



## ABSTRACT

The TM4C129x MCU is a highly-connected 32-bit Arm® Cortex®-M4F microcontroller with integrated Ethernet MAC and PHY and a wide variety of wire communication peripherals. This application report demonstrates various Ethernet application examples based on the Lightweight IP (lwIP) stack - an open-source TCP/IP stack designed for embedded systems. The [EK-TM4C1294XL](#) LaunchPad™ Development Kit that the examples are run on is an evaluation platform for the TM4C129x Ethernet MCUs.

Project collateral and source code discussed in this document can be downloaded from the following URL: <https://www.ti.com/lit/zip/spna248>.

## Table of Contents

<b>1 Introduction</b> .....	3
1.1 License.....	3
1.2 lwIP Version.....	3
1.3 lwIP Customization.....	3
1.4 lwIP API.....	3
1.5 lwIP Development Home Page and Support .....	6
<b>2 Application Examples</b> .....	7
<b>3 Application Setup</b> .....	8
3.1 Hardware Setup.....	8
3.2 Software Tools.....	8
<b>4 Download and Import the Ethernet examples</b> .....	9
<b>5 Enet_tcpecho_server_lwip Example Overview</b> .....	13
5.1 Build and Flash the Program.....	13
5.2 Check and Program the MAC Address.....	13
5.3 Configure the Terminal Window.....	16
5.4 Run the enet_tcpecho_server_lwip Example.....	16
<b>6 Enet_tcpecho_server_static_ip_lwip Example Overview</b> .....	19
6.1 How to Configure lwIP for Static Address.....	19
6.2 Run the enet_tcpecho_server_static_ip_lwip Example.....	20
<b>7 Enet_udpecho_server_lwip Example Overview</b> .....	22
7.1 Run the enet_udpecho_server_lwip Example.....	22
<b>8 Enet_dns_lwip Example Overview</b> .....	24
8.1 How to Configure lwIP for DNS.....	24
8.2 How to View the DNS Traffic on Wireshark.....	25
8.3 Run the enet_dns_lwip Example.....	26
<b>9 Enet_sntp_lwip Example Overview</b> .....	27
9.1 Run the enet_sntp_lwip Example.....	27
<b>10 Enet_tcpecho_client_lwip Example Overview</b> .....	28
10.1 Configure the Server IP Address.....	28
10.2 Configure the SocketTest Server and Run the enet_tcpecho_client_lwip Example.....	28
10.3 Wireshark Capture for enet_tcpecho_client_lwip Example.....	30
<b>11 Enet_adcsensor_client_lwip Example Overview</b> .....	30
11.1 Run the adcsensor_client_lwip Example.....	30
<b>12 Enet_udpecho_client_lwip Example Overview</b> .....	31
12.1 Run the enet_udpecho_client_lwip Example.....	31
<b>13 References</b> .....	32

## List of Figures

Figure 1-1. TCP Client Server Communication Using Raw API.....	5
--	---

Figure 1-2. UDP Client Server Communication Using Raw API.....	6
Figure 3-1. Hardware Setup for the Application Examples.....	8
Figure 4-1. Import CCS Projects Step 1.....	9
Figure 4-2. Import CCS Projects Step 2.....	10
Figure 4-3. Import CCS Projects Step 3.....	11
Figure 4-4. Import CCS Projects Step 4.....	12
Figure 5-1. Debug CCS Projects.....	13
Figure 5-2. MAC Address Programming Using LM Flash Programmer.....	14
Figure 5-3. MAC Address Programming Using CCS.....	15
Figure 5-4. MAC Address Programming Using Uniflash.....	15
Figure 5-5. Serial Terminal Settings.....	16
Figure 5-6. Enet_tcpecho_server_lwip Output.....	16
Figure 5-7. SocketTest Client Configuration for Enet_tcpecho_server_lwip .....	17
Figure 5-8. Client to Server Wireshark Capture for Enet_tcpecho_server_lwip .....	17
Figure 5-9. Client Confirmation of Data Received for Enet_tcpecho_server_lwip .....	18
Figure 5-10. Server to Client Wireshark Capture for Enet_tcpecho_server_lwip .....	18
Figure 6-1. Enet_tcpecho_server_static_ip_lwip Output.....	20
Figure 6-2. SocketTest Client Configuration for Enet_tcpecho_server_static_ip_lwip .....	20
Figure 6-3. Client to Server Wireshark Capture for Enet_tcpecho_server_static_ip_lwip .....	21
Figure 6-4. Server to Client Wireshark Capture for Enet_tcpecho_server_static_ip_lwip .....	21
Figure 7-1. Query IP Address of the PC.....	22
Figure 7-2. Enet_udpecho_server_lwip Output.....	22
Figure 7-3. Client to Server Wireshark Capture for Enet_udpecho_server_lwip .....	23
Figure 7-4. Server to Client Wireshark Capture for Enet_udpecho_server_lwip .....	23
Figure 8-1. Port Mirroring.....	25
Figure 8-2. Enet_dns_lwip Output.....	26
Figure 8-3. Wireshark Capture for Enet_dns_lwip .....	26
Figure 9-1. Enet_sntp_lwip Output.....	27
Figure 9-2. Wireshark Capture for Enet_sntp_lwip .....	27
Figure 10-1. SocketTest Server Configuration for Enet_tcpecho_client_lwip.....	29
Figure 10-2. Enet_tcpecho_client_lwip Output.....	29
Figure 10-3. Client Server Wireshark Capture for Enet_tcpecho_client_lwip .....	30
Figure 11-1. Client to Server Wireshark Capture for Enet_adcSENSOR_lwip .....	30
Figure 12-1. Enet_udpecho_client_lwip Output.....	31
Figure 12-2. Client Server Wireshark Capture for Enet_udpecho_client_lwip .....	32

## List of Tables

Table 1-1. Raw TCP APIs.....	4
Table 1-2. Raw UDP APIs.....	5
Table 2-1. Application Examples.....	7

## Trademarks

LaunchPad™, TivaWare™, and Code Composer Studio™ are trademarks of Texas Instruments.  
 Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.  
 All trademarks are the property of their respective owners.

## 1 Introduction

lwIP is a small implementation of the TCP/IP protocol suite. The implementation of lwIP allows a full-scale TCP support while using a very small amount of RAM and ROM, which is suitable for use in embedded systems. lwIP supports the following protocols:

- Address Resolution Protocol (ARP)
- Auto IP (AUTOIP). A server-less method for choosing an IP address.
- Domain names resolver (DNS)
- Dynamic Host Configuration Protocol (DHCP)
- Internet Control Message Protocol (ICMP)
- Internet Group Management Protocol (IGMP)
- Internet Protocol (IP)
- Simple Network Management Protocol (SNMP)
- Transmission Control Protocol (TCP)
- Point-to-Point Protocol (PPP)
- User Datagram Protocol (UDP)

### 1.1 License

lwIP is freely available under the BSD license.

### 1.2 lwIP Version

lwIP v1.4.1 has been ported to the TM4C129x MCU and is part of the TivaWare™ library release. All examples as described in this application note are built with lwIP v1.4.1 in the TivaWare v2.2.0.295 library.

### 1.3 lwIP Customization

Lwipopts.h is a user file that you can use to fully configure the lwIP and all of its modules. All of the options have default values set. You can optimize the code size by only compiling certain features that you really need for your application. The lwipopts.h has been configured to include features and memory management options that are suitable for the provided examples. The lwipopts.h can be found in the project folder for each example.

### 1.4 lwIP API

lwIP provides three styles of Application Programming Interface (API) for programs to use for communication with the TCP/IP code:

- Low-level Raw APIs
- Higher-level Netconn Sequential-style APIs
- BSD-style Socket APIs.

All examples demonstrated in this application report are based on the Raw API interface only. The Raw API provides a callback style interface to the application. The application first registers callback functions to different core events. Typical TCP events are:

- A new TCP connection is accepted.
- An acknowledgment by the remote host for the data that was sent to it.
- When new data arrives.
- A periodic interval event.

The user-supplied callback functions are called from the lwIP core when the corresponding event occurs. When using the Raw API interface, the TCP/IP code and the application program can both run in the same thread in a non-OS environment. The raw TCP/IP interface is faster in terms of code execution time and is also less memory intensive at the expense of coding complexity. The other two API styles are currently not supported by the TivaWare library. However, the BSD socket-based APIs are supported through the TI-RTOS NDK. Various socket-based Ethernet examples can be downloaded from the Code Composer Studio™ (CCS) [Resource Explorer](#).

### 1.4.1 TCP RAW APIs

Table 1-1 lists the typical lwIP Raw TCP APIs to be used in an application. Figure 1-1 shows a simplified flowchart of the TCP client-server communication using Raw APIs.

**Table 1-1. Raw TCP APIs**

Function Category	API	Description
TCP connection	tcp_new	Create a new TCP PCB (Protocol Control Block).
	tcp_bind	Bind the PCB to local IP address and port. The IP address can be specified as IP_ADDR_ANY in order to bind the connection to all local IP addresses.
	tcp_listen	Make PCB listen for incoming connections.
	tcp_accept	Set callback used for new incoming connections. When an incoming connection is accepted, the callback function specified with the tcp_accept() will be called.
	tcp_connect	Open connection to remote host. This function will return immediately and call the callback function specified in the fourth argument.
Receiving TCP data	tcp_recv	Set the callback function that will be called when new data arrives.
	tcp_recved	This function must be called when the application has received the data.
Sending TCP data	tcp_write	Enqueue data for transmission
	tcp_sent	Specifies the callback function that should be called when data has successfully been received (i.e. acknowledged) by the remote host.
	tcp_output	Force all enqueued data to be sent now.
Application poll	tcp_poll	Set application poll callback. When a connection is idle, lwIP will repeatedly poll the application by calling the specified callback function.
Closing connection and error handlings	tcp_close	Close the TCP connection.
	tcp_err	Set the callback function to call on connection errors.
	tcp_abort	Abort the connection.



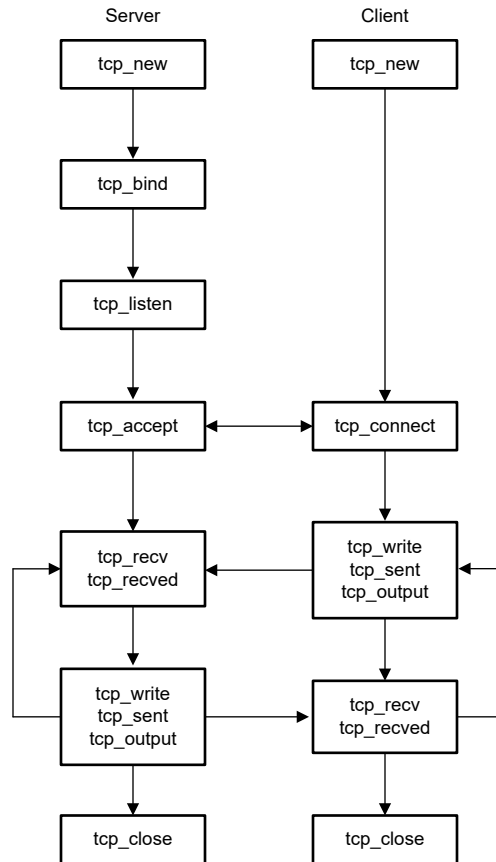


Figure 1-1. TCP Client Server Communication Using Raw API

### 1.4.2 UDP RAW APIs

Table 1-2 lists lwIP Raw UDP APIs to be used in an application. Figure 1-2 shows the simplified flowchart of the UDP client-server communication using Raw APIs.

Table 1-2. Raw UDP APIs

Function Category	API	Description
UDP connection	udp_new	Create a new UDP PCB (Protocol Control Block).
	udp_bind	Bind the PCB to local IP address and port. The IP address can be specified as IP_ADDR_ANY in order to bind the connection to all local IP addresses.
	udp_connect	Associate the PCB with the remote UDP peer address.
Receiving UDP data	udp_rcv	Set the callback function that will be called when new data arrives.
Sending UDP data	udp_send	Send UDP packet to the current remote host associated with the PCB.
	udp_sendto	Send data to a specified address using UDP.
Closing connection	udp_disconnect	Remove the remote end of PCB.
	udp_remove	Remove and deallocate the PCB.

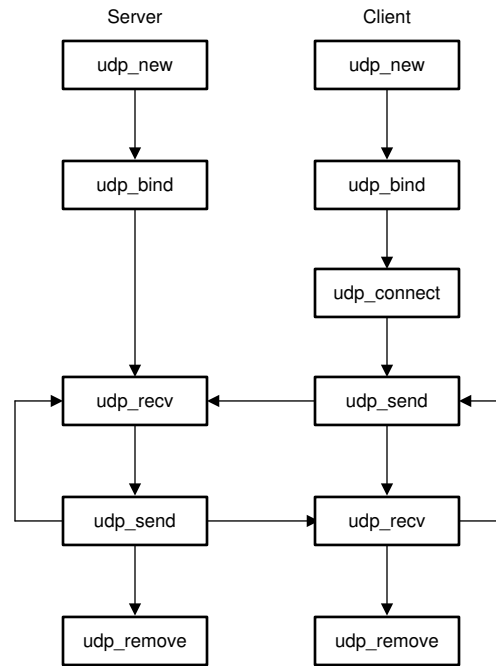


Figure 1-2. UDP Client Server Communication Using Raw API

## 1.5 lwIP Development Home Page and Support

Additional information about lwIP and support can be found in the below links:

- <https://savannah.nongnu.org/projects/lwip>
- [https://lwip.fandom.com/wiki/LwIP\\_Wiki](https://lwip.fandom.com/wiki/LwIP_Wiki)

## 2 Application Examples

The TivaWare library contains Ethernet examples that demonstrates creating HTTP web server applications based on lwIP stack. In this application report, the focus will be on demonstrating various echo-server and client applications. A collection of eight examples are presented here to show the TM4C129x MCU running with either the server applications or client applications.

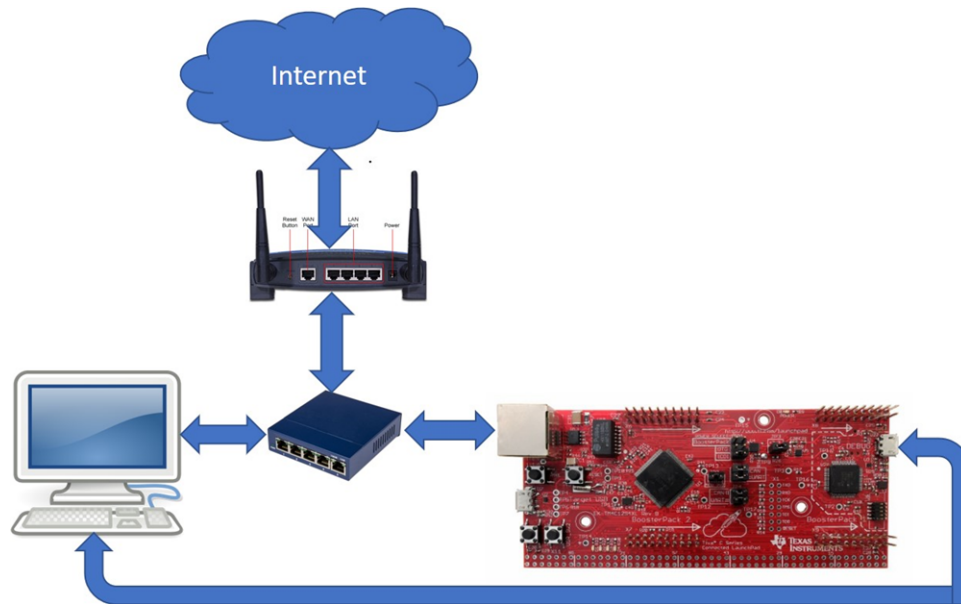
**Table 2-1. Application Examples**

Example	Type	Description
enet_tcpecho_server_lwip	Server	An echo server application using TCP protocol. The server echoes back the packets received from the client. lwIP is configured to acquire the IP address from a DHCP server.
enet_tcpecho_server_static_ip_lwip	Server	An echo server application using TCP protocol but with lwIP configured for static IP address generation.
enet_udpdecho_server_lwip	Server	An echo server application using UDP protocol. The server echoes back the datagrams it receives from the client.
enet_dns_lwip	Client	A client application that requests the DNS (Domain Name Server) server to translate domains names into IP addresses, making it possible for the DNS clients to reach the origin server.
enet_sntp_lwip	Client	A client application that reports the current network time based on the SNTP (Simple Network Time Protocol).
enet_tcpecho_client_lwip	Client	An echo client application using TCP protocol. The client sends a greeting message to the server and echoes back the packets it receives from the server.
enet_adcSENSOR_client_lwip	Client	A client application that uses the on-chip ADC to periodically take the temperature reading of the device and send it to the server.
enet_udpecho_client_lwip	Client	An echo client application using UDP protocol. The client sends a greeting message to the server and echoes back the datagrams it receives from the server.

## 3 Application Setup

### 3.1 Hardware Setup

- A network router that connects to the internet
- An Ethernet switch that connects devices to the LAN network. The switch is optional if you can connect the devices to the router directly. Most of the home routers have LAN ports for wired connection to devices.
- A PC is used to:
  - Debug the target device.
  - Act as either a server or client to generate or respond to network traffics to/from the target device.
  - Monitor the network traffic.
- The EK-TM4C1294XL LaunchPad evaluation kit that runs the applications provided in this document.



**Figure 3-1. Hardware Setup for the Application Examples**

### 3.2 Software Tools

Several tools are used to facilitate the debugging and testing of the application examples:

- **CCS**. The IDE tool is used to debug the target device and build the application examples. The CCS v10.1.1 and the TI v20.2.4.LTS compiler is used for this application report.
- **SocketTest**. A free small software tool that tests any server or client that uses TCP or UDP protocol to communicate.
- **Wireshark**. A free packet analyzer that is used for network troubleshooting and analysis in this application report.
- **Terminal Emulator**. The application examples use the terminal window to display its outputs. Any serial terminal emulator may be used. CCS has a built-in terminal emulator which is used in this application report. The terminal window can be invoked with “View” -> “Terminal”.

## 4 Download and Import the Ethernet examples

There are eight CCS project examples attached as collateral to this document. Click on the URL: <https://www.ti.com/lit/zip/spna248> to download the examples. You can unzip the project or keep it in the zip format. Both formats can be imported to the CCS.

1. To import the project into CCS, first select the “File” -> ”Import”.

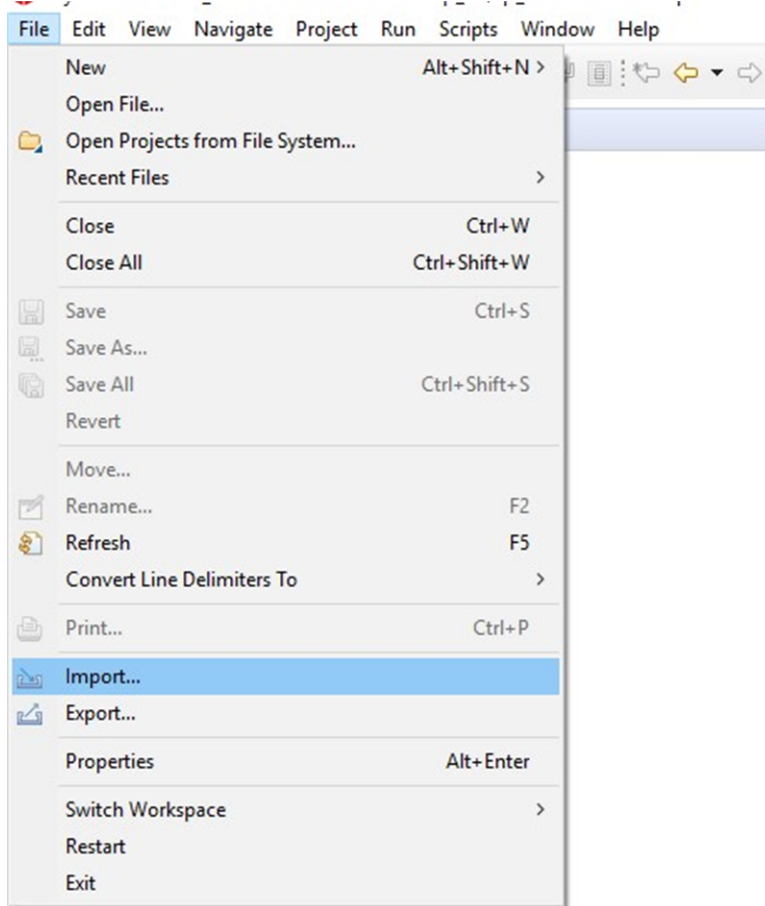
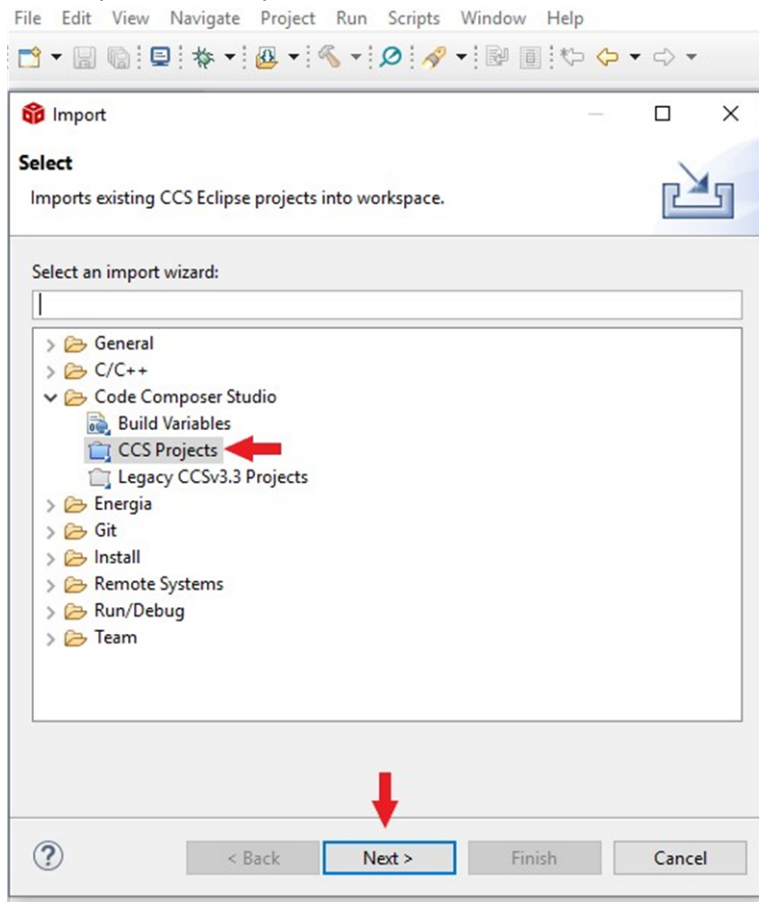


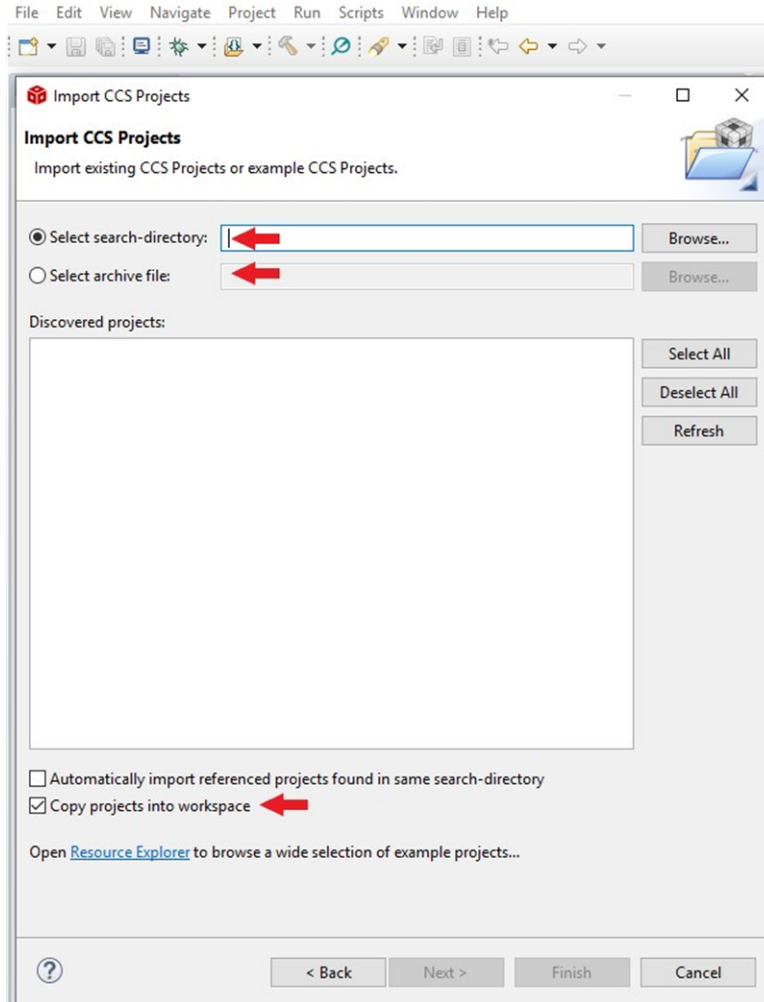
Figure 4-1. Import CCS Projects Step 1

2. Select “CCS Projects” to import the examples and then click “Next”.



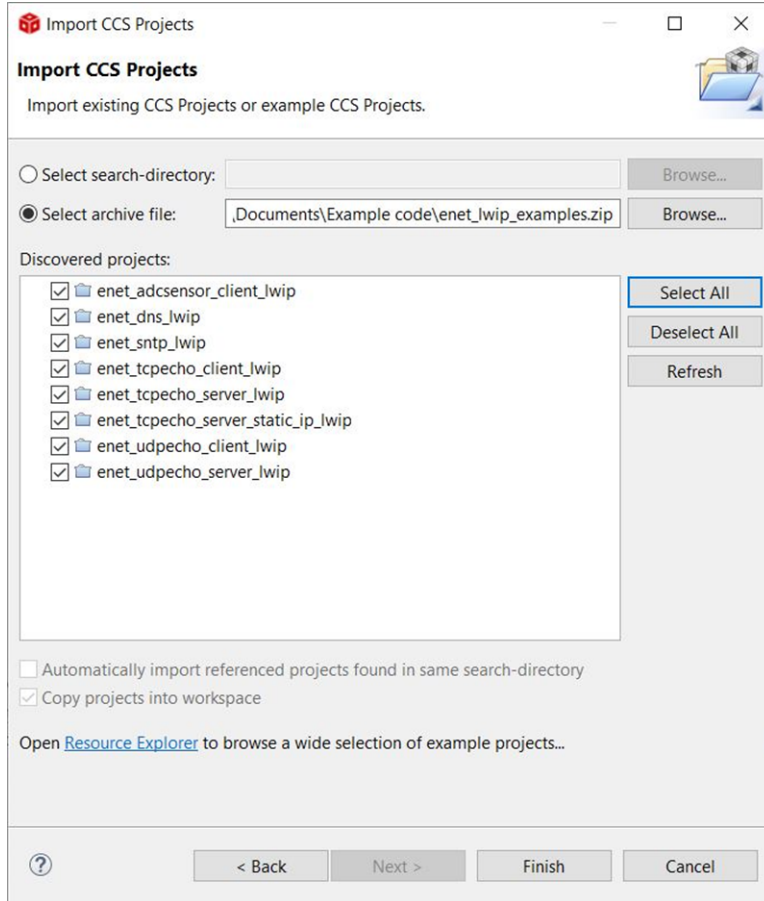
**Figure 4-2. Import CCS Projects Step 2**

3. Next, provide the path to either the unzipped project by selecting the first radio button or import the zip file directly by selecting the second radio button. Click the “Copy projects into workspace”.



**Figure 4-3. Import CCS Projects Step 3**

- After the project path is provided, a total of eight discovered projects will show up. First click the “Select All” button and then click the “Finish” button to complete the import.



**Figure 4-4. Import CCS Projects Step 4**



## 5 Enet\_tcpecho\_server\_lwip Example Overview

The enet\_tcpecho\_server\_lwip example demonstrates an echo-server application running on the TM4C129x MCU using TCP Transmission Control Protocol (TCP) as the underlying transport layer protocol. TCP is a connection-oriented protocol with built in error recovery and retransmission. The connection protocol is likened to a telephone connection. Both the sender and the receiver need to handshake for the connection (for example, the caller calls the number and the callee picks up the call) before communicating. The connection is there until one party hangs up the connection. TCP is used by applications when guaranteed error-free message delivery is required.

In this example, the TM4C129x MCU is acting as a server. The lwIP stack is configured for DHCP to automatically acquire an IP address. Once acquired, the IP address is displayed on the terminal window. The echo-server is ready by this time. The server will listen for the connection from the client. Once the client makes a connection the communication between the server and the client can start. The server implemented in this example will process the received characters by first inverting the case before echoing the inverted characters back to the client.

### 5.1 Build and Flash the Program

First, select the enet\_tcpecho\_server\_lwip as the active project. With the LaunchPad's ICDI USB port connected to the PC via the USB cable, build the project and load the program by clicking the Debug icon.

Click on the [CCS User's Guide](#) if you are new to the CCS.

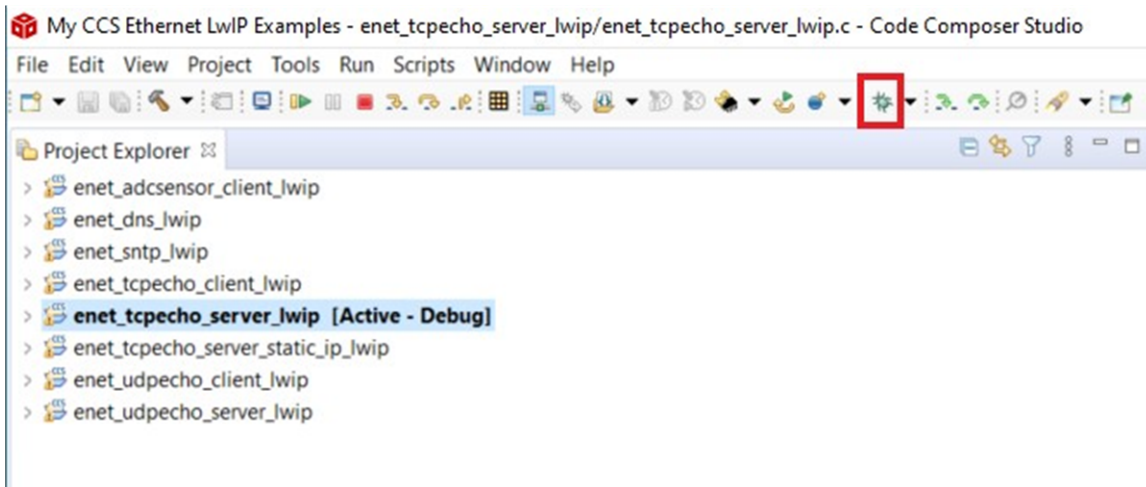


Figure 5-1. Debug CCS Projects

### 5.2 Check and Program the MAC Address

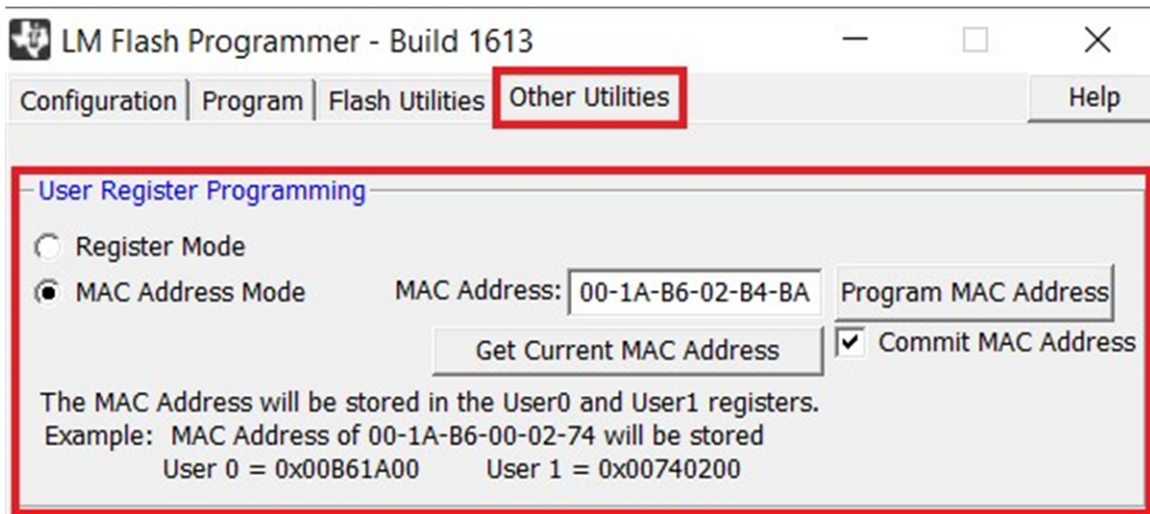
Every Network Interface Controller (NIC) on the network must be uniquely identified by a MAC address for communications within a network segment. A MAC address is a 48-bit value represented as six octets of two hexadecimal digits. MAC addresses are primarily assigned by device manufactures. The first three octets are the Organizationally Unique Identifier (OUI). The MAC address is normally pre-programmed on the EK-TM4C1294XL LaunchPad board. There is also a sticker on the back of the LaunchPad with the written MAC address. The pre-programmed MAC addresses will have the first three octets equal to 00:1A:B6 which uniquely identifies Texas Instruments. If you have a virgin device then the MAC address is not pre-programmed. You must program the MAC address yourself with the addresses that are allocated to your organization.

There are three tools to read and program the MAC address.

### 5.2.1 Using The LM Flash Programmer

This tool is most suitable if you are using the EK-TM4C1294XL LaunchPad. The LM Flash Programmer only supports the ICDI debug probe that is built-in on the LaunchPad. To check and program the MAC address follow the below steps:

1. Open the LM Flash Programmer and go to the “Other Utilities” tab.
2. Select the “MAC Address Mode” radio button.
3. To read the MAC address, press the “Get Current MAC Address” button.
4. To program the MAC address:
  - a. Type the six-octet MAC address into the “MAC Address” field.
  - b. Click the “Commit MAC Address” checkbox. This commit will permanently store the MAC address on the internal EEPROM. If the commit is not checked, the MAC address just entered will be temporary until the next power cycle.
  - c. Press the “Program MAC Address” button to complete the programming.



**Figure 5-2. MAC Address Programming Using LM Flash Programmer**

### 5.2.2 Using the CCS

CCS also has a built-in utility to program the MAC address. CCS is most suitable if CCS is your choice of IDE for software development.

To read and program the MAC address, first go to “Tools” -> “On-Chip Flash”. The steps to read and program the MAC address will be the same as described in [Section 5.2.1](#).

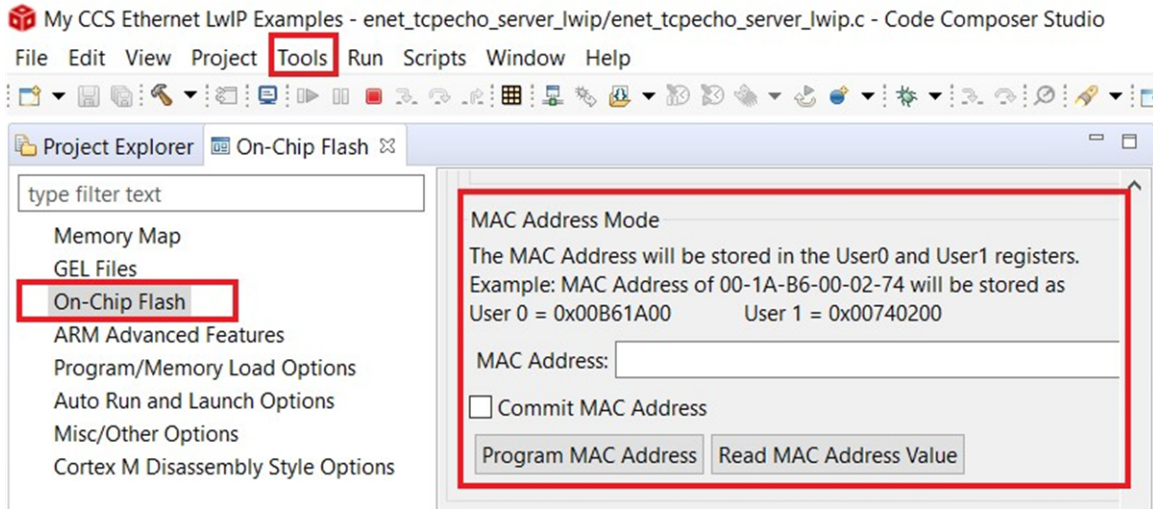


Figure 5-3. MAC Address Programming Using CCS

### 5.2.3 Using Uniflash

UniFlash is a TI standalone tool that supports programming of various TI devices including the MAC address for the TM4C129x MCU. UniFlash is most suitable if you are programming the MAC address on your custom board where your debug probe is not ICDI although ICDI is also supported by this tool.

To read and program the MAC address, first go to the “Settings and Utilities” tab. The steps to read and program the MAC address will be the same as described in [Section 5.2.1](#).

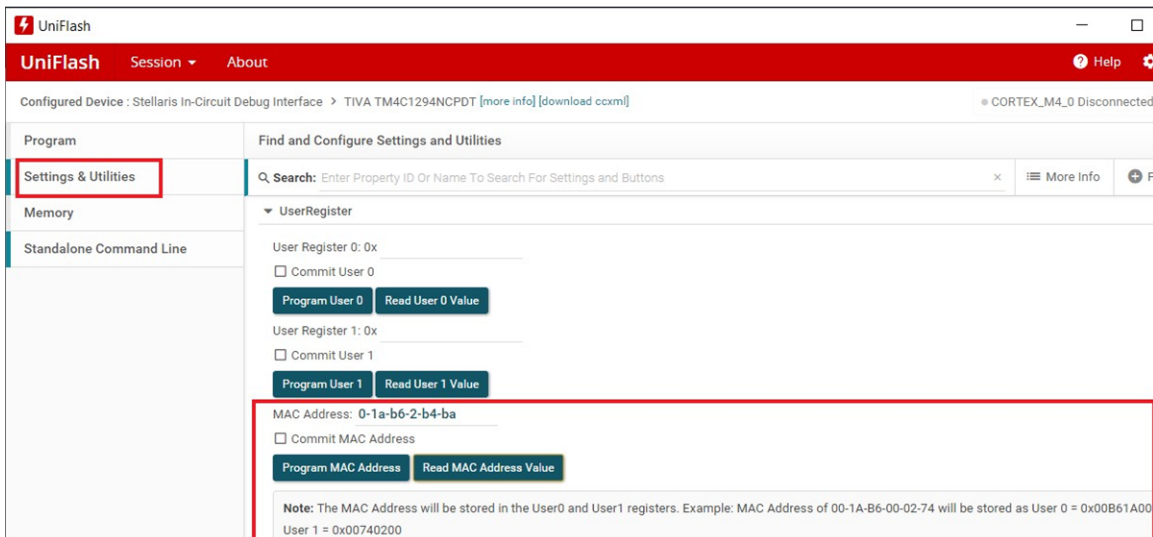
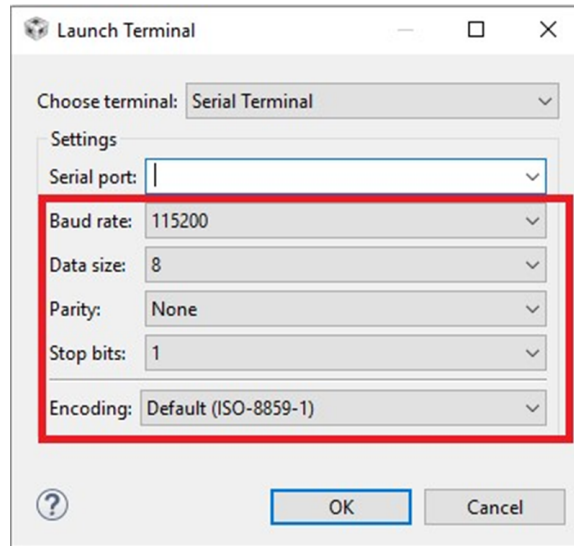


Figure 5-4. MAC Address Programming Using Uniflash

### 5.3 Configure the Terminal Window

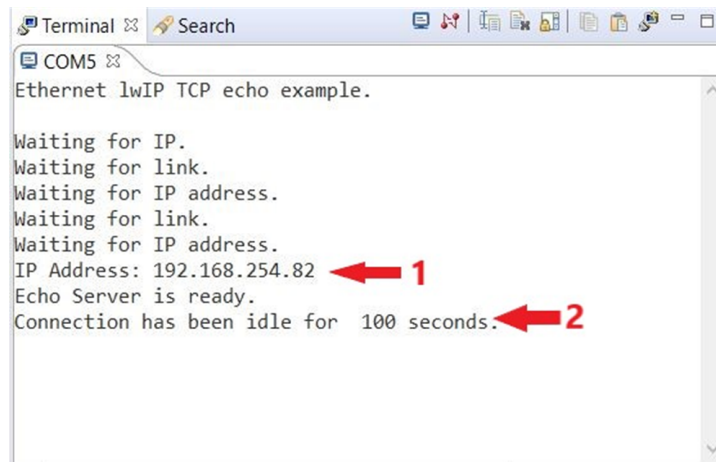
The example displays various information such as the acquired DHCP IP address and the progress of the communication on the terminal emulator. You can use any terminal emulator of your choice. The terminal emulator should be configured for 115200 Baud and 8-N-1 as shown in [Figure 5-5](#).



**Figure 5-5. Serial Terminal Settings**

### 5.4 Run the enet\_tcpecho\_server\_lwip Example

Connect the EK-TM4C1294XL LaunchPad to either the Ethernet switch or the router with an Ethernet cable as shown in [Figure 5-6](#). Run the example. With the terminal window opened, you should see the IP address (pointed by arrow 1) displayed and the server is ready as shown in [Figure 5-6](#). Record the IP address as you will need this information on the client side. Initially the server will be in a listening state waiting for the client to connect to it. Therefore, to continue the rest of the example, a remote client needs to be setup.



**Figure 5-6. Enet\_tcpecho\_server\_lwip Output**

The SocketTest tool is used that will act as the client running on the PC. Make sure the PC is connected to the same network as the EK-TM4C1294XL with the same subnet mask.

Follow the steps shown in Figure 5-7 to setup the client:

1. Open SocketTest and enter the server IP address as well as the port number 23. Port 23 is the default Telnet port number in TCP and UDP protocols. Finally, press the “Connect” button. In a short while the connection with the server will be established and you are ready for conversation with the server.
2. Go to the “Message” field and type in some messages and then hit the “Send” button.
3. The message you enter will be displayed in the conversation field. When the server receives the message, it will invert the case of the message and then echo back the message to the client. The server also replies to the client the number of characters it receives.

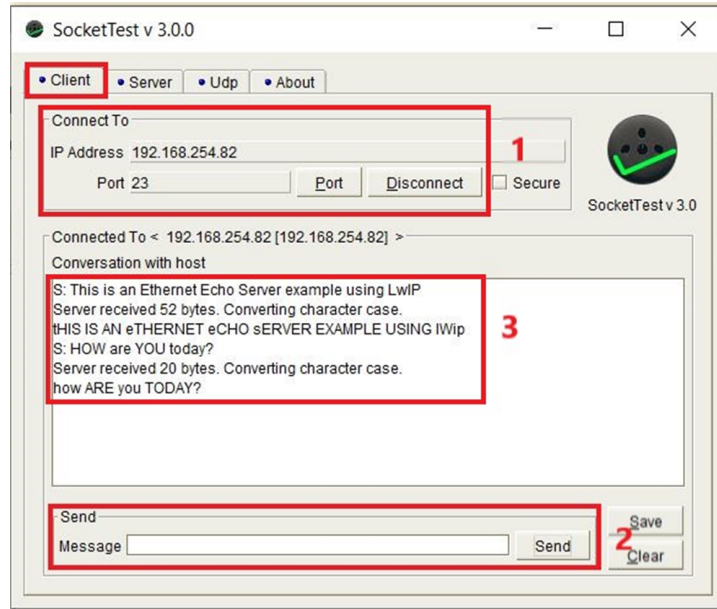


Figure 5-7. SocketTest Client Configuration for Enet\_tcpecho\_server\_lwip

Examine the second message “HOW are YOU today?” that was sent by the client to the server. If you count manually, this message has a total length of 20 bytes including the two \n\r escape characters. The \n is the New Line and \r is the Carriage Return character in the ASCII table.

The server replied with a message that says “Server received 20 bytes. Converting character case. how ARE you TODAY?”. First, the number of characters that was received by the server was indeed 20. The entire message that was sent back by the server has a total length of 73 bytes.

No.	Time	Source	Destination	Protocol	Length	Info
702	95.623365	192.168.254.75	192.168.254.82	TELNET	106	Telnet Data ...
703	95.635308	192.168.254.82	192.168.254.75	TELNET	159	Telnet Data ...
704	95.676244	192.168.254.75	192.168.254.82	TCP	54	58696 → 23 [ACK] Seq=53 Ack=106 Win=64135 Len=0
848	110.146923	192.168.254.75	192.168.254.82	TELNET	74	Telnet Data ...
849	110.491747	192.168.254.82	192.168.254.75	TELNET	127	Telnet Data ...
850	110.532014	192.168.254.75	192.168.254.82	TCP	54	58696 → 23 [ACK] Seq=73 Ack=179 Win=64062 Len=0

```

> Frame 848: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{24DD389D-57EF-43B1-8B10-771DF1
> Ethernet II, Src: IntelCor_65:b6:75 (8c:c6:81:65:b6:75), Dst: TexasIns_02:b4:ba (00:1a:b6:02:b4:ba)
> Internet Protocol Version 4, Src: 192.168.254.75, Dst: 192.168.254.82
> Transmission Control Protocol, Src Port: 58696, Dst Port: 23, Seq: 53, Ack: 106, Len: 20
  Telnet
    Data: HOW are YOU today?\r\n
  
```

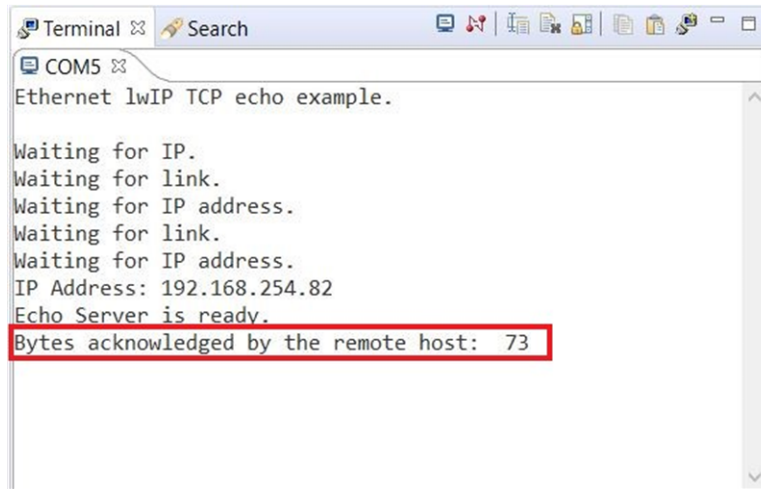
0000	00 1a b6 02 b4 ba 8c c6 81 65 b6 75 08 00 45 00	..... e-u..E..
0010	00 3c 78 5c 40 00 80 06 00 00 c0 a8 fe 4b c0 a8	<x\@... ..K..
0020	fe 52 e5 48 00 17 f6 79 b3 5e 00 00 1a 07 50 18	-R-H...y ..^...P..
0030	fa 87 7e 1e 00 00 48 4f 57 20 61 72 65 20 59 4f	.....HO W are YO
0040	55 20 74 6f 64 61 79 3f 0d 0a	U today? ..

Figure 5-8. Client to Server Wireshark Capture for Enet\_tcpecho\_server\_lwip



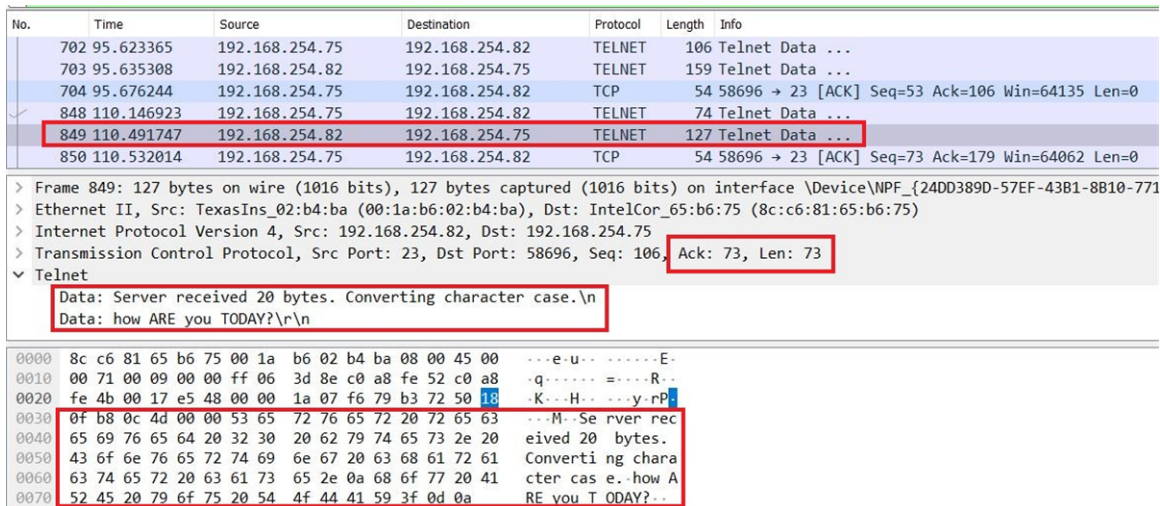
Figure 5-9 shows the corresponding message from the client (IP address 192.168.254.75) to the server (IP address 192.168.254.82).

When the server sends the message, it waits for the acknowledgment from the client confirming the data was received successfully. In the callback function called by tcp\_sent(), the server simply prints the number of bytes that was acknowledged by the client on the terminal window as shown in Figure 5-9.



**Figure 5-9. Client Confirmation of Data Received for Enet\_tcpecho\_server\_lwip**

Also, examine the Wireshark capture for the server's response in Figure 5-10. The server (IP Address 192.168.254.82) sends the 73 characters message to the client (192.168.254.75) and it was acknowledged by the client for the reception of the data.



**Figure 5-10. Server to Client Wireshark Capture for Enet\_tcpecho\_server\_lwip**

Suppose you let the connection idle, the server application utilizes the tcp\_poll() to periodically trigger the callback function every 5 seconds to print out the elapse time the server is idle. In a real application, it is possible to program the idle internal before closing the connection to save power. Refer to the box 2 in Figure 5-6 where it reports 100 seconds as the time the server is idle since the last transaction.

## 6 Enet\_tcpecho\_server\_static\_ip\_lwip Example Overview

The `enet_tcpecho_server_static_ip_lwip` example is very similar to the `enet_tcpecho_server_lwip` except that a static IP address is used for the server. There are various reasons where a static IP address is desired. One example would be that the device is setup as an FTP or web server. You would want to ensure that people can always access the server. If the server were assigned a dynamic address, it is possible that the address can change occasionally which would prevent the router from knowing which device on the network is the server.

When assigning static IP addresses for local devices on home or private networks, the addresses should be chosen from the private IP address ranges defined by the Internet Protocol (IP). The static addresses should be limited to the defined ranges:

- 10.0.0.0 – 10.255.255.255
- 172.16.0.0 – 172.31.255.255
- 192.168.0.0 – 192.168.255.255

Make sure the static address you choose is not in use by any other device in your private network or it will result in address conflict errors. Consult with your network system administrator for static IP address selection.

If you want to experiment with the static IP address and are unsure which address is currently unused in the network, you can first run the `enet_tcpecho_server_lwip` to let the DHCP choose the IP address. This dynamic address will be leased to your device for some time and this ensures no other devices on the same network will be given the same address until the address has expired. Record this address and use it for your static address generation.

### 6.1 How to Configure lwIP for Static Address

The `lwipopts.h` file allows different features to be enabled or disabled. To use static IP address, there are three steps to follow:

1. The DHCP and AUTOIP features needs to be disabled in the `lwipopts.h` file.

```
#define LWIP_DHCP                0
#define LWIP_AUTOIP              0
```

2. Define the static IP address, subnet mask and gateway mask in the `enet_tcpecho_server_static_ip_lwip.c` file. Note the below address is only an example. Users must change the static address per your own network, otherwise, the example will not work.

```
#define IPADDR "82.254.168.192"
#define NETMASK "0.0.255.255"
#define GWMASK "0.0.255.255"
#define PORT 23
```

3. Initialize lwIP with the `IPADDR_USE_STATIC` flag.

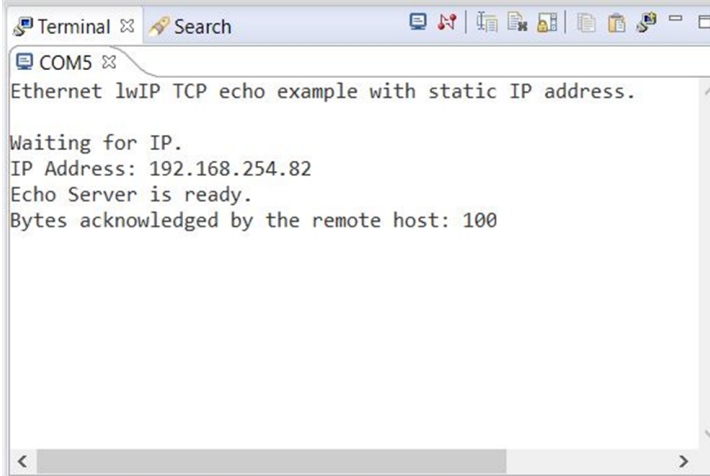
```
lwIPInit(g_ui32SysClock, pui8MACArray,
         inet_addr(IPADDR),
         inet_addr(NETMASK),
         inet_addr(GWMASK),
         IPADDR_USE_STATIC);
```

4. Recompile the project and you are ready to run the example.

## 6.2 Run the enet\_tcpecho\_server\_static\_ip\_lwip Example

Since this example is much the same as the enet\_tcpecho\_server\_lwip, it will only show the captures without detailed explanation.

The biggest difference between [Figure 5-6](#) and [Figure 6-1](#) is that the application instantly displays the static IP address as there is no need to wait for the DHCP server to return the address.

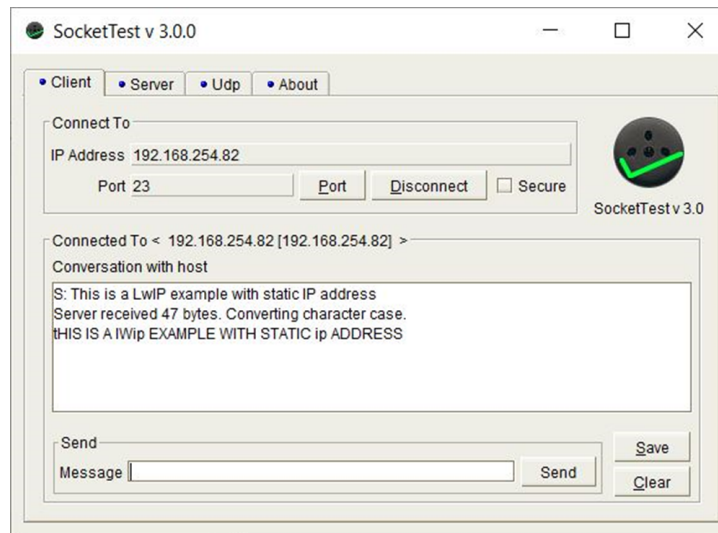


```

Terminal
COM5
Ethernet lwIP TCP echo example with static IP address.

Waiting for IP.
IP Address: 192.168.254.82
Echo Server is ready.
Bytes acknowledged by the remote host: 100
  
```

**Figure 6-1. Enet\_tcpecho\_server\_static\_ip\_lwip Output**



**Figure 6-2. SocketTest Client Configuration for Enet\_tcpecho\_server\_static\_ip\_lwip**



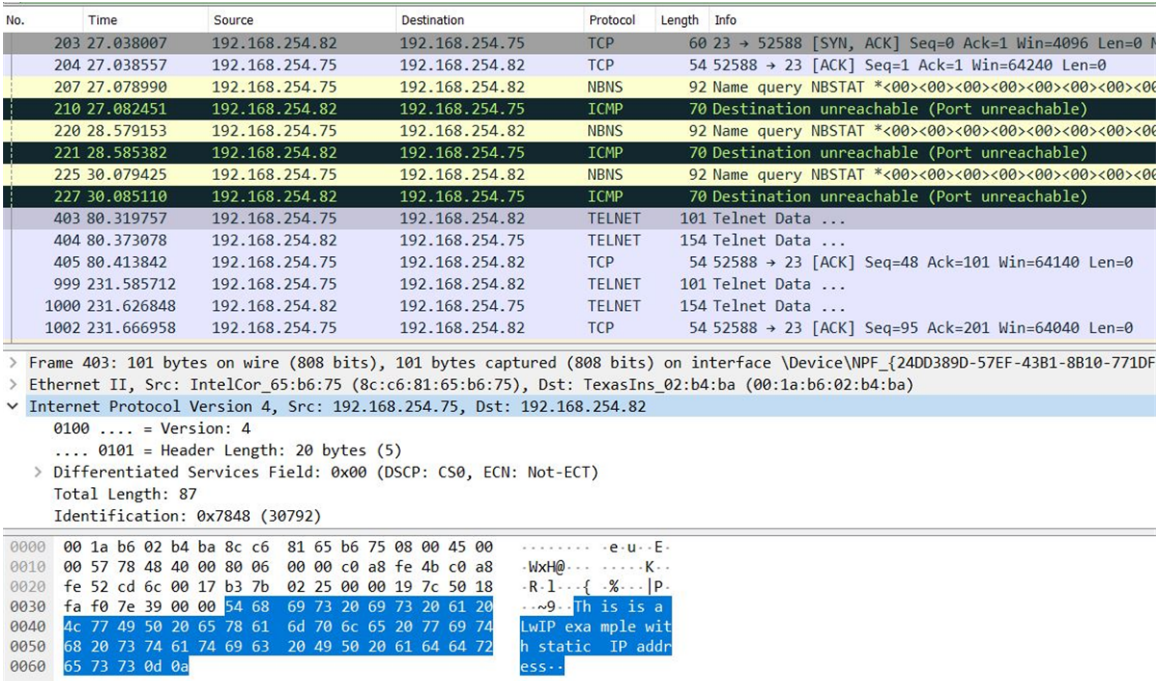


Figure 6-3. Client to Server Wireshark Capture for Enet\_tcpecho\_server\_static\_ip\_lwip

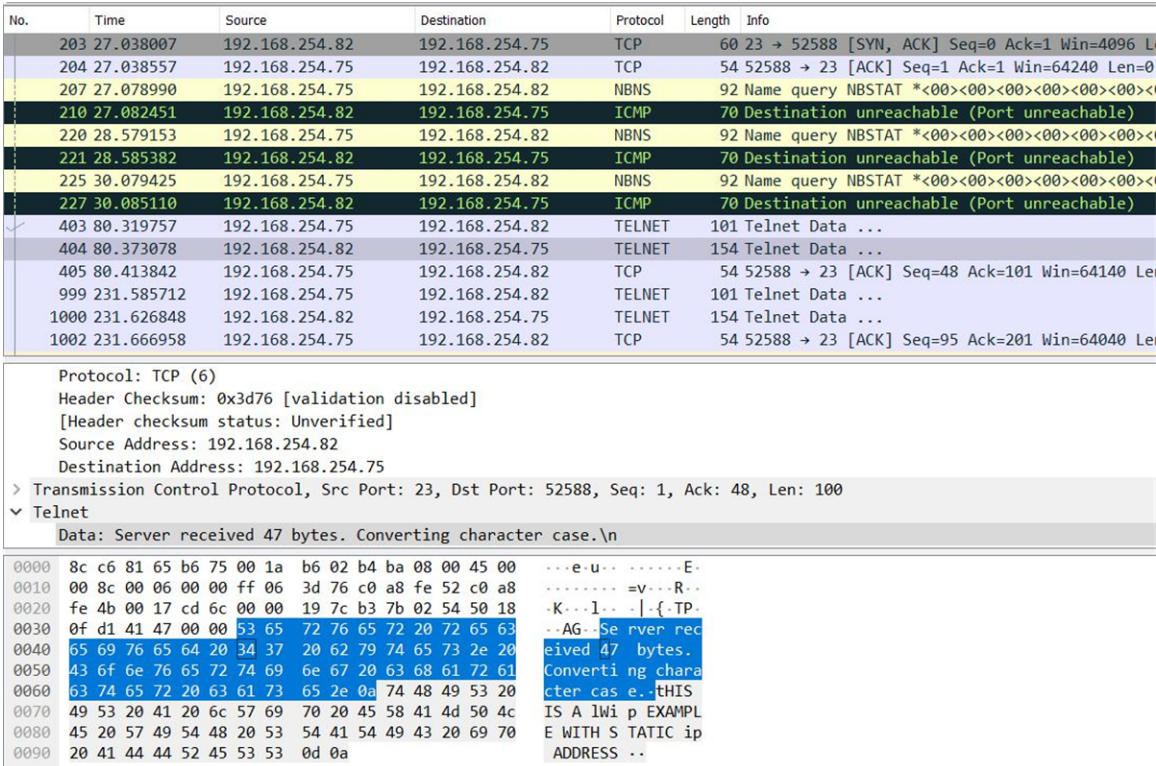


Figure 6-4. Server to Client Wireshark Capture for Enet\_tcpecho\_server\_static\_ip\_lwip

## 7 Enet\_udpecho\_server\_lwip Example Overview

The UDP (User Datagram Protocol) is another well-known transport layer protocol. The enet\_udpecho\_server\_lwip example demonstrates an echo-server application running on the TM4C129x MCU using the UDP protocol. The UDP is a connectionless protocol with no error recovery and retransmission. The connectionless protocol is likened to the mail delivery. You drop off your mail with the post office and there is no guarantee the mail will be delivered to the recipient and neither is the mail guaranteed to be delivered in its original form (for example, damage to the mail due to rain or mishandling).

### 7.1 Run the enet\_udpecho\_server\_lwip Example

The SocketTest tool that is used acts as the client running on the PC. Make sure the PC is connected to the same network as the EK-TM4C1294XL with the same subnet mask.

Follow the steps shown in [Figure 7-2](#) to setup the SocketTest:

1. Go to the “Client” tab.
2. Enter the PC’s IP address and the port number 23 and press the “Start Listening” button. The server IP address should be the address of the PC running SocketTest. To find out the PC’s IP address in your network, you can use the Windows’ ipconfig command. Bring up a Windows command window and at the prompt types “ipconfig” and you will see the IP address assigned to your PC. For example, see [Figure 7-1](#). Note in SocketTest, the Server address field will be the address of the PC regardless if the PC is the actual server or client. SocketTest just listens for any incoming data at the specified address and port.
3. Enter the MCU’s IP address in the “IP Address for Client”. The IP address assigned to the MCU is shown in the terminal window.
4. Go to the “Message” field and type in some messages and then hit the “Send” button.
5. Observe the conversation field in SocketTest as well as the Wireshark captures in [Figure 7-3](#) and [Figure 7-4](#).

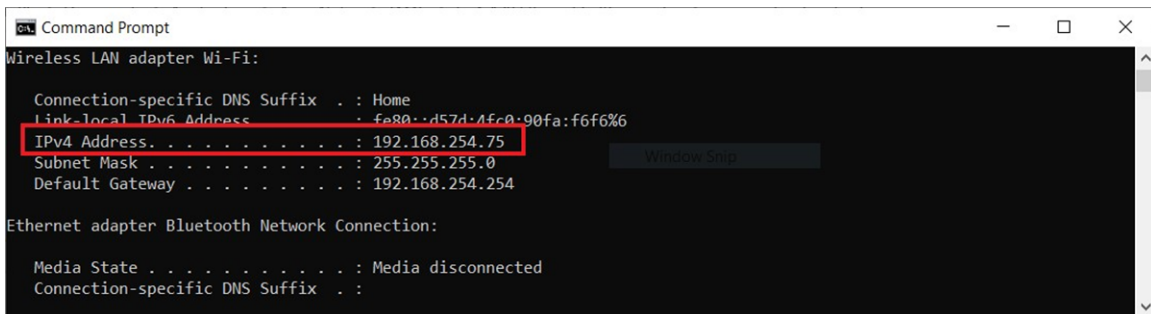


Figure 7-1. Query IP Address of the PC

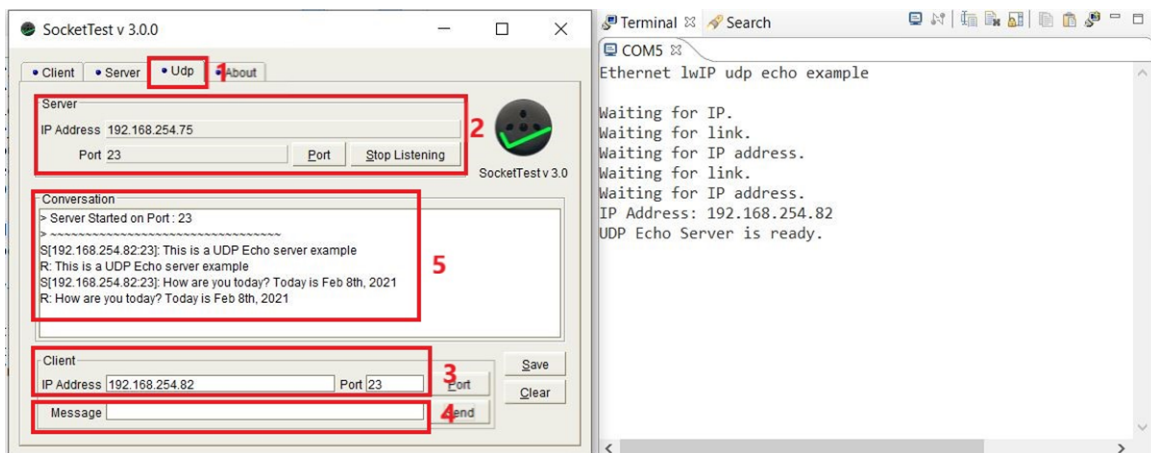


Figure 7-2. Enet\_udpecho\_server\_lwip Output

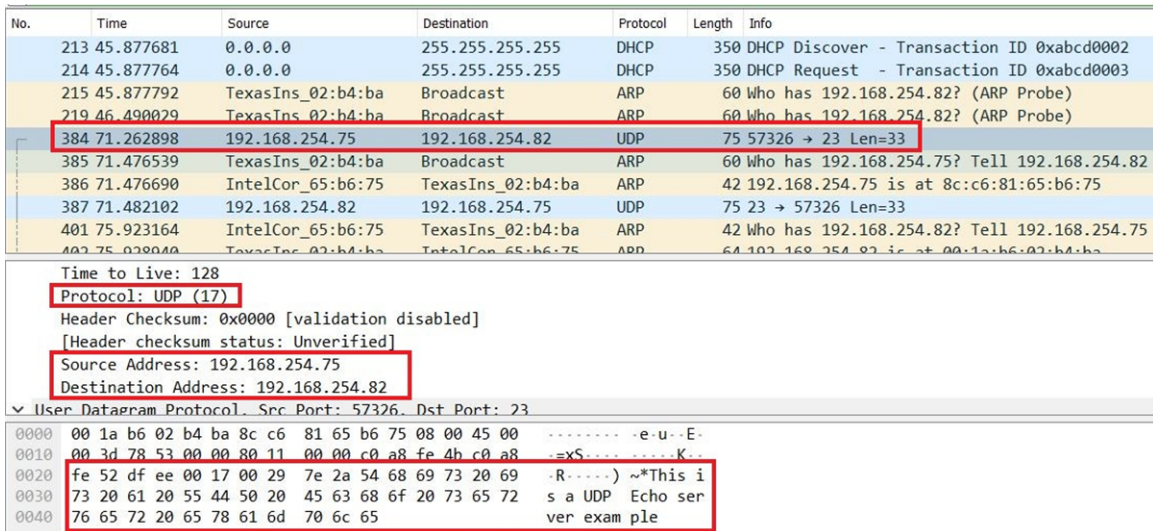


Figure 7-3. Client to Server Wireshark Capture for Enet\_udpcho\_server\_lwip

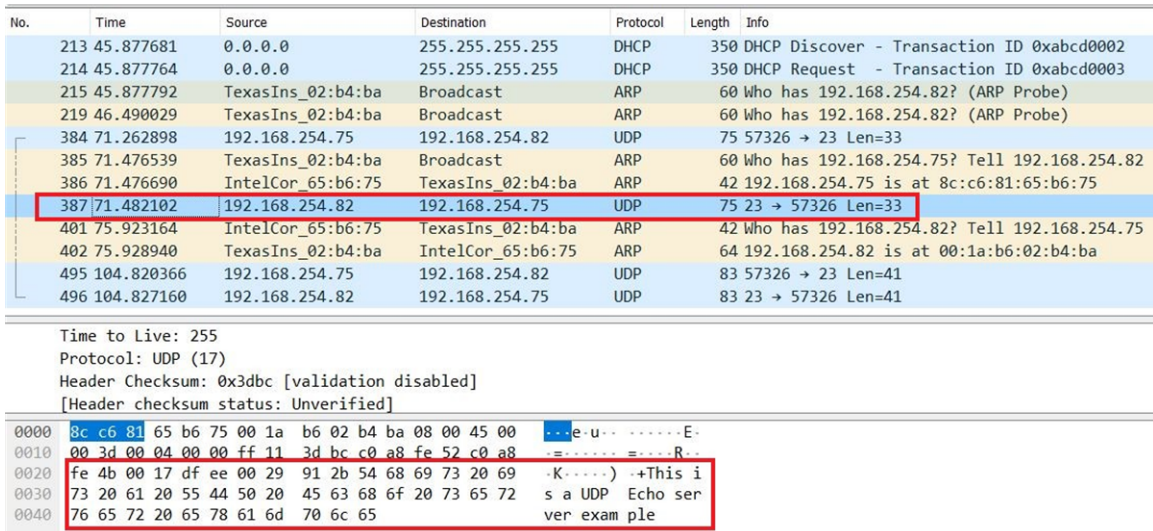


Figure 7-4. Server to Client Wireshark Capture for Enet\_udpcho\_server\_lwip



## 8 Enet\_dns\_lwip Example Overview

Every device connected to a network will have a unique IP address which relies on the IP (Internet Protocol) for communications. However, the IP address is either a 32-bit address as in IPv4 or 128-bit as in IPv6 which is hard to remember by humans. The DNS (Domain Name System) is an application layer service that translates the “human-friendly” domains names to IP addresses. It is easy for humans to remember [www.ti.com](http://www.ti.com) as opposed to its numerical IP address. Note the DNS is rather an application service that relies on the UDP protocol rather than a protocol per se.

This simple example demonstrates using DNS feature to retrieve the IP addresses of four different websites.

### 8.1 How to Configure lwIP for DNS

To use DNS, there are three steps to follow:

1. Configure or uncomment the below default #define in the lwipopts.h file.

```
#define LWIP_DNS                1
#define DNS_TABLE_SIZE        4
#define DNS_MAX_NAME_LENGTH   256
#define DNS_MAX_SERVERS       2
#define DNS_DOES_NAME_CHECK    1
#define DNS_USES_STATIC_BUF    1
#define DNS_MSG_SIZE          512
```

2. Since the DNS relies on the UDP protocol, the application needs to include the below header files.

```
#include "lwip/udp.h"
#include "lwip/inet.h"
#include "lwip/dns.h"
```

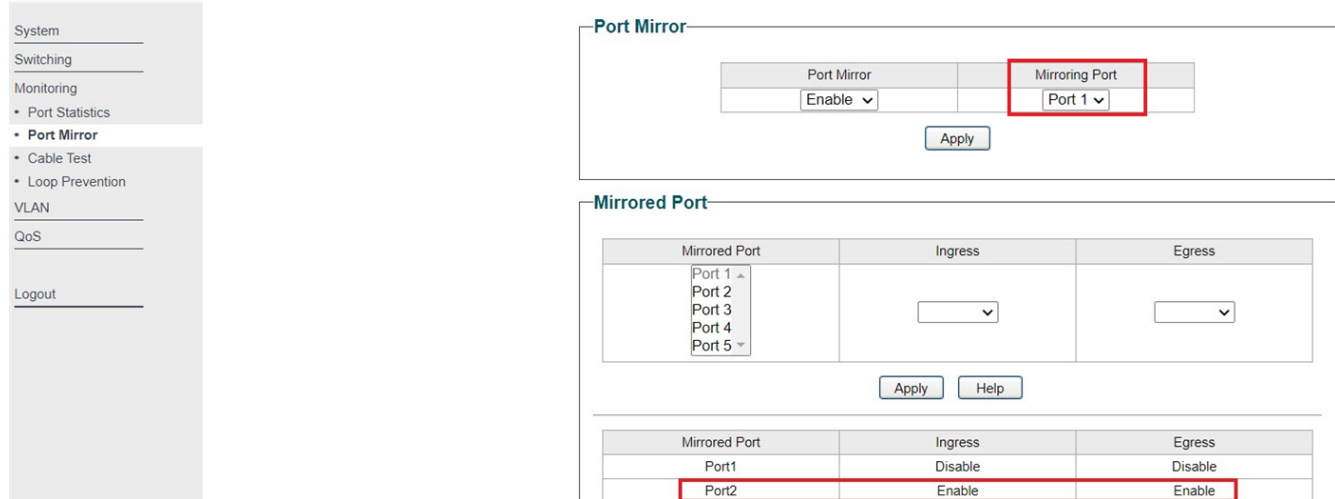
3. Use the `dns_gethostbyname()` API to translate the domain names. Below is an example usage. See the `enet_dns_lwip.c` file for detail.

```
err = dns_gethostbyname(host_addresses[current_host], &dns_resolved_address,
                        DnsFound, NULL);
```

4. Recompile the project and you are ready to run the example.

## 8.2 How to View the DNS Traffic on Wireshark

If you hook up the MCU device and the PC to the Ethernet switch, you will not be able to view the DNS traffic on Wireshark. The reason is that the DNS traffic is between the DNS server and the device. The switch will not route the traffic to the PC running Wireshark. In order to view the DNS traffic, you either have an Ethernet hub which broadcasts all traffic on the network to all ports connected to it or you need to configure the ethernet switch for “Port Mirroring”. An ethernet hub is hard to find these days. It is easy to find a Smart Switch that you can configure for Port Mirroring. Figure 8-1 shows where the EK-TM4C1294XL connecting to the switch Port 2 is mirrored onto Port 1 that is connected to the PC. With the port mirroring, all traffic to/from Port 2 will be snooped by the Wireshark on port 1. Consult your switch or router data sheet on how to configure port mirroring.



The screenshot shows a web-based configuration interface for a network switch. On the left is a navigation menu with options like System, Switching, Monitoring, and Port Mirror. The main area is divided into two sections: 'Port Mirror' and 'Mirrored Port'.

**Port Mirror Section:**

Port Mirror	Mirroring Port
Enable ▾	Port 1 ▾

Below this is an 'Apply' button.

**Mirrored Port Section:**

Mirrored Port	Ingress	Egress
Port 1 ▲ Port 2 Port 3 Port 4 Port 5 ▾	▾	▾

Below this is an 'Apply' button and a 'Help' button.

**Summary Table:**

Mirrored Port	Ingress	Egress
Port1	Disable	Disable
Port2	Enable	Enable

Figure 8-1. Port Mirroring

### 8.3 Run the enet\_dns\_lwip Example

As expected, running this example prints the four IP addresses of the corresponding websites on the terminal window. For more details, look at the Wireshark capture particularly for [www.google.com](http://www.google.com).

- In the highlighted box 1 and box 2 of the Wireshark capture in [Figure 8-3](#), four DNS packets are observed corresponding to the four websites.
- The source address of the transaction is from the IP address 192.168.254.254. This happens to be the IP address of the router of the home network to which this application is run from. A smart router normally will act as the DNS server and store the past visited domain names in its cache for fast retrieval.
- In box 3, the returned IP address for [www.google.com](http://www.google.com) is 172.217.1.132. You can confirm by typing this address on your browser's URL field and it should lead you to the Google website.
- In box 4, the IP address is expressed as a 32-bit binary value equal to 0xACD90184. The 0xAC is equal to decimal 172, the 0xD9 is equal to decimal 217 and likewise for the rest.

```

Terminal
COM5
Ethernet lwIP DNS example

Waiting for IP.
Waiting for link.
Waiting for IP address.
IP Address: 192.168.254.86
Starting DNS Service.
DnsRequest: Waiting for server address to be resolved.
hostname: pool.ntp.org           Resolved IP Address: 138.68.201.49
DnsRequest: Waiting for server address to be resolved.
hostname: www.google.com         Resolved IP Address: 172.217.1.132
DnsRequest: Waiting for server address to be resolved.
hostname: api.openweathermap.org Resolved IP Address: 192.241.245.161
DnsRequest: Waiting for server address to be resolved.
hostname: www.ti.com             Resolved IP Address: 104.100.228.145

```

Figure 8-2. Enet\_dns\_lwip Output

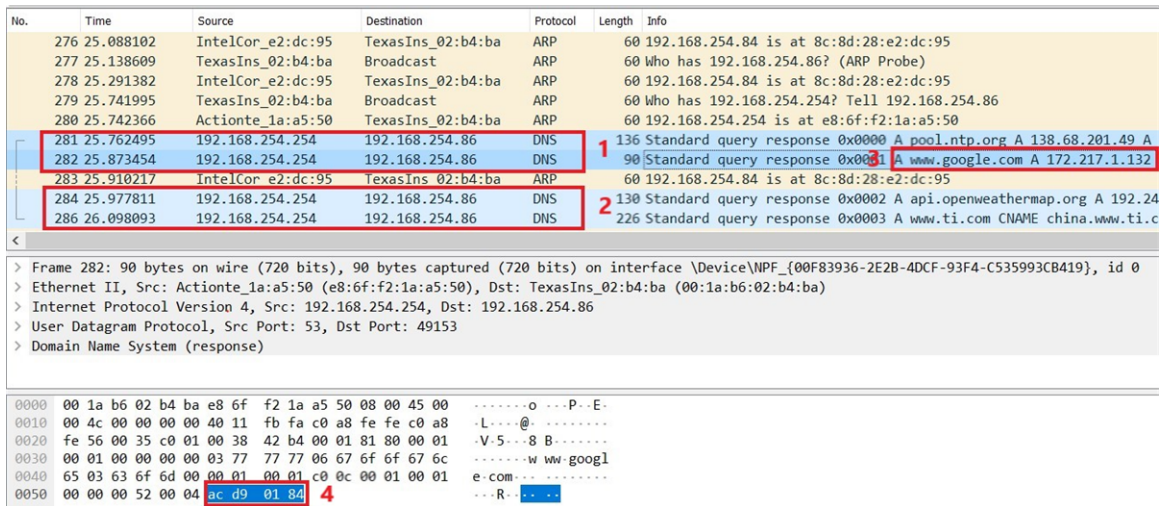


Figure 8-3. Wireshark Capture for Enet\_dns\_lwip

## 9 Enet\_sntp\_lwip Example Overview

Simple Network Time Protocol (SNTP) is a simplified version of NTP. SNTP typically provides time within 100mS of the accurate time without the complex filtering mechanisms of the NTP. The SNTP is an application layer protocol that is based on the UDP as the transport layer.

This example acquires the network time using the SNTP protocol and displays the current time adjusted for North American Central Time (CT) Zone on the terminal window.

### 9.1 Run the enet\_sntp\_lwip Example

The client sends the request to the pool.ntp.org website from which a randomly selected public time server is chosen to provide the network time as shown in Figure 9-2. The acquired network time is adjusted for Central Time Zone before displaying on the terminal window is shown in Figure 9-1.

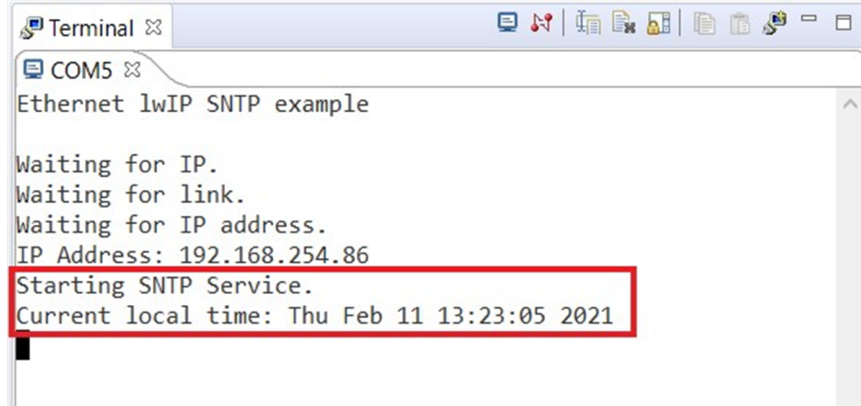


Figure 9-1. Enet\_sntp\_lwip Output

No.	Time	Source	Destination	Protocol	Length	Info
2816	115.662767	0.0.0.0	255.255.255.255	DHCP	350	DHCP Discover - Transaction ID 0xabcd0001
2817	115.667172	192.168.254.254	192.168.254.86	DHCP	326	DHCP Offer - Transaction ID 0xabcd0001
2818	115.667288	0.0.0.0	255.255.255.255	DHCP	350	DHCP Request - Transaction ID 0xabcd0002
2819	115.675017	192.168.254.254	192.168.254.86	DHCP	326	DHCP ACK - Transaction ID 0xabcd0002
2820	115.675110	TexasIns_02:b4:ba	Broadcast	ARP	60	Who has 192.168.254.86? (ARP Probe)
2823	115.724723	IntelCor_e2:dc:95	TexasIns_02:b4:ba	ARP	60	192.168.254.84 is at 8c:8d:28:e2:dc:95
2825	116.092307	TexasIns_02:b4:ba	Broadcast	ARP	60	Who has 192.168.254.86? (ARP Probe)
2826	116.133094	IntelCor_e2:dc:95	TexasIns_02:b4:ba	ARP	60	192.168.254.84 is at 8c:8d:28:e2:dc:95
2827	116.695674	TexasIns_02:b4:ba	Broadcast	ARP	60	Who has 192.168.254.254? Tell 192.168.254.86
2829	116.696395	Actionte_1a:a5:50	TexasIns_02:b4:ba	ARP	60	192.168.254.254 is at e8:6f:f2:1a:a5:50
2830	116.715972	192.168.254.254	192.168.254.86	DNS	136	Standard query response 0x0000 A pool.ntp.org A 50.205.244.23
2831	116.716105	192.168.254.86	50.205.244.23	NTP	90	NTP Version 4, client
2832	116.747180	IntelCor_e2:dc:95	TexasIns_02:b4:ba	ARP	60	192.168.254.84 is at 8c:8d:28:e2:dc:95

Figure 9-2. Wireshark Capture for Enet\_sntp\_lwip

## 10 Enet\_tcpecho\_client\_lwip Example Overview

The `enet_tcpecho_client_lwip` example demonstrates a client application that first connects to the server with a greetings message “Hello World!\n\r” and then echoes back whatever it receives from the server.

As illustrated in lwIP flowchart for TCP in [Figure 1-1](#), the client will use `tcp_connect()` to connect to the specified server address and port. Once the connection is established, the client will use `tcp_recv()` to setup a callback function for receiving data from the server and then echo the data back.

Another difference between the client and server in the flowchart is that the client doesn’t need to call `tcp_bind()`. Binding is normally not needed on the client side for TCP. There may be circumstances where binding the client is needed in which you will use `tcp_bind()` to bind the client. An example would be a firewall on the client that only allows outgoing connections on a certain port.

### 10.1 Configure the Server IP Address

To run this example, the server IP address and port number must be known during compile time so the client will make the connection with the server.

1. Define the `SERVER_IPADDR` and `SERVER_PORT` in the `enet_tcpecho_client_lwip.c` file. Note the below address is only an example. You must change the server IP address to the address which you are connecting to in your network, otherwise, the example will not work.

```
#define SERVER_IPADDR "192.168.254.75"
#define SERVER_PORT 8000
```

2. Recompile the project and you are ready to run the example.

### 10.2 Configure the SocketTest Server and Run the enet\_tcpecho\_client\_lwip Example

Since this is a client application, the server must be setup first and be in a listening state before the client can connect to it.

Follow the steps shown in [Figure 10-1](#) to setup the SocketTest server:

1. Open the Server tab in SocketTest.
2. Enter the server IP address as well as the port number. The IP address is the PC from which the SocketTest is running on. The IP and port number need to match the settings defined in the `enet_tcpecho_client_lwip.c` as shown in [Section 10.1](#). Finally, press the “Start Listening” button. Wait for the greetings message conversation window once the client connects.
3. After the client connects to the SocketTest server, it will send a “Hello World!\n\r” greetings message. The conversation field in box 3 of [Figure 10-1](#) displays the “Hello World!” message from the client IP address 192.168.254.85 as soon as the server accepts the connection from the client.
4. Go to the Message field in the SocketTest shown in box 4 and type some message. Whatever message is entered is echoed by the client prepended with a “Client:” heading. In this example, the message entered is “This is a tcp echo example\n\r” with a total length of 28 characters.

If you go back to the terminal window, it also indicates the same number of characters it receives from the remote host.



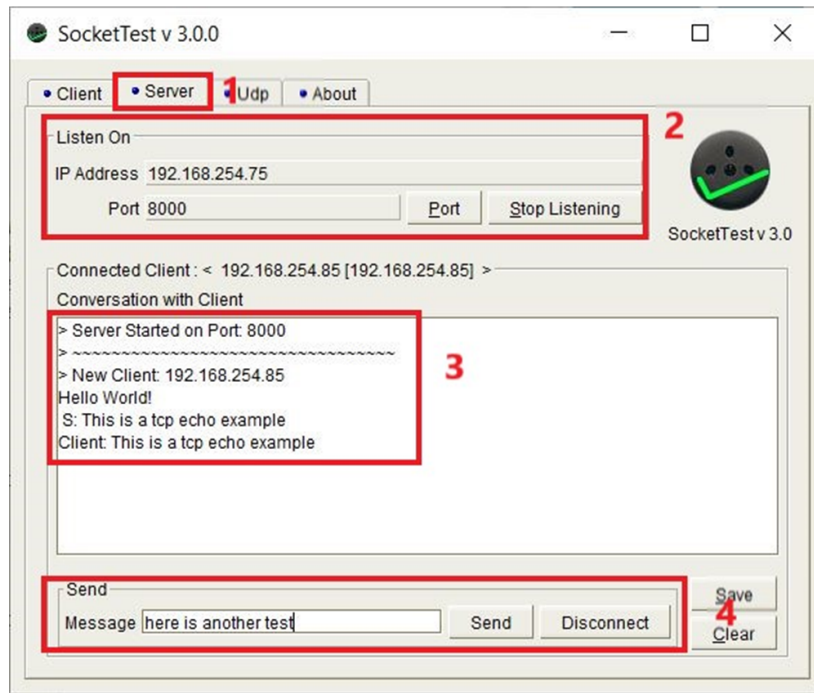


Figure 10-1. SocketTest Server Configuration for Enet\_tcpecho\_client\_lwip

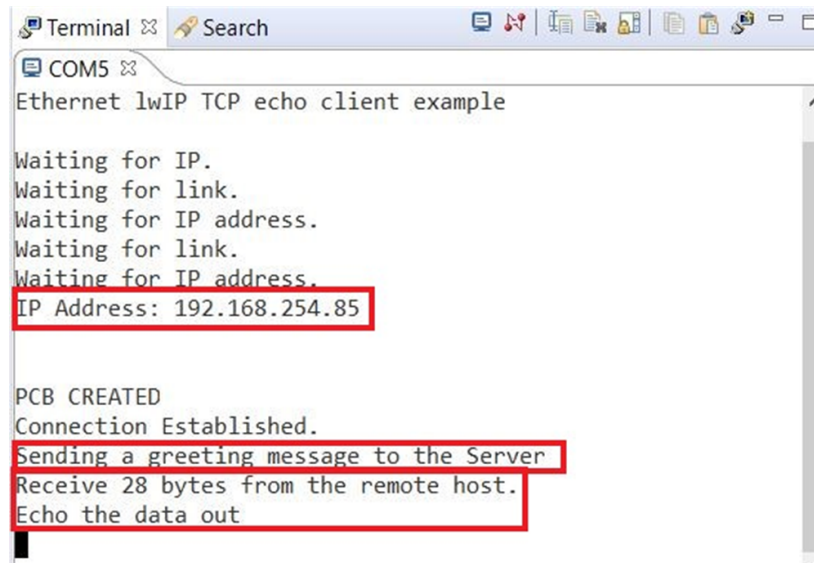


Figure 10-2. Enet\_tcpecho\_client\_lwip Output

### 10.3 Wireshark Capture for enet\_tcpecho\_client\_lwip Example

Figure 10-3 shows the Wireshark capture.

1. Wireshark provides various filtering capabilities. In this example, transactions are filtered that uses tcp port 8000, which is the port number configured for the SocketTest server.
2. As discussed previously, the TCP is a connection-based protocol. Here, you see the SYN segment sent by the client 192.168.254.85 to the server 192.168.254.75 to establish a connection and the ACK segment from the server to accept the connection.
3. After the connection is accepted, the client sends the “Hello World!\n\r” message. See box 3 at both the top and bottom on the capture. Notice the length of the message is 16 bytes which matches the total number characters in the message.
4. The server sends the message “This is a tcp echo example\n\r” for a total length of 28 bytes.
5. The client echoes back the message “Client: This is a tcp echo example\n\r” for a total length of 36 bytes.

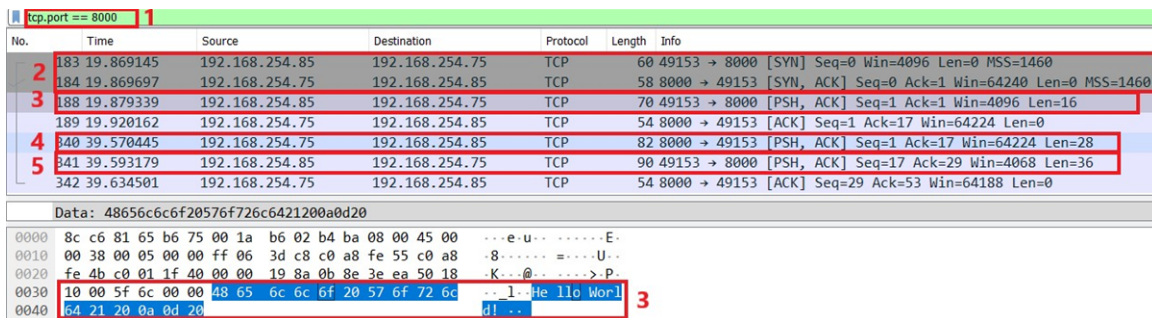


Figure 10-3. Client Server Wireshark Capture for Enet\_tcpecho\_client\_lwip

## 11 Enet\_adcsensor\_client\_lwip Example Overview

The enet\_adcsensor\_client\_lwip example is similar to the enet\_tcpecho\_client\_lwip example as a client application.

In this example, the intention is to measure the on-chip temperature with the ADC module and to periodically send the measured data to the server.

### 11.1 Run the adcsensor\_client\_lwip Example

Running the example will display the measured on-chip temperature on the terminal window as well as sending the measured temperature to the server.

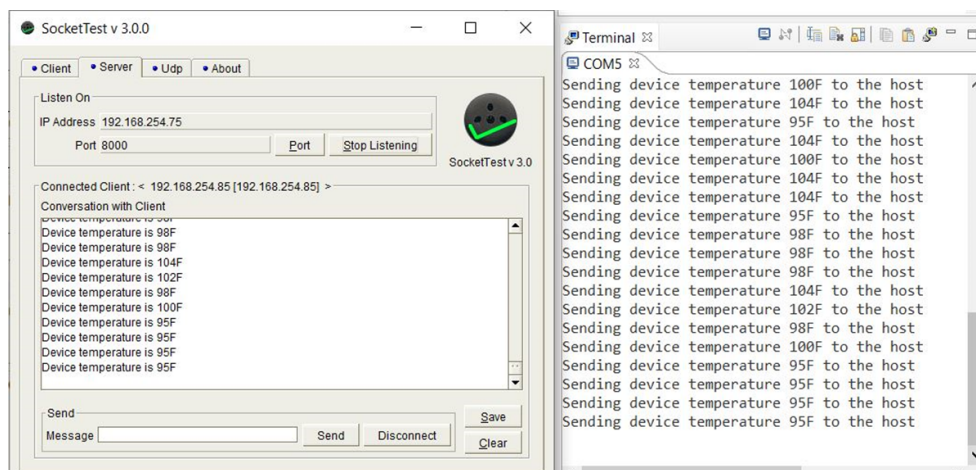


Figure 11-1. Client to Server Wireshark Capture for Enet\_adcsensor\_lwip

## 12 Enet\_udpecho\_client\_lwip Example Overview

The enet\_udpecho\_client\_lwip example is a client application using UDP protocol. It is similar to the server example in terms of the APIs as depicted in the UDP flowchart in Figure 1-2. The difference is that the client needs to call the udp\_connect() to first associate the PCB with the remote address. This example will use udp\_send() to send a greeting message “Hello! Greeting from EK-TM4C1294XL LaunchPad\n\r” to the remote server. It will then echo anything that it receives from the server.

### 12.1 Run the enet\_udpecho\_client\_lwip Example

For more information on how to setup the SocketTest for UDP testing, see Section 7.1. The SocketTest in Figure 12-1 shows the two messages sent and received by the server. The length of the two messages is displayed in the terminal window. You can cross-reference the messages captured by Wireshark in Figure 12-2. First, the greetings message is sent. Then the two echo messages, with their received lengths, are sent by the client.

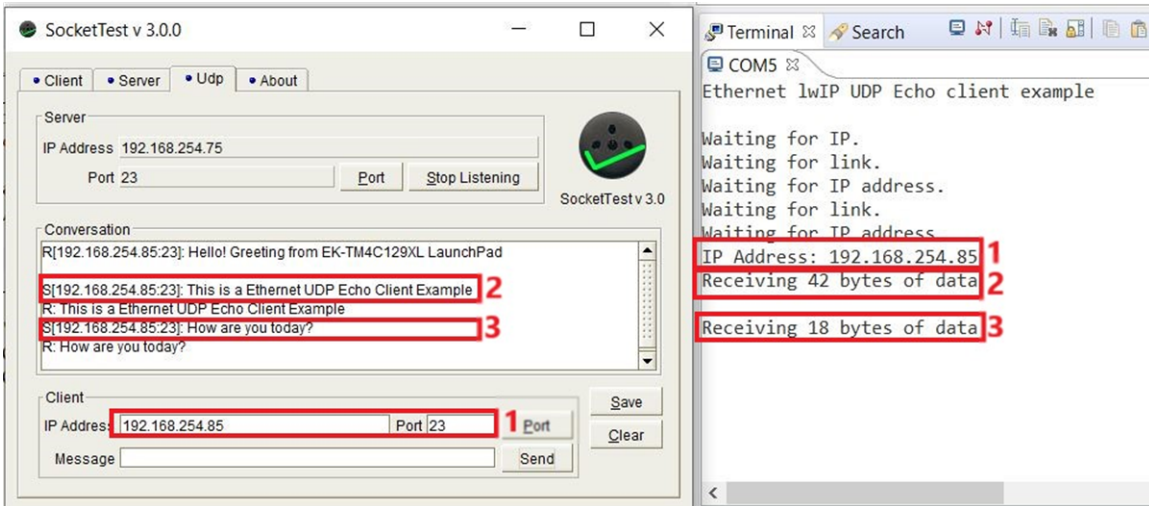


Figure 12-1. Enet\_udpecho\_client\_lwip Output

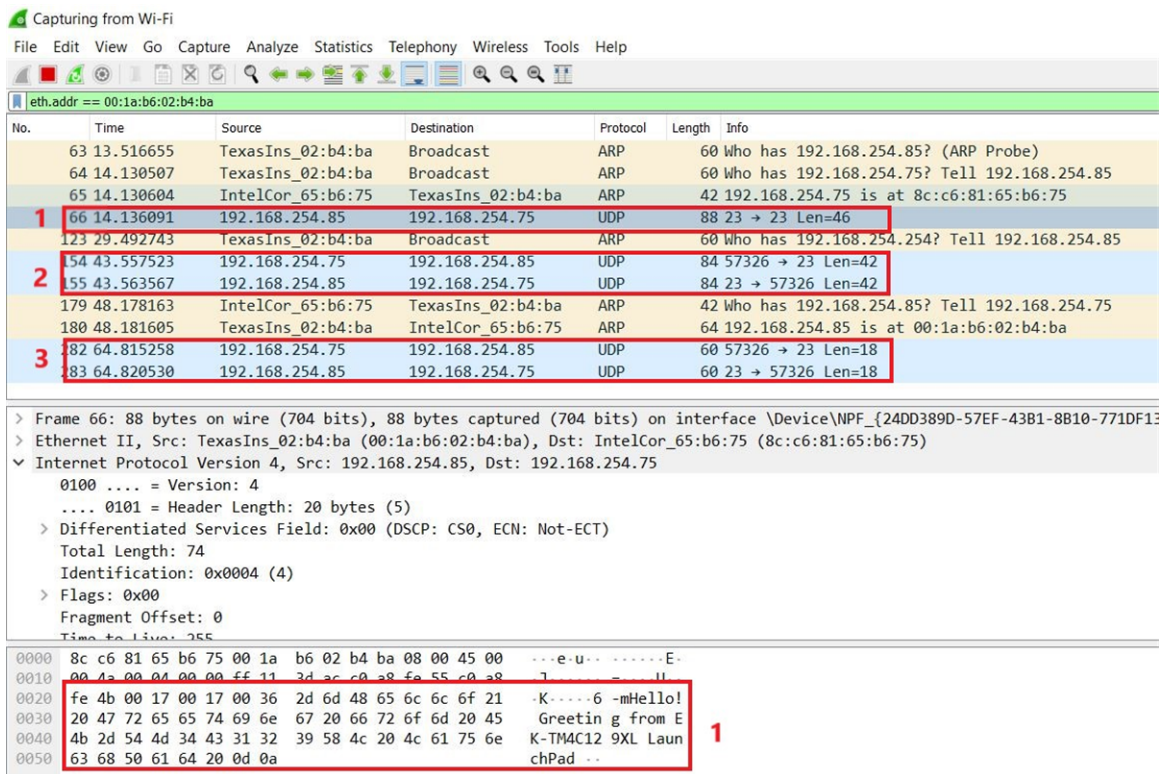


Figure 12-2. Client Server Wireshark Capture for Enet\_udpecho\_client\_lwip

### 13 References

- Texas Instruments: [Tiva C Series TM4C1294 Connected LaunchPad Evaluation Kit User's Guide](#)
- Texas Instruments: [TM4C1294NCPDT Snowflake Data Sheet](#)
- Texas Instruments: [TivaWare™ Peripheral Driver Library User's Guide](#)

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](http://ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2022, Texas Instruments Incorporated