Local and Congestion-Driven Fairness Algorithm in Arbitrary Topology Networks

Alain Mayer, Yoram Ofek, *Member, IEEE,* and Moti Yung

***Abstract*—Typically, bandwidth reservation is not made for data applications. Therefore, the only way to provide *minimum bandwidth guarantees* to such an application is by using a fairness mechanism to regulate the access to the network and by controlling the packet loss (i.e., congestion) inside the network. There are numerous works treating fairness in ring networks, however, there are almost no such works on fairness in arbitrary topology networks. The context of this work is fairness in an arbitrary topology network, the MetaNet, which employs *convergence routing*, a *loss-free routing technique* which is a variant on deflection routing. We note that minimum bandwidth guarantee combined with loss-free routing are the desired quality-of-service (QoS) attributes for most data applications.**

While developing the mechanisms, we also present *performance measures* to assess the new access- and flow-control algorithm:

- **i) *Locality and congestion-driven*—only the subnetwork containing conflicting traffic streams becomes involved in the fairness regulation. Furthermore, the fairness regulation is activated only when congestion occurs. This implies that when there is no congestion, nodes can access the network immediately and freely, which is a key requirement for distributed computing.**
- **ii) *Scalability*—the data-structure sizes used in the algorithm are a function of the switching node degree, and use *constant space* control signals of two bits only (the ATM standard, for example, dedicates four bits in the header of each cell to generic flow-control).**
- **iii) *Linear access time in the congested subnetwork*—measured by "the maximal clique in what we call the *conflict graph* to which a node belongs," and a *frequency* which is inverse linear in this parameter (when the traffic pattern stabilizes).**

I. Introduction

THE main result of this paper is a new technique for regulating the access from bursty traffic sources in the context of *arbitrary topology* packet switched networks. The proposed algorithm and complexity measures capture the notion of *fair access* via *local scheduling* that is activated only after some predefined congestion condition has occurred—otherwise, the access is immediate and free (as in Ethernet). This implies, on one hand, that whenever there is more capacity in the network, the congestion-driven fairness algorithm is activated less frequently, and consequently, the expected access time becomes shorter. On the hand, when a subnetwork is congested the fairness algorithm becomes active, which in turn reduces the congestion by regulating the traffic sources. The regulation, integrated with the routing scheme, assures no loss of messages.

This routing scheme is called *convergence routing* (see [15], [17], [16], [14]); it belongs to the class of routing scheme which maintains the following property: *under arbitrary traffic pattern, and with a single buffer per input port, there will be no packet loss due to congestion.*[1]

There are two other notable routing schemes in this class: Baran's "hot-potato" routing [2] and Maxemchuk's deflection routing [10]. In this class of routing schemes packets try to take the most direct path to their respective destinations but may be forced onto alternate routes because of congestion. Convergence routing distinguishes itself by the fact that it builds upon the idea of having a *global distance metric* which underlies the routing and access control. Specifically, it is based on partitioning the network into primary links (which are branches in a spanning tree) and secondary links (or "threads") and then defining a routing scheme in which packets normally follow a path along a logical ring obtained by tracing the perimeter of a spanning tree, but can take short cuts (see Section II for details). Both primary and secondary links guarantee that at each hop the packet is making forward progress towards its destination according to the global distance metric (based on distance along the logical ring).

Moreover, as it will be shown in this paper, convergence routing (and having a global distance metric) allows us to apply (access and flow) control in an orderly and elegant way. Namely, control where a part of the network is affected and participates in the regulation if and only if it is congested, and where all decisions are made locally (within the congested region) using simple two-bit control messages. In addition, the control scheme requires that a node of degree d maintains only d *single-packet sized buffers* and a data structure of size $O(d^2)$. As a result, this is a *scalable solution*—independent of the network size. This scalability constraint implies "fairness regulation" of sources—rather than buffer allocation inside the network.

Manuscript received August 2, 1994; revised January 1996 and January 1998; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Molle. The work of A. Mayer was supported in part by an IBM Graduate Fellowship. This work was previously published in IEEE INFOCOM'95, pp. 891–899, 1995, and as IBM Research Report RC 19686, August 1994.

A. Mayer was with Bell Labs, Lucent Technologies, Murray Hill, NJ 07974. He is now with Bitmo, Inc., San Francisco, CA USA (e-mail: amayer@bitmo.com).

Y. Ofek is with Synchrodyne, Inc., New York, NY 10038 USA (e-mail: ofek@synchrodyne.com).

M. Yung is with CertCo, Inc., New York, NY 10004 USA (e-mail: moti@certco.com).

Publisher Item Identifier S 1063-6692(00)04992-X.

[1]There are other routing techniques, see for example [18], which provide various strategies for optimizing the routing distance and traffic load. Such schemes often require advance knowledge of the global traffic pattern and load in order to avoid excessive congestion and packet loss. One key problem of such an approach is that in high-speed networks the data traffic pattern may change faster than the time it takes to update the global state. A typical example of dynamically changed traffic patterns is the "traffic" implied by world wide web users and distributed computing.

The fairness measures proposed in this work are based on the notion of a *conflict graph* among multiple source-destination sessions. We propose the measure of node *access time* and the measure of *frequency*. We show that the measure of "node access time in our solution is of the order of the *maximal clique size* in the conflict graph to which the node belongs. This measure evaluates the degree of fairness as it guarantees that within this bound any node (at any state and traffic conditions) is able to transmit a predefined quota of data units. If a steady state of the traffic pattern is reached, then it is possible to measure the *frequency* of a node's access, which in this case is inversely linear to the clique size in the conflict graph.

Next we review various notions of fairness. Bursty or asynchronous data traffic has been the "traditional design point" for local area networks (e.g., Token-rings, Ethernet). Bursty traffic access by a node is performed, based on local information, whenever possible and typically without prior negotiation of resources. In such networks the traffic already inside the network has priority over external traffic access into the network. Such a policy (especially in general topology networks) under high utilization, can cause "starvation."

A well known method for fairness is based on token passing, which is a global mutual-exclusion algorithm. Thus, no concurrency in bandwidth allocation is possible, which causes limited utilization of the transmission links. To overcome this problem the next step was the decoupling of access control from the fairness regulation mechanism, in order to enable concurrent access and spatial bandwidth reuse. For example, buffer insertion or slotted rings are mechanisms which enable spatial bandwidth reuse, but suffer from an inherent fairness problem. This fairness problem was solved in the MetaRing architecture [6], [13], in a *global manner*, by the SAT control signal, which operates independently from the media access control.

Global fairness provides "equal access opportunity" for all nodes. However, this may not be viewed as "fair" from the local perspective of some nodes, since it is possible that these nodes can access the network without interfering or conflicting with the access of other nodes. Thus, a stronger fairness is needed, which captures the notion of local regulation of conflicting nodes.

The known *Max-Min fairness definition* as introduced in [8], [9] and discussed in [3] provides a more natural access condition with respect to conflicts in using the network's links. In Max-Min fairness a node can increase its access rate as long as it does not decrease the access rate of a node with an already equal or smaller access rate. However, simply computing *exact* Max-Min rates on a general topology network—under the assumption of no intermediate buffering and minimal communication delay—has been shown to be NP-hard (see [1]).

Nevertheless, the Max-Min definition can be used as a yardstick when comparing the quality of fairness algorithms. For example, the local fairness that was proposed for *ring networks* with spatial bandwidth reuse like the MetaRing (which employs signal of size $O(\log n)$ bits), was demonstrated to approach the Max-Min definition under certain simulated traffic scenarios [4], [5]. In general, local fairness attempts to involve only the conflicting data streams in regulation of traffic as is perfectly done by a Max-Min fairness definition. We note that a close approximation of the Max-Min fairness for *arbitrary traffic scenarios* was recently proposed for a ring network with spatial bandwidth reuse [11], [12]. However, this fairness algorithm is somewhat more complex.

The local fairness algorithm introduced in this work uses the spanning-tree to identify acyclic paths leading back to the source of competing traffic streams. It exploits the fact that routing is done according to a distance metric which is called a *global sense of direction*. The basic idea of backtracking to the sources is somewhat similar to the local fairness on rings [4], but since a spanning tree is used the algorithm operates faster and more efficiently. More specifically, the delay is proportional to the spanning tree's diameter rather than the ring's circumference (i.e., $O(\lg n)$ path length on the average, instead of $O(n)$), and it is more efficient since it only requires $O(d^2)$ space in each node (independent of the size of the network n), and only two bits for control signals instead of $O(\lg n)$ bits in the ring case.

II. Network Model

The network has *arbitrary topology* and all operating links are *full-duplex*. The sources of the network are being regulated by a mechanism using *control signals* of constant (two-bit long) size. These signals use the same physical medium as the data but their transfer via the link is transparent to the data (e.g., these signals can be realized by redundant or unused serial codewords).

For the distributed implementation of the local fairness algorithm, a *controller* is added to the network interface of each node that includes the following components: (i) buffers for storing the different types of control signals; (ii) a timer for fault-detection; and (iii) a finite state machine which is triggered by the receipt of a control signal into its buffer or by a time-out, which sends control signals as part of its state-transition operations.

If node i has packets in its output buffer to send to node j, we say that node i is the *source node* and node j is the *destination node* of a *session* between i and j. Sessions can start and end in a dynamic fashion. For simplicity, it is assumed that a node can have one session at a time.

In the following we sketch the basic elements of the convergence-routing scheme, as proposed in [15], [16].

A. Tree-Embedded Ring

A *spanning tree* is maintained at all times. Links can thus be partitioned into *tree-links* and *thread-links*, which are all links not belonging to the spanning-tree. A virtual *tree-embedded ring* can be obtained by traversing the spanning tree on an Euler-tour; as shown in Fig. 1. The links of the tree-embedded ring are numbered sequentially from 0 to $m-1$. The starting point can be chosen arbitrarily and the numbers are incremented by one each time a node is traversed. The virtual ring should be closed, i.e., link $m-1$ is connected to link 0. We define two types of addresses on this network: physical and virtual. Each node has its own unique identifier (ID) or physical address, denoted by a capital letter A, B, C etc., as shown in Fig. 1. The number associated with each ring link (associated with a switch output port) constitutes a *virtual node* (VN). The virtual node receives the embedding numbering: $\mathrm{VN}_0, \mathrm{VN}_1, \ldots \mathrm{VN}_{m-1}$. A node is as-

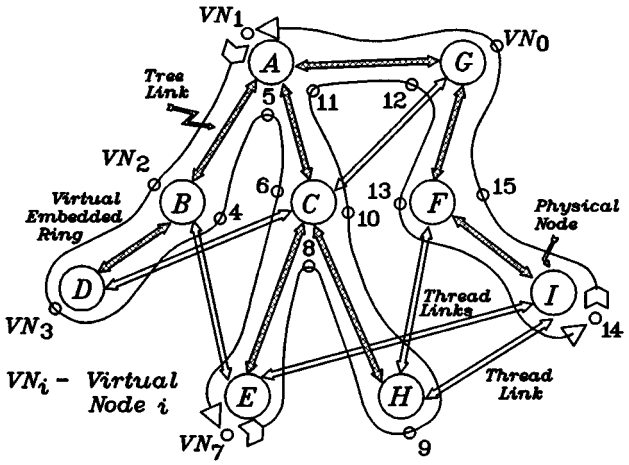

Fig. 1. Tree-embedded ring.

sociated with one or more virtual nodes, for example in Fig. 1, node A has three virtual nodes: VN_1, VN_5 and VN_{11}, and node F has two virtual nodes: VN_{13} and VN_{15}.

B. Convergence Routing with a Global Sense of Direction

The global virtual embedding (i.e., the ring) provides a *global sense of direction*, which is used for self-routing. A packet is routed such that its distance to its destination always decreases according to this global sense of direction and the *global distance metric* it underlines. To find the distance on the ring from virtual node u to virtual node v, simply compute $\mathrm{DIST}(u,v) := v - u(\mathrm{mod}\ m)$. The distance between two physical nodes is measured between the two virtual nodes that are the closest to one another on the virtual embedded ring, e.g., the distance from node D to node F, in Fig. 1, is $10 = 13 - 3(\mathrm{mod}\ 16)$ (since it is smaller than $12 = 15 - 3(\mathrm{mod}\ 16)$), but the distance from node F to node D is $4 = 3 - 15(\mathrm{mod}\ 16)$. The packet enters the virtual embedded ring via the virtual node that is closest to the destination. The destination field in the packet header contains the virtual node that is closest to the source.

The simplest method of routing is by following the virtual ring, which guarantees that the packet reaches its destination. This simple method is, of course, not very efficient. Therefore, the routing mechanism at every intermediate node tries to decrease the distance to the destination as much as possible. We use two methods:

1) *short-cut* to a virtual node on the same node that is closer to the destination, in the global sense of direction, and
2) *jump* on a thread link from a virtual node (on one node) to a virtual node (on a neighbor node) that is closer to the destination.

For example, in Fig. 1, a packet arrives at VN_6 and its destination is VN_{14} (node I). If VN_{10} is idle the packet may short-cut to VN_{10}, and if VN_{12} is idle the packet will jump to it using the thread link from VN_{10} to VN_{12}.

Note that the above embedding yields the following two properties: 1) Every session-path contains its (physical) source- and destination-node only once and 2) Every session-path which: a) has source-node i and b) contains node j has the same first tree-link, given that all better threads are disabled.

The internal flow control in the network is based on a buffer insertion ring principle. Under this principle the traffic already in the ring has priority to continue on the ring. Therefore, in this design the virtual ring embedding forms a buffer insertion ring (BIR).

This novel combination of a ring and a spanning tree yields the following properties:

α) *Local and Lossless Routing*: At every intermediate node of a session packets are forwarded after a minimal delay ("hot-potato" like). Each node only has one (or some suitable constant number) buffer per link. No packet is ever discarded because of congestion on the way.

β) *Local Metric for Forwarding*: For every source-destination pair there is some metric-function. The routing guarantees that every forwarding strictly decreases this metric, i.e., with every hop a packet makes strict progress towards its destination (with respect to the metric).

γ) *Spanning Tree for Control*: For every intermediate node there is a well-defined path to all possible sources of a packet.

Items 1 and 2 suggest a distributed scheduling approach to ensure fair access. Using item 3 obviously suggests using the structure of the spanning tree, to decide the paths on which control signals should be sent. Informally, the fairness control signals are forwarded from points inside the network where there is competition for links backward or upstream towards the traffic sources. While spanning trees have been used for routing many times before, their use for access control is novel.

III. Performance Measures

Here we introduce some basic definitions that will help when discussing performance issues, in particular time complexity.

Definition 3.1: The *conflict graph* $G = (V, E)$ is an undirected graph with V being the set of active sessions. There is an edge between two sessions if their paths conflict, i.e., have a common link. Let C_i^G denote the connected component of G containing session i. Let κ_i^G denote a maximal clique in C_i^G.

It should be clear that the above definition is time-dependent in two ways: 1) the set of sessions (and thus V) changes as new sessions arrive and others become inactive; and 2) since we use convergence routing, two sessions can conflict with each other only for a limited time. Still, we can think of a conflict graph as a snapshot at a given time. In order to express the performance measures in terms of a conflict graph, the relevant graph will be the union of all snapshots during a regulation-cycle (i.e., the period during which all conflicting sources can send a predefined *quota* of data units).

An interval graph (circular-arc graph) is a graph in which the vertices correspond to intervals on a line (circle), and two vertices are adjacent if the corresponding intervals intersect. It is well known that such graphs can be recognized in polynomial time [18]. It is easy to see that the chromatic number of a circular-arc graph is at most twice its maximum clique. We note that, given the network-model, the conflict graphs are a slight generalization of circular-arc graphs. The circle corresponds to

the tree-embedded ring. A session can represent several (nonadjacent) segments instead of a single interval, i.e., an interval can have "holes", since threads allow sessions to skip some of the perimeter. However, conflicts are restricted to the case where intervals in a hole do not interfere with the surrounding intervals. If such a conflict does arise, the surrounding interval (session) is forced to "fill in" the hole. This follows from the fact that a session s_1 which is using a short-cut (or a jump) at a certain network-node has lower priority over a session s_2 following the virtual ring. Hence, if a conflict among two such sessions arises s_1 can no longer use the short-cut and is then routed on the ring. It can readily be verified that the chromatic number of the resulting conflict graphs is always bounded by twice their maximum cliques. Therefore, it is always possible to schedule all sources at least once during $2\kappa_i^G$ by scheduling sessions with the same color at the same time.

In the following we define the performance measures' *access time* and *frequency* in the case of congestion, since when there is no congestion, the access to the network is immediate and free:

Definition 3.2: The *access time* of a node i is defined as the maximum time session i is blocked from accessing the network when it has an active session.

Intuitively, the above definition captures the notion of *delay* fairness.

If the set-up of sessions is static (i.e., G is time-independent or, equivalently, a steady state is reached), then a regulation-algorithm can be thought of producing a periodic schedule. Thus we can define the following:

Definition 3.3: If G is time-independent then the *frequency* of session i (f_i) is the fraction of time its session can access the network.

Intuitively, the above definition captures the notion of *throughput* fairness.

Definition 3.4: If G is time-independent then a schedule is said to be *rigid* if for each session i there exists a starting point s_i such that the session is scheduled on exactly the time units $s_i + r(1/f_i)$, for $r = 0, 1, \ldots$

The access time or frequency of a session should optimally only depend on the behavior of its neighbors in G. This locality-property is embodied in the idea of Max-Min fair frequencies (or rates) as introduced in [8], [9] and discussed in [3], which is considered the traditional way to capture throughput fairness:

Definition 3.5 ([3]): "A vector of rates is *Max-Min fair* if for each i f_i cannot be increased without decreasing f_j for some session j for which $f_j \leq f_i$."

The task of designing regulation-algorithms which implement schedules with exact Max-Min frequencies and small access time is a difficult problem. Algorithms presented in [1] for the special case of interval and circular-arc graphs are inherently centralized. Given that here we focus on *distributed algorithms* with only *constant space* per node, we content ourselves with approximating Max-Min behavior with respect to access time and frequency. The result is basically a two-step coarsening of exact max-min: First coarsening is due to the fact that we do not make use of the slack given by conflicting sessions which are in another, even larger conflict and are hence more restricted. Second, coarsening is a result of allowing dependency on the transitive closure of conflicts rather than just the immediate conflict.

IV. Overview: Algorithm Design and Interfacing with Network Model

The objective of the fairness algorithm design is to regulate the access to the network in such a way that the available bandwidth is efficiently used and every session (i.e., user) has a guaranteed bound on its access time in the case of congestion, and consequently, guaranteed minimum bandwidth. The algorithm is triggered only if a node cannot access the network for some predefined period of time. Otherwise, the algorithm is not active and no control messages are sent. In other words, the default state of the algorithm is *inactive*, which has three consequences:

a) nodes can transmit in an unlimited fashion and with no delay when there are no conflicts;
b) in the case of a failure, the algorithm can be stabilized by simply stopping the execution of the algorithm and starting afresh (i.e., self-stabilizing by a global restart when, for example, memory is corrupted);
c) whenever there is more capacity in the network the fairness algorithm will be activated less frequently, and consequently, the expected access time gets shorter.

The main idea of the solution, when the algorithm is active, involves "waves" of back-pressure traveling back and forth between each bottleneck tree link and the responsible traffic sources. A node finding itself in a bottleneck (because it wants to share an outgoing link with traffic arriving from "upstream" along the logical ring) sends a *regulate* control message to those sources in the upstream direction telling them they must stop after sending another quota of data traffic. The control message is sent along the spanning tree, so we know it will reach only upstream (i.e., competing) sources with respect to the bottleneck link. Eventually, the control message reaches all of the competing sources, those sources use up their quota, and the initiating node (at the bottleneck link) is able to send its own quota. Then this node sends an *unregulate* control message in the same fashion as it had sent the regulate message. This allows the competing sources to resume transmission, soon followed by another *regulate* control message from the original node if the bottleneck returns.

The solution optimizes this basic approach as follows: The protocol sends a control message only on those links in the spanning tree that lead to the sources of competing sessions. Also, it filters out duplicate control messages by recognizing that one *regulate* message is sufficient to control the flow of an upstream source and that all bottlenecks must agree on the *unregulated* decision before that flow control can be turned off.

From a local perspective, a node basically "watches" its neighborhood and regulates the sessions passing through it and its own session(s) according to its local view. The goal of the regulation is to divide the bandwidth fairly among competing sessions. The means by which this is achieved is to restrict the use by each session of bottleneck links to a predefined *quota* of data units. The size Q of a quota is a parameter. The allocation of quotas (i.e., when to give a session a new quota) is essentially a scheduling problem. The algorithmic problem,

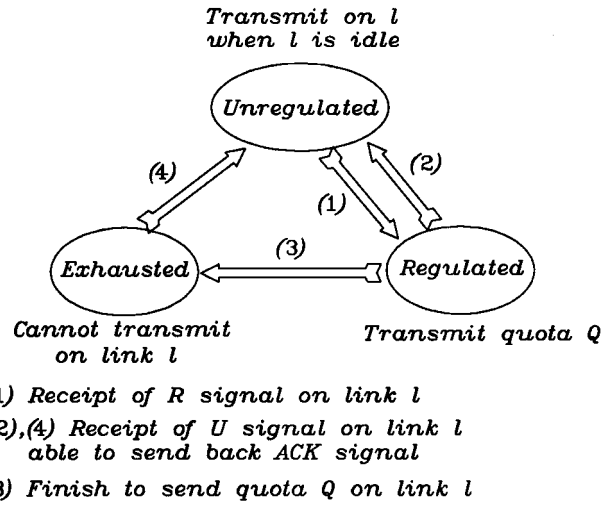

Fig. 2. Link l modes.

therefore, is to produce a scheduling strategy which achieves good use of the available bandwidth and a short bound on the access time for every session.

A. Overview of the Algorithm

1) Link Modes: The protocol maintains a mode for each link (both for a tree link or a thread). This mode is used to control access of adjacent sources that wish to use this link as the first hop for their traffic.

Tree-Link modes:

There are three modes for each directed tree-link l: a) *unregulated*; b) *regulated*; and c) *exhausted*. These modes regulate the access of a node whose session uses l as its first link. In the first mode the node can send freely through l, in the second mode it can use l for sending one more quota only, and in the third mode the node can no longer use link l for its session.

These modes are managed by the three control signals R, U and ACK ($R =$ regulate access, $U =$ unregulated access, and $\mathrm{ACK} =$ acknowledge the unregulation).

Let l ($\bar{l}$) be the directed link from node i to j (j to i) and let mode_l ($\mathrm{mode}_{\bar{l}}$) be its mode. A local regulation-cycle looks as follows: Assume that mode_l is initially *unregulated*. If the signal R is sent on link $\bar{l}$, then mode_l is set to *regulated* and if node i uses l for one more quota then mode_l will become *exhausted*. If subsequently a U-signal is sent on $\bar{l}$ and an ACK-signal on l, then mode_l will become *unregulated* once again. Fig. 2 illustrates such a local cycle in a self-explanatory way.

Thread-Link modes:

There are two modes for each directed thread-link l: a) *enabled* and b) *disabled*. These modes, unlike the modes for tree-links, regulate *all* traffic through l: *enabled* mode allows traffic through l and *disabled* mode disallows traffic from *every* source. A thread l moves from *enabled* to *disabled* whenever an R-signal would be sent on l if l were a tree-link and goes back to *enabled* when the corresponding U-signal would be sent.

2) Local Transmission Cycle: Nodes perform local transmission cycles of sending one *quota* of data. The choice of the quota size involves two issues: a) the packet size and b) the propagation delay. We assume that the packet size is fixed and that the number of packets in a quota is chosen such that the time needed to transmit one quota is equal to or larger than the network round trip delay. A node starts a transmission cycle whenever it (locally) receives enough data to transmit. Hence the start of a cycle is completely asynchronous with the rest of the network. The cycle completes when the node finishes sending one *quota* of data. Whenever node i wants to transmit a quota, i checks whether the corresponding tree-link (which can serve as the first hop for its session) is not *exhausted* (e.g., caused by the transmission of a previous quota). If this check is positive, i knows that it can send at least one quota and will do so. Note that i can also use any enabled incident *thread* as the first link. However, the mode of the corresponding *tree* link must still be non-exhausted, so that i can transmit one full quota, even if the thread becomes disabled in the middle of the quota. Node i also sends R-signals on all incoming tree-links that lead towards sources of competing sessions. In addition, node i disables incident threads that lie on a path towards the sources of competing sessions. Note that these decisions are made *locally* at i and are based on the parameters of the control signals, the spanning-tree structure and the global sense of direction, i.e., the tree-embedded ring. After transmitting one quota, i sends U-signals on all those incident tree-links, on which R-signals were sent at the beginning of the quota and enables the disabled threads. The details are discussed in Section V.

Forwarding and combining R- and U-Signals:

We now discuss how a node should forward incoming control signals. We can think of every node implementing a function, which has the incoming signals as its input and produces the outgoing signals as its output. Every function that guarantees that downstream sources (at which these signals originate) can effectively stop their competing upstream-sources is a viable candidate. Hence the actual forwarding decision has two components: a) On which outgoing links should an incoming signal be forwarded, based on the actual traffic pattern and the information obtained by the spanning-tree ad the routing scheme?; and b) How should several signals be combined, based on the local state of a node (i.e., based on what kind of signals has the node received and forwarded at an earlier point in time)?

If a node receives an R-signal on some incident tree-link l then this indicates that all the sources whose packets flow on l in the opposite direction should be regulated. Consequently, the R-signal should be forwarded on tree-links such that the R-signal reaches all those sources. Again this local decision is made based on the spanning-tree and the tree-embedded ring and is discussed in detail in Section V. For now just assume that there is a function *Candidate-Links*(l) at each node i which, given the current traffic-pattern, uses a d-by-d matrix to output the set of tree-links on which an R-signal arriving on incident link l needs to be forwarded.

If node i receives two or more incoming R-signals (on distinct links), then it is possible that these signals should be forwarded on the same outgoing link according to the *Candidate-Links* function. In this case, the signals should be merged. We say that an R-signal is *active* for link l if i has sent this signal on link l, but has not yet followed it up by a U-signal. We say that an R-signal is *relevant* for link l, if it needs to be forwarded on l according to *Candidate-Links* function. Note that in the case

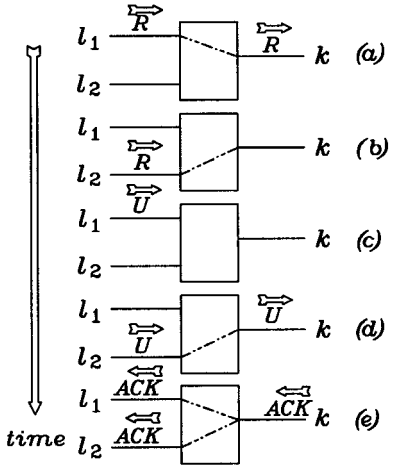

Fig. 3. Forwarding: example 1.

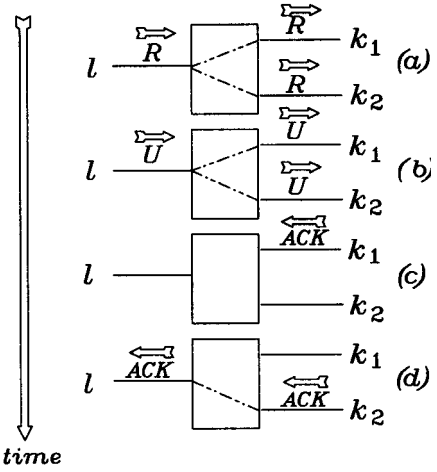

Fig. 4. Forwarding: example 2.

of merging signal, a signal can be relevant for l without having been actually forwarded on l.

The choice of the combine-function:

Combine-Function: Node i allows at most one active R-signal on each incident link l. Hence node i *merges* R-signals, which are supposed to be forwarded on links where there is already an active signal. But node i does remember these signals as being relevant for link l.

As noted earlier, a U-signal arriving on a link k essentially undoes the effect of having received an R-signal on k. Hence, node i tries to forward the U-signal on those links on which it forwarded the R-signal. However, node i only forwards a U-signal on such a link l if there are currently no *relevant* R-signal for l.

Figs. 3 and 4 illustrate the complete forwarding function. Assume that packet traffic is such that an R-signal arriving on either l_1 and l_2 should be forwarded on link k. Hence in Fig. 3(a) the incoming R is immediately forwarded. Since there is already an active R-signal on link k, the arriving signal is not forwarded in Fig. 3(b). In Fig. 3(c), the U-signal is not forwarded, since there is still a relevant R-signal for k (the one on l_2). Consequently, in Fig. 3(d) the U-signal arriving on l_2 is forwarded to k. Fig. 4(a), (b) show the case in which an incoming signal is split into several outgoing signals.

Sending and forwarding an ACK-Signal:

The ACK-signal indicates the end of the current regulation cycle and thus allows nodes to start a new cycle and a new quota.

A node which received but did not forward an R-signal (i.e., a *leaf* in the structure) will, on the receipt of the corresponding U-signal, immediately send an ACK-signal back on the link it previously received the R- and U-signal. The forwarding of an ACK-signal is essentially the reverse of the forwarding of the U-signal. An ACK can be received by some node i on link l, whenever i previously forwarded (from some link k) an R- and a U-signal on l. Node i forwards the ACK-signal on link k as soon as it received an ACK on *every* link on which it forwarded the R- and U-signals it received on link k_1. Fig. 4(c), (d) illustrates this as in Fig. 4(c), the incoming ACK on k_1 is delayed until the other ACK on link k_2 is received, as shown in Fig. 4(d).

B. Interfacing with the Network Model

We need the following definitions in order to describe how to compute the candidate links:

Definition 4.1: The *pseudosource* of a packet is the head-node of the last thread-link this packet used or its real source if it has not used any threads at all.

Definition 4.2: The *induced subtree* of node i and of one of its incident links l is the subtree of spanning-tree consisting of all nodes reachable from i using only l on the first hop.

We now define the condition on which a node i will forward an isolated R-signal received on link l to link k. Let σ denote a session, then $k \in Candidate\text{-}Links(l)$ if the following four conditions are true (see property γ of Section II):

1) σ leaves i on tree-link l (i.e., this session σ may be the cause of conflict that has triggered the incoming R-signal).
2) σ arrives at i on tree-link k (i.e., the R-signal should forwarded towards σ's (pseudo-) source).
3) σ's pseudosource is in the subtree induced by i and k (i.e., the R-signal should be forwarded to σ's (pseudo-) source on the shortest path on the tree).
4) $k \neq l$ (i.e., the R-signal cannot be forwarded on the same link it has been received on).

Note that the above allows for the possibility that the packets of session σ are forwarded from k into some other link m and arrive later back at node i and only then are forwarded into l, i.e., session σ does not short-cut at node i (due to other congestion). But the control signal always short-cut towards the pseudosource on the induced subtree. Every node has enough information to evaluate the above condition, i.e., has access to the pseudosource field of a packet and to the information stored in its switch. Subcondition 4 guarantees the acyclicity of the signal forwarding operation.

The signal forwarding function can be implemented without node i explicitly knowing about the existence of any session. It is sufficient for node i to observe locally how incoming packets are routed to outgoing links.

The condition on which node i needs to send an R-signal generated locally (as a result of its own session) is very similar to the above. If i's current session uses link l as its first hop,

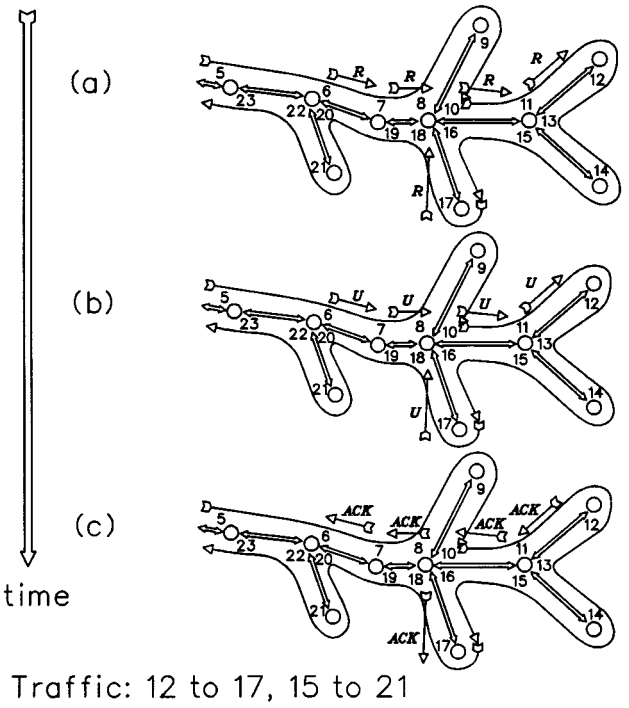

Fig. 5. Example: network signaling.

then the locally generated R-signal needs to be forwarded as if an R-signal arrived on link l. The only difference is that we need to leave out Subcondition 4.

Fig. 5 illustrates how the algorithm operates on an actual network. There is an active session s_1 from node 12 to node 17 and another session s_2 from node 15 to node 21. The following two paths are possible for s_1: 12–13–14–15–16–17 or 12–13–15–16–17 and for s_2 we have 15–16–17–18–19–20–21 and 15–16–18–19–20–21. Now if node 20 would like to start a session with destination 21 it has to compete with s_2 (on link 20–21). Hence it sends an R to 7 as s_2's source is in that induced subtree. According to the forward function, this R-signal will be forwarded on the path 7–8–10–11–12 as indicated in Fig. 5(a). Now if also node 17 decides to start a session to 18 then it has to compete with s_1. Hence it will send an R-signal to 18 as s_1's source is in that induced subtree. Since there is already an active R on 10–11, this R-signal is not further forwarded (by the combining operation). Given the situation in Fig. 5(a), s_1 and s_2 can only transmit one quota before their first tree-links become exhausted. After that, nodes 20 and 17 can transmit their quota. Consequently, they will send U-signals on links 6–7 and 17–18, respectively. The further forwarding of the U-signals follows that of the R-signals, as depicted in Fig. 5(b). Node 12 is a leaf of the R/U structure and hence will send an ACK-signal back on link 12–11 as soon as it receives the U-signal. The propagation of the ACK-signal is straightforward in this example, as shown in Fig. 5(c).

It should be mentioned here that the above is a *simple* scenario. It is not hard to come up with scenarios in which nodes send new R-signals when the structure is in the form of Fig. 5(b) or (c) and thus stop any further forwarding of U- or ACK-signals. Hence the phases of sending R-, U-, ACK-signals are not always as neatly separated as in Fig. 5.

V. Formal Description of Algorithm

A. Control Signals and Interface to the Model

Control signals:

1) "R" marks the local start of a regulation-cycle on the tree-link it is sent ("R"egulated);
2) "U" marks the local end of regulation-cycle on the tree-link it is sent ("U"nregulated);
3) "ACK" marks the global end of a regulation cycle.

Interface:

1) $r(j)$: receipt of an R-signal on incident tree-link j;
2) $u(j)$: receipt of a U-signal on incident tree-link j;
3) $\mathrm{ack}(j)$: receipt of an ACK-signal on incident tree-link j;
4) $\mathrm{start}(j)$: detects a new session using incoming link j;
5) $\mathrm{waiting/active-session}(j)$: own session with j as its first tree-link is waiting/active;
6) $\mathrm{upstream-busy}(j)$: active session with "upstream" source uses link j;
7) $\mathrm{send}(S, j)$: sends control signal S on tree-link j.

B. Data Structure

In the variable *count*, every node i keeps track of how much of the current quota it could already send. In the variables mode_l $(l \in N(i))$ every node i remembers the current status of its incident tree-link l which either is *unregulated*, *regulated* with one quota (of size Q) or *exhausted*. If node i has an active session which uses outgoing tree-link l, then the variable lm_l contains the last control signal received on link l since mode_l changed its value for the last time. The matrix *message-seq*$_{lk}$ at node i ($l \in N(i) \cup \{i\}$ and $k \in N(i)$) records R- and U-signals which arrive on an incident link l and which need to be forwarded on another incident link k. The possible values of this sequence are $(\,)$, (R), (U), and (RU). $(\,)$ indicates that currently there are no signal forwarded from l to k. (R) indicates that there is an R-signal which arrived on link l and is relevant for link k. The value (U) indicates that a U-signal has been forwarded from link l to link k, which has not yet been followed by an ACK-signal; and (RU) indicates that a U-signal has arrived on link l, but there is an R-signal on some other link which is relevant for k and hence U is not yet forwarded. The row *message-seq*$_{ik}$ $(k \in N(i))$ is used to record sending of R/U-signals which originate on node i itself. The variables *last-sent*$_j$ and *last-rec*$_j$ indicate the last control signal sent or received on link j.

C. The Algorithm

In this section we present a pseudocode formulation of the algorithm. Every node runs the same code consisting of a main (infinite) loop (Lines L1–L14) which continuously checks for incoming control signals (Lines L1–L3) and the possibility to advance its own current session (if any) (Lines L5–L12).

Thus, if at the beginning of a local cycle (i.e., count $= 0$) there is an active session, node i checks whether it is guaranteed to transmit at least one quota of this session (L5). Given property (β) of the routing scheme, there is exactly one outgoing link whose mode needs to be checked (denoted l on Line L5). In addition to this link we introduce a "virtual" link i that starts and

ends at node i. Node i wishing to start transmitting will simulate receiving an R-signal on this link (Line L6). This signal will be forwarded as if it arrived on link l. Its use is (a) to generate the necessary R-signals for i to restrict upstream-traffic and (b) in keeping i restricted until after it received the corresponding ACK-signal. Now if links i and l are not exhausted then node i can transmit one quota. First the variable *count* is set to 1 to indicate that the transmission of the current session has begun (Line L7). Once the above decision has been made, node i transmits one quota of packets. The only check before inserting a packet into the network is for upstream traffic (Line L8), following the buffer-insertion-like access-control strategy of giving priority to traffic already in the network. Once the transmission of the quota is completed, the mode of links i and l are updated (Line L12/13). Again, the same needs to be done as if a U-signal was received on link i (Line L14).

The main loop calls a few subroutines which are explained further below.

Main Loop at Node i:
LOOP (forever)
L1 IF $r(j)$ THEN CALL receive-R(j,j)
L2 ELSIF $u(j)$ THEN CALL receive-U(j)
L3 ELSIF $ack(j)$ THEN CALL receive-ACK(j)
L4 ELSIF start(j) THEN
CALL receive-START(j);
L5 IF (count $= 0$) and (waiting-session(l)) and
($\mathrm{mode}_l \neq$ exhausted) and
($\mathrm{mode}_i \neq$ exhausted) THEN
L6 CALL receive-R(l,i);
L7 count $:= 1$
L8 ELSIF (count $\neq 0$) and
(not upstream-busy(l)) THEN
L9 send packet;
L10 count $:=$ count $+ 1 \bmod(Q)$;
L11 IF count $= 0$ THEN
L12 $\mathrm{mode}_i :=$ exhausted;
L13 IF $\mathrm{lm}_l = R$ THEN
$mode_l$, $\mathrm{lm}_l :=$ regulated, none
ELSIF $\mathrm{lm}_l = ACK$ THEN
$mode_l$, $\mathrm{lm}_l :=$ unregulated, none
ELSIF $mode_l = regulated$ THEN
$mode_l :=$ exhausted
L14 CALL receive-U(i);
END LOOP

The subroutine receive-R at node i handles incoming R-signals. The parameters indicate which incoming link needs to be regulated (l) and the source of the R-signal (s). However, if the R-signal is generated at node i itself, $s = i$, l is the first link of i's session, and we always have $l = s$. If the last message on link s was an R-signal, then the mode of s remains unchanged (since no ACK was received in the meantime). Otherwise the mode will be set to "regulate" or if this link is currently used by i, the receipt of R is remembered in the variable lm. (Lines R1–R3). If it is necessary to forward the incoming R-signal on some outgoing link k, then this is remembered in the variable *message-seq*$_{sk}$ by setting it to (R) (Line R6). If an R-signal has been already sent on k (forwarded from some other incoming R) then nothing else needs to be done. If a U-signal was sent last, then the sending of R is delayed until the outstanding ACK-signal is received, which corresponds to this U-signal. Otherwise, an R-signal needs to be sent on k and this will be remembered in the variable *last-sent*$_k$ (Line R7).

Subroutine receive-R(l,s) **at Node** i:
R1 IF $\text{last-rec}_s \neq U$ THEN
R2 IF $\neg$active-session(s) THEN
$\mathrm{mode}_s :=$ regulated
R3 ELSE $\mathrm{lm}_s := R$;
R4 $\text{last-rec}_s := R$;
R5 FOR ($k \in$ *Candidate-Links*(l)) DO
R6 $\text{message-seq}_{sk} := (R)$;
R7 IF ($\text{last-sent}_k \neq R$) and ($\text{last-sent}_k \neq U$) THEN
send(R,k); $\text{last-sent}_k := R$

The subroutine receive-START(l) is called at node i when there is a new session using the incoming link l. It then must be checked which previously received R-signals needed to be forwarded on link l (Lines S1 and S3).

Subroutine receive-START(l) **at Node** i:
S1 IF $|\{k/\text{last} - \text{rec}_k = R \wedge$
$l \in$ *Candidate-Links*$(k)\}| \geq 1$ THEN
S2 IF $\text{last-sent}_l \neq R$ THEN
send(R,l); $\text{last-sent}_l := R$;
S3 FOR all $\{k/\text{last} - \text{rec}_k = R \wedge$
$l \in$ *Candidate-Links*$(k)\}$ DO
S4 $\text{message-seq}_{kl} := R$

The subroutine receive-U at node i handles incoming U-signals on link s (either i itself or some incoming link). If the signal does not need to be forwarded at all (i.e., node i is a leaf of the current R sub-tree on which the R signal has been propagated), then i will generate an ACK-signal and send it back on link s (Lines G1–G3). The mode of s will now become unregulated again (unless s is used by a session of i, in which case the sending of the ACK is remembered in lm).

Otherwise the receipt of U is remembered by updating the variable *last-rec*$_s$ (and lm). If the preceding R-signal on s has been forwarded on some outgoing links (as remembered in *message-seq*) then the following needs to be done (for each such link k): If there is at least one message-sequence with destination k which does not contain a U-signal, then the current U needs to be delayed (cf the *combine-function*). In that case the current U is simply appended to the sequence (Line G7). Otherwise, a U-signal is sent on link k and the R is deleted from all sequences with destination k (Lines G9–G14).

Subroutine receive-U(s) **at Node** i:
G1 IF $\{k/message\text{-}seq_{sk} = (R)\} = \emptyset$ THEN
G2 send(ACK, s); last-rec$_s$:= none;
G3 IF¬active-session(s) THEN
mode$_s$:= unregulated
ELSE lm$_s$:= ACK;
G4 ELSE
G5 last-rec$_s$:= U;
G6 IF active-session(s) THEN lm$_s$:= U;
G7 FOR k : message-seq$_{sk} = (R)$ DO
G8 IF $|\{l/\ l \neq s \wedge$ message-seq$_{lk} = (R)\}| \geq 1$ THEN
G9 message-seq$_{sk}$:= (RU);
G10 ELSE
G11 send(U, k);
G12 last-sent$_k$:= U;
G13 FOR $\{l/message\text{-}seq_{lk} = (R)\}$ DO
G14 message-seq$_{lk}$:= (U)

The subroutine receive-ACK at node i handles incoming ACK-signals on link l. The receipt of this signal neutralizes the previous sending (forwarding) of a U-signal in the opposite direction (Lines A1 and A3). It needs to be forwarded on those links from which an incoming U-signal was forwarded into l, granted that this U-signal does not "support" other outgoing U-signals anymore (Line A4–A5). If it is forwarded on a link, the mode of this link will become unregulated again. Lines A8, A9 take care of delayed R-signals, which due to asynchrony, arrived after the U-signal on link k, but were not forwarded, because the ACK on link l had not yet arrived (see Line R7 in receive-R).

Subroutine receive-ACK(l) **at Node** i:
A1 last-sent$_l$:= none;
A2 FOR k : (message-seq$_{kl} = (U)$) DO
A3 message-seq$_{kl}$:= ( );
A4 IF ($\forall m$ message-seq$_{km} = (\,)$) THEN
A5 send(ACK, k);
A6 IF ¬active-session(k) THEN
$mode_k$:= unregulated
ELSE lm$_k$:= ACK;
A7 last-rec$_k$:= none;
A8 FOR k : (message-seq$_{kl} = (R)$) DO
A9 send(R, l); last-sent$_l$:= R

VI. Results of the Algorithm

Next it is shown that the main result of the algorithm guarantees access time for each node i to be bounded by four times the size of κ_i^G. In the following theorem the propagation-delay of the control signals is ignored, but we note that the algorithm operates correctly under any assumption of (asynchronous) finite propagation-delay.

Theorem 6.1: The access time of each node i is bounded by $4|\kappa_i^G|$.

Proof: It follows directly from the algorithm (the conditions preventing transmission) that if node i has an active session and is delayed transmitting then it must be that at least one of the following two conditions holds: 1) session(s) from upstream prevent node i from advancing its own session (Line L8); or 2) the outgoing link for i's session is *exhausted* (Line L5).

Assume first that only condition 1) holds: In this case node i already executed Line L6 and thus has already generated all necessary R-signals. It remains to be shown that these signals reach all nodes whose session uses link l and thus contribute to the condition *upstream-busy*(l). Note that all those sessions are neighbors of i in G and hence their number is bounded by $|\kappa_i^G|$. In the following we show how those signals reach the closest pseudosource of these sessions. Once this pseudosource is reached, the thread is disabled and thus the session is forced to use tree-links only from that node on to its destination. This process is repeated until the session uses tree-links only. Since we assume that the delay of the control signals is negligible, it is sufficient to show how the control signal reaches any source (independent of being a real or pseudosource). From the function *Forward* and the *Combine-function*, and by (β) and (γ), we can conclude that any R-signal generated by node i is forwarded immediately (see Line R5) at any intermediate node on its way to the source of these sessions, and that each time an R-signal is forwarded, its distance to the closest source of the relevant session is decreased by at least 1. Note that since R-signals always short-cut a detour, it is never required that a chain of R-signals wrap around a leaf of the spanning tree and thus Subcondition 2 in the definition of *Forward* does not interfere with the above argument. By (i.) we are guaranteed that the R-signal always reaches the source of a session on the first link of the session's path, and thus succeeds at regulating the session's input by changing the mode of this link (see Line L13). By (ii.), we are also guaranteed that even if a node changes its destination (by switching sessions) during this forwarding process of the R-signal, the argument remains valid, i.e., a node cannot obtain another quota by switching its session. Since the granularity at which a node checks its incoming messages is a quota, it might take two quotas until it is stopped (i.e., the mode of its outgoing link becomes *exhausted*). Thus the time-bound follows for this case.

If condition (b) holds, then node i is regulated itself by sources of sessions "downstream". By the acyclicity of the flow of control signals (γ) we are guaranteed that node i is not one of these sources. From the definition of the *Forward*-function it follows that all these sessions are in C_i^G. Condition (b) does not hold on these sources (otherwise they could not send an R-signal themselves). Thus we can apply the above argument for condition (a) and conclude that all these sources will be able to transmit one quota by the time-bound. Consequently, each of these sources will send a U-signal after sending this one quota. U-signal are forwarded in the same fashion as their preceding R-signals (using the table *message-seq* at each intermediate node). The delay at each node i should be considered in forwarding this U-signal: The *combine-function* tells that a U-signal can be delayed if the corresponding outgoing R-signal (on link k) is still "supported" by another incoming R-signal which has not yet been followed up by a U (i.e., there is a link l such that message-seq$_{lk} = (R)$; see Line G8). But note that any source s which is responsible for the R on link l can send at most one quota as long as the R is not forwarded.

This is true since if s has sent one quota and thus has sent a U, the mode of its incoming (virtual) link will remain "exhausted" until s receives an ACK-signal. This ACK-signal must pass node i first. Thus it can be concluded for each intermediate node i of the R-path that after $2|\kappa_i^G|$ time a U-signal has passed. No new R-signal can be sent on links of the U-path until the ACK-signal is received in the opposite direction (see Line R7). Furthermore, any ACK-signal which is created at a leaf propagates up by the same time-bound, which implies the bound claimed.

Thus, after this time, node i can only be delayed by upstream sessions which have not affected any downstream sessions. Using the argument above, the time-bound follows in the general case. ■

Assuming steady state traffic conditions, the above theorem can be used to deduce that:

Corollary 6.2: If sessions are static, then every node i gets a frequency of at least $1/2|\kappa_i^G|$ and the algorithm produces a rigid schedule.

This statement implies that applications will be provided with minimum bandwidth guarantees, which is an essential QoS parameter for many multimedia applications.

VII. Fault-Tolerance

For the efficient operation of the regulation-algorithm no error recovery protocol is performed on the control signals. Only error detecting is performed on the signal, and as a result, it is only possible to determine reliably that a control signal has arrived, but not that it has been sent. Therefore, a lost signal will not be directly recovered and the regulation-algorithm will have to take special care in this event. In order to achieve the desired fault-tolerance a timer is used for each link at each node. This timer will use some bound b which is linear in the number of nodes (see the main theorem) to indicate an exceptional event (i.e., message loss, partitioning of the spanning tree or rejoining of separate partitions).

The following additions will make the algorithm robust against such events. Even more so, the algorithm is robust against a time-out which occurs too early.

(f-1) If a node times out while it is waiting for upstream-busy to become false, then it will resend all necessary R-signals.

(f-2) If a node times out while it is waiting for a U- or ACK-signal (since its adjacent link is *exhausted*) out then it will simulate the receipt of such a signal.

(f-3) If a node receives two consecutive R- or U-signals on one link, the same algorithm as for receiving the first R will be executed but without changing the mode of the link.

It is not hard to see that the modified algorithm will always transit back into a "legal state" after at most $O(b)$ time-units after the last exceptional event (from this state on the bounds of the original algorithm hold).

Furthermore, initialization of the system is simple since the default state is when this access-regulation algorithm is not active. This implies that in order to start or restart the system it is sufficient to stop the algorithm execution on all nodes and get the entire network into a non-active state. This can be done by an upper layer network reset (concurrent broadcast) protocol. This implies self-stabilization of the algorithm.

VIII. Discussion

In this work, motivated by high-speed networking environments, we have put forth a novel congestion-driven technique for regulating the access, by data applications, to an arbitrary topology network. In the following, two QoS attributes of this algorithm are discussed: 1) how fairness ensures minimum bandwidth guarantee; and 2) how this algorithm avoids traffic shaping whenever possible.

A. Fairness and QoS

As noted, since bandwidth reservation is typically not made for data applications, the only way to provide *minimum bandwidth guarantees* is by having some sort of fairness mechanism. There are many works on fairness for ring networks, however, there are almost no fairness results on general topology networks. The context of this work is MetaNet's *convergence routing*, which is a *loss-free and deadlock-free routing technique*. Minimum bandwidth guarantee together with loss-free and deadlock-free routing are the desired quality-of-service (QoS) attributes for many data applications.

One of the common ways to achieve loss-free routing is by using back-pressure to the source. It is done, for example, in current networks like fiber channel and gigabit Ethernet. Such an approach suffers from head-of-the-line blocking and possible deadlock. Furthermore, there are no known methods for combining back-pressure with fairness. Consequently, there is no possibility to provide *minimum bandwidth guarantees* for data applications over such networks.

The loss-free and deadlock-free properties of the MetaNet have been achieved at the expense of not maintaining first-in-first-out (FIFO) order routing. However, with the MetaNet's convergence routing (unlike deflection methods) it is possible to implement a semi-FIFO reliable transmission protocol [7]. Semi-FIFO means that FIFO is only partially maintained. It is done by restricting the routing of the transport protocol control (or supervisory) messages to travel, from source to destination, only along the virtual ring *without short-cuts or jumps*, while maintaining FIFO order. Data packets are still routed without FIFO, and reordering is done at the receiver interface, with small additional complexity. Given a bound on access time, one can better manage the end-to-end window of the semi-FIFO mechanism.

B. Distributed Computing While Minimizing Traffic Shaping

The algorithm presented in this paper is useful for distributed computing over general networks. Computing, in general, is *unpredictable* in *time* and *space*, e.g., transaction processing in large distributed data-base systems. As a result, in distributed/parallel environments also the communication (traffic) pattern is *unpredictable*. There are two consequences to this assumption: 1) the traffic is bursty; and 2) for achieving, whenever possible, near *linear speed-up* for parallel computing, the provided network latency should be as low as possible.

(Near *linear speed-up* means that n machines can provide, for a given task, almost n times higher computation throughput.)

For achieving this the network should not change or shape the *traffic source profile*. This means that whenever possible the network should allow the user's application to have *immediate access* to the network and with *full link utilization*. This is what our local and congestion-driven algorithm provides, since it is activated only when congestion occurs and is limited to the subnetwork containing conflicting traffic streams. This implies that when there is no congestion, nodes can access the network immediately and freely. Furthermore, whenever there is more capacity in the network, the congestion-driven fairness algorithm is activated less frequently, and consequently, the fraction of time with no traffic shaping gets larger.

REFERENCES

[1] A. Bar-Noy, A. Mayer, B. Schieber, and M. Sudan, "Guaranteeing fair service to persistent dependent tasks," presented at the Proc. ACM-SIAM Symp. Discrete Algorithms, 1995.

[2] P. Baran, "On distributed communication networks," *IEEE Trans. Commun. Syst.*, vol. CS-12, pp. 1–9, Mar. 1964.

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

[4] J. Chen, I. Cidon, and Y. Ofek, "A local fairness algorithm for the MetaRing and its performance study," in *GLOBECOM'92*, pp. 1635–1641.

[5] ——, "A local fairness algorithm for gigabit LANs/MANs with spatial reuse," *IEEE J. Select. Areas Commun.*, vol. 11, pp. 1183–1192, Oct. 1993.

[6] I. Cidon and Y. Ofek, "MetaRing—A full-duplex ring with fairness and spatial reuse," *IEEE Trans. Commun.*, vol. COM-41, pp. 110–120, Jan. 1993.

[7] R. Cohen and Y. Ofek, "Reliable transmission of data over a semi-FIFO routing layer," *Comput. Networks and ISDN Syst.*, vol. 27, no. 12, pp. 1633–1649, 1995.

[8] H. Hayden, "Voice Flow Control in Integrated Packet Networks," M.I.T. Laboratory for Information and Decision Systems, Cambridge, MA, LIDS Rep. TH-1152, 1981.

[9] J. M. Jaffe, "Bottleneck flow control," *IEEE Trans. Commun.*, vol. COM-29, pp. 954–962, July 1981.

[10] N. F. Maxemchuk, "Routing in the Manhattan street network," *IEEE Trans. Commun.*, vol. COM-35, pp. 503–512, May 1987.

[11] A. Mayer, Y. Ofek, and M. Yung, "Approximating max-min fair rates via distributed local scheduling with partial information," IBM Research Center, Yorktown Heights, NY, Rep. RC 19931 (1995), 1996.

[12] A. Mayer, Y. Ofek, and M. Yung, "Approximating max-min fair rates via distributed local scheduling with partial information," presented at the IEEE INFOCOM, 1996.

[13] Y. Ofek, "Overview of the MetaRing architecture," *Comput. Networks and ISDN Syst.*, vol. 26, no. 6–8, pp. 817–830, Mar. 1994.

[14] Y. Ofek, B. Yener, and M. Yung, "Concurrent asynchronous broadcast on the metanet," *IEEE Trans. Comput.*, vol. C-46, pp. 737–748, July 1997.

[15] Y. Ofek and M. Yung, "Principles for high speed network control: Lossless-ness and deadlock-freeness, self-routing and a single buffer per link," in *9th Annu. ACM Symp. Principles of Distributed Computing (PODC)*, Aug. 1990, pp. 161–175.

[16] ——, "Routing and flow control on the MetaNet: An overview," *Comput. Networks and ISDN Syst.*, vol. 26, no. 6–8, pp. 859–872, Mar. 1994.

[17] ——, "METANET: Principles of an arbitrary topology LAN," *IEEE/ACM Trans. Networking*, vol. 3, pp. 169–180, Apr. 1995.

[18] H. Rudin, "On routing and "delta routing": A taxonomy and performance comparison of techniques for packet-switched networks," *IEEE Trans. Commun.*, vol. COM-24, pp. 43–59, Jan. 1976.

[19] A. Tucker, "Matrix characterizations of circular-arc graphs," *Pacific J. Math.*, vol. 39, pp. 535–545, 1971.

Alain Mayer received the M.Sc. degree in computer science from Brown University, Providence, Rhode Island, and the Ph.D. degree in computer science from Columbia University, New York.

He is a recognized scientific expert in the areas of Internet technology and computer security. He is currently Chief Scientist at Bitmo, Inc., San Francisco, CA, a wireless ASP start-up. Prior to joining Bitmo, he was a Research Scientist at Bell Labs, Lucent Technologies, Murray Hill, New Jersey. While there he pioneered and led projects on firewall management, Web privacy, security, and secure Web scripting tools that have been incorporated into such ubiquitous software products as Netscape Communicator 5.0.

Dr. Mayer has been the recipient of both the USENIX Electronic Commerce and USENIX Security Best Paper awards.

Yoram Ofek (S'86–M'87) received the B.Sc. degree in electrical engineering from the Technion-Israel Institute of Technology in 1979, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Illinois, Urbana, in 1985 and 1987, respectively.

He is President and CEO of Synchrodyne Networks, Inc. From 1979 to 1982 he was a Research Engineer at RAFAEL, Haifa, Israel. From 1983 to 1984 he was a t Fermi National Accelerator Laboratory, Batavia, Illinois, and from 1984 to 1986 he was with Gould Electronics, Urbana, Illinois. From 1987 to 1998 he was a Research Scientist at IBM T. J. Watson Research Center, Yorktown Heights, New York. He has initiated, invented, and managed the research activities of five novel network architectures: 1) ring networks with spatial bandwidth reuse with a family of fairness algorithms, MetaRing, which is used as the underlying network for SSA (Serial Storage Architecture — ANSI Standard X3T10), and several IBM products; 2) optical hypergraph for combining multiple passive optical stars with novel conservative code for bit synchronization, pseudoisochronous flow control, and global clock synchronization techniques; 3) embedding of virtual rings in arbitrary topology network, the MetaNet – for bursty data traffic with no packet loss, fairness and reliable/real-time broadcast/multicast; 4) global packet networks for real-time and multimedia, which utilize GPS-based synchronization for providing deterministic quality-of-service (QoS) quarantees; and 5) fractional lambda (wavelength) switching for WDM networks, which is the focus of his current work.

Dr. Ofek was awarded the IBM Outstanding Innovation Award for his invention of the MetaRing and his contributions to the SSA products.

Moti Yung received the Ph.D. degree in computer science from Columbia University, New York, NY.

He is currently a Vice President with CertCo, Inc. (formerly Bankers Trust Electronic Commerce), dealing with electronic commerce infrastructure and applications and the cryptological aspects (combined with technological, legal, and business constraints). He was with IBM Research, Yorktown Heights, NY, from 1988 to 1996. He has been working on a wide range of cryptographic and network security issues, from foundations and principles to analysis and design of components, functions, primitives, and protocols. He was a key contributor to IBM products and offerings (authentication technology, network security products, server security, basic technology for Internet security and IBM's SecureWay). He also worked on networking protocols and architectures, algorithms, and distributed computing. He has published over 180 research papers and abstracts in top conferences and journals, and has been a member of over 35 scientific program committees for top international conferences.

Dr. Yung was awarded the IBM Outstanding Innovation Award for his work on authentication and its contributions to products.