NONUNIFORM TRAFFIC SPOTS (NUTS) IN MULTISTAGE INTERCONNECTION NETWORKS

Tomas Lang
Lance Kurisaki

January 1988
CSD-880001

Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks

Tomas Lang and Lance Kurisaki

Computer Science Department

University of California, Los Angeles

Abstract

The performance of multistage interconnection networks with blocking switches is degraded when the traffic pattern produces nonuniform congestion in the switches, that is, when there exist nonuniform traffic spots (NUTS). For some specific patterns we evaluate this degradation in performance and propose modifications to the network organization and operation to reduce the degradation. Successful modifications are the use of diverting switches and the extension of the network to include alternate paths. The use of these modifications to the basic blocking policy for control of contention makes the network more effective for a larger variety of traffic patterns.

Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks

Tomas Lang and Lance Kurisaki
Computer Science Department
University of California, Los Angeles

Abstract

The performance of multistage interconnection networks with blocking switches is degraded when the traffic pattern produces nonuniform congestion in the switches, that is, when there exist nonuniform traffic spots (NUTS). For some specific patterns we evaluate this degradation in performance and propose modifications to the network organization and operation to reduce the degradation. Successful modifications are the use of diverting switches and the extension of the network to include alternate paths. The use of these modifications to the basic blocking policy for control of contention makes the network more effective for a larger variety of traffic patterns.

1. Introduction

Multistage interconnection networks (MIN) are used in multiprocessor systems to connect processors with other processors or with memory modules. These networks provide a compromise between networks of low latency and high cost, such as the crossbar, and networks of high latency and low cost, such as the shared bus. Moreover, MINs can be pipelined to provide a bandwidth comparable to that of the crossbar for suitable traffic patterns. In addition, the control of routing is simple. A large body of work has been done on the structure, operation, and performance of these networks; a comprehensive reference is [SIEG85]. These networks were initially introduced for use in array computers of the SIMD type; in this context the interconnection networks are sometimes called permutation networks. More recently, they are being proposed and used in multiprocessors of the MIMD type, especially of the shared-memory variety [HWAN84, GOTT83, PFIS85, RoMa86, THOM86]. In this paper we are concerned with this second type of use.

A more extensive discussion of the operation and performance of multistage networks is given in the next section. In their basic form, these networks provide a unique path between any source-destination pair. However, the paths for different pairs are not disjoint and, therefore, conflicts might occur when simultaneous communication is established between several source-destination pairs. The basic method used to handle this problem is to use a packet-switched type of operation and to buffer the packets in the switches. Blocking occurs whenever the buffers become full.

It has been shown that the performance of these networks is satisfactory for uniform traffic [DiJu81, KrSn83], that is, for traffic in which the destinations are generated by a random variable with uniform distribution. More recently, several studies [PfNo85] have indicated that the performance of the network is degraded significantly when the traffic includes hot-spot traffic, that is, when each source generates a larger fraction of the traffic to one particular destination. This type of traffic occurs because of access to shared variables, such as semaphores. To overcome this degradation, a network with combining switches has been proposed.

The topic of this paper is a more general type of nonuniform traffic, in which there is no concentration of the traffic to one destination, but the traffic is not uniformly distributed among the switches, producing nonuniform traffic spots (NUTS). In this paper, we illustrate some typical cases of this type of traffic and show the degradation in network performance produced by them. We then explore solutions to reduce this performance degradation.

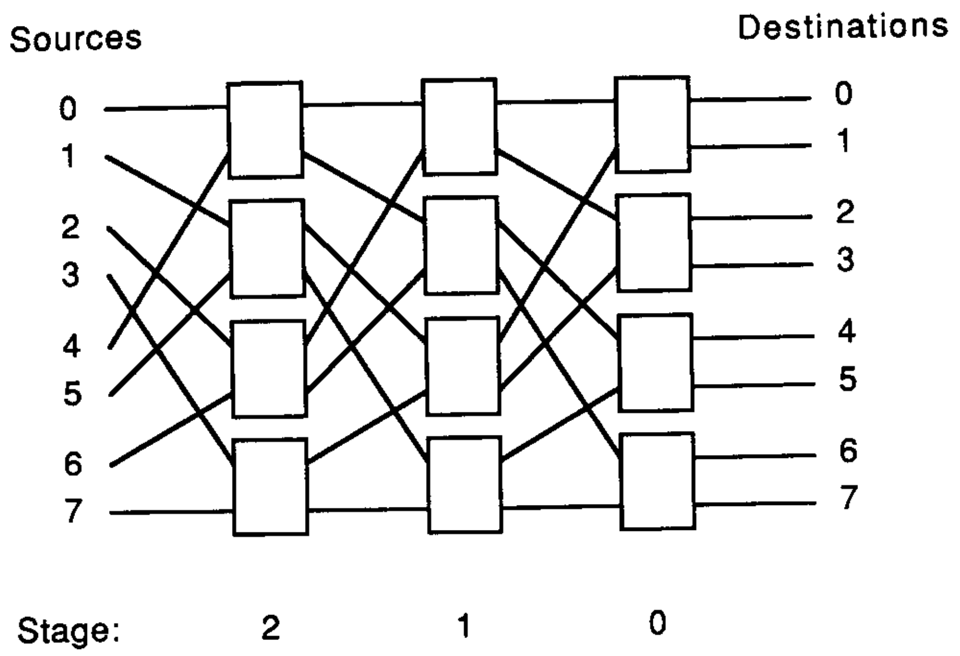
Of course, in this case the use of combining switches is not a solution since the contention packets do not necessarily have the same destination. We show that randomization of the traffic, proposed for reducing contention in multicomputers [VALI82, MITR86], is not suitable either. As positive alternatives to improve the performance, we consider the use of diverting switches, with several diverting policies, and networks with alternate paths. Because of the reduction in degradation produced, the proposed modifications to the basic network with blocking make the multistage network suitable for a larger variety of multiprocessor applications.

The performance of the proposed solutions is evaluated by simulation. The objective of this evaluation is to show that, under reasonable conditions, performance of the original network with blocking switches is badly degraded by the presence of NUTS and that the modifications proposed significantly reduce this degradation. On the other hand, it is not our objective to give an extensive set of graphs from which the performance of particular networks with specific traffic patterns can be determined. Consequently, we select a set of reasonable network parameters and traffic patterns and use these for the simulation.

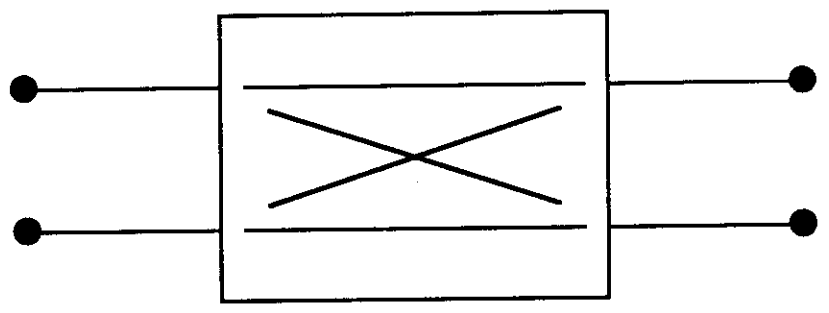
2. Multistage Network Structure and Operation

We now give a brief description of the structure and operation of the multistage network, emphasising the assumptions we make. A more detailed discussion can be found in [Sieg85]. The type of multistage interconnection network we are considering has $N = 2^n$ inputs (sources) and outputs (destinations), both labeled from 0 to $N-1$. It consists of n stages of $N/2$ 2x2 switches, as shown in Figure 1. The outputs of stage-i switches are connected to the inputs of stage-$(i-1)$ switches, with the network inputs going to stage-$(n-1)$ switches and the network outputs coming from the stage-0 switches.

Several specific multistage networks have been proposed, differing in the interconnection pattern between stages. Since the characteristics, in terms of type of operation and performance, are similar for all these different topologies, we consider here the Omega network [LAWR75], which has been extensively studied [CHEN81] and is being used in several multiprocessor systems. In this network, the interconnection pattern between stages corresponds to the perfect shuffle connection, as shown in Figure 1.



(a) An example 8x8 omega network



(b) A switch element

Figure 1

The routing of packets in the network is unique since there is a single path from a specific source to a specific destination. The control of routing is done using a destination tag that is associated with the message as part of each packet. At stage i, the routing depends only on the *ith* bit of the tag; if the bit has value 0(1), route to the upper(lower) output of the switch.

The operation of the network is synchronous and pipelined. In its basic form, each switch has one register per output and each cycle one packet is transferred from an output register in a switch of stage i to the corresponding output register of a switch of stage $i-1$. This implies that the packets are all of the same size. If the packet is large, the above mentioned cycle can be divided into several subcycles and a part of the packet be transferred per subcycle. However, we will not be concerned with this subdivision and will use the packet as the basic unit of transfer and the time of this transfer as the unit of time (one cycle).

Since each output register can receive only one message per cycle, there is a conflict when both packets entering a switch in a cycle have to be routed to the same output. One solution to this conflict is to have a buffer for each output and to store the additional packet in such a buffer. Of course, these buffers are finite so it is necessary to have an operation policy when the buffer is full. The basic scheme used is a **blocking** policy in which the predecessor switches do not send packets to a full buffer. To support this policy it is necessary to have signals from a switch to its predecessors indicating that the corresponding buffer(s) is full (Figure 2). Note that since both predecessors can send messages to the same buffer, it is necessary to establish a policy also for the case in which there is just one space in the buffer. In such a case, we select alternatively the predecessor that is blocked.

Several variations to the basic switch design are possible. The switch can have a single buffer pool to service both inputs/outputs, which leads to the best buffer utilization. However, this requires that two packets be accepted and sent from the queue per cycle and, if a FIFO policy is used, a packet in the front of the queue can block the sending of another packet. The other possibility is to have dedicated buffers, either servicing one input or one output. Input buffers have the advantage of simplifying the generation of the full signal and receive at most one packet per cycle. However, arbitration is necessary to determine which packets are sent to the output and a FIFO policy leads to the same blocking characteristics as for the single buffer. Output buffers have to be able to receive two packets per cycle and have a more complex generation of the full signal. Queue management policies can be FIFO or non-FIFO with some kind of priority scheme. More complex control algorithms are possible, leading to better utilization, but the cost and speed requirements of the switch limit the practicality of such complex algorithms. As technology improves, however, more options become available.

In this paper we do not evaluate the different buffering organizations and policies. The degradation produced by NUTS is inherent to the blocking operation of the network, which is present for any of the buffer organizations and policies. Moreover, the modifications we propose are applicable to all these organizations. Consequently, we perform our analysis using output buffers with FIFO policy.

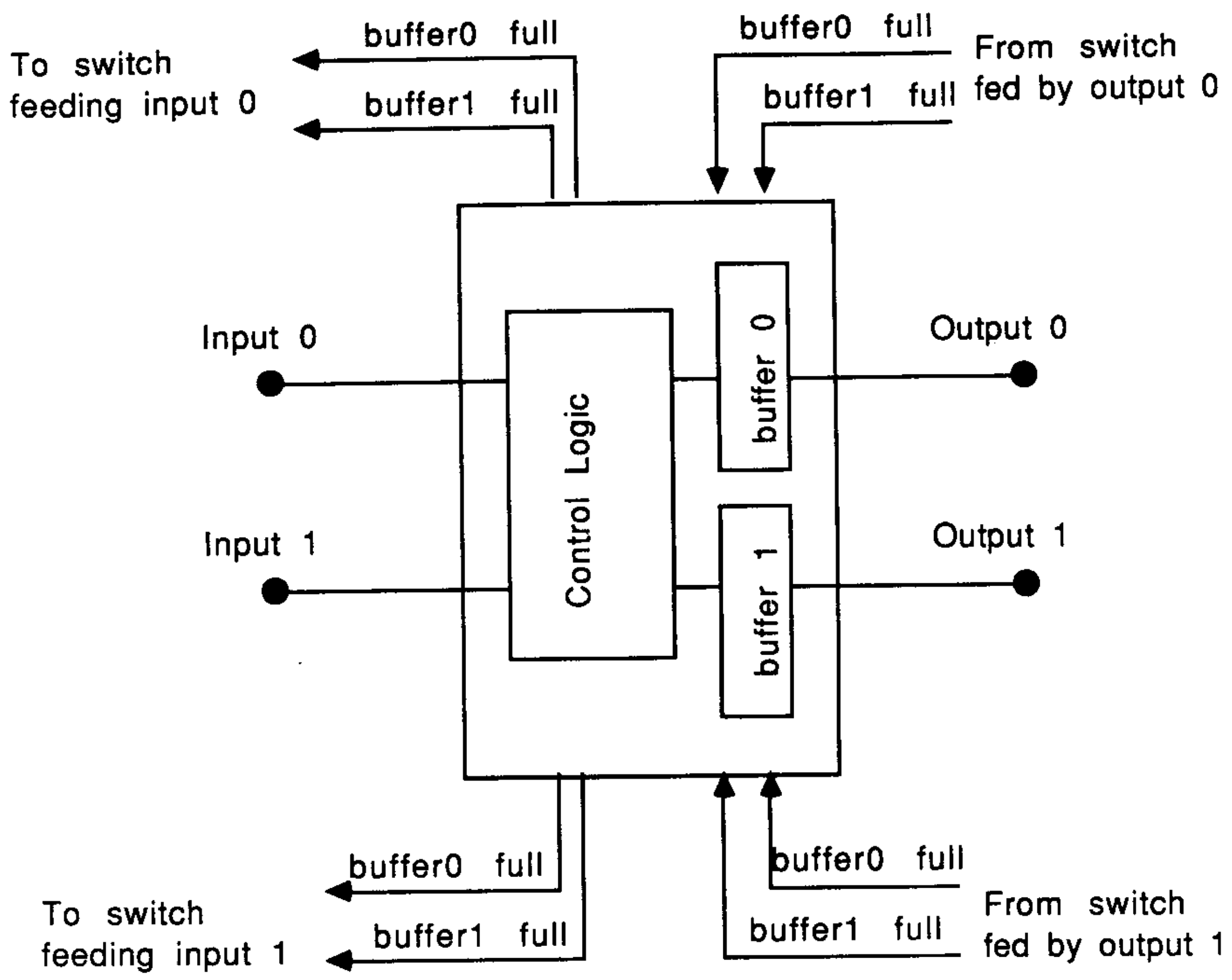


Figure 2. A blocking switch element with control lines

As mentioned, the basic network is composed of 2x2 switches. However, generalizations are possible in which *kxk* switches are used. This has the advantage of reducing the number of stages to $\log_k N$, with the corresponding reduction in delay. In this paper, we only consider 2x2 switches, but the results should be equally applicable to the general case.

When processors send request packets to remote memory modules, traffic in the opposite direction is also generated. These return packets must traverse an analogous network to reach the processors. The analysis of this type of traffic is similar to the request traffic, and is not considered here.

3. Performance evaluation by simulation

We now describe the measures that we will use to evaluate the performance of the network. We also indicate the types of traffic used and the different network parameters considered. As discussed in the introduction, we select a reasonable set of parameters and perform simulations to compare the performance for the original network and for the modifications proposed.

Of importance in our study are the different **traffic patterns** used, since the degradation due to nonuniform traffic spots (NUTS) and the applicable solutions depend on the traffic patterns considered. In the next section we present the patterns we will use.

In addition to the traffic pattern, the **traffic load** is of importance. We distinguish two types of systems: open and closed systems. In an **open** system, each processor generates a packet each r cycles, so that the load is specified by the fraction $1/r$. In a **closed** system, on the other hand, each processor has a maximum of r outstanding packets. We have found that the results are qualitatively similar for both cases for the same total throughput. Consequently, to concentrate on significant parameters, we only report on results for open systems.

We evaluate the **steady-state** behavior of the system, that is, we assume that the traffic pattern under consideration remains for a period long enough to achieve this steady state.

The fundamental parameters for the network are its size and the size of the queues. We have found that the relative performance of the network remains essentially the same for different values of these parameters. Consequently, we report our results for a network of size 64 and queues of size 2, which are also convenient because they produce a relatively small delay, except for the blocking case where, because of the way the full signal is generated, this size of queue is not adequate. In this latter case, we use a queue of size 4.

The main performance measures of interest are the **throughput** of the network in packets/cycle, and the **average delay** of the packets. The maximum throughput is of N packets per cycle and the minimum delay is of n cycles. This performance is obtained when there are no conflicts, that is, when in all cycles all switches receive two packets and route one to each output. For other cases, the performance is shown by the function **delay vs. throughput**.

In a multiprocessor system all processors cooperate in the execution of a task and have to synchronize periodically. Consequently, it is convenient for all processors to advance at a uniform pace, so that processors do not have to wait unnecessarily for slower processors. The measure we use to evaluate the relative advance of the processors is the **distribution of throughput**.

For the simulations we built a network simulator using as a basis SIMON, a general-purpose multiprocessor simulator developed at the University of Utah [FUJI86].

Several studies have been reported on the performance of multistage interconnection networks with **uniform traffic** [DiJu81, KrSn83]. In this case, the destinations are generated by a random variable with uniform distribution. Since our purpose is to consider the performance for nonuniform traffic, we performed simulations for uniform traffic only to validate the simulator and provide a reference point with which to compare the other traffic patterns. The results of our simulations for uniform traffic confirm what previous studies have indicated.

4. Traffic patterns producing NUTS

The evaluation studies that have been made for the "hot spot" problem point to a more general situation with nonuniform traffic. The same type of degradation should occur whenever the traffic is such that one or more switches carry a larger fraction of the total traffic than its share. This degradation is due to the same "tree saturation" effect observed in the hot-spot case. In the context of the Omega network, switch i of stage j carries the traffic going from a specific subset of 2^j sources to a specific subset of $2^{(\log p - j + 1)}$ destinations. Consequently, switch congestion occurs whenever this traffic is excessive. This can occur even in situations in which the fraction of traffic going to each destination is the same. The main objective of this research is to identify the traffic patterns that produce non-uniform traffic spots (NUTS), to evaluate the degradation in performance, and to propose and evaluate solutions to this problem.

To study the influence of NUTS on performance we have considered two types of traffic as follows. These types are just examples to illustrate the problem and evaluate the solutions; they correspond to situations that could occur, but are not specific practical patterns.

Traffic of Type I.

In the first type, each source issues all its requests to one destination and no pair of sources sends to the same destination. In the shared memory case, this type of pattern models a system in which each processor has a preferred memory module that contains both the code and the data for that processor. It might be argued that in such a case it would be better to assign to each processor a local memory module with direct access without going through the network (this is the scheme used, for example, in the BBN Butterfly). However, the use of the network to have a uniform access time from any processor to any memory module, permits a flexible dynamic scheduling approach that is not possible in the local scheme. In this dynamic scheduling model, a processor can have its code/data in any memory module, and this module can vary with time. This type of traffic would model also situations in which the communication is among pairs of processors.

Some specific instances of this type of traffic patterns produce significant NUTS while others do not. We use two different instances for our study. In instance 1, we use a **bit-reversal** permutation which is known to produce a large contention in the Omega network as evident from the switch positions shown in Figure 3. This is an extreme case, it shows a lower bound on the improvement that can be achieved with the techniques used. To model a more typical situation, as instance 2 we generated an **arbitrary** permutation.

The throughput-delay for these patterns is shown in Figure 4. As can be seen, there is significant degradation in performance, as compared with the uniform traffic case.

Moreover, in case of the arbitrary permutation there is a large variation between the throughput of the different processors (Figure 5). As mentioned before, this is not desirable when the processors are cooperating in a single task.

Traffic of Type II.

The second traffic pattern we consider consists of requests going from even numbered sources to destinations in the first half and from odd numbered sources to destinations in the second half (EFOS). This pattern serves to illustrate a case in which each source accesses a subset of destinations. In this case there are also NUTS. The performance of the net is shown in Figure 4.

We conclude from these simulations that the performance of the network is badly degraded by the NUTS, with respect to the performance for uniform traffic. We now explore ways to reduce this degradation.

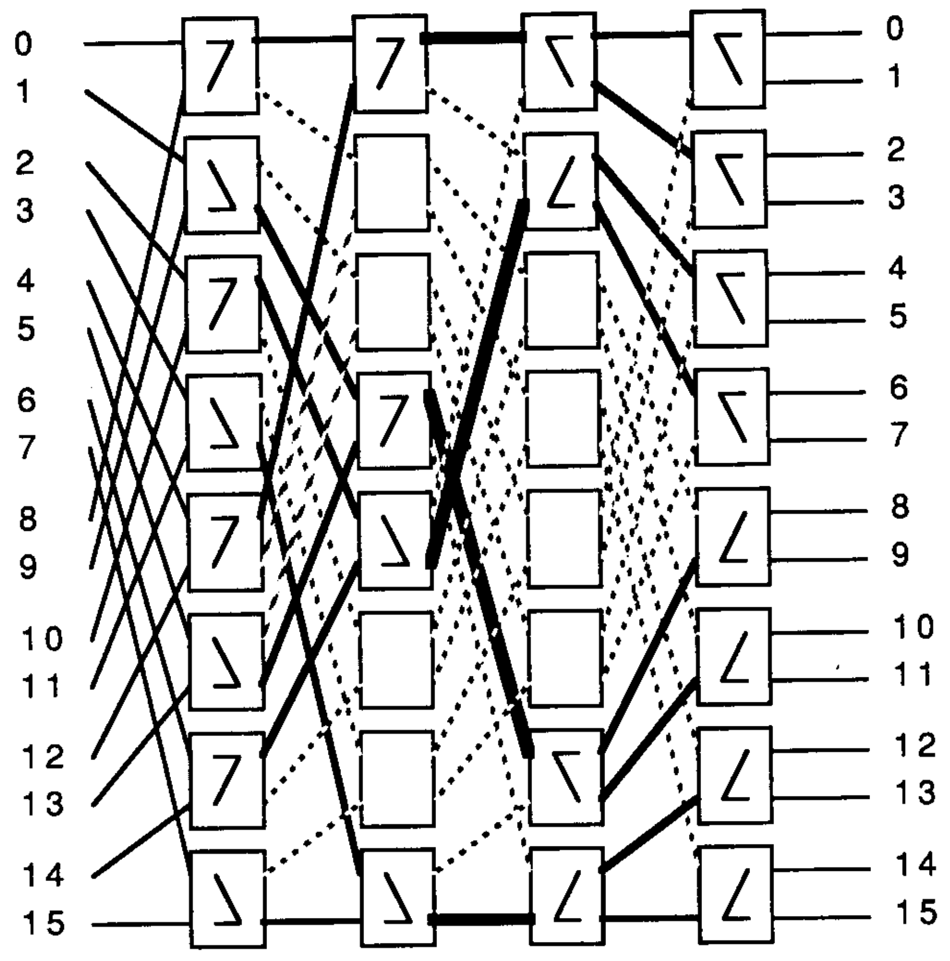
5. Two unsuccessful solutions: randomization and discarding

We now report on randomization and discarding, two approaches to reduce the degradation due to NUTS which turned out to be unsuccessful.

Randomization

As a first solution to the degradation due to NUTS, we consider the use of *randomization* of the traffic. In this approach, proposed previously to handle load imbalances in routing of multicomputers [VALI82, MITR86], packets are first sent to random destinations and then rerouted to their final destinations. This scheme has the effect of making the traffic pattern uniform and, therefore, of eliminating the added congestion of nonuniform traffic.

In the context of multistage networks, the use of this scheme implies that all messages make two passes through the network. This has the two negative effects of doubling the minimum delay and reducing the effective throughput to half, because of the additional traffic through the net produced by rerouting. The results of simulations for the two types of traffic described in the previous section are shown in Figure 6, which exhibits the expected throughput



Switch Positions For The Bit Reversal Permutation
Thick Lines Indicate Heavier Traffic Across Links.

Figure 3

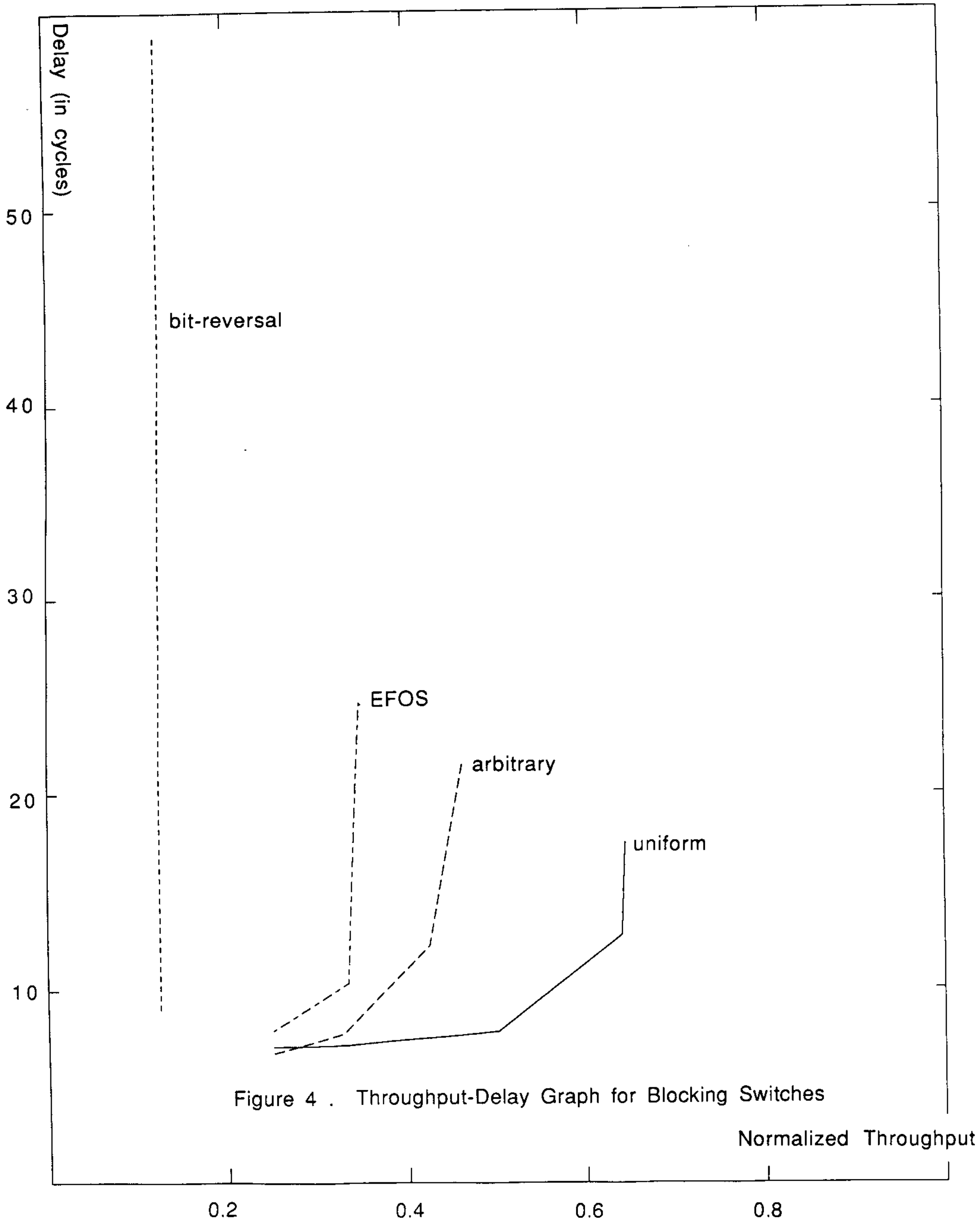


Figure 4 . Throughput-Delay Graph for Blocking Switches

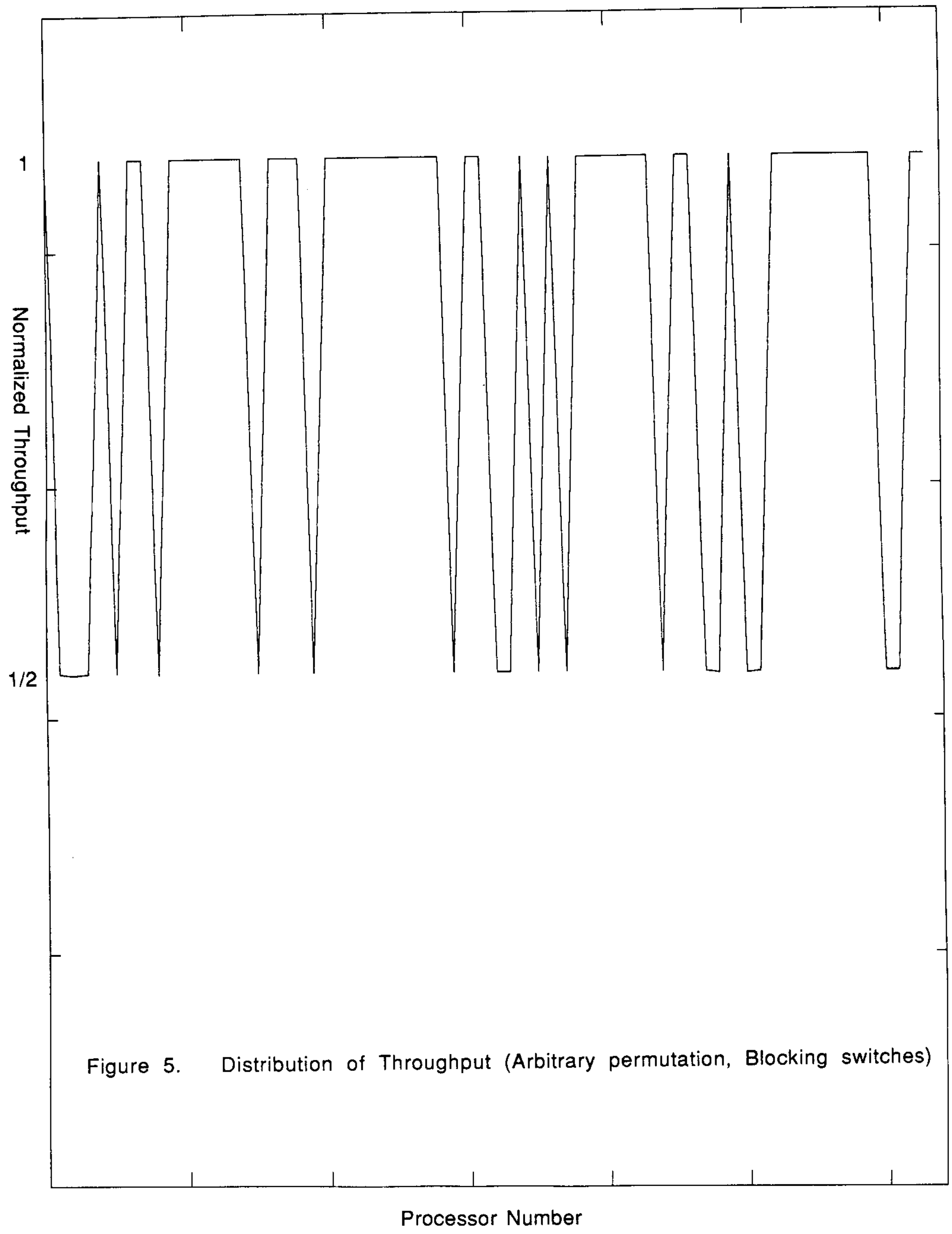


Figure 5. Distribution of Throughput (Arbitrary permutation, Blocking switches)

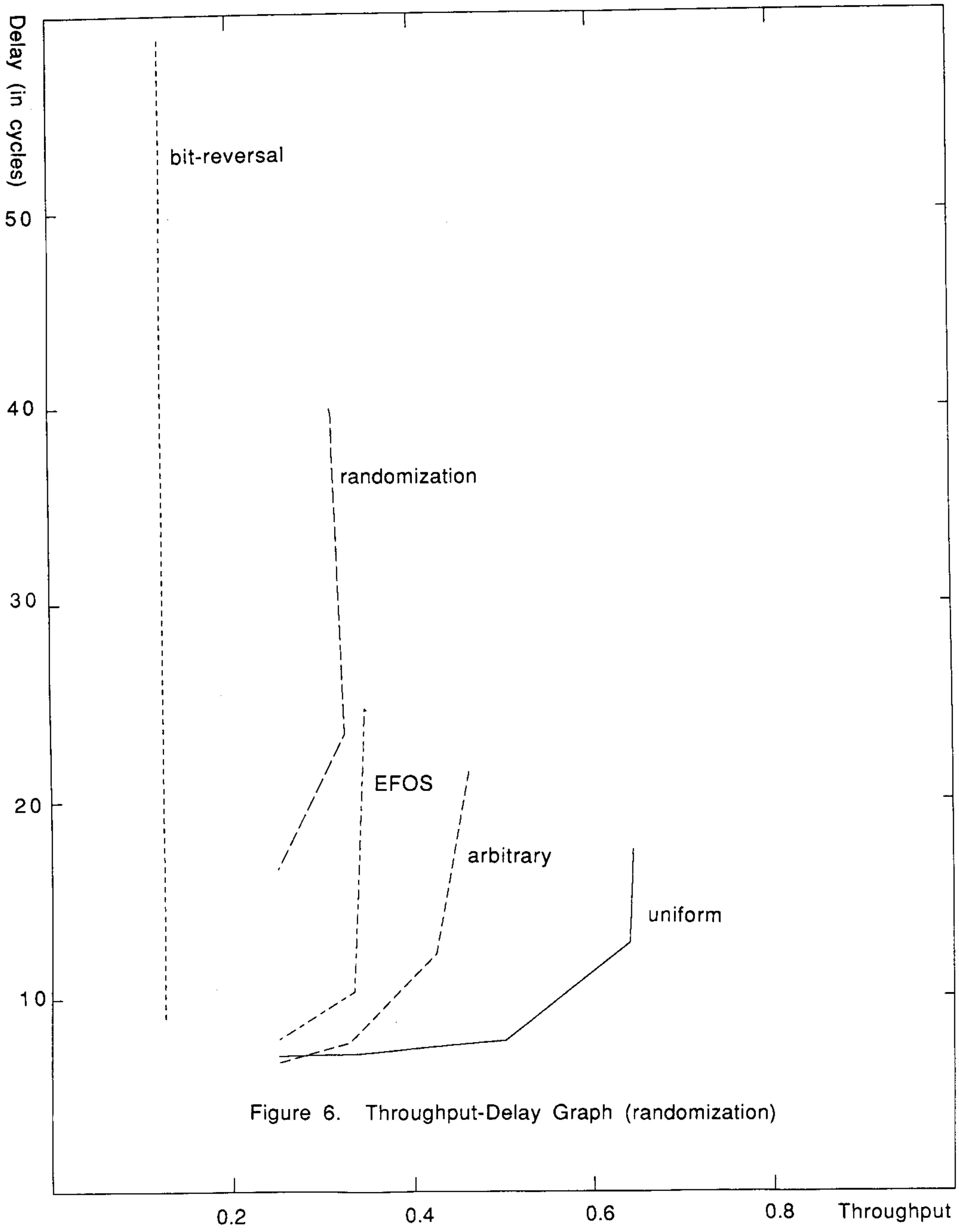


Figure 6. Throughput-Delay Graph (randomization)

and delay. As seen from there, randomization produces a relatively small improvement for the extreme bit-reversal case, while it is detrimental for the others.

Discarding Switch

Another solution we considered was the use of discarding switches. Switches of this type resolve congestion by discarding overflow packets instead of blocking. The original source of the packet is made aware of the status of the packet, through an explicit signal from the switch or a timeout mechanism, and retransmits it. Note that this requires the source to buffer all outstanding packets until an acknowledgement is received from the destination. The switches also require the ability to signal the appropriate source that a particular packet was discarded. This requires additional interconnect and more complex control.

This type of switch is used in the Butterfly Parallel Processor to deal with contention in the network and avoid ''tree saturation'' [Thom86].

The simulations reported in Figure 7, show that for the traffic patterns considered there is no improvement with respect to the network with blocking switch. This can be explained by the fact that the discarded traffic is reissued by the same processor as the first time and, therefore, follows the same path leading to the NUTS.

6. Diverting Switch

In a *diverting switch* the messages in front of the buffers are always sent to the successors, irrespective of whether there is space for them in the corresponding destination buffers. If both messages that arrive to a switch go to the same output buffer and there is no space for both, then one of the messages is diverted to the other buffer of the switch (Figure 8). Note that there is always at least space for one message in each buffer since one message departs from each buffer in every cycle. Of course, the diverted message will go to a wrong destination (since there is just one path in the network for each source/destination pair); therefore, the message will have to be resent into the network to the correct destination. Consequently, this mode of operation requires a connection between each network output and corresponding input (a wrapped-around organization).

Diversion has potentially a better performance than discarding because the packets are rerouted from a source that is **different** from the original source. This makes it possible for the message to avoid the NUTS in the second pass.

Since to obtain a good performance it is convenient to reduce the number of packets that are diverted, whenever a conflict occurs and one of the packets in the conflict has already been diverted (in that pass through the network) we give preference to the nondiverted packet (to go to the correct destination).

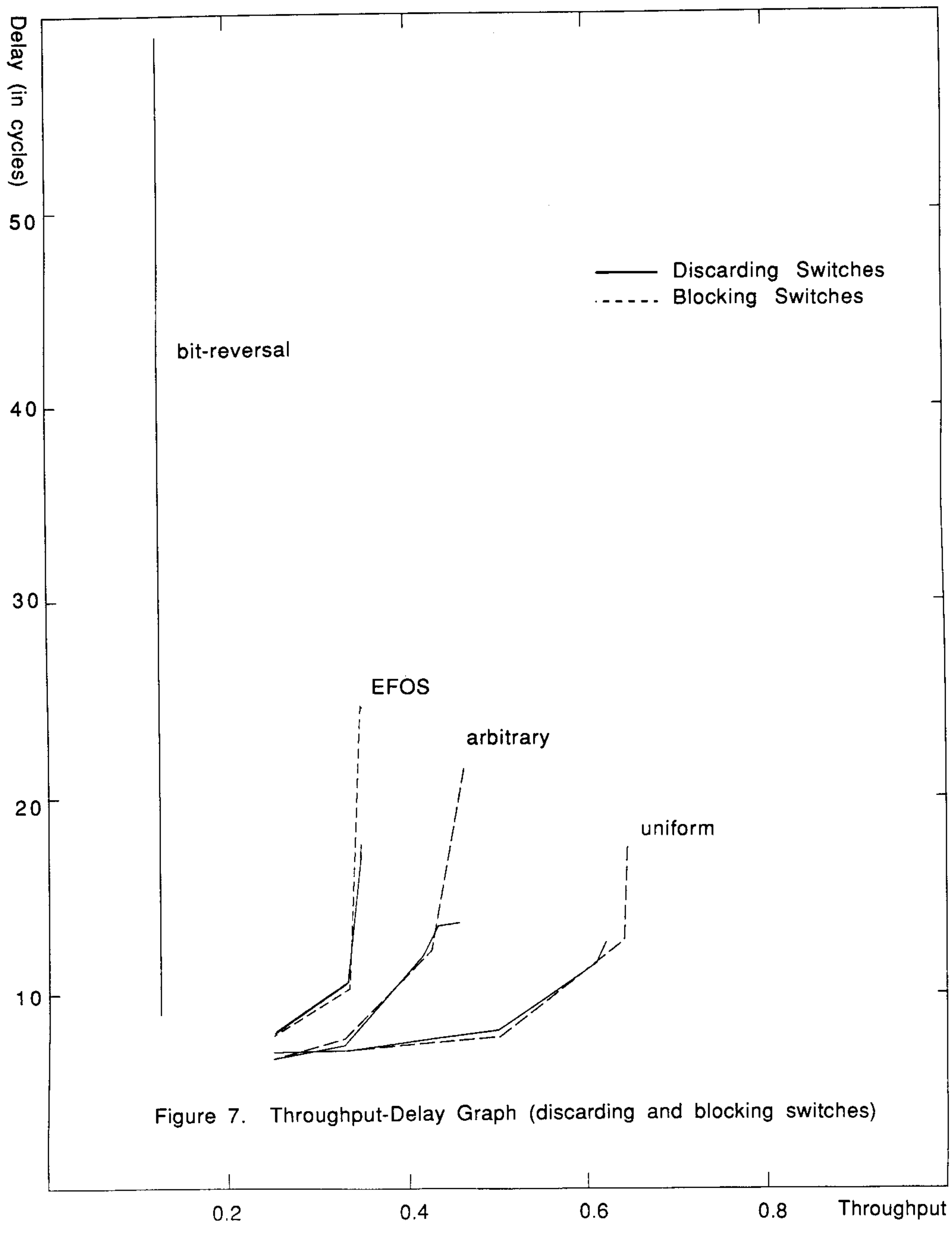
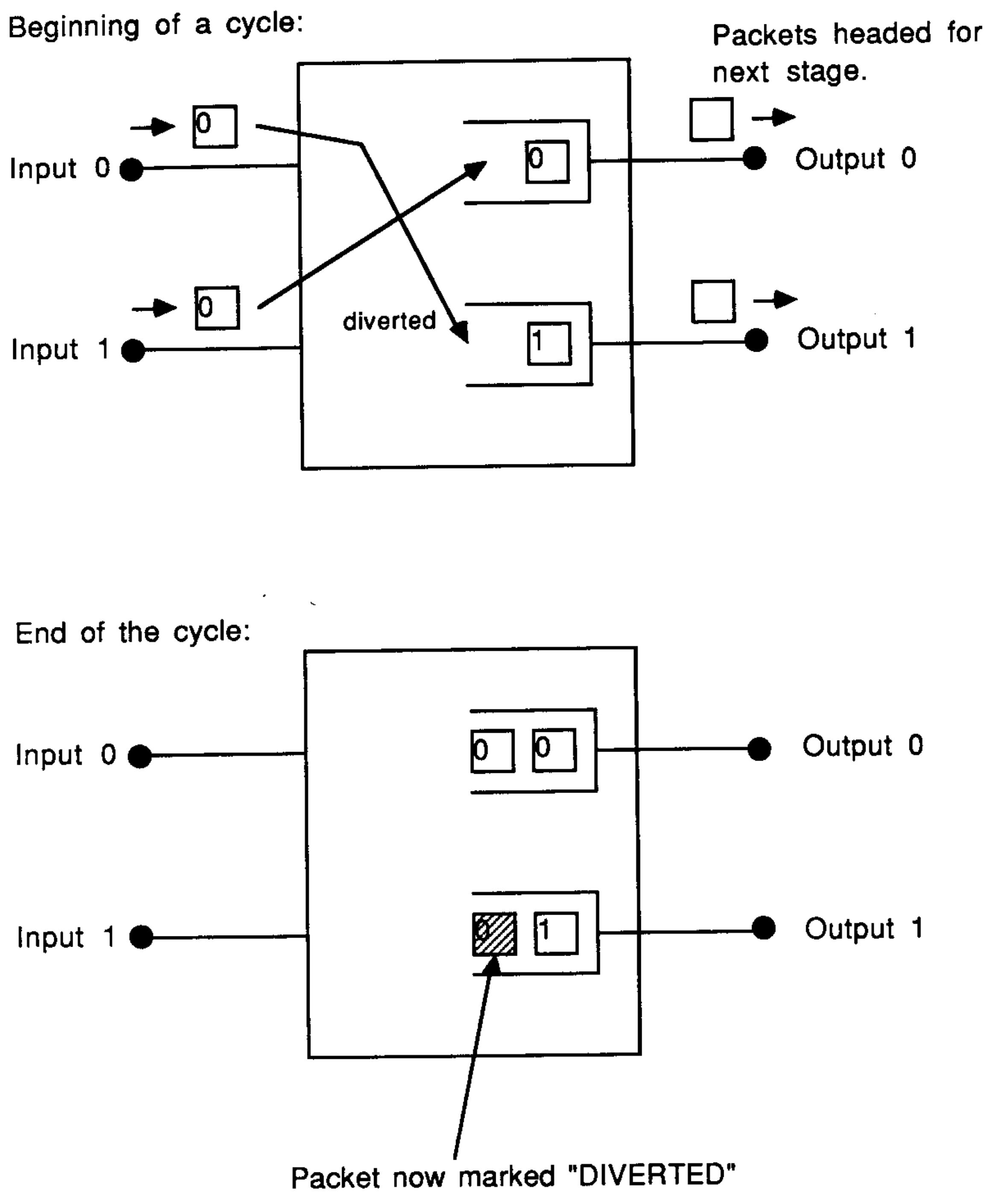


Figure 7. Throughput-Delay Graph (discarding and blocking switches)



Packets can be diverted into the wrong destination queue.

Figure 8

Because of the diversions, a message might traverse the network several times before getting to its destination. A possible problem with this form of operation is that it is not possible to assure that a particular message will have a bounded delay. To avoid this and make the delay more uniform, we give *preference to older packets*.

Diverting policies

Once a packet is diverted in the network, it cannot reach its desired destination during that pass; it has to go through the network again. This means that we now have a great deal of freedom in deciding where to route these diverted packets. The main goal is to route the diverted packet to an interim destination that has a "clear" path to the true destination, so that it will not be diverted again.

We have experimented with several diverting policies. We present here the results of two of them, to show that diverting produces an improvement in the performance and that the specific diverting policy has an impact.

The first diverting policy we call *direct diverting*. In it the routing of the diverted message continues using the destination tag. That is, each time the message is diverted actual destination is wrong in the corresponding bit. As shown in Figure 9 the performance is significantly better than with the blocking policy.

The second diverting policy we call *complement diverting*. In this case, once a message is diverted, instead of using the destination tag for routing, it is routed using a tag corresponding to the complement of the source. On its next pass through the network, the original destination tag is again used. This policy has the advantage that it assures that the rerouted message will avoid the NUTS where it was diverted in the first pass, as illustrated in Figure 10. Of course, it can pass through some other NUTS.

Figure 9 shows the corresponding performance for the various traffic patterns. We see that this policy produces a somewhat better performance than the direct policy.

These simulation results indicate that the use of diverting switch improves the throughput-delay characteristic of the network when the traffic produces NUTS. Moreover, the use of diverting switches makes the distribution of throughput more uniform, as shown in Figure 11.

7. Network with alternate paths

The MIN's previously considered have the characteristic of a unique path between each source-destination pair. Several reports [AdSi82, KuRe85, TZEN85] have described adding redundant paths to MIN's to improve fault tolerance characteristics. These alternate paths can also improve the performance of a fully functional network.

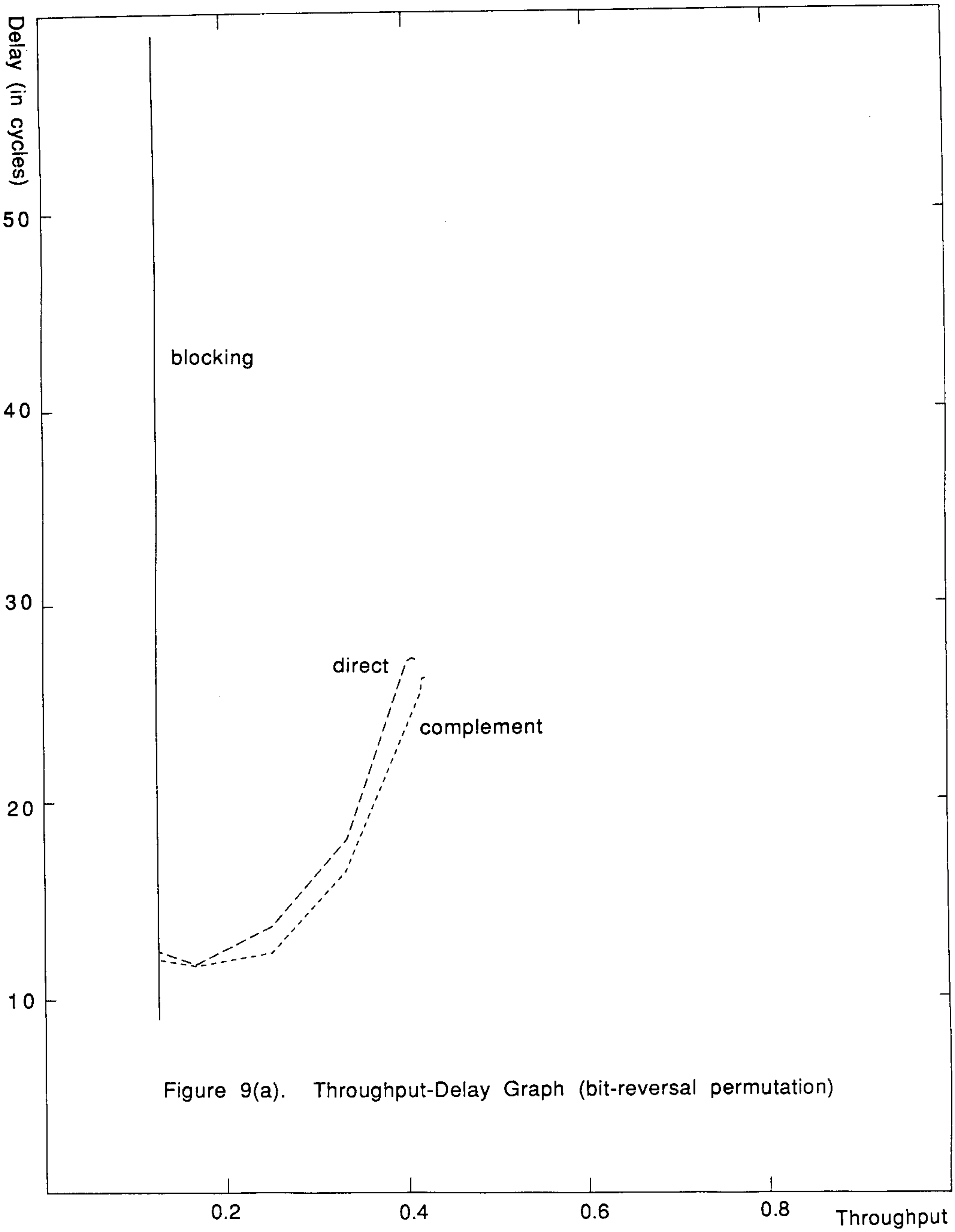


Figure 9(a). Throughput-Delay Graph (bit-reversal permutation)

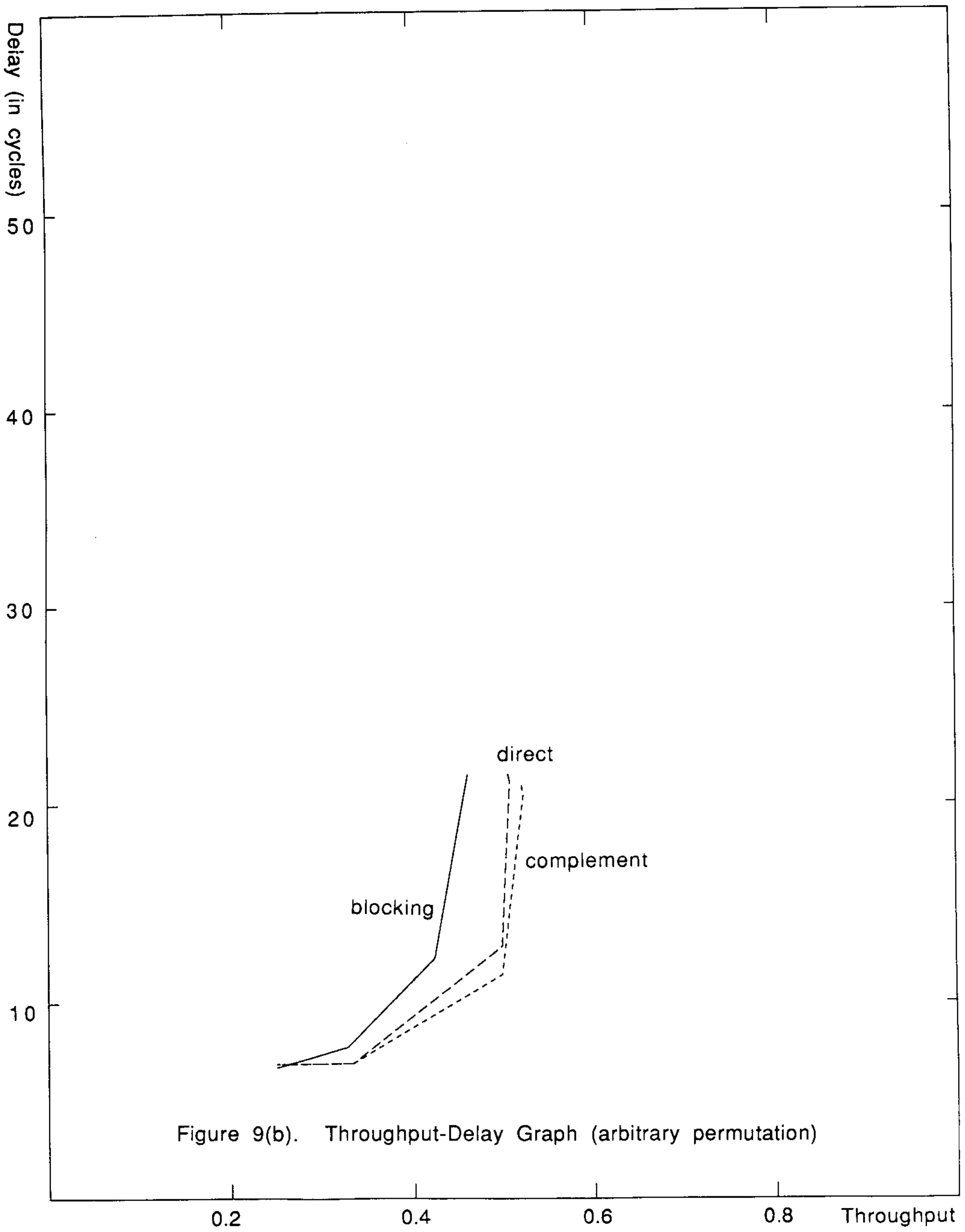


Figure 9(b). Throughput-Delay Graph (arbitrary permutation)

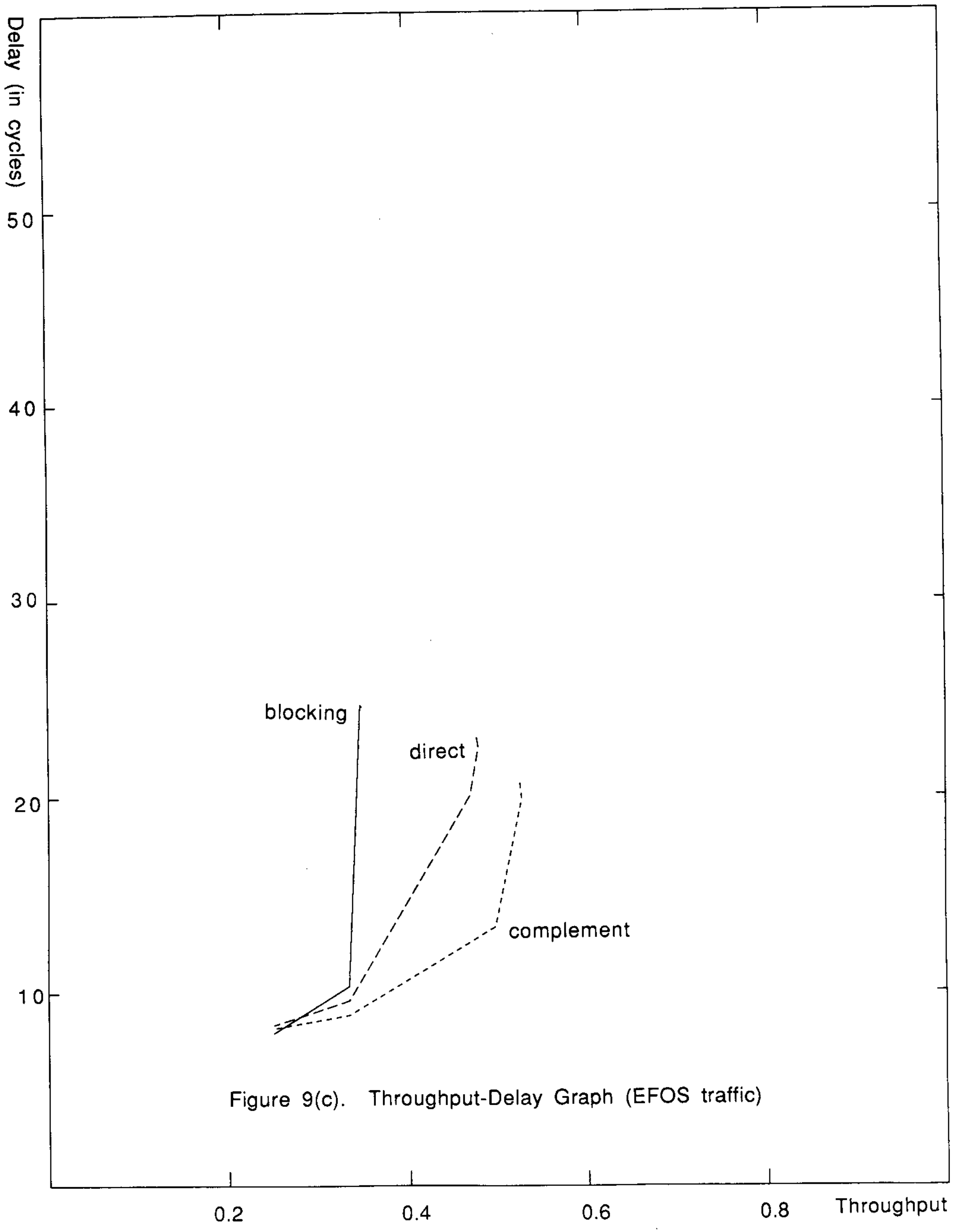
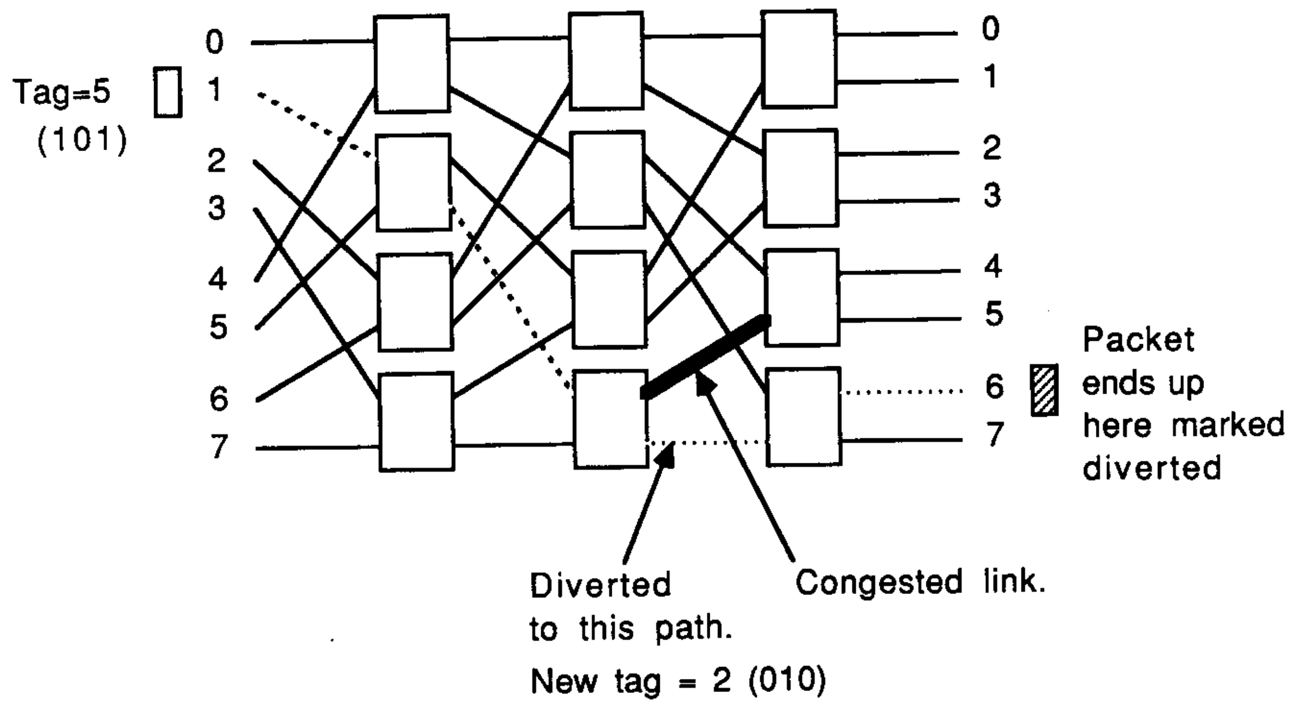
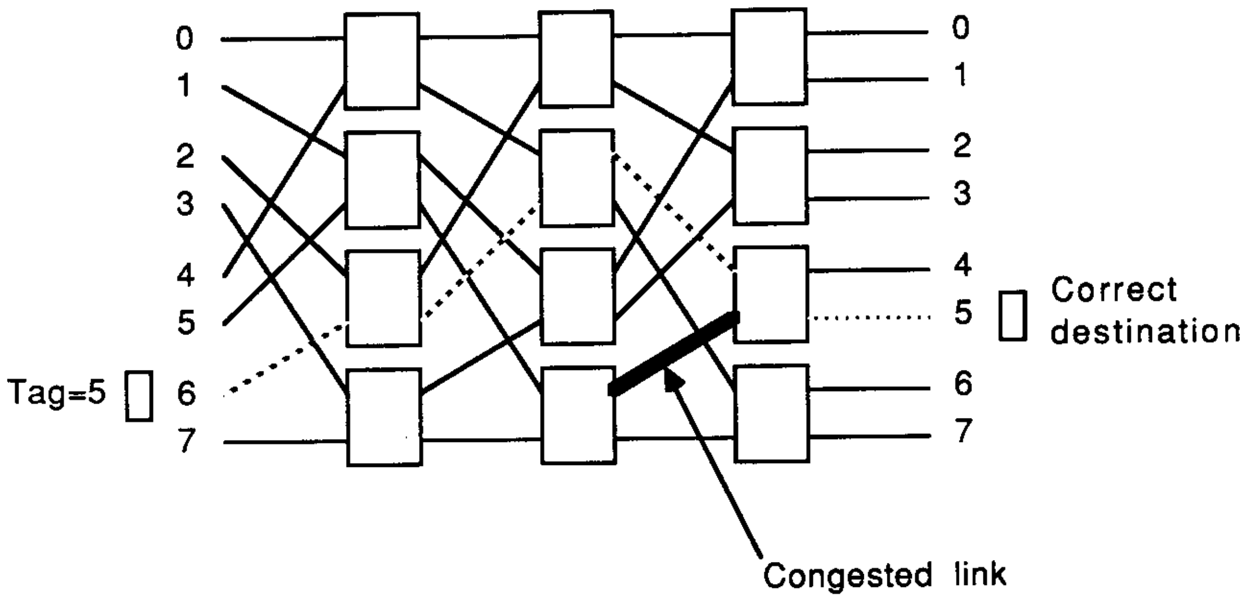


Figure 9(c). Throughput-Delay Graph (EFOS traffic)

First pass through network:



Second pass through network:



Diverting to the Complement of the Source avoids the congested link on the second pass.

Figure 10

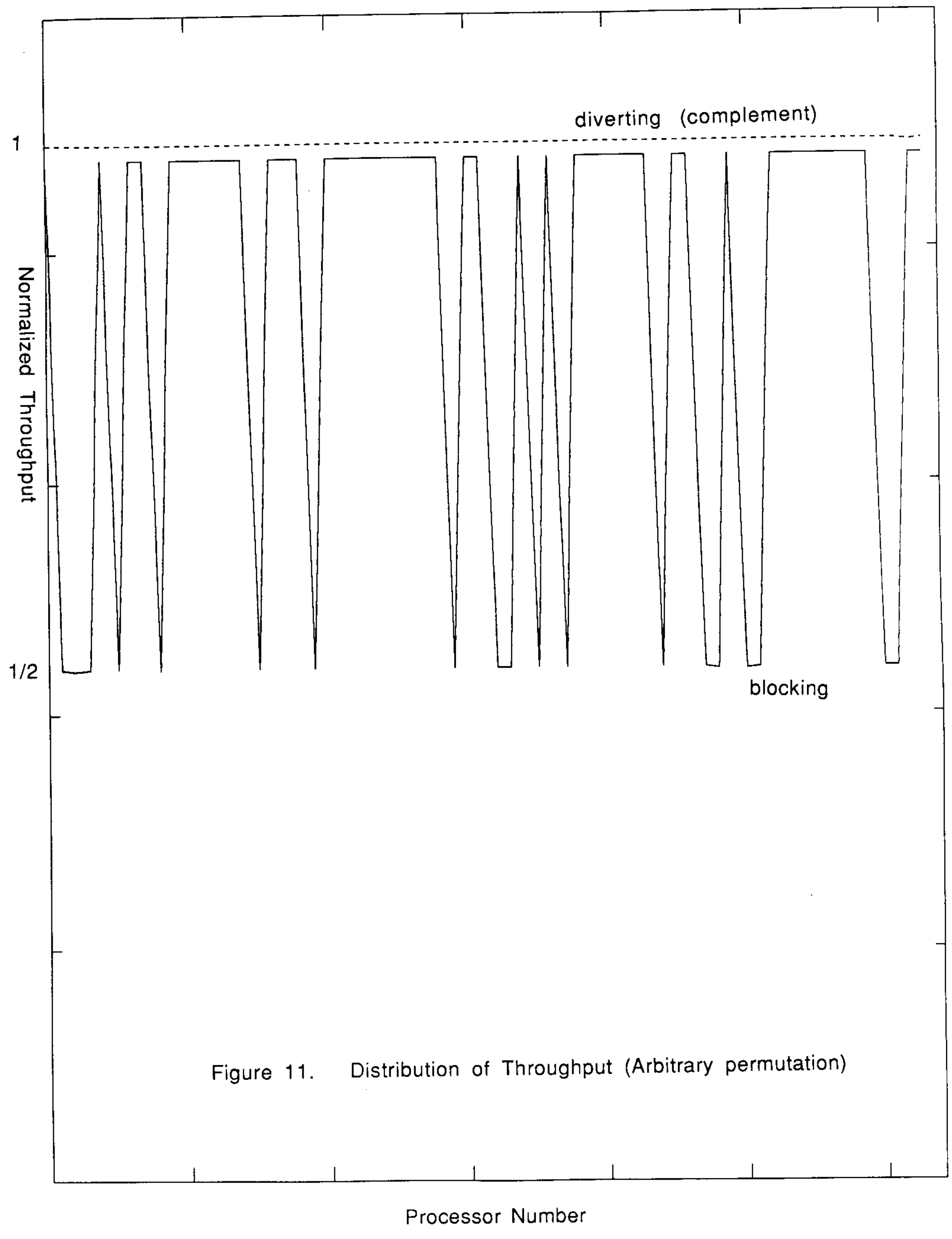


Figure 11. Distribution of Throughput (Arbitrary permutation)

In particular [TZEN85] and [KuRe85] propose the addition of links to connect switches in the same stage into rings so that from any switch in a particular ring packets can reach the same subset of destinations. The application of this technique to the OMEGA network is illustrated in Figure 12(a). If a packet entering a switch finds the desired output queue full, it can be re-routed to another switch in the same group via the alternate path link, and still be able to reach its true destination directly without a second pass through the network.

This modified OMEGA network requires augmented switches acting as a 3x3 crossbar, as shown in Figure 12(b). The routing control is somewhat more complex than for the original 2x2 switch. We still use a diverting policy, because this has given better performance for the original network and this policy is also simpler to control since no "full signals" are needed. Each cycle up to three packets enter the switch. They are placed in the output queues giving priority to the older packets that have not been diverted (in that pass). The highest-priority packet is always placed in the correct queue, since there is always at least one space in each queue (because one packet leaves each queue each cycle). The next packet is placed in the correct queue, if there is space, or in the alternate queue. Finally, the least-priority packet is placed in the correct queue, in the alternate queue, or in the wrong queue (diverted).

Figure 13 shows the performance of the network with alternate paths for the various traffic patterns. We can see that the introduction of alternate paths produces a big reduction in the degradation due to NUTS.

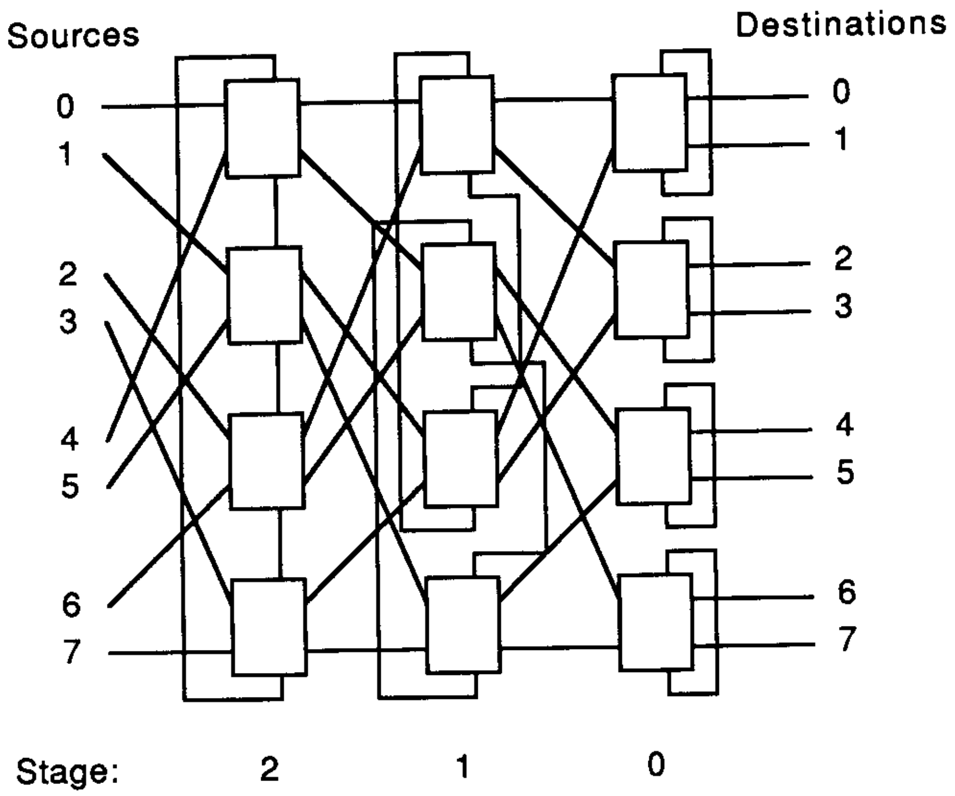
8. Conclusions

We have shown several traffic patterns that produce NUTS in multistage interconnection networks and therefore result in a degradation of performance. The randomization technique, proposed for eliminating imbalances of loading in multicomputers, is not appropriate in this case because it increases the delay of each packet and the real traffic through the network. The use of discarding switches instead of blocking switches is not advantageous either because the discarded traffic has to be resent through the same congested path.

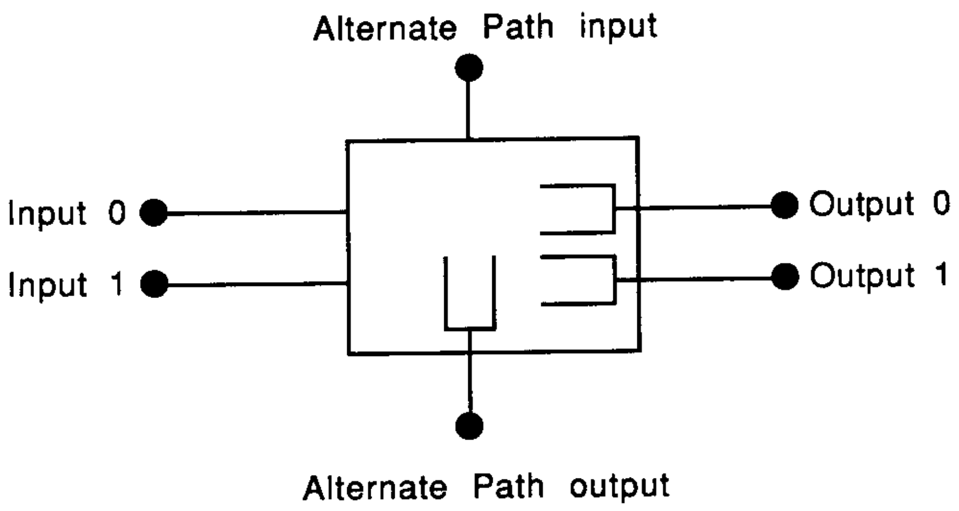
As positive solutions, we have shown that diverting switches produce a significant reduction in the degradation. Moreover, the control of congestion is simpler than that for blocking switches because no "full signals" are needed. However, to implement this policy, it is necessary to have a network with wrap-around connections.

The performance is much better using networks with alternate paths. However, this network require 3x3 switches instead of the basic 2x2, which complicates the implementation.

The use of these modifications to the basic blocking policy in the control of contention in multistage interconnection networks makes it possible to use the network effectively for a larger variety of traffic patterns.



(a) 8x8 Omega network with alternate path links



(b) An augmented switch element

Figure 12

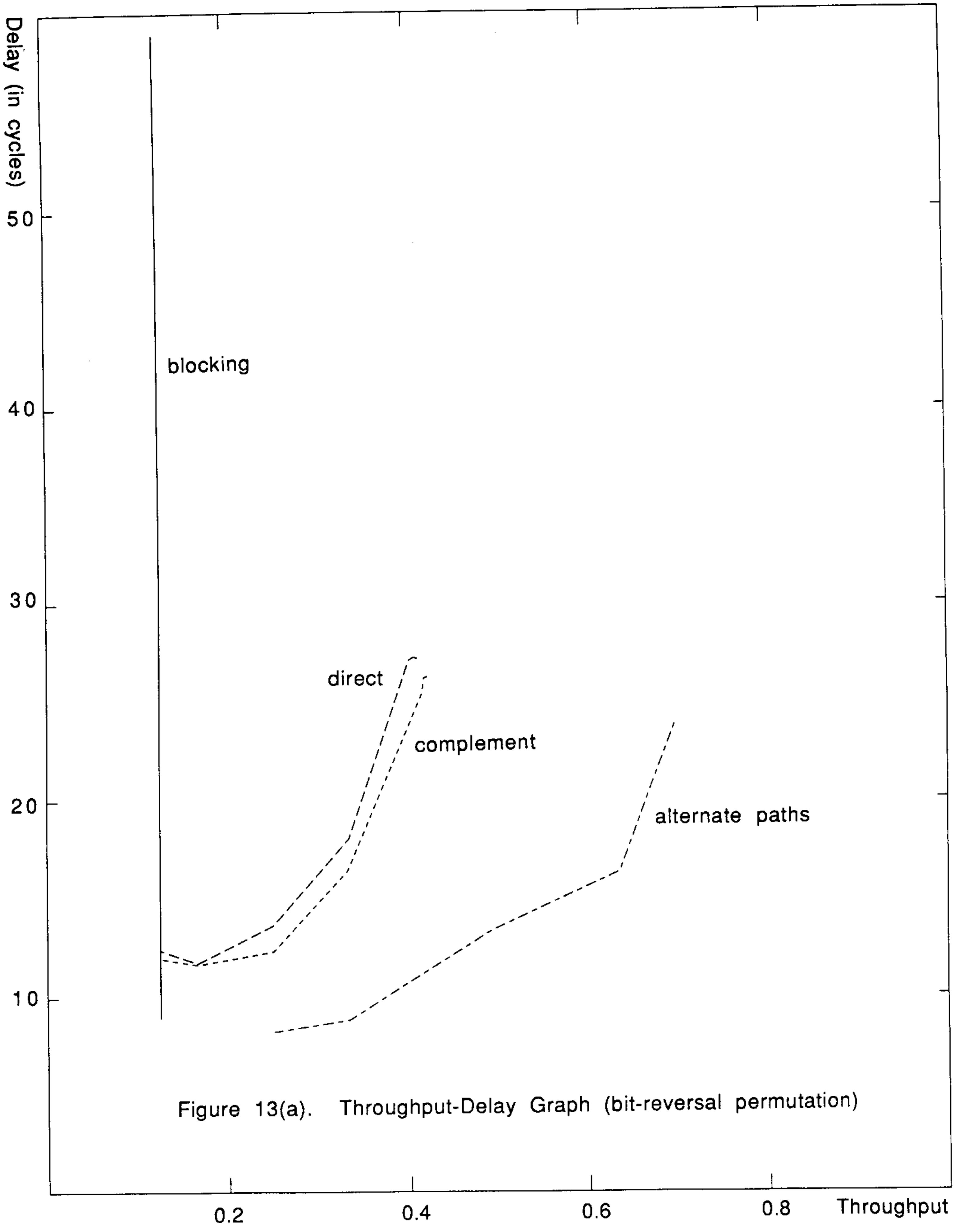


Figure 13(a). Throughput-Delay Graph (bit-reversal permutation)

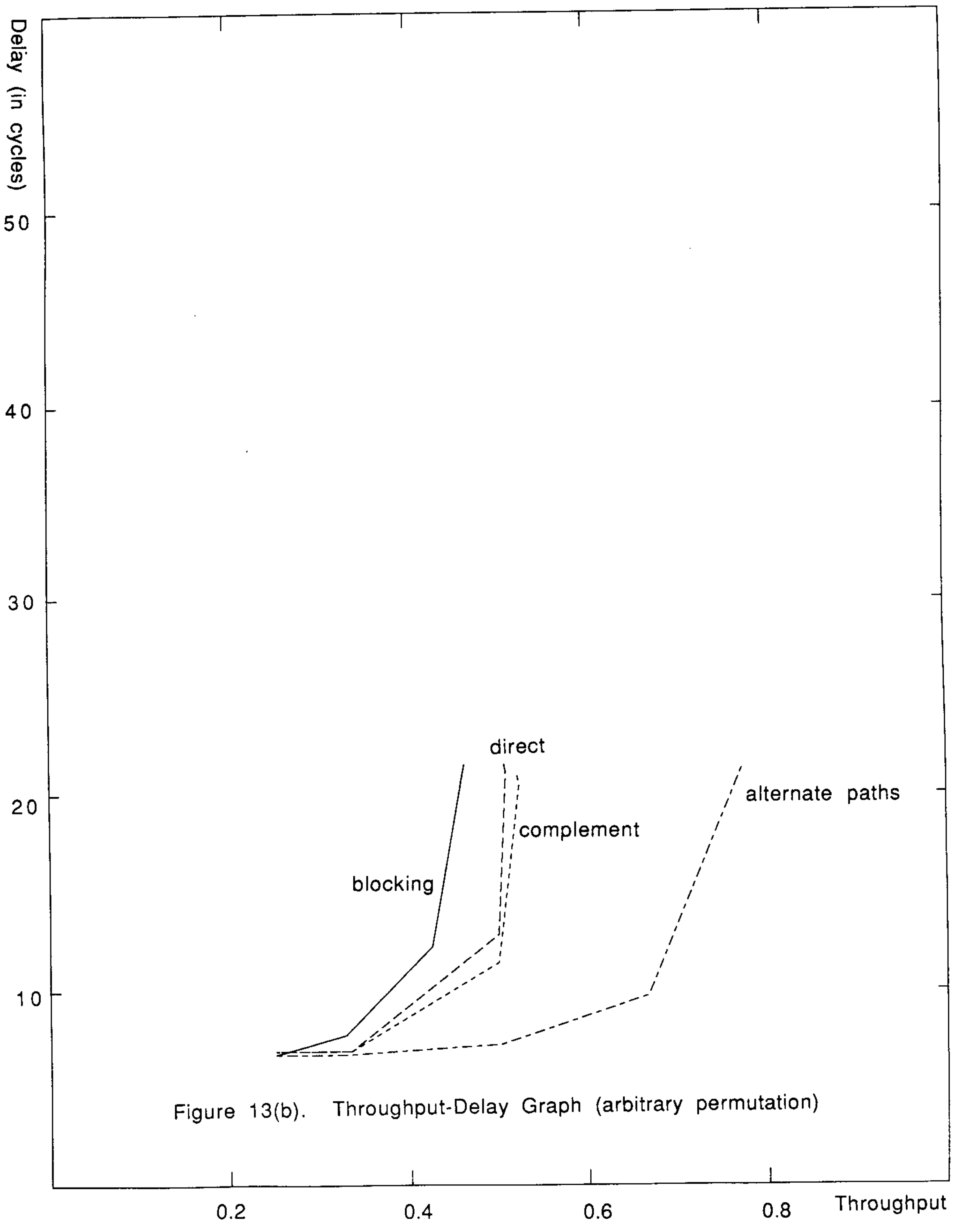


Figure 13(b). Throughput-Delay Graph (arbitrary permutation)

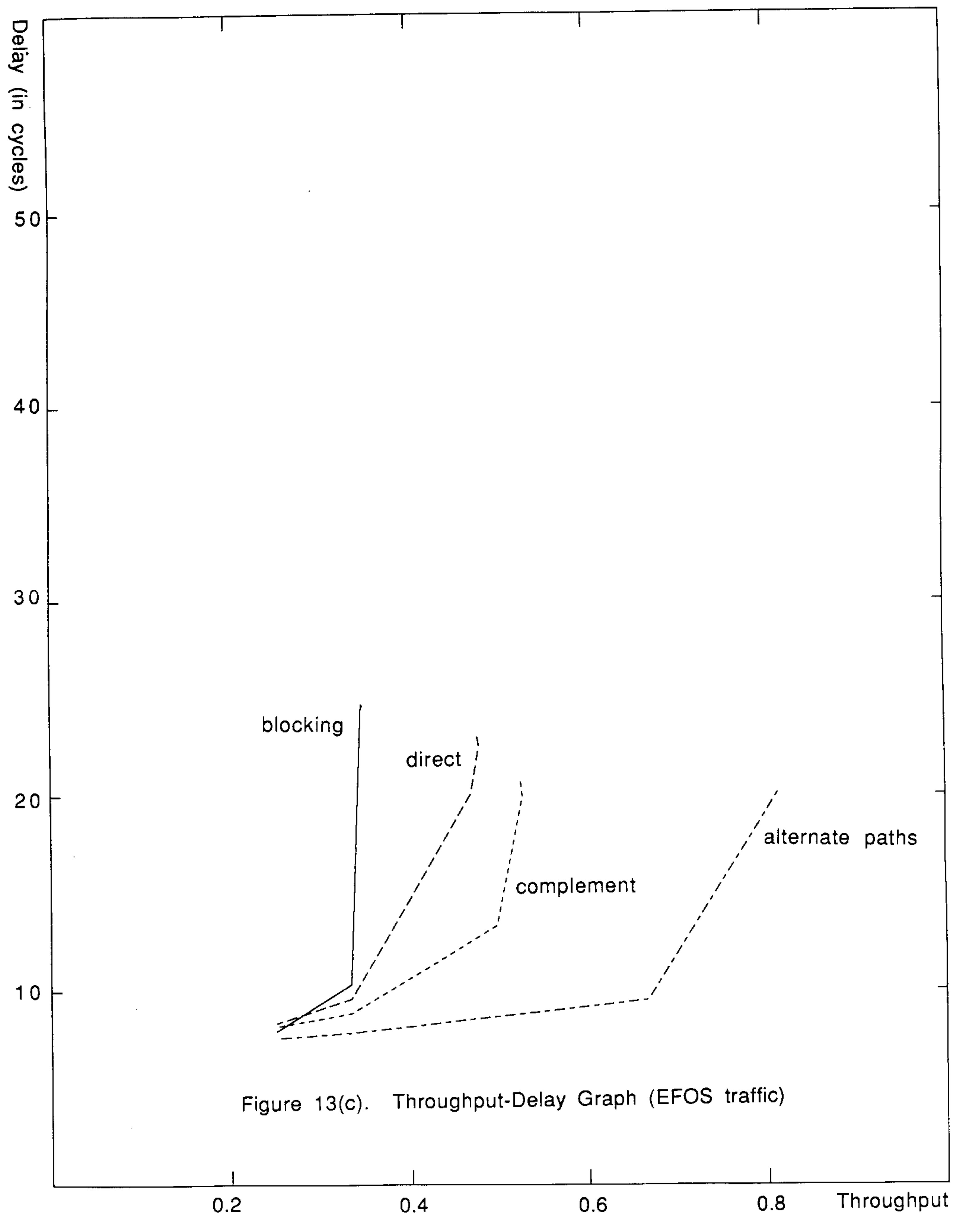


Figure 13(c). Throughput-Delay Graph (EFOS traffic)

Acknowledgement

This research was sponsored in part by the State of California MICRO program and Hughes Aircraft Co.

9. References

[AdSi82] G.B. Adams and H.J. Siegel, "The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems," IEEE Transactions on Computers, Vol. C-31, No. 5, May 1982, pp. 443-454.

[BMcW85] W. C. Brantley, K. P. McAuliffe, J. Weiss, "RP3 Processor-Memory Element", Proceedings of the 1985 International Conference in Parallel Processing, August 1985, pp. 782-789.

[CHEN81] P. Chen, D. Lawrie, and P. Yew, "Interconnection Networks Using Shuffles", Computer, December 1981, pp. 55-64.

[DiJu81] D. Dias and R. Jump, "Packet Switching Interconnection Networks for Modular Systems", Computer, December 1981, pp. 43-54.

[FUJI86] R.M. Fujimoto, "The CSIMON Interface", Computer Science Department, University of Utah, 1986

[GOTT83] A. Gottlieb, et al. "The NYU Ultracomputer- Designing an MIMD Shared Memory Parallel Computer," IEEE Transactions on Computers, Vol. C-32, No. 2, pp. 175-189.

[HWAN84] K. Hwang and F. Briggs, "Computer Architecture and Parallel Processing", McGraw Hill, 1984.

[KuPf86] M. Kumar and G. Pfister, "The Onset of Hot Spot Contention", Proceedings of the 1986 International Conference on Parallel Processing, August 1986.

[KuRe85] V. P. Kumar and S. M. Reddy, "Design and Analysis of Fault-Tolerant Multistage Interconnection Networks With Low Link Complexity", 12th Annual Symposium on Computer Architecture, pp. 376-386 (June 1985).

[KrSN83] C.P. Krustal and M. Snir, "The Performance of Multistage Interconnection Networks for Multiprocessors", IEEE Transactions on Computers, Vol. 32, No. 12, December 1983, pp. 1091-1098.

[LAWR75] D.H. Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Transactions on Computers, Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.

[LEE85] R. Lee, ''On Hot Spot Contention'', Computer Architecture News, Vol. 13, No. 5, December 1985, pp. 15-20.

[MITR86] D. Mitra, "Randomized Parallel Communications", Proc. of the 1986 Int. Conf. on Parallel Processing, pp. 224-230.

[PBGH85] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, J. Weiss, ''The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture'', Proceedings of the 1985 International Conference in Parallel Processing, August 1985, pp.764-771.

[PfNo85] G. F. Pfister and V. A. Norton, ''Hot Spot Contention and Combining in Multistage Interconnection Networks'', IEEE Transactions on Computers, vol. C-34, No. 10, October 1985, pp. 943-948.

[RoMa86] D. S. Rosenblum and E. W. Mayr, ''Simulation of an Ultracomputer with Several 'Hot Spots' '', Stanford Technical Report STAN-CS-86-1119, June 1986.

[Sieg85] H. J. Siegel, ''Interconnection Networks for Large-Scale Parallel Processing, Theory and Case Studies'', Lexington Books, 1985.

[Thom86] R. H. Thomas, ''Behavior of the Butterfly Parallel Processor in the Presence of Memory Hot Spots'', IEEE Parallel Processing, 1986.

[TYZ85] N. Tzeng, P. Yew, and C. Zhu, ''A Fault-Tolerant Scheme for Multistage Interconnection Networks'', 12th Annual Symposium on Computer Architecture, pp. 368-375 (June 1985).

[VALI82] L.G. Valiant "A Scheme for Fast Parallel Communication", SIAM J. Comput. Vol. 11, No. 2, May 1982, pp. 350-361.