Interconnection networks

Flow control
Routing
Addressing
Switch design
Performance

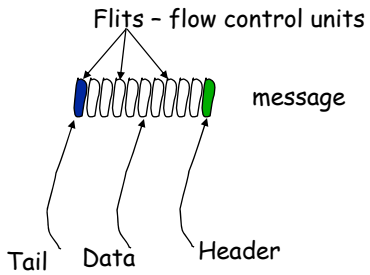
6.173
Fall 2010
Agarwal

Outline

- Topology
  - Arrangement of nodes, edges (switches, wires)
- Flow control
  - Allocation of node, channel resources
- Routing
  - Choosing paths
- Addressing and switch design
  - Mapping of names to physical locations
- Performance
  - Latency
  - Bandwidth
  - Efficient 2D layout

Flow Control (or Switching)

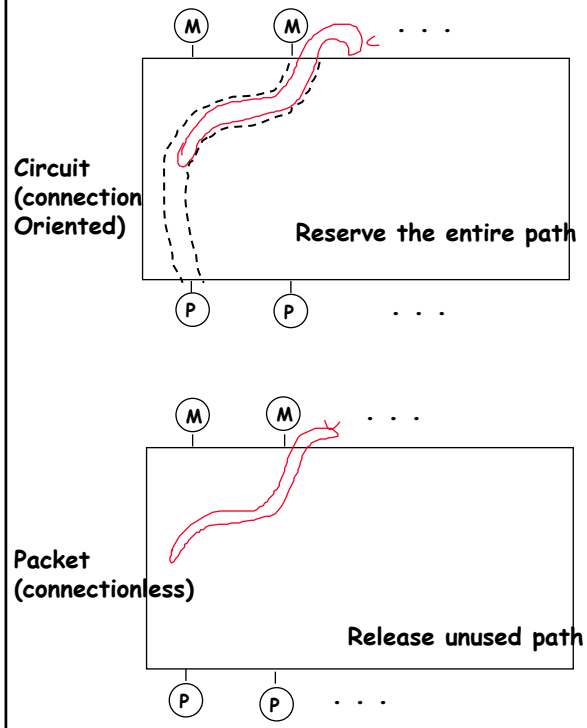
Notation



We will denote a message using this cartoon

Flow Control

Circuit-switched vs. Packet-switched

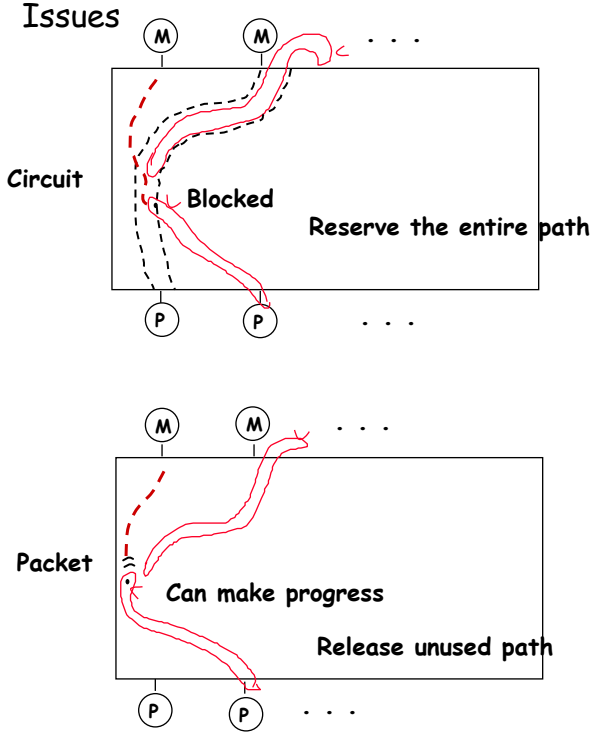


- 5 -

Flow Control

Circuit-switched vs. Packet-switched

Issues

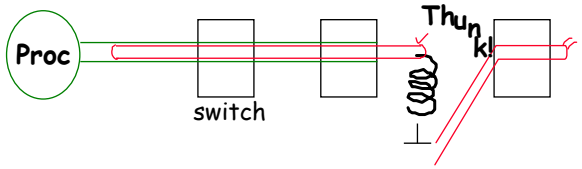


- 6 -

Flow Control

What do you do when you block?

- Dropping or non blocking
  - Commonly used with circuit switching



  - Reserve resources along entire path from source to destination
  - Ack has reserved return path!
  - Drop when blocked (non-blk network)
  - Tail can reverse path for NACK along the reserved backpath
  - Retry - can use backoffs
  - No need for buffering, packet is always moving forward (even if it is to nowhere!)

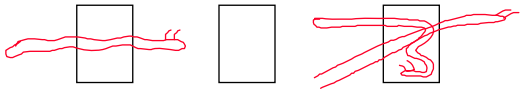
Flow Control

Mostly packet switched - release resources

- Store & forward

  - Always store msg. completely at each node, even if outport is free. Then send

- Cut through



  - Do not store message if outport is free; buffer only if outport is busy

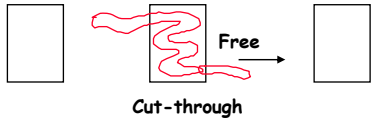
Two variants of cut through

1. virtual cut through: flow control on packet
2. pipelined or wormhole: flow control on flits

Flow Control

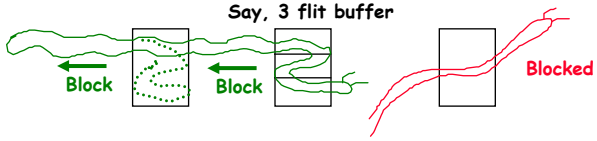
Cut through approaches

- 1. Virtual cut-through (no backward flow control)



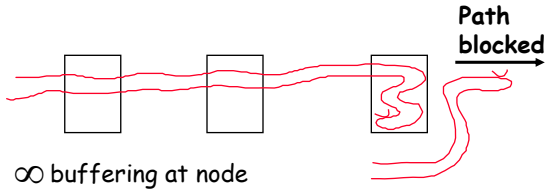
Cut-through

  - Send out as soon as outport is free
  - Packet-level flow control
  - Cannot stop packet halfway; need enough buffer space for whole pkt -- no blocking
- 2. Cut-through with backward flow control (piplelined, wormhole) works with finite buffers
  - Flit level flow control

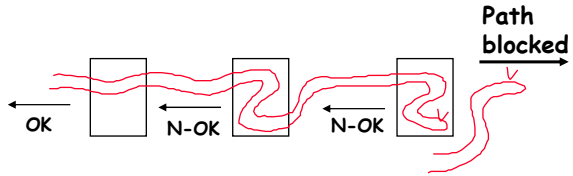


Flow Control

Cut through summary: two ways of message blocking



∞ buffering at node



Partial buffering in node, partial over network
Need backward flow-control signals

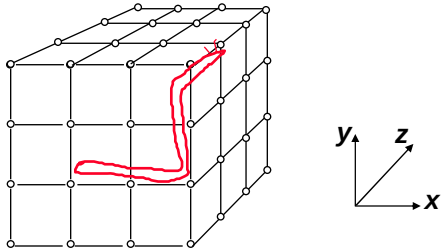
Routing

- Fixed
  - Packets always take the same predetermined path from source to destination
  - Simple
  - Easier to avoid deadlock
    - E.g. e-cube (aka dimension ordered)
    - Deadlock free when no end-around connections
  - Unbalanced loads
- Adaptive
  - Respond to network load
  - Uniform channel loading
  - Out of order packets
  - Be careful about deadlock

Routing

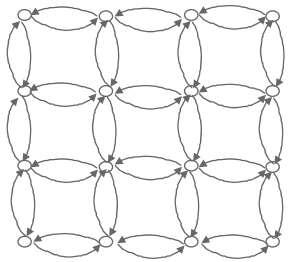
For k-ary n-cubes

E-cube routing (aka dimension ordered routing)



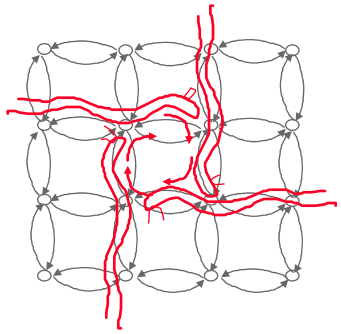
Route in dimension *n*, then *n*-1, *n*-2 ... 1

Deadlock Avoidance

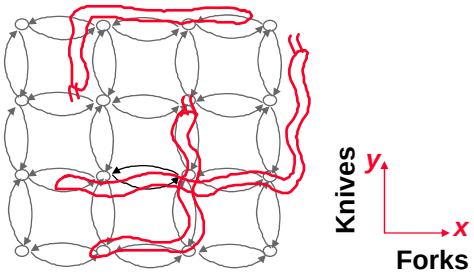


Deadlock Avoidance

Oblivious routing can deadlock



No end-around connections, E-cube Works!



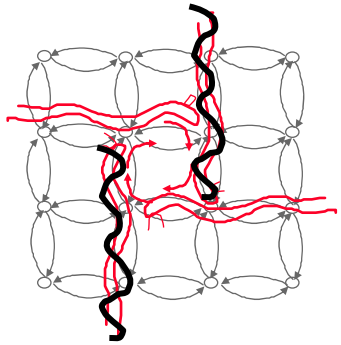
Key: Impose order on how messages acquire resources
e.g.: first *x*, then *y*

- No cycles in dependency graph.

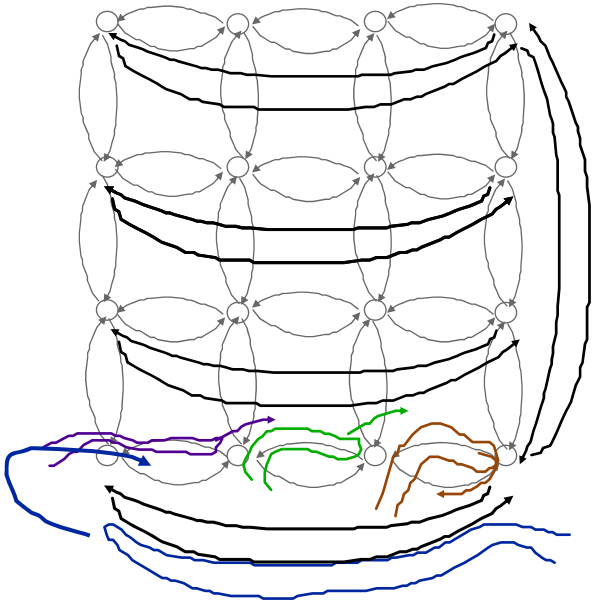
Remember dining philosophers!

Deadlock Avoidance

Which of these routes does e-cube disallow?



With end-around, E-cube No!



What about in a ring network?

Suppose each message wants to go 2 hops clockwise

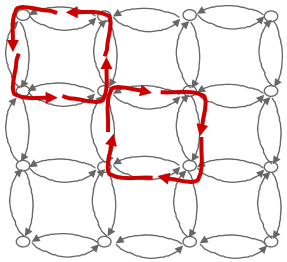
We need a more general framework for reasoning about deadlocks

A General Theory for Deadlock Avoidance in k-ary n-cubes

Problem with previous schemes:

- Not general, ad hoc
- Too restrictive, did not allow adaptive routing

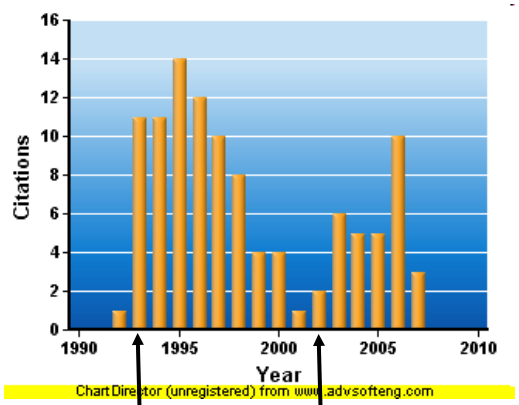
The turn model (Glass, Ni)



Idea: Break cycles by disallowing certain turns (and make sure that disallowed turns cannot be accomplished by 3 turns in allowed directions)

An Interesting Stat from Citeseer

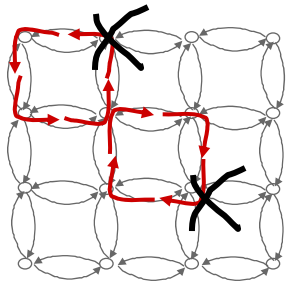
References to Glass and Ni's paper



Parallel computing is getting hot

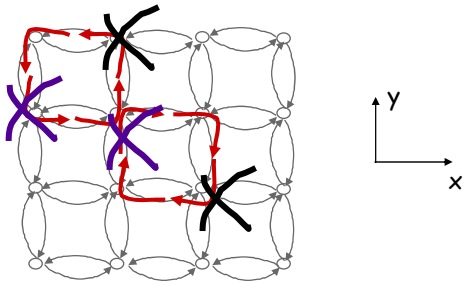
Multicore is getting hot

E.g., west-first routing



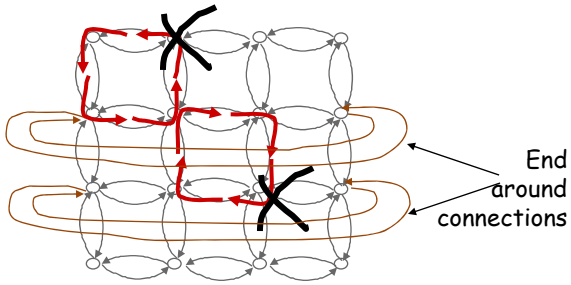
Proof involves, like dining philosophers, showing that it is possible to construct an ordering of all the channels (by numbering them with a tuple x,y, for example) so that the routing always takes place along strictly decreasing numbers

E-cube routing (or x-y routing)



x-y works, but is overkill!

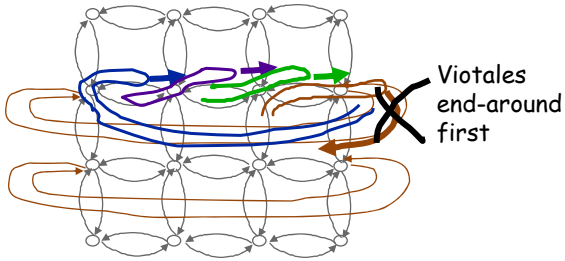
Generalizes to networks with end-around connections



Extend west-first, for example.
You are allowed to grab an end-around connection only as your first hop

Proof: Note that the west-first mesh is deadlock free. Since end-around is grabbed first, messages leaving end-around will eventually enter mesh because mesh is deadlock free.

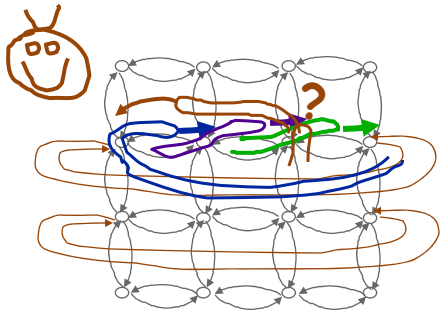
Generalizes to networks with end-around connections
A simple ring example



Suppose each message wants to go to the node that is 2 hops clockwise

Extend west-first, for example.
You are allowed to grab an end-around connection only as your first hop

Generalizes to networks with end-around connections
A simple ring example



Suppose each message wants to go to the node that is 2 hops clockwise

Extend west-first, for example.
You are allowed to grab an end-around connection only as your first hop

If energy and performance is all about wires

Deadlock and flow control is all about buffers