Fair Distributed Congestion Control in Multirate Multicast Networks*

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@eng.umd.edu, leandros@isr.umd.edu

Abstract

We study transmission of real time signals in arbitrary networks with multicast capabilities. Multicast traffic in internet and ATM provides a motivation for studying these networks. Real time signals can be encoded into several levels of precision, and the precision level of the signal delivered to a receiver depends on its path bandwidth and processing capabilities only. Different receivers of the same multicast session can receive service at different rates under this multirate encoding. This makes resource allocation in multirate, multicast networks significantly different from that in unicast or unirate multicast networks. We study fairness of resource allocation in multirate, multicast networks. We assume that the bandwidths required for transmission of different levels of precision can be dynamically adjusted depending upon the congestion level of links and the requirements of the receivers. This allows the receivers to receive service at any rate, subject only to feasibility constraints, and the receivers are not constrained to receive only a discrete set of rates dictated by the bandwidths consumed for different levels of multirate encoding. We develop a mathematical framework to model the fair allocation of bandwidth in this case, with minimum and maximum rate constraints. We present a necessary and sufficient condition for a rate allocation to be fair in a multirate, multicast network. We propose a distributed algorithm for computation of maxmin fair rates allocated to various source-destination pairs. This algorithm has a low message exchange overhead, and is guaranteed to converge to the maxmin fair rates in finite time.

1 Introduction

Multicasting provides an efficient way of transmitting information from a sender to a group of receivers. A single source node or a group of source nodes sends identical messages simul-

*Parts of this paper have been presented at GLOBECOM' 2000 and INFOCOM' 2000.
†Corresponding Author.

taneously to multiple destination nodes. Single destination or unicast and broadcast to the entire network are special cases of multicast. Multicasting reduces bandwidth consumption as message replication takes place at forking nodes only. This is particularly useful for real time group communications like audio or video teleconferencing, video-on-demand services, distance learning etc., as these applications consume a lot of bandwidth. We would study resource allocation for real time multicast applications. Multicasting has been used primarily in the Internet, but future ATM networks are likely to deploy multicasting in a large scale, particularly in applications like broadcast video, video-conferencing, multiparty telephony and workgroup applications[10].

There are two possible transmission modes in multicast networks for loss tolerant real time traffic like audio, video, etc. In one, all receivers in the same session receive information at the same rate. However, this unirate transmission has severe shortcomings for multicast networks. This is because of network heterogeneity. 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). 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. For real time traffic, multirate transmission can be used to counter network heterogeneity. The receivers of the same session are allowed to receive at different service rates. Multirate transmission can be attained by transcoding and layered encoding. Both of these exploit the fact that real time traffic is loss tolerant and can adapt to congestion by sacrificing precision of representation. In transcoding, the source transmits at a rate matching the fastest of its receivers. The transmission rates are transcoded at the intermediate nodes to match the capabilities of slower receivers downstream[29]. At every link, the transmission rate of a session is equal to that of the fastest session receiver downstream of the link. Video gateways are generally used for transcoding[2]. The other approach is to have a hierarchical or a layered transmission scheme. In this approach, a signal is encoded into a number of layers that can be incrementally combined to provide progressive refinement. The different layers of a multicast session are considered different multicast groups and receivers adapt to congestion by adding and dropping layers, where adding a layer is joining a multicast group and dropping a layer is leaving a group[12]. As each layer is added, there is an improvement of quality of the received signal and additional bandwidth is needed to transport the combined stream, and dropping a layer brings about graceful degradation in quality. Again the number of layers of a session in a link is the maximum of the number of layers of the session receivers downstream. This layered transmission scheme have been used for both video[29] and audio[6] transmissions over the internet and has potentials for use in ATM networks as well[13]. We would discuss coding schemes which attain multirate transmission for multicast networks, in greater details later.

Layer bandwidths may be flexible or predetermined. In the former, layer bandwidths can be tuned to match desired receiver rates and as a result any rates can be allocated, e.g., [21]. The feasible set is continuous. In the latter, layer bandwidths are predetermined and can not be tuned according to the needs of the network. A receiver either receives a layer fully or does not receive the layer, at all. It can not partially subscribe to a layer. Effectively, the network can only allocate a discrete set of rates to the receivers, whereas a continuous set of rates can be allocated when receivers can subscribe to fractional layers. We study the continuous case here and have studied the discrete case in [28]. As [28] indicates, the resource allocation problem is completely different in the two cases, and calls for separate investigation.

We would study resource allocation in multirate, multicast network from the point of view of fairness. Max-min fairness[5] is a well accepted definition of fairness. A bandwidth allocation is *max-min fair*, if no receiver can be allocated a higher service rate without lowering that of another receiver having equal or lower rate. As [24] points out, max-min fairness satisfies many intuitive fairness properties in a multirate multicast network. We would focus on maxmin fair allocation of rates in multirate multicast networks. The objective is to serve every receiver of every session at a fair rate. The service rate of a receiver should be independent of those of other receivers of the same session, and should depend only on the bandwidth of the path leading the receiver, and its processing capability. This is not an issue in a unicast network, since there is only one receiver per session there. Hence, the problem of allocating fair rates is more complicated in multicast networks.

We first review the previous work in this area. We have proposed a routing and scheduling policy in [25] which stabilizes the system, if the network can accommodate all the traffic demands, without congestion. However, resource limitations may not allow fulfilling all traffic demands, particularly for bandwidth expensive real time applications. Fairness of resource allocation becomes important in such a scenario. Tzeng *et al* has investigated the problem of fair allocation of bandwidths to multicast sessions under the constraint that all receivers of the same session must receive service at the same rate[30]. However, under this unirate transmission, service rate may not match the path bandwidth and the processing capability of every receiver, on account of network heterogeneity. As [24] demonstrates formally, fairness properties of a multicast network improves if multi-rate transmission is used instead of single rate transmission as [30] advocates. Now, [9] advocates simulcast, but that is bandwidth inefficient. Fair allocation of layers (rates) in presence of layered transmission (transcoding) provisions have not been cultivated very systematically, until recently[24]. There are two well known network protocols for layered transmission, RLM (Receiver-driven Layered Multicast)[22] and LVMR (Layered Video Multicast with Retransmissions)[17]. The goal of these approaches is to achieve improved fairness among members of the same session. However, as [18] points out, neither handles fairness among members of different sessions very well, when there are multiple sessions competing for bandwidth. A scheme for fair allocation of layers for multi-session layered video multicast in [18] strives to rectify this defect in RLM and LVMR. The authors present empirical evidence that the scheme improves inter-session fairness for networks with multiple video sessions sharing only one link. There is no experimental or analytical evidence that the scheme works well for more complex networks, with sessions sharing several links with each other. In absence of further mechanisms, like elaborate scheduling policies, it may not be possible to establish conclusively that the scheme attains fair allocation of rates as per some well defined notion of fairness, like maxmin fairness for example. Besides [18] does not make any effort towards the computation of the actual rates or the number of layers 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 [24]. We describe it later. 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 in Section 2. This section describes the problem of maxmin fair allocation in multicast transmission and presents an interesting property of the maxmin fair rate allocation. We present an algorithm for computation of maxmin fair rates in an arbitrary network with any number of multicast sessions in Section 3.1. This algorithm does not require global knowledge of system states during any stage and is thus amenable to distributed computation. We give a framework for scalable distributed implementations of the above algorithm in Section 3.2. It turns out that the algorithms for computation of maxmin fair rates in unicast networks or multicast networks with a single rate of transmission per session are special cases of the algorithm we present here. Our algorithm has low control message exchange overhead, and is tailormade for networks with overhead limitations. We present analytical and experimental performance evaluations. Analytical results guarantee that the distributed algorithm converges to the maxmin fair rates in finite time, and presents worst case convergence time bounds. Section 4 addresses generic mechanisms for allocation of the rates, once the rates are computed. Finally, we conclude with some future directions of research in Section 5. We think a unified mathematical framework can model the issue of fairness of resource allocation for multicast in both ATM and the internet. The modalities of the rate allocation and the implementational details may differ in the two scenario. As a first step, we do not distinguish between the two. 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 framework. On account of space limitation, we present outlines of our proofs in this paper. Details can be found in technical report[26].

2 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). There is a tree associated with each session that carries the traffic of the session. 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[11], MOSPF[20], CBT[4], PIM[12] and MIP[23] 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 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. In a connectionless network, sessions can not have any such requirement as the network would never know of these, and would have to make rate allocation irrespective of any such requirement. Such a scenario can very well be accommodated in our model, by assuming minimum rate requirement as 0 and maximum rate to be ∞ 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, $(n, v, u_1), \ldots, (n, v, u_t)$. The network shown in figure 1 has a single session, $(1, v, U)$, with $U = \{u_1, \ldots, u_4\}$. 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 will not serve our purpose. We assume that every virtual session (source-destination pair) has a minimum and a maximum rate.

At this point, we would like to mention that by "rate" of a virtual session, we generally mean the average rate at which the corresponding receiver receives information. Because of the burstiness of traffic, it may not be a good idea to utilize the full capacity of the links. To prevent excessive delay and severe performance degradation, the network designer may like to have the average rates sum up to a certain fraction of the actual capacity of a link. In this context, the term "capacity of a link" which we use henceforth, stands for that fraction of the actual link capacity, which the network designer wants the sessions to use.

Informally speaking a rate allocation for the virtual session 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 in all in the entire network. Rate allocation is an M-dimensional vector $(r_{11}, \ldots, r_{1m_1}, \ldots, r_{i1}, \ldots, r_{im_i}, \ldots, r_{N1}, \ldots, r_{Nm_N})$, with r_{ij} being the rate allocated to the jth virtual session of the ith session. For simplicity we will use a single index, henceforth. A rate allocation $(r_1, \ldots, r_M)$ is feasible if the following conditions hold.

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. Let $n(l)$ denote the set of sessions passing through link l, $m(k,l)$ denote the set of virtual sessions of session k passing through link l and C_l denote the capacity of link l. Then,

$$\sum_{i \in n(l)} \max_{j \in m(i,l)} r_j \leq C_l \qquad \text{(capacity condition)}$$

   The rate allocated to session i in link l under rate allocation $\vec{r}$ is λ_{il}. 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 $\sum_{i \in n(l)} \lambda_{il} \leq C_l$.

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

We would like to point out that resource allocation for multirate, multicast traffic is significantly different from that of unicast traffic. Let us consider a network with N sessions and M receivers in all. If all sessions are unicast, rates allocated to all receivers of a session are the same. Hence, we can consider session rates. In this case, rate allocation is an $N-$dimensional vector. The set of feasible allocations is defined by the following capacity constraints. An $N-$dimensional vector $(r_1, \ldots, r_N)$ is a feasible rate vector, if sum of the rates of the sessions traversing a link is less than or equal to the capacity of the link, for each link, and the session rates respect the minimum and maximum rate constraints. More formally, $(r_1, \ldots, r_N)$ is a valid rate allocation vector if

$$\mu_i \leq r_i \leq p_i, \quad \forall i \qquad \text{and} \qquad \sum_{i \in n(l)} r_i \leq C_l$$

where $n(l)$ is the set of sessions traversing link l. Note that the description of the feasible set is the same for unicast and unirate multicast network. The only difference is that the route of a session is a tree instead of a path. The description of the feasible set becomes more complicated for multirate multicast networks. Different receivers of the same session receive service at different rates. Hence, there is no session rate as such. Rather, the rate allocated to a session in a link is equal to the maximum of the rates allocated to receivers downstream of the link. So session rate is different in different links. Thus, we consider rates allocated to individual receivers instead. So rate allocation is an $M-$dimensional vector now. Also the capacity constraints are nonlinear and involves the max operator, as opposed to linear capacity constraints in the unicast case.

Resource allocation can be studied from several perspectives. We would study resource allocation from the point of view of fairness. Fair allocation of rates has several advantages. Networks often provide class based service. Users belonging to the same class have the same service agreements and pay the same revenues. So, it is expected that the network would treat the members of the same class similarly, i.e., the network should be fair within the same class. Besides, fair allocation of service does not penalize any user excessively, and naturally allocates a certain minimum quality of service to all users. User satisfaction is often a concave function, i.e., satisfaction increases rapidly with service rate increase at small service rates, and increases slowly with service rate increase at high service rates. Thus total user satisfaction is often improved if all users obtain an equitable quality of service, rather than some users obtaining high quality of service at the expense of others obtaining low quality of service. Another possible objective can be to maximize the revenue, or more generally the total system utility. Every receiver has a utility for the bandwidth allocated to the receiver, and the system utility is the sum of the receiver utilities. Utility of a receiver may be the revenue paid by the receiver, or may reflect the satisfaction of the receiver. The current utility maximization techniques need explicit knowledge of the utility function at the receiver[19]. But, a receiver may not know its utility function explicitly. For example, perceptual quality is a possible utility, and there is no exact expression explicitly connecting perceptual quality and service rate. More general resource allocation techniques like fair allocation of resources is preferrable in these cases, as these techniques do not assume explicit knowledge of utility functions, and perform well irrespectively.

There are several possible notions of fairness. A simplistic notion is to serve all users at the same rate. However, some users may receive information through poor quality links, while some others may receive information through high quality links. Serving all users at the same rate would tie down all users to low quality service in this case. So this notion of fairness is not useful. Max-min fairness is a well accepted notion of fairness. Other notions of fairness like proportional fairness[14] are also known. We would focus on maxmin fair allocation of rates in multirate multicast networks. 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.
Refer to figure 2 for an example of maxmin fair allocation.

Henceforth we shall ignore the maximum rate constraints. This does not cause any loss of 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 a significant 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 across 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 travelling across 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 possible rate of some virtual session in $m(\chi(j), l)$ (set of virtual sessions travelling across 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 $\mu_{\chi(j)l} = \max_{p \in m(\chi(j),l)} \mu_p$. The maximum of the minimum rate requirements of the virtual sessions in $m(i, l)$ is μ_{il}.

Refer to figure 2 for an example of bottleneck links.

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 very simple for multicast networks as well. 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 in link l are greater than the rate of virtual session s. We will use this lemma in proving the correctness of an algorithm for computation of a maxmin fair rate allocation. We have proved this lemma in [26]. We give the intuition behind the proof here. Ignore the minimum rate constraints for this intuition. If a virtual session is bottlenecked in a link, then it has the maximum rate amongst all virtual sessions traversing the link, and the capacity of the link is fully utilized. If a virtual session s has a bottleneck link, then any increase in rate of a virtual session must accompany a decrease in rate of some virtual session traversing the bottleneck link and all virtual sessions traversing the bottleneck link has rate equal or less than that of s. Sufficiency follows. If a link is not bottlenecked w.r.t. virtual session s, then either the capacity of the link is not fully utilized or some virtual session traversing the link has a greater rate. Thus the rate of a virtual session s can always be increased without hurting any other virtual session having equal or lower rate, if s does not have a bottleneck link. Necessity follows. Minimum rate requirements complicate the proof somewhat more. The formal proof in [26] includes these details.

3 A distributed scheme for computation of maxmin fair rates

We proceed to present a distributed scheme for computing the maxmin fair virtual session rates. We first give a generic description of the basic algorithm and then discuss its distributed implementation.

3.1 Basic Algorithm

We describe the basic algorithm here. We first present some notations. Terminologies $n(l)$, $m(i, l)$, $\chi(s)$ and μ_{il} are as defined in pages 6, 6, 9 and 9 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 session i in 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. This bandwidth would have been allocated to the unsaturated virtual sessions traversing the link, if there were no bandwidth constraints on other links and also if the virtual sessions did not have any minimum rate constraints. As we shall see later, if a virtual session is saturated, then its rate is already determined.

$\eta_{il}(k)$ 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. Session link parameter, $\eta_{il}(k)$ can be computed as $\eta_{il}(k) = \max(\eta_l(k), \mu_{il})$.

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 $\eta_l(k) = \eta_l(k-1)$. Here, $\eta_{il}(k) = \max(\eta_l(k), \mu_{il})$.*

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 $r_s(k) = r_s(k-1)$.

*This computation need be done for all unsaturated sessions traversing link l only.

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, $S(k) = S(k-1) \setminus \{s : \exists\, l \in L_s, \text{ s.t. } \sum_{i \in n(l)} \lambda_{il}(k) = C_l \text{ and } r_s(k) = \lambda_{\chi(s),l}(k)\}$.

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 link l at the end of the kth iteration: $\Xi_l(k) = \{n : n \in \{1, \ldots, N\},\, m(n,l) \cap S(k) \neq \phi\}$.

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 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 the first iteration, is the capacity of the link per session traversing the link. If a virtual session traversing link l had no bandwidth constraint in 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 through the link with the minimum link control parameter are saturated[26]. 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).

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 parameter on link l, $\eta_{\chi(s)l}(k)$, where $\chi(s)$ is the session of virtual session s. Session link 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, and the process continues till all virtual sessions are saturated.

We illustrate the operation of the algorithm with an example.

Example 3.1.1: Consider the network of figure 2. The maximum rate constraints do not exist. 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: $\eta_{e_1}(2) = 4$, $\eta_{e_3}(2) = 3.5$, $\eta_{e_5}(2) = 4$, and $\eta_l(2) = \eta_l(1)$ for the rest of the links. Now, $\eta_{1e_1}(2) = 4$, $\eta_{1e_3}(2) = 3.5$, $\eta_{1e_5}(2) = 4$. 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. The rates obtained upon termination are $(4, 3.5, 3)$.

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 *The algorithm terminates in at most M iterations, where M is the number of virtual sessions, and upon termination it yields the max-min fair rate vector.*

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 formal proof is presented in [26]. We sketch the proof in the appendix.

Some well known algorithms for computation of maxmin fair rates in the unicast scenario, e.g., [3], [5], are special cases of this algorithm. The simplest version of this algorithm for unicast networks is in [5]. Later [3] and [16] generalized this algorithm to take care of minimum rate constraints in ATM ABR unicast networks. We have considered a multicast network with minimum and maximum rate constraints. 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 generalization and the proof need 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. The same does not hold for multicast networks. Refer to figure 3 for a counter example.

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. This condition also holds in multicast networks without any minimum rate requirements.

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* $L_i \cap L_j \neq \phi$.

To show this, as in the proof of the second part of Theorem 1, we consider link $l_{\min}$, the link which attains the minimum link control parameter amongst all those which carry at least one unsaturated virtual session. In the more general case, every unsaturated virtual session, of a particular session, $i_{\min}$, traversing through $l_{\min}$ saturate. However in this case, all unsaturated virtual sessions traversing $l_{\min}$ saturate. Thus by $|\mathcal{L}|$ iterations, all virtual sessions are saturated. The result follows. The formal proof is in [26].

Every step of this algorithm has a complexity of $O(|\mathcal{L}|M)$. The algorithm must terminate in M iterations (Theorem 1). Thus the overall complexity of this algorithm is $O(|\mathcal{L}|M^2)$.

We conclude this section with a detailed comparison with the algorithm proposed in [24]. The algorithm proposed in [24] is the only other algorithm known for computation of maxmin fair rates in multirate, multicast networks. This algorithm is centralized in nature. We describe this algorithm and point out the steps which require global information. This algorithm iterates on the set of receivers. In every iteration, it identifies a set of candidate receivers whose rates would be incremented in the next iteration. Initially, the set of candidate receivers consist of all receivers of all sessions. This candidate set is trimmed as the computation proceeds. In every iteration, the rate of every receiver in the candidate set is incremented *uniformly* . The incrementation amount is the largest possible value permitted by the capacity constraints of the links. A receiver is removed from the candidate set once some link in its data path reaches full capacity. The incrementation amount in every iteration is decided with global knowledge. This is evident since the incrementation amount is the same for all receivers in the candidate set, and the rates allocated after incrementation must satisfy the capacity constraint in every link. Our algorithm does not require global information at any stage. The set of unsaturated receivers is similar to their set of candidate receivers. The key difference which allows distributed operation is that we allow the rate incrementation to be different for different unsaturated receivers. The incrementation amount for a receiver depends only on the information along its path. This allows distributed operation, but introduces its own complications. For example, the trimming of the candidate set is different from trimming the set of unsaturated receivers. It is not sufficient to consider a receiver saturated if a link in its information path reaches its capacity. In addition, the receiver must also receive the maximum bandwidth amongst all receivers of the same session traversing the link. The proof for convergence to maxmin fair rates differs substantially from the centralized case. Also, the centralized algorithm of [24] does not consider minimum rate requirements. In general, minimum rate requirements complicate the problem of maxmin fair rate allocation, and in particular, as Lemma 2 indicates, the algorithm complexity increases in presence of minimum rate requirements.

We describe the distributed implementation of this algorithm in the next subsection.

3.2 Distributed Implementation of the Basic Algorithm

We describe a framework for distributed implementation of the basic algorithm presented in the previous subsection. We will exploit the fact that the computation of the session link parameters of any link needs only information about the saturation status of sessions traversing the link and the previous iteration rates of the unsaturated sessions traversing the link. A virtual session (receiver) can determine its rate if it knows its session link parameters in 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. Our implementation framework is scalable in control information exchange overhead. i.e., the amount of control messages of a session in a link in an iteration is independent of the number of receivers downstream of the link. This is attained by merging the control information sent by different receivers at branching points. For example, control information sent by receivers u_1 and u_2 are merged at node I in the network of figure 2. This prevents control information implosion at the source. Also, the system does not maintain separate entries for separate virtual sessions traversing a link, but there is only one entry for each session in a link. This is an important requirement for any realistic distributed computation scheme, because current day multicast routers are oblivious of the number and identity of the receivers downstream of a link. At the same time, it is not obvious how to implement this distributed algorithm without maintaining these separate entries, because it assumes that a link processor knows the individual virtual sessions traversing the link. For example, we consider a session to be saturated in a link only when all its virtual sessions traversing the link are saturated(step 8). However, we present a storage and message exchange format which implements the distributed computation without explicit knowledge about the individual virtual sessions traversing a link.

We first describe the storage in every node. Every node maintains an information record for each of its outgoing links l. The record maintains the following entries for every session i traversing link l : (a) minimum session link rate: μ_{il}, (b) session link rate, (c) rate bit and (d) saturation entry. Note that the storage does not maintain separate information about different session i virtual sessions traversing link l. Computations under this implementation correspond to iterations of the basic algorithm. The implementation does not track the iteration numbers explicitly. If the rate bit is set, then the current iteration session link rate of the basic algorithm is known at the node. The session link rate is a rate value which equals the current iteration session link rate in the basic algorithm, if the rate bit is set. The saturation entry has three possible values to indicate whether the session is saturated or unsaturated in the link or its status is not known. Every link also stores its link control parameter. Refer to figure 4 for an illustration of the storage fields.

Now we describe the implementation briefly. Initially, all sessions and virtual sessions are assumed to be unsaturated. Receivers send backward rate packets towards the respective source with large rate values. Links compute their link control parameters. Nodes modify the rate values in the backward rate packets, as they travel towards the source. The rate value in a backward rate packet is decreased to the minimum of the link control parameter and the current value in the rate packet. The nodes record these modified rate values as the session link rates. After this modification, nodes merge the backward rate packets of the session in the downstream links originating from the node and in the path of the session. The merged backward rate packet is transmitted towards the source.

After the backward rate packet reaches the session source, the session generates a forward rate packet with the same rate value as the incoming backward rate packet, and transmits it downstream. Once a forward rate packet reaches a node, it updates the session link rates, in the links originating from the node and sets the rate bits in the session fields in each of its outgoing links on the path of the session. This modified value is the current iteration session link rate of the basic algorithm. The rate bits are set to indicate that the current iteration session link rates are known. The node then multicasts the forward rate packet in each of its outgoing links on the path of the session, with a rate value equal to that in the modified session link rate field of the session. When a receiver receives a forward rate packet, it records its rate and sends a probe packet towards the source to query its saturation status.

The probe packet contains the following entries: (a) status and (b) test bit. The status entry has three possible values: "saturation," "unsaturation" or "dont know." If the status entry indicates saturation, then all downstream receivers are saturated. If the status entry indicates "unsaturation", then at least one downstream receiver is unsaturated and hence the session is unsaturated in the link. If the status entry indicates "dont know," then the saturation status of some downstream receiver is not known, and all downstream receivers whose saturation status are known, are saturated. If the test bit is set, then the node which receives the probe packet, must test saturation condition. If the test bit is reset, then no such test need be conducted as saturation status of all downstream receivers are known. If the status indicates unsaturated, then one downstream receiver is unsaturated, and hence the session is unsaturated in the link, but the saturation status of all downstream receivers may not be known yet. The test bit becomes useful in this scenario. It indicates whether saturation status of all downstream receivers are known or not.

The receivers send probe packets with test bit set and status indicating "dont know." When a node receives a probe packet, it modifies the parameters and updates the saturation status of the session in the link in accordance with the rules in page 20. A node receives probe packets of a session in each of its outgoing links on the path of the session. It merges all of these into a single probe packet. The parameters of the new probe packet are determined as per the discussion in page 20. The new probe packet is sent upstream.

When the source receives the probe packet, it generates a "saturation" or an "unsaturation" message. The nature of the message depends on the contents of the probe packet. When a saturation or unsaturation message arrives at a node, the node updates the saturation status of the session in the links on its path, showing "dont know" values. After this update, saturation status of the session in the links originating from the node reflect the current iteration saturation status of the sessions in the basic algorithm. The node generates a similar message on each of its outgoing links which are on the path of the session. The message is a saturation(unsaturation) message if the saturation status of the session in the link indicates saturation(unsaturation). To be more accurate, the node need not send a saturation(unsaturation) message in a downstream link in which the session was saturated before this iteration. This is because saturation status of a receiver does not change once it is saturated, and every receiver downstream of a link is saturated, if the session is saturated in the link. Similarly, a node does not send a forward rate packet in a link where the saturation status of the session indicates saturation. Also, a node does not wait for a probe packet from such a link.

If a receiver receives an unsaturation message, it sends a new backward rate packet, with a very high rate value. If it receives a saturation message, then it does not send any further rate packet.

Now we describe the implementation precisely. We would use the term *unsaturated link set* of a session in a node to denote a subset of links originating from the node, on the path of the session. The saturation status of the session in any link in the subset should reflect an unsaturation value just prior to the last link control parameter computation.

Receiver Process:

1. Receiver sends a backward rate packet(r_p^b) towards the source, with rate value r_p^b very large (ideally infinity).

2. After receiving a forward rate packet(r_p^f) from the source, the receiver records r_p^f as the receiver rate, and sends a probe packet(**test**, **status**) towards the source, with **Test** $= 1$ and **status = dont know**.

3. Upon receiving a message from the source, the receiver updates its saturation status accordingly. If the message is a saturation message, the receiver does not send any further control information. If the message is an unsaturation message, the receiver restarts the process, i.e., sends a backward rate packet towards the source, with a large rate value.

Source Process:

1. Upon receiving a backward rate packet(r_p^b), the source generates a forward rate packet(r_p^f) with the same rate value ($r_p^f = r_p^b$), and transmits it downstream.

2. Upon receiving a probe packet(**test**, **status**), the source generates a message. The message is a saturation message if the probe packet has its **test** bit reset. Otherwise it is an unsaturation message.

Node Process:

Initially a node sets saturation status of every session record as unsaturated. It also sets the rate bit of every session record(setting a rate bit indicates that session link rate for the current iteration is known).

1. Node computes the link control parameter(η_l) for an outgoing link(l) as per step (3) of the basic algorithm, using the minimum session link rates, session rate values and the session saturation statuses in the session records.

2. Node resets rate bits of unsaturated sessions traversing the link, and updates the saturation status of unsaturated sessions traversing the link to "dont know" value.

3. After receiving a session i backward rate packet in a link l, the node examines the rate bit of session i in link l. If the bit is set[†], the node holds the rate packet. If the bit is reset, the node computes the session link parameter(η_{il}) of session i as $\eta_{il} = \max(\eta_l, \mu_{il})$. Subsequently, it updates the session link rate as the minimum of the session link parameter and the rate value in the rate packet and waits for backward rate packets of the session in other links in the unsaturated link set of the session.

4. After processing backward rate packets of a session in all links in the unsaturated link set of the session, a node generates a backward rate packet, with rate value equal to the maximum of the session link rates for the session in all its outgoing links in the path of the session and sends it upstream(towards the source). Note that while computing the rate value of the new backward rate packet, it is not sufficient to consider the session link rates in only the unsaturated link set of the session, but session link rates in all the links originating from the node and on the session path must be taken into consideration.

5. After receiving a forward rate packet of a session, the session modifies the session link rates in the session record in all links in the unsaturated link set of the session. The new value is equal to the minimum of the old session link rate and the rate value in the forward rate packet. Subsequently, it sets the rate bits in these links (session rate known now). Next it generates a forward rate packet for the session in each of the

[†]If the rate bit is set, then the node is yet to compute the current iteration link control parameter.

links in the unsaturated link set of the session. The rate value in such a forward rate packet is set equal to the session link rate in the session record in the corresponding link.

6. After receiving a probe packet(**test**, **status**), for a session i in link l, a node holds it till rate bits of all other sessions traversing the link are set. Then it modifies the probe packet and the saturation entry for session i in link l as follows. If **test** bit of the probe packet is set, the node tests for saturation. The saturation test is to determine whether the link capacity is fully utilized.

   (a) If the link capacity is fully utilized, then it resets the **test** bit and additionally if the **status** indicates "dont know" then the node updates the **status** to saturation.

   (b) If the link capacity is not fully utilized, but the session link rate of session i is strictly less than the maximum of the session link rates for session i in all links originating from the node and on session i path, it resets the **test** bit and additionally if the **status** indicates "dont know," the node updates the **status** to "unsaturation".

   If the **status** indicates "saturation" or "unsaturation," then the **status** is not altered in either of the two cases. If the **test** bit is reset, then the node does not check for saturation and **test** bit and **status** are not altered. The session saturation entry in the link is set equal to the **status** of the probe packet.

7. After processing probe packets of a session in all the links in the unsaturated link set of the session, a node generates a probe packet(**test**, **status**) and sends it towards the source of the session. If the test bit of at least one of the incoming probe packets remains set after modification, then **test** bit of the new probe packet is set. Otherwise, the **test** bit is reset. If

   (a) status entries of at least one incoming probe packet of the session indicates unsaturation after modification or

   (b) status entry of at least one incoming probe packet of the session indicates "dont know" and the session link rate in the respective link is strictly less than the maximum of the session link rates for links on the session path and originating from the node,

   then the **status** entry of the new probe packet is set "unsaturated." Else, if at least one of the incoming probe packets of the session indicate "dont know" after modification, then the **status** entry of the new probe packet is set “dont know”. Otherwise, the **status** entry is set “saturated.”

8. After receiving a message from the source of a session, the node updates the saturation statuses of the session in the links on the path of the session, accordingly. A saturation status is altered only when it reflects a “dont know” value. If it is altered, then it is set equal to the content of the message, which may indicate either saturation or unsaturation. After the update, the node generates a message in each of the links in the unsaturated link set of the session. The message reflects saturation, if the saturation status of the session in the link indicates saturation, and reflects unsaturation otherwise. Once the saturation statuses of all the sessions in a link reflect either saturation or unsaturation, and at least one saturation status reflects unsaturation, the node revokes link control parameter computation for the link, as per step 1.

We have illustrated the operation of this implementation in figures 5 to 9. If a session is saturated in a link in an iteration of the basic algorithm, its saturation entry in the record indicates saturation in the corresponding iteration of the implementation and if it is not saturated then its saturation entry indicates so. Also session link rates of the basic algorithm equal those in the session fields in the links at the end of every iteration. Thus, this implementation exactly emulates the basic algorithm. Note that here the nodes maintain information about session link rates and session saturation status only. It does not require explicit information about all virtual sessions which traverse the node. Besides, there is only one forward rate packet, backward rate packet, probe packet and saturation (unsaturation) message for a session in a link in one iteration, independent of the number of receivers downstream. This merger of control information prevents control information implosion and makes this computation scalable. One can also think of other modifications which should reduce the computation time in most cases. For example, a saturation (unsaturation) message can be generated immediately after the probe packet status reflects saturation (unsaturation), instead of waiting for the probe packet to reach the source. This will not affect the worst case computation time, but should speed up the computations in most cases.

The distributed implementation terminates in $2DM$ units of time where D is the maximum round trip delay 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 1 indicates, any node needs to perform at most M iterations. Note that the termination time does not explicitly depend on the number of links, whereas the complexity of the basic distributed algorithm is $O(\mathcal{L}M^2)$. However, the round trip delay D depends on the number of links in the path of a session. The convergence time in this case should depend on the number of links in the path of a session, and not on the total number of links traversed by sessions, as many of the link computations can be performed in parallel. The round trip delay combines the propagation and processing delays.

An advantage of this algorithm is that the intermediate rates are always feasible[26]. So during the computation period, sources can still transmit at intermediate rates, without any huge queue buildup. As we mentioned earlier, the algorithm is scalable. Also, a receiver knows when it is saturated, and does not exchange any message after it attains saturation. Also a node does not make any computation for a link, if all the sessions traversing the link are saturated. This reduces the message overhead and use of computation resources significantly. On the downside, the worst case convergence time for this algorithm is $O(DM)$. This can become large for a large network. However, as our experimental results indicate, in practice convergence is much faster. This computation requires all sessions to start at the same time and every session is considered unsaturated in every link in its path at start. This means that computation must restart, if there is any change in the network, i.e., if a new receiver joins a session, or an existing receiver leaves a session, or if there is any change in the available capacity of the link. So this algorithm is not appropriate for a dynamic network with short lived sessions. However, it operates very well if sessions have a long life, and session membership changes less frequently.

3.3 Experimental Results

We evaluated the performance of the scheme presented in Section 3.2 by simulation. Our aim was to test the algorithm in a dynamic network with session membership changes. Simulation results indicate that the convergence time is much less than the analytical worst case bound of $O(DM)$. Simulation results also corroborate the fact that message exchange overhead is indeed low for this algorithm. We were not able to compare this algorithm with any previous one, because no other distributed algorithm has been reported for multirate, multicast networks.

We considered a 15 session 400 node random network for experimental evaluation. Nodes are points on a $20X20$ grid. There exists an edge between any two nodes with a probability (p) that depend on the euclidean distance between the nodes (d) $(p = \exp(\alpha(1-d)))$, where α is the decay constant. We assumed $\alpha = 2$. We adopted this edge probability model because distant nodes are less likely to have an edge between them. Source and receivers of every session have been selected randomly. The session route consists of shortest paths between the source and the destinations. Propagation delay in a link is equal to its length (length of a link is the euclidean distance between its end nodes). Maximum round trip delay in this network is 45.54 units. This delay includes the propagation delay only, and not the processing delay. We assume that the node processors are fast, and the processing delays are negligible. We have assumed every link to be bidirectional, and the propagation delay in a link is equal for both directions of propagation. Receivers join and leave the sessions randomly, as per poisson random processes. We assume that receivers do not have any minimum rate requirement.

First we examine the convergence time of the algorithm in this dynamic network. The computation restarts whenever there is a session membership change. Computed rate for a user may not converge to its maxmin fair rate before the next session membership change. This occurs when the change events are very frequent and the computation does not have time to converge to the maxmin fair allocation between the changes. So we studied the discrepancy from the maxmin fair rate at any time. If maxmin fair rate of a receiver s at time t is $r_s^m(t)$, and the computed rate at time t is $r_s^c(t)$, then relative computation error for receiver s is $(1 - \frac{r_s^c(t)}{r_s^m(t)})$ at time t. Figure 10 plots a moving average of the maximum of the relative errors of the receivers and the average of the relative errors of the receivers. The maximum and average are taken over all currently active receivers, and the moving average is computed with a window of 5000 time units. The algorithm restarts and the relative computation error increases to 1 every time there is a session membership change. Subsequently, relative computation error decreases steadily till the next change event. Hence the moving average of the errors over the large time window we have selected does not drop to zero, and the moving average exhibits sharp rise over the periods with more frequent session membership changes. However, average errors are quite small in general, and always less than 0.2. Thus, despite the dynamism, computed rate of an average receiver is above 80% of its maxmin fair rate. Maximum relative error is somewhat higher, indicating that convergence is slower for a few receivers. Figure 11(a) plots percentage of receivers which attain the maxmin fair rate before the next change versus mean time between changes. As expected, this percentage is low for a high frequency of membership change, and increases with decrease in membership change frequency. However, on an average, around 80% receivers converge to their maxmin fair rates, when average time between changes equal 500 time units. Figure 11(b) presents the curve of average convergence time for the receivers which converge to their maxmin fair rates, in between two membership change event versus mean time between changes. This convergence time is very low for high membership change frequency, because when session membership changes rapidly, only the receivers who need little time to converge to the maxmin fair rates, actually converge to the maxmin fair rates, and others do not converge. As membership change frequency decreases, most of the receivers converge between change events, and the convergence time represents the convergence time of an average receiver in the network. When mean time between changes equal 2259.34 unit, around 95.6% receivers converge to their maxmin fair rates between changes, on an average. Mean convergence time of the converging receivers in this case is 102.81 units. This represents convergence time of an average receiver, because the percentage of converging ones is quite high. On an average, around 47 receivers are members of different sessions at any time. Thus worst case convergence time of $2DM$ units is 4280.76. Similarly, convergence time is 155.89 units in the same random network with 96 receivers. The worst case bound is 8743.68 in this case. The convergence time is normally a lot better than the analytical bound of $2DM$ units.

Now we shall examine the message exchange overhead in this network. We look at the number of control message bits traversing a link per unit time. The control message comprises of rate packets, probe packets and saturation/unsaturation messages. We plot the maximum of this overhead per unit time against the mean time between membership changes, in figure 11(c). The maximum is taken over all links. This maximum is always less than 10 bits per unit time, and decreases further with decrease in frequency of session membership change. This is because as session membership changes less frequently, larger number of receivers saturate in between the changes, and a receiver stops exchanging messages after it saturates. So, for a low frequency of membership change, receivers do not send control messages for long periods of times between the changes. Figure 11(c) indicates that message exchange complexity is indeed low for this algorithm.

4 Allocation of Rates

In this section, we 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. Video gateways are placed throughout the network to transcode signal into a lower bit rate coding, using either the same coding format with different parameters or a different coding scheme altogether[29]. 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 the 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[29]. For instance, the maxmin fair rates of virtual sessions $1, 2, 3$ are $4, 3.5, 3$ respectively in figure 2. Source of session 1 transmits at rate 4. The video gateway at intermediate node I transcodes session 1 signal to rate 3.5 for transmission through e_3 and further downstream to u_2. Session 1 signal is transmitted at rate 4 units through e_2 and e_4, though. In this way, heterogeneity is locally accommodated to fine tune the transmission rate to exactly match the fair share of link bandwidth allocated to receivers downstream. Rate adaptive video gateways [2] can be used for this purpose. Rate control is attained by employing more aggressive quantizers and by output frame rate control (dropping frames as necessary from the input stream to meet a given rate constraint). The latter attains fine tuning of the output bit rate. Active network architecture[31] provides a framework for deployment of rate adaptive video gateways within the network.

A second solution to the problem of fair allocation of rates is multirate transmission using hierarchical encoding. In this approach, an information stream is partioned in to a base layer, comprising the information needed to represent the lowest fidelity media and a number of enhancement layers. Some number of these enhancement layers are combined by the decoder with the base layer to recover a signal of incremental fidelity. An important characteristic of hierarchical encoding is that the layers are generally ordered in some manner. An enhancement layer yields useful information only when the base layer and the lower layers have been successfully decoded. As each layer is added, there is an improvement of quality of the received signal and additional bandwidth is needed to transport the combined stream. Hierarchical coding was first suggested for packet voice transmission[6] and subsequently several coding schemes have been proposed for video, e.g., [1], [15], [32], to name a few. Some of these are amenable to tuning of layer bandwidths. Tuning of layer bandwidths become trivial if an *embedded* code is used. In an embedded code, any prefix of a valid codeword is a valid codeword and if the code is chosen appropriately, the prefix codeword corresponds to a lower quality version of the longer codeword. Hence one can trivially form a layered code from an embedded code by breaking up the longer code at arbitrary boundaries. Moreover, one can generate as many layers as desired and tune the rates of the different layers since the embedded code can be broken at arbitrary points. A low complexity video codec is presented in [21]. It uses PVH(progressive video with hybrid transform) code, an instance of an embedded code and is thus amenable to dynamic rate control. Embedded code has also been used in speech coding[6]. A pyramidal video coding scheme with two layers and a quality parameter to control the bit rate of the enhancement layer has been suggested in [7]. This idea may be extended to multilayer coding with flexible layer bandwidths. Tuning of layer bandwidths can be incorporated in many other codecs by using adaptive quantizers instead of fixed ones. If the source coder is programmable, and the source can modify its signals hierarchical structure, as in the cases discussed above, 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. Total number of layers transmitted across a link is equal to the maximum number of layers allotted to the session receivers downstream. Layers are dropped or packets are replicated at forking points selectively. For instance, the source of session 1 in Example 3.1.1 (Figure 2) generates a code with 2 layers. The base layer will have bandwidth equal to 3.5 units, and the enhancement layer will have bandwidth equal to .5 unit. Both layers are transmitted across e_1, e_2, e_4. Only the base layer is transmitted across e_3, e_5. Only base layer packets are replicated at intermediate node I, for transmission across e_2 and e_3. The enhancement layer packets are not replicated at I as they are transmitted only across e_2. The source in Figure 1 will have a $4-$layer encoding. Receiver u_1 receives the first two layers, u_2 receives all 4 of them, u_3 receives the first 3 layers and the slowest receiver u_4, receives only the base layer. We would like to mention that layered coding is useful, if the nodes of the multicast tree are able to selectively forward flows to a specific branch. This mechanism is likely to be deployed in networks eventually. If network nodes do not have this ability, then transcoding is the only option.

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. Although PVH codec of [21] is amenable to dynamic layer bandwidth adaptation, the implementations still have a fixed layout strategy. Besides, many hierarchical coding schemes do not generate embedded codes, e.g., perceptually weighted wavelets using hierarchical vector quantization with wavelet decomposition (WWHVQ)[34]. In certain video codes, substreams can be extracted to produce a specific range of resolutions only, e.g., 3D Subband Video Coding[32]. Certain coding schemes are particularly successful only when some apriori structure or hierarchy can be found in the problem[33]. In many cases, generating a layer requires a dedicated filter, and the number of filters employed by the source is fixed. Thus the number of layers is limited. Similarly, requirements for minimum packet size and limitations on packetization delay, may impose a lower bound on bandwidth assigned to a single layer and thus adversely affect layer granularity. 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 instance, if the source of session 1 in Example 3.1.1 (Figure 2) can transmit unit bandwidth layers only, then initially receiver u_1 will be allotted 4 layers and receiver u_2 3 layers only. Receiver u_2 will be allotted one more layer in the second stage (4 layers in all), because its fair rate is 3.5 units and 3 layers consume 3 units of bandwidth. Intermediate node I should not replicate 50% of packets of layer 4. All layer 4 packets are transmitted across e_2 and only 50% of them are transmitted across e_3. So 50% of layer 4 packets should not be replicated at node I. Selective replication or selective dropping can be incorporated by using fair queueing for example, with different weights for different layers[8]. The required weights can be computed easily with the knowledge of layer bandwidths and the maxmin fair session link rates. For certain coding schemes, packet loss causes a graceful degradation in signal quality. For example, in [15] signal exhibits a reasonable enhancement in quality above the base layer for 10% enhancement layer packet loss. This transition is gradual up until 100% enhancement layer packet loss. This mechanism will be useful in these coding schemes.

We would like to comment on our assumption that all multicast packets of the same session move along the same tree. Different multicast layers are perceived as different multicast groups and trees for different groups may be completely different in general. However, trees for different multicast layers of the same session will remain the same if source rooted trees are used, as all these layers (multicast groups) have the same source. Besides, trees for different multicast layers of the same session should not differ very much, as that would complicate reconstruction of information at the receiver. For example, if we consider video transmission, and different layer packets of the same session traverse along different multicast trees, then different layer packets for the same frame may arrive at a receiver at different times, and frame reconstruction will involve a lot of packet reordering. This may incur an unacceptable delay jitter. Thus it may be a good idea to use source rooted trees in this case. Many video coders make the same assumption, e.g., [7].

5 Conclusion

Summarizing, we have presented a quantitative framework that can model the fair allocation of bandwidth in the multicast scenario, while taking care of both fairness among members of the same and different sessions. 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 or selective packet replication. 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. There exists an array of related intellectually challenging unsolved research problems.

Firstly, we have assumed that network has certain provisions, but that may not be the case. For example, transcoding places additional computational and administrative burden on the network. It may be possible to deploy transcoding gateways at strategic locations in the network but a system that requires transcoding at arbitrary bottleneck links may not be computationally and financially viable. For transcoding, internal nodes must perform intensive computational tasks and may need to process arbitrary coding algorithms. Transcoding process increases end to end delay and cannot be applied to a secure communication without entrusting the network with the encryption key. Depending on the security risks, this may be totally unacceptable. In these cases, network heterogeneity can be countered through hierarchical encoding only. As discussed before, hierarchical signal structure may be predetermined and layer granularity may be lower bounded. Network may not have selective replication provisions at congestion points. As a consequence, congestion will affect all the layers and the signal quality will be very poor. Besides trusting the network to drop a certain portion of the highest layer packets may produce perceptually annoying distortions in certain coding schemes. If the highest layer, employs a differential coding scheme, then even a small percentage loss of packets may garble the entire information in the final layer. In all these cases, "partial" subscription to a layer is useless and receivers can only subscribe to layers fully. 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 [28].

The current distributed algorithm needs to synchronize the starting time for the sessions, and needs to restart every time network topology changes. This is not an issue when the network has long lived sessions, but this is a disadvantage for a network with short lived sessions. We have presented another distributed algorithm for computation of maxmin fair rates in a multirate multicast network in [27], which is more suited to asynchronous operation, i.e., when sessions enter and exit at different times. The tradeoff is that the asynchronous algorithm in [27] consumes higher control information exchange overhead and has greater worst case convergence time bound. We think that these algorithms are suited for different scenarios.

We have discussed the generic mechanisms for fair rate allocation in presence of certain provisions like transcoding options and/or adaptive hierarchical encoding in Section 4, but we did not describe the exact protocol format for the purpose. The detailed protocol would depend on the particular choice (transcoding or adaptive hierarchical encoding). Developing the detailed protocols for the provisions described in Section 4 would be interesting from an implementation point of view. We have also not described the exact message exchange format for distributed computation of the maxmin fair rates, but have suggested the general framework. Developing the exact protocol format will have implementational significance. Besides, while describing the implementation frameworks, 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 investigation.

A Proof of Theorem 1

We present a sketch of the proof in this section. Refer to technical report [26] for details.

Lemma 3 *If the algorithm does not terminate in k iterations, then $S(k)$ is a proper subset of $S(k-1)$, for all iterations k.*

Remark: This lemma states that the set of unsaturated virtual sessions decrease steadily in every iteration. We will use this lemma to show that the basic algorithm terminates in at most M iterations, where M is the total number of virtual sessions. The proof shows that a new virtual session saturates every iteration. It looks at the link $l_{\min}$ which attains the minimum link control parameter, amongst all those which carry at least one unsaturated virtual session. It looks at that unsaturated session $i_{\min}$ which has the minimum rate requirement (minimum μ_{il}) amongst all the unsaturated sessions traversing through link $l_{\min}$ and argues that all unsaturated virtual sessions of $i_{\min}$ saturate in the current iteration. The result follows.

Lemma 4 *The rate allocation at the end of the kth iteration is feasible, $k \geq 0$.*

Remark: This lemma will be used in the proof of Theorem 1 to show that the output of the basic algorithm is feasible. Feasibility is an essential requirement for maxmin fairness of any rate allocation. Intuitive reasoning for the lemma is as follows. Rate of a virtual session at the 0th iteration is its minimum rate. Rate of a virtual session do not decrease in subsequent iterations. Thus minimum rate constraints are satisfied. Capacity constraints hold because no unsaturated session is offered a rate more than its session link parameter in any link. Session link parameters are computed such that the sum of the session link parameters for unsaturated sessions and bandwidths for saturated sessions equal the link capacity for every link.

Lemma 5 *Let $s \in S(k-1) \setminus S(k)$. Let $t \geq k$ and $S(t-1) \neq \phi$, i.e., the algorithm does not terminate in $t-1$ iterations. Then virtual session s has a bottleneck link, under rate vector $\vec{r}(t)$.*

Remark: We will use this lemma in the proof for Theorem 1 to argue that the output of the basic algorithm is maxmin fair. Intuitively the lemma can be argued as follows. Ignore the minimum rate constraints for time being. If a virtual session s saturates, then it traverses through a link l whose capacity is fully utilized. Capacity of the same link remains utilized in all subsequent iterations because session rates do not decrease. Thus the first bottleneck condition holds in all iterations. Actually, rates of sessions traversing the link do not change in subsequent iterations, because they can not increase in view of the full utilization of the capacity, and they can not decrease in any event. The rate of the particular virtual session concerned does not change any further. Thus, since the second bottleneck condition holds in the iteration in which the virtual session saturate (by condition for saturation), it holds in all subsequent iterations. We argue the third condition as follows. It can be shown that if s saturates in the kth iteration, then its rate is equal to the link control parameter of l in the kth iteration, and rate of any other virtual session traversing l in any subsequent iteration is upper bounded by the link control parameter of l in the kth iteration. Thus the rate of any other virtual session traversing l does not exceed that of s in any subsequent iteration. Thus the third bottleneck condition holds. The formal proof argues the lemma in a more general case, with the minimum rate constraints[26].

Proof of Theorem 1: The fact that the basic algorithm terminates in at most M iterations follows from Lemma 3. This is because by Lemma 3, $|S(k)| \leq |S(k-1)| - 1$, for all iterations k. Note that $|S(0)| = M$. Hence, $S(k) = \phi$, for some $k \leq M$.

If the algorithm terminates in k iterations, $S(k) = \phi$. Since $S(0)$ is the set of all virtual sessions, for every virtual session s, there exists a $t \leq k$, such that $s \in S(t-1) \setminus S(t)$, $t \leq k$. Thus by Lemma 5 virtual session s has a bottleneck link $l \in L_s$, under rate vector $\vec{r}(k)$. Rate vector $\vec{r}(k)$ is feasible by Lemma 4. Thus $\vec{r}(k)$ is max-min fair by Lemma 1. □

References

[1] E. Amir, S. McCanne and M. Vetterli. A layered DCT coder for Internet video. *Proceedings of IEEE International Conference on Image Processing,* pages $13 - 16$, Lusanne, Switzerland, September, 1996.

[2] 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.

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

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

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

[6] 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. $325 - 333$

[7] T. Brown, S. Sazzad, C. Schoeder, P. Cantrell and J. Gibson. Packet Video For Heterogeneous Network Using CU-Seeme, *Proceedings of IEEE International Conference on Image Processing,* Vol. 1, September, 1996, pp. $9 - 12$.

[8] J. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms *Proceedings of ACM SIGCOMM'* 96, *pp* $143 - 156$, Palo Alto, CA, August $'96$.

[9] S. Chiung, M. Ammar and X. Li. On the use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution *IEEE INFOCOM'*96

[10] F. Chiussi, Y. Xia and V. Kumar. Performance of Shared-Memory Switches under Multicast Bursty Traffic *IEEE Journal on Selected Areas In Communications*, Vol 15, No. 3, 1997.

[11] 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.

[12] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. An architecture for wide area multicast routing, *Proceedings of ACM SIGCOMM,* London, UK, Aug. 1994, pp. 126-135.

[13] F. Kishino, K. Manabe, Y. Hayashi and H. Yasuda. Variable Bit-Rate Coding of Video Signals for ATM Networks, *IEEE Journal on Selected Areas In Communications*, Vol. 7, No. 5, June 1989.

[14] F. Kelly, A. Maulloo, and D. Tan. "Rate control for communication networks: Shadow prices, proportional fairness and stability *Journal of Operations Research society,* 49(3):$237 - 252$, March 1998.

[15] M. Ghanbari. Two-Layer Coding of Video Signals for VBR Networks. *IEEE Journal on Selected Areas in Communications*, VOL. 7. No. 5. June 1989.

[16] 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*, San Francisco, CA, March 1998

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

[18] 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

[19] S. Low and D. Lapsley. Optimization Flow Control, I: Basic Algorithm and Convergence, *IEEE/ACM Transactions on Networking,* 7(6), Dec. 1999.

[20] J. Moy, Multicast routing extensions for OSPF, *Comm, ACM* vol 37, no. 8, pp 61-66, Aug 94.

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

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

[23] M. Parsa, J.J.Garcia-Luna-Aceves. A protocol for Scalable Loop-Free Multicast Routing, *IEEE Journal on Selected Areas In Communications*, Vol 15, No. 3, 1997.

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

[25] 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

[26] 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.

[27] S. Sarkar and L. Tassiulas: Distributed Algorithms for Computation of Fair Rates in Multirate Multicast Trees *Proceedings of INFOCOM'* 2000

[28] S. Sarkar and L. Tassiulas: Fair Allocation of Discrete Bandwidth Layers in Multicast Networks *Proceedings of INFOCOM'* 2000

[29] T. Turletti and J. C. Bolot Issues with multicast video distribution in heterogeneous packet networks, *Packet Video Workshop*, '94, Portland.

[30] 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.

[31] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *Computer Communication Review,*, $26(2):5-18$, April 1996.

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

[33] K. M. Uz, M. Vetterli and D. J. LeGall. Interpolative Multiresolution Coding of Advanced Television with Compatible Subchannels, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol 1, No. 1, March 1991.

[34] M. Vishwanath and P. Chou. An efficient algorithm for hierarchical compression of video. *Proceedings of IEEE International Conference on Image Processing,* Austin, TX, November, 1994.

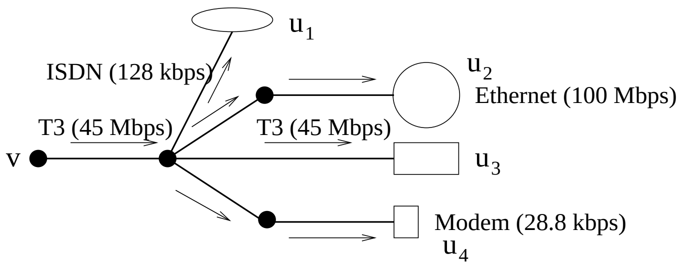

Figure 1: A sample network indicating network heterogeneity

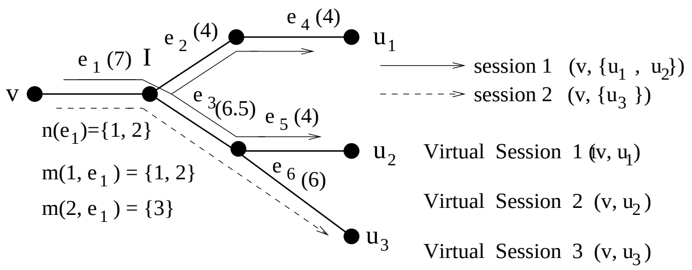

Figure 2: The numbers in brackets, () denote the capacities of the respective links. The capacity constraint for link e_1 is $\max(r_1, r_2) + r_3 \leq 7$ and that for link e_3 is $r_2 + r_3 \leq 6.5$. Thus minimum and maximum rate constraints are $4 \leq r_1 \leq 5$, $1 \leq r_2 \leq \infty$ and $0 \leq r_3 \leq 5$. The maxmin fair rate vector is $(4, 3.5, 3)$. Under the maxmin fair rate allocation, link e_1 is bottlenecked w.r.t. virtual session (v, u_1) and (v, u_3) and 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 another feasible rate vector $(4, 4.5, 2)$. Now no link is bottlenecked w.r.t. virtual session (v, u_3).

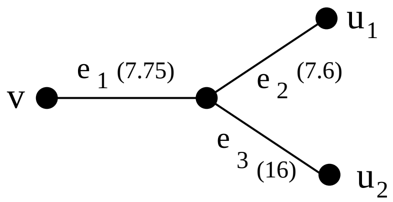

Figure 3: This figure shows that the basic distributed algorithm of Section 3.1 may take more than $\mathcal{L}$ iterations to terminate, where $\mathcal{L}$ is the set of links. There are 4 multicast sessions. Node v is the source and nodes u_1 and u_2 are the destinations for all of them. Thus there are 8 virtual sessions, $1, \ldots, 8$. Virtual session j corresponds to session $\lceil \frac{j}{2} \rceil$. Virtual sessions $1, 3, 5, 7$ have u_2 as destination and traverse edges e_1, e_3. The rest have u_1 as destination and traverse edges e_1, e_2. Node v is the source of all virtual sessions. The minimum rate requirements of virtual sessions $1, 3, 5, 7$ are $1, 2, 2.25, 2.5$ units. The rest of the virtual sessions have 0 as the minimum rate requirement. Here, $C_{e_1} = 7.75$, $C_{e_2} = 7.6$ and $C_{e_3} = 16$ units. The basic distributed algorithm terminates in 4 iterations, though there are 3 links.

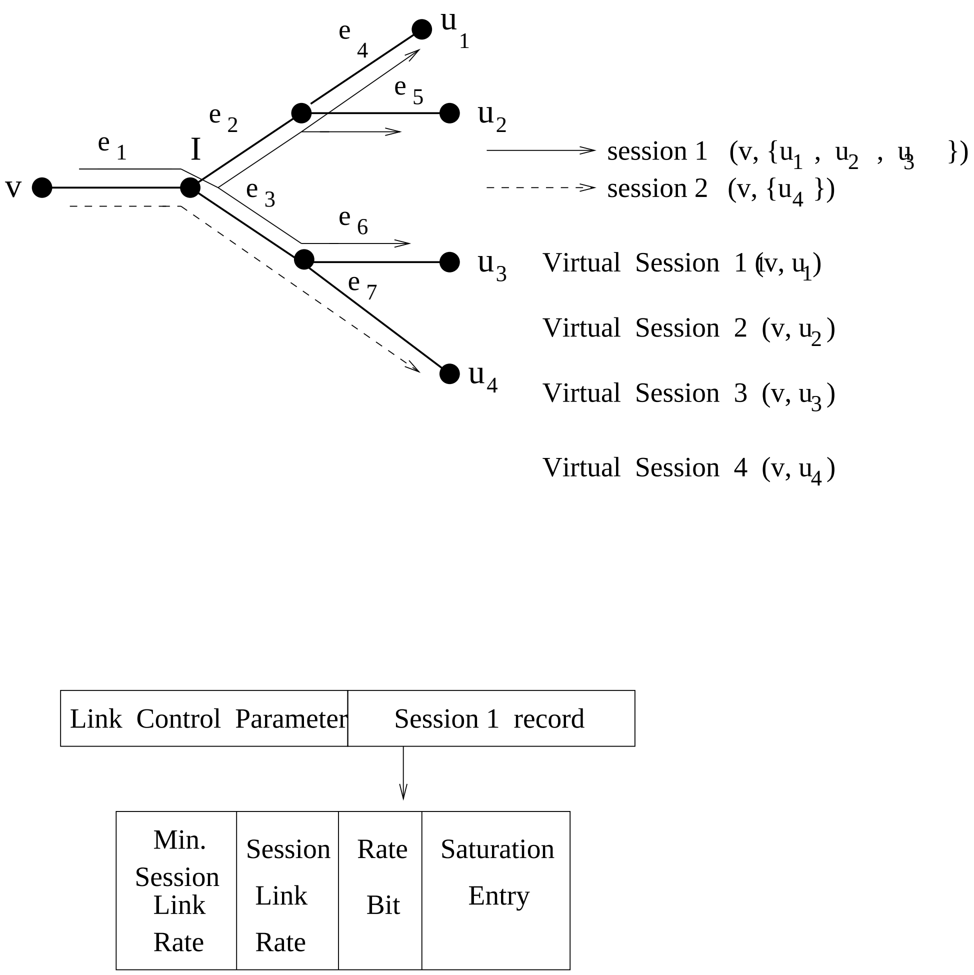

Figure 4: We show the storage at node I for link e_2 for the distributed implementation. Note that the node I does not maintain separate information about virtual sessions 1 and 2 traversing link e_2.

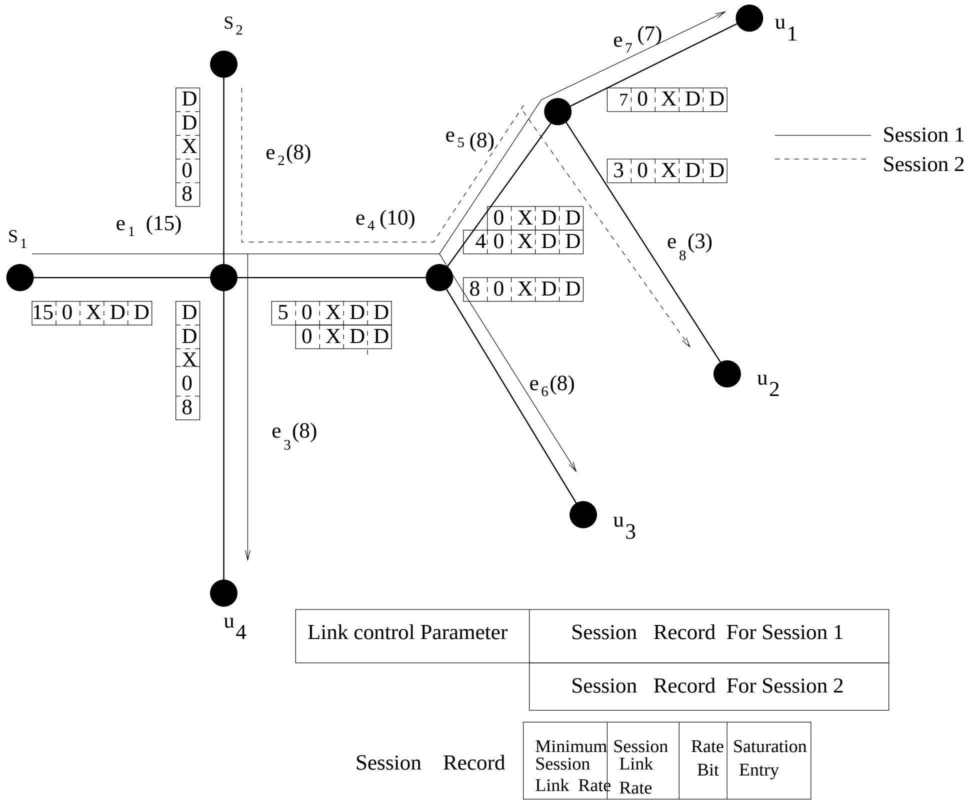

Figure 5: We shall illustrate the operation of the distributed computation of Section 3.2 in this and the next few figures. Here, we show the storages at the nodes, after the first link control parameter computations, and before processing any rate packets. The record against a link shows the storage for the link at the origin node of the link. For example, link control parameter for edge e_5 is 4, and hence the link control parameter entry for the corresponding storage shows 4. None of the receivers have any minimum rate requirement in this case. Session link rates are not known at this point, and hence the corresponding entries indicate “dont care” (X) value. Rate bits also indicate “dont know ” (D) value (equivalently rate bit reset). Also, session saturation states are not known, and hence indicate “dont know” values.

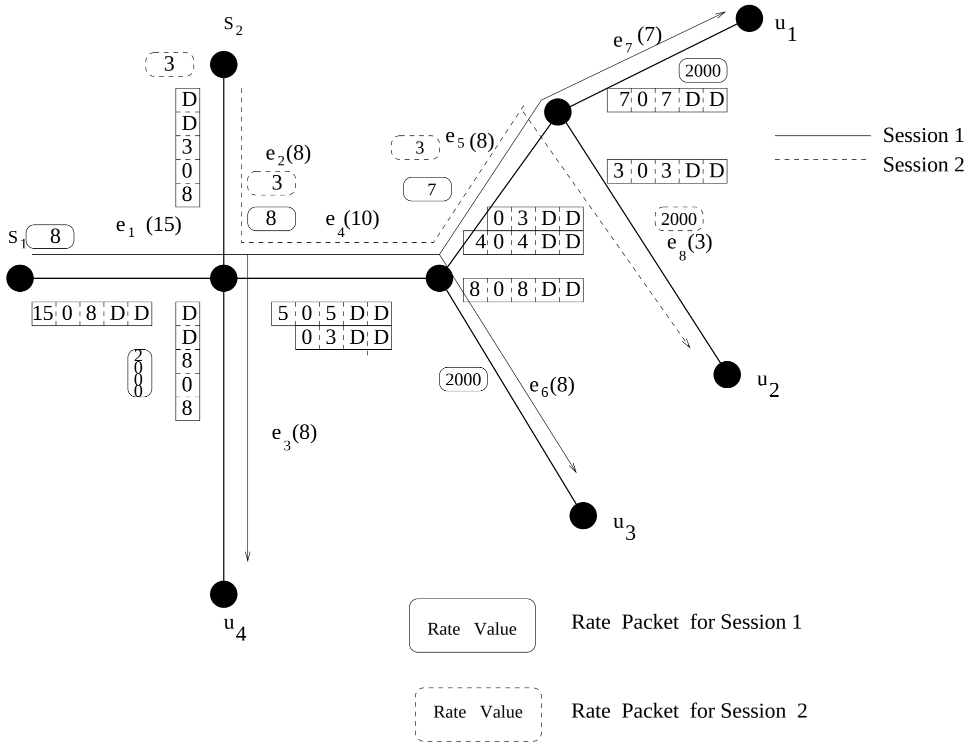

Figure 6: Now we show the storage states after the arrival of backward rate packets in the corresponding nodes. The session link rate values in the session records indicate the updated values after receiving the backward rate packets. We also show the backward rate packets near the links, with the rate value they would have at the time they reach the source of the link. For example, receiver u_1 sends a backward rate packet with a very high value, 2000. So a session 1 backward rate packet reaches the source of e_7 with rate value 2000. The current iteration session link rate values are still not known at this stage. So the session link rate field still indicates 'dont know' value.

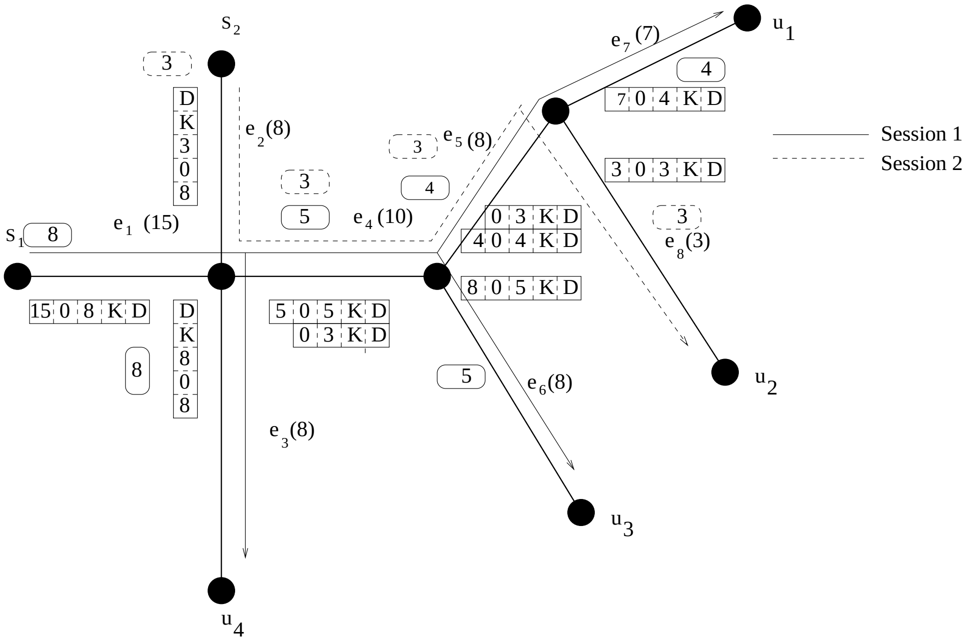

Figure 7: Now we show the storage states after the transmission of forward rate packets from the respective nodes. For example, the storage for link e_5 shows the states after transmitting the forward rate packets for sessions 1 and 2. The session link rates for the current iteration are known at the source nodes of the links, after transmitting the forward rate packets. Hence the rate field indicates "known" (K). Note that the session link rate values in the storage fields are equal to the session link rates computed in the first iteration of the basic algorithm of Section 3.1.

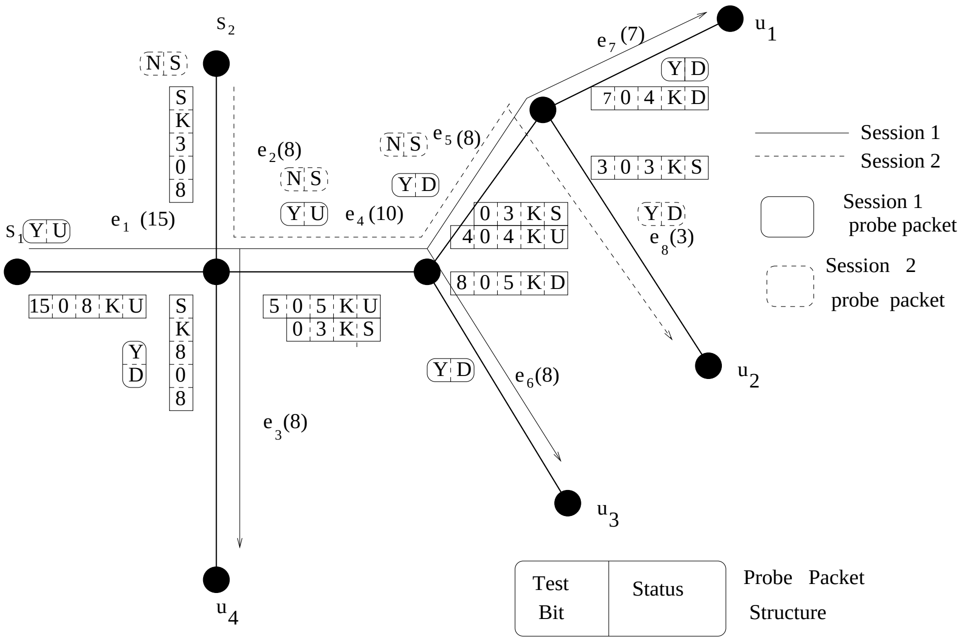

Figure 8: This figure shows the storage records after the probe packets are processed in the respective nodes. We show the contents of the probe packets before the source node of the link processes them. The test bit for a probe packet has two possible values,"Y" and "N" (equivalently set or reset respectively). If test bit is "Y," then the saturation conditions are tested. The status can have three possible values: saturated ("S"), unsaturated("U"), and "dont know" ("D"). Every receiver sends a probe packet with test bit as "Y" and status as "D." The "dont know" status are updated to saturation or unsaturation after the sources generate the saturation/unsaturation messages.

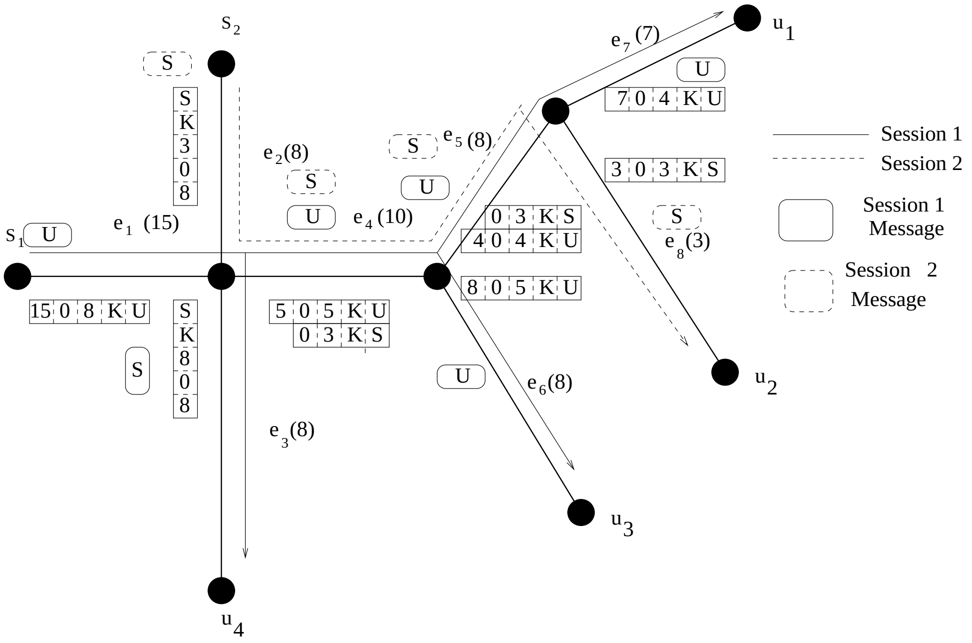

Figure 9: This figure shows the storage records after the nodes receive a message from the session sources. We have also shown the messages traversing the links. A message can have two possible values: saturation ("S") or unsaturation ("U"). Note that a message changes the content of session saturation status, only when the session saturation status reflects a "dont know" value. After processing the messages, a session saturation status represents the saturation state of the respective session in the respective link, at the end of the first iteration of the basic algorithm in Section 3.1.

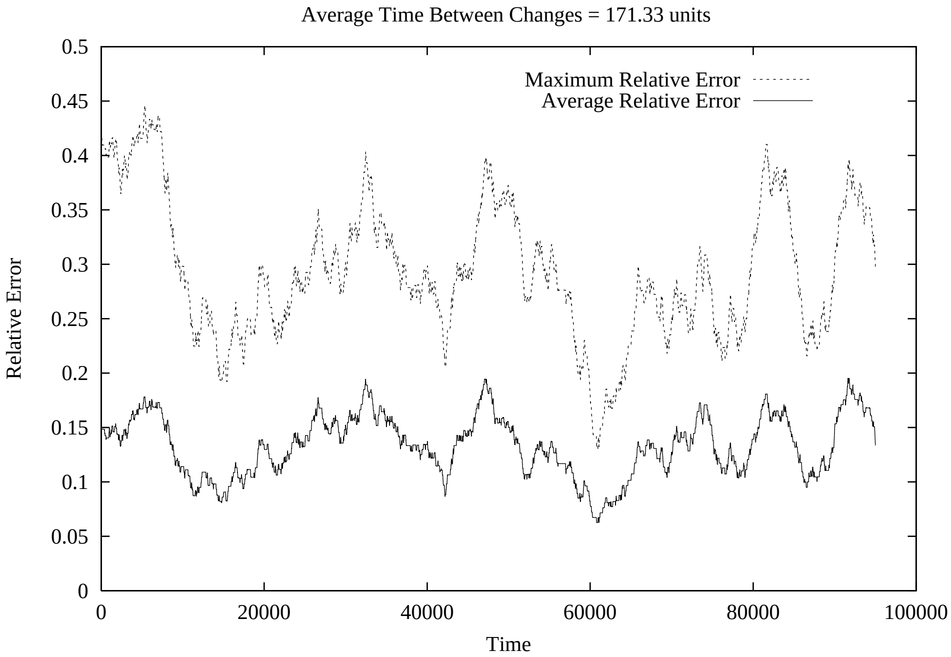

Figure 10: We study the computation error in a dynamic network with frequent session membership changes. If maxmin fair rate of a receiver s at time t is $r_s^m(t)$, and the computed rate at time t is $r_s^c(t)$, then computation error for receiver s is $(1 - \frac{r_s^c(t)}{r_s^m(t)})$ at time t. We plot the moving average of maximum and the average of the computation errors, maximum and average taken over all currently active receivers. Moving average is taken over a window of 5000 units. The average computation error is below 20% in this figure. The maximum computation error is somewhat larger. This indicates that computed rate of an "average" receiver is close to the maxmin fair rate, even in presence of frequent session membership changes. Some receivers converge slowly, making the maximum error higher. Mean time between session membership changes is 171.33 units in this figure.

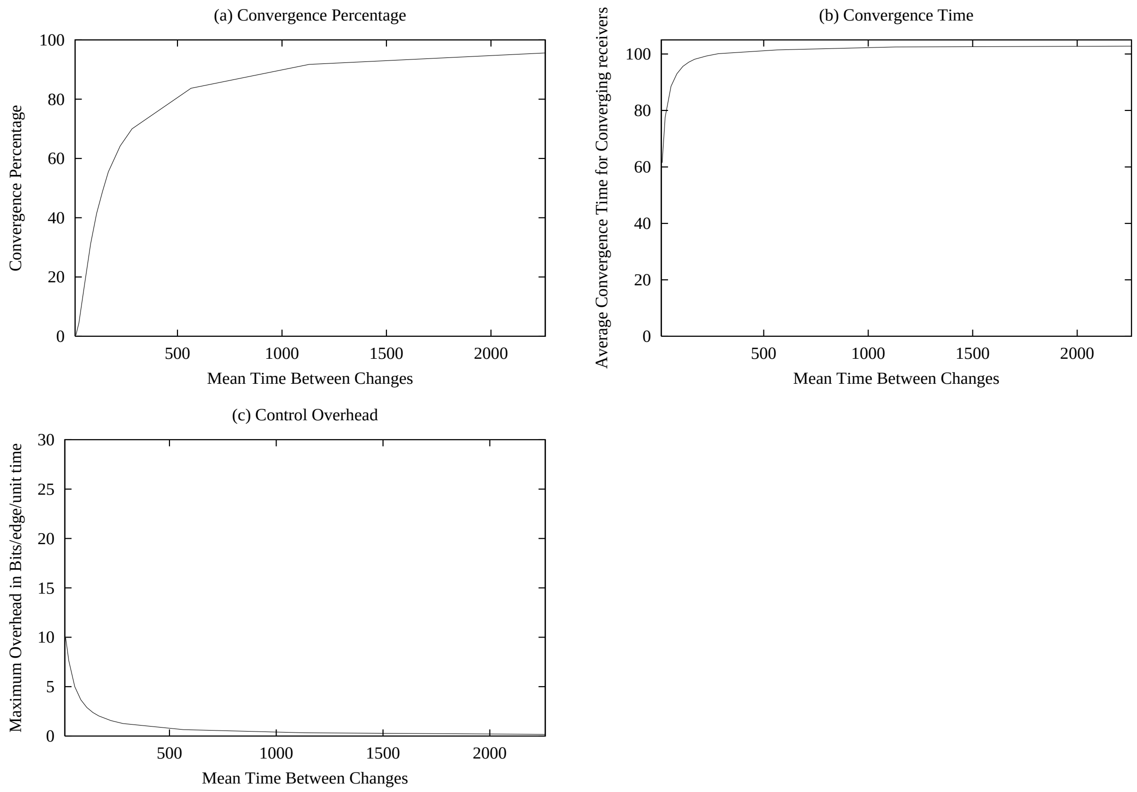

Figure 11: We study convergence speed and control information exchange complexity for a dynamic network with session membership changes. The distributed computation does not always converge in between the changes, on account of frequent session membership changes. Figure (a) plots the average percentage of receivers attaining their respective maxmin fair rates in between the changes. Figure (b) plots the average convergence time of these receivers. Figure (c) studies the control information traversing each link per unit time. It plots the maximum of this control information per unit time, the maximum taken over all links. This control information comprises of rate packets, probe packets and saturation/unsaturation messages.