Establishment of Isolated Failure Immune Real-Time Channels in HARTS

Qin Zheng, *Member, IEEE,* and Kang G. Shin, *Fellow, IEEE*

***Abstract*— Fault-tolerant, real-time communication in distributed systems is very important yet difficult to achieve. Traditional protocols like the TCP/IP achieve reliable communication through acknowledgment and retransmission schemes, where one achieves the reliability at the cost of performance. In this paper, we discuss how both the timeliness and fault-tolerance of communication can be achieved by using the concept of *real-time channel* [1] and exploring the inherent spatial redundancy of a given network topology. Specifically, we show how *isolated failure immune* real-time channels can be established in wrapped hexagonal mesh networks, thus ensuring timely delivery of messages in the presence of network component failures as long as the failures are isolated. This kind of fault-tolerance cannot be achieved with other commonly-known topologies like rings, rectangular meshes, and hypercubes. The proposed approach is to be implemented in an experimental distributed real-time system, called HARTS [2], whose construction is underway.**

***Index Terms*— Distributed computing systems, fault-tolerant real-time communications, wrapped hexagonal mesh, isolated failure immune networks, real-time channels.**

I. INTRODUCTION

RELIABLE and timely delivery of messages in point-to-point packet-switching networks has long been a challenge to system designers. To avoid unpredictable queueing delays at transmission links/nodes, real-time messages are usually transmitted along a pre-determined path on which the network resources are reserved to guarantee the actual delivery delay to be less than a pre-specified bound. Examples include circuit-switching transmission, synchronous transmission mode (STM), and the recently-proposed real-time channel [1], [3], [4]. Sending messages along a static path, however, reduces the fault-tolerance of real-time traffic, since a node/link failure in the path would disable the channel that runs over the path.

To alleviate this problem, Zheng and Shin [5] proposed a semi-dynamic routing scheme for real-time channels. By reserving resources at some extra links and nodes, these real-time channels with extra links and nodes can tolerate any *single* node/link failure in the network.

Manuscript received July 3, 1992; revised February 14, 1994.

The work reported in this paper was supported in part by the Office of Naval Research under Grant N00014-92-J-1080 and the National Science Foundation under Grants MIP-9012549 and MIP-9203895.

Q. Zheng is with Mitsubishi Electric Research Laboratories, Inc., Cambridge, MA 02139 USA (e-mail: zheng@merl.com).

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 (e-mail: kgshin@eecs.umich.edu).

IEEE Log Number 9408128.

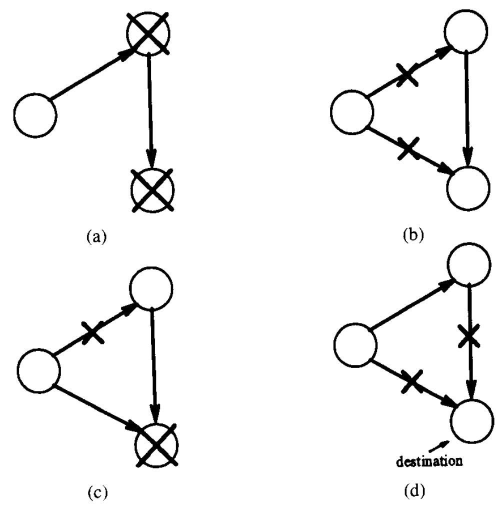


Fig. 1. Four types of nonisolated component failures. (a) Two faulty nodes which are adjacent. (b) Two faulty links which originate from the same functioning node. (c) Same as (b) except that one link is made unusable (thus regarded faulty) by the failure of another node. (d) Two incoming faulty links of the destination node.

Making a real-time channel more robust than just tolerating a single component failure turns out to be very difficult and requires reservation of significantly more network resources. In the Real-time Computing Laboratory of the University of Michigan, we have been exploring various network topologies to solve this problem and have found a wrapped hexagonal mesh [6] to be *isolated failure immune* (IFI). An IFI real-time channel guarantees the timely delivery of messages in the presence of network component failures as long as the failures are isolated with respect to the channel. Node failures are said to be *isolated* with respect to a real-time channel if the source and destination nodes of the channel are not faulty and any two faulty nodes in the channel are not adjacent. Link failures (a link failure is caused by either the failure of the link itself or the failure of the node which the link leads to) are said to be isolated if any two faulty links are not originated from the same functioning node or directed to the destination node. Fig. 1 shows four types of nonisolated component failures. Another two types of nonisolated failures are the failures of the source and destination nodes. Fig. 2 shows an example of an IFI channel from node 1 to node 6 and one pattern of (tolerable) isolated failures.

The isolated failure immune problem for *undirected networks* was first discussed in [7] where the authors proved

1045-9219/95$04.00 © 1995 IEEE

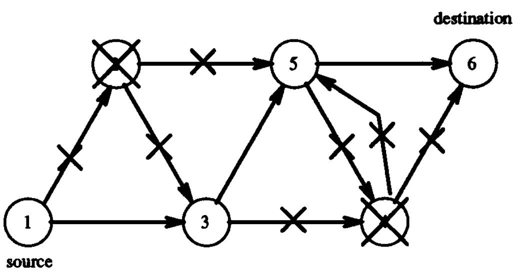


Fig. 2. An IFI channel and one pattern of tolerable link/node failures.

that a 2-tree[1] is a minimum IFI network. In other words, any IFI network must contain a spanning 2-tree. This result excludes almost all commonly-used network topologies (e.g., rings with more than 3 nodes, rectangular meshes/cubes, and hypercubes) from the candidate set of IFI networks, except for the hexagonal mesh.

An IFI real-time channel has the following advantages over a basic real-time channel:

- High Reliability: The channel can tolerate a large number of component failures as long as they are isolated. For example, the IFI channel shown in Fig. 2 can tolerate as many as 7 faulty links and 2 faulty nodes, which represent 70% of the links and 33% of the nodes that the channel runs through.
- Easy Failure Detection: Non-isolated failures in the network can be easily detected by a node using only its local information, i.e., the status of its own links and its neighbors. This makes the system maintenance extremely easy. A node can safely shut down one of its links or itself by checking the status of its links and neighbors.
- Accommodation of Emergency Messages: A path between any pair of nodes in a network can always be constructed using only those links whose failure will not cause nonisolated failures. So, in the absence of network component failures, it is always safe to break down some links of existing IFI real-time channels and use their full link transmission bandwidth to handle emergency messages.

This paper is organized as follows. For completeness, the HARTS topology and its routing algorithm are reviewed in Section II. The concept of real-time channel is also briefly discussed there. Section III presents the schemes of establishing isolated failure immune real-time channels in HARTS. The paper concludes with Section IV. Proofs of theorems in this paper are given in an appendix.

II. HARTS AND REAL-TIME CHANNELS

HARTS is an experimental distributed real-time system currently being built in the Real-Time Computing Laboratory of the University of Michigan [2]. As shown in Fig. 3, the interconnection network of HARTS is a wrapped hexagonal mesh which can be defined as follows.

[1] A 2-tree can be constructed as follows. Two nodes connected by a link is a 2-tree. A new node can be added to a 2-tree by connecting it to two neighboring nodes in the 2-tree.

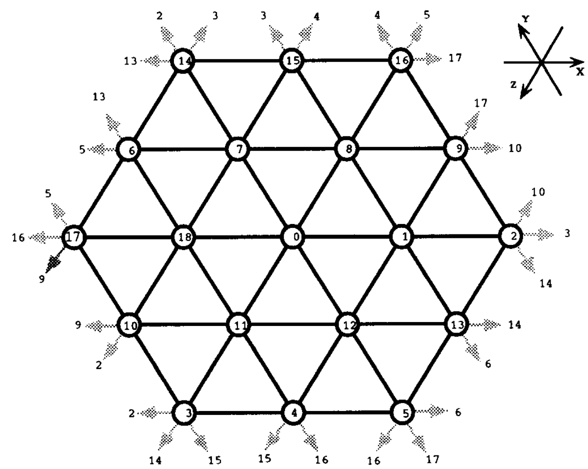


Fig. 3. A wrapped hexagonal mesh of size 3.

Definition 2.1: Let $[a]_b$ denote a mod b. Then a wrapped hexagonal mesh of size n (or the number of nodes on each peripheral edge) is composed of $N = 3n(n-1)+1$ nodes, labeled from 0 to $N-1$, such that each node s has six neighbors $[s+1]_N, [s+3n(n-1)]_N, [s+3n-2]_N, [s+3n^2-6n+3]_N, [s+3n^2-6n+2]_N$, and $[s+3n-1]_N$, in the $X, -X, Y, -Y, Z, -Z$ directions, respectively.

It was proved in [6] that a wrapped hexagonal mesh is homogeneous. Consequently, any node can view itself as the center of the mesh. Let m_x, m_y, and m_z be, respectively, the number of hops (negative values mean the moves in negative directions) from the source node to the destination node along the X, Y, and Z directions on a shortest path. The following routing algorithm [6] determines the values of m_x, m_y, and m_z for the shortest paths from a source node s to a destination node d in a wrapped hexagonal mesh of size n:

Algorithm 2.1 (Routing in HARTS):

- Step 0. Set $m_x := 0$, $m_y := 0$, $m_z := 0$. Let $p = 3n^2 - 3n - 1$, $k = (d-s)$ mod p, $r = (k-n)$ div $(3n-2)$, $t = (k-n)$ mod $(3n-2)$.
- Step 1. If $k < n$ then set $m_x := k$, stop. Else if $k > 3n^2 - 4n + 1$ then set $m_x = k - 3n^2 + 3n - 1$, stop. Else goto Step 2.
- Step 2. If $t \le n + r - 1$ then
  - If $t \le r$ then set $m_x := t - r$, $m_z := n - r - 1$, stop.
  - If $t \ge n - 1$ then set $m_x := t - n + 1$, $m_y := r + 1 - n$, stop.
  - Else set $m_y := r - t$, $m_z := n - t - 1$, stop.

  else

  - If $t \le 2n - 2$ then set $m_x := t + 2 - 2n$, $m_y := r + 1$, stop.
  - If $t \ge 2n + r - 1$ then set $m_x := t - 2n - r + 1$, $m_z := -r - 1$, stop.
  - Else set $m_y := 2n + r - t - 1$, $m_z := 2n - t - 2$, stop.

To send a packet, the source node calculates m_x, m_y, m_z using the above algorithm. It then sends the packet to an appropriate neighbor. Intermediate nodes update these values to indicate the remaining number of hops to take in X, Y, and Z directions before forwarding the message. Hence $m_x = m_y = m_z = 0$ indicates that the packet has reached its destination. The readers are referred to [6], [2] for a detailed account of the wrapped hexagonal mesh and its routing algorithm.

To meet the requirement of real-time communication, the HARTS communication subsystem is designed to support real-time channels [8]. A real-time channel is a simplex virtual connection between the source and destination nodes which guarantees the delivery of packets within a user-specified end-to-end delay bound. Two techniques are used to achieve this goal: admission control of channels and deadline scheduling of packet transmissions.

Admission control requires those processes requesting real-time communication to establish real-time channels before starting packet transmission. A channel-establishment request may be accepted or rejected, depending on the current network-load condition. Admission control is necessary because packet-delay bounds cannot be guaranteed without controlling the network load.

Packet transmissions are scheduled as follows. Real-time packets have a higher transmission priority than nonreal-time packets. Each real-time packet is assigned a deadline over each link it traverses which is determined according to the packet's generation time at the source node and the delay bounds d^i's assigned to the links of the real-time channel. When several real-time packets contend for use of the same link, the packet with the earliest deadline is transmitted first. The advantages of using deadline scheduling are the minimization of contention delays and protection between established channels [3], [4].

To set up a real-time channel, the requesting process must determine two parameters, T and C, specifying its traffic generation pattern, where T is the minimum packet inter-generation time and C is the maximum packet transmission time (directly proportional to the maximum packet length). It is reasonable to assume prior knowledge of these parameters for many real-time applications, such as interactive voice/video transmission and real-time control/monitoring. In other applications where the traffic pattern is less predictable, the estimated values of T and C could be used. A process may exceed its pre-specified maximum packet generation rate at the risk that its packets may be delivered with delays longer than the pre-specified bound or may even be discarded, but due to the deadline scheduling of packet transmissions, this particular process will not affect the guarantees of the other existing channels.

The process then sends a channel establishment request message containing T and C together with the end-to-end packet delay bound D and addresses of the source and destination nodes to a special node containing the *Network Manager* (NM), which maintains the information of all existing channels and executes the channel establishment algorithm of [3], [4] to check if the requested channel can be established over a specified route under the current network load condition. If the channel can be established, the algorithm also calculates the link delay bounds d^j's which will be used to determine the deadlines of the channel's packets.

Interested readers are referred to [1], [9], [3], [4] for a detailed discussion of real-time channels.

III. Isolated Failure Immune Real-Time Channels in HARTS

This section discusses how real-time channels can be enhanced to be Isolated Failure Immune (IFI) in HARTS. The first step is to find an *IFI path*, which is defined as a subnetwork containing a directed path from the source to the destination in the presence of any isolated failures. Let $d_S(v_1, v_2)$ denote the minimum number of hops (i.e., distance) from node v_1 to node v_2 in a network S. The following theorem gives a sufficient condition for S to be an IFI path from source node v_s to destination node v_d in a general directed network.

Theorem 3.1: A subnetwork S containing the source node v_s and the destination node v_d is an IFI path from v_s to v_d if

- C1: Every node $v \in S, v \neq v_d$, has at least two outgoing links to two other nodes, say v_1 and v_2, such that $d_S(v_1, v_d) < d_S(v, d_d)$, $d_S(v_2, v_d) \leq d_S(v, v_d)$, and v_1, v_2 are adjacent,
- C2: There is no loop in S whose nodes are all of the same distance $d > 1$ to the destination node v_d.

From the above theorem, one can see that each node in an IFI path needs only two outgoing links. We call one of them the *primary* link and the other the *secondary* link.

The primary link is the one which leads to a node closer to the destination. One can choose the primary link from the shortest path as determined by Algorithm 1. In case there exist multiple choices, i.e., more than one of m_x, m_y, m_z are nonzero, we will use the following algorithm to select a primary link L.

Algorithm 3.1 (Selection of the Primary Link): L Let $abs(x)$ and $sign(x)$ denote the absolute value and the sign of x, respectively, and let X, $-X$, Y, $-Y$, Z, $-Z$ denote the outgoing links of a node along the six different directions.

Then,
If $abs(m_x) > 1$ then set $L := sign(m_x)X$
else if $abs(m_y) > 1$ then set $L := sign(m_y)Y$
else if $abs(m_z) > 1$ then set $L := sign(m_z)Z$
else if $abs(m_x) = 1$ then set $L := sign(m_x)X$
else if $abs(m_y) = 1$ then set $L := sign(m_y)Y$
else if $abs(m_z) = 1$ then set $L := sign(m_z)Z$.

The logic behind the above algorithm is that one should first select the primary link in the direction which is more than one hop away from the destination. If there are more than one such directions, the primary link is selected in the order of X, Y, Z. On the other hand, if there are no such directions, the primary link is selected in the direction which is one hop away from the destination in the order of X, Y, Z. As will be clear later, the selection of the primary links in this specific way will facilitate the determination of the secondary links and reduce the number of nodes/links of the resulting IFI channel.

In a wrapped hexagonal mesh network, to ensure that the secondary link does not lead to a node which is farther away from the destination, it must be either 60 degree above or 60 degree below the primary link. Here "above" means counterclockwise and "below" means clockwise. We use the notation $L+1$ to denote the link which is 60 degree above L, and $L-1$ the one which is 60 degree below L. For example, if $L = X$, then $X + 1 = -Z$ and $X - 1 = -Y$.

Let $node[i]$ denote the ith node of an IFI path, and $node[i].p$ and $node[i].s$ denote the node's primary and secondary links, respectively. We propose the following algorithm to construct an IFI path from v_s to v_d.

Algorithm 3.2 (Construction of an IFI Path):

- Step 1. Calculate m_x, m_y, m_z for the source node v_s using Algorithm 1. Notice that at most two of them can be nonzero.
- Step 2. Set $i := 1$ and $node[1] := v_s$. Set the initial rotating direction for the secondary link $R := 1$ if one of the following conditions holds: (1) $abs(m_y) > abs(m_x) = 1$, (2) $abs(m_z) \geq abs(m_y) = 1$, (3) $abs(m_x) > 1, m_z \neq 0$, and (4) $abs(m_x) = abs(m_z) = 1$. Otherwise, set $R := -1$.
- Step 3. Calculate the primary link $L(i)$ using Algorithm 2. If $i > 1$, $L(i) \neq L(i-1)$, and $node[i-1]$ is not adjacent to v_d, set $R := -R$.
- Step 4. Set $node[i].p := L(i)$, $node(i).s := L(i)+R$, and set $node[i+1]$ to be the node which the secondary link of $node[i]$ leads to. Update m_x, m_y, m_z for $node[i+1]$.
- Step 5. If $node[i+1] = node[i-1]$, then set $node[i+1] := v_d$ and stop. The destination node has been reached. Otherwise, set $i := i+1,\ R := -R$, goto Step 3.

The correctness of Algorithm 3 is proved by the following theorem.

Theorem 3.2: The subnetwork obtained from Algorithm 3 is an IFI path from v_s to v_d.

We make several remarks on Algorithm 3 as follows.

1) In Step 4, the address of $node[i+1]$ can be obtained from that of $node[i]$ using Definition 1, which gives the addresses of the six neighboring nodes of a node in six directions. The values of m_x, m_y, m_z for $node[i+1]$ can be updated directly with Algorithm 1 using the address of $node[i+1]$. But a simpler way of doing this is as follows. Let u be the direction of link $node[i].s$ and v, w be the remaining two directions. Let $s = 1$ if link $L(i) + R$ is at the positive direction of u and $s = -1$ otherwise. Then, if ($m_u = m_v = 0$ and $sm_w > 0$) or ($m_u = m_w = 0$ and $sm_v > 0$), update $m_v := m_v - s, m_w := m_w - s$. Otherwise, update $m_u := m_u - s$. The correctness of this algorithm can be verified by placing the destination node v_d at the center of the wrapped hexagonal mesh and checking the changes of m_x, m_y, m_y as one moves from $node[i]$ to $node[i+1]$ along link $node[i].s$.
2) In Step 2, the initial rotating direction R for the secondary link is chosen such that if $node[1]$ has two links both on shortest paths[2] to the destination nodes, $node[1].s$ will take one of them. In this way, the resulting IFI path needs less links and nodes than all other cases. The way in which the primary link is chosen in Algorithm 2 also serves this purpose.
3) Since the primary links are always on the shortest path to the destination, they form a shortest path sinking tree to the destination. In other words, if a packet generated at any node in S is always forwarded using the primary links, it will take a minimum number of hops to the destination. This fact results in the following routing policy at each node: an arriving packet should be forwarded via the primary link whenever possible. The secondary link is used only if the primary link is down.

[2]Note that there could be multiple shortest paths between a pair of nodes.

We now discuss how an IFI real-time channel can be established over an IFI path obtained from Algorithm 3. The procedures to establish an IFI real-time channel are composed of the following three steps.

- Step 1. Calculate the packet-delay bound over each link of the channel.
- Step 2. Calculate the end-to-end delay bound using the link-delay bounds.
- Step 3. If the end-to-end delay bound is not larger than the requested one, the channel can be established. Calculate the link delay bounds to be assigned to the channel. Otherwise, the channel establishment request is rejected.

Results in [3], [4] can be used for the calculation of the link-delay bounds in Step 1. Let $node[i], i = 1, \cdots, k$ be the nodes of an IFI path obtained from Algorithm 3, where $node[1]$ is the source node and $node[k]$ is the destination node. Let $d[i].p$ and $d[i].s$ be the delay bounds over the primary and secondary links of $node[i]$, respectively. Then the end-to-end packet delivery delay bound in Step 2 can be calculated using the following algorithm.

Algorithm 3.3 (Calculation of the Packet Delivery Delay Bounds): The packet delivery delay bound $d[i]$ from $node[i]$ to the destination node $node[k]$ can be calculated as follows:

$$d[k-1] = \max\{d[k-1].p,\ d[k-1].s + d[k-2].p\},$$

$$d[k-2] = \max\{d[k-2].p,\ d[k-2].s + d[k-1].p\},$$

$$d[i] = \max\{d[i].p + d[i_p],\ d[i].s + d[i_s]\} \quad i = k-3, \cdots, 1.$$

where $node[i_p], node[i_s]$ are the nodes to which the primary and secondary links of $node[i]$ lead, respectively.

The correctness of Algorithm 4 can be verified as follows. From the proof of Theorem 2, the connections between $node[k-2], node[k-1]$, and $node[k]$ are shown in Fig. 4(b), from which the first two equations can be obtained. For $1 \leq i \leq k-3$, $node[i]$ is connected to $node[i_p]$ and $node[i_s]$ in the way shown in Fig. 4(a), which proves the remaining $k-3$ equations. Since i_p and i_s are always larger than i for $i \leq k-2$, the maximum delay bound from $node[i]$ to $node[k]$ can be obtained from the above equations.

If $d[1] \leq D$, the IFI real-time channel can be established, and we need to determine the link-delay bounds to be assigned to the channel. As discussed in [3], [4], the link-delay bounds of the channel should be set as large as possible to reduce the

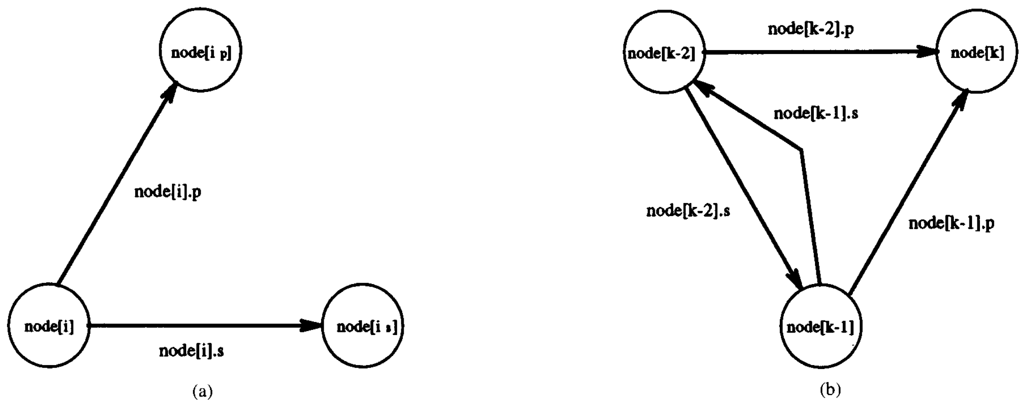


Fig. 4. Calculation of $d[i]$'s.

channel's influence on the links' ability to establish more real-time channels in future. This can be done using the following algorithm.

Algorithm 3.4 (Assignment of Link-Delay Bounds):

- Step 1. In Algorithm 4, for $i = k-1, \cdots, 1$, record the link (i.e., the primary or secondary link) $\ell[i]$ on which the maximum is attained for $d[i]$. Notice that there could be two such links for $i = k-2$ or $i = k-1$.
- Step 2. Record all the links traversed as one goes from $node[1]$ to $node[k]$ using only the links recorded in Step 1. This gives a $\underline{critical\ path}$ from the source to the destination which has the end-to-end delay bound $d[1]$ as calculated from Algorithm 4.
- Step 3. Let N be the total number of links on the critical path. For each link ℓ_j on the critical path, set the channel's delay bound $d^j := d_j + (D - d[j])/N$, where d_j is the minimum link-delay bound calculated for ℓ_j.
- Step 4. Recalculate $d[i]$'s in Algorithm 4 with the link delay bounds on the critical path replaced by d^j's. The channel's delay bounds of the links not on the critical path can then be calculated as the differences of $d[i]$'s of the nodes they connect.

In summary, we have the following algorithm for the establishment of an IFI real-time channel.

Algorithm 3.5 (Establishment of an IFI Real-Time Channel):

- Step 1. Calculate the minimum packet delay bounds $d[i].p_{min}$ and $d[i].s_{min}$ over the primary and secondary links of $node[i], i = 1, \cdots, k-1$.
- Step 2. Calculate the end-to-end delay bound $d[1]$ from Algorithm 4.
- Step 3. If $d[1]$ is larger than the user-requested end-to-end delay bound D, the channel request is rejected. Otherwise, the channel can be established with the link delay bounds calculated from Algorithm 5.

We now give an example to demonstrate the above ideas. Fig. 5 shows a portion of a hexagonal mesh. We want to establish an IFI real-time channel from node 1 to node 8 with channel parameters $(T, C, D) = (100, 5, 70)$.

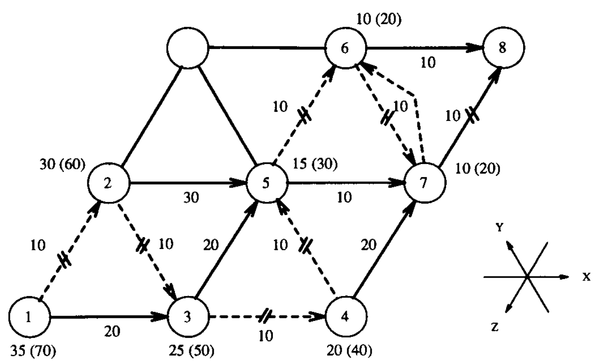


Fig. 5. An IFI real-time channel from node 1 to node 8. Solid arrows represent the primary links, and dashed arrows represent secondary links. Link delays assigned to the channel are shown near the links.

We first construct an IFI path from node 1 to node 8 using Algorithm 3. For $i = 1$, $node[1] =$ node 1. $(m_x, m_y, m_z) = (2, 0, -2)$. The initial rotating direction for the secondary link $R = 1$ since $abs(m_x) > 1$ and $m_z \neq 0$. From Algorithm 2, the primary link is calculated to be $node[1].p = L(1) = X$, and the secondary link is $node[1].s = L(1) + 1 = -Z$.

Set the next node to one which link $-Z$ leads to, then $node[2] =$ node 2. Update m_x, m_y, m_z for $node[2]$ as follows. The direction of $-Z$ is Z, so $u = Z$, and $v = X, w = Y$. Also, $s = -1$. Since $m_w = 0$ and $sm_v = -2 < 0$, we only need to update $m_u := m_u - s = -2 + 1 = -1$. Thus, for node 2, $(m_x, m_y, m_z) = (2, 0, -1)$.

Repeating the above procedure, we get an IFI path as shown in Fig. 5, where the primary links are denoted by solid arrows and the secondary links by dashed arrows. It is not difficult to see that a packet can be transmitted from node 1 to node 8 in the presence of any isolated failures. Also, all the primary links and the nodes form a shortest path sinking tree to the destination node.

We now establish an IFI real-time channel over the IFI path thus obtained by assigning delay bounds to the links using Algorithm 6. Suppose there is no other real-time traffic in the network. Then, for $i = 1, \cdots, 8$, $d[i].p_{min} = d[i].s_{min} = C = 5$. Using Algorithm 4, $d[i]$'s are calculated and shown

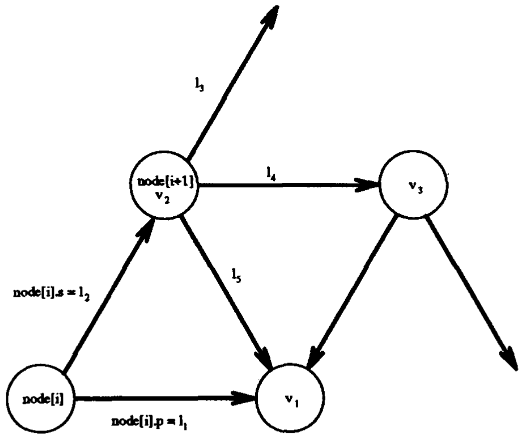


Fig. 6. Proof of the adjacency of v_1 and v_2.

near each node in Fig. 5. The requested real-time channel can be established since $d[1] = 35 < D = 70$.

The critical path can be determined by recording the links over which the maximum is achieved in Algorithm 4, which is in this example the ones marked by "//" in Fig. 5. There are a total of $N = 7$ links on the critical path. The channel's delay bounds over the links of the critical path are thus $d^j = d_j + (D - d[1])/N = 5 + (70 - 35)/7 = 10$. The updated values of $d[i]$'s calculated from Algorithm 4 are shown in the parentheses near each node. Then, the channel's delay bounds on the other links can be calculated as the differences of $d[i]$'s of the nodes they connect, which are shown near each link in Fig. 5.

From the above example, one can see that an IFI channel usually needs 3 to 4 times more links than a basic real-time channel. This means that more transmission bandwidth needs to be reserved for an IFI channel. This "over-reservation" reduces a network's ability of accommodating real-time channels. However, as discussed in [3], [4], real-time channels make only "soft" reservation since any unused bandwidth can be used for non real-time traffic. In this sense, the "cost" of an IFI channel to non real-time traffic is the same as a basic channel. So, in a network with a large portion of traffic being non real-time, IFI real-time channels is an economical means of achieving fault-tolerant real-time communications.

IV. Conclusion

We have in this paper discussed how IFI real-time channels can be established in HARTS by exploiting its wrapped hexagonal mesh topology. Thus far, the researchers of the HARTS project have implemented most of the basic real-time channel on the top of xKernel [10]. We do not expect to face any difficulty in enhancing the HARTS communication subsystem with IFI real-time channels due mainly to the following features of HARTS.

- Programmable Routing Controller: HARTS achieves maximum flexibility by using a custom-designed Programmable Routing Controller (PRC) [2], which can implement various switching and routing schemes. (The PRC is the front-end interface for each node in HARTS and contains six pairs of transmitters and microprogrammable receivers.) Thus, enhancing the basic real-time channel to be IFI is very simple; it requires a simple modification of the microprogram residing in each receiver of the PRC. HARTS can easily be made to support different types of real-time channels ranging from basic or single-failure-immune (SFI) channels [5] to IFI channels.
- Bit-by-bit feedback transmission links: The current version of HARTS is equipped with bit-by-bit feedback transmission links. Each receiver acknowledges every bit it receives from a sender. So, each node has up-to-date information about its neighboring nodes. This provides the error-detection capability required by the IFI channels.

In addition to its other salient features discussed in [2] such as homogeneity and fine scalability, the enhancement of basic real-time channel with the IFI capability will make HARTS an even more promising architecture for distributed fault-tolerant real-time applications.

Appendix

Proof of Theorem 3.1: From C1, every node $v \in S$ except the destination node has two outgoing links ℓ_1 and ℓ_2 which lead to a pair of adjacent nodes v_1 and v_2, respectively. Then, a packet will be blocked at node v only if (1) both ℓ_1 and ℓ_2 are disabled, or (2) both v_1 and v_2 are disabled, or (3) ℓ_1 and v_2 are disabled, or (4) ℓ_2 or v_1 are disabled. All of these situations represent nonisolated failures. Thus, in the absence of nonisolated failures, a packet from the source node can always progress unless it has reached the destination. Further, C1 ensures a packet will not move away from the destination, and C2 ensures a packet will not move around forever without reaching the destination node or cycling in a loop in which each node is directly connected to v_d. Since v_d cannot have more than one faulty incoming link, we conclude that a packet from the source node can always reach the destination node in a finite number of steps. □

Proof of Theorem 3.2: We prove that the resulting subnetwork S satisfies C1 and C2 of Theorem 1.

For any $node[i] \neq v_d$ in S, let v_1 and v_2 be the two respective nodes which links $node[i].p$ and $node[i].s$ enter. From the algorithm, $node[i+1] = v_2$. Thus $v_2 \in S$. To show that v_1 is also in S, and v_1 and v_2 are adjacent, we first prove that there is a link in S from v_2 to v_1.

Since a secondary link will never lead to the destination node, $v_2 \neq v_d$. Thus, $node[i+1]$ always has two outgoing links $node[i+1].p$ and $node[i+1].s$ in S. Assume $node[i].s$ is 60 degree above $node[i].p$. As shown in Fig. 6, from the direction of $node[i].p$ (which is on the shortest path from $node[i]$ to v_d), $node[i+1].p$ (i.e., the shortest path from $node[i+1]$ to v_d) has only three choices: ℓ_3, ℓ_4, ℓ_5. We claim that $node[i+1].p$ cannot take ℓ_3 since otherwise, from Algorithm 2, $node[i].p$ would have taken ℓ_2 instead of ℓ_1. If $node[i+1].p = \ell_5$, the primary link of $node[i+1]$ is the link from v_2 to v_1. Otherwise, $node[i+1].p = l_4$. From Algorithm 3, $node[i+1].s$ should be 60 degree below $node[i+1].p$ since $node[i].p$ and $node[i+1].p$ have the same direction and $node[i]$ is not adjacent to v_d ($node[i+1].p$ would otherwise have taken ℓ_5).

Thus $node[i+1].s = \ell_5$ is the link from v_2 to v_1. Similarly, it can be proved that there is a link from v_2 to v_1 in S when $node[i].s$ is 60 degree below $node[i].p$.

We now prove that $v_1 \in S$. If $node[i+1].s = \ell_5$, then $v_1 = node[i+2] \in S$. Otherwise, from the above proof, $node[i+1].p = \ell_5$. If $v_1 = v_d$, from Algorithm 3, $node[i+1].s$ directs back to $node[i]$. Then, $v_1 = node[i+2] \in S$. Otherwise, as shown in Fig. 6, $v_3 = node[i+2]$. Continuing this induction, we can conclude that either $v_1 \in S$, or the six neighbors of v_1 all have primary links directed to v_1. The latter case implies $v_1 = v_d$. Thus, $v_1 \in S$. Since there is a link in S from v_2 to v_1, v_1 and v_2 are adjacent in S.

Further, since $node[i].p$ is on the shortest path, $d_S(v_1, v_d) = d_S(node[i], v_d) - 1 < d_S(node[i], v_d)$. Since there exists a link in S from v_2 to v_1, $d_S(v_2, v_d) \leq d_S(v_1, v_d) + 1 = d_S(node[i], v_d)$. Thus C1 is proved.

We now prove that there does not exist any loop all of whose nodes are of a constant distance $d > 1$ to v_d by contradiction. First, notice that such a loop contains only secondary links since a primary link connects two nodes of different distances to v_d. Then, all the primary links of the nodes in the loop must lead to a common node v. This is from the fact proved above that either $node[i+1].p$ or $node[i+1].s$ must lead to a node v which $node[i].p$ leads to. But $node[i+1].s$ can not lead to v since it must lead to a node of the same distance to v_d as that of $node[i]$. This is possible only if $v = v_d$, i.e., $d = 1$. Thus, C2 is proved. □

REFERENCES

[1] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J. Select. Areas Commun.*, vol. SAC-8, pp. 368–379, Apr. 1990.

[2] K. G. Shin, "HARTS: A distributed real-time architecture," *IEEE Comput.*, vol. 24, pp. 25–35, May 1991.

[3] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Trans. Commun.*, vol. 42, pp. 1096–1105, Mar. 1994.

[4] Q. Zheng, "Real-time fault-tolerant communication in computer networks," Ph.D thesis, Univ. of Michigan, 1993. (PostScript version of the thesis is available via anonymous FTP from ftp.eecs.umich.edu in directory outgoing/zheng.)

[5] Q. Zheng and K. G. Shin, "Fault-tolerant real-time communication in distributed computing systems," in *Proc. 22nd Annual Int. Symp. Fault-tolerant Comput.*, pp. 86 – 93, 1992.

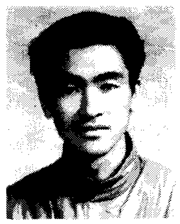
[6] M.-S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, routing and broadcasting in hexagonal mesh multiprocessors," *IEEE Trans. Comput.*, vol. 39, no. 1, pp. 10–18, Jan. 1990.

[7] A. M. Farley, "Networks immune to isolated failures," *Networks*, vol. 11, pp. 255–268, 1981.

[8] D. D. Kandlur and K. G. Shin, "A communication subsystem for HARTS: An experimental distributed real-time system," submitted for publication.

[9] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," in *Proc. Int. Conf. Distrib. Comput. Syst.*, pp. 300–307, May 1991.

[10] K. Shin, D. Kandlur, D. Kiskis, P. Dodd, H. Rosenberg, and A. Indiresan, "A distributed real-time operating system," *IEEE Software*, pp. 58–68, Sept. 1992.



Qin Zheng (S'89–M'94) received the B.S. and M.S. degrees in electrical engineering from the University of Science and Technology of China in 1982 and 1985, respectively, and Ph.D. degree in electrical engineering from the University of Michigan in 1993.

He joined Mitsubishi Electric Research Labs, Inc., Cambridge Research Center in 1993 and is currently working on high-speed computer networks and distributed real-time industrial systems.

Kang G. Shin (S'75–M'78–SM'83–F'94) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D degrees in electrical engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively.

He is Professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan. He has authored/coauthored over 250 technical papers (more than 110 of these in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He has also been applying the basic research results of real-time computing to manufacturing-related applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. Recently, he has initiated research on the open-architecture Information Base for machine tool controllers. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. 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**, to validate various architectures and analytic results in the area of distributed real-time computing. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA. He also chaired the Computer Science and Engineering Division, EECS Department, The University of Michigan for three years beginning Jan. 1991. He 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 1987 Aug. 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 numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-93, is a Distinguished Visitor of the Computer Society of the IEEE, an Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED COMPUTING, and an Area Editor of *International Journal of Time-Critical Computing Systems*.

In 1987, Dr. Shin 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.