Evaluation of Load Sharing in HARTS with Consideration of Its Communication Activities

Kang G. Shin, *Fellow, IEEE*, and Chao-Ju Hou, *Member, IEEE Computer Society*

Abstract—We rigorously analyze load sharing (LS) in a distributed real-time system, called HARTS (Hexagonal Architecture for Real-Time Systems), while considering LS-related communication activities, such as task transfers and state-change broadcasts.

First, we give an overview of the general distributed real-time LS approach described in [1], [2], and then adapt it to HARTS by exploiting the topological properties of HARTS. Second, we model task arrival/completion/transfer activities in HARTS as a continuous-time Markov chain from which we derive the distribution of queue length and the rate of generating LS-related traffic—task transfer-out rate and state-region change broadcast rate. Third, we derive the distribution of packet delivery time as a function of LS-related traffic rates by characterizing the hexagonal mesh topology and the virtual cut-through capability of HARTS. Finally, we derive the distribution of task waiting time (the time a task is queued for execution plus the time it would spend if the task is to be transferred), from which the probability of a task failing to complete in time, called the *probability of dynamic failure*, can be computed.

The results obtained from our analytic models are verified through event-driven simulations, and can be used to study the effects of varying various design parameters on the performance of LS while considering the details of LS-related communication activities.

Index Terms—Dynamic failure, distributed real-time systems, wrapped hexagonal mesh, virtual cut-through, point-to-point broadcasts, adaptive load sharing, queuing models, performance analysis.

1 Introduction

Due to their potential for high performance and high reliability, distributed systems are considered most suitable for real-time applications. To realize part of this potential, at the Real-time Computing Laboratory of the University of Michigan we are currently building an experimental distributed real-time system, called HARTS (Hexagonal Architecture for Real-Time Systems) [3].

Although there are many potentially attractive features of distributed systems, they cannot be realized without careful coordination of processing nodes in the system. For example, non-uniform and/or bursty task arrivals at processing nodes may temporarily overload some nodes while leaving some other nodes idle. This problem can be alleviated by enabling idle/underloaded nodes to share the loads of overloaded ones, which is termed *load sharing* (LS).

LS for general-purpose distributed systems has been addressed by many researchers [4], [5], [6], [7], [8], [9], 10], [11], [12], [13], [14], [15]. In particular, Eager et al. and Shivaratri et al. characterized LS with three component policies: the *transfer policy* which determines **when** to transfer a task, the *location policy* which determines **where** to transfer the task, and the information policy which determines **how** each node exchanges state information. Recently, there has been considerable interest in developing LS algorithms for real-time applications [1], [2], [16], [17]. The LS requirements for real-time applications differ significantly from those of traditional nonreal-time LS. First, real-time LS performance is assessed on a *per-task* basis, whereas nonreal-time systems deal with *average* performance. Second, a task that is not completed within a certain time limit (called the *deadline*) after its invocation is considered failed, regardless whether it is eventually completed or not. One consequence of these differences is that the primary performance objective is no longer to minimize *average response time*, but rather to minimize the probability of a node failing to complete a task before its deadline, called the *probability of dynamic failure*, P_{dyn} [18].

The above differences suggest that the transfer policy in a real-time system should not be of the *static* threshold type commonly used for nonreal-time systems [8], [14], but should instead depend on the *laxity* of each task, or the latest time the task must start execution in order to meet its deadline. In other words, upon arrival of a task, a node determines whether or not it can complete the task in time. Similarly, a node with an "overflow" task—a task that cannot be completed locally in time—locates a candidate node for task transfer using the information on whether or not the candidate node has the ability to complete the task in time. To this end, all LS approaches need to gather the workload information (or *state*) of other nodes by either periodic information exchange, state bidding/probing, or state-change broadcasts.

No matter which information collection strategy or location policy is used, we must consider an underlying communication subsystem responsible for exchanging messages or transferring tasks. In particular, the interconnection network affects how tasks or broadcast-messages are routed, and the underlying switching scheme determines whether tasks/messages may be queued at intermediate nodes or not. Consequently, we must consider the under-

- *K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, MI 48109-2122.*
- *C.-J. Hou is with the Department of Electrical and Computer Engineering, the University of Wisconsin-Madison, Madison, WI 53706. E-mail: jhou@ece.wisc.edu.*

For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95186.

lying communication subsystem that supports all LS-related communications.

In this paper, we adapt the concept of *buddy sets*, *preferred lists*, and *region-change broadcasts* proposed in [1], [2] to HARTS [3]. The nodes in HARTS are interconnected by a C-wrapped hexagonal mesh[1] and coordinated to evenly share "overflow" tasks. The HARTS routing and broadcasting algorithms in [19], [20] are used for transferring tasks and broadcasting state-changes. The virtual cut-through switching scheme[2] [21] implemented in HARTS [22] is used for inter-node communication. Moreover, by exploiting/integrating features of these algorithms for/into LS, we rigorously analyze the performance of LS in HARTS while considering all LS-related communication activities.

We first construct a continuous-time Markov chain to describe task arrival/transfer/completion activities under the proposed LS approach. Second, we derive the traffic overheads introduced by LS (i.e., the rate of task transfer and the rate of state-change broadcast) from the Markov chain. Then, using the LS traffic rates as input and characterizing the hexagonal mesh topology and the virtual cut-through switching scheme implemented in HARTS, we construct a queuing network model from which the distribution of packet delivery time is derived. Finally, we derive the distribution of task waiting time (i.e., the time a task is queued for execution plus the delay the task experiences if it is to be transferred), from which the probability of dynamic failure can be computed.

The impact of communication activities/delays on LS was first analyzed in [14]. They developed a continuous-time Markov chain for LS algorithms with state probing, and characterized task-transfer delays with some percentage of mean task service time, but did not consider the underlying communication topology and the switching scheme used. Shin et al. [2], [23] applied Bayesian analysis to alleviate the negative effect of communication delays on maintaining up–to–date state information. By on-line collection of time-stamped state information, they calculated for each node the posterior distribution of other nodes' true state given the corresponding observation. Although they considered the hypercube as the underlying interconnection system and included the effect of possible task/message queuing at intermediate nodes on the way to the destination node of each transfer/broadcast, their performance evaluation was solely based on simulations. To our best knowledge, this is the first attempt to consider the underlying communication subsystem along with its task transfer, state-change broadcasting, and message-passing functions as an integrated part of the LS mechanism, and **analytically** evaluate the integrated LS performance using P_{dyn} as a yardstick.

The rest of the paper is organized as follows. For completeness Section 2 gives a brief description of HARTS along with its routing and broadcasting algorithms, and its switching scheme. The LS algorithm proposed in [1], [2] is also described and adapted to HARTS in Section 2. Section 3 deals with an integrated performance analysis of the proposed LS approach, the underlying communication subsystem, and the interaction between them. Section 4 describes a simulator for modeling the LS operations in HARTS (and for verifying our analysis), and presents some representative analytic/simulation results. Section 5 concludes this paper.

1. To be defined in Section 2.1.
2. To be discussed in Section 2.1

2 System Model and Load Sharing Mechanism for HARTS

2.1 Overview of HARTS

HARTS is an experimental distributed real-time system being built at the Real-Time Computing Laboratory, the University of Michigan [3]. A set of Application Processors (APs) along with a Network Processor (NP) form a node of HARTS. These nodes are interconnected via a C-wrapped hexagonal mesh topology. The APs execute computational tasks, and the NP (which contains a custom-designed router, buffer memory, a RISC processor, and the interface to APs) handles both intra- and inter-node communications.

Specifically, a C-wrapped hexagonal mesh (H-mesh) can be defined succinctly as follows.

DEFINITION 1. *A C-wrapped hexagonal mesh of dimension e, denoted as H_e, is comprised of $M = 3e(e-1)+1$ nodes, labeled from 0 to $M-1$, such that each node k has six neighbors $[k+1]_M$, $[k+3e-1]_M$, $[k+3e-2]_M$, $[k+3e(e-1)]_M$, $[k+3e^2-6e+2]_M$, and $[k+3e^2-6e+3]_M$, where the dimension of an H-mesh is defined as the number of nodes on a peripheral edge of the H-mesh, and $[a]_b$ denotes a* mod *b.*

C-type wrapping is performed in the following three steps:

S1. Partition the nodes of a nonwrapped H-mesh of dimension e into rows in three different directions, d_0, d_1, d_2. The mesh can be viewed as composed of $2e-1$ horizontal rows (the d_0 direction), $2e-1$ rows in the 60 degree clockwise direction (the d_1 direction), or $2e-1$ rows in the 120 degree clockwise direction (the d_2 direction).

S2. Label from the top the rows L_0 through L_{2e-1} in each direction.

S3. Connect the last processor in L_i to the first processor in $L_{[i+e-1]_{2e-1}}$.

For example, Fig. 1 shows a simple C-wrapped H-mesh of dimension 5.

C-wrapped H-meshes have several nice properties as reported in [19]. First, C-type wrapping results in a simple, transparent addressing scheme, where the center node is labeled as node 0, and the other nodes are labeled in sequence along the d_0 direction. (An example of the addressing for an H_5 is shown in Fig. 1.) Second, C-type wrapping results in a **homogeneous** network. Every node may view the mesh as a set of concentric hexagons (where each hexagon has one more node on each edge than the one immediately inside of it) with itself as the center node. Consequently, all nodes are topologically equivalent. Third, the diameter of an H_e is $e-1$. Consequently, any routing/broadcast packet traverses at most $e-2$ intermediate nodes before reaching its destination node. Fourth, simple and efficient routing and broadcast algorithms can be devised, as discussed in [19], [20].

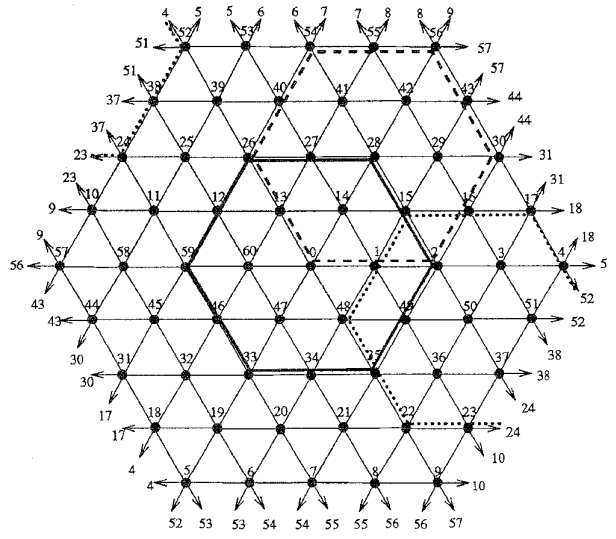


Fig. 1. A C-wrapped hexagonal mesh of dimension 5, H_5. Also shown is how the buddy sets of node 0 (the set of nodes within solid lines), node 28 (the set of nodes within dashed lines), and node 50 (the set of nodes within dotted lines) overlap with one another.

In what follows, we summarize the important features of the routing algorithm [19], the broadcast algorithm [20], and the virtual cut-through switching scheme [22], all of which support LS-related communication activities. These will be exploited in our analysis of the proposed LS mechanism.

Features of Routing Algorithm: The routing algorithm derived from the simple addressing scheme determines all shortest paths between any two nodes. In particular, it determines the number of hops, k_0, k_1, and k_2, from the source node to the destination node along the d_0, d_1, and d_2 directions, respectively.[3] Note that all shortest paths are completely specified by k_i, $i = 0, 1, 2$, and the number of shortest paths with k_i hops in direction d_i is $(|k_0| + |k_1| + |k_2|)!/(|k_0|!\ |k_1|!\ |k_2|!)$. All shortest paths between any two nodes are assumed to be equally used with probability $(|k_0|!\ |k_1|!\ |k_2|!)/(|k_0| + |k_1| + |k_2|)!$. Another notable feature is that at each node on a shortest path there are at most two different neighbors of the node to which the shortest path runs, i.e., at most two of k_is are nonzero [19]. That is, all shortest routes between a pair of nodes are formed by links along at most two different directions *only*.

Features of Broadcast Algorithm: An example of simple broadcasting in an H_4 is shown in Fig. 2, where, without loss of generality, the broadcasting node is placed at the center (node 0). For notational convenience, the $6h$ nodes which are h hops away from node 0 are said to be on the hth *ring* centered at node 0, $1 \le h \le e - 1$. The algorithm propagates a packet, ring by ring, toward the periphery of the mesh. The broadcasting node generates and transmits six copies of each broadcast message, one along each direction, d_i, $0 \le i \le 5$. Upon receiving the broadcast message, if the node is a corner node[4] relative to the broadcasting node (node 0) —the node lies along the direction, d_i $(0 \le i \le 5)$ with respect to node 0—then it transmits the broadcast packet along the direction 60 degree clockwise to the direction from which the packet arrived, in addition to propagating the packet to the next node along the direction of packet receipt. Otherwise, it just forwards the packet to the next node along the same direction in which it is received.

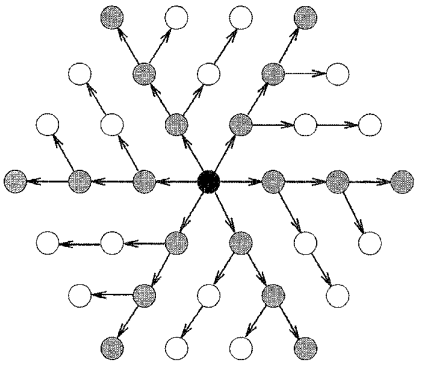


Fig. 2. Simple broadcast for a hexagonal mesh of dimension 4, H_4. Corner nodes are shaded. Links between nodes are not drawn for clarity.

Virtual Cut-Through: One of the switching schemes that support message routing and broadcasting in HARTS is commonly referred to as *virtual cut-through switching*. For this type of switching, packets arrived at an intermediate node are forwarded to the next node in the route without getting buffered if a circuit to the next node can be established. Specifically, the routing controller of HARTS contains six pairs of receivers and transmitters that are connected to a single *time-sliced*[5] bus. This bus is interfaced to the buffer management unit (BMU) in the node to store packets that cannot cut through the node, or are generated by or destined for the node. The operations for handling packets work as follows: when a packet is received, the receiver first recognizes the packet type. If the packet is of regular type, the receiver examines the routing tags in the packet header to check whether or not the packet has reached its destination. If not, it checks the directions in which a packet can be forwarded and tries to reserve a transmitter in one of these directions. If the reservation succeeds, the transmitter accepts the packet and forwards it to the next node (i.e., a cut-through occurs). If the reservation attempts do not succeed, the receiver requests BMU to store the packet for later transmission. If the received packet is of broadcast type, the receiver attempts to reserve the transmitter in the same direction, and drops a copy of the packet in the BMU simultaneously. Also, if the receiving node is a corner node, it will attempt to send a copy of the packet in the direction 60 degree clockwise to the direction from which the packet was received. If any of the reservation attempts does not succeed, the copy in the BMU will be transmitted later.

2.2 Load Sharing Mechanism

We use the LS approach in [1], [2] as the distributed sched-

3. Negative values mean the moves are in opposite directions.
4. Note that there are six corner nodes on each ring.

5. Each receiver is guaranteed to have access to the bus so that it may place on the bus the data it receives.

uling mechanism in HARTS. Salient features of this LS approach are that

1) It exploits the topological properties of the underlying interconnection network to evenly distribute overflow tasks over the entire system by using the concept of overlapping *buddy sets* and *preferred lists* in both location and information policies.
2) It reduces LS-related traffic while keeping the state information of other nodes as up–to–date as possible by using *region-change broadcasts* for the information policy.

We summarize in this subsection the transfer, information, and location policies used by the proposed LS approach, and discuss in the next subsection how to adapt this approach to HARTS. Specifically, we discuss how to construct the preferred list and buddy set in a C–wrapped H-mesh.

The transfer policy used works as follows: upon arrival of a task at node i, the node checks whether or not it can complete the task in time. For example, if the cumulative execution time (CET) contributed by those tasks queued on a node is the measure of workload, the node checks whether CET is greater than the task laxity or not. It is straightforward to extend this policy to cases with different measures of workload, such as the commonly–used measure, queue length (QL). If node i is capable of completing the task in time, the task will be accepted and queued for execution. Otherwise, the task will be transferred (in the form of routing packets) to a receiver node chosen by the location policy.

The following strategies are used in the proposed location and information policies.

Preferred Lists and Buddy Sets: To reduce the possibility of more than one node simultaneously transferring their overflow tasks to the same destination node and thus overloading the node, we order all other nodes according to the distance from node i into the *preferred list* of node i. If there are multiple nodes of the same distance, they are ordered based on their location. Each preferred list should be arranged so that every node in the system is selected as the kth preferred node of one and only one other node, for $k = 1, \ldots, 3e(e-1)$. One method of systematically constructing and preferred lists for hypercubes has been addressed in [1]. Once each node's preferred list is constructed, the node's *buddy set* can be formed with any required number of nodes counting from the top of its preferred list. A node with an overflow task can then select the *first* node found in its buddy set that is capable of completing the task in time, and transfers its overflow task to that node. (If all nodes in a buddy set are incapable of completing an overflow task at some time instant, the overflow task is declared failed and thus discarded.)

It is important to note that although the buddy set and preferred list of each node are generated *statically*, the node's actual preference in transferring an overflow task may change dynamically with the loading status of the nodes in its preferred list. That is, if a node's most preferred node gets overloaded, this fact will be known to the node via a state-change broadcast and its second preferred node will become the most preferred. (It will be changed to the second most preferred whenever the original most preferred node becomes underloaded, which will be again communicated via a state-change broadcast.)

By using the buddy sets constructed above, one can reduce the overhead of transferring tasks (as well as for collecting state information discussed below), because each node i is restricted to communicate with, maintain state information of, and transfer tasks to, only a set of N_B nodes in its proximity (e.g., those nodes that are within h hops for some h). Moreover, these buddy sets overlap with one another so that an overflow task may be transferred from an overloaded node to some other node that is included a different buddy set. For example, Fig. 1 shows how buddy sets overlap in HARTS. That is, those tasks arrived at a congested region—in which most nodes cannot complete all of their own tasks in time—can be shared by the entire system, rather than overloading the nodes in the region.

Region-Change Broadcasts: Each node exchanges state information via region-change broadcasts. Specifically, K_T state regions defined by $(K_T - 1)$ thresholds, TH_1, TH_2, ..., TH_{KT-1}, are used to characterize the workload of each node. Each node broadcasts a message, informing all the other nodes in its buddy set of its state-region change whenever its state crosses TH_i for some $i \in \{1, \ldots, K_T - 1\}$. The state information kept at each node is thus up–to–date as long as the broadcast delay is not significant. In contrast to other information policies such as bidding and state probing, this method avoids the need of collecting state information at the time of making LS decisions.

2.3 Construction of Preferred Lists and Buddy Sets in HARTS

We discuss how to construct preferred lists in a C-wrapped H-mesh to minimize the probability of more than one node simultaneously transferring their overflow tasks to the same node.

Due to the homogeneity of a C-wrapped H-mesh, any node can consider itself as the center (node 0) of the mesh. Without loss of generality, we can henceforth concentrate on constructing the preferred list of node 0. Each node on the "ring" of h hops away from node 0 (or simply the hth ring of node 0) can be reached from node 0 by a sequence of directions,

$$\underbrace{d_i\, d_i \ldots d_i}_{j \text{ items}} \underbrace{d_{[i+1]_6}\, d_{[i+1]_6} \ldots d_{[i+1]_6}}_{(h-j) \text{ items}} \triangleq d_i^j\, d_{[i+1]_6}^{(h-j)},$$

for some i and j, where we have used the shortest–route feature of routing algorithm in Section 2.1. Also note that all permutations of $d_i^j\, d_{[i+1]_6}^{(h-j)}$ lead us to the same node, i.e., all sequences of directions which are composed of j d_is and $(h-j)$ $d_{[j+1]_6}$s lead us to the same node. Consequently, the address of each node can be uniquely determined by the sequence of directions as follows.

LEMMA 1. *The node reachable from node i with any permutation of the sequence of directions,* $a_1\, a_2 \ldots a_k$, $a_j \in \{d_0, d_1, \ldots, d_5\}$, $\forall j \in \{1, \ldots, k\}$, *has address*

$$[i + \ell_0 + \ell_1 (3e^2 - 6e + 3) + \ell_2 (3e^2 - 6e + 2) + \ell_3 (3e^2 - 3e) + \ell_4 (3e^2 - 1) + \ell_5 3e^2]_{3e^2-3e+1}$$

where ℓ_j $(0 \le j \le 5)$ *is the number of times* d_j *appears in the sequence* $a_1\, a_2 \ldots a_k$, *and e is the dimension of the H-mesh.*

The proof of this lemma follows directly from the recursive use of Definition 1. For example, the node reachable from node 0 with the sequence, $d_0\, d_1^2$, has address

$$[0 + 1 + 2(3 \cdot 5^2 - 6 \cdot 5 + 3)]_{3\cdot 5^2 - 3\cdot 5 + 1} = [97]_{61} = 36$$

in an H_5 (Fig. 1).

Now, let the $6h$ nodes in the hth ring be ordered as

$$\begin{array}{l} d_0^h,\ \overline{d}_0^h,\ d_0^{(h-1)}d_1,\ \overline{d}_0^{(h-1)}\overline{d}_1,\ d_0^{(h-2)}d_1^2,\ \overline{d}_0^{(h-2)}\overline{d}_1^2,\ \ldots,\ d_0 d_1^{(h-1)},\ \overline{d}_0\overline{d}_1^{(h-1)}, \\ d_1^h,\ \overline{d}_1^h,\ d_1^{(h-1)}d_2,\ \overline{d}_1^{(h-1)}\overline{d}_2,\ d_1^{(h-2)}d_2^2,\ \overline{d}_1^{(h-2)}\overline{d}_2^2,\ \ldots,\ d_1 d_2^{(h-1)},\ \overline{d}_1\overline{d}_2^{(h-1)}, \\ d_2^h,\ \overline{d}_2^h,\ d_2^{(h-1)}d_3,\ \overline{d}_2^{(h-1)}\overline{d}_3,\ d_2^{(h-2)}d_3^2,\ \overline{d}_2^{(h-2)}\overline{d}_3^2,\ \ldots,\ d_2 d_3^{(h-1)},\ \overline{d}_2\overline{d}_3^{(h-1)}, \end{array} \tag{2.1}$$

where $\overline{d}_i \overset{\Delta}{=} d_{[i+3]_6}$. Node 0's preference in transferring an overflow task is then determined ring by ring, beginning with the innermost ring and terminating at the outermost ring. (Nodes within each ring are ordered as above.) Specifically, the kth preferred node of node 0 can be determined as follows:

1) find h such that $\sum_{j=1}^{h-1} 6j \le k < \sum_{j=1}^{h} 6j$, i.e., h determines which ring the k-th preferred node lies on;
2) set $\ell = k - \sum_{j=1}^{h-1} 6j$ specifies the position of the kth preferred node within the hth ring.

The address of the kth preferred node can then be determined by (2.1) and Lemma 1.

In what follows, we show that the preferred lists constructed above satisfy the requirements stated in Section 2.2.

LEMMA 2. *Each node in an H-mesh will be selected as the* kth *preferred node by one and only one other node,* $1 \le k \le 3e(e-1)$.

PROOF. An H-mesh forms a homogeneous processing surface where a sequence of directions, $d_i^j\, d_{[i+1]_6}^{(h-j)}$, leading to a given destination uniquely determines the corresponding source node. Thus, the lemma follows from the way preferred lists are constructed (i.e., k uniquely determines h and ℓ which in turn uniquely determine $d_i^j\, d_{[i+1]_6}^{(h-j)}$). □

LEMMA 3. *If node* i *is the* kth *preferred node of node* j, *then node* j *is the* $(k+1)$th $((k-1)$th$)$ *preferred node of node* i *if* k *is odd (even).*

PROOF. Suppose one follows the sequence, $d_\ell^m\, d_{[\ell+1]_6}^{(h-m)}$ to reach node j from node i, for some h, m, and ℓ. (Note that h, m, and ℓ are uniquely determined by k.) Then one can reach node i from node j by following the sequence, $\overline{d}_\ell^m\, \overline{d}_{[\ell+1]_6}^{(h-m)}$, where $\overline{d}_\ell^m\, \overline{d}_{[\ell+1]_6}^{(h-m)}$ is the direction right after (before) $d_\ell^m\, d_{[\ell+1]_6}^{(h-m)}$ within the hth ring in (2.1) if k is odd (even). □

This semi-symmetry property[6] implies that the task flow in one direction be approximately counter-balanced by that in the opposite direction. Now, the buddy set of a node can be formed with the first N_B nodes from the top of its preferred list. For ease of analysis, we assume that N_B is chosen such that the buddy set itself is also an H-mesh of dimension $m < e$, and $N_B = 3m(m-1) + 1$.

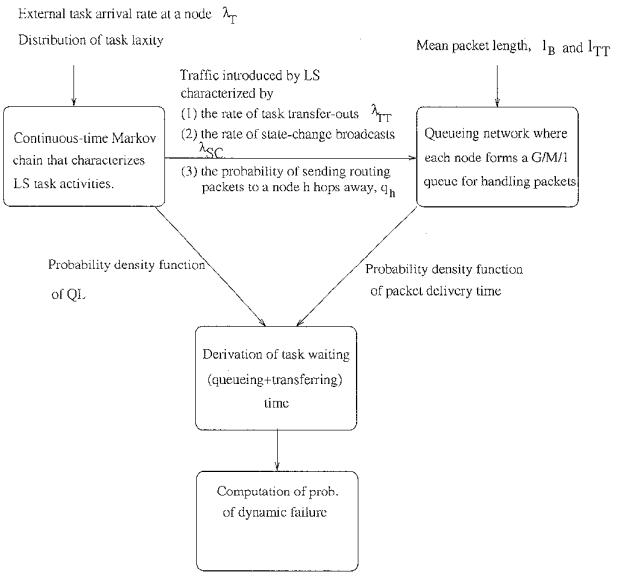


Fig. 3. Analysis methodology used for evaluating the integrated LS performance.

3 Performance Analysis

Our analysis method and the notations used are outlined in Fig. 3 and Table 1. We first construct an embedded continuous–time Markov chain to characterize task arrival/completion/transfer activities under the proposed LS mechanism in HARTS. Two sets of parameters are derived from the constructed Markov model:

1) the probability density function of QL, $\{p_N(n), n \ge 0\}$, and
2) the rate of transferring tasks, λ_{TT}, the rate of state-change broadcasts, λ_{SC}, and the probability of transferring an overflow task to a node h hops away, q_h.

The latter set of parameters is fed into the queuing network that models the handling of both task–transfer and broadcasting packets at each node in HARTS as a $G/M/1$ queue. The probability density function of packet delivery time, $f_{D_i}(t)$, can then be derived from the queuing network model and is used, along with $\{p_N(n), n \ge 0\}$, to derive the probability density function of task waiting time, $f_{W_k}(t)$. Finally, the probability of dynamic failure, P_{dyn}, can be computed from $f_{W_k}(t)$.

3.1 LS Analytic Model

As discussed in Section 2.1, every node in HARTS may view the C-wrapped mesh as a set of concentric hexagons with itself as the center node. Hence, the nodes in HARTS are topologically homogeneous and are identical in processing capability and speed. Besides, all nodes are assumed to have the same arrival rate of "external"[7] tasks. Consequently, the task arrival/transfer activities experienced by each node are stochastically identical over a long term. Un-

6. Unlike hypercubes, the symmetry property—if node i is the kth preferred node of node j, then node j is the kth preferred node of node i—cannot be achieved by any ordering of nodes due to the fact that the number of nodes in an H-mesh is odd.

7. By "external," we mean all arrived tasks minus transfer-in tasks at a node.

TABLE 1
NOTATIONS USED AND THEIR ASSOCIATED MEANINGS

Symbol	Meaning
λ_T	external task arrival rate at a node.
$1/\mu_T$	mean task execution time.
$\hat{p}_\ell$, $0 \le \ell \le L_{max}$	the probability that a task is associated with a laxity ℓ. L_{max} is the maximum task laxity in the task system.
$\{p_N(n), n \ge 0\}$	the probability density function of the queue length of a node.
$\alpha(n)$	the rate of transferring tasks out of a node given that the node's state is n. This parameter characterizes the transfer policy used.
$\beta(n)$	the rate of transferring tasks into a node given that the node's state is n. This parameter corresponds to the location policy used.
$\lambda(n)$	the composite (both external and transferred) task arrival rate at a node with $QL = n$.
$\gamma_j = P(N \ge j)$	the probability that a node's $QL \ge j$.
λ_{TT}	the rate of transferring tasks out of a node.
λ_{SC}	the rate of state–region–change broadcasts.
q_h	the probability of sending a task to a node h hops away.
R_i	the i-th state region.
T_{ii}	the mean recurrent time of R_i; the expected time until the first transition into R_i given that a node's state starts in R_i.
T_i	the average time the continuous-time Markov chain, $\mathcal{M}$, constructed in Section 3.1 spends in R_i.
π_k	the stationary probability that the underlying discrete time Markov chain for $\mathcal{M}$ stays in state k.
p_f	the probability that a packet arriving (and receiving services if necessary) at a node will be forwarded to one of its neighboring nodes.
Λ	the throughput rate of a node.
p_c	the probability of a packet cutting through an intermediate node.
$f_{D_i}(t)$	the probability density function of the time needed for a packet to travel i hops.
$\overline{\ell_B}$	the mean length of broadcast packets.
$\overline{\ell_{TT}}$	the mean length of task transfer packets.
$\overline{\ell}$	the mean packet length.
λ_B	the rate of generating broadcast packets at a node.
$\lambda_{A,B}$	the rate of terminal broadcast packets arrived at a node.
$\lambda_{T,B}$	the rate of transit broadcast packets arrived at a node.
λ_{TT}	the rate of generating task-transfer packets at a node.
$\lambda_{A,TT}$	the rate of terminal task-transfer packets arrived at a node.
$\lambda_{T,TT}$	the rate of transit task-transfer packets arrived at a node.
$f_{W_k}(t)$	the probability density function of task waiting time.
ρ	the traffic intensity of the queueing network of interest.

der this assumption, we can employ the general methodology—which was verified (via simulations) to be valid for homogeneous systems in [8], and was also used in [1], [16]—of first modeling the state evolution of a single node in isolation and then combining node-level models into a system-level model.

Specifically, we first model the state evolution of a node by a continuous-time Markov chain that serves as the underlying model. The parameters of this model are then derived to characterize task arrival/transfer/completion activities at the system level under the proposed LS strategy. Finally, a two-step iterative approach is taken to obtain a numerical solution to the Markov chain.

To facilitate the analysis, we make the following assumptions:

A1. External task arrivals at a node are Poisson with rate λ_T. Task execution times are exponentially distributed with mean $\frac{1}{\mu_T}$.

A2. Tasks are independent of one another, and are queued/executed on a node on a first–come–first–served (FCFS) basis.

A3. Each task is associated with a laxity ℓ (in units of mean task execution time) with probability $\hat{p}_\ell$, $0 \le \ell \le L_{max}$, where L_{max} is the maximum task laxity in the entire task system.

A4. The composite (both external and transferred) task arrivals can be approximated as Poisson. Also implied in this assumption is that task arrival/departure activities on a node are approximately independent of those on others over a long term.

A1–**A2** are consistent with those assumptions commonly used in the open literature [6], [8], [14]. The reason for **A2** is two-fold: first, FCFS is more analytically tractable than other local scheduling policies (e.g., minimum–laxity–first–served policy); second, as shown in [23] the choice of a local scheduling policy has only a minimal effect on the *qualitative* assessment of LS performance. That is, the LS performance lies more on the transfer, location and information policies. **A4** serves only as an *approximation*, because

1) the probability of sending a task to (or receiving a task from) a node depends on the state of both nodes, making the splitting process non-Poisson;
2) as will be derived later, task-transfer times in HARTS (as well as in many other distributed systems) are not exponentially distributed, making the arrival of transfer-in tasks non-Poisson.

However, **A4** is known to become realistic as the system size grows, and has been used implicitly in [1], [14]. We verified (via simulations) in [23] that this approximation is valid for reasonably large (≥ 12 nodes) homogeneous systems.

Underlying Model: For analysis simplicity, the state, N, of a node is defined as its queue length (QL), and each node is modeled as an $M/M/1$ queue. (The underlying model with the cumulative execution time (CET) as the state can be modeled as an $M^{[k]}/G/1$ queue with bulk arrivals at each node and was studied in [24].) The composite (both external and transferred) task arrival rate at a node with QL = n, denoted as $\lambda(n)$, depends on the node's QL, and the location and transfer policies used. The key issue in linking node-level models to a system-level model is to properly specify $\lambda(n)$, $n \geq 0$, so as to describe task arrivals and transfers in the system level. Once $\lambda(n)$, $n \geq 0$, is specified, the QL density function of a node, $\{p_N(n), n \geq 0\}$, can be obtained by solving

$$p_N(n) = p_N(0) \cdot \frac{\prod_{k=0}^{n-1} \lambda(k)}{\mu_T^n}, \quad n \geq 0, \tag{3.1}$$

and

$$\sum_{n=0}^{\infty} p_N(n) = 1. \tag{3.2}$$

By the definition of $\alpha(n)$, $\beta(n)$, and $\lambda(n)$ in Table 1, we have,

$$\lambda(n) = \lambda_T - \alpha(n) + \beta(n). \tag{3.3}$$

By appropriately specifying $\alpha(n)$ and $\beta(n)$ to describe both the transfer and location policies, one can use a Markov chain model to describe the operations of the proposed LS approach. Since a task is transferred to another node if it cannot be completed in time locally (i.e., QL > the laxity of the task), $\alpha(n)$ can be expressed as:

$$\alpha(n) = \lambda_T \cdot \sum_{j=0}^{n-1} \hat{p}_j, \tag{3.4}$$

i.e., all tasks arrived with laxities less than n will be transferred out of a node with QL = n.

On the other hand, $\beta(n)$ can be expressed as:

$$\beta(n) = \sum_{j=n}^{L_{max}} \lambda_T \hat{p}_j \cdot \gamma_{j+1}\left(1 + \gamma_{j+1} + \ldots + \gamma_{j+1}^{N_B - 1}\right)$$

$$= \sum_{j=n}^{L_{max}} \lambda_T \hat{p}_j \cdot \gamma_{j+1} \cdot \frac{1 - \gamma_{j+1}^{N_B}}{1 - \gamma_{j+1}}. \tag{3.5}$$

Note that

1) a node with QL = n can accept any task with laxity greater than n, hence the summation from n to L_{max}, and
2) the term $\lambda_T \hat{p}_j \cdot \gamma_{j+1}^k$ $(1 \leq k \leq N_B - 1)$ is "contributed" by the node whose first $(k-1)$ preferred nodes cannot complete an additional task with laxity j, and the term $\lambda_T \hat{p}_j \cdot \gamma_{j+1}^{N_B}$ accounts for the situation when all nodes in a buddy set cannot complete a task with laxity j in time.

Task Flow Conservation: Although $\alpha(n)$ and $\beta(n)$ are derived by considering task activities on a *single* node only, the law of flow conservation leads to the following relation between $\alpha(n)$ and $\beta(n)$:

THEOREM 1.

$$\sum_{k=0}^{\infty} \alpha(k) \cdot p_N(k) = \sum_{k=0}^{\infty} \beta(k) \cdot p_N(k). \tag{3.6}$$

PROOF. The theorem follows directly from the law of flow conservation, i.e., $\sum_{k=0}^{\infty} \lambda(k) \cdot p_N(k) = \lambda_T$, and (3.3). □

The correctness of (3.4) and (3.5) can now be verified by the following lemma.

LEMMA 4. *If no tasks are rejected, then* (3.6) *holds for the proposed LS approach.*

PROOF. With a little algebraic manipulation, we have

$$\sum_{k=0}^{\infty} \beta(k) \cdot p_N(k) = \sum_{k=0}^{\infty} \{\sum_{j=k}^{L_{max}} \lambda_T \hat{p}_j \cdot \gamma_{j+1} \frac{1 - \gamma_{j+1}^{N_B}}{1 - \gamma_{j+1}}\} \cdot p_N(k)$$

$$= \sum_{j=0}^{L_{max}} \lambda_T \hat{p}_j \cdot \gamma_{j+1} \frac{1 - \gamma_{j+1}^{N_B}}{1 - \gamma_{j+1}} \cdot \sum_{k=0}^{j} p_N(k)$$

$$= \sum_{j=0}^{L_{max}} \lambda_T \hat{p}_j \cdot \gamma_{j+1}\left(1 - \gamma_{j+1}^{N_B}\right),$$

where the second equality follows from interchanging the summation indices while preserving the range of summation. On the other hand,

$$\sum_{k=0}^{\infty} \alpha(k) \cdot p_N(k) = \sum_{k=0}^{\infty} \{\lambda_T \sum_{j=0}^{k-1} \hat{p}_j\} \cdot p_N(k) = \sum_{j=0}^{\infty} \lambda_T \hat{p}_j \sum_{k=j+1}^{\infty} p_N(k)$$

$$= \sum_{j=0}^{L_{max}} \lambda_T \hat{p}_j \cdot \gamma_{j+1}.$$

Inconsistency results from the nonzero probability of dynamic failure, since the term, $\gamma_{j+1}^{N_B}$, is the probability that in a buddy set a task with laxity j misses its deadline. If these "locally unsuccessful" tasks are not rejected or continuously transferred from node to node, $(1 + \gamma_{j+1} + \ldots + \gamma_{j+1}^{N_B - 1})$ in (3.5) is replaced by

$$\sum_{k=0}^{\infty} \gamma_{j+1}^k = 1/\left(1 - \gamma_{j+1}\right),$$

and

$$\beta(n) = \sum_{j=n}^{L_{max}} \lambda_T \hat{p}_j \cdot \gamma_{j+1} / \left(1 - \gamma_{j+1}\right),$$

from which (3.6) follows. □

Two-Step Iterative Approach: $\lambda(n)$ and $\beta(n)$ must be known before solving the Markov chain model for $\{p_N(n), n \geq 0\}$ (see (3.1)). However, $\lambda(n)$ depends on γ_j which in turn depends on $p_N(n)$. An iterative approach (first proposed in [24]) is taken to handle the difficulty associated with this recursiveness. In the first step, $p_N(n)$ is obtained by solving (3.1) and (3.2) with both $\alpha(n)$ and $\beta(n)$ set to 0, or, equivalently, $\lambda(n) = \lambda_T$, $\forall n$. The resulting $p_N(n)$s are used to compute and $\beta(n)$ and $\lambda(n)$ in the second step. Then $p_N(n)$s are recalculated with the new $\beta(n)$ and $\lambda(n)$. This procedure repeats until both $p_N(n)$ and $\lambda(n)$ converge to some fixed values.

3.2 LS-Related Traffic: Derivation of λ_{TT}, λ_{SC}, and q_h

In this subsection, we derive

1) the rate of transferring tasks out of a node, λ_{TT},
2) the rate of state-change broadcasts, λ_{SC}, and
3) the probability of sending a task (in the form of a task–transfer message) to a node h hops away, q_h, to characterize LS-related communication activities.

As indicated in Fig. 3, these parameters model the interaction between LS and the underlying communication subsystem.

Derivation of λ_{TT}: Since a task with laxity j arrived at a node has to be transferred if QL $\geq j+1$, the task transfer-out rate, λ_{TT}, of a node can be expressed as

$$\lambda_{TT} = \sum_{j=0}^{L_{max}} \lambda_T \hat{p}_j \gamma_{j+1},$$

or,

$$\lambda_{TT} = \sum_{j=0}^{L_{max}} \lambda_T \hat{p}_j \cdot \sum_{k=j+1}^{\infty} p_N(k) = \sum_{k=1}^{L_{max}+1} p_N(k) \cdot \lambda_T \sum_{j=0}^{k-1} \hat{p}_j$$
$$= \sum_{k=1}^{L_{max}+1} p_N(k) \cdot \alpha(k).$$

Derivation of λ_{SC}: Recall that there exist K_T state regions defined by $(K_T - 1)$ thresholds, $TH_1, TH_2, \ldots, TH_{(K_T-1)}$ in the state (QL) space, and a node broadcasts a message, informing the other nodes in its buddy set of its state-region change whenever its state crosses any of the broadcast thresholds. Thus, the rate of state-change broadcasts, λ_{SC}, is related to the mean recurrence time, T_{ii}, of the ith state region, R_i, $1 \leq i \leq K_T$. Specifically, let T_{ii} be the expected time until the first transition into the ith state region given that the node starts in the ith state region. Then λ_{SC} can be expressed as:

$$\lambda_{SC} = \sum_{i=1}^{K_T} \frac{1}{T_{ii}}.$$

T_{ii}, $1 \leq i \leq K_T$, can be derived from the continuous-time Markov chain, $\mathcal{M}$, constructed in Section 3.1. That is, for an irreducible, positive recurrent, and non-lattice continuous-time Markov chain (such as $\mathcal{M}$), we have

$$T_{ii} = \frac{T_i}{p_N(R_i)},$$

where T_i is the average time $\mathcal{M}$ spends in R_i, and can be expressed[8] as

$$T_1 = \frac{1}{\lambda(0)} \cdot \frac{\pi_0}{\sum_{k \in R_1} \pi_k} + \sum_{n \in R_1, n \geq 1} \frac{1}{\mu_T + \lambda(n)} \cdot \frac{\pi_n}{\sum_{k \in R_1} \pi_k},$$

and

$$T_i = \sum_{n \in R_i} \frac{1}{\mu_T + \lambda(n)} \cdot \frac{\pi_n}{\sum_{k \in R_i} \pi_k}, \quad i \geq 2,$$

where π_k is the stationary probability of the *underlying discrete–time* Markov chain for $\mathcal{M}$ stays in state k. $p_N(R_i)$ is the probability of $\mathcal{M}$ being in R_i, and can be expressed as

$$p_N(R_i) = \sum_{n \in R_i} p_N(n).$$

Derivation of q_h: Recall that node i transfers its overflow task to the first capable node found in its preferred list. By the way preferred lists are constructed, node i will transfer an overflow task to a node h hops away only if the first $3h(h-1)$ nodes (that are within $h-1$ hops) cannot complete the task in time and at least one of the $6h$ nodes on the hth ring can. By **A4** and Lemma 3, q_h can be expressed in terms of γ_j and $\hat{p}_j$ as:

$$q_h = \frac{1}{6h} \sum_{j=0}^{L_{max}} \gamma_{j+1}^{3h(h-1)} \left(1 - \gamma_{j+1}^{6h}\right) \cdot \hat{p}_j.$$

3.3 HARTS Queuing Network Model

Packet delivery in the H-mesh is modeled as a queuing network, where each of $3e(e-1)+1$ nodes forms a $G/M/1$ queue. Broadcast and task–transfer packets are generated at a node when its state region changes and when the node cannot complete a newly-arrived task in time, respectively. These occurrences do not follow Poisson, and hence a general distribution is needed to describe the packet arrival process on each node. Fortunately, characterizing packet arrival patterns *only* in terms of the rate of state-change broadcasts, λ_{SC}, and the rate of task transfers, λ_{TT}, while keeping the packet arrival process general suffices to derive the probability density function of packet delivery time. Both λ_{SC} and λ_{TT} depend on task arrival and departure activities in a node (and hence the LS mechanism used), and serve as the connection between the LS model and the HARTS queuing network.

A packet arriving at a node may go to one of its six immediate neighbors if it has not reached the destination node, or may otherwise exit from the system. The delay that a packet experiences at an intermediate node depends on whether or not it can cut through that node. If the packet establishes a circuit to a neighboring node, it will experience a negligible delay (so, we assume it to be 0). Otherwise, the packet requires "services," i.e., buffering and later transmission.

Thanks to the node homogeneity of HARTS, we can concentrate on evaluating the distribution of delivery time only for those packets generated at the center node, node 0. We will derive the following parameters in sequence. As will be clearer later, the derivation of one parameter depends on the other parameters derived before it.

1) Both transit and non-transit loads handled by node 0. By a "transit task-transfer packet," we mean a task-transfer packet traversing a node that is not destined for the node. By a "transit broadcast packet," we mean a broadcast packet that should be forwarded to

8. Since the distribution of the time until the next transition occurs given that $\mathcal{M}$ has just entered a state n is exponentially distributed with rate $\mu_T + \lambda(n)$.

the next neighbor node(s) (because it has not reached a periphery node relative to the broadcasting node). We use λ_B, $\lambda_{A,B}$, $\lambda_{T,B}$, λ_{TT}, $\lambda_{A,TT}$, and $\lambda_{T,TT}$ to describe different packet arrival rates. The definitions for these variables are listed in Table 1.

2) The probability, p_f, that a packet arriving (and receiving services if necessary) at a node will be forwarded to one of its neighboring nodes.
3) The throughput rate, Λ, at a node.
4) The probability, p_c, of a packet cutting through an intermediate node.
5) The probability density function, $f_{D_i}(t)$, of the time needed for a packet to travel i hops.

To facilitate the analysis, we make the following assumptions:

B1. The probability of a node sending a packet to a node h hops away is q_h which was derived in Section 3.2.

B2. All shortest paths between a pair of nodes are equally used for task–transfer packets.

B3. The lengths of broadcast packets and task–transfer packets are exponentially distributed with mean $\overline{\ell_B}$ and $\overline{\ell_{TT}}$, respectively. $\overline{\ell_B}$ is a constant in our algorithm.

B4. The length of a packet is regenerated at each intermediate node of its route independently of its length at other intermediate nodes.

B5. Only communication traffic required by load sharing is considered.

B2 can be justified as follows. The routing algorithm in [19] gives *all* shortest paths between a pair of nodes by specifying k_0, k_1, and k_2. Upon receipt of a task-transfer packet, the routing algorithm in [22] determines whether or not the packet has reached its destination node (i.e., $k_0 = k_1 = k_2 = 0$). If not, the packet is forwarded along one of the directions with nonzero k_i (and the corresponding k_i is updated) if cut-through can be established in that direction. Since the system is homogeneous, the probability of establishing a cut-through in any of the forwarding directions is assumed to be the same, meaning that all shortest paths are equally used for task-transfer packets. **B2** along with the topological properties of C-wrapped H-meshes will be used to determine the total number of shortest routes passing through node 0 for all pairs of communicating nodes. **B3**—**B4** coincide with Kermani and Kleinrock's assumptions in [21], and **B4** is commonly referred to as the independence assumption [21]. What is also implied in **B4** is the independence of the queue distributions in different nodes. Although **B4** is unrealistic in practice, several empirical studies in [21] have shown that the mean packet delay times computed under this assumption closely match the actual mean packet delivery times. **B5** is made to facilitate the analysis.

The method used to derive the various parameters is highlighted as follows. The rates λ_B, $\lambda_{A,B}$, and $\lambda_{T,B}$, contributed by broadcast packets (the rate λ_{TT}, $\lambda_{A,TT}$, and $\lambda_{T,TT}$, contributed by task-transfer packets) are first derived using the homogeneity property of a C–wrapped H-mesh and the feature of the broadcast algorithm (the routing algorithm). The probability, p_f, of forwarding a packet to neighboring nodes can then be computed as the ratio of the traffic bound for immediate neighbors (expressed in terms of λ_B, λ_{TT}, $\lambda_{T,B}$, and $\lambda_{T,TT}$) to all arrived traffic (expressed in terms of λ_B, λ_{TT}, $\lambda_{A,B}$, and $\lambda_{A,TT}$). The third quantity, the throughput, Λ_i, at node i, is derived, as a function of λ_B, λ_{TT}, and p_f, with use of the principle of *flow conservation*. The probability of virtual cut–through, p_c, is then derived, as a function of Λ_i and $\overline{\ell}$, with the use of the *utilization law*. Finally, by conditioning on whether or not a packet gets buffered at intermediate nodes, the distribution of the delivery time for a packet traveling i hops, D_i, can be derived using the feature of virtual cut through.

Derivation of Packet Arrival Rates:

LEMMA 5. *λ_B, $\lambda_{A,B}$, and $\lambda_{T,B}$ are given by*

$$\lambda_B = 6(m-1)\lambda_{SC},$$
$$\lambda_{A,B} = 3m(m-1)\lambda_{SC},$$
$$\lambda_{T,B} = 3(m-1)(m-2)\lambda_{SC},$$

where λ_{SC} is the rate of state–region changes at node 0, and $m < e$ is the dimension of a buddy set, H_m.

PROOF. As indicated in Fig. 2,

1) the broadcasting node generates six packets per broadcast,
2) every nonperiphery corner node, upon receiving (and storing) a broadcast packet, transmits an additional copy along the direction 60 degree clockwise to the direction of packet receipt.

By 1), the rate of generating broadcast packets for node 0 is $6\lambda_{SC}$. Also, by the homogeneity property of an H-mesh, node 0 acts as a nonperiphery corner node relative to $6(m-2)$ nodes in its buddy set (see Fig. 1), and is responsible for generating (and forwarding) a new packet upon arrival of a packet broadcast by those $6(m-2)$ nodes. Thus, by 2), node 0 generates broadcast packets for $6(m-2)$ other nodes in its buddy set at the total rate of $6(m-2)\lambda_{SC}$. The expression of λ_B thus follows from 1) and 2).

Node 0 receives broadcast packets from the other $3m(m-1)$ nodes in its buddy set, and since all nodes are homogeneous each with a state-change broadcast rate λ_{SC}, the expression of $\lambda_{A,B}$ follows. On the other hand, node 0 lies on the periphery of the buddy set of other $6(m-1)$ nodes whose broadcast packets will not be forwarded by node 0. That is, the non-transit load (destined for node 0) is $6(m-1)\lambda_{SC}$. $\lambda_{T,B}$ is thus $[3m(m-1) - 6(m-1)]\lambda_{SC} = 3(m-1)(m-2)\lambda_{SC}$. □

LEMMA 6. *$\lambda_{A,TT}$ and $\lambda_{T,TT}$ are given by*

$$\lambda_{A,TT} = \lambda_{TT}\sum_{k=1}^{m-1} 6k^2 \cdot q_k,$$
$$\lambda_{T,TT} = \lambda_{TT}\sum_{k=2}^{m-1} 6k(k-1) \cdot q_k,$$

where λ_{TT} is the rate of transferring tasks out of node 0, and q_k is the probability of a node transferring tasks to a node k hops away.

PROOF. To calculate $\lambda_{A,TT}$ at node 0, we need to determine

1) the load contributed to $\lambda_{A,TT}$ at node 0 by a shortest route that passes through node 0;
2) the total number of shortest routes passing through node 0 for all pairs of communicating nodes.

It was verified in [19] that all shortest routes between any pair of nodes are formed by links along at most two directions and thus can be represented by a sequence of directions, $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$, where $0 \le j \le 5$, ℓ is the length of the shortest route ($1 \le \ell \le m-1$), and i and $\ell - i$ ($1 \le i \le \ell$) are the number of hops from the source to the destination along d_j and $d_{[j+1]_6}$, respectively.

Now, consider a sequence of directions, $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$. The number of shortest routes associated with $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$ that pass through node 0 can be calculated as follows. There are $\binom{\ell}{i}$ permutations for d_js and $d_{[j+1]_6}$ s in the sequence $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$, each of which gives a possible shortest route. For each possible route, node 0 can be inserted in one of the ℓ positions (including the destination) to be an intermediate or destination node of the route. Thus, each sequence of the form $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$ represents $\binom{\ell}{i} \cdot \ell$ shortest routes that pass through node 0.

The load at node 0 contributed by a single shortest route (represented by $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$) that passes through node 0 can be expressed as $\lambda_{TT} \cdot q_\ell \Big/ \binom{\ell}{i}$, because a node sends packets to nodes ℓ hops away at a rate of $\lambda_{TT} \cdot q_\ell$ which is equally shared by all $\binom{\ell}{i}$ shortest routes (under **B2**).

The rate of task-transfer packets arriving at node 0 can now be expressed as

$$\lambda_{A,TT} = \sum_{j=0}^{5}\sum_{\ell=1}^{m-1}\sum_{i=1}^{\ell} \text{load contributed by all routes represented by } d_j^i\, d_{[j+1]_6}^{(\ell-i)}$$

$$= \sum_{j=0}^{5}\sum_{\ell=1}^{m-1}\sum_{i=1}^{\ell}\binom{\ell}{i} \cdot \ell \cdot \frac{\lambda_{TT} \cdot q_\ell}{\binom{\ell}{i}}$$

$$= \sum_{j=0}^{5}\sum_{\ell=1}^{m-1}\sum_{i=1}^{\ell} \lambda_{TT} \cdot q_\ell \cdot \ell = \lambda_{TT} \sum_{\ell=1}^{m-1} 6\ell^2 \cdot q_\ell.$$

The proof of the second part of the lemma resembles the first part except that for each possible route that can be represented by $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$, node 0 cannot be the destination node. Thus, there are ℓ–1 positions to insert node 0, and each sequence $d_j^i\, d_{[j+1]_6}^{(\ell-i)}$ gives $\binom{\ell}{i} \cdot (\ell-1)$ shortest routes that pass through node 0.

The rate of transit traffic, $\lambda_{T,TT}$, at node 0 can now be expressed as

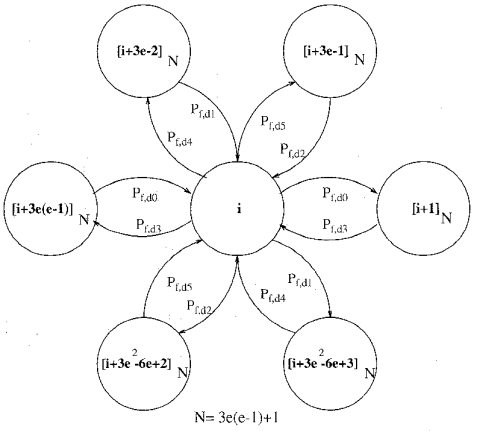


Fig. 4. Neighbors and packet flow around node i.

$$\lambda_{T,TT} = \sum_{j=0}^{5}\sum_{\ell=1}^{m-1}\sum_{i=1}^{\ell} \text{transit load contributed by all routes represented by } d_j^i\, d_{[j+1]_6}^{(\ell-i)}$$

$$= \sum_{j=0}^{5}\sum_{\ell=1}^{m-1}\sum_{i=1}^{\ell}\binom{\ell}{i} \cdot (\ell-1) \cdot \frac{\lambda_{TT} \cdot q_\ell}{\binom{\ell}{i}} = \lambda_{TT} \sum_{\ell=1}^{m-1} 6\ell(\ell-1) \cdot q_\ell. \qquad \square$$

Derivation of p_f: The probability, p_f, that a packet arriving at a node will be forwarded to one of its neighboring nodes is the ratio of the traffic bound for immediate neighbors to all traffic arriving at a node:

$$p_f = \frac{\lambda_B + \lambda_{T,B} + \lambda_{TT} + \lambda_{T,TT}}{\lambda_B + \lambda_{A,B} + \lambda_{TT} + \lambda_{A,TT}}.$$

Using the results of Lemmas 5 and 6 along with

$$\sum_{k=1}^{m-1} 6k \cdot q_k = 1,$$

we have

$$p_f = \frac{3m(m-1) \cdot \lambda_{SC} + \sum_{k=1}^{m-1} 6k^2 q_k \cdot \lambda_{TT}}{3(m+2)(m-1) \cdot \lambda_{SC} + \left[1 + \sum_{k=1}^{m-1} 6k^2 q_k\right] \cdot \lambda_{TT}}. \quad (3.7)$$

Derivation of Throughput of a Node: To derive the throughput, Λ_i, of a node i, we use the principle of *flow conservation*.[9] Specifically, let Λ_i and Λ_k, $0 \le k \le 5$, represent the throughput of node i and its neighboring nodes in directions d_0–d_5, respectively, and let p_{f,d_k} represent the probability that a packet completing its service will be forwarded to the neighboring node in direction d_k, then the flow conservation principle enforces (Fig. 4)

$$\Lambda_i = \left(\lambda_B + \lambda_{TT}\right) + \sum_{k=0}^{5} p_{f,\overline{d_k}} \cdot \Lambda_k.$$

Now, by the homogeneity of the C-wrapped H-mesh,

9. Which states that at any branching point in a queuing network there is no accumulation or loss of customers.

all Λ_i's are equal. Also, $\sum_{k=0}^{5} p_{f,\overline{d_k}} = p_f$. Thus, we have

$$\Lambda_i = \frac{\lambda_B + \lambda_{TT}}{1 - p_f} = \frac{3(m+2)(m-1)\cdot\lambda_{SC} + \left[1 + 6\sum_{k=1}^{m-1} k^2 q_k\right]\cdot\lambda_{TT}}{6(m-1)\lambda_{SC} + \lambda_{TT}} \cdot \left(6(m-1)\lambda_{SC} + \lambda_{TT}\right). \quad (3.8)$$

Derivation of p_c: As discussed in Section 2.1, a packet can cut through an intermediate node only if no packets are being transmitted or waiting at that node. Using the *Utilization Law* [25][10] the probability of having no packet at a node is $1 - \rho$, where ρ is the traffic intensity

$$\rho = \frac{\Lambda_i}{\mu_m} = \frac{\Lambda_i \cdot \overline{\ell}}{6}.$$

Here $\overline{\ell}$ is the mean packet length, and can be expressed as

$$\overline{\ell} = \frac{\lambda_B + \lambda_{A,B}}{\lambda_{TT} + \lambda_{A,TT} + \lambda_B + \lambda_{A,B}} \cdot \overline{\ell_B} + \frac{\lambda_{TT} + \lambda_{A,TT}}{\lambda_{TT} + \lambda_{A,TT} + \lambda_B + \lambda_{A,B}} \cdot \overline{\ell_{TT}}.$$

p_c can then be expressed as

$$p_c = 1 - \frac{\Lambda_i \cdot \overline{\ell}}{6},$$

where Λ_i is expressed in (3.8).

Derivation of Distribution of Packet Delivery Time: The delivery time for a packet traveling i hops, denoted by D_i, depends on whether or not the packet can cut through intermediate nodes. If the packet cuts through an intermediate node (the probability of which is p_c), its delay at the node is negligible and assumed to be 0. Otherwise, the packet experiences an exponential buffering delay, Y_k, with rate $\mu_m(1 - \rho)$. Note that the time experienced by a packet buffered at an intermediate node depends on the throughput of the node, and is accounted for by the factor $(1 - \rho)$. Moreover, the probability of a packet not cutting through j out of $i - 1$ intermediate nodes (when it travels i hops) is

$$\binom{i-1}{j}(1-p_c)^j p_c^{i-1-j}.$$

The distribution of D_i can then be expressed as

$$P(D_i \le t) = \sum_{j=0}^{i-1}\left(\begin{matrix} P(D_i \le t \mid \text{buffered at } j \text{ intermediate nodes}) \cdot \\ P(\text{buffered at } j \text{ intermdiate nodes}) \end{matrix}\right) = \sum_{j=0}^{i-1} P(Y_0 + \sum_{k=1}^{j} Y_k + Y_i \le t) \cdot \binom{i-1}{j}(1-p_c)^j p_c^{i-1-j},$$

where a packet always gets buffered at the source and destination nodes, and $Y_0 + \sum_{k=1}^{j} Y_k + Y_i$ can be shown to have an Erlang distribution with parameters $\mu_m(1 - \rho)$ and $j + 2$ under **B4**. That is, the probability density function of D_i can be expressed as

$$f_{D_i}(t) = \sum_{j=0}^{i-1}\left[\mu_m(1-\rho)\right]^{j+2} \cdot \frac{t^{j+1} e^{-\mu_m(1-\rho)t}}{(j+1)!} \cdot \binom{i-1}{j}(1-p_c)^j p_c^{i-1-j}.$$

10. Note that the Utilization Law is valid for a G/G/1 queue.

3.4 Derivation of Task Waiting Time and Probability of Dynamic Failure

Having derived the probability density function of QL, $\{p_N(n), n \ge 0\}$, and the probability density function of packet delivery time, $f_{D_i}(t)$, $t \ge 0$, $1 \le i \le m - 1$, we are now in a position to derive the distribution of task waiting time.

The probability density function of waiting time for a task queued on a node with state $N \le k$ is

$$f_{W|N\le k}(t \mid N \le k) = \sum_{n=0}^{k} f_{W_{(n)}}(t \mid N = n) \cdot \frac{p_N(n)}{1 - \gamma_{k+1}},$$

where $f_{W_{(n)}}(t \mid N = n)$ is the probability density function of waiting time given $N = n$, and can be shown under **A1** and **A2** in Section 3.1 to be n-stage Erlang. Thus,

$$f_{W|N\le k}(t \mid N \le k) = \delta(t) \cdot \frac{p_N(0)}{1 - \gamma_{k+1}} + \sum_{n=1}^{k} \frac{\mu_T(\mu_T t)^{n-1}}{(n-1)!} e^{-\mu_T t} \cdot \frac{p_N(n)}{1 - \gamma_{k+1}}, \quad (3.9)$$

where $\delta(t)$ is the impulse function such that $\delta(t) = 0$ for $t \ne 0$; $\delta(t) \ne 0$ for $t = 0$, and $\int_0^\infty \delta(t)\,dt = 1$.

The waiting time for a task with laxity k depends on whether or not the task arrives at a node with QL $\le k$. If the QL of the node at which the task arrives is $\le k$, then the task experiences the waiting time with density function $f_{W|N\le k}(t \mid N \le k)$. Otherwise, the task has to be transferred, possibly several times until it arrives at a capable node. That is, the task experiences the delivery time(s) and the waiting time on a capable node. The possibility of multiple transfers results from the fact that the state at the selected receiver node i hops away at the time *when the task is sent by the sender node* may be different from that *when the transferred task arrives at the receiver node*. The possibility of state inconsistency increases as the packet delivery time, D_i, increases.

Specifically, let $f_{W|N>k}(t)$ denote the density function of the waiting time experienced by a task with laxity k that arrived at an incapable node and is thus transferred out, then the density function of waiting time for tasks with laxity k can be expressed as:

$$f_{W_k}(t) = (1 - \gamma_{k+1}) \cdot f_{W|N\le k}(t \mid N \le k) + \gamma_{k+1} \cdot f_{W|N>k}(t), \quad (3.10)$$

where $f_{W|N>k}(t)$ is approximated as:

$$f_{W|N>k}(t) = \sum_{i=1}^{m-1}\left(\begin{matrix} \gamma_{k+1}^{3i(i-1)}\left(1 - \gamma_{k+1}^{6i}\right) \cdot f_{D_i}(t) * \\ \left[p_{sc,i}\, f_{W|N\le k}(t \mid N \le k) + (1 - p_{sc,i}) f_{W|N>k}(t)\right] \end{matrix}\right). \quad (3.11)$$

Here * denotes the convolution of two probability density functions, and $p_{sc,i}$ is the probability of *state consistency* within an i-hop delivery time, to be derived below. Note that $f_{W|N>k}(t)$ is expressed as a function of itself, thus a time–frequency

domain transformation (e.g., the Laplace transform) is necessary to obtain numerical solutions for $f_{W|N>k}(t)$, $t \geq 0$.

To derive $p_{sc,i}$, two scenarios in which state inconsistency may occur are considered. In the first case (Fig. 5a), during the transfer of an overflow task $\mathcal{T}$ from a sender node j to a receiver node i, a new task arrives at node i, making it become unable to complete the transferred task $\mathcal{T}$ upon its arrival. In the second case (Fig. 5b), node j receives an overflow task $\mathcal{T}$ and transfers it to an incapable node i before the broadcast packet (informing node j of node i's incapability) from node i reaches node j. Note that in both cases state inconsistency arises because a task arrives during the packet delivery time, D_i. We thus approximate[11] $p_{sc,i}$ as the probability that no tasks arrive during the packet delivery time, D_i, i.e.,

$$p_{sc,i} = P(\text{No task arrives within an } i \text{ - hop delivery time}) = \int_0^\infty \{ \sum_{n=0}^{L_{max}+1} e^{-\lambda(n)t} \cdot p_N(n) \} \cdot f_{D_i}(t)\, dt.$$

Finally, the probability of dynamic failure, $P_{dyn,k}$, experienced by a task with laxity k is

$$P_{dyn,k} = \int_k^\infty f_{W_k}(t)\, dt, \quad \text{and} \quad P_{dyn} = \sum_{k=0}^{L_{max}} \hat{p}_k \cdot P_{dyn,k}.$$

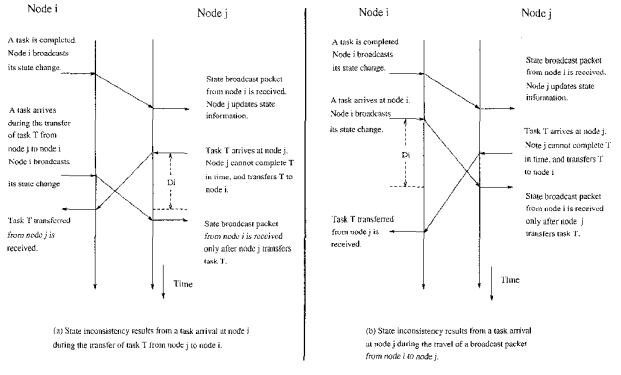


Fig. 5. Two situations state inconsistency may arise.

4 Numerical Examples

To evaluate the performance of LS while considering all LS-related communication activities, we used a discrete-event simulator which models the operations of the proposed LS mechanism in HARTS. The routing, broadcasting, and virtual cut-through schemes in [19], [20], [22] are used to facilitate LS-related communication activities. The goal of the simulation is two-fold:

1) examine the impact of approximations/assumptions made in the analysis, and
2) evaluate the integrated LS performance.

The simulator was originally developed by the authors of [22] which accurately models the delivery of each packet by emulating the routing hardware along the route of a packet at the microcode level. It also captures the internal bus access overheads experienced by packets as they pass through an intermediate node. For example, when a packet arrives at a node, the following sequence of events is initiated. First, the receiver for the link on which the packet arrived waits for the packet header to become available. It then examines the packet header to determine the packet type. Lastly, the receiver schedules events to signal the completion of the packet transmission at this node. This may involve unreserving a transmitter if the packet successfully cuts through and/or informing the module which handles buffered messages.

We modified the simulator to include modules that

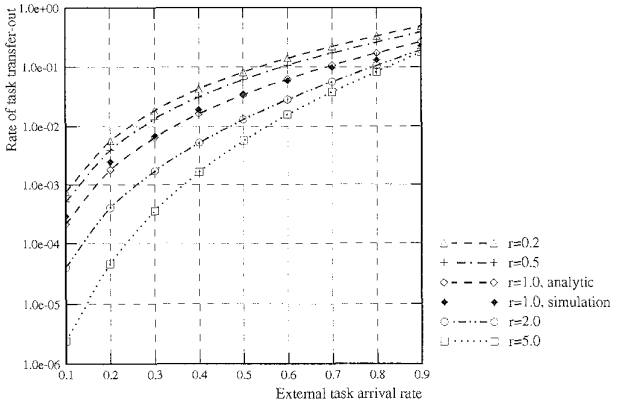
1) model task arrival, transfer, and completion activities under the proposed LS mechanism,
2) generate task-transfer and broadcast packets (along with their proper headers) at the time of task transfer and state-region change, and
3) update the preferred list of a node upon receipt of a broadcast packet, so that the main features of the proposed LS mechanism may be incorporated into the simulator.

The simulator differs from the analytical model in that: **(D1)** if a node considers none of the nodes in its preferred list is capable of completing its overflow task in time, the overflow task is declared failed and taken into account of the statistics for P_{dyn}; **(D2)** the length of a packet is determined at the time of its birth and remains unchanged while the packet traverses through the network. The latter is used to inspect the discrepancy between the packet delivery time analytically computed under **A4** in Section 3.3 and that observed in practice.

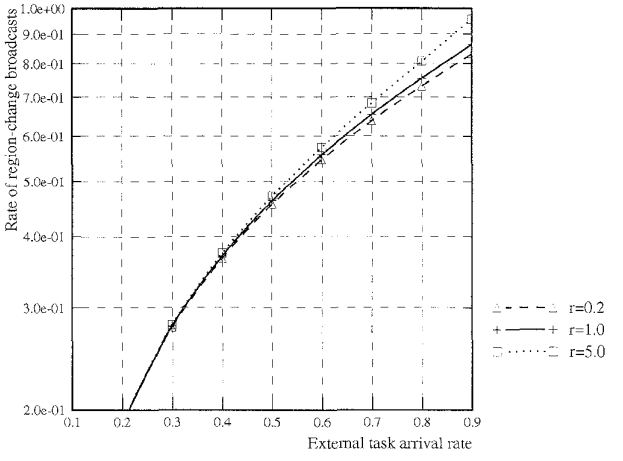
An H_5 is used as an example for all simulation runs. The buddy set is chosen to be an H-mesh of dimension 4 (hence each buddy set consists of 37 nodes). For convenience, μ_T is set to 1, and all time-related parameters are expressed in units of $1/\mu_T$. The threshold values are set to $TH_1 = 1$, $TH_2 = 3$, and $TH_3 = 5$ unless specified otherwise. Simulations are carried out for a task set with the external task arrival rate λ_T on each node varying from 0.1 to 0.9, the mean task-transfer packet length $\overline{\ell_R}$ varying from 0.1 to 5.0, and the mean broadcast packet length $\overline{\ell_B}$ varying from 0.01 to 0.15. The distribution of task laxity is assumed to be a geometric distribution with $\hat{p}_{\ell+1} = r \cdot \hat{p}_\ell$, where $1 \leq \ell \leq 5$, and r is chosen as 0.2, 0.5, 1.0, 2.0, and 5.0. Note that $r = 1.0$ gives a uniform distribution.

For each combination of parameters, the number of simulation runs needed is determined such that a 95% confidence level in the results for a maximum error of 5% of the specified probability can be achieved. We also compare the numerical results obtained with two other baselines whenever appropriate. The first baseline is an $M/M/1$ queue, representing the case of no load sharing, and the second baseline is an $M/M/37$ queue, representing the case of perfect load sharing where each node has perfect state information of other nodes in the buddy set and incurs no time overheads in task transfers and state-change broadcasts.

11. Note that $p_{sc,i}$ errs on the conservative side, because the selected receiver node (node i in Fig. 5) of a transferred task $\mathcal{T}$ may still complete $\mathcal{T}$ in time after receiving a new task as long as the QL of the receiver node after receiving the new task is still less than or equal to the laxity of $\mathcal{T}$.



(a) λ_{TT} vs. λ_T.

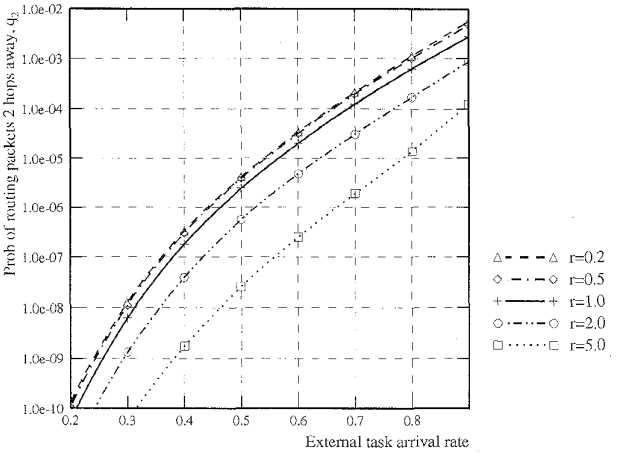


(b) λ_{SC} vs. λ_T.

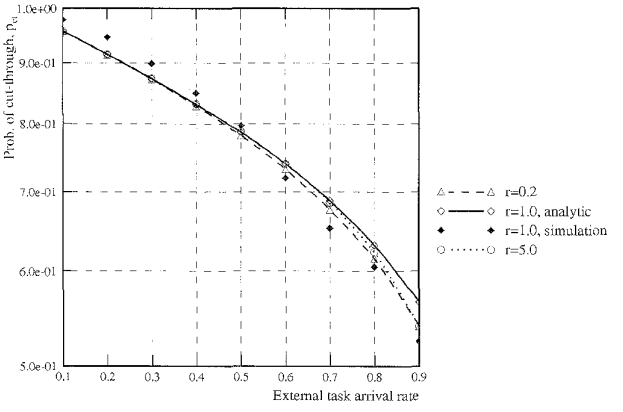
Fig. 6. Traffic generated by LS (measured in terms of λ_{TT}, λ_{SC}, q_2, and p_{ct}) for different external task arrive rate, λ_T. The distribution of task laxity is assumed to be uniformly distributed over [1, 5], $\overline{\ell_R} = 0.5$, $\overline{\ell_B} = 0.05$.

The traffic overhead introduced by LS and its impact on the possibility of cut-through are plotted in Fig. 6, where λ_{TT} vs. λ_T, λ_{SC} vs. λ_T, q_2 vs. λ_T, and p_{ct} vs. λ_T are plotted. As expected, λ_{TT}, λ_{SC}, and q_h all increase as λ_T increases. Consequently, cut-through is more unlikely to be established at intermediate nodes as λ_T increases (Fig. 6d). λ_{TT} and q_h also increase as the task laxity gets tighter, but λ_{SC} decreases as the laxity gets tighter, where the tightness of laxity is measured in terms of r as defined above. The latter phenomenon is perhaps due to the fact that an incapable node tends to locate *idle* nodes for its overflow tasks with tight laxities. Under such a scenario, both the sender node and the idle destination node need not broadcast a region-change message upon arrival of a tight-laxity task,[12] and hence λ_{SC} slightly decreases as task laxity gets tighter.

12. Note that $TH_1 = 1$, and thus when QL changes from 0 to 1, no message is broadcast.



(c) q_2 vs. λ_T.



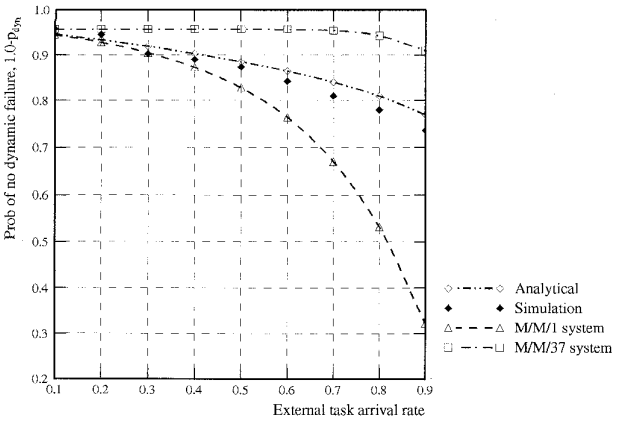
(d) p_{ct} vs. λ_T.

Fig. 6 (continued). Traffic generated by LS (measured in terms of λ_{TT}, λ_{SC}, q_h, and p_{ct}) for λ_T.

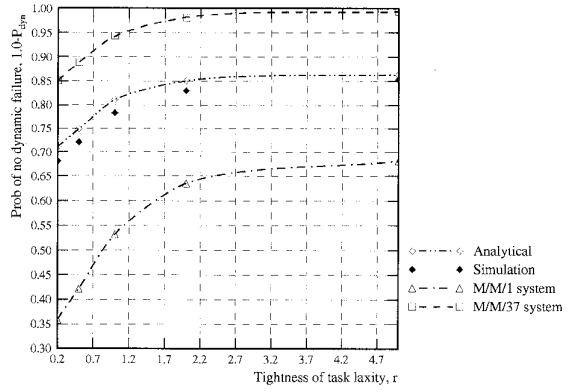
As shown in Fig. 6a and d, the analytic results predict, with a reasonable accuracy, the simulation results (usually within a 6% difference in all our simulations). The fact that the analytic model overestimates p_{ct} at higher loads is perhaps because the model does not consider the overhead of processing packet headers. This overhead becomes non-negligible when the number of packets traversing the network is high.

Fig. 7 shows the plots of $1 - P_{dyn}$ vs. λ_T and $1 - P_{dyn}$ vs. r. The proposed LS mechanism significantly outperforms the case of no LS (the $M/M/1$ system) especially at high system loads or tight task laxity, but is still inferior to perfect LS (the $M/M/37$ system). The latter also suggests that the time overheads in task transfers and state-change broadcasts cause deterioration in the LS performance and thus an efficient communication system that supports time-constrained communication is essential to real-time LS. The fact that the analytic model slightly overestimates $1 - P_{dyn}$ at higher loads is partly because of **D1** stated above.

The impact of communication delays on the performance of LS is studied by varying both $\overline{\ell_R}$ and $\overline{\ell_B}$ (and con-

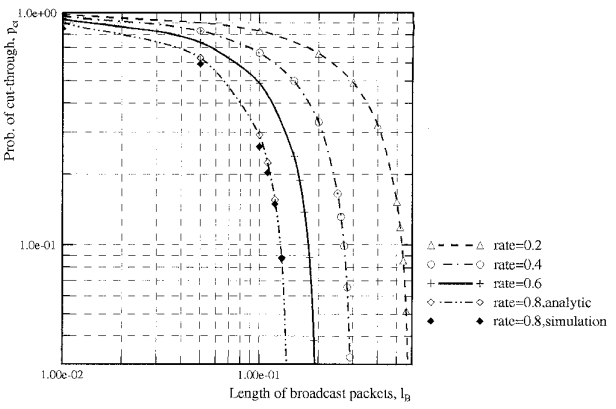


(a) $1 - P_{dyn}$ vs. λ_T

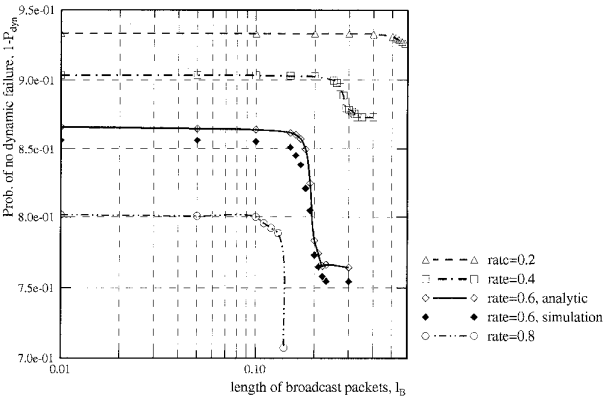


(b) $1 - P_{dyn}$ vs. r ($\lambda_T = 0.8$)

Fig. 7. $1 - P_{dyn}$ vs. λ_T and tightness of task laxity r. The distribution of task laxity is uniformly distributed over [1, 5] in (a) and is geometrically distributed with $\hat{p}_{\ell+1} = r \cdot \hat{p}_\ell$, for $1 \le \ell \le 5$. $\lambda_T = 0.8$, $\overline{\ell_R} = 0.5$, $\overline{\ell_B} = 0.05$.



(a) p_{ct} vs. $\overline{\ell_B}$



(b) $1 - P_{dyn}$ vs. $\overline{\ell_B}$

Fig. 8. The effect of $\overline{\ell_B}$ on p_{ct} and $1 - P_{dyn}$. The distribution of task laxity is uniformly distributed over [1, 5], and $\overline{\ell_R} = 0.5$.

sequently μ_m). Fig. 8a and b shows the plots of p_{ct} vs. $\overline{\ell_B}$ and $1 - P_{dyn}$ vs. $\overline{\ell_B}$, respectively. (The effect of varying $\overline{\ell_R}$ on LS is similar to, but less pronounced than,[13] that of $\overline{\ell_B}$, and thus omitted.) As shown in Fig. 8a, p_{ct} drops abruptly as $\overline{\ell_B}$ increases beyond a certain value. This indicates that when $\overline{\ell_B}$ becomes very large (or equivalently, the speed of the BMU is slow), the network becomes saturated and incapable of handling all incoming packets (introduced by LS). Consequently, packets will be queued at every intermediate node, thus delaying or even blocking the operation of task transfers and state-change broadcasts, and hence $1 - P_{dyn}$ decreases until it reaches approximately the value of P_{dyn} of an $M/M/1$ system (no LS).

13. This is because $\lambda_B = 6(n-1)\lambda_{SC}$ and $\lambda_{A,B} = 3n(n-1)\lambda_{SC}$ are usually much larger than λ_{TT} and $\lambda_{A,R}$, and thus dominate the determination of μ_m.

5 Conclusion

The analysis presented in this paper is essential to the design of any LS mechanism for real-time applications. First of all, the model gives a quantitative measure of traffic overheads (through λ_{TT}, λ_B, and $F_{D_i}(t)$) introduced by the LS mechanism. Second, one can determine many LS design parameters (e.g., the size of buddy sets, N_B, and the threshold values, TH_i, for state-change broadcasts) by determining the parameter values which minimize P_{dyn}. (This problem is currently under investigation.) Third, one can investigate whether or not the underlying routing and broadcasting schemes can support the LS mechanism in delivering packets and collecting & maintaining up–to–date state information.

The analysis methodology presented here (Fig. 3) is quite general in the sense that it can be extended to

1) other LS mechanisms such as LS with random selection, LS with state probing, and LS with perfect information, and
2) other homogeneous interconnection topologies such as hypercubes, rings, or square meshes, to name a few.

Extending this methodology to other LS mechanisms, one only needs to properly derive the parameters $\alpha(n)$ and $\beta(n)$ which characterize the transfer policy and the location policy, respectively. Once $\alpha(n)$ and $\beta(n)$ are determined, the derivation of the others (e.g., $p_N(n)$, λ_{TT}, λ_B, and q_h) follows the same approach. Similarly, extending the methodology to other homogeneous interconnection topologies, one only needs to specify the parameters p_f and p_c. The key to the specification of p_f (and consequently p_c) is the determination of transit loads at each node. That is, one must determine the fraction of shortest routes between all pairs of communicating nodes that pass through a given node as an intermediate node. Once p_f and p_c are determined, the derivation of packet delivery time distribution does not depend on the interconnection topology.

List of Symbols

P_{dyn}: the probability of a node failing to complete a task before its deadline.

H_e: A C–wrapped hexagonal mesh of dimension e.

M: the number of nodes in the distributed system, e.g., $M = 3e(e-1)+1$ in H_e.

m: the dimension of the buddy set which itself is a hexagonal mesh, i.e., the buddy set is a H_m.

N_B: the number of nodes in a buddy set, i.e., $N_B = 3m(m-1)+1$.

d_i, $0 \le i \le 5$: the $(60 \cdot i)$-degree clockwise direction to the horizon.

L_i: the ith row in each direction in an hexagonal mesh.

k_i, $i = 0, 1, 2$: the number of hops from the source node to the destination node along the d_i direction. Negative value means that the move is along opposite direction, $d_{[i+3]_6}$.

K_T: the number of state regions in region–change broadcasts.

TH_i, $1 \le i \le K_T - 1$: thresholds that divide the (workload) state space into K_T state regions.

λ_T: external task arrival rate at a node.

$1/\mu_T$: mean task execution time.

$\hat{p}_\ell$, $0 \le \ell \le L_{max}$: the probability that a task is associated with a laxity ℓ. L_{max} is the maximum task laxity in the task system.

$\{p_N(n), n \ge 0\}$:]the probability density function of the queue length of a node.

$\alpha(n)$: the rate of transferring tasks out of a node given that the node's state is n.

$\beta(n)$: the rate of transferring tasks into a node given that the node's state is n.

$\lambda(n)$: the composite (both external and transferred) task arrival rate at a node with $QL = n$.

$\gamma_j = P(N \ge j)$: the probability that a node's $QL \ge j$.

λ_{TT}: the rate of transferring tasks out of a node.

λ_{SC}: the rate of state–region–change broadcasts.

q_h: the probability of sending a task to a node h hops away.

R_i: the ith state region.

T_{ii}: the mean recurrent time of R_i; the expected time until the first transition into R_i given that a node's state starts in R_i.

T_i: the average time the continuous-time Markov chain, $\mathcal{M}$, constructed in Section 3.1 spends in R_i.

π_k: the stationary probability that the underlying discrete time Markov chain for $\mathcal{M}$ stays in state k.

p_f: the probability that a packet arriving (and receiving services if necessary) at a node will be forwarded to one of its neighboring nodes.

Λ: the throughput rate of a node.

p_c: the probability of a packet cutting through an intermediate node.

$f_{D_i}(t)$: the probability density function of the time needed for a packet to travel i hops.

$\overline{\ell_B}$: the mean length of broadcast packets.

$\overline{\ell_{TT}}$: the mean length of task transfer packets.

$\overline{\ell}$: the mean packet length.

λ_B, $\lambda_{A,B}$, $\lambda_{T,B}$: the rate of generating broadcast packets at a node, the rate of terminal broadcast packets arriving at a node, and the rate of transit broadcast packets arriving at a node, respectively.

λ_{TT}, $\lambda_{A,TT}$, $\lambda_{T,TT}$: the rate of generating task-transfer packets at a node, the rate of terminal task-transfer packets arriving at a node, and the rate of transit task-transfer packets arriving at a node, respectively.

ρ: the traffic intensity of the queuing network of interest.

r: the distribution of task laxity is assumed to be geometric with $\hat{p}_{\ell+1} = r \cdot \hat{p}_\ell$.

Acknowledgments

The work reported in this paper was supported in part by the U.S. National Science Foundation under Grant MIP-9203895, and the U.S. Office of Naval Research under Grants N00014-92-J-12080 and N00014-91-J-1226. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Office of Naval Research.

References

[1] K.G. Shin and Y.-C. Chang, "Load Sharing in Distributed Real-Time Systems with State Change Broadcasts," *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1,1224-1,142, Aug. 1989.

[2] K.G. Shin and C.-J. Hou, "Design and Evaluation of Effective Load Sharing in Distributed Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 7, pp. 704-719, July 1994.

[3] K.G. Shin, "HARTS: A Distributed Real-Time Architecture," *Computer*, vol. 24, no. 5, pp. 25-34, May 1991.

[4] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," *Proc. ACM Computer Network Performance Symp.*, pp. 47-55, 1982.

[5] A.B. Barak and A. Shiloh, "A Distributed Load-Balancing Policy for a Multicomputer," *Software-Practice and Experience*, vol. 15, no. 9, pp. 901-913, 1985.

[6] Y.T. Wang and R.J.T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. Computers*, vol. 34, no. 3, pp. 204-217, Mar. 1985.

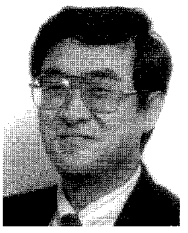
[7] L.M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. Software Eng.*, vol. 11, no. 5, pp. 491-496, May 1985.

[8] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Eng.*, vol. 12, no. 5, pp. 662-675, May 1986.

[9] C.-Y.H. Hsu and J.W.-S. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," *Proc. IEEE Sixth Int'l Conf. Distributed Computing Systems*, pp. 216-223, 1986.

[10] K.J. Lee and D. Towsley, "A Comparison of Priority-Based Decentralized Load Balancing in Distributed Computer Systems," *Proc. Performance '86*, pp. 70-78, May 1986.

[11] A. Hac and X. Jin, "Dynamic Load Balancing in a Distributed System Using a Decentralized Algorithm," *Proc. IEEE Seventh Int'l Conf. Distributed Computing Systems*, pp. 170-184, Sept. 1987.
[12] S. Pulidas, D. Towsley, and J.A. Stankovic, "Embedding Gradient Estimators in Laod Balancing Algorithms," *Proc. IEEE Eighth Int'l Conf. Distributed Computing Systems*, pp. 482-490, 1988.
[13] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Trans. Software Eng.*, vol. 14, no. 9, pp. 1,327-1,341, Sept. 1988.
[14] R. Mirchandaney, D. Towsley, and J.A. Stankovic, "Analysis of the Effect of Delays on Load Sharing," *IEEE Trans. Computers*, vol. 38, no. 11, pp. 1,513-1,525, Nov. 1989.
[15] O. Kremien and J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 11, pp. 747-760, Nov. 1992.
[16] J.F. Kurose and R. Chipalkatti, "Load Sharing in Soft Real-Time Distributed Computer Systems," *IEEE Trans. Computers*, vol. 36, no. 8, pp. 993-999, Aug. 1987.
[17] K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1,110-1,123, Aug. 1989.
[18] K.G. Shin, C.M. Krishna, and Y.H. Lee, "A Unified Method for Evaluating Real-Time Computer Controllers Its Application," *IEEE Trans. Automatic Control*, vol. 30, pp. 357-366, Apr. 1985.
[19] M.-S. Chen, K.G. Shin, and D.D. Kandlur, "Addressing, Routing, and Broadcasting in Hexagonal Mesh Multiprocessors," *IEEE Trans. Computers*, vol. 39, no. 1, pp. 10-19, Jan. 1990.
[20] D.D. Kandlur and K.G. Shin, "Reliable Broadcast Algorithms for HARTS," *ACM Trans. Computer Systems*, vol. 9, pp. 374-398, Nov. 1991.
[21] P. Kermani and L. Kleinrock, "Virtual Cut-Through; A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, pp. 267-286, 1979.
[22] J.W. Dolter, P. Ramanathan, and K.G. Shin, "Performance Analysis of Message Passing in HARTS: A Hexagonal Mesh Multiprocessor," *IEEE Trans. Computers*, vol. 40, no. 6, pp. 669-680, June 1991.
[23] C.-J. Hou and K.G. Shin, "Load Sharing with Consideration of Future Task Arrivals in Heterogeneous Distributed Real-Time Systems," *IEEE Trans. Computers*, vol. 44, no. 9, pp. 1,076-1,090, Sept. 1994.
[24] K.G. Shin and C.-J. Hou, "Analytic Models of Adaptive Load Sharing Schemes in Distributed Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 7, pp. 740-761, July 1993.
[25] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. John Wiley & Sons, 1975.



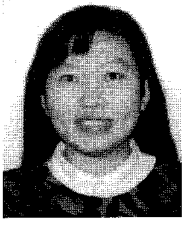
Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. He is a professor and director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor.

He has authored/coauthored more than 350 technical papers (more than 150 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He is currently writing (jointly with C.M. Krishna) a textbook *Real-Time Systems*, which is scheduled to be published by McGraw-Hill in 1996. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from the University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called **HARTS**, and middleware services for distributed real-time fault-tolerant applications.

He has also been applying the basic research results of real-time computing to multimedia systems, intelligent transportation systems, and manufacturing applications, ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes.

From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California at Berkeley, International Computer Science Institute, Berkeley, California, IBM T.J. Watson Research Center, and Software Engineering Institute at Carnegie Mellon University. He also chaired the Computer Science and Engineering Division, EECS Department, at the University of Michigan for three years beginning in January 1991.

Dr. Shin is an IEEE fellow, was the program chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the general chairman of the 1987 RTSS, the guest editor of the August 1987 special issue of *IEEE Transactions on Computers* on real-time systems, a program co-chair for the 1992 International Conference on Parallel Processing, and served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993, was a distinguished visitor of the IEEE Computer Society, and editor of *IEEE Transactions on Parallel and Distributed Systems*, and an area editor of *International Journal of Time-Critical Computing Systems*.



Chao-Ju Hou (S'88–M'94) received the BSE degree in electrical engineering in 1987 from National Taiwan University, the MSE degree in electrical engineering and computer science (EECS), the MSE degree in industrial and operations engineering, and the PhD degree in EECS, all from the University of Michigan, Ann Arbor, in 1989, 1991, and 1993, respectively.

She is currently an assistant professor in the Department of Electrical and Computer Engineering at the University of Wisconsin, Madison. Her research interests are in the areas of distributed and fault–tolerant computing, protocol design and implementation for real–time communications, estimation and decision theory, and performance modeling/evaluation. She is a recipient of the Women in Science Intiative Award from the University of Wisconsin–Madison. She is a member of the IEEE Computer Society, ACM Sigmetrics, and the Society of Woman Engineers.