International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 2, February 2014)

Implementation and Porting of Light Weight TCP/IP for Embedded Web Server (EWS)

HARIKRISHNA MUSINADA
Professor & HOD, ECE
RVR Institute of Engineering & Technology
Hyderabad, India
yemhechkay@gmail.com

G. Sravanthi
Student - M.Tech (Embedded Systems)
RVR Institute of Engineering & Technology
Hyderabad, India
sravanthisuman@gmail.com

ABSTRACT: **The development trend of embedded technology need the web/server technology applies into embedded fields and provides a flexible remote device monitoring and management function based on Internet browser. But, due to the limitation of hardware resource and the low-efficiency of general purpose TCP/IP protocol stacks and protocol models, it is quite difficult to implement full TCP/IP protocol into embedded system when accessing to Internet. This project analyses the Light-Weight TCP/IP and gives the detailed processing of the data happening at every layer of the stack. Embedded Ethernet development board from Silicon labs is used as hardware platform and real time kernel µC/OS-II as software platform. Implementation of web server involves two major challenges, first to port the kernel onto the board and secondly port the LwIP stack making use of OS real time kernel services. This includes writing the operating abstraction layer, timer and Ethernet driver for packet handling. A thin web server is then designed using the LwIP and the state transform of client and server when they were communicating was analyzed. At last, the EWS was tested on the home automation system where the different appliances connected to the target board can be monitored and controlled remotely. The result indicated the EWS can long-distance monitor the devices real-timely and perfectly. The advantages of this EWS are low cost, visualization, platform independent, flexible deployment, excellent remote accessing, etc. The equipments can be monitored and controlled flexibly in web pages through embedded web server. In industry control field, the using of embedded web server on intelligence device, instrument and sensor to realize flexible remote control has very high theoretical and application value.**

Key words: **Ping, TCP/IP, EWS, LwIP, Layer, OS, Stack, Protocol, ROM, API**

1. INTRODUCTION

The development trend of embedded technology need the web/server technology applies into embedded fields and provides a flexible remote device monitoring and management function based on Internet browser. But, due to the limitation of hardware resource and the low-efficiency of general purpose TCP/IP protocol stacks and protocol models, it is quite difficult to implement full TCP/IP protocol into embedded system when accessing to Internet. This project analyses the Light-Weight TCP/IP and gives the detailed processing of the data happening at every layer of the stack and this stack consists of limited number of protocols in each layer. Embedded target development board from Silicon labs is used as hardware platform and real time kernel µC/OS-II as software platform. Implementation of web server involves two major challenges, first to port the kernel onto the board and secondly port the lwIP stack making use of OS real time kernel services. This includes writing the operating abstraction layer, timer and Ethernet driver for packet handling. A thin web server is then designed using the lwIP and the state transform of client and server when they were communicating was analyzed. At last, the EWS was tested on the industrial automation system consisting of heat sensor and cooling device connected to the target board to monitor and control remotely.

The advantages of this EWS are low cost, visualization, platform independent, flexible deployment, excellent remote accessing, etc. The equipments can be monitored and controlled flexibly through web pages through embedded web server. In industry control field, the using of embedded web server on intelligence device, instrument and sensor to realize flexible remote control has very high theoretical and application value.

The objective of this paper is to build an embedded web server, which allows users to monitor and control their embedded applications using any standard browser. Implementation of web server involves two major challenges, first to port the kernel onto the target board and secondly port the LwIP stack making use of OS real time kernel services. This includes writing the operating abstraction layer, timer and Ethernet driver for packet handling. Understand the Real Time µC/OS-II concepts, porting the OS onto a 8051 Microcontroller and then testing the real time capabilities of the kernel with a real time application. The real time application for this paper is to monitor the industrial process temperature. In this paper each and every layer of the TCP/IP protocol suite is implemented as shown in the below figure 1.2, Physical layer and Data link layer are implemented on the Hardware and network layer, Transport layer and application layer on the software.

APPLICATION LAYER (HTTP)
TRANSPORT LAYER (TCP)
NETWORK LAYER (IP, ICMP, ARP)
DATALINK LAYER (CP2200 ETHERNET CONTROLLER)
PHYSICAL LAYER (CROSS OVER CABLE)

Figure.1. TCP/IP Layers

In this paper Data link layer is implemented on the Ethernet controller chip cp2200 and for implementing Network layer and Transport layer we use LwIP and HTTP protocol in the application layer.

2. LwIP STACK

The traditional Internet web/server is the fat server/thin client, this mode is perfect when translate and store abundance of data, but doesn't behave excellent in embedded field. And the Internet has become one of the most important basic information facilities in the world, the WWW service it offers has become one of the fastest growing and widest applied service, which have a great deal of advantages such as visualization, easy remote accessing, multi data format supporting, platform independent and thin client, etc. Connecting the embedded device to the Internet, implementing perfect Web service on it, and thus realizing a flexible remote monitoring and management through Internet browser has already become an inevitable development trend of embedded Technology. But, due to the limitation of hardware resource and the low-efficiency of general purpose TCP/IP protocol stacks and protocol models, it is quite difficult to implement full TCP/IP protocol into embedded system when accessing to Internet. Therefore, we need to port a subnet of TCP/IP into the embedded system.

2.1. Protocol layering: The protocols in the TCP/IP suite are designed in a layered fashion, where each protocol layer solves a separate part of the communication problem. This layering can serve as a guide for designing the implementation of the protocols, in that each protocol can be implemented separately from the other. Implementing the protocols in a strictly layered way can however, lead to a situation where the communication overhead between the protocol layers degrades the overall performance. To overcome these problems, certain internal aspects of a protocol can be made known to other protocols. Care must be taken so that only the important information is shared among the layers. Most TCP/IP implementations keep a strict division between the application layer and the lower protocol layers, whereas the lower layers can be more or less interleaved. In most operating systems, the lower layer protocols are implemented as a part of the operating system kernel with entry points for communication with the application layer process. The application program is presented with an abstract view of the TCP/IP implementation, where network communication drivers only very little from inter-process communication o. The implication of this is that since the application program is unaware of the buffer mechanisms used by the lower layers, it cannot utilize this information to, e.g., reuse buffers with frequently used data. Also, when the application sends data, this data has to be copied from the application process' memory space into internal buffers before being processed by the network code.

The operating systems used in minimal systems such as the target system of LwIP most often do not maintain a strict protection barrier between the kernel and the application processes. This allows using a more relaxed scheme for communication between the application and the lower layer protocols by the means of shared memory. In particular, the application layer can be made aware of the buffer handling mechanisms used by the lower layers. Therefore, the application can more efficiently reuse buffers. Also, since the application process can use the same memory as the networking code the application can read and write directly to the internal buffers, thus saving the expense of performing a copy. As in many other TCP/IP implementations, the layered protocol design has served as a guide for the design of the implementation of lwIP. Each protocol is implemented as its own module, with a few functions acting as entry points into each protocol. Even though the protocols are implemented separately, some layer violations are made, as discussed above, in order to improve performance both in terms of processing speed and memory usage. For example, when verifying the checksum of an incoming TCP segment and when demultiplexing a segment, the source and destination IP addresses of the segment has to be known by the TCP module. Instead of passing these addresses to TCP by the means of a function call, the TCP module is aware of the structure of the IP header, and can therefore extract this information by itself. lwIP consists of several modules. Apart from the modules implementing the TCP/IP protocols (IP, ICMP, UDP, and TCP) a number of support modules are implemented. The support modules consists of the operating system emulation layer , the buffer and memory management subsystems , network interface functions and functions for computing the Internet checksum.

2.2. Process model: The process model of a protocol implementation describes in which way the system has been divided into different processes. One process model that has been used to implement communication protocols is to let each protocol run as a standalone process. With this model, a strict protocol layering is enforced, and the communication points between the protocols must be strictly divided. While this approach has its advantages such as protocols can be added at runtime, understanding the code and debugging is generally easier, there are also disadvantages. The strict layering is not, as described earlier, always the best way to implement protocols. Also, and more important, for each layer crossed, a context switch must be made. For an incoming TCP segment this would mean three context switches, from the device driver for the network interface, to the IP process, to the TCP process and finally to the application process. In most operating systems a context switch is fairly expensive.

Another common approach is to let the communication protocols reside in the kernel of the operating system. In the case of a kernel implementation of the communication protocols, the application processes communicate with the protocols through system calls. The communication protocols are not strictly divided from each other but may use the

International Journal of Ethics in Engineering & Management Education
Website: www.ijeee.in (ISSN: 2348-4748, Volume 1, Issue 2, February 2014)

techniques of crossing the protocol layering. LwIP uses a process model in which all protocols reside in a single process and are thus separated from the operating system kernel. Application programs may either reside in the LwIP process, or be in separate processes. Communication between the TCP/IP stack and the application programs are done either by function calls for the case where the application program shares a process with LwIP, or by the means of a more abstract API. Having LwIP implemented as a user space process rather than in the operating system kernel has both its advantages and disadvantages. The main advantage of having lwIP as a process is that is portable across different operating systems. Since LwIP is designed to run in small operating systems that generally do not support neither swapping out processes not virtual memory, the delay caused by having to wait for disk activity if part of the LwIP process is swapped or paged out to disk will not be a problem. The problem of having to wait for a scheduling quantum before getting a chance to service requests still is a problem however, but there is nothing in the design of LwIP that precludes it from later being implemented in an operating system kernel.

2.3. The operating system emulation layer: In order to make LwIP portable, operating system specific function calls and data structures are not used directly in the code. Instead, when such functions are needed the operating system emulation layer is used. The operating system emulation layer provides a uniform interface to operating system services such as timers, process synchronization, and message passing mechanisms. In principle, when porting LwIP to other operating systems only an implementation of the operating system emulation layer for that particular operating system is needed. The operating system emulation layer provides a timer functionality that is used by TCP. The timers provided by the operating system emulation layer are one-shot timers with a granularity of at least 200 ms that calls a registered function when the time-out occurs. The only process synchronization mechanism provided is semaphores. Even if semaphores are not available in the underlying operating system they can be emulated by other synchronization primitives such as conditional variables or locks.

The message passing is done through a simple mechanism which uses an abstraction called mailboxes. A mailbox has two operations: post and fetch. The post operation will not block the process; rather, messages posted to a mailbox are queued by the operating system emulation layer until another process fetches them. Even if the underlying operating system does not have native support for the mailbox mechanism, they are easily implemented using semaphores.

2.4. Buffer and memory management: The memory and buffer management system in a communication system must be prepared to accommodate buffers of very varying sizes, ranging from buffers containing full-sized TCP segments with several hundred bytes worth of data to short ICMP echo replies consisting of only a few bytes. Also, in order to avoid copying it should be possible to let the data content of the buffers reside in memory that is not managed by the networking subsystem, such as application memory or ROM.

2.5. IP processing: LwIP implements only the most basic functionality of IP. It can send, receive and forward packets, but cannot send or receive fragmented IP packets nor handle packets with IP options. For most applications this does not pose any problems.

2.6. TCP processing: TCP is a transport layer protocol that provides a reliable byte stream service to the application layer. TCP is more complex than the other protocols described here, and the TCP code constitutes 50% of the total code size of lwIP. The basic TCP processing is divided into six functions; the functions tcp_input (), tcp_process (), and tcp receive () which are related to TCP input processing, and tcp write (), tcp_enqueue (), and tcp_output () which deals with output processing.

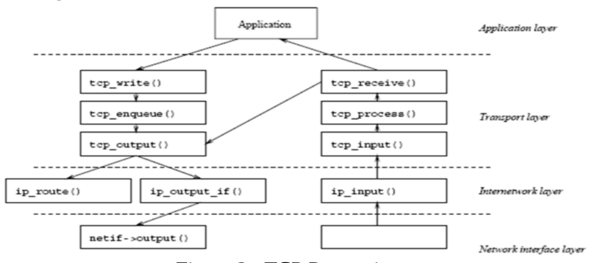


Figure.2. TCP Processing

When an application wants to send TCP data, tcp write () is called. The function tcp write ( ) passes control to tcp enqueue() which will break the data into appropriate sized TCP segments if necessary and put the segments on the transmission queue for the connection. The function tcp_output() will then check if it is possible to send the data, i.e., if there is enough space in the receiver's window and if the congestion window is large enough and if so, sends the data using ip_route() and ip_output if(). Input processing begins when ip_input () after verifying the IP header hands over a TCP segment to tcp input (). In this function the initial sanity checks (i.e., check summing and TCP options parsing) are done as well as deciding to which TCP connection the segment belongs. The segment is then processed by tcp_process(), which implements the TCP state machine, and any necessary state transitions are made. The function tcp_receive () will be called if the connection is in a state to accept data from the network. If so, tcp_receive () will pass the segment up to an application program. If the segment constitutes an ACK for unacknowledged (thus previously buffered) data, the data is removed from the buffers and its memory is reclaimed. Also, if an ACK for data was received the receiver might be willing to accept more data and therefore tcp output() is called.

2.7. Application Program Interface (API): The Application Program Interface (API) defines the way the application program interacts with the TCP/IP stack. The most commonly used API for TCP/IP is the BSD socket API which is used in most UNIX systems and has heavily influenced the Microsoft Windows WinSock API. Because the socket API uses stop-and wait semantics, it requires support from an underlying multitasking operating system. Since the overhead of task management, context switching and allocation of stack space for the tasks might be too high in the intended LwIP target architectures, the BSD socket interface is not suitable for our purposes. lwIP provides two APIs to programmers: proto sockets, a BSD socket-like API without the overhead of full multi-threading, and a "raw" event-based API that is more low-level than proto sockets but uses less memory.

3. HTTP PROTOCOL IN APPLICATION LAYER

In this paper http protocol is followed in the application layer to communicate with the clients. Http runs on the TCP port 80 and it is the standard followed by all the servers. Hypertext Transfer Protocol (HTTP) is the communication protocol used to exchange information between a client system and a Web server across a TCP/IP connection. The interchange is generally referred to as an HTTP transaction. In an HTTP transaction, the client system opens a TCP connection with the Web server and submits an HTTP request. The Web server, in turn, issues an HTTP response, completing one HTTP transaction. HTTP is stateless, in that there is no provision in the protocols design for information about any single request persisting beyond one transaction. A client connects to the server at port 80 and sends a request. The request line from the client consists of a request method, the address of the file requested and the HTTP version number. GET /HTTP/1.1 the response line contains information on the HTTP version number, a status code that indicates the result of the request from the client and a description of the status code in 'English'. HTTP/1.1 200 OK There are many status codes for each and every action. For example if the requested page was not found in the server than “404 error” will be served to the client.

3.1. HTTP Request Methods:

- GET Method: The Get method is used to getting the data from the server. Get method appends the parameters passed as query string to a URL, in the form of key-value pairs. for example, if a parameter is name = Williams, then this string will be appended in the URL. By default the method is Get.
- POST Method: The post method is used for sending data to the server. In post method the query string is appended along the request object, they do not get appended in the URL, so parameters transfer in hidden form.
- HEAD Method: When a user wants to know about the headers, like MIME types, char set, Content- Length then we use Head method. With this nobody content is returned.
- These three are commonly used methods. While Get and Post methods are most widely used. There are more methods of http protocols which are rarely used by they have been given here for your knowledge.
- TRACE Method: Trace on the resource returns the content of the resource. Asks for a loopback of the request message, so that the use can see what is being received on the other side.
- DELETE Method: It is used for delete the resources, files at the requested URL
- OPTIONS Method: It lists the Http methods to which the thing at the requested URL can respond.
- PUT Method: It put the enclosed information at the requested URL.
- CONNECT Method: It connects for the purpose of tunneling.

3.2. TCP Connection Establishment Process: The "Three-Way Handshake" As http uses TCP in the transport layer for communicating with the client we need to establish a connection prior to the http transaction and this method is popularly known as three way hand shake. To establish a connection, each device must send a SYN and receive an ACK for it from the other device. Thus, conceptually, we need to have four control messages pass between the devices. However, it's inefficient to send a SYN and an ACK in separate messages when one could communicate both simultaneously. Thus, in the normal sequence of events in connection establishment, one of the SYNs and one of the ACKs is sent together by setting both of the relevant bits (a message sometimes called a SYN+ACK). This makes a total of three messages, and for this reason the connection procedure is called a three-way handshake.

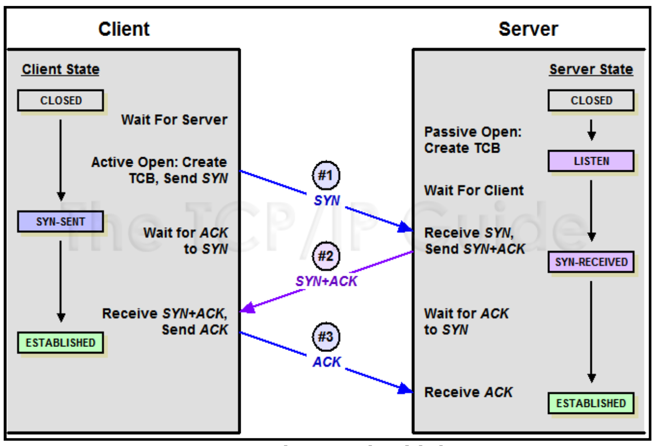


Figure.3. Three-way handshake.

3.3. TCP Connection Termination: In the normal case, each side terminates its end of the connection by sending a special message with the FIN (finish) bit set. This message, sometimes called a FIN, serves as a connection termination request to the other device, while also possibly carrying data like a regular segment. The device receiving the FIN responds with an acknowledgment to the FIN to indicate that it was received. The connection as a whole is not considered terminated until both sides have finished the shut down procedure by sending a FIN and receiving an ACK. Thus, termination isn't a three-way handshake like establishment: it is a pair of two-way handshakes. The states that the two devices in the connection move through during a normal connection shutdown are different because the device initiating the shutdown must behave differently than the one that receives the termination request. In particular, the TCP on the device receiving the initial termination request must inform its application process and wait for a signal that the process is ready to proceed.

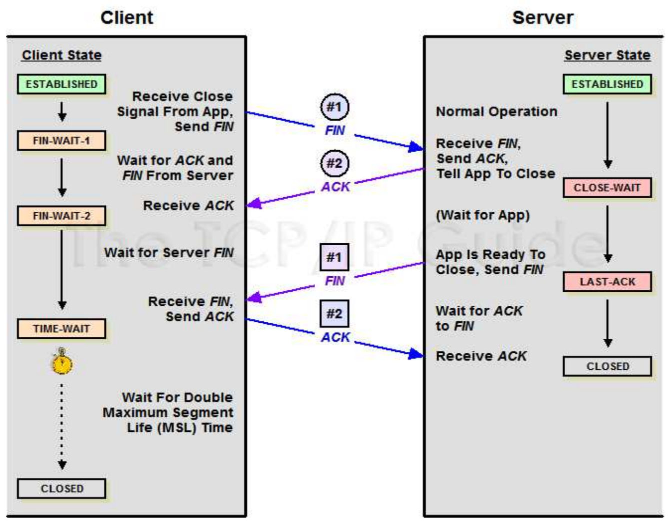
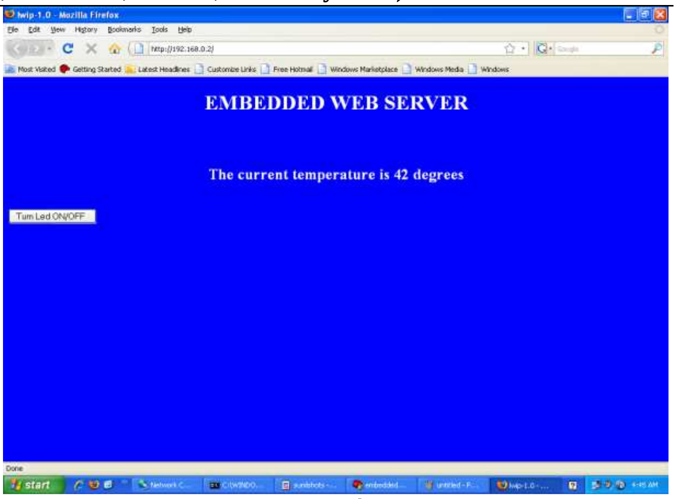


Figure.4. Four way handshake

4. TESTING AND RESULTS

The developed embedded web server is tested by opening any standard web browser like Mozilla Firefox and internet explorer and typing the embedded web server address i,e. http://192.168.0.2 and a web page is obtained on the browser window with the temperature data in it and with a button to monitor the devices as shown in figure 5, Using ping command the connectivity between the client and the server is verified and we can find that four requests are sent from the client and the same number of replies is received from the server as shown in the figure 7.

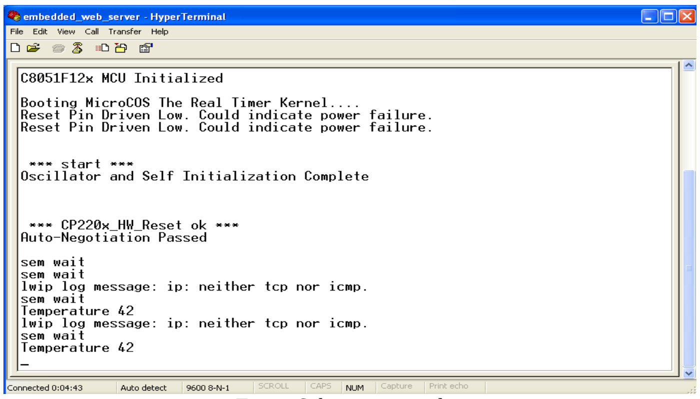
4.1. Wire shark: Wire shark is the open source packet analyzer by which we can check packet traffic which are coming and going in to the system. By using this tool we have verified the packets flow between the EWS and the client and the obtained results are shown in the figure 8. In this we can find the TCP packets traffic between the client and the server and the http packets sent and received from the server.


lwip-1.0 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.0.2/
Turn Led ON/OFF
Done
start

EMBEDDED WEB SERVER

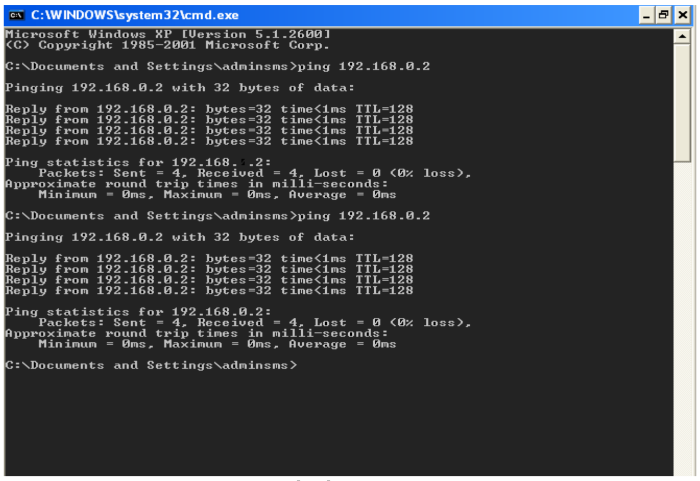
The current temperature is 42 degrees

Figure.5. web page


embedded_web_server - HyperTerminal
File Edit View Call Transfer Help
Connected 0:04:43
Auto detect
9600 8-N-1
SCROLL
CAPS
NUM
Capture
Print echo

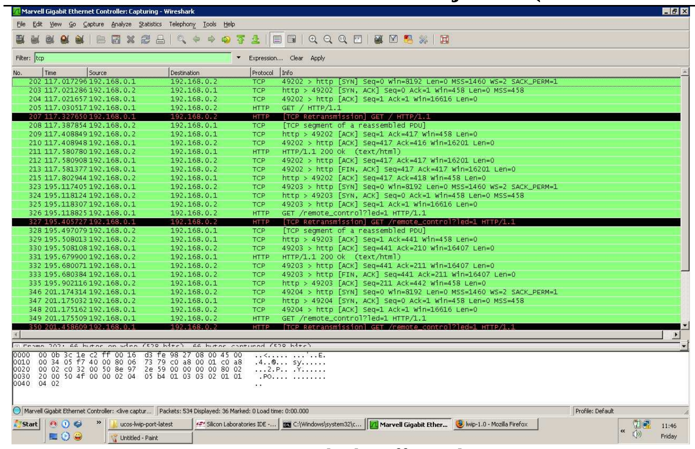
```
C8051F12x MCU Initialized

Booting MicroCOS The Real Timer Kernel....
Reset Pin Driven Low. Could indicate power failure.
Reset Pin Driven Low. Could indicate power failure.


 *** start ***
Oscillator and Self Initialization Complete



 *** CP220x_HW_Reset ok ***
Auto-Negotiation Passed

sem wait
sem wait
lwip log message: ip: neither tcp nor icmp.
sem wait
Temperature 42
lwip log message: ip: neither tcp nor icmp.
sem wait
Temperature 42
_
```

Figure.6. hyper terminal



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\adminsms>ping 192.168.0.2

Pinging 192.168.0.2 with 32 bytes of data:

Reply from 192.168.0.2: bytes=32 time<1ms TTL=128
Reply from 192.168.0.2: bytes=32 time<1ms TTL=128
Reply from 192.168.0.2: bytes=32 time<1ms TTL=128
Reply from 192.168.0.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168. .2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\adminsms>ping 192.168.0.2

Pinging 192.168.0.2 with 32 bytes of data:

Reply from 192.168.0.2: bytes=32 time<1ms TTL=128
Reply from 192.168.0.2: bytes=32 time<1ms TTL=128
Reply from 192.168.0.2: bytes=32 time<1ms TTL=128
Reply from 192.168.0.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\adminsms>
```

Figure.7. checking connectivity


Marvell Gigabit Ethernet Controller: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: tcp
Expression... Clear Apply
Start
ucos-lwip-port-latest
Untitled - Paint
Silicon Laboratories IDE -...
C:\Windows\system32\c...
Marvell Gigabit Ether...
lwip-1.0 - Mozilla Firefox
11:46
Friday

```
No.  Time       Source       Destination  Protocol Info
202 117.017296 192.168.0.1  192.168.0.2  TCP      49202 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2 SACK_PERM=1
203 117.021286 192.168.0.2  192.168.0.1  TCP      http > 49202 [SYN, ACK] Seq=0 Ack=1 Win=458 Len=0 MSS=458
204 117.021657 192.168.0.1  192.168.0.2  TCP      49202 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
205 117.030517 192.168.0.1  192.168.0.2  HTTP     GET / HTTP/1.1
207 117.327650 192.168.0.1  192.168.0.2  HTTP     [TCP Retransmission] GET / HTTP/1.1
208 117.387854 192.168.0.2  192.168.0.1  TCP      [TCP segment of a reassembled PDU]
209 117.408849 192.168.0.2  192.168.0.1  TCP      http > 49202 [ACK] Seq=1 Ack=417 Win=458 Len=0
210 117.408948 192.168.0.1  192.168.0.2  TCP      49202 > http [ACK] Seq=417 Ack=416 Win=16201 Len=0
211 117.580780 192.168.0.2  192.168.0.1  HTTP     HTTP/1.1 200 Ok  (text/html)
212 117.580908 192.168.0.1  192.168.0.2  TCP      49202 > http [ACK] Seq=417 Ack=417 Win=16201 Len=0
213 117.581377 192.168.0.1  192.168.0.2  TCP      49202 > http [FIN, ACK] Seq=417 Ack=417 Win=16201 Len=0
215 117.802944 192.168.0.2  192.168.0.1  TCP      http > 49202 [ACK] Seq=417 Ack=418 Win=458 Len=0
323 195.117405 192.168.0.1  192.168.0.2  TCP      49203 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2 SACK_PERM=1
324 195.118124 192.168.0.2  192.168.0.1  TCP      http > 49203 [SYN, ACK] Seq=0 Ack=1 Win=458 Len=0 MSS=458
325 195.118307 192.168.0.1  192.168.0.2  TCP      49203 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
326 195.118825 192.168.0.1  192.168.0.2  HTTP     GET /remote_control?led=1 HTTP/1.1
327 195.405727 192.168.0.1  192.168.0.2  HTTP     [TCP Retransmission] GET /remote_control?led=1 HTTP/1.1
328 195.497079 192.168.0.2  192.168.0.1  TCP      [TCP segment of a reassembled PDU]
329 195.508013 192.168.0.2  192.168.0.1  TCP      http > 49203 [ACK] Seq=1 Ack=441 Win=458 Len=0
330 195.508108 192.168.0.1  192.168.0.2  TCP      49203 > http [ACK] Seq=441 Ack=210 Win=16407 Len=0
331 195.679900 192.168.0.2  192.168.0.1  HTTP     HTTP/1.1 200 Ok  (text/html)
332 195.680071 192.168.0.1  192.168.0.2  TCP      49203 > http [ACK] Seq=441 Ack=211 Win=16407 Len=0
333 195.680384 192.168.0.1  192.168.0.2  TCP      49203 > http [FIN, ACK] Seq=441 Ack=211 Win=16407 Len=0
335 195.902116 192.168.0.2  192.168.0.1  TCP      http > 49203 [ACK] Seq=211 Ack=442 Win=458 Len=0
346 201.174314 192.168.0.1  192.168.0.2  TCP      49204 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2 SACK_PERM=1
347 201.175032 192.168.0.2  192.168.0.1  TCP      http > 49204 [SYN, ACK] Seq=0 Ack=1 Win=458 Len=0 MSS=458
348 201.175162 192.168.0.1  192.168.0.2  TCP      49204 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
349 201.175509 192.168.0.1  192.168.0.2  HTTP     GET /remote_control?led=1 HTTP/1.1
350 201.458609 192.168.0.1  192.168.0.2  HTTP     [TCP Retransmission] GET /remote_control?led=1 HTTP/1.1

0000  00 0b 3c 1e c2 ff 00 16  d3 fe 98 27 08 00 45 00   ..<..... ...'..E.
0010  00 34 05 f7 40 00 80 06  73 79 c0 a8 00 01 c0 a8   .4..@... sy......
0020  00 02 c0 32 00 50 8e 97  2e 59 00 00 00 00 80 02   ...2.P.. .Y......
0030  20 00 50 4f 00 00 02 04  05 b4 01 03 03 02 01 01    .PO.... ........
0040  04 02                                              ..

Marvell Gigabit Ethernet Controller: <live captur...   Packets: 534 Displayed: 36 Marked: 0 Load time: 0:00.000   Profile: Default
```

Figure.8. Wire-shark traffic analyzer

5. CONCLUSION

Using PING command to link EWS in local area network, we can get four response data packages and the time it used is less than 20ms and there is no data package lost. When we input the IP address of embedded web server, we can open the web page through the browser quickly and correctly. From the obtained web page we can monitor the temperature reading and control the LED. Thus we can believe the EWS can long-distance monitor and control the devices real-timely and perfectly.

REFERENCES

1. "The Porting and Implementation of Light-Weight TCP/IP for Embedded Web Server" IEEE-2008, WEI CHEN, SHU-BO QIU, YING-CHUN ZHANG. Department of Automation of Shandong Institute of Light Industry, Jinan, China.
2. "Design and implementation of the LWIP TCP/IP Stack" by Adam Dunkels.
3. "*µC/OS-II, the Real-Time Kernel*" by Jean J. Labrosse.
4. C8051F120TB manual from silicon laboratories
5. RFC-2616 HTTP/1.1

About the authors:

Professor **HARIKRISHNA MUSINADA** received Bachelor of Engineering and M.Tech degrees in ECE from Marathwada University, Aurangabad and JNTU-Hyderabad. He is currently Professor in ECE Department of RVR Institute of Engineering & Technology, Hyderabad and pursuing Ph.D degree at Department of ECE OU-Hyderabad. He has 8 Research papers into his credit published in various International Journals, Magazines and Conference Proceedings. He is an active life member of professional bodies like Indian Society for Technical Education **(MISTE),** Institution of Electronics and Telecommunication Engineers **(MIETE)**, Society of EMC Engineers (INDIA) - **SEMCE (I).** Secured **Best Teacher Award** in the course of teaching and inspiring many students in the academics. He has conducted many conferences, workshops, short term courses and was convener for many technical symposiums in the Engineering colleges he worked. He was Co-Chairman to one of the technical sessions of **2nd International Conference** on Innovations in Electronics and Communication Engineering (ICIECE) organized by ECE Department of Guru Nanak Institutions Technical Campus (GNITC) on 9-10 August, 2013 in association with IETE, ISTE, CSI and BESI. He is an Associate Editor for International Journal of Pure Research in Engineering and Technology (**IJPRET**) and Governing Body Member for International Journal of Ethics in Engineering and management Education (**IJEEE**). His currently research interests include Mixed Signal VLSI design, Bio Technology with Signal Processing.

G. Sravanthi Received B.Tech Degree in Electronics and Communication Engineering from the University of JNTU And M. Tech Studying In The University Of JNTU Hyderabad .Up to Now Attended Several National And International Conferences, Workshops Research Development Programs. Research Interested In Embedded Systems and Attended Several Faculty Development Programs.