Mini Project

Computer Communication Networks

Multi-Node Micro-Controller based Network for Implementation of Different MAC Protocols

Ammad Afsar	(09003)
Hafiz Maaz Ahmed	(09006)
Hafiz Muhammad Zaid Pirwani	(09007) [Group Leader]
Syeda Kulsoom Jafri	(09041)

Submitted To

Engr. Dr. Farah Haroon

Computer Communication Networks

Institute of Industrial Electronics Engineering, IIEE, PCSIR

St-22/C, Block-6, Gulshan-e-Iqbal, Karachi-75300.

Ph #: +92-21-34982353, Fax #: +92-21-34966274

www.iiee.edu.pk, www.ieee.org/iiee

Table of Contents

Mini Project Summary

The project comprises of designing and making a network of multiple micro-controllers, we are putting in 3 AVR micro-controllers for now but more may be added. Once the network is created, different MAC Protocols may be implemented by simply changing the firmware in the micro-controllers. This will allow for better understanding and hands-on learning of the various MAC Protocols.

By MAC Protocols we mean to implement Channel Access Method or Multiple Access Methods are required in such a network where there are more than 2 terminals, to be connected to the same multi-point transmission medium to transmit over it and to share its capacity.

Scope of Project

The scope of this project is designing of the hardware which will allow a network of micro-controllers to communicate with one another and allow students to have a better learning experience of Channel Access Methods with the hands on approach of programming micro-controllers to implement different MAC Protocols and analyzing the data on the main communication channel in real-time.

We have hence designed a circuit which allows 3 micro-controllers to communicate on a single channel (using a tri-state output state when channel not in use) and a computer side C# application which shows the channel data in real-time and can also be programmed to detect and display collisions, packet information and packet data of each node.

The hardware also includes an on-board Ejaaduino which is an AVR/Arduino Development Board and is used in this project as a USB-to-Serial Converter and as a programmer to program the micro-controllers on the board.

MAC Protocols

In the seven-layer OSI model of computer networking, media access control (MAC) data communication protocol is a sublayer of the data link layer, which itself is layer 2. The MAC sublayer provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multiple access network that incorporates a shared medium, e.g. Ethernet. The hardware that implements the MAC is referred to as a medium access controller.

The most widespread multiple access protocol is the contention based CSMA/CD protocol used in Ethernet networks. Examples of other common packet mode multiple access protocols are:

- CSMA/CD (used in Ethernet and IEEE 802.3)
- Token bus (IEEE 802.4)
- CSMA/CA (used in IEEE 802.11/WiFi WLANs)
- Slotted ALOHA, Reservation ALOHA (R-ALOHA), Mobile Slotted Aloha (MS-ALOHA)
- CDMA

Channel Access Method

In telecommunications and computer networks, a channel access method or multiple access method allows several terminals connected to the same multi-point transmission medium to transmit over it and to share its capacity. Examples of shared physical media are wireless networks, bus networks, ring networks, star networks and half-duplex point-to-point links.

A channel-access scheme is based on a multiplexing method, which allows several data streams or signals to share the same communication channel or physical medium. Multiplexing is in this context provided by the physical layer.

A channel-access scheme is also based on a multiple access protocol and control mechanism, also known as media access control (MAC). This protocol deals with issues such as addressing, assigning multiplex channels to different users, and avoiding collisions. The MAC-layer is a sub-layer in Layer 2 (Data Link Layer) of the OSI model and a component of the Link Layer of the TCP/IP model.

These are the four fundamental types of channel access schemes:

- Frequency Division Multiple Access (FDMA)
- Time division multiple access (TDMA)
- Packet mode multiple-access
- Code division multiple access (CDMA)/Spread spectrum multiple access (SSMA)

The network we have designed and this mini-project uses and implements Packet Mode Multiple Access.

Packet Mode Multiple-Access

Packet mode multiple-access is typically also based on time-domain multiplexing, but not in a cyclically repetitive frame structure, and therefore it is not considered as TDM or TDMA. Due to its random character it can be categorized as statistical multiplexing methods, making it possible to provide dynamic bandwidth allocation. This require a media access control (MAC) protocol, i.e. a principle for the nodes to take turns on the channel and to avoid collisions. Common examples are CSMA/CD, used in Ethernet bus networks and hub networks, and CSMA/CA, used in wireless networks such as IEEE 802.11.

The following are examples of packet mode channel access methods:

Contention based random multiple access methods

- Aloha
- Slotted Aloha
- Multiple Access with Collision Avoidance (MACA)
- Multiple Access with Collision Avoidance for Wireless (MACAW)
- Carrier sense multiple access (CSMA)
- Carrier sense multiple access with collision detection (CSMA/CD) - suitable for wired networks
- Carrier sense multiple access with collision avoidance (CSMA/CA) - suitable for wireless networks
- Distributed Coordination Function (DCF)
- Carrier sense multiple access with collision avoidance and Resolution using Priorities (CSMA/CARP)
- Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA) Based on constructive interference (CAN-bus)

Token passing:

- Token ring
- Token bus

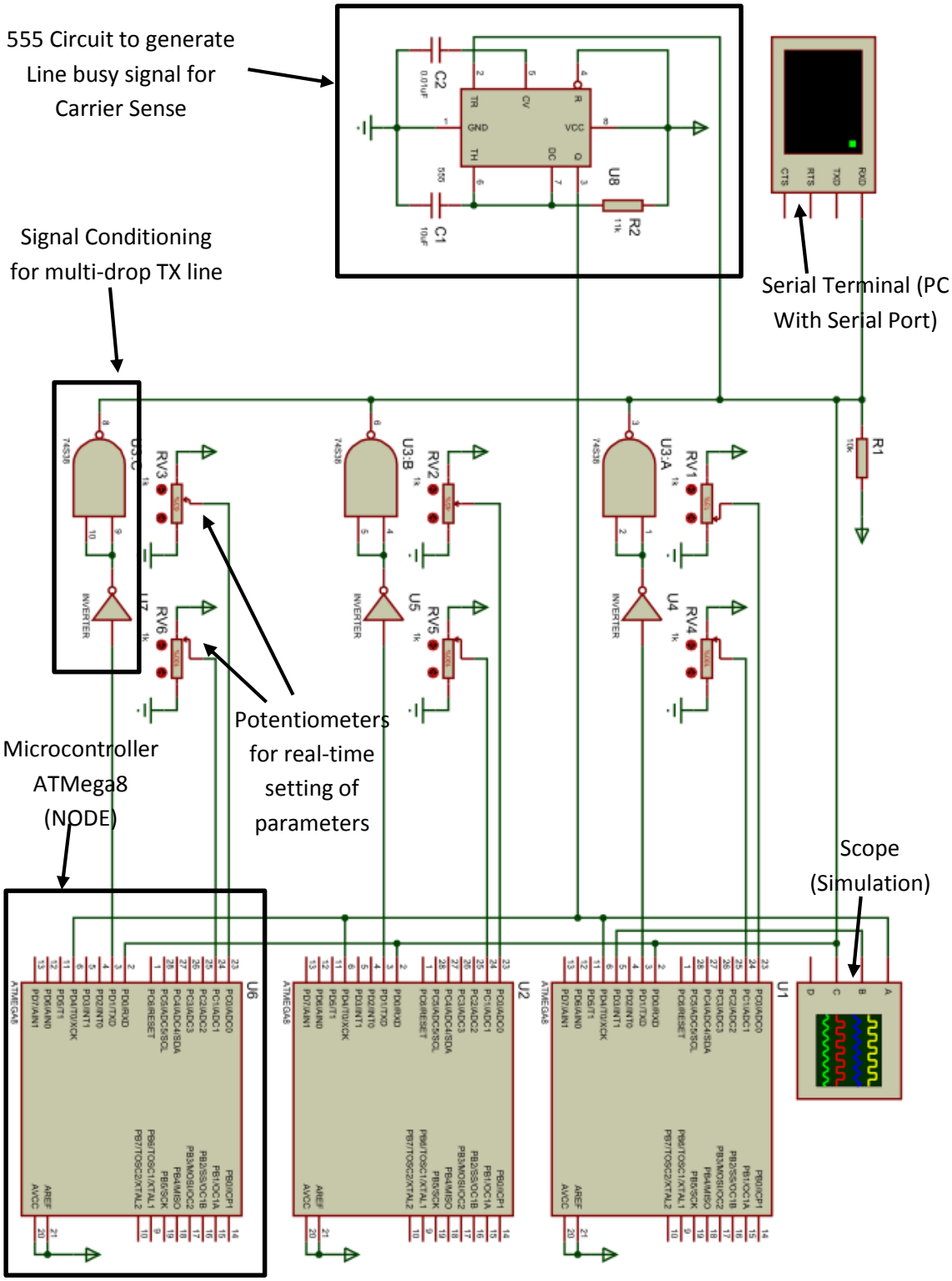
Polling

Resource reservation (scheduled) packet-mode protocols

- Dynamic Time Division Multiple Access (Dynamic TDMA)
- Packet reservation multiple access (PRMA)
- Reservation ALOHA (R-ALOHA)

Circuit Diagram

The circuit diagram of the network is shown below with 3 micro-controllers, connected via 2 UART lines (usually used for RS232). The TX from all controllers is signal conditioned and then combined and connected to the RX of all controllers and a Serial Terminal (for visualization of what is being sent).



Circuit Description

The circuit comprises of several components which can be described separately of one another.

Micro-controller

The micro-controller used as the nodes in the project is ATMega8 this is chosen because of our familiarity with this controller and also because we can easily program it with AVR C code or Arduino code which is easier for learning and hides most of the complexities involved with microcontroller programming, we have programmed the controllers with Arduino.

The controller has analog inputs for setting of various parameters in real time using potentiometers and also 14 digital IO lines and interrupt pins although most of these are not used in this project. The main components used are a few analog input and digital input/output pins and the Serial UART available in the controller providing the RX and TX lines.

Currently only a few pins of the micro-controller are in use:

- Digital Pin 0 is RX for receiving of Data, connected to the main TX Line
- Digital Pin 1 is TX for transmitting data, connected to main TX line after signal conditioning.
- Digital Pin 3 is used as an output pin for indicating Transmission in progress
- Digital Pin 4 is being used as pulled up input pin for Carrier Sense, the output off 555 IC is connected to this pin.
- Analog Pin 0 is being used for setting time between each transmission, 0-1023 milli seconds.
- Analog Pin 1 is being used to get a changing value to send in the transmission packet as data (can be some sensor data in a real-world scenario

Line Busy Signal (555 Circuit)

To implement Carrier Sense and to avoid conflicts a signal indicating TX line is busy is required, as while transmitting the TX line is constantly fluctuating between high and low and cannot be checked easily, so we tied the TX line with the trigger of a mono-stable 555 circuit with time constant of 120ms, this allows ample time for all micro-controllers to check the Line Busy Signal and is long enough to remain in the same state for the complete length of a packet, which is integral for Carrier Sense implementation.

Signal Conditioning (for TX Line)

The UART in micro-controllers is designed to be used in a 2 node network or with signal conditioning for RS232/RS485/etc for 2 node and 2+ node networks. We wanted the TX of all micro-controllers to be connected together so that it makes a single communication channel. The solution was to use Open-Collector output buffers (in the form of NAND gate 74LS38) and to keep the signals in correct polarity an inverter (74LS04) is also used.

Normally the TX line of a micro-controller goes to the RX of the other not connected with the TX of the other micro-controller as that would create electrical shorts when the signal state of both is different (when one is transmitting and other is idle or both transmitting). Open Collector output allows us to avoid this electrical short as the TX line goes LOW when any single TX line is LOW and is high when all are HIGH.

Serial Terminal

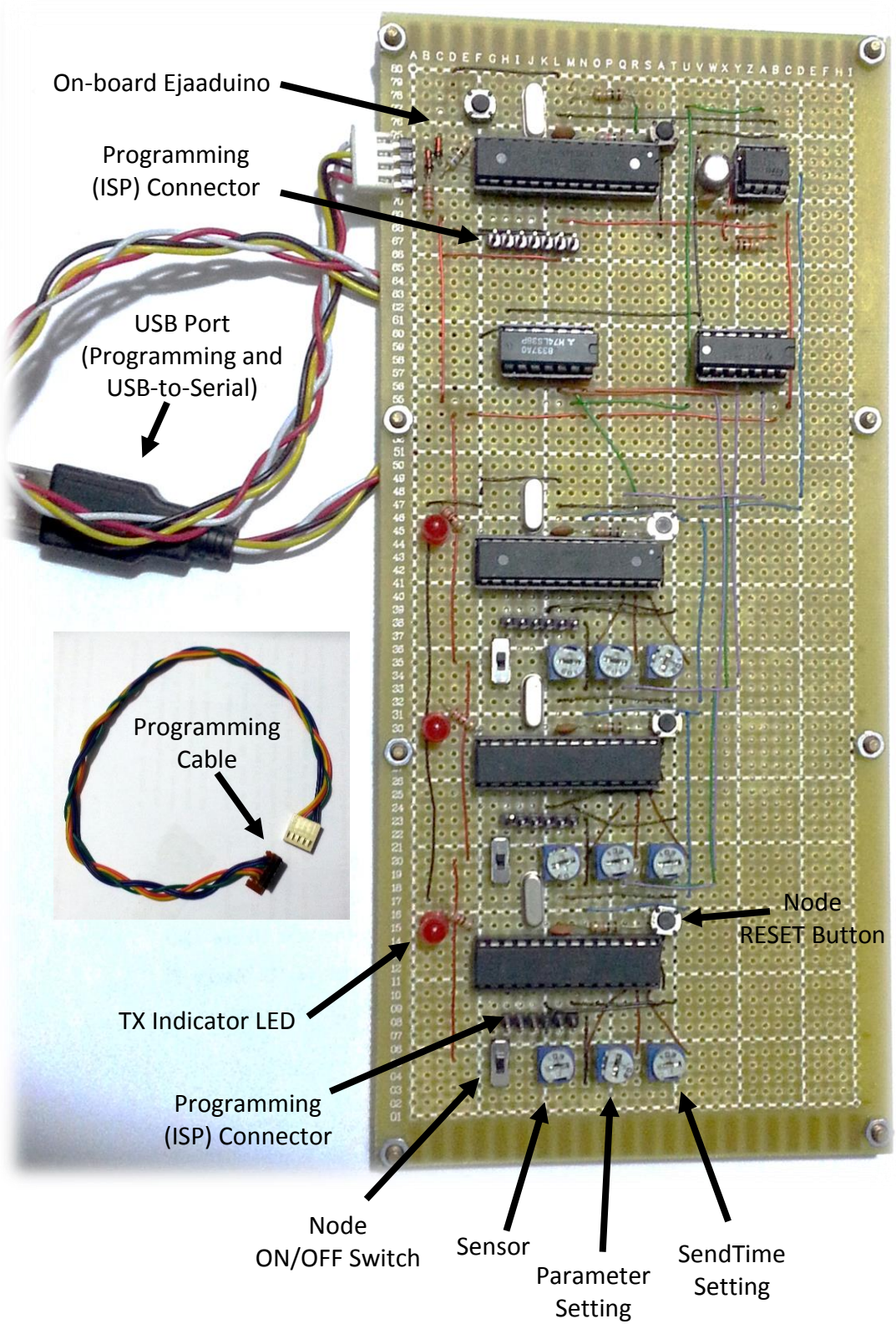
The Serial terminal in the schematic represents a PC with a Serial Port (or a USB to Serial Converter). This is required to see what is being transmitted on the TX line. Further use of this line can also be to send ACK packets or other information packets etc from the computer side. However, for the scope of this project we are only using the Serial Terminal as a way to observe the TX line.

As most PCs and laptops don't have a serial port, hence we have provided an on-board Ejaaduino (AVR/Arduino Development Board) which can work as the USB-to-Serial Converter and send the channel information to the PC as a serial port.

The on-board Ejaaduino also serves as the Programmer to program the 3 micro-controllers so that different algorithms may be tested. The procedure for implementing different algorithms and converting the on-board Ejaaduino from programmer to USB-to-Serial Converter is defined later in this report.

Finalized Circuit

The finalized circuit is shown below with all



Controller Software

The code/firmware inside the micro-controllers can be written for several MAC Protocols / Channel Access Methods. The Code is written in Arduino, which can be considered as an easier implementation of C language for AVR Micro-controllers.

The code we have provided simply serves as a template for writing of proper MAC protocols, ours is a simple implementation of 1-persistant carrier sense, however, lost packets are not sent again.

We have kept the packet size and format same during all our simulations and testing to avoid having to deal with packets of various sizes. The channel access methods we have implemented are also Packet-Mode Multiple Access where usually a fixed packet size is used in transmission.

Packet Details

In order to provide a template for the data communication and to design the C# app, we are providing here a sample packet format which may be used as a base for further work. The transmission packet format details are:

Packet Length: 23 characters

Packet Structure: ↵000:000:0000:0000:↵

New Line ↵ (2 bytes)	NODE ID 000 (3 bytes)	Separator : (1 byte)	Packet Count 0000 (4 bytes)	Separator : (1 byte)

TX Time Value 0000 (4 bytes)	Separator : (1 byte)	Sensor Value 0000 (4 bytes)	Separator : (1 byte)	New Line ↵ (2 bytes)

Serial Transmission Details:

Data Rate: 2400bps
Start Bits: 1
Stop Bits: 1
Data Bits: 8
Hence each byte requires *10 bits* in total for transmission.

Total Bits in a Packet: 23x10 = 230 bits
Time for a Single Packet: 230/2400 = 95.83 milliseconds

Template Code (implementation of Simple Carrier Sense)

The code is mostly self-explanatory with comments where necessary, implements a very simple Carrier Sense based channel access method. Whenever the controller has new data to send, before sending of data the channel sense pin is checked. If it is HIGH, indicating Channel in use, the transmission is NOT started, whereas if the channel sense pin is LOW, indicating a free channel, the data is transmitted.

There is a simulated process delay for each node which is taken as analog input saved in the variable `sendValue`, and ranges from 3 milliseconds to 1023 milliseconds, the analog input range (10-bit) is directly used as the processing delay timer.

```
// Initialization Code, Runs only ONCE
// Variables and constant definitions
#define SLAVE 0
// SLAVE ID / Address

#define processDelay 3
// min processing delay in milli seconds

#define TxIndicator 8
// TX Indicator LED Pin Number - OUTPUT

#define CarrierSensePin 4
// Carrier Sense Pin Number - INPUT


#define SendTimePin A5
#define ParamPin A4
#define SensorPin A3

unsigned int sendValue=0;
// time to wait before SENDing next packet

unsigned int sensorValue=0;
// some changing value to send while Transmitting

unsigned long currentMillis=0, previousMillis = 0;
unsigned int counter=0;
```

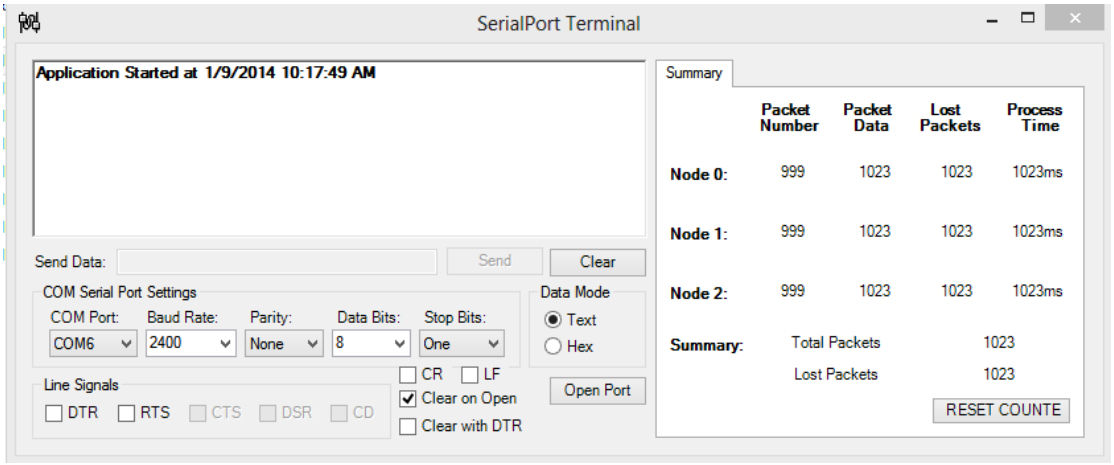
```
// setup() - Initializer function
// Runs only ONCE

void setup() {
  pinMode(TxIndicator, OUTPUT);
  // to show when TRANSMITTING
  pinMode(CarrierSensePin, INPUT);
  // to check for Carrier
  digitalWrite(CarrierSensePin, HIGH);
  // pulled-up input
  Serial.begin(2400);
  //2400 bits per second
}


// Main Loop function, loop()
// runs indefinitely after setup() finishes
// infinitely running loop

void loop() {
  delay(processDelay);
  // simulate processing delay on node

  currentMillis = millis();

  // get analog values from potentiometers
  sendValue = analogRead(SendTimePin);
  sensorValue = analogRead(SensorPin);

  if (counter>999)
    counter=0;
    // reset counter
```

```
  if(digitalRead(CarrierSensePin)==LOW){
    // CARRIER SENSE - checking state of 555 output pin
    if(currentMillis - previousMillis > sendValue) {
      previousMillis = currentMillis;
      // checking if time passed since last transmission
      // is greater than sendValue, for simulation of
      // additional processing delay. Min:3ms, Max:1023ms
      // Save last Send Time for next iteration
      digitalWrite(TxIndicator, HIGH);  // Tx Indicator
      // Print all information on the serial channel
      Serial.println("");
      Serial.print(SLAVE);
      Serial.print(SLAVE);
      Serial.print(SLAVE);
      Serial.print(":");
      Serial.print("0");
      if(counter<10)
        Serial.print("0");
      if(counter<100)
        Serial.print("0");
      Serial.print(counter++);
      Serial.print(":");
      if(sendValue<10)
        Serial.print("0");
      if(sendValue<100)
        Serial.print("0");
      if(sendValue<1000)
        Serial.print("0");
      Serial.print(sendValue);
      Serial.print(":");
      if(sensorValue<10)
        Serial.print("0");
      if(sensorValue<100)
        Serial.print("0");
      if(sensorValue<1000)
        Serial.print("0");
      Serial.print(sensorValue);
      Serial.println(":");
      Serial.flush();
      digitalWrite(TxIndicator, LOW);  // Tx Indicator
    }
  }
}
```

Computer Side App - C# Application

The software on the computer side is simply a Serial Port terminal, similar to hyper terminal software available in earlier versions of windows, however, we added the ability to monitor the incoming data and determine which packet it is and what is the data inside the package and which node sent it, as the sender node ID is included in the packet data.

A screenshot of the software is shown here



The portion on the left side is the serial terminal and has a basic terminal window to show the raw data, here the data packets and even collided packets can be seen. Below the window are the settings for the serial communication.

The portion on the right side is a table with data showing the current packet number received, the data in the packet and number of packets lost and the application determines a packet as lost if its number is not received. The processing time set on each node is also shown.

The C# app was downloaded with source as a simple serial terminal and the modification were done to make the right portion only. Hence the code for the serial terminal is omitted in this report and only the portions which are responsible for the right side are provided for reference.

The complete code can be seen online at

https://github.com/zaidpirwani/Serial-Terminal-CCN-Mini-Project

C# App Code (only parts of code shown)

```
// Variables and constant definitions
    int[] packetNumber = new int[3];
    int[] prevPacketNumber = new int[3];
    int[] packetData = new int[3];
    int[] lostPackets = new int[3];
    int[] packetTime = new int[3];
    bool startDetected = false;
    int nodeNo = -2;
    int packetIndex = 0;
    byte[] packet = new byte[23];


// UpdateLabels method called whenever there is
// new data incoming from serial port
private void updateLabels(string data)
{
  int z = 0;

  while (packetIndex == 0 && z < data.Length)
  {
      packet[packetIndex] = Convert.ToByte(data[z++]);
      if (packet[packetIndex] == '\r')
        packetIndex++;
  }

  while (packetIndex == 1 && z<data.Length)
  {
    packet[packetIndex] = Convert.ToByte(data[z++]); ;
    if (packet[packetIndex] == '\n')
      packetIndex++;
  }

  while (packetIndex > 1 && packetIndex < 23 && z <
data.Length)
  {
     packet[packetIndex] = Convert.ToByte(data[z++]); ;
     if(packet[2]!='\r' && packet[2]!='\n')
     packetIndex++;
  }
```

```
  if (packetIndex == 23)
  {
     if (packet[0] == '\r' && packet[1] == '\n' &&
packet[21] == '\r' && packet[22] == '\n')
       {
          if (packet[2] == '0' && packet[3] == '0' &&
packet[4] == '0')
            nodeNo = 0;
          else if (packet[2] == '1' && packet[3] == '1'
&& packet[4] == '1')
            nodeNo = 1;
          else if (packet[2] == '2' && packet[3] == '2'
&& packet[4] == '2')
            nodeNo = 2;

          if (nodeNo == 0 || nodeNo == 1 || nodeNo == 2)
            {
              packetNumber[nodeNo] = (packet[6] - 48) *
1000 + (packet[7] - 48) * 100 + (packet[8] - 48) * 10 +
(packet[9] - 48) * 1;
              packetTime[nodeNo] = (packet[11] - 48) *
1000 + (packet[12] - 48) * 100 + (packet[13] - 48) * 10 +
(packet[14] - 48) * 1;
              packetData[nodeNo] = (packet[16] - 48) *
1000 + (packet[17] - 48) * 100 + (packet[18] - 48) * 10 +
(packet[19] - 48) * 1;
              for (int tmp = 0; tmp < 23; tmp++)
                packet[tmp] = 0;
                packetIndex = 0;
                nodeNo = -2;
                if(packetNumber[0]==1){
                for (z = 0; z < 3; z++)
                {
                  packetNumber[z] = 0;
                  prevPacketNumber[z] = 0;
                }
              }
            }
          }
```

```
          else
          {
            for (int tmp = 0; tmp < 23; tmp++)
              packet[tmp] = 0;
            packetIndex = 0;
            nodeNo = -2;
          }
      }

      for (z = 0; z < 3; z++)
      {
       if (packetNumber[z] - prevPacketNumber[z] > 1)
        lostPackets[z] = packetNumber[z] -
prevPacketNumber[z];
        prevPacketNumber[z] = packetNumber[z];
       }

lbl_PcktNo_0.Invoke(new EventHandler( delegate{
lbl_PcktNo_0.Text = packetNumber[0].ToString();}));

lbl_PcktData_0.Invoke(new EventHandler( delegate {
lbl_PcktData_0.Text = packetData[0].ToString();}));

lbl_LostPckt_0.Invoke(new EventHandler( delegate {
lbl_LostPckt_0.Text = lostPackets[0].ToString();}));

lbl_PcktTime_0.Invoke(new EventHandler(delegate {
lbl_PcktTime_0.Text = packetTime[0].ToString();}));


lbl_PcktNo_1.Invoke(new EventHandler( delegate{
lbl_PcktNo_1.Text = packetNumber[1].ToString();}));

lbl_PcktData_1.Invoke(new EventHandler( delegate {
lbl_PcktData_1.Text = packetData[1].ToString();}));

lbl_LostPckt_1.Invoke(new EventHandler( delegate {
lbl_LostPckt_1.Text = lostPackets[1].ToString();}));

lbl_PcktTime_1.Invoke(new EventHandler(delegate {
lbl_PcktTime_1.Text = packetTime[1].ToString();}));

lbl_PcktNo_1.Invoke(new EventHandler( delegate{
lbl_PcktNo_1.Text = packetNumber[1].ToString();}));
```

```
lbl_PcktData_2.Invoke(new EventHandler( delegate {
lbl_PcktData_2.Text = packetData[2].ToString();}));

lbl_LostPckt_2.Invoke(new EventHandler( delegate {
lbl_LostPckt_2.Text = lostPackets[2].ToString();}));

lbl_PcktTime_2.Invoke(new EventHandler(delegate {
lbl_PcktTime_2.Text = packetTime[2].ToString();}));


lbl_TotalPckts.Invoke(new EventHandler(delegate {
lbl_TotalPckts.Text = (packetNumber[0] + packetNumber[1]
+ packetNumber[2] - lostPackets[0] + lostPackets[1] +
lostPackets[2]).ToString();}));

lbl_LostPckts.Invoke(new EventHandler(delegate {
lbl_LostPckts.Text = (lostPackets[0] + lostPackets[1] +
lostPackets[2]).ToString();}));

}
```