

Animation of Computer Networking Concepts

MARK A. HOLLIDAY

Western Carolina University

A number of the key concepts in the design of computer networks lend themselves to illustration through animation. Animation can make the main features of these concepts accessible to the beginning undergraduate student as well as to more advanced students. We have identified six of these networking concepts: packet encapsulation; packet fragmentation; error control; media access in Ethernet local-area networks; domain name resolution; and the hypertext transfer protocol. We developed Java applets and accompanying materials to illustrate four of these concepts. The applets serve two roles: one role as a visual representation of different scenarios with respect to the concept in question; the second role as a vehicle for experimentation. We discuss the applets, the sequence of points each applet is designed to convey, and how the user can conduct experiments to further understand the networking concept. For the other two concepts, we show how the student can use free, easily available software to observe how the concept is implemented in the Internet.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education--*Computer science education*; C.2.5 [**Computer-Communications Networks**]: Local and Wide-Area Networks--*Ethernet, Internet*

General Terms: Design, Performance, Reliability

Additional Key Words and Phrases: Protocol stack, encapsulation, fragmentation, error control, Selective Repeat, CSMA/CD, media access, Java applets, name resolution, hypertext transfer protocol.

1. INTRODUCTION

The importance of the Internet and the Web makes it clear that the basic principles of computer networks should be understood by every student majoring in computer science. In fact, making these principles accessible to an even broader audience is a worthwhile goal. This article reports on a long-term project whose aim is to make computer networking principles accessible to the broadest possible audience. The primary approach taken by this project is to use animation. Fortunately, a number of these principles lend themselves naturally to illustration by animation. We have identified four such principles and animated them as Java applets.

The applets serve two roles: one as a visual representation of different scenarios for the concept in question; the second as a vehicle for experimentation. The applets are designed to encourage user interaction. The user can make changes in parameter settings and then restart the applet to see the effect of the changes. As we show later, these experiments can be used to systematically extend the concept that each applet attempts to convey.

The first applet illustrates the movement of a packet through protocol layers on the source host, intermediate routers, and destination host. We illustrate two important

This research was supported by the National Science Foundation (DUE-96-50458)

Author's address: Department of Mathematics and Computer Science, Western Carolina University, Cullowhee, NC 28723; email: holliday@email.wcu.edu.

Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2004 ACM 1531-4278/04/0600-ART2 \$5.00

features of the traversal through the protocol layers: encapsulation and fragmentation. The second applet illustrates some key protocols to ensure reliable data transfer, that is, error control, in the Internet. The protocols are used at the data-link layer, the transport layer, and the application layer. The third applet illustrates media access in Ethernet local-area networks, as well as the key distinction between link bandwidth and propagation delay. We decided not to develop applets for two other computer networking concepts, i.e., resolution of Internet domain names to Internet Protocol (IP) addresses and the operation of the hypertext transfer protocol (HTTP). Instead, for these concepts, there is free, easily available software that the student can use to observe how the concept is implemented in the Internet. We feel this alternative approach for engaging the student has complementary benefits to the applet animation approach. This approach allows the student to observe the actual Internet in action, although there is not the visual representation of the applets.

The remainder of this article is organized as follows. Section 2 introduces the protocol stack applet. The error-control applet user interface and concepts are covered in Section 3. The Ethernet applet user interface and concepts are covered in Section 4. Section 5 outlines how the other two concepts, name resolution and the hypertext transfer protocol, are illustrated by showing the student how to use free, easily available software to see implementations of these concepts on the Internet. Section 6 reviews related work on the animation of computer networking concepts. We conclude in Section 7.

2. PROTOCOL STACK APPLET

The first fact that a student of computer networking learns is that networking software on the source host, the intermediate routers, and the destination host is divided into protocol layers that form a protocol stack. The packet moves down through the layers on the source host, up through the layers on each intermediate router, back down the layers on that intermediate router, and then finally up through the layers on the destination host. The first applet, the protocol stack, illustrates this fact with each protocol layer shown in a different color and with one intermediate router. This applet illustrates two other important concepts: encapsulation and fragmentation.

2.1 Encapsulation

Encapsulation means that each protocol layer requires its own header, so as the packet moves down through the protocol layers (on the source host, the router, or the destination host) a header is added at each protocol layer. Thus, the size of the packet changes as it moves down the layers. The bottom left side of the applet is a table showing that the original packet at the application layer is 2000 bytes and that the headers added at the transport layer, the network layer, and the data-link layer are 20 bytes, 20 bytes, and 16 bytes, respectively. Thus the packet being placed on the network (at the source host, the router, or the destination host) is 2056 bytes. As the packet moves up through the protocol layers, the headers are removed in reverse order; this is called de-encapsulation. All of the scenarios that the applet simulates show the process of encapsulation and de-encapsulation. When fragments are created, the details of encapsulation become a bit more subtle, as discussed in the section on fragmentation.

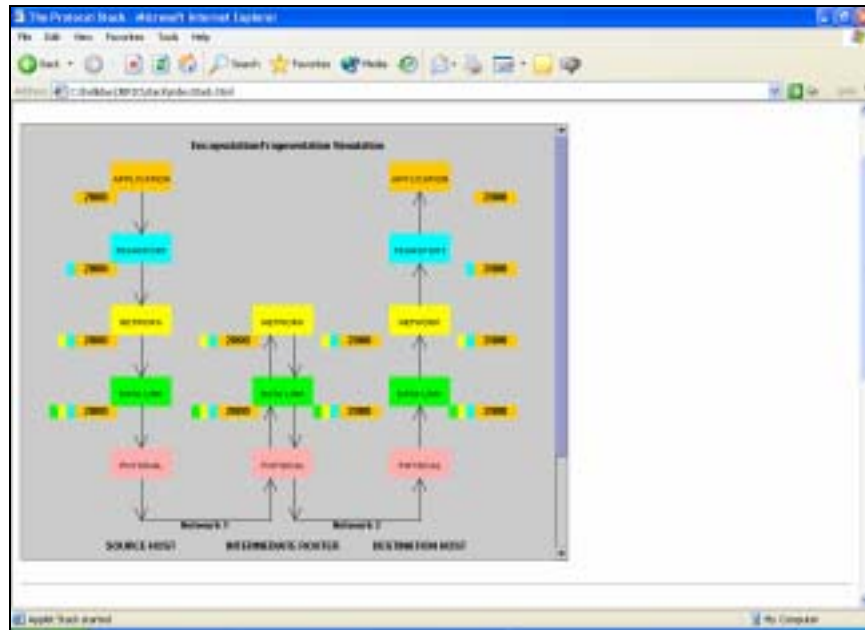


Fig. 1. Screenshot of the top part of the protocol stack applet. The packet has traveled down through all the layers on the source host, up and down the layers on the intermediate router, and up the layers on the destination host. Encapsulation and de-encapsulation by the addition and removal of packet headers is shown occurring along the way. In this scenario, fragmentation does not occur since the maximum transmission unit size of each network is large enough not to require fragmentation. The control area of the applet is visible in Figure 2.

Figure 1 shows the simplest scenario that the applet illustrates, where the packet traverses the protocol layers and undergoes encapsulation and de-encapsulation, but not fragmentation and reassembly. Even this simplest scenario shows that the intermediate router only goes up through the network layer. This aspect of the visual representation helps to reinforce to the user that the network layer is responsible for routing from the source host to the destination host. Hence, the routers do not need to look at the packet at any layer above the network layer.

To show all the protocol layers, including copies of the packet, requires a relatively large applet window, which may make it larger than small monitor screens. To solve this problem, I made the applet window size relatively small for all the applets, but added scrollbars so that all of the applet can be seen by scrolling. The protocol stack applet is where the scrollbar is usually visible. Figure 1 shows the top part of the applet, the part containing the animation; the control area is visible in Figure 2.

2.2 Fragmentation

Each type of network has a maximum transmission unit (MTU), which is the maximum size of the packet's payload as it goes across the network. In this context the packet's payload means the packet without the data-link header. In other words, the

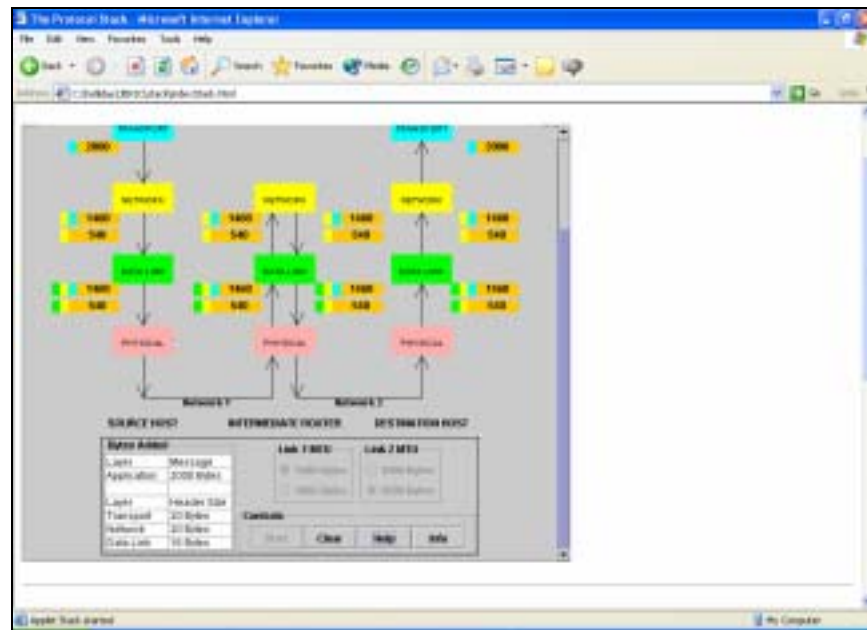


Fig. 2. Screenshot of the bottom part of the protocol stack applet. In this scenario, fragmentation occurs only at the first network. The application layer is not shown in the screenshot.

payload of the packet being put on the network is the size of the entire packet delivered to the data-link layer on the machine that places the packet on the network. The network layer of that machine is responsible for ensuring that the packet the network layer delivers to its data-link layer is no larger than the MTU of that network.

If the packet that the network layer creates by adding a network layer header onto the packet it receives from the transport layer is larger than the MTU, then the network layer has to fragment the packet. The packet the network layer received from its transport layer is divided into fragments, with a network layer header added to each fragment so that the size of each fragment (except for the last one, possibly) is the largest possible, but no larger than the MTU of the network. Thus, every fragment will have a network layer header and a data-link header, but only the first fragment of a single original packet will have a transport layer header within it. Thus, as mentioned earlier, when encapsulation interacts with fragmentation, the appearance of the packet's encapsulation will differ somewhat from the one without fragmentation.

The applet is designed to illustrate all the cases to the user. In particular, the choice of radio button for the MTU of the first network and for the second network will result in four cases for a given message size (2000 bytes) and header sizes (20 bytes for transport layer header, 20 bytes for the network layer header, and 16 bytes for the data-link layer header). In all the cases, by using color the animation shows which headers are in each fragment, and also gives the size (in bytes) of the part of the fragment that is not a header. The default setting of the radio buttons as shown in Figure 1 (3000-byte MTU1 and 3000-byte MTU2) causes no fragmentation, and so only illustrates a simple case of

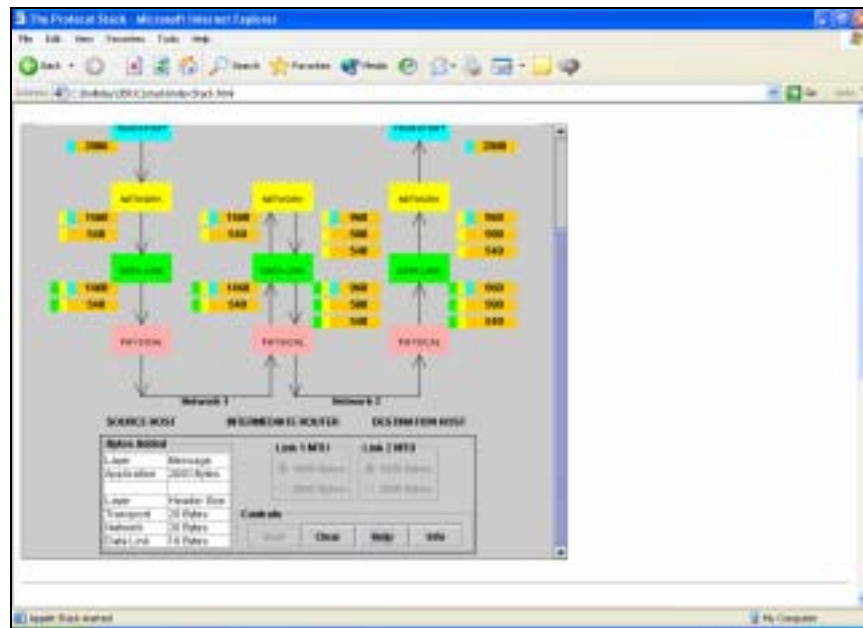


Fig. 3. Screenshot of the bottom part of the protocol stack applet. In this scenario fragmentation occurs at both networks. The application layer is not shown in the screenshot.

encapsulation and de-encapsulation. If the MTU1 is set to 1500 bytes and the MTU2 is left at 3000 bytes, then fragmentation occurs before the first network, but no further fragmentation occurs before the second network. Figure 2 shows this scenario. The applet window is scrolled down so that the control area is visible and most of the animation area is shown. The top part of the animation area is not shown, but is similar to the top part of Figure 1. If the MTU1 is left at 3000 bytes and the MTU2 is set to 1000 bytes, then no fragmentation occurs before the first network, but fragmentation does occur before the second network. If the MTU1 is set to 1500 bytes and the MTU2 is set to 1000 bytes, then fragmentation occurs before the first network and further fragmentation occurs before the second network. Figure 3 shows this case of fragmentation at both networks.

These scenarios can be understood at a superficial level (for example, “the network is too small, so you have to split the packet”), the intent, however, is to force the user to more thoroughly understand what is happening. Two aspects of the visual appearance help to force the user to think further. One is the color for the various headers, and in particular the appearance of the transport layer header only in the first fragment of the packet. The other aspect is displaying the number of bytes in each fragment, not counting the headers. To derive why those particular numbers are correct takes quite a bit of understanding on the student’s part. For example, in the case of a 1500-byte MTU1 and a 1000-byte MTU2, the size of the fragments entering the second network are different than in a 3000-byte MTU1 and a 1000-byte MTU2, even though in both cases the 1000-byte MTU2 is the smallest size. In the laboratory accompanying this applet, the user is encouraged to work through why the fragments are the sizes they are.

Reassembly occurs at the destination host as the fragments are reassembled into the original packet (with the headers also being removed through de-encapsulation). The notes point out that the alternative to reassembly, whenever the next network's MTU is large enough is possible, but not on the Internet. For example, reassembly in a 1500-byte MTU1 and a 3000-byte MTU2 at the intermediate router could be done, but is not (as illustrated by the applet animation).

2.3 Accompanying Materials

All the applets have the same set of accompanying web pages:

- *How do I use this applet?* This web page explains the main parts of the animation area: the protocol layers on the source host, the intermediate router, and the destination host, the networks, and the packet (and possibly its fragments) traversing the protocol layers. It also explains the control area, including the table with the sizes for the message and the headers, the radio buttons for the MTU of each network, and the buttons. There are four buttons. The *start* button causes a packet to start the traversal; the *clear* button clears all the steps of the packets along the traversal; the *help* button causes a dialog box to appear with some brief instructions about how to use the applet; the *info* button causes a dialog box to appear with a short explanation of the concepts that the applet illustrates.
- *What concepts does this applet illustrate?* This web page contains an in-depth discussion of the concepts that the applet addresses.
- *A laboratory* This web page has a series of exercises the student is encouraged to work through in order to understand the concepts that the applet illustrates in greater depth. As an example, here is a quote from the laboratory web page for one of the experiments.

Experiment Two: 1500 Byte MTU / 3000 Byte MTU

Repeat experiment one except for the first network select a 1500 Byte MTU.

- *What do you observe?*
- *Why does the network layer on the source machine fragment the packet?*
- *Why are the fragments the sizes that they are?*
- *Why does the network layer on the router not fragment the packet?*
- *Why do you think the network layer on the router does not reassemble the fragments it receives? Reassembling means merging the fragments back into a smaller number of fragments (ideally back to just one fragment).*
- *Observe that the network layer on the destination machine does reassembly. Why?*
- *Caveats and References* An applet cannot include every possible detail about how a packet traverses the protocol layers. For example, the applet does not illustrate the concept of multiplexing/de-multiplexing. This web page lists a number of such details and provides a list of references that can be used to learn more.

3. ERROR-CONTROL APPLET

Transferring data reliably across a channel (that is, providing error control) is an issue that can, and usually is, addressed at multiple layers of the protocol stack. The same family of protocols was developed and used in a number of the layers. We developed an applet and explanatory material to illustrate the key concepts in these protocols. More detailed and precise descriptions can be found in textbooks such as those by Cassel and Austing [2000] and Kurose and Ross [2003]; Holliday [2003] provides an introduction to this applet. Here we provide a much more in-depth treatment of how the applet illustrates the concepts it is trying to convey.

We teach computer networks using a top-down approach, starting with the application layer and progressing down to the physical layer. This approach is used in several computer network textbooks for the computer science audience, such as those by Cassel and Austing [2000] and Kurose and Ross [2003]. With this approach it is natural to introduce error control in the context of the Transmission Control Protocol (TCP) of the transport layer; we do so in our applet.

The most basic concept the applet conveys is the provision of error control through retransmission and using acknowledgments and timeouts to determine when to retransmit. The simplest protocol that uses this approach is called Stop-and-Wait, and it only allows one outstanding packet. The second important concept is that when more than one outstanding packet is allowed, performance improves; protocols that allow more than one outstanding packet are called Pipelined protocols. The third concept is that multiple outstanding packets complicate error control when losses occur. Pipelined protocols handle losses by taking two main approaches: Selective Repeat and Go-Back-N; the applet illustrates Selective Repeat. The fourth concept is that changes in the bandwidth and propagation delay in the channel change the behavior of the protocol. These concepts are discussed in order in the following sections.

3.1 Acknowledgments and Timers

One of the responsibilities of the Transmission Control Protocol (TCP) is to successfully transfer a packet (we use the general term, *packet*, though the more precise term is TCP segment) from the source host to the destination host. The key to achieving this requires that the TCP entity on the source host retransmit the packet until it is guaranteed that the TCP entity on the destination host has received the packet and received it in an uncorrupted state. This raises the issue of how to know when TCP is to retransmit and when to stop retransmitting. TCP uses two mechanisms: acknowledgments from the destination host, and a timer on the source host that is associated with each send of the packet.

The first and most important concept illustrated by the applet is the interplay between acknowledgments and timers and their roles in retransmission. Figure 4 is a screenshot of the applet; the middle of the applet shows packets moving from the sender to the receiver and acknowledgments moving from the receiver to the sender. To convey that time is moving to the right, the packet moves to the right as well as downward, and the acknowledgment moves to the right and upward. The legend at the top right of the applet shows that normal packets are in blue, normal acknowledgments are in green, and lost or corrupted packets or acknowledgments are in red. The number under a packet or acknowledgment is the sequence number.



Fig. 4. A screenshot of the error-control applet. One packet has been sent and its acknowledgment is now returning to the sender.

In Figure 4, one packet (with sequence number 1) has been sent successfully from the sender to the receiver and the corresponding acknowledgment is now being returned (successfully so far, since it is in green) to the sender. The acknowledgment contains the sequence number of the packet that it is acknowledging. When the acknowledgment reaches the sender, the sender sends the second packet (with sequence number 2). The sender knows that it does not need to retransmit packet 1. This scenario is the simplest and best, since no retransmissions are needed. It is displayed by selecting the Stop-and-Wait protocol from the group of radio buttons labeled “Protocol” and selecting the No Loss radio button from the group of radio buttons labeled “Loss Scenario.”

The second scenario occurs when the radio button labeled “Stop-and-Wait” is still selected, but the Lost Data radio button is selected as the Loss Scenario. In this case the packet turns red as it moves from the sender to the receiver and then disappears. Since the receiver never receives the packet, an acknowledgment is not sent. The timer associated with the packet expires (times-out), which triggers a retransmission.

The top left of the applet illustrates the timers. Each outstanding packet (we are assuming only one so far) has a line on its timer showing how much time has elapsed. When the line reaches the right end, a time-out occurs.

The third scenario takes place when the radio button labeled “Stop-and-Wait” is still selected, but the Lost ACK radio button is selected as the Loss Scenario. In this case the acknowledgment turns red as it moves from the receiver to the sender and then disappears. The timer associated with the packet expires (times-out), which triggers a retransmission.

By these three scenarios the applet conveys the basic concept of using acknowledgments and timers to detect when packets should be retransmitted. In all three scenarios the applet transmits a total of two packets, one at a time. Notice that there are issues we are not addressing. For example, one possible scenario is that the acknowledgment arrives at the sender uncorrupted but after the timer has expired; another is that a packet or acknowledgment instead of being lost could arrive corrupted.

3.2 Multiple Outstanding Packets

The second concept illustrated by the applet is that allowing multiple outstanding packets improves performance. The three scenarios displayed by the applet that we have described so far assume only one outstanding packet. The scenarios are easy to understand and help the user comprehend the role of acknowledgments and timers. The protocol that allows only one outstanding packet is called Stop-and-Wait, since the sender stops after transmitting the packet and waits until the sender receives the acknowledgment.

The protocols that allow multiple outstanding packets are called Pipelined protocols, since the sender can continue to send packets even when the acknowledgment for the first packet has not yet arrived. Thus, these protocols are related to the design of high-performance pipelined CPU designs that allow more than one instruction to be executed concurrently. There has to be an upper limit, called the window size, on the number of outstanding packets. In fact, Pipelined protocols are also called “Sliding Window” protocols, since the window with the packets that are eligible for sending slides to the right as acknowledgments of earlier packets arrive. In a Pipelined protocol the sender can start sending a second data packet before the sender receives the acknowledgment for the first data packet. Thus, if the sender needs to send several packets, then the time until the last of the packets is sent will be shorter with a Pipelined protocol. Hence Pipelined protocols demonstrate better performance than the Stop-and-Wait protocol.

In this case the goal of the applet is to illustrate the improvement in performance because the sender does not have to wait before sending the second packet. The Pipelined radio button is selected from the protocol group of radio buttons and the No Loss radio button is selected from the Loss Scenario group of radio buttons to show the simplest case of no errors. This scenario makes the clearest case for improving performance with pipelining, since the orthogonal issue of retransmission due to error does not appear. Figure 5 is a screenshot illustrating the performance improvement. We assume a window size of three packets. So to ensure that there are never more than three outstanding packets, the sender does not send another packet if three packets have already been sent and not acknowledged; the applet has the sender sending a total of four packets.

In the Stop-and-Wait scenarios the acknowledgment contains the sequence number of the packet that is being acknowledged. In the Pipelined protocols it turns out to be better for the acknowledgment to instead contain the sequence number of the next packet that the receiver expects. In case of no errors, this would be one more than the sequence number of the packet the receiver just received; but in the case of errors, it may not be (as we will discuss later). The reason is that the packet that the receiver wants to acknowledge may not be the packet that it just received.



Fig. 5. A screenshot illustrating multiple outstanding packets. The sender has sent three packets and the acknowledgment for packet 1 (which had sequence number 2 in it) has reached the sender and the acknowledgments for packets 2 and 3 (which have sequence numbers 3 and 4, respectively, in them) are still in transit. The applet for the Pipelined scenarios assumes a window size of four packets and that the sender can have at most three outstanding packets.

3.3. Selective Repeat

The third concept the applet conveys is how errors are handled in the case of multiple outstanding packets. The approach of acknowledgments, time-outs, and retransmissions is still used. This protocol differs from Stop-and-Wait in what the sender should do after retransmitting the packet that has been identified as needing retransmitting. Should the sender retransmit any of the other packets, even though those other packets have not yet had time-outs? The answer depends on how the receiver is implemented. There are two approaches to this problem. In one, called "Go-Back-N," the receiver is kept as simple as possible, in particular the receiver does not buffer out-of-order packets. Consequently, the sender needs to retransmit the packets that might have appeared as out-of-order to the receiver. In the second approach, called "Selective Repeat," the receiver does buffer out-of-order packets. So the sender only needs to retransmit the lost or corrupted data packet. Selective Repeat causes fewer packets to be retransmitted, but requires more overhead on the part of the receiver.

The obvious question is: Does TCP use Go-Back-N or Selective Repeat? The answer is that it uses a combination [Kurose and Ross 2003, pp. 245-246]. Explaining the specifics of that combination is beyond the scope of the applet. Instead, the applet illustrates Selective Repeat, since Selective Repeat is a good approximation that conveys the spirit of TCP error control. Selective Repeat (and Go-Back-N) is a Pipelined

protocol, so the Pipelined radio button in the Protocol group of radio buttons should still be selected. For the Loss Scenario to exhibit the special behavior of Selective Repeat we now need to create an error, so we select either the Lost Data radio button or the Lost ACK radio button.

In either case, since the window size is three, the sender initially sends three packets. Selecting the Lost Data radio button causes one of these first three packets to be lost in between the sender and the receiver. The lost packet briefly turns red and then disappears to reflect the loss. Which packet is lost is randomly generated, so that different runs of the applet will cause different situations to occur. How the receiver responds differs, depending on which packet was lost. Thus, the user will want to run the simulation multiple times to see all the cases.

As shown in the Stop-and-Wait simulations with losses, one way that the sender detects that a packet needs to be retransmitted is that the timer for the packet expires before an acknowledgement for the packet reaches the sender. So upon a timeout the sender retransmits the packet. Thus, the role of the acknowledgment is just to cancel the timer. In fact, in the Stop-and-Wait case the sequence number specified inside the acknowledgment is there to aid the user of the applet. Since there is only one outstanding packet, the sender knows what packet is being acknowledged.

In Pipelined protocols the acknowledgment has a more complex role. Since there can be multiple outstanding packets, it is not as clear which packet the acknowledgment is acknowledging. Thus, the sequence number within the acknowledgment is important information to the sender (sender means sender of the packet; not the sender of the acknowledgment). The approach is that if the acknowledgment contains the sequence number $N + 1$, then the receiver means that the receiver has received all the packets numbered 1 through N . This can be used in two ways.

- (1) When an acknowledgment containing the sequence number $N + 1$ reaches the sender, the sender can cancel the timers that are still active for any packets numbered 1 through N . Even if an acknowledgment is lost, but a later acknowledgment reaches the sender, then the sender still knows that the packets acknowledged by the earlier acknowledgment reached the receiver. Thus, the sender does not need to retransmit those packets.
- (2) If a receiver receives a packet that is not the next one it expects (either an earlier sequence number than expected or a later sequence number than expected), then the receiver just resends an acknowledgment with the sequence number that it is expecting. So the receiver can send duplicate acks; that is, multiple acknowledgments that have the same sequence number. Thus, when the sender receives a duplicate ack, there is a good chance that a packet was lost. In this case, the sender could retransmit the later packets, just in case, even though their timers have not expired.

One of the original three packets sent by the sender is lost before reaching the receiver in the class where the Lost Data radio button was selected. Which one is lost is selected randomly. If the first packet is lost, the receiver buffers packets two and three (unlike Go-Back- N) and sends an acknowledgment when packet two is received and an acknowledgment when packet three is received. Both acknowledgments specify a packet



Fig. 6. A screenshot illustrating Selective Repeat with the second packet (packet 2) lost. The receiver sent an acknowledgment for packet 1 which has already reached the sender. Due to the arrival of that acknowledgment, the sender has slid its sliding window over by one packet, which allowed it to transmit packet four. Packet four is shown in blue, moving down from the sender to the receiver. The screenshot also shows an acknowledgment (in green) arriving at the sender. This is the acknowledgment the receiver sent when the receiver received packet three. This acknowledgment has the number two in it, since the receiver is saying that it is still expecting packet two (since packet two was lost, it was never received). Next, the sender retransmits packet two, owing to the arrival of the duplicate ack.

sequence number, since that is still the next packet number that the receiver expects. When the sender receives the acknowledgment with packet number one in it, the sender knows that packet one has been lost, even though the timer for packet one has not expired. Consequently, the sender retransmits packet one and restarts the timer for packet one.

Note that the sender only retransmits packet one and not packets two and three. In other words, the sender is selective about which packets to repeat (that is, retransmit), and hence uses Selective Repeat. When the second copy of packet one reaches the receiver, the acknowledgment returned by the receiver has packet sequence number four in it because the receiver had buffered packets two and three. So the sender can cancel the timers for packets two and three, and also cancel the timer for packet one, since the sender knows the receiver must have received packet one also. If the second packet is lost, the sequence of events is similar, but just shifted over one. The receiver acknowledges packet one with an acknowledgment with sequence number two in it. When that acknowledgment reaches the sender, the sender cancels the packet one timer and transmits packet four (since up to three packets can be outstanding). The receiver buffers packet three and sends acknowledgment with sequence number two in it, since

the receiver is still expecting packet two. When this second acknowledgment with sequence number two in it reaches the sender, the sender knows packet two is lost. Consequently, it retransmits packet two and restarts the timer for packet two. When packet four reaches the receiver, the receiver buffers packet four and still replies with an acknowledgment that still has packet sequence number two in it. When, finally, the new copy of packet two reaches the receiver, the receiver replies with an acknowledgment with packet sequence number five in it. Thus, the sender will know that packets three and four have been received by the receiver as well as packet two. Figure 6 illustrates this case.

If the third packet is lost, the sequence of events is similar, but just shifted over one more; the details are left to the reader.

In the case where the Lost ACK radio button was selected, the original three packets are received by the receiver without any problems. The receiver then sends an acknowledgment in response to the reception of each of the packets. However, before all the acknowledgments reach the sender, one of them is lost; in other words, turns red and then disappears. Which acknowledgment is lost is randomly generated so that different runs of the applet will cause different situations to occur. How the sender responds will differ depending on which acknowledgment was lost. Thus, the user will want to run the simulation multiple times to see all the cases.

The applet in the Lost ACK case illustrates how receiving a later acknowledgment results in not needing to retransmit. If the first acknowledgment is lost, the second one causes the sender not to retransmit; if the second acknowledgment is lost, the third one causes the sender not to retransmit; if the third acknowledgment is lost, then the fourth one causes the sender not to retransmit. The fourth acknowledgment occurs because once the first acknowledgment reaches the sender, the sender slides its window and can now send its fourth packet. The fourth acknowledgment is sent by the receiver when that fourth packet reaches the receiver.

3.4 Bandwidth and Propagation Delay

The two choices for the protocol, Stop-and-Wait and Pipelined, as well as the three choices for the loss scenario, No Loss, Lost Data, and Lost ACK, result in six cases (actually, there are more cases because the lost packet or lost acknowledgment is randomly generated in the case of Pipelined protocols). The implications of the different cases were discussed in the sections above. However, what happens in those cases also depends on the settings of four other parameters: the time-out length, the packet size, the link bandwidth, and the propagation delay. The final concept the applet conveys is the effect on the protocol's behavior due to changes in the bandwidth and propagation delay of the channel between the source host and the destination host. The user can change the bandwidth and the propagation delay using two of the text fields on the left-side of the control area at the bottom of the applet. The user can also change the packet size with another text field on the left side. The applet does not allow the time-out length to be changed.

The animations resulting from the six possible combinations (depending on which radio buttons are selected) will look different according to changes in packet size, bandwidth, and propagation delay. Thus, by allowing the user to change the values of the packet size, bandwidth, and propagation delay, the applet encourages the user to

experiment. Increasing the bandwidth reduces the time needed between transmitting the first and the last bit of a packet. So the packet's rectangle becomes shorter and wider in the applet. In contrast, increasing the propagation delay causes an increase in the time it takes for the first bit of the packet to reach the receiver. The applet illustrates this by the packet moving more slowly from the sender to the receiver. The timers are also slowed by a corresponding amount, since we do not want time-outs in a time less than it takes the packet to reach the receiver and the acknowledgment to return.

3.5 Accompanying Materials

All the applets have the same set of accompanying web pages:

- *How do I use this applet?* This web page explains the main parts of the animation area: the timer array, the legend that explains the meaning of the different colors for packets, and the area that shows the packets between the sender and the receiver. This web page also explains the control area, including the group of text fields for changing network parameters (link bandwidth, packet size, and link length); the group of radio buttons that specify the protocol (Stop-and-Wait or Pipelined); and the group of radio buttons that specify the loss scenario (No Loss, Lost Data, or Lost ACK) and the four regular buttons. The Start button begins a simulation based on the radio buttons that have been selected. When a loss occurs in Pipelined protocol cases, the particular packet or acknowledgment that is lost is selected at random. The Suspend button suspends the current simulation, which helps the user understand the simulation by being able to stop its progress and study its current status. The Resume button causes a suspended simulation to resume. The Help button causes a dialog box to appear with brief instructions about using the applet.
- *What concepts does this applet illustrate?* This web page contains the in-depth discussion of the concepts addressed by the applet.
- *A laboratory* This web page has a series of exercises the student is encouraged to work through for a deeper understand of the concepts illustrated by the applet. As an example, here is a quote from the laboratory web page for one of the experiments.

Experiment Three: Stop-and-Wait Protocol and Lost Acknowledgment
Repeat experiment two except select the Lost ACK radio button.

1. What is different about the simulation this time?
2. Why does the receiver send an acknowledgment this time unlike in experiment two?
3. How is the timer at the top of the animation area involved?

- *Caveats and references* An applet cannot include every possible detail on how to ensure reliable data transfer in the transport layer; for example, as noted earlier, TCP actually uses a combination of Selective Repeat and Go-Back-N. Furthermore, the sequence number used by TCP is the next byte expected, not the next packet expected. This web page lists a number of such details and provides a list of references that can be used to learn more.

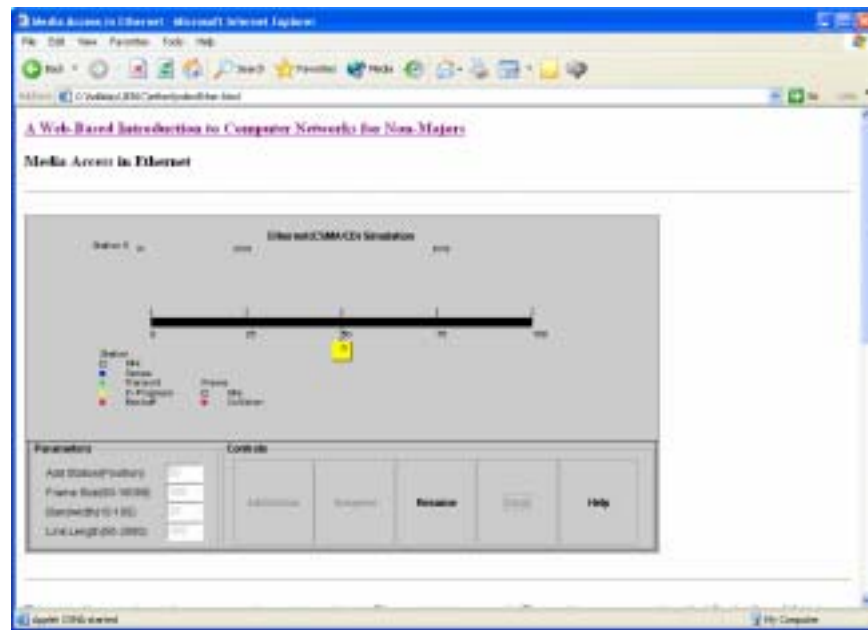


Fig. 7. A screenshot showing a single station in the In Progress state with both frame copies propagating to the ends of the link.

4. MEDIA ACCESS APPLLET

Ethernet is an important type of local-area network. Multiple stations (that is, computers) can be on a single Ethernet link; but only one station can transmit on the link at one time. So how should access to the link (that is, the media) among the stations be arbitrated? The protocol is Carrier Sense Multiple Access/Collision Detection (CSMA/CD), for which we developed an applet and explanatory material to illustrate some of its implications. Holliday [1997] gives a brief introduction to an early version of this applet. Here we provide a much more in-depth treatment of how the applet demonstrates the concepts it tries to convey. In particular, we emphasize how the applet is designed to encourage the student to experiment in order to derive a more in-depth understanding of the implications of CSMA/CD.

The applet primarily illustrates the states that a station goes through when attempting to transmit a frame both with and without conflict. However, there are two important related concepts that naturally develop from the same animation. One is why the correct operation of CSMA/CD requires a minimum frame size. The other concept is the effect of bandwidth and propagation delay on the behavior of the propagating copies of the frame and on the states of the station.

4.1 The Simplest Case

The simplest concept conveyed by the applet is how a station transmits a packet successfully, assuming no other station is trying to transmit. As shown in Figure 7, the

main part of the applet is the Ethernet link with the numbers on the vertical marks that shows the distance from the left end. The user places a station on the link by entering a position in the text field labeled “Station Position” and then clicking the Add Station button. The rectangle for the station then appears just below that position on the link. The station rectangle contains the number that is the identity of the station, starting with 1 for station one.

As shown in the legend on the left side of the applet, a station can be in one of five states, each state displayed with a different color. A transparent rectangle means the station is in the *Idle* state, which is the initial state of the station. Clicking on the station’s rectangle causes the station to start transmitting a frame. The station first enters the *Carrier Sense* state (blue rectangle) to *sense* (i.e., check) whether the frame of another station is passing by the sensing station. The carrier (that is the link) is idle, so the station moves to the *Transmit* state (green rectangle) and starts transmitting the frame. The length of time it takes to transmit the frame (that is, the time from the emission of the first bit of the frame to that of the last bit of the frame) depends on the link’s bandwidth. The higher the bandwidth the shorter the transmit time. When transmitting ends, the station moves to the *In Progress* state (yellow rectangle) until the frame has reached both ends of the link, when the station returns to the *Idle* state (transparent rectangle). In this simplest case, in addition to showing the station state transitions, the applet also shows the movement of the frame. There is a bracket for station 1 above the link. When the station transmits the frame, copies of the frame appear in that bracket, propagating in both directions to the ends of the link. The frame copies are transparent, meaning that neither has experienced a collision.

4.2 Undetected Collisions

The next concept illustrated by the applet is the case where multiple stations attempt to transmit frames at about the same time and the protocol works incorrectly. The default link length, the default frame size, and the default bandwidth are designed to make it easy for the protocol to fail. The user by experimenting with station placement and the timing of the user’s click on each station (which initiates an attempt to transmit) can determine when the protocol fails and when it succeeds.

Place two stations on the link and then click on both station rectangles to cause both stations to attempt transmitting. A problem becomes evident with the default link length, the default frame size, and the default bandwidth values. The *Carrier Sense* state does not prevent all the attempted transmissions that we would like to prevent. Any time a frame of another station is propagating along the link, the station in *Carrier Sense* state would, ideally, realize that it should not start transmitting because this would inevitably cause the new frame to collide with the frame already on the link. However, the *Carrier Sense* state simply detects whether a frame is propagating past the sensing station. If the frame has not yet reached the sensing station or has already completely passed it, then the sensing station will not detect the frame.

When the *Carrier Sense* state does not detect a propagating frame, then the station in state *Carrier Sense* moves to the *Transmit* state and starts transmitting its frame. One of its frame copies and one of the copies of the already propagating frame will then collide. The applet shows this by turning the frame copies red when they encounter each other in the bracketed area at the top of the applet.

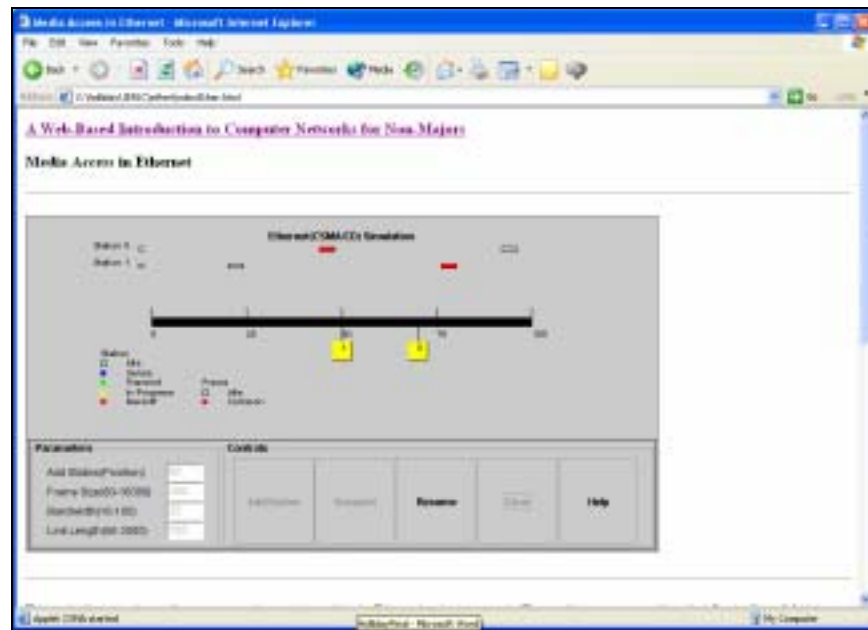


Fig. 8. A screenshot illustrating a frame collision that is not detected by the transmitting stations. The red state of the frame copies indicates the collision, but station rectangles are yellow, which means they are in the *In Progress* state and have not (and will not) detect the collision.

Another problem is that the stations may not detect that their frames have collided. Using the default, link length, default frame size, and default bandwidth, it is easy to place stations and click on them so that both stations transmit frames, the frames collide, but the stations do not detect the collisions (shown in Figure 8). This situation occurs because a station detects a collision only if a copy of the frame that has experienced a collision propagates past the station while the station is still transmitting its own frame (that is, while it is in *Transmit* state). The goal of these experiments is to make clear to the user that a correct solution to this problem is more subtle than the user might have expected.

4.3 A Solution

The next concept the applet intends to convey is what the user needs to do to change the applet settings to solve the protocol failure identified in the previous section. The first problem in the previous section, that the *Carrier Sense* state does not detect all cases of a frame already propagating, does not have a practical solution. To solve this problem, the sensing station would have to have global knowledge about the presence of frames at all points along the link at the same time. That is not possible, so frame collisions will occur. However, the second problem identified in the previous section, i.e., that a station does not detect that a frame collision has occurred, can be solved. The user is encouraged to experiment with the settings of the link length, frame size, and bandwidth to create situations where no frame collision will go undetected by all the stations involved. The

applet includes an accompanying laboratory that walks the user through a series of experiments (<http://cs.wcu.edu/~holliday/cware/ether/etherLab.html>) and helps the user derive the constraint on the link length (which, given the fixed speed of the media, determines the propagation delay), frame size, and bandwidth so that frame collisions are always detected. In the next two paragraphs we briefly summarize some of the key ideas in those experiments.

Since a station only detects collisions while the station is in *Transmit* state, every transmitting station must stay in the *Transmit* state long enough to ensure that it has detected all possible collisions. How long is enough? The worst case occurs when the two stations are at opposite ends of the link and the second station senses the carrier (i.e., the Ethernet link) just before the frame from the first station reaches the second station. In this case, the second station will transmit its frame and the first station will not detect the collision until the second station's frame reaches the first station. Thus, the first station must be in state *Transmit* for a time equal to at least twice the maximum propagation delay on the link. To determine that this is the worst case, the student can experiment with station placement and the timing of clicking on stations

How do you ensure that the first station is in state *Transmit* long enough? The time in state *Transmit* is proportional to the frame size and inversely proportional to the bandwidth. In particular, the minimum *Transmit* state time equals the number of bits in the minimum frame size times the number of bits per second of the bandwidth. That time must be larger than twice the maximum propagation delay. The laboratory accompanying the applet explains how the Ethernet specification specifies a minimum frame size, bandwidth, and propagation delay (which is determined by the link length) so that this constraint is met to ensure that the protocol works correctly.

4.4 Detecting All Collisions

The next concept the applet intends to convey is that the CSMA/CD protocol works correctly when the minimum frame size is large enough. In this case, each station does detect a collision. As shown in Figure 9, each station goes to state *Backoff* (red rectangle) and a timer appears underneath each red rectangle. The timer counts down and the station then moves to the *Carrier Sense* state to try and retransmit. A station may encounter a collision several times in succession. In such a case the applet correctly reflects the behavior of CSMA/CD. Upon each collision the exponential backoff time is randomly chosen from an interval twice as long as the previous interval.

4.5 Bandwidth versus Propagation Delay

The final concept conveyed by the applet is the distinction between bandwidth and propagation delay. The bracket along the top of the applet shows the frame that is propagating for each station. The size of the bracket reflects the bandwidth of the link. When the link bandwidth is increased, each bracket automatically becomes wider. Consequently, when a frame propagates, the rectangle for each copy of the frame becomes wider. Thus, the time from when the station transmits the first bit of the frame until it transmits the last bit (that is, the transmit time) decreases because the rectangle does not have to be as long, since the rectangle has become wider and the area of the rectangle (the number of bits) is constant. The student can experiment with different bandwidth and frame sizes to see these changes. Figure 10 illustrates these experiments. In Figure 10 as in Figure 9 the frame size has been increased to 1000 bytes from the

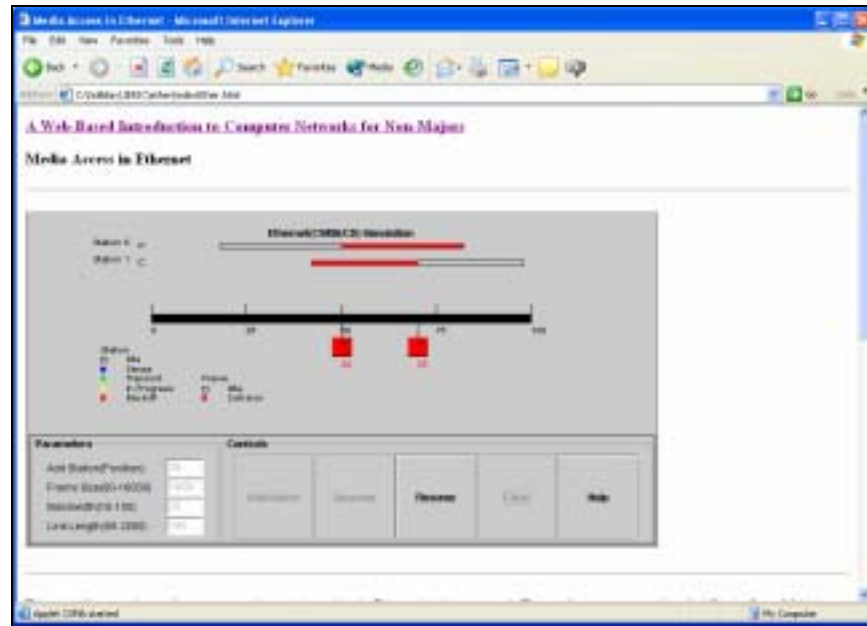


Fig. 9. A screenshot showing a frame collision detected by the transmitting stations. The red state of two of the frame copies indicates the collision. The station's red rectangles indicate that both have detected the collision. Note the timers counting down below each station as each station does its backoff, and that the frames are much longer because the frame size has increased to 1000 bytes. Thus, the transmit time has increased because the bandwidth has not changed.

default value of 100 bytes. However, in contrast to Figure 9, the bandwidth in Figure 10 has also been increased to 100 megabits per second from the default value of 25 megabits per second. As a result, the rectangle for each frame copy has a much different shape in Figure 10 than it does in Figure 9, and both differ from the shapes in Figures 7 and 8.

The total latency of a frame consists of two parts: the transmit time and its propagation delay. Its transmit time is the time from the beginning of its transmission (that is, its first bit leaves the station) until its last bit leaves the station. The frame's propagation delay is the time that any particular bit (for example, the last bit of the frame) leaves its station to the time that bit reaches the furthest end of the link. The applet makes clear that changing the link bandwidth changes the frame's transmit time by the change in the shape of the rectangle in the copies of the frame. The applet also makes clear that changing the link bandwidth does not change the frame's propagation delay. Thus, the experiments help the student understand an important point: that increasing the link bandwidth has only limited effect on the total latency of the frame because only the transmit time of the frame decreases, not its propagation delay.

In contrast, since the propagation delay is the length of time it takes a particular bit of the frame to reach the end of the link after leaving the station, the propagation delay is determined by how fast the frame moves and not on how long or how wide the frame is. The user can change the propagation delay by changing the link-length text field. The

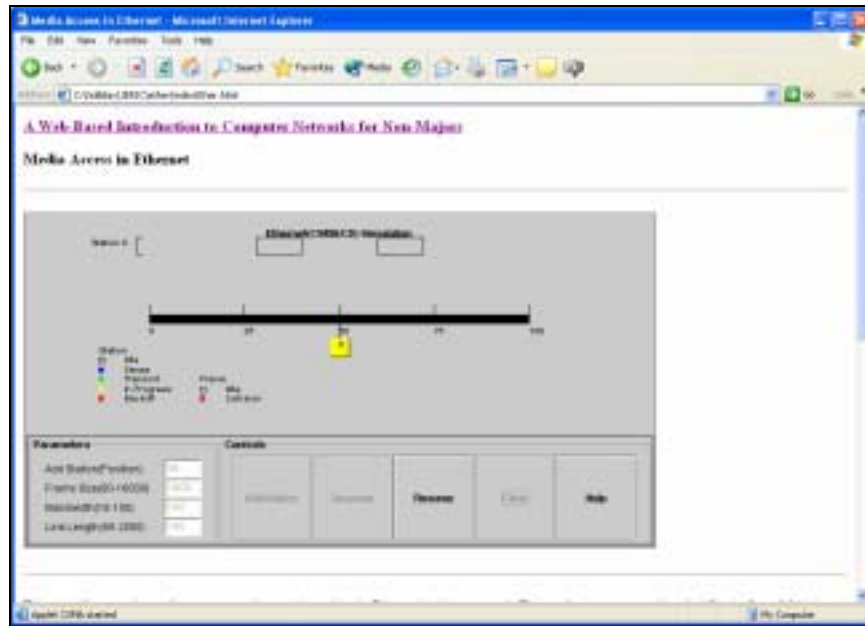


Fig. 10. A screenshot illustrating how increasing the bandwidth causes the rectangle of each frame copy to become wider and how increasing the frame size causes the rectangle of each frame copy to become longer.

result is that the numbers underneath the link that indicate the distance from the left end to different points change to reflect the new link length. That the change in link length changes the propagation delay is shown by a change in the speed at which the frame copies move towards the link ends. For example, if the link length has increased, the frame copies move more slowly to reflect the longer propagation delay due to the longer link length.

After observing how changing the link length changes propagation delay (as reflected by the speed with which the frame copies move), the user can then conduct an experiment and see how much changing the link length changes the total latency for different frame sizes for a given link bandwidth. In some cases of frame size and link length, total latency is dominated by the transmit time (and thus by the value of the link bandwidth); but for other cases total latency is dominated by the propagation delay (and thus by the value of the link length).

4.6 Accompanying Materials

All the applets have the same set of accompanying web pages:

- *How do I use this applet?* This web page explains the main parts of the animation area: the area showing the propagating frame copies, including how their heights change as the bandwidth changes; the link itself with the station rectangles at the specified positions; and the number below a station rectangle showing the countdown of an exponential backoff when that station is in a backoff; and the legend explaining the meaning of the different colors for a

station's rectangle and for the frame copies. This web page explains how clicking on a station's rectangle in the animation area causes that station to attempt to transmit a frame.

This web page also explains the control area, including the group of text fields for specifying the position of a new station, the link bandwidth, the frame size, and the link length, and the five regular buttons. The Suspend button suspends the current simulation, which helps the user understand the simulation by enabling the user to stop the progress of the simulation and to study the current simulation. The Resume button causes a suspended simulation to resume; the Clear button removes all the stations and frame copies from the animation area; the Help button causes a dialog box to appear with some brief instructions on how to use the applet.

- *What concepts does this applet illustrate?* This web page contains the in-depth discussion of the concepts addressed by the applet.
- *A laboratory* This web page has a series of exercises that the student is encouraged to work through in order to more deeply understand the concepts that the applet illustrates. As an example, here is a quote from the laboratory web page for one of the experiments.

Experiment Three: Backoff Behavior

Experiment two considers one complication that can arise when two or more stations are on the link and attempting to transmit. This experiment examines another complication. This complication involves the protocol behavior when a station detects that a frame it is transmitting has or will collide with a frame from another transmitting station. Place two stations on the link. Click on one station to start it transmitting a frame. Click on the second station to start it transmitting a frame before the frame of the first station reaches the second station; thus, the complication described in experiment two will not occur. Initially set the simulation parameters (frame size, link bandwidth, and link length), the position of the stations, and the timing of the clicks so that each station has finished emitting its frame before the frame copy of the other station reaches it.

- 1. Describe what happens to the frame copies of the two stations. Explain when each frame copy changes state.*
- 2. Describe what happens to each of the stations. Explain when each station changes state.*
- 3. Now modify the timing of the clicks so that each station has not finished emitting its frame before the frame copy of the other station reaches it.*
- 4. Describe what happens to each of the stations in this case. Explain when each station changes state.*
- 5. What is the meaning of the number that appears below each station's rectangle? The number appears below a station's rectangle only after that station has detected that its frame is involved in a collision.*

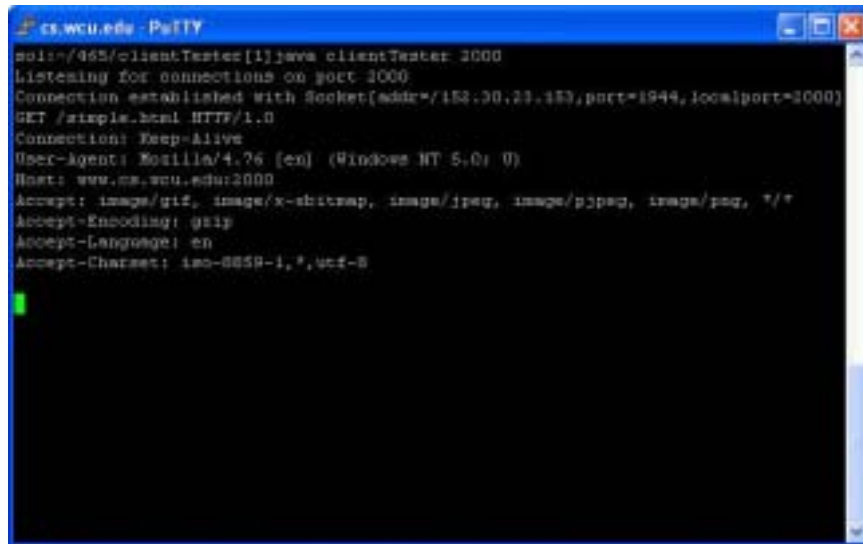
- *Caveats and references* An applet cannot include every possible detail about how media access is ensured in Ethernet local-area networks. For example, the calculations for determining the maximum link length are more complex than I have indicated, and depend on the type of media (e.g., coaxial cable or fiber optic cable) and how many repeaters are included. This web page lists a number of such details and provides a list of references that can be used to learn more.

5. NAME RESOLUTION AND HTTP

The website described in this article is more than just the three applets and their accompanying material. There is also material to help the student understand other aspects of computer networking so as to place the applets in context and to introduce two other concepts: name resolution and the hypertext transfer protocol. This overview and two new concepts are covered in three web pages.

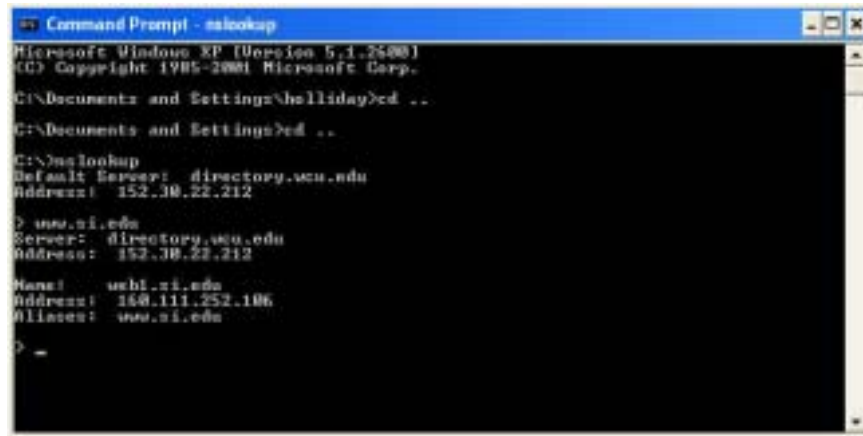
- *Preliminaries*: This web page is the first page that the student reads. It is written for an audience of undergraduates who are not majoring in computer science. It introduces the basic ideas of a network, a local-area network, a wide-area network, a protocol stack, an internetwork, the Internet as a specific internetwork, and the TCP/IP protocol stack as the one used by the Internet. This page also identifies and gives brief descriptions of the key concepts introduced by the website, i.e., encapsulation (protocol stack applet), fragmentation (protocol stack applet), error control (error control applet), name resolution (no applet), the hypertext transfer protocol (HTTP), and the web (no applet), and media access (media access applet).
- *Name resolution*: This web page introduces the following: Internet Fully Qualified Domain Names (FQDN); various simplifications or aliases so that users do not have to write a complete FQDN; IP addresses; that name resolution is used to convert a domain name into an IP address; how name servers are used in name resolution; discussions of the network utilities: nslookup, ping, tracert, netstat, and arp. Examples including screenshots are included for each of those network utilities, showing how users themselves can use that utility to observe network behavior. Figure 11 is an example of the screenshot illustrating the use of the nslookup utility.
- *HTTP and the web*: This web page is intended to explain to the user how the web is implemented as an application running at the application layer of the Internet. First, that a web page is simply HTML that is rendered by the web browser is shown by screenshots of a simple web page as seen by a web browser and a regular editor (which just shows the HTML of the web page). Users are shown how to repeat the illustration themselves. The HTTP protocol is then introduced as the application layer protocol running on top of TCP used to transfer web pages between web servers and web browsers. The steps involved in a HTTP protocol are presented, including the messages and their formats.

Users are then shown how to demonstrate on their own the messages exchanged by HTTP. The clientTester program in Harold [2000] can be used by students to show the HTTP request message sent by a web browser, as shown in Figure 12. The clientTester program simply listens on a specified TCP port and



```
cs.wcu.edu - Perl
sol:/465/clientTester[1]java clientTester 2000
Listening for connections on port 2000
Connection established with Socket[addr=/152.30.22.153,port=1944,localport=2000]
GET /simple.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.76 [en] (Windows NT 5.0; U)
Host: www.cs.wcu.edu:2000
Accept: image/gif, image/x-bitmap, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

Fig. 11. A screenshot illustrating the use of the nslookup utility to find the IP address of the domain name web1.si.edu (using the name server at domain name directory.wcu.edu).



```
Command Prompt - nslookup
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\helliday>cd ..
C:\Documents and Settings>cd ..
C:\>nslookup
Default Server: directory.wcu.edu
Address: 152.30.22.212
> www.si.edu
Server: directory.wcu.edu
Address: 152.30.22.212
Name: web1.si.edu
Address: 160.111.252.106
Aliases: www.si.edu
> -
```

Fig. 12. A screenshot illustrating how the clientTester program can be used to display a HTTP request message.

echoes to the console any message that it receives from a client that establishes a connection to that port. Users cause the web browser to establish a connection to that port by specifying the port number as part of the URL in the location window of the browser.

Users are then shown how to observe a HTTP response message sent by a web server. This is done by the users simply creating a command prompt

```

cs.wcu.edu - PuTTY
sol:~/465/clientTester[1]telnet
telnet> open cs.wcu.edu 80
Trying 152.20.5.1...
Connected to tinuviel.cs.wcu.edu.
Escape character is '^]'.
GET /simple.html HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 17 Feb 2004 00:15:38 GMT
Server: Apache/1.3.26 (Unix) Debian GNU/Linux PHP/4.1.2
Last-Modified: Tue, 17 Feb 2004 00:14:26 GMT
ETag: "2be010-e2-40318ee+"
Accept-Ranges: bytes
Content-Length: 226
Connection: close
Content-Type: text/html; charset=iso-8859-1

<html>
<head>
<meta name="author" content="Mark Holliday">
<title>A very simple web page</title>
</head>

<body>

<!-- This is a comment -->
Hello world!
<br>
Hello <font color="red">Mark!</font>

</body>
</html>Connection closed by Foreign Host.
sol:~/465/clientTester[1]

```

Fig. 13. A screenshot illustrating how the telnet program can be used to display a HTTP response message.

window and then using the telnet command to make a TCP connection to the port on which the web server is listening. The users then enter the character strings of a legal HTTP request message. The web server will then send to the command prompt window the character strings of the corresponding HTTP response message, as shown in Figure 13.

6. RELATED WORK

Transferring data reliably across a channel (that is, providing error control) is an issue that can and usually is addressed at multiple layers of the protocol stack. The same family of protocols has been developed and used in a number of the layers. The simplest of these protocols is called the Stop and Wait protocol; Shifroni and Ginat [1997] describe a simulation of this protocol. Our error-control applet also includes Stop and Wait as one example. The Shifroni and Ginat [1997] simulation however differs in that the students act out the protocol as a game instead of viewing it as an animation.

McDonald at the University of Western Australia [McDonald 1991] developed the cnet network protocol simulator. Cnet is a sophisticated and well-documented simulator used for over a decade in undergraduate computer networking courses. It simulates networks consisting of point-to-point links and Ethernet segments; cnet visually displays hosts connected by links with a user-written topology file that determines how the links are connected. Students can add protocols to the simulator by writing the protocols in the ANSI-C programming language. The cnet approach complements the one taken by the Java applet described in this article. Our Java applet approach attempts to convey visually some of the key user interaction concepts solely through buttons and text fields. Cnet

supports more ambitious user experimentations, with the users writing topology files and source code for network protocols. Finally, the website for the computer networks textbook by Kurose and Ross [2003] contains a collection of Java applets to illustrate computer networking concepts. None of those applets is similar to our protocol stack applet. However, some of their applets are related to our error-control applet and our Ethernet applet. The two more sophisticated protocols for error control, are called Go-Back-N and Selective Repeat. Our applet illustrates Selective Repeat; their applet illustrates Go-Back-N.

For media access in Ethernet, Kurose and Ross [2003] provide an applet that does show the sequence of steps when two stations both attempt to transmit. Our applet is more general. We allow users to place an arbitrary number of stations at arbitrary locations instead of the number of stations and their locations being fixed. This, plus allowing changes in the frame size, the link bandwidth, and the link length, lets users create situations in which the protocol fails. This is important because users can then conduct experiments to determine when the protocol will work correctly and when it will not, including determining the minimum frame size. Our applet also uses visual cues extensively to indicate state changes. For example, the frame copies change to red when they collide and the station rectangle changes colors depending on the protocol state the station is in. Our applet also displays the changes in frame shape due to bandwidth and propagation delay changes. Understanding the implications of bandwidth and propagation delay on the protocol behavior is an important theme of our applet. Moreover, we provide a significant amount of accompanying explanatory material suggesting sequences of experiments for users.

7. CONCLUSIONS

This article describes an educational resource that can help students understand computer networking. A series of Java applets and explanatory material was developed to illustrate six key computer networking concepts. This resource has been used successfully in an introductory computer science course taken by students who are not planning to major in computer science. It has also been used successfully in an upper-level undergraduate computer networks course for computer science majors. The protocol stack applet illustrates the process of protocol layer traversal on the source machine, intermediate routers, and the destination machine. The encapsulation and de-encapsulation of the packet is shown in the animation. Through experimentation it also allows the user to learn how fragmentation and reassembly change the packets.

In the error-control applet, by selecting different scenarios with respect to the protocol and to the loss event, the user can see how reliable data transfer is ensured through retransmissions and time-outs. Through various experiments, users can also see an improvement in performance due to pipelining and the effect of changes to link bandwidth, frame size, and propagation delay. The Ethernet applet supports extensive experimentation by the user. By station placement, the timing of clicks on station rectangles, and changes in frame size and link bandwidth, users can animate an arbitrarily large number of scenarios involving the CSMA/CD media access protocol. In particular, users can experiment to determine those situations in which the protocol will work correctly and those in which it will fail.

Beyond the applets and their accompanying material, there are additional web pages that provide an overview of the Internet, the distinction between domain names and IP addresses, how name resolution takes place, and how the web is implemented using HTML and HTTP, and the format of HTTP messages. These additional web pages identify and demonstrate free and easily available software that students can use to understand the concepts of name resolution and the operation of HTTP.

Versions of the applets and the other material, including suggested experiments, have been used successfully in several courses at our university. The applets and other materials were originally used in my university's computer science course for students not majoring in computer science, to give an idea of the nature of computer science. The work described here was originally developed as a module for that non-major course. The experience was positive. In general, the students found the material interesting and useful. As the material increased, it became more than we could cover in one part of the course, so we sampled portions of the material instead of covering all of it.

After initial versions of the material had been used in the non-major course, I also started to use it for the computer networks course that I teach to junior- and senior-level computer science majors. I found that this group also found the material helpful and could understand its more complex ideas. For example, many of the students in the non-major course found the reason for a minimum frame size in a network using a CSMA/CD protocol difficult to grasp, but this idea was not a problem for the computer science students in the upper-level computer networks course.

Updated information and current versions of the applets and its accompanying material can be found at <http://cs.wcu.edu/~holliday/cware/>.

ACKNOWLEDGMENTS

I would like to acknowledge Rupa Bhagavan for her help in implementing an early version of the protocol stack applet, and also Michael Johnson for his help in implementing the error-control applet, while they were students at Western Carolina University.

REFERENCES

- CASSEL, L. AND AUSTING, R. 2000. *Computer Networks and Open Systems, An Applications Perspective*, Jones and Bartlett.
- HAROLD, E. R. 2000. *Java Network Programming, Second Edition*, O'Reilly, 2000.
- HOLLIDAY, M. A. 1997. An Ethernet Java applet for a course for non-majors. *Mathematics and Computer Education* 31, 2 (1997), 158-166.
- HOLLIDAY, M. A. 2003. A Java applet for illustrating Internet error control. Submitted for publication.
- KUROSE, J. F. AND ROSS, K. W. 2003. *Computer Networking: A Top-Down Approach Featuring the Internet, Second Edition*. Addison Wesley, Reading, MA. (Open Resources section of http://wps.aw.com/aw_kurose_network_2/).
- MCDONALD, C. S. 2003. A network specification language and execution environment for undergraduate teaching. In *Proceedings of the 21st SIGCSE Technical Symposium on CS Education* (San Antonio, TX, March 1991). ACM, New York, 25-34. Cnet is also linked to the SIGCSE Educational Links page (<http://www.acm.org/sigcse/topics/>) and available at <http://www.csse.uwa.edu.au/cnet/>.
- SHIFRONI, E., AND GINAT, D. 1997. Simulation game for teaching communication protocols. In *Proceedings of the 27th SIGCSE Technical Symposium on CS Education*. ACM, New York, 184-188.

Received August 2003; revised February 2004; accepted March 2004