Fast Switching and Fair Control of Congested Flow in Broad-Band Networks

Manolis G.H. Katevenis

April 1987

RCC/CSI/TR/1987/001
Technical Report Series
Computer Science Institute, Research Center of Crete
P.O.Box 1385, Heraklio, Crete, 711-10 Greece
Tel.: +30.81.229302, Telex: 262389 CCI GR
E-mail — UUCP: mcva!ariadnelkatevenis

Fast Switching and Fair Control of Congested Flow in Broad-Band Networks

Manolis G.H. Katevenis

Computer Science Institute, Research Center of Crete

Abstract

A new switching architecture is proposed, based on the tradeoffs of modern VLSI technology — inexpensive memory and two-dimensional layout structures. Today, it is economically feasible to preallocate buffer space individually to each virtual circuit in every node, so that "congestion" ceases to have negative effects. On the contrary, when some low-priority circuits offer more traffic than the network can carry, full utilization of the link bandwidth is achieved. In this context, the allocation of bandwidth can be done automatically and in a "fair" way, if packets are multiplexed by circularly scanning all virtual circuits and transmitting one packet from each "ready" circuit. This multiplexing algorithm equally distributes all the available link BW to all the VC's that can use it (other than equal distribution is also possible), while it also guarantees an upper bound for the total packet delay through non-congested VC's (VC's that use less than their share of BW). We present methods for hardware implementation of fast such circular scans, and propose a structure for the switching nodes of such networks, consisting of a cross-bar arrangement like a systolic array that performs merge-sorting. It is ideally suited for physical layout on printed-circuit boards or with wafer-scale integration.
1. Introduction

The continuing advances in the technology of integrated circuit fabrication have been the driving force for important changes in the architecture of digital systems. In data networks, modern VLSI technology makes possible some architectures that were completely impractical a few years ago. Specifically, optical fibers enable broad-band point-to-point links. VLSI chips are being used to implement in hardware not only the data-link but also the network layers of protocols. In this paper, we will focus our attention on the consequences of a third aspect of modern technology: inexpensive memory.

As shown in section 2, it is possible, even when memory use is wasteful, to require no more than about 10 memory chips for every kilometer of 1 Gbit/s optical fiber. In other words, buffer memory is no more a scarce resource in networks. Instead, bandwidth is becoming the scarce resource, since the demand for it is not likely to cease growing. For continuous and full utilization of the available link bandwidth, the traffic that is offered to the network must be more than the network's carrying capacity. This situation is described traditionally as "congestion". We use this term, however, in a radically different sense than the one adopted that signifies the situation where "performance degrades due to too much offered load" [Tan81, p.215]. Our main theme is that, today, "congestion" should cease to have a negative connotation, because all of its negative effects can be eliminated by using enough (inexpensive) buffer memory and an appropriate method of multiplexing packets.

We consider networks that consist of point-to-point single-access links, and virtual circuits. Single-access links have the advantage over multiple-access ones of simpler control and of not wasting bandwidth during the turn-around periods that are required between transmissions by different stations. Concerning the bandwidth that each individual "connection" uses, this paper follows the principle that it should vary on an instantaneous basis. Fixed bandwidth allocation is inappropriate or wasteful for variable bit-rate traffic, whether data or compressed voice or video. Packet-switching, which achieves variable bit-rate, does not imply long delays any more with the advent of special-purpose VLSI processors which implement the protocols in hardware.

This paper deals with "virtual-circuit" networks, and not "datagram" ones, for two reasons. First, with virtual-circuits (VC), the cost of routing decisions is paid only once per VC set-up, while with datagrams it has to be paid for each and every packet. The second reason has to do with flow-control and dealing with congestion. With datagrams, packets "lose their identity" once they enter the network: it is not possible to deal collectively with all the packets that belong to a certain "connection" at any point after their entrance into the net, since they may be scattered all around it.

For convenience and simplicity, in this paper we assume end-to-end (as opposed to node-by-node) retransmission of corrupted or lost packets. However,
this is not an essential assumption, since the ideas and architectures presented can be modified to apply as well to networks that provide retransmission at the link level.

Section 2 of the paper computes the amount of buffer memory needed in order to eliminate the negative effects of congestion. Section 3 deals with the packet-multiplexing algorithm, which becomes important under congestion conditions. A circular-scan method and its properties are studied in detail, and we compute the upper bound guaranteed by the method for the switching delay of packets through non-congested VC's, in the presence of congestion in other VC's. Section 4 presents methods for the hardware implementation of the circular scan method, which are appropriate for modern VLSI technology. An approximate implementation is also presented, which is less expensive and equally good as the exact one in terms of performance. In section 5, we propose an internal organization for the switching nodes that incorporate the above buffer-allocation and multiplexing policies. The circular scan is done in a distributed and parallel way, in a structure that resembles a systolic-array performing a merge sort. This structure is ideal for planar layout on Printed-Circuit Boards or with Wafer-Scale Integration.

2. Control of Congested Flow

2.1 Node-to-Node Flow-Control and Corresponding Buffer Size

In the network architecture assumed in this paper, flow control must be applied on a node-to-node basis — it cannot work in an end-to-end fashion. This is so because we intend to keep the links busy with traffic all the time. It would be impossible then to control and to orchestrate such a complex flow process remotely, without local feedback signals. Flow-control can be done via a sliding-window protocol. This involves the transmission of some sort of acknowledgement signals in the backward direction. Since we make the simplifying assumption that only end-to-end retransmissions are provided, these backwards signals have nothing to do with acknowledging correct reception of packets; they are usually called “permits”. One such permit is sent back to the transmitting node every time when the buffer in the receiving node has room for one more packet, because some other packet has just been forwarded on.

We assume that nodes are connected by links running in both directions (we anticipate that these will consist of two separate simplex channels, as opposed to a single duplex one, to avoid the long turn-around delays). Packets will travel then, in each direction, intermixed with the permits for the traffic of the other direction, or with these permits piggy-back’ed inside them. This means that the overhead of permits is a portion of the total bandwidth equal to the ratio of the size of one permit relative to the size of one packet. If this overhead is high, it can be reduced by a factor of n by sending permits only once for every n packets that have been
forwarded on; in that case, the buffer size that is computed below should be increased by \((n-1)p\).

In order for the sliding-window protocol not to slow down a virtual circuit (VC) which could otherwise proceed at higher rate, the buffer space \(BU\) that is needed in each node has to be:

\[
BU \geq 2t_p \cdot BW_{\text{peak}} + p
\]

where \(t_p\) is the (speed-of-light) propagation delay between the previous and this node, \(BW_{\text{peak}}\) is the maximum possible instantaneous bandwidth of the VC, and \(p\) is the size of one packet. This formula results from the following reasoning: When the traffic proceeds at maximum rate, a new packet is transmitted forward and a corresponding permit is transmitted backwards every \(T = p / BW_{\text{peak}}\) time period. Thus, there are \(t_p / T\) packets on their way from the previous node to this one, and that many more permits on their way from this node to the previous one. In case the outgoing traffic from \(BU\) suddenly stops, \(BU\) must be large enough to hold all those \(2t_p / T\) packets which it will receive until “the pipeline dries out”. That is, \(BU\) must be \(\geq [2t_p / (p / BW_{\text{peak}})]p = 2t_p \cdot BW_{\text{peak}}\). This same buffer size is also enough to guarantee that when the outgoing traffic from \(BU\) starts again, \(BU\) will be able to supply it with enough packets for the period of \(2t_p\) that it will take for the first permit to travel back to the previous node and for the first packet from that node to arrive at \(BU\), that is for the time that it will take “for the pipeline to fill in again”. This period has to be increased by \(t_{\text{sv}}\), the switching delay of the previous node, that is the time that that node takes from the moment it receives a permit to the moment it sends a corresponding packet. In section 3.3 we will see that \(t_{\text{sv}} \leq p / BW_{\text{sn}}\), where \(BW_{\text{sn}}\) is an upper bound for the bandwidth that any VC can get under the momentary load conditions. This additional \(t_{\text{sv}}\) contributes an additional \(t_{\text{sv}} \cdot BW_{\text{peak}} = p\) to the minimum buffer size \(BU\).

As an example of buffer size, let us assume that the packet size is \(p = 256\) bits (32 bytes), and the speed of light (in the dielectric) is 250Mm/s. Then \(t_p\) is 4\(\mu\)s per kilometer of link length, and thus the buffer size is: 32 bytes, plus one byte per km of link and per Mbit/s of peak bandwidth.

### 2.2 Reservation of One Buffer per Virtual Circuit

When one VC encounters heavy traffic somewhere along its path, packets of that VC get concentrated and stay in nodes that precede that heavy-traffic region, taking up buffer space which thus becomes unavailable for other VC's; these other VC's then get slowed down, even though their own path may not pass through heavy-traffic areas. The remedies for this undesirable situation are to provide a lot of buffers or to prevent congestion. In the past, cost considerations and the software nature of protocol implementation forced network designers to opt for congestion prevention, in order to avoid buffer-hogging and deadlocks.
Today, VLSI memory technology reaches the point where it is feasible to preallocate one buffer exclusively to each and every virtual circuit, in each node through which it passes. Such a preallocation policy completely solves the problem of undesirable interactions between VC's, because it makes them independent of each other [Tane81, p.216], and thus congestion ceases to have negative effects. This policy means that each node must contain more buffers than the number of VC's that could “ever” simultaneously pass through it. Obviously, we would like to make this hard limit on the number of simultaneously active VC's as high as possible. The examples that follow show that allowing as many as thousands of VC's per node is well within the capabilities of the technology of today or of the near future. Also, note that similar hard limits on the maximum number of VC's already exist, in the form of the size of the routing table, or the number of bits in the VC identifier.

For the numerical example considered at the end of the previous sub-section, we can compute the requirements shown in table 2.1. The figures in the first example may refer e.g. to one thousand video virtual circuits. Those in the second example may refer e.g. to 65 thousand VC’s of high-performance data. The third example illustrates the following situation: if up to n VC's are allowed through a link of total capacity 1 Gbit/s, and if each of those VC’s is allowed to have a peak instantaneous data-rate ($BW_{peak}$) of one thousand times more than its average data-rate ($BW_{avg}$), then the incremental buffer-memory cost per kilometer of link is:

$$n \cdot BW_{peak} (8 \mu s/km) = 1000 \cdot n \cdot BW_{avg} (8 \mu s/km) \approx 1000 \cdot (1 \text{ Gb/s}) \cdot (8 \mu s/km) \approx 1 \text{ MByte/km}.$$  

<table>
<thead>
<tr>
<th>Virtual Circuits per Node</th>
<th>Buffer Memory per Node (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000 VC's, peak BW = 100 Mbit/s each</td>
<td>32 KB + 100 KB per kilometer</td>
</tr>
<tr>
<td>65,000 VC's, peak BW = 4 Mbit/s each</td>
<td>2 MB + 256 KB per kilometer</td>
</tr>
<tr>
<td>$n$ VC’s, $\frac{BW_{peak}}{BW_{avg}} = 1000, 1 \text{ Gb/s link}$</td>
<td>$32n + 1 \text{ MB per kilometer}$</td>
</tr>
</tbody>
</table>

Overall, we see that the buffer memory requirements per node for individual buffer preallocation do not exceed 10 memory chips (1 Mbit DRAM’s) originally plus 10 more for each kilometer of high-performance link (e.g. optical fiber). Section 3.4 will discuss cases where less than the above memory may suffice.
3. Allocation of Bandwidth

3.1 Bandwidth Allocation under Congestion

In the lack of congestion, the network does not have to make any bandwidth allocation decision; since the offered traffic is less than the network’s capacity, all the packets that are sent will eventually get-through. When congestion exists, the network has to make the BW allocation decisions, i.e. it has to arbitrate among all the VC’s that try to use more BW than there is of it. In that case, the multiplexing method is the primary means of making these decisions. For example, figure 3.1 illustrates First-In-First-Out (FIFO) multiplexing. Packets from A and from N1 arrive simultaneously at node N2, and they get interleaved in the output queue. If the BW of link OUT is the bottleneck, then FIFO multiplexing will probably allocate as much BW to A as to both B and C combined, which is probably undesirable.

![Figure 3.1: FIFO Multiplexing](image)

Besides multiplexing, the method of buffer-sharing (or the buffer-space per VC) also affects BW allocation. We will not use such methods for controlling the BW allocation, since, when the buffer space per VC consists of many packets, these methods give rise to long queues, long delays for packets in non-congested VC’s, and non-uniform distribution of the packets of each VC along the queue.

When the network has to make BW allocation decisions, a policy for making them has to be defined. The simplest reasonable policy is the equal treatment of all VC’s within a certain priority level. We will follow this policy in sections 3.2 and 3.3, and examine possible modifications of it in section 3.4.

3.2 The Circular Scan (Round-Robin) Method

At any given point in time, for each outgoing link, a number of VC’s in the node will be “ready” for transmission over that link. A VC is “ready” iff its buffer on the local node contains at least one packet and its buffer on the destination node is known to have empty space for at least one packet. The transmission of one packet from a certain VC corresponds to giving “one unit of communications
resource" (of BW) to this particular VC. If we are to equally allocate the available BW to all the VC's in a certain priority level, we should give one such unit to each one of the (ready) VC's, before giving a second unit to any one of them. Thus, multiplexing should be performed by a "Round-Robin" method — by circularly scanning all the VC's (of a given priority) and by transmitting one packet from each one of them that is found to be ready. Figure 3.2 illustrates this. It shows one outgoing link of a switching node N1, the buffers of 5 VC's that go through that link, and N1's idea of empty buffer space at the other end of the link. The marked VC's are currently ready for transmission. For full link utilization and no transmission gaps, the search for the next ready VC should be completed by the end of the present packet transmission time. Methods for implementing such fast circular scans will be presented in section 4.

![Diagram](image)

**Figure 3.2: Multiplexing by Circular Scan**

The aforementioned buffer reservation and packet forwarding methods have been used in "TYMNET", whose VC's are low-speed (one end of them is a terminal) [Tyme81, p. 395]. Also, a TRANSPAC early design called for fixed buffer reservation for every VC in every node, but this was abandoned [SiDa79].

In order to better understand how the circular scan method works, let us study figure 3.3. The horizontal axis in this figure is time, and 4 vertical lines have been drawn at the times when two particular VC's are scanned. The period T of this scanning is shown as constant, but in fact it will vary as the instantaneous load changes. The two VC's that are shown in the figure differ in one important respect. The first one, VC#m, is "congested": it receives packets and permits (it becomes ready), more frequently than it gets a chance to transmit a packet. This means that it tries to pass through more data than the link allows it to. For this VC, exactly one packet is transmitted per circular scan. The second virtual circuit, VC#n, is "non-congested": it becomes ready less frequently than it gets a chance to transmit a packet. This may be so because it receives packets less frequently, meaning that it is bandwidth-limited somewhere earlier along its path, or because it receives permits less frequently, meaning that it is bandwidth-limited somewhere further down along its path. One way or another, packets on this VC get
transmitted during the very first circular scan following the moment when they become ready to be transmitted. At this point, a clarification is in order: the terms “congested” and “non-congested” VC refer solely to the link under consideration: a VC which is non-congested on this link will be congested (i.e. bandwidth-limited) on some other link, or at its source, or at its destination (in traditional non-congested networks, all the VC's are bandwidth-limited at their source).

From the discussion above we can conclude the following facts:

- A VC which is non-congested on a particular link uses up all of the bandwidth that it needs on that link, since it gets to transmit a packet during the first circular scan following when it becomes ready for that.

- On each particular link, every non-congested VC uses up less bandwidth (on the average) than every congested one, since the latter transmits exactly one packet per circular scan while the former transmits less than that.

- All congested VC's on a particular link use up the same amount of bandwidth each, since they all transmit exactly one packet per circular scan.

- As long as there is at least one congested VC on a link, that link is continuously used (i.e. fully utilized), since there is always at least one ready VC from which to transmit a next packet.

Therefore:

- the method of circular scan equally distributes all the available link bandwidth to all the virtual circuits that can use it;

- if some virtual circuits do not need to or cannot use all of their share of bandwidth, then they get allocated as much BW as they need, and the remaining portion is equally distributed to all the other VC's that can use it.

*Figure 3.3: Circular Scan of Congested and Non-Congested VC's*
Figure 3.4 shows an example of such bandwidth allocation. The squares represent data sources and destinations, with six VC's set up among them. The numbers in circles represent maximum bandwidth capacity; notice that these are defined for links as well as for source and destination stations. The numbers that are not inside circles represent actual bandwidth. Five VC's pass through the link of capacity 50, so if all of them were congested on that link each of them would get 10 units of BW. However, VC D→D is congested (bandwidth-limited) at its destination; this terminal station, by means of flow-control, restricts the VC to using only 5 units of BW throughout its path. The remaining 45 BW units on the above link would get equally allocated to the other 4 VC's that pass through it (11.25 to each). However, VC's B→B and C→C are limited to using only 10 BW units each, because they share the link of capacity 30 equally with VC A→A. Eventually, 25 BW units remain on the main link, to be used by VC's E→E and F→F, which thus get 12.5 units each.

![Diagram of bandwidth allocation]

Figure 3.4: Example of Bandwidth Allocation

During transient periods, bandwidth allocation is not as above. Consider a VC (say D→D in figure 3.4) which has just been set up, and which will turn out to be bandwidth-limited to $BW_{lim}$ at some place $P$ along its path. When this VC starts transmitting data, it will use more bandwidth than $BW_{lim}$ along the portion of its path before $P$, until the buffers along that portion of the path get filled up.

3.3 Packet Delay through Non-Congested VC's

The discussion in the previous sub-section suggests the introduction of the notion of "bandwidth-share", $BW_{sh}$, on a particular link:

$$BW_{sh} = \frac{BW_{link}}{N_{scan}}$$
where $BW_{\text{link}}$ is the total link bandwidth, and $N_{\text{scan}}$ is the number of ready VC's that the circular scan at the entrance to the link encounters during one of its cycles. $BW_{sh}$ is equal to the bandwidth given to each VC that is congested on this particular link. If $p$ is the packet size, then the period of the circular scan is $T_{\text{scan}} = N_{\text{scan}} \times (p / BW_{\text{link}}) = p / BW_{sh}$, and thus the data-rate on a congested VC is $p / T_{\text{scan}} = BW_{sh}$.

On each particular link, those VC's with bandwidth less than $BW_{sh}$ are non-congested. Under steady state, the bandwidth of a VC is the same throughout its path. Thus, under steady state, a VC is congested only on the link(s) which has (have) the minimum value of $BW_{sh}$ among all the links on its path, including the source station and the destination station. In all the nodes before the (first) congested link along the path of a VC, the buffers of the VC are "almost" full, because they are filled more frequently than they are emptied. In all the nodes after the (last) congested link, the buffers are "almost" empty, because they are emptied more frequently than they are filled. Following the traditional notion of congestion, we will call "non-congested VC" (in general — not over a particular link) a virtual circuit which is congested (i.e. bandwidth-limited) at its source station. For such a VC, the source station determines the data-rate, and that rate is less than the minimum $BW_{sh}$ of all the links through which the VC passes. All its buffers are "almost" empty, since they all lie after the last point of congestion (the source). Specifically, for "continuous-time behavior", each of these buffers may contain at most one packet at any time. In our case, some buffers may at some times contain more than one packet, because of "quantized-time" effects among the various asynchronous circular scans (of different period each). Without loss of generality, our analysis assumes at most one packet in every buffer.

For congested VC's, it makes no sense to talk about the delay of packets, since their data source produces packets at a rate higher than the network transports them; thus, the packets that come later in the transmitted "message" will suffer increasingly longer delays relative to those that came earlier. For such a congested VC, the meaningful performance figure is the effective bandwidth which it gets, and that is equal to the minimum $BW_{sh}$ over the links of its path.

For non-congested VC's, on the other hand, the important performance measure is the total delay $t_{d,\text{det}}$ for a packet to go through the network. This delay is:

$$t_{d,\text{det}} = t_{\text{prop}} + \sum_{i=1}^{N_{\text{sw}}} t_{\text{sw},i}$$

where $t_{\text{prop}}$ is the total propagation delay through the VC, $t_{\text{sw},i}$ is the switching (multiplexing) delay through the node at the entrance to link $i$, and $i$ in the summation ranges through all the links that constitute the VC (except the one from the source to the first node). Since packets in non-congested VC's are forwarded during the first circular scan following their arrival, it follows that each switching delay is not more than the period $T_{\text{scan},i} = p / BW_{sh,i}$ of the corresponding circular scan.
\[ t_{d,tot} \leq t_{prop} + \sum_{i=1}^{N_{\text{links}}} t_{\text{scan},i} = t_{prop} + \sum_{i=1}^{N_{\text{links}}} \frac{p}{BW_{sh,i}} \]

Now, using the fact that \( \sum_{i=1}^{N} 1/a_i \leq N/\min\{a_i\} \), we get:

\[ t_{d,tot} \leq t_{prop} + N_{\text{links}} \cdot \frac{P}{BW_{sh,min}} \]

where \( BW_{sh,min} \) is the minimum of the bandwidth-shares among the VC's links.

This result is important for two reasons. Firstly, it is different from the traditional statistical queueing delay (although this one also has a statistical component \( N_{\text{scan}} \) is a random variable). Secondly, this result holds true for an important subset of the VC's of a congested network, and it is very important to be able to guarantee a maximum packet delay even in the presence of congestion in other VC's.

Consider now the following example: Let \( P = 256 \) bits, and \( N_{\text{links}} = 20 \). Imagine that the worst of the links through which this VC passes, is a 1 Gbit/s fiber that carries 1000 voice VC's and 54 video VC's. Setting up that many VC's has been allowed by estimating their average BW (given intelligent encoding) as follows: \( 1000 \times 25K\text{bits/s} + 54 \times 18M\text{bits/s} = 25Mb/s + 972Mb/s = 997Mb/s \).

Now, imagine that all of a sudden all these VC's become very busy and try to use their peak BW: \( 1000 \times 64Kb/s + 54 \times 40Mb/s = 64Mb/s + 2160Mb/s = 2224Mb/s \). All the video VC's (and only them) will become congested, and BW will get allocated as follows: the 1000 voice VC's will each get 64Kb/s (i.e. as much as they want), and the 54 video VC's will equally share the remaining \( (1000-54) = 936 \) Mbits/s: \( 936 \text{Mbits/s} / 54 = 17.3 \text{Mb/s} \) for each video VC. According to the terminology above, this means that \( BW_{sh,min} = 17.3 \text{Mb/s} \). Thus, the total delay of the voice packets getting through the net will be:

\[ t_{d,tot} \leq t_{prop} + 20 \times \frac{256\text{bits}}{17.3\text{Mbits/s}} \approx t_{prop} + 300\mu\text{s} \]

In this example only voice and video traffic was assumed, but the situation would be entirely analogous if some of the voice VC's were actually carrying interactive data, or the video VC's were actually performing bulk file transfers. Notice that the upper bound for the switching delay obtained above is about 50 times less than what is acceptable for voice traffic, or about 100 times less than the average delay of a block access on the local hard disk of a computer system.

### 3.4 Overall Policies, Comments, Modifications

In a large Integrated Services Data Network, allocating bandwidth and guaranteeing a level of maximum delay for all the different kinds of traffic in it (e.g. voice, interactive data, video, bulk file transfers), and deciding what to do in
cases of conflict, is not a simple matter. The method of multiplexing by circular scan should be seen as one of the mechanisms available to the network designer for implementing an overall policy of bandwidth allocation. Such policies are not the topic of this paper, but it is useful to mention some of the mechanisms that may be used for achieving them, namely:

- **Bandwidth Quota** per VC. This may be enforced at the source, or using the size of buffers that are preallocated to the VC (section 2.1).

- **Refusal to Set-Up** a new virtual circuit.

- **Priority Levels** for various VC's. When this scheme is combined with circular scans, a separate scan should be defined for the VC's of each priority level.

- **Weighted Circular Scan.** If certain VC's are visited more than once during each cycle, then they will get a proportionately higher share of the bandwidth.

- **Bit Dropping.**

Thus, there is a wealth of mechanisms that can be combined with the circular scan method for implementing overall bandwidth allocation policies. Of course, not all of them necessarily have to be included in a network. For example, using the numbers of the previous sub-section, we see that there may be cases where some VC's are guaranteed to be non-congested (assuming refusal to set-up too many VC's). In that example, the circular scan method automatically guarantees some kind of “high priority” for voice VC's, since their packets get forwarded in less than about \( \frac{256 \text{ bits}}{16 \text{ Mbits/s}} = 16 \mu s \). Additionally, as noted in section 3.2, the buffers of the VC's that are not congested (that are bandwidth-limited at their source), remain “almost empty”, because their packets are forwarded more frequently than they arrive. Thus, for this kind of VC's, flow-control is not necessary, hence permits and buffer pre-allocation are not needed. For another example, assume that some VC's carry real-time traffic that can tolerate lost data; then again, permits may not be needed, and flow-control may be done by dropping packets (however, buffer pre-allocation would be needed).

Another possible modification of the basic architecture is the temporary merging of multiple virtual circuits into one, over a common portion of their path. At the entrance to the merged VC, packets should be multiplexed into it using a circular scan of the VC's that are merged; the corresponding permits have to originate from the exit of the merged VC, that is more than one links away. Another level of permits, referring to the merged VC itself, has to be used from link to link along the common path. An example application will be seen in section 5.4. This technique may be useful within “switching centers” composed of multiple switching nodes: the number of VC’s, and consequently the number of buffers in the internal nodes may be reduced by merging together, throughout the switching center, the VC’s that come into it and get out of it on the same pair of links.
4. Implementations of the Circular Scan

In the previous section, we saw that the circular scanning of all VC's, in search for the "ready" ones, is the most natural method for automatic and fair allocation of the BW of congested links. In this section we will discuss two exact and one approximate hardware implementations of this method. All of them are realistic, but the approximate one is the most attractive. In the next section, one more, distributed implementation will be examined. It is crucial that the implementation be in hardware, because the time to find the next ready VC, while the previous packet is being transmitted, may be as short as 250 ns for 1 Gbit/s links and 256-bit packets.

4.1 Exact Implementation 1: Priority Encoder

The essence of the circular scan operation can be formulated as follows: given a (circular) array of flag bits, and a starting position in it, find the next bit (circularly after that position) which is ON. In our case, each flag is the boolean AND of two other bits that describe the status (non-empty) of the local VC buffer and the status (non-full) of the remote VC buffer.

The above is the definition of a circular priority encoder. Figure 4.1 shows a straightforward hardware implementation for it, that is like a token ring: a token is introduced at the starting position \( i_0 \), and it is allowed to propagate until it encounters the first non-zero flag bit. At most one of the outputs can be non-zero, and that identifies the next VC to send a packet from. For large priority encoders, with \( N \) = thousands of inputs, like the ones we are interested in, the token propagation delay through the ring is too long. The solution then is to use a token look-ahead scheme, similar to the carry look-ahead for adders. This is based on forming the \( N/n \) OR functions of the inputs taken in blocks of \( n \); then, using a priority encoder of size \( N/n \) (thus \( n \) times faster), one can find the next block of \( n \) inputs which has a flag bit ON. Applying this recursively, a tree of look-ahead circuitry is built and the total delay is made proportional to \( \log_n N \).

A realistic circular-scan chip would include, besides the priority encoder, the latches to hold the buffer-status bits, and it would be arranged as a two-dimensional array. Figure 4.2 shows the block structure of such a chip. The priority encoding operation can now be performed in steps: (i) the flags on each row raise wired-OR horizontal buses, and the row priority encoder along the vertical edge of the array selects the next row that contains at least one true flag; (ii) the selected row is read out through vertical buses, and the column priority encoder along the horizontal edge of the array finds the next true flag in it. Note, however, that a slight complication has to be resolved: before step (i), the row that contains the starting index \( i_0 \) has to be read out and checked for a possible next true flag within itself; if none is found, then we proceed with steps (i) and (ii).
Figure 4.1: Combinational Priority Encoder

\[ \text{out}_i = \text{token}_{i-1} \land \text{flag}_i \]
\[ \text{token}_i = \text{token}_{i-1} \land \neg \text{flag}_i \]

Figure 4.2: Block Structure of Circular-Scan Chip
4.2 Exact Implementation 2: Hardware Tree

A different method to get a large and fast priority encoder is to implement in hardware a manipulation algorithm for a tree data-structure. Figure 4.3 shows an example of a tree that can be searched for the next "1" leaf, or updated on insertions or deletions of "1" leaves, all in logarithmic time in the number of leaves (size of the tree). The leaf nodes contain the flag bits that were defined above. Each internal node contains the boolean OR of its child nodes. The example of figure 4.3 shows a 3-way tree for \( N = 3^3 = 27 \) VC's; the nodes and the leaves of the tree can be stored in three memories, M0, M1, and M2 (one per tree level), which can be organized in 3-bit words. Since real hardware systems can handle 16-bit or 32-bit words, such real systems can be implemented with 16-way or 32-way trees. These will have a height of only 3 or 4 levels for some hundred thousand flag bits.

![Figure 4.3: A Tree for Finding the Next "1" Leaf](image)

The basic idea of the search is that, in order to find the leftmost "1" leaf, we traverse the tree from the root downwards, at each level selecting (with a priority encoder) the leftmost "1", and then visiting the children of that node and again selecting the leftmost "1" of them. The full search algorithm, to find the leftmost "1" leaf after a starting position \( i_0 \), is slightly more complicated, requiring first an upwards and then a downwards tree traversal: Start from the word that contains the leaf \( i_0 \) and read its ancestor words, until a word is found that contains a "1" to the right of the "1" which is the ancestor of \( i_0 \). At that point, start traversing downwards the subtree rooted at that "1", looking for leftmost "1's" in the children, as before. Inserting a new "1" leaf is easy: we just blindly set to "1" all the flip-flops along the "ancestry chain" of the desired leaf. Deleting a "1" leaf requires a little more work: we read the word that contains the bit to be deleted, modify it to clear the bit, and simultaneously check to see if the "1" that we deleted was the last "1" in that word. If it was, then its parent bit also has to be
cleared in a similar way, and so on upwards the tree.

We just saw that insertions, deletions, and searches can all be done in a number of steps equal to once or twice the height of the tree; we will see below modifications of the algorithms which reduce the number of steps to once the height of the tree, in all cases. We also mentioned that the height of real trees will be about 3 or 4. The complexity of each of those 3 or 4 steps is approximately that of one memory cycle plus one priority encoding of 16 to 32 inputs. The situation may still be a little tight: the following may have to be done per transmitted packet: (1) the VC becomes ready (insert it into the tree), (2) search and find it, and (3) the VC becomes non-ready (delete it from the tree). Then, about 10 cycles per packet will be needed for tree operations, that is the cycle time should be as low as 25ns if the packet time is to be 250ns (for 256-bit packets at 1Gbit/s).

In cases where the cycle-time requirements of the above mode of operation are too tight, one can go to a pipelined design. The basic idea of this is to have one pipeline stage per tree level. The memories that hold the nodes of each level have to be physically separate, lying one in each pipeline stage. In order for data to flow through the pipeline in a regular way, we would like all operations to traverse the tree in the same direction — either all from root to leaves, or all from leaves to root. The tree operations were not formulated in this way above, but a reformulation is possible so that all operations traverse the tree downwards. The basic idea is that, during the downwards phase of a search, information is also collected and kept, to assist (i) in the search for the next “1”, and/or (ii) in a possible subsequent deletion of the “1” that is found. (Notice that VC’s become non-ready, and have to be deleted from the tree, only right after the transmission of one of their packets). The additional information that has to be collected at each level during a downwards search is: (i) whether there is another “1” to the right of that which is found, and (ii) whether there is another “1” at all in the word.

4.3 Approximate Implementation: Linked List of Ready VC’s

The two circular scans examined so far have the following properties:

(i) they visit each and every VC exactly once per cycle of the scan; and

(ii) they always visit the VC’s in the same order.

Property (i) is essential for guaranteeing the fairness of BW allocation. Property (ii) however is not so important, and thus we may drop it. The data structure which can then be used for implementing the scan is a linked list of all the ready VC’s, as illustrated in figure 4.4. Notice that this list contains only ready VC’s, and that it contains them in random order — requiring a non-random order would preclude updates in constant time. The list is traversed (scanned) repeatedly (circularly), and exactly one packet is transmitted from every VC that is visited. As a result of such transmission, some VC’s may become non-ready, because they run out of packets or permits, and they have to be removed from the linked list. On
the other hand, arriving packets and permits may cause non-ready VC's to become ready, and these have to be inserted into the list. (If the arriving packets or permits are destined to VC's that already are ready, no change to the list is required).

![Diagram of list of ready VC's with arrows indicating movement between VC's and a section labeled non-ready VC's.](image)

*Figure 4.4: Linked List of Ready VC's*

The linked list can be implemented in hardware with a memory $M[ ]$ containing as many words as the maximum number of VC's, where each word is a pointer to (the index of) the next VC; three external registers should contain pointers to the beginning of the List, to the Current VC, and to the Previous one. Finding the next ready VC can be done with one memory read: $\text{Previous} := \text{Current}$, $\text{Current} := M[\text{Current}]$. Finding the next and deleting the current VC can be done with one additional memory write: $M[\text{Previous}] := M[\text{Current}]$. Inserting a new VC before the current one can be done with two memory write's: $M[\text{Previous}] := \text{New}$, $M[\text{New}] := \text{Current}$.

When a new VC has to be inserted into the list of ready ones, the crucial question is where to insert it. There are four basic choices:

(i) after the “current” VC;

(ii) before the “current” VC;

(iii) after the “end” of the list; and

(iv) before the “beginning” of the list.

Choice (i) is the most favorable for the newly-ready VC. Choice (ii) is the least favorable for it. Choices (iii) and (iv) are similar, since the list is circular, but in case (iii) the insertion point moves forward, while in case (iv) it moves backwards. Intuitively, insertions (iii) and (iv) will give results somewhere between those of cases (i) and (ii).

An important point is that in (i) and (iii), the position of insertions moves forward as the insertions proceed. This can lead to a dangerous race between insertions and scanning, as illustrated in figure 4.5. If the arrivals of packets on some
VC or VC's happen to get "synchronized" with the scanning, then that or those VC's will monopolize the outgoing link's bandwidth. In figure 4.5, the last packet of VC#i is transmitted, the VC is removed from the list, but a new packet for that same VC happens to arrive soon thereafter, and the VC gets re-inserted at the end of the list, right in time to be resersved by the scanning process, and so on.

Because of the race problem, insertions (i) and (iii) are not recommended. Insertions (i) are still very useful, but for a different purpose: implementing higher priority for the VC's to which they are applied. In connection to this, note that, under steady state circumstances, the VC's which temporarily become non-ready and then ready again are precisely the ones which are not congested on the local outgoing link (i.e. they are congested elsewhere). For these VC's, their period of "becoming ready" is longer than the period of the local circular scan (§ 3.2), and thus they couldn't sustain a race like the one described above. Unfortunately, a race can very easily be triggered by transient circumstances on locally congested VC's. Still, insertions of type (i) can be used for all VC's, if one takes specific measures, like e.g. recording, for each VC, the number of the last circular scan during which it was serviced, and comparing that, upon insertions, to the number of the current scan. This will give short switching delays to the VC's on the links where they are not congested.

If all VC's are to be treated equally, and the "measures" just mentioned are not implemented, insertions should only be made before the "current" VC (method (ii)), or before the "beginning" of the list (method (iv)). Insertions (ii) are like traditional insertions at the end of a queue. They guarantee that between two consecutive servicings of a VC that remains ready, or between the time the VC becomes ready and the time when it is first serviced after that, no other VC will ever be serviced more than once. Insertions (iv) guarantee a weaker property: between the same two events as above, no other VC will ever be serviced more than twice, while for continuously ready VC's, in between n consecutive servicings, no other VC will ever be serviced more than n times (not 2n). With insertions (iv) there is no danger of races because the position of insertions moves backwards — i.e. this scheme has a LIFO flavor, as opposed to the FIFO behavior of (ii).
In summary, maintaining a linked list of all ready VC's is a good approximation of the circular scan algorithm, which is easy to implement in hardware, and which can easily be adapted to various performance policies. Thus, it is the most attractive of the implementations presented in this section.

5. Switching Node Structure

In this section we combine the issues of buffer memory and of multiplexing algorithm, to achieve a practical switching node design, that has high throughput, implements circular-scan multiplexing, lends itself well to planar layout for integration, and can be used as building block for large networks.

5.1 Switching Nodes as Network Building Blocks

To build a large system, it is desirable to use replicated identical building blocks. For data networks, the natural building blocks are switching nodes that can be interconnected directly to each other to form almost arbitrary network topologies. For VLSI implementation, it is preferable that these nodes have relatively small size — e.g. 3 to 32 inputs and outputs [Fuji83]. Figure 5.1(a) shows the basics of such a node: it has $M$ inputs, $N$ outputs, and internally it has to contain at least the buffer memory, the various tables of VC's (e.g. routing table, list of ready VC's), and the hardware which scans the ready VC's and dispatches packets from them to the outputs.

The node structure of figure 5.1(a) is the slowest and least expensive. Space-switching occurs by time-multiplexing on the memory bus. The peak buffer-memory bandwidth is $(M + N) \cdot BW_{\text{link}}$. If the buffer-memory is a 32-bit-wide array of DRAM chips, with an average access rate of as much as one word every 32 ns (using nibble- or page-mode accesses), the memory BW is 1 Gbit/s. Then, even if $M = N = 4$ (only), the maximum bandwidth of a link can only be $BW_{\text{link}} = 125 Mbits/s$. If one wants to go to higher link bandwidths or to more links per node, alternative designs must be investigated.

The design of figure 5.1(b) uses $N$ copies of the hardware of design (a); each copy only deals with those incoming packets that are destined for one specific outgoing link. The peak buffer-memory BW in this scheme is $(M + 1) \cdot BW_{\text{link}}$. Notice that this is much higher than the average buffer-memory BW, which is $2 \cdot BW_{\text{link}}$. The design of figure 5.1(c) relaxes the peak BW requirement on the buffer-memory and makes it $2 \cdot BW_{\text{link}}$. The speed requirement on the multiplex hardware is quite high, but in the next two sub-sections we will see that there are practical and inexpensive methods for meeting that.

The design of figure 5.1(d) is a less expensive version of design (c), but there are cases where it is not able to keep all the outgoing links busy. In this design, all or the majority of the buffer memory is "pulled" out of the multiplex array, to
the left. This is desirable because the buffer memory is large relative to the multiplex array, and thus it is better for layout reasons and for circuit compactness (and consequently speed) to keep that memory out of the array. This design is basically the dual of (b): in order to be able to sustain peak output traffic, the horizontal buses and the memories that feed them should have a capacity of $N \cdot BW_{\text{link}}$, since all $N$ outgoing links might decide at once to read their packets from the same input. Now, if one incorporates little buffer memories in the inputs of the multiplex units, then one can relax the BW requirements to something closer to the average values of $BW_{\text{link}}$ for the buses and $2 \cdot BW_{\text{link}}$ for the large buffer.
memories. It is easier to make this compromise in design (d) rather than in its
dual (b), because whenever the instantaneous traffic requirements exceed the bus
and memory capacities, all that happens in (d) is inefficient utilization of the out-
going link bandwidth, while in (b) packets have to be dropped. In what follows,
we adopt the general structure of design (d).

5.2 Circular Scanning by Merge Sorting

We consider now how to implement the circular scanner of all the VC's that
have to be multiplexed onto the same outgoing link. This scanner has to deal with
VC's that come in on multiple links, and thus the status (ready or not) of many
VC's may change simultaneously, as packets come in from all inputs. Also, from a
layout point of view, this scanner has to deal with information that is topologi-
cally scattered. We will see a way to decompose the circular scanner and multi-
plexer into an array of identical cells, each one of them dealing with only one
incoming link, and all of them operating in parallel — in the style of a systolic
array.

The circular scan algorithm, in its exact implementation (§ 4.1, 4.2), produces
sequences of packets from different VC's, in the fixed order in which VC's appear
in the cycle of the scanning process. If we associate with each VC a numerical
identifier that increases in that order, then the packets will form a sequence that is
sorted (ordered) by increasing identifier. After one sorted sequence is produced and
sent out, a new one, consisting of packets from the now-ready VC's, is produced,
and so on. Figure 5.2 shows an array of processing elements that can produce such
sorted sequences. For the VC's of each incoming link, an independent circular scan
is performed, yielding a sorted sub-sequence of packets. Then, the vertical chain of
elements merges these sub-sequences into a single sorted sequence. This is done by
the algorithm of merge-sorting: each element looks at the two next packets on its
two inputs, chooses the one with the smaller identifier, and forwards it to its out-
put. Of course, provisions must be made to recognize the end of one sorted
sequence and the beginning of another.

In figure 5.2, merge sorting is done using a linear array of elements. It could
also be done using a binary tree of such elements, in which case most packets
would pass through a smaller number of switching elements. However, the VLSI
layout would be harder.

5.3 Multiplexing by Cycle Matching

In section 4.3, it was noted that the order of the VC's in the circular scan is
not essential — what is important is that each VC gets a chance to send a packet
exactly once per cycle of the scan. Thus, in merging the packets by the method of
the previous sub-section, sorting is not essential — the important thing is to match
the beginnings and the ends of the "sequences" (cycles). Figure 5.3 illustrates this.
Figure 5.2: Circular Scanning by Merge Sorting

The first packet of each cycle of a circular scan is marked with some marker. The multiplexing element merges two streams into one, selecting packets “at random” except that it has to match packets that are marked as beginnings of new cycles. Since the cycles on the incoming streams contain at most one packet from each VC, the cycles on the outgoing stream are also guaranteed to contain at most one packet from each VC.

When incorporating the above multiplexing elements (in the form of a chain as in fig. 5.2) into the design of figure 5.1(d), attention has to be paid to two points. First, when a multiplexing element encounters a marked packet on one of its inputs while no packet remains on its second input, it should not wait until it
Figure 5.3: Multiplexing by Cycle Matching

sees a marked packet on that second input, since the second input has nothing to send. Second, no input should be allowed to introduce too many cycles of packets into the multiplexing chain, before some of the older cycles are completely out of the chain. The reason is as follows. Suppose that input $A$ once has very short cycles, e.g. consisting of a single VC each. Say that $A$ pumps 5 packets into the chain; all of them are marked, and they correspond to 5 cycles. Now, if input $B$, located somewhere further down the chain, has very long cycles, it will be quite some time until the packets of 5 cycles have all been sent out of the chain. If, in the meantime, a new VC in $A$ becomes ready, the packet from that new VC will have to wait 5 (long) cycles before it is sent out, because 5 marked packets from $A$ are ahead of it. The conclusion is that the individual scanning processes at the individual inputs should not be allowed to proceed too much ahead of the equivalent overall "scanning" that the chain of multiplexing elements performs.
5.4 Organization of the Switching Node and of the Multiplexing Elements

Figure 5.4 shows the switching node organization that results from the application of the above merging methods in the structure of figure 5.1(d). On the left, there are $M=4$ input buffer units, one for each incoming link. To their right, there is a rectangular array of multiplexing elements that resembles a systolic array. Each one of these (identical) elements consists of two small FIFO buffers and some simple selection logic to perform the cycle matching.

![Diagram of the switching node and multiplexing elements]

**Figure 5.4: Organization of the Switching Node and of the Multiplexing Elements**

The multiplexing elements communicate among themselves and with the input buffer units by means of FIFO buffers and permits. The input buffer units
receive two levels of permits: B2, for example, receives permits both from the element M1 and from the buffers at the other end of link L1 (as well as from M0 and L0, and from M2 and L2). Permits from the link L1 refer to individual VC's, whereas permits from the element M1 concern all those VC's that go out on link L1. This means that two levels of scanning have to exist within B2. On the first level, the 3 groups of VC's are circularly scanned, to find the next group for which the corresponding multiplex element (M0, M1, or M2) is ready to accept a packet. Then, the VC's of that group are scanned (starting right after the one that was serviced last), to find the next ready VC and to transmit its next packet to the multiplexing element. This is a special case of the technique mentioned at the end of section 3.5. The two levels of scanning can be implemented e.g. by maintaining a separate linked list of ready VC's for each group of VC's that go off the same link.

It was mentioned in section 5.1 that the design of figure 5.1(d), which is the same as that of figure 5.4, has the problem that if suddenly all outgoing links happen to expect packets from the same input buffer unit, say B2, that unit will not be able to satisfy all of them at once (as opposed to the design of figure 5.1(c)). With respect to this problem, the cycle-matching method (§ 5.3) is better than the merge-sorting method (§ 5.2), in addition to also being simpler. The reason is as follows. Since with cycle-matching M1 is free to select packets at random (within one cycle), it could give some priority to packets coming from B2. When these packets get depleted (because B2 cannot supply them as fast as M0, M1, and M2 can consume them), then M1 can select packets from M1'. With merge-sorting, M1 could be forced to wait for a packet from B2 just so as to maintain the proper packet order.

The connection of the multiplexing elements in a linear chain, as shown in figures 5.2 through 5.4, may lead to long switching delays for the packets that come in through the top input buffer units, because they have to go through more FIFO queues. As noted at the end of section 5.2, for switching nodes with a lot of inputs, it may be better to connect the multiplexing elements in a tree topology. Because a binary tree is awkward to be laid out on the plane, an intermediate solution is a shallow multi-way tree, whose nodes are implemented as multiple-input elements or as linear chains of two-input elements.

5.5 Possibilities for VLSI and WSI Implementation

The switching-node organization of figure 5.4 is ideal for implementation with modern technology. The input buffer units are relatively large, due to the large buffer memory which they contain, but they have a single input and a single output (plus secondary ones for the permits). They don't need to be physically located right next to the array of multiplexing elements — they may even be connected to it by bit-serial links.

The array of multiplexing elements, on the other hand, can be implemented in a very compact and elegant way. The individual elements are very small, since
they only need to contain a few packets' worth of buffer memory (few kilobits) and some very simple logic. The array, with its rectangular wiring and with its high percentage of local communications, is ideal for layout on Printed-Circuit Boards or using Wafer-Scale Integration (WSI). Since the elements are so small, the entire array can be made very compact and thus very fast. In particular, the techniques of Soft-Configurable WSI presented in [KaBi85] are ideally suited to this application. The buses and switches that are provided for wafer configuration, are also used for routing, pipelining, and flow-control of data. Using the results of that work, we can estimate that a single wafer will suffice for integrating the multiplexing array of a $16 \times 16$ (or even $32 \times 32$) switching node, with a bandwidth of $1\text{Gbit/s}$ on each one of these I/O links. It is anticipated that such a wafer will use internally 32-bit-wide buses, operated with a cycle-time of 25 to 30 ns ($\text{BW} \approx 1.1 \text{ to } 1.3 \text{ Gb/s}$), and that each multiplexing element with its associated bus lines will occupy an area less than $2 \times 2 \text{ mm}^2$.

Conclusions

The tradeoffs of modern VLSI technology are such that buffer memory is no more a scarce resource in networks. This makes it possible to design networks in which some virtual circuits utilize all the bandwidth that remains available after the needs of the other circuits have been satisfied. Although the former virtual circuits are "congested", the total packet delay through the latter ones is still guaranteed to be less than a very reasonable upper limit. The network can allocate bandwidth to the congested virtual circuits in a fair way, using an exact or an approximate circular-scan method. This can be implemented by simple, special-purpose hardware. In the case of high-performance switching nodes, this hardware can take the form of a compact and fast rectangular structure (similar to a systolic array) where multiplexing occurs in a distributed and parallel fashion. This structure is ideal for layout on Printed-Circuit Board or with Wafer-Scale Integration. We expect that further study of this new and simple switching architecture — especially of the options for insertions into the list of ready VC’s, and of multiplexing by cycle-matching — will show its advantages over the more traditional architectures, stemming from the exploitation of modern VLSI technology.

Acknowledgements

Many thanks are due to Christos Nikolaou, who suggested the approximate implementation of the circular scan by a linked list of the ready VC’s (§ 4.3), and to Tony Ephremides for his valuable suggestions. This research was supported by the Computer Science Institute of the Research Center of Crete, Greece.
References


