A NEW ARCHITECTURE FOR DISTRIBUTED SENSOR INTEGRATION

D. Nadig and S. S. Iyengar (*)

Department of Computer Science
Louisiana State University
Baton Rouge, LA 70808

D. N. Jayasimha (**)

Department of Computer and Information Sciences
Ohio State University
Columbus, OH 43210

Abstract: The *computational* issues related to information integration in multisensor systems and distributed sensor networks has become an active area of research. From a computational viewpoint, the efficient extraction of information from noisy and faulty signals emanating from many sensors requires the solution of problems related a) to the architecture and fault tolerance of the distributed sensor network, b) to the proper synchronization of sensor signals, and c) to the integration of information to keep the communication and the centralized processing requirements small. In this paper, we propose a versatile architecture for a distributed sensor network which consists of a multilevel network with the nodes (processing element/sensor pairs) at each level interconnected as a de Bruijn network. We show that this multilevel network has reasonable fault tolerance, admits simple and decentralized routing, and offers easy extensibility.

We model information from sensors as real valued intervals and state an interesting property related to information integration in the presence of faults. Using this property, the search for a fault is narrowed down to two potentially faulty sensors or communication links. In a distributed environment, information has to be integrated from "temporally close" signals in the presence of imperfect clocks in a distributed environment. We apply the results of past research in this area to state various relationships between the clocks of the processing elements in the network for proper information integration.

Keywords and Phrases: Distributed Sensor Networks, De Bruijn Networks, Information Integration, Abstract Estimate, Clock Synchronization, Fault Tolerance.

1.0 INTRODUCTION

In recent years there has been increasing interest in the development of distributed sensor networks (DSNs) for information gathering. This is partly because of the availability of new technology which makes the DSNs economically feasible to implement and the increasing complexity of today's information gathering tasks to which they are applied. These tasks are usually time-critical and rely on the reliable delivery of accurate information for their completion. To meet these requirements, a DSN must be able to dynamically respond to fault conditions, reconfiguring its activities as necessary to compensate for disturbances. Thus, the search for efficient, fault-tolerant architectures for DSNs has become an important area in research. A DSN consists of a set of sensors, a set of *p*rocessing *e*lements (PEs), and a communication network interconnecting the various PEs. One or more sensors is associated with each PE. We refer to the PE and its associated sensor(s) as a *node*.

The integration of multiple, disparate sensors into a useful sensor network involves the solution of several different problems. For an excellent discussion of the problems and the current state of the art in multisensor integration, the reader is referred to the survey paper by Luo and Kay [9]. From a computational viewpoint, however, the efficient extraction of information from noisy and possibly faulty signals emanating from many sensors requires the solution of problems relating a) to the architecture and the fault tolerance of the distributed sensor network, b) to the proper synchronization of sensor signals, and c) to the integration of information to keep the communication and the processing requirements small.

Wesson et al. [2] were the first to attempt designing efficient networks for distributed sensing. They proposed the hierarchical and committee interconnection topologies. A sensor network based on a fixed number of complete binary trees fully interconnected at their roots (we will refer to this network as a flat tree network) was considered in [11, 12] and the following issues were studied:

(1) the integration of information in real time when clocks at the nodes are not perfect,

(2) the transmission of information without incurring heavy communication costs, and

(3) the fault tolerance of the network to certain types of faults.

In this paper, which is a continuation of research reported in [11, 12], we propose a new versatile architecture which has several advantages over the flat tree network. Specifically, the proposed network has better fault-tolerant properties and supports more nodes than the latter with the same diameter. We show how information integration could be achieved in this network and state an interesting property related to such integration in the presence of faults.

This paper is organized as follows. Section 1.1 has a brief overview of sensor integration. The notations and definitions used in the paper are presented in Section 1.2. After motivatating the need for a new sensor network in Section 2.1, we propose a multi-level network with each level having the de Bruijn interconnection in Section 2.2. Algorithms for routing in this network are described in Section 2.3. We describe sensor integration in the presence of faults in Section 2.4. The fault tolerant properties of the network are the subject of Section 2.5. In a DSN, it is necessary that the clocks on the nodes be synchronized. A variant of a previously known method for synchronizing clocks is described for the network in Section 3.0. We conclude the paper by highlighting the features of the proposed network and indicate the future directions this area of research could possibly take.

1.1 An Overview of Sensor Integration

The PEs of a DSN combine the sensor output readings to derive an accurate value of the physical process that the sensors monitor. This process of combining the sensor outputs is called *information integration* or *data fusion*.

The method used to integrate the information passed by the sensors depends on whether the sensors provide competitive information or complementary information. In the former case, each sensor ideally provides identical information. This redundancy of the sensor readings helps in enhancing the reliability and fault tolerance of the network. Also, noise in the signals can be detected and removed. This is because the noise in different sensor signals tend to be uncorrelated while the signals of interest are correlated. It is therefore necessary for the information from the sensors to be combined in a meaningful and effective manner, so that the result is fairly accurate. Complementary information integration is done when only partial information is available from each sensor; such information is then integrated to obtain the result.

Following Marzullo [10], we distinguish between a *concrete sensor* and an *abstract sensor*. A concrete sensor is a device that can be used to sample a physical state variable. An abstract sensor is a piecewise continuous function from a physical state variable to a dense interval of real numbers. The reasons for using an abstract sensor rather than a concrete sensor are detailed in [10, 11]. Determining the function which maps a concrete sensor to an abstract sensor depends on many factors such as the choice of a particular sensor type (e.g., motion detecting sensor, range finding sensor, vision sensor), the compensation that has to be applied to the raw sensor value which is itself dependent on the local values of certain parameters (e.g., design parameters of the sensor), the nature of the application, etc. For instance, if a sensor reads a value to be S and its maximum error is known to be E, then an abstract sensor, albeit simple, could be the interval $(S-E, S+E)$. A PE at a node converts a concrete sensor to an abstract sensor. The abstract sensors are *combined* or *integrated* to form an *abstract estimate*. The particular method of combining depends on the integration algorithm used. To keep the terminology simple, we refer to the abstract

(*) The authors work was supported in part by the Office of Naval Research under Grant No. ONR-N00014-91-J-1306 and in part by the LEQFS - Board of Regents under Grant No. LEQFS-RD-A-04. (**) Supported in part by the National Science Foundation under Grant No. CCR 8908189.

0-7803-1257-0/93/$3.00 © 1993 IEEE.

sensor as the abstract estimate also. An abstract estimate could, in turn, be combined with one or more abstract estimates to form a new abstract estimate.

Marzullo [10] considers the case of a processor receiving input from several sensors whose outputs are intervals. He gives a fault tolerant integration algorithm which takes as input the intervals representing the sensors and gives as output an interval which always contains the actual physical value. *A correct sensor is one whose interval contains the actual physical value.* Hence, any two correct sensors must intersect since they both contain the physical value being measured.

Marzullo considers the case when at most f ($f < n$) sensors are faulty in a n-sensor network. The physical value would then be contained in *any* of the $(n - f)$ intersecting intervals. Since it is not possible to decide which intersection contains the physical value, the smallest connecting interval containing all the $(n - f)$ intersections is taken to be the output of the processor. It can be seen that this final estimate contains the actual physical value. The final estimate, however, becomes arbitrarily wide as the number of faulty sensors becomes large. In such cases, an integration method described in [12] reduces the width of the final abstract estimate. For simplicity, we will use Marzullo's model for information integration in the proposed network.

In this paper we concentrate on competitive information integration. The architecture described here could be used effectively for complementary information integration in the presence of noise and possibly faulty sensors.

1.2 Notations and Definitions

We model the DSN by an undirected graph $G = (V, E)$, where each node represents one or more sensors and an associated PE of the network, and each edge represents a communication link of the network. The *length* of a path between two nodes is the number of edges encountered while going from one node to another. The *distance* between two nodes is the shortest length between the nodes. The *diameter* of the network is the largest distance between any two nodes in the network. The *degree of a node* is the number of edges associated with that node. The *degree of the network* is the largest degree of any node in the network.

Let y^f represents a binary number with bit y repeated f times; $\overline{y}$ represents the complement of y, and x represents the *don't care* bit. For example, the binary number 00011xx is represented by $0^3 1^2 x^2$. A node i in a network with $N = 2^k$ nodes has the binary address $i_{k-1}\ i_{k-2} \ldots i_1\ i_0$ where i_{k-1} (i_0) is the most (least) significant bit. The following definitions describe two address transforming functions append (*app*) and strip (*str*).

Let M be a k-bit number. Then,

$$app(M, y) = My$$

$$str(i_{k-1}i_{k-2}\cdots i_1 i_0) = i_{k-1}i_{k-2}\cdots i_1$$

For example, $app(000, 1) = 0001$ and $str(0010) = 001$. Note that $str(app(M, y)) = M$.

Our interest lies in multi-level networks (MLNs) in which each node of the network can be associated with a *level* number. An l-level network has l levels numbered from 0 to $l - 1$. The set of nodes at level m to which a node i at level m is connected form the *neighbors* of i. The set of nodes to which i is connected at level $m - 1$ form the *parents* of i. The set of nodes to which i is connected at level $m + 1$ form the *children* of i. In the MLN that we consider for the proposed DSN, there is a single node called the *root* at level 0, and each node at a higher level number has at most one parent and at most r children. We refer to such a network as a r-ary MLN. The node i at level $m(> 0)$ has the address $i_{m-1}i_{m-2}\cdots i_1 i_0$, where each digit $i_j \in \{0, 1, \ldots, r-1\}$ $(0 \le j \le m)$. This node i is connected to at most r children nodes whose addresses are $app(i, 0), app(i, 1), \ldots, app(i, r-1)$, and to its parent node whose address is $str(i)$. For every node i at level m, the relation $\Phi_m(i)$ yields the set of nodes to which i is connected at level m. In the network proposed, all but the the 0^{th} level of the network have the same interconnection scheme at each level.

Hence two nodes i and j in this network are connected if
(i) j = app(i,b), or
(ii) j = str(i), or
(iii) j = Φ(i)

where b ∈ {0,1, ... ,r-1}.

A real interval R = (R_l, R_u) is represented by a pair of real numbers; R_l is called the *lower bound* and R_u is called the *upper bound* of the interval R. We shall refer to real intervals simply as intervals.

The width of the interval, |R|, equals $(R_u - R_l)$. The *set theoretic intersection* of two intervals, X and Y is defined as

$$X \cap Y = \{c \mid c \in X \text{ and } c \in Y\}$$

Correspondingly, two intervals are said to *intersect* (or *overlap*) if their set theoretic intersection is non empty. Hence, if the set theoretic intersection of X and Y is non empty then their interval intersection is the interval $(max(X_l, Y_l), min(X_u, Y_u))$. A special case of interval intersection is interval inclusion. X *includes* Y if $X_l \le Y_l$ and $X_u \ge Y_u$. The *span* of two intervals X and Y is defined as

$$X \cup Y = (X_l, Y_u)$$

Note that the span operation between two non-overlapping intervals may result in an interval that includes points not lying in either of the intervals.

Intervals X and Y are said to be *non distinct* if either X includes Y or Y includes X; otherwise, X and Y are said to be *distinct*.

The following table defines the symbols we use in this paper.

Symbol	Description
ε	Maximum allowable deviation in time of a clock on a PE
α	Maximum allowable deviation in time of a clock on the central time server
κ	Maximum allowable drift rate in time of a clock on a PE
σ	Maximum allowable drift rate of the clock in the central time server
δ	Channel transmission delay
ξ	Delay in receiving the message sent by the central time server to any PE
γ	Maximum difference in time that a node can tolerate between intervals that can be integrated.

2.0 Architecture of the Distributed Sensor Network

This section describes the architectural features of the proposed network. We provide the motivation for this architecture by reviewing the past work of other researchers and pointing out the shortcomings of their approaches. In the next subsection, we list desirable features of a DSN and later show how the proposed network provides many of these features.

2.1 Motivation for a New Architecture

Wesson et al. [2] have described two architectures for a DSN. The first is the hierarchical or tree organization and the the second is the committee organization whose nodes can send messages to one, some, or all nodes in the network. The hierarchical network has several advantages like constant node degree and easy extensibility. It is not a good choice for a DSN, however, because a faulty node can disconnect an entire subtree. The committee organization does not have this disadvantage but is expensive and is not easily extensible.

It is clear from the above observations that both the committee organization and the tree organization have disadvantages; a combination that uses the merits of both the types of architectures is hence desirable. The flat tree network, referred to earlier, incorporates some of the merits of both these organizations. The nodes in this network are organized as many complete binary trees, the roots of which are completely connected. Figure 1 shows a flat tree network with 12 nodes. It has some disadvantages, however. For example, integration errors of the lower nodes *accumulate* as the information goes up the hierarchy. One way to overcome this problem is to interconnect nodes in the lower levels of this network.

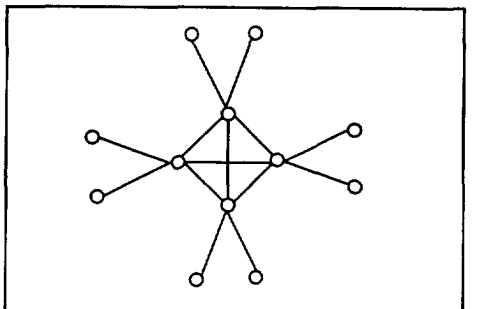

Fig 1. A flat tree network with 12 nodes

This motivates our proposal for a new class of networks which basically consists of the flat tree network with nodes at every level connected as a de Bruijn network. These networks have a committee organization at each level and an overall hierarchical organization. We will show that this class of networks has several advantages such as

i) they allow the construction of large networks with bounded degree,
ii) the diameter of these networks grows only logarithmically with the the number of nodes,
iii) they admit simple routing schemes,
iv) they possess fault tolerant capabilities, and
v) they have low addressing complexity.

2.2 The Proposed Architecture

The proposed DSN is a modified l-level MLN with the top level completely connected and with each of the other levels interconnected as a de Bruijn network. Before describing the proposed architecture for DSN, we briefly review the evolution of the de Bruijn network and mention its important features.

The use of de Bruijn networks as interconnection topologies for fault-tolerant parallel and distributed architectures was first proposed by Pradhan [3]. Pradhan [3] was also the first to propose the use of the de Bruijn network for VLSI architectures. Recently, de Bruijn networks have gained significant practical importance with the on-going implementation of a 8096 PE de Bruijn architecture by JPL for the Galileo project, scheduled for completion by 1995.

An important feature of the de Bruijn network is that it can be configured as many useful computational networks in spite of faults. In addition, de Bruijn networks have

(i) a small diameter
(ii) admit simple routing, and
(iii) possess good fault tolerant capabilities.

For a detailed discussion on the above mentioned features of de Bruijn networks, see the paper by Samantham and Pradhan [8]. And, for a summary of the evolution of the de Bruijn network, the reader is referred to the paper by Pradhan [13].

Using graph theoretic notation, the undirected de Bruijn network $DG(d,k)$ has $N=d^k$ nodes with diameter k and degree $2d$. We are interested in binary de Bruijn networks $DG(2,k)$ which have $N=2^k$. A node i of the network with the binary address $a_{k-1}\ a_{k-2}\ \ldots\ a_1\ a_0$ has neighbors:

$$a_{k-2}a_{k-3}\cdots a_1a_0a_{k-1} \quad \text{(i1)}$$

$$a_{k-2}a_{k-3}\cdots a_1a_0\overline{a}_{k-1} \quad \text{(i2)}$$

$$a_0a_{k-1}a_{k-2}\cdots a_2a_1 \quad \text{(i3)}$$

$$\overline{a}_0a_{k-1}a_{k-2}\cdots a_2a_1 \quad \text{(i4)}$$

The address of neighbors $i1$ and $i3$ is obtained by the left shift-end-around operation and the right shift-end-around operation on i respectively- they are called the *LR* and the *RR* neighbors of i. The address of nodes $i2$ and $i4$ is obtained by complementing the rightmost bit of $i1$ and the leftmost bit of $i3$ respectively- they are correspondingly called the *LRC* and the *RRC* neighbors of i.

The proposed DSN is organized as follows:
(i) The nodes in the topmost level are called commander nodes. There are 4 commander nodes which are completely connected.
(ii) The nodes in the underlying levels are interconnected as a binary de Bruijn
network.
(iii) Each node X, at level m in the network is connected to two children nodes app(X,1) and app(X,0) at level $m+1$ ($m<l-1$) and and is connected to its parent node str(X) at level m-1 ($m>0$).

Henceforth we shall refer to the proposed network as the multi-level binary de Bruijn network (BMD). Since the topmost level of the BMD contains 2^2 nodes, it is convenient to assign it level 2. Hence, an l-level BMD has l levels numbered from 2 through $l+1$. Figure 2 shows a 2-level BMD- the inter-level connections are shown by dashed lines and the intra-level connections by solid lines.

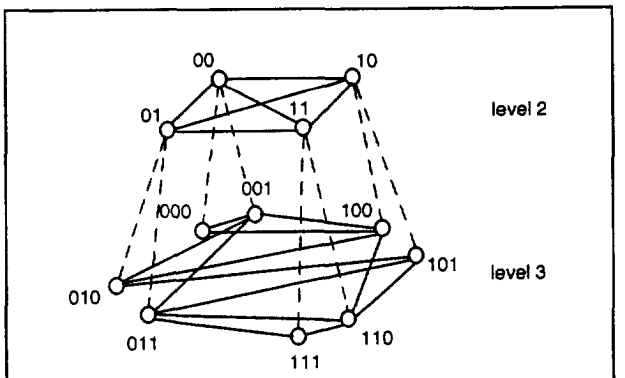

Fig. 2. BMD with 2 layers (level 2 and 3)

Each node of the BMD has a PE, a clock which maintains real time, an associated sensor which samples the physical variable(s) of interest, and an associated buffer. The PE translates the sensor reading into an abstract estimate, time stamps the estimate with the current time, and places the abstract estimate in the associated buffer. There is also a buffer associated with each link. The PEs connected to the link have access to this buffer. Figure 3 shows the architectural details of a node of the BMD. The additional details in the figure will be referred in Section 2.4. (*Note:* With slight modifications, we could allow for multiple sensors at each node.)

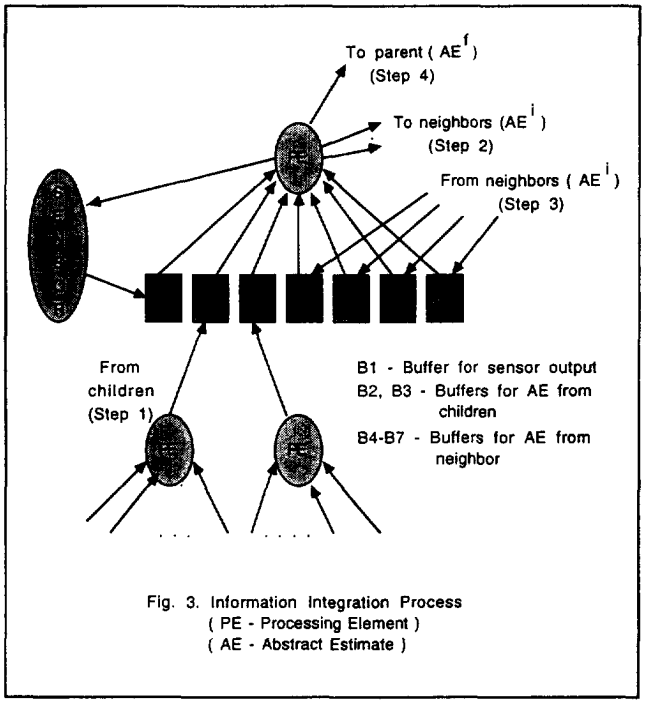

Fig. 3. Information Integration Process
(PE - Processing Element)
(AE - Abstract Estimate)

2.2.1 Topological Properties: The following lemmas describe the topological properties of the BMD.

Lemma 1: The number of nodes in BMD with L levels is $4(2^L-1)$.

Proof: The number of nodes at level m ($2<m\le l+1$),

$$n_m = 2\times n_{m-1};\ n_2 = 4$$

Solving this equation yields the total number of nodes as

$$N = 4(2^L-1) \quad (1)$$

Lemma 2: The BMD with L levels has degree 7 and diameter $L+1$.

Proof: The nodes at the top and bottom levels have degree at most 5. Now, consider an internal node in the network. This node is in a de Bruijn network and hence has at most 4 neighbors. The same node is also connected to its 2 children nodes and a parent node. Hence a node in the BMD has degree equal to at most 7. For deriving the diameter of the network, consider the lowermost level in the BMD. This corresponds to $DG(2,L+1)$ with diameter $L+1$. Note that

the farthest distance between nodes in the uppermost and lowermost level is only L. Hence the farthest nodes in the BMD lie in the lowermost level, i.e., the diameter equals $L + 1$. From (1), the diameter of the BMD is $O(\log N)$.

2.2.2 Addressing Scheme: Consider a BMD with L levels. By our convention, the highest level number in this network is $(L + 1)$. The address of a node in this network consists of two parts -

(i) the level number of the BMD in which it is present. This requires $\lceil \log(L) \rceil$ bits for its representation.

(ii) index of the node in that level. This requires at most $(L + 1)$ bits to index a node in any level, because the lowermost level (i.e., level $(L + 1)$) contains $2^{(L+1)}$ nodes.

The address of a node in a BMD with L levels, hence needs $\lceil \log(L) \rceil + (L + 1)$ bits.

2.2.3 Extensible Issues: To extend a BMD with L levels, we can add the additional nodes at the lowermost level. Thus, extending the network requires a fixed number of interconnections between the new nodes and the nodes at level $(L + 1)$ only. Note that the information integration process will not get affected at any other level of the BMD. Additional bits may be needed to address the nodes in the new level.

2.3 Routing

We show that messages can be routed efficiently in a decentralized manner in the BMD. We first consider routing within a level and then consider routing across levels. To evaluate the routing complexity, we assume that a message takes unit time to traverse a link.

2.3.1 Intra-Level Routing: Routing in the top level takes unit time step since the nodes are completely connected. Routing in a de Bruijn network is a well studied problem [4, 6]- consider the routing algorithm presented in [4]. In this algorithm, tag bits are appended to the message at the source before routing. These tag bits are used by intermediate nodes to compute the address of the next node in the path. This method assumes that all the nodes in the path are fault-free. Hence the algorithm will fail if any of the intermediate nodes or links are faulty.

In this section we describe two distributed routing algorithms PATH.1 and PATH.2 in which the address of the next node is computed at the previous node in the path. PATH.1 takes $\Theta(logN)$ steps in a de Bruijn network with N nodes, and PATH.2 takes $O(logN)$ steps.

Let a binary de Bruijn network have $N = 2^k$ nodes and let $S = s_{k-1}\ s_{k-2}\ ...\ s_1\ s_0$ be the source node that sends a message to the destination node $D = d_{k-1}\ d_{k-2}\ ...\ d_1\ d_0$.

The message consists of the data and the message header. The message header contains a routing tag whose content depends on the type of routing being performed. Two types of routing tags are used- one for normal routing (Type 1) and the other for fault tolerant routing (Type 2).

The Type 1 routing tag contains the source and destination node addresses, a counter (z), and an inter-level routing bit (IRB). The IRB bit is *set* if the source and the destination nodes are in different levels and is *reset* if the nodes are in the same level. The number of *message hops* from the source node to the current node is recorded in the counter, and is used to generate the address of the next node in the path. Figure 4 shows a Type 1 routing tag.

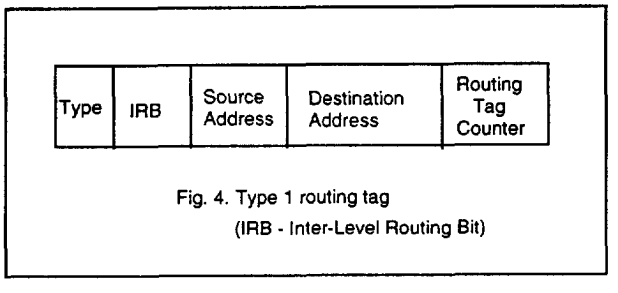

Fig. 4. Type 1 routing tag
(IRB - Inter-Level Routing Bit)

PATH.1 Algorithm

From the construction of the de Bruijn network we know that the source node has the following neighbors - $d_0\ s_{k-1}\ s_{k-2}\ ...\ s_1$ and $s_{k-2}\ ...\ s_1\ s_0\ d_{k-1}$. Using this property we can now generate two routes by appending successive bits of the destination node to the source address.

Route1

$(z = 0)$ $s_{k-1}s_{k-2}\cdots s_1s_0$ *(source)*

$(z = 1)$ $d_0s_{k-1}s_{k-2}\cdots s_1$

$(z = 2)$ $d_1d_0s_{k-1}\cdots s_2$

.

.

.

$(z = k)$ $d_{k-1}d_{k-2}\cdots d_1d_0$ *(destination)*

Route2

$(z = 0)$ $s_{k-1}s_{k-2}\cdots s_1s_0$ *(source)*

$(z = 1)$ $s_{k-2}s_{k-3}\cdots s_0d_{k-1}$

$(z = 2)$ $s_{k-3}\cdots s_0d_{k-1}d_{k-2}$

.

.

.

$(z = k)$ $d_{k-1}d_{k-2}\cdots d_1d_0$ *(destination)*

Clearly Route 1 and Route 2 take exactly $k = \log N$ steps.

Let $i_{k-1}\ i_{k-2}\ ...\ i_1\ i_0$ be the address of the node under consideration. The following steps (executed by the node) describe the PATH.1 algorithm:

1. If the label of the node is the same as the destination address in the routing tag, then accept the message.
2. Otherwise, check the value of the routing tag counter z. The address of the next node in the path is
   $d_z\ i_{k-1}\ i_{k-2}\ ...\ i_1$ (Route 1) OR $i_{k-2}\ i_{k-3}\ ...\ i_1\ i_0\ d_{k-z-1}$ (Route 2).
3. Increment the counter z and route the message to the next node.

Figure 5 shows a path from node 011 to node 101 in a DG(2,3) network using Route 2 of the PATH.1 algorithm.

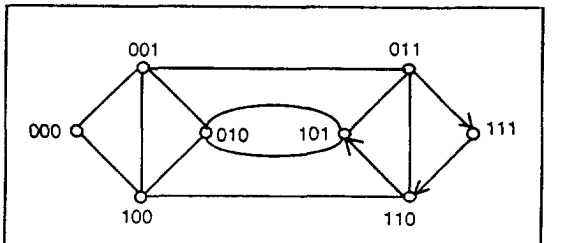

Fig. 5. PATH.1 route between 011 and 101

PATH.2 Algorithm

PATH.2 algorithm routes the message along the shortest path between the source and destination nodes. To find the shortest path, we treat the node addresses as binary strings and use a string matching algorithm described in [1].

Find the largest x such that $s_{x-1}\ s_{x-2}\ ...\ s_1\ s_0 = d_{k-1}\ d_{k-2}\ ...\ d_{k-x}$, and the largest y such that $s_{k-1}\ s_{k-2}\ ...\ s_{k-y} = d_{y-1}\ d_{y-2}\ ...\ d_1\ d_0$. We can compute x and y in $O(k)$, i.e., $O(logN)$ time. The following three cases arise depending on the relationship between x and y:

Case 1: ($x > y$)- the shortest path is given by the following sequence of nodes:

($z = 0$) $s_{k-1}s_{k-2} \cdots s_1 s_0$ (*source*)

($z = 1$) $s_{k-2}s_{k-3} \cdots s_0 d_{k-x-1}$

($z = 2$) $s_{k-3} \cdots s_0 d_{k-x-1} d_{k-x-2}$

.

.

($z = k - x$) $s_{x-1}s_{x-2} \cdots d_1 d_0$ (*destination*)

The destination $s_{x-1}s_{x-2} \cdots d_1 d_0 = d_{k-1}d_{k-2} \cdots d_1 d_0$ is reached after $(k - x)$ steps.

Case 2: ($x < y$)- the shortest path is given by the following sequence of nodes:

($z = 0$) $s_{k-1}s_{k-2} \cdots s_1 s_0$ (*source*)

($z = 1$) $d_y s_{k-1} \cdots s_2 s_1$

($z = 2$) $d_{y+1} d_y s_{k-1} \cdots s_2$

.

.

($z = k - y$) $d_{k-1}d_{k-2} \cdots s_{k-y+2}s_{k-y+1}$ (*destination*)

The destination is reached after $(k - y)$ steps.

Case 3: ($x = y$)- choose either of the above routings to obtain the shortest path.

Figure 6 shows the shortest path between nodes 011 and 101 in a DG(2,3) network using the PATH.2 algorithm. In this case x = 1 and y = 2; hence, the shortest path is of length 1. Using the PATH.1 algorithm yields a path length of 3.

The following steps describe the PATH.2 algorithm (as executed by node i):

1. If the label of the node ($i_{k-1}\ i_{k-2}\ \ldots\ i_1\ i_0$) is the same as the destination address in the routing tag, then accept the message.
2. Find the largest x such that $s_{x-1}\ s_{x-2}\ \ldots\ s_1\ s_0 = d_{k-1}\ d_{k-2}\ \ldots\ d_{k-x}$, and the largest y such that $s_{k-1}\ s_{k-2}\ \ldots\ s_{k-y} = d_{y-1}\ d_{y-2}\ \ldots\ d_1\ d_0$. If $x > y$ then the address of the next node in the path is $i_{k-2}\ i_{k-3}\ \ldots\ i_1\ i_0\ d_{k-x-z}$ where z is the value of the counter. If $y > x$ then the address of the next node in the path is $d_{y+z-1}\ i_{k-1}\ i_{k-2}\ \ldots\ i_1$.
3. Increment the counter z and route the message to the next node whose address was generated in step 2.

Note that the value of x and y need not be computed by all nodes in the path. Instead, the value of x or y can be transmitted in the message header.

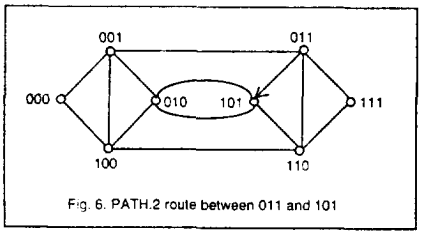

Fig. 6. PATH.2 route between 011 and 101

2.3.2 Routing between layers: Let the source (S) and destination (D) nodes be at levels L and $L - X$ respectively. At the source the *inter-level routing* (IRB) bit is set to "1" to indicate that the source and destination nodes are in different levels. Further, when the IRB bit is set, the routing tag counter is not incremented in order to maintain a proper value of the counter for intra-level routing following the inter-level routing.

The source node S first routes the message to its parent $str(S)$. This procedure is repeated recursively till the message is received by a node at the same level as the destination node D. The IRB bit is reset to "0" now, and the source address is replaced by the address of the node that received the message. The message can be then routed to the destination using PATH.1 or PATH.2 algorithm.

When the destination is at a higher level than the source, routing can be similarly done by using *app()* to generate the address of the next node in the path till the message reaches the same level as that of the destination node. The message can then be routed using PATH.1 or PATH.2 algorithm. Note that messages in the BMD are usually routed from higher to lower levels only.

2.4 Information Integration

In this section we describe the process of information integration in the BMD. The idea behind the integration is to a) keep the communication requirements small- this is done by communicating the abstract estimate as a *single* interval and b) maintain accuracy by ensuring that the physical values of interest is *always* contained in the abstract estimate.

Since the de Bruijn network has a connectivity of 2, the BMD can tolerate at most one node fault or link fault per level (except at the topmost level which is fully connected). We first state some results related to fault tolerance when abstract estimates (or intervals) are to be integrated in the presence of faults in the network.

Lemma 3: Consider n ($n \geq 3$) intervals of which at most one can be faulty. Then there can be at most two ($n - 1$) distinct interval intersections among these n intervals.

A direct consequence of Lemma 3 is the following theorem. Using the theorem, the search for a faulty node is narrowed down to at most nodes for each fault.

Theorem 1: Given a set of n intervals containing at most one faulty interval,

(i) there is no faulty interval if there is no *n-1*-interval intersection,

(ii) the interval not intersecting with an *n-1*-interval intersection is faulty if there is exactly one *n-1*-interval intersection, and

(iii) there are two potentially faulty intervals if there are two *n-1*-interval intersections one of which is incorrect.

In case (iii), the two potentially faulty nodes can be traced by taking the set difference of the interval names that belong to each (*n-1*) interval intersection.

We now describe the information integration process. For convenience, we will refer to the information integration of abstract estimates between distinct levels as "integration" and refer to the information integration within a level as "comparison."

Abstract estimates move upward from the leaf nodes to the commander nodes. Every non-leaf node of the network combines the abstract estimates of its two children and the local sensor (sensor associated with this PE) to arrive at a new abstract estimate (AE^i). This step is called the "integration" step.

In the integration step, we assume that at most one of the three (local sensor and 2 children) received abstract sensor estimates is incorrect. The new abstract estimate is found from the three cases (refer Figure 7) that could arise (Theorem 1). If there are two 2 interval intersections, then the smallest interval containing these intervals forms the new abstract estimate. It can also be shown [10] that this new estimate is at most as wide as one of the input abstract estimates.

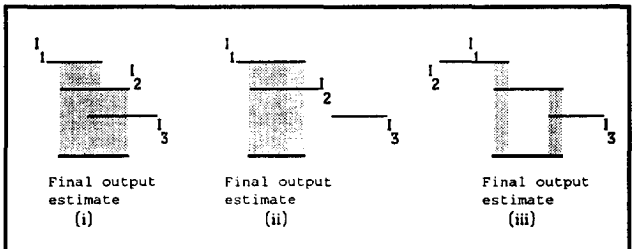

Fig.7. Integration step
(I - I are the 3 intervals)
I 3

Next, each node sends its AE^i to all its neighbors. When a node receives AE^is from its neighbors, it combines them to arrive at a new estimate AE^f. This step is called the "comparison" step and the algorithm used to combine the estimates is similar to the one described for the integration step. In this step, however, a node combines 3, 4, or 5 estimates depending on the number of its neighbors (2, 3, or 4 respectively).

Since the BMD can tolerate at most one fault (node or link) per level, one of the estimates received from a neighbor could be incorrect. Hence, when a node receives i ($i = 3$, 4 or 5) intervals in the comparison step, it chooses the smallest interval containing all (which is at most two as shown in lemma 3) the *i-1*-interval intersections as the output. The width of this abstract estimate is again at most as wide as one of the input correct intervals.

Figure 8 shows the comparison process in a node of the network.

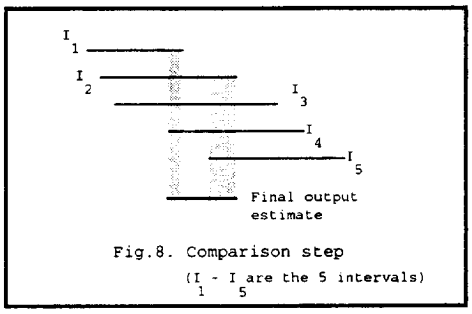

Fig.8. Comparison step

(I - I are the 5 intervals)
1 5

If there are two i-1-interval intersections in the comparison step, then we know that there exists an incorrect interval. Identifying the faulty node which sent this incorrect interval requires the diagnostic testing of at most two nodes as we showed in Theorem 1. Once a node has been identified as faulty, appropriate action can be taken so as to either "repair" the faulty node or replace it and notify the parent and children of the faulty node. In this paper we shall not concern ourselves with the problems of identifying the cause of faulty behavior and attempting to rectify that cause.

The following steps summarize the process of information integration:

Step 1: Receive abstract estimates from children and integrate them with abstract estimate from local sensor to get AE^i.

Step 2: Send AE^i to neighbors.

Step 3: Receive abstract estimate from neighbors and "compare" with own abstract estimate to compute AE^f. Identify any faulty node in the process.

Step 4: Send AE^f to parent node.

Note that this process of information integration ensures that only the "correct" estimates move up to the commander nodes in the network. Further, *the width of the estimates moving upwards is bounded by the width of one of the correct estimates of that level of the BMD.* An incorrect estimate would be received by a parent only when the child or the link connecting the two nodes is faulty.

Figure 3 shows the complete information integration process at a node in the network.

2.5 Fault Tolerant Issues

In a large network it is unrealistic to expect all the nodes or links along a path to be fault-free at all times. When some nodes or links fail, an alternative path that avoids the faulty node or link must be derived. One of the major advantages of our network over the network proposed in [11] is that abstract estimates can be routed around faults using the interconnections between nodes at the same level.

A node is faulty if it sends an incorrect abstract estimate to its parent or to any of its neighbors. Link faults can be detected if a node does not receive the abstract estimate of its neighbor during the comparison step. When a node (a node failure is assumed to be equivalent to the failure of all links associated with it) or link failure is detected, any of the following actions can be taken -

(1) The fault can be ignored during the integration process. After integration is complete and abstract estimates have been sent to the upper level of the BMD, the node which detected this fault can run a diagnostic algorithm on the faulty node or link after isolating it.

(2) If the faulty node or link is in the path of an abstract estimate transmitted towards its destination, this abstract estimate can be re-routed around the fault to the destination. After the integration process is complete the node which detected the fault can run a diagnostic algorithm on the faulty component.

The BMD network provides fault tolerance by taking both of the remedial actions mentioned above.

Suppose a node X, with children Y and Z, is faulty. In the flat tree [11] network the subtree rooted at X is unusable. In the BMD network, however, the abstract estimates of Y and Z are also read by the neighbors of Y and Z. Thus the abstract estimates of Y and Z get factored into the final abstract estimates produced by the neighbors of Y and Z. Hence the subtree rooted at X does not become unusable - only the faulty node is unusable. Moreover, X is identified as a faulty node during the comparison step because its abstract estimate (which it sends to its neighbors) may not contain the physical value.

If action (2) is taken by the neighbor of the faulty node, then it must reroute the abstract estimate received, around the faulty node to the destination. This means that the destination node must wait for more time to receive the abstract estimate, because additional hops may be required for rerouting the message. This requires that the value of γ (maximum difference in time that a node can tolerate between intervals that can be integrated- please see the next section on clock synchronization) be increased to maintain the "near synchronous" behavior of the sensor network. Note that by increasing the value of γ, the network would tolerate single node/link fault but the process of sensor integration would be slowed down. Samantham and Pradhan [8] mention that four additional hops are enough to avoid a single node fault in a binary de Bruijn network. Since the nodes in every level (except the top level) in the BMD are arranged in a binary de Bruijn network, the value of γ will have to be increased by four time units.

In the remaining part of this section, we show one way of avoiding a single node fault using exactly four hops, when routing in any level of the BMD except the topmost level. Let $s_{k-1}\ s_{k-2}\ ...\ s_1\ s_0$ be the source node and $d_{k-1}\ d_{k-2}\ ...\ d_1\ d_0$ be the destination node. Application of the PATH.1 algorithm yields the following path:

$(z = 0)$ $s_{k-1}s_{k-2}\cdots s_1 s_0$ *(source)*

$(z = 1)$ $s_{k-2}s_{k-3}\cdots s_0 d_{k-1}$

$(z = 2)$ $s_{k-3}\cdots s_0 d_{k-1}d_{k-2}$

.

.

$(z = m-1)$ $s_{k-m}s_{k-m-1}\,..\,s_0 d_{k-1}\,..\,d_{k-m+1}$ (i1)

$(z = m)$ $s_{k-m-1}s_{k-m-2}\,..\,s_0 d_{k-1}\,..\,d_{k-m+1}d_{k-m}$ (i2)

$(z = m+1)$ $s_{k-m-2}\,..\,s_0 d_{k-1}\,..\,d_{k-m}d_{k-m-1}$ (i3)

.

.

$(z = k)$ $d_{k-1}d_{k-2}\cdots d_1 d_0$ *(destination)*

Assume that either node $i2$ or the link between $i1$ and $i2$ has failed. We now show an alternative route (reroute) between $i1$ (rerouting source) and $i3$ (rerouting destination) that takes only four additional hops. Note that this technique is independent of the number of nodes in the level.

When $i1$ receives a message (consisting of the abstract estimate and the Type 1 tag), it appends four fields to the Type 1 tag which enable rerouting of the message- (1) source address ($i1$), (2) destination address ($i3$), (3) *r*eroute counter (RC), and (4) *r*erouting *b*it (RRB). We shall refer to the tag, formed by appending reroute fields to the Type 1 tag, as Type 2 tag. Figure 9 shows a Type 2 tag. To initiate rerouting, $i1$ increments z and sets RRB = "1". When RRB = "1", a node does not increment z; instead it uses RC to compute the address of the next node in the reroute. When the message reaches $i3$, $i3$ removes the reroute fields from the tag, and increments z. Routing from $i3$ then proceeds normally using PATH.1 or PATH.2 algorithms.

Type	IRB	Source Address	Destination Address	Routing Tag Counter	Source Address	Destination Address	RC	RRB

Fig 9. Type 2 routing tag (rerouting tag)
(RRB - Re-Routing Bit)
(RC - Rerouting Counter)

The alternative route between i_1 and i_3 is shown below:

$(z = m-1)$ $s_{k-m}s_{k-m-1}\cdots s_0 d_{k-1}\cdots d_{k-m+1}$

$(z = m)$ $s_{k-m-1}s_{k-m-2}\cdots s_0 d_{k-1}\cdots d_{k-m+1}\overline{d}_{k-m}$

$(z = m)$ $s_{k-m-2}s_{k-m-3}\cdots s_0 d_{k-1}\cdots d_{k-m+1}\overline{d}_{k-m}d_{k-m-1}$

$(z = m)$ $d_{k-m-1}s_{k-m-2}\cdots s_0 d_{k-1}\cdots d_{k-m+1}\overline{d}_{k-m}$

$(z = m)$ $d_{k-m}d_{k-m-1}\cdots s_0 d_{k-1}\cdots d_{k-m+1}$

$(z = m)$ $d_{k-m-1}s_{k-m-2}\cdots s_0 d_{k-1}\cdots d_{k-m+1}d_{k-m}$

$(z = m+1)$ $s_{k-m-2}s_{k-m-3}\cdots s_0 d_{k-1}\cdots d_{k-m+1}d_{k-m}d_{k-m-1}$

The above route takes 6 steps- only 4 more than the normal route between i_1 and i_3. Figure 10 shows fault tolerant routing in a DG(2,3) (level 3) between nodes 001 and 110 when the node 011 is faulty. This alternative route can be chosen when a faulty node is encountered in the path to the destination node.

Hence the routing algorithms given earlier can be easily adapted to take the alternative path in case of faults. Our rerouting algorithm is also more adaptive to faults than the one presented in [4] since our algorithm does not require that the presence of a fault be known to the source node as the other algorithm does.

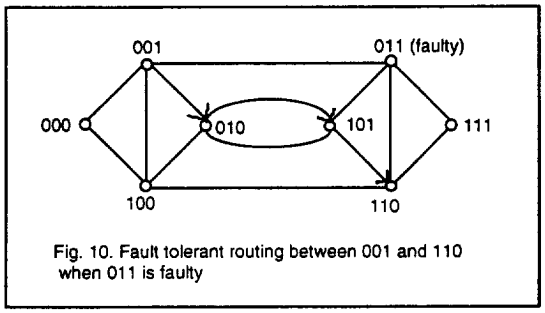

Fig. 10. Fault tolerant routing between 001 and 110 when 011 is faulty

Finally, since the network can sustain one node or link fault at every level, the BMD network with L levels and $N = 2(2^L - 1)$ nodes can sustain L, i.e., approximately $\log N$, node or link faults.

3.0 Clock Synchronization Issues

So far we have assumed that any two abstract estimates can be integrated. In reality, since the sensor outputs typically change as a function of time, only estimates that are temporally "close to each other" must be integrated if meaningful results are desired. This is achieved by time-stamping each estimate. The condition under which two estimates may be integrated is given at the end of the next subsection. In a distributed environment such as ours, there is no central synchronized clock which regulates the activities of each node. Instead, each node is under the control of its own clock. Since the sensor responds to real-time events it is convenient for the clock to provide the real, i.e., physical time. Further , since the estimates from different sensors have to be integrated, it is necessary for the time provided by the clocks of the sensors to be "close to each other". The clock at each node may not be accurate because of a variety of reasons such as clock shift, change in temperature, etc. Each clock must therefore synchronize with a more accurate clock. We assume the existence of a central time server which when requested for the time at t, provides the time $C(t)$. The PEs in our DSN spatially lie within tens of feet from each other, hence the existence of a single time server for the clocks on all the PEs can be assumed. The central time server itself periodically synchronizes with a universal time server, which is always accurate and lies outside our environment.

We use the clock model described in [11] to synchronize the clocks in the BMD. We summarize the basic results of the model in the next two sections.

3.1 Clock behavior and Synchronization

Let $C_p(t)$ be the time provided by the clock on PE p at time t (t is the time provided by a universal time server). We assume that the clocks run continuously rather than in discrete "ticks". Hence $\frac{dC_p(t)}{dt}$ denotes the rate at which the clock is running at time t. We also assume that that this rate is non-negative; hence, the time on the clocks monotonically increase.

We now state the conditions on the clocks for proper synchronization.

Clock Condition 1: The deviation in time of each clock is bounded, i.e., for PE p, there exists $\varepsilon_p << 1$ and $\alpha << 1$ such that

$$\left| t - C_p(t) \right| \leq \varepsilon_p \tag{2a}$$

$$|t - C(t)| \leq \alpha \tag{2b}$$

Clock Condition 2: Between synchronizations, the drift rate of the clock is bounded, i.e., for PE p, there exists $\kappa_p << 1$ and $\sigma << 1$ such that

$$\left| \frac{dC_p(t)}{dt} - 1 \right| \leq \kappa_p \tag{3a}$$

$$\left| \frac{dC(t)}{dt} - 1 \right| \leq \sigma$$

Clock Condition 3: The clock on each of the PEs and the central time server advance monotonically.

For simplicity, we assume that ε_p and κ_p is the same for all PEs, i.e. $\varepsilon_p = \varepsilon$ and $\kappa_p = \kappa$. From Clock Condition 1 we have,

Synchronization Bound: If p and q are two PEs then

$$\left| C_p(t) - C_q(t) \right| \leq 2\varepsilon \tag{4}$$

Let δ_{min} and δ_{max} be the minimum and maximum values of delay for a message sent by a PE to its neighbor. Let γ be the maximum difference in time that a node can tolerate between intervals that can be integrated. Note that the value of γ will depend on the longest path between leaf nodes and commander nodes, which is equal to L in a BMD with L levels.

The following lemma and theorem state precisely the conditions for combining abstract sensor estimates which are temporally "close to each other." The discussion in this section follows closely the discussion presented in [11]. We therefore state all the results without proof. The interested reader is referred to [11].

Lemma 4: Let a message be received by PE p at $C_p(t)$. Then this message was sent in the interval $(C_p(t) - 2\varepsilon - \delta_{max}, C_p(t) + 2\varepsilon - \delta_{min})$.

The time stamp of an abstract estimate may not belong to the interval given above if the channel is faulty. The following theorem gives the condition under which estimates are "temporally close" and may be integrated.

Theorem 2: Let the three proper abstract sensor estimates I_1, I_2 and I_3 be received by PE p at times

$$C_p(t1) < C_p(t2) < C_p(t3)$$

respectively. Then I_i ($i = 2, 3$) can be integrated, iff

$$(C_p(ti) - C_p(t1) + 4\varepsilon + \delta_{max} - \delta_{min}) \leq \gamma$$

Since the clocks on the central time server and each of the PEs drift, they have to be periodically reset. We now state a bound on the time period between synchronizations. Let T_s be the time period of synchronizations of the central time server, and let T_c be the time period between synchronizations of the PE p.

The central time server synchronizes itself every T_s seconds with a perfect universal time server which exists outside the environment of our DSN. The central time server also synchronizes the clock on a PE every T_c seconds.

Let ξ_{min} and ξ_{max} be the minimum and maximum of the delay in receiving the message sent by the central time server to any PE. Also, let T^i_s and T^i_c be the periods corresponding to T_s and T_c as *observed* by the central time server. The bounds on these observed periods can be shown to be [11]:

Theorem 3: The time period as observed by the central time server between synchronizations of the central time server is bounded by

$$T^i_s \leq \alpha(\frac{1}{\sigma} - 1)$$

Theorem 4: The time period as observed by the central time server between synchronizations of the clock on a PE is bounded by

$$T_c{}^i \leq \frac{\varepsilon - 2\alpha - (\xi_{max} - \xi_{min})}{\kappa} - \alpha$$

4.0 Conclusions

The effective use of multiple sensor systems requires the solution of various problems relating to sensor models, the architecture of the sensor network, the integration of information at each node of the network, the cost of information transmission, and the fault tolerance of the network. The integration of information in real time requires the clocks at each of the nodes be synchronized. Synchronization of clocks is a non-trivial task in such distributed sensor networks. In an earlier paper [11], some issues related to the architecture of DSNs, information integration, and clock synchronization had been addressed. This paper extends the study by considering a more sophisticated architecture for DSNs which has a number of advantages including the ability to tolerate single node or link faults at each level.

Since our focus has been primarily on computational issues, we have chosen to represent sensor output information by Marzullo's simple and elegant model which is based on real valued intervals. We have also used a relatively simple information integration algorithm. We are aware that sensor modeling is itself a very detailed area of study [7] and that very sophisticated methods exist for information integration. We have also assumed that the output of each sen-

sor is a physical value. The above discussion and results easily extend to the case when the output of a sensor is a vector rather than a single value.

This study could be extended in several directions. A straightforward extension is to assign weights to the abstract estimates produced as a function of its level in the hierarchy. We also plan to investigate more sophisticated fault tolerant strategies for the de Bruijn network than the scheme presented here. A future goal of our project is to investigate the computation and communication requirements of more sophisticated integration algorithms executing on large scale DSNs.

Acknowledgements

The authors would like to thank Dr. Dhiraj K. Pradhan for his help during the preparation of the paper.

References

[1] Knuth, D. E., Morris, J. H., and Pratt, V. R., "Fast pattern matching in strings," SIAM J. Computing, Vol. 6, pp. 323-350, 1977.

[2] Wesson, R. et. al., "Network Structures for Distributed Situation Assessment," IEEE Trans. on SMC., Jan. 1981, pp. 5-23.

[3] Pradhan, D. K., "Interconnection Topologies for Fault-Tolerant Parallel and Distributed Architectures," Proc. 10th Int. Conf. Parallel Processing, Aug. 81, pp. 238-242.

[4] Pradhan, D. K., and Reddy, S. M., "A Fault-Tolerant Communication Architecture for Distributed Systems," IEEE Trans. on Comp., Vol. c-31, No. 9, 1982.

[5] Pradhan, D. K., "Dynamically Restructurable Fault-Tolerant Processor Network Architecture," IEEE Trans. on Comp., Vol. c-34, No. 5, May 1985.

[6] Esfahanian, Abdol-Hossein, and Hakimi, S. L., "Fault-Tolerant Routing in de Bruijn Communication Networks," IEEE Trans. on Comp., Vol. c-34, No. 9, 1985.

[7] Durrant-Whyte, H. F., "Sensor Models and Multisensor Integration," Int. J. Robot. Res., Vol. 7, No. 6, 1988.

[8] Samantham, Maheswara R., and Pradhan, D. K., "The De Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI," IEEE Trans. on Comp., Vol. 38, No. 4, 1989.

[9] Luo, R. C., and Kay, M. G., "Multisensor Integration and Fusion in Intelligent Systems," IEEE Trans. on SMC, Vol. 19, No. 5, 1989, pp. 901-927.

[10] Marzullo, K., "Tolerating Failures of Continuous-Valued Sensors," ACM Trans. on Computer Systems, vol. 4, no. 4, Nov. 1990, pp. 284-304.

[11] Jayasimha, D. N., Iyengar, S. S., and Kashyap, R. L., "Information Integration and Synchronization in Distributed Sensor Networks," IEEE Trans. on SMC, vol. 21, no. 5, Sept. 1991, pp. 1032-1043.

[12] Prasad, Lakshman, Iyengar, S. S., Kashyap, R. L., Madan, R. N., "Functional Characterization of Sensor Integration in Distributed Sensor Networks," IEEE Trans. on SMC, vol. 21, no. 5, Sept. 1991.

[13] Pradhan, D. K., "Fault-Tolerant VLSI Architectures Based on de Bruijn Graphs (Galileo in the Mid Nineties)," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 5, 1991.