broadcast can be carried out in time $O(n)$. Other algorithms can be implemented using divide and conquer trees. A divide and conquer tree is a tree whose leaves are labeled with the $p \in S_n$, and if p is the label of an interior vertex then its sons are labeled $p, p(1i)$ for some i. Essentially optimal divide and conquer trees are constructed in [7], of height $O(n \log n)$; this construction can readily be adapted to tQ graphs.

Algorithms which can be implemented using a divide and conquer tree thus run with hypercube performance on a star. Examples are numerous, and include single link synchronous broadcast, associative binary operation on one element per processor, and all prefixes for the latter where the processors are ordered according to the order of the leaves of the divide and conquer tree. It is an interesting question whether there are divide and conquer trees of height $O(n \log n)$, whose leaves are ordered according to the standard enumeration of S_n. We suspect that these might exist.

A universal sequence is a sequence of transpositions $(1i_1), \ldots, (1i_r)$ such that every permutation in S_n occurs as a product of a subsequence. Universal sequences of length $O(n \log n)$ are constructed in [1]. They can also be found by finding a common supersequence of the branches of a routing tree. Determining the shortest common supersequence of a set of sequences is NP complete [15]. For small n however they may easily be found by computer search. For example for $n = 5$ the shortest universal sequence of this type has length 12; an example is

$$234523425342.$$

A universal sequence for S_n yields one for a tQ graph, but there may be shorter ones.

As mentioned in [1], given a universal sequence a divide and conquer tree can be derived. The height of the tree is bounded by the length of the universal sequence. To derive the divide and conquer tree, start with ι and the first transposition in the sequence. For each leaf p of the tree constructed so far, if $p(1i)$ does not occur in the tree, where $(1i)$ is the current transposition in the universal sequence, add sons labeled $p, p(1i)$ and advance the current transposition. Continue until all p occur in the tree. There are algorithms which require a universal sequence and not just a divide and conquer tree, such as the multinode broadcast algorithm of [7].

The mesh embedding of Section III has been used to obtain algorithms for some problems. These run on tQ graphs, but since many of them have been superseded by [4] we will not consider this further.

X. Conclusion

We have given what appears to be a useful notion of a "standard" enumeration of the star graph S_n. A bound is given on the diameter of prefixes and suffixes, and an $O(n)$ interval broadcast algorithm is given. The prefixes of length $t(n-1)!$ for some t are especially well behaved. These graphs would be the same in any coset respecting enumeration, but the standard enumeration is useful in discussing them, as it is for S_n itself.

References

[1] S. B. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," in *Proc. Int. Conf. Parallel Process.*, 1986, pp. 216–223.

[2] S. B. Akers, D. Harel, and B. Krishnamurthy, "The star graph: An attractive alternative to the n-cube," in *Proc. Int. Conf. Parallel Process.*, 1987, pp. 393–400.

[3] P. Fragopoulou and S. Akl, "Optimal communication algorithms on star graphs using spanning tree constructions," Queen's Univ. Dep. Comput. Inform. Sci., Tech. Rep. 93-346.

[4] N. Bagherzadeh, M. Dowd, and S. Latifi, "Faster column operations in star graphs," Dep. Elec. Comput. Eng., Univ. California, Irvine, CA, Tech. Rep. ECE-94-02-01, 1994 (submitted for publication).

[5] J. S. Jwo, S. Lakshmivarahan, and S. K. Dhall, "Embedding of cycles and grids in star graphs," *J. Circ., Syst., Comput.*, vol. 1, pp. 43–74, 1991.

[6] K. Kim and V. K. P. Kumar, "An iterative sparse linear system solver on star graphs," in *Proc. Int. Conf. Parallel Process.*, vol. III, 1991, pp. 9–16.

[7] V. Mendia and D. Sarkar, "Optimal broadcasting on the star graph," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, pp. 389–396, 1992.

[8] M. Nigam, S. Sahni, and B. Krishnamurthy, "Embedding Hamiltonians and hypercubes in star interconnection networks," in *Proc. Int. Conf. Parallel Process.*, 1990, pp. 340–343.

[9] H. Katseff, "Incomplete hypercubes," *IEEE Trans. Comput.*, vol. 37, pp. 604–608, 1988.

[10] S. Sur and P. Srimani, "Super star: A new optimally fault tolerant network architecture," manuscript.

[11] S. Latifi and N. Bagherzadeh, "Incomplete star: An incrementally scalable network based on the star graph," *IEEE Trans. Parallel Distrib. Comput.*, vol. 5, 1994.

[12] K. Day and A. Tripathi, "A Comparative study of topological properties of hypercube and star graphs," to appear in *IEEE Trans. Parallel Distrib. Syst.*

[13] F. Annexstein and M. Baumslag, "A unified approach to off-line permutation routing on parallel networks," in *ACM Symp. Parallel Algorithms, Architect.*, 1990, pp. 398–406.

[14] H. Wielandt, *Finite Permutation Groups*. New York: Academic, 1964.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP Completeness*. San Francisco, CA: W. H. Freeman, 1979.

[16] M. Tchuente, "Generation of permutations by graphical exchanges," *Ars Combinatoria*, vol. 14, pp. 115–122, 1982.

Prevention of Congestion in Packet-Switched Multistage Interconnection Networks

Jyh-Charn Liu, Kang G. Shin, and Charles C. Chang

***Abstract*—This paper proposes a simple, yet effective scheme to prevent congestion in a packet-switched multistage interconnection network (MIN) caused by *hot spots*. In this scheme, switches in the second and third stages of the MIN monitor their buffer occupancy to detect any notable nonuniform access behavior. When a switch detects congestion, packets generated by processors will be blocked from entering the congested switch until the congestion is cleared. Our scheme is compared with two well known schemes [1], [2], and shown to exhibit significantly better performance than these two.**

***Index Terms*—Congestion tree, packet-switching, multistage interconnection networks (MIN), hot spots, buffer occupancy.**

Manuscript received February 26, 1993; revised May 16, 1994. This work was supported in part by the Texas A&M College of Engineering Excellence Fund, and by the National Science Foundation under Grant MIP-9203895.

J.-C. Liu is with the Department of Computer Science, Texas A&M University, College Station, TX 77843 USA.

K. G. Shin is with Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109 USA.

C. C. Chang is with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77843 USA.

IEEE Log Number 9409874.

I. Introduction

The multistage interconnection networks (MIN's) are an important class of high-bandwidth networks that can be used for multiprocessor/multicomputer systems. The MIN has a good performance when the network traffic pattern is uniform, so that network resources (link bandwidths and buffers) may be shared by different communication channels in a fair and efficient manner. However, when demands of different sources are not uniform, communication channels may not be able to share the network resources fairly. Even worse, if the service demand on a resource exceeds its capacity, packets blocked by this resource may build up quickly in the network. The blocking effect can quickly propagate backward, and most paths in the network will soon be blocked, thereby resulting in a severe drop of network throughput.

The network congestion caused by nonuniform traffic patterns is also called the *hot spot* problem in a multiprocessor system, where an MIN is used as the interconnection network between processors and memory modules. In this case, a hot spot is a memory module whose service rate is exceeded by the rate of requests to use it. Once a hot spot is formed, most packets in the network would be blocked, and the network throughput may drop to nearly zero. When a network becomes congested due to the presence of a hot spot in the network, the set of switches on the paths to the hot spot forms a *congestion tree* with the hot spot as the root and processors as the leaves. In a congestion tree, packets destined for the root are called the *hot packets* and the other packets are called *cold packets*.

Many researchers had studied the hot spot problem using stochastic models and simulations [3]–[7], [9], and proposed several control schemes to alleviate the resulting severe performance degradation. The hot spot problem was first observed in the IBM RP3 project [8], [11], and the *combining switch* was then proposed to handle excessive hot packets. Since this initial effort, several software [16], [17] and hardware based solutions [1], [2], [10], [12]–[15], [18], [19] to the hot spot problem had been proposed, and some of the representative solutions would be reviewed below.

In this paper, we propose a distributed algorithm for detection and prevention of network congestions caused by hot spots. Similar to the scheme proposed by Scott and Sohi [1], we adopt a feedback control principle to prevent network congestions. However, our congestion detection scheme uses the *buffer occupancy* for congestion detection, while Scott and Sohi's scheme uses a continuous stream of hot packets arriving at their destinations. Through extensive simulations we find that in an uncontrolled network, most packets are blocked in switches very close to network input. Therefore, the buffer occupancy of switches close to the network input is used for early detection of the congestions caused by hot spots. Each buffer-occupancy monitor controls a set of processors, and occupancy monitors (at the same stage) make packet-blocking decisions independently. Therefore, at any instant, only a fraction of processors in the system can (not) input packets to the network. Our scheme thus has a much smoother throughput than other schemes. We compare the performance of our scheme with those of the two schemes proposed in [1], [2]. Our simulation results show that the three schemes have a similar behavior when the hot access rate is low. However, our scheme has a much more stable, and significantly better performance, than those of the other schemes at higher hot-access rates.

II. Detection and Prevention of Network Congestion

The MIN under consideration is assumed to be constructed with 2×2 switches. Each input port of a switch has a packet buffer. The Banyan, or any other topologically-equivalent topology of Banyan, is assumed for the network. Processors are connected to the network input ports, and memory modules are connected to network output ports. It is assumed that the duration of a memory cycle is the same as a network cycle, and a memory module can serve only one memory-access request in one memory cycle.

Network congestion is a statistical phenomenon, and its effect is measured by how fast it affects the performance of a network. Thus, we define *network congestion* as a condition under which the communication delay of cold packets becomes very large because of excessive demands of hot packets for network resources. A *congested area* is composed of a congested switch and those switches that have forward paths to the congested switch. In a congested area, the number of cold packets may not be significantly smaller than that of hot packets, but both cold and hot packets are being blocked for a very long time, and hence, the throughput of the area is very low. Based on the above definition, switches close to a hot spot are not considered as congested, because although most packets are destined for the hot spot, the throughput of switches in this area may be relatively higher than that of the rest of the network. It is more appropriate to consider the network input as the congested area, since most cold packets in an uncontrolled and congested network are blocked close to network input. In this case, hot spots cause the network congestion.

Our design prevents network congestion based on an optimal combination of *congestion-detection*, *congestion-resolution* and *congestion-blocking*. Our objective is to maintain a stable network throughput and to provide good quality of service to these packets already in the network. A distributed control approach is helpful to achieve this objective. For congestion detection we need to define the optimal conditions under which the other two schemes should be activateded to handle congestion. The congestion-detection scheme must consistently detect congestion, whether the congestion-resolution and blocking mechanisms are activated or not. The congestion-resolution scheme is used to handle packets in the network after detecting a network congestion. This could be done through specially designed switches, or dynamic assignment of packet priority, or simply doing nothing. The congestion-blocking scheme is used to stop packets from entering the network at proper time instants. The efficiency and simplicity of these schemes are particularly important, so that they can be used in high-speed networks without appreciable interference with the normal network operation. As it will become clear later, we only need to use the congestion-detection and congestion-blocking mechanisms for our design. Hence, we will not consider the congestion-resolution scheme any further.

Explicit recognition of excessive packets destined for a particular memory module [1] is the most commonly-suggested approach for detecting a hot spot. This approach was shown to offer a fairly good performance when the hot-access rate is low, but its performance deteriorates rapidly with increase of the hot-access rate, as observed in our simulation. To avoid this problem, the switch buffer occupancy for congestion detection. The occupancy of a buffer increases only when the packet input rate is higher than the packet output rate of the corresponding switch. A continual increase of the buffer occupancy in a switch implies that the buffer space of the switch may soon be used up and all inbound packets would be blocked. Therefore, either the packet input rate to the congested area should be lowered, or the output rate of the switch should be increased, until the buffer occupancy drops to an acceptable level. This is true even when the congestion-blocking mechanism is activated.

Next, we decide on how and where to detect network congestions to maximize the network throughput and minimize the packet delay. To reduce the possibility of false alarms, we use different rules for congestion detection *before*, *during*, and *after* the network becomes congested due to the hot spot(s). In an uncontrolled network, if the network becomes congested due to a hot spot, most switches can be in one of the two states: heavily-loaded or under-loaded. Switches would be heavily-loaded if they are on the congestion tree, or would be extremely under-loaded otherwise. The under-loaded switches are called the *free switches*, and they are empty for most of the time, because most packets destined for them have been blocked in the congestion tree.

Through interactive graphical simulations, we observed that, when a congestion tree is being formed, the buffer occupancy of switches on the congestion tree becomes sharply different from those of their adjacent free switches in a very short time period. Therefore, the difference in the buffer occupancies in adjacent switches (of certain adjacent stages) is used as the congestion detection rule, called *rule A*, *before* the congestion-blocking mechanisms are activated. This change in the buffer occupancy of adjacent switches becomes less dramatic after the congestion-blocking mechanisms are activated (to block excessive packets). Thus, even though *rule A* could still be used for congestion detection, the network throughput may fluctuate. Therefore, *during* the time when congestion-blocking mechanisms are activated, we propose to use the buffer occupancy of each individual switch as the congestion-detection rule, called *rule B*. If the buffer occupancy of a switch exceeds a given threshold then it implies that the congestion-blocking mechanism related to this switch is allowing too many packets to enter the network, and hence, packets should be blocked until the buffer occupancy drops to a lower value.

Finally, after the hot accesses cease, we need to de-activate the congestion-blocking mechanisms to avoid unnecessary throughput loss due to false detection of transient traffic fluctuations. This post-congestion detection rule, called *rule C*, can be defined as the condition when no congestion is detected for a chosen period of time. The three congestion detection rules are formally summarized as follows.

Rule A: The occupancy-monitor of a passive CUTH detects congestion if the difference between the buffer occupancies of its adjacent switches at the next stage exceeds the last recorded ρ_h value.

Rule B: The buffer-occupancy monitor of an active CUTH detects congestion, if the buffer occupancy of the switch in which the monitor resides becomes higher than ρ_h, then its controlled flow-throttles begin to block packets from entering the congested area. The flow-throttles allow packets to enter the network if the reading of the occupancy monitor becomes lower than a prespecified value ρ_ℓ, $\rho_\ell < \rho_h$.

Rule C: The hot access is deemed to *cease*, if no congestion is detected for a pre-specified period by an active CUTH.

The above three congestion-detection rules are integrated with congestion blocking mechanisms on a distributed control structure, called *ocCUpancy-monitors and flow-THrottles* (CUTH). Each CUTH consists of one buffer-occupancy monitor and a set of flow-throttles placed in processors, each of which has a path to the buffer-occupancy monitor. The buffer-occupancy monitors are placed in switches located at stages close to the network input, and thus, each buffer-occupancy monitor controls only a small number of flow-throttles. CUTH's can overlap with one another, and a flow-throttle must follow packet-blocking commands sent by any occupancy monitors, since the flow-throttle can be on the congestion trees of several hot spots.

Before congestion is detected, a CUTH is in the *passive* state, and it does not block packets from entering the network. Occupancy monitors periodically sample and store the buffer-occupancy level ρ_h, as long as *Rule A* does not detect any congestion. The parameter ρ_h is taken as the background traffic intensity, which will be later used as a reference in determining the existence of a network congestion by *Rule B*, when the congestion-blocking mechanisms are activated. A passive CUTH enters the *active* state once it detects a congestion condition. Then, *Rule B* is used for congestion detection by the occupancy monitor, and the flow-throttles begin to block or allow packets to enter the congested area based on the detection results from the occupancy monitors.

Placing flow-throttles in processors is a relatively easy decision, because by blocking excessive packets from entering the network, we can resolve network congestion without modifying the switch design. The difficult design issues are how to embed occupancy monitors into switches, and how to coordinate them with flow-throttles. These design issues need to be done through a careful analysis of the network topology and the congestion behavior, as discussed next.

To implement the congestion detection rules *B* and *C*, the occupancy monitor in a switch needs to examine its own buffer occupancy. To implement *rule A*, we note that any switch on an arbitrary congestion tree is also connected to a free switch through one of its output ports. That is, any switch on a congestion tree has one output port connected to a switch on the congestion tree, and the other connected to a free switch. If the switch size is $r \times r$, then a switch on a congestion tree has one of its output ports connected to another switch on the same congestion tree, and all other output ports connected to free switches. Therefore, to implement *rule A*, the occupancy monitor in a switch also needs to be connected to the occupancy monitors of its adjacent switches at the next stage. Moreover, the occupancy monitor of a switch may also need to inform its upstream neighboring switches of its status as either *congested*, or *not-congested*. This way the congestion condition can be propagated to the processors to control traffic flow.

A. Hardware Implementation

We propose a sliding-window based CUTH design, where the window size is w clock cycles. The occupancy monitor can be implemented with a w-stage shift-register array and a counter, where w is the window size. Each register is $m = \log_2(q+1)$ bits wide, where q is the number of buffers in a packet queue. The number of packets in a packet queue is sampled into the shift-register array of the occupancy monitor, and the register array is shifted forward one stage in each clock cycle. The counter storing the buffer occupancy is updated to be $C = C + v_i - v_o$ in each cycle, where C is the current value of the counter, and v_i and v_o are the input and output of the shift-register array, respectively.

If an occupancy monitor is in the passive state, the current counter value is stored into its memory M periodically as ρ_h. When the occupancy monitor becomes active, and the value of its counter becomes higher than ρ_h, all the flow-throttles in the same CUTH will have to be activated to block packets from entering the congested area. When an occupancy monitor detects congestion, it needs to inform all processors that can reach the switch of its location, so that flow-throttles can block packets destined for the congested area. This communication mechanism can be implemented in two different ways. In the first approach, each switch is assigned a *subspace* ID, which indicates the address subspace of the memory modules that can be reached by the switch [20]. Counting from input to output stages, a switch at the ith stage has i bits of subspace ID. When the occupancy monitor at stage i detects a congestion, it can send a message, which contains its subspace ID, back to all processors that can reach this switch. The flow-throttle in a processor blocks any messages whose addresses are in the received subspace ID(s). If several hot spots become active at the same time, then packets can be divided into groups based on the subspace ID's of the (detected) congested areas. A new hot-spot group is added to the processor after a new hot spot is detected. The hot-spot group can be deleted when *rule C* with respect to the hot spot detects the disappearance of the hot spot. The overhead in dealing with the hot packets in the communication buffer of the processor should not impose any serious performance penalty. This is because when hot accesses begin, the *effective packet generation rate* will drop to lower than the original packet generation rate of processors. The drop of packet generation rate can be substantial for most existing schemes, as our simulation results indicate. Therefore, our scheme outperforms many existing schemes as long as the time for sorting packets is smaller than the average packet interdeparture rate (from the processor buffer to the

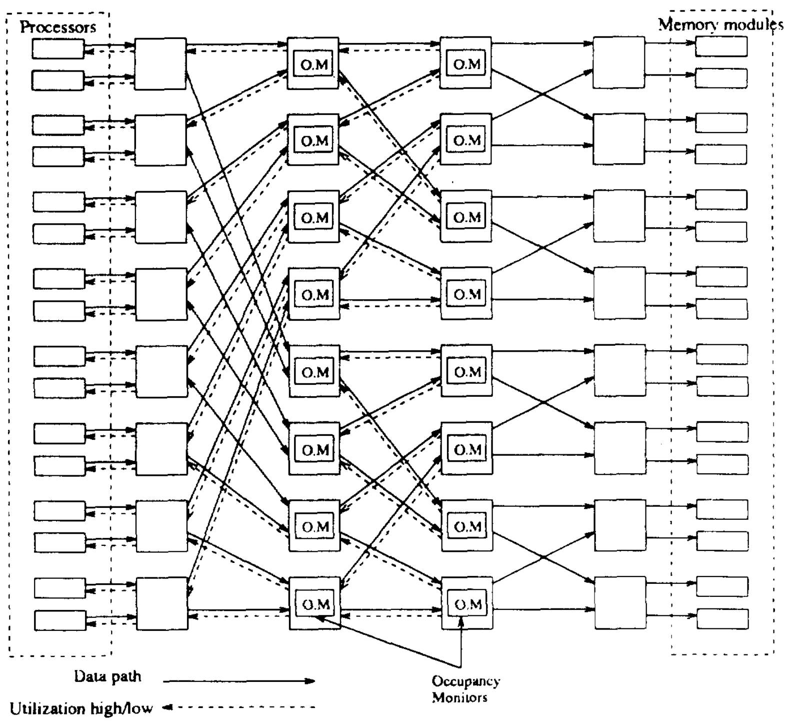

Fig. 1. The configuration of CUTH's implemented in a 16 × 16 MIN.

network), which is also called the *effective packet departure rate.*

An alternative design for coordination of occupancy monitors and flow-throttles is by direct propagation of control signals. To pass congestion condition from an occupancy monitor to its flow-throttles, an active occupancy monitor located at stage h asserts a *block*-signal to its neighbors at stage $h-1$. A switch S_x at stage $h-1$ that receives an asserted block-signal propagates it backward to its neighbors at stage $h-2$, with one additional *hot-address* line denoting the outport of S_x that is connected to the asserted occupancy monitor. Then, S_{y1} and S_{y2}, both of which are connected to the input ports of S_x, continue to propagate the block-signals and the hot-address signals one more stage, after adding one more bit denoting the outport of $S_{y1}\#(S_{y2})$ that is connected to S_x, to the received hot-address lines. This approach is more suitable for the case when the occupancy monitors are very close to processors. Since our simulation results show that placing the occupancy monitors on stages 2 and 3 yields best performance even under extremely high hot-access rates, the hardware costs of both schemes are reasonably low. A 16 × 16 MIN with embedded CUTH's is illustrated in Fig. 1.

To reduce the packet blocking effect under congestion, we modify the conventional first-in-first-out (FIFO) buffers into a *parallel-access-buffers* (PAB) structure. Based on the PAB structure, cold packets need not be blocked in congested switches in the presence of hot spots. The circuit design of the PAB is illustrated in Fig. 2, in which a *packet-selector* is associated with each buffer in the PAB to control movement of the packet, if any. RD$_i$ denotes the existence of a packet in the ith buffer of a PAB. The *Packet_in, In-enable, queue-full* and RD$_i$'s together decide where to insert a new packet into the PAB. This part of control circuits is also necessary for conventional FIFO-buffers. The rest of control signals are used for dynamic routing of blocked packets as follows. Whenever two packets from the head of the two PAB's contend for the same output port, only one of the two requests will be granted, and the lost PAB is allowed to route another packet, if any, destined for the other output port. To realize this operation, a tristate transmission gate array needs to be added to the output of each packet buffer, in addition to control signals. At the beginning of each routing cycle, a PAB issues a *request* signal, which can be the RD bit of the leading buffer, to the switching arbitrator. In the meantime the routing tag of the leading packet is also made available to the entire PAB. If the PAB wins the contention, then its leading packet is routed to the next stage. Otherwise, the *Alternate* signal will be set to 1 and propagated backward to the PAB. The first packet buffer with Alternate_in = 1, which has a different tag value from that of the leading packet, should assert its out-enable to route its packet to the switch at the next stage. In the meantime, the packet buffer needs to set its alternate_out to 0. Packets behind the routed packets will be shifted up one register after the grant-signal returns. A buffer can accept a shifted-up packet if its *Advance* signal is asserted. After a new packet arrives, the empty buffer next to the last occupied buffer will receive the packet and change its RD bit accordingly. Note that the additional complexity of the PAB over the FIFO buffers is limited, because only a small amount of combinational circuits would be needed to implement the algorithm. This way network resources can be released for servicing other packets in the shortest time possible. We also note that the buffer-occupancy monitors are quite flexible, because they can be embedded into any existing switch design in a nonintrusive manner. (Their interference with the normal operation of switches is minimal.) On the other hand, most existing, if not all, hot spot prevention algorithms are based on the detection of a large number of packets destined for a particular network output port. This usually leads to a solution which indeed tries to resolve the congestion already formed in the network.

III. Performance Comparison

We compare the performance of our scheme against that of two simple yet effective schemes proposed in [1] and [2] by simulation.

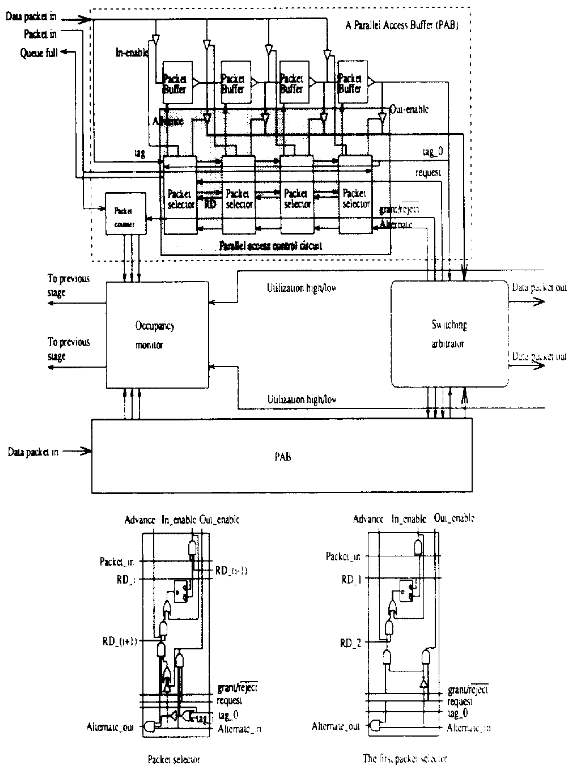

Fig. 2. Network throughput and packet delay with eight hot memory modules ($N = 512, p = 0.7, h = 0.1$).

Since the two schemes being compared were found to have similar behaviors, we explain only the experiment setup of the feedback control scheme [1]. In the feedback control scheme, a memory module was detected as a "hot spot," if the number of consecutive packets destined for the memory module became greater than a threshold value T, where the optimal value of T was suggested in [1] to be 4. Although this scheme was designed only to handle a single hot spot at a time, for fair comparison we assumed that it could detect multiple hot spots, and the "limiting-damping" scheme was incorporated in the simulation for its better performance. In all simulation runs, packet routing is synchronized, i.e., routing of packets in all switches is driven by one global clock. Each clock period is one network cycle, and it takes one network cycle to route a packet from one stage to next.

Simulation results of different combinations of cold and hot packet rates have been obtained while varying parameters queue size, ρ_h, and ρ_ℓ. Due to the extremely large run-time and data storage requirements, we present only a few representative cases. In examples given below, the MIN connects 512 processors to 512 memory modules. The cold-packet generation rate is set to 0.7, and processors initially generate only cold packets for 100 cycles, and then generate hot packets for 100 to 800 cycles. The hot accesses cease after 800 network cycles. The generation rates of hot packets are set to different values in different experimental runs.

A critical issue that needs to be examined is where to embed the buffer-occupancy monitors. The system performance was not good if buffer-occupancy monitors and input blocking were embedded in the whole network. However, by placing the buffer-occupancy monitors only at the second and third stages, and the flow-throttles in the processors, we obtain the best performance in all the cases studied. This is because we can detect the blockage of cold and hot packets caused by the hot spot before the whole network becomes congested. Also, the downstream stages of the network are intentionally left uncontrolled so that all the hot packets already in the network can depart from the network at the maximum network service rate.

Our experimental results show that, when the window size is 5 to 7 cycles long, congestion can be prevented with relatively little fluctuation in the network throughput and packet delay. When the window size was set to 8 cycles or longer, the control algorithm could not respond to the formation of congestion tree in time, and a notable throughput loss was registered. On the other hand, if the window size was set to 4 cycles or shorter, the network throughput fluctuates, because of over-reaction to transient traffic fluctuation.

The parameter ρ_h is dynamically sampled from the background traffic when the CUTH is at its passive state. The value of ρ_ℓ reflects the condition that allows new packets to enter an area that was detected to have been congested. An interesting observation is that for a wide range of values, ρ_ℓ has relatively little impact on the network throughput under the congestion condition, but it has a significant impact on the packet delay, as confirmed in numerous runs of simulation. The use of a smaller ρ_ℓ implies that we allow packets to enter a previously-congested area only after much of the congestion had disappeared. The use of a large ρ_ℓ value implies that we allow packets to enter the previously-congested area shortly after the congestion condition eased up. Therefore, with a high ρ_ℓ value the packet delay tends to increase (moderately fast). But the packet delay for hot spots remains stable if a low ρ_ℓ value is used, since packets already in the network will receive a better service.

The effect of ρ_ℓ value on the network throughput was very limited, because, when a congested network is being controlled, a large number of packets will likely be blocked at processors. Thus, whenever a throttle allows packets to enter the network, a large number of packets will try to enter the network, thus increasing the network throughput. In the meantime, whenever the congestion condition is detected (based on the ρ_h value), the flow-throttle will block packets. Therefore, the network throughput can be sustained for a wide range of ρ_ℓ values. Moreover, the throttles operate in a distributed manner. At any instant only some of the flow-throttles are blocking packets, thus making the network throughput much more stable than that of any centralized algorithm. From our experiments, we found that the packet delay becomes notably high and fluctuating if $\rho_\ell > 0.3$, but its effect on the network throughput was insignificant. As a result, the ρ_ℓ value was set to 0.1 to achieve a high network throughput with negligible fluctuation in the routing delay.

We observed that the two schemes being compared worked well when the hot-packet request rates were relatively low, but their network throughputs were reduced to 0.1 when the hot-packet request rate from each processor is greater than 0.1. For Scott and Sohi's scheme, this is mainly because of the relatively slow detection of hot spots, and thus, at a high hot-access rate; there were already too many hot packets in the congestion tree before detecting a hot memory, as pointed out in [1]. For Peir-Lee's scheme, hot packets could enter the network even after a hot spot was detected, thus making it difficult to dissipate the hot packets already in the network when the hot-packet generation rate was high. When the hot-memory request rate was 0.03 or lower, our scheme was only slightly better than the other two schemes being compared. However, our control scheme was shown to be much better than the other two when the hot-memory request rate was higher than 0.03, regardless of the number of hot spots in the system. The network throughputs and packet delays under the three schemes for a 512×512 MIN are plotted in Fig. 3, in which p and h denote the cold and hot packet generation rates by each processor, respectively. In addition, buffer-occupancy monitors are placed at the second and third stages. Since

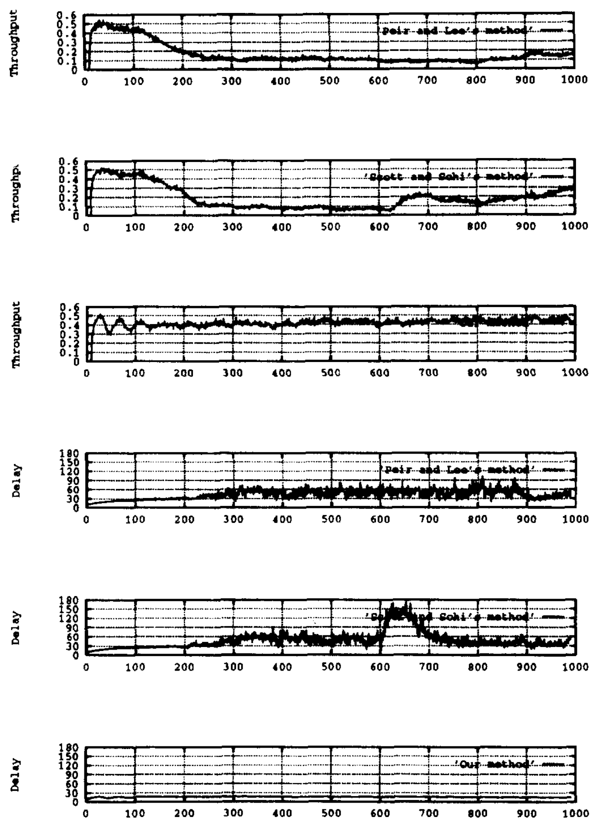

Fig. 3. Network throughput and packet dealy with eight hot memory modules ($N = 512, p = 0.7, h = 0.1$).

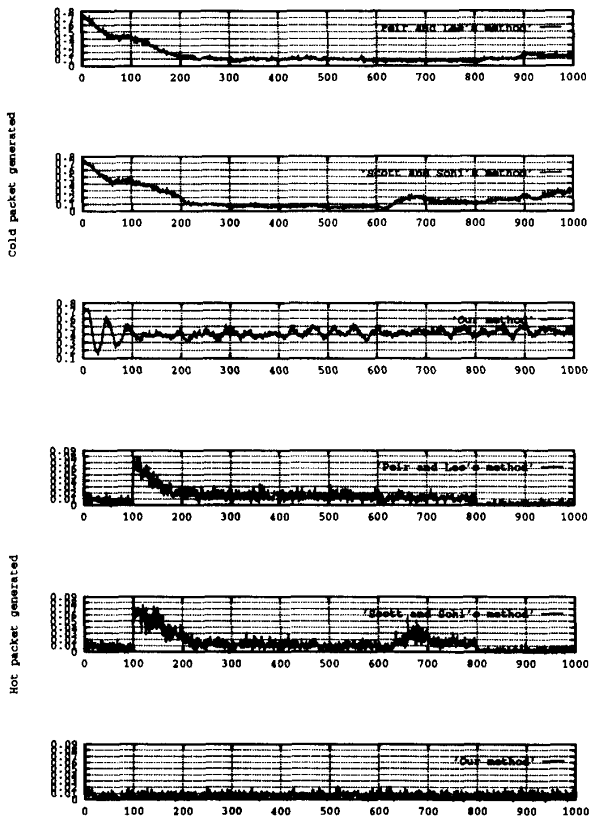

Fig. 4. Cold and hot packets generated under different schemes with eight hot memory modules ($N = 512, p = 0.7, h = 0.1$).

the network was far less congested in our scheme, the *effective* (*cold*) *packet generation rate*—which was the actual (cold) packet output rate from processors—under our scheme was much higher than that of the two schemes being compared, as can be seen from Fig. 4. In these figures, p denotes the background packet generation rate, and h denotes the hot-packet generation rate in each processor. After comparing our scheme with the other two, we proceeded to study the performance impact of network sizes on our control scheme, and we observed little performance difference.

All the performance results reported so far are based on the assumption that it takes one cycle network for the congestion monitor to inform the flow throttles. The network performances under different combinations of the window size and message delay are also evaluated through simulation. When the message delay was in the range of 1 to 4, no notable performance degradation was observed. So, it was important to keep the message delay as short as possible.

IV. Conclusion

We have introduced a cost-effective congestion detection and prevention scheme for packet-switched MIN's. Our simulation results show that our scheme can sustain the network throughput under extremely nonuniform traffic. The effectiveness of our scheme stems from the facts that 1) it detects network congestion caused by hot spots, instead of the hot spots themselves, and 2) excessive packets are not allowed to enter the network, so that packets blocked in the network can be dissipated quickly. Our scheme can be implemented with simple, inexpensive hardware mechanisms, making it an attractive solution to prevention of network congestion.

References

[1] S. L. Scott and G. S. Sohi, "Using feedback to control tree saturation in multistage interconnection in networks," in *Proc. 16th Int. Symp. Comput. Architect.*, 1989, pp. 167–176.

[2] J. K. Peir and Y. H. Lee, "Improving multistage network performance under uniform and hot-spot traffics," in *Proc. IEEE Symp. Parallel Distrib. Process.*, 1990, pp. 548–551.

[3] C. P. Kurskal, M. Snir, and A. Weiss, "On the distribution of delays in buffered multistage interconnection networks for uniform and nonuniform traffic," in *Proc. 1984 Int. Conf. Parallel Process.*, 1984, pp. 215–219.

[4] N. M. Patel and P. G. Harrison, "On hot-spot contention in interconnection networks," *Perform. Eval. Rev.*, vol. 16, no. 1, pp. 114–123, 1988.

[5] T. H. Lee, "Throughput performance of multistage interconnection networks in presence of multiple memory hot spots," *Electron. Lett.*, vol. 26, pp. 536–537, Apr. 1990.

[6] P. G. Harrison, "On nonuniform packet switched delta networks and the hot-spot effect," *IEE Proceedings*, vol. 138, pp. 123–130, May 1991.

[7] S. Edirisooriya and G. Edirisooriya, "New traffic model for performance analysis of processor-memory multistage interconnection networks," *Electron. Lett.*, vol. 27, pp. 1813–1816, Sept. 1991.

[8] G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-34, pp. 943–948, Oct. 1985.

[9] M. Kumar and G. F. Pfister, "The onset of hot spot contention," in *Proc. 16th Int. Conf. Parallel Process.*, Aug. 1986.

[10] D. M. Dias and M. Kumar, "Preventing congestion in multistage networks in the presence of hotspots," in *Proc. 18th Int. Conf. Parallel Process.*, 1989, pp. 9-13.
[11] G. F. Pfister *et al.*, "The IBM research parallel processor prototype (RP3): Introduction and architecture," in *Proc. IEEE 1985 Int. Conf. Parallel Process.*, Aug. 1985.
[12] T. Lang and L. Kurisaki, "Nonuniform traffic spots (NUTS) in multistage interconnection networks," in *Proc. 17th Int. Conf. Parallel Process.*, Aug. 1988.
[13] W. Ho and D. Eager, "A Novel strategy for controlling hot spot congestion," in *Proc. Int. Conf. Parallel Process.*, 1989, pp. 14-18.
[14] N. F. Tzeng, "Design of a novel combining structure for shared-memory multiprocessors," *Proc. Int. Conf. Parallel Process.*, 1989, pp. 51-58.
[15] ——, "An approach to the performance improvement of multistage interconnection networks with nonuniform traffic spots," in *Proc. 1991 Int. Conf. Parallel Process.*, Oct. 1991, pp. 542-545.
[16] P. C. Yew, N. F. Tzeng, and D. Lawrie, "Distributing hot-spot addressing in large-scale multiprocessors," in *Proc. Int. Conf. Parallel Process.*, 1986, pp. 51-58.
[17] ——, "Distributing hot-spot addressing in large-scale multiprocessors," *IEEE Trans. Comput.*, vol. C-36, pp. 388-395, Apr. 1987.
[18] L. G. Valiant, "A Scheme for fast parallel communication," *SIAM J. Comput.*, vol. 11, pp. 350-361, May 1982.
[19] Y. Tamir and G. Frazier, "High-performance multiqueue buffers for VLSI communication switches," in *Proc. 15th Int. Symp. Comput. Architect.*, May 1988, pp. 343-354.
[20] C. L. Wu and T. Y. Feng, "Tutorial: Interconnection networks for parallel and distributed processing," *IEEE*, 1984.
[21] S. Chalasani and A. M. Varma, "Method and apparatus for dynamic detection and routing of nonuniform traffic in parallel buffered multistage interconnection networks," U.S. Pat. 5 274 782, Dec. 1990.

Properties of Generalized Branch and Combine Clock Networks

Ahmed El-Amawy and Priyalal Kulasinghe

***Abstract*—In a recent development a new clock distribution scheme has been introduced. The scheme called Branch-and-Combine or BaC, is the first to guarantee constant skew bound regardless of network size. In this paper we generalize and extend the work on BaC networks. Our study takes the approach of defining a general graph theoretic model which is then utilized to define a general network model taking into account node function. We use the models to establish some interesting results on clocking paths, node input sequences, node inputs' relative timings, and skew bound. We prove that a network adhering to our general model is stable (will not oscillate) despite its cyclic nature. We also prove that no tree of any kind can be used to distribute the clock in two or more dimensions such that skew bound is constant. The paper then exploits the derived properties to describe the inherent interplay between topology, timing, node function, and skew bound.**

***Index Terms*—Branch-and-combine network, clock distribution, skew bound, synchronous system, VLSI, large system, network stability, cyclic clock networks.**

Manuscript received April 26, 1993; revised July 12, 1994. This work was supported in part by NSF under Grant MIP-9117206 and in part by NSF/LEQSF under Contract ADP-04-1992-6.

The authors are with the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803 USA (e-mail: amawy@gate.ee.lsu.edu).

IEEE Log Number 9409875.

I. Introduction

In order to meet the ever increasing demand for higher computing power, the size and complexity of computing systems have grown steadily. Clock skew has been widely recognized as a major problem in implementing very large synchronous systems. Typically, a single global clock is used to provide the basic timing and synchronization mechanism for different parts of the system. Due to several factors such as variation in buffer delays, variation in signal propagation delays on wires, and different threshold voltages, clocking events generally reach clocked cells at different times. The maximum difference in arrival times at two cells, which directly communicate, is what we mean by clock skew. Many researchers have investigated the clock skew problem as exemplified by the work in [1]–[8], [10]–[13]. In one study Fisher and Kung [7] have shown that if an H-Tree is used to clock a 2-D mesh of processors, it is impossible to guarantee a constant skew upper bound, provided that small delay variations are not ignored.

In a recent development, El-Amawy [3], [5] introduced a novel clock distribution scheme called *Branch and Combine* (or BaC for short) which is the first to guarantee a constant upper skew bound irrespective of network size. The scheme distributes the global clock through a set of simple interconnected nodes whose main function is to process the clock signal such that it adheres to certain timing constraints. In [3]–[5], three BaC network designs for clocking a 2-D mesh of processors have been introduced. A single network was analyzed in detail in [5] and shown to be stable under the assumed model. Also skew was shown to be bounded above by a constant for each of the three networks. In [6], BaC clocking was compared to H-tree clocking of a 2-D mesh of processors in a VLSI context. That study shows that the BaC clocking is superior to the H-tree in almost all aspects, for medium and large size meshes.

In this paper we extend, generalize, and improve on the previous work on BaC networks. We define a general class of BaC networks. We do not confine our study to any particular topology or dimensionality. Instead we base our study on a graph theoretic model which only requires that each pair of adjacent vertices be included in a directed cycle of finite length. We define a network model which specifies the underlying assumptions such as node function and clock signal constraints which are dependent on the properties of the underlying graph representation. We then utilize the network model to derive important properties of the generalized class of BaC networks. We prove that a network adhering to our general model is stable (will not oscillate) despite its cyclic nature. We prove that nodes will be triggered via the shortest delay paths from the source and that clocking events will reach nodes in proper order separated in time by at least $n\Delta$, where Δ is the maximum delay through a node and one of its output links. We further prove that if small delay variations are not ignored, it is impossible to guarantee constant upper bound on clock skew in the absence of cycles in any clock distribution network of two or more dimensions. This establishes the necessity of cycles in the clock distribution networks clocking large data networks; which is the characterizing feature of BaC networks. The implication of this is that it will not be possible to design a clock network which provides constant upper bound on skew in m-D networks ($m > 1$) using a distribution tree of any kind.

Based on the derived properties we are able to predict maximum clocking rates which are almost double those predicted in [5], [8]. We describe a simple method for achieving these new bounds on maximum clocking rates for BaC networks.

Section II introduces and defines the generalized graph and network models. Section III represents the main body of the paper. In it we