Scheduling of Multicast Traffic in High-Capacity Packet Switches

Aleksandra Smiljanić, *Member, IEEE*

Abstract— **Switches with input buffers are scalable due to their simplicity. In these switches, the port that sources a multicast session might easily get congested as it becomes more popular. We propose that destination ports should forward copies of multicast packets to other destination ports in a specified order. In this way, the multicast traffic load is evenly distributed over the switch ports. Packets are scheduled according to the weighted sequential greedy algorithm.**

I. Introduction

As the Internet grows, high-capacity switches are needed. Also, the network should be more efficiently utilized, and better quality of service should be provided to users. For these reasons, explicit routing with bandwidth reservations and delay guarantees has been supported with frameworks such as RSVP and MPLS. Since applications on the Internet have a wide range of bandwidth requirements and holding times, high-capacity packet switches should be designed to support agile bandwidth reservations with fine granularity. Packet switches with input buffers can potentially provide high capacity because they require minimal buffering speed and switching fabric complexity [1]. But a switch with input buffers requires a more complex controller than a switch with output buffers where packets in different output buffers are scheduled independently [2], [1], [3]. We have proposed the weighted sequential greedy scheduling (WSGS) protocol to be applied in switches with input buffers [4]. The implementation of the WSGS scales well by using a pipelining technique. The WSGS provides agile bandwidth allocation due to its simple admission control. These features of the WSGS provide motivation to extend it to support multicast traffic as well.

Multicast traffic is transmitted from one source to multiple destinations (we will call them a multicast group). The diverse Internet includes a significant amount of multicast traffic. Today, a source usually sends a copied multicast packet separately to all destinations. In this case, the source and some links close to it might become overloaded. Alternatively, multicast packets could be sent along precalculated multicast trees. Here, a packet is copied at branch nodes of the tree, so the transmission load is distributed over those nodes, and the links closer to the source carry less traffic. Signaling and processing required to calculate these multicast trees is burdensome in wide area networks with a large number of nodes and edges [5], [6]. For the given switch sizes, the number of nodes increases faster than linearly with the capacity required by end-users, because of the increasing number of transit switch ports. This observation amounts to the importance of high-capacity switches for simple network control, efficient port utilization, and, consequently, support of multicast traffic on the Internet.

It has been recognized that large switches with input buffers do not well support multicasting of popular content with large fan-outs (numbers of destinations). For example, it was shown in [7] that a three-stage Clos switch requires a speed-up equal to the maximum fan-out to ensure strict non-blocking. We have shown, [4], that the non-blocking condition in a cell-based switch with input buffers and a three-stage Clos circuit switch are equivalent. So, a cell-based switch with moderate speed-up would not carry popular multicast sessions properly. In addition, users attached to the port that multicasts popular content would be clogged. However, the multicast transmission load can be distributed over the multicast destination ports. We propose that the multicast destination ports should forward the multicast information through the switch to the other destination ports that have not received it yet. Packets are scheduled according to the WSGS protocol. A switching fabric with moderate speed-up run by the proposed algorithm is non-blocking for an arbitrary multicast traffic pattern.

II. Weighted Sequential Greedy Scheduling

We proposed earlier a practical way to schedule unicast traffic in high-capacity switches [4]. Our approach is sequential greedy scheduling based on credits. We will present this approach for the sake of completeness. Inputs choose outputs one after another in a pipeline fashion. Packets are stored in different queues according to their destinations, so that the information about any queue status (empty or non-empty) and its heading packet is readily obtained. Such an input buffer organization is often referred to as a buffer with virtual output queueing (VOQ) [1]. A schedule for one time slot is calculated in multiple earlier time slots, and multiple schedules are calculated in each time slot. Here, the schedule is a set of input-output pairs to be connected in a time slot, so that inputs in question transmit packets to outputs to which they are connected. Figure 1 shows the time diagram for pipelining where in each time slot, only one input selects an output for a particular time slot in the future. If $\mathrm{I}_i \rightarrow \mathrm{T}_k$ is assigned to some time slot T_j, it means that input I_i reserves an output for time slot T_k, and this reservation is made during time slot T_j. Bold vertical lines enclose a calculation of one schedule, which lasts N time slots in the given example. Here N denotes the number of input and output ports. In the more general case, in any time slot multiple inputs might select outputs for some future time slot, or it might take multiple time slots for an input to select an

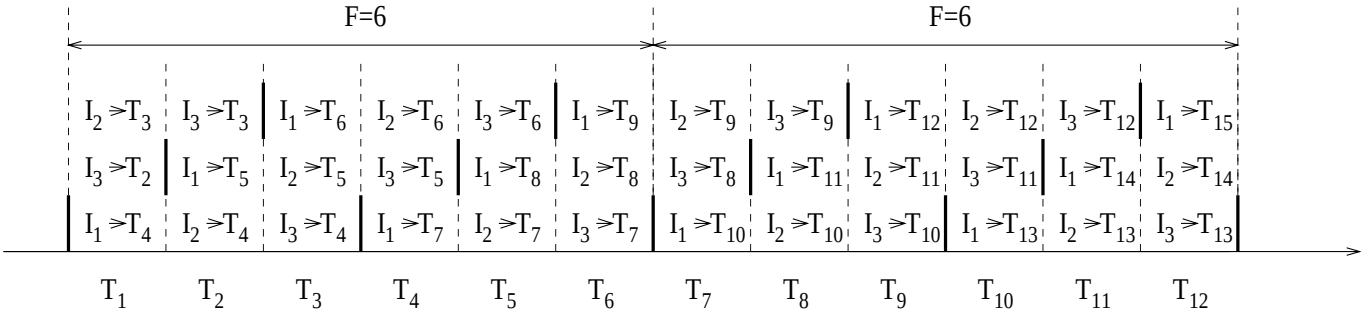


Fig. 1. Pipelining of the sequential scheduling protocol in a 3×3 switch.

output for a future time slot.

Time is further divided into frames comprising a fixed number of time slots, F (as shown in Figure 1). Input-output pairs are guaranteed negotiated numbers of time slots (credits) per frame. Before some input chooses an output for the first time slot within a frame, its counters are set to the negotiated numbers of credits. In the example shown, input i sets its counters c_{ij} to negotiated values a_{ij}, $c_{ij} = a_{ij}$, $1 \leq j \leq N$, in time slots $k \cdot F - N + i$, $k \geq 1$. Only queues with positive counters would compete for service, and whenever a queue is served its counter is decremented by 1. After inputs schedule packets from the queues with positive counters, they might schedule packets from the remaining queues in the same pipelined fashion, as was described in [4]. In this way, best effort traffic can be accommodated if there is some bandwidth left after the higher priority traffic is served.

The pipelined sequential greedy scheduling algorithm is easy to implement, and it scales well with increasing number of ports and decreasing packet transmission time. An advantage of the proposed protocol is that it requires communication only among adjacent input modules, and, consequently, the simple scheduler implementation as shown in Figure 2. Also, by using pipelining the requirements on the speed of electronics are relaxed. In addition, the WSGS implies an extremely simple admission control protocol that provides agile bandwidth reservations. When bandwidth b_{ij} is requested by input-output pair (i, j), then $a_{ij} = \lceil b_{ij} \cdot F/B \rceil$ time slots per frame, i.e. credits, should be assigned to it. Here, B is the port bit-rate. We have shown earlier ([4]) that the bandwidth can be allocated to input-output pair (i, j) if the following condition holds:

$$T_i + R_j \leq F + 1. \tag{1}$$

where $T_i = \sum_k a_{ik}$ is the transmission load of input i, and $R_j = \sum_k a_{kj}$ is the reception load of output j. Consequently, the bandwidth can be allocated in a switch if each input and output are allocated less than half time slots per frame. In other words, the switching fabric should have a speed up of two, in order to pass the traffic that does not overload any of the outputs.

III. Balancing of the Multicast Traffic Load

If a multicast packet could be scheduled for transmission to multiple destinations in the same time slot, the available

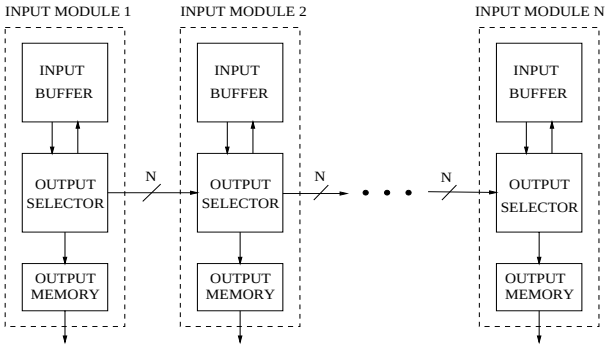


Fig. 2. Central controller implementing sequential greedy scheduling protocol.

bandwidth might depend on the multicast traffic pattern. The corresponding admission control protocol would be intricate, because it must consider the multicast traffic pattern. Also, some high-capacity switching fabrics do not allow the transmission of a packet to multiple destinations at the same time [8]. In the simplest implementation of both the scheduler and the switching fabric, multicast packets are independently scheduled for different outputs in a multicast group according to the described greedy algorithm. This algorithm is scalable, and implies simple admission control. However, if an input sends a multicast packet serially to all designated outputs, its bandwidth will be wasted in multiple transmissions of the same packet. Let's denote the bandwidth of multicast session (i, s) sourced by input i by b_{is}^m, the set of outputs that receive the multicast information by multicast group $\mathcal{M}_{is}$, and the number of outputs in set $\mathcal{M}_{is}$ by $|\mathcal{M}_{is}| \leq N$. Note that for a unicast session $|\mathcal{M}_{is}| = 1$. If the input sends multicast packets separately to all outputs in the multicast group, their transmission will consume bandwidth equal to $b_{is}^m \cdot |\mathcal{M}_{is}|$. In high-capacity switches, a multicast fan-out might get large, and a popular multicast session would clog the port. If, e.g., 250 ports request the multicast session of 10Mb/s, the source port would consume 2.5Gb/s for the transmission of this session. So, all users attached to the OC-48 port would be clogged due to the popularity of a single session. Since the popularity of some content varies over time, it is also not wise to dedicate the entire OC-48 port to the content in question. Namely, as its popularity fades away, the OC-48

port would remain underutilized. Generally, utilization of the port capacity becomes low when a significant amount of multicast traffic that it transmits has a large fan-out. The performance degradation is more severe in a high-capacity switch with the large numbers of ports, N.

Let us observe that once any port from the multicast group receives a multicast packet, it may as well forward it to $P \geq 1$ ports of that multicast group which have not received the packet. Here, each port comprises one input and one output. In this way, the transmission burden would be balanced over all ports in the multicast group if the forwarding fan-out P is chosen to be adequately small. We have seen that for $P = N$, a multicast packet can be transmitted to all outputs within one frame, but a large number of credits might have to be allocated for the multicast session and the input port would get clogged. On the other hand, if each port forwards a packet to only one port, i.e. $P = 1$, then each port uses a small additional capacity for forwarding, but the multicast packet might experience delay of up to N frames. Namely, in the worst case, a packet would be forwarded only once per frame. This delay would become excessive in a high-capacity switch with large numbers of ports N, and large frame length F. Apparently, there is a trade-off between utilized capacity and packet delay that depends on the chosen parameter P.

We will analyze the switch capacity that can be guaranteed to its ports in terms of the parameter P. The bandwidth demand and the packet forwarding order determine the credit allocation. It follows from equation (1) that credits can be assigned to some input-output pair (i, j) if it holds that $T_i + E_i + R_j \leq F + 1$, where T_i is the number of time slots per frame reserved for packets that are transmitted by input i, E_i is the number of time slots per frame reserved for input i to forward its multicast packets, and R_j is the number of time slots per frame reserved for packets bound to output j. In the worst case, input i should forward all packets that it receives, and $E_i = P \cdot R_i$. It is fairly easy to show, that the maximum capacity can be utilized for an arbitrary traffic pattern if for all ports, $1 \leq i \leq N$, the following conditions hold:

$$T_i \leq \frac{F+1}{P+2},$$
$$R_i \leq \frac{F+1}{P+2}. \qquad (2)$$

So, the maximum portion of the port capacity that can be reserved is $1/(P+2)$. Note that the admission conditions agree with our previous result for unicast traffic when $P = 0$, see [4]. Here, a port capacity is the bit-rate at which data is transmitted from an input port through the switching fabric to an output port. Therefore, it could be referred to as the internal port capacity. In other words, the switching fabric requires a speed-up of $P + 2$ in order to pass all the traffic admissible by outputs. The port hardware facing the network is fully utilized.

Next, we will calculate the packet delay in terms of the parameter P. Let us assume that a multicast packet of session (i, s) is forwarded to all outputs within S frames. In the first frame, the port that receives the packet from an input forwards it to P ports. In the kth frame, P^{k-1} ports forward the packet to P^k other multicast ports. Since $|\mathcal{M}_{is}| > 1 + P + \ldots + P^{S-2} = (P^{S-1} - 1)/(P - 1)$, it holds that:

$$S < \log_P \left((P-1) \cdot |\mathcal{M}_{is}| + 1 \right) + 1. \qquad (3)$$

The switch capacity that can be utilized equals $C = N \cdot B/(P+2)$. There is an obvious trade-off between granularity of bandwidth reservations $G = B/F$ and packet delay $D = S \cdot F \cdot T$, where B is the port bit rate, and T is the packet transmission time. Assuming that $P = 2$, $B = 10$Gb/s, $T = 50$ns, we calculate C, G and D in Table I in terms of N and F. As the frame length is increased, the granularity is refined, but the packet delay is prolonged. However, the packet delay does not increase significantly with the switch size. Since packets would pass through a small number of high-capacity switches, the long packet delay through one switch could be tolerated even by delay-sensitive applications (voice and video conferencing). The frame length should be chosen according to the requirements of these applications, and the granularity could be further refined for other applications by sharing the credits.

TABLE I
THE QUALITY OF THE SWITCH SERVICE

N	10^3		4×10^3	
F	10^4	10^5	10^4	10^5
C [Tb/s]	2.5	2.5	10	10
G [Mb/s]	1	0.1	1	0.1
D [ms]	5	50	5.5	55

Table I shows that the forwarding fan-out of $P = 2$ would provide satisfactory packet delay through a swith. For $P = 2$, the switch utilization is $1/(P + 2) = 25\%$. In [9], an unfortunate multicast traffic pattern was found for which the capacity utilized by greedy scheduling algorithms drops below 40% for large switches. By forwarding multicast packets, our algorithm still guarantees a significant portion of the switching capacity for arbitrary traffic pattern. In addition, the proposed protocol would switch all the traffic that can be admitted by output ports.

IV. Scheduling of Multicast Traffic

When a new multicast session is requested, an admission controller checks if there is enough spare capacity according to condition (2). If some outputs request to join an existing multicast session, the admission controller needs only to check if that output has enough spare capacity for this session reception. In the more general case, only a subset of multicast outputs have enough spare capacity, and they are admitted. Whenever a new multicast session is admitted, the order in which packets are forwarded by ports in the multicast group should be calculated. Also, whenever some outputs are joining or leaving an existing session, the forwarding order should be updated. Obviously, when a

multicast session is initiated, or when multiple ports are joining or leaving a session, they would do so sequentially, one after another.

Forwarding order is described by a forwarding tree in which nodes represent ports, and each port forwards packets to the ports that it reaches through its branches. We propose the following simple algorithm for adding and removing a port to the tree. Each port of the tree should store the parent (previous) port and the children (next) ports. Each port should also store its branch fan-outs, where the branch fan-out is the number of ports that could be reached through that branch. A request for adding a port to the multicast group is sent to the tree root. It then travels through the tree, always taking the branch with the smallest fan-out. The fan-out of every branch that this request passes is increased by one. The new port is added as the leaf to the tree (the port without children) that is reached by the request through the shortest tree path. Similarly, when a port wants to leave the tree it sends a request to the tree root. This request now travels through branches with the largest fan-outs until it gets to a leaf, and the fan-outs of these branches are decremented by one. This leaf at the end of the longest tree path will replace the port that is leaving the multicast session. The port to leave sends, along with the request, the information about its parent and children ports, as well as about its branch fan-outs, so that the chosen leaf would store these parameters. Then, this leaf port informs its parent to stop forwarding packets to it, and the parent of the port leaving to start forwarding packets to it. By adding the new port to the tree at the end of its shortest path, and replacing the leaving port by the leaf of the longest path, the tree is kept in balance, and the forwarding delay is minimized. We believe that in this way, minimal memory and processing per port are required for tree calculation and updates.

Figure 3 shows how a new port is added to the multicast group, while Figure 4 shows how a new port is removed from the multicast group. Both figures assume $P = 2$. Nodes in the shown directed graphs denote ports. Edges (branches) denote forwarding. The encircled branch fan-out denotes the number of ports reached by multicast packets using that branch. In Figure 3, port 11 requests to join the multicast group. It sends the request to port 9, which is the multicast input and the tree root. This request is forwarded along the branches with the smallest fan-outs as shown by the dashed line. Branch fan-outs are updated as shown. Port 11 is added as a leaf to the tree after port 7, which will forward multicast packets to it in the future. In Figure 4, port 4 requests to leave the multicast group. It sends the request to input port 9. This request is forwarded along the branches with the largest fan-outs as shown by the dashed line. Port 8, which is the leaf, is chosen to replace port 4 on leave. Port 8 informs ports 6 and 2, its parent and the port 4 parent, that it will replace port 4, so that port 6 should stop forwarding packets to it, and port 4 should start forwarding packets to it. Each port has to store only three ports and two fan-outs in its forwarding table for each multicast session.

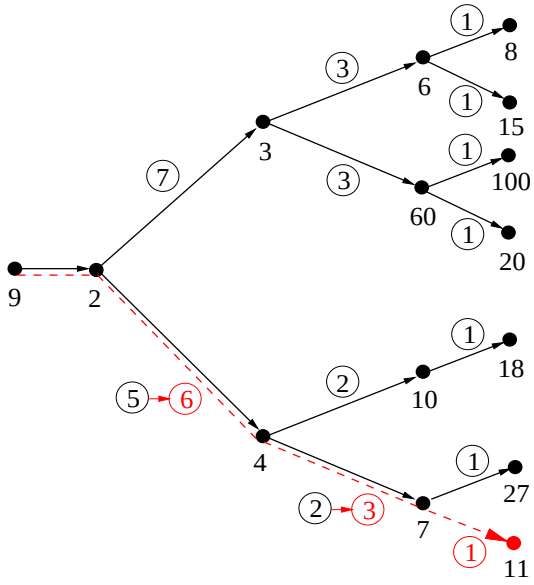


Fig. 3. Adding a port to the multicast group

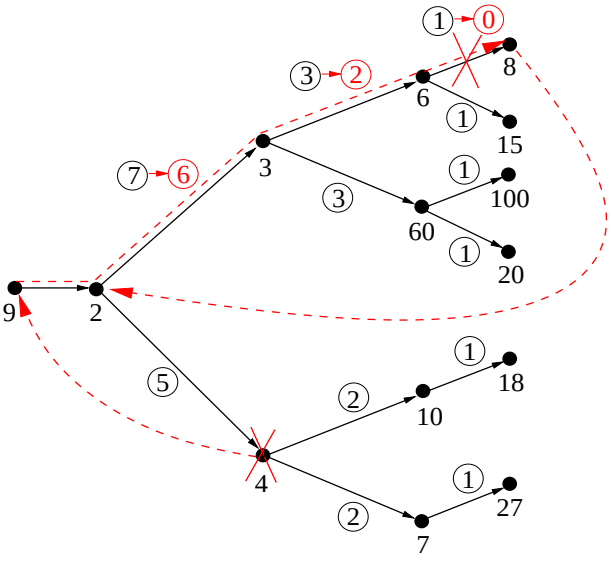


Fig. 4. Removing a port from the multicast group

When a new multicast session is initiated, or an old one is terminated, a multicast input updates its credits ($a_{ip(i)} \leftarrow a_{ip(i)} \pm a^m_{is}$, where i is the multicast input, and it forwards packets to output $p(i)$). Similarly, when a port joins or leaves a multicast session, it will update its negotiated credits ($a_{jp_k(j)} \leftarrow a_{jp_k(j)} \pm a^m_{is}$, where port j forwards packets to the port in question $p_k(j)$).

The admission of a multicast session can also be pipelined. In addition, the multicast session may be released in a pipelined fashion. Such pipelined admission control might better utilize the available bandwidth. For example, the bandwidth for a multicast session is reserved in one frame, but packets are transmitted only to the first port in the forwarding tree in the next frame. So, the bandwidth reserved for forwarding of these multicast packets to the rest of the ports is wasted because they have not arrived into the appropriate queues yet. But since the input transmits packets to the first port in the forwarding tree within one frame, then the bandwidth for forwarding packets by this port should be reserved in the same frame (which is one frame after the bandwidth has been reserved for transmission from input), and so on. Similarly, when a multicast session has ended, the input will stop transmitting packets, but packets that were previously transmitted might be forwarded through the switch for a while. Alternatively, the bandwidth reserved for forwarding of multicast packets from the first port in the forwarding tree should be released one frame after the bandwidth reserved for transmission from the multicast input has been released, and so on. The pipelined admission control can be summarized as follows. Input i reserves the bandwidth for transmission to port $j \in \mathcal{P}_1 = \{p(i)\}$ by updating the assigned credits in some frame t if conditions (2) for i and $j \in \mathcal{P}_1$ hold. Then, port $j \in \mathcal{P}_1$ reserves bandwidth for packet forwarding to ports $j \in \mathcal{P}_2 = \{p_1(j), ..., p_P(j)\}$ for which conditions (2) hold, by updating the assigned credits in frame $t+1$. In general, ports $j \in \mathcal{P}_l$ reserve the bandwidth for packet forwarding to the associated ports $j \in \mathcal{P}_{l+1} = \{p_k(j) | j \in \mathcal{P}_l, 1 \leq k \leq P\}$ for which conditions (2) hold, by updating the assigned credits in frame $t+l$. Similarly, if this multicast session ends in frame t, input i releases the bandwidth reserved for port $p(i)$ in frame t, and ports $j \in \mathcal{P}_l$ release the bandwidth reserved for forwarding packets to their associated ports $j \in \mathcal{P}_{l+1}$ in frame $t+l$, by updating their associated credits.

In summary, when a new multicast session is requested, or a port joins an existing multicast session, the appropriate admission conditions are checked. If the bandwidth request is granted, the forwarding tree is modified, and the credits of the associated ports are updated. Before an input chooses an output for the beginning of some frame, its counters are set to their negotiated numbers of credits, $c_{ij} = a_{ij}$, $1 \leq i, j \leq N$. Packets are scheduled according to the previously described pipelined WSGS in which the queues with positive counters are served with priority. When a multicast session is terminated, or a port leaves the session, the forwarding tree is modified and credits of the associated ports are updated. Note also that the switch does not have to control the admission if enough bandwidth is provisioned to the users attached to each port (i.e. sum of the users' peak bit-rates is smaller than the port bit-rate). In that case, a transmitting user should only check if the destination user can receive the data at the specified rate. If it can, then the transmitting user sends data during the time slots whose number per frame has been negotiated.

V. Reliability Considerations

When a port in the multicast group fails, all ports belonging to the multicast subtree rooted at this port will not be receiving multicast packets, because the failed port will not be forwarding these packets. So, when the port fails it should be replaced by some of the tree leaves in a way described earlier. However, the port is not aware of its failure in advance, so it cannot signal its departure from the multicast group. As the port fails, its forwarding table might become also unavailable. For this reason, it would be advantageous for a port to store not only its parent, children ports and branch fan-outs, but also, e.g., its grandparent (parent's parent), sibling (parent's children) ports and branch fan-outs of the parent port. For example, in Figure 3, port 10 stores its parent port 4, its child port 18, and the fan-out of branch 10-18 as before, but also, its grandparent port 2, sibling port 7, and the fan-outs of branches 4-7 and 4-10. In this way, when a port fails, and the failure is detected (e.g., by the children of that port) its children inform the root about this failure, and send to the root information about the parent, children and branch fan-outs of the failed port, so that a chosen leaf port could replace the failed port in a forwarding tree in a manner similar to that described above. After this replacement, the new port would learn about its grandparent, sibling and branch fan-outs of the parent port, by using a specified signaling procedure. In summary, each port should store up to $4P+1$ entries (P children ports, P branch fan-outs, 1 parent port, 1 grandparent port, P parent branch fan-outs, $P-1$ sibling ports).

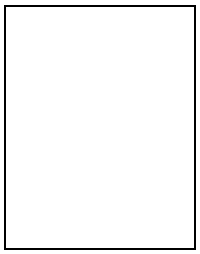
In an alternative approach, each port could store its grandchildren ports, and its children branch fan-outs. When a port recognizes that its child port failed, it would inform the tree root and send the required information about this failed port (about its children, and branch fan-outs). As before, some leaf port will replace the failed port, and it will learn from its children about its grandchildren, and children branch fan-outs. In this case, each port should store $2P(P+1)$ entries. Signaling of the failure is somewhat simplified in this latter approach, because there is only one parent of a failed port that will inform the tree root, whereas in the former reliability enhancing scheme, all the children of the failed node would inform the tree root about the failure.

VI. Conclusion

We proposed the scheduling protocol for flexible bandwidth allocation in high-capacity switches that support multicast traffic. In a network with a large number of low capacity switches, the network control is hindered by

required signaling and high complexity admission control algorithms. Our protocol takes advantage of today's high capacity switching fabrics. The proposed centralized scheduler can make fast decisions and provide agile resource allocation. Multicast packets are forwarded through the switch, so that their transmission is balanced over the switch ports. As a result, a switch with a moderate speedup transports contents whose popularities change arbitrarily in magnitude and over time. Since the proposed switch is non-blocking, it will pass an arbitrary traffic pattern as long as the output ports are not overloaded. The non-blocking property of the high-capacity switch significantly simplifies provisioning and network planning.

Acknowledgement: I thank Nick Frigo, Sheri Woodward and Bob Doverspike from AT&T Labs for their comments.



Aleksandra Smiljanić (M '96) received M.A. and Ph.D. degrees in electrical engineering from Princeton University in 1996 and 1999, respectively. She completed B.Sc. in electrical engineering at Belgrade University in 1993. She has worked for AT&T Labs since 1999 on communication protocols and optical networks. She worked for two summers at NEC USA designing a packet switch with terabit capacity. Aleksandra has taught several courses at Princeton and Belgrade Universities.

Aleksandra Smiljanić is the author of the Best Papers at the IEEE Conference on High Performance Switching and Routing 2000, and IEICE/IEEE Workshop on High Performance Switching and Routing 2002. She is a recipient of the Aleksandar Damjanović Prize as the best student in her class at Belgrade University, 1993. Before university, she won numerous prizes in Yugoslav and international competitions in mathematics and physics.

References

[1] N. McKeown *et al.*, "The tiny tera: a packet switch core," *IEEE Micro*, vol. 171, Jan.-Feb. 1997, pp. 26-33.

[2] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems,* vol. 11, no. 4, November 1993, pp. 319-352.

[3] A. Mekkittikul, and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," *IEEE INFOCOM*, March 1998, pp. 792-799.

[4] A. Smiljanić, "Flexible bandwidth allocation in high-capacity packet switches," *IEEE/ACM Transactions on Networking,* vol. 10, April 2002, pp. 287-293.

[5] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-aware multicast routing protocol," *IEEE Journal on Selected Areas in Communications,* vol. 18, no. 12, December 2000, pp. 2580-2592.

[6] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "Global routing based on Steiner Min-Max trees," *IEEE Transactions on Computer Aided Design,* vol. 9, no. 12, December 1990, December 1990, pp. 1318-1325.

[7] M. Listanti, and L. Veltri, "Non blocking multicast three-stage interconnection networks," *IEEE GLOBECOM*, Decemeber 1999, pp. 1401-1405.

[8] J. Gripp, P. Bernasconi, C. Chan, K. L. Sherman, and M. Zirngibl, "Demonstration of a 1Tb/s optical packet switch fabric (80*12.5GB/S), scalable to 128 Tb/s (6400*20Gb/s)," *Postdeadline Paper in ECOC 2000.*

[9] A. M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "On the throughput of input-queued cell-based switches with multicast traffic," *Proceedings of IEEE INFOCOM,* 2001, pp. 1664-1672.