Question 1. Suppose that five computers named A, B, C, D, and E are waiting for a transmission to finish on an Ethernet. All transmit at once when the previous packet is finished and collide.

Table 1 on the last page shows a partial history of how these five computers might contend to obtain access to the network after this collision. In constructing this table, we have deliberately ignored variability in collision detection times so that if two computers choose the same random number when deciding to delay their next attempt to transmit, then they will begin their next transmissions at exactly the same time. We have also assumed the amount of time consumed by a collision is exactly equal to the time a computer will pause before transmitting if it randomly decides to wait for one slot.

In the table, we indicate which of computers A through D attempted to transmit in each time slot by showing the range of back-off times (e.g., “0..3”) from which each computer that is involved in a collision must choose its next back-off time and the random back-off value it actually selected (e.g., “:2”). We left the entries in the table for computer E empty.

Fill in the column for computer E in a way that minimizes the number of times E attempts to transmit under the assumption that no station successfully transmits a packet until round 8 when computer C transmits alone. Show which slots E transmits in by indicating the range of back-off values it must choose between and the back-off value it selects after each collision in which it is involved. There are several possible solutions. You need only provide one. If you need an extra copy of the table for scrap work, you can access the PDF for this assignment on the course web page.

Question 2. When a network tries to deliver a packet, the efficiency of the process can be measured by dividing the time spent actually sending the bits of the packet by the total time from when the system starts trying to send the packet to the point when the last bit is sent. If no time is wasted, the efficiency will be 1. If something delays the beginning of the transmission of the packet, the efficiency will be:

\[
\frac{\text{time\_spent\_sending}}{\text{length\_of\_delay} + \text{time\_spent\_sending}}
\]

In Ethernet, the main cause of inefficient transmission is collisions. If a computer encounters one or more collisions before it manages to send a packet successfully, then the time spent resolving the collisions is a non-zero delay and the efficiency will be less than one. Note that we talk about the time spent sending a packet rather than the time spent delivering a packet. That is, we are only considering the time required to transmit the packet. This does not include the time required for the packet to travel through the cable to its destination.

Suppose that on some Ethernet, two computers, A and B, both have a packet to send. Assume that A starts sending before B, but not long enough before B to avoid a collision. Also assume that after detecting the collision, A attempts to send its packet again as soon as it detects that the network is again idle and that nothing collides with this second attempt allowing A to deliver its packet successfully. For the following questions assume the network is 1 kilometer long, computers A and B are at opposite ends of the network, that the packet is 1000 bits long and the data rate is 10 million bits per second and that signals can travel at the speed of light (3 × 10^8 meters/sec).

(a) What is the time spent sending the packet? Provide both a formula and a numeric answer to this and parts (b) and (c).

(b) What is the delay caused by a single collision?

(c) What is the efficiency if exactly one collision occurs as described before A transmits successfully?
Question 3. On the next page, you will find an abridged definition of a class named XMPPClient designed to provide a GUI for chatting similar to what you implemented in Lab 4. Since you should be familiar with such a class, we have only included the code to declare relevant instance variables and enough of the buttonClicked method to let you see how this client reacts when the send button is clicked.

On the page after the code for the XMPPClient, you will find a nearly complete definition of a second class named XMPPMessageStanza. As the XMPPStanza class is used in Lab 5 to encapsulate messages received from the chat server, the XMPPMessageStanza class is intended to encapsulate the message stanzas sent to an XMPP server when users send chat messages.

The XMPPMessageStanza class shown below is not quite complete. It is missing the declarations of its instance variables, local variables, and formal parameters. For this problem, we want you to add the missing declarations. You should be able to complete this task without really understanding what the class does, but to help you out, we provide a brief explanation at the end of this problem’s statement.

The names for which declarations must be provided are:

- updated
- symPos
- contents
- replacement
- source
- destination
- message
- original
- encoded
- to
- from
- msg

We have left extra blank lines or spaces at all the points in the code where the missing declarations could go. You will not need to fill in all of the blank spaces we have provided, but you should not need to place declarations anywhere outside these spaces. Each name should be declared as locally as possible.

The XMPPMessageStanza constructor expects parameter values that specify the essential details required to send an IM message, the JabberID of the person the message should be sent to, the full JabberID of the sender, and the text of the message. The toString method associated with the class then returns the stanza that should be sent to the XMPP server. That is, if you created a XMPPMessageStanza by saying

    stanza = new XMPPMessageStanza( "pinbawl@gmail.com", "fran@gmail.com/TCL2930", "Hi" );

you could later send the appropriate stanza to the server by saying

    toServer.out.println( stanza.toString() );
public class XMPPClient extends GUIManager {
    // Connection to the server
    private GTalkConnection toServer;

    // Buttons used to log in and out, and to send
    private JButton login = new JButton( "Login" );
    private JButton logout = new JButton( "Logout" );
    private JButton send = new JButton( "Send" );

    // Area in which incoming and outgoing chat messages are displayed
    private JTextArea convo = new JTextArea( 21, 50 );

    // Field used to enter messages to be sent
    private JTextField message = new JTextField( 30 );

    // Menu containing items for all accounts in the user’s roster.
    // Messages will be sent to the account selected in this menu
    private JComboBox roster = new JComboBox();

    // User’s base and full JabberIDs
    private String myJid, fullJid;

    public XMPPClient( ) {
        this.createWindow( WINDOW_WIDTH, WINDOW_HEIGHT );

        // Lotsof code to create the user interface would appear here
    }

    // Handles login, logout, and sending messages
    public void buttonClicked( JButton which ) {
        if ( which == login ) {
            // Create connection and set myJid and fullJid to correct values
        } else if ( which == send && roster.getSelectedItem().toString().contains("@" ) ) {
            // Send a message to the user selected in the roster menu
            String friend = roster.getSelectedItem().toString();
            XMPPMessageStanza outgoing =
                new XMPPMessageStanza( friend, fullJid, message.getText() );
            toServer.out.println( outgoing.toString() );

            // display the message in the user interface
            convo.append(myJid + " : " + message.getText() + "\n");
        } else if ( which == logout ) {
            // Send closing stanza to the server
        }
    }

    // The definitions of dataAvailable and connectionClosed would appear here
}
public class XMPPMessageStanza {

  /**
   * Create a message stanza given a message and its destination and source
   */
  public XMPPMessageStanza( ) {

    source = from;
    destination = to;
    contents = msg;
  }

  // Replace all occurrences of one string in the message with another
  private String encodeSymbol( ) {

    // Look for the first special symbol
    symPos = message.indexOf( original );
    updated = "";
    while ( symPos != -1 ) {
      // replace one occurrence of the special symbol
      updated = updated + message.substring(0, symPos ) + replacement;
      message = message.substring( symPos + original.length() );
      symPos = message.indexOf( original );
    }
    return updated + message;
  }

  // Return the text of a complete XML stanza that can be used to send the message
  public String toString( ) {

    encoded = this.encodeSymbol( contents, "&", "&amp;" );
    encoded = this.encodeSymbol( encoded, "<", "&lt;" );
    encoded = this.encodeSymbol( encoded, ">", "&gt;" );
    return "<message to=" + destination + 
      " from=" + source + 
      " type="chat" > + 
      "<body>" + encoded + "</body>" + 
      "</message>";
  }
}
<table>
<thead>
<tr>
<th>SLOT</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>backoff range 0.1: delay used: 0</td>
<td>backoff range 0.1: delay used: 1</td>
<td>backoff range 0.1: delay used: 1</td>
<td>backoff range 0.1: delay used: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delay used: 0</td>
<td>delay used: 1</td>
<td>delay used: 1</td>
<td>delay used: 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>backoff range 0.3: delay used: 3</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>backoff range 0.3: delay used: 3</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>backoff range 0.3: delay used: 3</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>backoff range 0.3: delay used: 3</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td>backoff range 0.3: delay used: 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>backoff range 0.7: delay used: 2</td>
<td>backoff range 0.7: delay used: 6</td>
<td>backoff range 0.7: delay used: 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>backoff range 0.7: delay used: 2</td>
<td>backoff range 0.7: delay used: 6</td>
<td>backoff range 0.7: delay used: 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>backoff range 0.7: delay used: 2</td>
<td>backoff range 0.7: delay used: 6</td>
<td>backoff range 0.7: delay used: 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>backoff range 0.7: delay used: 2</td>
<td>backoff range 0.7: delay used: 6</td>
<td>backoff range 0.7: delay used: 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C transmits alone!