MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Technical Report No. 1224 February, 1990

Fat-Tree Routing for Transit

André DeHon
andre@ai.mit.edu

Abstract: As an alternative to using a bidelta network topology for large Transit networks, I consider the requirements to extend the base Transit network into Leiserson's Fat-Tree configuration. Transit will be a high-speed, low-latency, fault-tolerant network interconnection for high performance multi-processor computers. The initial interconnect scheme planned for Transit will use a bidelta style network to support up to 256 processors. Scaling beyond 256 processors by simply extending that network topology will result in a uniform degradation of network latency across processors. A fat-tree network structure will allow the Transit network to be scaled arbitrarily while taking advantage of the locality and universality of fat-trees to minimize the impact of scaling upon network latency. I consider the topology and construction issues for integrating the Transit routing network component and technology into a fat-tree configuration. I also characterize the resulting network's size, locality, and performance and compare these characteristics with those of bidelta networks.

Acknowledgements: This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence Research is provided in part by the Advanced Research Projects Agency under Office of Naval Research contracts ONR N00014-88-K-0825 and N00014-85-K0124.

Acknowledgments

The Transit project was conceived and started by Tom Knight with the help of Henry Minsky. They are responsible for most of the interesting ideas that Transit brings together. Knight originally suggested the concept of using fat-tree structures to interconnect Transit bidelta stacks.

Knight and Minsky's role in this development cannot be understated. All throughout the design and analysis of these fat-tree networks, they were continually willing to discuss the developing ideas. Their feedback and suggestions were invaluable.

Thanks to Pat Sobalvarro for his work on fleshing out the probabilistic models for routing statistics. He was especially helpful in providing the basic tools that made the probabilistic analysis tractable. He also helped in establishing the analytical difficulti posed by this structure and finding valid methods for accommodating them

Thanks to Fred Drenckhahn for providing detailed information on packaging and for accommodating many of my suggestions.

Thanks to Alex Ishii for reading and proofing the more theoretical portions of this work. His comments on the proper application of several theoretical results helped set me straight on a number of issues.

Thanks are in order to James Tetazoo, Jack Florey, Larry, $\phi\rho\epsilon\delta$, and their many friends. They certainly offered no positive support toward completing the work necessary for this thesis; however, they provided valuable diversions from thesis which allowed me retain adequate sanity to complete it in their spite.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Overview

Transit [Knight 89] is a high-performance network for large scale MIMD computers. Transit is intended to provide the underlying network support for a wide range of parallel processing paradigms including message passing, shared memory, and dataflow. Transit provides low-latency interconnect between processors via an indirect circuit switche network. The network is constructed as a bidelta multi-stage shuffle exchange network [Kruskal 86] utilizing 4×4 crossbar routing elements. The Transit bidelta network allows connections between source and destination to be made through a few routing stages for moderate size networks. In order to provide fault-tolerance and improve network routing efficiency, redundant paths are provided through the network. Routing switches are kept simple, and thus fast, by implementing a source responsible connection protocol; this frees the individual routing elements from the complexity of collision avoidance.

For moderate sized networks (*i.e.* 64 to 256 processors), the bidelta network construction keeps the delay uniformly small between all processors in the network. Additionally, using three-dimensional wiring strategies, the size and wire length can be kept reasonably small. This allows the network to be constructed in a single physical package within current technology limitations.

Scaling to larger network sizes by simply extending this strategy leads to a few difficulties. Network delays become uniformly large. The network itself becomes large enough that it must be divided into multiple parts for packaging. In addition, wire lengths grow such that reducing the clock rate of the system, due to wire delays, becomes a serious concern.

To avoid this uniform performance degradation, I propose a scheme for constructing larger networks using Leiserson's volume-universal fat-tree network [Leiserson 85]. In thi manner, locality can be exploited to provide short interconnect between closely situated processors; at the same time, interconnection between more distant processors is still possi ble without significant performance reduction from that of the bidelta network. The fat-tree network also allows the network to be decomposed into constituent elements for packaging in a straightforward manner.

The extension to a fat-tree network scheme primarily impacts interconnection topology and routing schemes. I describe how the fat-tree routing network can be realized utilizing the existing Transit routing element and packaging technology. This extension can be implemented with minor revisions to the Transit routing component.

In the remainder of this chapter, I briefly review the relevant properties of Transit (Sections 1.2 through 1.5) and fat-trees (Section 1.6). In Chapter 2, I develop some of the topology issues for building the fat-tree network structure. Chapter 3 follows providing concrete possibilities for the realization of the fat-tree network. Chapter 4 then proceeds t quantify some of the properties of the network and compares it with Transit bidelta networks

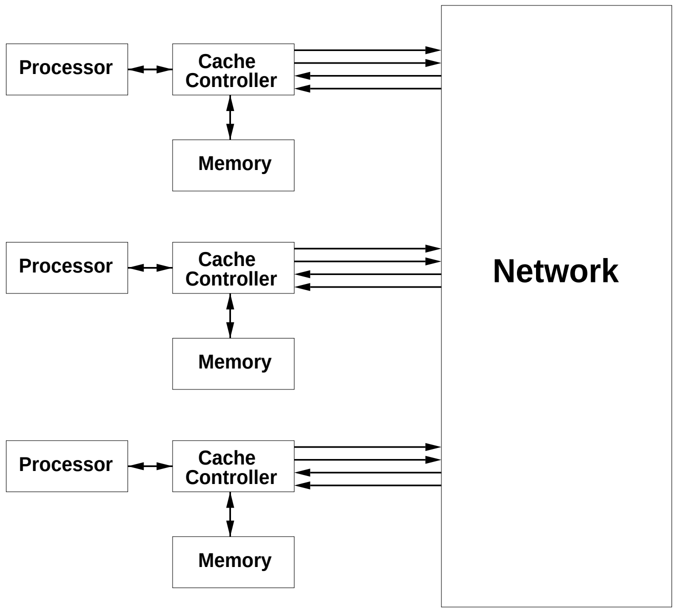

Figure 1.1: Network Processor Interface

of comparable size. Finally, Chapter 5 closes with conclusions and future directions.

1.2 Processor Network Interface Model

The development provided throughout this paper is concerned entirely with the construction of interconnection networks. In order for the network to be useful in the context of a large scale parallel computer, it must interface coherently with its set of processors. The network processor interface is shown in Figure 1.1. Each network endpoint is a processor with its own local memory and a cache-controller for maintaining its local cache and keeping the cache coherent with the rest of the network. Each processor has a pair of inputs and a pair of outputs to the network. Transit design intends the cache-controller to serve as the processor's interface to the network so that the processor need not explicitly deal with network interactions; however, this is a separate architectural issue and need not necessarily be the case.

The network input and output connections are paired for fault tolerance and improved routing success probabilities. Fault tolerance issues are discussed further in Section 1. The processor will generally only use one of its two inputs to the network, guaranteeing that the network will never be loaded above 50%.

1.3 Routing Component

Basic switching is provided by the routing element, **RN1**. This is a custom CMOS routing component designed to provide simple high speed switching. RN1 has eight nine-bit wide input channels and eight nine-bit wide output channels. This provides byte wide data transfer with the ninth bit serving as a signal for the beginning and end of transmissions.

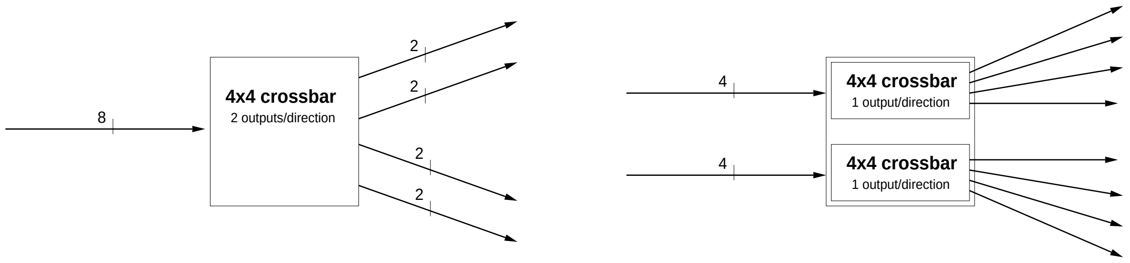

Figure 1.2: **RN1** Logical Configurations

RN1 can be configured in one of two ways as shown in Figure 1.2. RN1's primary configuration is as a 4×4 crossbar router with 2 equivalent outputs in each logical direction. In this configuration, all 8 input channels are logically equivalent. Alternately, RN1 can be configured as a pair of 4×4 crossbars, each with 4 logically equivalent inputs and a single output in each logical direction.

Simple routing is performed by using the first two bits of a transmission to indicate the the desired output destination. If an output in the desired direction is available, the dat transmission is routed to one such output. Otherwise, the data is ignored. In either case, when the transmission completes, RN1 informs the sender of the connection status so that the sender will know whether or not it is necessary to retry the transmission.

To allow rapid responses to network requests, RN1 allows connections opened over the network to be turned around; that is, the direction of the connection can be reversed allowing data to flow back from the destination to the source processor. The ability to turn a network connection around allows a processor requesting data to get its response quickly without requiring the processor it is communicating with to open a separate connection through the network.

Since RN1 always looks at the most significant two bits of the first byte of a transmission to determine switching direction, it is necessary for the bits within this byte to be rotate between network stages. This rotation guarantees each network stage a fresh set of bits from the routing byte. The rotation property must be provided by the network wiring scheme in which RN1 is used.

When configured as a 4×4 crossbar with two outputs in each logical direction, RN1 provides redundant paths through the network since it provides multiple outputs in each logical direction. This serves to increase both fault tolerance and the probability of routi success. When both logically equivalent outputs are available, RN1 randomly selects one of the two for use. In this manner, networks built from RN1 will have a level of fault tolerance in that multiple attempts to send a transmission through the network will most likely take separate paths through the network. The random selection of the preferred output port in each direction gives transmissions a good chance of avoiding any faulty component(s) entirely in successive connection attempts.

The alternative configuration for RN1 as a pair of 4×4 crossbars is provided for further fault tolerance. As Section 1.2 described, there are two outputs from the network for each processor. Using only the standard RN1 configuration, these two outputs would have to come from a single routing component making that routing component critical to

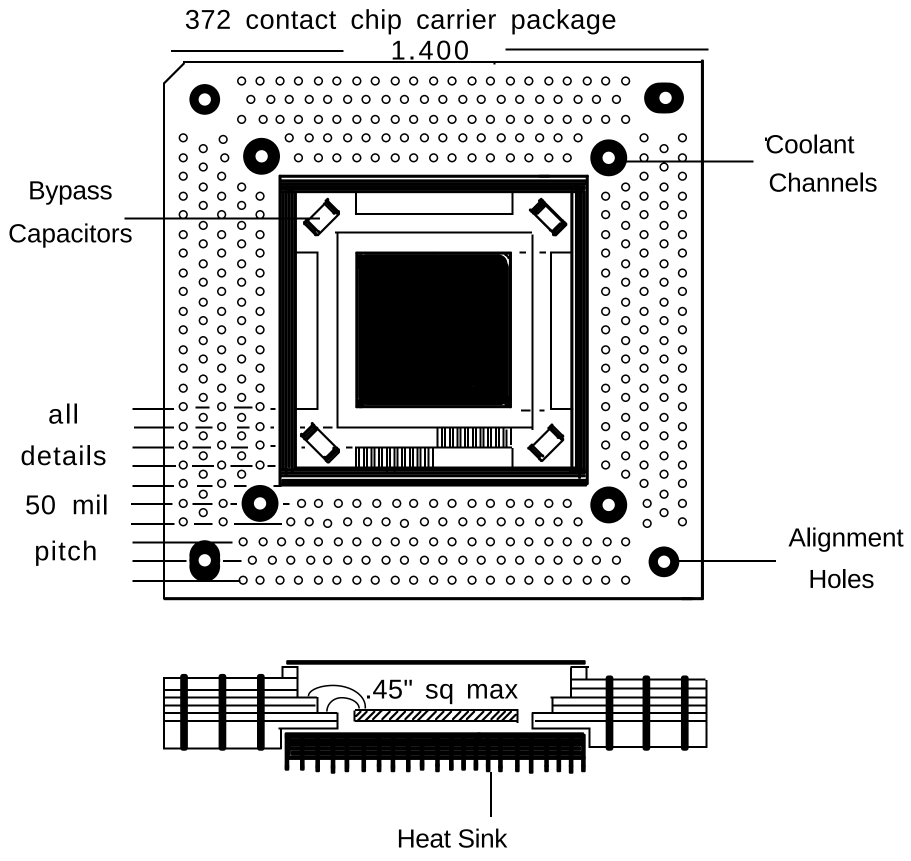

Figure 1.3: **RN1** Package (Diagram courtesy of Fred Drenckhahn)

the proper functioning of the network [Leighton 89-2]. With this alternate configuration, the two outputs from the network for each processor can come from two different RN1 components so neither component is critical. Section 1.5 explains this fault tolerance issu in the context of the Transit bidelta networks.

The Transit routing component is described further in [Knight 89] and [Minsky 90].

1.3.1 Physical Description

RN1 will be packaged as a pad grid array with all signals appearing on pads on both sides of the package. Figure 1.3 shows a top and side view of the RN1 package. A total of 76 periphery pads will provide through routing conduits for signals. Holes in the package are provided for packaging alignment and coolant flow. Since RN1 is designed to drive 144 signal pins simultaneously at 100MHz, provisions for liquid cooling are essential. The target physical statistics for RN1 are summarized in Table 1.1.

Clock Rate	100MHz
Size	$1.4'' \times 1.4'' \times 0.11''$
Signal Pins	150+
Through Routing Pins	76
Total Contacts/Side on RN1 package	372

Table 1.1: RN1 Physical Statistics

1.3.2 Routing to more than 256 destinations

Using the first byte to specify routing destinations as described above becomes limiting when attempting to address a large number of destinations. A single byte can only distinguish 256 distinct destinations.

To address this potential problem, RN1 has a provision for dropping the leading byte of a stream of data before looking at it. It then uses the remainder of the stream as if the first byte never existed. Thus, RN1 starts using the new routing byte, which was originally the second byte in the message, at the stage where the first was dropped. By properly designating the stages in the network at which components should drop, or *swallow*, the first byte in this manner, we can specify arbitrarily many distinct destinations.

This *swallow property* of a network routing stage is a static property. In the simply case of bidelta networks, the swallow property can be set on a per chip basis, since all inputs to a network stage will need fresh routing bytes at the same time. For full generality in the networks considered here, it is beneficial to be able to configure this property on a per input basis. This allows a single routing component to have inputs that connect to paths of varying lengths.

1.4 Construction Technology

The basic unit of network packaging for Transit is the *stack*. A stack is a three-dimensional interconnect structure constructed by sandwiching layers of RN1 routing components between horizontal pc-board layers. The pc-boards perform inter-stage wiring and the bit rotations described in the previous section while the routing stages provide switching. Figure 1.4 shows a partial cross-section of a stack. The dominant direction of signal flow is vertical as connections are made vertically through the stack. At each horizontal routing layer, each path through the network will make a connection through a single routing component. Between routing layers, the connection is routed horizontally to the appropriate routing component in the next layer.

When the transmission reaches the top of the routing stack it is brought straight down, back through the stack, to connect to the destination processors. This is necessary because the set of source and destination processors will normally be the same. All routing through the layers of routing components is provided by the through routing pins on the RN1 package as described in the previous section.

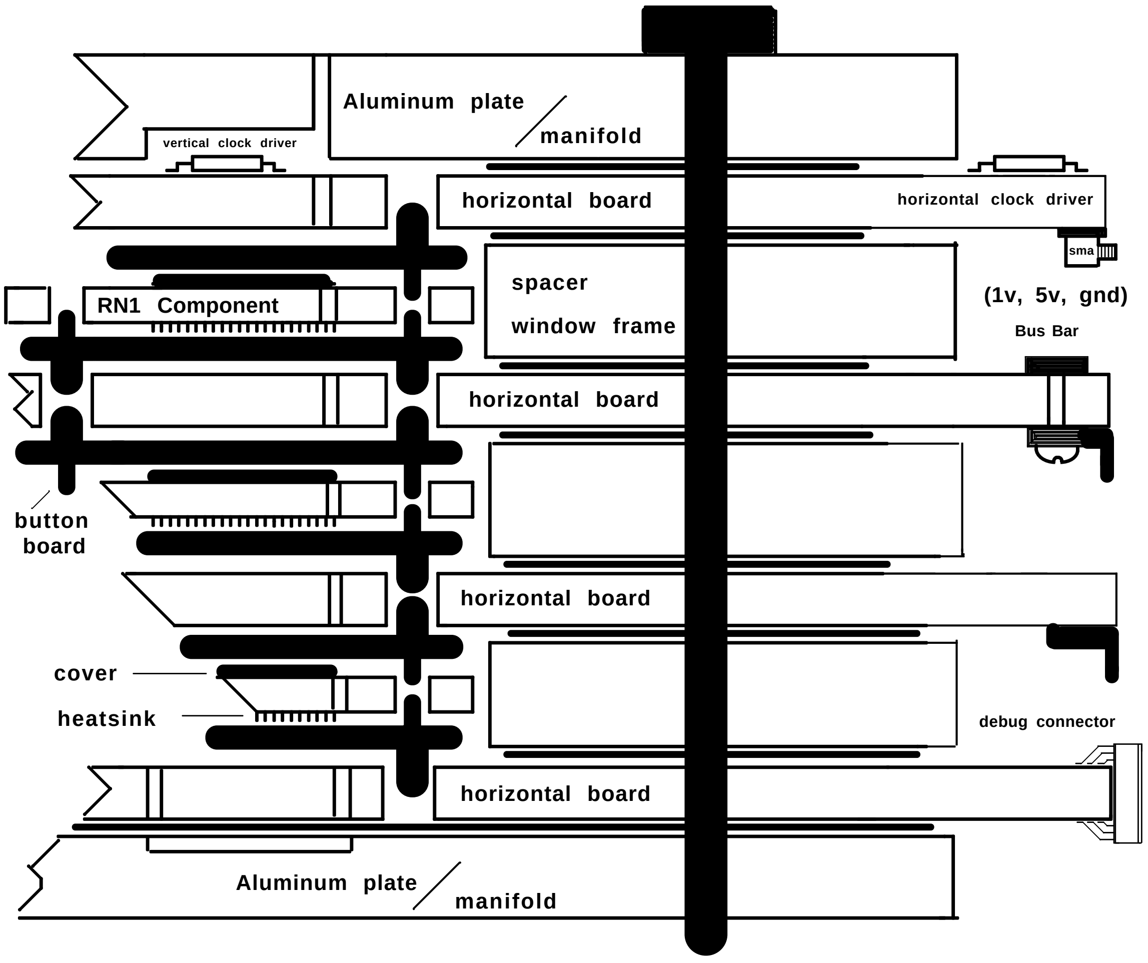

Figure 1.4: Cross-Section of Bidelta Routing Stack (Diagram courtesy of Fred Drenckhahn)

Contact is made between the routing components and the horizontal pc-boards through button board carriers. These carriers are thin boards, roughly the same size as the routing chip, with button balls [Smolley 85] aligned to each pad on the routing chip. These button balls are 25 micron spun wire compressed into 20 mil diameter by 40 mil high holes in the button board connector. They provide multiple points of contact between each routing component and horizontal board when the stack is compressed together; in this manner they effect good electrical contact without the need for solder. This allows ease of packaging construction and component replacement.

Channels are provided both in the stack and through each routing component for liquid cooling. FC-77 Fluorinert will be pumped through these channels to provide efficient heat removal during operation.

At the targeted clock rate of 100MHz for network operation, wire delay consumes a significant portion of the clock cycle. Thus, the physical size of the horizontal routing boards is an important consideration for network performance. Additionally, with current technology for fabricating pc-boards, it is not possible fabricate pc-boards any larger tha $2' \times 2'$ with reliable yield.

Using 12 layer pc-boards for the horizontal routing, a stack is expected to have a side length of roughly: $2 \times$ (chip side length) $\times$ (number of chips across side). Each layer, including pc boards, routing components, and connectors, will add $0.25''$. The height of a stack will be roughly: $0.25'' \times$ (number of routing layers).

A more detailed description of Transit packaging technology is given in [Knight 89].

1.5 Fault Tolerance in Bidelta Routing Stacks

The network interface model described in Section 1.2 along with the properties designed into RN1 allow a reasonable level of fault tolerance to be built into Transit bidelta networks.

Two inputs are provided from the processor or cache-controller to the network. The processor is expected to utilize only one of these two inputs at a given point of time. Leaving the second input unused guarantees that the network will never be congested above the 50% level (Section 1.2). Allowing the processor to chose which of the two inputs to the network to actually use at the beginning of each network transaction, prevents a single link between the processor and network from being critical to the processor's ability to communicate over the network. When two inputs are provided to the network through separate routing components, we guarantee that no single component failure will isolate a set of processors from the network.

Within the network, the standard RN1 configuration provides multiple outputs in each logical direction. This allows the number of different paths through the network to expand. No single routing component in the interior of the bidelta network is critical since there is a complete path from each source to each destination which avoids any given routing component.

The final routing stage of the bidelta network is constructed with the RN1 routing components configured in the alternative manner in which RN1 acts as a pair of single output 4×4 crossbars. In this way, the final switching stage for the pair of outputs destined to the same processor can be distributed across two different routing components. Since any connection to a given processor could be routed through either output, and hence either of the two routing components in this final stage, neither of the routing components is critical.

A more detailed description of this scheme for achieving fault tolerance in multistage networks is given in [DeHon 90].

1.6 Fat-Trees

Fat-Trees are universal routing networks for interconnecting processors in a multiprocessor environment [Leiserson 85]. Fat-Trees interconnect processors using a complete tree structure. The processors are located at the leaves of the tree while the tree's internal node compose the interconnection network. Fat-Trees parameterize the bandwidth between internal nodes according to their distance from the root node. In general, nodes closer to

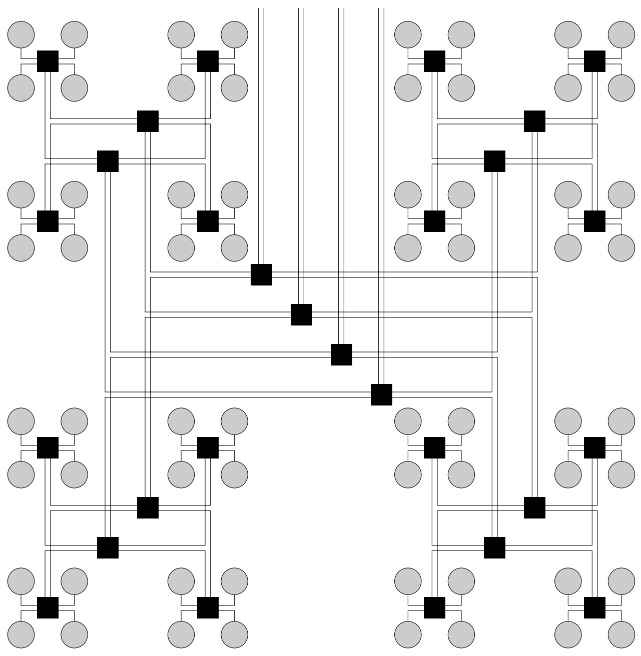

Figure 1.5: Area-Universal Fat-Tree with Constant Size Switches (Greenberg and Leiserson)

the root have greater bandwidth to accommodate additional message traffic. Connections between processors within a sub-tree can be made without consuming bandwidth higher in the tree. This allows locality to be exploited while reserving critical "long distance bandwidth for connections between widely separated processors. A fat-tree achieves these properties with only a linear increase in the number of routing stages that must be traversed in the worst case over a comparably sized bidelta network.

Fat-Trees are of particular interest because they can be area- or volume-universal when the bandwidth growth toward the root is selected properly. A fat-tree is volume-universal if it can simulate any other network constructed from the same amount of hardware with at most a polylogarithmic slowdown. This property guarantees that the hardware dedicated to constructing the routing network can be utilized efficiently.

In [Greenberg 85] Greenberg and Leiserson propose the area-universal fat-tree shown in Figure 1.5. The fat-tree shown demonstrates a basic construction structure for fat-trees using constant size switching elements. The internal node capacity doubles on each successive level toward the root. This average capacity growth is sufficient to guarantee area-universality for a quaternary fat-tree.

Fat-Tree networks were developed by Charles Leiserson. He provides a detailed description in [Leiserson 85] as well as proofs for the universality properties of fat-trees. [Greenberg 85], Leiserson and Greenberg expand the generality of fat-trees by proving that the universality properties hold for on-line routing algorithms.

2.1 Hybrid Fat-Tree Approach

The fat-tree network structure will be used to interconnect small Transit bidelta routing networks. That is, the leaf nodes of the fat-tree will themselves be bidelta networks rather than individual processors. This allows a moderate number of processors to be clustered together and share uniformly short interconnection paths amongst themselves. All these bidelta clusters are then interconnected through the fat-tree network allowing locality to be exploited between adjacent clusters while making it possible for any pair of processor to communicate in a moderately efficient manner.

The terminal clusters are built in stacks much like the standard Transit bidelta network stacks. The only difference is that a bidelta cluster stack must have some bandwidth between itself and the fat-tree network rather than dedicating all bandwidth in and out of the stack to processors. This can be done in a straightforward manner as shown diagrammatically in Figure 2.1. The processors connect to the bidelta network, but instead of consuming all of the bandwidth into the first stage of the bidelta network, they consume only three-fourth of the bandwidth. The first routing stage routes in four logical directions as before. However, only three of these directions route to further switching stages in the bidelta cluster. One of the logical destinations out of the first routing stage connects direct to the fat-tree network. Thus, the remaining routing stages will only be three-quarters the size of the first routing stage since one-quarter of the interconnect bandwidth leaves the stack after the first routing stage.

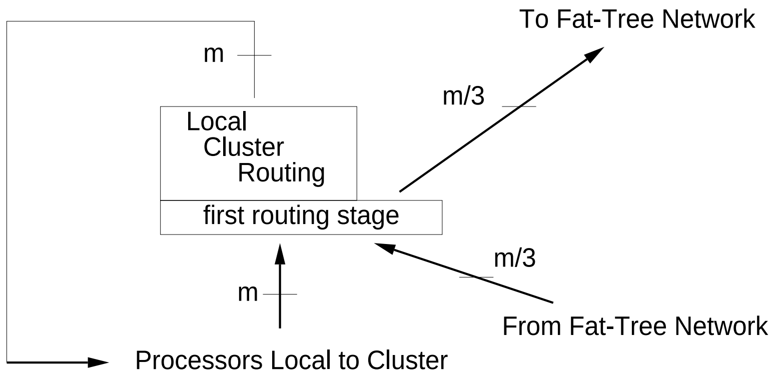

Figure 2.1: Bidelta Cluster at Leaves of Fat-Tree

2.1.1 Allocation of Bandwidth to and from Fat-Tree

In the manner described, the fat-tree network consumes one-quarter of the total bandwidth into and out of the bidelta routing stack. Clearly, the one-quarter of the bandwidth from the first stage of routing to the fat-tree network must come from all the routing components in the first stage. Naively, it would be possible to connect the bandwidth back from the fat-tree network to the bidelta cluster to any of the inputs of the first stage routing components as all these inputs are logically equivalent; however, it turns out that many possible configurations prove not to be optimal.

One extreme would be to connect all the input to the bidelta cluster from the fat-tree network to all of the inputs of one-quarter of the routing elements in the first stage. This would be non-optimal in terms of fault-tolerance because an unfortunately placed single chip failure in this first routing stage could eliminate 8 of the inputs from the fat-tree network. In terms of routing, this would be wasteful of bandwidth to the fat-tree network. The outputs from the first routing stage to the fat-tree network that come from the set of routing components with all their inputs origination from the fat-tree network would be unused. This results because there is no reason to route a connection through a leaf node. Clearly, this extreme is undesirable.

The alternate extreme is to distribute the bandwidth from the fat-tree network to the cluster over all the routing components in the first routing stage. In this scheme, each routing chip in the first stage has 2 of its 8 inputs connected to the fat-tree network. A single chip failure will always reduce the bandwidth from the fat-tree network to the bidelta cluster by two. Through each routing component at most six of the inputs will need to be switched in the direction of the fat-tree network. Given even connection frequency distributions across all processors of requests requiring the fat-tree network, each pair outputs to the fat-tree will be equally loaded.

A brief consideration of alternatives between these two extremes makes it clear that the later extreme is the most equitable distribution. Any configuration between these two extremes would make it more likely for some processors to be able to make a non-local connection than others. Similarly, any scheme other than the later extreme would necessarily lose more bandwidth due to a worst-case single chip failure. Thus, the most equitable configuration is the one in which all the bandwidth between the fat-tree network and the bidelta cluster is evenly distributed over all the routing components composing the first stage of routing in the bidelta cluster.

2.1.2 Fault Tolerance

The bidelta leaf cluster is configured for fault tolerance in the same manner as Transit bidelta networks (Section 1.5). The pair of inputs from each processors should be distributed to different routing components. All but the final stage of routing is provided by RN1 components in the standard 4 crossbar configuration with 2 outputs in each logical direction. The final output stage is constructed with the RN1 components configured in the alternate manner such that each processor receives its pair of network outputs from two distinct routing components.

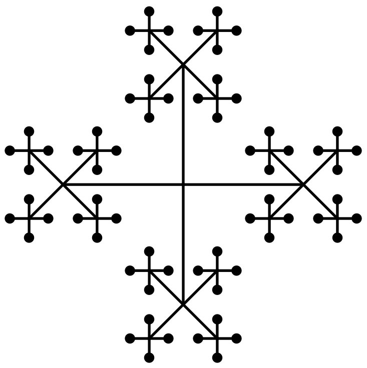

Figure 2.2: Logical Topology of Quaternary Tree

2.2 Fat-Tree Topology

2.2.1 Quaternary Tree

The appropriate fat-tree to construct using RN1 is logically a quaternary fat-tree (Figure 2.2) as opposed to the binary fat-trees prevalent in the literature. This is the natural selection since our routing component is a 4×4 crossbar switch. Using less than four directions would require overspecifying every routing path through the network since two logical output ports from a routing chip would actually be going in the same logical direction. Similarly, constructing a tree with a branching factor greater than four would require multiple stages of routing components to construct each virtual routing stage. The quaternary tree, of course, has fewer levels of routing for a given network size than a comparable binary tree.

2.2.2 Separate Up and Down Trees

The logical structure of a fat-tree as described in [Leiserson 85], [Greenberg 85], and elsewhere integrates up and down routing into a single tree. For construction purposes based on the Transit routing components and technology, it is preferable to actually construct this logical structure with the up and down routing trees separated. Lateral connections are provided between the two routing trees at every level to allow connections to cross-over from the up routing to the down routing tree as soon as appropriate.

2.2.3 Structure

Thre fat-tree structure will be constructed in three-dimensional space. A separate plane is allocated for each tree level. One way to view this, is to take the logical structure shown in Figures 2.2 and raise each internal tree node up to a plane equivalent to its height in the tree. Figure 2.3 shows this three-dimensional mapping of the tree structure.

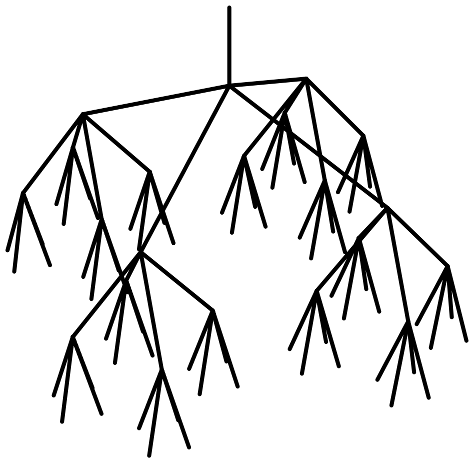

Figure 2.3: Three-Dimensional View of Tree Structure

2.2.4 Upward Routing

RN1 is a crossbar switching element. Since RN1 has multiple outputs in each logical direction, it does some amount of concentration. Its primary strength, though, is in routing. We can take advantage of this by routing in the up tree as well as the down tree. Rather than actually going through every logical tree level on the journey up the tree to the desired height, some routing is done to allow the up routing path to short-cut around some tree stages. With this short-cutting, we don't need an up routing stage for every level of the tree. Using RN1 which distinguishes four routing directions, each up routing stage can route a connection to one of the next three lateral crossovers in the tree or to the next up routing stage in the tree. The next up routing stage will perform similarly. In this manner, one up routing stage is needed for every three levels of the tree. Figure 2.4 shows a cross-sectiona view of an upward router and its logical connections to components in the up and down routing trees. This cross-section spans three logical tree levels which are realized as o physical up routing stage and three physical down routing stages; the cross-section shows only a single component at each up and down routing stage.

Utilizing the routing capability of RN1 in this manner, routing up the fat-tree is accomplished in at most $\frac{\log_4 N}{3}$ stages as opposed to the $\log_4 N$ that would be required if no routing were performed in the up tree. At the same time, RN1 still performs some concentration at each upward routing stage but not full concentration[1]. That is, at each stage in the up routing tree, all inputs destined for the same parent node cannot end up on any wire in

[1]Leiserson did point out that if the routing component were modified slightly to allow a specification of "route to height x" in the fat-tree in addition to "route in direction y" it would be possible to specify large heights in the fat-tree with only a small number of bits and offer more freedom in terms of concentration and bandwidth allocation [Leiserson 89].

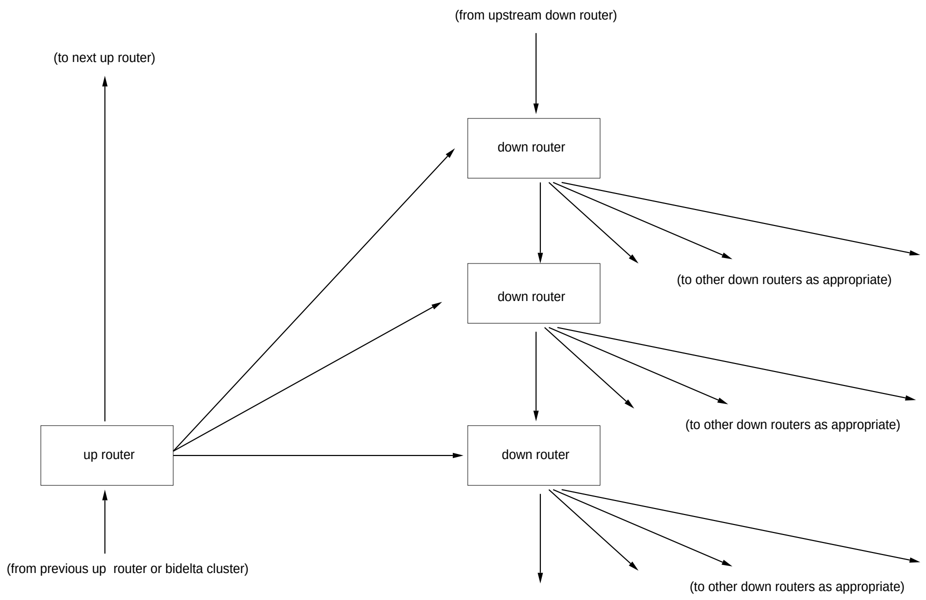
(from upstream down router)
(to next up router)
down router
down router
(to other down routers as appropriate)
(to other down routers as appropriate)
up router
down router
(from previous up router or bidelta cluster)
(to other down routers as appropriate)

Figure 2.4: Cross-Section View of Up and Down Routing Trees

the up the tree connected to that parent node. Each up routing stage allows a given input wire to be connected to any of two wires in each logical direction when wired properly; a concentration of two is effectively achieved at each upward routing stage. The best known means of achieving full concentration at each stage will cost $O(\log N)$ time and hardware [Ajtai 83] [Cormen 86] whereas RN1 performs its limited concentration in constant time and hardware. In this manner, the concentration in this fat-tree configuration is much like the fat-tree with constant size switches of Leiserson and Greenberg [Greenberg 85] shown in Figure 1.5. The number of signals into each level up the fat-tree does grow because fan-in occurs from more and more stages.

Figure 2.5 shows what the up tree connections look like in three dimensions. The routing components are at the base. The connections to further up routing stages are shown straight up from each routing component. Figure 2.6 shows horizontal cross-sections of Figure 2.5 to make the convergence and routing clear. Layer 0 is the layer of routing components. Layers 1 through 3 show where the logical connections from each routing component fan-in to the lateral connection of the next three crossover stages. The fourth logical connection out of the routing components, of course, connects directly upward to the next up routing stage.

Quick Routing for Very Large Systems

It is actually possible to do better than the $\frac{\log_4 N}{3}$ stages of up routing just described. By using a series of routing stages to switch a connection to the maximum height up the tree to which it needs to be routed, the connection to the appropriate lateral crossover between the up and down routing tree can be made in $\log_4$ of number of levels in tree. Since there are $\log_4(N)$ levels in the tree, this means only $\log_4(\log_4(N))$ stages are needed

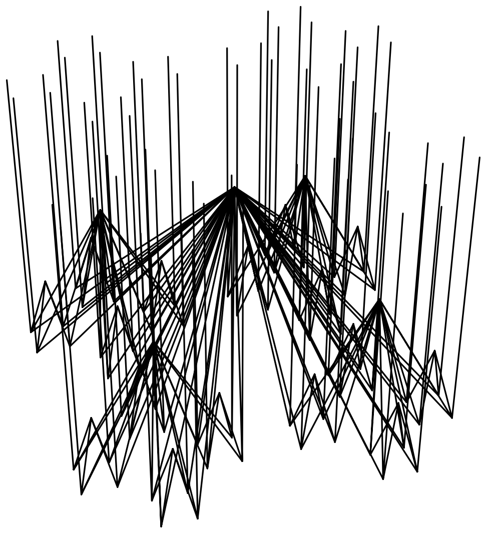

Figure 2.5: Three-Dimensional View of Connections from One Up Routing Stage

to perform upward routing.[2] In essence, this routing scheme builds a bidelta network from the processors to the various tree levels. Figure 2.7 depicts how an bidelta network can be used to reach any height in a tree of depth 16 in only 2 stages of routing.

However, this up routing scheme is only of interest when the fat-tree has a large number of tree stages. In essence, we are already getting the benefit from this scheme achievable when the size is on the order of 4 levels. The point where it actually becomes interesting to use this scheme occurs when the number of levels is roughly $4 \times 4 = 16$. In our scheme however, 16 tree levels implies a network supporting 4^{16} clusters each with 48 to 192 processors. Thus even in the the smallest case we will have $3 \cdot 4^{18}$ or about **200 billion** processors. The capacity analysis and geometry requirements for this case are not considered here since this is clearly not a scheme of current practical interest.

2.2.5 Down Routing

The down routing path is simply the straight-forward tree structure. Each down routing stage switches in four logical directions among its four sub-trees. The logical down routing path looks just like the logical tree structure shown in Figure 2.3. Most of the inputs to a down routing stage come from its parent in the fat-tree. The rest of the inputs come from the lateral cross-over from the up routing tree as described in the previous section.

2.2.6 Desired Capacity Growth Rate

One of the nice properties which fat-trees can have is volume-universality. This property essentially states that a fat-tree fat-tree can efficiently simulate any other network of

[2] *N.B.* It will always take this many stages regardless of the height routed by such an up routing tree.

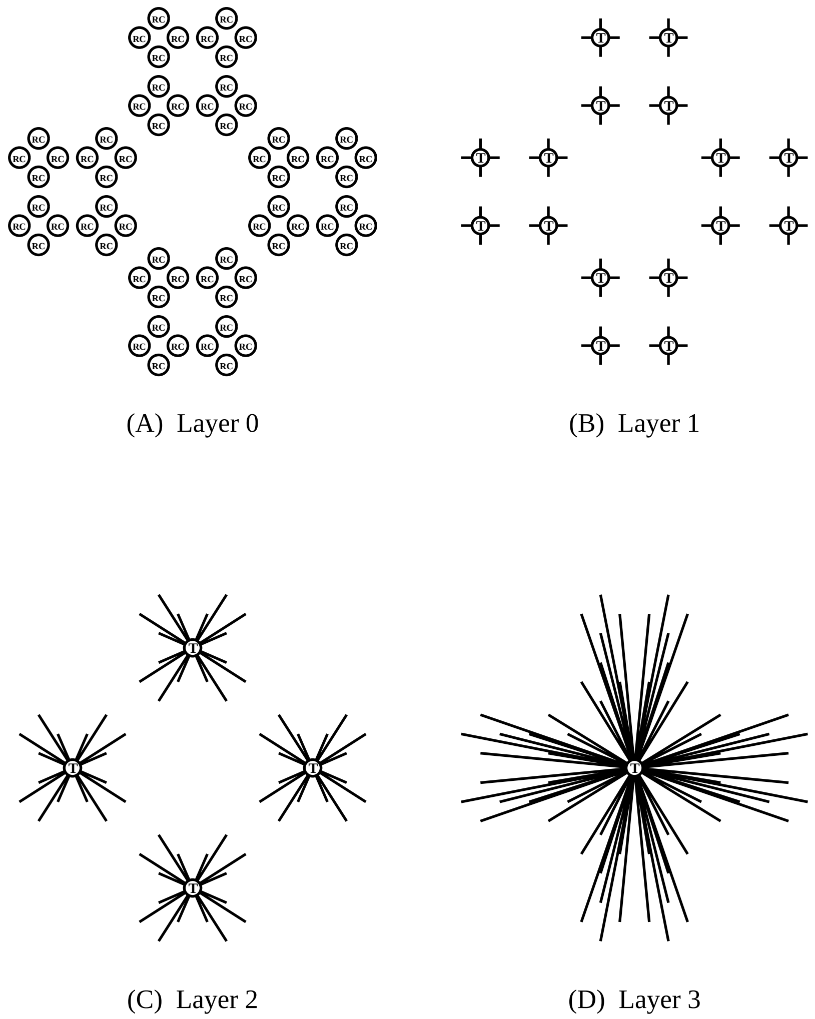

Figure 2.6: Horizontal Cross-Sections One Up Routing Stage

comparable volume with at most a polylogarithmic slowdown [Leiserson 85]. Additionally, volume-universality implies that larger fat-trees can be created by simply adding levels and scaling the fat-tree structure in the three-dimensional world.

To assure volume-universality, the bisection bandwidth must be properly correlated with the volumes contained within each portion of the network. For three dimensional structures, bandwidth into a volume is generally proportional to the surface area of the enclosed volume; this volume will in turn be proportional to the number of terminal nodes or network end points from which the volume is composed. Thus we assume:

- Volume $= v \propto$ network endpoints.
- Bandwidth $\propto$ Surface area $= a$

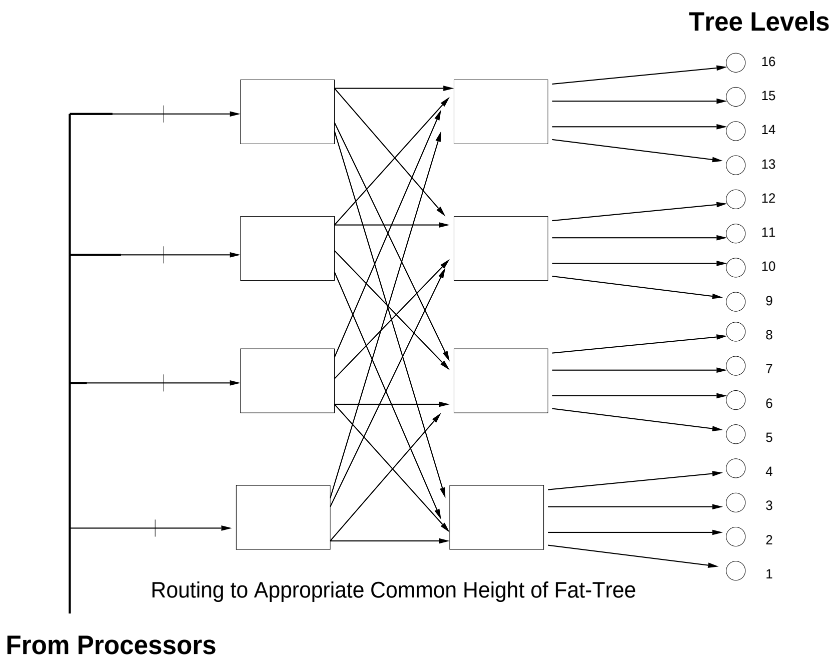

Figure 2.7: Hybrid Up Routing Scheme with $\log_4 N$ Stages in Up Routing Tree

- Surface area $= a \propto (\sqrt[3]{v})^2$

Using v as some measure of the number of network endpoints, and a as some measure of the network bandwidth, the important relation is $a \propto (\sqrt[3]{v})^2$. (If we assume that the enclosed volume increases by a factor of 4 ($v_n = 4v_{n-1}$) at each level, as one might expect for a quaternary tree, it is clear that volume universallity can be achieved only if the bandwidth increases no faster than the factor derived in Equation 2.1.

$$a_n \propto (\sqrt[3]{v_n})^2 = (\sqrt[3]{4v_{n-1}})^2 = (\sqrt[3]{v_{n-1}})^2(\sqrt[3]{4})^2 = a_{n-1}(\sqrt[3]{4})^2 = a_{n-1}\sqrt[3]{16} \qquad (2.1)$$

Thus, on average, bandwidth should increase by a factor of $\sqrt[3]{16}$ at each successive level up the tree to maximize the bandwidth available in the network while keeping the growth rate appropriately bounded. Whether this bandwidth increase can actual be realized within the assumed factor of four growth rate is still an open question. A generalization of the universallity proofs in [Leiserson 85] and [Greenberg 85] to this rate of growth may be possible, but has yet to be demonstrated.

2.2.7 Channel Capacity Growth

It is easy to see the average channel capacity, number of distinct physical connections in or out of an internal tree node, growth by looking at the channel capacity growth across three tree levels. For simplicity, consider the portion of a fat-tree network shown i Figure 2.6. The channel capacity of each logical channel in and out of the bottom of this

Layer	Tree Level	Capacity	Logical Channels
0		8	64
1	n	8	16
2	n+1	32	4
3	n+2	128	1
0		*128*	1
1	n+4	*128*	$\frac{1}{4}$

Table 2.1: Up Tree Bandwidth Allocation

structure is eight. The channel capacity of the single logical channel in and out of the top of this structure is 128. Thus, the channel capacity has grown by a factor of 16 across 3 tree levels giving the desired average growth of $\sqrt[3]{16}$.

Since the up routing and down routing trees are separate, the channel capacity growth in the up tree differs somewhat from the capacity of the the corresponding network stages in the down tree.

Up Tree

To see how the up tree routing capacity grows, it is easiest to look at the channel capacities for the first few stages of the network. Consider the portion of the up routing tree shown in Figures 2.5 and 2.6. Layer 0 simply performs the routing to the next three layers so it does not correspond to an actual level in the tree. Each base routing component in layer 0 is an RN1 routing chip. Thus each of the 64 routing components shown in Figure 2.6(A) has 8 inputs and outputs. The 8 outputs are divided into four logical directions with two going to each of the next three layers and 2 going further up the tree. Thus layer 1 has 64 pairs of outputs converging to 16 different sub-trees (Figure 2.6(B)); the result is 16 logica channels of capacity 8. Similarly, layer 2 has 64 pairs of outputs converging in 4 sub-trees making 4 logical channels each of capacity 32 (Figure 2.6(C)). Finally, layer 3 has 64 pairs of outputs converging to a single sub-tree which gives a single logical channel with capacity 128 (Figure 2.6(D)). This leaves the remaining 64 pairs of outputs which connect to the next routing stage as a single logical channel with capacity 128. This logical channel will then encounter another routing stage just like layer 0 and the progression will continue in this manner. This capacity progression is summarized in Table 2.1. The final items in the table are the corresponding layer 0 and 1 progressions for the next routing stage. Thus the progressions of growth factors is 1, 4, 4 giving a capacity growth of 16 in 3 tree stages or an average growth of $\sqrt[3]{16}$.

Downward

The bottommost down routing level must provide 8 outputs in each logical destination to scale properly with input to the up routing tree. This dictates $8 \cdot 4 = 32$ outputs or 4

Tree Level	Capacity
n	8
$n+1$	24
$n+2$	64
$n+3$	128
$n+4$	*384*

Table 2.2: Down Tree Bandwidth Allocation

routing chips. This implies that the total fan-in to this level is also 32. $2 \cdot 4 = 8$ of tha
fan-in is consumed by lateral connections leaving $32 - 8 = 24$ inputs which must fan-in from above. At the next level, there need to be 24 outputs for each logical destination as just determined. This dictates $24 \cdot 4 = 96$ outputs or 12 routing components and a total fan-in of 96. There are $2 \cdot 4 \cdot 4 = 32$ lateral fan-in signals to this level, leaving $96 - 32 = 6$
inputs for fan-in from the previous down routing stage. Similarly each logical destination
at level three has 64 outputs associated with it. This dictates $64 \cdot 4 = 256$ total outputs or 32 routing chips. This gives a total fan-in to this level of 256. The lateral fan-in i
$2 \cdot 4 \cdot 4 \cdot 4 = 128$ leaving $256 - 128 = 128$ inputs for fan-in from above.

Beyond this point, the structure replicates. The next level up looks like the bottom level scaled so that the input size in each direction matches the output from the previous stage. progression may continue in this manner *ad infinitum.* The progression of channel capacities in the downward route are thus as shown in Table 2.2. Level $n+4$ is equivalent to the level n stage since the series begins to repeat at that point. Thus the progressions of growth factors is $\frac{8}{3}, 3, 2$, giving a capacity growth of 16 across 3 tree levels. This again gives the desired average growth of $\sqrt[3]{16}$. This is identical to the growth in the upward routing tree, but the growth does not coincide with the upward tree on a stage per stage basis.

2.3 Wiring Constraints for Efficient Bandwidth Distribution

In both the up and down routing portions of the fat-tree network, a logical routing stage is composed of many separate routing components. The inputs to any logical routing stage come from different logical directions; that is the inputs to the logical routing stage may be lateral crossover connections from different subtrees or may come from the immediate parent routing stage. The distribution of these inputs to physical routing components presents a more general case of the input bandwidth allocation discussed in Section 2.1.1. In this section, the issue of bandwidth distribution to routing components is examined in more detail in order to develop some constraints for the general problem of optimally wiring the fat-tree network.

Again, we start by examining the two extreme cases for bandwidth distribution. I use the word *dispersion* to refer to the degree to which inputs from different logical directions are distributed across distinct routing components. Dispersion is closely related to the

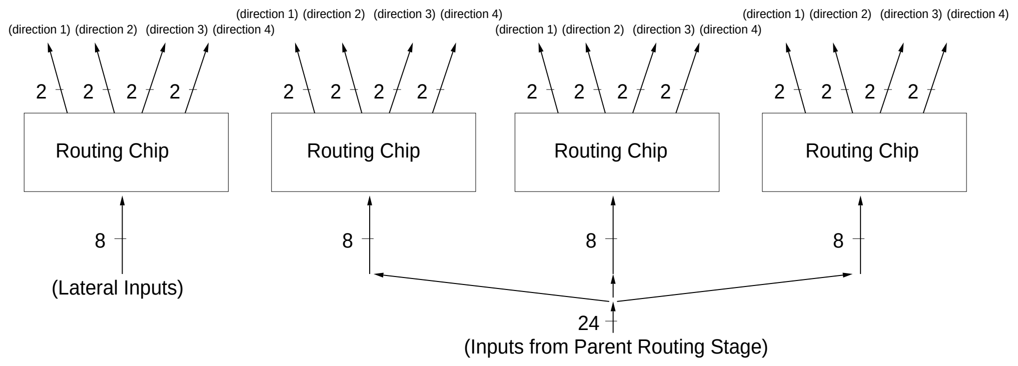
(direction 1) (direction 2) (direction 3) (direction 4)
2
Routing Chip
8
(Lateral Inputs)
24
(Inputs from Parent Routing Stage)

Figure 2.8: Non-Disperse Example

notion of *expansion* in the theory literature.

2.3.1 No Dispersion

In the non-disperse case at each such logical routing stage, the inputs from a given direction are each routed to their own set of routing components. That is, if the inputs from a given direction make up a fraction α of the inputs into that level, then these are all destined for an equivalent fraction of the routing components at that level. Components will be shared between logical input directions only in the case that there is an uneven division of components among the input directions.

At any stage, inputs from a given direction can only make use of α of the bandwidth to each logical destination. This means the inputs from that stage can, at most, reach α of the routing components at the next level. This effectively minimizes the number of different components, and hence paths, reachable; This is non-optimal for fault-tolerance and minimizing routing congestion. On the positive side, at least α of the bandwidth in a given direction is guaranteed to the inputs from each input direction; since the inputs from a single logical direction do not, in general, share routing components with inputs from other directions, those inputs are guaranteed the entire α of the bandwidth. Non-dispersion means that at each level, inputs from the same direction are competing only with each other for the bandwidth to the next level.

Example Consider Figure 2.8 for a concrete example of this wiring strategy. This switching could represent Level 1 of the fat-tree in the downward routing path. The logical routing stage is thus composed of four routing components as shown. In this case, 8 wires enter from the lateral direction while 24 enter from the previous down routing stage in the downward routing tree. Eight wires leave level 1 in each of the 4 logical directions.

Here, the 32 wires are partitioned based on their direction of origination. The 8 lateral wires all go into a single routing chip. The 24 wires from farther up the tree are connected to the remaining 3 routing chips. Of the 8 lateral connections through this level, at most two connections can be made in each logical output direction; however, if more than two

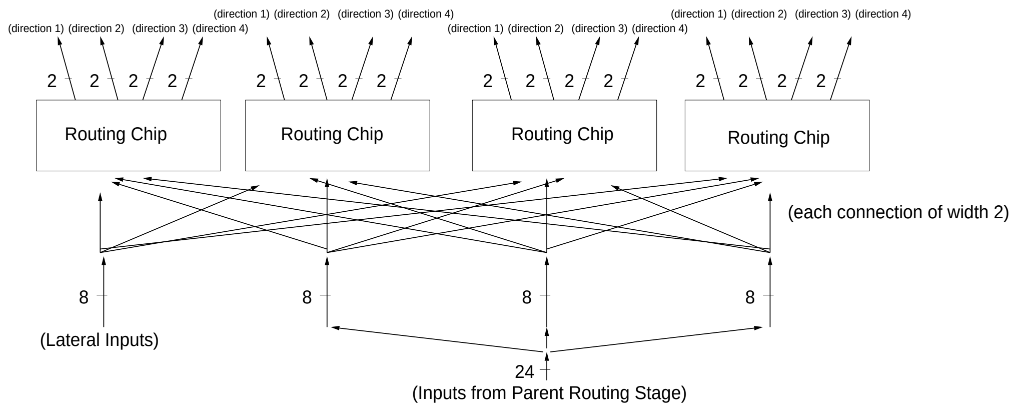
(direction 1) (direction 2) (direction 3) (direction 4)
2
Routing Chip
(each connection of width 2)
8
(Lateral Inputs)
24
(Inputs from Parent Routing Stage)

Figure 2.9: Disperse Example

lateral connections need to be made in a particular direction, at least 2 of the connections will be made. A single chip failure at this level would either prevent lateral communication entirely or cut down the bandwidth from farther up in the tree by 33%.

2.3.2 Full Dispersion

Full dispersion is achieved when the input wires to each level from a given input direction are spread out amongst as many different routing components as possible. The pair of outputs from the same direction of a single chip at a previous level always go to different routing components.

The number of potential paths from point to point expand at each level. This results because each input wire from a logical input direction has the opportunity to leave on any of two outputs; at the same time, when a connection from a logical input direction does exit the routing level via a particular output port, it is not diminishing the bandwidth potential for any other inputs from that same logical direction. Thus inputs from all directions compete for the bandwidth to the next level. To a limited extent, this allows averaging of the load from several input directions across the total available bandwidth. Similarly, the effects of component failure is spread evenly over all logical input directions. When a chip fails, it effectively reduces a fraction of the fan-in from each logical direction.

Example Figure 2.9 gives the concrete example for the full dispersion case. This corresponds to the disperse wiring of the same level 1 downward routing path given in the example of the previous section.

In this case, the 32 wires are distributed evenly, based on direction of origination, acros the 4 routing chips. If all 8 lateral connection were destined for the same output direction, there is a possibility that they could all be routed; there is also the possibility that th could all be blocked because all the bandwidth in that particular direction is consumed by connections farther up the tree. A single chip failure at this level would effectively cut th bandwidth out of this level down by 25%.

2.3.3 Potential Path Growth

From the above, we see that full dispersion will maximize the number of paths available for a connection through the network. This factor of two increase at each level in the number of paths over which a given connection can be made is the best achievable using RN1. This occurs when each wire on which a connection could potentially be made by a common source is connected to a different routing component. There are then twice as many potential paths toward the desired destination out of each routing level as there were into the level. As long as this property can be maintained, each routing level will be able to achieve this effective doubling of the number of paths between a given source and its destination.

Certainly, it is possible to wire the network with less potential paths between each source destination pair. If two or more wires which a given source could have routed a message through are connected to the same routing chip, then the path growth will be smaller.

In order to guarantee that we can achieve the maximal path growth or path *expansion* described above, at each level, it need only be the case that there is not a single set of wires that can be reached from a single processor that accounts for more than $\frac{1}{8}$ of the total bandwidth into any level. This is not the same as saying that the inputs from a single logical direction must compose less than $\frac{1}{8}$ of the inputs to a given logical routing level. The inputs from a given logical direction are not necessarily fully concentrated when they enter a routing level. As long as the constraint given above on the fraction of potential inputs from a single source can be maintained, each input wire from a potentially common source can be routed through a different routing component at any level in question. For the fat-tree routing structure described here, this property is maintained at each logical routing level composing the network.

Leighton and Maggs [Leighton 89-2] point out that the above constraint is sufficient only to guarantee optimal path expansion at the micro-level; that is when looking at the number of paths between a single source and destination. When viewing path expansion at the macro-level, additional constraints are necessary to guarantee optimal path expansion; macro-level path expansion is the growth in the number of paths between sets of source and destination processors rather than simply individual processors. Leighton and Maggs demonstrate the advantages of optimal or near-optimal macro-level expansion wiring when with a scheme where the switching elements can arbitrate with one another concerning network loading [Leighton 89-1]. RN1, however, selects output paths obliviously. As such, it is not clear that their macro-level constraints will have any significant affect in practice on the routing performance of a network built using RN1 switches. Recent work [DeHon 90] would indicate that this kind of expansion is useful to maximizing the fault tolerance of the network.

The complexity of Leighton and Maggs' switching element [Leighton 89-1] is much greater than that of RN1. As such, it would certainly represent a much slower component. Additionally, their switching scheme requires approximately $4(\log N)$ clock cycles to route a connection as opposed to the $(\log N)$ clock cyles required by RN1. Both of these properties make its routing much slower than RN1. However, if its routing efficiency turns out to be greater than that of the oblivious routing done by RN1 by a comparable factor, then perhaps it would be beneficial to look further into the alternative of constructing such a switching element.

3. Construction

This chapter expands on the technical details involved in the construction of the network structure described in the previous chapter. Section 3.1 discusses a few further issues about the construction of the bidelta clusters at the leaves of the fat-tree. Section 3.2 address a couple of issues important to the realization of a such network using RN1. Section 3.3 introduces the *unit tree*, a stack structure that can serve as the basic building block for this kind of fat-tree network. Then Section 3.4 describes a possible geometry for arranging these unit tree structures in three-dimensions to realize networks of reasonable size. Section 3. follows describing construction issues for this geometry. Section 3.6 then deals with the long wires that necessarily occur in this structure. Section 3.7 briefly details processo placement with respects to such a network. Finally, Section 3.8 details an efficient scheme for computing routing sequences for this network.

3.1 Bidelta Leaf Clusters

Section 2.1 described the composition of the bidelta leaf clusters. This section fills i construction details not readily apparent from the previous description.

3.1.1 Connections To Fat-Tree

For simplicity, the 11 logical routing direction out of the first stage of routing in the bidelta cluster is chosen to route into the fat-tree network. That is, if the first two routin bits of the routing sequence are both ones, the connection will be routed out of the bidelta cluster and into the fat-tree network. The choice of which logical direction to use for this purpose is somewhat arbitrary, but it is useful to settle on a single direction for referenc purposes.

3.1.2 Connections From Fat-Tree

The first stage routing components will be configured to swallow the first routing byte they encounter, as described in Section 1.3.2, for cluster inputs from the fat-tree network This means the swallow property must be set only on the two inputs from the fat-tree network to each routing component in the first routing stage; the other six inputs to each routing component will come directly from processors and not require this swallow property to be set. In this manner, a fresh routing bytes is always used to route through the bidelta cluster regardless of the source of the connection. This provides a logical separation betwee the portion of the routing sequence used to route through the fat-tree and that used to route within the cluster making routing calculation moderately easy. It also guarantees that a fresh set of routing bits is available to route through the bidelta cluster.

3.1.3 Size

The number of processors that a single bidelta clusters stack can support is constrained by physical size, bandwidth, and granularity. Since the routing component distinguishes four distinct directions, each additional routing stage will increase the size, number o processors, by a factor of four. Note, however, that since one logical direction out of the first stage connects to the fat-tree rather than the bidelta network, that the first stage only distinguishes three directions within the network. This granularity constraint alone specifie the construction of clusters which support $3 \cdot 4^i$ processors, for any non-negative i. When considering physical size we are limited by the size we can construct the horizontal routing boards (see Section 1.4). Considering the projected size of RN1, the largest possible singl horizontal routing plane we can construct will support $8 \times 8 = 64$ routing components; this would allow the construction of the 4 stage network supporting $3 \cdot 4^3 = 192$ processors. Additionally, the bidelta cluster will have to provide bandwidth to and from the fat-tree network which will match that of the fat-tree network being constructed.

For notational convenience, a bidelta clusters supporting n processors will be denoted by B_n. *e.g.* a 3 stage bidelta clusters will support $3 \cdot 4^2 = 48$ processors and will thus be referenced as B_{48}

3.2 Inter-Stage Details

3.2.1 Up Path Directions

Routing components in an up routing stage route in four distinct directions as shown in Figure 2.4. Here, again, the assignment of actual routing directions to each of the four logical directions is somewhat arbitrary. It is useful, though, to make this assignment in a logical manner for the sake of references and routing simplicity. The 11 direction is assigne to the direction which routes further up the tree. This is consistent with the choice of the 11 direction to route out of the bidelta cluster. The 00, 01, and 10 directions each respectivel route to the each of the next three successive down routing stages. That is the 00 direction routes to the next down routing stage, 01 to its parent, and 10 to the next parent.

3.2.2 Swallows

In order to allow routing to arbitrarily many destinations, we need to make sure that the fat-tree network is configured to discard an old routing byte when it is exhausted. Thus the *swallow* stages need to be arranged in a functional manner. This task is complicated by the fact that connections can be made through arbitrary heights in the tree such that complete paths from source to destination are not entirely homogeneous; that is, paths will differ in length based on where they entered the down routing tree.

Ideally we would like each of the swallow stages to be placed a a maximum distance from each other. This is desirable because each swallow stage effectively costs one clock cycle of delay while the old routing byte is being discarded so the new can take its place. Thus to minimize the time to make a connection, swallow stages should be placed as infrequently as possible.

Since there are 8 bits in each routing byte and each routing stage consumes 2 bits, swallows are required at least every four routing stages. Swallow stages will certainly be needed at least every 4 stages on the up path and on the down path. It is possible to change from up routing to down routing at any up stage. In order to be assured that the down routing stages get fresh routing bytes when needed, the old routing byte will be stripped off as part of the lateral crossover from the upward tree to the downward. This routing byte change will be realized by setting the swallow property on all lateral inputs to a down routing stage. This lateral swallow provides a clear separation between the up and down routing portion of a connection.

3.2.3 Bit Rotations

Recall from Section 1.3 that the appropriate bits from the data path are used to perform routing. In order to assure that a different pair of bits are used at each level, the data path is rotated between routing levels. This poses a similar problem to the swallow. Whereas, in a bidelta network, all paths in the network are of the same length, all paths through the fat-tree are not. More importantly, all paths are not even of the same length modulo four. Even paths through a given routing component can differ in length. Thus, some attention must be given to assuring that the bits are rotated uniformly through the tree network.

From the source processor to a given height in the tree, the amount of rotation incurred will be known. Similarly, from that height down to a destination, the number of rotations will be known. What we must assure is that the total number of bit rotations modulo four through the network is the same regardless of the height at which the lateral crossover occurs. In order to guarantee this, we must use the lateral crossover between the up routing tree and the down routing tree to shuffle the bits into a consistent state; that is make sure that the lateral bits into a given down routing level have the same rotation applied to them as those coming into the down routing level from above. This is closely related to need to swallow on the lateral crossover in order to be synchronized with the point of entry into the downward routing path and similarly guarantees that, regardless of the history of a connection's path, it merges consistently into the downward tree.

3.3 Unit Tree

In building the fat-tree network, we are constrained in the size we can reliably fabricate the component stacks. Recall from Section 1.4 that the horizontal pc-boards are limited to about two feet square. There are also other difficulties that arise if we attempt to build larger stacks. The on-board wire lengths begin to become a more serious concern slowing the clock cycle of each routing stage. The already dense wiring on the pc-boards becomes proportionally more severe.

It is clear that the fat-tree network will have to be packaged in multiple component stacks. From this realization, it is necessary to determine how to separate the fat-tree int stacks and how each constituent stack is composed. We wish to avoid building many stacks of different geometries, and hence needing to design and fabricate many different pc-boards and other components. We want as much as possible to be reusable when scaling to larger and larger sized fat-trees. Essentially, we need a standard replicable structure that can serve as a building block for fat-trees in much the same way that a set of identical routing components can be used to construct a routing stack.

3.3.1 Unit Tree Structure

The *unit tree* is a single stack structure from which a fat-tree interconnection of a wide variety of sizes can be built. This basic building block, when replicated and properly arranged will realize the fat-tree structure described herein.

Size

As noted, current technology limits the size of a single horizontal pc-board routing layer to about $2' \times 2'$. Given the target size of RN1 as $1.4'' \times 1.4''$, if we leave an equal amount of space between each routing component for routing wires, we can place about $8 \times 8 = 64$ routing components in a single routing layer. As such, a unit tree is constrained to this size for the present time. As technology improves, the number of components per layer will, no doubt, increase; however, this scheme is very scalable and can easily be modified to take advantage of improved technology. For the remainder of this paper, 64 routing components will be assumed as the maximum layer size.

The basic structure of the fat-tree described in Section 2.2 showed that the fat-tree is built by replicating a series of one upward routing stage and 3 downward routing stages. These 4 stages describe the natural structure of the tree. This gives a natural division between routing levels appropriate for inclusion in each unit tree stack.

The unit tree is thus constructed in 4 layers as shown in Figures 2.5, 2.6, and 2.3. The up routing layer is at the base and is composed of the maximum 64 routing components. It is followed by the first down routing stage which is also composed of 64 routing components in order to match bandwidth with the up routing stage. The next down routing stage provides input to only three-fourth of the first down routing stage since lateral inputs consume one-quarter of the input capacity to the first down routing stage. It is thus composed of only 48 routing components. The third, and final down routing stage provides inputs to the second down routing stage. Since the second down routing stage has one-third of its inputs dedicated to lateral crossover inputs, the third stage is only two-thirds the size of the second. This makes it one-half the size of the first down routing stage which is 32 components. Thus, the component count for a unit tree is shown in Tables 3.1. Each such unit tree thus requires a total of 208 routing components. With four routing layers, the stack should be about $1.5''$ tall making the entire stack roughly $2' \times 2' \times 1.5''$. Since there are several variations of this basic unit tree worth considering, when it is necessary to differentiate them, this particular unit tree will be referenced as UT_{64x8}

Characteristics

With this development, the size and geometry of a basic unit tree is fully constrained. As such we can summarize the bandwidth characteristics as given in Tables 3.2.

Level	Routing Components
0	64
1	64
2	48
3	32

Table 3.1: Unit Tree Component Summary

	Total Bandwidth	Total Logical Channels	Capacity
Up from Leaves	512	64	8
Down toward Leaves	512	64	8
Up toward Root	128	1	128
Down from Root	128	1	128

Table 3.2: $UT_{64\times8}$ Bandwidth

Stack	Component	Available	Required Bandwidth	
Layer	Count	Bandwidth	Down Path	Up Path
0	64	512	512	0
1	64	512	0	384
2	48	384	0	256
3	32	256	0	128

Table 3.3: $UT_{64\times8}$ Vertical Through Bandwidth

Through Bandwidth

All of the vertical straight through connections between horizontal boards in the stack structure are contained on the routing components' package as described in Section 1.3.1. This limits the number of vertical routing conduits available for up or down routing connections which simply pass through a layer without being involved in the routing occurring on that layer. Each package provides sufficient through routing conduits for 8 bundles of 9 wires apiece. No vertical vias are needed for the downward routing path, except when the entire downward bandwidth must route through the upward routing level. The upward routing path requires through bandwidth on all the down routing levels. Available and required bandwidth is summarized in the Tables 3.3. Clearly, there is adequate vertical through bandwidth available for this scheme.

3.3.2 Tree Root

Two alternatives exist for terminating the fat-tree. It is possible simply to build the tree up to a desired size and leave the bandwidth out of the top of the topmost stage of unit trees unused. Alternately, we can build a capping level which utilizes the logical directio which would have routed further up the tree to route laterally to another down routing level.

The first option is the conceptually cleanest. All of the unit tree stacks will be identical. Routing will be identical on trees of all sizes.

The later option allows more size flexibility in some cases. Knowing the size of the stack becomes important to routing considerations when the root stack is capped in this manner. Stacks with a cap level will be slightly different from other stacks so that not all stacks will be identical; however, these capped stacks can differ only by the addition of an additional routing layer to the top of a standard unit tree stack.

Cap Level

Conceptually, the root layer constructed for a capped unit tree will be for the convergence of four unit trees; however, the components that compose this layer can be distributed across the four stacks. This distribution is necessary since we cannot make a single stack larger. At the top of a standard unit tree there are 128 channels that would go further toward the root and 128 coming from the root. With a down bandwidth of 128 into each of the 4 stacks and hence logical directions, we need a total of 64 routing components to compose the final level. One-fourth of these components can be placed in each stack. This appropriately gives 128 outputs from this new level to the 128 inputs to the down routing level which was formerly the top of each stack. Each stack will then have $\frac{128}{4} = 32$ connections through the capped root back to itself and 32 connections from each of the three adjacent stacks into its original top down routing level. Similarly, each stack will connec to 3 other stacks with 32 connections. Obviously, if the root level of unit trees is composed of more than 4 stacks, the 32 connection in each logical direction should be maximally distributed over all stacks composing each logical direction.

For the sake of clarity, this capped unit tree will be referenced as $UT_{64\times8c}$

3.3.3 Building a Fat-Tree From Unit Trees

Each bidelta leaf cluster has a bandwidth of $\frac{2 \cdot N_{leaf}}{3}$ into the fat-tree, where N_{leaf} is the number of processors in the leaf cluster. A single unit tree has a bandwidth of 8 in each of the 64 logical directions it distinguishes. As such, $\frac{N_{leaf}}{3 \cdot 4}$ unit trees are required to construct the smallest fat-tree which has only a single layer of unit trees. The next larger size can then be constructed by replicating this smallest structure 64 times and using another layer of unit trees to interconnect these 64 subtrees. Enough unit trees at this second level will be needed to match the bandwidth out of the top of the first stage of unit trees. This progression can be continued to build fat-trees arbitrarily large.

Examples

12K As an initial example, consider building a 12K processor machine. We can use a B_{192} leaf cluster for the leaf. The next level is then built from $\frac{192}{12} = 16$ unit trees. This gives $192 \cdot 64 = 12288$ processors using 64 B_{192} bidelta clusters and 16 $UT_{64\times 8}$ unit tree blocks.

48K Using a capping level at the top of a single layer of unit trees, we can support 48K processors, four times as many processors as the previous example. The number of unit trees simply needs to grow by a factor of four. Similarly, four times as many bidelta clusters are needed. Thus, this is constructed from 256 B_{192} clusters and 64 $UT_{64\times 8c}$ unit trees.

768K Adding a full second level of unit trees allows us to connect $192 \cdot 64^2 = 786K$ processors. This requires a bidelta leaf cluster for each 192 processors:

$$\frac{768K}{192} = 4K$$

The lowest level of unit trees is composed of unit trees given by the following:

$$\underbrace{\frac{768K}{192}}_{a} \cdot \overbrace{\underbrace{\frac{192}{12}}_{b} \cdot \underbrace{\frac{1}{64}}_{c}}^{d} = 1K$$

(a) is the number of bidelta clusters that connect to this level. (b) is the number of unit tree stacks necessary to satisfy the bandwidth for a single B_{192} bidelta block. (c) is the fraction of the unit trees from (b) that are actually used by a single bidelta block. (d) which is the product of (c) and (b) is the ratio of the number of bidelta clusters to unit tree stacks required to build this first level. The bandwidth out of the top of a unit tree is one-fourth the bandwidth into it. Thus the next level will only require one fourth as many unit trees:

$$\frac{1024}{4} = 256$$

This brings the total structure to 4096 B_{192} bidelta clusters and 1280 $UT_{64\times 8}$ unit tree stacks.

Generalizing

We can generalize network sizes and composition in terms of the number of unit tree layers used for network construction. Again, N_{leaf} is the number of processors composing a single bidelta leaf cluster. i is a parameter denoting the total number of stack levels, including the leaf cluster stack; i is constrained only to be a non-negative integer greater than one. The total number of processors supported by a network with $i-1$ unit tree layers all of type $UT_{64\times 8}$ is thus N_{total} as shown in Equation 3.1.

$$N_{total} = 64^{(i-1)} \cdot N_{leaf} \tag{3.1}$$

This requires one bidelta cluster for each N_{leaf} processors.

$$N_{bidelta} = \frac{N_{total}}{N_{leaf}} \qquad (3.2)$$

The first layer of unit trees is composed of $\left(\frac{N_{leaf}}{12} \cdot \frac{1}{64}\right)$ $UT_{64\times 8}$ unit trees for each bidelta cluster. Each successive layer requires one-fourth as many unit trees since the bandwidth out of the top of each unit tree is one-fourth the bandwidth into the bottom Thus the total number of $UT_{64\times 8}$ unit trees necessary to construct a network of size N_{total} is simply as given by Equation 3.3.

$$\begin{aligned} N_{unit} &= \sum_{j=1}^{i-1} \left(\frac{N_{total}}{N_{leaf}} \cdot \frac{N_{leaf}}{12} \cdot \frac{1}{64} \cdot \frac{1}{4^{(j-1)}} \right) \\ &= \left(\frac{N_{total}}{12 \cdot 64} \right) \left(\frac{1 - \left(\frac{1}{4}\right)^{(i-1)}}{1 - \frac{1}{4}} \right) = \left(\frac{N_{total}}{576} \right) \left(1 - 4^{(1-i)} \right) \end{aligned} \qquad (3.3)$$

Thus a network of N_{total} processors is constructed with $N_{bidelta}$ leaf clusters and N_{unit} $UT_{64\times 8}$ unit trees.

Using capped unit trees at the top level, the composition is slightly different. The total number of processors is greater by a factor of four because of the extra routing stage.

$$N_{total} = 4 \cdot 64^{(i-1)} \cdot N_{leaf} \qquad (3.4)$$

The number of bidelta leaf clusters needed is computed as before.

$$N_{bidelta} = \frac{N_{total}}{N_{leaf}} \qquad (3.5)$$

The number of stacks progresses the same. The difference here is that the top level is constructed out of $UT_{64\times 8c}$ unit trees. Thus the number of $UT_{64\times 8}$ unit trees, N_{unit} and $UT_{64\times 8c}$ unit trees, N_{cunit} are computed by Equations 3.6 and 3.7.

$$\begin{aligned} N_{unit} &= \sum_{j=1}^{i-2} \left(\frac{N_{total}}{N_{leaf}} \cdot \frac{N_{leaf}}{12} \cdot \frac{1}{64} \cdot \frac{1}{4^{(j-1)}} \right) = \left(\frac{N_{total}}{576} \right) \left(1 - 4^{(2-i)} \right) && (3.6) \\ N_{cunit} &= \frac{N_{total}}{768 \cdot (4^{i-2})} && (3.7) \end{aligned}$$

3.3.4 Alternative Unit Tree

It is also worthwhile to consider an alternative unit tree structure. In particular, this necessary if we wish to construct entirely fat-tree networks; that is if we wish to construct networks that have processors as leaves rather than bidelta clusters. Alone, the unit tree just described is inadequate for this purpose because it will always provide at least 8 connections in each logical direction. To match the connections to a single processor, we need a unit tree that provides two logical connections per direction. This unit tree can be used by itself to construct such a complete fat-tree network, or used only as the bottommost tree stage of a fat-tree network utilizing $UT_{64\times8}$ unit trees for the construction of the upper portion of the tree. For distinction, this unit tree will be referred to as $UT_{64\times2}$.

Fault Tolerance

To get the desired two outputs, and hence inputs, in each logical direction, we essentially need to scale down the size of the unit tree constructed by a factor of four from that of the $UT_{64\times8}$ unit tree. In naively scaling down the unit tree size, we encounter one problem at the final down routing stage. Each single routing component becomes critical in order to provide network connectivity to four processors; that is, unlike the component in other stages, if one of these components fail, four processors will become unreachable.

This identical sort of problem occurs in the bidelta clusters. As described in Sections 1.3 and 1.5, this was the reason for implementing the alternative configuration for RN1 as two separate 4×4 crossbars. With RN1 configured in this manner at the bottommost down routing stage, instead of having a single component that makes the final routing connection to 4 processors, we have 2 components that make the final connection to 8 processors. This way no single component is critical to the functionality of the network.

To avoid having a similar problem on the inputs, we also configure these as in the bidelta configuration. While each processor will use only a single connection into the network at a time to guarantee that the network is not overloaded, each processor has two network connections. Each of these network connection should be connected to different routing components. The processor then guarantees to only use one of the two network connections at a time. In this manner, no single component in the first up routing stage of the network is critical since a given processor can always originate a connection through the network through either of the two routing components to which it is connected.

Size

The entire $UT_{64\times2}$ ends up being one-quarter the size of the $UT_{64\times8}$ unit tree. A $UT_{64\times2}$ is thus composed of a total of 52 RN1 routing components. The stages are each composed of one-fourth as many routing components as the corresponding stages in the $UT_{64\times8}$ unit tree summarized in Table 3.4. The largest routing layer has 16 components. These can be arranged as 4×4 components in the horizontal plane. This makes each side of the $UT_{64\times2}$ roughly half the size of each side of the $UT_{64\times8}$ unit tree. Thus each side of the $UT_{64\times2}$ unit tree will be about one foot long. The number of stages is the same between the $UT_{64\times2}$ and $UT_{64\times8}$ unit trees so they will be the same height.

Characteristics

This composition gives the $UT_{64\times2}$ unit tree the characteristics shown in Tables 3.5.

Level	Routing Components
0	16
1	16
2	12
3	8

Table 3.4: $UT_{64\times 2}$ Component Summary

	Total Bandwidth	Total Logical Channels	Capacity
Up from Leaves	128	64	2
Down toward Leaves	128	64	2
Up toward Root	32	1	32
Down from Root	32	1	32

Table 3.5: $UT_{64\times 2}$ Bandwidth

Through Bandwidth

As with the $UT_{64\times 8}$ unit tree all vertical routing vias between routing levels in the stack structure are contained in the routing components. Since this is only a scaled down version of the $UT_{64\times 8}$ unit tree, we have no new problems with vertical through routing bandwidth.

Building Trees

Building fat trees with $UT_{64\times 2}$ unit trees is done in much the same manner as before. This unit tree can be used as the only kind of unit tree in constructing the fat-tree. Alternately, it can be used as the leaf node in place of the bidelta clusters and the $UT_{64\times 8}$ tree can be used to build the rest of the structure for the tree. Using $UT_{64\times 8}$ unit trees for the upper tree structure gives better routing performance and is hence preferable. This difference in routing performance arises from the usage of the alternate RN1 configuration in the final down routing stage of the $UT_{64\times 2}$ unit tree. This makes the $UT_{64\times 2}$'s routing performance slightly less desirable than that of the $UT_{64\times 8}$ unit tree where the last stage is configured normally.

Since there is a performance differential, its reasonable to only consider using the $UT_{64\times 2}$ unit trees at the bottommost layer of the fat-tree. The fat-trees so constructed are free to be any power of 64.

$$N_{total} = 64^{i} \qquad (3.8)$$

One $UT_{64\times 2}$ unit tree is needed for each 64 processors. Thus the total number needed, N_{funit}, is given by Equation 3.9.

$$N_{funit} = \frac{N_{total}}{64} \qquad (3.9)$$

The number of $UT_{64\times8}$ unit trees in the next level is determined by matching the total bandwidth out of the tops of the $UT_{64\times2}$ unit tree layer with the bandwidth into each $UT_{64\times8}$ unit tree. As before successive unit tree levels each require one-quarter the number of the unit trees as the preceding level. As such, Equation 3.10 gives the number of $UT_{64\times8}$ unit trees needed, N_{unit}.

$$\begin{aligned} N_{unit} &= \sum_{j=1}^{i-1}\left(\frac{N_{total}}{64}\cdot\left(\frac{32}{512}\right)\cdot\frac{1}{4^{(j-1)}}\right) = \sum_{j=1}^{i-1}\left(\frac{N_{total}}{256\cdot 4^{j}}\right) = \\ & \frac{4}{3}\left(1-\left(\frac{1}{4}\right)^{(i-1)}\right)\frac{N_{total}}{1024} = \left(1-4^{(1-i)}\right)\frac{N_{total}}{768} \end{aligned} \tag{3.10}$$

Example Using two levels of $UT_{64\times8}$ style unit trees on top of one level of $UT_{64\times2}$ style unit trees, the fat-tree network will support $64^3 = 256K$ processors. This requires unit trees as follows:

$$\begin{aligned} N_{funit} &= \frac{64^3}{64} = 64^2 = 4K \\ N_{unit} &= \left(\frac{64^3}{768}\right)\left(1-4^{(1-3)}\right) = 320 \end{aligned}$$

3.3.5 Wiring Details for Unit Trees

As described in Sections 1.3.2 and 3.2.2, in order to specify more than 8 bits of routing information, the first routing byte must be periodically stripped from the data stream. Within the unit tree structure, this operation should logically be performed at three places.

1. every fourth unit tree up on up the routing path
2. at the lateral crossovers between the up and down routing trees
3. upon entrance of a unit tree at the top down routing level when the connection crosses over higher up in the fat-tree.

The first case will only be necessary when more than 8 bits are needed to specify up routing. This becomes necessary only when we have more than $N_{leaf} = 64^4$ processors so will not be likely to be necessary in the near future. The need for lateral crossovers is described i Section 3.2.2. Recall that the wiring constraints of Section 2.3 recommend that inputs from different logical directions be distributed across as many different routing components as possible. As such, lateral crossover inputs will only make up a fraction of the inputs to a given routing component and hence be configured differently than the remaining inputs; fortunately, RN1 allows the the swallow property to be configured independently for each input. The logical place for stripping bytes in the downward path is at the entrance of each unit tree from higher in the fat-tree. This provides the most regular place for this function. This is somewhat non-optimal in that the swallow occurs between every 3 stages of down routing, as opposed to every 4; this means that the routing byte is being changed slightly more frequently than it could be in best case. As such the low two bits of each down routing byte are unused.

The rotation of bits between unit tree stages must be carefully arranged so that all paths through the fat-tree network rotate the routing bits equivalently, as described in Section 3.2.3. Obviously, each byte-wide routing path must be rotated by 2 bits following each up routing stage and following each down routing stage. Since there are only three down routing stages through each unit tree, following the last down routing stage there must actually be a four bit rotation so that the bits are correctly aligned to enter the following unit tree. The lateral crossover connections pose the least straightforward bi rotations. Each lateral crossover must guarantee that as the connection enters the down routing path, the bit rotation is consistent with the point at which the down routing path is entered. While this is easy enough to guarantee, this implies that the amount of rotation in each crossover will differ depending on the height of the unit tree in the fat-tree; this resul from the difference in the length of the up routing path traversed before encountering the crossover. This requirement unfortunately, forces unit trees to differ slightly depending upon which layer of the fat-tree they are implementing. The effect of this difference can be minimized by locating the necessary rotation difference entirely in the horizontal pc-routing board just above the up routing stage. This localizes the difference in unit trees to a single horizontal routing board. Of course, since the four stages of two bit rotations return the rotation to the original rotation, at most four such distinct boards will be necessary to build an arbitrarily large fat-tree.

3.3.6 Wire Accounting

There are a large number of wires entering and leaving each unit tree stack in order to properly connect the unit tree in the network. These wires, by necessity must go to a number of different locations. While each data path of 9 wires could go to a different destination, actually allowing them to do so would unnecessarily complicate the wiring pattern. To prevent this, we would like to group together reasonably sized wire bundles to interconnect unit trees and connect unit trees to leaf nodes.

$UT_{64\times8}$

The bandwidth entering and leaving the bottom of the $UT_{64\times8}$ unit tree stack is logically segregated into 64 directions each of which is 8 channels wide. Each channel is composed of 9 bits. There are equally many channels going both into the stack from below and downward out of the stack. Grouping this together, we have $2 \times 8 \times 9 = 144$ wires in each logical direction. It makes sense to use this is a the standard wire bundle size.

The bandwidth leaving the top of one of these unit trees is composes a single logical direction. However, since it will in turn be connecting to other $UT_{64\times8}$ trees, it makes most sense to use the same wire bundle size. Since there are only one-fourth the bandwidth exiting the top of the stack, there will only be 16 such bundles out of the top of a unit tree.

Unit Tree Type		Number of Bundles	Bundles Size	Total Wires
$UT_{64\times8}$	Bottom	64	144	9216
	Top	16	144	2304
$UT_{64\times8c}$	Bottom	64	144	9216
	Top	12	144	1728
$UT_{64\times2}$	Bottom	64	36	2304
	Top	4	144	576

Table 3.6: Unit Tree Wire Bundling for External Connections

$UT_{64\times8c}$

Since the capped unit tree, $UT_{64\times8c}$, is simply a variation on the $UT_{64\times8}$, it is very similar. The bandwidth into the bottom is identical to the $UT_{64\times8}$ case. The wires out of the top are different. In this case, each unit tree will potentially be connecting to 3 or more others. For consistency, bundles of 144 wires can be used here as well. There are three logical directions out of the top of a $UT_{64\times8c}$; each direction has with one-fourth of the total bandwidth out of the top of the $UT_{64\times8}$. This gives us 4 bundles of 144 wires for each of the 3 logical directions.

$UT_{64\times2}$

The bandwidth into the bottom of the $UT_{64\times2}$ unit tree is distributed as 64 pairs of channels. This means there are $2 \times 2 \times 9 = 36$ wires in each logical direction. Again, the bandwidth out of the top of the unit tree is one logical unit. This top bandwidth needs to be divided into bundles the same size as those out of the bottom of the $UT_{64\times8}$ unit trees to which this will be connecting. This means the top bandwidth should be broken down into 4 bundles of 144 wires each.

Unit Tree External Bundle Summary

Table 3.6 summarizes the bundles in and out of the unit trees discussed in this section.

3.4 Geometry

3.4.1 Basic Properties

From the preceding section, we see that the total number of unit trees needed to construct each level decreases by a factor of four on each successive level toward the tree root. Looking at the composition of each unit tree, we see a common convergence from 64 units of a given size at one stage to 16 such units at the next physical stage up the tree. The size of these units involved in the convergence increases from a single stack at the first stage by a factor of 16 for each successive stage.

Example Looking back at the 768K processor example of Section 3.3.3, the terminal level is composed of 4096 $\mathcal{B}_{192}$ stacks, the bottom tree level of 1024 $\mathcal{UT}_{64\times8}$ stacks, and the top tree level of 256 $\mathcal{UT}_{64\times8}$ stacks. The convergence at the first level is from 64 $\mathcal{B}_{192}$ stacks to a set of 16 $\mathcal{UT}_{64\times8}$ stacks. At the next level, these sets of 16 $\mathcal{UT}_{64\times8}$ stacks form the basic logical unit. Sixty-Four of these units, which is $64 \cdot 16 = 1024$ $\mathcal{UT}_{64\times8}$ stacks, converge to 16 of these sets of 16 $\mathcal{UT}_{64\times8}$ stacks at the root level. These $16 \cdot 16 = 256$ $\mathcal{UT}_{64\times8}$ stacks then form the root structure for the fat-tree.

3.4.2 Growth

There is no recursively repeatable, three-dimensional structure that keeps inter-stage interconnection distances constant. This can easily be seen by noting that:

- The number of components needing interconnect grows exponentially.
- Keeping inter-stage distances constant, the three-dimensional space of candidate locations for components is bounded by cubic growth.

Thus, there is no way to prevent inter-stage delays from growing between successive levels as the system is scaled up in size. At best, we can hope to keep the inter-stage delay growth down to a reasonable level.

Note, however, that since the number of processors grows as 64^i, the system grows very rapidly. As such, we need only accommodate a few stages of growth in order to build the networks of interesting size for the near future.

3.4.3 Hollow Cubes

A natural approach to accommodating this 4:1 convergence in a world limited to three-dimensions, is to build hollow cubes. If we select one side as the "top" of the cube, the four sides can accommodate four times the surface area of the top, and naturally four times the number of routing stacks of a given size. As such, the "sides" contain the converging stacks from one level, and the "top" contains the set of stacks at the next level up the tree to which the "side" stacks are converging. The remaining side will remain open, free of stacks. This last side could be used to shorten wires slightly farther, but utilizing it in t manner would decrease accessibility (see Section 3.4.5 for further discussion of this issue

To start, let us consider the first level of convergence of the fat-tree structure. We want to connect 64 stacks of a given size to some number of others. Particularly, we wish to connect: 64 leaf nodes $\frac{N_{Leaf}}{192}$ $\mathcal{UT}_{64\times8}$'s or 64 $\mathcal{UT}_{64\times2}$'s to 4 $\mathcal{UT}_{64\times8}$'s. Each side is made of 16 stacks of the leaf size. Each side is constructed by laying out the 16 constituent stacks into a 4×4 array. The top will be of comparable size depending on the relative size of the stacks at each fat-tree level. Figure 3.1 shows a hollow cube configuration in which the stacks at each stage are of equivalent size (*e.g.* this will be the case when the lower level is $\mathcal{B}_{192}$ leaf clusters and the top is $\mathcal{UT}_{64\times8}$ unit trees). Figure 3.2 shows a hollow cube configuration where the top level stacks are four times the size of the lower level (*e.g.* this would be case when on the first level of a full fat-tree network in which the top level was

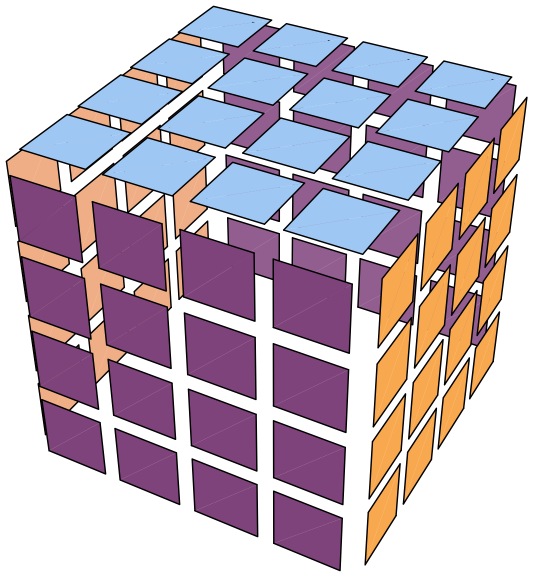
Figure 3.1: First Level Hollow Cube Geometry

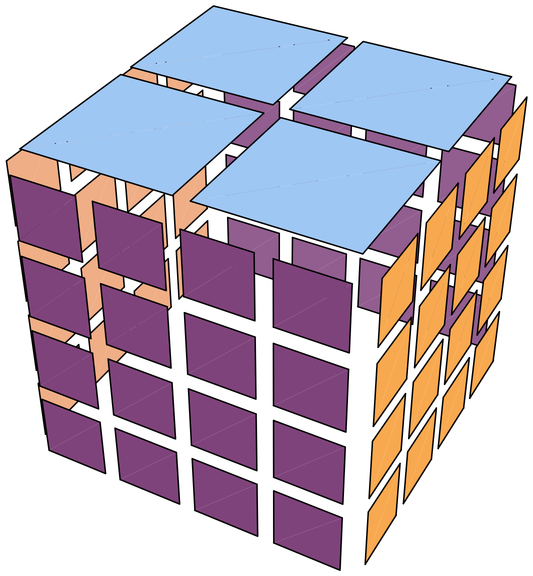
Figure 3.2: Hollow Cube with Top and Side Stacks of Different Sizes

constructed from $UT_{64\times8}$ unit trees and the bottommost stage was composed of $UT_{64\times2}$ unit trees).

At the next stage of convergence, we treat the unit we just created as the base unit. These can be arranged such that the tops of these hollow cube units are treated as the basic unit structure for this next stage. Then we arrange a plane of 4×4 of these for each side. A plane of unit trees of size equal to each of the sides is then used for the “top”. This

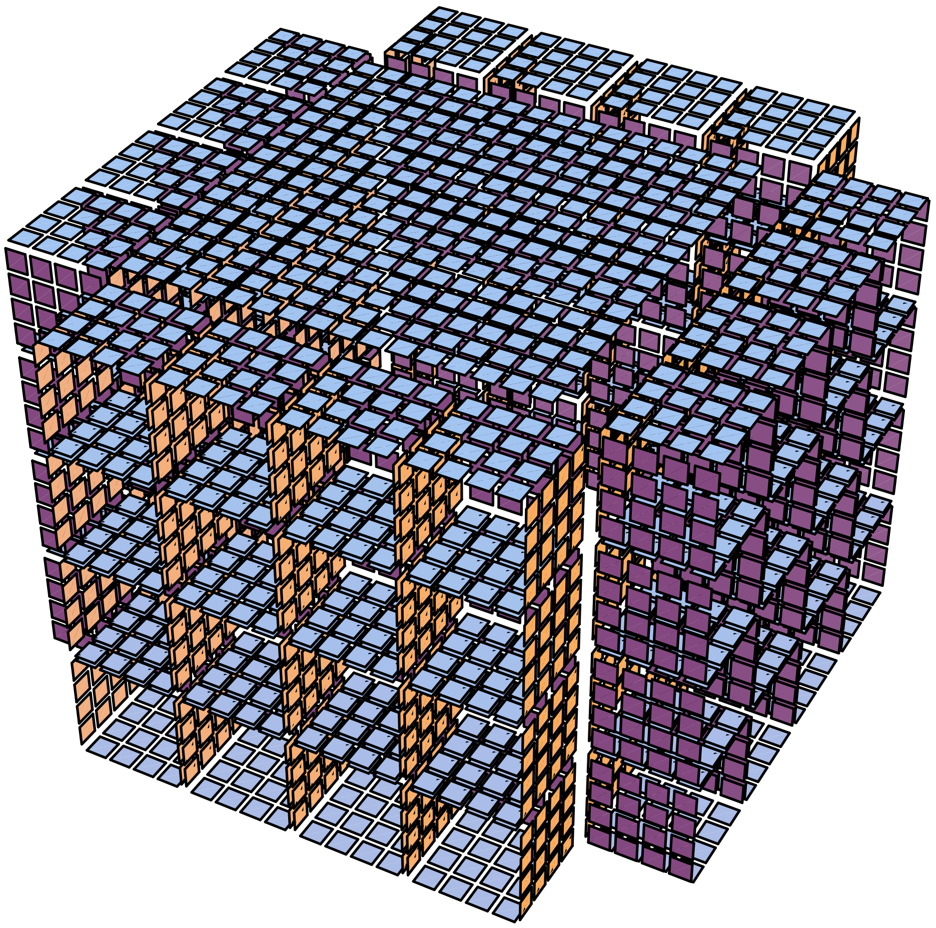

Figure 3.3: Second Level Hollow Cube Geometry

next stage is shown in Figure 3.3 as the natural extension of Figure 3.1. This progression can be continued in this manner *ad infinitum*.

3.4.4 Convergence Size

Beyond the first level, the size progression is fairly straightforward. The side size will grow by 4 at each successive stage since the size growth of 16 is accommodated in two dimensions.

At the first level, the size of convergence depends on the size of the leaf stacks used compared to the $IT_{64\times8}$'s to which they connect. Less $IT_{64\times8}$'s will be required the smaller the leaf stack size. In general 64 leaf stacks connect to $\frac{N_{leaf}}{12}$ $IT_{64\times8}$ stacks. From this we can determine the side size of the initial convergence square, or "top", and hence the side length for the hollow cube. The side size for the cube will be the greater of the side size of the "top" and the side size of the "sides". Let l_{ut} be the length of the side of a $IT_{64\times8}$ stack and l_{leaf} be the side length for leaf stack; these lengths will be as given by Equations 3.11 and 3.12, respectively. The length of a cube side, l_s, is then as given in Equation 3.13.

$$\begin{aligned} l_{ut} &= 2 \cdot 8 \cdot s_{chip} && (3.11) \\ l_{leaf} &= 2 \left\lceil \sqrt{\frac{N_{leaf}}{3}} \right\rceil s_{chip} && (3.12) \\ l_s &= \max\left(4 l_{leaf}, \left\lceil \sqrt{\frac{N_{leaf}}{12}} \right\rceil l_{ut}\right) \\ &= \max\left(16 s_{chip} \left\lceil \sqrt{\frac{N_{leaf}}{12}} \right\rceil, 8 s_{chip} \left\lceil \sqrt{\frac{N_{leaf}}{3}} \right\rceil\right) && (3.13) \end{aligned}$$

As described in Section 3.1, N_{leaf} will always be a multiple of three. For almost all sizes of interest, N_{leaf} will also be a multiple of four. As such, the component arrangements will always be a square number and the ceiling functions can be dropped. This reduces the two arguments of the max function to same expression. So Equation 3.13 reduces to Equation 3.14.

$$l_s = 8 s_{chip} \left(\sqrt{\frac{N_{leaf}}{3}} \right) \qquad (3.14)$$

The fact that the two lengths reduced to essentially the same expression was predictable since the bandwidth out of the top of a stack is one-quarter the bandwidth into its bottom. The four side bandwidths, which are all from the top of stacks match the bandwidth of the bottom of the stacks on the "top". The surface area is proportional to bandwidth in this configuration.

3.4.5 Features

While this hollow cube structure may not be the most compact structure, it does exhibit a number of nice properties.

It exposes the entire surfaces which are interconnected to one other. Since the bandwidth of the interconnection is largely surface area limited, this allows maximum exposure of the areas that need to be connected.

Since the structure is "hollow" the connections can be wired through the free-space in the center. Wiring through free-space in this manner, the maximum wire length is only $\sqrt{3}$ times the length of a side. This maximum length increases by roughly a factor of 4 every time we increase the number of processors by a factor of 64. This factor of four increase in size is due to the factor of four growth in side size for each successive level of the hollow cube.[1]

[1] As this continues, it will also be necessary to take the size of the "sides" of each square into account, making the increase somewhat more than a factor of four; for the sizes of present interest, this is not a real issue.

This structure is highly replicable. The progression described in Section 3.4.3 can be continued arbitrarily. The growth in wire lengths may, however, prove to be undesirable for very large structures.

In this form, the individual stacks are reasonably accessible for repair. Since the cubes are hollow, it is possible to get at any individual stack without moving any other stacks. This should allow repair and inspection without interfering with the bulk of the network operation. There may be some difficulty with accessibility due to the mass of wire inside the cube, but perhaps this can be minimized (see Section 3.5.3). The "missing" sixth wall of the cube can be used to allow entrance into the center of the structure for maintenance and repair.

3.4.6 Optimality

This solution seems practical while giving reasonable performance for the sizes of interest for the next decade. It is not known to be optimal for minimizing the inter-stage wiring distances. Finding an optimal solution that retains adequate accessibility to be of practica interest is still an open issue.

3.5 Hollow Cube Construction

3.5.1 Structure Size

Recall from Sections 3.3.1 and 3.3.4 that each 64×8 UFT tree is roughly two feet square and one and a half inches tall while 64×2 UFT is roughly one foot square. Arranging these, or similarly sized bidelta leaf clusters in 4×4 grids and building cubes as just describe in Section 3.4, creates structures with sides between 4 and 8 feet long. Following this progression one step further, we see that the side lengths grow to between 16 and 32 feet.

Clearly, networks of this size, in current technology, are room and building sized entities not something to put on your desk top.

3.5.2 Structure

To achieve the hollow cube structure of Section 3.4, it is necessary to construct "rooms" for networking. The "walls" of these "rooms" will be tiled with stacks in the 4×4 arrangement described, as will be the "ceiling". These stacks which tile the walls and ceiling will be arranged such that the tops of the wall stacks face into the room, and the bottoms of the ceiling stacks face into the room. The wall and ceiling thickness will be relatively smal since the stacks are thin. Current projections are for the stacks to be about 1.5″ thick. In some cases, it will be appropriate for the "ceiling" structure to be something other than the top or actual ceiling of the room, a review of Figure 3.3 will make this point clear. This should pose no additional problems.

To hold these stacks in place, the wall will be a structural grid. It will be similar to a raised floor where the stacks are analogous to tiles and the wall structure is analogous to the floor grid which supports the tiles. This grid provides sites for each of the stacks. It will then be possible to slide the stacks in and out of their sites, and "lock" them into place.

Due to the size and nature of this structure, the wall will also have to provide structural support. Adding an additional "real" wall to support the structure would prevent close packing and require the structure to be much larger.

The grid structure supporting the stacks will also need conduits to allow the fluorinert which cools the stacks to flow to the stacks. Similarly, conduits for power supply connections will also be needed. The actual fluorinert pumps and power supplies can be placed in adjacent rooms or on the sixth side of the cube.

3.5.3 Wiring

Interconnection signal wiring will be routed through the free-space in the center of the room

Wire Frames

Instead of connecting the wire bundles directly to each stack, these will be wired to a wiring frame. This wiring frame is a unit attached to the structural grid of the wall and will serve as a hatch-door on each routing side of a stack. All the wires to a stack are connected to the wiring frame. When the wiring frame is closed and locked into position, it will be compressed directly against the routing stack. The compression makes electrical contact between the stack and the connecting wires establishing signal flow. The wires are wired to the wiring frames instead of the stack to facilitate repair and ease of access to each stack. This scheme allows a stack to be changed without the need to disconnect and reconnect wires from 80 or more different sources. With the wiring frame, the frame can simply be opened to replace a stack.

The wire frame is "locked" into place when closed. This allows fluorinert and power to flow into the stack. When a frame is "unlocked" to remove a stack, several things should happen. The power to the stack should be cut. The fluorinert flow to the stack should also cease, and the fluorinert in the stack should be drained. Additionally, it may be necessary to terminate the transmission lines in some well-behaved manner for the sake of the rest of the network. When the frame is "re-locked", the fluorinert flow will need to be resumed. Once the system has had time to refill with fluorinert and get adequate power, it should be go through a reset sequence so that it comes online in a consistent manner.

Optical Connections

While the hollow cube provides adequate space for the wiring required, the wiring within the cube will still be quite formidable. Keeping track of the large quantity of wires will be a serious task, especially when problems occur with wiring conductivity.

An alternative that may be technically viable by the time one of these networks is actually constructed would be to use optical interconnections to provide the wiring through the cube. The cube is basically free-space so that each light beam need only be aimed at its appropriate receiver in order to effect interconnect. Since light beams will not interfe with each other, there will be no need to worry about the three-dimensional wiring problem of avoiding interference in the center of the cube. [Bergman 86] and [Wu 87] discuss early work to provide large scale optical interconnect for VLSI systems. Of particular interest is their use of a holographic optical element to direct optical beams for interconnections and the potential for adaptive and dynamic connection reconfiguration.

With optical interconnect, signal propagation time across the connection is less sensitiv to wiring materials. Long electrical connections will have delay proportional to both wire length and the permitivity of the material (ϵ_r) as given by Equation 3.15 where c is the speed of light.

$$t_{\text{wire}} = \frac{l_{\text{wire}}\sqrt{\epsilon_r}}{c} \qquad (3.15)$$

Long optical interconnection, in contrast, comes closer to the fundamental limit posed by the speed of light as given by Equation 3.16 [Kiamilev 89].

$$t_{\text{optical interconnect}} = \frac{l}{c} \qquad (3.16)$$

Without wires occupying volume in the center of the cube, the whole interior of the cube would be much more accessible for repair and maintenance.

One potential problem that would arise with optical interconnect is keeping the millions of laser connections in proper alignment. This could be a very hard task and alignment might prove very sensitive to repair operations within the cube. Adaptive alignment schemes would virtually be a necessity to make the fine alignment of a system of this size tractable. Adaptive alignment would allow the system to self adjust itself into proper configuration. Perhaps a mature version of the programmable optical interconnect of [Kiamilev 89] would provide a potential candidate for such adaptive alignment.

If utilized, the optical interconnect would, of course, be integrated as part of the wirin harness.

3.5.4 Maintenance

One very important issue for the hollow cube is that maintenance is possible, and that it is possible while the machine is in operation. With a system of this size, and necessarily expense, extensive down time will be expensive. Additionally, with a system this large, the number of failures per unit time is necessarily proportionally larger than in small systems. As a first line of defense against these problems the system is designed to be fault tolerant. When faults do occur, however, it will be necessary to fix them before they accumulate. It is very desirable to be able perform such repairs without completely disabling the machine.

The "unused" sixth side of the cube provides access into the interior of the cube. On this face a door or hatch can be placed to allow access into the internal structure. With the individual stacks laid out contiguously along the walls and ceiling, each is readily accessib without any need to displace any other stacks.

A single stack can be removed and replaced relatively quickly. With the wiring attaching to the wire frame, the whole operation of replacing a faulty stack can be moderately short. The faulty stack needs first to be located. Once located, its wiring-frame can be "unlocked". The stack can then be replaced and the wiring-frame "relocked" allowing the network to return to normal operation. The stack which has just been removed can then be taken elsewhere so that its faults can be identified and repaired.

Since there are redundant paths through the network using different intervening stacks for routing, it will be possible to remove an entire stack from the network while the rest of the network remains in operation. The practical effect of this will be that some fraction of the bandwidth will be "missing" or rather appear faulty; this is how it would looked anyway if much of the routing stack was faulty. The network should be wired such as to guarantee that each output from one level of the network can route through several different stacks at the next level. With this property and proper termination of the connections to the removed stack, the network will simply fail to route any connections which attempt to traverse the stack being replaced. The originating processor will then retry the failed connections later and either utilize another path through the network, or the same one after the replacement stack is powered up.

3.5.5 Technology Scaling

These considerations, and the sizes assumed throughout this work, are based mostly on current, usable technology. Certainly, as interconnect technology increases and packaging sizes diminish further, this structure can similarly decrease in size. This decrease in size linearly translate directly to improvements in the interconnect speed between components and stages.

One technology that will perhaps be feasible by the time systems of this size become of real practical interest is the kind of packaging used on the Cray III [Cray 89]. Utilizing thi kind of technology would allow the size of the component structures, and thus the resulting structure, to be diminished by about a factor of four to six. Similarly, prospects of wafer scale integration offer roughly the same size scaling advantages.

3.6 Long Wires

It should be clear from Section 3.5.1 that long wires will be required for interconnection between unit tree stacks. Here *long wires* are any wires whose length must be longer than the longest wire within a unit tree stack.

3.6.1 Strategy

The clock cycle on the unit tree and bidelta stacks will be optimized to be as short as possible given the operational speed of the routing component and the length of the longest wire in the unit tree. This means it will necessarily take multiple clock cycles for data to traverse these long wires between unit tree stacks. It is possible to place multiple data bit on a set of wires simultaneously, but we must be careful that the data is kept in proper phase with the clock in order to assure proper behaviour of the routing chip. The proper phase can be assured in either of a couple of ways. The interconnection wire lengths can be carefully chosen such that their delays are always integer multiples of length of the unit tree clock cycle. In this manner, the phase is preserved by guaranteeing the delay through the wire interconnection is sufficiently well behaved. Alternately, tapped delay line buffers can be used to insure that the data is presented in the proper phase relation with the unit tree clock. A scheme similar to [Rettberg 87] can be used for this purpose. If optical interconnect is used (Section 3.5.3), using wires to match interconnect delays to the phase of the data will not be possible. In such a case it would be necessary to use the tapped delay line alternative.

This scheme will necessarily increase the latency of a connection, but the increased latency is inevitable given the geometric constraints of routing. This scheme does manage to minimize the effects of scaling on latency by only slowing down those interconnection stages which must be long.

3.6.2 Requirements

The only additional requirements this scheme poses is on RN1, the routing component. In particularly, it only requires different behaviour from RN1 when a connection is turned around (see Section 1.3). Currently, RN1 expects to get valid data in the new direction of flow two clock cycles following the reception of a byte indicating it should turn the connection around. The addition of a single clock cycle's worth of wire delay causes this turn around time to be increased by two clock cycles; that is one additional clock cycle is required for the turn byte to propagate across the interconnect to the next routing component, and one additional clock cycle is required for the return data to propagate back across the connection. With no extra clock cycles of wire delay between RN1 routing components, this turn will occur in two clock cycles. With k clock cycles of wire delay between a pair of successive routing stages, the turn will occur in $2(k+1)$ clock cycles. In order for the turn sequence to function properly, the routing component at each end of such a long wire must be capable of dealing with the extra cycles. RN1 will need to know the number of delay cycles to expect and be able to deal with them accordingly.

The delay size will need to be configurable for ever input and output port on the routing component. With 8 input ports and 8 output ports, this makes for a total of 16 ports that need to be configured on a single RN1 component. It is necessary to be able to configure each input and output port separately as established in Section 2.3 to allow input and output ports from the same component to connect to interconnections of differing delay lengths. Additionally, to allow for the large range of delay values that must be accommodated for reasonably large networks[2], the delay length configuration will require multiple bits to specify the delay of a single input or output port. While this configuration information could be provided to the chip by configuration pins, it should be obvious that this would require the addition of quite a large number of signal pins. RN1 already has tight constraints on the number of pins that its package will support. Thus, an alternate means must be used to configure an RN1 routing component with this information.

One such alternative is to use UV programmable cells in the routing chip to store this configuration data. This would require the addition of only a few signal pins to facilitate the programming of the UV configuration cells. Cells would be programmed by initiating a program sequence and shifting the configuration data into the UV cells while the

[2] See Section 4.1 for projected wire and delay cycle lengths in systems of various sizes.

component is exposed to UV light. [Glasser 85] describes a technique for constructing UV programmable cells of this nature which is applicable to the one micron CMOS process in which RN1 is fabricated. This scheme has the drawback that a component cannot simply be replaced with another one off the shelf. The replacement component will first need to be configured before it can serve as a replacement.

A simple programming board can easily be built to program the configuration into a component as necessary.

With the additional configuration information provided to RN1, it must also be updated to deal appropriately with the configured delays of various lengths. This will required updating the finite state machine logic in the routing component to deal with varying delay lengths. This will be a minor change, but will necessitate adding additional state information to the FSMs. During the additional delay cycles, some reasonable data or pattern should be sent to the component not directly connected to the long wire so that the system is guaranteed to be well behaved. After the status and checksum bytes are sent to this component, the component on what was originally the sending side of the long wires, will need to send this additional data. This additional data could simply be a repetition of the status and checksum bytes.

3.7 Processors

Processing elements can be attached in a straightforward fashion to this network configured in the geometry described. Conceptually, each network terminal point will consist of a processor, memory, and a cache-controller (See Figure 1.1). A given number of processors will be associated with each leaf stack, whether a bidelta leaf cluster or a unit tree $\mathcal{UT}_{64,\frac{1}{2}}$. As previously noted, each leaf stack is rather thin. The processors and their associated components for a given leaf can also be arranged in a stack structure. The sides of this stack structure will be made the same size as the relevant routing stack. The processor stack can then be layered until it accommodates all the necessary components. Since the number of processors associated with a leaf routing stack is generally about the same as the number of routing components in the leaf stack, the processor stack structure will be of a thickness which is the same order of magnitude as that of the leaf routing stack. This processor stack can then be abutted directly to the leaf stack or even directly connected, making it part of the same physical stack. Since the stack structure tends to keep vertical distances short, this allows the processors to be within reasonable proximity of the routing network. Also since the stacks are thin, this will accommodate space for the processors by only making the "walls" of the hollow cube (Sections 3.4 and 3.5) slightly thicker. This additional thickness should not be significant enough to change the overall size of the hollow cube structure appreciably.

3.8 Routing Computation

The arrangement of components and structures described so far essentially determine the composition of a routing sequence for this network. In this section, this routing is summarized including a scheme for the proper generation of the routing sequence.

3.8.1 Distinguishing a Processor

Each leaf processor needs to have a unique specification so that it can be referenced. In a fat-tree structure, the logical specification for a processor is its "address" or locat relative to the root of the fat-tree. In the case of a full fat-tree, this will simply be the p down the fat-tree to the processor. In a hybrid fat-tree with bidelta clusters as leaves of the fat-tree network rather than individual processors, this address will be the path from the root and the location of the processor in the leaf cluster. Thus, a processor is specifie as, $L = C \circ P$, where C is the path from the fat-tree root to the leaf, and P is the location of the processor within the bidelta cluster. Recall from Section 3.1.1 that routing out of the bidelta leaf cluster stack is accomplished when the high two bits of the initial routing byte are 1's; as such, the high two bits of a processor specification will never be 11. It is easiest for routing if C is exactly the routing sequence necessary to get from the root to the desired leaf. Recall from Section 3.3.5 that the low two bits of each byte in a down routing sequence through unit trees is unused since only six bits of routing is performed on the down routing path through each unit tree. The value of these unused bits is irrelevant, but for clarity, they should probably be considered zero (0).

3.8.2 Routing

The routing sequence will be the series of bytes necessary to open a connection to a desired destination. In general this consists of three parts, the up routing sequence, R_U, the down routing sequence, R_D, and the routing within the bidelta cluster, R_C. Of course, full fat-trees will not have the R_C component. Each of the components of the routing sequence will occupy an integral number of bytes; a component will be padded to byte length when necessary. This keeps the portions of the routing sequence conceptually separated and is consistent with the previous specifications for the placement of swallows (Sections 3.3.5 and 3.1.2). The separation of the components of the routing sequence into their own bytes allows the routing bytes to be generated without any need to shift bits around within bytes. A complete routing sequence, R, is simply $R_U \circ R_D \circ R_C$.

3.8.3 Computing the Routing Sequence

With this configuration, we can compute the necessary routing sequence R with moderate ease. Unlike the bidelta case, it is necessary to know the source location in the network in order to determine an optimal routing sequence; this additional information is necessary in order to exploit locality in the fat-tree structure. Let $L_1 = C_1 \circ P_1$ be the source processor and $L_2 = C_2 \circ P_2$ be the destination processor.[3] The computation of the routing sequence then proceeds as follows:

1. $I = C_1 \oplus C_2$; that is I is the bit-wise logical exclusive-or of C_1 and C_2.
2. M = a bit vector with 0's for all leading zero bytes and 1's for all bytes from the leading non-zero byte to the end; that is M is a vector which marks all significant bytes.

[3] This degenerates to the specific case of full fat trees when $P_1 = P_2 = \epsilon$ (the empty string).

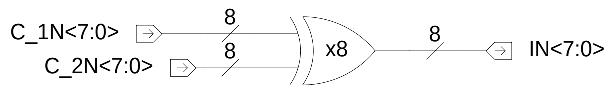

Figure 3.4: Byte-wide **xor** Slice.

3. j = location of highest-order non-zero bit in M (1 based).

4. $$O = \begin{cases} 10 & \text{if bit 6 or 7 is highest non-zero bit in high non-zero byte of I} \\ 01 & \text{if bit 4 or 5 is highest non-zero bit in high non-zero byte of I} \\ 00 & \text{if bit 2 or 3 is highest non-zero bit in high non-zero byte of I} \end{cases}$$

5. $RU = (11)^{j} \circ (o) \circ (00)^{((3-j) \bmod 4)}$; the trailing 0's simply suffice to pad RU to an integral byte quantity.

6. HD = low j bytes of C_2. This will, in fact, include unnecessary routing information. However, this quantity needs to be padded to an integral number of bytes in any case. When the bit-rotations are wired appropriately in the crossover routing paths, this allows HD to be generated with minimal calculations.

7. $RC = P_2$

Note that when M evaluates to all zero, it is the case that $C_1 = C_2$ and RU and HD are unnecessary; in this case, RC full specifies the routing to the final destination since the two processors are in the same leaf cluster.

3.8.4 Implementation of Computation

It is enlightening to consider possible structures to efficiently implement the computation just described. In order to talk about individual bytes and bits, let us adopt the following notation:

- B$<n>$ refers to the nth bit of B, with the lowest bit being bit zero.
- BN refers to the Nth byte of B, with the lowest bit being bit zero.
- B$N<n>$ thus refers to the nth bit of the Nth byte of B.

The intermediate quantity I, being an **xor**, is designed to be cheap to calculate.

The value M requires only the knowledge of which byte is the first to contain non-zero bits. This allows all the bits in each byte of I to be **or**'ed together for comparison. The result is a small bit vector. Since this quantity is then a small number of bits, it is possible to let the first non-zero bit stimulate the others so that a vector results with all the significant bytes marked with a 1 bit. Figure 3.5 shows this computation for a single bit in M

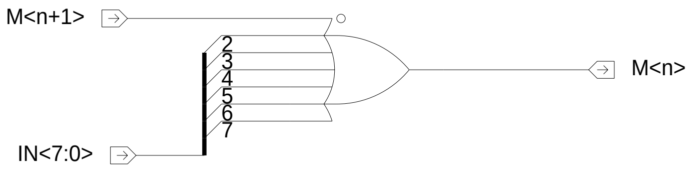

Figure 3.5: Calculation of a Single Bit of M

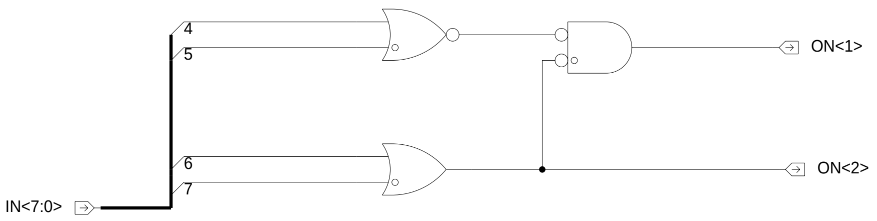

Figure 3.6: Calculation of a Single Bit of O

Both of the computation of I and M can easily be done either completely in parallel, all of C_1 and C_2 at once, or in a byte serial manner as is appropriate to a particular implementation.

In both cases the computation of O can be calculated on each byte in I simultaneously and in parallel with the computation of M. The value of M will determine which computation of O should be used. The computation of O can proceed similarly to M only with a granularity of pairs of bits instead of bytes. In fact it is necessary to look only at the hi two pairs of bits in order to determine O. In this case, instead of stimulating the lower bit(s) in the bit vector, we inhibit so that $ON<2:1>$ is the correct value for use in the construction of RU. See Figure 3.6 for an example of the calculation of O.

We can assume for the moment that RU will be a single byte quantity.[4] RU can be computed from M and O in a couple of gate delays as shown in Figure 3.7.

Since bytes are sent byte serial into the network, no computation is really done for HD. M identifies which byte of C_2 to start with when it is time to start sending HD into the network. Similarly, no computation is done for RC.

From these simple constructs, we can see that the computation of a routing sequence is an inexpensive operation and can be done in a reasonably small amount of time. A quick simulation of this implementation in a 1 micron CMOS process[5] calculates RU and M in less than 6ns.

[4] This will be the case until more than about 50 million processors are connected in this manner; this schema easily generalizes to the multiple byte case.

[5] the same processes in which RN1 is fabricated

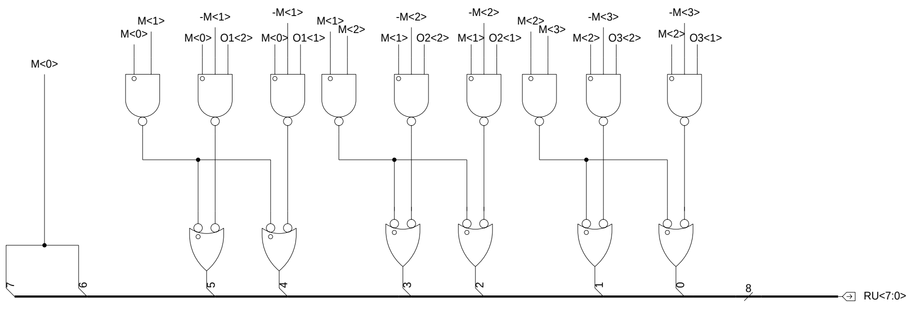

Figure 3.7: Calculation of RU from M and O

3.8.5 Example

Consider the fat-tree built with B_{192} leaves and two levels of $UT_{64,8}$ unit trees described in Section 3.3.3. This gives a fat-tree with 786,432 processors; C will be 2 bytes long and P will be one byte long. Let $L_1 = 0xCC0234$ and $L_2 = 0xD84267$.

1. $I = 0xCC02 \oplus 0xD842 = 0x1440$
2. $M = 0x03$
3. $j = 2$
4. $O1 = 10$; $O2 = 01$; since $M = 0x03$, $O2$ is the correct value for O
5. $RU = (11)^2 \circ (01) \circ (00)^1 = 11110100 = 0xF4$
6. $HD = 0xD842$
7. $RC = 0x67$

Thus the correct routing sequence is $RU \circ HD \circ RC = 0xF4D84267$. This will open a connection out of the bidelta stack, through the first stage unit tree, and into level 2 of the next unit tree. At that point, it has crossed over to the down path so the first byte, RU is swallowed. It routes down through that stack with the relevant portion of the next byte. That byte is then swallowed and the next byte, 0x42, is used to route back through the appropriate unit tree in the first stage. This final HD byte is swallowed and the 0x67 byte is used to route through the bidelta leaf cluster to the final destination.

4. Analysis

Chapters 2 and 3 described the construction of Transit fat-tree networks. This chapter quantifies some characteristics of the resulting networks. Considering fat-tree networks with both $UT_{64\times2}$ leaves and bidelta clusters leaves, a progression of networks from full bidelta networks to full fat-trees can be compared and analyzed. Section 4.1 describes the distribution of wires by length in hollow cube configurations. Section 4.2 extends the bidelta network structure beyond the single stack size for the purpose of comparisons. Section 4.3 summarizes network size freedoms for the various network configurations. It then uses size as a parameter to quantify hardware requirements and network latencies. Section 4.4 uses the development of the previous sections to provided some concrete network examples for comparison. Finally, Section 4.5 uses simple probabilistic models to provide basic routing statistics for the networks of Section 4.4.

4.1 Wire Lengths

Wire lengths are a significant concern in building large networks. As the system grows, the number of clock cycles required to route between unit trees becomes the dominant component of network latency. The hollow cube structure (Section 3.4) keeps wires moderately short considering the magnitude of the wire convergence that must occur. This section quantifies the distribution of wires by length in hollow cube geometries.

4.1.1 Worst Case

Utilizing free-space wiring in the hollow cube, the longest wires will be those that traverse the cube's diagonal. These wires will be $\sqrt{3}$ times the length of the cube side long. The length of the cube side depends on the size of convergence for the cube. As seen in Section 3.4.4, for standard hybrid fat-trees, the sides will be four bidelta leaf st
side lengths long at the first convergence and increase by a factor of four at each successive stage. Full fat-trees similarly progress from sides of length equal to four $UT_{64\times2}$ side lengths at the first convergence and progress by a factor of four at successive stages. The length of the side of a stack is determined by the maximum number of routing components on a single routing stage.

4.1.2 Distribution of Lengths

Only a small fraction of the total wires within a hollow cube are of the worst case length. Perhaps a more interesting metric is the distribution of wires by their length.

For simplicity, let us normalize wire lengths to the length of the stack sides. This allows the derived distributions to be applied generally regardless of stack size. The normalizin length, l_{stack}, will be the length of a side of the component stacks at the terminal level of the fat-tree. N_s will denote the number of stacks along the side of cube.

Each interconnection within the hollow cube connects from a "side" to the "top" of the cube (See Figures 3.1 and 3.2). Connection endpoints are evenly distributed across the two dimensions composing each the sides of the hollow cube; this results since each stack in the sides has the same distribution of interconnection to the top. Each of the stacks on the hollow cube sides will connect to endpoints which are evenly distributed across the surface of the top of the cube. Using this uniformity, we can easily characterize the wire length distributions.

We can start by decomposing the distribution into separate dimensional components, x, y, and z. Since these distributions are independent, we can then form the total distribution from the product of the dimensional distributions. To express the dimensional distributions, a couple of simple probability distribution are needed.

Uniform Unidimensional Distribution

One case of interest occurs when the distribution is completely uniform across the possible space of lengths. For such a case, we have a uniform distribution. Thus the distribution function is simply that of Equation 4.1.

$$p_x(x_0) = \begin{cases} \frac{1}{N_s} & 1 \le x_0 \le N_s \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Unifrom Difference Distribution

The other case of interest occurs when the distribution is that of the difference between two values picked randomly from the same uniform distribution. In this case, we have a distribution $p_x(x_0)$ as in Equation 4.1, and we want the distribution for the quantity $|x_0 - x_1|$ where x_0 and x_1 are described by p_x. This distribution is described by Equation 4.2.

$$p_{dx}(x_0) = \begin{cases} \frac{1}{N_s} & x_0 = 0 \le N_s \\ \frac{2(N_s - x_0)}{N_s^2} & 0 < x_0 < N_s \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Hollow Cube Distribution

With the simple distributions just described, the total distribution for wire lengths i
a hollow cube can be derived. The distance the wire must traverse in the vertical (z) dimension will be described by p_x since all interconnections connect from the top to some distance down the side of the cube. Similarly, the distance the wire must traverse "into" the cube, normal to the surface of the side, will be determined solely by the location of the destination on the top surface. This dimension, which for the current development will be called y[1], can also be described by p_x. In the remaining dimension, which will be referred

[1] *NB* In an absolute frame of reference, this dimension would be the x dimension for two faces and the y dimension for the two faces adjacent to those.

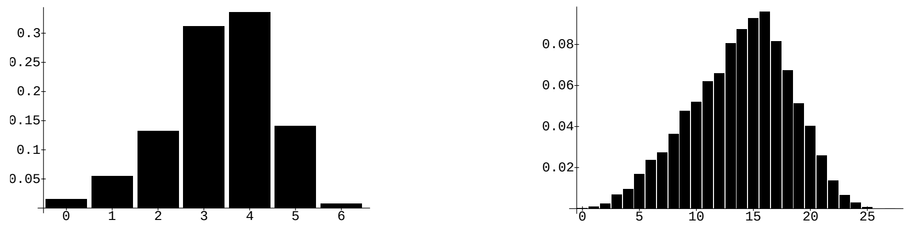

Figure 4.1: Distribution of Normalized Wire Lengths for $N_s = 4$ and 16, Respectively.

N_s	$E(l)$
4	3.4
16	13.9

Table 4.1: Expected Wire Lengths Normalized to Stack Size, l_{stack}

to as x; the distance traversed by the wire depends on the relative placement of both the source and destination of the wire and thus will be described by p_{dx}.

With this understanding of the dimensional distance distributions, we can describe the wire length distribution by Equation 4.3.

$$p_l(l_0) = \sum_{z=0}^{\max(l)} \sum_{y=0}^{\max(l)} \sum_{dx=0}^{\max(l)} \left[p_x(z) \cdot p_x(y) \cdot p_{dx}(dx) \cdot w_l(l_0, dx, y, z)\right] \tag{4.3}$$

$$\max(l) = \left\lceil \sqrt{3} N_s \right\rceil \tag{4.4}$$

The function w_l is simply used to determine whether or not to include the probability for a given (dx, y, z) combination in the summand is described by Equation 4.5.

$$w_l(l_0, dx, y, z) = \begin{cases} 1 & l_0 = \left\lceil \sqrt{dx^2 + y^2 + z^2} \right\rceil \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

The distribution, p_l, is easily computed for a given value of N_s and gives the resulting lengths in units of l_{stack}. Figure 4.1 shows the distributions of p_l for $N_s = 4$ and 16, the first two side lengths for hollow cubes. The expected, or average, wire lengths derived from this distribution are shown in Table 4.1.

4.1.3 Distribution by Delay

Once we have the normalized distributions, it is easy to convert these into delay distributions for a particular leaf stack size, l_{stack}. This, of course, is the important performance metric.

The previous distribution, p_l, can be converted into delay distribution when the clock cycle and stack size are given. With clock cycle, t_c, and l_{stack}, the number of delay cycles between stages is related to the wire delay by Equations 4.6.

$$n_{cycles}(l_0) = \left\lceil \frac{t_{delay}(l_0)}{t_c} \right\rceil \tag{4.6}$$

Using an appropriate model for the propagation time of a signal across a given length of wire, we can relate the number of delay cycles directly to the normalized wire length l_0. Equation 4.7 gives this relation assuming the signal is free to propagate at the speed of light, c, as would be the case for optical interconnect.

$$t_{delay}(l_0) = \frac{l_0 \cdot l_{stack}}{c} \tag{4.7}$$

With these relations, the wire length distribution can be converted into a distribution for the number of delay cycles between stages is computed as Equation 4.8.

$$p_n(n_0) = \sum_{0}^{\max(l)} \left[p_l(l_0) \cdot w_n(n_0, l_0) \right] \tag{4.8}$$

Once again a selection function, $w_n(n_0, l_0)$, is used to select the appropriate lengths which correspond to a given delay.

$$w_n(n_0, l_0) = \begin{cases} 1 & n_0 = \lceil n_{cycles}(l_0) \rceil \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Current projections make $t_c \approx 10$ns (Section 1.3). Recall that for $UT_{64/2}$'s, $l_{stack} \approx 1'$ (Section 3.3.4). Equation 3.12 gives a rough approximation of leaf stack size for bidelta clusters. Using these values, several distributions for a hybrid and full fat-tree hollowcu structure are shown in the following figures. Figure 4.2 is the delay cycle distribution in a hybrid fat-tree hollow cube using B_{192} leaves. Figure 4.3 describes the distribution for both hybrid fat-trees using B_{48} leaves and full fat-trees using $UT_{64/2}$ leaves. Figure 4.4 corresponds to the distribution for a hybrid fat-tree hollow cube with B_{12}. These figures all parallel Figure 4.1, but the distributions here are given in terms of cycles of dela Table 4.2 gives the expected number of delay cycles for these configurations.

4.2 Large Bidelta Networks

For comparison, it is necessary to consider bidelta networks scaled to sizes comparable to the fat-tree networks under consideration. However, since we cannot simply build arbitrarily large stacks (Section 1.4), large bidelta networks must be decomposed into multiple stack structures.

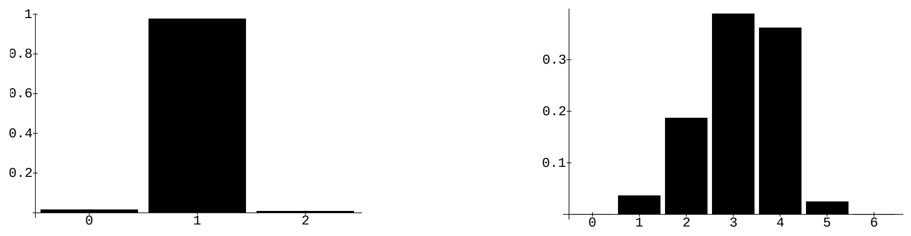

Figure 4.2: Distribution of Delay Cycles for $N_s=4$ and 16, respectively Using B_{192} Leaves.

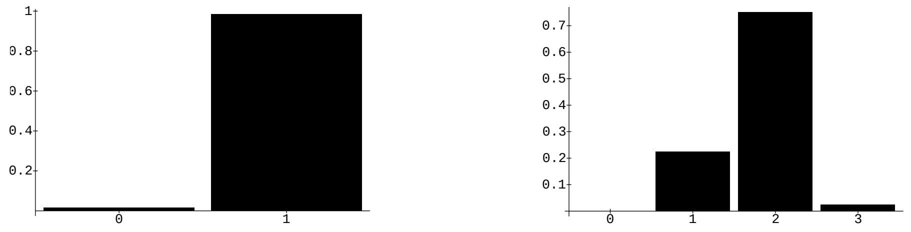

Figure 4.3: Distribution of Delay Cycles for $N_s=4$ and 16, respectively Using B_{48} Leaves.

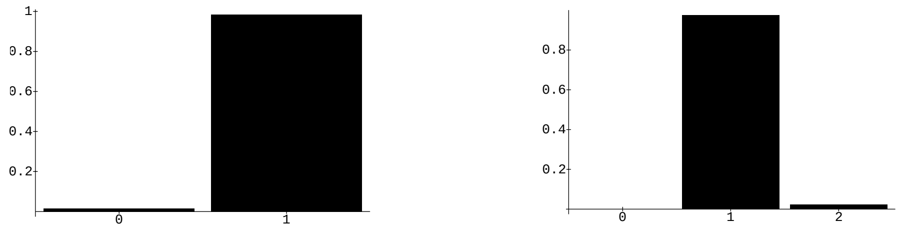

Figure 4.4: Distribution of Delay Cycles for $N_s=4$ and 16, respectively Using B_{12} Leaves.

Leaf Stack Type	l_{stack}	N_s	$E(n)$
B_{192}	$2'$	4	1.0
		16	3.2
B_{48}	$1'$	4	1.0
		16	1.8
B_{12}	$6''$	4	1.0
		16	1.0

Table 4.2: Expected Number of Delay Cycles

4.2.1 Bidelta Stacks

Perhaps, the ideal size for the constituent stacks is four routing stages. With four stages of routing, the stack distinguishes 256 logical destination. Thus each stack perform a byte's worth of routing; a fresh routing byte can be used to route through each stack.

The final size of such a configuration will then depend on the number of outputs provided in each of the logical directions. The smallest configuration would provide two outputs per logical direction. In general, such a configuration would only be desirable as the final routing stack composing a large bidelta network. In order to provide proper fault tolerance with two outputs in each logical direction, the final stage of the network must utilize the alternate RN1 configuration where each crossbar provides a single output per logical direction (Section 1.5). Since this routing stage's performance is inferior to that of t other stages, in order to achieve optimal routing performance only the final routing stack should concentrate the number of outputs in each direction to two.

Being limited to at most 64 components in each routing stage (Section 3.3.1), we are not free to consider bidelta stacks that both distinguish 256 different destinations and provide more than two outputs in each logical direction. In general if a stack distinguishes n logical destinations and has l outputs in each logical destination, N_c routing components will make up each routing stage as given by Equation 4.10. This relation follows trivially from the fact that each RN1 component has eight outputs.

$$N_c = \frac{n \cdot l}{8} \qquad (4.10)$$

As such, we are only free to consider stacks in which $n \cdot l \leq 512$. For clarity bidelta stacks with parameters n and l will be referenced as $B_{n\times l}$. From this discussion, we can conclude that $B_{256\times 2}$, $B_{64\times 2}$, and $B_{16\times 2}$ stacks should be used only as the final stage of stacks in large bidelta networks. The largest stack reasonable for use in forming the earlier network stages is the $B_{64\times 8}$.

4.2.2 Arranging Bidelta Stacks

The bidelta stacks are then treated as the standard routing units. They are arranged into stages in order to build larger bidelta networks. Within the bidelta stacks, the clock cycle can be as fast as a single stack constrains it to be. Additional stage delays will be incurred between stack stages as is the case with fat-trees because the wires will be long. However, this extra delay is only incurred between stack stages and at the final recycle path. The long wires can be dealt with as described in Section 3.6. This configuration will give better performance than uniformly slowing the clock rate down everywhere in order to accommodate the propagation delay across the long wire paths.

For replication, it will be necessary to place swallows between the stages of routing stacks. In this manner, a separate byte is used to route through each bidelta stack.

Since the number of inter-stage delays does not affect the total clock rate of the system, it makes sense to inter-wire the stacks in an indirect binary cube network style. This allows the wires at the first few stages to be relatively short. Successively longer wires ar used between successive stages of routing stacks. This structure can be laid out in two

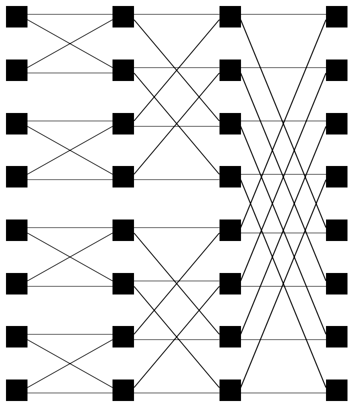
Figure 4.5: Indirect Binary Cube Topology

dimensions so that the critical geometric parameter at each stage is the square root of the number of stacks involved in the convergence; that is the subsets of the routing plane that will be interconnecting between successive levels will be squares of stacks. At the final stage, the convergence will be one big square; at earlier stages, there will be many smaller squares of convergence. Figure 4.5 shows the interconnection topology of an indirect binary cube using binary switching components. Using bidelta stacks for switching, each stack will switch in 16, 64, or 256 directions, and the inter-stage wiring will occur in two dimensions rather than one.

4.2.3 Wire Lengths

We can apply the analysis of Section 4.1 to analyze the distribution of wires by length in this scheme as well. Here we have a sequence of routing planes with the interconnect between planes. In this configuration, the wires are distributed with various displacements x and y direction and an essentially constant displacement between planes. Assuming the inter-plane displacement is negligible, we can get a feel for the distribution of wire lengt

Here the distribution of displacements necessary in the x and y directions are, to a first-order approximation, uniform difference distributions as described in Section 4.1.2. From this, we can easily derive the distribution of wire lengths. Let N_s be the length of the side of the square of convergence at the stage of interest, again normalized to stack size. p_l will again be our distribution function and w_l will be the selection function as before. Equations 4.11 through 4.13 summarize these relations.

$$p_l(l_0) = \sum_{dy=0}^{\max(l)} \sum_{dx=0}^{\max(l)} \left[p_{dx}(dx) \cdot p_{dx}(dy) \cdot w_l(l_0, dx, dy) \right] \quad (4.11)$$

$$\max(l) = \left\lceil \sqrt{2} N_s \right\rceil \quad (4.12)$$

$$w_l(l_0, dx, dy) = \begin{cases} 1 & l_0 = \left\lceil \sqrt{dx^2 + dy^2} \right\rceil \\ 0 & \text{otherwise} \end{cases} \quad (4.13)$$

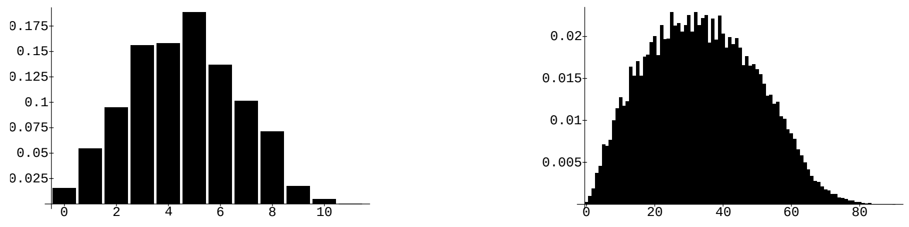

Figure 4.6: Distribution of Normalized Wire Lengths for $N_s = 8$ and 64, Respectively.

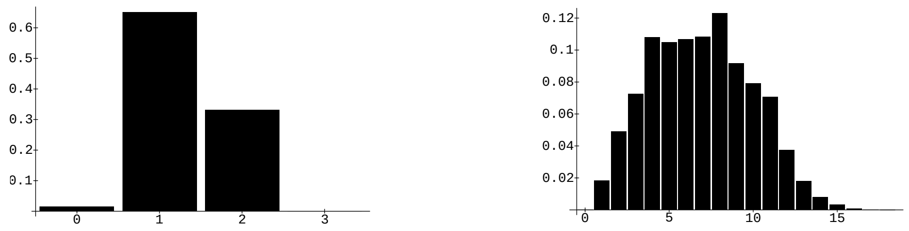

Figure 4.7: Distribution of Delay Cycles for $N_s = 8$ and 64, Respectively using $l \times 2^{l}$ Stacks.

Figure 4.6 shows the distributions for normalized side lengths of 8 and 64. If a large bidelta network were built with three stack stages, two built from $B_{64\times8}$ and the last from $B_{256\times2}$, N_s would be 8 between the first and second stage and 64 between the second and third. Normalized average wire lengths are 4.6 and 33.9 for $N_s = 8$ and 64, respectively.

4.2.4 Delay Cycles

The number of delay cycles required can be determined exactly as described in Section 4.1.3. Making the same assumptions as in Section 4.1.3, Figure 4.7 parallels Figure 4.6 in terms of delay cycle units. The side length for both the $B_{64\times8}$ and $B_{256\times2}$ stacks is two feet. For this configuration, the average number of delay cycles for is 1.3 when $N_s = 8$ and 6.9 when $N_s = 64$.

4.3 Network Characterizations

This section summarizes a few quantizations for various network characteristics param eterized by network size. Quantizations are provided for full fat-tree, hybrid fat-tree, a bidelta networks. This allows some comparison across the range of networks between full fat-tree and full bidelta networks. For each network, characterizations are provided for required hardware and network latency.

Table 4.3 summarizes most of the variables used in the remainder of this section.

N_p	number of processors
N_c	number of routing components
N_{leaf}	number of processors supported by a leaf stack
$N_{c_{leaf}}$	number of routing components in a bidelta leaf stack
$N_{bidelta}$	number of bidelta leaf clusters
N_{unit}	number of $UT_{64\times8}$ unit trees
N_{funit}	number of $UT_{64\times2}$ unit trees
i	a parameter describing selection freedom, i represents the number of stack stages in a network i must be a positive integer
j	a parameter describing selection freedom, j represents the number of routing stages in a bidelta stack
h	total number of routing stages in a bidelta network
s_{chip}	length of a side of the routing chip
t_c	clock period
l_{wire}	length of longest wire
t_{wire}	wire delay
c	speed of light
$n_{(m,o)}$	stage delays between stage m and o
L_{open}	latency opening connection from source to destination
$L_{connect}$	latency of an open connection from source to destination

N.B. For the following, latency is used to refer to the period of time between the time the message enters the network and the time the message arrives at the destination. The actual latency of a network operation will be a function of this metric and depending on the end to end protocol used. Connection latency, $L_{connect}$, differs from the latency opening a connection, L_{open}, due to the need to change routing bytes during the opening of a connection.

Table 4.3: Variable Summary

4.3.1 Full Fat-Tree

Recall from Section 3.3.4 that full fat-tree networks are built entirely from unit trees. Full fat-trees have processors as the ultimate leaves of the fat-tree structure.

Size

Full fat-tree networks are built with $UT_{64\times2}$ unit trees forming the bottommost stage of the fat-tree and $UT_{64\times8}$ unit trees forming the remainder of the tree structure. Equation 4.14 characterizes the sizes of constructible full fat-trees.

$$N_p = 64^i \tag{4.14}$$

A full fat-tree network of a given size will have $3i$ tree levels made from i stages of unit trees. The fat-tree will be composed of i up routing stages and $3i$ down routing stages.

Hardware Requirements

Each $UT_{64\times 8}$ unit tree requires 208 RN1 routing components (Section 3.3.1) while each $UT_{64\times 2}$ requires 52 RN1 components (Section 3.3.4). Equations 4.15 and 4.15 respectively describe the required number of $UT_{64\times 2}$ and $UT_{64\times 8}$ unit trees necessary to build full fat-trees of size N_p (Section 3.3.4).

$$
\begin{aligned}
N_{funit} &= \frac{N_p}{64} && (4.15)\\
N_{unit} &= \left(1 - 4^{(1-i)}\right)\frac{N_p}{768} && (4.16)
\end{aligned}
$$

Combining these requirements, we find the total number of routing components required as expressed in Equation 4.17.

$$
N_c = (52)\left(\frac{N_p}{64}\right) + (208)\left(1 - 4^{(1-i)}\right)\left(\frac{N_p}{768}\right) = \left(\frac{1}{16} + \frac{\left(1 - 4^{(1-i)}\right)}{48}\right) 13N_p \qquad (4.17)
$$

Latency

As described in Section 3.6 there will be some number of stage delays between stack stages. Let $n_{(m,o)}$ be the number of clock cycles of delay between stack stage m and stage o. Between any pair of stages, the values of $n_{(k,k+1)}$ and $n_{(k+1,k)}$ will be distributed as described in Section 4.1. Since there is a distribution of possible values for $n_{(k,k+1)}$ between the same stack stages, no single value describes this quantity. The reverse stage delays, $n_{(k+1,k)}$ will be distributed similarly; however, a particular $n_{(k+1,k)}$ need not be related to the corresponding $n_{(k,k+1)}$ utilized as part of the same interconnection.

To route through a given level of the full fat-tree, a connection must route through one up routing level for each 3 levels up the tree. The connection will then route through all the down levels. Between stack stages on both the up and down path, the route will suffer the additional stage delays due to the long wires between stages. The latency through level k of tree is given by Equation 4.18.

$$
L_{connect}(k) = \left(\left\lceil\frac{k}{3}\right\rceil + k + \sum_{j=1}^{\lceil\frac{k}{3}\rceil-1} n_{(j,j+1)} + \sum_{j=1}^{\lceil\frac{k}{3}\rceil-1} n_{(j+1,j)}\right) \times t_c \qquad (4.18)
$$

In opening a connection through level k, an additional cycle of delay is incurred between stacks on the down route in order to swallow the leading routing byte. Similarly, at least one up routing byte will need to be dropped in crossing over to the down routing path; every fourth stage between unit trees requires an additional stage of delay for the leading routing byte to be dropped. The latency to open a connection through level k of the fat-tree is summarized in Equation 4.19.

$$\begin{aligned} L_{open}(k) &= L_{connect} + \left(\left(\left\lceil \frac{k}{3} \right\rceil - 1\right) + \left\lceil \frac{k}{12} \right\rceil\right) \times t_c \\ &= \left(\left\lceil \frac{k}{12} \right\rceil + 2\left\lceil \frac{k}{3} \right\rceil + k - 1 + \sum_{j=1}^{\lceil \frac{k}{3} \rceil - 1} n_{(j,j+1)} + \sum_{j=1}^{\lceil \frac{k}{3} \rceil - 1} n_{(j+1,j)}\right) \times t_c \quad (4.19) \end{aligned}$$

The best-case routing occurs when the most locality can be exploited. This occurs when a processor connects to its nearest neighbor. In this case, the route occurs through level one of the fat-tree. Thus Equations 4.20 and 4.21 describe the best-case performance for a full fat-tree.

$$\begin{aligned} L_{connect} &= 2t_c & (4.20) \\ L_{open} &= 3t_c & (4.21) \end{aligned}$$

In the worst-case, the only common ancestor of the source and destination processors is the tree root. In this case, the connection must travel all i stages up to the root of the fat-tree and then all $3i$ stages back down to the destination leaf. Equations 4.22 and 4.23 give the worst-case latencies for the full fat-tree.

$$\begin{aligned} L_{connect} &= \left(4i + \sum_{j=1}^{i-1} n_{(j,j+1)} + \sum_{j=1}^{i-1} n_{(j+1,j)}\right) \times t_c & (4.22) \\ L_{open} &= \left(\left\lceil \frac{i}{4} \right\rceil + 5i - 1 + \sum_{j=1}^{i-1} n_{(j,j+1)} + \sum_{j=1}^{i-1} n_{(j+1,j)}\right) \times t_c & (4.23) \end{aligned}$$

4.3.2 Hybrid Fat-Tree

Hybrid fat-trees are built with bidelta leaf clusters forming the leaves of the fat-tre structure.

Size

The leaf stage of the network is built with the bidelta leaf clusters described in Section 3.1. These are interconnected with a fat-tree structure constructed from $UT_{64\times 8}$ trees. Each leaf cluster is built from j routing stages and supports N_{leaf} processors as described by Equation 4.24.

$$N_{leaf} = 3 \cdot 4^{(j-1)} \quad (4.24)$$

For technical reasons described in Section 3.1.3, $1 < j \leq 4$. Using $i-1$ stages of $UT_{64\times 8}$ unit trees, the total number of processors supported is thus given by Equation 4.25.

$$N_p = N_{leaf} \cdot 64^{(i-1)} = 3 \cdot 4^{(j-1)} \cdot 64^{(i-1)} \quad (4.25)$$

The fat-tree structure provides $i-1$ up routing stages and $3(i-1)$ tree levels and hence down routing stages.

Hardware Requirements

Each leaf is constructed from $N_{c_{leaf}}$ RN1 routing components as described by Equation 4.26.

$$N_{c_{leaf}} = 4^{(j-1)} + (j-1)3\cdot 4^{(j-2)} = (3j+1)4^{(j-2)} \tag{4.26}$$

The total number of bidelta leaf cluster required is described by Equation 4.27 (Section 3.3.3).

$$N_{bidelta} = \frac{N_p}{N_{leaf}} = 64^{(i-1)} \tag{4.27}$$

The number of unit trees required for a hybrid fat-tree with $i-1$ unit tree stages is given by Equation 4.28 (Section 3.3.3).

$$N_{unit} = \left(\frac{N_p}{576}\right)\left(1-4^{(1-i)}\right) \tag{4.28}$$

Each $UT_{64\times8}$ unit tree is constructed from 208 RN1 components (Section 3.3.1). Consolidating these relations, we get Equation 4.29 which described the total number of routing components in a hybrid fat-tree.

$$\begin{aligned} N_c &= (3j+1)4^{(j-2)}\cdot 64^{(i-1)} + 208\left(\frac{N_p}{576}\right)\left(1-4^{(1-i)}\right) \\ &= \left[(3j+1)4^{(j-2)} + \left(\frac{13}{12}\right)4^{(j-1)}\left(1-4^{(1-i)}\right)\right]64^{(i-1)} \end{aligned} \tag{4.29}$$

Latency

The best-case interconnect in the hybrid fat-tree occurs when a connection is made within the bidelta leaf cluster. When this level of locality can be exploited, the latency f
a connection is given by Equation 4.30.

$$L_{connect} = L_{open} = j \times t_c \tag{4.30}$$

Opening and continuing a connection are identical in this case because j is constrained to not exceed four due to current technology limitations.

When communications do need to occur between processors in different leaf clusters, connections must be made through the fat-tree. As connections are routed between stack stages, delay cycles will be incurred as described for the full fat-tree (Section 4.3.1). routing through some stage of the fat-tree, the connection first requires one cycle to route into the fat-tree network. The connection traverses one up routing stage every three stages up the tree it must go. Once the connection reaches the root of the smallest common sub-tree between the source and destination processor, the connection must then be routed down all the down routing stages to the desired bidelta leaf stack. Once the connection finally enters the destination leaf stack, there will be an additional j stages of routing withi
the leaf stack. Equation 4.31 summarizes the latency of a connection through level k of the

fat-tree network.

$$L_{connect}(k) = \left(1 + j + k + \left\lceil \frac{k}{3} \right\rceil + \sum_{m=1}^{\lceil \frac{k}{3} \rceil} n_{(m,m+1)} + \sum_{m=1}^{\lceil \frac{k}{3} \rceil} n_{(m+1,m)}\right) \times t_c \qquad (4.31)$$

When opening a connection, additional delay cycles are required due to the need to change routing bytes. The hybrid fat-tree requires the additional cycles for swallowing bytes on the up routing path, down routing path, and in the crossovers as described for full fat-trees (Section 4.3.1). Since two bits of routing are required to route into the fat-tree th swallow on the up path will occur one stack stage earlier in this configuration than in the full fat-tree configuration. The hybrid fat-tree will require an additional change of routin bytes when the connection enters the destination leaf cluster. Equation 4.32 combines these additional delays to describe the latency for opening a connection through level k of the fat-tree portion of a hybrid fat-tree.

$$L_{open}(k) = L_{connect}(k) + \left(\left\lceil \frac{k}{3} \right\rceil + \left\lceil \frac{k+3}{12} \right\rceil\right) \times t_c$$

$$L_{open}(k) = \left(1 + j + k + \left\lceil \frac{k+3}{12} \right\rceil + 2\left\lceil \frac{k}{3} \right\rceil + \sum_{m=1}^{\lceil \frac{k}{3} \rceil} n_{(m,m+1)} + \sum_{m=1}^{\lceil \frac{k}{3} \rceil} n_{(m+1,m)}\right) \times t_c \qquad (4.32)$$

Worst-case connections in the hybrid fat-tree occur when interconnection must be made through the top level of the top stage of unit trees. Here the connection is made into the fat-tree, up the $i-1$ up routing stages to the root, down the $3(i-1)$ stages to a leaf cluster, then across the j stages of the leaf cluster to the destination processor. Equations 4.33 and 4.34 describe the latency for this worst-case interconnection.

$$L_{connect} = \left(1 + 4(i-1) + j + \sum_{m=1}^{i-1} n_{(m,m+1)} + \sum_{m=1}^{i-1} n_{(m+1,m)}\right) \times t_c \qquad (4.33)$$

$$L_{open} = \left(1 + j + 5(i-1) + \left\lceil \frac{i}{4} \right\rceil + \sum_{m=1}^{i-1} n_{(m,m+1)} + \sum_{m=1}^{i-1} n_{(m+1,m)}\right) \times t_c \qquad (4.34)$$

4.3.3 Bidelta Network

Section 4.2 described the construction of large bidelta networks. This is a generalizatio of the Transit bidelta network to multiple stack sizes.

Size

The constituent stack stages are each constructed with j layers of routing. i stack stages can then be combined to construct the large bidelta network. The j's for each stage in a single network need not be identical across all stack stages. Thus the number of processors in such a network is described by Equation 4.35.

$$N_p = 4^{j_1} \cdot 4^{j_2} \cdots 4^{j_i} \qquad (4.35)$$

Recall from Section 4.2.1 that size and performance constraints limit j as given by Relations 4.36 and 4.37.

$$\text{for } m<i\text{: } 1 \le j_m \le 3 \tag{4.36}$$

$$2 \le j_i \le 4 \tag{4.37}$$

The total number of routing stages in this network is summarized in Equation 4.38.

$$h = j_1 + j_2 + \cdots + j_i = \log_4(N_p) \tag{4.38}$$

Hardware Requirements

The number of routing components needed can be determined by looking at the entire network without a need to do individual accounting at each stack stage. Considering the bandwidth in and out of the beginning and end of the network, Equation 4.39 gives the number of routing components needed at each routing stage.

$$N_{c_{stage}} = \frac{N_p}{4} \tag{4.39}$$

Combining this with Equation 4.38, the total number of RN1 routing components required for the bidelta network will be determined by Equation 4.40.

$$\begin{aligned} N_c &= h \times \frac{N_p}{4} \\ &= \frac{N_p \log_4(N_p)}{4} \end{aligned} \tag{4.40}$$

4.3.4 Latency

Delay stages are incurred between successive stack stages as previously described. Section 4.2.3 describes the distribution characteristics the extra delay cycles required betwe stack stages. Since the set of source processors is the same as the destination set, connections must loop-back from the end of the bidelta network to the beginning. Since this requires only translation in the vertical stack dimension, we will assume this loop-back can occur in a single clock cycle (t_{stack}). Note that since routing only occurs in one direction through the bidelta network, unlike the tree structures, the delays between stack stages is only incurred once in this structure.

With the exception of the number of delay cycles incurred between stages due to long wires, all paths through the bidelta network are the same length. Each path traverses the h routing stages described above. Inter-stage delay is incurred as well as a cycle for the final loop-back. As such, the connection latency for a bidelta network is described by Equation 4.41.

$$L_{connect} = \left(h + \sum_{k=1}^{i-1} n_{(k,k+1)} + 1 \right) \times t_{stack} \tag{4.41}$$

In opening a new connection through the bidelta network, additional latency occurs as a result of changing routing bytes. Routing bytes will be changed between routing stack

Number of Processors	Unit Tree Stacks		Total Components
64	1 $UT_{64\times2}$		52
4,096	64 $UT_{64\times2}$	4 $UT_{64\times8}$	4,160
262,144	4096 $UT_{64\times2}$	320 $UT_{64\times8}$	279,552

Table 4.4: Full Fat-Tree Hardware Requirements

stages as describe in Section 4.2.2. Due to the constraints placed on j, there will never be a need to change routing bytes within a bidelta stack. Equation 4.42 gives the latency required to open a connection through the bidelta network.

$$\begin{aligned} L_{open} &= L_{connect} + (i-1) \times t_{stack} \\ &= \left(h + i + \sum_{k=1}^{i-1} n_{(k,k+1)} \right) \times t_{stack} \end{aligned} \qquad (4.42)$$

4.4 Comparisons

In order to offer a clearer comparison between the various networks described herein, this section provides some concrete examples. Using the equations and assumptions from Section 4.3 and the geometric configurations described in Sections 3.4 and 4.2, representative numbers are determined for network size, composition, and latency. All values are characterized as describe in Section 4.3.

4.4.1 Full Fat-Tree

Tables 4.4 summarizes the hardware requirements for a few full fat-trees of interesting sizes. Table 4.5 characterizes the extreme latency cases for the full fat-trees described Table 4.4. The worst-case latency included here uses the worst-case wire delays between unit tree stages on both paths through the fat-tree and assumes the connection must be made through the tree root. Review Section 4.1 to get a feel for the distribution of latencies between the best and worst cases.

4.4.2 Hybrid Fat-Tree

Table 4.6 describes several sizes of hybrid fat-trees. Table 4.7 summarizes the latency range for these configurations. Just as for the full fat-trees the worst-case latency shown i for a connection through the tree root with the longest possible wires between stack stages on both the up and down tree traversal. Again, Section 4.1 describes the distributions of wire delays for configurations such as these.

Number of	Latency of Connect		Latency of Open	
Processors	Worst Case	Best Case	Worst Case	Best Case
64	40ns	20ns	50ns	30ns
4,096	100ns	20ns	120ns	30ns
262,144	200ns	20ns	230ns	30ns

Table 4.5: Full Fat-Tree Latencies

Number of Processors	Bidelta Stacks	Unit Tree Stacks	Total Components
768	64 B_{12}	1 $UT_{64\times8}$	656
3,072	64 B_{48}	4 $UT_{64\times8}$	3,392
12,288	64 B_{192}	16 $UT_{64\times8}$	16,640
49,152	4096 B_{12}	80 $UT_{64\times8}$	45,312
196,608	4096 B_{48}	320 $UT_{64\times8}$	230,400
786,432	4096 B_{192}	1280 $UT_{64\times8}$	1,118,208

Table 4.6: Hybird Fat-Tree Hardware Requirements

Number of	Latency of Connect		Latency of Open	
Processors	Worst Case	Best Case	Worst Case	Best Case
768	90ns	20ns	110ns	20ns
3,072	100ns	30ns	120ns	30ns
12,288	130ns	40ns	150ns	40ns
49,152	170ns	20ns	200ns	20ns
196,608	200ns	30ns	230ns	30ns
786,432	290ns	40ns	320ns	40ns

Table 4.7: Hybrid Fat-Tree Latencies

4.4.3 Full Bidelta

Tables 4.8 and 4.9 summarize the characteristics for some bidelta networks of comparable size to the full fat-tree and hybrid fat-tree configurations listed above. For this cas the worst-case latencies are for the case in which the wires are maximal length between stages. The best-case latencies assume a single cycle of delay due to inter-stage wiring. Section 4.2.3 describes the delay cycle distributions for this configuration.

Number of Processors	**Bidelta Stacks**	**Total Components**
64	$B_{64\times2}$	48
256	$B_{256\times2}$	256
1,024	$B_{64\times8}$ $B_{16\times2}$	1,280
4,096	$B_{64\times8}$ $B_{64\times2}$	6,144
16,384	$B_{64\times8}$ $B_{256\times2}$	28,672
65,536	$B_{64\times8}$ $B_{64\times8}$ $B_{16\times2}$	131,072
262,144	$B_{64\times8}$ $B_{64\times8}$ $B_{64\times2}$	589,824
1,048,576	$B_{64\times8}$ $B_{64\times8}$ $B_{256\times2}$	2,621,440

Table 4.8: Bidelta Network Hardware Requirements

Number of	**Latency of Connect**		**Latency of Open**	
Processors	**Worst Case**	**Best Case**	**Worst Case**	**Best Case**
64	30ns	30ns	30ns	30ns
256	40ns	40ns	40ns	40ns
1,024	70ns	70ns	80ns	80ns
4,096	90ns	80ns	100ns	90ns
16,384	110ns	90ns	120ns	100ns
65,536	160ns	110ns	180ns	130ns
262,144	210ns	120ns	230ns	140ns
1,048,576	310ns	130ns	330ns	150ns

Table 4.9: Bidelta Network Latencies

4.5 Routing Statistics

Using the simply probabilistic models for routing analysis developed in [Knight 90], this section provides some basic routing statistics for the network topologies describe
here. Statistics are provided for all of the configurations detailed in the previous sectio
All statistics summarized here are based on the network model presented in Section 1.2. This means that each processor will use at most one of its two inputs into to the network at a given point in time. The probabilistic models used to derive these statistics simply characterize routing probability in terms of network loading. The effect of input queuing at each source is not modeled.

Traffic distribution will have a considerable effect on the actual performance of any of these networks. The fat-tree structured networks require considerable communication locality in order to perform optimally. Each network is considered with the loading distributions for which it is most favorable. Along with the routing statistics, this section provid characterizations for the locality required by each network configuration to achieve near optimal performance.

4.5.1 Full Fat-Tree

A full fat-tree network is optimized to take advantage of locality. The actual construction and bandwidth allocation determine the locality characteristics necessary to utilize th network most effectively. The optimal loading case occurs when each connection through an RN1 routing component is equally likely to be made through each of the component's outputs. Within a unit tree structure, this means that a connection is equally likely to crossover at each lateral crossover stage. In a unit tree stage other than the root, the connection further up the tree will also be utilized with equal likelihood as each lateral crossover. Takin these two considerations together, the distribution of connections through each tree level in the same unit tree is roughly equal; the distribution of connections through unit tree stages drops off by a factor of four at each successive stage toward the root.

There are two ways of looking at the locality distribution. As described, we can think of the distribution in terms of the probability of routing through a given height in the fat-tree. This is the probability, $P_{any}(n)$, that a processor will need to communicate with *any* processor a fixed distance, n, away from the source processor. The probability that the processor will need to communicate with *a particular* processor a fixed distance form the source processor is an alternate way of viewing this locality. This probability, $P_{particular}(n)$, is simply the previously mentioned probability normalized by the number of processor that are located the fixed distance away from the source processor. Table 4.10 through 4.12 summarizes the locality distributions assumed for the full fat-trees considered here in term of these locality measures.

With these locality models, Figures 4.8 through 4.10 show the routing statistics on each of these full fat-trees. Each graph includes a separate curve for the routing statistic throug each level of the fat-tree. The topmost curve characterizes the statistics for a connectio through the first level of the fat-tree. Successive curves down the graph give routing statistics for routing through successive tree levels; the bottommost curve characterizes routing through the tree root.

Figure 4.11 shows the normalized routing probabilities for these three sizes of full fat trees. The normalized statistics are roughly identical for the three sizes, so the individu normalized curves are indistinguishable. The normalized probabilities give an idea of overal network performance under the assumed loading conditions. The normalized probabilities are determined by weighting the probability of routing through a given tree level by the probability that a connection can successfully be made through that tree level as summarized in Equations 4.43.

$$P_{norm} = \sum_{1}^{\max(n)} P_{any}(n) \cdot P_{route}(n) \qquad (4.43)$$

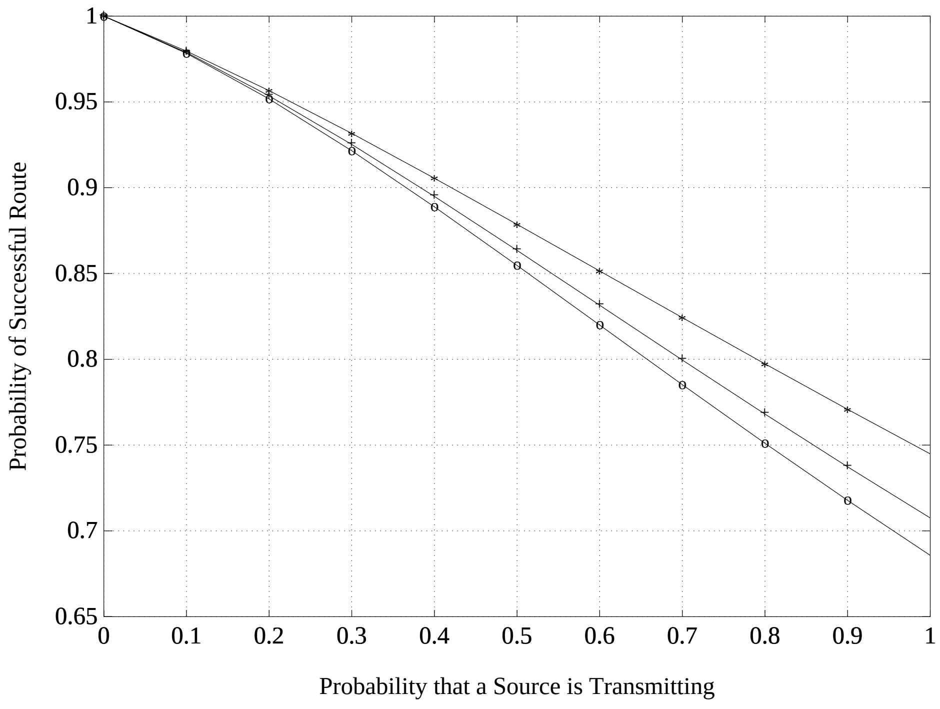

Figure 4.8: Routing Statistics for Full Fat-Tree (64 processors)

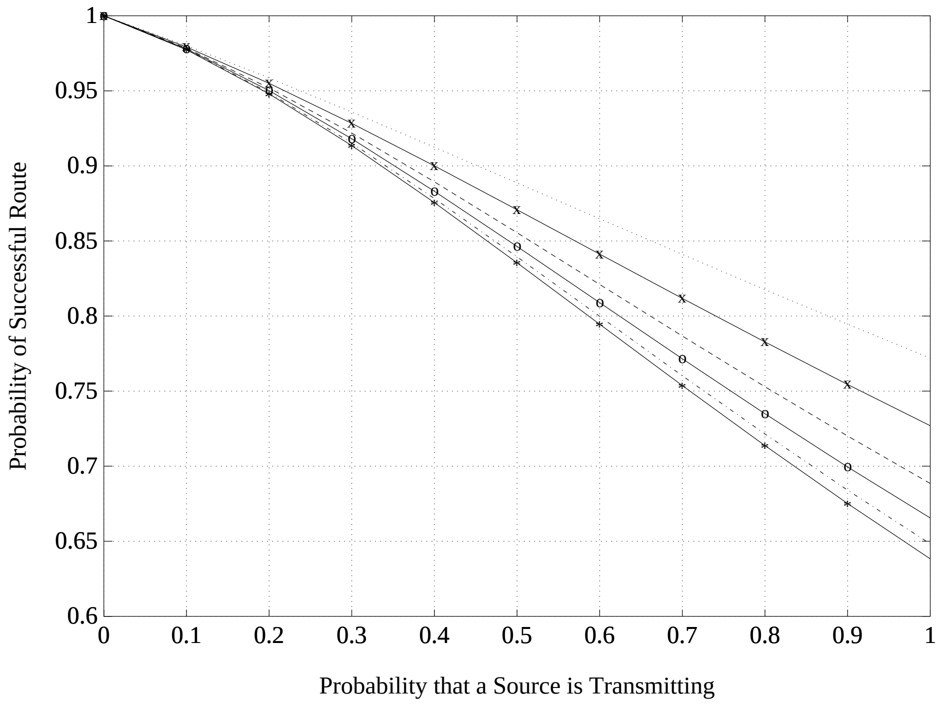

Figure 4.9: Routing Statistics for Full Fat-Tree (4096 processors)

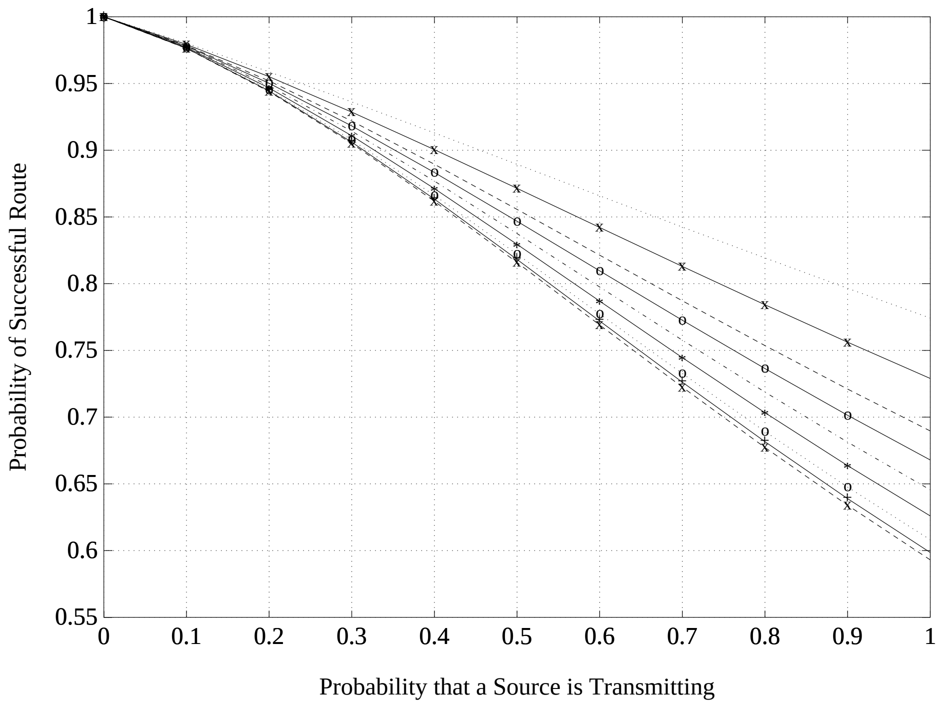

Figure 4.10: Routing Statistics for Full Fat-Tree (262144 processors)

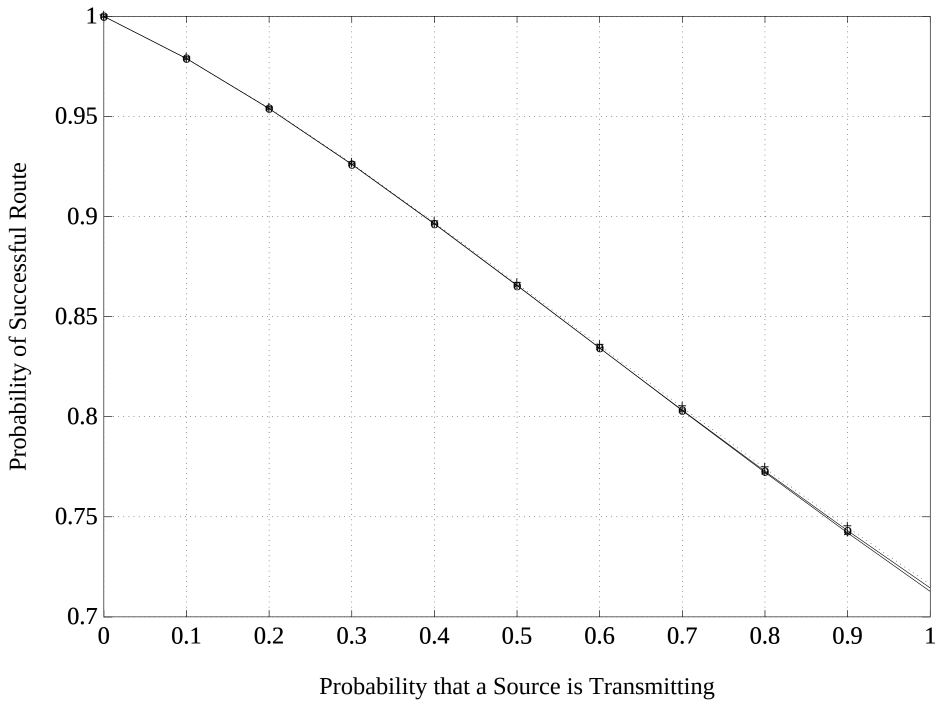

Figure 4.11: Normalized Routing Statistics for Full Fat-Trees

n	$P_{any}(n)$	$P_{particular}(n)$
1	0.333	0.111
2	0.333	2.78×10^{2}
3	0.333	6.94×10^{3}

Table 4.10: Locallity Structure of Full Fat-Trees (64 Processors)

n	$P_{any}(n)$	$P_{particular}(n)$
1	0.278	9.27×10^{2}
2	0.278	2.32×10^{2}
3	0.278	5.79×10^{3}
4	5.53×10^{2}	2.88×10^{4}
5	5.53×10^{2}	7.20×10^{5}
6	5.53×10^{2}	1.80×10^{5}

Table 4.11: Locallity Structure of Full Fat-Trees (4096 Processors)

n	$P_{any}(n)$	$P_{particular}(n)$
1	0.274	9.13×10^{2}
2	0.274	2.28×10^{2}
3	0.274	5.71×10^{3}
4	3.17×10^{2}	1.65×10^{4}
5	3.17×10^{2}	4.13×10^{5}
6	3.17×10^{2}	1.03×10^{5}
7	1.06×10^{2}	8.62×10^{7}
8	1.06×10^{2}	2.16×10^{7}
9	1.06×10^{2}	5.39×10^{8}

Table 4.12: Locallity Structure of Full Fat-Trees (262144 Processors)

4.5.2 Hybrid Fat-Tree

The hybrid fat-tree also takes advantage of locality much like the full fat-tree. Since only one-quarter of the bandwidth out of the first stage connects to the fat-tree structure, optimal performance occurs when three-quarters of the processor initiated traffic is local to the bidelta leaf clusters. Within the leaf, the distribution of traffic is uniform, that is, i equally likely to need to connect to any of the processors within the leaf. For connections through the tree, the arrangement is identical to that of a full fat-tree so the distributic of requests by tree level will progress in the same manner. Tables 4.13 and 4.14 summarize the distributions assumed for the hybrid fat-tree network traffic.

n	$P_{any}(n)$	$P_{particular}(n)$		
		B_{12}	B_{48}	B_{12}
0 (leaf)	0.75	6.82×10^{2}	1.60×10^{2}	3.93×10^{3}
1	8.33×10^{2}	2.31×10^{3}	5.78×10^{4}	1.45×10^{4}
2	8.33×10^{2}	5.78×10^{4}	1.45×10^{4}	3.62×10^{5}
3	8.33×10^{2}	1.45×10^{4}	3.62×10^{5}	9.04×10^{6}

Table 4.13: Locallity Structure of Hybrid Fat-Trees (Single Unit Tree Stage)

n	$P_{any}(n)$	$P_{particular}(n)$		
		B_{12}	B_{48}	B_{12}
0 (leaf)	0.75	6.82×10^{2}	1.60×10^{2}	3.93×10^{3}
1	6.95×10^{2}	1.93×10^{3}	4.83×10^{4}	1.21×10^{4}
2	6.95×10^{2}	4.83×10^{4}	1.21×10^{4}	3.02×10^{5}
3	6.95×10^{2}	1.21×10^{4}	3.02×10^{5}	7.54×10^{6}
4	6.95×10^{2}	3.02×10^{5}	7.54×10^{6}	1.89×10^{6}
5	6.95×10^{2}	7.54×10^{6}	1.89×10^{6}	4.71×10^{7}
6	6.95×10^{2}	1.89×10^{6}	4.71×10^{7}	1.18×10^{7}

Table 4.14: Locallity Structure of Full Fat-Trees (Two Unit Tree Stages)

Figures 4.12 through 4.17 show the routing statistics for the hybrid fat-trees described in Section 4.4.2. The first three graphs are for the configurations with one stage of unit trees, while the later three graphs are for the two tree stage configurations. The topmost curve in each graph represents the routing probabilities within the bidelta cluster. Each successive curve down a graph shows routing statistics for connecting through a successively higher level in the tree structure.

Figure 4.18 shows the normalized routing probabilities for these six hybrid fat-tree configurations. The normalized statistics for hybrid fat-trees with one or two unit tree stages coincide as long as the size of their bidelta leaf clusters is identical; that is normalized statistics differ only by the size of the bidelta leaf cluster. The topmost curve shows statistics for hybrid fat-trees using B_{12} bidelta leaves, the middle for B_{48} bidelta leaves, and the bottom for B_{192} leaves.

4.5.3 Full Bidelta

To a processor on a full bidelta network, all destinations are equally distant. The routing to all destinations is topologically identical. The bidelta network will provide i best overall performance when message traffic is uniformly distributed across all processors; that is, its most favorable loading condition occurs when each source will open a connection to all destinations with equal likelihood. This flat, random distribution of connections i

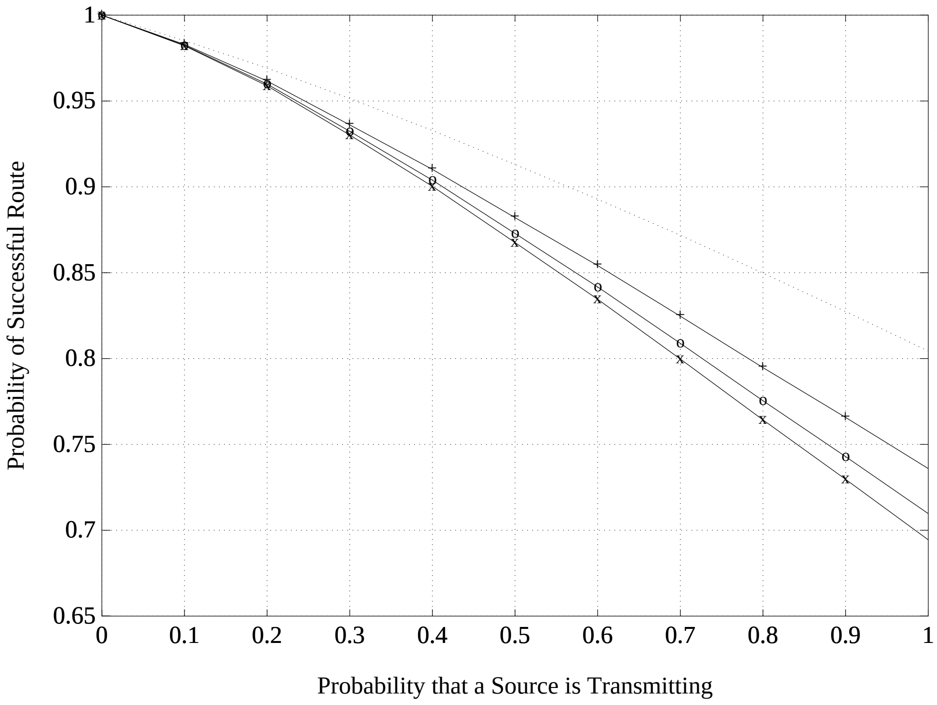

Figure 4.12: Routing Statistics for Hybrid Fat-Tree (768 processors)

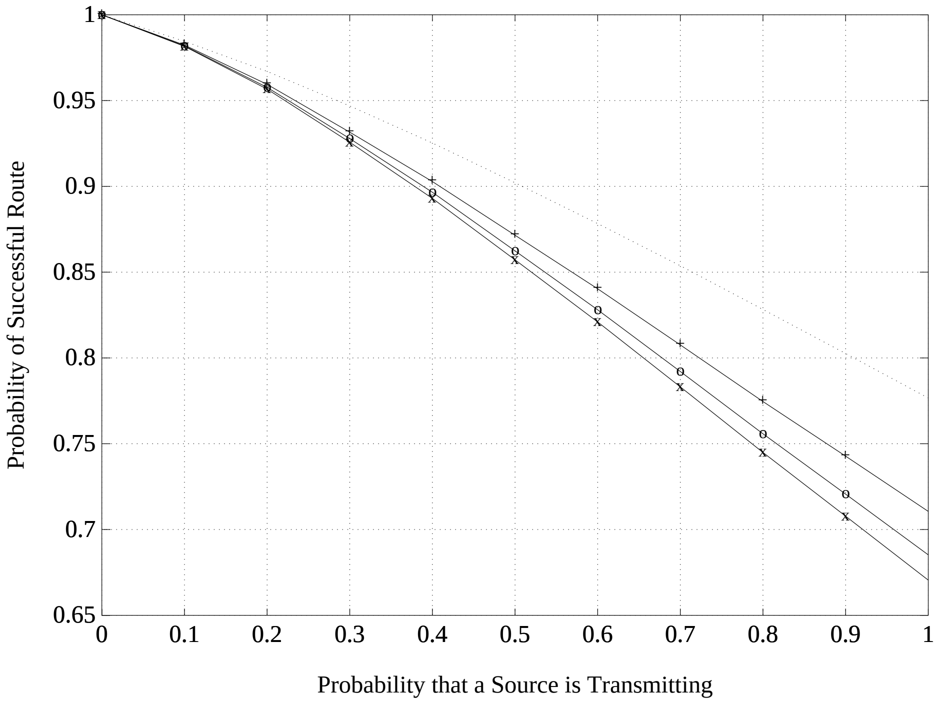

Figure 4.13: Routing Statistics for Hybrid Fat-Tree (3072 processors)

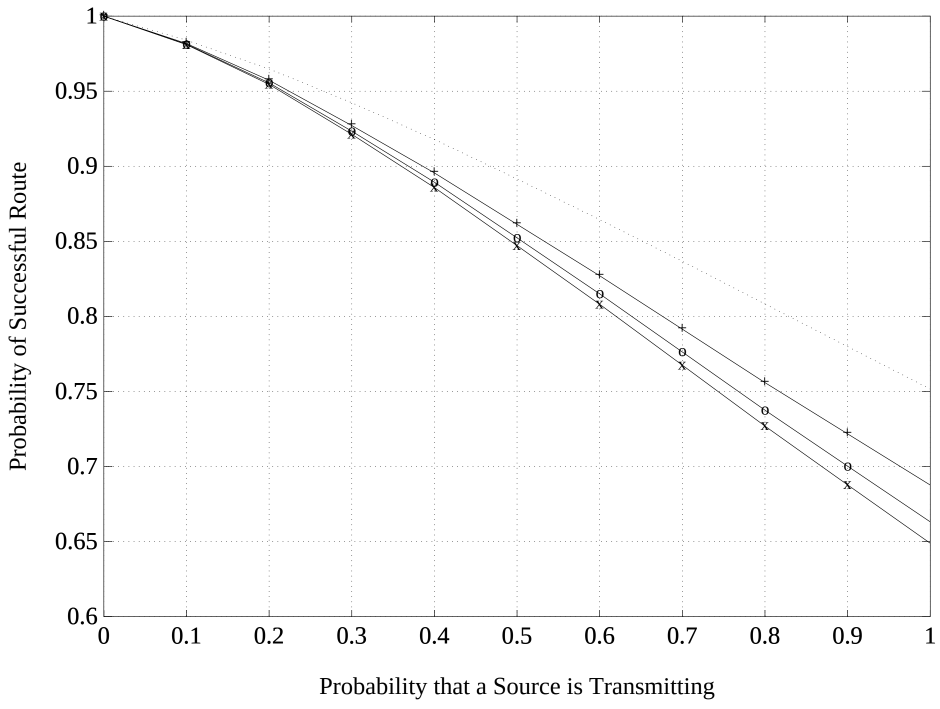

Figure 4.14: Routing Statistics for Hybrid Fat-Tree (12288 processors)

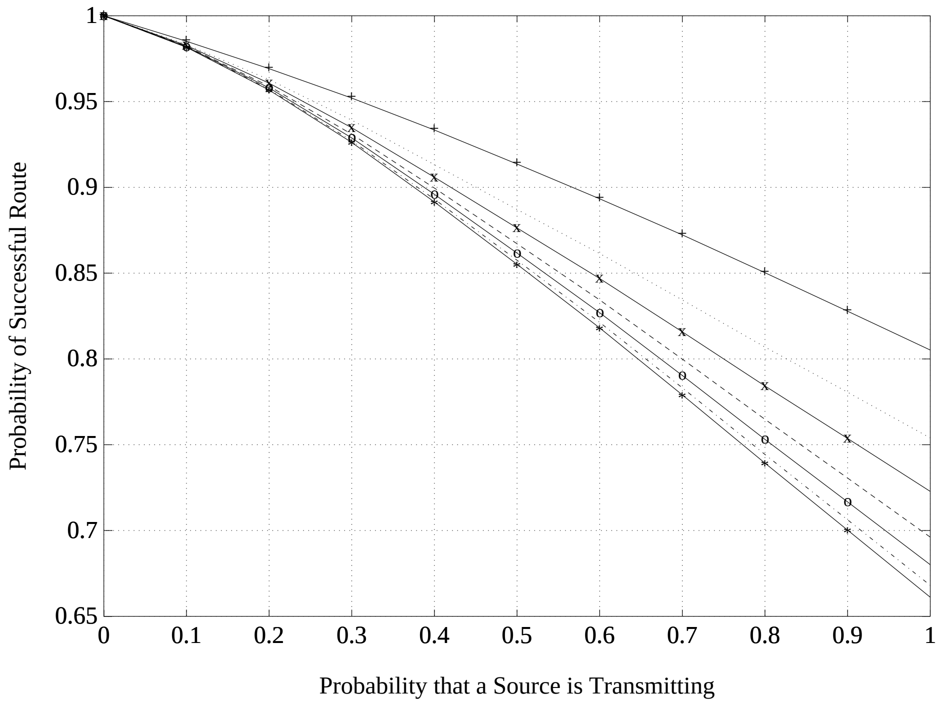

Figure 4.15: Routing Statistics for Hybrid Fat-Tree (49152 processors)

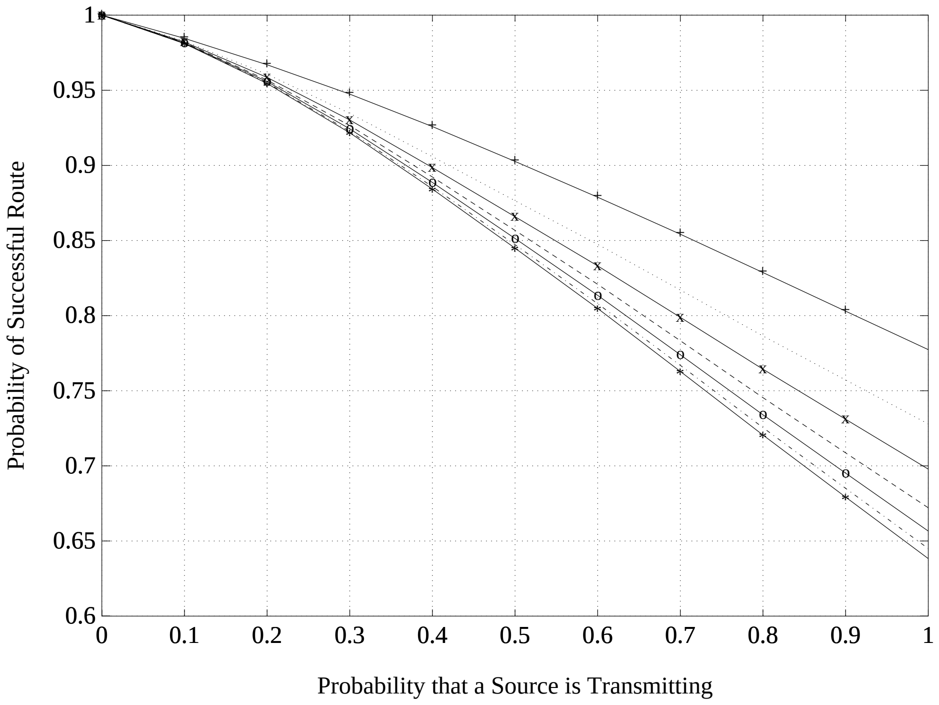

Figure 4.16: Routing Statistics for Hybrid Fat-Tree (196608 processors)

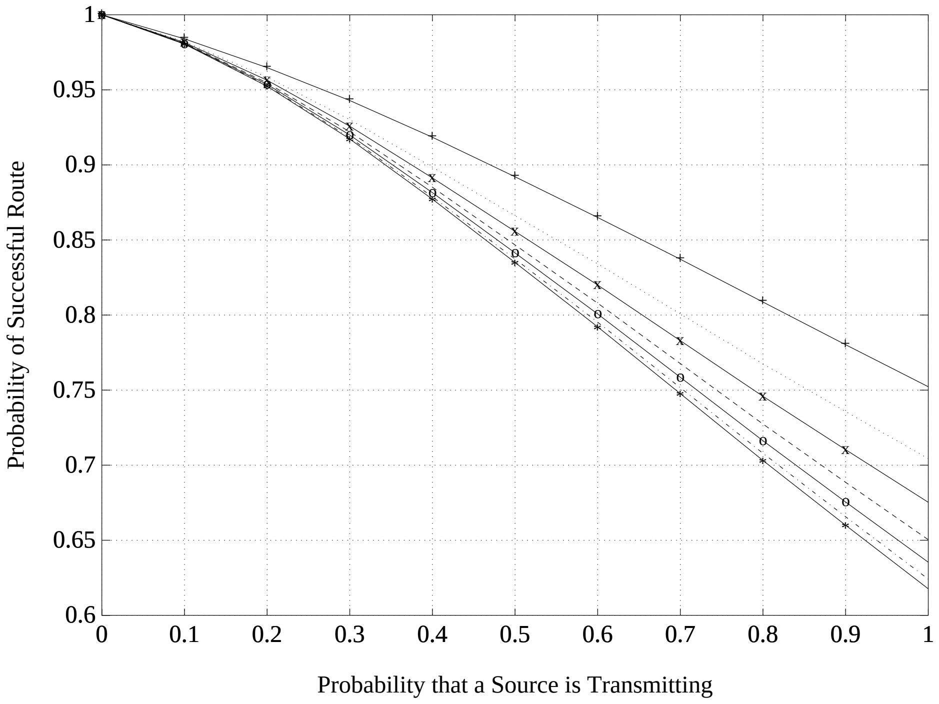

Figure 4.17: Routing Statistics for Hybrid Fat-Tree (786432 processors)

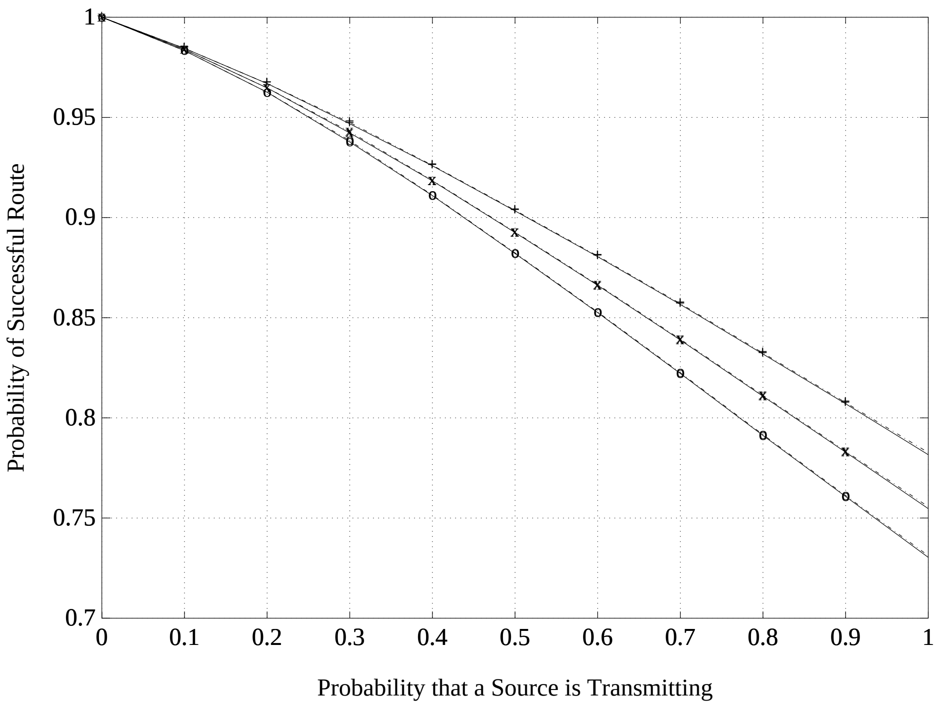

Figure 4.18: Normalized Routing Statistics for Hybrid Fat-Tree

essentially the extreme case in which no locality is exploited. As traffic deviates from this flat distribution, the performance will deteriorate. Figure 4.19 shows the routing statistics for the bidelta networks described in Section 4.4.3. The topmost curve in the figure corresponds to a 3 stage, 64 processor, bidelta network. Each successive curve down the graph plots statistics for a network with an additional stage of routing, making the total network size a factor of four large. The bottommost curve thus corresponds to a 10 stage network supporting 1,048,576 processors.

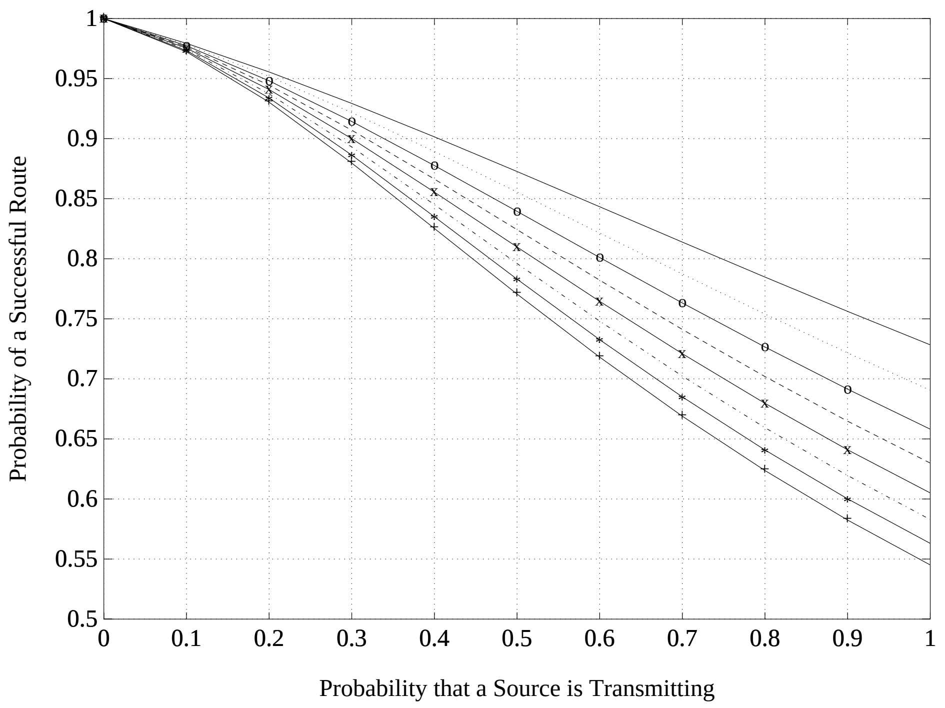

Figure 4.19: Routing Statistics for Full Bidelta Networks of Various Sizes

5. Conclusion

Chapters 2 and 3 presented the topology and construction details for constructing fat-tree style networks using Transit technology. Chapter 4 then summarized many of the parameters and characteristics implied by the network structure in order to provide a basis for comparison with other architectures. This chapter summarizes some of the requirements and consequents of these fat-tree structures and identifies components of the design which might merit further study.

5.1 Routing Component Requirements

The manner in which the RN1 routing component could be utilized as the primitive routing element in network construction was always a primary consideration throughout this work.

In order to build a maximally fault tolerant network structure, two properties of the RN1 component were identified as critical. The ability to separately configure the byte dropping characteristics of each input port was established as necessary for optimal dispersion of connections (Section 3.1.2 and 3.3.5). The alternate configuration of RN1 as an independent pair of 4×4 crossbars with one output in each logical direction proved critical to the construction of a network resilient against single component failures. This also helps minimize the effects of each component failure (Section 2.1.2 and 3.3.4).

For this network structure the only functionality that is lacking from the current RN1 design is the ability to deal with round trip delays on long wires (Section 3.6). This ability turns out to be necessary for optimal performance when building any networks of these sizes regardless of whether the network is arranged as a bidelta or fat-tree network (Section 3.6 and 4.2). Section 3.6 describes a scheme for rectifying this deficiency in future revisions o RN1.

5.2 Characteristics

The network design presented overcomes the potential problems identified in Section 1.1 due to network size, decomposition, and topology. A number of interesting structures were developed to surmount these problems. The result is a fat-tree network structure that has a number of desirable properties.

5.2.1 Constructable

An overriding concern in the development of the physical network structure was establishing a design which could be physically realized in the real world. Attention was paid to the real world constraints posed by technology limitations. Also, some care was taken to minimize the construction complexity. The intent was to design a structure which could realistically be fabricated within the next few years without relying on any technological breakthroughs. At the same time, attention was paid to technology trends so that the architecture itself would still prove valuable with improved technological capabilities.

The unit tree structures (Section 3.3) provide a powerful component for dealing with technological size limitations. They provide a decomposition of the large network into components that can be reliably fabricated. The unit tree also serves as a building block, limiting design complexity. Since each unit tree is virtually identical, the complexity o configuring a network is reduced to that of configuring a couple of unit trees and then inter-connecting unit trees accordingly.

The hollowcube geometry (Section 3.4.3) provides a simple and straightforward manner in which to arrange unit trees. Its structure reflects the natural growth characteristics o the fat-trees constructed with unit trees. With careful design (Section 3.5), the hollowcube structures can provide a framework for the interconnection of unit trees; in this manner, they will significantly simplify the complexity of configuring and maintaining unit tree interconnections.

5.2.2 Fault Tolerance

Utilizing the properties of RN1 and judicious wiring patterns, the fat-tree network will be reasonably fault tolerant. All the fat-tree structures described are resilient against si component failures (Sections 2.1.2 and 3.3.4). With the redundant paths through the network, the effects of any component failures are minimized. Sections 2.1.1 and 2.3 presented constraints on wiring to maximize the fault tolerance of these networks. Additionally, the accessibility afforded by the hollowcube structure will allow faults to be repaired while the network is in operation (Section 3.5.4).

5.2.3 Cheap Routing

The network is structured to make routing on the fat-tree both conceptually and practically simple. This allows routing to be performed cheaply as described in Section 3.8.

5.2.4 Performance

The resulting fat-tree based networks attempt to minimize the latency of the interconnect while maximizing the probability of performing a successful route.

Latency is improved over a naive approach in a number of ways. Routing on the upward tree path (Section 2.2.4) reduces by a factor of three the number of stages of routing incurred while traveling up the tree. Utilizing the RN1 component, which is a constant sized switch, keeps the routing delay at each stage constant at the cost of incomplete concentration (Section 2.2.4). Making the routing clock cycle insensitive to the signal delays incurrec while crossing long wires, allows fast pipelining of data transmissions; a long wire only affect the latency of a connection which actually traverses it (Section 3.6).

When building networks of the magnitude described here, it is clear that locality will be necessary to obtain reasonable performance. The fat-trees constructed from unit trees have a natural architectural locality resulting from the allocation of hardware. This natural locality is summarized for full fat-trees in Section 4.5.1 and for hybrid fat-trees Section 4.5.2. Knowing this optimal level of locality may prove useful in designing parallel algorithms and software for large systems. With proper locality, the routing statistics fo these fat-tree structures is reasonable; the probability of obtaining a successful route i fully loaded network is in the 70% to 80% range even for networks with three-quarters of a million processors (Sections 4.5.1 and 4.5.2).

5.3 Future

I have attempted to develop the construction of these fat-tree structures in reasonable detail. Sufficient detail was provided to demonstrate the feasibility of such networks. Also, this development gives enough information about the structure that good estimates of critical parameters such as hardware requirements and performance can be obtained. This development, however, is by no means definitive. A number of issues are open for further study and optimization while a few issues require additional specification.

5.3.1 Routing Statistic Modeling

The routing statistics provided in Section 4.5 model the probability of successfully finding a route through the network as a function of network loading. As such, it does not take into account the manner in which the network will normally used; in general, when a message fails to get routed, the processor will resend it later. This will have a feedback effect on the network loading. In order to see network performance under this more realistic model of network traffic, a more detailed statistical model must be utilized which takes input queuing into account. Additionally, this analysis would provide a means for estimating the amount of time required, on average, in order to acquire a complete connection through the network.

5.3.2 Simulations

As a complement to statistical modeling, it will be enlightening to simulate this network structure. This will provide a good means of checking the validity of the statistical assumptions under various loading conditions.

5.3.3 Interconnection Details

Section 2.3 provided a number of constraints necessary to obtain good performance. As mentioned there, it is unclear whether or not the expansion properties proposed by Leighton and Maggs should be used to further dictate the details of interconnection wiring. Once we have functional simulations, we should be able to establish the importance of these constraints. Once this is known, detailed wiring patterns must be developed for the unit tree structures.

The wiring between stack stages merits additional attention and detail. If optical interconnection is to actually be used, additional study on the integration of this technology wil be required. Additional work on schemes for adaptive alignment will be a virtual necessity in order to utilize free-space interconnection on this scale. For maximal efficiency, custom components may need to be designed and fabricated for this purpose.

5.3.4 Geometry

As stated in Section 3.4.6, the hollow cube geometry is not known to be optimal. Its possible future study may produce a geometry that provides shorter interconnection distances for the basic network structure described while retaining the properties of maintainabilit and constructibility.

5.3.5 Packet Switching

The basic Transit networking scheme is circuit switched. Circuit switching is used to minimize the latency inherent in getting a response from a remote processor on the network. Using circuit switching, no buffering is needed within the network. This avoids problems due to internal buffer overflow or network congestion by blocked packets.

For large networks where the latency from one end of the network to the other is greater than the time required to transmit the standard quanta of data, circuit switching may inefficiently utilize network bandwidth. If such is the case, it might be worthwhile to consider packet routing schemes. The basic fat-tree structure and interconnect described here would be applicable to a packet switched network scheme. The difference that arise would occur in the routing protocol and policies. Almost all of these differences would thus be limited to the design of the cache-controller and routing component.

5.3.6 Construct Prototypes

Certainly, the most definitive way to evaluate the worth of this fat-tree network structure is to actually construct prototypes. The construction exercise will guarantee that no essential construction details are overlooked. Such construction will, no doubt, uncover issues and problems not yet considered.

Bibliography

[Mnsky 90] Mnsky, Henry Q., An Enhanced Crossbar Routing Chip for a Shared Memory Mltiprocessor, S.M Thesis, MT, *forthcoming.*

[DeHon 90] Dehon, André M, Knight, T. F., Mnsky, Henry Q., Fault Tolerance Dsign for Mltistage Routing Networks, MT A.I. Lab Memo 1225, April 1990.

[Knight 90] Knight, T. F. and Sobalvarro, P. G., Routing Statistics for Unqueued Banyan Networks, MT A.I. Lab Memo 1103, *forthcoming.*

[Leighton 89-2] Leighton, F. T. and Maggs, B.M, *Personal Communications*, October 1989.

[Leighton 89-1] Leighton, F. T. and Maggs, B. M, Expanders Mght Be Practical: Fast Algorithms for Routing Around Faults in Mltibutterflies, 30th Annual Symposiumon the Foundations of Computer Science, October 1989.

[Leiserson 89] Leiserson, Charles E., *Personal Communications*, October 1989.

[Kiamilev 89] Kiamilev, F. E., Esener, S. C., Paturi, R., Fainman, Y., Mercier, P., Guest, C. C., Lee, S. H., Programmable Optoelectrontic Mltiprocessors and their comparison with Symbolic Substitution for Digital Optical Computing, Optical Engineering 28(4), April 1989, pp. 396-408.

[Knight 89] Knight, T. F., Technologies for Low Latency Interconnection Switches, ACM Symposiumon Parallel Algorithms and Architectures, June 1989, pp. 351-358.

[Cray 89] Cray, S., What's All This About GalliumArsenide?, Distinguished Lecture Series, Volume II: Insustry Leaders In Computer Science and Electrical Engineering, November 1988.

[Rettberg 87] Rettberg, R., and Glasser, L., Digital Phase Adjustment, U.S. Patent No. 4,700,347, October 1987.

[Wu 87] Wu, W H., Bergman, L. A., Johnston, A. R., Guest, C. C., Esener, S. C., Yu, P. K., Feldman, M R., and Lee, S. H., Implementation of Optical Interconnections for VLSI, IEEE Transactions on Electron Devices 34(3), March 1987, pp. 706-714.

[Kruskal 86] Kruskal, Clyde P. and Snir, Marc, A Unified Theory of Interconnection Network Structure, Theoretical Computer Science, 1986.

[Cormen 86] Cormen, Thomas H., Leiserson, Charles E., A Hyperconcentrator Switch for Routing Bit-Serial Messages, Proceedings of the 15th Annual International Conference on Parallel Processing, August 1986, pp. 721-728.

[Bergman 86] Bergman, L. A., Johnston, A. R., Nixon, R., Esener, S. C., Guest, C. C., Yu, P. K., Drabik, T. J., Feldman, M. R., and Lee, S. H., Holographic Optical Interconnects for VLSI, Optical Engineering 25(10), October 1986, pp. 1009-1118.

[Smolley 85] Smolley, R., Button Board, A New Technology Interconnect for 2 and 3 Dimensional Packaging, International Society for Hybrid Microelectronics Conference, November 1985.

[Greenberg 85] Greenberg, Ronald I. and Leiserson, Charles E., Randomized Routing on Fat-Trees, IEEE 26th Annual Symposium on the Foundations of Computer Science, November 1985.

[Leiserson 85] Leiserson, Charles E., Fat Trees: Universal Networks for Hardware Efficient Supercomputing, IEEE tr. on Computers, Vol. C-34 No. 10, October 1985, pp. 892-901.

[Glasser 85] Glasser, L. A., A UV Write-Enabled PROM, Proceedings, 1985 Chapel Hill Conference on VLSI, May 1985.

[Ajtai 83] Ajtai, M, Komlós, and Szemerédi, E., Sorting in $c \log n$ Parallel Steps, Combinatorica, Vol. 3, No. 1, 1983, pp. 1-19.