Assignment 6: Reliability and Transport Protocols

Due the week of October 22/23, 2015

Mechanisms to ensure the reliable transfer of information through a network exist in at least two levels of the OSI hierarchy: the data link layer and the transport layer. To appreciate the role each of these layers play in ensuring reliable transmission, I would like you to start by reading a paper entitled "End-to-End Arguments in System Design".

There are two components to any solution to the reliability problem: detecting errors and correcting them. We will save the study of encoding techniques designed to enable a receiver to detect and in some cases correct errors for later in the semester when we discuss other issues relevant to the data link layer. This week, however, we will look at how to correct errors by retransmitting packets, an issue relevant at both the data link layer and the transport layer. For background on these techniques, please being by reading §2.5 of Peterson and Davie.

Next, to learn about the transport layer in general and about retransmission techniques used by TCP, read the beginning of Chapter 5 of Peterson and Davie through §5.2.

Exercises

1. Certain variations in network protocols seem certain to improve performance. For example, the technique of "piggybacking" acknowledgments seems certain to improve performance by reducing the number of overhead bits transmitted.

   The following problem is taken from "Data Networks" by Bertsekas and Gallagher.

   > Consider a stop-and-wait system with two way traffic between nodes A and B. Assume that data frames are all of the same length and require D seconds for transmission. Acknowledgment frames require R seconds for transmission and there is a propagation delay P on the link. Assume that A and B both have an unending sequence of packets to send, assume that no transmission errors occur, and assume that the time-out interval at which a node resends a previously transmitted packet is very large. Finally, assume that each node sends new data packets and acknowledgments as fast as possible, subject to the rules of the stop-and-wait protocol.
   >
   > (a) Show that the rate at which packets are transmitted in each direction is $(D + R + 2P)^{-1}$. *Show that this is true whether or not the starting time between A and B is synchronized.*
   >
   > (b) Assume, in addition, that whenever a node has both an acknowledgment and a data packet to send, the acknowledgment is piggy-backed onto the data frame (which still requires D seconds). Assume that node B does not start transmitting data until the instant it receives the first frame from A (i.e. B's first frame contains a piggy-backed ACK). Show that all subsequent frames in each direction are data frames with a piggy-backed ACK. Show that the rate in each direction is now $(2D+2P)^{-1}$. Note that if $D > R$ this rate is less than that in part (a).

   In case you didn't notice, the conclusion of this problems seems to say that, contrary to intuition, piggy-backing of ACK's reduces throughput. Is this true in general, is the problem wrong, or is this interpretation of the result too general?

2. In their description of the sliding window protocol, Peterson and Davie suggest that the appropriate value for SWS is always the bandwidth-delay product while the RWS might be any value less than or equal to SWS. For this problem I would like you to explore situations in which smaller values for RWS might result in situations where setting SWS to a value smaller than the bandwidth delay product might make sense.

   One common value for RWS is 1. That is, the receiver buffers no packets, and, at any point, the only packet it will accept is a packet whose sequence number is one larger than the last packet received. When used in this way, the sliding window protocol is sometimes called GO-BACK-N because if a single packet is lost, the received will reject all subsequent packets until the sender times out and retransmits the lost packet. At this point, SWS packets (otherwise known as N packets) will have to be retransmitted.

   Assume that node A is sending packets to node B using GO-BACK-N over a link that fails to deliver packets (and acknowledgments) with probability p. Suppose that A has transmitted packets with sequence numbers 1 through N where N is small enough that A would not yet expect to have received ACKs for any of the packets sent. At this point, if SWS was large enough, the algorithm described in the text would just send packet $N + 1$. It is not clear that this is the best choice. Obviously (I hope), A's optimal strategy would be to send the packet that has the highest probability of being received and accepted by B. So,

   (a) Determine the probability that packet number $N + 1$ will be received and accepted by B if transmitted at this point (By "received and accepted" I mean that it gets there intact and that B doesn't discard it as a packet outside the range of sequence numbers it is currently willing to accept.).

   (b) Determine the probability that a repeated copy of packet number M (where $1 <= M <= N$) will be received and accepted by B if transmitted at this point.

   (c) Either argue that the value of the probability you determined for part a will always exceed that for part b (in which case A should never repeat packets before timing out) or give as simple a formula as you can expressing the relationship that must exist between M and N to justify sending packet M rather than $N + 1$.

   (d) Now, suppose that

   $$1 < N < RWS < \text{bandwidth-delay product measured in packets} <= SWS$$

   Would it ever make sense to retransmit packet number M (where $1 <= M <= N$) rather than sending $N + 1$ in this case? Assume that if a packet loss occurs, the receiver uses its buffers to hold the first RWS packets send after the missing packet.

3. Do problem 2.33 from Peterson and Davie.

4. Want to see some REAL packets?

   The program you used to examine packets on the panic machines, wireshark, is also available on the other Linux machine in our lab. Unfortunately, for the sake of other's privacy, it won't actually let you intercept packets with the ability you have to use "sudo" on the panic machines. Wireshark can, however, be used on these machines to examine the contents of packets intercepted earlier and stored in the appropriate format in a file.

   I have borrowed one such file (and the questions below) from an instructor at Cornell University. The file can be accessed using the path:

`~tom/shared/336/traces/trace2.cap`

To view this file, type

`wireshark ~tom/shared/336/traces/trace2.cap`

(or just start wireshark and open the file using the "File" menu). It shows the packets that formed a TCP connection used to transmit a stream of packets from a machine with address 132.236.227.44 to a machine with the address 132.236.227.44.

Please use the wireshark to examine the trace in this file and to answer the following questions about the TCP connection shown in the trace.

(a) Using the "TCP Stream Graph" item under the "Statistics" menu, look at the "Stevens" time sequence stream graph for the packets of the TCP stream. You should select the first packet of the TCP connection before selecting the menu item. Look at the stream of packets received in the first 20 ms (that is 0.02 seconds). Do you see any evidence of packets arriving out of order? Where?

(b) Look at the same graph and the first 100 ms. How many of the packets sent were not received the first time; can you point out which ones? What is the approximate value of TCP's timeout used to detect lost packets? How do you know?

(c) Look at the same graph and the first 250 ms. What is the size of the send window (in bytes) during the gap between the first and second flight of packets? How did you determine this?

5. Want to see even more real packets.

Several years ago, I used the Unix ftp client to transfer a compressed version of one of the trace files in ~tom/shared/336/traces to my machine at home and then back to bull again. While doing this pointless pair of file transfers, I was running a packet sniffer named tcpdump on both bull and my home system to capture all the packets sent between the two machines. The resulting trace files are named `ftpfrombull.cap` (the trace captured by the sniffer running on bull) and `ftpfromhome.cap` (the file captured at home).

These files can be examined using wireshark. The only odd feature is that my router does something called NAT (Network Address Translation) which translates IP addresses and port numbers used within my home network into different values in the real Internet. So, while my machine thinks it is 10.0.1.2, bull thinks it is 24.15.13.215.

The complete transcript of the ftp session I conducted while collecting these traces is shown below.

```
% ftp cs.williams.edu
Connected to cs.williams.edu.
220-
220-The Department of Computer Science at Williams College.
220-
220-
220 bull FTP server (Version wu-2.6.1(1) Thu Jul 26 14:54:18 EDT 2007) ready.
Name (cs.williams.edu:thomasmu): tom
331 Password required for tom.
Password:
230-
230-This is the Department of Computer Science at Williams College.
```

```
230-Please be aware that anonymous ftp is available through our
230-main server, and that daytime access should be kept to a minimum.
230-It is currently Wed Apr 16 17:04:01 2008 in Williamstown.
230-
230-E-mail labstaff@cs.williams.edu with any problems.
230-
230-
230 User tom logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd shared/336/traces
250 CWD command successful.
ftp> get trace1copy.cap.gz
local: trace1copy.cap.gz remote: trace1copy.cap.gz
500 'EPSV': command not understood.
227 Entering Passive Mode (137,165,8,2,252,136)
150 Opening BINARY mode data connection for trace1copy.cap.gz (40454 bytes).
100% |************************************| 40454      54.93 KB/s    00:00 ETA
226 Transfer complete.
40454 bytes received in 00:00 (48.48 KB/s)
ftp> ls
227 Entering Passive Mode (137,165,8,2,137,3)
150 Opening ASCII mode data connection for /bin/ls.
total 4544
-rw-r--r--  1 tom  CompSciFaculty     6173 Mar 30 00:25 ftptrace.cap
-rw-r--r--  1 tom  CompSciFaculty    13193 Mar 28 23:49 random.cap
-rw-r--r--  1 tom  CompSciFaculty   519855 Mar 28 15:15 smallhttp.cap
-rw-r--r--  1 tom  CompSciFaculty   195420 Mar 28 13:51 trace1.cap
-rw-r--r--  1 tom  CompSciFaculty    40454 Apr 16 17:03 trace1copy.cap.gz
-rw-r--r--  1 tom  CompSciFaculty    11384 Mar 28 13:51 trace2.cap
-rw-r--r--  1 tom  CompSciFaculty   344168 Mar 28 13:51 trace3.cap
-rw-r--r--  1 tom  CompSciFaculty    64184 Mar 28 13:51 trace4.cap
-rw-r--r--  1 tom  CompSciFaculty  1012004 Mar 28 13:51 trace5.cap
-rw-r--r--  1 tom  CompSciFaculty  1384204 Mar 28 13:51 trace6.cap
-rw-r--r--  1 tom  CompSciFaculty   383340 Mar 28 13:51 trace7.cap
-rw-r--r--  1 tom  CompSciFaculty      649 Mar 28 13:51 traces.txt
-rw-r--r--  1 tom  CompSciFaculty   593119 Apr  7 16:04 twoftps.cap
226 Transfer complete.
ftp> put trace1copy.cap.gz
local: trace1copy.cap.gz remote: trace1copy.cap.gz
227 Entering Passive Mode (137,165,8,2,178,240)
150 Opening BINARY mode data connection for trace1copy.cap.gz.
100% |************************************| 40454      63.75 KB/s    00:00 ETA
226 Transfer complete.
40454 bytes sent in 00:03 (12.59 KB/s)
ftp> quit
221-You have transferred 80908 bytes in 2 files.
221-Total traffic for this session was 83340 bytes in 3 transfers.
```

```
221-Thank you for using the FTP service on bull.
221 Goodbye.
```

Using wireshark, I would like you to determine:

(a) What was the first packet sent as a result of each line that I typed during the FTP session in the ftpfromhome file? (Use the packet numbers displayed on the leftmost column of the wireshark packet display in your answer. I am assuming you are familiar enough with the use of the ftp command to distinguish the lines I typed from the program's responses. This might be foolish since programs that provide nice GUI interfaces to FTP functionality have largely replaced the command line version. So, if you are not familiar with how to use FTP from the command line a) talk to me, or b) notice that except for the log in process, all lines I typed follow the prompt "ftp>".)

(b) How many TCP connections were made during this FTP session?

- Identify the first packet of each connection.
- Who (i.e. my machine or bull) created each of the TCP connections?
- Who closed each of the TCP connections? Identify the packet that initiated the process of closing each connection.

(c) The session involved two file transfers. How many data packets were sent during each of these connections? How many ACKs were sent?

(d) During the two file transfer connections, the ACKs are sent in packets with no data rather than being piggy-backed with data packets. Why? Can you find any examples of ACKs that acknowledge data that the receiver had not previously tried to acknowledge and that are piggy-backed with actual data? If so, identify the first such packet.

(e) What password did I enter while logging into the FTP server?

6. Answer question 5.4 from Peterson and Davie.