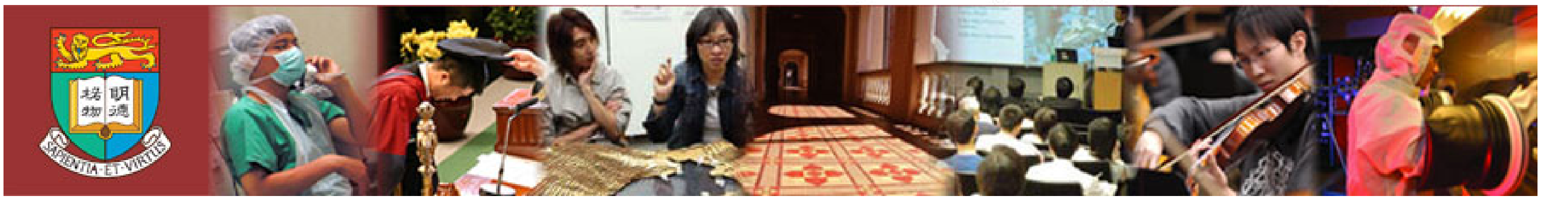

Title	FTMS: an efficient multicast scheduling algorithm for feedback-based two-stage switch
Author(s)	He, C; Hu, B; Yeung, LK
Citation	The 2012 IEEE Global Communications Conference (GLOBECOM 2012), Anaheim, CA., 3-7 December 2012. In Globecom. IEEE Conference and Exhibition, 2012, p. 2541-2546
Issued Date	2012
URL	http://hdl.handle.net/10722/165315
Rights	Globecom. IEEE Conference and Exhibition. Copyright © IEEE.

FTMS: An Efficient Multicast Scheduling Algorithm for Feedback-based Two-stage Switch

Chunzhi He*, Bing Hu† and Kwan L. Yeung*
*Department of Electrical and Electronic Engineering
The University of Hong Kong, Hong Kong, PRC
Email: {czhe, kyeung}@eee.hku.hk
†Department of Information Science and Electronic Engineering
Zhejiang University, Hangzhou, PRC
E-mail: binghu@zju.edu.cn

***Abstract*—Two major challenges in designing high-speed multicast switches are the expensive multicast switch fabric and the highly complicated central scheduler. While the recent load-balanced switch architecture uses simple unicast switch fabric and does not require a central scheduler, it is only good at handling unicast traffic. In this paper, we extend an existing load-balanced switch called feedback-based two-stage switch to support multicast traffic. In particular, an efficient multicast scheduling algorithm (FTMS) is designed. With FTMS, head-of-line (HOL) packet blocking at each input port is eliminated by adopting "pointer" queues. To cut down queuing delay, packet replication is carried out at middle-stage ports. As compared with other multicast scheduling algorithms, simulation results show that our FTMS always provides the highest throughput.**

I. Introduction

Due to an ever increasing amount of multicast and broadcast services on the Internet, the need for building high-speed switches/routers for efficient support of multicast traffic becomes urgent. Multicast switches are usually designed based on their unicast counterparts [1], [2]. To this end, a unicast switch can be designed based on either an input-queued or an output-queued switch architecture. Although output-queued switch provides the best delay-throughput performance, it is not scalable due to the high speedup requirement. Input-queued switch [2] does not require speedup and is thus the preferred choice for high-speed switch design.

In an input-queued unicast switch, a central scheduler is adopted for resolving packet contention. Virtual output queueing (VOQ), where at each input port a dedicated queue is maintained for packets destined for each output, is also adopted to eliminate the head-of-line (HOL) blocking phenomenon. The associated scheduling problem is equivalent to the matching problem in a bipartite graph. While optimal scheduling algorithm that yields 100% throughput exists (e.g. Maximum Weight Matching (MWM) [3]), fast iterative scheduling algorithms (e.g. iSLIP [1]) with sub-optimal performance are more desirable in practice.

This project is supported by the small project funding 201007176210, The University of Hong Kong.

Supporting multicast traffic is inherently more challenging [4]. Many input-queued multicast switches have been designed based on their unicast counterparts and they usually require in-switch packet replication, or multicast switch fabric. Packet replication is a process of cloning a multicast packet for sending to different output ports according to its multicast address, or fanout set. As compared with unicast switch fabric, multicast switch fabric is more expensive to build. Besides, multicast scheduling algorithms are also more complicated than their unicast counterparts. Therefore, two major challenges in designing high-speed multicast switch are the expensive multicast switch fabric and the highly complicated central scheduler.

Recently, unicast switches based on the concept of load-balancing have been designed [5]–[8]. A load-balanced switch consists of two stages of switch fabrics as shown in Fig. 1, where the first stage is responsible for load balancing and the second stage for packet delivery. It is shown that load-balanced switch is scalable, requires no central scheduler, and each of its (unicast) switch fabrics only needs to realize N switch configurations instead of $N!$ switch configurations. More recently, the notorious packet out-of-order problem [5] associated with the load-balanced switch has also been solved by the feedback-based two-stage switch in [8].

In this paper, we follow the load-balanced approach to design an input-queued multicast switch. Motivated by the fact that the feedback-based switch elegantly solves the out-

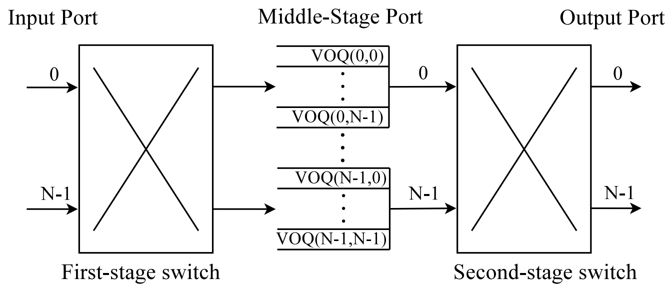

Fig. 1. A load-balanced two-stage switch.

of-order problem for unicast traffic, we extend it to support multicast. In particular, we propose a simple distributed multicast scheduling algorithm, called Feedback-based Two-stage Multicast Scheduling (FTMS). With FTMS, the HOL packet blocking at input ports is solved by adopting the "pointer" queues. Packet replication is carried out at middle-stage ports, where the two stages of switch fabric remain unicast/simple. As compared with other multicast scheduling algorithms, simulation results show that our FTMS effectively reduces average packet delay and achieves the highest throughput under various traffic conditions.

The rest of the paper is organized as follows. In Section II, recent efforts on designing multicast scheduling algorithms are reviewed. In Section III, the original feedback-based unicast switch [8] is summarized. Our FTMS is detailed in Section IV, and simulation results are presented in Section V. Finally, we conclude the paper in Section VI.

II. Related Work

Most existing multicast switches [9]–[12] require in-switch packet replication (i.e. multicast switch fabric), and a sophisticated central scheduler for maximizing switch performance.

TATRA [9] is a single FIFO queue based multicast algorithm, where each input port has a single shared (physical) queue for both unicast and multicast traffic. The central scheduler maintains N virtual queues and each is destined for one output. In each time slot, the head-of-line (HOL) packet of each input queue is scheduled to join different virtual queues according to its destination output ports. Fanout splitting [13], which allows a multicast packet to be sent to a subset of its outputs, is adopted to increase switch throughput. Nevertheless, TATRA suffers from the severe HOL blocking due to its single queue nature.

To reduce HOL blocking, multiple dedicated multicast queues are used in [10] and [11]. In [10], each input port maintains a set of multicast queues. When a multicast packet arrives, it selects one of the multicast queues to join according to its load-balancing policy. In each time slot, scheduling priority is given to either a unicast packet or a multicast packet according to a certain service ratio between the two types of traffic. An iterative scheduling algorithm is also adopted to maximize the switch throughput.

To further reduce the HOL blocking, a multicast packet *split* scheme is proposed in [11]. In [11], the set of output ports is divided into m non-overlapped subsets, and each input port maintains m unicast/multicast shared queues and each is dedicated to a subset of outputs. When a multicast packet arrives, if its fanout set completely fit into a queue, it joins that queue; otherwise, the multicast packet is split into "smaller" ones (each with a modified fanout set) to join multiple queues. Again, an iterative scheduler is adopted to maximize throughput.

In [12], an efficient multicast scheduling algorithm called FIFOMS is proposed to avoid the HOL blocking. The basic idea is to separately store unicast/multicast packets and their memory addresses (i.e. pointers). FIFOMS uses the classic unicast VOQs (virtual output queues) as pointer queues. Specifically, when a multicast packet with a fanout size of f ($f = 1$ for unicast packet) arrives, it is time-stamped and stored in a shared memory, and its memory address/pointer joins f different VOQs according to the fanout set. In each time slot, scheduling priority is given to the pointers (which respesent the unicast copies of a multicast packet) with the smallest timestamp. In effect, every multicast packet is "converted" into unicast in scheduling. To this end, HOL blocking is completely eliminated. But in order to maximize switch throughput, in-switch packet replication is still used for sending multiple copies of a multicast packet in the same slot. This is achieved by an iterative scheduling algorithm, which incurs considerable amount of communication overheads.

Notably, the feedback-based two-stage switch has been extended in [14] to support multicast traffic. In the feedback-based unicast switch (see Section III), each input port maintains N unicast VOQs . To support multicast, m shared queues for multicast traffic are added at each input port and the same packet split scheme used in [11] is adopted. Priority is given to schedule multicast traffic. When a multicast packet is selected, only one copy of this multicast packet together with an N-bit *replication vector* is sent to the middle-stage port, where packet replication occurs based on the replication vector. Unlike other existing multicast switches, this algorithm is simple because it requires neither a central scheduler nor multicast switch fabric. However, it still suffers from the HOL blocking.

III. Feedback-based Two-stage Switch for Unicast Traffic

Our multicast scheduling algorithm is based on the feedback-based two-stage switch [8]. In this section, we review its basic unicast operation. Consider the feedback-based switch in Fig. 2, where VOQs are located at both input and middle-stage ports. We use $\text{VOQ}_1(i,k)$ to represent the VOQ at input port i with packets destined for output k (not middle-stage port k). Similarly, $\text{VOQ}_2(j,k)$ is used to denote the VOQ at middle-stage port j with packets destined for output k.

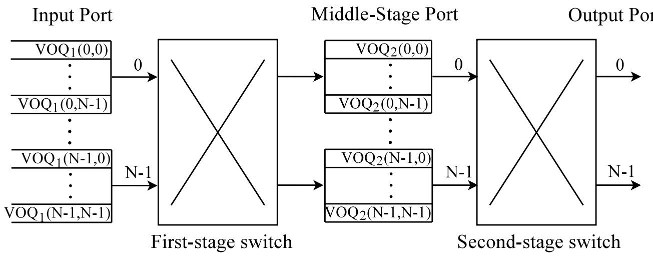

Fig. 2. A feedback-based two-stage switch.

Each stage of fabric is configured following a deterministic and periodic sequence of N configurations. An example of constructing such a sequence is that at time slot t, input i (for $i = 0, 1, 2, \ldots, N-1$) is connected to output j, where j is given by

$$j = (i + t) \ mod \ N. \quad (1)$$

Assume the sequence of N configurations used in the first-stage switch is obtained from Eqn. (1). Let c_t denote the configuration in slot t. Then $[c_0, c_1, \ldots, c_{N-1}]$ denotes the resulting sequence of N configurations. The same set of N configurations is used in the second-stage switch, but in a different order/sequence for providing the necessary feedback path. Specifically, a sequence in the reverse order of that in the first stage, or $[c_{N-1}, c_{N-2}, \ldots, c_0]$, is used, such that at time t (for $0 \leq t < N$), middle-stage port j is connected to output k, where k is given by

$$k = (j + N - 1 - t) \ mod \ N \quad (2)$$

If t is within $[xN, (x+1)N)$, set $t = t - xN$ before applying (2).

Combining the two sequences of configurations, the two-stage switch is configured according to the joint sequence of $[c_0 \ c_{N-1}, c_1 \ c_{N-2}, \ldots, c_{N-1} \ c_0]$. A joint sequence example is shown in Fig. 3, where the solid lines show the configurations used by the first fabric and the dashed lines show the configurations used by the second fabric.

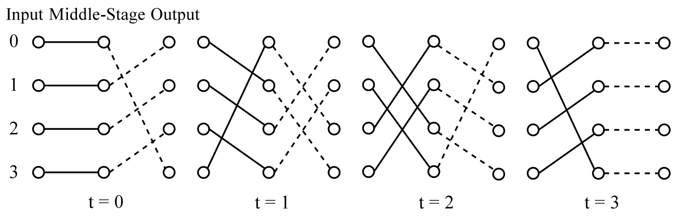

Fig. 3. Two sequences of configurations for a 4×4 feedback-based switch.

Each VOQ at the middle-stage port in Fig. 2 can store at most one packet. At time slot t, the occupancy of the N VOQs at a middle-stage port j is reported to its connected output k by piggybacking the N-bit occupancy vector onto the data packet. Since output k and input k are located on the same switch linecard, the occupancy vector is then made available to input k at negligible cost. At slot $t+1$, based on the occupancy vector received, input k selects a packet for sending to its connected middle-stage port j. Without loss of generality, we assume input k selects the packet from its longest VOQ, yet ensuring the corresponding VOQ at middle-stage port j is empty (as learned from the occupancy vector).

Unlike other load-balanced switches [5]–[7]. the feedback-based switch guarantees in-order packet delivery because packets belonging to the same flow, though traversing through different middle-stage ports, always experience the same middle-stage port delay. For more details, please refer to [8].

IV. FTMS: Feedback-based Two-stage Multicast Scheduling Algorithm

A simple distributed multicast scheduling algorithm based on the feedback-based switch is proposed in this section. We call it Feedback-based Two-stage Multicast Scheduling algorithm (FTMS). In order to eliminate the HOL blocking at input ports, the pointer-based multicast VOQ [12] is adopted (see the review in Section II). Besides, packet replication occurs at middle-stage ports and in-switch multicast capability is thus not required.

A. Pointer-based multicast VOQ

For an $N \times N$ input-queued unicast switch, each input port maintains N VOQs, one for each output/flow. The HOL blocking can then be eliminated. But for multicast traffic, each input needs to handle up to $2^N - 1$ possible multicast flows. In practice, this is not feasible. In [12], a pointer-based multicast VOQ is introduced to solve this problem.

A pointer-based multicast VOQ consists of two buffers, packet buffer and pointer buffer. When a multicast packet with a fanout size of f arrives, it is stored (once) in the packet buffer, and its memory address (i.e. a pointer) is copied to f different pointer VOQs according to its fanout set. Since the destinations of a (multicast) packet are independent and a pointer only consists of a few bytes, the copy operation can be done in parallel by hardware with no/slight speedup. Fig. 4 shows a multicast VOQ for a 4×4 switch. Specifically, Packet1 with fanout set {1, 2, 3} and Packet2 with fanout set {0, 2, 3} arrive at the input port in time slots 1 and 2 respectively. While packets are stored in the packet buffer, their pointers, P1 and P2, join the corresponding pointer VOQs.

B. Multicast scheduling

Without loss of generality, we assume the two stages of switch fabrics in Fig. 2 are configured following the joint sequence of switch configurations given by Eqns. (1) and (2). Besides, the unicast VOQ at each input port is replaced by the multicast VOQ structure in Fig. 4.

Our Feedback-based Two-stage Mulitcast Scheduling (FTMS) algorithm consists of the following three steps:

Pointer selection: In each time slot, based on the received occupancy vector of middle-stage port j, input port i selects a HOL pointer from the N pointer VOQs. Priority is given to the longest VOQ. If a pointer P is selected, then check the other $N-1$ HOL pointers. Any pointers that satisfy the following requirements are also selected: (1) the corresponding VOQ at middle-stage port j is empty (as learned from the occupancy

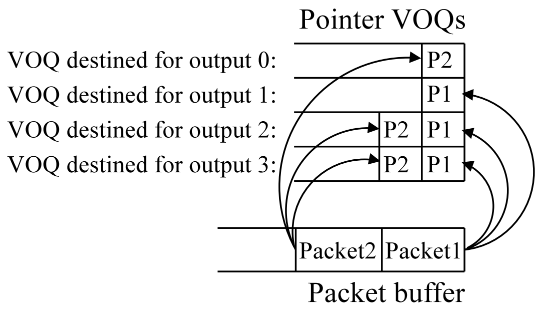
Pointer VOQs
VOQ destined for output 0:
VOQ destined for output 1:
VOQ destined for output 2:
VOQ destined for output 3:
P2
P1
P2
P1
P2
P1
Packet2
Packet1
Packet buffer

Fig. 4. Pointer-based multicast VOQ for a 4×4 switch.

vector); and (2) the pointer must point to the same (multicast) packet as pointer P does. Therefore, multiple HOL pointers can be selected in this step.

Generating replication vector: Based on the selected pointers, an N-bit *replication vector* is generated, which identifies the (middle-stage port) VOQs these pointers belong to.

Packet transmission: Since all selected pointers point to the same (multicast) packet, only one [1] copy of this packet is sent to middle-stage port j, together with its N-bit replication vector. Then these selected pointers are removed from their pointer VOQs. Note that a packet will be deleted from the packet buffer, if all of its pointers are removed from the VOQs. When a packet arrives at the middle-stage port, based on the piggybacked replication vector, it will be copied to join the corresponding empty (unicast) VOQs.

Note that the (unicast) packet transmission in the second stage switch is the same as the original feedback-based two-stage switch. When a middle-stage port is connected to an output (according to Eqn. (2)), a packet from the corresponding middle-stage VOQ is sent. Since packet replication is carried out at middle-stage port, an interesting question is if a speedup is required at a middle-stage port for packet replication? The answer is no because the replication process can span over multiple time slots because each time slot a middle-stage port can send at most one packet to an output.

Finally, let us wrap up our FTMS algorithm by an example in Fig. 4. Assume in the current time slot, the input port shown in Fig. 4, say input port i, receives an occupancy vector (1,0,1,1) of middle-stage port j, where "1" indicates the corresponding VOQs at middle-stage port j are empty. Based on the occupancy vector, assume the HOL pointer P1 of VOQ$_1(i,3)$ (the VOQ destined for output 3) is selected using the LQF scheme. According to the selection rules, pointer P1 of VOQ$_1(i,2)$ should also be selected. Then a copy of Packet1 is sent to middle-stage port j together with a replication vector (0,0,1,1) ("1" identifies the middle-stage VOQ which the packet should be replicated to), and the two HOL pointers of VOQ$_1(i,2)$ and VOQ$_1(i,3)$ are removed. When the copy of Packet1 arrives at middle-stage port j, it is cloned to both VOQ$_2(j,2)$ and VOQ$_2(j,3)$, and the cloning process can span over multiple time slots.

C. Discussion

In our FTMS algorithm, packet replication only occurs at the middle-stage port. When a multicast packet arrives at an input port, it is stored as a unicast packet and its pointer joins different VOQs. (Note that a stored packet can be read out multiple times. But we do not consider such an implicit fanout splitting as packet replication.) In the first stage switch fabric, a (copy of) multicast packet is sent to a middle-stage port as a unicast packet. In the second stage, "real" unicast packets (which are cloned at middle-stage ports) are delivered to the correct destination ports. Therefore, the two switch fabrics in Fig. 2 are unicast. Besides, such a unicast switch fabric only needs to realize N switch configurations, whereas a regular unicast switch fabric needs to realize $N!$ configurations.

Our FTMS and the multicast scheduling algorithm proposed in [14] (we call it load-balanced multicast scheduler (LBMS) in this paper) are based on the same feedback-based two-stage switch. In LBMS, packet replication takes place at both input ports and middle-stage ports, and m ($1 \leq m \leq N$, N is the switch size) dedicated multicast VOQs are added to each input port. As m increases, the HOL blocking is reduced, and more packets are replicated at the input port. Two major drawbacks of packet replication at input ports are additional speedup requirement for packet replication, and additional queuing delay due to the large number of replicated packets. When $m = N$, LBMS completely eliminates the HOL blocking problem, but all packet replication is done at the input port and it suffers the highest input port queuing delay (as can be seen from simulation results later). In other words, LBMS improves its throughput performance at the cost of increased delay.

Our FTMS avoids HOL blocking by adopting the pointer-based multicast VOQ structure in Fig. 2, and only allows packet replication to take place at the middle-stage port. Note that feedback-based two-stage switch uses a single-packet buffer at each middle-stage VOQ, so packet replication at the middle-stage port will not increase queueing delay. Again, this can be confirmed by the simulation results in the next section.

Further notice that pointers destined for a particular output port are stored in a FIFO queue, so the multicast VOQ structure does not cause packet out-of-order. The multicast VOQ structure can also be applied to the middle-stage port. We can just buffer a multicast packet once, and store the pointer to its buffer memory location at the corresponding (pointer) VOQs. To send a packet, it is retrieved based on the pointer value in the VOQ. Since at most one (unicast) packet is sent from each middle-stage port (to the second stage switch) in each time slot, no speedup is required.

V. Simulation Results

The delay-throughput performance of our proposed FTMS is studied by simulations in this section. Two representative centralized scheduling algorithms are implemented for comparison, TATRA [9] and FIFOMS [12]. We also compare FTMS with the multicast scheduling algorithm proposed in [14], or LBMS. Note that both TATRA and FIFMOS require multicast switch fabric, whereas LBMS and our FTMS only use two stages of simple unicast switch fabrics. In addition to hardware complexity, FTMS and LBMS do not require a central scheduler and thus are suitable for distributed implementation.

FIFOMS with single iteration and two iterations are denoted by FIFOMS-1 and FIFOMS-2 respectively. LBMS with m dedicated multicast queues at each input port is denoted by LBMS-m (m = 2 and 32 in the simulations). For brevity, the simulation results presented below are based on a 32×32 switch. Three major traffic patterns are considered: uniform

[1]Accordingly, no speedup is required because at most one packet is sent from each input port in each time slot.

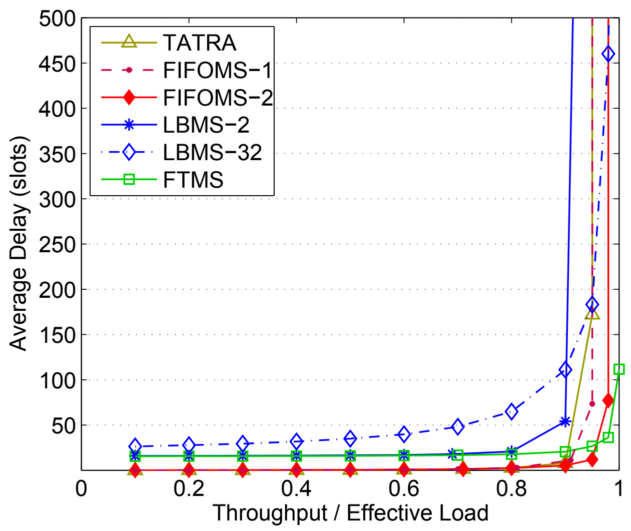

Fig. 5. Uniform Bernoulli mixing traffic.

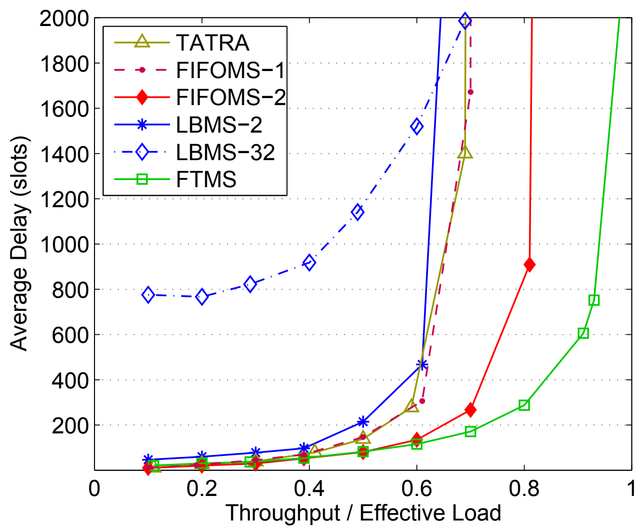

Fig. 6. Uniform bursty mixing traffic.

Bernoulli, uniform bursty and binomial. We assume that the input port buffer size is large enough for avoiding buffer overflow.

A. Uniform Bernoulli mixing traffic

Uniform traffic is generated as follows. At every time slot for each input, a packet arrives with probability λ (input load). We consider mixed unicast and multicast traffic in the simulation and each packet has equal probability of being unicast or multicast. If the packet is unicast, it destines to each output with equal probability; if the packet is multicast, its fanout set consists of f outputs randomly chosen from all output ports, where f is the fanout size randomly selected between [2, 32]. Then the effective load p is given by

$$p = \lambda[0.5 + 0.5(2 + 32)/2] \tag{3}$$

When $p \leq 1$ (or $\lambda \leq 1/9$), no input or output will be overloaded.

Fig. 5 shows the switch delay performance against the effective load, or throughput. We can see that both FIFOMS and TATRA yield good delay-throughput performance when the traffic load is not high. This is because under uniform traffic patten, the output contention is greatly reduced using fanout splitting, and thus FIFOMS and TARTA can take full advantage of in-switch packet replication. FIFOMS-2 (FIFOMS with two iterations) outperforms FIFOMS-1 and TATRA. On the other hand, LBMS-32 (LBMS with 32 dedicated multicast queues) yields higher throughput than LBMS-2 because the HOL blocking is eliminated. But the delay performance of LBMS-32 is poorer than LBMS-2 when load is not high. This supports our analysis in Section IV that packet replication at the input port increases queueing delay at input ports. Finally, our FTMS experiences a constant delay of about 20 slots for low to medium load, which is generally smaller than LBMS-2 and LBMS-32. This is because packet replication only occurs at the middle-stage port in FTMS. Although FTMS has higher delay than TATRA and FIFOMS for low to medium load, its implementation complexity is the lowest (as noted before) among the three algorithms. Besides, when the traffic load is high, FTMS provides the best and close-to-100% throughput.

B. Uniform bursty mixing traffic

We use the same traffic generator except the bursty arrivals are based on the ON/OFF model. Specifically, in the ON state, a packet arrival is generated in every time slot; in the OFF state, no packet arrives. Packets of the same burst have the same fanout set. Given the average input load λ and average burst size q, the state transition probabilities from OFF to ON is $\lambda/[q(1-\lambda)]$ and from ON to OFF is $1/q$. Without loss of generality, we set burst size $q = 32$.

From Fig. 6 we can see that under bursty traffic, delay increases quickly with throughput. Nevertheless, our FTMS still achieves the highest throughput. The gap between FTMS and other schedulers becomes wider as the traffic load increases. This is because FTMS inherits the load balancing mechanism from its unicast counterpart, and is able to load balance the bursty multicast traffic. Notably, TATRA and FIFOMS are overwhelmed by the bursty arrivals even with in-switch packet replication capability. TATRA and FIFOMS-1 have the similar delay-throughput performance and FIFOMS-2 outperforms FIFOMS-1 because it can allow more packets to be sent in each time slot. Again, LBMS-32 yields higher throughput than LBMS-2, but it suffers from longer delay.

C. Binomial mixing traffic

Binomial mixing traffic [15] is the same as the uniform Bernoulli mixing traffic model except in generating the fanout size of a multicast packet. Let P_f be the probability of generating a fan-out set with size f. The f destinations are uniformly distributed over all output ports. The value of f is chosen according to a non-uniform binomial distribution:

$$P_f = C_N^f(\frac{F}{N})^f(1-\frac{F}{N})^{N-f}$$

where F is the mean fanout size. In our simulations, we set $F = 17$, which is equivalent to the average fanout size in the uniform Bernoulli mixing traffic. Then the effective load is still given by Eqn. (3). Because of the similarity between the binomial mixing traffic and the uniform Bernoulli mixing traffic, Fig. 7 shows a similar trend as that in Fig. 5.

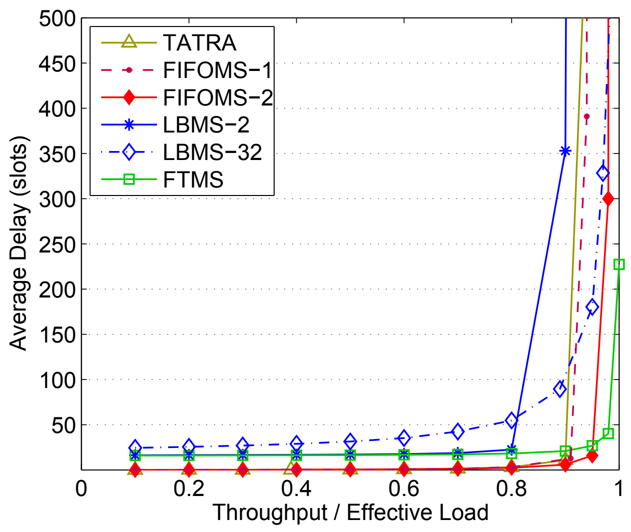

Fig. 7. Binomial mixing traffic.

VI. Conclusion

To address the two major challenges in designing high-speed multicast switches, namely, the expensive multicast switch fabric and the highly complicated central scheduler, we proposed FTMS, an efficient multicast scheduling algorithm based on the feedback-based two-stage switch architecture. The feedback-based two-stage switch is selected because it does not require a central scheduler, its switch fabric is unicast and very simple, and it elegantly solves the packet mis-sequencing problem faced by other load-balanced switches. With FTMS, head-of-line (HOL) packet blocking at input port is eliminated by adopting "pointer" queues. To cut down queuing delay, packet replication is carried out at middle-stage ports. As compared with other multicast scheduling algorithms, simulation results showed that our FTMS always provides the highest throughput.

References

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

[2] H. J. Chao and B. Liu, *High Performance Switches and Routers*. John Wiley & Sons, Inc, 2007, ch. 7, pp. 225–231.

[3] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proc. IEEE Fifteenth Annual Joint Conf. of the IEEE Computer Societies. Networking the Next Generation INFOCOM '96*, vol. 1, 1996, pp. 296–302.

[4] M. Andrews, S. Khanna, and K. Kumaran, "Integrated scheduling of unicast and multicast traffic in an input-queued switch," in *Proc. IEEE Eighteenth Annual Joint Conf. of the IEEE Computer and Communications Societies INFOCOM '99*, vol. 3, 1999, pp. 1144–1151.

[5] 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, no. 6, pp. 611–622, 2002.

[6] Y. Shen, S. Jiang, S. S. Panwar, and H. J. Chao, "Byte-focal: a practical load balanced switch," in *Proc. HPSR High Performance Switching and Routing 2005 Workshop*, 2005, pp. 6–12.

[7] J. J. Jaramillo, F. Milan, and R. Srikant, "Padded frames: A novel algorithm for stable scheduling in load-balanced switches," *IEEE/ACM Transactions on Networking*, vol. 16, no. 5, pp. 1212–1225, 2008.

[8] B. Hu and K. L. Yeung, "Feedback-based scheduling for load-balanced two-stage switches," *IEEE/ACM Transactions on Networking*, vol. 18, no. 4, pp. 1077–1090, 2010.

[9] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast scheduling for input-queued switches," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 5, pp. 855–866, 1997.

[10] W. Zhu and M. Song, "Integration of unicast and multicast scheduling in input-queued packet switches," *Computer Networks*, vol. 50, pp. 667–687, April 2006.

[11] S. Gupta and A. Aziz, "Multicast scheduling for switches with multiple input-queues," in *Proc. 10th Symp. High Performance Interconnects*, 2002, pp. 28–33.

[12] D. Pan and Y. Yang, "FIFO-based multicast scheduling algorithm for virtual output queued packet switches," *IEEE Transactions on Computers*, vol. 54, no. 10, pp. 1283–1297, 2005.

[13] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Multicast traffic in input-queued switches: optimal scheduling and maximum throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 465–477, 2003.

[14] B. Hu and K. L. Yeung, "Multicast scheduling in feedback-based two-stage switch," in *Proc. Int. Conf. High Performance Switching and Routing HPSR 2009*, 2009, pp. 1–6.

[15] A. Bianco, P. Giaccone, C. Piglione, and S. Sessa, "Practical algorithms for multicast support in input queued switches," in *Proc. Workshop High Performance Switching and Routing*, 2006.