A Framework for Routing and Congestion Control in Multicast Networks

Saswati Sarkar and *Leandros Tassiulas*
Dept. of Electrical 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 propose a new multicast routing and scheduling algorithm called multipurpose multicast routing and scheduling algorithm (MMRS). The routing policy load balances amongst various possible routes between the source and the destinations, basing its decisions on the message queue lengths at the source node. The scheduling amongst various sessions sharing links is devised such that the flow of a session depends on the congestion of the next hop links. MMRS is throughput optimal and computationally simple. It can be implemented in a distributed, asynchronous manner. It has several parameters which can be suitably modified to control the end to end delay, packet loss in a topology specific manner. These parameters can be adjusted to offer limited priorities to some desired sessions. MMRS is expected to play a significant role in end to end congestion control in the multicast scenario.**

I. Introduction

Multicasting provides an efficient way of transmitting data from a sender to a group of receivers. A single source node or a group of source nodes sends identical messages simultaneously to multiple destination nodes. Single destination or unicast and broadcast to the entire network are special cases of multicast. Multicast applications include collaborative applications like audio or video teleconferencing, video-on-demand services, distributed databases, distribution of software, financial information, electronic newspapers, billing records, medical images, weather maps and experimental data, distributed interactive simulation (DIS) activities such as tank battle simulations. Many distributed systems such as the V System[3] and the Andrew distributed computing environment[21], popular protocol suites like Sun's broadcast RPC service[18] and IBMs NetBIOS[11] are using multicasting. 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[4].

There may be more than one possible route between a source and a group of destinations. More than one multicast session may share the same link. This gives rise to the fundamental issues of routing and scheduling in multicast communications. Until now scheduling in multicast networks has primarily been best effort service. As traffic increases congestion control, class based scheduling would be required to improve performance. MMRS uses a scheduling policy which can be tuned to distinguish amongst various classes.

Significant amount of research has been directed towards multicast routing. Tree construction is the commonly used approach in solving the multicast routing problem. Multicast trees minimize data replication; messages need only to be replicated at forking nodes. Multicast trees can be broadly classified into shortest-path trees (SPT's), also known as source based trees and group shared trees. SPT is currently used in distance-vector multicast routing protocol (DVMRP)[5] for Internet multicast traffic on the virtual multicast backbone (Mbone) networkand multicast extensions for open shortest path first OSPF(MOSPF)[15]. The core-based tree(CBT)[1] uses a group shared tree also known as the center based tree. Recently some hybrid routing protocols like the protocol independent multicast (PIM)[6] and the multicast internet protocol MIP[16] have been proposed. These allow the system to switch modes between SPT and group shared trees.

However none of these routing policies support more than one tree per source-destination pair at a time. Thus only a single route is determined depending upon the topology and then the messages are sent along the same route till the topology or the destination group changes. The existing protocols do not provide for *load balancing* i.e., having more than one possible tree simultaneously and allowing the system to dynamically choose amongst them, the routing decisions being taken not too infrequently. However load balancing can meet very effectively the technical challenge of minimizing the link loads given some network load and thus serve as an important weapon for congestion control. This would increase throughput, decrease delay and message loss in the network. Congestion control is critically important in various resource expensive applications.

Load balancing may cause out of order delivery of messages. However some applications do not need ordered delivery of messages, e.g., many audio and video conferencing applications like vic[13], vat[27], nv[9], rat[10] and freephone[2] (audio packets are reordered in application play out buffer), workspace applications like Wb[29]. Application level protocols can be used to enforce a particular delivery order, if necessary. However ATM applications need ordered delivery. So load balancing can be done per session. Besides out of order delivery can be reduced by choosing large routing decision intervals, depending upon the requirement of the application. Load balancing would increase the routing table entries in the routers because more than one routing tree may be used simultaneously; however the network can choose the number of simultaneously active trees depending upon the router memories and a tradeoff can be reached.

However to do load balancing effectively the network needs to route a message through a tree selected judiciously amongst many possible trees. Choice of the least total cost tree amongst all possible trees between a source and a group of destinations is the usual approach. The tree cost is usually the sum of the link costs and the link costs may be a measure of a number of possible parameters, e.g., actual or anticipated congestion, error rate, propagation delay etc. So computation of a good or rather near optimal route based on the total tree cost at reasonably regular intervals is computationally very expensive if the set of possible trees is even a moderately large subset of all possible trees between a source and the destinations.

We propose a novel routing and scheduling policy which retains the benefits of load balancing, yet overcomes the above difficulty. We call this policy the *Multipurpose Multicast Routing and Scheduling* policy (MMRS). The routing scheme takes routing decisions at possibly random, but bounded intervals and base the decision *only* on the message queue lengths of the diffrent possible trees at the source node. It takes routing decisions in favor of the tree with least queue length or rather a least scaled queue length at the source node. The scheduling policy has been so chosen that this quantity represents the congestion state of the tree. At any link the scheduling policy gives priority to sessions which have the congestion[1] at downstream[2] buffers less than that at the upstream buffer and does not serve any session at a link if all the sessions have the downstream buffers more congested than the corresponding upstream buffer at the link. This spreads the congestion to the upstream nodes. Thus if a tree is congested then after some time the congestion would be reflected in the source node, more precisely the source node will soon have a large message queue for the tree and the routing policy would route message through other trees with less message queue length at the source node and hence better congestion state throughout. This attains load balancing without any intensive computation based on the state of the entire tree.

MMRS has various parameters. If the parameters are properly chosen the scheduling policy becomes computationally simple and needs only local information[3] and not the state of the entire network. The parameters can be further adjusted to obtain various other desirable features. We discuss these later.

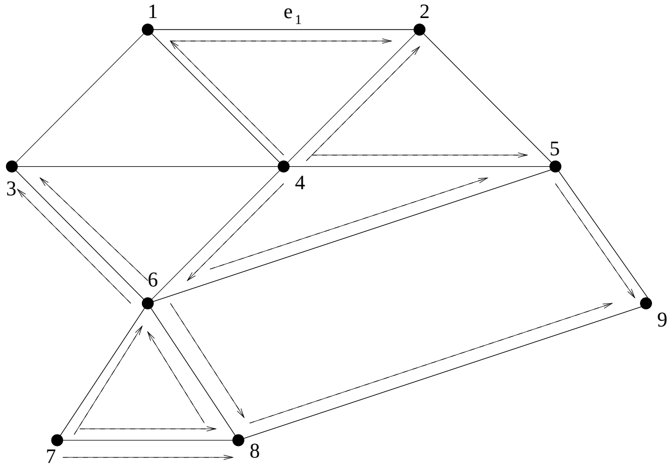

Fig. 1. A directed graph representing a network.

Under some statistical assumptions on the message arrival and service process MMRS attains maximum possible throughput. We believe that this *maximum throughput* property of MMRS does not depend on the assumptions and holds for much more general arrival and service processes. MMRS retains this throughput optimality even if the routing and the scheduling decisions are not taken every slot but at bounded intervals.

The rest of the paper is organized as follows. We describe the multicast network model in Section II. Section III describes the general routing, scheduling and congestion control problem in multicast networks. Section IV describes MMRS in detail. We discuss some interesting aspects of MMRS in Section V. We describe our stability criterion for any network in Section VI and outline the proof of the maximum throughput property of MMRS in Section VII.

II. Multicast transport network model

The network is modelled by an arbitrary topology directed graph $G = (V, E)$. V represents the set of nodes and E the set of directed links. There exists an edge directed from u_1 to u_2 in G iff there exists a directed link between the corresponding nodes in the network. A multicast session is identified by the pair (v, U), where v is the source node of the session and U is the group of intended destination nodes. There are N multicast sessions $(v_1, U_1), \dots, (v_N, U_N)$. A directed tree T_v^U, $v \in V$, $U \subseteq V$

[1] Intuitively congestion at a buffer obviously depends on the queue length at the buffer. More technically it is some quantity which our scheduling policy uses. We describe our scheduling policy more rigorously later.

[2] "Downstream" here means destination of a link and "upstream" means source of a link.

[3] Our scheduling policy requires the knowledge of the queue lengths at both source and destination of a link whereas the scheduler generally resides at the source. So strictly speaking some amount of nonlocal information is necessary, but the scheduler can compute this "nonlocal" information from binary bits sent from the destination buffers sometime during the scheduling decision interval. We discuss this in details later.

is a subgraph of G, $G' = (V', E')$, such that $V' \subseteq V$, $E' \subseteq E$, $v \cup U \subseteq V'$, there exists a unique directed path from v to u, for all $u \in V'$ and an edge must originate from every vertex in $V' \setminus U$. A directed tree $T_{v_n}^{U_n}$ can carry the traffic of session n. A collection $\mathcal{T}_n$ of eligible multicast trees are prespecified as the trees through which session n traffic can be transported. $\mathcal{T}_n$ may include all possible trees $T_{v_n}^{U_n}$ which can carry the traffic from v_n to U_n, or a proper subset thereof. $|\mathcal{T}_n| = M_n$. A tree in $\mathcal{T}_n$ can be described by an indicator vector, $T_n^m = (t_e, e \in E)$, where $t_e = 1$ if edge e is a part of T_n^m and $t_e = 0$ otherwise. We now illustrate the model with an example.

Example II.1: Consider the network represented by the directed graph in figure 1. There are 9 nodes represented by the vertices and 15 links represented by the edges. There are two multicast sessions, session 1 and 2. 4 and 7 are the source nodes of sessions 1 and 2 respectively. The destination nodes are $\{2, 5, 9\}$ and $\{3, 6, 8\}$ respectively. Thus $(v_1, U_1) = (4, \{2, 5, 9\})$ and $(v_2, U_2) = (7, \{3, 6, 8\})$. $\mathcal{T}_1 = \{T_1, T_2\}$ $\mathcal{T}_2 = \{T_3, T_4\}$. $T_1 = \{(4, 2), (4, 5), (5, 9)\}$. $T_2 = \{(4, 1), (1, 2), (4, 6), (6, 5), (6, 8), (8, 9)\}$. $T_3 = \{(7, 8), (7, 6), (6, 3)\}$, $T_4 = \{(7, 8), (8, 6), (6, 3)\}$, where (v_1, v_2) represents the directed edge from v_1 to v_2. Note that neither $\mathcal{T}_1$ nor $\mathcal{T}_2$ contains all possible trees for the respective sessions, e.g., $T_5 = \{(4, 3), (3, 1), (1, 2), (4, 6), (6, 5), (6, 8), (8, 9)\}$ can also carry session 1 traffic but is not included in $\mathcal{T}_1$. The indicator vector for T_1 is $(1, 1, 1, 0, \ldots, 0)$, if $(4, 2), (4, 5), (5, 9)$ corresponds to the first 3 edges.

We do not impose any particular structure on $\mathcal{T}_n$. $\mathcal{T}_n$ can consist of all directed trees $T_{v_n}^{U_n}$ between source v_n and the set of destinations U_n. However in a virtual circuit like scenario, where the packets contain only route identifiers and not the entire route, this would generate huge routing table entries at the routers because the total number of possible multicast trees $T_{v_n}^{U_n}$ is considerably large. So it is realistic to assume that $\mathcal{T}_n$ will be a proper subset of the set of all directed trees $T_{v_n}^{U_n}$. The actual size of this subset should depend on the available router memories. In a datagram like scenario, packets contain the entire path to be followed in the header and routers need only have entries regarding currently active trees. Thus memory constraint may not force $\mathcal{T}_n$ to be a proper subset of the set of all directed trees $T_{v_n}^{U_n}$. But there may be other constraints, e.g., all multicast trees $T_{v_n}^{U_n}$ may not satisfy the requirements of session n, e.g., session n may demand certain quality of service gurantees in terms of the end-to-end delay along the individual paths from the source to each of the destination nodes and possibly a bound on the variation among the delays along the individual source destination paths. For instance during a teleconference it is important that all participants hear the speaker around the same time, else the communication lacks the feeling of an interactive face-to-face discussion[17]. In high speed environments the end to end delays depend primarily on the propagation delays. So $\mathcal{T}_n$ can consist of only those trees which satisfy the requirements of session n, or a proper subset thereof.

Informally the necessary condition for system stability[4] is that the sum of arrival rates of traffic in all the trees of the same or different sessions passing through a link e does not exceed the link capacity, i.e.,

$$\sum_{n=1}^{N} \sum_{m=1}^{M_n} a_n^m T_n^m \leq (C_1, \ldots, C_{|E|}) \qquad \text{(capacity condition)},$$

where a_n^m is the traffic arrival rate in the mth tree of the nth session, T_n^m is the indicator vector for the mth tree of the nth session and C_e is the capacity of the eth link. However it is not obvious whether this condition gurantees stability in any arbitrary network. A major contribution of this paper is to prove that it is indeed so or rather "almost" so, that is if we allocate the resources as per MMRS the necessary condition for system stability with strict inequality, turns out to be sufficient as well. It is in this sense that MMRS maximizes throughput. We investigate this issue later.

III. Routing, Scheduling and Congestion Control in Multicast Networks

Session n traffic can reach the destinations through one of the many possible trees in $\mathcal{T}_n$, e.g., incoming session 1 traffic can reach its destination via trees T_1 and T_2 in Example II.1. The resource allocation policy would decide at the appropriate time which tree the traffic would follow. It should *load balance*, that is respond to congestion in the currently active trees and route incoming traffic to relatively lightly loaded trees. It is also expected to compute the congestion status of the trees efficiently. This is not likely to be the case if the decision is based on any suitably defined weight of the entire tree as per the discussion in Section I.

In general the trees of the same or different sessions would overlap on the links and at most one of them can be served at one time, e.g., trees T_3 and T_4 overlap on link $(7, 8)$ in Example II.1. So the resource allocation policy would decide at the appropriate time the trees that should be scheduled on the links. Intuitively, it should "spread out" the congestion in the network, i.e., if an upstream node of a session is heavily congested, while the downstream node is not, then traffic from that session should be served on the link. This would decrease the congestion in the heavily loaded upstream node at the expense of increasing the congestion at the lightly loaded downstream node.

Another interesting question worth investigating is how often the routing and scheduling decisions should be taken. These decisions can be taken at intervals of fixed or

[4]We defer the formal definition of "arrival rate" and "system stability" till Section VI.

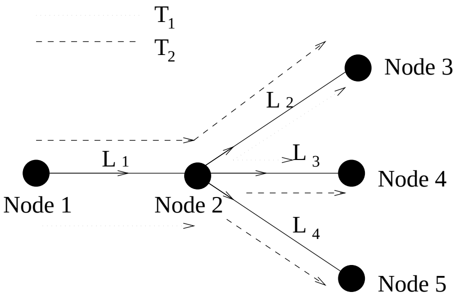

Fig. 2. Tree T_1 passes through links L_1, L_2, and L_3. Tree T_2 passes through links L_1, L_2, L_3 and L_4.

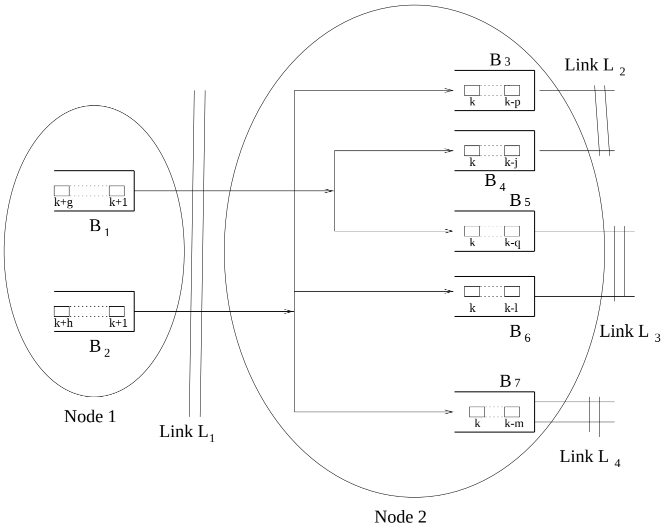

Fig. 3. Logical buffers at Node 1 and Node 2 of Figure 2.

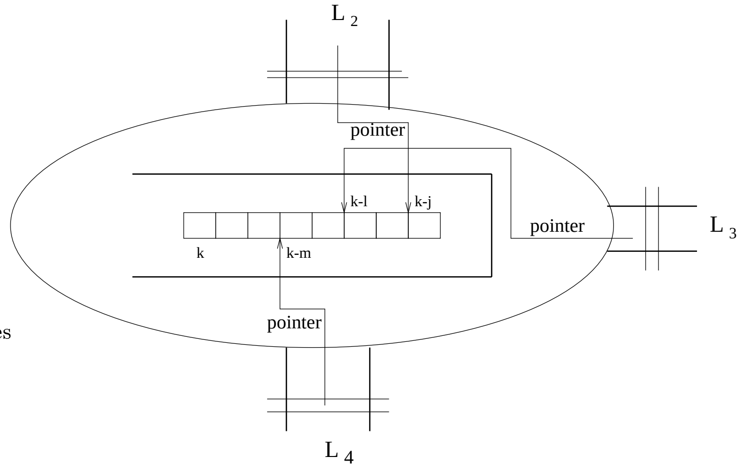

Fig. 4. A physical buffer storing T_2 packets at Node 2 of Figure 2.

bounded length. These decisions can also be taken based on the queue lengths at the nodes.

To the best of our knowledge there does not exist any generalized routing and scheduling policy which effectively addresses the above issues in multicast networks. The multipurpose multicast routing and scheduling policy (MMRS) addresses all of the above issues in a flexible manner. We describe MMRS in the following section.

IV. Multipurpose Multicast Routing and Scheduling Policy

We first present an informal description of MMRS. It takes routing and scheduling decisions at intervals, the intervals satisfy some properties to be described later. Routing decision is to route an incoming session n message to the tree which has the shortest weighted queue length at the origin, v_n, at the decision instant. The routing policy for session n remains valid till the next routing decision instant. The scheduling decision is to schedule service at a link to the tree which has the maximum difference between the queue lengths of its upstream buffer weighted by a scale factor and a weighted sum of the queue lengths of its downstream buffers, amongst all the trees with nonempty buffers contending for service on the same outgoing link (ignoring some bias terms for the moment), at the decision instant. The scheduling policy for trees on the same outgoing link remains the same till the next scheduling decision instant (assuming that the scheduled tree does not empty in between). The buffers we have referred to need not be *physical buffers* but are rather *logical buffers*. We explain our concept of *logical buffers* below.

Consider session n, (v_n, U_n) and edge e of $G = (V, E)$. Every directed edge e of G has an origin vertex $o(e)$ and a destination vertex $d(e)$, e.g., $o(e_1) = 1$ and $d(e_1) = 2$ in Figure 1. $B_{mne}(t)$ is the number of session n packets[5] travelling through the mth tree in $\mathcal{T}_n$ waiting at $o(e)$ at the end of slot t (or the beginning of slot $t+1$) for travelling to $d(e)$ through link e in slot $t+1$. B_{mne}s are the logical buffers.

Example IV.1: Consider Figures 2 and 3. Tree T_1 of session 1 passes through link L_1, L_2, L_3 and T_2 of session 2 passes through link L_1, L_2, L_3, L_4 in Figure 2. Figure 3 shows the $B_{mne}(t)$s. $B_1(t)$ is actually $B_{T_1 1 L_1}(t)$ and $B_2(t)$ is actually $B_{T_2 2 L_1}(t)$. Similarly $B_3(t)$ is $B_{T_2 2 L_2}(t)$, $B_4(t)$ is $B_{T_1 1 L_2}(t)$ $B_5(t)$ is $B_{T_1 1 L_3}(t)$, $B_6(t)$ is $B_{T_2 2 L_3}(t)$, and $B_7(t)$ is $B_{T_2 2 L_4}(t)$.

We would like to point out that the B_{mne}s may not always represent separate memory locations, particularly for the different es with the same origin node $o(e)$. It may be necessary to transmit the same packet over multiple links originating from the same node on account of multicast, e.g., in Figure 2 any T_1 packet must be transmitted across links L_2 and L_3 from node 2. Similarly any T_2 packet at Node 2 must be transmitted across links L_1, L_2 and L_3 from node 2. It is wasteful to store copies of the same packet in different physical buffers B_{mne} meant for

[5]For simplicity we state MMRS for slotted arrival and service, i.e., consider packetized traffic only. It can be easily generalized to more general cases.

different links at the same node ($B_{T_1 1 L_2}(t)$ and $B_{T_1 1 L_3}(t)$ i.e., $B_4(t)$ and $B_5(t)$ respectively in Figure 3). However it is necessary to keep track of the $B_{mne}(t)$s for the implementation of the policies we shall describe shortly. This can be done without storing multiple copies of the same packet in separate memory locations. There can be a single physical buffer at each node for storing all packets (one copy each) of a tree travelling through the node. A packet remains in the buffer till it has been transmitted across all the necessary links originating from the same node. Each link can easily keep track of the number of packets of each tree, it has still to transmit using pointers, i.e., by maintaining a pointer at the first packet it needs to transmit and sliding the pointer backwards when the packet is transmitted. These numbers are precisely the $B_{mne}(t)$s.

Example IV.2: Figure 4 shows the physical buffer at Node 2 for tree T_2 of Example IV.1. This buffer contains the tree T_2 packets at Node 2. It currently has packets numbered $k-j, \ldots, k$. All of them need to be transmitted across link L_2. L_2 maintains a pointer at the first packet it has to transmit, packet $k-j$. Link L_3 has transmitted packets $k-j, \ldots, k-l-1$. It has still to transmit packets $k-l$ to k. So it maintains a pointer at the $k-l$th packet. Similarly L_4 has already transmitted packets $k-j, \ldots, k-m-1$ but needs to transmit packets $k-m$ to k. It has a pointer at the $k-m$th packet. Figure 3 shows the contents of the separate logical buffers for this tree at Node 2 (B_3, B_6 and B_7). $k+1$th and subsequent packets of T_2 are still waiting at Node 1 for transmission across Link L_1 to Node 2 (Refer to figure 3).

For simplicity, we will refer to $B_{mne}(t)$s, $m = 1, \ldots, M_n$, $e \in E$, $n = 1, \ldots, N$ as $B_1(t), B_2(t), \ldots, B_M(t)$. (e.g., $B_1(t)$ denotes $B_{T_1 1 L_1}(t)$ in figure 3.). Also unless otherwise mentioned buffers indicate logical buffers.

Note that the packets in the logical buffer B_i belongs to the same session for every slot t. We denote this session by $n(i)$. Also all packets in B_i are physically located at the same vertex, $u(i)$ and all packets in B_i will be transmitted through the same link, $e(i)$. Since all the packets in B_i travel through the same tree, they had been at the same buffer B_j before they reached B_i. We denote j by $p(i)$ (predecessor of i). Also a packet will move to a set of buffers after transmission from B_i. For example if e_1, e_2 are in the mth tree in $\mathcal{T}_{n(i)}$, and e_1, e_2 are incident from $d(e(i))$, then a packet traversing along the mth tree will have to be transmitted across e_1 and e_2 from $d(e(i))$ and hence must reach buffers B_{j_1} and B_{j_2} at $d(e(i))$ after transmission from B_i, where $p(j_1) = p(j_2) = i$, $n(j_1) = n(j_2) = n(i)$ and $u(j_1) = u(j_2) = d(e(i))$, $e(j_l) = e_l$, $l \in \{1,2\}$. Thus for every buffer B_i, there exists a set of buffers, Z_i, such that any packet transmitted from buffer B_i reaches every buffer in Z_i, at the end of the transmission time. $Z_i = \phi$, if the mth tree terminates at $d(e(i))$.

Example IV.3: Refer to Example IV.1. $n(1) = n(4) = n(5) = 1$. $n(2) = n(3) = n(6) = n(7) = 2$ because buffers B_1, B_4, B_5 carry session 1 packets and the other buffers carry session 2 packets. $u(1) = u(2) = 1$ and $u(3) = \ldots = u(7) = 2$, $e(1) = e(2) = L_1$, $e(3) = e(4) = L_2$, $e(5) = e(6) = L_3$, $e(7) = L_4$. All packets in tree T_2 move from B_2 at node 1 to node 2 through L_1. All of these packets must be transmitted across L_2, L_3, L_4. Thus any packet transmitted from B_2 reaches B_3, B_6, B_7. Thus $Z_2 = \{B_3, B_6, B_7\}$. Also $p(3) = p(6) = p(7) = 2$, since any packet in B_3, B_6, B_7 comes from B_2. Similarly $p(4) = p(5) = 1$. $Z_1 = \{B_4, B_5\}$.

Every tree can be described by a sequence of logical buffers and every buffer B_i corresponds to a unique tree in $\mathcal{T}_{n(i)}$. We denote this tree by $m(i)$. The logical buffers corresponding to different trees are mutually disjoint.

Routing policy is necessary for exogeneous arrivals. Exogeneous arrivals for session n are routed to trees in $\mathcal{T}_n$ and thereafter it is only scheduling service to contending buffers, i.e., buffers at the origin of the same link.

The policies can be described as follows. For simplicity let the trees be denoted by integers, i.e., $\mathcal{T}_n$ is a subset of integers. The routing vector, $\Gamma(t) = (T_1(t), \ldots, T_N(t))$, $1 \leq T_n(t) \leq M_n$. Let O_{mn} be the set of buffers of the mth tree of $\mathcal{T}_n$ at v_n, the source node of seesion n, i.e., $O_{mn} = \{B_i : n(i) = n, m(i) = m, u(i) = v_n\}$. Consider Example IV.1. Let T_1, T_2 be the first trees of the respective sessions. Let node 1 be the source nodes of both sessions 1 and 2. $O_{11} = \{B_1\}$ and $O_{12} = \{B_2\}$. O_{mn} may consist of multiple buffers, e.g., if T_1 had originated from node 2 instead of node 1, then $O_{11} = \{B_4, B_5\}$. When a session n packet arrives at slot $t+1$, it is routed to the $T_n(t+1)$th tree in $\mathcal{T}_n$ and it arrives at every buffer in $O_{T_n(t+1)n}$. $T_n(t)$ is updated at time instants ω_ι^n, i.e.,

$$T_n(t+1) = \begin{cases} \arg min_{1 \leq m \leq M_n} \sum_{B_k \in O_{mn}} c_k B_k(t) & t+1 \in \{\omega_\iota^n\}_{\iota=1}^{\infty} \\ T_n(t) & \text{otherwise.} \end{cases}$$

c_is are positive constants. $\{\omega_\iota^n\}_{\iota=1}^{\infty}$ are the time instants at which routing decisions are taken for session n, The routing decision taken at $\{\omega_\iota^n\}$ is to route a session n packet arriving exogeneously at a slot t, $\omega_\iota^n \leq t < \omega_{\iota+1}^n$, to the tree which has the shortest weighted queue lengths at ω_ι^n at the origin buffer v_n.

The routing times $\{\omega_\iota^n\}$ may be:

1. At fixed intervals $\omega_{\iota+1}^n - \omega_\iota^n = T_n$. R1
2. Of bounded difference $0 \leq \omega_{\iota+1}^n - \omega_\iota^n \leq T_r$, $\omega_{\iota+1}^n - \omega_\iota^n$ i.i.d.[6] R2

[6] $\omega_{\iota+1}^n - \omega_\iota^n$ i.i.d. $\forall \iota$. We do not need $\omega_{\iota+1}^{n1} - \omega_\iota^{n1}$ to be identically distributed as $\omega_{\iota+1}^{n2} - \omega_\iota^{n2}$. We also allow dependence amongst the residual times $\omega_{\iota+1}^{n1} - t$, and $\omega_{\iota+1}^{n2} - t$, $\forall n_1, n_2$, where $\omega_\iota^{n1} \leq t < \omega_{\iota+1}^{n1}$ and $\omega_\iota^{n2} \leq t < \omega_{\iota+1}^{n2}$.

3. Depend on the queue lengths. A routing decision will be taken for session n at $t+1$ if the weighted queue lengths at the origin buffer, v_n of the currently active tree ($T_n(t)$th tree) exceeds that of another tree by a certain amount, $\varrho_n \geq 0$, i.e., $t+1 \in \{\omega_\iota^n\}_{\iota=1}^{\infty}$ if there exists $m \in M_n$ s.t.

$$\sum_{B_i \in O_{T_n(t)n}} c_i B_i(t) > \sum_{B_i \in O_{mn}} c_i B_i(t) + \varrho_n \text{ R3}$$

Informally this means that routing decision is not taken for session n until the currently active tree is "sufficiently" congested, and the congestion is reflected in the origin buffer lengths. Routing decision always brings about a change in the currently active tree.

(Note that $R1$ is a subset of $R2$, we mention it explicitly to highlight its importance.)

Example IV.4: Refer to Example IV.1. Now let both T_1 and T_2 be session 1 trees numbered 1, 2 respectvely. Let node 1 be the source of session 1. $O_{11} = \{B_1\}$, $O_{21} = \{B_2\}$. Let $c_1 = c_2 = 1$. Let figure 3 show the buffers just prior to ω_ι^1 (a routing decision instant for session 1). $B_1(\omega_\iota^1 - 1) = g$, $B_2(\omega_\iota^1 - 1) = h$. Let $g < h$. $T_1(t) = 1$, for all t such that $\omega_\iota^1 \leq t < \omega_{\iota+1}^1$. Every session 1 packet which arrives at t, $\omega_\iota^1 \leq t < \omega_{\iota+1}^1$ is routed to tree T_1. A fresh routing decision is taken at $\omega_{\iota+1}^1$. $T_1(t) = 2$ if $B_1(\omega_{\iota+1}^1 - 1) > B_2(\omega_{\iota+1}^1 - 1)$ and $T_1(t) = 1$ if $B_1(\omega_{\iota+1}^1 - 1) < B_2(\omega_{\iota+1}^1 - 1)$, $\omega_{\iota+1}^1 \leq t < \omega_{\iota+2}^1$. If $T_1(t) = 2$ at $t = \omega_{\iota+1}^1$ then all new packets of session 1 are routed to T_2 till the next routing decision is made, else all new packets are still routed to T_1. Note that $T_1(t)$ always changes value at the routing decision instants, if the routing decision intervals follow property (R3).

This routing policy applies to all datagram like networks, including the internet. It does not apply to networks where routing decision is taken once for every session.

Next we describe the scheduling. $\vec{E}(t+1)$ is the activation vector with M compnents, at the $t+1$th slot.

$$E_i(t+1) = \begin{cases} 1, & \text{if a packet from } B_i \text{ is served at } e(i) \\ & \text{at slot } t+1 \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $E_i(t+1) = 1$ iff the ith buffer is scheduled for packet transmission on $e(i)$ at the $(t+1)$th slot. If $Z_i \neq \phi$, this packet reaches every buffer in Z_i, and also a destination if $d(e(i))$ is a destination of session n. If $Z_i = \phi$, then $d(e(i))$ is in $U_{n(i)}$ and the packet reaches its destination. $\vec{E}$ is updated at time instants Ω_ι^e, $e \in E$. Let $P_e(t)$ be the set of nonempty buffers in slot t whose packets have to be transmitted across link $e \in E$.

$P_e(t+1) = \{B_i : e(i) = e, B_i(t) > 0\}$

$$\begin{aligned} D_i(t+1) &= c_i B_i(t) - \sum_{B_k \in Z_i} c_k B_k(t) \;\; i = 1, \ldots, M \\ s_e(t+1) &= \arg\max_{i : B_i \in P_e(t+1)} \left(l_i(t+1) + D_i(t+1)\right), \\ & \quad \text{if } P_e(t+1) \neq \phi \\ s_e(t+1) &= -1, \text{ if } P_e(t+1) = \phi \end{aligned}$$

$D_i(t)$ is the difference between the queue length of buffer i weighted by a scale factor and a weighted sum of the queue lengths of the destination buffers of buffer i at the beginning of slot t. l_i can be interpreted as a $\vec{B}$ dependent or a constant bias added to D_i. Let $S_e = \{i : e(i) = e, 1 \leq i \leq M\}$ (the set of buffers contending for service from link e). For example $S_{L_1} = \{1, 2\}$, $S_{L_4} = \{7\}$ in Example IV.3. If $t+1 \in \{\Omega_\iota^e\}_{\iota=1}^{\infty}, i \in S_e$,

$$E_i(t+1) = \begin{cases} 1, & i = s_{e(i)}(t+1), \\ & l_i(t+1) + D_i(t+1) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

If $t+1 \notin \{\Omega_\iota^e\}_{\iota=1}^{\infty}, i \in S_e,\ E_i(t+1) = \begin{cases} E_i(t), & B_i(t) > 0, \\ 0 & \text{otherwise.} \end{cases}$

A packet is transmitted across link $e(i) \in E$, at slot $t+1$, if $E_i(t+1) = 1$, for some B_i, such that $e(i) = e$, else the link idles, i.e., no packet is transmitted across the link.

$\{\Omega_\iota^e\}_{\iota=1}^{\infty}$ are the time instants at which the scheduling decisions are taken for the link e, The scheduling decision is to choose a buffer B_i which has maximum $D_i + l_i$ at Ω_ι amongst all the buffers nonempty at Ω_ι and contending for service from outgoing link e. If $D_i(\Omega_\iota^e) + l_i(\Omega_\iota^e) > 0$, $i \in S_e$, then the scheduling decision is to serve a packet from B_i at each slot t, $\Omega_\iota^e \leq t < \Omega_{\iota+1}^e$, unless B_i becomes empty at $\Omega_\iota^e < t < \Omega_{\iota+1}^e$. If B_i becomes empty at some $\Omega_\iota^e < t < \Omega_{\iota+1}^e$, then the link idles till the next scheduling slot $\Omega_{\iota+1}^e$. If $D_i(\Omega_\iota^e) + l_i(\Omega_\iota^e) \leq 0$, then the link idles during the entire scheduling decision interval $[\Omega_\iota^e, \Omega_{\iota+1}^e)$.

Like the routing times $\{\omega_\iota^n\}$, the scheduling times $\{\Omega_\iota^e\}$ may be:

1. At fixed intervals $\Omega_{\iota+1}^e - \Omega_\iota^e = T_e$. S1
2. Of bounded difference $\Omega_{\iota+1}^e - \Omega_\iota^e \leq T_s$, $\Omega_{\iota+1}^e - \Omega_\iota^e$ i.i.d. S2
3. Depend on the $D_i + l_i$s, $i \in S_e$. S3

$$\begin{aligned} j_e(t) &= \arg\max_{i \in S_e} E_i(t) \\ p_e(t) &= \arg\max_{i \in S_e \cap P_e(t)} (D_i(t) + l_i(t)) \end{aligned}$$

A scheduling decision is taken for link e at the beginning of slot $t+1$ if any of the following conditions is satisfied:

[6] We had used the term "congestion" at a buffer in Section I rather loosely. $c_i B_i(t) + l_i(t)$ can be thought of as a measure of "congestion" (as used in Section I) at logical buffer B_i when it is considered as the source buffer of a link and $c_i B_i(t)$ as the measure when it is considered as the destination buffer.

(a) Currently scheduled buffer has $D_i + l_i$ sufficiently less than that of some other contending buffer. $E_{j_e(t)}(t) = 1$, $D_{j_e(t)}(t+1) + l_{j_e(t)}(t+1) \leq D_{p_e(t)}(t+1) + l_{p_e(t)}(t+1) - \varsigma_{e1}$ S3a

(b) The $D_i + l_i$ of the currently scheduled buffer becomes sufficiently negative. $E_{j_e(t)}(t) = 1$, $D_{j_e(t)}(t+1) + l_{j_e(t)}(t+1) \leq -\varsigma_{e2}$ S3b

(c) The link is currently idle but the $D_i + l_i$ of some nonempty buffer which can possibly be served by the link becomes sufficiently positive. $E_{j_e(t)} = 0$, $D_{p_e(t)}(t+1) + l_{p_e(t)}(t+1) \geq \varsigma_{e3}$ S3c

$\varsigma_{e1}, \varsigma_{e2}, \varsigma_{e3}$ are prespecified positive real numbers. Again informally this means that scheduling decision is not taken for a link till the last scheduling decision becomes "too bad" for the current state of the network.

(Note that $S1$ is a subset of $S2$.) This scheduling policy can be used both for the internet and the ATM networks.

$c_i > 0\ \forall i$. $\vec{c} = (\sqrt{c_1}, \ldots, \sqrt{c_M})$, $\vec{l}(t+1) = (l_1(t+1), \ldots, l_M(t+1))$, $l_i(t+1) = g_i(\vec{B}(t))$s, where $\vec{B}(t) = (B_1(t), \ldots, B_M(t))^T$. $g_i : R^M \rightarrow R$, $i = 1, \ldots, M$. g_is can be any arbitrary function satisfying the following property.

$$\lim_{||c\vec{b}|| \rightarrow \infty} \frac{g_i(\vec{b})}{||c\vec{b}||} = 0, \forall i \quad \text{where} \quad ||c\vec{b}|| = \sqrt{\sum_{i=1}^{M} c_i b_i^2}$$

Note that a large class of g_is satisfies the above property, e.g., any bounded function g, any linear function of $\sqrt{b_1}, \ldots, \sqrt{b_M}$, etc.

We explain the scheduling policy with an example.

Example IV.5: Refer to Example IV.1. Let Figure 3 show the buffers just prior to $\Omega_\iota^{L_1}$ (a scheduling decision instant for link L_1). $B_1(\Omega_\iota^{L_1} - 1) = g$, $B_2(\Omega_\iota^{L_1} - 1) = h$, $B_3(\Omega_\iota^{L_1} - 1) = j + 1$, $B_4(\Omega_\iota^{L_1} - 1) = p + 1$, $B_5(\Omega_\iota^{L_1} - 1) = q + 1$, $B_6(\Omega_\iota^{L_1} - 1) = l + 1$, $B_7(\Omega_\iota^{L_1} - 1) = m + 1$. Let $c_i = 1$ and $g_i(\vec{b}) = 0\ \forall i$, i.e., $l_i(t) = 0$, for all i, t. $D_1(\Omega_\iota^{L_1}) = g - p - q - 2$. $D_2(\Omega_\iota^{L_1}) = h - j - l - m - 3$. Let $p = q = j = l = m = 1$, $g = 5$, $h = 8$. Thus $D_1(\Omega_\iota^{L_1}) + l_1(\Omega_\iota^{L_1}) = 1$ and $D_2(\Omega_\iota^{L_1}) + l_2(\Omega_\iota^{L_1}) = 2$. Thus the scheduling decision is to serve a packet from B_2 in every slot till the next scheduling decision instant assuming that B_2 does not empty in between. If B_2 empties in between, then L_1 idles till the next scheduling decision instant. Now let $g = h = 1$. Both $D_1(\Omega_\iota^{L_1}) + l_1(\Omega_\iota^{L_1})$ and $D_2(\Omega_\iota^{L_1}) + l_2(\Omega_\iota^{L_1})$ are negative and hence L_1 idles till the next scheduling instant, independent of the $D_i + l_i$s in between. Now let $g_1(\vec{b}) = 4$, $g_2(\vec{b}) = 0$. In both the above cases the scheduling decision is to serve a packet from B_1 in every slot till the next scheduling decision instant assuming that B_1 does not empty in between. If B_1 empties in between, then again L_1 idles till the next scheduling decision instant. This change in the bias term $g_1(\vec{B})$ gives limited priority to session 1 on link L_1. Again if $p = q = j = l = m = 1$, $g = 5$, $h = 8$, $g_i(\vec{b}) = 0\ \forall i$, but $c_1 = c_4 = c_5 = 3$ and $c_2 = c_3 = c_6 = c_7 = 1$, then $D_1(\Omega_\iota^{L_1}) + l_1(\Omega_\iota^{L_1}) = 3$, $D_2(\Omega_\iota^{L_1}) + l_2(\Omega_\iota^{L_1}) = 2$. Thus the scheduling decision is taken in favor of B_1.

Whenever a packet arrives at a node (not necessarily exogeneously) the node must know the logical buffers B_is (or B_{mne}s) to which the packet belongs. This is necessary to keep track of the $B_i(t)$s. Thus the packet header must contain information specifying its session and the tree it has been routed to. If there is a connection setup phase associated with every session initiation, like in virtual circuit scenario, every node may know the outgoing links of every tree traversing through it. In that case the tree information may be contained in a number identifying the tree and the node can determine the outgoing edges along which the packet must travel from the tree and the session numbers. This enables the node to determine the B_{mne}s to which the packet belongs. However in datagram like scenario there is no connection establishment process. Thus the nodes do not necessarily know the outgoing links of the trees passing through them. The packet header must contain explicit information about the edge sequences of the tree in addition to the tree and the session number. Immediately after the packet arrives exogeneously it is routed to a tree as per the last routing decision and the necessary information is incorporated in the packet header. The necessary information can be the tree and the session numbers or the explicit tree path, the tree number and the session number depending upon whether the nodes know the tree paths or not.

V. Discussion

- Informally speaking, MMRS attains the maximum throughput for any arbitrary network (the precise technical statement is given in Section VI). This holds as long as the scheduling and routing decision intervals are bounded and c_is are strictly positive. We outline the proof in Section VII.
- It is interesting to note that the congestion in the entire trees need not be taken into account for routing. However any congestion downstream will ultimately cause congestion in the origin buffers and will prevent arrivals at the tree and this would help to clear the downstream congestion. One can think of policies which take congestion in the entire tree into account while routing and thus respond early to downstream congestion. These policies can not generate greater throughput. However it is not clear how MMRS would compare with those with respect to other performance criteria, e.g., delay, expected queue lengths at the buffers, etc. As we have pointed out in Section I that taking routing decision on the basis of congestion in the entire trees is infeasible from the computational complexity point of view even if the

routing decision intervals are long. Besides it is unrealistic to assume that the routers will have access to the necessary current global information particularly in large networks.

- Increasing the c_is for a session over others decreases the delay of the packets of the session at the expense of greater delay experienced by packets of other sessions. Thus c_is can be used to give limited priority to the real time sessions like audio, video and possibly to applications which fetch greater revenue. $g_i(\vec{b})$s can serve the same purpose. A judicious choice of $g_i(\vec{b})$s can decrease the expected delay of the packets in the entire network.
- If $g_i(\vec{b})$ is a function of b_i only and not of the entire vector $\vec{b}$, then MMRS can be implemented in a decentralized manner.
- If $g_i(\vec{b})$ is a computationally efficient function of b_i, then MMRS can be implemented in real time.
- Typically we would expect the l_is to be constants independent of $B_i(t)$s.
- The scheduling policy spreads the congestion in the system. A link would not serve a tree if the corresponding buffer at the destination of the link has a large message queue length as compared to the source buffer, even if it has to idle. Again if a source buffer at a link has a large queue length as compared to the destination buffer, then the corresponding tree would be served by the link, even if there exists other contending trees with destination buffers not so overcrowded as compared to source buffers. This reduces the congestion at an already congested buffer at the expense of that at a possibly lightly loaded buffer. This would reduce packet loss. The parameters c_is and $g_i(\vec{b})$s can be suitably modified to reduce packet loss even further, if necessary.
- Since the message queue lengths at the source reflect the congestion status of the possible routes and hence that of the session, end to end congestion control measures may be based on this observation. Thus the message queue lengths at the source of the session give *implicit* feedback about the congestion state of the paths followed. This is a significant advantage for multicast applications because explicit feedbacks often lead to feedback *implosion*[28]. This solution of implicit feedback is inherently scalable. However this implicit feedback may not be able to substitute the need for explicit feedback entirely, but may be used in conjunction, so that much more infrequent explicit feedback may serve the purpose.
- MMRS is completely adaptive, in the sense that its implementation is independent of the statistics of the arrival process. However if the parameters c_i, g_i are selected to minimize delay, etc., then the knowledge of the statistics of the arrival process may help in making better choices.

VI. Formal Throughput Properties of MMRS

Intuitively the concept of stability of a queueing system is associated with the queue length process at the *physical buffers*(buffers corresponding to actual storage locations). Let $X_u(t)$ be the number of packets in a physical buffer u by the end of slot t (or the beginning of slot $t+1$). We define the system to be stable if there exists a family of random vectors, $\hat{X}^{(i,j)}$ $i = 0, \dots, P-1$, $j = 0, \dots, Q-1$, $E\hat{X}^{(i,j)} < \infty$, $\forall i, j$, P, Q finite, such that $\{\vec{X}(t)\}_{t=0}^{\infty}$ can be partitioned into Q subsequences $\{\vec{X}(td+\theta)\}_{t=0}^{\infty}$, $\theta = 0, \dots, Q-1$, and $\vec{X}(td+\theta)$ converges weakly to a random vector $\hat{X}^{(i,j+\theta\%Q)}$, where (i,j) are uniquely specified given the initial phase of the system. The *phase*, of the system should contain some information not contained in the queue lengths at the physical or the logical buffers. For example the phase of the system could be the residual times for the next routing, scheduling decisions for MMRS with routing, scheduling decision intervals satisfying R1 or R2 and S1 or S2 respectively. The system may not have any phase (as when R3, S3 are satisfied) and then $P = Q = 1$. The intuition behind this definition is that we generally consider a system to be stable, as long as the physical buffers do not "blow up" and that would not happen if $\vec{X}(t)$ converges weakly to one of finitely many finite mean random vectors and hence those systems should be considered stable.

A node is a possibly multi-input multi-output multicast switch with the ability to serve packets to several outgoing links simultaneously. Figure 2 shows a node, Node 2 with one incoming link and three outgoing links. Refer to Example IV.1. Packets reach at Node 2 via trees T_1 and T_2. The node can simultaneously transmit a T_2 packet into L_4 and a T_1 packet into L_3. Both packets reach Node 2 via link L_1. The packets can be queued at the input or at the output of the node or queueed in a shared memory mode. The physical buffers are the memory locations which store these packets. The relation between the physical buffers and the logical buffers depend upon whether the packets are input queued or output queued or stored in a shared memory mode.

- If the packets are input queued, then the physical buffers are as those described in page 5. Note that here packets are replicated[7] (if at all) only when they are transmitted to the output (*replication-at-sending* (RAS)[4]). Let E_{Ti} be the set of outgoing links of tree T at node i, e.g., $E_{T_1 2} = \{L_2, L_3\}$, $E_{T_1 1} = \{L_1\}$, $E_{T_2 2} = \{L_2, L_3, L_4\}$, in Example IV.1, Figure 2. Let physical buffer u store packets of tree T of session n at node i. $X_u(t) = \max_{e \in E_{Ti}} B_{Tne}(t)$.
- If the packets are output queued, then every outgoing link has a physical buffer to store all packets transmitted to it from its origin node. If physical buffer u stores the packets output queued at e, then $X_u(t) =$

[7] Replication occurs only when the corresponding tree forks at the node

$\sum B_{mne}(t)$, where the summation is over all trees m of all sessions n which pass through link e. Note that here packets are replicated (again replication occurs if the corresponding tree forks at the node) immediately upon arrival and subsequently transmitted to the output queue (*replication-at-receiving* (RAR)[4]).

- The node may be a shared memory switch with the memory fully shared between all queues. There is only a single physical buffer per node. Replication can be RAR or RAS. Let the physical buffer w store the packets at node w. $X_w(t) = \sum_{i:u(i)=w} B_i(t)$ for RAR. $X_w(t) = \sum_{T\in\mathcal{V}} \max_{e\in E_{Tw}} B_{Tn_Te}(t)$ for RAS, where $\mathcal{V}= \cup_{n=1}^{N}\mathcal{T}_n$, E_{Tw} is the set of outgoing links of tree T at node w, n_T is the session corresponding to tree T. The RAR scheme has been used in several shared-memory multicast ATM switches[12]. The RAS scheme has recently been adopted in shared-memory switches[19].

It follows from the relation between physical buffer and logical buffer queue lengths that if there exists a family of random vectors, $\hat{B}^{(i,j)}$ $i = 0,\ldots,P-1, j = 0,\ldots,Q-1$, $E\hat{B}^{(i,j)} < \infty$, $\forall i,j$, P, Q finite, and $\{\vec{B}(t)\}_{t=0}^{\infty}$ can be partitioned into Q subsequences $\{\vec{B}(td+\theta)\}_{t=0}^{\infty}$, $\theta = 0,\ldots,Q-1$, such that $\vec{B}(td+\theta)$ converges weakly to a random vector $\hat{B}^{(i,j+\theta\%Q)}$, $i \in \{0,\ldots,P-1\}$, $j \in \{0,\ldots,Q-1\}$, i,j uniquely determined given the initial phase of the system, then the system is stable.

Let a_n be the expected number of session n packets arriving in a slot. We call $(a_1, a_2, \ldots, a_N)$ the arrival rate vector. The necessary condition for system stability is that the arrival rate vector must satisfy the capacity condition stated in Section II. We have proved the above in technical report[20]. We call an arrival rate vector feasible if for each session n the total traffic a_n can be split in portions a_n^m, $m = 1,\ldots,M_n$, and $\sum_{m=1}^{M_n} a_n^m = a_n$, where M_n is the total number of trees in $\mathcal{T}_n$ and a_n^m, $m = 1,\ldots,M_n$, $n = 1,\ldots,N$ satisfies the capacity condition with strict inequality. Intuitively a_n^m is the amount of traffic routed through tree m in $\mathcal{T}_n$.

For simplicity assume that each packet has a deterministic service time equal to 1 slot. In that case an arrival rate vector is feasible if

$$\sum_{n=1}^{N}\sum_{m=1}^{M_n} a_n^m T_n^m < (1,\ldots,1) \tag{1}$$

T_n^m is the indicator vector of the mth tree in T_n. MMRS renders the system stable for every feasible arrival rate vector. The condition for feasibility of an arrival rate vector is the same as the necessary condition for stability for all practical purposes. Technically speaking, if an arrival rate vector $(a_1,\ldots,a_N)$ satisfies the necessary condition for stability, then MMRS renders the system stable for any arrival vector $(a_1-\epsilon,\ldots,a_N-\epsilon)$, for any arbitrarily small ϵ. Quite possibly, MMRS renders the system stable for arrival vector $(a_1,\ldots,a_N)$ itself (that is if it satisfies the capacity condition with strict inequality). Thus MMRS is throughput optimal for all practical purposes. We outline the proof in the next section.

VII. Outline of Proof

We make the following assumptions for the purpose of analysis. Arrival and service are slotted. Each session has its own exogeneous i.i.d. arrival stream of packets, $\{A_n(t)\}_{t=1}^{\infty}$, where $A_n(t)$ is the number of session n packets arriving in slot t. $A_n(t) \le K_n$, $\forall n,t$. K_n is a positive integer for all n. As mentioned in Section VI, each packet has a deterministic service time equal to 1 unit. Note that the dependences between $A_{n_1}(t)$ and $A_{n_2}(t)$ have not been ruled out for any $n_1 \ne n_2$. However $A_{n_1}(t_1)$s and $A_{n_1}(t_2)$s need to be independent for $t_1 \ne t_2$.

The detailed proof is long, involved and technical in nature and has been presented in technical report[20]. The space limitations constrain us to just outline the proof here. We study the evolution of a state vector $\vec{Y}(t)$, which captures the virtual buffer queue lengths $\vec{B}(t)$, the scheduling vector $\vec{E}(t)$, the routing vector $\vec{\Gamma}(t)$ and some other state variables depending upon whether the routing policy satisfies (R1) or (R2) or (R3) and whether the scheduling policy satisfies (S1) or (S2) or (S3). $\{\vec{Y}(t)\}_{t=0}^{\infty}$ is a countable state discrete time Markov chain, state space $\mathcal{X}$ with a single closed communication class S of periodicity d and possibly other transient states. We showed that a quadratic liapunov function of $\{\vec{Y}\}$ has negative drift when the arrival rate vector is feasible. The result follows from work relating drift analysis and stability of Markov chains.

Technical report[20] also contains proof for the same results under much more general statistics for the arrival process. Now $\{A_n(t)\}_{t=1}^{\infty}$ is a markov modulated process and a_n is the expectation of $A_n(t)$ under the stationary distribution of the underlying markov process $S(t)$. We showed that if the arrival rate vector $(a_1,\ldots,a_n)$ is feasible then a quadratic liapunov function of $\{\vec{Y}, S\}$ has negative drift when averaged over a sufficiently long interval. Again stability results follow from work relating drift analysis and stability of Markov chains. [25] adopted a similar approach while proving stability results for markov modulated service process. We believe that in general the results hold under much more general statistics for the arrival and the service process.

VIII. Conclusion

As discussed in Section I, most of the existing research in multicast routing have advocated the use of a single multicast tree per session. Significant amount of research have been directed towards the construction and the nature of the tree, e.g., whether the tree should be a shortest path tree or a core based tree and how to form these trees for a source and a set of destinations. No existing routing protocol provides for load balancing. We have assumed that the set of possible multicast trees are known for a session and have focussed on the selection in a dynamic manner of the appropriate tree for an incoming packet. The scheduling in current multicast network is still best effort service. We have proposed a class based scheduling based on local information. The maximum throughputs attained by existing multicast routing protocols like DVMRP[5], CBT[1], PIM[6], MIP[16] are not known. We proposed a throughput optimal routing and scheduling. Throughput optimal algorithms in generalized multicast networks were not known before. However some previous work exists for broadcast networks. [14] proposes a throughput optimal algorithm for broadcasts in a mesh network. [26] proposes a routing policy which attains at least 50% of the maximum possible throughput in an arbitrary broadcast network. Since unicast and broadcast are special cases of multicast, MMRS applies to both unicast and broadcast networks as well.

Throughput optimal routing and scheduling policies are known for unicast networks[24], [23]. However the intricacies of *traffic multiplication* in the multicast scenario can not be captured by the system model introduced in [24] because as opposed to traffic from a buffer reaching one of many possible buffers in the unicast scenario, traffic from a buffer may need to reach multiple buffers in the multicast case. Thus scheduling needs to be modified suitably to take this into account. Besides, our scheduling policy is more general even in the unicast context, because we allow taking scheduling decision at intervals whereas [23] advocates taking scheduling decision every slot. We also introduce the use of some scale factors and queue length dependent or constant bias terms in making scheduling decisions. These scale factors and bias terms can be used to allow limited priority to sessions over one another, reduce overall delay, packet loss etc. Neither [23] nor [24] uses these scale factors and bias terms. The routing policies are inherently different in the two cases. Here the entire path which the packet follows is decided once for each packet and immediately after its arrival and as we have discussed before this decision is computationally simple. There the routing decision is taken freshly at every buffer. None of these decisions determine the entire path alone, but all these decisions cumulatively determine the entire path. This makes MMRS simpler to implement.

References

[1] 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

[2] J. Bolot and A. Vega Garcia. Control mechanisms for packet audio in the Internet *Proceedings of IEEE Infocom* '96, San Francisco, April 96, pp. 232-239.

[3] D. Cheriton and S. Deering. Host groups: A multicast extension for datagram internetworks. *Proceedings of the 9th Data Communications Symposium* (Sept. 1985), ACM/IEEE New York, 1985, 172-179

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

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

[6] S. Deering, D. Estrin, D. Farinacci and 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.

[7] C. Diot, W. Dabbous, J. Crowcroft. Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms *IEEE Journal on Selected Areas In Communications,* Vol. 15 No. 3 April 1997.

[8] H. Eriksson. Mbone: The multicast backbone, *Comm. ACM,*vol 37, no. 8, pp. 54-60, Aug 1994.

[9] R. Frederick. NV-X11 video conferencing tool, *Unix Manual Page,* XEROX PARC, 1992.

[10] V. Hardeman, M-A. Sasse and I. Kouvelas. Successful multiparty audio communication over the Internet, *Commun. ACM,* 1997.

[11] International Business Machines Corporation. *Technical Reference PC Network.* Doc. 6322916 *Acta Informatica* 15,(1981) pp. 141-145.

[12] T. Kozaki, N. Endo, Y. Sakurai, O. Matsubara, M. Mizukami and K. Asano. *32x32* shared buffer type ATM switch VLSI's for B-ISDN's, *IEEE Journal on Selected Areas In Communications,* Vol. 9, pp. 1239 – 1247 Oct. 1991.

[13] S. McCanne and V. Jacobson. Vic: A flexible framework for packet video *Proceedings of ACM Multimedia,* Nov 1995.

[14] E. Modiano and A. Ephremides. Efficient Algorithms for Performing Packet Broadcasts in a Mesh Network *IEEE Transactions on Networking,* Vol. 4. No. 4. August' 96

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

[16] 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.

[17] G. Rouskas and I. Baldine. Multicast Routing with End-to-End Delay and Delay Variation Constraints *IEEE Journal on Selected Areas In Communications,* Vol. 15 No. 3 April 1997.

[18] Sun Microsystems. *Remote Procedure Call Reference Manual.*Mountain View, California, Oct. 1984

[19] H. Saito, H. Yamanaka, H. Yamada, M. Tuzuki, H. Koudoh and Y. Matsuda and K. Oshima. Multicasting function and its LSI implementation in a shared multibuffer ATM switch, *Proceedings of IEEE INFOCOM'94,*, Toronto, Canada, June 1994, pp. 315 – 322.

[20] S. Sarkar and L. Tassiulas. A Framework for Routing and Congestion Control in Multicast Networks, *Technical Report* TR98 – 64, Institute for Systems Research and University of Maryland, 1998.

[21] M. Satyanarayanan and F. Siegal. MultiRPC: A parallel remote procedure call mechanism. Tech Rep. CMU-CS-86-139, Carnegie-Mellon Univ., Aug. 1986

[22] A. Shaikh and K. Shin. Destination-Driven Routing for Low-Cost Multicast *IEEE Journal on Selected Areas In Communications,* Vol. 15 No. 3 April 1997.

[23] L. Tassiulas and A. Ephremides. Stability properties of controlled queueing systems and scheduling policies for maximum throughput in multihop radio networks, *IEEE Transactions on Automatic Control,* 37(12) : 1936 – 1946, December 1992.

[24] L. Tassiulas. Adaptive Back Pressure Congestion Control based on Local Information *IEEE Transactions on Automatic Control,* Vol. 40 No. 2 February 1995.

[25] L. Tassiulas. Scheduling and performance limits of networks with constantly changing topology. *IEEE Transactions on Information Theory,* Vol. 43 No. 3 pp. 1067 – 1073, May 1997.

[26] E. Varvarigos and A. Banarjee. Routing Schemes for Multiple Random Broadcasts in Arbitrary Network Topologies *IEEE Transactions on Parallel and Distributed Systems* Vol 7, No. 8, Aug' 96.

[27] Vat web server: http://www-nrg.ee.lbl.gov/vat/.

[28] B. Vickers, M. Lee and T. Suda. Feedback Control Mechanisms for Real-Time Multipoint Video Services *IEEE Journal on Selected Areas In Communications,* Vol. 15 No. 3 April 1997.

[29] White board software available through ftp://ftp.ee.lbl.gov/conferencing/wb/.