Efficient Multiple Multicast on Heterogeneous Network of Workstations

Jan-Jan Wu Shih-Hsien Yeh
Institute of Information Science
Academia Sinica
Taipei, Taiwan, R.O.C.
wuj@iis.sinica.edu.tw

Pangfeng Liu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
pangfeng@csie.ntu.edu.tw

Abstract

In recent years, network of workstations/PCs (so called NOW) are becoming appealing vehicles for cost-effective parallel computing. Due to the commodity nature of workstations and networking equipment, LAN environments are gradually becoming heterogeneous. The diverse sources of heterogeneity in NOW systems pose a challenge on the design of efficient communication algorithms for this class of systems.

In this paper, we propose efficient algorithms for multiple multicast on heterogeneous NOW systems, focusing on heterogeneity in processing speeds of workstations/PCs. Multiple multicast is an important operation in many scientific and industrial applications.

Multicast on heterogeneous systems has not been investigated until recently. Our work distinguishes itself from others in two aspects: (1) In contrast to the *blocking* communication model used in prior works, we model communication in a heterogeneous cluster more accurately by a *non-blocking* communication model, and design multicast algorithms that can fully take advantage of non-blocking communication, (2) While prior works focus on single multicast problem, we propose efficient algorithms for general, multiple multicast (in which single multicast is a special case) on heterogeneous NOW systems. To our knowledge, our work is the earliest effort that addresses multiple multicast for heterogeneous NOW systems.

These algorithms are evaluated using a network simulator for heterogeneous NOW systems. Our experimental results on a system of up to 64 nodes show that some of the algorithms outperform others in many cases. The best algorithm achieves completion time that is within 2.5 times of the lower bound.

Keywords: collective communication heterogeneous network of workstations, parallel processing, multiple multicast, scheduling algorithms.

1 Introduction

Due to the commodity nature of workstations and networking equipments, LAN environments are gradually becoming heterogeneous. The heterogeneity could be due to the difference in processing speed and communication capability of the workstations, or coexistence of multiple network architectures or communication protocols. This trend is forcing network of workstations/PCs to be redefined as Heterogeneous Network of Workstations (HNOW).

Many research projects are currently in progress to provide efficient communication for NOW systems [1, 5, 6, 7, 9, 12, 10, 22, 16, 13, 21, 23]. However, most of these research projects focus on homogeneous NOWs, systems comprising of the same type of PCs/workstations connected over a single network architecture. Communication algorithms designed for homogeneous clusters have been shown to be very inefficient for heterogeneous clusters [2].

Collective communication, where all processors contribute data to a result that may arrive at one or all processors, provides important functionality for many applications, and thus efficient implementations of collective communication is crucial for achieving maximum performance in message-passing systems. Typical examples of collective communication include *multicast*, *broadcast*, *multiple multicast*, *gather*, *scatter*, and *all-to-all personalized* communication.

In this paper we study the multicast problem in HNOW systems. Multicast is an important operation in many scientific, industrial, and commercial applications. In a single multicast, the source node sends the same message to a subset of nodes in the system. Multiple multicast is a general case in that multiple source nodes issue multicast communication simultaneously. Multiple multicast is frequently used in sparse matrix computation and many scientific simulation problems [15].

Multicast can be implemented at different levels: hardware-supported, network interface firmware-supported, and software implementation based on point-to-point messages. We focus on software implementation of multicast because it does not require modification of hardware/firmware and therefore is portable to different cluster environments.

Software-based multicast on heterogeneous systems has not been investigated until very recently. In software-based approach, a multicast is implemented as p sequences of send/receive tasks, where p is the number of processors involved in the multicast. The fact that finding optimal sequences of tasks for multicast on heterogeneus systems is NP-complete has led to a number of research works on devising heuristic algorithms [2, 17, 19]. These works have two restrictions however. First, they all focus on single multicast. Although a multiple multicast operation can be implemented by performing the single multicast operations individually, it may not be the most efficient way to do it. Secondly, prior works all assume a *blocking* communication model; that is, a node cannot send a message until the previously sent message has been received by the destination node. However, many networks and operating systems support *non-blocking* communication; that is, after an initial start-up time, the sender can send the next message. The previously sent messages can be completed by the network without intervention of the sender. Thus, a node can send out several messages before the first message is received by the destination. To optimize performance of multicast communication, it is extremely important to take the factor of non-blocking communication into consideration when designing algorithms, which is not possible under a *blocking* communication model.

In this paper, we design efficient algorithms for multiple multicast in heterogeneous systems under a non-blocking communication model. Our communication model represents the communication latency between two processors P_i and P_j using three parameters: (1) send overhead on sender P_i, (2) data transmission time, and (3) receive overhead on P_j. Under this model, we formalize the multiple multicast problem as finding for each processor a sequence of send/recv operations (call it *task schedule*) that lead to minimum completion time of the multicast. Since the problem of finding optimal task schedule for multicast is NP-complete, we have developed several heuristic algorithms based on our communication model.

In this paper, we present the heuristic algorithms that we have implemented. Two algorithms, the *Fastest Edge First* and the *Earliest Completion First* are modified based on existing algorithms for single multicast. The algorithms *Earliest Available Earliest Completion First*, *Round-Robin*, and a randomized algorithm are motivated by scheduling strategies in operating systems and parallel and distributed computing systems. The algorithm *Work Racing* is new. The *Work Racing* algorithm proposes a notion of *virtual time*, which estimates the time a destination node has spent in receiving messages. The algorithm selects receiving nodes based on their virtual time. We show that the *Work Racing* algorithm has low cost and generates the best task schedule.

In addition to these algorithms, we have also proposed an optimization that can fully take advantage of non-blocking communication. This optimization fills the receiving node's idle time frames with useful send operations as much as it allows. With this optimization, the completion time of multiple multicast can be greatly shortened.

These algorithms are evaluated using a network simulator for HNOW systems. Our experimental results on a system of up to 64 nodes demonstrate performance improvement of multiple multicast by 20% to 160% compared to existing algorithms.

The rest of the paper is organized as follows. Section 2 defines our communication model for heterogeneous NOW systems. Section 3 formulates the multiple multicast problem based on our communication model. Section 4 presents our heuristic algorithms for multiple multicast. Section 5 describes the optimization for non-blocking communication. Section 6 uses an example to illustrates the algorithms and the optimization. Section 7 reports our simulation results for multiple multicast on a heterogeneous NOW system. Section 8 gives some concluding remarks and outlines our future work.

2 Communication Model for Heterogeneous Systems

We assume that a collective operation is implemented by a series of point-to-point message passing. Since in this paper, we focus on the heterogeneity in processing speeds of the processors, we make an assumption that there is a direct link between every pair of processors, so each pair of processors can pass messages independent of other processors. Under this assumption, we adopt the model for point-to-point communication on heterogeneous network of workstations proposed by Banikazemi et al. [3]. To make this paper self-contented, we give a brief overview of the communication model.

The model measures the cost of a point-to-point message transfer between the sender P_i and the receiver P_j using three parameters (1) the send overhead $S(i,m)$, which represents the message initialization cost on sender P_i for sending a message of length m, (2) the network link transmission rate $X(i,j)$, which accounts for the unit transmission time between P_i and P_j, and (3) receive overhead $R(j,m)$, which represents the software overhead on receiver P_j for receiving and copying the message from the network buffer to the user space. Based on these three parameters, the latency for transmitting a message of length m between the two nodes is given in Equation 1.

$$T(i,j,m) = S(i,m) + X(i,j) * m + R(j,m) \tag{1}$$

For a short message, the send overhead and receive overhead are nearly constant and independent of the message size. However, for a long message, the message size-dependent cost contributing to the software overhead dominates the overall software cost. Let $S_c(i)$ and $R_c(i)$ be the constant factors of the send and receive costs of node P_i respectively, and S_mi and R_mi be the message-dependent parts. The send overhead $S(i,m)$ and receive overhead $R(i,m)$ of node P_i for a message of size m can be further refined as the following. Each parameter in the equation can be measured using the ping-pong scheme described in [3].

$$S(i,m) = S_c(i) + S_m(i) * m \quad (2)$$
$$R(i,m) = R_c(i) + R_m(i) * m \quad (3)$$

Note that in HNOW systems these parameters may vary for different pairs of nodes. Furthermore, in this paper we make the following assumptions.

- A send operation is *non-blocking.* In other words, after an initial *start-up* time (i.e. the send overhead), the sender can execute its next send operation.
- A receive operation is *blocking.* That is, after issuing a receive operation the receiving node can continue to execute its next operation only when the received message arrives and is removed from the local network buffer to the local memory of the receiving node. We consider this assumption reasonable as in most applications, the computation following a receive operation is very likely to require data in the received message and hence the computation cannot start until the required data is ready in the local memory.

3 Multiple Multicast Problem

This section formulates the *multiple multicast* problem using the notion of task schedule.

3.1 Definitions

A multiple multicast has multiple source nodes multicasting their messages to their destination nodes simultaneously. Consider a HNOW system consisting of N nodes and let $\mathbf{N} = \{P_1, \mathrm{P}_2, \ldots, P_N\}$ be the set of all nodes. Let $\mathbf{S}$ denote the set of the multicast source nodes, and $\mathbf{D_i} \subseteq \mathbf{N} - \{P_i\}$ be the set of destination nodes for source node $P_i \in \mathbf{S}$. After the multiple multicast communication, each node in $\mathbf{D_i}$ has a copy of the message from source node P_i, denoted by m_i. Both a single broadcast and an all-to-all broadcast are special cases of multiple multicast. Figure 1(a) shows an example of two multicast operations.

The multiple multicast is accomplished by a series of *communication tasks.* A communication task (op, i, j, k) is a send or receive operation that P_i is scheduled to execute. For $op = send$, P_i is scheduled to send m_k to P_j, where $P_i \in \mathbf{D_k} + \{P_k\}$ and $P_j \in \mathbf{D_k}$. For $op = recv$, P_i is scheduled to receive m_k from P_j where $P_i \in \mathbf{D_k}$ and $P_j \in \mathbf{D_k} + \{P_k\}$. The *task schedule* of P_i, denoted by $\mathbf{TaskList_i}$, is an ordered list of communication tasks (op, i, j, k). The tasks in the task schedule are executed in the order they appear. Figure 1(b) illustrates a communication schedule for the multicast with source P_1.

The multiple multicast scheduling problem is to determine a task schedule for each participating node so that the time to deliver all the messages is as short as possible. In addition, for dynamic patterns of multicast, the scheduling needs to be done on-line. Thus the scheduling algorithm must generate efficient task schedule within reasonable amount of time.

3.2 A Lower Bound

We first derive a lower bound on the time to solve a multiple multicast problem as the basis for evaluating our algorithms. Since it is too computationally expensive to determine the optimal completion time of a multiple multicast, we idealize our model and deduce an *idealized optimal completion time* as a lower bound. In this model we assume that a node is allowed to receive a message and, meanwhile, send multiple messages in parallel. That is, a receive operation will not be delayed by any send operation, and the messages from one node to different destinations can be processed in parallel.

Since the completion time of a multiple multicast is determined by the maximum of the *task completion time* of all the destination nodes, we can compute the lower bound on the task completion time for each destination node, and select the maximum as the lower bound on the overall completion time. First we compute the shortest path for each pair of source and destination in the idealized model. Let $L[P_j, \mathrm{P_i}]$ denote the cost of the shortest path from source P_j to its destination P_i, i.e., $L[P_j, \mathrm{P_i}]$ represents the *earliest-reach-time* at which the message from P_j can reach P_i. Thus, the lower bound on the task completion time of a destination node can be defined as follows.

Definition 1 *Assume that a node P_i needs to receive messages from n_i source nodes. Let $I(k)$ denote the kth source node from which P_i receives the message. Suppose that P_i receives messages in the increasing order of the earliest reach times of the messages, i.e., $L[I(k), P_i] \geq L[I(k-1), P_i]$ for $1 < k \leq n_i$, then we call this receiving order the* earliest-reach-first *order.*

Let l_k be the message size of the kth source node $I(k)$. The earliest receiving time for P_i to complete receiving the message from the kth source node $I(k)$, denoted by $T_r(i,k)$, can be derived recursively as follows. Note that Equation 4 uses the fact that the receive overhead from different source nodes cannot overlap, and the earliest-reach-first order can minimize $T_r(i,k)$.

$$T_r(i,k) = \begin{cases} L[I(k), i] & k = 1 \\ \max\{T_r(i,k-1) + R(i,l_k),\ L[I(k), i]\} & 1 < k \leq n_i \end{cases} \tag{4}$$

The lower bound on the completion time of the multiple multicast is the maximum of all $T_r(i, n_i)$, for all i.

4 Scheduling Algorithms for Multiple Multicast

In this section, we present the heuristic algorithms we have devised for implementing multiple multicast on HNOW systems. The heuristics are *Fastest-Edge-First* (FEF), *Earliest-Completion-First* (ECF), *Earliest-Available-First* (EAF), *Round-Robin* (RR), *Receiver Random Seletion* (RRS) and *Work Racing* (WR). We first give some definitions.

Available time. The *available time* of P_i, denoted as $Avail_i$, is the earliest time at which P_i can execute a new task. Initially, the available time of the participating nodes is zero.

Arrival time. Assume that P_i is scheduled to send m_k to P_j. Let l_k be the size of m_k, then m_k will arrive at the network buffer of P_j at time $ArrivalTime(i,j,k)$.

$$ArrivalTime(i,j,k) = Avail_i + S(i,m) + X(i,j) * l_k \tag{5}$$

Completion time. The completion time, $CompleteTime(i,j,k)$ of a task $(send, i, j, k)$ is defined as follows. If the destination node P_j is not available when m_k arrives at its network buffer, it will not be processed until P_j becomes available.

$$CompleteTime(i,j,k) = \max(ArrivalTime(i,j,k), Avail_j) + R(j, l_k) \tag{6}$$

4.1 Fastest-Edge-First Algorithm

The Fastest Edge First (FEF) heuristic algorithm was first proposed in [20] for implementing single multicast on heterogeneous NOW systems. We extend the FEF algorithm to solve the multiple multicast problem. We keep a sender set A_k and a receiver set B_k for each source node P_k of the multiple multicast. A node in A_k can relay the message it received to other destination nodes. Initially $A_k = \{P_k\}$ and B_k contains all the destination nodes of P_k.

In each iteration we select the $(i,\ j,\ k)$ that has the smallest weight, where node P_i belongs to A_k and node P_j belongs to B_k. We then move P_j from B_k to A_k. The same steps repeat until all B_k become empty. The *weight* of a tuple $(i,\ j,\ k)$ is defined as the point-to-point latency between the sender P_i and the receiver P_j as follows.

$$Weight(i,\ j,\ k) = S(i, l_k) + X(i,j) * l_k + R(j, l_k) \tag{7}$$

Similar to [19], we keep a sorted tuple list and a sorted sender list according to their weights. The sorting process takes $O(N^3 \log N)$ time steps where N is the number of processing nodes in the network, and this process dominates the total execution time of this algorithm. Therefore, the complexity of this algorithm is $O(N^3 \log N)$.

Fastest-Edge-First Algorithm

```
Step 1: Select a $(i,\ j,\ k)$ with minimum $Weight(i,\ j,\ k)$, where $P_i \in A_k$ and $P_j \in B_k$.
Step 2:
        $A_k \leftarrow A_k + \{P_j\}$
        $B_k \leftarrow B_k - \{P_j\}$
        $AppendTask(\mathbf{TaskList_i}, (send, i, j, k))$
        $AppendTask(\mathbf{TaskList_j}, (recv, j, i, k))$
Step 3: Repeat Step 1 and Step 2 until all $D_k$ become empty.
```

4.2 Earliest-Completion-First Algorithm

The *Earliest-Completion-First* (ECF) heuristic algorithm was also proposed in [20] for single multicast. The ECF algorithm determines the schedule iteratively. In each iteration the ECF algorithm selects the task (i.e. the sender and receiver pair) that has the earliest completion time as defined in Equation 4 as the next task to schedule. The same process repeats until all destination nodes have received the message.

We extend the ECF algorithm to implement multiple multicast. We also modify the definition of available time of a node to model non-blocking communication more accurately. Since with non-blocking send, a sender can send a new message right after the previous message is transmitted into the network, the available time for the sender and the receiver is updated in different ways as shown in the following pseudo code algorithm.

Similar to the FEF algorithm, for each source node P_k of the multiple multicast, we keep a *sender set* A_k and a *receiver set* B_k. In each iteration, the algorithm selects the earliest completing task for each source node which has not yet completed its multicast operation. Then, among these earliest-completing tasks, it selects the task with the minimum completion time as the next task to be scheduled. The receiver of the selected task is then moved from the receiver set to the sender set. Note that the tasks from different multicasts may interleave as a result of the ECF selection process.

In [19], a sorted sender list according to their minimum completion time is maintained. Unfortunately, for multiple multicast problems this is not possible because the completion time of a sender/receiver/message tuple also depends on the available times of the sender and the reciever. Therefore, the available time of a node needs be calculated dynamically during each iteration of the algorithm. In each iteration, it requires $O(N^2)$ steps to compute the available times of the senders and receivers. The algorithm iterates N^2 times. Therefore, the total time for the ECF algorithm is $O(N^4)$.

Earliest-Completion-First Algorithm

Step 1: For all k that B_k is not empty, choose (i, j, k) with minimum $C_k = CompleteTime(i, j, k)$ where $P_i \in A_k$ and $P_j \in B_k$.

Step 2: Choose k such that C_k is the minimum.

$$
\begin{aligned}
&A_k \leftarrow A_k + \{P_j\} \\
&B_k \leftarrow B_k - \{P_j\} \\
&AppendTask(\mathbf{TaskList_i}, (send, i, j, k)) \\
&Avail_i \leftarrow Avail_i + S(i, l_k) \\
&AppendTask(\mathbf{TaskList_j}, (recv, j, i, k)) \\
&Avail_j \leftarrow CompleteTime(i, j, k)
\end{aligned}
$$

Step 3: Repeat Step 1 and Step 2 until all B_k become empty.

Our experimental study shows that the *Earliest Completion First* algorithm delivers good performance in multiple multicast. However, the scheduling time of the ECF algorithm $O(N^4)$ is substantially higher than that of the *Fasted-Edge-First* algorithm $O(N^3 logN)$ in a large system (up to 64 nodes), since it has to consider all the possible sender-receiver pairs per iteration. This makes the ECF algorithm impractical for scheduling dynamic multicast patterns. In the following subsections, we present several algorithms that has lower scheduling complexity of $O(N^3)$. The idea is to reduce the search space by selecting the receiver node first and then select the sender and the message within the source set of the receiver node. We propose four heuristics for this purpose, namely *Work Racing*, *Earliest Available First*, *Round Robin*, and *Receiver Random Selection.*

Some of these algorithms generate task schedules that are competitive to the ECF algorithm. In the following, we describe each of these algorithms.

4.3 Work Racing Algorithm

In our communication model, a receive operation is *blocking*, that is, if multiple messages arrive at the receiving node, they will be queued in the buffer until the receiving node has finished receiving previous message. The key idea in the *Work Racing* (WR) heuristic is that, if the chance of message built-up at the destination nodes is reduced, then the messages can be delivered as early as possible, resulting in earlier completion of the multicast. One way to implement this strategy is to allow faster destination node to receive messages more often than slower nodes. However, the scheduling should also be fair so that slower nodes will not be starved forever.

We define the notion of *virtual time* of a destination node, which estimates the time this destination node has spent in receiving messages. WR selects the destination node with the earliest virtual time as the receiver of the next message. Then WR selects a message and a sender for this message based on the earliest-completion-first principle. The new send and receive tasks are then appended to the task schedules of the sender and the receiver respectively. After that, the *virtual time* of the receiver is increased by the amount of work it has just accomplished. This mechanism ensures that the faster destination node will be selected more often, and each destination node will be scheduled fairly according to their communication capability. In the following we elaborate on this algorithm.

- For each destination P_i, assuming P_i has received messages from k sources so far, the WR algorithm keeps the record of the *virtual time*, denoted by $W_{i,k}$, which indicates the services that P_i has received from its sources. Initially, $W_{i,0}$ is set to *zero*.
- While scheduling a new task, the WR algorithm selects the destination node which has the earliest virtual time. If there are multiple possibilities, it chooses the fastest one.
- Each destination node P_i keeps a set of source nodes SR_i of the multiple multicast in which P_i is a destination. Each source node P_k keeps a set of sender nodes A_k which contains the nodes that have received message m_k. Nodes in A_k may relay m_k to other nodes. When a destination P_i is selected, the Work Racing algorithm chooses a source P_m from SR_i and a sender P_j from A_m such that the completion time of the task $(send, j, i, m)$ is the minimum.
- The destination P_i will be scheduled a receive task $(receive, i, j, k)$. Assume that this is the kth receive task that P_i has been scheduled (and we will call the source node of this message the kth source node of P_i), the virtual time $W_{i,k}$ of P_i is updated as follows:

$$W_{i,k} = \begin{cases} A_r[k,i] + R(i, l_k) & k = 1 \\ \max\{W_{i,k-1},\ A_r[k,i]\} + R(i, l_k) & k > 1 \end{cases} \tag{8}$$

  where l_k is the message length of this receive task and $A_r[k,i]$ is the time at which the message from the kth source node arrives at P_i. Let the source node of the message be the nth source in sender P_j. Then, $A_r[k,i]$ is calculated as follows:

$$A_r(k,i) = \begin{cases} W_{s,n} + S(s, l_k) + X(s,i) * l_k & \text{if } P_s \text{ is not the source node} \\ S(s, l_k) + X(s,i) * l_k & \text{otherwise} \end{cases} \tag{9}$$

- Finally, the source node P_j is removed from the source set SR_i of the destination P_i, and the destination P_i is added to the sender set A_k of the source P_k of the multicast.

In general, the completed work of a faster node advances more slowly than that of a slower node. Thus a faster destination nodes can be scheduled earlier than a slower one. Consequently, a faster node has more chances to relay messages for the sources to the slower nodes.

Work-Racing Scheduling Algorithm

Let SR_i be the set of source nodes for a receiving node P_i. Initially, SR_i contains all the source nodes of the multiple multicast which has P_i in their destination sets. Similar to the FEF and the ECF algorithms, we define a sender set A_k for each source P_k of the multiple multicast.

Step 1: Let P_i be the node with the minimum completed work whose source set SR_i is not empty. If there are multiple possibilities, choose the fastest one.

Step 2: Choose $P_k \in SR_i$ and $P_j \in A_k$ such that the completion time of the task $(send, j, i, k)$ is the minimum. Assume the source of this task (i.e. P_k) is the mth source in P_i.

$$A_k \leftarrow A_k + \{P_i\}$$
$$SR_i \leftarrow SR_i - \{P_k\}$$
$$Compute\ W_{i,m}$$
$$AppendTask(\mathbf{TaskList_j}, (send, j, i, k))$$
$$Avail_j \leftarrow Avail_j + S(j, l_k)$$
$$AppendTask(\mathbf{TaskList_i}, (recv, i, j, k))$$
$$Avail_i \leftarrow CompleteTime(j, i, k)$$

Step 3: Repeat Step 1 and Step 2 until all SR_i become empty.

For each destination node P_i, we sort its source nodes in increasing order of their message lengths. In each iteration of the algorithm, when the destination node and its message source and sender have been selected, those senders that are not selected are marked and do not need be checked again. Therefore, for a selected receiver, it only takes $O(N)$ time steps to select the message source and the sender. The algorithm iterates at most N^2 times for multiple multicast. Therefore, the overall complexity of the WR algorithm is $O(N^3)$.

4.4 Summary

This section describes the heuristic algorithms that we have devised for implementing multiple multicast on HNOW systems. The ECF algorithm has very high scheduling overhead $O(N^4)$. The receiver-based algorithms, including the EAF, RR, RRS, and WR, reduce scheduling time to $O(N^3)$. The performance comparison of the task schedules generated by these algorithms will be reported in Section 7.

5 Optimization for Non-blocking Communication

Our communication model assumes that send operations are non-blocking and receive operations are blocking. After issueing a non-blocking send operation, a sender only has to wait until the message goes into the network, then it can start its next communication. In contrast, a node performing a blocking receive operation cannot issues another communication until the incoming message is received completely (Equation 1). For small messages, network transmission time is small compared to the software overhead on the sender and the receiver. However, for large messages, network transmission time dominates the communication latency, therefore, the time a receiver spends in waiting for the messages to go through the network can substantially degrade communication performance.

We illustrate this phenomenon by an example. In previous scheduling algorithms we always append the new task to the end of the task schedule. This can incur much longer waiting time than necessary. In Figure 2(a), a receive operation $recv[k]$ is issued at time $t1$. However, the message will not arrive until time $t4$, making the receiver idle waiting from $t1$ to $t4$.

To overcome this problem, we propose an optimization to preempt blocking receive operation with non-blocking send operation, under the assumption that it will not invalidate the original multicast schedule. As illustrated in Figure 2(b), if we preempt the receive task, $recv[k]$, with the send tasks $send[2]$ and $send[3]$, and issue $send[k]$ at $t3$ instead, the waiting time is reduced from $(t4 - t1)$ to

$(t4-t3)$. The idea is that if a new task can preempt a prescheduled receive task safely, we can reduce the waiting time by delaying the preempted receive task.

To preempt a receiving task without invalidating the original schedule, we need to observe the following rules.

- The send tasks must be executed in the order as they appear in their schedule list. As in Figure 3(a), the task $send[k]$ must appear before $send[k+1]$.
- If the sender of a task is not the source node of the message, the new send task must be scheduled *after* the sender has received the message. This is to ensure that a sender will not relay a message that it has not yet received. Figure 3(b) shows that only after the task $recv[h+u]$ in which the sender of $send[k+1]$ receives the message will it relay to the receiver of $send[k+1]$.
- The preempting send tasks cannot delay the completion time of the preempted receiving task. To avoid interfering the execution of the preempted receive $task[k]$ in Figure 3(c), the completion time of the preempting send task must be earlier than the arrival time of the message that $task[k]$ is waiting for. Let l_k and l_t be the message sizes in $task[k]$ and the new send task respectively, and $task[j]$ be the task preceding $task[k]$ in the task schedule before the new task is scheduled. The following condition must hold for the new send task to preempt $task[k]$.

$$(CompletionTime(task[k]) - R(i, l_k)) - CompletionTime(task[j]) \geq S(i, l_t) \tag{10}$$

To facilitate the implementation of this optimization, we classify the available time of a node into two: the *receive available time* and the *send available time*. The receive available time of P_i is the earliest time that P_i can issue a receive, which is defined as the maximum between the completion time of the last send task and the last receive task. The send available time of P_i *for* P_k is defined as the earliest time that P_i can send out m_k that satisfies all three rules listed above. The send available time should be later than both the completion time of the last send task by P_i and the completion time of the receive task in which P_i receives m_k. In the optimized algorithm, the sender and the receiver will update their completion time of a communication task with new definitions of available times.

Figure 16 and Figure 15 illustrate the details of finding the location to insert a new send task. The procedure *UpdateSenderState* (Figure 13) deals with the preempting process of a new send task in the sender. The procedure *UpdateReceiverState* (Figure 14) deals with the process of appending the new receive task to the task schedule of the receiver.

We improve our previous algorithms including ECF, WR, EAF, RR and RRS algorithms by this preemption optimization. The optimized versions of these algorithms are named as ECFP, WRP, EAFP, RRP and RRSP algorithms. All these algorithms have the same structure of updating available time and task schedules. The ECFP algorithm is listed below as an example.

Let SR_i be the set of source nodes for a receiving node P_i. Initially, SR_i contains all the source nodes of the multiple multicast which has P_i in their destination sets. Also, we define a sender set A_k for each source P_k of the multiple multicast.

Earliest-Completion-First Preemptive Algorithm(ECFP)

Step 1: For all k that B_k is not empty, choose (i, j, k) with minimum $C_k = CompleteTime(i, j, k)$ where $P_i \in A_k$ and $P_j \in B_k$.

Step 2: Choose k such that C_k is the minimum.

$$A_k \leftarrow A_k + \{P_j\}$$
$$B_k \leftarrow B_k - \{P_j\}$$
$$UpdateReceiverState(j, i, k, CompleteTime(i, j, k))$$
$$UpdateSenderState(i, j, k)$$

Step 3: Repeat Step 1 and Step 2 until all B_k become empty.

Since the preemption optimization does not change the structure of the unoptimized algorithms, the optimized algorithms also generate communication schedule in $O(N^3)$ time steps.

6 An Example

In this section, we use an example to illustrate these non-preemptive and preemptive algorithms. The communication involves three multicast patterns, with P_0, P_1 and P_2 as the sources respectively as shown in Figure 4 and Figure 5.

Let (i, j, k) be a communication task where P_i sends the message to the receiver P_j for source P_k. In Figure 4(a), the sequence of the tasks selected by the Fastest-Edge-First algorithm is $(0, 1, 0)$, $(2, 0, 2)$, $(0, 1, 2)$, $(0, 2, 0)$, $(1, 2, 1)$, $(0, 3, 2)$, and $(1, 3, 1)$, where i denotes the sender, j denotes the receiver and k denotes the message source. In Figure 4(b), the sequence of the tasks (i, j, k) selected by the Earliest-Completion-First algorithm is $(0, 1, 0)$, $(2, 0, 2)$, $(2, 1, 2)$, $(0, 2, 0)$, $(0, 3, 2)$, $(1, 2, 1)$ and $(1, 3, 1)$. The completion times corresponding to these tasks are 4, 5, 7, 12, 13, 18, and 19.

In Figure 4(c), the Work Racing algorithm selects task schedules according to the receiving nodes' completed work. In the first step node P_0 is chosen as the receiver and then the source P_2 from its source set is selected. This task is denoted by $(2, 0, 2)$. After task $(2, 0, 2)$ is scheduled, the completed work for this first receive task in P_0, denoted by $W_{0,1}$, is updated as $W_{0,1} = S(2) + R(0) = 2 + 3 = 5$, and so on. At the fourth step, the task $(0, 3, 2)$ is scheduled and it is the first receive task of P_3. Since for this task P_0 relays the message for source P_2, the completed work for this task, denoted by $W_{3,1}$, schould be updated as follows.

$$W_{3,1} = W_{0,1} + S(0) + R(3) = 5 + 1 + 6 = 12.$$

The resulting task schedule is $(2, 0, 2)$, $(2, 1, 2)$, $(0, 2, 0)$, $(0, 3, 2)$, $(0, 1, 0)$, $(1, 2, 1)$ and $(1, 3, 1)$.

In Figure 4(d), the Earliest Available First algorithm selects the destination with the minimum available time as the receiver of a new task. Thus, the sequence of tasks selected by the Earliest Available First algorithm is $(2, 0, 2)$, $(2, 1, 2)$, $(2, 3, 2)$, $(0, 2, 0)$, $(0, 1, 0)$, $(1, 2, 1)$ and $(1, 3, 1)$. In Figure 4(e), the task schedule derived by the Round Robin algorithm is $(2, 0, 2)$, $(2, 1, 2)$, $(0, 2, 0)$, $(0, 3, 2)$, $(0, 1, 0)$, $(1, 2, 1)$ and $(1, 3, 1)$. From the previous sequence, it can be observed that the destination nodes take turn to be the receivers of the tasks. In Figure 4(f), the Receiver Random Selection algorithm selects the receiver randomly for a new task. The resulting sequence of tasks is $(0, 1, 0)$, $(2, 1, 2)$, $(2, 0, 2)$, $(2, 3, 2)$, $(0, 2, 0)$, $(1, 3, 1)$ and $(1, 2, 1)$.

For the preemptive scheduling algorithms, the results are illustrated in Figure 5. In Figure 5(a), the sequence of tasks derived by the Earliest Completion First Preemptive algorithm is $(0, 1, 0)$, $(2, 0, 2)$, $(2, 1, 2)$, $(1, 3, 1)$, $(0, 2, 0)$, $(0, 3, 2)$ and $(1, 2, 1)$. Compared with the sequence in the ECF scheme, for example, at the fourth step the ECFP scheme schedules the task $(1, 3, 1)$ instead of the task $(0, 2, 0)$ in the ECF scheme. The trick in this difference is that the ECFP scheme preempts the receive task $(0, 1, 0)$ and utilizes the waiting time to schedule the send task $(1, 3, 1)$.

In Figure 5(b), by preempting receive operations with non-blocking send operations, the Work Racing Preemptive algorithm generates the following task schedule: $(2,0,2)$, $(0,1,0)$, $(0,2,0)$, $(1,3,1)$, $(0,1,2)$, $(1,2,1)$, and $(0,3,2)$. In Figure 5(c), the resulting sequence of tasks generated by the Earliest Available First Preemptive scheme is $(2,0,2)$, $(0,1,0)$, $(1,3,1)$, $(0,2,0)$, $(0,1,2)$, $(0,3,2)$ and $(1,2,1)$. In Figure 5(d), the Round Robin Preemptive algorithm derives the following sequence of tasks: $(2,0,2)$, $(0,1,0)$, $(0,2,0)$, $(1,3,1)$, $(0,1,2)$, $(1,2,1)$ and $(0,3,2)$. In Figure 5(e), the sequence of tasks generated by the Receiver Random Selection Preemptive algorithm is $(0,1,0)$, $(2,1,2)$, $(2,0,2)$, $(1,3,1)$, $(0,2,0)$, $(0,3,2)$ and $(1,2,1)$.

7 Experimental Results

To evaluate the performance of the scheduling algorithms, we developed a software simulator for heterogeneous networks. The simulation is modeled by the number of nodes, the processing speeds of the nodes, the network bandwidth, and the multicast patterns. Since in this paper we do not consider the effect of network contention, we assume the processors in the network are fully connected (each pair of processors can communicate independent of others).

Recall the point-to-point latency between two nodes given in Equation 2 and Equation 3 in Section 2. The send/receive overheads of each processor are chosen randomly from an interval as described below. The constant parts (S_c and R_c) are in the range of $80\mu s$ to $400\mu s$, and the length-dependent parts (S_m and R_m) are in the range of $0.0001\mu s/byte$ to $0.01\mu s/byte$. The network bandwidth between a pair of processors is randomly chosen as either $155Mbps$ (slow links) or $1Gbps$ (fast links). For each data point, the multicast performance was averaged over 100 different system configurations.

We consider three sets of message lengths in our experiments – all small messages ($\leq 1kbytes$), all large messages ($1Mbytes$ and $1.5Mbytes$), and a mixture of small and large messages. In the following, we present and discuss the results for three multicast patterns: single broadcast, all-to-all broadcast, and general multiple multicast.

7.1 Single Broadcast

The broadcast time on a N-node system is measured as follows. We repeat the broadcast for N times and each time a different processor is chosen as the source. The time is measured as the average of the completion time from the N broadcasts.

Figure 6 shows the single broadcast time on different numbers of nodes. We note that the completion time of the FEF algorithm is significantly higher than those of the others. The reason is that FEF only considers the latency of a communication task instead of the completion time of a task.

For single broadcast, all the algorithms generate almost identical schedules, resulting in similar completion time of multiple multicast. Randomized selection (the RRS algorithm) does not perform as well as others, but still outperforms the FEF algorithm.

7.2 All-to-all Broadcast

For the all-to-all broadcast problem, we compare the timing results of the non-preemptive and preemptive algorithms, with different number of nodes (up to 80 nodes), two message sizes ($1kbytes$ and $1Mbytes$), and two network transmission rates ($155Mbps$ and $1Gbps$). Figure 7 and Figure 8 compare the performance of these algorithms in a fast network ($1Gbps$) and a slow network ($155Mbps$) respectively.

In a fast network as shown in Figure 7, for short messages we note that the completion times of the non-preemptive and preemptive versions of ECF, WR, EAF, and RR algorithms are comparable to each other and are all within twice the lower bound. Also shown in Figure 7(a), although the performance of the Receiver Random Selection (RRS) algorithm is not as good as the others, its completion times are still acceptable compared to the FEF algorithm. For large messages, as shown in Figure 7(c), the EAF and RR algorithms perform much worse than the ECF, WR and RRS algorithms.

In Figure 7(b) and (d), the results of the preemptive schemes in a fast network for short and large messages are shown respectively. It can be observed that the completion times of all the five algorithms are comparable with each other. Especially for the EAFP and RRP algorithms with large messages, the preemption optimization provides great improvement over their unoptimized counterparts.

In a slow network, as shown in Figure 8, the performance advantage of the preemptive algorithm is more significant since the network transmission latency becomes more substantial. We also note that the EAF and RR schemes perform more worse with large messages in a slow network. The completion time of the WRP algorithm is within 2.5 times of the lower bound. The WR algorithm is competitive to the ECF algorithm and outperforms ECF in many cases.

The computation (scheduling) times of all these algorithms are measured and listed in Table 1. The WR, EAF, RR and RRS algorithms all have lower scheduling overhead, and the ECF algorithm has the highest, making ECF impractical for dynamic multicast patterns with short messages. The computation times of the preemptive algorithms are about 3 to 5 times as much as those of their non-preemptive counterparts.

7.3 Multiple Multicast

For multiple multicast we consider a 64-node HNOW system, with three different combinations of message lengths. Given the set of source nodes, we randomly choose the destination nodes for each source node. The completion time is taken from the average of 1000 runs of simulation with random configuration of source and destination nodes.

Figure 9 and Figure 10 show the completion times in a fast network and a slow network respectively. Similar to the results of the all-to-all broadcast, the ECF and the WR algorithms perform the best among all non-preemptive algorithms, especially for large messages (Figure 9(c) and Figure 10(c)). The preemptive algorithms all have comparable performances and significant improvement over the non-preemptive ones. Figure 11 and Figure 12 show the *competitive ratio* of the non-preemptive and preemptive algorithms. The *competitive ratio* of an algorithm is defined as the ratio of its completion time against the lower bound (defined in Section 2). It can be observed that the completion times of the WRP algorithm are within 2.5 times of the lower bounds in large systems.

Table 2 shows the computation time of the algorithms on a 64-node HNOW system with different number of source nodes. Compared with the computation times of ECF and ECFP, the non-preemptive and preemptive versions of WR, EAF, RR and RRS algorithms have much lower scheduling time. The overhead is negligible compared to the completion time of the multicast.

8 Conclusion and Future Work

In this paper we have presented six algorithms for software message-passing based implementation of multiple multicast on heterogeneous NOW systems. Our implementations exploit several advantages over existing works: (1) They are portable to different HNOW systems because they are implemented based on the message-passing primitives provided at the application layer of interconnection networks, (2) they are efficient for both general multiple multicast and their degenerate cases (single multicast, broadcast, all-to-all broadcast), and (3) they are developed based on a more realistic, non-blocking communication model, and are optimized to take advantage of non-blocking communication that are available in current networks and operating systems.

Collective communication on heterogeneous systems is a challenging research issue. There are a lot of ground to be covered. In the following we outline some of our future works.

Minimizing Contention in Multicast. The communication model and the scheduling algorithms make an assumption that all the processors in the network are fully connected, so the impact of message contention is neglegible. In today's cluster environments, switches (such as Myrinet [4], Autonet [8], ServerNet [11], and GigaNet) are a common choice for building a cost-effective cluster. A switch usually has eight to thirty-two ports that can interconnect with workstations/PCs or other switches. The processors (workstations and PCs) within a switch are fully connected so each pair of processors can pass messages independent of others. However, message passing between processors residing in different switches will have to travel through the links connecting the switches. If multiple messages travel through a link simultaneously, contention will occur. The contention problem is crucial to the performance of collective communication.

Kesavan et. al [14] A number of research effort has devoted to multicasting with reduced contention on irregular switch-based wormhole networks with unicast message passing [18]. The idea was to find appropriate ordering of the procesing nodes so as to reduce contention. In this line of works, Kesavan et. al [14] proposed the *chain concatenation ordering* heuristic for irregular networks with up-down routing. This algorithm was shown to be the best among three algorithms to implement multicast with reduced contention and minumum latency. Fan and King [9] proposed a two-level multicast algorithm based on the Eulerian Trail of processing nodes in the network. Contention is reduced by finding and reducing a subtrail containing the switches involved in the multicast. However, both works only focus on single multicast on homogeneous swithing networks. Algorithms for reducing contention in multiple multicast have been reported in existing literatures [15, 25, 24]. All of them only focus on homogeneous systems with uniform network topologies, therefore they may not be efficient for heterogeneous systems.

Minimizing contention in multiple multicast on heterogeneous systems is much more complicated. We plan to extend our communication model and scheduling algorithms to implement multiple multicast with reduced contention and minimum completion time. The communication model may be extended by incorporating a parameter measuring the delay caused by message contention. The scheduling algorithms will update the available time of selected sender and receiver according to the new communication model.

Incremental Scheduling. Although the $O(N^3)$ algorithms and the preemption optimiation that we have proposed can determine efficient communication schedules with negligible overhead for multicast of long messages, there is room for improvement for short messages. We are considering an incremental optimization for dynamic multiple multicast patterns. That is, given a multiple multicast pattern P and a good schedule S for it, our goal is to derive a good schedule for a similar pattern P' from S instead of recompute the schedule from scratch.

Collective Communication in Wide Area Networks. We are also investigating the possibility of extending this work to handle collective communication over Wide-Area-Networks (WAN). The first step toward WAN communication is to enhance our communication model with the ability to predict the behavior of communication in WAN. In order to do so, we need to statistically analyze the effect of the cross-traffics from other sessions and the traffic pattern of a communication in order to measure the network transmission time more accurately.

Scheduling with QoS or Constraints. In this paper, the goal of the scheduling algorithms has been to minimize the completion time of multiple multicast. In the future, we will study the possibility of revising these algorithms for collective communication on heterogeneous systems that have quality of service requirement or resource constraints.

References

[1] JV. Bala, Bruck, R. Cypher, P. Elustando, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *Journal of Parallel and Distributed Computing*, 6(2):154–164, February 1995.

[2] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of International Conference on Parallel Processing*, pages 460–467, 1998.

[3] M. Banilazemi, J. Sampathkumar, S. Prabhu, D. K. Panda, , and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *Proceedings of the Heterogenenous Computing Workshop*, 1999.

[4] N. J. Boden, D. Cohen, R. F. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic, and W. Su. Myrinet - a gigabit per second local area network. *IEEE Micro*, pages 29–36, Feb. 1995.

[5] J. Bruck, R. Cypher, P. Elustando, A. Ho, C.-T. Ho, V. Bala, S. Kipnis, and M. Snir. Efficient message passing interface (MPI) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, pages 19–34, January 1997.

[6] A. Chien, S. Pakin, M. Lauria, M .Badanan, K. Hane, and L. Giannini. High performance virtual machines (HPVM): clusters with supercomputing and performance. In *Proceedings of 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1996.

[7] D. Culler et al. Parallel computing on the berkeley now. In *9th Joint Symposium on Parallel Processing*, 1997.

[8] M. D. Schroeder et. al. Autonet: A high-speed, self-configuring local area network using point-to-point links. Technical Report SRC research report 59, DEC, April 1990.

[9] K.-P. Fan and C.-T. King. Efficient multicast on wormhole switch-based irregular networks of workstations and processor clusters. In *Proceedings of Parallel and Distributed Computing Symposium (PDCS)*, 1997.

[10] William Gropp and Ewing Lusk. User's guide for mpich, a portable implementation of mpi. Technical Report ANL/MCS-TM-ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[11] R. Horst. Servernet deadlock avoidance and fractahedral topologies. In *Proceedings of the International Parallel Processing Symposium*, pages 274–280, April 1996.

[12] C. Huang, Y. Huang, and P. K. Mckinley. A thread-based interface for collective communication on ATM networks. In *Proceedings of ICDCS*, pages 254–261, 1995.

[13] Y. Hwang and P. K. Mckinley. Efficient collective operations with atm network interface support. In *Proceedings of ICPP*, 1996.

[14] R. Kesavan, K. Bondalapati, and D. K. Panda. Multicast on irregular switch-based networks with wormhole routing. In *International Symposium on High Performance Computer Architecture*, Feb. 1997.

[15] R. Kesavan and D. K. Panda. Multiple multicast with minimized node contention on wormhole k-ary n-cube networks. *IEEE Transacations on Parallel and Distributed Systems*, 10(4):371–393, April 1999.

[16] M. Lauria. High performance MPI implementation on a network of workstations. Technical Report Master Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1996.

[17] B. Lowekamp and A. Beguelin. ECO: Efficient collective operations for communication on heterogeneous networks. In *Proceedings of International Parallel Processing Symposium*, pages 399–405, 1996.

[18] P.K. McKinley, H. Xu, A. Esfahanian, and L.M. Ni. Unicast-based multicast communication in wormhole-routed direct networks. *IEEE Transactions on Parallel and Distributed Systems*, (5):1252–1265, 1994.

[19] C. S. Raghavendra V. K. Prasanna P. B. Bhat. Efficient collective communication in distributed heterogeneous systems. In *Proceedings of Intnl. Conf. Distributed Computing Systems (ICDCS)*, 1999.

[20] V. K. Prasanna P. B. Bhat and C. S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *Proceedings of the 7th IEEE Intnl. Symposium on High Performance Distributed Computing (HPDC)*, 1998.

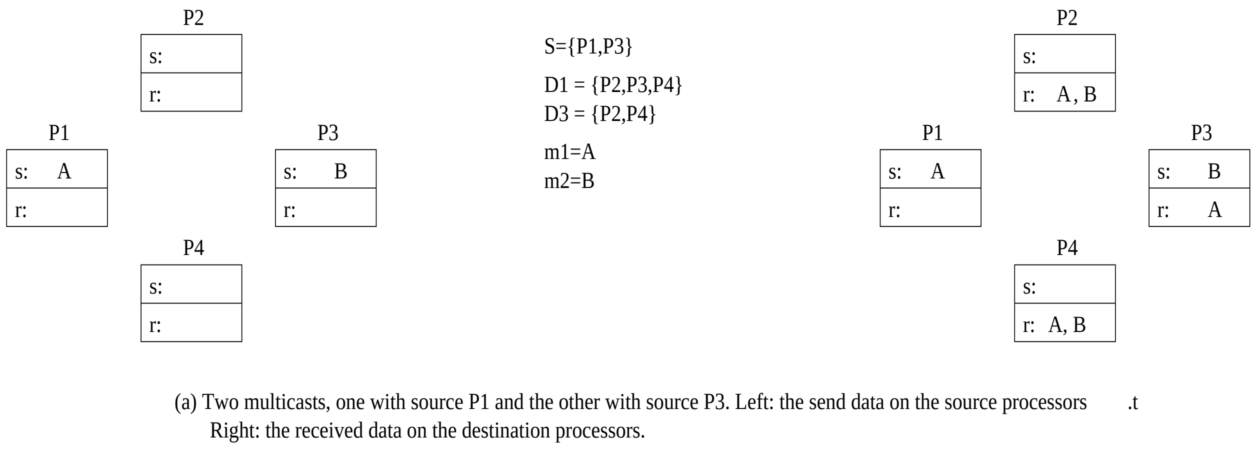
[21] J.Y.L. park, H. A. Choi, N. Nupairoj, and L. M. Ni. Construction of optimal multicast trees based on the parameterized communication model. In *Proceedings of the International Conference on Parallel Processing*, 1996.

[22] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM : An operating system coordinated high-performance communication library. In *Proceedings of High Performance Computing and Networking (LNCS vol. 1225)*, 1997.

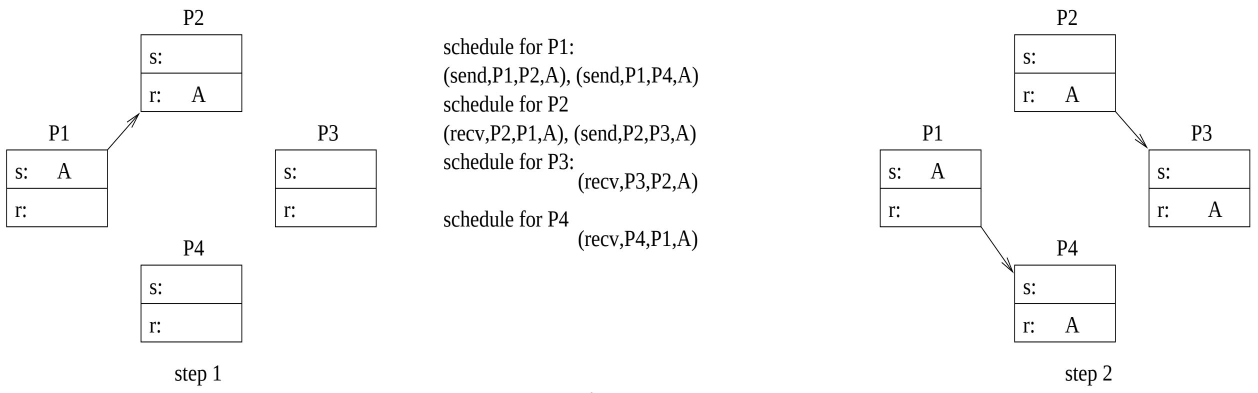
[23] K. Verstoep, K. Langendoen, and H. Bal. Efficient reliable multicast on myrinet. In *Proc. Intl. Conf. Parallel Processing*, volume III, pages 156–165, August 1996.

[24] S.Y. Wang, Y.C. Tseng, C.S. Sheu, and J.P. Sheu. Balancing traffic load for multi-node multicast in a wormhole 2d torus/mesh. In *International Parallel and Distributed Processing Symposium*, pages 611–616, 2000.

[25] Z. Zhou and Z. Tang. Optimal algorithms to reduce contention in multiple multicast. Technical report, Institute of Computing Technology, Chinese Academy o Sciences, 2000.



(a) Two multicasts, one with source P1 and the other with source P3. Left: the send data on the source processors .t
Right: the received data on the destination processors.



(b) Scheduling of the multicast with source P1

Figure 1: An Example of Multiple Multicast

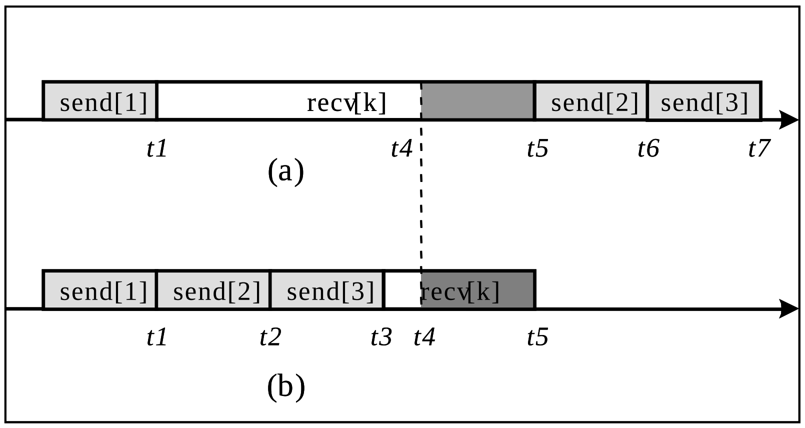


Figure 2: Preemptive Scheduling of Send Tasks

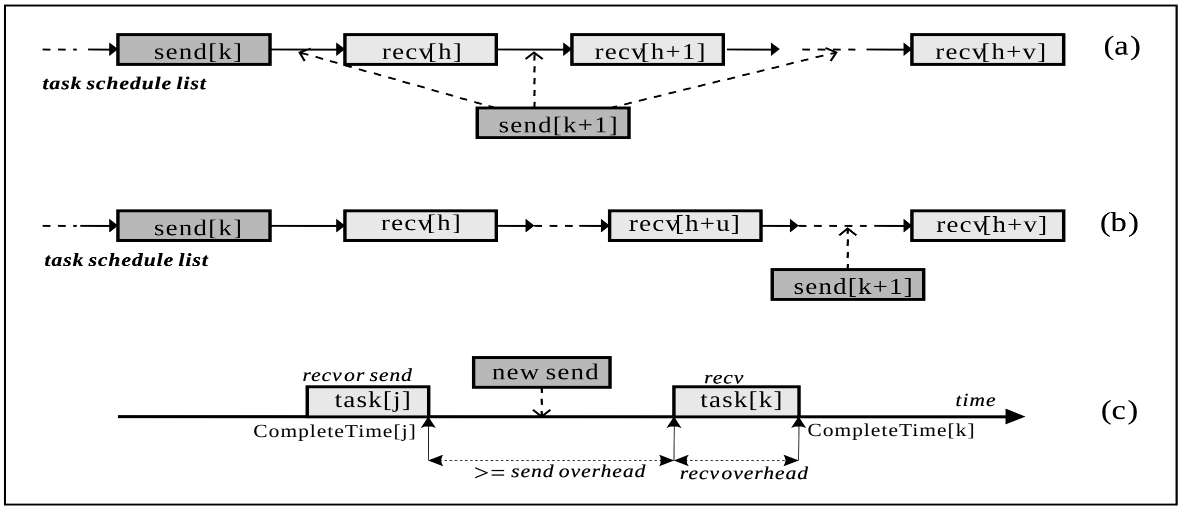


Figure 3: Examples of Preemption Rules

# of nodes	non-preemptive schemes						preemptive schemes				
	FEF	ECF	WR	EAF	RR	RRS	ECFP	WRP	EAFP	RRP	RRSP
8	0.4	0.6	0.2	0.2	0.15	0.22	2	0.3	0.28	0.26	0.29
16	5	17	1	1	1	1	47	2	2	2	3
24	32	121	5	3	4	5	423	14	11	10	13
32	127	504	17	9	9	15	2072	44	29	28	42
40	304	1514	39	17	22	32	7634	117	72	68	103
48	602	3714	58	35	38	66	19585	158	139	130	242
56	810	8168	85	59	67	89	46323	315	250	233	356
64	1033	15871	136	97	104	150	88053	633	426	451	716
72	1494	29026	189	143	160	240	140020	786	672	619	827
80	2568	45597	296	210	243	364	232947	1345	1043	1031	1575

Table 1: The computation times (in μs) of the scheduling algorithms for all-to-all broadcast.

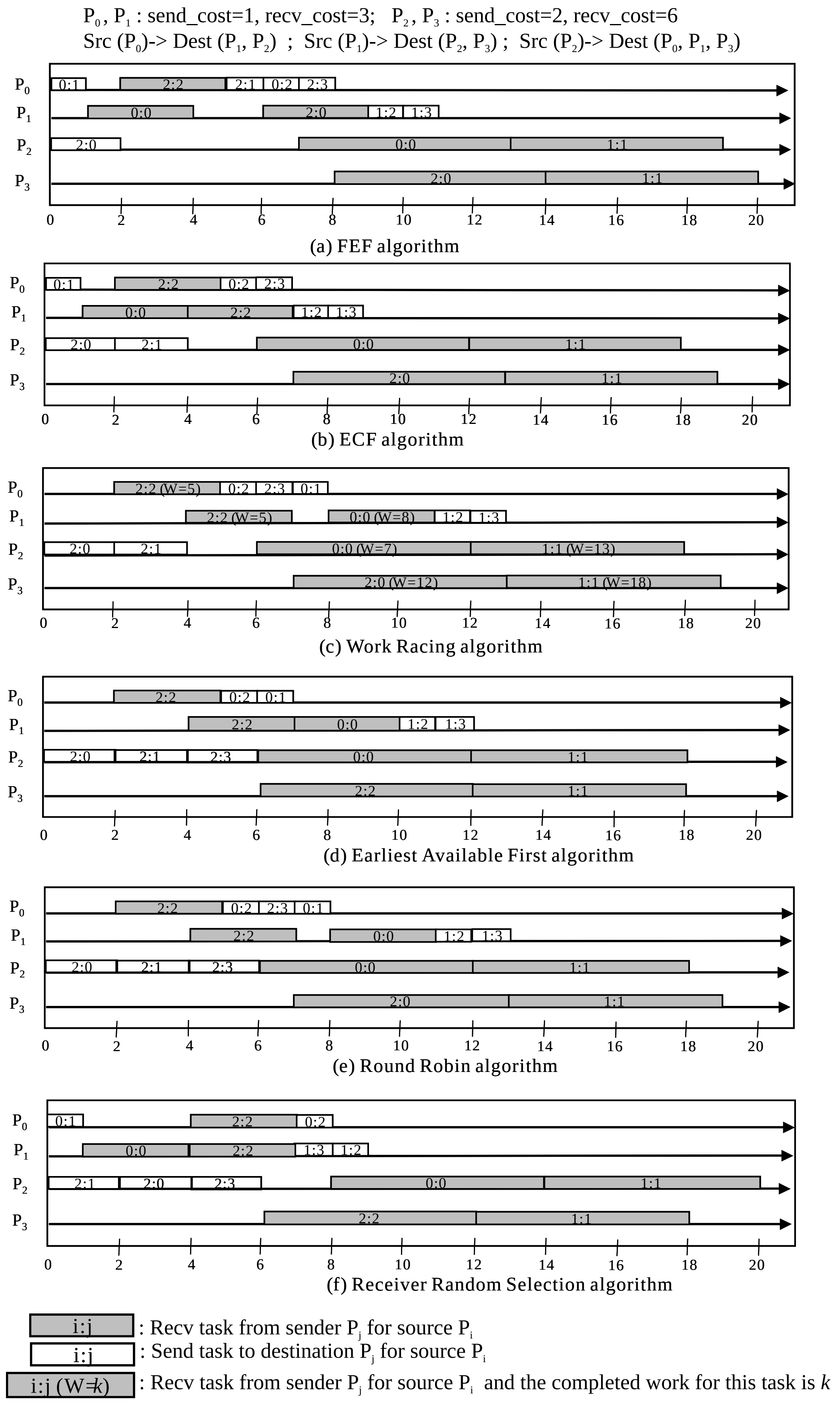


Figure 4: An example of task schedules generated by the non-preemptive algorithms.

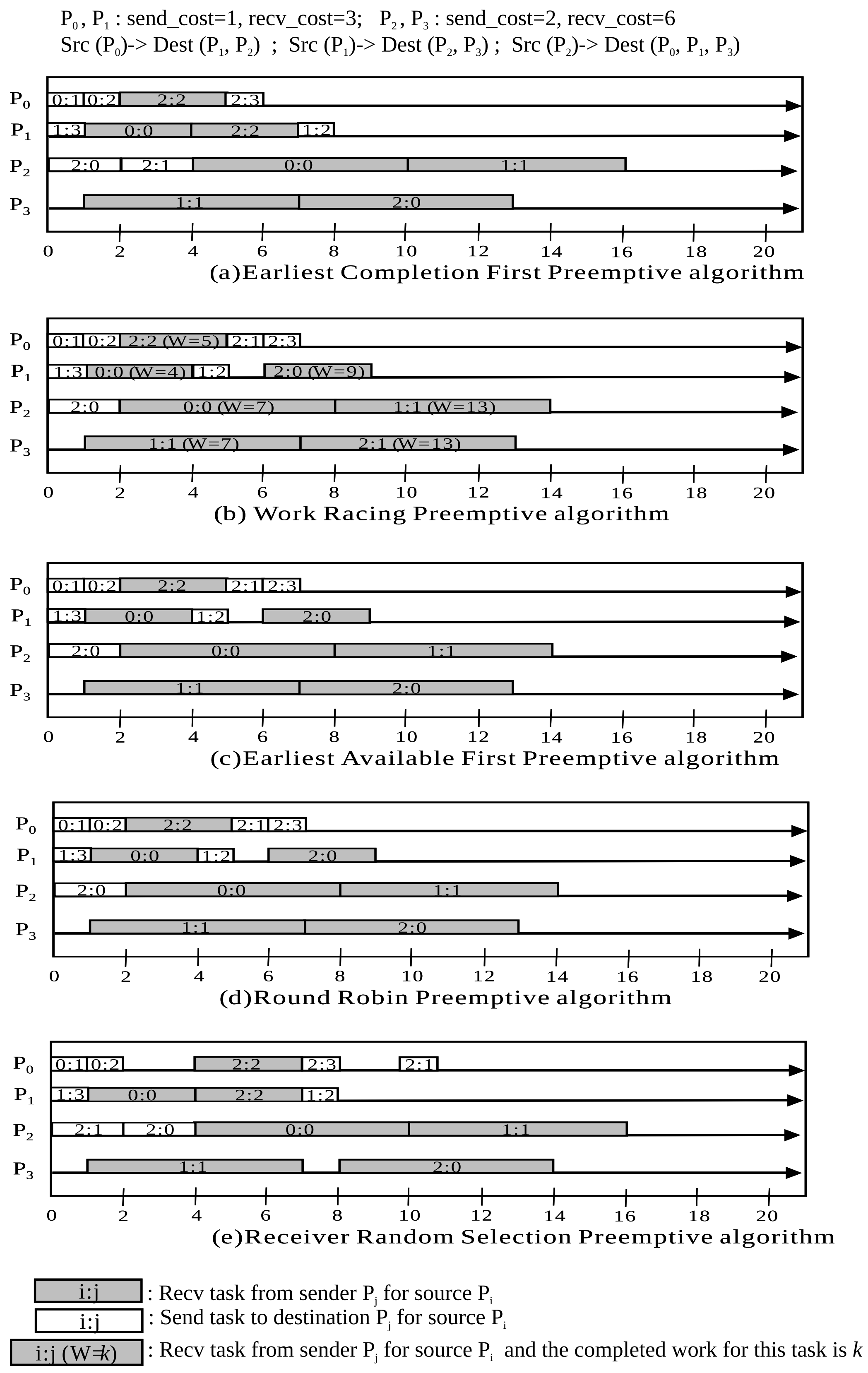
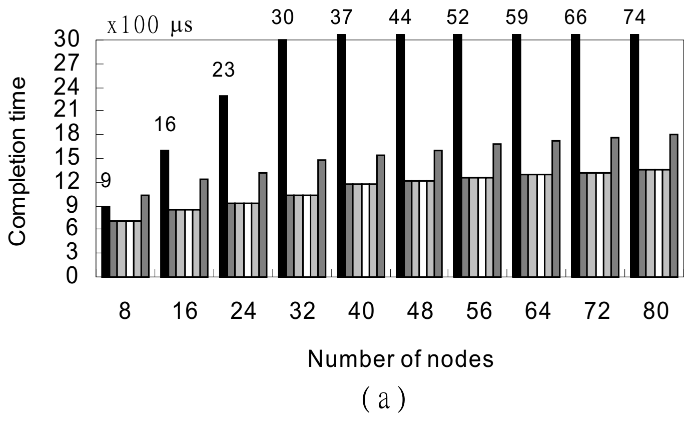


Figure 5: An example of task schedules generated by the preemptive algorithms.



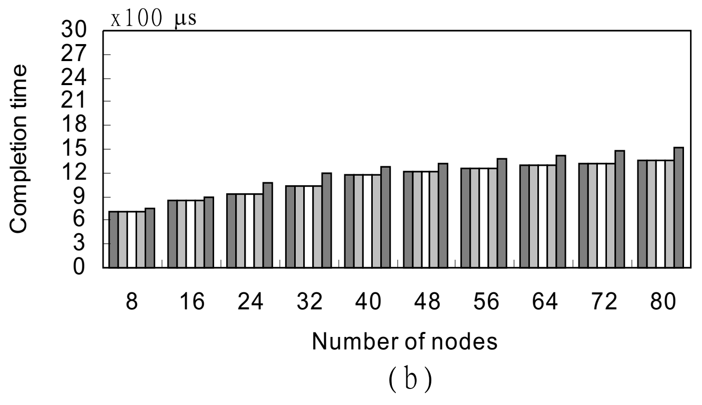
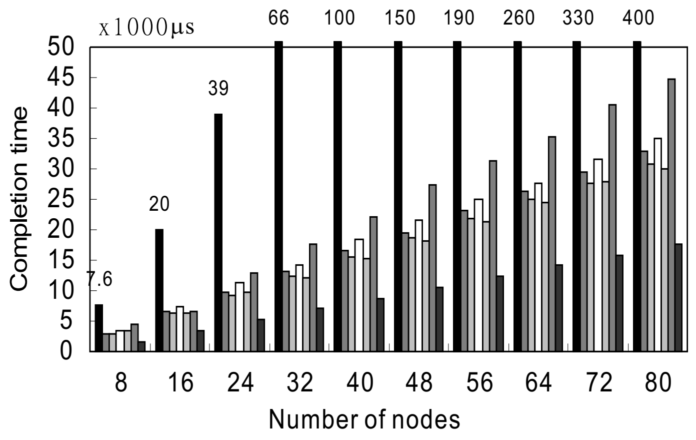
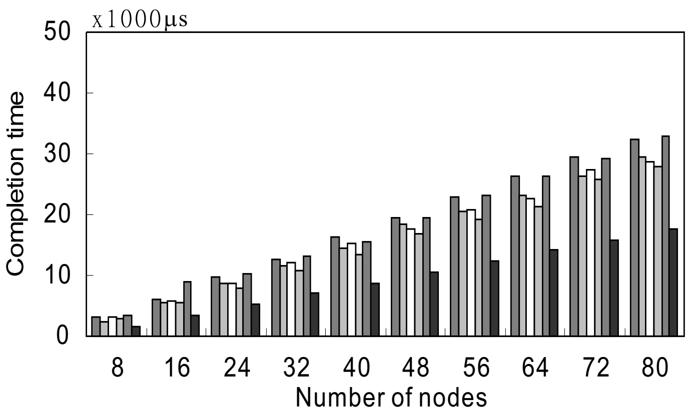


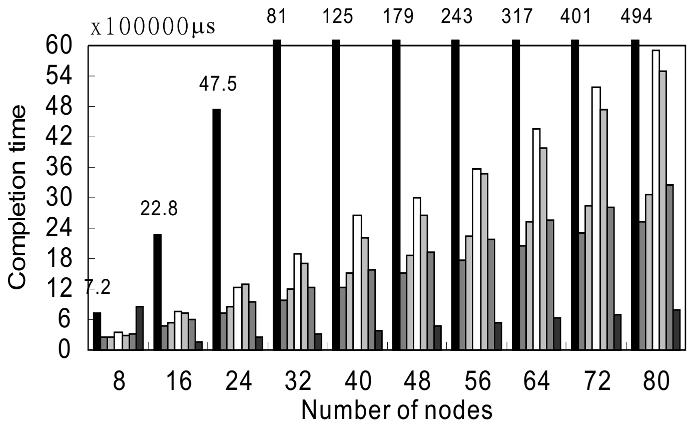
Figure 6: Broadcast completion time on a HNOW system: (a)non-preemptive schemes:from left to right, the results of FEF, ECF, WR, EAF, RR, and RRS are shown. (b)preemptive schemes: from left to right, the results of ECFP, WRP, EAFP, RRP, and RRSP are shown.



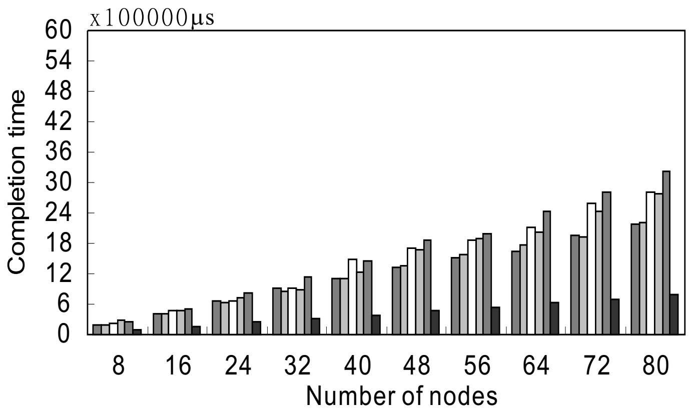
(a) Fast network with small message: (left to right) FEF, ECF, WR, EAF, RR, RRS and Lower bound



(b) Fast network with small message: (left to right) ECFP, WRP, EAFP, RRP, RRSP and Lower bound

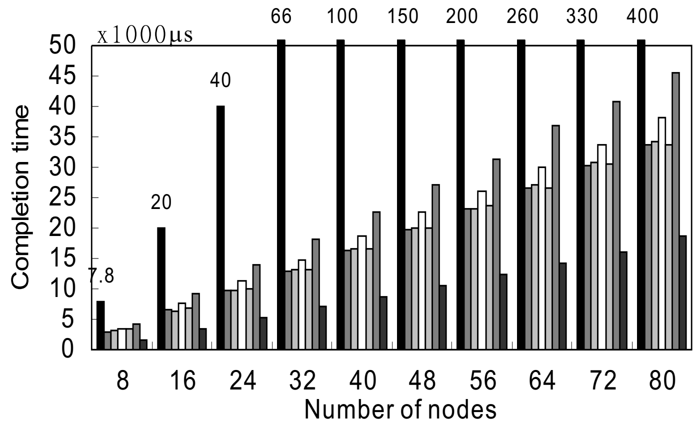


(c) Fast network with large message: (left to right) FEF, ECF, WR, EAF, RR, RRS and Lower bound

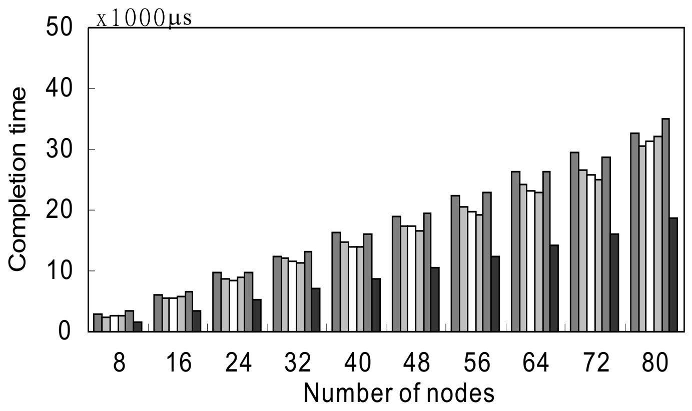


(d) Fast network with large message: (left to right) ECFP, WRP, EAFP, RRP, RRSP and Lower bound

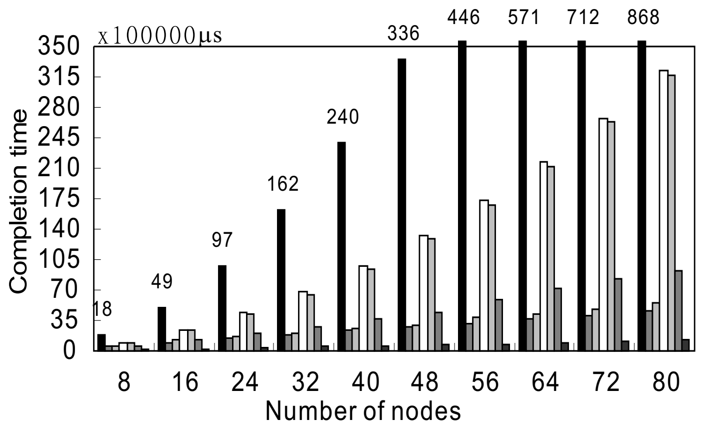
Figure 7: Completion time of all-to-all broadcast in a fast network



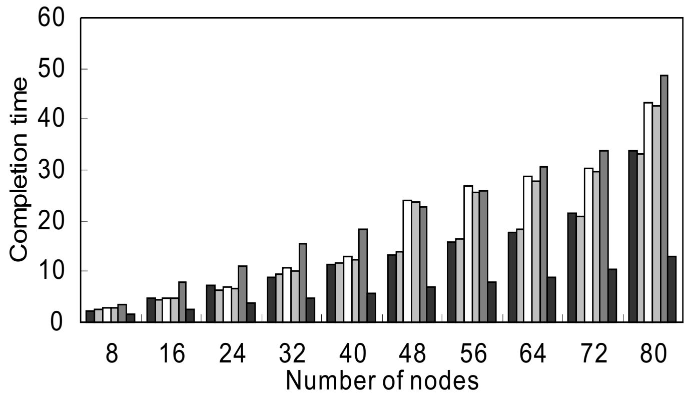
(a) Slow network with small message: (left to right) FEF, ECF, WR, EAF, RR, RRS and Lower bound



(b) Slow network with small message: (left to right) ECFP, WRP, EAFP, RRP, RRSP and Lower bound



(c) Slow network with large message: (left to right) FEF, ECF, WR, EAF, RR, RRS and Lower bound

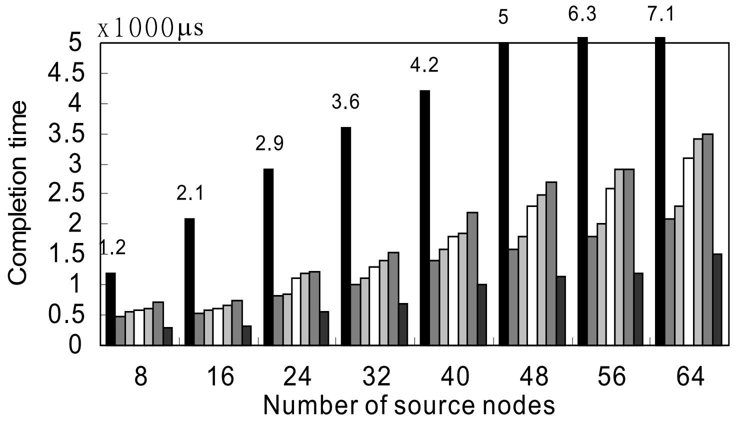


(d) Slow network with large message: (left to right) ECFP, WRP, EAFP, RRP, RRSP and Lower bound

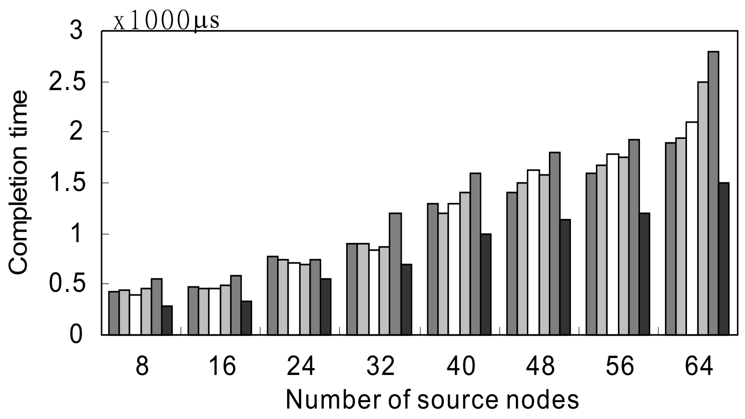
Figure 8: Completion time of all-to-all broadcast in a slow network

# of nodes	non-preemptive schemes						preemptive schemes				
	FEF	ECF	WR	EAF	RR	RRS	ECFP	WRP	EAFP	RRP	RRSP
4	1.35	2.13	0.3	0.27	0.14	0.26	4.25	0.41	0.4	0.27	0.41
8	23.7	50.58	2.15	2.01	1.29	2.15	124.6	3.97	3.72	3.23	3.83
16	44.6	86.2	3.25	3.11	1.98	3.18	226	6.34	6.4	5.2	6.1
24	136	429	10.7	8.8	7.3	10.9	1344	24.8	21.9	20.1	23.8
32	368	836	16.7	12.5	11.1	17.7	2823	39.5	30.7	31.8	42.8
48	639	1611	27.9	20.2	19.8	32.1	6427	76.9	55.5	63.8	72.3
56	973	1956	32.1	23	22.1	39.3	8136	87.4	67.7	73.3	98.7
64	1558	3000	47.2	31.1	29.1	56.4	12548	134.9	102.5	112	157

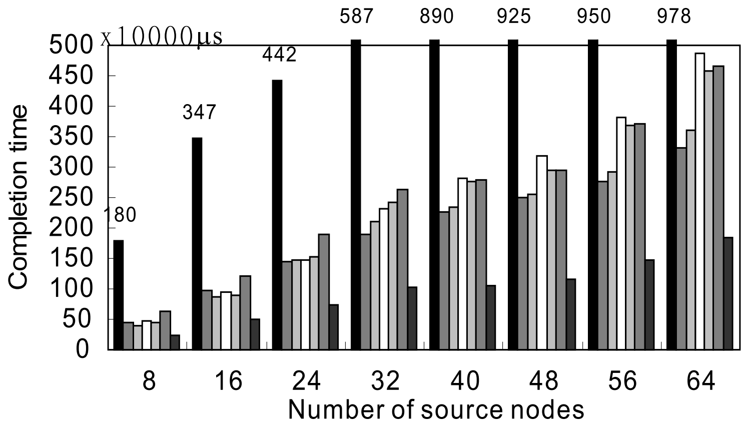
Table 2: The computation times (in μs) of the scheduling algorithms for multiple multicast



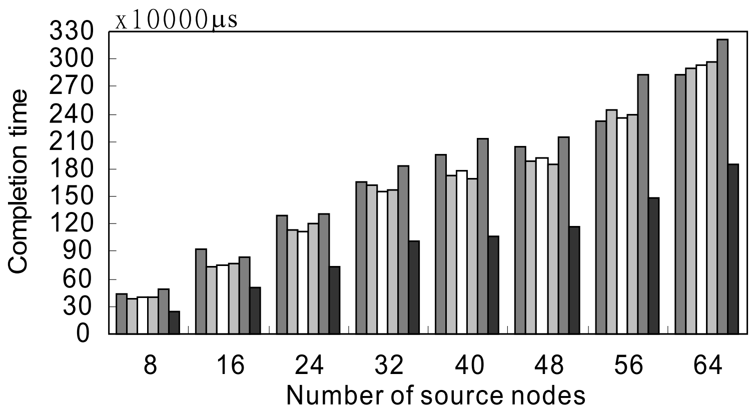
(a)Small messages (left to right): FEF, ECF, WR, EAF, RR, RRS and Lower Bound



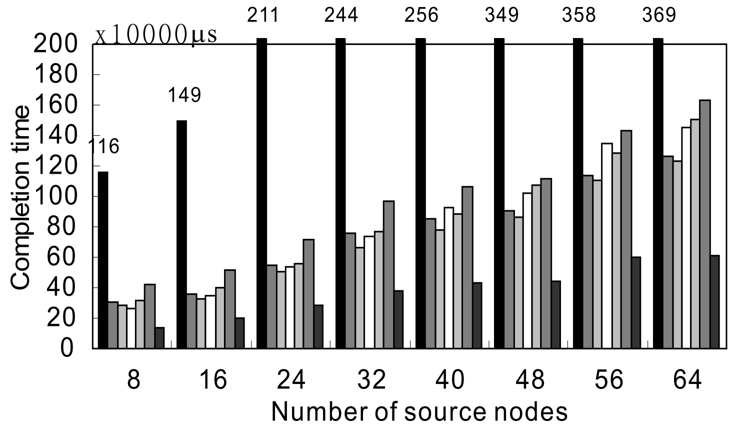
(b) Small messages (left to right): ECFP, WRP, EAFP, RRP, RRSP and Lower Bound



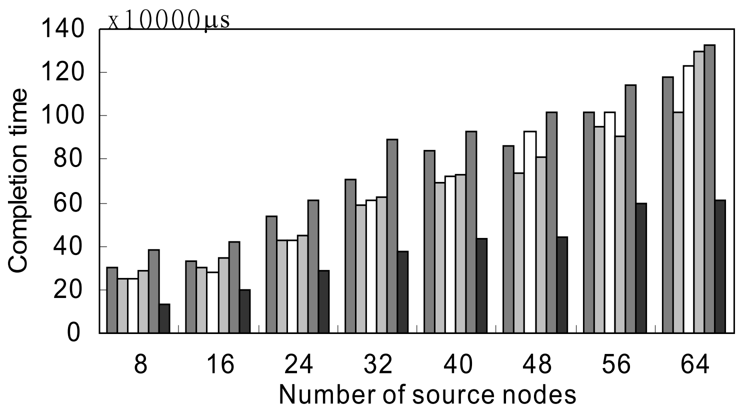
(c) Large messages(left to right): FEF, ECF, WR, EAF, RR, RRS and Lower Bound



(d)Large messages (left to right): ECFP, WRP, EAFP, RRP, RRSP and Lower Bound

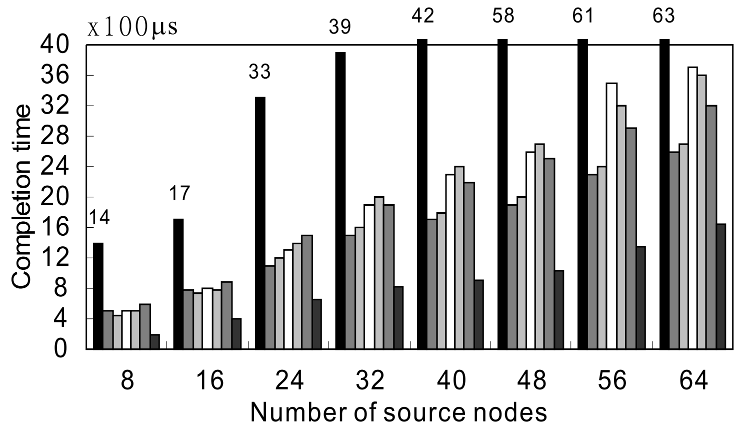


(e)Hybrid messages (left to right): FEF, ECF, WR, EAF, RR, RRS and Lower Bound

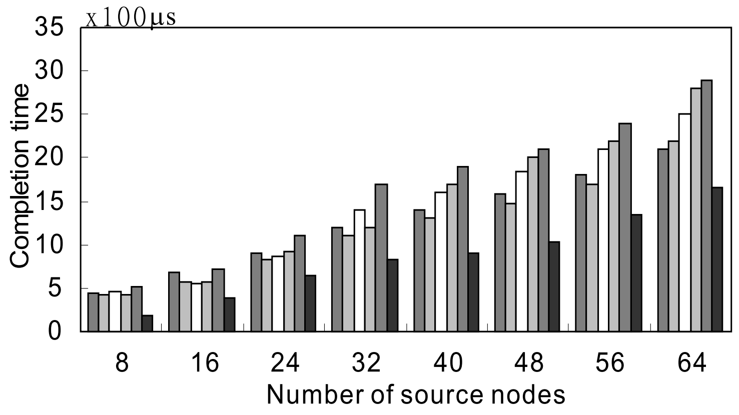


(f) Hybrid messages (left to right): ECFP, WRP, EAFP RRP, RRSP and Lower Bound

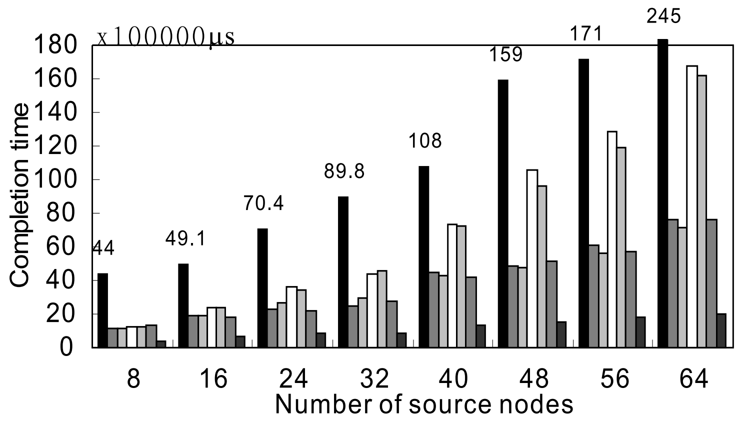
Figure 9: Completion time of multiple multicast in a fast network



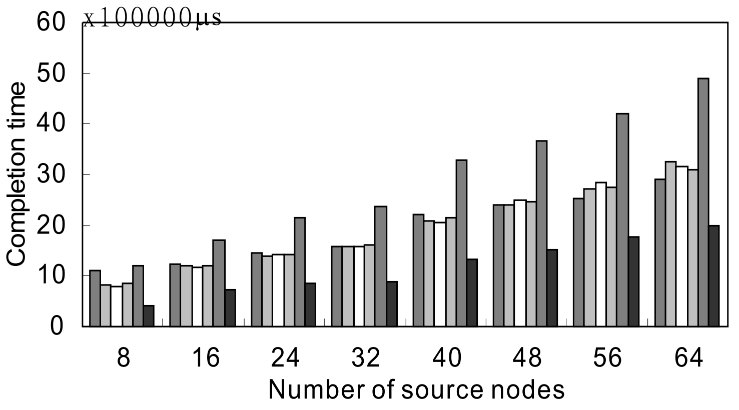
(a)Small messages (left to right): FEF, ECF, WR, EAF, RR, RRS and Lower Bound



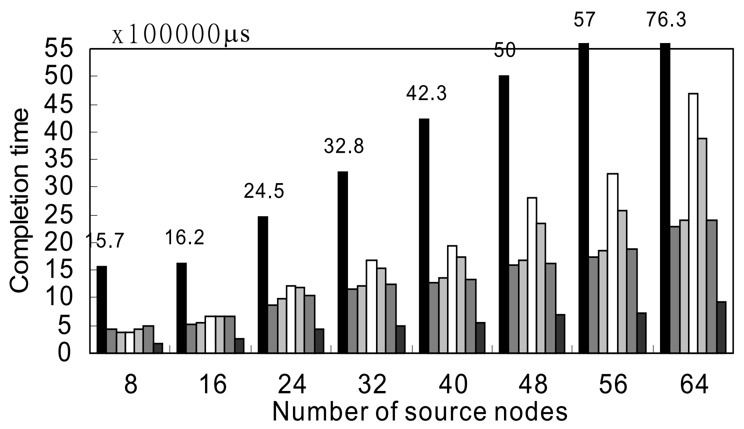
(b) Small messages (left to right): ECFP, WRP, EAFP, RRP, RRSP and Lower Bound



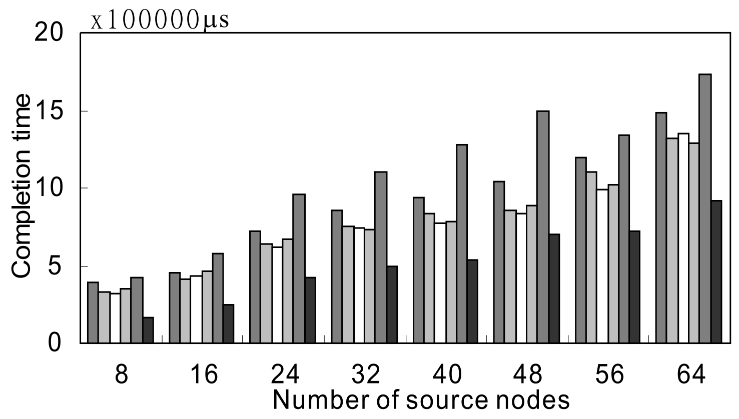
(c) Large messages(left to right): FEF, ECF, WR, EAF, RR, RRS and Lower Bound



(d)Large messages (left to right): ECFP, WRP, EAFP, RRP, RRSP and Lower Bound

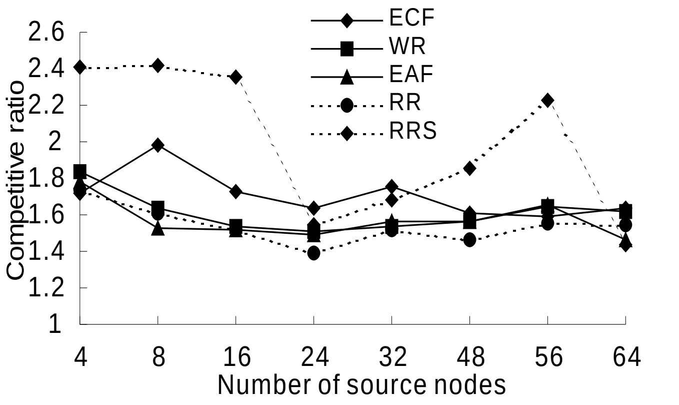


(e)Hybrid messages (left to right): FEF, ECF, WR, EAF, RR, RRS and Lower Bound

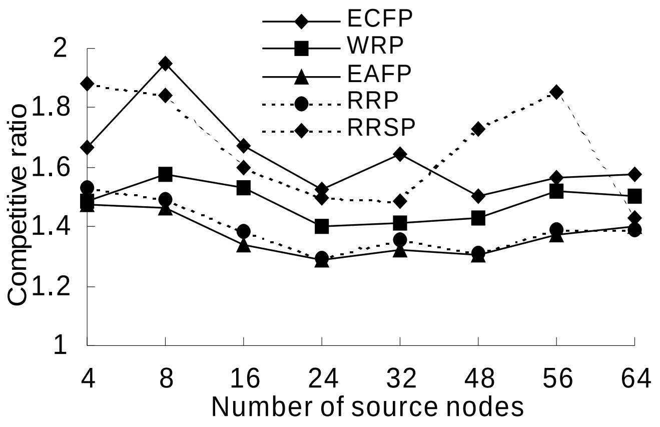


(f) Hybrid messages (left to right): ECFP, WRP, EAFP RRP, RRSP and Lower Bound

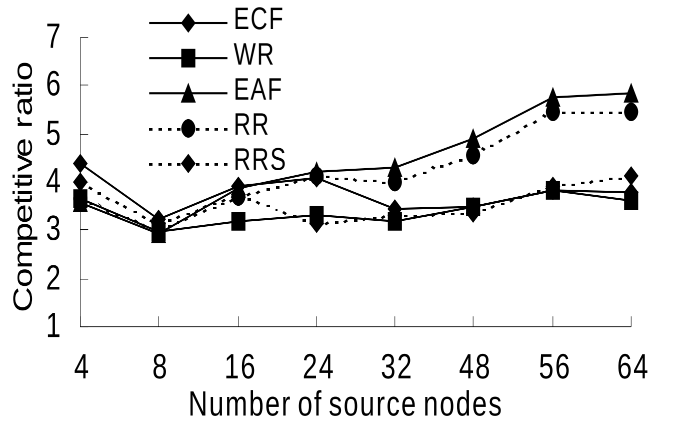
Figure 10: Completion time of multiple multicast in a slow network



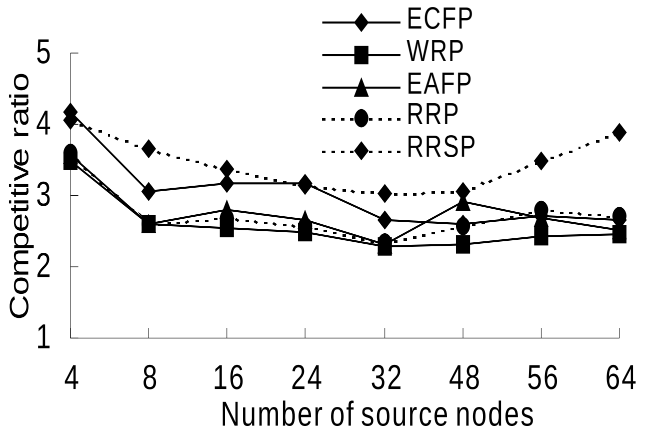
(a) Small message: non-preemptive schemes



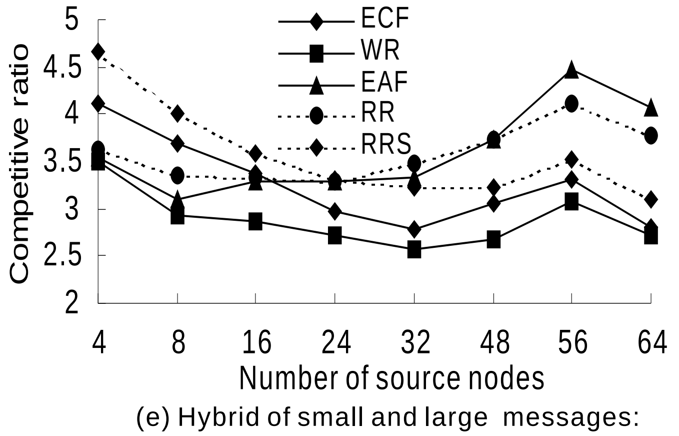
(b) Small message: preemptive schemes



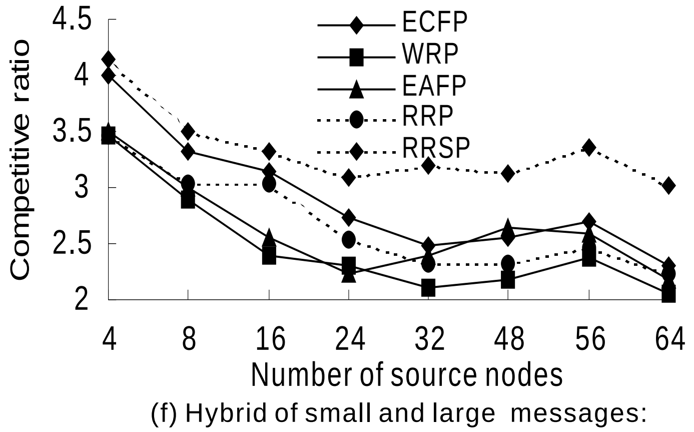
(c) Largel message: non-preemptive schemes



(d) Largel message: preemptive schemes

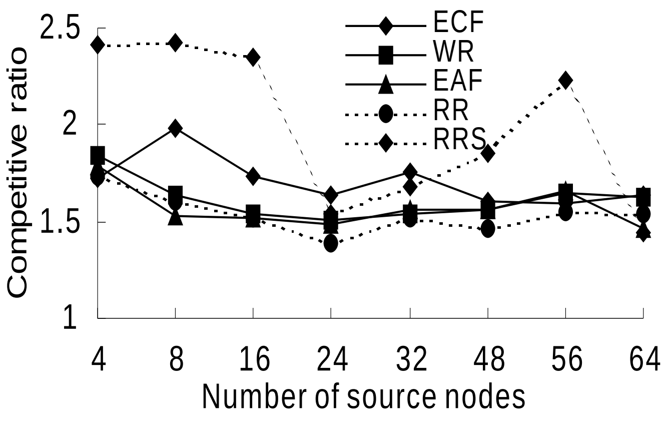


(e) Hybrid of small and large messages: non-preemptive schemes

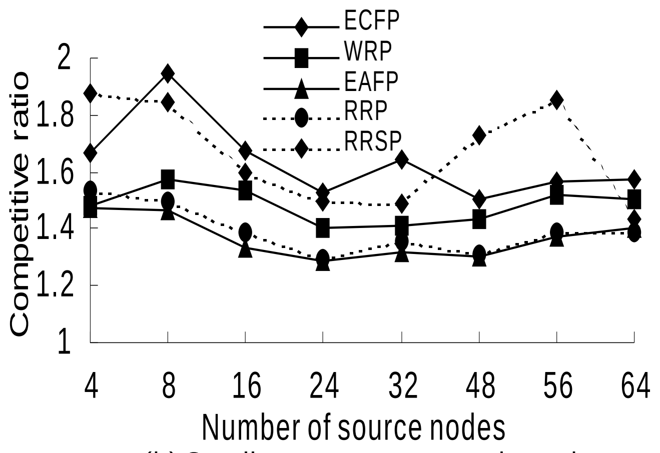


(f) Hybrid of small and large messages: preemptive schemes

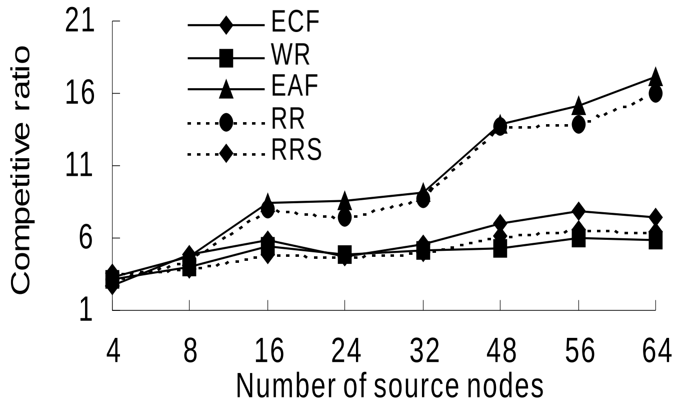
Figure 11: Competitive ratio for the multiple multicast in a fast network



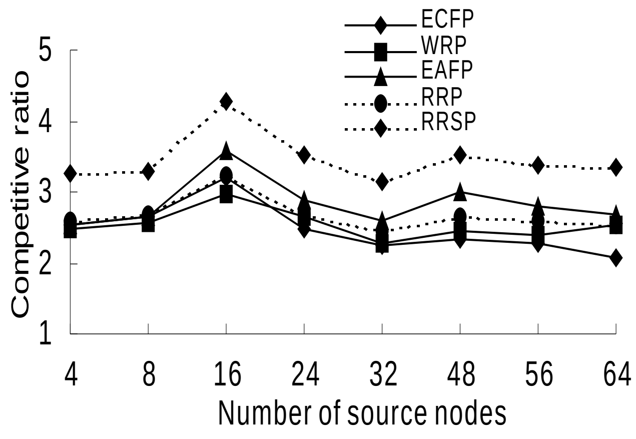
(a) Small message: non-preemptive schemes



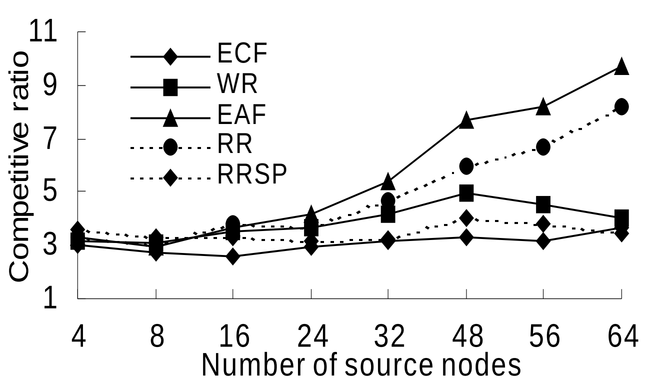
(b) Small message: preemptive schemes



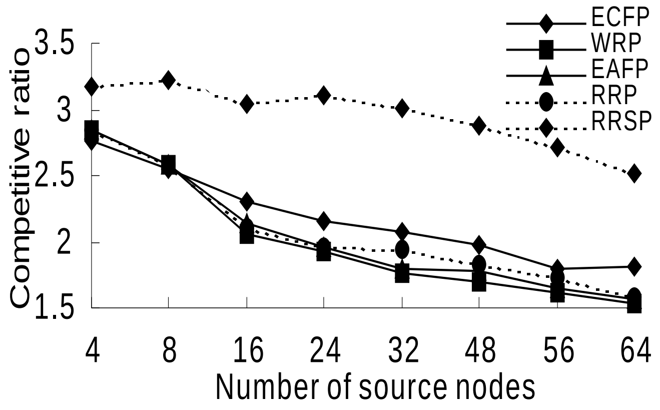
(c) largel message: non-preemptive schemes



(d) large message: preemptive schemes



(e) Hybrid of small and large messages: non-preemptive schemes



(f) Hybrid of small and large messages: preemptive schemes

Figure 12: Competitive ratio for the multiple multicast in a slow network

```
procedure $UpdateSenderState(i,\ j,\ k)$
    $i$ :the index of the sending node.
    $j$ :the index of the receiving node.
    $k$ :the index of the source node.
begin
    $NewTask \leftarrow$ Create task record $task(send,\ j,\ i,\ k)$;
    if $(j = k)$ /*the sender itself is the source node*/
        $TargetTask \leftarrow LastSendTask_j$;
    else{
        $TargetTask \leftarrow$ Get the receive task record whose message
                              source is node $P_k$ from $TaskList_j$;
        if $(TargetTask.CompleteTime < LastSendTask_j.CompleteTime)$
            $TargetTask \leftarrow LastSendTask_j$;
    }


/*Thereafter, the task next to the target task is a receive task or NULL*/
    $NextTask \leftarrow TargetTask.next$;
    $Done \leftarrow$ FALSE;
    while(($NextTask \neq$ NULL)AND($Done =$ FALSE))
    {
        $TimeSpan \leftarrow NextTask.CompleteTime - RecvCost(j,\ MsgSize(NextTask))$
                          $-TargetTask.CompleteTime$;
        if $(TimeSpan < SendCost(j,\ Msg[k]))$
        {
            $tmp \leftarrow NextTask$;
            $NextTask \leftarrow NextTask.next$;
            $TargetTask \leftarrow tmp$;
        }
        else $Done \leftarrow$ TRUE;
    }/* end while loop*/

    $NewTask.next \leftarrow NextTask$;
    $TargetTask.next \leftarrow NewTask$;
    $NewTask.CompleteTime \leftarrow TargetTask.CompleteTime + SendCost(j,\ Msg[k])$;
    $LastSendTask_j \leftarrow NewTask$;
end
```

Figure 13: UpdateSenderState procedure

```
procedure $UpdateReceiverState(i,\ j,\ k,\ CompleteTime)$
    $i$ :the index of the receiving node.
    $j$ :the index of the sending node.
    $k$ :the index of the source node.
    $CompleteTime$ :the completion time of the new receive task.
begin
    $NewTask \leftarrow$ Create task record $task(recv,\ i,\ j,\ k)$;
    $NewTask.CompleteTime \leftarrow CompleteTime$;
    Append $NewTask$ to the tail of $TaskList_i$;
    $LastRecvTask_i \leftarrow NewTask$;
end
```

Figure 14: UpdateReceiverState procedure

```
function $ReceiveAvailable(i)$
    $i$ :the index of the receiving node.
begin
    if $(LastSendTask_i.CompleteTime < LastRecvTask_i.CompleteTime)$
        return $LastRecvTask_i.CompleteTime$;
    else
        return $LastSendTask_i.CompleteTime$;
end
```

Figure 15: ReceiveAvailable function

```
function $SendAvailable(j,\ k)$
  $j$ :the index of the sending node.
  $k$ :the index of the source node.
begin
  if $(j = k)$ /*the sender itself is the source node*/
    $TargetTask \leftarrow LastSendTask_j$;
  else{
    $TargetTask \leftarrow$ Get the receive task record whose message
                  source is node $P_k$ from $TaskList_j$;
    if $(TargetTask.CompleteTime < LastSendTask_j.CompleteTime)$
      $TargetTask \leftarrow LastSendTask_j$;
  }



/*Thereafter, the task next to the target task is a receive task or NULL*/
  $NextTask \leftarrow TargetTask.next$;
  $Done \leftarrow$ FALSE;
  while(($NextTask \neq$ NULL)AND($Done =$ FALSE))
  {
    $TimeSpan \leftarrow NextTask.CompleteTime - RecvCost(j,\ MsgSize(NextTask))$
                $-TargetTask.CompleteTime$;
    if $(TimeSpan < SendCost(j,\ Msg[k]))$
    {
      $tmp \leftarrow NextTask$;
      $NextTask \leftarrow NextTask.next$;
      $TargetTask \leftarrow tmp$;
    }
    else $Done \leftarrow$ TRUE;
  }/* end while loop*/

  return $TargetTask.CompleteTime$;
end
```

Figure 16: SendAvailable function