International Journal of Parallel Programming, Vol. 33, No. 4, August 2005 (© 2005)
DOI: 10.1007/s10766-005-6468-8

Efficient Broadcast in Heterogeneous Networks of Workstations Using Two Sub-Networks

Chao Lin[1,2] and Jang-Ping Sheu[1]

Received June 11, 2004; revised January 8, 2005; accepted January 21, 2005

This paper presents efficient algorithms for broadcasting on heterogeneous switch-based networks of workstations (HNOW) by two partitioned sub-networks. In an HNOW, many multiple speed types of workstations have different send and receive overheads. Previous research has found that routing by two sub-networks in a NOW can significantly increase system's performance (*Proc. 10th International Conference on Computer Communications and Networks*, pp. 68–73, 2001). Similarly, EBS and VBBS (*Proc. 8th IEEE International Symposium on Computer and Communication*, pp. 1277–1284, (2003)), designed by applying the concept of fastest nodes first, can be executed in $O(nlog(n))$ time, where n is the number of workstations. This paper proposes two schemes TWO-EBS and TWO-VBBS for broadcasting in an HNOW. These two schemes divide an HNOW into two sub-networks that are routed concurrently and combine EBS and VBBS to broadcast in an HNOW. Based on simulation results, TWO-VBBS outperforms EBS, VBBS, VBBSWF (*Proc. 8th IEEE International Symposium on Computer and Communication*, pp. 1277–1284, (2003)), the postorder recursive doubling (*Proc. Merged IPPS/SPDP Conference*, pp. 358–364, (1998)), and the optimal scheduling tree (*Proc. Parallel and Distributed Processing Symposium, Proc. 15th International* (2001)) generated by dynamic programming in an HNOW.

KEY WORDS: Wormhole routing; myrinet; heterogeneous networks of workstation; network partitioning; up*/down* routing; unicast-based broadcast; postorder recursive doubling algorithm.

[1]Department of Computer Science and Information Engineering, National Central University, Chung-Li, 320, Taiwan. E-mail: {chaolin, sheujp}@axp1.csie.ncu.edu.tw
[2]To whom correspondence should be addressed.

1. INTRODUCTION

This paper considers to design an efficient broadcast routing algorithm for wormhole routed heterogeneous switch-based network of workstations (HNOWs). These networks consist of a collection of routing switches and workstations interconnected in some arbitrary topology.

Efficient broadcasting in an HNOW has been presented.[1] In this paper, two schemes referred to as SPOC and FNF are proposed. The SPOC first constructs a binomial scheduling tree, then chooses faster workstations to nodes in the binomial tree which have to send more messages. The FNF uses the idea of Fastest-Node First. Although FNF is a near optimal scheme, it can not guarantee that it is a contention-free routing. If contention occurs, the system should be delayed. The optimal solution is an NP-complete problem.[1] The running time for finding the optimal solution is $O(n^2 2^{2n})$.[1] Singhal *et al.*[14] used the optimal scheduling tree in a NOW to design a broadcast algorithm in an HNOW with two speed types.[8] The optimal tree algorithm that is generated by the dynamic programming has time complexity $O(n^3)$, but it can not provide a routing with more than two speed types. Libeskind-Hadas and Hartline[7] proposed that solutions within a bounded ratio of the receive overhead to the send overhead can be found in time $O(n \log(n))$, when the ratio among nodes is bounded by constants. If the number of distinct types of workstations is fixed then optimal solutions can be found in polynomial time. Banikazemi *et al.*[2] proposed a new model, henceforth referred to as the heterogeneous receive–send model, which associates both a send and a receive overhead with each workstation. They also verified that this model is suitable for a heterogeneous NOW test-bed.

Chao presented three contention-free routing algorithms with the complexity of $O(n \log(n))$ in an HNOW with multiple send and receive speeds.[9] These algorithms are contention-free and very efficient for routing in an HNOW. They can outperform the optimal scheduling tree presented by Singhal *et al.* with two speed types in an HNOW. In this paper, we present two schemes which combine network partitioning[10] with VBBS and EBS[9] to broadcast in an HNOW. From simulation results, TWO-VBBS outperforms all other previous schemes.

The receive and send model as presented by Banikazemi *et al.*[2] is also used here. The system model is modified by adding the wormhole routing to perform our experiments. Any two workstations can have different speeds as illustrated from Tables I to IV. Our proposed new model involves wormhole routing, which differs from the previous model.[2]

The rest of this paper is organized as follows: Section 2 presents preliminary. Section 3 proposes previous broadcasting strategies. Section

Table I. Send and Receive Parameters (slow speed)

Type	$S_c(\mu s)$	$S_m(\mu s)$	$R_c(\mu s)$	$R_m(\mu s)$
1	60	0.05	110	0.03
2	90	0.40	140	0.06
3	120	0.80	170	0.48
4	150	1.6	200	0.96
5	180	2.40	230	1.92
6	300	3.2	360	2.2
7	400	3.8	500	2.8
8	500	4.2	600	3.2

Table II. Send and Receive Parameters (fast speed)

Type	$S_c(\mu s)$	$S_m(\mu s)$	$R_c(\mu s)$	$R_m(\mu s)$
1	60	0.05	110	0.03
2	90	0.20	140	0.12

Table III. Send and Receive Parameters (slow speed)

Type	$S_c(\mu s)$	$S_m(\mu s)$	$R_c(\mu s)$	$R_m(\mu s)$
1	60	0.05	110	0.03
2	360	0.50	440	0.3

Table IV. Send and Receive Parameters (slow speed)

Type	$S_c(\mu s)$	$S_m(\mu s)$	$R_c(\mu s)$	$R_m(\mu s)$
1	60	0.05	110	0.03
2	90	0.40	140	0.32
3	120	1.6	170	1.28
4	150	6.4	200	4.8

4 proposes Two-Sub-networks broadcasting scheme. Section 5 presents broadcasting on an HNOW using TWO-VBBS. Broadcasting on a partially connected NOW is in Section 6. Section 7 analyzes performance. Section 8 presents the simulation results and comparison. Section 9 presents conclusions.

2. PRELIMINARIES AND RELATED WORKS

2.1. Preliminaries

2.1.1. Switch-Based Network of Workstations

Workstations in the HNOWs have different receive and send latency. An HNOW can be represented as $G(V, E)$, where V=$V_1 \cup V_2$, V_1 represents the set of all the switches; V_2 is the set of all workstations, E is the set of all the links. Each switch has many workstations and includes multiple ports connected by workstations. The links are of three types. The first type is a link from one workstation to another workstation within a single switch. The second is a link from one switch to another switch, and the third is a link from a workstation to a switch. For each directed edge $(u, v) \in E$, the opposite edge is (v, u), and both edges are bi-directional edges. In Autonet and Myrinet, these two opposite edges (u, v) and (v, u) can be used simultaneously without contention. Following above, the graph $G(V, E)$ is said to be symmetric if vertex u is connected to vertex v and they have two opposite edges between them. If any vertex in V_2 is connected to a single vertex in V_1, then a workstation in V_2 is connected to a switch in V_1. If two vertexes u, v in the V_1 are connected, then the two switches are said to be connected. Figure 1 illustrates this system model. For example, Fig. 1(a) depicts a heterogeneous switch-based network. Figure 1(b) shows its graph representation. Figure 1(c) represents a spanning tree. The small symbols such as the circle and the square indicate distinct workstations with different speeds. Worms can pass through many switches, but can not be absorbed by them. First, we should construct a BFS spanning tree as in Fig. 1(c), which is used for routing. Furthermore, *the density* is defined as the ratio of the number of workstation to the number of switch; it is used in our experiments. The higher the density, the better performance should be.

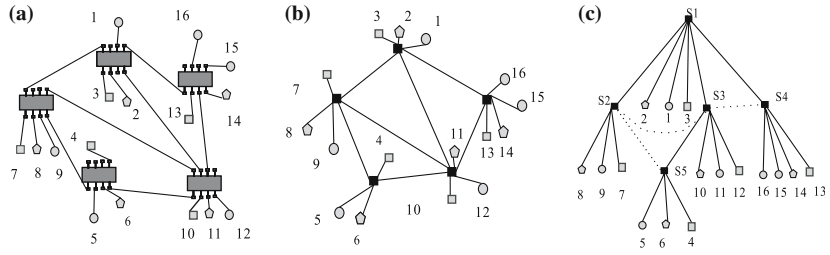

Fig. 1. (a) an HNOW example, (b) its graph representation, and (c) its spanning tree. The dotted lines in (c) are cross edges. $S_1, S_2, \ldots,$ and S_5 are switches.

2.1.2. Unicast-Based Broadcast

Suppose that $G(V, E)$ is an HNOW, V is the set of all workstations and switches, and E is the set of all edges, *a directed trail* in the network is an alternating sequences of vertices and edges $v_0, e_0, v_1, e_1, \ldots, v_{k-1}, e_{k-1}, v_k$, such that $e_i = (v_i, v_{i+1}) \in E$, $0 \leqslant \mathrm{i} \leqslant k-1$, and all edges are distinct. $P(v_0, v_k)$, which is $\{v_0, v_1, \ldots, v_{k-1}, v_k\}$, is called a path. A unicast is defined as an ordered quadruple $(u, v, P(u, v), t)$, where u, v are the source and destination nodes, $P(u, v)$ is a directed path from u to v, and t is the positive integer communication step in which the unicast begins. In this paper, a broadcast can be divided into many unicasts. This kind of broadcast is called unicast-based broadcast, as previous described.[8,11] These unicasts are utilized to construct an ordered scheduling tree. Every edge which is a unicast in the tree is assigned a step number which represents a routing order in a broadcast as shown in Fig. 3.

2.1.3. Point to Point Message Transfer in an HNOW

When worms are routed on such HNOWS, the communication latency is the sum of cost components. These cost components are the send overhead, transmission cost and the receive overhead. These components consist of a message size dependent factor and a constant factor. Thus, the one-way time for a single point-to-point message transfer between two adjacent workstations,[2] can be expressed as

$$T_{ptp} = O_{\text{send}} + O_{\text{trans}} + O_{\text{receive}}, \tag{1}$$

$$O_{\text{send}} = S_c^{\text{send}} + S_m^{\text{sender}} \times m, \tag{2}$$

$$O_{\text{trans}} = X_c + X_m \times m, \tag{3}$$

$$O_{\text{receive}} = R_c^{\text{receive}} + R_m^{\text{receive}} \times m, \tag{4}$$

where m is the message size, S_m the sending latency per byte and S_c is the initiation of sending. X_c is the initiation of transmitting message and X_m is the transmitting latency per byte. In addition R_c is the latency of initiation of receiving message and R_m is the receiving latency per byte. For example, a Pentium Pro 200 MHz PC with 128 MB memory and 16 KB/256 KB L1/L2 cache reveals the speed parameters S_c, S_m, R_c, and R_m are 90, 0.18, 140, and 0.08 μs. A Pentium II 300 MHz PC with 128 MB memory and 32 KB/256 KB L1/L2 cache shows that S_c, S_m, R_c, and R_m are 60, 0.05, 110, and 0.03 μs.[2] Tables I–IV, they illustrate these speed parameters for simulation.

2.1.4. Wormhole Routed Model in an HNOW

The wormhole switching technology[4,12] has low communication latency and is applied in Myrient,[3] Server-Net,[5] and Auto-net.[13] In a wormhole routed heterogeneous network of workstations, the system model is a little different.[12] Suppose a path from nodes v_1 to v_2 has a path length of d, and this unicast will transmit m bytes from v_1 to v_2, then this model can be demonstrated as

$$T_{ptp} = O_{\text{send}} + O_{\text{trans}} + O_{\text{receive}}, \tag{5}$$

$$O_{\text{send}} = S_c^{\text{send}} + S_m^{\text{sender}} \times m, \tag{6}$$

$$O_{\text{trans}} = X_c + X_m \times (m + d), \tag{7}$$

$$O_{\text{receive}} = R_c^{\text{receive}} + R_m^{\text{receive}} \times m. \tag{8}$$

These modified formulas model the latency of transmitting a m-byte message from a source node to a destination node in a contention-free situation. In a wormhole routing, if the path length is d, the speed of any switch is $1\mu s$ and m bytes are transmitted, then the latency is $(m + d)\mu$s.

2.1.5. Fully Connected and Partially Connected HNOW

In an HNOW, a switch can have no workstation to be connected. Therefore, this paper considers two types of HNOW. In One, every switch is connected to one or more than one workstation. Such a HNOW is called *a Fully Connected HNOW*. The other HNOW in which some switches are connected to no workstation. This HNOW is called *a Partially Connected HNOW*. A spanning tree is generated from a Fully Connected HNOW, this spanning tree is called *a Fully Connected Spanning Tree*, while a spanning tree is obtained from a partially connected HNOW is called *a Partially Connected Spanning Tree*. Our proposed schemes are applied to both fully connected and partially connected HNOWs.

2.2. Related Works

2.2.1. Up*/Down* Routing

In a NOW such as Myrinet or Autonet, the up*/down* routing is most frequently used as base routing.[6,8] This paper adopts the unicast-based routing as a basic step,[8,11] so this scheme is reviewed below. First, a rooted breadth-first search (BFS) spanning tree is constructed. For each communication step above, the paper suggests two ways to route worms.[8]

First, an edge (u, v) is classified as an *up link* if the level of u exceeds the level of v, or if u and v are at the same level and the postorder number of u exceeds that of v. A link which is not an up link is a *down link*. Based on such classification, a *strict up-first* routing is one in which each path must go through up tree links before down tree links (only links of the BFS tree are called *tree links*), the rest are cross edges. An alternative and more flexible way is *relaxed up-first* routing, where both tree and cross edges can be used.[8] However, each path must still pass through up links before down links.

2.2.2. The Postorder Recursive Doubling Algorithm

Libeskind-Hadas *et al.*[8] presented a tree-based broadcasting algorithm. This routing algorithm which uses the minimal number of steps and offers the depth-contention contention-free routing is called the postorder recursive doubling algorithm.[8] When a scheduling tree is constructed by this algorithm, it can not perform well and provide an optimal routing in an HNOW because fastest nodes can not be easily arranged in the internal nodes of the scheduling tree. This scheduling tree is used for the unicast-based broadcast. Let M denote a multicast request with a source s and d destination nodes. The set $M - s$ is partitioned into two sets, M_1 and M_2 such that M_1 contains the destination in M whose postorder ID's are greater than the postorder ID of s and M_2 contains the destinations in M whose postorder ID's are less than the postorder ID of s. A list L is constructed in which s is the first element, followed by the elements in M_1 sorted by increasing postorder ID's, followed by the elements in M_2 sorted by increasing postorder ID's. Let $L = v_0, v_1, v_2, \ldots, v_d$ where $s = v_0$. In the first communication step, v_0 sends messages to the node at the midpoint of the list, $V_{\lceil (d+1)/2 \rceil}$, using up*/down* path with cross-edges. Now, v_0 is response for delivering the message to the first half of the nodes in L. Each of these two nodes recursively applies the algorithm by sending the message to the midpoint of each of their respective sublists using up*/down* routing path with cross-edges. This process continues until after $\lceil \log_2(d+1) \rceil$ iterations that every destination has received the message. This algorithm is referred to henceforth as the postorder recursive doubling algorithm. The list $S \cup M_1 \cup M_2$ is called *a rotation list*.

For example, in Fig. 1(c) the postorder list is {8, 9, 7, 2, 1, 3, 5, 6, 4, 10, 11, 12, 16, 15, 14, 13}. Suppose the source is 10, then M_1 is {11, 12, 16, 15, 14, 13}, and M_2 is {8, 9, 7, 2, 1, 3, 5, 6, 4}. Let L be {10, 11, 12, 16, 15, 14, 13, 8, 9, 7, 2, 1, 3, 5, 6, 4}. The list L is utilized to construct a scheduling tree in which the source is 10. L is also called a rotation list.

3. PREVIOUS BROADCASTING STRATEGIES FOR AN HNOW

3.1. Exchange Broadcasting Scheme, EBS

3.1.1. Basic Concept

In a switch-based HNOW, a spanning tree T is built first, and the rotation list headed by a source s from T is obtained. From this postorder scheme, a scheduling tree which is a binomial tree or similar to a binomial tree is constructed. The FNF is basic principle to decrease routing latency in an HNOW. The basic idea behind EBS[9] is that the fastest nodes in every switch must move to the internal nodes of the upper levels in the scheduling tree to decrease the latency as much as possible. This policy can be explained as follows.

Suppose in a scheduling tree z is the parent of x; y is a child of x, and x and y are in the same switch, thus this routing order is $z \rightarrow x \rightarrow y$. Since x and y are in the same switch and y is faster than x, the routing order can be changed into $z \rightarrow y \rightarrow x$ in the scheduling tree. This change can not cause contention because x and y are routed in the same switch. The structure of the original scheduling tree can not be changed, and the minimal routing steps like the scheduling tree constructed by the postorder recursive doubling algorithm are maintained. Figure 2(a) obtained from Fig. 1(c) is a rotation list led by source 8. Figure 2(b) shows that nodes 9 and 7 are in the same switch, and can be exchanged for each other to increase performance. Similarly, node y continues this same process until it is a leaf in the scheduling tree constructed by EBS.

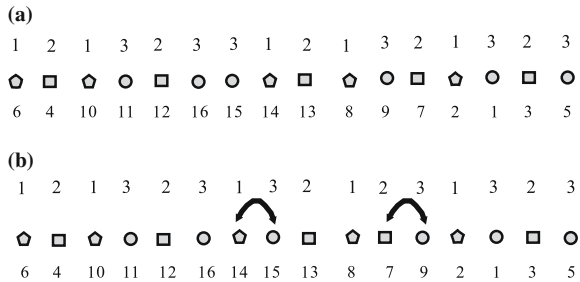

Fig. 2. The rotation list led by the source 6 with distinct speed types is generated from the postorder list in Fig. 1(c). (a) This list includes various speed types. The numbers at the top of the line represent speed types. This HNOW has three speed types. (b) Nodes 9 and 7 are in the same switch in the postorder list, they can be exchanged for building a scheduling tree. Nodes 14 and 15 are in the same situation.

3.1.2. EBS Algorithm

Algorithm 1 shows that x and y represent node pointers in a scheduling tree. The EBS uses a binomial-like scheduling tree for routing. The EBS is a contention-free routing and has $O(n \log(n))$ time complexity.[9]

Algorithm 1

(1) Suppose T is a spanning tree of an HNOW, and M is a rotation list led by a source S. Construct a scheduling tree from M using the postorder recursive doubling algorithm.
(2) Let L be the postorder list from the scheduling tree.
(3) **For** each node x obtained sequentially from L, **Do**
  While x is not a leaf in the scheduling tree **Do**
    (i) Find a node y in the children of x, in which y has faster speed than that of x, and x and y are in the same switch.
    (ii) Exchange x and y.
    (iii) $x = y$.
    (iv) x becomes a current processing node.
  end-while
end for

Step 3(i) shows that any internal node x of the scheduling tree must exchange with its child y whose speed is faster than that of x, if node y and node x are in the same switch. Steps 3(iii) and (iv) explain that node x continues the same swapping procedure until x is a leaf in the scheduling tree. The while loop will be terminated when x is a leaf in the scheduling tree.

3.1.3. EBS Example

For example, Fig. 2 is rotation list led by the node 6. Figure 3(a) shows an original scheduling tree constructed by EBS. The number in the middle of square is the speed type. The rotation list headed by the source 6 from Fig. 1(c) is {6, 4, 10, 11, 12, 16, 15, 14, 13, 8, 9, 7, 2, 1, 3, 5}. Their speed types are {1, 2, 1, 3, 2, 3, 3, 1, 2, 1, 3, 2, 1, 3, 2, 3}. A smaller number corresponds to a faster speed; speed type 1 is faster than speed type 2. The EBS is a contention-free routing and also has minimal routing steps. Suppose an HNOW has n workstation, then it takes $\lceil \log_2(n) \rceil$ steps to complete a broadcast from a source s.[9]

In Fig. 3(b), because nodes 9 and 7 are in the same switch, and the speed of node 9 is greater than that of node 7, these two nodes can change

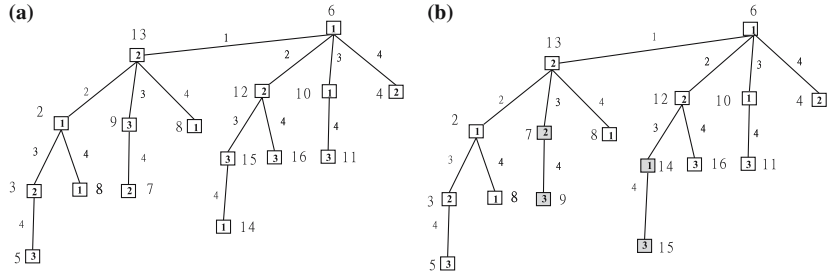

Fig. 3. (a) is an original scheduling tree. (b) is the scheduling tree rearranged by EBS. Nodes 7 and 9 are exchanged; they are in the same switch. The speed of node 7 is faster than that of node 9. Nodes 14 and 15 are in the same situation. The number in the middle of square is the speed type.

places with each other. Similarly, nodes 14 and 15 can be exchanged to improve the performance. Rearrangement yields a new scheduling tree and the new unicast-based scheduling tree improves performance.

3.1.4. The Aim of the Sorted Rotation List

Let the source be node 8. The rotation list that is headed by node 8 is {8, 9, 7, 2, 1, 3, 5, 6, 4, 10, 11, 12, 16, 15, 14, 13} in Fig. 1(c). If this rotation list is sorted on all workstations within each switch in order of ascending speed type, we shall obtain the newly sorted list L_1, which is {8, 7, 9, 2, 3, 1, 6, 4, 5, 10, 12, 11, 14, 13, 15, 16}. Their respective speed types are {1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 3}. The aim of the sorted list is to distribute the equal number of speed types into two sub-networks. Such a sorting can decrease the routing latency using TWO-VBBS and TWO-EBS as shown in later section. Figure 4 shows that it is a scheduling tree obtained from the list L_1.

3.2. Variable Bucket-Based Broadcasting Scheme, VBBS

3.2.1. Definition of a Bucket

This section uses the idea of buckets to design a new algorithm.[9] The postorder list is obtained from a spanning tree T, and the rotation list M led by a source s, can be divided by all the fastest nodes. The idea of *a bucket* is naturally generated. The bucket, which is always located in some local area of an HNOW, is used to multicast from the head. When broadcasting, the messages are first sent to the heads of these buckets, then many buckets can be routed concurrently from their heads. Fortunately, this broadcasting by using these buckets are all contention-free, if they are

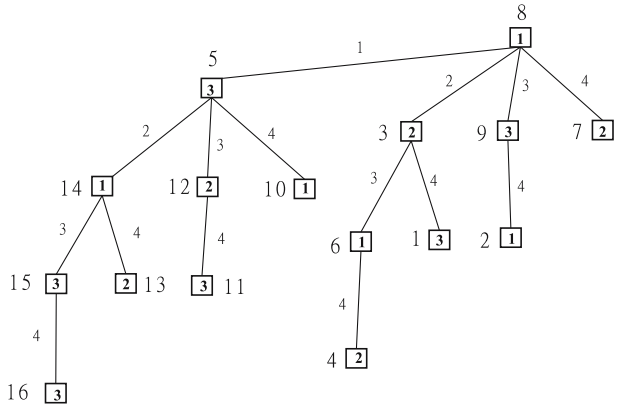

Fig. 4. It is the scheduling tree obtained from the list $L\{8, 7, 9, 2, 3, 1, 6, 4, 5, 10, 12, 11, 14, 13, 15, 16\}$. L is sorted from the original rotation list on all workstations in each switch, respectively, according to speed type.

routed from heads. This feature is very useful and efficient for broadcasting, and can reduce contention. The VBBS uses buckets for routing. The definition of a bucket is defined as follows.

Definition 1. Suppose an HNOW $G(V, E)$, T is a spanning tree with n nodes of G and a known source s is known. M is a rotation list led by the source s. A bucket is represented as B which is a subset of the rotation list M. All elements in B are followed by the continual ascending order in M. If a bucket has only one node, then it is called *a single bucket*.

3.2.2. The VBBS Algorithm

Different workstations can have distinct speed types. The fastest nodes in M are divided into many buckets. Each of these buckets is headed by the fastest node at the head. If two buckets from its head (source) are routed by the postorder recursive doubling, they should be contention-free. A proper bucket length $l = \lceil \log_2(\text{Length}) \rceil$, where Length is chosen as the length of the biggest bucket in M, is to preserve the minimal step as much as possible. The next step is to combine all nearby small buckets whose length is less than or equal to 2^l. All fastest heads of these new buckets are used to construct a scheduling tree T_{head} by using the postorder recursive doubling. For every bucket, EBS is applied to construct another scheduling tree T_x. The tree T_x is connected to T_{head} by the

root of T_x. The root is also a node in T_{head}. The VBBS, which uses the idea of bucket for broadcasting, is as follows.

Algorithm 2

(1) A source S is known. Obtain a rotation list M, headed by the source S from a spanning tree T with n nodes. M is a broadcast request. Assign M to B.

(2) Find all the fastest nodes in B and use them to divide B into k buckets $B_1, B_2, \ldots, B_k$. Each bucket is headed by a fastest node. Let $l = \lceil \log_2(\text{Length}) \rceil$, where Length is the length of the longest bucket.

(3)
(i) $h = 1$
(ii) $i = 1$
(iii) $B_{\text{buffer}} = \phi$
(iv) $d = \lceil \log_2(k) \rceil$
(v) $z = \lceil \log_2(|B|) \rceil$
(vi) **if** $(d + l > z)$
For each bucket B_i **Do**, $i \leqslant k$
if (length$[B_i \cup B_{\text{buffer}}] \leqslant 2^s$) **then**
$B_{\text{buffer}} = B_{\text{buffer}} \cup B_i$
else
$B_h = B_{\text{buffer}}$
$B_{\text{buffer}} = B_i$
$h = h + 1$
endif
end for
$B_h = B_{\text{buffer}}$
endif

(4) Obtain a list L_{head} of the heads of these new buckets.

(5) Construct a scheduling tree T_{head} from L_{head} using postorder recursive doubling.

(6) **For** every new bucket **Do**
Construct a scheduling tree T_x using EBS and connect it to T_{head} by the head of this new bucket.
end for

Step 3 shows that if $(d + s) > z$ then the nearby small buckets are combined into a large bucket. The aim of compression is to decrease routing steps. Step 5 constructs the scheduling tree composed of all heads in new compressed buckets. Step 6 builds its own scheduling tree in every new bucket and this scheduling sub-tree is connected to the scheduling tree

constituted by heads. If $\lceil \log_2(\text{Length}) \rceil$ is equal to $\lceil \log_2(n) \rceil$, it is a special case. The original bucket B is divided into only two buckets. The number of steps in the scheduling tree is $\lceil \log_2(|B|) \rceil + 1$.

3.2.3. VBBS Example

For example, Fig. 5 shows a scheduling tree using VBBS. The sorted rotation list headed by source 8 is {8, 7, 9, 2, 3, 1, 6, 4, 5, 10, 12, 11, 14, 13, 15, 16}. Their speed types are {1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 3}. The VBBS first uses the fastest node with speed type 1 to divide the rotation list into five buckets, which are $B_1 = \{8, 7, 9\}$, $B_2 = \{2, 3, 1, 5\}$, $B_3 = \{6, 4\}$, $B_4 = \{10, 12, 11\}$, and $B_5 = \{14, 13, 15, 16\}$. Now, the largest bucket B_5 whose length is 4 is found. The length of 4 is chosen as the standard length for compressing buckets. These buckets are led by five fastest nodes, which are nodes 8, 2, 6, 10, and 14. Third, nodes 8, 2, 6, 10, and 14 are used to build the first level scheduling tree. Then the second level spanning tree is constructed from the above five new buckets. Figure 5 shows

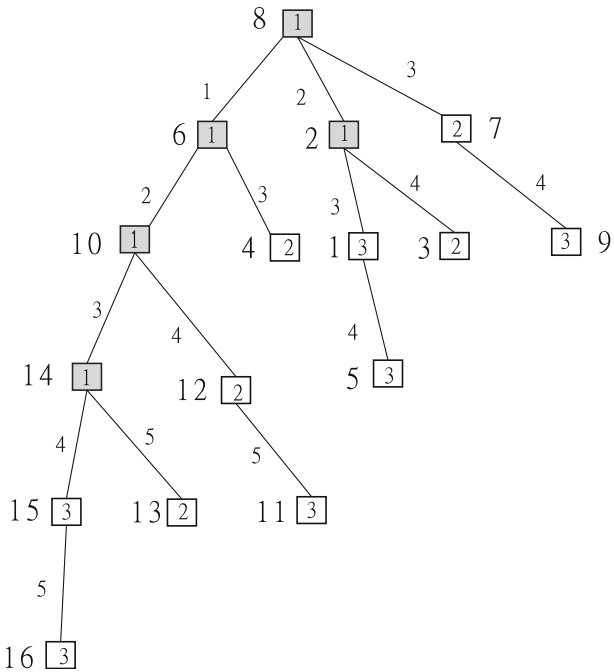

Fig. 5. The scheduling tree is constructed by VBBS.

a scheduling tree constructed by VBBS. The VBBS performs better than EBS. The VBBS requires $\lceil \log_2(n) \rceil + 1$ steps to broadcast.[9]

3.2.4. The VBBSWF Algorithm

VBBSWF is an extension of VBBS. The VBBSWF utilizes the concept of variable bucket led by the fastest node or some fast nodes to perform broadcast. If a bucket, headed by some fastest nodes, means that the head in a bucket may have continual fastest nodes or some fast nodes, and of course, this bucket is led by some fast nodes. All nodes are supposed to have m multiple speed types. The VBBSWF separates these nodes into new buckets and combines all nearby small buckets into a longest bucket by recursion until every bucket contains the same speed types except the head or the process has been performed around $m - 1$ times. Then, EBS is applied to the final bucket.[9]

4. TWO SUB-NETWORKS BROADCASTING ALGORITHM

This section considers that the proposed scheme broadcasts on a Fully Connected HNOW using two sub-networks. Fist, we introduce the *Switch-Reduced HNOW*, which is used to divide the original HNOW into two sub-networks, when broadcasting. The partitioning algorithm used here is called the *Odd-Even Partitioning Algorithm*.

4.1. Switch-Reduced HNOW

Switches are not workstations in an HNOW. We define a *Switch-Reduced HNOW* as a rearranged HNOW, in which every switch should be replaced by the last workstation connected to itself. These two nodes, switch and the last workstation, are merged into a single node as illustrated in Fig. 6(b). The reason to select the last workstation in an HNOW is that we can keep the postorder sequence similar to the original HNOW. The merged graph is called *Switch-Reduced Graph*. The switched-reduced HNOW is used to construct two sub-networks and DCNs for exchanging messages.

4.2. A General Broadcasting Model in Two Partitioned Networks

In order to efficiently broadcast a message, all workstations should be sorted according to ascending speed types, the aim of sorting is to distribute as equal as possible the number of speed types into two sub-networks. The two-partition scheme[10,15] works in three phases. First, two *Data Distributed Networks* (DDN) are constructed. These two DDN are

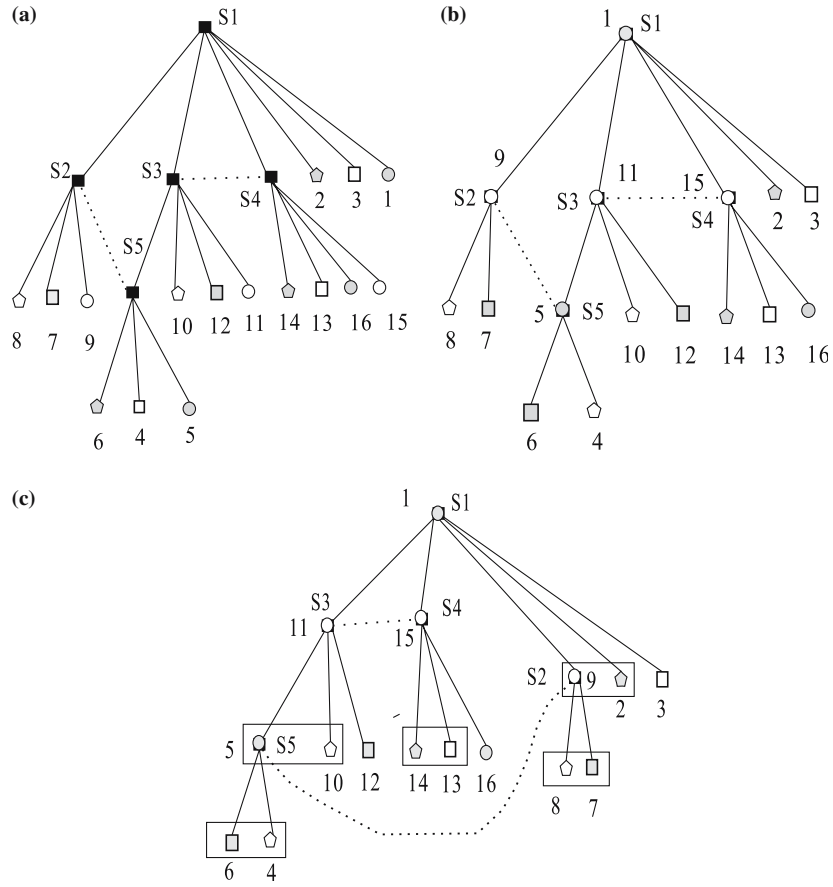

Fig. 6. (a) The spanning tree is sorted on all workstations in each switch in Fig. 1 by ascending speed types. (b) The switch-reduced HNOW, the switch node is replaced by the last workstation in each switch. (c) The rearranged switch-reduced HNOW, the spanning tree is reordered only by putting the even number of sub-trees on the left and the odd numnber of sub-trees on the right by Algorithm 3, and preserves even-numbered of sub-trees and odd-numbered of sub-trees sequences as shown in (a). This rearranged switch-reduced HNOW makes any DCN require one step to exchange message.

simply obtained from two-subnetworks divided by Odd–Even partitioning algorithm. Then, the broadcast message is evenly divided into two sub-messages, each being sent to one representative node called the source in a DDN. Second, each sub-message is independently distributed in its DDN. Finally, through a sub-message combination step by Data Collecting Networks (DCNs), each node will obtain the whole broadcast message.

The nodes in a DCN are in distinct DDNs. They are used to exchange messages in different DDNs. Given any network G, a general broadcasting model is here proposed. First, the spanning tree must be constructed by BFS. The node of maximal degree is chosen as the root of the spanning tree. Then two kinds of sub-networks DDN and DCN are constructed.[15] Any DCN contains nodes of which one belongs to a DDN, and DCNs are used to exchange message. The network is divided into two sub-networks, which are DDN_0, DDN_1, similarly k $DCNs$ which are $DCN_0, \ldots, DCN_{k-1}$ are also constructed. These networks have the following properties in proposed model. Furthermore, channels means that all edges in a spanning tree, which is used for routing.

P1: DDN_0 and DDN_1 are two data-distributed networks. Let V_1 represent all nodes in DDN_0. Let V_2 represent all nodes in DDN_1, C_1 is the set of all channels in DDN_0, C_2 is the set of all channels in DDN_1, $V_1 \cap V_2 = \phi$, $C_1 \cap C_2 = \phi$ or $C_1 \cap C_2 \neq \phi$. If $C_1 \cap C_2 = \phi$, they are independent sub-networks.

P2: $DCN_0, \ldots, DCN_{k-1}$ are divided into two sets DC_1 and DC_2. The DC_1 and DC_2 are the set whose element is one of DCN_i, $0 \leqslant i \leqslant k-1$. If $DCN_i \in DC_1$, it takes one step to exchange messages. If $DCN_i \in DC_2$, it requires two steps to exchange messages.

P3: DDN_i and DCN_j intersect one node, for all $0 \leqslant i \leqslant 1$, $0 \leqslant j \leqslant k-1$. Theses channels are not all mutually independent, sometimes they will be overlapped when two sub-networks are routed in an HNOW. Most sub-networks are dependent.

4.2.1. Three Phases of the General Broadcasting Scheme

In an HNOW, when messages are broadcast on a NOW, if the source S_1 is known, another source S_2 must be chosen from different sub-network. The source S_2 must be selected carefully to avoid contention. Our broadcasting schemes work in three phases. Phases are synchronous and they should be executed consecutively. The following are three phases of the general broadcasting scheme.

Phase 1: Use the partitioning algorithm to divide the spanning tree into two partitions (sub-networks), and then two data distributed networks DDN_0, DDN_1 are constructed. The DCN_s are also constructed by DCN constructive algorithm. A source in one partition is known as S_1, and another source S_2 is selected from the other partition. The message is divided into two parts, one of which is sent from the source S_1 to another source S_2. Then, VBBS or EBS[9] is used to establish two ordered scheduling trees.

Phase 2: Broadcast one half of the message from the source S_1 to all nodes in one sub-network and simultaneously from the source S_2 broadcast the other half of the message from the source S_2 to all nodes in the other sub-network.

Phase 3: On each DCN, all nodes exchange messages until the sub-messages are concurrently recollected and combined into the original message in each node.

4.3. How to Divide an HNOW into Two Sub-Networks

This section presents the *Odd–Even Partition Algorithm*. Odd–even partition is a simple scheme that uses the odd or even IDs to divide an HNOW into two partitions or sub-networks. First, a postorder list must be constructed in a spanning tree in a *Switch-Reduced HNOW*. One sub-network (DDN) contains all workstations with even postorder in a spanning tree. The other sub-network (DDN) contains all workstations with odd postorder. Two DDNs are routed concurrently and use DCNs to exchange messages by Algorithm 3. Figure 6(a–c) shows that the even sub-network and the odd sub-network are determined naturally according to even nodes and odd nodes. Two sub-networks are {8, 9, 4, 10, 11, 13, 15, 3} and {7, 6, 5, 12, 14, 16, 2, 1} in Fig. 6.

4.4. DCN Construction Algorithm

When two sub-networks complete their routing concurrently, contention is not so much as we predicted. Two sub-networks use DCNs to exchange messages. Algorithm 3 shows how to build DCNs using a *Switch-Reduced HNOW*. Initially, Algorithm 3 requires counting the number of sub-trees using the postorder of a spanning tree at every level. Notably, the spanning tree should be rearranged by putting the sub-trees with old number of nodes on the left and the sub-trees with the odd number of nodes on the right. This rearrangement preserve the sub-tree sequence as the original spanning tree. This new spanning tree is called a *Rearranged Switch-Reduced HNOW* as shown in Fig. 6(c). It can be easily used to construct DCNs, not for routing. To count the number of any sub-tree, the code can be easily written by a recursive procedure. Furthermore, this algorithm also traverses the postorder in a spanning tree. When a node is traversed, then it is marked. The marked node can not be used again to combine with un-marked node as a DCN. The DCNs that are constructed by Algorithm 3 take one or two steps to exchange messages. Algorithm 3 demonstrates how to construct $DCNs$.

Algorithm 3

(1) **For** $i = 1$ to h, where h is the number of levels in T, **Do**
For each node in level i, if it has sub-tree then

(i) Count the number of nodes in its sub-tree.
(ii) Reorder all sub-trees with the even number of nodes on the left, which preserve the sub-tree sequence of the original spanning tree.
(iii) Reorder all sub-trees with the odd number of nodes on the right, which also preserve the sub-tree sequence of the original spanning tree.

endfor
endfor
(2) Let $P = R$, where R is the root of the spanning tree.
(3) DCN-construction(P)

(i) If P has children then let $y_1, y_2, \ldots, y_k$ be the children of node P.
(ii) **If** (P is a leaf) **then**
If (P is unmarked) **then**

If (P has a right sibling Sib and they belong to distinct partitions) **then** group Sib and P as a DCN and mark P and Sib.
else if (P and its parent belongs to distinct partitions) and (its parent is unmarked) **then** group P and its parent as a DCN, mark P and its parent.
endif

endif
Return.
endif
(iii) **If** (P is not a leaf)**then**
For each node y_i, $1 \leqslant i \leqslant k$, **Do**
Let $P = y_i$, **DCN-construction(P)**.
endfor
endif
(iv) **if** (P is unmarked) **then**
If (P is the root) **then** group P and y_k as a DCN
else if (P has a right sibling Sib and they belong to distinct partitions) **then** group Sib and P as a DCN and mark Sib and P.

else group P and its parent as a DCN, and mark P and its parent.
endif

Step 3(ii) demonstrates how to construct a DCN when the node is a leaf. The leaf with its siblings or its parent can be grouped as a DCN. Step 3(iii) is a recursive call. Step 3(iv) shows how to construct a DCN in the internal node of a spanning tree. If $k \neq 0$ then node P is not a leaf. The advantage of rearrangement is to reduce the number of steps to exchange message. After rearrangement, any node together with its sibling are in distinct partitions, and the node also will be in different partitions from its parent. The rearranged switch-reduced graph is used only to construct DCNs, not for routing.

4.5. Example of the DCN Construction Algorithm

We use *Switch-Reduced HNOW* to construct DCNs and obtain two subnetworks. When odd–even partition algorithm is used to broadcast, all workstations in each switch will be individually sorted by ascending speed types. In Fig. 1(c), all workstations in each switch are individually sorted by ascending speed type. The sorted list is {8, 7, 9, 6, 4, 5, 10, 12, 11, 14, 13, 16, 15, 2, 3, 1} in order of postorder in Fig. 6. The corresponding speed types are {1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 3, 1, 2, 3}. This sorting can distribute the equal number of speed types into two sub-networks and make two distinct routing times on two respective sub-networks be the same as close as possible. Two sub-networks are {8, 9, 4, 10, 11, 13, 15, 3} and {7, 6, 5, 12, 14, 16, 2, 1} by odd–even partitioning. We select the two nodes 8 and 6 as two sources in different sub-networks. Then, EBS and VBBS are applied to these two partitions. Figure 7 shows two scheduling trees are built by EBS. Figure 8 shows two scheduling trees are constructed by VBBS.

Figure 6(a) shows that all workstations are sorted in each switch. Figure 6(b) shows that each switch is replaced with the last workstation connected with this switch. Figure 6(c) the spanning tree is rearranged by putting the even numbered of sub-trees on the left and the odd numbered sub-tree on the right. The set of DCNs is {(6, 4), (5, 10), (12, 11), (14, 13), (16, 15), (8, 7), (9, 2), (3, 1)}. The maximal swap latency is the DCN (16,15). The speed type of node 16 is 3, and the speed type of node 15 is 3. The speed parameters are taken from Table IV. The latency of exchanging messages by nodes 16 and 15 is $120 + 1024 \times 1.6 + 1024 \times 0.125 + 170 + 1024 \times 1.28 + 16 = 3382$. The latency of transmitting through a switch is 0.125 μs.

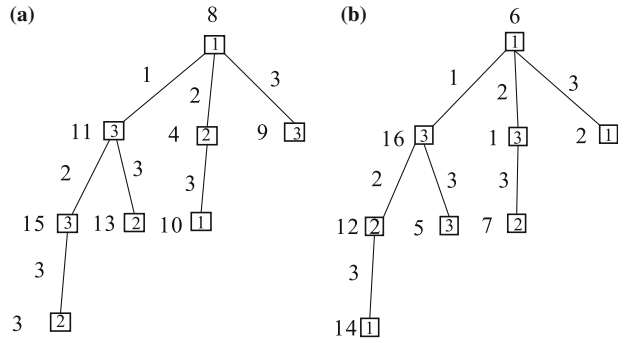

Fig. 7. Two scheduling trees in distinct two partitions are constructed by EBS.

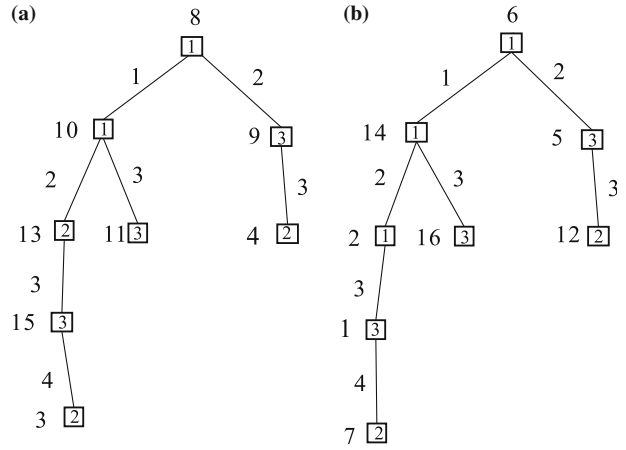

Fig. 8. Two scheduling trees in distinct two partitions are constructed by VBBS.

4.6. Effect and Characteristics of Sub-Trees Rearrangement

Algorithm 3 shows that the rearranged tree is not a real spanning tree, but it is used to construct DCNs. Step 1(i) of Algorithm 3 shows that this special kind of sorting only by putting the even-numbered of sub-trees on the left and the sub-trees with the old number of nodes on the right, and maintains the same sub-trees sequence as the original spanning tree. After rearrangement, the spanning tree is changed. Fortunately, however, we only uses the new tree for DCNs construction, not for routing. The newly rearranged tree has the following properties.

- When DCNs are used to exchange messages, they can not cause contention.
- All the number of nodes in odd-numbered sub-trees of the original spanning tree are arranged in such a sequence by having the odd ID follow even ID or vice versa. After rearrangement, the sequence of odd-numbered sub-trees in a rearranged switch-reduced HNOW also remain unchanged by Algorithm 3.

4.6.1. Why DCNs Have No Contention in the Time of Exchanging Messages

Although this rearrangement changes positions of workstations at the same level, exchanging messages via DCNs causes no contention. This can be explained as follows:

- Two workstations in a DCN are connected by a single switch, for instance, Fig. 6(c) shows that nodes 14, and 13 are connected to a single switch.
- Two workstations in a DCN are connected by two linked switches, for instance, Fig. 6(c) shows that nodes 10, and 5 are connected to two linked switches.
- Two workstations in a DCN are connected by three linked switches, for example, Fig. 6(c) shows that nodes 15, and 11 can be connected to three linked switches.

Of course, when messages are transmitted through switches, DCNs have no contention in the time of exchanging messages.

4.6.2. Postorder ID in Roots of Odd-Numbered of Sub-Trees

In any spanning tree, the following lemma shows that the postorder IDs of odd-numbered sub-trees are arranged by having the even ID follow odd ID or vice versa. This characteristic benefits DCN construction as in Fig. 6.

Lemma 1. Given a spanning tree T, the root R of T, the sub-trees of the root R with even-numbered nodes in T are $T_{e_1}, \ldots, T_{e_z}$, the sub-trees with odd-numbered nodes are $T_{o_1}, \ldots, T_{o_h}$, and these sub-trees preserve the original sub-tree sequence in T. Suppose that the roots of odd-numbered sub-trees are $r_{o_i}, r_{o_{i+1}}, \ldots,$ and r_{o_h}, then the postorder ID of the roots in these odd-numbered sub-trees are shown by having the even ID follow the odd ID or vise versa in T. Similarly, for any sub-tree having children in T, this lemma also holds.

Proof. For any two odd-numbered sub-trees, T_{o_i} and $T_{o_{i+1}}$, where $1 \leqslant i \leqslant h-1$, their roots are r_{o_i} and $r_{o_{i+1}}$, and they contains zero or at least one even-numbered sub-trees between them. Suppose that the post-order ID of r_{o_i} is even, then the ID of $r_{o_{i+1}}$ is odd since we add the even number obtained from even-numbered nodes of sub-trees between T_{o_i} and $T_{o_{i+1}}$ and the odd-numbered nodes of sub-tree $T_{o_{i+1}}$ to the ID of $r_{o_{i+1}}$. Thus the ID of $r_{o_{i+1}}$ is odd. Similarly, the ID of the next sub-tree will be even. We conclude that the ID of roots of odd-numbered sub-trees are in order of odd–even or even–odd periodically. Clearly, any sub-tree in T has the same property. ▮

4.7. Number of Steps to Exchange Messages in DCNs

The below, two theorems show DCNs require one step or two steps to exchange messages in DCNs in an HNOW. Algorithm 3 uses the postorder sequence to construct DCNs since it traverses the postorder of a spanning tree. To understand the following theorems, we refer to Fig. 6(b) and (c).

Algorithm 3 puts the odd-numbered sub-trees on the left and preserves their sequence as the original spanning tree. Therefore, they also have the same property like Lemma 1 in odd-numbered sub-trees.

Theorem 1. Given a spanning tree T, suppose that the number of nodes m in T is even, then Algorithm 3 can construct $m/2$ DCNs. Messages can be exchanged in a single step in these DCNs.

Proof.

(1) Suppose that the root of T is r and the number of nodes m in T is 2. Clearly, $DCN = \{1, 2\}$. Thus, a message can be exchanged in one step. Now we prove it by induction. Suppose $m = 2k; k \in N$ holds, we want to prove that $m = 2(k+1)$ is true.

(2) Suppose that the sub-trees of T are $T_{e_1}, \ldots, T_{e_z}, T_{o_1}, \ldots, T_{o_h}$ and the sub-trees are rearranged by putting the even numbers of sub-trees on the left and the odd numbers of sub-trees on the right, then the number of nodes in these sub-trees are $e_1, e_2, \ldots, e_z, o_1, \ldots, o_{h-1}$, and o_h, where $h \in$ odd, $e_i \in$ even, $0 \leqslant i \leqslant z, o_i \in$ odd, and $1 \leqslant j \leqslant h$. Furthermore, their roots are $r_{e_1}, \ldots,$ and r_{e_z} in those even number of sub-trees, and $r_{o_1}, \ldots,$ and r_{o_h} in odd number of sub-trees.

(3) For each sub-tree with an even number of nodes n, $n \leqslant 2m$, there can be constructed $n/2$ $DCNs$. Consider two adjacent sub-trees

whose roots are r_{o_i}, and r_{o_j}, $1 \leqslant i, j \leqslant h$. If the postorder of r_{o_i} is even, then the postorder of r_{o_j} is odd, and vice versa, as shown in Lemma 1 and in the rearranged switch-reduced HNOW. Therefore, r_{o_1} combines with r_{o_2} as a DCN, and we continue the same grouping for the next two roots until the last two roots $r_{o_{h-2}}$ and $r_{o_{h-1}}$ can generate a DCN. Finally, the rest is the root r and the root of last sub-tree r_{o_h} are grouped as a DCN. All DCNs require one step to exchange a message. ▮

Theorem 2. Given a spanning tree T, suppose the number of nodes m in T is odd, then $\lceil m/2 \rceil$ *DCNs* can be constructed by algorithm 3. Two steps are required to exchange messages in these DCNs.

Proof

(1) Suppose that the root of T is r, let $m = 2k + 1$, the sub-trees of T are $T_{e_1}, \ldots, T_{e_z}, T_{o_1}, \ldots T_{o_h}$, and the sub-trees are sorted by putting the even numbers of sub-trees on the left and the odd numbers of sub-trees on the right, then T has sub-trees in which the number of nodes are $e_1, e_2, \ldots, e_z, o_1, \ldots, o_{h_1}, \ldots .o_{h-1}$, and o_h, where $h \in$ even, $e_i \in$ even, $0 \leqslant i \leqslant z$, $o_j \in$ odd, and $1 \leqslant j \leqslant h$.

(2) For each sub-tree with an even number of nodes o_h, where $o_h \leqslant 2k$, thus $h/2$ DCNs can be constructed. Consider two adjacent sub-trees whose roots are r_{o_i}, and r_{o_j}, $1 \leqslant i \leqslant h$. If the postorder of r_{o_i} is even then the postorder of r_{o_j} is odd, and vice versa, as shown in Lemma 1 in the rearranged switch-reduced HNOW. Therefore, r_{o_1} can be combined with r_{o_2} as a DCN, we continue the same grouping for the next two roots until the last two roots $r_{o_{h-1}}$ and r_{o_h} are grouped accordingly. Finally, the root r is a single node, which can connect with its child y_k as a DCN, as shown in Algorithm 3. These new $DCNs$ require two steps to exchange messages. The total number of $DCNs$ is $\lceil m/2 \rceil$. ▮

5. BROADCASTING ON AN HNOW USING TWO-VBBS

All workstations in each switch must be sorted individually according to their speed types. Suppose that two-subnetworks are constructed by odd–even partition. TWO-EBS means any of two scheduling trees is constructed by all nodes of its subnetworks using EBS algorithm. Figure 7 shows that two scheduling trees are obtained from two-subnetworks using EBS. Similarly, TWO-VBBS means any of two scheduling tree is also

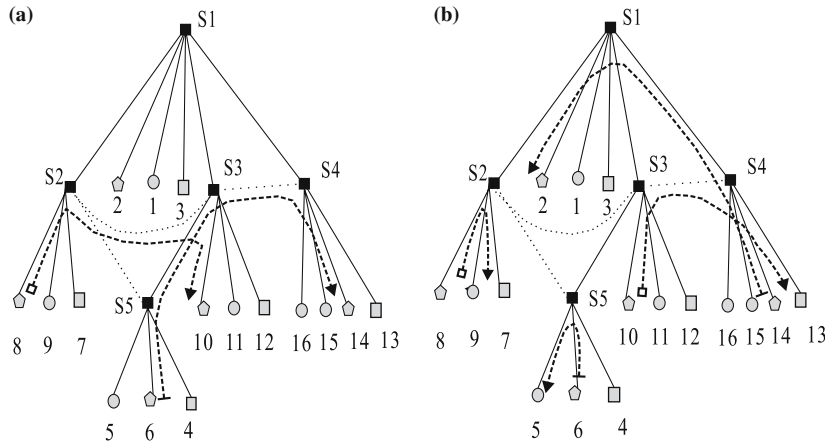

Fig. 9. (a) Step 2 in the scheduling tree of Fig. 8 by TWO-VBBS. (b) Step 3 in the sched uling tree of Fig. 8 by TWO-VBBS.

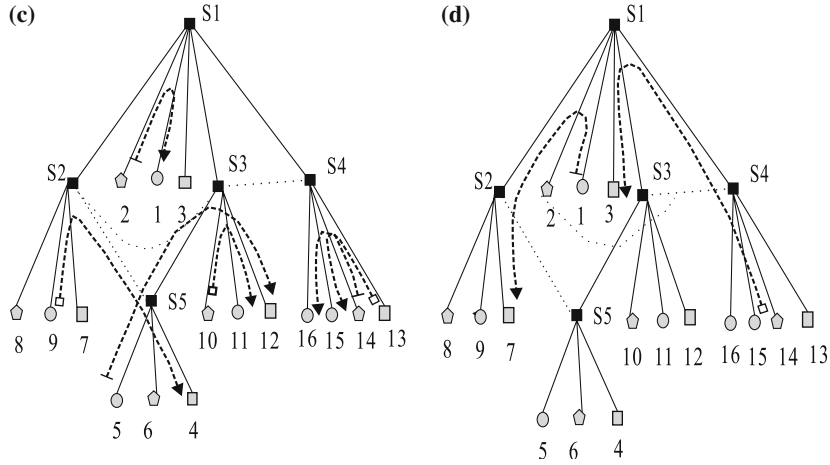

Fig. 10. (c) Step 4 in the scheduling tree of by Fig. 8 TWO-VBBS. (d) Step 5 in the sched uling tree of Fig. 8 by TWO-VBBS.

constructed by all nodes of its subnetworks using VBBS algorithm. Figure 8 shows that two scheduling trees are obtained from two-subnetworks using VBBS. Figures 9 and 10 shows how to use TWO-VBBS to broadcast in an HNOW. This routing is contention-free. TWO-VBBS can cause a contention-free routing.

5.1. An Example of TWO-VBBS Comparing to EBS, VBBS, and TWO-EBS

This section computes the latency of EBS, VBBS, TWO-EBS, and TWO-VBBS. We compare the longest latency of TWO-VBBS, TWO-EBS, EBS, VBBS, and VBBSWF.[9] The first three parameters are taken from Table IV to determine the longest latency in EBS, VBBS, TWO-EBS and TWO-VBBS. The size of flits is 2048.

The longest latency obtained by EBS in Fig. 4 is $(8 \rightarrow 5), (5 \rightarrow 14), (5 \rightarrow 12), (12 \rightarrow 11)$. X_m is $0.125\,\mu$s. X_c is $16\,\mu$s. Table V shows the longest routing latency.

The longest latency obtained by VBBS in Fig. 5 is $(8 \rightarrow 6), (6 \rightarrow 10), (10 \rightarrow 14), (14 \rightarrow 15), (15 \rightarrow 16)$. X_m is $0.125\,\mu$s. Table VI shows the longest routing latency.

The longest latency obtained by TWO-EBS in Fig. 7(b) is $(8 \rightarrow 11), (11 \rightarrow 15), (15 \rightarrow 3)$. X_m is $0.125\,\mu$s. Two sources 8 and 2 are selected. First, the half message of 1024 flits must be transmitted from sources 8 to 6. Then two sub-networks {8, 9, 4, 10, 11, 13, 15, 3} and {6, 5, 12, 14, 16, 2, 1, 7}, are routed concurrently. Two distinct sub-networks exchange messages in one step. Table VII shows the longest routing latency. The DCN between nodes 16

Table V. Computing the Longest Latency by EBS

Type	Source to destination	Path	Latency
1	$8 \rightarrow 5$	$8 \rightarrow s_2 \rightarrow s_5 \rightarrow 5$	3225
2	$5 \rightarrow 14$	$5 \rightarrow s_5 \rightarrow s_3 \rightarrow s_4 \rightarrow 14$	3840
3	$5 \rightarrow 12$	$5 \rightarrow s_5 \rightarrow s_3 \rightarrow 12$	4464
4	$12 \rightarrow 11$	$12 \rightarrow s_3 \rightarrow 11$	3972
Total			15501 μs

Table VI. Computing the Longest Latency by VBBS

Type	Source to destination	Path	Latency
1	$8 \rightarrow 6$	$8 \rightarrow s_2 \rightarrow s_5 \rightarrow 6$	605
2	$6 \rightarrow 10$	$6 \rightarrow s_5 \rightarrow s_3 \rightarrow 10$	605
3	$10 \rightarrow 14$	$6 \rightarrow s_3 \rightarrow s_4 \rightarrow 14$	605
4	$14 \rightarrow 15$	$14 \rightarrow s_4 \rightarrow 15$	3225
5	$15 \rightarrow 16$	$15 \rightarrow s_4 \rightarrow 16$	6460
Total			11500 μs

Table VII. Computing the Longest Latency by TWO-EBS

Type	Source to destination	Path	Latency
1	$8 \rightarrow 6$	$8 \rightarrow s_2 \rightarrow s_5 \rightarrow 6$	396
2	$8 \rightarrow 11$	$8 \rightarrow s_2 \rightarrow s_3 \rightarrow 11$	1736
3	$11 \rightarrow 15$	$11 \rightarrow s_3 \rightarrow s_4 \rightarrow 15$	3382
4	$15 \rightarrow 3$	$15 \rightarrow s_4 \rightarrow s_1 \rightarrow 3$	2370
5	Message swap time		3382
Total			11266 μs

Table VIII. Computing the Longest Latency by TWO-VBBS

Type	Source to destination	Path	Latency
1	$8 \rightarrow 6$	$8 \rightarrow s_2 \rightarrow s_5 \rightarrow 6$	396
2	$8 \rightarrow 10$	$8 \rightarrow s_2 \rightarrow s_3 \rightarrow 10$	396
3	$10 \rightarrow 13$	$10 \rightarrow s_3 \rightarrow s_4 \rightarrow 13$	753
4	$13 \rightarrow 15$	$10 \rightarrow s_4 \rightarrow 15$	2124
5	$15 \rightarrow 3$	$15 \rightarrow s_4 \rightarrow 3$	2370
6	Message swap time		3382
Total			9421 μs

and 15 has the maximal latency. The swap latency is 3382 μs. The latency of transmitting through a switch is 0.125μs per byte. x_c is 16 μs.

The longest latency obtained by TWO-VBBS in Fig. 8 (b) is $(8 \rightarrow 10)$, $(10 \rightarrow 13)$, $(13 \rightarrow 15)$, $(15 \rightarrow 3)$. X_m is 0.125 μs. Two sources are 8 and 6. First, the half message of 1024 flits must be transmitted from sources 8 to 6 through switches. Then, two sub-networks which are {8, 9, 4, 10, 11, 13, 15, 3} and {7, 6, 5, 12, 14, 16, 2, 1} are routed concurrently. Finally, two distinct sub-networks exchange messages in one step. Table VIII shows the longest routing latency. The DCN between nodes 16 and 15 has the maximal latency. The swap latency is 3382 μs.

6. BROADCASTING ON A PARTIALLY CONNECTED HNOW

6.1. Compact Spanning Tree

Compact Spanning Tree is only used to divide an HNOW into two sub-networks and construct DCNs to exchange messages. It is not used for routing since it is not a real spanning tree in an HNOW. *A Compact Span-*

ning Tree is obtained from the real spanning tree by deleting some empty switches.

In an HNOW, not all switches are connected to workstations. As so far presented, TWO-VBBS and TWO-EBS cannot be applied to a partially connected HNOW. Therefore, we propose *a Compact Spanning Tree* to solve the above issue, and define *an empty switch*, which has no workstation to be connected, while *a non-empty switch* has some workstations to be connected. The basic idea is as follows. Suppose that workstations x and y are connected through empty switches, we claim they are also adjacent. Therefore, two nodes are also adjacent after deleting empty switches between them via the postorder sequence. How to construct a compact spanning tree is described as follows:

- Suppose that an empty switch x in internal nodes of the spanning tree is found, then a non-empty switch y is selected with the maximal postorder in its sub-tree, whose root is switch x.
- Move switch y and its sub-tree containing all switches and workstations to this empty switch x.
- All the links of switch x connected to other switches are reserved.
- All empty switches between x and y in the postorder list L should be deleted from the spanning tree.
- If the empty switch is a leaf, i.e. without child switch, remove it from the spanning tree.

6.2. Algorithm to Obtain a Compact Spanning Tree

A fully connected spanning tree can be obtained from a partially connected spanning tree using Algorithm 4.

Algorithm 4

(1) Suppose T is a spanning tree constructed from an HNOW, L contains all switches ordered by the postorder in T.

(2) **For** every switch x in L, **Do**

If x is an empty switch and has a sub-tree including other non-empty switches in T **then**

(i) Find the non-empty switch y having the maximal postorder in the sub-tree of switch x.

(ii) Move y and its own sub-tree with their connected switches and workstations to the switch x and all links that connected to x are left.

(iii) Delete all empty switches in the spanning tree between y and x by the postorder.

else if all the switches in the sub-tree of x are empty **then** delete all the sub-tree.

end for

(3) **For** every switch x in L, **Do**
if x is the empty switch without any child switch in T **then** eliminate this switch.
end for

After Algorithm 4 generates a fully connected spanning tree, we can apply DCN construction algorithm to exchange messages in an HNOW. Figure 11 shows how to build a compact spanning tree. Switch S_5 and its connected sub-tree is moved to switch S_3. Switch S_3 disappears.

Lemma 2. Given a spanning tree T, T is obtained from an HNOW and T is a partially connected spanning tree. Algorithm 4 can generate a fully connected spanning tree from T.

Proof. Algorithm 4 is executed from bottom to top and from left to right according to the postorder in T. Observe all sub-trees which are composed by switches and each has more than one child. Notably, a linear array is also a kind of sub-tree. If the root of this sub-tree R is

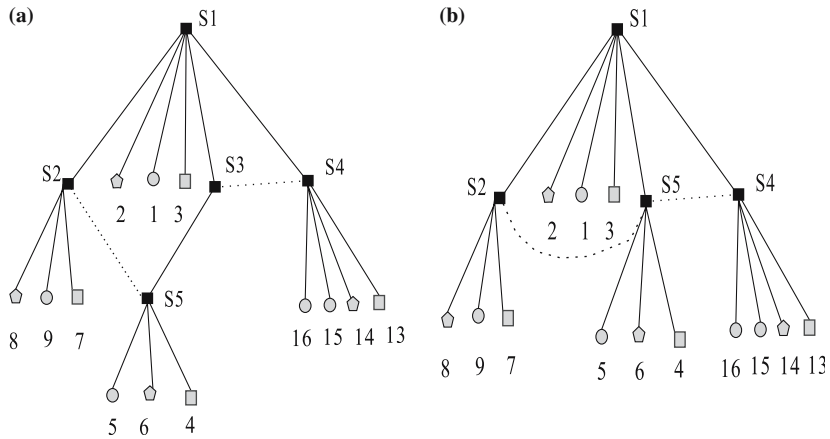

Fig. 11. Compact spanning tree, (a) a partially connected HNOW, and switch S_3 connects to no workstation. (b) The compact spanning tree is reconstructed by Algorithm 4. Switch S_3 disappears. The tree is a fully connected HNOW, that is easily partitioned.

empty, then the non-empty switch with maximal postorder in the sub-tree is selected as the root to substitute R and all the links that connected to other switches are unchanged. After step (2) is executed, all switches has workstations to be connected except the leaf (the switch without child switch). Finally, if the empty switch is a leaf, then it also should be deleted from the spanning tree. The re-processed spanning tree is concluded to be fully connected. ▮

The concept of compact spanning tree can be used to multicast in an HNOW because workstations in some switches do not require to receive the messages. This is the same as broadcasting on a partially connected HNOW. In such a situation, compact spanning tree can be used for multicasting.

7. PERFORMANCE ANALYSIS

The performance of an HNOW is determined by the enumeration of two important factors in this paper. The first is that the number of steps in routing is decreased as much as possible; the second factor is that in a wormhole routing every fastest node is walked on first, hence it is called *FNF*. An algorithm that guarantees taking minimal steps and routes through the fastest nodes first is an optimal routing in HNOW. Unfortunately, such an algorithm is difficult to obtain.

The recursive doubling is not flexible and it can not arrange the fastest nodes in the internal nodes of a scheduling tree by exchanging workstations in the same switch, the recursive doubling has the worst performance of all other schemes. Presented algorithms for routing in Section 3, EBS has the minimal step but it can not guarantee that all fastest nodes are routed first and easily influenced by density. The EBS usually performs worse than VBBS and VBBSWF.[9] The reason is that EBS has few opportunities to exchange all slow nodes in the upper level (height) with fastest nodes in lower level in a scheduling tree. When an HNOW is partitioned into two sub-networks and TWO-EBS and TWO-VBBS are applied, TWO-VBBS performs the best due to several reasons.

- TWO-VBBS uses two-level routing. The fastest nodes in two sub-networks are used for concurrent routing in the first level. This routing causes less contention and is likely to be contention-free if two sources are chosen by adjacent sub-trees.
- TWO-VBBS uses buckets for routing in the second level. All workstations in a bucket are occupied in some local areas and easily cause a contention-free routing.
- If two unicasts are routed in a single bucket, they usually have different time steps in the scheduling tree or two unicasts will be contention-free in a single switch.

- The network partitioning is proven to be effective and parallel routing improves performance in a NOW.[10]
- TWO-VBBS uses two sub-networks to broadcast concurrently. Two sources send one half of the original message to two sub-networks, respectively. The latency in any unicast can be reduced one half of total routing time.

Furthermore, TWO-EBS performs worse than TWO-VBBS because TWO-EBS has few opportunities to exchange all slow nodes in the upper level (height) with fastest nodes in lower level of a scheduling tree, the number of contentions is increased when two sub-networks are routed concurrently.

8. SIMULATION RESULTS AND COMPARISONS

The topology of any HNOW that composed of switch and workstation in our experiments is randomly generated. The network sizes are 16, 32, 64, 128, 256, and 512 nodes. The densities are 2, 4, 8, and 16. The number of workstation in a high density HNOW is usually more than that of workstations in a low density HNOW. Each broadcasting message with a length of 2048 flits is a small message, while the message with 10240 flits is a large message.

Tables I–IV, they show timing parameters. Table I shows fast speed parameters. There are at most eight distinct speed types in these HNOWs as shown in Table I. Table II shows two types of speed parameters in a fast speed HNOW. Table III also has two types of speed parameters for a slow speed HNOW. Table IV shows four speed types with slow speed parameters. The parameter X_m is $0.125\,\mu$s. X_m is the speed for message to pass through a switch. Here, X_m is a constant. x_c is $16\,\mu$s.

8.1. Network Size

In Fig. 12, Tables II and III are used for simulation. Network sizes are 16, 32, 64, 128, and 256 workstations in an HNOW. The Flit size is 10240. The density in Fig. 12(a) is 4. The density in Fig. 12(b) is 8. The types of workstations are generated randomly. In Fig 12(a), TWO-VBBS outperforms the postorder by about 50%. In Fig. 12(b), TWO-VBBS outperforms the postorder around 78%. The VBBSWF and VBBS performs at the same level since they have only two speed types. As the network size is enlarged, the performance increases significantly. TWO-VBBS outperforms TWO-EBS, VBBS, EBS, and VBBSWF.

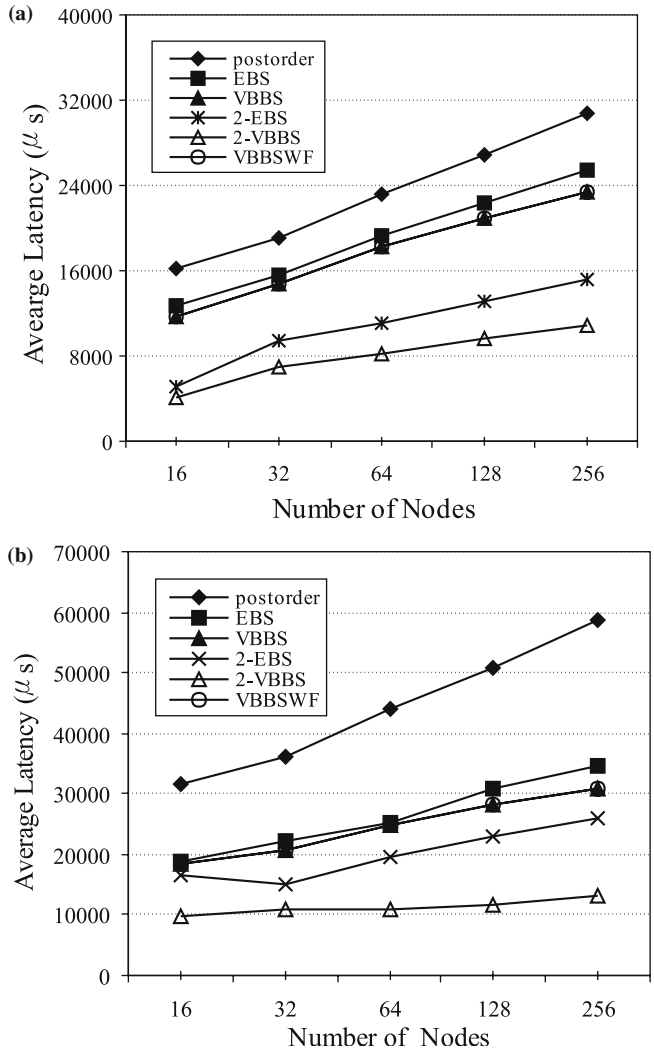

Fig. 12. Latency versus network size, (a) Two types obtained from Table II, with a message size of 10240 flits. The types of workstations are generated randomly. The density is 4. (b) Two types are obtained from Table III, with a message size of 10240 flits. The types of workstations are generated randomly. The density is 8.

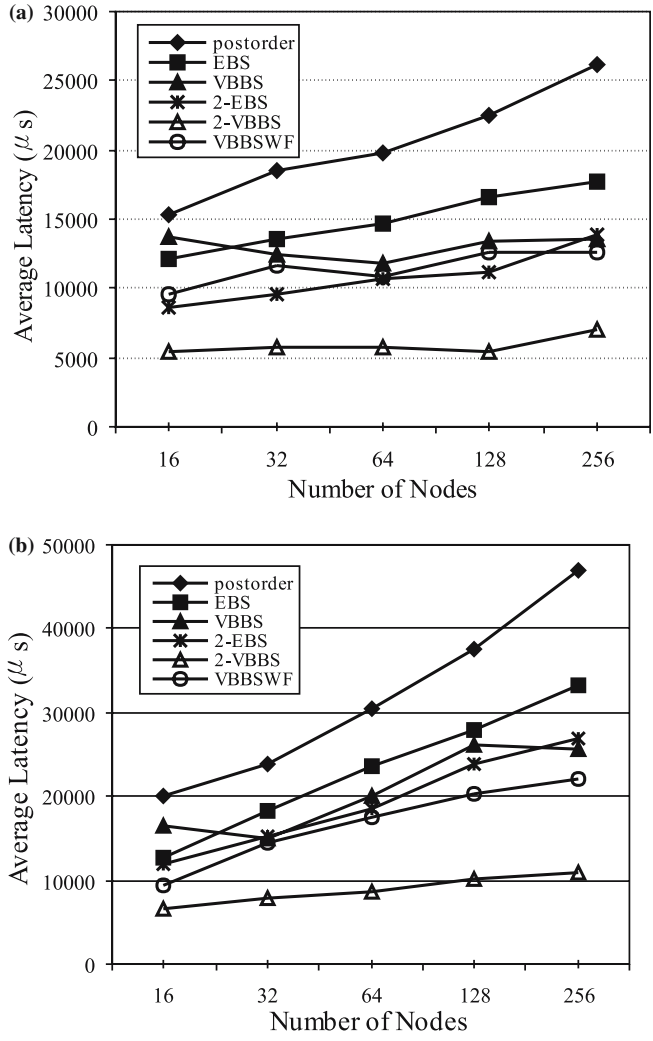

Fig. 13. Latency versus network size, (a) four types obtained from Table I, with a message size of 2048 flits. (b) The first six speed types are taken from Table I, with a message size of 2048 flits. The types of workstations are generated randomly. The density is 4.

8.2. Speed Types

This section compares the performance of eight distinct speed types in Table I. The speed types of workstations are also generated randomly. The density is 8. The filt size is 2048. The experiments are performed by two types, four types, six types and eight types. Figure 13(a) with four speed

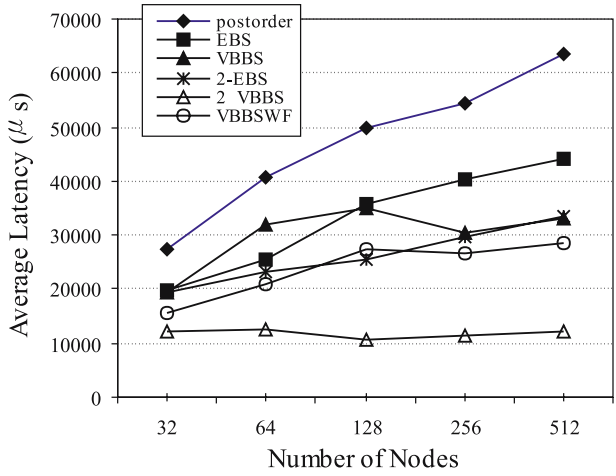

Fig. 14. Latency vs. network size, 8 types obtained from Table I, with a message size of 2048 flits. The types of workstations are generated randomly. The density is 8.

types shows that TWO-VBBS outperforms the postorder around 74%. Figure 14(b) with six speed types illustrates TWO-VBBS outperforms the postorder around 81%. Figure 13 with eight speed types shows TWO-VBBS outperforms the postorder around 81%. Thus, as the number of speed types is increased. TWO-VBBS outperforms TWO-EBS, EBS, VBBS, and VBBSWF.

8.3. Density

In the following experiments, the first three speed parameters are selected from Table IV. The types of workstations are generated randomly. The flit size is 2048 flits. The densities vary from 2 to 16. The performance of EBS increases with high density. EBS performs poorly when the density is low because there are few opportunities to exchange the slow node with the fastest node in the same switch. Figure 15(a) and (b) demonstrate that TWO-VBBS outperforms the postorder about 80% and 82% in networks of 128 and 512 nodes, respectively. As the density increases, fast nodes can be easily arranged in internal nodes of the scheduling tree by EBS, VBBS, and VBBSWF. The performance is significantly increased accordingly.

8.4. Number of the Slowest Workstations

This sub-section considers the simulation results obtained by varying the number of the slowest workstations in an HNOW. The flit size

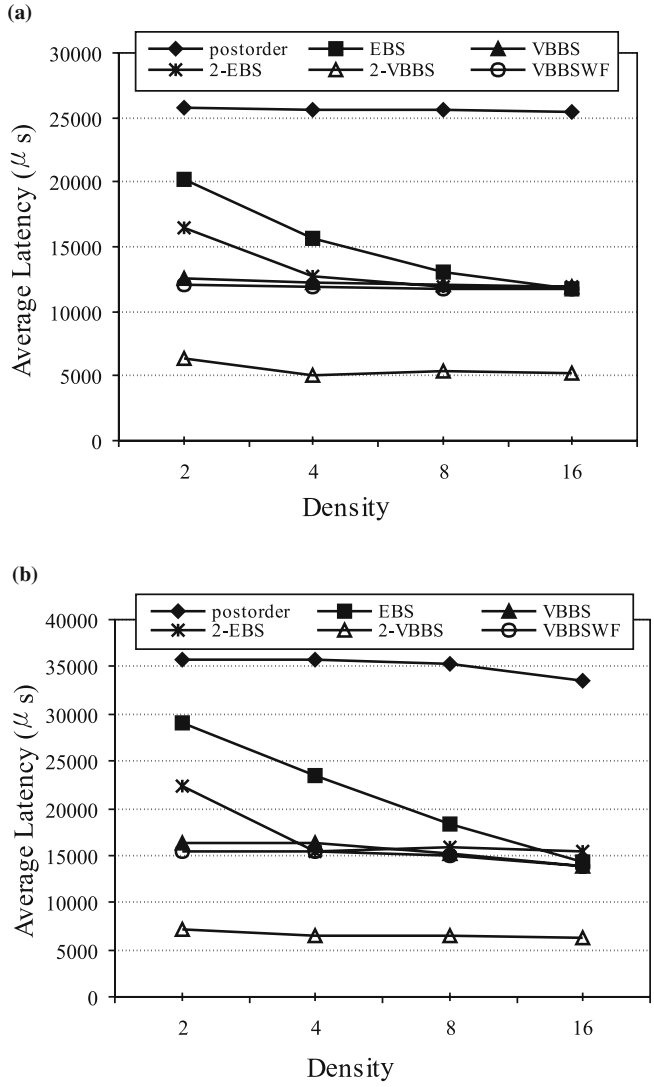

Fig. 15. Latency versus density, (a) the network size is 128 nodes, and (b) the network size is 512 nodes. The types of workstations are generated randomly. Three speed types, and 2048 flits.

is 2048. The density is eight. The speed parameters with four types from Table IV are used. The slowest nodes are randomly distributed in an HNOW. The ratio of slowest workstations to all workstations is 0.2 in Fig. 16(a). The ratio of slowest workstations to all workstations is 0.4 in Fig. 16(b). From the simulation results, TWO-VBBS outperforms the postorder around 73% in Fig. 16(a). TWO-VBBS outperforms the postorder around 73% in Fig. 16(b). TWO-VBBS outperforms TWO-EBS, VBBS, EBS, and VBBSWF.

8.5. Distribution of the Slowest Workstations

This sub-section considers the variance of the distribution of the slowest workstations in an HNOW. Here, we only consider uniform and local distribution of the slowest nodes. The density is eight, and the flit size is 2048 in Fig. 17(a) and (b). Figure 17(a) shows uniformly distributed slowest nodes. The total proportion of the slowest node is 20%. TWO-VBBS outperforms TWO-EBS, VBBS, EBS, and VBBSWF. Figure 17(a) shows TWO-VBBS outperforms the postorder around 83%. Figure 17(b) shows locally distributed slow nodes. The performance is increased by around 84%. A large number of locally distributed slowest nodes do not benefit broadcast. TWO-VBBS outperforms TWO-EBS, VBBS, EBS, and VBBSWF.

8.6. Partially Connected HNOWS

Figure 18(a) shows that only 60% of switches in an HNOW have workstations to be connected. Figure 18(b) shows that only 80% of switches in an HNOW have workstations to be connected. They all have empty switches. Three parameters are taken from Table IV. The slowest nodes are uniformly distributed. The flits size is 2048. These two HNOWs have the same number of workstations. The density is increased when these two HNOWs have 20% or 40% of empty switches. TWO-VBBS performs the best and outperforms the postorder around 73% in Fig. 18(a). TWO-VBBS performs the best and outperforms the postorder around 72% in Fig. 18(b).

8.6.1. Comparison

Recently, Singhal *et al.*[14] presented a similar study in an HNOW. Their algorithm, called CFOOT, used the postorder recursive doubling[8] for routing. The CFOOT used dynamic programming to obtain the optimal tree. In comparing our schemes to CFOOT, we mention first that

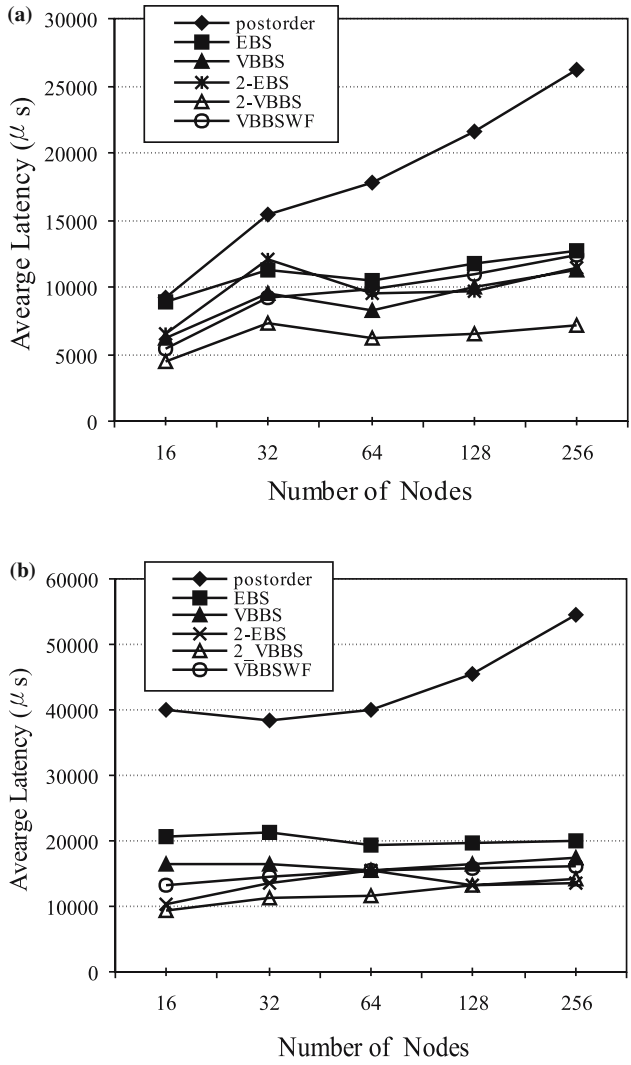

Fig. 16. Latency versus network size, (a) the ratio of slow nodes is 0.2, four types from Table IV, and the flits size 2048 flits and the density is eight. (b) the ratio of slowest workstations to all workstations is 0.4, four types are taken from Table IV , the message size 2048 flits, and the density is eight. The slowest nodes are generated randomly.

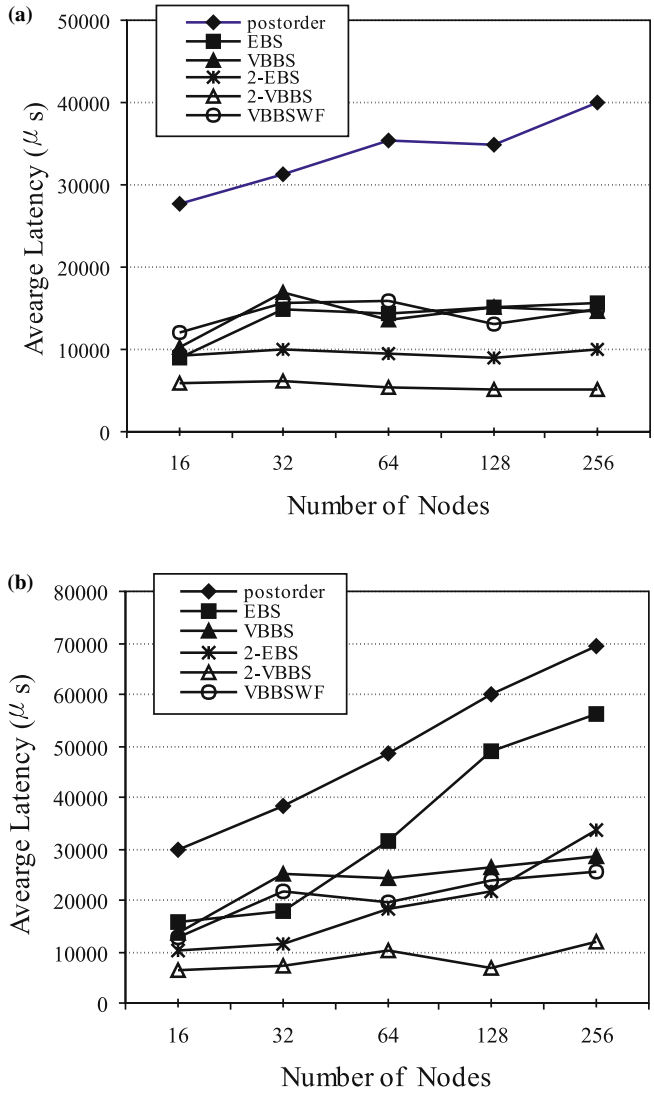

Fig. 17. Latency versus network size, (a) the ratio of slow nodes 0.2, four types from Table IV, 2048 flits, uniform distribution, and the density is eight. (b) The ratio of slow nodes 0.2, four speed types from Table IV 2048 flits, local distribution, and the density is eight.

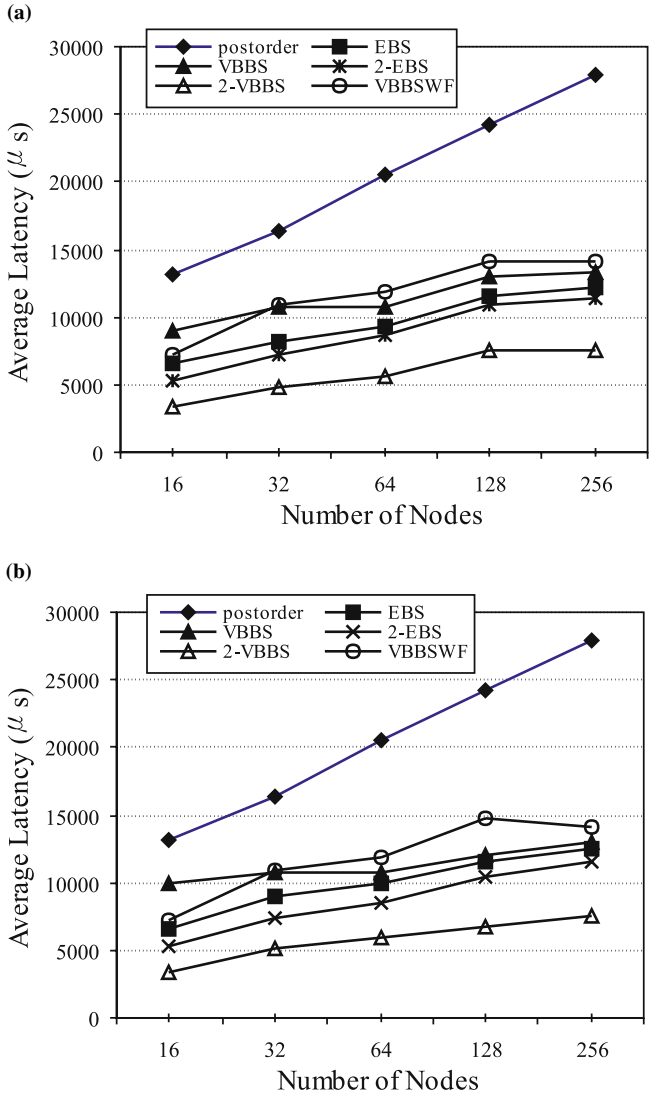

Fig. 18. Latency versus network size, (a) the ratio of partially connected now is 0.6. Three types taken from Table IV The HNOW has 2048 flits, uniform distribution, and the density 13. (b) The ratio of slow nodes 0.8. Three speed types are taken from Table IV. The HNOW has 2048 flits, uniform distribution and the density 11.

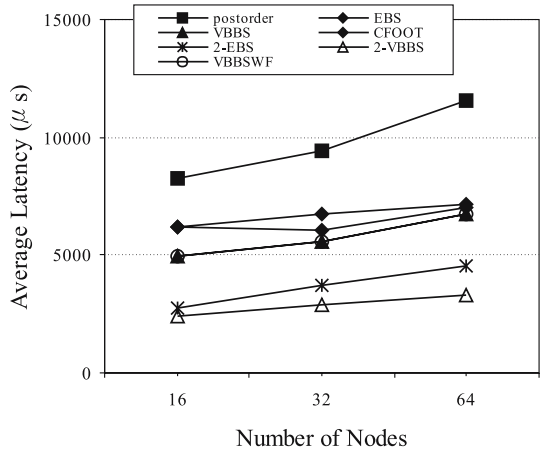

Fig. 19. The latency of TWO-VBBS, TWO-EBS, VBBSWF, and EBS compares with that of CFOOT designed by the dynamic programming. TWO-VBBS outperformsed TWO-EBS, VBBSWF, EBS and CFOOT. Notably, the density is 4, and speed parameters are obtained from Table III.

CFOOT only uses two speed types, fast and slow, for routing, whereas our new scheme uses multiple speed types for routing. Second, the model presented here includes receive overhead latency, but CFOOT does not. Third, our schemes is more flexible than CFOOT because workstations in the same switch positions can be exchanged in a rotation list or a scheduling tree. The idea of exchanging positions can improve performance. The CFOOT does not use this idea. Fourth, although CFOOT creates an optimal tree by dynamic programming, its complexity is $O(n^3)$. This scheme requires too much time to obtain the optimal scheduling tree. Obtaining the optimal solution by dynamic programming involves great complexity. Thus, VBBS, EBS, and VBBSWF can outperform CFOOT. In Fig. 19, TWO-VBBS used two sub-networks to broadcast concurrently. TWO-VBBS outperformed CFOOT with two speed types and TWO-VBBS also outperformed CFOOT, VBBS, VBBSWF, and EBS with multiple send and receive speeds.

9. CONCLUSION

This paper has presented network partitioning for routing in an HNOW which demonstrates an improvement over.[8,10,14] Much latency can be saved by using the TWO-VBBS schemes for broadcasting. Although the optimal scheduling tree with two speed types can be obtained through dynamic programming,[14] but this method is not flexible and is executed

in $O(n^3)$. Finding the optimal solution with multiple speed types is very difficult, and is also an NP-complete problem. TWO-VBBS uses two-level routing. In the first level, two sub-networks are routed on the fastest nodes and, the number of contentions is decreased. In the second level, two sub-networks are passed in buckets. All workstations in a bucket are occupied in some local areas. Routing via buckets also can reduce contention. Based on the simulation results, TWO-VBBS outperforms CFOOT, VBBS,VBBSWF, and EBS. Clearly, routing with network partitioning is more efficient than routing without network partitioning in an HNOW. Open issues to be investigated in the future include how to obtain more than two sub-networks and determine the optimal number of sub-networks in an HNOW.

REFERENCES

1. M. Banikazemi, V. Moorthy, and D. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations, *In International Conference on Parallel Processing*, p. 460–467 (1998).
2. M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda, and P. Sadayappan. Communication Modeling of Heterogeneous Networks of Workstations for Performance Characterization of Collective Operations, In *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth, 1999*, pp. 125–133 (1999).
3. N. J. Boden and D. C. Myrinet, et al., A Gigabit-Per-Second Local Area Network, *IEEE Micro*, **15**(1):62–76 (1995).
4. W. J. Dally and C. L. Seitz, The Torus Routing Chip. *Journal of Parallel and Distributed Computing*, **1**(3):187–196 (1986).
5. R. Horst, Servernet Deadlock Avoidance and Fractahefral Topologies. In *Proc. Int'l Parallel Processing Symp*, 1996 pp. 274–280.
6. R. Kesavan and D. K. Panda, Efficient Multicast on Irregular Switch-based Cut-Through Networks With Up-Down Routing, *IEEE Transaction on Parallel and Distributed Systems*, pp. 808–828 (2001).
7. R. Libeskind-Hadas and J. Hartline, Efficient Multicast in Heterogeneous Networks of Workstations, In *Parallel Processing, 2000. Proceedings. 2000 International Workshops on, 2000*, pp. 403–410 (2000).
8. R. Libeskind-Hadas, D. Mazzoni, and R. Rajagopalan, Optimal Contension-Free Unicast-Based Multicasting in Switched-Based Networks of Workstations, In *Merged IPPS/SPDP Conf.*, pp. 358–364 (1998).
9. C. Lin, Efficient Contention-Free Broadcast in Heterogeneous Network of Worstation with Multiple Send and Recive Speeds. In *Proc. 8th IEEE International Simposium on Computer and communication*, pp. 1277–1284 (2003).
10. C. Lin, Y.-C. Tseng, and J.-P. Sheu, Efficient Single Node Broadcast in Switched-Based Network of Workstations with Network Partitioning, In *Proc. 10th International Conference on Computer Communications and Networks*, pp. 68–73 (2001).
11. P. K. Mckinley, H. Xu, A.-H. Esfahanian, and L. Ni, Unicast-Based Multicast Communication in Wormhole Routed Networks, *IEEE Trans. on Parallel. and Distributed. Syssten*, **5**(12):1252–1265 (1994).

12. L. M. Ni and P. Mckinley, A Survey of Wormhole Routing Techiniques in Directed Network, *IEEE Computers*, **26**(2):62–76 (1993).
13. M. Schroeder et al., Autonet A high-speed, Self-Configuring Local Area Network Using Point-to-Point Links, *IEEE Journal on Selected Areas in Communications*, **9**(8):1318–1335 (1991).
14. A. Singhal, M. Banikazemi, P. Sadayappan, and D. Panda, Efficient Multicast Algorithms for Switch-Based Irregular Heterogeneous Networks of Workstations, In *Parallel and Distributed Processing Symposium., Proceedings 15th International*, 2001 8 pp.
15. Y.-C. Tseng, S.-Y. Wang, and C.-W. Ho, Efficient Single-Node Broadcasting in Wormhole-Routed Multicomputers: A Network-Partioning Approach, *IEEE Trans. on Parallel and Distributed System* **10**(1):44–61 (1999).