244), and by the Greek Ministry of Education. Email: spirakis@cti.gr

Abstract

Flow control is the dominant technique currently used in communication networks for preventing excess traffic from "flooding" the network, and for handling congestion. In *rate-based* flow control, transmission rates of sessions are adjusted in an end-to-end manner through a sequence of *operations,* each preserving the *capacity constraint* at each individual network link. In this work, we present a theory of *max-min fair,* rate-based flow control sensitive to *priorities* of different sessions, as a significant extension of the classical theory of max-min fair, rate-based flow control to the case of networks with guaranteed *quality-of-service* (QoS).

We assume a *priority function* for each individual session, which maps the session's nominated priority to a transmission rate. We introduce *priority max-min fairness,* as a novel and motivatable fairness condition which requires that assigned rates correspond, through the priority functions, to priorities comprising a max-min vector. A cornerstone of our theory is the notion of the *priority share* of a link; a priority share is defined as a root of an algebraic equation, called *priority max-min fairness equation,* which determines the saturation of the link. It is shown that, under reasonable mathematical assumptions on the priority functions, priority shares exist uniquely.

Priority bottleneck algorithms finalize the rate of a session only if its priority is "restricted" on a *priority bottleneck edge* of the network. We establish a collection of interesting combinatorial properties of priority bottleneck algorithms. Most significantly, we show that they can only converge to priority max-min fairness.

As an application of our general theory, we embed priority bottleneck algorithms in the more realistic *optimistic* framework for rate-based flow control, which was recently proposed by Afek *et al.* [1]. The optimistic framework allows for both decreases and increases of session rates. We exploit the additional semantics provided by explicitly allowing for rate decreases in order to prove further combinatorial properties for the termination of priority bottleneck algorithms in the optimistic framework. In reward, we build on these properties to devise and analyze the first optimistic algorithms for efficient, max-min fair, rate-based flow control sensitive to priorities.

1 Introduction

1.1 Motivation-Overview

Current *communication protocols* ranging from ATM with Available Bit Rate (ABR) traffic [4, 13, 19, 27] to TCP/IP [6, 11] have been employing a relatively simple, "best-effort" service policy in order to allow for large amounts of high-speed traffic. Nevertheless, modern distributed applications such as remote video, multimedia conferencing, and data visualization, often require the network to go beyond the "best-effort" policy and allow data traffic stream, henceforth called *flow,* to reserve critical network resources, such as bandwidth, on its own. Thus, in recent years, there has been an increasing tendency to embed new communication protocols and service models into existing network architectures in order to accomodate these new application requirements and provide the applications with guaranteed *Quality-of-Service,* commonly abbreviated as *QoS.* Nowadays, the need for networks with guaranteed QoS is widely recognized; see, e.g., [26].

In particular, ATM networks have been accomodating Constant Bit Rate (CBR) and Variable Bit Rate (VBR) flows in an attempt to serve widely varying kinds of traffic.* In parallel to this, there has been a flurry of research activity towards the equally challenging task of incorporating mechanisms for flow specification, resource reservation, and admission control into Internet (see, e.g., [10, 23, 33, 34]), in an effort to make Internet conform to the new application requirements. (See also [5].)

It is worth mentioning a recent resource reservation protocol called *RSVP* [37], which has been advocated to accomodate heterogeneous requirements of distributed user applications in single multicasting groups. In *RSVP,* distributed applications signal to the network their requirements, and the protocol reserves resources via the network switches (see [37] or [36, Section 3.9.1] for an expanded discussion.) Remarkably enough though, *RSVP* has not relied on any underlying formal model of fairly servicing the requirements of distributed applications; instead, it has employed an interesting and intuitive scheme, called *weighted fair queuing,* that reserves network resources on the basis of priorities called "weights". Preliminary experimental investigations [37] reveal that *RSVP* implementations are both flexible and scalable.

The assignment of *priorities* to distributed applications provides a way of modeling their requirements on reservation of network resources. On the other hand, *fairness* considerations

*Roughly speaking, both CBR and VBR allow for service guarantees such as delay, cell loss ratio, and bandwidth to be negotiated at call set-up time through admission control and bandwidth allocation in a rather ad-hoc way. In contrast, the ABR service guarantees a cell loss ratio only to those connections whose source dynamically adapts to feedback received from the network. (See, e.g., [36].)

aim at prohibiting the network from overly crediting prioritized applications with reservations of excessive resource amounts on the account of poorly treated applications. Thus, a fundamental challenge of modern networking is the incorporation of both priorities and fairness in a formal, integrated service model for a heterogeneous distributed system encompassing user applications with varying requirements and reservation rights to network resources.

In this work, we take a first step in this direction by assembling a theory of *max-min fair, rate-based flow control* sensitive to priorities assigned to distributed applications, as a significant extension of the classical theory of max-min fair, rate-based flow control (see, e.g., [2, 7, 20, 21, 22]) to the setting with priorities. In a nutshell, our theory advocates a new fairness condition we call *priority max-min fairness,* which accounts for priorities, the study of a class of algorithms, termed *priority bottleneck algorithms,* that are found to converge to priority max-min fairness, and the derivation and analysis of specific such algorithms in the more realistic, *optimistic* framework recently put forward by Afek *et al.* [1] and further studied by Fatourou *et al.* [15, 16].

1.2 Detailed Description

We consider a *communication network* with a collection of distributed applications, modeled as virtual circuits of infinite duration[†] and called *sessions,* running on it. Associated with each session we envision a non-negative real number, called *priority,* that provides a working abstraction which we use in our work for modeling the session's relative merit to network bandwidth; this merit is substantiated by the *rate* it subsumes on transmission. However, *capacity* limitations on individual network *links* constrain the achievable rates.

We assume a *priority function* for each session, which maps its priority to a transmission rate. Priority functions may be postulated on the basis of network management policies, traffic levels, or pricing considerations (cf. [12, 25, 32]). In our setting, priority functions provide a means for expressing the *desired* level of QoS for each particular application, while the values of priority functions, on abstract arguments that we called priorities, specify the *actual* level of QoS supplied to them. Naturally, we assume common mathematical properties, such as increasing monotonicity, continuity and convexity, on the priority functions.

The value at zero of each particular priority function determines the lowest possible rate the corresponding session may attain. In order to guarantee *adequacy,* we assume that each link capacity is no less than the sum of the values at zero of the priority functions of sessions

[†]This assumption is appropriate for applications such as ftp, batch jobs, and Internet navigation tools.

traversing the link. We consider an abstract *operation* manipulating session rates; by invertibility of the priority functions, this operation may be viewed in an equivalent way as manipulating priorities. Naturally, we assume that the operation respects the capacity constraint for each individual link.

One of the most challenging aspects of modern networking is treating all sessions fairly when it is necessary to turn some of the bursty traffic away from the network. The most widely accepted fairness condition is *max-min fairness* [1, 2, 7, 15, 16, 17, 18, 20, 21], which, roughly speaking, requires that it be impossible to infinitesimally increase the rate of any application without decreasing the rate of some other with a smaller rate. However, this condition is inadequate in case different applications have acquired varying priorities on the basis of disproportional charging.

Our first major contribution is the formulation of a "crisp" fairness condition that takes priorities into account in an explicit and formal way. We define *priority max-min fairness* by requiring that it be impossible to infinitesimally increase the rate of any application without decreasing the rate of some other with smaller priority. Thus, according to priority max-min fairness, the significant factor that determines whether or not the rate of an application may further increase is its priority (rather than its rate itself). So, priority max-min fairness allows for a quantifiable discrimination against different sessions on the basis of their priorities; thus, an intensive and demanding distributed application may invest in purchasing a priority function taking large values for itself, which, in turn, will secure sufficient bandwidth for its transmission in a fair manner (in view of its investment). Thus, priority max-min fairness is a fairness condition that explicitly allows for arbitrarily prioritized allocation of bandwidth.

A further cornerstone of our general theory is the notion of the priority share of a link. In order to accomodate this notion, we formulate, for each link, a particular algebraic equation called the *priority max-min fairness equation*; roughly speaking, this equation expresses the natural requirement that sessions traversing the link are all assigned *equal* priorities that correspond, via the priority functions, to rates saturating the link. We define the *priority share* of the link as a root of its priority max-min fairness equation. Most importantly, we use the definition of priority share, mathematical properties of the priority functions, the adequacy assumption, and the fact that operations respect the capacity constraints to carefully prove that priority shares exist *uniquely*. This uniqueness outlaws any possible ambiguity that may arise while using the priority share for further developing our theory.

In particular, priority bottleneck edges are well defined owing to the unique existence of priority shares. Roughly speaking, each session traversing a *priority bottleneck edge* "observes" the smallest priority share, among all edges it traverses, on this edge. We proceed to define *priority bottleneck algorithms* to be those that never finalize the rate of a session unless the priority it attains matches the priority share of some edge it traverses that is priority bottleneck.

These definitions are demonstrated to be both natural and stable generalizations of the classical definitions of *bottleneck edges* and *bottleneck algorithms* [20, 21, 22]: we prove a collection of interesting combinatorial properties of priority bottleneck algorithms that generalize corresponding properties of bottleneck algorithms in an elegant way. Most significantly, we establish the priority "analog" of the relation between bottleneck algorithms and (classical) max-min fairness: we prove that priority bottleneck algorithms may only converge to priority max-min fairness, much in the same way bottleneck algorithms converge to max-min fairness (see [2, Section 6.5] for a textbook discussion).

We finally proceed to apply our general theory to the more realistic, *optimistic* framework for rate-based flow control that was recently proposed by Afek *et al.* [1] and further examined by Fatourou *et al.* [15, 16]. (See [14] for a survey.) In contrast to the so called conservative approach where session rates never decrease during a sequence of adjustments, *optimistic* algorithms allow rates to intermediately go above their final rates. This assumption is more realistic since it provides for both increases and decreases of sessions in order to accomodate new entering sessions in a dynamic manner. Moreover, this approach comes "closer" to the proposed rate-based approach of the ATM Forum as the standard for congestion control of ABR services.

We enhance the optimistic framework with a *priority update operation,* which is a simple abstraction of adjusting, on the basis of priorities, the rates of individual sessions in an end-to-end manner. The complexity measure of an algorithm converging to priority max-min fairness, called here *convergence complexity* [1, 15, 16], is the number of priority update operations executed in the worst case. We exploit the additional semantics of priority update operations in order to establish further combinatorial properties of priority bottleneck algorithms within the optimistic framework; these properties concern the way in which individual sessions reach their priority max-min fair rates when manipulated by such optimistic algorithms.

Our final result is the derivation of, and analysis of convergence complexity for, *specific* priority bottleneck algorithms in the optimistic framework. One such algorithm is *oblivious*[‡] and reasonably efficient in terms of convergence complexity; a second algorithm is *non-oblivious,* hence, less convenient to implement, but it achieves optimal efficiency in terms of convergence complexity. These results provide an elegant generalization of corresponding recent results of Fatourou *et al.* [15] for the much simpler case of bottleneck algorithms, to the setting with priorities.

[‡]Say that an algorithm is *oblivious* [15] if it schedules sessions for an update operation without "knowledge" of either network structure or session rates (or priorities).

1.3 Road Map

The rest of this paper is organized as follows. Section 2 surveys and contrasts related work. Our formal definitions are laid out in Section 3. Section 4 introduces priority max-min fairness and priority share, while Section 5 defines and studies priority bottleneck algorithms. Fundamental termination properties of priority bottleneck algorithms are shown in Section 6. The class of optimistic, priority bottleneck algorithms is studied in Section 7. We conclude, in Section 8, with a discussion of our results.

2 Related Work and Comparison

Similar versions of max-min fairness were proposed independently by Hayden [20], Jaffe [21, 22], and Luss and Smith [29]. The classical theory of fair, rate-based flow control, also referred to as *bottleneck flow control* [21], was built around max-min fairness (see, e.g., [2, 18, 20, 21, 22]). All of that theory adopted the conservative approach. The optimistic framework was introduced by Afek *et al.* [1] and further studied by Fatourou *et al.* [15, 16]. (A survey of work in the optimistic framework appears in [14].) None of these previous works deals with priorities (with [21] being a single exception; see below).

Asynchronous distributed algorithms that converge to max-min fair rates have been presented in [7, 8, 9, 24, 31, 35] (this list is not exhaustive); all of them are conservative. Corresponding optimistic algorithms have been pointed out by Fatourou *et al.* [16], as a result of implementing in an asynchronous and distributed way corresponding algorithms presented in [15] which were described to work in a sequence of non-overlapping operations. Still, priority issues were left untouched by those works.

Work by Jaffe [21, Section X] and by Gafni and Bertsekas [17] (motivated by a voice coder scheme) are the only works known to us that attempt to address priority issues in the context of max-min fair, rate-based flow control.

Jaffe introduces the notion of *throughput priority* [21, Section X] to capture preferential treatment of some sessions (users) in a heterogeneous session set. Roughly speaking, the throughput priority of a session is a positive real number that multiplies its rate to yield its desired throughput; this satisfies the additional requirement that the ratio of throughputs of sessions interfering in a (saturated) bottleneck link is equal to the ratio of their throughput priorities. Jaffe [21, Section X] uses these concepts to provide a reformulation of max-min fairness. (The original formulation of max-min fairness in [21, Section IX] is somehow different from the version of max-min fairness in [2, Section 5], although it can be easily shown that one implies the other.)

To the best of our understanding, Jaffe's formulation is comparable to the special case in our general framework where rates are linearly related to priorities. Since, however, we allow priority functions to be *arbitrary,* not necessarily linear, functions relating priorities and rates, our framework provides more flexibility in modeling a heterogeneous set of users with arbitrarily specified priorities. Jaffe shows that a simple modification of some *specific,* conservative and iterative algorithm that he provides for the case of a homogeneous set of sessions [21, Section V] still satisfies his reformulation of max-min fairness. In contrast, our work is far more general in that we study and prove results for *any* priority bottleneck algorithm (Sections 5 and 6), whether conservative or optimistic. In addition, we also show results for *any* optimistic, priority bottleneck algorithm, which we use to derive specific such algorithms.

In a different avenue, Gafni and Bertsekas [17] formulate elegant, combinatorial optimization conditions, involving abstract functions of both rates and capacities, which are intended to model the effect of preferences among different sessions on allocating the capacity of each individual link. Gafni and Bertsekas provide a *single* iterative algorithm, which is conservative and works in synchronous rounds, for calculating rates that satisfy these conditions. In contrast, we explicitly introduce priorities as functions of rates (and vice versa) in a more direct and intuitive way; we formulate a fairness condition suitable for any general setting with priorities, and we develop a coherent theory around this condition. In addition, as we already pointed out, a major part of our work concerns the entire class of priority bottleneck algorithms, and their further study and instantiation in the optimistic framework [1, 15, 16].

3 Model

Our definitions closely follow and adapt corresponding ones in [15, Section 3]. However, some increase in generality and abstraction over [15, Section 3] allows for some particular simplifications.

3.1 Network, Sessions and Rate Vectors

A *communication network* is a directed graph $G = (V, E)$. Each vertex $v \in V$ represents a switching node, and each edge $e \in E$ represents a point-to-point link in the network. Associated with each edge $e \in E$ is a finite *capacity* $cap(e) > 0$.

The network supports a fixed set of sessions with virtual circuit routing. Formally, a *session* is a sequence of edges, which is a simple path in G between a *source* and a *destination*; a session models a *virtual circuit* of the network. We consider a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n sessions laid

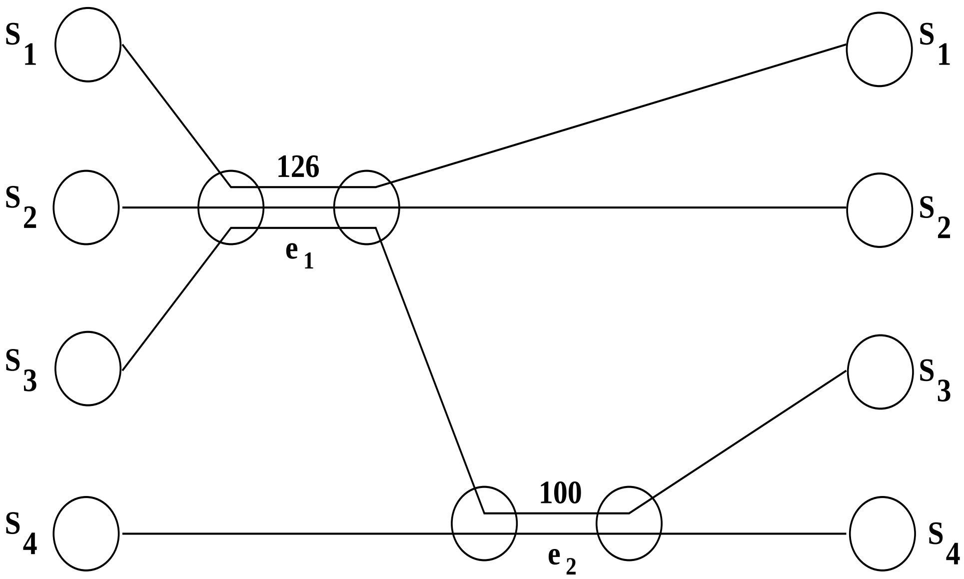

Figure 1: A simple network

out on G, where $n \geq 1$. For each edge $e \in E$, denote *sessions*(e) the set of sessions passing through e; for any set of sessions $\mathcal{S}' \subseteq \mathcal{S}$, denote $\mathcal{S}' \mid e = \mathcal{S}' \cap sessions(e)$. In our discussion, we will often abuse notation by identifying a session S_i with its index i. Even more so, 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 $e \in S_i$.

Example 3.1 Figure 1 depicts a simple network with four sessions S_1, S_2, S_3 and S_4, and two edges, e_1 and e_2 with $cap(e_1) = 126$ and $cap(e_2) = 100$, respectively. Notice that both sessions S_1 and S_2 traverse only edge e_1 and no other edge; session S_4 traverses only edge e_2, while session S_3 traverses both edges. □

Allocated to each session S is a *rate* $r(S)$, which is a non-negative real number. A *rate vector* is a vector $\mathbf{r} = \langle r(S_1), r(S_2), \ldots, r(S_n) \rangle^{(\mathrm{T})}$. The *capacity constraint* requires that for each edge $e \in E$,

$$\sum_{S \in sessions(e)} r(S) \quad \leq \quad cap(e)\,;$$

that is, the sum of rates of sessions sharing an edge does not exceed the edge capacity. A rate vector is *feasible* if it satisfies the capacity constraint. Say that an edge e is *saturated* if

$$\sum_{S \in sessions(e)} r(S) \quad = \quad cap(e)\,;$$

we will also say that $\mathbf{r}$ *saturates* e in this case.

A rate vector is *maximal* if it is feasible and it is not possible to increase the rate of any vector, while still maintaining feasibility, without decreasing the rate of some other session.

3.2 Active Sessions and Active Edges

The communication network is abstracted as a (possibly infinite) state machine. Each state Q of the network consists of two components: a *feasible rate vector* $\mathbf{r} \in \Re^n$, and an *active set* $\mathcal{A} \subseteq \mathcal{S}$, which is a set of *active sessions in state* Q; that is, $Q = \langle \mathbf{r}, \mathcal{A} \rangle$. We will sometimes use Q as an index and write $\mathbf{r}_Q$ and $\mathcal{A}_Q$, in order to declare the state of reference. Intuitively, an active session is one that has not yet reached its "final" rate. Denote $\mathcal{D}_Q = \mathcal{S} \setminus \mathcal{A}_Q$; that is, $\mathcal{D}_Q$ is the set of *done sessions in state* Q, which are sessions whose rates have been finalized and will not change any further (cf. Section 3.5). The *status of a session in state* Q is defined to be either active or done in the natural way.

In the *initial state* $Q^{in} = \langle \mathbf{r}^{in}, \mathcal{A}^{in} \rangle$, $\mathbf{r}^{in}$ is a vector whose entries are all equal to 0, and $\mathcal{A}^{in} = \mathcal{S}$; that is, all sessions are initially active and have zero rates. A state Q is *final* if $\mathcal{A}_Q = \emptyset$; that is, in a final state, all sessions have reached their final rates.

The set of *active edges of the network in state* Q, denoted AE_Q is defined to be

$$AE_Q = \bigcup_{S \in \mathcal{A}_Q} \{e \mid e \in S\} ;$$

that is, AE_Q contains all edges of the network traversed by at least one active session in state Q.

For any edge $e \in E$ and state Q, the *allotted capacity of* e *in state* Q [1, Section 2.1], denoted $allot_Q(e)$, is defined to be the total rate already allocated to done sessions passing through the edge; that is,

$$allot_Q(e) = \sum_{S \in \mathcal{D}_Q|e} r_Q(S) .$$

Clearly, the capacity constraint implies that

$$\sum_{i \in \mathcal{A}_Q|e} r_Q(S_i) \leq cap(e) - allot_Q(e) .$$

3.3 Operations

Roughly speaking, 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 `operation` that takes as input a session index i and a state Q and gives as output rates for all sessions j such that $S_i \cap S_j \neq \emptyset$ in a way that the resulting rate vector $\mathbf{r}' = \texttt{operation}(i, Q)$ preserves the capacity constraint.

Although our results are shown for *any* operation,[§] we are only interested in operations that use but "local information" with respect to session S_i (cf. [22, Section IV]). Such information may include the capacity of each link in session S_i and the current rates of all interfering sessions. We emphasize that we do not specify here the mechanics of when and how this information is made available and in what form the information arrives. Each link may know to dispatch information to all sessions sharing the link at regular intervals of time, or, instead, calculation and gathering of information may be prompted by a control message (also called *Resource Management* cell, or *RM* cell for short) looping around the path of the session on which an operation is applied (see [21, Section IV] or [1, Section 1] for related discussions). A complete specification of how information is gathered and subsequently disseminated to interfering sessions is usually part of the design and implementation of a distributed algorithm that entails successive applications of the operation in order to arrive at a final set of rates. (Such specification is beyond the scope of the present article.)

Say that `operation` is *conservative* if, going from $\mathbf{r}_Q$ to $\mathbf{r}'$, no entry decreases; else `operation` is *optimistic*.

3.4 Schedulers and Terminators

A *scheduler* [1, 15] is a function Sched that maps a triple of a pair $\langle G, \mathcal{S} \rangle$ of a network G and a set of session $\mathcal{S}$ laid out on G, a network state Q, and a state index $l \geq 1$[¶] to a session index $i = \mathsf{Sched}(\langle G, \mathcal{S} \rangle, Q, l)$. Intuitively, Sched determines the order of scheduling sessions for applying operations on them.

A *terminator* [1, 15] is a function Term that maps a pair $\langle G, \mathcal{S} \rangle$ of a network G and a set of sessions $\mathcal{S}$ laid out on G, and a network state Q to a set of sessions $\mathcal{T} = \mathsf{Term}(\langle G, \mathcal{S} \rangle, Q))$ such that $\mathcal{T} \subseteq \mathcal{A}_Q$. Intuitively, Term decides which sessions among those still active should be

[§] Our results for the optimistic case in Section 7 are an exception to this since they assume some specific operation.

[¶] We could have defined the state index as a component of state; in that case, we would also have to specify how state transitions modify the state index. For clarity of presentation, we chose not to do so.

"terminated".[‖] For sake of simplicity, we shall sometimes omit the argument $\langle G, \mathcal{S}\rangle$ of Term when such is clear from context. Whenever $S \in \mathsf{Term}(Q)$, we will say that S *becomes done in state* Q (cf. Section 3.5).

An algorithm Alg is a pair $\mathsf{Alg} = \langle \mathsf{Sched}, \mathsf{Term}\rangle$. The classification of operations into conservative and optimistic leads to a corresponding classification of algorithms in the natural way.

3.5 Executions

We model a computation "manipulating" the rates and status of sessions laid out on the network as a sequence of operations, each of which changes the rates of a scheduled session and of other sessions that "interfere" with it. Formally, an *execution of* Alg *on network* G *with session set* $\mathcal{S}$ is a (possibly infinite) sequence of alternating states and session indices $\alpha = Q_0, i_1, Q_1, \ldots, i_l, Q_l, \ldots$, satisfying the following conditions:

(1) $Q_0 = Q^{in}$ and $\mathcal{A}_{Q_1} = \mathcal{S}$;

(2) for each integer $l \geq 1$, $i_l = \mathsf{Sched}(\langle G, \mathcal{S}\rangle, Q_{l-1}, l-1)$;

(3) for each integer $l \geq 1$, if $i_l \in \mathcal{D}_{Q_l}$, then $\mathbf{r}_{Q_l} = \mathbf{r}_{Q_{l-1}}$ and $\mathcal{A}_{Q_{l+1}} = \mathcal{A}_{Q_l}$, else

  (a) $\mathbf{r}_{Q_l} := \mathtt{operation}(i_l, Q_{l-1})$;

  (b) $\mathcal{A}_{Q_{l+1}} = \mathcal{A}_{Q_l} \setminus \mathsf{Term}(\langle G, \mathcal{S}\rangle, Q_l)$.

We continue with an informal explanation of execution. Condition (1) asserts that the starting state in execution α is the initial state Q^{in}, and that no session termination takes place in the initial state. Condition (2) guarantees that the session indices in α are determined according to Sched. Condition (3) specifies how the rates and status of sessions change from a state to the next. If the scheduled session is done, then nothing happens; otherwise, the new rates are determined by `operation` (condition (3/a), while condition (3/b) provides for any possible termination of sessions in the subsequent state according to Term. For any integer $l \geq 1$, we will say that *session* i_l *is scheduled in front of state* Q_l *in execution* α.

We continue to state an immediate consequence of the definitions of allotted capacity and execution.

[‖] In all of our discussion, a "terminated" session is meant to be one whose rate is finalized.

Claim 3.1 *For any integer $l \geq 1$ and edge $e \in E$,*

$$allot_{Q_l}(e) \quad = \quad allot_{Q_{l-1}}(e) + \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_{l-1}}(S_i)\,.$$

We finally show a similar simple fact that will be helpful in our later proofs.

Claim 3.2 *For any integers l_0 and l, $0 \leq l_0 < l$, and for any edge $e \in E$,*

$$\sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_{l_0}}(S_i) \quad = \quad allot_{Q_l}(e) - allot_{Q_{l_0}}(e)\,.$$

Proof: Clearly,

$$\begin{aligned}
i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) \mid e \quad &\Leftrightarrow \quad i \in \mathcal{A}_{Q_{l_0}} \mid e \text{ and } i \notin \mathcal{A}_{Q_l} \mid e \\
&\Leftrightarrow \quad i \notin \overline{\mathcal{A}_{Q_{l_0}}} \mid e \text{ and } i \in \overline{\mathcal{A}_{Q_l}} \mid e \\
&\Leftrightarrow \quad i \notin \mathcal{D}_{Q_{l_0}} \mid e \text{ and } i \in \mathcal{D}_{Q_l} \mid e \\
&\Leftrightarrow \quad i \in (\mathcal{D}_{Q_l} \setminus \mathcal{D}_{Q_{l_0}}) \mid e\,.
\end{aligned}$$

Hence, it follows that

$$\begin{aligned}
\sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i) \quad &= \quad \sum_{i \in (\mathcal{D}_{Q_l} \setminus \mathcal{D}_{Q_{l_0}}) | e} r_{Q_l}(S_i) \\
&= \quad \sum_{i \in \mathcal{D}_{Q_l} | e} r_{Q_l}(S_i) - \sum_{i \in \mathcal{D}_{Q_{l_0}} | e} r_{Q_l}(S_i) \\
&\qquad (\text{since } \mathcal{D}_{Q_{l_0}} \subseteq \mathcal{D}_{Q_l}) \\
&= \quad allot_{Q_l}(e) - allot_{Q_{l_0}}(e) \\
&\qquad (\text{by definition of allotted capacity})\,,
\end{aligned}$$

as needed. ■

Call each state Q_l, $l \geq 0$, a *reachable state for execution α*. For any reachable states Q_l and $Q_{l'}$ in execution α, write $Q_l \xrightarrow{\alpha} Q_{l'}$ if $l < l'$; that is, $Q_l \xrightarrow{\alpha} Q_{l'}$ if Q_l precedes $Q_{l'}$ in execution α. For any reachable state Q in an infinite execution α, define $\overrightarrow{Q}$ to be the state immediately following Q in execution α. For any reachable state Q other than the initial state Q_0 in an infinite execution α, define $\overleftarrow{Q}$ to be the state immediately preceding Q in execution α. (This notation will sometimes allow us to drop using the index of state Q in α, while still being able to refer to its successor state.)

Since neither rates or status of sessions may change from any (reachable) state Q in execution α such that $\mathcal{D}_Q = \mathcal{S}$ on, whenever in the sequel we consider a final state Q in execution α, we will be implicitly assuming in our later proofs that Q is the *earliest* final state that is reachable in α.

3.6 Convergence Complexity

Intuitively, an operation is "executed" in execution α each time it holds, for any integer $l \geq 1$, that $\mathbf{r}_{Q_l} \neq \mathbf{r}_{Q_{l-1}}$;** that is, an operation is executed each time changes in rates of sessions occur.

We are naturally interested in the *shortest* possible prefix (if any) of an execution that ends with a final state. Thus, we define the *number of operations in execution* α to be the number of integers $l \geq 1$ such that both $\mathbf{r}_{Q_l} \neq \mathbf{r}_{Q_{l-1}}$ and none of $Q_0, \ldots, Q_{l-1}$ is a final state.

The *convergence complexity of* Alg *on network* G *with session set* $\mathcal{S}$ is defined to be the maximum, over all executions of Alg on network G with session set $\mathcal{S}$, of the number of update operations in any execution. The *convergence complexity of* Alg is defined to be the maximum, over all pairs of a network G and a session set $\mathcal{S}$ laid out on G, of the convergence complexity of Alg on network G with session set $\mathcal{S}$.

4 Priority Max-Min Fairness and Priority Share

In this section, we introduce priorities and priority functions; in turn, we use them to define and study priority max-min fairness and priority share.

4.1 Priorities and Priority Functions

Associated with each session S_i is a real number $\mathcal{P}(S_i)$, called *priority*; intuitively, $\mathcal{P}(S_i)$ represents the priority to bandwidth access offered to session S_i. For each session S_i, the *priority function* $\mathbf{F}_i : \Re^+ \rightarrow \Re^+$ is a continuous, strictly increasing, and convex function that maps the priority $\mathcal{P}(S_i)$ to a rate $r(S_i)$. Since $\mathbf{F}_i$ is strictly increasing, it is invertible; denote $\mathbf{F}_i^{-1}$ its inverse function, which is also strictly increasing. Clearly, a rate adjustment implies a corresponding priority adjustment and vice versa, by way of the priority function and its inverse. Thus, an execution may equivalently be viewed as "manipulating" the priorities and status of sessions laid out on the network.

** By definition of execution, this implies that S_{i_l} is active in state Q_{l-1}.

Clearly, rates are "real" network entities; in contrast, however, priorities are but abstract parameters that are invoked in our framework and serve only as vehicles for the determination of rates via the priority functions. Thus, prioritized rate allocation materializes by appropriately selecting the priority functions and using them to determine the rates.

4.2 Priority Max-Min Fair Rate Vector

Loosely speaking, a priority max-min fair rate vector assigns to each session the maximum possible rate, while respecting at the same time sessions with lower priorities; thus, a priority max-min fair rate vector generalizes the (classical) max-min fair rate vector (see [2, Section 6.4.2] for a textbook discussion) by taking priorities into consideration. Formally, a *priority max-min fair rate vector* is a maximal rate vector $\mathbf{r}$ such that for each session S_i, $r(S_i)$ cannot be increased, while still maintaining feasibility, without decreasing $r(S_j)$ for some session S_j such that $\mathcal{P}(S_j) \leq \mathcal{P}(S_i)$.

Priority max-min fairness is easily seen to be a generalization of (classical) max-min fairness [20, 21, 22, 29]. Indeed, a *max-min fair rate vector* [20, 21, 22, 29] is a maximal rate vector $\mathbf{r}$ such that for each session S_i, $r(S_i)$ cannot be increased, while still maintaining feasibility, without decreasing $r(S_j)$ for some session S_j such that $r(S_j) \leq r(S_i)$. Hence, a max-min fair rate vector can be cast in an intuitive way as a priority max-min fair rate vector for a set $\mathcal{S}$ of sessions such that for each session $S_i \in \mathcal{S}$, $\mathbf{F}_i(\mathcal{P}(S_i)) = \mathcal{P}(\mathcal{S}_i)$; however, since $\mathbf{F}_i(\mathcal{P}(S_i)) = r(S_i)$ (by definition of priority functions), it follows that this is the special case where the rate and priority of each session coincide.

Example 4.1 Consider the network depicted in Figure 2. Assume that $r(S_1) = \mathbf{F}_1(\mathcal{P}(S_1)) = \mathcal{P}(S_1)$, $r(S_2) = \mathbf{F}_2(\mathcal{P}(S_2)) = \mathcal{P}(S_2)/2$ and $r(S_3) = \mathbf{F}_3(\mathcal{P}(S_3)) = 3\,\mathcal{P}(S_3)$.[††] Consider the rate vector $\langle 28, 14, 84\rangle^{(\mathrm{T})}$, which saturates the edge e. Thus, no session rate can be increased without decreasing the rate of some other session; since the inverses of the priority functions are strictly increasing functions, this implies a corresponding decrease to the priority of the session whose rate is decreased. However, simple calculations using the priority functions reveal that all priorities are equal to 28. Hence, any such increase must necessarily cause decreasing an equal priority. It follows that $\langle 28, 14, 84\rangle^{(\mathrm{T})}$ is a priority max-min fair rate vector. We encourage the reader to verify that the classical max-min fair rate vector is the *different* vector $\langle 42, 42, 42\rangle^{(\mathrm{T})}$. Moreover, this simple example confirms that the more rapidly increasing the priority function of a session is, the larger its rate will be. □

[††] We remark that it is only for the sake of simplicity and ease in calculations that priority functions have been chosen to be linear in this and later examples.

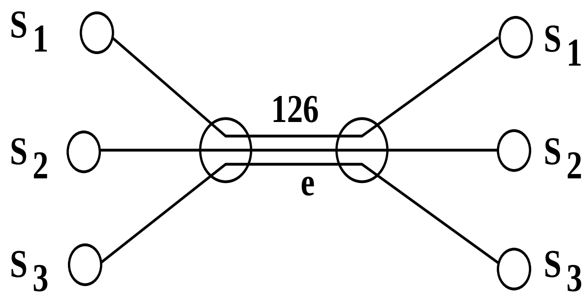

Figure 2: A network consisting of a single edge

4.3 The Adequacy Assumption

Assume now that for any session S_i, $\mathbf{F}_i(0) > 0$; that is, even a session with zero priority receives some non-zero rate. Since $\mathbf{F}_i$ is a strictly increasing function, this implies that $r(S_i) = \mathbf{F}_i(\mathcal{P}(S_i)) \geq \mathbf{F}_i(0) > 0$ in any state; that is, a non-zero lower bound $\mathbf{F}_i(0)$, called *rate demand,* is imposed on the rate of session S_i.

An *adequate edge* is an edge e such that

$$\sum_{i \in sessions(e)} \mathbf{F}_i(0) \quad \leq \quad cap(e) \, ;$$

that is, the capacity of edge e suffices to accomodate rate demands of all sessions traversing e. Naturally, the *adequacy assumption* insists that every edge $e \in E$ is adequate.

4.4 Priority Max-Min Fairness Equation

Take any arbitrary edge e active in state Q. By the capacity constraint,

$$\sum_{i \in \mathcal{A}_Q | e} r_Q(S_i) \quad \leq \quad cap(e) - allot_Q(e) \, ,$$

or, by definition of priority functions,

$$\sum_{i \in \mathcal{A}_Q | e} \mathbf{F}_i(\mathbf{F}_i^{-1}(r_Q(S_i))) \quad \leq \quad cap(e) - allot_Q(e) \, .$$

Insisting that for each $i \in \mathcal{A}_Q \mid e$, $\mathbf{F}_i^{-1}(r_Q(S_i)) = p$, for some parameter $p \geq 0$, and also that the edge e is saturated in state Q, yields the *priority max-min fairness equation for edge* *e in state Q*, namely that

$$\sum_{i \in \mathcal{A}_Q | e} \mathbf{F}_i(p) \quad = \quad cap(e) - allot_Q(e)\,.$$

We remark that the priority max-min fairness equation for edge e in state Q is an algebraic equation in the (unspecified) parameter p.

4.5 Priority Share

4.5.1 Definition

Define the *priority share of edge e in state Q*, denoted $PS_Q(e)$, to be a positive root p of the priority max-min fairness equation. Thus,

$$\sum_{i \in \mathcal{A}_Q | e} \mathbf{F}_i(PS_Q(e)) \quad = \quad cap(e) - allot_Q(e)\,.$$

Example 4.2 Consider again the simple network depicted in Figure 2, and retain all assumptions on priority functions for this network that were made in Example 4.1. Consider state Q_0 where $allot_{Q_0} = 0$ and all three sessions are active. The priority max-min fairness equation for state Q_0 is $\sum_{i \in \mathcal{A}_{Q_0} | e} \mathbf{F}_i(p) = cap(e) - allot_{Q_0}(e)$, or $p + p/2 + 3p = 126$, which implies that $p = 28$. Thus, by definition of priority share, $PS_{Q_0}(e) = 28$.

We remark that the priority share generalizes the notion of the *fair share* [1, Section 2.1] to the setting with priorities. Indeed, the point of departure of both is the *saturation equation for edge e in state Q*, namely that

$$\sum_{i \in \mathcal{A}_Q | e} r_Q(S_i) \quad = \quad cap(e) - allot_Q(e)\,.$$

Insisting that for each session $i \in \mathcal{A}_Q \mid e$, $r_Q(S_i) = r$, for any parameter $r > 0$, allows the saturation equation to admit (trivially) the priority share

$$FS_Q(e) \quad = \quad \frac{cap(e) - allot_Q(e)}{|\mathcal{A}_Q \mid e|}$$

as its *unique* root in this case. In a similar manner, the priority share is a root of the saturation equation after it is expressed, via the priority functions, in terms of some priority p that is common for all active sessions traversing the edge. We establish that, in fact, the priority share exists uniquely as well.

4.5.2 Uniqueness

We show:

Theorem 4.1 *For any edge e and state Q in execution α, $PS_Q(e)$ exists uniquely.*

Proof: Since each individual priority function $\mathbf{F}_i$, $i \in \mathcal{A}_Q \mid e$, is continuous, increasing and convex, it follows that the sum $\sum_{i\in\mathcal{A}_Q|e} \mathbf{F}_i(p)$ is a continuous, increasing, and convex function of p. The value at $p = 0$ of this function is $\sum_{i\in\mathcal{A}_Q|e} \mathbf{F}_i(0)$. By convexity, this function increases to infinity. By continuity and increasing monotonicity, it follows that this function attains any single value no less than $\sum_{i\in\mathcal{A}_Q|e} \mathbf{F}_i(0)$ exactly once. In particular, the function (in p) $\sum_{i\in\mathcal{A}_Q|e} \mathbf{F}_i(p)$ attains (exactly once) the value $cap(e) - allot_Q(e)$ if and only if

$$cap(e) - allot_Q(e) \quad \geq \quad \sum_{i\in\mathcal{A}_Q|e} \mathbf{F}_i(0)\,.$$

Hence, by definition of priority share, it suffices to show that the last inequality holds.

There are two cases to consider. Assume first that $Q = Q_0$; then,

$$\sum_{i\in\mathcal{A}_Q|e} \mathbf{F}_i(0) \quad = \quad \sum_{i\in sessions(e)} \mathbf{F}_i(0)$$

and $cap(e) - allot_Q(e) = cap(e)$, so that the inequality to show is but the adequacy assumption. Assume now that $Q = Q_l$, for some $l > 0$. Clearly,

$$\begin{aligned}
cap(e) - allot_{Q_l}(e) \quad &= \quad cap(e) - allot_{Q_{l-1}}(e) - \sum_{i\in(\mathcal{A}_{Q_{l-1}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(S_i) \\
&\qquad \text{(by Claim 3.1)} \\
&= \quad cap(e) - allot_{Q_{l-1}}(e) - \Big(\sum_{i\in\mathcal{A}_{Q_{l-1}}|e} r_{Q_{l-1}}(S_i) - \sum_{i\in\mathcal{A}_{Q_l}|e} r_{Q_{l-1}}(S_i)\Big) \\
&= \quad cap(e) - allot_{Q_{l-1}}(e) - \sum_{i\in\mathcal{A}_{Q_{l-1}}|e} r_{Q_{l-1}}(S_i) + \sum_{i\in\mathcal{A}_{Q_l}|e} \mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) \\
&\qquad \text{(by definition of priority functions)} \\
&\geq \quad cap(e) - allot_{Q_{l-1}}(e) - (cap(e) - allot_{Q_{l-1}}(e)) + \sum_{i\in\mathcal{A}_{Q_l}|e} \mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) \\
&\qquad \text{(by the capacity constraint)} \\
&= \quad \sum_{i\in\mathcal{A}_{Q_l}|e} \mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) \\
&\geq \quad \sum_{i\in\mathcal{A}_{Q_l}|e} \mathbf{F}_i(0) \\
&\qquad \text{(since } \mathbf{F}_i \text{ is strictly increasing)}\,,
\end{aligned}$$

as needed. ∎

4.5.3 Further Properties

We continue to establish further properties of the priority share. These properties refer to an execution $\alpha = Q_0, i_1, Q_1, \ldots, i_l, Q_l, \ldots$ of *any* algorithm; we shall omit reference to α when such is clear from context. The first property describes the evolution of priority shares in the execution.

Lemma 4.2 *For any integer $l \geq 1$, and for any edge $e \in E$ active in state Q_l,*

$$\sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_l}(e)) = \sum_{i \in \mathcal{A}_{Q_{l-1}}|e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(S_i) .$$

Proof: By definition of $PS_{Q_l}(e)$,

$$\begin{aligned} \sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_l}(e)) &= cap(e) - allot_{Q_l}(e) \\ &= cap(e) - allot_{Q_{l-1}}(e) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(S_i) . \end{aligned}$$

By definition of $PS_{Q_{l-1}}(e)$,

$$\sum_{i \in \mathcal{A}_{Q_{l-1}}|e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) = cap(e) - allot_{Q_{l-1}}(e) ,$$

so that

$$\sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_l}(e)) = \sum_{i \in \mathcal{A}_{Q_{l-1}}|e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(S_i) ,$$

as needed. ∎

We continue to prove that the saturation of an edge depends in a critical way on how priorities and priority shares of sessions traversing the edge compare to each other. The proof uses the definition of priority share and the fact that both priority functions and their inverses are strictly increasing.

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:*

(1) *if for each* $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $\mathcal{P}_{Q_l}(S_i) = PS_{Q_{l_0}}(e)$, *then* e *is saturated in* Q_l;

(2) *if for each* $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $\mathcal{P}_{Q_l}(S_i) < PS_{Q_{l_0}}(e)$, *then* e *is not saturated in* Q_l;

(3) *there exists no index* $j \in \mathcal{A}_{Q_{l_0}} \mid e$ *such that* $\mathcal{P}_{Q_l}(S_j) > PS_{Q_{l_0}}(e)$, *while for each* $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $i \neq j$, $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$.

Proof: We start by proving (1). By definition of $PS_{Q_{l_0}}(e)$,

$$\sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) \quad = \quad cap(e) - allot_{Q_{l_0}}(e)\,.$$

By assumption, for each $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $\mathcal{P}_{Q_l}(S_i) = PS_{Q_{l_0}}(e)$; hence, it follows that

$$\sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) \quad = \quad cap(e) - allot_{Q_{l_0}}(e)\,.$$

By definition of priority functions, this implies that

$$\sum_{i \in \mathcal{A}_{Q_{l_0}} | e} r_{Q_l}(S_i) \quad = \quad cap(e) - allot_{Q_{l_0}}(e)\,.$$

Hence,

$$\begin{aligned}
\sum_{i \in \mathcal{A}_{Q_l} | e} r_{Q_l}(S_i) \quad &= \quad \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} r_{Q_l}(S_i) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i) \\
&\quad\quad (\text{since } \mathcal{A}_{Q_l} \mid e \subseteq \mathcal{A}_{Q_{l_0}} \mid e) \\
&= \quad cap(e) - allot_{Q_{l_0}}(e) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i)\,. \\
&= \quad cap(e) - allot_{Q_{l_0}}(e) - (allot_{Q_l}(e) - allot_{Q_{l_0}}(e)) \\
&\quad\quad (\text{by Claim 3.2}) \\
&= \quad cap(e) - allot_{Q_l}(e)\,,
\end{aligned}$$

as needed to establish that e is saturated in state Q_l.

We continue to prove (2). Clearly,

$$\begin{aligned}
\sum_{i \in \mathcal{A}_{Q_l} | e} r_{Q_l}(S_i) \quad &= \quad \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} r_{Q_l}(S_i) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i) \\
&\quad\quad (\text{since } \mathcal{A}_{Q_l} \mid e \subseteq \mathcal{A}_{Q_{l_0}} \mid e) \\
&= \quad \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i)
\end{aligned}$$

$$
\begin{aligned}
& \text{(by definition of priority functions)} \\
= \; & \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - (allot_{Q_l}(e) - allot_{Q_{l_0}}(e)) \\
& \text{(by Claim 3.2)} \\
< \; & \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) \\
& \text{(since } \mathbf{F}_i \text{ is strictly increasing and } \mathcal{P}_{Q_l}(S_i) < PS_{Q_{l_0}}(e)) \\
= \; & cap(e) - allot_{Q_{l_0}}(e) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) \\
& \text{(by definition of priority share)} \\
= \; & cap(e) - allot_{Q_l}(e) \,,
\end{aligned}
$$

as needed to establish that e is *not* saturated in state Q_l.

We finally prove (3). Assume, by way of contradiction, that there exists some session $i_0 \in \mathcal{A}_{Q_{l_0}} \mid e$ such that $\mathcal{P}_{Q_l}(S_{i_0}) > PS_{Q_{l_0}}(e)$, while for each $i \in \mathcal{A}_{Q_{l_0}} \mid e$, $i \neq i_0$, $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$. Clearly,

$$
\begin{aligned}
\sum_{i \in \mathcal{A}_{Q_l} | e} r_{Q_l}(S_i) \; = \; & \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} r_{Q_l}(S_i) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i) \\
& \text{(since } \mathcal{A}_{Q_l} \mid e \subseteq \mathcal{A}_{Q_{l_0}} \mid e) \\
= \; & \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i) \\
& \text{(by definition of priority functions)} \\
= \; & \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - (allot_{Q_l}(e) - allot_{Q_{l_0}}(e)) \\
& \text{(by Claim 3.2)} \\
= \; & \mathbf{F}_{i_0}(\mathcal{P}_{Q_l}(S_{i_0})) + \sum_{i \in \mathcal{A}_{Q_{l_0}} | e, i \neq i_0} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) \\
> \; & \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e)) + \sum_{i \in \mathcal{A}_{Q_{l_0}} | e, i \neq i_0} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) \\
& \text{(since } \mathbf{F}_j \text{ is strictly increasing and } \mathcal{P}_{Q_l}(S_j) > PS_{Q_{l_0}}(e)) \\
\geq \; & \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e)) + \sum_{i \in \mathcal{A}_{Q_{l_0}} | e, i \neq i_0} \mathbf{F}_i(PS_{Q_{l_0}}(e)) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e) \\
& \text{(since } \mathbf{F}_i \text{ is strictly increasing and } \mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e) \text{ for } i \neq i_0) \\
= \; & \sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e)) \\
= \; & cap(e) - allot_{Q_{l_0}}(e) - allot_{Q_l}(e) + allot_{Q_{l_0}}(e)
\end{aligned}
$$

(by definition of priority share)

$$= \quad cap(e) - allot_{Q_l}(e) \,,$$

which establishes that e violates the capacity constraint in state Q_l. A contradiction. This completes the proof of (3). ■

4.5.4 Minimum Priority Share and Minimum Priority Share Edges

For each session $i \in [n]$, the *minimum priority share of session S_i in state Q,* denoted $MPS_Q(S_i)$, is defined to be

$$MPS_Q(S_i) \quad = \quad \min_{e \in S_i} PS_Q(e) \,;$$

so, $MPS_Q(S_i)$ is the least among all priority shares attained in state Q by edges traversed by S_i. The minimum priority share is a generalization of the *minimum fair share* [1, Definition 3.6] to the setting with priorities.

For each state Q, a *minimum priority share edge for Q* is an edge e active in Q such that

$$PS_Q(e) \quad = \quad \min_{e' \in AE_Q} PS_Q(e') \,;$$

that is, a minimum priority share edge for Q attains the least priority share among all edges active in state Q. A minimum priority share edge is a generalization of a *minimum fair share edge* defined in [15, Section 3.4] to the setting with priorities.

Define $MPSE_Q$, to be the *set of minimum priority share edges for Q.*

5 Priority Bottleneck Algorithms

In this section, we use priority share to define and study priority bottleneck edges and priority bottleneck algorithms. We discover that priority bottleneck algorithms allow priority shares. priority bottleneck edges, and minimum priority share edges to enjoy some additional properties.

5.1 Priority Bottleneck Edges and Priority Bottleneck Algorithms

Priority Bottleneck Edges

For any state Q, a *priority bottleneck edge for Q* is an edge $e \in E$ such that for each session $S \in \mathcal{A}_Q \mid e$,

$$MPS_Q(S) \quad = \quad PS_Q(e) \,;$$

that is, for each active session traversing e, e is the one, among all edges traversed by the session, that attains the minimum priority share of the session in state Q. We remark that a priority bottleneck edge generalizes a *bottleneck edge* [21] to the setting with priorities.

Example 5.1 Consider again the simple network depicted in Figure 1. Assume that $r(S_1) = \mathbf{F}_1(\mathcal{P}(S_1)) = \mathcal{P}(S_1)$, $r(S_2) = \mathbf{F}_2(\mathcal{P}(S_2)) = \mathcal{P}(S_2)/2$, $r(S_3) = \mathbf{F}_3(\mathcal{P}(S_3)) = 3\,\mathcal{P}(S_3)$, and $r(S_4) = \mathbf{F}_4(\mathcal{P}(S_4)) = \mathcal{P}(S_4)/3$. Consider a state Q such that $allot_Q(e_1) = allot_Q(e_2) = 0$, while all sessions are active in state Q.

The priority max-min fairness equation for edge e_1 in state Q is $\sum_{i \in \mathcal{A}_Q | e_1} \mathbf{F}_i(p) = cap(e_1) - allot_Q(e_1)$, or $p + p/2 + 3p = 126 - 0$, which implies that $p = 28$. By definition of priority share, it follows that $PS_Q(e_1) = 28$.

Similarly, the priority max-min fairness equation for edge e_2 in state Q is $\sum_{i \in \mathcal{A}_Q | e_2} \mathbf{F}_i(p) = cap(e_2) - allot_Q(e_2)$, or $3p + p/3 = 100 - 0$, which implies that $p = 30$. By definition of priority share, it follows that $PS_Q(e_2) = 30$.

Since session S_3 traverses both edges e_1 and e_2, the minimum priority share of session S_2 in state Q is $MPS_Q(S_3) = \min\{PS_Q(e_1), PS_Q(e_2)\} = \min\{28, 30\} = 28$. Since both sessions S_1 and S_2 traverse only edge e_1, their minimum priority shares in state Q are $MPS_Q(S_1) = MPS_Q(S_2) = PS_Q(e_1) = 28$. It follows that edge e_1 is a priority bottleneck edge for state Q. On the contrary, edge e_2 is *not* a priority bottleneck edge for state Q because $PS_Q(e_2) = 30$, while e_2 is traversed by session S_3 for which $MPS_Q(S_3) = 28$. $\square$

We continue to show that the priority shares of any two priority bottleneck edges shared by any single session are equal.

Lemma 5.1 *Let e and e' be priority bottleneck edges for state Q such that $(\mathcal{A}_Q \mid e) \cap (\mathcal{A}_Q \mid e') \neq \emptyset$. Then, $PS_Q(e) = PS_Q(e')$.*

Proof: Take any session $S \in (\mathcal{A}_Q \mid e) \cap (\mathcal{A}_Q \mid e')$. Since e and e' are priority bottleneck edges for state Q, both $MPS_Q(S) = PS_Q(e)$ and $MPS_Q(S) = PS_Q(e')$ hold. It follows that $PS_Q(e) = PS_Q(e')$, as needed. $\blacksquare$

The next simple claim is a direct consequence of the definition of a priority bottleneck edge and a minimum priority share edge.

Lemma 5.2 *For any state Q, consider any minimum priority share edge e for Q. Then, e is a bottleneck edge for Q.*

Proof: Take any session $S \in \mathcal{A}_Q \mid e$. Since $e \in MPSE_Q$, for any edge $e' \in AE_Q$, $PS_Q(e) \leq PS_Q(e')$. Thus, in particular, for any edge e' traversed by S, $PS_Q(e) \leq PS_Q(e')$, so that $PS_Q(e) \leq \min_{e' \in S} PS_Q(e')$. Since S traverses e, $PS_Q(e) \geq \min_{e' \in S} PS_Q(e')$. It follows that $PS_Q(e) = \min_{e' \in S} PS_Q(e')$. Since S was chosen arbitrarily, this implies that e is a bottleneck edge for Q, as needed. ■

Priority Bottleneck Algorithms

Say that terminator Term is *priority bottleneck* if for any state Q and session S, $S \in \mathsf{Term}(Q)$ if (and only if) there exists some edge $e \in E$, where S traverses e, such that:

(1) e is a priority bottleneck edge for Q;

(2) $\mathcal{P}_Q(S) = PS_Q(e)$.

Say that an algorithm $\mathsf{Alg} = \langle \mathsf{Sched}, \mathsf{Term} \rangle$ is *priority bottleneck* if Term is. A priority bottleneck algorithm generalizes a *bottleneck algorithm* [20, 21, 22] to the setting with priorities.

5.2 Increasing Monotonicity of Priority Share

We show that the priority share of an edge may not decrease, as long as there are still active sessions traversing the edge, while a priority bottleneck algorithm is "running".

Proposition 5.3 *Assume that* Alg *is a priority bottleneck algorithm. Then, for any integer* $l \geq 1$ *and for any edge* $e \in AE_{Q_l}$,

$$PS_{Q_l}(e) \quad \geq \quad PS_{Q_{l-1}}(e)\,.$$

Proof: Assume, by way of contradiction, that $PS_{Q_l}(e) < PS_{Q_{l-1}}(e)$. Since for each session $i \in [n]$, $\mathbf{F}_i$ is a strictly increasing function, this implies that for each $i \in [n]$, $\mathbf{F}_i(PS_{Q_l}(e)) < \mathbf{F}_i(PS_{Q_{l-1}}(e))$, so that

$$\sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_l}(e)) \quad < \quad \sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_{l-1}}(e))\,.$$

Now, by Lemma 4.2,

$$\sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_l}(e)) \quad = \quad \sum_{i \in \mathcal{A}_{Q_{l-1}}|e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(S_i)\,.$$

Since Alg is priority bottleneck, there exists, for each index $i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) \mid e$, a priority bottleneck edge $e' \in S_i$ such that $\mathcal{P}_{Q_{l-1}}(S_i) = PS_{Q_{l-1}}(e')$. Since e' is a priority bottleneck edge, and $e \in S_i$, it follows by definition of priority bottleneck edge that $PS_{Q_{l-1}}(e') \leq PS_{Q_{l-1}}(e)$. Hence, $\mathcal{P}_{Q_{l-1}}(S_i) \leq PS_{Q_{l-1}}(e)$. Since $\mathbf{F}_i$ is a strictly increasing function, this implies that $\mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) \leq \mathbf{F}_i(PS_{Q_{l-1}}(e))$, or, by definition of priority functions $r_{Q_{l-1}}(S_i) \leq \mathbf{F}_i(PS_{Q_{l-1}}(e))$. It follows that

$$\begin{aligned}\sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_l}(e)) \;&\geq\; \sum_{i \in \mathcal{A}_{Q_{l-1}} | e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) | e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) \\ &= \sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_{l-1}}(e)),\end{aligned}$$

a contradiction. ■

We remark that Proposition 5.3 generalizes [1, Lemma 3.4], which establishes increasing monotonicity of fair shares [1, Section 2.1], to the setting with priorities.

5.3 Stability Properties of Priority Bottleneck Edges

We prove natural *stability* properties for any edge that becomes priority bottleneck in the course of an execution of a priority bottleneck algorithm. Formally, we show:

Proposition 5.4 *Assume that* Alg *is a priority bottleneck algorithm. For any integer* $l_0 \geq 0$*, fix any edge* $e \in E$ *that is a priority bottleneck edge for* Q_{l_0}*. Then, for any integer* $l \geq l_0$ *such that* $\mathcal{A}_{Q_l} \mid e \neq \emptyset$*, the following hold:*

(1) $PS_{Q_l}(e) = PS_{Q_{l_0}}(e)$;

(2) *e is a priority bottleneck edge for Q_l;*

(3) *for any session* $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e$, $\mathcal{P}_{Q_l}(S_i) = PS_{Q_{l_0}}(e)$.

Roughly speaking, Proposition 5.4 establishes that no change in the priority share of a priority bottleneck edge may occur as long as it remains active, so that the edge remains priority bottleneck; moreover, the final priority of any active session traversing the edge is equal to this constant priority share. The proof follows.

Proof: By induction on l. For the basis case where $l = l_0$, (1) holds trivially, (2) holds by assumption, and (3) holds by definition of a priority bottleneck algorithm. Assume inductively that for some integer $l > l_0$, the following hold for any integer l' such that $l_0 < l' \leq l$:

(1) $PS_{Q_{l'-1}}(e) = PS_{Q_{l_0}}(e)$;

(2) e is a priority bottleneck edge for $Q_{l'-1}$;

(3) for any session $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l'}}) \mid e$, $\mathcal{P}_{Q_{l'-1}}(S_i) = PS_{Q_{l_0}}(e)$.

For the induction step, we show that:

(1) $PS_{Q_l}(e) = PS_{Q_{l_0}}(e)$;

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

(3) for any session $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e$, $\mathcal{P}_{Q_l}(S_i) = PS_{Q_{l_0}}(e)$.

We start by proving (1). This will follow immediately from a technical claim we first show.

Lemma 5.5 *For each session $i \in \mathcal{A}_{Q_l} \mid e$,*

$$\mathbf{F}_i(PS_{Q_l}(e)) \quad = \quad \mathbf{F}_i(PS_{Q_{l_0}}(e))\,.$$

Proof: Assume, by way of contradiction, that there exists some session $i_0 \in \mathcal{A}_{Q_l} \mid e$ such that

$$\mathbf{F}_{i_0}(PS_{Q_l}(e)) \quad \neq \quad \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e))\,.$$

Since $e \in AE_{Q_l}$, Proposition 5.3 implies that $PS_{Q_l}(e) \geq PS_{Q_{l_0}}(e)$. Since $\mathbf{F}_{i_0}$ is a strictly increasing function, this implies that $\mathbf{F}_{i_0}(PS_{Q_l}(e)) \geq \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e))$. Since $\mathbf{F}_{i_0}(PS_{Q_l}(e)) \neq \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e))$, it follows that $\mathbf{F}_{i_0}(PS_{Q_l}(e)) > \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e))$.

Clearly,

$$\begin{aligned}
\sum_{i \in \mathcal{A}_{Q_l} \mid e} \mathbf{F}_i(PS_{Q_l}(e)) &= \mathbf{F}_{i_0}(PS_{Q_l}(e)) + \sum_{\substack{i \in \mathcal{A}_{Q_l} \mid e \\ i \neq i_0}} \mathbf{F}_i(PS_{Q_l}(e)) \\
&\geq \mathbf{F}_{i_0}(PS_{Q_l}(e)) + \sum_{\substack{i \in \mathcal{A}_{Q_l} \mid e \\ i \neq i_0}} \mathbf{F}_i(PS_{Q_{l_0}}(e)) \\
&\quad \text{(by Proposition 5.3 and since } \mathbf{F}_i \text{ is strictly increasing)} \\
&> \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e)) + \sum_{\substack{i \in \mathcal{A}_{Q_l} \mid e \\ i \neq i_0}} \mathbf{F}_i(PS_{Q_{l_0}}(e)) \\
&\quad \text{(since } \mathbf{F}_{i_0}(PS_{Q_l}(e)) > \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e))\text{)} \\
&= \sum_{i \in \mathcal{A}_{Q_l} \mid e} \mathbf{F}_i(PS_{Q_{l_0}}(e))\,,
\end{aligned}$$

or

$$\sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_l}(e)) \quad > \quad \sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_{l_0}}(e))\,.$$

Now, by Proposition 4.2,

$$\sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_l}(e)) \quad = \quad \sum_{i \in \mathcal{A}_{Q_{l-1}} | e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_{l-1}}(S_i)\,.$$

Consider any session $i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) \mid e$. Since $\mathcal{A}_{Q_{l-1}} \subseteq \mathcal{A}_{Q_{l_0}}$, this implies that $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) \mid e$. Thus, by induction hypothesis (condition (3)), $\mathcal{P}_{Q_{l-1}}(S_i) = PS_{Q_{l_0}}(e)$, so that $\mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) = \mathbf{F}_i(PS_{Q_{l_0}}(e))$. Hence, by definition of priority function, $r_{Q_{l-1}}(S_i) = \mathbf{F}_i(PS_{Q_{l_0}}(e))$. It follows that

$$\begin{aligned}
\sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_l}(e)) \quad &= \quad \sum_{i \in \mathcal{A}_{Q_{l-1}} | e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) | e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) \\
&= \quad \sum_{i \in \mathcal{A}_{Q_{l-1}} | e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) | e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) \\
&\qquad \text{(by induction hypothesis (condition (1)))} \\
&= \quad \sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_{l_0}}(e))\,,
\end{aligned}$$

a contradiction. ■

Take any session $i \in \mathcal{A}_{Q_l} \mid e$. Since $\mathbf{F}_i^{-1}$ is strictly increasing, it is a bijection, so that Lemma 5.5 implies that $PS_{Q_l}(e) = PS_{Q_{l_0}}(e)$, as needed. The proof of (1) is now complete.

We continue to show (2). Take any session index $i \in \mathcal{A}_{Q_l} \mid e$. Clearly, $i \in \mathcal{A}_{Q_{l_0}} \mid e$. Since e is a priority bottleneck edge for Q_{l_0}, it follows that

$$PS_{Q_{l_0}}(e) \quad = \quad \min_{e' \in S_i} PS_{Q_{l_0}}(e')\,.$$

Consider now any edge $e' \in S_i$. Since $S_i \in \mathcal{A}_{Q_l} \mid e$, it follows that e' is active in state Q_l. Thus, by Proposition 5.3, $PS_{Q_l}(e') \geq PS_{Q_{l_0}}(e')$. Since e' was chosen arbitrarily, this implies that

$$\min_{e' \in S_i} PS_{Q_l}(e') \quad \geq \quad \min_{e' \in S_i} PS_{Q_{l_0}}(e')\,.$$

It follows that

$$\min_{e' \in S_i} PS_{Q_l}(e') \quad \geq \quad PS_{Q_{l_0}}(e)\,.$$

By (1), this implies that

$$\min_{e' \in S_i} PS_{Q_l}(e') \quad \geq \quad PS_{Q_l}(e) \, .$$

Clearly, since $e \in S_i$,

$$\min_{e' \in S_i} PS_{Q_l}(e') \quad \leq \quad PS_{Q_l}(e) \, .$$

It follows that

$$PS_{Q_l}(e) \quad = \quad \min_{e' \in S_i} PS_{Q_l}(e') \, .$$

Since i was chosen arbitrarily, this implies that e is a priority bottleneck edge for Q_l. The proof of (2) is now complete.

We finally show (3). Take any session index $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e$. Since $i \notin \mathcal{A}_{Q_{l+1}}$, there exists some integer l', $l_0 < l' \leq l$, such that i becomes done in state $Q_{l'}$. Since Term is priority bottleneck, there exists some edge $e' \in S_i$ such that *(i)* e' is a priority bottleneck edge for $Q_{l'}$, and *(ii)* $\mathcal{P}_{Q_{l'}}(S_i) = PS_{Q_{l'}}(e')$.

Either $l_0 < l' < l$ or $l' = l$; thus, it follows either by induction hypothesis (condition (2)) or by (2) that e is a priority bottleneck edge for $Q_{l'}$ in either case. Since both e and e' are priority bottleneck edges for state $Q_{l'}$ and $S_i \in (\mathcal{A}_{Q_{l'}} \mid e) \cap (\mathcal{A}_{Q_{l'}} \mid e')$, Lemma 5.1 implies that $PS_{Q_{l'}}(e) = PS_{Q_{l'}}(e')$. Since $\mathcal{P}_{Q_{l'}}(S_i) = PS_{Q_{l'}}(e')$, this implies that

$$\mathcal{P}_{Q_{l'}}(S_i) \quad = \quad PS_{Q_{l'}}(e) \, .$$

Since $i \in \mathsf{Term}(Q_{l'})$ and $l \geq l'$,

$$\mathcal{P}_{Q_{l'}}(S_i) \quad = \quad \mathcal{P}_{Q_l}(S_i) \, .$$

Either $l_0 < l' < l$ or $l' = l$; thus, it follows either by induction hypothesis (condition (1)) or by (1) that

$$PS_{Q_{l'}}(e) \quad = \quad PS_{Q_{l_0}}(e) \, .$$

Hence, it follows that

$$\mathcal{P}_{Q_l}(S_i) \quad = \quad PS_{Q_{l_0}}(e) \, ,$$

which completes the proof of (3). ∎

5.4 Stability Property of Minimum Priority Share Edges

In this section, we show a simple stability property for any edge that becomes a minimum priority share edge in the course of an execution of a priority bottleneck algorithm. Roughly speaking, we establish that the edge remains a minimum priority share edge.

Proposition 5.6 *Assume that* Alg *is a priority bottleneck algorithm. For any integer* $l_0 \geq 0$, *fix any edge* $e \in MPSE_{Q_{l_0}}$. *Then, for any integer* $l \geq l_0$ *such that* $e \in AE_{Q_l}$, $e \in MPSE_{Q_l}$.

Proof: Consider any edge $e' \in AE_{Q_l}$; clearly, $e' \in AE_{Q_{l_0}}$. Since $e \in MPSE_{Q_{l_0}}$, this implies that $PS_{Q_{l_0}}(e) \leq PS_{Q_{l_0}}(e')$. By Lemma 5.2, e is a priority bottleneck edge for Q_{l_0}; thus, by Proposition 5.4(1), $PS_{Q_l}(e) = PS_{Q_{l_0}}(e)$. By Proposition 5.3, $PS_{Q_{l_0}}(e') \leq PS_{Q_l}(e')$. It follows that $PS_{Q_l}(e) \leq PS_{Q_l}(e')$. Since e' was chosen arbitrarily, this implies that $e \in MPSE_{Q_l}$, as needed. ■

6 Termination Properties of Priority Bottleneck Algorithms

In this section, we prove termination properties of priority bottleneck algorithms. Section 6.1 establishes the existence of maximum priority saturated edges upon termination of a priority bottleneck algorithm. In Section 6.2, we characterize the output rate vector of a priority bottleneck algorithm upon its termination.

6.1 Maximum Priority Saturated Edges

Define a *maximum priority saturated edge for session* S *in state* Q to be an edge $e \in S$ such that:

(1) for each session $i \in \mathit{sessions}(e)$, $\mathcal{P}_Q(S) \geq \mathcal{P}_Q(S_i)$;

(2) e is saturated in Q.

Example 6.1 We refer again to the simple network in Figure 1, and we retain all assumptions on priority functions for this network that were made in Example 5.1. Consider any state Q with rate vector $\mathbf{r}_Q = \langle 28, 14, 84, 10\rangle^{(\mathrm{T})}$. We will demonstrate the existence of a maximum priority saturated edge for session S_3 in state Q. Since session S_3 traverses edges e_1 and e_2 only, e_1 and e_2 are the only candidates for being a maximum priority saturated edge for session S_3 in state Q.

We first consider edge e_2. Clearly, $\sum_{i \in sessions(e_2)} r_Q(S_i) = r_Q(S_3) + r_Q(S_4) = 84 + 10 = 94$, while $cap(e_2) = 100$. It follows that edge e_2 is not saturated in state Q and, therefore, that e_2 is not a maximum priority saturated edge for session S_3 in state Q.

We now turn to consider edge e_1. Clearly, $\sum_{i \in sessions(e_1)} r_Q(S_i) = r_Q(S_1) + r_Q(S_2) + r_Q(S_3) = 28 + 14 + 84 = 126$, while $cap(e_1) = 126$. It follows that e_1 is saturated in Q. Furthermore, $\mathcal{P}_Q(S_3) = \mathbf{F}_3^{-1}(r_Q(S_3)) = r_Q(S_3)/3 = 84/3 = 28$, while $\mathcal{P}_Q(S_1) = \mathbf{F}_1^{-1}(r_Q(S_1)) = r_Q(S_1) = 28$, and $\mathcal{P}_Q(S_2) = \mathbf{F}_2^{-1}(r_Q(S_2)) = 2\, r_Q(S_2) = 2 \cdot 14 = 28$; thus, both $\mathcal{P}_Q(S_3) \geq \mathcal{P}_Q(S_1)$, and $\mathcal{P}_Q(S_3) \geq \mathcal{P}_Q(S_2)$. It follows that edge e_1 is a maximum priority saturated edge for session S_3 in state Q.

We encourage the reader to verify that edge e_1 is also a maximum priority saturated edge for both sessions S_1 and S_2 in state Q, and that there is no maximum priority saturated edge for session S_4 in state Q. $\square$

We remark that our definition of maximum priority saturated edge for a session generalizes the definition due to Hayden [20] (see also [18, Section 3]) of a *bottleneck edge for a session* to the setting with priorities. We prove:

Theorem 6.1 (Hayden's analog) *Assume that* Alg *is a priority bottleneck algorithm. Then, for any reachable final state Q of* Alg*, there exists, for each session S, a maximum priority saturated edge for S in state Q.*

Proof: Consider a final state Q in execution α of Alg, and take any sesion S. Since Q is final, $S \in \mathcal{D}_Q$. Thus, there exists some state Q_{l_0} in execution α, $Q_0 \stackrel{\alpha}{\longrightarrow} Q_{l_0} \stackrel{\alpha}{\longrightarrow} Q$ such that $S \in \mathsf{Term}(\langle G, \mathcal{S} \rangle, Q_{l_0})$; that is, session S becomes done in Q_{l_0}, so that $S \in \mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l_0+1}}$.

Since Alg is a priority bottleneck algorithm, there exists an edge $e \in E$, where S traverses e, such that *(i)* e is a priority bottleneck edge for Q_{l_0}, and *(ii)* $\mathcal{P}_{Q_{l_0}}(S) = PS_{Q_{l_0}}(e)$. We will show that e satisfies conditions (1) and (2) in the definition of a maximum priority saturated edge for session S in state Q. We start by showing (1).

Lemma 6.2 *Fix any session $i \in sessions(e)$. Then, $\mathcal{P}_Q(S) \geq \mathcal{P}_Q(S_i)$.*

Proof: There are two cases to consider.

1. Assume first that $i \in \mathcal{A}_{Q_{l_0}}$.

   Denote Q_l the latest state in α such that $\mathcal{A}_{Q_l} \mid e \neq \emptyset$; that is, Q_l is the latest state in α such that e remains active in Q_l. Clearly, since Q is a final state with $\mathcal{A}_Q = \emptyset$, $Q_l \stackrel{\alpha}{\longrightarrow} Q$.

Clearly, both $S \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e$ and $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e$. Hence, Proposition 5.4(3) implies that $\mathcal{P}_{Q_l}(S) = PS_{Q_{l_0}}(e)$ and $\mathcal{P}_{Q_l}(S_i) = PS_{Q_{l_0}}(e)$, so that $\mathcal{P}_{Q_l}(S) = \mathcal{P}_{Q_l}(S_i)$.

By definition of Q_l, no further change to the priorities of sessions traversing e occurs in any state following Q_l. Since $Q_l \xrightarrow{\alpha} Q$, it follows that $\mathcal{P}_Q(S) = \mathcal{P}_Q(S_i)$, as needed.

2. Assume now that $i \notin \mathcal{A}_{Q_{l_0}}$.

   Denote $Q_{l'}$ the state in α such that $i \in \mathsf{Term}(\langle G, \mathcal{S}\rangle, Q_{l'})$. Clearly, $Q_{l'} \xrightarrow{\alpha} Q_{l_0}$. Since Alg is a priority bottleneck algorithm, there exists an edge $e' \in E$, where S_i traverses e', such that *(i)* e' is a priority bottleneck edge for $Q_{l'}$, and *(ii)*

$$\mathcal{P}_{Q_{l'}}(S_i) \quad = \quad PS_{Q_{l'}}(e') \, .$$

   By definition of a priority bottleneck edge, $PS_{Q_{l'}}(e') = MPS_{Q_{l'}}(S_i)$. Since i traverses e, this implies that

$$PS_{Q_{l'}}(e') \quad \leq \quad PS_{Q_{l'}}(e) \, .$$

   Since $S \in \mathcal{A}_{Q_{l_0}} \mid e$, e is active in state Q_{l_0}. Thus, by Proposition 5.3,

$$PS_{Q_{l'}}(e) \quad \leq \quad PS_{Q_{l_0}}(e) \, .$$

   Since $i \in \mathsf{Term}(\langle G, \mathcal{S}\rangle, Q_{l'})$ and $Q_{l'} \xrightarrow{\alpha} Q$,

$$\mathcal{P}_Q(S_i) \quad = \quad \mathcal{P}_{Q_{l'}}(S_i) \, .$$

   It follows that

$$\mathcal{P}_Q(S_i) \quad \leq \quad PS_{Q_{l_0}}(e) \, .$$

   Since $S \in \mathsf{Term}(\langle G, \mathcal{S}\rangle, Q_{l_0})$ and $Q_{l_0} \xrightarrow{\alpha} Q$,

$$\mathcal{P}_Q(S) \quad = \quad \mathcal{P}_{Q_{l_0}}(S) \, .$$

   Since $\mathcal{P}_{Q_{l_0}}(S) = PS_{Q_{l_0}}(e)$, it follows that

$$\mathcal{P}_Q(S_i) \quad \leq \quad \mathcal{P}_Q(S) \, ,$$

   as needed.

■

We continue to show (2).

Lemma 6.3 *e is saturated in state Q.*

Proof: Consider the latest state Q_l in execution α such that e remains active in Q_l. By Proposition 5.4(3), for any session $i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_{l+1}}) \mid e = \mathcal{A}_{Q_{l_0}}$, $\mathcal{P}_{Q_l}(S_i) = PS_{Q_{l_0}}(e)$. Hence, by Lemma 4.3(1), edge e is saturated in Q_l. However, by definition of Q_l, no change to the rates of sessions traversing e occurs in any state following Q_l. Since $Q_l \xrightarrow{\alpha} Q$, it follows that e remains saturated in Q, as needed. ■

By Lemmas 6.2 and 6.3, it follows that e is a maximum priority saturated edge for session S in state Q, as needed. ■

We remark that Proposition 6.1 generalizes a corresponding result of Hayden [20] shown for classical max-min fairness to the setting with priorities. (See also [18, Section 3] for a discussion of this result.)

6.2 Output of Priority Bottleneck Algorithms

We now show that the output rate vector of any priority bottleneck algorithm is a priority max-min fair rate vector.

Theorem 6.4 (Output of priority bottleneck algorithms) *Assume that* Alg *is a priority bottleneck algorithm. Then, for any reachable final state Q of* Alg*, $\mathbf{r}_Q$ is a priority max-min fair rate vector.*

Proof: Assume, by way of contradiction, that $\mathbf{r}_Q$ is *not* a priority max-min fair rate vector. By definition of the priority max-min fair rate vector, there exists some session S_i such that increasing $r_Q(S_i)$, while still maintaining feasibility, is possible without decreasing $r_Q(S_j)$ for any session S_j such that $\mathcal{P}_Q(S_j) \leq \mathcal{P}_Q(S_i)$.

Hence, consider an increase to $r_Q(S_i)$. We will derive a contradiction by showing that a decrease to $r_Q(S_j)$, for some session S_j such that $\mathcal{P}_Q(S_j) \leq \mathcal{P}_Q(S_i)$, is necessary.

By Theorem 6.1, there exists a maximum priority saturated edge e for S_i in state Q. Since e is saturated in Q, while an increase to $r_Q(S_i)$ is possible, there must be at least one other than S_i session S_j traversing e, whose rate must decrease in order to maintain the capacity constraint for e. Moreover, since e is a maximum priority shared edge for S_i in state Q and S_j traverses e, $\mathcal{P}_Q(S_j) \leq \mathcal{P}_Q(S_i)$. This completes the contradiction. ■

We remark that Theorem 6.4 provides a generalization to the setting with priorities of a classical result of Hayden [20] that bottleneck algorithms converge to max-min fair rate vectors (see [2, Section 6.4.5] for a textbook discussion).

7 The Optimistic Case

In this section, we focus on the optimistic case. In Section 7.1, we introduce the priority update operation upon which optimistic algorithms are built. Additional combinatorial properties of minimum priority share edges are shown in Section 7.2 for the case of optimistic, priority bottleneck algorithms. In turn, these properties are used to show termination properties of optimistic, priority bottleneck algorithms in Section 7.3. Specific such algorithms are designed in Section 7.4 to exploit these termination properties.

7.1 The Priority Update Operation

Our presentation is heavily influenced by the one in [15, Sections 2.2 & 3.5]. Throughout this section, for any vector $\mathbf{r}$, denote $\|\mathbf{r}\|_1 = \sum_{l \geq 1} r_l$.

We will introduce the priority update operation, which is a generalization of the *update operation* (see [1, Section 2.1] or [15, Section 3.5]). Roughly speaking, the priority update operation determines an increase to the priority of a scheduled session; the increase is such that the scheduled session becomes the session with the maximum priority on some particular link of the network.

We start with an abstract mathematical function.

The Priority Update Function

Fix any integer m, $2 \leq m \leq n$. The *priority update function* is a function `pupdate` that takes as input a triple of a vector $\mathbf{r} \in \Re^m$, an integer $i \in [m]$, and a real number $c \in \Re$ such that $\|\mathbf{r}\|_1 \leq c$, and returns as output a pair $\langle \mathbf{r}', \Delta \rangle = \texttt{pupdate}(\langle \mathbf{r}, i, c \rangle)$ of a vector $\mathbf{r}'$ and a real number $\Delta \geq 0$, so that the following conditions are satisfied:

(1) $\|\mathbf{r}'\|_1 = c$;

(2) $r'_i = \mathbf{F}_i(\mathbf{F}_i^{-1}(r_i) + \Delta)$;

(3) for each $l \in [m]$, $l \neq i$, if $\mathbf{F}_l^{-1}(r_l) \leq \mathbf{F}_i^{-1}(r_i) + \Delta$, then $r'_l = r_l$;

(4) for each $l \in [m]$, $l \neq i$, if $\mathbf{F}_l^{-1}(r_l) > \mathbf{F}_i^{-1}(r_i) + \Delta$, then $r'_l = \mathbf{F}_l(\mathbf{F}_i^{-1}(r_i) + \Delta)$.

Whenever $\langle \mathbf{r}', \Delta \rangle = \texttt{pupdate}(\langle \mathbf{r}, i, c \rangle)$, call $\mathbf{r}'$ a *priority reform* for $\langle \mathbf{r}, i, c \rangle$ that corresponds to *priority reform increase* Δ (cf. [15, Section 3.5] or [1, Appendix A, Definition A.1]). Intuitively, a priority reform "saturates" c (condition (1)) by increasing entry r_i to its image, under the priority function $\mathbf{F}_i$, of an increase by Δ to its corresponding priority (condition (2)); moreover, entries whose priorities do not exceed the priority of the increased entry are not affected (condition (3)), while the reform preserves "fairness" in a sense since, in addition, there can be left no entries with priorities larger than the priority of the increased entry (condition (4)). (We refer the reader to [16] for a formal proof that `pupdate` is indeed a function.)

We remark that the definition of the priority update function explicitly uses the priority functions $\mathbf{F}_i$, $1 \leq i \leq n$. The *update function* `update` [15, Section 2.2] is easily seen to be the special case of the priority update function where all priority functions are equal to the identity function.

Priority Increases

For any session S_i, the *priority increase for S_i in state Q imposed by edge $e \in S_i$* (cf. [1, Section 2.1]), denoted $\Delta_Q(i, e)$, is either 0 if S_i is not active in state Q, or equal to the priority reform increase for $\langle \mathbf{r}_{\mathcal{A}_Q} \mid e, i, \mathit{cap}(e) - \mathit{allot}_Q(e) \rangle$; that is, $\Delta_Q(i, e)$ is the real number $\Delta > 0$ such that

$$\langle \mathbf{r}', \Delta \rangle \quad = \quad \texttt{pupdate}(\langle \mathbf{r}_{\mathcal{A}_Q} \mid e, i, \mathit{cap}(e) - \mathit{allot}_Q(e) \rangle) \, ,$$

for some rate vector $\mathbf{r}'$. (Note that, by definition of priority update function, $\mathbf{r}'$ is "compatible" with $\mathbf{r}_{\mathcal{A}_Q} \mid e$; so, it contains only entries that correspond to sessions active in state Q that traverse edge e.) In more detail, $\Delta_Q(i, e)$ is the *unique* possible increase to the priority of session S_i in state Q that "saturates" edge e, while it possibly decreases down to the increased rate of S_i the rates of other active sessions passing through e whose priorities exceed that of the increased rate, but it does not affect the active sessions traversing e whose priorities are smaller than the increased priority of S_i, neither, of course, the done sessions traversing e, or any other session in the network that does not cross edge 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.

The *priority increase for S_i in state Q* (cf. [1, Section 2.1]), denoted $\Delta_Q(i)$, is either 0 if S_i is not active in state Q, or the *minimum,* over all edges $e \in S_i$, of the priority increase for S_i in state Q imposed by edge e; that is,

$$\Delta_Q(i) \quad = \quad \min_{e \in S_i} \Delta_Q(i, e) \, ;$$

thus, assuming S_i is active in state Q, $\Delta_Q(i)$ is the minimum among all possible priority increases to the rate of S_i that are imposed by the edges it traverses.

Since the rate of S_i can be increased by a certain amount only if it can be increased by this amount, while still maintaining feasibility, on every edge it traverses, it follows that $\Delta_Q(i)$ is the *maximum* possible increase to the rate of S_i in state Q that maintains feasibility. We remark that $\Delta_Q(i)$ can be computed from information that is "local" to session S_i (cf. [22, Section IV]).

The Priority Update Operation

Roughly speaking, a priority update operation defines a "procedure" to compute new rates for a set a sessions, on the basis of their old rates and their priority functions. In more detail, a *priority update operation* is a set of specific instantiations of the update function, which refer to some particular session index i and state Q, followed by a corresponding computation of new rates for some sessions. In such update operation, each edge e traversed by S_i is associated with a particular instantiation of the update function. In the instantiation of the update function associated with edge e, $\Delta_Q(i, e)$ is computed by

$$\langle \mathbf{r}', \Delta_Q(i, e)\rangle \quad = \quad \texttt{pupdate}(\langle \mathbf{r}_{\mathcal{A}_Q} \mid e, i, \mathit{cap}(e) - \mathit{allot}_Q(e)\rangle) .$$

Finally, new rates for session S_i and for active sessions that "interfere" with S_i are computed, using $\Delta_Q(i)$, as shown below, where we use assignment notation to denote the new rates.

(1) $r(S_i) := \mathbf{F}_i(\mathbf{F}_i^{-1}(r(S_i)) + \Delta_Q(i))$;

(2) for each $l \in \mathcal{A}_Q$, $l \neq i$, such that $S_l \cap S_i \neq \emptyset$,

$$r(S_l) \quad := \quad \min\{r(S_l), \mathbf{F}_i(\mathbf{F}_i^{-1}(r(S_i)) + \Delta_Q(i))\} .$$

Notice that each instantiation of the priority update function computes an increase $\Delta_Q(i, e)$ that "saturates" edge e (according to the definition of the priority update function). However, an increase of $\Delta_Q(i)$ for any edge e traversed by S_i is, by definition of $\Delta_Q(i)$, no more (and possibly less) than the increase $\Delta_Q(i, e)$ that would "saturate" it (according to the definition of the priority update function). It follows that the final rates computed by the priority update operation "saturate" only the edge(s) "realizing" $\Delta_Q(i)$, and no other edges.

Example 7.1 Consider our example network depicted in Figure 1 once more. We still retain all assumptions on priority functions for this network that we made in Example 5.1.

Consider the initial state Q_0 and assume that S_1 is the session to be scheduled first. Since priorities of all sessions are initially 0, the rate of session S_1, and thus its priority, may be increased to a value that saturates edge e_1, which is the only edge traversed by it. The reader can easily verify that $\Delta_{Q_0}(1, e_1) = 126$, so that $\Delta_{Q_0}(1) = \Delta_{Q_0}(1, e_1) = 126$. Thus, $\mathcal{P}_{Q_0}(S_1) = \mathcal{P}_{Q_0}(S_1) + \Delta_{Q_0}(1) = 0+126 = 126$, which implies that $r_{Q_1}(S_1) = \mathbf{F}_1(\mathcal{P}_{Q_1}(S_1)) = \mathcal{P}_{Q_1}(S_1) = 126$. Since rates of all other sessions are initially 0, (2) implies that they will remain zero in state Q_1. Note also that the rate assignment in state Q_1 saturates edge e_1.

Assume now that session S_2 is scheduled next. Recall that session S_2 traverses edge e_1, which is saturated in state Q_1. Hence, any increase to the priority of session S_2 imposed by edge e_1 necessarily causes the priority of session S_1 to fall off to a value equal to that of S_2, while edge e_1 still remains saturated. We ask the reader to verify that the priority increase $\Delta_{Q_1}(2, e_1) = 84$ satisfies both of these constraints. Since session S_2 does not traverse any edge other than e_1, it follows that $\Delta_{Q_1}(1) = 84$, so that $\mathcal{P}_{Q_2}(S_1) = 84$. Moreover, $r_{Q_2}(S_2) = \mathbf{F}_2(\mathcal{P}_{Q_2}(S_2)) = \mathcal{P}_{Q_2}(S_2)/2 = 84/2 = 42$, while $r_{Q_2}(S_1) = \mathbf{F}_1(\mathcal{P}_{Q_1}(S_1)) = \mathcal{P}_{Q_1}(S_1) = 84$. Clearly, $r_{Q_2}(S_1) + r_{Q_2}(S_2) + r_{Q_2}(S_3) = 84 + 42 + 0 = 126$. Since $cap(e_1) = 126$, it follows that edge e_1 remains saturated in state Q_2.

Assume finally that session S_3 is scheduled next. We ask the reader to verify that the priority increase for S_3 in state Q_2 imposed by edge e_1 is $\Delta_{Q_2}(3, e_1) = 28$, and that the priority increase for S_3 in state Q_2 imposed by edge e_2 is $\Delta_{Q_2}(3, e_2) = 100/3 > 28$. Thus, $\Delta_{Q_2}(3) = 28$, which implies that $\mathcal{P}_{Q_3}(S_3) = 28$, while priorities of sessions S_1 and S_2 are also decreased to 28 in state Q_3. Thus, $r_{Q_3}(S_1) = \mathbf{F}_1(\mathcal{P}_{Q_3}(S_1)) = \mathcal{P}_{Q_3}(S_1) = 28$, $r_{Q_3}(S_2) = \mathbf{F}_2(\mathcal{P}_{Q_3}(S_2)) = \mathcal{P}_{Q_3}(S_2)/2 = 28/2 = 14$, and $r_{Q_3}(S_3) = \mathbf{F}_3(\mathcal{P}_{Q_3}(S_3)) = \mathcal{P}_{Q_3}(S_3) = 84$. Thus, $r_{Q_3}(S_1) + r_{Q_3}(S_2) + r_{Q_3}(S_3) = 28 + 14 + 84 = 126$. Since $cap(e_1) = 126$, it follows that edge e_1 remains saturated in state Q_3. $\square$

In the sequel, we will abuse notation by using `pupdate`, the notation for priority update function, to denote a priority update operation as well.

For an *optimistic algorithm,* assume that `operation` = `pupdate`; that is, an optimistic algorithm uses the (optimistic) priority update operation to adjust session rates and priorities. Since the priority update operation preserves the capacity constraint, Proposition 4.1 immediately implies that priority shares exist uniquely in an execution of an optimistic algorithm.

7.2 Properties of Minimum Priority Share Edges

In this section, we show that optimistic, priority bottleneck algorithms allow minimum priority share edges to enjoy additional combinatorial properties.

Safety Property

We start with some kind of a "safety" property for any edge that becomes a minimum priority share edge in the course of an execution of an optimistic, priority bottleneck algorithm.

Proposition 7.1 *Assume that* Alg *is an optimistic, priority bottleneck algorithm. For any integer* $l_0 \geq 0$*, fix any edge* $e \in MPSE_{Q_{l_0}}$*. Consider any session* $S_i \in \mathcal{A}_{Q_{l_0}} \mid e$ *such that* $\mathcal{P}_{Q_{l_0}}(S_i) \geq PS_{Q_{l_0}}(e)$*. Then, for any integer* $l \geq l_0$*,*

$$\mathcal{P}_{Q_l}(S_i) \quad \geq \quad PS_{Q_{l_0}}(e)\,.$$

Proposition 7.1 considers any (active) session traversing a minimum priority share edge; roughly speaking, it establishes that no decrease of its priority below this particular minimum priority share is possible if, in the first place, its priority is no less than the minimum priority share. The proof follows.

Proof: By induction on l. For the basis case where $l = l_0$, the claim holds by assumption.

Assume inductively that for some integer $l > l_0$, $\mathcal{P}_{Q_{l-1}}(S_i) \geq PS_{Q_{l_0}}(e)$. For the induction step, we show that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$.

Assume first that $\mathcal{P}_{Q_l}(S_i) \geq \mathcal{P}_{Q_{l-1}}(S_i)$; that is, no decrease to the priority of session S_i occurs going from state Q_{l-1} to state Q_l. By induction hypothesis, this implies that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$, as needed.

Assume now that $\mathcal{P}_{Q_l}(S_i) < \mathcal{P}_{Q_{l-1}}(S_i)$; that is, the priority, of session S_i decreases going from state Q_{l-1} to state Q_l. By definition of priority update operation, session S_i must intersect session S_{i_l}, the one scheduled in front of state Q_l.

Let e' be an edge such that $\Delta_{Q_{l-1}}(i_l) = \Delta_{Q_{l-1}}(i_l, e')$; roughly speaking, the edge e' "realizes" the least, among all edges traversed by session S_{i_l}, of all maximum priority reform increases for session S_{i_l} in state Q_{l-1}. We prove:

Lemma 7.2 $\mathcal{P}_{Q_l}(i_l) \geq PS_{Q_l}(e')$

Proof: Assume, by way of contradiction, that $\mathcal{P}_{Q_l}(i_l) < PS_{Q_l}(e')$.

By definition of priority update operation, e' is saturated in Q_l.

By definition of priority update operation, for any session index $k \in \mathcal{A}_{Q_l} \mid e'$, $\mathcal{P}_{Q_l}(i_l) \geq \mathcal{P}_{Q_l}(k)$; thus, $PS_{Q_l}(e') > \mathcal{P}_{Q_l}(k)$. Hence, Lemma 4.3(2) implies that e' is not saturated in Q_l. A contradiction. ∎

By definition of priority update operation $\mathcal{P}_{Q_l}(S_i) = \mathcal{P}_{Q_l}(i_l)$; thus, Lemma 7.2 implies that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_l}(e')$. By Proposition 5.3, $PS_{Q_l}(e') \geq PS_{Q_{l_0}}(e')$, which implies that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e')$. Since $e \in MPSE_{Q_{l_0}}$, $PS_{Q_{l_0}}(e') \geq PS_{Q_{l_0}}(e)$. It follows that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$, as needed. ■

We remark that Proposition 7.1 provides a generalization of [15, Proposition 4.3] to the setting with priorities.

Liveness Property

We continue with a "liveness" property for any edge that becomes a minimum priority share edge in the course of an execution of a priority bottleneck algorithm.

For any state Q_l in an execution α of an (optimistic) algorithm, denote by $\widehat{l}$ the *least* integer such that all sessions $S_i \in \mathcal{A}_{Q_l}$ have been scheduled for an increase (at least) once in front of some state following Q_l but not preceded by $Q_{\widehat{l}}$ in the execution α, or infinite if no such integer exists. Similarly, denote by $\widehat{l} \mid e$ the *least* integer such that all sessions $S_i \in \mathcal{A}_{Q_l} \mid e$ have been scheduled for an increase (at least) once in front of some state following Q_l but not preceded by $Q_{\widehat{l}|e}$ in the execution α, or infinite if no such integer exists. We prove:

Proposition 7.3 *Assume that* Alg *is an optimistic, priority bottleneck algorithm. For any integer* $l_0 \geq 0$*, fix any edge* $e \in MPSE_{Q_{l_0}}$ *such that* $\widehat{l_0} \mid e < \infty$*. Consider any session* $S_i \in \mathcal{A}_{Q_{l_0}} \mid e$*. Then, for any integer* $l \geq \widehat{l_0} \mid e$*,*

$$\mathcal{P}_{Q_l}(S_i) \quad \geq \quad PS_{Q_{l_0}}(e)\,.$$

Roughly speaking, we consider any session traversing a minimum priority share edge; we establish that no decrease to its priority below this particular minimum priority share is eventually possible, once all sessions traversing this minimum priority share edge have been scheduled for an increase at least once. The proof follows.

Proof: We start with an informal outline of our proof. We consider the point of the execution following state Q_{l_0} where S_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 priority of S_i is no less than the priority share of edge e in state Q_{l_0}. We also argue that e remains a minimum priority share edge beyond state Q_{l_0}; this allows us to exploit the safety property of minimum priority share edges that was established in Proposition 7.1 in order to argue that the priority of S_i

will subsequently remain no less than the priority share of e in state Q_{l_0}. We now present the details of the formal proof.

Since $e \in MPSE_{Q_{l_0}}$, Lemma 5.2 implies that e is a priority bottleneck edge for state Q_{l_0}. Since $S_i \in \mathcal{A}_{Q_{l_0}} \mid e$, it follows by definition of $\widehat{l_0} \mid e$ that there exists a least index l', $l_0 < l' \leq \widehat{l_0} \mid e$, such that S_i is scheduled in front of state $Q_{l'}$. Since S_i is active in state Q_{l_0}, it follows by definition of execution that S_i is active in $Q_{l'}$. Since S_i traverses edge e, it follows that e is active in state $Q_{l'}$. By Proposition 5.4(1), $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$, while by Proposition 5.4(2), e is a priority bottleneck edge for $Q_{l'}$. We continue to prove:

Lemma 7.4 $\mathcal{P}_{Q_{l'}}(S_i) \geq PS_{Q_{l_0}}(e)$.

Proof: Assume, by way of contradiction, that $\mathcal{P}_{Q_{l'}}(S_i) < PS_{Q_{l_0}}(e)$. Let e' be an edge such that $\Delta_{Q_{l'}}(S_i) = \Delta_{Q_{l'}}(S_i, e')$; roughly speaking, the edge e' "realizes" the least, among all edges traversed by S, of all maximum priority reform increases for session S_i in state $Q_{l'}$. By definition of priority update operation, e' is saturated in state $Q_{l'}$.

Since e is a priority bottleneck edge for $Q_{l'}$, and S traverses both e and e', $PS_{Q_{l'}}(e) \leq PS_{Q_{l'}}(e')$. Since $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$, this implies that $PS_{Q_{l_0}}(e) \leq PS_{Q_{l'}}(e')$. Since $\mathcal{P}_{Q_{l'}}(S_i) < PS_{Q_{l_0}}(e)$, it follows that $\mathcal{P}_{Q_{l'}}(S_i) < PS_{Q_{l'}}(e')$. By definition of priority update operation, for any session index $k \in \mathcal{A}_{Q_{l'}} \mid e$, $\mathcal{P}_{Q_{l'}}(S_k) \leq \mathcal{P}_{Q_{l'}}(S_i)$, so that $\mathcal{P}_{Q_{l'}}(S_k) < FS_{Q_{l'}}(e')$. It follows by Lemma 4.3(2) that e' is not saturated in state $Q_{l'}$. A contradiction. ■

Take now any integer $l \geq \widehat{l_0} \mid e$. Clearly, $l \geq l'$. Since e is a minimum priority share edge for Q_{l_0}, Proposition 5.6 implies that e is a minimum priority share edge for $Q_{l'}$ as well. Moreover, by Lemma 7.4, $\mathcal{P}_{Q_{l'}}(S_i) \geq PS_{Q_{l_0}}(e)$. Since $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$, this implies that $\mathcal{P}_{Q_{l'}}(S_i) \geq PS_{Q_{l'}}(e)$. It follows by Proposition 7.1 (taking l' for l_0) that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$, as needed. ■

We remark that Proposition 7.3 provides a generalization of [15, Proposition 4.5] to the setting with priorities.

7.3 Termination Properties of Optimistic, Priority Bottleneck Algorithms

We proceed to establish fundamental termination properties of optimistic, priority bottleneck algorithms.

Termination of Sessions Through a Minimum Priority Share Edge

The first property considers active sessions that traverse a minimum priority share edge; we show that after each such session has been scheduled at least once, all of these sessions must have become done.

Theorem 7.5 *Assume that* Alg *is an optimistic, priority bottleneck algorithm. For any integer* $l_0 \geq 0$*, fix any edge* $e \in MPSE_{Q_{l_0}}$ *such that* $\widehat{l_0} \mid e < \infty$*. Then, for any session* $S_i \in \mathcal{A}_{Q_{l_0}} \mid e$*,*

$$S_i \;\; \in \;\; \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0}|e}}} .$$

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

Fix any session $S_i \in \mathcal{A}_{Q_{l_0}} \mid e$. We start by proving:

Lemma 7.6 $\mathcal{P}_{Q_{\widehat{l_0}|e}}(S_i) = PS_{Q_{l_0}}(e)$

Proof: Assume, by way of contradiction, that $\mathcal{P}_{Q_{\widehat{l_0}|e}}(S_i) \neq PS_{Q_{l_0}}(e)$. By Proposition 7.3, $\mathcal{P}_{Q_{\widehat{l_0}|e}}(S_i) \geq PS_{Q_{l_0}}(e)$. It follows that $\mathcal{P}_{Q_{\widehat{l_0}|e}}(S_i) > PS_{Q_{l_0}}(e)$. We proceed by case analysis.

1. Assume first that there exists no session $S_j \in \mathcal{A}_{Q_{l_0}} \mid e$, $j \neq i$. Thus, $\mathcal{A}_{Q_{l_0}} \mid e = \{i\}$. Denote Q_l the latest state in execution α, $\overleftarrow{Q_{l_0}} \longrightarrow Q_l \longrightarrow \overrightarrow{Q_{\widehat{l_0}|e}}$, such that $\mathcal{A}_{Q_l} \neq \emptyset$. Thus, $\mathcal{A}_{Q_l} \mid e = \mathcal{A}_{Q_{l_0}} \mid e = \{i\}$ and $allot_{Q_l} = allot_{Q_{l_0}}$. Clearly,

$$\begin{aligned}
\sum_{j \in \mathcal{A}_{Q_l}|e} r_{Q_l}(S_j) \;\; &= \;\; r_{Q_l}(S_i) \\
&= \;\; r_{Q_{\widehat{l_0}|e}}(S_i) \\
&\qquad \text{(by definition of state } Q_l\text{)} \\
&= \;\; \mathbf{F}_i(\mathcal{P}_{Q_{\widehat{l_0}|e}}(S_i)) \\
&\qquad \text{(by definition of priority functions)} \\
&> \;\; \mathbf{F}_i(PS_{Q_{l_0}}(e))
\end{aligned}$$

$$
\begin{aligned}
& \quad \text{(since } F_i \text{ is strictly increasing and } \mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) > PS_{Q_{l_0}}(e)) \\
= & \quad cap(e) - allot_{Q_{l_0}}(e) \\
& \quad \text{(by definition of priority share)} \\
= & \quad cap(e) - allot_{Q_l}(e)\,,
\end{aligned}
$$

which establishes that e violates the capacity constraint in state Q_l. A contradiction.

2. Assume now that there exists session $S_j \in \mathcal{A}_{Q_{l_0}} \mid e$, $j \neq i$. Consider any such session S_j. By Proposition 7.3, $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_j) \geq PS_{Q_{l_0}}(e)$. Together with $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) > PS_{Q_{l_0}}(e)$, this implies a contradiction to Lemma 4.3(3).

This completes the proof. ∎

We proceed by case analysis.

1. Assume first that $S_i \in \mathcal{D}_{Q_l}$ for some state Q_l in execution α such that $Q_{l_0} \longrightarrow Q_l \longrightarrow \overrightarrow{Q_{\widehat{l_0|e}}}$. By definition of execution, it follows that $S_i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0|e}}}}$, as needed.

2. Assume now that for each state Q_l in execution α such that $Q_{l_0} \longrightarrow Q_l \longrightarrow \overrightarrow{Q_{\widehat{l_0|e}}}$, $S_i \in \mathcal{A}_{Q_l}$.

   Denote Q_l the latest state in execution α such that both $Q_{l_0} \longrightarrow Q_l \longrightarrow \overrightarrow{Q_{\widehat{l_0|e}}}$ and $\mathcal{P}_{Q_l}(S_i) = \mathcal{P}_{Q_{\widehat{l_0|e}}}$.
   Since $e \in MPSE_{Q_{l_0}}$, Lemma 5.2 implies that e is a priority bottleneck edge for Q_{l_0}. Since $S_i \in \mathcal{A}_{Q_l}$ and S_i traverses edge e, it follows that $\mathcal{A}_{Q_l} \mid e \neq \emptyset$. Hence, Proposition 5.4(1) implies that

$$
PS_{Q_l}(e) \quad = \quad PS_{Q_{l_0}}(e)\,,
$$

   while Proposition 5.4(2) implies that e is a priority bottleneck edge for Q_l. By Lemma 7.6,

$$
\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) \quad = \quad PS_{Q_{l_0}|e}(e)\,.
$$

   It follows that

$$
\mathcal{P}_{Q_l}(S_i) \quad = \quad PS_{Q_l}(e)\,.
$$

   In total, e is a priority bottleneck edge for state Q_l, traversed by session S_i for which $\mathcal{P}_{Q_l}(S_i) = PS_{Q_l}(e)$. Since Alg is a priority bottleneck algorithm, it follows that $S_i \in \mathcal{D}_{\overrightarrow{Q_l}}$. Since $Q_l \rightarrow \overrightarrow{Q_{\widehat{l_0|e}}}$, it follows that $S_i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0|e}}}}$, as needed.

■

We remark that Theorem 7.5 provides a generalization of [15, Theorem 4.7] to the setting with priorities.

Termination of at Least One Session

Our final termination property is a direct consequence of Theorem 7.5. Roughly speaking, we establish that scheduling any sequence of sessions that includes all that are currently active results in finalizing the rate of *at least* one active session.

Theorem 7.7 *Assume that* Alg *is an optimistic, priority bottleneck algorithm. Then, for any integer* $l_0 \geq 0$ *such that* $\mathcal{A}_{Q_{l_0}} \neq \emptyset$ *and* $\widehat{l_0} \mid e < \infty$*. there exists some session* $S \in \mathcal{A}_{Q_{l_0}}$ *such that*

$$S \in \mathcal{D}_{\overline{Q_{\widehat{l_0}}}} .$$

Proof: Clearly, $\mathcal{A}_{Q_{l_0}} \neq \emptyset$ implies that $MPSE_{Q_{l_0}} \neq \emptyset$ as well. So fix any edge $e \in MPSE_{Q_{l_0}}$, and consider any session $S \in \mathcal{A}_{Q_{l_0}} \mid e$. Clearly, $S \in \mathcal{A}_{Q_{l_0}}$. By Theorem 7.5, $S \in \mathcal{D}_{\overline{Q_{\widehat{l_0}}}}$, as needed. ■

We remark that Theorem 7.7 provides a generalization of [15, Theorem 4.9] to the setting with priorities.

7.4 Specific Optimistic, Priority Bottleneck Algorithms

Say that Sched is *oblivious* [15, Section 3.6.1] if for all pairs $\langle G, \mathcal{S} \rangle$ and Q, and $\langle \widehat{G}, \widehat{\mathcal{S}} \rangle$ and $\widehat{Q}$, and for any integer $l \geq 1$, $\mathsf{Sched}(\langle G, \mathcal{S} \rangle, Q, l) = \mathsf{Sched}(\langle \widehat{G}, \widehat{\mathcal{S}} \rangle, \widehat{Q}, l)$; loosely speaking, an oblivious scheduler uses no "knowledge" of either the topology of the network, or the rates and status (active or done) of sessions in choosing the session to schedule next for an increase. Thus, an oblivious scheduler may be identified with a (finite or infinite) sequence $\mathsf{Sched} = i_1, i_2, \ldots$, where for each $l \geq 1$, $i_l \in [n]$. As a particular example of an oblivious scheduler, consider the "elevator" scheduler

$$\mathsf{ElevSched} = 1, 2, \ldots, n, n, \ldots, 2, 1, 1, 2, \ldots, n, n, \ldots, 2, 1, \ldots, 1, 2, \ldots, n, n, \ldots, 2, 1, \ldots .$$

Say that Alg is *oblivious* [15, Section 3.6.1] if Sched is; else, say that Alg is *non-oblivious.*

An Oblivious Algorithm

Theorem 7.7 motivates an oblivious algorithm, called Permutation, to compute the priority max-min fair rate vector.

The scheduler of Permutation, denoted PermSched, contacts n scheduling rounds. In each round, all n sessions are scheduled in any arbitrary order. Moreover, Permutation is priority bottleneck.

By definition of PermSched, each session is scheduled once in each round. Thus, Theorem 7.7 implies that at least one session becomes done in each of the n rounds. It follows that all n sessions are done after n rounds are conducted, whence the network enters a final state. Hence, Theorem 6.4 implies that Permutation computes the priority max-min fair rate vector.

Clearly, Permutation performs no more than n^2 priority update operations in the worst case. Thus, we have:

Theorem 7.8 Permutation *computes the priority max-min fair rate vector within n^2 priority update operations.*

We remark that the algorithm Permutation generalizes the algorithm RoundRobin [15, Section 7.1] to the setting with priorities.

A Non-Oblivious Algorithm

Theorem 7.5 motivates a non-oblivious algorithm, called PLinear, to compute the priority max-min rate vector.

Roughly speaking, the scheduler of PLinear maintains an edge of minimum priority share that is currently active, and schedules all active sessions traversing it in any arbitrary order. Once it finishes with all of them, it chooses some other (currently active) edge of minimum priority share, and so on. Moreover, PLinear is priority bottleneck.

Consider any state Q_{l_0} such that $e \in MPSE_{Q_{l_0}}$. By definition of the scheduler of PLinear, each session traversing e is scheduled exactly once, so that the state $Q_{\widehat{l_0|e}}$ is reached; by Theorem 7.5, each such session is done in state $Q_{\widehat{l_0|e}}$. By definition of the scheduler of PLinear, it follows that all sessions eventually become done, and the network enters a final state. Hence, Theorem 6.4 implies that PLinear computes the priority max-min fair rate vector.

Clearly, the total number of priority update operations incurred by PLinear is exactly n in every execution, which is optimal. Thus, we have:

Theorem 7.9 PLinear *computes the priority max-min fair rate vector within exactly n priority update operations.*

We remark that the algorithm PLinear is an adaptation of the algorithm Linear [15, Section 7.3] to the setting with priorities; PLinear works with minimum priority share edges, while Linear works with minimum fair share edges.

8 Discussion

We have laid out a theory of max-min fair, rate-based flow control sensitive to priorities of distributed applications, as a significant extension of the classical theory of max-min fair, rate-based flow control [2, 3, 18, 20, 21, 22, 31] to the case of networks with guaranteed QoS. Our theory yields an elegant scheme for general prioritized allocation of bandwidth to conflicting distributed applications; this scheme encompasses issues of modeling priorities via priority functions, defining fairness with respect to these priorities, and efficiently computing rates that conform to the defined fairness. As a by-product, our scheme provides a novel method for pricing distributed applications by assigning monetary prices to priority functions and allowing the applications to purchase transmission rate by selecting the priority function of their like.

We feel that our work takes a significant step towards developing an algorithmic framework for currently major trends in network algorithmics, such as flow control and routing, that explicitly takes QoS issues into account. (See also [28, 30] for recent theoretical works that address QoS issues in the context of networking problems.) As the need for guaranteed QoS becomes more widely recognized, algorithmic frameworks that capture and formalize QoS requirements, and corresponding algorithms that reach these requirements both fairly and efficiently, become more indispensable. Such frameworks and algorithms (and their evaluation) may relate issues concerning data network architectures, specification and engineering of requirements, and combinatorial optimization. We hope that our work contributes toward a more solid ground for this interaction.

References

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

[2] D. P. Bertsekas and R. G. Gallager, *Data Networks,* second edition, Prentice Hall, Englewood Cliffs, NJ, 1992.

[3] K. Bharath-Kumar and J. M. Jaffe, "A New Approach to Performance-Oriented Flow Control," *IEEE Transactions on Communications,* Vol. COM-29, No. 4, pp. 427–435, April 1981.

[4] F. Bonomi and K. Fendick, "The Rate-Based Flow Control for Available Bit Rate ATM Service," *IEEE Network,* Vol. 9, No. 2, pp. 25–39, March/April 1995.

[5] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, *Recommendations on Queue Management and Congestion Avoidance in the Internet,* April 1998.
(Available at `http://info.internet.isi.edu:90/in notes/rfc/files/rfc2309.txt`)

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

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

[8] A. Charny, D. D. Clark, and R. Jain, "Congestion Control with Explicit Rate Indication," *Proceedings of the International Conference on Communications (ICC'95),* pp. 1954–1963, June 1995.

[9] A. Charny, K. K. Ramakrishnan, and A. Lauck, "Time Scale Analysis and Scalability Issues for Explicit Rate Allocation in ATM Networks," *IEEE/ACM Transactions on Networking,* Vol. 4, No. 4, pp. 569–581, August 1996.

[10] D. D. Clark, S. Shenker and L. Zhang, "Supporting Real-Time Applications in an Intergrated Services Packet Network: Architecture and Mechanism," *Proceedings of ACM SIGCOMM,* August 1992.

[11] D. Comer, *Internetworking with TCP/IP,* Prentice Hall, Englewood Cliffs, NJ, 1995.

[12] C. Courcoubetis, F. Kelly and R. Weber, "Measurement-Based Usage Charges in Communication Networks", *Operations Research,* to appear.

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

[14] P. Fatourou, M. Mavronicolas and P. Spirakis, "Advances in Rate-Based Flow Control," *Proceedings of the 4th International Colloquium on Structural Information and Communication Complexity,* pp. 266–281, Carleton University Press, June 1997.

[15] P. Fatourou, M. Mavronicolas and P. Spirakis, "Efficiency of Oblivious versus Non-Oblivious Schedulers for Optimistic, Rate-Based Flow Control," *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing,* pp. 139–148, August 1997.

[16] P. Fatourou, M. Mavronicolas and P. Spirakis, "The Global Efficiency of Distributed, Rate-Based, Flow Control Algorithms," *Proceedings of the 5th International Colloquium on Structural Information and Communication Complexity,* pp. 244–258, Carleton University Press, June 1998. (Also appears as Brief Announcement in *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing,* p. 311, June/July 1998.)

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

[18] E. Hahne, "Round-Robin Scheduling for MaxMin Fairness in Data Networks," *IEEE Journal on Selected Areas in Communications,"* Vol. 9, No. 7, pp. 1024–1039, September 1991.

[19] R. Händel, M. N. Huber and S. Schröder, *ATM Networks: Concepts, Protocols, Applications,* Addison-Wesley, 1998.

[20] H. Hayden, "Voice Flow Control in Integrated Packet Networks," Report LIDS/TH/1152, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1981.

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

[22] J. M. Jaffe, "Flow Control Power is Nondecentralizable," *IEEE Transactions on Communications,* Vol. COM-29, No. 9, pp. 1301–1306, September 1981.

[23] S. Jamin, S. Shenker, L. Zhang and D. Clark, "Admission Control Algorithm for Predictive Real-Time Service," *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video,* November 1992.

[24] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "An Efficient Rate Allocation Algorithm for ATM Networks Providing Max-Min Fairness", *Proceedings of 6th IFIP International Conference on High Performance Networking (HPN'95),* pp. 143–154, September 1995.

[25] F. P. Kelly, "On Tariffs, Policing and Admission Control for Multiservice Networks," *Operations Research Letters*, Vol. 15, pp. 1-9, 1994.

[26] S. Keshav, *An Engineering Approach to Computer Networking,* Addison-Wesley, 1997.

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

[28] E. Koutsoupias and C. H. Papadimitriou, "Worst-case Equilibria," *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science,* G. Meinel and S. Tison eds., pp. 387–396, Vol. 1563, Lecture Notes in Computer Science, Springer-Verlag, Trier, Germany, March 1999.

[29] H. Luss and D. R. Smith, "Resource Allocation among Competing Entities: A Lexicographic Minimax Approach," *Operations Research Letters,* Vol. 5, No. 5, pp. 227–231, November 1986.

[30] Y. Mansour and B. Patt-Shamir, "Jitter Control in QoS Networks," *39th Annual IEEE Symposium on Foundations of Computer Science,* pp. 50–59, October 1998.

[31] J. Mosely, "Asynchronous Distributed Flow Control Algorithms," Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, June 1984.

[32] A. Odlyzko, "The Paris Metro Proposal for Pricing and Congestion Control on the Internet," invited paper in *DIMACS Workshop on Networks in Distributed Networking,* Special Year on Networks, DIMACS Center for Discrete Mathematics and Theoretical Computer Science, October 1997.

[33] C. Partridge and S. Pink, "An implementation of the Revised Internet Stream Protocol (ST-2)," *Internetworking: Research and Experience*, Vol. 3, No. 1, pp. 27-54, March 1992.

[34] C. Topolcic, "Experimental Internet Stream Protocol: Version 2 (ST-II)," Technical Report TR-455, Laboratory for Computer Science, Massachusetts Institute of Technology, 1989.

[35] W. K. Tsai, Y. Kim, and L. C. Hu, "ASAP: A Non-Per-VC Accounting Max-Min Protocol for ABR Flow Control with Optimal Convergence Speed," *Proceedings of IEEE SICON'98,* June/July 1998.

[36] J. Walrand and P. Varaiya, *High-Performance Communication Networks,* Morgan Kaufmann, 1996.

[37] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network,* Vol. 7, No. 5, pp. 8–18, 1993.