EE482: Advanced Computer Organization Lecture #8

Interconnection Network Architecture and Design
Stanford University 22 April 1999

Deadlock and Router Micro-Architecture

Lecture #8: 22 April 1999
Lecturer: William J. Dally
Scribe: Wilson Chin, Weihaw Chuang

Adaptive Deadlock

In a network with a routing relationship R, the network is prone to deadlock if there are cycles of channels.

- If Oblivious routing is used, deadlock will occur if there are cycles of channels.
- If Adaptive routing is used, deadlock will not necessarily occur if there are cycles of channels.

Duato's Algorithm

Given the set of routing relationships R over all channels C

$$R - C$$

we can route in a deadlock-free way using Duato's algorithm.

For example, given an 8-ary 2-cube (a 2-dimensional 8x8 torus) with the sub-relationship

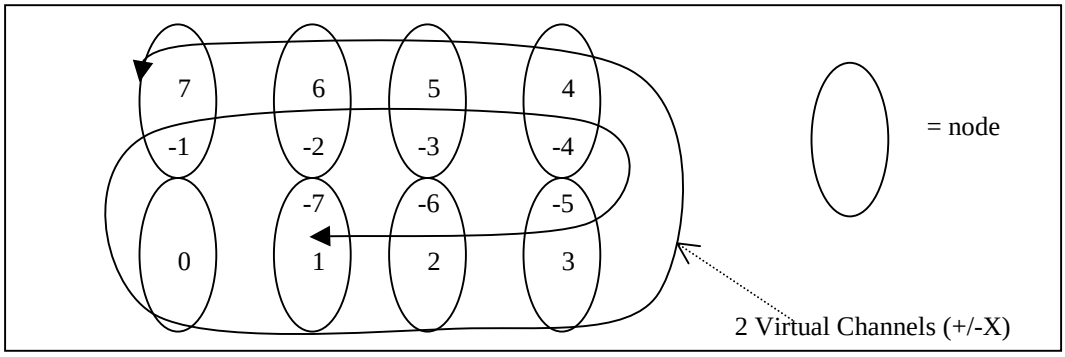
$$R1 - C1 \subseteq C$$

in which C1 contains 2 Virtual Channels per Physical Channel and is dimension-ordered, and the remaining subset

$$C2 \equiv C - C1$$

in which C2 contains 1 Virtual Channel per Physical Channel and is any minimal route, we can route all traffic in a deadlock-free way.

In order to break the cycle in dimension-ordered C1, we "unroll" the torus in the X-dimension by creating two virtual channels (Positive and Negative X) and aliasing the X-coordinate of all nodes such that the nodes along the x dimension are ordered:



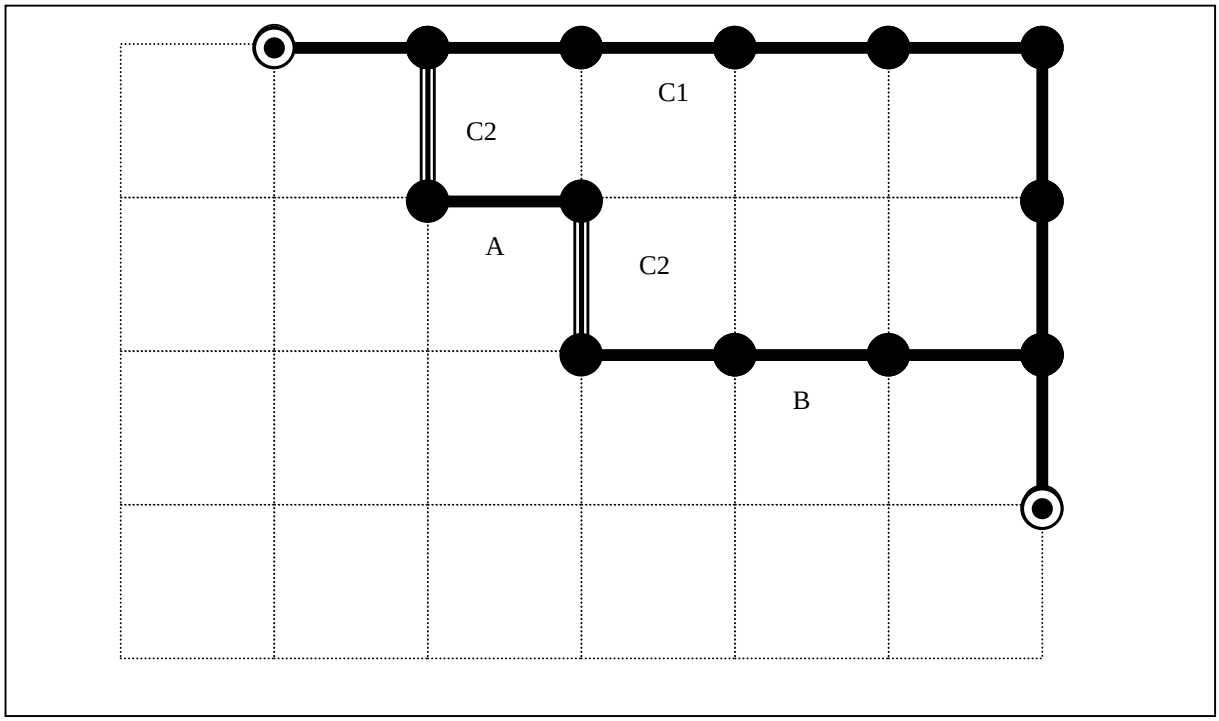
We can jump back and forth between C1 and C2 and still route traffic deadlock-free if C1 is deadlock free across R. This is because we can always make forward progress-- even in the case of blocking on C2 we can then jump to C1 and continue. We are constrained in C1 by a partial ordering on the dimensions and nodes. In the example we used dimension ordering routing which is acyclic and is defined over C1 as:

Xi > Xi+k for any k

Xi > Yi

The first relationship defines a relationship while we are routing in the X dimension. The second relationship applies when we start to route over Y, and implies that once we start to route Y we cannot route along X again. This partial order relationship constrains our selection of destinations when in C1. The converse for C2 is not similarly constrained.

We can consider direct and indirect dependencies. In the case of A and B with different Y values, they don't have a direct dependency meaning over R1 A<>B. A and do have an indirect dependency over R, with the following ordering relationship A > B.



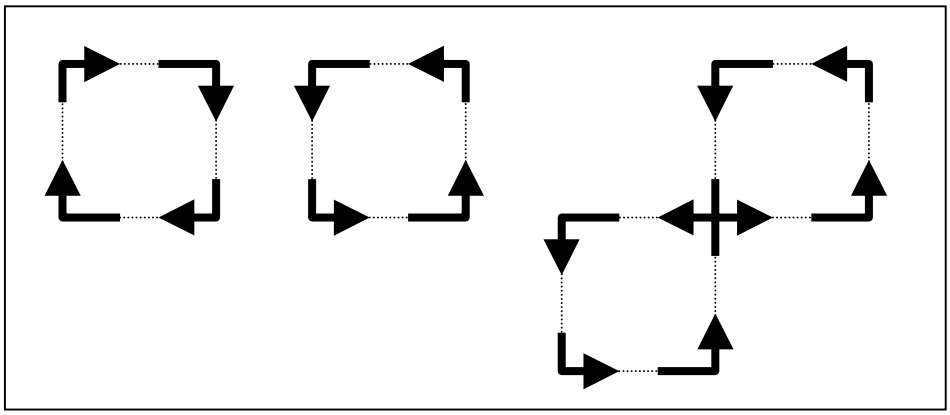
Extended Ordering

We can consider the above partial ordering taking into account direction. This enables us to "overshoot" routing past the destination without deadlocking, or breaking the partial ordering given above. Over C1 we consider whether the channel direction is negative or positive, and apply the following:

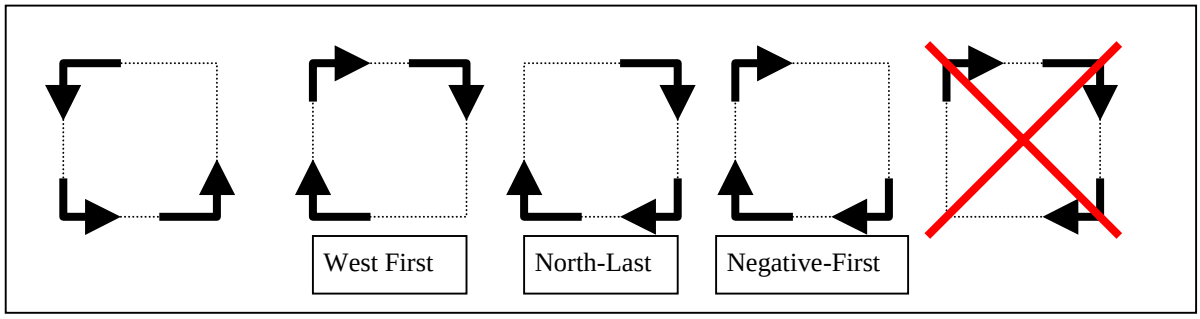
X+ > X- > Y+ > Y-

Turn Model

An alternate way of analyzing deadlock is to consider routing "turn" behavior, or the possible directions a given message can take out of a node, relative to the direction that message entered the node. We have a cycle if we make enough left-turns or right turns to complete a cycle:



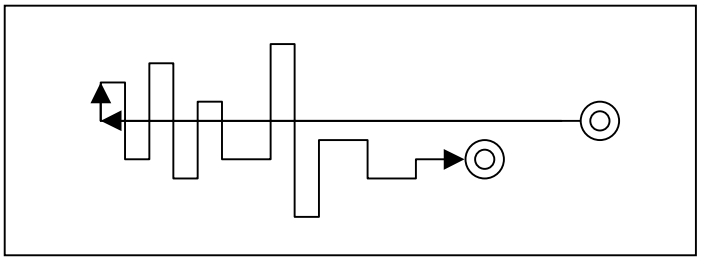
By limiting the possible turns, we can guarantee that a given channel does not have cycles, and is therefore deadlock free. For the 2-D torus, depending on which turns we eliminate, we can create three different deadlock free routing patterns.



Given the counter-clockwise pattern on the left, we can eliminate three of the four corner turns in the clockwise pattern shown on the right for deadlock-free routing. If we eliminate the remaining corner turn depicted on the far right, it is possible to deadlock in a figure-8. Here, the three right turns are equivalent to the missing North-to-West left turn in the counter-clockwise pattern on the left, completing the cycle.

West-First behavior

If we eliminate the South-to-West and North-to-West turn, we observe that it is impossible to turn westward when travelling in any other direction. All westward movement must occur at the beginning of the route, because once the packet has moved in the north/east/south direction, it cannot turn westward again. Non-minimal routes are allowed: packets may travel westward and overshoot the destination, and return on a meandering North/East/South path, as shown below.



North-Last behavior

If we eliminate the North-to-East and North-to-West turn, we observe that once a packet has moved north, it is impossible for that packet to turn to another direction. All northward movement must occur at the end of the route. Similar to the West-first case, non-minimal routes are allowed: packets may travel on a meandering East/South/West path overshooting the destination, and return on a north-only path.

Negative-First (Positive-last?) behavior

If we eliminate the East-to-South and North-to-West turn, we observe that once a packet has moved in a "positive" direction (north or east), it is impossible for that packet to turn to a "negative" direction (west or south). All positive movement must occur at the end of the route. Again, non-minimal routes are allowed: packets may overshoot the destination on a meandering negative path, and return on a positive path.

Generalization to higher dimensional topologies

The turns model can be extended to higher dimension topologies by observing that the effect of restricting turns is to limit the movement in a particular direction in a given dimension at either the beginning or end of a route. In the K-ary 2-cube West-first route discussed above, we restrict movement in the negative direction of one dimension by eliminating all turns into that direction/dimension. We can construct the analog of West-first routing in an arbitrary K-ary N-cube iteratively as follows:
We define the sets

$D \equiv \{positive, negative\}$ (all possible directions)

$N \equiv \{n \mid 0 \le n < NumDimensions\}$ (all possible dimensions, numbered)

T (all possible turns)

and denote all possible direction/dimensions as ordered pairs $(d,n) \mid d \subseteq D, n \subseteq N$.

We start by select a particular starting direction and dimension (d_0, n_0) where

$d_0 \subseteq D$

$0 \le n_0 < N$

We define T_0 to be the set of all turns from all other $\{(d,n) \mid d \subseteq D, n \subseteq N - \{n_0\}\}$ into (d_0, n_0).

We then iterate for all $\{i \mid 0 < i < N-1\}$
For every i, we select another starting direction and dimension (d_i, n_i) where

$d_i \subseteq D$

$n_i \subseteq N - \{n_j \mid 0 \le j < i\}$

We define T_i to be the set of all turns from all other $\{(d,n) \mid d \subseteq D, n \subseteq N - \{n_j \mid 0 \le j < i\}\}$ into (d_i, n_i).

The set of allowable turns is then given by the set

$$A \equiv T - \bigcup_{i=0}^{N-1} T_i$$

In other words, we chose a 0th direction/dimension, eliminate all turns into this direction/dimension, chose a 1st direction/dimension, eliminate all turns into this direction/dimension except the 0th, chose a 2nd direction/dimension, eliminate all turns into this direction/dimension except from the 0th and 1st, … until we reach the (N-1)th dimension.

We can do a similar construction for the other schemes (North-Last, Negative-First).

Three-dimensional cube example

For a k-ary 3-cube, we begin by selectin an arbitrary starting direction and dimension, say (positive, X), and we eliminate all turns into (positive, X) from the set $\{(d,n) \mid d \subseteq D, n \subseteq \{Y,Z\}\}$. In this step, we eliminate the following turns T_0:

- (postive, Y) to (positive, X)
- (negative, Y) to (positive, X)
- (positive, Z) to (positive, X)
- (negative,Z) to (positive, X)

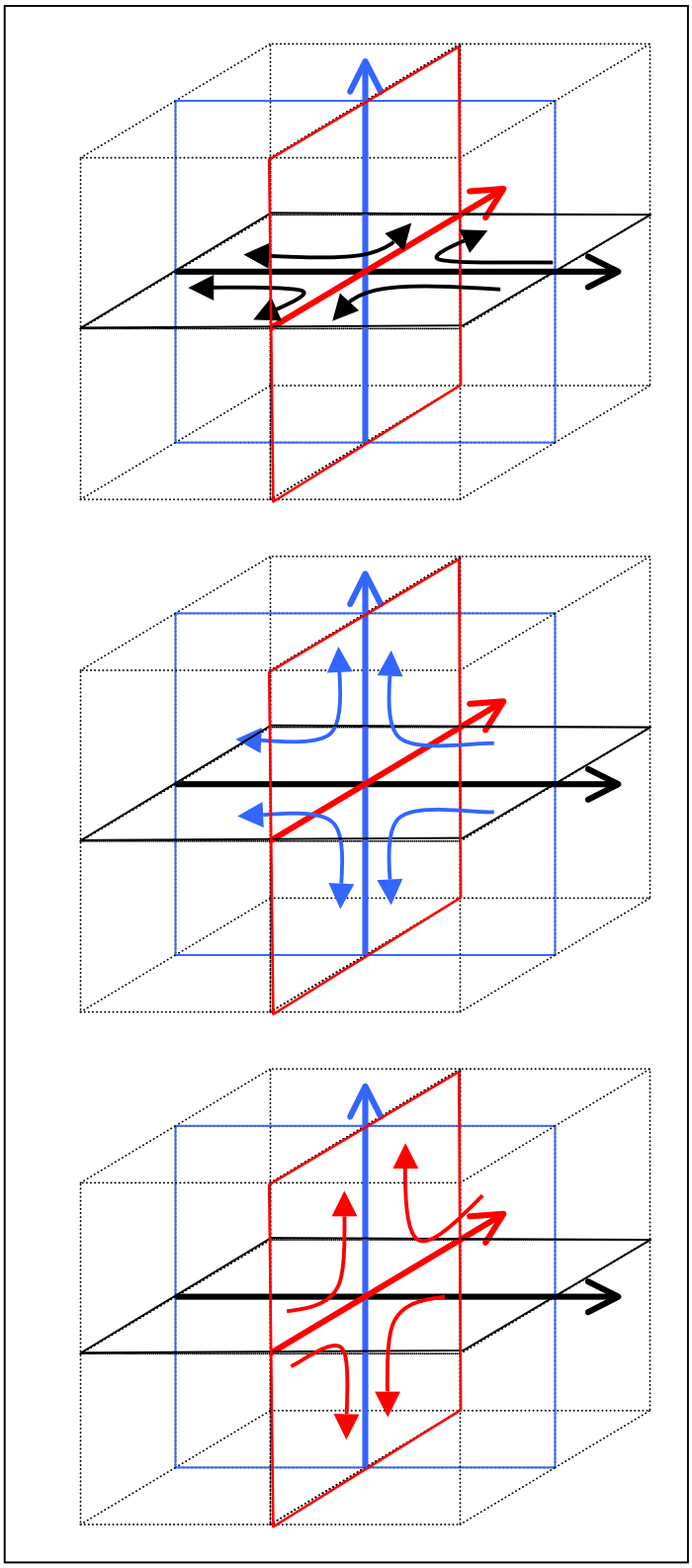
Next, we select another direction/ dimension from the remaining dimensions {Y,Z}, say (positive, Y). We eliminate all turns into (positive, Y) from the set $\{(d,n) \mid d \subseteq D, n \subseteq \{Z\}\}$. In this step we eliminate the following turns T_1:

- (positive, Z) to (positive, Y)
- (negative, Z) to (positive, Y)

We are left with the allowed turns A:

- (positive, Y) to (negative, X)
- (negative, Y) to (negative, X)
- (positive, Z) to (negative, X)
- (negative, Z) to (negative, X)
- (positive, X) to (positive, Y)
- (negative, X) to (positive, Y)
- (positive, X) to (negative, Y)
- (negative, X) to (negative, Y)
- (positive, Z) to (negative, Y)
- (negative, Z) to (negative Y)
- (positive, X) to (positive, Z)
- (negative, X) to (positive, Z)
- (positive, Y) to (positive, Z)
- (negative, Y) to (positive, Z)
- (positive, X) to (negative, Z)
- (negative, X) to (negative, Z)
- (positive, Y) to (negative, Z)
- (negative, Y) to (negative, Z)

These are illustrated in the adjacent diagram



Deadlock Recovery

As an alternative to building networks that are deadlock-free by design, we can route the network traffic in an arbitrary way, and then recover from deadlock if it occurs. In order to do this, we need to:

- Detect deadlock
- Recover from deadlock

Deadlock detection

True deadlock detection would require global knowledge of traffic within the network, so instead we can use timeouts to detect when deadlock is likely to have occurred. The amount of time to wait before timeout is typically some multiple of a packet-time through a node, since at each collision, we must wait for the packet which has won the arbitration to pass entirely through the router.

Deadlock recovery

Deadlock recovery can be handled in a number of ways.

- Drop packet: We can simply drop deadlocked packets, until the deadlock condition clears. This solution may be acceptable for "best-effort" networks (eg. IP routers), but is often unacceptable for other kinds of interconnection networks in which reliable delivery is expected.
- Deflection: If we have infinite buffering at each node, we can "absorb" deadlocked packets into a node until the deadlock condition clears, and re-inject the packet into the network when the deadlock condition clears. However, networks are typically designed such that the network bandwidth is often much greater than the memory bandwidth at any given node, so it may be infeasible to implement this solution.
- Reserved channel: We can hold a strongly-connected deadlock free channel in reserve, and drain deadlocked packets through this channel until the deadlock condition clears.

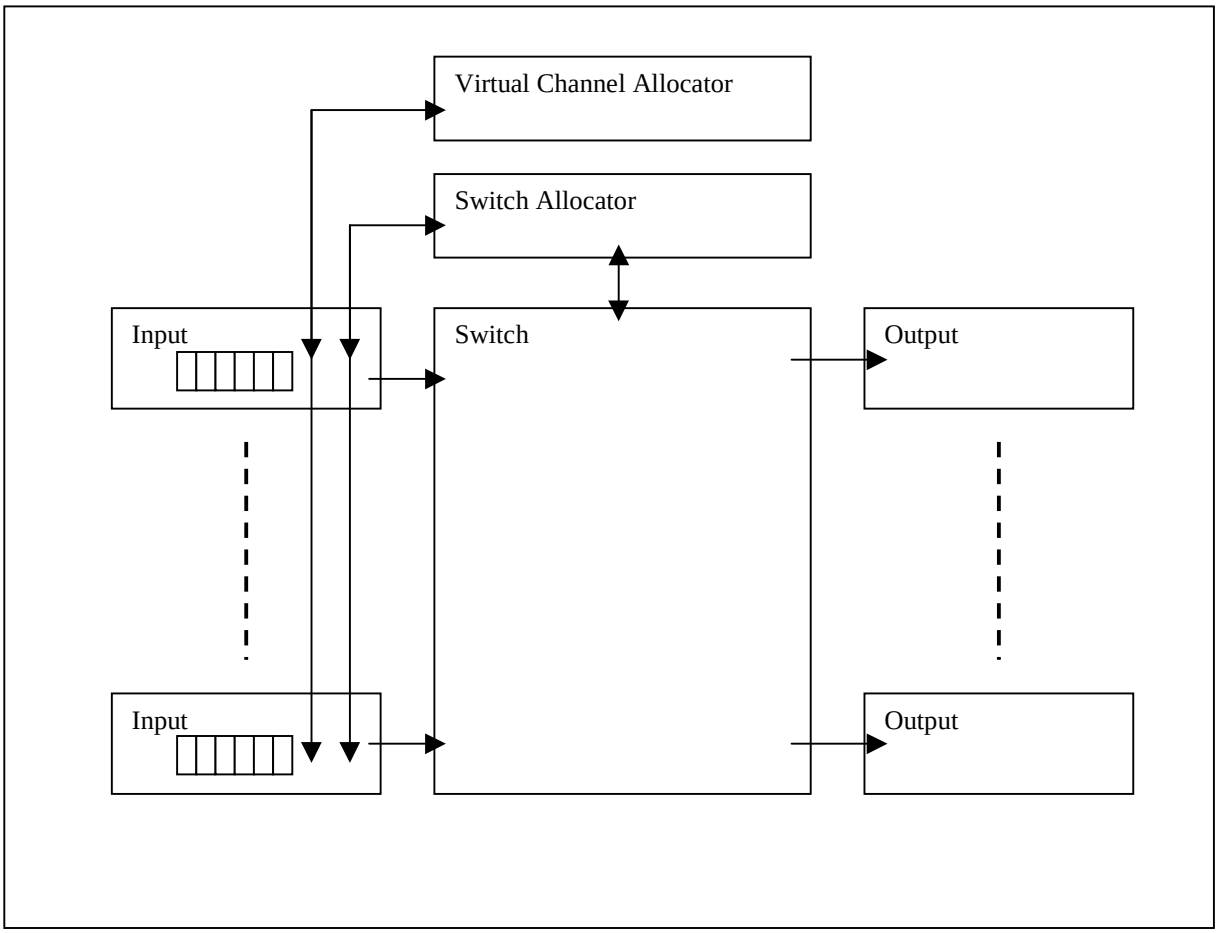
Deadlock recovery vs. Duato's algorithm

Deadlock recovery is similar to Duato's algorithm in that both make use of deadlock-free virtual channels to avoid deadlock failure. However, Deadlock recovery makes use of these channels only in the event of a timeout.

Router Architecture

A block diagram of a wormhole router is shown below. The router contains the following components:

- Input ports: The router consists of a series of input ports which contain buffering for incoming packets. Also associated with each input port is a state vector containing the following fields:
  - (G) Global state {Routing, Active, Idle}
  - (R) Routing info
  - (O) Output Port
  - (P) Input Buffer Pointer
  - (C) Output Credits available
- Output ports: Outgoing flits are received from the switch, staged, then sent out to the downstream router.
- Virtual Channel Allocator (VC Allocator): TheVirtual Channel Allocator arbitrates bids for output channels. Many implementations of switches locate the input port state vector registers within this block.
- Switch Allocator (SW Allocator): The Switch Allocator arbitrates switching requests to connect input ports to output ports and schedules the Switch.
- Switch: The switch is an interconnection network which connects input ports to output ports according to the configuration determined by the Switch Allocator



- At time 1, the packet header arrives at the input buffer (Buffer Pointer=1). The global state is set to "Routing" (G=R) and the virtual channel allocator examines the routing information in the header and determines whether any of the output channels which were bid on by the inbound packet are available. (We make the simplification that the determination of which outgoing channels to bid on is known during this cycle.)

- At time 2, the next flit has arrived (in this example, the Tail flit), so the Buffer Pointer is advanced (Buffer Pointer=2). The maximum capacity of the downstream buffer is recorded in the Credits field (eg. Credits=4). If the packet is allocated an output channel, the global state is changed to "Active" (G=R) and the allocated channel is recorded in the Output field (eg. C2, negative Y direction). In addition, the switch allocator determines whether the switch can be configured in the next timeslot to move data from the input buffer to the selected output port.

- At time 3, if the switch allocator has allocated a path through the switch and the Credit count is nonzero, the header is removed from the input buffer and sent through the switch to the output port, so we decrement the Buffer Pointer and decrement the available Credit count. Again, the switch allocator determines whether the switch can be configured in the next timeslot to move the next piece of data (Tail flit) from the input buffer to the selected output port.

- At time 4, if the switch allocator has allocated a path through the switch and the Credit count is nonzero, the data (Tail flit) is removed from the input buffer and sent through the switch to the output port. We record this activity by decrementing the Buffer Pointer and decrementing the available Credit count. Also at this time, the header flit passes out of the output port.

- At time 5, the tail flit passes out of the output port. At this point, the entire packet has passed through the router, but the acknowledgements to replenish the credit count back to maximum have not yet been

received. We must wait until the tail flit has been acknowledged before we can send new packets out of this channel. If we simply set the global state to "Idle" at this point, we would need some other way of recording the returning acknowledgements to know when the output channel is available. However, if we left the global state "Active" then we would needlessly be tying up an input port.

State Vector

Time	Global State	Routing Info	Output	Buffer Pointer	Credits
1	Routing	+/-		1	
2	Active	"	C2, Y(-)	2	4
3	"	"	"	1	3
4	Inactive**	"	"	0	2
5	Inactive**	"	"	0	2
6	Inactive**	"	"	0	2
7	Inactive**	"	"	0	1
8	Inactive**	"	"	0	0

Router Pipeline Diagram

Event	1	2	3	4	5	6
Virtual Channel Allocate	Header					
Switch Allocate		Header				
Switch			Header	Tail Flit		
Output				Header	Tail Flit	