PACKET SNIFFING

Objectives

- Recognizing and decoding certain packets of interest using Wireshark Ethernet packet sniffing tool.
- Learn about various network protocols such as IP, TCP, and ICMP.

Lab Setup

- PC with Network Interface Card (NIC) connected to a network.
- Wireshark tool installed on the PC

Background

"Tell me and I forget. Show me and I remember. Involve me and I understand."

Chinese proverb

One's understanding of network protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols" – observing the sequence of messages exchanged between two protocol entities, delving down into the details of protocol operation, and causing protocols to perform certain actions and then observing these actions and their consequences. This can be done in simulated scenarios or in a "real" network environment such as the Internet.

What is Packet Sniffer?

Packet sniffer is a program that captures all of the packets of data that pass through a given network interface, and recognizes and decodes certain packets of interest without modifying it. A packet sniffer is sometimes referred to as a network monitor, or network analyzer. It is normally used by network or system administrator to monitor and troubleshoot network traffic. However, it is sometimes also used by malicious intruders for illicit purpose such as stealing a user's password of credit-card number. By comparison, a firewall sees all of a computer's packet traffic as well, but it has the ability to block and drop any packets that its programming dictates. Packet sniffers merely watch, display, and log this traffic.

One disturbingly powerful aspect of packet sniffers is their ability to place the hosting machine's network adapter into "promiscuous mode." Network adapters running in promiscuous mode receive not only the data directed to the machine hosting the sniffing software, but also ALL of the traffic on the physically connected local network. Unfortunately, this capability allows packet sniffers to be used as potent spying tools. A packet sniffer can only capture packets within a given subnet.

The use of powerful packet sniffing software by people who lack a thorough understanding of TCP/IP and Internet protocols will — without question — create significant confusion and raise a large number of questions.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The packet capture library receives a copy of every link-layer frame that is sent from or received by your computer. Recall that messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 1, the assumed physical media is an Ethernet, and so all upper layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

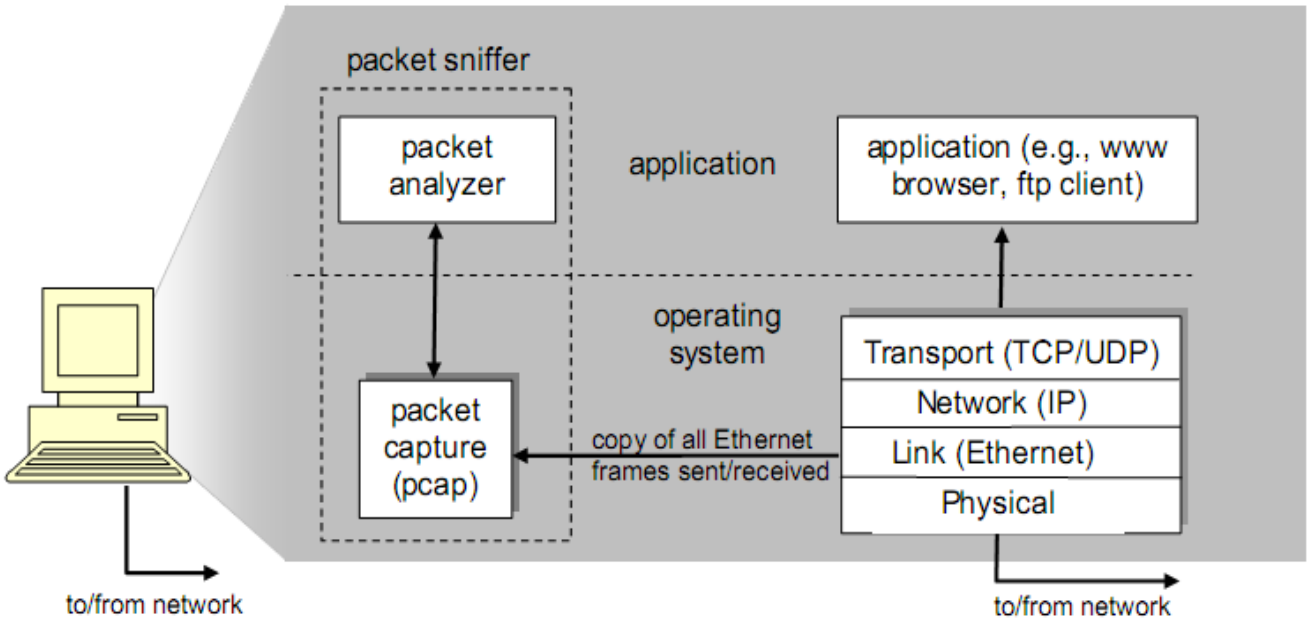

Figure 1: Packet sniffer structure

The second component of a packet sniffer is the packet analyzer, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer

must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol.

Packet Sniffing Tool

There exist various commercial and free packet sniffer tools. We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for this experiment, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It's an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support, rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, Token-Ring, FDDI, serial (PPP and SLIP), 802.11 wireless LANs, and ATM connections.

In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the libpcap or WinPCap packet capture library. The libpcap software will be installed for you, if it is not installed within your operating system, when you install Wireshark.. See http://www.wireshark.org/download.html for a list of supported operating systems and download sites

When you run the Wireshark program, the Wireshark graphical user interface shown in Figure 2 will be displayed. Initially, no data will be displayed in the various windows.

In this experiment you'll be running various network applications in different scenarios using a computer on your desk, at home, or in a lab. You'll observe the network protocols in your computer "in action," interacting and exchanging messages with protocol entities executing elsewhere in the Internet.

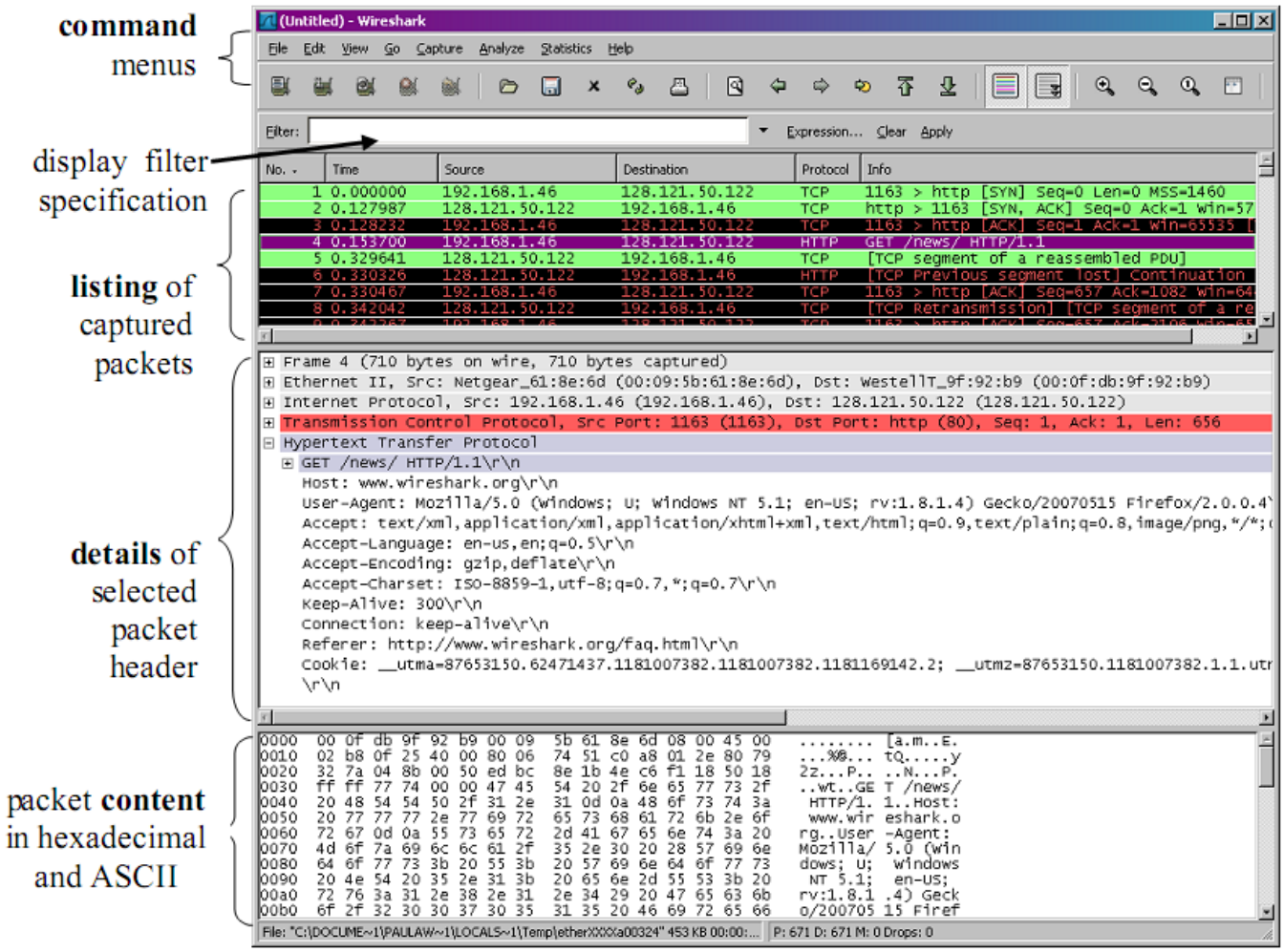

Figure 2: Wireshark Graphical User Interface

Q#1:
Can we make a packet analyzing on the data link layer? If yes, what are the purposes of packet sniffing under MAC layer?

PreLab:
Analyzing ICMP messages generated by the Traceroute program

Internet Control Message Protocol is part of the Internet Protocol Suite as defined in RFC 792. ICMP messages are typically generated in response to errors in IP datagrams (as specified in RFC 1122) or for diagnostic or routing purposes.

Traceroute program can be used to figure out the path a packet takes from source to destination. Traceroute is implemented in different ways in Unix/Linux and in Windows. In Unix/Linux, the source sends a series of UDP packets to the target destination using an unlikely destination port number; in Windows, the source sends a series of ICMP packets to the target destination. For both operating systems, the

program sends the first packet with TTL=1, the second packet with TTL=2, and so on. Recall that a router will decrement a packet's TTL value as the packet passes through the router. When a packet arrives at a router with TTL=1, the router sends an ICMP error packet back to the source. As a result of this behavior, a datagram with a TTL of 1 (sent by the host executing traceroute) will cause the router one hop away from the sender to send an ICMP TTL-exceeded message back to the sender; the datagram sent with a TTL of 2 will cause the router two hops away to send an ICMP message back to the sender; the datagram sent with a TTL of 3 will cause the router three hops away to send an ICMP message back to the sender; and so on. In this manner, the host executing traceroute can learn the identities of the routers between itself and destination X by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages. In this experiment use the native Windows tracert program.

Prelab Procedure:

1. Open the Windows Command Prompt application
2. Start up the Wireshark packet sniffer, and begin Wireshark packet capture.
3. In the MS-DOS command type either "tracert hostname" or "c:\windows\system32\tracert hostname" (without quotation marks), where hostname is a host on another continent. Then run the Traceroute program by typing return.
4. When the Traceroute program terminates, stop packet capture in Wireshark.

Questions:

Based on the result answer the following:

1. What is the IP address of your host? What is the IP address of the target Destination host?
2. If ICMP sent UDP packets instead (as in Unix/Linux), would the IP protocol number still be 01 for the probe packets? If not, what would it be?
3. Examine the ICMP error packet in your screenshot. It has more fields than the ICMP echo packet. What is included in those fields?
4. Examine the last three ICMP packets received by the source host. How are these packets different from the ICMP error packets? Why are they different?
5. Within the tracert measurements, is there a link whose delay is significantly longer than others? Is there a link whose delay is significantly longer than others? On the basis of the router names, can you guess the location of the two routers on the end of this link?

In Lab

Part A: IP (Internet Protocol)

In the first part of this laboratory experiment, we'll investigate the IP protocol, focusing on the IP datagram. We'll do so by analyzing a trace of IP datagrams sent and received by an execution of the traceroute program. We'll investigate the various fields in the IP datagram, and study IP fragmentation in detail.

Procedure:

We'll want to run traceroute and have it send datagrams of various lengths. The tracert program (used for our ICMP Wireshark lab) provided with Windows does not allow one to change the size of the ICMP echo request (ping) message sent by the tracert program. A nicer Windows traceroute program is pingplotter, available both in free version and shareware versions at http://www.pingplotter.com. The size of the ICMP echo request message can be explicitly set in pingplotter by selecting the menu item Edit-> Options->Packet Options and then filling in the Packet Size field. The default packet size is 56 bytes. Once pingplotter has sent a series of packets with the increasing TTL values, it restarts the sending process again with a TTL of 1, after waiting Trace Interval amount of time. The value of Trace Interval and the number of intervals can be explicitly set in pingplotter.

Do the following:

1. Start up Wireshark and begin packet capture (Capture->Option) and then press OK on the Wireshark Packet Capture Options screen (we'll not need to select any options here).

2. Start up pingplotter and enter the name of a target destination in the "Address to Trace Window." Enter 3 in the "# of times to Trace" field, so you don't gather too much data. Select the menu item Edit->Advanced Options->Packet Options and enter a value of 56 in the Packet Size field and then press OK. Then press the Trace button.

3. Next, send a set of datagrams with a longer length, by selecting Edit->Advanced Options->Packet Options and enter a value of 2000 in the Packet Size field and then press OK. Then press the Resume button.

4. Finally, send a set of datagrams with a longer length, by selecting Edit->Advanced Options->Packet Options and enter a value of 3500 in the Packet Size field and then press OK. Then press the Resume button.

5. Stop Wireshark tracing.

A look at the captured trace

In your trace, you should be able to see the series of ICMP Echo Request sent by your computer and the ICMP TTL-exceeded messages returned to your computer by the intermediate routers. Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. To print a packet, use File->Print, choose Selected packet only, choose Packet summary line, and select the minimum amount of packet detail that you need to answer the question.

1. Select the first ICMP Echo Request message sent by your computer, and expand the Internet Protocol part of the packet in the packet details window. What is the IP address of your computer?

2. Within the IP packet header, what is the value in the upper layer protocol field?

3. How many bytes are in the IP header? How many bytes are in the payload of the IP datagram? Explain how you determined the number of payload bytes.

4. Has this IP datagram been fragmented? Explain how you determined whether or not the datagram has been fragmented.

Next, sort the traced packets according to IP source address by clicking on the Source column header; a small downward pointing arrow should appear next to the word Source. If the arrow points up, click on the Source column header again. Select the first ICMP Echo Request message sent by your computer, and expand the Internet Protocol portion in the "details of selected packet header" window. In the "listing of captured packets" window, you should see all of the subsequent ICMP messages (perhaps with additional interspersed packets sent by other protocols running on your computer) below this first ICMP. Use the down arrow on your keyboard to move through the ICMP messages sent by your computer.

5. Which fields in the IP datagram always change from one datagram to the next within this series of ICMP messages sent by your computer?

6. Which fields stay constant? Which of the fields must stay constant? Which fields must change? Why?

7. Describe the pattern you see in the values in the Identification field of the IP datagram.

Next (with the packets still sorted by source address) find the series of ICMP TTL-exceeded replies sent to your computer by the nearest (first hop) router.

8. What is the value in the Identification field and the TTL field?

9. Do these values remain unchanged for all of the ICMP TTL-exceeded replies sent to your computer by the nearest (first hop) router? Why?

To study IP Fragmentation, sort the packet listing according to time again by clicking on the Time column.

10. Find the first ICMP Echo Request message that was sent by your computer after you changed the Packet Size in pingplotter to be 2000. Has that message been fragmented across more than one IP datagram?

11. Print out the first fragment of the fragmented IP datagram. What information in the IP header indicates that the datagram been fragmented? What information in the IP header indicates whether this is the first fragment versus a latter fragment? How long is this IP datagram?

12. Print out the second fragment of the fragmented IP datagram. What information in the IP header indicates that this is not the first datagram fragment? Are the more fragments? How can you tell?

13. What fields change in the IP header between the first and second fragment?

Now find the first ICMP Echo Request message that was sent by your computer after you changed the Packet Size in pingplotter to be 3500.

14. How many fragments were created from the original datagram?

15. What fields change in the IP header among the fragments?

Part B: TCP (Transport Control Protocol)

In this lab, we'll investigate the behaviour of TCP in detail. We'll do so by analyzing a trace of the TCP segments sent and received in transferring a 150KB file from your computer to a remote server. We'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer; we'll see TCP's congestion control

algorithm – slow start and congestion avoidance – in action; and we'll look at TCP's receiver-advertised flow control mechanism. We'll also briefly consider TCP connection setup and we'll investigate the performance (throughput and round-trip time) of the TCP connection between your computer and the server.

Capturing a bulk TCP transfer from your computer to a remote server

Before beginning our exploration of TCP, we'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer, and then transfer the file to a Web server using the HTTP POST method. We're using the POST method rather than the GET method as we'd like to transfer a large amount of data from your computer to another computer. Of course, we'll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.

Do the following:

1. Start up your web browser. Go the http://gaia.cs.umass.edu/wiresharklabs/alice.txt and retrieve an ASCII copy of *Alice in Wonderland*. Store this file somewhere on your computer.
2. Next go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html.
3. Use the *Browse* button in this form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland* (or do so manually). Don't yet press the "*Upload alice.txt file*" button.
4. Now start up Wireshark and begin packet capture *(Capture->Options)* and then press *OK* on the Wireshark Packet Capture Options screen (we'll not need to select any options here).
5. Returning to your browser, press the "*Upload alice.txt file*" button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
6. Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown below.

A first look at the captured trace

Before analyzing the behaviour of the TCP connection in detail, let's take a high level view of the trace. First, filter the packets displayed in the Wireshark window by entering "tcp" (lowercase, no quotes, and don't forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window.

What you should see is series of TCP and HTTP messages between your computer and gaia.cs.umass.edu. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message and a series of "HTTP Continuation" messages being sent from your computer to gaia.cs.umass.edu. You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

Answer the following questions, by opening the Wireshark captured packet file cpethereal-trace-1 in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip (that is download the trace and open that trace in Wireshark).

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details of the selected packet header window".
2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

If you have been able to create your own trace, answer the following question:

3. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?

Since this lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select Analyze->Enabled Protocols. Then uncheck the HTTP box and select OK.

TCP Basics

We will use the packet trace that you have captured (and/or the packet trace tcp-ethereal-trace-1 in http://gaia.cs.umass.edu/wireshark-labs/wiresharktraces.zip) to study TCP behaviour in the rest of this lab.

Answer the following questions for the TCP segments:

4. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?
5. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the

ACKnowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?

6. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command; you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.
7. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation all subsequent segments.

Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: Statistics->TCP Stream Graph->Round Trip Time Graph.

8. What is the length of each of the first six TCP segments?
9. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
10. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment
12. What is the throughput (bytes transferred per unit time) for the TCP connection?Explain how you calculated this value.

TCP congestion control in action

Let's now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we'll use one of Wireshark's TCP graphing utilities - Time-Sequence-Graph(Stevens) – to plot out data.

Select a TCP segment in the Wireshark's "listing of captured-packets" window. Then select the menu : Statistics->TCP Stream Graph-> Time-Sequence-Graph(Stevens).

In the graph, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.

Answer the following questions for the TCP segments the packet trace tcp-etherealtrace-1 in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip

13. Use the Time-Sequence-Graph (Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.
14. Answer each of two questions above for the trace that you have gathered when you transferred a file from your computer to gaia.cs.umass.edu