A DISTRIBUTED I/O ARCHITECTURE FOR HARTS

Kang G. Shin and Greg Dykema

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122

Abstract

The issue of I/O device access in HARTS — a distributed real-time computer system under construction at the Real-Time Computing Laboratory (RTCL), The University of Michigan — is explicitly addressed. Several candidate solutions are introduced, explored, and evaluated according to cost and complexity, reliability, and performance: (1) "node-direct" distribution with the intra-node bus and a local I/O bus, (2) use of dedicated I/O nodes which are placed in the hexagonal mesh as regular applications nodes but which provide I/O services rather than computing services, and (3) use of a separate I/O network which has led to the proposal of an "interlaced" I/O network. The interlaced I/O network is intended to provide both high performance without burdening node processors with I/O overhead as well as a high degree of reliability. Both static and dynamic multi-ownership protocols are developed for managing I/O device access in this I/O network. The relative merits of the two protocols are explored and the performance and accessibility which each provide are simulated.

1 Introduction

To date, work on distributed computing systems — by which we mean loosely-coupled networks of processing elements — has centered on interconnection networks, programming and communications paradigms, algorithms, and task decomposition. However, little has been said specifically about the I/O subsystem in a distributed environment, despite its obvious importance. Work which has been done has focused primarily on the hypercube and has not addressed the accessibility of I/O devices in case of failures in the system [9, 11], and research which looks at fault-tolerance has not considered the multi-accessibility required by a distributed environment [7]. Clearly, a computer can process data no faster than it can acquire the data; this has been the rationale behind the attention paid to memory subsystems and increasing the accessibility and access speed of memories. But one cannot assume that data somehow appear in memory for the computer to use and process. We must look realistically at the accessibility and capability of I/O devices, especially as more powerful computing systems place more and more demands on all of their subsystems.

Distributed computing systems are being used for demanding applications, such as binary hypercubes for scientific processing, and a variety of systems, such as HARTS (Hexagonal Architecture for Real-Time Systems [1, 3, 4, 8]), for real-time processing. The demands of a real-time computing system include both high performance and reliability [10]. In the case of a distributed I/O subsystem, this also means accessibility.

The focus of this paper is on developing an I/O subsystem for HARTS, an experimental system for research in distributed real-time computing under construction at the RTCL. HARTS uses a wrapped hexagonal mesh as its interconnection topology. The wrapped hexagonal topology is known to be quite attractive due mainly to its hardware constructibility, fine scalability, and fault–tolerance. (See [1, 2] for a detailed account of the advantages of this topology and its comparison with other topologies.) We will begin with a brief discussion of the architecture of HARTS. Following this we will examine a variety of ways of implementing the I/O subsystem in this distributed environment. Although much of what will be said will apply to many other distributed computer system topologies, we will look specifically at I/O device placement and management in HARTS. Each proposal for the design of an I/O subsystem will be analyzed with regard to cost and complexity, accessibility, and performance.

CH2887-8/90/0000/0332$01.00 © 1990 IEEE

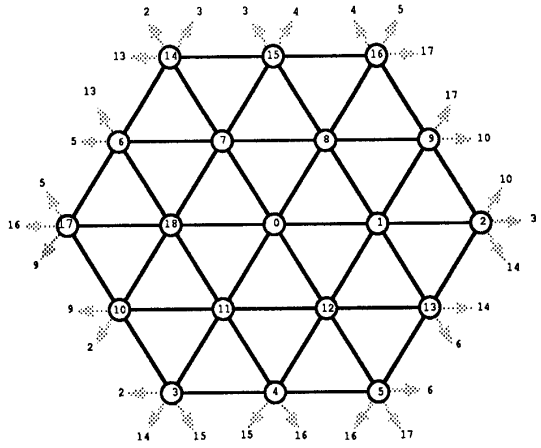

Figure 1: A hexagonal mesh of dimension 3.

2 Description of HARTS

The interconnection topology used in HARTS is a C-wrapped[1] hexagonal mesh (H–mesh). Each node in an H–mesh is connected to six neighboring nodes. As the solid lines of Fig. 1 shows, however, the peripheral nodes of a non-wrapped H–mesh are connected to only three or four neighbors rather than six. The C-wrapping used in HARTS is a means of connecting every node to six other nodes to create a homogeneous network [1]. Any node can thus be viewed as the "center" of the network. Moreover, a transparent addressing scheme can be developed for any size H–mesh such that the shortest paths between any two nodes can be computed with a $\Theta(1)$ algorithm given the addresses of the two nodes [1]. This addressing scheme also makes possible a simple message routing algorithm that can be efficiently implemented in hardware [3].

The *dimension* of an H–mesh is defined as the number of nodes on its peripheral edge. Fig. 1 shows a C-wrapped H–mesh of dimension three, with gray arrows indicating the extra connections between the peripheral nodes. A C-wrapped H–mesh of dimension n is comprised of $p = 3n^2 - 3n + 1$ nodes, labeled 0 to $3n^2 - 3n$, where each node s has six neighbors labeled $[s+1]_p$, $[s+3n-1]_p$, $[s+3n-2]_p$, $[s+3n(n-1)]_p$, $[s+3n^2-6n+2]_p$, and $[s+3n^2-6n+3]_p$, where $p = 3n^2 - 3n + 1$ and $[a]_b$ denotes $a \bmod b$ [1].

One can better visualize what is happening in the C-wrapping by first partitioning the nodes of a non-wrapped H–mesh into rows in three different directions. The mesh can be viewed as composed of $2n-1$ horizontal rows, $2n-1$ rows in the 60-degree counter-clockwise direction, or $2n-1$ rows in the 120-degree counter-clockwise direction. In each of these partitions we label from the top the rows R_0 through R_{2n-2}. The C-wrapping is then performed by connecting the last node in R_i to the first node in $R_{[i+n+1]_{2n-1}}$ for each i in each of the three partitions [1].

The version of HARTS presently under construction at the RTCL is of dimension three and composed of 19 nodes. Each node of HARTS consists of from one to three applications processors (AP's) (thus permitting multiprocessor nodes) and a custom–designed network processor (NP) for handling inter-process communications. The nodes currently used in HARTS are VME-bus systems with 68020-based AP's. The NP's front-end is a custom VLSI routing controller designed to manage the six pairs of half–duplex communications links and route messages between nodes. The routing controller provides support for routing based on packet switching, circuit switching, and *virtual circuit cut-through* [6], where messages are not buffered at intermediate nodes if a circuit to the next node in the message route can be established. The design of the routing controller is detailed in [3]. Also part of the NP will be a buffer management unit to buffer messages and a general-purpose processor to perform various functions of HARTOS, an operating system being developed for HARTS [5].

HARTOS is built on top of a real-time kernel called pSOS (by Software Components Group. Inc.), which implements a multi-tasking environment with message exchanges and events for communications. HARTOS extends the uniprocessor pSOS to provide similar communication facilities over the network.

3 Critiques of Candidate Solutions

Before proposing a viable solution, it is important to explore and evaluate other candidate solutions to see what problems are presented in trying to develop a distributed I/O subsystem with high accessibility and performance together with low cost and complexity. These alternate solutions represent the evolution by which we shall arrive at our solution, showing the rationale for our decisions and why we did not choose certain more obvious solutions. We shall touch on some of these points in the discussion of the proposed multi-ownership solution and explore them in more detail.

[1] 'C' stands for the word 'continuous'.

3.1 "Node-Direct" Distribution of I/O

Since centralizing I/O results in performance and accessibility bottlenecks, I/O devices must be physically distributed. One way of doing this is to connect sets of I/O devices directly to computation nodes. In a system such as HARTS which uses a dedicated processor to handle inter-process communications (IPC), there are two principal methods for connecting the I/O devices to the node: via the intra-node bus or a separate I/O bus connected to either an AP or the NP.

A. Intra–Node Bus

If I/O devices are connected to the intra-node bus via a suitable I/O controller (IOC), there remains the question of which node will control and administer the devices. Logically, if the NP is the "node master" (as is the case in HARTS, since it manages all communication, both intra-node in the case of a multiprocessor node and inter-node communication in all cases), then it would also administer the I/O devices. It would then be natural for all AP's, whether residing in the given node or not, to use the existing IPC methods for communicating with I/O devices. However, if the NP is not powerful enough to take on administration of the I/O devices, then an AP can also perform such tasks. An AP used as the I/O administrator can be dedicated to this task or may perform these functions in addition to its other duties.

This is one of simplest designs for the I/O subsystem since all that is required is a suitable interface card for the intra-node bus used (VME in the case of HARTS). This gives all processors in the home node easy access to the I/O devices, although access will probably be controlled by a single I/O master—the NP or an AP. The accessibility of the I/O devices depends on the correct operation of the I/O master. If an AP serves as I/O master in addition to acting as a general AP, it would be possible for another AP in the (multiprocessor) node to take over should it fail.

The two major disadvantages of this method are obvious: poor accessibility and potentially poor performance. If other nodes need to access the I/O devices belonging to a given node, the I/O transactions must pass through the NP at the home node. Should this NP fail, all access to the I/O devices would be lost. It may be possible to replicate the I/O devices in question, but this would increase the total network and intra-node overhead (now for more than one node) required to access the device. Even without the overhead introduced by replication, I/O traffic, regardless of its destination, will always pass over the home node's intra-node bus, possibly penalizing the home node even for remote I/O transactions.

Regardless of whether or not the consumption of VME-bus bandwidth by I/O is acceptable, it is certainly not in keeping with the philosophy of HARTS communications, in which traffic not bound for a node should not penalize the performance of a node and in fact, need not even be buffered there temporarily, i.e., virtual cut–through.

B. Local I/O Bus

Connecting I/O devices to the intra-node bus increases the traffic on this bus, perhaps beyond its bandwidth. Another solution is to use a separate, perhaps simpler, I/O bus, connected to either the NP at the node or an AP. This time it makes more sense to connect the I/O bus to the NP; otherwise, I/O traffic bound for other nodes or AP's must still travel through the intra-node bus, thus defeating our purpose.

Assuming the NP can handle the I/O service and control overhead in addition to its primary functions, this method is approximately as complex and cost-effective as the former. The dedicated I/O bus need not be as versatile as the intra-node bus, so the interfacing requirements may in fact be simpler.

The advantage of this method is clearly that the only I/O traffic which must travel over the local intra-bus is traffic destined for a processor at the node itself. In a sense we are just trading processor bandwidth for bus bandwidth, but this is a justifiable tradeoff in that the job of the NP is likely to be better specified than that of the rest of the node. We will always have the problem of characterizing the NP traffic and workload regardless of whether it handles only I/O communications or I/O communications and rudimentary administration. This method allows us to eliminate the variable of non-local I/O traffic from the intra-node bus.

Finally, this solution opens up the possibility of connecting these I/O buses together to form a separate I/O network, allowing direct access to non-local I/O devices without having to use the regular IPC channels, a prospect we will be looking at in Section 3.3.

3.2 I/O Nodes

A typical node in a distributed computing system may be thought of as a computation node if its processors perform strictly computational tasks as opposed to I/O tasks, e.g., I/O device drivers. Similarly, a node which serves only to connect I/O devices to the computation nodes in the network would be referred to as an *I/O node*. Thus in the given topology of the node interconnection network, some nodes can be made to provide strictly I/O services while others provide computational services. All I/O traffic uses the IPC

channels provided by the network.

The I/O devices at a given I/O node can be interfaced to the intra-node bus and serviced by the NP. If sufficiently powerful processors were used as I/O controllers, the operating system interface between the NP and the I/O controllers could be the same as that between an NP and the AP's in an ordinary node (e.g., send and reply mailboxes) [5]. I/O–process communication (IOPC) could be handled exactly like inter-computation-process communication (ICPC), making the operation of the specialized I/O node and I/O controllers completely transparent to the rest of the network.

However, this is one of the most complex and expensive solutions because an NP, including routing controller and associated hardware, must be dedicated to a group of I/O devices. Decreasing the cost per I/O device by increasing the number of I/O devices per I/O node begins to defeat the purpose of distributing the I/O access in the first place. Also, *all* access to I/O devices must use the same network/protocol as ICPC, which may or may not be a good idea depending on the nature of IOPC traffic vs. ICPC traffic. Special care may have to be taken to ensure adequate bandwidth for I/O bursts as well as the timely delivery of critical inter-process messages. If both IOPC and ICPC traffic are heavy, it may not be economical to build enough bandwidth into the one network to handle both traffic streams.

Access to the I/O devices at the I/O node depends on the correct operation of the NP, a device which may well be more complex than the average AP. However, the node-direct designs also suffer from this vulnerability, assuming that access to the I/O device(s) connected to a node is required outside the node (if this is never the case then the flexibility offered by the I/O node design is probably not needed).

All IOPC with the devices connected to a given I/O node must take place over the H–mesh interconnection network; however, no AP is saddled with service/control overhead for the I/O devices and the same processor which controls and services peripherals can also format data, handle the results of multiple sensors, and so on.

This approach also requires that load distribution algorithms be part of any application running on such a system and that the application be flexible enough not to depend on a particular number of AP's. The effect is similar to that of a network in which several applications processors have failed, where the I/O nodes can be visualized as the "holes" in the computation network. Applications which are intended to survive node failures will have to take this effect into account anyway, so this is not a drawback for serious fault-tolerant applications.

3.3 I/O Network

All of the alternate methods presented thus far require the use of the standard IPC channels to send I/O information to processors on remote nodes. Furthermore, they are also dependent on the correct operation of the NP at the node administering a given I/O device. If the NP should fail, access to the devices administered at the node in question would be impossible. Thus to increase the accessibility of the I/O devices and the reliability of the system as a whole, we could give the I/O devices (relatively) simple control processors which are in turn connected via a separate I/O network to each other and to the nodes. This method is likely to be the most expensive since it requires a completely separate network and I/O control processors in addition to the applications and network processors already in place. However, it has potentially the highest accessibility and performance because it can support multiple I/O transactions in parallel as well as providing more than one way of accessing given devices.

The issue of what network topology to use for this separate I/O network is somewhat problematic. If we want to provide direct contact between each of the computation nodes and the I/O network, then we need some way of mapping the I/O network onto the H–mesh. An important problem is the fact that while most interconnection topologies which have been explored involve an even number of nodes or a number of nodes which is a power of two, an H–mesh will always have an odd number of nodes, and for small-diameter meshes it tends to have a prime number of nodes.

One obvious choice is to use a completely separate H–mesh mesh for the I/O network. One way to do this would be to give each computation node two NP's, one for the computation network and one for the I/O network. Although this solves the mapping problem, it is an expensive solution because it would require an additional p NP's to manage the communication in the I/O network where p is the total number of nodes in the H–mesh. Moreover, access to I/O devices still depends on the correct operation of a complex device, the NP. There are some interesting advantages to having essentially two complete H–meshes. IOPC and ICPC could be completely separated, possibly simplifying bandwidth and message scheduling and delivery issues. Taking a different approach, it would also be possible to "borrow" one network to deliver messages for the other. Thus if a computation–NP decided that it could not send an inter-process message in time, it could ask the I/O–NP if it could send the message before its deadline on the I/O network, possibly allowing a greater number of messages

to be delivered on time. The same applies to delivering messages at all—if one NP ascertained that its messages to a particular node were not getting through, it could forward them to the other NP to see if it could successfully deliver the messages on its network.

Despite the advantages of a separate I/O network, we have not adopted it due mainly to the mapping problem on the one hand and the expense of a completely separate mesh on the other.

4 Non-Distributed I/O

Obviously, I/O does not have to be distributed in a distributed computer system; nodes can simply be connected to some central I/O handling facility. This may involve using a single node as an I/O center, with I/O traffic using the standard IPC channels or some single entity acting as the I/O center with connections to all of the computation nodes (e.g., the original NCUBE design and other early hypercubes). Although this may be the simplest way of handling I/O distribution, accessibility and performance problems will make it unusable in all but a few circumstances. The rationale for using a distributed computer system for a particular application is to obtain a desired level of performance and/or accessibility which could not be obtained (at least not as cost-effectively) with a uniprocessor or multiprocessor architecture. Thus by centralizing I/O access, performance suffers because the I/O center becomes a bottleneck, and accessibility suffers because the system is susceptible to single-point failures.

We will next present our proposed I/O subsystem design which we believe to provide the best accessibility and performance at the lowest cost and complexity.

5 Multi-Owner I/O Devices

To avoid the accessibility problems of non-distributed I/O, we would like I/O devices to be managed or "owned" by relatively simple, and reliable, controllers. Moreover, to improve both accessibility and performance, we want multiple access paths to these I/O devices.

The desire for simple I/O controllers presents a problem in HARTS, because the natural tendency would be to have I/O devices belong to individual nodes or network processors, both relatively complex and expensive devices, and use the given IPC channels in HARTS to handle the I/O traffic. We can still use the given IPC channels, but instead of permanently tying down a given I/O device to one node, we will allow several nodes to communicate with each I/O device. There are at least two fundamentally different protocols for managing this communication, but we will explain the architectural considerations first and then discuss the protocols.

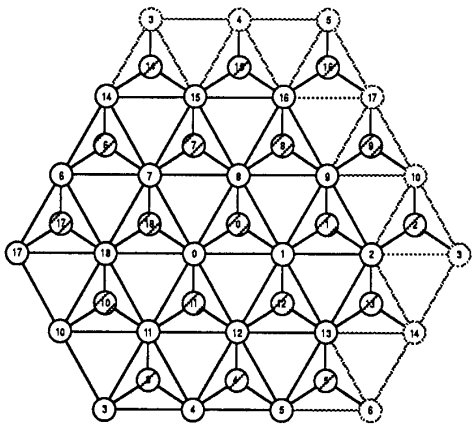

Figure 2: I/O controller placement.

5.1 Interconnection Architecture

We will cluster I/O devices together and give them a controller to manage access to the devices. However, the controller can be made simple because we will be using simple data links to the HARTS nodes (presumably serial since the standard inter-node links in HARTS are serial, but this is not required). The I/O controller need only be able to handle sending and receiving simple messages via a set of full-duplex links, not providing virtual cut-through capabilities and other features of a full-blown NP. To keep the number of I/O controllers and the number of I/O links down to a reasonable number, we will restrict the number of I/O controllers (IOC's) to be no greater than the number of computation nodes in the mesh, p. This will have certain benefits for one of the management protocols explained later.

Now that we have established the potential number of I/O stations, we need to decide how many nodes each IOC will be connected to. If we assume the maximum number of IOC's in an H_3, for example, then Fig. 2 suggests a logical connection scheme. Each IOC can be thought of as being in the center of one of the upward-pointing triangles created by this representation of the hexagonal mesh interconnections and the IOC is then connected to each of the nodes which

make up this triangle. This gives three possible avenues of access to each IOC. Note that if the maximum number of IOC's are used, the number of I/O links required will be equal to the number of standard communication links, or $9n^2 - 9n + 3$ for an H_n. There is no particular reason that one could not similarly place IOC's at the (logical) center of the downward-pointing triangles as well, allowing for up to $2p$ IOC's, but this will double the maximum possible number of I/O links required and will disturb certain homogeneous effects of limiting the number of IOC's to the number of nodes, as we will see shortly.

5.2 Management Protocol

The first management protocol we will look at assigns one node to each IOC as its owner, but with the important provision that the owner can be changed if the original owner becomes faulty. In this protocol one of the IOC links is defined to be the primary or active link and the rest remain inactive as spares. The second protocol allows the IOC owner to be defined dynamically, allowing for greater accessibility in some cases and fewer average hops required to reach the IOC owner. In this protocol the IOC decides which link will be active at any given time.

Let us call the three nodes to which an IOC is connected its "I/O partners." We will number these nodes 0, 1, and 2, where 0 is the "left partner," 1 is the "upper partner," and 2 is the "right partner" (see Fig. 3). For the purpose of our discussion, we will label each IOC with the node number of its left partner. This only a notational convenience for explaining the ownership protocols; IOC's themselves have no real identity as far as the system in general is concerned. The relevant labels are those of the I/O partners (for routing the I/O messages within the network) and those of the I/O devices themselves (so that they can be located in the network and their I/O partners identified).

5.2.1 Static Ownership

Under this protocol each IOC is initially assigned an owner node through which all I/O traffic will pass until such a time as the the owner fails or becomes unreachable. In this case a new owner is chosen, which then retains ownership until it fails or can longer reach the IOC. To access a particular I/O device, a process broadcasts an I/O inquiry on the network, similar to the process for finding a message exchange in HARTOS [5]. This inquiry contains the system name of the desired I/O device. The owner of the IOC which controls the I/O device will find the device name in its I/O name server and respond to the I/O inquiry. The process desiring

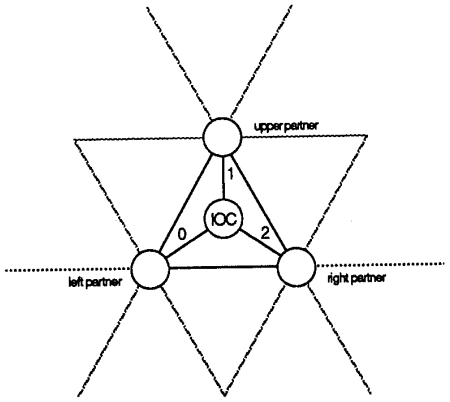

Figure 3: I/O controller partners.

I/O service will then send its requests to that particular node.

Upon boot-up, each IOC sends a message to its default owner node, partner 0, to notify the node that it owns the IOC. This message contains information regarding the I/O devices this controller has at its disposal, their system names, boot-up status, etc. During its operation, the IOC periodically sends a test message to its owner to ascertain the owner's status. If it fails to obtain a satisfactory response from its owner node (e.g., time-out), the IOC will attempt to find itself a new owner. When a suitable partner has been found, the IOC will declare this partner its new owner and send the I/O device information to it.

The owner and the IOC also agree in advance on the frequency at which the IOC will send the test message to the owner. The owner can then use this information to detect missing test messages and attempt to communicate with the IOC to see if the link is still operational. If it cannot establish a satisfactory dialogue with the IOC, it will resign as owner and go about its regular business. It is then up to the IOC to recognize the situation on its end as just explained (which it will, assuming it is operating correctly) and find itself a new owner. Furthermore, each time the owner receives a test message from the IOC or its watchdog timer wakes it up to check the IOC, the owner will evaluate the status of its mesh links. If it finds that none of its mesh links are functioning, it will send a resignation message to the IOC informing it that it is resigning as the IOC owner and that it should choose a new owner. If it cannot communicate this information, then the IOC will have already detected the unreachability of the owner and

will choose a new owner on its own. In either case the owner resigns.

Both of these schemes are required to handle the ownership issue correctly. If only the first method were used, where the IOC assumes all the responsibility of checking the link, then we could have the new owner send a message to the old owner to inform it that it is no longer the owner. If it could not send this message, it means only that the old owner is not reachable by the new owner. The old owner could still be reachable by other nodes requesting I/O service and there would be no way to prevent the old owner from responding to these requests and trying to service them. If only the second method were used, then an owner which found itself unable to communicate successfully with its IOC could try to inform the I/O partners that a new owner needed to be chosen, but it may be unable to communicate with them due to mesh link failures. Thus the IOC is given the task of finding itself a new owner if the old owner becomes unreachable since only if the IOC itself fails or all three partners have failed is it impossible to find a new owner. The owner is given the responsibility of revoking its ownership status if it finds itself unable to reach its IOC.

If a process requesting I/O service does not obtain a response to its inquiry, then either there is no owner for the IOC (which can be because the IOC itself is faulty, because all links between the IOC and its partner nodes have failed, or because all of its partners are faulty) or the owner of the IOC is unreachable. In either case, the I/O device in question is unreachable and the situation should be handled in the same manner regardless of the cause—no node will reply to the I/O service inquiry and the process must execute whatever contingency plan it has. (An obvious plan is to retry one or more times before bringing in more expensive recovery methods.)

We still need a method of finding a new owner for the IOC when it determines that its old owner is no longer reachable. The IOC can maintain the status of its three links/partners (the IOC has no way of distinguishing between a link failure and a partner node failure). When looking for a new owner, it polls each of the remaining intact partners, asking them how many other IOC's they each own and how many of their mesh links are still intact, information which a computation node can and should maintain. It will then choose the partner with the greatest number of functional links, to increase the chances of other nodes reaching the owner, and if each of two remaining partners have the same number of operating links, it will choose the partner which owns the fewest IOC's. If no distinction can be made on this basis, it will choose a new owner at random.

This management protocol is desirable for a number of reasons. First, it is simple to implement and efficient because the only overhead involved with establishing a partner's right to access to the IOC occurs when a new owner is being chosen. Second, for certain I/O devices it is desirable to perform some level of the I/O device management in some predetermined node. For a disk, for example, this might involve maintaining some level of the file system on the owner node. Otherwise, the IOC might have to be made more complex to handle this function. But under this management protocol it is possible for the owner of an IOC to be unreachable to a particular process desiring I/O service while there is another partner of the IOC which is reachable by that process. Thus if this other partner were the owner instead of the current owner, the process in question would be able to obtain I/O service. In Fig. 4 we have an example where a process in node 13 wants service from IOC 18, but since node 18 is the owner and is not reachable from 13, it cannot obtain service. If node 0 were the owner instead of 18, it could obtain service. Also one IOC partner may be closer to the node requesting service than the current owner. If ownership could be determined per request instead of remaining in effect until the owner is forced to resign due to failures, we could provide faster average service. These two factors make up the rationale for the next management protocol using dynamic ownership.

5.2.2 Dynamic Ownership

Under this protocol the owner of an IOC is determined on a per–request basis. A process desiring I/O service will send its request to the nearest partner of the IOC it wants to access. This partner will then petition the IOC for access, which the IOC will grant as soon as it is free. To access a particular I/O device, a process broadcasts an I/O inquiry on the network. This inquiry contains the system name of the desired I/O device. Each partner of the IOC which controls the I/O device in question will find the device name in its I/O name server and respond to the I/O inquiry. The process desiring I/O service will then collect the responses to its inquiry and after a certain time-out period it will compute the closest partner and sends its I/O request to this node. The time-out period is necessary because the process requesting service has no way of knowing how many partners are operational or can currently reach the IOC. This may result in a request being to a node which is not the closest if the closest node does not respond within the time-out period.

Upon boot-up, each IOC will send a message to each of its partner nodes. This message contains information regarding the I/O devices this controller has at its disposal,

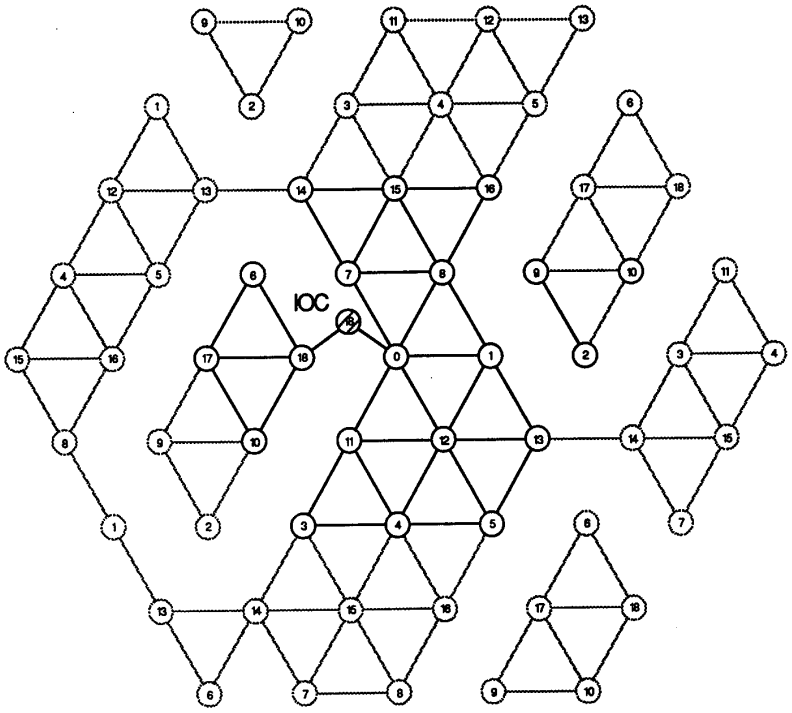

Figure 4: Unreachable static owner.

their system names, boot-up status, etc. During operation each IOC will monitor its links for access requests coming from its partners. When it receives a valid request, it will declare the sending partner to be the current "owner" and allow it to send an I/O request. While it is servicing the request it will send back deferment messages to any other partners requesting access and place these requests on its deferred queue. Upon receipt of a deferment message, a partner requesting I/O access will refrain from sending any more requests and wait until it has been granted access. When it has finished servicing the current request, the IOC will grant access to the next partner on its deferred queue.

In addition to leaving IOC's accessible where static ownership would make them inaccessible, this protocol also takes into account the fact that one partner may be closer to a node requesting service than the other partner. Since this protocol chooses the closest of the partners that respond, the I/O traffic may have fewer hops to travel. However, its disadvantages are that it is more difficult to implement, it involves arbitration overhead after each I/O request has been serviced, and it may be undesirable because there is no single node through which all I/O requests will travel and which could perform some I/O management tasks.

5.3 Simulation of Static vs. Dynamic Ownership

In order to determine how much of an advantage dynamic ownership offers over static ownership, a simulator was written to evaluate the average accessibility of IOC's and the average number of hops required to reach the owner of an IOC from each node in an H_3. This simulator is similar in function to the one used in [8] for evaluating a fault-tolerant message routing algorithm for HARTS. It tests the effects of link failures alone and the combined effect of link and node failures on accessibility and shortest-path distance to IOC owners.

Mesh links (links between nodes) and IOC links (links from IOC's to their partners) are assumed to fail with equal probability. This is a reasonable assumption since the phys-

ical implementation of these two types of links is likely to be similar. Moreover, the failure of a link of either type is assumed to be as likely as the failure of any other link of that type. In the simulations that we ran, we also assumed that nodes are always more likely to fail than links of any type, because processors are more complex than links. The ratio of the probability of node failure to the probability of link failure is a parameter in the simulation.

The simulator works by selecting a random set of IOC links, node links, and processor nodes to declare faulty and then testing to see if a path exists between each node and each IOC owner, if one exists. Note that in the case of the dynamic ownership protocol, an IOC owner is a partner with an intact IOC link. If a path does exist, this fact and the length of the path (measured in "hops," the number of links which must be traversed) are noted. This process is repeated a sufficient number of times to assure consistent results (between 0.7 and 1.2 million) for each total number of failures in the system.

Initially, only the effect of link failures was simulated. Here, numbers of faulty links between 0 and 85 were simulated for both static and dynamic ownership. The simulator computes the average number of hops a message must travel from source to destination (IOC owner, not the IOC itself) and the number and percentage of messages which were and were not deliverable.

The effects of combined node and link failures are then simulated. Again, the total number of failures is varied, this time between five and 85 in increments of five, and the relative probability of node vs. link failures is used to declare this number of components faulty. The same statistics are measured as for the case where only link failures were tested.

Although the dynamic ownership can indeed improve the reachability of I/O devices, the improvement is not significant—no more than 3% better than static ownership in the best case. Naturally, with relatively few link failures, the difference is negligible. Thus dynamic ownership may not be worth implementing for this reason, especially if I/O management can be simplified by using static ownership instead. Moreover, dynamic ownership both places greater demands on the IOC and results in somewhat higher protocol (arbitration) overhead.

When both node and link failures are taken into account, the average improvement in reachability achieved by dynamic ownership is even less. However, it should be noted that when considering the combined effect of node and link failures, the net effect on the system of a 50% component failure rate is much more deleterious than in the case where only link failures are considered, so the apparent loss of improvement in the dynamic case may make little difference in practice because so much of the system's processing power has been lost.

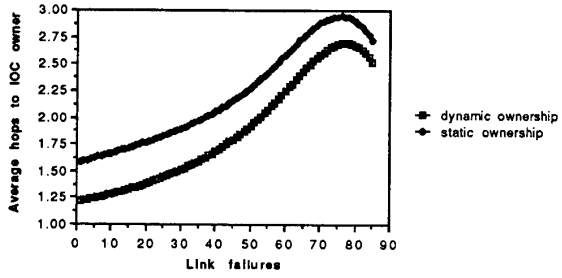

Figure 5: Average hops to reach IOC owner vs. link faults.

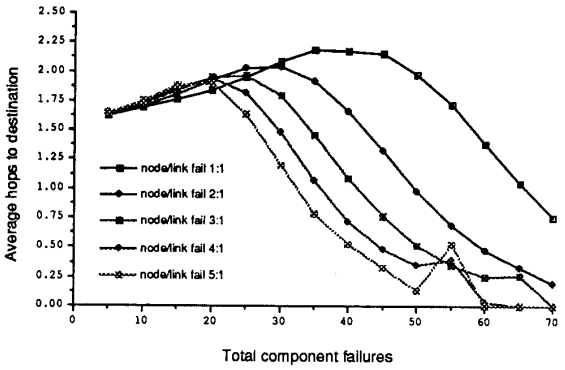

Figure 6: Static ownership: Average hops to reach IOC owner vs. system faults.

However, Fig. 5 shows that in all cases we can substantially reduce the average shortest-path length by using dynamic ownership rather than static. With as many as 50% of the links faulty, the improvement is still at least 0.35 hops shorter average path length. If I/O traffic is expected to be high, especially with large transfers, then dynamic ownership may well be worthwhile in order to improve I/O performance, with the additional benefit of a slight improvement in I/O accessibility.

From Figs. 6 and 7 we can see that although dynamic ownership always results in a shorter average path length, the difference grows less and less as the probability of node failure increases, as mentioned above for deliverability.

These are the relative differences in performance between the two protocols; the simulations also show that even with half of the links faulty, 85% of I/O traffic can still be routed.

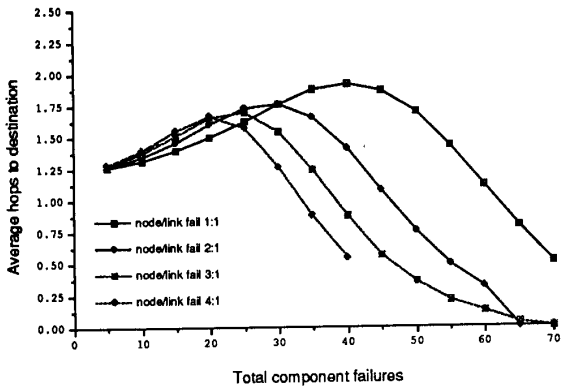

Figure 7: Dynamic ownership: Average hops to reach IOC owner vs. system faults.

In the case where both node and link failures are considered, this number drops dramatically as the ratio of node to link failures increases, again due to the very serious effects of so many processor failures.

This is, of course, assuming a routing algorithm which can always find the shortest path, even in a faulty mesh. The simple algorithm proposed in [8] is not perfect, but even under 50% link failures, it fails to deliver only less than 10% of actually deliverable messages.

5.4 IOC Architecture

We have assumed that the connection between the IOC's and their partner nodes is the same as that used to connect the processor nodes themselves. In HARTS this implies the use of high-speed full-duplex serial lines. Other mechanisms are certainly possible, but given that there may well be good reasons for physically isolating IOC's from the processor nodes, serial links will offer the most flexibility.

An IOC must therefore provide three serial links as its interface to the partner nodes as well as some interface to whatever I/O devices it actually services. This could be SCSI or ESDI in the case of mass-storage devices, IEEE-488 or RS-232/422 in the case of instrumentation, or custom/proprietary interfaces or buses as needed.

Two problems must be addressed in an effective IOC design: general applicability of the IOC and the communication protocol used on the I/O links to the partner nodes. One way to address both of these concerns without necessarily limiting the actual I/O device interface would be to implement a time-division multiplexed (TDM) serial protocol and allow several I/O processors to use the I/O links in

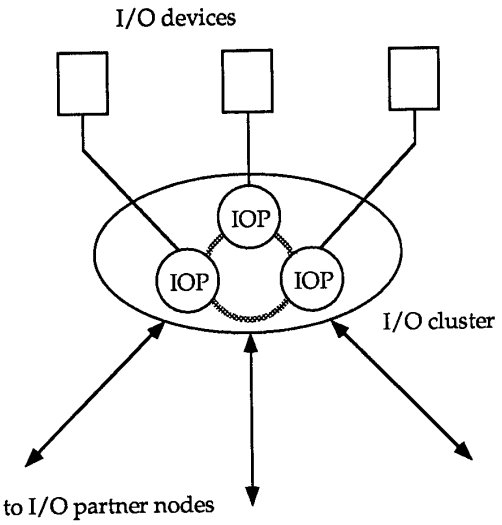

Figure 8: I/O cluster architecture.

turn. This provides greater flexibility because the only standard to which an I/O processor need conform is that of the TDM protocol—it is free to use whatever interface is most appropriate for the I/O devices for which it is responsible. It may also offer an arbitrary level of services as well, ranging from simple one-way data transmission in the case of a sensor interface, to file-system support in the case of a disk interface. It also permits a wide variety of I/O processors to be used.

Today's high-speed microprocessors are more than capable of implementing this style of IOC design with sustained transfer rates well in excess of 5 megabits/second. Keeping in mind that the total bandwidth of the I/O links is shared among some number of I/O processors residing at a cluster, this should be adequate. If a high-performance device requires greater bandwidth, it would be possible for it to be allocated more than one time-slot in the TDM protocol, provided the I/O processor could keep up.

This approach has the effect of turning an "I/O controller" into an "I/O cluster" consisting of one to n processors, where n is determined by the exact nature of the TDM protocol used. Fig. 8 illustrates this design. This in turn offers a degree of scalability to the IOC concept. Instead of forcing all I/O traffic at the level of a given IOC to pass through a single processor, multiple I/O processors can easily be used to support devices which require more processor attention. Moreover, this further increases the reliability of the I/O cluster. If one I/O processor should fail, there would

simply be a wasted time-slot in the transport protocol (assuming that the processor had not failed in such a manner as to disrupt the protocol). Other processors would be able to continue operation.

Thus not only is there flexibility in the number of IOC's used in a particular system, there is flexibility in the number of the I/O processors in a given I/O cluster. This flexibility and the multiple connectivity of the IOC's permit an I/O system with a high degree of performance, accessibility, and scalability.

6 Conclusion

We have examined a variety of solutions for implementing a distributed I/O subsystem for HARTS: node-direct connection, where nodes own sets of I/O devices, dedicated I/O nodes, separate I/O network, and multi-owner I/O controllers. The multi-owner method is judged to be the best solution in terms of cost and complexity, accessibility, and performance. It is scalable with the hexagonal mesh itself and allows simultaneous access to I/O controllers. It has many of the accessibility benefits of the separate I/O network while using a simple, less expensive interconnection scheme. We have developed two different management protocols, static ownership and dynamic ownership, which offer different advantages and options. Static ownership is simpler and allows certain I/O device management processes to reside on an owner node. Dynamic ownership makes nodes accessible in instances where they would not be accessible under static ownership, and can also reduce the distance which I/O traffic must travel.

Acknowledgement

The work reported in this paper was supported in part by the Office of Naval Research (ONR) under Contract No. N00014–85–0122. 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 ONR.

The authors would like to thank Alan Olson and Jim Dolter of the Real-Time Computing Laboratory, The University of Michigan for the preparation of figures in this paper, and Andre van Tilborg and Gary Koob of the Office of Naval Research for financial support of the work reported in this paper.

References

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

[2] A. Davis, R. Hodgson, B. Schediwy, and K. Stevens, "Mayfly system hardware," Technical Report, HPL–SAL–89–23, Hewlett–Packard Corp., Apr. 1989.

[3] J. W. Dolter, P. Ramanathan, and K. G. Shin, "Microprogrammable VLSI routing controller for HARTS," *Proc. ICCD'89*, pp. 160–163.

[4] J. W. Dolter, P. Ramanathan, and K. G. Shin, "Performance analysis of message passing in HARTS: A hexagonal mesh multicomputer," *IEEE Trans. on Comput.* (in press).

[5] D. D. Kandlur, D. L. Kiskis, and K. G. Shin, "HARTOS: A distributed real-time operating system," *ACM Operating Systems Review*, vol. 23, no. 3, pp. 72–89, Jul. 1989.

[6] P. Kermani and L. Kleinrock, "Virtual cut–through: A new computer communication technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

[7] J. H. Lala, A. Ray, R. Harper, and D. B. Mulcare, "A Fault and Damage Tolerant Network for an Advanced Transport Aircraft," *Proc. of American Automatic Control Conf.*, pp. 1138–1142, June 1984.

[8] A. Olson and K. G. Shin, "Message routing in HARTS with faulty components," *Digest of Papers, FTCS-19*, pp. 331-338, Jun. 1989.

[9] J. Salmon, "Cubix: An I/O system for the hypercube," Caltech Concurrent Computation Project, Technical Report 293, Jul. 1986.

[10] K. G. Shin, C. M. Krishna, and Y.–H. Lee, "A unified method for evaluating real-time computer controllers and its application," *IEEE Trans. on Automatic Control*, vol. AC–30, no. 4, pp. 357–366, Apr. 1985.

[11] A. Witkowski, K. Chandrakumar, and G. Macchio, "Concurrent I/O system for the hypercube multiprocessor," *Proc. Third Conference on Hypercube Concurrent Computers and Applications*, pp. 1398–1407, Jan. 1988.