Chapter 6

Optical FIFO Buffers

All-optical packet switches require a means of delaying optical packets and buffering them when the output link is busy. This chapter addresses the problem of emulating the FIFO queueing scheme using optical delay lines.

6.1 Elements of optical buffering

Figure 6.1(a) shows how a single optical delay line and a 2×2 optical switch can create an optical buffering element. When a packet arrives at the buffer, the switch goes to the crossed state in order to direct the packet to the delay line. The length of the delay line is large enough to hold the entire packet. Once the entire packet enters the delay line, the switch changes to the parallel state. This forms a loop where the packet circulates until its departure time, when the switch goes to the crossed state again and lets the packet leave the buffer.

A FIFO buffer of size N can be built by concatenating N delay loops, each capable of holding one packet (Figure 6.1(b)). Each packet is stored in the rightmost delay line which is not full. When a packet leaves the buffer, all other packets are shifted to the right. However, given the complexity of building optical switches, this architecture may not be practical for large N, as the number of 2×2 switches grows linearly with the number of packets that need to be stored. Moreover, the power attenuation caused by passing through the optical switches imposes a constraint on the number of packet

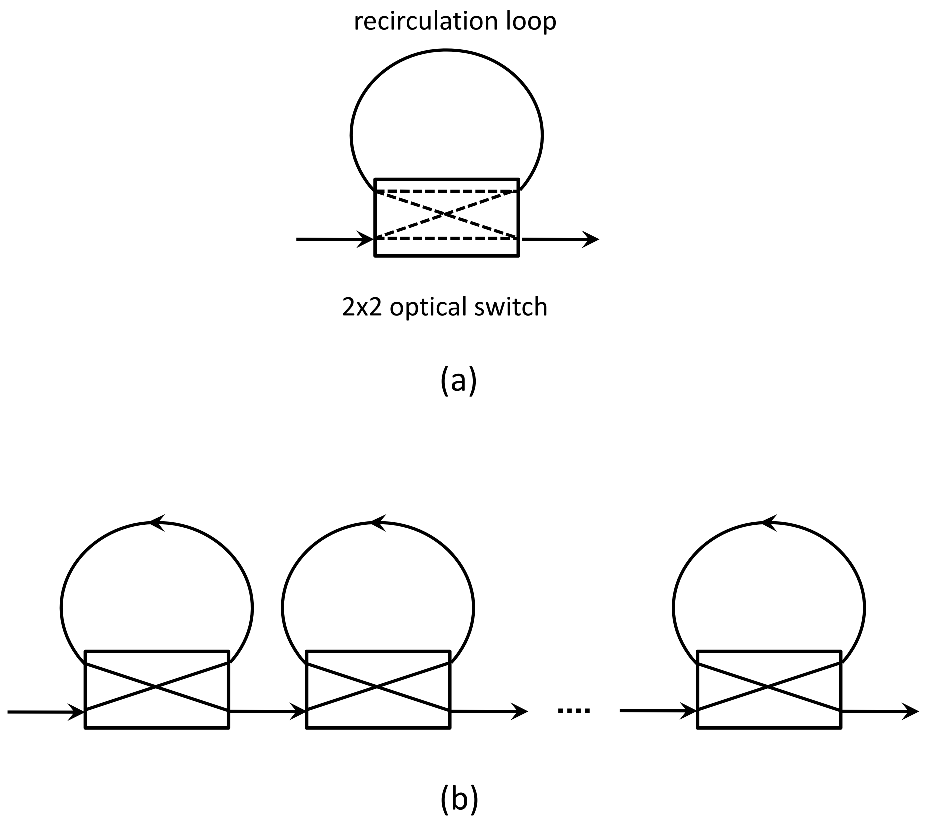

Figure 6.1: (a) Building an optical buffer with a delay line and a 2×2 optical switch. (b) Buffering multiple packets.

recirculations. The challenge is to design a buffer that works with a smaller number of switches, yet has the capacity of the linear architecture shown in Figure 6.1(b).

The problem of constructing optical queues and multiplexers with a minimum number of optical delay lines and optical switches has been studied in a number of recent works [28, 18, 19, 22, 43]. Sarwate and Anantharam have shown that for emulating any priority queue of length N, the minimum number of required delay lines is $O(\log N)$ [43]. They also propose a construction that achieves this emulation by $O(\sqrt{N})$ delay lines.

Recently, C.S. Chang et al. proposed a recursive approach for constructing optical

FIFO buffers with $O(\log N)$ delay lines [18]. Their proposed construction needs to keep track of the longest and the shortest queues in each step of the construction. Here, we present an architecture with a same number of delay lines, which is simpler in the sense that packet scheduling is independent for each delay line, as we will explain in the following sections.

6.2 Preliminaries and assumptions

We assume that all packets are of equal sizes, time is slotted, and the length of a time slot is the time needed for a packet to completely enter (or leave) an optical delay line. At each time slot, the buffer receives at most one arrival request and one departure request. The arrival and departure sequences are not known in advance.

The length of a delay line is the total number of back-to-back packets the delay line can hold. If we have a delay line of length L, it takes $L+1$ time slots for a packet to enter the delay line, traverse it once, and leave.

The states of the optical switches are controlled by a scheduling algorithm, and are changed at the beginning of each time slot. Depending on the state of a switch, a packet at the head of a delay line is either transferred to the tail of the same delay line, to a different delay line, or to the output port.

We define the (departure) order of a packet p as the total number of packets in the buffer that have arrived before p. In a FIFO queueing system, the order of each packet is decremented by one after each departure.

6.3 Buffering architecture

Exact emulation of a FIFO queue requires that with any arbitrary sequence of arrival and departure requests, the buffer has exactly the same departure and drop sequences as its equivalent FIFO buffer. What makes the delay-line based design of optical FIFO challenging is that the arrival and departure requests are not known in advance.

Figure 6.2 shows our optical FIFO construction for buffering $N-1$ packets. The delay loops have exponentially growing lengths of $1, 2, 4, ..., N/2$, and each comes with a 2×2 switch connecting the delay loop to the path between the input and the output ports. Without loss of generality, we assume that N is a power of 2.

Incoming packets are buffered by going through a subset of the optical delay lines. When a packet arrives at the buffer, the scheduler decides if the arriving packet needs to go through the waiting line first, or if it can be directly delivered to one of the delay lines. As time progresses, optical packets in each delay line move towards the head of the delay line. At the end of each time slot, the scheduler determines the next locations of the head-of-line packets, and directs these packets towards their scheduled locations by configuring the 2×2 switches.

The waiting line W operates as a time regulator of the arriving packets. Since the time intervals between successive arriving packets are not known in advance, the scheduler uses this waiting line to adjust the locations of the packets in the delay lines. When the scheduler decides to direct an arriving packet to the waiting line, it knows exactly how long the packet needs to be kept there before moving to one of the delay lines. Packets leave W in a FIFO order. Section 6.3.1 shows that our proposed scheduling algorithm can make this buffering system emulate a FIFO buffer of size $N-1$. Section 6.3.2 describes how the waiting line W, too, can be constructed by $\log N - 1$ delay lines. This makes the total number of delay lines equal to $2 \log N - 1$.

6.3.1 Packet scheduling

The main idea of our scheduling algorithm is to place the packets in the delay lines consecutively, i.e., packets with successive departure orders are placed back to back in the delay lines [1]. To achieve this, the scheduler places an arriving packet in a delay line only if the preceding packet has been in a tail location in the previous time slot. Otherwise, the scheduler holds the packet in W for a proper number of time slots. The waiting time is equal to the time it takes for the preceding packet to traverse its current delay line and go to a tail location. During this waiting time, the scheduler keeps all the arriving packets in W in a FIFO order.

When there is a departure request, if the buffer is nonempty, the packet with order

[1]This is in contrast with what is proposed by Sarwate and Anantharam [43], where arriving packets are allowed to fill out any available locations at the tail of the delay lines.

one is delivered to the output port. The departure orders of all the remaining packets are then reduced by one.

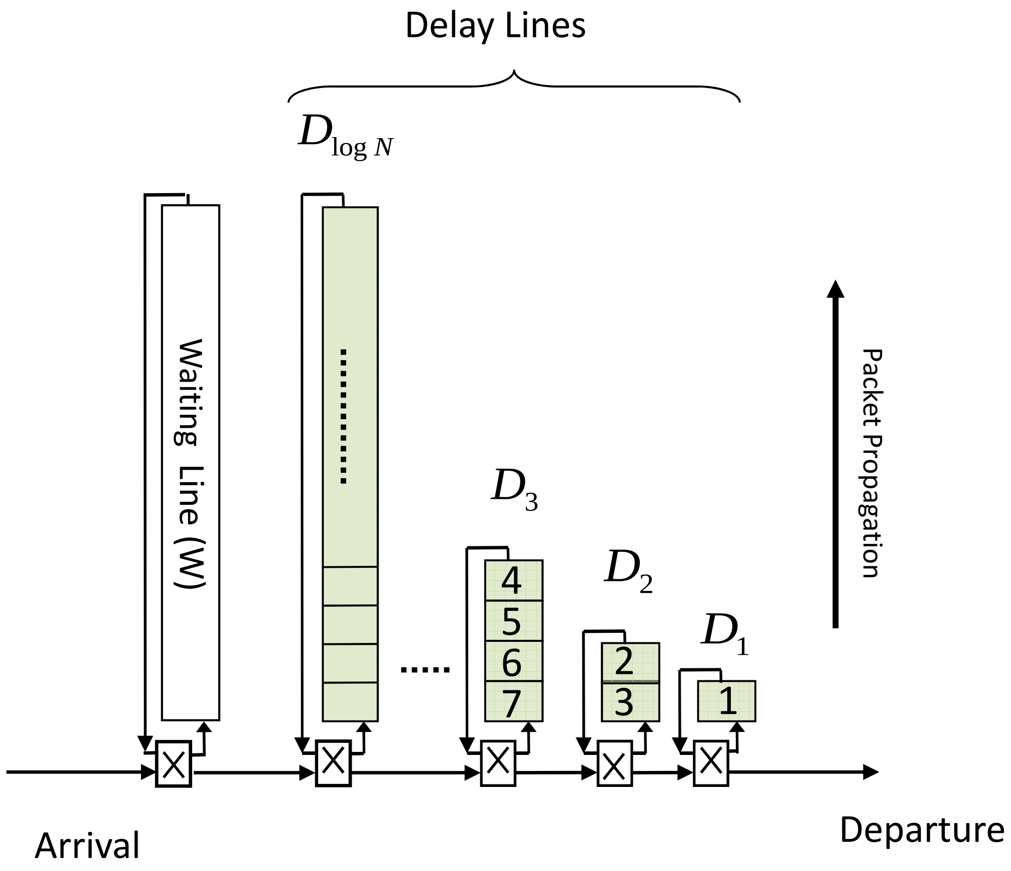

Figure 6.2: Constructing a FIFO buffer with optical delay loops. The scheduler controls the locations of the arriving packets by passing them through a waiting line first. Here, the delay loops are shown as straight lines for the sake of presentation.

At the end of each time slot, the scheduler determines the next locations of the head-of-line packets. The scheduler decides whether to recirculate a packet in its current delay line or to send it to a shorter delay line. This decision is made independently for each head-of-line packet, based only on the order of the packet: in the following time slot, the packet will be transferred to the tail location of the longest delay line whose length is not greater than the departure order of the packet. More precisely, if the head-of-line packet has departure order k, then it will be placed in delay line $D_{\lfloor \log k \rfloor + 1}$. This ensures that the packet with departure order one is always at the head of one of the delay lines.

The same scheduling policy applies when the waiting time of the head-of-line packet in W is decremented to zero; the packet will be transferred to the tail of the longest delay line whose length is not greater than the departure order of the packet.

Theorem 6.1. Under scheduling Algorithm A, the set of delay lines $\{D_i\}$ and waiting line W exactly emulates a FIFO buffer of size $N-1$.

The proof is in Appendix E, where we show that Algorithm A has the following three properties.

(i) Occupancy of W is always smaller than $N/2$.

(ii) There is no contention among head-of-line packets.

(iii) The packet with departure order one is always at the head of a delay line.

The first property is required for constructing the waiting line W with $\log N - 1$ delay lines. This construction is explained in the next section.

The second property is required to avoid dropping packets after they enter the buffer. The scheduling algorithm needs to relocate the head-of-line packets in such a way that there is no contention for a given location, i.e., at any time slot, at most one packet should be switched into each delay line.

The last property is required so that the departure sequence will exactly follow that of a FIFO, without any delay.

The above properties guarantee that no arriving packet will be dropped as long as the total number of packets in the buffer remains less than N. Packets depart in the same order they arrive, and each packet is delivered to the output link at the same time as in the emulated FIFO buffer.

Algorithm *A*- Packet Scheduling

```
1. arrival event

      let $k$ be the total number of packets in the buffer.
      if $k = N - 1$ then
          drop the arrived packet
      else
          denote by $p_{k+1}$ the arrived packet and by $p_k$ its preceding packet in
          the buffer.
          if $p_k$ is in $W$ then
              place $p_{k+1}$ in $W$
          else
              $waiting\_time \leftarrow (h+1) \mod d$, where $d$ is the length of the delay
              line which contains $p_k$, and $h$ is the distance of $p_k$ from the head
              of the line.
              if $waiting\_time > 0$, place the arrived packet in $W$. Otherwise,
              place it in the longest queue with length smaller than or equal to
              the order of the packet.

2. departure event

      remove the packet with order 1 from the buffer, and decrease the order of
      all the remaining packets by 1.

3. scheduling the head-of-line packets

      for $i = 1, 2, ..., \log N$, move the packet at the head of $D_i$ to the longest
      delay line with length smaller than or equal to the order of the packet.

      if $(waiting\_time > 0)$ then
          $waiting\_time \leftarrow waiting\_time - 1$

      if $(waiting\_time = 0)$ & ($W$ is nonempty) then
          remove the head-of-line packet from $W$. Place the packet in the longest
          delay line with length smaller than or equal to the order of the packet.
```

6.3.2 Constructing the waiting line

To avoid future contention among head-of-line packets, the scheduling algorithm does not allow any void places between packets with successive departure orders. To achieve this, an arriving packet is kept in the waiting line until its preceding packet goes to a tail location. The waiting line needs to be able to buffer at most $N/2$ packets because our algorithm always keeps the number of packets in the waiting line smaller than $N/2$ (property (i) in Theorem 6.1).

Consider a set of delay lines $D'_i, i = 1, 2, ..., \log N - 1$, where the lengths of the delay lines grow as $1, 2, 4, ..., 2^{(logN)-1}$, generating an overall delay length of $N - 1$. Upon the arrival of each packet, the scheduler knows how long the packet needs to be kept in W before being transferred to one of the delay lines. Based on the availability of this information, we develop Algorithm B in the following way.

When a packet p arrives at the buffer and is scheduled by Algorithm A to be placed in W, then

- Calculate the binary expansion of the waiting time of p, i.e., the duration that p needs to wait in W before moving to one of the delay lines $\{D_i\}$. This determines which delay lines from the set $\{D'_i\}$ the packet should traverse before leaving W: if the jth bit in the binary representation is non-zero, then the packet needs to traverse delay line D'_j.
- Starting from the shortest line, when the packet reaches the end of a delay line, place it in the next delay line corresponding to the next non-zero bit.

Packet p leaves W after it traverses all the delay lines corresponding to the non-zero bits in its binary expansion. A *waiting* packet traverses any delay line in the set $\{D'_i\}$ at most once, and never recirculates in the same delay line.

Because the waiting time of each packet is known upon its arrival, the sequence of switch states in the following time slots can be determined at the time the packet arrives. Therefore, the switching decisions do not need to be delayed until packets reach tail locations. Consider a packet that arrives at time t and the binary representation of its waiting time is $b_{(\log N)-1}...b_2b_1$. For each nonzero bit b_i, the switch of line D'_i needs to be closed twice; at time $t + \sum_{j=0}^{i-1} b_j 2^j$, to let the packet enter the delay line, and at time $t + \sum_{j=0}^{i} b_j 2^j$ to let the packet move to the next scheduled delay line (b_0 is assumed to be zero).

Tradeoff between number of switches and total fiber length. Although the number of 2×2 switches in our proposed scheme is only $2 \log N$, the total length of delay lines used for exact emulation is $1.5N$. The scheduling algorithm needs this extra $N/2$ length in order to place the packets in delay lines $\{D_i\}$ consecutively by sending the arriving packets to the waiting line first. The length of the waiting line must be equal to the length of the longest delay line (which corresponds to the longest possible time it takes for a packet in any of the delay lines goes to a tail location). Consequently, the extra fiber length will be reduced if the length of the longest delay line is reduced.

Figure 6.3 illustrates an example where the longest delay line is replaced by two half-sized delay lines. With this configuration, the size of the waiting time W can be halved too, which results in only 0.25% extra fiber length. By repeating this procedure k times, the total number of switches added to the system will be $2^{k+1} - 2k - 2$, while the extra fiber length will approximately be $\frac{1}{2^{k+1}}$ that of the original scheme. By setting $K = 2^k$, the exact tradeoff is stated in the following theorem.

Theorem 6.2. A FIFO queue of size $N - 1$ can be emulated by $2(\log N + K - \log K) - 3$ delay lines with an aggregate length of $N - 1 + \frac{N-2K}{2K}$, where $K = 2^0, 2^1, ..., 2^{(\log N)-1}$.

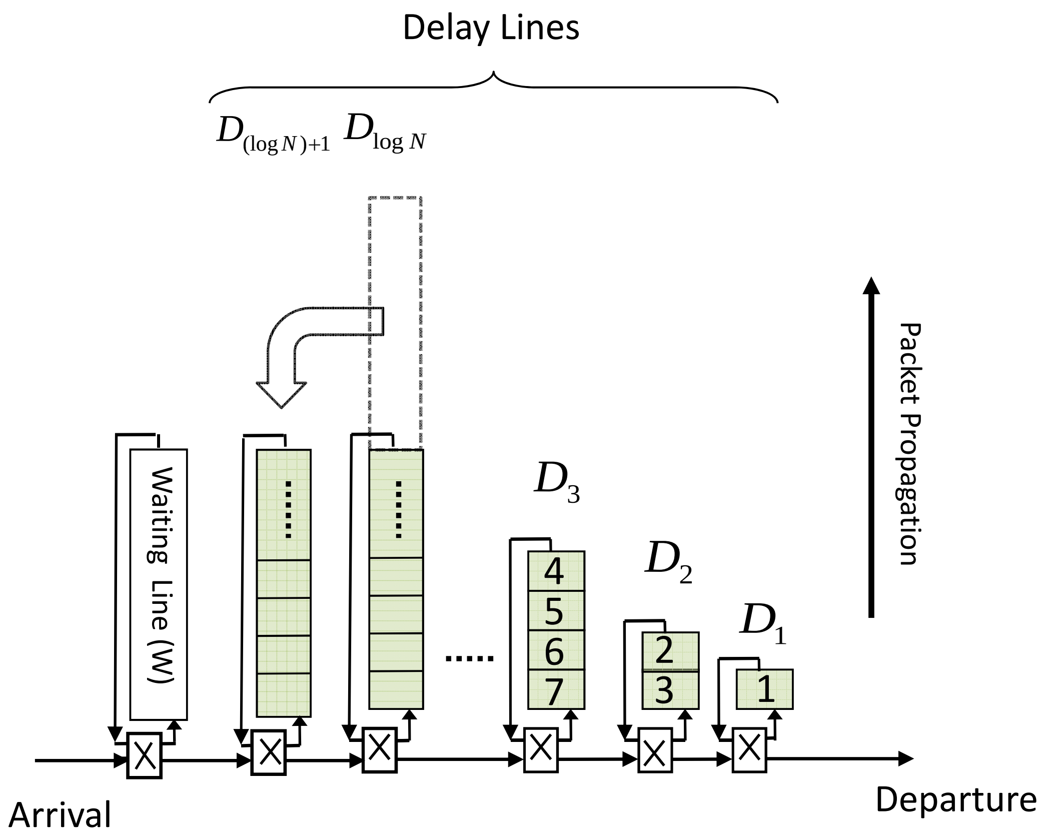

Figure 6.3: Trade-off between the number of delay lines and the maximum delay line length.