TPSS: A Flexible Hardware Support for Unicast and Multicast on Networks-on-Chip

Wenmin Hu[1,2], Zhonghai Lu[2], Hengzhu Liu[1], Axel Jantsch[2]
[1]School of Computer, National University of Defense Technology, Changsha, P.R. China
[2]KTH The Royal Institute of Technology, Stockholm, Sweden
[1]{huwenmin,hengzhuliu}@nudt.edu.cn [2]{whu,zhonghai,axel}@kth.se

***Abstract*—Multicast is an important traffic mode that runs on multi-core systems, and an efficient hardware support for multicast can greatly improve the performance of the whole system. Most multicast solutions use the dimension-order routing to generate the mutlicast trees, which are neither bandwidth nor power efficient. This article presents a synthesizable router for network-on-chip (NoC) which supports arbitrarily shaped multicast path based on a mesh topology. In our scheme, incremental setup is adopted to simplify the process of multicast tree construction. For each sub-path setup, we present a novel scheme called two period sub-path setup (TPSS). TPSS is divided into two periods: routing to a predeterminate intermediate router, and updating lookup tables from the intermediate router to destination. This novel setup makes it feasible to support arbitrarily shaped path setup. In our case study, Optimized tree algorithm (OPT) and Left-XY-Right-Optimized tree algorithm (LXYROPT) are proposed for power-efficient path searching, but they need to be pre-configured for the reason of high computation cost. Moreover, Virtual Circuit Tree Multicasting (VCTM) is also supported in our scheme for dynamic construction of multicast path, which needs no computation in path searching. The performance is evaluated by using a cycle accurate simulator developed in SystemC, and the hardware overhead is estimated by using a synthesizable HDL model. Compared to VCTM (without FIFO, multicast table and network adapter), the area overhead of implementing our router is negligible (less than 0.5%).**

***Index Terms*—Network-on-Chip, System-on-Chip, Multicast**

I. Introduction

Many-core architectures have become the mainstream for designing System-on-Chip. Efficient communication among the cores is key to the performance of the whole system. The traditional bus structure works efficiently in systems with limited amount of cores. For MPSoCs with a large number of cores, increased contentions over buses lead to poor performance. The concept of Network-on-Chip (NoC) has emerged as a scalable solution to the global interconnection problem of these systems. Various NoCs have been developed, such as NOSTRUM [1], RAW [2], TRIPS [3], SPIN [4], etc. Furthermore, Intel Teraflop [5] and Tilera [6] have benefited from high communication bandwidth via 2D mesh-based NoCs.

Multicast is commonly seen in large-scale multiprocessor systems which run numerous parallel algorithms, such as parallel search and parallel graph algorithms [7]. In the single-program multiple-data (SPMD) model, the same programm instructions are executed on several processors and some data are processed in parallel, which is still not immune to multicast [7]. In the data-parallel programming model, collective communication plays an important role in improving the performance of replication and barrier synchronization [7]. In distributed shared-memory systems, coherence protocol can benefit from multicast when the shared data are invalidated and updated [7].

Current state-of-the-art NoCs can implement traffic multicasting by replicating multiple unicast messages to different destinations. However, this is inefficient since the limitation of network interface causes late startup times for some messages. It also wastes valuable network bandwidth. Therefore, hardware support for efficient multicasting is desirable for these applications.

Most hardware-based multicasting schemes on regular mesh networks, for example, VCTM [8], use dimension-order routing to generate multicast trees. This is simple, but not efficient in power and bandwidth. Overcoming the shortcomings of previous approaches, this paper has the following main contributions:

• *Multicast path setup*: We propose a novel path setup approach that supports arbitrarily shaped multicast path. In this approach, a two-period setup process is used to construct the sub-paths, and the incremental setup is adopted to combine the sub-paths into a complete multicast tree.

•*Multicast path searching*: We present two power-efficient and bandwidth-efficient tree-based algorithms: Optimized tree(OPT) and Left-XY-Right-Optimized tree(LXYROPT). VCTM [8] is also supported in our router by disabling the first period of the sub-path setup.

•*Incremental multicast setup expansion*: We propose a scheme to utilize an existing multicast path to form a new path instead of evicting the existing path and rebuilding a new one.

The remainder of the paper is organized as follows.

Some parts of this material appeared in proceedings of the 16th Asia and South Pacific Design Automation Conference (ASP-DAC2011) held in Yokohama, Japan, between January 25th and 29th, 2011[W. Hu et.al. 2011]. This paper significantly improves over the ASP-DAC 2011 paper by expanding the validation section, implementing the router in HDL, including a new section (Section VI) that shows the multicast setup using existing path.

Section 2 discusses related work. Section 3 presents the multicast path setup scheme and its protocol. In Section 4, our router architecture is presented. Section 5 introduces three tree-based algorithms: OPT, LXYROPT and VCTM. In Section 6, an approach utilizing an existing multicast path to rebuild a new multicast path is described. Section 7 shows experiment results that validate our approach in performance, power consumption and area overhead. Finally, we conclude in Section 8.

II. Related work

Multicast in off-chip networks were well researched [9]–[12]. The results show that multicast in off-chip networks has outstanding effect on improving performance. Such advantage can also be applied in on-chip communications. Both Æthereal [13] and Nostrum [1] NoCs declared multicast support in their NoC architectures. They used a time-division multiplexing approach to support multicasting. However, verification and evaluation of multicast performance on their NoCs and the way to solve the deadlock problem have not been published so far.

Multicast algorithms can be classified into two categories: *path based* [14]–[17] and *tree based* [8], [18]. In a path-based approach, the number of outgoing ports is limited to at most two, one of which should be the local port. In the path based approach, if the destination nodes are spread widely, it may lead to longer latency compared to the tree-based approach.

In a tree-based approach, the number of outgoing ports to forward packets can be 1 to $n - 1$, where n is the number of outgoing channels that a router has. The advantage of a tree-based method lies in the low latency for message transmission. A hardware support for tree-based multicast named XHiNoC is proposed in [18]. In this scheme, they used ID-manager (IDM) to manage multicast [18]. Each multicast is assigned a table ID at a port, by which the packet can get the routing result. The same multicast packet may get a different ID at a different router. When the packet is transferred to an outgoing channel, the IDM will change the ID field of the packet with the new table ID [18]. Another tree-based routing approach named Virtual Circuit Tree Multicasting (VCTM) is introduced in [8]. Different from XHiNoC, VCTM used an identical ID to manage multicast. VCTM constructs the multicast tree incrementally by sending several unicast setup packets to destinations. Each setup packet is routed using the Dimension-Ordered Routing (DOR) algorithm and the routing result is stored in a table according to the identical ID [8]. For multicast on an irregular network, the work presented in [19] implements multicast using a logic-based broadcast within a domain, which makes isolating of the traffic into different domains possible.

However, the aforementioned approaches [8], [18], [19] cannot support routing on a predetermined path based on special requirements. In this paper, we focus on designing a hardware-based multicast scheme for arbitrarily shaped multicast path. Our scheme inherits the efficiency of VCTM while it provides flexibility to support optimized multicast path.

III. The Multicast Path Setup Scheme

Arbitrarily shaped multicast path support is integrated in our router by using a novel path setup method. In this section, we discuss the packet format supported by our scheme, and offers an example to walk through the process of multicast path setup.

A. Example of multicast path setup

Each multicast forms a tree connecting the source with the destination set, which is identified by a Multicast ID number to each source node and its destination nodes combination. For a tree-based approach, a multicast packet travels along a common path until it arrives at a branch node, where it is replicated and forwarded to corresponding outgoing ports. Once a multicast tree is set up, the packet will be routed based on the multicast table (MCT) number at each router. At the source node, a destination set content addressable memory (CAM) is integrated to record the destination set for multicast trees. Each entry is a *n*-bit vector, where *n* represents the number of nodes on the NoC. If one bit is set, it means that the corresponding node is the destination node. One additional bit indicating whether the entry is valid is also included. At each router, a MCT is partitioned into *n* sub-tables corresponding to each source node. Each sub-table has 16 entries or more.

In our scheme, incremental setup is adopted to simplify the process of multicast path building which is similar to VCTM [8]. In VCTM, each setup packet is routed using the Dimension-Ordered Routing (DOR) algorithm and the routing result is stored in a table according to the identical ID. The sub-path begins from the source node and ends at one destination node. So for a certain source-destination set, the multicast path shape is determinate. However, in our approach, the process of each sub-path setup is divided into two periods (TPSS), where during the first period the setup packet just routes to a predeterminate intermediate node, and during the second period, the packet routes to the real destination node with the routing results stored in the lookup table in the router passing through. The intermediate node is a branch node of the multicast tree, which is determined by a path searching algorithm. A sub-path begins from the intermediate node and ends at one destination node. Multiple unicast setup packets can be injected into the network to setup the sub-paths in parallel. When all are done, a multicast tree is constructed successfully.

Fig. 1 walks through the process of constructing a new multicast tree, which begins at node 0 and ends at node 5, node 7. In the first step, the source node initiates setup packets according to the result of path searching. Here are the packet *A* and *B1* shown in Fig. 1 (c). Packet *A* sets up the path from node 0 to node 7, of which *PKT TYPE* field is set as *MC_SET_2*. The routing information is obtained via lookahead routing unit and encoded into the fields of *X DIR*, *Y DIR*, *X DIS*, *Y DIS* and *OUTPUT PORT*. Different from packet *A*, packet *B1* is set to *MC_SET_1*, which means the setup process would have two periods. Not only the routing information from node 0 to node 4 is encoded into the packet field , but also that from node 4 to node 5 is filled into the first 11 bits of *PAYLOAD* field. Both two packets' *MCT#* fields are set 0 to indicate updating of the first entry of the table .

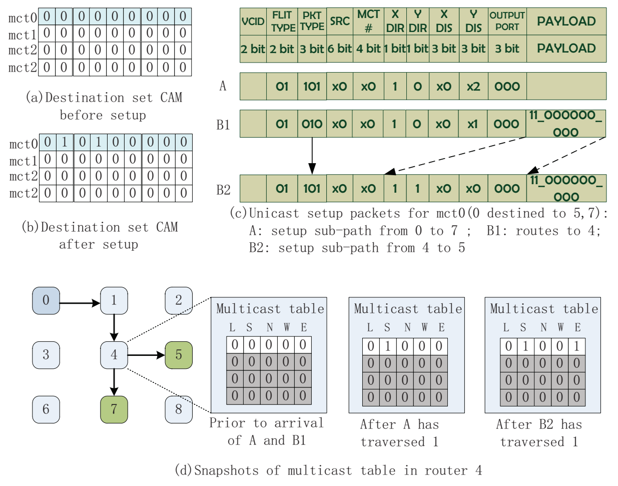

mct0 0 0 0 0 0 0 0 0 0
mct1 0 0 0 0 0 0 0 0 0
mct2 0 0 0 0 0 0 0 0 0
mct2 0 0 0 0 0 0 0 0 0
(a)Destination set CAM before setup
mct0 0 1 0 1 0 0 0 0 0
mct1 0 0 0 0 0 0 0 0 0
mct2 0 0 0 0 0 0 0 0 0
mct2 0 0 0 0 0 0 0 0 0
(b)Destination set CAM after setup
VCID FLIT TYPE PKT TYPE SRC MCT # X DIR Y DIR X DIS Y DIS OUTPUT PORT PAYLOAD
2 bit 2 bit 3 bit 6 bit 4 bit 1 bit 1 bit 3 bit 3 bit 3 bit PAYLOAD
A 01 101 x0 x0 1 0 x0 x2 000
B1 01 010 x0 x0 1 0 x0 x1 000 11_000000_000
B2 01 101 x0 x0 1 1 x0 x0 000 11_000000_000
(c)Unicast setup packets for mct0(0 destined to 5,7): A: setup sub-path from 0 to 7 ; B1: routes to 4; B2: setup sub-path from 4 to 5
0
1
2
3
4
5
6
7
8
Multicast table
L S N W E
Prior to arrival of A and B1
After A has traversed 1
After B2 has traversed 1
(d)Snapshots of multicast table in router 4

Fig. 1. A multicast tree setup example

Each packet is injected into the network sequentially. They are all routed by the XY routing unit. Fig. 1 (d) shows the snapshots of the multicast table in router 4 at different times. Before packets *A* and *B1* reach node 4, the entry bits highlighted are all set to 0. After A has traversed , the bit denoting *S* outport is set to 1. Before packet *B1* arrives at router 4, it routes like unicast with no operation to MCT. After arriving at router 4, it will be changed to packet *B2* as shown in Fig. 1 (c), of which *PKT TYPE* field is reset to *MC_SET_2* and the fields of *X DIR*, *Y DIR*, *X DIS*, *Y DIS* and *OUTPUT PORT* are covered by the first 11 bits of *PAYLOAD* field. Hence, after packet *B2* traverses, the bit denoting *E* outport is set to 1. The similar updates happen in other routers where *MC_SET_2* traverses.

After packets *A* and *B2* have been injected into the destination node, two reply packets are sent to node 0 to acknowledge the setup success. Once node 0 receives the reply packet, the corresponding bit in the destination vector of CAM will be set. Fig. 1 (a)shows the previous content of the CAM while Fig. 1 (b) exhibits the content after getting all the reply packets.

When a multicast destined for 5 and 7 reaches node 0, a destination set matching is performed in the destination set CAM. A MCT number will be generated according to the matching result, which is filled into the *MCT#* field of the packet. Later, it will be used to lookup the MCT at each router to get the outgoing ports. The multicast packet sharing the same links until it arrives at node 4, which is a branch node of the tree. It is replicated and transferred to the downstream router 5 and 7. Finally, two packets are injected into node 5 and node 7 successfully.

B. Protocol

	VCID	FLIT TYPE	PKT TYPE	SRC	MCT#	X DIR	Y DIR	X DIS	Y DIS	OUTPUT PORT	PAYLOAD
	2 bit	2 bit	3 bit	6 bit	4 bit	1 bit	1 bit	3 bit	3 bit	3 bit	payload
Unicast	🟩	🟩 00 Head	🟩 000 UC	🟩	XXX	🟩	🟩	🟩	🟩	🟩	🟩
Multicast Setup_first	🟩	🟩 01 HBT	🟩 010 MC_SET_1	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩 11 bits 2nd DST
Multicast Setup_second	🟩	🟩 01 HBT	🟩 101 MC_SET_2	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
Multicast Normal	🟩	🟩 01 HBT	🟩 110 MC_NORMAL	🟩	🟩						🟩
Multicast Clear	🟩	🟩 01 HBT	🟩 100 MC_CLR	🟩	🟩						
Multicast Clear_reply	🟩	🟩 01 HBT	🟩 001 MC_CLR_RPLY	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
Multicast Setup_reply	🟩	🟩 01 HBT	🟩 011 MC_SET_RPLY	🟩	🟩	🟩	🟩	🟩	🟩	🟩	

🟩 Valid field for corresponding packet kinds

Fig. 2. Packet format supported in the proposed router.

As shown in Fig. 2, packets supported in our router are classified into four categories: *MC_SET_1*, *MC_SET_2* and *MC_SET_RPLY* are multicast setup; *MC_CLR* and *MC_CLR_RPLY* are multicast evicting,; *MC_NORMAL* is multicast data packet; and *UC* is unicast data packet. The fields of packet are defined as follow: *X DIR* and *Y DIR* indicate the directions of destination node, *X DIS* and *Y DIS* are the Manhattan distance between the source node and destination node in the X direction and the Y direction. Since lookahead routing is employed in our router, *OUTPORT* field stores the routing result of downstream router. *MCT#* is the id of multicast table entry. Source address is also encoded in *SRC*. Four different flit types are supported by a 2-bit field of *FLIT TYPE*: Head, Body, Tail and HBT (single flit packet).

TPSS is executed by routing packet *MC_SET_1* and *MC_SET_2*. During the first period, the setup packet *MCT_SET_1* routes like unicast packet until it reaches the intermediate node. On arriving at the intermediate node, the first 11 bits of *PAYLOAD* is replicated to the field of *X DIR*, *Y DIR*, *X DIS*, *Y DIS* and *OUTPORT* to form the new destination while the PACKET TYPE is also changed to *MC_SET_2*. TPSS enters the second period. When the packet traverses a router, the routing result is used to update multicast table entry corresponding to the combination of *SRC* field and *MCT#* field. Once the *MCT_SET_2* reaches destination, a multicast reply packet (*MC_SET_RPLY*) is sent to the source. The other setup packets can be injected into network without waiting for the reply of the former setup packet. Each branch of the tree can be built simultaneously. When the source node receives the replies of all the destination nodes, the setup process is completed.

When a multicast destination set is missed in the CAM and there is no free entry to be utilized, a used multicast tree has to be evicted. Only the source node has the right to evict a multicast tree. *MC_CLR* packet will be routed by looking up MCT just like normal multicast data packet (*MC_NORMAL*). After getting the outgoing ports, the corresponding table entry will be cleared in the next cycle. When the *MC_CLR* packet sinks at the destination node, the destination node will generate a reply packet (*MC_CLR_RPLY*). Once the source node receives all the reply packets, the multicast tree is evicted.

IV. Proposed Router Structure

A special router architecture is proposed to support multicast. We will first introduce the router micro-architecture, and then present the routing mechanism of unicast/multicast. Afterwards, we discuss the packet type conversion approach, which plays a key role in our contributions.

A. Micro-architecture

We use the wormhole router due to its small buffer requirement and high throughput. Fig. 3 shows the architecture of the proposed router. It has five input ports, each of which contains four Virtual Channels (VCs). A register file with five read ports and one write port is integrated into the router to store outgoing ports for the multicast packet. Both unicast and multicast follow the pipeline stages: buffer write/routing computation (BW/RC), switch allocation/virtual channel allocation(SA/VA), switch traversal/link traversal(ST/LT). We use look ahead routing [20] to compute the output port for the next router and store it into the *OUTPUT PORT* field of the head flit. Output directions of head flit in current router is achieved by selecting one from *OUTPUT PORT* field and *OP2* from the multicast table. The criterion for selecting output direction is the packet type. For example, *MC_NORMAL* and *MC_CLR* use OP2 as the output direction while others use the *OUTPUT PORT* field in their head flit. *MCTSG* shown in Fig .3 is the logic block to generate operation signals to the multicast table according to some fields of head flit (*FLIT TYPE, PKT TYPE, SRC, MCT#*). The operations to the multicast table involve reading outgoing ports from the multicast table (*MC_NORMAL, MC_CLR*), setting some bits in one entry of the multicast table (*MC_SET_2*), and clearing all the bits in one entry of the multicast table (*MC_CLR*).

For the multicast packet routing, the result may contain multiple ports. The flit is replicated to one port at one ST/LT stage when successfully getting the grant signal in SA/VA. Only when the flit is successfully transmitted to all the destination ports, can the flit be deleted from the buffer. To keep the state of each multicast flit, the input virtual channel (VC) reserves a separate VC state register and buffer pointers. It is necessary to forward and control the pipeline stage by using the state register and buffer pointers. If the port belongs to the RC results, then its state register will be set to advance to SA/VA. Otherwise the state will be idle. The buffer pointers contain a head pointer for an input VC, and five read pointers for all the destination ports.

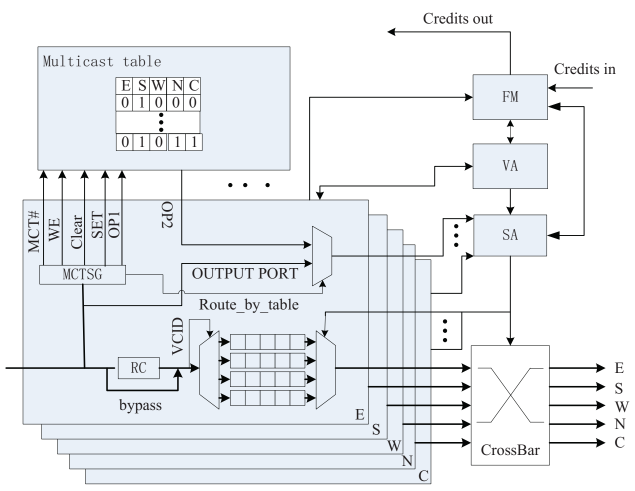
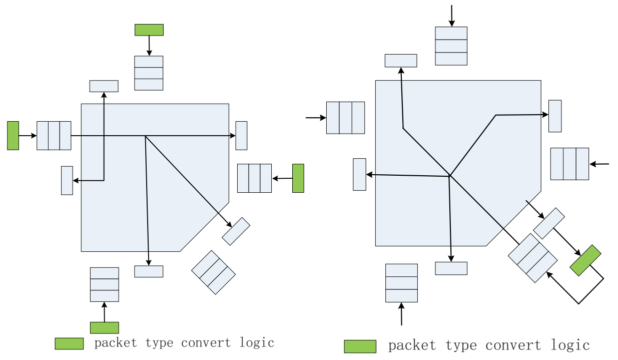


Fig. 3. Router micro-architecture

B. Packet type conversion logic

As mentioned previously, arbitrarily shaped multicast path is supported in our NoCs by injecting multiple setup packets to form the multicast path incrementally. *MC_SET_1* needs to be changed to *MC_SET_2* at an intermediate node. So packet type conversion logic (PTC) should be integrated into the router. Here we propose two solutions to achieve this goal.



(a) Integrated in router (b) Integrated in network adapter

Fig. 4. Packet type convert logic

•The first approach sets PTC in the input port, as shown in Fig. 4(a). In the PTC module, if *MC_SET_1*'s *X DIS* field and *Y DIS* field are 0, it will be changed to *MC_SET_2*. The first 11 bits of *PAYLOAD* is also copied to *X DIR, Y DIR, X DIS, Y DIS* and *OUTPUT PORT*. This routing information is for the downstream router which is computed at the source node, so it is not necessary to compute the output direction due to the change of destination.

•The second approach sets PTC in the network adapter (NA), as shown in Fig. 4(b), which we call it *absorb and re-inject*. When *MC_SET_1* reaches an intermediate router, it is absorbed by the local port, converted by PTC and then re-injected into network after some cycles.

Comparing the two approaches, we conclude that the second is better than the first for the following reasons:

•As shown in Fig. 4, the first approach needs four PTCs while the second needs only one. The function of the PTC is also different between the two approaches. The first needs to check if *X DIS* field and *Y DIS* field are 0 to make sure the current node is an intermediate node; the second does not need to do so, because only the packet arriving at the destination can be injected into the local port. Most importantly, the first one causes more complexity to the arbiter and multiplexer in the router. In some cases, the setup packet needs to turn back via the same port to build the sub-path. So each input port should have five possible output ports. However, the absorb and re-inject method can reduce the number of output ports to four, as shown in Fig. 4(b). Thus, the second approach is more area-efficient.

•The drawback of the second approach lies in possibly longer setup latency, because the packet's entering NA, being converted by PTC, and waiting at the end of the FIFO will consume some cycles. Since most optimized path searching algorithms are expensive and impossible to be implemented in hardware. The applications that have limited number of multicast groups with high reusing rate would be suitable. In this situation, the multicast setup can be pre-configured before application runs. So the two-period sub-path setup is not sensitive to the setup time.

Base on the above reasons, we decided to implement the second approach in our HDL model.

V. Multicast Mechanism and Algorithm Realization

We first propose two multicasting algorithms, which are both tree based. A tree can be decomposed into several node pairs. The first node of the pair is considered as the starting point of a branch (the intermediate node aforementioned), while the second node is the end point. The multicast tree is built incrementally by adding branch one by one to the existing tree. The initial tree is just the source node. The first step of constructing multicast tree is to find all the node pairs that form the tree. The same destinations may be covered by different shape of trees, which may cause different performance and power consumption. Both proposed algorithms are power-efficient and bandwidth-efficient. In the last subsection, we introduce how to implement VCTM [8] in our scheme.

A. Optimized Tree (OPT) Algorithm

OPT is an optimized tree based on the west-first turn model [21], which avoids deadlock on mesh networks. In order to minimize the number of links in the tree, an algorithm similar to the minimal spanning tree algorithm is proposed, which is shown in Fig. 5. D_{pair} is defined as the pair set that forms the tree. D_{node} is the set of the nodes covered by the existing tree, which contains the forwarding nodes that are not the destination nodes. It is used as candidates of branch nodes for the proposed algorithm. D is the set of destinations. The first step is to add the most western node to the multicast tree. Add all the nodes in the path from the source node to the most western node into D_{node}. This makes it possible to find a node later in D_{node} to connect other destination nodes to conform to the west-first turn model. Add the source node and the most western node as an element of D_{pair}. The most western node is removed from D.

Algorithm: Generate the optimized multicast tree based on the west-first turn model
Input: Destination set D, Source node $s : (x_0, y_0)$;
Output: Pair set D_{pair};
Define: $k(a, b) = |a.y - b.y| + |a.x - b.x|$;
Initial: $D_{node} \leftarrow s, D_{pair} \leftarrow \emptyset$;

```
1: Find the node $v \in D, \forall a \in D, v.y \le a.y$. Add $(s, v)$ into $D_{pair}$, remove $v$ from $D$, add the nodes on the path from $s$ to $v$ into $D_{node}$
2: **while** $D$ is not empty **do**
3:     $D_{pair\_tmp} \leftarrow \{(u, v)|u \in D_{node}, v \in D, that \forall a \in D_{node}, \forall b \in D, k(u, v) \le k(a, b)\}$
4:     Select $(u, v) \in D_{pair\_tmp}$, that $\forall(a, b) \in D_{pair\_tmp}$, $v.y \le b.y$
5:     Add $(u, v)$ into $D_{pair}$, remove $v$ from $D$, add the nodes on the path from $u$ to $v$ into $D_{node}$
6: **end while**
```

Fig. 5. Algorithm for generating OPT.

Then the algorithm enters a stage similar to the minimal spanning tree construction when D is not empty. First, find a pair of nodes (u, v) from D_{node} and D which has the shortest distance and conforms to the west-first turn model. If some pairs have the same shortest distance, select the pair whose v is more towards western. This makes it more possible to later find a node pair with the shortest distance and less branches. Second, add (u, v) to D_{pair} . Add all the nodes in the path from u to v to the set D_{node}. Remove v from D. If D is not empty, the sequence will be repeated. Note that it is possible to find a node pair that u and v are the same node, for the reason that the first path may contain the destination nodes. However, this does not matter and we can just put the pair to D_{pair}. When D is empty, the procedure is finished.

B. Left-XY-Right-Optimized Tree (LXYROPT) Algorithm

OPT is a power-efficient and bandwidth-efficient multicasting algorithm which optimizes the multicast tree generation globally by using less links. But this may increase multicast latency. To obtain both low latency and low power consumption, we propose another algorithm named Left-XY-Right-Optimized tree (LXYROPT). In this algorithm, the destination set is partitioned into two subsets. One contains the nodes that lie to the left of the source node, while the other contains the rest. For the destinations that are left of the source node, the XY algorithm is used to generate multicast path. For the rest of nodes, the algorithm takes both the minimum hops for each node and the link sharing into consideration. Fig. 6 shows the details of LXYROPT. For the purpose of optimization, it should first make sure that the routing distance from the source node to a destination node on the multicast tree is the same as the Manhattan distance from the source node to the destination node. Base on this, we select node u from D_{node}, v from $D_{mid-right}$ that the Manhattan distance between u and v is the minimum. This means that a new destination node is added to the existing tree with minimum links. Similar to OPT, pair (u, v) is added into D_{mrpair}, and v is removed from $D_{mid-right}$. The nodes from u to v are also added into D_{node}. If $D_{mid-right}$ is not empty, the sequence will be repeated. Otherwise, the procedure is finished.

```
Algorithm: Generate the LXYROPT multicast tree based on the west-first turn model
Input: Destination set $D$, Source node $s : (x_0, y_0)$;
Output: Pair set $D_{lpair}$, $D_{mrpair}$;
Define: $k(a, b) = |a.y - b.y| + |a.x - b.x|$;
Initial: $D_{node} \leftarrow s, D_{lpair} \leftarrow \emptyset, D_{mrpair} \leftarrow \emptyset$;
 1: $D_{left} \leftarrow \{(x, y)|(x, y) \in D, y < y_0\}$
 2: $D_{mid-right} \leftarrow \{(x, y)|(x, y) \in D, y \geq y_0\}$
 3: while $D_{left}$ is not empty do
 4:     Find a node $v \in D_{left}$, add $(s, v)$ into $D_{lpair}$,
        remove $v$ from $D_{left}$
 5: end while
 6: while $D_{mid-right}$ is not empty do
 7:     $D_{pair\_tmp} \leftarrow \{(u, v)|u \in D_{node}, v \in D_{mid-right}$,
        that $k(s, v) = k(s, u) + k(u, v)$
 8:     Select $(u, v) \in D_{pair\_tmp}$, that
        $\forall(a, b) \in D_{pair\_tmp}$, $k(u, v) \leq k(a, b)$
 9:     Add $(u, v)$ into $D_{mrpair}$, remove $v$ from
        $D_{mid-right}$, add the nodes on the path from $u$ to
        $v$ into $D_{node}$
10: end while
```

Fig. 6. Algorithm for generating LXYROPT.

C. VCTM

VCTM is also supported in our scheme by disabling the first period of the sub-path setup. Compared with OPT and LXYROPT, VCTM does not need computation. When the setup packets are injected into network, they are set as *MC_SET_2* directly. VCTM is suitable under the condition that the multicast group number of running application is larger than the number of table entries and can be reconfigured dynamically. Other optimized algorithms, such as OPT, LXYOPT, etc., are more efficient in bandwidth, performance and power consumption. They are convenient when the multicast group number is fixed and less than the available number of table entries. We can construct and configure the multicast trees before the application runs.

D. An example for proposed algorithms

Fig. 7 shows 3 multicast trees built with different algorithms. Node 36 wants to send a multicast packet to node 9, 10, 3, 20, 29 and 22. VCTM routes the packet following the XY algorithm, and uses about 20 links to connect all the destinations with the source; LXYROPT optimizes the part tree located on the right side of the source node, which costs about 18 links; OPT optimizes the multicast tree globally, and just occupies 14 links. This suggests that our proposed algorithms can reduce the cost significantly.

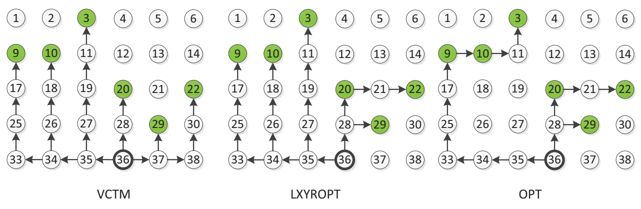


Fig. 7. An example for VCTM, OPT, LXYROPT

VI. Multicast path setup by utilizing existing path

As mentioned previously, if a new multicast destination set is missed in destination set matching in the CAM, an existing multicast tree will be selected and evicted. The replacement policy can be LRU or FCFS, etc, which we do not discuss in detail. A *MC_CLR* packet is injected into the network to clear the corresponding entry of the multicast table in the router on the path. The packet is routed like multicast data packets, so it will reach all the routers on the multicast tree. After the packet is injected into the destination node, a *MC_CLR_RPLY* is sent back to the source node. Once all the reply packets from destination nodes are received at the source node, the evicting process is finished. Then the source node sends out multiple setup packets to construct the new multicast tree.

Generally, the setup process of a new multicast tree consists of evicting and constructing. However, we find that if the destination set to be evicted is a subset of the new multicast destination set, it is unnecessary to evict and rebuild. It just needs to inject the setup packets into the network to add some destinations to existing multicast tree. This reduces not only the overhead of setup time but also the number of packets injected into the network. To realize this, only a little hardware resource is needed.

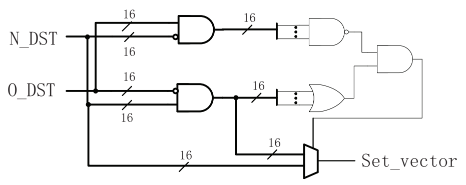


Fig. 8. The logic of setup vector generator

Fig. 8 shows the logic of setup vector generator. Here the destination vector is 16 bits, which means that the number of total nodes is 16. If the *n*th bit is set, it means that the *n*th node belongs to the destination set. *N_DST* denotes the new destination set while *O_DST* is the one to be evicted. As for the input signals of the multiplexer, one is the original vector *N_DST* while the other is the result of *N_DST* minus *O_DST*. The selected signal is generated by the judgment logic which indicates whether *O_DST* is a subset of *N_DST*. If so, the vector resulted from *N_DST* minus *O_DST* outputs as *Set_vector* to initiate the setup packets. Otherwise, the *N_DST* outputs directly as *Set_vector*. Before sending out the setup packets, the evicting process must be completed first.

VII. Experimental Evaluation

We developed a simulator in SystemC. The simulator is a wormhole on-chip network with the mesh topology. The number of virtual channels is set to 4, while the depth of FIFO is 5. The credit-based flow control is used to avoid buffer overflow.

To verify the flexibility of our scheme, we simulated VCTM, OPT and LXYROPT on the simulator. The performance was obtained by evaluating them with different synthetic multicast workload and mixed traffic. We define the packet latency as the interval between the packet entering to source queue and the tail flit being absorbed by the destination node.

A. Multicast traffic profile

We evaluated multicasting performance with different network sizes. To facilitate the explanation, the simulation configuration parameter is defined as (a, b, c, d), where a indicates the network size, b denotes the simultaneous source number at each injection slot, c is the destination set size range, and d is the number of flits for a packet. For example, (256, 4, 5-10, 3) means in each injection slot, four sources send a 3-flit packet to 5-10 destinations on a 16×16 NoC. The source and destination are uniformly distributed. The number of destination is selected randomly ranged from 5 to 10.

We set two scenarios: (64, 8, 5-20, 5), (256, 8, 10-40, 5). Since OPT and LXYROPT are off-line algorithms, the setup time is not included in the performance comparison, we assume 100% entries of multicast table reusable for VCTM. As can be seen in Fig. 9, LXYROPT outperforms other algorithms. Compared with VCTM , LXYOPT is about 2%-4.5% lower in terms of packet latency while OPT is about 10%-22% higher; The results are consistent with our expectations. Small-sized packets (here are 5 flits) are sensitive to the maximum hops between the source and destination. LXYROPT and VCTM, in which the distance between destination to source is always minimum, perform better than others. OPT, of which the purpose is to get a bandwidth-efficient and power-efficient tree, may increase the distance by connecting the destination node to branch node which is far away from the source node during the path searching.

B. Multicast and Unicast (mixed) traffic profile

We also investigated the performance of mixture of unicast and multicast traffic. The multicast traffic is set as (64, 2, 5-20, 5), (256, 2, 10-40, 5). The unicast is also uniformly distributed, and the injection rate is the same as the multicast. As can be seen in Fig. 10, LXYROPT still delivers better performance than others.

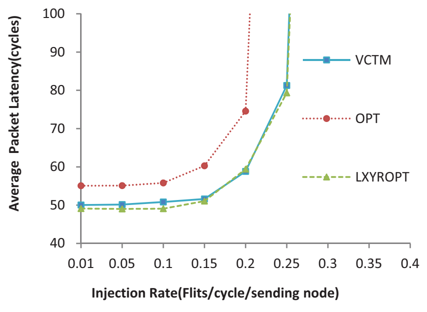
C. large-sized packet performance

We also investigated the performance of large-sized packets for these algorithms. In Section 5.4, Fig. 7 gives an example showing the distribution of source-destination communication under different algorithms. We selected it as the experiment configuration. We evaluated the performance with different packet sizes under both multicast only traffic and mixed traffic. As can be seen in Fig. 11, when the packet size is small($a1$ and $a2$), LXYROPT and VCTM outperform OPT, which reduce latency about 20%. In this situation, the maximum manhattan distance plays a key role in performance. As the packet size increases, the maximum manhattan distance becomes negligible, and the packet size is more important. In scenario $a5$, the difference between LXYROPT and OPT is only 0.5%. VCTM gives the worst result in performance, because node 36 has 3 branches, and our router transfers one flit to only one desired output port at one ST stage, the bandwidth is only 1/3 of the unicast. For other algorithms, the maximum branch number in the tree is 2. Hence, the bandwidth is about 1/2 of the unicast. The latency of VCTM should be about 1.5 times as that of other algorithms. The result shown in Fig. 11 $a5$ is consistent with our expectation. Although the experiment is a special case, and it may not allow for general conclusion, it is certain that when the network load is low, the large-sized packet affects the latency significantly [22].

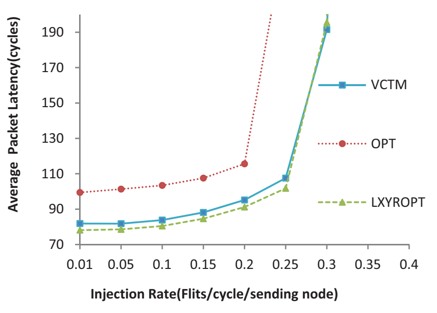
In the mixed traffic scenario, unicast is uniformly distributed. In this scenario, the packet latency contains the waiting time. Waiting time includes the time waiting for routing and switching. The result shows that LXYROPT gives the best performance when the packet size is large. VCTM is still worst due to its bottleneck in node 36.

D. Multicast path setup

We also evaluated the efficiency of our multicast path setup scheme. As aforementioned, the existing multicast path is used to construct a new multicast path when it is a subset of the new one. Fig. 12 shows the mapping relation between the nodes on the mesh and the bits in the destination set vector. We set four different scenarios for multicast setup, which are shown in Table I. They are different in old group size and new group size. From the simulation results in Fig. 13, we can conclude that if the multicast group to be evicted is a subset of the new one, our scheme can significantly reduce the setup cycles. The performance improvement comes from two main sources: elimination of evicting process and reduction of setup

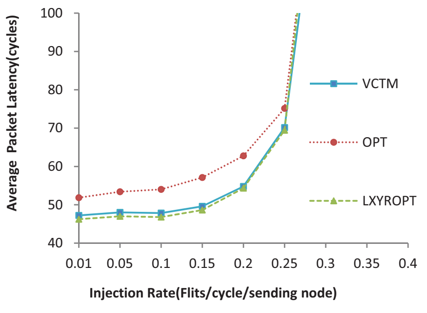


(a) (64,8,5-20,5)

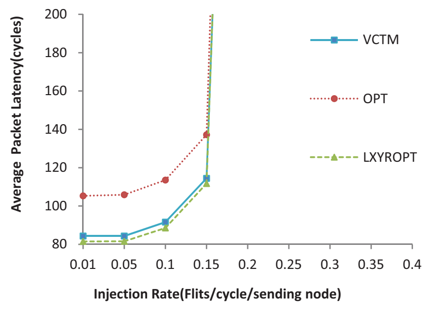


(b) (256,8,10-40,5)

Fig. 9. Multicast packet latency under the multicast only traffic



(a) (64,2,5-20,5)



(b) (256,2,10-40,5)

Fig. 10. Multicast Packet latency under the mixed traffic

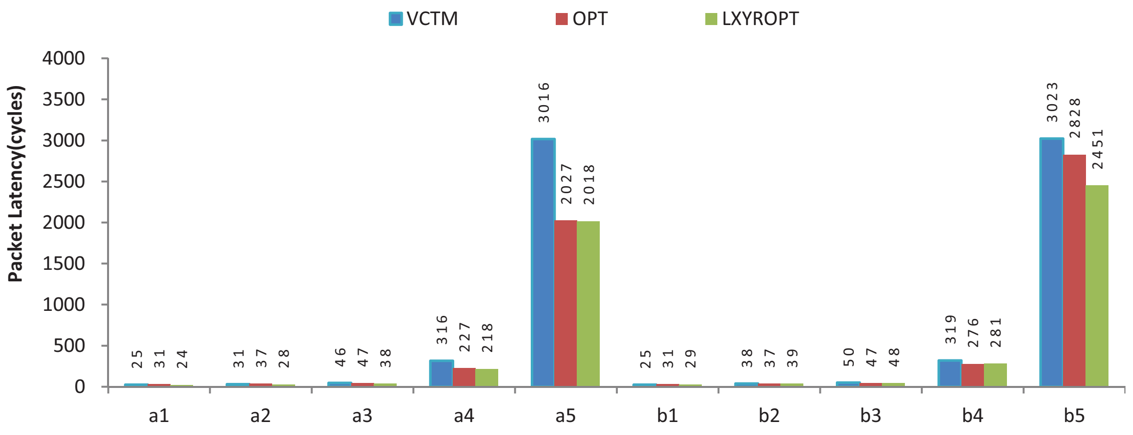


Fig. 11. Packet latency. Series a is the multicast only traffic. a1: 1 flits/packet; a2: 5 flits/packet; a3:10 flits/packet; a4: 100 flits/packet; a5:1000 flits/packet. Series b is the mixed traffic, other nodes (except the source node of multicast) send unicast packets at the rate of 0.1 (flits/cycle/sending node), the unicast traffic is uniformly distributed. b1: 1 flits/packet; a2: 5 flits/packet; b3:10 flits/packet; b4:100 flits/packet; b5:1000 flits/packet.

packets. This scheme is only suitable for the VCTM, which can re-configure the multicast tree at runtime.

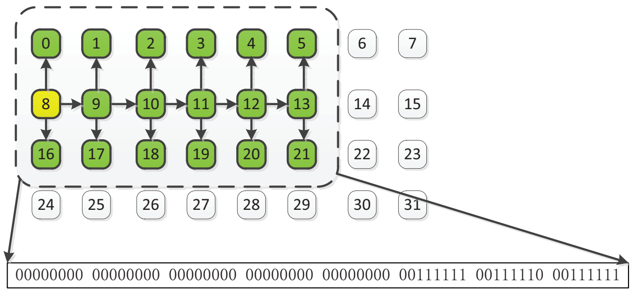


Fig. 12. mapping relation between nodes to vector

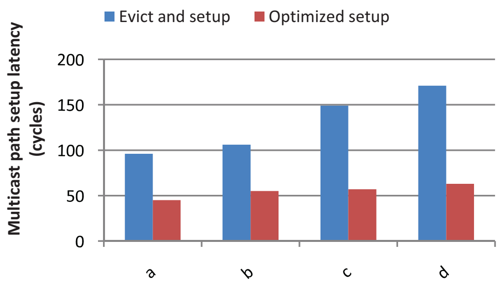


Fig. 13. Setup latency

E. Power

We calculated the power consumption by using the library of Noxim [23]. In this power model, the power

TABLE I
DESTINATION SET FOR MULTICAST

scenario	multicast tree (bit vector) /source node: 8
a	old: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011111
	new: 00000000 00000000 00000000 00000000 00000000 00000000 00111110 00011111
b	old: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011111
	new: 00000000 00000000 00000000 00000000 00000000 00011111 00111110 00011111
c	old: 00000000 00000000 00000000 00000000 00011111 00011111 00111110 00011111
	new: 00000000 00000000 00000000 00011111 00011111 00011111 00111110 00011111
d	old: 00000000 00000000 00000000 00011111 00011111 00011111 00111110 00011111
	new: 00000000 00000000 00011111 00011111 00011111 00011111 00111110 00011111

TABLE II
POWER PARAMETER

Operation	energy(nJ)
routing	0.185
incoming	0.002
selection	0.006
forwarding	0.384
standby	0.00005

consumption contains the power of routing, selection, forwarding, incoming and standby, which is added to the total consumption when corresponding operation happens during the simulation. Table II shows the energy for each operation. We ignored the static power of multicast table which is sensitive to the size of register file. Furthermore, OPT, LXYROPT and VCTM are integrated in the same router, they should consume the same static power. The power consumption of VCTM is normalized as 1. We calculated the power consumptions in 7 groups of experiments. s1(256, 8, 10-40, 5), s4(64, 8, 5-20, 5) are executed in the multicast only traffic mentioned previously, while a1-5 are executed in the large-sized packet evaluation.

As can be seen from Fig. 14, OPT achieves the most power reduction. Compared with VCTM, it saves 16%-31% power dissipation. LXYROPT, on the other hand, reduces about 7%-12%. The power saving comes from reducing redundant link switching and buffer operations.

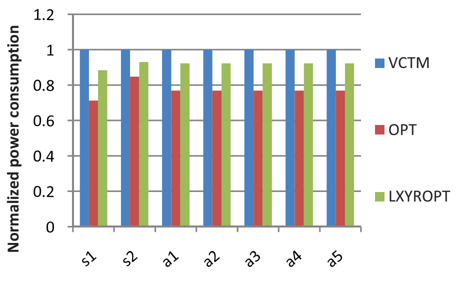


Fig. 14. Power consumption under only multicast traffic

F. Area comparison

Our router and the baseline unicast router have been synthesized using a CMOS stand-cell technology library from Chartered Semiconductor Manufacturing. Table III shows the synthesis results with 90-nm CMOS stand-cell technologies. Compared with the unicast router, area of each component increases significantly for our router. Unlike unicast, maintaining the status of information for multicast packet is more complicated, and about four times registers are needed to store the information of multi-port and virtual channel id. More arbiters, multiplexer and other combinational logic are also needed to be integrated. With respect to FIFO (depth:5 width: 44 bits), our router increases about 30% due to the asynchronous replication mechanism. The Packet Conversion Logic (PTC) is set in NA (Network adapter) which leads to about 3% more area overhead. Since VCTM [8] did not present the router micro-architecture in detail, we assume that it is similar to our proposed router in VA, SA, SW, FM (Flow Manager) and RC. FIFO, NA and MCT are sensitive to the size of buffers, so we ignore the area of them in comparison to VCTM. PTC only brings about extra 0.5% area compared with VCTM, though the area of FIFO, NA and MCT is excluded.

TABLE III
ROUTER AREA BREAKDOWN (μm^2)

	VA	SA	SW	FM	RC	FIFO	MCT	NA
unicast	21848	2962	6346	6332	6860	123200	0	12085
proposed	39518	6635	6346	6881	8985	159940	113164	(PTC)12443

VIII. CONCLUSIONS

In this paper, a flexible multicast support scheme (TPSS) is proposed. Its basic idea is to divide the sub-path setup into two periods: routing to an intermediate node, and updating the multicast table from the intermediate node to the destination node. It supports arbitrarily shaped multicast path construction by adding little logic to the existing VCTM router. VCTM can be easily supported in our router by disabling the first period of the sub-path setup. Two power-efficient and bandwidth-efficient algorithms, OPT and LXYROPT, are proposed to explore this scheme. Experimental results show that LXYROPT gives the best performance and a modest power reduction. OPT achieves the most power saving with the cost of decreased performance, which is suitable for the scenario that is sensitive to power dissipation. We also optimize the setup scheme by utilizing the existing multicast path when it is a subset of the new one. Simulation results indicate it can reduce the setup latency significantly.

For future work, since increasing the network size may cause the multicast table to consume large hardware area, we plan to explore the dynamic partition scheme to save area.

ACKNOWLEDGEMENT

This work is supported by the National Science Foundation of China, under grant No.60970037, and Doctor Program Foundation of Education Ministry of China,under grant No. 20094307110009 and No. 20114307130003.

References

[1] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proceedings of the Design Automation and Test in Europe Conference*, February 2004.

[2] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff *et al.*, "The Raw microprocessor: A computational fabric for software circuits and general purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[3] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim *et al.*, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *Proceedings of the 30th annual international symposium on Computer architecture*, February 2003.

[4] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2000.

[5] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[6] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. C. Miao, J. F. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[7] X. Lin, P. K. McKinley, and L. M. Ni, "Deadlock-free multicast wormhole routing in 2-d mesh multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 8, pp. 793–804, 1994.

[8] N. E. Jerger, L. S. Peh, and M. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in *Proceedings of the 35th annual international symposium on Computer architecture*, June 2008.

[9] M. P. Malumbres and J. Duato, "An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors," *Journal of Systems Architecture*, vol. 46, no. 11, pp. 1019–1032, 2000.

[10] J. S. Turner, "An optimal non-blocking multicast virtual circuit switch ," in *IEEE international conference on Computer Communications(INFOCOM)*, 1994, pp. 298–305.

[11] M. Chiang and L. Ni, "Multi-address encoding for multicast," *Parallel Computer Routing and Communication*, vol. 853, no. 5, pp. 146–160, 1994.

[12] R. Sivaram, D. K. Panda, and C. B. Stunkel, "Efficient Broadcast and Multicast on Multistage Interconnection Networks using Multiport Encoding," in *8th IEEE Symposium on Parallel and Distributed Processing(SPDP)*, 1996, pp. 36–45.

[13] E. Rijpkema, K. Goossens, J. D. A. Rădulescu, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip," *IEE Proceedings: Computers and Digital Technique*, vol. 150, no. 5, pp. 294–302, September 2003.

[14] X. lin and L. M. Ni, "Multicast communication in multicomputer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 10, pp. 1105–1117, 1993.

[15] Z. Lu, B. Yin, and A. Jantsch, "Connection-oriented multicasting in wormhole-switched networks on chip," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, Karlsruhe, Germany, March 2006, pp. 205–210.

[16] R. V. Boppana, S. Chalasani, and C. S. Raghavendra, "Resource deadlocks and performance of wormhole multicast routing algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 6, pp. 535–549, 1998.

[17] M. Daneshtalab, M. Ebrahimi, S. Mohammadi, and A. Afzali-kusha, "Low-distance path-based multicast routing algorithm for network-on-chips," *IET computers & digital techniques*, vol. 3, no. 5, pp. 430–442, 2009.

[18] F. A. Samman, T. Hollstein, and M. Glesner, "Multicast parallel pipeline router architecture for network-on-chip," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2008.

[19] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, "Efficient Unicast and Multicast Support for CMPs," in *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture*, 2008, pp. 364–375.

[20] M. Galles, "Scalable Pipeline Interconnect for Distributed Endpoing Rouing: The SGI SPIDER Chip ," in *Proc. Hot Interconnect*, 1996, pp. 141–146.

[21] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proceedings of the 19th annual international symposium on Computer architecture(ISCA)*, 1992, pp. 278–287.

[22] M. Moadeli and W. Vanderbauwhede, "Communication modeling of multicast in all-port wormhole-routed NoCs," *Journal of Systems and Software*, vol. 83, no. 8, pp. 1327–1336, 2010.

[23] "http://noxim.sourceforge.net/projects/noxim."

Wenmin Hu received the B.S. degree in Computer Science and Technology from Northwestern Polytechnical University, China in 2005, and M.S. degree in Electronic Science and Engineering from National University of Defense Technology(NUDT), China in 2007. He is currently working toward the Ph.D. degree in the School of Computer Science, NUDT. His research focuses on Network-on-Chip and multicast.

Zhonghai Lu received the B.S. degree from Beijing Normal University, China in 1989 and the M.S. and Ph.D. degrees from KTH The Royal Institute of Technology, Stockholm, Sweden, in 2002 and 2007, respectively. He is currently an Associate Professor at KTH. His research interests include computer systems and VLSI architectures, system modeling, refinement and synthesis, and design automation.

Axel Jantsch received a Dipl.Ing. (1988) and a Dr. Tech. (1992) degrees from the Technical University Vienna, Austria. He is currently a professor at KTH. His research interests include Embedded systems, Network-on-Chip, VlSI and SoC Design, HW/SW Codesign, and Design methodology.

Hengzhu Liureceived the Ph.D. degree in computer science from National University of Defense Technology (NUDT), China in 1999. Currently, he is a professor at micro-electronics and microprocessor institute of NUDT. His research interests include VLSI design, processor architecture and microarchitecture and Network-on-Chip.