Max-Min Fair Flow Control Sensitive to Priorities*

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

(June 22, 2004)

*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. This work has been partially supported by the ESPRIT Program of the European Union under the Long Term Research Project ALCOM-IT (contract number 20244), by the IST Program of the European Union under Projects ALCOM-FT (contract number IST-1999-14186) and FLAGS (contract number IST-2001-33116), by funds from the Program E.Π.E.A.E.K. II of the Greek Ministry of Education, by research funds at University of Cyprus, and by AT&T/NCR research funds.

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

‡Department of Computer Science, University of Cyprus, Nicosia CY-1678, Cyprus. Part of the work of this author was performed while at Department of Computer Science and Engineering, University of Connecticut, Storrs, 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@ucy.ac.cy`

§Department of Computer Engineering and Informatics, University of Patras, Patras GR-26500, Greece, & Research and Academic Computer Technology Institute, Patras GR-26110, Greece. 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.* 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 networks supporting applications with diverse requirements on network resources.

Each individual session bears a *priority,* abstracting the session's priority to bandwidth access, and a *priority function,* which maps the session's priority to a transmission rate. Priority functions enable the specification of requirements on bandwidth access by distributed applications, and the formal handling of such requirements. We present *priority max-min fairness,* as a novel and well motivated fairness condition which requires that assigned rates correspond, through the priority functions, to priorities comprising a max-min vector. We also introduce *priority bottleneck algorithms* gradually update a session's rate until when 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. The optimistic framework allows for both decreases and increases of session rates. We exploit these additionally provided semantics to prove further combinatorial properties for the termination of priority bottleneck algorithms in the optimistic framework. We use these properties to conclude the first optimistic algorithms for efficient, max-min fair, rate-based flow control sensitive to priorities.

1 Introduction

Current *communication protocols,* ranging from ATM with Available Bit Rate (ABR) traffic to TCP/IP, have been mostly employing a simple, *best-effort* service policy in order to allow for large amounts of high-speed traffic. Nevertheless, the emerging *Integrated Services Network* will be supporting applications with diverse performance objectives (e.g., remote video, multimedia conferencing and data visualization); these often require the network to go beyond the best-effort policy and allow data traffic stream to individually determine its requirements on critical network resources such as bandwidth. Thus, there has recently been an increasing tendency to design new communication protocols and service models for the better utilization of resources. So, the *Quality-of-Service (QoS)* metrics need to be appropriately adapted, while networks must be enhanced in order to incorporate *priorities* into servicing diverse traffic classes.

Priorities can offer a basis for providing customized service to users with diverse requirements (see, e.g., [2, 7, 8]). Consider, for example, applications with a wide spectrum of delay constraints, ranging from interactive conferencing with a need for small network delays, to playback of stored video, which can more conveniently tolerate larger network delays. Given this diversity, it is only natural to prioritize differently applications with diverging requirements. In addition, the need for assigning priorities to applications often arises in the contexts of network management policies and pricing (cf. [17]).

A lot of research work has been put on incorporating priority mechanisms into both ATM (see, e.g., [8, 23, 24]) and the Internet (see, e.g., [7, 25]). For example, a resource reservation protocol called *RSVP* [26] has allowed distributed applications to signal to the network their bandwidth preferences, and the protocol reserves resources via the network switches (see [26] for an expanded discussion). *RSVP* has not relied on any underlying formal model of fairly servicing diverging user requirements; instead, it has employed an intuitive scheme, called *weighted fair queuing,* that reserves network resources on the basis of priorities called *weights.*

While priorities model resource requirements for distributed applications, *fairness* aims at prohibiting highly prioritized applications from reserving excessive resource amounts on the account of low priority applications. So, a fundamental challenge of modern networking is the incorporation of both priorities and fairness into a formal, integrated services model. In this work, we take a step in this direction by assembling a theory of *max-min fair, rate-based flow control* sensitive to priorities, as a significant extension of the classical theory of max-min fair, rate-based flow control (see, e.g., [3, 13, 14, 15]) to the setting with priorities.

In this work, we consider a *communication network* with a collection of virtual circuits, called *sessions,* running on it. Associated with each session we envision a non-negative real number

called *priority*. Priority provides a working abstraction for modeling the session's merit to network bandwidth; this merit is useful for determining the *rate* subsumed by the session on transmission. More importantly, we assume a *priority function* for each session, mapping its priority to a rate. Priority functions may be postulated on the basis of network management policies, traffic levels, or pricing considerations (cf. [17]). In our setting, priority functions provide a vehicle for expressing the *desired* level of rate service for each particular application, while the values of priority functions, on their abstract arguments that we call priorities, specify the *actual* level of rate service supplied to them. We assume common mathematical properties, such as *increasing monotonicity, continuity* and *convexity,* on the priority functions.

The most widely accepted fairness condition is *max-min fairness* [1, 3, 9, 10, 11, 12, 13, 14]; roughly speaking, it 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 varying priorities. 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.

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 for itself a priority function taking large values, which, in turn, will secure sufficient bandwidth for its transmission in a fair manner (in view of its investment). Thus, priority max-min fairness explicitly allows for arbitrarily prioritized allocation of bandwidth.

We use priority max-min fairness to define the new class of *priority bottleneck algorithms* as a stable generalization of the classical *bottleneck algorithms* [13, 14, 15]; we prove a collection of interesting combinatorial properties of these new 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 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 [3, Section 6.5]).

We finally apply our generalized theory to the more realistic, *optimistic* framework for rate-based flow control that was proposed by Afek *et al.* [1] and further examined by Fatourou *et al.* [9, 10]. In contrast to the so called *conservative* algorithms where rates never decrease during a sequence of adjustments, *optimistic* algorithms allow rates to intermediately go above their

final values. This assumption is more realistic since it provides for both increases and decreases to rates in order to accommodate sessions entering in a dynamic manner.

We enhance the optimistic framework with a *priority update operation,* which is a simple abstraction of the adjustment, on the basis of priorities and through the priority functions, of the rates of individual sessions. The complexity of an algorithm converging to priority max-min fairness, called here *convergence complexity* [1, 9, 10], 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.

Our final result is the derivation of, and analysis of convergence complexity for, *specific* priority bottleneck algorithms in the optimistic framework. This result provides an elegant generalization of corresponding recent results of Fatourou *et al.* [9] for the much simpler case of bottleneck algorithms to the setting with priorities.

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, while Section 5 defines and studies priority bottleneck algorithms; their termination properties are shown in Section 6. Optimistic, priority bottleneck algorithms are 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 [13], Jaffe [14, 15], and Luss and Smith [18]. The classical theory of fair, rate-based flow control, also referred to as *bottleneck flow control* [14], was built around max-min fairness (see, e.g., [3, 12, 13, 14, 15]). 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.* [9, 10]. None of these previous works deals with priorities (with [14] being a single exception; see below).

Asynchronous distributed algorithms that converge to max-min fair rates have been presented in [5, 6, 16, 20] (this list is not exhaustive); all of them are conservative. Optimistic algorithms have been pointed out by Afek *et al.* [1] and by Fatourou *et al.* [10]. Still, priority issues were left untouched by those works.

Work by Jaffe [14, Section X] and by Gafni and Bertsekas [11] (motivated by a voice coder scheme) are the only works known to us, published before the conference publication of this paper, that embed priority issues into max-min fair, rate-based flow control.

Jaffe introduces the notion of *throughput priority* [14, Section X] to capture preferential treatment of some sessions (users) in a heterogeneous network. Roughly speaking, the throughput priority of a session is a positive real number that multiplies its rate to yield its desired throughput. Jaffe [14, Section X] uses these concepts to provide an (equivalent) reformulation of max-min fairness. Jaffe's formulation is comparable to the special case in our general framework where rates are linearly related to priorities. This special case is too restrictive to either model bandwidth requirements in today's networks, or to allow flexible and fair pricing mechanisms [17]. For example, *elastic* applications, which are rather tolerant of end-to-end delays and throughput, can be accurately modeled by (non-linear) *utility* functions that are convex in available bandwidth (cf. [22])*. Also, *real-time* applications obeying real-time constraints are best described by *step* utility functions. Since we allow priority functions to be arbitrary functions, our framework provides larger flexibility.

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 [14, Section V] still satisfies max-min fairness. In contrast, we study and prove results for *any* priority bottleneck algorithm, whether conservative or optimistic. In addition, we show results for *any* optimistic, priority bottleneck algorithm, which we use to derive some specific such algorithms.

In a different avenue, Gafni and Bertsekas [11] formulate elegant, combinatorial conditions, involving abstract functions of both rates and link *capacities,* which are intended to model the effect of preferences among different sessions. 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, 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, 9, 10].

A number of papers (e.g., [4, 19, 21]) addressing possible generalizations of max-min fairness in order to accomodate issues of priority and utility were independently published at least one year after the original conference publication of our work in 1998. Cao and Zegura [4] discuss *utility max-min*, a fairness condition similar to priority max-min fairness. Utility max-min fairness was further discussed in [19, 21]. In particular, the necessity of the convexity assumption in the context of utility max-min is discussed in [21].

*The convexity assumption has been frequently used in modeling performance or cost functions in the networking literature, especially in network optimization problems for which a *unique* solution is saught; see, e.g., [11, 22] for studies adopting the convexity assumption in the contexts of routing and flow control.

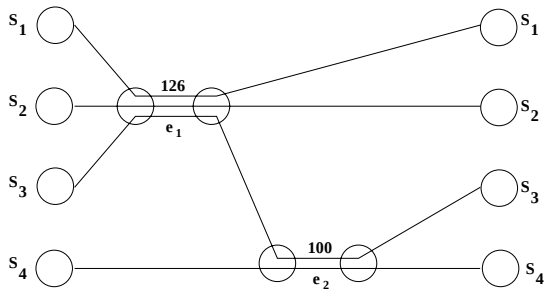

Figure 1: A simple network

3 Model

Our definitions closely follow, adapt and generalize corresponding ones in [9, Section 3].

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; 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*. We consider a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n sessions laid 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 $\mathcal{S}' \mid e = \mathcal{S}' \cap sessions(e)$. We will often abuse notation by identifying a session S_i with its index i; we will sometimes treat a session S_i as the set of its links, so that $e \in S_i$ for any edge e traversed by S_i.

Example 3.1 Figure 1 depicts a simple network with sessions S_1, S_2, S_3 and S_4, and edges, e_1 and e_2 with $cap(e_1) = 126$ and $cap(e_2) = 100$, respectively. Sessions S_1 and S_2 traverse only edge e_1; session S_4 traverses only edge e_2, while session S_3 traverses both edges. □

Allocated to each session S is a *rate* $r(S) \geq 0$. The vector $\mathbf{r} = \langle r(S_1), r(S_2), \ldots, r(S_n) \rangle$ is the *rate vector*. The *capacity constraint* requires that 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 the sum of rates of sessions sharing the edge equals the edge capacity; we will also say that $\mathbf{r}$ *saturates* e in this case.

The network is abstracted as a state machine. Each state Q of the network consists of two components: a feasible rate vector $\mathbf{r} \in \Re^n$ and a 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 $\mathcal{D}_Q = \mathcal{S} \setminus \mathcal{A}_Q$, the set of *done* (or *terminated*) sessions in state Q whose rates have been finalized and will not change any further. The *status* of a session in state Q is either *active* or *done*. In the *initial state* Q^{in}, all sessions are active and have zero rates. A state in which all sessions are done is *final*.

The set of *active edges* of the network in state Q, denoted AE_Q, contains all edges of the network traversed by at least one active session in state Q. For any edge $e \in E$ and state Q, the *allotted capacity of e in state Q* [1, Section 2.1], denoted $al_Q(e)$, is the total rate already allocated to done sessions passing through the edge. Clearly, the capacity constraint implies that $\sum_{i \in \mathcal{A}_Q | e} r_Q(S_i) \leq cap(e) - al_Q(e)$.

3.2 Operations

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` taking as input a session i and a state Q and giving as output rates for i and for all sessions j such that $S_i \cap S_j \neq \emptyset$ so that the resulting rate vector $\mathbf{r}' = \texttt{operation}(i, Q)$ is feasible[†]. Say that `operation` is *conservative* if no rate decreases in the resulting rate vector; else `operation` is *optimistic*.

3.3 Schedulers and Terminators

A *scheduler* [1, 9] is a function Sched that maps a pair $\langle G, \mathcal{S} \rangle$ of a network G and a set of sessions $\mathcal{S}$ laid out on G, a 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 applying operations on sessions.

A *terminator* [1, 9] 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 state Q to a set of sessions $\mathsf{Term}(\langle G, \mathcal{S} \rangle, Q) \subseteq \mathcal{A}_Q$. Intuitively, Term decides the sessions among those still active to be terminated. For sake of simplicity, we

[†]A complete specification of how the required information (for each specific operation) is gathered and disseminated to interfering sessions is usually part of the design and implementation of a distributed algorithm that entails successive applications of the operation. Such specification is beyond the scope of the present article.

[‡]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.

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.

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.

3.4 Executions

We model a computation manipulating the rates and statuses 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) $\mathbf{r}_{Q_l} := \texttt{operation}(i_l, Q_{l-1})$;
- (b) $\mathcal{A}_{Q_{l+1}} = \mathcal{A}_{Q_l} \setminus \mathsf{Term}(\langle G, \mathcal{S} \rangle, Q_l)$.

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 statuses 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.

Two immediate consequences of the definitions of allotted capacity and execution were observed in [9].

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

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

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

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

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 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 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 α.

3.5 Convergence Complexity

Intuitively, an operation counts as 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 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 the number of operations in the execution of Alg on G with $\mathcal{S}$. The *convergence complexity of* Alg is the maximum, over all pairs of a network G and a session set $\mathcal{S}$ (with n sessions) 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 $r(S_i)$; roughly speaking, the priority function allows the determination

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

of a session's rate from its priority[¶]. Since $\mathbf{F}_i$ is strictly increasing, it is invertible; denote $\mathbf{F}_i^{-1}$ its inverse function, which is also strictly increasing. Clearly, the rate $r_Q(S_i)$ of session S_i in state Q uniquely defines the priority $\mathcal{P}_Q(S_i)$ of the session in state Q by way of its inverse priority function. Moreover, 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 statuses) of sessions laid out on the network[‖].

Clearly, rates are "real" network entities; in contrast, however, priorities are only 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 and Adequacy

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. Formally, a *priority max-min fair rate vector* is a feasible 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 [13, 14, 15, 18]. Indeed, a *max-min fair rate vector* [13, 14, 15, 18] is a feasible 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 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

[¶]Increasing monotonocity expresses but the natural requirement that session rates be proportional to priorities. Continuity provides some additional convenience when numerically handling priority functions in particular applications; it is also used as a technical assumption in one of the proofs, as increasing monotonicity and convexity also are.

[‖]We point out that we could have let $\mathbf{F}_i^{-1}$ play the role of the priority function instead of $\mathbf{F}_i$. Although it could have been considered more natural to define priorities as functions of rates rather than vice versa, we chose the opposite because it better fits in our later analysis where we need to calculate the rates from the priorities. Since our priority functions are shown to be invertible, the two formulations are clearly equivalent and our results carry through in both.

[**]We clarify 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.

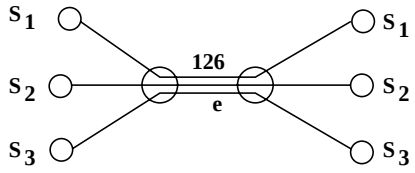

Figure 2: A network consisting of a single edge

vector $\langle 28, 14, 84\rangle$ saturating the edge e. Thus, no rate can be increased without decreasing the rate of some other session; this implies a corresponding decrease to the priority of the session whose rate is decreased. Simple calculations reveal that all priorities are equal to 28. Hence, any such increase must necessarily decrease an equal priority. It follows that $\langle 28, 14, 84\rangle$ 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 $\langle 42, 42, 42\rangle$. This simple example confirms that the more rapidly increasing the priority function of a session is, the larger its rate will be. □

Assume now that for any session S_i, $\mathbf{F}_i(0) \geq 0$. Since $\mathbf{F}_i$ is a strictly increasing function, this implies that $r(S_i) = \mathbf{F}_i(\mathcal{P}(S_i)) \geq \mathbf{F}_i(0) \geq 0$ in any state; that is, we allow each session S_i to specify a non-negative lower bound $\mathbf{F}_i(0)$, called *rate demand,* on its rate. So, the value at zero of each particular priority function determines the lowest possible rate the corresponding session may attain. An *adequate edge* is one whose capacity suffices to accomodate rate demands of all sessions traversing it. The *adequacy assumption* assures that every edge is adequate.

4.3 Priority Share

Take any arbitrary edge e active in state Q. By the capacity constraint and the definition of priority functions, it follows that $\sum_{i \in \mathcal{A}_Q | e} \mathbf{F}_i(\mathbf{F}_i^{-1}(r_Q(S_i))) \leq cap(e) - al_Q(e)$. Insisting that for each $i \in \mathcal{A}_Q \mid e$, $\mathbf{F}_i^{-1}(r_Q(S_i)) = p$ for some real number $p \geq 0$, 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

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

The priority max-min fairness equation for edge e in state Q is an equation in the variable p. Define the *priority share* of edge e in state Q, denoted $PS_Q(e)$, to be a positive root of this equation. Thus, $\sum_{i \in \mathcal{A}_Q | e} \mathbf{F}_i(PS_Q(e)) = cap(e) - al_Q(e)$.

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

The priority share generalizes the notion of the *fair share* $FS_Q(e)$ [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 $\sum_{i \in \mathcal{A}_Q|e} r_Q(S_i) = cap(e) - al_Q(e)$. Insisting that for each session $i \in \mathcal{A}_Q \mid e$, $r_Q(S_i) = r$, for some parameter $r > 0$, allows the saturation equation to admit (trivially) the fair share $FS_Q(e) = \frac{cap(e) - al_Q(e)}{|\mathcal{A}_Q|e|}$ as its *unique* root in this case. In a similar manner, the priority share is a root of the saturation equation once this 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.

Proposition 4.1 *For any edge e and state Q, $PS_Q(e)$ exists uniquely.*

Proof: Since each individual priority function $\mathbf{F}_i$, $i \in \mathcal{A}_Q \mid e$, is continuous, increasing and convex, the sum $\sum_{i \in \mathcal{A}_Q|e} \mathbf{F}_i(p)$ is also 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 $\sum_{i \in \mathcal{A}_Q|e} \mathbf{F}_i(p)$ attains the value $cap(e) - al_Q(e)$ if and only if $cap(e) - al_Q(e) \geq \sum_{i \in \mathcal{A}_Q|e} \mathbf{F}_i(0)$. Hence, by definition of priority share, it suffices to show that the last inequality holds.

If $Q = Q_0$, then, $\sum_{i \in \mathcal{A}_Q|e} \mathbf{F}_i(0) = \sum_{i \in sessions(e)} \mathbf{F}_i(0)$ and $cap(e) - al_Q(e) = cap(e)$, so that the inequality to show is but the adequacy assumption. So, assume that $Q = Q_l$ for some $l > 0$. Clearly,

$$
\begin{array}{lll}
& cap(e) - al_{Q_l}(e) & \\
= & cap(e) - al_{Q_{l-1}}(e) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(S_i) & \text{(by Claim 3.1)} \\
= & cap(e) - al_{Q_{l-1}}(e) - \left(\sum_{i \in \mathcal{A}_{Q_{l-1}}|e} r_{Q_{l-1}}(S_i) - \sum_{i \in \mathcal{A}_{Q_l}|e} r_{Q_{l-1}}(S_i) \right) & \\
= & cap(e) - al_{Q_{l-1}}(e) - \sum_{i \in \mathcal{A}_{Q_{l-1}}|e} r_{Q_{l-1}}(S_i) + \sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) & \\
\geq & cap(e) - al_{Q_{l-1}}(e) - \left(cap(e) - al_{Q_{l-1}}(e)) + \sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) \right) & \text{(by the capacity constraint)} \\
= & \sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(\mathcal{P}_{Q_{l-1}}(S_i)) & \\
\geq & \sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(0)\,, &
\end{array}
$$

as needed. ■

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 α.

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: Clearly,

$$\begin{aligned}
&\sum_{i \in \mathcal{A}_{Q_l} | e} \mathbf{F}_i(PS_{Q_l}(e)) \\
&= \quad cap(e) - al_{Q_l}(e) && \text{(by definition of } PS_{Q_l}(e)) \\
&= \quad cap(e) - al_{Q_{l-1}}(e) - \textstyle\sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_{l-1}}(S_i) && \text{(by Claim 3.1)} \\
&= \quad \textstyle\sum_{i \in \mathcal{A}_{Q_{l-1}} | e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_{l-1}}(S_i) && \text{(by definition of } PS_{Q_{l-1}}(e))\,,
\end{aligned}$$

as needed. ■

We next prove that the saturation of an edge depends critically on how priorities and priority shares of sessions traversing the edge compare to each other.

Lemma 4.3 *For any integer $l_0 \geq 0$, assume that edge e is active in state Q_{l_0}. Then, for each integer $l \geq l_0$, the following hold:*

(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 with (1). Clearly,

$$\begin{aligned}
&\sum_{i \in \mathcal{A}_{Q_l} | e} r_{Q_l}(S_i) \\
&= \quad \textstyle\sum_{i \in \mathcal{A}_{Q_{l_0}} | e} r_{Q_l}(S_i) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i) && \text{(since } \mathcal{A}_{Q_l} \mid e \subseteq \mathcal{A}_{Q_{l_0}} \mid e) \\
&= \quad \textstyle\sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - \sum_{i \in (\mathcal{A}_{Q_{l_0}} \setminus \mathcal{A}_{Q_l}) | e} r_{Q_l}(S_i) \\
&= \quad \textstyle\sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - (al_{Q_l}(e) - al_{Q_{l_0}}(e)) && \text{(by Claim 3.2)} \\
&= \quad \textstyle\sum_{i \in \mathcal{A}_{Q_{l_0}} | e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) - al_{Q_l}(e) + al_{Q_{l_0}}(e) && \text{(since } \mathcal{P}_{Q_l}(S_i) < PS_{Q_{l_0}}(e)) \\
&= \quad cap(e) - al_{Q_{l_0}}(e) - al_{Q_l}(e) + al_{Q_{l_0}}(e) && \text{(by definition of } PS_{Q_{l_0}}(e)) \\
&= \quad cap(e) - al_{Q_l}(e)\,,
\end{aligned}$$

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

Condition (2) is proved in an almost identical way. (The only difference is that now the assumption implies that $\sum_{i\in\mathcal{A}_{Q_{l_0}}|e}\mathbf{F}_i(\mathcal{P}_{Q_l}(S_i))-(al_{Q_l}(e)-al_{Q_{l_0}}(e)) < \sum_{i\in\mathcal{A}_{Q_{l_0}}|e}\mathbf{F}_i(PS_{Q_{l_0}}(e))-al_{Q_l}(e)+al_{Q_{l_0}}(e)$.)

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,

$$
\begin{aligned}
&\sum_{i\in\mathcal{A}_{Q_l}|e} r_{Q_l}(S_i) \\
= \quad & \textstyle\sum_{i\in\mathcal{A}_{Q_{l_0}}|e} r_{Q_l}(S_i) - \sum_{i\in(\mathcal{A}_{Q_{l_0}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_l}(S_i) && \text{(since } \mathcal{A}_{Q_l} \mid e \subseteq \mathcal{A}_{Q_{l_0}} \mid e\text{)} \\
= \quad & \textstyle\sum_{i\in\mathcal{A}_{Q_{l_0}}|e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - \sum_{i\in(\mathcal{A}_{Q_{l_0}}\setminus\mathcal{A}_{Q_l})|e} r_{Q_l}(S_i) \\
= \quad & \textstyle\sum_{i\in\mathcal{A}_{Q_{l_0}}|e} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - (al_{Q_l}(e) - al_{Q_{l_0}}(e)) && \text{(by Claim 3.2)} \\
= \quad & \mathbf{F}_{i_0}(\mathcal{P}_{Q_l}(S_{i_0})) + \textstyle\sum_{i\in\mathcal{A}_{Q_{l_0}}|e,i\neq i_0} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - al_{Q_l}(e) + al_{Q_{l_0}}(e) \\
> \quad & \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e)) + \textstyle\sum_{i\in\mathcal{A}_{Q_{l_0}}|e,i\neq i_0} \mathbf{F}_i(\mathcal{P}_{Q_l}(S_i)) - al_{Q_l}(e) + al_{Q_{l_0}}(e) && \text{(since } \mathcal{P}_{Q_l}(S_{i_0}) > PS_{Q_{l_0}}(e)\text{)} \\
\geq \quad & \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e)) + \textstyle\sum_{i\in\mathcal{A}_{Q_{l_0}}|e,i\neq i_0} \mathbf{F}_i(PS_{Q_{l_0}}(e)) - al_{Q_l}(e) + al_{Q_{l_0}}(e) && \text{(since } \mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)\text{)} \\
= \quad & \textstyle\sum_{i\in\mathcal{A}_{Q_{l_0}}|e} \mathbf{F}_i(PS_{Q_{l_0}}(e)) - al_{Q_l}(e) + al_{Q_{l_0}}(e)) \\
= \quad & cap(e) - al_{Q_{l_0}}(e) - al_{Q_l}(e) + al_{Q_{l_0}}(e) && \text{(by definition of priority share)} \\
= \quad & cap(e) - al_{Q_l}(e)\,,
\end{aligned}
$$

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

4.4 Minimum Priority Share and Minimum Priority Share Edges

For each session $S_i \in \mathcal{S}$, the *minimum priority share* of session S_i in state Q, denoted $MPS_Q(S_i)$, is the least among all priority shares attained in state Q by edges traversed by S_i; thus, $MPS_Q(S_i) = \min_{e\in S_i} PS_Q(e)$. 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 that attains the least priority share among all edges active in Q; thus, $PS_Q(e) = \min_{e'\in AE_Q} PS_Q(e')$. A minimum priority share edge is a generalization of a *minimum fair share edge* [9, 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

For any state Q, a *priority bottleneck edge for* Q is an edge $e \in E$ such that for each session $S \in \mathcal{A}_Q \mid e$, $MPS_Q(S) = PS_Q(e)$; 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* [14] to the setting with priorities.

Example 5.1 Consider the network 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 $al_Q(e_1) = al_Q(e_2) = 0$, while all sessions are active in state Q. The priority max-min fairness equation for edge e_1 in state Q is $\sum_{i \in \mathcal{A}_Q|e_1} \mathbf{F}_i(p) = cap(e_1) - al_Q(e_1)$, 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 implies that $p = 30$, so 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 $MPS_Q(S_3) = 28$. □

An immediate consequence of priority share and priority bottleneck edge is that the priority shares of any two priority bottleneck edges shared by a 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, $PS_Q(e) = PS_Q(e')$.*

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, the claim follows. ∎

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, and (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* [13, 14, 15] 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 $e \in AE_{Q_l}$, $PS_{Q_l}(e) \geq PS_{Q_{l-1}}(e)$.*

Proof: Assume, by way of contradiction, that $PS_{Q_l}(e) < PS_{Q_{l-1}}(e)$. Since for each session i, $\mathbf{F}_i$ is a strictly increasing function, this implies that $\mathbf{F}_i(PS_{Q_l}(e)) < \mathbf{F}_i(PS_{Q_{l-1}}(e))$, so that $\sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_l}(e)) < \sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_{l-1}}(e))$. Now, by Lemma 4.2, $\sum_{i \in \mathcal{A}_{Q_l}|e} \mathbf{F}_i(PS_{Q_l}(e)) = \sum_{i \in \mathcal{A}_{Q_{l-1}}|e} \mathbf{F}_i(PS_{Q_{l-1}}(e)) - \sum_{i \in (\mathcal{A}_{Q_{l-1}} \setminus \mathcal{A}_{Q_l})|e} r_{Q_{l-1}}(S_i)$.

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. ∎

Proposition 5.3 generalizes [1, Lemma 3.4], which established 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 claims hold for any integer l' such that $l_0 \leq l' < l$. For the induction step, we will show that the claims hold for integer l.

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

Lemma 5.5 *For each session $i \in \mathcal{A}_{Q_l} \mid e$, $\mathbf{F}_i(PS_{Q_l}(e)) = \mathbf{F}_i(PS_{Q_{l_0}}(e))$.*

Proof: Assume, by way of contradiction, that there exists some session $i_0 \in \mathcal{A}_{Q_l} \mid e$ such that $\mathbf{F}_{i_0}(PS_{Q_l}(e)) \neq \mathbf{F}_{i_0}(PS_{Q_{l_0}}(e))$. 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 strictly increasing, 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,

$$\sum_{i \in \mathcal{A}_{Q_l} \mid e} \mathbf{F}_i(PS_{Q_l}(e))$$

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

Now, by Proposition 4.2,

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

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

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

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 for (1).

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 $PS_{Q_{l_0}}(e) = \min_{e' \in S_i} PS_{Q_{l_0}}(e')$. 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 $\min_{e' \in S_i} PS_{Q_l}(e') \geq \min_{e' \in S_i} PS_{Q_{l_0}}(e')$. It follows that $\min_{e' \in S_i} PS_{Q_l}(e') \geq PS_{Q_{l_0}}(e)$. By (1), this implies that $\min_{e' \in S_i} PS_{Q_l}(e') \geq PS_{Q_l}(e)$. Clearly, since $e \in S_i$, $\min_{e' \in S_i} PS_{Q_l}(e') \leq PS_{Q_l}(e)$. It follows that $PS_{Q_l}(e) = \min_{e' \in S_i} PS_{Q_l}(e')$. Since i was chosen arbitrarily, this implies that e is a priority bottleneck edge for Q_l, as needed for (2).

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 (1) e' is a priority bottleneck edge for $Q_{l'}$, and (2) $\mathcal{P}_{Q_{l'}}(S_i) = PS_{Q_{l'}}(e')$.

Either $l_0 < l' < l$ or $l' = l$. If $l_0 < l' < l$, then, induction hypothesis (condition (2)) implies that e is a priority bottleneck edge for $Q_{l'}$; if, on the other hand, $l' = l$, then, the already shown condition (2) implies that e is a priority bottleneck edge for $Q_{l'}$. Thus, in either case, e is a priority bottleneck edge for $Q_{l'}$. 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 $\mathcal{P}_{Q_{l'}}(S_i) = PS_{Q_{l'}}(e)$. Since $i \in \mathsf{Term}(Q_{l'})$ and $l \geq l'$, $\mathcal{P}_{Q_{l'}}(S_i) = \mathcal{P}_{Q_l}(S_i)$.

Recall that either $l_0 < l' < l$ or $l' = l$. If $l_0 < l' < l$, then induction hypothesis (condition (1)) implies that $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$; if $l' = l$, then the already shown condition (1) implies that $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$. Thus, in either case, $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$. Hence, it follows that $\mathcal{P}_{Q_l}(S_i) = PS_{Q_{l_0}}(e)$, as needed for (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 (condition (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, the claim follows. ■

6 Termination Properties of Priority Bottleneck Algorithms

In this section, we prove termination properties of priority bottleneck algorithms.

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 $i \in \mathit{sessions}(e)$, $\mathcal{P}_Q(S) \geq \mathcal{P}_Q(S_i)$, and (2) e is saturated in Q.

Example 6.1 Consider again the simple network in Figure 1; we retain all assumptions on priority functions made in Example 5.1. Consider any state Q with rate vector $\mathbf{r}_Q = \langle 28, 14, 84, 10 \rangle$. We will demonstrate the existence of a maximum priority saturated edge for session S_3 in state Q. Clearly, e_1 and e_2 are the only candidates. 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. So, e_2 is not a maximum priority saturated edge for S_3 in 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 $\mathcal{P}_Q(S_3) \geq \mathcal{P}_Q(S_2)$. So, e_1 is a maximum priority saturated edge for S_3 in Q. We encourage the reader to verify that e_1 is also a maximum priority saturated edge for both S_1 and S_2 in Q, and that there is no maximum priority saturated edge for S_4 in Q. □

Our definition of maximum priority saturated edge for a session generalizes Hayden's definition [13] 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, $Q_0 \xrightarrow{\alpha} Q_{l_0} \xrightarrow{\alpha} Q$ such that $S \in \mathsf{Term}(Q_{l_0})$; 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 $e \in E$ traversed by S such that (1) e is a priority bottleneck edge for Q_{l_0}, and (2) $\mathcal{P}_{Q_{l_0}}(S) = PS_{Q_{l_0}}(e)$. 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). Fix any session $i \in sessions(e)$. 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 $\mathcal{A}_Q = \emptyset$, $Q_l \xrightarrow{\alpha} Q$. 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 (condition (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 for (1) in this case.

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 (1) e' is a priority bottleneck edge for $Q_{l'}$, and (2) $\mathcal{P}_{Q_{l'}}(S_i) = PS_{Q_{l'}}(e')$. By definition of a priority bottleneck edge, $PS_{Q_{l'}}(e') = MPS_{Q_{l'}}(S_i)$. Since i traverses e, this implies that $PS_{Q_{l'}}(e') \leq PS_{Q_{l'}}(e)$. Since $S \in \mathcal{A}_{Q_{l_0}} \mid e$, e is active in state Q_{l_0}. Thus, by Proposition 5.3, $PS_{Q_{l'}}(e) \leq PS_{Q_{l_0}}(e)$. Since $i \in \mathsf{Term}(Q_{l'})$ and $Q_{l'} \xrightarrow{\alpha} Q$, $\mathcal{P}_Q(S_i) = \mathcal{P}_{Q_{l'}}(S_i)$. It follows that $\mathcal{P}_Q(S_i) \leq PS_{Q_{l_0}}(e)$. Since $S \in \mathsf{Term}(Q_{l_0})$ and $Q_{l_0} \xrightarrow{\alpha} Q$, $\mathcal{P}_Q(S) = \mathcal{P}_{Q_{l_0}}(S)$. Since $\mathcal{P}_{Q_{l_0}}(S) = PS_{Q_{l_0}}(e)$, it follows that $\mathcal{P}_Q(S_i) \leq \mathcal{P}_Q(S)$, as neededo for (1) in this case.

We continue to show (2). Consider the latest state Q_l in execution α such that e remains active in Q_l. By Proposition 5.4 (condition (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 (condition (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 for (2). ■

Thoerem 6.1 generalizes a classical result of Hayden [13] for max-min fairness to the setting with priorities.

6.2 Output of Priority Bottleneck Algorithms

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

Theorem 6.2 (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 Q. Since e is saturated in Q, while an increase to $r_Q(S_i)$ is possible, there must be at least one session S_j other

than S_i traversing e, whose rate must decrease in order to maintain the capacity constraint for e. Since e is a maximum priority share edge for S_i in Q and S_j traverses e, $\mathcal{P}_Q(S_j) \leq \mathcal{P}_Q(S_i)$. A contradiction. ■

Theorem 6.2 provides a generalization to the setting with priorities of a classical result of Hayden [13] that bottleneck algorithms converge to max-min fair rate vectors (see [3, 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 algorithms are designed in Section 7.4 to exploit these termination properties.

7.1 The Priority Update Operation

Our presentation is influenced by the one in [9, Sections 2.2 & 3.5]. We will introduce the priority update operation, which is a generalization of the *update operation* (see [1, Section 2.1] or [9, 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, called `pupdate`, presented in Figure 3.

We remark that the *update function* `update` from [9, Section 2.2] is the special case of `pupdate` where all priority functions are equal to the identity function.

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 $\langle \mathbf{r}_{\mathcal{A}_Q} \mid e, i, cap(e) - al_Q(e) \rangle$; that is, $\Delta_Q(i, e)$ is the real number $\Delta > 0$ such that $\langle \mathbf{r}', \Delta \rangle = \texttt{pupdate}(\mathbf{r}_{\mathcal{A}_Q} \mid e, i, cap(e) - al_Q(e))$, for some rate vector $\mathbf{r}'$. 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; at the same time, this increase affects neither the active sessions traversing e whose priorities are smaller than the

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. [9, Section 3.5] or [1, Appendix A, Definition A.1]).

Figure 3: The `pupdate` function. 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)).

increased priority of S_i, nor, 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, $\Delta_Q(i) = \min_{e \in S_i} \Delta_Q(i, e)$. So, 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[††].

A *priority update operation* is a set of specific instantiations of the priority update function,

[††]We remark that $\Delta_Q(i)$ can be computed from information that is local to session S_i (cf. [15, Section IV]).

which refer to some particular session index i and state Q, followed by a procedure for the computation of new rates for some sessions. In such priority update operation, each edge e traversed by S_i is associated with a particular instantiation of the priority update function where $\Delta_Q(i,e)$ is computed. Finally, new rates for session S_i and interfering active sessions are computed, using $\Delta_Q(i)$:

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

(2) for each $l \in \mathcal{A}_Q$, $l \neq i$, and $S_l \cap S_i \neq \emptyset$, $r(S_l) := \min\{r(S_l), \mathbf{F}_i(\mathbf{F}_i^{-1}(r(S_i)) + \Delta_Q(i))\}$.

Notice that each instantiation of the priority update function computes an increase $\Delta_Q(i,e)$ that saturates edge e. However, an increase of $\Delta_Q(i)$ for any edge e traversed by S_i is no more than the increase $\Delta_Q(i,e)$ that would saturate it. 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 in Figure 1. We still retain all assumptions on priority functions for this network made in Example 5.1.

Consider the initial state Q_0 and assume that S_1 is scheduled first. Since priorities of all sessions are initially 0, the rate of S_1, and thus its priority, may be increased to a value that saturates e_1 (the only edge traversed by it). The reader can easily verify 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 0 in Q_1. Note also that e_1 is saturated in Q_1.

Assume now that S_2 is scheduled next. Since S_2 traverses e_1 which is saturated in Q_1, any increase to the priority of S_2 imposed by e_1 necessarily causes the priority of S_1 to fall off to a value equal to that of S_2, while e_1 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 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 e_1 remains saturated in Q_2.

Assume finally that S_3 is scheduled next. We ask the reader to verify that the priority increase for S_3 in Q_2 imposed by e_1 is $\Delta_{Q_2}(3,e_1) = 28$, and that the priority increase for S_3 in Q_2 imposed by 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 S_1 and S_2 are also decreased to 28 in Q_3. So, $r_{Q_3}(S_1) = \mathbf{F}_1(\mathcal{P}_{Q_3}(S_1)) = \mathcal{P}_{Q_3}(S_1) = 28$, $r_{Q_3}(S_2) = \mathbf{F}_2(\mathcal{P}_{Q_3}(S_2)) = \mathcal{P}_{Q_3}(S_2)/2 = 28/2 = 14$,

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 e_1 remains saturated in Q_3. □

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. 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

We show that optimistic, priority bottleneck algorithms allow minimum priority share edges to enjoy additional combinatorial properties. 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 $l \geq l_0$, $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$.*

Proposition 7.1 considers any (active) session traversing a minimum priority share edge and establishes that no decrease to its priority below this minimum priority share is possible if, in the first place, its priority is no less than the minimum priority share.

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

If $\mathcal{P}_{Q_l}(S_i) \geq \mathcal{P}_{Q_{l-1}}(S_i)$, then induction hypothesis implies that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$, as needed. So, assume that $\mathcal{P}_{Q_l}(S_i) < \mathcal{P}_{Q_{l-1}}(S_i)$. 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 $\Delta_{Q_{l-1}}(i_l) = \Delta_{Q_{l-1}}(i_l, e')$. We argue that $\mathcal{P}_{Q_l}(i_l) \geq PS_{Q_l}(e')$. 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; moreover, 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 (condition (2)) implies that e' is not saturated in Q_l. A contradiction. So indeed $\mathcal{P}_{Q_l}(i_l) \geq PS_{Q_l}(e')$.

By definition of priority update operation $\mathcal{P}_{Q_l}(S_i) = \mathcal{P}_{Q_l}(i_l)$; since $\mathcal{P}_{Q_l}(i_l) \geq PS_{Q_l}(e')$, it follows 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. ■

Proposition 7.1 provides a generalization of [9, Proposition 4.3] to the setting with priorities. 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 (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 (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 7.2 *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 $l \geq \widehat{l_0} \mid e$, $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$.*

We consider any session traversing a minimum priority share edge; we establish that it will eventually obtain a priority no less than this minimum priority share, once all sessions traversing this minimum priority share edge have been scheduled for an increase at least once.

Proof: We start with an informal outline. We consider the point of the execution following state Q_{l_0} where S_i is scheduled; clearly, that 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 (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 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 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 $Q_{l'}$. Since S_i is active in Q_{l_0}, it follows that S_i is active in $Q_{l'}$ as well. Since S_i traverses e, it follows that e is active in $Q_{l'}$. By Proposition 5.4 (condition (1)), $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$, while by Proposition 5.4 (condition (2)), e is a priority bottleneck edge for $Q_{l'}$. We continue to prove:

Lemma 7.3 $\mathcal{P}_{Q_{l'}}(S_i) \geq PS_{Q_{l_0}}(e)$.

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 $\Delta_{Q_{l'}}(S_i) = \Delta_{Q_{l'}}(S_i, e')$. 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 $k \in \mathcal{A}_{Q_{l'}} \mid e$, $\mathcal{P}_{Q_{l'}}(S_k) \leq \mathcal{P}_{Q_{l'}}(S_i)$, so that $\mathcal{P}_{Q_{l'}}(S_k) < FS_{Q_{l'}}(e')$. It follows by Lemma 4.3 (condition (2)) that e' is not saturated in $Q_{l'}$. 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 7.3, $\mathcal{P}_{Q_{l'}}(S_i) \geq PS_{Q_{l_0}}(e)$. Since $PS_{Q_{l'}}(e) = PS_{Q_{l_0}}(e)$, this implies that $\mathcal{P}_{Q_{l'}}(S_i) \geq PS_{Q_{l'}}(e)$. By Proposition 7.1 (taking l' for l_0), it follows that $\mathcal{P}_{Q_l}(S_i) \geq PS_{Q_{l_0}}(e)$, as needed. ■

Proposition 7.2 provides a generalization of [9, Proposition 4.5] to the setting with priorities.

7.3 Termination Properties of Optimistic, Priority Bottleneck Algorithms

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

Theorem 7.4 *Assume that* Alg *is an optimistic, priority bottleneck algorithm. For any integer* $l_0 \geq 0$*, fix an edge* $e \in MPSE_{Q_{l_0}}$ *with* $\widehat{l_0} \mid e < \infty$*. Then, for any* $S_i \in \mathcal{A}_{Q_{l_0}} \mid e$, $S_i \in \mathcal{D}_{\underset{Q_{\widehat{l_0}|e}}{\longrightarrow}}$.

Proof: We start with an informal outline. We consider any session active in Q_{l_0}; we argue that once all sessions have been scheduled, the session will receive such a rate 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 priority bottleneck, 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 7.5 $\mathcal{P}_{Q_{\widehat{l_0}|e}}(S_i) = PS_{Q_{l_0}}(e)$

Proof: Assume, by way of contradiction, that $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) \neq PS_{Q_{l_0}}(e)$. By Proposition 7.2, $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) \geq PS_{Q_{l_0}}(e)$. It follows that $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) > PS_{Q_{l_0}}(e)$.

If there existed a session $S_j \in \mathcal{A}_{Q_{l_0}} \mid e$ with $j \neq i$, Proposition 7.2 would imply that $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_j) \geq PS_{Q_{l_0}}(e)$; together with $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) > PS_{Q_{l_0}}(e)$, this contradicts Lemma 4.3 (condition (3)). So, there exists no session $S_j \in \mathcal{A}_{Q_{l_0}} \mid e$, $j \neq i$, and $\mathcal{A}_{Q_{l_0}} \mid e = \{i\}$. Denote Q_l the latest state in execution α, with $\overleftarrow{Q_{l_0}} \xrightarrow{\alpha} Q_l \xrightarrow{\alpha} \overrightarrow{Q_{\widehat{l_0|e}}}$ such that $\mathcal{A}_{Q_l} \neq \emptyset$. Thus, $\mathcal{A}_{Q_l} \mid e = \mathcal{A}_{Q_{l_0}} \mid e = \{i\}$ and $al_{Q_l}(e) = al_{Q_{l_0}}(e)$. Clearly,

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

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

We continue with the proof of Theorem 7.4. If $S_i \in \mathcal{D}_{Q_l}$ for some state Q_l in execution α with $Q_{l_0} \xrightarrow{\alpha} Q_l \xrightarrow{\alpha} \overrightarrow{Q_{\widehat{l_0|e}}}$, the definition of execution implies that $S_i \in \mathcal{D}_{\overrightarrow{Q_{\widehat{l_0|e}}}}$, as needed. So, assume that for each state Q_l in execution α with $Q_{l_0} \xrightarrow{\alpha} Q_l \xrightarrow{\alpha} \overrightarrow{Q_{\widehat{l_0|e}}}$, $S_i \in \mathcal{A}_{Q_l}$.

Denote Q_l the latest state in execution α such that both $Q_{l_0} \xrightarrow{\alpha} Q_l \xrightarrow{\alpha} \overrightarrow{Q_{\widehat{l_0|e}}}$ 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 (condition (1)) implies that $PS_{Q_l}(e) = PS_{Q_{l_0}}(e)$, while Proposition 5.4 (condition (2)) implies that e is a priority bottleneck edge for Q_l. By Lemma 7.5, $\mathcal{P}_{Q_{\widehat{l_0|e}}}(S_i) = PS_{Q_{l_0}}(e)$. It follows that $\mathcal{P}_{Q_l}(S_i) = PS_{Q_l}(e)$. In total, e is a priority bottleneck edge for Q_l, traversed by 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. ■

Theorem 7.4 provides a generalization of [9, Theorem 4.7] to the setting with priorities. Our final termination property is a direct consequence of Theorem 7.4. 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 7.6 *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* $S \in \mathcal{D}_{\underset{Q_{\widehat{l_0}}}{\longrightarrow}}$.

Proof: Since $\mathcal{A}_{Q_{l_0}} \neq \emptyset$, $MPSE_{Q_{l_0}} \neq \emptyset$ as well. Fix any edge $e \in MPSE_{Q_{l_0}}$ and consider a session $S \in \mathcal{A}_{Q_{l_0}} \mid e$. Clearly, $S \in \mathcal{A}_{Q_{l_0}}$. By Theorem 7.4, $S \in \mathcal{D}_{\underset{Q_{\widehat{l_0}}}{\longrightarrow}}$, as needed. ■

Theorem 7.6 provides a generalization of [9, Theorem 4.9] to the setting with priorities.

7.4 Specific Optimistic, Priority Bottleneck Algorithms

Say that Sched is *oblivious* [9, Section 3.6.1] if for all pairs $\langle G, \mathcal{S} \rangle$ and Q, and $\langle \widetilde{G}, \widetilde{\mathcal{S}} \rangle$ and $\widetilde{Q}$, and for any integer $l \geq 1$, $\mathsf{Sched}(\langle G, \mathcal{S} \rangle, Q, l) = \mathsf{Sched}(\langle \widetilde{G}, \widetilde{\mathcal{S}} \rangle, \widetilde{Q}, l)$; loosely speaking, an oblivious scheduler uses no knowledge of either the topology of the network, or the rates and statuses (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 $\mathsf{ElevSched} = 1, 2, \ldots, n, n, \ldots, 2, 1, 1, 2, \ldots, n, n, \ldots, 2, 1, \ldots, 1, 2, \ldots, n, n, \ldots, 2, 1, \ldots$ scheduler. Say that Alg is *oblivious* [9, Section 3.6.1] if Sched is; else, say that Alg is *non-oblivious*.

Theorem 7.6 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 7.6 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 6.2 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 7.7 Permutation *computes the priority max-min fair rate vector within* n^2 *priority update operations.*

Permutation generalizes the algorithm RoundRobin [9, Section 7.1] to the setting with priorities.

Theorem 7.4 motivates a non-oblivious algorithm, called PLinear, to compute the priority max-min rate vector. 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, 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 7.4, 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 6.2 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 7.8 PLinear *computes the priority max-min fair rate vector within exactly n priority update operations.*

PLinear is an adaptation of the algorithm Linear [9, Section 7.3] to the setting with priorities; while Linear works with minimum fair share edges, PLinear works with minimum priority 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 [3, 12, 13, 14, 15] to networks with differentiated levels of service. 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 applications to control their rates by purchasing the priority function of their like.

We feel that our work takes a step towards developing an algorithmic framework for currently major trends in network algorithmics, such as flow control and routing, that explicitly takes priority issues into account. For related subsequent work, see, e.g., [4, 19, 21]. As the need for providing prioritized services becomes more widely recognized, algorithmic frameworks that

capture and formalize priority 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," *Journal of Algorithms,* Vol. 30, No. 1, pp. 106–143, January 1999.

[2] S. Bajaj, L. Breslau and S. Shenker, "Is Service Priority Useful in Networks?," *Proceedings of ACM SIGMETRICS,* pp. 66–77, June 1998.

[3] D. P. Bertsekas and R. G. Gallager, *Data Networks,* second edition, Prentice Hall, 1992.

[4] Z. Cao and E. Zegura, "Utility Max-Min: An Application-Oriented Bandwidth Allocation Scheme," *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies,* pp. 793–801, March 1999.

[5] A. Charny, D. D. Clark, and R. Jain, "Congestion Control with Explicit Rate Indication," *Proceedings of the 30th IEEE International Conference on Communications,* pp. 1954–1963, June 1995.

[6] 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.

[7] 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,* pp. 14–26, August 1992.

[8] A. Dailianas and A. Bovopoulos, "Design of Real-Time Admission Control Algorithms with Priority Support," *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies,* pp. 819–826, June 1995.

[9] 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. Full version submitted for journal publication.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] 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,* pp. 143–154, September 1995.

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

[18] 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.

[19] P. Marbach, "Priority Service and Max-Min Fairness," *CD-ROM Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies,* June 2002.

[20] 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.

[21] B. Radunović and J. Le Boudec, "A Unified Framework for Max-Min and Min-Max Faierness with Applications," EPFL TR02_048, Switzerland, July 2002.
(Available at `http://lcawww.epfl.ch/Publications/Radunovic/TR02_048.ps`)

[22] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE Journal on Selected Areas in Communication,* Vol. 13, No. 7, pp. 1176–1188, September 1995.

[23] S. Suri, D. Tipper and G. Meempat, "A Comparative Evaluation of Space Priority Strategies in ATM Networks," *Proceedings of the 13th Annual Joint Conference of the IEEE Computer and Communications Societies,* pp. 516–523, June 1994.

[24] Y. Takagi, S. Hino and T. Takahashi, "Priority Assignment Control of ATM Line Buffers with Multiple QoS Classes," *IEEE Journal on Selected Areas in Communications,* Vol. 9, No. 7, pp. 1078–1092, 1991.

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

[26] 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, September 1993.