Lecture 2: Network Computing

The following lecture on network computing will focus mainly on describing and analyzing butterfly networks and their related Beneš networks.

1 Ideal Networks

In some sense, an"ideal" data parallel computer may be depicted as in Figure 1.

Ideal Data Parallel Computer

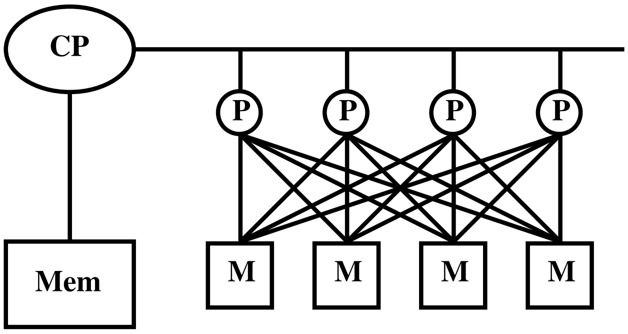

Figure 1: A serial central processor (CP) with its own memory (Mem) is connected to an array of processors (P). Each processor is connected to all of the memory units (M).

Each processor is connected to each memory unit. This fully connected network can be simulated by a "crossbar" interconnection network, as in Figure 2, although there are some slight differences due to arbitration.

But the problem with building a fully connected network is in the constraints on the number of pins on the units. The degree of a node in a fully connected network grows linearly with the number of nodes, n. Typically, however, we are constrained to a constant number of pins, or at best $\lg n$ pins. This constraint has driven a lot of interest in networks with low degree nodes.

2 Butterfly Networks

One of the most popular low degree networks is the "butterfly network" also known as the "FFT network."

It is possible to build a butterfly network recursively. Two butterfly networks can be composed into a larger butterfly network by connecting line i of the first network to line

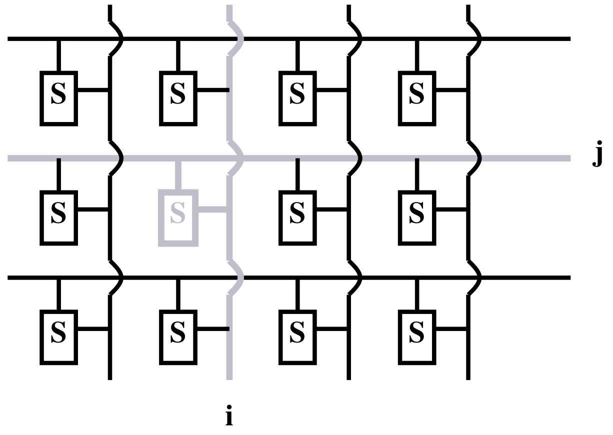

Figure 2: Processors along the bottom are connected to memory units along the right hand side through a crossbar interconnection network of routing switches. The grey path shows the connection created by a switch between processor i and memory unit j.

i of the second network, for all lines in the two networks. Lines, running horizontally, are numbered consecutively from top to bottom, for convenience. This process constructs the next "level" of nodes. It may not be immediately obvious but if you remove the first (2 by 2) level of nodes, the resulting network consists of two interleaved but separate butterfly networks. The recursive structure of the butterfly network makes it a convenient network for designing certain algorithms.

2.1 Properties of an n-input Butterfly Network

Diameter of a network is the longest minimum distance between two nodes.

The diameter of a butterfly network is $\Theta(\lg n)$. You can view a butterfly network as a sort of binary tree. Each step through the network divides the number of accessible nodes in half. In fact, if you trace out the paths for one node on one side of the network to all the nodes on the other side, you will trace out a fully binary tree. This works from either direction through the network.

The number of nodes in the network, $N = \Theta(n \lg n)$ because there are $\lg n$ levels or columns of nodes in the net, and each level consists of n nodes.

Bisection width is the number of wires you would need to cut in order to divide the network in half.

In general we would like large bisection widths in order to prevent bottlenecks in the network. The butterfly network has a $\Theta(n)$ bisection width.

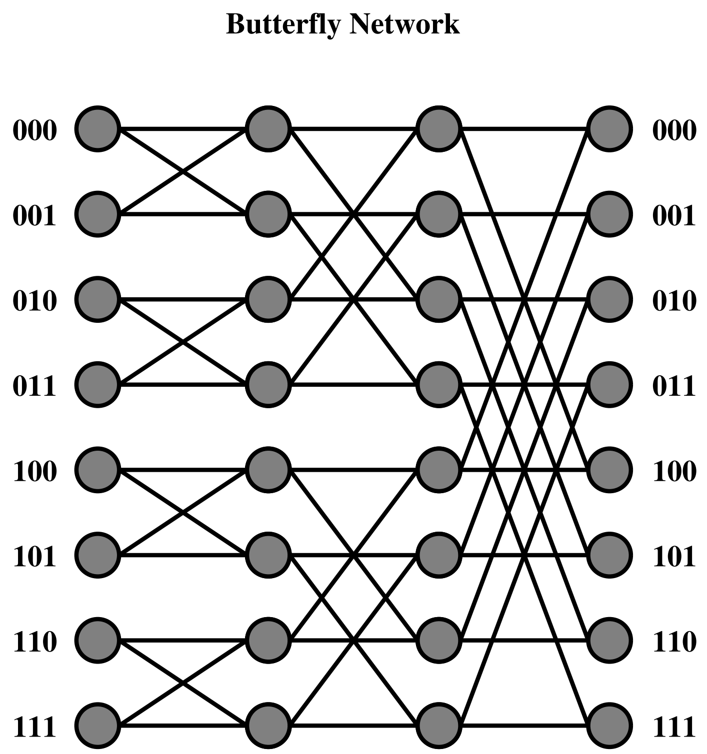

Figure 3: This 8-input butterfly network has 4 levels, or columns. Each row is labeled with a binary address.

3 Routing

Typically, there are two ways to organize the placement of the processor and memory units. The "dance-hall" organization puts all the processors on one side of the network and all the memory units on the opposite side. In contrast, the "distributed memory" organization merges a memory unit with each processor so that each processor has fast access to its local memory. Then the butterfly network must be "wrapped around" to give the processors access to non-local memory units. That is, the nodes on the opposite sides of the ordinary dance-hall butterfly network are merged into a single column of nodes.

One nice aspect of butterfly networks is that it is easy to do the routing through the network. If we number the nodes in binary on either side of an ordinary butterfly network, the number of the destination node determines the route through the network. If we start on the left side of the network (the side with the simple 2 by 2 butterfly networks) then the path to a node on the right is determined by taking the low order bits of the destination node's number, one at a time, and interpreting a 0 as "take the upper path" and a 1 as "take the lower path" at each intermediate node.

Example 1 The route from node 011 on the left to node 101 on the right is "down-up-down" as illustrated in Figure 5.

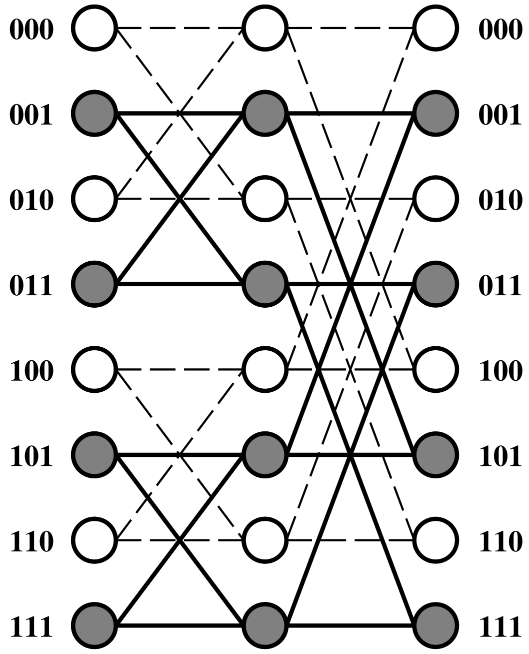

Figure 4: If the first level of an 8-input butterfly network is removed, the result is a pair of interleaved 4-input butterfly networks. The nodes of the two networks have been distinguished as either white or grey.

This algorithm works in the opposite direction as well if the directions are taken from the high order bits first, instead of the low order bits.

Minor cycles are the small, 2 by 2 cycles in the network.

Major cycles are the larger cycles connecting the minor cycles.

3.1 Packet Routing

Consider routing a packet from a node labeled with the bits $x_1x_2 \dots x_d$ to the node $y_1y_2 \dots y_d$. The route proceeds by correcting one bit at a time starting from the major cycles and progressing to the minor cycles as such.

$$
\begin{gathered}
x_1x_2 \dots x_d \\
y_1x_2 \dots x_d \\
y_1y_2 \dots x_d \\
\vdots \\
y_1y_2 \dots y_d
\end{gathered}
$$

Thus it takes d steps to move the packet through the network. This is reasonably fast. In practice, people often build 4-ary or 8-ary butterfly networks to speed them up.

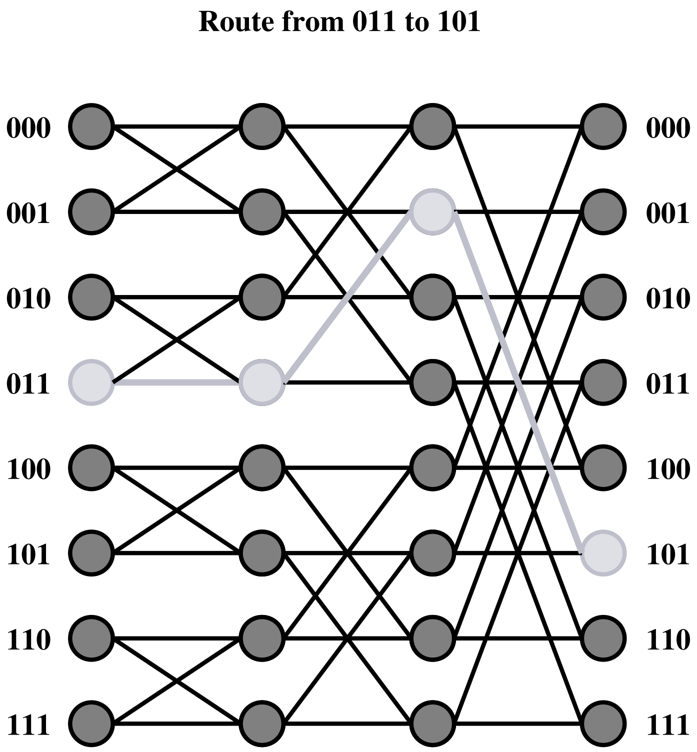

Figure 5: The route from node 011 to node 101 has been highlighted in grey. The choice of wires follows the bits of the destination node. Here, 101 translates as the three choices of lower-upper-lower.

But the number of steps is not the whole story. Congestion in the network is a critical factor influencing the speed of the network. Suppose we have n packets on an n-input butterfly network. If they all have the same destination it will take $\Omega(n)$ time for all the packets to reach their destination. On the other hand, if the mapping from source nodes to destination nodes is a permutation (no two packets have the same destination) then it takes $\Omega(\sqrt{n})$ time in the worst case. This can be seen in the following example.

Example 2 Consider sending packets through an n-input butterfly network. There may be a subset of $\sqrt{n}$ of the packets, where each packet has the form $x_1x_2x_3x_4 0000$ and is routed to the destination $0000x_1x_2x_3x_4$. We are assuming that the different packets have unique combinations of bits $x_1x_2x_3x_4$ and so there is a one-to-one mapping between source and destination nodes for these packets. Halfway through the network, all of these packets will go through the same node on line 00000000 thus producing congestion of at least $\sqrt{n}$ packets.

4 Beneš Networks

Can we design a network to do better than $\Omega(\sqrt{n})$ time in the worst case? Consider routing a packet through a butterfly network and then back again the opposite direction. That is, consider attaching another butterfly network to the back of the first one as in Figure 6. This is a Beneš network.

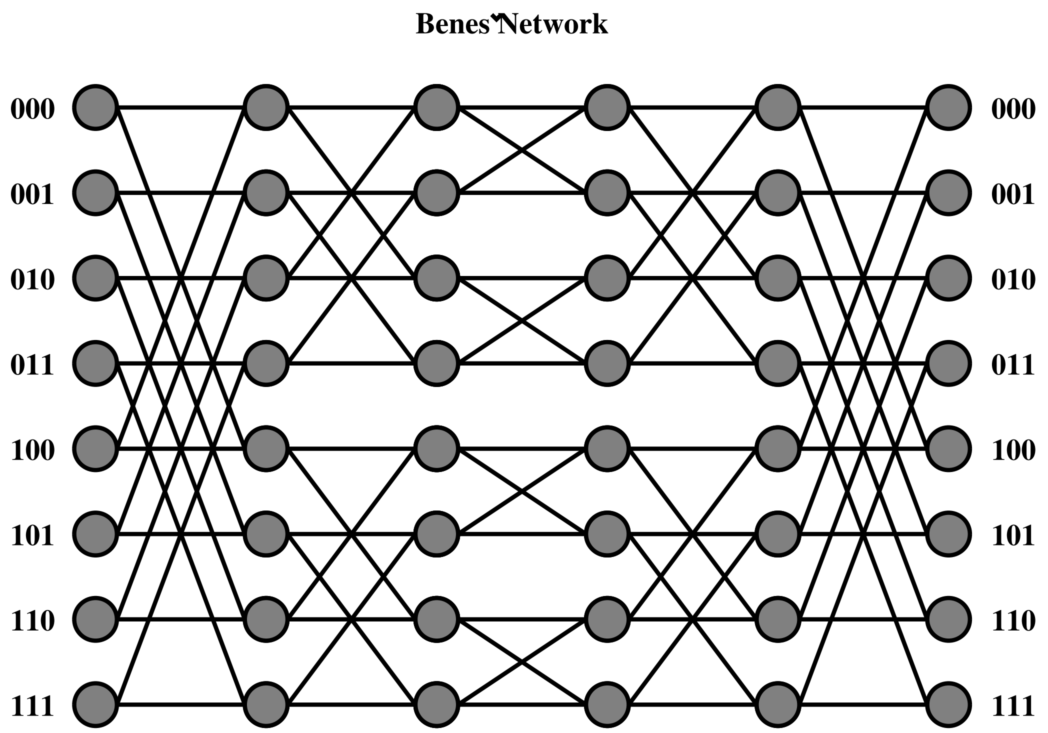

Figure 6: A Beneš network is composed of two butterfly networks. The minor cycles of one of the butterfly networks have been removed because they do not increase the power of the network.

The second layer of 2 by 2 butterfly networks in the middle does not give us any added power. At the level of a 2 by 2 butterfly network, either permutation of input to output nodes can be routed without any congestion.

A Beneš network has the following nice properties:

1. One can determine the router switch setting for a permutation from input to output nodes in $O(\lg n)$ time offline.

2. There is no congestion!

The algorithm for determining the router switch settings is essentially a greedy strategy. We work in recursively from the outside two layers to the inner, smaller Beneš networks. At any one recursive step, each input node has two connections, one to each of two smaller,

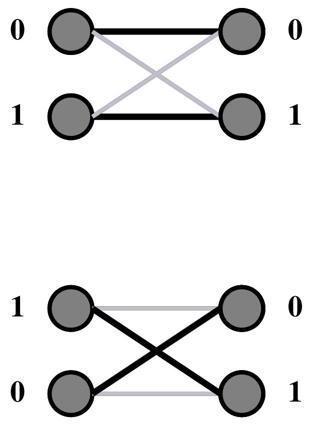

Figure 7: There are only two possible permutations to be routed on a 2 by 2 Beneš network. Neither of them produce a conflict. The routings have been colored black.

independent Beneš networks. We begin the selection of wires for packets by arbitrarily choosing one of the two wires for the packet on one of the input nodes. This forces a series of routing decisions in order to avoid any congestion. At some point, another arbitrary decision may arise. Each time that happens, we will choose the upper wire of the two wires. Eventually, all the packets will be assigned a route that does not conflict with any other packet's route. We then proceed by induction on the smaller two Beneš networks.

Why does this work? Whenever we move from left to right, we go through the upper box (inner Beneš network). But whenever we move from right to left we go through the lower box. Upon returning to the left side, the "mate" of that input node has either not yet been routed, or it was routed on a move from left to right and so went through the upper box.[1] In either case there is no conflict. We then proceed by induction on the smaller Beneš networks until we come to the 2 by 2 base case.

Bounded degree means $O(1)$ degree.

Corollary 1 *An n-input Beneš network can simulate any bounded degree network on n nodes in $O(\lg n)$ time.*

Proof. The proof of this is left as an exercise. The basic idea of the proof is to use the fact (easily proved using a greedy strategy) that any graph with maximum vertex degree δ can be edge-colored using $\delta + 1$ colors.[2] The edge coloring gives a decomposition of the graph into routing problems, each of which is a permutation or sub-permutation.

Once the proper configuration for the switches has been determined, setting the switches of the Beneš network takes $O(\lg n)$ time. In fact, if you are clever, you can pipeline the setting of the switches with the routing of the packets.

[1] This can be seen as a conflict graph with an even number of nodes in each cycle. The problem is then to do a two-coloring of the nodes.

[2] An edge coloring is an assignment of colors to each edge in the graph such that no vertex has two incident edges of the same color.

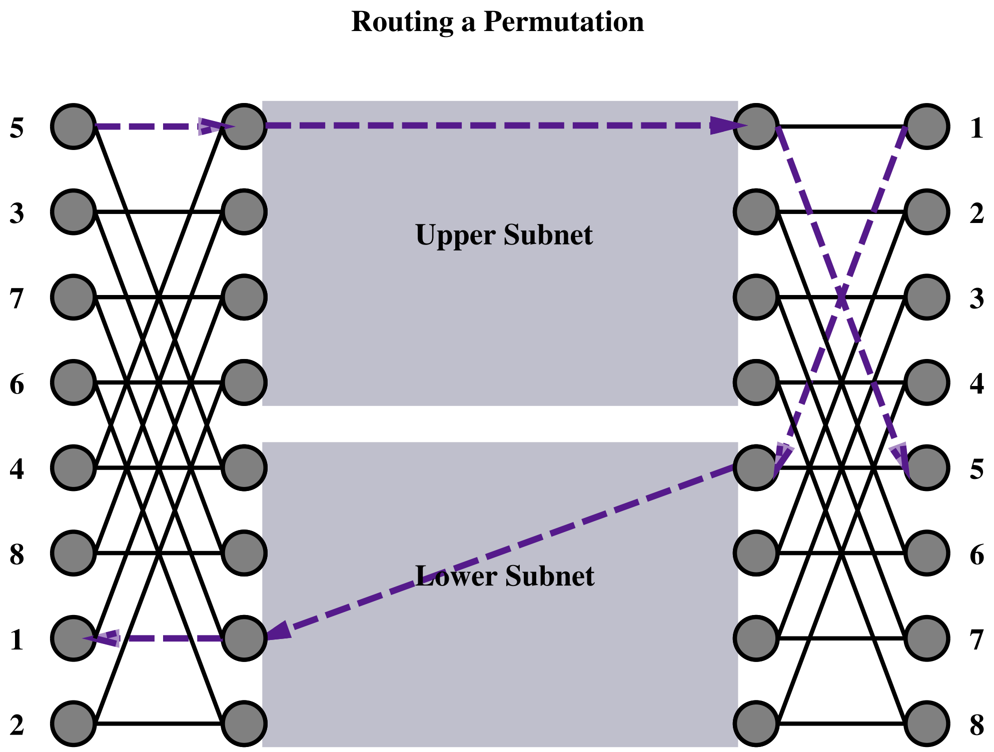

Figure 8: The grey rectangles represent abstractions of 4-input Beneš networks. The dashed lines show the beginnings of routing a permutation. The choice of routing the first step of the packet with the destination of node 5 is arbitrary. But once it has been chosen it determines the routing of the packet with the destination of node 1, the node which is paired to node 5 in the outer layer of this Beneš network. We work backwards from output node to input, routing through the lower sub-network. The next step would be to route the packet destined for node 7 because that choice is forced by the routing of the packet for node 1. This process can be continued until all the packets are routed.

We can simulate a mesh by doing each of the four directions separately. The proof of the corollary requires generalizing this to non-uniform networks.

The Beneš network is a "universal" network. It can simulate an ideal computer. Unfortunately, the algorithm for setting the switches is off-line. We would like it to be done on-line.

5 Congestion Bounds

While the butterfly network performs poorly in the worst case, it turns out to be pretty good in the average case.

Theorem 2 *Consider the N^N N-packet routing problems on an N-node butterfly network. These are the problems where every node has a packet to send. At least $N^N(1 - \frac{1}{N^{\Omega(1)}})$ of these problems are routed in $O(\lg N)$ time by the greedy algorithm.*

Note that there are N^N of these problems because any node may need to send its packet to any other node in the network. Also note that we are speaking of an N is the number of nodes in the whole network while n is the number of input nodes. If we want to determine n, we know that $N = n \lg n$ implies $\lg N = \lg n + \lg\lg n$ which implies that $\lg N = \Theta(\lg n)$ which finally implies that $n = \Theta(\frac{N}{\lg N})$.

The theorem states that only a polynomially small fraction $(1/N)$ of these problems are bad for the butterfly network. It turns out that you can inflate this with any constant k so that only $1/N^k$ problems are bad, at the expense of multiplying the time needed for the "good" problems by a factor of k.

Proof. We will only prove a congestion bound.

$N = n \lg n$. We will first consider an arbitrary node x at level k in the network. We first take $O(\lg N)$ time to push all packets along the horizontal wires until they are queued up on the left side of the network. We want to route them through to the other side of the network in order to move each packet to the line of its destination node. Then we will push all the packets back along the horizontal wires and allow the nodes to pick off the packets as they reach their destinations.

We will first consider a random N-packet routing problem and show that the probability that more than $O(\lg N)$ packets go through any node is $\le \frac{1}{N^{\Omega(1)}}$. We will do this by determining how many packets use node x at level k. Then we will combine the probabilities for the different nodes.

The number of packets that can reach node x is $2^k \lg n$. We know that each input node has $\lg n$ packets queued up because each of the $\lg n$ nodes on that line contributed a single packet when they were all pushed to the left.

The probability that a given packet passes through x is $\le 2^{-k}$. Note that $k \le \lg n$ because there are only $\lg n$ layers in the network.

Consider any set of r specific packets. The probability that they all pass through node x is 2^{-kr}. Consequently, the probability that at least r packets pass through node x is at most

$$\Pr(\text{at least } r \text{ packets pass through } x) \le \binom{2^k \lg n}{r} 2^{-kr} \tag{1}$$

Note that this over counts. If $r + \Delta$ packets pass through x, this event will be counted $\binom{r+\Delta}{r}$ times within the $\binom{2^k \lg n}{r}$ ways of choosing r packets. Thus we have an upper bound. The rest is math.

We will use the simplification $\binom{a}{b} \le \left(\frac{ea}{b}\right)^b$ from chapter 6 of CLR.

$$\binom{2^k \lg n}{r} 2^{-kr} \le \left(\frac{e2^k \lg n}{r}\right)^r 2^{-kr} = \left(\frac{e \lg n}{r}\right)^r \tag{2}$$

So we need to pick $r > e \lg n$ for our probability bound to be less than 1. We choose $r = 2e \lg N$ which is $O(\lg N)$.

$$\left(\frac{e \lg n}{r}\right)^r \le (1/2)^{2e \lg N} = N^{-2e} \le N^{-5.4} \tag{3}$$

Now we want to find an upper bound for any node. We use Boole's inequality which states that the probability of the union of events is less than or equal to the sum of the individual probabilities of each event. Therefore, the probability that any node has $\ge 2e \lg N$ packets routed through it is $\le N(N^{-5.4}) = N^{-4.4}$.

Therefore, at least $N^N(1 - \frac{1}{N^{\Omega(1)}})$ problems create $\le 2e \lg N$ congestion. $\square$

By adjusting our choice of r by a constant k such that $r = (2e + k) \lg N$ we get an upper bound of $N^{-(4.4+k)}$ bad problems.

It turns out that most of the typical permutations that we use, like reversing the nodes or transposing a matrix, are "bad" routing problems for a butterfly network.

The article by Ranade is an example of a delay sequence argument. He shows that you only need $O(1)$ queues at the nodes of an N-node butterfly network in order to emulate a CRCW PRAM in $O(\lg N)$ time.

Valiant has addressed the problem of the difficulties of efficiently implementing the common forms of permutation. Valiant suggests using two butterfly networks back to back. In order to implement a permutation you route the input to a randomly chosen permutation on the first butterfly network and then use the second butterfly network to route the random permutation into the desired output permutation. In this way, both permutations are random and so the butterfly network will almost always perform well.

Finally, Leighton, Maggs and Rao introduced the multibutterfly network. This is essentially a butterfly network with randomized interconnections. The nodes of level i are grouped into 2^i contiguous clusters. So the zeroth level is all one cluster. The nodes of the next level are divided into the upper half and lower half of the nodes, and so on. Then the nodes of each level are randomly connected to a constant k number of nodes in each of the two clusters parallel with the current node's cluster in the next level. It is then possible to prove that you never have congestion for permutations that take more than $O(\lg n)$ time. There is a small chance that the random connections generated a "bad" network. Unfortunately, checking to see if a network is "bad" is an NP-complete problem.