

1. CS 164 - COMPUTER NETWORKS

2. Assignment 6 Answers

- Breakdown of points:

3. - 27% for 1st question (9% each part),
4. - 13% for 2nd question,
5. - 13% for the 3rd question,
6. - 20% for the 4th question (10% each part),
7. - 13% for the 5th question,
8. - 14% for the 6th question.

1.

The TFTP protocol is a reasonable model although with some idiosyncrasies that address other issues; see RFC 1350. TFTP's first packet, called Read Request, RRQ, simply names a file. Upon receipt, the server creates a new ephemeral port from which to answer, and begins sending data from that new port. The client assumes that the first well formed packet it receives is this server data, and records the data's source port. Any subsequent packets from a different port are discarded and an error response is sent. The basic stop and wait transfer is standard, although one must decide if sequence numbers are allowed to wrap around or not. Here are approaches, TFTP's and otherwise, for (a)(c):

(a) The most basic approach here is to require the server to keep track of connections, as long as they are active. The problem with this is that the client is likely to be simply an application, and can exit at any time. It may exit and retransmit a request for a different file, or a new request for the same file, before the server knows there was a problem or status change. A more robust mechanism for this situation might be a CONNECT NUM field, either chosen randomly or clock driven or incremented via some central file for each client connection attempt. Such a field would correspond roughly with TCP's ISN. In TFTP, if the RRQ is duplicated then the server might well create two processes and two ports from which to answer. (A server that attempted to do otherwise would have to maintain considerable state about past RRQ's.) Whichever process contacted the client first would win out, though, while the other would receive an error response from the client. In one sense, then, duplicate TFTP RRQ's do duplicate the connection, but only one of the duplicates survives.

(b) The TFTP approach here is to have the client enter a "dallying" period after the final data was received, so that the process is still around (perhaps moved to the background) to receive and reacknowledge any retransmissions of the final data. This period roughly corresponds to TIMEWAIT.

(c) The dallying approach of (b) also ties up the client socket for that period, preventing another incarnation of the connection. (However, TFTP has no requirement that dallying persist for a time interval approaching the MSL.) TFTP also specifies that both sides are to choose "random" port numbers for each connection (although "random" is generally interpreted as "assigned by the operating system"). If either side chooses a new port number, then latearriving packets don't interfere even if the other side reuses its previous port number. A CONNECT NUM field would also be effective here.

2.

The two segment life time timeout results from the need to purge old late duplicates, and uncertainty of the sender of the last ACK as to whether it was received. For the first issue we only need one connection endpoint in TIMEWAIT; for the second issue, a host in the LAST ACK state expects to receive the last ACK, rather than send it.

3.

Timeouts indicates that the network is congested and that one should send fewer packets rather than more. Exponential backoff immediately gives the network twice as long to deliver packets (though a single linear backoff would give the same); it also rapidly adjusts to even longer delays, thus it in theory readily accommodating sharp increases in RTT without further loading the already overtaxed routers. If the RTT suddenly jumps to 15 times the old TimeOut, exponential increase retransmits at $T=1, 3, 7, \text{ and } 15$; linear increase would retransmit at $T=1, 3, 6, 10, \text{ and } 15$. The difference here is not large. Exponential backoff makes the most difference when the RTT has increased by a very large amount, either due to congestion or network reconfiguration, or when “polling” the network to find the initial RTT.

4.

If every other packet is lost, we transmit each packet twice.

(a) Let E be the value for EstimatedRTT, and $T = 2 * E$ be the value for TimeOut. We lose the first packet and back off TimeOut to $2 * T$. Then, when the packet arrives, we resume with EstimatedRTT = E , TimeOut = T . In other words, TimeOut doesn't change.

(b) Let T be the value for TimeOut. When we transmit the packet the first time, it will be lost and we will wait time T . At this point we back off and retransmit using TimeOut = $2 * T$. The retransmission succeeds with an RTT of 1 sec, but we use the Backedoff value of $2 * T$ for the next TimeOut. In other words, TimeOut doubles with each received packet. This is Not Good.

5.

0.01/0.99

0.01/0.5

49/99

6.

(a) We can send multiple pkts at once and observe the RTTs. The second case is faster.

(b) Case1: congestion window size will decrease.

Case2: congestion window size will stay the same.