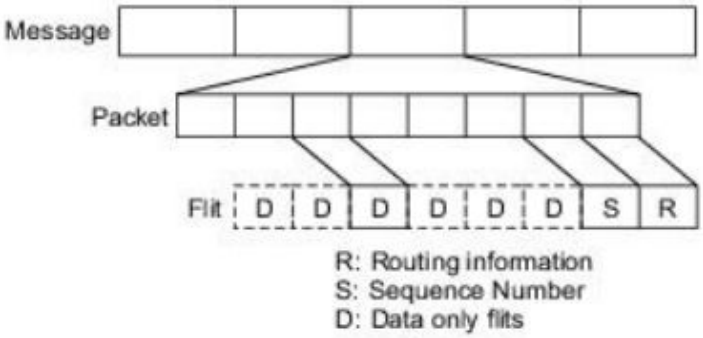
Module 4
Message passing mechanisms

4.1 Message Routing Schemes

Message formats

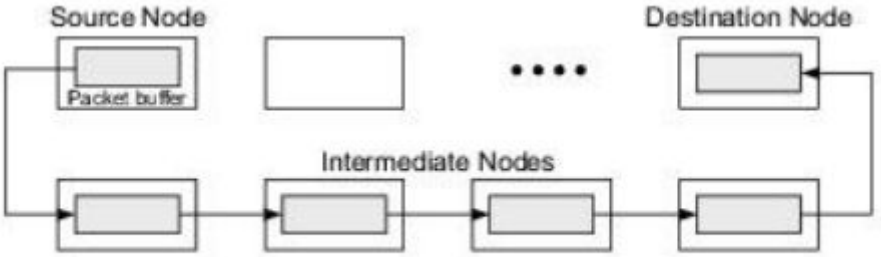
- A message is the logical unit for internode communication.
- A packet is the basic unit containing the destination address for routing purposes.
- Since different packets may arrive at the destination asynchronously, a sequence number is needed in each packet to allow reassembly of the message transmitted.
- A packet can be further divided into a number of fixed-length *flits* (flow control digits).
- Routing information (destination) and sequence number occupy the header flits. Remaining flits are data elements.
- Typical packet length range from 64 to 512 bits.



The format of message, packets, and flits (control flow digits) used as information units of communication in a message-passing network

Store- and - Forward Routing

- Packets are the basic unit of information flow in a store-and-forward network.
- Each node is required to use a packet buffer.
- When a packet reaches an intermediate node, it is first stored in the buffer. Then it is forwarded to the next node if the desired output channel and a packet buffer in the receiving node are both available.

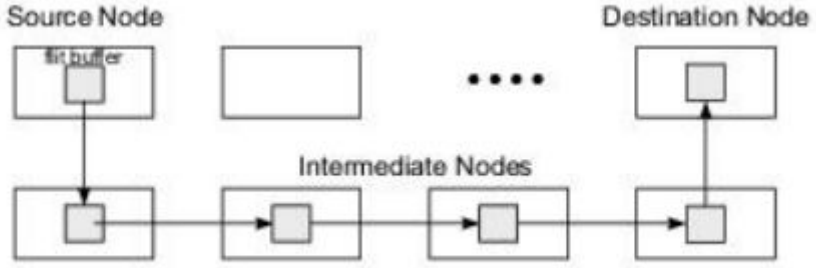


(a) Store-and-forward routing using packet buffers in successive nodes

Wormhole Routing

- Flits are the basic unit of information flow in a wormhole Routing.
- Flit buffers are used in the hardware routers attached to nodes.
- The transmission from the source node to the destination node is done through a sequence of routers.

- All the flits in the same packet are transmitted in order as inseparable companions in a pipelined fashion.
- Only the header flit knows where the packet is going. All the data flits must follow the header flit.



(b) Wormhole routing using flit buffers in successive routers

Asynchronous Pipelining

- The pipelining of successive flits in a packet is done asynchronously using a handshaking protocol.
- Along the path, a 1-bit *ready/request* (R/A) line is used between adjacent routers.

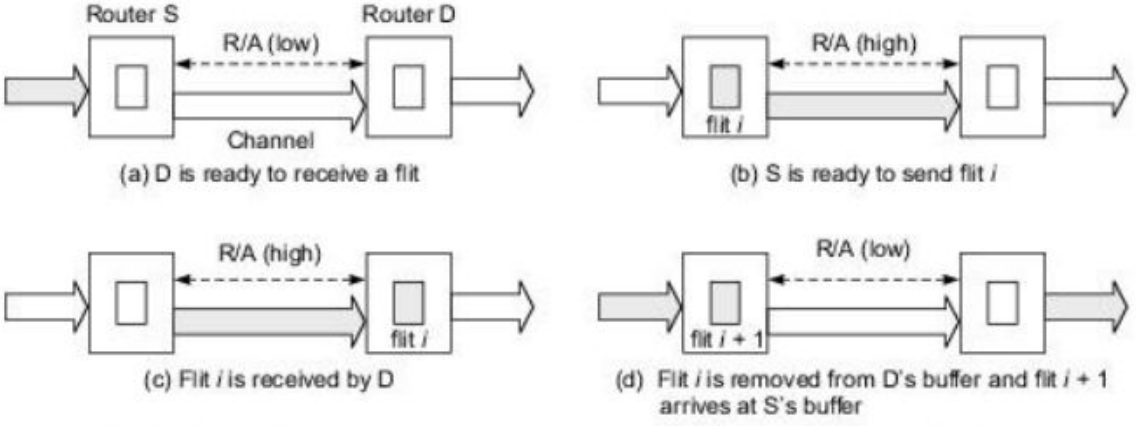


Fig. 7.28 Handshaking protocol between two wormhole routers (Courtesy of Lionel Ni, 1991)

Latency Analysis

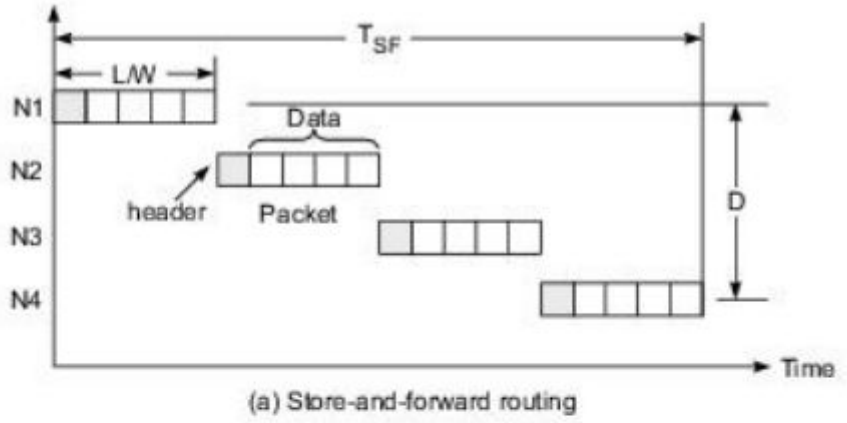
A time comparison between store-and-forward and wormhole routed networks is given in fig.

- Let L be the packet length (in bits), W the channel bandwidth (in bits/s), D the distance (number of nodes traversed minus 1), and F the flit length (in bits).
- The communication latency T_{SF} for a store-and-forward network is

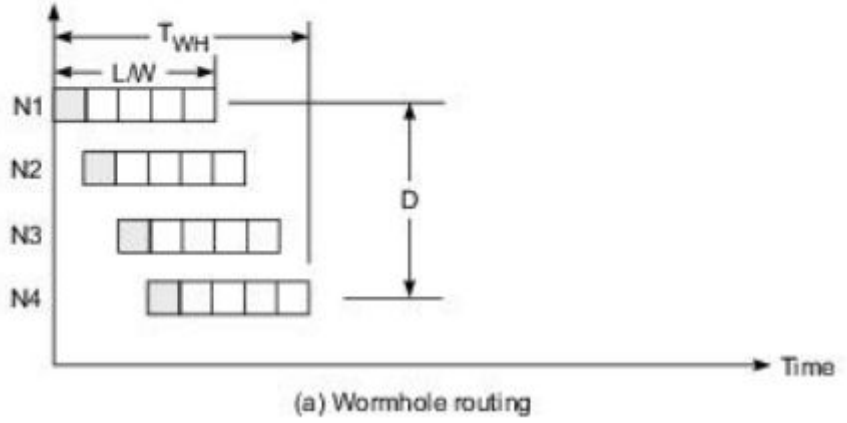
$$T_{SF} = \frac{L}{W}(D+1)$$

- The latency T_{WH} for a wormhole-routed network is

$$T_{WH} = \frac{L}{W} + \frac{F}{W} \times D$$



(a) Store-and-forward routing



(a) Wormhole routing

Fig. 7.29 Time comparison between the two routing techniques

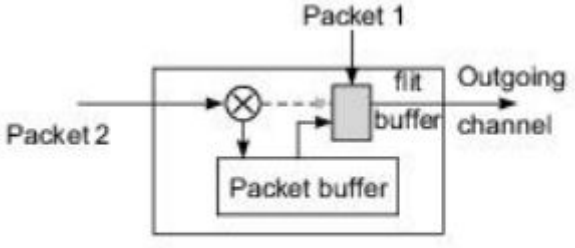
4.2 Flow Control Strategies

When two or more packets collide at a node when competing for buffer or channel resources, policies must be set regarding how to resolve the conflict.

Packet Collision Resolution

Four methods for resolving the conflict between two packets competing for the use of the same outgoing channel at an intermediate node.

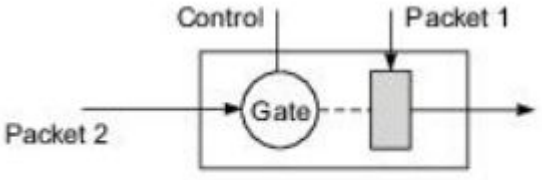
1. Virtual cut-through Routing scheme
    - Packet 1 is being allocated the channel
    - Packet 2 is temporarily stored in a packet buffer.
    - When the channel becomes available later, it will be transmitted then.
    - Advantage of not wasting the resources already allocated.
    - But requires the use of a large buffer to hold the entire packet.



(a) Buffering in virtual cut-through routing

2. Blocking policy

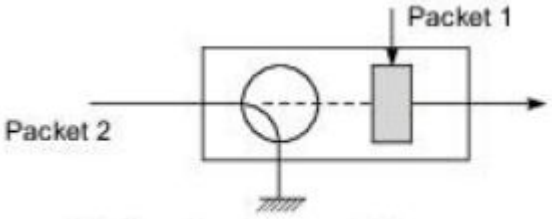
- Pure wormhole Routing uses a blocking policy in case of packet collision.
- The second packet is being blocked from advancing, however, it is not being abonded.



(b) Blocking flow control

3. Discard Policy

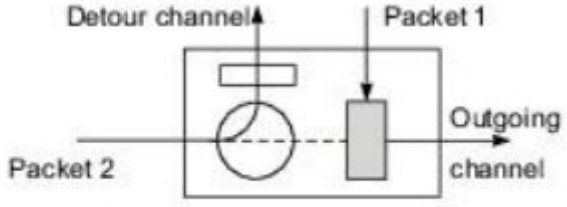
- It simply drops the packet being blocked from passing through.
- It may result in a severe waste of resources and it demands packet retransmission and acknowledgement.
- Otherwise the packet may be lost after discarding.



(c) Discard and retransmission

4. Detour routing

- The blocked packet is routed to a detour channel.
- This blocking policy is economical to implement but may result in the idling of resources allocated to the blocked packet.
- Detour routing offers more flexibility in packet routing.
- The detour may waste more channel resources than necessary to reach the destination.



(d) Detour after being blocked

Dimension- Order Routing

- Packet routing can be conducted deterministically or adaptively.
- In *deterministic routing*, the communication path is completely determined by the source and destination address. ie, the routing path is uniquely predetermined in advance, independent of network condition.
- *Adaptive routing* may depend on network conditions, and alternate paths are possible.
- Two such deterministic routing algorithms are given below, based on a concept called *dimension order routing.*

- Dimension order routing requires the selection of successive channels to follow a specific order based on the dimensions of a multidimensional network.
    1. In case of a two dimensional mesh network, the scheme is called *X-Y routing.*
    2. For hypercube networks, the scheme is called E-cube routing.

E-cube Routing on Hypercube

- Consider an *n*- cube with N = 2^n nodes.
- Each node *b* is binary coded as $b = b_{n-1}b_{n-2}....b_1b_0$
- Source node is $s = s_{n-1}....s_1s_0$
- Destination node is $d = d_{n-1}....d_1d_0$
- We want to determine a route from *s* to *d* with a minimum number of steps.
- We denote the *n* dimensions as i = 1,2, ..., n, where the ith dimension corresponds to the ($i-1$)st bit in the node address.
- Let $v = v_{n-1}....v_1v_0$ be any node along the route.
  The route is uniquely determined as follows:
    1. Compute the direction bit r_i = $s_{i-1} \oplus d_{i-1}$ for all *n* dimensions (*i = 1,....,n).* Start the following with dimension *i*= 1 and $v = s.$.
    2. Route from the current node v to the next node $v \oplus 2^{i-1}$ if r_i =1. Skip this step if r_i = 0.
    3. Move to dimension $i+1$ (ie $i \rightarrow i+1$). If $i \leq n$, go to step 2, else done.

Example: E- cube routing on a four - dimensional hypercube

n =4

s = 0110

d = 1101

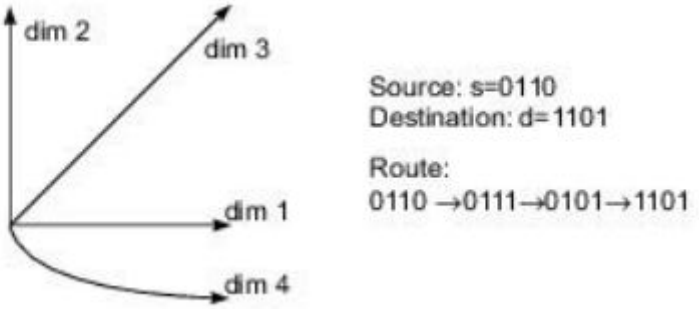
r = $r_4r_3r_2r_1$ = 1011

Route from *s* to $s \oplus 2^0$ = 0111 since $r_1 = 0 \oplus 1 = 1$.

Route from *v* = 0111 to $v \oplus 2^1$ = 0101 since $r_2 = 1 \oplus 0 = 1$

Skip dimension *i*= *3* because $r_3 1 \oplus 1 = 0.$

Route from *v* = 0101 to $v \oplus 2^3$ = 1101 =*d* since r_4 = 1.



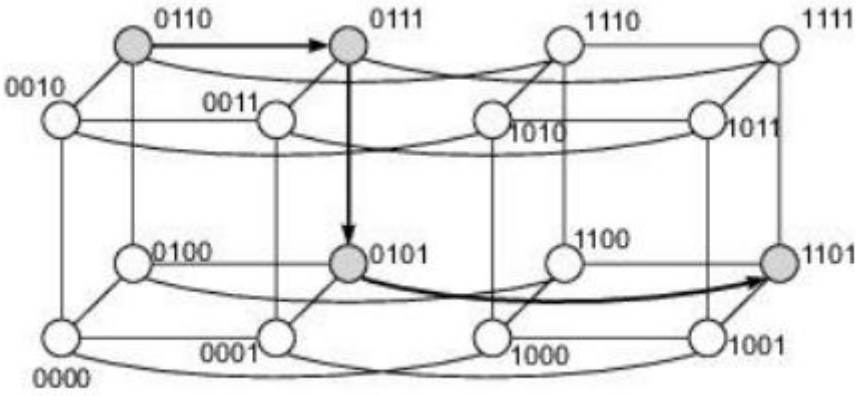


Fig. 7.34 E-cube routing on a hypercube computer with 16 nodes

X-Y Routing on a 2D Mesh

- From any source node $s = (x_1, y_1)$ to any destination node $d = (x_2, y_2)$,route from s along the X-axis first until it reaches the column y_2 , where d is located. Then route to d along the Y-axis.
- There are four possible X-Y routing patterns corresponding to the east-north, east-south, west-north, and west-south paths chosen.

Example: X-Y routing on a 2D mesh-connected multicomputer

- An east-north route is needed from node (2,1) to node (7.6). An east-south route is set up from node (0,7) to node (4,2). A west-south route is needed from node (5,4) to (2,0). The fourth route is west-north bound from node (6,3) to node (1,5).
- If the X-dimension is always routed first and then the Y-dimension, a deadlock or circular wait situation will not exist

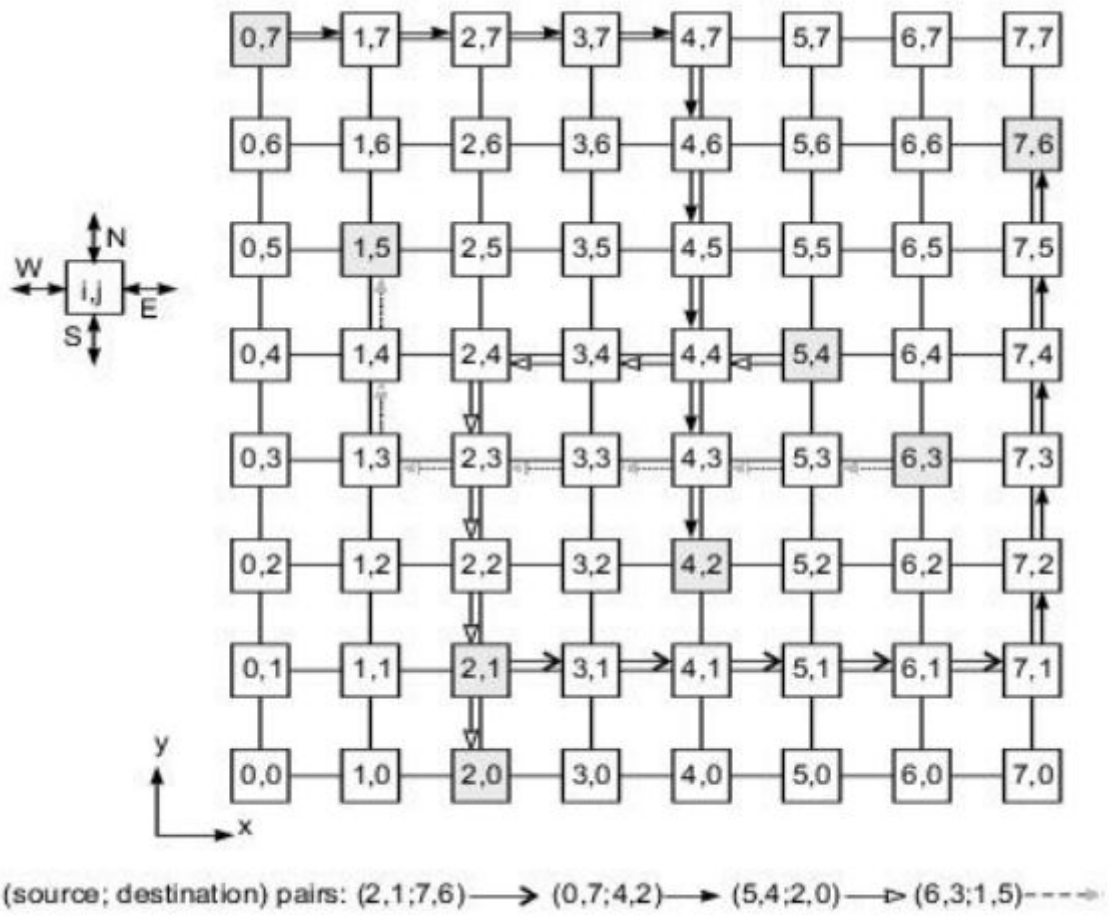


Fig. 7.35 X-Y routing on a 2D mesh computer with 8 × 8 = 64 nodes

Adaptive Routing

- The main purpose of using adaptive routing is to achieve efficiency and avoid deadlock.
- The concept of virtual channels makes adaptive routing more economical and feasible to implement.

Example : Adaptive X-Y routing using virtual channels

- This example uses two pairs of virtual channels in the Y-dimension of a mesh using X-Y routing.
- For westbound traffic, the virtual network in Fig. 7.36c can be used to avoid deadlock because all eastbound X-channels are not in use.
- Similarly, the virtual network in Fig 7.36d supports only eastbound traffic using a different set of virtual Y-channels.
- The two virtual networks are used at different times; thus deadlock can be adaptively avoided.

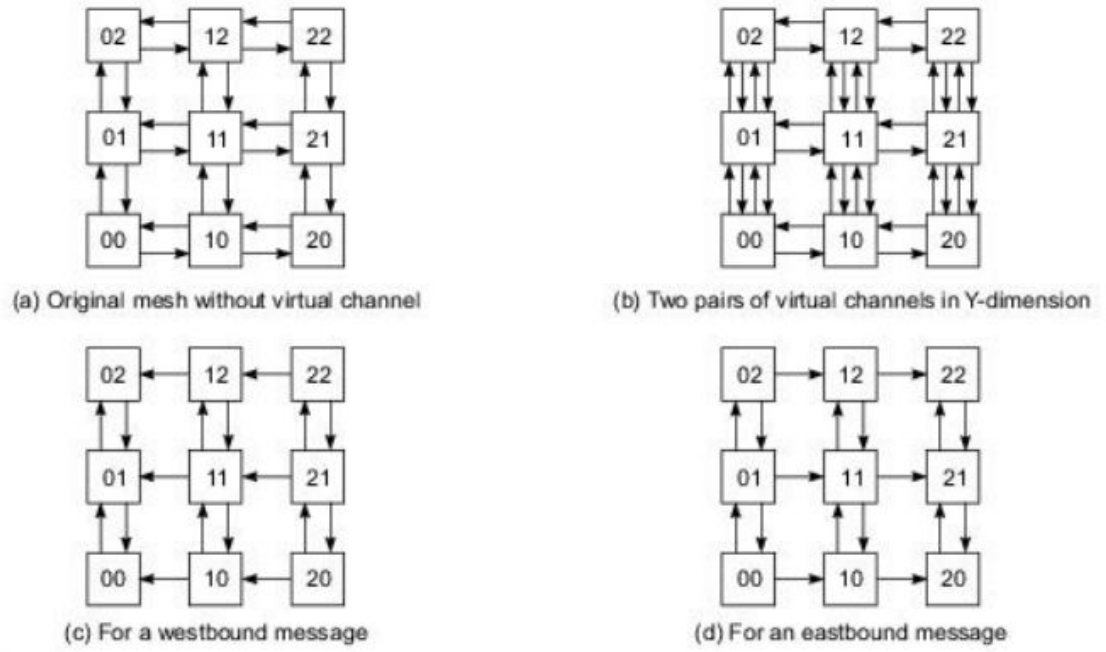
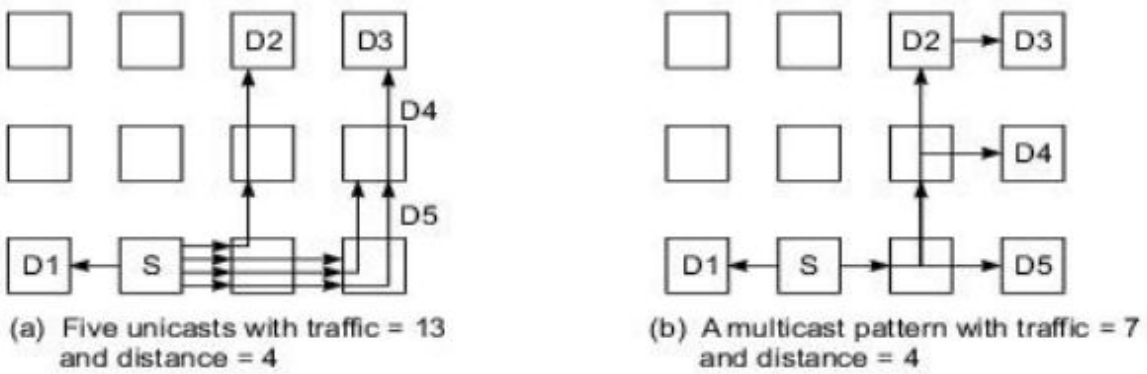


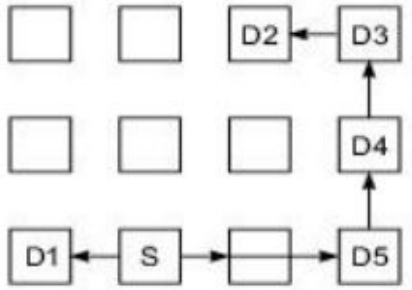
Fig. 7.36 Adaptive X-Y routing using virtual channels to avoid deadlock; only westbound and eastbound traffic are deadlock-free (Courtesy of Lionel Ni, 1991)

4.3 Multicast Routing Algorithms

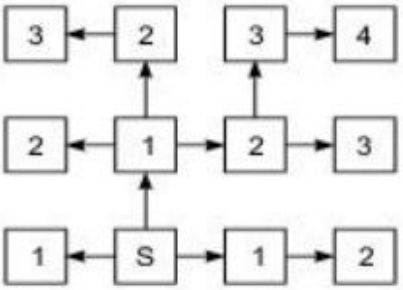
Four types of communication patterns may appear in multicomputer networks.

1. What we have implemented in previous sections is the one-to-one *unicast pattern* with one source and one destination.
2. A *multicast pattern* corresponds to one-to-many communication in which one source sends the same message to multiple destinations.
3. A *broadcast pattern* corresponds to the case of one-to-all communication.
4. The most generalized pattern is the many-to many *conference* communication.





(c) Another multicast pattern with traffic = 6 and distance = 5



(d) Broadcast to all nodes via a tree (numbers in nodes correspond to levels of the tree)

4.4 LINEAR PIPELINE PROCESSORS

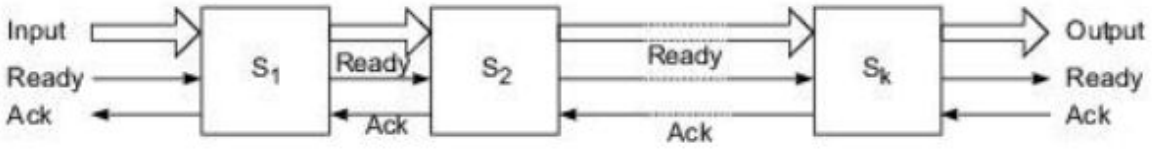
A linear pipeline processor is a cascade of processing stages which are linearly connected to perform a fixed function over a stream of data flowing from one end to the other.

Asynchronous and Synchronous Models

- A linear pipeline processor is constructed with k processing stages.
- External inputs (operands) are fed into the pipeline at the first stage S_1. The processed results are passed from stage S_i to stage S_{i+1}, for all i= 1, 2,...k-1.
- The final result emerges from thc pipeline at the last stage S_k.

Depending on the control of data flow along the pipeline, we model linear pipelines in two categories: *asynchronous* and *synchronous.*

Asynchronous Model

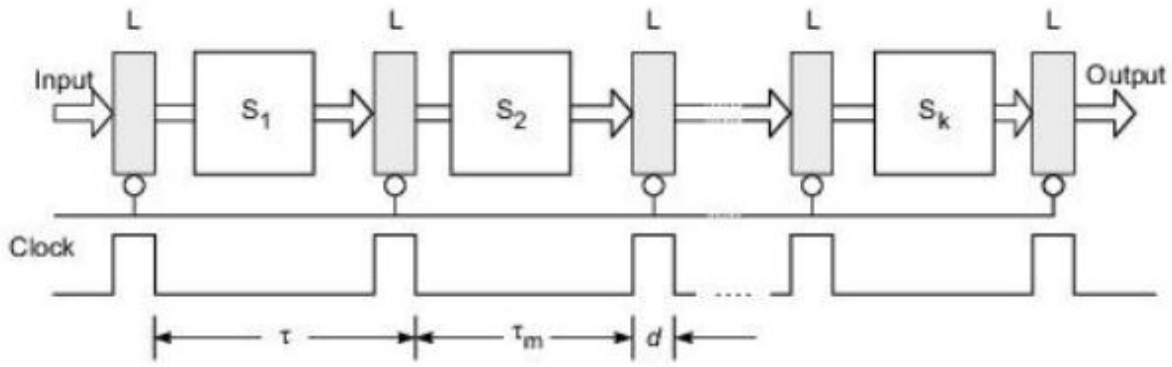


(a) An asynchronous pipeline model

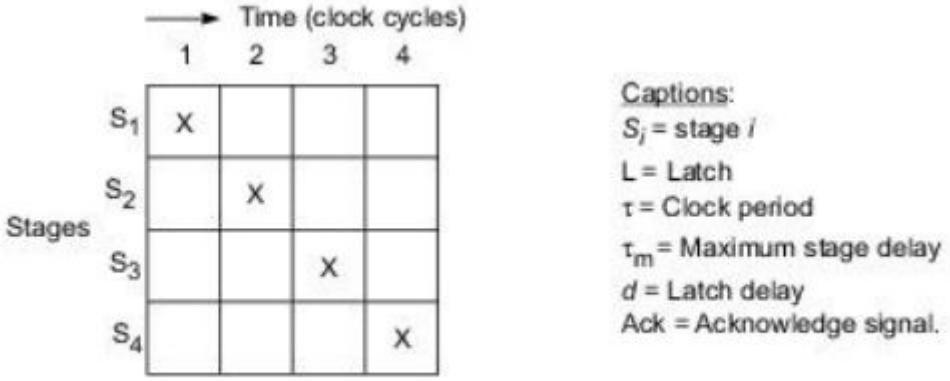
- data flow between adjacent stages in an asynchronous pipeline is controlled by a handshaking protocol.
- When stage S_i is ready to transmit, it sends a ready signal to stage S_{i+1}.
- Afler stage S_{i+1} receives the incoming data, it returns an *acknowledge* signal to S_i.
- Asynchronous pipelines may have a variable throughput rate.
- Different amounts of delay may be experienced in different stages.

Synchronous Model

- Clocked latches are used to interface between stages.
- The latches are made with master-slave fiip-flops, which can isolate inputs from outputs.
- Upon the arrival of a clock pulse, all latches transfer data to the next stage simultaneously.
- The pipeline stages are combinational logic circuits. It is desired to have approximately equal delays in all stages. These delays determine the clock period and thus the speed of the pipeline.
- The utilization pattern of successive stages in a synchronous pipeline is specified by a *reservation table*.



(b) A synchronous pipeline model



(c) Reservation table of a four-stage linear pipeline

- For a linear pipeline, the utilization follows the diagonal streamline pattern.
- This table is essentially a space-time diagram depicting the precedence relationship in using the pipeline stages.

Clocking and Timing Control

- The clock cycle τ of a pipeline is determined below.
- Let τ_i be the time delay of the circuitry in stage S_i, and d the time delay of a latch.

Clock Cycle and Throughput

Denote the maximum stage delay as τ_m , and we can write τ as

$$\tau = \tau_m + d$$

The *pipeline frequency* is defined as the inverse of the clock period:

$$f = \frac{1}{\tau}$$

Clock Skewing Ideally, we expect the clock pulses to arrive at all stages (latches) at the same time. However, due to a problem known as *clock skewing*, the same clock pulse may arrive at different stages with a time offset of s. Let t_{max} be the time delay of the longest logic path within a stage and t_{min} that of the shortest logic path within a stage.

To avoid a race in two successive stages, we must choose $\tau_m \geq t_{max} + s$ and $d \leq t_{min} - s$ .These constraints translate into the following bounds on the clock period when clock skew takes effect:

$$d + t_{max} + s \leq \tau \leq \tau_m + t_{min} - s$$

In the ideal case s=0, $t_{max} = \tau_m$ and $t_{min} = d$. Thus, we have $\tau = \tau_m + d$

Speedup, Efficiency, and Throughput

Ideally, a linear pipeline of *k* stages can process *n* tasks in k +(n - 1) clock cycles, where k cycles are needed to complete the execution of the very first task and the remaining n -1 tasks require n -1 cycles. Thus the total time required is

$T_k = [k + (n-1)]\,\tau$

where τ is the clock period. Consider an equivalent-function nonpipelined processor which has a *flow-through delay* of $k\tau$. The amount of time it takes to execute n tasks on this nonpipelined processor is $T_1 = nk\tau$

Speedup Factor

The speedup factor of a k-stage pipeline over an equivalent non pipelined processor is defined as

$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau+(n-1)\tau} = \frac{nk}{k+(n-1)}$

Optimal Number of Stages

The optimal choice of the number of pipeline stages should be able to maximize the performance/cost ratio for the target processing load.

A pipeline *performance/cost ratio* [PCR] has been defined by

$PCR = \frac{f}{c+kh}$

c- cost of all logic states

h- cost of each latch

Efficiency andThroughput

Thc efficiency E_k of a linear k-stage pipeline is defined as

$E_k = \frac{S_k}{k} = \frac{n}{k+(n-1)}$

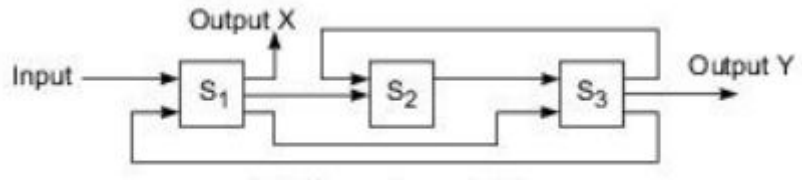
The pipeline throughput is defined as the number of tasks (operations) performed per unit time:

$H_k = \frac{n}{[k+(n-1)]\tau} = \frac{nf}{k+(n-1)}$

4.5 NONLINEAR PIPELINE PROCESSORS

- A *dynamic pipeline* can be reconfigured to perform variable functions at different times.
- The traditional linear pipelines are *static pipelines* because they are used to perform fixed functions.
- A dynamic pipeline allows feedforward and feedback connections in addition to the streamline connections.

Reservation and Latency Analysis

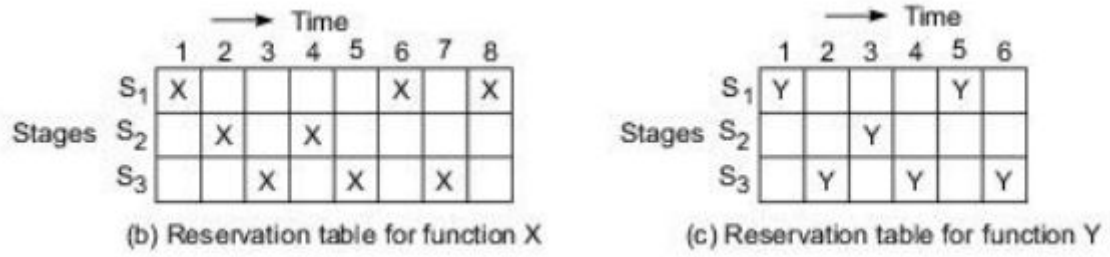


(a) A three-stage pipeline

- This pipeline has three stages. Besides the streamline connections from S1 to S2 and from S2 to S3, there is a feed forward connection from S1 to S3 and two feedback connections from S3 to S2 and from S3 to S1.
- The output of the pipeline is not necessarily from the last stage.

Reservation Table

- Each function evaluation is specified by one reservation table.
- A static pipeline is specified by a single reservation table.
- A dynamic pipeline may be specified by more than one reservation table.
- Each reservation table displays the time-space flow of data through the pipeline for one function evaluation.
- The no. of columns in a reservation table is called the evaluation time of a given function.



- Function X requires eight clock cycles to evaluate.
- Function Y requires six cycles.
- The checkmarks in each row of the reservation table corresponds to the time instants that a particular stage will be used.
- There may multiple checkmarks in a row, which means repeated usage of the same stage in different cycles

Latency Analysis

- The no. of time units (clock cycles) between two initiations of a pipeline is the latency between them.
- A latency of k means that 2 initiations are separated by k clock cycles.
- Any attempt by 2 or more initiations to use the same pipeline stage at the same time will cause a collision.
- A collision implies resource conflicts between two initiations.
- All collisions must be avoided in scheduling a sequence of pipeline initiations.
- Latencies that cause collisions are called forbidden latencies.

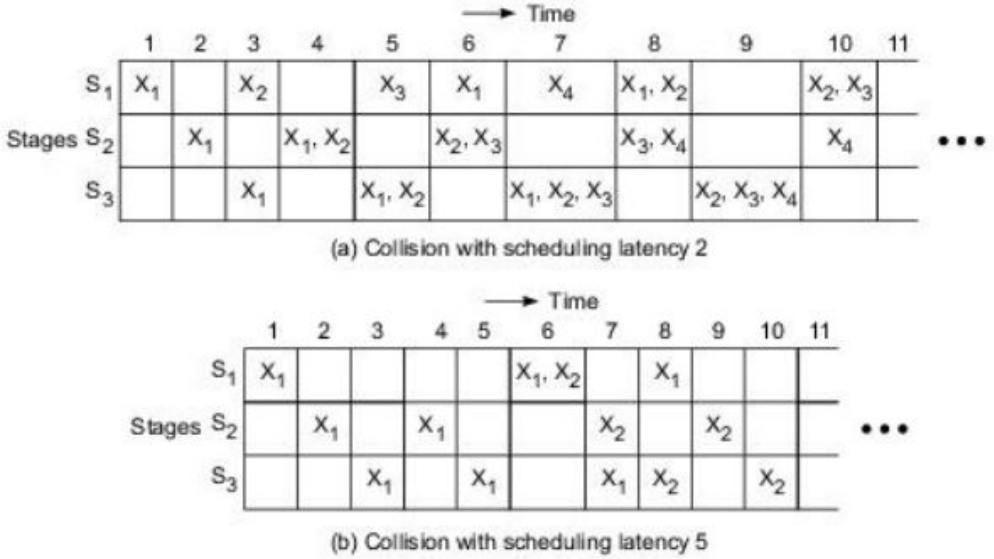


Fig. 6.4 Collisions with forbidden latencies 2 and 5 in using the pipeline in Fig. 6.3 to evaluate the function X

- In using the pipeline to evaluate the function X, latencies 2 & 5 are forbidden.
- To detect a forbidden latency, one needs simply to check the distance between any 2 checkmarks in the same row of the reservation table.
- Eg:- similarly latencies 2,4,5 & 7 are all forbidden.
- Forbidden latencies for function Y are 2 & 4.
- A latency sequence is a sequence of permissible non-forbidden latencies between successive task initiations.
- A latency cycle is a latency sequence which repeats the same subsequence (cycle) indefinitely.
  Eg:- (1,8) is a latency cycle
- The average latency of a latency cycle is obtained by dividing the sum of all latencies by the no. of latencies along the cycle. Eg:- avg latency of (1,8) = (1+8)/2 = 4.5
- A constant cycle is a latency cycle contains only one latency value
  Eg:- cycles (3) & (6) are both constant cycles.

Collision free scheduling

- When scheduling events in a nonlinear pipeline, the main objective is to obtain the shortest avg latency between initiations without causing collisions.

Collision Vectors

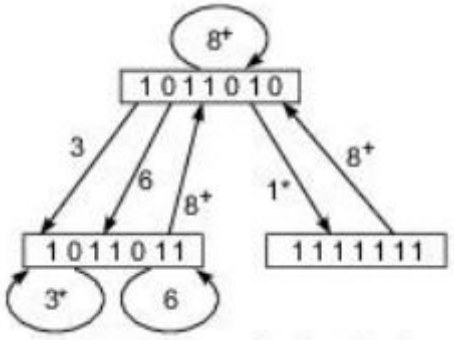
- For a reservation table with n columns the maximum forbidden latency m $\leq$ n-1
- The combined set of permissible & forbidden latencies can be easily displayed by a collision vector, which is an m-bit binary vector $C = (C_m C_{m-1} .... C_2 C_1)$
- The value of $C_i = 1$ if latency causes a collision and $C_i = 0$ if latency i is permissible.
- $C_m = 1$ in all states.

Eg:- In reservation table for X, the latencies are 2,4,5,7
Maximum forbidden latency =7, So m=7
So there will be 7 bits in the collision vector.
ie, $C_X = (1011010)$
Eg:- In reservation table for Y, the latencies are 2 & 4
Maximum forbidden latency =4, So m=4
So there will be 4 bits in the collision vector.
ie, $C_X = (1010)$

State Diagram

- A state diagram will specify the permissible state transitions among successive initiations.
- The collision vector, C_X corresponds to the initial state of the pipeline at time 1 and thus is called an initial collision vector.
- Consider the initial state 1011010
- Find out the permissible latencies in the collision vector. ie 1,3,6
- The new state can be obtained with the permissible latency 1. ie.by shifting the initial state 1 bit to the right & OR it with the initial state. After shifting 0101101
- After shifting, logical 0 enters from the left end of the shift registers.
- New state is calculated by
  0101101
  1011010
  1111111

- The new state contains only forbidden latencies. So no further proceedings is possible.
- With latency 3
  1011010
  0001011
  1011011
- With latency 6
  1011010
  0000001
  1011011
- In the new state the permissible latencies are 3 & 6
- So with latency 3
  0001011
  1011010
  1011011
- With latency 6
  1011010
  0000001
  1011011



(b) State diagram for function X

Note

- The bitwise ORing of the shifted version of the present state with the initial collision vector is meant to prevent collisions from future initiations.
- When the number of shifts is m + 1 or greater, all transitions are redirected back to the initial state.
- For example, after eight or more shifts (denoted as 8^+), the next state must be the initial state, regardless of which state the transition starts from.

Greedy Cycles

- Latency cycles from the state diagram are (1,8),(1,8,3,8), (3),(6),(3,8),(3,6,3).....
- A simple cycle is a latency cycle in which each state appears only once.
- eg:--(3), (6), (8), (1, 8), (3, 8), and (6,8) are simple cycles.
- The cycle (1,8,6,8) is not simple because it travels through the state (1011010) twice.
- Some of the simple cycles are greedy cycles.
- A greedy cycle is one whose edges are all made with minimum latencies from their respective starting states.
- For example, the cycles (1, 8) and (3) are greedy cycles.

- ie, Consider the initial state as the starting state. The minimum latency is 1. So (1,8) is a greedy cycle.
- If the starting state is 1011011 the minimum latency is 3. So (3) is a greedy cycle.
- The average latency of greedy cycle must be lower than those of other simple cycles.
- Average latency of greedy cycle (1,8) is (1+8)/2 = 4.5
- Average latency of simple cycle (6,8) = (6+8)/2 = 7
- Minimum average latency (MAL) is calculated by:
    1. Find the greedy cycles
    2. Find the average latency of the greedy cycles.
    3. Find the minimum value in the average latencies ie the MAL

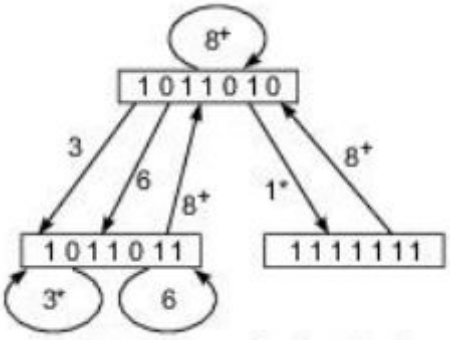
    Eg:- avg latency of (1,8) = 4.5

    Avg latency of (3) = 3

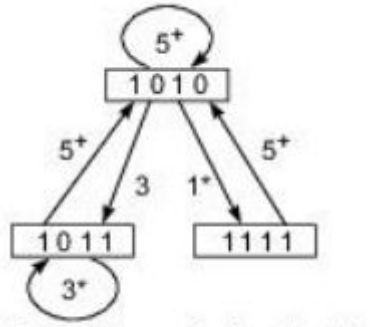
    Minimum average latency = 3

Note

The minimum-latency edges in the state diagrams are marked with asterisks.



(b) State diagram for function X



(c) State diagram for function Y

Pipeline Schedule Optimization

- The purpose is to yield an optimal latency cycle, which is absolutely the shortest.

Bounds on the MAL

1. The MAL is lower-bounded by the maximum number of checkmarks in any row of the reservation table.
2. The MAL is lower than or equal to the average latency of any greedy cycle in the state diagram.
3. The average latency of any greedy cycle is upper-bounded by the number of 1‘s in the initial collision vector plus 1. This is also an upper bound on the MAL.

Problem

Consider the following pipeline reservation table.

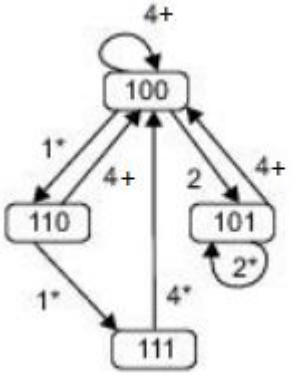
	1	2	3	4
S1	X			X
S2		X		
S3			X	

A. What are the forbidden latencies?

B. Draw the state transition diagram.
C. List all the simple cycles and greedy cycles.
D. Determine the optimal constant latency cycle and the minimal average latency.
E. Let the pipeline clock period be $\tau = 20ns$. Determine the throughput of this pipeline.

Answer

A. Forbidden latency = 3
B. Collision vector $C_X = C_3C_2C_1 = 100$
   Permissible latencies are 1 & 2



C. Simple Cycle are (2), (4), (1,4), (1,1,4) and (2,4).
   Greedy cycle are (2) & (1,4), (1,1,4)
D. Optimal constant latency cycle is (2), MAL = 2.
E. Pipeline clock period, $\tau = 20ns$ & MAL = 2
   Pipeline produces each output in 2 x 20ns = 40ns
   Throughput is the output of the pipeline in 1 sec
   ie, $\frac{1}{40\times10^{-9}} = \frac{10^9}{40} = 25 \times 10^6 = 25MIPS$