Combined Routing and Scheduling of Concurrent Communication Traffic in Hypercube Multicomputers

Bing-rung Tsai and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
Email: {iast,kgshin}@eecs.umich.edu

Abstract— *We propose and evaluate low-complexity, low-overhead schemes for distributed message scheduling and routing in binary hypercube multicomputers equipped with a hardware communication adapter at each node. The goal is to optimize the network performance not only for steady traffic flow, but also for concurrent bursty traffic. We comparatively evaluate the performance of different scheduling-routing combinations for several switching methods, such as message switching, circuit switching and virtual cut-through. The evaluation results have indicated that in case of heavy transient traffic, a partially-adaptive routing scheme, when combined with an appropriate message-scheduling policy, can outperform a fully-adaptive routing scheme.*

1 Introduction

As pointed out in [1], the critical component of a multicomputer is its interconnection network. Many algorithms are communication rather than processing limited. Some fine-grain concurrent programs execute as few as ten instructions in response to a message. To execute such programs efficiently, the communication network must be able to handle heavy concurrent traffic. Message or traffic routing/scheduling in multicomputer interconnection networks has received considerable attention. Most of the existing work has assumed inter-node communication traffic to be composed of a set of steady, independent flows. Network throughput or mean message latency over a certain period of time is often used as a performance measure. This type of performance evaluation reflects more of the "steady-state" behavior of a network. However, in the task level — where a computation task is decomposed into a set of communicating modules — intermodule traffic, and hence interprocessor communication when the modules are assigned to different processors, tends to be bursty. A large number of messages are often generated within a short span of time. Furthermore, delivering each of these messages may not be mutually independent. For instance, an algorithm may not continue its execution until the partial results are collected from, or exchanged among, the participating modules.

The work reported in this paper was supported in part by the Office of Naval Research under grants N00014-92-J-1080 and N00014-91-J-1115, and by the NSF under Grant MIP-9203895. Any opinions, findings, and recommendations in this paper are those of the authors and do not reflect the views of the funding agencies.

The work described in this paper differs from others in several ways: (1) it stresses the importance of combining low-complexity adaptive routing with message scheduling as opposed to complex, fully-adaptive routing; (2) it zooms on the case of bursty traffic where a network is congested with concurrent communication traffic; and (3) it deals with the transient (rather than just steady-state) performance of a network.

We define a *communication mission*, or *mission* for short, to be a set of messages to be exchanged among the nodes such that the completion of delivery of these messages as a whole is crucial to the completion of the whole task. The *makespan* of a mission denotes the time span since the generation of the first message until the last message reaches its destination. (A formal definition of makespan will be given in Section 2.) In the execution of parallel algorithms, such as parallel state-space-search [2], parallel sorting [3] and parallel Fourier-Transform [4], their communication behavior in a multicomputer network can often be characterized as a series of communication missions. In the case of missions containing heavy concurrent traffic, the network may become congested and turn into a bottleneck of execution efficiency.

Our goal is to improve communication efficiency by implementing low-complexity, distributed message scheduling and routing in a distributed-memory system equipped with a communication adapter like SPIDER [5] at each node. We will focus on the case where SPIDER is configured to work in k-ary n-cubes in general, and binary n-cubes in particular. Large-buffer message switching and virtual cut-through as well as circuit switching will be considered for the hypercube topology. Wormhole switching is found to be more suitable for high-radix low-dimension hypercube networks (such as meshes) with multiple virtual channels, and hence, covered in a separate paper [6].

The paper is organized as follows. Necessary notation and definitions are introduced in Section 2. In Section 3, the proposed scheduling policies and routing algorithms are described. Section 4 deals with the performance evaluation of various scheduling–routing combinations. This paper concludes with Section 5.

2 Notation and Definitions

The following notation will be used throughout the paper.

- $(T, T + \Delta t]$: a *mission time frame* or time window during which messages of a mission arrive. Without loss of generality, we will assume $T = 0$ from now on.
- M: the number of messages arrive in $(0, \Delta t]$.
- m_i: message i of length ℓ_i, $0 \le i < M$.
- t_i^a: the arrival time of m_i, i.e., the time when m_i is ready to be sent.
- t_i^c: the completion time of m_i, i.e., the time when m_i reaches its destination.

For a set of M messages $\{m_i, 0 \le i < M\}$, each with arrival time t_i^a, and completion time t_i^c, the *makespan* of this set, $\hat{t}$, is defined as $\max_{0 \le i < M} \{t_i^c\} - \min_{0 \le i < M} \{t_i^a\}$. Given this set of messages to be exchanged among the nodes of a Q_n, we want to minimize this set's makespan and hence maximize link utilization of the network. Depending on the system implementation, we may achieve this goal with various combinations of message scheduling and routing.

A network with a mission arrival can be viewed as a physical system with a certain amount of injected *energy*, i.e., the total communication bandwidth required for the mission. $\hat{t}$ is essentially the time span required to *dissipate* the injected energy. Given the same mission, different message scheduling–routing combinations will affect the rate the energy is dissipated in the system, thus resulting in different $\hat{t}$ values. Measuring the network performance using this approach is analogous to evaluating the transient response of an electronic component to impulse-function inputs. Similar to the case with an electronic circuit, a network that performs well under this kind of load will perform well for most of the time under a steady-state condition.

3 Proposed Schemes

In this section we will describe several message scheduling policies and an adaptive routing algorithm that can be implemented with the aid of a SPIDER-like communication adapter. Note that in the presence of concurrent communication traffic, the system must use a traffic control mechanism that is efficient and doesn't cause excessive overhead. The schemes introduced here don't modify the underlying message format, and can all be implemented with minimal hardware, or, as in SPIDER, with a simple microprogram.

3.1 Scheduling

We will evaluate the performance of several distributed message scheduling policies, while focusing on non-preemptive, low-complexity policies which utilize the existing message format. Each message has a *length field* as well as a *destination field*. The routing controller on a node can use the information in these two fields to determine which message to be sent first over each link of the node. The only major overhead comes from the implementation of a priority queue for each link, which can be implemented on a SPIDER-like adapter or by gate-level logic circuit. The time overhead can be ignored since in such an adapter, message queueing and communication can be done in parallel for non-preemptive scheduling.

In the default, FIFO, policy, no length or destination information is used, and the message at the front of a FIFO queue will be transmitted over a link. Also, in any of the following priority scheduling policies, a tie is broken by the FIFO principle. With the shortest-first (SF), and longest-first (LF) policies, messages in the queue of a link are arranged according to their lengths. When a message arrives, and the outgoing link it requests is busy, it is entered into a priority queue. SF gives a shorter message higher priority, while LF awards higher priority to a longer message. The nearest-first (NF) and farthest-first (FF) policies take the destination field of a message and calculate the Hamming distance of the current node to the destination of the message. Entries in the priority queue are arranged in the order of increasing (as in NF) or decreasing (as in FF) Hamming distances.

The *Remaining Bandwidth Requirement* (RBR) of a message on a node is defined as the product of the length of the message and the remaining Hamming distance to its destination. The *Smallest-RBR-First* (SBF) policy gives higher priority to smaller RBR messages. The *Largest-RBR-First* (LBF) policy gives priority to larger RBR messages. Implementing these two policies induces a slightly more overhead than the other policies due to a multiplication operation required before each message is entered into a queue. Again, with a SPIDER-like adapter, this overhead can be ignored since queueing and routing can be done in parallel.

Note that all of the above scheduling policies can be combined with the *earliest due-date* (EDD) scheduling of message transmissions. With EDD scheduling, starvation can be avoided when a certain type of messages arrives continually. For example, with the LBF policy, short messages may be blocked indefinitely if long messages arrive continually. This can be remedied by giving earlier arrivals higher priority.

The effectiveness of a scheduling policy can be measured by comparing its performance to that of optimal schedules. An optimal schedule is obtained by assuming that the following parameters are known *a priori*: t_i^a, ℓ_i, and the path each message is routed through. Thus, we have a classical scheduling problem in this case. For example, when message switching is used, by treating each link as a "processor", and each message as a "job", the optimization problem is essentially a special case of the job-shop scheduling problem, which is NP-hard. For other switching methods such as circuit switching and virtual cut-through, there are no equivalent known scheduling problems, but their computational complexity is also NP-hard. A branch-and-bound algorithm is used to find optimal schedules. At each node in the search tree, we calculate an estimated bound to decide which node to be expanded next. The efficiency of the algorithm depends on the accuracy of the estimated bounds. In [7], we developed algorithms and equations to be used for computing bounds which are then used for finding optimal schedules for message switching, circuit switching, and virtual cut-through.

3.2 Routing

One major drawback of the *e*-cube routing algorithm is that the path chosen between a pair of processors is fixed regardless of the network traffic condition. When a certain link leading toward the destination of a message is busy, the *e*-cube algorithm simply holds the message and waits until the link becomes free, even if there can be an alternative free path. In situations where interprocessor communication traffic is light, this may not cause any serious performance degradation. However, under heavy traffic, it can become a major performance bottleneck. With distributed adaptive routing, a processor can select an alternative (free) link to route a message when the originally-selected link is busy. By doing this, it is possible that link utilization and network efficiency can be greatly enhanced.

There are numerous adaptive routing algorithms proposed for hypercubes. Each scheme has its own advantages and disadvantages. More complex routing algorithms such as the DFS (Depth-First-Search) routing algorithm [8] require extra fields in each message to avoid livelock and achieve the backtracking capability. The major advantage of such algorithms is that all possible paths between source and destination nodes will be explored. However, their disadvantages include the overhead for storing information on detouring and backtracking, and more complex routing controllers. Furthermore, the information stored in a message can become out-of-date before it reaches the destination. For example, at a certain intermediate node, the routing algorithm may find all links that lead the message closer to its destination are busy, and hence decide to take a detour, i.e., a non-shortest path. But when the message is routed via a detour, one of the shortest paths may become available. This can result in a waste of network bandwidth, and hence degrade the performance. Also, unpredictable path lengths in this routing algorithm can result in inaccuracies in calculating the RBR of a message, and therefore reduce the effectiveness of bandwidth-sensitive message scheduling policies such as SBF and LBF.

On the other hand, if we restrict the routing algorithm to the shortest paths only, albeit the full connectivity of the network is not explored, the above shortcomings can be avoided. Besides, in a network equipped with SPIDER-like adapters, virtually no additional overhead is added to the default e-cube routing algorithm, since no modification needs to be done on the original message format, and the routing controller can be easily programmed with low-complexity code. Also, it is easy to have it work in tandem with any message scheduling policies since the length of each path is predictable. Characteristics of this distributed routing algorithm, called the *Progressive Adaptive* (PA) algorithm, are described as follows.

- As in *e*-cube routing, PA tries to route messages from the lowest dimension to the highest dimension based on the results of exclusive-ORing the source and destination addresses.

- When the link corresponding to the lowest dimension is busy, the next lowest dimension is tested, and so on, until a free link is found to route the message. If no link leading the message closer to its destination is available, the message is blocked and entered into a queue.
- Only one message queue is maintained on each node. All blocked messages are entered into the queue. Their order in the queue is determined by the underlying scheduling policy. When a link becomes free, the message closest to the head of the queue that can use the link to move closer to its destination is routed through this link.

The performance of the PA algorithm will be compared with DFS routing and a centralized path-selection (CPS) algorithm. As in the case of finding optimal schedules, the CPS algorithm operates on the premise that accurate message lengths are known *a priori*, and assigns paths to messages to balance the traffic in the network. With concurrent communication traffic, $\hat{t}$'s are usually dominated by the most heavily-loaded link. The problem of selecting a path in order to minimize $\hat{t}$ is equivalent to minimizing the maximum link load, and can thus be formulated as a mini-max optimization problem. This is a special case of Decision Problem 1 in [9], which was shown to be NP-hard. However, the true optimal solutions may not be meaningful since $\hat{t}$ is eventually determined by scheduling messages. Therefore, our goal is to find sufficiently good solutions that, when combined with the branch-and-bound scheduling algorithm, can achieve near-optimal performance. The CPS algorithm used is based on the simulated annealing method and is described in [7].

4 Performance Evaluation

When the number of messages being sent concurrently into a network becomes larger, their interactions make the network behavior too complicated to predict and analyze. This calls for simulations to assess the performance of various scheduling–routing combinations. Our simulation model is summarized as follows:

- We assume processors — including their routing controllers — as well as communication links to be fault-free.
- Each communication link is half-duplex, i.e., at any instant of time, only one message can be sent in either direction of a link.
- As was supported in SPIDER, the routing controller on each processor can send or receive multiple messages at the same time, provided the links needed are not in use. Also, incoming message buffering/queueing and outgoing message transmission are done in parallel for all switching methods.
- For circuit switching, the "call signal" for establishing a circuit is transmitted out-of-band, and doesn't interfere with the existing communication traffic.
- Δt is relatively small, i.e., the communication traffic is highly concurrent. As a result, we assume there can be at most one message sent from node i to node j, $i \neq j$, within Δt. If there are more than one message, they will be combined into a long message. We use "*density*" to denote the probability that one processor sends a message to another processor in a mission. A higher value of *density* means heavier traffic. In the uniformly-distributed traffic pattern, ℓ_i is generated by the following routine:

```
**for** $i := 0$ **to** $M - 1$
    **for** $j := 0$ **to** $M - 1$
        **if** $i \neq j$ **and** $rand1 < density$ **then** $\ell_i := rand2$;
        **else** $\ell_i := 0$;
```

  In the above pseudo code, *rand*1 is a uniformly-distributed random number in the interval $[0, 1]$, and *rand*2 is a normally-distributed random number with $\mu = 10$, $\sigma = 5$ in obtaining most of our numerical results. It can be seen that as *density* and σ vary, the generated communication patterns extend over a wide range. In the *hot-spot* traffic pattern, the routine is similar except that the traffic is directed only to a given number of hot spots in a hypercube.

- In all of the presented data, we set $\Delta t := 0$. The results with $\Delta t > 0$ were found to be similar with the case of $\Delta t = 0$ combined with lower *density* values.
- Each data point is obtained by averaging the results of 10,000 iterations. The hypercube dimension used is 4. Results of this problem size are found to be typical among all tested sizes and are therefore selected for presentation.

4.1 Scheduling

Table 1 shows typical evaluation results of various scheduling polices. The data shown is obtained with *density* = 0.95 for uniformly-distributed traffic, i.e., a highly-congested condition. It is obvious that LBF, with performance very close to the optimal schedules (OPT), outperforms the other policies.

	MS	CS	VCT
OPT	193.1	193.7	193.0
FIFO	231.1	229.0	218.5
LF	216.8	221.4	202.5
SF	240.5	233.7	221.4
FF	199.9	217.3	208.7
NF	244.4	232.7	219.5
LBF	195.8	208.2	195.3
SBF	250.1	231.5	226.6

Table 1: Performance of scheduling policies under the e-cube routing algorithm.

Our simulation results in evaluating the various scheduling policies are summarized as follows:

- SF, NF, SBF are all worse than FIFO under all three switching methods, i.e., message switching (MS), circuit switching (CS), and virtual cut-through (VCT). Typically, SBF has the worst performance among all the scheduling policies considered. It should be noted, however, that this is not the case with wormhole switching. As we have shown in [6], SF, NF and SBF outperform LF, FF and LBF in a virtual-channel network with wormhole switching.

- FF has the performance characteristics closest to LBF. In most situations FF outperforms LF, especially when the variance of message length is small. In message and circuit switching, FF generally outperforms LF, meaning that the distances to destinations have more pronounced effects. It is found that only under virtual cut-through, LF has performance closer to LBF when the variance of message length is large. But as σ gets smaller, it gradually becomes closer to, and eventually gets outperformed by, FF.

- Under circuit switching, the performance differences are smaller for all distributed scheduling policies. The difference between the best and the worst are only $\approx$ 25 time units. Since messages never get buffered under circuit switching, distributed scheduling policies can only determine the order of sending messages at the source. It is expected that a distributed scheduling policy is less effective than a centralized policy in this case.

- Under virtual cut-through, messages are buffered only when a cut-through attempt fails, so the effects of distributed scheduling policies lie between those of message switching and circuit switching. However, given the same scheduling policy, virtual cut-through has consistently shown the best performance among the three switching methods. It is also interesting to note that as shown in Table 1, in case of heavy concurrent traffic, with a better scheduling policy such as LBF, message switching can approach the performance of virtual cut-through.

- In all three switching methods, network performance generally gets better if message lengths are closer to uniform, i.e., σ is small.

In Figs. 1 to 3, the performance of LBF scheduling is compared with the optimal schedules and FIFO scheduling under various *density* value for uniformly-distributed traffic. Under message switching, for *density* > 0.3, and in most cases of hot-spot traffic, LBF improves significantly over FIFO and produces schedules whose makespans are within 10% of the optimal schedules. Its performance is only slightly degraded under very light traffic, where all distributed scheduling policies become less effective and degenerate into FIFO scheduling. Performance of LBF under virtual cut-through is very similar to the case under message switching, which is predictable since under heavy concurrent traffic, virtual cut-through essentially degenerates into message switching. In circuit switching, all distributed scheduling policies are less effective than in other switching schemes since messages are never buffered and are scheduled only at the source nodes. Nevertheless, LBF can still approach within 12% of optimal schedules for *density* > 0.5. The only case where LBF does not improve significantly over FIFO is under circuit switching and hot-spot traffic.

4.2 Routing

The performance of PA routing combined with the LBF scheduling is compared against the DFS routing working with the LBF scheduling, the e-cube routing algorithm with FIFO message schedules (EQ-FIFO), and the CPS centralized path selection algorithm with optimal message schedules (CPS-OPT). The results are plotted in Figs. 4 to 6, for uniformly-distributed traffic.

Our simulation results indicate that under all three switching methods, PA-LBF significantly improves over the e-cube routing in all cases and outperforms DFS-LBF in cases of heavy concurrent traffic. In most situations, it also approaches the performance of CPS-OPT closely within 12% in the message switching case, and in cases of circuit switching and virtual cut-through, within 7%.

It is only under light traffic condition ($density < 0.4$) or a very small number of hot spots (< 4) that the DFS routing has an advantage over the PA routing. It can approach CPS-OPT under very light uniformly-distributed traffic, and outperform CPS-OPT in case of a very small number of hot spots. However, under heavy traffic, the DFS algorithm does not fare much better than the e-cube routing, especially under message switching. This indicates that detouring and backtracking in the DFS algorithm have negative effects on network performance when the network is heavily congested. Besides, with the unpredictability of path lengths of the DFS routing, LBF scheduling becomes less effective and nearly degenerates into FIFO scheduling.

The DFS algorithm performs significantly better under circuit switching or virtual cut-through than in the case of message switching. In fact, virtual cut-through with the DFS routing essentially degenerates into circuit-switching in our simulations. With both of these switching methods, the DFS algorithm either successfully routes a message to the destination, or blocks the message at the source. Therefore, there is no bandwidth waste in detouring and backtracking. While in the case of message switching where a message can be blocked at any node in the network, taking a detour and blocking a message at a node farther from its destination can degrade the performance. However, in implementing the DFS algorithm with circuit switching or virtual cut-through, the information on detouring and backtracking must be stored in the header used for establishing the path to avoid livelock. This can become a major overhead for large networks.

From the above results, one can conclude that, to improve network efficiency under heavy concurrent traffic, shortest-path adaptive routing combined with the LBF message scheduling policy can approach the performance of near-optimal centralized routing and scheduling. Also, it is a more cost-effective alternative than a high-complexity fully-adaptive routing algorithm such as the DFS routing.

4.3 Steady-State Performance

Here we compare the performance of scheduling and routing mechanisms under steady-state traffic arrivals. Message arrivals at each node are assumed to follow a Poisson process. The mean latency of messages over a period of 20,000 units of time is plotted versus the mean message inter-arrival times. When scheduling messages, the *ages* of messages are also taken into account and combined with either FIFO or LBF scheduling policies. That is, "older" messages are given higher priority to minimize the mean message latency and avoid starvation on continual message arrivals.

In Figs. 7 to 9, the performance of PA-LBF, DFS-LBF and EQ-FIFO are compared. Centralized schemes such as CPS-OPT are not included because their high computation cost makes them unsuitable for "on-line" applications in which message arrivals are continual. The steady-state results indicate that a scheme that performs well in transient conditions also performs well in a steady-state situation. Also, as in the transient case, the DFS algorithm outperforms the PA algorithm only under light traffic conditions.

5 Concluding Remarks

We have dealt with the problem of optimizing interprocessor-communication performance in a hypercube multicomputer equipped with SPIDER-like adapters under concurrent traffic. Branch-and-bound algorithms have been developed to find optimal schedules for various switching methods under the non-adaptive e-cube routing algorithm. Though computationally expensive, these optimal schedules serve to measure the effectiveness of various scheduling policies. A centralized path selection algorithm based on the simulated annealing method is also developed in Appendix A and serves as a reference for evaluating distributed routing algorithms.

Several distributed message scheduling policies under the e-cube routing algorithm were examined for systems with message switching, circuit switching, and virtual cut-through. In our simulations, the Largest remaining Bandwidth First (LBF) scheduling policy was found to be very effective. It could approach the performance of optimal schedules in many situations, and being a distributed scheduling policy, it was also more practical and computationally much less expensive than centralized approaches.

A low-complexity adaptive routing algorithm, called the Progressive Adaptive (PA) algorithm, was also evaluated. When combined with the LBF scheduling policy, this routing algorithm was found to be very effective in improving performance over the e-cube algorithm. It outperformed the more complex DFS routing in heavy traffic conditions, and could closely match the performance of centralized, near-optimal approaches.

Though we evaluated network performance under transient communication loads, it was shown that an improvement in transient performance almost always offered better steady-state performance.

References

[1] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. on Computers*, vol. 39, no. 6, pp. 775–785, June 1990.

[2] S. Anderson and M. C. Chen, "Parallel branch-and-bound algorithms on the hypercube," in *Hypercube Multiprocessors*, pp. 309–317, 1987.

[3] T. Tang, "Parallel sorting on the hypercube concurrent processor," in *Proc. of the 5th Distributed Memory Computing Conference*, pp. 237–240, April 1990.

[4] L. Desbat and D. Trystram, "Implementing the discrete Fourier Transform on a hypercube vector-parallel computer," in *Proc. of the 4th Distributed Memory Computing Conference*, pp. 407–410, March 1989.

[5] J. W. Dolter, S. Daniel, A. Mehra, J. Rexford, W.-C. Feng, and K. G. Shin, "SPIDER: Flexible and efficient communication support for point-to-point distributed systems," in *Proc. of the 14-th Int'l Conf. Distributed Computing Systems*, May 1994.

[6] B.-R. Tsai and K. G. Shin, "Sequencing of concurrent communication traffic in mesh multicomputers with virtual channels," in *Proc. of the 23rd International Conference on Parallel Processing*, August 1994.

[7] B.-R. Tsai, *Mapping and Scheduling of Concurrent Communication Traffic in Multicomputer Networks*, PhD thesis, The University of Michigan, 1994.

[8] M. S. Chen and K. G. Shin, "Depth-first search approach for fault-tolerant routing in hypercube multicomputers," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 152–159, April 1990.

[9] D. D. Kandlur and K. G. Shin, "Traffic routing for multicomputer networks with virtual cut-through capability," *IEEE Trans. on Computers*, vol. 41, no. 10, pp. 1257–1270, October 1992.

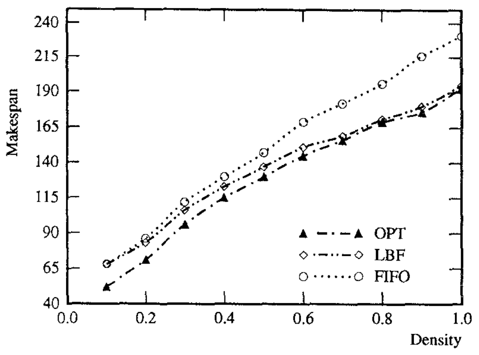

Figure 1: Uniformly distributed traffic, message switching.

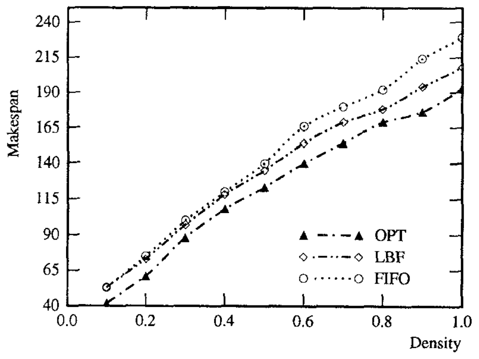

Figure 2: Uniformly distributed traffic, circuit switching.

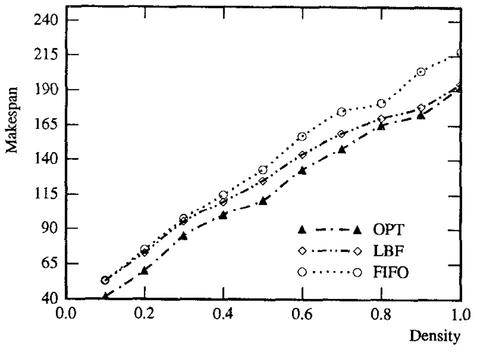

Figure 3: Uniformly distributed traffic, virtual cut-through.

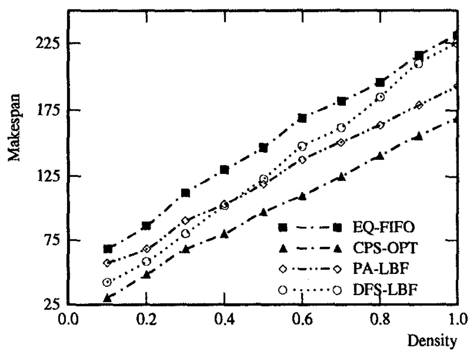

Figure 4: Performance comparison under message switching, uniformly distributed traffic.

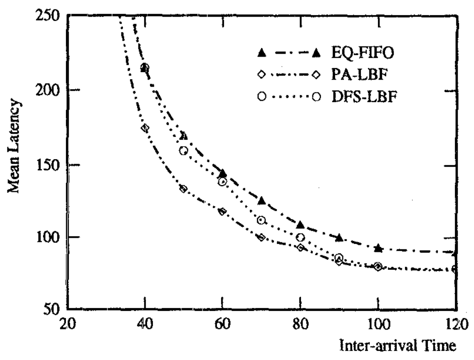

Figure 7: Steady-state performance under message switching.

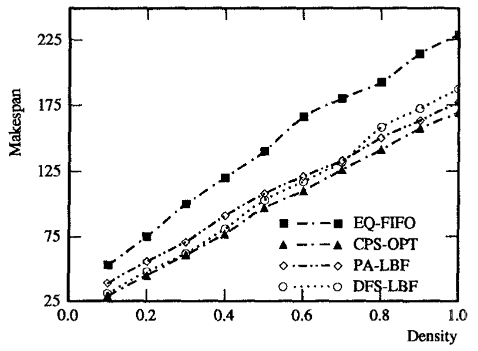

Figure 5: Performance comparison under circuit switching, uniformly distributed traffic.

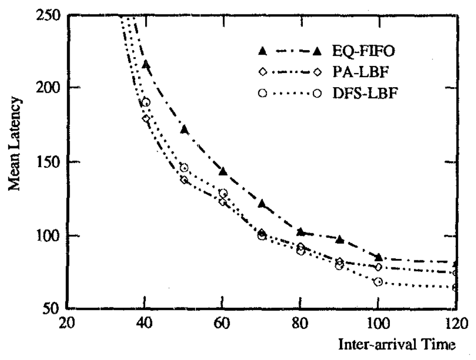

Figure 8: Steady-state performance under circuit switching.

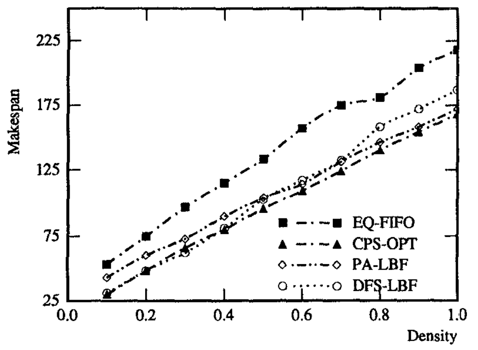

Figure 6: Performance comparison under virtual cut-through, uniformly distributed traffic.

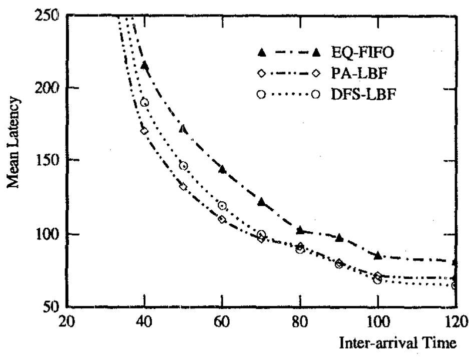

Figure 9: Steady-state performance under virtual cut-through.