Convergence Complexity of Optimistic Rate Based Flow Control Algorithms*

Yehuda Afek[†] Yishay Mansour[‡] Zvi Ostfeld[§]
Computer Science Department,
Tel-Aviv University, Israel 69978.

December 12, 1997

Abstract

This paper studies basic properties of rate based flow-control algorithms and of the max-min fairness criteria. For the algorithms we suggest a new approach for their modeling and analysis, which may be considered more "optimistic" and realistic than traditional approaches. Three variations of the approach are presented and their rate of convergence to the optimal max-min fairness solution is analyzed. In addition, we introduce and analyze approximate rate based flow control algorithms. We show that under certain conditions the approximate algorithms may converge faster. However, we show that the resulting flows may be substantially different than the flows dictated by the max-min fairness. We further demonstrate that the max-min fairness solution can be very sensitive to small changes, i.e., there are configurations in which an addition or deletion of a session with rate δ may change the allocation of another session by $\Omega(\delta \cdot 2^{\frac{n}{2}})$, but by no more than $O(\delta \cdot 2^n)$. This implies that the max-min fairness criteria may provide a bad estimate of how far a set of flow allocations is from the optimal allocation.

1 Introduction

The ATM Forum on Traffic Management has adopted rate based flow control [Cha94, Jaf81, Hay81, Mos84, GB84, Gaf82, BG87] as the basis for flow control in its networks for ABR (Available Bit Rate) traffic (see [BF95] for example). The major arguments for rate based flow control, is the simplicity and modest hardware requirements per virtual circuit, compared with those of the credit based scheme [KM95]. Conceptually, the rate based flow control adjusts transmission rates of different sessions (virtual circuits) in the network in an end-to-end manner

*A preliminary version of this paper was presented in the 28^{th} Annual ACM Symposium on Theory of Computing, (STOC, Philadelphia, PA, 1996,) 89-98.

[†]Telephone: +972-3-6408952,Fax: +972-3-6409357, e-mail: afek@math.tau.ac.il

[‡]Telephone: +972-3-6408829,Fax: +972-3-6409357, e-mail: mansour@math.tau.ac.il

[§]Telephone: +972-3-6407439,Fax: +972-3-6409357, e-mail: ostfeld@math.tau.ac.il

aiming to achieve a good utilization of the network while maintaining fairness between the sessions.

Most implementations of rate based flow control work as follows: A control message (Resource Management cell, RM) is looping around the path of each session.[1] On its way the control message calculates the minimum share the session may take from the excess capacity of the links along the path. Roughly, the share a session may take from a link is the excess capacity of the link divided by the number of sessions that may still increase their transmission rate and share the excess. The rate of the session is then increased by this minimum (see [Cha94]).

A key point in these algorithms is the computation in each step, by how much each session may be increased. This computation essentially determines how the network resources (link capacities) are divided among the different virtual circuits. The basic principle guiding this computation is the desire to share the link capacities in a fair way among the different sessions. The *max-min* fairness criterion [BG87, Jaf81, GB84, Gaf82, Hay81, Mos84, Cha94] is widely accepted as the theoretical criteria to guide this computation. A set of sessions is said to be in the state of *max-min fairness* if it is impossible to infinitesimally increase the rate of any session without decreasing the rate of some sessions whose rate is equal or smaller. It can be shown that there is exactly one unique set of flow values that achieves the max-min fairness for a set of sessions with requirements in a given network. The max-min criterion is considered fair because no session gets a larger flow on the account of a session with a smaller or equal flow.[2]

Given a network with link capacities and a set of sessions with transmission requirements, there is a simple iterative algorithm to compute the set of transmission rates that correspond to the max-min fairness rates [BG87]: In each iteration fix the rates of sessions that use the "most congested" link (i.e., the link which restricts a minimum rate for sessions that have not yet been fixed with their "final" rate). In the next iteration the rates that have been fixed are subtracted from the link capacities, and the process is iterated. (Thus, we are maximizing the minimum session and continuing recursively on the remaining sessions, after updating the capacities.) While this algorithm seem to be efficient, it is impractical since in each iteration the new globally most congested link has to be selected.

Most theoretical rate based flow control algorithms that guarantee to converge to the state of *max-min* fairness adjust the rates of sessions in a conservative way. An algorithm is considered to be conservative if it never overshoots. That is, given a fixed set of sessions, it converges to the max-min rates without ever assigning a session a rate that is larger than its final rate. In other words, the rates that are assigned during the computation may only increase, until the final value is reached [BG87, GB84, Cha94].

One problem with most of the conservative algorithms is that they do not include a decrease operation that reduces the rate of sessions (with the exception of [Cha94]). This does not model the situation of real networks that have to be able to decrease the rate of some sessions (e.g., after a new session has been added). Furthermore, the network has some current set of rates, which should be adjusted, and it would be preferable to perform an incremental update rather than starting each time from scratch.

In this paper we examine a more aggressive approach, which we term "optimistic". According

[1] Note that according to the ATM concept each session has a fixed path. Hence, the flow control mechanisms addressed in this paper have nothing to do with the issues of routing.

[2] The problem of finding the *max-min fairness* can be expressed as a set of linear inequalities, as presented in [Hay81].

to this approach, during the computation, sessions may be given a rate that is larger than their final rate. Hence, during the convergence process rates may go up and down. This approach is closer to the proposed rate based approach of the ATM Forum [Rob94, ST94, JKV94].

We choose a simple abstraction for the update operations. The update operation is performed atomically on a session. This means that an update operation is not performed until the previous one has completed. Another assumption is that the switch has full knowledge of the current rates of all the sessions crossing it. This model is presented for the purpose of analysis and is not expected to be implemented. However, we believe that this model captures the essential properties of the problem as faced by practical distributed rate based algorithms, such as [Rob94, ST94, JKV94].

The optimistic approach is close to the approach used by Roberts' algorithm ([Rob94]). However, since in our model the update operation is atomic and the switch knows the rates of all sessions that cross it, the sum of the rates of the sessions that use a link may never exceed the link capacity. In Roberts' algorithm the switch has only partial knowledge about the rate of the sessions that cross it and since link updates are not atomic an output link might be overloaded.

Our model is somewhat similar to that of Charny [Cha94] where the switch knows the rates of all sessions that cross it. However, Charny uses a conservative approach and does not assume atomic updates. Furthermore, the convergence complexity in ([Cha94]) is measured in time while in this paper the complexity measure is abstracted by the number of update operations.

Each of the algorithms in this paper is considered as a sequence of atomic operations, called *update operations* that are applied on the network. Each update operation increases the rate of one session and decreases the rate of a subset of sessions that share an edge with it. An update operation is performed on session S_i only if its rate may be increased without decreasing the rate of any other session, below the new final rate of S_i. The magnitude of increase is the maximum that satisfies the above condition. The main difference between the different algorithms is the scheduler that in each network state decides which session to increase next. For example, if a link with capacity 19 is shared by five sessions with rates $1, 2, 3, 6$, and 7, and the update operation is applied to the second session (whose rate is 2), then the resulting rates are: $1, 5, 3, 5$, and 5.

Two parameters distinguish between the different algorithms considered here. One is the scheduler that in each state selects the next session to be considered for an increase, and the second is the rule by which the algorithm decides whether to perform the update or not. The complexity measure used in analyzing the different algorithms is the total number of update operations applied until the algorithm quiesces (stops to change rates), which we call the convergence complexity.

For the first mechanism, the scheduler, we mark sessions that have reached their final rate as *done*. Sessions that have not yet reached their final rate are considered *active*. We consider and analyze three different schedulers: (1) A *global min* scheduler in which the next update operation is performed using an *active* session with the global smallest rate, (2) A *local min* scheduler in which the next update operation is performed using an *active* session whose rate is smallest among all the *active* sessions that share a link with it, and (3) An *arbitrary* scheduler that arbitrarily selects the next session to be increased. Note that the *local min* is more appropriate for a distributed environment than the *global min*, and the *arbitrary* is even more "distributed" than both. In discussing the algorithms in this order we go from the easier (to analyze) to the more difficult.

For the update decision we consider two variants: (1) a selected session is increased as long as its rate may be increased, and (2) approximate algorithms, in which the rate of a session is increased only if the increase is by more than δ. The approximate algorithms reach quiescence when no session may be increased by more than δ.

The main results presented in this paper are:

1. In both the global and the local minimum algorithms $\Theta(n^2)$ update operations are both necessary and sufficient to reach the max-min fairness state, where n is the total number of sessions in the network (i.e., it is sufficient for all scenarios, and there are scenarios for which it is necessary).

2. In the arbitrary algorithm $O(\Phi^n)$ update operations are shown to be sufficient to reach the max-min fairness state (where $\Phi = \frac{\sqrt{5}+1}{2}$). An example is given in which the adversary may force $\Omega(2^{\frac{n}{2}})$ update operations.

3. Any approximate algorithm (any algorithm that refrains from updates that are smaller than or equal to δ), quiesces at a state in which the rate of every session may be smaller or larger than its rate in the optimal max-min fairness allocation by at most $O(\delta \cdot 2^n)$.

4. An example of a configuration in which each session can not be increased by more than δ, yet some session is $\Omega(\delta \cdot 2^{\frac{n}{2}})$ away from its allocation in the max-min vector is provided. This is a lower bound on the possible gap between the rates computed by any approximate algorithm and the rates in the max-min fairness allocation. Although this property is exhibited only under peculiar conditions, it is shared by any algorithm that refrains from small updates. Furthermore, this lower bound phenomena is possible also if the rate of only one session can be increased by δ, i.e., if the rate of no other session may be increased. However, this phenomena does not indicate a disadvantage of the approximate algorithms, since in particular, the removal or addition of a session with a small rate to the network could have the same effect. Hence, this example mainly suggests that the state of *max-min fairness* may be unstable in the sense that small fluctuations in the rate of one session may cause dramatic changes in the rate of other sessions.

5. If the number of sessions that share an edge is bounded by a constant, then the approximate algorithm under either the global minimum scheduler or the enhanced local minimum scheduler (a slight modification of the local minimum scheduler) reaches quiescence in $O(\min(n^2, n \cdot \log \frac{W}{\delta}))$ update operations, where W is at most the maximum capacity of an edge in the network (a tighter bound on W is given in Section 7). Note that these improved bounds for the approximate algorithms hold for the schedulers that are based upon a minimum concept (either local or global). For arbitrary schedulers, it can be shown that the example that gives the lower bound of $\Omega(2^{\frac{n}{2}})$ on the number of steps still holds regardless of the value of δ.

The results mentioned in 3 and 4 answer an open question raised by Charny [Cha94] about the convergence of rate based flow control algorithms if the rates are restricted to discrete values.

Notice that the goal of this paper is to analyze the behavior of flow control in ATM networks. Hence, we do not consider specific implementations of the above models (in compliance with the ATM standards) and are contended by the claim that the model reflects the behavior of ATM networks. The results presented in this paper provide a deeper understanding of the behavior of fair rate-based flow control algorithms. One conclusion is that it is beneficial to update the sessions with the lower rates before updating the sessions with the larger rates. The lower bound examples point to the hopefully rare cases that one should watch out for, in which either the convergence to the max-min state is very slow, or the possibility in which a small gap or a small change in the rate of one session may cause a large gap in the rates of other sessions.

We assumed throughout this work that every session is greedy and would like to consume as much rate as allowed by the network. In case there is a permanent self restriction of a source - (Peak Cell Rate in terms of ATM) it is done by adding a single edge that connects the source to a switch, in which the capacity is the restricted bandwidth.

The main issue addressed here is the convergence complexity, i.e., the number of update operations necessary to reach the final state. Therefore, we assume that sessions do not change their requirements during the execution of the algorithm and all session rates are initially zero. However, none of the algorithms and bounds discussed need an all zero initial state to correctly operate, they may start from any (legal) initial set of flows.

The model, general definitions and notations are given in Section 2. In Section 3 basic properties that are common to all of our algorithms are presented. The *global minimum* algorithm is analyzed in Section 4, the *local minimum* algorithm is analyzed in Section 5 and the *arbitrary* algorithm is analyzed in Section 6. Approximate algorithms are introduced and analyzed in Section 7.

2 General Definitions and Notations

We model a network as a directed graph $G = (V, E)$, where each edge $e \in E$ is a link in the network. Each edge $e \in E$, has a certain capacity denoted by $Cap(e)$.

A set of n sessions $\{S_1, \ldots, S_n\}$ is laid out in the network, where session S_i is a set of links that constitutes a simple path in G between a source and a destination. The bandwidth allocated to session S_i is denoted by $rate(S_i)$. For every edge $e \in E$ we define $\gamma(e)$ to be the set of sessions that use e, i.e., $\gamma(e) = \{S_i | e \in S_i\}$. The capacity constraint requires that for every edge $e \in E$ the sum of the rates of the sessions that share it is at most the edge capacity, i.e., $\sum_{S_i \in \gamma(e)} rate(S_i) \leq Cap(e)$. If all the capacity constraints are satisfied then the allocation is a *feasible allocation*. A feasible allocation is maximal if no session can increase its rate without decreasing the rate of any other session. A fair allocation is a feasible max-min allocation if in order to increase the rate of any session one needs to decrease the rate of sessions with lower or equal rates.

We now give a formal definition for the term *fair flow* (similar definitions may be found in other works [Cha94, Jaf81, Hay81, Mos84, GB84, Gaf82, BG87]). A vector $x = (x^1, x^2, \ldots, x^n) \in R^n$ is said to be *lexicographically* greater than $y = (y^1, y^2, \ldots, y^n) \in R^n$ if $\exists\, k, 1 \leq k \leq n$ such that $x^i = y^i$, for $i < k$, and $x^k > y^k$, in which case we denote this by $x >_{lex} y$.

For a vector $x = (x^1, x^2, \ldots, x^n) \in R^n$ we define $sort(x) = (x^{i_1}, x^{i_2}, \ldots, x^{i_n}) \in R^n$ to be the elements of x arranged in a non decreasing order.

An allocation vector x (in which $x^i = rate(S_i)$) is called *max-min vector* if it is a feasible allocation and is lexicographically maximum among all feasible allocation vectors, with respect to the sort order (i.e., for every feasible allocation y, we have that $sort(x) \geq_{lex} sort(y)$). Note that in the max-min vector allocation each session has an edge on which it has a rate equal to the maximum rate that passes through that edge.

The goal of our algorithms is to compute a max-min vector x, which constitutes a fair allocation.

2.1 Algorithms' Progress

Our algorithms start at some initial allocation of rates and converge to a final rate allocation. In the processes the algorithms mark the sessions that reached their final rate as *done*, while the other sessions are marked *active*. We denote by $\mathcal{A}$ the set of *active* sessions and its complement, the *done* sessions, by $\mathcal{D}$. When our algorithms start all the sessions are *active* (i.e., $\mathcal{D} = \emptyset$). The algorithms quiesces when all the sessions are *done* ($\mathcal{A} = \emptyset$).

Each execution of an algorithm is modeled as a sequence $Q_0 o_0 Q_1 o_1 \ldots Q_i o_i Q_{i+1} \ldots$ where, Q_i represents a *state* of the network at a specific point in time and o_i an *update operation*. The procedure that selects the next session to be increased is called the scheduler. The *active* and *done* markings are used by the scheduler in order to decide which session may be a candidate for an increase in the next operation. However, this marking is not used by the computation of the new rate values. We denote by $S(o_i)$ the session that was selected for an increase in operation o_i. The variables in state Q_i are denoted by superscript i, e.g., $rate(S_j)^i$ is the rate of session S_j in state Q_i.

Each algorithm is described as a sequence of operations, called *update operations* that modify the rates of sessions in the network. In each update operation o_i the rate of one session, S_j, is increased and the rate of other sessions that share an edge with S_j may be decreased.

An update operation o_i is performed on session S_j only if there is a way to increase $rate(S_j)^i$ to a new value, $rate(S_j)^{i+1} = \eta > rate(S_j)^i$, while the rate of any other session $S_l \neq S_j$ is either unaffected or decreased to a new value which is greater or equal to η. Given a vector of rates x, the outputs of the update operation on S_j will result in a maximal increase in the rate of S_j without violating the above condition. The possible increase in $rate(S_j)^i$ is $\Delta^i(j)$, i.e., if $rate(S_j)$ is increased by o_i then $rate(S_j)^{i+1} = rate(S_j)^i + \Delta^i(j) = \eta$. (A formal specification of the update operation is given in Appendix A.)

Example: consider $x = (1, 2, 3, 4, 6)$, and assume that all the sessions share a single edge e whose capacity is 16. First note that x is maximal. Session S_5, with rate 6 can not be updated in this configuration. Session S_4 can be updated, resulting in the following rate allocation $(1, 2, 3, 5, 5)$. Session S_1, S_2 and S_3 can also be updated, e.g., the result of updating S_2 is $(1, 4, 3, 4, 4)$.

The above example was for the simple case of a single edge. In general a session may have several edges, and it may be increased only if it can be increased on *every* edge. For each edge e through which session S_j passes we define $\Delta^i(e, j)$ to be the maximum amount by which $rate(S_j)$ may be increased if edge e is the only edge constraining S_j (i.e., as if e is the whole network). The possible update, $\Delta^i(j)$, by definition, is the minimum over all the increases possible by the different edges of S_j, i.e., $\Delta^i(j) = \min_{e \in S_j}\{\Delta^i(e, j)\}$. Note that any session that shares an edge with the increased session and whose rate was larger than η may be reduced in a fair manner to a value that is larger or equal to η. For example consider Figure 1 where $x = (40, 45, 10, 80)$ (note that x is not maximal). The result of an update operation on S_3 would be the rate allocation $(35, 35, 35, 75)$.

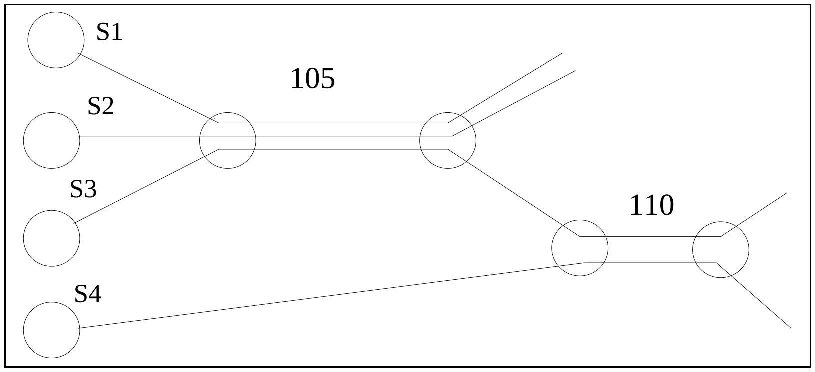

Figure 1: An example for an update operation

For each edge e, $N^i(e)$ is the number of *active* sessions that use e in state Q_i, i.e. $N^i(e) = |\mathcal{A}^i \cap \gamma(e)|$.

We define the *allotted* capacity of an edge e at state Q_i as the total capacity already allocated to *done* sessions, i.e.,

$$allotted^i(e) = \sum_{S_j \in \{\mathcal{D}^i \cap \gamma(e)\}} rate(S_j)^i.$$

Intuitively, the available capacity of an edge is the capacity that can be still divided between the *active* sessions. This leads to the following definition of a *fair share* of an edge in state Q_i:

$$FS^i(e) = \frac{Cap(e) - allotted^i(e)}{N^i(e)}.$$

3 Basic properties of Bottleneck algorithms

In this section we define bottleneck algorithms and prove several basic properties on them.

In an algorithm that computes the exact max-min vector an edge e is called *bottleneck edge* in state Q_i if for every *active* session $S_j \in \gamma(e) \cap \mathcal{A}^i$, $FS^i(e)$ is the smallest fair share among the edges along S_j [BG87].

Definition 3.1 *A* general bottleneck algorithm *is an algorithm that computes the exact max-min vector for a set of sessions, and in any state Q_i in its execution, a session S is marked* done *(i.e., has reached its final rate) only if there is a bottleneck edge $e \in S$ in Q_i and $rate(S)^i = FS^i(e)$.*

In some algorithms (e.g. [BG87, Cha94]) that compute the max-min vector, when a session has reached its final value (i.e., marked *done*) then in addition to the above condition, all the *active* sessions that use a bottleneck edge have the same rates. (That is, they share the edge available capacity in a fair manner.) When all the sessions satisfy this condition (i.e., marked *done*) the algorithm terminates.

All the algorithms presented in this paper except for those in Section 7 are called *bottleneck algorithms.*

Definition 3.2 *A* bottleneck algorithm *is a* general bottleneck algorithm *that in addition, uses the update operation, as described in Section 2.*

The main point that distinguishes our algorithms from the previous ones is that in transient states, when the max-min condition has not yet been reached, some sessions may have a rate that is larger than their rate in the max-min vector.

In this section we present and prove some basic general lemmas concerning bottleneck algorithms. These basic lemmas are shared by the different algorithms considered and are used in the sequel for their proof and analysis.

The following theorem shows the correctness of a bottleneck algorithm by stating that when it terminates (i.e., all sessions are marked *done*) then $(rate(S_1), rate(S_2), \ldots, rate(S_n))$ is the max-min vector.

Theorem 3.3 *[BG87] In a bottleneck algorithm, if $\mathcal{A}^i = \emptyset$ then the rate allocation is the max-min vector.* □

The next lemma shows that the *fair share* of an edge is monotonically non decreasing function of time.

Lemma 3.4 *In any bottleneck algorithm the fair share $FS(e)$ of edge e may not decrease as the algorithm progresses, that is, for $i \geq 0$, if there are active sessions that use e in state Q_{i+1} (i.e., $N^{i+1}(e) > 0$), then $FS^{i+1}(e) \geq FS^i(e)$.*

Proof: If in operation o_i, a number of sessions that use edge e change their rate, but none becomes *done*, then clearly the fair share of e, $FS(e)$ remains unaffected. The only way $FS(e)$ is affected is if some sessions that use e become *done* in the last step. This increases $allotted(e)$ and decreases $N(e)$. If the rate of all the *done* sessions equals $FS^i(e)$ then it does not effect the Fair Share. Otherwise, some of the *done* sessions must get less than $FS^i(e)$ and some of the *done* sessions must get exactly $FS^i(e)$. Hence, the remaining active sessions may get more, thus increasing the Fair Share. Formally: If $N^{i+1}(e) = N^i(e)$ then, by definition $FS^{i+1}(e) = FS^i(e)$. Otherwise, $N^{i+1}(e) = N^i(e) - k$ where $N^i(e) > k \geq 1$, and in state Q_i there are k sessions, $\{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\} \subsetneq \{\mathcal{A}^i \cap \gamma(e)\}$ with edges $e'_1 \in S_{j_1}, e'_2 \in S_{j_2}, \ldots, e'_k \in S_{j_k}$ such that for every l, where $k \geq l \geq 1$ $rate(S_{j_l})^i = FS^i(e'_l) \leq FS^i(e)$. Then, by the definition of *Fair Share*, it follows that:

$$
\begin{aligned}
FS^{i+1}(e) &= \frac{Cap(e) - allotted^{i+1}(e)}{N^{i+1}(e)} \\
&= \frac{Cap(e) - allotted^i(e) - \sum_{l=1}^{k} rate(S_{j_l})^i}{N^i(e) - k} \\
&\geq \frac{Cap(e) - allotted^i(e) - k \cdot FS^i(e)}{N^i(e) - k} = FS^i(e)
\end{aligned}
$$

□

The following lemma states that after increasing the rate of a session via an update operation, its rate is larger or equal to the smallest Fair Share along its path.

Lemma 3.5 *Consider an operation o_i in which the rate of session $S(o_i) = S_j \in \mathcal{A}^i$ is increased, then in the succeeding state, Q_{i+1}, there is at least one edge $e \in S_j$ such that,*

1. *e is saturated (i.e., $\sum_{S_l \in \gamma(e)} rate(S_l)^{i+1} = Cap(e)$).*

2. *$rate(S_j)^{i+1} \geq FS^i(e)$.*

Proof: Let e be the edge according to which the amount of increase was determined, that is, the edge that enabled the smallest increase in the rate of S_j (i.e. $\Delta^i(j) = \Delta^i(e,j)$ as defined in Appendix A). Following Definition A.1 and Definition A.2, after the increase, e is saturated and $rate(S_j)^{i+1}$ is the maximum rate among all sessions in $\gamma(e)$, which completes the proof. □

Definition 3.6 *Let $S_j \in \mathcal{A}^i$. We define ${LB_j}^i$ to be the smallest Fair Share along the path of S_j in state Q_i, formally ${LB_j}^i = \min_{e \in S_j}\{FS^i(e)\}$.*

Our goal now is to prove Theorem 3.10 in which we state that if the rate of an arbitrary session S_j is adjusted in state Q_i (either increased or decreased) then the rate of this session will never be smaller than ${LB_j}^i$. This is proved in two steps. First, in Lemma 3.7, we prove that if the rate of session S_j is increased in state Q_i or afterwards, then in the subsequent state its rate is larger than ${LB_j}^i$. Second in Corollary 3.9, we show that no decrease in the rate of S_j subsequent to state Q_i may decrease it below ${LB_j}^i$.

Lemma 3.7 *Consider state Q_i of a bottleneck algorithm and let e be an edge along the path of $S_j \in \mathcal{A}^i$ where $FS^i(e) = {LB_j}^i$. Let $l \geq i$ where $S(o_l) = S_j$. Then, $rate(S_j)^{l+1} \geq FS^i(e)$.*

Proof: Let e' be an edge along the path of S_j such that $FS^l(e') = {LB_j}^l$. Hence, $rate(S_j)^{l+1} \geq FS^l(e') \geq FS^i(e') \geq FS^i(e)$ (the first inequality follows from Lemma 3.5, the second inequality follows from Lemma 3.4 and the third inequality follows from the fact that $FS^i(e)$ is the minimum *Fair Share* among all edges of S_j in state Q_i). □

The next lemma shows that if a session's rate is decreased, then it is still above the *Fair Share* of at least one of its edges. The intuition for this lemma is that a decrease of a session is done according to a constraint caused by an edge which is common to this session and to the session that was chosen to be increased. Therefore, the updated rate of both the increased session and the decreased session will be at least the *Fair Share* of that edge.

Lemma 3.8 *Consider state Q_i of an arbitrary bottleneck algorithm for computing the max-min vector of rates. If $rate(S_j)^{i+1} < rate(S_j)^i$ (i.e., S_j is decreased in o_i) then $rate(S_j)^{i+1} \geq {LB_j}^i$.*

Proof: Let $S(o_i) = S_l$ be an arbitrary session that is increased. Following Definition A.2, one can see that $rate(S_j)^i$ is decreased only if there exists an edge e' such that $\{S_l, S_j\} \subseteq \gamma(e')$, e' is saturated in state Q_{i+1} and $rate(S_j)^{i+1}$ is maximum in Q_{i+1} among all sessions of $\gamma(e')$. Hence, $rate(S_j)^{i+1} \geq FS^i(e') \geq {LB_j}^i$. □

Corollary 3.9 *Consider states Q_i and Q_l of an arbitrary bottleneck algorithm for computing the max-min vector of rates, where $l \geq i$. If $rate(S_j)^{l+1} < rate(S_j)^l$ (i.e., S_j is decreased in o_l) then $rate(S_j)^{l+1} \geq {LB_j}^i$.*

Proof: Follows immediately from Lemma 3.8 and the fact that ${LB_j}^l \geq {LB_j}^i$ □

Theorem 3.10 *Consider state Q_i of a bottleneck algorithm where either $S(o_i) = S_j$ or $rate(S_j)^{i+1} < rate(S_j)^i$. Then, for all $l \geq i$ we have $rate(S_j)^l \geq {LB_j}^i$.* □

4 Global Min Scheduler

In this section we analyze a bottleneck algorithm in which the scheduler selects in each state the globally smallest *active* session for an increase. The following rules are used in each state of the algorithm:

Marking a session done: The smallest rate *active* session in the network is marked *done* if it cannot be increased.

Session selected for increase: In state Q_i, $S(o_i) = S_j$ such that $rate(S_j)^i = \min_{t \in \mathcal{A}^i} rate(S_t)^i$.

Termination: The algorithm terminates when $\mathcal{A}^i = \emptyset$.

This technique is quite intuitive and indeed it turns out that the algorithm converges in $\Theta(n^2)$ update operations. The upper bound is proved in Theorem 4.3 and the lower bound is proved in Theorem 4.4. Note that this technique is used in [BG87] for a different model where an upper bound analogous to that in Theorem 4.3 is proved.

Observation 4.1 *The Global Min algorithm is a bottleneck algorithm.*

Proof: We have to show compatibility to Definition 3.2. Since the algorithm confirm to the conventions of Section 2 we just have to show that it is a general bottleneck algorithm. Let $S_j \in \mathcal{A}^i$ be a session where $rate(S_j)^i$ is global minimum and can not be increased. Since $rate(S_j)^i$ is minimum among all sessions in $\mathcal{A}^i$ then in particular, for every edge $e \in S_j$ $rate(S_j)^i \leq rate(S_m)^i$ for all sessions $S_m \in \{\mathcal{A}^i \cap \gamma(e)\}$ and hence $rate(S_j)^i \leq FS^i(e)$ ($rate(S_j)^i > FS^i(e)$ implies that the amount of flow in the edge exceeds its capacity).

Since $rate(S_j)^i$ can not be increased, then there is an edge $e' \in S_j$ where $\Delta^i(e', j) = 0$ which implies that e' is saturated and for all $S_m \in \{\mathcal{A}^i \cap \gamma(e')\}$, $rate(S_j)^i \geq rate(S_m)^i$.

Hence, for all sessions $S_m \in \{\mathcal{A}^i \cap \gamma(e')\}$, $rate(S_j)^i = rate(S_m)^i = FS^i(e')$.

Hence, also $rate(S_m)^i$ is minimum among all sessions in $\mathcal{A}^i$ which implies that for any edge $e'' \in S_m$, $FS^i(e'') \geq rate(S_m)^i = FS^i(e')$. □

The following lemma shows that if a session rate becomes global minimum in two different states Q_l and Q_i then at least one *active* session becomes *done* between those two states.

Lemma 4.2 *In the Global Min algorithm, if for $l > i$, $S(o_l) = S(o_i) = S_j$ then $\mathcal{A}^l \subsetneq \mathcal{A}^i$ (i.e., $\mathcal{A}^l$ is a proper subset of $\mathcal{A}^i$).*

Proof: Let $e \in S_j$ such that $FS^i(e) = LB_j{}^i$. By Theorem 3.10, for all $k > i$, $rate(S_j)^k \geq FS^i(e)$. Hence, if $\mathcal{A}^l = \mathcal{A}^i$ then $FS^l(e) = FS^i(e)$ which implies that $rate(S_j)^l \geq FS^l(e)$. If $rate(S_j)^l$ is global minimum of all sessions in $\mathcal{A}^l$ it is in particular smallest in all sessions of $\mathcal{A}^l \cap \gamma(e)$ which implies that the rate of all sessions in $\mathcal{A}^l \cap \gamma(e)$ is larger than or equal to $FS^l(e)$. Following the definition of $FS^l(e)$ this might happen only if the rate of all sessions in $\mathcal{A}^l \cap \gamma(e)$ is exactly $FS^l(e)$. However, in such a case the rate of all the sessions in $\mathcal{A}^l \cap \gamma(e)$ (and in particular, the rate of session S_j) can not be increased is state Q_l and they should be transfered from $\mathcal{A}$ to $\mathcal{D}$ before the execution of o_l, completing the proof. □

Theorem 4.3 *The number of update operations in any run of the Global Min algorithm is $O(n^2)$.*

Proof: First notice that as long as $\mathcal{A}^i \neq \emptyset$ the algorithm progresses since in state Q_i every *active* session in which the rate is global minimum can be either increased or marked done. Following Lemma 4.2 the size of the active set must decrease between every pair of updates to the same session; this immediately implies an upper bound of
$n + (n-1) + (n-2) + \ldots + 1 = O(n^2)$. □

The next theorem establishes a tight lower bound for this algorithm.

Theorem 4.4 *There is an infinite family of networks with n sessions in which the number of update operations in a run of the Global Min algorithm is $\Omega(n^2)$.*

Proof: We construct recursively a network denoted $Net(m, bottom, potential)$. The network has $2m$ sessions, where *bottom* is the initial rate for each of the sessions in the network (and a lower bound for the rate it can achieve in the future) and *bottom* + *potential* is an upper bound for the amount of flow (either temporary or final) that any session can achieve in the future. Figure 2 is an example of a network $Net(4, 0, 128)$.

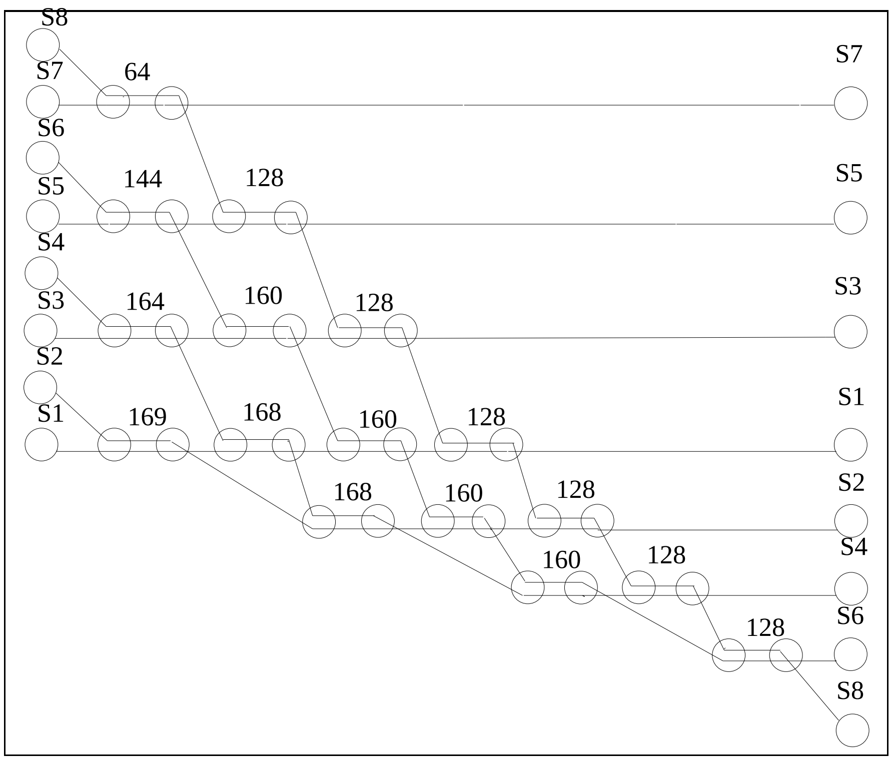

Figure 2: An example of a network $Net(4, 0, 128)$

The procedure that builds that network is as follows:

Procedure Net$(m, bottom, potential)$:

1. If $m = 1$ then S_1 and S_2 share a common edge e such that $Cap(e) = 2 \cdot bottom + potential$.

2. If $m > 1$ then

   (a) S_{2m-1} and S_{2m} share a common edge e such that $Cap(e) = 2 \cdot bottom + \frac{potential}{2}$.

   (b) construct configuration $\mathbf{Net}(m-1, bottom + \frac{potential}{2}, \frac{potential}{4})$.

   (c) Each of the sessions $S_1, S_2, \ldots, S_{2(m-1)}$ share an edge e with S_{2m} where $Cap(e) = 2 \cdot bottom + potential$.

We build the network by calling the procedure $\mathbf{Net}(m, 0, P)$, with a large constant P ($P = O(4^m)$).

Note that the network is identified by the edges in which the sessions cross each other where there is no importance to the order between those edges along a session path.

Now we describe a "bad" scenario that achieves the required number of operations. Note that in some states of the algorithm the global minimum among the *active* sessions is not unique and hence, there are more than one *active* session that can be selected for increase. Therefore, there are other scenarios beside the one that is described below, where some of the other scenarios may achieve a lower complexity. The scenario of a run of the Global Min Algorithm is then constructed by calling the following recursive procedure **Scenario**(m,0,P) (where $m > 1$) on the network $Net(m, 0, P)$. See appendix B for an example of $Scenario(4, 0, 128)$.

Procedure Scenario$(m, bottom, potential)$:

In the beginning of the procedure we assume that the *active* sessions are $\{S_1, S_2, \ldots, S_{2m}\}$, their rate is *bottom* and any update operation can increase the rate of an *active* session to be at most *bottom* + *potential*.

1. If $m = 1$ then apply step (a) followed by step (b):

   (a) The rate of session S_2 is increased to $bottom + potential$. (The edge which is common to S_1 and S_2 becomes saturated.)

   (b) The rates of sessions S_1 and S_2 are set to $bottom + \frac{potential}{2}$ (due to an increase in the rate of S_1). Sessions S_1 and S_2 become *done*.

2. If $m > 1$ then apply the following steps:

   (a) The rate of session S_{2m} is updated to $bottom + \frac{potential}{2}$. (The edge which is common to S_{2m-1} and S_{2m} becomes saturated.)

   (b) The rate of each of the sessions S_j where $j = 1, 2, \ldots, 2m-2$ is updated to be $bottom + \frac{potential}{2}$ (total of $2m-2$ updates, where in each update the edge which is common to S_j and S_{2m} becomes saturated.)

   (c) The rates of sessions S_{2m-1} and S_{2m} are set to $bottom + \frac{potential}{4}$ (due to an increase in the rate of S_{2m-1}). Sessions S_{2m-1} and S_{2m} become *done*.

   (d) The scenario is recursively extended by calling
   $\mathbf{Scenario}(m-1, bottom + \frac{potential}{2}, \frac{potential}{4})$ on the network
   $\mathbf{Net}(m-1, bottom + \frac{potential}{2}, \frac{potential}{4})$.

Note that after sessions S_{2m-1} and S_{2m} become *done* then the rate of all *active* sessions is indeed $bottom + \frac{potential}{2}$ and their potential is the $\frac{potential}{4}$ flow that was released by S_{2m}.

One can see that when $m = 1$ there are two update operations. Hence, for an arbitrary $m > 1$, there are m iterations where for all i, the i^th iteration in $Scenario(i, *, *)$ requires $2i$ update operations. Therefore, the total number of update operations is $\sum_{i=1}^{n/2}(2i) = \Omega(n^2)$.

Note that in most cases we can assume that initially $bottom = 0$ (when we start from scratch).

□

5 Local Min Scheduler

In this algorithm an *active* session is selected to be increased if its rate is smaller than or equal to the rates of all the *active* sessions it shares an edge with. In each state of the algorithm the following rules are applied:

Marking a session done: Session $S_j \in \mathcal{A}^i$ is marked *done* if the following three conditions hold:

1. $rate(S_j)^i$ can not be increased.
2. $rate(S_j)^i$ is smaller than or equal to the rates of all the *active* sessions it shares an edge with (i.e., S_j is a local minimum).
3. For every *active* session S_m such that $rate(S_m)^i = rate(S_j)^i$ and S_m shares an edge with S_j, S_m is also a local minimum (i.e., $rate(S_m)^i$ is smaller or equal to the rates of all the *active* sessions S_r that share an edge with S_m).

Session selected for increase: In state Q_i, $S(o_i) = S_j \in \mathcal{A}^i$ such that S_j can be increased and $rate(S_j)^i$ is smaller than or equal to the rates of all the sessions it shares an edge with.

Termination: The algorithm terminates in state Q_i if $\mathcal{A}^i = \emptyset$.

This algorithm allows the rate of a session to be increased according to local considerations. Notice that the "local" consideration reflects only the decision which session should be increased (or marked done) and by "local" we mean that only edges along the session path need to be checked (in a very long session that covers many links this may end up not being very "local"). The execution of an update operation is identical in all the schedulers (and might involve a scan of several links). Hence, it seems to be closer to "real" networks behavior than the global min scheduler. Here we show that this algorithm has the same convergence complexity as the global min algorithm, i.e., $\Theta(n^2)$ update operations.

Observation 5.1 *The Local Min algorithm is a bottleneck algorithm.*

Proof: We have to show compatibility to Definition 3.2. Since the algorithm confirms to the conventions of Section 2 we just have to show that it is a general bottleneck algorithm. One can see that an *active* session S_j is marked *done* in state Q_i only if there is an edge e where:

1. $rate(S_j)^i \geq FS^i(e)$ (since it can not be increased).

2. For all *active* sessions $S_m \in \mathcal{A}^i \cap \gamma(e)$, $rate(S_m)^i \geq rate(S_j)^i$ (since S_j is a local minimum). Hence, since S_j can not be increased we have that for all those sessions (including S_j) $rate(S_m)^i \geq FS^i(e)$ which implies that $rate(S_m)^i = FS^i(e)$.

3. All *active* sessions $S_m \in \mathcal{A}^i \cap \gamma(e)$ (including S_j) are local minimum (since S_j is marked *done*). Hence, e is a bottleneck edge for all those sessions.

□

The following lemma and its proof are similar to Lemma 4.2. It shows that if a session rate becomes local minimum in two different states then at least one *active* session becomes *done* between those two states.

Lemma 5.2 *In the Local Min algorithm, if for $l > i$ $S(o_l) = S(o_i) = S_j$ then $\mathcal{A}^l \subsetneq \mathcal{A}^i$.*

Proof: The proof of Lemma 4.2 applies verbatim. □

Hence, we have the following theorem, which is analogous to Theorem 4.3.

Theorem 5.3 *The number of update operations using the Local Min algorithm is $O(n^2)$.* □

As for the tight lower bound, we have the following Theorem:

Theorem 5.4 *There is an infinite family of networks with n sessions in which the number of update operations in a run of the Local Min algorithm is $\Omega(n^2)$.*

Proof: Since the global minimum is also a local minimum the construction of Theorem 4.4 applies to the Local Min algorithm as well. □

6 Arbitrary Scheduler

In this case, the session that is increased in each round of the algorithm is chosen arbitrarily from $\mathcal{A}^i$. In each step of the algorithm the following rules are applied:

Marking a session done: Session S_j is marked *done* in state Q_i if there is an edge $e \in S_j$ that is a bottleneck edge in state Q_i, and for every session $S_r \in \gamma(e) \cap \mathcal{A}^i$, $rate(S_r)^i$ equals to $FS^i(e)$.

Session selected for increase: At each state the scheduler selects an arbitrary session that may be increased and updates it.

Termination: The algorithm terminates in state Q_i if $\mathcal{A}^i = \emptyset$.

Note the following:

1. The arbitrary scheduler is a bottleneck algorithm.

2. The *done* (or the *active*) markings are used by this algorithm only for termination detection (they do not affect the decision of the scheduler).

3. We can slightly change the definition of the scheduler and allow it to select an arbitrary session. If the selected session can not be increased then we consider this event as if nothing happens (this operation is not considered as an update operation).

This scheduler allows the rate of a session to be increased without any consideration of other sessions. Hence, it seems to be closer to "real" networks behavior than the schedulers presented previously. (In fact, the only major difference between this scheduler and the behavior of real rate based networks stems from the fact that here the update operations are done sequentially, i.e., there are no concurrent update operations.)

In this section we show that using this mechanism, the rate vector always converges to the max-min vector after no more than $O(\Phi^n)$ update operations, where $\Phi = \frac{\sqrt{5}+1}{2}$ is the golden ratio (and n is the number of sessions). An exponential lower bound of $\Omega(2^{\frac{n}{2}})$ update operations is presented (in Theorem 6.11) for an infinite family of networks.

First we give some definitions that are used in the sequel to obtain the required results.

Definition 6.1 *For $k \geq 1$, we denote by $F(k)$ the maximum number of update operations that may be performed in a network with k sessions under the arbitrary scheduler.*

The results presented in the sequel are achieved using recursive techniques. Here we define a prefix and later we show how the prefix is used to form the recursive formula.

Definition 6.2 *Let G be a network with n sessions $\{S_1, S_2, \ldots, S_n\}$. For every state Q_i of the algorithm $prefix(i) = \{S_{i_1}\ ,\ S_{i_2}, \ldots, S_{i_k}\}$ (where $k \leq n$) is defined to be the set of sessions that were updated (either increased or decreased) at least once in $o_1, o_2, \ldots, o_{i-1}$.*

Following the definition of the prefix we describe a projection of the network. This projection tries to isolate the environment of the sessions that are updated during the early operations.

Definition 6.3 *Let $G = (V, E)$ be a network with n sessions $\{S_1\ ,\ S_2, \ldots, S_n\}$ in which the arbitrary scheduler is implemented. Let $prefix(i) = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ and let $\{S_{i_{k+1}}, S_{i_{k+2}}, \ldots, S_{i_n}\}$ be the rest of the sessions in G. The projection of G over i, $PROJ_i(G)$, is a new network $G' = (V, E')$ that is defined as follows:*

1. *$E' = S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k}$.*
2. *For every session S_{i_j}, where $j \geq k+1$, the value of $rate(S_{i_j})^0$ in G' is the same as in G.*
3. *For every edge $e' \in E'$ we set $Cap(e') = Cap(e) - \sum_{S_{i_j} \in \gamma(e) \cap j \geq k+1} rate(S_{i_j})^0$, where $e \in E$ is the analogue edge to e'. (For each edge we subtract the total amount of flow of sessions for which the rate has not been adjusted.)*

The following observation bounds the length of a prefix as a function of the number of sessions that it updates.

Observation 6.4 *Let $G = (V, E)$ be a network with n sessions $\{S_1\ ,\ S_2\ \ldots\ S_n\}$ in which the arbitrary scheduler is implemented. Let Q_i be a state where $prefix(i) = \{S_{i_1}\ ,\ S_{i_2}\ \ldots\ S_{i_k}\}$. Then, $i \leq F(k)$ (i.e., the number of possible update operations that involve no more than k sessions in an arbitrary network is less than or equal to $F(k)$).* □

The next definition is related to the maximum rate of a session.

Definition 6.5 *For every session S_j we define $bound(S_j)$ to be an upper bound on $rate(S_j)$ during the algorithm execution, i.e., for all $i \geq 0$ we have $bound(S_j) \geq rate(S_j)^i$.*

The notion of $bound(S_j)$ is used in order to have a more refined restriction for the potential rate of sessions that are sharing a link with session S_j, as shown in the sequel.

The following definition extends the definition of Fair Share:

Definition 6.6 *For every edge e, where $N^i(e) > 0$, and for every session $S_j \in \mathcal{A}^i$, $PFS^i(e,j)$, the Potential Fair Share of link e in state Q_i, is defined as follows:*
If $S_j \in \gamma(e)$ and $N^i(e) > 1$ then $PFS^i(e,j) = \frac{Cap(e) - bound(S_j) - allotted^i(e)}{N^i(e) - 1}$
Else $PFS^i(e,j) = FS^i(e)$.

The following definition is an extension for the term LB:

Definition 6.7 *Let $S_j \in \mathcal{A}^i$ and let $S_r \neq S_j$. Then,*

$$MLB_j{}^i(r) = \min_{e \in S_j}\{PFS^i(e,r)\}$$

($MLB_j{}^i(r)$ is the smallest Potential Fair Share with respect to S_r along the path of S_j in state Q_i.)

Since the *Potential Fair Share* is non decreasing (by similar arguments to those in Lemma 3.4), then one can see that after the rate of session S_j is updated in state Q_l then $rate(S_j)^{l+1}$ is greater or equal to the values of $MLB_j{}^i(r)$ for every $S_r \neq S_j$ and for every $i \leq l$ (by similar arguments to those in Lemma 3.7 and in Corollary 3.9). Hence, we have the following extension of Theorem 3.10:

Theorem 6.8 *Consider state Q_i of a bottleneck algorithm where either $S(o_i) = S_j$ or $rate(S_j)^{i+1} < rate(S_j)^i$. Then, for all $l \geq i$ and for all $S_r \neq S_j$, $rate(S_j)^l \geq MLB_j{}^i(r)$.* □

The next lemma bounds the number of operations performed after the rate of all sessions have been updated at least once.

Lemma 6.9 *Let $G = (V,E)$ be a network with n sessions $\{S_1, S_2, \ldots, S_n\}$ and let Q_i be a state of the algorithm where $prefix(i)$ contains all the n sessions. Then, there are at least two sessions $\{S_j, S_r\}$ that are not updated in all operations o_l, such that $l \geq i$ (i.e., for every $l \geq i$, $rate(S_j)^l = rate(S_j)^i$ and $rate(S_r)^l = rate(S_r)^i$).*

Proof: Let $e \in E$ be an edge where $FS^0(e)$ is the smallest Fair Share among all edges in Q_0. Since $\gamma(e) \subseteq prefix(i)$ then for every session $S_k \in \gamma(e)$ there is $m < i$ where $rate(S_k)$ is updated in state Q_m. Since the *Fair Share* is non decreasing then, by Theorem 3.10, we have that for every session $S_k \in \gamma(e)$, and for every $l \geq i$, $rate(S_k)^l \geq FS^0(e)$ which implies that for every $l \geq i$, $rate(S_k)^l = FS^0(e)$. If $\gamma(e)$ contains at least two sessions we are done. Else, $\gamma(e)$ contains exactly one session S_r and $rate(S_r)^i = bound(r) = Cap(e)$. Let $\bar{E} \subseteq E$ be a set of edges where $\bar{e} \in \bar{E}$ iff $\gamma(\bar{e}) - S_r \neq \emptyset$. Let $e' \in \bar{E}$ be an edge where $PFS^i(e', r)$ is minimum among all edges in $\bar{E}$. Clearly, for every session $S_k \in \gamma(e') - S_r$, and for every $l \geq i$ we have that $rate(S_k)^l \geq PFS^0(e', r) = MLB_k{}^0(r)$. However, since $rate(S_r)^i = bound(r)$ then it implies that for every session $S_k \in \gamma(e') - S_r$, and for every $l \geq i$ we have that $rate(S_k)^l = MLB_k{}^0(r)$. We take an arbitrary session in $\gamma(e') - S_r$ to be S_j and we are done. □

The following theorem proves an upper bound on the number of update operations.

Theorem 6.10 *Let $F(n)$ be the number of update operations performed in the arbitrary scheduler in a network with n sessions. Then, $F(n) = O(\Phi^n)$ where $\Phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.*

Proof: Clearly $F(1) = 1$ and $F(2) = 2$. To prove the theorem we show that for every $k > 2$, we have the inequality $F(k+1) \leq F(k) + 1 + F(k-1)$.

Any scenario $\alpha = Q_0 o_0 Q_1 o_1 \ldots$ can be divided into three parts $\beta\tau\sigma$, such that $\beta = Q_0 o_0 \ldots Q_{l-1} o_{l-1}$, $\tau = Q_l o_l$ and $\sigma = Q_{l+1} o_{l+1} \ldots$, where l is selected such that: $|prefix(l)| \leq k$ (i.e., no more than k sessions were updated during the first l operations) and $|prefix(l+1)| = k+1$ (i.e., Q_l is the first time that the $k+1^{st}$ session is updated).

By Observation 6.4 the first part may take no more than $F(k)$ operations. By Lemma 6.9, there are at least two sessions that are not updated after operation o_l is finished. Hence, in the third part we have a network with no more than $k-1$ sessions that may be changed, which implies (by Observation 6.4) that this part may take at most $F(k-1)$ additional update operations. □

The following theorem gives a lower bound on the convergence complexity of the algorithm.

Theorem 6.11 *For every odd number n, there exists a network with n sessions, for which there is a schedule that requires $\Omega(2^{\frac{n}{2}})$ update operations.*

Proof: We construct a network denoted $Exp\_Net(m, potential, progress)$. The network has $2m-1$ sessions, where *potential* is an upper bound on the amount of flow (either temporary of final) that any session can achieve and *progress* is a parameter that is used to decrease *potential*. Figure 3 is an example of a network $Exp\_Net(3, 100, 10)$. We use an array, called *pot* where for every session S_i, $pot(i)$ is an exact upper bound for the amount of flow that this session can achieve in the future. We assume that $potential > 2 \cdot m \cdot progress$.

The procedure that builds the network is as follows:

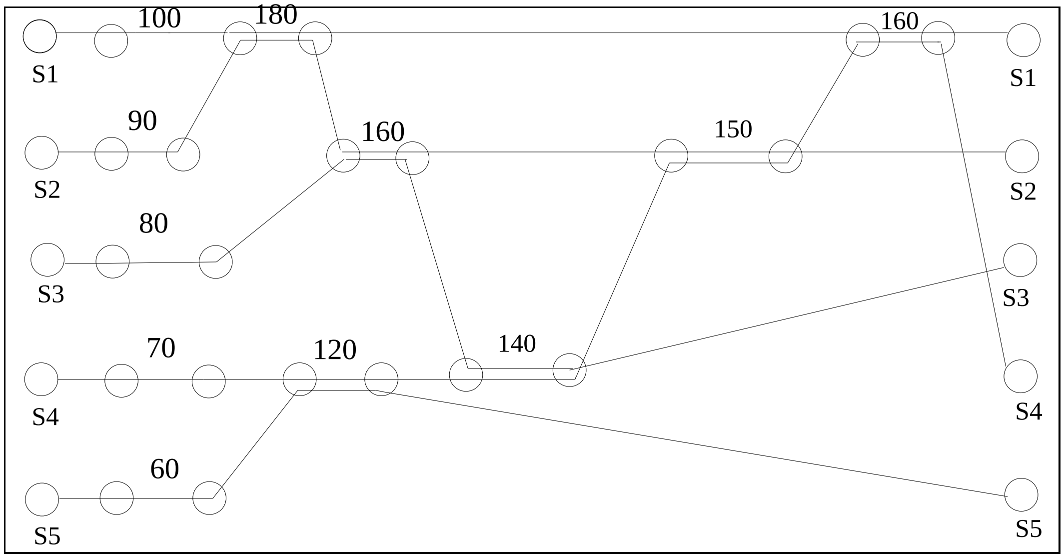

Figure 3: An example of a network $Exp\_Net(3, 100, 10)$

Procedure Exp_Net$(m, potential, progress)$:

for i:=1 to $2 \cdot m - 1$ do
begin
  S_i uses solely an edge e where $Cap(e) := potential$
  $pot(i) := potential$
  $potential := potential - progress$
  if i is even then
    for j:= 1 to $i - 1$ do
      S_i and S_j share an edge e where $Cap(e) := pot(i) + pot(j) - progress$
  else {i is odd } if $i > 1$ then
    S_i and S_{i-1} share an edge e where $Cap(e) := pot(i) + pot(i-1) - progress$
end { of for }

Notice that the network is identified by the edges in which the sessions cross each other where there is no importance to the order between these edges along a session path.

Now we describe a "bad" scenario that achieves the required number of operations. Notice that in some states of the algorithm there are more than one *active* session that can be selected for increase. Therefore, there are other scenarios beside the one that is described below, where some of the other scenarios may achieve a lower complexity. The scenario of a run of the Global Min Algorithm is then constructed by calling the following recursive procedure **Exp_Scenario**(m,pot) on the network $Exp\_Net(m, potential, progress)$ (where *pot* is the array that was computed by $Exp\_Net(m, potential, progress)$ as described above). We assume that the scenario starts from scratch such that for every session S_j, the initial rate is less than $pot(j)$.

Procedure Exp_Scenario(m, pot) :

1. If $m = 1$ then the rate of session S_1 is increased to $pot(1)$. (The edge that is used solely by S_1 becomes saturated.)

2. If $m > 1$ then
   (a) call $Exp\_Scenario(m-1, pot)$.
   (b) The rate of session S_{2m-2} is increased to $pot(2m-2)$. This causes the rates of all sessions S_j, where $j < 2m-2$ to be $pot(j) - progress$.
   (c) The rate of session S_{2m-1} is increased to be $pot(2m-1)$. This decreases the rate of session S_{2m-2} to $pot(2m-1) = pot(2m-2) - progress$. Now for all sessions S_j, such that $j < 2m-2$, the rate is $pot(j) - progress$ and there is no restriction imposed by S_{2m-2} that disables their rate from increasing to $pot(j)$. Hence, we can repeat step (a) as follows:
   (d) call $Exp\_Scenario(m-1, pot)$.

Let $H(m)$ be the number of update operations in $Exp\_Scenario(m, pot)$. One can see that when $m = 1$ there is one update operation (i.e., $H(1) = 1$) and for an arbitrary $m > 1$, $H(m) = H(m-1) + 2 + H(m-1) = 2 \cdot H(m-1) + 2$. Therefore, the total number of update operations is $\Omega(2^m) = \Omega(2^{\frac{n}{2}})$

In appendix C we show this scenario for $m = 3$, $potential = 100$ and $progress = 10$. $\square$

7 Approximations

In this section we present algorithms in which a session rate is increased only if the increase is by more than δ. The algorithm terminates when the rate of no single session can be increased by more than δ.

More formally, we say that:

1. For every i, j, either $rate(S_j)^{i+1} \leq rate(S_j)^i$ or $rate(S_j)^{i+1} > rate(S_j)^i + \delta$.
2. In Q_{final}, the final state of the algorithm, for every session S_j, $\Delta^{final}(j) \leq \delta$.

We call an allocation that satisfies the second condition a *δ-max-min allocation.* The motivation for studying this issue is that in dynamic networks, the computation of the exact max-min vector may not be reached (due to the dynamic nature of the system) and hence we are willing to pay the overhead required by the update operations only if the benefit is significant. Also, an algorithm that converges fast to the new Q_{final} but slowly to the "standard" max-min vector may be considered better than another algorithm in which it takes more time to reach the new Q_{final} state even if it converges faster to the "standard" max-min vector.

In Subsection 7.1 we analyze the implications of such an approximation. It turns out that every session S_j may differ from its optimal rate (the value of $rate(S_j)$ in the max-min vector) by at most $O(\delta \cdot 2^n)$ (where n is the number of sessions). We show an example in which there is a feasible allocation, such that the rate of only one session can be improved by δ (and the rate of all the other sessions can not be increased). Yet, in this example, there is some session that is $\Omega(\delta \cdot 2^{\frac{n}{2}})$ from its value according to the max-min vector.

Note that this potential "bad" property of the network can not be overcome by any approximation algorithm for rate control. However, this difference between the two resulting rate vectors is not necessarily a negative indication. It mainly suggests that the max-min vector may be unstable in the sense that small fluctuations may change it dramatically. The proofs presented here show that the addition or deletion of a session with a small rate from the max-min vector may cause a dramatic (exponential in the number of sessions) change in the rate of some other sessions. Thus, the max-min vector is not necessarily better than the vector computed by an approximate algorithm (it may suffer from large instability that may result from a modest dynamic update). One of the implications of these results is the settlement of an open question raised by Charny [Cha94] (where she asks about the convergence of rate based algorithms in the case that rates are restricted to discrete values).

In Subsection 7.2 we show that the Approximate Global Min algorithm and the Approximate Enhanced Local Min algorithm (to be described in the sequel) with parameter δ reach quiescence in at most $O(\min(n^2, n \cdot \log_{\frac{h}{h-1}} \frac{\mathcal{W}}{\delta}))$ operations, where h is the maximum number of sessions that share a single edge (we assume $h \geq 2$ otherwise it is trivial) and $\mathcal{W}$ is defined as follows:

$$\mathcal{W} = \max_{S_j} \min_{e \in S_j} \{Cap(e)\}$$

(i.e., for each session we choose the edge with the minimum capacity that it passes and $\mathcal{W}$ is the maximum over all these edges). Hence, if h is bounded by a constant then the number of operations is no more than $O(\min(n^2, n \ \log \frac{\mathcal{W}}{\delta}))$.

Remark: A bound of $O(\min(n^2, n \cdot \log_{\frac{h}{h-1}} \frac{\mathcal{W}}{\delta}))$ on the number of update operations in either the Approximate Global Min algorithm or the Approximate Enhanced Local Min algorithm (given in Subsection 7.2) is valid for the conservative approach as well, and can be shown using similar arguments to those presented in this section.

7.1 The difference between the δ-max-min allocation and max-min

In this subsection we bound the difference between the rate of a session in δ-max-min allocation and its rate in the max-min vector.

We consider an arbitrary δ-max-min allocation and use the following definitions:

$\bar{X} = (x^1, x^2, \ldots, x^n)$: The max-min vector, where x^i is the rate of session S_i.

$\bar{Z} = (z^1, z^2, \ldots, z^n)$: The rate vector as computed by an approximate algorithm such that $z^i = rate(S_i)^{final}$. The main assumption is that the rate of no session can be increased by a single atomic operation in more than δ flow units.

We assume without loss of generality that the sessions are enumerated such that:

1. $\bar{X}$ is a non descending vector.
2. Considering a set of sessions all of which get the same rate in the max-min vector. We divide this set into two disjoint groups: (i) The group of sessions in which the rate obtained by the approximate algorithm is smaller than or equal to the allocation according to the max-min vector. (ii) The group of sessions in which the rate obtained by the approximate algorithm is larger than the allocation according to the the max-min vector. We assume without loss of generality that the sessions in (i) have a lower index than the sessions in (ii).

The following lemma states that x^i is above z^i by at most δ plus the amount of flow by which sessions with smaller index in $\bar{X}$ are above their rate in the max-min vector. The intuition is that $rate(S_i)^{final}$ can not be increased by more than δ without decreasing other sessions with equal or smaller rate (in $\bar{Z}$). Hence, this increase will be possible only if the rates of these other sessions will be decreased by other restrictions.

Lemma 7.1 *For all $i \geq 1$, $x^i - z^i \leq \delta + \sum_{l<i} |x^l - z^l|$.*

Proof: Clearly, since S_i can not be increased by more than δ, there is an edge $e \in S_i$, such that $\Delta^{final}(e, i) \leq \delta$. Hence, the restriction of $Cap(e)$ and the max min fairness concept imply that any further increase in the rate of S_i is at most the amount of rate that is released by other sessions that are sharing e. It is obvious that the max min allocation of sessions that share e and release the rate, is lower than the fair allocation of S_i which implies that their index is lower than i. $\square$

The following lemma is the analogous to the amount of flow by which x^i may be below z^i. This amount is at most the amount of flow by which sessions with smaller index in $\bar{X}$ are below their rate in the max-min vector. The intuition is that in the adjustment process from $\bar{Z}$ to $\bar{X}$, a decrease in $rate(S_i)^{final}$ is caused only by sessions that are increased but do achieve a rate value which is less than or equal to the rate of S_i.

Lemma 7.2 *For all $i \geq 1$, $z^i - x^i \leq \sum_{l<i} |x^l - z^l|$.*

Proof: Clearly, there is an edge $e \in S_i$, where

1. $\sum_{S_k \in \gamma(e)} x^k = Cap(e)$.

2. $x^i \geq x^k$ for all $S_k \in \gamma(e)$.

As e is saturated according to the max-min vector it is clear that the amount of rate by which S_i is reduced in the adjustment process from $\bar{Z}$ to $\bar{X}$ is no more than the amount of allocation added to all other sessions that share e. $\square$

The following theorem bounds from above the difference between the δ-max-min allocation and the max-min allocation.

Theorem 7.3 *For all $i \geq 1$, $|x^i - z^i| \leq \delta \cdot 2^{i-1}$.*

Proof: From Lemma 7.1 and Lemma 7.2 one can see that $|x^1 - z^1| \leq \delta$ and $|x^i - z^i| \leq \delta + \sum_{l<i} |x^l - z^l|$. Hence, the Theorem follows by induction on i. $\square$

Theorem 7.3 shows that the rate of a session as computed by the approximate algorithm may differ by at most $\delta \cdot 2^{n-1}$ from its rate in the max-min vector. The following theorem states a matching lower bound on the possible difference between $rate(S_j)^{final}$ and the rate of S_j in the max-min vector.

Theorem 7.4 *For all $1 \leq i \leq n$, $|x^i - z^i|$ can be $\Omega(\delta \cdot 2^{\frac{i}{2}})$. Hence, $|x^n - z^n|$ can be $\Omega(\delta \cdot 2^{\frac{n}{2}})$.*

Proof: We prove the theorem by presenting a network configuration with n sessions for every $n = 4m + 3$, $m = 1, \ldots$. In the configuration, the rate of one session, S_1, may be increased by δ units, while the rate of no other session may be increased. We show that for every i, $1 \leq i \leq \frac{n-1}{2}$ the rate of sessions S_{2i} and S_{2i+1} is $2^i \cdot \delta/6$ away from their value in the max-min vector.

In the example there are two sessions, S_2 and S_3 sharing an edge with S_1 such that increasing S_1 by δ would decrease each of them by $\delta/3$. In addition these two sessions share two more edges: one edge with session S_4 and another edge with session S_5, such that their decrease by $\delta/3$ would increase both the rate of S_4 and of S_5 by $\delta \cdot 2/3$. S_4 and S_5 share two more edges: one edge with session S_6 and another edge with session S_7, such that their increase by $\delta \cdot 2/3$ would decrease both the rate of S_6 and of S_7 by $\delta \cdot 4/3$. Then this construction is repeated inductively on top of S_6 and S_7.

Now we provide the formal description of the network.
The network has $n = 4m + 3$ sessions.
The edges of the network are described as follows:

1. Edge e_1 is shared by sessions S_2 and S_3.
2. Edge e_2 is shared by sessions S_1, S_2 and S_3.
3. For each k, where $m \geq k \geq 1$, there are four more edges, $e_{4k-1}, e_{4k}, e_{4k+1}, e_{4k+2}$ that are shared by $\{S_{4k-2}, S_{4k-1}, S_{4k}\}$, $\{S_{4k-2}, S_{4k-1}, S_{4k+1}\}$, $\{S_{4k}, S_{4k+1}, S_{4k+2}\}$ and by $\{S_{4k}, S_{4k+1}, S_{4k+3}\}$ respectively.
4. $Cap(e_1) = 8 \cdot C$ and $Cap(e_2) = 9 \cdot C$, where C is an arbitrary constant.
5. $\forall j, 2 \leq j \leq 2m+1, Cap(e_{2j-1}) = Cap(e_{2j}) = 2^{j+2} \cdot C$.

We consider the case in which $rate(S_1)^{final} = 0$ and $\forall i$, $1 \leq i \leq \frac{n-1}{2}$, $rate(S_{2i})^{final} = rate(S_{2i+1})^{final} = 2^{i+1} \cdot C$. Clearly, $\Delta^{final}(1) = 3 \cdot C$. We consider this value as δ. For all other sessions S_i, such that $i \neq 1$, $\Delta^{final}(i) = 0$.

The adjustment of the rates is as follows:

1. S_1 is increased by δ (from 0 to $3C$, due to the restrictions of e_2).
2. S_2 and S_3 are decreased by C (from $4C$ to $3C$, due to the restrictions of e_2).
3. S_4 and S_5 are increased by $2C$ (from $8C$ to $10C$, due to the restrictions of e_3 and e_4).
4. S_6 and S_7 are decreased by $4C$ (from $16C$ to $12C$, due to the restrictions of e_5 and e_6).

In general, for every l, where $1 \leq l \leq \frac{n-3}{4}$, S_{4l} and S_{4l+1} are increased by $2^{2l-1} \cdot C$ while S_{4l+2} and S_{4l+3} are decreased by $2^{2l} \cdot C$. Consider the difference vector of the form $\bar{X} - \bar{Z}$ (i.e., the i-th coordinate of the vector equals $x^i - z^i$). As $C = 1/3 \cdot \delta$ the vector has the following values: $1/3 \cdot \delta \cdot (3, -1, -1, 2, 2, -4, -4, 8, 8, -16, -16, 32, 32, \ldots)$. Hence, for all i, $1 \leq i \leq \frac{n-1}{2}$, we have that $|x^{2i} - z^{2i}| = |x^{2i+1} - z^{2i+1}| = \delta \cdot 2^i/6$. $\square$

Note that the lower bound is shown in a case in which only one session can be increased.

7.2 Two Approximate Algorithms and their convergence analysis

Here we analyze the convergence of two algorithms: Approximate Global Min algorithm and Approximate Enhanced Local Min algorithm.

In the Approximate Global Min algorithm, the scheduler increases the rate of a session only if it is minimum among all *active* sessions and if the increase is by more than δ. The formal description of the algorithm is as follows:

Marking a session done: If the smallest rate *active* session in the network can not be increased by more than δ, then it is marked *done*.

Session selected for increase: The scheduler selects session $S_j \in \mathcal{A}^i$ such that $rate(S_j)^i$ is minimum among all sessions in $\mathcal{A}^i$ and can be increased by more than δ. S_j is selected as $S(o_i)$.

Termination: The algorithm terminates when $\mathcal{A}^i = \emptyset$.

The Approximate Enhanced Local Min algorithm is based upon a local scheduler and is an enhancement of the local min technique. Before giving the definition of the algorithm we need the following definition:

Definition 7.5 *A session $S_j \in \mathcal{A}^i$ is* dis3-local-min *in state Q_i if for every* active *session S_m that share an edge with S_j:*

1. $rate(S_j)^i \leq rate(S_m)^i$.
2. *For every* active *session S_r that shares an edge with S_m $rate(S_j)^i \leq rate(S_r)^i$.*
3. *For every* active *session S_p that shares an edge with S_r $rate(S_j)^i \leq rate(S_p)^i$.*

In each state of its run the Approximate Enhanced Local Min algorithm follows the following rules:

Marking a session done: A *dis3-local-min* session whose rate can not be increased by more than δ is marked *done*.

Session selected for increase: In state Q_i, the scheduler chooses a *dis3-local-min* session S_j whose rate can be increased by more than δ and sets S_j as $S(o_i)$.

Termination: The algorithm terminates if $\mathcal{A}^i = \emptyset$.

We show a few properties that enable us to prove the theorems about the convergence of the algorithms. The next claim states that in the Approximate Global Min algorithm the rate of the session with the minimum rate is a non decreasing function.

Claim 7.6 *In the Approximate Global Min algorithm, $\forall l$ and i, if $l \geq i$ then $rate(S(o_l))^l \geq rate(S(o_i))^i$.*

Proof: Let $S(o_i) = S_j$. Following Definition A.1 and Definition A.2, for every session S_m decreased in operation o_i, $rate(S_m)^{i+1} \geq rate(S_j)^{i+1} > rate(S_j)^i$. Also, for an *active* session S_m which is not decreased, $rate(S_m)^{i+1} = rate(S_m)^i \geq rate(S_j)^i$. Therefore, $rate(S(o_i))^i \leq rate(S(o_{i+1}))^{i+1} \leq \ldots \leq rate(S(o_l))^l$. □

As for the Approximate Enhanced Local Min algorithm, in order to have an analogous statement we need the following lemma.

Lemma 7.7 *Consider state Q_i in the Approximate Enhanced Local Min algorithm in which either $S(o_i) = S_j$ or $rate(S_j)^{i+1} < rate(S_j)^i$. Then, for all $l \geq i$ $rate(S_j)^l \geq LB_j{}^i$.*

Proof: We use statements (and proofs) which are analogous to those in Section 3, from Lemma 3.4 to Theorem 3.10. □

The following lemma shows a basic monotonous property related to sessions that become *dis3-local-min*.

Lemma 7.8 *Let S_j and S_m share an edge and let S_m and S_r share an edge (it is possible that $S_r = S_m$ or $S_r = S_j$). Let $l > i$ in the Approximate Enhanced Local Min algorithm, such that $S(o_i) = S_j$ and $S(o_l) = S_r$. Then $rate(S_j)^i \leq rate(S_r)^l$.*

Proof: As S_j is *dis3-local-min* in state Q_i then $rate(S_j)^i \leq rate(S_r)^i$ and $rate(S_j)^i \leq LB_r{}^{i+1}$. Hence, by Lemma 7.7 for all $k \geq i+1$ $rate(S_r)^k \geq LB_r{}^{i+1} \geq rate(S_j)^i$. □

The monotonous properties proved in the above statements for the two algorithms turn out to be useful when setting some bounds, as the following two corollaries.

Corollary 7.9 *In the Approximate Global Min algorithm, consider an edge $e \in G$, such that $\sum_{S_m \in \gamma(e)} rate(S_m)^i = Cap(e)$, $S_t \in \{\gamma(e) \cap \mathcal{A}^i\}$ and $rate(S_t)^i \geq rate(S_m)^i$ for all $S_m \in \{\gamma(e) \cap \mathcal{A}^i\}$. Then, for every* active *session $S_m \in \{\gamma(e) \cap \mathcal{A}^i\}$ and for every $l \geq i$:*

$$\begin{aligned} rate(S_m)^i - rate(S_m)^l &\leq rate(S_m)^i - rate(S(o_i))^i \\ &\leq rate(S_t)^i - rate(S(o_i))^i \end{aligned}$$

□

The analogous corollary applies for the Approximate Enhanced Local Min algorithm.

Corollary 7.10 *In the Approximate Enhanced Local Min algorithm, consider an edge $e \in G$, such that $\sum_{S_m \in \gamma(e)} rate(S_m)^i = Cap(e)$, $S_t \in \{\gamma(e) \cap \mathcal{A}^i\}$, $S(o_i) \in \gamma(e)$ and $rate(S_t)^i \geq rate(S_m)^i$ for all $S_m \in \{\gamma(e) \cap \mathcal{A}^i\}$.*

Then, for every active *session $S_m \in \{\gamma(e) \cap \mathcal{A}^i\}$ and for every $l \geq i$:*

$$\begin{aligned} rate(S_m)^i - rate(S_m)^l &\leq rate(S_m)^i - rate(S(o_i))^i \\ &\leq rate(S_t)^i - rate(S(o_i))^i \end{aligned}$$

□

Corollary 7.11 *In the Approximate Global Min algorithm, for every $i \geq 0$ let e be an edge in G and let $S_t \in \{\gamma(e) \cap \mathcal{A}^i\}$ such that $rate(S_t)^i \geq rate(S_m)^i$ for all $S_m \in \gamma(e)$. If $\sum_{S_m \in \gamma(e)} = Cap(e)$ then, for all $S_j \in \{\gamma(e) \cap \mathcal{A}^i\}$ and for all $l \geq i$ the following holds: $rate(S_j)^l - rate(S_j)^i \leq (h'-1) \cdot (rate(S_t)^i - rate(S(o_i))^i)$, where $h' = |\gamma(e)|$.* □

The analogous corollary for the Approximate Enhanced Local Min algorithm is:

Corollary 7.12 *In the Approximate Enhanced Local Min algorithm, for every $i \geq 0$ let e be an edge in G and let $S_t \in \{\gamma(e) \cap \mathcal{A}^i\}$ such that $rate(S_t)^i \geq rate(S_m)^i$ for all $S_m \in \gamma(e)$. If $S(o_i) \in \gamma(e)$ and*
$\sum_{S_m \in \gamma(e)} = Cap(e)$ then, for all $S_j \in \{\gamma(e) \cap \mathcal{A}^i\}$ and for all $l \geq i$ the following holds: $rate(S_j)^l - rate(S_j)^i \leq (h' - 1) \cdot (rate(S_t)^i - rate(S(o_i))^i)$, where $h' = |\gamma(e)|$. □

The next claim is an extension of Lemma 3.5 and follows immediately from Definition A.1 and Definition A.2.

Claim 7.13 *If a rate of a session S_j is changed by operation o_{i-1} (i.e., $rate(S_j)^i \neq rate(S_j)^{i-1}$) then there is an edge $e \in S_j$ where $rate(S_j)^i \geq rate(S_m)^i$ for all $S_m \in \gamma(e)$ and $\sum_{S_m \in \gamma(e)} = Cap(e)$.* □

For every session S_j we observe the list $j_k, k = 1, 2, \ldots$ where $S(o_{j_k}) = S_j$. We define

$$P_j(k) = \max_{l \geq j_k}\{rate(S_j)^l - rate(S_j)^{j_k}\}$$

namely the potential increase in the rate of session S_j after its first k increase operations. Intuitively, this is an upper bound on the maximum distance between the value of $rate(S_j)^{j_k}$ and the value of $rate(S_j)$ in the final state.

Claim 7.14 *In the Approximate Global Min algorithm, $P_j(k) \leq (h-1)(rate(S_j)^{j_k} - rate(S_j)^{j_{k-1}})$ for all $k = 2, 3, \ldots$ where h is the maximum number of sessions that share a single edge $e \in S_j$.*

Proof: For every $k = 2, 3, \ldots$ let $l > j_{k-1}$ be such that Q_l is the first state where $rate(S_j)^l = rate(S_j)^{j_k}$ (i.e., $rate(S_j)^{l-1} \neq rate(S_j)^{j_k}$). Clearly, by Claim 7.13 in state Q_l there is an edge $e \in S_j$ such that $rate(S_j)^l \geq rate(S_m)^l$ for all $S_m \in \gamma(e)$ and $\sum_{S_m \in \gamma(e)} = Cap(e)$. By Claim 7.6, $rate(S(o_l))^l \geq rate(S(o_{j_{k-1}}))^{j_{k-1}} = rate(S_j)^{j_{k-1}}$ and hence (by Corollary 7.11 and the fact that $h' \leq h$):

$$\begin{aligned} P_j(k) &\leq (h'-1)(rate(S_j)^l - rate(S(o_{j_{k-1}}))^{j_{k-1}}) \\ &= (h'-1)(rate(S_j)^{j_k} - rate(S_j)^{j_{k-1}}) \\ &\leq (h-1)(rate(S_j)^{j_k} - rate(S_j)^{j_{k-1}}) \end{aligned}$$

□

Claim 7.15 *In the Approximate Enhanced Local Min algorithm,*
$P_j(k) \leq (h-1)(rate(S_j)^{j_k} - rate(S_j)^{j_{k-1}})$ for all $k = 2, 3, \ldots$ where h is the maximum number of sessions that share a single edge $e \in S_j$.

Proof: The proof of Claim 7.14 applies verbatim except for the need to replace Claim 7.6 by Lemma 7.8. □

In the following statements we assume that $h \geq 2$ is the maximum number of sessions that share a single edge (if $h = 1$ then the convergence process is trivial).

Observation 7.16 *In either the Approximate Global Min algorithm or in the Approximate Enhanced local Min algorithm, for* $k = 3, 4, \ldots$ $P_j(k) \leq \min\{(h-1) \cdot A - B, (h-1) \cdot B\}$ *where* $A = (rate(S_j)^{j_{k-1}} - rate(S_j)^{j_{k-2}})$ *and* $B = (rate(S_j)^{j_k} - rate(S_j)^{j_{k-1}})$. $\square$

The last observation implies an exponential degradation in the value of P_j.

Corollary 7.17 *In either the Approximate Global Min algorithm or in the Approximate Enhanced local Min algorithm, for* $k = 3, 4, \ldots$ *we have* $P_j(k) \leq \frac{(h-1)}{h} P_j(k-1)$. $\square$

Lemma 7.18 *In either the Approximate Global Min algorithm or in the Approximate Enhanced Local Min algorithm, for all* $k > 2 + \log_{\frac{h}{(h-1)}} \frac{P_j(2)}{\delta}$ *we have* $P_j(k) < \delta$.

Proof: The lemma follows immediately from Corollary 7.17. $\square$

Let us first analyze the maximum number of operations taken by each of the two algorithms before it terminates. After that we show that when each of the algorithms terminates, no session can be increased by more than δ, without affecting the fairness concept.

The following is analogous to Lemma 4.2 (for the case of the Global Min algorithm) and to Lemma 5.2 (for the Approximate Enhanced Local Min algorithm).

Lemma 7.19 *In either the Approximate Global Min algorithm or in the Approximate Enhanced Local Min algorithm, for every session* S_j *and for every* $k = 2, 3, \ldots$ *we have that* $\mathcal{A}^{j_k} \subsetneq \mathcal{A}^{j_{k-1}}$. $\square$

Hence, using the same arguments as in Theorem 4.3 and in Theorem 5.3 we set a bound on the convergence complexity.

Theorem 7.20 *The number of update operations in any run of either the Approximate Global Min algorithm or the Approximate Enhanced Local Min algorithm is* $O(n^2)$. $\square$

Now we aim at setting a second bound on the convergence complexity. This bound is based on the exponential degradation in the value of P_j.

The following definition considers the maximum initial potential.

Definition 7.21 *In every network* G, $\mathcal{W}$ *is* $\max_{S_j} \min_{e \in S_j}\{Cap(e)\}$, *i.e., for each session we choose the edge with the minimum capacity and* $\mathcal{W}$ *is the maximum weight of all these edges.*

The next theorem presents the second bound.

Theorem 7.22 *In either the Approximate Global Min algorithm or in the Approximate Enhanced Local Min algorithm the number of increase operations for each session is* $O(\log_{\frac{h}{(h-1)}} \frac{\mathcal{W}}{\delta})$ *where* h *is the maximum number of sessions that share a single edge.*

Proof: Note that for each session S_j, $P_j(2) \leq Cap(e)$ for all edges $e \in S_j$. Hence, by Lemma 7.18, for all $k > 2 + \log_{\frac{h}{(h-1)}} \frac{P_j(2)}{\delta}$ and for all $l > j_k$ we have $\Delta^l(j) \leq P_j(k) < \delta$. $\square$

We thus get the combination of the above two bounds:

Theorem 7.23 *Both the Approximate Global Min algorithm and the Approximate Enhanced Local Min algorithm quiesce after* $O(\min(n^2, n \ \log_{\frac{h}{(h-1)}} \frac{\mathcal{W}}{\delta}))$ *update operations where h is the maximum number of sessions that share a single edge.* $\square$

Note that if h is a constant then the convergence complexity is $O(n \cdot log\frac{\mathcal{W}}{\delta})$.

To conclude, we need to show that when each of the two algorithms reaches quiescence, the computed allocation is in fact δ-max-min allocation. To show this, we state (in Theorem 7.31 and in Theorem 7.32) that after a session becomes *done* there is no state in which it may be increased by more than δ.

Observation 7.24 *In either the Approximate Global Min algorithm or in the Approximate Enhanced local Min algorithm let $S_j \in \mathcal{A}^i$ where $rate(S_j)^{i+1} < rate(S_j)^i$. Then $rate(S_j)^{i+1} > rate(S(o_i))^i + \delta$.*

Proof: Following Definition A.2, S_j is decreased in state Q_i only if $rate(S_j)^{i+1} \geq rate(S(o_i))^{i+1} > rate(S(o_i))^i + \delta$. $\square$

In the following we define a variable which is used to bound the rate of a session in the Approximate Global Min algorithm.

Definition 7.25 *In the Approximate Global Min algorithm, for every session S_j, $PLV(S_j)^i$, the Potential Lowest Value of S_j in state Q_i is $\min(rate(S_j)^i, rate(S(o_i))^i + \delta)$.*

The following is the analogous definition in the Approximate Enhanced Local Min algorithm.

Definition 7.26 *In the Approximate Enhanced Local Min algorithm, for every session S_j, $LPLV(S_j)^i$, the Local Potential Lowest Value of S_j in state Q_i is defined as follows:*

1. *If $S(o_i)$ shares an edge with S_j (including the case where $S_j = S(o_i)$) then $LPLV(S_j)^i = \min(rate(S_j)^i, rate(S(o_i))^i + \delta)$.*
2. *Else ($S(o_i)$ does not share an edge with S_j), let $\bar{S}$ be the set of sessions $\{S_{i_1}, S_{i_2}, \ldots\}$ that share an edge with S_j and are marked* done *in state Q_i. If $\bar{S} \neq \emptyset$ we choose the session S_{i_k} where $rate(S(i_k))^i$ is maximum among all sessions in $\bar{S}$ and set $LPLV(S_j)^i = \min(rate(S_j)^i, rate(S(i_k))^i + \delta)$.*
3. *Else ($S(o_i)$ does not share an edge with S_j and $\bar{S} = \emptyset$), if $i = 0$ then $LPLV(S_j)^0 = 0$ and if $i \geq 1$ then $LPLV(S_j)^i = LPLV(S_j)^{i-1}$.*

The next two observations show a monotonous property of the two new variables.

Observation 7.27 *In the Approximate Global Min algorithm, for every session S_j and for every $l \geq i$, $PLV(S_j)^l \geq PLV(S_j)^i$.*

Proof: By Claim 7.6 $rate(S(o_l))^l \geq rate(S(o_i))^i$. Hence, if $rate(S_j)^l \geq rate(S_j)^i$ we are done. Else, let Q_m be the last state before Q_l in which $rate(S_j)$ was decreased (i.e., $l > m \geq i$ and $rate(S_j)^{m+1} < rate(S_j)^m$). Then, by Observation 7.24
Since $rate(S(o_l))^l \geq rate(S(o_i))^i$ then $PLV(S_j)^l \geq rate(S(o_i))^i + \delta \geq PLV(S_j)^i$. $\square$

Observation 7.28 *In the Approximate Enhanced Local Min algorithm, for every session S_j and for every $l \geq i$, $LPLV(S_j)^l \geq LPLV(S_j)^i$.*

Proof: Let state Q_p, where $i \leq p$ be the last time in which the value of $LPLV(S_j)$ was updated before state Q_i and let state Q_k, where $l \geq k$, be the last time in which the value of $LPLV(S_j)$ was updated before state Q_l. Let session S_m be *dis3-local-min* in state Q_p that shares an edge with session S_j and let session S_r be *dis3-local-min* in state Q_k that shares an edge with session S_j. By Lemma 7.8 $rate(S_m)^p \leq rate(S_r)^k$. Also, by Observation 7.24 and by Lemma 7.8 after state Q_p, the rate of session S_j can not be decreased below the value of $rate(S_m)^p + \delta$. □

The following observation is a direct consequence of Definition A.1 and Definition A.2.

Observation 7.29 *Let $Q_i \neq Q_{final}$ be a state in the Approximate Global Min algorithm and let S_j be an* active *session in which the rate is minimum among all sessions in $\mathcal{A}^i$. The inequality $\Delta^i(j) \leq \delta$ holds if and only if there is an edge $e \in S_j$ where $\Delta^i(e,j) = \Delta^i(j)$ and*
$Cap(e) - \sum_{S_r \in \{\gamma(e) - S_j\}} PLV(S_r)^i \leq rate(S_j)^i + \Delta^i(e,j) \leq rate(S_j)^i + \delta$. □

The analogous observation is as follows:

Observation 7.30 *In the Approximate Enhanced Local Min algorithm let S_j be an* active *session that is chosen by the algorithm is state Q_i, either to be marked* done *or to be increased. Then, $\Delta^i(j) \leq \delta$ if and only if there is an edge $e \in S_j$ where $\Delta^i(e,j) = \Delta^i(j)$ and*
$Cap(e) - \sum_{S_r \in \{\gamma(e) - S_j\}} LPLV(S_r)^i \leq rate(S_j)^i + \Delta^i(e,j) \leq rate(S_j)^i + \delta$.

Proof: Following Definition A.1 and Definition A.2, there is an edge e such that $Cap(e) = rate(S_j)^i + \Delta^i(e,j) + \sum_{S_r \in \{\gamma(e) - S_j\}} \min(rate(S_r)^i, rate(S_j)^i + \Delta^i(e,j))$. If $\Delta^i(j) \leq \delta$ then, following Definition 7.26 one can see that for every session S_r that shares an edge with S_j, $\min(rate(S_r)^i, rate(S_j)^i + \delta) \leq LPLV(S_r)^i$. □

Thus we get the two concluding theorems.

Theorem 7.31 *Let Q_i be a state in the Approximate Global Min algorithm where S_j is an* active *session in which the rate is minimum. If $\Delta^i(j) \leq \delta$, then for every $l \geq i$, $\Delta^l(j) \leq \delta$.*

Proof: If $Q_i = Q_{final}$ we are done. Otherwise, by Observation 7.29 we have that in state Q_i there is an edge e such that:
$Cap(e) - \sum_{S_r \in \{\gamma(e) - S_j\}} PLV(S_r)^i \leq rate(S_j)^i + \Delta^i(e,j) \leq rate(S_j)^i + \delta$. By Claim 7.6, the first update of $rate(S_j)$ after state Q_i can not be a decrease. Let l be the smallest index such that in Q_l, $l > i$, $\Delta^l(e,j) > \delta$. However, we have that $rate(S_j)^l = rate(S_j)^i$, and hence, following Observation 7.27 we get that the restriction stated in Observation 7.29 that caused $\Delta^i(e,j)$ to be less than δ still holds for $\Delta^l(e,j)$. Thus, as the restriction in Observation 7.29 is of the form "if and only if" we get a contradiction. □

Theorem 7.32 *Let Q_i be a state in the Approximate Enhanced Local Min algorithm where S_j is an* active *session that is* dis3-local-min. *If $\Delta^i(j) \leq \delta$, then for every $l \geq i$, $\Delta^l(j) \leq \delta$.*

Proof: If $Q_i = Q_{final}$ we are done. Otherwise, by Observation 7.30 in state Q_i there is an edge e such that:
$Cap(e) - \sum_{S_r \in \{\gamma(e) - S_j\}} LPLV(S_r)^i \leq rate(S_j)^i + \Delta^i(e,j) \leq rate(S_j)^i + \delta$. By Lemma 7.8, the first update of $rate(S_j)$ after state Q_i can not be a decrease. Let l be the smallest index such that in Q_l, $l > i$, $\Delta^l(e,j) > \delta$. However, $rate(S_j)^l = rate(S_j)^i$, and hence, following Observation 7.28 the restriction stated in Observation 7.30 that caused $\Delta^i(e,j)$ to be less than δ still holds for $\Delta^l(e,j)$. Thus, as the restriction in Observation 7.30 is of the form "if and only if" we get a contradiction. $\square$

8 Acknowledgments

We would like to thank Yair Oren for introducing us to the ATM Forum discussions on flow control. We also thank Moshe Sidi for helpful comments and discussions. We would like also to thank the referees for helpful comments that have improved the presentation.

References

[BF95] F. Bonomi and K. Fendick. The rate based flow control for available bit rate atm service. *IEEE Networks*, 9(2):24–39, March/April 1995.

[BG87] D. P. Bertsekas and R. G. Gallager. *Data Networks.* Prentice Hall, 1987.

[Cha94] A. Charny. An algorithm for rate allocation in a packet-switching network with feedback. Technical Report MIT/LCS/TR-601, MIT Laboratory for Computer Science, April 1994.

[Gaf82] E. Gafni. *The Integration of Routing and Flow Control for Voice and Data in a Computer Communication Network.* PhD thesis, Massachusetts Institute of Technology, Dept. of Elec. Eng. and Comp. Science, Cambridge, MA, August 1982.

[GB84] E. Gafni and D. P. Bertsekas. Dynamic control of session input rates in communication networks. *IEEE Trans. on Automatic Control*, AC-29(11):804–823, November 1984.

[Hay81] H. Hayden. Voice flow control in integrated packet networks. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering, Cambridge, MA, 1981.

[Jaf81] J. M. Jaffe. Bottleneck flow control. *IEEE Trans on Communication*, COM-29,1(7):954–962, July 1981.

[JKV94] R. Jain, S. Kalyanaraman, and R. Viswanathan. The OSU scheme for congestion avoidance using explicit rate indication. Technical Report ATM-FORUM/94-0883, The Ohio State University, September 1994.

[KM95] H.T. Kung and R. Morris. Credit-based flow control for atm networks. *IEEE Networks*, 9(2):40–48, March/April 1995.

[Mos84] J. Mosely. *Asynchronous Distributed Flow Control Algorithms*. PhD thesis, MIT, June 1984.

[Rob94] L. Roberts. Enhanced PRCA (proportional rate-control algorithm). Technical Report ATM-FORUM/94-0735R1, ATM Systems, August 1994.

[ST94] K. Y. Siu and H. Y. Tzeng. Adaptive proportional rate control (APRC) with intelligent congestion indication. Technical Report ATM_FORUM/94-0888, University of California, Irvine, September 1994.

A Formal Description of an update operation

Definition A.1 *Let the triplet (R, j, C) be a vector of session rates $(R_1, R_2, \ldots, R_m) \in R^m$, a session index, and a constant (representing capacity) such that, $1 \leq j \leq m$ and $\sum_{l=1}^{m} R_l \leq C$. The* update *of this set is the pair (R', Δ) such that $R' = (R'_1, R'_2, \ldots, R'_m) \in R^m$ is the new vector of session rates and $\Delta \in R^+$ is the amount by which the rate of session R_j is increased. The update is computed such that it satisfies the following four conditions:*

1. $\sum_{l=1}^{m} R'_l = C$.
2. $R'_j = R_j + \Delta$
3. $\forall l \neq j,\ R_l \ \leq \ R_j + \Delta \ \Rightarrow \ R'_l = R_l$.
4. $\forall l \neq j,\ R_l \ > \ R_j + \Delta \ \Rightarrow \ R'_l = R_j + \Delta$.

Let o_i be an atomic operation such that $S_j = S(o_i)$, and let R be a set of sessions that share an edge e, where $Cap(e) = C$. Hence, the update $\Delta^i(e, j)$ is the maximum increase in S_j that is restricted by e, following the fairness concept (according to which a session is increased as long as this increase does not cause a decrease in another session with equal or lower rate). Notice that the four conditions imply that Δ is well defined and unique. Condition 1 implies that the edge is saturated, condition 2 implies that an amount of Δ flow units is added to the session that is increased, condition 3 implies that sessions with equal or lower rate are not affected and condition 4 guarantees the fairness condition. Notice also that the four conditions imply that Δ is the maximum possible increase in S_j.

The increase in $rate(S_j)^i$ during operation o_i, $\Delta^i(j)$, is the minimum $\Delta^i(e, j)$ over all edges $e \in S_j$.

After Δ^i was calculated, we calculate for each session that shares an edge with $S_j = S(o_i)$, the amount of flow by which it should be reduced, as follows;

Definition A.2 *Let (R, j, C, Δ) be a set such that $R = (R_1, R_2, \ldots, R_m) \in R^m$ is the original vector of session rates, $\{C, \Delta\} \in R$ (C is the capacity of the edge that is shared by these sessions and Δ is the increase in R_j), j be an index $1 \leq j \leq m$ and $\sum_{l=1}^{m} R_l \leq C$. The* restriction *of this set is the set (R', Bar), where $R' = (R'_1, R'_2, \ldots, R'_m) \in R^m$ is the new vector of rates, $Bar \in R^+$ and the following conditions are satisfied:*

1. $\sum_{l=1}^{m} R'_l \leq C$.

2. $R'_j = R_j + \Delta$.

3. $\forall l \neq j,\ R_l \leq Bar \Rightarrow R'_l = R_l$.

4. $\forall l \neq j,\ R_l > Bar \Rightarrow R'_l = Bar$.

Notice that these four conditions imply that Bar^i is well defined and unique. Hence, in operation o_i, $Bar^i(e)$ is the restriction of all sessions that share edge e. Here again we follow the fairness concept and decrease a sessions only if no sessions with higher rate can be decreased. Therefore, $Bar^i(e)$ is an upper bound for the rate of all sessions of $\gamma(e)$ in state Q_{i+1} (and in particular $R_j + \Delta \leq Bar$). Notice that for all $r \neq j$ $rate(S_r)^{i+1}$ is the minimum over $rate(S_r)^i$ and $\{Bar^i(e)\}$ for all $e \in \{S_r \cap S_j\}$.

B Global Min algorithm lower bound example

Recall that Figure 2 is an example of a network, $Net(4, 0, 128)$ as described in Theorem 4.4.

The following scenario is produced by the recursive procedure **Scenario**(4,0,128):

- $rate(S_8)$ is increased from 0 to 64.
- The rates of sessions $S_1, S_2, S_3, S_4, S_5, S_6$ are increased from 0 to 64.
- $rate(S_7)$ is increased from 0 to 32. This causes $rate(S_8)$ to go down from 64 to 32.
- Sessions S_7 and S_8 become *done*. The potential in each of the *active* sessions is 32.
- $rate(S_6)$ is increased from 64 to 80.
- The rates of sessions S_1, S_2, S_3, S_4 are increased from 64 to 80.
- $rate(S_5)$ is increased from 64 to 72. This causes $rate(S_6)$ to go down from 80 to 72.
- Sessions S_5 and S_6 become *done*. The potential in each of the *active* sessions is 8.
- $rate(S_4)$ is increased from 80 to 84.
- The rates of sessions S_1, S_2 are increased from 80 to 84.
- $rate(S_3)$ is increased from 80 to 82. This causes $rate(S_4)$ to go down from 82 to 80.
- Sessions S_3 and S_4 become *done*. The potential in each of the *active* sessions is 1.
- $rate(S_2)$ is increased from 84 to 85.
- $rate(S_1)$ is increased from 84 to 84.5. This causes $rate(S_2)$ to go down from 85 to 84.5.
- Sessions S_1 and S_2 become *done*.

C Example for a Lower Bound in the General Scheduler

Recall that Figure 3 is an example of a network, $Exp\_Net(3, 100, 10)$ as described in Theorem 6.11. The value of array *pot* is calculated by $Exp\_Net(3, 100, 10)$ and has the following values: $pot(1) = 100$, $pot(2) = 90$, $pot(3) = 80$, $pot(4) = 70$ and $pot(5) = 60$.

Note that $H(3) = 2 + 2 \cdot H(2) = 2 + 2(2 \cdot H(1)) = 10$.

We follow the next scenario, **Exp_Scenario**$(3, pot)$:

- $rate(S_1)$ is increased to 100.
- $rate(S_2)$ is increased to 90. This causes $rate(S_1)$ to go down from 100 to 90.

• $rate(S_3)$ is increased to 80. This causes $rate(S_2)$ to go down from 90 to 80.
• $rate(S_1)$ is increased to 100.
• $rate(S_4)$ is increased to 70. This causes $rate(S_1)$ to go down from 100 to 90, and $rate(S_3)$ to go down from 80 to 70.
• $rate(S_5)$ is increased to 60. This causes $rate(S_4)$ to go down from 70 to 60.
• $rate(S_1)$ is increased to 100.
• $rate(S_2)$ is increased to 90. This causes $rate(S_1)$ to go down from 100 to 90.
• $rate(S_3)$ is increased to 80. This causes $rate(S_2)$ to go down from 90 to 80.
• $rate(S_1)$ is increased to 100.