Ordered Chaos: Imposing Logical Structure in Nanotechnology Substrates

Derek R. Hower
Duke University
Department of Electrical and Computer Engineering
Email: drh5@ee.duke.edu

April 25, 2006

Abstract

We expect emerging nanotechnology substrates to consist of a vast number of small processing nodes that are randomly assembled, have defect rates as high as 30%, and have globally asynchronous communication. Directly constructing an architecture on such a substrate presents many challenges to a system designer. An alternative approach, that would ease the constraints faced by a nanotechnology architect, would be to provide a layer of abstraction between the physical substrate and the architectural design space that logically structures the system components.

We propose an abstraction layer that can impose a defect-free logical grid on randomly oriented and randomly connected nodes. This layer is based on the observation that logical neighbors in a network need not be constrained to physical neighbors but instead can be connected through a series of communication hops. Our distributed ordering algorithm guarantees that all nodes reachable from a micro-world connection node will be included in the logical grid.

1 Introduction

The era of Moore's Law and the exponential increases in CMOS device density that accompanies it is quickly reaching an end. Both economic and technological roadblocks stand in the way of this trend that we have enjoyed for nearly half a century. In response to the coming "red brick wall" predicted by the ITC roadmap, researchers have been investigating new devices fabricated on the nanometer scale. Such devices are likely to be created using bottom-up self-assembly (e.g., using DNA), and as such lack the exacting precision employed in top-down lithography.

In this work, we assume that a nanotechnology substrate is composed of a vast number of small processing *nodes* connected in a network (e.g., [1]). Because self-assembly cannot

produce large complex designs, each node has a very limited amount of storage and some limited computational ability (e.g., a 1-bit ALU). Nodes are placed and connected in an arbitrary topology network. Such a physical substrate leaves system designers with the daunting task of implementing a system that can perform in the presence of massive defect rates, random interconnections, and limited local computational and storage resources. The potential reward, however, is great, since these devices can be created in previously unheard of densities and at a fraction of the cost of a current CMOS process.

Few algorithms or computational paradigms have been designed that can operate on such a substrate. As such, our goal in this work is to create a logical structure within the system that serves as an abstraction layer between the physical substrate and the architecture that will ease the burden on system designers. We identify the following as desirable characterstics that a logical structuring scheme in a nanotechnology substrate should possess:

- **high utilization** Any structuring scheme should be able to include as a many defect-free devices as possible in order to reap the benefits of a nanotechnology substrate.
- **low implementation cost** As nodes in a nanotechnology substrate are expected to possess limited computational power and storage capacity, any logical structuring mechanism must require no more than a few bits of storage and simple logic.
- **low network contention** The logical structure should not place unnecessary stress on the communication network.

Our contribution in this work is a viable scheme for providing structure on a nanotechnology substrate. Specifically, we provide the abstraction of a fully-populated, two-dimensional grid of nodes as a logical overlay to the substrate. A grid is uniform and easy to program. Indeed, many preexisting computations are tailored to grid structures such as cellular automata and matrix operations. We also provide an evaluation of our scheme based on the above criteria and find that our scheme achieves the maximum utilization possible, has a fraction of the implementation cost that is found in previous logical ordering schemes for CMOS substrates, and can achieve low network contention by enforcing scheduled communication and by limited certain system parameters.

The remainder of this paper is organized as follows. Section 2 describes our fairly generic system model and outlines the minimum resources that we assume present in the system. Section 3 discusses related work. Section 4 describes our logical ordering technique in detail, and Section 5 discusses routing. Section 6 presents an evaluation of our approach, and we conclude in Section 7.

2 Target Substrate

While our proposed technique is designed to be as general as possible, there are certain basic system characteristics that are assumed. A target system is composed of many small

processing nodes that are randomly connected to each other via at least a single-bit wire. I/O communication occurs by way of any number of *anchor nodes* that are connected to *vias* in the micro-scale world. A single anchor node connects to a single micro via. We make no assumptions about node orientation, connectivity, placement or quantity of anchor nodes, or the nature of the vias that connect to the nanocomputing substrate. We model all defects in the system as fail-stop, such that any defect will cause a node to become unresponsive.

3 Related Work

Most previous work on arbitrary topologies has been developed in the context of a CMOS system with vastly more per node resources that those expected to be present in nanocomputing. Schemes to impose structure within local and wide area networks that are physically connected in unordered meshes are especially prevalent in the literature. Two such schemes in particular,the MetaNet [2, 3] and AutoNet [4] architectures, resemble the technique we propose in that they embed logical trees and rings over the physical network. However, because the LAN architectures were designed with different goals and for a system with vastly more local resources, the resulting architectures contained many attributes that make them infeasible in a nanotechnology network.

The work in arbitrary topology LANs was motivated by the desire to both provide structure in a network and, equally important, to maximize bandwidth. As such, the techniques they used were designed to dynamically reduce contention in the network, which affected the manner in which they structured and routed the nodes. For example, the complex routing algorithms in both architectures relied on the presence of large forwarding tables and unique node identifiers. Due to the limited storage in a nanocomputing node, it would be unreasonable to assume that routing tables could be present. This is especially true considering that destination IDs would have to be at least 40 bits in order to uniquely identify all the nodes in a network, assuming a fabrication process could produce 10^{12} nodes as reported in [1].

Other research in CMOS systems has been performed with the goal of implementing a system that performs correctly in the presence of massive defect rates. The Teramac multiprocessor machine built at Hewlett-Packard Laboratories was constructed using a mix of defective and fault-free FPGA components. The researchers were able to build a system that contained over 220,000 defects yet (correctly) performed 100 times faster than a high end uniprocessor [5]. This achievement proved that it was possible to build a working system in the presence of massive defects, although its direct application to bottom-up nanotechnology fabrication is limited because the authors rely on an a priori knowledge of fault mappings in the FPGAs in order to configure the components in a logical topology, which is a luxury unlikely to be afforded in emerging nano-scale devices.

Patwardhan et al. [6] addressed the need for imposing structure on a nanocomputing substrate similar to the target outlined in this work by adapting the reverse path forwarding

algorithm [7] to provide gradient routing. While their method was successful in providing a loose sense of direction for packets within the system, no absolute or deterministic structure was created. Useful work in the system relied on the ability of active network packets to find free processing or memory elements. The proposed scheme is much more general, allowing for either homogeneous or heterogeneous nodes in the system and providing a uniform, defect free logical structure on which to develop an architecture.

4 Distributed Logical Structuring Technique

It is the goal of this work to implement an ordering technique that has most of the features provided by previous arbitrary topology structuring schemes designed for CMOS substrates, yet is still implementable in a nanotechnology network. Such a technique needs to be distributed, have minimal storage requirements, and be scalable to extremely high node densities.

Our logical structuring technique is a three stage process that progressively transforms the substrate into a logical tree, a ring, and finally into a fully populated two-dimensional grid. The first two of these steps are similar to previous work in principle but not in implementation. In this section, we present our three stage process for imposing order, and we describe how to route on this ordered network.

4.1 Step 1: Spanning Tree

The first step in our structuring technique is to create one or more spanning trees out of the physical network. Considering the network as a graph, where nodes are vertices and links are edges, this is done by performing a breadth-first traversal of the graph at each anchor node. It has been proven that a bread-first traversal will include all nodes in the network that are reachable by an anchor node [8]. As a result, the proposed technique will maximize node utilization throughout the network.

A breadth-first traversal algorithm can be implemented efficiently in a nanotechnology substrate by introducing a directional pointer and one bit of storage in each node. Tree creation begins when a *tree-ring* packet is injected into the network that indicates that the tree formation procedure should proceed. When a node receives the tree-ring formation packet, it marks the 1-bit indicator to remember that it has already been included in the network traversal, sets the pointer to the parent node by remembering on which internal transceiver the packet was received, and then broadcasts the signal to its remaining neighbors. For reasons we will discuss shortly, the newly added child also sends an acknowledgment to the parent indicating that it has just been inserted into to the tree. Nodes that have already been initialized ignore incoming initialization packets, and as a result the procedure will terminate when all nodes reachable from an anchor have been found. An example of an embedded spanning tree is shown in Figure 1a.

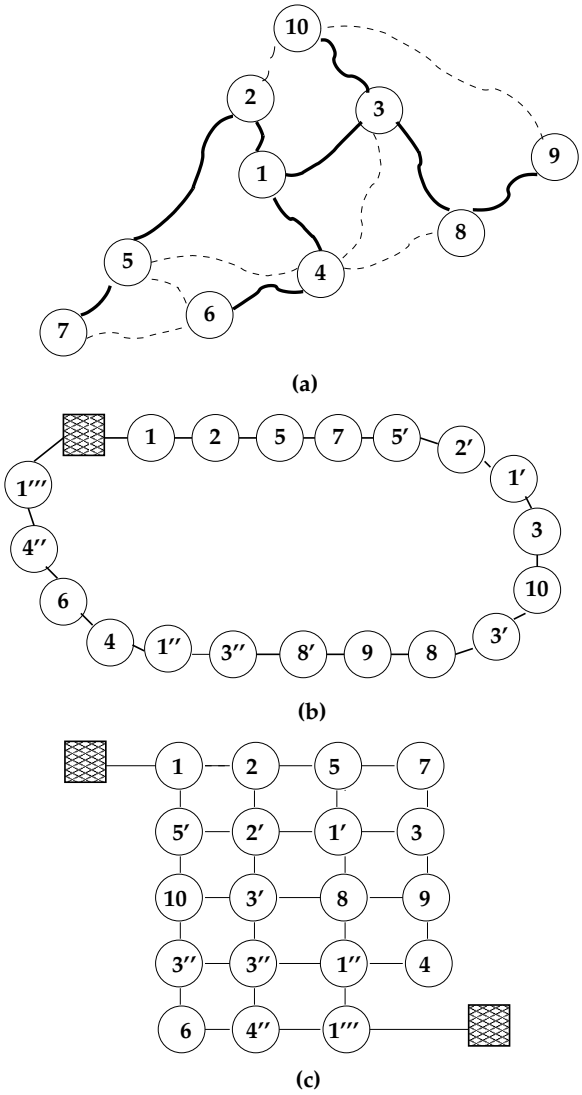

Figure 1: **An example of the three stage ordering process.** (a) An example network with the embedded tree rooted at node 1. Dashed lines show the connections not involved in the logical tree. (b) The resulting ring. Different virtual nodes are indicated by primes. The shaded box represents the micro via that interfaces with the nanocomputing network. (c) The resulting grid.

4.2 Step 2: Ring

The second step is to embed a ring on each of the anchor-rooted spanning trees by performing a depth-first traversal. The traversal progresses by proceeding deeper into the tree whenever possible and backtracking otherwise. Because of the backtracking, the path that a traversal takes around the tree forms a closed loop that begins and ends at the root node. Any node that is not a leaf is in the tree is visited multiple times, and is thus located at several positions along the closed loop path. So that we may map our ring along the traversal path, we adopt the concept of *virtual nodes*. Each time the traversal touches a node, a new virtual node is created that acts as the node's representation at that point in the ring. In general, the maximum number of virtual nodes needed is equal to the maximum number of connections that a node can make (which is a function of the specific nanotechnology substrate).

At the same time that the tree is being created, the virtual nodes of the embedded ring can also be established by deterministically assigning virtual ring connections based on the structure of the spanning tree. Child nodes of a parent in the spanning tree are given a logical position based on the order in which acknowledgments are received, so that the first child to respond becomes child one, the second becomes child two, and so on until all children have acknowledged. Using the child ordering as a reference, the *east* and *west* connections of the ring are assigned such that the connections of successive virtual nodes follow the progression of parent-to-first child, first child-to-second child, ... last child-to-parent.

For example, in Figure 1b, the east connection of the first virtual node of node 2 is connected to its parent (node 1) and the west connection is connected to its first child (node 5). The second virtual node (node 2') is created between the first child (node 5) and the parent (node 1). If node 2 had a second child in the tree, then the process would be repeated so that node 2's second virtual node made a connection between its first and second child, and the third virtual node would be connected to the second child and the parent. In Figure 1b, such a progression can be seen in node 3. Using this ordering technique to assign virtual ring connections assures that the ring will always follow in the direction of a depth-first traversal.

To implement this concurrently with the tree formation process, ring connection pointers within a node are adjusted each time a local connection in the tree is established. The first virtual representation of a physical node is created when a node adds itself as a child in the spanning tree. This virtual node sets both the east and west pointers to the parent because at this point in the process the node appears to be a leaf in the spanning tree and as such has only one path, the link to the parent, on which to communicate. When and if a node receives an acknowledgment from one of the neighbors that it forwarded the tree formation packet to, a new virtual node is created that has an east pointer set to the acknowledging node and a west pointer set to the parent. Additionally, the west pointer of the previously created virtual node is changed to point to the acknowledging neighbor node. The final result is a ring that starts and ends with the micro via that is connected to an an-

chor node (because it is the logical parent of an anchor node). Because this via is included in the ring, we have maintained the ability to insert and remove data from the network.

4.3 Step 3: Rectangular Grid

The third and final step in the ordering technique is to fold the logical ring into a rectangular grid. Of the four logical connections needed by an element in a uniform rectangular grid, two of these (east and west) have already been established by the ring formation process. The remaining two connections can be established by abstracting *north* and *south* neighbors as N hops east or west, respectively, along the ring. Using uniform distance and direction hops for north and south neighbors has the effect of slicing the ring into lines of length N and stacking them on top of each other as shown in Figure 1c.

There are two issues that must be resolved when using the ring folding technique described above. First, the nodes on the edges of the grid must be informed that they have only a subset of the four possible neighbors. Second, unless the number of nodes in the ring is a multiple of N, there will be excess nodes in the ring that will form an incomplete line in the final grid row. The first of these problems can be solved by introducing a *grid-formation* packet into the system that traverses the ring and structures the grid.

A data value in the grid formation packet will be incremented each time the packet is received in a new node. When this value exceeds N, it will be reset to 1. Nodes that receive the packet when the data is set to 1 mark themselves as being on the left side of the grid and thus contain no west neighbor. Likewise, nodes receiving the packet when the data is set to N mark themselves as being on the right side of the grid with no east neighbor. Eventually, the micro via connected to the anchor node will receive the packet from the last virtual node in the ring. When this happens, the data in packet represents the number of nodes E that are excess in the last grid row. A final initialization packet, the *forward-only* packet, can then be inserted in the opposite direction of the ring that the grid-formation packet was sent. It marks the first E nodes as forward-only and is then sunk in the network, effectively eliminating any nodes that form an incomplete last row from the logical grid. .

The north-most and south-most rows need no extra configuration because their north and south neighbors, respectively, will be routed through the micro via. The decision of what to do with that data can be made based on the operation being run in the network. If the data was forwarded through the micro via, then the grid would roll into a cylinder such as those used in several cellular automata applications [9]. If the via instead sinks the data, the grid will remain flat.

Note that at no point during the ordering process does any part of the system require global knowledge of the logical structures. The entire algorithm is distributed and requires minimal storage, which will be quantified in Section 6..

5 Routing

In our structuring scheme, the basic default routing is deterministic within the logical network. A node may only communicate with its immediate virtual neighbors north, south, east, and west by traveling the ring N hops west, N hops east, 1 hop east, or 1 hop west, respectively. When a packet travels through the network it must specify both the ring direction it is traveling and the number of hops remaining to its destination. Routing thus needs only to support directional movement along the ring, making it unnecessary to maintain individual node IDs or complex routing tables as is done in previous work.

5.1 Routing Optimization

The static routing scheme above is simple enough to be viable in our target system, but it sacrifices some performance. We attempted to minimize the performance penalty induced by the default routing mechanism by investigating several routing optimizations. Ideally, we would like to reduce the impact of the high network contention while concurrently maintaining the simplicity afforded by the default routing.

5.1.1 Routing Through Shortcuts

Within the logical network, links between two virtual neighbors are at most N hops away. Because the depth first traversal of the spanning tree that created the logical ring backtracks and because all routing occurs along the ring, one would expect that the physical distance between two nodes is often much less than N. We refer to routing optimizations that attempt to exploit this observation as shortcut routing.

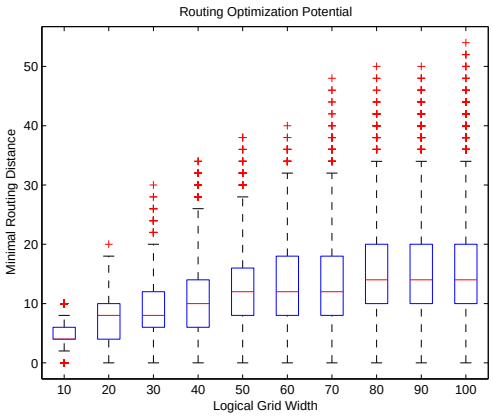

Figure 2: **Routing Optimization Potential.** A boxplot of minimum inter-node routing distance for varying logical grid widths (N).

We evaluated the potential of shortcut routing optimizations by finding the minimum number of physical hops between two logically separated neighbors. In our evaluation, we assume that nodes in the nanotechnology substrate can make up to four connections with neighboring nodes and that there are 1000 nodes with 1 anchor in the network. The devices are modeled after those found in [6], although this choice was arbitrary and does not impact the results. Figure 2 shows that the benefits are significant when logical separation is large. For a logical separation of 100, which corresponds to north and south neighbor communications in a logical grid with $N = 100$, the average

physical distance between nodes is less than 15 hops and the longest physical separation observed was only 54 hops. Because the maximum physical distance between two logical neighbors is significantly less than the logical distance, the data confirms that it is common for the static routing technique to traverses the same physical link multiple times in a single logical communication and indicates that there is a strong possibility for routing optimizations.

We separate shortcut optimizations into two categories. The first is *physical link forwarding* and the second is *virtual node forwarding*. Physical link forwarding involves shortening the routing path by following a path of physical routing directions between logical neighbors. These directions do not necessarily correspond to the links involved in the logical network, and can include the links identified by dashed lines in Figure 1a. Virtual node forwarding occurs when a packet internally hops from one virtual node to another. For example, in Figure 1b, a packet that needed to travel from 5 to 2' could be forwarded from 5 to 5' and then proceed to 2'.

Application of physical link forwarding requires the addition of routing tables and extra routing logic in the nodes. The total storage needed to route through shortcuts depends on the maximum number of connections that a node can make (C) and the maximum distance that the shortcut can span (D). In general, a C-entry table will be needed with $2Dlog_2(C)$ bits per table entry. The factor of two is included because shortcuts for both north and south directions must be maintained.

While a table of any non-trivial size would most likely be beyond the capabilities of a nanotechnology node, we considered the effects of keeping a 2-entry table ($D = 1$) for shortcuts 1 hop away. Individual entries would correspond to logical north and south neighbors, respectively, and would be used to indicate when a logical neighbor was 1 hop away. Such a scheme would keep routing simple and storage minimal. Figure 3 shows the distribution of the minimum physical link distances between two logical north and south neighbors in a grid of width 20. Similar results were obtained for grids of larger width, but were omitted due to space constraints.

The data indicates that a 2-entry table would not yield a significant improvement in routing, as the percentage of logical neighbors that are less than two hops away is small. Moving beyond a one hop shortcut is difficult, as the complexity of routing logic grows sharply because the routing no longer follows the path of the ring but instead must be routed along physical node directions. Packets traversing the network would have to carry with them a list of physical directions that indicate the path to take at each node along the route. Thus, while the potential of physical link forwarding is great, the complexity it introduces is not reasonable in a nanotechnology substrate.

The effects of virtual node forwarding were also studied. We found that, on average, a 10% reduction in path length could be achieved when packets were allowed to forward through virtual nodes. Implementing this optimization would require that a register of size log_2N bits, where N is the logical grid width, be kept for each virtual node. This register would indicate the number of hops that could be skipped by a jump from one virtual node to the next and would be used to calculate the shortcut distance for a packet traversing the

ring. Additional logic would also have to be added to the nodes so that the forwarding paths could be determined during initialization. Based on this data, we conclude that virtual node forwarding is most likely not worth the cost of implementing.

5.1.2 Scheduling

Depending on the specific architecture running on the logical grid, it may be possible to optimize routing by enforcing scheduling rules in the network. For example, if a cellular automation was running in the network, neighboring communication could be synchronized such that all concurrent inter-node communication occurred in a single direction along the ring. In a cellular automata program, the state of a node on the next logical time step depends on the state of its neighbors in the current time step. Thus, every node in the system must communicate with all of its neighbors before moving to a new state. If this communication is synchronized, such that all nodes communicate to their north, south, east, and west neighbors at the same time, then contention in the network would be eliminated because all existing packets in the network will move in the same direction along the ring without passing one another.

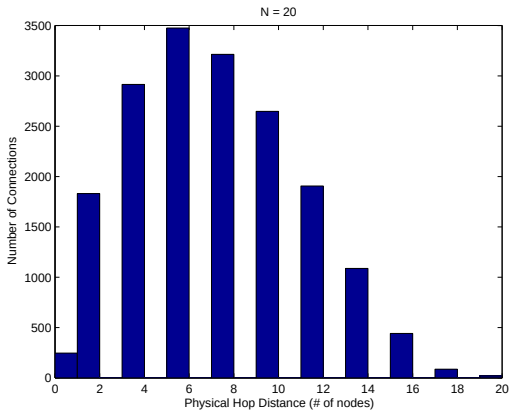

Figure 3: **Histogram of physical node distances**
The distribution of minimal physical distance between logical neighbors in a grid. Odd distances are absent due to the nature of routing through a tree. The percentage of logical neighbors that are only one physical hop away is small.

Such a scheduling scheme can be enforced in a globally asynchronous substrate by issuing a total order of communication (i.e. north first, east second, etc.) and by mandating that the number of packets sent by a node is never more than one plus the number of packets received by the node.

Other scheduling optimizations may be possible, but they are dependent on the overlaying architecture and as such are left to the system designer.

5.2 Deadlock

For any network to be useful, it must be able to avoid routing deadlock. Because of the logical structure that we have already added to the network, we can use up/down routing that has been proven deadlock-free [4]. Because the tree structure is built on top of individual physical nodes, it can be used to assign the node degrees required in up/down routing to physical links In such a scheme, communication with a parent node would be considered

up routing and communication with all children would be down. Fetch deadlock, or request/response deadlock, is a function of the execution model running in the network, and as such is a problem left to the designer of the overlaying architecture.

6 Evaluation

In this section, we evaluate our logical ordering scheme based on the three characteristics identified as desirable for a logical structuring scheme: high device utilization, low implementation cost, and low network contention.

6.1 Utilization

Cormen, et.al. [8] proved that a breadth first search of an undirected graph will touch every node reachable from the root. As such, our technique is optimal given the constraints of the target substrate. However, specific parameters of the target substrate will affect the amount of utilization our scheme is able to extract. To evaluate the design, we found the node utilization for varying link connectivities, measured as the percentage of available links that are defect free in the system. Figure 4 shows that for connectivities above 70%, our scheme is able to extract a nearly 100% utilization. For connectivities below 50%, the nodes within the network are too sparse and the anchor node is not able to reach more than a few neighboring devices.

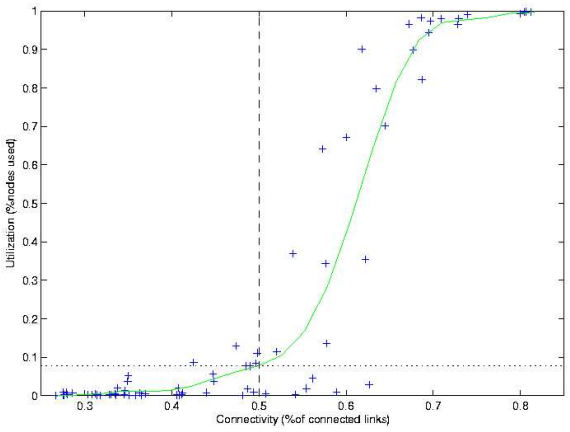

Figure 4: **Utilization of Nodes vs. Node Connectivity.** A system of 1000 nodes and 1 anchor was evaluated to determine the degree of node utilization in the logical network.

6.2 Implementation Cost

The goal of our design is to provide logical ordering with minimal computation and storage costs. In this section, we examine the costs incurred by our scheme.

The logic to handle the three initialization procedures must be added in the nodes. The tree-ring formation packet requires the ability to broadcast a message to all the node's neighbors and send an acknowledgment to the parent. The grid-formation packet and forward-only packet mandate that a small amount data can travel along with a packet. We claim that neither of these system constraints are beyond the basic functionality that would be required in any useful network, whether or not our logical ordering technique was used, and as such are justifiable complexities in the system.

In addition to the logic complexity needed for the proposed technique, a certain amount of storage must also be present in the system. To support virtual nodes, we must replicate anything considered to be architected state in the node, thus resulting in Va bits of overhead, where V is the maximum number of virtual nodes allowed and a is the number of bits of architected state. While this may seem like an unreasonable requirement given the limited amount of resources, our initial node designs indicate that the architected state will be dwarfed by the logic for inter-node communication and is thus only a small percentage of the total inter-node communication.

Each node must maintain $2V + 1$ pointers, where *V* is the maximum number of virtual nodes in the system. Each virtual node must maintain a left and right pointer that indicate neighbors in the logical ring. Another pointer is also required to remember the direction to a node's parent in the spanning tree. Pointers can be stored with a minimal amount of state, as they need only indicate which internal transceiver to use in a given communication.

For deadlock avoidance using up/down routing, we must also add two virtual channels to each node transceiver. Each virtual channel would need at least a one entry send and receive queue. The size of the buffer would depend on packet sizes, and is thus a function of the high level system architecture.

The largest overhead incurred by the proposed scheme is due to the *N* hop counting that must be supported in packet communication. A buffer of size $log_2 N + k$, where N is the logical grid width and k is a constant amount of bits that comprise the packet header, tail, and data that are functions of the overlying architecture, must be present in each NI of each virtual channel within a node. Additionally, each node must have the ability to subtract a $log_2 N$ number, although a bit-serial adder can be used to minimize the overhead required to do so. Our estimates of the capabilities of nanotechnology nodes indicates that the choice of N must be kept small for the proposed scheme to feasible.

Compared to previous ordering techniques in nanotechnology architectures, our implementation costs are slightly higher. However, the increased overhead provides a higher degree of ordering the nanotechnology substrate than has previously been available. Patwardhan et al. [1, 6] created logical structure by establishing five gradients in the network, one for each global direction north, south, east, and west, and one to the micro via connector much in the same manner that the tree formation packet operates in the proposed technique. Routing is based on an active network architecture in which packets in the network use the gradients as a compass to find resources scattered throughout the network. Thus, to enforce structure and routing in the RPF architecture, a node needs 5 direction pointers, two virtual channels, a k bit buffer at each NI, and routing logic to move packets through the active network.

To quantitatively compare the proposed scheme to the previous work, we developed a simple analytical model to evaluate area overhead. Our model is based on the following parameters:

- **a** - The number of bits of architected state in each node.
- **C** - The number of connections a node can make. Also equal to V, the maximum

number of virtual nodes allowed.

- **N** - The width of the logical grid in the proposed scheme.
- **k** - The number of bits of overhead in a communication packet (i.e. the header)

Equation 1 is the analytical model used for the proposed sheme. It represents the storage overhead costs, in number of bits, as described above. We omitted the overhead of the additional logic complexity used by our mechanism because the specific overhead numbers are largely implementation dependent and are similar to the prior work we are comparing to. Equation 2 is a corresponding analytical model for the prior work that uses an RPF algorithm. Figure 5 gives the ratio of these models ($storage_{proposed}$ / $storage_{prior}$) as N and a are varied while C and k are kept constant at 4 and 8 respectively. As intuition would suggest, the value of N has the greatest impact on overhead. However, at small values of N the relative overhead is kept low.

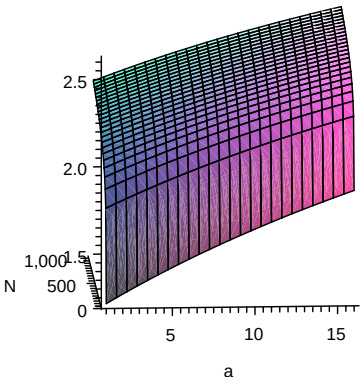

Figure 5: **Comparison of Overhead Cost** The X axis is bits of architected state (a), the Y axis is the logical grid width (N), and the Z axis is the relative increase in the bits of storage needed for the proposed scheme over that needed by the RPF algorithm previously published

$$storage_{proposed} = Ca + (2C+1)log_2C + 2(C+1)(log_2N + k) \quad (1)$$

$$storage_{prior} = a + 5log_2C + 2(C+1)k \quad (2)$$

6.3 Network Contention

The amount of contention in a network is highly dependent both on the physical parameters of the substrate and the characteristics of the overlaying architecture. A list of variables that determine the contention in a network include:

- **connectivity** - as link defect rates increase, a higher number of logical links will be mapped over individual physical links.

- **width of logical grid** - logical paths become longer as the grid size increases, which elongates the amount of time a communication packet spends in the network.

- **communication latency** - the longer it takes for a packet to transmit, the less time a link is free to accept a new communication

- **computational latency** - packets that take longer to operate on decrease contention because communication rates are decreased.

- **average source and sink rates** - these architecture specific parameters impact the amount of active packets present in a network at any given time.

Traditional measurements of network contention would be meaningless in the absence of a well defined architecture or device parameters. Therefore, we quantify network contention as the number of logical hops per utilized physical link. This metric captures the effect of the logical network on contention by indicating how frequently independent communications will be forced to cross paths in the physical network. Furthermore, it is a function of the only variable listed above that is dependent solely on the logical structure, the grid width, and is therefore ideal for capturing the performance consequences of the proposed technique without having to account for other parameters that are determined by system designers.

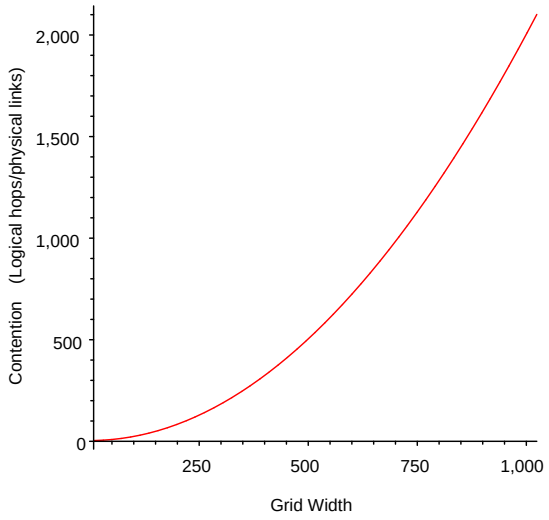

Figure 6: **Evaluation of Network Contention**
Contention (evaluated as logical hops per physical link) vs. logical grid width

Figure 6 shows the contention metric for varying grid widths. As shown, the network contention induced by the logical structure grows exponentially with grid width. While this result is not promising, it must be taken in context. The metric is only meant to reveal the worst case contention for an architecture and physical implementation that communicate frequently and randomly. Any architecture in which communication can be scheduled or where communication is relatively infrequent will perform orders of magnitude better than Figure 6 indicates. However, the graph is useful in that it indicates that the grid width in a network should be kept small to minimize contention. When N is large, even a small reduction in grid width will yield significant performance enhancements.

7 Conclusions

In this paper, we have presented a viable scheme for imposing ordering on a the type of nanocomputing substrate that we expect to emerge. With self-assembled systems, we will be restricted to small nodes (albeit vast numbers of them) and random interconnections. We have shown a distributed algorithm for structuring this type of substrate into a grid that can be more easily used for computation. Our scheme has similar goals to prior work in logical topologies, but our contribution is in greatly reducing the storage and complexity so as to make this approach feasible for nodes with limited resources.

Our results were not as promising as we would have liked, but with intelligent scheduling the performance penalties could be managed. Further study needs to be done on methods to exploit the routing optimization potential in the logical network. Our estimates of the capabilities of future nanotechnology substrates indicate that logical grid width must be kept small so that the design can be implemented with small enough costs.

Acknowledgements

I would like to thank Dr. Daniel Sorin and Dr. Chris Dwyer for their contributions in this research.

This work was sponsored by the Pratt Engineering Fellows program and in part by a grant from the Intel corporation.

References

[1] 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 (FNANO)*, 2004.

[2] Y. Ofek and M. Yung, "Metanet: Principles of an arbitrary topology lan," in *IEEE/ACM Transactions on Networking*, pp. 169 – 180, 1995.

[3] B. Yener, Y. Ofek, and M. Yung, "Configuration and performance issues in the metanet design," in *International Conference on Local Computer Networks*, pp. 291 – 299, 1993.

[4] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker, "Autonet: a high-speed, self-configuring local area network using point-to-point links," *Selected Areas in Communications*, vol. 9, pp. 1318 – 1335, October 1991.

[5] J. R. Heath, P. J. Kuekes, G. S. Snider, and S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, pp. 1716 – 1721, June 1998.

[6] J. P. Patwardhan, C. Dwyer, A. R. Lebeck, and D. J. Sorin, "Evaluating the connectivity of self-assembled networks of nano-scale processing elements," in *International Workshop on Design and Test of Defect-Tolerant Nano-scale Architectures (NANOARCH)*, May 2005.

[7] Y. K. Dalal and R. M. Metcalfe, "Reverse path forwarding of broadcast packets," *Communications of the ACM*, vol. 21, no. 12, pp. 1040–1048, 1978.

[8] T. H. Cormen, C. E. Lieserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2001.

[9] E. Jen, "Cylindrical cellular automata," in *Communications in Mathematical Physics*, vol. 118, pp. 569–590, 1988.