Virtual Channel, Flit Reservation Flow Control and Deadlock

Lecture #7: Tuesday, April 20, 1999
Lecturer: Bill Dally
Scribe: Chuck Fraleigh, Vicky Wong
Reviewer: Who???

1 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. However, they can also be used to provide quality of service and to increase network throughput.

VCs can be used to support QoS by assigning each type of traffic to a different VC. The traffic types typically include low priority traffic, such as file transfers, which requires high throughput, and high priority traffic, such as system messages, which requires low latency. The low priority traffic can tolerate some latency, and the high priority traffic does not require high throughput due to its low volume. By allocating a separate VC for each traffic class and giving precedence to the high priority traffic, the network can simultaneously provide low latency to the high priority traffic and high throughput to the low priority traffic.

Figure 1 shows the latency observed by two traffic classes in a network which uses VCs. The average delay for the low priority traffic is not very long, but the delay distribution has a very long tail. The low priority traffic can incur delays as high as 140 cycles. The delay for the high priority traffic, however, is always very small. It has a peak at about 20 cycles, the value for the round trip time of a channel. The maximum delay for high priority traffic is roughly twice this value. Without VCs, the high priority traffic would incur the same delay as the low priority traffic and would not be able to satisfy its latency requirements.

VCs can also be used to improve the throughput of the network. Figures 2 and 3 illustrate how this is accomplished. In this scenario packet A needs to be forwarded on the East channel, and packet B needs to be forwarded on the North channel. The East channel, however, is being used by some other packet passing through the router.

Figure 2 shows a router which does not implement VCs. When packet A arrives, it fills the input buffer and waits for the East channel to become available. Packet B arrives behind packet A and is blocked until packet A can be transmitted.

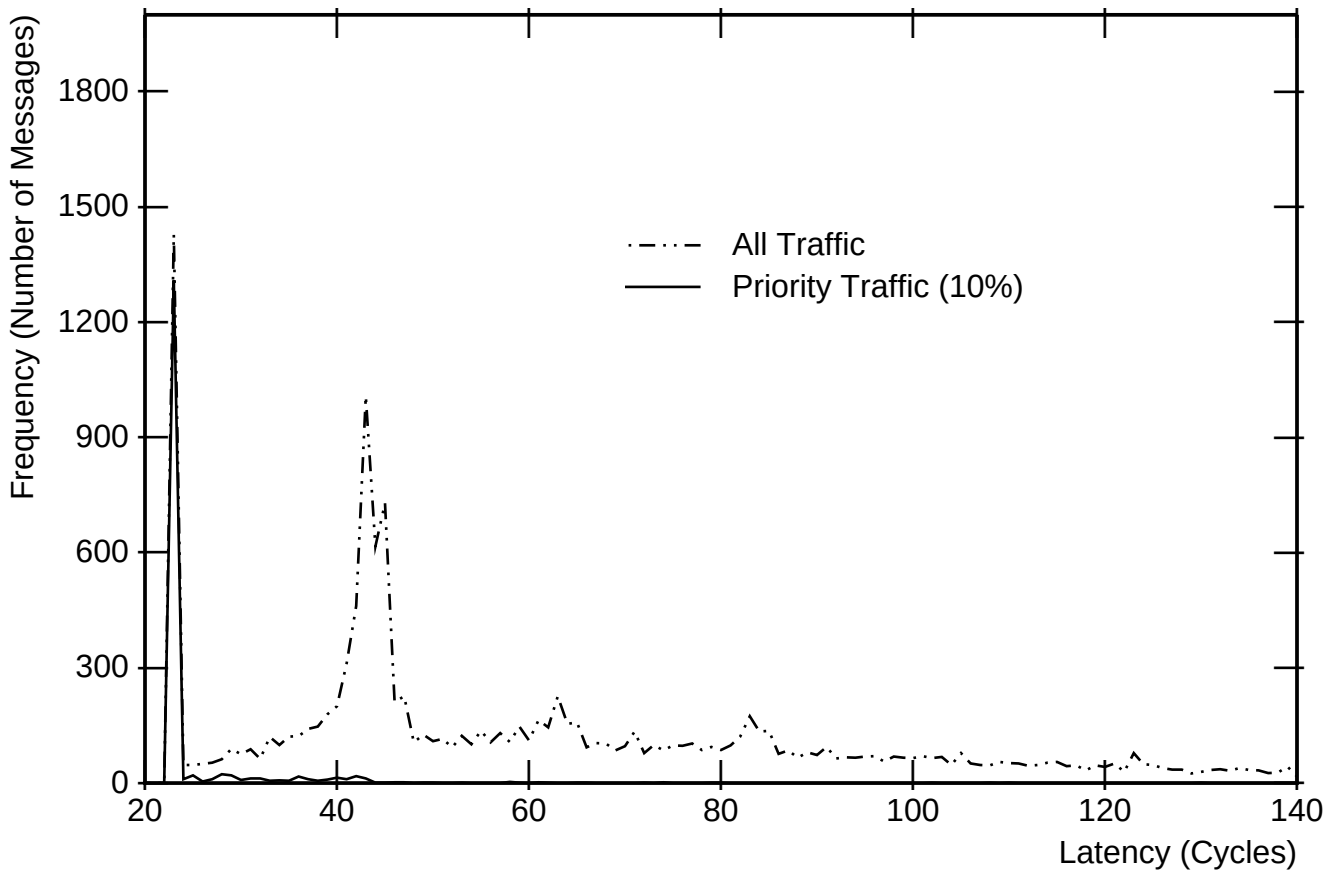


Figure 1: Latency of high priority and low priority packets

Figure 3 shows a router which does implement VCs. When packet A arrives in this case, it only fills the input buffer for VC1. The buffers for VC2 are independent of the VC1 buffers, so packet B can be transmitted on the North channel without having to wait for packet A. VCs prevent large packets from consuming all the resources in a router and blocking other traffic.

The size of the buffer for each VC should be a little larger than (round trip time * channel bandwidth). When a flit becomes unblocked, the buffer needs to contain enough data so that it does not empty before the upstream router can begin sending additional data. Performance can be improved by increasing the buffer size beyond the rtt since a blocked packet will occupy fewer nodes, but the improvements are marginal. These buffers can be better utilized for additional virtual channels to further increase throughput.

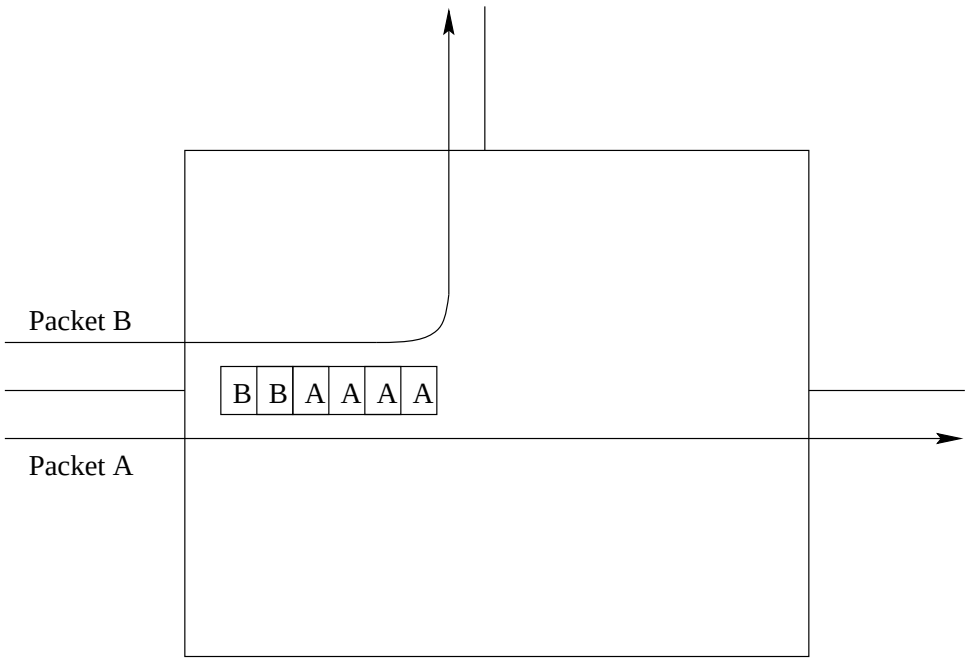


Figure 2: Throughput without VCs

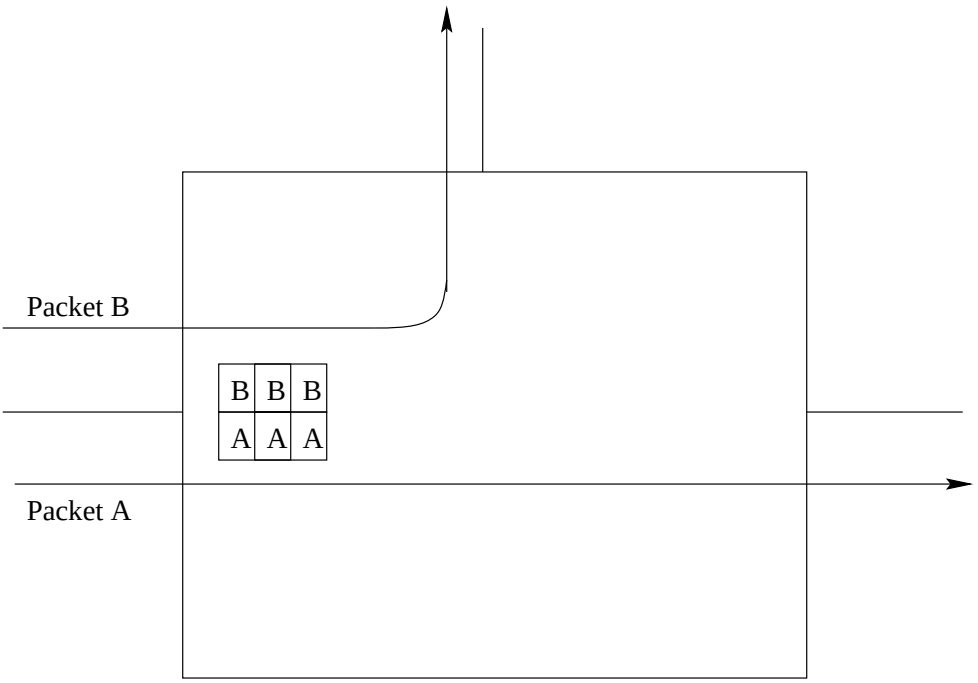


Figure 3: Throughput with VCs

2 Virtual Channels Implementation

Figure 4 shows a diagram of a router which supports virtual channels. The input buffers of the router are segmented such that each VC has its own buffer space. Buffer state is stored at the output ports of the upstream router (the state shown in the figure is state for the downstream router). The buffer state includes a count of the available downstream buffers, and a bit to indicate if the VC has been allocated. A crossbar or other network connects the input and output ports.

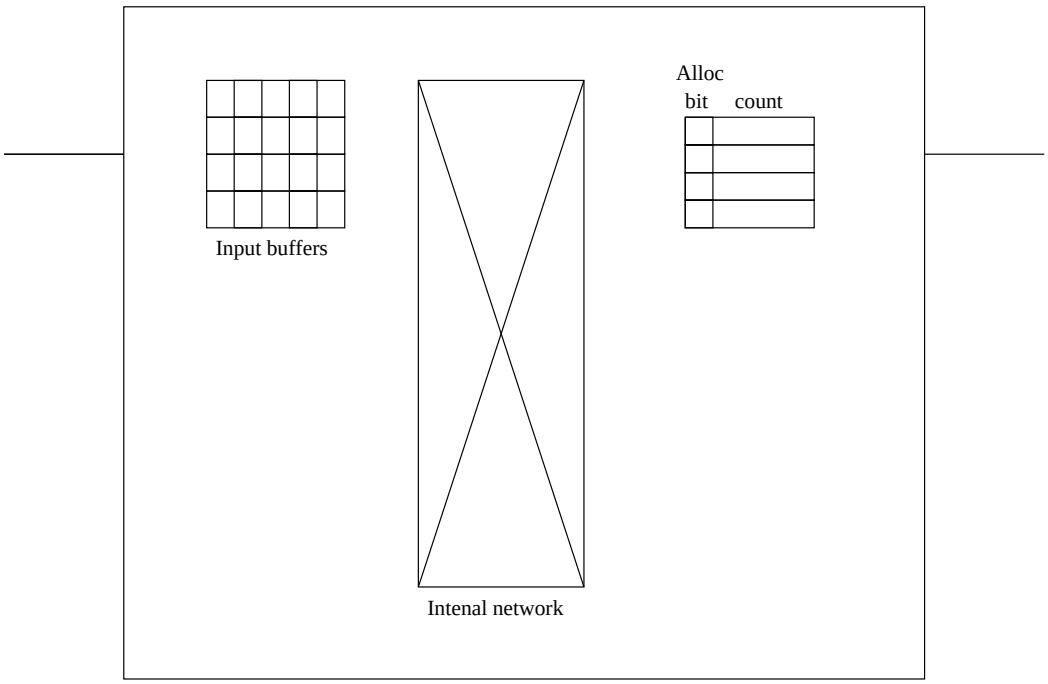


Figure 4: Virtual Channels Implementation

Figure 5 shows a more detailed implementation of VCs. When a head flit arrives at an input port, it is placed in the head register for the appropriate incoming VC. The routing relation R reads the contents of the head register and generates a preferred direction vector which indicates all possible output ports that could be used for this packet. A separate module reads the state of the allocate bits on all the output ports and determines which VCs for each output port are not being used. The router must find a match between the preferred direction vectors for all the new head flits and the set of VCs which are unused on the output ports. Once the bipartite match has been solved, the router allocates the appropriate VCs to the entire incoming packet.

Once a packet has been assigned a VC, the router must allocate internal switch bandwidth to all the incoming flits. This allocation is done independently for every flit in a packet. The allocation is another bipartite matching problem, this time between the set of input ports and the set of output ports. The biggest problems occur when two

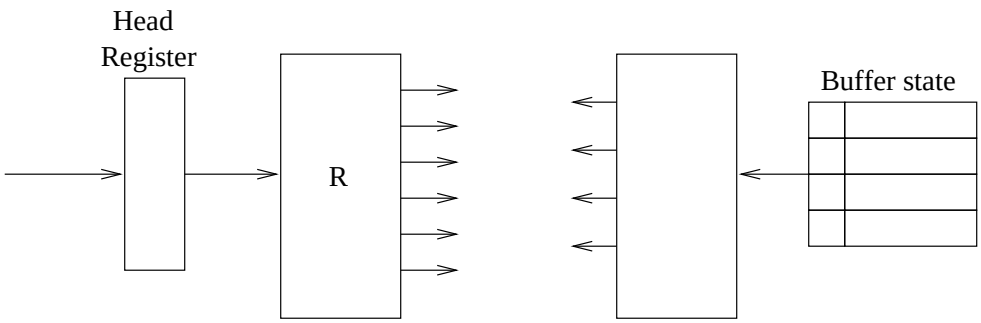


Figure 5: Virtual Channels Details

VCs on the same input port need to be forwarded to two different output ports. Since each input port only has a single channel to the internal switch fabric, one flit must be blocked.

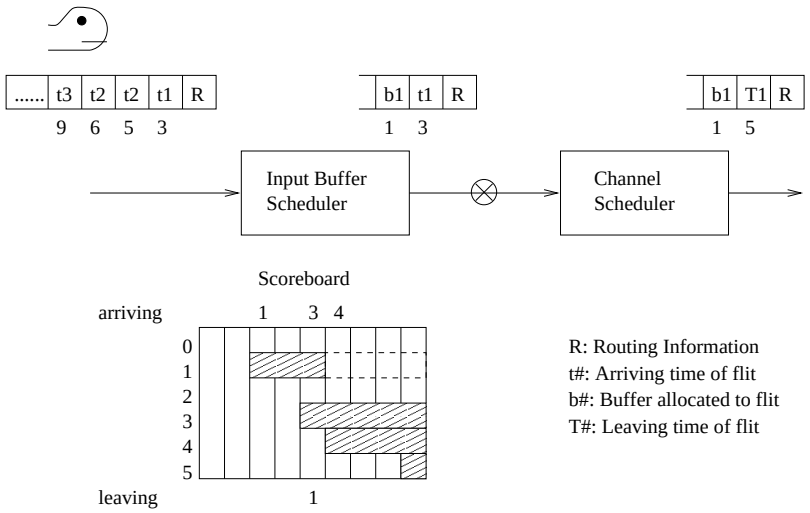
The final step is to assign output channel bandwidth to the different VCs using the output channel. Unlike the previous steps which involved allocating multiple resources to multiple requesters, this step is simply an arbitration for a single resource by many users. The most important criteria for this arbitration is fairness among the various VCs.

3 Flit Reservation Flow Control

As mentioned earlier in this lecture, when a buffer becomes available, it takes roughly a round trip time before it is used by another packet. How can we fix this? Well, we want to allocate resources ahead of time. One approach is to allocate the resources at the beginning of time. This doesn't allow packets to change their mind (for example, a different address). It is called static scheduling, since the network movements are orchestrated by the compiler. Another approach is the flit reservation flow control.

Suppose we have the head of the packet ahead of the rest of the packet. It carries the routing information and a bunch of times. These are the times in cycle after the head arrives, in which each data element will arrive. In some cases you can do this naturally. For example, in a multiprocessor, you send a memory request to a memory bank and it may take 100ns or 150ns to read the memory. You know exactly when the data will be available so you can send the head of the reply ahead of time and reserves the resources.

3.1 Process of Reserving Resources



When the head arrives, we want to reserve incoming buffer space for the packet. Imagine there is a scoreboard. For each buffer, it has information of when it's occupied. Since we know when each flit will arrive, we can reserve resources for them by locating a buffer that is free from the time of arrival onwards and marking it busy. In any given cycle, we can say which buffer is going to arrive and which one is going to leave.

After we schedule the input buffers, it goes through some routing function. Now we need to schedule the output port. Suppose we have a flit arriving. Before we can send the flit out, we need a channel that is free and buffer space to store it at the far end.

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 6: Output reservation table (before the data flit is scheduled)

In the example (see figure 6), the channel is free in cycle 8 but we can't reserve that because at cycle 11, there is no free buffers available on next node. When we reserve buffer, we need to reserve it for the time from that cycle onward.

In this example, the earliest time we can send the flit is cycle 12. So we mark that channel busy and decrement the buffer count from that cycle onward (see figure 7). Eventually, the next node will schedule a departure, which corresponds to the release of the buffer. At that point, a control flit is sent back over the credit channel and the counter will be incremented.

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

Figure 7: Output reservation table (after the data flit is scheduled)

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 8: Input reservation table

After the assignment of input buffers, we have a time and a buffer number for each flit. And after we schedule the output port, we have the output time . We can then go back to the scoreboard and unmark the buffer after the cycle in which that flit is scheduled to leave (see figure 8).

When the body flits start coming in, all we need is a write address and a read address (which can be obtained from the scoreboard). The write address is where the incoming flit lands and the read address is where the outgoing flit comes from. If they happen to be the same, bypass around the memory. You just need to sequence the finite state machine. The flits fly right through. (See figure 9)

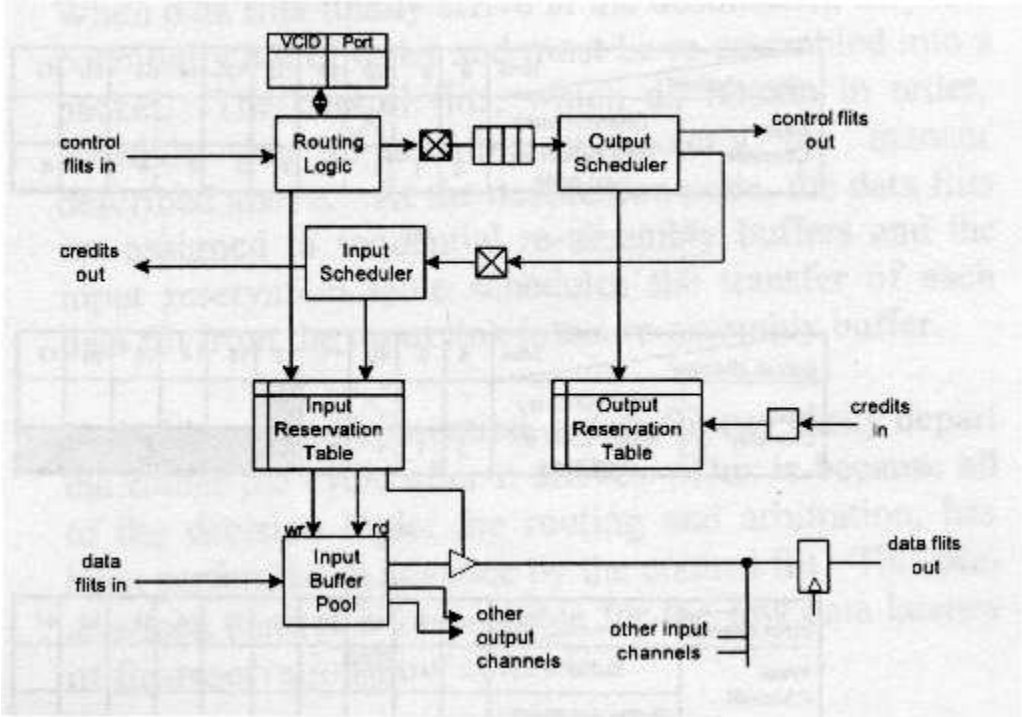


Figure 9: Block diagram of a router using flit reservation flow control

3.2 Why is this a good idea?

Buffers are not idle between a flit departing and a flit coming in. When thing gets crowded, the table gets completely filled.

Refering to the graph in figure 10, with virtual channel and 8 buffers (for an 8x8 mesh), it saturates at about 60%. With flit reservation and the same number of buffers, it saturates at about 80%. It would take twice the number of virtual channel buffers to achieve the same performance.

4 Deadlock

What causes deadlock? Cyclic resources dependency.

For wormhole routing, the resources are the channels (or more precisely, the state space). For store and forward (or virtual cut-through), the resources are the buffers (since the whole packet has to be stored in one node).

Here is a figure showing a deadlock situation in a 2-D mesh. It takes 4 packets to deadlock, assuming we only use minimal routes. If we allow non-minimal routing, it is less likely get deadlock. It would only take 2 packets to deadlock though.

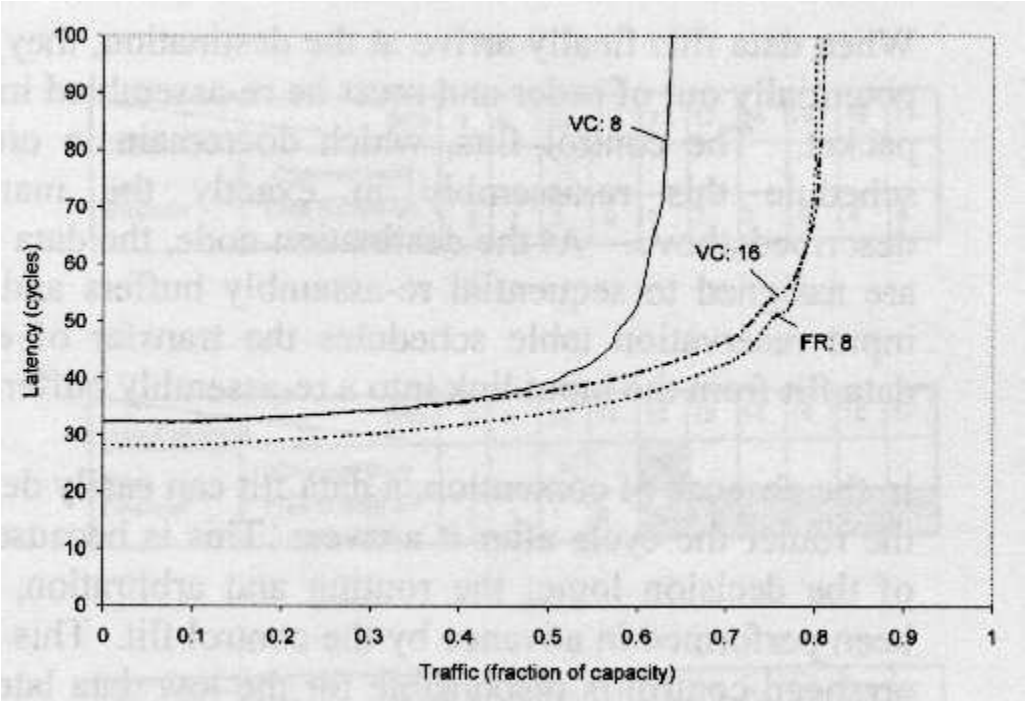
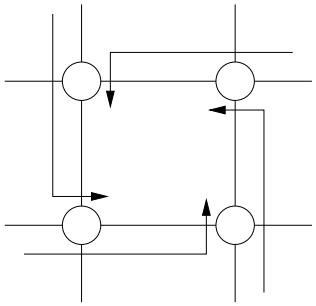
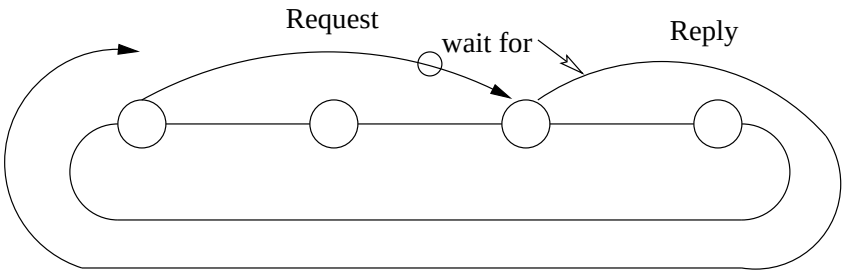


Figure 10: Comparison of flit reservation flow control and virtual channel flow control



Consumption Assumption: When a packet gets to its destination, it is removed from the network (consumed) and the resources allocated to it is released.

An example of cases in which this does not hold is multiprocessors with cache conherency protocol. A cache controller can only handle a certain amount of requests at a time. Until it finishes some prior request, which involves sending a reply into the network, it will not accept more incoming request. If you just created another request, you also created a resource dependency. Even if the network is deadlock free, the protocol itself can cause deadlock. The situation is illustrated below.

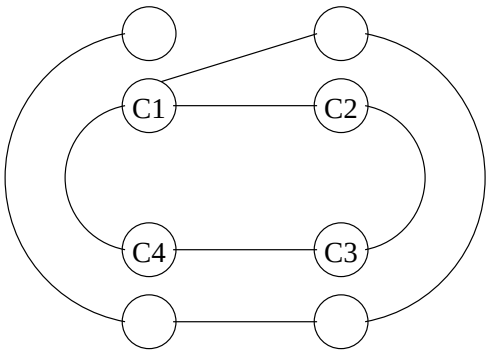


4.1 Oblivious Wormhole Routing

$R : C \times N \mapsto \{C\}$

We pick C randomly.

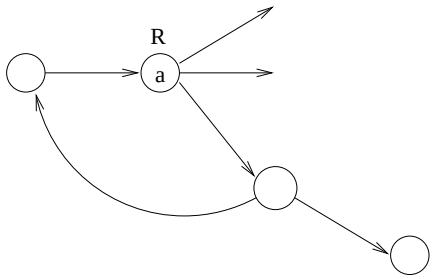
If the routing relation R is cyclic, the network is not deadlock free. i.e. $C1$ leads to $C2$, $C2$ leads to $C3$, and $C3$ leads back to $C1$. One way to prevent deadlock (intra-dimension deadlock) is to use virtual channels. We break the cycle by restricting the routing relation. Like this:



Another way to prevent deadlock is to send the head first and try to route all the way to the destination. If it's blocked, it would track back to the source to try again. The body will be sent when the head reaches the destination.

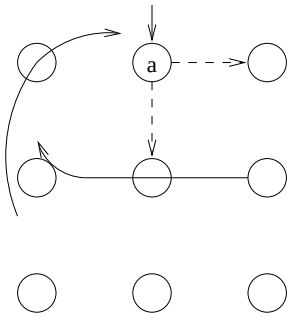
4.2 Adaptive deadlock

With adaptive routing, several routing choices may be available at each node. This makes it difficult to determine when cyclic dependencies can occur. Consider the following example.



When we come into node a, the routing relation R allows multiple routing directions. One of them happens to be part of a cycle, Is it true that we can not use that channel?

Here's another example. If we go south at node a, it will deadlock. It will not if we go east.



However, with adaptive routing, it is not necessary to remove all cyclic dependencies. The Duato's algorithm shows that it is sufficient to have a subset of channels, $C1$, which provides a set of acyclic, connected routes, $R1$. If we block, we can always drop to the $C1$ channels, which can not cause deadlock. However, the routes must be selected carefully to avoid indirect dependence. For example, a dependency can be created when a route uses channels in $C1$, say, a and b. Then it uses channels not in $C1$, say x and y. And finally it uses channels which are $C1$, say c and d. Even though you can't get from b to c in $C1$, if you block at c, you block b. This may create a cyclic dependency. A simple way to prevent this is to prohibit the use of non-$C1$ channels once a packet has used a channel in $C1$.

Here is a concrete example of a network the uses the Duato's algorithm.

$C1, R1$: 2 VCs/PC, channels used by dimension order routing

$C - C1, R - R1$: 1 VC/PC, arbitrary routing

The restriction is that once you drop into C - $C1$, you cannot go back to $C1$.