Routing

Lecture #6:	Monday, 23 April 2001
Lecturer:	Prof. Bill Dally
Scribe:	Mattan Erez, Brian Towles
Reviewer:	Kelly Shaw

1 Routing Algorithm Taxonomies

Routing algorithms can be characterized along two axes, representing how and when routing decisions are made.

How is the routing decision made?

- *Deterministic*: Given a source and destination node, the single path taken between them is completely defined. Deterministic routing does not contain path diversity and therefore is subject to poor performance on some traffic patterns.
- *Non-deterministic*: Instead of a single possibility, paths between a source and destination are chosen dynamically. The size of the set of possible paths is greater than one, and therefore path diversity exists. Further classification is possible considering how these paths are chosen dynamically:
  - *Oblivious*: Paths are randomly selected from a set of possibilities.
  - *Adaptive*: Additional state is used in the selection of a path. This information could include local channel loads, past path choices, buffer availability, etc. As a rule of thumb, "global" information is of little use because assembling this information takes time and would therefore be stale when used in a routing decision. The network changes so rapidly that this stale information would give little or no indication of the current network state.

When is the routing decision made?

- *All-at-once (source-routing)*: The path of a message is determined completely at the source node. All-at-once routing reduces the number of routing decisions to one per packet, but prevents the use of state collected at other nodes of the network. This method is useful when there is not enough time to perform a routing decision at every hop.

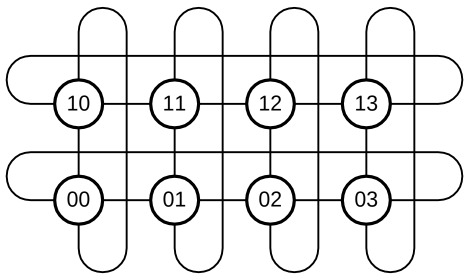

Figure 1: 2,4-ary 2-cube

- *Incremental*: Only the next hop of a message is determined at a particular node. With incremental routing, the number of routing decisions per packet equals the number of hops and state at each of the intermediate nodes can be used in the routing decision.

Using these taxonomies, the set of all routing relations R can be expressed as

$$R : \alpha \times \{N, C\} \times N \mapsto \{\mathbb{P}(\{P, C\}), \{P, C\}\}.$$

In the above expression, α is the network state, N is the set of all nodes, C is the set of all channels, and P is the set of all paths. So, for example, adaptive, all-at-once routing could be expressed as

$$R : \alpha \times N \times N \mapsto P.$$

And oblivious, incremental routing as

$$R : N \times N \mapsto \mathbb{P}(C).$$

2 Table-based Routing

Any routing relation can be stored in a table. The inputs to a routing relation form the index into the table, and the table entries stored at each of the indices correspond to the output(s) of the relation.

The in class explanation of table-based routing followed a simple example. Considering the network shown in Figure 1. A packet is being routed from node 00 to node 12. Noting that all four quadrants contain minimal routes, the total number of minimal routes between these nodes is

$$4 \cdot \binom{3}{2} = 12.$$

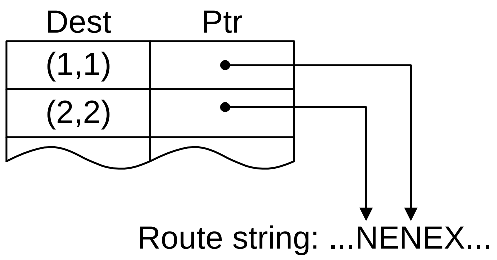

Figure 2: Encoding paths as pointers into a long route string

The routing table encodes a subset of the 12 possible routes to reduce storage requirements. This reduction also decreases path diversity, which in turn diminishes performance for some traffic patterns. It is therefore important that the routes stored in the table have little channel-overlap to allow for maximal diversity.

2.1 Table encoding

Tables can either designed for use with all-at-once (source) routing or incremental routing, where the latter requires a smaller amount of storage and is cheaper to implement:

- *Source table*: Used with all-at-once routing, entire paths are encoded in the route table. While this approach requires only one table access per packet, the storage required is on the order of the number of nodes in the system N. An alternative to storing a complete path per table entry is to instead store a pointer into a long string of channels. This encoding takes advantage of the fact that many paths may be suffixes of longer paths. For example, as shown in Figure 2, the sequences NEX and $NENEX$ can be derived from a longer string.

- *Node table*: Instead of storing the routing table at each terminal, the table can be distributed across the nodes. In a node table, only the next hop information is stored per network destination. While this can greatly reduce storage requirements, much of the flexibility of source routing is sacrificed. For example, once two paths destined to the same node share a link, the paths cannot diverge. In source routing, this is easily accomplished.

  Also, instead of performing a single routing look-up per packet, a node table must be accessed once per hop. This can greatly increase packet latency. To mitigate this latency problem, a *look-ahead* approach can be adopted. With look-ahead it is assumed that the packet already carries the next hop information when arriving at an intermediate node. The node table access then gives the next hop information not for the current node, but the following node. Effectively, the node table look-up has been pipelined. This removes the dependency between the node table access

Destination	Next Hop	Remarks
`100 100 100`	X	This node
`100 100 0XX`	W	Nodes directly west
`100 100 11X`	E	Nodes directly east
`100 100 101`	E	Neighbor east
`100 0XX XXX`	S	Nodes south
`100 11X XXX`	N	Nodes north
`100 101 XXX`	N	Nodes one row north
`0XX 0XX 0XX`	W	Octant down, south, and west
`0XX 0XX 1XX`	S	Octant down, south, and east (or equal)
`0XX 1XX XXX`	D	Quadrant down and north (or equal)
`11X XXX XXX`	U	Half-space two or more planes above
`101 XXX XXX`	U	Plane immediately above

Table 1: Hierchical routing table for node 444_8 of an 8-ary 3-cube.

and switch arbitration and therefore allows switch arbitration to begin as soon as a packet reaches a node. Look-ahead is employed in the SGI Spider chip.

- *Associative node table*: The size of node tables can be decreased further by employing a content addressable memory (CAM). Ranges can be captured by encoding the table entry addresses using trits: 0, 1, and X. The 'X' trit represents a wildcard or don't care and matches both 0 and 1. An example of such an encoding appears in Table 1. One line must be devoted to the exit path, and the others encode the rest of the routing information. Note that the masks must cover all possible destination addresses without allowing a multiple match on the current node address (exit path).

Finally, any table can contain multiple entries per address to increase path diversity. Either an oblivious or adaptive scheme can be used to chose between the set of possible paths or channels in these cases.

2.2 Non-minimal routes

Non-minimal routes can also be encoded into a table to aid in load balancing. However, if non-minimal routes are used with node-table routing, care must be taken to ensure the progress of packets. To illustrate this point, consider an incremental table for our example packet going from node 00 to 12. A directed graph can be constructed by drawing edges between nodes based on the next hops stored in the table. So, for our example, a packet starting at node 00, travelling to node 12 can either proceed north to node 10 or east to 01. By continuing to follow the possible paths of this packet, a complete graph can be constructed (Figure 3). In this example, a cycle exists: **node 01**$\xrightarrow{north}$node 11$\xrightarrow{north}$**node**

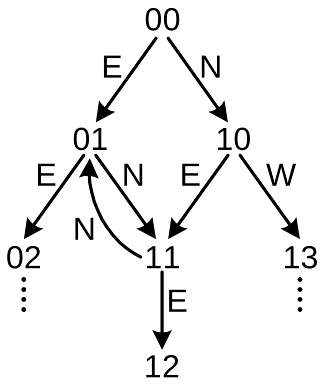

Figure 3: Directed routing graph from node 00 to node 12

01. Because of this loop, there is no guarantee that a packet travelling from node 00 to node 12 will progress.

Thus, loops can in the routing mechanism can create livelocks whereas deadlock can only be caused by the flow control mechanism. Additional state can be added to the routing table in order to avoid livelocked packets. For example, packets could be allowed to only take a fixed number of non-minimal hops before defaulting to a strictly minimal path.

2.3 Route Encoding

When performing source routing, the routing information must be transmitted with the head of a packet. This information can be encoded using a different symbol for every possible routing direction, including the exit direction. In order to allow for arbitrary path lengths a mechanism must be provided for the routing information to span multiple phits and flits. Thus, two more symbols are required - a *phit continuation* symbol to signal that the route continues on the next phit, and a *flit continuation* symbol for marking the continuation of the route on the following flit (Figure 4). Also, to reduce the size of the transmitted route various forms of compression can be used.

3 Algorithmic Routing

An alternative to table-based routing is to *compute* the next hop based on current information:

$$NextChannel = f\left(\substack{CurrentNode \\ PreviousChannel} , DestNode , \substack{\overline{r} \\ \alpha} \right)$$

α is additional load information used for adaptive routing and $\overline{r}$ is a set of random bits used for randomizing output channel selection. α and $\overline{r}$ are required for utilizing path

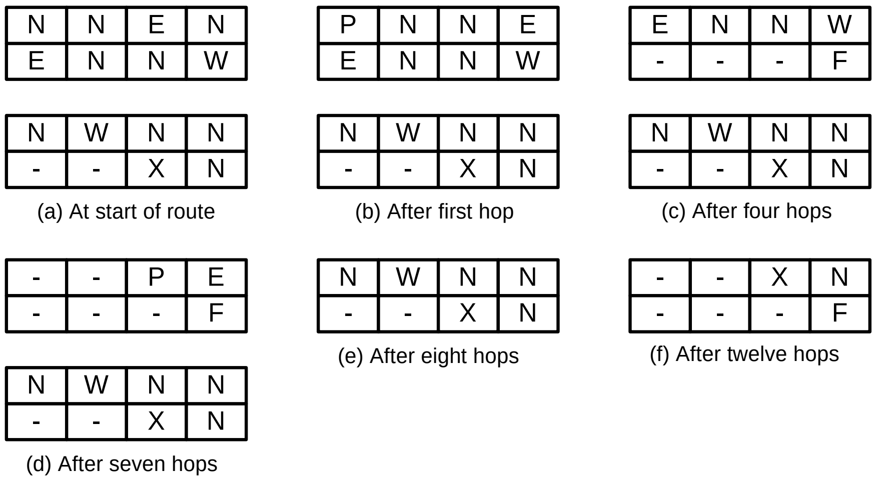

Figure 4: Arbitrary length encoding of source routes

diversity.

Example 1: Butterfly Network

The butterfly network uses *destination tag routing*, where the route is determined solely based on the destination:

$$NextChannel = f(DestNode)$$

A router based on this algorithm is much simpler than one based on a routing table.

Example 2: Extended Butterfly

In this network we can utilize path diversity by relying on load information or randomization:

$$NextChannel = f\left(DestNode,\ \overset{\overline{r}}{\alpha}\ \right)$$

Example 3: 2D Torus - Dimension Ordered Routing

This algorithm *deterministically* calculates a minimum path by routing along one dimension at a time.

$$NextChannel = f(CurrentNode, DestNode)$$

For example, routing first along the x-dimension, the router evaluates the following signals (where k_x and k_y are the number of nodes in the x and y dimensions respectively):

$$\begin{aligned}
\Delta x &= x_{dest} - x_{curr} \\
\Delta y &= y_{dest} - y_{curr} \\
West &= -\frac{k_x}{2} \le \Delta x < 0 \\
East &= 0 \le \Delta x < \frac{k_x}{2} \\
North &= (\Delta x = 0) \cap (0 \le \Delta y < \frac{k_y}{2}) \\
South &= (\Delta x = 0) \cap (-\frac{k_y}{2} \le \Delta y < 0) \\
Exit &= (\Delta x = 0) \cap (\Delta y = 0)
\end{aligned}$$

To allow for path diversity randomization can be used to encode all minimal paths simply. We studied three different algorithms for randomization:

1. The simplest implementation is to calculate the sign of Δx and Δy at every hop and to randomally choose a direction according to the sign of a single dimension. An improvement is to calculate a Δ vector at the start and update it on every hop.

   $$\Delta = (x_{dest} - x_{src}, y_{dest} - y_{src}) \mod (\frac{k_X}{2}, \frac{k_y}{2})$$

   Every hop, a random direction (towards the destination) is selected and the Δ vector is updated accordingly, until one of its elements is 0. Then routing proceeds similarly along all non-zero dimensions.

   When randomization is used an interesting property of an algorithm is the expected channel load for a particular source-destination pair. In the algorithm above, if the random direction selected at every hop is weighted so that all minimal paths are equi-probable, the expected load on a channel is given by the following analysis. In order for a channel $(x_{c,H}, y_{c,H})$ (the horizontal channel from node (x_c, y_c) to $(x_c + 1, y_c)$) to be traversed we must first reach the node (x_c, y_c) and the continue from the node $(x_c + 1, y_c)$.

   There are $\binom{x_c+y_c}{x_c}$ minimal paths from the source node $(0, 0)$ to (x_c, y_c), and $\binom{\Delta x - x_c - 1 + \Delta y - y_c}{\Delta x - x - 1}$ minimal paths continuing from $(x_c + 1, y_c)$ to the destination $(\Delta x, \Delta y)$. Finally, there are $\binom{\Delta x + \Delta y}{\Delta x}$ total minimal paths from the source to the

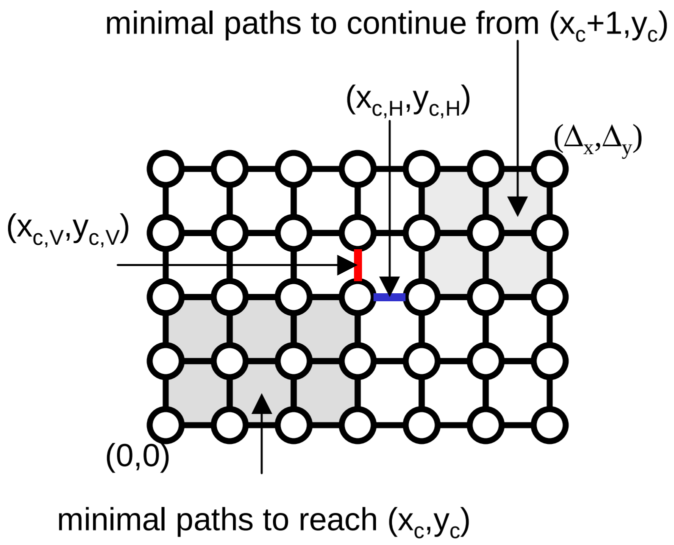
minimal paths to continue from (x_c+1,y_c)
(x_c,H,y_c,H)
(Δx,Δy)
(x_c,V,y_c,V)
(0,0)
minimal paths to reach (x_c,y_c)

Figure 5: Random selection of minimal path

destination, and the expected load on a horizontal channel is

$$load_{Horizontal} = \frac{\binom{x_c+y_c}{x_c}\binom{\Delta x - x_c - 1 + \Delta y - y_c}{\Delta x - x_c - 1}}{\binom{\Delta x + \Delta y}{\Delta x}}$$

Similarly, the expected load on a vertical channel is:

$$load_{Vertical} = \frac{\binom{x_c+y_c}{x_c}\binom{\Delta x - x_c + \Delta y - y_c - 1}{\Delta y - y_c - 1}}{\binom{\Delta x + \Delta y}{\Delta x}}$$

The resulting load distribution places greater load on the center channels.

2. A slightly different algorithm achieves uniform load on all center channels. Instead of utilizing all minimal paths, only a subset is used. Randomization is achieved by selecting an intermediate node from within the range $([0, \Delta x], [0, \Delta y])$ and using dimension ordered routing through it. In order to avoid a preferred direction in the dimension ordered routing, half the load is sent in the y-direction first, and half in the x-direction. A packet will go through the channel $(x_{c,H}, y_{c,H})$ on either of two conditions:

   (a) The intermediate node is (x_c, y_c) when routing x dimension first.

(b) The intermediate node is $(x_c + 1, y_c)$ when y is the preferred dimension.

So the probability of traversing $(x_{c,H}, y_{c,H})$, where $0 < x_c < \Delta x$, $0 < y_c < \Delta y$ (center channels) is

$$\begin{aligned}
Pr\{(x_{c,H}, y_{c,H})\} &= Pr\{[(Preferred = x) \cap ((y_{intermediate} = y_c) \cap (x_{intermediate} \le x_c))] \\
&\qquad \cup [(Preferred = y) \cap ((y_{intermediate} = y_c) \cap (x_{intermediate} > x_c))]\} \\
Pr\{(x_{c,H}, y_{c,H})\} &= \frac{1}{2} Pr\{[(y_{intermediate} = y_c) \cap (x_{intermediate} \le x_c)]\} \\
&\quad + \frac{1}{2} Pr\{[(y_{intermediate} = y_c) \cap (x_{intermediate} > x_c)]\} \\
Pr\{(x_{c,H}, y_{c,H})\} &= \frac{1}{2} Pr\{[y_{intermediate} = y_c] \cap [(x_{intermediate} \le x_c) \cup (x_{intermediate} > x_c)]\} \\
Pr\{(x_{c,H}, y_{c,H})\} &= \frac{1}{2} Pr\{y_{intermediate} = y_c\} \\
Pr\{(x_{c,H}, y_{c,H})\} &= \frac{1}{2\Delta y}
\end{aligned}$$

Similarly,

$$Pr\{(x_{c,V}, y_{c,V})\} = \frac{1}{2\Delta x}$$

And since we know the load on all the center channels ($load = probability$ for unit load at source), we can calculate the expected load on the perimeter channels as well. For example, the load on the bottom-left x-channel ($(0, 0)$) is $\frac{1}{2}$ since x is the preferred direction half the time. Every y-channel carries $\frac{1}{2\Delta x}$ so the load on the bottom x-channels drops linearly by $\frac{1}{2\Delta x}$ per channel.

$$\begin{aligned}
Pr\{(x_{c,H}, 0)\} &= \frac{1}{2} - \frac{x_c}{2\Delta x} \\
Pr\{(x_{c,H}, \Delta y)\} &= \frac{1}{2} - \frac{\Delta x - x_c - 1}{2\Delta x} \\
Pr\{(0, y_{c,V})\} &= \frac{1}{2} - \frac{y_c}{2\Delta y} \\
Pr\{(\Delta x, y_{c,V})\} &= \frac{1}{2} - \frac{\Delta y - y_c - 1}{2\Delta y}
\end{aligned}$$

3. Finally, to achieve perfect load-balance regardless of the communication pattern, a packet can be routed through a random node not restricted to the minimal path box. The downside of this approach is that the expected traffic crossing the bisection is doubled [L.G. Valiant, "Optimality of a two-phase strategy for routing in interconnection networks," *IEEE Transactions on Computers*, vol C-32, no. 9, September 1983].

4 Search Based Routing

A third mechanism for determining a route is to perform a search on the network, instead of using an algorithm or routing table. This approach is useful for fault tolerance, where it can be used to discover valid routes even in the presence of missing links. There are two classes of search methods:

- *Flood Search* - The network is *flooded* with small probe packets going from the source node to the destination node. When a probe reaches the destination node it send an acknowledge packet back to the source node, which contains the traversed path. The first acknowledgment packet that returns to the source-node determines the route. This method determines a good route relatively quickly, but generates considerable load on the network due to the probe packets.

- *Backtrack* - Here a scout packet is sent towards the destination. Each time it encounters a blocked channel, it backtracks until a free channel is found. As long as the network is not completely blocked, the scout will eventually reach the destination node and "report" back to the source node with the path. The downside of this algorithm is the possibly long search latency.