Computer Systems Engineering Technology

CST 417 – Embedded Networking

Lab 3

Name: Dakota Kanner

Fall 2014

Due Date: 10/28/2014

Instructor: Troy Scevers

Instructions

In this lab we will be modifying the application from lab2. We will re-enable sections of the TCP/IP stack. We will remove the code for checking buttons from the main program. Answer the questions for each section and demo the sections that say demo to the lab instructor.

Procedure

Part 1

Open main_demo.h and find the following section and make it look as below.

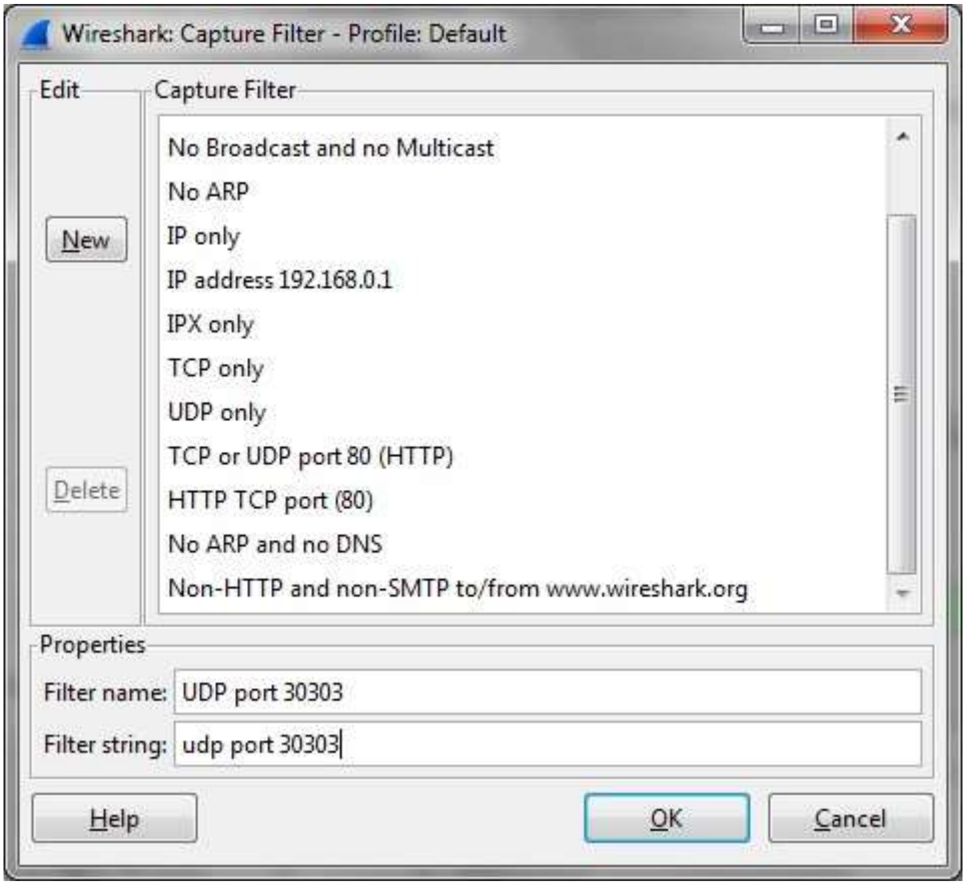
```
// enable the demo-applications that you want to run
#define APP_USE_GENERIC_TCP_CLIENT_DEMO
#define APP_USE_GENERIC_TCP_SERVER_DEMO
//#define APP_USE_GENERIC_SSL_CLIENT_DEMO
//#define APP_USE_SMTP_CLIENT_DEMO
//#define APP_USE_PING_DEMO
//#define APP_USE_SNMP_TRAP_DEMO
//#define APP_USE_SNMP_V2_TRAP_DEMO
//#define APP_USE_BERKELEY_API_DEMO
```

Next open tcpip_config.h and make it look like below.

```
// TCPIP Stack Module Selection
//   Uncomment or comment the following lines to enable or
//   disabled the following high-level application modules.
#define TCPIP_STACK_USE_IPV4
#define TCPIP_STACK_USE_ICMP_SERVER
//#define TCPIP_STACK_USE_HTTP2_SERVER
//#define TCPIP_STACK_USE_SSL_SERVER
//#define TCPIP_STACK_USE_SSL_CLIENT
#define TCPIP_STACK_USE_DHCP_CLIENT
//#define TCPIP_STACK_USE_SMTP_CLIENT
//#define TCPIP_STACK_USE_TELNET_SERVER
#define TCPIP_STACK_USE_ANNOUNCE
#define TCPIP_STACK_USE_DNS
#define TCPIP_STACK_USE_NBNS
//#define TCPIP_STACK_USE_REBOOT_SERVER
#define TCPIP_STACK_USE_SNTP_CLIENT
```

```
//#define TCPIP_STACK_USE_DYNAMICDNS_CLIENT
//#define TCPIP_STACK_USE_BERKELEY_API
//#define TCPIP_STACK_USE_IPV6
#define TCPIP_STACK_USE_TCP
#define TCPIP_STACK_USE_UDP
//#define TCPIP_STACK_USE_ZEROCONF_LINK_LOCAL
//#define TCPIP_STACK_USE_ZEROCONF_MDNS_SD
#define TCPIP_STACK_COMMAND_ENABLE
//#define TCPIP_STACK_USE_IPERF
//#define TCPIP_STACK_USE_SNMP_SERVER
//#define TCPIP_STACK_USE_SNMPV3_SERVER
//#define TCPIP_STACK_USE_FTP_SERVER
//#define TCPIP_STACK_USE_DHCP_SERVER
#define TCPIP_STACK_USE_ICMP_CLIENT
```

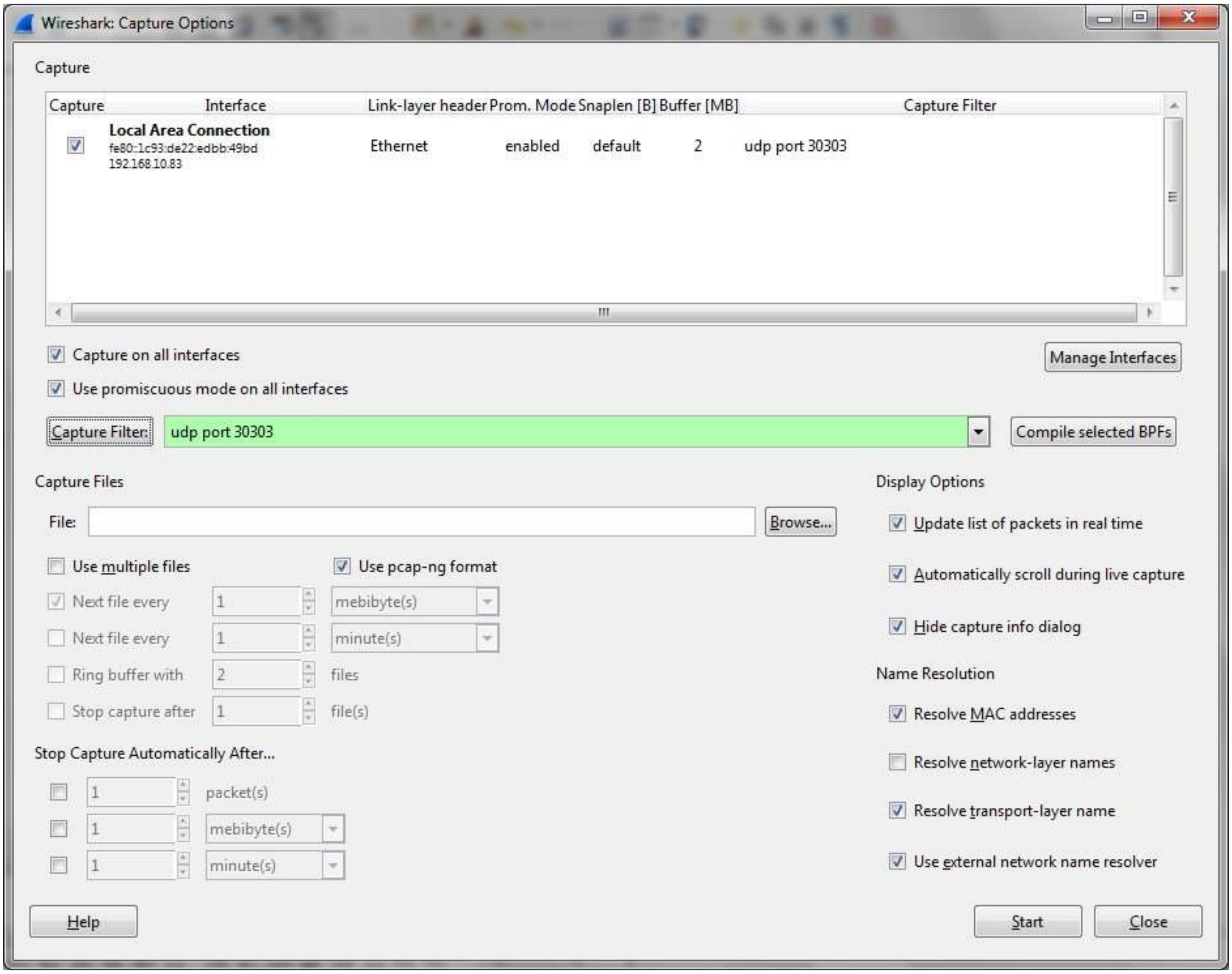
Recompile your project and run it on the PIC32 Ethernet starter kit. Next on your PC run wireshark. Once inside wireshark click on Capture->Options menu. Click on Capture Filter and create a new like below.


Wireshark: Capture Filter - Profile: Default
Edit
New
Delete
Capture Filter
No Broadcast and no Multicast
No ARP
IP only
IP address 192.168.0.1
IPX only
TCP only
UDP only
TCP or UDP port 80 (HTTP)
HTTP TCP port (80)
No ARP and no DNS
Non-HTTP and non-SMTP to/from www.wireshark.org
Properties
Help
OK
Cancel

Filter name: UDP port 30303

Filter string: udp port 30303

You should now have a screen that looks something like the following.



Wireshark: Capture Options

Capture

Capture	Interface	Link-layer header	Prom. Mode	Snaplen [B]	Buffer [MB]	Capture Filter
☑	**Local Area Connection** fe80::1c93:de22:edbb:49bd 192.168.10.83	Ethernet	enabled	default	2	udp port 30303

☑ Capture on all interfaces

☑ Use promiscuous mode on all interfaces

Capture Filter: udp port 30303

Capture Files

File:

□ Use multiple files ☑ Use pcap-ng format

☑ Next file every 1 mebibyte(s)

□ Next file every 1 minute(s)

□ Ring buffer with 2 files

□ Stop capture after 1 file(s)

Stop Capture Automatically After...

□ 1 packet(s)

□ 1 mebibyte(s)

□ 1 minute(s)

Display Options

☑ Update list of packets in real time

☑ Automatically scroll during live capture

☑ Hide capture info dialog

Name Resolution

☑ Resolve MAC addresses

□ Resolve network-layer names

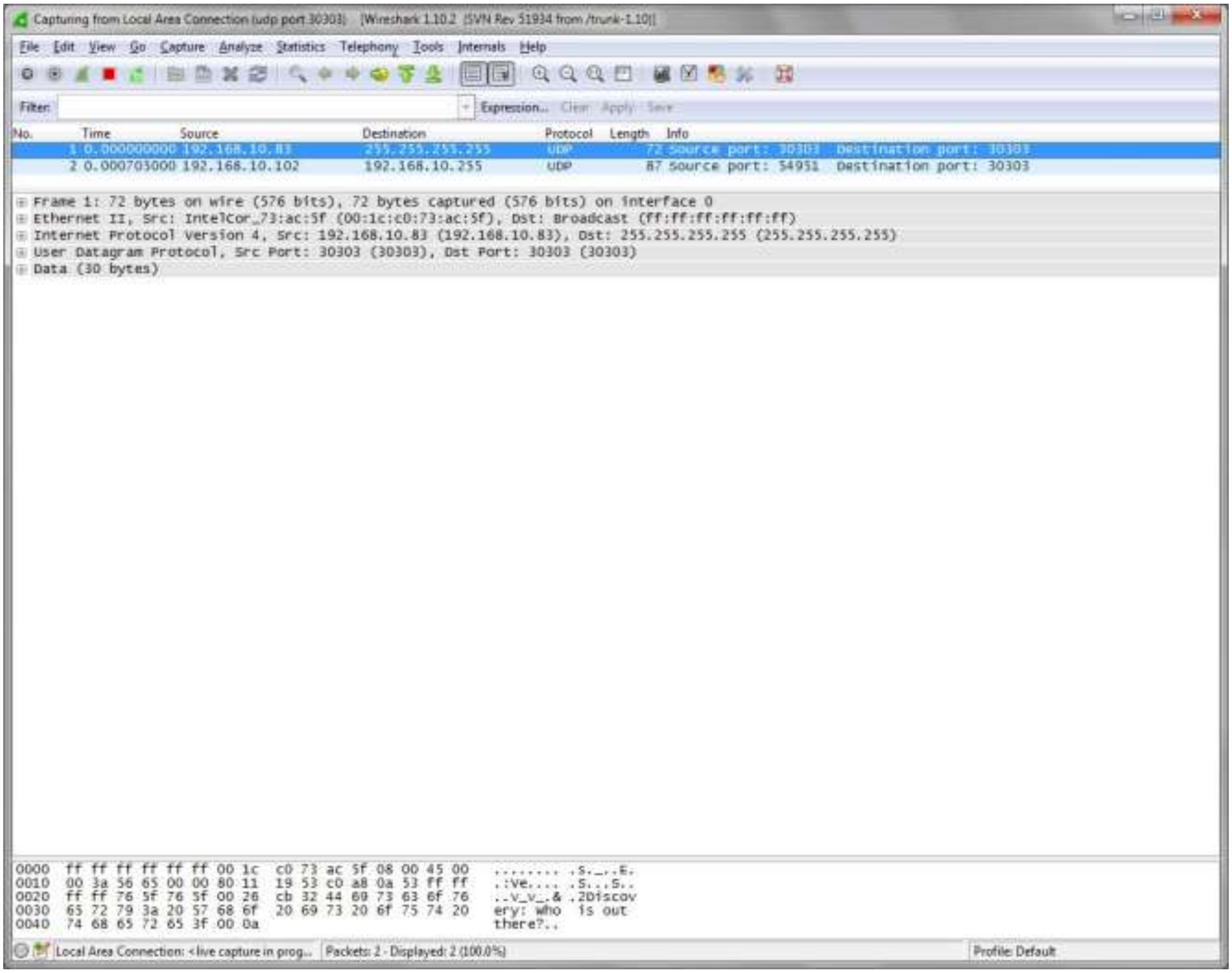
☑ Resolve transport-layer name

☑ Use external network name resolver

Click on the start button to start capturing packets.

Next start the tcpip_discoverer utility included in the microchip TCP/IP stack. It should be located in the directory microchip\tcpip\utilities.

Wireshark should capture two ethernet frames as shown below.


Capturing from Local Area Connection (udp port 30303)   [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
Profile: Default

```
No.     Time        Source            Destination       Protocol  Length  Info
      1 0.000000000 192.168.10.83     255.255.255.255   UDP           72  Source port: 30303  Destination port: 30303
      2 0.000703000 192.168.10.102    192.168.10.255    UDP           87  Source port: 54951  Destination port: 30303
```

```
⊞ Frame 1: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0
⊞ Ethernet II, Src: IntelCor_73:ac:5f (00:1c:c0:73:ac:5f), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol Version 4, Src: 192.168.10.83 (192.168.10.83), Dst: 255.255.255.255 (255.255.255.255)
⊞ User Datagram Protocol, Src Port: 30303 (30303), Dst Port: 30303 (30303)
⊞ Data (30 bytes)
```

```
0000  ff ff ff ff ff ff 00 1c  c0 73 ac 5f 08 00 45 00   ........ .s._..E.
0010  00 3a 56 65 00 00 80 11  19 53 c0 a8 0a 53 ff ff   .:Ve.... .S...S..
0020  ff ff 76 5f 76 5f 00 26  cb 32 44 69 73 63 6f 76   ..v_v_.& .2Discov
0030  65 72 79 3a 20 57 68 6f  20 69 73 20 6f 75 74 20   ery: Who  is out 
0040  74 68 65 72 65 3f 00 0a                            there?..
```

Local Area Connection: <live capture in prog...   Packets: 2 · Displayed: 2 (100.0%)

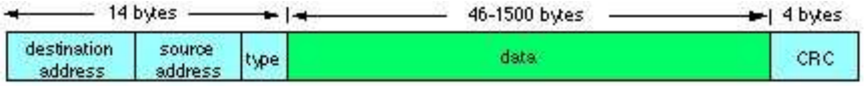
Answer the following questions about these captured frames.

1. What is the text message contained in your UDP packets?
    1. Sent from the PC?
       "Discovery: Who is out there?"
    2. Sent from the PIC?
       "PIC32INT MCHPDEMO_E"

2. What is the destination IP address of the UDP packet sent from the lab computer?
   255.255.255.255

3. What is the meaning of this destination IP address?

   It broadcasts to all devices on the LAN.



MAC encapsulation of a packet of data

4. Draw the Ethernet frame for both the PC and the PIC below filling in the fields with the actual transmitted data.

PC:

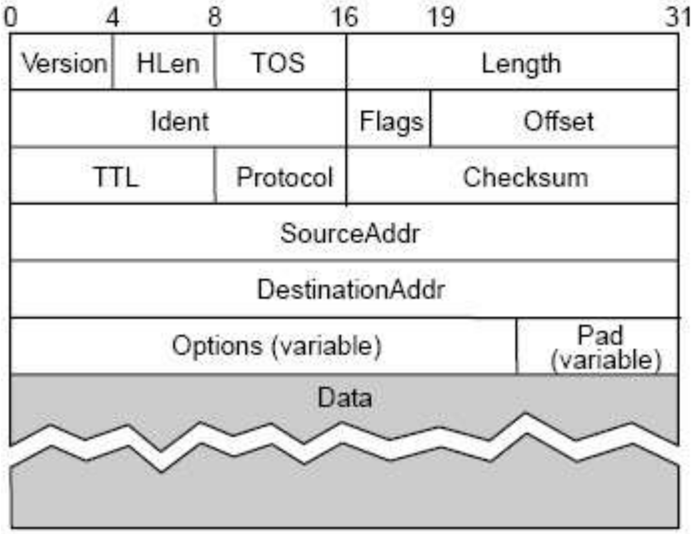
Dest: ff:ff:ff:ff:ff:ff	Source: 74:d0:2b:2c:30:b7	Type: 0x0800 (IP)

…	CRC: 0x000A

PIC:

Dest: ff:ff:ff:ff:ff:ff	Source: 00:04:a3:53:d6:c0	Type: 0x0800 (IP)

…	CRC: 0x0D0A

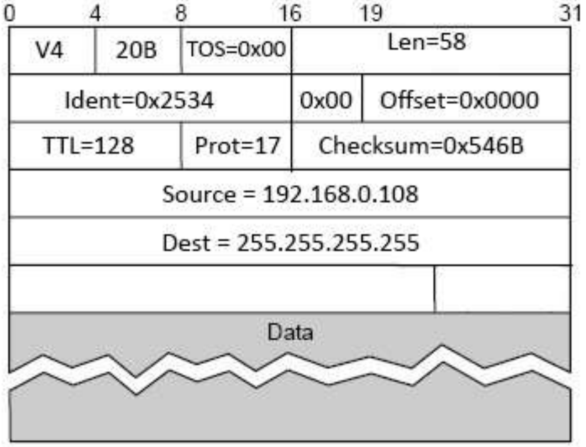


Version	HLen	TOS	Length	
Ident			Flags	Offset
TTL		Protocol	Checksum	
SourceAddr				
DestinationAddr				
Options (variable)				Pad (variable)
Data				

14 fields comprise the IP Datagram

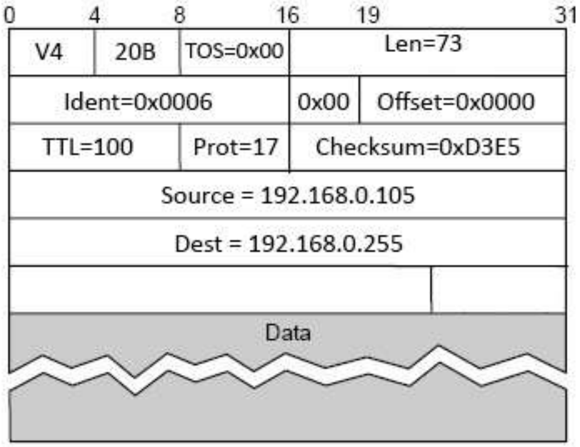
5. Draw the IP datagram for each the PC and the PIC filling in all fields with the actual transmitted data.

PC

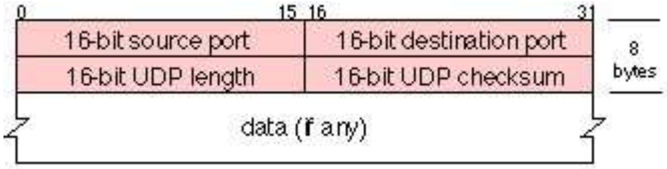


V4	20B	TOS=0x00	Len=58	
Ident=0x2534			0x00	Offset=0x0000
TTL=128		Prot=17	Checksum=0x546B	
Source = 192.168.0.108				
Dest = 255.255.255.255				
Data				

PIC



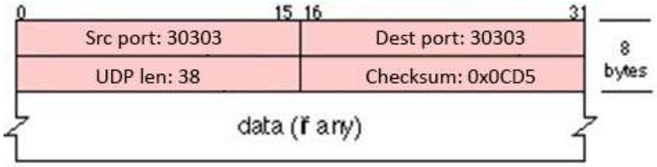
V4	20B	TOS=0x00	Len=73	
Ident=0x0006			0x00	Offset=0x0000
TTL=100		Prot=17	Checksum=0xD3E5	
Source = 192.168.0.105				
Dest = 192.168.0.255				
Data				



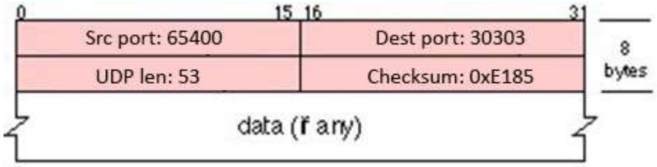
UDP Header format

6. Draw the UDP header for both the PC and the PIC filling in the values with the actual transmitted data (including actual data message sent).

PC



PIC



Part 2

Open up a command prompt on the Lab machine and type in **telnet 10.101.0.xxx 9760.** Where xxx is the ip of your board. Once connected start typing. You should see every character you type echoed back to you in uppercase (press the escape key to end session). This is performed by the program generic_tcp_server.c. Open this file and look at the code.

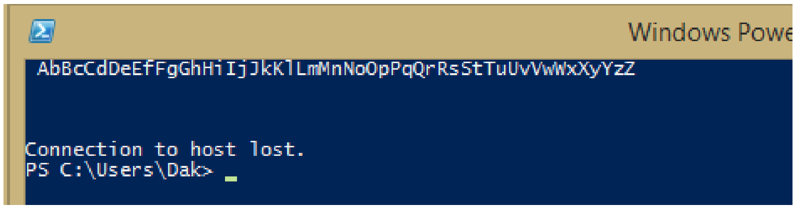
Now modify this code. When the user enters a w return your name. (Demo This to the Lab Instructor)

```
for(w = 0; w < wMaxGet; w += sizeof(AppBuffer))
{
  if(w + sizeof(AppBuffer) > wMaxGet)
      wCurrentChunk = wMaxGet - w;
  TCPGetArray(MySocket, AppBuffer, wCurrentChunk);

  for(w2 = 0; w2 < wCurrentChunk; w2++)
  {
    i = AppBuffer[w2];
    if(i == 'w')
    {
      AppBuffer[w2] = 'D';
      TCPPutArray(MySocket, AppBuffer, wCurrentChunk);
      AppBuffer[w2] = 'a';
      TCPPutArray(MySocket, AppBuffer, wCurrentChunk);
      AppBuffer[w2] = 'k';
    }
    else if(i >= 'a' && i <= 'z')
    {
      i -= ('a' - 'A');
      AppBuffer[w2] = i;
    }
    else if(i == '\e')   //escape
    {
      TCPServerState = SM_CLOSING;
    }
  }

  // Transfer the data out of our local processing buffer and into the TCP TX FIFO.
  TCPPutArray(MySocket, AppBuffer, wCurrentChunk);
}
```

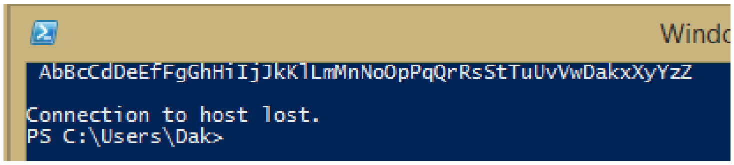
Before:



```
 AbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ


Connection to host lost.
PS C:\Users\Dak> _
```

After:



```
 AbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwDakxXyYzZ

Connection to host lost.
PS C:\Users\Dak>
```

Part 3

Open the file generic_tcp_client.c and convert all calls to *SYS_CONSOLE_MESSAGE()* to be *DBPRINTF().* Now compile and run your code. Make sure the DBPRINTF windows is open and then press the SW2 button. Observe the results and record them below. Play with this code. We will use the TCP_Server and TCP_Client as a basis for our next lab.

```
SM_HOME
SM_WAIT_DNS
…
SM_DNS_RESOLVED
SM_SOCKET_OBTAINED
…
SM_DISCONNECT
```