Appears in IEEE INFOCOM'98, San-Francisco, CA, USA, April 1998

Converging to Approximated Max-Min Flow Fairness in Logarithmic Time

Baruch Awerbuch[*]
baruch@cs.jhu.edu
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218

Yuval Shavitt[†]
shavitt@lucent.com
Bell Laboratories, Lucent Technologies
101 Crawfords Corner Rd., room 4G-627
Holmdel, NJ 07733-3030

Abstract

Max-min is a frequently praised criterion for flow control despite its limitations. In practice, the fast rate of changes in the connection lay-out in modern networks makes it hard for any flow control algorithm to converge to an optimal point. It such an environment, it might be better to trade accuracy with speed.

We present algorithms that relax the optimality criterion of the max-min flow fairness but achieve a fast convergence time that is logarithmic in the link bandwidth and not a function of the number of active connections (or sessions). The relaxation we use requires rates to be increased or decreased by a certain factor, $1+\varepsilon$, or in other words, assigned rates can only be a natural power of some basic bandwidth $1+\varepsilon$. Under this criterion, the quiescent time of our flow control algorithms is $\log_{1+\varepsilon} B$, where B is the maximum link bandwidth in minimum allocation units. This is a great improvement over the super-linear quiescent time of known algorithms both exact and approximated.

1 Introduction

Modern communication networks transfer traffic in a connection-oriented fashion. The connection between a source-destination pair is called a session and is assigned a route and bandwidth in the network. However, when many sessions are routed through the same links their bandwidth allocation should be controlled to prevent over utilization of the link which cause buffer overflow and data-packet loss.

Flow control algorithms [10, 12, 13] are designed to prevent congestion from occurring by assigning bandwidth to sessions on a timely basis. Two main technologies where flow control is in great need are ATM ABR traffic flow control [5, 1, 2, 7, 9] and the control of TCP streams [11, 6]. A common criterion to assign bandwidth is called max-min flow fairness [12]. The criterion allows a rate increase for a session providing this increase does not cause a decrease in the rate of a session with a lower or equal rate.

In practice, the lay-out of connections in modern networks is constantly changing due to arrival of new connections, termination of existing ones, and, some times, rerouting of existing connections. In this environment, reaching an optimal working point is a Sisyphean task never to be achieved. Instead, the best a network flow control algorithm can hope for is to converge the network fast enough to a state which is close enough to the optimal state.

In this paper, we present algorithms that relax the optimality criterion of the max-min flow fairness but achieve a fast convergence time that is logarithmic in the link bandwidth and is not a function of the number of active connections (or sessions) or the network size. The relaxation we use requires rates to be increased or decreased by a certain factor, $1+\varepsilon$, or in other words, assigned rates can only be a natural power of some basic bandwidth $1+\varepsilon$. Under this criterion, the quiescent time of our flow control algorithms is $\log_{1+\varepsilon} B$, where B is the maximum link bandwidth in minimum allocation units. This is a great improvement over the super-linear quiescent time of most known algorithms both exact and approximated [1, 9]. We show in the appendix that when the number of sessions is sufficiently large (more than ε^{-1} in each link) not more than one change in allocation per session is needed,

[*]also with MIT Lab. for Computer Science. Supported by Air Force Contract TNDGAFOSR-86-0078, ARPA/Army contract DABT63-93-C-0038, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM.

[†]Corresponding author.

and thus the quiescent time is constant in a highly loaded network.

For example, in a network with link capacity of 155Mbps and a minimum allocation unit of 64Kbps, $B = 2424$. If we select $\varepsilon = 0.05$ (5%) then the quiescent time of our flow control algorithms is not more than $c \cdot 160$, were c is some small constant (we prove that c is equal to two round trip delays), regardless of the network size and the number of connections. This is several orders of magnitude less than an algorithm with quiescent time that is linear in the number of connections, that is measured in millions and more.

A similar approach was presented by Charny et al. [7] for explicit rate schemes in ATM. However, our approach is simpler as the protocol works in a pure asynchronous fashion and the switches synchronize through the protocol messages. To the contrary, fine tuning of calculation cycles should be made to achieve good convergence time in [7]. The general approach we took make our algorithm implementable in both explicit rate schemes, e.g., ATM explicit rate, and in networks where the only signaling is congestion indication such as control of TCP connections or ATM relative rate marking. In section 6, we show how to adapt our algorithm to the ATM relative rate marking.

We present two algorithms, the first requires only constant space per link, but when session rates need to be decreased, e.g., due to the addition of new sessions, the algorithm must reset and start from scratch. Although theoretically this is not a problem, as this process requires only a logarithmic number of time steps, it might not be desired in practice due to the high-changing nature of the connection lay-out. To this end we present a second algorithm that requires logarithmic space per link but allows both increase and decrease adjustments in the rates due to changes in the lay-out of the sessions or due to arrival or tear-down of sessions.

The rest of the paper is organized as follows. In the next section, we give the model and explain our approximation. In sections 3 and 4 we present a constant space and a logarithmic space algorithms, correspondingly. In section 5 we prove the convergence time of the logarithmic-space algorithm, in section 6 we describe an ATM implementation, and in section 7 we present simulation results of our algorithms. We present our concluding remarks in the last section.

2 Model

We model the network as a directed graph $G = (V, E)$, where each vertex $v \in V$ represent a network node, and each edge $e \in E$ represents a network link. Associated with each link $e \in E$ is a capacity $cap(e) \leq B$ which is a real number.

Sessions start and terminate at arbitrary times. Each session is associated with a route which is a simple path in G between the source and the destination.

Control communication between the session sources and the network nodes is via periodic refresh messages, $MSG(\cdot)$, that are generated by each source, traverse the session route where the nodes may change the message content, and are sent back from the destination to the source along some arbitrary route. We assume no other means of communication unless otherwise specified.

A change of a session rate is called an *update*. Unlike [1, 9] updates are not required to be atomic, and can be done concurrently for several sessions.

We approximate max-min flow allocation by requiring all the components of the max-min vector to be non-negative integer powers of $1+\varepsilon$. Next we describe how a centralized algorithm for computing accurate max-min fairness can be modified to achieve the approximated allocation. A centralized algorithm to achieve an accurate max-min vector contains the following steps [4, section 6.5.2]:

C.1 Calculate the fair share on all non-saturated links.

C.2 Select the most congested link and assign a fair share to the sessions sharing this link.

C.3 Return to step 1 if not done

To achieve the approximated max-min vector we change the calculation in step C.1 to consider only non-negative integer powers of $1+\varepsilon$. Instead of selecting a single link as in line C.2, we assign a fair share to sessions passing through any of the links that achieve the minimal fare share in the last calculation.

To demonstrate our approximation consider the example of figure 1. In this example the number of sessions that use routes $r_1 - r_5$ are: 6, 1, 1, 3, and 1, respectively. The link capacities are shown in the figure (8, 8, and 5). For calculation simplicity, assume $\varepsilon = 1$. The exact max-min vector for this example is $v^* = (\frac{8}{7}, \frac{8}{7}, \frac{8}{7}, \frac{8}{7}, \frac{8}{7}, \frac{8}{7}, \frac{8}{7}, \frac{12}{7}, \frac{12}{7}, \frac{12}{7}, \frac{12}{7}, \frac{23}{7})$.

Applying the centralized modified algorithms, we get in the first iteration the fair share values $\lfloor \log_2 \frac{8}{7} \rfloor = 1$, $\lfloor \log_2 \frac{8}{5} \rfloor = 1$, and $\lfloor \log_2 \frac{5}{2} \rfloor = 2$ for links (A,B), (B,C), and (C,D), correspondingly. As a result, rate 1 is allocated to the session sharing links (A,B) and (B,C) and the session that uses route r_5 receives rate 4 in the next iteration, yielding the vector $\bar{v} = (1,1,1,1,1,1,1,1,1,1,1,4)$. Note that the vector $v_1 = (1,1,1,1,1,1,1,2,2,2,2,2)$ is a better vector for this lay-out since lexicographically it is greater. The algorithms presented here achieve, at least, the vector

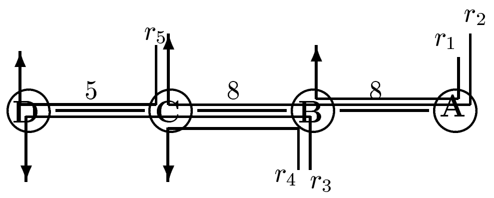

Figure 1: An approximate max-min fairness criterion example.

obtained by running the above modification to the centralized algorithm. However, they may achieve a different vector that is superior (larger) to the resulted approximated vector in a lexicographical comparison. In the example of figure 1 our algorithm is guaranteed to achieve $\bar{v}$ but may achieve v_1.

3 A constant-space algorithm

The constant-space algorithm assumes that, initially, a set of n sessions is laid out in the network, and no bandwidth is allocated. To allocate bandwidth to the sessions following the approximate max-min fairness criterion, the algorithm tries to imitate the operation of the centralized approximated algorithm by working in cycles. It starts by granting each session a unit of bandwidth. In each successive cycle, if the reminder of the bandwidth is sufficient, the bandwidth of all the sessions is updated to be $1+\varepsilon$ times the previous bandwidth. In other words, all sessions starts with *level* 0 allocation, and in each cycle their allocation level is increased by one. Sessions whose bandwidth cannot be increased since they reached their fair share allocation (up to an ε) are *blocked* and their allocation stop changing. Clearly, if global synchronization among the sessions' rate increases is kept through the cycle mechanism, an approximated max-min fair allocation can be reached in $\Theta(\log B)$ time (taking the longest end-to-end propagation delay to be one time unit) and $\Theta(\log B)$ update operations per session.

For clarity, we first present an algorithm that uses linear space, and then show how to simulate its operation with only constant space. For the linear-space algorithm, each edge[1] maintains three lists of sessions:

$\mathcal{B}$ Blocked sessions. Sessions whose rate cannot be increased due to congestion in some other link.

$\mathcal{U}$ Unsynchronized sessions. Sessions that are at a lower level, lagging in the synchronization process.

$\mathcal{S}$ Synchronized sessions. Sessions that are at current level, and wait to synchronize with unsynchronized sessions.

In addition, each edge maintains CL, a level mark indicating the higher level it is ready to grant bandwidth permissions. Initially CL is set to 0, meaning that sessions can be granted basic unit allocation. Sessions try to increase their bandwidth allocation by sending periodic messages $MSG^s(level,\ flag)$, where s is the session id (sometimes omitted in the presentation), $level$ is the current session allocation level, and $flag$ can be either, p*ending*, b*locked*, or s*talled*.

A session starts its activity by sending $MSG(-1,\ \mathtt{p})$. If all the edges along the message route agree to accept the session, the message is forwarded unchanged, and the destination sends the message $MSG(-1,\ \mathtt{p})$ back to the source. The source increases its bandwidth allocation to level 0, and start transmitting $MSG(0,\ \mathtt{p})$. An edge that receives a message $MSG^s(l,\ \mathtt{p})$ with $l < CL$ forwards the message and puts s in $\mathcal{U}$. If $l = CL$ s is moved to $\mathcal{S}$. If $\mathcal{U}$ becomes empty, and sufficient capacity exists, the edge increases its CL counter and forwards the message. If $\mathcal{U}$ is not empty, the message is forwarded with the flag s to indicate that synchronization is not achieved, yet. If $l = CL$ reached the fair share of the link, all messages are forwarded with the flag b to signal the source about this situation. The session source starts transmitting $MSG(l,\ \mathtt{b})$ to free sessions in less congested edges the session traverses from synchronizing with the blocked session. A message that arrives with a b flag, inserts its session to $\mathcal{B}$ and thus the session is not considered for synchronization. A source periodically sends $MSG(l,\ \mathtt{p})$ where l is its current allocation level until it receives a b flag. At this point it starts transmitting $MSG(l,\ \mathtt{b})$.

Note that if a session starts a little later than others, it blocks all rate increases of sessions in its cluster until it reaches their level. This is an improvement over TCP where if two sessions start sharing a link with different bandwidth, they may never equalize their bandwidth allocation [11].

For achieving constant space consumption[2] in the edges we require that only one control message per session be traveling in the network at a time, i.e., a source can send a control message only after it receives the previous one. The source is required to send more information in the message that contains now five fields and has the form $MSG(level,\ lchange,\ flag,\ fchange,\ newflag)$,

[1] The algorithm is performed by the node the link (or edge) emanates from. The 'edge' is the corresponding copy of the algorithm that is executed in the node.

[2] A constant amount of memory cells each should be $\log_2 B$ bits.

where *level* is the current session level, *lchnage* is a bit that indicates whether a level increase occurred due to the previous message reception, *flag* indicates if the session is b*locked* or not, *fchange* is a bit that indicates if the session has become blocked in the previous round, and *newflag* is used to signal to the source if the session is b*locked* or unsynchronized. *newflag* is the only part of the message that can be altered by intermediate nodes. A formal description of the constant-space algorithm appears in [3].

4 A log-space algorithm

The algorithm presented in the previous section requires a reset operation whenever there is a need to decrease the bandwidth of some session, e.g., due to the addition of a session to a bottleneck link. In this section, we present an algorithm that has the same convergence time but enables us to 'role back' when sessions are added. In other words, the algorithm can run constantly and adapt the rates of the sessions to the changing pattern of session lay-out in the network, ensuring logarithmic quiescent time. The memory consumption of the algorithm in each edge is logarithmic in the capacity of the link and independent of the number of sessions in the network.

The algorithm operates similarly to the constant-space algorithm. For the additional 'role-back' feature it uses $2\log B$ counters that keep track of the number of sessions in each level. We separately count blocked sessions and active ones, hence the factor of two. When a rate change is required, it is done as before in a synchronized fashion. A session is not allowed to increase its rate more than one level above any other active (i.e., not blocked) session, or to decrease its level less than one level below the session with the maximum allocation.

A formal description of the algorithm is given in [3]. The algorithm uses C, a bidimensional array, to count the number of b*locked* and active (termed p*ending*) sessions in each level. When a new session starts, the source sends a message of the form $MSG(level, pr\_level, flag, pr\_flag, newflag)$ where *level* and *pr_level* are the current and previous level of the session respectively; *flag* and *pr_flag* are the current and previous session state either `b` or `p`, respectively; and *newflag* is the only field that an intermediate node may alter. *newflag* can have four values d*ecrease* $<$ `b` $<$ `s` $<$ `p` it is initialized by the source to the highest value, `p`, and can be decreased by intermediate nodes. As for the constant space algorithm, a session source can send a control message only after the receipt of the previous one. For a flag f we define $\tilde{f}$ to be `b`if $f =$`b`, and `p` otherwise.

Instead of using a single rotating message per session one can use periodic submission of messages and keep the state in the intermediate edges. This change costs linear space in each of the edges but yields an algorithm that is robust to control message loss. It is also closer in concept to the ATM use of resource management (RM) cells, and to RSVP soft state refresh messages. A detailed description of the required changes is given in section 6.

5 Convergence Time

Due to space constraints, we give here the theorems and the lemmata without their proofs that can be found in [3]. We start by stating the correctness of the algorithm:

Theorem 5.1 *Sufficient time after all changes ceases all sessions are blocked, and for every link, CL is equal to the level of the session with the highest rate.*

To prove the convergence time, assume first that at time $t = 0$ no bandwidth is allocated in the network, and some sessions start sending control messages asking for bandwidth allocation. Let the round trip end-to-end delay (RTD) be one time unit for all sessions. Initially, $CL = 0$ for all network edges.

Let l_{min} be the appropriate allocation level for the most congested bottleneck link, and let N_e be the number of sessions sharing an edge e. We shall first prove

Lemma 5.1 *At time $t = 2l_{min}$, $level(s) = l_{min}$ for all sessions, and $C[l_{min}-1][pending] = N_e$ and $CL = l_{min}$ for all network edges.*

Lemma 5.2 *At time $t = 1 + 2\log_{1+\varepsilon} B$, all sessions have their appropriate rates.*

Note that if some of the sessions have a shorter round trip propagation delay, they simply wait and have 'idle' cycles in which a message is not triggering any state change. To capture this characteristic, assume that the network is at a stable state and some change occurs in a single link either a session addition or deletion. Assume that some sessions are slow, i.e., they have long RTD, used as the time unit for our measurements, and some sessions are fast, i.e., have very short RTD with respect to the fast sessions. The difference enable the fast sessions to have an infinite number of control packets traveling through a link between any two consecutive traversals of control packets of the slow connections. Under these assumptions, we prove the following lemmata.

Lemma 5.3 *The addition of a new session to a stable network requires logarithmic quiescent time.*

Appears in IEEE INFOCOM'98, San-Francisco, CA, USA, April 1998

Lemma 5.4 *If at time t_0, all sessions have their appropriate rates, and a session whose bottleneck link is e is deleted, then after 2 time units either all sessions sharing link e as their bottleneck link increase their rate or non does.*

Theorem 5.2 *The quiescent time of the algorithm is in $O(\log_{1+\varepsilon} B)$.*

We show in [3] that when the number of sessions is sufficiently large (more than ε^{-1} in each link) not more than one change in allocation per session is needed. Applying this result to the proof of theorem 5.2 implies that the quiescent time is constant in a highly loaded network.

6 Applicability for ATM signaling

In this section, we examine the applicability of our algorithms to the current ATM traffic management specification [8]. We assume that switches do not support explicit rate signaling, as for this case the implementation is trivial. Thus, we concentrate on relative rate marking and show how our algorithm can be implemented in cooperation of non participating switches, i.e., with switches that do not implement our algorithm but only relative rate marking.

For the simplicity of the discussion, we shall refer to a linear space implementation of the log-space algorithm. For this implementation, we assume that link processors maintain, $Tab[s]$, a table of linear size where each connection, s, has an entry that holds its current rate and its status flag (`b` or `p`). Each link processor should maintain the number of connections in the two states in each level (the array C), and should register the current allocation level, CL. As a result, fields pl and pf are not required in the RM cell.

Let us first assume that the MCR parameter is zero for all connections. We show that by using several fields in the RM cell [8, section 5.10] in a way coherent with the standard, our algorithm can be implemented in a heterogeneous environment where not all switches implement the suggested algorithm.

The CCR (Current Cell rate) field is set by the source to indicate the current ACR (Allowed Cell rate). The flag nf, can be implemented by using the two RM-cell bit-fields: CI and NI. CI is used to indicate congestion, NI is used to prevent rate increase. These two bit-fields map to our flag nf value as follows:

`p` — A connection is free to increase its rate. CI=0. NI=0.

`s` — A connection is waiting for synchronization. CI=?. NI=?.

`b` — A connection reached its fair share. CI=0. NI=1.

`d` — A connection is required to decrease its rate. CI=1. NI=1.

The mapping of `p`, `b`, and `d` is natural. The mapping of the `s` value requires a careful examination. The only combination not in use is CI=1 and NI=0. However, at this state, the switch is not congested, thus, we do not wish to set the CI flag. If CI=0, any selection for the NI mapping will result in two values mapped to the same flag combination. To differentiate between the two values, an additional bit from the reserved bits in the RM cell should be used.

Any mapping should consider the option that some of the switches along the route may not be programmed to use our algorithm and are only capable of setting the NI and CI bits (according to *relative rate marking* [8, sec. 5.10.6]). If such a switch on the way is setting the NI bit, the setting must be understood as "the session is *blocked*", or otherwise the algorithm may livelock by waiting to a synchronization that cannot occur. This can be done by mapping the `s` value to CI=0 and NI=0 with the additional reserved bit set. This way, setting NI either by a participating or by a non-participating switch results in the blocking of this session without livelocking others. A source that does not implement our algorithm, may increase its rate by one level without synchronizing with the rest of the sessions but in the next cycle it will be forced to decrease its rate back since its rate will be greater than CL and thus participating switches will set both the CI and NI bits. An additional reserved bit is required to implement the f field.

Switches that implement EFCI marking or explicit rate marking can coexist with switches that use our relative rate marking variant. It is important that the rate decrease factor (RDF) and the rate increase factor (RIF) are set to corresponding values and that these two parameters are equal for all sessions. If MCR is greater than zero, our scheme can be easily used to achieve the "MCR plus equal share" as suggested in [8, sec. I.3].

7 Simulation Results

We simulated both versions of the logarithmic algorithm. We first report the earlier version that requires a single circulating control message per connection.

Figures 2 and 3 show the session allocation level and link utilization for a single link network with RTD of 2μsec. The minimum allocation level is 64Kbps, the link capacity is 25Mbps, and $\varepsilon = 0.05$.

At time t=0, five sessions become active and raise their allocation level in a synchronous fashion, as can

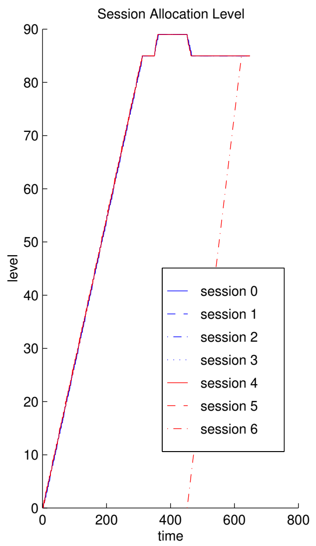

Figure 2: A one link network - session allocation level.

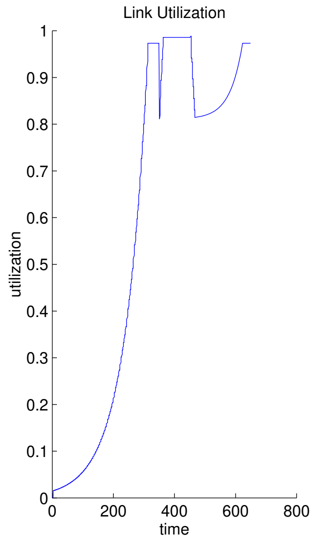

Figure 3: A one link network - link utilization.

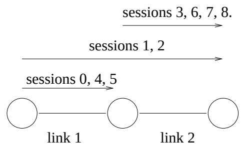

Figure 4: A two link network - session layout.

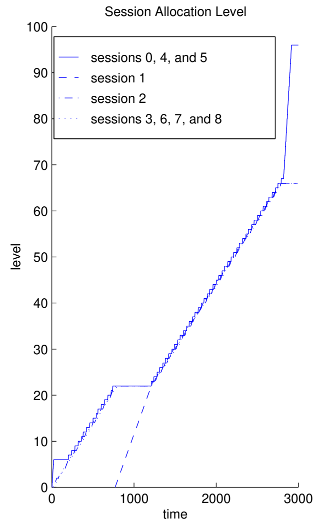

Figure 5: A two link network - session allocation level.

be seen in figure 2. Note that the Y-axis unit is in allocation levels, i.e., the graph shows the logarithm of the bandwidth used by the session. The five sessions reach their fair share in a time equal to about two RTD times their fair level as stated by lemma 5.2. At time t=350, session 5 terminated, and as a result the other four sessions increase their allocation level in full synchrony. At time t=450, a new session becomes active. The four active sessions reduce their rate to the proper level in synchrony, while the new session increases its rate until it reaches its proper level.

Figure 3 shows that link utilization is always above 97% in steady state. When session 5 terminates the freed bandwidth is shared by the rest of the sessions very quickly, since these session are already at high allocation level and every increase by a factor of ε is substantial. This is not the case when a new session joins, as it has to increase its rate from level 0. This time can be shorten by the addition of a scout control message that is sent by a new session when it becomes active and traverses the route to check for the bottleneck link allocation level. After waiting RTD time, the session can start transmitting at a level slightly lower than the bottleneck link level.

Figure 5 shows the session allocation level for a two link network, where link 1 has a delay of 1μsec, and

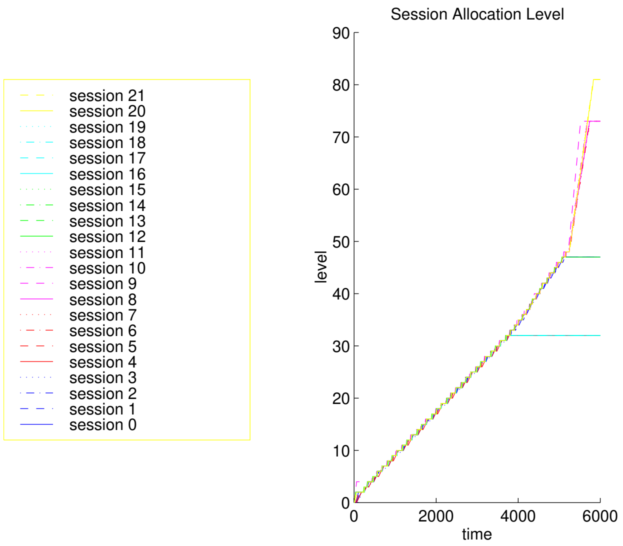

Figure 6: A multi link network - session allocation level.

link 2 a delay of 9μsec. The session layout is depicted in figure 4. All the sessions become active at time 0, except session 1 that becomes active at time 750.

The following can be observed from figure 5.

- In the beginning, before session 2 is registered in link 1 (due to its long RTD), sessions 0, 4, and 5 raise their allocation level at a fast rate and are stopped after one RTD of session 2 (20μsec) waiting to synchronize with it.
- When session 1 becomes active, all the other sessions stop raising their level and wait to synchronize with it. Session 1 raises its rate one level every RTD until it reaches the level of the other sessions. At this point all sessions continue to raise their level every 2RTD (40μsec).
- After the sessions with high RTD value reach their fair level, the remaining sessions (0, 4, and 5) continue to increase their rate every 2RTD of the active sessions, i.e., every 4μsec.

Figure 6 shows the simulation results for the network simulated in [7]. The link delay is set to 6μsec to match the switching and the propagation delay in [7].

Note that once the allocation level of the sessions with long RTD is set, the rest of the sessions can increase their allocation level at a higher rate since the synchronization is required only among *pending* sessions. As expected utilization increases exponentially.

The behavior of the second version of the algorithm is similar, only the time axis changes. When the sessions start transmitting and their rates are low, the level increase is done every 2 TRM, an ATM parameter that specifies the minimum time between two RM cells for a connection. When the rates of the sessions increase, they reach a point where RM cells are transmitted at a rate higher than TRM, and then the level increase rate increases as the sessions increase their data transmission rate.

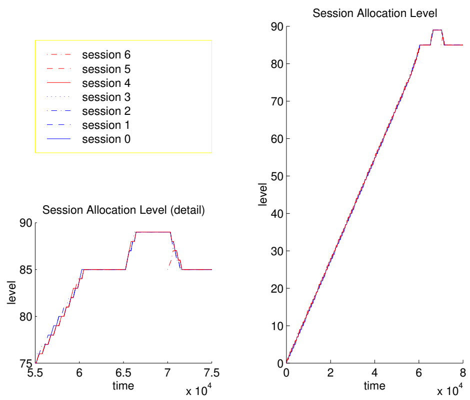

Figure 7: A one link network - session allocation level.

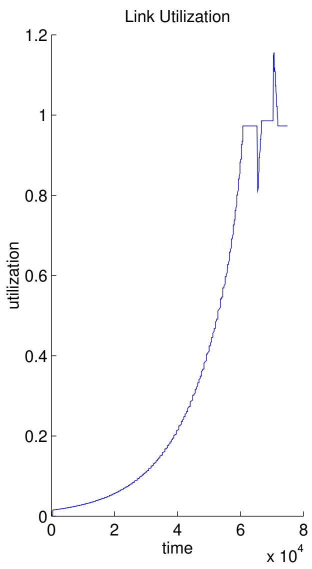

Figure 8: A one link network - link utilization.

Figures 7 and 8 show the simulation results of the second version of the algorithm for the one link network configuration which was simulated to obtain figures 2 and 3. TRM is set to 400μsec. Up until level 72 the link level is increased every 800μsec, at this point the rate of RM cells is higher and the rate of level increase is increased.

In this simulation we used a scout message, i.e., before a message starts transmission it sends a control message to verify the allocation level on the most congested link and start at a level somewhat lower (here 5 levels below). The result is a very fast convergence to the fair allocation level, but at a cost of transient overallocation. Note that the overallocation is about $1/n_b$ where n_b is the number of sessions in the bottleneck link. In the simulated scenario $n_b = 5$ and the overallocation peak is 1.156. However in practice, the number of sessions sharing the most congested link should be higher, and the overallocation - lower. To reduce the bad effect of overallocation, a bound on the starting allocation level can be set.

8 Concluding Remarks

The logarithmic-space algorithm can be generalized to allow session to be assigned any finite set of rates as required by the nature of the applications. To this end each node and end-user (session) should hold a finite table of sorted rates to be used instead of the natural powers of $1+\varepsilon$. A more practical generalization is to change the value of ε for different ranges of rates. For example, for lower and higher allocation levels to use a rather high ε and for the middle levels that are the most frequently used to set a low ε. This allocation scheme does not consume memory, minimizes the wasted bandwidth due to quantization, and speeds-up the rate at which sessions reach their appropriate allocation level at start-up (assuming no scout message).

In the worst case, up to ε of the link bandwidth may be wasted due to quantization. For the case of non greedy sessions, wasted bandwidth can be reduced by measuring the bandwidth that is used at every link, instead of summing the allocated bandwidth. This way permanent non-used allocated bandwidth can be reused.

Note that if RM cells are assumed not to be lost or corrupted a constant space algorithm can be easily devised based on this work.

Acknowledgement

We would like to thank Lei Zhang for her help in obtaining the simulation results.

References

[1] Y. Afek, Y. Mansour, and Z. Ostfeld. Convergence complexity of otimistic rate based flow control algorithms. In *28th Annual ACM Symposium on Theory of Computing*, pages 89 – 98, May 1996.

[2] Y. Afek, Y. Mansour, and Z. Ostfeld. Phantom: A simple and effective flow control scheme. In *ACM SIGCOMM*, pages 169 – 182, Aug. 1996.

[3] B. Awerbuch and Y. Shavitt. Converging to approximated max-min flow fairness in logarithmic time. Technical Report TR 97-01, Johns Hopkins University, Jan. 1997.

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

[5] F. Bonomi and K. W. Fendick. The rate-based flow control framework for the available bit rate ATM service. *IEEE Network*, 9(2):25 – 39, March/April 1995.

[6] L. S. Brakmo and L. Peterson. TCP Vegas: end to end congestion aviodance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465 – 1480, Oct. 1995.

[7] A. Charny, K. K. Ramakrishnan, and A. G. Lauck. Scalability issues for distributed explicit rate allocation in ATM networks. In *INFOCOM96*, pages 1198 – 1205, Mar. 1996.

[8] T. A. F. T. Committee. Traffic management specification version 4.0, Apr. 1996. af-tm-0056.000.

[9] P. Fatourou, M. Mavronicolas, and P. Spirakis. Efficiency of oblivious versus non-oblivious schedulers for optimistic rate-based flow control. In *PODC*, Aug. 1997.

[10] M. Gerla and L. Kleinrock. Flow control: A comparative survey. *IEEE Transactions on Communications*, COM-28(4):553 – 574, Apr. 1980.

[11] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pages 314 – 329, Aug. 1988.

[12] J. M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, COM-29(7):954 – 962, July 1981.

[13] S. Keshav. A control-theoretic approach to flow control. In *ACM SIGCOMM*, pages 3 – 16, Sept. 1991.