Virtual Channels

Lecture #8: Monday, April 30 2001
Lecturer: Prof. William Dally
Scribe: Arjun Singh,Mohamed Ali Kilani,Kiran Goyal
Reviewer: Kelly Shaw

1 Flow Control

Suppose we have a message that needs to be routed from a source node S to a destination node D and S is connected to D as shown in figure 1. The flow control method must reserve resources for the message to be routed. Resources may be channel bandwidth and/or buffers.

In order to load balance across alternate routes, we may break up messages into packets and send some of the packets across each of the alternate paths. At each node, there are control states which also need to be reserved by packets. Packets are further

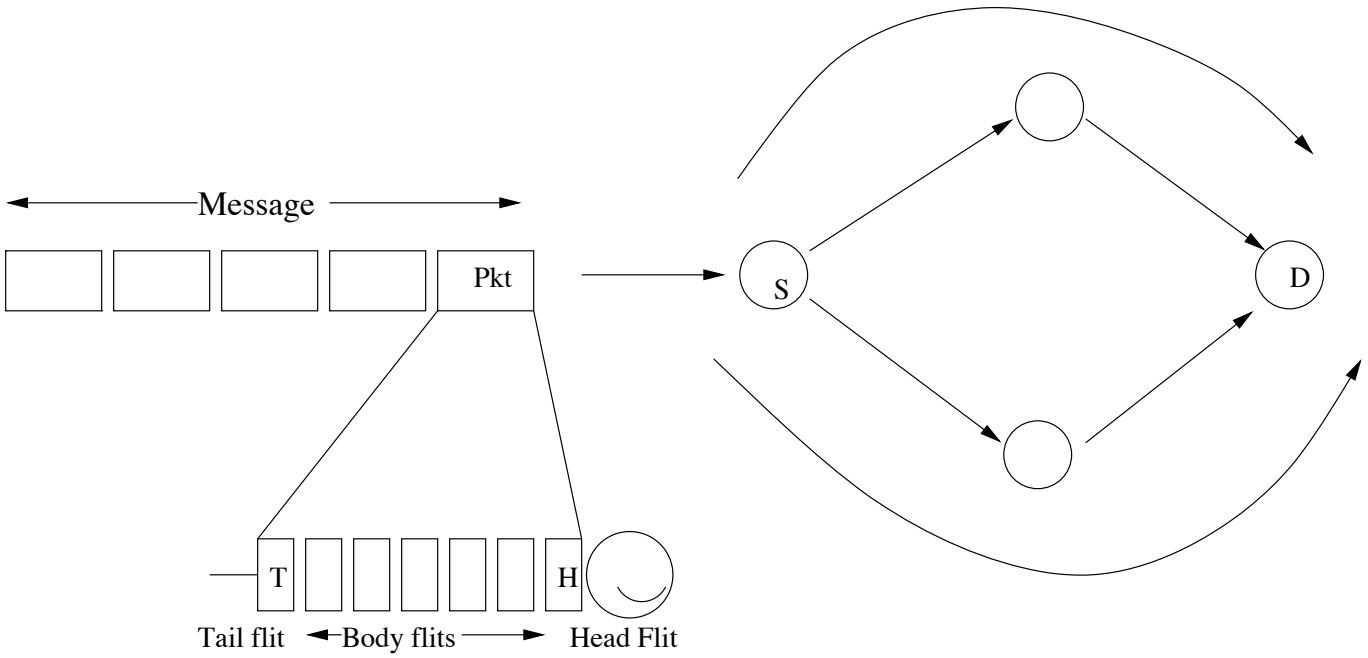

Figure 1: The message, packet, flit hierarchy

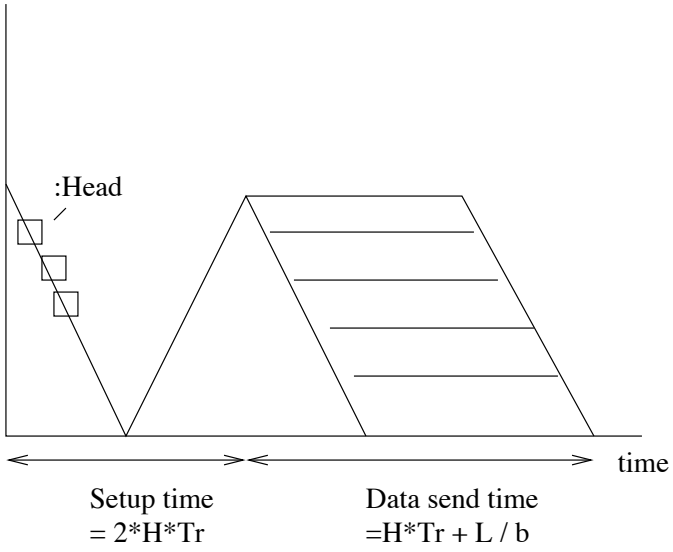

Figure 2: Time line for circuit switching

subdivided into flits(head flit, data flits and a tail flit). Packets have sequence numbers and routing information so they can take different paths. However, flits don't have any routing information or sequence numbers except the virtual channel id which is assigned as its route is set up. The tail flit deallocates the resources that were allocated by the head flit.

1.1 Circuit switching flow control

There are no buffers in circuit switching. The head is sent all the way to the destination and then an acknowledgment is sent back and then the data is sent in the route reserved by the head. However,circuit switching takes a greater time in setting up the route than in other flow control schemes (an overhead of 2*H*Tr minimum if the circuit is established in the first attempt),where Tr is the router delay and H is the hop count. See figure 2 for the exact picture. Moreover, in circuit switching, blocking can occur and a lot of resources are wasted(especially during the set up phase).

1.2 Store and forward flow control

In this scheme, the entire packet is sent from one hop to the next instead of sending flits. The disadvantage is that the latency is H*(L/b). For a good flow control scheme we should be summing the H component and the L/b component; here they are multiplied. This means for very long routes we will be incurring a lot of delay for a message transfer since we must pay this overhead every time a packet is sent.

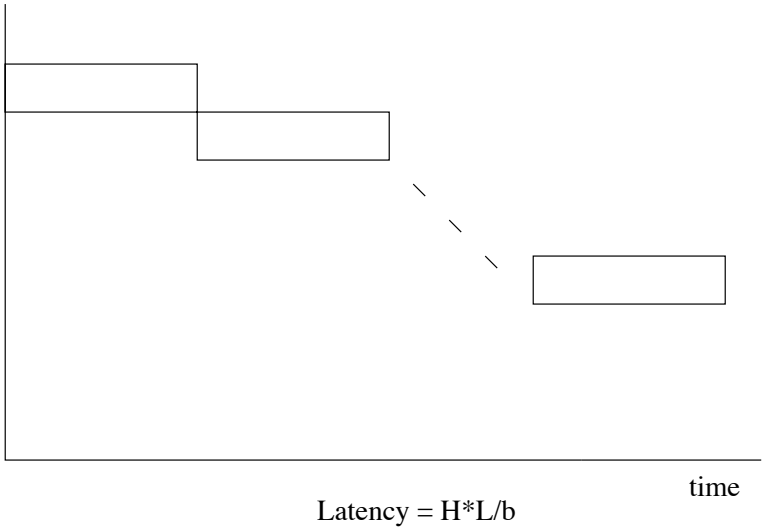

Figure 3: Time line for Store and forward

1.3 Virtual Cut through flow control

A possible solution to the store and forward latency problem is to start sending the packet as soon as the header information is received. We still need to buffer the whole packet at the hop but we don't need to wait for the tail before we start forwarding the packet. A disadvantage to this approach is that an error in the packet will only be detected when the CRC is received at the very end and so we might end up propagating an error all the way to the destination. Moreover, a channel is held for the entire length of the packet.

1.4 WormHole flow control

The difference between wormhole and the previous methods is that in the wormhole scheme we break up packets into flits, ie. we do not buffer the packet in its entirety at a given switch but rather flits. The advantage is that now a second packet does not have to wait for the entire first packet to be transferred before it can be forwarded across a given channel. Another difference is that in store and forward and virtual cut through, the buffer size is large (equal to the packet size) , but in wormhole we can restrict that to be a few flit size big. The disadvantage of wormhole is that, in effect it blocks the resources for a certain packet over many nodes (from head flit to tail flit). This means we may end up with channels that are idling even if some other packets' flits could be sent on them.

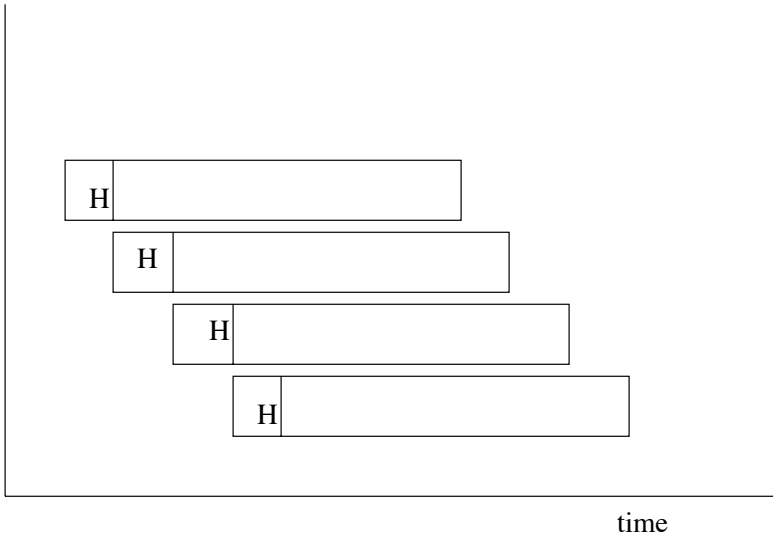

Figure 4: Time line for virtual cut through

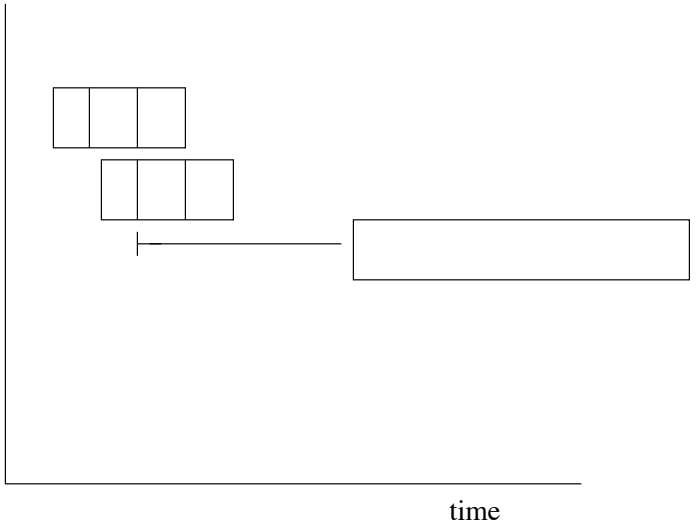

Figure 5: Time line for Wormhole

2 Virtual Channels

Virtual Channels(VCs) are independent logical channels that are multiplexed on a single physical channel. They were originally developed to prevent deadlock in networks which use wormhole routing. These days, VC's solve several other problems. The analogy is with a two lane road. If you are stuck behind a person trying to turn left (as shown in the figure 6) and the person ahead of you cannot turn (due to some reason say too many

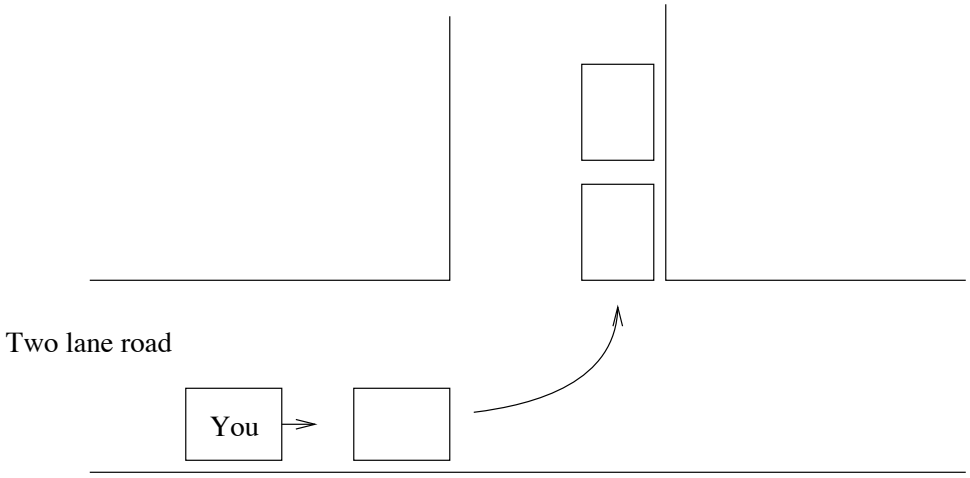

Figure 6: Two lane road analogy of blocking

cars in the left road), then even though you wanted to go straight which was free, you end up being blocked. The solution is to have a different lane for those trying to turn left (in other words a 'virtual road').

The concept of virtual channels should not be confused with that of Head of line blocking(HOL) whereby separate queues are kept for each output at each input. Providing virtual output queues only prevents blocking in one stage. Virtual channels, however prevent blocking completely. This is because virtual channels logically split up buffers as well as channel bandwidth so two packets might be on the same channel at the same time. A clearer picture is shown in the figure 7. In the first part of the figure packet B is blocked behind packet A while all physical channels remain idle. In the second part of the figure, virtual channels allow packet B to pass blocked packet A using additional buffers.

3 The canonical node employing virtual channels

Figure 8 shows a diagram of a router which supports virtual channels. The input buffers are segmented such that each VC has its own buffer space. Buffer state for the next node is stored at the output ports of the current node. The buffer state includes a count of the available next node buffers for a given VC, and a bit to indicate if that VC has been allocated. A crossbar or switch connects the input and output ports.

As stated above, the states of the router(as seen in Fig. 8) may be I (Idle)- when the router has nothing in its input buffers, R - routing, when the router gets a head flit and thus tries to find out its physical output channel, V -VC allocation - when the router tries to give the head flit a VC, A - active - when the router sends all the following body flits along the same allocated VC and T - tail deallocation when it receives a tail flit,

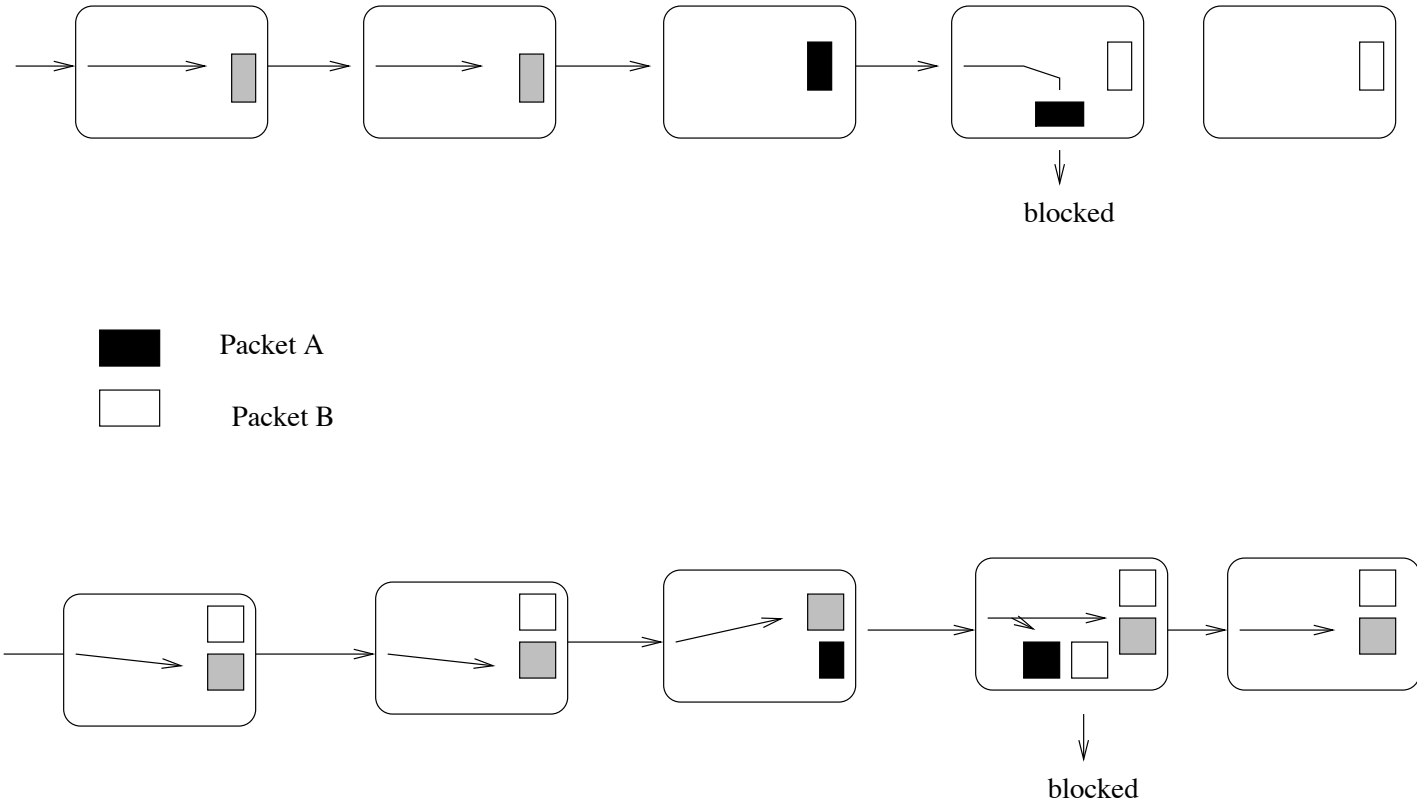

Figure 7: Virtual channels at work

routes it and frees up the VC and goes back to routing or idle state.

The output side state includes a count for the buffer available downstream and a busy bit indicating whether the virtual channel is allocated. If the count reaches zero, it means that there is no space in the input buffers in the downstream router and so this router should hold its flit. As soon as the next hop input buffer gets some free space, a signal(credit) is sent back to the router through either an explicit back channel or by piggy-backing on some other flit. The count is now increased by 1 and the flit that was being held is sent. If this flit was the tail, the router deallocates that VC by setting the busy bit down (see Fig 8). After this, the state for that VC goes back to idle state.

Figure 9 shows details of VC implementation. A head flit arriving at the input port is placed in to the head register of the corresponding VC. Using the Routing relation and the contents of the head register, a preferred direction vector is generated which indicates all possible output ports that could be used by the packet. Another module reads the state of the allocate bits on all output ports and finds out which VCs are not allocated as yet. Thus, the router has to do a bipartite matching. Once the matching is done, the incoming packet is allocated to the VC using the bipartite matching.

Now, bandwidth in the switch must be allocated for the incoming flits. This is done

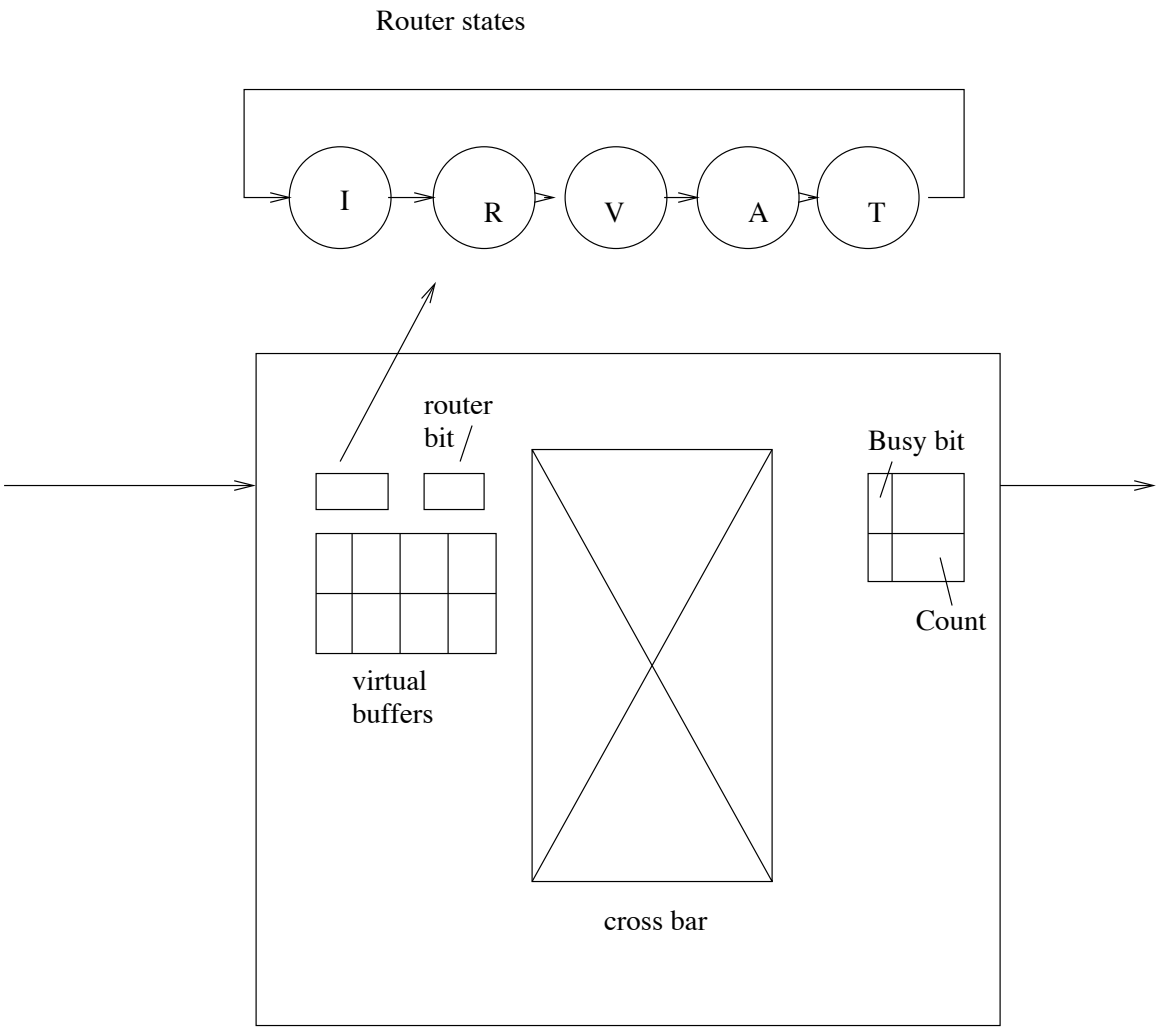

Figure 8: A canonical node/router using virtual channels

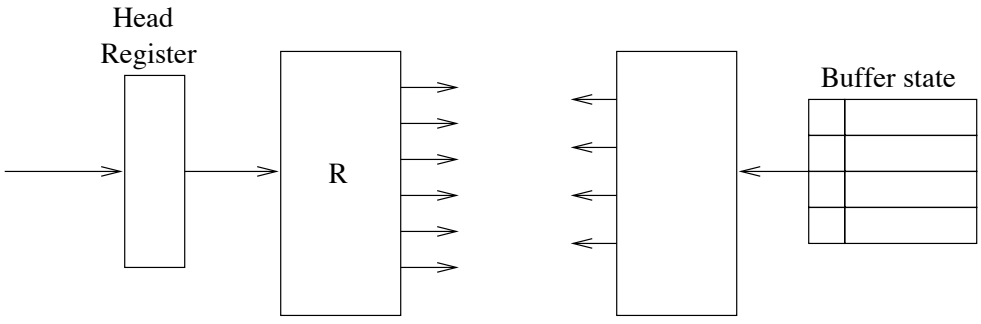

Figure 9: Details of Implementation of VCs

on a per flit basis. This problem is another bipartite matching problem between input ports and the output ports.

The output channel bandwidth is split between the VCs using some scheme (maybe fair or even greedy). The previous two were bipartite matching kind of problems. However, this one is arbitration for a single resource. Simply arbitrating is simple, the difficult part is to allocate fairly.

The size of the buffer of each virtual channel should be (RTT+think_time at src + think_time at dst)*bandwidth. This is because when a flit is unblocked, the buffer should contain enough data that it does not empty before the upstream router can start sending data.In addition to this roundtrip delay, at source, as well as at destination, we need to consider a 'Think time' which consists of the source (resp. destination) node 'making' decision for flow control purpose. Reducing buffer size below this would result in inefficiencies. Increasing the buffer size more than this results in very marginal improvements. The only benefit is that a blocked packet will occupy fewer nodes. Buffer space can be better utilized elsewhere.

4 Uses of Virtual Channels

As mentioned before, there may be several virtual channels(VCs) which are logical channels on one physical channel. Virtual channels are allocated to packets while physical channels are allocated to flits. In an interconnection network, it is very important to be non-interfering. Non-interfering means that any other node should not suffer just because lots of traffic is being sent to some other node and is getting clogged. This is because it often happens that one node is being sent traffic that is several times what it can handle. When this happens, the neighboring nodes slowly start getting clogged and so on until a whole tree rooted at the clogged destination is clogged. This is known as tree saturation.

One way to solve this problem is to drop packets destined to that node (call it A). Another solution uses virtual channels. We can have a virtual channel dedicated to each of the nodes in the network. A virtual channel per node is not required. Several nodes which do not block each other can be assigned the same channel and this will accomplish the purpose equally well. This solves the storage problem.

By using VCs, we split the bandwidth of an outgoing link among VC's. So if A is blocked, B's share of bandwidth over the same link can still be used.

Virtual channels also support QOS. Traffic may be of various types. There could be low priority traffic like file transfers or higher priority like system/control messages or real time traffic like audio/video. By allocating a different virtual channel to different priority traffic, we can provide low latency to high priority traffic. At the same time we can maintain high throughput for low priority traffic.

Figure 10 shows the latency observed by two traffic classes in a network using VCs. The delay for the high priority traffic is quite low. Even the average delay for low priority traffic is not large. The only problem is that the delay distribution has a very long tail.

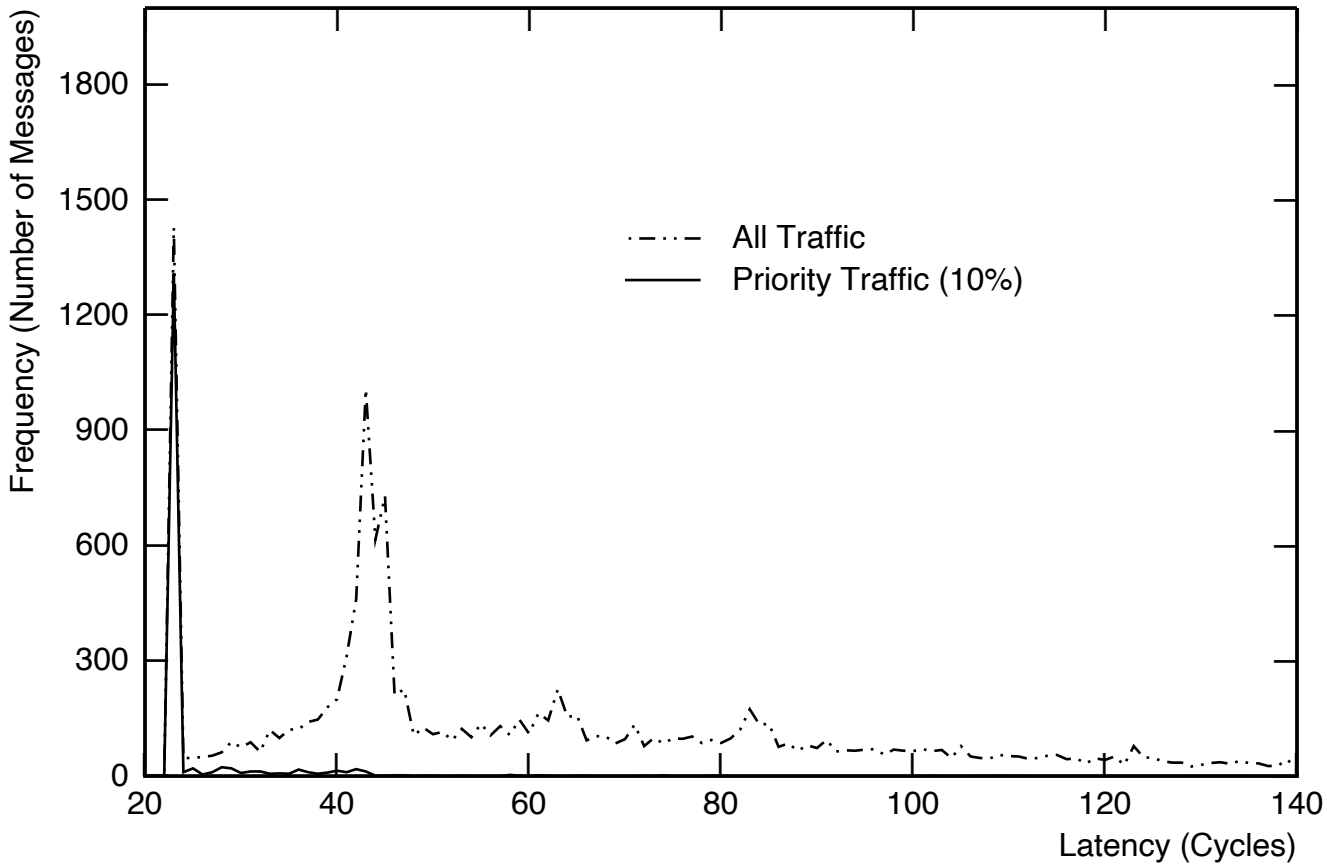

Figure 10: Latency of High Priority vs Low priority Packets

This means that low priority packets may hang on in the network for a very long time. This tail can be shortened significantly by giving priority to older packets within a virtual channel. For example, we may send the oldest packet first within a virtual channel.

VCs could also improve throughput of the network. Figures 11 and 12 illustrate this. Packet A is to be forwarded East while packet B is to be forwarded North. The East Channel is occupied and hence packet A is blocked. However, the North channel is free and Packet B should be forwarded. However, since it arrived after A, it is blocked resulting in under utilization of the North channel. This is shown in Figure 11.

Figure 12 shows how this is improved using Virtual channels. Now Packet A occupies only its virtual Channel. Packet B comes in to Virtual Channel 2 and is not blocked by A and can be forwarded to the North Channel. Thus, VCs improve the utilization of the channels. Hence, a very big plus point of using VCs is that it achieves much higher

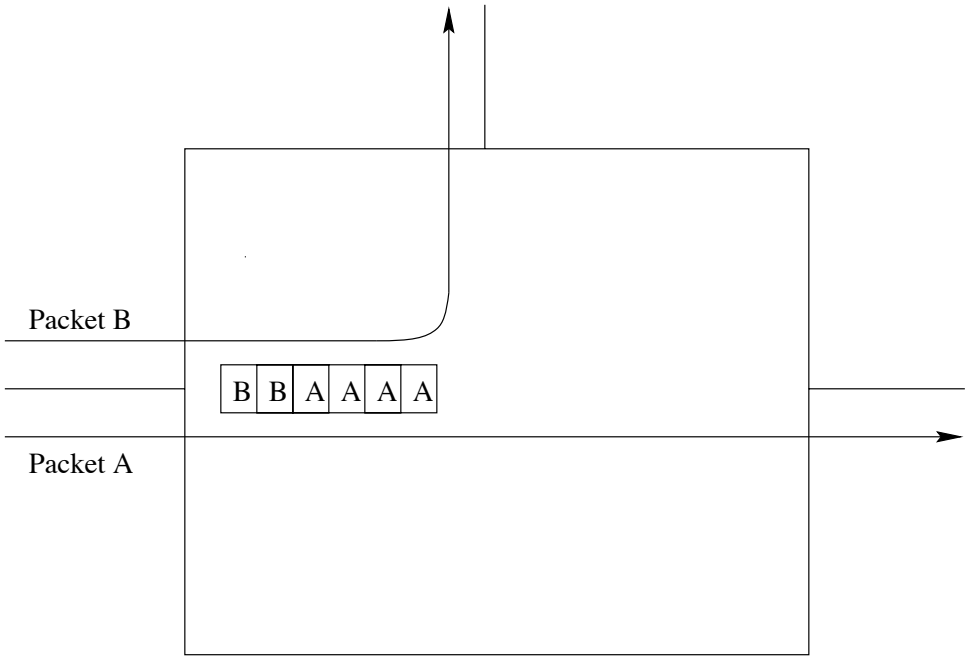

Figure 11: Throughput without VCs

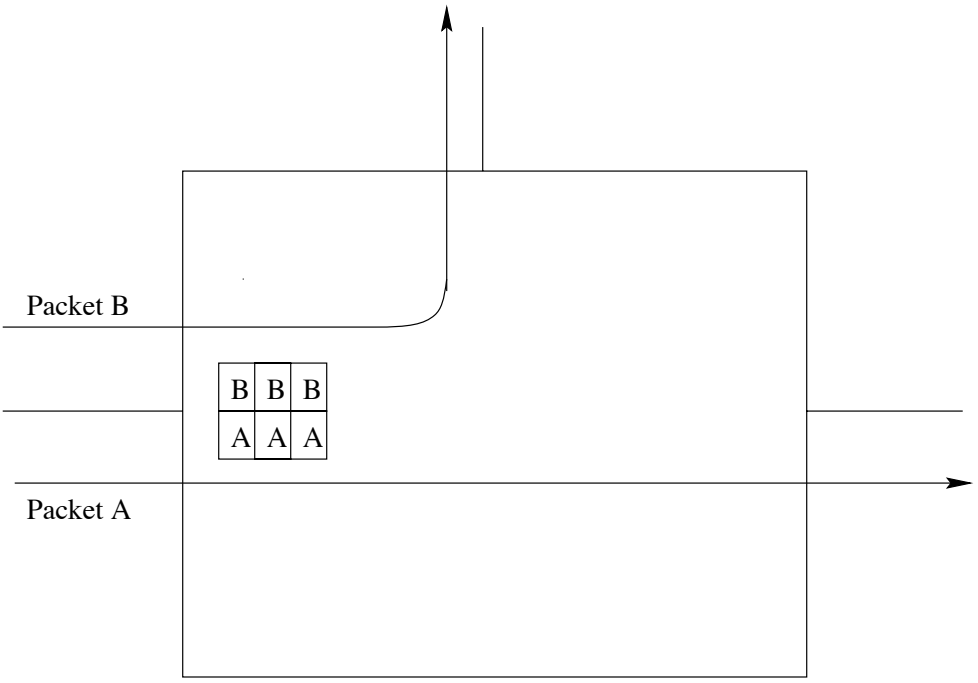

Figure 12: Throughput with VCs

throughput with much less buffer space.

5 Flit Reservation Flow Control

When a buffer becomes available it takes a round-trip time for it to be used by data sent by an upstream router. The flit leaves, there is some think time then a credit is sent. A wire delay later it arrives and then there is again some think time and the flit is sent which comes after a wire delay. In all, it really takes 2 * think time + round-trip time. One way to get around this is to statically allocate all resources at the start. Flit Reservation Flow Control allows us to allocate dynamically.

Suppose we have the head of the packet before the rest of the packet. Let it carry routing information and the times at which the data packets are to come. There are several places where such a thing happens. For example, a multiprocessor requesting a read from memory. The data will be available at a known time. Thus, the read request can reserve resources for the data ahead of time.

To reserve buffer space for incoming packet, a scoreboard is maintained. For each buffer, we maintain the times when it is occupied. Since, we know when a flit is arriving, we can reserve a buffer by allocating it in the scoreboard.

From the input buffers, the flits go through the routing function. Here the output port is to be scheduled. Before sending the flit out, we need to allocate a channel and buffer space at the far end to receive it.

We illustrate the output reservation table with an example in Figure 13. The channel is free in cycle 8, but we can't reserve that because there are no buffers available at time 11 on the next node. We must be able to hold the buffer indefinetely because we don't know when the flit will be scheduled to leave to the next node. The earliest time we can send the flit is 12. Thus, we mark the channel busy and decrement the buffer count from that cycle onwards (Figure 14). When the next node schedules the departure of the packet, the buffer is released and a control flit is sent back over the credit channel. This results in the counter being incremented.

After the assignment of input buffers, we have a time and a buffer number for each data flit. After we schedule the output port, we have the output time. We can then go to the scoreboard and unmark the buffer after the cycle in which the flit is scheduled to leave. See Figure 15.

When the body flit comes, we get the write address and the read address from the scoreboard. The write address is where the incoming flit is written and the read address is where the outgoing flit comes from. It is essentially a FSM.

6 Benefits of Flit Reservation

With flit reservation, buffers are more efficiently utilized. The buffers are not idle just after a flit leaves a full buffer due to the reservation. When the network is under high

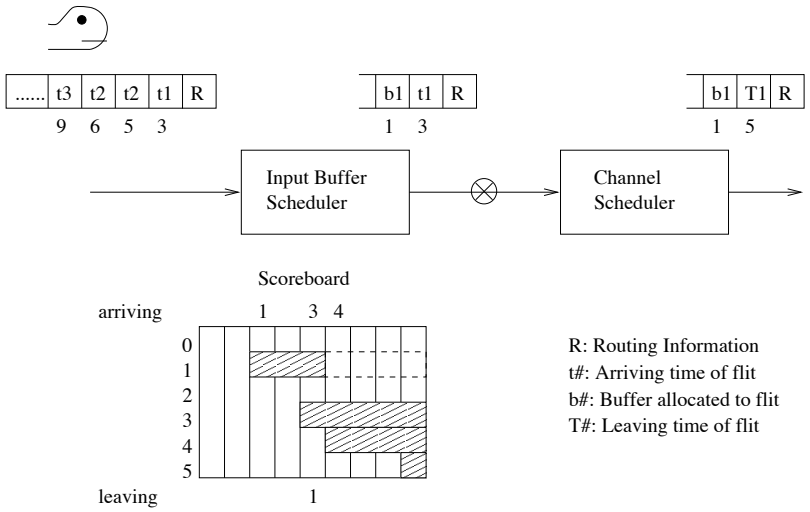

Figure 13: Flit reservation flow control

output channel \ time		8	9	10	11	12	13	14	15	16	17
East Channel	Channel busy			X							
	Free buffers on next node	2	1	1	0	1	2	3	4	4	4

Figure 14: Output Reservation Table before data flit is scheduled

load the table is completely filled. The graph in Figure 16 illustrates the benefits. The graph is for an 8x8 mesh with 8 buffers. Without flit reservation, the network saturates at 60%. Flit Reservation takes the throughput to 80%. It would take almost twice the number of virtual channel buffers to achieve the same performance as Flit Reservation.

Example: Memory Access in a Shared memory Multiprocessor
We know in advance that a memory access will generate a reply so during the access time to the memory, we allocate the path through the network for the data to be transmitted. By this, we hide the routing/arbitration process by the latency of the memory. We take advantage of the fact that we know that smth will be transmitted back from the memory. See figure 17

input channel \ time		8	9	10	11	12	13	14	15	16	17
West Channel	Buffer in			5							
	Buffer out (North)										
	Buffer out (South)										
	Buffer out (East)					5					
	Buffer 0										
	Buffer 1										
	Buffer 2										
	Buffer 3										
	Buffer 4										
	Buffer 5			X	X						
	Buffer 6										
	Buffer 7										

Figure 15: Input Reservation Table

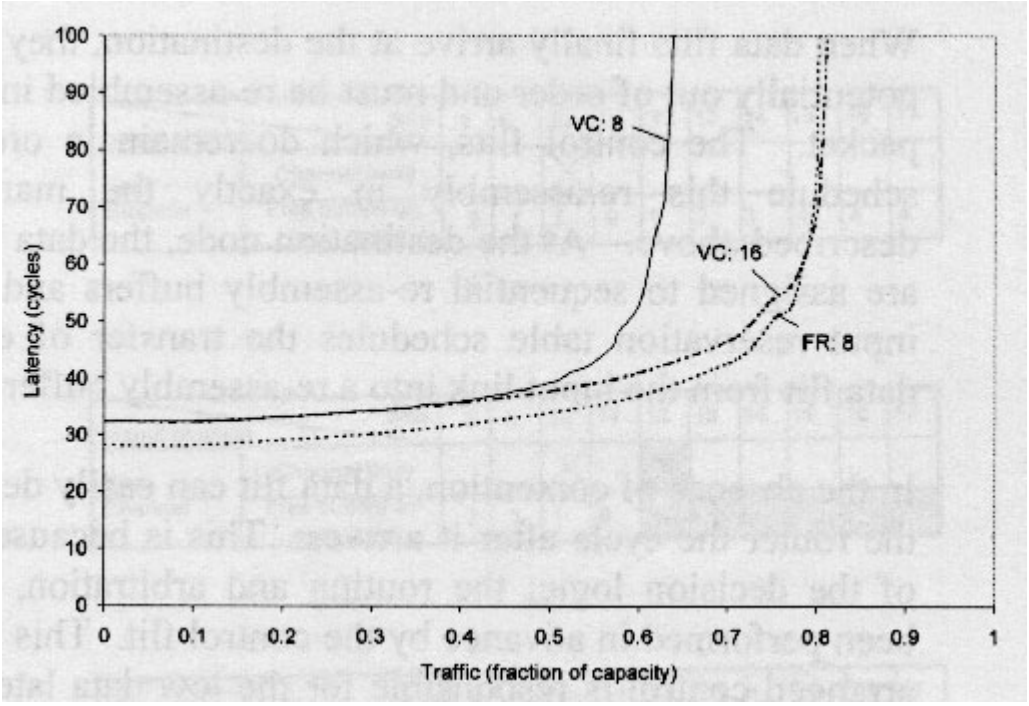

Figure 16: Comparison of Flit Reservation and Virtual Channel Flow Control

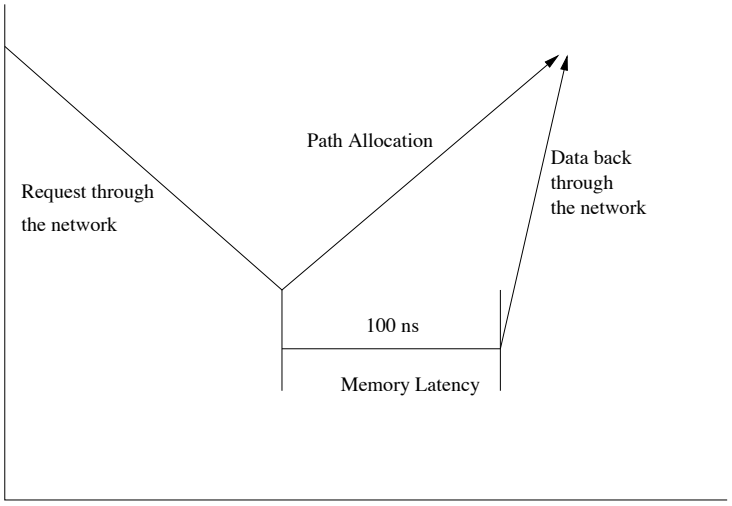

Figure 17: Hiding the latency of the network by the latency of the memory

7 TDM: Time Domain Multiplexing

In some applications, the traffic patterns encountered are very predictable in the sense that they require always the same bandwidth and these requests are periodic. Example: System on Chip processing an incoming Video Stream. The traffic is characterized by a given frequency of sent packets and requires a consistant delivery time. Each time we send a packet, it goes through the whole process of routing and arbitrating for resources at each switch in its route. In order to take advantage of this predictability, we could assign a VC to this streams with a high priority which allows it to "win" all the arbitrations but the resources could be unavailable (All the VCs used for example). This leads to a variation in the latency of the delivred packets. So the solution is to statically allocate the resources through the path to this stream packets. We can decide to give the output port at each hop where the packet is to this packet one cycle out of ten if the stream bandwidth is 10Mbit/sec and the links bandwidth is 100Mbit/sec. (See Figure 18)

This figure shows the static allocation of the output ports at each hop given that the input stream packet comes at cycle 5 and given that it takes two cycles to go from one hop to another. This allocation is repeated each ten cycles.

This scheme allows a guaranteed QoS for this stream through the network since it gives this stream enough bandwidth to have consistent performance without really affecting the other traffics on the network.

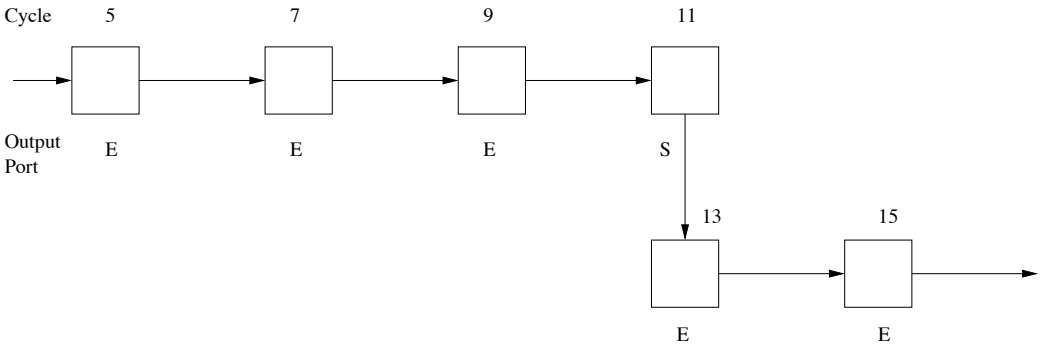

Figure 18: Time Domain Multiplexing