```
using System;
using System.Management;
using System.IO.Ports;
using System.Net;

namespace testProgram
{
    class MainClass
    {
        //Local Network Settings Struct
        public struct local_net
        {
            public string myMode, myHostname, myIPv4, mySubnet, myCIDR, myGateway;
        }

        static void Main()
        {
            local_net _net = new local_net();

            while (true)
            {
                get_info(ref _net);

                var portNames = SerialPort.GetPortNames();

                foreach (var port in portNames)
                {
                    try
                    {
                        Console.Clear();
                        Console.WriteLine("Configuration Dongle CONNECTED!");
                        Console.WriteLine("------------------------------------");
                        Console.WriteLine("Communicating w/ " + port + "\n");

                        //Setup COM Port
                        SerialPort serialPort = new SerialPort(port, 9600, Parity.None, 8, StopBits.One);
                        serialPort.DtrEnable = true;
                        serialPort.Open();

                        string rcv_data = "", rcv_ip = "", rcv_subnet = "";

                        //Read Request from Dongle
                        rcv_data = serialPort.ReadLine();

                        //Parse Request
                        if (rcv_data == "Mode\r")
                        {
                            serialPort.Write(_net.myMode);
                            rcv_data = "";
                        }

                        else if (rcv_data == "Host\r")
                        {
                            serialPort.Write(_net.myHostname);
                            rcv_data = "";
                        }

                        else if (rcv_data == "IP\r")
                        {
                            serialPort.Write(_net.myIPv4);
                            serialPort.DiscardOutBuffer();
                            serialPort.DiscardInBuffer();
                            rcv_data = "";
                        }

                        else if (rcv_data == "Sub\r")
                        {
                            serialPort.Write(_net.mySubnet);

                            serialPort.DiscardOutBuffer();
                            serialPort.DiscardInBuffer();
                            rcv_data = "";
                        }

                        else if (rcv_data == "Gate\r")
                        {
                            serialPort.Write(_net.myGateway);

                            serialPort.DiscardOutBuffer();
                            serialPort.DiscardInBuffer();
                            rcv_data = "";
                        }

                        else if (rcv_data == "DHCP\r")
                        {
                            serialPort.Write("OK");
                            setDHCP();
                            serialPort.Write("OK");

                            serialPort.DiscardOutBuffer();
                            serialPort.DiscardInBuffer();
                        }

                        else if (rcv_data == "Static\r")
                        {
                            serialPort.Write("OK");
                            rcv_ip = serialPort.ReadLine();
                            rcv_ip = rcv_ip.Replace("\r\n", "").Replace("\r", 
                      "").Replace("\n", "");

                            serialPort.Write("OK");
                            rcv_subnet = serialPort.ReadLine();
                            rcv_subnet = rcv_subnet.Replace("\r\n", "").Replace("\r", "").Replace("\n", "");

                            setStatic(rcv_ip, rcv_subnet);
                            serialPort.Write("OK");

                            serialPort.DiscardOutBuffer();
                            serialPort.DiscardInBuffer();
                        }

                        //Debugging purposes prints to console so I can see the info
                        Console.WriteLine("Hostname: " + _net.myHostname);
                        Console.WriteLine("IP: " + _net.myIPv4 + "/" + _net.myCIDR);
                        Console.WriteLine("Subnet: " + _net.mySubnet);
                        Console.WriteLine("Gateway: " + _net.myGateway + "\n");

                        //close Port
                        serialPort.Close();
                        break;
                    }
                    catch (Exception ex)
                    {
                        Console.Clear();
                        Console.WriteLine("Configuration Dongle DISCONNECTED!");
                        Console.WriteLine("------------------------------------");
                        Console.WriteLine("Error opening port " + port + ": {0}", ex.Message);
                    }
                }
            }
        }

        public static void get_info(ref local_net _net)
        {
            //Retrieve Hostname
            IPHostEntry hostInfo = Dns.GetHostEntry(Dns.GetHostName());
            _net.myHostname = hostInfo.HostName;

            //Retrieve IPv4 Address of Hostname
            IPAddress[] address = Dns.GetHostAddresses(Dns.GetHostName());

            //Set network settings struct values to NULL
            _net.myMode = "No Internet Access";
            _net.myIPv4 = address[1].ToString(); //Default IPv4 address 127.0.0.1 w/ no network connection
            _net.mySubnet = " ";
            _net.myCIDR = " ";
            _net.myGateway = " ";
            
            //Creating instance of ManagementClass for network adapter settings
            ManagementClass objMC = new ManagementClass("Win32_NetworkAdapterConfiguration");

            //Gets all the info for all the network adapters
            ManagementObjectCollection objMOC = objMC.GetInstances();

            //Parse through the info to find the network adapter with Network connection
            foreach (ManagementObject objMO in objMOC)
            {
                //If an IP exists in one of the adapters, then that's the active network we are working with
                if ((bool)objMO["IPEnabled"])
                {
                    try
                    {
                        //Get Mode, IPv4 Address, Subnet Mask, Gateway
                        string mode = ((bool)objMO["DHCPEnabled"]).ToString().ToLower() == "true" ? "DHCP" : "Static";
                        string[] ipaddress = (string[])objMO["IPAddress"];
                        string[] subnet = (string[])objMO["IPSubnet"];
                        string[] gateway = (string[])objMO["DefaultIPGateway"];

                        //Assign struct values w/ the values retrieved from code above
                        _net.myMode = mode;
                        _net.myIPv4 = ipaddress[0];
                        _net.mySubnet = subnet[0];

                        //Assign Gateway w/ a try function since gateway is optional during STATIC mode
                        try
                        {   if (gateway == null)
                                _net.myGateway = "unavailable";
                            else
                                _net.myGateway = gateway[0];
                        }
                        catch (Exception)
                        {
                            _net.myGateway = "unavailable";
                            throw;
                        }

                        //Calculate CIDR from Subnet
                        string[] tokens = _net.mySubnet.Split('.');
                        string result = "";
                        foreach (string token in tokens)
                        {
                            int tokenNum = int.Parse(token);
                            string octet = Convert.ToString(tokenNum, 2);
                            while (octet.Length < 8)
                                octet = octet + '0';
                            result += octet;
                        }

                        //Assign CIDR struct
                        _net.myCIDR = (result.LastIndexOf('1') + 1).ToString();

                    }
                    catch (Exception)
                    {
                        throw;
                    }
                }
            }
        }

        public static void setDHCP()
        {
            //Creating instance of ManagementClass for network adapter settings
            ManagementClass objMC = new ManagementClass("Win32_NetworkAdapterConfiguration");
            //Gets all the info for all the network adapters
            ManagementObjectCollection objMOC = objMC.GetInstances();

            //Parse through the info to find the network adapter with Network connection
            //If an IP exists in one of the adapters, then that's the active network we are working with
            foreach (ManagementObject objMO in objMOC)
            {
                if ((bool)objMO["IPEnabled"])
                {
                    try
                    {
                        //Enable DHCP
                        var ndns = objMO.GetMethodParameters("SetDNSServerSearchOrder");
                        ndns["DNSServerSearchOrder"] = null;
                        objMO.InvokeMethod("EnableDHCP", null, null);
                        objMO.InvokeMethod("SetDNSServerSearchOrder", ndns, null);
                    }
                    catch (Exception)
                    {
                        throw;
                    }
                }
            }
        }

        public static void setStatic(string ip_address, string subnet_mask)
        {
            //Creating instance of ManagementClass for network adapter settings
            ManagementClass objMC = new ManagementClass("Win32_NetworkAdapterConfiguration");
            //Gets all the info for all the network adapters
            ManagementObjectCollection objMOC = objMC.GetInstances();

            //Parse through the info to find the network adapter with Network connection
            //If an IP exists in one of the adapters, then that's the active network we are working with
            foreach (ManagementObject objMO in objMOC)
            {
                if ((bool)objMO["IPEnabled"])
                {
                    try
                    {
                        ManagementBaseObject setIP;
                        ManagementBaseObject newIP = objMO.GetMethodParameters("EnableStatic");

                        //Set IPv4 Address and Netmask recieved from Configuration Dongle
                        newIP["IPAddress"] = new string[] { ip_address };
                        newIP["SubnetMask"] = new string[] { subnet_mask };

                        //Enable Static Mode
                        setIP = objMO.InvokeMethod("EnableStatic", newIP, null);
                    }
                    catch (Exception)
                    {
                        throw;
                    }
                }
            }
        }

        public static UInt16 ModRTU_CRC(string buf, int len)
        {
            UInt16 crc = 0xFFFF;

            for (int pos = 0; pos < len; pos++)
            {
                crc ^= (UInt16)buf[pos];          // XOR byte into least sig. byte of crc

                for (int i = 8; i != 0; i--)
                {    // Loop over each bit
                    if ((crc & 0x0001) != 0)
                    {      // If the LSB is set
                        crc >>= 1;                    // Shift right and XOR 0xA001
                        crc ^= 0xA001;
                    }
                    else                                // Else LSB is not set
                        crc >>= 1;                        // Just shift right
                }
            }
            // Note, this number has low and high bytes swapped, so use it accordingly (or swap bytes)
            return crc;
        }
    }
}


```