On the Convergence of Rate Based Flow Control

Yehuda Afek Yishay Mansour Zvi Ostfeld
Computer Science Department,
Tel-Aviv University, Israel 69978.

July 15, 1995

Abstract

Several issues in the design and analysis of rate based flow control algorithms are addressed in this paper. We suggest a new approach for rate based flow control algorithms which may be considered more "optimistic" than traditional approaches. Three variations of the approach are presented and their rate of convergence to an optimal max-min fairness solution is analyzed. Finally, we suggest and analyze approximate rate based flow control algorithms. We show that under certain conditions the approximate algorithms may converge faster. However, in rare configurations we show that the resulting rates may be substantially different than the rates of the max-min vector (this difference is not necessarily bad).

1 Introduction

The ATM Forum on Traffic Management has recently adopted rate based flow control [Cha94, Jaf81, Hay81, Mos84, GB84, Gaf82, BG87] as the basis for flow control in its networks (see [Rob94] for example). The major arguments for rate based flow control, is the simplicity and modest hardware requirements of per virtual circuit, compared with those of the credit based scheme. Conceptually, the rate base flow control adjusts transmission rates of different sessions (virtual circuits) in the network in an end-to-end manner aiming to optimize the 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, RM) is looping around the path of each session. 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 [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) will be 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 guideline for this computation. The network is considered to be in a state of *max-min fairness* if it is impossible to infinitesimally increase the rate of any session without decreasing the rate of sessions whose rate is equal or smaller. Thus, the fairness is that each session is allocated the "maximum" rate without affecting sessions with equal or smaller rates.

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 flow 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 restarted. (Thus, we are maximizing the minimum flow and continuing recursively on the remaining circuits, after updating the capacities.)

Most known rate based flow control algorithms adjust the rates of sessions in a conservative way. That is, given a fixed set of sessions, these algorithms converge 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].

In this paper we examine a more aggressive approach, that we call "optimistic". According to this approach, during the computation, sessions may be given a rate that is larger than their final rate. Hence, during the convergence the rates of sessions may be decreased, and not only increased as in the conservative approach. Our theoretical results indicate that in many cases this approach converges faster to the max-min vector of rates.

One problem with most of the conservative approaches 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 (after a new session has been added). Furthermore, the network has some arbitrary set of rates, which should be adjusted, and it would be preferable to perform an

incremental update rather than starting each time from scratch.

The algorithms in this paper are based on an update operation which is applied in a particular order to the different sessions in the network. Informally, in an update operation a particular session on a specific link increases its rate until it becomes the largest session on the link. This is done by computing a certain value as a function of the link capacity and the rates of sessions that share the link. The rate of the chosen session is increased to that value while any session whose rate is above this value and that uses this link, is reduced to this value. The rate of the other sessions that share that link (whose rate is smaller than the computed value) does not change, even if their rate was larger than the previous rate of the chosen session. Thus, after the update operation the rate of the chosen session is the largest rate on that link. For example, if a link with capacity 16 is shared by five sessions with rates $1, 2, 3, 4$, and 6, and the update operation is applied to the second session (whose rate is 2), then the resulting rates are: $1, 4, 3, 4$, and 4. The algorithms considered in this paper, all apply update operations (each in an atomic way) in an iterative way to different sessions. When a session is selected for an update, the maximum increase that is allowed by each of its links is computed, and the minimum of these is applied to the session.

Two parameters distinguish between the different algorithms considered here. One is the procedure that selects the next session that is increased (i.e., the scheduler), and the other 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).

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 session that is selected to be increased is an *active* session with the global minimum rate, (2) A *local min* scheduler in which the session that is selected to be increased is an *active* session in which the rate is smallest among all the *active* sessions it shares a link with, 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.

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 at least δ. The approximate algorithms reach quiescence when no session may be increased by at least 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 calculate the exact max-min vector, 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 an $O(\Phi^n)$ number of update operations are shown to be sufficient to reach the optimal max-min vector(where $\Phi = \frac{\sqrt{5}+1}{2}$ is the golden ratio). $\Omega(2^{\frac{n}{2}})$ number of update operations are shown to be necessary in an example scenario.

3. Any approximate algorithm (any algorithm that refrains from updates that are smaller than δ), might quiesce at a state in which the rate of at least one session is smaller or larger than its rate in the optimal max-min vector by $O(\delta \cdot 2^n)$. An example of a configuration in which a gap of $\Omega(\delta \cdot 2^{\frac{n}{2}})$ exists is provided, thus proving that this is a lower bound on the possible gap between the rates computed by any approximate algorithm and the rates of the max-min vector. Although this property is exhibited only under peculiar conditions, it is shared by any algorithm that refrains from small updates (i.e., it is not a property of the algorithm but a property of the network configuration). Furthermore, this lower bound phenomena may be caused by a possible gap of δ in only one session. Note however that this phenomena is not necessarily bad, since in particular, the removal or addition of a small session to the max-min vector could have the same effect. Note that one of the implications of the results mentioned in this paragraph is the settlement of an open question raised by Charny [Cha94]. (Where she asks about the convergence of rate based algorithms if the rates are restricted to discrete values.)

4. If the number of sessions that are shared by 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 scheduler) reaches quiescence in $O(n \cdot \log \frac{W}{\delta})$ update operations, where W is the maximum capacity of any edge in the network (a refined value for W will be given in Section 7).

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 be aware of, in which either the convergence to the max-min vector will be very slow, or the possibility in which a small gap in the rate of one session may cause a large gap in the rates of other sessions.

To enable the clean analysis initiated in this paper we made the assumption that the rates of sessions are updated atomically (sequentially). That is, that an update operation is

not performed until the previous one has finished. This is an approximation of the behavior of distributed rate based algorithms. The analysis of rate based algorithms without this assumption is beyond the scope of this paper and is left as an open subject for further research.

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. Implementation issues are discussed in Section 8.

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 use 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 maximal allocation in which 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 appear also in [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 $x^k > y^k$. 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 is called *max-min vector* if it is a feasible allocation which is lexicographically maximum (among all feasible allocation vectors) with respect to the sort order (i.e. for every feasible allocation y, we have that $sort(x) \geq_{lex} sort(y)$). The goal of our algorithms is to compute a max-min vector x.

2.1 Algorithms' Progress

The following definitions and notations apply to all the algorithms for computing the set of transmission rates that are considered in this paper.

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 the rate of one session, S_i, is increased and the rate of other sessions that share an edge with S_i may be decreased. A discussion about the implementation of this update operation appears in Section 8.

An update operation is performed on session S_i only if there is a way to increase $rate(S_i)$ up to a new rate value $nrv > rate(S_i)$, while the rate of any other arbitrary session S_j is either unaffected or decreased to a new value which is greater or equal to nrv. Given a vector of rates x, the outputs of the update operation on S_i will result in a maximal increase in the rate of S_i without violating the above condition. (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, and as a result of this update is the following new 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 where there is only a single edge. In general a session may have many edges, and it can be increased, only if it can be increased on *every* edge. The update, by definition, would be the minimum update over all the edges (see Appendix A). Note that any session that shares an edge with the increased session and whose rate was larger than nr may be reduced in a fair manner to a value that is larger or equal to nr.

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 $(35, 35, 35, 75)$.

Our algorithms start at some initial allocation of rates and converge to a max-min vector. In the processes the algorithms mark the sessions that reached their final rate value 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 sessions are *active* (i.e. $\mathcal{D} = \phi$). The algorithms quiesce when all the sessions are *done* ($\mathcal{A} = \phi$).

Each execution of an algorithm is modeled as a finite or infinite 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* marking are used by the scheduler in order to decide which session is selected to be increased in the next operation. However, this marking is not used

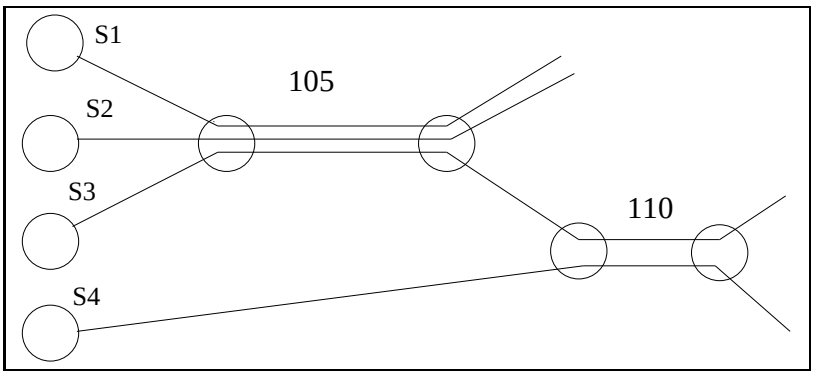

Figure 1: An example for an update operation

by the computation of the new rate values. We denote by $S(o_i)$ the session that operation o_i is increasing. 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.

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.,

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

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 present and prove several basic properties of bottleneck algorithms. Let us first discuss when a session is marked *done*.

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 $S_m \in \gamma(e) \cap \mathcal{A}^i$, $FS^i(e)$ is the smallest fair share among the edges of S_m [BG87].

Definition 3.1 *An algorithm that computes the exact max-min vector for a set of sessions, is called* general bottleneck algorithm *if in any state Q_i of it, a session S is marked* done*(i.e. has reached its final rate) only if there is an edge $e \in S$ that is a bottleneck edge 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 in a fair manner.) When all the sessions satisfy this condition (i.e., marked *done*) the algorithm may quiesce.

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

Definition 3.2 *A* bottleneck algorithms *is a* general bottleneck algorithms *that in addition, confirms to the convensions 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 our 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 $\mathcal{A}^i = \phi$ then the rate of every session is the same as its rate in the max-min vector.* □

In the next lemma we show 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 a session that uses e becomes *done* in the last step. This increases $allotted(e)$ and decreases $N(e)$. If the rate of the *done* session equals $FS^i(e)$ then it does not effect the Fair Share. Otherwise, the *done* session must get less than $FS^i(e)$ and thus the other 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, $N^{i+1}(e) < N^i(e)$ and in state Q_i there is a session $S_j \in \{\mathcal{A}^i \cap \gamma(e)\}$ with an edge $e' \in S_j$ such that $rate(S_j)^i = FS^i(e') \leq FS^i(e)$. Then, by the definition of *Fair Share*, it follows that:

$$\begin{aligned} FS^{i+1}(e) &= \frac{Cap(e) - allotted^{i+1}(e)}{N^{i+1}(e)} \\ &= \frac{Cap(e) - allotted^{i}(e) - rate(S_j)^i}{N^{i}(e) - 1} \\ &\geq \frac{Cap(e) - allotted^{i}(e) - \frac{Cap(e) - allotted^i(e)}{N^i(e)}}{N^{i}(e) - 1} = FS^i(e) \end{aligned}$$

□

The next 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 is applied, 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 $S_j \in \mathcal{A}^i$. Then,*

$$LB_j{}^i = \min_{e \in S_j}\{FS^i(e)\}$$

($LB_j{}^i$ is the smallest Fair Share along the path of S_j in state Q_i.)

Our goal now is to prove Theorem 3.9 in which we state that if the rate of an arbitrary session S_j is updated 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 after each increase in the rate of a session, its rate is still larger than ${LB_j}^i$. Secondly, in Lemma 3.8, we prove 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 where $FS^l(e') = {LB_j}^l$. 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, ${rate(S_j)}^{i+1} \geq FS^i(e') \geq {LB_j}^i$. □

As we have shown that after every update of the rate of a session the new rate is above the current *Fair Share* of at least one of its edges then, since the *Fair Share* is non decreasing we have thus the following theorem:

Theorem 3.9 *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 $l \geq i$ ${rate(S_j)}^l \geq {LB_j}^i$.* □

4 Global Min Scheduler

In this section we analyze a bottleneck algorithm that in each step selects to increase the globally smallest *active* session. The following rules are used in each state of the algorithm:

Marking a session done*:* If the smallest rate *active* session in the network can not be increased then this session is marked *done*.

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

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

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] (in a different model).

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 $\mathcal{A}^l \subsetneq \mathcal{A}^i$.*

Proof: Let $e \in S_j$ such that $FS^i(e) := LB_j{}^i$. By Theorem 3.9, for all $k > i$, $rate(S_j)^k \geq FS^i(e)$. If $rate(S_j)^l$ is global minimum of all sessions in $\mathcal{A}^l$ it is in particular smallest in all

sessions of $\mathcal{A}^l \cap \gamma(e)$ and hence, $rate(S_j)^l \leq FS^l(e)$. If $\mathcal{A}^l = \mathcal{A}^i$ then $FS^l(e) = FS^i(e)$ which implies that all sessions in $\mathcal{A}^l \cap \gamma(e)$ have the same rate in Q_i and Q_l and can not be increased. Hence, all sessions in $\mathcal{A}^l \cap \gamma(e)$ 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 $\mathcal{A}^i \neq \phi$ the algorithm progresses since in state Q_i every *active* session in which the rate is global minimum can be either increased or transfered to $\mathcal{D}$. Following Lemma 4.2, for all $i > 0$, if $|\mathcal{A}^i| = h$ then $|\mathcal{A}^{i+h}| \leq h - 1$. Since the algorithm starts at state Q_0 where $|\mathcal{A}^0| \leq n$ and terminates in state Q_k where $|\mathcal{A}^k| = 0$ the total number of update operations in the algorithm is at most $\sum_{h=n}^{1} h = \frac{n(n-1)}{2}$. □

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 *potential* is an upper bound for the amount of flow (either temporary of final) that any session can achieve in the future. Figure 2 is 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 where $Cap(e) = 2 * bottom + potential$.
2. If $m > 1$ then
   (a) S_{2m-1} and S_{2m} share a common edge e where $Cap(e) = 2 * bottom + \frac{potential}{2}$.
   (b) construct configuration **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 $Cap(e) = 2 * bottom + potential$.

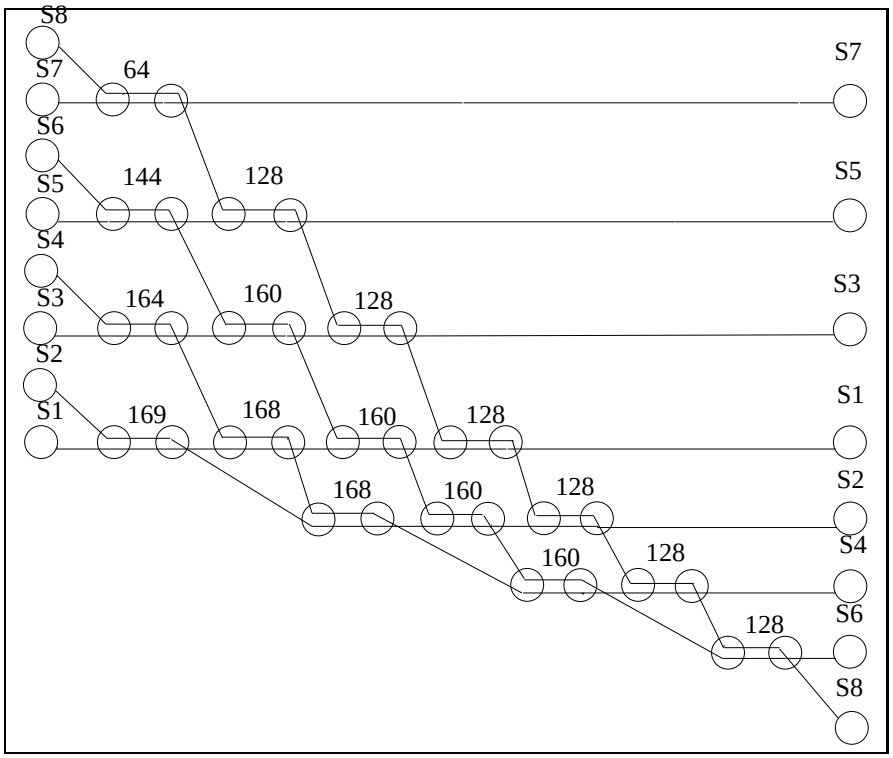

Figure 2: An example of a network $Net(4, 0, 128)$

We build the network by calling the procedure **Net**$(m, 0, P)$, with a large constant P.

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 achieve 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 $bottom + \frac{potential}{2}$. Sessions S_1 and S_2 become *done*.
2. If $m > 1$ then
   (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 $bottom + \frac{potential}{4}$. Sessions S_{2m-1} and S_{2m} become *done*.
   (d) The scenario is recursively extended by calling **Scenario**$(m-1, bottom + \frac{potential}{2}, \frac{potential}{4})$ on the network $Net(m-1, bottom + \frac{potential}{2}, \frac{potential}{4})$.

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 first 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 it follows the following rules:

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

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 (i.e. $S_j \cap S_m \neq \phi$) , $rate(S_m)^i$ is smaller or equal to the rate of all the *active* sessions S_r that share an edge with S_m (which means that S_m is also a local minimum).

Session selected for increase: The scheduler chooses an *active* session $S_j \in \mathcal{A}^i$ such that $rate(S_j)^i$ is smaller than or equal to the rates of all the sessions it shares an edge with. I.e., for every *active* session $S_m \in \mathcal{A}^i$ if there is an edge $e \in \{S_j \cap S_m\}$ then $rate(S_j)^i \leq rate(S_m)^i$. If the rate of the session can be increased then S_j is selected as $S(o_i)$.

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

This algorithm seems somewhat more natural for implementation in a distributed environment because its progress is done according to information that can be calculated locally (in contrast to the Global Min algorithm in which the progress depends upon a knowledge of the rates of all the *active* sessions in the network). 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 confirm 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 in 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. □

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

Proof: Since the global minimum is also a local minimum the proof of Theorem 4.3 applies verbatim. □

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: 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* if there is an edge $e \in S_j$ that is a bottleneck edge, 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: The scheduler selects an arbitrary session S_j that can be increased and sets $S(o_i)$ to be S_j.

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

Note the following:

1. The arbitrary scheduler is a bottleneck algorithm.

2. The restriction that the session S_j selected by the scheduler can be increased implies that $S_j \in \mathcal{A}^i$

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

4. 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 is very natural and simple to implement because it is:

1. Distributed - the rate of a session can be increased without any consideration whether it is local or global minimum.

2. Greedy.

3. Dynamic - if we ignore termination detection then sessions can be added to the network without any initialization process or removed from it without alerting other sessions.

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 6.9) for an infinite family of networks.

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.

Observation 6.1 *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 $\{S_1\ ,\ S_2\ \ldots\ S_k\}$ (where $k \leq n$) be the sessions for which the rate has been modified (either increased or decreased) at least once. Then, the number of update operations in G is less than or equal to $F(k)$.*

Proof: We build a network called $G' = (V', E')$ as follows:

1. $E = S_1 \cup S_2 \cup \ldots \cup S_k$.

2. For every edge $e' \in E'$ we set $Cap(e')$ to be $Cap(e) - \sum_{j=k+1}^{n} rate(S_j)^0 \times (S_j \in \gamma(e))$, where $e \in E$ is the analogue edge to e' and $(S_j \in \gamma(e))$ is an operation that returns 1 if $S_j \in \gamma(e)$ and 0 otherwise. (For each edge we subtract the total amount of flow of sessions for which the rate is not modified.)

One can see that every sequence of update operations in G can be simulated by an analogue sequence of update operations in G'. □

Based on that observation, we will give a recursive function for the analysis of the upper bound for the maximum number of update operations.

We now give some definitions that are used in the statements that will follow.

Definition 6.2 *For every session S_j we define $bound(S_j)$ to be an upper bound for $rate(S_j)$, i.e., for all $i \geq 0$ $bound(S_j) \geq rate(S_j)^i$*

The following definition is an extension for the term Fair Share:

Definition 6.3 *For every edge e and for every session $S_j \in \mathcal{A}^i$, $PFS^i(e,j)$, the Potential Fair Share in state Q_i, is defined as follows:*
If $S_j \in \gamma(e)$ 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 6.4 *Let $S_j \in \mathcal{A}^i$ and let $S_r \neq S_j$. Then,*

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

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

One can see that after any update in the value of $rate(S_j)$ the new value is above the current value of $MLB_j{}^i(r)$ for every $S_r \neq S_j$ (by similar arguments to those in Lemma 3.7 and in Lemma 3.8). Since the *Potential Fair Share* is non decreasing (by similar arguments to those in Lemma 3.4), we have the following extension for Theorem 3.9.

Theorem 6.5 *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 $S_r \neq S_j$, $rate(S_j)^l \geq MLB_j{}^i(r)$.*

□

Definition 6.6 *Let G be a network with n sessions $\{S_1, S_2, \ldots, S_n\}$. For every state Q_i of the algorithm $prefix(i)$ is defined to be the set of sessions that were updated at least once during the first i operations.*

Lemma 6.7 *Let G 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 for all states Q_l, where $l \geq i$ (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 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 3.9, 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 $rate(S_r)^i = bound(r) = Cap(e)$. We Observe now all the edges except those where $\gamma(\bar{e}) = S_r$ and select an edge e', where $PFS^i(e', r)$ is minimum among all those edges. 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 achieves an upper bound on the number of update operations.

Theorem 6.8 *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 $k > 2$, $F(k+1) \leq F(k) + 1 + F(k-1)$.

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:
$|prefix(l-1)| \leq k$ (i.e., only k sessions were updated during the first l operations) and $|prefix(l)| = k+1$ (i.e., Q_l is the first time when the $k+1-st$ session is updated).

By Observation 6.1 the first part may take no more than $F(k)$ operations. By Lemma 6.7 in Q_{l+1} there are at least two sessions whose rate have reached the final value. Hence, in the third part we have a network with no more than $k-1$ sessions that can be changed, which implies (by Observation 6.1) that this part may take at most $F(k-1)$ additional update operations. □

The following theorem states a lower bound for the algorithm.

Theorem 6.9 *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 for the amount of flow (either temporary of final) that any session can achieve in the future. and $progress$ is a parameter that is used to decrease $potenial$i. 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 $potential > 2 \cdot m \cdot progress$.

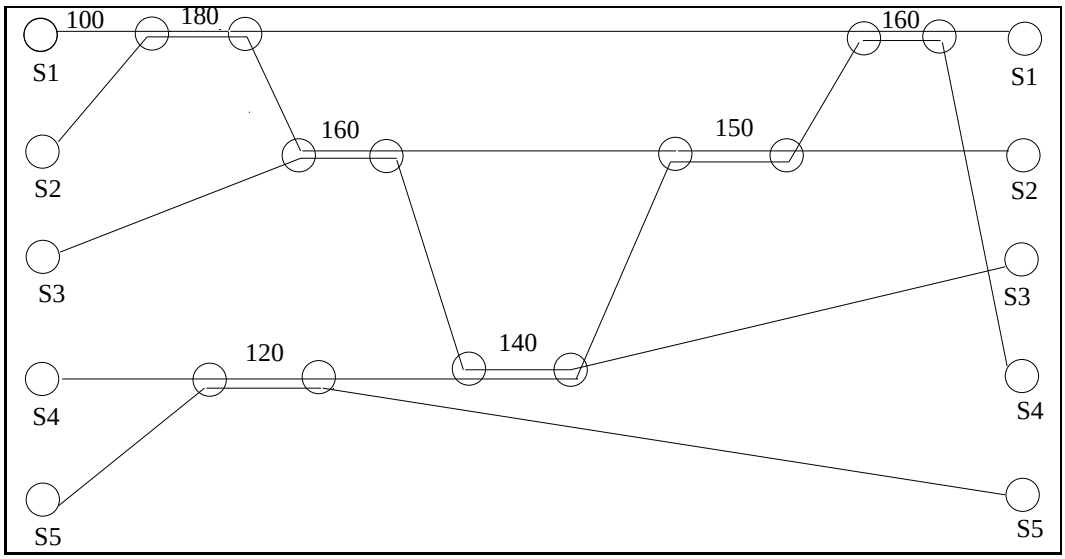

Figure 3: An example of a network $Exp_Net(3, 100, 10)$

The procedure that builds the network is as follows:

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

S_1 use solely an edge e where $Cap(e) := potential$
$pot(1) := potenial$
$potential := potential - progress$
for i:=2 to $2 * m - 1$ do
begin
    $pot(i) := potential$
    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 }
S_i and S_{i-1} share an edge e where
$Cap(e) := pot(i) + pot(i-1) - progress$
$potential := potential - progress$
end { of for }

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 achieve the required number of operations. Note 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 where 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 be $pot(2m-2)$. This cause 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 causes the rates of session S_{2m-2} to be decreased to $pot(2m-1) - progress$.
   Now for all the sessions S_j, where $j < 2m-2$, the rate is $pot(j) - progress$ and there is no restriction imposed by S_{2m-2} that disables them to be increased to the vale $top(j)$. Hence, we can repeat step (a).
   (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 $progress = 10$.

7 Approximations

In this section we present algorithms in which a session rate is increased only if the increase is larger than δ. The termination of such an algorithm is when for every session the largest possible improvement is less than or equal to δ. More formally, we say that for every i, j, either $rate(S_j)^{i+1} \leq rate(S_j)^i$ or $rate(S_j)^{i+1} - rate(S_j)^i > \delta$ and in the final state Q_{final} of the algorithm, for every session S_j, $\Delta^{final}(j) \leq \delta$. The motivation for studying this issue is that in "real" dynamic networks, the computation of the exact max-min vector may not be achieved (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 converges to the "normal" final state may be considered better than another algorithm in which it takes longer to reach the new Q_{final} state even if it converges faster to the "normal" final state.

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 then provide an example of a configuration in which one session may be increased by δ and no other session may be increased. However, if that particular session is increased, then we get a scenario which results in a state where some other sessions may be increased by $\Omega(\delta \cdot 2^{\frac{n}{2}})$

Note that this potential "bad" property of the network can not be overcome by any approximation algorithm for rates 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 can be used to show that the addition or deletion of a small rate session from the max-min vector may also cause a dramatic (exponential in the number of sessions) change in the value of some other rates. Thus, the max-min vector is not necessarily better than the vector computed by an approximate algorithm (it may suffer from huge instability that may result from a modest dynamic update operation). 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 $O(n \cdot \log_{\frac{h}{h-1}} \frac{\mathcal{W}}{\delta})$ 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

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

(i.e., for each session we choose the edge which have minimum weight and $\mathcal{W}$ is the maximum weight of all those edges). Hence, if h is bounded by a constant then the number of operations is no more than $O(n \ \log \frac{\mathcal{W}}{\delta})$.

7.1 Analysis

In this subsection we analyze how far the rate of a session can be from its rate in the max-min vector if rates are computed by an approximate algorithm.

We consider an arbitrary approximate algorithm for computing fair flow where Q_{final} is the final state as described above. We use the following definitions:

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

$\bar{Y} = sort(X)$: A non descending order permutation of the coordinates of X (such that $y^i \leq y^{i+1}$ for $i = 1, \ldots, n-1$). If y^i corresponds to x^j we say that $i = perm(j)$ and $j = perm^{-1}(i)$.

$\bar{Z} = (z^1, z^2, \ldots, z^n)$: The flow rate 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.

$\bar{W}$: The differences between Y and Z. $\forall i$, $w^i = y^i - z^j = x^j - z^j$ where $i = perm(j)$.

$\bar{W}_{neg}$: Represents the sessions whose rate in $\bar{Z}$ is larger than their value in the max-min vector, i.e., if $w^i < 0$ then $w^i_{neg} = w^i$. For the other sessions (where the rate in $\bar{Z}$ is less or equal to the value in the max-min vector) $w^i_{neg} = 0$. Formally, $w^i_{neg} = \min\{w^i, 0\}$.

$\bar{W}_{pos}$: Represents the sessions whose rate in $\bar{Z}$ is less than the value in the max-min vector i.e., if $w^i > 0$ then $w^i_{pos} = w^i$. For the other sessions (where the rate in $\bar{Z}$ is greater or equal to the value in the max-min vector) $w^i_{pos} = 0$. Formally, $w^i_{pos} = \max\{0, w^i\}$.

In state Q_{final}, the rate of session S_j $(= z^j)$ is w^i away from its value in x^j (where $j = perm^{-1}(i)$). The following lemma states that z^j is below x^j by at most δ plus the amount of flow by which sessions with smaller index in $\bar{Y}$ are above their rate in the max-min vector. The intuition is that $rate(S_j)^{final}$ 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 those other sessions will be decreased by other restrictions,

Lemma 7.1 *For all* $i \geq 1$, $w^i \leq \delta + \sum_{l<i} -w_{neg}{}^l$.

Proof: Let $j = perm^{-1}(i)$. Clearly, since S_j can not be increased by more than δ then there is an edge $e \in S_j$, where $\Delta^{final}(e,j) \leq \delta$. Hence, the maximum difference between $rate(S_j)^{final}$ and the value of S_j in the fair vector is composed of two sources:

1. $rate(S_j)^{final}$ may be increased by (at most) $\Delta^{final}(e,j)$ and becomes maximum rate among all sessions in $\gamma(e)$.
2. $rate(S_j)$ may be increased by the sum of flow by which other sessions in $\gamma(e)$ would be decreased.

Let z'^k be the rate of session $S_k \in \gamma(e)$ after performing the first increase. Clearly,

1. $\sum_{S_k \in \gamma(e)} z'^k = Cap(e)$.
2. $z'^j \geq z'^k$ for all $S_k \in \gamma(e)$.
3. $z'^j \leq z^j + \delta$.
4. $z'^k \leq z^k$ for all $S_k \in \{\gamma(e) - S_j\}$.

Hence, one can see that

$$\begin{array}{lcll} \sum_{S_k \in \gamma(e)} x^k & \leq & Cap(e) & \Rightarrow \\ 0 & \leq & sum_{S_k \in \gamma(e)}(z'^k - x^k) & \Rightarrow \\ x^j - z'^j & \leq & \sum_{S_k \in \{\gamma(e) - S_j\}}(z'^k - x^k) & \Rightarrow \\ w^i & \leq & \delta + \sum_{l<i} -w_{neg}{}^l & \end{array}$$

Note that the fourth inequality follows from the third due to the fact that l should be $perm(k)$ and the fact that after the first step $rate(S_j)$ is maximum among the sessions in

$\gamma(e)$. Hence, the restriction of $Cap(e)$ and max min fairness concept implies that the increase of $rate(S_j)$ in the second step may be no more than the amount of flow that was released from other sessions in $\gamma(e)$. However, if a session $S_k \in \gamma(e)$ has released flow (due to any restriction) then its rate value was decreased (i.e. $x^k < z'^k$) and since $x^j > z^j \geq z^k$ the sessions that are decreased in $\gamma(e)$ has a rate in the fair vector which is less then y^i.

□

The following lemma is the analogous for the amount of flow in which z^j may be above y^i. This amount is at most the amount of flow by which sessions with smaller index in Y are below their rate in the max-min vector. The intuition is that in the adjustment process from Z to X, a decrease in $rate(S_j)^{final}$ 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_j.

Lemma 7.2 *For all $i \geq 1$, $w^i \geq \sum_{l<i} -{w_{pos}}^l$.*

Proof: Let $i = perm(j)$. Clearly, there is an edge $e \in S_j$, where

1. $\sum_{S_k \in \gamma(e)} x^k = Cap(e)$.
2. $x^j \geq x^k$ for all $S_k \in \gamma(e)$.

Since $\sum_{S_k \in \gamma(e)} z^k \leq Cap(e)$ (e is a bottleneck edge of S_j) we have that:

$$\begin{array}{lcll} 0 & \leq & \sum_{S_k \in \gamma(e)} x^k - z^k & \Rightarrow \\ z^j - x^j & \leq & \sum_{S_k \in \{\gamma(e) - S_j\}} x^k - z^k & \Rightarrow \\ -w^i & \leq & \sum_{l<i} {w_{pos}}^l & \Rightarrow \\ \sum_{l<i} -{w_{pos}}^l & \leq & w^i & \end{array}$$

Note that the third inequality follows from the second due to the fact that j is $perm(l)$ and the fact that $y^i \geq y^l$ for all $S_l \in \gamma(e)$ and hence, the maximum amount by which x^j was decreased is the amount of the rate in which the other sessions in $\gamma(e)$ were increased. □

Theorem 7.3 *For all $i \in \{1, \ldots, n, \}$ $|w^i| \leq \delta \cdot 2^{i-1}$.*

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

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 the session rate in the max-min vector. The next theorem, Theorem 7.4, shows that a difference of at least $\delta \cdot 2^{n-1}$ is actually possible.

The following theorem states a lower bound for the possible difference between $rate(S_j)^{final}$ and the rate of S_j in the max-min vector.

Theorem 7.4 *For all* $1 \leq i \leq n$, $|w^i|$ *can be* $\Omega(\delta \cdot 2^{\frac{i}{2}})$. *Hence,* $|w^n|$ *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 of the form $4m+3$. 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 $1 \leq i \leq \frac{n-1}{2}$ 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 essentially 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 $m \geq 1$, there are four more edges, $e_{4m-1}, e_{4m}, e_{4m+1}, e_{4m+2}$ that are shared by $\{S_{4m-2}, S_{4m-1}, S_{4m}\}$, $\{S_{4m-2}, S_{4m-1}, S_{4m+1}\}$, $\{S_{4m}, S_{4m+1}, S_{4m+2}\}$ and by $\{S_{4m}, S_{4m+1}, S_{4m+3}\}$ respectively.
4. $Cap(e_1) = 8 \cdot C$ and $Cap(e_2) = 9 \cdot C$, where C is an arbitrary constant.
5. $\forall i, 1 < i \leq 2m+1, Cap(e_{2i-1}) = Cap(e_{2i}) = 2^{i+2} \cdot C$.

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

The adjustment of the rates is as follows:

1. S_1 is increased by δ (from 0 to $3C$ because of the restrictions of e_2.
2. S_2 and S_3 are decreased by C (from $4C$ to $3C$, because of the restrictions of e_2).

3. S_4 and S_5 are increased by $2C$ (from $8C$ to $10C$ because of the restrictions of e_3 and e_4).

4. S_6 and S_7 are decreased by $4C$ (from $16C$ to $12C$).

In general, S_{4i} and S_{4i+1} are increased by $2^{2i-1} \cdot C$ while S_{4i+2} and S_{4i+3} are decreased by $2^{2i} \cdot C$. As $C = 1/3 \cdot \delta$ we get the following difference vector
$\bar{W} = 1/3 \cdot \delta \cdot (3, -1, -1, 2, 2, -4, -4, 8, 8, -16, -16, 32, 32, \ldots)$. Hence, for all i, $|w^{2i}| = \delta \cdot 2^i/6$.
□

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

7.2 Convergence of the Approximate Global Min algorithm

Here we analyze the convergence of the Approximate Global Min algorithm. In this 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.

Before giving the formal definitions of the algorithm, we prove some claims and lemmas that will enable us to obtain the theorem about the convergence of the algorithm.

The next claim states that in the Global Min algorithm the rate of the session with the minimum rate is a non decreasing function.

Claim 7.5 *In the Global Min algorithm, $\forall l, i$, 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 that is decreased in o_i $rate(S_m)^{i+1} \geq rate(S_j)^{i+1} > rate(S_j)^i$. Also, for an *active* session S_m that is not decreased, $rate(S_m)^{i+1} = rate(S_m)^i \geq rate(S_j)^i$ (see Figure 4 for an illustration). Therefore, $rate(S(o_i))^i \leq rate(S(o_{i+1}))^{i+1} \leq \ldots \leq rate(S(o_l))^l$. □

Corollary 7.6 *In the 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 $l \geq i$:*

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

□

For a saturated edge, the following corollary states an upper bound on the amount of rate a session can achieve.

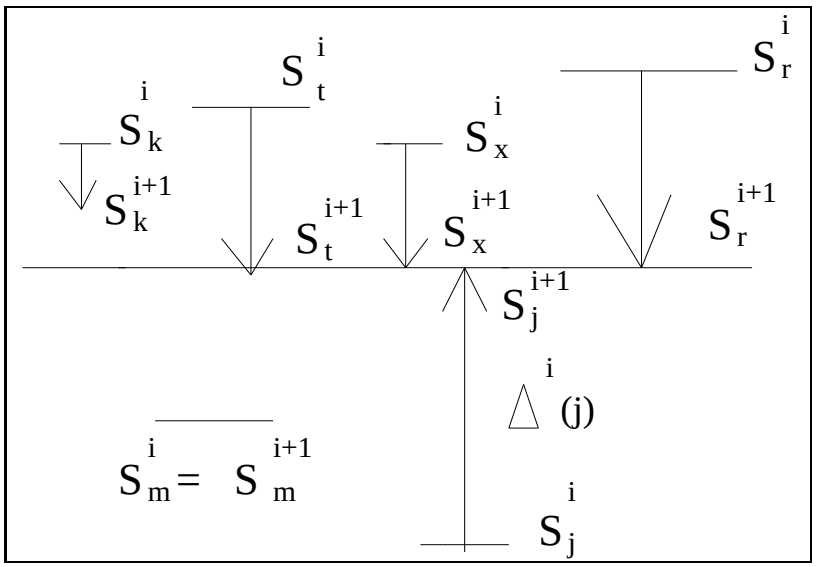

Figure 4: An illustration for the increase in the value of the global minimum

Corollary 7.7 *In the Global Min algorithm, for every $i \geq 0$ let e be an edge in G where $\sum_{S_m \in \gamma(e)} rate(S_m)^i = Cap(e)$, and let $S_t \in \{\gamma(e) \cap \mathcal{A}^i\}$ such that $rate(S_t)^i \geq rate(S_m)^i$ for all $S_m \in \gamma(e)$. Then for all $S_j \in \{\gamma(e) \cap \mathcal{A}^i\}$ and for all $l \geq i$ we have $rate(S_j)^l - rate(S_j)^i \leq (h'-1) \cdot (rate(S_t)^i - rate(S(o_i))^i)$, where $h' = |\gamma(e)|$.*

Proof: By assumption,

$$rate(S_j)^i + \sum_{S_m \in \{\gamma(e) - S_j\}} rate(S_m)^i = Cap(e)$$

By the capacity constraints,

$$rate(S_j)^l + \sum_{S_m \in \{\gamma(e) - S_j\}} rate(S_m)^l \leq Cap(e)$$

Therefore,

$$\begin{aligned} rate(S_j)^l - rate(S_j)^i &\leq \sum_{S_m \in \{\gamma(e) - S_j\}} (rate(S_m)^i - rate(S_m)^l) \\ &\leq (h'-1) \cdot (rate(S_t)^i - rate(S(o_l))^l) \\ &\leq (h'-1) \cdot (rate(S_t)^i - rate(S(o_i))^i) \end{aligned}$$

□

For every session S_j we observe the list $j_k, k = 1, 2, \ldots$, where $S(o_{j_k}) = S_j$. We denote $P_j(k)$ to be max $\{rate(S_j)^l - rate(S_j)^{j_k}\}$ for all $l \geq j_k$, 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 7.8 *In the 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: Let $l > j_{k-1}$ be 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 Definition A.2, in state Q_l there is one edge $e \in S_j$ that is saturated and $rate(S_j)^l \geq rate(S_m)^l$ for all $S_m \in \gamma(e)$. Let $h' = |\gamma(e)|$. By Claim 7.5, $rate(S(o_l))^l \geq rate(S(o_{j_{k-1}}))^{j_{k-1}} = rate(S_j)^{j_{k-1}}$ and hence (by Corollary 7.7 and the fact that $h' \leq h$):

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

□

In the coming statements we assume that $h \geq 2$ is the maximum number of sessions that share a single edge.

Observation 7.9 *In the Global Min algorithm, for* $k = 3, 4, \ldots$,
$P_j(k) \leq \min\{(h-1)A - B, (h-1)B\}$ *where* $A = (rate(S_j)^{j_{k-1}} - rate(S_j)^{j_{k-2}})$ *and* $B = (rate(S_j)^{j_k} - rate(S_j)^{j_{k-1}})$.

□

Corollary 7.10 *In the Global Min algorithm, For* $k = 3, 4, \ldots$, $P_j(k) \leq \frac{(h-1)}{h} P_j(k-1)$.

□

Lemma 7.11 *In the Global Min algorithm, For all* $k > \log_{\frac{h}{(h-1)}} \frac{P_j(2)}{\delta}$, $P_j(k+1) < \delta$.

Proof: The lemma follows immediately from Corollary 7.10. □

We give now a formal description of the Approximate Global Min algorithm. In each state of the algorithm it follows the following rules:

Marking a session done*:* If the smallest rate *active* session in the network can not be increased by more than δ, then this session 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 $\mathcal{A}^i = \phi$.

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

Theorem 7.12 *In the Approximate Global Min algorithm the number of increase operations for each session is* $O(\log_{\frac{h}{(h-1)}} \frac{\mathcal{W}}{\delta})$ *where*

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

and h *is the maximum number of sessions that share a single edge (i.e., for each session we choose the edge which have minimum weight and* $\mathcal{W}$ *is the maximum weight of all those edges).*

Proof: The difference between the Global Min algorithm and the Approximate Global Min algorithm does not affect any of the proofs of the claims in Claim 7.5 through Lemma 7.11 and they apply verbatim to the Approximate Global Min algorithm. Note that for each session S_j, $P_j(2) \leq Cap(e)$ for all edges $e \in S_j$. Hence, by Lemma 7.11, for all $k > \log_{\frac{h}{(h-1)}} \frac{P_j(2)}{\delta}$ and for all $l > j_k$ we have $\Delta^l(j) \leq P_j(k+1) < \delta$. □

We thus get the following:

Theorem 7.13 *The Approximate Global Min algorithm quiesces after* $O(n \log_{\frac{h}{(h-1)}} \frac{\mathcal{W}}{\delta})$ *update operations where*

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

and h *is the maximum number of sessions that share a single edge.*

□

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

Finally, we show that in Q_{final}, when the Approximate Global Min algorithm reaches quiescence, no session rate may be increased by more than δ (without affecting the fairness concept). To show this, we show that after a session becomes *done* there is no state in which it may be increased by more than δ.

Observation 7.14 *In the Approximate Global 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 $rate(S_j)^{i+1} \geq rate(S(o_i))^{i+1} > rate(S(o_i))^i + \delta$. □

Definition 7.15 *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* $\min(rate(S_j)^i, rate(S(o_i))^i + \delta)$.

Observation 7.16 *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 7.5 $rate(S(o_l))^l \geq rate(S(o_i))^i$. Hence, if $rate(S_j)^l \geq rate(S_j)^i$ we are done. Else, let $l \geq m \geq i$ be such that Q_m is the last state before Q_l where $rate(S_j)$ was decreased (i.e. $rate(S_j)^{m+1} < rate(S_j)^m$). Then, by Observation 7.14

$$\begin{aligned} rate(S_j)^l &\geq rate(S_j)^{m+1} \\ &\geq rate(S(o_m))^m + \delta \\ &\geq rate(S(o_i))^i + \delta. \end{aligned}$$

since $rate(S(o_l))^l \geq rate(S(o_i))^i$ then $PLV(S_j)^l \geq rate(S(o_i))^i + \delta \geq PLV(S_j)^i$. □

Observation 7.17 *Let Q_i be a state in the Approximate Global Min algorithm and let S_j be an* active *session in which the rate is minimum. $\Delta^i(j) \leq \delta$ if and only if there is an edge $e \in S_j$ where $\Delta^i(e,j) = \Delta^i(j)$ and*
$Cap(e) - \sum_{S_r \in \{\gamma(e) - S_j\}} PLV(S_j)^i \leq rate(S_j)^i + \Delta^i(e,j) \leq rate(S_j)^i + \delta$.

Proof: Follows immediately from Definition A.1. □

Theorem 7.18 *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 $l > i$ $\Delta^l(j) \leq \delta$.*

Proof: By Observation 7.17 we have that in state Q_i there is an edge e where $Cap(e) - \sum_{S_r \in \{\gamma(e) - S_j\}} PLV(S_j)^i \leq rate(S_j)^i + \Delta^i(e,j) \leq rate(S_j)^i + \delta$. By Claim 7.5, the first update of $rate(S_j)$ after state Q_i can not be a decrease. Let $l \geq i$ where Q_l is the first state after i when $\Delta^l(e,j) > \delta$. However, we have that $rate(S_j)^l = rate(S_j)^i$, and hence, following Observation 7.16 we get that the restriction stated from Observation 7.17 that caused $\Delta^i(e,j)$ to be less than *delta* still exists for $\Delta^l(e,j)$. Thus, as the restriction in Observation 7.17 is of the form "if and only if" we get a contradiction. □

7.3 Convergence of the Approximate Enhanced Local Min algorithm

Before giving the definition of the Approximate Enhanced Local Min algorithm we define another algorithm - the Enhanced Local Min algorithm.

The Enhanced Local Min algorithm is based upon a local scheduler and is an enhancement of the local min technique.

Definition 7.19 *A session $S_j \in \mathcal{A}^i$ is* dis2-local-min *in state Q_i if for all the* active *sessions 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 share an edge with S_m $rate(S_j)^i \leq rate(S_r)^i$.*

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

Marking a session done*:* A *dis2-local-min* session whose rate can not be increased is marked *done.*

Session selected for increase: In state i, the scheduler chooses a *dis2-local-min* session S_j whose rate can be increased and set S_j as $S(o_i)$.

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

The following theorem states basic properties of the Enhanced Local Min algorithm.

Theorem 7.20 *The Enhanced Local Min algorithm is a Bottleneck algorithm that converges after $\Theta(n^2)$ update operations.*

Proof: Since the Enhanced Local Min algorithm is a special case of the Local Min algorithm then it is a bottleneck algorithm with an upper bound of $O(n^2)$ on the number of update operations. As for the tight lower bound, the same construction as in Theorem 4.4 applies to the Enhanced Local Min algorithm. □

The Approximate Enhanced Local Min algorithm is similar to the Enhanced Local Min algorithm except for the fact that an increase is done only if it is larger than δ. Otherwise, if the increase in the *dis2-local-min* that was chosen is smaller than δ, then it is marked *done.*

Before we present the main theorem concerning the convergence of the Approximate Enhanced Local Min algorithm we need the following lemma:

Lemma 7.21 *Let S_j and S_m share an edge. For all $l > i$ if in the Approximate Enhanced Local Min algorithm, $S(o_i) = S_j$ and $S(o_l) = S_m$ then $rate(S_j)^i \leq rate(S_m)^l$.*

Proof: S_j is chosen by the scheduler in state Q_i to be updated since it is *dis2-local-min* which implies that for all the *active* sessions S_m it shares an edge with and for every edge $e \in S_m$, $FS^i(e) \geq rate(S_j)^i$ (since for every session $S_t \in \{\gamma(e) \cap \mathcal{A}^i\}$, $rate(S_t)^i \geq rate(S_j)^i$). Hence, by Lemma 3.8, for all $k > i$ the rate of S_m will not be decreased in Q_k below the value of $rate(S_j)^i$. □

Theorem 7.22 *The approximate Enhanced Local Min algorithm converges after $O(n \log_{\frac{h}{(h-1)}} \frac{\mathcal{W}}{\delta})$ update operations where h is the maximum number of sessions that share a single edge and*

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

.

Proof: The proof is obtained by following the same statements as in Subsection 7.2 where Claim 7.5 is replaced by Lemma 7.21. □

The following theorem assures that in the termination of the Enhanced Approximate Local Min algorithm, every session can not be increased by more than δ, without affecting the fairness concept.

Theorem 7.23 *Let Q_i be a state in the Enhanced Approximate Local Min algorithm where S_j is an* active *session in which the rate is minimum. If $\Delta^i(j) \leq \delta$, then for every $l > i$ $\Delta^l(j) \leq \delta$.*

Proof: The proof is obtained by following the same arguments as in Theorem 7.18. □

8 Implementation

In this paper we examined an optimistic approach for flow control calculation and analyzed the convergence of some natural algorithms that use this approach. In our model each update operation is done atomically. This assumption is done in order to simplify the analysis of the algorithms. In order to implement an "atomic update operation" we should follow the following rules:

1. At the beginning of the algorithm, all sessions are in a static mode.

2. A session is chosen to be increased only when it and all its neighbors session are in a static mode. When it is chosen, both it and its neighbor session become non static.

3. A chosen session becomes static again only after the amount of increase Δ has been calculated and the rate was modified. Its neighbors become non static only after the amount of decrease was calculated and modified.

In all the algorithms presented here the sessions have to know the rate of their neighbors. Hence, a session can easily know whether it is a local minimum. A session has the responsibility to inform all its neighbors in the following cases:

1. Its rate - in case it is changed.

2. In case it becomes *done* (needed in all algorithms except for the case of the Arbitrary Scheduler).

3. In case it can not be increased (it is needed in the Local Min algorithm).

4. The smallest rate among its *active* neighbors (which is needed in the Enhanced Local Min Algorithm in order to find a *dis2-local-min*).

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

In our analysis we assume that each session lifetime is infinite and that the main complexity issue concerns the number of update operations that are necessary to attain the final state. These assumptions are relevant to applications such as ftp (with big amounts of data), batch jobs and Mosaic.

A possible extension of this work is to do the analysis without the atomic assumption.

9 Acknowledgment

The authors would like to thank Yair Oren for some helpful discussions.

References

[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.

[Mos84] J. Mosley. *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.

A Formal Description of 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 $\sum_{i=1}^{m} R_i \leq C$. The* reform *of this set is the pair (R', Δ) where $R' = (R'_1, R'_2, \ldots, R'_m) \in R^m$ is the new vector of session rates and $\Delta \in R^+$ the amount of flow that is added to the rate of session R_j. The reform is computed to satisfy the following four conditions:*

1. $\sum_{i=1}^{m} R'_i = C$.

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

3. $\forall i \neq j,\ R_i \leq R_j + \Delta \Rightarrow R'_i = R_i.$

4. $\forall i \neq j,\ R_i > R_j + \Delta \Rightarrow R'_i = R_j + \Delta.$

Let o_i be an atomic operation where $S_j = S(o_i)$. R is a set of sessions that share an edge e, where $Cap(e) = C$. Hence, the reform $\Delta^i(e,j)$ is the maximum increase in S_j that is restricted by e, following the concept of fairness (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). Intuitively, restriction 1 implies that the edge is saturated, restriction 2 implies that an amount of Δ flow units is added to the session that is increased, restriction 3 implies that sessions with equal or lower rate are not affected and restriction 4 guarantees the fairness condition. Note 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 where $R = (R_1, R_2, \ldots, R_m) \in R^m$ is the original vector of rate flow, $\{C, \Delta\} \in R$ (C is the capacity of the edge and Δ is the increase in R_j), j be an index $1 \leq j \leq m$ and $\sum_{i=1}^m R_i \leq C$. The* construction *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 restrictions are satisfied:*

1. $\sum_{i=1}^m R'_i = C.$

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

3. $\forall i \neq j,\ R_i \leq Bar \Rightarrow R'_i = R_i.$

4. $\forall i \neq j,\ R_i > Bar \Rightarrow R'_i = Bar.$

Hence, in operation o_i, $Bar^i(e)$ is the construction 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 in $\gamma(e)$ (and in particular $R_j + \Delta \leq Bar$). Note that for all $r \neq j$ $rate(S_r)^{i+1}$ is the minimum over $\{Bar^i(e)\}$ 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 decrease 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 decrease 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 decrease 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 decrease 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 6.9. 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 decrease from 100 to 90.
- $rate(S_3)$ is increased to 80. This causes $rate(S_2)$ to decrease from 90 to 80.
- $rate(S_1)$ is increased to 100.
- $rate(S_4)$ is increased to 70. This causes $rate(S_1)$ to decrease from 100 to 90,

and $rate(S_3)$ to decrease from 80 to 70.
- $rate(S_5)$ is increased to 60. This causes $rate(S_4)$ to decrease from 70 to 60.
- $rate(S_1)$ is increased to 100.
- $rate(S_2)$ is increased to 90. This causes $rate(S_1)$ to decrease from 100 to 90.
- $rate(S_3)$ is increased to 80. This causes $rate(S_2)$ to decrease from 90 to 80.
- $rate(S_1)$ is increased to 100.