Flexible Bandwidth Allocation in High-Capacity Switches with Multicasting

A. Smiljanić
AT&T Laboratories, Middletown, NJ 07748, USA
phone: (732) 420-9052, fax: (732) 368-9477
e-mail: aleks@research.att.com

Abstract—Information exchanged on Internet consume the wide range of bandwidths and might have very short holding times. In addition, significant portion of Internet traffic is multicast in nature. To our best knowledge, no work has been done on agile bandwidth allocation in high-capacity grooming switches that support multicast traffic. We propose a simple and efficient scheduling protocol that allows fast bandwidth reservations for multicast traffic in high-capacity grooming switches. A fraction of the port capacity is guaranteed to be utilized regardless of the traffic pattern. The network planning is significantly simplified since only traffic transmitted to each input port and traffic received from each output port should be provisioned.

I. INTRODUCTION

The challenge of grooming switches in today’s network is to provide high-capacity, fine granularity, agile bandwidth reservations, and to be able to support multicast traffic as well. These requirements are driven by the nature of the traffic on Internet. Applications often consume large bandwidths, and may last very shortly. In addition, a significant portion of Internet traffic is multicast, i.e. transmitted from one source to multiple destinations. Owing to the developing electronic and optical technologies, such grooming switches may become a reality.

The capacity of the electronic and optical switching fabrics is increasing substantially in recent times driven by the traffic demand on Internet. It is anticipated that the core of the wide area network will consist of large optical cross connects (OXCs) with coarse granularity. The traffic would be groomed to the large optical pipes by high-capacity circuit or packet switches. A packet switch might be built in optics as long as the switch configuration time is less than the packet transmission time [2], [7]. In both kinds of grooming switches, the central controller checks if the bandwidth is available and assigns it accordingly. In a circuit switch, the central controller assigns circuits to input-output pairs if their bandwidth requests were admitted. In a packet switch, the central controller schedules packets so that the bandwidth reservations are provided.

The sequential greedy scheduling based on credits and associated admission control mechanism are simple to implement in high-capacity switches with input queuing [8]. Packet switches with input buffers are favored because they require minimal buffering speed and switching fabric complexity [6]. The implementation of the sequential greedy scheduling protocol scales well by using a pipelining technique. The admission controller has to check only two inequalities when new bandwidth is requested, i.e. it checks if the input and output ports have enough of the spare capacity. A similar algorithm can be used for bandwidth reservations in circuit switches.

Multicast traffic is sent from one input to multiple outputs. In the most straightforward scheme, each multicast packet would be scheduled and transmitted separately to all outputs. Consequently, bandwidth would be separately reserved between the given input and each of the addressed outputs. In this scheme, the total switch capacity might significantly drop for some unfortunate multicast traffic pattern, e.g. when a large amount of multicast traffic sourced by some input is bound for a large number of outputs. In this paper, we propose a scheduling protocol that efficiently utilizes the resource of a high-capacity switch for all multicast traffic patterns. Packets are again scheduled according to the previously proposed sequential greedy algorithm based on credits. But, the input sends multicast packets to only one port in a multicast group. Then, each port in the multicast group forwards the information to the small number of ports in this group that has not received it yet. In this way, the transmission...
load is balanced over all ports in the multicast group, so that it is not a bottleneck for switching of the multicast traffic. The admission controller should only check if the input and each of the output ports in the multicast group have enough of the spare capacity. The same underlying idea can be used for setting up connections for data flows through high-capacity circuit switches.

II. SCALABLE SCHEDULING PROTOCOL AND AGILE ADMISSION CONTROL PROTOCOL IN THE CASE OF UNICAST TRAFFIC

We proposed earlier a straightforward way to schedule unicast traffic in high-capacity switches [8]. Our approach is the sequential greedy scheduling based on credits. Inputs choose outputs one after another in a pipeline fashion. Namely, a schedule for one time slot is calculated in multiple earlier time slots, and multiple schedules are calculated in each time slot. Here, a schedule is the 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 future. If \( I_i \rightarrow T_k \) is assigned to some time slot \( T_j \), it means that input \( I_i \) reserves an output for time slot \( T_k \) in time slot \( T_j \). Bold vertical lines enclose the 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 input to select an output for a future time slot. Time is further divided into frames comprising fixed number of time slots, \( F \) (as shown in Figure 1). In the specified time slot of a frame, counters of some input are set to the negotiated values. In the shown example, input \( i \) sets its counters \( c_{ij} \) to negotiated values \( a_{ij} \), \( c_{ij} = a_{ij} \), \( 1 \leq j \leq N \), in time slots \( k \cdot F - N + i - 1 \), \( k \geq 1 \). 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 queues with positive counters, they might schedule packets from the remaining queues in the same pipelined fashion, as was described in [8]. In this way, the best effort traffic can be accommodated if there is some bandwidth left after the higher priority traffic is served. Packets are stored into 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 input buffer organization is often referred to as a buffer with virtual output queuing (VOQ) [6].

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, it 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 \( a_{ij} = \left\lfloor b_{ij} \cdot F/B \right\rfloor \) time slots per frame, i.e. credits, should be assigned to it. We have shown earlier ([8]) that the bandwidth can be allocated to input-output pair \((i, j)\) if the following condition holds:

\[
\sum_k a_{ik} + \sum_k a_{kj} \leq F + 1. \tag{1}
\]

Consequently, the bandwidth can be allocated in a switch
Fig. 2. Central controller implementing greedy sequential scheduling protocol. IM: Input Module, IB: Input Buffer, RR: Round Robin module, M: memory that stores outputs selected for future time slots.

if for all $1 \leq i \leq N$ the following conditions hold:

$$T_i = \sum_k a_{i,k} \leq \frac{F + 1}{2},$$

$$R_i = \sum_k a_{k,i} \leq \frac{F + 1}{2}. \quad (2)$$

Here $T_i$ is the number of credits assigned to input $i$ for transmission, and $R_i$ is the number of credits assigned to output $i$ for reception. Half of the time slots per frame can be allocated to any input or output, meaning that 50% of the port capacity can be reserved for any unicast traffic pattern.

In the above protocol, bandwidth is reserved for a flow of packets transmitted from input to output, and each packet is scheduled separately. The same algorithm can be used for scheduling the entire flow of packets. Namely, when a bandwidth request for a new flow of packets is accepted, fixed time slots within each frame are assigned to this flow until it is terminated by end-users. The time slot assignment is calculated according to the pipelined sequential greedy scheduling algorithm. Here, each input has to consider a table of outputs reserved in the previous frames, and time slots which it reserved in the previous frames; input updates its tables in each frame. New flow is admitted if inequalities (2) are fulfilled. Fixed assignment of time slots within frames to data flows is basically circuit switching.

In data networks individual sessions might consume large bandwidths and have short durations. For this reason, bandwidth reservations were initially avoided. Namely, a network consists of many low capacity switches, and setting up a connection through these switches might have lasted longer than data session durations [3]. The described sequential greedy scheduling based on credits can be implemented with current technology to provide fast bandwidth reservations in terabit switches.

III. BALANCING OF THE MULTICAST TRANSMISSION LOAD

Multicast packets are bound from one input to multiple outputs. In switches with output buffers, a multicast packet can be simultaneously transmitted to all output buffers in question. Unfortunately, the capacity of a switch with output queueing is limited to the buffer speed throughput [6]. Consequently, many switches with output buffers should be interconnected to support high switching capacity. In such a network, bandwidth reservations for multicast traffic require signalling among many switches and high-complexity admission control protocols [1].

On the other hand, switches with input buffers provide high capacity, but require scheduling since each output can receive at most one packet at the time. Algorithms that minimize the number of time slots in which a packet is transmitted from input to all multicast outputs were also shown to be difficult to implement [5]. In switches employing such algorithms, the available switching capacity depends on the multicast traffic pattern which hardens bandwidth reservations. Alternatively, multicast packets might be independently scheduled for different outputs in the multicast group according to the described greedy algorithm. The main advantage of such an algorithm is its scalability. In addition, it can schedule the switching fabric in which an input transmits to at most one output at the time [2]. 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. In the worst case, an input transmits a multicast packet to all $N$ outputs. Let’s denote by $a_{i,s}^m$ the number of time slots per frame assigned to multicast session $(i,s)$ sourced by input $i$, by $|\mathcal{M}_{i,s}|$ the set of outputs in this multicast group, and by $|\mathcal{M}_{i,s}|$ the number of outputs in set $\mathcal{M}_{i,s}$. Note that for an unicast session $|\mathcal{M}_{i,s}| = 1$. It follows from equation (2) that credits can
be assigned to some input-output pair \((i, j)\) if:
\[
\sum_s a^m_{is}[M_{is}] + \sum_{k,s,j \in M_{ks}} a^m_{ks} \leq F + 1. \tag{3}
\]

In the worst case, input \(i\) sends packets to all \(N\) outputs, \([M_{is}] = N\), where \(N\) is the number of ports, and from (3), the transmitting port is underutilized:
\[
T_i = \sum_s a^m_{is} \leq \frac{F + 1}{N}. \tag{4}
\]

One \(N\)th of the time slots in a frame can be allocated to input \(i\), meaning that only \(1/N\) of the transmitting port capacity can be utilized. Generally, the 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 high-capacity switches with a large number 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 any of the ports of that multicast group that 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. Since each of the ports has to forward only traffic that it receives, the transmitted traffic is automatically limited to be a portion of the port bit-rate. From equation (2) credits can be assigned to some input-output pair \((i, j)\) if it holds that:
\[
T_i + E_i + R_j \leq F + 1, \tag{5}
\]
where
\[
T_i = \sum_s a^m_{is}, \quad E_i \leq R_i = \sum_{k,s,j \in M_{ks}} a^m_{ks}, \quad R_j = \sum_{k,s,j \in M_{ks}} a^m_{ks}. \tag{6}
\]

Here, \(T_i\) is the total 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 \(j\). Conditions (5,6) imply that credits can be assigned to input-output pair \((i, j)\) if:
\[
T_i + R_i + R_j \leq F + 1. \tag{7}
\]

It further follows that the bandwidth allocation is possible if for all ports \(i, 1 \leq i \leq N\), it holds that:
\[
T_i \leq F_i, \quad R_i \leq F_r, \quad F_i + 2 \cdot F_r = F + 1. \tag{8}
\]

The maximum number of time slots per frame that can be reserved for both transmission from a port and reception by a port equals:
\[
\max \min (F_i, F_r) = \max \min (F_r, F + 1 - 2 \cdot F_r) = \frac{F + 1}{3}. \tag{9}
\]

And, the bandwidth allocation is possible if for all ports, \(1 \leq i \leq N\):
\[
T_i \leq \frac{F + 1}{3}, \quad R_i \leq \frac{F + 1}{3}. \tag{10}
\]

So the maximum percentage of the port capacity that can be reserved is 30\% for arbitrary multicast traffic pattern.

Finally, time slots can be reserved for new multicast session \((i, n)\) if the following inequalities are fulfilled:
\[
\sum_s a^m_{is} \leq \frac{F + 1}{3}, \tag{11}
\]
\[
\sum_{k,s,j \in M_{ks}} a^m_{ks} \leq \frac{F + 1}{3}, \tag{12}
\]
for all \(j \in M_{in}\).

In the case of unicast traffic, a packet is delayed for at most one frame as long as peak bit-rate is less than the reserved bit-rate. However, the delay of a multicast packet may be multiple frames due to the proposed forwarding mechanism even when the transmitting bit-rate is smaller than the reserved bit-rate. In the worst case, a packet in multicast session \((i, s)\) is forwarded only once per frame, and its delay is equal to \([M_{is}]\) frames. Such long delay may be critical for interactive applications such as voice calls or video conferencing. In order to decrease the delay, each port might be allowed to forward a packet to \(P \geq 1\) ports that have not received it yet. Similarly as in (7), credits can be allocated to input-output pair \((i, j)\) if:
\[
T_i + P \cdot R_i + R_j \leq F + 1. \tag{13}
\]
And, the bandwidth allocation is feasible if it holds that:

\[
T_i \leq \frac{F + 1}{2 + P},
\]
\[
R_i \leq \frac{F + 1}{2 + P}.
\]

(14)

So, the portion of the port capacity that can be reserved is \(1/(2 + P)\) in this case. Let us assume that a multicast packet of some session is forwarded to all outputs within \(S\) steps. In the first step, the port that receives the packet from an input, forwards it to \(P\) ports. In the next step, each of these ports forwards the packet to \(P^2\) other multicast ports. In the last step the packet is sent to at most \(P^{S-1}\) remaining ports. So, the number of ports that are receiving the packet \(|\mathcal{M}_{is}| > 1 + P + \ldots + P^{S-2} = (P^{S-1} - 1)/(P - 1) \Rightarrow S < \log_P ((P - 1) \cdot |\mathcal{M}_{is}| + 1) + 1\). As a result, the port capacity that can be reserved will somewhat decrease, but the maximum packet delay will significantly decrease from \(|\mathcal{M}_{is}|\) frames to \(\log_P ((P - 1) \cdot |\mathcal{M}_{is}| + 1)\) frames. For \(P = 2\), the maximum packet delay is as short as 10 frames even for a switch with 1024 ports, while the percentage of the port capacity that can be reserved is 25%.

A set of conditions for admitting multicast session \((i, n)\) becomes:

\[
\sum a_{n} \leq \frac{F + 1}{2 + P},
\]
\[
\sum_{k, s, j \in \mathcal{M}_{ks}} a_{m_{ks}} \leq \frac{F + 1}{2 + P},
\]

(15)

(16)

for all \(j \in \mathcal{M}_{in}\).

IV. SCALABLE SCHEDULING PROTOCOL AND AGILE ADMISSION CONTROL PROTOCOL IN THE CASE OF MULTICAST TRAFFIC

A multicast packet may be forwarded through the switch in a predetermined or arbitrary order. The packet forwarding in a predetermined order would significantly simplify the protocol implementation.

If a port is forwarding the packet to any of the remaining ports in a multicast group, it must store the multicast group for each of its multicast sessions and it must consider all remaining outputs in the multicast group when scheduling a packet transmission. The latter requirement would require the information about remaining outputs in the packet header, and also complicate design of VOQ buffers. A possible implementation of VOQ buffers has been described in [4]. In principle, a fixed allocation of the memory locations to different queues would be inefficient. Instead, each queue should be a dynamic list of memory locations assigned to packets, each of which points to the next one. A pointer to the beginning, the first unscheduled packet, and the end of each queue should be stored and updated when a packet is transmitted, or new packet arrives. If a multicast packet can be transmitted to any of its outputs, then it should be included in the corresponding queues. Pointers of the preceding packets in these queues have to be updated when a packet in question is scheduled for transmission, or transmitted to any of the outputs. But, the packet in question also has to point these packets for their easier update. So, each packet should be able to point \(N\) (the maximum multicast group size) other packets in the buffer. Finally, if a packet can be forwarded to arbitrary ports, credit allocation to input-output pairs is ambiguous. Instead, credits should be allocated to individual multicast sessions at their inputs. It is clear that the packet forwarding in an arbitrary order significantly increases the implementation complexity.

If the forwarding order is predetermined, each port has to store only the port to which a packet should be forwarded for each multicast session to which it belongs. Also, a port schedules the packet transmission for a specified multicast output which simplifies the design of VOQ buffers. Credits can be allocated to input-output pairs for the given multicast traffic pattern, and the forwarding pattern. So, conditions (15,16) for accepting new multicast session \((i, n)\) become:

\[
a_{in} + \sum_{k} a_{ik} \leq \frac{F + 1}{2 + P},
\]
\[
a_{in} + \sum_{k} a_{kj} \leq \frac{F + 1}{2 + P},
\]

(17)

(18)

for \(j \in \mathcal{M}_{in}\). In the more general case, only a subset of multicast outputs have enough spare capacity, and they are admitted. Assume that the bandwidth is reserved for multicast session \((i, n)\), and that the admitted multicast group of outputs is \(\mathcal{M}_{in}^n\). Assume that source \(i\) transmits
packets to port $p(i)$, and port $j$ forwards packets to ports $p_k(j)$, $1 \leq k \leq P$. If conditions (17,18) are fulfilled, a new multicast session $(i, n)$ is admitted, and credits are updated like:

$$a_{ip(i)} \leftarrow a_{ip(i)} + a_{in}^m,$$  \hspace{1cm} (19)

$$a_{jp_k(j)} \leftarrow a_{jp_k(j)} + a_{in}^m,$$  \hspace{1cm} (20)

for $j, p_k(j) \in \mathcal{M}_{in}^a$, $1 \leq k \leq P$. Similarly, when the multicast session is released, the following updates are made:

$$a_{ip(i)} \leftarrow a_{ip(i)} - a_{in}^m,$$  \hspace{1cm} (21)

$$a_{jp_k(j)} \leftarrow a_{jp_k(j)} - a_{in}^m,$$  \hspace{1cm} (22)

for $j, p_k(j) \in \mathcal{M}_{in}^a$, $1 \leq k \leq P$. It is also a realistic scenario that one or more ports request to join an already existing multicast session. They will be admitted if (18) is fulfilled. Ports that were the last to receive multicast packets would forward them to the newly admitted ports.

When a forwarding sequence of multicast outputs is pre-determined, 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 according to (19,20), but packets are transmitted only to the first port of the forwarding sequence 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 multicast group 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 still be forwarded by the switch ports. So, the bandwidth should be released according to (21,22) $|\mathcal{M}_{in}^a|$ frames after the termination of the multicast session. Alternatively, the bandwidth reserved for forwarding of multicast packets from the first port in a forwarding sequence, 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 according to (19) in some frame $k$ if conditions (17) and (18) for $j \in \mathcal{P}_1$ hold. Then, port $j \in \mathcal{P}_1$ reserves bandwidth for packet forwarding to ports $j \in \mathcal{P}_2 = \{p_1(j), \ldots, p_P(j)\}$ by updating the assigned credits according to (20) in frame $k + 1$ if conditions (18) for $j \in \mathcal{P}_2$ hold. In general, ports $j \in \mathcal{P}_i$ reserve the bandwidth for packet forwarding to associated ports $j \in \mathcal{P}_{i+1} = \{p_k(j)|j \in \mathcal{P}_i, 1 \leq k \leq P\}$ by updating the assigned credits according to (20) in frame $k + l$ if conditions (18) for $j \in \mathcal{P}_{i+1}$ hold. Each port that reserves bandwidth in some frame stores port addresses from which it will receive packets, and also port addresses to which it will forward packets. This admission process lasts until the bandwidth is reserved for all ports with enough spare capacity. Similarly, if this multicast session ends in frame $t$, input $i$ releases the bandwidth reserved for port $p(i)$ in frame $t$, and ports $j \in \mathcal{P}_1$ release bandwidth reserved for forwarding packets to their associated ports $j \in \mathcal{P}_{t+1}$ in frame $t + l$.

At the beginning of each frame, counters associated with input-output pairs 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 sequential greedy algorithm in which queues with positive counters are served with the priority.

Instead of scheduling individual packets for transmission in each frame, the same algorithm can be used for scheduling flows of packets. Namely, specified time slots within each frame are assigned to each flow. The assigned time slots are released when the multicast session in question is released. As before, each port should store a table of previously reserved time slots in order to schedule new flow of packets.

V. Summary

We proposed a novel method for efficient scheduling and fast bandwidth reservations in high-capacity multicast switches. A set of input-output pairs to be connected in each time slot is determined according to the sequential greedy algorithm based on credits. Pipelined sequential greedy scheduling can be implemented in a switch with a large number of ports, and a high port bit-rate. Multicast packets are forwarded through all ports
to which they are bound. Namely, when some port receives a multicast packet, it will forward this packet to the small number of ports that have not received it yet. In this way, the transmission load of an input port is distributed over all ports that receive packets of the particular multicast session. As a result, the large portion of each port capacity is utilized for any multicast traffic pattern. In the most straightforward implementation, a multicast port forwards packets to the predetermined multicast ports. The forwarding order is determined when the multicast session is admitted. The implied admission controller has only to check if multicast input and outputs have enough of spare capacity. Consequently, the network planning becomes very simple. It should only be planned that all users attached to some port transmit and receive data consuming the specified amount of bandwidth, while the traffic pattern between the ports can be arbitrary. Finally, the pipelined admission of the multicast sessions might provide better utilization of the switching capacity.

It is advantageous to provide bandwidth reservations for the multicast traffic in a high-capacity switch. In the network with a large number of low capacity switches, bandwidth reservations are hindered by the required signalling and high complexity admission control algorithms. Our protocol takes advantage of the high capacity switching fabrics that are about to exceed terabit per second nowadays. Centralized scheduler can make fast decisions and provide agile resource allocation. Its linear design accompanied with the pipeline technique are simple to implement, and allow this scheduler to control a switching fabric with the terabit capacity.

Acknowledgement: I thank to Nick Frigo and Bob Doverspike for their comments.

REFERENCES