

CZECH TECHNICAL UNIVERSITY IN PRAGE

FACULTY OF ELECTRICAL ENGINEERING



BACHELOR THESIS

Use of TCP/IP for embedded application

Author:

Phuc Trinh Gia

Supervisor:

Ing. Michal Brejcha, Ph.D

Reviewer:

Ing. Jiří Hájek, Ph.D

PRAGUE

2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Electrotechnology

BACHELOR PROJECT ASSIGNMENT

Student: **Trinh Gia Phuc**

Study programme: Electrical Engineering, Power Engineering and Management

Specialisation: Applied Electrical Engineering

Title of Bachelor Project: **Use TCP/IP for embeded application**

Guidelines:

1. Study the use of TCP/IP stack in embeded aplication.
2. Implement Web Server using lwIP Stack on a evaluation MCU board.
3. Create a Web Site on your server. The Web Site is supposed to allow to display states of chosen inputs and control expansion board.
4. The proper function of Web Server is supposed to be tested via PC.

Bibliography/Sources:

- [1] DUNKELS, Adam. Design and Implementation of the lw IP TCP/IP Stack [online]. 2001 [cit. 2016-02-18]. Dostupné z: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.1795>
- [2] AN4739 Application note: STM32Cube firmware examples for STM32F4 Series [online]. STMicroelectronics, 2015 [cit. 2016-02-18]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/DM00213525.pdf
- [3] STM32F4DISCOVERY: Discovery kit with STM32F407VG MCU. STMicroelectronics [online]. 2016 [cit. 2016-02-18]. Dostupné z: <http://www.st.com/web/en/catalog/tools/FM116/CL1620/SC959/SS1532/LN1848/PF252419>

Bachelor Project Supervisor: Ing. Michal Brejcha, Ph.D.

Valid until the end of the summer semester of academic year 2016/2017

L.S.

Ing. Karel Dušek, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 8, 2016

Abstract

This thesis describes the use of TCP/IP protocol for embedded application. It was done during implementation of web server using LwIP (Light weight TCP/IP stack) on evaluation microcontroller board. The working of web server is tested via PC. Connecting to the server is based on TCP sockets. The web server is supposed to performs some applications such as display states of chosen inputs and control expansion board. Besides LwIP the server is developed by using of FreeRTOS (Free Real Time Operating System) for embedded devices.

Acknowledgement

I wish to express my sincere thanks to my supervisor Ing. Michal Brejcha PhD, for providing me with all the necessary facilities for the research as well as for his sharing expertise, sincere and valuable guidance and encouragement extended to me.

I am also grateful to Ing. Jiří Hájek Ph.D, for his review, comments and evaluation for my thesis which gives plenty of experiences for my works.

I place on record, my sincere thank you to Ing. Karel Dušek Ph.D, Head of Department of Electrotechnology for giving me the opportunity to be participated in this work.

I am thankful to Ing. Karel Künzel CSc, for sharing his experiences and his advices as well as providing me necessary facilities for my work.

I take this opportunity to express gratitude to all members of the Electrical engineering faculty for their teaching and support.

I also thank my parents for the encouragement, support and attention which they intended for me through this venture.

Declaration

"I hereby declare

- that I have written this writing thesis without any help from others and without the use of documents and aids other than those stated in the Bibliography

- that I have mentioned all the sources used and that I have cited them correctly according to established academic citation rules."

Phuc Trinh Gia

.....

Prague, 27 May 2016

Contents

Introduction.....	1
1.1 Structure of thesis.....	1
1.2 Hardware and materials overview.....	2
1.3 Software and material overview.....	4
TCP/IP protocol.....	6
2.1 Introduction to TCP/IP protocol.....	6
2.2 Model and data flow of TCP/IP protocol	7
2.3 Application layer	10
2.4 Transport layer.....	12
2.5 Internet layer (Network layer)	16
2.6 Physical layer.....	21
LWIP Stack.....	23
3.1 Introduction to LWIP stack	23
3.2 Socket Programming.....	25
3.3 Impelentatin of Application layer	28
3.4 Implementation of Transport layer.....	30
3.5 Implementation of Network layer	33
3.6 Implementation of Physical layer	34
FreeRTOS	38
4.1 Introduction to FreeRTOS	38
4.2 Using FreeRTOS for developing application on embedded server	40
4.3 Demo Webpages and demonstration of server.....	44
Conclusion	47
Bibliography	48

List of Figures

Figure 1. Hardware overview.....	2
Figure 2. Development boards	3
Figure 3. Application board (PLC01A1)	3
Figure 4 Software overview	4
Figure 5. Differences between TCP/IP and OSI models (fiberbit.com.tw)	8
Figure 6. Data flow of TCP/IP stack (blog.buildingautomationmonthly.com).....	9
Figure 7. Working of application layer (hardwaresecrets.com)	11
Figure 8. Working of Transport layer (hardwaresecrets.com)	14
Figure 9. Header frame of Transport layer (ptgmedia.pearsoncmg.com)	14
Figure 10. Working of Internet (Network) layer (hardwaresecrets.com)	17
Figure 11. Data fragmentation by routers	18
Figure 12. Header frame of Network layer (ptgmedia.pearsoncmg.com)	19
Figure 13. IP address ranges	20
Figure 14. Working of Physical layer (hardwaresecrets.com)	22
Figure 15. APIs of LwIP.....	24
Figure 16. Sockets API in the model of TCP/IP (doc.oracle.com)	25
Figure 17. Client sends requests.....	26
Figure 18. Server accepts and establishes the connection.....	26
Figure 19. Working cycle of TCP/IP socket	26
Figure 20. Structure of LwIP buffer for handling data	28
Figure 21. Data flow in Physical layer	35
Figure 22. Communication between DMA and static memory	35
Figure 23. Task states cycle in freeRTOS (freertos.org)	39
Figure 24. Main working cycle of the embedded server	40
Figure 25. Working principle of Application tasks	41
Figure 26. Control application tasks in main cycle	42
Figure 27. Home page of the embedded server	44
Figure 28. Demonstration of Controlling of PLC board	45
Figure 29. Real time monitoring the state of tasks inside Microcontroller.....	46

Chapter 1

Introduction

1.1 Structure of thesis

This thesis describes the using of TCP/IP protocol on an embedded web server, which is based on a microcontroller with embedded web pages in its flash memory. The embedded server is able to be tested with any computer-based device via Ethernet connection. The embedded web server demonstrates the functions of TCP/IP protocol as well as its applications (Real time control and Real time monitoring). The embedded server comprises of two parts, those are hardware and software. For hardware part, it is built from a 32-bit microcontroller, on the other hand the software part is created by using LwIP stack and freeRTOS (Free real time Operating system). Since this thesis explains the developing processes of the embedded web server and its application, it is started from the bottom layer to the top layer of real system. More exactly, it is started from the hardware part to the software part and it is not only included all materials which are used to build the system but it also contains explanations of particular part. The concise structure of the thesis is described by following chapters:

Chapter 1 – Introduction. This chapter is an introduction of whole server with its functions and brief review of its construction and it also includes all materials which are used to create the server.

Chapter 2 – TCP/IP. This chapter is my studying about TCP/IP protocol from its model and structure to its individual layers with detailed explanations, not only their roles of each layer in the system but also how these layers can be applied on a network.

Chapter 3 – LwIP stack. Regardless the TCP/IP in general, this chapter explains in detail the implementations of layers of TCP/IP on embedded server using the LwIP. It also deals with problems during implementation process such as point-to-point model(or client-server model), Socket programming and others.

Chapter 4 – FreeRTOS. This chapter is about FreeRTOS (free Real Time Operating System). The chapter covers the aspect of implementation of FreeRTOS on embedded server together with LwIP stack and application for controlling the extension board.

Chapter 5 – Conclusion. Final chapter contains the summary of whole process in order to develop the embedded system.

1.2 Hardware and materials overview

The embedded server is based on a 32-bit microcontroller and it is supposed to be tested with a computer-based device (PC/Laptop or others) via Ethernet connection. For testing the functionalities of the embedded server, the expansion board (see below) is used and connected with the embedded server through SPI (Serial Peripheral Interface) connection. The embedded server communicates with both devices, transferring the commands from computer-based device (commands from user) to the application boards. The expansion board (application board) is a PLC board and it can perform some basic functions of a PLC system with 24V inputs and outputs. The list of elements and their descriptions are described below. However, it does not contain the detail connections (interconnections) among devices. Figure 1 below shows the construction of whole system from hardware point of view.

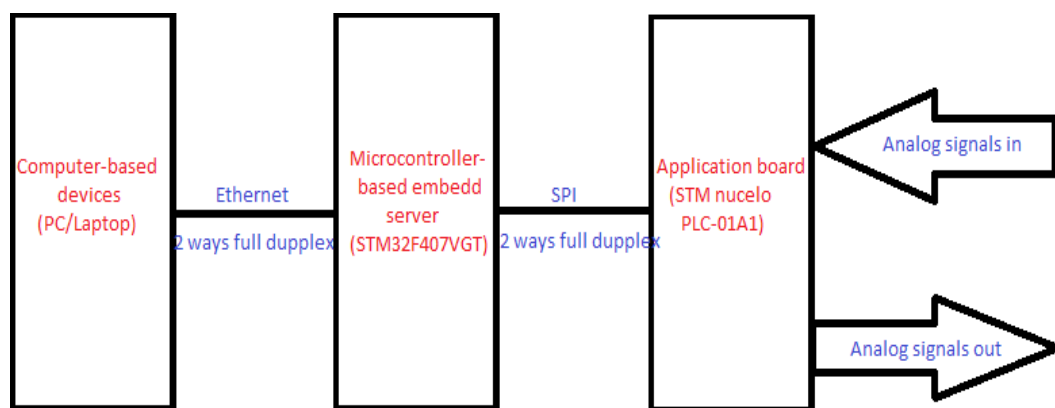


Figure 1. Hardware overview

List of elements used to construct the hardware part:

- Development board DevKit407 is combination of STM32F4 Discovery with extension board STM32F4DISCO-BB based on microcontroller STM32F407VGT6 for implementing of embedded server.
- X-Nucleo PLC-01A1 board for demonstrating applications of embedded server.
- Pc/Laptop with Ethernet connector, Ethernet Cable (RJ-45), small electrical wire with jumpers for SPI connection.

Descriptions:

- DevKit407 development board provides the possibility to interface the Ethernet connection with the microcontroller STM32F407VGT6 through an Ethernet connector. Furthermore, the STM32F407VGT6 is 32-bit microcontroller with high speed working clock gives a good performance of the embedded serve as well as it has enough flash memory for storing the embedded webpages.

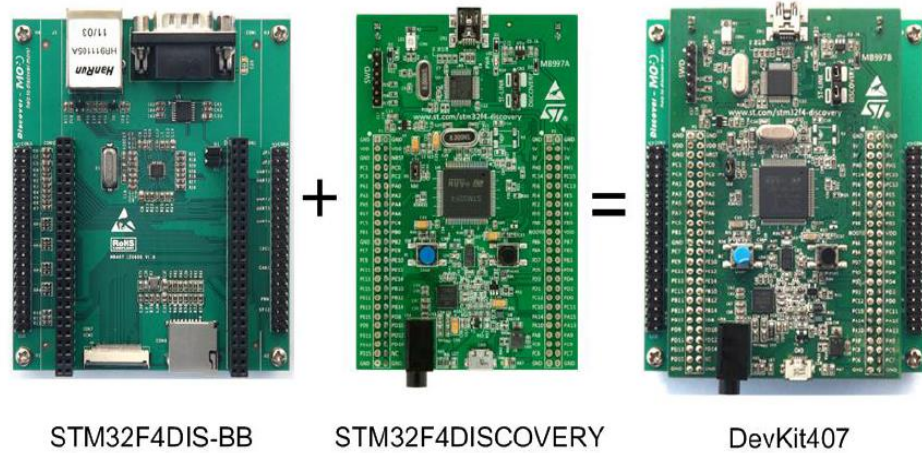


Figure 2. Development boards

- X-Nucleo PLC-01A1 has two integrated circuits which are CLT01-38SQ7 (8x inputs, high-speed protected digital termination array) and VNI8200XP (8x output, high-side solid state relay). The result of combination of two ICs is a compact industrial PLC capable of managing eight analog inputs and eight outputs through the SPI peripheral. The 24 V power supply makes it possible to manage industrial range inputs (i.e. sensors, valves) and outputs (i.e. lamps, alarms).

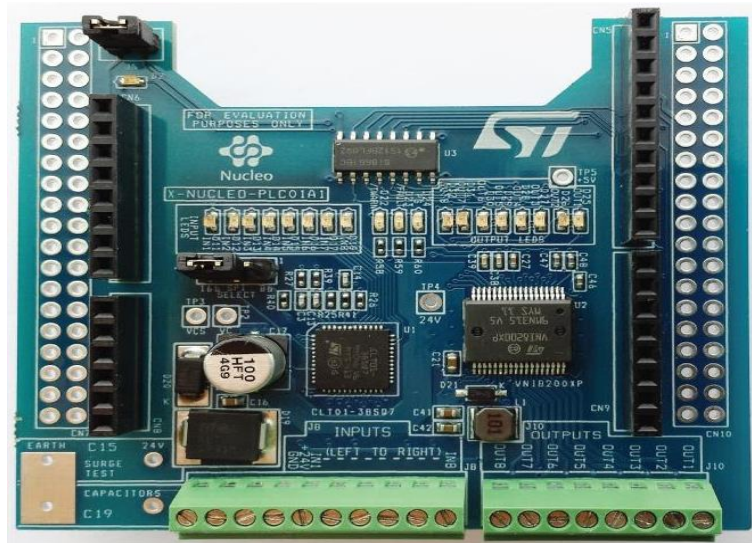


Figure 3. Application board (PLC01A1)

1.3 Software and material overview

Based on hardware part and followed by its idea, this section gives the overview of how the system works from software point of view. The computer-based devices provide a user interface environment such as web browser for users. The browser generates HTTP commands to embedded server by using TCP/IP protocol. Once, the server received the command, it prepares data to send back to computer (the data that the user needs to get), if the data is an application data, the server will process the data for application board and sends them to application board. Because the extension board has inputs then whenever it read the input data and transfers the data back to server, the user can observe the data (Real-time monitoring feature). The construction of software part can be seen as below.

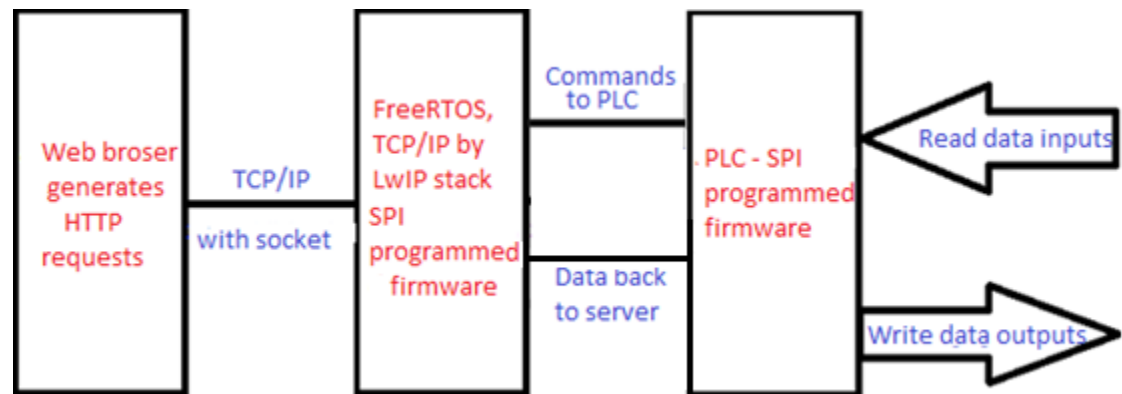


Figure 4 Software overview

List of elements used to construct the software part:

- LwIP stack for implementing of TCP/IP protocols on embedded server.
- FreeRTOS library for functioning of embedded system (to manage all tasks of embedded server)
- STM standard library for interfacing software and hardware.
- SPI programmed firmware for developing applications.
- Keil μ version 5 IDE (Integrated Development Environment) for programming and compiling all source code. (All source codes are programmed in C language)

Descriptions:

- LWIP stack is free open source code which contains functionalities of TCP/IP protocol
- FreeRTOS is free open source code which provides many abilities for optimizing of embedded system.
- STM standard library is from ST microelectronic company for developing the ST's microcontrollers.
- SPI programmed firmware for developing the applications between server and expansion board (PLC).

Chapter 2

TCP/IP protocol

2.1 Introduction to TCP/IP protocol

TCP/IP (Transmission Control Protocol/ Internet Protocol) is a set of protocols (Protocol Suit) that enable communication between devices in a network. Protocols represent rules or standards that govern communications. In other way TCP/IP is a language for devices in order to communicate with each other inside a network. Like in real life, people from different countries can understand each other by using a common language such as English. Obviously, there are many different network protocols to use in a network, however TCP/IP is the industry standard. It is supported by many flat-forms as well as operating systems. Nowadays the most common network in the world is internet also works on TCP/IP. The TCP/IP protocol was introduced about 40 years ago and it has been tested and proved since the first time. It is holding some specified characteristics which are suited in industry such as:

Multi-vendor support. TCP/IP is implemented by many hardware and software vendors. It is not limited to any specific vendor.

Cross flat-forms. TCP/IP can be used in many flat-forms, typically in this thesis we can see this characteristic because we can use such a computer-based (running on Windows or Linux or any other operating system) to communicate with a microcontroller-based serve and perform some specified applications.

Encapsulation and decapsulation. TCP/IP provides an ability to encapsulate data from different applications on a host and transfers these data through its layers to different applications which are listening by other clients in networks. When clients receive the data from network, they decapsulate the data to correct application, for that the data was meant for. For example we can run many network applications on a computer and these applications when they send their data to somewhere in the network, these data are always recognized to desired applications on receiving sides. TCP/IP provides means for delivering packets to the correct application by using ports. Ports are identified using TCP or UDP port number.

Logical addressing. Every network adapter has a globally unique and permanent physical address, which is known as MAC address. This MAC address is burnt into the adapter while manufacturing. This feature gives high possibility to manage a small network (ex. LAN) because every device inside that network determines whether a message is addressed to its own physical address or not. Obviously for bigger network such as WAN or even internet, there are millions of transmissions per second and this may cause some conflicts for devices inside. Thus, to avoid this problem, a big network

can be divided into smaller networks using devices such as routers to reduce network traffic. Interestingly, these devices also use TCP/IP protocol for their functions.

Routability. As mentioned above, a router is a network infrastructure device which can read logical addressing information and direct data across the network to its destination. TCP/IP is a routable protocol which means the TCP/IP data packets can be move from one network segment to another.

DHCP (Dynamic Host Configuration Protocol). This is a standardized network protocol used on Internet Protocol (IP) networks for dynamically distributing network configuration parameters, such as IP addresses for interfaces and services. With DHCP, computers request IP addresses and networking parameters automatically from a DHCP server, reducing the need for a network administrator or a user to configure these settings manually.

DNS (Domain name services). In human life, people can recognize each other by many ways, commonly by a name of a person. Similarly, in computer world each device in a network is given an IP address. However it is impossible for humans to remember such a long sequences of numbers therefore TCP/IP is used because TCP/IP can transfer these number into readable form for humans. (For example google.com is a name which is transferred from specified IP address).

Error and Flow control. The TCP/IP protocols has feature that ensure the reliable delivery of data from source host to the destination client. TCP defines many of these error-checking, flow-control, and acknowledgement functions. [7]

2.2 Model and data flow of TCP/IP protocol

Like other network protocols, TCP/IP has its own network model which is related to OSI network model. However, this thesis focuses on TCP/IP thus the details of OSI model are not considered. Inherited from OSI network model, TCP/IP also has a OSI-based network model. TCP/IP was on the path of development when the OSI standard was published and there was interaction between the designers of OSI and TCP/IP standards. The TCP/IP model is not same as OSI model because original OSI has seven layers while TCP/IP has four layers. In this section, the introduction of TCP/IP model is considered thus the details for each layer are in the following sections.

There are main fours layers which characterize the function of TCP/IP protocol:

- Application layer
- Transport layer
- Network layer (or Internet layer)
- Physical layer (or Network Access layer or Network Interface layer)

Application layer is the top most layer of four layer TCP/IP model and it is on the top of the Transport layer. Application layer defines TCP/IP application protocol and how the application program interface with next layer (Transport layer) services to use the network. This layer also contains all the higher-level protocols like DNS (Domain Name System), HTTP (Hypertext Transfer Protocol), Telnet, TFTP and etc.

Transport layer is next layer under the Application layer and above the Internet layer, its role is to permit devices on the source and destination hosts to carry on a conversation. Transport layer defines the level of service and status of the connection used when transporting data. The main protocols included at Transport layer are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

Network layer is the layer in between the Transport layer and Network Access layer. This layer packs data in to data packets known as IP datagrams, which contain source and destination addresses (MAC – logical address or IP address) and information that is used to forward the datagrams between hosts and clients across networks. The Internet layer is also responsible for routing of IP datagrams. At the destination side data packets may appear in a different order than they were sent. It is the job of the higher layers to rearrange them in order to deliver them to proper network applications operating at the top Application layer. The main protocols included in Internet layer are IP (Internet Protocol), ICMP (Internet Control Message Protocol), ARP (Address Resolution Protocol), RARP (Reverse Address Resolution Protocol) and IGMP (Internet Group Management Protocol).

Physical layer is the last layer of the four layer TCP/IP model. Network Access layer defines details of how data is physically sent through the network, including how bits are electrically or optically signaled by hardware devices that interface directly with a network medium, such as optical fiber or twisted pair copper wire and etc. The protocols included in Network Access layer are Ethernet, Token Ring, FDDI, X.25, Frame relay etc.

TCP/IP model	Protocols and services	OSI model
Application	HTTP, FTTP, Telnet, NTP, DHCP, PING	Application
Transport		Presentation
Network		Session
Network Interface	TCP, UDP	Transport
	IP, ARP, ICMP, IGMP	Network
	Ethernet	Data Link
		Physical

Figure 5. Differences between TCP/IP and OSI models (fiberbit.com.tw)

From figure 5, we can see how the TCP/IP layers work, for each layer sticks with specified protocols. Thus the characteristic of TCP/IP is combination from a set of different smaller protocols.

Another important aspect is data flow of TCP/IP. Data flow represents how the data are encapsulated and transferred through network by TCP/IP protocol. The data flow of TCP/IP is shown in figure below.

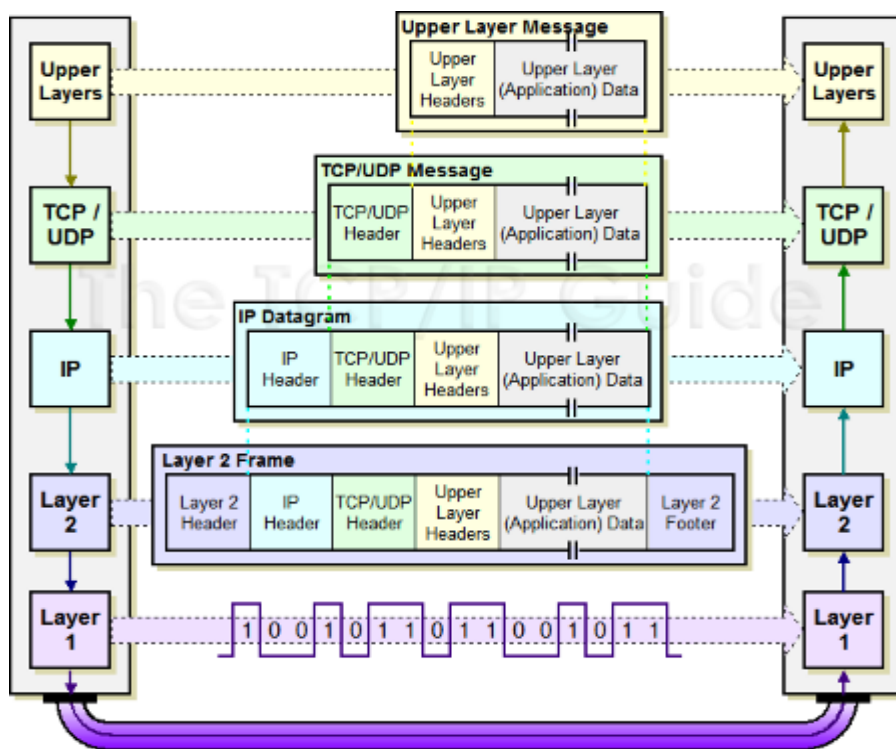


Figure 6. Data flow of TCP/IP stack (blog.buildingautomationmonthly.com)

From figure 6, the device on the left side is a host (or sender) and the device on the right is a client (or receiver).

The host send its data (or request) by using some high-level protocol (HTTP or TFTP, etc) from Application layer to below layer which is Transport layer with a specified header for further process on receiver side.

In Transport layer the data encapsulated with a header which contains the information of sources and destination TCP ports (ex: 80,20,etc), transport layer also handles the segments to Network layer. A track of sent data and received data could be maintained on both sides.

At the next layer which is Network layer the segments (or headers) are being encapsulated into an IP packet adding logical source and destination IP addresses.

Final layer is Physical layer encapsulate the IP packet(s) received from Network layer into a frame with headers contain the physical addresses (MAC). This layer also converts all frames of data which are encapsulated from all above three layers one at a time to stream of bites, encodes the bits into signal based on the type of media used (twisted pair, copper wire or wireless, optical , etc) The information in this layer would help in packet travelling over the network. The physical layer makes sure that the data transfer to or from the physical layer to other is done properly.

The client receive the data which are sent from host with the same series of interaction happens but in reverse order.

The packets of data are first received at the Physical layer, at this layer the information (that was encapsulated before at the Physical layer of the host) is read and rest of the data is passed to the upper layer.

Similarly at the Network layer, the information set by the Network layer protocol of host machine is read and rest of the information is passed on the next upper layer. Same happens at the transport layer and finally the HTTP request sent by the host application (browser) is received by the target application (Website on server).

In general, TCP/IP is a set of protocols with four-layered model. The data flow inside the structure of TCP/IP by adding headers (or segments) through each layer. For the next following chapters, the details of each layer and how it was implemented in my work on embedded server is considered.

2.3 Application layer

Continue from previous section, this section explains in detail the function of application layer and how it can be applied in my embedded server. It also aims to some ideas which are mentioned in the next chapter about LwIP stack. As we already known, this layer makes the communication between high-level application program (user interfaced application program) such as web browser, email ,etc. The most known is HTTP and it is also used as the main method in my embedded web server. The websites which are built from HTML format and compiled into Hexadecimal values, those websites are embedded to flash memory of the microcontroller. When a user asks for a website from the microcontroller, it will request this task to the TCP/IP application layer, being served by HTTP protocol. When www addresses applied on web browser to open a web page, the browser requests this task to the TCP/IP application layer by HTTP method "GET", once the website is opened then user can make send data to server by "POST" method. The two methods "GET" and "POST" send the packets of data to the

lower layer. At this stage the Application layer talks to the Transport layer through a port. Ports are numbered and standard applications always use the same port. For example, HTTP protocol always use port 80 and FTP protocol always use ports 20. Particularly, in this work because I use HTTP protocol therefore only port 80 is considered. The use of a port number allows the Transport protocol (typically TCP) to know which kind of contents is inside the packet (for example, to know that the data being transported is a HTTP request such as “GET” or “POST”) allowing it to know, at the reception side, to which Application protocol it should deliver the received data. So, when receiving a packet target to port 80, TCP protocol will know that it must deliver data to the protocol connected to this port, usually HTTP, which in turn will deliver data to the program that requested it (the web browser). In figure 7 below we illustrate how the application layer works.

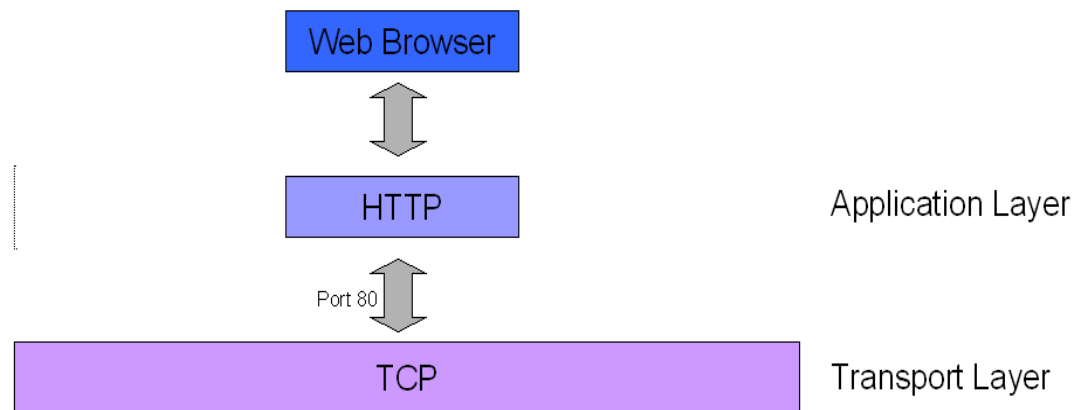


Figure 7. Working of application layer (hardwaresecrets.com)

This layer prepares the data need to be send and sends them to the next layers in order the data will be wrapped by headers for transmission process.[8]

2.4 Transport layer

The transport layer includes two major protocols which are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). These two protocols help in data communication as explained below.

- An interface for network applications to access the network.
- Provides means accepting data from different applications and directing that data to the recipient application on the receiving device (Multiplexing). Same way, on the receiving device the data need to be directed to the correct application, for that data was meant for (Demultiplexing).
- Error checking, flow control and verification. Two major protocols at the Transport layer, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) work differently to achieve these goals.

User Datagram Protocol (UDP) is not reliable protocol. There is no error checking features for UDP and it is designed for situations when extensive control features are not necessary. UDP is called as a connectionless protocol.

In this work, because I used Transmission Control protocol (TCP) and also it is the most common protocol for transferring data therefore the characteristic of TCP is considered intensively.

Transmission Control Protocol (TCP) is a reliable protocol. TCP provides extensive error control and flow control to ensure the successful delivery of data. TCP is called as a connection-oriented protocol. It is responsible for breaking up the message (Data from Application layer) into TCP segments and reassembling them at the receiving side. It is not sure that the data reaching at the receiving side is in the same order as the sending side, because of the problems in network of different paths packets flow to the destination. TCP is also responsible for keeping the unordered segments in the right order. However, TCP assures a reliable delivery by resending anything that gets lost while traveling the network. In general, we can consider following characteristics of TCP: Stream Data transfer: Applications working at the Application layer transfers a nearby stream of bytes to the bottom layers. It is the task for TCP to pack this byte stream into packets, known as TCP segment, which are passed to the IP layer for transmission to the destination. The application layer does not have to bother to divide the device stream data packets. TCP protocol holdings following characteristics which are listed below:

- **Reliability:** The most important feature of TCP is reliable data while delivering them. In order to provide reliability, TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the Network Layer. TCP assigns a sequence to each byte it transmitted, and expects a positive acknowledgement (ACK) from the receiving TCP layer on receiving side. If the ACK is not received within a specified time out interval, the data is retransmitted. This process could be repeated for several times until on sending side received the ACK. On the receiving side TCP uses the sequence numbers to

rearrange the TCP Segments when they arrived out of order, and to eliminate duplicated TCP segments.

- **Flow Control:** Network devices operate at different data rate because of various factors like speed of device or available bandwidth of transmission line. There is a possibility happened when sending device to send data at a much faster rate than the receiver can handle. For example, in this work the embedded server (main role like a receiver) is microcontroller-based device while the client (sender) is computer-based with higher speed of processing as well as transferring the data. In such these cases TCP uses a sliding window mechanism for implementing flow control. The number assigned to each segment is called a sequence number and thus numbering is done at the byte level. The TCP at the receiving side, when sending an ACK back to sender, also indicates to the TCP at the sending side that the number of bytes it can receive (beyond the last received TCP segment) without causing serious problem in its internal buffers.
- **Multitasking:** Multitasking achieved through the use of port numbers.
- **Connections-oriented:** Before application processes can send data by using TCP, the devices on network must establish a connection. The connections are made between the port's numbers of the sender and receiver. A TCP connection identifies the end point involved in the connection. At this stage a special idea is introduced which is a **socket**, socket is a combination of IP address and port number, which can uniquely identify a connection. The details about socket programming are presented in next chapter (LwIP).
- **Full duplex:** TCP provides concurrent data stream in both directions.

Back to the principle of working of Transport layer, Both UDP and TCP will get the data from the Application layer and add a header to it when transmitting data. When receiving data, the header will be removed before sending data to the proper port. On this header there are several control information, in particular the source port number, the target port number, a sequence number (for the acknowledge and reordering systems used on TCP) and a checksum (which is a calculation used to check whether data arrived intact at destination or not). UDP header has 8 bytes while TCP header has 20 or 24 bytes (it is depended if the options field is not used or used respectively).

In figure 8 below we illustrate the data packet generated on the transport layer. This data packet will be sent to the Internet layer (if we are transmitting data) or was sent from Internet layer (in case of receiving data)

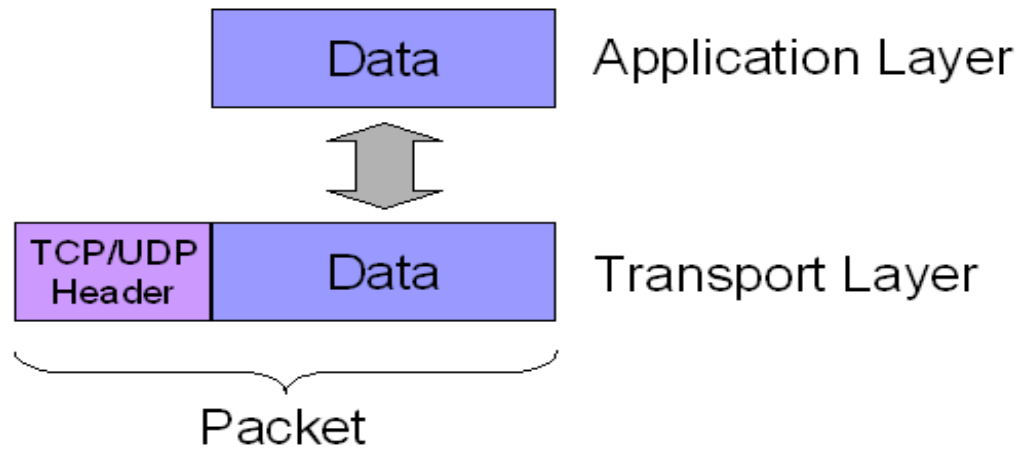


Figure 8. Working of Transport layer (hardwaresecrets.com)

The last aspect should be considered in this section is about a TCP header because TCP header is mainly used in the implementation of the embedded server. In this part, the detail of a TCP header is expressed as figure below and following explanations come across.

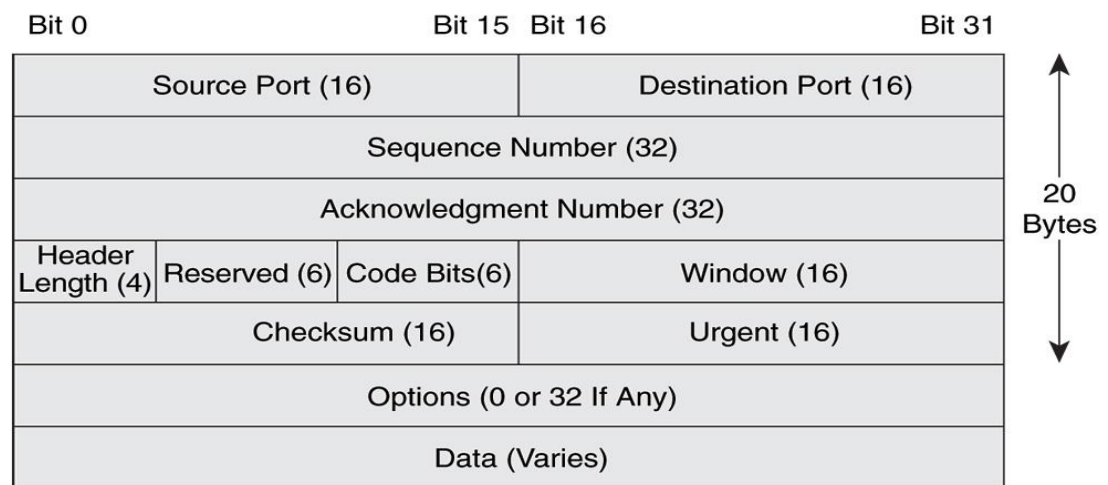


Figure 9. Header frame of Transport layer (ptgmedia.pearsoncmg.com)

- **Source port:** 16 Bit number which identifies the Source Port number (sending computer's TCP port)
- **Destination port:** 16 bit number which identifies the Destination port number (Receiving port)
- **Sequence number:** 32 bit number used for byte level numbering of TCP segments. If TCP is used, each byte of data is assigned a sequence number. If SYN flag is set (during the initial three way handshake connection initiation), then this is the initial sequence number. The sequence number of the actual first data byte will then be this sequence number plus 1. For example, let the

first byte of data by a device in a particular TCP header will have its sequence number in this field 50000. If this packet has 500 bytes of data in it, then next packet sent by this device will have sequence number of $50000+500+1=50501$.

- **Acknowledgment Number:** 32 Bit number field which indicates the next sequence number that the sending device is expecting from sending device is expecting from the other device.
- **Header Length:** 4 Bit field which shows the number of 32 bit words in the header. Also known as the Data Offset field. The minimum size header is 5 words (binary pattern is 0101).
- **Reserved:** Always set to 0 (size 6 bits)
- **Control Bit Flags (Code bits):** As discussed before the TCP is a Connection oriented protocol. The meaning of it is that, before any data can be transmitted, a reliable connection must be obtained and acknowledged. Control Bits govern the entire process of connection establishment, data transmissions and connection termination. The control bits are listed as follows:
 - URG: Urgent Pointer.
 - ACK: Acknowledgement.
 - PSH: Push function, allows a sending application to specify that the data must be pushed immediately.
 - RTS: Reset connection due to unrecoverable error.
 - SYN: Synchronize sequence number.
 - FIN: Finish transmission, no more data from the sender.
- **Window:** Indicates the size of the receive window, which specifies the number of bytes beyond the sequence number in the acknowledgment field that the receiver is currently willing to receive.
- **Checksum:** The 16-bit check sum field is used for error-checking of the header and data.
- **Urgent pointer:** shows the end of the urgent data so that interrupted data streams can continue. When the URG bit is set, the data is given priority over other data streams (16 bits long).

The next section is about lower layer in TCP/IP model which is Internet layer (Network layer). The Internet layer is inherited from this layer by adding necessary header to the chain of data.[9]

2.5 Internet layer (Network layer)

From section 2.4 mentioned that the Internet layer packs data in to data packets known as IP datagrams, which contain source and destination addresses (MAC – logical address or IP address) and information that is used to forward the datagrams between hosts and clients across networks. However, the using MAC address is only good for small LAN (Local Area Network) where devices in network are directly connected, therefore with the bigger network it necessary to add more information in order to identify exactly which device should be communicated. As I mentioned before, the special device is being used for bigger network which is router. Routing is the path that a data packet should use in order to arrive at destination. When requesting data from an Internet server, for example, this data passes through several locations (routers) before arriving at final device.

On every network that is connected to the Internet there is a router, which makes the bridge between the computers on local area network and the Internet. Every router has a table with its known networks and also a configuration called default gateway pointing to another router on the Internet. When a computer sends a data packet to the Internet, the router connected to network first looks if it knows the target computer – in other words, if the target computer is located on the same network or on a network that the router knows the path to. If it doesn't know, it will send the packet to its default gateway, i.e., to another router. Then the process repeats until the data packet arrives at its destination. Above information is again briefly expression how the Internet layer acts in a network (from small to bigger network).

Back again to details working of Internet layer, this layer adds a header to the data packet received from the Transport layer where, among other control data, it will add the source IP address and the target IP address (or combination of IP addresses and MAC addresses)² – i.e., the IP address of the computer that is sending the data and the IP address of the computer that should receive the data. There are several protocols that work on the Internet layer: IP (Internet Protocol), ICMP (Internet Control Message Protocol), ARP (Address Resolution Protocol) and RARP (Reverse Address Resolution Protocol). Data packets are sent by embedded sever in this thesis using the IP protocol, so that is the protocol we will explain.

IP protocol gets the data packets received from the Transport layer (from TCP protocol with data and TCP header added) and divide them into datagrams. Datagram is a packet that does not have any kind acknowledge system, meaning that IP does not implement any acknowledge system, thus it is an unreliable protocol. The reason is that when transferring data TCP protocol will be used on top, and TCP implements an acknowledge system. Thus even though IP protocol does not check whether the datagram arrived at the destination, the TCP protocol will. The connection will be then reliable, even though

IP protocol alone isn't reliable. Each IP datagram can have a maximum size of 65,535 bytes, including its header, which can use 20 or 24 bytes (from previous section with Transport layer), depending whether a field called "options" is used or not. Thus IP datagrams can carry up to 65,515 or 65,511 bytes of data. If the data packet received from the Transport layer is bigger than 65,515 or 65,511 bytes, IP protocol will cut the packet down in as many datagrams as necessary.

In figure below we illustrate the datagram generated on the Internet layer by the IP protocol. It is interesting to notice that what the Internet layer sees as "data" is the whole packet it got from the Transport layer, which includes the TCP or UDP header. This datagram will be sent to the Physical layer (if we are transmitting data) or was sent from Physical layer (if we are receiving data).

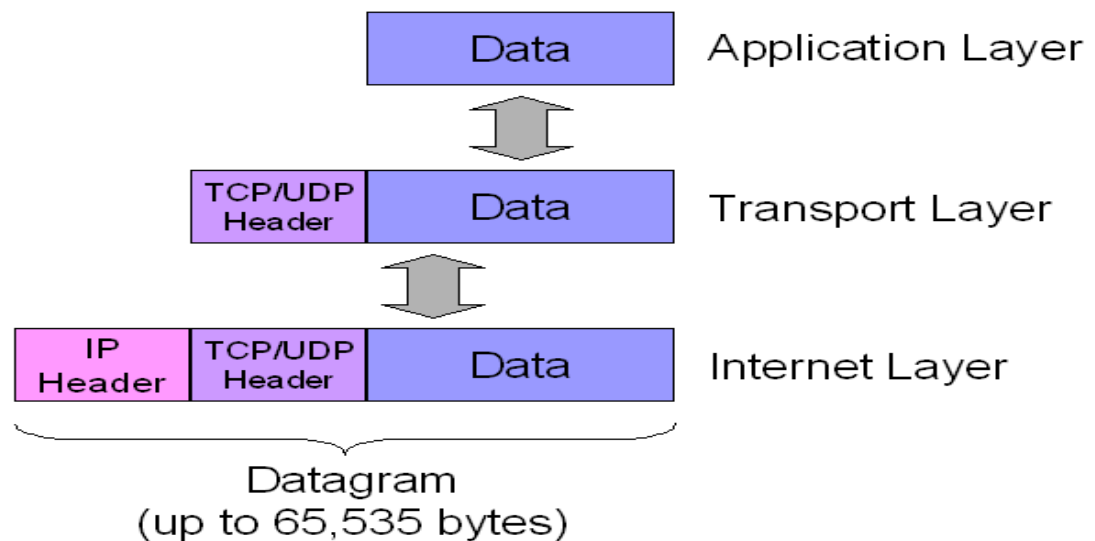


Figure 10. Working of Internet (Network) layer (hardwaresecrets.com)

As we can see from figure the datagram can have up to 65,535 bytes. This means that the data field of the datagram does not have a fixed size. Since datagrams will be sent over the network inside frames produced by the Network Interface layer, usually the operating system will configure the size of the IP datagram to have the maximum size of the data area of the data frames used on your network. The maximum size of the data field of the frames that will be sent over the network is called MTU, Maximum Transfer Unit.

Ethernet networks – which is the most common type of network available and also used in the embedded server in this thesis, including its wireless– can carry up to 1,500 bytes of data, i.e., **its MTU is of 1,500 bytes**. Thus usually the operating system automatically configures the IP protocol to create IP datagrams that are 1,500 bytes long, instead of 65,535 bytes which is not fitted to the frame. **Later then in the Third chapter about**

LwIP, I expressed this problem by setting the length of buffer in memory of microcontroller to 1500 bytes.

Another feature that IP protocol allows is fragmentation we mentioned, until arriving at its destination, the IP datagram will probably pass through several other networks in the middle of the road. If all networks in the path between the transmitting computer and the receiving one use the same kind of network (e.g., Ethernet) then everything is fine, as all routers will work with the same frame structure (i.e., the same MTU size). However, if those other networks are not Ethernet networks, they may use a different MTU size. If that happens, the router that is receiving the frames with the MTU set to 1,500 bytes will cut the IP datagram inside each frame in as many as necessary in order to cross over the network with the small MTU size. Upon arriving at a router that has its output connected to an Ethernet network, this router will re-assemble the original datagram.

In the Figure below, the original frame uses a MTU of 1,500 bytes. When it arrived at a network with a MTU size of 620 bytes, each frame had to be broken into three frames (two with 600 bytes and one with 300 bytes). Then the router at the exit of this network (router 2) re-assembled the original datagram.[10]

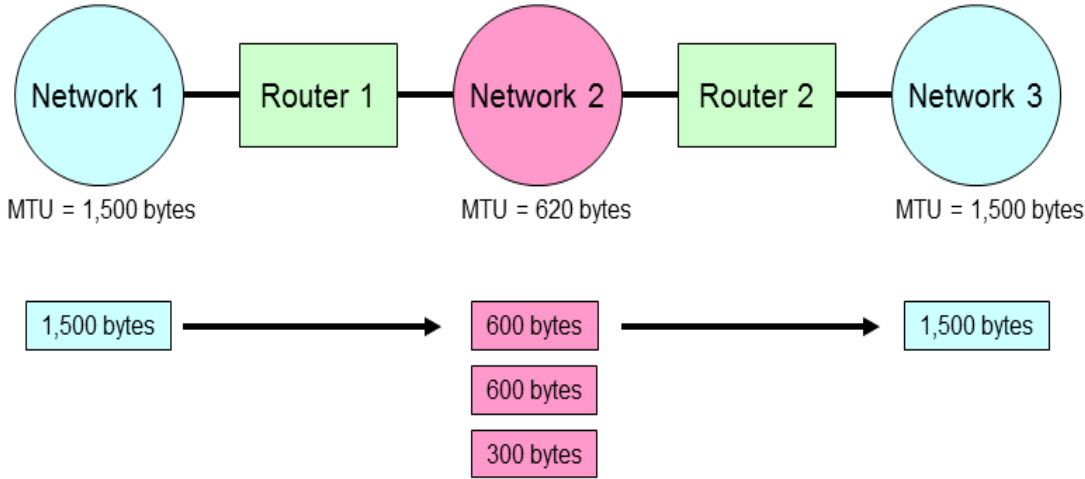


Figure 11. Data fragmentation by routers

The same as in previous section, before sending data to below layer which is Physical layer, Internet layer added the header which is contain the IP addresses and other information for the destination receiver. Thus, the explanation for IP header is necessary in order to understand the function of Internet layer. The figure below is an illustration of IP header which will be added to data.

4-bit	8-bit	16-bit	32-bit	
Ver.	Header Length	Type of Service	Total Length	
Identification			Flags	Offset
Time To Live	Protocol	Checksum		
Source Address				
Destination Address				
Options and Padding				

Figure 12. Header frame of Network layer (ptgmedia.pearsoncmg.com)

Version (Ver): This 4 bits field indicate which version of Internet Protocol (IP) is being used. In this paper work, the IPv4 is used thus we focus only on this version of IP addresses, however the IPv4 is moving out of capacity since the devices on around the world are increasing and the new version IPv6 is being introduces as the next solution. The binary pattern for IPv4 is 0100.

Header length: This 4 bits field gives length of the IPv4 header in 32-bit words. The minimum length of an IPv4 header is 32-bit words. The header bit pattern is 0101.

Type of Services: 8 bits length express the Type of Service (ToS) in the Internet Protocol (IPv4), which provides an indication of the Quality of Service (QoS), such as precedence, congestive, delay, throughput and reliability.

Total Length: The total length is a 16 bits field which identifies the length (in byte) of the Internet Protocol datagram. Total Length includes the length of IPv4 Datagram in 20 bytes (The minimum length of an IP header is 20 bytes and this is the case of an IPv4 header carrying no data) and the maximum is 65,535 bytes. The binary represent of 1500 bytes length will be 0000010111011100 (16 bits).

Identification: Identification field in the Internet Protocol Version 4 (IPv4) header is a 16 bits field which indicates and identifying value assigned by the sender to aid in assembling the fragments of an IPv4 Datagram. When a Data gram is fragmented into multiple Datagrams, IPv4 gives all the fragments the same identification number and this number is used to identify IPv4 fragments at the receiving side.

Flags: The first flag, if set, is ignored. If the DF (Do Not Fragment) flag is set, under no circumstances can the packet be fragmented. If the MF (More Fragments) bit is turned on (1), there are packet fragments to come, the last of which is set to off (0).

Offset: If the Flag field returns a 1 (on), the Offset field contains the location of the missing piece(s) indicated by a numerical offset based on the total length of the packet.

Time To Live (TTL): Typically 15 to 30 seconds, TTL indicates the length of time that a packet is allowed to remain in transit. If a packet is discarded or lost in transit, an indicator is sent back to the sending computer that the loss occurred. The sending machine then has the option of resending that packet.

Protocol: The protocol field holds a numerical value indicating the handling protocol in use for this packet.

Checksum: The checksum value acts as a validation checksum for the header.

Source Address: The source IP address field indicates the address of the sending host.

Destination Address: The destination IP address field indicates the address of the destination client.

Options and Padding

The Options field is optional. If used, it contains codes that indicate the use of security, strict or loose source routing, routing records, and time stamping. If no options are used, the field is called *padded* and contains a 1. Padding is used to force a byte value that is rounded. Table 3.2 indicates the bit counts for the options available.

Note: This part is further information regard to IPv4, The 32 bits of an IPv4 address are broken into 4 octets, or 8 bit fields (0-255 value in decimal notation). For networks of different size, the first one (for large networks) to three (for small networks) octets can be used to identify the network, while the rest of the octets can be used to identify the node on the network.[10]

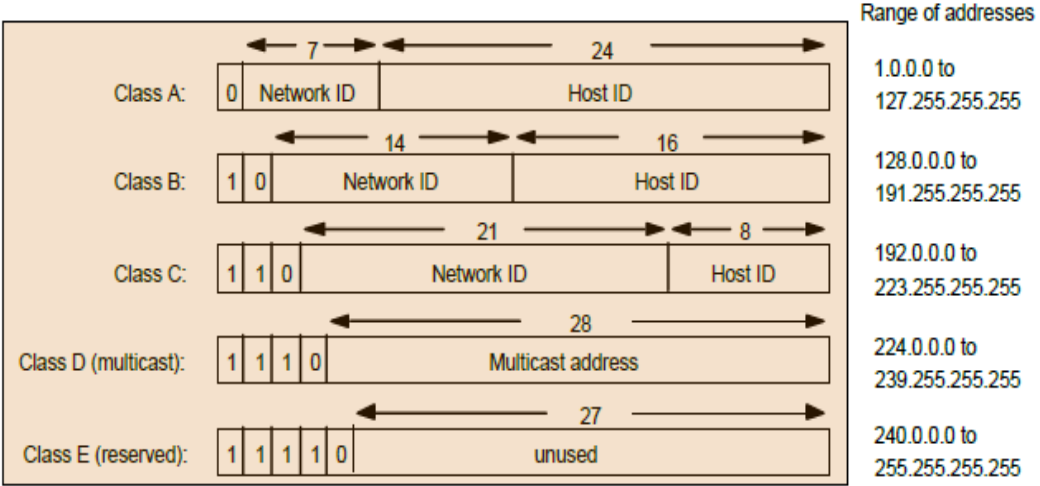


Figure 13. IP address ranges

2.6 Physical layer

Datagrams generated on the Internet layer will be sent down to the Network Interface layer, if we are sending data, or the Network Interface layer will get data from the network and send it to the Internet layer, in case of we are receiving data.

The Logic Link Control layer (LLC) is in charge of adding information of which protocol on the Internet layer delivered data to be transmitted, so when receiving a frame from the network this layer on the receiving computer has to know to which protocol from the Internet layer it should deliver data. This layer is defined by IEEE 802.2 protocol.

The Media Access Control layer (MAC) is in charge of assembling the frame that will be sent over the network. This layer is in charge of adding the source MAC address and the target MAC address – as we explained before, MAC address is the physical address of a network card. Frames that are targeted to another network will use the router MAC address as the target address. This layer is defined by IEEE 802.3 protocol, if a cabled network is being used, or by IEEE 802.11 protocol, if a wireless network is being used.

The Physical layer is in charge of converting the frame generated by the MAC layer into electrical signals (if a cabled network is being used) or into electromagnetic waves (if a wireless network is being used). This layer is also defined by IEEE 802.3 protocol, if a cabled network is being used, or by IEEE 802.11 protocol, if a wireless network is being used.

The LLC and MAC layers add their own headers to the datagram they receive from the Internet layer. So a complete structure of the frames generated by these two layers can be seen in figure below . Notice that the headers added by the upper layers are seen as “data” by the LLC layer. The same thing happens with the header inserted by the LLC layer, which will be seen as data by the MAC layer.

The LLC layer adds a 3-byte or 5-byte header and its datagram has a maximum total size of 1,500 bytes, leaving a maximum of 1,497 or 1,492 bytes for data. The MAC layer adds a 22-byte header and a 4-byte CRC (data correction) data at the end of the datagram received from the LLC layer, forming the Ethernet frame. Thus the maximum size of an Ethernet frame is of 1,526 bytes. [12][13]

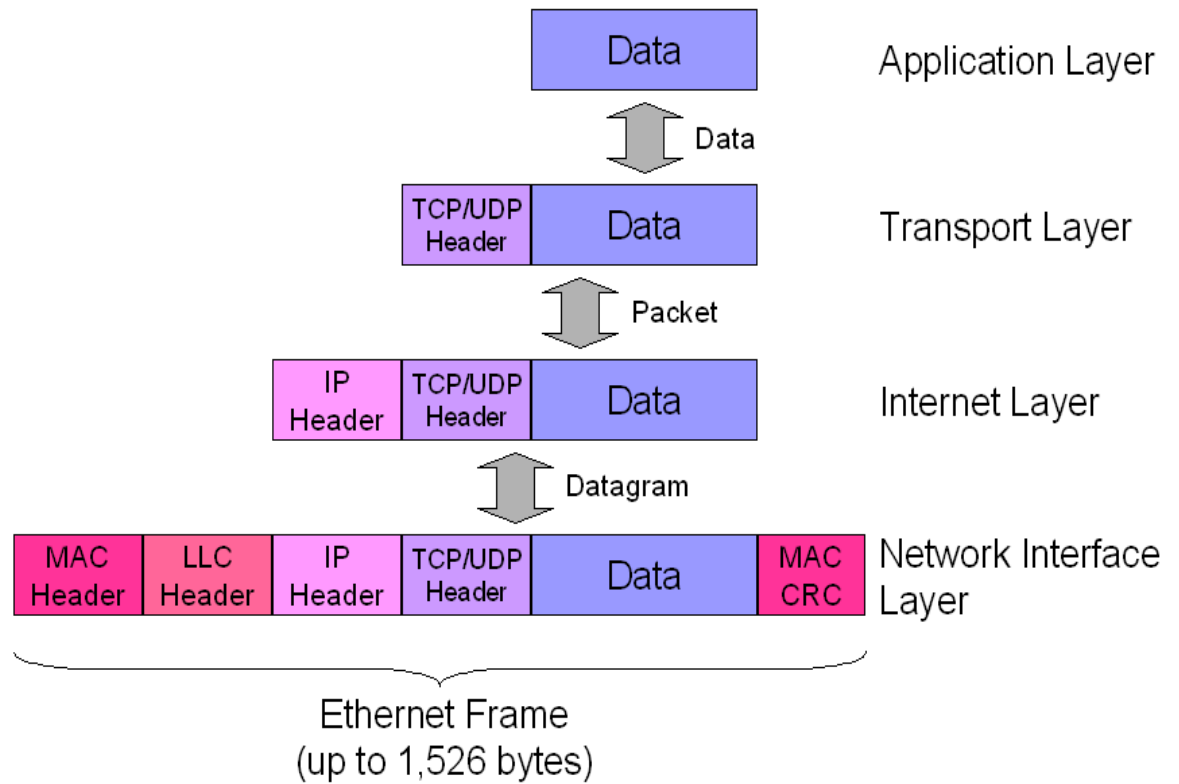


Figure 14. Working of Physical layer (hardwaresecrets.com)

Note: From section 3.5 (Network layer), I explained that the MTU (Maximum Transfer Unit) is set to 1500 bytes for standard Ethernet Frame of any system.

Chapter 3

LwIP Stack

3.1 Introduction to LwIP stack

LwIP is a small independent implementation of the TCP/IP protocol suite that has been initially developed by Adam Dunkels at the Swedish Institute of Computer Science and is now developed and maintained by a worldwide network of developers.

LwIP (lightweight IP) is a set of APIs (Application Programming Interfaces) that contains almost full scale of TCP/IP protocols in order to reduce resources usage. This makes LwIP suitable for using in embedded system with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM. It is a widely used open source TCP/IP stack designed for embedded systems. For such small usage of hardware resources but LwIP stack provides a lot of important characteristics of TCP/IP protocols for each layer of TCP/IP network model. In detail we can see these characteristics as listed below:

- Application layer
 - DNS (Domain Name Services).
 - DHCP (Dynamic Host Configuration Protocol).
 - SNMP (Simple Network Management Protocol).
- Transport layer
 - TCP (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery/fast retransmit.
 - UDP (User Datagram Protocol) including experimental UDP-Lite extensions.
- Internet Layer
 - IP (Internet Protocol) including packet forwarding over multiple network interfaces. (IPv4, IPv6).
 - ICMP (Internet Control Message Protocol) for network maintenance and debugging.
 - IGMP (Internet Group Management Protocol) for multicast traffic management.
- Physical layer
 - PPP (Point-to-Point Protocol).
 - ARP (Address Resolution Protocol) for Ethernet.
- Other
 - Specialized raw/native API for enhanced performance.
 - Optional Berkeley-like socket API.

- AUTOIP / Link-local address (for IPv4, conforms with RFC 3927).

The LwIP used in this thesis which offers three different APIs designed for different purposes:

- Raw API is the core API of lwIP. This API aims at providing the best performances while using a minimal code size. One drawback of this API is that it handles asynchronous events using callbacks which makes the application design more complex.
- Netconn API is a sequential API built on top of the Raw API. It allows multi-threaded operation and therefore requires an operating system. It is easier to use than the Raw API at the expense of lower performances and increased memory footprint.
- BSD Socket API is a Berkeley like Socket implementation (Posix/BSD) built on top of the Netconn API. Its interest is portability. It shares the same drawback than the Netconn API.

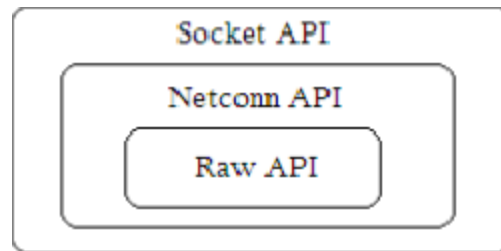


Figure 15. APIs of LwIP

In this work, for developing of the embedded server I chose the **Socket API** which is on the top most of three APIs, **because it keeps all the functionalities of two lower APIs as well as the drawbacks of them**. Another reason led me to choose Socket API because it gives high possibility to interface to the system and developing of applications (we can easily develop web-based multi-tasking later). In next sections 4.2 we will discuss about the socket programming interface in detail.

This Socket API provides ability to connect the 'data' from Application layer to Transport layer and Internet layer. More exactly, this API provides the tool for building first three layers in TCP/IP model which is described in Chapter 3 section 3.2.

For the last bottom layer which is Physical layer (Network interface layer) the combination of Netcoon API and STM32 library is used.

3.2 Socket Programming

A **socket** is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a **port number and a IP address** so that the Transport layer can identify the application that data is destined to be sent to. It is convenient in order to create multi-tasking (or multi applications) we just need to set up the socket which is corresponded to the applications. For example a program can run simultaneously HTTP protocol (uses port 80) and FTP protocol (uses port 20) by using two ports (or sockets) at the same time. For better understanding the figure below expresses the role of a socket in a TCP/IP layer model (which is mentioned above).

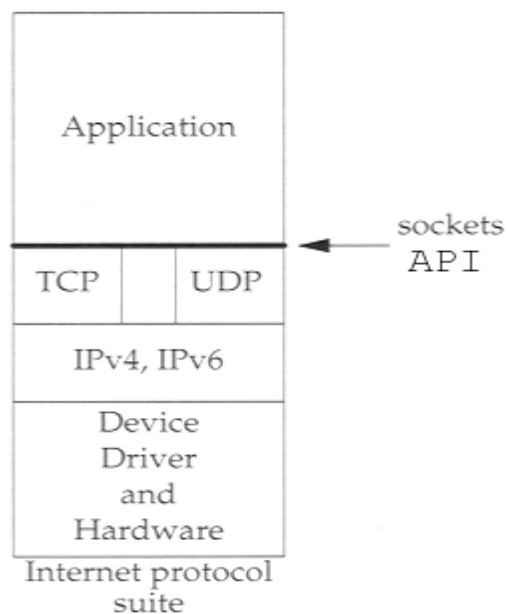


Figure 16. Sockets API in the model of TCP/IP (doc.oracle.com)

For this embedded server, I used TCP protocol in Transport layer so that in this case the socket(s) is called **Stream socket**. As the socket(s) is created by using TCP protocol then it has all the features of TCP protocol (which I mentioned in previous chapter 3.4). On the other hand and we could use also the UDP protocol and in this case it is called **Datagram socket**.

It is now necessary to introduce the new idea, which is client-server, this idea is also attached with the embedded server in this thesis.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

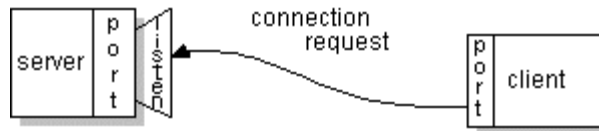


Figure 17. Client sends requests

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.

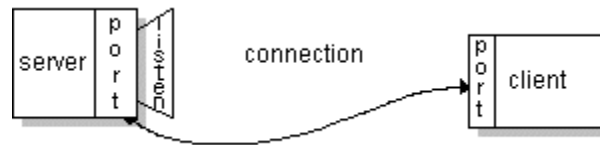


Figure 18. Server accepts and establishes the connection

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

Next conception of socket programming is about its procedures (or its cycles) of working. The next figure gives the overview of how the connection between client and server looks like.

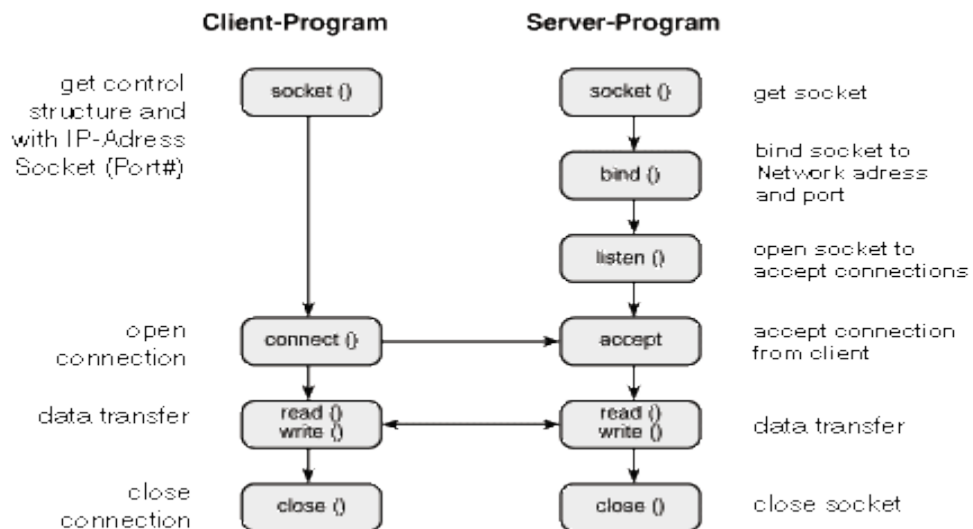


Figure 19. Working cycle of TCP/IP socket

- Socket: Cretes a new socket.

- Bind: Binds a socket to a local IP address and port number.
- Listen: Listens for incoming connections on socket.
- Accept: Accepts and incoming connection on a listening socket.
- Connect: Connects a socket to a remote host using IP address and port number.
- Read: Reads data from a socket.
- Write: Writes data to socket.
- Close: Close a socket.

Implementation of LwIP stack on embedded server

This section contains the implementation of layers of TCP/IP model on embedded server using LwIP stack and other tools, which are introduced in chapter 2 about TCP/IP protocol. The structure of this sections is followed by same structure of TCP/IP model, that means we start from the top most layer to the bottom most layer respectively.

However as I introduced in above section (section 3.2) this thesis only covers the simplest model of network, which is **Client-Server** model. This model is point-to-point model means one server and one client work together at a time. Furthermore the client side is a computer running on given Operating system with all of functionality of TCP/IP. The user can get the webpages and send desired data to server by using any kind of web browser. From that reason in following sections I mainly focused on the explanation of implementation process for server side only.

Before starting any explanation, we should recall that the data processed by TCP/IP protocols are layered data (it is explained in previous chapter 2). The data frame at the end of Physical layer before it is being sent through the network is always cutted with length of 1526 bytes. Because we start from the top most layer (Application layer) so the explanation is based on the data of this layer and continued to the bottom most layer (Physical layer). Thus I considered the data from Ethernet frame is transferred to a buffer (the space memory of microcontroller) with the length of 1500 bytes (this buffer contains the core data from client or data we want to send to client).

Before going in to detail of implementation of each layer, I would like to introduce a feature of LwIP which is used for manage the data among layers. That feature is **pbuf** services.

LwIP manages packet buffers using a data structure called pbuf. **The pbuf structure enables the allocation of a dynamic memory to hold a packet content and lets packets reside in the static memory.** Pbufs can be linked together in a chain. This enables packets to span over several pbufs. LwIP has a specific API for working with pbufs. This API is implemented in the *pbuf.c* core file.

- *“pbuf” can be a single pbuf or a chain of pbufs.*
- *When working with the Netconn API, netbufs (network buffers) are used for sending/receiving data.*

- A netbuf is simply a wrapper for a pbuf structure. It can accommodate both allocated and referenced data.
- A dedicated API (implemented in file netbuf.c) is provided for managing netbufs (allocating, freeing, chaining, extracting data,...).

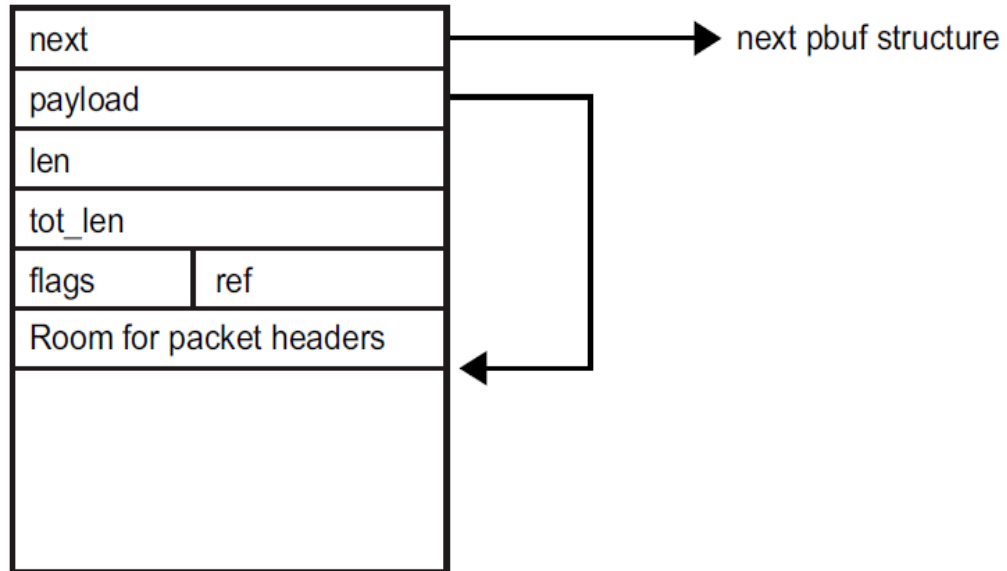


Figure 20. Structure of LwIP buffer for handling data

As we know from theory that TCP/IP protocol encapsulates/decapsulates data through its layers by adding (or extracting) headers. Thus in the next explanations below about implementations of layers of TCP/IP protocol with LwIP are described as using the pbuf. The headers will be written(read)to(from) the pBuf.

3.3 Impelentatin of Application layer

From section 3.3 in chapter 3, as I described before, the microcontroller contains inside its flash memory the embedded websites which are created by HTML (Hyper Text Markup Language) which is most common language for creating website. The picture of sample website could be seen in Index part of this thesis. After HTML web pages were created, it is necessary to transfer these web pages into Hexadecimal form in order these webpages can be stored and processed by Microcontroller. For this purpose, I used the open source software which is provided by opens source community and named **Makefsdata**.

Hence, the webpages are stored in memory we need a method (or Protocol) in order to retrieve the stored pages or sending the data from user to server so that the server can process them.

Of course, the HTTP protocol is the most suitable choice for this layer. The HTTP protocol gives two methods for receiving and sending the data which are "GET" and "POST" methods. As user request the website the HTTP protocol sends a header start with "GET" and following by the information that user wants to retrieve from server. From that idea, we need to program our embedded server so that it can capture this header and answer the client with dedicated information.

For example in this work, I programmed the home webpage which is stored in microcontroller (or in embedded server) as **index.html** so when the use need to retrieve this webpage by typing on web browser the HTTP protocol will send to server this header: "GET /index.html" with the length of 12 characters. After then inside the embedded server we need to trap this header and process them. The piece of code below (in C language) shows how the server acts when it trapped this header.

```
if (strncmp((char *)recv_buffer,"GET /index.html",12)==0)
    {
        file = fs_open("/index.html");
        write(conn, (const unsigned char*)(file->data), (size_t)file->len);
        if(file) fs_close(file);
    }
```

By using the command `strncmp` (string compare) the server search and compare the buffer which contains the input header send from client (server) if it contains the string "GET /index.html" or not. Since the header contains the string above, the server will open the file (or webpage) **index.html** and then send this webpage (the Hexadecimal data which were transferred from HTML file before) by using **write** command to next layers by using socket.

We use "GET" method for retrieving data which are stored in the server, then when we need to send the data to server we use "POST" method. The implementation of the "POST" method is a bit different because after we trapped the "POST" method, we need to process the data which are followed by. Typical POST method is described as below:

```
POST /test/demo_form.asp HTTP/1.1 name1=value1&name2=value2
```

After telling the server the POST method with the action the client wants to perform, the client send all the values need to process. In my work, I used the POST method to perform such a simple task is blinking the LED on embedded server so in a webpage I programmed the POST by setting the values as:

```
POST /blink value1= start &value2=stop
```

Similar to "GET" method, for POST method, we need to trap these values. This process can be seen in following code:

```
if (strncmp((char *)recv_buffer,"POST /blink",6)==0)
    {
        for (i=0;i<buflen;i++)
```

```

        { if (strncmp ((char*)(recv_buffer+i), "Start ", 5)==0)
          { STM_EVAL_LEDOn(LED3);
            break;
          }
        }
else if (strncmp ((char*)(recv_buffer+i), "Stop ", 4)==0)
        { STM_EVAL_LEDOn(LED3);
          break;
        }
}

```

Again the `strncmp` command is used for classifying the type of method, in this case is POST method, also it is used for determining which kind of action the client wants to perform (in this case is `/blink`). In this example I perform the action blinking a led with two sub-actions (stop and start), however, we could use many actions as we want. For example we can perform simultaneously two tasks blinking a led and accessing a SD card (with write and read). In order to use many applications on the same webpage we just change this setting to other value (for example `/blink` can be changed to `/accessSDcard`) and also applied for sub-values. After detecting the POST method, we need to create a loop for searching the given values from client (or user). Then if the server catches the correct value, it will perform the actions.

The whole process for application layer (receiving and sending), can be done by calling the function **`http_server_serve(newconn)`**. This function is set of commands for trapping the GET and POST methods which are described above. We will pay attention to this function because it contains some settings for Transport layer. Finally, if we look back to the figure 19, the explanations which were covered in this section are about the last 3 steps in the diagram. (write, read, close).

3.4 Implementation of Transport layer

For transport layer, as it was mentioned in section 4.2 (socket programming), socket interfacing method is used. Also from previous section, the socket contains the information of Port and IP address which are used. The following piece of code is showing the implementation of Transport layer on the embedded system.

```

static void http_server_socket_thread(void *arg)
{
    int sock, newconn, size;
    struct sockaddr_in address, remotehost;
    /* create a TCP socket */

```

```

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("can not create socket");
    return;
}
/* bind to port 80 at any interface */
address.sin_family = AF_INET;
address.sin_port = htons(80);
address.sin_addr.s_addr = INADDR_ANY;
if (bind(sock, (struct sockaddr *)&address, sizeof (address)) < 0)
{
    printf("can not bind socket");
    return;
}
/* listen for incoming connections (TCP listen backlog = 5) */
listen(sock, 5);
size = sizeof(remotehost);
while (1)
{
    newconn = accept(sock, (struct sockaddr *)&remotehost, (socklen_t *)&size);
    http_server_serve(newconn); }
}

```

As it can be observed from above code, the creating socket process is expressed by **int sock, newconn, size;** and **sock = socket(AF_INET, SOCK_STREAM, 0)**. After creating the socket we need to bind the socket to appropriate port (port 80) for HTTP protocol. **At this state, the IP address is not initialized, the program created the space of memory (buffer) for storing the IP header (which is contain the IP addresses) and fills it by null values (0.0.0.0), that process can be seen by the command address.sin_addr.s_addr = INADDR_ANY. The filling of IP address will be done in next layer which is Network layer.**

Back to our process, after binding the socket, the server start to **listen** any information from client by executing the command.

```

/* listen for incoming connections (TCP listen backlog = 5) */
listen(sock, 5);

```

where,

Socket is the socket has just been created upper and 5 is the listen backlog.

The value 5 is corresponding to header length (which is described before in section 2.4), the connection is waiting for the handshake package with the length of 5 bytes from client.

All the other segments (which are needed to be written into the header of Transport layer (described in section 3)) are automatically done by TCP and Netcon APIs, which are inside the Socket API. All the segments were set to standard value by APIs

themselves. That why the only information we need to provide are port, IP address, MAC address, Default gateway.

Whenever the server finishes its tasks about **creating, binding, listening** a socket, it will **accept** the connection and start to perform the HTTP protocol commands by using the following code:

```
while (1)
{
    newconn = accept(sock, (struct sockaddr *)&remotehost, (socklen_t *)&size);
    http_server_serve(newconn);
}
```

The actions of accepting a connection and serving the HTTP protocol are repeated frequently.

Again we see here the function **http_server_serve(newconn)** because after the connection between the server and client was accepted, the server start to read and write cycles as it was described in the figure 15 above.

Now it is necessary to see how the function **http_server_serve(newconn)** is implemented by observing following code.

```
void http_server_serve(int conn)
{size_t size;
struct fs_file * file;
unsigned char recv_buffer[1500];
ret = read(conn, recv_buffer, buflen);
if(ret < 0) return;
if (strncmp((char *)recv_buffer, "GET / ", 6) == 0)
{
    file = fs_open("/index.html");
    write(conn, (const unsigned char*)(file->data), (size_t)file->len);
    if(file) fs_close(file);
}
..... ( next pieces of code are here, trapping the POST or GET methods)
close(conn);
}
```

As I mentioned above, **the recv_buff is set to 1500 bytes**, that shows the standard MTU is implemented here.

3.5 Implementation of Network layer

The Network layer provides the IP address, MAC address, Default gateway of the server into its header in order to establish the communication between the server and client.

The embedded server is also implemented with function of DHCP (Dynamic Host Configuration Protocol). If DHCP is used, it will provide an automatic assignment of IP address, MAC address, Default Gateway for client. That means we do not need to configure the IP address, Default Gateway for client, when a client device connected to the embedded server it is automatically given an IP address which is generated by server.

However, in this work, I did not turn on DHCP function for server, thus, it is necessary to configure the IP address and Default Gateway for the client manually. The IP address which is being configured for client must be different from the IP address of the server.

More detail for this section, in this thesis, the server is configured by following information:

- IP address of server: 192.168.0.10
- MAC address of server: 02:00:00:00:00:00
- Default Gateway: 192.168.0.10
- Net Mask: 255.255.255.0

Note: A **netmask** is a 32-bit mask used to divide an IP address into subnets and specify the network's available hosts. In a netmask, two bits are always automatically assigned. For example, in 255.255.225.0, "0" is the assigned network address. In 255.255.255.255, "255" is the assigned broadcast address. The 0 and 255 are always assigned and cannot be used.

On the other hand, for client the configurations should be similar as:

- IP address for client: 192.168.xxx.xxx (where x can be any number that user want to set)
- MAC address for client is set by manufacturer of the client device's network adapter.
- Default Gateway for client: 192.168.0.10 (this address should be set to the same value as server in case the is one more network served by the server, but the embedded server in this work only works with one client at a time, thus, we do not need to change the default Gateway.
- NetMask for client should be set the same as in server.(255.255.255.0)

That from the user point of view for configuration of the address for establishing the network. Now we back again to the implementation of the server on microcontroller, at this stage, the buffer of data (or frame data) which was sent from upper layers (Application and Transport layers) to this layer, contains the information from user (or

to user) with the header (inside it are information about ports and others segments of TCP protocol) . The implementation of network layer is done by adding data which are listed above (IP addresses, MAC addresses, Default Gateway, Netmask) in to the new header. For doing such that action, let's consider the following code:

```

/* MAC ADDRESS*/
#define MAC_ADDR0 02
#define MAC_ADDR1 00
#define MAC_ADDR2 00
#define MAC_ADDR3 00
#define MAC_ADDR4 00
#define MAC_ADDR5 00
/*NETMASK*/
#define NETMASK_ADDR0 255
#define NETMASK_ADDR1 255
#define NETMASK_ADDR2 255
#define NETMASK_ADDR3 0
/*Gateway Address*/
#define GW_ADDR0 192
#define GW_ADDR1 168
#define GW_ADDR2 0
#define GW_ADDR3 1
IP4_ADDR(&ipaddr, IP_ADDR0, IP_ADDR1, IP_ADDR2, IP_ADDR3);
IP4_ADDR(&netmask, NETMASK_ADDR0, NETMASK_ADDR1 , NETMASK_ADDR2,
NETMASK_ADDR3);
IP4_ADDR(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2, GW_ADDR3);

```

The function **IP_ADDR** is being called from Netcoon api which provides the ability to record the addresses in to the header of frame data. This function **IP_ADDR** is inside the function **LwIP_Init()**, which response for completing the initialization of this layer.

3.6 Implementation of Physical layer

The physical layer on the embedded system is represented by how the data are encapsulated and physically sent by Ethernet connection. From that reason, I will explain the implementation of Ethernet with MAC/DMA functions. The MAC address, which is set from previous layer (Network layer or Internet layer) and in this layer it is used as the final header for the data frame (in previous, we set the MAC address by in to the format with the desired number but in this layer it is attached to the frame of data). The DMA (Direct Memory Access) function is supported by STM32F4 microcontroller.

That means at the end of data flow (after three previous layers), the data which are encapsulated will be allocated in the static memory, the DMA function will just reads/writes the data from the static memory and transfers it through Ethernet. The reason for using DMA in this case is to improve the performance of the server, DMA provides preparations for data transferring by many methods (FIFO, Full duplex, half duplex, etc..)

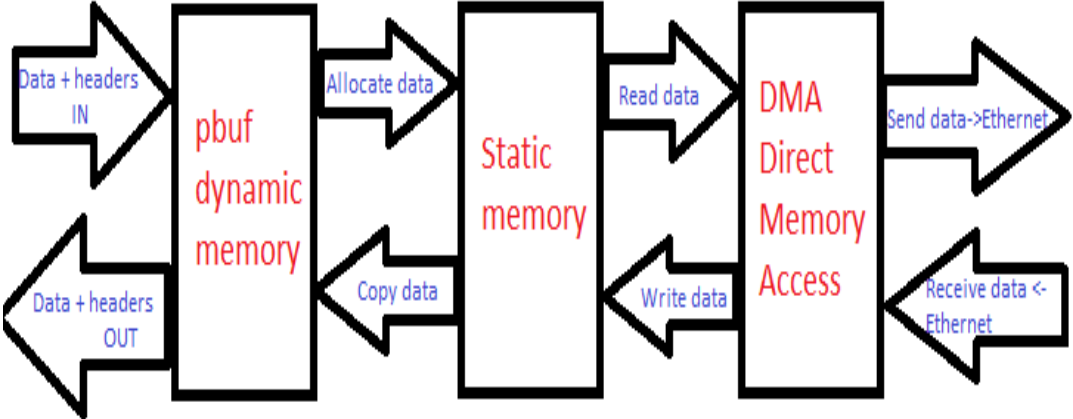
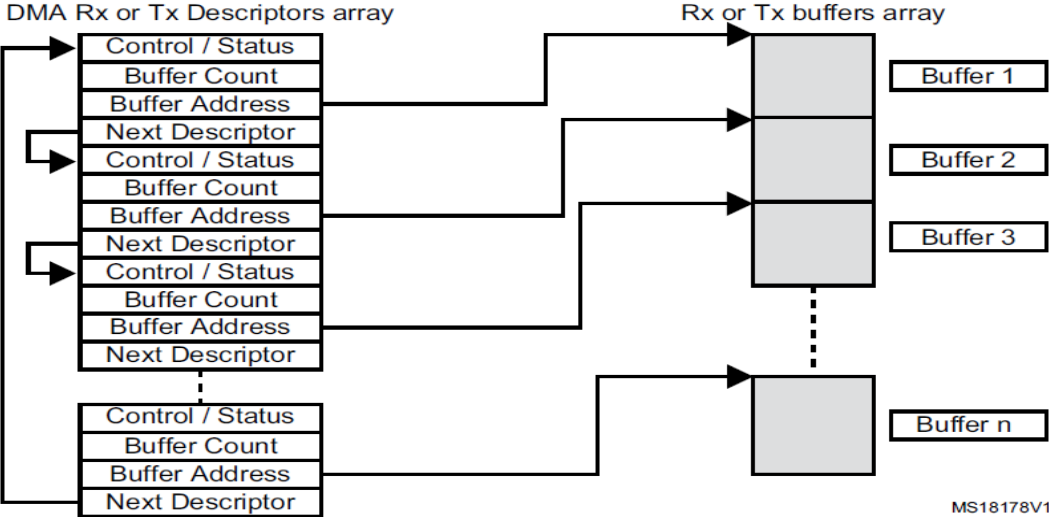


Figure 21. Data flow in Physical layer

The diagram below shows the method of communication between DMA and static memory.



MS18178V1

Figure 22. Communication between DMA and static memory

The following arrays are statically allocated in the STM32F4x7

Ethernet driver:

- Two arrays for the DMA descriptors, one for DMA Rx and another for DMA Tx:
 - `ETH_DMADESCTypeDef DMARxDscrTab[ETH_RXBUFNB], DMATxDscrTab[ETH_TXBUFNB];`
- Two arrays of driver buffers, one array for receive buffers and another array for transmit buffers:
 - `uint8_t Rx_Buff[ETH_RXBUFNB][ETH_RX_BUF_SIZE];`
 - `uint8_t Tx_Buff[ETH_TXBUFNB][ETH_TX_BUF_SIZE];`

where:

- `ETH_RXBUFNB`: number of driver receive buffers
- `ETH_TXBUFNB`: number of driver transmit buffers
- `ETH_RX_BUF_SIZE`: size in bytes of a receive buffer
- `ETH_TX_BUF_SIZE`: size in bytes of a transmit buffer

The default values for these parameters as defined in file *stm32f4x7_eth.h* are:

- `ETH_RXBUFNB = 4`
- `ETH_TXBUFNB = 2`
- **`ETH_RX_BUF_SIZE = 1524 (max size of Ethernet packet (1524) + 2 bytes for 32-bit alignment)`**
- **`ETH_TX_BUF_SIZE = 1524 (max size of Ethernet packet (1524) + 2 bytes for 32-bit alignment)`**

The above parameter values can be changed depending on user specific application needs. This can be done by enabling `CUSTOM_DRIVER_BUFFERS_CONFIG` and writing custom values in the *stm32f4x7_eth_conf.h* configuration file.

As we can see that we set the value of buff size is 1526 bytes which is satisfied with the condition we mentioned in section 2.6 (chapter 2).

Finally, whole process of implementation for this layer can be express by following funtion:

```
void ETH_BSP_Config(void)
    /* Configure the GPIO ports for ethernet pins */
    ETH_GPIO_Config();
    /* Config NVIC for Ethernet */
    ETH_NVIC_Config();
    /* Configure the Ethernet MAC/DMA */
    ETH_MACDMA_Config();
    while(1);
}
```

This function when it will be called, it does the initialization of Physical layer for the embedded system.

In general, for configuration of the server, we need to using following functions:

- **ETH_BSP_Config()** (Initializes Physical layer)
- **LwIP_Init()** (Initializes Transport and Network layers)
- **Http_server_socket_init()** (Initializes Application layers)

Because I used the FreeRTOS, thus at this stage for three functions which are mentioned above, whenever these functions are called when the server start its initializations, they creates corresponded “**tasks**” for scheduler of FreeRTOS. These problems will be explained in following chapter.

Chapter 4

FreeRTOS

4.1 Introduction to FreeRTOS

FreeRTOS is a class of RTOS that is designed to be small enough to run on a microcontroller - although its use is not limited to microcontroller applications. A microcontroller is a small and resource constrained processor that incorporates, on a single chip, the processor itself, read only memory (ROM or Flash) to hold the program to be executed, and the random access memory (RAM) needed by the programs it executes. Typically the program is executed directly from the read only memory. Microcontrollers are used in deeply embedded applications that normally have a very specific and dedicated job to do. The size constraints, and dedicated end application nature, rarely warrant the use of a full RTOS implementation - or indeed make the use of a full RTOS implementation possible. FreeRTOS therefore provides the core real time scheduling functionality, inter-task communication, timing and synchronisation primitives only. This means it is more accurately described as a real time kernel, or real time executive. Additional functionality, such as a command console interface, or networking stacks, can be then be included with add-on components. From now we can use the word **scheduler** for describing the functionalities of FreeRTOS. Before we go more in detail, we need to consider some basic ideas for FreeRTOS which are **Task, task states, task priorities**. [16]

- **Task:** Each executing program is a **task** (or thread) under control of the operating system. If an operating system can execute multiple tasks in this manner it is said to be **multitasking**.
- **Task States:** A task can exist in one of following state
 - **Running:** A task is actually executing it is said to be in the Running state. It is currently utilising the processor. If the processor on which the RTOS is running only has a single core then there can only be one task in the Running state at any given time.
 - **Ready:** Ready tasks are those that are able to execute (they are not in the Blocked or Suspended state) but are not currently executing because a different task of equal or higher priority is already in the Running state.
 - **Blocked:** A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event.[17]

- **Suspended:** Like tasks that are in the Blocked state, tasks in the Suspended state cannot be selected to enter the Running state, but tasks in the Suspended state do not have a time out.
- **Task priorities:** Each task is assigned a priority from 0 to (configMAX_PRIORITIES - 1), where configMAX_PRIORITIES is defined within FreeRTOSConfig.h. Low priority numbers denote low priority tasks. The idle task has priority zero (tskIDLE_PRIORITY). The FreeRTOS scheduler ensures that tasks in the Ready or Running state will always be given processor (CPU) time in preference to tasks of a lower priority that are also in the ready state. **In other words, the task placed into the Running state is always the highest priority task that is able to run.**

A real time application that uses an RTOS can be structured as a set of independent tasks. Each task executes within its own context with no coincidental dependency on other tasks within the system or the RTOS scheduler itself. Only one task within the application can be executing at any point in time and the real time RTOS scheduler is responsible for deciding which task this should be. The RTOS scheduler may, therefore, repeatedly start and stop each task (swap each task in and out) as the application executes. As a task has no knowledge of the RTOS scheduler activity it is the responsibility of the real time RTOS scheduler to ensure that the processor context (register values, stack contents, etc) when a task is swapped in is exactly that as when the same task was swapped out. To achieve this each task is provided with its own stack. When the task is swapped out the execution context is saved to the stack of that task so it can also be exactly restored when the same task is later swapped back in. The scheduler deals with the task by using APIs such as vTaskResume, vTaskSuspend. The figure gives the view of task and its states and how it can be manage by scheduler.

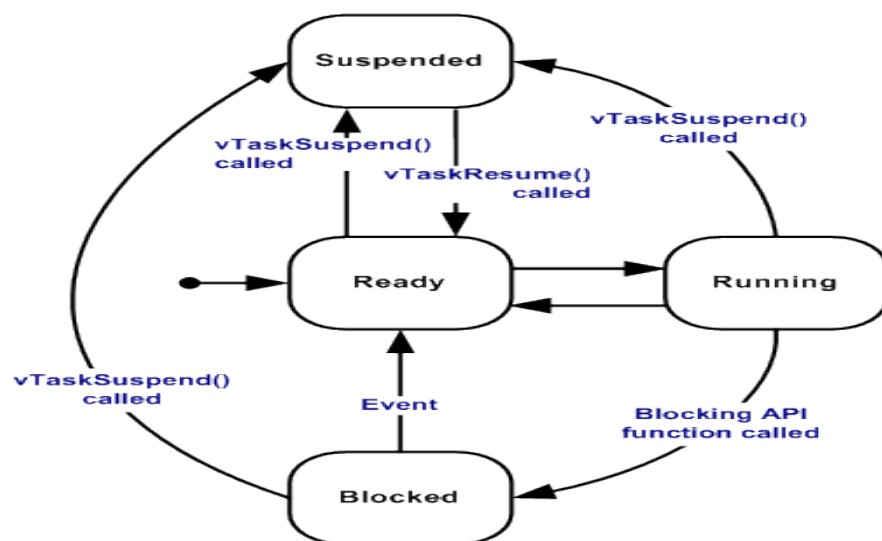


Figure 23. Task states cycle in freeRTOS (freertos.org)

4.2 Using FreeRTOS for developing application on embedded server

From the beginning of this thesis, I introduced about particular application which is being developed for this embedded server by using expansion PLC board (PLC 01A1). The PLC board works by reading inputs values, processes it by initialized command and simultaneously sending to outputs. The PLC board provides some following functions:

- Turn off all outputs
- Turn on all outputs
- AND function (gives the results on outputs are AND combinations of inputs values with netmask bits which are provided by user through web interface)
- XOR function (gives the results on outputs are XOR combinations of inputs values with netmask bits which are provided by user through web interface)
- OR function (gives the results on outputs are OR combinations of inputs values with netmask bits which are provided by user through web interface)
- NOT function reverses all the given values of inputs to the outputs.
- INPUT MIRROR function gives the outputs corresponded values from inputs.

Before going into details of implementation of above functions for PLC, I would like to mention about the whole system working with FreeRTOS as in following figure:

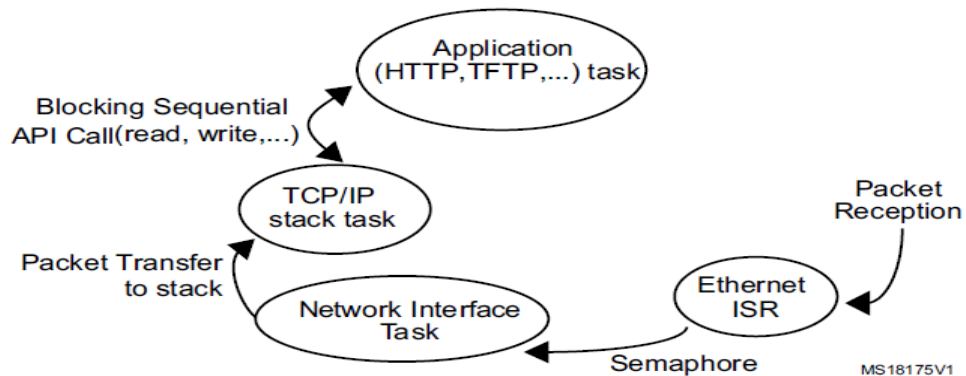


Figure 24. Main working cycle of the embedded server

As we can see from the figure, there are three tasks created by FreeRTOS and these are:

- Ethernet task
- TCP/IP task
- HTTP task

The Ethernet task was given highest priority and added with semaphore and then TCP/IP task with lower priority, the last task with lowest priority is HTTP task. When the whole system in working state, the Ethernet task with highest priority should be always in running state, however for saving the time in order we need to do something else, we

set the Ethernet with semaphore function. That means the interruption was set to Ethernet task and whenever there is a packet that needed to be received or transfer (the server always listens to client if the client wants to request or send data) this task will wake up. By that reason, the Ethernet task was put in blocked state and it only able to run unless there will be a packet is coming. The TCP/IP task is also added to Ethernet task by using semaphore but lower priority and also it is in blocked state. When the packet is received, the Ethernet and TCP/IP both wake up, however the Ethernet will work first because it has highest priority and then when it finish its duty, the TCP/IP task will start to work immediately. The task HTTP is always set to ready state but have lower priority means it is always ready to run after any higher priority tasks finish their cycles. [3]

Note: I denoted from now the working of three tasks above is main cycles. That means at the end of this main cycle (HTTP task is the final task of main cycle) are the data that the server receives from client (or send to client if client made a request).

Base on the main cycle of working (see figure 24), for developing the application of PLC board, we need to create seven tasks which are corresponded to the functions of PCL board (mentioned at the beginning of this section) and put them together into main cycle of embedded server. When one of the application task is called, it will execute following sub-tasks which are demonstrated in figure below.

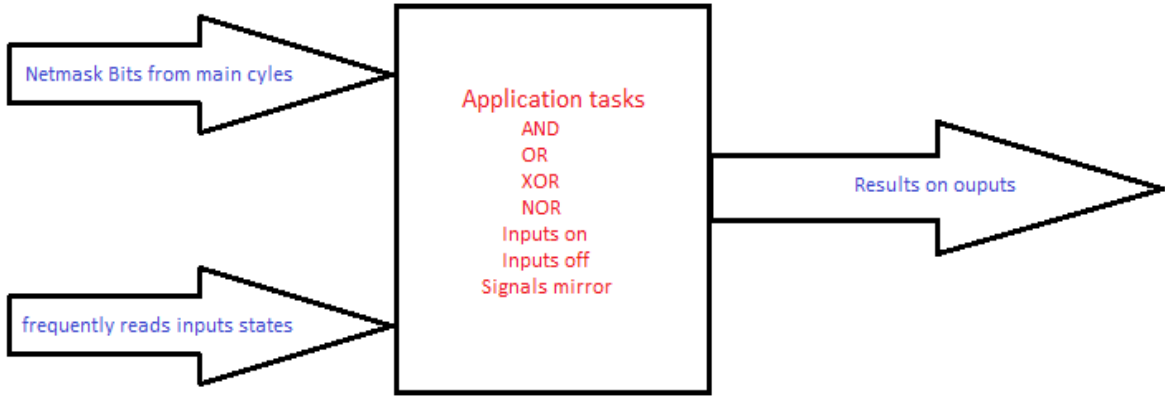


Figure 25. Working principle of Application tasks

After creating seven tasks, the only problem we need to solve is how to manage these tasks in order they can work with the main cycle. The solution is that we put all these seven tasks into suspend state. The reason for this solution is whenever the main cycle finishes, the HTTP task retrieves the data to client or store the command (or data) to server itself, furthermore, in this case the server receives the commands and netmask bits from user and then we just need to deal with these data. More exactly, we just need to add these seven tasks in the task list of the scheduler, after then when the

command from HTTP task is specified as an application task the scheduler switch the corresponded task to working state by calling vTaskResume. It is just adding one more process into the main cycle, we can see in the figure below for more detail.

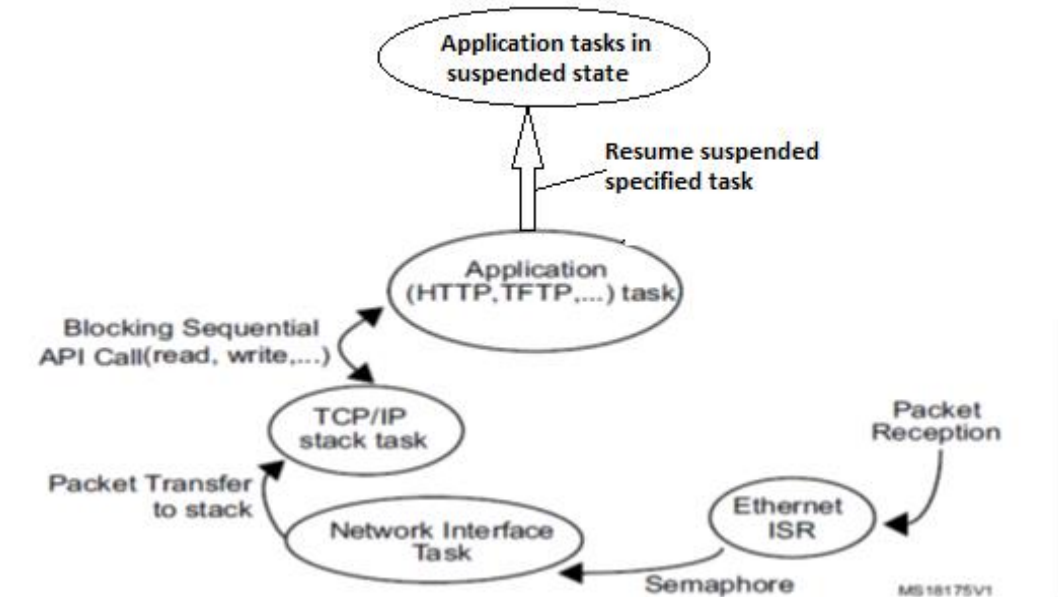


Figure 26. Control application tasks in main cycle

In explanations above, I covered the ideas for implementation of application tasks with FreeRTOS. The real implementation of mentioned process will be shown in next section.

- The creation of the application “**turn on outputs**” task is expressed by following code:

```
xTaskCreate(output_on, "on", configMINIMAL_STACK_SIZE, NULL, LED_TASK_PRIO,
&xon);
void output_on(void * pvParameters)
{ for ( ;; )
  DigitalInputArrayHandler();
  SsrelayHandler_on();
}
}
```

Note: Others functions are implemented as the same method

- Setting all application tasks into suspended state is expressed by following code:
vTaskSuspend(xon);
vTaskSuspend(xoff);
vTaskSuspend(xmirror);
vTaskSuspend(xand);


```
vTaskSuspend(x_or);  
vTaskSuspend(xnot);  
vTaskSuspend(xxor);
```

- Connecting the application tasks with main cycle is expressed by following code (in this case is "turn on outputs"):

```
for (i=0;i<bufflen;i++)  
{  
    if (strncmp ((char*)(recv_buffer+i), "outputon", 8)==0)  
    {  
        vTaskResume(xon);  
        vTaskSuspend(xoff);  
        vTaskSuspend(xmirror);  
        vTaskSuspend(xand);  
        vTaskSuspend(x_or);  
        vTaskSuspend(xnot);  
        vTaskSuspend(xxor);  
        file = fs_open("/outputon.html");  
        write(conn, (const unsigned char*)(file->data), (size_t)file->len);  
        if(file) fs_close(file);  
        break;  
    }  
}
```

Note: Others functions are implemented as the same method

4.3 Demo Webpages and demonstration of server

192.168.0.10/index.html

 **ČESKÉ
VYSOKÉ
UČENÍ
TECHNICKÉ
V PRAZE**

**FAKULTA
ELEKTROTECHNICKÁ**
Spojujeme elektrotechniku a informatiku

DEPARTMENT OF ELECTROTECHNOLOGY
K13113
DEMONSTRATION OF EMBEDDED SERVER WITH LWIP AND TCP/IP PROTOCOL


Home page	Application page
------------------	-------------------------

Introduction

This is demonstration of embed web server from studing of TCP/IP protocal and its application on microprocessor. It is also a part of bachelor thesis which is assigned for student from Department of Electrotechnology in the faculty of electrical engineering in The Czech technical university in Prague. The following aspects are considered in this project:

- Impelentation of TCP/IP protocal on Microprocessor.
- Based on LwIP API.
- Supported by FreeRTOS.
- Simple application for controlling extend board.

Instruction how to use this demonstration and addicional information are described below.



Technical specification

This demonstration based on LwIP socket and TCP/IP and it performs applications of http web server. This server performs common applications of an embedded http server however it is implemented using socket programing and FreeRTOS therefore it optimized multitasking services for other applications on embedded systems. The technical specification of this project is describe...

Figure 27. Home page of the embedded server

This page is always loaded first whenever the server starts and user typed the IP address of server in web browser.

There are two possibilities which user can choose:

- Again load the home page
- Access the application page (click on tap)

CTU IN PRAGUE
DEPARTMENT OF ELECTROTECHNOLOGY
K13113
DEMONSTRATION OF EMBEDDED SERVER WITH LWIP AND TCP/IP PROTOCOL

Back to Home page
Real-time control
Real-time monitoring

REMOTE CONTROL PLC BOARD

This page is application for controlling of PLC board
 Please select the command you want to use by clicking on the button to send command to PLC board
 If you want to force stop PLC's operation please click button "Force Stop"

Commands to PLC	Status of command / instruction	PLC state
Turn on all outputs	<input type="button" value="output"/>	Not Running
Turn off all outputs	<input type="button" value="outputoff"/>	Not Running
Mirror signals	<input type="button" value="outputmirror"/>	Not Running
Input AND	<input type="button" value="inputs_and"/>	Not Running
Input OR	<input type="button" value="inputs_or"/>	Not Running
Input NOT	<input type="button" value="inputs_not"/>	Not Running
Input XOR	<input type="button" value="inputs_xor"/>	Not Running
<input type="button" value="Force stop"/>		

Figure 28. Demonstration of Controlling of PLC board

This is the application page when user clicked on Application page tab (mentioned above). In this web page, there are user interface for real time control of the PLC board with listed functions (Referred to section 4.2). Besides the application functions, user can choose other possibilities:

- Back again to Home page (mentioned in figure 27 above)
- Reload the real time control page (application page)
- Access the Real-time monitoring page (described below)

DEPARTMENT OF ELECTROTECHNOLOGY

K13113

DEMONSTRATION OF EMBEDDED SERVER WITH LWIP AND TCP/IP PROTOCOL

[Back to Home page](#)
[Real-time control](#)
[Real-time monitoring](#)

Real time monitoring
31

Name	State	Priority	Stack	Num
HTTP	R	3	274	2
IDLE	R	0	106	10
Eth_if	B	6	308	1
TCP_IP	B	5	894	0
and	S	1	118	6
or	S	1	118	7
ot	S	1	118	8
xor	S	1	118	9
on	S	1	118	3
off	S	1	118	4
mirror	S	1	118	5

B : Blocked, R : Ready, D : Deleted, S : Suspended

Figure 29. Real time monitoring the state of tasks inside Microcontroller

The last page of the embedded server demonstrates real-time monitoring function. It shows the states of the tasks which are listed in the scheduler of FreeRTOS as well as their related information (stack of memory, Number of task, priority of the task). The updating of this webpage are done by server every half of a second (0.5s). There is a counter for counting the number of loading time since this web page is being loaded. Similar to application page, this web page provides three possibilities for user:

- Back again to Home page (mentioned in figure 27 above)
- Reload the real time control page (application page in figure 28)
- Reload the Real-time monitoring page

Chapter 5

Conclusion

The aim of this thesis was to exploit the functionalities of TCP/IP protocol for embedded system, once the server was established the connection with a client (PC) then it is supposed to perform some particular application tasks.

The development of the embedded server was based on LwIP which is a set of C source code contains the implementation of TCP/IP protocol for embedded system. LwIP was used as a library for programming purpose. Nowadays, there are many types of library which are being used for developing an embedded server or Network function of embedded devices and those are similar to LwIP such as uIP, LUA, etc. However, LwIP is the most common because the size of LwIP is small, that makes it compatible with most of microcontrollers. The LwIP stack is obligatory for working of the embedded server. It is responses for all characteristics as well as functionalities of TCP/IP, for all four layers of TCP/IP.

The embedded server mentioned in this thesis contains inside its flash memory embedded web pages which are built by HTML language. Because of limitation of HTML's functionalities then the web pages are also limited. We can create web pages with more functionalities and animations by using other programming language such as PHP, Java script or others.

The application of the embedded server is controlling the expansion board which is used in industrial. However, we can develop many other applications based on the embedded server such as stream data, TFTP server (for storing data), monitoring center, ect.

On the top of TCP/IP and application programs is FreeRTOS, using FreeRTOS reduces the complexity of the system. Under the point of view of programmer, using FreeRTOS helps to create and manage the whole system easier and optimizes the utilization of resources. Its advantages are suitable for developing of middle-scale to big-scale embedded system. In this thesis I used a part of FreeRTOS because the embedded system is not complicated.

Finally, TCP/IP protocol provides huge abilities to develop embedded system with networking function. The protocol is industrial standard therefore it is used in many devices and supported by many manufacturers. It is not only used in wired network but also in wireless network. Furthermore, TCP/IP protocol is suitable for all network regardless the scale of system and hardware flat-form in which the network is installed. In this thesis, the results of the embedded server emphasize the viability of using TCP/IP protocol on embedded system and development of applications. It is old protocol but still offers a promising solution for embedded systems in futures.

Bibliography

- [1] Dunkels, Adam. Design and Implementation of the LwIP TCP/IP stack [online]. 2001 [cit.2016-02-18].
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.1795>
- [2] AN4739 Application note: STM32Cube firmware examples for STM32F4 Series [online].
STMicroelectronics, 2015 [cit. 2016 -02-08].
http://www.st.com/content/ccc/resource/technical/document/application_note/ca/34/db/d3/71/50/44/9b/DM00213525.pdf/files/DM00213525.pdf/jcr:content/translations/en.DM00213525.pdf
- [3] AN3966 Application note: LwIP TCP/IP stack demonstration for STM32F4x7 microcontrollers.
STMicroelectronics,2013.
http://www.st.com/content/ccc/resource/technical/document/application_note/fd/5d/64/cf/7c/38/4c/30/DM00036052.pdf/files/DM00036052.pdf/jcr:content/translations/en.DM00036052.pdf
- [4] AT04055 Application note: Using the lwIP Network Stack.
Atmel Cooperation, 2014.
http://www.atmel.com/Images/Atmel-42233-Using-the-lwIP-Network-Stack_AP-Note_AT04055.pdf
- [5] UM1918 User manual: Getting started with the X-NUCLEO-PLC01A1 industrial input/output expansion board for STM32 Nucleo.
STMicroelectronics 2016.
http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/9c/25/64/62/4f/bc/4d/9f/DM00213568/files/DM00213568.pdf/jcr:content/translations/en.DM00213568.pdf
- [6] STM32F4DISCOVERY: Discovery kit with STM32F407VG MCU. STMicroelectronics [online]. 2016 [cit. 2016-02-18].
<http://www.st.com/en/catalog/tools/FM116/CL1620/SC959/SS1532/LN1848/PF252419>
- [7] TCP/IP android application by Saitutorial, author venkat4java7@gmail.com, Version 2.0.2, December 21,2015.
<https://play.google.com/store/apps/details?id=com.saiuniversalbookstore.tcpip>
- [8] Author Gabriel Torres: How TCP/IP Protocol works – Part1 Introduction, 28 March 2012.
<http://www.hardwaresecrets.com/how-tcp-ip-protocol-works-part-1/>

[9] Author Gabriel Torres: How TCP/IP Protocol works – Part1 Application layer, 28 March 2012.

<http://www.hardwaresecrets.com/how-tcp-ip-protocol-works-part-1/2/>

[10] Author Gabriel Torres: How TCP/IP Protocol works – Part1 Transport layer, 28 March 2012.

<http://www.hardwaresecrets.com/how-tcp-ip-protocol-works-part-1/3/>

[11] Author Gabriel Torres: How TCP/IP Protocol works – Part1 Internet layer, 28 March 2012.

<http://www.hardwaresecrets.com/how-tcp-ip-protocol-works-part-1/4/>

<http://www.hardwaresecrets.com/how-tcp-ip-protocol-works-part-1/5/>

[12] Author Gabriel Torres: How TCP/IP Protocol works – Part1 Network interface layer, 28 March 2012.

<http://www.hardwaresecrets.com/how-tcp-ip-protocol-works-part-1/6/>

[13] Article: What Is a Socket?

<http://docs.oracle.com/javase/tutorial/networking/sockets/definition.html>

[14] Ece Gelal: Introduction to Socket Programming note,
Department of Computer Science & Engineering, University of California, Riverside.

<http://alumni.cs.ucr.edu/~ecegela/TAW/socketTCP.pdf>

[15] HIMANSHU ARORA: TCP/IP Protocol Fundamentals Explained with a Diagram,
2 November 2011.

<http://www.thegeekstuff.com/2011/11/tcp-ip-fundamentals/>

[16] Article: What is An RTOS?

<http://www.freertos.org/about-RTOS.html>

[17] Article: Tasks

<http://www.freertos.org/RTOS-task-states.html>