Distributed Algorithms for Computation of Fair Rates in Multirate Multicast Trees

Saswati Sarkar and *Leandros Tassiulas*
Dept. of Electrical and Computer Engineering and Institute for Systems Research
University of Maryland, College Park, MD, USA
email addresses: swati@isr.umd.edu, leandros@isr.umd.edu

Abstract— **We study fairness in arbitrary networks with multicast capabilities. Multicast traffic in internet and ATM provides a motivation for studying these networks. A study of fairness in multicast networks poses several interesting problems e.g., the issue of *intra-session* fairness in addition to that of *inter-session* fairness in unicast networks. We develop a mathematical framework to model the fair allocation of bandwidth in multirate multicast networks with minimum and maximum rate constraints. We present distributed algorithms for computation of maxmin fair rates allocated to various source-destination pairs.**

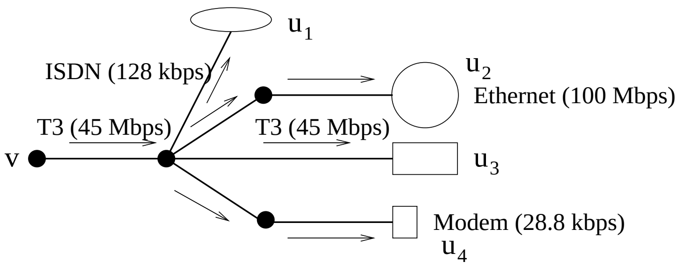

Fig. 1.

I. Introduction

Multicasting poses some specific fairness challenges. This is because of network heterogenity. A single session may have many destinations, and end systems can have widely varying bandwidth connectivities. On one hand there are fast ethernets (100 Mbps) and on the other hand there are slow modems (28.8 kbps). The paths to different destinations may have different bandwidth capacities, e.g., one may consist of multi-megabit links, such as, $T3$ (45 Mbps) and another may have a 128 kbps ISDN line (Refer to the network shown in figure 1 for an example). Every receiver would like to receive service at a rate commensurate with its capabilities and the capacity of the path leading to it from the source independent of the capabilities of the other receivers of the same session. This is the issue of *intra-session* fairness. Besides, like in unicast, there is the issue of *inter-session* fairness, that is fairness of members across multiple sessions. So multicasting poses the issue of *intra-session* fairness in addition to that of *inter-session* fairness.

A single rate of transmission per session is likely to either overwhelm the slow receivers or starve the fast ones, in absence of additional provisions. Multirate transmission is necessary to counter network heterogenity. The receivers of the same session are allowed to receive at different rates. The source transmits at a rate matching the fastest of its receivers. At every link, the transmission rate of a session is equal to that of the fastest session receiver downstream of the link. We discuss various multirate schemes later. Multi rate schemes can be used to attain inter-session and intra-session fairness, in a bandwidth efficient manner, by having the receivers subscribe to a "fair" rate.

We have proposed a routing and scheduling policy in [14] which stabilizes the system, if the network can accomodate all the traffic demands, without congestion. However, resource limitations may not permit this always. Fairness of resource allocation becomes important in such a scenario. It has been formally shown that fairness properties of a multicast network improves if multirate transmission is used instead of single rate transmission[13]. However well known network protocols for multirate transmission, RLM (Receiver-driven Layered Multicast)[12] and LVMR (Layered Video Multicast with Retransmissions)[9] achieve improved intra-session fairness, but do not handle inter-session fairness very well[10]. A scheme for fair resource allocation for multi-session layered video multicast which strives to rectify this defect in RLM and LVMR has been proposed in [10]. The authors present empirical evidence that the scheme improves inter-session fairness for networks with multiple video sessions sharing only one link. But, there is no experimental or analytical evidence that the scheme works well for more complex networks, with sessions sharing several links with each other. Besides, [10] does not compute fair rates allocated to the receivers in an arbitrary network, under some well defined notion of fairness. An algorithm for computation of maxmin fair rates in a multirate multicast network has been proposed in [13]. However, this algorithm requires global knowledge of system states for computation of the maxmin fair rates. Current networks have large sizes. Thus global knowledge of system states may not be present at any single point in the network. Distributed algorithms for computation of fair rates are more useful in this context.

We start with a mathematical formulation of the problem. We adopt the notion of maxmin fairness[4]. We would define this notion more precisely later, but informally speaking a rate allocation is maxmin fair, if no receiver can be allocated a higher rate without hurting another receiver having equal or lower rate. Maxmin fairness is a good notion of fairness and as [13] points out, maxmin fairness satisfies many intuitive fairness properties in a multirate multicast network. We present a distributed algorithm for computation of maxmin fair rates in an arbitrary network with any number of multicast sessions. However, this distributed algorithm makes certain assumptions which may not hold always. We present another distributed algorithm for computation of maxmin fair rates which is slower than the first one in the worst case, but operates under very general assumptions. We also present complexity analysis for these algorithms. We address generic mechanism for allocation of the rates in multirate multicast networks, once the rates are computed. Finally, we conclude with some future directions of research. We do not assume any properties specific to internet or ATM. Thus our algorithms are applicable in very general scenario. Keeping in mind ATM networks, we have incorporated minimum rate requirements and maximum rate constraints in our model. Incidentally, no existing model for multirate multicast networks incorporates minimum rate requirements.

We have omitted all proofs on account of space restrictions. Refer to technical report[15] for proofs and details.

II. Network Model

We consider an arbitrary topology network with N multicast sessions. A multicast session is identified by the triplet (n, v, U), n is a unique number assigned to the session, v is the source node of the session and U is the group of intended destination nodes (n has been incorporated to distinguish between different sessions with same source destination pair). We assume that the traffic from node v is transported across a predefined multicast tree to nodes in U. The tree can be established during connection establishment phase if the network is connection oriented or can be established by some well known multicast routing protocol like DVMRP[7], CBT[3], etc. in internet type network. The receivers may have minimum rate constraints. Also some sources may not be able to transmit at a rate higher than a certain threshold. Some receivers may have a low processing ability. In that case, it is useless to allocate higher rates to receivers of that session. So rate allocations can have peak rate constraints as well. These parameters are useful for ATM like scenarios, where session establishment is preceded by a negotiation stage and the network can be informed of these requirements during the negotiation stage. Connectionless networks like internet generally do not have any such requirements. For these, we assume minimum and maximum rates as 0 and ∞ respectively for each receiver.

We call every source destination pair of a session a virtual session. For example, if a session n has source v and destination set U, where $U = \{u_1, \ldots, u_t\}$, then this session would correspond to t virtual sessions, $(n, v, u_1), \ldots, (n, v, u_t)$. For example, the network shown in figure 1 has a single session, $(1, v, U)$, with $U =$

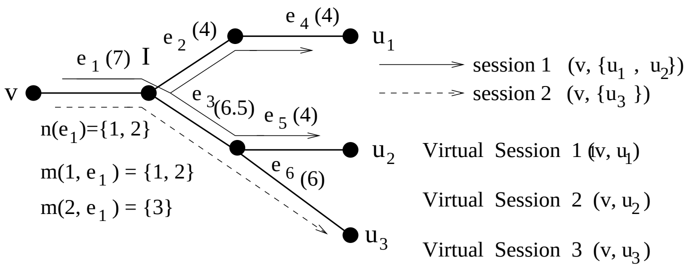

Fig. 2. Session 1 includes virtual sessions, 1, 2. Session 2 includes virtual session 3. The figure shows some sample $n(e_i)$ and $m(i, e_j)$s. The numbers in brackets, () denote the capacities of the respective links. The capacity constraint for link e_1 is $\max(r_1, r_2) + r_3 \leq 7$ and that for link e_3 is $r_2 + r_3 \leq 6.5$. The minimum and maximum rate constraints are $4 \leq r_1 \leq 5$, $1 \leq r_2 \leq \infty$ and $0 \leq r_3 \leq 5$.

$\{u_1, \ldots, u_4\}$. This session corresponds to 4 virtual sessions, $(1, v, u_1)$, $(1, v, u_2)$, $(1, v, u_3)$, $(1, v, u_4)$. Our objective would be to achieve a maxmin fair rate allocation for the virtual sessions. We look at rates attained by virtual sessions instead of those attained by the actual sessions because our requirement is that every receiver of every session should get a bandwidth commensurate with its fair share of the capacity of the path between the source and the receiver. So ensuring maxmin fairness of the session rate allocation and allocating the maxmin fair session rate to all the virtual sessions would cause intra-session unfairness. We assume that every virtual session (source-destination pair) has a minimum and a maximum rate.

A rate allocation for virtual sessions is feasible, if the rate for every virtual session is between the minimum and the maximum possible rates for the virtual session. Besides if session n corresponds to virtual sessions $m_{n1}, \ldots, m_{nt}$ in link l, then the maximum of the rates allocated to the virtual sessions $m_{n1}, \ldots, m_{nt}$ is the bandwidth consumed by session n in link l. Total bandwidth consumed by all sessions traversing through link l can not exceed the capacity of link l. More formally let there be M virtual sessions. A M-dimensional vector $(r_1, \ldots, r_M)$ is a feasible rate allocation if

1. $\mu_i \leq r_i \leq p_i \ \forall i$, where μ_i and p_i are respectively the minimum and maximum rates of virtual session i, $p_i \geq \mu_i \geq 0$,
2. $\sum_{i \in n(l)} \max_{j \in m(i,l)} r_j \leq C_l$, where $n(l)$ denotes the set of sessions passing through link l, $m(k, l)$ denotes the set of virtual sessions of session k passing through link l and C_l denotes the capacity of link l. Let λ_{il} denote the rate allocated to the session i on link l under rate allocation $\vec{r}$. It is actually the maximum of the rates allocated to the virtual sessions in $m(i, l)$, i.e., $\lambda_{il} = \max_{j \in m(i,l)} r_j$. The capacity condition can also be stated as
$\sum_{i \in n(l)} \lambda_{il} \leq C_l$ (capacity condition).

Figure 2 illustrates en example network with a few capacity and maximum and minimum rate constraints.

A feasible rate vector is maxmin fair if it is not possible to maintain feasibility and increase the rate of a virtual session without decreasing that of any other virtual session which has equal or lower rate. More formally, a feasible rate allocation vector $\vec{r}^1$ is maxmin fair if it satisfies the following property with respect to any other feasible rate allocation vector $\vec{r}^2$: if there exists i such that the ith component of $\vec{r}^2$ is strictly greater than that of $\vec{r}^1$ ($r_i^2 > r_i^1$), then there exists j such that the jth component of $\vec{r}^1$, r_j^1 is less than or equal to the ith component of $\vec{r}^1$, r_i^1 ($r_j^1 \leq r_i^1$) and the jth component of $\vec{r}^2$ (r_j^2) is strictly less than the jth component of $\vec{r}^1$ ($r_j^2 < r_j^1$). The bandwidth allocations according to $\vec{r}^2$ are less even than those according to $\vec{r}^1$ in some sense.

Example II.1: The maxmin fair rate vector in the network of Figure 2 is $(4, 3.5, 3)$. It is easy to check that this rate vector is feasible. It is not possible to decrease r_1 below 4 because of the minimum rate constraint. This and the capacity constraint of link e_1 forces r_3 to be at most 3. Any increase in r_1 or r_2 will cause a decrease in r_3 which is less than both.

Henceforth we shall ignore the maximum rate constraints. This does not cause any loss in generality because maximum rate constraints can be incorporated by adding artificial links between receivers with maximum rate constraints and the rest of the network, with capacities of the artificial links equal to the maximum rates of the respective receivers. The size of the augmented network is comparable to that of the given network. So complexity of any algorithm for computation of the maxmin fair rates in a network should remain the same, if we use the augmented network instead.

Next we present an important property of the maxmin fair rate vector. We first introduce the concept of bottleneck links. A link l is said to be *bottlenecked* with respect to a virtual session k traversing it, if the following conditions are met:

- Capacity of the link is fully utilized, i.e., the sum of the rates allocated to the sessions traversing the link must be equal to the capacity of the link: $\sum_{i \in n(l)} \lambda_{il} = C_l$.
- The virtual session has the maximum rate amongst all virtual sessions of the same session travelling through the link, i.e., $r_k = \lambda_{\chi(k)l}$, where $\chi(k)$ is the session corresponding to virtual session k.
- If any other virtual session j traversing through link l, has a rate higher than that of virtual session k, then rate of virtual session j is less than or equal to the minimum rate required by some virtual session in $m(\chi(j), l)$ (set of virtual sessions traversing link l and belonging to the same session as virtual session j). In other words, if $r_j > r_k$ then $r_j \leq \mu_{\chi(j)l}$ where $\mu_{il} = \max_{p \in m(i,l)} \mu_p, i.e., \mu_{il}$ is the maximum of the minimum rate requirements of the virtual sessions in $m(i, l)$.

Example II.2: Refer to the network of Figure 2. Consider the rate vector $(4, 3.5, 3)$. Link e_1 is bottlenecked w.r.t. virtual sessions (v, u_1) and (v, u_3), e_3 is bottlenecked w.r.t. virtual session (v, u_2). Link e_6 is not bottlenecked w.r.t. any virtual session because its capacity is not fully utilized. Consider the rate vector $(4, 4.5, 2)$. This is also a feasible rate vector. Now no link is bottlenecked w.r.t. virtual session (v, u_3).

The definition of a bottleneck link is similar to that in the unicast context. In the unicast context, a link is bottlenecked w.r.t. a session i if its capacity is fully utilized and if any other session j traversing the link has greater bandwidth, then the rate of j is equal to its minimum required rate. In multicast, the difference is that the rate of a different virtual session j can be more than that of s on the bottleneck link of s, not only because of its minimum rate requirement, but because of the minimum rate requirement of some other virtual session of the same session as j, traversing this bottleneck link. In absence of minimum rate requirements, the bottleneck condition becomes simpler. A link is bottlenecked w.r.t. a virtual session if its capacity is fully utilized and no other virtual session traversing the link has a greater rate.

Lemma 1 (Bottleneck Lemma) A feasible rate vector is maxmin fair iff every virtual session has a bottleneck link.

Remark: Bottleneck Lemma serves as a test for maxmin fairness of a feasible rate allocation vector. It indicates that if a rate vector is maxmin fair, then the rate of receiver s is at least $C_l/|n(l)|$, for some link l on its path, if there are no minimum rate requirements. In presence of minimum rate requirements, this lower bound becomes $\frac{C_l - \sum_{i \in \tau(l)} \mu_{il}}{|n(l) \setminus \tau(l)|}$, where $\tau(l)$ is the set of sessions traversing link l, whose session rates on link l are greater than the rate of virtual session s.

III. A SYNCHRONOUS DISTRIBUTED ALGORITHM FOR COMPUTATION OF MAXMIN FAIR RATES

We proceed to present a synchronous distributed algorithm for computing the maxmin fair virtual session rates shortly. It is synchronous in the sense that all sessions must start at the same time. We first give a generic description of the algorithm and then discuss its distributed implementation.

A. Description of Algorithm

$n(l)$, $m(i, l)$, $\chi(s)$ and μ_{il} are as defined in pages 2, 2, 3 and 3 respectively. We introduce some additional terminologies.

L_s is the set of links traversed by virtual session s.

$r_s(k)$ is the bandwidth allocated to virtual session s at the end of the kth iteration. $\vec{r}(k)$ denotes the rate vector at the end of the kth iteration, with components $r_s(k)$.

$\lambda_{il}(k)$ is the rate allocated to the session i on link l at the end of the kth iteration. It is actually the maximum of the rates allocated to the virtual sessions in $m(i, l)$ at the end of the kth iteration.

A virtual session s is *saturated* under rate vector $\vec{r}(k)$ if there exists a link l on its path such that the capacity of the link is fully utilized and s has the maximum rate amongst all virtual sessions of $m(\chi(s), l)$, i.e.,

$$\sum_{i \in n(l)} \lambda_{il}(k) = C_l \text{ and } r_s(k) = \lambda_{\chi(s)l}(k).$$

A session is *saturated* on a link l if all the virtual sessions of the session travelling through the link l are saturated.

$S(k)$ denotes the set of unsaturated virtual sessions at the end of the kth iteration.

$\Xi_l(k)$ denotes the set of unsaturated sessions passing through link l at the end of the kth iteration.

$F_l(k)$ denotes the total bandwidth consumed by the saturated sessions passing through link l at the end of the kth iteration.

$\eta_l(k)$ denotes the link control parameter of link l at the end of the kth iteration. Link control parameter is the iterate which would be used in computation of the maxmin fair rates. It is an estimate of the fair share of the bandwidth of the link which can be allocated to the unsaturated virtual sessions traversing the link.

$\eta_{il}(k)$ denotes the session link parameter of session i traversing through link l. It is the bandwidth assigned to session i if there were no bandwidth constraints for any of its virtual sessions on other links.

The following algorithm computes the maxmin fair rates for the virtual sessions.

1. $k = 0$, $\eta_l(0) = 0$, $F_l(0) = 0$, $S(0) = \{1, \ldots, M\}$, $\Xi_l(0) = n(l)$ $\forall$ link l, $r_s(0) = \mu_s$ $\forall$ virtual session s.
2. $k \rightarrow k + 1$.
3. For every link l in the network, compute the link control parameter. If $\Xi_l(k-1) \neq \phi$, then $\eta_l(k)$ is the maximum possible θ, which satisfies the equation, $F_l(k-1) + \sum_{i \in \Xi_l(k-1)} \max(\theta, \mu_{il}) = C_l$ else $\eta_l(k) = \eta_l(k-1)$. $\eta_{il}(k) = \max(\eta_l(k), \mu_{il})$.
4. Compute $r_s(k)$ for all virtual sessions s, where $r_s(k) = \min_{l \in L_s} \eta_{\chi(s)l}(k)$, if $s \in S(k-1)$, else $r_s(k) = r_s(k-1)$
5. For every link l in the network compute the session rate in link l, for every session in $n(l)$, $\lambda_{il}(k) = \max_{s \in m(i,l)} r_s(k)$.
6. Compute the set of virtual sessions unsaturated after the kth iteration, $S(k) = S(k-1) \setminus \{s : \exists\, l \in L_s$, s.t. $\sum_{i \in n(l)} \lambda_{il}(k) = C_l$ and $r_s(k) = \lambda_{\chi(s)l}(k)\}$.
7. If $S(k) = \phi$, i.e., all virtual sessions are saturated, the algorithm terminates, else go to next step.
8. For every link l, compute the set of unsaturated sessions passing through the link l at the end of the kth iteration: $\Xi_l(k) = \{n : n \in \{1, \ldots, N\}$, $m(n,l) \cap S(k) \neq \phi\}$ (N is the number of sessions).
9. For every link l, for which $\Xi_l(k) \neq \phi$, compute the bandwidth consumed by the saturated sessions passing through link l, $F_l(k) = \sum_{i \in n(l) \setminus \Xi_l(k)} \lambda_{il}(k)$.
10. Go to step (2).

At every iteration k, the algorithm computes a "fair share" of the link bandwidth for every session i, the session link control parameter, $\eta_{il}(k)$. This bandwidth is offered to all virtual sessions of session i traversing the link. If a virtual session is constrained to have a rate less than its fair share because it is assigned a lower bandwidth on another link, then it can not use some of this bandwidth. If all virtual sessions of the same session release some bandwidth because of constraints on other links, then there is residual bandwidth. The residual bandwidth is split fairly among other sessions in the next iteration and the process continues. For deeper insight, ignore the minimum rate constraints for the time being. Initially all the link control parameters are assigned zero values. All sessions and virtual sessions are unsaturated. Next the algorithm computes the link control parameters as per step (3). The link control parameter for link l at 1st iteration, is the capacity of the link per session traversing the link. If a virtual session traversing link l, had no bandwidth constraint on other links, then it is assigned a rate equal to the link control parameter. On account of the bandwidth constraint in other links, the virtual session gets a rate equal to the minimum of the link control parameters on its path. All virtual sessions traversing the link with the minimum link control parameter are saturated[15]. A session is saturated if all its virtual sessions are saturated. The bandwidth consumed by the saturated sessions, if any, are computed. This bandwidth is subtracted from the link capacity, and the link control parameters are recomputed at the beginning of every iteration. The link control parameter in the next iteration is the residual capacity per unsaturated session traversing the link, if any, else it is the same as the link control parameter of the previous iteration. The virtual session bandwidths are assigned in the same manner and the process continues. Since at least one virtual session is saturated every iteration, the algorithm terminates in at most M iterations (M is the number of virtual sessions)[15].

In presence of nonzero minimum rates, link control parameter of link l is computed as per step (3). In absence of bandwidth constraints on other links, a virtual session s traversing through link l is assigned a rate equal to its session link control parameter on link l, $\eta_{\chi(s)l}(k)$. Session link control parameter is the maximum of the link control parameter and the minimum allowable rates (μ_ss) of the virtual sessions of the session traversing through the link. On account of the bandwidth constraint in other links, the virtual session gets a rate equal to the minimum of the above quantity, the minimum taken over all links in the path of the virtual session. Again, at least one virtual session is saturated every iteration[15]. The process continues till all the virtual sessions are saturated.

We illustrate the operation of the algorithm with an example.

Example III.1: Consider the network of Figure 2. Ignore the maximum rate constraints. $L_1 = \{e_1, e_2, e_4\}$, $L_2 = \{e_1, e_3, e_5\}$, $L_3 = \{e_1, e_3, e_6\}$. Link control parameters are as follows. $\eta_{e_1}(1) = 3$, $\eta_{e_2}(1) = 4$, $\eta_{e_3}(1) = 3.25$, $\eta_{e_4}(1) = 4$, $\eta_{e_5}(1) = 4$, $\eta_{e_6}(1) = 6$. Now, the session link control parameters are as follows. $\eta_{1e_1}(1) = 4$, $\eta_{2e_1}(1) = 3$, $\eta_{1e_2}(1) = 4$, $\eta_{1e_3}(1) = 3.25$, $\eta_{2e_3}(1) = 3.25$, $\eta_{1e_4}(1) = 4$, $\eta_{1e_5}(1) = 4$, $\eta_{2e_6}(1) = 6$. Computing the $r_s(1)$s as per step 4, we have $r_1(1) = 4$, $r_2(1) = 3.25$, $r_3(1) = 3$. Observe that virtual sessions 1 and 3 are saturated, while virtual session 2 is not. $S(1) = \{2\}$. Thus session 2 is saturated on all links. Session 1 is unsaturated on only those links which are on the path of virtual session 2. $\Xi_{e_1}(1) = \Xi_{e_3}(1) = \Xi_{e_5}(1) = \{1\}$, $\Xi_l(1) = \phi$, if $l \notin \{e_1, e_3, e_5\}$. $F_{e_1}(1) = F_{e_3}(1) = 3$, and $F_{e_5}(1) = 0$. Computations for the next iteration are as follows. $\eta_{e_1}(2) = 4$, $\eta_{e_3}(2) = 3.5$, $\eta_{e_5}(2) = 4$. For the rest of the links, $\eta_l(2) = \eta_l(1)$. $\eta_{1e_1}(2) = 4$, $\eta_{1e_3}(2) = 3.5$, $\eta_{1e_5}(2) = 4$. Thus $r_2(2) = 3.5$. $r_s(2) = r_s(1)$, $s \in \{1, 3\}$. Now virtual session 2 is also saturated. So $S(2) = \phi$ and the algorithm terminates with rates $(4, 3.5, 3)$.

The rates allocated to the virtual sessions, upon termination of the algorithm are the maxmin fair rates. This follows from the following theorems.

Theorem 1: If the algorithm terminates in k iterations, then $\vec{r}(k)$ is the max-min fair rate vector.

The formal proof is long and omitted here for brevity. The intuition behind the result is as follows: max-min fair sharing implies that if there are k sessions sharing a link, each session should get a "fair share" of the link bandwidth. If a session is constrained to have a rate less than its fair share because it is assigned a lower bandwidth on another link, then the residual bandwidth is split fairly among other sessions. This is exactly what the algorithm does. The next theorem ensures that the algorithm terminates in finite number of iterations.

Theorem 2: The algorithm terminates in at most M iterations, where M is the number of virtual sessions.

The proof considers the link which attains the minimum link control parameter in the kth iteration amongst all those links which carry at least one unsaturated session. We show that the capacity of the link is fully utilized, and at least one unsaturated virtual session traversing the link has rate equal to its session rate on the link. This virtual session saturates in the kth iteration. Thus all virtual sessions saturate by M iterations.

Some well known algorithms for computation of maxmin fair rates in the unicast scenario, e.g., [2], [4], are special cases of this algorithm. The introduction of multicast complicates the problem significantly. For example, the basic constraints for feasibility are different. We need to think in terms of virtual sessions instead of sessions. The condition for a link to be bottlenecked w.r.t. a virtual session is different from that of a link to be bottlenecked w.r.t. a session in the unicast scenario. Also a virtual session can have a rate higher than the link control parameter of a link it traverses, not only if its minimum bandwidth requirement is higher, but also if the minimum bandwidth requirement of some other virtual session of the same session traversing through the same link is higher. The proof needs to take care of these intricate details. For example, the algorithm for computation of maxmin fair rates would terminate in at most $|\mathcal{L}|$ iterations, for unicast networks, where $\mathcal{L}$ is the set of links. Technical Report [15] presents a counter-example where the algorithm takes more than $|\mathcal{L}|$ iterations to terminate. However, the algorithm terminates in at most $|\mathcal{L}|$ iterations for multicast networks, if a specific condition is met, i.e., if all virtual sessions of the same session sharing a link at some point, have the same minimum rate requirement ($\mu_i = \mu_j$ if $\chi(i) = \chi(j)$ and $L_i \cap L_j \neq \phi$) (Lemma 2). Note that this condition on minimum rates always holds in unicast networks because every session has only one virtual session.

Lemma 2: The algorithm terminates in at most $\min(|\mathcal{L}|, M)$ iterations, if $\mu_i = \mu_j$ for all i, j such that $\chi(i) = \chi(j)$ and $L_i \cap L_j \neq \phi$.

Every step of this algorithm has a complexity of $O(|\mathcal{L}|M)$. The algorithm must terminate in M iterations (Theorem 2). Thus the overall complexity of this algorithm is $O(|\mathcal{L}|M^2)$.

B. *Distributed Implementation of the Synchronous Algorithm*

We describe a framework for synchronous distributed implementation of the algorithm presented in the previous subsection. We will exploit the fact that the computation of the session link control parameters of any link needs only information about the saturation status of sessions traversing the link and the previous iteration rates of the saturated sessions traversing the link. A virtual session (receiver) can determine its rate if it knows its session link control parameters of the links on its path. To know whether it is saturated or unsaturated, it needs to know only the session link rates on its path.

Every node maintains an information field for every outgoing link. The information field has an entry for the link control parameter and a separate record for every session traversing the link. The record for session i traversing link l maintains the session link rate, the minimum required rate for the session, session saturation map and a rate bitmap for the session. The session saturation map has one entry for every virtual session of $m(i, l)$. The entry for virtual session $s \in m(i, l)$ indicates whether it is saturated or unsaturated or its status is not known. The rate bitmap has one bit for every virtual session of $m(i, l)$. The entry for virtual session $s \in m(i, l)$ indicates whether its rate is known or not known.

At first every entry in the saturation maps indicates unsaturated, and every entry in the rate bitmaps indicates that rate is known. All session link rates are initialized to the minimum rates. Source rates are initialized to the minimum allowable rates (maximum of the minimum rates required by all receivers of the session). Consider link l. Link control parameter is computed as per step (3) at the origin of link l. Next the session link control parameters are computed for every unsaturated session traversing the link. Next, the entries for unsaturated virtual sessions in the saturation map and the rate bitmap for the sessions are changed to "dont know" values. The link now waits for "rate packets" from the virtual sessions with "dont know" rate bits. The unsaturated virtual sessions send backward rate packets towards the source with very high rate values. If a backward rate packet from a receiver reaches a node while the receiver entry in the rate bitmap shows that rate is known, then this means that the rate packet needs to know the rate computed in the next iteration, while the node has not yet performed the next iteration computation. In this case, the node holds the rate packet, till the corresponding entry in the rate bitmap indicates that rate is not known and then it decreases the rate value in the packet to the session link control parameter, $\eta_{\chi(s)l}$, if the rate packet reaches it through link l and if the rate value of the packet is greater than the session link control parameter. Subsequently, the node forwards the rate packet towards the source. After the rate packet reaches the source, the source updates its rate to the maximum of the current rate and the rate value in the rate packet and subsequently generates a forward rate packet towards the receiver, with the same rate value. The forward rate packet contains the minimum session link control parameter on the path of the virtual session and this value is the current iteration rate of the virtual session. When a forward rate packet reaches the node, it updates the corresponding virtual session rate bit to indicate that its rate is known, and updates the session link rate to maximum of the rate in the rate packet and the existing session link rate. When the forward rate packet reaches the receiver, the receiver knows its current iteration rate and generates a probe message to querry its saturation status. The probe message contains the current iteration rate of the virtual session. The probe message travels towards the source. When the probe message reaches a node through link l, the node holds the probe message till all entries in the rate bitmap for link l indicates that the rates are known and then if the node determines that link l satisfies the conditions for saturation, it updates the corresponding entry in the saturation map in its information field and generates forward and backward saturation messages. The forward message travels towards the receiver and the backward message travels towards the source. If the node determines that the link does not satisfy the conditions for saturation, it forwards the probe message towards the source. If the probe message reaches the source, then the source sends a forward unsaturation message towards the receiver. Whenever a node receives a forward or backward, saturation or unsaturation message, it updates its saturation bitnmap accordingly. After a virtual session receives an unsaturation message, it sends a new backward rate packet, because unsaturation message means that all nodes on its path know the saturation status and rates of all virtual sessions and sessions respectively and is thus ready for another iteration computation. Unsaturation message also means that the virtual session has not received its final rate. If a virtual session receives a saturation message, then it knows its final rate and it does not generate any further rate packet. When all entries in the saturation maps are updated to indicate saturation or unsaturation for all sessions traversing a link, and at least one entry is unsaturated, then the origin node of the link computes the link control parameter for another iteration and the process repeats. If all entries are saturated, then the node does not make any further computation.

The distributed implementation terminates in $2DM$ units of time where D is the maximum round trip time from a source to a receiver in the network. This is because after a link control parameter is computed, rates of all virtual sessions traversing the node are known at the node in D units of time. A probe is sent at or before this time and thus the saturation or unsaturation message reaches the receivers and hence the nodes on the way at most D units after this time. Thus a new iteration can start in $2D$ units of time after the first. As Theorem 2 indicates, any node needs to perform at most M iterations. Simulations indicate that in practice, convergence is much faster than the theoretical $2DM$ worst cast bound.

We have described only a framework for distributed computation. This description gives the necessary intuition. Various optimizations can be introduced to make the computation more efficient. For example, it is not actually necessary to maintain a separate entry for every virtual session of a session in its saturation and rate maps. One entry per session per link for rate and saturation status of the session is sufficient. Besides, the rate and probe packets of different virtual sessions of the same session can be merged at branching points, so that there is only one rate (probe) packet of a session in a link. This prevents control information implosion at the source. One can also think of other modifications which should reduce the computation time in most cases. Refer to Technical report [15] for these details. An advantage of this algorithm is that the intermediate rates are always feasible[15]. So during the computation period, sources can still transmit at intermediate rates, without any huge queue buildup. The disadvantage of this distributed computation is that it requires all sessions to start at the same time. That may not be the case. Besides, multicast group membership is dynamic in internet. Receivers join and leave during the lifetime of a session. We present another distributed algorithm for computation of maxmin fair rates which is more suitable for a dynamic scenario. The essential principle behind the two algorithms is the same, but there are certain philosophical differences. We describe it in the next section.

IV. An asynchronous distributed algorithm for computation of maxmin fair rates

We briefly describe the asynchronous algorithm below. For details, refer to the pseudocode in the next subsection. For simplicity, we state without minimum rate requirements in this case. Refer to Technical report [16] for a version with minimum rate requirements.

A. Generic Description

Every source sends a forward rate packet with a rate value (r_p), which is initialized to a very high value initially. The forward rate packet also contains a state bit (u_p). The state bit indicates whether the source should increase its rate or not. The source sets the state bit to 0. Every node maintains a link record for every outgoing link. The link record maintains an estimate of the link control parameter (ψ_l) and a session field for every session traversing the link l. The session field of session i on link l on its path contains the forward and backward session link rates (f_{il} and b_{il} respectively), and a session link saturation bit (w_{il}). Whenever a node receives a forward rate packet of a session, it performs the following sequence of actions for each of its outgoing links on the path of the session *:

1. Generates a forward rate packet. The rate value of the forward rate packet on the link is initially equal to the minimum of the rate value of the incoming forward rate packet and backward session link rate on the link. State bit of the new forward rate packet is initially reset.
2. If the rate value in the newly generated rate packet is less than or equal to the estimated link control parameter, then the session link saturation bit is set. If the rate value in the newly generated rate packet is greater than or equal to the estimated link control parameter of the link, then the state bit of the newly generated rate packet is set and the rate value of the new rate packet is set equal to the estimated link control parameter.
3. Copies the rate value of the new forward rate packet to the forward session link rate entry and transmits the forward rate packet in the respective outgoing link.
4. Computes a new estimate of the link control parameter. This computation is done as follows. Available capacity of a link is the difference between the link capacity and the sum of the forward session link rates of saturated sessions (sessions with session link saturation bit set).
   (a) If no session traverses a link, then link control parameter estimate is equal to the link capacity.
   (b) If all sessions traversing the link have their session link saturation bits set, then link control parameter is equal to the sum of the available capacity of the link and the maximum session link rate, the maximum taken over all sessions traversing the link.
   (c) Otherwise, the link control parameter is the available capacity per unsaturated session traversing the link (a session is unsaturated on a link if its session link saturation bit is reset).

   After the link control parameter is estimated, the session link saturation bits of sessions with forward session link rate strictly greater than the link control parameter estimate are reset. Estimation of link control parameter is repeated with the new set of saturated sessions, if the set of saturated sessions change. (We do not need to repeat this computation more than twice[15].)

Once the forward rate packet reaches the receiver, it records the rate value in the packet as its current iteration rate and generates a feedback rate packet, with the same parameters as the forward rate packet. When a feedback rate packet reaches a node through link l, the node updates the backward session link rate field for the session on link l. The backward session link rate is set equal to the rate value in the feedback rate packet, if the state bit of the backward rate packet is set, otherwise the backward session link rate is set to ∞. Then the node waits for the arrival of feedback rate packets of the same session on other links originating from the node. Once all of them arrive, the node generates a feedback rate packet towards the source of the session. The rate value of this feedback rate packet is equal to the maximum of the rate values of the incoming feedback rate packets, if all the feedback rate packets have the state bit set, otherwise, this rate value is set equal to the rate value of the forward rate packet which reached the node from the source of the session. If the state bit of the same forward rate packet were set or all of the state bits of the incoming feedback rate packets are set, then the state bit of the new feedback rate packet is 1, otherwise it is 0. When the feedback rate packet reaches the source, the source generates another forward rate packet. If the state bit of the feedback rate packet is 0, the new forward rate packet has a very high value of rate (ideally ∞). If the state bit is 1, the rate value in the new forward rate packet is equal to the rate value in the feedback rate packet.

Sessions can enter and exit, receivers can join existing sessions, or leave continuing sessions and link capacities can change any time during the execution of the algorithm. When new sessions join, or links are added to the path of an existing session on account of new receiver joins, the backward session link rates are initialized to very high values, typically ∞. A session is initially considered unsaturated on all the newly added links on its path and the link control parameter estimates of the links on its path are computed. When a session exits, or links are removed from its path on account of receiver leave events, link control parameters on its path are also estimated.

*For example, if a session traverses links l_1 and l_2 originating from node n, then it performs this sequence of actions for both l_1 and l_2.

Theorem 3: If the system stabilizes† at time $t = t_0$, then for all $t \geq t_0 + 5DM$, the receiver rates are equal to the maxmin fair rates. The forward session link rates are equal to the maxmin fair session link rates for all $t \geq t_0 + 5DM$.

The following is the intuition behind the convergence result. The state bit in the rate packet indicates whether session link rates can be increased or not. If the estimated rate (rate value in the rate packet), is less than the fair share of the link capacities (link control parameters) on its path, then the rate packet returns with state bit reset and the rate estimate is increased next time. If the estimated rate is greater than or equal to the fair share of some link on its path, then the estimated rate is reduced to this fair share and the state bit is set, so that the next rate packet does not contain a greater estimate of the rate. The forward session link rates recorded at the nodes are set to the most recent value of the estimate (rate value in the most recent forward rate packet). Initially, the estimate is set to a very high value (typically, ∞). Subsequently, increase and decrease triggered by the state bits bring the convergence of the session link rates to the fair shares and consequently the receiver rates converge to the respective fair shares. Again, experimental studies indicate that in practice, convergence is much faster than the theoretical $5DM$ worst cast bound.

The advantage of this algorithm is that, unlike the synchronous algorithm, it need not restart every time there is a small change in the system. Thus, in practice, the algorithm may be faster than the sysnchronous one in a dynamic scenario, e.g., if multicast session membership changes frequently. The disadvantage of this algorithm is that its worst case convergence time is greater than that of the synchronous one. Besides, it needs to send control packets and estimate link control parameters all along the operation of the system. The synchronous algorithm need not send any control packets nor make any computation after the virtual sessions are saturated, till any further change in the system. So the synchronous algorithm is a better choice if changes are infrequent. Another disadvantage of the asynchronous algorithm is that the intermediate virtual session rates may not be feasible and this may create huge queues in transience period. This does not happen for the synchronous algorithm. Depending on the requirements of the system, any one of the above algorithms may be chosen.

A distributed asynchronous algorithm for computation of maxmin fair rates in unicast network, presented in [6], is a special case of ours. If the network is unicast, then it is not necessary to keep track of backward session link rates at the nodes at all. The algorithm in [6] maintains only the forward session link rates and saturation bits for sessions at the nodes. However, we need to keep track of this additional information for multicast networks (at least at nodes where fanout is greater than 1) to account for the subtlety that a session rate in a link may be greater than rates of some session receivers downstream.

†System stabilizes means link capacities do not change or sessions do not enter and exit any further.

B. Pseudocode for asynchronous distributed algorithm

We first introduce certain terminologies.
ρ_i is the rate of transmission of session i
σ_s is the rate of reception of receiver s.

$$\left.\begin{array}{ll} r_p & \text{Rate Value of Rate Packet} \\ u_p & \text{State Bit of Rate Packet} \end{array}\right\} \text{Rate Packet}$$

$$\left.\begin{array}{ll} v_i^n & \text{Forward session bit for session } i \\ g_i^n & \text{Incoming session rate for session } i \\ x_i^n & \text{Feedback session bit for session } i \\ y_i^n & \text{Feedback session rate for session } i \\ \tau_i^n & \text{Set of links originating from node } n \\ & \text{on the path of session } i \\ & \left.\begin{array}{ll} f_{il} & \text{Forward Session link rate} \\ b_{il} & \text{Backward Session link rate} \\ w_{il} & \text{Session link saturation bit} \\ h_{il} & \text{Session link hold bit} \end{array}\right\} \text{(b)} \end{array}\right\} \text{(a)}$$

(a) gives the information field for session i on node n, if session i traverses node n. (b) gives the information field for session i on outgoing link l of node n, if session i traverses link l.

$$\left.\begin{array}{ll} \psi_l & \text{Estimate of link control parameter} \\ \kappa_l & \text{Set of sessions traversing link } l \end{array}\right\} \text{Link } l \text{ record}$$

$\vec{f}^l$ is the forward session link rate vector, with components f_{il}s.
$\vec{w}^l$ is the session link saturation bits vector, with components w_{il}s.
$|(\&)$ indicates logical or (and) operation.

The pseudocode follows.

At source of session i:
1. If this is the first rate packet, $\rho_i = \infty$.
2. Otherwise, when a feedback rate packet(r_p, u_p) is received,
   (a) If $u_p = 0$, $\rho_i = \infty$.
   (b) If $u_p = 1$, $\rho_i = r_p$.

generate-forward-rate-packet$(\rho_i, 0)$ and send it downstream.

At node n :
When session i forward rate packet(r_p, u_p) is received:
1. $g_i^n = r_p$, $v_i^n = u_p$, $x_i^n = 1$, $y_i^n = 0$, and $\forall\ l \in \tau_i^n$, perform the following actions:
2. $h_{il} = 1$, $z_{il1} = \min(b_{il}, r_p)$, $z_{il2} = 0$,
3. If $z_{il1} \leq \psi_l$, $w_{il} = 1$ and if $z_{il1} \geq \psi_l$, $z_{il1} = \psi_l$ and $z_{il2} = 1$.
4. $f_{il} = z_{il1}$,
5. **generate-forward-rate-packet**(z_{il1}, z_{il2}) and transmit it on link l,

6. **estimate-link-control-parameter**$(\vec{f}^l, \vec{w}^l, C_l)$.

z_{il1} and z_{il2} are local variables.

When session i feedback rate packet(r_p, u_p) reaches node n via link l,

1. If $u_p = 1$, $b_{il} = r_p$, else $b_{il} = \infty$,
2. $x_i^n = x_i^n \& u_p$,
3. If $u_p = 0$, $y_i^n = g_i^n$, else $y_i^n = \max(y_i^n, r_p)$,
4. $h_{il} = 0$,
5. If $\sum_{l \in \tau_i^n} h_{il} = 0$, **generate-feedback-rate-packet**$(y_i^n, v_i^n | x_i^n)$ and send this feedback rate packet towards the source of session i.

At a receiver s of session i :
When a forward rate packet(r_p, u_p) is received, $\sigma_s = r_p$, **generate-feedback-rate-packet**(r_p, u_p) and send it towards the source of session i.

Let n be the origin node of link l and let link l be added to the path of a session i. Session initiation can cause addition of links on its path. A link may also be added to the path of an existing session on account of some new receiver joins.

1. $w_{il} = 0$, $\kappa_l = \kappa_l \cup \{i\}$, $b_{il} = \infty$,
2. If session i was not traversing node n, prior to addition of link l, $\tau_i^n = \{l\}$, else $\tau_i^n = \tau_i^n \cup \{l\}$.
3. **estimate-link-control-parameter**$(\vec{f}^l, \vec{w}^l, C_l)$ (f_{il} can be initialized arbitrarily).

Note that we do not need to initialize other parameters because of the operation of the algorithm.

Let n be the origin node of link l and let link l be removed from the path of a session i. Session exit can cause removal of all links on its path. A link may also be removed from the path of a continuing session, if all receivers downstream leave.

1. $\tau_i^n = \tau_i^n \setminus \{l\}$, $\kappa_l = \kappa_l \setminus \{i\}$,
2. **estimate-link-control-parameter**$(\vec{f}^l, \vec{w}^l, C_l)$.

If capacity of a link changes anytime, then the link control parameter need simply be reestimated, with the new link capacity.

Subroutine **generate-forward-rate-packet**(r_p, u_p) (**generate-feedback-rate-packet**(r_p, u_p)) simply generates a forward(feedback) rate packet with rate value equal to r_p and the state bit equal to u_p. We describe **estimate-link-control-parameter**$(\vec{f}^l, \vec{w}^l, C_l)$ below.

estimate-link-control-parameter$(\vec{f}^l, \vec{w}^l, C_l)$ {
If $\kappa_l = \phi$, $\psi_l = C_l$.
If $w_{il} = 1$, $\forall\ i \in \kappa_l$, $\psi_l = C_l - \sum_{i \in \kappa_l} f_{il} + \max_{i \in \kappa_l} f_{il}$
If $w_{il} = 0$, for some $i \in \kappa_l$, $\psi_l = \frac{C_l - \sum_{i \in \kappa_l} w_{il} f_{il}}{|\kappa_l| - \sum_{i \in \kappa_l} w_{il}}$
While $w_{il} = 1$, and $f_{il} > \eta_l$, for some $i \in \kappa_l$, then
{ $w_{il} = 0$,
estimate-link-control-parameter$(\vec{f}^l, \vec{w}^l, C_l)$}
end subroutine;}

V. Allocation of Rates

In this section, we briefly discuss the generic mechanisms for allocation of the fair rates, once they are computed.

Once the fair rates are computed, the rate of a signal flow can be adjusted from within the network. Source transmits at a rate equal to the maximum of the fair rates allocated to its receivers. Nodes know the maxmin fair session link rates in the outgoing links on account of the distributed computation. At forking points, video gateways transcode signal into a lower bit rate such that the rate in every link is equal to the maxmin fair session link rate, which is the maximum of the fair rates allocated to the session receivers downstream. Rate adaptive video gateways [1] can be used to fine tune the transmission rate to exactly match the maxmin fair session link rate.

Another possibility is to use hierarchical encoding. In this approach, a signal is encoded into a number of layers that can be incrementally combined to provide progressive refinement, at the expense of additional bandwidth requirement. Hierarchical coding was first suggested for packet voice transmission[5] and subsequently for video, e.g., [18]. Some hierarchical coding schemes are amenable to tuning of layer bandwidths, e.g., [11]. Once the fair rates are computed, the source partitions its signal to form as many layers as there are distinct receiver rates and tunes the layer bandwidths to match the receiver rates. Each receiver is allotted layers as per its fair share of bandwidth. A node forwards a layer on an outgoing link only if it is allotted to some receiver downstream.

A perfect match between the layer bandwidths and the maxmin fair receiver rates is not always feasible due to limitations on such parameters as the number of layers and the minimum bandwidth assigned to a layer on account of filter restrictions, minimum packet size requirements and limitations on packetization delay[15]. The following approach can be adopted in cases of fixed or partially adaptive signal hierarchy. Allocate to the receivers as many layers as permitted by the computed fair rate. If the total bandwidth consumed by the layers allocated to a receiver is strictly less than the fair computed rate, allocate one more layer to the receiver and let the network drop a certain portion of packets of the last layer at a forking point. For certain coding schemes, packet loss causes a graceful degradation in signal quality. This mechanism will be useful in those coding schemes

VI. Conclusion

Summarizing, we have presented an algorithm for computation of the maxmin fair rates in arbitrary multicast networks and a framework for distributed implementation of the same. The algorithm takes care of minimum rate requirements, which are often present in ATM ABR multicast scenario. This algorithm can be used to make maxmin fair rate allocation, in presence of bandwidth adaptive hierarchical coding or when the network has certain provisions like transcoding option. Several well known algorithms like the algorithm for computation of maxmin fair rates in arbitrary unicast network and algorithm for computation of maxmin fair rates in multicast network with the restriction that all receivers of the same session must receive service at the same rate are in fact special case of ours. We have also presented another distributed algorithm for computation of maxmin fair rates in a multirate multicast network which is more suited to asynchronous operation, i.e., when sessions enter and exit at different times. No distributed algorithm (synchronous or asynchronous) has been developed before in multirate multicast networks, with or without maximum, minimum rate constraints. The only existing algorithm for computation of maxmin fair rates in multicast networks is centralized[13]. Reference[17] presents a distributed algorithm for computation of maxmin fair rates in unirate multicast networks (all receivers of the same session must have same rate in unirate multicast networks) though . However, unirate multicast networks do not differ significantly from unicast networks. As discussed before, multirate multicast introduces new complications. Before this work, no distributed algorithm has been proved to converge to maxmin fair rates even in unicast networks with minimum rate constraints. Reference[8] presents one such algorithm. This algorithm does not attain its claimed convergence on account of certain subtle flaws[15].

There exists an array of related intellectually challenging unsolved research problems. On account of computational, financial, and end-to-end delay constraints, transcoding may not be a feasible option. Hierarchical signal structure may be predetermined and layer granularity may be lower bounded. "Partial" subscription to a layer by dropping a certain proportion of packets may not be possible. The consequence is that *any* rate vector in the current feasible set can not be allocated. Rather, only a discrete subset of rate vectors in the feasible set can actually be allocated on account of the constraints on the signal structure. Fairness in a discrete feasible set is vastly different from fairness in our current feasible set. It opens up many new research problems. We address those in [16].

We have not described the exact message exchange format for distributed computation of the maxmin fair rates, but have suggested the general framework. Also we have only discussed the generic mechanisms for fair rate allocation in presence of certain provisions like transcoding options and/or adaptive hierarchical encoding, but we did not describe the exact protocol format for the purpose. Developing the detailed protocols for message exchanges in these cases would be interesting from an implementation point of view. Besides, while describing both the distributed algorithms, we had implicitly assumed that the control messages (e.g., rate packets, probe messages, saturation, unsaturation messages) are never lost. However, in internet like lossy networks, control messages may be lost. We need to devise retransmission and timeout strategies. This is a topic of future investigatation.

References

[1] E. Amir, S. McCanne and H. Zhang An Application Level Video Gateway *Proceedings of ACM Multimedia' 95*, pp 255 – 265, San Francisco, CA, November 1995. ACM.

[2] S. P. Abraham and A. Kumar. Max-Min Fair Rate Control of ABR Connections with Nonzero MCRs. *Proceedings of IEEE Globecom' 97*, pp 498 – 502

[3] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees: an architecture for scalable inter-domain multicast routing, *Proceedings of ACM SIGCOMM,* Sept. 1993, pp. 85-95

[4] D. Bertsekas and R. Gallager, *Data Networks*, Englewood Cliffs, NJ:Prentice-Hall, 1987.

[5] T. Bially, B. Gold, and S. Seneff. A technique for Adaptive Voice Flow Control in Integrated Packet Networks, *IEEE Transactions on Communications,* Vol. 28, No. 3, March 1980, pp. 325 – 333

[6] A. Charny, An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback. *M.S. thesis*, Massachusetts Institue of Technology, May 1994.

[7] S. Deering and D. Cheriton. Multicast routing in datagram internet works and extended LANs, ACM Transactions on Computer Systems, vol 8, no. 2, pp. 54-60, Aug. 1994.

[8] Y. T. Hou, H. H. Y. Tzeng and S. S. Panwar. A Generalized Max-Min Rate Allocation Policy and its Distributed Implementation Using the ABR Flow Control Mechanism, *Proceedings of IEEE Infocom' 98*, March 1998

[9] X. Li, S. Paul and M. H. Ammar. Layered Video Multicast with Retransmission (LVMR): Evaluation of Hierarchical rate control, *Proceedings of IEEE Infocom' 98*, March 1998

[10] X. Li, S. Paul and M. H. Ammar. Multi-session Rate Control for Layered Video Multicast *Technical Report* GT-CC-98 – 21, College of Computing, Georgia Institute of Technology, 1998

[11] S. McCanne, Scalable Compression and Transmission of Internet Multicast Video. *PhD thesis*, Univ. of California, Berkeley, December 1996.

[12] S. McCanne, V. Jacobson and M. Vetterli. Receiver-Driven Layered Multicast, *Proceedings of ACM SIGCOMM '96*, Stanford, CA, September 1996

[13] D. Rubenstein, J. Kurose and D. Towsley. The Impact of Multicast Layering on Network Fairness *Proceedings of ACM SIGCOMM '99, Cambridge, MA, September,* 1999

[14] S. Sarkar and L. Tassiulas: A Framework for Routing and Congestion Control in Multicast Networks *Proceedings of IEEE INFOCOM'99*, New York, NY, March' 99

[15] S. Sarkar and L. Tassiulas: Distributed Algorithms for Computation of Fair Rates in Multirate Multicast Trees *Technical Report* TR 99 – 42, Institute for Systems Research and University of Maryland, 1999.

[16] S. Sarkar and L. Tassiulas: Fair Allocation of Discrete Bandwidth Layers in Multicast Networks *Submitted to IEEE INFOCOM' 2000*

[17] H. Y. Tzeng and K. Y. Siu. On Max-Min Fair Congestion Control for Multicast ABR Service in ATM, *IEEE Journal on Selected Areas In Communications*, Vol 15, No. 3, 1997.

[18] D. Taubman and A. Zakhor. Multirate 3-D Subband Coding of Video, *IEEE Transactions on Image Processing*, Vol 3, No. 5, September 1994.