A Hybrid Multicast Routing Approach with Enhanced Methods for Mesh-Based Networks-on-Chip

Chun-Wei Wu, *Member, IEEE*, Kuen-Jong Lee, *Fellow, IEEE* and Alan P. Su, *Member, IEEE*

Abstract—Multicast communication can greatly enhance the performance of Networks-on-Chip. Currently most multicast routing algorithms are either tree-based or path-based. The former has low latency but needs to solve multicast deadlocks through additional hardware resources. The latter can avoid deadlocks easily but may require long routing paths. In this paper we propose a hybrid multicast routing approach that combines the advantages of both path- and tree-based methods. The proposed approach ensures deadlock-free multicast routing without requiring additional virtual channels or large buffers to hold large packets. High routing performance is achieved using an adaptive routing strategy considering the traffic load in nearby routers. Two techniques, namely node balancing and path balancing, are further developed to enhance this hybrid routing algorithm. Extensive experiments with different buffer sizes, packet sizes and numbers of destinations per packet under random and Rent's rule traffic at various traffic injection rates have been conducted. The results show that the average latency of our approach is lower than previous multicast routing algorithms in most cases, and the saturation points of our approach are always at much higher injection rates.

Index Terms—Networks-on-chip, multicast, deadlock-free, routing algorithms, adaptive routing, Rent's rule traffic.

1 Introduction

SYSTEMS on-chip (SoC) with multi-processors, or MPSoCs [1], are widely used in multimedia, internet, telecommunication, and artificial intelligence applications. One of the critical issues in MPSoCs is the scalability of processor cores. Bus-based systems lack scalability because both wire routing complexity and arbitration delay of buses increase significantly with the number of cores [2].

Networks-on-Chip (NoC) have been proposed to solve the problems of traditional buses. In a NoC-based MPSoC, the communication is carried out through a network of modular components such that high scalability can be easily achieved. Fig. 1(a) shows a typical mesh-based network that is composed of three basic functional units, namely routers, links, and network interfaces. Routers dispatch packets in the network. Network interfaces (NIs) connect processor elements (PEs) to the network, and links connect routers to other routers. Fig. 1(b) shows a typical router that has 5 input buffers (channels) to accept packets from its 4 neighbor routers (west, east, south, and north) and its local PE.

The communication in NoC-based systems can be unicast or multicast depending on the number of destinations that a packet needs to be routed to. Multicast communication has been commonly used in multiprocessor systems to enhance the performance in applications such as barrier synchronization [3], [4], cache coherency [5], and clock synchronization [6], [7]. In this paper, we focus on multicast communication.

- C.-W. Wu is with MediaTek, Taiwan. E-mail: jeffwu0429@gmail.com.
- K.-J. Lee is with the Department of Electrical Engineering, National Cheng Kung University, Taiwan. E-mail: kjlee@mail.ncku.edu.tw.
- A. P. Su is with the Department of Electrical Engineering, National Cheng Kung University, Taiwan. E-mail: alansuphd@mail.ncku.edu.tw.

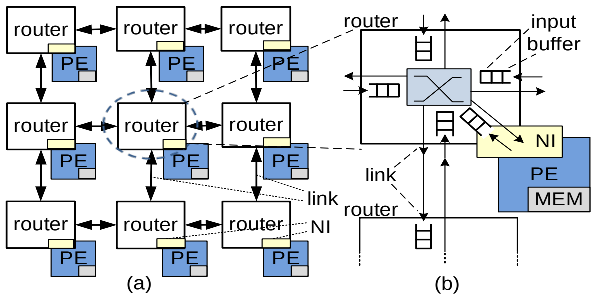


Fig. 1. A 3×3 mesh-based NoC.

Two main challenges exist for multicast communication. One is how to deliver multi-destination packets efficiently, and the other is how to resolve the deadlock problem that occurs when two or more packets form a circular wait-and-hold loop. The deadlock problem in multicast systems is generally more difficult than that in unicast systems, as the latter can be easily avoided by prohibiting some routing turns [8], [9]. In this paper we propose an efficient deadlock-free approach to prevent multicast deadlocks.

The proposed approach includes a multicast routing procedure and a router design supporting multicast communication. The key component of the multicast routing procedure is a routing algorithm that adopts a hybrid strategy to combine the advantages of two most commonly used routing methods, namely the tree-based [10], [11] and the path-based [4], [7] methods. To leverage the pros and eliminate the cons of both methods, in this work we propose a novel adaptive strategy that mainly follows an efficient path-based routing scheme but also allows branches to be taken when some specific conditions are satisfied. These conditions take the load of the routers and the deadlock problem into account and hence can lead to more efficient overall routing performance without causing any deadlock. Moreover, the proposed routing approach deals with the multicast deadlock problem without using extra routing channels or large buffers and hence has low area overhead and low power dissipation features.

Some preliminary work of this paper on the hybrid routing algorithm (HRA) has been presented in [12]. In this paper, in addition to describing the algorithm in much more detail, we have extended this work by

1. enhancing the router architecture and the routing procedure that supports HRA, including a novel stall mechanism,
2. developing two advanced techniques, called node balancing and path balancing, to further improve the performance of HRA,
3. formally proving that the proposed routing approach is deadlock-free, and
4. carrying out extensive experiments considering the sizes of packets, the numbers of destinations, the sizes of buffers in routers and the packet injection rates, all with a random traffic model and a more realistic Rent's rule traffic model [13]. Experimental results show that our approach outperforms previous methods in most cases and always has saturation points at higher injection rates.

For completeness of this paper, some contents of [12] are included in this paper. The rest of this paper is organized as follows. Section 2 describes the background and prior arts regarding multicast routing. The proposed routing approach is presented in Section 3, which includes the router design, the multicast routing procedure, and the hybrid multicast routing algorithm. Section 4 describes two advanced methods to enhance the performance of our routing algorithm. The proof of the proposed routing approach being deadlock-free is given in Section 5. Section 6 presents the experimental results and the comparison with prior arts. Finally, Section 7 concludes this work.

2 Background & Previous Work

In this section we give necessary background information and survey prior arts. We will describe the wormhole switching technique [4] in NoCs, the deadlock problem of multicast routing, and two most commonly used multicast routing methods, i.e., tree- and path-based methods.

2.1 Wormhole Switching and Packet Format

We adopt the widely used wormhole switching technique [4] in our approach. This technique divides a packet into a number of small pieces called flits (FLow control digITs). As shown in Fig. 2, a flit consists of two parts, the flit type and the flit content. The flit type indicates whether the flit is a head, data, or tail flit.

There are mainly two packet formats for multicast destination encoding, namely the one-flit-per-destination format [7], [10], [12], [14], [15], [16], [17], and the bit-string encoding format [18], [19], [20], [21], [22], [23], [24], [25], [26]. The packet size of the bit-string encoding format is in general smaller and thus we choose it in this work. In our bit-string format, as shown in Fig. 2, the contents of a head flit include the destination information of the package

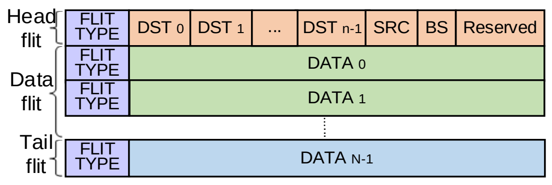


Fig. 2. Bit-string multicast packet format.

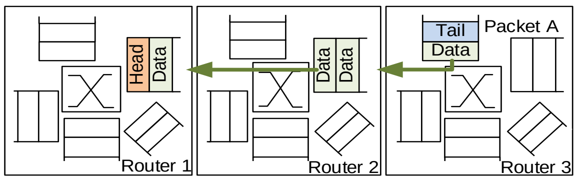


Fig. 3. Wormhole switching.

(DST_0 to DST_{n-1}), the location of the source node (SRC), and the *Branch State* of the packet (BS, to be described in Section 3.2). The destination nodes are recorded in fields DST_0 to DST_{n-1} with DST_i = 1 (0) implying that node i is (is not) in the destination set, where n is the number of nodes in the network. The contents of all data flits and the tail flit altogether are the data to be delivered. The tail flit further signals the end of the packet.

Fig. 3 illustrates how a packet moves using the wormhole switching technique. In this technique each routing buffer consists of a number of storage elements and all flits have the same size (width) as that of a storage element. When the routing direction of a packet is determined and its routing request is approved, the head flit of the packet is forwarded to the router in the determined direction and the data flits and the tail flit follow the head flit in a pipeline fashion. Thus, flits will hop from storage element to storage element like a worm.

2.2 Deadlock Problem in Multicast Routing

One problem in NoCs supporting multicasts is that the contention among packets for routers may lead to a deadlock [9], [16], [27]. Fig. 4 shows an example of a multicast deadlock. A dashed line represents a port-request not yet granted, and a solid line represents a granted and assigned port-request. Packet A is requesting the southern port of Router 2, which is being used (held) by Packet B, while Packet B is requesting the southern port of Router 3, which is being held by Packet A. Thus, each packet holds one output port and waits for another port held by the other packet. In most routing approaches that allow branches, the two packets cannot move forward in such a case, as to be explained in Section 2.3.1. This will result in a multicast deadlock. This kind of deadlocks is due to the branching of packets and will not occur in unicast routing where no branch is needed. Previous solutions to avoid such kind of deadlocks include using large buffers to hold entire packets, increasing the number of virtual channels, each of which will be multiplexed with existing channel(s) on a single physical link, and using path-based methods without branching [2], [4], [27], [28].

2.3 Multicast Routing Methods

Two general multicast routing methods have been proposed, namely tree-based [10], [11], [18], [19], [20], [21]

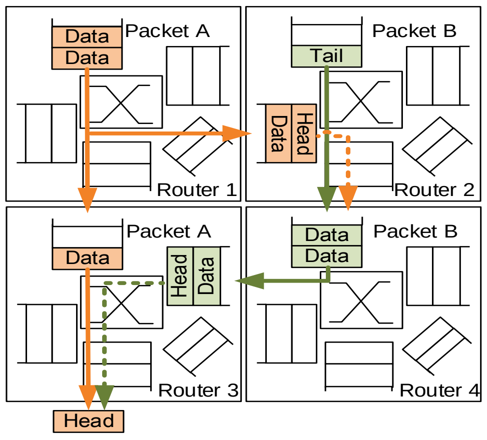


Fig. 4. A multicast deadlock.

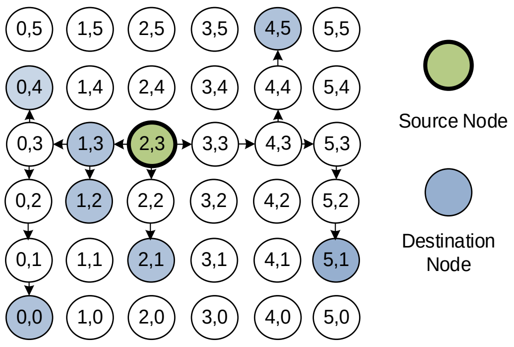


Fig. 5. An example of the tree-based routing.

and path-based methods [4], [7], [14], [17], [22], [23], as described next.

2.3.1 Tree-Based Multicast Routing Method

In the tree-based multicast routing method, the routes that a packet traverses through form a tree, with the source node being the root and all destination nodes being the leaves. Each path from the root to a destination node is a shortest path between the two nodes [10]. Branch operations occur during packet traversal attempting to reduce the latency of routing a multicast packet. An example of the tree-based routing method is shown in Fig. 5, where node (2, 3) is the source node and all nodes shaded in blue are destination nodes. The nodes traversed and the indicated directed edges form a tree.

There are two ways to transmit a packet to be branched. One is to duplicate the packet into two packets, each with its own head, data and tail flits. Both packets are then sent to their own requested direction independently. This method requires more buffer space to hold the two packets simultaneously. In the other one only the flits that pass the "branch point" are duplicated. Fig. 6 gives an example of the latter case where a branch occurs at node (4, 3). In Fig. 6(a) the shown packet requests the northern and the eastern output ports. After both requests are granted, both output ports are assigned to the packet by the router. The head flit of the packet is then used to generate two head flits with different destination sets, which are then sent to

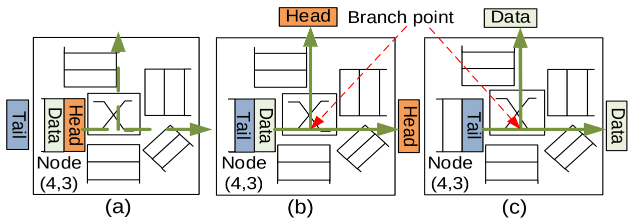


Fig. 6. Branching of the tree-based multicast routing method.

two different output ports as shown in Fig. 6(b). The following data and tail flits are duplicated when they pass through the branch point as shown in Fig. 6(b)(c). Most multicast routing methods, including ours, adopt the latter method because it requires less buffer space. However, this implies that when a branch occurs, the flits can move forward only when both directions allow the movement simultaneously. If either direction is blocked, the other one must wait, which may cause a deadlock as shown in Fig. 4.

In general, in a router design packets created by branch operations must compete with other existing packets for the available routes, which may increase contention in links and router ports, leading to performance degradation or even a deadlock. All tree-based methods need to handle these two problems. Most of them try to enhance the performance by minimizing the link usage, i.e., the total number of edges to be traversed, while solving the deadlock problem simultaneously. The Hardware Tree-Based Multicast Routing Algorithm (HTA) [10] implements a deadlock queue in each router to prevent deadlocks. The work in [11] proposes a router design that avoids deadlocks by using multiple pointers for duplicated packets. The Recursive Partitioning Multicast method (RPM) [18] and the Improved Minimal Multicast Routing method (IMM) [19] use two virtual channels to transmit up and down streams separately to avoid deadlocks. The IMM further builds a tree with a fully used shared path to obtain a minimal multicast tree. The Aware Dual-Partition Multicast Routing method (LADPM) [20] and the Adaptive Partition-Based Multicast Routing method (APBM) [21] use two virtual channels and two turn models, North-Last and West-Last, to avoid deadlocks [4]. APBM not only reduces the link usage but also adaptively chooses routing paths based on the available buffer sizes. Experimental results show that it outperforms all other previous tree-based methods [21] .

All the above methods either employ additional virtual channels or require buffers large enough to hold the largest packet(s) in order to solve the deadlock problem. In this paper we develop a routing method that solves the deadlock problem without using any virtual channel, and the sizes of the buffers employed can be smaller than that of the largest packet.

2.3.2 Path-Based Multicast Routing Method

A path-based routing method routes packets based on the following node-labeling method. Each node (x, y) in an a by

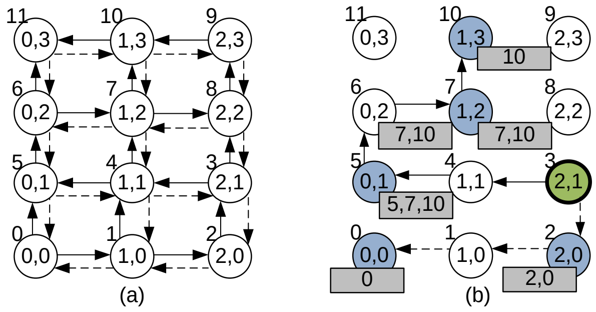


Fig. 7. A labelled 4x3 2D mesh NoC.

b 2-D mesh is assigned with a label $L(x,y)$ defined as

$$L(x,y) = \begin{cases} b \times y + x & \text{if } y \text{ is even} \\ b \times (y+1) - x - 1 & \text{if } y \text{ is odd} \end{cases} \quad (1)$$

Fig. 7(a) gives an example of a 4 by 3 mesh NoC with all routers labeled. A common strategy of path-based methods is to divide all nodes into a certain number of groups based on the labeling and send a duplicated packet to each group that contains at least one destination node.

In the dual path routing (DP) method [4], the destinations are partitioned into two groups: the low group containing destinations with labels smaller than the source node and the high group containing destinations with larger labels. The destinations of a packet are divided into two sets, those with larger labels and those with smaller labels, and sent to the higher and lower groups, respectively. With this path-based method, a packet will visit all destinations in the same row in a strictly ascending (or descending) order as the solid (or dashed) arrows show in the figure, before routing to the destinations in the next rows. One major advantage of this method is that no deadlock can occur. Fig. 7(b) illustrates the DP routing method, where the source is node 3 and the destinations are nodes 0, 2, 5, 7, and 10. There will be two routing paths, 3→2→1→0 and 3→4→5→6→7→10. Note that the path-based method may take a "short cut" when there are no destination nodes between the current node and the node in its vertical direction. Thus, the packet at node 7 will be sent to node 10 directly.

Some path-based algorithms propose different partitioning techniques to further shorten path lengths. The Multi-Path algorithm (MP), also presented in [4], proposes to partition nodes into four disjoint 'rectangular' subsets based on the location of the source node as illustrated in Fig. 8(a). Only destination nodes in the same subset are routed in the same path to avoid long paths though different subsets. Thus, the total length of paths can be reduced. However, this example also shows unbalanced partitioning that may result in long routing paths. The Column-Path algorithm (CP) [7] can partition nodes into up to 2×k subsets, where k is the number of columns in the mesh. With a high degree of parallelism, this scheme generally has more balanced partitions and shorter paths than the MP method. The Recursive Partitioning (RP) [22]

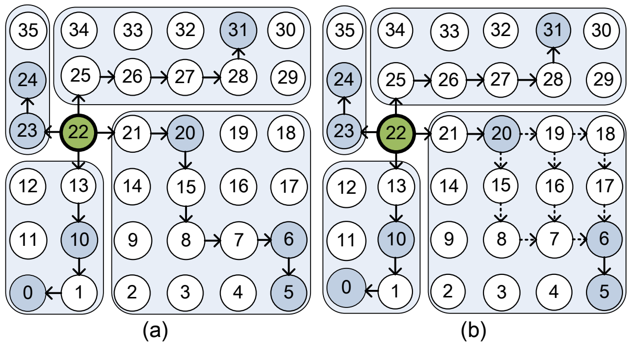


Fig. 8. Examples of (a) multi-path method (MP) and (b) Adaptive MP method (AMP.)

optimizes the number of nodes to be included in a partition for 3D NoCs by recursively partitioning the nodes until each partition contains a comparable number of nodes. The RP can also be applied to 2D NoCs. However, the RP only considers balanced partitions but does not consider the locality of traffic loads that may result in congestion in some clusters of nodes.

In [7], the authors also propose an adaptive routing model called Hamiltonian Adaptive Multicast Unicast Model (HAMUM) that allows more routing paths to be chosen so as to improve the MP and CP algorithms. The new algorithms are named Adaptive MP (AMP) and Adaptive CP (ACP), respectively. For example, Fig. 8(b) shows that with AMP the routing path can go south at node 20, 19 or 18. Thus, this mechanism increases the probability of a packet to choose a non-congested path and hence can further shorten the routing time. In [23], a Hamiltonian-based Odd-Even (HOE) turn model to maximize the degree of adaptiveness while remaining deadlock-free without adding virtual channels is proposed. The turn rules of the model are applied to MP and CP and the resulting algorithms are named HOEMP and HOECP, respectively. However, when the source node is located near the boundary, e.g., node 22 in Fig. 8, MP, AMP and HOEMP all suffer from unbalanced partitioning, which results in long routing paths. On the other hand, CP, ACP and HOECP require preparing and dispatching a large number of packets when the number of destinations grows, which may cause severe congestion at the source node.

3 Hybrid Multicast Routing Approach

The main idea of the proposed hybrid multicast routing approach is to combine the advantages of tree-based and path-based methods such that the routing can vary adaptively according to the traffic of the network. The multicast packet delivery basically follows the path-based method with the direction directed to the first destination node. We call this direction the *leading direction*. However, under the circumstances that a branch does not and will not cause deadlocks, a branch may be taken. We call the routing direction of the branch the *branch direction*.

Our proposed routing approach includes a hybrid

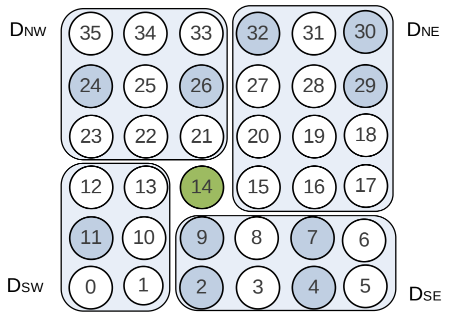


Fig. 9. An example of Multi-path partitioning (MP).

multicast routing procedure, a router design that supports the procedure, and an efficient algorithm called the Hybrid Routing Algorithm.

3.1 Hybrid Multicast Routing Procedure & Router Architecture

Our multicast routing procedure contains a Destination Partition step that is executed in the network interface (NI) to prepare multicast packets, and another three pipelined steps that are executed in each router to transmit packets to appropriate destinations.

The Destination Partition step proceeds as follows. When a PE requests a multicast transaction, the network interface partitions the destinations into four subsets, namely D_{NW}, D_{NE}, D_{SW}, and D_{SE}, using the Multi-path partition method (MP) proposed in [4]. The source node then constructs a packet for each non-empty subset. Fig. 9 shows an example of the partition, where node 14 is the source node. All nodes with labels larger than 14 are partitioned into subsets D_{NE} and D_{NW}, with those nodes having larger x-coordinates than node 14 in D_{NE} and the others in D_{NW}. All nodes with labels smaller than 14 are partitioned into subsets D_{SW} and D_{SE} similarly.

After all destination nodes are assigned to their subsets, the network interface prepares a packet for each non-empty subset with its head flit storing the destinations in fields DST_0 to DST_{n-1} using bit-string encoding (see Fig. 2). For example, in the packet for D_{NE}, the values of the fields DST_{29}, DST_{30} and DST_{32} are set to 1 to represent destination nodes 29, 30 and 32.

The packets prepared by the NI are then sent one by one to the router connected to the NI through the local input buffer. Each packet entering the local input buffer will request the output port in its direction. This request together with requests from other packets are then arbitrated in the router whose architecture is shown in Fig. 10. *In our router architecture, the requests are arbitrated by an arbiter in a round-robin manner, and at any time the arbiter will either grant both requests or decline both requests from a packet requesting a branch.* This policy is an important factor to prevent deadlocks in our method as described later.

Each router has four major units: a set of *input channels*, a set of *routing computation units*, a *switch allocator*, and a *crossbar switch*, all described below together with the stages (steps) each packet will go through in a router.

As shown in Fig. 11, each packet goes through a three-stage pipeline in a router: buffer writing (BW), routing computation & switch allocation (RC/SA), and switch

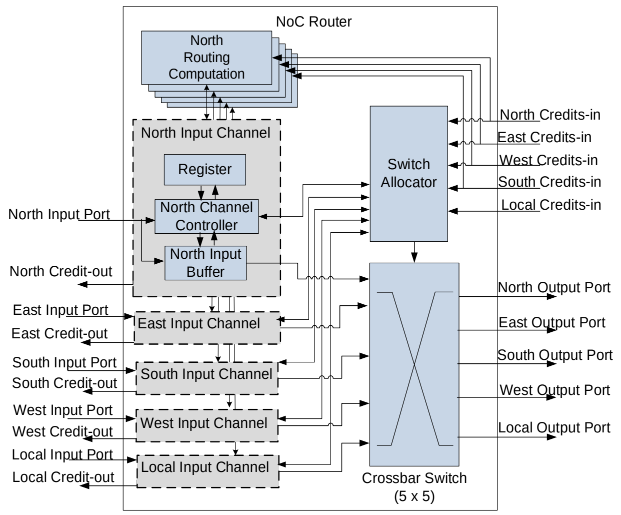


Fig. 10. The router architecture.

Head flit	BW	RC/SA	ST		
Data flit		BW	SA	ST	
Tail flit			BW	SA	ST

Fig. 11. The pipeline stages for packet switching in a router.

traversal (ST).

At the BW stage, a packet is written to the input buffer of an input channel when it is determined that the input channel is available to it. Note that the local channel is treated the same as the other four channels. Fig. 10 shows the details of the *North input channel*. It has an *input buffer* to store packet(s) that are using the channel. It also has a *channel controller* to control the operation of the channel and keep track of the unused buffer size. This information, called the *credit-out* signal, is provided to adjacent routers. The *register* is used to record the routing directions and the destination nodes of each direction. This information is used to guide the transmission of the data and tail flits to their proper output port(s). The other input channels, including the local channel, have similar components and functions.

At the RC/SA stage, the *routing computation* unit executes the proposed hybrid routing algorithm (HRA) and evaluates the congestion situation of adjacent routers through the *credit-in* signals, i.e., the *credit-out* signals from adjacent routers, to determine the forwarding directions, i.e., *leading direction* and/or *branch direction*, for a packet. If a packet is from NI, then it only has the *leading direction*. The routing computation together with the HRA will be detailed in Section 3.2. The *channel controller* then sends the forwarding request(s) of a head flit to the *switch allocator*. The *switch allocator* receives the requests from all input channels and arbitrates the connection between input channels and output ports in a round-robin policy. If a head flit is not granted, it will repeat the RC/SA stage in the next cycle. Data and tail flits do not require *routing computation*; they always follow the head flit. As mentioned before, if a branch is also requested for a packet, then either both leading and branch direction requests are granted, or both are declined for the packet.

A packet will enter the ST stage when its request(s) is granted. Each granted packet is transmitted to its required output port(s) through the *crossbar switch*. In our architecture, an input channel can be connected to one or two output ports simultaneously. As such, in a branch operation a packet is transmitted through the *crossbar switch* to corresponding output ports of both leading and branch directions. Notice that in our router architecture only input channels have buffers to store packets; no buffers are needed in output ports. Also, our proposed router uses only one input buffer for each direction without adding additional virtual channels. Next, we describe the *routing computation unit* in detail.

3.2 Routing Computation

The *routing computation unit* executes the Hybrid Routing Algorithm (HRA) to determine the *leading direction* and the *branch direction* of a packet. Both directions follow the following *label routing rule*: *A flit will always be routed to a node with a smaller (larger) label if the label of its destination node is smaller (larger) than the label of the current node; else (the labels are equal), it is routed to the local PE* [7]. It has been proven in [7] that path-based routing methods that always follow the *label routing rule* avoid deadlocks since the rule prevents *circular wait*, one of the necessary conditions of deadlocks. However, multicast deadlock may still occur when branch operations are allowed during routing, as illustrated in Fig. 4. Next, we describe the HRA.

3.2.1. Hybrid Routing Algorithm (HRA)

The Hybrid Routing Algorithm (HRA), shown in Algorithm 1, determines the *leading direction* and the *branch direction* of a packet as well as the direction each destination node should be assigned to in four steps.

Step 1 sorts the destination nodes according to which subset (partition) the first destination node belongs to, so as to facilitate the *label routing rule* (lines 2-6).

Step 2 examines whether the first destination node is the local PE or not. If it is, a copy of the packet is sent to the local output port and the first destination node is removed from the destination list (lines 7-10).

Step 3 determines the *leading direction* for our path-based method based on the position of the first destination node and the traffic load in the NoC. Our path-based method combines the *label routing rule* and Adaptive MP (APM [7]), and further considers the traffic load on the routing path. First, if the current node is the source node, then the packet(s) is from the NI, and the direction(s) is determined based on which destination node set(s) it has (lines 12 and 13). Otherwise, the leading direction should be the y-direction if

1. the current node is a boundary node and can only go to the y-direction according to the label routing rule (lines 15-16), or
2. the y-downstream output port is available, and the labels of the first destination node and the y-downstream node are both larger (smaller) than the label of the current node or the current node and the first destination node have the same x-coordinate (lines 17-18)

Algorithm 1. *Hybrid Routing Algorithm*

```
Input: Destination Set: $D = \{d_i | 1 \le i \le N\}$ where $d_i$ is a destination node with coordinate ($d_{ix}$, $d_{iy}$), $N$ is size of $D$
       Source node s with coordinate ($s_x$, $s_y$)
       Current node c with coordinate ($c_x$, $c_y$)
       Credit-in of each direction.
Output: Leading Direction $dir_L$ and Branch Direction $dir_B$.
        Leading Direction Set $D_L$ ,
        Branch Direction Set $D_B$.
1: $D_L = \emptyset$; $D_B = \emptyset$;
2: if ($d_1$ is in $D_{NW}$ or $D_{NE}$) then /*Step 1 Sort D */
3:   Sort D in ascending order
4: else if ($d_1$ is in $D_{SW}$ or $D_{SE}$) then
5:   Sort D in descending order
6: endif
7: if($L(c_x, c_y)$ is equal to the label of $d_1$ in D) then /*Step 2 Arrive at a destination */
8:   Send one packet to the local output port
9:   Remove $d_1$ from $D$ and update the indices of remaining $d_i$
10: endif
11: Add $d_1$ to $D_L$ /*Step 3 Path decision for $d_1$ */
12: if ($L(c_x, c_y) == L(s_x, s_y)$) then // $L(x,y)$ is label of node (x,y)
13:   Determine $dir_L$ according to the $D_{NW}$, $D_{NE}$, $D_{SW}$, and $D_{SE}$
14: else if ($d_1$ is in $D_{NW}$) then
15:   if ($c_x$ is on boundary of $D_{NW}$ and x-direction is invalid) then
16:     $dir_L$ is north
17:   else if ( (the northern output port is available) and
              ($L(d_{1x}, d_{1y}) > L(c_x, c_{y+1})$ or ($c_x$==$d_{1x}$)) ) then
18:     $dir_L$ is north
19:   else if ($L(c_{x-1}, c_y) > L(c_x, c_y)$) then $dir_L$ is west
20:                                  else $dir_L$ is east
21: else if ($d_1$ is in $D_{NE}$)
22:   -- similar to $d_1$ in $D_{NW}$
23: else if ($d_1$ is in $D_{SW}$)
24:   -- similar to $d_1$ in $D_{NW}$
25: else if ($d_1$ is in $D_{SE}$)
26:   -- similar to $d_1$ in $D_{NW}$
27: endif
28: Remove $d_1$ from $D$
29: if ($dir_L$ is north or south) then add all nodes in $D$ to $D_L$
30: else for each $d$ in $D$ // Step 4 determines Tree-based nodes
30:   if ($d$ is in $D_{NE}$ or $D_{NW}$) then
31:     if ( (Branch Condition I is true) or
            (Branch Condition II is true) ) then
32:       $dir_B$ is north and add $d$ to $D_B$
33:     else add $d$ to $D_L$
34:   else if ($d$ is in $D_{SE}$ or $D_{SW}$)
35:     -- similar to $d$ in $D_{NE}$ or $D_{NW}$
36: endfor
```

If the above conditions are not satisfied, the routing direction will be east or west according to the label routing rule (lines 19-20).

Compared to AMP that may keep waiting if the y-direction output port is not available, our method does not wait and allows to take a longer but available path for a packet.

Step 4 of Algorithm 1 is for our tree-based method. It determines which nodes should be in the *leading direction set* (D_L) and which should be in the *branch direction set* (D_B). Clearly a branch operation can occur only when the leading direction is in the x-direction (line 29). Algorithm 1 examines the following two branch conditions for each destination node (except for the first destination) when the *leading direction* is in the x-direction. If either condition is met, a branch operation is taken and the *branch direction* is

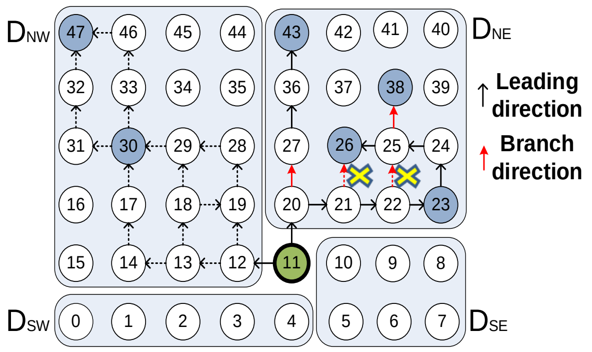


Fig. 12. An example of the hybrid multicast routing.

always in the y-direction (lines 30-32).

Branch Condition I:

1. Current router's y-direction output port is available,
2. current node has the same x-coordinate as the destination node, and
3. the unused buffer size of the y-downstream router is large enough to hold the entire packet.

Branch Condition II:

1. Current router's y-direction output port is available,
2. current node has the same x-coordinate as the destination node,
3. the destination node is one hop away from the current node, and
4. the buffer of the y-downstream router is empty.

The Branch Condition I requires that the input channel of the y-downstream router, i.e., the y-direction router that the packet is to be sent to, has enough unused buffer space to accommodate the entire packet. This requirement combined with the *stall process* described in Section 3.2.2 ensures that multicast deadlocks cannot occur due to this type of branches. The Branch Condition II is for a packet that will immediately reach a destination node in the y-direction after branching. We prove that both branch types do not lead to deadlocks in Section 5. Next we use an example to illustrate the execution of HRA.

Example: An example of the HRA execution is shown in Fig. 12 where one packet is sent from source node 11 to 6 destination nodes: two (nodes 30 and 47) in D_{NW}, and 4 (nodes 23, 26, 38, and 43) in D_{NE}. For D_{NW}, the routing path using HRA is path-based similar to Adaptive MP (AMP) [7]. When the current node is Node 31, the packet can only go north since Node 31 is a boundary node satisfying the first condition of Step 3. When the current node is Node 12 or Node 13, since both the first destination node and the y-direction downstream node have larger labels than the current node, it will go north if the y-downstream output port is available (Condition 2 of Step 3). Otherwise it goes west. If the packet arrives at Node 18, HRA may be different from AMP because AMP can only route the packet to node 29 and hecne has to wait if node 29 is not available, while with HRA the packet can be routed to a longer path through node 19 without waiting.

For the D_{NE} subset two branches are taken, one meeting Branch Condition I and the other one meeting Branch Condition II of Step 4. When the packet arrives at node 20, assuming the northern output port of node 20 is available and the unused buffer size in the southern input channel of node 27 is large enough, Branch Condition I is met for the destination node 43. Thus, HRA generates a *branch direction set* (D_B) containing destination node 43. The router then sends the packet to both nodes 21 and 27 if the requests are granted. Next assume the north output ports of both nodes 21 and 22 are not available for the packet in the *leading direction*. The packet will keep moving toward the leading direction through nodes 21, 22, 23, 24 and arrive at node 25. Now assume the northern output port of node 25 is available and the southern input buffer of node 38 is empty. Then the packet meets Branch Condition II. The router will make a branch and send the packet to both nodes 38 and 26. The packet thus can be delivered to all destinations in D_{NE} with a latency of 7, where the latency is defined as the number of edges on the longest path from the source to any destination node. Compared to a path-based method that has a path length of 12 (11→20→21→22 →23→24→25→26→37→38→41→42→43), and to the tree-based methods that need to wait for the northern output ports of nodes 21 and 22, the HRA is shorter in path length than AMP and has a higher chance to shorten the latency than the tree-based methods. □

It should be pointed out that HRA only determines the leading direction and the branch direction (if any); whether the requests will be granted or not is determined by the arbiter in the switch allocator of the router. If the requests are declined, HRA will be executed for the packet again and a new set of requests will be generated for the next arbitration.

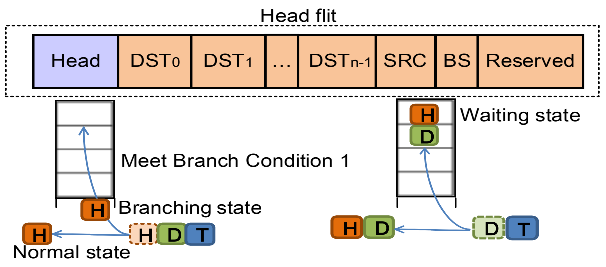


Fig. 13. The branching state and waiting state.

3.2.2. Stall Process

For Branch Condition I in our implementation we require a branched packet to enter the input buffer of its y-downstream node completely such that no packet can hold one output port while waiting for another output port. This implies that the packet transfer is always an *atomic operation* in the y-direction, i.e., the whole packet can be considered as a single entity that can be sent to the node in the y-direction completely and thus the output channel of the current node supporting the routing can always be released.

We implement a *stall process* to warrant this *atomic operation*. In this process a packet has three states, namely the normal state, the branching state, and the waiting state. These states help avoid any deadlock and are recorded in the BS entry of a head flit as shown in Fig. 13. The initial state is the normal state. When a packet meets Branch Condition I, the requests to use the output ports in both the leading and branch directions are sent to the arbiter in the switch allocator. If both requests are granted, the packet can be sent to both the x- and y-downstream routers. The head flit of the packet will be duplicated and the state of the flit in the y-direction will change from the normal to the branching state. After the head flit in the branch direction is sent to the input buffer of the neighbor router, the state of the packet changes to the waiting state and it needs to be stalled there until the tail flit enters the buffer. Only then the state is changed back to the normal state and the head flit of the packet can request to be transmitted further. During this process the head flit in the leading direction remains in the normal state.

Note that in traditional tree-based methods, the buffer size must be larger than any possible packet in order to handle the multicast deadlock problem [27], [28]. Thus, the buffer size must be large even if only very few packets have large sizes. In our method we can have any size of buffers. When the size of a packet is larger than the available buffer size and Condition I is not satisfied, we will proceed to check Condition II. Condition II only requires the destination node to have an empty buffer but not a large buffer. Thus even if the buffer size is too small to hold a large packet, a branch can still occur if the destination is only one hop away from the current node. Note that the HRA does not execute a branch operation if neither condition is satisfied. Thus, unlike traditional tree-based approaches in which the packet will wait until the buffer size becomes large enough, with HRA the packet will be routed through the path-based method without waiting. Since no packet will be waiting for a router due to insufficient buffer size, the HRA is more flexible than traditional tree-based methods.

3.3 Analysis of Overhead and Power Dissipation

The area of a router is typically dominated by its buffer size. The tree-based method needs large input buffers to hold an entire packet of any size and/or some additional virtual channels to prevent deadlocks [11], [18], [19], [20], [21]. In our proposed method, deadlocks are prevented no matter what the size of the buffers are. If the data size of a packet is larger than the available buffer size, the packet will not meet Branch Condition I. However, the packet may still meet Branch Condition II and thus the length of the path may still be shortened. If neither condition is satisfied, branching will not occur, and the routing is just like the path-based one. Therefore, our algorithm provides deadlock-free routing with both smaller area overhead and higher routing efficiency.

Duplicating more packets results in more power dissipation since more packets are to be transferred. The proposed hybrid multicast method duplicates fewer packets than the tree-based method, thus the power dissipation of our method is likely less than the tree-based method. Even though there is no branch in the path-based methods, the lengths of routing paths are usually longer than those in our method. Thus, the consumed power of the path-based methods may be large when the routing paths are long or the number of packets in the whole network is large. Overall speaking, HRA reduces link contention over both tree-based and path-based methods and thus less power dissipation can be expected.

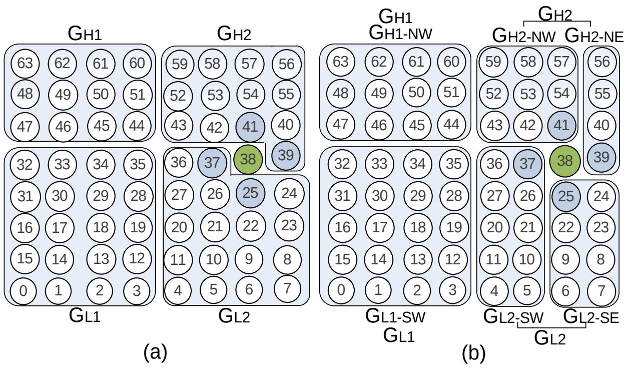


Fig. 14. Examples of (a) *4*CP and (b) *4*-CMP.

4 Enhancement Methods

The HRA combines the advantages of both tree- and path-based methods. However, we find that in some cases certain nodes exhibit much more traffic than other nodes. Also, some paths tend to be much longer than other paths. In this section, two enhancement methods to balance the load of nodes and paths, called *Node Balancing Method* and *Path Balancing Method,* are respectively presented in Section 4.1 and 4.2. The path balancing method (PBM) can be an exhaustive or a heuristic algorithm. We shall refer to the HRA with node balancing and exhaustive PBM as EPBM+NB and to the HRA with node balancing and heuristic PBM as HPBM+NB.

4.1 Node Balancing Methods

We first propose a *k*-Column-Path (*k*CP) method to partition nodes into groups. Each group contains *k* columns to balance the traffic of nodes. We then further propose the *k*-Column based Multi-Path method (*k*CMP) that integrates the MP method into *k*CP to further balance the node traffic.

4.1.1 k-Column-Path (kCP)

In order to balance nodes across partitions, the network is partitioned into a number of subsets as follows. Following the *label routing rule* in Section 3.2, the network is first divided into two subnetworks. One contains nodes with higher labels and the other contains nodes with lower labels than the source node. Each subnetwork is then partitioned into $\lceil b/k \rceil$ subsets, where k can be $\lceil b/2^n \rceil$, $n = 1, 2, \ldots, \lceil \log_2(b/2) \rceil$ in an $a \times b$ mesh network. For simplicity, hereafter we assume $b/2^n$ is an integer, i.e., $b = 2^i$ for some integer i. We call this method the *k-Column-Path method* (*k*CP) since each subset has *k* columns. A 4CP example in an 8×8 mesh is illustrated in Fig. 14(a), where the network is partitioned into 4 subsets (G_{H1}, G_{H2}, G_{L1} and G_{L2}) and each subset has 4 columns. For *k*CP we need to select the value of *k* carefully. Choosing a too small *k* creates too many subsets, induces too many packets in the system and thus leads to more congestion and higher power dissipation, while choosing a too large *k* reduces parallelism, creates long paths and thus results in higher latency. With a proper value of *k*, this scheme retains the benefit of the hybrid adaptive routing strategy, increases

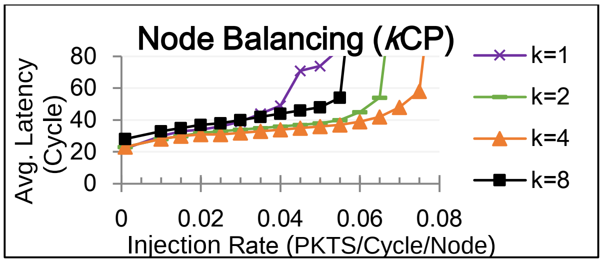


Fig. 15. Performance in 8*8 mesh with 4 destinations, 3 flits and a buffer size of 20 under random traffic.

the degree of parallelism, balances the number of nodes in each partition, and eliminates severe congestion at the source node. Through extensive simulation we find that setting $n = 1$ and $k = \lceil b/2^1 \rceil$ in an $a \times b$ mesh network tends to achieve higher routing performance. For example, Fig.15 shows the simulation results on an 8×8 mesh network with buffer size of 20 and each PE randomly injecting 4-destination, 3-flit multicast packets onto the network per second. We can see that $k = 4$ achieves the best routing performance.

*k*CP aims to eliminate long paths through balanced partitions while still holding the benefit of the hybrid adaptive strategy. However, from Fig. 14(a) we can see that to reach nodes 41 and 25, paths through nodes 39 and 37 are needed, which results in unnecessarily longer paths and increases traffic in nodes 37 and 39. This situation may occur frequently because the delivery of a packet to nearby routers is more likely to happen. This drawback is addressed by *k*CMP as described next.

4.1.2 -Column based Multi-Path (kCMP)

In the *k*-Column based Multi-Path method (*k*CMP), *k* again can be $\lceil b/2^n \rceil$, $n = 1, 2, \ldots \lceil \log_2(b/2) \rceil$ in an $a \times b$ network. However, *k*CMP combines the *k*CP and MP methods, where *kCP* balances the partitions and MP further eliminates unnecessary long paths.

Conceptually *k*CMP first uses *k*CP to partition the network and then uses MP to further partition the two subsets obtained by *k*CP that cover the column of the source node (see Fig. 8(a)). As an example, Fig. 14(b) illustrates a 4CMP partition in an 8×8 mesh. The 4CMP method first uses *k*CP to partition the network into four subsets (G_{H1}, G_{H2}, G_{L1}, and G_{L2}). It then uses MP to partition the two subsets (G_{H2} and G_{L2}) into four subsets ($G_{H2\text{-}NW}$, $G_{H2\text{-}NE}$, $G_{L2\text{-}SW}$, and $G_{L2\text{-}SE}$). Thus, the network is partitioned into six subsets.

The *k*CMP algorithm is shown in Algorithm 2. We actually implement this algorithm by integrating MP into *k*CP. Thus, all destination nodes are assigned to their subsets in one **for** loop starting at line 1. Line 3 and line 4 check whether the current destination node is in the range of the *j*-th *k*CP subset (*j* starts from 0). If it is, the node will be assigned to the (*j*+1)th G_{NW} (G_{NE}) subset if the node is at the west (east) side of the source node. For the nodes in the subset that contains the source node, the nodes may be assigned to either the G_{NW} or G_{NE} subset depending on its relative location with the source node, implying that the (*j*+1)th subset will be further divided into two subsets.

Algorithm 2. *k Column based Multi-Path Partitioning (kCMP)*

Input: N: number of destinations, $a \times b$ network,
Destination Set: $D = \{d_i | 0 \le i \le N-1\}$ where d_i is a destination node with coordinate (d_{ix}, d_{iy}),
Source node s with coordinate (s_x, s_y)
k can be $\lceil b/2^n \rceil$ ($n = 1, 2, \ldots \lceil \log_2(b/2) \rceil$)
Output: high groups (G_{H1-NW}, G_{H2-NW}, ... $G_{H\lceil b/k \rceil-NW}$, G_{H1-NE}, G_{H2-NE}, ... $G_{H\lceil b/k \rceil-NE}$)
low groups (G_{L1-SW}, G_{L2-SW}, ... $G_{L\lceil b/k \rceil-SW}$, G_{L1-SE}, G_{L2-SE}, ... $G_{L\lceil b/k \rceil-SE}$)

```
 1: for $i$ = 0 to N-1 //
 2:   if (L($d_i$) > L($s_x$, $s_y$)) then //high groups
 3:     for $j$ = 0 to ($\lceil b/k \rceil$-1) //for each subset of $k$CP
 4:       if ((($j \times k$)≤ $d_{ix}$) and ($d_{ix}$ ≤(($j$+1)×$k$-1))) then
            // $d_i$ is in the $j$-th subset of $k$CP
 5:         if ((($d_{ix}$≤$s_x$) and ($s_y$ is even)) or
               (($d_{ix}$<$s_x$) and ($s_y$ is odd))) then
 6:           $G_{H(j+1)-NW}$ <= node $d_i$; //$k$CP+MP-$G_{NW}$
 7:         else $G_{H(j+1)-NE}$ <= node $d_i$; // $k$CP+MP-$G_{NE}$
 8:         endif
 9:       endif
10:     endfor
11:   else //low groups
12:     -- similar to high groups
13: endfor
```

4.2 Path Balancing Methods

In this section, we propose two path balancing methods that attempt to balance the lengths of all paths so as to eliminate long paths and reduce the latency. The locality of traffic loads is considered and the destinations that can share routing paths efficiently are packed into the same packet. Therefore, all paths have more balanced lengths and clustered nodes can share the same path. We first propose an exhaustive path balancing method (EPBM) with a complexity of $O(2^n)$, where *n* is the number of nodes in the network. We then further propose a heuristic path balancing method (HPBM) with the time complexity of $O(\sqrt{n})$.

4.2.1 Exhaustive Path Balancing Method (EPBM)

The key idea of path balancing is to move some nodes in the D_L set to the D_B set such that the total number of hops to reach all destination nodes in the two sets is minimized and the two sets have about the same number of hops to reach all their destination nodes.

To describe this method, we first define six terms for a node (i.e., current node) that may have a branch operation. These definitions are based on the D_L and the D_B sets generated by HRA described in Section 3.2.

1. *Leading path* P_l: From the current node, the *leading path* P_l is a path using the path-based routing scheme to traverse all destinations in D_L. Its length, or the number of hops on P_l, is denoted as $|P_l|$.
2. *Branch path* P_b: From the current node, the *branch path* P_b is a path using the path-based routing scheme to traverse all destinations in D_B. Its length is denoted as $|P_b|$.
3. H_{sum}: the sum of $|P_l|$ and $|P_b|$.
4. H_{max}: the maximum of $|P_l|$ and $|P_b|$.
5. N_{FB}: the first destination node in D_B.

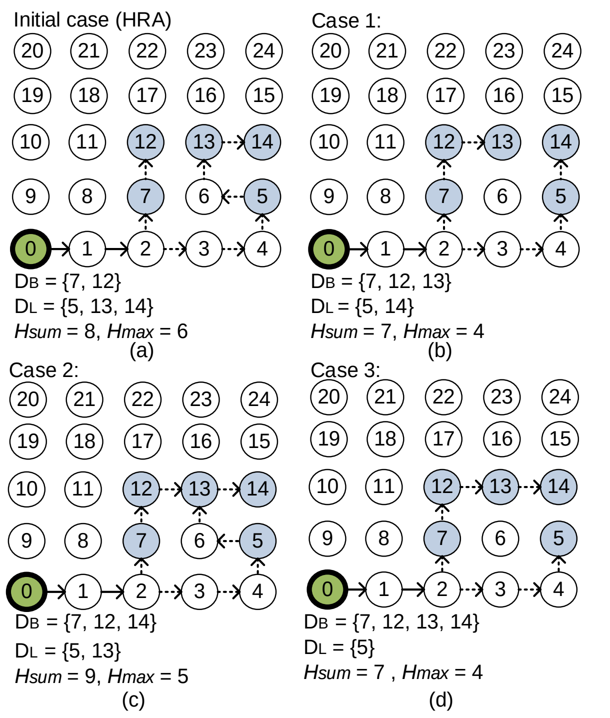


Fig. 16. Various branch conditions for EPBM.

6. N_{ALC}: the nodes in D_L whose labels are bigger (or smaller) than both the current node and N_{FB}.

Let us use Fig. 16 to explain the concept. Fig. 16(a) shows the initial state generated by HRA without Path Balancing at the current node 2, where the *leading path* P_l is 2→3→4→5→6→13→14 and the *branch path* P_b is 2→7→12. Thus, we have $|P_l| = 6$ and $|P_b| = 2$, and hence the hop sum H_{sum} is 8 and the maximum number of hops H_{max} is 6. N_{FB} is labelled 7, and there are two nodes in N_{ALC} which are labelled 13 and 14.

The key idea of the *Exhaustive Path Balancing Method* (EPBM) is to balance the lengths of the *leading path* and the *branch path* by moving nodes *in* N_{ALC} from D_L to D_B in an iterative and exhaustive manner. EPBM examines the possible moves one by one, and each time it finds a move with better H_{sum} and H_{max}, it records the move as the current best solution. This procedure iterates until all possible moves are processed. For our example in Fig. 16, there are 3 ways to move the two nodes in N_{ALC}: moving node 13, moving node 14, or moving both, as shown in Fig. 16(b)-(d). We can see that Case 1 shown in Fig. 16(b) which moves only node 13 yields a better result ($|P_l| = 4$ and $|P_b| = 3$) than the original case as H_{sum} is dropped from 8 to 7 and H_{max} is dropped from 6 to 4. No move after Case 1 obtains a better result.

Note that the minimization of H_{max} implies that $|P_b| \cong |P_l|$ and thus the generated paths are more balanced, while the minimization of H_{sum} implies that the total path length is shortened. Both can lead to better routing performance. However, there exist $2^m - 1$ possible ways of movements, where m is the number of nodes in N_{ALC}. Thus, this EPBM has a complexity of $O(2^n)$ for an n-node NoC. Because of the exponential complexity, we propose a heuristic PBM (HPBM) below.

Algorithm 3. *Heuristic Path Balancing Method (HPBM)*

Input: *Leading direction set* D_L, *Branch direction set* D_B, *leading direction* (dir_L) and *branch direction* (dir_B) generated by HRA as the initial case. // D_B and D_L are in bit string format Coordinates of the *current node* (c_x , c_y), $a \times b$ network, H_{max}: *Maximum number of hops*, and H_{sum}: *Hop Sum*
Output: $D_{L\text{-}best}$ and $D_{B\text{-}best}$: The best result of D_L and D_B.

```
1: Calculate the $H_{max}$ and $H_{sum}$ for $D_B$ and $D_L$
2: $H_{sum\text{-}best} = H_{sum}$; $H_{max\text{-}best} = H_{max}$; $D_{L\text{-}best} = D_L$; $D_{B\text{-}best} = D_B$;
3: $D_B = D_B$ xor $D_L$; $D_L = 0$;
4: if ($dir_B$==North) then $\gamma = \lceil (a - c_y + 1)/2 \rceil$;
5: else if ($dir_B$==South) then $\gamma = \lceil c_y/2 \rceil$;
6: for $i = 0$ to $\gamma - 1$
7:   if ($dir_B$==North) then
8:     $\alpha = L(c_x, c_{y+2i}) + 1$;
9:     $\beta = (c_{y+2i} == (a-1))?\ (a \times b - 1) : (L(c_x, c_{y+2i+1}) - 1)$;
10:  else if ($dir_B$==South) then
11:    $\alpha = L(c_x, c_{y-2i}) - 1$;
12:    $\beta = (c_{y-2i} == 0)?\ 0 : L(c_x, c_{y-2i-1}) + 1)$;
13:  endif
14:  $D_L[\alpha, \beta] = D_L[\alpha, \beta]$ xor $D_B[\alpha, \beta]$; $D_B[\alpha, \beta] = D_B[\alpha, \beta]$ & 0;
15:  $D_B$ and $D_L$ already modified and then calculate the $H_{max}$
     and $H_{sum}$
16:  if (($H_{max} < H_{max\text{-}best}$) && ($H_{sum} < H_{sum\text{-}best}$)) then
17:    $H_{sum\text{-}best} = H_{sum}$; $H_{max\text{-}best} = H_{max}$; $D_{L\text{-}best} = D_L$; $D_{B\text{-}best} = D_B$;
18:  endif
19: endfor
```

4.2.2 Heuristic Path Balancing Method (HPBM)

The HPBM algorithm is shown in Algorithm 3. We use the 5×5 mesh network shown in Fig. 17 to illustrate this algorithm, where $a = b = 5$, node 2 is the current node with $(c_x, c_y) = (2, 0)$, the *leading direction* is east and the *branch direction* is north. Fig. 17(a) shows the result of HRA without path balancing. The *branch direction set* (D_B) contains only two nodes, i.e., nodes 7 and 12, both with the same x-coordinate as the current node. To take advantage of bit-string encoding and reduce the computational complexity, HPBM first moves all nodes in set D_L to D_B using XOR operations as shown in line 3 of Algorithm 3. All nodes in the opposite side of the *leading direction* of the current node (only node 18 in this case) will stay in D_B in the following operation because they are obviously closer to the nodes in the initial set D_B.

We then gradually move nodes at the same side of the *leading direction* back to D_L with the nodes in the two rows that are closest to the current node moved in the first iteration of the loop starting at line 6. In Fig. 16 these two rows are row 0 and row 1 from the bottom of the network and [α:, β] in Algorithm 3 indicates the range of the nodes to be moved, i.e., [α, β] = [3, 6] as shown in Fig. 16(b). If the movement improves the result in terms of H_{sum} and H_{max}, it becomes the best current result. In the next iteration the nodes in the next two rows from the bottom (i.e., rows 2 and 3 with [α…β] = [13..16]) will be further moved back to set D_L as shown in Fig. 16(c), and the movement of nodes in the 4 rows will become the current best case if its result is better than the current best result. The reason to move 4 rows in the second iteration is that the current branch must occur at the current node and the nodes with the same x-coordinate as the current node must be put in D_L. Hence

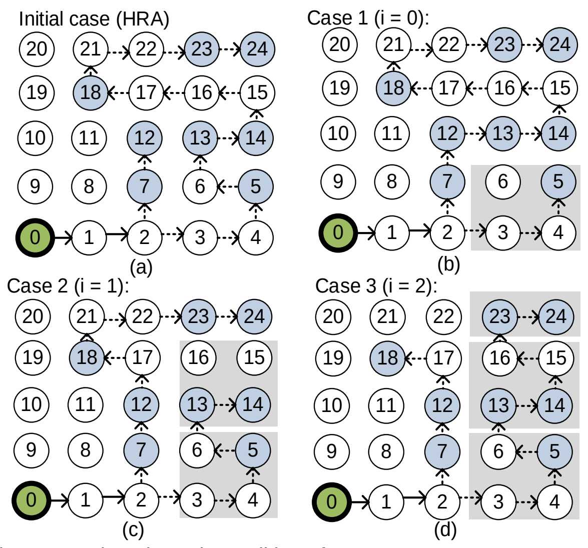


Fig. 17. Various branch conditions for HPBM.

the nodes in the first two rows must be traversed before the nodes in the next two rows. In this way each time nodes in two more rows are moved back and the result is compared to and possibly replaces the best current result, until all rows are processed (see Fig. 16(d)). This procedure will iterate $\gamma = \lceil (a - c_y + 1)/2 \rceil$ ($\lceil c_y/2 \rceil$) times when the branch direction is north (south), where c_y is the y-coordinate of the current node. The complexity of γ depends on the number of rows, i.e., a, in the network, which is $O(\sqrt{n})$ assuming that $a \approx b$. Since this movement can also be done by the constant-time XOR operations as shown in line 14, the time complexity of HPBM is $O(\sqrt{n})$.

After HPBM, the best result obtained during this process determines the final sets D_L and D_B. The resulting complete algorithm is called the Heuristic Path Balancing HRA with Node Balancing (HPBM+NB).

4.2.3 Comparison of HPBM+NB with EPBM+NB

We have invested how HPBM+NB performs compared to EPBM+NB with various settings. We found that the two methods perform similarly and, interestingly, in some cases HPBM+NB performs even better than EPBM+NB. This is probably because HRA is itself a heuristic algorithm. Thus when EPBM is integrated into HRA together with NB (node balancing), the whole EPBM+NB becomes a heuristic one and hence there is a chance that it cannot find a better solution than HPBM+NB. Due to this fact, in all our experiments we will only use HPBM+NB.

5 Deadlock Avoidance and Proof

In this section we prove the deadlock-free property of HRA. Coffman et al. show that a deadlock occurs if and only if four conditions, namely *mutual exclusion*, *hold and wait*, *no preemption* and *circular wait*, all hold simultaneously [29]. Applied to HRA, these four conditions, also known as *Coffman conditions*, can be defined as:

1. *Mutual exclusion*: each output port is not shareable,
2. *Hold and wait*: a packet is holding one output port and waiting for another,
3. *No preemption*: output ports cannot be preempted, and
4. *Circular wait*: a circular chain of packets exists.

We prove HRA to be deadlock-free by showing that the *Coffman conditions* never hold simultaneously. As mentioned earlier, the path-based algorithm following the *label routing rule* has been proven to be deadlock-free in [7]. Thus we only need to prove that the tree-based operations in HRA do not lead to deadlocks.

We first prove that there is no multicast deadlock in a single router, followed by the same in multiple routers.

Theorem 1. *A multicast deadlock in a single router never occurs in HRA.*

Proof. With HRA, at any given instance only the request(s) of one routing computation unit can get the grant from the arbiter of a router. If the routing computation unit only requests one output port, i.e., no branch for the packet issuing the request, then no deadlock can occur. Thus, we only need to consider the case when two packets are to branch, and both request the same two output ports. A deadlock can occur only when one packet holds one port and requests the other port while the other packet holds the port requested by the other packet and requests the port held by the other packet. This cannot occur since in our router architecture, if a packet requests two output ports, the two requests are either both granted, or both declined. Thus, no packet can hold one port while requesting another port in a router and hence no hold and wait condition can occur between any two packets and *Coffman conditions* cannot simultaneous hold at any point of time. As a result, a deadlock that involves only a single router cannot occur. □

Theorem 2. *A multicast deadlock among multiple routers never occurs in our HRA with the stall scheme.*

Proof. A packet branches when it meets one of the two branch conditions in HRA:

Branch Condition I: when a packet is branched under this condition, the proposed switch allocator of the current router, say router A, initiates the branch to a y-direction port, and the y-downstream router, say router B, must have enough unused buffer to store the entire packet. The HRA utilizes the *stall process* described in Section 3.2.2 to transfer a packet atomically (as a single entity) and will not hold the output port of router A in the branch direction once the packet enters router B completely. Also, the packet in router B cannot request an output port of router B before the whole packet enters the buffer of router B, but once the whole packet enters the buffer of router B completely, the output port of router A will be released. Thus, no packet can hold the output port of router A while requesting an output port of router B. This applies to any branched packet satisfying Branch Condition 1 and hence no packet can be in a hold and request (wait) state. Thus, no deadlock can occur according to *Coffman conditions.*

Branch Condition II: when a packet is branched under this condition, the y-downstream router, say router B,

TABLE 1
THE CONFIGURATIONS OF THE NOC AND THE PACKETS INJECTED

Topology	8×8 mesh
Buffer size per input channel	10, 20, 40 (flits)
Packet size	1 head flit 1, 3, 8 data flits 1 tail flit
Traffic	Random distribution, Rent's rule Traffic, Mixed Rent's rule Traffic
Operation frequency	0.2 GHz
Router ports	3 for the four corner nodes 4 for the boundary nodes that are not corner nodes 5 for others

must be a destination of the packet and the corresponding buffer of router B must be empty. Hence the packet entering router B will be sent to the PE of the router without holding the buffer infinitely. Thus, the branched packet will be sent to its destination immediately and will not request or hold any output port. Therefore, the branch also cannot cause any deadlock. □

Theorems 1 and 2 respectively ensure prevention of multicast deadlocks in a single router and multiple routers. As a result, deadlocks never occur in HRA. Since node balancing only affects the partitioning of nodes, it is also deadlock-free. Regarding HPBM, if a branch is due to Condition 1, no deadlock can occur since the packet size is the same for all of its destination nodes. However, if the branch is due to Condition 2, then no node will be moved since no node in D_L can be one hop away from the current node in the y-direction.

6 EXPERIMENTAL RESULTS

6.1 Simulation Environment

We have implemented the proposed hybrid multicast routing approach HPBM+NB and various prior arts on a NoC system with SystemC. As mentioned earlier, setting $k = \lceil b/2 \rceil$ yields the best routing performance of HPBM+NB. In our experiments we use an 8×8 NoC mesh, thus $b = 8$, $k = 4$ and 4CMP are used throughout the experiments. As a performance metric, we define the latency of a packet as the number of cycles required to deliver the packet from the source node to all of its destinations. TABLE 1 shows the configurations of the NoC and the information about the injected packets. Next we describe the experimental results.

6.2 Compare HPBM+NB with Prior Methods

We compare HPBM+NB with 5 previous methods (ACP [7], ARP [22], HOECP [23], APBM [21] and HRA [12]) in various situations (random traffic patterns, different numbers of destinations, different buffer sizes different packet sizes, Rent's rule traffic and mixed Rent's rule traffic). For fair comparison, we use identical total buffer size for all algorithms.

6.2.1 Performance for random traffic patterns

We first compare HPBM+NB against APBM, ACP, ARP, HOECP and HRA with random traffic patterns in a typical cache memory application, where each packet contains three flits (1 head flit, 1 data flit and 1 tail flit). In the *Basic Setting*, we use a buffer size of 20, and each PE sends 100 randomly injected 4-destination multicast packets into the network. We define the *saturation point* as the point at which the latency is twice the zero load latency [30], where the zero-load latency refers to the latency required to send a packet from its source node to all of its destination nodes with no other packet in the network. Fig. 18 shows the average latencies of our algorithm and previous works. It shows that the average latencies of HPBM+NB with the injection rate of 0.01 are 12.9%, 10.0%, 20.6%, 3.6%, and 6.9% smaller than APBM, ACP, ARP, HOECP and HRA, respectively. When the injection rate increases, APBM, ACP, ARP, HOECP and HRA reach their saturation points at injection rates of 0.055, 0.04, 0.04, 0.05 and 0.03, respectively, which are all smaller than HPBM+NB. At these injection rates, the average latencies of HPBM+NB are 31.4%, 32.7%, 40.0%, 45.2% and 45.6% lower than APBM, ACP, ARP, HOECP and HRA, respectively.

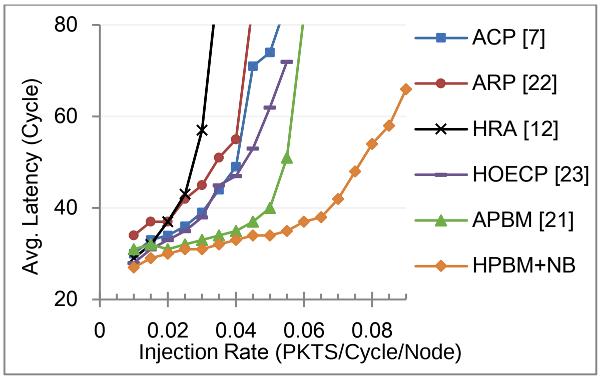


Fig. 18. Performance in 8*8 mesh with 4 destinations, 3 flits and a buffer size of 20 under random traffic.

6.2.2 Performance comparisons in different scenarios

Experiments in different scenarios are conducted. We reuse the Basic Settings in Section 6.2.1 but vary the network parameters. TABLE 2 shows three more settings, Setting 1, Setting 2, and Setting 3, with different numbers of destinations, flits and buffer sizes. The injection rates at the saturation points of all methods are shown in the second column. A consistent observation is that HPBM+NB reaches saturation points at much higher injection rates than all other methods. Column 3 shows the latencies of all methods at their saturation points. Column 4 shows the latencies of HPBM+NB at the saturation points of other methods. For example, in the Basic Setting, the saturation point of APBM is at the injection rate of 0.055 and its latency is 51 at this point, while Column 4 shows our latency is only 35 at this point. Column 5 shows the improvement ratios over the other methods, i.e., the reduction ratios from column 3 to column 4, which range from 30.0% to 57.3%.

Note that in all the above comparisons, we tried to implement all other methods in our environment and were able to obtain similar results as those published in literature except for APBM. Thus, for APBM we directly use the data provided in [21]. We found that APBM may have smaller latencies than our method for some settings when the

TABLE 2
IMPROVEMENT IN AVERAGE LATENCY UNDER VARIOUS SCENARIOS

	Saturation point	Avg. latency	HPBM+NB Avg. latency	Improvement (%)
Basic Setting : 4 destinations, 3 flits, buffer size of 20				
APBM [21]	0.055	51	35	31.4%
ACP [7]	0.04	49	33	32.7%
ARP [22]	0.04	55	33	40.0%
HOECP [23]	0.05	62	34	45.2%
HRA [12]	0.03	57	31	45.6%
HPBM+NB	0.085	58	58	
Setting 1: 8 destinations, 3 flits, buffer size of 20				
APBM [21]	0.045	79	46	41.8%
ACP [7]	0.03	67	40	40.3%
ARP [22]	0.02	67	37	44.8%
HOECP [23]	0.035	67	43	44.9%
HRA [12]	0.035	78	43	35.8%
HPBM+NB	0.08	67	67	
Setting 2: 4 destinations, 5 flits, buffer size of 20				
APBM [21]	0.035	83	46	44.6%
ACP [7]	0.03	63	44	30.2%
ARP [22]	0.03	84	44	47.6%
HOECP [23]	0.035	98	46	53.1%
HRA [12]	0.035	80	46	42.5%
HPBM+NB	0.05	71	71	
Setting 3: 4 destinations, 3 flits, buffer size of 40				
APBM [21]	0.06	82	35	57.3%
ACP [7]	0.06	59	35	40.7%
ARP [22]	0.04	58	33	43.1%
HOECP [23]	0.06	50	35	30.0%
HRA [12]	0.03	60	31	48.3%
HPBM+NB	0.1	51	51	

TABLE 3
AVERAGE LATENCY COMPARISON WITH APBM FOR LOWER INJECTION RATE

Injection rate	0.01	0.02	0.03	0.04	0.05	0.06
Basic Setting : 4 destinations, 3 flits, buffer size of 20						
APBM [21]	31	31	33	35	40	82
HPBM+NB	27	30	31	33	34	37
Improve. (%)	12.9	3.2	6.1	5.7	15	54.9
Setting 1: 8 destinations, 3 flits, buffer size of 20						
APBM [21]	33	35	38	57	NA	NA
HPBM+NB	32	37	40	44	47	50
Improve. (%)	3.0	-5.7	-5.3	22.8	NA	NA
Setting 2: 4 destinations, 5 flits, buffer size of 20						
APBM [21]	38	38	50	NA	NA	NA
HPBM+NB	37	40	44	50	71	256
Improve. (%)	2.6	-5.3	12	NA	NA	NA
Setting 3: 4 destinations, 3 flits, buffer size of 40						
APBM [21]	31	31	33	35	40	82
HPBM+NB	27	30	31	33	34	35
Improve. (%)	12.9	3.2	6.1	5.71	15.0	57.3

injectin rate is between 0.02 and 0.03. We thus show more detailed comparisons with APBM in TABLE 3. For Setting 1 with injection rates of 0.02 or 0.03 and for Setting 2 with injection rate of 0.02, APBM has lower latencies than HPBM+NB. Except for these cases, HPBM+NB has better performance. We believe this is because APBM always takes the shortest path and minimizes the link usage due to its tree-based nature. Hence, when the injection rate is low, the tree-based method can branch without causing too much congestion. However, when the injection rate is high, our adaptive method can determine whether to branch or not depending on the load, hence can lead to better results.

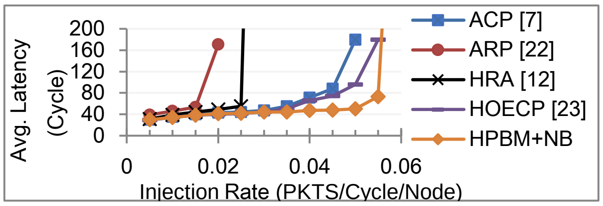


Fig. 19. Performance in 8*8 mesh with 10 destinations, 5 flits and the buffer size of 10 under Rent's rule traffic.

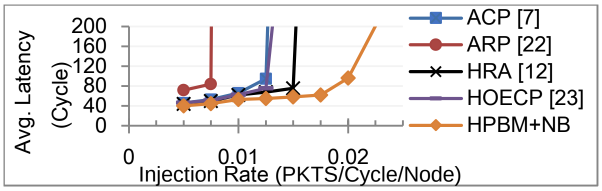


Fig. 20. Performance in 8*8 mesh with 20 destinations, 5 flits and the buffer size of 10 under Rent's rule traffic.

6.2.3 Performance with Rent's rule traffic

The above experiments use uniform random traffic. We also evaluate our algorithm using a traffic generator based on Rent's rule [31] to mimic realistic traffic behaviour [13]. We employ the traffic generator by Bezerra *et al.* [13] that formulates the traffic as $B = bN^p$, where B is the bandwidth to access data in a cluster of N network nodes, b is the average bandwidth per node. Rent's exponent p, $0 \le p \le 1$, models communication locality with smaller p indicating more localized communications. To compare with other methods using the Rent's rule traffic, we employ 8 data flits per packet, a buffer size of 10 and let each PE send 100 injected 10-destination multicast packets into the network with Rent's exponent of 0.75. The results are shown in Fig. 19. When the injection rate increases, ACP, ARP, HOECP and HRA reach their saturation points at injection rates of 0.04, 0.015, 0.045 and 0.025, respectively. At these injection rates, the average latencies of HPBM+NB are 33.8%, 28.3%, 35.1% and 23.6% lower than ACP, ARP, HOECP and HRA, respectively. To further compare the methods, we re-run the experiments with all packets having 20 destinations. The results are shown in Fig. 20. When the injection rate increases, ACP, ARP, HOECP and HRA reach their saturation points at injection rates of 0.0125, 0.0075, 0.0125, and 0.015, respectively. At these injection rates, the average latencies of HPBM+NB are 42.1%, 46.4%, 27.6%, and 23.7% lower than ACP, ARP, HOECP and HRA, respectively. Both experiments show that HPBM+NB can handle high traffic much better than other algorithms.

6.2.4 Performance with mix Rent's rule traffic

This experiment mimics the traffic in distributed shared-memory multiprocessors, in which cache misses cause unicast messages, while updates and invalidations cause multicast messages. We re-run the experiments in Section 6.2.3 and increase the number of flits to 10 under the mixed Rent's rule traffic with 20% multicast and 80% unicast. The results are shown in Fig. 21 and Fig. 22. In Fig. 21, when the injection rate increases, ACP, ARP, HOECP, and HRA reach their saturation points at injection rates of 0.06, 0.05, 0.06, and 0.06, respectively. At these injection rates, the average

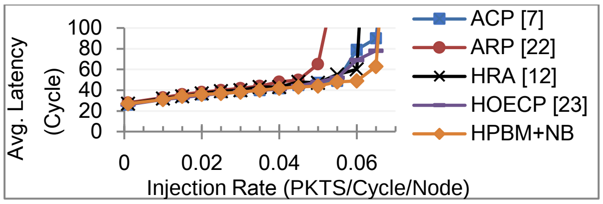


Fig. 21. Performance in 8*8 mesh with 10 destinations, 10 flits and the buffer size of 10 under mixed Rent's rule traffic.

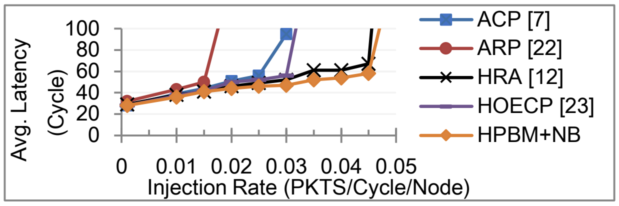


Fig. 22. Performance in 8*8 mesh with 20 destinations, 10 flits and the buffer size of 10 under mixed Rent's rule traffic.

latencies of HPBM+NB are 39.0%, 32.3%, 28.9% and 18.3% lower than ACP, ARP, HOECP, and HRA, respectively. In Fig. 22, with the number of destinations per packet increasing to 20, ACP, ARP, HOECP and HRA reach their saturation points at injection rates of 0.025, 0.015, 0.03 and 0.045, respectively. At these injection rates, the average latencies of HPBM+NB are 17.9%, 18.0%, 16.1%, and 13.4% lower than ACP, ARP, HOECP, and HRA, respectively. Both experiments show that the proposed HPBM+NB also outperforms other methods.

In summary, experiments using random traffic patterns with different numbers of destinations, different buffer and packet sizes, as well as using Rent's rule traffic and mixed Rent's rule traffic have been conducted to demonstrate the effectiveness of HPBM+NB. The results show that when the injection rate increases, all prior routing algorithms reach their saturation points earlier than HPBM+NB, indicating that the proposed algorithm has better performance than prior arts at high injection rates. When the injection rate is low, our method still outperforms all previous methods in most cases.

6.3 Power Dissipation Analysis

A power dissipation analysis between HOECP and HPBM+NB is conducted since HOECP has the lowest power dissipation compared with other methods [23]. Both algorithms' network power dissipation (including input buffers, virtual channel arbiters and crossbar switches) are evaluated based on the power model of ORION 3.0 [32] in a 65nm technology with an operating point of 0.2GHz and a supply voltage of 1V. We re-run the experiment in Section 6.2.3. The average power near the saturation points of HOECP is shown in TABLE 4. With 10 and 20 destinations per packet and injection rates of 0.04 and 0.0125, the average power of HPBM+NB is 7.2% and 11.1% lower than that of HOECP, respectively. This result supports the prediction made in Section 3.3 that by reducing link contention, HPBM+NB reduces power dissipation as well.

TABLE 4
AVERAGE POWER IN 8*8 MESH WITH 10 AND 20 DESTINATIONS UNDER RENT'S RULE TRAFFIC

Num. of destinations	Injection rate	Pavg of HOECP [23]	Pavg of HPBM+NB
10	0.04	11.18 (mW)	10.38 (mW)
20	0.0125	11.12 (mW)	9.97 (mW)

7 CONCLUSIONS

Path-based multicast algorithms tend to generate long routes. To solve the problem, one divides the network into more subsets to shorten the latency. Yet this creates more packets in the system and leads to traffic congestion. Branches in a tree-based multicast system can reduce the lengths of routing paths and shorten the routing latency. However, packet replication increases the traffic load, which also increases the chance to wait for buffer availability and may result in performance degradation.

In this work, we propose a hybrid multicast method to leverage advantages of path-based and tree-based routing methods. Our method utilizes an adaptive approach to determine whether a multicast packet should branch or not according to the buffer usage of the neighboring routers. Two enhanced methods, node balancing and path balancing, are further proposed, which shorten the multicast routes and reduce the number of packets in the network. The proposed algorithm, HPBM+NB, avoids deadlocks without using additional virtual channels or large buffers in the router. Extensive experiments are conducted with different buffer sizes, packet sizes, number of destinations per packet, traffic injection rates and Rent's traffic patterns. The results show that our algorithm reaches the saturation point at higher injection rate than all other methods and have lower latencies in almost all cases at lower injection rates.

REFERENCES

[1] W. O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava, "Multiprocessor SoC platforms: a component-based design approach," *IEEE Design & Test. Computers*, vol. 19, no. 6, pp. 52–63, Nov.-Dec. 2002.

[2] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of Network-on-chip," *J. ACM Computting Surveys*, vol 38, no. 1, pp. 1-51, June. 2006.

[3] H. Xu, P. K. McKinley, E. Kalns, and L. M. Ni, "Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers," *J. Parallel and Distributed Computing*, vol 16, pp. 172-184, Oct. 1992.

[4] X. Lin, P. K. McKinley, and L. M. Ni, "Deadlock-free multicast wormhole routing in 2-D mesh multicomputers," *IEEE Trans. Parallel and Distributed Syst*ems, vol. 5, no. 8, pp. 793-804, 1994.

[5] K. Li, and R. Schaefer, "A Hypercube Shared Virtual Memory," *Proc. Int. Conf. Parallel Processing*, pp. 125-132, 1989.

[6] M. Azevedo, and D. Blough, "Fault-Tolerant Clock Synchronization of Large Multicomputers via Multistep Interactive Convergence," *Proc. Int. Conf. Distributed Computing Systems*, pp. 249-257, 1996.

[7] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and H. Tenhunen, "HAMUM – A novel routing protocol for unicast and multicast traffic in MPSoCs," *Proc. 18th Euromicro Int. Conf. Parallel*

Distributed and Network-Based Processing, pp. 525-532, 2010.

[8] C. J. Glass and L. M. Ni, "Adaptive routing in mesh-connected networks," *Proc. 12th Int. Conf. Distributed Computing Systems*, pp. 12–19, 1992.

[9] C. Ge-Ming, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel and Distributed Systems.*, vol. 11, no. 7, pp. 729-738, Jul. 2000.

[10] D. R. Kumar, W. A. Najjar, P. K. Srimani, "A new adaptive hardware tree-based multicast routing in k-ary n-cubes," *IEEE Trans. Computers*, vol. 50, no.7, pp. 647–659, Jul. 2001.

[11] W. Hu, Z. Lu, A. Jantsch, and H. Liu, "Power-efficient tree-based multicast support for Networks-on-Chip," *Proc. 16th Asia and South Pacific Design Automation Conf.*, pp. 363-368, 2011.

[12] K.J. Lee, C.Y. Chang, and H.Y. Yang, "An efficient deadlock-free multicast routing algorithm for mesh-based networks-on-chip," *Proc. IEEE Int'l Symp. VLSI Design Automation and Test*, pp.1-4, Apr. 2013.

[13] G. B.P. Bezerra, S. Forrest, M. Moses, A. Davis, and P. Zarkesh-Ha, "Modeling NoC traffic locality and energy consumption with Rent's communication probability distribution." *Proc. ACM Int'l Workshop System level interconnect prediction*, pp. 3–8, 2010.

[14] M. Daneshtalab, M. Ebrahimi, S. Mohammadi, and A. Afzali-Kusha, "Low-Distance Path-Based Multicast Routing Algorithm for Network-on-Chips," *J. IET Computer & Digital Techniques*, vol. 3, no. 5, pp. 430-442, Sep. 2009.

[15] F. A. Samman, T. Hollstein and M. Glesner, "Planar Adaptive Router Microarchitecture for Tree-Based Multicast Network-on-Chip," *Proc. Int'l Workshop Network-on-Chip Architecture*, pp. 6-13, 2008.

[16] F. A. Samman, T. Hollstein, and M. Glesner, "New Theory for Deadlock-Free Multicast Routing in Wormhole-Switched Virtual-Channelless Networks-on-Chip," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 4, pp. 544–557, Apr. 2011.

[17] F. A. Samman, T. Hollstein, and M. Glesner, "Adaptive and Deadlock-Free Tree-Based Multicast Routing for Networks-on-Chip," *IEEE Trans. Very Large Scale Integr*ation, vol. 18, no. 7, pp. 1067–1080, Jul. 2010.

[18] L. Wang, Y. Jin, H. Kim, and E. J. Kim, "Recursive partitioning multicast: A bandwidth-efficient routing for Networks-on-Chip," *Proc. IEEE Symp. Networks-on-Chip*, pp. 64-73, May 2009.

[19] Zhong, Ming, et al. "An improved minimal multicast routing algorithm for mesh-based Networks-on-Chip.", *Proc. IEEE Int'l Conf. Signal Processing, Communications and Computing*, pp. 755-779, Aug. 2014.

[20] Li, Jianhua, Chun Jason Xue, and Yinlong Xu. "LADPM: Latency-aware dual-partition multicast routing for mesh-based network-on-chips," *Proc. IEEE Int'l Conf. Parallel and Distributed Systems, pp. 423-430,* Dec. 2010.

[21] Z. Wang, H. Gu, Y. Yang, H. Zhang, and Y. Chen, "An adaptive partition-based multicast routing scheme for mesh-based Networks-on-Chip," *J. Computers & Electrical Engineering*,vol. 51, pp. 235-251, Apr. 2016.

[22] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, J. Flich, and H. Tenhunen, "Path-based partitioning methods for 3D networks-on-chip with minimal adaptive routing," *IEEE Trans. Computers*, vol. 63, no. 3, pp. 718-733, Mar. 2014.

[23] P. Bahrebar and D. Stroobandt, "The Hamiltonian-based odd–even turn model for maximally adaptive routing in 2D mesh networks-on-chip," *J. Computers & Electrical Engineering*, vol. 45, pp. 386-401, Jul. 2015.

[24] Y. H. Kang, J. Sondeen, and J. Draper, "Implementing Tree-Based Multicast Routing for Write Invalidation Messages in Networks-on-Chip," *Proc. IEEE Int'l Midwest Symp. Circuits and Systems*, pp. 1118-1121, Aug. 2009.

[25] S. Ma, N. E. Jerger, and Z. Wang, "Supporting Efficient Collective Communication in NoCs," *Proc. Int'l Symp. High Performance Computer Architecture*, pp. 1-12, Feb. 2012.

[26] F. A. Samman, T. Hollstein and M. Glesner, "Multicast Parallel Pipeline Router Architecture for Network-on-Chip," *Proc. Design, Automation, and Test in Europe.*, pp. 1396-1401, Mar. 2008.

[27] Y.H. Kang, J. Sondeen, and J. Draper, "Multicast routing with dynamic packet fragmentation," *Proc. ACM Great Lakes symp. VLSI*, pp. 113-116, May 2009.

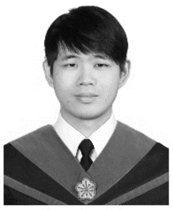
[28] R. V. Boppana, S. Chalasani, and C. S. Raghavendra, "Resource deadlocks and performance of wormhole multicast routing algorithms," *IEEE Trans. Parallel and Distributed Syst*ems, vol. 9, no. 6, pp. 535-549, Jun. 1988.

[29] E. G. Coffmam, M. Elphick, and A. Shoshani, "System deadlocks," *J. ACM Computing Surveys*, vol. 3, no. 2, pp. 67-78, June 1971.

[30] N.E. Jerger, L.S. Peh, and M. Lipasti, "Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support," *Proc. Int'l Symp. Computer Architecture*, pp. 229-240, June 2008.

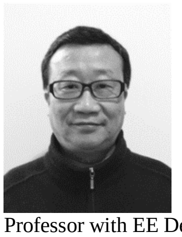
[31] M. Y. Lanzerotti, G. Fiorenza, and R. A. Rand, "Microminiature packaging and integrated circuitry: The work of E. F. Rent, with an application to on-chip interconnection requirements," *J. IBM Research and Development*, vol. 46, no. 4.5, pp. 777-803, July 2005.

[32] A. B. Kahng, B. Lin, and S. Nath, "Explicit modeling of control and data for improved NoC router estimation," *Proc. ACM Conf. Design Automation*, pp. 392-397, June 2012.



Chun-Wei Wu received his bachelor degree in electronic engineering from Chang-Gung University in 2014 and master degree in electrical engineering from National Cheng Kung University in Jan. 2018. He is with MediaTek Inc. now.

Kuen-Jong Lee received his BS, MS and PhD degrees from National Taiwan University (Taiwan), University of Iowa (USA) and University of Southern California (USA), respectively. Since 1991, he has been a faculty member at the Department of Electrical Engineering of National Cheng Kung University (NCKU), Taiwan, where he is a NCKU Distinguished Professor now. Professor Lee invented the Broadcast Scan test technique that has been widely used in industry. He has served as the general/program chairs/co-chairs of several professional events, including IEEE Asian Test Symposium (ATS), VLSI Design, Automation and Test Symposium (VLSI-DAT), Workshop on RTL and High-Level Testing and the 1st International Test Conference in Asia. Professor Lee was the chairman of the steering committee of ATS during 2014-2016. Previously he also served as the Chair of Taiwan IC Design Association and the Director of the SOC Research Technology Center of NCKU. His current research interests include test compression, silicon debug, fault diagnosis, in-field testing, and hardware security. He is an IEEE fellow.



Alan P. Su received his bachelor degree in computer science from Chung-Yuan Christian University in 1986, M.S. degree from University of Missouri Rolla in 1994 and doctorate from University of California Riverside in 1998. Dr. Su has been conducting various Electronic System Level (ESL) tool implementations and methodology research since his doctorate till today, with HP, ITRI, Springsoft (now Synopsys) and GUC. Since 2013 he serves as an Adjunct Associate Professor with EE Department of National Cheng Kung University, teaching SystemC and ESL and conducting ESL related research. At the same time he is the Accellera Representative to Taiwan Region, organizes and hosts Accellera Users' Group Forums in Taiwan.