ROUTING TABLES FOR MESSAGE ROUTING IN DISTRIBUTED DOUBLE LOOP NETWORKS

Dulal C. Kar  
Department of Computing and Mathematical Sciences  
Texas A&M University-Corpus Christi  
Corpus Christi, TX 78412, USA  
dkar@falcon.tamucc.edu

V.V. Bapeswara Rao  
Department of Electrical and Computer Engineering  
North Dakota State University  
Fargo, ND 58105, USA  
Bapeswara.Rao@ndsu.nodak.edu

Abstract
Distributed double loop networks are very advantageous for the design and implementation of local area networks and parallel processing architectures, mainly due to their fault-tolerance, low communication latency, and regularity. In this paper, an algorithm for the formulation of routing tables that facilitate routing of messages in a distributed double loop network is presented. The algorithm takes $O(d)$ steps to formulate the routing tables, where $d$ is the diameter of the network. This algorithm is based on breadth first search technique [8]. It finds the distance table for a given node first and then computes the routing table from the distance table. In contrast with some similar works, knowledge of the diameter of the network is not required. The algorithm can also be applied to any similar graph with appropriate modification.

1 INTRODUCTION

The ring topology has been widely used for deployment of local area networks for its many inherent advantages. The topology is simple, regular, and flexible. A token passing protocol is used to control accesses of the nodes to the network for data transmission. The token passing protocol is very fair as the token circulates on the ring network through all nodes and each node gets a chance to grab the token once at each round and can transmit its data. However, the ring network is very vulnerable to link or node failures. The connectivity of a unidirectional link network is one. Any single fault in the ring can bring down the entire network as there is no alternative path to bypass the faulty link or node. In addition, in a large ring the communication latency between two farthest nodes can be too excessive. For a directional ring network of $n$ nodes, the distance between two farthest nodes, known as diameter, is $(n-1)$. The diameter of an undirected ring network of $n$ nodes is $\lceil n/2 \rceil$. By adding some extra links to a ring network in some systematic way, it is possible to improve its fault-tolerant capability and reduce the diameter at the same time. A distributed double-loop network topology is rooted on this basic idea.

A distributed double-loop network provides a fault-tolerant, expandable architecture for connecting computers in a local area environment or for connecting processors in a parallel or distributed processing environment [1]-[7]. The network also provides a structure to communicate among computers or processors with low communication latency. Let a directed graph $G(n, \pm 1, \pm h)$ represent a double-loop distributed network having $n$ nodes and $2n$ links, where each node $k$ is adjacent to 4 nodes, $k-1$, $k+1$, $k-h$, and $k+h$ (mod $n$) and $h$ is a positive number that represents a fixed link-jumps from node $k$ to its neighbors $k-h$ and $k+h$ and the nodes are labeled as $0, 1, 2, \ldots, n-1$. Figure 1 shows the architecture of a distributed double-loop network $G(8, \pm 1, \pm 3)$.

![Figure 1. A distributed double-loop network $G(8, \pm 1, \pm 3)$.](image)

Many properties of double-loop networks have become the focus of recent research in computer networks as well as in distributed multiprocessor systems. The aim of this paper is to present an algorithm for routing a message from node $p$ (source node) to node $q$ (destination node) via the shortest path. Particularly, the algorithm finds out all the shortest paths from a source node to all other nodes on the network in $O(d)$ steps (where $d$ is the diameter of the network) time steps. The algorithm is based on a distance table that it computes first and may be applied to any similar graph. With some simple transformations, any node can use the same routing table for message routing on the network. In contrast with earlier similar works [2], [4], [5], [9] our proposed
procedure is simple and finds not just one but all the shortest paths from a source node to all other nodes. In addition, in contrast with earlier works, the proposed procedure does not require any prior computation or knowledge of \(d\).

2 ROUTING TABLE

A node uses its routing table to decide to which neighbor it should forward a message so that the message reaches the destination by traversing the shortest possible path. When a message reaches node 5 on its way to node \(q\) \((q \neq 5)\), the node to which it can then be sent is, \(5–1 = 4, 5+1 = 6, 5–3 = 2, \) or \(5+3 = 8\). However, choosing any of the four neighboring nodes may lead some non-optimal path to the destination. Actual selection is based on the shortest path from node 5 to node \(q\). For that, a routing table of a node contains a column for destination nodes and a column for next nodes used for forwarding a message for a given destination over the shortest path. Figure 2 shows a routing table for node 5 in a double loop network \(G(10, \pm 1, \pm 3)\). When the "next node" column has more than one node, the routing node may pick any one of them for forwarding.

<table>
<thead>
<tr>
<th>Destination Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0)</td>
</tr>
<tr>
<td>Next Node</td>
</tr>
<tr>
<td>Destination Node</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>(5)</td>
</tr>
<tr>
<td>Next Node</td>
</tr>
</tbody>
</table>

Figure 2. Routing table for node 5.

Thus, when a message whose destination node is 7 arrives at node 5, it may be sent either to node 4, 6, or 8 for onward transmission, without violating the minimum distance criterion. In the following, we present how to obtain such routing table for a node.

3 COMPUTATION OF DISTANCE TABLE

It is convenient to obtain the distance table for a node first before computing its routing table. The distance table of a node can be used to find the minimum distance from the node to some node on the network. It is to be noted that a distance from the present node to some given node indicates how many links or hops away the given node is. We first provide the illustration on the algorithm for distance table computation of the double loop distributed network \(G(20, \pm 1, \pm 7)\) for node 0 and later we provide a general algorithm on distance table computation for node 0. From node 0, nodes \((0 – 1)\) mod 20, \((0 + 1)\) mod 20, \((0 + 7)\) mod 20, and \((0 – 7)\) mod 20 i.e., nodes 19, 1, 7, and 13 are one hop away and accordingly, we list them in the row for distance 1 in the distance table \(D\) as shown in Figure 3. Nodes 19, 1, 7, and 13 are also the next nodes (neighbors) for node 0. Accordingly, node 0 must forward its message destined to some other node either through 19, 1, 7, or 13. Therefore, its routing table will have 19, 1, 7, or 13 in the next node column for some given destination node.

In Figure 3 for the distance table \(D\) of node 0, we designate a column for each next node 19, 1, 7, and 13. Next we determine what new nodes can be reached one hop away from nodes 19, 1, 7, and 13 and list them in the row for distance 2 in the table. From node 19, we see new nodes \((19 – 1)\) mod 20, \((19 + 7)\) mod 20, and \((19–7)\) mod 20 i.e., new nodes 18, 6, and 12 can be reached and we list them in the cell of the column for 19 and the row for distance 2. These nodes are distance 2 away from node 0 and messages to them from node 0 are to be forwarded through node 19. Similarly, we find the new nodes one hop away from node 1, 7, and 13 and fill the cells of the row of distance 2 and columns of nodes 1, 7, and 13 respectively. Next we determine the new nodes from the rest that are distance 3 away from node 0. At this point we have the nodes in the distance table that are distance 2 or 1 away from node 0. By considering a node \(p\) at distance 2 away from node 0 in the table, we can easily find its new nodes \((p \pm 1)\) mod 20, \((p \pm h)\) mod 20 and correspondingly the new nodes that are distance 3 away from node 0. Thus by making use of node \(p \in \{6, 12, 18\}\) we obtain a list of nodes \{5, 11, 17\} at distance 3 away from node 0 and any message to any of them is to be forwarded through node 19. Similarly, we obtain the list of nodes that are reachable through 1, 7, and 13 and are distance 3 away from node 0. By repeating the same technique progressively, we complete the table and stop when all 20 nodes have been included in the table.

![Figure 3. Distance table D for node 0.](image)

\[
S = 0011 1110 1111 1011 1110 \text{ after formulation of row 1}
\]
\[
S = 0001 1100 0111 0001 1100 \text{ after formulation of row 2}
\]
\[
S = 0000 1000 0010 0000 1000 \text{ after formulation of row 3}
\]
\[
S = 0000 0000 0000 0000 0000 \text{ after formulation of row 4}
\]

Figure 4. Instances of bit array \(S\) during construction of table \(D\) of Figure 3.

We now present a general algorithm to determine distance table \(D\) for node 0 in \(G(n, \pm 1, \pm h)\). The distance
table $D$ has 4 columns as there are four adjacent nodes to node 0 and the number of rows is equal to the diameter of $G$. We notice that we do not need to compute the diameter of $G$ to form $D$. Each cell $D[i,j]$ represents a set that contains node numbers as elements that are distance $i$ away from node 0. Let $S[0:n−1]$ be a bit vector having $n$ bits. Let $S[j,i] = 0$ indicate that node $j$ has been included in table $D$ and $S[j,i] = 1$ indicate that it has not been included in table $D$. The bits of $S$ are updated as $D$ is formed. Let $S[0:0] = 0$. Let $D[1,1] = n−1;$ $D[1,2] = 1;$ $D[1,3] = h$ and $D[1,4] = n−h$. Consequently, $S[n−1:n−1] = S[1:1] = S[h:h] = S[n−h:n−h] = 0$. Other rows of $D$ are successively formed based on the following rule: A node $m$ is entered in row $p+1$ and column $q$ of $D$ if node $m$ is at distance 1 from some node in $D[p,q]$ and $S[m:m] = 1$. Next for each node entry $m$ in $D[p+1,1]$, $D[p+1,2]$, $D[p+1,3]$, and $D[p+1,4]$, indicator bit $S[m:m]$ is set to 0. The distance table $D$ and the associated indicator arrays are shown in Figures 3 and 4. It may be seen that the minimum distance between node $k$ and a node listed in row $p$ is $p$ and accordingly the table is labeled with distance. The algorithm terminates when all bits in $S$ are 0 i.e., the numeric value of $S = 0$. The algorithm is summarized in Figure 5.

**Step 1.** Initialize $S[0:n−1]$ to 1 and assign 0 to bit $S[0:0]$.

**Step 2.** Include nodes $n−1, 1, h,$ and $n−h$ in cells $D[1,1], D[1,2], D[1,3],$ and $D[1,4]$ respectively and assign 0 to bits $S[n−1:n−1], S[1:1], S[h:h],$ and $S[n−h:n−h]$. Let $p = 1$.

**Step 3.** For every node $m$ in cell $D[p,1]$, find its adjacent nodes $(m ± 1) \ mod \ n$ and $(m ± h) \ mod \ n$ and include in cell $D[p+1,1]$ only adjacent nodes for which the corresponding bit in $S$ is 1. Repeat this for nodes in cells $D[p,2], D[p,3],$ and $D[p,4]$ and correspondingly fill in cells $D[p+1,2], D[p+1,3],$ and $D[p+1,4]$.

**Step 4.** Set $p = p + 1$.

- For every node $m$ in cell $D[p,1], D[p,2], D[p,3],$ and $D[p,4]$ set $S[m:m] = 0$.

**Step 5.** Stop if $S = 0$ i.e. all bits in $S$ are 0, otherwise go back to **step 3**.

Figure 5. Algorithm to obtain distance table $D$ for node 0.

It is evident from Figures 3 and 5 that each round of computation to obtain $D$ completes one row of the distance table with nodes in each cell that are one unit distance away (one link away) from the nodes listed in the cell above in the previous row. When the algorithm terminates, the final value of $p$ indicates the last row in the table and $d = p$ is the diameter of the network. We can prove by induction that the above algorithm correctly finds the minimum distance for all nodes. The induction hypothesis is the fact that the minimum distance from nodes $n−1, 1, h,$ and $n−h$ to node 0 is 1. This is true as the nodes are adjacent to node 0. Let us assume that the minimum distance to any node $k$ to node 0 is $p$. Then the minimum distance from any node adjacent to $k$ can be at most $p+1$ from node 0. Let us assume that node $m$ is adjacent to node $k$ but its distance to node 0 is already known because of previous steps in the algorithm. In that case, node $m$ can be found in a cell for distance $p−1$ instead of distance $p+1$. Hence the algorithm correctly finds the minimum distance to node 0 for each node in the network.

### 4 ROUTING ALGORITHM

Once the distance table for a node is known, the computation of the routing table for the node is straightforward. Here, we describe how to obtain the routing table $T$ for node 0. Since nodes $n−1, 1, h,$ and $n−h$ are adjacent to node 0, for any destination node other than node 0 for a message, the next node (the message to be forwarded to) will be either $n−1, 1, h,$ or $n−h$. Specifically, messages to all nodes listed in the cells of the column with node $(n−1)$ in the distance table are to be forwarded through node $(n−1)$ from node 0. In the same way, messages to all nodes in the columns of nodes $1, h,$ and $n−h$ respectively in the distance table are to be forwarded respectively through nodes $1, h,$ and $n−h$. That is, given a node $m$, we need to find out whether node $m$ occurs in the column of nodes $n−1, 1, h,$ or $n−h$ in the distance table and accordingly include the nodes from the set of nodes $(n−1, 1, h, n−h)$ in the routing table as the next nodes for destination node $m$. Based on the distance table $D$ in Figure 3, we provide the routing table $T$ in Figure 6 for node 0 in $G(20, ±1, ±7)$.

<table>
<thead>
<tr>
<th>Destination Node</th>
<th>Next Node</th>
<th>Destination Node</th>
<th>Next Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>10</td>
<td>19,1,7,13</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>11</td>
<td>19,13</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>19,13</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>19,1,7,13</td>
<td>14</td>
<td>1,7,13</td>
</tr>
<tr>
<td>5</td>
<td>19,7,13</td>
<td>15</td>
<td>1,7,13</td>
</tr>
<tr>
<td>6</td>
<td>19,7,13</td>
<td>16</td>
<td>19,1,7,13</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>1,7</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>1,7</td>
<td>19</td>
<td>19</td>
</tr>
</tbody>
</table>

Figure 6. Routing table $T$ for node 0.

For more formal description of the algorithm, let us assume that $T[0..n−1]$ is a collection of sets $T[0], T[1], \ldots$
.. $T[n-1]$ and $d$ is the diameter of the network $G(n, \pm 1, \pm h)$. Then the algorithm can be given as:

**Step 1.** Compute the distance table $D$. Set $j = 0$.

**Step 2.** For each $i$ such that $1 \leq i \leq d$ and for each $j$ in cell $D[i,1]$; Include the node listed in $D[1,1]$ in $T[j]$,
in cell $D[i,2]$; Include the node listed in $D[1,2]$ in $T[j]$,
in cell $D[i,3]$; Include the node listed in $D[1,3]$ in $T[j]$, and
in cell $D[i,4]$; Include the node listed in $D[1,4]$ in $T[j]$.

**Step 3.** Repeat step 2 for $j = 1, 2, \ldots, n−1$.

It is possible to compute the routing tables for all other nodes from the routing table for node 0. The routing table for node $m$ may be obtained from the routing table for node 0 by replacing each node $q$ in $T[0..n−1]$ by node $(q + m) \mod n$ and the corresponding destination node $k$ has to be changed as: $(k + m) \mod n$ in the routing table. It is also possible to obtain the distance table for all other nodes from the distance table for node 0 using the same transformation technique.

Computing the distance table is the major part in our proposed routing algorithm. As shown in Figure 5, step 3 in the algorithm is the major step that finds all nodes at a particular distance from node 0. The step is repeated $d$ times, where $d$ is the diameter of the network (the largest distance from node 0 to some node on the network). In this sense the complexity of the algorithm is $O(d)$. In this case we are assuming that step 3 is going to take a constant amount of time to execute. However, if we consider all cells in the distance table $D$, we see that there are $4d$ cells in the table and each cell can have multiple node entries. Then the time complexity of the algorithm depends on the node entries in the entire distance table and hence can be bounded by $O(dM)$, where $M$ is the largest possible number of nodes in a cell of the distance table $D$.

5 CONCLUSION

In this work, an algorithm for the formulation of message routing tables in a distributed double loop network is presented. It takes $O(d)$ time steps (where $d$ is the diameter of the network) for the algorithm to formulate the routing table for a specific node. Given a source node, the algorithm finds the shortest paths (distances) to all other nodes on the network. From the routing table and the distance table of a given node, the routing table and the distance table for other nodes for message routing can be easily obtained through simple transformation. Similar technique can be applied to obtain the routing table and the distance table for any other type of distributed loop networks.

6 REFERENCES


