Problem of Packet Mis-Sequencing in Switches

Siddhartha Saha
ECE 284 Project Report, Fall 2003

---

1. Introduction:

With the current growth in Internet traffic, and the increased use of fiber optics technology, electronic routers in the Internet may very well the bottleneck in future. This observation is justified by the fact that till date the growth rate of the network in terms of speed is much more than the rate at which the capacity of routers are growing. This situation have motivated many researchers to focus on the ideas to come up with new and scalable routing architectures which can operate at Giga or Peta bit speed and provide close to 100% throughput guarantees with low latency.

More and more researchers are coming up with ideas to use optics inside routers. A recent paper by Chang et al. [1], which uses cyclic shift inside a router without any decision-making architecture, is a promising architecture in view of implementation using optics. This switch has provably 100% throughput with constant computation requirement. But one of the primary problem with this switch and many more switches that has similar kind of operating structure is mis-sequencing of packets.

A mis-sequencing problem arises when the packets leave the switch in an order different to the order in which the packets arrive at the input port of the switch. As an example, mis-sequencing occurs of two packets, p_1 and p_2, both the packets are destined to go to the same output port, arrive at a particular input port; p_1 arriving before p_2 and p_2 leaves the switch before p_1 at the output port.

In this report, we will explore some of the switch architectures that have faced the problem of out of sequence packets at the output port, and we will discuss the methods and schemes undertaken by the corresponding architecture to deal with this problem.

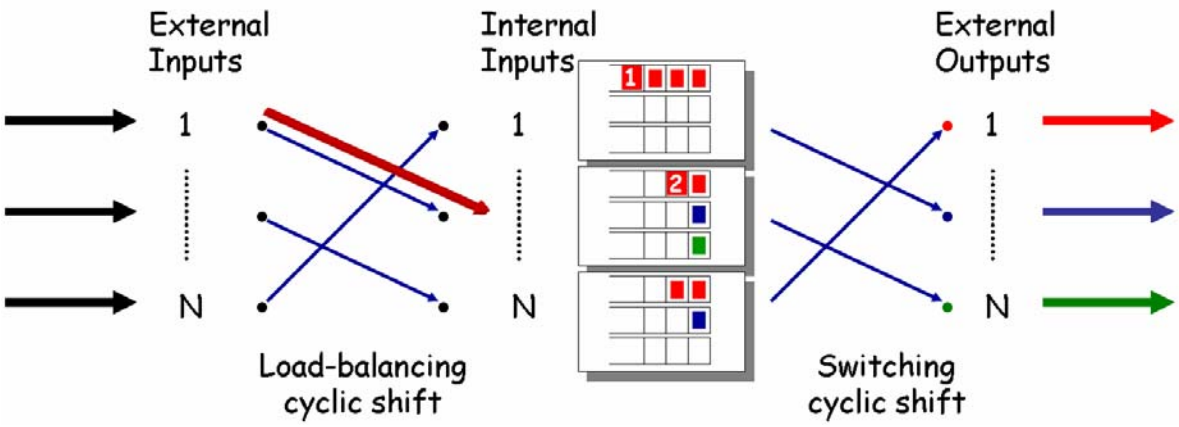
Broadly speaking, There are two approaches to preventing mis-sequencing: To prevent packets from becoming mis-sequenced anywhere in the router or to bound the amount of mis-sequencing, and use a re-sequencing buffer in the third stage. Each of the examples that we will see in this report falls to either one of these two divisions.

The organization of this report is as follows:

In the next section we will briefly examine a simple example to understand the cause of mis-sequencing. After that, in next few sections we will go through different architectures and discuss about their schemes to prevent sequencing and the performance overheads. In the next section that follows, we will outline an algorithm to solve this packet mis-sequencing problem that we have developed. In the last section, we will present a brief comparative discussion on the relative merits and demerits of the approaches that we have discussed in this report.

2. Out of Sequence Packets [Example]:

The following diagram tries to show how packets can be mis-sequenced in the case of a Load Balanced BvN switch with single stage of buffering:



In the figure, the red cells denote the packets that are destined to go to output 1. Now, there are two cells that are labeled with numbers 1 and 2. The packet labeled 1 has arrived at cell one and the first cyclic shift load balancing stage of LB BvN switch has put it in the VOQ corresponding to the first output in the first middle stage. In the next cycle, packet labeled 2 has come in input one, and it has been put in the VOQ corresponding to the first output in the second middle stage. But, as we can see in the picture, the VOQ of output 1 at the second middle stage is less congested than the VOQ of output 1 at the first middle stage. Since, the packets from the VOQs of the middle stage are pushed to the output in a cyclic manner, the packet labeled 2 will reach the output port 1 before the packet labeled 1, and we have mis-sequencing at the output port 1.

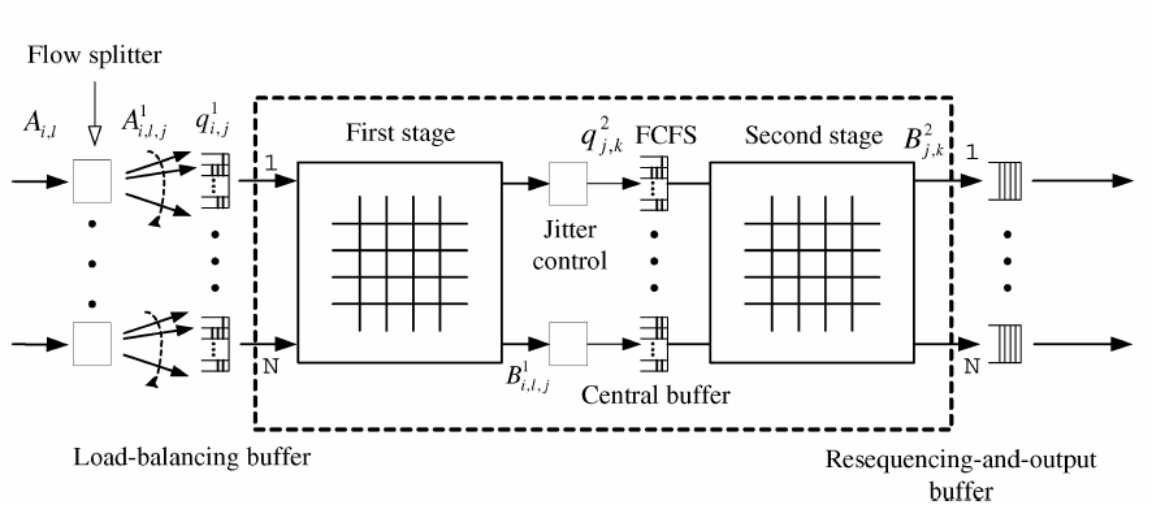
3. Load Balanced BvN Switch:

The previous example shows how packets can be mis-sequenced for a Load Balanced BvN Switch. In the single-stage buffering scheme [1], the authors did not provide any methods to tackle the problem. This problem has been addressed in a follow up companion paper from the same authors [2].

One quick fix idea comes to mind that seems to solve the out-of-sequence problem, namely having a resequencing buffer at the output. But since the packets are distributed according to the arrival time in the first stage, no bounds can be applied to the resequencing buffer at the output. In order to have a bound on the size of the resequencing buffer at the output, we need to load balance each flow rather than load balancing on only the arrival time. To achieve this, a flow splitter and a load-balancing buffer is added in front of the first stage of original LB BvN switch [1]. Just as a reminder, it is worthwhile to mention here that in the case of a switch, a flow can be thought as the ordered set of packets coming to the same input port and destined to go to same output port. In this case, for an N x N switch, the load-balancing buffer at each input port of the first stage consists of N VOQs destined for the N output ports of that stage. Packets

from the same flow (that is, packets residing in a single VOQ of an input stage) are split in the round robin fashion to the N VOQs of the middle stage. By doing so, packets belonging to each flow get almost evenly distributed to the input ports of the second stage. Packets are served from the middle stage VOQs to the output stage in a FCFS scheme.

This figure shows the architecture of the two-stage buffering scheme of LB BvN:



[Fig 1] Load Balanced BvN Switch

Finally, there is a resequencing buffer at the output stage. But now, since the input packets are load balanced according to the flows, we can expect that the resequencing buffer at the output will be bounded. This is indeed the case – the authors have proven in this paper, that if FCFS is used at the beginning of stage two, then the following results are true:

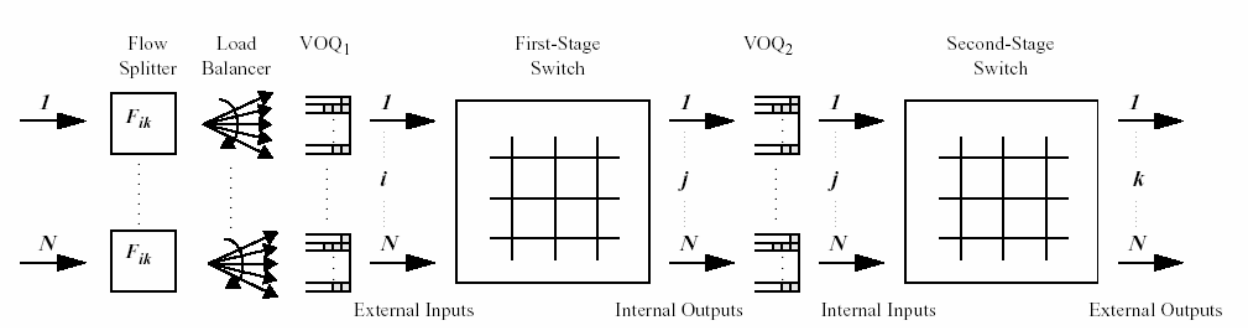
(i) The load-balancing buffer at an input port of the first stage is bounded above by NL_{max}.

(ii) The resequencing and output buffer at an output port is bounded above by NM_{max}, Where L_{max} is the maximum number of flows at an input port, and M_{max} is the maximum number of flows at an output port.

There is some delay involved due to the Jitter Control and the resequencing buffer. The delay can be reduced somewhat if instead of FCFS at the input port of the second stage. But this approach requires a large resequencing buffer (= $(N-1)(M_{max} + L_{max})$) at the output. Moreover, computing the *deadline time* required to implement the EDF scheme requires global knowledge, which may be a problem for efficient implementation.

4. 3DQ – Extension to VOQ

McKeown et al. from Stanford University [3] have also done some work on packet mis-sequencing problem. They also tried to follow the structure of LB BvN [1], but have introduced some methods to tackle the out of sequence problem. In fact, their switch architecture closely follows that of [1]. Below is a diagram showing the switch architecture used by [3]:



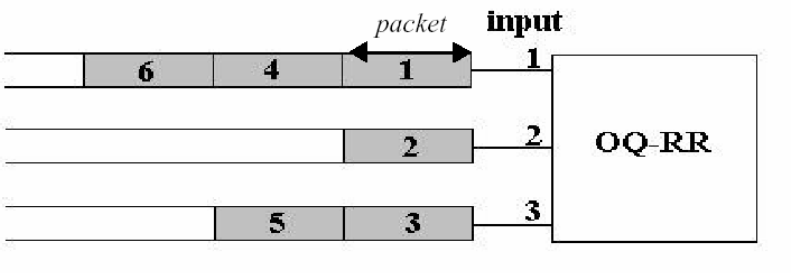
[Fig 2]

The authors of [3] have formulated the problem is a different way. If we follow the method of [2], then there will not be any mis-sequencing but there will be a requirement of a resequencing buffer at the output. The approach given here tries to do away with this requirement. In the default algorithm of [1], there can be *Head of Line Blocking* in VOQ_2 in the above figure [fig2]. As an example, consider a packet, p that sits in the VOQ_2 *(j, k)*. Assume that was the earliest arriving packet to the switch among all packets in its VOQ_2 *(j, k)*, but that is not currently sitting at head-of-line (HOL) in its VOQ_2. Packet is obviously the earliest arriving packet of its flow in VOQ_2, and therefore sits in front of the other packets of its flow in VOQ_2. However, it is blocked by packets ahead of it that arrived later to different external inputs and are also scheduled to depart from same external output *k*. This is classical HOL blocking, and the solution is to subdivide each VOQ_2 into a separate queue for each external input.

To handle this HOL blocking problem, the VOQ_2 s are replaced by 3DQs. 3DQs are three-dimensional queues with one queue per *(i, j, k)* – and so, there are total of N_3 3DQs. With 3DQs, the earliest cell for *(j, k)* is always the HOL cell in its queue.

The 3DQ improvement prevents HOL blocking. The authors have introduced one more technique that they have used in addition to 3DQ to prevent mis-sequencing. This idea is referred as FFF, or Full Frames First. This is different from section 5 in the sense that, in section 5, we don't need 3DQs, but we need some resequencing buffers at the output. But in this case, by using 3DQ and FFF simultaneously, we do not need any resequencing buffer at the output end. The following is a brief discussion about how FFF works:

Consider the following figure:



[Fig 3] OQ-RR Example

In figure 3, assume all packets are going to same output. The numbers on the packets correspond to the order in which they will be serviced. Therefore OQ-RR will service packet 5 before packet 6, even if packet 6 has arrived earlier. And OQ-RR is output conserving, since if there is at least one packet in the queue, then OQ-RR is not idle. Now if we switch our attention from packets to *frames*, where each frame contains *N* packets. The new algorithm, called Frames-RR, first services all full frames in round robin manner. When there are no full frames, the algorithm services non-full frames in round robin order. A frame is full if its N^{th} slot contains a packet. This algorithms is not work conserving for packets, but it is work conserving with respect to full frames, i.e., if there is a full frame, then the system will not be idle.

The FFF algorithm applies Frames-RR to the 3DQs in the second stage of the switch. To implement this operation, each output stage stores two round robin pointers to remember from which of the first stage input it got the last full frame and non-full frame of packets.

Lets define $p_{ff}(k)$ as the pointer to the input stage from where the k^{th} output got its last full frame. And $p_{nff}(k)$ is the pointer to the input stage from where the k^{th} output stage got its last non-full frame. To implement the FFF algorithm, output stage performs these three operations each cycle:

(i) Determine which of the frames $f(i, k)$ is full, where $i \in \{1, 2, \ldots, N\}$. In this case a frame for (i, k) is defined as $f(i, k) = \{(i, p_{ik}, k), (i, p_{ik} + 1, k) \ldots (i, N, k)\}$ and p_{ik} is the pointer to the last intermediate input for the next in-order cell. A frame $f(i, k)$ is full if for every 3DQ (i, j, k) for $j = \{ p_{ik}, p_{ik} + 1, \ldots, N\}$ is non-empty.

(ii) Staring at $p_{ff}(k)$ find the first full frame. And update the pointer. If the first full frame arrived from input i_{ff}, then $p_{ff}(k) = i_{ff} + 1 \text{ modulo } N$. if there is no full frame, $p_{ff}(k)$ does not change.

(iii) If there is no full frame, staring at $p_{nff}(k)$ find the first non-full frame. And update the pointer as $p_{nff}(k) = p_{nff}(k) + 1 \text{ modulo } N$

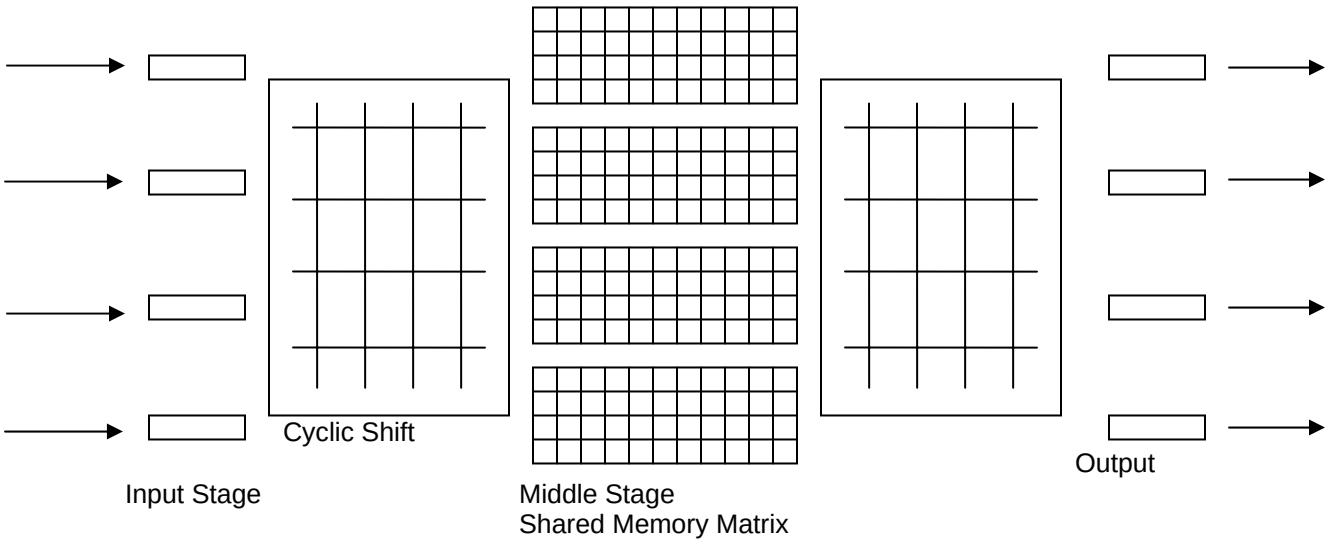
The main advantage of this algorithm is that there is no need for any resequencing buffer at the output, and the authors have claimed that is it simple to implement also. But is has some significant hardware and memory requirements, like, it needs two stage switching and a load balancer and flow splitter in the front of the switch. The requirement of N^3 3DQs at the intermediate stage requires more memory and complex buffer management scheme. And this cubic exponent makes the algorithm difficult to scale to switches of thousands of input and output port.

5. Full Ordered Frame First (FOFF):

The authors of [4] have shown another method to eliminate the mis-sequencing problem. This is almost similar to the approach described in section 4, but with the significant difference that in this case we don't need to have 3-Dimensional queues at the intermediate stage. But unlike the approach in section 4, we need to have a resequencing buffer at the output stage. FOFF algorithm bounds the difference in lengths of the VOQs in the second stage and then uses a resequencing buffer at the end stage. FOFF runs on each linecard using information locally available. So, there is no inter-linecard communication complexity involved. The input linecard keeps N FIFO queues – one for each output. A FIFO is served only when it contains N or more packets, if possible. So, the algorithm works as follows: Every N time slots, an input *i* selects to a queue to serve for the next *N* time cycle. First it picks round robin from the queues, which contain more than N packets, and if there is no such queues, it selects round robin from the non-empty queues. Clearly, if there is always a queue with more than N packets, there will not be *any* mis-sequencing. Mis-sequencing arises only when no queue has *N* packets; but the amount of mis-sequencing is bounded, and is corrected in the third stage using a fixed length re-sequencing buffer. The authors have shown that the size of the resequencing buffer is bounded by $N^2 + 1$.

6. Our Algorithm

We have attempted to solve the mis-sequencing problem in a different way. The following is a brief description of the algorithm:



[Fig 4] Our Algorithm

Consider the middle stage of the switch as a shared memory matrix. For each middle stage, we have N rows in the matrix corresponding to N VOQs of N output ports. Each input port *i* keeps the record C_{ij}, for $j = 1$ to N, which denotes the column number of the last packet where a packet from *i* to *j* has been written in the shared memory in some intermediate stage. Now, we will avoid mis-sequencing at the output if we always make sure that $C_{ij}(time = t+1) \geq C_{ij}(time = t)$. To achieve this, we follow this algorithm:

1. Input *i* receives a packet going to output *j*, and at that time the cyclic shift rotation at the first stage connects input i to intermediate stage *k*.
2. Input stage *i* sends the packet to the k^{th} intermediate stage with the value of C_{ij}.
3. The memory manager at the k^{th} input stage finds a free memory block that has column index $\geq C_{ij}$. It puts the packet in that free slot, say, at column C'_{ij}, and then it sends back the value of C'_{ij} to input stage *i*. The i^{th} input stage then updates its C_{ij} value with this new value C'_{ij}.
4. The second cyclic shift operates in the usual fashion as the LB BvN Switch.

The memory manager algorithm can be implemented using a Set-Union-Find algorithm, which runs in amortized $O(Log^*(n))$, which is almost a constant for all practical purposes.

The correctness of the algorithms is evident. But, we are yet to analyze the throughput guarantees of this algorithm, and towards this goal, we are trying to do some simulation of the system to see empirically how good the throughput of the system will be. Admittedly there will be some holes in the memory matrix in the middle stage, but intuitively it seems that any algorithm, which relies on the dumb cyclic shift algorithm and later tries to prevent mis-sequencing or have some resequencing buffer at the output, will incur at least the same amount of delay that we incur here due to the holes. But as of now, this is an intuition only, and the formal mathematical proof is not yet done.

Moreover, other proposed algorithms that depend on the idea of FOFF, does not scale arbitrarily with the number of input-output ports. Those algorithms have the possibility of breaking down for large N (~hundreds of thousands). But, it seems that this algorithms will work even on those cases. Moreover, the amortized complexity is really low, almost a constant.

The only drawback of this algorithm seems to be that it needs to send a value back to the input ports from the intermediate ports. But that is a price we have to pay in order to have this simplified, yet efficient algorithm.

7. Discussion:

The previous sections presented a very brief overview of how the packet mis-sequencing problem arises in the case of LB BvN switches or switches derived from that architecture, and various measures to tackle the problem. In section 2, the solution proposed by Chang et al [2] seems a feasible solution, but there is a lot of complexity involved in implementation of the resequencing buffer and the jitter control mechanism. In contrast to that, the FFF and 3DQ scheme [3] is a really clever idea to circumvent the problem which alleviates the necessity of resequencing buffers at the output. Chang et al. has adopted the solution of the frame-based scheme of [3] in their extension [5] of the LB BvN switch in order to provide QOS guarantees in a derivative of LB BvN switch.

8. References:

1. 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, 2002.
2. C. S. Chang, D. S. Lee and C. M. Lien, "Load Balanced Birkhoff-von Neumann Switches, Part II: Multi-Stage Buffering", Computer Communications, vol. 25, pp. 623-634, 2002
3. I. Keslassy, N. McKeown, "Maintaining packet order in two stage switches," Proceedings of the IEEE Infocom, June 2002.
4. I. Keslassy, S. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, N. McKeown, "Scaling Internet Routers Using Optics", ACM SIGCOMM, Karlsruhe, Germany, Aug. 200
5. C. S. Chang, D. S. Lee, and C. Y. Yue, "Providing Guaranteed Rate Services in the Load Balanced Birkhoff-von Neumann Switches", IEEE INFOCOM, San Francisco, March 2003.