STEP BY STEP IMPLEMENTATION OF ETHERNET AND TCP-IP PROTOCOLS ON AN EMBEDDED SYSTEM - Part II

Tariqul Haque, PhD
Electrical Engineering Technology
Southeastern Community College
Whiteville, NC 28472

Michael Urbaniak
Ebtron, Inc
Loris, SC 29569

ABSTRACT

In Part II of this paper we describe testing of the ethernet system we designed in Part I[1], which requires such communication protocols as ARP and ICMP. As an application we present a new approach to investigate the fields of IP and TCP protocols to create a basic working HTTP server on an embedded system. The server performs equally well with browsers under a Microsoft Windows or a Linux host.

INTRODUCTION

When it comes to implementation of communication protocols, most authors approach the problem by explaining properties of the fields of protocols. This is appropriate when software is developed in a personal or higher computer with a powerful operating system to support it. But when software is developed for an embedded system, instead of field-level explanation, byte-level explanation will be more appropriate for programmers. This will allow programmers to pursue their effort in C or C++ and also in Assembly language, if necessary. First we describe testing of ethernet hardware; then, we present detailed descriptions on how an embedded system with ethernet capability can serve HTML files.

ADDRESS RESOLUTION PROTOCOL (ARP)

Connection between two computers by ethernet is commonly tested by an application called 'ping'. An IP address (Internet Protocol) of the destination computer is supplied on the command line. Each computer with ethernet connection must have a 6-byte long hardware address (also known as Ethernet Address or hardware MAC number). In addition to an IP address the ping application also requires the hardware address of the destination computer. Since the source computer is not aware of the ethernet address of the destination computer, a pre-ping packet is sent to the destination computer in order to retrieve its ethernet address. This is known as an Address Resolution Protocol or ARP packet. It is also common for a computer to exchange ARP information in the background with other systems in a network without any user intervention. All headers and associated fields of protocols used in this paper are minimum requirements or in compact forms for successful communications between computers.[2] For this reason, the header structures and data fields of various protocols used in this paper may differ from those generally found in technical books or many websites on these topics.

A typical ARP packet, shown in block diagram, contains an ethernet header, an ARP header and padding bytes. All numbers used in the protocol sections are in Hexadecimal (Hex) unless indicated otherwise. Throughout this paper our embedded system is the destination and a computer communicating with our system will be termed as the host computer or source computer.

Ethernet Header

Destination Ethernet address *6 Bytes*	Source Ethernet address *6 Bytes*	Type/Length *2 Bytes*

ARP Header

Hardware Type 2 Bytes	Protocol Type 2 Bytes	MAC length 1 Byte	IP length 1 Byte	Operation 2 Bytes

Source Ethernet address 6 Bytes	Source IP address 4 Bytes	Destination Ethernet address 6 Bytes	Destination IP Address 4 bytes

Padding Bytes

All frames in hexadecimal values, shown in this paper, are taken from the MS Windows environment. The essential part of an ARP frame, received from a host computer, and polled from the ethernet controller chip is displayed byte by byte. The embedded system used in this paper has an ethernet hardware address of 00-C0-F0-34-AC-7B, and an IP address of 0A 0A 01 64 (10.10.1.100).

A Received ARP Frame

```
FF FF FF FF FF FF 00 40 33 2E 99 4F 08 06 00 01
08 00 06 04 00 01 00 40 33 2E 99 4F 0A 0A 01 02
00 00 00 00 00 00 0A 0A 01 64 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20
```

The frame is sub-divided into constituent headers, and bytes are also grouped to exhibit their significance as described in the block diagram of ARP packet above. Finally a response packet is formed from these analysis.

Ethernet Header

[1] FF FF FF FF FF FF [7] 00 40 33 2E 99 4F [13] 08 06

ARP Header

[15] 00 01 [17] 08 00 [19] 06 [20] 04 [21] 00 01 [23] 00 40 33 2E 99 4F
[29] 0A 0A 01 02

[33] 00 00 00 00 00 00 [39] 0A 0A 01 64

Padding Bytes

[43] 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20

Explanation of the bytes and groups of bytes follow. A decimal number adjacent to each box represents the position of the first byte in the box in the entire packet. The first 14 bytes are ethernet header.

[1] FF FF FF FF FF FF These 6 bytes are the ethernet address of the destination. An ethernet address of FF FF FF FF FF FF is called a broadcast address because the packet is accepted by all nodes on the network [8]. The response packet also uses FF FF FF FF FF FF as its destination ethernet address.

[7] 00 40 33 2E 99 4F These 6 bytes are the ethernet address of the source computer. Since roles of source and destination are reversed in the response packet, these 6 bytes are replaced with the ethernet address of the embedded system.

[13] 08 06 When this number is below 0600, it represents the length of the packet. A number above 0600 indicates the type of ethernet protocol used. Below are a few commonly used types[4]:

Code	Type
0800	Internet Protocol
0806	ARP
0835	Reverse ARP

The bytes that follow ethernet header are ARP header bytes.

[15] 00 01 These 2 bytes represent the hardware type used for this communication. For ethernet, this value is 00 01.[4] The same 2 bytes are returned in response.

[17] 08 00 These two bytes are used for type of protocol address. For IP protocol version 4 (IPv4) this value is 08 00. The same 2 bytes are returned in the response packet.

[19] 06 The length of ethernet address is 6 bytes; hence, a value of 06 is returned in this byte location.

[20]04 The length of IP address is 4 bytes. A value of 04 is returned in this byte position.

[21]00 01 These two bytes represent the type of operation is being requested.

00 01: ARP request operation
00 02: ARP response operation

Thus 00 02 must be returned by the embedded system in the return packet .

[23]00 40 33 2E 99 4F The source ethernet address, which will be returned as destination ethernet address.

[29]0A 0A 01 02 The source IP address, which will be returned as destination (host) IP address.

[33]00 00 00 00 00 00 This byte locations are reserved for the destination ethernet address. Since the source system is unaware of the destination Ethernet address, it sends 6 bytes of zeros. In the return packet the embedded software must replace these zeros with 00 40 33 2E 99 4F, the numbers received in positions 23-28.

[39]0A 0A 01 64 Four bytes of destination IP address. The address 0A 00 00 02, received in positions 29-32, will replace these 4 bytes in the return packet.

The remaining 18 bytes are padding bytes which are returned as they are in the return packet. With the explanations presented above we now can form the entire return packet as

[1]FF FF FF FF FF FF [7]00 C0 F0 34 AC 7B [13]08 06

[15]00 01 [17]08 00 [19]06 [20]04 [21]00 02 [23]00 C0 F0 34 AC 7B [29]0A 0A 01 64

[33]00 40 33 2E 99 4F [39]0A 0A 01 02

[43]20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20

Apart from interchanging the positions of ethernet and IP addresses of source and destination, the only other changes made are at byte positions 21 and 22. These 2 locations now have 00 02 to indicate that this is an ARP response. There are only 60 bytes shown here because 4 bytes of CRC are added by the ethernet controller chip to make a total of 64 bytes, the minimum byte size of an ethernet frame.

INTERNET CONTROL MESSAGE PROTOCOL (ICMP)

When a ping command is issued from a host, the ARP frames are exchanged. The source computer then transmits an ICMP frame with a few bytes of data to be returned exactly by the responder. Most Windows system use a 74-byte packet, but this packet length varies from operating system to operating system. The frame has two parts – protocol headers and data to be echoed back. A complete ICMP frame contains an Ethernet Header, an IP Header, an ICMP header, and some data.[2]

Ethernet Header

Destination Ethernet address *6 Bytes*	Source Ethernet address *6 Bytes*	Protocol Type *2 Bytes*

IP Header

Version +Header Length. *1 Byte*	Service *1 Byte*	Length of IP+TCP *2 Bytes*	Ident value *2 Bytes*	Flag, Fragment Offset. *2 Bytes*

Time to live *1 Byte*	Protocol *1 Byte*	Checksum of IP Header *2 Bytes*	Source IP Address *4 bytes*	Destination IP Address *4 Bytes*

ICMP Header

Type	Code	Checksum	ID Number	Sequence Number
1 Bytes	*1 Bytes*	*2 Bytes*	*2 Bytes*	*2 Bytes*

Echo Data
32 bytes

A typical ICMP packet received by an embedded system will contain hex bytes of the following values.

A Received ICMP (ping) Frame

```
00 C0 F0 34 AC 7B 00 40 33 2E 99 4F 08 00 45 00
00 3C CA 00 00 00 20 01 BA 47 0A 0A 01 02 0A 0A
01 64 08 00 47 5C 01 00 05 00 61 62 63 64 65 66
67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76
77 61 62 63 64 65 66 67 68 69
```

The ICMP packet can be sub-divided into individual protocol headers to explicitly demonstrate the constituents of a frame. Then we shift our focus to individual byte or group of bytes and their significance in order to form a response packet.

Ethernet Header

[1]00 C0 F0 34 AC 7B [7]00 40 33 2E 99 4F [13]08 00

IP Header

[15]45 [16]00 [17]00 3C [19]CA 00 [21]00 00 [23]20 [24]01 [25]BA 47 [27]0A 0A 01 02 [31]0A A0 01 64

ICMP Header

[35]08 00 [37]47 5C [39]01 00 [41]05 00

[43] Data to Return

```
61 62 63 64 65 66 67 68 69
6A 6B 6C 6D 6E 6F 70 71 72
73 74 75 76 77 61 62 63 64
65 66 67 68 69
```

First 14 bytes are Ethernet header.

[1]00 AC C0 F0 34 7B This is the 6-byte ethernet address of destination, in this case, the embedded system. The number is already stored in the ethernet chip during initialization, and is used by the chip for comparison and rejection or acceptance of a packet. The number should also be stored in RAM by embedded system software and retrieved for the response packet.

[7]00 40 33 2E 99 4F These 6 bytes are the hardware address of the host. These 6 bytes must be copied into return packet as destination hardware address.

[13]08 00 These 2 bytes represent protocol type or frame length. For internet protocol its value is 08 00 [1].

The next 20 bytes are the IP header.

[15]45 This byte represents the IP version and header length combined in a strange way. '4' of '45' is used for the IP version, IPV4.0. And '5' of '45' represents the length of IP header in 32-bit words (or 4 bytes). For applications in an embedded system, optional bytes in the IP header are generally dropped making a header of only 20 bytes. The IP header length then becomes five 32-bit words (or 20 bytes). This '45' is returned in the return package in the same byte location.

[16]00 This byte is called service byte - representing quality of service desired, such as delay, reliability, control, priority, etc. For normal operation this byte is 00 in the return packet.

[17]00 3C Two important bytes for total length. These 2 bytes represents the number of bytes included in the packet, starting from the first byte of the IP header to the last byte of the packet. In the above example, the number of bytes from 45 (byte position 15) to 69, is decimal 60 (or hexadecimal 3C). In the return packet, this bytes are changed depending on the number of bytes being sent. This total length never includes the ethernet header.

[19] CA 00 These 2 bytes represent Identification value sent by a host for reassembling fragmented data. Generally a random number or a linearly increasing number is sufficient for this field when no fragmentation is required.[6] A value of 00 00 in the return packet is also acceptable for simple network environment.

[21] 00 00 These 2 bytes represent Flag and Fragmentation. If no fragmentation is used, which is the normal case for embedded system, a 40 00 or 00 00 is generally sent in the return packet.[3]

[23] 20 This byte is called 'time to live', the number of nodes (routers) the packet is supposed to pass through in the media until discarded.[3] Usual values of 20 to 80 are used.

[24] 01 A very important byte. This byte tells the receiver the type of protocol used for encapsulated data within the frame. The receiver must act depending on the value of this byte. The return packet formed by the embedded system depends on the value of this byte. This also tells the receiver that the header and data that follows the IP header is either ICMP or TCP or UDP or any other. The values of a few very common protocols used in ethernet systems are:

protocols	values
ICMP:	01
TCP:	06
UDP:	11 (dec 17)

This byte is returned as 01 in the return frame.

[25] BA 47 Two bytes of checksum. This checksum is computed on the IP header bytes only. The algorithm for any checksum computation is described later.

[27] 0A 0A 01 02 In this example, the 4-byte IP address of the host is 10.10.1.2. Embedded software should copy these 4 bytes in the return frame as destination IP address.

[31] 0A 0A 01 64 In this example, the 4-byte IP address of destination is 10.10.1.100 . These bytes are already stored in the embedded software and are copied into the return packet as source IP address.

The next 8 bytes are the ICMP (Ping)header.

[35] 08 00 Although these 2 bytes separately represent two different parameters, they are generally used together in source and destination frames. Here are a few meanings of these bytes[6]:

type	code	representation
08	00	Echo Request
00	00	Echo Reply
00	03	Destination Unreachable

The host system will use 08 00 requesting an echo, and the responding system must use 00 00.

[37] 47 5C Checksum bytes. The checksum is computed for all bytes starting from the first byte of the ICMP header to the last byte of echo data.

[39] 01 00 Identification number. Used by a host for request-response matching.

[41] 05 00 Sequence number. This increasing number is also used for matching. Identification number and sequence number are returned in the response packet without change for correct matching.[6]

The next 32 bytes are ICMP data to be returned exactly as received in order to verify the reliability of ethernet connection. Some operating systems use different number of bytes than 21; it is thus necessary to capture all data from the host computer and return them.

With this explanations of bytes and groups of bytes received from a host system, the frame sent by the embedded system in response to a ICMP packet will take the following form. This frame is formed after checking byte number 24, which in this case is 01, indicating that an ICMP (ping) frame is requested.

[1] 00 40 33 2E 99 4F [7] 00 C0 F0 34 AC 7B
[13] 08 00

[15] 45 [16] 00 [17] 00 3C [19] 00 00 [21] 40 00 [23] 80 [24] 01
[25] E4 47 [27] 0A 0A 01 64 [31] 0A A0 01 02

[35] 00 00 [37] 4F 5C [39] 01 00 [41] 05 00

[43]
61 62 63 64 65 66 67 68 69 6A
6B 6C 6D 6E 6F 70 71 72 73 74
75 76 77 61 62 63 64 65 66 67
68 69

The differences between a received packet and a transmitted packet are summarized as:

- Source ethernet address and destination ethernet address have interchanged their byte positions.
- Source IP address and destination IP address have interchanged their byte positions.
- Byte position 35 is changed from 08 to 00, for the receiver to know this is an echo reply.
- Checksums for return IP header is computed and the values are inserted at positions 25 and 26.
- Checksums for return ICMP header is computed and the bytes are inserted at locations 37 and 38.

TRANSMISSION CONTROL PROTOCOL-INTERNET PROTOCOL (TCP-IP)

The Transmission Control Protocol (TCP) is intended for use as a highly reliable protocol between hosts in computer communication networks. The TCP is able to transfer a continuous stream of data (bytes) in each direction between two participating systems by packaging the data into segments when necessary. The TCP fits into a layered protocol architecture and interfaces user data with a lower level protocol such as Internet Protocol (IP).

TCP-IP has the most dominant role in Internet communications. For this reason, an example application of TCP-IP protocols in internet environment is appropriate for understanding the inner workings of this protocols. A simple internet application of an embedded system is a web server that serves only HTML pages, also known as HTTP server. The development of an application software for an embedded system which functions as a HTML page server is described next in detail. This is an implementation of the most essential part of TCP-IP protocol which enables an embedded system to successfully respond to a request for a HTML page. The HTML page may contain text, a jpg or gif picture, and/or a java applet embedded in the page.

Communications between a host computer requesting a HTML file and the responses from a server (the embedded system) takes place in four steps.[2] These steps are illustrated in Figure 2.

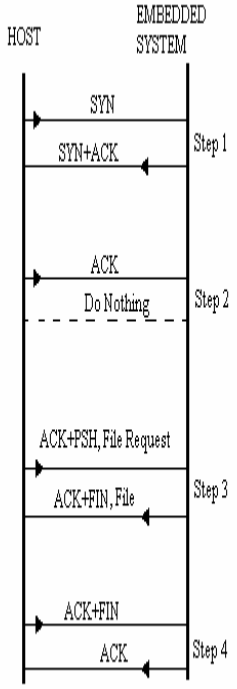


Figure 2: Transmission of a single TCP-IP packet requires 4 steps of information exchange between a host and a server.

In Step 1 the host sends a Synchronization frame (with SYN flag); the embedded system

replies with a SYN and Acknowledgement (ACK) frame. In Step 2 the host sends an ACK frame but the embedded system sends nothing in response. The host, in Step 3, sends a frame with ACK and PSH flag , and requests a HTML file. The embedded system, in response, sends the entire HTML file in one frame, without segmentation, and also includes ACK and Finish (FIN) flags. Connection is terminated in Step 4 when the host sends a frame with a ACK and FIN flag, and the embedded system responds with the ACK frame. Each frame, whether sent by a host or a response by an embedded system contains an ethernet header, an IP header, a TCP header and, when needed, TCP data, which are generally user data. The complete packet structure is shown below.

Ethernet Header

Destination Ethernet address	Source Ethernet address	Protocol Type
6 Bytes	*6 Bytes*	*2 Bytes*

IP Header

Version +Header Length	Service	Length of IP+TCP	Ident value	Flag, Fragment Offset
1 Byte	*1 Byte*	*2 Bytes*	*2 Bytes*	*2 Bytes*

Time to live	Protocol	Checksum of IP Header	Source IP Address	Destination IP Address
1 Byte	*1 Byte*	*2 Bytes*	*4 bytes*	*4 bytes*

TCP Header

Source Port	Destination Port	Sequence Number	Acknowledgement Number
2 Bytes	*2 Bytes*	*4 bytes*	*4 bytes*

TCP Header Length	Flags	Window	Checksum	Urgent Pointer
1 Byte	*1 Byte*	*2 Bytes*	*2 Bytes*	*2 Bytes*

User Data When Necessary

Step 1

A typical frame with a SYN flag received from a host, in Step 1, will have the following values.

```
00 C0 F0 34 AC 7B 00 40 33 2E 99 4F 08 00 45 00
00 30 5B 04 40 00 80 06 89 4A 0A 0A 01 02 0A 0A
01 64 04 0B 00 50 02 39 B6 A2 00 00 00 00 70 02
20 00 8F 6F 00 00 02 04 05 B4 01 01 04 02
```

The bytes in the frame are now grouped according to their fields.

Ethernet Header

[1]00 C0 F0 34 AC 7B [7]00 40 33 2E 99 4F [13]08 00

IP Header

[15]45 [16]00 [17]00 30 [19]5B 04 [21]40 00 [23]80 [24]06 [25]89 4A [27]0A 0A 01 02 [31]0A 0A 01 64

TCP Header

[35]04 0B [37]00 50 [39]02 39 B6 A2 [43]00 00 00 00 [47]70 [48]02 [49]20 00 [51]8F 6F [53]00 00

[55]02 04 05 B4 01 01 04 02

Ethernet and IP Header bytes are explained before in connection with ICMP and ARP frames. Explanation of the TCP header bytes follows:

[35]04 0B Port number of the source. The same port number is sent back to the host as destination port number in return packet.

[37] 00 50 Port number of destination. There are many port numbers assigned for different applications, a few of them are shown in the table below. For more port numbers see the reference.[2,8]

Operation	Port Number (Decimal)
Echo	07 (7)
ftp	15 (21)
smtp	19 (25)
finger	4F (79)
http	50 (80)

The embedded system is the http server. Thus 00 50 will be sent as source port number in the return packet.

[39] 02 39 B6 A2 Sequence number of the host. Sequence number and Acknowledgement numbers are designed to keep track of number of bytes sent and received by the host and the embedded server. In the return frame the sequence number of the host becomes the acknowledgement number of the server, and acknowledgement numbers of the host becomes the sequence number of the server. The sequence number is incremented by 1 whenever a SYN, FIN or URG frame is received.[3] For this reason, the embedded server adds 1 to this number and returns 02 39 B6 A3 as the acknowledgement number.

[43] 00 00 00 00 Acknowledgement number of the host. In the first frame, the host sends all zeros as the acknowledgement number. The server can start with any number; but a convenient number is 00 00 FF FF. Because of a SYN frame, the host will add 1 to this number and will send back 00 01 00 00 as the acknowledgement number[2] in the next frame.

[47] 70 TCP Header length. Actual length is determined by shifting the byte two positions to the right. For this example, 1C is obtained when 70 is shifted 2 bit positions to the right. 1C (dec 28) is the TCP header length including 8 optional bytes. Another interpretation of the number is there are 7 32-bit words.

[48] 02 Flag. This byte is very important in each packet (reception or transmission), because the response depends on the value of this byte and the embedded server can also keep track of Steps with this byte. The following table shows names of a few flags and their hexadecimal values.

Flag	Value	Function
SYN	02	Synchronization
ACK	10	Acknowledgement
FIN	01	Finish
RST	04	Reset
PSH	08	Push

In the return packet a combination of SYN and ACK flags, 12, is sent as 1 byte.

[49] 20 00 Window size. This field is important in the return frame, because the server can notify the host the size of its receiving buffer size in bytes. The buffer size does not have to depend on the available RAM of the embedded system, because most ethernet chips contain built-in buffers for reception and transmission of frames. For example, the ethernet controller chip CS8900 has a buffer of size 1516 bytes. In all return frames this field will be replaced with hex number 05AA (dec. 1450).

[51] 8F 6F Checksum. Checksum is computed for all bytes starting from the first byte of TCP header to the last byte of TCP data, if any, and a 12-byte long Pseudo IP header. The Pseudo header will have the following components.[2]

Server IP Address	Host IP Address	Protocol (TCP)	TCP Header+Data Length
0A 0A 01 64	0A 0A 01 02	00 06	00 14 (if no data)

[53] 00 00 Urgent pointer. This field in the return frame is left unchanged.

[55] 02 04 05 B4 01 01 04 02 Optional bytes of TCP header [7]. Generally contains information about maximum segment size. The server is not required to send these bytes back.

With the above explanations the return packet in Step 1 will take the following form.

[1]00 40 33 2E 99 4F [7]00 C0 F0 34 AC 7B [13]08 00

[15]45 [16]00 [17]00 28 [19]00 01 [21]40 00 [23]80 [24]06 [25]E4 55 [27]0A 0A 01 64 [31]0A 0A 01 02

[35]00 50 [37]04 0B [39]00 00 FF FF [43]02 39 B6 A3 [47]50 [48]12 [49]05 AA [51]D6 77 [53]00 00

The bytes in fields that underwent major changes in the return packet are summarized below.

- No fragmentation is used (byte position 21). For fragmentation, see [3][4].
- Starting server sequence number is 00 00 FF FF (position 39-42).
- 1 is added to the received sequence number and returned as acknowledgement number (pos 43-46).
- TCP header length is 50 (five 32-bit word or decimal 20 bytes; byte position 47).
- Flag byte is 12 (SYN+ACK; byte position 48).
- Embedded system's buffer size is 05 AA (position 49-50).

Step 2

The message 'web site found waiting for reply' frequently appears on the status bar of a web browser. This message is displayed when Step 1 is completed. The TCP-IP packet sent by the host at this stage is grouped into its various fields for quick identification and displayed below.

Ethernet Header

[1]00 C0 F0 34 AC 7B [7]00 40 33 2E 99 4F [13]08 00

IP Header

[15]45 [16]00 [17]00 28 [19]5D 04 [21]40 00 [23]80 [24]06 [25]87 52 [27]0A 0A 01 02 [31]0A 0A 01 64

TCP Header

[35]04 0B [37]00 50 [39]02 39 B6 A3 [43]00 01 00 00 [47]50 [48]10 [49]21 80 [51]BA A2 [53]00 00

The data in each byte position is explained in previous sections but a few byte positions are elaborated.

Positions 17-18: Total length is 0028 (or decimal 40). Length of IP and TCP header is 20 bytes each.

Position 24: Protocol type is 06 (TCP).

Positions 39-42: Sequence number. This is the number sent by the embedded system in Step 1.

Positions 43-46: Acknowledgement number. The host added 1 to 0000FFFF sent by the server in step 1.

Position 47: TCP length is 50. Which is 5 32-bit word or decimal 20 bytes.

Position 48: Flag is 10, an Acknowledgement flag.

The embedded system recognize this packet from the flag byte and sends no response but wait for the next packet from the host.

Step 3

This is the most important and complicated Step for a server because in this step a server not only receives a larger packet, it must also identify a request for a file from the host and construct and transmit a larger packet with a file. Because of the larger size of this packet, only the header portions are displayed in Hexadecimal and the remaining portion is displayed in ASCII codes for the reason of readability.

Ethernet Header

[1]00 C0 F0 34 AC 7B [7]00 40 33 2E 99 4F [13]08 00

IP Header

[15]45 [16]00 [17]01 35 [19]5E 04 [21]40 00 [23]80 [24]06 [25]85 45 [27]0A 0A 01 02 [31]0A 0A 01 64

TCP Header

[35]04 0B [37]00 50 [39]02 39 B6 A3 [43]00 01 00 00 [47]50 [48]18 [49]21 80 [51]24 4E [53]00 00

[55] **TCP Data in ASCII**

GET / HTTP/1.1(CR+LF)
Accept: application/vnd.ms-excel, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*(CR+LF)
Accept-Language: en-us(CR+LF)
Accept-Encoding: gzip, deflate(CR+LF)
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)(CR+LF)
Host: 10.10.1.100(CR+LF)
Connection: keep-Alive(CR+LF)
(CR+LF)

The ASCII codes of the above lines of text, in Hexadecimal, are received right after the last byte of TCP header. The bytes starting from position 55 will be termed as data bytes. (CR+LF), after each line above, represents two bytes of ASCII codes for Carriage Return and Line Feed, which are 0D 0A. The number of characters in the box, including 'spaces' and 'commas', are exactly 269. Byte positions 17 and 18 of the received packet, which are used for packet length including IP header and TCP header but not Ethernet header, contain 0135 (decimal 309). When we subtract 40, the header length for IP and TCP, from 309, we obtain 269 (or Hex 10D), which is the exact number of data bytes in the packet. For this reason the numbers in positions 17 and 18 are important in forming the acknowledgement number in the transmitted packet. Byte position 48 shows a combination of ACK and PSH flag. PSH flag tells the receiver to act on the packet immediately.

At this point the embedded server must read a few bytes starting from position 55, which are 'GET / HTTP….'. For example, if the host is requesting a file named BLUE.HTML, the received codes will be 'GET /BLUE.HTML'. But for a homepage, such as INDEX.HTML or DEFAULT.HTML, only a single blank space may appear after '/', which is 'GET / '.

Once the embedded server is able to determine which file to transmit, the formation of the return packet begins. The following two lines of text of 44 characters, or their variations, must precede every html file.

HTTP/1.0 200 OK(CR+LF)
Content-type: text/html(CR+LF)(CR+LF)

In this example a simple html file is composed for demonstration. With this short html file the return packet will take the following form.

Ethernet Header

1	00 40 33 2E 99 4F	7	00 C0 F0 34 AC 7B	13	08 00

IP Header

15	45	16	00	17	00 DC	19	00 02	21	40 00	23	80	24	06	25	E3
A0	27	0A 0A 01 64	31	0A 0A 01 02											

TCP Header

35	00 50	37	04 0B	39	00 01 00 00	43	02 39 B7 B0
47	50	48	11	49	05 AA	51	82 94	53	00 00

55

```
HTTP/1.0 200 OK(CR+LF)
Content-type: text/html(CR+LF)(CR+LF)
<html>
<head><title>Ebtron Inc.</title></head>
<body>
<center><h1>EBTRON
INC.<br><br>SOUTHEASTERN COMM
COLLEGE</h1></center>
</body>
</html>(space)
```

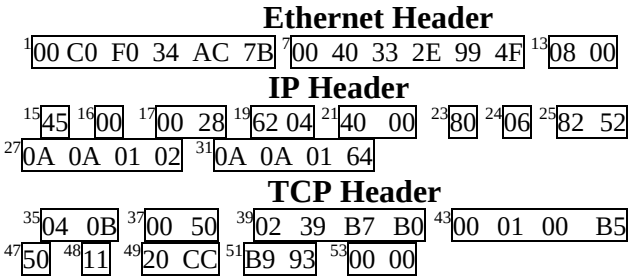
This simple default html file starts with the tag <html> and finishes with the tag </html>. There is no (CR+LF) at the end of a line in the html file. In fact, the whole file can be written in one line. One 'space' character is added at the end of the file to make the number of characters even. There are exactly 180 data bytes in the return frame. These 180 data bytes and 40 header bytes make a total of 220 bytes (or Hex 00DC). This is reflected in the byte positions 17 and 18 of the return packet. The other important change in this return packet is the acknowledgement number. As previously noted, the total number of data bytes in the received frame in Step 3 is 010D (or Decimal 269). When 010D is added to the received sequence number 0239B6A3, one obtains 0239B7B0, which is sent as the acknowledgement number as shown in byte positions 43-46. The computation of sequence number and acknowledgement number for a return packet can be summarized as

- Subtract Hex 28 (decimal 40, assuming no optional bytes are present in TCP header) from the received number in positions 17-18. Add the result to the sequence number received in this Step. Insert the last result in positions 43-46 for acknowledgement number.

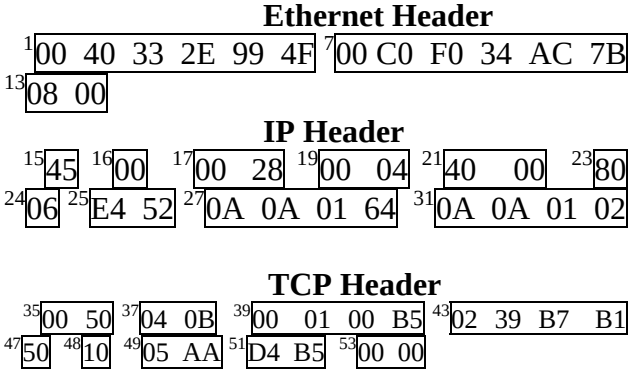
- Insert the received acknowledgement number as sequence number without change.
- Add decimal 40 (number of bytes in IP and TCP header) to the number of data bytes to be transmitted. Insert the Hex result in positions 17-18.

Step 4

In the last Step host sends a ACK+FIN frame with an acknowledgement number indicating the number of data bytes it has received and signaling an end of session. A typical frame in this Step takes the form of



In response, the embedded server transmits an ACK frame and updates the acknowledgement number.



This completes transfer of a simple html file from an embedded server to a host. If the html file refers to other files, such as image files, communication continues from Step 3 where the host requests the image file with another 'GET…' command. Other variations of the Content-type line are[2]

Content-type: image/gif(CR+LF)(CR+LF) [for GIF images]

Content-type: text/plain(CR+LF)(CR+LF) [for UDP text or java class]

Efficient software will at first check byte position 22 (or 21-22) in the received packet for an ARP frame. If it is not an ARP frame, the software should then check byte position 24 for a ICMP, TCP, or UDP frame and respond accordingly. The web server software should also check byte position 48, the flag byte, to keep track of the 4 Steps and construct its responses accordingly. Some protocols also require checking port numbers.

CHECKSUM COMPUTATIONS FOR ALL PROTOCOLS

One of the most important parts of all computer communication is checksum computations over the bytes received or to be transmitted. Checksum computation is also sometimes the most inconvenient for many embedded systems because of their RAM limitations and the computation time required before a quick response. But some pre-planning can overcome both these obstacles.

All protocols described in this paper follow identical procedures for computations of checksum.[2] The procedure is summarized below, where the words 'checksum data' is used to refer to all data over which checksum is being computed.

- Take the first 2 bytes (16 bit word) from the checksum data
- Add next 2 bytes of the checksum data. This way continue adding next 2 bytes from the data until an overflow (carry) occurs. Then discard the carry but add 1 to the least significant bit of the result of addition.
- When all 2-byte words are added, take 1's compliment of the result. This is the final checksum.

A few important aspects of this checksum computation must be noticed. (a) The total number of bytes must be an even number. (b) 2 bytes of zeros, 00 00, will not affect the checksum. For this reason, the byte positions where checksum is inserted can be filled with 00 00 before checksum computation. (c) It does not

matter which 2 bytes are added first or last. This allows us to pre-compute checksum of the fixed portion of the packet (without 1's compliment), such as, a HTML file, a GIF file, etc. and store the result. When formation of the dynamic portion of the packet, such as protocol headers (where the bytes change frequently) are complete, the header data, 2 bytes at a time, are added to the pre-computed checksum and eventually 1's compliment to the final summation is performed. With this method, computation of checksum of a large packet containing a large file will appear to take the same amount of time as needed for a 40-byte header.

CONCLUSIONS

There are several freeware and shareware software available on internet which are able to capture the hexadecimal frames, displayed in this paper, traveling through ethernet media. Advanced features of the protocols, presented in this paper, are not supported by this simple algorithm for an embedded system. Server software must be able to recover from any error in communication, such as when the Steps get out of sync. Static data, such as HTML files, image files, etc. can be stored and transmitted directly from flash or EPROM memory. Small amount of variable user data can easily be incorporated in a HTML file with some preplanning of the file. This basic algorithm, with some minor modification, can also be extended to respond to UDP packets and even DHCP packets and other TCP or IP based protocols, such as MODBUS TCP, BACnet IP, etc. The hardware and software techniques presented in the paper will respond equally well when the host computer runs under MS Windows operating system or Linux operating system.

REFERENCES

1. Tariqul Haque, Michael Urbaniak, Step by Step Implementation of Ethernet and TCP-IP Protocols on an Embedded System, Computer in Education Journal,Volume XIV, Number 4, 2004

2. Jeremy Bentham, TCP/IP Lean, CMP Books, 2000.

3. W. Richard Stevens, TCP/IP Illustrated, Volume I, Addison-Wesley, 1994.

4. Ata Elahi, Network Communications Technology, Delmar Thomson, 2001.

5. RFC 791 www.rfc-editor.org

6. RFC 792 www.rfc-editor.org

7. RFC 793 www.rfc-editor.org

8. www.etherimage.com/ethernet

BIOGRAPHICAL INFORMATION

Tariqul Haque received his Ph.D. in Electrical Engineering in 1986 from Clarkson University, Potsdam, NY. Since then he has served as a faculty member at Suffolk University, Boston, MA, and image processing specialist with Kontron Electronik, GmbH. Currently he is a faculty member at Southeastern Community College, Whiteville, NC and a consultant with Ebtron, Inc. His interests include microprocessors, communications, image and speech processing and Windows programming.

Michael J. Urbaniak graduated in 1984 from Somerset County Technology Institute, Bridgewater, NJ. For the last 20 years Mr. Urbaniak has been involved in the design of microcontroller based instrumentation hardware and software. Currently he is senior vice president of Ebtron Inc. Loris, SC, and is involved in developing distributed network sensors and controllers.