A Novel Algorithm for Maintaining Packet Order in Two-Stage Switches

Xiao Ning Zhang, Du Xu, and Le Min Li

ABSTRACT—To enhance the scalability of high performance packet switches, a two-stage load-balanced switch has recently been introduced, in which each stage uses a deterministic sequence of configurations. The switch is simple to make scalable and has been proven to provide 100% throughput. However, the load-balanced switch may mis-sequence the packets. In this paper, we propose an algorithm called full frame stuff (FFS), which maintains packet order in the two-stage load-balanced switch and has excellent switching performance. This algorithm is distributed and each port can operate independently.

Keywords—Two-stage load-balanced switch, scalability, throughput, mean delay.

I. Introduction

Recently, a novel switching structure called a load-balanced Birkhoff-von Neumann switch was proposed by C. S. Chang and others. [1]. The switch eliminates a scheduler and is simple to make scalable.

In the load-balanced switch, because of the load-balancing of the first stage, the lengths of virtual output queues (VOQs) in the intermediate inputs are not identical, and the packets through the switch can be mis-sequenced. Although strictly not disallowed in an Internet router, mis-sequencing causes problems for some protocols, for example., TCP does not perform well when out-of-order packets arrive at their destination. Out-of-order packets can be perceived as loss indicators and trigger an unnecessary reconnection and retransmission. There are two methods to prevent the packets from being out-of-order in the load-balanced switch. The first method permits the packets to be out-of-order in the switch but reorders the packets using a finite reordering buffer at the output. The second method is to make sure that packets arrive in order to the outputs, thus keeping packets in order throughout the switch. In [2], the author introduces an algorithm called uniform frame spreading (UFS). UFS is based on the second method and a distributed algorithm; however, UFS requires that packets be transferred only when there is a full frame in the queue (we call N packets from the same flow "a full frame"). This greatly increases the waiting time of the packets at each input. In [3], the authors propose an algorithm called full ordered full frame (FOFF). FOFF is based on the first method and a distributed algorithm. As in UFS, each input keeps a separate FIFO queue for each output. When a queue contains at least one full frame, FOFF behaves as UFS. However, FOFF allows for a non-empty queue to send packets when no queue has any full frame, which causes packets to be out-of-order. So in FOFF, each output has N FIFO queues corresponding to the N intermediate inputs to reorder the packets. FOFF can avoid the starvation in UFS. FOFF also has its own disadvantage. FOFF requires sending packets from the same flow uniformly to the intermediate inputs, so the output can reorder the arriving packets based on its intermediate input. Assume that a given queue is the last served as a no-full-frame queue and that the pointer keeps track of the last intermediate input to which the packet is transferred. When the queue is served again, the packet is only transferred starting from the time connected with the next intermediate input, which prevents the inputs from fully utilizing the N timeslots. This increases the mean delay of the switch.

In this paper, we propose a novel algorithm called full frame stuff (FFS). FFS orders the packets through the switch. FFS is a

Manuscript received Nov. 05, 2004; revised June 06, 2005.

This work is supported by National Natural Science Foundation of China (NSFC) under grant no. 60372011.

Xiao Ning Zhang (phone: +86 28 83202468, email: xiaoning.z@163.com), Du Xu (email: xudu@uestc.edu.cn), and Le Min Li (email: lml@uestc.edu.cn) are with Key Lab of Broadband Optical Fiber Transmission and Communication Networks, the University of Electronic Science and Technology of China, Chengdu, China.

distributed algorithm with good switching performance.

II. Switch Architecture

Throughout the paper, we'll assume the number of switch ports by N (N>2); the sequence is one in which input i is connected to output ((t+i) modulo N) at timeslot t in the first stage, and input j is connected to output ((t-j) modulo N) at timeslot t in the second stage; a flow is the set of all packets with the same input and output destination; and consecutive N packets from the same flow is called a full frame.

At each input in the switch architecture with FFS, there are N VOQs, one-per output. For example, an arriving packet destined to j is placed in Q_j($1 \leq j \leq N$), and there are a total of N^2 VOQs at N inputs.

At each intermediate input, three-dimensional queues are employed. There are N^2 VOQs at each intermediate input, and there are a total of N^3 VOQs at N intermediate inputs. We'll use a three-dimensional coordinate (i,j,k) ($1 \leq i, j, k \leq N$) to denote a VOQ in the intermediate inputs, and a packet whose input is j and output destination is k arriving at intermediate i will be placed in VOQ (i,j,k).

FFS makes sure the packets through the switch are orderly; when packets arrive at the output, it can leave the switch directly without output buffering, so there are no reordering queues at the outputs.

III. Full Frame Stuff

The basic idea of FFS is to allow a non-full frame to be sent after being stuffed with "idle packets." In FFS, each input sends one full frame in every continuous N timeslot as in UFS, but starvation in UFS can be avoided. In what follows, we'll define how the FFS algorithm works at each input, intermediate input, and output.

1. Input Implementation

At each input, an arriving packet destined to output j is placed in Q_j, and adds two fields of "flow packets" and "total packets" to the packet. The "flow packets" field indicates whether the packet is an "idle packet" created based on the FFS algorithm or a "non-idle packet" from input ports of the switch, while the "total packets" field indicates the number of "non-idle packets" in one stuffed full frame. Here, we assume for every mN (m is an integer parameter, m is a integer parameter $\geq$1) timeslots, the input selects the m longest queues from N queues to serve for the next mN timeslots (because each input only sends one stuffed full frame every N timeslots, and for every mN timeslots each input only sends m full frames). The reason for choosing the m longest queues is to decrease the number of idle packets in the stuffed full frame and fully utilize timeslots to send "non-idle packets." The m queues are arranged in order of length as $\{Q_1,Q_2,\ldots,Q_m\}$, and the corresponding length of the queues is recorded as $\{L_1,L_2,\ldots,L_m\}$. In the first N timeslots, the input sends packets from Q_1. If L_1 equals zero, which means Q_1 is an empty queue, then the input doesn't send packets in the N timeslots. If Q_1 is a non-empty queue, in every timeslot the packet of the head-of-line is taken out from Q_1. If Q_1 is not empty, the packet's field "flow packet" is set as "1" and "total packets" is set as "L_1." If Q_1 is now empty, then an "idle packet" to be stuffed is created, whose input and output destination are set on the current situation, and the "flow packet" is set as "0" and "total packets" is set as "L_1." The packets (including stuff packets) are sent from the input to the currently connected intermediate input through the first stage. In N timeslots, a full frame leaves the input in order, and the j-th packet of the full frame is transferred to intermediate input j. In this way, the second N timeslots is for Q_2, the third N timeslots is for Q_3 … and the m-th N timeslots is for Q_m, until the input selects the m longest queues again.

In FFS, the input orderly transfers a full frame to the intermediate inputs, just like in UFS. And if there is not a full frame at the input, UFS must wait for the packets while FFS forms a full frame by producing the stuff packets. From this, m is an important parameter in FFS. In section IV we'll determine the value of m from the computer simulations.

2. Intermediate Input Implementation

FFS uses three-dimensional queues in the intermediate inputs. Assume that a packet arrives at intermediate input i through the first stage, and its input is j and output destination is k, then the packet will be placed in VOQ (i, j, k) at intermediate input i.

When the timeslot is t, assume that intermediate input i is connected to output k through the second stage. Intermediate input i detects the contents of all VOQs (i, j, k) (j=1,2,…,N) destined to the output k in the order of j. Below are the steps for the examination of the queues. First we initialize three variables: j=1; *length*=0; *queue_for_send*=N+1.

Step 1: Detect the head-of-line packet of VOQ (i, j, k). If the queue is empty, turn to step 3. Otherwise, go to step 2.

Step 2: Access the head-of-line packet from VOQ (i,j,k) and read the value of its field's "total packet." If the value is more than the value of *length*, the value is assigned to *length* and j is assigned to *queue_for_send*. Then, go to step 3.

Step 3: j=j+1. If j is more than N, go to step 4; otherwise, return to step 1.

Step 4: Examine the value of *queue_for_send*. If it equals N+1, it means these queues are all empty and no packet is transferred. If it is less than N+1, intermediate input i transfers the head-of-line packet from VOQ (i, *queue_for_send*, k) to output k.

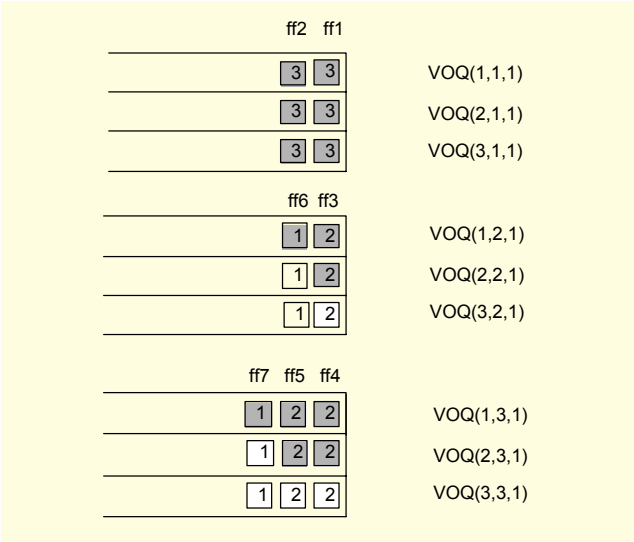

Fig. 1. FFS algorithm in intermediate inputs for output 1.

Because all the inputs transfer full frames to the intermediate inputs, the lengths of VOQs in the intermediate inputs are identical. When all intermediate inputs are orderly connected to a given output, by examination in step 4, the packets (if there are no-empty-queues) transferred are all from the same full frame. As the full frame is produced by stuffing packets, we select the queue whose head-of-line packet's "total packet" is the most, which means there are the least number of stuffing "idle packets" in the full frame. So the intermediate inputs can utilize the timeslots fully to transfer the useful packets to the outputs and assure the throughput of the switch. Figure 1 illustrates the FFS algorithm in intermediate inputs in a 3-port load-balanced switch. In Fig. 1, the three-dimensional queues are rearranged so that all the queues in intermediate inputs containing packets from the same input are adjacent to each other. In practice, of course, the queues are not arranged like this, but they have been redrawn to help explain the algorithm. The shadowed packet indicates a "non-idle packet" and the transparent packet indicates an "idle packet." The number in each packet represents the value of the field's "total packets." The numbers above the full frames indicate the order in which they will be served. Below are the processes of FFS in intermediate inputs. In the example, FFS serves the first full frame (ff1) from input 1 because the value of its "total packets" is the largest in all head-of-line frames to output 1. FFS serves it over three consecutive timeslots, delivering the three packets to output 1. FFS then serves ff2 from input 1 in the next three consecutive timeslots. Currently, there is no full frame from input 1 in intermediate inputs. The third served full frame is ff3 from input 2 (ff4 has the same value of "total packets" with that of ff3; according to our definition, FFS chooses ff3). FFS serves ff4 and ff5 from input 3. Then ff6 is served (ff6 has the same value of "total packets" with that of ff7; according to our definition, FFS chooses ff6). No full frame from input 2 exits in the intermediate inputs. At last, the served full frame is ff7 from input 3.

3. Output Implementation

When the packet arrives at the output through the second stage, the two fields of "flow packet" and "total packet" are removed from it. The value of the "flow packet" is examined. If it equals 1, the packet is a "non-idle packet". Then, the packet is sent out of the switch; if it equals 0, the packet is an "idle packet" and FFS then destroys the packet immediately.

4. Properties of FFS

FFS has the following properties:

- FFS is a distributed algorithm. The inputs, intermediate inputs, and outputs can work independently. It needs no additional communication of information among them.
- In order to know the amount of congestion in the switch (for example to implement a drop-policy), we need to know how many packets are in each VOQ of the intermediate inputs. But because each VOQ has the same occupancy (one packet from each full frame), it is sufficient to look at just one intermediate input.
- FFS can avoid pathological traffic patterns. It does so by spreading each flow evenly across the intermediate inputs. FFS guarantees that the cumulative number of packets sent to each intermediate input for a given flow is the same. This even spreading prevents a traffic pattern from concentrating packets to any individual intermediate inputs.

IV. Simulation Results and Analysis

In the simulation, we assume that in every timeslot the input and intermediate input can only transfer one packet, and the physical delay of the switch stage is zero. This means a packet takes a timeslot through the switch stage. The packet arrival process is a Bernoulli process and its destinations are uniformly distributed.

First, we simulate an 8-port load-balanced switch using the FFS algorithm with parameter m. Figure 2 shows the throughput–traffic load curves, and Fig. 3 shows the mean delay-traffic load curves.

Figure 2 shows that in FFS, when m equals any value of (1,…, 8), a 100% throughput can be assured. In fact, when the mean rate matrix of input traffic satisfies "no overbooking conditions," the switch can provide 100% throughput [1]. This means that for uniform Bernoulli i.i.d. traffic, a 100% throughput can be

guaranteed when at most one packet arrives at an input per timeslot. Stuffing packets into full frame satisfies the requirement, so FFS can provide 100% throughput with any value of m.

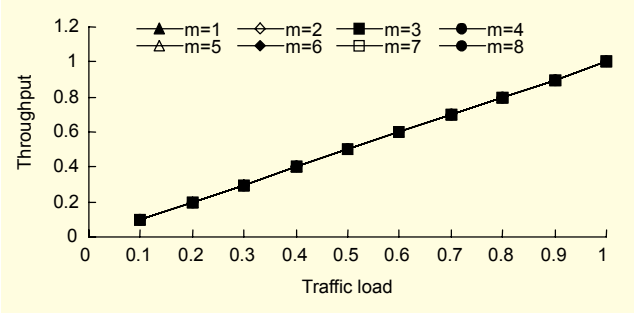

Fig. 2. Throughput-traffic load curves in FFS.

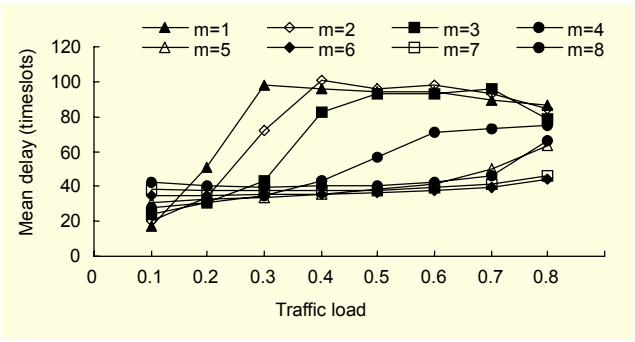

Fig. 3. Mean delay-traffic load curves in FFS.

Figure 3 shows that when m is less than 6, the mean delay decreases with the increase of m; when m is more than 6, the mean delay increases with the increase of m. In every mN timeslots, the input selects the m longest queues from N queues to serve for the next mN timeslots. Assume that at i-th N timeslots ($1 \le i \le m$) the N inputs serve queues $\{Q_{1i}, Q_{2i}, \ldots, Q_{Ni}\}$. Although the two-stage switch is non-blocking and collision-free, in the intermediate input there exists output-competition from VOQs corresponding to the same output. When m increases, the interval of the detecting time increases and the number of the same queues in $\{Q_{1i}, Q_{2i}, \ldots, Q_{Ni}\}$ will decrease. So when the intermediate inputs transfer packets to the outputs, the probability of the appearance of a collision will decrease and the switching performance can be improved. But when m is too large, the interval of the detecting time also becomes large. The last several queues to be served may not be the relatively longer queues. Because the inputs produce full frames by stuffing packets, this will greatly increase the mean delay of the switch, especially in a high traffic load. In the simulation to the 8-port load-balanced switch using FFS, we discovered when m equals 6 the switch has good switching performance. When the number of the switch port equals 16 or 24, we discovered that when m equals 14 or 22 the switching performance is satisfactory. So we suggest m equals N-2 for the N-port load-balanced switch in this paper.

In the above, we determine the value of m in FFS. We simulate the 8-port load-balanced switch using FFS (m=6), UFS, and FOFF. Figure 6 shows their mean delay-traffic load curves.

Figure 4 shows that the mean delay of FFS is apparently less than that of UFS, especially in a low traffic load. The reason for this is that it needs more time to form a full frame in a low traffic load in UFS, and the mean delay decreases with the increase of traffic load. When the traffic load is less than 0.2, the mean delay of FFS is larger than that of FOFF. This is because FFS needs to stuff more "idle packets" in full frame with low traffic load and can not sufficiently utilize the bandwidth of the switch. But when traffic load increases, the mean delay of FFS is less than that of FOFF. The reason is that in FOFF the input requires transferring packets uniformly to the intermediate inputs and can not efficiently use the N timeslots to send packets.

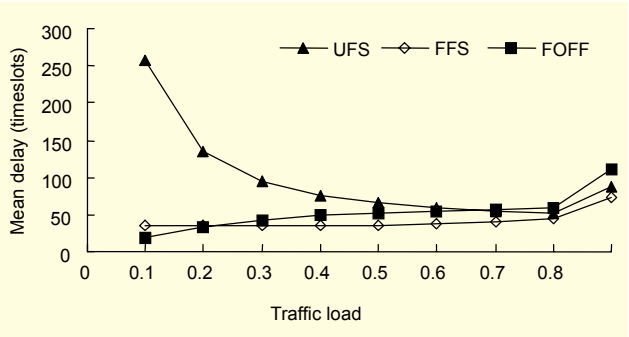

Fig. 4. Mean delay-traffic load curves in FFS, UFS, and FOFF.

V. Conclusion

We propose a novel algorithm called full framed stuff (FFS), which can keep the packets in order throughout the load-balanced switch. FFS can provide good switching performance (as throughput and delay). FFS is also a distributed algorithm, and each port can operate independently.

References

[1] C.S. Chang, D.S. Lee, and C.M. Lien, "Load Balanced Birkhoff - von Neumann Switches, Part I: One-Stage Buffering," *Computer Comm.*, Vol. 25, 2002, pp. 611-622.

[2] I. Keslassy, Ph.D. dissertation, http://comnet.technion.ac.il/~isaac/p/thesis.pdf.

[3] I. Keslassy, S.-T. Chuang, K. Yu, et al, "Scaling Internet Routers Using Optics" *ACM SIGCOMM '03*, Karlsruhe, Germany, August 2003.