FIFO-Based Multicast Scheduling Algorithm for Virtual Output Queued Packet Switches

Deng Pan, *Student Member*, *IEEE*, and Yuanyuan Yang, *Senior Member*, *IEEE*

Abstract—Many networking/computing applications require high speed switching for multicast traffic at the switch/router level to save network bandwidth. However, existing queuing-based packet switches and scheduling algorithms cannot perform well under multicast traffic. While the speedup requirement makes the output queued switch difficult to scale, the single input queued switch suffers from head of line (HOL) blocking, which severely limits the network throughput. An efficient yet simple buffering strategy to remove the HOL blocking is to use the virtual output queued (VOQ) switch structure, which has been shown to perform well under unicast traffic. However, the traditional VOQ switch is impractical for multicast traffic because a VOQ switch for multicast traffic has to maintain an exponential number of queues in each input port (i.e., $2^N - 1$ queues for a switch with N output ports). In this paper, we give a novel queue structure for the input buffers of a multicast VOQ switch by separately storing the address information and data information of a packet so that an input port only needs to manage a linear number (N) of queues. In conjunction with the multicast VOQ switch, we present a first-in-first-out based multicast scheduling algorithm, FIFO Multicast Scheduling (FIFOMS), and conduct extensive simulations to compare FIFOMS with other popular scheduling algorithms. Our results fully demonstrate the superiority of FIFOMS in both multicast latency and queue space requirement.

Index Terms—Multicast, scheduling, virtual output queued (VOQ) switch, head of line (HOL) blocking, crossbar switch, multicast switch.

1 Introduction and Background

Multicast is an operation to transmit information from a single source to multiple destinations and is a requirement in high performance networks [1]. Many networking/computing applications require high speed switching for multicast traffic at the switch/router level to save network bandwidth. Scheduling multicast traffic on packet switches has received extensive attention in recent years, see, for example, [5], [6], [7], [8], [9], [11], [21]. Although there have been many scheduling algorithms proposed for different types of packet switches, how to efficiently organize and schedule multicast packets remains a challenging issue.

In general, packet switches can be divided into two broad categories: output queued (OQ) switches and input queued (IQ) switches, based on where the blocked packets are queued at the switch. Readers may refer to [4] for a good taxonomy of queuing-based switch architectures. A typical OQ switch, as shown in Fig. 1a, has a first-in-first-out (FIFO) queue at each output port to buffer the packets destined to that output port. OQ switches are shown to be able to achieve unity throughput and can easily meet different QoS requirements, such as delay and bandwidth, by applying various scheduling algorithms. However, since there is no buffer at the input side, if the packets arriving at different input ports are destined to the same output port, all the packets must be transmitted simultaneously. Therefore, in order for OQ switches to work at full throughput, the switching speed of the internal fabric and the receiving speed of the output port must be N times faster than the sending speed of the input port in an $N \times N$ switch. This speedup requirement makes OQ switches difficult to scale. In particular, when the switch has a large number of input ports or the speed of a single input port increases to gigabit/s, it is impractical to achieve the N speedup [10], [25].

On the other hand, for IQ switches, the switching fabric and the output port only need to run at the same speed as that of the input port and, therefore, IQ switches have been the main research focus of high speed switches. The single input queued switch, as shown in Fig. 1b, has a FIFO queue at each input port to store the incoming packets waiting for transmission. Since only the packet at the head of line (HOL) of each input queue can participate in the scheduling, the packets behind the HOL packet suffer from the so-called "head of line" blocking, which means that, even though their destination output ports may be free, they cannot be scheduled to transfer because the HOL packet is blocked. Furthermore, it was proven in [27] that, when N is large, a single input queued switch running under the unicast i.i.d. Bernoulli traffic can reach a maximum throughput of approximately 58.6 percent and, under bursty traffic, the throughput can be even lower [18].

Ahuja et al. [11] proposed a multicast scheduling algorithm called TATRA, based on the single input queued switch structure, by mapping the general multicast switching problem onto a variant of the popular block packing game, Tetris. The basic idea of TATRA is to schedule HOL packets in such a way that it leaves the residue, i.e., the set of packets that lose contention for output ports and remain at the HOL of the input queues, on the smallest number of input ports. Therefore, more new packets in input queues can participate in the scheduling process in the next time slot. However, the performance of TATRA is

- *D. Pan is with the Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794. E-mail: pandeng@cs.sunysb.edu.*
- *Y. Yang is with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794. E-mail: yang@ece.sunysb.edu.*

Manuscript received 25 May 2004; revised 2 Nov. 2004; accepted 19 Nov. 2004; published online 16 Aug. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0176-0504.

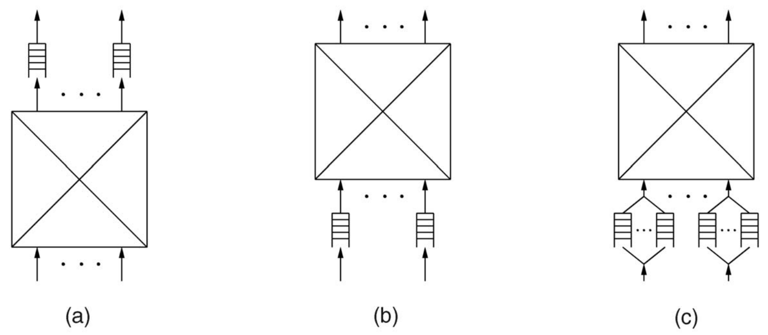

Fig. 1. Packet switches can be divided into two categories based on where the unserved packets are buffered. (a) Output queued switch. (b) Single input queued switch. (c) Multiple input queued switch.

restricted by the HOL blocking with the single input queued structure, especially when the incoming traffic has mixed multicast and unicast packets or the multicast packets have a relatively small average number of destinations (or fanout).

An efficient yet simple buffering strategy to remove the HOL blocking is to adopt the multiple input queued switch structure, which was introduced in [26]. A typical multiple input queued switch has a separate FIFO queue corresponding to each output port at each input port, resulting in a total of N^2 input queues, as shown in Fig. 1c. It is also called the virtual output queued (VOQ) structure since each queue stores those packets which have arrived at a given input port and are destined to the same output port. HOL blocking is eliminated because a packet cannot be held up by a packet ahead of it that goes to a different output. It is known that the VOQ switch structure can achieve 100 percent throughput for all independent arrival processes by using the maximum weight matching algorithm [3] or by using other maximum matching algorithms with speedup [14], [15], [16], [17]. The traditional VOQ structure buffers packets to different destinations in different queues. However, since a multicast packet may be destined to multiple output ports, it has $2^N - 1$ possible destinations. This means that a VOQ switch for multicast traffic needs to maintain $2^N - 1$ separate queues at each of its input ports, which is obviously infeasible, especially for a large N.

Based on the VOQ switch structure, a lot of scheduling algorithms have been proposed, such as iSLIP [2], PIM [25], 2DRR [19], and SERENA [12], [13], but most of them are mainly designed for unicast traffic because, as stated above, the traditional VOQ switch cannot handle multicast traffic efficiently.

ESLIP [22] adopts the VOQ structure to buffer unicast packets and puts all the multicast packets in a special single queue at each input port. It uses a variant of the iSLIP [2] algorithm to schedule mixed unicast and multicast traffic. As can be expected, ESLIP eliminates the HOL blocking for unicast traffic, but not for multicast traffic. In an extreme situation, where all the incoming packets are multicast packets, ESLIP cannot benefit from the VOQ structure and it is actually working on the single input queued switch in this case.

In order to eliminate the HOL blocking and, at the same time, to make the VOQ structure practical for multicast traffic, in this paper, we present a novel scheme to organize the packets in the input buffers of a VOQ switch by separately storing the address information and the data information of a packet. In conjunction with the new structure of the multicast VOQ switch, we present a first-in-first-out-based multicast scheduling algorithm, called FIFO Multicast Scheduling (FIFOMS). As will be seen, FIFOMS fully uses the multicast capability of a crossbar fabric, does not suffer from the HOL blocking, and performs well under both multicast traffic and unicast traffic. It can provide fairness guarantee and can achieve 100 percent throughput under uniformly distributed traffic. Our simulation results show that FIFOMS significantly outperforms other scheduling algorithms for input queued switches on average packet delay and buffer space requirement.

As discussed in [22], fixed length packet scheduling has significant advantages over variable length packet scheduling and most of the implemented high speed switches internally operate on fixed length packets as well, such as Cisco 12000 GSR [22], MGR [23], Tiny Tera [24], and AN2 [25]. Therefore, we make the same assumption in this paper. For variable length packets, they can be segmented into fixed size units to schedule and transfer across the switch and the segmentations can be reassembled back into variable length packets at the output ports before being transmitted on the outline. In the following, we assume a switch model of N input ports and N output ports with a multicast capable crossbar as its switching fabric. The switch runs in a synchronous time slot mode and the incoming traffic includes fixed length unicast and multicast packets.

The rest of the paper is organized as follows: Section 2 presents the new scheme to organize multicast packets in the input buffer of a VOQ switch. Section 3 describes the corresponding multicast scheduling algorithm FIFOMS. Section 4 discusses some implementation issues and the complexity of the algorithm and Section 5 presents the simulation results. Finally, Section 6 concludes the paper.

2 Queue Structure for Multicast VOQ Switches

As mentioned above, under the existing queuing scheme of a VOQ switch, each input port needs to maintain $2^N - 1$ separate queues to handle multicast packets, which makes the VOQ structure impractical for multicast scheduling. In the following, we describe a new scheme for organizing packets in the input buffers of a multicast VOQ switch so that the number of queues at each input port can be reduced to N.

In general, the main task of a switch includes two separate functions:

- Scheduling: Deciding for each input port which output port the packet should be sent to and

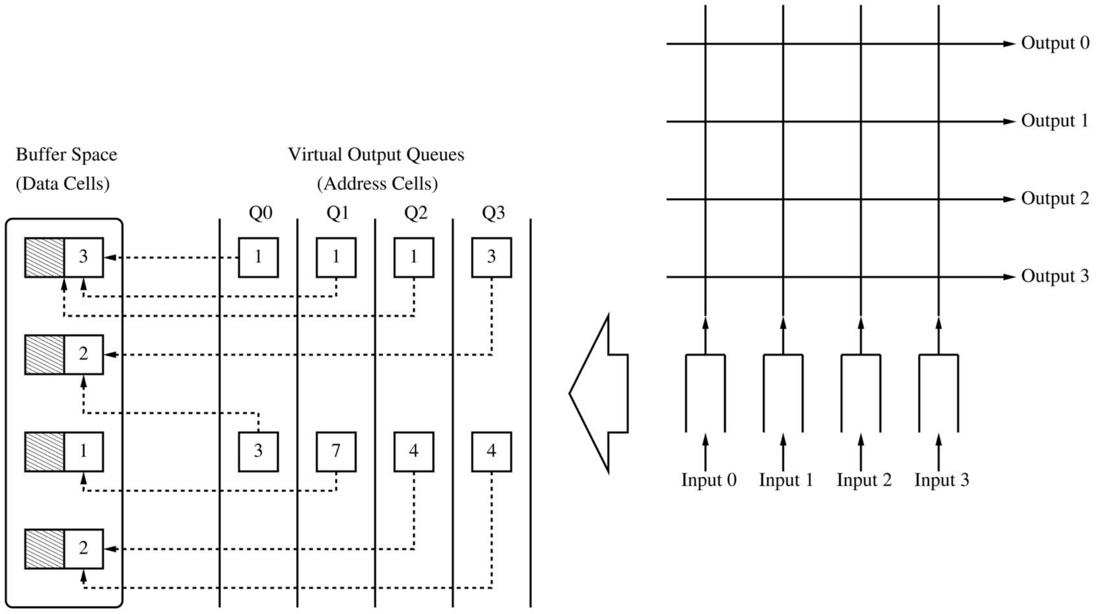

Fig. 2. An example of a 4×4 multicast VOQ switch. The left part shows the details of input port 0.

arbitrating when more than one input port requests for the same output port.

- Data forwarding: Sending the packet data from input ports to output ports according to the scheduling decision.

Accordingly, the information that a packet carries can be divided into two parts. The first part is the data content to be transferred. The second part is the destination address information of the packet, which is also used by the switch to make scheduling decisions. When the switch handles only unicast traffic, where the data content of a packet needs to be sent only once from an input port to a single output port, it is natural to combine the two functions into a single unit and use it for both scheduling and transmission. However, when multicast traffic is involved, a packet may need to be sent to multiple output ports. Although the destinations are different, the data content to be sent is the same. Therefore, there is no need to store multiple copies of the same data content. A more efficient way would be to store the address and data content of a packet separately: The data are stored once and used for all destination addresses of the packet.

We use two different types of cells to store the two parts of a packet: the data cell to store the data content of the packet and the address cell to store the destination information of the packet.

A new data cell is created to store the payload when a new packet arrives at the switch. Its data structure can be described as follows:

```
DataCell {
    binary dataContent;
    int fanoutCounter;
}
```

The dataContent field stores the data content of a packet. Since we assume the incoming traffic includes only fixed length packets, it can be implemented as a fixed size field. The fanoutCounter field records the number of destination output ports that the dataContent is going to be sent to. When a packet arrives at the switch, the fanoutCounter field of its data cell is equal to the fanout of the packet. As the dataContent is sent to part or all of the destinations of the packet, the number in the fanoutCounter field is decremented accordingly. When it becomes 0, it means that all the destination output ports have been served and, therefore, the data cell can be destroyed so as to return the buffer space to the switch.

The address cell stores the destination address information of a packet. Specifically, an address cell represents one of the destination output ports of the packet and serves as a place holder in the virtual output queue corresponding to that output port. When a new packet with fanout k enters the switch, k address cells are created for these destination output ports. The data structure of an address cell can be described as follows:

```
AddressCell {
    int timeStamp;
    pointer pDataCell;
}
```

The timeStamp field records the arrival time of the packet that the address cell is related to. It also has extra precision digits to differentiate the multiple packets of a single input port arriving in the same time slot. In such a case, an arbitrary order is given to these packets by assigning different values to their extra precision digits. The timeStamp field will be used by the scheduling algorithm FIFOMS for two purposes: On the one hand, because all the address cells of the same packet have the same arrival time, the timeStamp field can be used to identify the address cells that belong to the same multicast packet. On the other hand, the time stamp value can be used as a scheduling criterion of the first-in-first-out principle, where the address cells of earlier arrived packets have smaller values.

The pDataCell field is a pointer to the data cell that the address cell corresponds to. Each address cell points to exactly one data cell and each data cell may be pointed to by one or more address cells due to multicast traffic. When an

TABLE 1
Packet Preprocessing Algorithm

```
Input: A new packet with destination vector dest[$N$], in which dest[$i$] = true means
        output port $i$ is one of its destinations.

Output: One data cell in the buffer, and $k$ address cells in the virtual output queues,
        where $k$ is the fanout of the multicast packet.

dc = new DataCell(); // generate a new data cell
dc.dataContent = newPacket.payload; // set the dataContent field
dc.fanoutCounter = newPacket.fanout; // set the fanoutCounter field

for (i = 0; i < N; i++) {
    // generate the address cell for output port i, and enqueue it
    if (newPacket.dest[i] == true) {
        ac = new AddressCell();
        ac.timeStamp = currentSlot;
        ac.pDataCell = dc;
        queue[i].enqueue(ac);
    }
}
```

address cell is scheduled to transfer, the input port will actually send to the corresponding output port the dataContent of the data cell that the address cell's pDataCell field points to.

After explaining the two types of cells used, we now present the entire picture of the queue structure in a multicast VOQ switch. In each input port, there is a buffer used to store the data cells and there are N virtual output queues to store the address cells for the N output ports. All the address cells in the same virtual output queue are destined for the same output port and only the address cells at the head of the queues can be scheduled. If an address cell receives the grant from a particular output port in the scheduling, the crosspoint connecting the corresponding input port and output port will be set and the data cell that the address cell's pDataCell field points to will be transferred. After the data are sent, this address cell is removed from the head of its queue and the fanoutCounter field of the corresponding data cell is accordingly decreased by one.

Fig. 2 gives an example of a 4×4 multicast VOQ switch. The input ports and output ports are connected by a crossbar fabric and the incoming packets are buffered at the input side. The details of input port 0 are shown in the left part of the figure, in which there is a buffer for data cells and four virtual output queues for address cells. The shaded area of the data cell represents the dataContent field and the number is the current value of the fanoutCounter. The number in the address cell stands for the timeStamp field and the arrow points to its related data cell. Input port 0 has four packets that have not been fully transferred, and the packets entered the switch at the first, third, fourth, and seventh time slots, respectively. The fanout of the first slot packet is 3 and the packet still needs to be sent to output ports 0, 1, and 2, the destinations of the third slot packet are output ports 0 and 3, the destinations of the fourth slot packet are output ports 2 and 3, and the seventh slot packet is a unicast packet to output port 1.

3 First-In-First-Out Multicast Scheduling Algorithm (FIFOMS)

By using the modified queue structure, the VOQ switch can now efficiently handle multicast packets. However, no appropriate algorithms are available for scheduling multicast traffic on the VOQ switch. On the one hand, existing multicast scheduling algorithms for input queued switches, such as TATRA [11], are based on the single input queued switch structure and, therefore, suffer from the HOL blocking. On the other hand, current scheduling algorithms for VOQ switches, see, for example, iSLIP [2], PIM [25], 2DRR [19], and SERENA [13], were mainly designed for unicast traffic because the traditional VOQ switch queue structure is not suitable for multicast traffic. The scheduling principle of these scheduling algorithms is that an input port can only send its packet to one output port in a single time slot.

In this section, we propose a new multicast scheduling algorithm, called FIFO Multicast Scheduling (FIFOMS), for working with the multicast VOQ switch. As will be seen, the multicast VOQ switch structure completely removes the HOL blocking and enables FIFOMS to achieve 100 percent throughput under uniformly distributed traffic. And, at the same time, FIFOMS utilizes the multicast capability of a crossbar switch to send a multicast packet to all its destination output ports in the same time slot whenever possible, which significantly reduces the multicast latency.

It should be mentioned that, for any multicast scheduling algorithm, there is an inherent conflict in scheduling. In order to make use of the multicast characteristics and achieve short average packet delay, it is preferred that a multicast packet be sent to all its destination output ports in the same time slot or, in other words, all the output ports should choose the same multicast packet in the scheduling arbitration. However, for the sake of fast scheduling, each output port should make arbitration concurrently. Then, the question is: How could the independently made decisions choose the same packet? FIFOMS solves this problem by adopting the first-in-first-out rule. It assigns every incoming packet a timestamp with the value equal to its arrival time and uses the timestamp as a criterion in the scheduling arbitration. The timestamp criterion makes the multicast packets which arrived earlier have a better chance of being chosen by all their destination output ports when the output ports make scheduling decisions independently. Next, we

TABLE 2
First-In-First-Out Multicast Scheduling Algorithm

```
Input: Input ports with address cell queues and data cell buffers.

Output: Scheduling decision.

// initialization
initially, all input and output ports are free;

do {
    // request step
    for all input ports do {
        if the input port is free {
            smallestTimeStamp = the smallest timeStamp of all HOL address cells
                whose corresponding output ports are free;

            for all HOL address cells {
                if the address cell's corresponding output port is free AND
                    its timeStamp is equal to smallestTimeStamp {
                    the address cell sends a request to the corresponding output port,
                        with its timeStamp as the weight;
                }
            }
        }
    }

    // grant step
    for all output ports do {
        select the smallest time stamp from all its requests;
        if there are more than one such requests, randomly select one;
        grant the address cell corresponding to the selected request;
        mark the output port and the input port of the granted address cell as reserved;
    }
} while some input port and output port pairs match in this round;

// data transmission
set the crosspoints of the switch fabric;
for all input ports do {
    find the data cell through the pDataCell pointer field of the scheduled address cell;
    send the data cell to all the scheduled output ports;
}

// post-transmission processing
for all input ports do {
    for each scheduled address cell {
        decrease the fanoutCounter field of the related data cell by 1;
        if the data cell's fanoutCounter field becomes 0 {
            destroy the data cell;
        }
        remove the address cell from the head of queue;
}
```

will describe FIFOMS and its associated packet preprocessing algorithm.

3.1 Preprocessing Incoming Packets

In order to fit into the multicast VOQ switch queue structure, a multicast packet needs to be preprocessed upon arriving. One data cell is generated in the data buffer to store the content of the packet. A separate address cell is generated for each of the destination output ports, with its timeStamp field assigned the value of the current time slot, and is put at the end of the corresponding queue.

Details of the packet preprocessing algorithm are described in Table 1.

3.2 First-In-First-Out Multicast Scheduling Algorithm (FIFOMS)

Similarly to iSLIP [2] or PIM [25], FIFOMS is an iterative algorithm, and each iterative round consists of two steps:

- Request—Address cells at each input port make requests to their destination output ports for possible transmission.
- Grant—Each output port selects one request from all the requests it received and grants the transmission to the corresponding address cell.

However, differently from iSLIP and PIM, the accept step is not needed in FIFOMS because, in our request step, all the address cells that make requests must point to the same data cell. Therefore, only one of the data cells in an input port can be granted the transmission and there is no potential conflict in which an input port needs to send more than one data cell in a single time slot. Thus, in a scheduling round, FIFOMS has one fewer operational step and less data exchange between input ports and output ports.

Initially, all the input ports and output ports are free. After an iterative round, some pairs of input ports and output ports are matched and marked as "reserved" so that they are no longer considered in future rounds of the

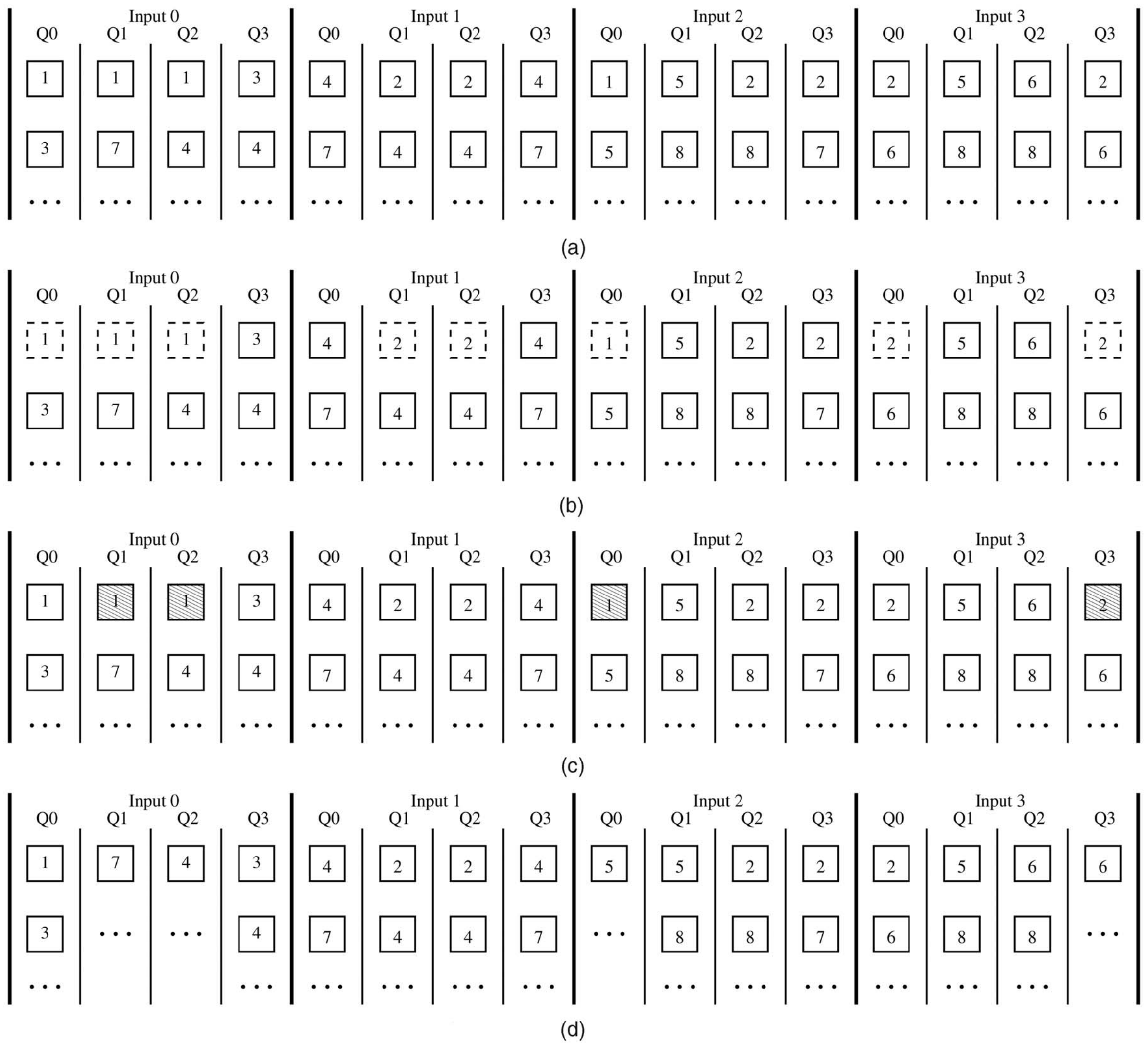

Fig. 3. A simple FIFOMS scheduling example for a 4×4 switch. (a) Before scheduling. (b) Request step. (c) Grant step. (d) After transmission.

current time slot. The FIFOMS scheduling algorithm is described in Table 2 and we will explain each step in more detail next.

3.2.1 Request Step

In the request step, an input port finds the earliest HOL address cells of free input ports and gives them priorities to send transmission requests. There are two possible cases:

1. If the input port is free in the current iterative round, it simply selects the HOL address cells whose time stamps are the smallest and whose corresponding output ports are free. Then, the selected address cells send requests to their output ports with the scheduling weight being its time stamp. Note that, in an input port, there may be more than one such address cell with the same smallest time stamp which came from the same multicast packet.
2. Otherwise, if the input port is reserved and some address cells have been scheduled to transfer in the earlier rounds of the current time slot, it means that all the other HOL address cells with the same time stamp, if there are any, must have made requests in the earlier rounds but were not selected by the output ports. Since one input port can send at most one data cell in a single time slot, the input port can no longer make requests.

3.2.2 Grant Step

After the request step, each free output port has collected some requests with different weights. Following the first-in-first-out rule, an output port grants the request with the smallest time stamp. It is possible that several requests have the same smallest time stamp. In this case, the output randomly selects one to grant.

The iterative rounds of the request and grant steps continue until there are no possible matchable pairs of free input ports and free output ports.

3.2.3 Data Transmission

After the scheduling decisions are generated during the iterative matching rounds in the form of matched input and output pairs, the corresponding crosspoints connecting the scheduled input ports and output ports are set and the input port begins to send the data cell. Note that an input port may be connected to more than one output port simultaneously. Thus, the algorithm can fully use the built-in multicast capability of the crossbar switching fabric.

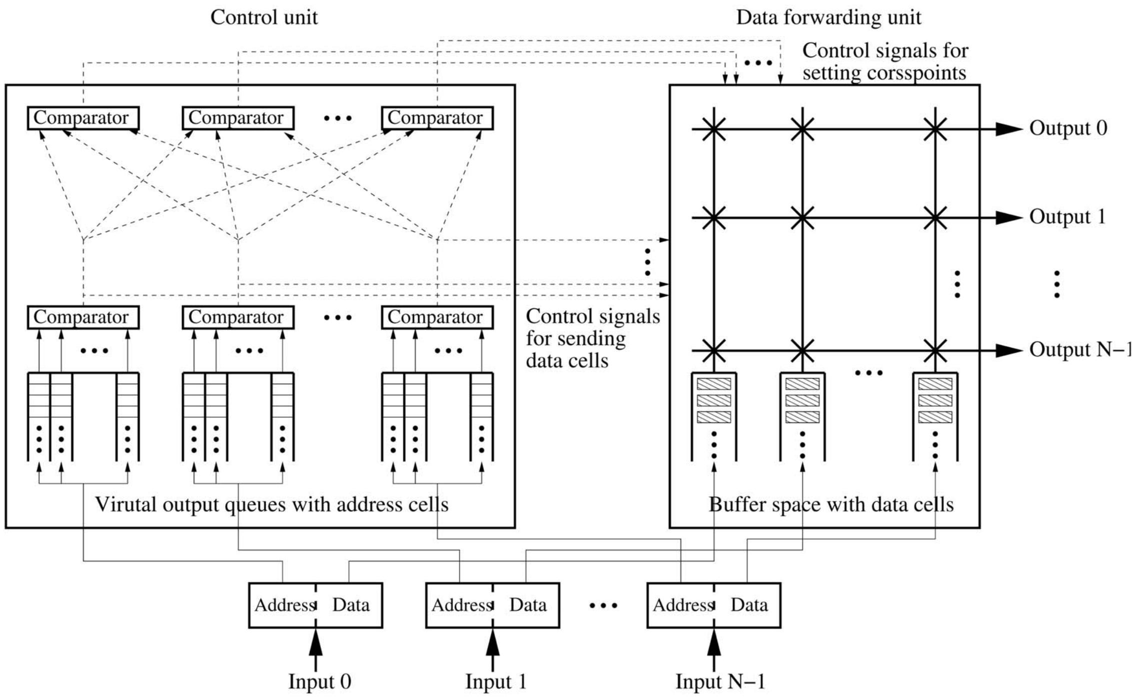
Control unit
Data forwarding unit
Control signals for setting corsspoints
Comparator
Control signals for sending data cells
Output 0
Output 1
Output N−1
Virutal output queues with address cells
Buffer space with data cells
Address Data
Input 0
Input 1
Input N−1

Fig. 4. The overall FIFOMS scheduler can be logically divided into two units, the control unit on the left and the data forwarding unit on the right.

3.2.4 Posttransmission Processing

After the transmission is completed, some postprocessing work needs to be performed to update the address cells and data cells that have been transferred. The served HOL address cells are removed from the heads of their queues and the fanoutCounter fields of the related data cells are decreased accordingly. If a data cell's fanoutCounter field becomes 0, i.e., it has been sent to all destination output ports, the data cell is destroyed to return the buffer space.

3.3 A Scheduling Example

Fig. 3 gives a simple example of FIFOMS for a 4×4 multicast VOQ switch. For clarity, data cells are not shown in the figure. Each input port includes four queues of address cells. The number on each address cell indicates its time stamp value. Fig. 3a shows the status of the four input ports after the incoming packets of the current time slot have arrived. Fig. 3b explains the request step. The address cells in dashed lines are the earliest HOL address cells in each input port and each of them makes a request to the corresponding output port. As a result, output port 0 receives a total of three requests from input ports 0, 1, and 3, output port 1 receives two requests from input ports 0 and 1, output port 2 receives two requests from input ports 0 and 1, and output port 3 receives one request from input port 3. Fig. 3c shows the result of the grant step in which each output port independently selects one of its requests to grant and makes a random arbitration if there is any contention. The address cells in gray are the address cells that receive grants. Since, in this case, all four of the output ports have made their grant decisions in a single round of FIFOMS, the algorithm converges and the scheduling for the current time slot is completed. Otherwise, more rounds of matching may be needed. At the end of the time slot, the data cells that the scheduled address cells point to are sent to the corresponding output ports and the address cells are removed from the HOL of their queues, as shown in Fig. 3d.

4 Hardware Implementation and Complexity Analysis

In this section, we discuss some implementation and performance issues of the newly proposed scheduling algorithm and analyze the complexity of the algorithm.

4.1 Hardware Implementation

One important property of a practical scheduling algorithm is that it should be easy to implement. In the following, we briefly discuss the hardware implementation of the FIFOMS scheduler. As can be seen, FIFOMS can be fairly easy to implement in hardware and, thus, achieves high speed switching in practice.

The scheduler can be logically divided into two units, as shown in Fig. 4, corresponding to the scheduling functionality and data forwarding functionality, respectively.

In the control unit on the left, the input side consists of all the address cell queues because the information provided by the address cells is used for making scheduling decisions. A comparator is used at each input port to select the HOL address cells with the smallest time stamp. Since the comparison operations of each input port do not depend on each other, it can be performed in parallel. The selected address cells send their requests with timestamps as weights to the corresponding output ports. Then, each output port uses a comparator to select the request with the smallest timestamp and grants the transmission to the corresponding address cell. Finally, before the next iterative round of FIFOMS can start, the grant results of the current round are fed back to the input ports.

The data forwarding unit consists of the data cell buffer space and the crossbar switching fabric. The scheduling decisions made by the control unit are forwarded to the data forwarding unit as control signals. The output of the comparator of each input port is used to select, from the buffer space, which data cell should be sent. And, the output of the comparator of each output port controls which

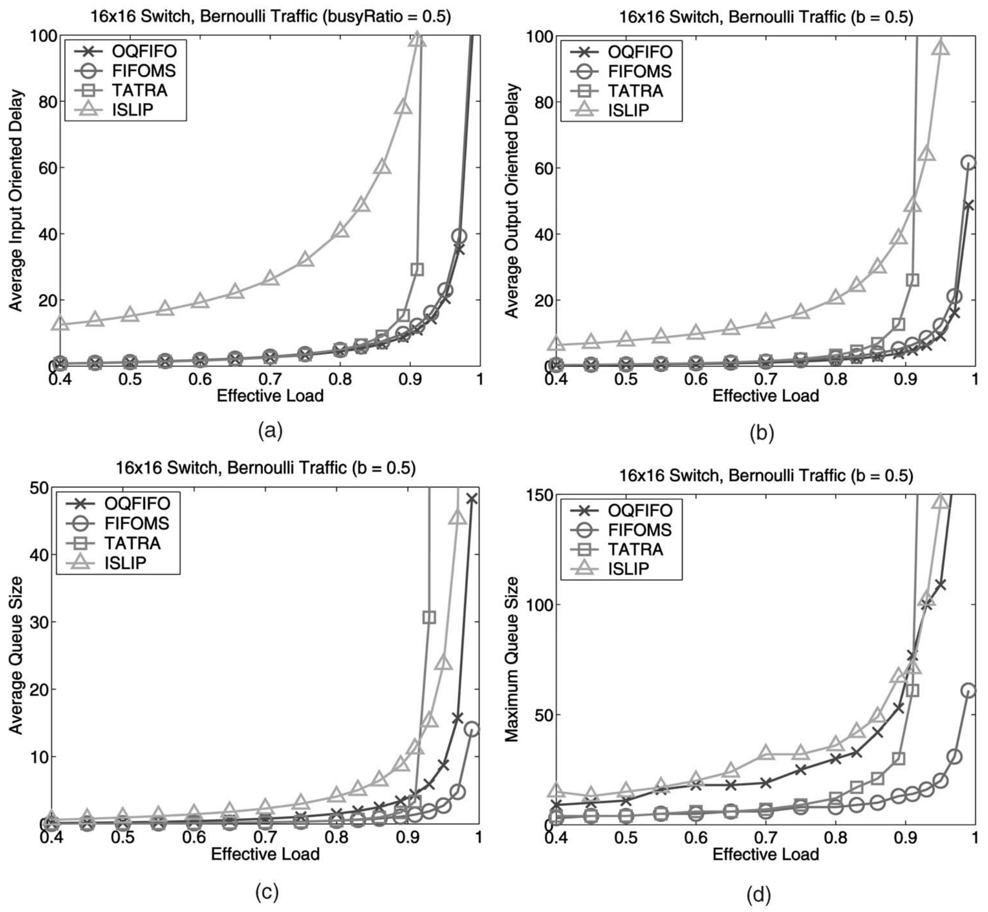

Fig. 5. Simulation results for a 16×16 switch under Bernoulli traffic with $b = 0.5$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

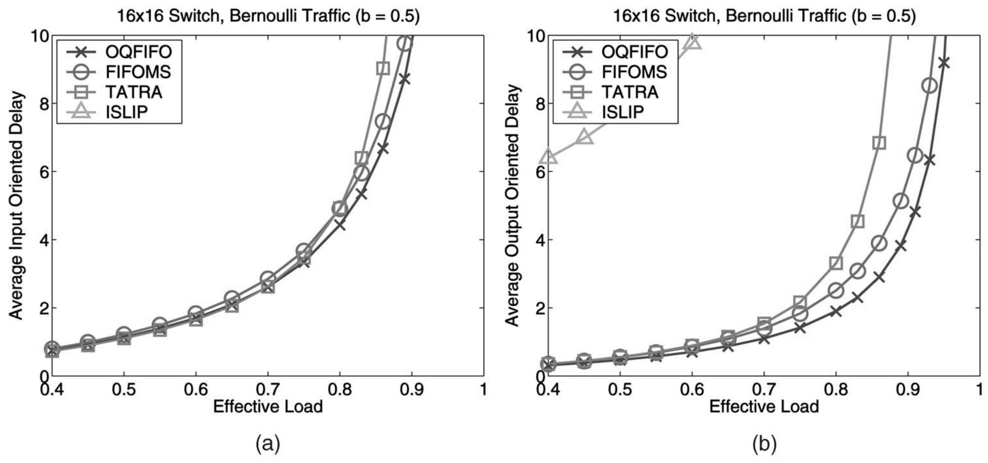

Fig. 6. Under light traffic load, the average input oriented packet delay of TATRA is slightly shorter than that of FIFOMS, but FIFOMS always outperforms TATRA on the average output oriented packet delay. (a) Average input oriented delay. (b) Average output oriented delay.

crosspoint should be set to connect a particular input port with this output port.

4.2 Space Complexity of the Algorithm

As has been seen, by separately storing the data and address information of a packet, a VOQ switch is able to handle multicast traffic efficiently. The multicast VOQ switch saves buffer space by storing only one copy of the data content of a multicast packet. Compared to the single input queued switch, the multicast VOQ switch consumes slightly more storage space. The main cost comes from the separately stored multiple address cells of a packet, in which case, a single packet may need up to N times the size of an address cell. Fortunately, the data structure of an address cell only includes an integer field and a pointer field and a small constant number of bytes should be sufficient.

4.3 Time Complexity of the Algorithm

The time complexity for preprocessing an arriving packet is $O(N)$ because, when a multicast packet arrives at the switch, up to N address cells may need to be created. Stiliadis [28] pointed out the potential memory speedup problem, but, since the destinations of a packet are independent and an address cell is comprised of only a few bytes, the operation can be done in parallel by hardware to achieve $O(1)$ complexity. Furthermore, the preprocessing of new packets can be overlapped with the scheduling and the switching in the switch. Thus, it would not introduce extra time delay.

The most time consuming operation in each iterative round of FIFOMS is for an input port to find the smallest time stamp from those of all the HOL address cells and for an output port to select the request with the smallest timestamp. If the operation is executed in a serial fashion, the time complexity is $O(N)$. If we use the parallel comparators as in the WBA scheduler [20], the time complexity can be reduced to $O(1)$.

The convergence time has been a major concern for iterative matching algorithms like FIFOMS. In the worst case, FIFOMS runs N rounds to converge because, in each round, at least one output port is scheduled for receiving a data cell from an input port and will not be considered any more in the future rounds of the current time slot. But, as will be seen later in the simulation results section in Fig. 7, for the average case, the convergence rounds of FIFOMS is much smaller than N. And, we have an interesting observation that FIFOMS and iSLIP require almost the same number of rounds to converge under a relatively light traffic load.

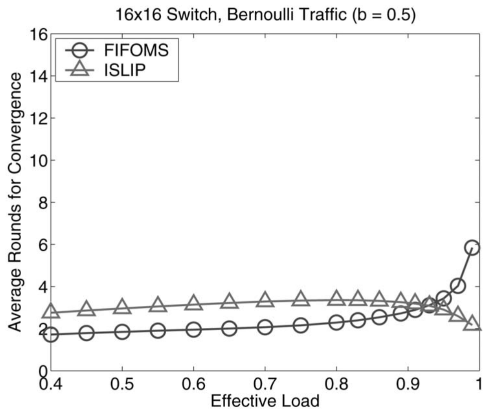

Fig. 7. Average convergence rounds of FIFOMS and iSLIP for a 16×16 switch under Bernoulli Traffic with $b = 0.5$.

5 Simulation Results

We have conducted extensive simulations to compare the performance of FIFOMS with three other scheduling

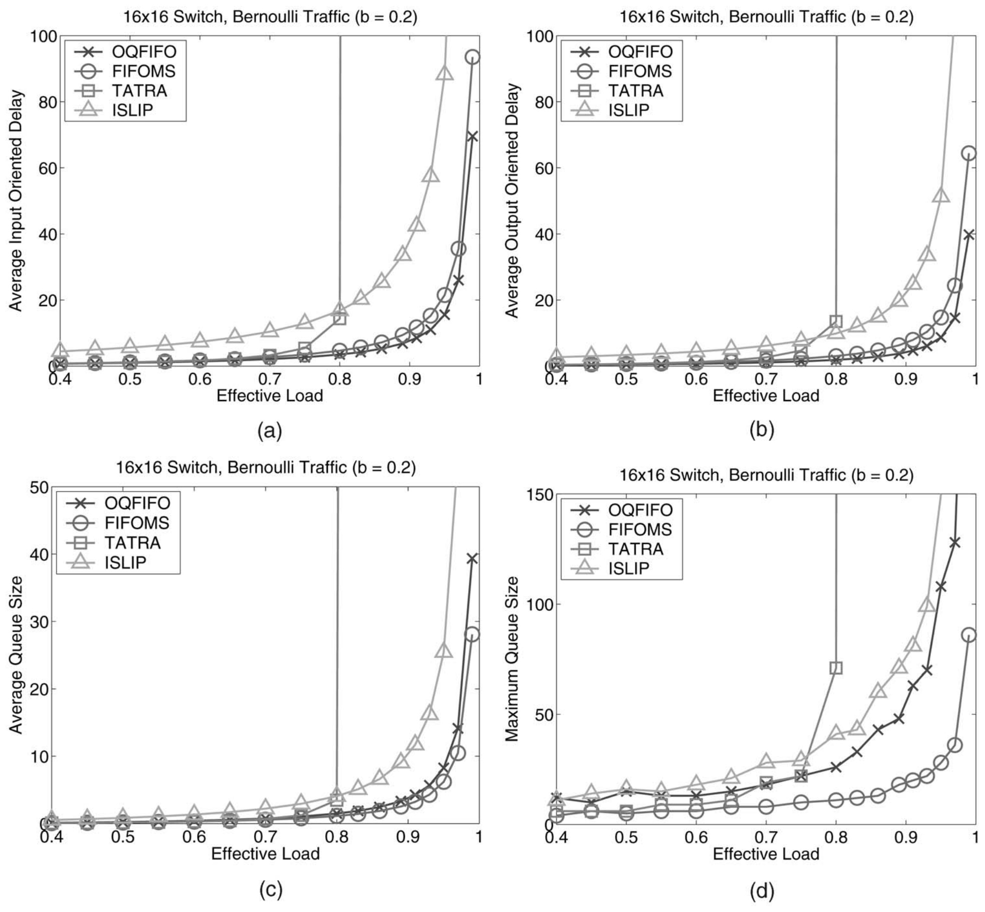

Fig. 8. Simulation results for a 16×16 switch under Bernoulli traffic with $b = 0.2$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

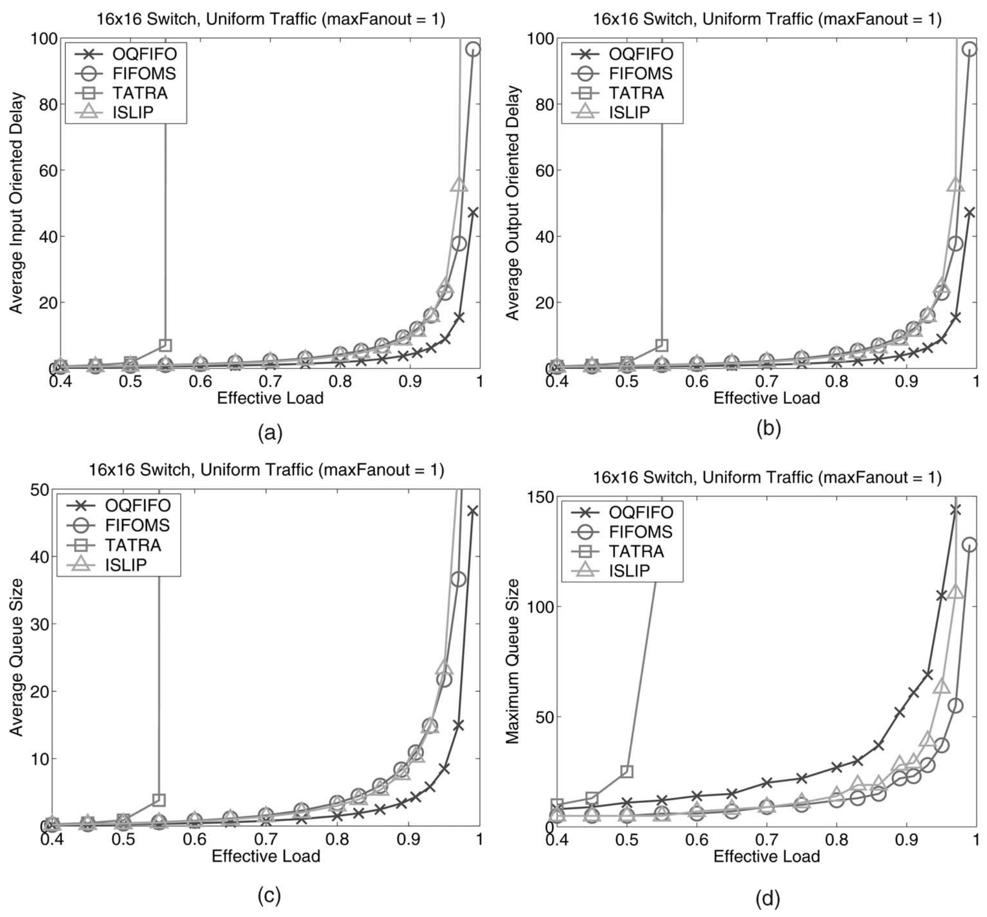

Fig. 9. Simulation results for a 16×16 switch under uniform traffic with $\mathrm{maxFanout} = 1$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

algorithms: TATRA [11], iSLIP [2], and a simple FIFO scheduling algorithm on the output queued switch structure:

- TATRA is a multicast scheduling algorithm based on the single input queued switch structure. By minimizing the number of input ports with the set of cells that lose contention for output ports and remain at the HOL of the input queues in each cycle, it achieves short multicast latency and is free of starvation as well. Through the comparison with TATRA, we demonstrate that FIFOMS successfully removes the HOL blocking, which restricts the maximum throughput TATRA can reach.
- iSLIP is a scheduling algorithm mainly designed for unicast traffic based on the VOQ switch structure. iSLIP was originated from PIM [25] and achieves better performance by desynchronizing round-robin schedules. Under heavy load, the round-robin pointers of different input/output ports automatically point to different output/input ports so that there is no conflict and fast scheduling decisions can be made. In the simulations, iSLIP schedules a multicast packet as separate (independent) unicast packets. Through the comparison with iSLIP, we show that FIFOMS can make use of the characteristics of multicast traffic and take advantage of the multicast capability of the crossbar switch. As a result, FIFOMS has much shorter average packet delay than iSLIP for multicast traffic.
- As discussed in the introduction section, the output queued switch structure is known to be superior to the input queued structure in performance, but requires N times faster switching ability. Despite its much stronger hardware requirement, in our simulations, we also include a simple FIFO scheduling algorithm on the output queued structure (OQFIFO) as an ultimate performance benchmark for FIFOMS.

In the simulations, we collect the following four types of statistics:

- Average input oriented delay: A multicast packet may be transmitted to its different destination output ports at different time slots. Input oriented delay is the longest delay among all the destinations of a multicast packet and represents the transmission latency from the sender's point of view. Consequently, average input oriented delay is the average maximum delay for all the packets entering the switch.
- Average output oriented delay: Average output oriented delay represents the transmission latency from the receiver's point of view. It can be computed as the average of the arrival-to-departure time intervals of all the packets.
- Average queue size: Average queue size tells how long a new incoming packet needs to wait before transmission and it also reflects the space requirement of the algorithm. Since TATRA is based on the single input queued switch, its queue size is the

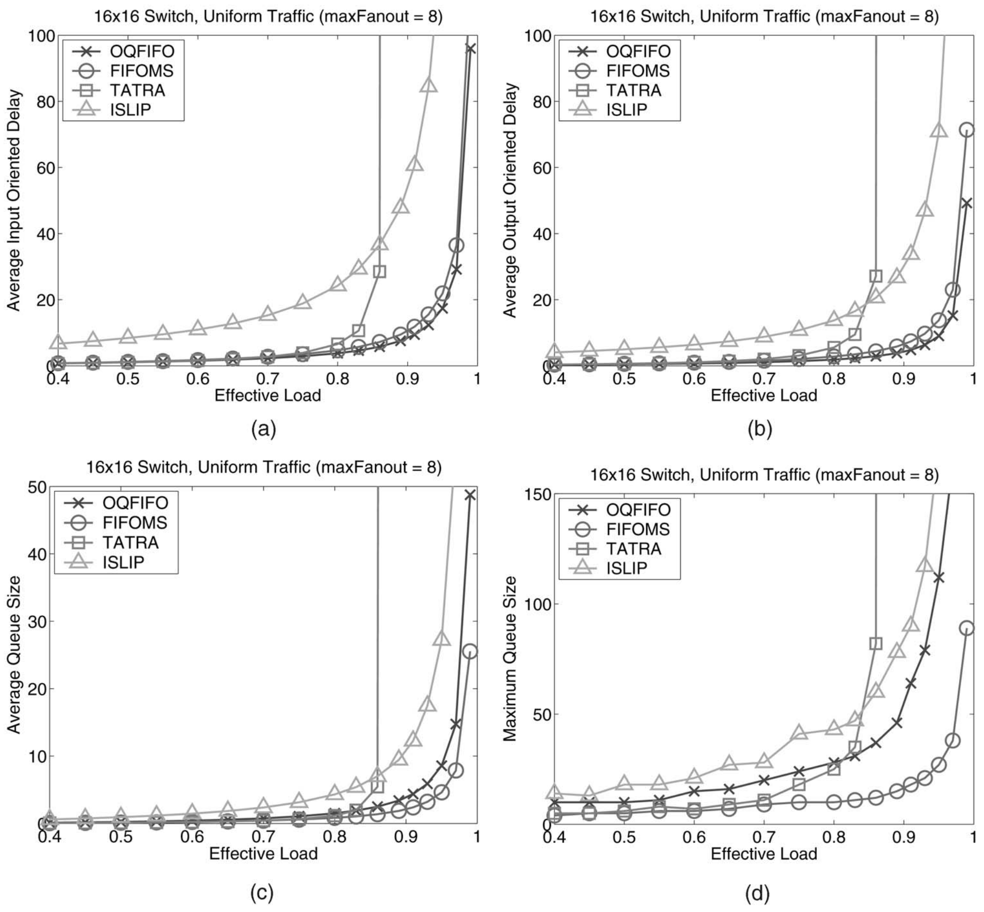

Fig. 10. Simulation results for a 16×16 switch under uniform traffic with $\mathrm{maxFanout} = 8$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

length of the single queue of each input port. For FIFOMS and iSLIP, the queue size is defined as the number of data cells in the buffer of an input port, in the sense of how many unsent packets an input port needs to hold.

- Maximum queue size: Maximum queue size gives the maximum buffer space for an algorithm to work without dropping packets at the input side and is equal to the longest length of all the queues throughout the simulation run.

All the simulated switches are assumed to operate in a discrete time slot manner with fixed size packets. In each simulation run, there is a sufficient warmup period (typically half of the total simulation time) to obtain stable statistics. The simulation runs for a fixed amount of simulation time (10^6) unless the switch becomes unstable (i.e., the switch reaches a stage where it is not able to sustain the offered load).

In order to compare the performance of the algorithms in various networking environments, we consider several different types of traffics, including Bernoulli traffic, uniform traffic, geometric traffic, and burst traffic.

5.1 Simulation Results under Bernoulli Traffic

The Bernoulli traffic is one of the most widely used traffic models in the simulation of scheduling algorithms. A Bernoulli traffic can be described using two parameters, p and b. p is the probability that an input port is busy at a time slot, i.e., the probability an input port has some packet to arrive at the beginning of a time slot. (p in the description of other traffic models has the same meaning.) And, a packet has the probability of b of being addressed to each output port. Therefore, for an $N \times N$ switch, the average fanout of a multicast packet is $b \times N$ and the effective load is $p \times b \times N$.

The simulation results for a 16×16 switch under the Bernoulli traffic with $b = 0.5$, which means that the incoming packet is uniformly distributed over all possible multicast destinations and a series of different p values, are shown in Fig. 5. As can be seen from the figure, in terms of input and output oriented average packet delays, FIFOMS closely matches OQFIFO, which has the best performance. In addition, FIFOMS outperforms all three of the other algorithms in terms of both average queue size and maximum queue size. On the other hand, due to the HOL blocking in the single input queued switch structure that TATRA is based on, when the effective load goes beyond 90 percent, the delay of TATRA increases dramatically and it becomes unstable. It can also be observed that iSLIP has a much longer average packet delay than all the other algorithms. This is because iSLIP is a scheduling algorithm specially designed for unicast traffic.

Although it is hard to notice, we would like to point out that, under a light traffic load, the average input oriented packet delay of TATRA is slightly better than that of FIFOMS, as shown in Fig. 6a. The reason is that TATRA

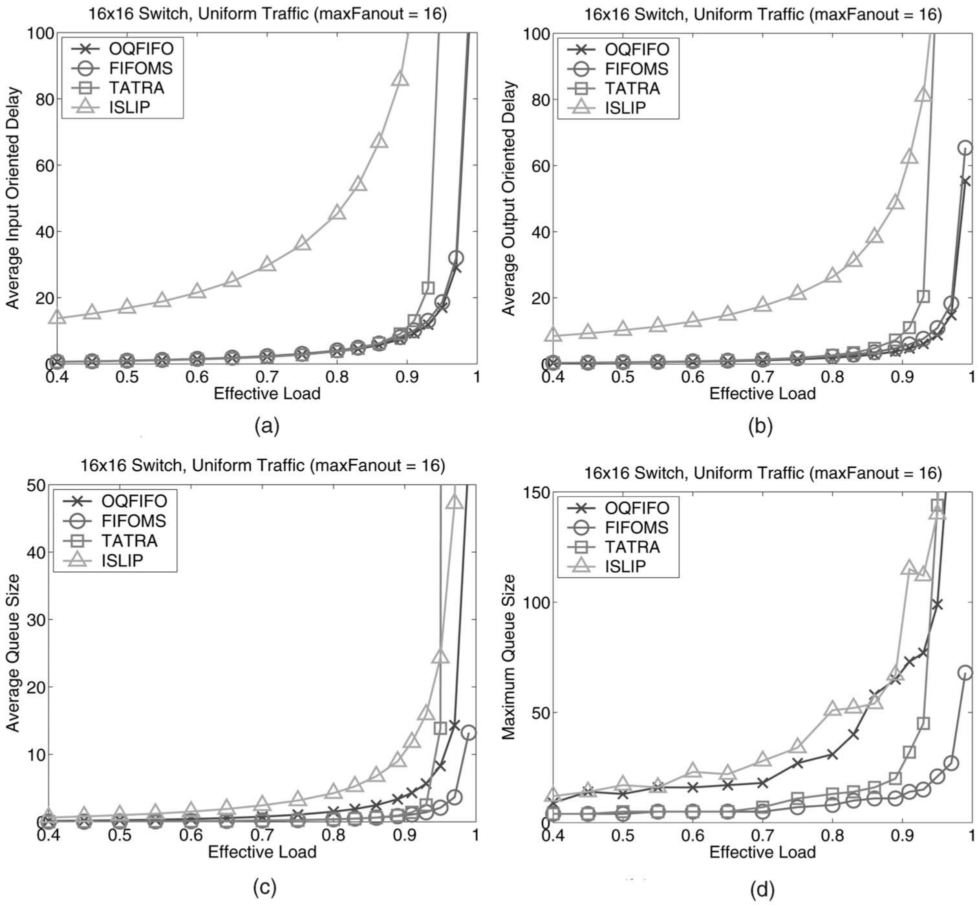

Fig. 11. Simulation results for a 16×16 switch under uniform traffic with $\mathrm{maxFanout} = 16$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

does not adopt a strict FIFO discipline and the cells in its Tetris box can move back and forth. The Tetris box of TATRA is comprised of $N \times N$ small blocks and each of its columns holds cells to a specific output port. The column of the Tetris box is different from the ordinary FIFO queue and a cell is allowed to jump ahead of other cells in the same column if necessary. For all HOL packets, a cell is inserted into the column corresponding to each destination output port of the packet. When an HOL multicast packet cannot be sent to all its destinations in a single time slot, its cells can be moved backward to make space for those packets that can fully pass the switch in the time slot. This operation does not increase the input-oriented delay of the original packet, but can accelerate the transmission of other packets. However, if we take a look at the average output oriented packet delay in Fig. 6b, FIFOMS always outperforms TATRA under any traffic loads, which justifies the multicast scheduling mechanism of FIFOMS.

Fig. 7 compares the convergence rounds between FIFOMS and iSLIP. We can see that, when the load is less than 90 percent, the convergence rounds of both FIFOMS and iSLIP are not sensitive to the increase of the traffic and are much smaller than N $(= 16)$. Also, it is interesting to notice that FIFOMS and iSLIP take roughly the same number of iterative rounds to converge. To be more specific, FIFOMS outperforms iSLIP until the effective load reaches above 90 percent, under which iSLIP has become unstable.

The simulation results for a 16×16 switch under the Bernoulli traffic with $b = 0.2$ is shown in Fig. 8. We can see that, as b decreases, the performance of TATRA is more affected by the HOL blocking, which leads TATRA to saturate at about 80 percent effective load.

5.2 Simulation Results under Uniform Traffic

In real-world applications, the fanout of most multicast connections is limited by some upper bound value instead of being uniformly distributed over all the possible destinations. In this case, we can use the uniform traffic with a restricted maximum fanout to capture these characteristics.

Uniform traffic can be described using two parameters, p and $maxFanout$, where $maxFanout$ is the maximum possible fanout of any incoming packet. The fanout of a packet is uniformly distributed from 1 to $maxFanout$ and the individual destination output ports are randomly selected from all the N output ports. Thus, for an $N \times N$ switch, the average fanout is $(1 + maxFanout)/2$ and the effective load is $p \times (1 + maxFanout)/2$.

Fig. 9 shows the simulation results when $maxFanout$ is set to 1, which makes the traffic become the pure unicast traffic. There is no doubt that the well-known unicast scheduling algorithm iSLIP achieves short average packet delay. Although mainly designed for multicast traffic, FIFOMS manages to match and even surpass iSLIP on average packet delay and is the best in terms of buffer requirement. On the contrary, the performance of TATRA is severely affected by

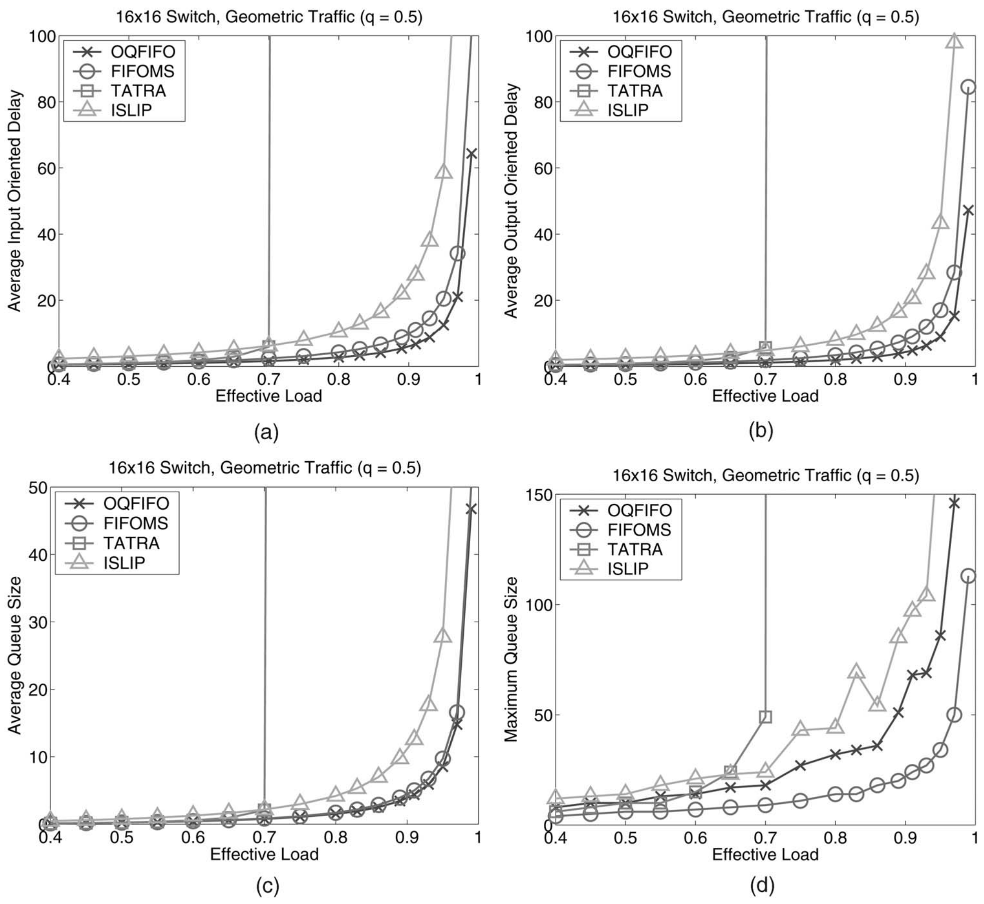

Fig. 12. Simulation results for a 16×16 switch under geometric traffic with $q = 0.5$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

the HOL blocking; it can only reach a maximum effective load of about 55 percent, which is consistent with the theoretical analysis result of 58.6 percent in [27].

Simulations are also conducted under the uniform traffic with $maxFanout$ equal to 8 and 16 and the corresponding results are shown in Fig. 10 and Fig. 11, respectively. In all cases, FIFOMS consistently gives a satisfactory performance. It has the shortest average packet delay (both input oriented and output oriented) among the three input queued scheduling algorithms and even excels OQFIFO on buffer requirement. It also can be observed that, as the $maxFanout$ value becomes larger, TATRA has better performance because it has more choices of where to move the cells in the Tetris box.

5.3 Simulation Results under Geometric Traffic

Alternatively, the geometric traffic can be used to simulate the behavior of different algorithms under small fanout dominated multicast traffic. The fanout of an incoming packet follows the following truncated geometric distribution and the individual destination output ports are randomly selected from all the N output ports. A truncated geometric traffic can be described using two parameters p and q, where q is the ratio of the probability of the packets with fanout k to the probability of the packets with fanout $k-1$ $(1 < k \le N)$.

$$\Pr\{fanout = k\} = \begin{cases} \frac{(1-q)q^{k-1}}{1-q^N}, & \text{if } 1 \le k \le N \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, for an $N \times N$ switch, the average fanout is $\sum_{k=1}^{N} k(1-q)q^{k-1}/(1-q^N) = 1/(1-q) - Nq^N/(1-q^N)$ and the effective load is $p \times \left(1/(1-q) - Nq^N/(1-q^N)\right)$.

The simulation results for a 16×16 switch under the geometric traffic with $q = 0.5$ are shown in Fig. 12. FIFOMS and OQFIFO have the shortest average packet delay and the smallest buffer requirement, whereas the HOL blocking makes the TATRA the first algorithm to become unstable at an effective load of about 70 percent. Although iSLIP has a relatively large packet delay, it is not so sensitive to the increasing load of the traffic compared to TATRA.

5.4 Simulation Results under Burst Traffic

In practice, network packets are usually highly correlated and tend to arrive in a burst mode. For a discrete time slot switch, we generally use a two state Markov process which alternates between the off and on states to describe the burst nature. In the off state, there is no packet to arrive. In the on state, packets arrive at every time slot and all have the same destinations. At the end of each slot, the traffic can switch between the off and on states independently. A burst traffic can be described using three parameters, E_{off}, E_{on}, and b. E_{off} is the average length of the off state or, alternatively, the probability of switching from the off state to the on state is $1/E_{off}$. E_{on} is the average length of the on state or the probability of switching from the on state to the off state is $1/E_{on}$. b is the probability of a packet being addressed to a specific output port. Therefore, for an $N \times N$ switch, the

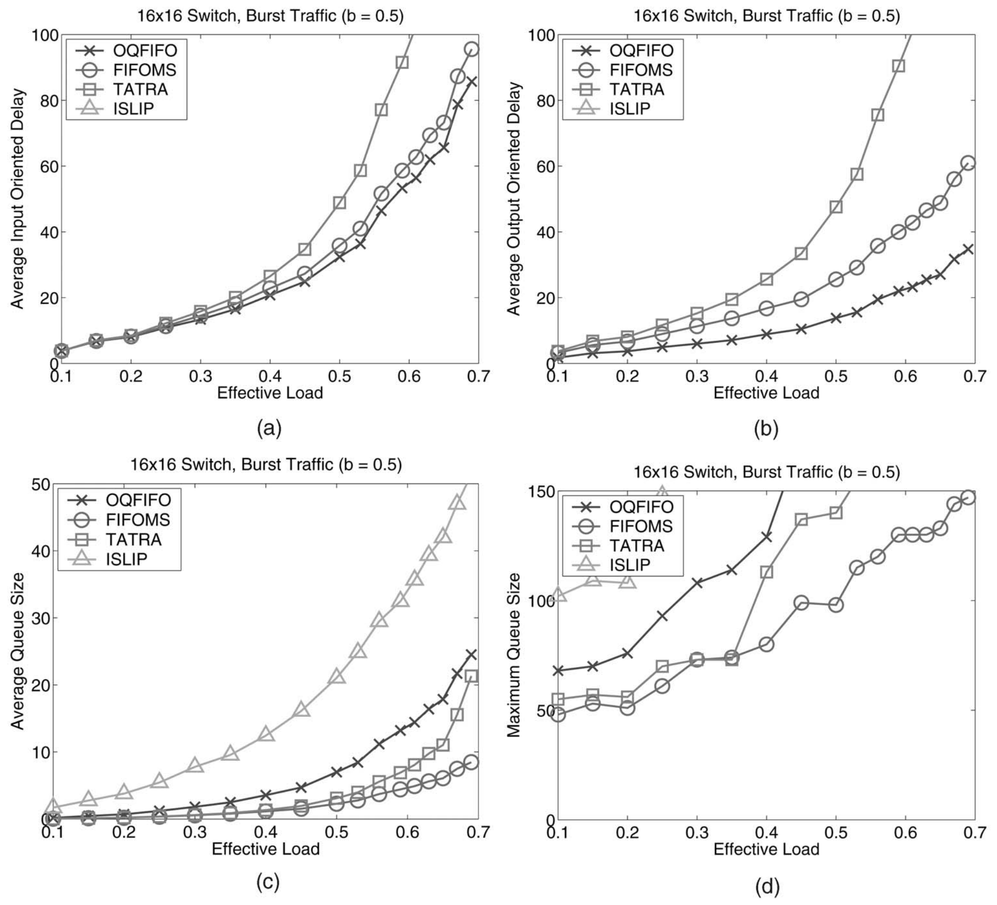

Fig. 13. Simulation results for a 16×16 switch under burst traffic with $b = 0.5$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

average fanout is $b \times N$, the arrival rate is $E_{on}/(E_{off} + E_{on})$, and the effective load is $b \times N \times E_{on}/(E_{off} + E_{on})$. For easy comparison, we set E_{on} to be the same value, 16, as in [11].

The simulation results for a 16×16 switch with $b = 0.5$ are shown in Fig. 13. Due to the burst nature, iSLIP saturates at so small a value that it cannot even be seen in the first two graphs, which is consistent with the theoretical analysis of [18]. As to average packet delay, FIFOMS outperforms TATRA, but is not as good as OQFIFO. As under other traffic modes, FIFOMS has the smallest queue space.

6 Conclusions

In this paper, we first gave a novel scheme to organize the multicast packets in input buffers of a VOQ switch. By separately storing the address and data of a packet, the new queue structure enables the VOQ switch to handle multicast traffic efficiently because it reduces the number of queues an input port needs to manages from exponential to linear and, at the same time, it keeps all existing advantages of the VOQ switch.

In conjunction with the multicast VOQ switch, we also designed a multicast scheduling algorithm, FIFO Multicast Scheduling (FIFOMS). The main features of FIFOMS can be summarized as follows:

- Performs well under both multicast and unicast traffic: FIFOMS is designed for scheduling multicast traffic and fully uses the inherent multicast capability of the crossbar switch. Furthermore, even under the pure unicast traffic, the performance of FIFOMS can also match the specifically designed unicast scheduling algorithms.
- Achieves 100 percent throughput under uniformly distributed traffic: Under uniform 100 percent offered load, all the $N \times N$ virtual output queues have sustaining backlogs. As a result, each output port can receive one data cell in each time slot and, therefore, FIFOMS achieves 100 percent throughput.
- Starvation free: Because of the FIFO property, FIFOMS provides a fairness guarantee. In other words, the time a packet can stay in the switch is bounded by a maximum value since an address cell will definitely get scheduled after all its competitors are served, which includes the earlier address cells in the other queues of the same input port and the earlier address cells in the virtual queues corresponding to the same output port of the other input ports.
- Enables fanout splitting: The destination output ports of a multicast packet can be served in separate time slots. It is allowed to send the data cell to some output ports in a slot and leave others for later slots. Fanout splitting is necessary for an algorithm to achieve high throughput under multicast traffic.

We have conducted extensive simulations to compare the performance of FIFOMS with other popular scheduling algorithms. And, the results fully demonstrate the superiority of FIFOMS in both the average packet delay and the queue space requirement.

ACKNOWLEDGMENTS

This research was supported in part by the US National Science Foundation under grant numbers CCR-0073085, CCR-0207999, and ECS-0427345.

REFERENCES

[1] H. Eriksson, "MBONE: The Multicast Backbone," *Comm. ACM,* vol. 37, no. 8, pp. 54-60, 1994.

[2] N. McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Trans. Networking,* vol. 7, no. 2, pp. 188-201, 1999.

[3] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input Queued Switch," *IEEE Trans. Comm.,* vol. 47, no. 8, pp. 1260-1267, 1999.

[4] R. Sivaram, C.B. Stunkel, and D.K. Panda, "HIPIQS: A High-Performance Switch Architecture Using Input Queuing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 275-289, Mar. 2002.

[5] H. Chao, B. Choe, J. Park, and N. Uzun, "Design and Implementation of Abacus Switch: A Scalable Multicast ATM Switch," *IEEE J. Selected Areas in Comm.,* vol. 15, no. 5, pp. 830-843, 1997.

[6] W. Chen, C. Huang, Y. Chang, and W.-Y. Hwang, "An Efficient Cell-Scheduling Algorithm for Multicast ATM Switching Systems," *IEEE/ACM Trans. Networking,* vol. 8, no. 4, pp. 517-525, 2000.

[7] M. Andrews, S. Khanna, and K. Kumaran, "Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch," *Proc. IEEE INFOCOM '99,* vol. 3, pp. 1144-1151, Mar. 1999.

[8] G. Han and Y. Yang, "A Random Graph Approach for Multicast Scheduling and Performance Analysis," *Proc. IEEE Int'l Conf. Computer Comm. and Networks (ICCCN '03),* pp. 270-275, Oct. 2003.

[9] Z. Zhang and Y. Yang, "Multicast Scheduling in WDM Switching Networks," *Proc. IEEE Int'l Conf. Comm. (ICC '03),* pp. 1458-1462, May 2003.

[10] S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Comm. Magazine,* vol. 36, no. 5, pp. 144-151, 1998.

[11] R. Ahuja, B. Prabhakar, and N. McKeown, "Multicast Scheduling Algorithms for Input-Queued Switches," *IEEE J. Selected Areas in Comm.,* vol. 15, no. 5, pp. 855-866, 1997.

[12] P. Giaccone, B. Prabhakar, and D. Shah, "An Efficient Randomized Algorithm for Input-Queued Switch Architecture," *Proc. IEEE Hot Interconnects 9,* Aug. 2001.

[13] P. Giaccone, B. Prabhakar, and D. Shah, "Towards Simple, High Performance Schedulers for High-Aggregate Bandwidth Switches," *Proc. IEEE INFOCOM '02,* June 2002.

[14] J.G. Dai and B. Prabhakar, "The Throughput of Data Switches with and without Speedup," *Proc. IEEE INFOCOM '00,* vol. 2, pp. 556-564, Mar. 2000.

[15] D. Shah, "Maximal Matching Scheduling Is Good Enough," *Proc. IEEE INFOCOM '03,* Dec. 2003.

[16] E. Leonardi, M. Mellia, M. Ajmone Marsan, and F. Neri, "Stability of Maximal Size Matching Scheduling in Input-Queued Cell Switches," *Proc. IEEE Int'l Conf. Comm. (ICC '00),* vol. 3, pp. 1758-1763, June 2000.

[17] E. Leonardi, M. Mellia, F. Neri, and M.A. Marson, "Bounds on Average Delays and Queue Size Averages and Variances in Input-Queued Cell-Based Switches," *Proc. IEEE INFOCOM '01,* vol. 2, pp. 1095-1103, Aug. 2001.

[18] S. Li, "Performance of a Nonblocking Space-Division Packet Switch with Correlated Input Traffic," *IEEE Trans. Comm.,* vol. 40, no. 1, pp. 97-108, 1992.

[19] R. LaMaire and D. Serpanos, "Two Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues," *IEEE/ACM Trans. Networking,* vol. 2, pp. 471-482, 1994.

[20] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast Scheduling for Input-Queued Switches," *IEEE J. Selected Areas in Comm.,* vol. 15, no. 5, pp. 855-866, 1997.

[21] N. McKeown and B. Prabhakar, "Scheduling Multicast Cells in an Input Queued Switch," *Proc. IEEE INFOCOM '96,* vol. 1, pp. 261-278, Mar. 1996.

[22] N. McKeown, "A Fast Switched Backplane for a Gigabit Switched Router," *Business Comm. Rev.,* vol. 27, no. 12, 1997.

[23] C. Partridge et al., "A 50-Gb/s IP Router," *IEEE/ACM Trans. Networking,* vol. 6, no. 3, pp. 237-248, 1998.

[24] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellesick, and M. Horowitz, "The Tiny Tera: A Packet Switch Core," *IEEE Micro,* vol. 17, no. 1, pp. 26-33, Feb. 1997.

[25] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Trans. Computer Systems,* vol. 11, no. 4, pp. 319-352, Nov. 1993.

[26] Y. Tamir and G. Frazier, "High Performance Multi-Queue Buffers for VLSI Communication Switches," *Proc. 15th Ann. Symp. Computer Architecture,* pp. 343-354, June 1988.

[27] M.J. Karol, M.J. Hluchyj, and S.P. Morgan, "Input versus Output Queueing on a Space-Division Packet Switch," *IEEE Trans. Comm.,* vol. 35, pp. 1347-1356, 1987.

[28] D. Stiliadis, "Efficient Multicast Algorithms for High-Speed Routers," *Proc. IEEE Workshop High Performance Switching and Routing (HPSR '03),* pp. 117-122, June 2003.

Deng Pan received the BS and MS degrees in computer science from Xian Jiaotong University, China, in 1999 and 2002, respectively. He is currently studying toward the PhD degree in computer science at the State University of New York at Stony Brook. His research interests include multicast scheduling and QoS guaranteed scheduling. He is a student member of the IEEE.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a professor of computer engineering and computer science at the State University of New York at Stony Brook. Dr. Yang's research interests include parallel and distributed computing and systems, high-speed networks, optical and wireless networks, and high-performance computer architecture. Her research has been supported by the US National Science Foundation (NSF) and US Army Research Office (ARO). Dr. Yang has published extensively in major journals and refereed conference proceedings and holds six US patents in these areas. She is an editor for the *IEEE Transactions on Parallel and Distributed Systems* and an editor for the *Journal of Parallel and Distributed Computing*. She has served on NSF review panels and the program/organizing committees of numerous international conferences in her areas of research. She is a senior member of the IEEE and a member of the IEEE Computer Society.