

11 Communication Networks

Modeling communication networks is an important application of digraphs in computer science. In such models, vertices represent computers, processors, and switches; edges will represent wires, fiber, or other transmission lines through which data flows. For some communication networks, like the internet, the corresponding graph is enormous and largely chaotic. Highly structured networks, by contrast, find application in telephone switching systems and the communication hardware inside parallel computers. In this chapter, we’ll look at some of the nicest and most commonly used structured networks.

11.1 Routing

The kinds of communication networks we consider aim to transmit packets of data between computers, processors, telephones, or other devices. The term *packet* refers to some roughly fixed-size quantity of data—256 bytes or 4096 bytes or whatever.

11.1.1 Complete Binary Tree

Let’s start with a *complete binary tree*. Figure 11.1 is an example with 4 inputs and 4 outputs.

In this diagram, and many that follow, the squares represent *terminals*—sources and destinations for packets of data. The circles represent *switches*, which direct packets through the network. A switch receives packets on incoming edges and relays them forward along the outgoing edges. Thus, you can imagine a data packet hopping through the network from an input terminal, through a sequence of switches joined by directed edges, to an output terminal.

In a tree there is a unique path between every pair of vertices, so there is only one way to route a packet of data from an input terminal to an output. For example, the route of a packet traveling from input 1 to output 3 is shown in bold.

11.1.2 Routing Problems

Communication networks are supposed to get packets from inputs to outputs, with each packet entering the network at its own input switch and arriving at its own output switch. We’re going to consider several different communication network

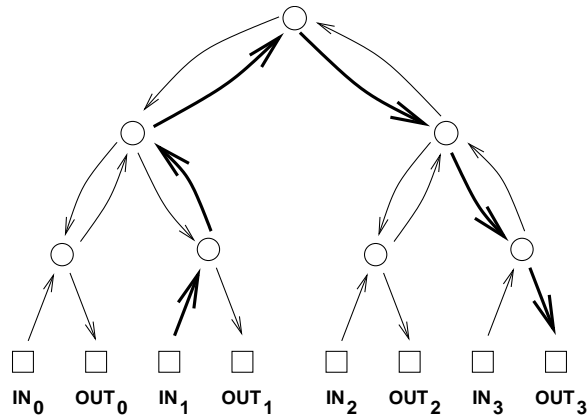


Figure 11.1 Binary Tree net with 4 inputs and outputs

designs, where each network has N inputs and N outputs; for convenience, we’ll assume N is a power of two.

Which input is supposed to go where is specified by a permutation of $[0..N - 1]$. So a permutation π defines a *routing problem*: get a packet that starts at input i to output $\pi(i)$. A *routing* that *solves* a routing problem π is a set P of paths from each input to its specified output. That is, P is a set of paths P_i where P_i goes from input i to output $\pi(i)$ for $i \in [0..N - 1]$.

11.2 Routing Measures

11.2.1 Network Diameter

The delay between the time that a packets arrives at an input and arrives at its designated output is a critical issue in communication networks. Generally, this delay is proportional to the length of the path a packet follows. Assuming it takes one time unit to travel across a wire, the delay of a packet will be the number of wires it crosses going from input to output.

Packets are usually routed from input to output by the shortest path possible. With a shortest-path routing, the worst-case delay is the distance between the input and output that are farthest apart. This is called the *diameter* of the network. In other words, the diameter of a network¹ is the maximum length of any shortest

¹The usual definition of *diameter* for a general *graph* (simple or directed) is the largest distance between *any* two vertices, but in the context of a communication network we’re only interested in the

path between an input and an output. For example, in the complete binary tree above, the distance from input 1 to output 3 is six. No input and output are farther apart than this, so the diameter of this tree is also six.

More broadly, the diameter of a complete binary tree with N inputs and outputs is $2 \log N + 2$. This is quite good, because the logarithm function grows very slowly. We could connect up $2^{10} = 1024$ inputs and outputs using a complete binary tree and the worst input-output delay for any packet would be $2 \log(2^{10}) + 2 = 22$.

Switch Size

One way to reduce the diameter of a network is to use larger switches. For example, in the complete binary tree, most of the switches have three incoming edges and three outgoing edges, which makes them 3×3 switches. If we had 4×4 switches, then we could construct a complete *ternary* tree with an even smaller diameter. In principle, we could even connect up all the inputs and outputs via a single monster $N \times N$ switch.

Of course this isn't very productive. Using an $N \times N$ switch would just conceal the original network design problem inside this abstract switch. Eventually, we'll have to design the internals of the monster switch using simpler components, and then we're right back where we started. So, the challenge in designing a communication network is figuring out how to get the functionality of an $N \times N$ switch using fixed size, elementary devices, like 3×3 switches.

11.2.2 Switch Count

Another goal in designing a communication network is to use as few switches as possible. In a complete binary tree, there is one “root” switch at the top, and the number of switches doubles at successive rows, so the number of switches in an N -input complete binary tree is $1 + 2 + 4 + 8 + \cdots + N$. So the total number of switches is $2N - 1$ by the formula for geometric sums (Problem 5.4). This is nearly the best possible with 3×3 switches.

11.2.3 Network Latency

We'll sometimes be choosing routings through a network that optimize some quantity besides delay. For example, in the next section we'll be trying to minimize packet congestion. When we're not minimizing delay, shortest routings are not always the best, and in general, the delay of a packet will depend on how it is routed. For any routing, the most delayed packet will be the one that follows the longest path in the routing. The length of the longest path in a routing is called its *latency*.

distance between inputs and outputs, not between arbitrary pairs of vertices.

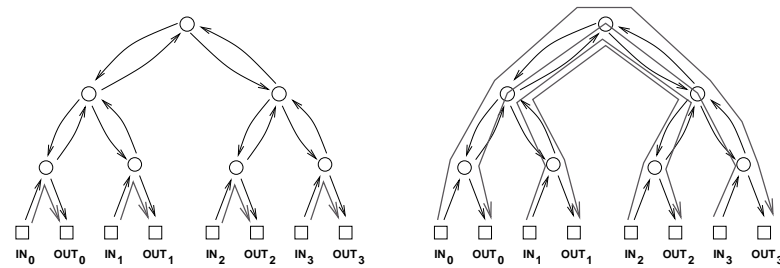


Figure 11.2 Two Routings in the Binary Tree Net

The latency of a *network* depends on what’s being optimized. It is measured by assuming that optimal routings are always chosen in getting inputs to their specified outputs. That is, for each routing problem π , we choose an optimal routing that solves π . Then *network latency* is defined to be the largest routing latency among these optimal routings. Network latency will equal network diameter if routings are always chosen to optimize delay, but it may be significantly larger if routings are chosen to optimize something else.

For the networks we consider below, paths from input to output are uniquely determined (in the case of the tree) or all paths are the same length, so network latency will always equal network diameter.

11.2.4 Congestion

The complete binary tree has a fatal drawback: the root switch is a bottleneck. At best, this switch must handle right and vice-versa. Passing all these packets through a single switch could take a long time. At worst, if this switch fails, the network is broken into two equal-sized pieces.

It’s true that if the routing problem is given by the identity permutation, $\text{Id}(i) ::= i$, then there is an easy routing P that solves the problem: let P_i be the path from input i up through one switch and back down to output i . On the other hand, if the problem was given by $\pi(i) ::= (N - 1) - i$, then in *any* solution Q for π , each path Q_i beginning at input i must eventually loop all the way up through the root switch and then travel back down to output $(N - 1) - i$. These two situations are illustrated in Figure 11.2. We can distinguish between a “good” set of paths and a “bad” set based on congestion. The *congestion* of a routing P is equal to the largest number of paths in P that pass through a single switch. For example, the congestion of the routing on the left is 1, since at most 1 path passes through each switch. However, the congestion of the routing on the right is 4, since 4 paths pass through the root switch (and the two switches directly below the root). Generally, lower congestion is better since packets can be delayed at an overloaded switch.

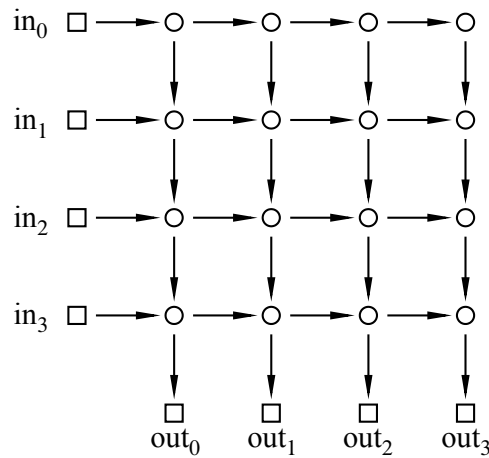


Figure 11.3 Two-dimensional Array with $N = 4$.

By extending the notion of congestion to networks, we can also distinguish between “good” and “bad” networks with respect to bottleneck problems. For each routing problem π for the network, we assume a routing is chosen that optimizes congestion, that is, that has the minimum congestion among all routings that solve π . Then the largest congestion that will ever be suffered by a switch will be the maximum congestion among these optimal routings. This “maximin” congestion is called the *congestion of the network*.

So for the complete binary tree, the worst permutation would be $\pi(i) ::= (N - 1) - i$. Then in every possible solution for π , *every* packet would have to follow a path passing through the root switch. Thus, the max congestion of the complete binary tree is N —which is horrible!

Let’s tally the results of our analysis so far:

network	diameter	switch size	# switches	congestion
complete binary tree	$2 \log N + 2$	3×3	$2N - 1$	N

11.3 Network Designs

11.3.1 2-D Array

Communication networks can also be designed as *2-dimensional arrays* or *grids*. A 2-D array with four inputs and outputs is shown in Figure 11.3.

The diameter in this example is 8, which is the number of edges between input 0

and output 3. More generally, the diameter of an array with N inputs and outputs is $2N$, which is much worse than the diameter of $2 \log N + 2$ in the complete binary tree. But we get something in exchange: replacing a complete binary tree with an array almost eliminates congestion.

Theorem 11.3.1. *The congestion of an N -input array is 2.*

Proof. First, we show that the congestion is at most 2. Let π be any permutation. Define a solution P for π to be the set of paths, P_i , where P_i goes to the right from input i to column $\pi(i)$ and then goes down to output $\pi(i)$. Thus, the switch in row i and column j transmits at most two packets: the packet originating at input i and the packet destined for output j .

Next, we show that the congestion is at least 2. This follows because in any routing problem π , where $\pi(0) = 0$ and $\pi(N - 1) = N - 1$, two packets must pass through the lower left switch. ■

As with the tree, the network latency when minimizing congestion is the same as the diameter. That’s because all the paths between a given input and output are the same length.

Now we can record the characteristics of the 2-D array.

network	diameter	switch size	# switches	congestion
complete binary tree	$2 \log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2

The crucial entry here is the number of switches, which is N^2 . This is a major defect of the 2-D array; a network of size $N = 1000$ would require a *million* 2×2 switches! Still, for applications where N is small, the simplicity and low congestion of the array make it an attractive choice.

11.3.2 Butterfly

The Holy Grail of switching networks would combine the best properties of the complete binary tree (low diameter, few switches) and of the array (low congestion). The *butterfly* is a widely-used compromise between the two.

A good way to understand butterfly networks is as a recursive data type. The recursive definition works better if we define just the switches and their connections, omitting the terminals. So we recursively define F_n to be the switches and connections of the butterfly net with $N ::= 2^n$ input and output switches.

The base case is F_1 with 2 input switches and 2 output switches connected as in Figure 11.4.

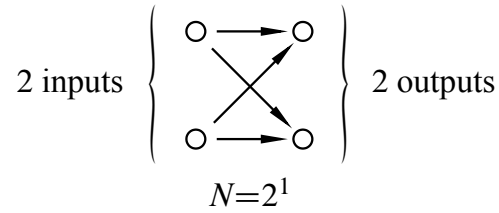


Figure 11.4 F_1 , the Butterfly Net with $N = 2^1$.

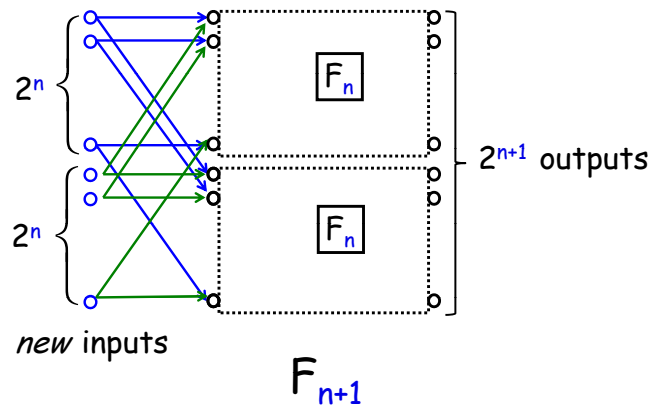


Figure 11.5 Butterfly Net F_{n+1} with 2^{n+1} inputs from two F_n 's.

In the constructor step, we construct F_{n+1} out of two F_n nets connected to a new set of 2^{n+1} input switches, as shown in as in Figure 11.5. That is, the i th and $2^n + i$ th new input switches are each connected to the same two switches, the i th input switches of each of two F_n components for $i = 1, \dots, 2^n$. The output switches of F_{n+1} are simply the output switches of each of the F_n copies.

So F_{n+1} is laid out in columns of height 2^{n+1} by adding one more column of switches to the columns in F_n . Since the construction starts with two columns when $n = 1$, the F_{n+1} switches are arrayed in $n + 1$ columns. The total number of switches is the height of the columns times the number $2^{n+1}(n + 1)$ of columns. Remembering that $n = \log N$, we conclude that the Butterfly Net with N inputs has $N(\log N + 1)$ switches.

Since every path in F_{n+1} from an input switch to an output is length- $n + 1$ the diameter of the Butterfly net with 2^{n+1} inputs is this length plus two because of the two edges connecting to the terminals (square boxes)—one edge from input terminal to input switch (circle) and one from output switch to output terminal.

There is an easy recursive procedure to route a packet through the Butterfly Net. In the base case, there is only one way to route a packet from one of the two inputs to one of the two outputs. Now suppose we want to route a packet from an input switch to an output switch in F_{n+1} . If the output switch is in the “top” copy of F_n , then the first step in the route must be from the input switch to the unique switch it is connected to in the top copy; the rest of the route is determined by recursively routing the rest of the way in the top copy of F_n . Likewise, if the output switch is in the “bottom” copy of F_n , then the first step in the route must be to the switch in the bottom copy, and the rest of the route is determined by recursively routing in the bottom copy of F_n . In fact, this argument shows that the routing is *unique*: there is exactly one path in the Butterfly Net from each input to each output, which implies that the network latency when minimizing congestion is the same as the diameter.

The congestion of the butterfly network is about \sqrt{N} . More precisely, the congestion is \sqrt{N} if N is an even power of 2 and $\sqrt{N/2}$ if N is an odd power of 2. A simple proof of this appears in Problem 11.8.

Let’s add the butterfly data to our comparison table:

network	diameter	switch size	# switches	congestion
complete binary tree	$2 \log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2
butterfly	$\log N + 2$	2×2	$N(\log(N) + 1)$	\sqrt{N} or $\sqrt{N/2}$

The butterfly has lower congestion than the complete binary tree. It also uses fewer switches and has lower diameter than the array. However, the butterfly does not capture the best qualities of each network, but rather is a compromise somewhere between the two. Our quest for the Holy Grail of routing networks goes on.

11.3.3 Beneš Network

In the 1960’s, a researcher at Bell Labs named Václav E. Beneš had a remarkable idea. He obtained a marvelous communication network with congestion 1 by placing *two* butterflies back-to-back. This amounts to recursively growing *Beneš nets* by adding both inputs and outputs at each stage. Now we recursively define B_n to be the switches and connections (without the terminals) of the Beneš net with $N ::= 2^n$ input and output switches.

The base case B_1 with 2 input switches and 2 output switches is exactly the same as F_1 in Figure 11.4.

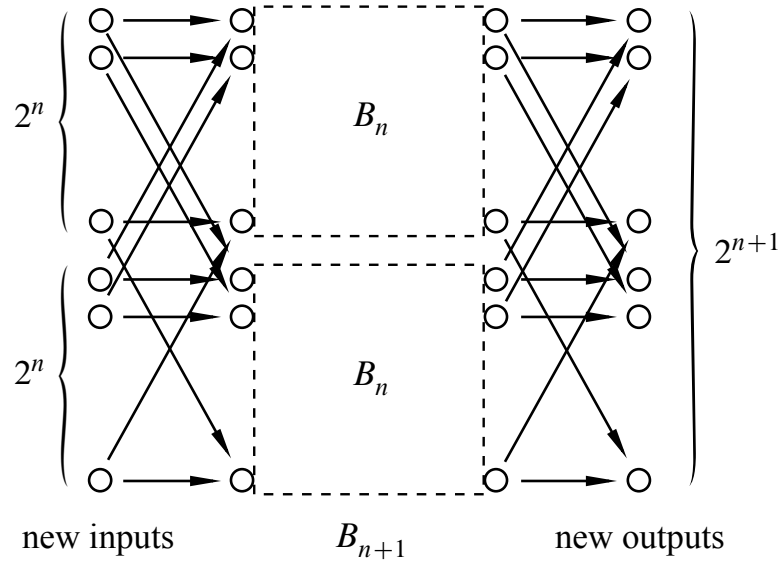


Figure 11.6 Beneš Net B_{n+1} with 2^{n+1} inputs from two B_n 's.

In the constructor step, we construct B_{n+1} out of two B_n nets connected to a new set of 2^{n+1} input switches *and also* a new set of 2^{n+1} output switches. This is illustrated in Figure 11.6.

The i th and $2^n + i$ th new input switches are each connected to the same two switches: the i th input switches of each of two B_n components for $i = 1, \dots, 2^n$, exactly as in the Butterfly net. In addition, the i th and $2^n + i$ th new *output* switches are connected to the same two switches, namely, to the i th output switches of each of two B_n components.

Now, B_{n+1} is laid out in columns of height 2^{n+1} by adding two more columns of switches to the columns in B_n . So, the B_{n+1} switches are arrayed in $2(n + 1)$ columns. The total number of switches is the number of columns times the height $2(n + 1)2^{n+1}$ of the columns.

All paths in B_{n+1} from an input switch to an output are length $2(n + 1) - 1$, and the diameter of the Beneš net with 2^{n+1} inputs is this length plus two because of the two edges connecting to the terminals.

So Beneš has doubled the number of switches and the diameter, but by doing so he has completely eliminated congestion problems! The proof of this fact relies on a clever induction argument that we'll come to in a moment. Let's first see how the

Beneš network stacks up:

network	diameter	switch size	# switches	congestion
complete binary tree	$2 \log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2
butterfly	$\log N + 2$	2×2	$N(\log(N) + 1)$	\sqrt{N} or $\sqrt{N/2}$
Beneš	$2 \log N + 1$	2×2	$2N \log N$	1

The Beneš network has small size and diameter, and it completely eliminates congestion. The Holy Grail of routing networks is in hand!

Theorem 11.3.2. *The congestion of the N -input Beneš network is 1.*

Proof. By induction on n where $N = 2^n$. So the induction hypothesis is

$$P(n) ::= \text{the congestion of } B_n \text{ is 1.}$$

Base case ($n = 1$): $B_1 = F_1$ is shown in Figure 11.4. The unique routings in F_1 have congestion 1.

Inductive step: We assume that the congestion of an $N = 2^n$ -input Beneš network is 1 and prove that the congestion of a $2N$ -input Beneš network is also 1.

Digression. Time out! Let’s work through an example, develop some intuition, and then complete the proof. In the Beneš network shown in Figure 11.7 with $N = 8$ inputs and outputs, the two 4-input/output subnetworks are in dashed boxes.

By the inductive assumption, the subnetworks can each route an arbitrary permutation with congestion 1. So if we can guide packets safely through just the first and last levels, then we can rely on induction for the rest! Let’s see how this works in an example. Consider the following permutation routing problem:

$$\begin{array}{ll} \pi(0) = 1 & \pi(4) = 3 \\ \pi(1) = 5 & \pi(5) = 6 \\ \pi(2) = 4 & \pi(6) = 0 \\ \pi(3) = 7 & \pi(7) = 2 \end{array}$$

We can route each packet to its destination through either the upper subnetwork or the lower subnetwork. However, the choice for one packet may constrain the choice for another. For example, we cannot route both packet 0 and packet 4 through the same network, since that would cause two packets to collide at a single switch, resulting in congestion. Rather, one packet must go through the upper network and the other through the lower network. Similarly, packets 1 and 5, 2 and

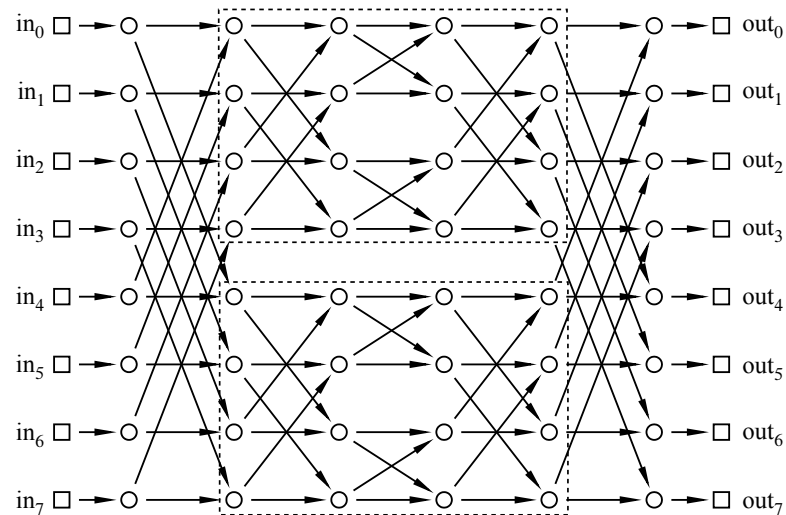
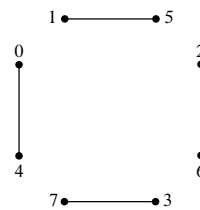


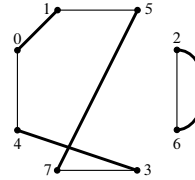
Figure 11.7 Beneš net B_3 .

6, and 3 and 7 must be routed through different networks. Let’s record these constraints in a graph. The vertices are the 8 packets. If two packets must pass through different networks, then there is an edge between them. Thus, our constraint graph looks like this:



Notice that at most one edge is incident to each vertex.

The output side of the network imposes some further constraints. For example, the packet destined for output 0 (which is packet 6) and the packet destined for output 4 (which is packet 2) cannot both pass through the same network; that would require both packets to arrive from the same switch. Similarly, the packets destined for outputs 1 and 5, 2 and 6, and 3 and 7 must also pass through different switches. We can record these additional constraints in our graph with gray edges:



Notice that at most one new edge is incident to each vertex. The two lines drawn between vertices 2 and 6 reflect the two different reasons why these packets must be routed through different networks. However, we intend this to be a simple graph; the two lines still signify a single edge.

Now here’s the key insight: suppose that we could color each vertex either red or blue so that adjacent vertices are colored differently. Then all constraints are satisfied if we send the red packets through the upper network and the blue packets through the lower network. Such a *2-coloring of the graph corresponds to a solution to the routing problem*. The only remaining question is whether the constraint graph is 2-colorable, which is easy to verify:

Lemma 11.3.3. *Prove that if the edges of a graph can be grouped into two sets such that every vertex has at most 1 edge from each set incident to it, then the graph is 2-colorable.*

Proof. It is not hard to show that a graph is 2-colorable iff every cycle in it has even length (see Theorem 12.8.3). We’ll take this for granted here.

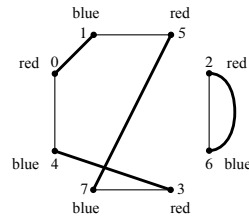
So all we have to do is show that every cycle has even length. Since the two sets of edges may overlap, let’s call an edge that is in both sets a *doubled edge*.

There are two cases:

Case 1: [The cycle contains a doubled edge.] No other edge can be incident to either of the endpoints of a doubled edge, since that endpoint would then be incident to two edges from the same set. So a cycle traversing a doubled edge has nowhere to go but back and forth along the edge an even number of times.

Case 2: [No edge on the cycle is doubled.] Since each vertex is incident to at most one edge from each set, any path with no doubled edges must traverse successive edges that alternate from one set to the other. In particular, a cycle must traverse a path of alternating edges that begins and ends with edges from different sets. This means the cycle has to be of even length. ■

For example, here is a 2-coloring of the constraint graph:



The solution to this graph-coloring problem provides a start on the packet routing problem:

We can complete the routing in the two smaller Beneš networks by induction! Back to the proof. **End of Digression.**

Let π be an arbitrary permutation of $[0..N - 1]$. Let G be the graph whose vertices are packet numbers $0, 1, \dots, N - 1$ and whose edges come from the union of these two sets:

$$E_1 ::= \{ \langle u-v \rangle \mid |u - v| = N/2 \}, \text{ and}$$

$$E_2 ::= \{ \langle u-w \rangle \mid |\pi(u) - \pi(w)| = N/2 \}.$$

Now any vertex u is incident to at most two edges: a unique edge $\langle u-v \rangle \in E_1$ and a unique edge $\langle u-w \rangle \in E_2$. So according to Lemma 11.3.3, there is a 2-coloring for the vertices of G . Now route packets of one color through the upper subnetwork and packets of the other color through the lower subnetwork. Since for each edge in E_1 , one vertex goes to the upper subnetwork and the other to the lower subnetwork, there will not be any conflicts in the first level. Since for each edge in E_2 , one vertex comes from the upper subnetwork and the other from the lower subnetwork, there will not be any conflicts in the last level. We can complete the routing within each subnetwork by the induction hypothesis $P(n)$. ■

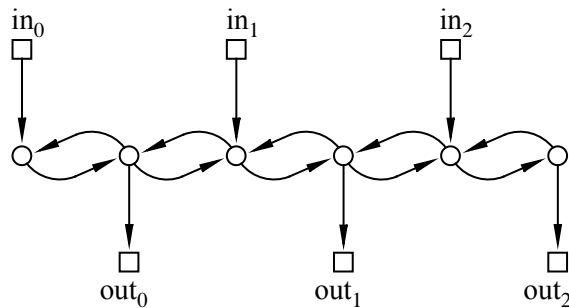
Problems for Section 11.2

Exam Problems

Problem 11.1.

Consider the following communication network:

- (a) What is the max congestion?
- (b) Give an input/output permutation π_0 that forces maximum congestion.
- (c) Give an input/output permutation π_1 that allows *minimum* congestion.



(d) What is the latency for the permutation π_1 ? (If you could not find π_1 , just choose a permutation and find its latency.)

Problems for Section 11.3

Class Problems

Problem 11.2.

The Beneš network has a max congestion of one—every permutation can be routed in such a way that a single packet passes through each switch. Let’s work through an example. A diagram of the Beneš network B_3 of size $N = 8$ appears in Figure 11.7. The two subnetworks of size $N = 4$ are marked. We’ll refer to these as the *upper* and *lower* subnetworks.

(a) Now consider the following permutation routing problem:

$$\begin{array}{ll} \pi(0) = 3 & \pi(4) = 2 \\ \pi(1) = 1 & \pi(5) = 0 \\ \pi(2) = 6 & \pi(6) = 7 \\ \pi(3) = 5 & \pi(7) = 4 \end{array}$$

Each packet must be routed through either the upper subnetwork or the lower subnetwork. Construct a graph with vertices numbered by integers 0 to 7 and draw a *dashed* edge between each pair of packets that cannot go through the same subnetwork because a collision would occur in the second column of switches.

(b) Add a *solid* edge in your graph between each pair of packets that cannot go through the same subnetwork because a collision would occur in the next-to-last column of switches.

(c) Assign colors red and blue to the vertices of your graph so that vertices that are adjacent by either a dashed or a solid edge get different colors. Why must this be possible, regardless of the permutation π ?

(d) Suppose that red vertices correspond to packets routed through the upper subnetwork and blue vertices correspond to packets routed through the lower subnetwork. Referring to the Beneš network shown in Figure 11.6, indicate the first and last edge traversed by each packet.

(e) All that remains is to route packets through the upper and lower subnetworks. One way to do this is by applying the procedure described above recursively on each subnetwork. However, since the remaining problems are small, see if you can complete all the paths on your own.

Problem 11.3.

A *multiple binary-tree network* has N inputs and N outputs, where N is a power of 2. Each input is connected to the root of a binary tree with $N/2$ leaves and with edges pointing away from the root. Likewise, each output is connected to the root of a binary tree with $N/2$ leaves and with edges pointing toward the root.

Two edges point from each leaf of an input tree, and each of these edges points to a leaf of an output tree. The matching of leaf edges is arranged so that for every input and output tree, there is an edge from a leaf of the input tree to a leaf of the output tree, and every output tree leaf has exactly two edges pointing to it.

(a) Draw such a multiple binary-tree net for $N = 4$.

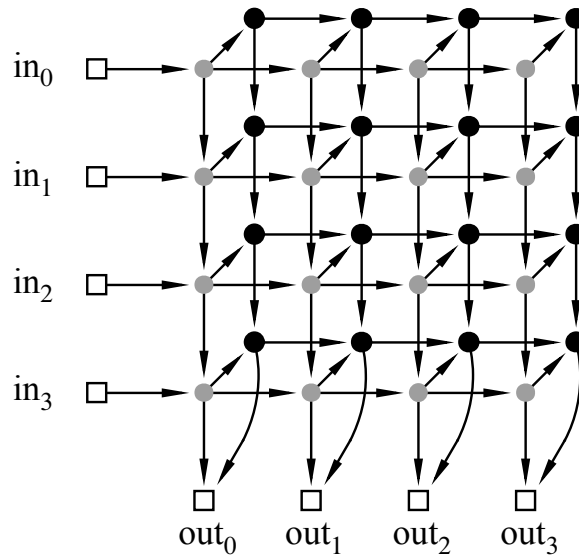
(b) Fill in the table, and explain your entries.

# switches	switch size	diameter	max congestion

Problem 11.4.

The n -input 2-D array network was shown to have congestion 2. An n -input 2-layer array consisting of two n -input 2-D Arrays connected as pictured below for $n = 4$.

In general, an n -input 2-layer array has two layers of switches, with each layer connected like an n -input 2-D array. There is also an edge from each switch in the first layer to the corresponding switch in the second layer. The inputs of the 2-layer



array enter the left side of the first layer, and the n outputs leave from the bottom row of either layer.

(a) For any given input-output permutation, there is a way to route packets that achieves congestion 1. Describe how to route the packets in this way.

(b) What is the latency of a routing designed to minimize latency?

(c) Explain why the congestion of any minimum latency (CML) routing of packets through this network is greater than the network's congestion.

Problem 11.5.

A 5-path communication network is shown below. From this, it's easy to see what an n -path network would be. Fill in the table of properties below, and be prepared to justify your answers.

network	# switches	switch size	diameter	max congestion
5-path				
n -path				

Problem 11.6.

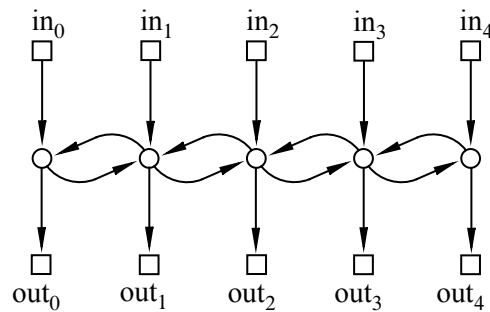
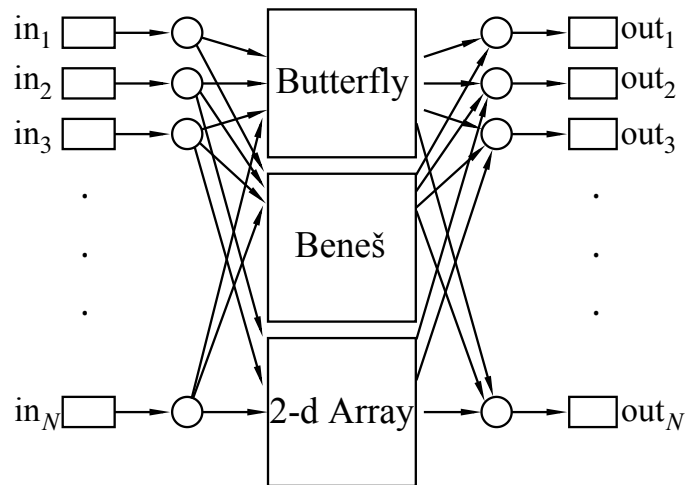


Figure 11.8 5-Path

Tired of being a TA, Megumi has decided to become famous by coming up with a new, better communication network design. Her network has the following specifications: every input node will be sent to a butterfly network, a Beneš network and a 2-d array network. At the end, the outputs of all three networks will converge on the new output.

In the Megumi-net a minimum latency routing does not have minimum congestion. The *latency for min-congestion (LMC)* of a net is the best bound on latency achievable using routings that minimize congestion. Likewise, the *congestion for min-latency (CML)* is the best bound on congestion achievable using routings that minimize latency.



Fill in the following chart for Megumi's new net and explain your answers.

network	diameter	# switches	congestion	LMC	CML
Megumi’s net					

Homework Problems

Problem 11.7.

Louis Reasoner figures that, wonderful as the Beneš network may be, the butterfly network has a few advantages, namely: fewer switches, smaller diameter, and an easy way to route packets through it. So Louis designs an N -input/output network he modestly calls a *Reasoner-net* with the aim of combining the best features of both the butterfly and Beneš nets:

The i th input switch in a Reasoner-net connects to two switches, a_i and b_i , and likewise, the j th output switch has two switches, y_j and z_j , connected to it. Then the Reasoner-net has an N -input Beneš network connected using the a_i switches as input switches and the y_j switches as its output switches. The Reasoner-net also has an N -input butterfly net connected using the b_i switches as inputs and the z_j switches as outputs.

In the Reasoner-net a minimum latency routing does not have minimum congestion. The *latency for min-congestion (LMC)* of a net is the best bound on latency achievable using routings that minimize congestion. Likewise, the *congestion for min-latency (CML)* is the best bound on congestion achievable using routings that minimize latency.

Fill in the following chart for the Reasoner-net and briefly explain your answers.

diameter	switch size(s)	# switches	congestion	LMC	CML

Problem 11.8.

Show that the congestion of the butterfly net, F_n , is exactly \sqrt{N} when n is even.

Hint:

- There is a unique path from each input to each output, so the congestion is the maximum number of messages passing through a vertex for any routing problem.

- If v is a vertex in column i of the butterfly network, there is a path from exactly 2^i input vertices to v and a path from v to exactly 2^{n-i} output vertices.
- At which column of the butterfly network must the congestion be worst?
What is the congestion of the topmost switch in that column of the network?

