Chapter 6

Broadcast Networks

The cable TV industry has changed a lot over the years. In 1973, most cable companies would have been completely confused if you called up to ask for high speed Internet service. It would be decades before cable systems began offering Internet access. The Internet did not even exist in 1973. The ARPANET, which would eventually grow up to become the Internet, provided interconnections between less than 50 computers at the time.

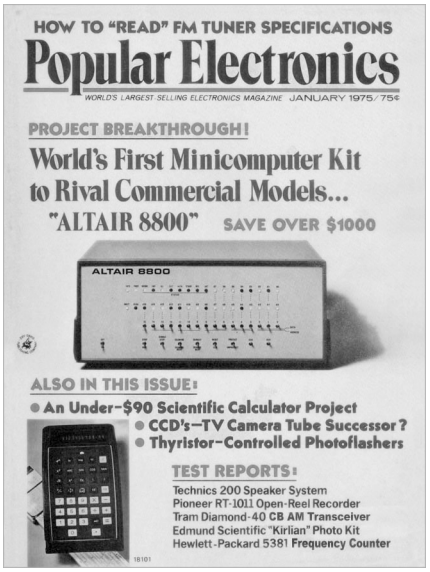

Figure 6.1: An early "PC"

Even if you did somehow get your cable company to provide your home with network access in 1973, it isn't clear what you would have done with it. Home computers were very rare in 1973, and if you did have one it would probably look more like the Altair 8800 pictured in Figure 6.1 than any computer you have every used. In fact, even the Altair wasn't available yet.

Surprisingly, there was at least one place where you could find a computer that would seem comfortably modern to you back in 1973. The place was a research center established just a few years earlier by the Xerox Corporation. The center was known as Xerox PARC, short for Xerox Palo Alto Research Center. The computer was named the Alto.

Figure 6.2 shows a picture of an Alto. Its components should look more familiar than those of the Altair. Instead of a panel of small flashing lights, it had a CRT display screen. In place of the row of toggle switches across the front panel of the Altair, the Alto had a keyboard and a mouse. The software provided with the Alto used these devices to offer the first GUI interface. The processor enclosure is large by modern standards, but all in all, the Alto looks quite a bit like a modern desktop system. At a time when quite a few computer systems still used punched cards, the researchers at Xerox PARC had leapt ahead and invented personal computing. The Alto was an amazing accomplishment.

Along with many other innovative features included in the Alto was one feature particularly relevant to our interests. The Alto's designers wanted their new machines to be interconnected through a network. This network would be used both to facilitate sharing information between

machines and so that the machines could easily access another remarkable device invented at Xerox PARC, the first laser printer.

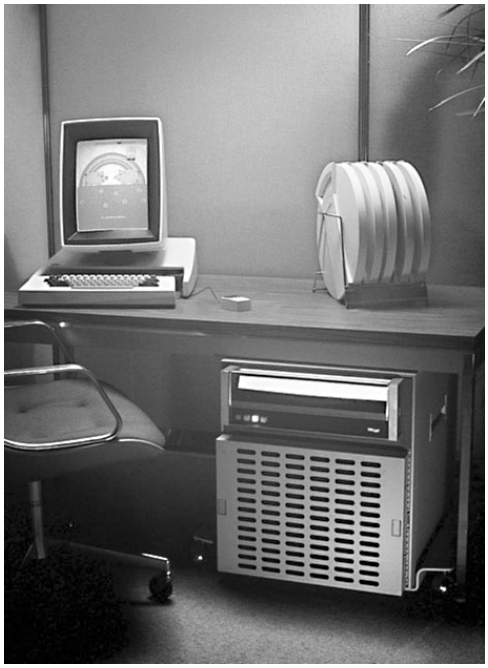

Figure 6.2: A Xerox Alto

Today, setting up such a network would be easy. You would head to a nearby computer store, buy some network interface cards, plug them in to the computers, and be done. At the time, however, there were no network interface cards or even standards for local networks in existence. The network for the Altos had to be designed and built from scratch by members of the research team at PARC.

Two members of the PARC staff were assigned to complete this project, Robert Metcalfe and David Boggs. Together, in the space of a few months, they designed and implemented a networking standard that would eventually become what we know as Ethernet. In this chapter, we will explore how Ethernet works. In addition to being one of the most important networking standards itself, the techniques employed within an Ethernet have a great deal in common with those used in wireless networks. We will also discuss the basics of how wireless networks function at the end of this chapter.

So what does all of this have to do with asking your cable company for Internet service? Although cable companies didn't know it at the time, the "cable" from which they get their name, *coaxial cable*, is a very good medium to use for transmitting data signals. Metcalfe and Boggs built their first network with "off-the-shelf" cable television taps and connectors. More importantly, they also imitated cable television systems in a more fundamental way. They broadcast their signals.

6.1 Broadcast vs. Point-to-point

In the world of television, cable networks are considered the opposite of broadcast networks. Even though most of us receive our television through a cable, we still distinguish broadcast networks like ABC, NBC, and CBS from cable networks like HBO, CNN, and ESPN. Broadcast networks transmit their signals through the air for direct reception by those television sets that still have antennas. That is, broadcast networks use the airwaves while cable networks use wires.

If we restrict our attention to wires, however, the notion of broadcast takes on a different meaning. Compare how the wires connecting your TV to the cable company are used to how the phone company uses the wires connecting your phone to its local office.[1]

[1]The convergence of communication technologies makes some of the distinctions I am hoping to draw here somewhat trickier than they might be. Talking about the wires that connect your phone to the phone company may not be compelling for those who have given up having a "land line" in favor of a cell phone. Worse yet, there is now the option of getting VOIP phone service from your cable company. With this in mind, as you read on, try to think back to a simpler time when the Simpsons was a hot new show, cable TV companies only provided television service, and phones were wired.

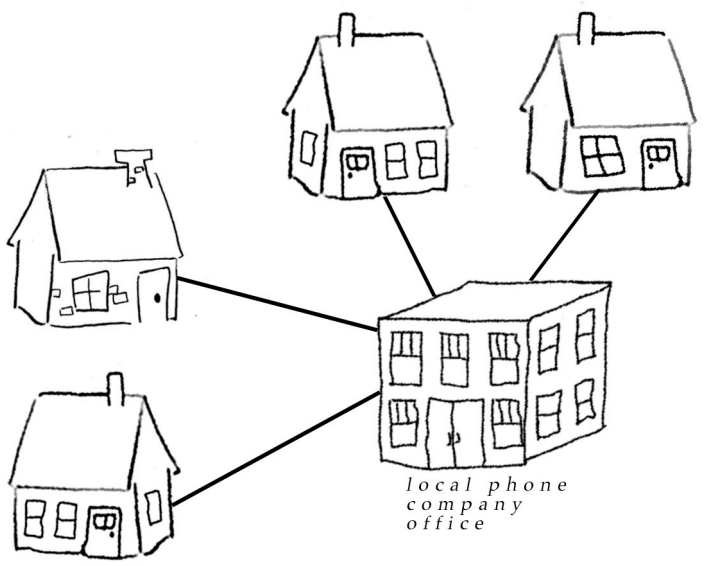

Figure 6.3: Point-to-point local lines in the phone network

The structure of the connections between a local phone company office and the customers it serves is illustrated in Figure 6.3. The local office is also connected to other phone company offices, but in this figure we only show connections between customers and the local office.

When you receive a phone call from a friend who lives nearby, the electrical signals carrying your friend's voice first travel from her home to the phone company's local office. There, the phone company's equipment arranges to send the signals through the wires connecting your home phone to the local office. Similarly, the signals carrying your voice travel only through the lines connecting your house and your friend's house to the system. In particular, the signals carrying your conversation are not sent to any other home phones. The signals are said to travel *point-to-point* from your phone to your friend's and from your friend's phone to yours.

Cable television signals are delivered quite differently from phone calls. Figure 6.4 illustrates the structure of the connections between a local cable company office and its customer's homes. In a cable system, there are not separate wires connecting each customer directly to the local office. Instead, the signal sent by the cable company travels through what is logically a single cable that is connected to all of the homes in your neighborhood. Each home's television is connected to this cable and can receive any signal sent through the cable. When you watch your favorite show on HBO, the cable company does not send the electrical signal used to carry the show to your home separately from your neighbor's. Instead, the same signal is sent through the main cable and reaches every customer's television. This signal describes not just the show you want to watch, but all of the shows available on all of the channels provided by the cable company. The tuner in your television or cable set-top box then extracts the show you want from this shared signal. The cable television signal is more like the broadcast network signal than one might have thought. Cable systems broadcast on a wire while broadcast television networks broadcast through the air.

There is a primitive communications system that we can use to provide insight into how broadcasting on a cable works without getting into the mysteries of electrical engineering. The system

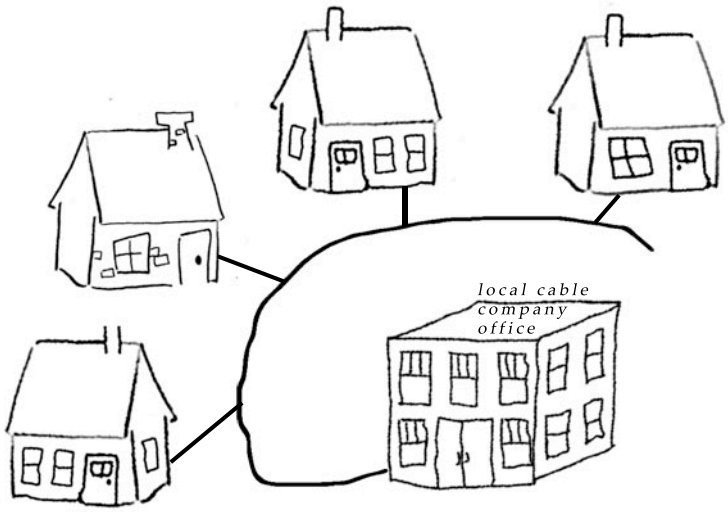

Figure 6.4: Structure of a simple cable television network

is called talking.

If you stand in a field with several other people standing around and you start shouting some important thoughts for all to hear, you are clearly broadcasting your ideas. In fact, even if there is only one other person listening to you, you are still broadcasting in the sense that what you say spreads out to fill the space around you.

On the other hand, have you ever talked to a friend through a really long cardboard tube like the ones that come with rolls of wrapping paper? This form of talking is more akin to point-to-point communications. If you have been on the listening end of a tube when someone spoke into it you have probably noticed that the speaker sounds very loud. When you speak normally, the energy in the sound waves spreads out in three dimensions. When you speak through a tube, all of the energy of the sound waves is constrained to travel in one direction. Therefore, the sound that reaches the end of the tube is much louder than it would have been if you had not been speaking through the tube.

This fact can be exploited to provide for communications over reasonable distances by installing a long tube through which two individuals can talk. Such devices are a popular feature in playgrounds. The image in Figure 6.5 shows a boy shouting into one end of such a *talking tube*. A horn like the one the boy is shouting into is placed on one end of the playground and connected through segments of tubing buried underground to a similar horn on the opposite side of the play area as suggested in Figure 6.6. Note that if the boy spoke quietly rather than shouting, the talking tube provides a form of point to point communications. The person on the far side of the tube can hear what is said, but others standing around the playground cannot.

Figure 6.5: A talking tube

Imagine that you visit a playground with a rather unusual talking tube system. Instead of having two ends, this tube has three ends interconnected as shown in Figure 6.7. As suggested in the figure, if someone talks into the horn at point A, the sound waves will travel to the point where

Figure 6.6: Sound traveling through a talking tube

the three branches interconnect and split so that the sound reaches both of the ends labeled B and C. It should not be hard to imagine a similar system of tubes interconnecting even more "ends". This is no longer point to point communications. By speaking into one end of this system of tubes, you are broadcasting. You are not broadcasting in the air so that anyone nearby can hear, but your are broadcasting to everyone with an ear positioned near one of the ends of the tubes.

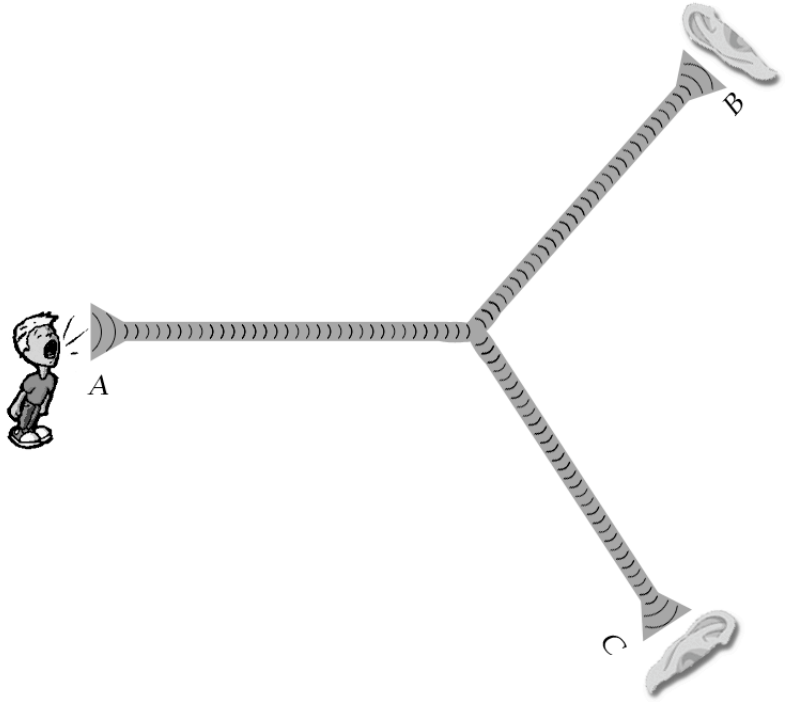

Figure 6.7: Broadcasting through a system of talking tubes

Now, reconsider the behavior of the cable system shown in Figure 6.4. The electrical signals that a cable company sends from its local office through the cable behaves much like the sound waves in the branching tubes of Figure 6.7. When the signal reaches a branch where a home is connected to the main cable it splits so that the signal reaches that home and also continues down the main cable to reach other branch points and other homes.

To fully appreciate how the ability to broadcast through a cable is used in an Ethernet, observe one more critical property of the system of tubes depicted in Figure 6.7. The diagram shows

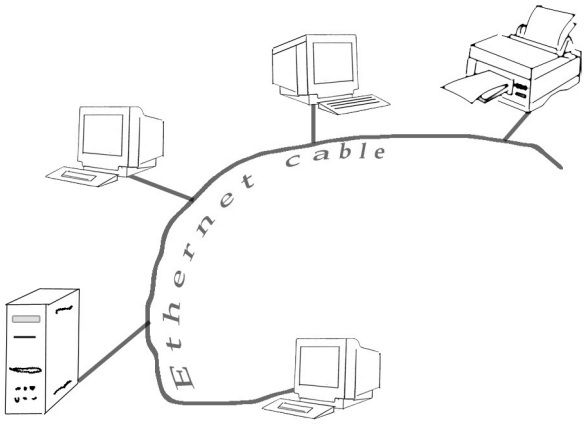

Figure 6.8: Structure of a simple Ethernet

someone broadcasting by talking into the horn labeled A. It should be clear, however, that there is nothing special about horn A. If the person at horn B stopped listening and started talking, the sound waves produced would travel from point B to both A and C. That is, we can broadcast from the end of any of the tubes in this system to all of the other ends.

The same is true when electrical signals are sent through a system of cables. In cable television systems, signals are mainly sent from the cable company office to the homes of its customers. In principle, however, a signal could originate from one of the customer homes. Such a signal would spread through the cable and be received both by all other customers and the cable company office.

The designers of the Ethernet choose to use a cable in this way. All of the devices that used the network were linked to a single cable. These devices could include desktop computers, servers, and printers. Any device connected to the cable could broadcast a signal to all of the other devices. A sketch of such a network designed to emphasize its similarity to a cable television system is shown in Figure 6.8.

There were several factors that led Metcalfe and Boggs to design their network based on broadcast rather than point-to-point communications. One issue was reliability. If Ethernet had been designed to provide point-to-point links like the telephone system, there would have to be an automatic switch somewhere in the system. Each computer would be connected to the switch by a separate cable. The switch would contain the electronic components required to determine how to direct each message to its destination. A switch is a complex device that can fail in many ways. If the switch failed for some reason, the whole system would fail. On the other hand, other than being cut, there is nothing that can really go wrong with a cable. Basically, a simple cable is more reliable than a switch.

A cable is also less expensive than a switch. The designers of the Alto were attempting to build a machine that could someday be marketed as a personal computer. As a result, they planned the machine with a tight budget. Each component had to be designed to fit within this budget. Metcalfe and Boggs were allocated 5% of the total cost of the machine to spend on its networking components. Avoiding the cost of building an automatic switch was therefore desirable.

While Metcalfe and Boggs did not want to pay for the cost of a switch, they did want some of the functionality a switch would provide. They wanted a machine to be able to communicate with one other specific machine rather than with all the machines in the network. They accomplished

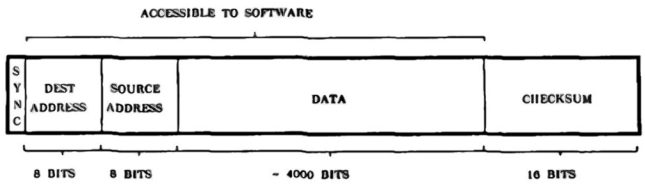

Figure 6.9: Original Ethernet packet format

Preamble	*Destination address*	*Source address*	*Length*	*Data*	*Checksum*
64 bits	*48 bits*	*48 bits*	*16 bits*	*variable* (up to 1500 bytes)	*32 bits*

Figure 6.10: Current Ethernet packet format

this by associating a numeric addresses with each machine on the network and placing the address of the intended destination of each packet at a fixed position within the packet. While packets were broadcast on the Ethernet cable, the network adapter installed in each computer checked the destination address in the packets received and ignored all packets addressed to other computers.

Figure 6.9, shows a diagram of the layout of data in packets on the original Ethernet.[2] The destination address appeared immediately after a bit that functioned as a start bit for the packet. It was followed by the address of the computer that sent the packet and the actual data being sent. The packet ended with a field used to check for errors in the packet. Figure 6.10 shows the packet format used on modern Ethernets. Most of the fields and their order is the same. The start bit has grown to be a 64 bit preamble. The addresses have grown from 8 bits to 48, and a field that can be used to store the length of the data has been added.

6.2 Broadband vs. Baseband

In the preceding section, we emphasized an important similarity shared by Ethernets, cable television networks, and broadcast television networks. In all three systems, signals are broadcast simultaneously to all receivers in the system. In this section, on the other hand, we will focus on an important difference between the way signals are transmitted on an Ethernet compared to the techniques used by both cable and broadcast television systems.

In television systems, several signals are transmitted simultaneously through a single medium. These signals are what we call channels. In most cities, several broadcast stations transmit simultaneously to all the customers in their area. The tuner within a television set extracts the desired

[2]This diagram appeared as a figure in the original paper published by Metcalfe and Boggs that described the system.

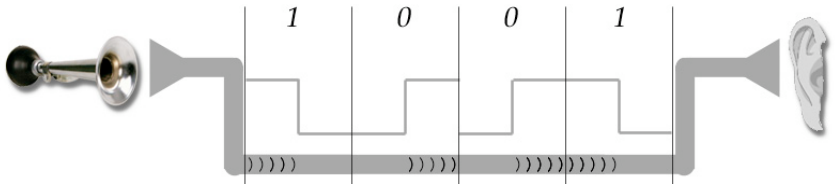

Figure 6.11: Manchester encoding using sound waves

signal from the combination of signals received depending on which channel is selected. Cable systems typically provide many more channels, all transmitted through a single cable at the same time.

In an Ethernet, by contrast, only a single signal can be transmitted at any particular time. This difference is very significant because in an active Ethernet there are likely to be several different computers that want to send messages at any particular time. Somehow, the system has to be designed so that computers will take turns sending one at a time.

This difference results from a more fundamental difference in the ways Ethernet transmitters and television systems encode signals. The approach used by Ethernet falls into a category known as *baseband transmission* while television systems use *broadband transmission* techniques. In this section, we will explore the difference between broadband and baseband transmission to explain why Ethernet requires that simultaneous transmissions be avoided. In the following section, we will discuss the techniques used to enable the devices connected to an Ethernet to take turns transmitting.

The talking tube that we used as an analogy earlier can also help us explain the difference between broadband and baseband transmission. As a starting point, think about how we could adapt techniques like on-off keying and Manchester encoding to enable us to send binary signals through a talking tube.

To refresh your memory, on-off keying and Manchester encoding depended on dividing time into fixed length periods each of which would be used to send a single binary digit. With on-off keying, the transmitter would send an electrical signal or a beam of light during a time period to send a "1" and send nothing to send a "0". With Manchester encoding, a "1" was encoded by sending nothing for half a period followed by sending light or electricity during the second half. A "0" was encoded by sending light or electricity for the first half of a period followed by nothing in the second half.

We can adapt either of these systems to a talking tube rather easily. Rather than using light or electricity, we will use sound. For example, given a nice noise maker like a bicycle horn, we could simply replace sending light or electricity by tooting our horn. Figure 6.11 suggests how this might be accomplished using Manchester encoding. This figures shows a horn on the left being used to send the binary message 1001 to a listener at the right end of the tube.

The figure is drawn to suggest that the time it would take for a sound to travel from one end of the tube to the other is equal to the time required to transmit the four bits.[3] Therefore, all of the sound waves used to transmit the bits are shown traveling through the tube at the same time.

[3]Sound travels at about 1100 feet per second. The timing suggested in the diagram could occur in reality in several configurations. For example, if one bit was sent every second, the tube would have to be 4400 feet long. In this case, a one would be sent by being quiet for half a second and then tooting the horn for half a second.

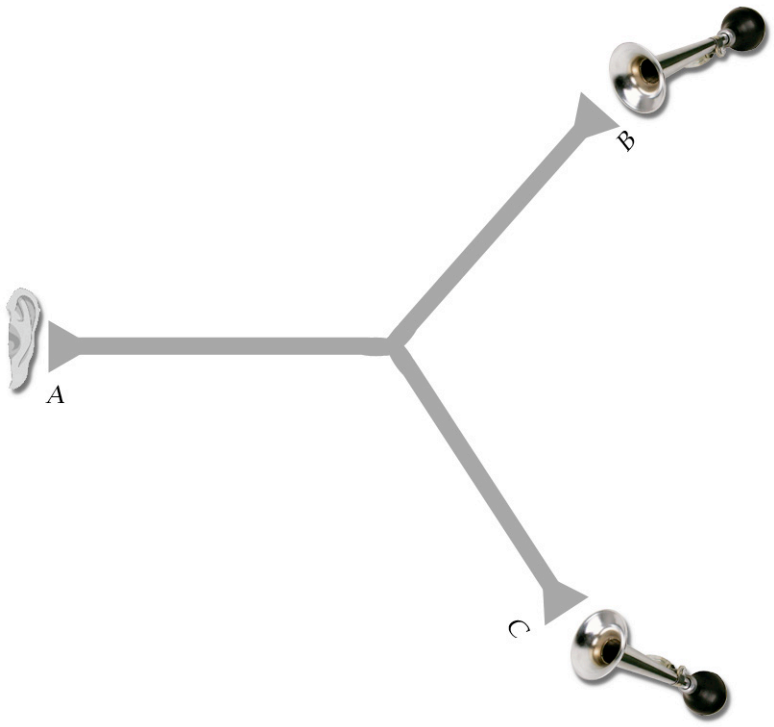

Figure 6.12: Two horns prepare to honk

Above the tube we have included thin vertical lines to delineate the boundaries between the sounds used to represent each of the four bits and shown the traditional square-wave representation of the Manchester encoding of the bits. Curved wavefronts are shown in the tube to show areas of the tube where a toot of the horn would be heard. The rightmost 1 shown in the diagram represents the first bit sent (and received). The leftmost 1 represents the last bit sent.

Shortly after the moment in time represented in the figure, the receiver would hear a short period of silence followed by two toots of the horn followed by another short period of silence, another toot, and so on. Following the rules of Manchester encoding, the sounds heard during the first time period, silence followed by a toot, would be interpreted as a 1. The toot followed by silence in the next time period would be interpreted as a 0, and so on.

This system only works well if one transmission is sent at a time. To see this, imagine what would happen if two horns were used to send signals simultaneously through a 3-ended system of tubes as suggested in Figure 6.12. It clearly would be hard for the listener at point A to simultaneously interpret signals sent from points B and C. While it would still be easy for the receiver to distinguish times when no noise was heard from those when a noise was heard, it might be difficult to separate the sound of one horn honking from that of two honking. More importantly, when only one horn was heard, there would be no way to tell whether it was horn B or horn C.

This scenario explains why it is not possible to transmit two signals on an Ethernet at the same time. This, however, is not that surprising. The more interesting question is how multiple signals can be transmitted simultaneously on a cable system. We can provide some insight into the techniques used by simply replacing our bicycle horns with more refined musical instruments.

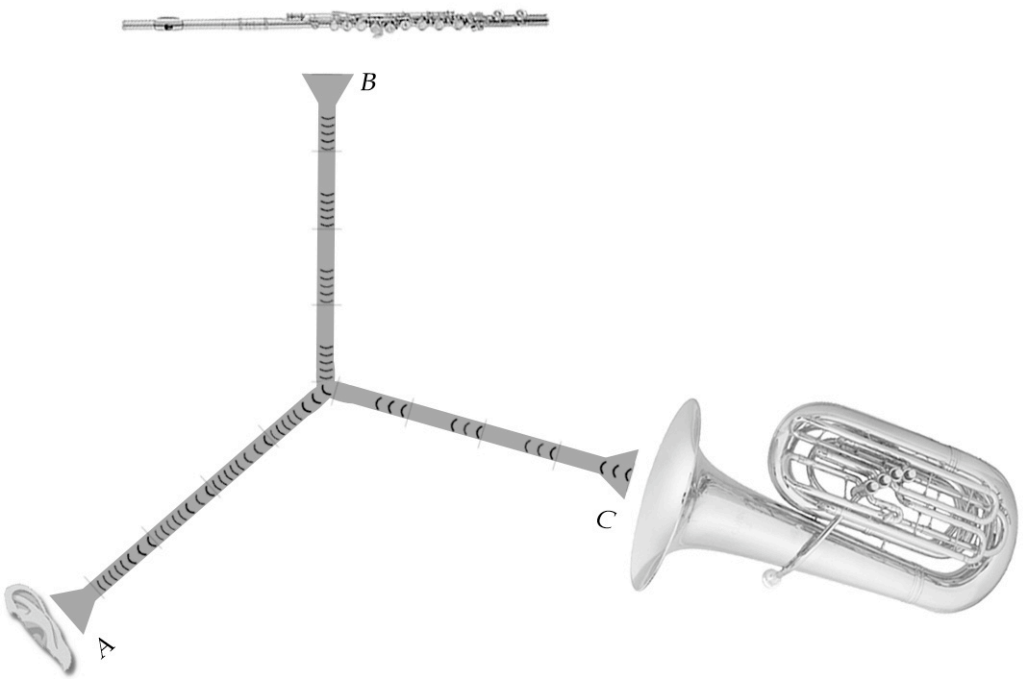

Figure 6.13: Harmonizing in binary

Suppose that instead of using two identical bicycle horns we transmit two different messages simultaneously using two very different instruments to produce sounds. For example, as suggested in Figure 6.13 we might use a tuba and a flute. Each of these instruments will be used in basically the same way we used the bicycle horn. A 1 will be sent by following a period of silence by playing a short note. A 0 will be sent by playing a note followed by a period of silence. The tones produced by the two different instruments, however, will sound very different. A human listener at point A would be able to distinguish the sounds of the tuba and the flute from one another and from the sound of the two of them being played together. This would make it possible for a listener at point A to correctly retrieve messages even if two messages were sent simultaneously.

The wavefronts shown traveling through the tubes in Figure 6.13 are designed to suggest the physical properties of the sounds that make this possible. The sounds produced by a flute and by a tuba have very different frequencies. That is, the rate at which a tuba vibrates is much slower than the rate at which the flute vibrates. To suggest this in the figure, each short note produced by the flute is represented by more wavefronts than those produced by the tuba.

When a television signal is transmitted through either a cable or the air, the signal actually takes the form of waves traveling through the medium. They are certainly not sound waves. Instead they are called *electromagnetic* waves. Just as we suggested we could transmit two binary signals simultaneously using sound waves of different frequencies, multiple radio and television signals are broadcast simultaneously using electromagnetic waves of different frequencies.

Electromagnetic waves of different frequencies behave and are perceived quite differently. At high frequencies, electromagnetic waves appear to travel in straight lines. Certain high frequencies become visible as light. At lower frequencies, electromagnetic waves spread in three dimensions

and are not visible to the eye. Such waves are used to broadcast television and radio signals. Each radio station and broadcast television station in a given area uses a different frequency. This is most obvious with radio stations. When you tune to 99.5 on your FM radio, you are selecting the station that transmits using electromagnetic waves with a frequency of 99,500,000 cycles per second. Just as a sensitive ear can distinguish the difference between a flute and a tuba, the tuners in your radio, television, and cable box are electronic devices designed to be able to extract a signal of a particular frequency from a medium in which many signals are being transmitted at the same time using different frequencies.

A system like our bicycle horns, in which data is transmission is based only on whether a signal is or is not being sent regardless of any underlying signal frequency is said to be a *baseband transmission* system. A system like radio or television in which different frequencies are used to encode different signals is called a *broadband transmission* system.

Ethernet uses baseband transmission. Given that broadband transmission is possible, the obvious question is why doesn't Ethernet use this ability. First, it is cheaper. While it would have been possible to incorporate a tuner into the circuits that connected a computer to an Ethernet, those circuits could be built for less if no tuner was required. Remember, the designers of the first Ethernet were working within a tight budget.

Another issue was ease of configuration. The broadband system used by radio stations works because each radio station is assigned a distinct frequency by a government agency. Metcalfe and Boggs wanted it to be very easy to reconfigure an Ethernet by adding or removing computers. If every computer had to be assigned a distinct frequency, configuring the system would be more complex.

Of course, the decision to use baseband transmission implied that there had to be some way to prevent stations from transmitting at the same time. This added some complexity of its own, as we will see in the next section.

6.3 Carrier-sense Multiple Access

Technical people love making up impressive names for simple things. The full name for the technique Ethernet uses to ensure that messages get transmitted one at a time is "carrier-sense multiple access with collision detection." Those in the know call it CSMA/CD to truly mystify the uninitiated. The amazing thing is that this technique with a fancy name is actually little more than an imitation of polite, human conversational behavior.

Most people consider it polite behavior not to talk at the same time as someone else. People accomplish this by following two rules. The first rule is that one should not start talking if someone else is already speaking. In the Ethernet system, this rule is called *carrier sense*. Before a computer starts sending a message on an Ethernet, it listens or "senses" to see if there is already a signal on the cable. If so, the computer waits patiently until the computer that is already sending finishes.

The second rule is that if two individuals start trying to talk at the same time, both speakers should quickly offer an "Excuse me" or an "Oh. I'm sorry." Then something magic happens. By some combination of eye contact, body language, and good luck, they figure out which one of them really is sorry and the other one starts to speak.

The rough equivalent of this behavior in Ethernet is called *collision detection*. A *collision* is just the technical term for a situation in which two computers begin to transmit at the same (or nearly the same) time. The Ethernet protocol requires that any computer that is transmitting data

listen to the signal on the wire at the same time that it is transmitting. As long as what it hears is identical to what it is sending, it keeps transmitting. If, however, the signal it hears is different from the one it is transmitting, then it assumes some other computer is transmitting at the same time. In this case, we say the machine has detected a collision.

When a computer detects a collision, it stops its transmission immediately. It does not, however, stop transmitting completely. It first sends a short transmission called a *jamming signal*. The jamming signal ensures that even if the collision was brief, all other computers will realize that a collision did occur and stop their transmissions. In some sense, the jamming signal is the opposite of saying "I'm sorry." It is more like saying "Hey! I wanted to say something." Once a computer finishes sending the jamming signal it stops transmitting. Then, the magic happens.

Computers on an Ethernet cannot depend on eye contact or body language to decide which of the computers involved in a collision will transmit next. The only means they have to communicate is the Ethernet cable itself, and they cannot use it to figure out who can use it. Accordingly, Ethernet uses a much simpler approach. After a collision, a computer waits until the network is again idle and then picks a random amount of time to wait before trying again. During the waiting time, the computer does nothing. Once the wait is over, the computer behaves like it just got the urge to send its message for the first time. That is, it again does carrier sense to see if anyone else is sending, etc.

Even though the computer is expected to pick a "random" wait time, there are rules it must follow when making this random choice. We will discuss these rules later. For now, the main point is to recognize that randomness is necessary. If both computers involved in a collision followed the same non-random algorithm to decide how long to wait after a collision, they would both pick the same waiting time and collide again.

Of course, if two computers pick random waiting times independently, they may pick the same waiting time and collide again. Such situations are handled by the mechanisms already described. The two computers simply detect the second collision as they detected the first and pick new random waiting times. With a bit of luck, after just a few tries, one computer will pick a shorter time than the other and transmit successfully. The entire process is called *collision resolution*.

Note that the random waiting process is only used immediately after a collision and not in conjunction with carrier sense. If a computer discovers that another computer is already transmitting before it begins to transmit, it does not wait randomly. Instead, it continues to sense the cable until it detects the end of the packet that is being sent. Then, it begins to send its own packet. This behavior is called *persistence* or *1-persistence*. It should be noted that it may lead to collisions. It is fairly uncommon for two computers to start transmitting at almost the same time on an idle network. It is more likely that two computers will try to start to transmit during the time some third computer is sending a long message. In fact, there might even be more than two computers waiting by the end of a very long transmission. In this case, all of the computers will wait for the message in progress to end. Once it does, they will all start to transmit as soon as they detect that the network is idle and all collide with one another. This collision will then have to be resolved using the process described above.

6.3.1 When Packets Collide

To correctly understand the functioning of an Ethernet, one has to look carefully at how packet collisions actually occur. There are two physical properties of electromagnetic signals that make this important.

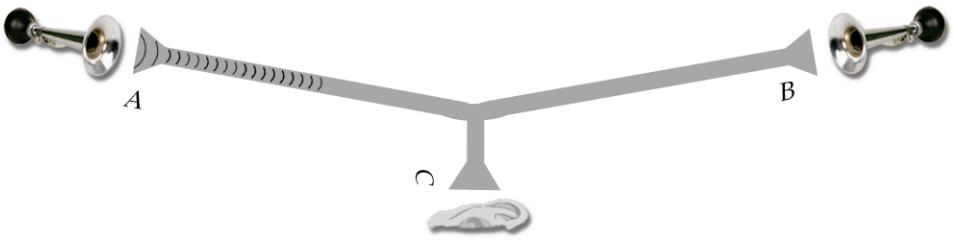

Figure 6.14: The first step toward a packet collision

- They take time to travel through a cable, and
- when two signals pass through a section of cable at the same time, they interfere, but don't damage or destroy one another.

As a result, we will see that computers on an Ethernet will detect both the beginnings and ends of collisions at different times. Since the computers stop sending when they detect a collision and can only start again after they detect that the network is idle, the timing of these events is critical.

Most of us have a hard time appreciating just how much can happen in a microsecond.[4] Electrical signals travel through a cable at about 200,000,000 meters per second.[5] Thus, if one computer is connected to another through a cable that is 200 meters long, it will take

$$\frac{200 \text{ m}}{200,000,000 \text{ m/sec}} = 10^{-6} \text{ seconds} = 1 \text{ microsecond}$$

for a signal to travel from one computer to another. That doesn't seem like very long, but many Ethernets transmit data at a rate of 100 million bits per second. That is, in one microsecond, the time it takes a signal to travel 200 meters, a computer can easily transmit 100 binary digits.

To work our way through the steps that would occur when packets collide in an Ethernet in a way that avoids microseconds or other unfamiliar time units, we will return to our talking tube analogy. Figure 6.14 shows a simple network of talking tubes that we will use to approximate the behavior of an Ethernet. The tube network has three ends corresponding to an Ethernet with three computers attached. We have placed images of horns at ends A and B, the network connection points from which the colliding signals will originate. We will assume that the messages being sent from A and B are both destined for the listener at point C and that A and B will follow the Ethernet rules for carrier sense and collision detection when sending messages.

Recall that sound travels about 1100 feet per second. With this in mind, we will assume that the distance from A to B in our tube network is 4200 feet so that the total time it takes for a signal to travel from A to B is just under 4 seconds. For example, the diagram in Figure 6.14 attempts to show the network 1 second after the transmitter at point A begins to send a message. The train of arcs used to indicate the presence of a signal in the tube extends a little more than one quarter of the way from A to B.

[4] A microsecond is one millionth of a second and a millisecond is one thousandth of a second.

[5] The "speed of light" discussed in most physics classes is the speed of electromagnetic waves in a vacuum, 300,000,000 meters per second. When traveling through air, glass, coaxial cable, or any other medium, these waves travel at a slightly slower speed that depends on the medium. Electrical signals passing through the cables typically used in Ethernets actually travel at about 200,000,000 meters per second.

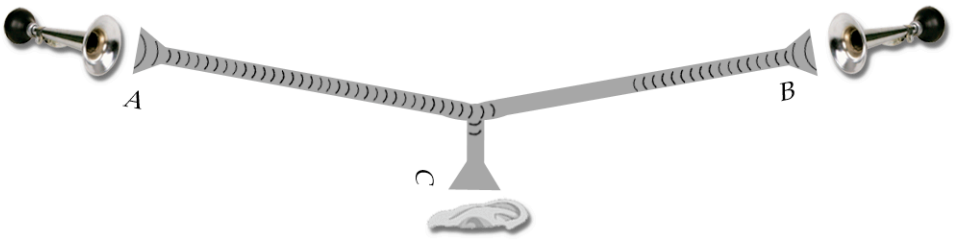

Figure 6.15: Signal propagation after 2 seconds

Suppose that B gets the urge to send a message at the moment depicted in Figure 6.14. The carrier sense rule says that B cannot start to send this message if it can sense that any other message is already being transmitted. Even though A has been transmitting for a second, we can see that its message has not reached B. Therefore, B will mistakenly conclude that the network is idle and begin transmitting a message of its own.

Figure 6.15 represents the state of the network after two seconds. The beginning of the message from A has travelled a bit more than half way from A to B and has almost reached C. The message from B has travelled one quarter of the way to A. Even though two message are traveling through the network simultaneously, no collision has occurred or been detected. Sometime soon, however, a collision between the messages from A and B will occur.

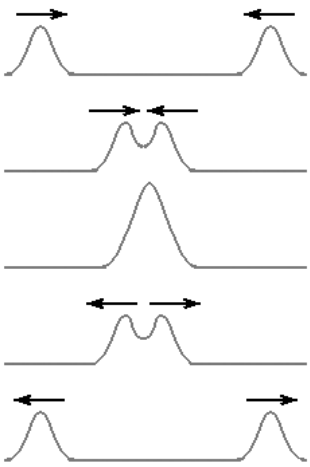

Figure 6.16: Wave superposition

The use of the word collision suggests two packets barreling down a highway at high speed, the sudden squealing of brakes, skid marks, twisted metal, broken glass, etc. The reality of packet collisions is very different.

As we have suggested, the electrical signals used to transmit data through network wires travel much like sound waves travel through a pipe. Waves of all sorts travel through a medium by temporarily changing the local, physical properties of the medium. As a sound wave passes by, the air pressure at a given point might rise slightly and then fall slightly. As a wave passes through water, the height of the water at a given point rises and then falls.

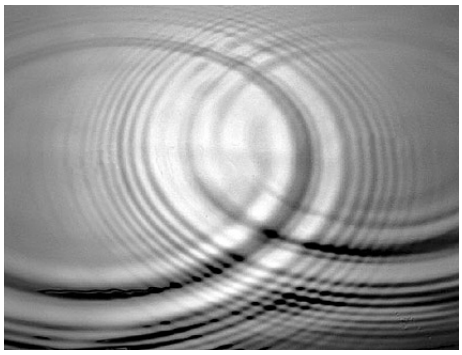

Figure 6.17: Colliding waves

When two waves pass one another at some point in a medium, their effects on the medium at that point are combined. This process is called *superposition*. Figure 6.16 show how superposition may alter the shapes of two waves as they pass the same point. Their temporary combination, however, does not effect the way in which either of the waves continues as it propagates through the medium. Two waves can pass through one another and then continue on their way undamaged by the experience!

Figure 6.17 illustrates a common example of this behavior. If you drop two pebbles into a body of still water,

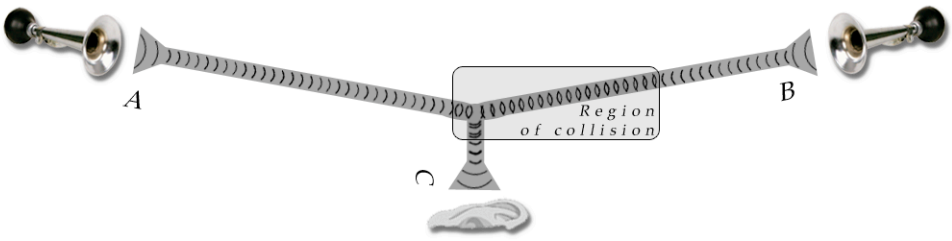

Figure 6.18: Signal propagation after 3 seconds

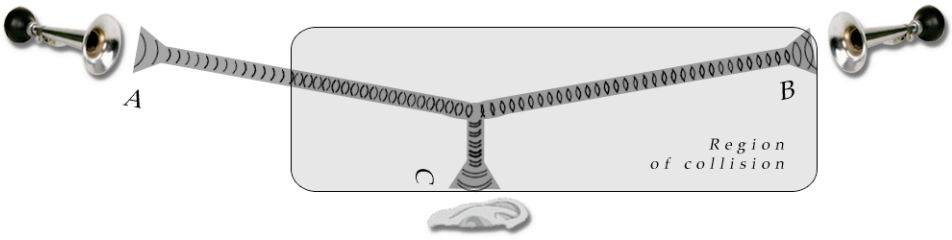

Figure 6.19: Signal propagation after 4 seconds

the rings of waves that spread out from the spots where the pebbles hit the water form interesting patterns while they overlap, but continue to spread as simple rings after they pass through one another.

Given this behavior, in Figure 6.18 we show two overlapping sets of wave fronts in the regions of our network where signals from both A and B would be simultaneously present 3 seconds after A's first transmission.

There is a proverbial question about whether a tree that falls in a forest makes any noise if no one is there to hear it. In our case, the corresponding question is whether there really is a collision on a network if no one hears it. As we can see in Figure 6.18, although their packets are in some sense colliding, neither A, B, or C can determine this by listening to the signal on the network at the point where they are connected. Accordingly, neither sender will detect the collision at this point. They will both continue transmitting as if no other signal was present.

Of course, the day of reckoning is not far away. A's signal is approaching B rapidly and B's message is very close to C. In fact, by the time 4 seconds elapse, B will begin to receive the signal from A and C will begin to receive B's signal as shown in Figure 6.19. At this point, B will stop transmitting its message, transmit a brief jamming signal, and then stop transmitting altogether. C will realize it is no longer accurately receiving bits from A. A, on the other hand, will still be oblivious to the fact that a collision has occurred. Since B started sending after A, B's message will not yet have reached A.

A will finally realize that its message has encountered interference a little less than 5 seconds after it started to transmit. The state of the network at this point is illustrated by Figure 6.20. As B did earlier, A will now stop its own transmission, send a jamming signal, and then stop all transmission.

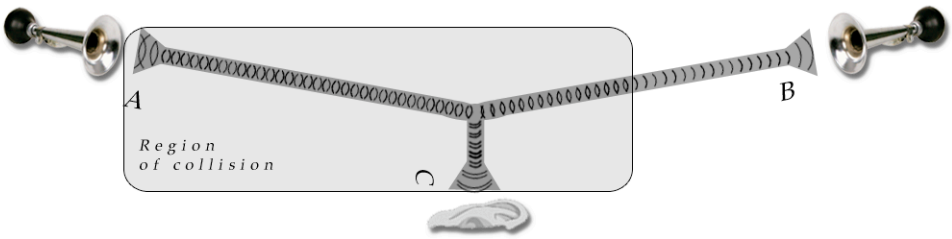

Figure 6.20: Signal propagation after 5 seconds

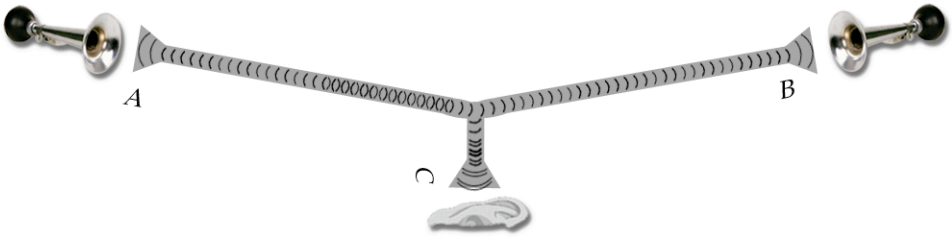

Figure 6.21: Signal propagation after 6 seconds

Note that in this figure, the region of the network in which two signal are simultaneously present has moved but not grown. There are no longer two signals present on the rightmost half of the tube that leads to B. Because B stopped transmitting when it detected A's signal, the last waves of B's signal have moved down the network toward A. At this point, all B will sense on its end of the network is A's signal.

The signals sent by A and B before they each detected the collision will continue to propagate through the network until they reach its ends. This process is shown in Figures 6.21, 6.22, and 6.23. After 6 seconds, all of B's signal will have moved out of the leg of the network connected to B. Its remnants will fill most of the legs to A and C. By 7 seconds after the beginning or A's transmission, only the left half of the segment of the network connecting to point A will contain portions of B's transmission. The remnants of A's transmission will occupy much of the segments leading to C and B.

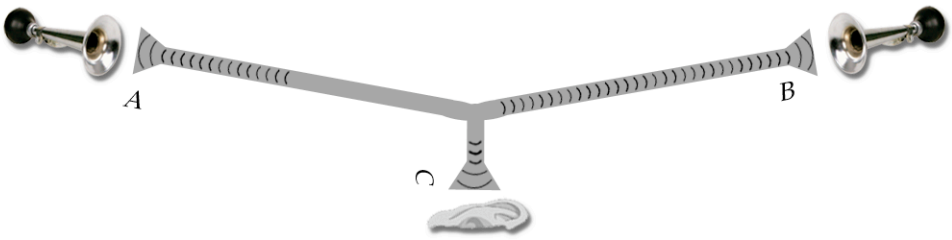

Figure 6.22: Signal propagation after 7 seconds

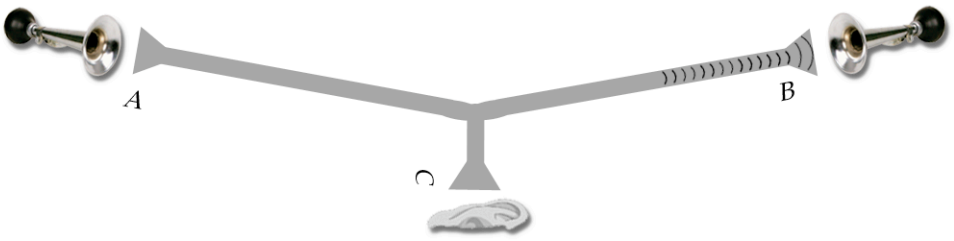

Figure 6.23: Signal propagation after 8 seconds

The state of the network shown in Figure 6.23 is interesting. At this point, if A or C sense the network they will conclude that it is idle while if B senses the network it will appear to still be in use. In our description of Ethernet collision detection, we stated that after a computer detects a collision it must wait for the network to become idle and then pick a random waiting time before attempting to transmit again. What Figure 6.23 shows us is that after a collision, one computer may detect an idle network and begin its random wait before another. Within another second, B will also sense that the network is idle (even if A has actually already started another transmission!) and begin its own random waiting period.

6.4 Not Too Long, ...

We claimed that the techniques used to avoid collisions in an Ethernet were "little more than an imitation of polite, human conversational behavior." Our description of two tooting horns, however, may not sound very much like a description of human conversational behavior. The basic rule illustrated by the tooting horns is a rule followed by humans. If two people notice that they are talking at the same time, they both stop and try again later. In human conversations, however, both speakers seem to notice the conflict instantly and simultaneously. In our tooting horn example, horn B notices the conflict 2 seconds before horn A and it takes a total of 5 seconds before both horns detect the collision.

In most human conversations, the sound waves that leave our mouths travel less than 10 feet before reaching an attentive ear. In our horns example, on the other hand, we assumed the sound waves had to travel 4200 feet. If we used our horns to communicate through a tube that was only 10 feet long, collisions would be detected much more quickly. It only takes sound 9 milliseconds to travel 10 feet. If A tooted its horn, a collision could only occur if B decided to toot before the sound from A traveled 10 feet to reach B. Therefore, B would have to toot less than 9 milliseconds after A. B's toot would reach A 9 milliseconds later. Within 18 milliseconds, both A and B would have detected the collision. That is less than 1/50th of a second. To most of us, that is instantaneous!

On the other hand, try to imagine holding a conversation through a tube that was 4200 feet long. An individual at one end of the tube could easily be talking for 5 seconds before realizing that the individual at the other end had started to talk at nearly the same time. It is quite easy to speak 10 to 15 words in 5 seconds. If you tried to hold a conversation through such a long tube you could easily have completed an entire sentence before realizing the person you were talking to was talking at the same time.

Obviously, the details of the way communications occur in such a system depend on the amount

of time it takes signals to travel through the medium of communications. With this in mind, consider the propagation of signals on an Ethernet. A typical Ethernet connects computers spread throughout a building or several buildings located relatively close to one another. The size of the network is therefore more similar to our 4200 foot tube than to the 10 feet sounds might travel in a typical conversation. To make the numbers work out nicely in our computations, we will consider an Ethernet that spans 1000 meters.

The electrical signals that travel through an Ethernet propagate at a speed of about 2×10^8 meters per second. Therefore, the total time it would take a signal to travel from a computer at one end of a 1000 meter Ethernet to a computer at the opposite end is

$$\frac{1000 \text{ meters}}{2 \times 10^8 \text{ meters/second}} = 5 \times 10^6 \text{ seconds} = 5 \text{ microseconds}$$

This is even less time than it takes for sound waves to travel between speakers in a typical conversation. By the standards we apply in human interactions, collision detection on an Ethernet would be considered instantaneous. In an Ethernet, however, the standards we use in human interactions don't apply.

To appreciate how long 5 microseconds is on an Ethernet, we have to consider how many "words" a computer on an Ethernet can transmit in 5 microseconds. Computers on the Ethernets found in most homes, offices and dormitories transmit between 10 million and 100 million binary digits in a single second. Thus, a computer on a 1000 meter Ethernet might transmit between 50 and 500 binary digits in the 5 microseconds it takes its signal to travel from one end of the network to the other. On an Ethernet, 5 microseconds is far from instantaneous!

Collisions on an Ethernet are a source of inefficiency. While two computers are colliding, the network's capacity is being wasted. From the time that the first of the colliding computers begins its transmission to the point where they have both detected the collision and stopped their transmissions, no useful work can be done on the network. The designers of the Ethernet were aware of this and therefore wanted to minimize the amount of time occupied by collisions.

One way to limit the time wasted by a collision is to limit the time that can elapse between when the first computer involved starts to transmit and when all computers involved have detected the collision. This time is related to the length of the network. In our example, our assumption that the network was 1000 meters long implied at most 5 microseconds would elapse between the point when the second computer began to transmit and the original computer detected the collision. If we had assumed the network was 2000 meters long, then it would be possible for 10 microseconds to elapse before the collision was detected, doubling the amount of time wasted. In general, the longer an Ethernet, the more time could be wasted through collisions.

With this in mind, the designers of the Ethernet placed an upper limit on the total length of the network cables. Actually, it is a bit more complicated than this. The key is not the length of the cables but the amount of time it takes a signal to travel from one end of the network to the other. This is called the *propagation delay*. Furthermore, it isn't really how many seconds or microseconds that matter. It is how many bits can be transmitted before a collision is detected that actually distinguishes a short time from a long time. Therefore, the Ethernet protocol specification places limits on the size of the network that guarantee that the total time required for a signal to travel from one end of the network to the other cannot exceed the time required to transmit 256 binary digits.[6] On an Ethernet that transmits at 10 megabits per second, this corresponds to a signal propagation time of 25.6 microseconds.

[6]In reality, the limit on the propagation delay does not translate simply into a limit on the length of the Ethernet

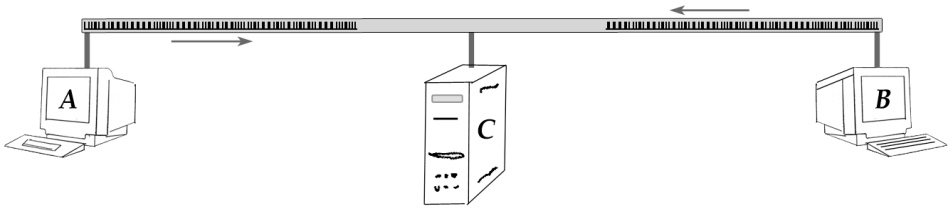

Figure 6.24: Two computers start short, simultaneous transmissions

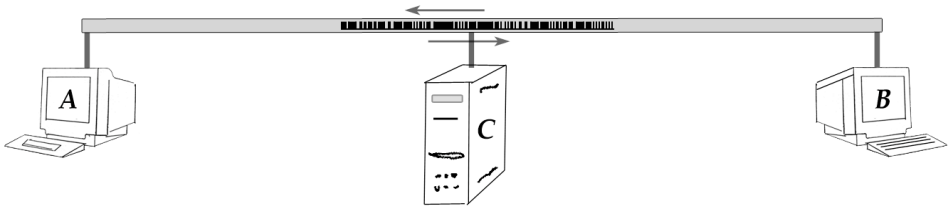

Figure 6.25: Signals from A and B collide as they pass their destination

6.5 And Not Too Short ...

In addition to placing an upper bound on how much time can be wasted by a single collision, the limit on the propagation delay also provides a simple means to guarantee that all collisions are detected. To appreciate the need for this mechanism, imagine that two computers simultaneously transmit very short messages on a long network. In particular, suppose that in the network shown in Figure 6.24, it takes 256 bit transmission times for a signal to travel from A to B and that both A and B decide to send messages containing 12 bytes (i.e., 96 bits) to computer C at the same time. In that case, as shown in the figure, each of the signals will have traveled less than half way to the other end of the network in the time it takes to transmit all 96 bits. As a result, both A and B will be finished transmitting their messages before any collision has occurred or been detected.

Unfortunately, as these messages pass the section of the network to which C is attached, they will collide as shown in Figure 6.24. Both signals will be passing by C at the same time, so C will be unable to interpret either of the signals. Both attempts to send messages to C will therefore be unsuccessful.

Worse yet, B will have stopped transmitting long before A's signal reaches B. Therefore, B will have no way to realize that the signal it is receiving from A actually collided with its earlier transmission. Similarly, as shown in Figure 6.26, B's signal will arrive at A intact. The cause of this problem is simple. Computers on an Ethernet only check for collisions while they are transmitting. In this example, A and B did not transmit long enough to detect the collision of their messages.

The Ethernet protocol avoids such situations by simply forbidding short messages. The protocol insists that all messages be long enough that the time required to transmit them will exceed the

cable. In addition to network cables, an Ethernet can include repeaters that interconnect two cables and amplify signals as they travel from one cable to another. These repeaters introduce additional delay in the movement of signals from one end of the cable to the other. Therefore, the Ethernet protocol specification includes limits on both the length of the cables used and the number of repeaters between any two computers on the network.

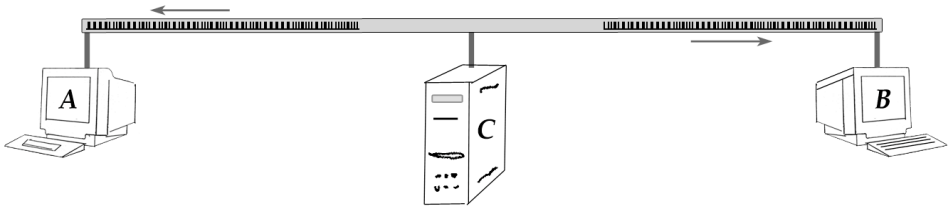

Figure 6.26: Signals reach A and B undamaged

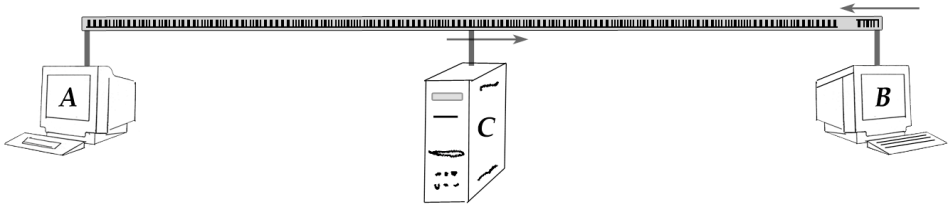

Figure 6.27: Signal from A just before reaching B

time required to detect a collision in the worst case where two computers start their transmission as far apart as possible while still colliding. This occurs when the second computer starts its transmission just before the transmission from the first computer reaches it. Figure 6.27 illustrates how this can happen.

The figure shows a signal traveling from A to B. The signal has not yet quite reached B. Once the signal reaches B, a collision would become impossible. B has to sense the network before beginning its transmission and would delay its transmission if A's signal had reached it. In the situation shown in the figure, however, B has just started its own transmission. If the time required for a signal to travel from one end of the network to the other is equal to the time required to send 256 bits, then the situation shown in the figure can only occur at some point at which A has not quite completed sending 256 bits. It might have transmitted 255 bits or even 255 1/2, but not 256.

If B starts to transmit when the network is in the state shown in Figure 6.27, B will detect a collision very quickly, send a jamming signal, and abort. A, however, will be unaware that a collision has occurred until B's brief transmission and its jamming signal travel down the network to reach A. This can take any amount less than 256 bit times. Therefore, in the worse case the time between when A starts to transmit and when A detects a collision can be as large as twice the time it takes a signal to travel from one end of the network to the other. That is, it may take up to 512 bit transmission times for A to detect the collision.

For this reason, the Ethernet standard requires that all messages sent on the network contain at least 512 bits. This includes not just the data in the message, but also fields like the source address, destination address, and error check. It does not, however, include the preamble. A computer that wants to send a message shorter than this has to pad the message with blanks, zeroes, or other data that will be ignored by the receiver.

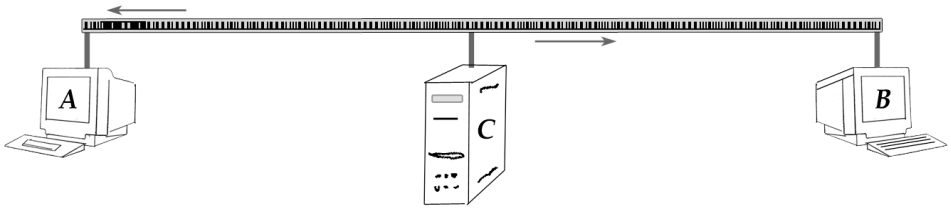

Figure 6.28: Colliding signal from B just about to reach A

6.6 The Slot Machine

Now that you know everything you need to know about how collisions occur, it is time to talk about how the computers on an Ethernet react when a collision is detected. We have already explained the basics. All computers that detect a collision independently choose random waiting times and attempt to transmit again after the waiting times selected have elapsed. What we have not discussed is the details of the distributions of these random waiting times.

There are two characteristics of the waiting times we need to discuss: their range and their granularity. The need for a limit on the range is fairly obvious. We do not want a computer to randomly choose to wait 3 years before retrying its transmission. The waiting times chosen by computers should range between 0 and some reasonably small upper limit. We will discuss how this upper limit is set in Section 6.10.

By granularity, we refer to how finely the range of waiting times should be subdivided. When you ask someone to pick a number between 1 and 10, you are probably expecting an answer like 4 or 9 rather than 4.58239. However, unless you say "Pick an integer between 1 and 10," 4.58239 is technically a valid response. By adding "integer" you are specifying the granularity of the answer you are hoping for.

To understand why the granularity of waiting times matters, recall that two messages can collide as long as the second transmission is started before the first transmission's signal has reached the second computer. As a result, if two computers choose waiting times that are too similar, their next attempts may collide. The goal in setting the granularity is to make sure that if two computers choose different waiting times they will be different enough to avoid another collision.

While we normally use units like hours, minutes, seconds, and microseconds to measure amounts of time, a different approach is often more appropriate for measuring time in a network. When we need to describe an amount of time, we can describe the number of bits that could be transmitted in that time. For example, we could say that Figure 6.27 depicts the state of the network about 250 bit times after A started its transmission. If this seems odd at first, just remember that astronomers measure distances in years (i.e., light years)!

To understand how different waiting times should be, we will explore what might happen after the collision that is about to occur in Figure 6.27. Recall that we assumed the network shown in the figure was of the maximal length allowed by the Ethernet protocol. That is, we assumed the time required for a signal to travel from A to B was 256 bit times. In the figure, B is just about to detect A's transmission. This would happen 256 bit times after A started. By this time, B would have been transmitting for about 10 bit times. B would add a brief jamming signal and then stop transmitting.

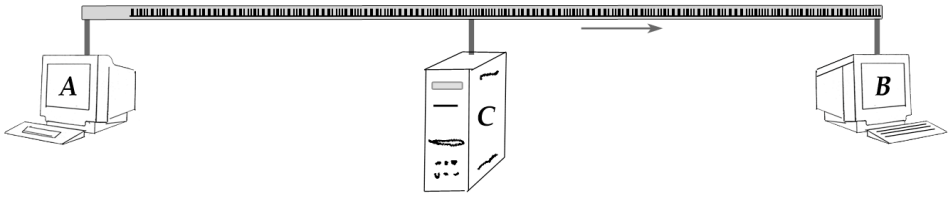

Figure 6.29: Moments after B's signal disappears from the network

Figure 6.28 shows the network about 249 bit times later. B's signal (shown as a short dark area just to the right of A in the figure) is just about to reach A. Because it has not yet reached A, however, A is unaware of the collision at this point and is therefore still transmitting.

The next few instants are critical. Figure 6.29 shows the network a little more than 256 bit times after A begins transmitting. When B's signal reaches A, A stops its own transmission. By the point shown in the figure, A has stopped its transmission and the signal from B has left the network. When A senses the network at this point, the network appears to be idle. At this point, A will start the process of trying again by choosing a random amount of time to wait.

B, on the other hand, still thinks that the network is busy. The network will not appear idle at B's end until the last bit sent by A has time to travel through the network past B. Therefore, B will realize that the network is idle about 256 bit times after A.

Recall, that we would like to ensure that if A and B choose different random waiting times, they will not collide again. Suppose, however, that A chooses to wait 356 bit times and B chooses to wait 100 bit times. Even though the waiting times they choose are different, they will actually start to transmit at nearly the same time because A will start its 356 bit time wait when it first detects an idle network, about 256 bit times before B starts its 100 bit time wait.

The worst case is when B chooses to wait W bit times and A chooses to wait just less than $W + 512$ bit times. Because of the difference between the times at which they start their waits, this will actually result in A starting just a little less than 256 bit times after B. This means A will start just before B's signal reaches it producing a collision scenario identical to the previous case except that A and B have reversed roles.

To avoid such scenarios, the random waiting times that a computer on an Ethernet can choose are limited to be multiples of the time required to send 512 bits. A computer can either not wait at all, wait 512 bit times, wait $2 \times 512 = 1024$ bit times, wait $3 \times 512 = 1536$ bit times, or wait any other multiple of 512 bit times up to an upper limit we will discuss later.

The time required to send 512 bits is clearly rather important in the Ethernet protocol. It determines the size of the smallest packet, the size of a largest network, and the spacing of the waiting periods used after a collision. It is called the *slot time*. This name reflects the fact that it is possible to think of the time immediately after a collision on the network as a sequence of 512 bit time long slots in which a computer randomly chooses to make its next transmission attempt.

It is important to note that the use of the slot time represents an effort to simplify the implementation of the network by assuming the worst case. Most real networks are shorter than the maximum. Such networks would function correctly if computers sent packets that were shorter than 512 bits or used shorter waiting times. To do this safely and efficiently, however, the computers on an Ethernet would have to determine the actual size of the network to which they were connected.

By having the computers use 512 bit times as both the minimal packet size and the slot time, the Ethernet protocol makes it unnecessary for a computer to know the actual size of the network to which it is connected.

6.7 Playing the Slots

The other constraint on the way a computer picks its random delay after a collision is the range of waiting times used. As we mentioned in the last section, it is clearly necessary to limit this range. If a machine chose to wait for 10 minutes before retrying after a collision, the user of that machine would get very frustrated.

On the other hand, if the range of delay times is too short, it may take a long time to resolve a collision. Suppose only two delay times were allowed: 0 delay or a 1 slot delay. Also, assume that 5 computers were trying to transmit simultaneously. To see that multiple collisions would be extremely likely in this situation, we can calculate exactly how likely a second collision would be.

If only two delay times are allowed, the process a computer performs is equivalent to flipping a coin. "Heads means 0 delay, tails means 1 slot delay." Given that five computers are making this choice simultaneously, we can describe the choices made by listing the number of slots chosen by each machine. For example, 0, 1, 1, 0, 0 would mean that the first computer choose to wait 0 slots, the second and third waited 1 slot, and the remaining computers waited 0 slots. Thus, every set of choices corresponds to a five digit binary number and every five digit number corresponds to a set of choices. We know that there are $32(= 2^5)$ 5-digit binary sequences, so there are 32 possible ways the five computers may behave while choosing their delay times.

In this scenario, there is really only one way the computers can avoid a second collision. If one computer chooses to send with 0 delay and the four other computers delay for one slot, the signal from the computer using 0 delay will reach all of the other computers before their delay elapses. Carrier sense will therefore ensure that no other computer begins a transmission that would collide with the first computer's message. The transmission would therefore be successful. There are five sets of delay choices in which this can happen, one for each of the five computers. Thus, the probability that some computer will be able to begin a successful transmission in the slot time immediately after the initial collision is $\frac{5}{32} = 0.15625$.

All other combinations of waiting times will require additional time to resolve the collision. There are 27 such combinations, so the probability that additional time will be required is $\frac{27}{32} = 0.84375$. Note that $0.84375 + 0.15625 = 1$. It must always be the case that the probabilities of all possible outcomes equal 1. This leads to the general rule

$$Prob(\text{all outcomes other than A }) = 1 - Prob(\text{outcome A})$$

There are two distinct ways that the computers involved may fail to resolve the initial collision immediately. If they are very unlucky, they may all choose to delay for 1 slot time. In this case, the first slot will be completely unused and a five-way collision will occur at the start of the second slot after the initial collision. There is only one way this can happen, corresponding to the sequence of delays 1, 1, 1, 1, 1, so the probability of this outcome is $\frac{1}{32} = 0.03125$. A much more likely outcome is that more that one computer chooses a delay of 0 while at least one computer waits 1 slot time. This also leads to a second collision during the first slot after the initial collision, but there are many ways that the collision resolution process can unfold after this second collision.

If a second collision occurs among several computers that chose 0 as their initial delay, the computers that were involved in the collision will try again by choosing new random delays. At the

same time, any computers that chose to delay 1 slot after the initial collision will be unaware that a second collision has occurred. When a computer delays for its randomly selected wait time, it really does nothing. It does not sense the network until the delay time expires and therefore remains unaware of transmissions and/or collisions that occur completely within the delay time. Therefore, any computer that chooses to delay one slot after the initial collision may end up competing with computers that are in the process of trying to transmit again after choosing a delay of 0 and experiencing a secondary collision after the initial collision.

To say much more about the interactions that might occur between such computers we need to make some simplifying assumptions. In particular, we will assume that the computers on the network we are discussing are all separated from one another by distances close to the maximum allowed. This means that the time that will elapse before the computers involved in a collision detect that the network is again idle will be approximately equal to the length of a slot.

We make this assumption simply because there would be too many possibilities to consider otherwise. The computers on an actual network may be much closer together than the maximum separation allowed by the Ethernet standards. In this case, a collision may be detected and all involved transmissions terminated long before the end of a slot time. As a result, if several computers choose a delay of 0 after an initial collision, they may collide again and choose new delays quickly enough that any computer that picks 0 as its second delay will transmit well before a computer that picked 1 slot as its delay time after the initial collision. On the other hand, if the computers involved are separated by distances close to the maximum allowed, the time occupied by the second collision will correspond closely to 1 slot delay.

The assumption we have made allows us to avoid these complexities. It enables us to assume that the "slots" associated with all collisions will be synchronized. This assumption is not likely to accurately describe many real networks. It is only an approximation to real networks. Using such approximations is common in scientific analyses. Our assumption that all slots are synchronized is similar to the practice of ignoring friction when solving mechanics problem in Physics or treating air as an ideal gas.

Given the assumption that all slot times will be synchronized, we can further subdivide the possible outcomes when more than one computer picks a delay of 0 after the initial collision. Any computer that does not choose a delay of 0 must choose a delay of 1. If there is more than one such computer, there will definitely be a collision during the second slot after the initial collision. Therefore, the only way that there can be a collision in the first slot followed by a successful transmission in the second slot is if all but one computer chooses a delay of 0 initially, and then after the 4 computers that picked 0 as their initial delay collide, they all pick 1 as their second delay.

As you might imagine, this is a fairly unlikely event. Just as there are 5 ways that exactly one computer chooses to transmit in the first slot, there are 5 ways that exactly one computer can choose to wait until the second slot. Therefore, the probability that 4 computers collide again immediately after the initial collision is $\frac{5}{32}$. These 4 computers now have $2^4 = 16$ possible ways to choose their next delay times. Only if all of them choose to delay 1 slot time will it be possible for the fifth computer to transmit successfully when its wait time expires. The four computers will choose such delays with probability $\frac{1}{16}$.

Now we need to apply another basic rule that applies to the probabilities of events. The probability that two events both occur is just the product of their separate probabilities as long as neither event affects the outcome of the other. That is, if A and B are events that occur

independently,

$$Prob(\text{both A and B occur }) = Prob(\text{A occurs }) \times Prob(\text{B occurs })$$

If four computer collide in the first slot, their second delay slot choices are made independently of the initial delay choices. Therefore, the probability that exactly 4 computers collide in the first slot and that these four computer then all choose to delay 1 slot time is $\frac{5}{32} \times \frac{1}{16} = \frac{5}{512} = 0.009765625$. This is the only way one of the five computers can start a successful transmission in the second slot after the initial collision. The total probability of a successful transmission within either of the first two slot times after the initial collision is therefore $0.15625 + 0.009765625 = 0.166015625$. This is about one chance out of 6. Not very good odds!

6.8 Role of the dice

The preceding analysis suggests that more than 2 slots would usually be needed to resolve a collision in such a scenario. Extending our approach to calculating the probability of success in the first two slots to the third slot, however, would be difficult. The outcome in the third slot would depend on whether collisions occurred in the first slot, the second slot, or both and on how many computer were involved in each collision. Extending this approach to later slots would be a daunting undertaking.

Worse yet, the analysis we have performed has been simplified by the assumption that only two possible delay choices were possible. What we would really like to do is see how the chances of resolving a collision change as we allow computers to choose delays from other ranges. Clearly 2 choices seems to be inefficient, and in the introduction to this section we pointed up that allowing very large numbers of delay possibilities would also waste time. What we would really like to do is find the happy medium between these extremes.

In addition, we don't want a result that only applies when exactly 5 computers are competing to transmit. We would like to explore the general case where N computer have collided and are trying to resolve the collision and each of these computers is allowed to choose between Q different possible delay times after a collision.

To make the analysis of this broader problem feasible, we will have to make another simplifying approximation. Think about the collision resolution process from the point of view of a single computer and ask "What is the probability that this computer tries to transmit in slot 1?" Given that we assumed Q possible delay times would be used, the answer is clearly $\frac{1}{Q}$. Similarly, the probability that the computer transmits in the second or third slot is also $\frac{1}{Q}$. Suppose that instead of having the computer pick a random number of slots, we instead had it simply roll a Q sided die at the start of each slot and transmit only if the die came up showing "1". It would transmit with probability $\frac{1}{Q}$ in the first slot and with probability $\frac{1}{Q}$ in the second and third slots too. If you don't think about it too hard, it seems like having computers decide when to transmit by rolling dice in this way would be equivalent to having them pick random waiting times after each collision.

Unfortunately, if you think about the processes a bit more carefully, you will realize they are not the same. For one thing, if a computer first picks a delay between 1 and Q slots, it is guaranteed to try to transmit within the next Q slots. If it instead decides when to transmit by rolling a die at the beginning of each slot, it may roll "0" Q times in a row and therefore not transmit within Q slots. The dice-rolling approach, however, does approximate some features of picking waiting times. In particular, the probability that a computer will transmit in the first slot is $\frac{1}{Q}$ under both approaches. As a result, studying the dice-rolling model can give us some insights into how

an actual Ethernet would behave. Best of all, the dice-rolling model is much easier to work with mathematically.

To resolve the collision in the first slot immediately after the collision, exactly one computer must decide to transmit and $N-1$ computers must decide not to transmit. The computers decide whether or not to transmit in a slot completely independently. Therefore, to determine the probability that a certain computer transmits successfully in the first slot, we multiply together the independent probabilities of these N events: 1 computer deciding to transmit and $N-1$ computers deciding not to transmit. If the probability that a station will transmit is $\frac{1}{Q}$, then the probability that a station will decide not to transmit is just its complement, $1-\frac{1}{Q}$. Therefore, the probability that some particular computer transmits alone is

$$\frac{1}{Q} \times (1-\frac{1}{Q})^{N-1}$$

This can happen to any of the N computers involved in the initial collision, so the probability, p, that any computer transmits successfully is

$$p = \frac{N}{Q} \times (1-\frac{1}{Q})^{N-1}$$

Now, we can consider the probability of a successful transmission in the second slot time. Using the dice-rolling model, if no computer is able to start a successful transmission in the first slot, then they will all behave exactly as they did in the first slot during the second slot. Therefore, the probability of a successful transmission in the second slot is just p times the probability that no successful transmission started in the first slot, $1-p$. That is, the probability of a successful transmission in the second slot is

$$Prob\{\text{success in slot 2}\} = p \times (1-p)$$

The same reasoning applies to the third, four, and in general the Sth slot. To have a successful transmission in slot S, the computers must fail to transmit successfully for $S-1$ slots. The probability of having $S-1$ failures is just $(1-p)^{S-1}$. Therefore, the probability of a successful transmission in slot S is

$$Prob\{\text{success in slot S}\} = p \times (1-p)^{S-1}$$

Note that this formula applies even to slots 1 and 2.

6.9 Get What You Expect

We would like to get some estimate of how to pick the number Q that determines how computers will pick their random waiting times. We would like to pick Q in a way that enables stations to resolve collisions as quickly as possible. Therefore, rather than looking at the probability of success in a given slot, what we would like is an estimate of the expected number of slots required before some station is able to transmit successfully after a collision.

Recall that given the probability of a set of outcomes, we can compute the expected outcome by adding together the product of each possible outcome and its probability. That is, the expected number of the slot that must pass before a collision will be resolved is

$$\sum_{S=1}^{\infty} (S-1) \times Prob\{\text{success in slot S}\}$$

or

$$\sum_{S=1}^{\infty}(S-1)\times p\times(1-p)^{S-1}$$

The expression shown above can be rewritten as

$$p\times\sum_{S=0}^{\infty}S\times(1-p)^{S}$$

Then a miracle happens...

For reasons explained in Figure 6.30, this can then be transformed into the closed form

$$p\times\sum_{S=0}^{\infty}S\times(1-p)^{S}=p\times\frac{1-p}{(1-(1-p))^2}=\frac{1-p}{p}$$

For values of p between 0 and 1, the function $\frac{1-p}{p}$ is monotonically decreasing, as shown in Figure 6.31. Accordingly, to minimize the number of slots wasted, we should pick a value for Q that maximizes

$$p=\frac{N}{Q}\times[1-\frac{1}{Q}]^{N-1}$$

This requires a bit more magic. To find the value of Q that maximizes p, we use a standard technique from calculus, we determine the derivative of p with respect to Q and then determine the value of Q that makes the derivative equal 0. p will be largest for this value of Q.

If you know/remember enough to apply the product rule and the chain rule, you will recognize that

$$\frac{dp}{dQ}=\frac{-N}{Q^2}[1-\frac{1}{Q}]^{N-1}+\frac{N}{Q}(N-1)[1-\frac{1}{Q}]^{N-2}\frac{1}{Q^2}=\frac{N}{Q^3}[1-\frac{1}{Q}]^{N-2}(N-Q)$$

Given that the maximum value of p occurs when

$$\frac{dp}{dQ}=\frac{N}{Q^3}[1-\frac{1}{Q}]^{N-2}(N-Q)=0$$

we can conclude that the best value for Q is $Q=N$. That is, if N computers collide, they should each choose to delay between 0 and $N-1$ slot time. This will result in

$$p=\frac{N}{Q}\times[1-\frac{1}{Q}]^{N-1}=(1-\frac{1}{N})^{N-1}$$

which is the highest probability of success possible given N computers. There is no fixed best range from which delays should be chosen! The best range depends on the number of computers competing to use the network. The more computers involved, the less aggressive they need to be about attempting to retransmit after a collision.

6.10 You Better Backoff!

Given the analysis in the preceding section, computers on an Ethernet should increase the number of delay slots used after a collision when the number of stations involved in a collision increases. Unfortunately, this is easier said than done. When messages collide on an Ethernet, all that a

Justification of formulas for infinite sums

Consider the two infinite sums

$$G = \sum_{S=0}^{\infty} q^S \qquad\qquad E = \sum_{S=0}^{\infty} S \times q^S$$

In the case that $\mid q \mid < 1$, both of these sums converge. There is a clever technique that can be used to determine the closed forms for both of these sums in the cases where they do converge.
For the sum G, note that

$$(1-q)G = G - qG = \sum_{S=0}^{\infty} q^S - \sum_{S=0}^{\infty} q^{S+1} = \sum_{S=0}^{\infty} q^S - \sum_{S=1}^{\infty} q^S = q^0 = 1$$

Therefore, if the sum G converges, it must be the case that

$$G = \frac{1}{1-q}$$

Similarly, note that

$$(1-q)E = E - qE = \sum_{S=0}^{\infty} S \times q^S - \sum_{S=0}^{\infty} S \times q^{S+1} = \sum_{S=1}^{\infty} S \times q^S - \sum_{S=1}^{\infty} (S-1) \times q^S = \sum_{S=1}^{\infty} q^S = G - 1$$

Given that

$$(1-q)E = G - 1$$

we can conclude that

$$E = \frac{G-1}{1-q} = \frac{\frac{1}{1-q} - 1}{1-q} = \frac{q}{(1-q)^2}$$

Figure 6.30: Determining the value of sums involving geometric series

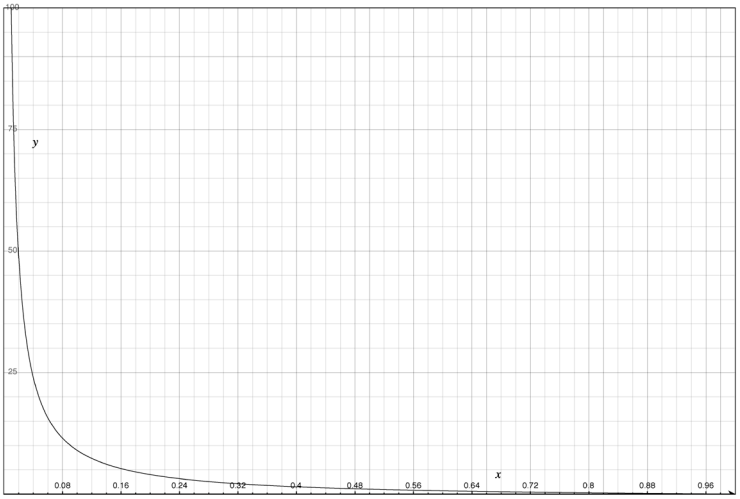

Figure 6.31: Slots wasted as a function of p

computer transmitting can tell for sure is that at least one other message is colliding with its own message. There is no direct way for a computer to tell how many computers there are somewhere out on the Ethernet who want to transmit at the same time.

Fortunately, there is a way for the computers to at least estimate how many other computers are competing for the network. Basically, if a collision occurs when the stations involved use Q delay slots, chances are that Q is not big enough for the number of stations involved. On the other hand, if a computer transmits successfully, it may be the only computer still trying to transmit and should lower the value of Q it uses.

With this in mind, Metcalfe and Boggs proposed an algorithm called *binary exponential backoff* for determining the number of slots a computer chooses from when deciding how long to wait after a collision. When a collision first occurs, a computer assumes that there is only one other computer involved in the collision. That is, it sets its own value for Q equal to 2. If additional collisions occur, each computer involved in the collision doubles its value of Q. That is, after two consecutive collisions, a machine will choose delays between 0 and 3 slot times. After three collisions, it will choose between 0 and 7 slot times, and so on. Finally, when a computer succeeds in transmitting, it resets its value of Q back to 2 for the next time it encounters a collision.

Metcalfe and Boggs wanted to make sure computers would never backoff for extremely long time periods. They decided that if a computer encountered 16 consecutive collisions, it probably should not actually wait for up to $2^{16} = 65536$ time slots. Therefore, after 16 collisions, the Ethernet hardware in a computer will abandon hope and declare that a transmission error has occurred.

A critical thing to notice about this algorithm is that each computer maintains its own value of Q. As a result, two computers on the network may be using different values for Q at the same time. For example, suppose two computers named A and B collide. Suppose that after this collision, both A and B choose to delay for 1 slot and that just as they make their second attempt, another computer named C tries to transmit. All three computers will detect this second collision. For A and B, this will be a signal to increase their values of Q to 2. Since C was not involved in the first collision, it will view the second collision as its first collision and start with Q equal to 2. This

Figure 6.32: Examples of wireless hubs

means that as the three computers try to resolve their 3-way collision they will be using different ranges of delay times. A and B will choose delays between 0 and 3 slots. C will choose between 0 and 1.

6.11 Semper WiFi

So far in this chapter, we have focused on Ethernet. Wireless or WiFi networks share many properties with Ethernet. In this section, we will briefly discuss the techniques used in standard wireless local networks stressing features they share with and ways in which they differ from Ethernets.

Any device on an Ethernet can send a message to any other device on the Ethernet. The devices on an Ethernet may include user computers, servers, printers, etc., but from the point of view of the network they are all simply computers that can send and receive messages. This is also true of wireless networks, in theory. In practice, however, most wireless networks have a distinguished device that serves as the focal point of most of the communications that takes place on the network. This device is the *wireless hub*.

When you use a wireless network in a college library, a local coffee shop, or even in a rest stop on the New York Thruway, the network you are using was installed to provide *flexible* access to the resources of the Internet. Not having to find a jack and plug in is wonderful when you are checking your email or the news. When setting up a mail server or web server, flexibility is less important than reliability and efficiency. As a result, most of the computers on wireless networks belong to users looking for information. They do not act as servers providing information. The servers are typically located on separate, wired networks. Some mechanism is needed to enable the computers on a wireless network to talk to servers on wired networks.

That is where wireless hubs like those shown in Figure 6.32 fit in. These devices are designed to simultaneously be plugged in to a wired network like an Ethernet and to communicate wirelessly. When you try to access a web page or read your mail through a wireless network, your computer sends the request to a wireless hub which forwards it to a server through the wired network. The

server then sends its response to the wireless hub which forwards it to your computer. As a result, on many wireless networks almost all message travel to or from the hub.

While it is common for a hub to play this role, it is not essential. Just as on an Ethernet, any device on a wireless network can in theory send messages to any other device in the network without going through the hub. On the wireless network in my home, there is one example of this. Our printer has a built-in wireless network interface. When anyone using a computer in our house wants to print, the computer sends requests and receives responses from the printer directly through the wireless network without using the hub.

As one might expect, the main differences between Ethernet and wireless networks relate directly to the fact that wireless networks don't use wires. The wires used in an Ethernet do one thing that is critical to the functioning of the CSMA/CD techniques we have described in the preceding sections. They guarantee that computers on an Ethernet are always close enough for a signal sent by one computer on the network to reach every other computer on the network.

As an electrical signal travels through a wire, it gradually becomes weaker the farther it travels from its source. This process is called *attenuation*. We have already seen that the Ethernet protocol limits the total size of an Ethernet to ensure that signals do not take too long to travel from one end of the network to the other. The protocol also places strict limits on the size of individual segments to ensure that signals are never attenuated to a point where they can no longer be accurately detected while traveling through the network. If an Ethernet has been installed following the guidelines in the protocol specification, any two computers plugged in to the network will be close enough to ensure they can receive one another's signals.

This is not possible on a wireless network. Since computers using a wireless network are not plugged in, there is nothing to prevent a user from moving a few yards farther away from the other computers using the network. Basically, the "installer" of a wireless network typically only controls the placement of the hub. The actual dimensions of the network are determined by the network's users.

If a person using a wireless network moves so far away from the network that the signals from her computer can not reach the wireless hub or any of the computers using the wireless network, she will be unhappy, but the network will work just fine for everyone else. The real problem occurs when a computer is positioned close enough to reach some of the computers that form a wireless network, but not all of them. In this case, the "carrier sense" mechanism included in Ethernet can no longer be used effectively.

To understand this issue, let us first see how we might hope that carrier sense would work in a wireless network. Figure 6.33 shows three computers set up to communicate with one another using a wireless network. In the figure, the series of concentric arcs located to the left of the computer identified as "C" are intended to indicate that C is sending a message while the other two computers, A and B, are idle. The large circle surrounding C in Figure 6.33 is meant to represent the area that falls within the range of the signals C is transmitting.

Wireless signals weaken as they get farther from their source for two reasons. First, just as the electrical signals are attenuated as they travel through an Ethernet cable, the radio waves carrying a wireless signal are attenuated as they travel through the air (not to mention the occasional wall) from source to destination. Second, the signals weaken because they are dispersed in three dimensions as they travel from the source. The signals in an Ethernet cable, by comparison, are likely to be much stronger when they reach a receiver because they can only travel in one dimension along the path of the cable.

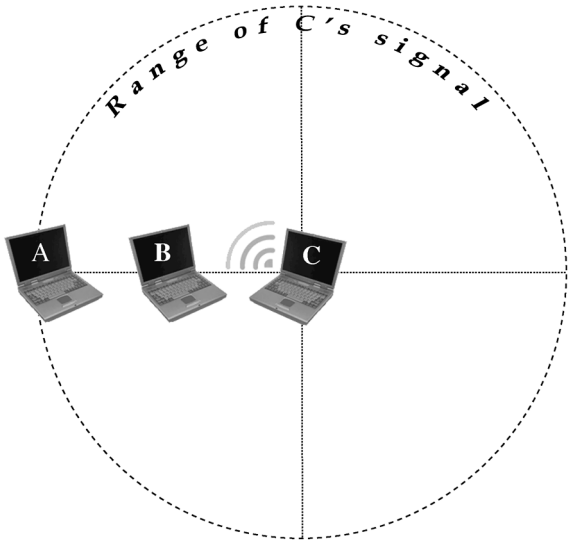

Figure 6.33: Carrier sense prevents A from transmitting

Note that in Figure 6.33, both A and B are located within the circle indicating the range of C's transmission. As a result, both A and B will detect any message C sends. In particular, suppose that C was sending a message to B and A became ready to send a message of its own to B while C was still sending. If A follows the carrier sense rule, it will notice that C is already sending and delay its transmission to avoid colliding with C's.

Consider how things change if we move computers A, B, and C a bit farther from one another. Figure 6.34 shows such a configuration. Again, we have included a circle indicating the range of C's transmission. This time, B is still close enough to receive messages from C, but A is out of range. It will not detect signals sent by C.

In this situation, even if A follows the carrier sense rule, it may still collide with C's transmission. If A decides to start sending after C, when A listens for conflicts it will not hear C's transmission because C is too far away. A will therefore incorrectly conclude that it is safe to start its transmission to B. Unfortunately, while A is too far from C for signals from either A or C to reach the other machine, A is close enough to B for its transmission to reach B and interfere with the message B was already receiving from C as shown in Figure 6.35. This issue is known as the *hidden node problem*. It means that the "Carrier sense" part of "Carrier sense multiple access with collision detection" will not really work on a wireless network designed for mobile computers.

Worse yet, if you think about Figure 6.35 for just a moment, you will realize that the collision detection mechanisms we described earlier will not work either. Suppose that computers A and C had started transmitting at nearly the same time as one another. On an Ethernet, they would eventually detect one another's transmissions, stop transmitting, and try again later at random times. In the scenario shown in the figure, they will fail to detect any collision since they can not hear each other's transmissions. As a result, their messages will not be retransmitted even though neither message will have been received by B.

Standard wireless networking protocols use two techniques to provide functionality similar to what collision detection and carrier sense provide in Ethernet. In the absence of the ability to

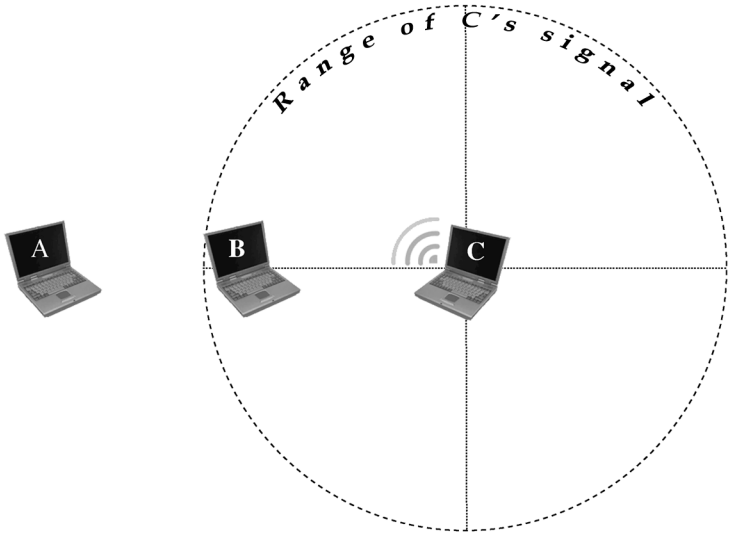

Figure 6.34: Distant computer A can not hear C

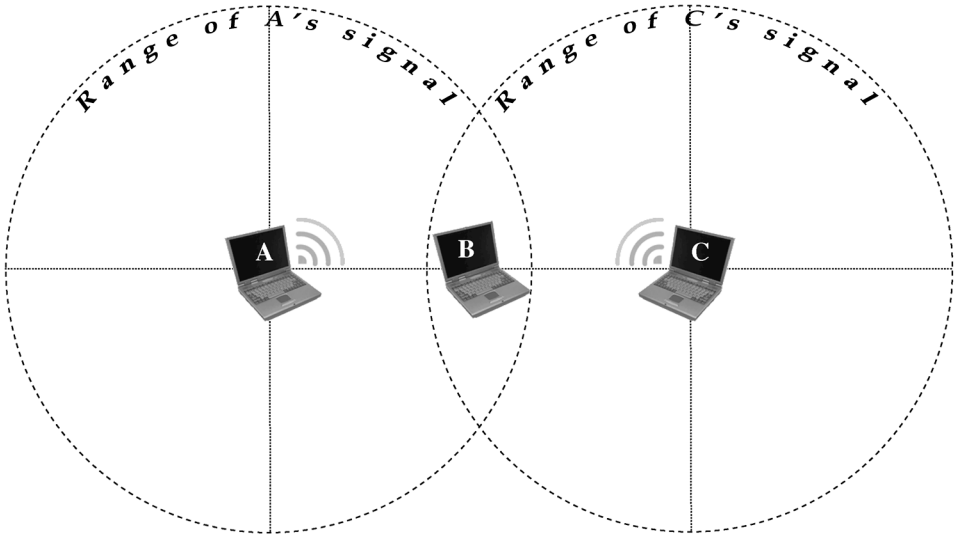

Figure 6.35: A's transmission collides with C's at B

detect collisions, computers on wireless networks must depend on the recipient of each message to send back a message acknowledging the receipt of a transmission. That is, in a situation like the one shown in Figure 6.33 where C's message should be received by B without any collision, B would send a message to C immediately after it received the message from C just to confirm the receipt. If C receives such an acknowledgment from B, it can safely assume no collision occurred. If C does not receive such an acknowledgment, it must assume that a collision occurred and retransmit its packet to B.

Unfortunately, while this technique does detect collision, it does not do so very efficiently. If a wireless network depends upon acknowledgments to detect collisions, the amount of time wasted by a single transmission depends on the length of messages sent rather than on the physical size of the network. In Figure 6.35, if A and C are sending messages containing thousands of bits, they can not detect the collision until their messages are completed and they fail to receive any acknowledgments. Such a collision can waste far more time than the 512 bit time limit imposed on collisions on an Ethernet.

To avoid such wasteful collisions, wireless networks support a two-phase process for sending messages. First, the source computer and destination computer exchange two very short messages called a request to send (RTS) and a clear to send (CTS). The source computer starts the process by transmitting an RTS containing the size of the data it wants to send. If the destination receives the RTS (i.e., the destination is within range and no other transmission collides with the RTS), then the destination responds by sending a CTS which also contains the length of the data to be sent. When the source computer receives the CTS, it goes ahead and sends its actual message. Finally, just to be sure, the destination sends an acknowledgment for this message.

The advantage derived from sending the RTS and CTS messages comes from the way that are processed by computers other than the source and destination. Since both the RTS and CTS message contain the size of the data the source computer really wants to transmit, any computer that receives either the RTS or CTS can estimate how long it will be before the source computer finishes sending its data. Any computer other than the source that hears either the RTS or CTS assumes that the network will be busy for this long and refrains from starting any transmission of its own during this period. This process is referred to as *virtual carrier sense*.

To appreciate the advantage of virtual carrier sense, return to the scenario shown in Figure 6.34. This time, however, assume that what C is sending to B is not its actual packet, but is instead an RTS. Just as before, A will not be able to hear this transmission. Therefore, if A gets the urge to send a message while A is sending the RTS to B, A will go ahead and collide with C's RTS, requiring both A and C to try again later. Compared to the data C really wants to send, however, the RTS is likely to be quite short. As a result, the probability that a collision will occur is lower. There is simply a shorter period of time during which A can collide with the RTS than with a data message.

If A does not send a message that collides with the RTS, B will soon send a CTS message to let C know that it is ready to receive data from C. The state of the network while B is transmitting its CTS is shown in Figure 6.36. The key is that both A and C will receive the CTS. In fact, any computer close enough to B to collide with a transmission from C to B will receive B's CTS. Since the CTS contains the length of the message that C plans to send to B, A and all similarly located computers will refrain from sending new messages until enough time has passed for C to complete its transmission. Virtual carrier sense effectively prevents collisions after the RTS and CTS have been exchanged. Collisions can still happen during the transmission of the RTS and CTS, but since

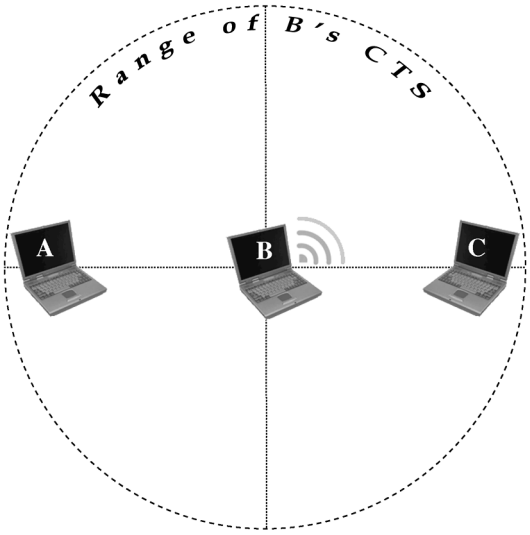

Figure 6.36: B's CTS reaches A and C

these message are small, the amount of time wasted by the collisions is likely to be smaller than would be wasted if a collision occurred involving the actual data transmission.