Laboratory 15

Networking

Objectives

- Study how networks route packets to various destination hosts.
- Learn how networks ensure reliable delivery.

References

Software needed:

1) A web browser (Internet Explorer or Netscape)
2) Applets from the CD-ROM:
   a) TCP/IP
   b) Network router

Textbook reference: Chapter 15, pp. 476–483, 488–491

Background

Chapter 15, "Networks," explains the basic concepts; this lab elaborates on two aspects of computer networking.

Activity

Part 1

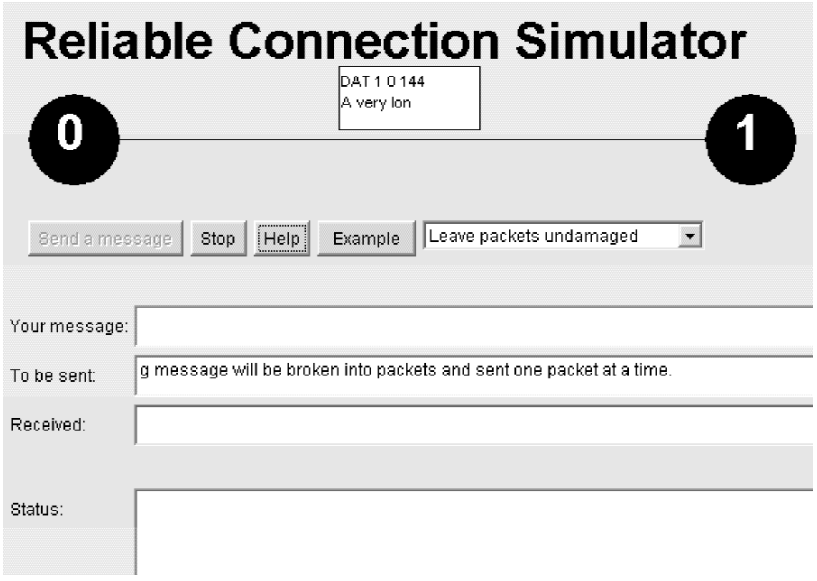
Though we tend to take networks for granted (except when they're down!), they are surprisingly complicated. Setting one up involves much more than just running a wire between two computers. Not only must the machines agree on a whole series of protocols and identifying conventions before any communication can happen, but the wire itself is subject to seemingly malevolent forces working to corrupt the fragile data traveling through it!

The TCP/IP protocol suite is a whole system of software, protocols, and management decisions. At its heart are several protocols that break messages into packets and send them from source computers to destination computers. In this activity, we will see a simplified form of TCP (Transmission Control Protocol) as it reliably sends a message to a destination computer.

Start the "TCP/IP" applet. This applet simulates a reliable connection, which means that a big message is correctly sent in its entirety from source to destination. If anything goes wrong, the software makes heroic attempts to recover the information. While it is impossible to absolutely guarantee that the message will arrive intact, the networking software can make it very likely that it will be delivered correctly.

In this applet, there are only two hosts, or nodes, numbered 0 and 1. Host 0 is a computer that is trying to send *your message* to host 1. You will play the role of one of those malevolent forces of nature, damaging and even destroying data packets. Will the software recover the data properly and save the day? Let's hope so!

Click on the *Example* button, which inserts a message into the *Your message* text area. Click *Run*, and then *Send a message*. Host 0 moves the message to the *To be sent* area and proceeds to "packetize" it into 10 character packets. Here you can see the first



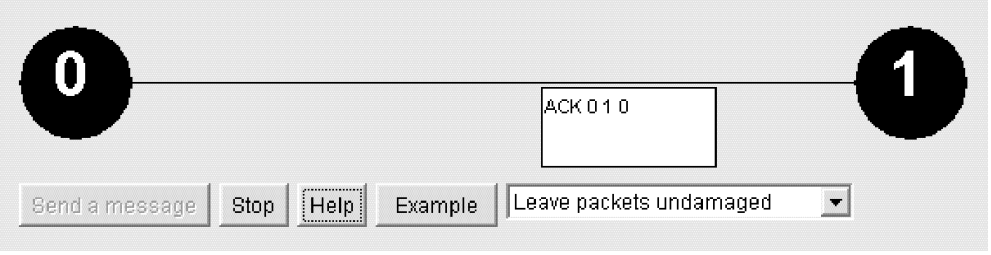
10 characters—`A very lon` (the two spaces are part of the count)—moving slowly in a packet across the wire.

What exactly does the `DAT 1 0 144` in the packet mean? This is the packet's *header* and contains crucial information. First, `DAT` identifies the type of packet being sent. There are three types of packets: `DAT` for *data*, `ACK` for *acknowledgment*, and NAK for *negative acknowledgment*. We will talk more about these in a minute.

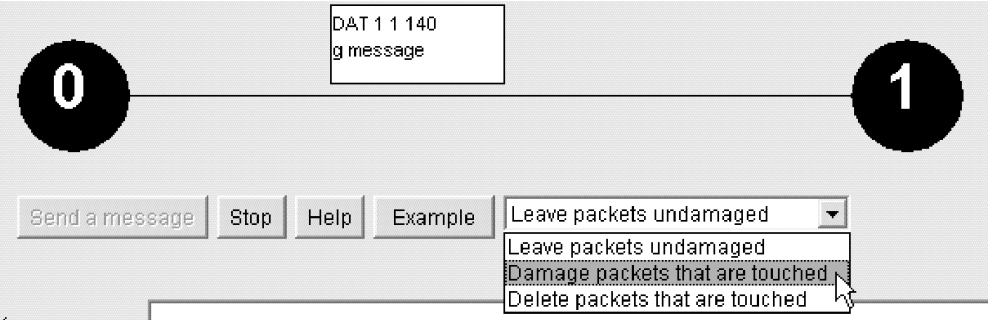
Next in the header comes the *destination address*, which is 1. Though this seems a bit silly in a two-node network, bigger networks obviously require a destination address. The number `0`, which appears next, is the *sequence number*. As messages are broken into packets, each packet is assigned a number: `0` in the first packet, 1 in the second packet, and so forth. If any packet were to arrive out of order, the destination computer could examine these sequence numbers and re-assemble the message in the correct order. After all, if part of the message were scrambled the meaning could change completely ("You owe us $1000" means something quite different from "You owe us 000$1".)

Finally, `144` is the *checksum*, which alerts the destination computer that the packet was damaged. This applet uses an extremely simple checksum algorithm, merely taking the ASCII value of each character in the packet, adding these together, and performing modulo 256 (keep the remainder after dividing by 256) so that the checksum is always between 0 and 255, inclusive. This checksum algorithm was actually used in some early protocols, but is prone to serious problems. Nowadays more advanced ones are used, most notably the *cyclic redundancy code*. We'd tell you how it works, but that would change our lab into an advanced mathematics class!

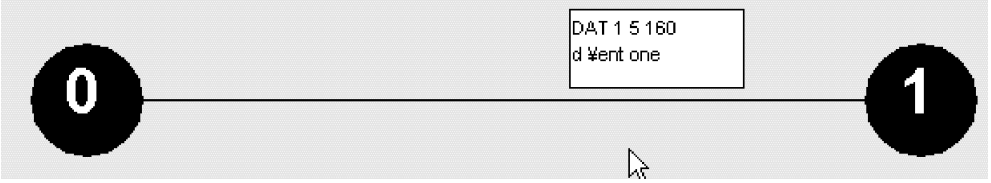
After the first packet arrives at host 1, it sends back an `ACK` packet to acknowledge that the data arrived successfully. The type of the packet is `ACK`; the destination is 0 (host 0), and the sequence number that it sends back is 1, meaning that it expects host 0 to send it a packet with the sequence number 1 next. Finally, there is no real data in an `ACK` packet, so the checksum is `0`. This explains why the header is `ACK 0 1 0`.



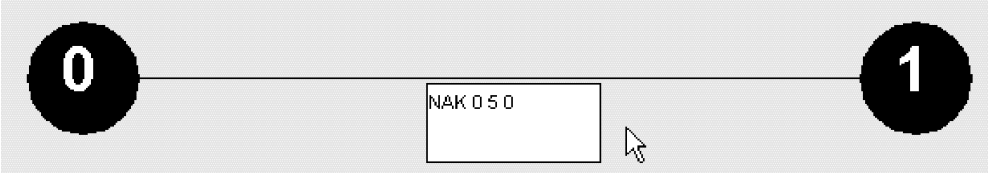
Networks are always contending with the forces of chaos that can corrupt the data being sent: noise on the line, equipment failures, and other sorts of mayhem. So let's wreak havoc on our poor little network and see what happens. Select *Damage packets that are touched* from the pull-down menu. You can do this while the applet is sending packets.



As a packet comes sliding along the wire, click on it with the mouse. This should somehow corrupt the data. For example, the s below turned into the Japanese yen symbol!

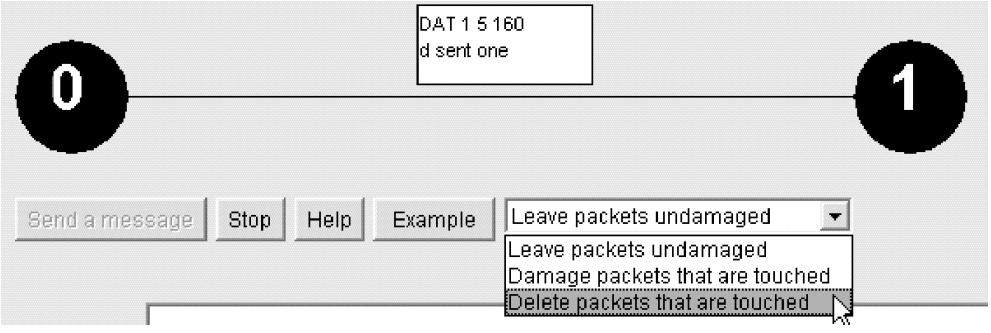


What does host 1 do when this packet arrives? How does it even know that the packet is damaged? It re-computes the checksum using the data it receives, and it matches the checksum in the header with the re-computed one. They don't match this time, so host 1 sends host 0 a NAK packet with the same sequence number. This tells host 0 that the packet was corrupted and needs to be re-sent.



When Host 0 gets the NAK, it re-sends the packet. Various status messages appear in the large text area at the bottom of the applet, explaining what is going on.

Now let's experiment with more ways to make life difficult. Select *Delete packets that are touched* from the pull-down menu. When another packet comes along, click on it with your mouse and it should disappear.



Packets that do not even arrive pose a bigger challenge than ones that are damaged. How does host 1 ever know that a packet is missing? Actually, it doesn't, but host 0 is looking for an `ACK` or `NAK` packet in response to each `DAT` packet that it sends. So if one never arrives, host 0 assumes the packet is lost, and it sends the packet again.

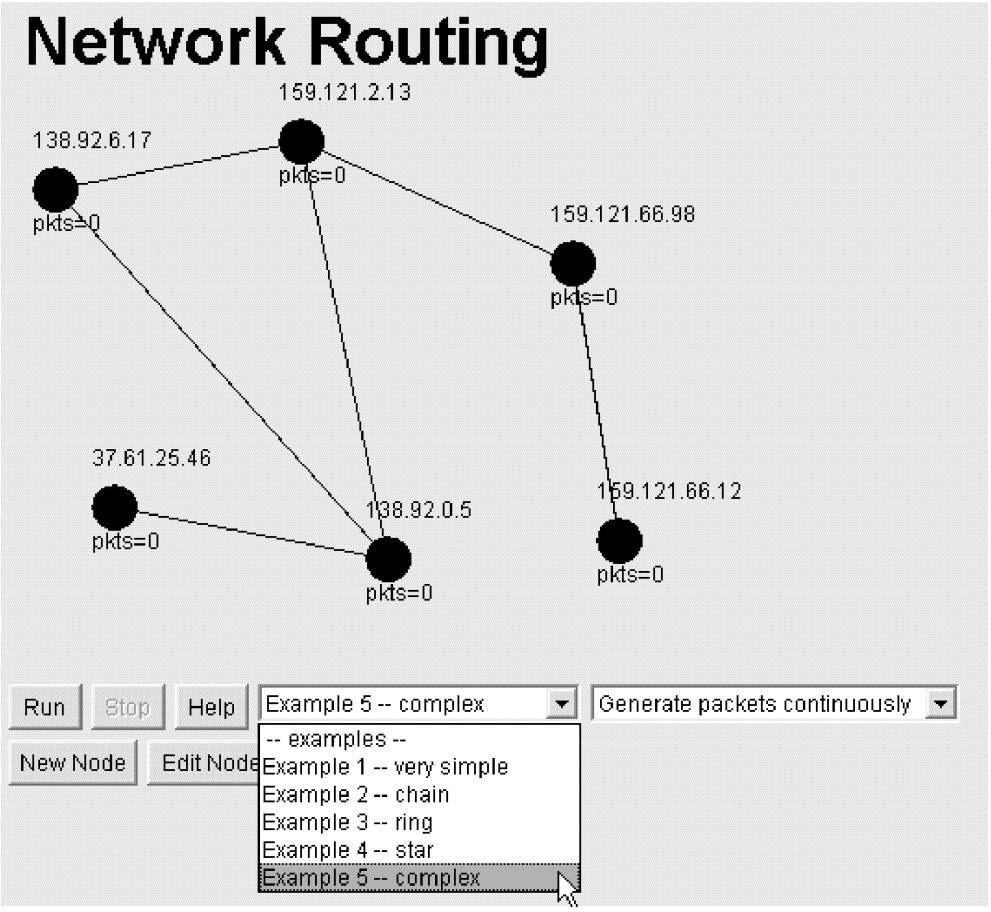
The host won't wait forever to receive the `ACK` or `NAK` packet It sets a timer on each packet and if nothing arrives before the timer counts down to 0, it *times out* and re-sends the packet.

Of course, all of this assumes that everything, including the hosts and the software, is working correctly to catch any errors that may occur. As you can imagine, there are about a million things that can happen to foul up this pretty picture, and we will explore some of them in the exercise. But consider this: What if the packets are temporarily held up somehow? They aren't lost, but the timer goes off anyway. So what does host 1 do when it gets a duplicate packet? And does it send another ACK? Yikes! This networking is rapidly spinning out of control! Scientists who designed the Internet had to think about all these possibilities and decide how to handle them. Fortunately, they found reliable solutions so today we can sit in the comfort of our homes and travel the entire world with our fingertips.

Part 2

The second applet simulates how computers *route* packets to their final destinations. Routing is enormously complicated because there are so many variables. We will see some of this unexpected complexity as we watch this applet.

Start the "Network router" applet. Let's go for the most complex example right away. Select *Example 5* from the pull-down menu.

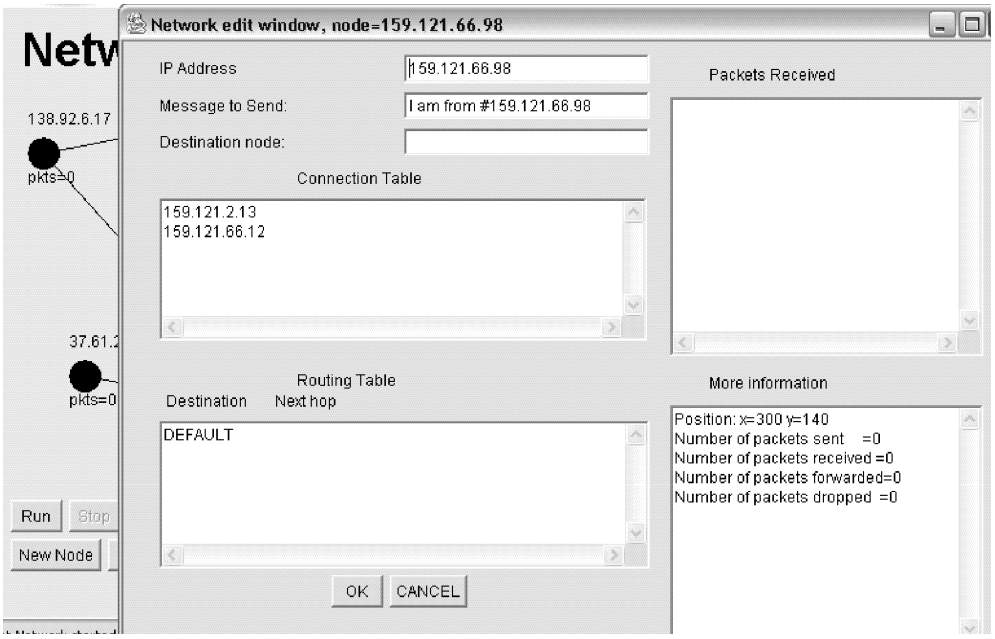


Each blue disk represents a node, or a computer in the network. The lines between nodes are telecommunication lines, perhaps telephone wires or Ethernet cables. Each node has its own IP address, a 32-bit number that is always represented as four octets separated by dots (see textbook pp. 488–489).

In this applet, the lines are *full duplex*, which means that packets can flow in either direction between the two connected hosts. This isn't always the case in the real world, but many lines are full duplex.

You can move the nodes around on the screen by dragging them. Their attached lines move with them. You can even add new nodes. To make a new node, double-click on any open space and the applet places there a blue dot with the unusable address of 0.0.0.0. You are expected to change that right away!

To change the characteristics of a node, double-click on the node and a new window appears:



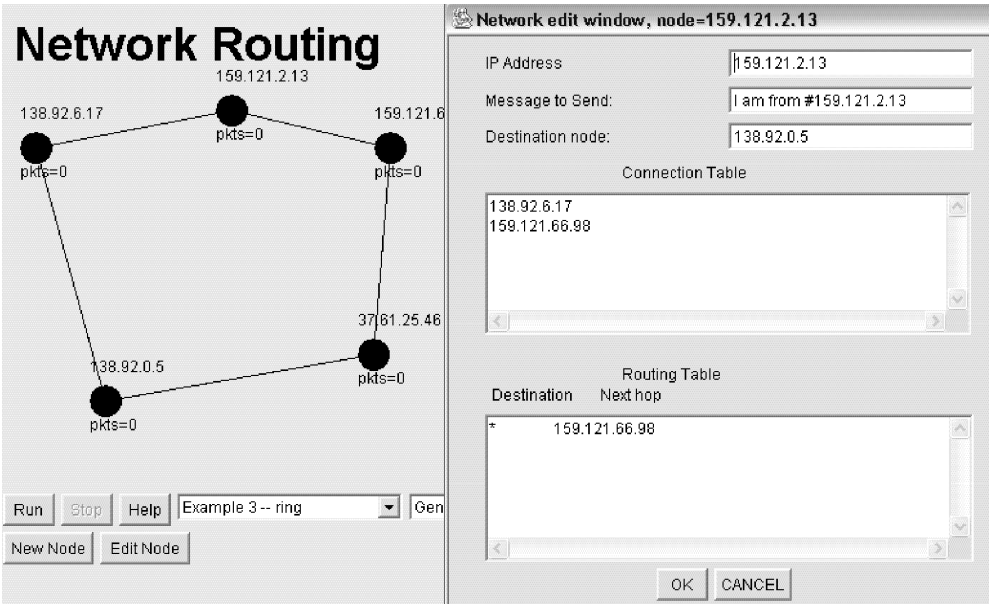
This window displays status information in the right column, showing packets that this node has received and some statistics. At the top are three fields that you can change, the first being the IP address of the node. You can also customize the message you want to send, and you can add the address of the recipient. If the *Destination node* field is blank, no messages will be sent by this node. More than one node can send and receive messages at the same time when this applet runs.

The connection and routing tables below these fields are the focus of this applet's study. The connection table lists the IP addresses of nodes that *directly connect* to this node. Most networks assume, quite logically, that two nodes can find each other and send packets if they are directly connected. It is when nodes are not directly connected to their destinations that things get interesting!

The purpose of the routing table is to tell the node what to do when it is not directly connected to the destination node. For example, if node X wants to send a packet to node Y, which is not directly connected, it must send the packet to node Z instead, and node Z then assumes the responsibility of getting it to node Y. Hopefully, Z will not mistakenly send it back to X! If so, a *routing loop* would occur and the packet would travel the wires endlessly, like the Flying Dutchman. This prospect so worried early developers of TCP/IP that they put a timer on each packet so that it would just evaporate after going through too many nodes.

The routing table has two columns. The first column lists the destination, and the second lists the next hop. The next hop must be a directly connected node. There are also a number of special cases that can shorten routing tables. If the word `DEFAULT` appears in the routing table, then the node merely sends packets to the first node in the directly connected table. This works nicely for *leaf* nodes, which connect to only one other node, but won't work for a complex topology (*topology* refers to the configuration of the network; various topologies are shown on p. 478 of your text).

Most of the time, nodes with two or more connections need to explicitly list all distant nodes, along with the next hop. However, this quickly gets tedious and requires too much work to update, as many new nodes are added. One possible alternative is to list some of the destination/next hop pairs, and leave the others unspecified. The "Network router" applet lets you do this by putting an asterisk in the destination column, as shown below for the ring network (*Example 3*):



Finally, what can we do if the network is under attack and the routing table is now obsolete due to out-of-commission routers? Early TCP/IP developers worried about this, too, and came up with the *hot potato algorithm*. This applet allows you to type `HOT POTATO` in the routing table. Nothing else is needed. Hot potato works this way: When a packet comes in, the node makes a random guess as to which directly connected node will get the packet to its final destination, and it sends it to that next hop. This may not work and it may result in routing loops, but it just might do the trick, too. Imagine a bunch of people on the beach around a campfire passing around a hot sweet potato wrapped in aluminum foil. As they energetically toss it to one another, there is a good chance that it will end up in the hands (or lap) of someone who wants to eat it!

There are many other routing algorithms that were designed with various emergencies or unusual conditions in mind. One such algorithm is called *flooding*, in which every packet is copied to every outgoing wire, not just to one random one, as in hot potato.

What happens in the Internet, which is built on TCP/IP? What routing algorithms are used? The local area networks that attach to the Internet often use specialized routing algorithms. For example, Ethernet is like a party-line telephone where everyone can hear everyone else's calls. So when the phone rings, or a new packet comes in, each computer listens to the beginning of the packet to see if it is meant for them.

The backbone network that manages the long-distance, high-volume traffic on the Internet uses several *dynamic* routing protocols. In such systems, nodes change their routing tables from time to time as they are forwarding packets. Unlike the "Network router" applet, in which you, the *network administrator*, decide once and for all what the routing table for node X is, dynamic routing systems measure the characteristics of

the traffic and the attached lines, and adjust their routing tables so that packets will flow most efficiently through the entire system. This sounds daunting and it is, kind of like continuously adjusting the timing on all traffic lights so that cars on the streets get to their destinations in the least amount of time. While the system isn't perfect, it's worked on the Internet for over 30 years.

Exercise 1

Name _____ Date _____

Section _____

1) Start the "TCP/IP" (reliable connection simulator) applet.
2) In the textfield labeled "Your message:", type the following:

   Computer networking is essential in our world today.

   Then press the button "Send a message."
3) Watch the entire sequence of packets that are sent for the sample message. How many `DAT` packets were sent? How many `ACK` packets were sent?

4) If each character or blank in a packet header counts for one character, and all `DAT` packets except the last carry 10 characters, and `ACK` packets have no data characters (though they do have a header), count up how many characters were sent in total.

5) There are 52 characters in the *Example* message, including blanks and punctuation. Subtract 52 from the total number of characters sent in both directions in all packets. (Do not damage or delete packets this time.) Divide this number by the total number of characters to get the *overhead*, expressed as a percentage.

6) Imagine that you have a million-character message to send, perhaps a large file. How many characters total would be sent in all packets necessary to move it from node 0 to node 1?

7) What would be an obvious way to decrease the overhead? Why might this solution backfire? Under what conditions?

Exercise 2

Name _____ Date _____

Section _____

1) Repeat Steps 1 and 2 from Exercise 1.
2) Select *Delete packets that are touched.*
3) Delete some data packets by clicking on them as they move along the wire and watch the re-transmission after timeout.
4) What happens if you delete the re-transmitted packet? Does the "TCP/IP" applet need to take any special action?

5) Now try deleting some `ACK` or `NAK` packets. What happens?

Exercise 3

Name _____ Date _____

Section _____

1) See if you can compute the checksum as TCP/IP does. Run the applet, using any message. Select a data packet, but don't select the last packet because it might be too short—less than 10 characters.
2) Count up the characters, including blanks. If they do not equal 10, assume there are blanks at the end so that the character count is 10.
3) Using the "Text Translator into ASCII" applet from Lab 3b, type in the characters from the data packet, and click on *translate text*. (You can use an ASCII chart instead if you'd prefer.)
4) Add up the values for each of the characters. Then take the modulus of this number using 256. The modulus operator is available on some computer calculators and is represented by *mod*, but you can easily compute it by dividing the total by 256 and saving only the remainder. For instance, suppose the total is 7452. Since $7452 \div 256 = 29.109375$, but we only want the remainder. Multiply 0.109375 by 256, which gives us 28. Or: $7452 - (29 \times 256) = 28$.
5) Compare your computed checksum against what the applet shows for the packet. Do they agree?
6) Now damage your packet by altering one character. Re-compute the checksum. Do you see how TCP/IP can spot errors?
7) Think of a way that a packet can be damaged and still have the same checksum as the undamaged version. (There are several possibilities. Imagine that two or more bytes are altered at the same time.)

Exercise 4

Name _____ Date _____

Section _____

1) Start the "Network router" applet. Select *Example 3*, the ring network.
2) Become familiar with the applet. Move a few nodes around by dragging them.
3) Double-click on the node 37.61.25.46. List the nodes it is directly connected to.

4) If 37.61.25.46 wants to send packets to a node that is not directly connected, to which node will it first send the packets? (Check the routing table by double clicking on the node.)

5) Run the applet for a while, letting it generate packets continuously. Double-click on 37.61.25.46 again and look at its statistics. How many packets were sent? Received? Forwarded?

6) Who else is sending messages, and to whom?

7) Click on 138.92.6.17. Write down its statistics.

Exercise 5

Name _____ Date _____

Section _____

1) Start the "Network router" applet. Select *Example 4*, the star network.
2) Look at the routing and connection tables for the center node and several other nodes. Describe any pattern you can see in these tables.
3) How is the connection table for the center node different from the other nodes?

4) Select *Generate when I click on a node* from the pull-down menu. This means a user at this computer wants to send packets to the 126.14.5.46 computer.
5) If you double-click on 159.121.2.13, you will see that its destination node is 126.14.5.46. Run the applet, click on 159.121.2.13, and watch the packets go. What color does the sending computer turn briefly? What color does the destination computer turn? What does it mean if a node flashes green?

6) Many early computer networks used the star topology (*Example 4* on the applet). What will happen if the center node in this type of network dies?

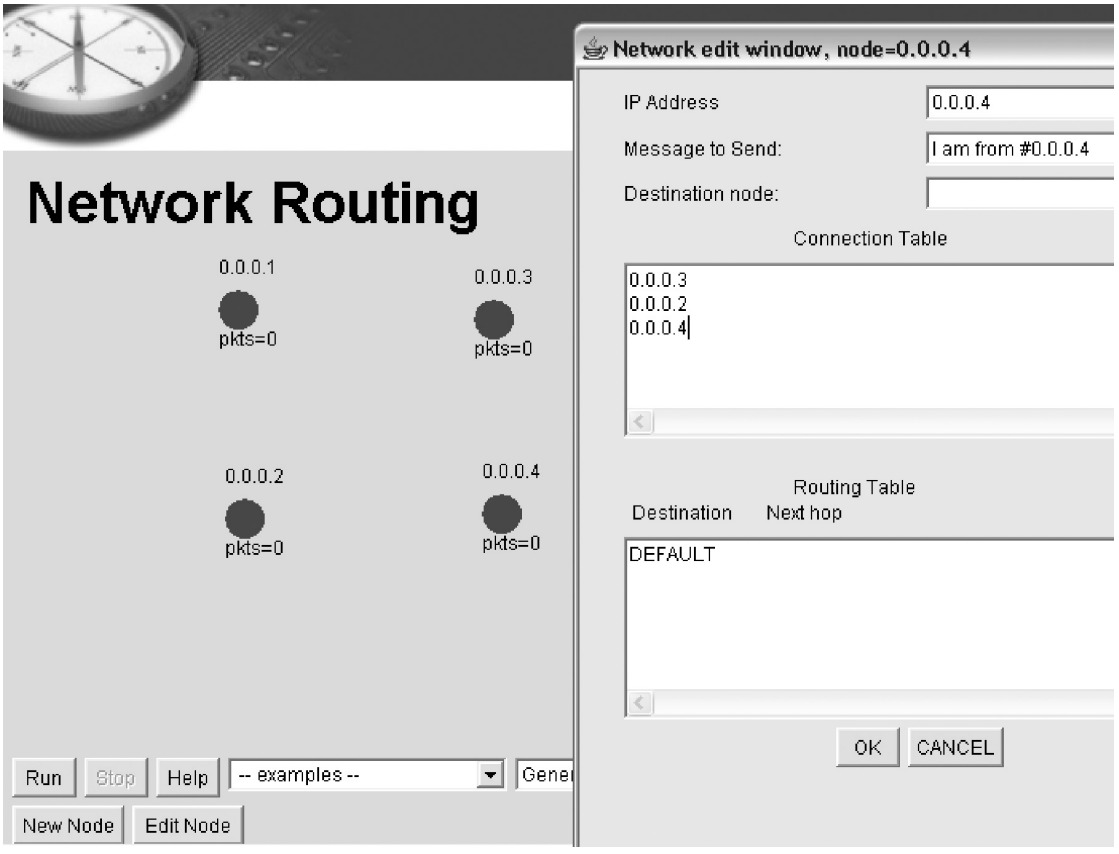
Exercise 6

Name _____ Date _____

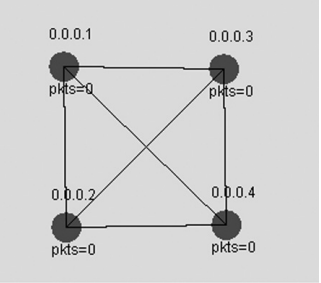
Section _____

1) Start the "Network router" applet but don't select an example network.

2) Click on the "New Node" button four times, which will place four nodes on your screen. Move them around into a square. Then double click on node 0.0.0.1. When the edit window appears, type the IP addresses of the other nodes into the Connection Table, as shown:



3) Now edit nodes 0.0.0.2, 0.0.0.3, and 0.0.0.4. In each case, add every other node to their Connection Tables. When you are done, every node will have a direct wire to every other node:



This is what is called a *directly connected network*. It is also called a *fully connected network*, for obvious reasons. Every node in the network has a direct connection to every other node. Is there a need for a routing table in this network?

4) Sketch out below a network of 2 nodes and make it directly connected. Do not bother to assign address numbers to the nodes. Just draw circles with lines between them.

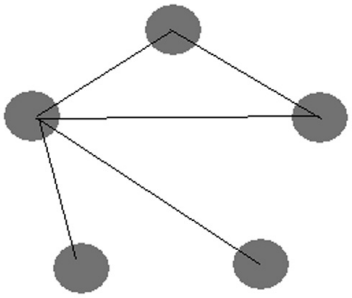
5) Now sketch out a network of 3 nodes and directly connect every node.

6) Fill in the table below, which compares the number of nodes in a directly connected network to the total number of wires in the network. Do not double count wires between the same nodes. That is, since there is a wire between 0.0.0.1 and 0.0.0.2, there is also a wire going the other direction, from 0.0.0.2 to 0.0.0.1. Count this as just one wire, not 2.

Number of nodes	Total number of wires
2	_____
3	_____
4	_____

7) Make a prediction for a 5-node network.

8) Now click on New Node to add a new node, whose number will be 0.0.0.5. Edit the other four nodes and add 0.0.0.5 to their Connection Tables. Once you are done, move your nodes around so that the nodes sit at the corners of a pentagon, as shown below. (Not all wires are shown.)



9) Fill in the table for 5. Did your guess match the final network?

10) Either sketch a 6-node network by hand or use the applet, building on your 5-node network. Complete the previous table by filling in the total number of wires. Are you surprised?

11) List one major advantage of a directly connected network.

12) List one major disadvantage of a directly connected network.

13) The telephone system is a network where each telephone is a node in the network. Do you imagine that the telephone system is a directly connected network? If not, why not?

Deliverables

Turn in your hand-written answers.

Deeper Investigation

One possible network topology is the *fully connected network*. In this topology, every node has a direct line to every other node. This would be ideal for speed! Draw five nodes and fully connect the network. What is the downside of this topology?

Draw fully connected networks for sizes two, three, four, six, and 10 nodes. Can you figure out a formula that predicts how many lines will be needed if there are n nodes in the network?

Fully connected networks are too costly in general (imagine every computer in the whole world having a direct wire to every other computer!). But star networks and others are not *fault tolerant*; they isolate some computers if one or more nodes dies or a communication wire breaks. Can you invent a network topology that is not fully connected, yet provides redundancy so that every node has at least two paths to other nodes?