COSC 650 Project Fall 2014

This is a group project due on Thursday Dec 4 at 11:59 pm. Please upload a single zip file named groupnameProject containing only the files specified below (one zip file per group) to Blackboard. If multiple attempts are made, each attempt should have all the files for your group since only the last attempt will be graded. Please name the files as specified.

1. Use Wireshark to capture a DHCPv6 (Solicit XID) packet:

1.1 Include the actual output from Wireshark as a .pcap file named as groupnameQ1.pcap and give the Wireshark frame number of the DHCPv6 message. What is the reason for sending this message?

1.2 For each protocol that occurs in the message in 1.1, make a table listing the protocol and its header field names and include a brief (1-2 line) but clear description in the table that explains the general meaning of each header field in the protocol.

1.3 Give a brief (1-2 line) interpretation of the actual value for each header field in your packet that appears in the table (exclude unused values of zero (0) unless the zero value has some relevance). For example, if this had been a DNS query response message, within the “answers field” Wireshark may indicate the value of the “time to live subfield” as: 1 minute, 51 seconds. An interpretation of this value should indicate the actual hex value for the subfield (hex 00 00 00 6f), and say clearly what this particular value for “time to live” means in this case. The table and answers for 1.1-1.3 should be included in a file named groupnameQ1Results.

2. Write a Java socket program with a client and a server that satisfy the following conditions: whenever the user is expected to enter input, the program indicates the form of the input to be entered.

2.1 The client makes a UDP socket connection to the localhost server on port 12345, or TCP socket connections to a set of servers on port 80. The client user enters either localhost, or a list of Web server names with one name per line, and the last name followed by a period. For example, the user may enter:

www.towson.edu

www.google.com.

2.2 If a list of Web server names is entered, the client makes GET requests concurrently to each Web server on port 80, and prints the text of the received pages (preceded by the respective server’s name) in the order that the requests are completed. Only one GET request is made to each server. It is in the form: GET http://servername (for example, GET http://www.towson.edu, GET http:www.acm.org, and GET http://www.google.com).

2.3 If the server is localhost, the client connects to the server, which listens on UDP port 12345. The client user enters a single line with the full path of a file name of a text file (for example, c:\pathxyz\test.txt) file name of a text file to be retrieved from the server using the UDP socket, and then enters a timeout period Tc in milliseconds. The client sends the filename and the UDP port number (on which it will receive the file) to the server. The client waits for the packets containing the bytes of the file to arrive over UDP. The first packet has the file size S (<65536) in bytes of the text file. When the client receives the first packet, it prints the file size.

2.3.1 The localhost server reads the timeout Ts in milliseconds, which is specified by the server user at server startup. It then listens on UDP port 12345. When the client connects, it sends the file name of the text file; the server reads the file name and sends the file size S (<65536) to the client. It then sends the bytes of the file in packets over UDP. Each packet except the last has a fixed number of bytes B from the file. The value of B (<1024) is specified by the server user at server startup; the last packet has the remaining bytes (if any) in the file. Each packet also contains the number of the first byte in the packet (the bytes in the file are numbered sequentially from 1 to S), and the packet size P (P=B for all packets except the last packet; P≤B for the last packet).

2.3.2 If the whole file is received before the timeout Tc, the client sends an ack saying that the last byte received is S, and prints “file received” followed by the contents of the file. It then closes the UDP socket. If only some of the bytes were received when the timeout Tc occurs (or no bytes were received), the client sends a cumulative ack specifying the largest byte X that was received in order (this means that all bytes ≤ X were received; if no bytes were received in order, X=0). In this case, any bytes received out of order are stored, and the client prints “waiting”. The client then resets the timer Tc and repeats step 2.3.2.

2.3.3 The server reads the timeout Ts in milliseconds, which is specified by the server user at server startup. If an ack arrives before the timeout Ts, the server checks the value of the last byte received. If it is S, the server prints “file sent” and closes the UDP socket. If it is X, where X is less than S, the server retransmits bytes X+1, … , S, and prints “sending”. The server then resets the timer Ts and repeats step 2.3.3. If no ack arrives before the timeout Ts, the server retransmits all the packets of the file, and prints “resending”. The server then repeats step 2.3.3.

Test the program to verify that it works correctly. Upload the Java source code in files named: groupnameQ2Server.java and groupnameQ2Client.java.