Multicast Scheduling in Feedback-based Two-stage Switch

Bing Hu and Kwan L. Yeung
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong, PRC
E-mail: {binghu, kyeung}@eee.hku.hk

Abstract — **Scalability is of paramount importance in high-speed switch design. Two limiting factors are the complexity of switch fabric and the need for a sophisticated central scheduler. In this paper, we focus on designing a scalable multicast switch. Given the fact that the majority traffic on the Internet is unicast, a cost-effective solution is to adopt a unicast switch fabric for handling both unicast and multicast traffic. Unlike existing approaches, we choose to base our multicast switch design on the load-balanced two-stage switch architecture because it does not require a central scheduler, and its unicast switch fabric only needs to realize N switch configurations. Specifically, we adopt the feedback-based two-stage switch architecture [10], because it elegantly solves the notorious packet mis-sequencing problem, and yet renders an excellent throughput-delay performance. By slightly modifying the operation of the original feedback-based two-stage switch, a simple distributed multicast scheduling algorithm is proposed. Simulation results show that with packet duplication at both input ports and middle-stage ports, the proposed multicast scheduling algorithm significantly cuts down the average packet delay and delay variation among different copies of the same multicast packet.**

Keywords—**Feedback-based two-stage switch, scalable multicast switch, load-balanced switch**

I. INTRODUCTION

The migration of broadcasting and multicasting services, such as cable TV and multimedia-on-demand to packet oriented networks, will play a dominant role in the near future. These highly popular applications have the potential of loading up the Internet. To keep up with the bandwidth demand of such applications, the next generation of packet switches/routers need to provide efficient multicast switching and packet replication.

When a multicast packet arrives at a switch, the set of output ports the packet destined for, i.e. the packet's *fan-out set*, is retrieved from the local forwarding table (like IP multicast). The cardinality of the fan-out set, i.e. its *fan-out*, denotes the number of copies that the packet should be cloned. Packets arrived at the same input port and destined for the same fan-out set belong to the same multicast flow. The total number of possible multicast (and unicast) flows at an input port is 2^N-1. An *admissible multicast traffic pattern* requires no over-subscribed input and output ports. That means the packet arrival rate at each input port should be less than or equal to its capacity, or 1 packet/slot. Similarly, the aggregated packet arrival rate at each output port (after packet duplication) must also be smaller than or equal to 1 packet/slot. A multicast switch aims at providing 100% throughput for any admissible multicast traffic pattern with minimum possible packet delay.

For the sake of scalability, multicast switches are mainly designed based on input-queued switch architecture, where a centralized scheduler is responsible for scheduling. Switch fabrics used can be bufferless [1-5] or buffered [6-9]. For multicast switches based on bufferless switch fabrics [1-5], in-switch multicast capability (i.e. in-switch packet duplication and forwarding) is usually assumed, where an input port can send a (multicast) packet to multiple output ports in a single time slot. Such multicast fabrics are more expensive than their unicast counterparts. Besides, the centralized scheduling algorithms are usually derived from their unicast counterparts. Note that even for (simpler) unicast switches, a major bottleneck is the implementation of the centralized scheduler.

For multicast switches with buffered switch fabrics, they mainly adopt the buffered crossbar [6-9] as their switch fabrics. A crossbar switch has N^2 crosspoints. Each crosspoint requires a dedicated/expensive in-switch buffer for temporarily storing packets coming from an input. Nevertheless, the centralized scheduler is simpler than those with bufferless fabrics, and it operates in two phases: input arbitration (for dispatching a packet from an input to a crosspoint buffer), and output arbitration (for sending a packet from a crosspoint buffer to an output port). Both arbitration processes rely on the occupancy of the distributed queues (one at each crosspoint buffer). The communication overheads for gathering the queue occupancy are significant.

In short, two limiting factors for high-speed multicast switch design are the switch fabric complexity and the need for a sophisticated central scheduler. Given the fact that the majority traffic on the Internet is unicast, a cost-effective solution is to use a unicast switch fabric to carry both unicast and multicast traffic. To this end, we propose to base our multicast switch design on the load-balanced two-stage switch architecture [11] because it does not require a central scheduler, and its switch fabric is only required to realize N switch configurations (as compared to $N!$ switch configurations for an input-queued switch). Specifically, we adopt the feedback-based two-stage switch architecture [10], because it elegantly solves the notorious packet mis-sequencing problem associated with the load-balanced switch, and yet renders an excellent throughput-delay performance. By slightly modifying the operation of the feedback-based two-stage unicast switch, a simple distributed

This work was supported in part by Cisco Research Center.

978-1-4244-5174-6/09/$26.00 ©2009 IEEE

multicast scheduling algorithm is proposed. Simulation results show that with packet duplication at both input ports and middle-stage ports, our proposed multicast scheduling algorithm is effective in cutting down both average packet delay and delay variation among different copies of the same multicast packet.

The rest of the paper is organized as follows. In the next section, we review some related work on multicast switch design. The feedback-based two-stage switch [10] is recapped in Section III. The new multicast scheduling algorithm is introduced in Section IV and simulation results are presented in Section V. We conclude the paper in Section VI.

II. Related Work on Multicast Swicth Design

A. Multicast switches based on bufferless switch fabrics

Multicast switches based on bufferless switch fabrics [1-5] usually assume in-fabric multicast capability (i.e. in-fabric packet duplication and forwarding), and require a rather sophisticated central scheduler. In [1], each switch input port maintains N+1 virtual queues, N for unicast and one for multicast. Priority is given to schedule multicast traffic. If there are still idle inputs/outputs after scheduling multicast packets, unicast packets are considered to increase switch utilization. Although a multicast packet can be "split" to send in multiple time slots, multicast traffic suffers from severe head-of-line (HOL) blocking due to the single multicast queue.

In [2], the number of multicast queues is increased to m to reduce HOL blocking. When a multicast packet arrives, it selects a multicast queue to join in order to balance the loading among different multicast queues. But packets assigned to different queues generally have overlapped fan-out sets. Priority is given to schedule a unicast packet first or a multicast packet first depending on the service ratio between the two classes. An iterative algorithm is also adopted to maximize the throughput in each time slot.

In [3], *packet splitting* is allowed to further cut down the HOL blocking. Specifically, each input maintains k unicast/multicast shared queues, one for each non-overlapped set of outputs. When a multicast packet arrives and if its fan-out set intersects with the fan-out sets of multiple queues, packet-splitting "breaks" the original packet into "smaller" ones, each with a modified fan-out set (such that no intersection with the fan-out set of the queue it joins). An iterative algorithm is then used to maximize the switch throughput. Simulation results show that high throughput can only be achieved with a large number of iterations. But a large number of iterations is not suitable for high-speed implementation.

In [4], the number of unicast/multicast shared pointer queues increases to $k = N$, one for each output port (like the classic VOQs for unicast traffic). When a multicast/unicast packet arrives, it is time-stamped and stored in a shared memory. Then its memory address (i.e. a pointer) is stored in all pointer queues that overlap with the packet's fan-out set. An iterative scheduling algorithm based on the timestamps of buffered packets is designed for maximizing throughput. The major problem with this approach, again, is its high communication overheads.

In [5], dynamic queuing policies are studied, where packet splitting upon arrival is not allowed. The switch needs to identify active flows and then assign them to different shared multicast queues based on the current switch load.

B. Buffered crossbar based multicast switches

Buffered crossbar switch architecture [12] is touted for its technology feasibility and simpler central scheduler. The scheduling consists of input arbitration (for dispatching a packet from an input to a crosspoint buffer), and output arbitration (for sending a packet from a crosspoint buffer to an output port). Both arbitration processes rely on accurate knowledge of distributed queues (one at each crosspoint buffer). But collecting queue occupancy may incur high communication overheads.

Buffered crossbar has been extended to support multicast traffic [6-9]. MURS [6] gives priority to schedule unicast and multicast traffic in a round robin fashion. Specifically, if unicast gets priority in time slot t, unicast traffic will be scheduled first. If there are still idle outputs after scheduling unicast traffic, multicast traffic is considered. Then in slot t+1, multicast traffic gets the scheduling priority.

To reduce the hardware cost, I-SMCB [7] and O-SMCB [8] aim at cutting down the crosspoint buffers from N^2 to $N^2/2$. The key idea is to share one crosspoint buffer by two adjacent input ports [7] or two adjacent output ports [8]. But such a hardware cost reduction is offset by its throughput degradation. In [9], the theoretical relationship between throughput performance and crosspoint buffer size is studied under a special multicast traffic pattern. It is concluded that to avoid throughput degradation, the amount of buffer to be deployed at every crosspoint must scale logarithmically with the switch size N.

III. Feedback-based Two-stage Switch

A load-balanced two-stage unicast switch [11] does not require a central scheduler, yields close to 100% throughput, and has very simple switch fabric. But it faces the problem of packet mis-sequencing due to the variable delay a packet can experience at middle-stage ports. Among various efforts in solving the packet mis-sequencing problem, the feedback-based two-stage switch architecture [10] provides an elegant solution while not sacrificing the switch's throughput-delay performance. Before extending the feedback-based two-stage switch to effectively support multicast traffic, we summarize its basic operations in this section.

An $N \times N$ feedback-based two-stage switch [10] is shown in Fig. 1, where $VOQ_1(i,k)$ represents the Virtual Output Queue (VOQ) at input port i with packets destined for output k, and $VOQ_2(j,k)$ denotes the VOQ at middle-stage port j with packets destined for output k. Each $VOQ_2(j,k)$ only requires a single packet buffer. In a load-balanced two-stage switch [11], each of the two switch fabrics is configured by a pre-determined and

periodic sequence of N switch configurations. The only requirement is that each input visits each output (of the same switch fabric) exactly once in the sequence. Accordingly, the switch fabric only needs to realize N switch configurations instead of $N!$ for an input-queued switch.

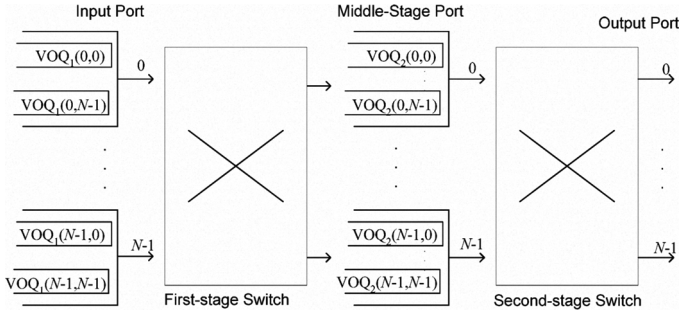

Fig. 1: Feedback-based two-stage switch.

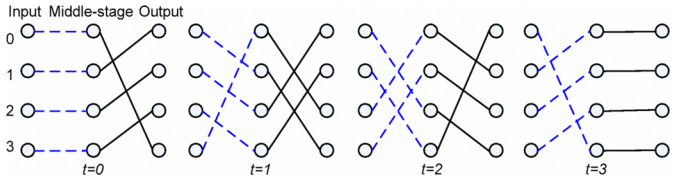

Fig. 2: Two sequences of configurations for a 4 x 4 feedback-based two-stage switch

The feedback-based two-stage switch [10] has some additional requirements on the two sequences of configurations to be used by the two switch fabrics in Fig. 1. An example is shown in Fig. 2, where the dashed lines show the configurations used by the first fabric and the solid lines show the configurations used by the second fabric. The first configuration sequence is constructed such that input port i is connected to middle-stage port j at time slot t, where $j = (i + t) \bmod N$. The second configuration sequence is constructed based on the property of staggered symmetry, which refers to the fact that for any middle-stage port j, if it is connected to output k at time slot t, then at next slot $(t+1)$ input k is connected to the same middle-stage port j.

Further note that each $VOQ_2(j,k)$ in the feedback-based two-stage switch (Fig. 1) only has a single packet buffer. As such, an N-bit vector is sufficient to denote the occupancy of the N $VOQ_2(j,k)$'s at each middle-stage port. This vector is piggybacked onto the data packet sent to output k, and is then made available to input k at negligible cost, because both input k and output k reside on the same switch linecard. Due to the staggered symmetry property of the two sequences of configurations used, input k will connect to middle port j in the next time slot. Therefore, the received occupancy vector provides the just-in-time feedback to the local packet scheduler at input k. Without loss of generality, the local packet scheduler at each input port is based on longest queue first (LQF), where a packet from the longest $VOQ_1(i,k)$ is selected for sending if the corresponding middle-stage $VOQ_2(j,k)$ is empty. The timing diagram in Fig. 3 further shows the pipelined packet transmission in the two switch fabrics. Note that the occupancy vector is generated by taking the in-flight packet in the first fabric into account.

In each time slot, the (pre-determined) configuration in the second stage switch fabric allows up to N packets to be delivered to their respective outputs. The in-order packet delivery is guaranteed because every packet belonging to the same flow will experience the same amount of middle-stage port delay, no matter which middle-stage port it passes through. For details, please refer to [10,13].

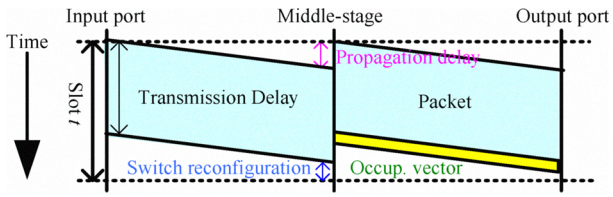

Fig. 3: The timing diagram for feedback-based two-stage switch

IV. Multicast Scheduling using Feedback-based Two-stage Switch

A. Multicast scheduling

We extend the feedback-based two-stage switch in Fig. 1 to effectively support multicast traffic. At each input port, in addition to the N unicast $VOQ_1(i,k)$'s, we add another m shared queues for multicast (not shown). We adopt a simple queuing policy that divides the outputs into m equal and non-overlapped sets (assuming N/m is an integer), where set x $(1 \le x \le m)$ contains outputs $\{(x\text{-}1)N/m, (x\text{-}1)N/m+1, \ldots, x \cdot N/m\text{-}1\}$. Packet splitting is used to "split" multicast packets to join different queues. So when a multicast packet arrives and if its fan-out set intersects with the fan-out sets of multiple queues, then the original packet is "split" into "smaller" ones, each with a modified fan-out set (which will not intersect with the fan-out set of the target queue). Note that the packet after splitting usually remains as a multicast packet but with a smaller fan-out set. It is worth to note that when m=1, all multicast packets share the same multicast queue; and when m=N, packet splitting converts all multicast packets into unicast.

Without loss of generality, we assume the two stages of switch fabrics are configured using the two sequences of configurations shown in Fig. 2. In each time slot, based on the received occupancy vector of middle-stage port k, input i selects a packet for sending among its $N+m$ local queues. Priority is given to schedule multicast traffic by examining the m multicast queues first. Specifically, the HOL packet whose fan-out set has the largest overlap with the set of empty queues at middle-port k is selected. (If no overlap, a unicast packet is selected instead.) A copy of the selected packet is sent to the middle-port together with an N-bit *duplication vector*, which identifies the overlap between the empty $VOQ_2(j,k)$'s and the packet fan-out set. Then, the fan-out set of the selected multicast packet is updated to exclude those in the duplication vector. If the updated fan-out set is empty, the selected multicast packet is removed from the multicast queue. When a

packet arrives at the middle-stage port, it will be cloned and stored at the corresponding empty (unicast) $VOQ_2(j,k)$'s based on the duplication vector.

If there are no backlogged multicast packets *or* none of them can be selected (due to zero-overlap between the empty $VOQ_2(j,k)$'s and any multicast packet's fan-out set), we select a unicast packet for sending using the LQF scheduler. In this case, the duplication vector is set to all 0's. Note that the packet transmission in the second-stage switch fabric is the same as in a unicast switch [10]. Following the pre-determined sequence of configurations, when middle-stage port j connects to output k, the packet (if any) at $VOQ_2(j,k)$ is sent together with the occupancy vector of middle-port j.

B. Discussions

In our proposed multicast scheduling algorithm, packet duplication takes place at both input ports and middle-stage ports. Packet duplication at input ports "breaks" a multicast packet into smaller ones. Since multicast packets in different multicast queues have non-overlapped fan-out sets, both HOL blocking and output contention can be eased. Besides, storing multicast packets at inputs reduces the input port buffer requirement. Since both two switch fabrics in Fig. 1 are unicast, a multicast packet is sent in the first fabric as a unicast packet. In-fabric duplication (as [1-5]) is *not* required. When a split multicast packet arrives a middle-stage port, the second stage packet duplication occurs, which converts all multicast packets into unicast for delivering by the second switch fabric.

When there is only a single multicast queue (m = 1), all packet duplication is carried out at middle-stage ports. Under light traffic, input port queue size can be minimized. But for heavy traffic, the switch will experience severe HOL blocking because a multicast packet will not be removed (from the only queue) until all its copies are sent. With $m > 1$, packet splitting ensures that packet duplication occurs partially at input ports and packets in different queues have non-overlapped destinations. This reduces the HOL blocking. Let the switch size be N. When m=N, all packet duplication is carried out at input ports. In this case, there is no need for "multicast" queues because they only store unicast packets. In other words, each input port only needs to maintain N unicast queues. The HOL blocking is also completely eliminated.

Unlike the feedback-based two-stage unicast switch [10], the load-balancing in the first stage switch is based on multicast packets. Extensive simulation results show that the final unicast traffic presented to the second stage switch is generally uniform. This accredits to the use of the single-packet-buffer per middle-stage $VOQ_2(j,k)$, and the efficient feedback mechanism for reporting the middle-stage port occupancy. To further increase the buffer utilization, we can use pointer queues [4] to separately store a packet and its memory address. So a multicast packet is only required to store once at an input port, and an entry in $VOQ_1(i,k)$ only contains the memory address of the packet. Likewise, this can be applied to buffers at middle-stage ports.

The proposed multicast scheduling algorithm inherits the in-order packet delivery property from its unicast counterpart [10]. This is because we can treat each distributary of a multicast flow as a unicast flow. In [10], it has been shown that packets belonging to the same unicast flow always experience the same middle-stage port delay. Therefore, when they arrive at the output port, they will be in order. If packets belonging to every distributary flow orderly arrive at their respective outputs, the corresponding multicast flow will not experience packet mis-sequencing problem.

V. PERFORMANCE EVALUATIONS

To the best of our knowledge, our proposed multicast scheduling is the only one that does not rely on a central scheduler, and its switch fabric only needs to realize N switch configurations (instead of $N!$). To study its performance, we vary the number of multicast queues (m) at each input port. In our simulations, we try to distinguish between the overall *average* delay experienced by all copies (T_c) of a multicast packet and the average delay experienced by the *last-copy* (T_p) of all multicast packets. T_p corresponds to the worst-case delay and provides us some insight on the delay variation among different copies of a multicast packet. For multicast packets with fan-out k, $T_c(k)$ and $T_p(k)$ denote their average delay and average last-copy delay respectively. They show the fairness performance in handling packets with different fan-outs. Although we only present simulation results for switch with size N=32 below, the same conclusions and observations apply for other switch sizes.

A. Bernoulli uniform mixing traffic

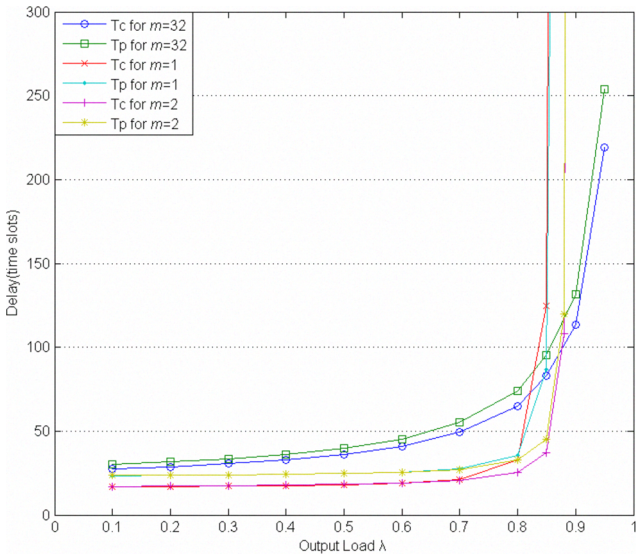

Fig. 4: Delay vs output load, with uniform mixing traffic

At every time slot for each input, a packet arrives with probability p (i.e. input load is p). If a packet arrives, it has equal probability of being unicast or multicast, i.e r=0.5. If the packet is unicast, it destines to each output with equal

probability. If the packet is multicast, its fan-out size k is randomly selected between [2, 32], and the identity of each output in the fan-out set is also randomly selected from all output ports. Fig. 4 shows the switch delay performance against switch output load λ, where

$$\lambda = p[0.5+0.5(2+32)/2] = 9p. \quad (1)$$

To ensure the traffic in our simulations is always admissible, we must have $\lambda \leq 1$ (or $p \leq 1/9$).

From the delay-throughput performance in Fig. 4, we can see that for output load $\lambda < 0.85$, m=1 and m=2 provide a lower average packet delay than m=32. At $\lambda = 0.7$, m=2 cuts down the overall average delay (T_c) by 58.8% and the average last-copy delay T_p by 51%. When $\lambda > 0.85$, m=32 (packet duplication at input ports only) yields a better/lower delay performance because there is no HOL blocking, while the HOL in m=1 (packet duplication at middle-stage ports only) is intensified with the traffic load. This also explains why m=2 (packet duplication at both input and middle-stage ports) is better than m=1.

Fig. 5 shows the delay performance against different fan-outs, while fixing $\lambda = 0.7$. When m=2, we can see that $T_c(k)$, the average delay for packets with fan-out k, is the lowest, and remains almost constant at 20 slots as fan-out k increases. Even $T_p(k)$, the average last-copy delay for packets with fan-out k, increases rather slowly with k. This shows that m=2 is fair in handling packets with different fan-outs. On the contrary, with m=32, both $T_c(k)$ and $T_p(k)$ increase more rapidly with fan-out size.

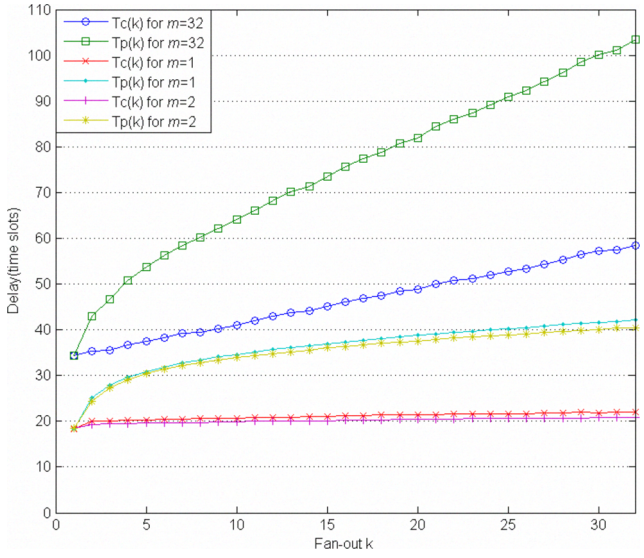

Fig. 5: Delay vs fan-out, with uniform mixing traffic at λ=0.7

B. *Bursty mixing traffic*

We use the same traffic generator except that bursty arrivals are modeled by the ON/OFF traffic model. In the OFF state, no packet arrives. In the ON state, a packet arrival is generated in every time slot, which has equal probability of being unicast or multicast. Given the average input port load p and average burst size s, the state transition probability from OFF to ON is $p/[s(1-p)]$, and from ON to OFF is $1/s$. Simulation results in Figs. 6 & 7 are based on s=30 packets. Again, we can express the aggregated load at each output port by (1).

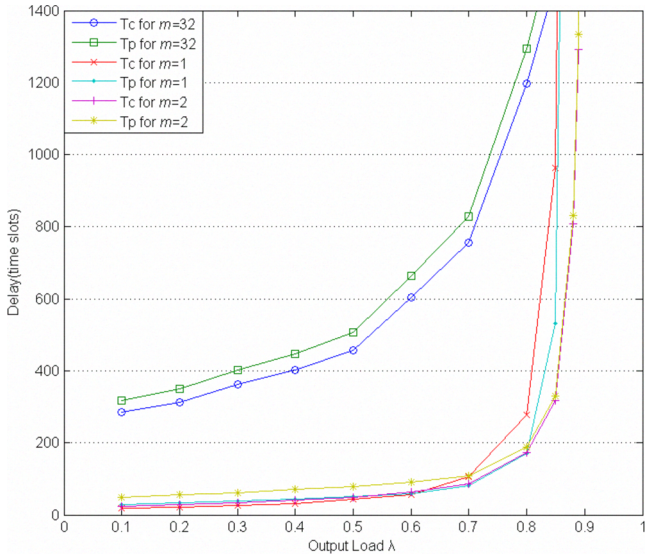

Fig. 6: Delay vs output load, with bursty mixing traffic

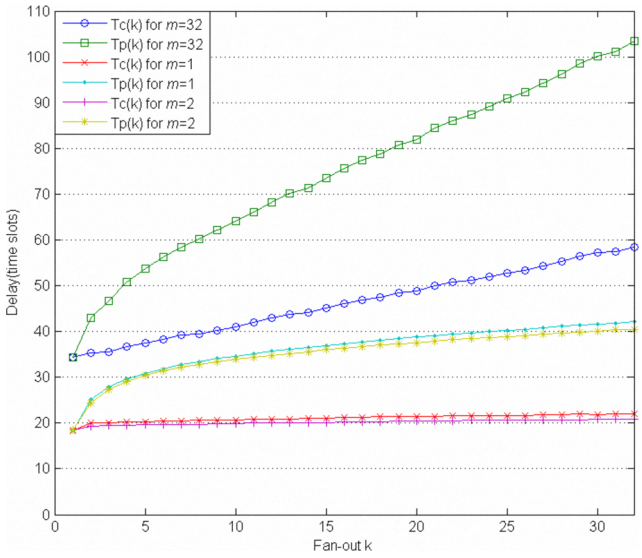

Fig. 7: Delay vs fan-out, with bursty mixing traffic at λ=0.7

From Fig. 6, the performance gap between m=2 and m=32 is much wider than that in Fig. 4. This is because bursty traffic causes more unevenly distributed queue sizes in the input ports when m=32. With m=2, packet duplication *mainly* occurs at middle-stage ports. In this case, both input port queue size and input port delay are reduced. With m=1, packet duplication only occurs at middle-stage ports. The throughput is suffered from the severe HOL blocking. From Fig. 7, we can again see that m=2 is fair in handling packets with different fan-outs. Although m=32 also gives improved fairness performance, this

is at the cost of very high average delay ($T_c(k)$> 750 slots).

C. Binomial mixing traffic

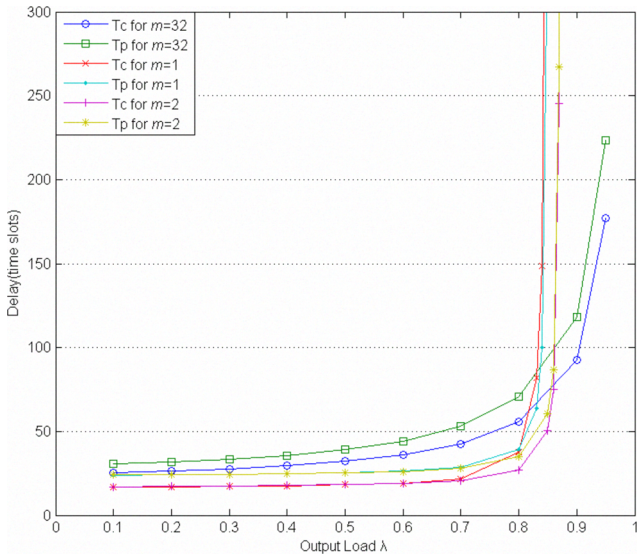

Fig. 8: Delay vs output load, with non-uniform binomial mixing traffic

Binomial mixing traffic [5] is the same as the Bernoulli uniform mixing traffic model except in generating the fan-out size of a multicast packet. Let P_k be the probability of generating a fan-out set with size k. The k destinations are uniformly distributed over all output ports. The value of k is chosen according to a non-uniform binomial distribution, with mean fan-out h:

$$p_k = C_N^k (\frac{h}{N})^k (1-\frac{h}{N})^{N-k}$$

In our simulations, we set mean fan-out h = 17. Then the output load λ is:

$$\lambda = p[0.5+0.5\times 17] = 9p$$

The delay performance shown in Fig. 8 is comparable with that in Fig. 4. This is because the two traffic models are quite similar. Specifically, they have the same Bernoulli packet arrival, same average fan-out size of 17, and their fan-out sets are all uniformly selected from all outputs. We skip the figure of delay vs fan-out because it has a similar trend as that in Fig. 5.

From the simulation results above, we can see that setting m=2 is sensible as it ensures sufficiently low packet delay and high throughput. Besides, the extra complexity involved in maintaining two multicast queues is marginal.

VI. Conclusions

In this paper, we focused on designing a scalable multicast switch based on the feedback-based two-stage switch architecture. The feedback-based switch is selected because it does not require a central scheduler, its switch fabric is very simple, and it elegantly solves the notorious packet mis-sequencing problem without sacrificing the switch throughput-delay performance. By slightly modifying the operation of the feedback-based two-stage unicast switch, a simple distributed multicast scheduling algorithm was proposed. Simulation results showed that with packet duplication at both input ports and middle-stage ports, the proposed multicast scheduling algorithm is effective in cutting down both average packet delay and delay variation among different copies of the same multicast packet.

References

[1] M. Andrews, S. Khanna and K. Kumaran, "Integrated scheduling of unicast and multicast traffic in an input-queued switch," *INFOCOM,* pp. 1144–1151, March 1999, New York, USA.

[2] W. Y. Zhu and M. Song, "Integration of unicast and multicast scheduling in input-queued packet switches," *Computer networks*, Vol. 50, pp. 667-687, 2006.

[3] S. Gupta and A. Aziz, "Multicast scheduling for switches with multiple queues", *IEEE Hot Interconnects'02,* August 2002, Stanford, CA, USA

[4] D. Pan and Y. Y. Yang, "FIFO-based multicast scheduling algorithm for virtual output queued packet switches," *IEEE Tran. on Computers*, Vol. 54, pp. 1283 – 1297, Oct. 2005.

[5] A. Bianco, P. Giaccone, C. Piglione and S. Sessa, "Practical algorithms for multicast support in input queued switches" *IEEE Workshop on High Performance Switching and Routing*, June 2006, Poznan, Poland.

[6] L. Mhamdi and S. Vassiliadis, "Integrating uni- and multicast scheduling in buffered crossbar switches," *IEEE Workshop on High Performance Switching and Routing*, June 2006, Poznan, Poland.

[7] Z. Q. Dong and R. R. Cessa, "Packet switching and replication of multicast traffic by crosspoint buffered packet switches," *IEEE Workshop on High Performance Switching and Routing*, May 2007, New York, USA.

[8] Z. Q. Dong and R. R. Cessa, "Input- and output-based shared-memory crosspoint-buffered packet switches for multicast traffic switching and replication," *ICC 2008*, May 2008, Beijing, China.

[9] P. Giaccone and E. Leonardi, "Asymptotic performance limits of switches with buffered crossbars supporting multicast traffic," *IEEE Tran. on Information theory*, Vol. 54, No. 2, Feb. 2008

[10] K. L. Yeung, B. Hu and N.H. Liu, "A novel feedback mechanism for load balanced two-stage switches," *ICC 2007*, June 2007, Glasgow, Scotland.

[11] C. S. Chang, D. S. Lee and Y. S. Jou, "Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering," *Computer Communications*, Vol. 25, pp. 611 – 622, 2002.

[12] K. Yoshigoe and K. J. Christensen, "An evolution to crossbar switches with virtual output queueing and buffered crosspoint," *IEEE Network*, vol. 17, no. 5, pp. 48–56, Sep. 2003.

[13] B. Hu and K. L. Yeung, "On joint sequence design for load-balanced two-stage switch architecture," *IEEE Workshop on High Performance Switching and Routing*, May 2008, Shanghai, China.