We have been considering the problem of designing a *non-blocking* network, i.e. a network in which we can simultaneously connect every input i to an output $\pi(i)$ where π can be any permutation. We considered two candidates for this: the Butterfly and the Benes. On the Butterfly we saw that for some permutations π it was indeed possible to make all N connections, where N is the number of inputs, but for some others it was possible to connect only $\sqrt{N}$ of the inputs to the required outputs. On the Benes, we found that the connections could be made for all π.

An important issue in designing such networks is the algorithm for setting up the connections. To solve the problem on the Benes network, we needed to solve a colouring problem. While the algorithm for this is simple, it is not *distributed* – essentially we need to have one computer know all of π and calculate the paths, and send that configuration to the network. It is preferred, if instead, the input nodes (which have a limited amount of processing in them) can decide only how to establish connections locally, based on some local querying, i.e. each node coordinates with its neighbours to determine the path for the message it holds. Unfortunately, it doesnt seem that such a distributed algorithm is possible for the Benes network.

The Multibutterfly network[4] solves this problem and others. A variant of this which we will call the *reduced multibutterfly*, will be seen to have the non-blocking property, i.e. disjoint paths can be established for any permutation π. Further, the paths can be found using a distributed algorithm that runs in time $O(\log^2 N)$. There exists a more complex $O(\log N)$ time algorithm as well, but we will not discuss this.

Multibutterflies also possess some fault-tolerance properties, and a few other interesting properties which we will not study.

1 Multibutterflies

The overall structure of a multibutterfly is quite similar to a butterfly. An (α, β, N, d) multibutterfly B has N inputs and N outputs, and is made up of 3 networks:

1. An (α, β, N, d) splitter network X on $N = 2^k$ inputs and two sets of

outputs, respectively called the up outputs and down outputs, each of size $N/2$. The precise wiring between the inputs and the outputs is discussed later.

2. Two $(\alpha, \beta, N/2, d)$ multibutterfly networks B_u, B_d.

The inputs of X form the inputs of B. The up and down outputs respectively form the inputs of B_u, B_d. The outputs of B_u, B_d form the outputs of B. The parameters α, β will explained later. The parameter β is also known as the expansion of the splitter.

Like a butterfly, a multibutterfly has $(n+1)2^n$ nodes, arranged in $n+1$ levels, and the recursive structure is also similar. The key difference between the butterfly and the multibutterfly is the splitter – it provides a much richer connection than what is available in the buttefly.

1.1 Concentrators and Splitters

A splitter is made out of two concentrators.

A (α, β, N, d) concentrator has two sets of vertices, U, V, with $|U| = N$ and $|V| = N/2$. The max degree of any node in U is d, and that in V is $2d$. Further every subset of U having $k \leq \alpha N$ nodes is required to be connected to at least βk nodes in V.

Since we can choose $k = \alpha N$, we require $\beta\alpha N \leq |V| = N/2$, i.e. α, β may only be chosen such that $\alpha\beta \leq 1/2$. Typically we will be interested in $\beta > 1$, and α, d to be constants independent of N. Whether this choice of parameters is feasible is discussed later.

A (α, β, N, d) splitter has three sets of vertices, U, V_u, V_d, with $|U| = N$ and $|V_u| = |V_d| = N/2$. The subgraph on vertex sets U and V_u as well as the subrgaph on vertex sets U, V_d are each required to form an (α, β, N, d) concentrator. The set U is called input, the set V_u as the upper outputs, and the set V_d as the down/lower outputs, and the corresponding concentrators as the upper and lower concentrators.

2 Constructing a concentrator

We defined a concentrator graph, but that does not mean that it exists for interesting values of the parameters!

In fact explicitly constructing such graphs (i.e. writing down their adjacency matrix) turns out to be a very hard problem, requiring a fair amount of sophisticated mathematics. Proving existence is easier – and the proof is non-constructive.

The proof is based on the so called "probabilistic method". In this we give a procedure for constructing a graph, and show that there is non-zero probability that the constructed graph will have all properties required of a concentrator. The probability is typically small, and this procedure is thus not too useful for the actual construction; it nevertheless shows that a graph with the required properties exists! Otherwise how could we get non-zero probability?

Theorem 1 *For all $\alpha > 0$, $\beta > 1$ s.t. $\alpha\beta < 1/2$, there exists an (α, β, N, d) concentrator, for*

$$d > \beta + 1 + \frac{\beta + 1 + \ln 4\beta}{\ln \frac{1}{2\alpha\beta}}$$

Proof: The construction is as follows: (1) Start with 2 sets, U, and V containing respectively $N, N/2$ vertices. (2) Expand each vertex in U and V into d and $2d$ vertices respectively. Call the resulting sets of Nd vertices U', V' respectively. (3) Pick a random perfect matching on U' and V'. (4) Collapse each d-tuple of vertices in U' and $2d$ tuple in V' back to the vertex from which they were created. At this point we will have a multigraph in which each vertex has degree $d, 2d$ in U, V respectively. (5) Convert this to a graph by replacing multiple edges (if any) connecting the same pair of vertices with a single edge. This finishes the construction.

If the graph is not a concentrator, then there is some $S \subseteq U$ of size $k \leq \alpha N$ and $T \subseteq V$ of size βk, such that all the neighbors of vertices in S are strictly contained in T. We first consider the probability that this happens for fixed sets S and T. Then we sum over all choices of S, T to get the probability that the resulting graph is not a concentrator with the given parameters.

We need the probability that the neighbours of S are a proper subset of T, we will instead consider the overestimate: the probability that the neighbours of S are contained in T. Let S' and T' denote the sets resulting from S and T respectively in step 2 above. The number of ways in which the kd vertices in S' can be matched with the Nd vertices in V' is

$$|V'|\,(|V'|-1)\ldots(|V'|-|S'|+1) = Nd(Nd-1)\ldots(Nd-kd+1)$$

The number of ways in which the kd vertices of S' can be matched with the $2\beta kd$ vertices in T' is

$$|T'|\,(|T'|-1)\ldots(|T'|-|S'|+1) = 2\beta kd(2\beta kd-1)(2\beta kd-kd+1)$$

Thus the probability that S' has all its neighbors in T' is

$$\frac{2\beta kd\cdot 2\beta kd-1\cdot 2\beta kd-2\ldots\cdot 2\beta kd-kd+1}{Nd\cdot Nd-1\cdot Nd-2\ldots\cdot Nd-kd+1}\leq\left(\frac{2\beta kd}{Nd}\right)^{kd}=\left(\frac{2\beta k}{N}\right)^{kd}$$

This upper bounds the probability that neighbours of S are properly contained in T. Now, observe that there are $\binom{N}{k}$ ways of choosing S, and $\binom{N/2}{\beta k}$ ways of choosing T for every $k\leq\alpha N$ Thus, the probability that the graph is not an expander is at most:

$$\sum_{k=1}^{\alpha N}\binom{N}{k}\binom{N/2}{\beta k}\left(\frac{2\beta k}{N}\right)^{kd}\leq\sum_{k=1}^{\alpha N}\left\{\left(\frac{k}{N}\right)^{d-\beta-1}e^{\beta+1}(2\beta)^{d-\beta}\right\}^{k}\leq\sum_{K=1}^{kN}\left\{\alpha^{d-\beta-1}e^{\beta+1}(2\beta)^{d-\beta}\right\}^{k}$$

Using $\binom{n}{r}\leq\left(\frac{ne}{r}\right)^{r}$, and noting that $\frac{k}{N}\leq\alpha$. The series above is geometric, and will be smaller than 1 if we choose

$$\alpha^{d-\beta-1}\cdot e^{\beta+1}\cdot(2\beta)^{d-\beta}\leq 1/2$$

Taking natural log and simplifying we get:

$$d>\beta+1+\frac{\beta+1+\ln 4\beta}{\ln\frac{1}{2\alpha\beta}}$$

For this choice of d, the probability that the graph is not a concentrator is stictly less than 1, i.e. there is non zero probability of finding a (α,β,N,d) concentrator, i.e. such concentrators exist. ■

Note that by choosing larger values of d we can boost up the probability and make it extremely likely that the above construction gives concentrators (Exercises). Note, however, that in any case no efficient procedure is known for verifying that the graph we generated is an expander!

Instead of randomized constructions such as the one above, one might attempt a more direct, deterministic construction. Such constructions have been found quite hard, and the bounds obtained are weaker[3].

For now, the best course is very likely to use the randomized construction and use simulations to see how well the graph does on standard packet routing problems. This has been found to be quite promising[2].

2.1 Establishing paths on Multibutterflies

Path selection on a multibutterfly is similar to a Butterfly, in level 0 we determine whether the packet is destined for outputs belonging to the upper submultibutterfly or lower, and continue the process recursively. This, as in the Butterfly, can be done by considering the bits in the destination address. The crucial difference from the Butterfly is that we are no longer restricted to a unique path from any input to output. Each node in level 0 has d edges going to the top multibutterfly and d edges going to the bottom multibutterfly. Thus at each level each path can be extended forward in d ways.

3 Non-blocking properties

We will show that given any permutation π, it is possible even in the worst case, to establish paths from input i to output $\pi(i)$ for about $2\alpha N$ inputs, in an (α, β, N, d) expander having $\beta \geq 1$. This is much better than the Buttefly in which only $\sqrt{N}$ paths are possible in the worst case. This worse than the Benes network, in which we could establish all N paths. However, by modifying the multibutterfly network slightly, we can build a network that establishes all paths. Furthermore, for suitable choices of the parameters, we can get a fast distributed algorithm to establish the paths, which does not appear to be possible for a Benes network.

Theorem 2 *Suppose π is a permutation over $\{0, \ldots, N-1\}$. Let $L \leq 1/2\alpha$ be as large a power of 2 as possible. It is possible to establish vertex disjoint paths from input i to output $\pi(i)$ of an (α, β, N, d) multibutterfly for all i such that $\pi(i) = a (\mathrm{mod} L)$ for any integer a, provided $\beta \geq 1$.*

Proof: Without loss of generality, let $a = 0$. Consider only the requests to destinations $0 (\mathrm{mod} L)$. So there are only N/L requests. Suppose we have been able to build paths for these requests till some level i of the multibutterfly. Let us consider how the paths can be extened to level $i+1$.

Consider any splitter S with inputs in level i. It has $m = N/2^i$ inputs, and $m/2$ outputs connected to the top (sub)multibutterfly M_t, and $m/2$ to the bottom (sub)multibutterfly M_b. Let I_t denote the set of inputs of S which have received paths that need to be connected to M_t. We know that $m/2L$ of the $m/2$ outputs of M_t (whose numbers are multiples of L) will

receive requests. These must come through splitter S, and hence we have $|I_t| = m/2L \leq m\alpha$. We will show that the paths of all the requests in I_t can be extended to level $i+1$.

Let $I' \subseteq I_t$, and let $\Gamma(I')$ denote the neighbours of I' in M_t. Then $|I'| \leq |I_t| = m/2L \leq m\alpha$. Thus the expansion property applies to every I', and we have $|\Gamma(I')| \geq \beta|I'| \geq |I'|$. Thus Hall's condition is satisfied for set I_t, and hence we can match every input in I_t to a distinct input of M_t. In a similar manner we can also extend the paths of the requests destined for M_b. ■

Next, it may be observed that we can prune the multibutterfly into a network with just $N' = N/L$ inputs and outputs, say by considering only inputs $\{0, \ldots, N'-1\}$ and outputs $\{i | i = 0 \pmod{L}\}$. Of course, we can throw out all nodes of the multibutterfly which do not have a path from the above mentioned inputs, or a path to one of the above mentioned outputs. Notice that this network, which we will call a *reduced multibutterfly* will have $O(N' \log N')$ vertices and edges, N' inputs and outputs and in which any permutation can be established from the inputs to the outputs. The reduced multibutterfly is thus similar to the Benes in hardware requirements and capability.

Do note, however, that finding the paths is not simple – at each level we need to run a matching algorithm. We will overcome this problem next.

3.1 Finding the matching quickly

The matching can be found quickly if $\beta > d/2$. In this case we have the so called *unshared neighbours property.*

Definition 1 *Any (α, β, M, d) splitter has a δ unshared neighbour property if in every subset I of inputs where $|I| \leq M\alpha$, there are $\delta|I|$ nodes in I that have an up neighbour that is not adjacent to any other node in I, and similarly for down neighbours.*

Lemma 1 *Any (α, β, M, d) splitter with $\beta > d/2$ has a $2\beta/d - 1$ unshared neighbour property.*

Proof: Consider any set I of inputs with $|I| \leq M\alpha$. These have at least $\beta|I|$ up neighbours. Of these neighbours, let n_1 be incident to exactly one

input in I, and n_2 to 2 or more inputs. Since from I there are $d|I|$ edges incident on the neighbours we have $n_1 + 2n_2 \leq d|I|$. But there are at least $\beta|I|$ neighbours, thus $n_1 + n_2 \geq \beta|I|$. Solving for n_1 we get $n_1 \geq (2\beta - d)|I|$. But these neighbours must connect to at least $(2\beta/d - 1)|I|$ nodes from I. ■

So now we have the following algorithm to compute the matching:

1. Each input having a request sends a token to all its up neighbours if the request needs to move up, or to all its down neighbours if the request needs to move down.

2. Each splitter output receiving just 1 token replies with an "accept" response to the input from which it received the token.

3. An input that receives an "accept" response sends its request to any of the nodes from which it received the response.

4. Repeat from step 1 until all tokens have moved.

The key is that in each iteration of the above algorithm, at least a fraction δ of the inputs holding requests must receive an "accept" response. Thus in time $\log_{1/(1-\delta)} M\alpha = O(\log N)$ iterations all requests in a splitter move forward. (Question: can you prove that all requests must move forward? Is it possible that other requests take all positions that a certain request could have moved to?)

So the overall algorithm runs in phases; in each phase (of precalculated duration above) the one more edge is added to each path. The total time is thus $O(\log^2 N)$.

Exercises

1. Show that an ordinary Butterfly is a $(1, 0.5, 2^n, 2)$ multibutterfly. Show that it is not really possible to improve the estimates of the first two parameters.

2. Consider the following bipartite graph having $2N$ left vertices and N right vertices. Vertex i on the left is connected to vertices $i \bmod N$ and $i + 3 \bmod N$, with N a power of 2. Does this have $\beta > 1$ when considered a concentrator for any constant α?

3. Consider the decision problem of deciding whether a given bipartite graph is an expander. Show that this is in co-NP.

4. Estimate the values for which the construction will give an expander with probability at least 0.99.

5. A (α, β, N, d) concentrator has two sets of vertices, U, V, with $|U| = N$ and $|V| = N/2$. For this exercise suppose that the degree of every node in U is exactly d, and that in V is exactly $2d$. Further every subset of U having $k \leq \alpha N$ nodes is connected to at least βk nodes in V. Suppose we are also given that α, β are constants such that $\beta > 1, \alpha\beta > 1/4$.

   Show that the diameter of this graph is $O(\log N)$.

References

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

[2] F. T. Leighton and B. M. Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Transactions on Computers*, 41(5):578–587, May 1992.

[3] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

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