Evaluating the Connectivity of Self-Assembled Networks of Nano-scale Processing Elements

Jaidev P. Patwardhan[†], Chris Dwyer[‡], Alvin R. Lebeck[†], and Daniel J. Sorin[‡]
{jaidev,alvy}@cs.duke.edu, {dwyer,sorin}@ee.duke.edu.

†Department of Computer Science
Duke University
Durham, NC 27708

‡Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708

Abstract

Architectures built using bottom-up self-assembly of nanoelectronic devices will need to tolerate defect rates that are orders of magnitude higher than those found in current CMOS technologies. In this paper, we describe and evaluate an approach to provide defect isolation in such an architecture that consists of a large number of simple computational nodes, each of which can communicate with four neighbors on single-bit asynchronous links. Our approach does not require an external defect map, nor does it require redundancy of complex computational circuits, either of which will limit the scalability of the system. We use the reverse path forwarding broadcast routing algorithm, commonly used in wide-area networks, to map out defective nodes at startup. The algorithm guarantees two things (a) the broadcast eventually terminates and (b) all functional nodes that have a path to the broadcast source will receive it. Thus, all functional and reachable nodes are connected through a broadcast tree, resulting in defect isolation. Simulations show that, for a fail-stop model of node failure, the broadcast connects all nodes that are reachable from the source. In case of low defect rates (≤ 10%), the broadcast reaches more than 97% of non-defective nodes. For a network of nodes in the form of a grid, our results show that, in most cases, the time taken to complete the broadcast is proportional to the square root of the number of nodes in the system. Finally, we also present an analysis of the characteristics of the trees generated by our broadcast mechanism.

1 Introduction

The ability to scale down the feature size in CMOS has allowed the semiconductor industry to match and even surpass the fast pace of progress dictated by Moore's law. However, we are fast approaching hard physical limits that will make it difficult if not impossible to shrink CMOS devices below a certain threshold. Recent semiconductor industry roadmaps [12] have encouraged the investigation of alternate device technologies to replace CMOS. This has led to the development of a variety of interesting electronic devices, including nanocells [23], carbon nanotube transistors (CNT) [1,21], silicon nanowires [3,10], and silicon nanorods [17]. These devices are extremely small, and thus need very low charge transfers to switch state. This small size and low charge give rise to desirable power consumption characteristics but also makes circuits made using these devices susceptible to defects and faults.

One of the primary advantages of using emerging nanoelectronic devices is the potential for greater device density. It will be hard to adapt conventional top-down fabrication techniques like optical lithography for use with these nanoscale devices. This is largely because of the conflict between the small wavelengths required in the lithography process, the high energy associated with shorter wavelengths and the accuracy needed to fabricate devices. There has been significant research in bottom-up alternatives to optical lithography, particularly in DNA self-assembly using DNA as a scaffold material to attach electronic devices [2,13,14,20,25]. Self-assembly is well suited to assemble large numbers of dense circuits, however, it is also prone to higher defect rates than those produced by optical lithography. This is because self-assembly does not have the precise control over the placement of devices that can be achieved by optical lithography.

Systems built using bottom-up self-assembly of nanoelectronic devices will need to incorporate defect tolerance in their design to maintain their advantage over CMOS. Past work on defect tolerance has included schemes like NAND multiplexing [8,18], voting mechanisms [16,24] and obtaining defects maps to allow configuration around defects [4,7,9]. The requirement of an external defect map, or extensive redundancy make it difficult to adapt these schemes systems with billions or trillions of processing elements.

In this paper, we describe a mechanism for tolerating defects in a large system built using DNA-guided self-assembly of nanoelectronic devices. We use the reverse path forwarding (RPF) algorithm for broadcast routing [5] once at startup to create a broadcast tree of non-defective nodes. This maps out defects in the system at run-time and allows us to use the largest connected subset of the random network reachable from the source of the broadcast for useful computation. Our approach does not require us to extract a defect map to configure the system to avoid defects. In our previous work [19], we used the broadcast mechanism described in this paper to isolate defects and impart logical structure to a random network of nodes. The structure was then used by an architecture to build a memory system and an execution network to run simple programs.

The goal of this work is to evaluate the characteristics of self-assembled networks. We make the following contributions: 1) we evaluate the efficiency of our broadcast mechanism by computing the latency and "coverage" of the broadcast (the fraction of the non-defective nodes that the broadcast reaches) for different network sizes, 2) we evaluate the connectivity of the self-assembled network and analyze the properties of the broadcast tree generated by the broadcast, 3) we outline some limitations of the current approach and suggest techniques to overcome the limitations.

The rest of the paper is organized as follows. Section 2 presents an overview of the target system, Section 3 describes our defect tolerance mechanism. In Section 4 we describe our simulation setup, and in Section 5 we present and analyze our results. Section 6 discusses related work, and we conclude in Section 7.

2 Large Scale Self-Assembled Systems

The parallel nature of self-assembly enables the manufacture of systems with a large number of identical components. However, self-assembly is not as precise a manufacturing technique as optical lithography. This lack of precision increases the probability that some of the manufactured components will be defective. Our defect tolerance mechanism is targeted towards systems with up to 10^{12} self-assembled processing elements. This is several orders of magnitude larger than previous systems that have incorporated defect tolerance in their design. The scale of the targeted system makes it impractical to use an external defect map [4,7,9] to configure the system around defects. In the rest of this section, we describe the properties of our self-assembled system. While we use a specific instance [19] of such a system in our evaluation, in general, our defect tolerance mechanism is applicable to large systems that are composed of processing elements that need to communicate with each other. We break up our discussion of the system into three components. First, we discuss the use of self-assembly in our system. Next, we discuss the capabilities of each node in the system, and then describe our interface with the external micro-scale world. Finally, we provide a high-level overview of the capabilities of the whole system.

2.1 Self-Assembly

The system we target is composed of a self-assembled network of nano-scale nodes. To build a large scale system with up to 10^{12} interconnected nodes, we use hierarchical self-assembly. At the lowest level, we use DNA-guided self-assembly of nanoelectronic devices to create small processing elements. The self-assembled DNA lattice acts as a scaffolding for the nanoelectronic devices. Figure 1 shows an image of a DNA lattice taken with an atomic force microscope. Using appropriate DNA segments during the construction of the lattice, the lattice can be made "addressable". This addressability allows us to place active devices at specific positions in the lattice to form a circuit.

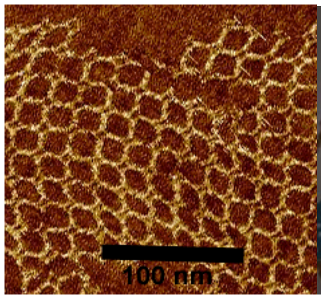

FIGURE 1. A DNA scaffolding for nanoelectronic circuits

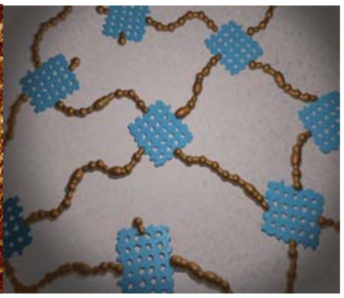
FIGURE 2. Schematic of self-assembled network of nodes

Once individual nodes have been self-assembled, they need to be linked together to form a network of nodes. This is achieved by growing DNA nanotubes between nodes, and then metallizing them to make them conductive [15,26]. This second level of self-assembly gives rise to a random network of nodes. Figure 2 shows a schematic of a section of the network of nodes, including regions with defective or disconnected links.

2.2 Nodes

We impose minimal requirements on the capabilities of each node. Each node is assumed to have four transceivers, a simple single-bit ALU and some control circuitry. Each transceiver controls the transmission and reception of data over a single-bit asynchronous link that connect nodes in the random network. A node has up to four active links, each connected to a transceiver. The ALU can perform simple arithmetic and logic operations on single-bit data. It can store a single-bit of data, like a carry bit. Each node has some storage space for global and local state. Finally, nodes have circuitry to control the flow of data through them. This includes control over the routing and actual decisions about performing operations in the ALU. While the nodes described here are very simple, they represent a lower bound of the requirements for our scheme. Any system that uses more complex nodes would work equally well with our defect isolation mechanism.

2.3 External Interface

We need to have a way to interface the system with the external world. In this work, we assume that there are multiple such interfaces (called "vias") (see Figure 3) scattered across the random network of nodes. Each via overlaps several nodes, but is controlled through a single node, called the anchor node. All external input is inserted through the vias.

2.4 System Architecture

The hierarchical self-assembly process builds a random network connecting the nodes described in Section 2.2. The total computing power of such a system with a large number of nodes is tremendous. For example, if each node operates at 1GHz, and can perform a single bit ALU operation per cycle, the peak performance of a system with 10^{12} nodes would be 31.25×10^{18} 32-bit operations per second. However, this assumes the ability to use and perfectly co-ordinate all nodes, which is unrealistic. To tap the tremendous computing power of this system, nodes must be able to communicate with each other.

In order for the nodes to communicate, we need to impose some structure on the random network. There are multiple ways in which this can be achieved. We have explored one architectural approach [19] that imposes logical structure on the network using the broadcast mechanism. Our design is similar to an "Active Network" [22]. Nodes communicate using "packets" of information that hold both instructions and data that the instructions operate on (thus the resemblance to an active network). The architecture provides a mechanism for configuring a memory system, defines the execution-memory interface and the execution model. We have demonstrated the execution of simple programs on this architecture. The architecture relies on the broadcast mechanism to impose logical structure on the random network of nodes. In the next section, we describe how we impose this structure and isolate defects.

3 Defect Tolerance

Our defect tolerance strategy involves a simple configuration step at startup to connect non-defective nodes together that results in the isolation of defective regions. Before we describe the defect tolerance mechanism, we present the defect model used in this work.

3.1 Defect Model

For the purposes of this work, we assume a simple fail-stop defect model for the node. If a node fails or is defective, it is completely isolated from its neighbors, i.e, it cannot perform any processing or communication. We do not examine more complex defect models involving partially defective nodes. Short-circuited links are handled at the architectural level using a back-off mechanism. At start-up, nodes assert signals on their links. If a node detects more than two asserts on a link, it assumes that there are already two active nodes on the link and shuts down the corresponding transceiver.

3.2 Isolating Defects

The key to defect tolerance in our scheme is isolating defects using the reverse path forwarding (RPF) broadcast routing algorithm [5]. Section 2.3 introduced our concept of a via which is an interface between the system and the micro-scale world. We use a via close to the boundary of the system to insert a special broadcast packet into the network. Each node then forwards the packet using the RPF algorithm. On receiving this packet (called a gradient packet), the node broadcasts it on all its links, except the link that it received the packet on. Before forwarding the packet, the node stores the id of the link it received the packet on. Once a node gets a gradient packet, it does not forward any other gradient broadcast packets it receives. This ensures that the broadcast eventually terminates. Once all broadcast activity stops, we have effectively established a "gradient" [11] broadcast tree rooted at the via where we inserted the broadcast packet. Each node that received a gradient packet knows how to get a packet to this via.

We use vias located at four ends of the system to broadcast four "gradients" across the system. The idea is to set up a general routing framework with the ability to route in four directions (corresponding to each of the gradients). This routing framework can be used by a higher level architecture to route instructions and data across the system. To allow multiple gradient broadcasts in the network, we add a gradient ID (GID) field to each packet, such that each node runs the RPF algorithm once per gradient. By examining the GID in the packets, the nodes can decide whether to propagate the broadcast (in case of a GID not seen before), or to squash the broadcast (in case of a repeated GID).

The gradient broadcast mechanism also helps us achieve defect isolation. Since defective nodes cannot participate in the gradient forwarding process, no node ever receives a gradient packet from a defective node or link. This implies that we can never route data into a defective node, thus achieving defect isolation. The gradient broadcast mechanism is fairly robust as defect rates increase. As long as there are large connected components in the random network, the gradient mechanism will connect all nodes within that region if the gradient source is also included in that region.

We illustrate gradients in a network in Figure 3. The figure shows a small network, with each node having an arrow pointing in the direction that it received the gradient from (the gradient that originated from via 1). The absence of nodes (i.e. white spaces in place of nodes) corresponds to defects. The network in the figure has five vias, one in each corner four vias and one in the center (via 5). The figure illustrates how the gradient broadcast covers a large part of the network. It also shows how defects can cause regions of non-defective nodes to get isolated (region 1 and 2). In the next section, we describe the experimental setup we use to

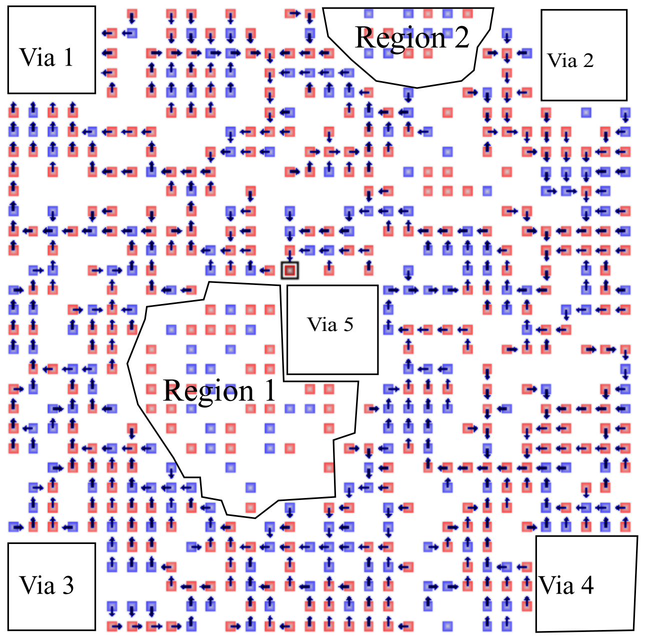

FIGURE 3. Gradient directions in a small network of nodes

evaluate the connectivity of our network of nodes equipped with this defect tolerance mechanism.

4 Experimental Setup

We use a custom event driven simulator to evaluate the defect tolerance mechanism. In this work, the network of nodes is assumed to be a regular grid, with defects distributed randomly on the grid. The user specifies various system parameters, including defect rate and network size (number of nodes), as input to the simulator. The simulator first creates the nodes and arranges them in a square grid, connecting each node to its four nearest neighbors. Then, using the defect probability and a random number generator, we mark certain nodes to be "defective". Once a node has been marked defective, it ceases to be part of the network.

In our experiments, we vary the defect rate from 0% to 50% defects. We vary network size from 30x30 nodes to 100x100 nodes arranged in a regular grid. The simulator is capable of running larger topologies, but we are limited by the simulation time required to generate statistically meaningful results. For each configuration, we present the average of 50 runs with random seeds used to generate distinct node topologies with different defect locations. All experiments use a single gradient source on the side of a square grid (except in Section 5.3).

5 Evaluation and Analysis

To evaluate the performance of our defect tolerance mechanism, we ask the following questions.

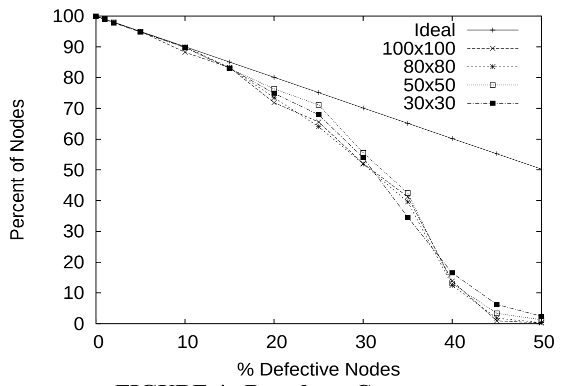

FIGURE 4. Broadcast Coverage

What is the coverage of the broadcast? Ideally, the broadcast should reach all non-defective nodes. However, there could be cases where some nodes are cut-off due to the presence of surrounding defects. (Section 5.1)

What is the latency of a gradient broadcast as a function of network size? The best case latency in a network with NxN nodes would be O(N). This would be obtained in the absence of all defects. In the worst case, the gradient needs to traverse the entire network in a linear manner, giving a worst case latency of O(N^2). (Section 5.2)

What is the effect of changing the location of the gradient insertion point in the network? The location of the source of the gradients should make a difference in the coverage and latency of the broadcast mechanism. Conceptually, the source should be placed in a region that minimizes the chances of it being cut-off from a majority of the network. (Section 5.3)

What are the properties of the broadcast trees? Ideally, we want to minimize the distance between the source and leaves of the tree. This will minimize the time spent in moving the data around the network. The minimum distance can be achieved if the broadcast follows the shortest path from the source to any other node. (Section 5.4)

5.1 Broadcast Coverage

The broadcast mechanism can get packets to all nodes that are "connected" to the gradient source. This means that any functional node that has a path to the gradient source, will receive a gradient packet. However, as the defect rate increases, there is an increasing probability that regions of non-defective nodes will be cut-off from the gradient source because of a wall of defective nodes (see Figure 3). Figure 4 plots the percentage of non-defective nodes receiving the broadcast, for a range of defect rates. Each line corresponds to a different network size. Data for limited (10) runs each for networks of 400x400, 500x500 and 800x800 nodes show trends similar to those observed for smaller networks.

As expected, we see that as defect rates increase, the percentage of nodes receiving the broadcast drops because of

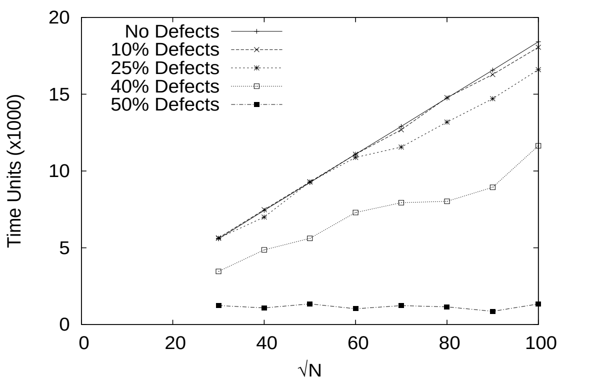

FIGURE 5. Broadcast latency as a function of $\sqrt{NetworkSize}$

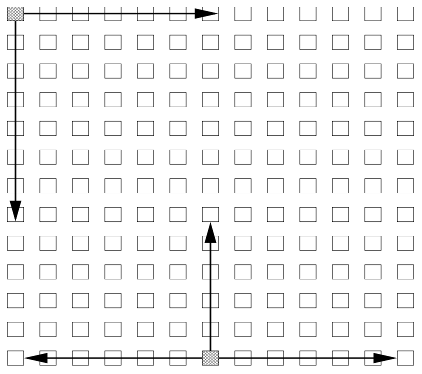

FIGURE 6. Two possible options for gradient sources

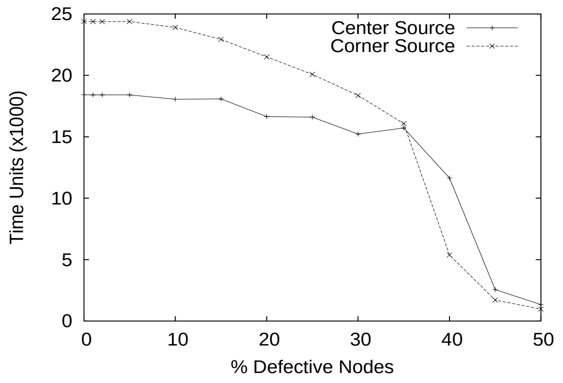

FIGURE 7. Broadcast latency as a function of defect rate and broadcast source

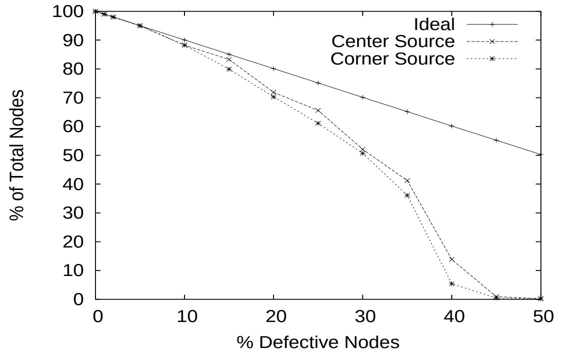

FIGURE 8. Varying Gradient Source:% Reachable Nodes

regions of non-defective nodes being cut-off. In addition, we see that for defect rates up to 20%, the broadcast mechanism typically reaches 90% of the non-defective nodes in the network. This shows that for low defect rates ($\leq$ 20%), the gradient broadcast is a good mechanism for isolating defective nodes and connecting non-defective nodes.

5.2 Broadcast Latency

One of the reasons we choose to use a self-configuring system is to eliminate the time overhead of obtaining an external defect map of the system. However, the gradient broadcast itself takes a non-zero time to complete. If a node can process and forward a gradient packet in unit time, we would expect that it would take at most 2N time units to finish broadcasting in an NxN system (corresponding to the manhattan distance between the nodes in opposite corners). In Figure 5 we plot the time taken to broadcast the gradients as a function of the square root of the number of nodes in the system, for different defect rates. For a system with no defects, we see that the time taken to complete a gradient broadcast is a linear function of the square root of the number of nodes in the system (it is proportional to the maximum distance the broadcast packet has to cover, which for a square network of NxN nodes is N). We see similar trends for larger networks (up to 800x800). As the defect rate increases, we see that the time taken to complete the gradient broadcast decreases. This happens due to the fact that as defect probabilities increase, the probability of isolating a region of non-defective nodes increases. Thus, there are fewer “reachable” nodes in the system, reducing the time taken to complete the broadcast. Indeed, for a system with 50% defects, the time taken to complete the broadcast is almost independent of the number of nodes. This is because, as we see in Figure 4, the broadcast reaches very few nodes.

Our analysis shows that, in general, the latency of the broadcast is directly proportional to the maximum distance a broadcast packet has to cover in the network. This allows us to scale to very large systems and still have a broadcast latency low enough for practical use.

5.3 Changing Broadcast Source

Intuitively, the placement of the gradient source vias in the random network will have an effect on how many non-defective nodes successfully receive a gradient. We run two configurations, one with gradients injected from the corner, and another configuration with the gradient injected from one of the sides of the network grid as shown schematically in Figure 6. The result of this analysis is useful in choosing between the corners and the side midpoints as the source of the four planar gradients.

Figure 7 shows a graph where we compare the two schemes in terms of the time taken to complete a broadcast for a network with 10,000 nodes. From the figure we see that

TABLE 1. Properties of Broadcast Trees (100x100 network)

Defect Rate (%)	# of Nodes	Number of Children				Tree Height	
		0	1	2	3	Max	Avg
0	10000	2430	5192	2329	49	149	75
10	8822	1872	5167	1696	87	146	74
20	7186	1708	3926	1397	155	135	70
30	5203	1409	2553	1075	166	123	65
40	1382	394	641	301	46	93.6	50
50	23.22	6.22	11.8	4.6	0.6	9.2	4.8

for defect rates less than 35%, having a source in the corner takes longer to complete a gradient broadcast than having a source at the midpoint of a side of the grid. This is expected since a broadcast from a corner needs to travel a longer distance to get to all parts of the grid. However, once we have more than 35% defects, the probability of a corner source being cut off is higher than a source on the side being cut off. If a source is cut off from a large part of the network, it will "complete" the broadcast faster, as seen in the figure. In Figure 8 we compare how the two schemes compare in terms of the number of non-defective nodes reached by a gradient. For defect rates less than 10%, the two schemes perform equally well, reaching most non-defective nodes. However, as we increase defect rates beyond 10%, the corner source reaches fewer nodes on average, since it has a higher probability of being cut off due to defects. Our analysis shows that, as expected, the midpoint of a side of the grid is a better choice for the gradient source. A broadcast originating at this source is able to reach a larger fraction of nodes, with lower latency than one originating at a corner.

5.4 Broadcast Tree Properties

The gradient broadcast builds a spanning tree over the graph of all non-defective nodes that are reachable from the source. In most cases, there exist several spanning trees that can be built using the gradient source as a root. In the ideal case, we want a balanced 3-ary tree. However, given our grid-like topology, it is not possible to build a perfectly balanced 3-ary tree. An alternative to a balanced tree would be a tree that minimizes the number of hops between the source of the gradient and any other node in the network (i.e. minimizes the manhattan distance).

We analyze the broadcast trees generated by the gradient broadcast to determine their characteristics. Table 1 shows the results from this analysis on a network with 10,000 nodes. The source of the gradient is the midpoint of a side of the 100x100 square. The average manhattan distance from the source to any other point in the network is 74.5 hops,

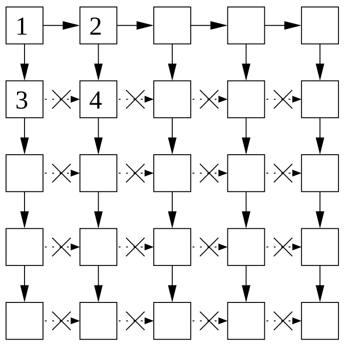

FIGURE 9. Gradient Broadcast: Cause of low branching factor

while the maximum distance is 149 hops. For the case with no defects, we see that the maximum and average height of the tree correspond exactly to the maximum and average manhattan distance between the gradient source and other nodes in the network. This implies that in case of a defect free system, the broadcast finds a minimum-manhattan distance path between the gradient source and any other node. As we increase defect rates, the efficiency of a path from the gradient source to another node decreases. For example, in a network with 20% defects, we see 7,186 nodes in the broadcast tree with an average manhattan distance of 70 hops between the gradient source and other nodes. If we had a square grid with 7,186 nodes (~85x85), the average manhattan distance between the gradient source and other nodes would be 63 hops. This shows that the broadcast can no longer pick the ideal path because of defects, but picks the shortest path that avoids defects.

Another interesting property of the broadcast tree is the branching factor of nodes in the broadcast tree. It is preferable to have nodes with three children as that reduces the distance between the root and the leaves. If we have a large number of nodes with only one child, a failure in the link connecting this node to its child could potentially cut off a large section of nodes (Section 5.5 discusses one possible mechanism to avoid this single point of failure). From the table we see that as defect rates increase, the number of nodes with three children actually increases, which is desirable. At first, this seems counter-intuitive, but is the result of a peculiarity of the broadcast mechanism. As a broadcast packet spreads through the network, it often follows a "preferred" path. We illustrate this phenomenon in Figure 9. As the packet reaches node 1, it is sent to nodes 2 and 3. Now, nodes 2 and 3 both try to send the packet to node 4, however, only one of them (node 2) succeeds in this. All the crossed arrows show broadcasts that are not accepted. This "selection" of one direction over the other has a cascading effect and most nodes end up receiving a particular gradient from the same general direction. However, in the presence of defects, this phenomenon gets disrupted, creating more opportunities for the broadcast tree to branch out. This is also sensitive to the timing of the communication between nodes. If two nodes are not identical, one node will broadcast faster, reducing this problem in systems with low defects.

From our analysis of the properties of broadcast trees presented in this subsection, we conclude the following: a) the broadcast mechanism picks the shortest path consisting of non-defective nodes, but defects often cause the length of this path to deviate from the manhattan distance in a grid, b) defects in the network could improve the average out-degree of nodes in the broadcast tree.

5.5 Extending Gradient Broadcast

Our evaluation shows that gradient broadcast using the RPF algorithm should be an efficient way of achieving defect isolation in large scale systems of self-assembled nodes. If the system has high defect rates, the gradient broadcast scheme can still be used on smaller scales using vias distributed across the network of nodes. By broadcasting a gradient per via, we can establish small “cells” of connected nodes.

The gradient broadcast mechanism presented here has no provision for handling transient or permanent faults during system operation. One simple extension to the current system to handle runtime faults would be to maintain redundant path information at each node. Nodes often get the same broadcast packet on multiple links. The original scheme discards all but the first packet. If we use information from subsequent gradient packets to maintain multiple paths, the system could possibly handle transient faults. In addition, this redundant path information could also be used by higher level protocols for load-balanced routing. There is a trade-off to be made in maintaining multiple paths. Each additional path that needs to be stored requires extra storage at each node. There is also no guarantee that a node will actually receive multiple paths. In addition, path information will need to be periodically refreshed to keep it up to date. This will add to the overhead of gradient broadcast.

Our work assumed a fail-stop defect model for nodes. In a real system, it is far more likely that only a part of a node is defective. Evaluating the performance of gradient broadcast with a node defect model that allows partially defective nodes will be much more complex. However, except in the case of byzantine failures, partially defective nodes will not reduce the effectiveness of gradient broadcasts. It is important to point out that our system assumes the existence of some sort of built-in-self-test (BIST) or external test circuitry to verify node operation. A variation of the BIST would be the ability to inject a test vector packet into the network and have it propagate. Each node would execute the packet and get disabled if it fails the test.

6 Related Work

Recently, there has been a large body of work focused on self-assembly, emerging nano-electronic devices and techniques to tolerate defects in these architectures. We focus here on related work in defect tolerance.

Work in emerging nano-electronic devices has led to an increased interest in defect tolerance. The Teramac [4,9] was one of the first machines built that incorporated defect tolerance in the design. The machine consisted of a large number of defective FPGAs. The defect tolerance mechanism relied on obtaining an external defect map, and then configuring the FPGAs to not use defective regions. Such an approach would not scale to very large systems, as extracting a defect maps would not be feasible. The Nanofabrics [7] work from CMU was similar to the teramac in its use of reconfigurable devices as well as its approach to defect tolerance. Nanofabrics relies on using an external defect map to configure the system.

Another technique is to overprovision the system with “spare” units. If the active unit has a defect, one of the spare units is activated and becomes the primary unit. There are two flavors of sparing. The first, or hot sparing runs the spare units all the time, but does not use answers from them. If the active unit fails, the hot spares have all the state from the active unit and can be directly switched in to replace the faulty unit. The second scheme is cold sparing, where spare units are kept inactive until needed. Molecular electronic systems using array based circuits have proposed using sparing to tolerate defects [6].

Another common approach, which is a special case of hot-sparing, is the use of N-modular redundancy (NMR) [24]. Triple-modular redundancy (TMR) [16] is one commonly used implementation of NMR. TMR and, in general, NMR operates on the premise that it is easier to verify the operation of a simple voting circuit, than it is to verify the operation of a complex computing unit. The system has N (N is 3 for TMR) replicas of the computing unit. The output of all these units is fed into a voting unit. This unit then picks the answer that corresponds to a majority of the N units. NMR works well for defect rates below 50%, beyond which, the voting mechanism cannot reliably decide between the correct and incorrect answers. NMR wastes a lot of computing resources that need to be dedicated to computing the same answer in multiple units.

Han et al. [8] and Nikolic et al. [18] present a comparison of N-modular redundancy, NAND multiplexing, cascaded TMR and other approaches to using redundancy for defect tolerance. Our approach differs fundamentally from theirs in that we are trying to isolate defects from the active parts of the system while their approach compensates for defects in active parts of the system.

7 Conclusions

Self-assembly may enable the construction of large scale systems with more than a trillion processing elements. However, self-assembled systems will need to be designed to include mechanisms for defect tolerance. In this paper, we have presented one mechanism to achieve defect tolerance in a system that is composed of up to 10^{12} processing elements. We have adapted the reverse path forwarding broadcast routing algorithm for use in a self-assembled network of nodes. We have shown that our mechanism can isolate defects in a system and create a broadcast tree that connects most of the functional nodes. We have also presented an analysis of the connectivity of such a network of self-assembled nodes. Our mechanism can be extended to include multiple paths, thus providing robustness in the face of runtime faults. This extension involves a tradeoff in terms of the storage required at each node, and the desired path redundancy.

Acknowledgments

This work is supported by an NSF ITR grant CCR-0326157, a grant from the Duke University Provost's Common Fund, an AFRL contract FA8750-05-2-0018, and a Warren Faculty Scholarship (Sorin). We thank the members of the TROIKA project for their help with this work.

References

[1] A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker. Logic Circuits with Carbon Nanotube Transistors. *Science*, 294:1317–1320, November 2001.

[2] E. Braun, Y. Eichen, U. Sivan, and G. Ben-Yoseph. DNA-Templated Assembly and Electrode Attachment of a Conducting Silver Wire. *Nature*, 391:775–778, 1998.

[3] Y. Cui and C. M. Lieber. Functional Nanoscale Electronic Devices Assembled Using Silicon Nanowire Building Blocks. *Science*, 291:851–853, February 2001.

[4] W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider. The Teramac Custom Computer: Extending the Limits with Defect Tolerance. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, November 1996.

[5] Y. K. Dalal and R. M. Metcalfe. Reverse Path Forwarding of Broadcast Packets. *Communications of the ACM*, 21(12):1040–1048, 1978.

[6] A. DeHon. Array-Based Architecture for Molecular Electronics. In *Proceedings of the First Workshop on Non-Silicon Computation (NSC-1)*, February 2002.

[7] S. C. Goldstein and M. Budiu. NanoFabrics: Spatial Computing Using Molecular Electronics. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 178–191, July 2001.

[8] J. Han and P. Jonker. A Defect- and Fault-Tolerant Architecture for Nanocomputers. *Nanotechnology*, 14:224–230, January 2003.

[9] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology. *Science*, 280:1716–1721, June 1998.

[10] Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K. Kim, and C. M. Lieber. Logic Gates and Computation from Assembled Nanowire Building Blocks. *Science*, 294:1313–1317, November 2001.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[12] International Technology Roadmap for Semiconductors, 2003.

[13] K. Keren, R. S. Berman, E. Buchstab, U. Sivan, and E. Braun. DNA-Templated Carbon Nanotube Field-Effect Transistor. *Science*, 302:1380–1382, November 2003.

[14] R. A. Kiehl, K. Musier-Forsyth, N. C. Seeman, B. I. Shklovskii, and T. Andrew Taton. Assembly of Nanoelectronic Components by DNA Scaffolding. In *Proceedings of the NSF Nanoscale Science and Engineering Grantees Conference*, December 2003.

[15] D. Liu, S-H. Park, J. H. Reif, and T.H. LaBean. DNA Nanotubes Self-assembled from TX Tiles as Templates for Conductive Nanowires. *Proceedings of the National Academy of Science*, 101(3):717–722, 2004.

[16] R.E. Lyons and W. Vanderkulk. The Use of Triple-Modular Redundancy to Improve Computer Reliability. *IBM Journal*, pages 200–209, 1962.

[17] B. R. Martin, D. J. Dermody, B. D. Reiss, M. Fang, L. A. Lyon, M. J. Natan, and T. E. Mallouk. Orthogonal Self-Assembly on Colloidal Gold-Platinum Nanorods. *Advanced Materials*, 11(12):1021–1025, August 1999.

[18] K. Nikolic, A. Sadek, and M. Forshaw. Fault-Tolerant Techniques for Nanocomputers. *Nanotechnology*, 13:357–362, 2002.

[19] J. P. Patwardhan, C. Dwyer, A. R. Lebeck, and D. J. Sorin. Circuit and System Architecture for DNA-Guided Self-Assembly of Nanoelectronics. In *Foundations of Nanoscience: Self-Assembled Architectures and Devices*, pages 344–358, April 2004.

[20] N.C. Seeman. DNA Engineering and its Application to Nanotechnology. *Trends in Biotech*, 17:437–443, 1999.

[21] S.J. Tans, A.R.M. Verschueren, and C. Dekker. Room-temperature Transistor Based on a Single Carbon Nanotube. *Nature*, 393:49–52, 1998.

[22] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *Computer Communication Review*, 26(2), 1996.

[23] J. M. Tour. Molecular Electronics. Synthesis and Testing of Components. *Accounts of Chemical Research*, 33(11):791–804, 2000.

[24] J. von Neumann. Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, Princeton, NJ, 1956.

[25] H. Yan, S. H. Park, L. Feng, G. Finkelstein, J. H. Reif, and T. H. LaBean. 4x4 DNA Tile and Lattices: Characterization, Self-Assembly, and Metallization of a Novel DNA Nanostructure Motif. In *Proceedings of the Ninth International Meeting on DNA Based Computers (DNA9)*, June 2003.

[26] H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and Thomas H. LaBean. DNA Templated Self-Assembly of Protein Arrays and Highly Conductive Nanowires. *Science*, 301(5641):1882–1884, September 2003.