Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture

Donpaul C. Stephens and Hui Zhang
Carnegie Mellon University
donpaul, hzhang@cs.cmu.edu

Abstract—**To support the Internet's explosive growth and expansion into a true integrated services network, there is a need for cost-effective switching technologies that can simultaneously provide high capacity switching and advanced QoS. Unfortunately, these two goals are largely believed to be contradictory in nature. To support QoS, sophisticated packet scheduling algorithms, such as Fair Queueing, are needed to manage queueing points. However, the bulk of current research in packet scheduling algorithms assumes an output buffered switch architecture, whereas most high performance switches (both commercial and research) are input buffered. While output buffered systems may have the desired quality of service, they lack the necessary scalability. Input buffered systems, while scalable, lack the necessary quality of service features. In this paper, we propose the construction of switching systems that are both input and output buffered, with the scalability of input buffered switches and the robust quality of service of output buffered switches. We call the resulting architecture Distributed Packet Fair Queueing (D-PFQ) as it enables physically dispersed line cards to provide service that closely approximates an output-buffered switch with Fair Queueing. By equalizing the growth of the virtual time functions across the switch system, most of the PFQ algorithms in the literature can be properly defined for distributed operation. We present our system using a crossbar for the switch core, as they are widely used in commercial products and enable the clearest presentation of our architecture. Buffering techniques are used to enhance the system's latency tolerance, which enables the use of pipelining and variable packet sizes internally. Our system is truly distributed in that there is neither a central arbiter nor any global synchronization. Simulation results are presented to evaluate the delay and bandwidth sharing properties of the proposed D-PFQ system.**

I. Introduction

With the rise of the Internet's popularity, attention is turning to emerging integrated services packet-switched networks to simultaneously support applications with diverse performance objectives and traffic characteristics. In packet switched networks, packets from different sessions belonging to different service and administrative classes interact with each other when they are multiplexed in the network switches. The packet scheduling algorithms within these switches play a critical role in controlling the interactions among different sessions. There has been a lot of research on designing scheduling algorithms that support Quality of Service[23]. Most of it assumes output buffered switches where queueing happens only at output links.

For switches of moderate scale (5 Gbps at current technology), output buffered switches may be constructed and the ideal model can be realized [17]. However, for larger switches, the speeds become too great to implement a shared or output buffered switch. The problem is twofold: First, the memory size and speed at an output card becomes expensive. Second, control logic must operate at a very high rate. These problems will be present for *any* output buffered switch, including one utilizing the FIFO scheduling policy. In order to reduce cost and simplify implementation, most high performance switches (both research [15] and commercial [6], [7]) have chosen architectures employing input buffering. While these additional queueing points are required to enable cost-effective system scalability, the behavior of the traffic patterns internal to the switch can detract from the output scheduler's ability to provide service guarantees. This is because packets that should be transmitted out a link can be temporarily blocked at the switch inputs. Another problem facing practitioners is that most research on designing scalable systems has been for switches servicing fixed sized cells. However, there is a great need for high performance switches that support variable length packet networks, including IP and Ethernet.

In this paper, we propose mechanisms for providing service that closely approximates an output buffered switch employing Fair Queueing. Besides supporting QoS, these mechanisms achieve two additional goals: efficiently supporting variable sized packets and maximizing throughput in a switch core. The resulting Distributed Packet Fair Queueing (D-PFQ) architecture requires no global synchronization and can be pipelined for high speed operation. We describe our system in the context of a network utilizing Ethernet frame sizes. Simulation results are presented to illustrate the bandwidth distribution and delay properties of D-PFQ.

The rest of the paper is organized as follows. We first review issues in designing high performance switches and PFQ algorithms in Section II. We then describe the design of our D-PFQ switch architecture in Section III and discuss some of its properties in Section IV. Simulation experiments are presented in Section V to evaluate the delay and bandwidth performance of D-WF^{2}Q, D-WF^{2}Q+, D-SFQ, and D-SCFQ.

II. Background

A. Switching

To support QoS guarantees, advanced scheduling algorithms have been proposed that manage queues on a per-session basis. In particular, a class of Packet Fair Queueing algorithms [4], [8], [9], [10], which can be used to support both guaranteed and adaptive services, have received a lot of attention. However, most of these algorithms are described in the context of an output buffered system. While output buffering may be the ideal system in terms of providing service guarantees, it is not practical for construction of switches with large aggregate switching capacity. The reason is that an output buffered system needs to buffer all packets arriving to it at any time. Because these arrivals may

This research was sponsored by DARPA under contract numbers N66001-96-C-8528 and N00174-96-K-0002, and by a NSF Career Award under grant number NCR-9624979. Additional support was provided by Intel Corp., MCI, and Sun Microsystems. Views and conclusions contained in this document are those of the authors and should no be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, MCI, Sun, or the U.S. government.

occur simultaneously from many input ports, the output buffer needs sufficient memory bandwidth to enqueue traffic at a much higher rate than a single port may dequeue it. In the worst case, N (the number of line cards in the switch) packets could arrive in the amount of time a port could send one. This requires that the memory bandwidth and control systems speed to scale as a function of the number of cards in the switch, which places stringent limits on the system size.

Since it is impractical to build large switches with output buffering, most high speed switches utilize some form of input buffering. By having buffers at input ports, it is possible to build high performance switches with speedup, which is defined to be the ratio of the line card's bandwidth into/from the switch core to the link speed, much smaller than N.

Buffering at the input changes the contention problem inside the switch. While contentions only happen at output links in an output buffered switch, they also happen at input and output cards [1] in an input buffered switch – multiple packets from the same input card may be destined to the different output cards and multiple packets from different input cards may be destined to the same output card. If the input buffer is FIFO, there is no contention at input cards, but it introduces the problem of Head-of-Line (HOL) blocking [11]: if the packet at the head of the queue is blocked due to contention of the output card, packets that are on the same input card but destined to other contention-free output cards cannot be forwarded. By maintaining at an input card a separate queue for each output card [1], the HOL problem can be eliminated. Additional flexibility can be obtained by having buffering at both input and output cards [15], [6], [7], [19], [5]. For switch architectures that employ input-buffering and use a relatively small speedup, there is still the issue of how to resolve contentions at input and output cards. Most of the research in the literature has focused on designing switch scheduling or matching algorithms to maximize the throughput of the switch [1], [19], [14], [5]. Relatively little has been done to study how to provide QoS in switches with input buffering and modest speedup.

B. Packet Fair Queueing

Fair Queueing algorithms are packet algorithms that approximate the fluid GPS system. A GPS with N sessions is characterized by N positive real numbers, $\phi_1, \phi_2, \cdots, \phi_N$. During any time interval when there are exactly M non-empty queues, the server serves the M packets at the head of the queues simultaneously, in proportion to their service shares.

Each Fair Queueing algorithm maintains a system virtual time $V(\cdot)$. In addition it associates with each session i a virtual start time $S_i(\cdot)$, and a virtual finish time $F_i(\cdot)$. Intuitively, $V(t)$ represents the normalized fair amount of service time that each session should have received by time t, $S_i(t)$ represents the normalized amount of service time that session i has received by time t, and $F_i(t)$ represents the sum of $S_i(t)$ and the normalized service that session i should receive for serving the packet at the head of its queue. Since $S_i(t)$ keeps track of the service received by session i by time t, $S_i(t)$ is also called the virtual time of session i, and alternatively denoted $V_i(t)$. The goal of all PFQ algorithms is then to minimize the discrepancies among $V_i(t)$'s and $V(t)$.

$$S_i(t) = \begin{cases} max(V(t), S_i(t-)) & i \text{ becomes active} \\ S_i(t-) + \frac{L_i^k}{r_i} & p_i^k \text{ finishes service} \end{cases} \quad (1)$$

$$F_i(t) = S_i(t) + \frac{L_i^k}{r_i} \quad (2)$$

While all Fair Queueing algorithms use (1) to update $S_i(\cdot)$ and $F_i(\cdot)$, they differ in two aspects, the computation of the system virtual time function, and the packet selection policy. The role of the system virtual time is to reset $V_i(\cdot)$ whenever an unbacklogged session i becomes backlogged again. Examples of system virtual time functions are the start time of the packet being currently served [10], the finish time of the packet being currently served [9], and a monotonically increasing function that is at least the minimum of the start times of all packets at the head of currently backlogged queues [3]. Examples of packet selection policies are: Smallest Start time First (SSF) [10], Smallest Finish time First (SFF) [9], and Smallest Eligible Finish time First (SEFF) [3], [21]. The choice of different system virtual time functions and packet selection policies will affect the real-time and fairness properties of the resulting PFQ algorithm.

As will be discussed later in the paper, we will use multiple Fair Queueing servers distributed among input and output ports (distributed Fair Queueing) to approximate the service of a single Fair Queueing server at the output port. The key difference between a D-PFQ and a single PFQ server is that with a centralized PFQ, a single system virtual time is used to reset the per session virtual times for *all* sessions, therefore, keeping all $V_i(\cdot)'s$ synchronized. Whereas in a D-PFQ system, sessions may use different system virtual times to reset their per session virtual times, there may be discrepancies among the session virtual times due to the discrepancies of system virtual time functions. The solution, as discussed in detail later, is then to account for the discrepancies of the system virtual time functions during scheduling.

III. Distributed Switch Architecture

Having reviewed the present state of the art in switch designs and the mechanisms by which Fair Queueing operates, we now turn our attention to the design of a switch architecture capable of scaling to large size (measured in terms of aggregate switching capacity) for variable sized packets while retaining the important properties of Fair Queueing.

Our architecture uses a crossbar for interconnecting the cards on which ports reside. Despite the N^2 complexity, a crossbar is actually the most popular switch core currently used by both commercial vendors and researchers investigating very high speed switching. For commercial systems, the most important aspect of scalability is aggregate switching capacity, not physical ports. If a switch can have a few tens of very high speed cards, it is relatively easy to construct multiplexors and demultiplexors for connecting lower speed ports to the switch.

Buffers are placed at all three potential contention points: at the inputs, at the outputs, and within the crossbar. While having

[1] It is common for large switches to have cards that connect many slower speed external ports to an internal card. Thus we use "card" to refer to the internal organization, and "port" to refer to the external connections.

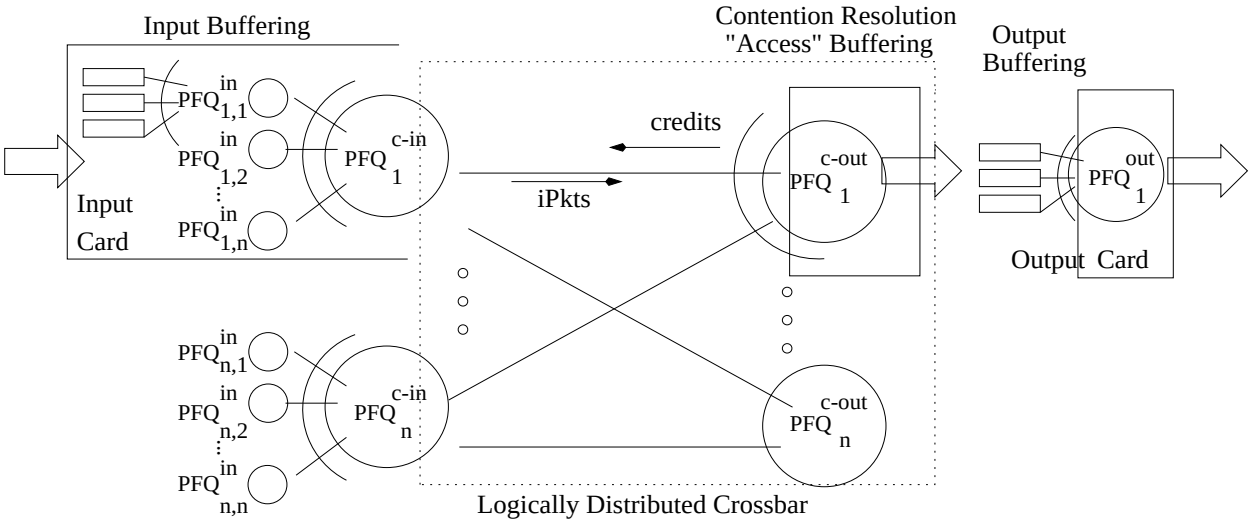

Fig. 1. Distributed Switch Architecture

I_i	system virtual time function for PFQ_i^{c-in}
O_j	system virtual time function for PFQ_j^{c-out}
$V_{i,j}$	system virtual time function for $PFQ_{i,j}^{in}$
$x_{i,j}$	offset between $X_{i,j}$ and $V_{i,j}$
$y_{i,j}$	offset between $Y_{i,j}$ and $V_{i,j}$
$X_{i,j}$	input normalized virtual time function for $V_{i,j}$
$Y_{i,j}$	output normalized virtual time function for $V_{i,j}$
$iMTU$	the MTU used internal to the switch core

TABLE I
NOTATION USED IN THE PAPER

separate buffers for different output cards at each input card can achieve 100% switch throughput without substantial speed-up at either the input or output buffer [13], it is not sufficient to support per session QoS guarantee. In our architecture, there are per session queues at the input card. Within the crossbar, there is buffering for each input card associated with every output card. These access buffers are made relatively small, so that it is practical that they may be placed on the crossbar chips themselves. A simple credit flow control mechanism is used to manage access to the access buffers. An input may send data as long as sufficient space is available to send one maximum sized (internal) packet. When packets are forwarded from the switch core to the output, a credit flows back to the input card to inform it of the space that was freed. Per-session queues are used once again at the output. This distributed buffering architecture is shown in Figures 1 and 2.

To provide quality of service guarantees, a fair queueing server will be placed at every arc on these figures. Intuitively, an input-output buffered switch differs from an ouput-buffered switch in that it needs to manage not only the bandwidth of the output link, but also the access to the switch core from input cards and the access to output cards from the switch core. While an output buffered switch needs only N PFQ servers, one on each output card, an input-output buffered switch needs additional PFQ servers to manage the additional queueing points.

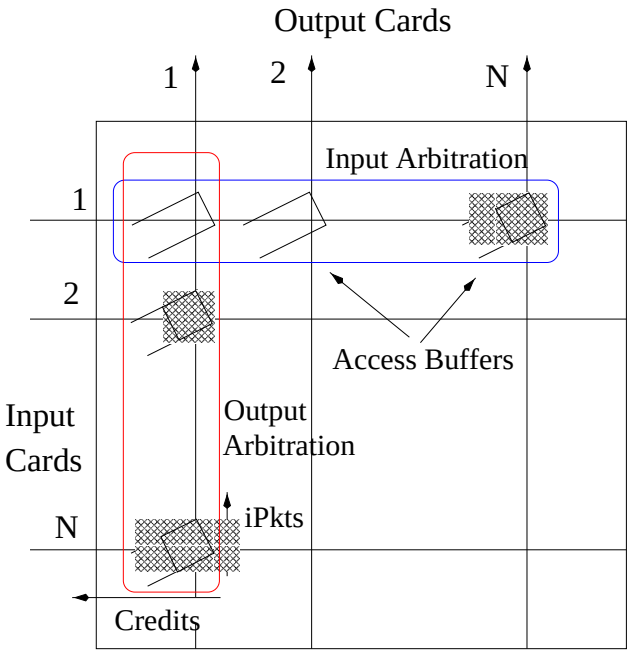

Fig. 2. Crossbar Internals

At each input card i, there are N PFQ servers, one for each output card. We refer to the PFQ server at input card i for output card j as $PFQ_{i,j}^{in}$. Therefore, there are total N^2 $PFQ_{i,j}^{in}$ servers on all input cards, $i = 1, \cdots, N, j = 1, \cdots, N$. In addition, there are 2N PFQ servers inside the core, arbitrating the access into and out of the access buffers on the switch core. We refer to them as $PFQ_i^{c-in}, i = 1, \cdots, N$, and $PFQ_j^{c-out}, j = 1, \cdots, N$. Finally, there are N PFQ servers managing the output link bandwidth, one on each output card. We refer to them as PFQ_j^{out}. While this architecture seems to be rather complex as there are conceptually $N^2 + 3N$ PFQ servers, in practice only $3N$ PFQ servers are needed, as the N $PFQ_{i,j}^{in}$ servers, $j = 1, \cdots, N$, and the one PFQ_i^{c-in} server on input card i can share the same Fair Queueing engine. In addition, since per session state is only kept on the input and output cards that are

traversed by the session, the total amount of control state that needs to be maintained by all the PFQ servers in our switch is in proportion to the total number of sessions passing through the switch, comparable to that maintained by an output-buffered switch.

For each output card j, the N $PFQ_{i,j}^{in}$ servers ($i = 1, \cdots, N$) on the N input cards and PFQ_j^{c-out} collectively manage all the sessions arriving from different input cards that are destinated for the output card j . The key design issue is then how to coordinate these $N+1$ PFQ servers so that they can emulate one PFQ server in an output buffered switch. In particular, in the case of a single PFQ server in an output buffered switch, the system virtual time $V_j(\tau)$ is used to represent the normalized fair amount of service that *all* backlogged sessions at output j should have received by time τ. It is used to re-initialize the per session virtual time when the session becomes backlogged. However, in the case of an input-output buffered switch like ours, all sessions destined to the same output card j are physically distributed among N input cards. Since we use $N + 1$ PFQ servers to manage these sessions and they all have different system virtual times, in order to emulate the service of a single PFQ server, there is a need to normalize these $N + 1$ system virtual times so that they can be compared.

Let $V_{i,j}(\cdot)$ and $O_j(\cdot)$ denote the system virtual time functions for $PFQ_{i,j}^{in}$ and PFQ_j^{c-out} servers respectively. The output (into output card j from the switch core) normalized virtual time for $V_{i,j}(\cdot)$, which is used by PFQ_j^{c-out} for arbitrating the access to the acess buffer to output card j among all input cards, is:

$$Y_{i,j}(t) = V_{i,j}(t) + y_{i,j}(t) \tag{3}$$

where $y_{i,j}(t)$ is the offset value that is maintained at the access buffer to output port j. $y_{i,j}(t)$ is updated according to the following when a packet that is from input card i and destined to output card j arrives at an *access buffer*:

$$y_{i,j}(t) = \begin{cases} y_{i,j}(t) & if\ V_{i,j}(t) + y_{i,j}(t) \geq O_j(t) \\ O_j(t) - V_{i,j}(t) & if\ V_{i,j}(t) + y_{i,j}(t) < O_j(t) \end{cases} \tag{4}$$

i.e., if the old offset would place the normalized timestamp $Y_{i,j}(t)$ before the current output virtual time $O_j(\cdot)$, the offset is changed so that they are now equal.

Conceptually, this is a PFQ server using $O_j(\cdot)$ as the system virtual time function, and servicing N queues that correspond to each input card. The normalized virtual times $Y_{i,j}(t)$ are analogous to session virtual times in that they are reset to the system virtual time value when arriving with a lower value. Since the actual values themselves cannot be modified, the server "normalizes" them into a timespace in which they can be. The goal of the algorithm is to minimize the discrepancies among these normalized timestamp values. The algorithm does this by servicing queue i (access buffer from input card i to output card j) with the smallest $Y_{i,j}$ value.

While the above algorithm addresses the issue of arbitration of the access out of the switch core and into each output card, there is still the issue of the arbitration of the access into the switch core from each input card. In our design, these two problems are symmetric and we apply the same technique.

Let $I_i(\cdot)$ be the system virtual time function of the PFQ_i^{c-in} server. The input (into the switch core from the input card i) normalized virtual time for $V_{i,j}(\cdot)$ is:

$$X_{i,j}(t) = V_{i,j}(t) + x_{i,j}(t) \tag{5}$$

where $x_{i,j}(t)$ is the offset value that is maintained at input card i. $x_{i,j}(t)$ is updated according to the following when (a) the previously filled access buffer has drained and has space to accept more packets, which is triggered by a credit packet flowing from the access buffer to the input card, or (b) a packet destined for output j arrives at input i, there are no packets queued in input i for output j, and there is enough space in the switch access buffer to accept an internal packet from input i to output j,

$$x_{i,j}(t) = \begin{cases} x_{i,j}(t-) & V_{i,j}(t) + x_{i,j}(t-) \geq I_i(t) \\ I_i(t) - V_{i,j}(t) & V_{i,j}(t) + x_{i,j}(t-) < I_i(t) \end{cases} \tag{6}$$

The updates are analogous to those for the output's normalizing offset. The symmetry of the problem results from the use of the access buffers on the crossbar. They are a clean level of abstraction that can be used by the input to know that the output is constrained (which occurs when the buffers are depleted), and by the output to know that the input is constrained (which occurs when the buffers are empty).

In summary, we propose a distributed switch architecture with buffering at the input, output and crossbar. PFQ servers are used to manage all queueing and contention points in the system. To approximate the service of a single PFQ server in an output buffered switch using multiple distributed PFQ servers, it is important to advance the virtual times of the set of PFQ Fair Queueing servers in unison. This is achieved by normalizing the system virtual time functions at different PFQ servers.

IV. Discussion

In our distributed switch architecture, the problem of controlling the crossbar can be divided into two separate tasks:

1. each of the N outputs *independently* reads from *any* input that has a packet in its access buffer, and
2. each of the N inputs *independently* sends packets to *any* output with an access buffer that has sufficient buffering for an internal packet.

Such an architecture clearly separates the different contention points both within the switch core and at its periphery that *any* distributed scheduling algorithm will have to address. The outputs select which input card to serve within the crossbar. The inputs select which output to serve and which session for that output to serve, as they have the most complete knowledge of how to meet the respective guarantees. We believe that a system that does not require completely synchronous operation of all end cards enables higher performance due to latency tolerance.

The distributed nature of the architecture also eliminates the requirement in a centralized switching system that fixed size packets be used internally for switching purposes, thus significantly reducing the speedup penalty associated with these switches when they are used in variable-size packet networks such as Ethernets and IP networks.

The speedup of a system is the ratio of the line card's bandwidth to/from the switch core to the link speed. A speedup higher

than 1 may be needed for two reasons: (1) reducing contentions at the input and output cards; (2) compensating for the bandwidth lost due to mismatch between the size of internal packets used by the switch core and the size of packets used on the wire.

To illustrate the second aspect, consider a variable-size packet system and a switch core using fixed-size packets of L bytes internally. If all packets arriving at the switch have a length of $L + 1$ bytes, it will take two internal packets to switch one packet on the wire. To sustain the link speed, the speedup has to be $\frac{2L}{L+1}$ or approximately 2x. Since a higher speedup incurs a higher cost, we would like to minimize the speedup penalty that is due to the size mismatch between internal and external packets. Notice that this speedup penalty does not exist in a fixed size packet system such as ATM networks. It only happens in a variable-size packet system with switches employing internal fixed packets.

Since all switches employing input-buffered switches we are aware of use some form of centralized matching algorithms and all centralized switch scheduling or matching algorithms impose the restriction that the data be switched in units of fixed size, all existing input-buffered switches need to pay a worst-case 2x speedup overhead penalty when used in variable size packet networks.

Since no centralized matching algorithm is needed in our D-PFQ switch, we can use variable-size packets called $iPkts$ inside the switch core to eliminate the speedup penalty due to size mismatch between internal and external packets. We note that most crossbar switch fabrics can support variable-size packets. An interesting design issue is how to choose the MTU for the internal packet, or iMTU. There are two design goals: (a) we want to minimize the amount of buffer needed inside the core. Since each access buffer needs to have space to store at least two internal packets, one for receiving from the input and one for sending to the output, a small iMTU is desirable. And (b), since each internal packet has some header that contains control information such as routing information, there will be additional speedup required to account for this header overhead. We would like to minimize this speedup overhead while being able to keep up with the full link speed for a wide range of packet sizes. Combining these two design goals, we want to find the minimum size iMTU such that the speedup overhead is minimized while the switch is still able to support full link speeds for wide range of packet sizes.

Given an iMTU (excluding the header length, which is assumed to be H) the worst case arrival pattern for the switch is when the length of all packets is iMTU + 1, in which case two internal packets need to be sent for each external packet, with sizes (including header) H + iMTU and H + 1 respectively. The speedup needed to achieve full line speed in this case is:

$$\frac{iMTU + 1 + 2H}{iMTU + 1} \tag{7}$$

Another special arrival pattern is when the length of all packets is mTU, the *minimum* packet size allowed on the wire. Assuming $iMTU \geq mTU$, only one internal packet of length iMTU + H is needed for each external packet in this case. The speedup

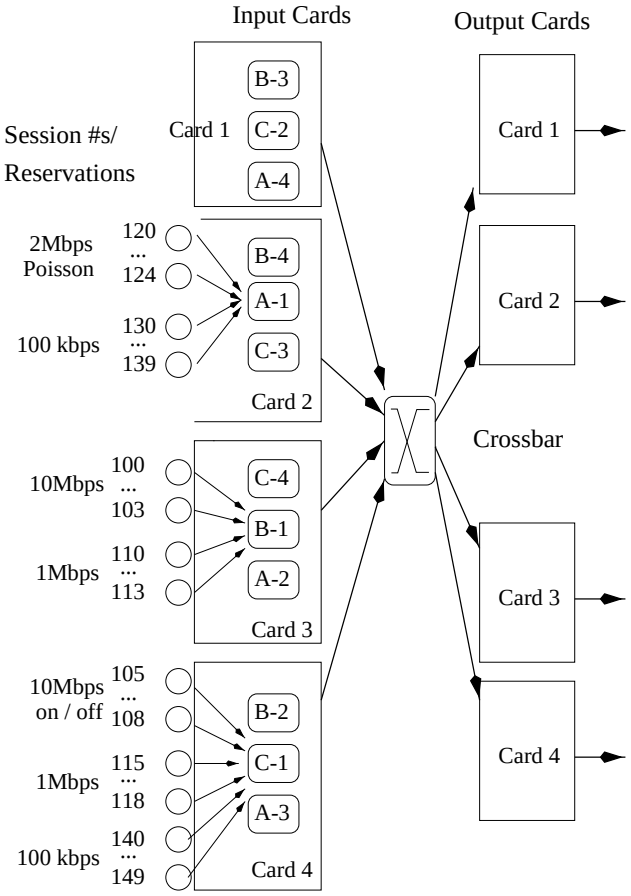

Fig. 3. Switch Configuration

needed to achieve full line speed is:

$$\frac{mTU + H}{mTU} \tag{8}$$

Since $\frac{mTU+H}{mTU}$ is a lower bound for the speedup needed to achieve full link speed for any packet arrival patterns, if we can choose an $iMTU$ such that

$$\frac{iMTU + 1 + 2H}{iMTU + 1} \leq \frac{mTU + H}{mTU} \tag{9}$$

then we get the minimum iMTU such that even with the minimum speedup ($\frac{mTU+H}{mTU}$), the switch can forward packets at link speed for any packet sizes. Solving this inequality shows that the minimum iMTU is 2mTU - 1.

We believe that the distributed D-PFQ switch architecture is relatively simple to implement. Such a control system should be compatible with architectures such as [15]. In fact, the communication between the control logic and the input cards is actually reduced as compared to typical switch desgins. Because the data no longer needs to be provided with strict timing constraints, the process can be pipelined, thus enabling higher speed operation. The distribution of control has numerous other benefits for robust systems. The price of these features is N^2 buffers, where N is number of line cards. However, this cost will be incurred on the crossbar chips, whose size is limited by the pincount, not by the internal silicon area [19].

SessionIDs	Traffic type	Reserved rate	Arrival rate	Packet length	WFQ Delay bound	Bandwidth (idle on/off)
100 – 103	CBR	10 Mbps	20 Mbps	1250 Bytes	N/A	17.9 Mbps
105 – 108	On/Off	10 Mbps	10 Mbps	1250 Bytes	1 ms	N/A
110 – 113	CBR	1 Mbps	1 Mbps	1250 Bytes	10ms	1Mbps
115 – 118	CBR	1 Mbps	1 Mbps	1250 Bytes	10ms	1Mbps
120 – 124	Poisson	2 Mbps	6 Mbps	500 Bytes	N/A	3.58Mbps
130 – 139	CBR	100 kbps	100 kbps	100 Bytes	8ms	100kbps
140 – 145	CBR	100 kbps	100 kbps	100 Bytes	8ms	100kbps
145 – 149	CBR	100 kbps	2 Mbps	100 Bytes	N/A	179kbps

TABLE II
TRAFFIC SOURCES

V. SIMULATION EXPERIMENTS

In this section, we present simulation experiments to evaluate the delay and bandwidth properties of the proposed D-PFQ switch architecture. We assume a switched Ethernet environment with packet sizes between 64 to 1500 Bytes and a link speed of 100 Mbps. Internal overheads are 8 Bytes per variable sized iPkt with an iMTU of 128 Bytes. As discussed in Section IV, to switch packets at full link speed, a minimum speedup of $\frac{64+8}{64} = 1.125$ is needed. The speedup overhead of 12.5% is a direct consequence of the internal packet header overhead. Keeping with convention to state internal speedup with respect to the amount of external bandwidth that can pass internally, we refer to this as a 1.0x speedup. To reduce the input/output contention to/from the switch core, additional speedup may be needed.

All experiments were performed using a 4x4 switch architecture as illustrated in Figure 3. For simplicity, we set up three session groups, labeled as "A", "B", and "C". Each input and output card receives sessions from all groups. For example, the block "B-3" at input card 1 in the Figure represent the session group B destined for output 3. The net effect is that for given any input/output card and any session in the session groups, the session traverses the card exactly once. In the Figure, we only depicts the sessions that are destined to the output card 1. Table II lists the source characteristics of the sessions. As can be easily verified, the sum of the reserved rates for all sessions in three traffic groups is 100 Mbps. Most sessions are CBR sources that transmit according to their reserved rates. The exceptions are (a) sessions 100-103, which are CBR sources that have a reservation of 10 Mbps, but transmit at a higher rate of 20 Mbps, (b) sessions 105-108, which are on-off sources with an on/off period of 1 second and that transmit at a reserved rate of 10 Mbps during the on period, and (c) sessions 120-124, which are Poisson sources that transmit at an average rate of 6 Mbps, much higher than their reserved rate of 2 Mbps.

An observing reader may notice that the sum of the rates arriving at an input card is actually higher than 100 Mbps, which is impossible in a switch with a link speed of 100 Mbps. In the simulation, we set the input link speed to be 200 Mbps and the output link speed to be 100 Mbps. The reason we set up the experiment this way is to create a scenario in which all output links being observed are overloaded. This setup can be viewed as an emulation of an 8x8 switch in which only four output cards are overloaded.

For each session, Table II also lists the delay bound that could be provided by WFQ in an output buffered switch, and the bandwidth that the session should ideally achieve when the on-off sources are not transmitting. When the on-off sources are transmitting, the output link has the entire capacity reserved by the member sessions, and each session should ideally receive the service of its reserved rate.

For all the experiments, we evaluate the delay and bandwidth performance provided by the proposed D-PFQ system. For delay performance, we plot delay distributions of source-constrained sessions in the presence of bursty and overloading cross traffic. In particular, we choose three sessions: 105, which is an on-off source with 1 second on/off period and transmits at a rate of 10 Mbps during the on period, 110 and 130, which are CBR sources that transmit at 1 Mbps and 100 Kbps respectively. For bandwidth performance, we plot bandwidth distribution of bursty and unconstrained sessions in the presence of on-off sources. In this case, we choose three sessions: 100, which is a CBR source with a reservation of 10 Mbps but transmitting at 20 Mbps, 105, which is an on/off source, and 120, which is a Poisson source with a reserved rate of 2 Mbps but transmitting at an average rate of 6 Mbps. In bandwidth distribution Figures, the bandwidth is averaged over non-overlapping time intervals of 10 ms.

We evaluate the performance of our system using four different types of Packet Fair Queueing algorithms: WF^{2}Q [4], WF^{2}Q+ [3], SCFQ [9], and SFQ [10]. WF^{2}Q uses the system virtual time function of GPS, which requires emulation of GPS, and the SEFF packet selection policy. It has been shown that WF^{2}Q is the most accurate packet approximation algorithm of GPS. The other three algorithms use system virtual time functions that are computed solely based on the state in the packet system (self-clocked) without emulating the fluid GPS system. While these algorithms provide delay bounds and fairness similar to those provided by GPS, their system virtual time functions may deviate from the ideal GPS virtual time function over a short time interval.

The first set of experiments were run using a 1.0x speedup. Figures 4 and 5 show the delay and bandwidth distributions. As can been seen, only D-WF^{2}Q performs well in this case. This is because the accuracy of the virtual times at the input cards is critical with a low speed-up.

The second set of experiments were run using a 1.25x speedup. Figures 6 and 7 show the delay and bandwidth distributions. As

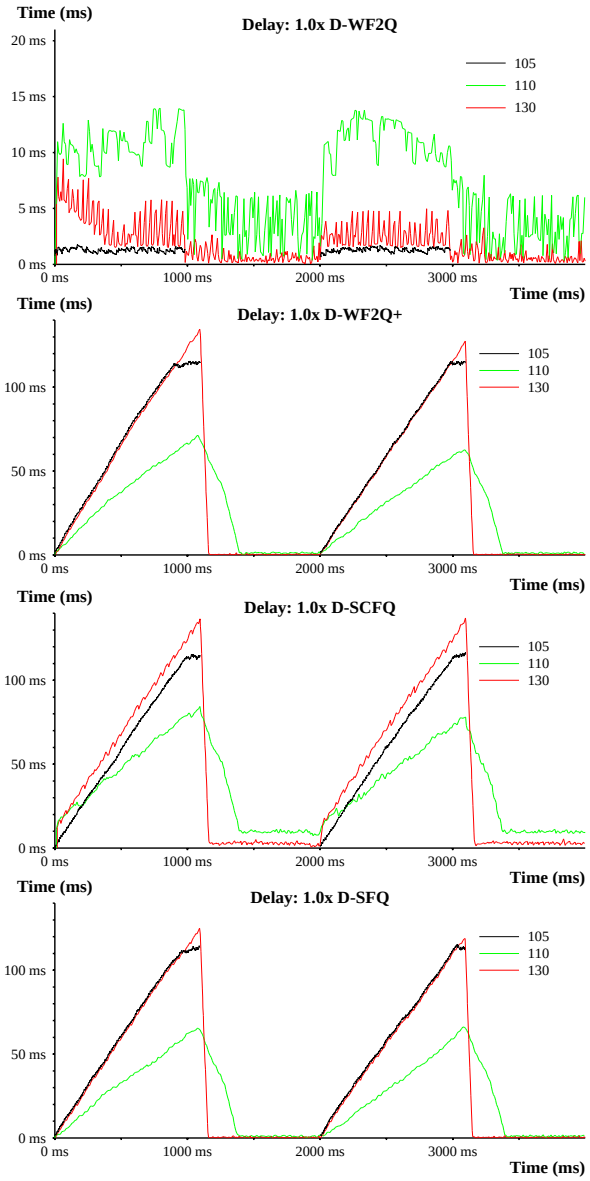

Fig. 4. Delay plots w/ 1.0x speedup

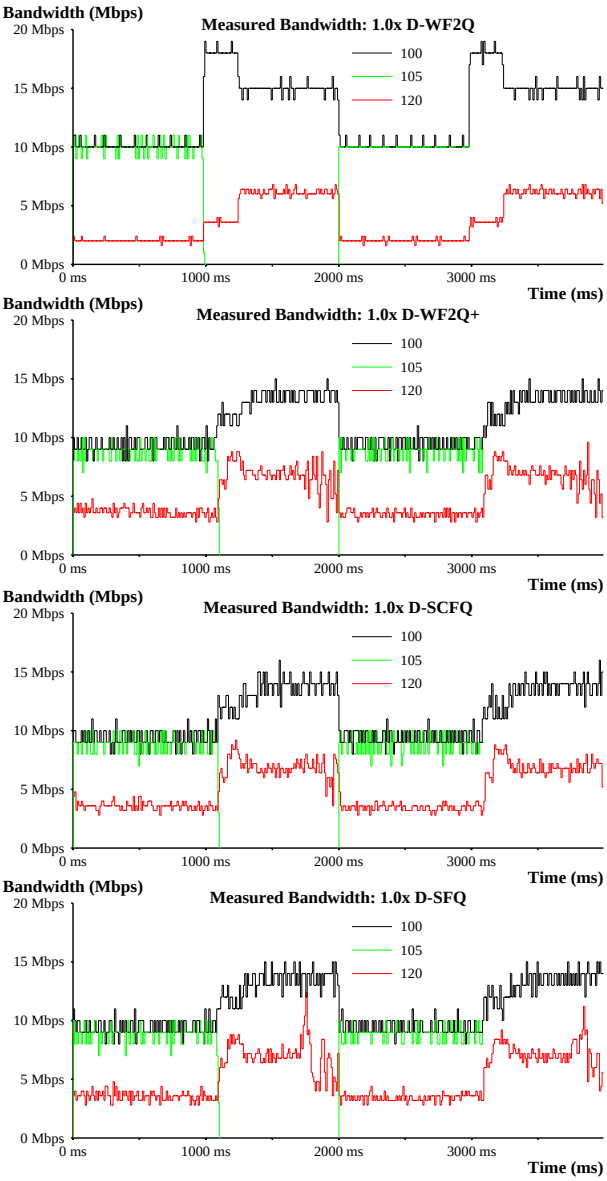

Fig. 5. Bandwidth plots w/ 1.0x speedup

can been seen, all four systems perform reasonably well. Packets of session 110 experience higher delay in a D-SCFQ system than in D-WF^{2}Q and D-WF^{2}Q+ systems. This is because even with a standalone PFQ system, SCFQ provides a higher delay bound than WF^{2}Q and WF^{2}Q+. D-SFQ provides lower delay for lower rate sessions 110 and 130 while higher delay for higher rate session 105 compared to the other algorithms. Again, this is consistent with the case of standalone PFQ servers.

In summary, with no speedup, for D-PFQ to achieve good performance it is critical for the component PFQ algorithms to have accurate virtual time functions. With modest speedup, the proposed D-PFQ architecture can closely approximate the performance of an output buffered switch that implements any of the Packet Fair Queueing algorithms discussed here.

VI. Related Work

The separation of switch scheduling from data forwarding was proposed by Anderson et al [1]. With the objective of high system throughput, they proposed Parallel Iterative Matching (PIM) to quickly pair inputs and outputs using a sequence of grants and acknowledgements. Their crossbar is bufferless, thus requiring the matching to be synchronized in time. A static scheduling table is used to support CBR traffic.

A discussion of the benefits of using a switch core similar to ours with input, output, and internal buffering may be found in [5]. We believe our techniques can be easily augmented to such an architecture to enable it to provide QoS guarantees.

Lund, Phillips, and Reingold proposed a mechanism for providing per-VC fairness in [12]. Their system employs both input and output buffering with a speedup of approximately 20% in a crossbar switch using a derivative of PIM for performing matching. The resulting system closely approximates a prioritized round robin. Conceptually, FARR is the closest to our work in that it uses a timestamp instead of a weight for scheduling among the input nodes. However, the nature of the timestamp comparison dictates the system be synchronized, and bandwidth

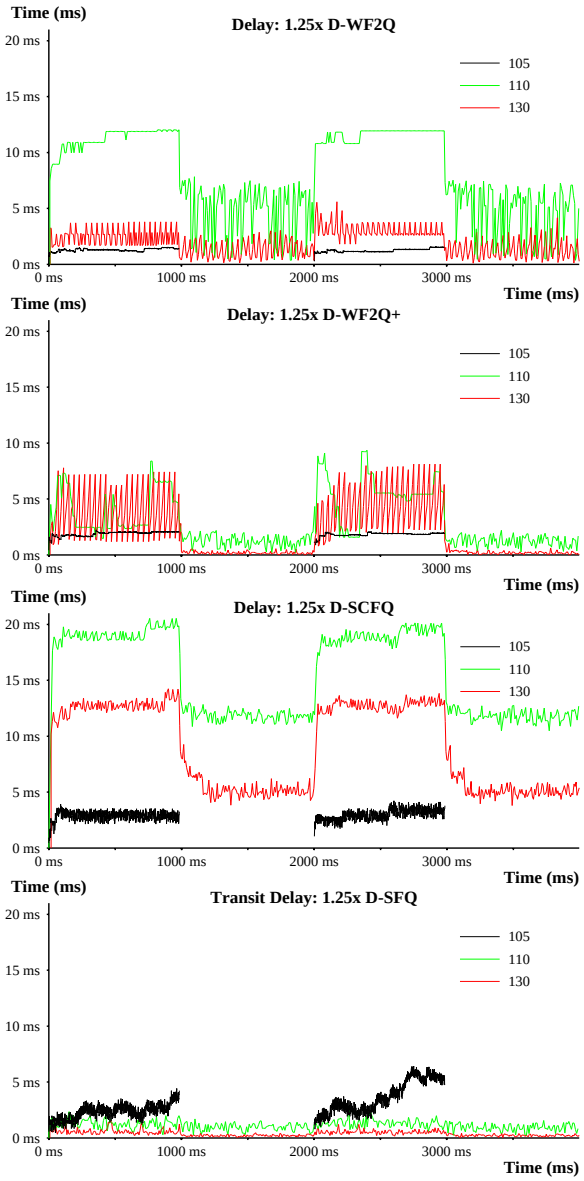

Fig. 6. Delay plots w/ 1.25x speedup

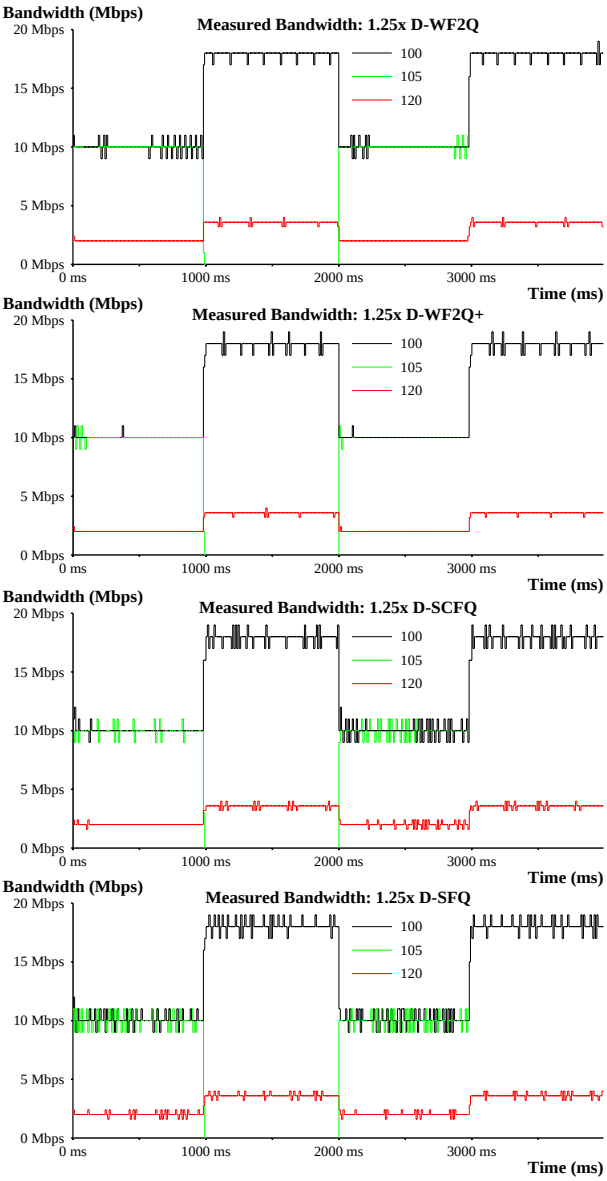

Fig. 7. Bandwidth plots w/ 1.25x speedup

guarantees are not provided. In addition, it is designed solely for ATM.

Stilliadis and Verma proposed a scheme to provide bandwidth guarantees in an input buffered crossbar switch [20]. Their system employs a variant of PIM called WPIM (Weighted Probabilistic Iterative Matching) that attempts to perform a maximal match with the selection probability weighted by the sum of bandwidth reservations between the line cards. While a mechanism using a static weight assignment should be able to provide minimum bandwidth allocation, the effects of a conflict in the matching are not clear. Furthermore, the distribution of excess bandwidth is dependent on the input card when it is not the bottleneck.

A survey on switch design issues may be found in [2]. Hop-by-hop flow control algorithms, such as the one we employ, need to pay careful attention to details of synchronization and loss. These issues and their solutions are described in [18]. The issue of performing multicast in input buffered switches has been studied in [22], [16].

VII. Conclusion

We have presented the design of a switch that can both scale to high switching capacity and provide advanced QoS. We have made several contributions in this paper. First, we have developed mechanisms that enable the construction of a high capacity switch without a central arbiter to be tolerant of relatively high internal latency. These mechanisms enable both higher system throughput, and the use of variable sized packets internal to the switch core. The resulting reductions in both fragmentation and internal overhead makes possible for the switch to support full link speed for a wide range of packet sizes while running at a modest speedup. Secondly, we propose techniques to use multiple Fair Queueing servers distributed among input and output cards to approximate the service of a single Fair Queueing server at the output port. The key is to ensure that the virtual times of the set of distributed Fair Queueing servers advance in

unison. However, no global synchronization is required, as the core arbiters operate on the growth of the component virtual time functions, not their actual values. Results from simulation experiments are presented to demonstrate that the resulting system provides service that closely approximates an output buffered switch employing Fair Queueing with modest speedup.

VIII. Acknowledgement

We would like to thank Anna Charney, Tom Des Jardins, Andrew Myers, Jennifer Rexford, and the anonymous reviewers for providing comments on earlier drafts.

References

[1] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High speed switch scheduling for local area networks. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, October 1992.

[2] R. Y. Awdeh and H. T. Mouftah. Survey of ATM switch architectures. *Computer Networks and ISDN Systems*, pages 1567–1613, September 1995.

[3] J.C.R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of the ACM-SIGCOMM 96*, pages 143–156, Palo Alto, CA, August 1996.

[4] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, San Francisco, CA, March 1996.

[5] F.M. Chiussi, Y. Xia, and V.P. Kumar. Backpressure in shared-memory-based atm switches under multiplexed bursty sources. In *Proceedings of IEEE INFOCOM'96*, pages 830–843, San Francisco, CA, March 1996.

[6] Ascend Communications. GRF family of switches. www.ascend.com.

[7] Digital Equipment Corporation. GIGAswitch. www.networks.digital.com.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[9] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, CA, June 1994.

[10] P. Goyal, H.M. Vin, and H. Chen. Start-time Fair Queuing: A scheduling algorithm for integrated services. In *Proceedings of the ACM-SIGCOMM 96*, pages 157–168, Palo Alto, CA, August 1996.

[11] M. J. Karol, M. G. Hluchyj, and S. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, 35(12):1347–1356, December 1987.

[12] C. Lund, S. Phillips, and N. Reingold. Fair prioritized scheduling in an input-buffered switch. In *Proceedings of Broadband Communications '96*, 1996.

[13] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE INFOCOM'96*, San Francisco, CA, March 1996.

[14] N. McKeown and T. E. Anderson. A quantitative comparison of scheduling algorithms for input queued switches. To Appear.

[15] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick, and M. Horowitz. The tiny tera: A packet switch core. *IEEE Micro*, pages 26–33, January 1997.

[16] N. McKeown and B. Prabhakar. Scheduling mulitcast cells in an input-queued switch. In *Proceedings of IEEE INFOCOM'96*, San Francisco, CA, March 1996.

[17] MMC Networks. ATMS2200 XStream per-flow queuing chip set. www.mmcnet.com.

[18] C. Ozveren, R. Simcoe, and G. Varghese. Reliable and efficient hop-by-hop flow control. *IEEE JSAC*, 13(4):642–6502, May 1995.

[19] R. Simcoe and T. Pei. Perspectives on ATM switch architecture and the influence of traffic patterns on switch design. *Computer Communication Review*, 25(2):93–105, April 1995.

[20] D. Stilliadis and A. Verma. Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch. In *Proceedings of INFOCOM 95*, Boston, MA, April 1995.

[21] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. A proportional share resource allocation for real-time, time-shared systems. In *Proceedings of the IEEE RTSS 96*, pages 288 – 289, December 1996.

[22] J. Turner. An optimal nonblocking multicast virtual circuit switch. In *Proceedings of IEEE INFOCOM'94*, pages 298–305, June 1994.

[23] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1399, October 1995.