Building a Better Butterfly: The Multiplexed Metabutterfly

Frederic T. Chong ftchong@lcs.mit.edu
Eric A. Brewer brewer@cs.berkeley.edu
F. Thomson Leighton ftl@math.mit.edu
Thomas F. Knight, Jr. tk@ai.mit.edu

Massachusetts Institute of Technology
University of California at Berkeley

Abstract

Multistage networks are important in a wide variety of applications. Expander-based networks, such as multibutterflies, are a tremendous improvement over traditional butterflies in both fault and congestion tolerance. However, multibutterflies cost at least twice as much in chips and wiring as butterflies. It is also impossible to build large multibutterflies due to their wiring complexity.

We show that we can build an expander-based network that has comparable cost to a butterfly with the same number of endpoints, yet has substantially better fault and congestion performance. Specifically, we introduce a hierarchical construction that dramatically reduces wiring complexity and makes large expanders buildable. We are able to exploit the hierarchical structure to find large numbers of logical wires to multiplex over a smaller number of physical wires. Since many of the wires in an expander-based network are used to provide alternate paths, not useful bandwidth, substantial multiplexing can be done without significantly degrading performance. We present simulation results to support our conclusions. In comparing a butterfly with the comparable 2-to-1 multiplexed metabutterfly, we found that the metabutterfly performed better by nearly a factor of two on random traffic and greater than a factor of five on worst-case traffic.

1 Introduction

Butterflies and related regular, multistage interconnection networks have been broadly used in many applications, but exhibit extremely poor performance under faults, congestion, and worst-case permutations. Recent work has shown that randomly wired expander-based networks, such as multibutterflies, are a substantial improvement. They possess tremendous fault tolerance and very low congestion even under high loads. Unfortunately, the wiring complexity of such networks becomes physically problematic as they become large. Further, multibutterflies use at least twice as many wires and chips as butterflies for the same number of endpoints.

This work addresses the cost and complexity of expander-based networks. We present a sequence of multistage networks, shown in Figure 1, that transforms the traditional butterfly network into a new

---

A short version of this paper will appear in the *International Symposium on Parallel Architectures, Algorithms, and Networks*, Kanazawa, Japan. December 14-16, 1994. An earlier version of this paper also appeared in the Parallel Computer Routing and Communication Workshop, May 16-18, 1994, University of Washington, Seattle, Washington.
Acknowledgments: This work was supported in part by National Science Foundation grant CCR-8716884; by ARPA contracts N00014-91-J-1698 and N00014-92-J-1799; by Air Force Contract AFOSR F49620-92-J0125; by NSF Experimental Systems grant MIP-9012773; by a NSF Presidential Young Investigator Award; by an Office of Naval Research Graduate Fellowship; by an equipment grant from Digital Equipment Corporation; and by grants from AT&T, LSI Logic and IBM.

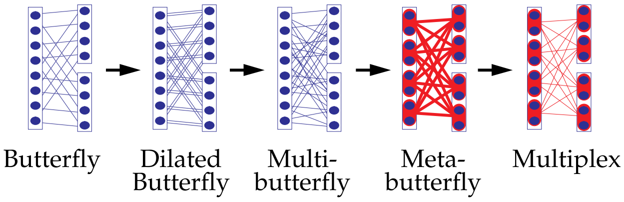


Figure 1: Our sequence of transformations from the butterfly to the multiplexed metabutterfly

network, called a *multiplexed metabutterfly*, that requires similar hardware but provides substantially better performance and fault tolerance. The following sections cover this sequence in detail, but it is useful to first walk through it at a high level:

Butterfly: Simple and easy to build, but provides no fault tolerance and suffers from internal congestion.

Dilated Butterfly: Doubles each wire of the butterfly to achieve some fault and congestion tolerance, but the extra hardware could be used much more effectively.

Multibutterfly: Same number of wires as the dilated butterfly, but provides tremendous fault and congestion tolerance through the use of random wiring. It has strong theoretical underpinnings for its performance and fault tolerance. Although the advantages are probably worth the extra hardware for large sytems, the random wiring makes large multibutterflies impractical because of the wiring complexity. This represents the state of multistage networks prior to our work.

Metabutterfly: Uses the same amount of hardware as the multibutterfly, but constrains the randomness of the wiring to ensure that large versions are practical. Despite the limited randomness, we proved that all the theoretical properties hold [BCL94]. In this paper we present evidence that the fault and congestion tolerance is identical in practice as well. The metabutterfly thus provides all of the benefits of the multibutterfly without the wiring complexity, thus allowing very large networks.

Multiplexed Metabutterfly: Although the metabutterfly provides superior performance and far superior fault tolerance over the traditional butterfly, it also uses twice as many wires. However, these wires really only provide alternate paths rather than additional bandwidth. Thus, we can use multiplexing to reduce the number of wires substantially without proportionally reducing performance. It is important to note that the multiplexing option was enabled by the limited randomness, which allows wires to be grouped together into cables. These cables can then be multiplexed onto thinner cables or onto optical fiber. The goal of the multiplexed metabutterfly is the performance and fault tolerance of the metabutterfly with hardware requirements close to those of the traditional butterfly. We show through simulation that this goal is achievable.

2 Butterflies

In this context, it is useful to view the traditional butterfly as the simplest form of a splitter network, as shown in Figure 2(A). A *splitter network* is composed of multiple stages of routers. It is helpful to view routing a message through a splitter network as a sorting function through equivalence classes of fewer and fewer routers. Specifically, for the i^{th} stage there are r^i equivalence classes, each with r^{s-i} routers, where r is the *radix* of the routers (the number of directions among which the router selects) and s is the number of stages in the network. Each equivalence class is connected to r

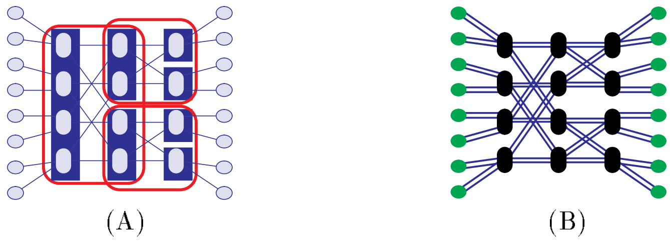


Figure 2: (A) A butterfly viewed as a splitter network. The butterfly is radix 2 and three splitters are shown outlined in round rectangles. (B) A dilated butterfly with radix 2 and multiplicity 2.

equivalence classes in the next stage. An individual splitter consists of an equivalence class and its r associated equivalence classes in the next stage.

The dilated butterfly, shown in Figure 2(B), is also a splitter network. The only difference from the traditional butterfly is that each wire has been split into two, so that one can fail or be busy without affecting the other. Looking at it as a splitter network, each splitter now has two wires to the equivalence classes in each of the r directions.

3 Multibutterflies

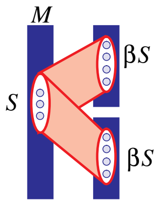
A multibutterfly also has two wires in each direction, but they are not constrained by the butterfly wiring pattern. The only constraint is the equivalence classes: each of the r pairs of wires must go to a distinct equivalence class, but they can go to any node in their class. For the multibutterfly, these nodes are selected randomly from their equivalence class. The key reason for random selection is that it ensures that the resulting network is an *expander graph* (with high probability), which leads to very strong theoretical properties that are also valuable in practice.

A bipartite graph with M inputs and N outputs is an (α, β, M, N)-*expander* if every set of $S \leq \alpha M$ inputs reaches at least βS outputs. For a radix-r splitter network to have expansion, each splitter must achieve expansion in each of the r directions. Figure 3(A) shows an M-input, radix-2 splitter with expansion. To achieve expansion, a splitter network must have routers with redundant connections in each of its r directions. We refer to this redundancy, d, as the *multiplicity*; so far we have implicitly assumed $d = 2$ as it is the most practical, but higher multiplicity leads to better fault tolerance at a higher hardware cost. Given multiplicity d, the degree of any node is dr.

A multibutterfly is a splitter network in which each splitter is an $(\alpha, \beta, M, \frac{M}{r})$-expander in each of the r directions, where M is the number of inputs of the splitter.

Bassalygo and Pinsker [BP74] first studied splitter networks with expansion. Recently, numerous results have been discovered that indicate that multibutterflies are ideally suited for message-routing applications. Among other things, multibutterflies can solve any one-to-one packet routing [Upf89], circuit-switching [ALM90], or non-blocking routing problem [ALM90] in optimal time, even if many of the routers in the network are faulty [LM89]. No other networks are known to be as powerful.

The reason behind the power of multibutterflies is that expansion roughly implies that βk outputs must be blocked or faulty for k inputs to be blocked, and thus it takes β^j faults to block one input j levels back. In contrast, *one* fault in a radix-2 butterfly blocks 2^j inputs j levels back. As a consequence, problems with faults and congestion that destroy the performance of traditional networks can be overcome easily in multibutterflies. (For a survey of the research on multibutterflies see [Pip93] [LM92].)



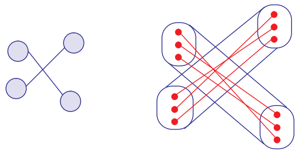


Figure 3: (A) A splitter with expansion. (B) A random 3-extension (right) of a graph (left).

3.1 Wiring Complexity

Multibutterflies are generally constructed by randomly wiring redundant connections between equivalence classes of each splitter. Although deterministic constructions are known [WZ93], none are known to produce expansion comparable to random wiring.

Unfortunately, random wiring (as well as the known constructions of good expanders) scales poorly in practice. For example, a 4K-endpoint machine with multiplicity $d = 2$ has 8K wires in the first stage, almost all of which would be long cables with distinct logical endpoints. For comparison, a fat-tree [Lei85] might have a similar number of cables for the root node, but there are few logical endpoints, so huge groups of wires can be routed together. The groups connect to many boards, but the boards are located together and the connection of cables to boards is arbitrary and thus low labor. In the multibutterfly, the cables cannot be grouped and the connection of cables to boards is constrained. The other early stages also suffer from this problem.

At first glance, it appears that this wiring complexity is inherent to both expanders and random wiring. Indeed, given a splitter with M boards of input routers, M boards of output routers, and k routers per board, we can expect each board to be connected to about $\Theta(\min(M, dkr))$ other boards when using random wiring. For typical values of M, d, k, and r, this means that we would need to connect nearly *every* input board to nearly *every* output board in a randomly wired splitter. Clearly, this becomes infeasible as M gets large and thus the randomly wired multibutterfly does not scale well in the practical setting where the network consists of boards of chips. A similar problem arises at the level of cabinets of boards for very large machines.

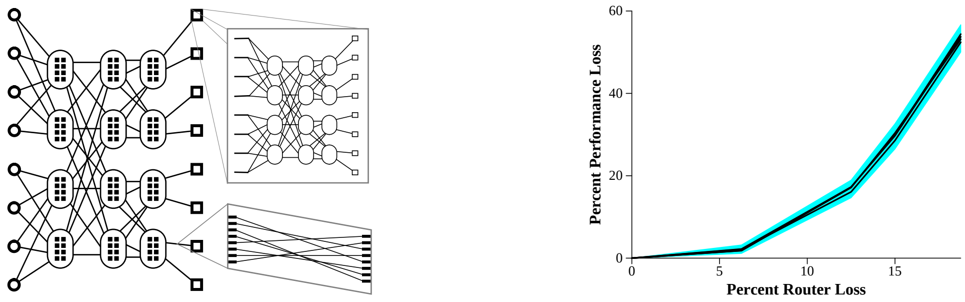
In what follows, we show how to (randomly) construct a special kind of expander for which there is no explosion in cabling cost. In particular, we show how to build a multibutterfly for which each board is connected to only dr other boards, no matter how large M and k become, thereby achieving full scalability. In effect, there are a few fat cables connected to each board instead of many thin cables. At the same time, the resulting network will still have all the same nice routing properties as a randomly wired multibutterfly. Hence, we will gain wiring scalability without sacrificing performance.

4 Random k-extensions

The wiring complexity of large expanders can be dramatically decreased by constructing them hierarchically. A *hierarchical expander* is an expander constructed from the application of a sequence of *random k-extensions* to an expander.

Given a directed graph $G \equiv (\mathcal{V}, \mathcal{E})$, an integer $k \geq 1$, and a set of permutations of $[1,k]$, $\Pi = \{\pi_e | e \in \mathcal{E}\}$, we define the *$k$-extension of G induced by* Π to be the graph $G' \equiv (\mathcal{V}', \mathcal{E}')$ where:

$$\mathcal{V}' \equiv \{\langle v, i\rangle \mid v \in \mathcal{V},\ i \in [1,k]\} \text{ and } \mathcal{E}' \equiv \{(\langle u, i\rangle, \langle v, j\rangle) \mid (u,v) \in \mathcal{E} \text{ and } \pi_{(u,v)}(i) = j\}$$



(A) (B)

Figure 4: (A) A radix-2, multiplicity-2, 64-endpoint metabutterfly with metanodes of size 8. Each circle on the left contains 8 inputs, and each oval metanode contains 8 routers. Each router (solid square) is a 4×4 switch. Each output metanode (hollow square), shown expanded at the top right, is an 8-input multibutterfly; each channel, shown expanded at the bottom right, contains 8 wires. Typically, the metanodes correspond to boards.
(B) The four curves represent performance for a 1024-endpoint multibutterfly and 1024-endpoint metabutterflies with metanodes of size 8, 16, and 64. The shaded area represents the aggregate 95 percent confidence interval of the curves.

We define a *random k-extension of a graph G* to be a k-extension induced by some Π such that each $\pi_e \in \Pi$ is an independently and uniformly chosen random permutation of $[1,k]$. Equivalently, a random k-extension of a graph G can be obtained by selecting randomly and uniformly over all of the $(k!)^{|\mathcal{E}|}$ possible k-extensions of G. Figure 3(B) shows an example of a random 3-extension.

5 Metabutterflies

We apply random k-extensions to a multibutterfly to build a scalable, hierarchical network, the *metabutterfly*. Each splitter of the metabutterfly is a random k-extension of the corresponding splitter of the multibutterfly, with the possible exception of the last few stages. The value of k may differ for each splitter.

For example, a 1024-node metabutterfly can be implemented with a 64-extended 16-node multibutterfly plus 16 64-node multibutterflies for the output metanodes. The total counts of nodes, wires, and stages are each the same as for a 1024-node multibutterfly; the only difference is the wiring pattern. Figure 4(A) shows a radix-2 64-input metabutterfly that is an 8-extended 8-input multibutterfly. This example has a two-level hierarchy, but deeper hierarchies are possible and actually make sense for very large networks.

5.1 Metabutterfly Properties

Brewer, Chong, and Leighton [BCL94] have proven that any (α, β, M, N)-expander with $\alpha M \geq 1$ can be scaled with a random k-extension into an $(\alpha', \beta', kM, kN)$-expander with probability at least $1 - 2e^{-\alpha M}$, where $\alpha' = \frac{\alpha^2}{\beta^2 e^4 + 4\alpha}$ and $\beta' = \beta - 2$. This means that we have a method of scaling expanders with very little change in β, and some degradation in α.

Although the theoretical bounds on α degrade with each level of hierarchy, we now demonstrate empirically that the performance and fault tolerance of metabutterflies is identical to that of multibutterflies. We use the methodology of previous studies [CED92] [CK92] and investigate performance with uniformly distributed router failures within each network.

We compared performance for a 1024-endpoint multibutterfly and for 1024-endpoint metabutterflies with metanode sizes of 8, 16, and 64. The routers had a radix of 4 and a multiplicity of 2. We used a synthetic load based upon shared-memory applications. Figure 4(B) shows that the performance of all four networks is statistically indistinguishable.

6 Multiplexing

Now that we have established the attractive theoretical and empirical properties of our construction, we would like to use it to reduce the wiring and chip cost of expander-based networks. We will use multiplexing techniques to reduce the number of wires in the metabutterfly to be comparable to the number in a butterfly with the same number of endpoints. However, we will still have d times more cables. This complexity is unavoidable and is the cost of providing the multiplicity that gives our networks their performance and fault tolerance.

From wire utilization studies, we discovered that many of the wires in an expander-based network are only used to provide alternate paths, not raw bandwidth. To reduce wiring cost, we would like to provide alternate paths with logical wires that are multiplexed over a smaller number of physical wires. Recall that we have replaced the many thin cables of the multibutterfly with a small, constant number of fat cables in the metabutterfly.

With large cables, the option of multiplexing becomes both more cost effective and more likely to deliver good performance. For example, optical fiber would provide enough bandwidth to replace very large cables. Note that a packet routing scheme across such fiber would provide little degradation in performance until the load reached bandwidth limits. The number and capacity of fibers can be designed to accommodate expected load rather than peak load. The larger the original cable being multiplexed, the more likely the average load will be significantly lower than the peak load. We can exploit such differences to build more cost-effective networks.

Figure 5(A) shows the performance of a radix-4 ($r = 4$), dilation-2 ($d = 2$), 5-stage, 1024-endpoint metabutterfly with metanodes of size 8 ($k = 8$) under various multiplexing conditions and using the same benchmark as in Section 5.1. Even though the benchmark loads the network heavily, a large portion of the curve is flat. In other words, many of the physical wires can be eliminated from the metabutterfly without significant performance degradation. For comparison, Figure 5(B) also shows the performance of a butterfly on this benchmark, which performs almost two times worse than the comparable metabutterfly.

The butterfly is plotted directly over the multiplexed metabutterfly with approximately the same amount of wiring (2-to-1 multiplexing). Note that our multiplexing techniques do not apply in the end stages, since metanodes are of size one. These last two stages have twice as much hardware as the butterfly, but this hardware is necessary for fault and congestion tolerance. Fortunately, these end stages have high physical locality and can be fabricated without expensive wiring, such as on PC boards. We are able to multiplex where it matters the most, in the beginning stages where the wires are long and expensive. Further, the effect of the end stages on network cost diminishes as we build larger networks.

Figure 5(B) shows the performance of the same multibutterfly (metanodes of size 8) under multiplexing and the same benchmark except that destinations are chosen by bit-inverse rather than randomly. Regular networks such as the butterfly perform very poorly on such permutations, but randomized networks perform much better. Once again, we see little performance degradation up to a high degree of multiplexing. For comparison, we can see that the butterfly performs poorly on this permutation, over five times worse than the comparable metabutterfly.

Note that the bit-inverse benchmark degrades more severely with multiplexing than the random benchmark. This is primarily due to less balanced loading of the outputs of the routers in the first stage. In the random case, every output has very close to the average number of messages to

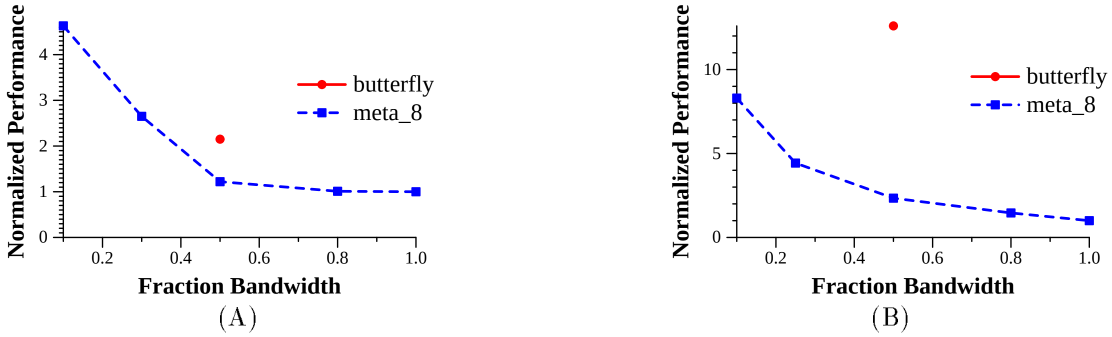


Figure 5: Multiplexed performance with random destinations (A) and bit-inverse destinations (B). The performance of a full butterfly plotted above the comparable 2-to-1 multiplexed metabutterfly.

route. In the bit-inverse case, however, some outputs may have zero messages, while others have nearly twice the average. Because our simulations are barrier-synchronized, our overall performance is limited by the slowest of these first-stage outputs.

This load imbalance can be no worse than a factor of rd, as compared to the $\mathcal{O}(\sqrt{n})$ factor which can arise in the butterfly. In practice, the full metabutterfly has the bandwidth to accommodate the imbalance. Performance degradation due to the imbalance only occurs as we multiplex the channels. Note that degradation tends not to occur under random router failures for two reasons. First, input partitioning may break up the bit-inverse pattern. Second, the use of the Leighton-Maggs algorithm for partitioning results in high bandwidth channels.

The load imbalance arises because, in bit-inverse, all the messages in any given input metanode are destined for only one of r equivalence classes of destinations (for a 1024-endpoint, radix-4 network, the r equivalence classes are: 0-255, 256-511, 512-767, and 768-1023). In other words, all the messages from a given input metanode must be routed to one of d outputs in a single direction. We can divide our input metanodes into r equivalence classes corresponding to which destination class their messages are going. If any first-stage metanode is wired to more than its share of input metanodes from each of the r equivalence classes, an imbalance may occur. We can fix this problem for bit-inverse by guaranteeing that each first-stage metanode reaches an equal number of input metanodes from each equivalence class. The bad cases would still exist, but would be randomized to permutations less likely to occur in real applications.

In summary, the 2-to-1 multiplexed metabutterfly outperforms the comparable butterfly, but has a factor of d higher cabling complexity. Recall, however, this higher complexity also results in extremely high fault tolerance. The butterfly has no fault tolerance.

6.1 Randomized Multiplexing

Observe that, since the logical wires within each metachannel cable were assigned via a random permutation, we really do not care to which logical wire a packet belongs. We can perform *randomized multiplexing*, where the message packets across a cable are demultiplexed to random destinations. This results in increased efficiency in terms of bandwidth used on the physical wire or optical fiber. In traditional multiplexing, packets must either be assigned to fixed time slots or have extra headers identifying their destination upon demultiplexing. The use of fixed time slots is inefficient because there may be empty slots. Every destination must have a slot, but not every destination receives a packet at all times. The use of headers is also inefficient because the headers take up bandwidth. With randomized multiplexing, only the actual packets are sent and demultiplexing proceeds by assigning any packet to any free destination.

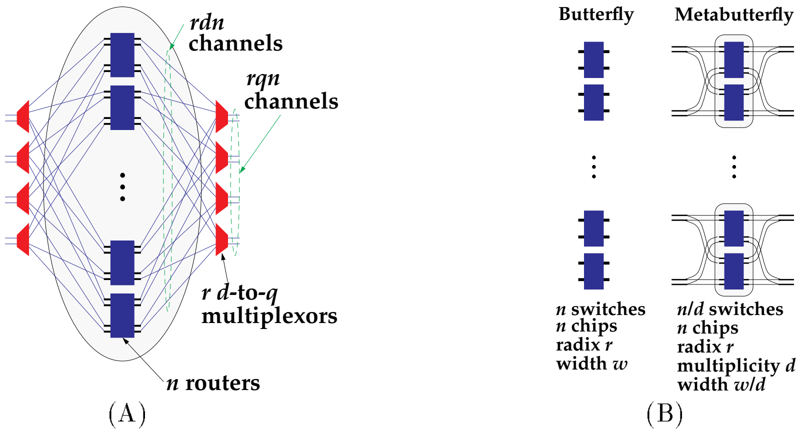


Figure 6: (A) Multiplexors at metachannel endpoints. (B) Implementing a metanode (right) with a cascaded routers and hardware equivalent to the butterfly (left).

An important result of randomized multiplexing is that only one type of metabutterfly cable is needed. The randomization is provided by the multiplexing hardware, not the physical cables. Having only one standard type of cable greatly simplifies manufacturing and assembly.

This notion of time-varied multiplexing is similar to the time domain switching used in STS switches in the telephone community. However, the random nature of our logical wire clusters provides novel opportunities for efficient implementation. In fact, randomized multiplexing results in networks with a stronger, more dynamic form of randomness than networks with physical connections. This dynamic randomness may provide a fairer share of resources in the face of faults or congestion, especially in the early stages of the network where expansion has less effect. Over time, dynamic randomness allows every router on the input side of the cable to reach every router on the output side, effectively providing each input-side router with enormous multiplicity over time. We have yet to quantify the benefits of randomized multiplexing.

7 Multiplexed Implementations

We have now established the tools and motivation to construct expander-based networks comparable in cost to butterflies. In this section, we describe three implementations. All three optimize for average bandwidth as suggested in Section 6. The three differ, however, in cost, fault-tolerance, and latency. The first implementation reduces wiring costs through multiplexing. The second replaces each metanode with a single high-multiplicity switch. The third replaces each metanode with a number of high-multiplicity switches.

First, the obvious implementation, shown in Figure 6(A) uses a multiplexing chip or chips at the endpoints of every metachannel. Pin limitations would limit the number of routers and channel wires multiplexed by any one chip. This implementation could use either conventional or randomized multiplexing. Latency would depend upon whether an optical or electrical technology was used. The critical nature of the multiplexing chips reduces fault tolerance, as does the reduced number of physical channels. However, this implementation is at least as fault tolerant as the small multibutterflies used to wire the metastructure.

The multiplexed metanode in Figure 6(A) can be viewed as a single, large radix-r, multiplicity-qn switch. Our original structure switches rdn channels internally, but only rqn channels are visible externally. Such a structure reduces internal collisions, but directly switching the external inputs and outputs with high-multiplicity switches will have even fewer internal collisions. Our next two implementations focus on such direct switching. Note that multiplexing technology will still be required between the network endpoints and the metanodes, as well as between the end stages of the network and the metanodes.

Our second implementation replaces each metanode with a single time-domain ATM switch such as the AN2 [AOST93]. The high-bandwidth, high-switching capabilities of such an ATM switch make them ideal for metanode implementation. Although ATM switches could support randomized multiplexing, such ATMs are actually powerful enough to implement a radix-r, multiplicity-dq switch, where q is the number of physical channels per directions, $q \leq n$, and the metanode to be implemented had radix r, multiplicity d, and size n. Once the functionality of a metanode can be fit in a single switch, a high-multiplicity switch makes more sense than a randomly multiplexed switch. Fault-tolerance is once again comparable to that of the small multibutterflies formed by the metanodes. Latency, however, is high in optical technologies.

The ATM switch represents the extreme case in which a metanode is implemented by a single high-multiplicity switch. In our third implementation, we implement metanodes with several high-multiplicity switches, each simulated by a cascade of routing chips. The goal is to use the same number of chips, each with the same number of pins, as the corresponding portion of the conventional butterfly. A metanode in our first implementation would have n chips each with rdw input and rdw output pins, where the width, w, is the number of bits in a channel. An equivalent portion of a butterfly would have n chips with only rw inputs and rw outputs. Therefore, we have nrw input pins and nrw output pins in total to implement our metanode.

In order have a multiplicity of d, we must reduce the number of routers from n to $\frac{n}{d}$. This gives us dw pins per physical channel. However, each chip can only have rw inputs and rw outputs. Therefore, we must implement each of our radix-r, multiplicity-d, width-w routers with d cascaded chips, each with radix r, multiplicity d, and width $\frac{w}{d}$. Figure 6(B) compares a cascaded metanode to the equivalent butterfly hardware. The resulting metanode is a radix-r, multiplicity-n router. Since the original metanode had dn channels in each direction, our cascade implementation effectively performs d-to-1 multiplexing.

The cascading of router chips is described in [DeH93]. Basically, a set of d cascaded width-$\frac{w}{d}$ chips work in tandem to simulate a single width-w router. Since pad delays dominate high-performance switches, latency of these cascaded routers should be comparable to those of the original butterfly. Fault tolerance is much better than the butterfly, but can be substantially worse than our other two implementations. For a given chip failure probability P_f, a cascaded metanode remains operable with probability $(1 - P_f)^d$. We still have the fault tolerance of the underlying metanode small multibutterfly, but we have increased our metanode failure probability by a power of d. This is a problem with cascaded routers, but should be manageable at small values of d. Alternatively, we could try to use error-correcting codes, but we would need to use an extra $\mathcal{O}(\log n)$ *chips* in each n-chip cascade to tolerate a single chip failure.

In summary, we have described methods to build multiplexed metabutterflies that are comparable in cost to butterflies. Our direct switching implementations are simpler and may result in fewer collisions in the heavily loaded case. However, our first implementation, involving multiplexed metanodes, can provide higher bandwidth in the lightly loaded case by providing all of a metanode's bandwidth to a small number of active logical channels.

8 Conclusion

We have presented a sequence of transformations from the traditional butterfly to the dilated butterfly, multibutterfly, metabutterfly, and finally to the multiplexed metabutterfly. Our hierarchical wiring techniques transformed the multibutterfly from a non-scalable network to a the metabutterfly, a highly-scalable network. Our multiplexing techniques resulted in the multiplexed metabutterfly, which can be comparable in cost to a butterfly with the same number of endpoints, but provides nearly twice the performance for random traffic and over five times the performance for worst-case traffic.

Although the multiplexed metabutterfly is less fault tolerant than the metabutterfly, it is far more fault tolerant than the comparable butterfly. Most importantly, the multiplexed multibutterfly provides alternate paths with logical rather than physical wires, achieving high performance without excess hardware cost. Our simulations demonstrate this, revealing little performance degradation even for 2-to-1 multiplexed cables.

We also introduced the notion of randomized multiplexing, which allows us to build metabutterflies with standard cables. Randomized multiplexing also provides extremely high effective multiplicity, which should result in higher performance and is a direction of future study.

References

[ALM90] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a non-blocking network. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 149–158, May 1990.

[AOST93] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems*, 11(4):319–352, November 1993.

[BCL94] Eric A. Brewer, Frederic T. Chong, and F. Thomson Leighton. Scalable expanders: exploiting hierarchical random wiring. In *Proceedings of the 26nd Annual ACM Symposium on Theory of Computing*, pages 144–152, May 1994.

[BP74] L. A. Bassalygo and M. S. Pinsker. Complexity of optimum nonblocking switching networks without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.

[CED92] Frederic Chong, Eran Egozy, and André DeHon. Fault tolerance and performance of multipath multistage interconnection networks. In Thomas F. Knight Jr. and John Savage, editors, *Advanced Research in VLSI and Parallel Systems 1992*, pages 227–242. MIT Press, March 1992.

[CK92] Frederic T. Chong and Thomas F. Knight, Jr. Design and performance of multipath MIN architectures. In *Symposium on Parallel Architectures and Algorithms*, pages 286–295, San Diego, California, June 1992. ACM.

[DeH93] André DeHon. Robust, high-speed network design for large-scale multiprocessing. Master's thesis, MIT, 545 Technology Sq., Cambridge, MA 02139, February 1993.

[Lei85] Charles E. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[LM89] Tom Leighton and Bruce Maggs. Expanders might be practical: Fast algorithms for routing around faults on multibutterflies. In *IEEE 30th Annual Symposium on Foundations of Computer Science*, 1989.

[LM92] Tom Leighton and Bruce Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Transactions on Computers*, 41(5):1–10, May 1992.

[Pip93] Nicholas Pippenger. Self-routing superconcentrators. In *25th Annual ACM Symposium on the Theory of Computing*, pages 355–361. ACM, May 1993.

[Upf89] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *21st Annual ACM Symposium on Theory of Computing*, pages 241–250. ACM, May 1989.

[WZ93] Avi Wigderson and David Zuckerman. Expanders that beat the eigenvalue bound: explicit construction and applications. In *25th Annual ACM Symposium on the Theory of Computing*, pages 245–251. ACM, May 1993.