CS 440 – Fall 2005 Name: Key
HW 3 Dec. 2, 2005

5.5 When closing a TCP connection, why is the two-segment-lifetime timeout not necessary on the transition from LAST_ACK to CLOSED? (2)

On this transition, TCP is waiting to receive an ACK. When it finally does, it knows that there is no possibility that the other side would be retransmitting a FIN – the fact that it receives the ACK indicates that the other side has already sent its FIN and won't resend it – the other side is already to state **FIN_WAIT_1** or later, and it won't ever go back to a state that would cause it to resend the FIN before the connection is completely closed.

5.18 Diagnostic programs are commonly available that record the first 100 bytes, say of every TCP connection to a certain <host, port>. Outline what must be done with each received TCP packet, **P**, in order to determine if it contains data that belongs to the first 100 bytes of connection to host **HOST**, port **PORT**. Assume the IP header is **P.IPHEAD**, the TCP header is **P.TCPHEAD**, and header fields are named as in Figures 4.3 and 5.4. Hint: To get initial sequence numbers (ISNs) you will have to examine every packet with the **SYN** bit set. Ignore the fact that sequence numbers will eventually be reused. (2)

A program that is receiving TCP packets would need to keep a table of (source IP, source port, dest IP, dest port, starting sequence). It would get the source and destination IP addresses from **P.IPHEAD** and the source and destination ports from **P.TCPHEAD**. Any time it sees a packet with the **SYN** bit set in **P.TCPHEAD**, it should create the entry for that combination of source and destination addresses/ports and record the sequence number. Then any time it sees a TCP packet going between those packets that has a sequence number less than the starting value + 100, it knows that the packet contains data that belongs in the first 100 bytes of the connection between the two.

Any time it sees a packet being exchanged between these hosts/ports that has the **FIN** flag set in **P.TCPHEAD**, it can discard the entry from the table.

5.20 The Nagle algorithm, built into most TCP implementations, requires the sender to hold a partial segment's worth of data (even if **PUSH**ed) until either a full segment accumulates or the most recent outstanding **ACK** arrives. (3)

(a) Suppose the letters **abcdefghi** were sent, one per second, over a TCP connection with an RTT of 4.1 seconds. Draw a timeline indicating when each packet is sent and what it contains.

When the user types the first character, it will go into the buffer. Assuming there are no other segments in transit, the first packet, containing **a**, could be sent immediately. Now, assuming a segment size greater than 10 bytes, when the **b**, **c**, **d**, and **e** are entered, the buffer is not full, and there is an ACK still outstanding, so the characters would be buffered. The ACK would arrive shortly after the **e** was entered, so the next packet would be sent, containing **bcde**. There would again be an ACK outstanding, so **f**, **g**, **h**, and **i** would be buffered. When the ACK for the second packet arrived, the packet containing **fghi** would be sent.

(b) If the above were typed over a full-duplex telnet connection, what would the user see?

A full-duplex connection implies that the server would be echoing the characters back, so that the telnet client wouldn't print out anything that was typed, only what was received from the server. So the client would send in the segment containing the **a**, and the server would be able to immediately send the **a** back. However, the RTT is 4.1 sec, so the user wouldn't see the **a** echoed until after she had typed the **d**. Now the client would send in the **bcde** segment, and the same TCP packet could ACK the **a**. When the server receives this, it would have no outstanding ACKs, so it could immediately send the response segment containing **bcde**. But again, the user wouldn't see this response until after the **i** was typed. Finally, 4.1 sec. after typing the **i**, the user would see **fghi** echoed.

(c) Suppose that mouse position changes are being sent over the connection. Assuming that multiple position changes are sent each RTT, how would a user perceive the mouse motion with and without the Nagle algorithm?

Without the Nagle algorithm, each mouse position change could be sent over the network as soon as it was triggered; however, the user wouldn't see the updates to the cursor position from the server for 4.1 sec after each mouse movement. With Nagle's algorithm, multiple moves might be batched up and sent together, so the user would see jerkier position updates, where several of them might be executed suddenly after they had been batched up into a single segment.

5.27 Suppose, in TCP's adaptive retransmission mechanism, that **EstimatedRTT** is 90 at some point and subsequent measured RTTs are all 200. How long does it take before the **TimeOut** value, as calculated by the Jacobson/Karels algorithm, falls below 300? Assume initial **Deviation** value of 25; use $\delta = 1/8$. (2)

The formula says that

Difference = **SampleRTT** – **EstimatedRTT** = 200 – 90 = 110
EstimatedRTT = EstimatedRTT + (δ x **Difference**) = 90 + (110 / 8) = 103.75
Deviation = **Deviation +** δ(|**Difference**| **- Deviation**) = 25 + (110 – 25) / 8 = 35.625
TimeOut = μ x **EstimatedRTT** + φ x **Deviation** = 1 x 103.75 + 4 x 35.625 = 246.25

This assumes the typical values of $\mu = 1$ and $\varphi = 4$. Note that the initial **TimeOut** would have been 1 x 90 + 4 x 25 = 190, which was already below 300.

5.32 Consult *Request for Comments* 793 to find out how TCP is supposed to respond if a **FIN** or an **RST** arrives with a sequence number other than **NextByteExpected**. Consider both when the sequence number is within the receive window and when it is not. (3)

Assume the connection has already been established.

If a Reset is outside the receive window, it is ignored. If the Reset is within the receive window, even if it is not **NextByteExpected**, the connection is reset.

A Finish message is also ignored if it is outside the window – note that the finish is assumed to follow the last octet of the segment, so it is possible for some data in the segment to be accepted even though the FIN is ignored. If the FIN sequence number is not **NextByteExpected**, TCP must buffer the segment until the preceding segments are received before it can process the FIN.