FIFO Based Multicast Scheduling Algorithm for VOQ Packet Switches

Deng Pan and Yuanyuan Yang

***Abstract*—Many networking/computing applications require high speed switching for multicast traffic at the switch/router level to save network bandwidth. However, existing queueing 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 the 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 queueing (VOQ), which has been shown to perform well under unicast traffic. However, it is impractical to use the traditional virtual output queued (VOQ) switches for multicast traffic, because a VOQ multicast switch has to maintain an exponential number of queues in each input port. In this paper, we give a novel queue structure for the input buffers of a VOQ multicast 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 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.**

I. 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. 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, [3] [4] [5] [6] [11]. Although there have been many scheduling algorithms proposed for different types of packet switches, how to efficiently organize and schedule multicast packets on the switches 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. A typical OQ switch, as shown in Fig.1(a), has a first-in-first-out (FIFO) queue at each output port to buffer the packets destined for that output port. OQ switches are shown to be able to achieve unity throughput, and can easily meet different QoS requirements, such as delay, bandwidth and fairness, by applying various scheduling algorithms. However, 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, where N is the number of the input ports of the switch. This deficiency makes OQ switches difficult to scale [12].

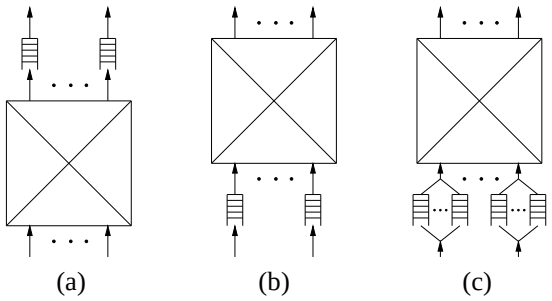

Fig. 1. Packet switches can be divided into two categories based on where the unserved packets are buffered. (a) Output queued switch. (b) Single input queued switch. (c) Multiple input queued Switch.

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.1(b), 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 the packet 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 is proved in [13] that when N is large, a single input queued switch running under the unicast i.i.d. Bernoulli traffic saturates at an offered load of approximately 0.586, and with correlated input traffic throughput can be even lower [8].

[6] 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. However, the performance of TATRA is 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 . 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.1(c). It is also called virtual output queue (VOQ) structure, since each queue stores those packets arrived from a given input port and destined for the same output port. HOL blocking is eliminated because a packet cannot be held up by a packet ahead of it that is destined for a different out-

This research was supported by the U.S. National Science Foundation under grant numbers CCR-0073085 and CCR-0207999.

Deng Pan is with Dept. of Computer Science, State University of New York, Stony Brook, NY 11794, USA.

Yuanyuan Yang is with Dept. of Electrical and Computer Engineering, State University of New York, Stony Brook, NY 11794, USA.

put. It is known that the VOQ switch structure can achieve 100% throughput for all independent arrival processes by using the maximum weight matching algorithm [2]. However, one problem for the traditional VOQ structure to be applied to multicast traffic is that a multicast packet has too many possible destinations, which is equal to $(2^N - 1)$ for a switch with N output ports. 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 [1] , PIM [12], 2DRR [9] and SERENA [7], but most of them are mainly designed for unicast traffic, because, as stated above, the traditional VOQ switch cannot handle multicast traffic. Recently, [15] extended the VOQ unicast scheduling algorithm WSGS [14] to multicast scheduling, but it restricts the maximal forwarding fanout and therefore is not able to fully utilize the multicast capability of a crossbar switching fabric.

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 VOQ multicast switch, we present a first-in-first-out based multicast scheduling algorithm, called FIFO Multicast Scheduling (FIFOMS). As will be seen, FIFOMS can fully use 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 achieve 100% throughput under uniformly distributed traffic. Our simulation results show that FIFOMS outperforms other input queueing based scheduling algorithms in average packet delay and buffer space requirement.

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.

II. Queue Structure for Multicast VOQ Switches

As mentioned above, under the existing queueing scheme of a VOQ switch, each input port needs to maintain $(2^N - 1)$ separate queues, which makes the VOQ switch impractical for scheduling multicast traffic. 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: 1) Scheduling - deciding for each input port which output port the packet should be sent to, and making arbitration when more than one input ports request for the same output port. 2) 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 the scheduling decision. 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 data content 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 size 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 port 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 will be used by the scheduling algorithm FIFOMS for two purposes: On the one hand, because all the address cells of the same packet

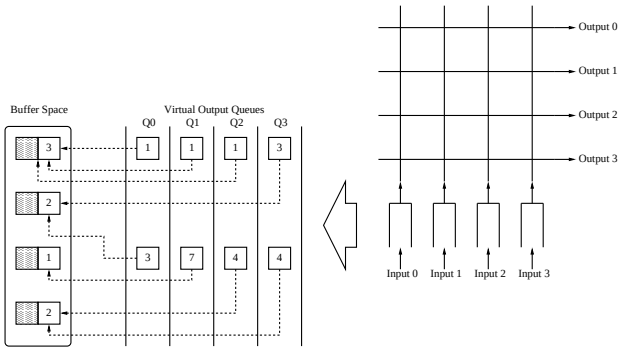

Fig. 2. An example of a 4×4 multicast VOQ switch. Left part shows the details of input port 0.

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. When an 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 queue are destined for the same output port, and only the address cells at the head of the queues can be scheduled.

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. Input port 0 has four packets that have not been fully transferred, and the packets entered the switch at the 1st, 3rd, 4th and 7th time slots, respectively. The fanout of the first packet is 3, and the packet still needs to be sent to output ports 0, 1 and 2, the destinations of the third packet are output ports 0 and 3, the destinations of the fourth packet are output ports 2 and 3, and the seventh packet is a unicast packet to output port 1.

III. First-In-First-Out Multicast Scheduling Algorithm (FIFOMS)

By using the modified queue structure, the VOQ switch now can 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, such as TATRA, 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, [1] [12] [9], [7] 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. Apparently, it does not take the characteristics of multicast traffic into consideration.

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 VOQ switch structure completely removes the HOL blocking, and enables FIFOMS to achieve 100% 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 cell delay, it is preferred for a multicast packet to 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 time stamp with the value equal to its arrival time, and uses the time stamp as a criterion in the scheduling arbitration. The time stamp criterion makes the multicast packets arrived earlier have better chance to be chosen by all its destination output ports when the output ports make scheduling decisions independently. Next, we will describe FIFOMS and its associated packet preprocessing algorithm.

A. 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 current time slot, and is put at the end of the corresponding queue.

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

B. First-In-First-Out Multicast Scheduling Algorithm (FIFOMS)

Similar to iSLIP [1] or PIM [12], FIFOMS is an iterative algorithm, and each iterative round consists of two steps: 1) Request - address cells at each input port make requests to their destination output ports for possible transmission. 2) Grant - each output port selects one request from all the requests it received, and grants the transmission to the corresponding address cell.

However, different from iSLIP and PIM, the accept step is not needed in FIFOMS, because in our request step, all the

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 = new_ packet_ body; // copy the message body

for (int 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 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 cells in a single time slot. In a scheduling round, FIFOMS has one fewer operational step, and less data exchange between inputs and outputs. The FIFOMS scheduling algorithm is described in Table 2, and we will explain each step in more detail next.

B.1 Request Step

In the request step, an input port finds the earliest HOL address cells, and give them priorities to send transmission requests. There are two possible cases. 1) If the input port is free in the current scheduling round (an input port or an output port is free if it has not been scheduled to send or receive a packet in the current round), it simply selects the HOL address cells whose time stamp is the smallest and 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 there may be more than one such address cells with the same smallest time stamp in an input port, which came from the same multicast packet. 2) Otherwise, if 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 is 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 cannot make requests any more.

B.2 Grant Step

After the request step, each 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 select one to grant.

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

TABLE 2
FIRST-IN-FIRST-OUT MULTICAST SCHEDULING ALGORITHM

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

Output: Scheduling decision.

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

            for all HOL address cells {
                if address cell's corresponding output port is free AND
                    its time stamp is equal to smallest_time_stamp {
                    the address cell makes a request to the corresponding
                        output port, and sends its time stamp as 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 granted address cell as reserved;
    }
} while some output port and input 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 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 data cell that
            the address cell points to by 1;
        if the data cell's fanoutCounter field becomes 0 {
            destroy the data cell;
        }
        remove the address cell from the head of queue;
}
```

B.3 Data Transmission

After the scheduling decisions are generated during the iterative 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 ports simultaneously. Thus, the algorithm can fully use the built-in multicast capability of the crossbar switch fabric.

B.4 Post Transmission Processing

After the transmission is completed, some post processing 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.

IV. Hardware Implementation and Complexity Analysis of the FIFOMS Scheduling Algorithm

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

A. 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 achieve high speed switching in practice.

The scheduler can be logically divided into two units as shown in Fig.3, 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 are 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 operation of each input port does not depend on each other, it can be performed in parallel. The selected address cells send their requests with time stamps as weights to the corresponding output ports. Then each output port uses a comparator to select the request with the smallest time stamp and grants the transmission to the corresponding address cell. Finally, before the next iterative round of FIFOMS could 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 crosspoint should be set to connect a particular input port with this output port.

B. 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 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 of 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.

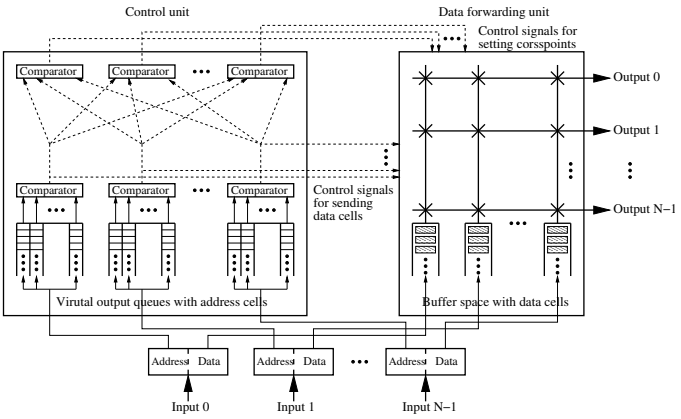

Fig. 3. The overall FIFOMS scheduler can be logically divided into two units, the control unit on the left and the data forwarding unit on the right.

C. 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. [16] pointed out the potential memory speedup problem, but since the destinations of a packet are independent and an address cell comprises only several 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 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 time stamp. If the operation is executed in a serial fashion, the time complexity is $O(N)$. If we use the parallel comparators as that in the WBA scheduler [10], the time complexity can be reduced to $O(1)$.

The convergence time has been a big 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 in the future rounds. But as will be seen later in the simulation results section in Fig.5, 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 relatively light traffic load.

V. Simulation Results

We have conducted extensive simulations to compare the performance of FIFOMS with other three scheduling algorithms: TATRA [6], iSLIP [1] 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 good performance as well as strict fairness. 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. In the simulation, 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 cell 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 fast switching ability. Despite its much stronger hardware requirement, in our simulations we also include a simple FIFO scheduling algorithm on the output queued structure as an ultimate performance benchmark for FIFOMS.

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

- Average input oriented delay: Input oriented delay represents the transmission delay from the sender's point of view. Specifically, it is equal to the maximum delay that the last destination output port of a multicast packet receives the packet.
- Average output oriented delay: Average output oriented delay represents the transmission delay from the receiver's point of view. It can be computed as the average of the delay that the multicast packet is delivered to all its destination output ports.
- Average queue size: Average queue size tells how long a new incoming packet needs to wait before transmission, and it also represents the space requirement of the algorithm. For FIFOMS and iSLIP, the queue size is defined to be the number of data cells in the buffer of an input port, in the sense that 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 loss of packets.

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. it reaches a stage where it is unstable to sustain the offered load).

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

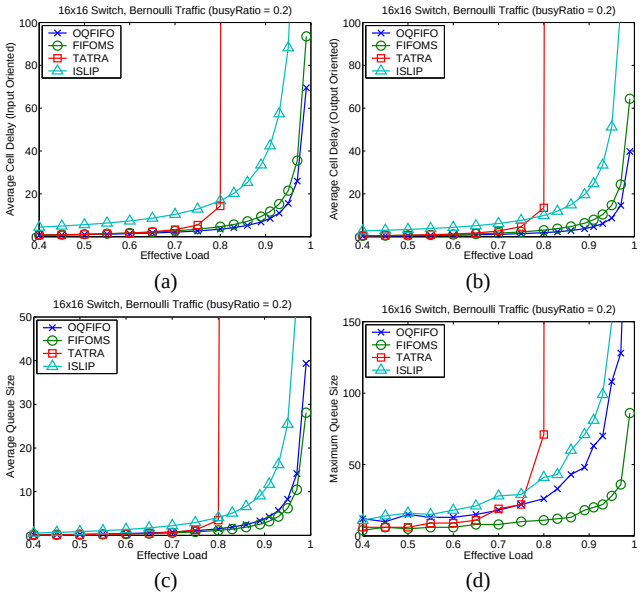

Fig. 4. Simulation results for a 16×16 switch under Bernoulli traffic with $b = 0.2$ (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

A. 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. The destination of the incoming packet is uniformly distributed over all possible multicast destinations. To be precise, a packet has the probability of b to be 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 $b = 0.2$ and a series of different p values are shown in Fig.4. As can be seen from the figure, in terms of input and output oriented average cell delays, FIFOMS closely matches OQFIFO, which has the best performance. In addition, FIFOMS outperforms all other three 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 80%, the delay of TATRA increases dramatically and it becomes unstable. It can also be observed that iSLIP has much longer average cell delay than all the other algorithms. This is because iSLIP is a scheduling algorithm specially designed for unicast traffic.

Fig.5 compares the convergence rounds between FIFOMS and iSLIP. We can see that the convergence rounds of both FIFOMS and iSLIP are not sensitive to the increasing of the traffic. 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%, under which iSLIP has

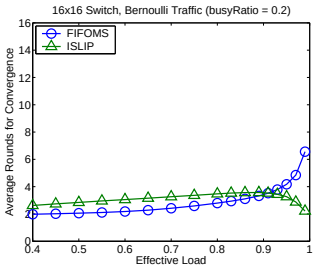

Fig. 5. Average convergence rounds of FIFOMS and iSLIP for a 16×16 switch under Bernoulli Traffic with $b = 0.2$.

already become unstable.

B. 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 this characteristics.

A uniform traffic can be described using two parameters p and $maxFanout$, in which p is the probability that an input port has a packet to arrive at a time slot, and $maxFanout$ 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. Therefore, for an $N \times N$ switch, the average fanout is $(1+maxFanout)/2$, and the effective load is $p \times (1 + maxFanout)/2$.

First, let's look at the simulation results when $maxFanout$ is set to 1, which is exactly the pure unicast traffic. There is no doubt that the well-known unicast scheduling algorithm iSLIP achieves short average cell delay. Although mainly designed for multicast traffic, FIFOMS manages to match and even surpass iSLIP on average cell delay, and is the best in terms of buffer requirement. On the contrary, the performance of TATRA is greatly affected by the HOL blocking, it can only reach a maximum effective load of about 55%, which is consistent with the theoretical analysis result of 0.586 in [13].

Simulations are also conducted under uniform traffic with $maxFanout = 8$ and the corresponding results are shown in Fig.7. FIFOMS consistently gives a satisfactory performance. It has the shortest average cell 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 to move the cells in the Tetris box.

C. 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 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

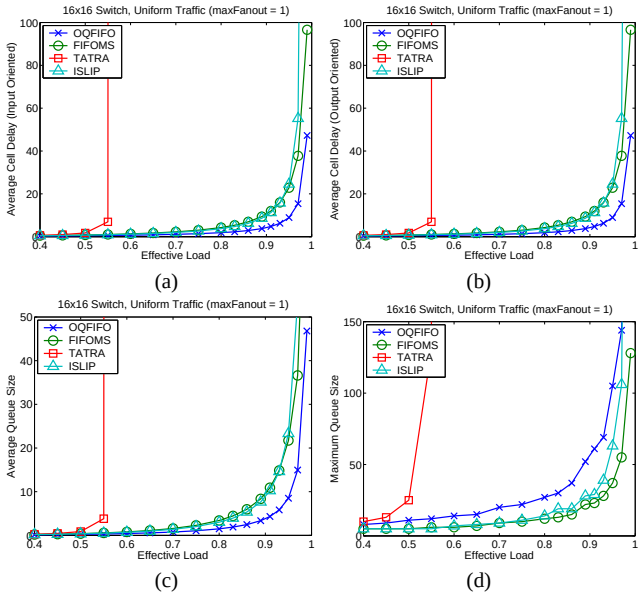

Fig. 6. Simulation results for a 16×16 switch under uniform traffic with maxFanout$= 1$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

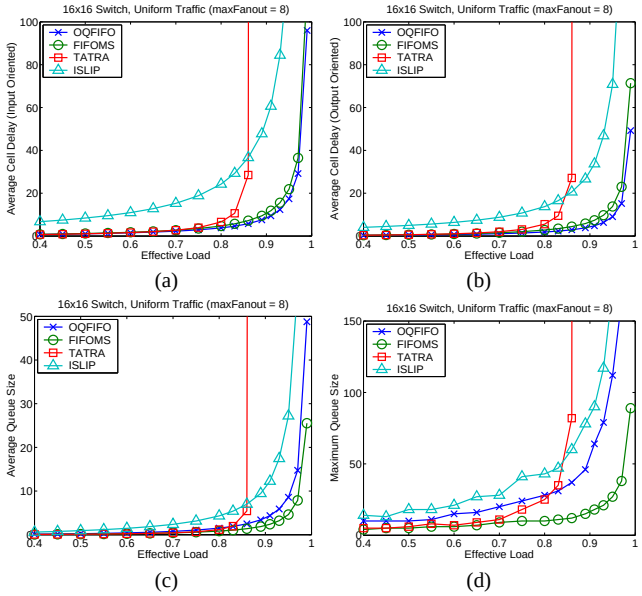

Fig. 7. Simulation results for a 16×16 switch under uniform traffic with maxFanout$= 8$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

between 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 to switch 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 to switch 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 average fanout is $p \times N$, the arrival rate is $E_{on}/(E_{off} + E_{on})$, and the effective load is $p \times N \times E_{on}/(E_{off} + E_{on})$. For easy comparison, we set E_{on} to be the same value 16 as in [6].

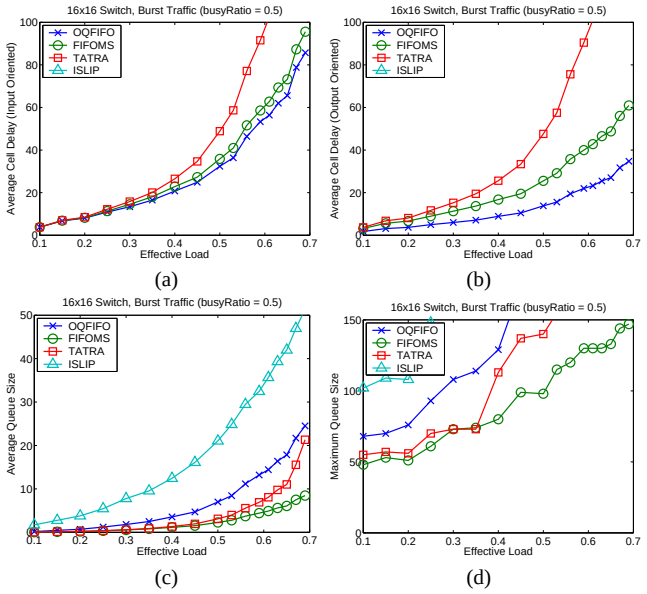

Fig. 8. Simulation results for a 16×16 switch under burst traffic with $q = 0.5$. (a) Average input oriented delay. (b) Average output oriented delay. (c) Average queue size. (d) Maximum queue size.

The simulation results for a 16×16 switch with $b = 0.5$ are shown in Fig.8. Due to the burst nature, the saturated throughput of all the algorithm becomes much lower. As to average cell delay, FIFOMS outperforms TATRA, but is not as good as OQFIFO. iSLIP saturates at a so small value that it cannot even be seen in the first two graphs, which is consistent with the theoretical analysis of [8]. As under other traffic modes, FIFOMS has the smallest queue space.

VI. 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 decreases 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, first-in-first-out 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% throughput under uniformly distributed traffic: Under uniform 100% 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% throughput.
- Starvation free: Because of the FIFO property, FIFOMS provides 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 include 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 chances. 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 cell delay and the queue space requirement.

References

[1] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188-201, Apr. 1999.

[2] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input queued switch," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1260-1267, Aug. 1999.

[3] W.-T. Chen, C.-F. Huang, Y.-L. 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, Aug. 2000.

[4] G. Han and Y. Yang, "A random graph approach for multicast scheduling and performance analysis," *Proc. of 12th IEEE International Conference on Computer Communications and Networks*, pp. 270-275, Dallas, Texas, October 2003.

[5] Z. Zhang and Y. Yang, "Multicast scheduling in WDM switching networks," *Proc. of IEEE 2003 International Conference on Communications (ICC '03)*, pp. 1458-1462, Anchorage, Alaska, May 2003.

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

[7] P. Giaccone, B. Prabhakar, and D. Shah, "Towards simple, high performance schedulers for high-aggregate bandwidth switches," *Proc. of IEEE INFOCOM '02*, New York, NY, June 2002.

[8] S.-Q. Li, "Performance of a nonblocking space-division packet switch with correlated input traffic," *IEEE Trans. Commun.*, vol. 40, no. 1, pp. 97-108, Jan. 1992.

[9] 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, Oct. 1994.

[10] B. Prabhakar, N. McKeown and R. Ahuja, "Multicast scheduling for input-queued switches," *IEEE J. Select. Areas Commun.*, vol. 15, no. 5, pp. 855-866, June 1997.

[11] N. McKeown and B. Prabhakar, "Scheduling multicast cells in an input queued switch," *Proc. of IEEE INFOCOM '96*, San Francisco, CA, USA, vol. 1, pp. 261-278, Mar. 1996.

[12] T. Anderson, S. Owicki. J. Saxe, and C. Thacker, "High-speed switch scheduling for local-area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319-352, Nov. 1993.

[13] M. J. Karol, M. J. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Commun.*, vol. 35, pp. 1347-1356, Dec. 1987.

[14] A. Smiljanic, "Flexible bandwidth allocation in high-capacity packet switches," *IEEE/ACM Trans. Networking*, vol. 10, no. 2, pp. 287-293, April 2000.

[15] A. Smiljanic, "Scheduling of multicast traffic in high-capacity packet switches," *Proc. of IEEE HPSR '02*, Kobe, Japan, pp. 72-77, May 2002.

[16] D. Stiliadis, "Efficient multicast algorithms for high-speed routers," *Proc. of IEEE HPSR '03*, Torino, Italy, pp. 117-122, Jun. 2003.