

**DESIGN AND IMPLEMENTATION
OF A TCP/IP STACK FOR GEEKOS
OPERATING SYSTEM**

Alper BİLGE
Master of Science Thesis

Computer Engineering Program
June, 2008

JÜRİ VE ENSTİTÜ ONAYI

Alper Bilge'nin "GeekOS işletim sistemi için TCP/IP yığını tasarımı ve gerçekleştirilmesi" başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki, Yüksek Lisans Tezi 27.06.2008 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

Adı-Soyadı		İmza
Üye (Tez Danışmanı):	Yard. Doç. Dr. CÜNEYT AKINLAR
Üye	: Yard. Doç. Dr. EMİN GERMEN
Üye	: Yard. Doç. Dr. HÜSEYİN POLAT

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun
..... tarih ve sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ABSTRACT**Master of Science Thesis****DESIGN AND IMPLEMENTATION OF A TCP/IP STACK FOR GEEKOS
OPERATING SYSTEM****Alper BİLGE****Anadolu University
Graduate School of Sciences
Computer Engineering Program****Supervisor: Assist. Prof. Dr. Cüneyt AKINLAR
2008, 74 pages**

GeekOS is a tiny and simple operating system that was designed at the University of Maryland to show undergraduate students the fundamentals of operating systems. GeekOS was implemented and extended as a senior project at Anadolu University in 2006, and the new kernel was named OSman. While preserving the basic functionality available in GeekOS, OSman incorporates additional features such as multi-programming support, VESA graphics device driver, and a PCI Device Driver System Base.

Despite having many nice features, the current version of OSman lacked crucial networking support. The goal of this thesis was to add a sample Ethernet device driver and a TCP/IP Protocol stack implementation so that networked applications can be deployed in a machine running OSman. Realtek RTL8139 is used as the network interface card (NIC) for this thesis, and a standards-compliant, five-layer TCP/IP protocol stack is implemented on top of the designed device driver.

To demonstrate the capabilities of the developed Ethernet device driver and the TCP/IP, a popular client-server network game called tic-tac-toe is implemented. This game shows the communication between two computers one running OSman and the other running in Windows XP.

Keywords: OSman, GeekOS, Operating Systems, Network Interface Card Driver, PCI Device Driver, TCP/IP Protocol Stack.

ÖZET

Yüksek Lisans Tezi

GEEKOS İŞLETİM SİSTEMİ İÇİN TCP/IP YIĞINI TASARIMI VE GERÇEKLEŞTİRİLMESİ

Alper BİLGE

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Cüneyt AKINLAR
2008, 74 sayfa

GeekOS Maryland Üniversitesi'nde lisans öğrencilerine işletim sistemleri temellerini göstermek üzere tasarlanmış küçük ve basit bir işletim sistemidir. GeekOS 2006 yılında Anadolu Üniversitesi'nde bir bitirme projesi olarak gerçekleştirildi ve genişletildi. Bu yeni geliştirilen işletim sistemi çekirdeğine OSman adı verildi. OSman'ın GeekOS'in temel işlemleri yanında çoklu-programlama, VESA grafik kart sürücü desteği ve PCI Aygıt Sürücü Sistem Tabanı gibi ek fonksiyonları vardır.

OSman bir sürü güzel özelliğe sahip olmasına rağmen çok önemli ağ desteğine sahip değildi. Bu tezin amacı OSman'a bir Ethernet ağ sürücüsü ve TCP/IP yığını ekleyerek ağ üzerinden haberleşebilen uygulamalar geliştirilmesine olanak sağlamaktır. Ağ Ara Yüz Kartı olarak Realtek RTL8139 Ethernet kartı seçilmiş ve işletim sistemi üzerine bir ağ aygıt sürücüsü yazılımı gerçekleştirilmiştir. Gerçekleştirilen ağ aygıt sürücüsü üzerine standartlara uygun, beş parçadan oluşan bir TCP/IP yığını geliştirilmiştir.

Son olarak tüm sistemin çalıştığını göstermek üzere bir istemci-sunucu uygulama programı, cüz oyunu, yazılmıştır. Bu oyun, biri üzerinde OSman diğeri Windows XP çalışan iki bilgisayarın arasındaki iletişimi göstermektedir.

Anahtar Kelimeler: OSman, GeekOS, İşletim Sistemleri, Ağ Ara Yüz Kartı Sürücüsü, PCI Aygıt Sürücüsü, TCP/IP Protokol Yığını

ACKNOWLEDGEMENTS

I would like to thank my advisor Assist. Prof. Dr. Cüneyt AKINLAR for his patience and support during this study. It was my pleasure to work with him during my thesis. I would like to thank to my dear friend Serhan GÜRMERİÇ for his scientific support and inspiration. I would also like to thank my family and my dear Deniz ERİK. Without their incorporeal support, I would not complete this thesis.

Alper Bilge

June, 2008

CONTENTS

ABSTRACT	i
ÖZET.....	ii
ACKNOWLEDGEMENTS.....	iii
CONTENTS.....	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
1. INTRODUCTION	1
1.1 What Is GeekOS?.....	1
1.2 What Is OSman?	2
1.2.1 Multi-programming.....	2
1.2.2 VESA standard.....	3
1.2.3 Enhanced multiple device supported boot sector code	4
1.3 What Is A Network Interface Card and Realtek RTL8139?	4
1.4 What Is A Device Driver?.....	6
1.5 What Is A Protocol Stack?	7
2. OSman PCI DRIVER SYSTEM	9
2.1 PCI Configuration Space.....	9
2.2 OSman Interrupt Handler System	11
2.3 RTL8139 NIC Driver.....	12
2.3.1 PCI Configuration space functions	15

2.3.2	RTL8139 Register descriptions	24
2.3.3	RTL8139 Interrupt handler	36
2.3.4	RTL8139 Transmit operation.....	36
2.3.5	RTL8139 Receive operation	36
3.	DESIGN AND REALIZATION OF TCP/IP STACK	38
3.1	OSI Model.....	38
3.2	TCP/IP Model	39
3.2.1	Architectural principles.....	40
3.2.2	Layers in the TCP/IP model.....	41
3.3	Physical Layer.....	42
3.4	Implementation of the Packet Structures and Functions In Stack.....	43
3.5	Data Link Layer	46
3.6	Implementation of the Data Link Layer.....	46
3.7	Network Layer	47
3.8	Implementation of the Network Layer.....	48
3.9	Transport Layer.....	56
3.10	Implementation of Transport Layer	58
3.11	Application Layer.....	61
3.12	Implementation of Application Layer.....	61
3.12.1	Tic-tac-toe game.....	61
3.12.2	Structure of client/server application	62

4. CONCLUSIONS AND FUTURE WORK.....	64
REFERENCES.....	65

LIST OF TABLES

2.1	PCI configuration space table	15
2.2	Configuration space command register	17
2.3	Configuration space status register	19
2.4	Configuration space IOAR register	20
2.5	Configuration space MEMAR register	20
2.6	Configuration space CISPTr register bits 2-0	21
2.7	Configuration space CISPTr register address space offset values	21
2.8	Configuration space BMAR register	22
2.9	Receive status register	25
2.10	Transmit status register	26
2.11	Command register	27
2.12	Interrupt mask register	28
2.13	Interrupt status register	29
2.14	Transmit configuration register	30
2.15	Receive configuration register	32
3.1	Address resolution protocol packet structure	48
3.2	Internet protocol packet structure	50
3.3	Option bits in IP header	54
3.4	User datagram protocol header	58

LIST OF FIGURES

1.1 GUI supported welcome screen of OSman	3
1.2 PCI device detection sequence of OSman	4
2.1 Standard registers of PCI configuration space	9
2.2 Block diagram of RTL8139 NIC	13
3.1 Network connections over two routers and corresponding stack layers	40
3.2 Packet encapsulation over layers for a UDP packet	41
3.3 A sequence of tic-tac-toe game	61
3.4 UDP packet receiving and printing in OSman	62
3.5 A game sequence in GUI platform of OSman	62

1. INTRODUCTION

A tiny operating system, called GeekOS, which is designed for undergraduate students at the University of Maryland in USA, is implemented with a large number of enhancements as a senior project at Anadolu University in 2006. This newly developed operating system kernel is called OSman, and has the capability to run on real hardware.

While OSman has many features such as multi-programming support, a PCI Device Driver System Base and a simple GUI, it lacks communication support. The goal of this thesis is to add a standards-compliant TCP/IP stack implementation to OSman so that applications running on top of OSman would be able to communicate with the world via a network interface card (NIC). For this purpose, an Ethernet card is used since Ethernet is the most widely used NIC in the networking community. As there are many Ethernet cards on the market, one of the most favorite Ethernet cards, Realtek RTL8139, is chosen in this thesis, and a network device driver for OSman is implemented.

Since a network driver is not enough for networked communication with the Internet, OSman also needs a TCP/IP protocol stack. For this purpose, a standard-compliant, five-layer TCP/IP stack is developed on top of the Ethernet driver.

1.1 What Is GeekOS?

GeekOS is a tiny operating system kernel for x86 PCs. Its main purpose is to serve as a simple but realistic example of an OS kernel running on real hardware.

The goal of GeekOS is to be a tool for learning about operating system kernels. It comes with a set of projects suitable for use in an undergraduate operating systems course, or for self-directed learning. GeekOS has been used in courses at a number of colleges and universities [20]. The following list summarizes GeekOS's features:

- Interrupt Handling
- Heap Memory Allocator
- Time-Sliced Kernel Threads With Static-Priority Scheduling
- Mutexes And Condition Variables For Synchronization Of Kernel Threads
- User Mode With Segmentation-Based Memory Protection And A Simple System Call Interface
- Device Drivers For Keyboard And VGA Text Mode Display

1.2 What Is OSman?

OSman is a GeekOS based project which is started at Anadolu University as a senior project. Although it is a GeekOS based operating system project, it has extended features over GeekOS. The following list summarizes the main differences of OSman over GeekOS:

- Real Multi-Programming System over CPU
- High Resolution VESA Graphic Card Driver
- Virtual x86 Mode Support
- Enhanced Multiple Device Supported Boot Sector Code
- New EDD Based Disk Drive Support
- PCI Driver System Base Implementation
- MFC like Event Table Based GUI System (Dream) (Figure 1.1)
- PCI Device Detection (Figure 1.2)
- True Type Font Support
- New Heap Management System
- Cross Platform *gcc* and *g++* Port for OSman

1.2.1 Multi-programming

Multitasking is a method by which multiple tasks, also known as processes, share common processing resources such as a CPU. In the case of a computer with a

single CPU, only one task is said to be running at any point in time, meaning that the CPU is actively executing instructions for that task. Multitasking solves the problem by scheduling which task may be the one running at any given time, and when another waiting task gets a turn. The act of reassigning a CPU from one task to another one is called a context switch. When context switches occur frequently enough, the illusion of parallelism is achieved. Even on computers with more than one CPU (called multiprocessor machines), multitasking allows many more tasks to be run than there are CPUs [18].



Figure 1.1 GUI supported welcome screen of OSman

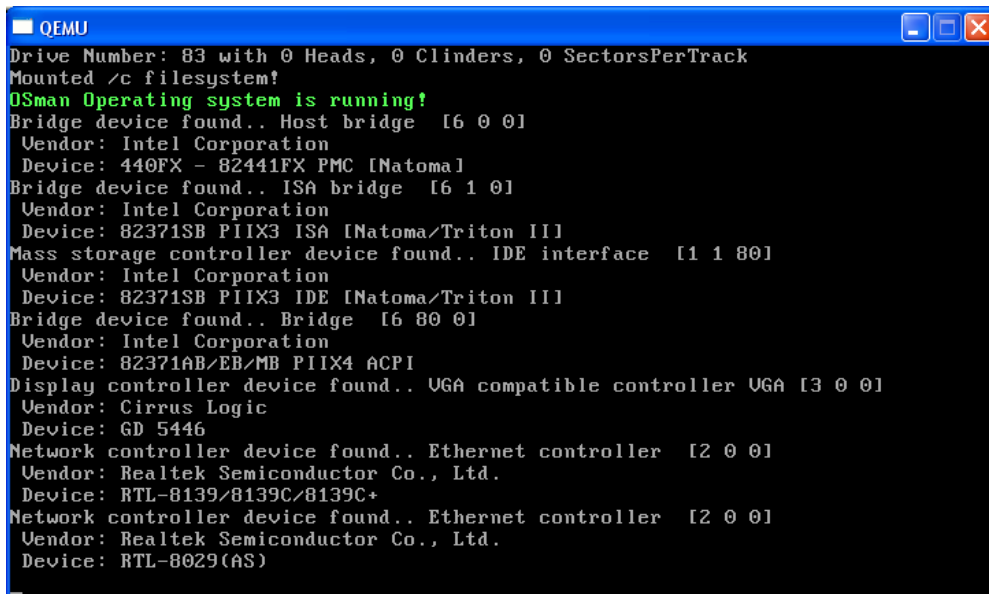
1.2.2 VESA standard

The VBE Standard defines a set of extensions to the VGA ROM BIOS services. These extensions also provide a hardware-independent mechanism to obtain

vendor information and serve as an extensible foundation. The purpose of the VESA VBE is to provide standard software support for the many unique implementations of Super VGA (SVGA) graphics controllers on the PC platform that provides features beyond the original VGA hardware standard [21].

1.2.3 Enhanced multiple device supported boot sector code

With the enhancement of boot sector code, multiple device boot is enabled with OSman. While GeekOS can only be booted by a floppy drive, OSman is able to boot from a floppy drive as well as a hard disk drive, CD, or a USB device with one boot sector code.



```

QEMU
Drive Number: 83 with 0 Heads, 0 Clinders, 0 SectorsPerTrack
Mounted /c filesystem!
OSman Operating system is running!
Bridge device found.. Host bridge [6 0 0]
Vendor: Intel Corporation
Device: 440FX - 82441FX PMC [Natoma]
Bridge device found.. ISA bridge [6 1 0]
Vendor: Intel Corporation
Device: 82371SB P11X3 ISA [Natoma/Triton III]
Mass storage controller device found.. IDE interface [1 1 80]
Vendor: Intel Corporation
Device: 82371SB P11X3 IDE [Natoma/Triton III]
Bridge device found.. Bridge [6 80 0]
Vendor: Intel Corporation
Device: 82371AB/EB/MB P11X4 ACPI
Display controller device found.. UGA compatible controller UGA [3 0 0]
Vendor: Cirrus Logic
Device: GD 5446
Network controller device found.. Ethernet controller [2 0 0]
Vendor: Realtek Semiconductor Co., Ltd.
Device: RTL-8139/8139C/8139C+
Network controller device found.. Ethernet controller [2 0 0]
Vendor: Realtek Semiconductor Co., Ltd.
Device: RTL-8029(AS)

```

Figure 1.2 PCI Device detection sequence of OSman

1.3 What Is A Network Interface Card and Realtek RTL8139?

A network card, network adapter, LAN Adapter or Network Interface Card (NIC) is a piece of computer hardware designed to allow computers to communicate

over a computer network. It is both an OSI layer 1 (physical layer) and layer 2 (data link layer) device, as it provides physical access to a networking medium and provides a low-level addressing system through the use of MAC addresses. It allows users to connect to each other either by using cables or wirelessly [18].

Although other network technologies exist, Ethernet has achieved near-ubiquity since the mid-1990s. Every Ethernet network card has a unique 48-bit serial number called a MAC address, which is stored in ROM carried on the card. Every computer on an Ethernet network must have a card with a unique MAC address. No two cards ever manufactured share the same address. This is accomplished by the Institute of Electrical and Electronics Engineers (IEEE), which is responsible for assigning unique MAC addresses to the vendors of network interface controllers [1].

Whereas network cards used to be expansion cards that plug into a computer bus, the low cost and ubiquity of the Ethernet standard means that the newest computers have a network interface built into the motherboard. These either have Ethernet capabilities integrated into the motherboard chipset, or implemented via a low cost dedicated Ethernet chip, connected through the PCI. A separate network card is not required unless multiple interfaces are needed or some other type of network is used. Newer motherboards may even have dual network (Ethernet) interfaces built-in.

The card implements the electronic circuitry required to communicate using a specific physical layer and data link layer standard such as Ethernet or token ring. This provides a base for a full network protocol stack, allowing communication among small groups of computers on the same LAN and large-scale network communications through routable protocols, such as IP [18].

There are four techniques used to transfer data, the NIC may use one or more of these techniques.

- Polling is where the microprocessor examines the status of the peripheral under program control.
- Programmed I/O is where the microprocessor alerts the designated peripheral by applying its address to the system's address bus.

- Interrupt-driven I/O is where the peripheral alerts the microprocessor that it's ready to transfer data (the technique which is used in this thesis).
- DMA is where the intelligent peripheral assumes control of the system bus to access memory directly. This removes load from the CPU but requires a separate processor on the card.

A network card typically has a twisted pair, BNC, or AUI socket where the network cable is connected, and a few LEDs to inform the user of whether the network is active, and whether or not there is data being transmitted on it. Network Cards are typically available in 10/100/1000 Mbit/s varieties. This means they can support a transfer rate of 10, 100 or 1000 Megabits per second.

Realtek manufactures and sells a wide variety of products throughout the world, and its product lines can be broadly categorized into two subdivisions: Communications Network ICs, and Computer Peripheral and Multimedia ICs. Included among the communications network IC products manufactured and provided by Realtek are: network interface controllers, (both the traditional 10/100M Ethernet controllers and the more advanced Gigabit Ethernet controllers), physical layer controllers, network switch controllers, gateway controllers, wireless LAN ICs, as well as ADSL router controllers. In particular, the RTL8139 series 10/100M Fast Ethernet controllers met their height during the late 90's, and continued to take up a significant and eventually predominant share in the worldwide market in the following years.

Realtek's single-chip fast Ethernet controller, the RTL8139, receives "Best Component" and "Best of Show" awards at Computex-Taipei '97 [18].

1.4 What Is A Device Driver?

A device driver is a computer program allowing higher-level computer programs to interact with a device. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware is connected. When a calling program invokes a routine in the driver, the driver issues

commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating system specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

A device driver simplifies programming by acting as a translator between a device and the applications or operating systems that use it. The higher-level code can be written independently of whatever specific hardware device it may control. Every version of a device, such as a printer, requires its own specialized commands. In contrast, most applications access devices (such as sending a file to a printer or sending packet to network) by using high-level, generic commands. The driver accepts these generic statements and converts them into the low-level commands required by the device [13, 14].

Device drivers can be abstracted into logical and physical layers. Logical layers process data for a class of devices such as Ethernet ports or disk drives. Physical layers communicate with specific device instances. For example, a serial port needs to handle standard communication protocols such as XON/XOFF that are common for all serial port hardware. This would be managed by a serial port logical layer. However, the logical layer needs to communicate with a particular serial port chip. The physical layer addresses these chip specific variations. Conventionally, OS requests go to the logical layer first. In turn, the logical layer calls upon the physical layer to implement OS requests in terms understandable by the hardware. Inversely, when a hardware device needs to respond to the OS, it uses the physical layer to speak through the logical layer [5].

1.5 What Is A Protocol Stack?

A protocol stack is a particular software implementation of a computer networking protocol suite. The terms are often used interchangeably. Strictly speaking, the suite is the definition of the protocols, and the stack is the software implementation of them [10].

Individual protocols within a suite are often designed with a single purpose in mind. This modularization makes design and evaluation easier. Because each protocol module usually communicates with two others, they are commonly imagined as layers in a stack of protocols. The lowest protocol always deals with "low-level", physical interaction of the hardware. Every higher layer adds more features. User applications usually deal only with the topmost layers.

In practical implementation, protocol stacks are often divided into three major sections: media, transport, and applications. A particular operating system or platform will often have two well-defined software interfaces: one between the media and transport layers, and one between the transport layers and applications.

The media-to-transport interface defines how transport protocol software makes use of particular media and hardware types. For example, this interface level would define how TCP/IP transport software would talk to Ethernet hardware [10].

The application-to-transport interface defines how application programs make use of the transport layers. For example, this interface level would define how a web browser program would talk to TCP/IP transport software. Examples of these interfaces include Berkeley sockets and System V streams in the UNIX world, and Winsock in the Microsoft world [2].

2. OSman PCI DRIVER SYSTEM

OSman operating system provides necessary functions to build PCI drivers, such as low level PCI configuration reading and writing functions. OSman has also a PCI device list which enables the detection of PCI devices connected to the system. Therefore it creates an environment to develop drivers for PCI cards such as network interface cards, graphic cards, and sound cards etc.

2.1 PCI Configuration Space

One of the major improvements Peripheral Component Interconnect (PCI) had over other I/O architectures was its configuration mechanism. In addition to the normal memory-mapped and port spaces, each device on the bus has a configuration space. This is 256 bytes that are addressable by knowing the 8-bit PCI bus, 5-bit device, and 3-bit function numbers for the device. This allows up to 256 buses, each with up to 32 devices, each supporting 8 functions. A single PCI expansion card can respond as a device and must implement at least number zero function. The first 64 bytes of configuration space are standardized. Standard registers of PCI Configuration Space is shown Figure 2.1.

In order to allow more parts of configuration space to be standardized without conflicting with existing uses, there is a list of capabilities. Each capability has one byte that describes which capability it is, and one byte to point to the next capability. The number of additional bytes depends on the capability ID.

PCI-X 2.0 and PCI Express introduced an extended configuration space, up to 4096 bytes. The only standardized part of extended configuration space is the first 4 bytes at 0x100 which are the start of an extended capability list. Extended capabilities are very much like normal capabilities except that they can refer to any byte in the extended configuration space (by using 12 bits instead of 8), have a 4-bit version number and a 16-bit capability ID. Extended capability IDs overlap with normal capability IDs, but there is no chance of confusion as they are in separate lists [5].

The Vendor ID and Device ID registers identify the device, and are commonly called the PCI ID. The 16-bit vendor ID is allocated by the PCI SIG. The 16-bit device ID is then assigned by the vendor. There is an ongoing project to collect all known Vendor and Device IDs.

The Subsystem Vendor ID and the Subsystem Device ID further identify the device. The Vendor ID is that of the chip manufacturer, and the Subsystem Vendor ID is that of the card manufacturer. The Subsystem Device ID is assigned by the subsystem vendor, but is assigned from the same number space as the Device ID.

The Command register contains a bitmask of features that can be individually enabled and disabled.

The Status register is used to report which features are supported and whether certain kinds of error have occurred.

The Cache Line Size register must be programmed before the device is told it can use the memory-write-and-invalidate transaction. This should normally match the CPU's cache line size, but the correct setting is system dependent [5, 18].

31		16 15		0	
Device ID		Vendor ID			00h
Status		Command			04h
Class Code			Revision ID		08h
BIST	Header Type	Lat. Timer	Cache Line S.		0Ch
Base Address Registers					10h
					14h
					18h
					1Ch
					20h
					24h
Cardbus CIS Pointer					28h
Subsystem ID		Subsystem Vendor ID			2Ch
Expansion ROM Base Address					30h
Reserved			Cap. Pointer		34h
Reserved					38h
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line		3Ch

Figure 2.1 Standard registers of PCI configuration space [18]

In order to address a device through port space or memory space, system firmware or the OS will program the Base Address Registers (commonly called BARs) by writing configuration commands to the PCI controller. Since all PCI devices are in an inactive state upon system boot, they do not have any addresses assigned to them by which the operating system or device drivers can communicate with them. Either the BIOS or the operating system itself geographically addresses the PCI slots through the PCI controller. Since there is no direct method for the BIOS or OS to determine which PCI slots have devices and which functions they implement, the PCI bus must be enumerated. Bus enumeration is performed by attempting to read the VID/DID for each combination of bus, device, and function. If there is no device that implements the specified function, the bus master performs an abort and returns all 1's in binary (F's in hexadecimal). This is an invalid VID/DID so the BIOS or OS know the specified device does not exist. If a read to function zero of specified bus/device master aborts, then it is assumed that no such device exists on the bus since devices are required to implement function number zero. In this case reads to the remaining functions are not necessary.

When a read to a specified BDF succeeds, the BIOS or OS programs the memory and port addresses into the PCI devices' on-chip memory. These addresses stay valid as long as the system remains turned on. On power off, all these settings are lost and on the next system boot, the configuration procedure is repeated all over again. Since this entire process is fully automated, the computer user is spared the task of configuring any newly added hardware manually by modifying settings of dip switches on the cards themselves. This is how plug and play is implemented [5].

Each non-bridge device can implement up to 6 BARs, each of which can respond to certain areas of port or memory space. A device can have a ROM.

2.2 OSman Interrupt Handler System

An interrupt handler, also known as an interrupt service routine (ISR), is a callback subroutine in an operating system or device driver whose execution is

triggered by the reception of an interrupt. Interrupt handlers have a variety of functions, which vary based on the reason the interrupt was generated and the speed at which the Interrupt Handler completes its task.

An interrupt handler is a low-level counterpart of event handlers. These handlers are initiated by either hardware interrupts or interrupt instructions in software, and are used for servicing hardware devices and transitions between protected modes of operation such as system calls [5, 18].

OSman has a very simple interrupt number and handler connection function. It is really easy to connect an IRQ number with a standard C function by `Install_IRQ(IRQ#, &Foo)` function and control of interrupt by `Enable_IRQ(IRQ#)` and `Disable_IRQ(IRQ#)` functions.

2.3 RTL8139 NIC Driver

The Realtek RTL8139C is a highly integrated and cost-effective single-chip Fast Ethernet controller that provides 32-bit performance, PCI bus master capability, and full compliance with IEEE 802.3u 100Base-T specifications and IEEE 802.3x Full Duplex Flow Control. It also supports Advanced Configuration Power management Interface (ACPI), PCI power management for modern operating systems that are capable of Operating System Directed Power Management (OSPM) to achieve the most efficient power management possible. The RTL8139C is suitable for applications such as CardBus or mobile devices with a built-in network controller. The CIS data can be stored in either a 93C56 EEPROM or expansion ROM. The block diagram of RTL8139 is shown in Figure 2.2.

In addition to the ACPI feature, the RTL8139C also supports remote wake-up (including AMD Magic Packet, LinkChg, and Microsoft wake-up frame) in both ACPI and APM environments. The RTL8139C is capable of performing an internal reset through the application of auxiliary power. When auxiliary power is on and the main power remains off, the RTL8139C is ready and waiting for the Magic Packet or Link Change to wake the system up. Also, the LWAKE pin provides 4 different

output signals including active high, active low, positive pulse, and negative pulse. The versatility of the RTL8139C LWAKE pin provides motherboards with the Wake-On-LAN (WOL) function. The RTL8139C also supports Analog Auto-Power-down, that is, the analog part of the RTL8139C can be shut down temporarily according to user requirements or when the RTL8139C is in a power down state with the wakeup function disabled. In addition, when the analog part is shut down and the IsolateB pin is low (i.e. the main power is off), then both the analog and digital parts stop functioning and power consumption of the RTL8139C will be negligible. The RTL8139C also supports an auxiliary power auto-detect function, and will auto-configure related bits of their own PCI power management registers in PCI configuration space.

The PCI Vital Product Data (VPD) is also supported to provide the information that uniquely identifies hardware (i.e., the RTL8139C LAN card). The information may consist of part number, serial number, and other detailed information.

To provide cost down support, the RTL8139C is capable of using a 25MHz crystal or OSC as its internal clock source.

The RTL8139C keeps network maintenance costs low and eliminates usage barriers. It is the easiest way to upgrade a network from 10 to 100Mbps. It also supports full-duplex operation, making 200Mbps bandwidth possible at no additional cost. To improve compatibility with other brands' products, the RTL8139C is also capable of receiving packets with InterFrameGap no less than 40 Bit-Time. The RTL8139C is highly integrated and requires no "glue" logic or external memory. It includes an interface for a boot ROM and can be used in diskless workstations, providing maximum network security and ease of management [16, 17].

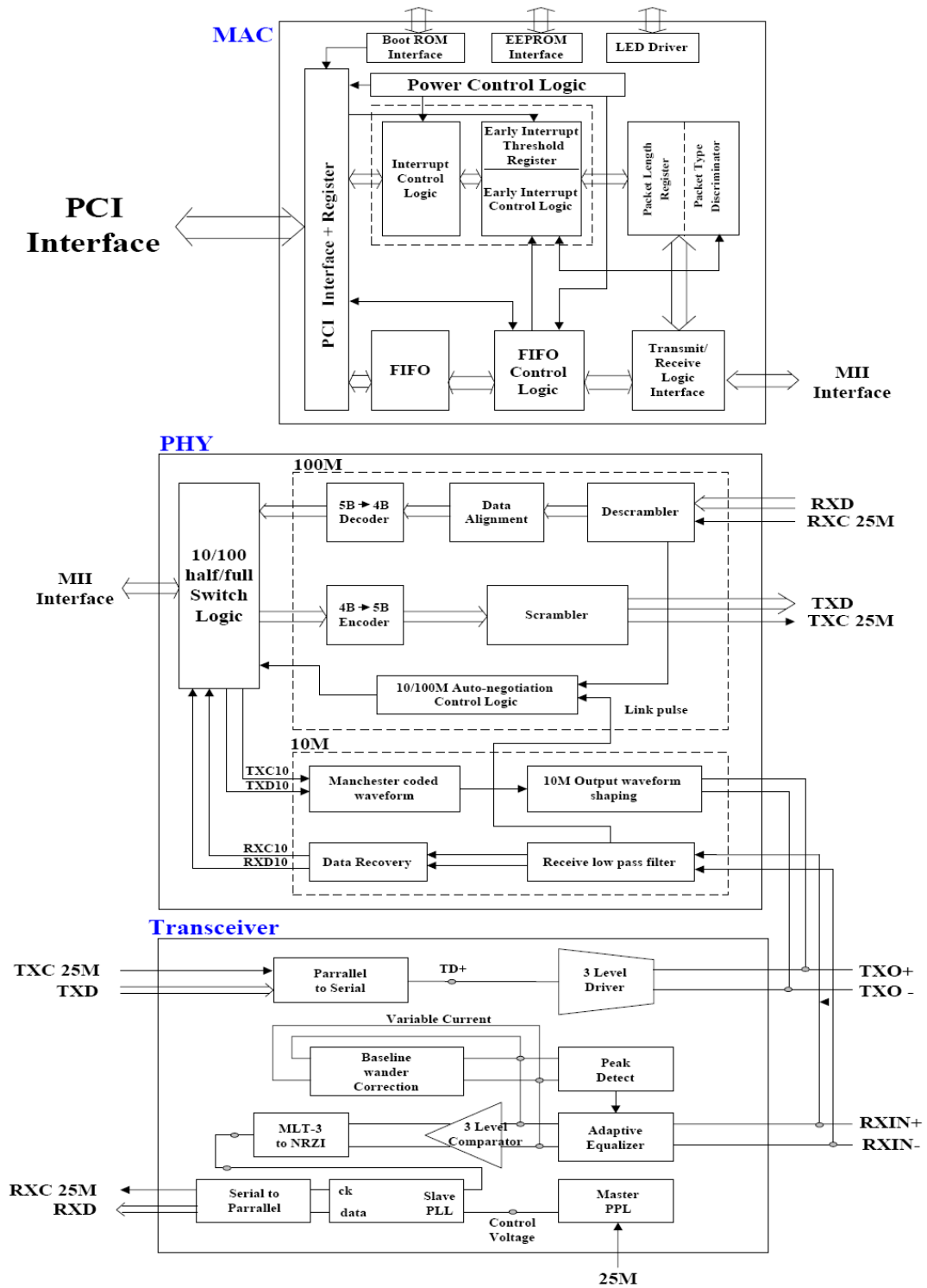


Figure 2.2 Block Diagram of RTL8139 NIC [16]

2.3.1 PCI Configuration space functions

The PCI configuration space is intended for configuration, initialization, and catastrophic error handling functions. The functions of RTL8139C's configuration space and RTL8139 registers are described below and shown in Table 2.1-15.

VID: Vendor ID. This field will be set to a value corresponding to PCI Vendor ID in the external EEPROM. If there is no EEPROM, this field will default to a value of 10ECh which is Realtek Semiconductor's PCI Vendor ID.

DID: Device ID. This field will be set to a value corresponding to PCI Device ID in the external EEPROM. If there is no EEPROM, this field will default to a value of 8129h.

Command: The command register is a 16-bit register used to provide coarse control over a device's ability to generate and respond to PCI cycles. Command Register's fields are explained in detail in Table 2.2.

Status: The status register is a 16-bit register used to record status information for PCI bus related events. Reads to this register behave normally. Writes are slightly different in that bits can be reset, but not set.

RID: Revision ID Register. The Revision ID register is an 8-bit register that specifies the RTL8139C controller revision number.

PIFR: Programming Interface Register. The programming interface register is an 8-bit register that identifies the programming interface of the RTL8139C controller. Because the PCI version 2.1 specification does not define any specific value for network devices, PIFR = 00h.

SCR: Sub-Class Register. The Sub-class register is an 8-bit register that identifies the function of the RTL8139C. SCR = 00h indicates that the RTL8139C is an Ethernet controller.

BCR: Base-Class Register. The Base-class register is an 8-bit register that broadly classifies the function of the RTL8139C. BCR = 02h indicates that the RTL8139C is a network controller.

CLS: Cache Line Size. Reads will return a 0, writes are ignored.

Table 2.1 PCI configuration space table [16]

No.	Name	Type	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00h	VID	R	VID7	VID6	VID5	VID4	VID3	VID2	VID1	VID0
01h		R	VID15	VID14	VID13	VID12	VID11	VID10	VID9	VID8
02h	DID	R	DID7	DID6	DID5	DID4	DID3	DID2	DID1	DID0
03h		R	DID15	DID14	DID13	DID12	DID11	DID10	DID9	DID8
04h	Command	R	0	PERRSP	0	0	-	BMEN	MEMEN	IOEN
		W	-	PERRSP	-	-	-	BMEN	MEMEN	IOEN
05h		R	0	0	0	0	0	FBT BEN	SERREN	
W		-	-	-	-	-	-	-	SERREN	
06h	Status	R	FBBC	0	0	NewCap	0	0	0	0
07h		R	DPERR	SSERR	RMABT	RTABT	STABT	DST1	DST0	DPD
		W	DPERR	SSERR	RMABT	RTABT	STABT	-	-	DPD
08h	Revision ID	R	0	0	0	0	0	0	0	0
09h	PIFR	R	0	0	0	0	0	0	0	0
0Ah	SCR	R	0	0	0	0	0	0	0	0
0Bh	BCR	R	0	0	0	0	0	0	1	0
0Ch	CLS	R	0	0	0	0	0	0	0	0
0Dh	LTR	R	LTR7	LTR6	LTR5	LTR4	LTR3	LTR2	LTR1	LTR0
		W	LTR7	LTR6	LTR5	LTR4	LTR3	LTR2	LTR1	LTR0
0Eh	HTR	R	0	0	0	0	0	0	0	0
0Fh	BIST	R	0	0	0	0	0	0	0	0
10h	IOAR	R	0	0	0	0	0	0	0	IOIN
		W	-	-	-	-	-	-	-	-
11h		R/W	IOAR15	IOAR14	IOAR13	IOAR12	IOAR11	IOAR10	IOAR9	IOAR8
12h		R/W	IOAR23	IOAR22	IOAR21	IOAR20	IOAR19	IOAR18	IOAR17	IOAR16
13h		R/W	IOAR31	IOAR30	IOAR29	IOAR28	IOAR27	IOAR26	IOAR25	IOAR24
14h	MEMAR	R	0	0	0	0	0	0	0	MEMIN
		W	-	-	-	-	-	-	-	-
15h		R/W	MEM15	MEM14	MEM13	MEM12	MEM11	MEM10	MEM9	MEM8
16h		R/W	MEM23	MEM22	MEM21	MEM20	MEM19	MEM18	MEM17	MEM16
17h		R/W	MEM31	MEM30	MEM29	MEM28	MEM27	MEM26	MEM25	MEM24

Table 2.1 (Continued) PCI configuration space table

18h-27h	RESERVED									
28h-2Bh	CISPtr	Cardbus CIS Pointer								
2Ch	SVID	R	SVID7	SVID6	SVID5	SVID4	SVID3	SVID2	SVID1	SVID0
2Dh		R	SVID15	SVID14	SVID13	SVID12	SVID11	SVID10	SVID9	SVID8
2Eh	SMID	R	SMID7	SMID6	SMID5	SMID4	SMID3	SMID2	SMID1	SMID0
2Fh		R	SMID15	SMID14	SMID13	SMID12	SMID11	SMID10	SMID9	SMID8
30h	BMAR	R	0	0	0	0	0	0	0	BROMEN
		W	-	-	-	-	-	-	-	BROMEN
31h	BMAR	R	BMAR15	BMAR14	BMAR13	BMAR12	BMAR11	0	0	0
		W	BMAR15	BMAR14	BMAR13	BMAR12	BMAR11			
32h	BMAR	R/W	BMAR23	BMAR22	BMAR21	BMAR20	BMAR19	BMAR18	BMAR17	BMAR16
33h	BMAR	R/W	BMAR31	BMAR30	BMAR29	BMAR28	BMAR27	BMAR26	BMAR25	BMAR24
34h	Cap_Ptr	R	0	1	0	1	0	0	0	0
35h-3Bh	RESERVED									
3Ch	ILR	R/W	ILR7	ILR6	ILR5	ILR4	ILR3	ILR2	ILR1	ILR0
3Dh	IPR	R	0	0	0	0	0	0	0	1
3Eh	MNGNT	R	0	0	1	0	0	0	0	0
3Fh	MXLAT	R	0	0	1	0	0	0	0	0
40h-4Fh	RESERVED									
50h	PMID	R	0	0	0	0	0	0	0	1
51h	NextPtr	R	0	0	0	0	0	0	0	0
52h	PMC	R	Aux_I_b1	Aux_I_b0	DSI	Rsr	PMECLK	Version		
53h		R	PME_D3	PME_D3	PME_D2	PME_D1	PME_D0	D2	D1	Aux_I_b2
54h	PMCSR	R	0	0	0	0	0	0	Power State	
		W	-	-	-	-	-	-	Power State	
55h	PMCSR	R	PME_Status	-	-	-	-	-	-	PME_En
		W	PME_Status	-	-	-	-	-	-	PME_En
56h-5Fh	RESERVED									
60h	VPDID	R	0	0	0	0	0	0	1	1

Table 2.1 (Continued) PCI configuration space table

61h	NextPtr	R	0	0	0	0	0	0	0	0
62h	FlagVPD	R/W	VPD7	VPD6	VPD5	VPD4	VPD3	VPD2	VPD1	VPD0
63h	Address	R/W	Flag	VPD14	VPD13	VPD12	VPD11	VPD10	VPD9	VPD8
64h	VDP	R/W	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0
65h	Data	R/W	Data15	Data14	Data13	Data12	Data11	Data10	Data9	Data8
66h		R/W	Data23	Data22	Data21	Data20	Data19	Data18	Data17	Data16
67h		R/W	Data31	Data30	Data29	Data28	Data27	Data26	Data25	Data24
68h- FFh		RESERVED								

Table 2.2 Configuration space command register [16]

Bit	Symbol	Description
15-10	-	Reserved
9	FBTBEN	Fast Back-To-Back Enable: Config3<FBtBEn>=0:Read as 0. Write operation has no effect. The RTL8139C will not generate Fast Back-to-back cycles. When Config3<FbtBEn>=1, This read/write bit controls whether or not a master can do fast back-to-back transactions to different devices. Initialization software will set the bit if all targets are fast back-to-back capable. A value of 1 means the master is allowed to generate fast back-to-back transaction to different agents. A value of 0 means fast back-to-back transactions are only allowed to the same agent. This bit's state after RST# is 0.
8	SERREN	System Error Enable: When set to 1, the RTL8139C asserts the SERRB pin when it detects a parity error on the address phase (AD<31:0> and CBEB<3:0>).
7	ADSTEP	Address/Data Stepping: Read as 0, write operation has no effect. The RTL8139C never performs address/data stepping.

Table 2.2 (Continued) Configuration space command register

6	PERRSP	Parity Error Response: When set to 1, RTL8139C will assert the PERRB pin on the detection of a data parity error when acting as the target, and will sample the PERRB pin as the master. When set to 0, any detected parity error is ignored and the RTL8139C continues normal operation. Parity checking is disabled after hardware reset (RSTB).
5	VGASNOOP	VGA palette SNOOP: Read as 0, write operation has no effect.
4	MWIEN	Memory Write and Invalidate Cycle Enable: Read as 0, write operation has no effect.
3	SCYCEN	Special Cycle Enable: Read as 0, write operation has no effect. The RTL8139C ignores all special cycle operation.
2	BMEN	Bus Master Enable: When set to 1, the RTL8139C is capable of acting as a bus master. When set to 0, it is prohibited from acting as a PCI bus master. For the normal operation, this bit must be set by the system BIOS.
1	MEMEN	Memory Space Access: When set to 1, the RTL8139C responds to memory space accesses. When set to 0, the RTL8139C ignores memory space accesses.
0	BOEN	I/O Space Access: When set to 1, the RTL8139C responds to IO space access. When set to 0, the RTL8139C ignores I/O space accesses.

LTR: Latency Timer Register. Specifies, in units of PCI bus clocks, the value of the latency timer of the RTL8139C. When the RTL8139C asserts FRAMEB, it enables its latency timer to count. If the RTL8139C deasserts FRAMEB prior to count expiration, the content of the latency timer is ignored. Otherwise, after the count expires, the RTL8139C initiates transaction termination as soon as its GNTB is deasserted. Software is able to read or write, and the default value is 00H.

HTR: Header Type Register. Reads will return a 0, writes are ignored.

BIST: Built-in Self Test. Reads will return a 0, writes are ignored.

Table 2.3 Configuration space status register [16]

Bit	Symbol	Description
15	DPERR	Detected Parity Error: When set indicates that the RTL8139C detected a parity error, even if parity error handling is disabled in command register PERRSP bit.
14	SSERR	Signaled System Error: When set indicates that the RTL8139C asserted the system error pin, SERRB. Writing a 1 clears this bit to 0.
13	RMABT	Received Master Abort: When set indicates that the RTL8139C terminated a master transaction with master abort. Writing a 1 clears this bit to 0.
12	RTABT	Received Target Abort: When set indicates that the RTL8139C master transaction was terminated due to a target abort. Writing a 1 clears this bit to 0.
11	STABT	Signaled Target Abort: Set to 1 whenever the RTL8139C terminates a transaction with target abort. Writing a 1 clears this bit to 0.
10-9	DST1-0	Device Select Timing: These bits encode the timing of DEVSELB. They are set to 01b (medium), indicating the RTL8139C will assert DEVSELB two clocks after FRAMEB is asserted.
8	DPD	Data Parity error Detected: This bit sets when the following conditions are met: <ul style="list-style-type: none"> • The RTL8139C asserts parity error(PERRB pin) or it senses the assertion of PERRB pin by another device. • The RTL8139C operates as a bus master for the operation that caused the error. • The Command register PERRSP bit is set. Writing a 1 clears this bit to 0.
7	FBBC	Fast Back-To-Back Capable: Config3<FbtBEn>=0, Read as 0, write operation has no effect. Config3<FbtBEn>=1, Read as 1.
6	UDF	User Definable Features Supported: Read as 0, write operation has no effect. The RTL8139C does not support UDF.
5	66MHz	66 MHz Capable: Read as 0, write operation has no effect. The RTL8139C has no 66MHz capability.
4	NewCap	New Capability: Config3<PMEn>=0, Read as 0, write operation has no effect. Config3<PMEn>=1, Read as 1.
0-3	-	Reserved

IOAR: This register specifies the BASE IO address which is required to build an address map during configuration. It also specifies the number of bytes required as well as an indication that it can be mapped into IO space.

Table 2.4 Configuration space IOAR register [16]

Bit	Symbol	Description
31-8	IOAR31-8	BASE IO Address: This is set by software to the Base IO address for the operational register map.
7-2	IOSIZE	Size Indication: Read back as 0. This allows the PCI bridge to determine that the RTL8139C requires 256 bytes of IO space.
1	-	Reserved
0	IOIN	IO Space Indicator: Read only. Set to 1 by the RTL8139C to indicate that it is capable of being mapped into IO space.

MEMAR: This register specifies the base memory address for memory accesses to the RTL8139C operational registers. This register must be initialized prior to accessing any of the RTL8139C's register with memory access.

Table 2.5 Configuration space MEMAR register [16]

Bit	Symbol	Description
31-8	MEM31-8	Base Memory Address: This is set by software to the base address for the operational register map.
7-4	MEMSIZE	Memory Size: These bits return 0, which indicates that The RTL8139C requires 256 bytes of Memory Space.
3	MEMPF	Memory Prefetchable: Read only. Set to 0 by The RTL8139C.
2-1	MEMLOC	Memory Location Select: Read only. Set to 0 by The RTL8139C. This indicates that the base register is 32-bit wide and can be placed anywhere in the 32-bit memory space.
0	MEMIN	Memory Space Indicator: Read only. Set to 0 by The RTL8139C to indicate that it is capable of being mapped into memory space.

CISPtr: CardBus CIS Pointer. This field is valid only when CardB_En (bit3, Config3) = 1. The value of this register is auto-loaded from 93C46 or 93C56 (from offset 30h-31h).

- Bit 2-0: Address Space Indicator

Table 2.6 Configuration space CISPtr register bits 2-0 [16]

Bit 2-0	Meaning
0	Not supported. (CIS begins in device-dependent configuration space.)
1-6	The CIS begins in the memory address governed by one of the six Base Address Registers. Ex., if the value is 2, then the CIS begins in the memory address space governed by Base Address Register 2.
7	The CIS begins in the Expansion ROM space.

- Bit 27-3: Address Space Offset
- Bit31-28: ROM Image Number

Table 2.7 Configuration space CISPtr register address space offset values [16]

Bit 2-0	Space Type	Address Space Offset Values
0	Configuration Space	Not Supported.
X; 1≤X≤6	Memory Space	0h≤value≤FFFF FFF8h. This is the offset into the memory address space governed by Base Address Register X. Adding this value to the value in the Base Address Register gives the location of the start of the CIS. For RTL8139C(L), the value is 100h.
7	Expansion ROM	0 ≤ image number ≤ Fh, 0h ≤ value ≤ 0FFF FFF8h. This is the offset into the expansion ROM address space governed by the Expansion ROM Base Register. The image number is in the uppermost nibble of the CISPtr register. The value consists of the remaining bytes. For RTL8139C(L), the image number is 0h.

This read-only register points to where the CIS begins, in one of the following spaces:

- Memory space: The CIS may be in any of the memory spaces from offset 100h and up after being auto-loaded from 93C56. The CIS is stored in 93C56 EEPROM physically from offset 80h-FFh.
- Expansion ROM space: The CIS is stored in expansion ROM physically within the 128KB max.

SVID: Subsystem Vendor ID. This field will be set to a value corresponding to the PCI Subsystem Vendor ID in the external EEPROM. If there is no EEPROM, this field will default to a value of 11ECh which is Realtek Semiconductor's PCI Subsystem Vendor ID.

BMAR: This register specifies the base memory address for memory accesses to the RTL8139C operational registers. This register must be initialized prior to accessing any of the RTL8139C's registers with memory access.

Table 2.8 Configuration space BMAR register [16]

Bit	Symbol	Description																																				
31-18	BMAR31-18	Boot ROM Base Address																																				
17-11	ROMSIZE	<p>These bits indicate how many Boot ROM spaces to be supported. The Relationship between Config 0 <BS2:0> and BMAR17-11 is the following:</p> <table border="1"> <thead> <tr> <th><u>BS2</u></th> <th><u>BS1</u></th> <th><u>BS0</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>No Boot ROM</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>8K Boot ROM</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>16K Boot ROM</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>32K Boot ROM</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>64K Boot ROM</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>128K Boot ROM</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>unused</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>unused</td> </tr> </tbody> </table>	<u>BS2</u>	<u>BS1</u>	<u>BS0</u>	<u>Description</u>	0	0	0	No Boot ROM	0	0	1	8K Boot ROM	0	1	0	16K Boot ROM	0	1	1	32K Boot ROM	1	0	0	64K Boot ROM	1	0	1	128K Boot ROM	1	1	0	unused	1	1	1	unused
<u>BS2</u>	<u>BS1</u>	<u>BS0</u>	<u>Description</u>																																			
0	0	0	No Boot ROM																																			
0	0	1	8K Boot ROM																																			
0	1	0	16K Boot ROM																																			
0	1	1	32K Boot ROM																																			
1	0	0	64K Boot ROM																																			
1	0	1	128K Boot ROM																																			
1	1	0	unused																																			
1	1	1	unused																																			
10-1	-	Reserved (read back 0)																																				
0	BROMEN	Boot ROM Enable: This is used by the PCI BIOS to enable accesses to Boot ROM.																																				

SMID: Subsystem ID. This field will be set to a value corresponding to the PCI Subsystem ID in the external EEPROM. If there is no EEPROM, this field will default to a value of 8129h.

ILR: Interrupt Line Register. The Interrupt Line Register is an 8-bit register used to communicate with the routing of the interrupt. It is written by the POST software to set interrupt line for the RTL8139C.

IPR: Interrupt Pin Register. The Interrupt Pin register is an 8-bit register indicating the interrupt pin used by the RTL8139C. The RTL8139C uses INTA interrupt pin. Read only. IPR = 01H.

MNGNT: Minimum Grant Timer: Read only Specifies how long a burst period the RTL8139C needs at 33 MHz clock rate in units of 1/4 microsecond. This field will be set to a value from the external EEPROM. If there is no EEPROM, this field will default to a value of 20h.

MXLAT: Maximum Latency Timer: Read only Specifies how often the RTL8139C needs to gain access to the PCI bus in units of 1/4 microseconds. This field will be set to a value from the external EEPROM. If there is no EEPROM, this field will default to a value of 20h [16].

2.3.2 RTL8139 Register descriptions

There are plenty of registers in RTL8139 to handle different operations. The most important ones of them are configuration, command, and status registers to implement the driver.

Receive Status Register holds bits to handle situations when a packet is received from the network. Those situations can be multicast packet receiving, Destination MAC match, broadcast packet receiving, small and long packet receiving, CRC error, frame alignment error, and successful receive. Detailed explanation of the meanings of bits in this register is given in Table 2.8.

Transmit Status Register holds bits to handle situations when a packet is transmitted through the network. Those situations can be carrier sense lost,

transmission abort, out of window collision, cd heart beat, number of collision count, early transmit threshold, successful transmission, and transmit FIFO underrun. Detailed explanation of the meanings of bits in this register is given in Table 2.9.

Command register is used for issuing commands to the RTL8139C. These commands are issued by setting the corresponding bits for the function. Reset bit sets all default values to transmit and receive configuration registers when set to 1. Also it enables or disables transmit and receive functions of RTL8139 by setting or clearing the related bit in this register. Detailed explanation of the meanings of bits in this register is given in Table 2.10.

Table 2.9 Receive status register [16]

Bit	R/W	Symbol	Description
15	R	MAR	Multicast Address Received: This bit set to 1 indicates that a multicast packet is received.
14	R	PAM	Physical Address Matched: This bit set to 1 indicates that the destination address of this packet matches the value written in ID registers.
13	R	BAR	Broadcast Address Received: This bit set to 1 indicates that a broadcast packet is received. BAR, MAR bit will not be set simultaneously.
12-6	-	-	Reserved
5	R	ISE	Invalid Symbol Error: (100BASE-TX only) This bit set to 1 indicates that an invalid symbol was encountered during the reception of this packet.
4	R	RUNT	Runt Packet Received: This bit set to 1 indicates that the received packet length is smaller than 64 bytes (i.e. media header + data + CRC < 64 bytes)
3	R	LONG	Long Packet: This bit set to 1 indicates that the size of the received packet exceeds 4k bytes.
2	R	CRC	CRC Error: When set, indicates that a CRC error occurred on the received packet.
1	R	FAE	Frame Alignment Error: When set, indicates that a frame alignment error occurred on this received packet.
0	R	ROK	Receive OK: When set, indicates that a good packet is received.

Table 2.10 Transmit status register [16]

Bit	R/W	Symbol	Description
31	R	CRS	Carrier Sense Lost: This bit is set to 1 when the carrier is lost during transmission of a packet.
30	R	TABT	Transmit Abort: This bit is set to 1 if the transmission of a packet was aborted. This bit is read only, writing to this bit is not affected.
29	R	OWC	Out of Window Collision: This bit is set to 1 if The RTL8139C encountered an "out of window" collision during the transmission of a packet.
28	R	CDH	CD Heart Beat: The same as RTL8139(A/B). This bit is cleared in the 100 Mbps mode.
27-24	R	NCC3-0	Number of Collision Count: Indicates the number of collisions encountered during the transmission of a packet.
23-22	-	-	Reserved
21-16	R/W	ERTXTH 5-0	Early Tx Threshold: Specifies the threshold level in the Tx FIFO to begin the transmission. When the byte count of the data in the Tx FIFO reaches this level, (or the FIFO contains at least one complete packet) The RTL8139C will transmit this packet. 000000 = 8 bytes These fields count from 000001 to 111111 in unit of 32 bytes. This threshold must be avoided from exceeding 2K byte.
15	R	TOK	Transmit OK: Set to 1 indicates that the transmission of a packet was completed successfully and no transmit underrun occurs.
14	R	TUN	Transmit FIFO Underrun: Set to 1 if the Tx FIFO was exhausted during the transmission of a packet. The RTL8139C can re-transfer data if the Tx FIFO underruns and can also transmit the packet to the wire successfully even though the Tx FIFO underruns. That is, when TSD<TUN>=1, TSD<TOK>=0 and ISR<TOK>=1 (or ISR<TER>=1).
13	R/W	OWN	OWN: The RTL8139C sets this bit to 1 when the Tx DMA operation of this descriptor was completed. The driver must set this bit to 0 when the Transmit Byte Count (bit0-12) is written. The default value is 1.
12-0	R/W	SIZE	Descriptor Size: The total size in bytes of the data in this descriptor. If the packet length is more than 1792 byte (0700h), the Tx queue will be invalid.

Table 2.11 Command register [16]

Bit	R/W	Symbol	Description
7-5	-	-	Reserved
4	R/W	RST	Reset: Setting to 1 forces The RTL8139C to a software reset state which disables the transmitter and receiver, reinitializes the FIFOs, resets the system buffer pointer to the initial value (Tx buffer is at TSAD0, Rx buffer is empty). The values of IDR0-5 and MAR0-7 and PCI configuration space will have no changes. This bit is 1 during the reset operation, and is cleared to 0 by The RTL8139C when the reset operation is complete.
3	R/W	RE	Receiver Enable: When set to 1, and the receive state machine is idle, the receive machine becomes active. This bit will read back as a 1 whenever the receive state machine is active. After initial power-up, software must insure that the receiver has completely reset before setting this bit.
2	R/W	TE	Transmitter Enable: When set to 1, and the transmit state machine is idle, then the transmit state machine becomes active. This bit will read back as a 1 whenever the transmit state machine is active. After initial power-up, software must insure that the transmitter has completely reset before setting this bit.
1	-	-	Reserved
0	R	BUFE	Buffer Empty: The Rx buffer is empty; There is no packet stored in the Rx buffer ring.

Interrupt mask register masks the interrupts that can be generated from the ISR. Writing a “1” to the bit enables the corresponding interrupt. During a hardware reset, all mask bits are cleared. Setting a mask bit allows the corresponding bit in the ISR to cause an interrupt. ISR bits are always set to 1, however, if the condition is

present, regardless of the state of the corresponding mask bit. Detailed explanation of the meanings of bits in this register is given in Table 2.11.

Table 2.12 Interrupt mask register [16]

Bit	R/W	Symbol	Description
15	R/W	SERR	System Error Interrupt: 1 => Enable, 0 => Disable.
14	R/W	TimeOut	Time Out Interrupt: 1 => Enable, 0 => Disable.
13	R/W	LenChg	Cable Length Change Interrupt: 1 => Enable, 0 => Disable.
12-7	-	-	Reserved
6	R/W	FOVW	Rx FIFO Overflow Interrupt: 1 => Enable, 0 => Disable.
5	R/W	PUN LinkChg	Packet Underrun/Link Change Interrupt: 1 => Enable, 0 => Disable.
4	R/W	RXOVW	Rx Buffer Overflow Interrupt: 1 => Enable, 0 => Disable.
3	R/W	TER	Transmit Error Interrupt: 1 => Enable, 0 => Disable.
2	R/W	TOK	Transmit OK Interrupt: 1 => Enable, 0 => Disable.
1	R/W	RER	Receive Error Interrupt: 1 => Enable, 0 => Disable.
0	R/W	ROK	Receive OK Interrupt: 1 => Enable, 0 => Disable.

Interrupt Status Register indicates the source of an interrupt when the INTA pin goes active. Enabling the corresponding bits in the Interrupt Mask Register (IMR) allows bits in this register to produce an interrupt. When an interrupt is active, one of more bits in this register are set to a “1”. The interrupt Status Register reflects all current pending interrupts, regardless of the state of the corresponding mask bit in the IMR. Reading the ISR clears all interrupts. Writing to the ISR has no effect. Detailed explanation of the meanings of bits in this register is given in Table 2.12.

Table 2.13 Interrupt status register [16]

Bit	R/W	Symbol	Description
15	R/W	SERR	System Error: Set to 1 when The RTL8139C signals a system error on the PCI bus.
14	R/W	TimeOut	Time Out: Set to 1 when the TCTR register reaches to the value of the TimerInt register.
13	R/W	LenChg	Cable Length Change: Cable length is changed after Receiver is enabled.
12-7	-	-	Reserved
6	R/W	FOVW	Rx FIFO Overflow: Set when an overflow occurs on the Rx status FIFO.
5	R/W	PUN LinkChg	Packet Underrun/Link Change: Set to 1 when CAPR is written but Rx buffer is empty, or when link status is changed.
4	R/W	RXOVW	Rx Buffer Overflow: Set when receive (Rx) buffer ring storage resources have been exhausted.
3	R/W	TER	Transmit (Tx) Error: Indicates that a packet transmission was aborted, due to excessive collisions, according to the TXRR's setting
2	R/W	TOK	Transmit (Tx) OK: Indicates that a packet transmission is completed successfully.
1	R/W	RER	Receive (Rx) Error: Indicates that a packet has either CRC error or frame alignment error (FAE). The collided frame will not be recognized as CRC error if the length of this frame is shorter than 16 byte.
0	R/W	ROK	Receive (Rx) OK: In normal mode, indicates the successful completion of a packet reception. In early mode, indicates that the Rx byte count of the arriving packet exceeds the early Rx threshold.

Transmit Configuration Register defines the transmit configuration for the RTL8139C. It controls such functions as Loopback, Heartbeat, Auto Transmit Padding, programmable Interframe Gap, Fill and Drain Thresholds, and maximum DMA burst size. Detailed explanation of the meanings of bits in this register is given in Table 2.13.

Table 2.14 Transmit configuration register [16]

Bit	R/W	Symbol	Description				
31	-	-	Reserved				
30-26	R	HWVERID	Hardware Version ID				
25-24	R/W	IFG1,0	<p>Interframe Gap Time: This field allows adjustment of the interframe gap time below the standards of 9.6 us for 10Mbps, 960 ns for 100Mbps. The time can be programmed from 9.6 us to 8.4 us (10Mbps) and 960ns to 840ns (100Mbps). Note that any value other than (1, 1) will violate the IEEE 802.3 standard.</p> <p>The formula for the inter frame gap is:</p> <table style="margin-left: 40px;"> <tr> <td>10 Mbps</td> <td>8.4us + 0.4(IFG(1:0)) us</td> </tr> <tr> <td>100 Mbps</td> <td>840ns + 40(IFG(1:0)) ns</td> </tr> </table>	10 Mbps	8.4us + 0.4(IFG(1:0)) us	100 Mbps	840ns + 40(IFG(1:0)) ns
10 Mbps	8.4us + 0.4(IFG(1:0)) us						
100 Mbps	840ns + 40(IFG(1:0)) ns						
23	R	8139A-G	RTL8139A rev.G ID = 1. For others, this bit is 0.				
22-19	-	-	Reserved				
18-17	R/W	LBK1 LBK0	<p>Loopback test: There will be no packet on the TX+/- lines under the Loopback test condition. The loopback function must be independent of the link state.</p> <table style="margin-left: 40px;"> <tr> <td>00: normal operation</td> <td>01: Reserved</td> </tr> <tr> <td>10: Reserved</td> <td>11: Loopback mode</td> </tr> </table>	00: normal operation	01: Reserved	10: Reserved	11: Loopback mode
00: normal operation	01: Reserved						
10: Reserved	11: Loopback mode						
16	R/W	CRC	<p>Append CRC:</p> <p>0: A CRC is appended at the end of a packet</p> <p>1: No CRC appended at the end of a packet</p>				
15-11	-	-	Reserved				

Table 2.14 (Continued) Transmit configuration register

10-8	R/W	MXDMA 2, 1, 0	Max DMA Burst Size per Tx DMA Burst: This field sets the maximum size of transmit DMA data bursts according to the following table: 000 = 16 bytes 001 = 32 bytes 010 = 64 bytes 011 = 128 bytes 100 = 256 bytes 101 = 512 bytes 110 = 1024 bytes 111 = 2048 bytes
7-4	R/W	TXERR	Tx Retry Count: These are used to specify additional transmission retries in multiples of 16 (IEEE 802.3 CSMA/CD retry count). If the TXRR is set to 0, the transmitter will re-transmit 16 times before aborting due to excessive collisions. If the TXRR is set to a value greater than 0, the transmitter will re-transmit a number of times equal to the following formula before aborting: Total retries = 16 + (TXRR * 16) The TER bit in the ISR register or transmit descriptor will be set when the transmission fails and reaches to this specified retry count.
3-1	-	-	Reserved
0	W	CLRABT	Clear Abort: Setting this bit to 1 cause The RTL8139C to retransmit the packet at the last transmitted descriptor when this transmission was aborted. Setting this bit is only permitted in the transmit abort state.

Receive Configuration Register is used to set the receive configuration for the RTL8139C. Receive properties such as accepting error packets, runt packets, setting the receive drain threshold etc. are controlled within this register. Detailed explanation of the meanings of bits in this register is given in Table 2.14.

Table 2.15 Receive configuration register [16]

Bit	R/W	Symbol	Description																
31-28	-	-	Reserved																
27-24	R/W	ERTH 3, 2, 1, 0	<p>Early Rx threshold bits: These bits are used to select the Rx threshold multiplier of the whole packet that has been transferred to the system buffer in early mode when the frame protocol is under The RTL8139C's definition.</p> <table> <tr> <td>0000 = no early rx threshold</td> <td>0001 = 1/16</td> </tr> <tr> <td>0010 = 2/16</td> <td>0011 = 3/16</td> </tr> <tr> <td>0100 = 4/16</td> <td>0101 = 5/16</td> </tr> <tr> <td>0110 = 6/16</td> <td>0111 = 7/16</td> </tr> <tr> <td>1000 = 8/16</td> <td>1001 = 9/16</td> </tr> <tr> <td>1010 = 10/16</td> <td>1011 = 11/16</td> </tr> <tr> <td>1100 = 12/16</td> <td>1101 = 13/16</td> </tr> <tr> <td>1110 = 14/16</td> <td>1111 = 15/16</td> </tr> </table>	0000 = no early rx threshold	0001 = 1/16	0010 = 2/16	0011 = 3/16	0100 = 4/16	0101 = 5/16	0110 = 6/16	0111 = 7/16	1000 = 8/16	1001 = 9/16	1010 = 10/16	1011 = 11/16	1100 = 12/16	1101 = 13/16	1110 = 14/16	1111 = 15/16
0000 = no early rx threshold	0001 = 1/16																		
0010 = 2/16	0011 = 3/16																		
0100 = 4/16	0101 = 5/16																		
0110 = 6/16	0111 = 7/16																		
1000 = 8/16	1001 = 9/16																		
1010 = 10/16	1011 = 11/16																		
1100 = 12/16	1101 = 13/16																		
1110 = 14/16	1111 = 15/16																		
23-18	-	-	Reserved																
17	R/W	MulERINT	<p>Multiple early interrupt select: When this bit is set, any received packet invokes early interrupt according to MULINT<MISR[11:0]> setting in early mode. When this bit is reset, the packets of familiar protocol (IPX, IP, NDIS, etc) invoke early interrupt according to RCR<ERTH[3:0]> setting in early mode. The packets of unfamiliar protocol will invoke early interrupt according to the setting of MULINT<MISR[11:0]>.</p>																
16	R/W	RER8	<p>The RTL8139C receives the error packet whose length is larger than 8 bytes after setting the RER8 bit to 1. The RTL8139C receives the error packet larger than 64-byte long when the RER8 bit is cleared. The power-on default is zero. If AER or AR is set, the RER will be set when The RTL8139C receives an error packet whose length is larger than 8 bytes. The RER8 is “Don’t care “ in this situation.</p>																

Table 2.15 (Continued) Receive configuration register [16]

15-13	R/W	RXFTH 2, 1, 0	<p>Rx FIFO Threshold: Specifies Rx FIFO Threshold level. When the number of the received data bytes from a packet, which is being received into The RTL8139C's Rx FIFO, has reached to this level (or the FIFO has contained a complete packet), the receive PCI bus master function will begin to transfer the data from the FIFO to the host memory. This field sets the threshold level according to the following table:</p> <table> <tr> <td>000 = 16 bytes</td> <td>001 = 32 bytes</td> </tr> <tr> <td>010 = 64 bytes</td> <td>011 = 128 bytes</td> </tr> <tr> <td>100 = 256 bytes</td> <td>101 = 512 bytes</td> </tr> <tr> <td>110 = 1024 bytes</td> <td>111 = no rx threshold.</td> </tr> </table> <p>The RTL8139C begins the transfer of data after having received a whole packet in the FIFO.</p>	000 = 16 bytes	001 = 32 bytes	010 = 64 bytes	011 = 128 bytes	100 = 256 bytes	101 = 512 bytes	110 = 1024 bytes	111 = no rx threshold.
000 = 16 bytes	001 = 32 bytes										
010 = 64 bytes	011 = 128 bytes										
100 = 256 bytes	101 = 512 bytes										
110 = 1024 bytes	111 = no rx threshold.										
12-11	R/W	RBLLEN 1, 0	<p>Rx Buffer Length: This field indicates the size of the Rx ring buffer.</p> <table> <tr> <td>00 = 8k + 16 byte</td> <td>01 = 16k + 16 byte</td> </tr> <tr> <td>10 = 32K + 16 byte</td> <td>11 = 64K + 16 byte</td> </tr> </table>	00 = 8k + 16 byte	01 = 16k + 16 byte	10 = 32K + 16 byte	11 = 64K + 16 byte				
00 = 8k + 16 byte	01 = 16k + 16 byte										
10 = 32K + 16 byte	11 = 64K + 16 byte										
10-8	R/W	MXDMA 2, 1, 0	<p>Max DMA Burst Size per Rx DMA Burst: This field sets the maximum size of the receive DMA data bursts according to the following table:</p> <table> <tr> <td>000 = 16 bytes</td> <td>001 = 32 bytes</td> </tr> <tr> <td>010 = 64 bytes</td> <td>011 = 128 bytes</td> </tr> <tr> <td>100 = 256 bytes</td> <td>101 = 512 bytes</td> </tr> <tr> <td>110 = 1024 bytes</td> <td>111 = unlimited</td> </tr> </table>	000 = 16 bytes	001 = 32 bytes	010 = 64 bytes	011 = 128 bytes	100 = 256 bytes	101 = 512 bytes	110 = 1024 bytes	111 = unlimited
000 = 16 bytes	001 = 32 bytes										
010 = 64 bytes	011 = 128 bytes										
100 = 256 bytes	101 = 512 bytes										
110 = 1024 bytes	111 = unlimited										
7	R/W	WRAP	<p>0: The RTL8139C will transfer the rest of the packet data into the beginning of the Rx buffer if this packet has not been completely moved into the Rx buffer and the transfer has arrived at the end of the Rx buffer.</p>								

2.3.3 RTL8139 Interrupt handler

During packet receptions and transmissions, RTL8139 sends an interrupt which is handled by the function `static void rtl8139_Interrupt_Handler (struct Interrupt_State* state)`. There would be 4 possible situations causing an interrupt: (1) transmit complete, (2) transmit error, (3) receive complete, (4) receive error.

An interrupt handler should be as short as possible since it stops the CPU for a while to serve the interrupt. Therefore this serving operation should be completed in the fastest way. In this context, 3 out of 4 possible situations are handled so fast since there is nothing to do in those situations. They are transmit complete, transmit error and receive error states.

On the other hand when receive complete interrupt occurs, packet should be copied to receive buffer if size and status values are reasonable and since there is only one receive buffer in RTL8139, the next packet's address should be calculated. After the calculation, a kernel thread is invoked to copy the data into the buffer by a function `void TakeAPack(ulong_t arg)`.

2.3.4 RTL8139 Transmit operation

The host CPU initiates a transmit operation by storing an entire packet of data in one of the descriptors in the main memory. When the entire packet has been transferred to the Tx buffer, the RTL8139C is instructed to move the data from the Tx buffer to the internal transmit FIFO in PCI bus master mode. When the transmit FIFO contains a complete packet or is filled to the programmed threshold level, the RTL8139C begins packet transmission [17].

2.3.5 RTL8139 Receive operation

The incoming packet is placed in the RTL8139C's Rx FIFO. Concurrently, the RTL8139C performs address filtering of multicast packets according to its hash

algorithms. When the amount of data in the Rx FIFO reaches the level defined in the Receive Configuration Register, the RTL8139C requests the PCI bus to begin transferring the data to the Rx buffer in PCI bus master mode [17].

3. DESIGN AND REALIZATION OF TCP/IP STACK

To provide data communication between computers running GeekOS, it is needed to provide a protocol stack which describes all rules to communicate. For this purpose, a common TCP/IP model split the design into five layers which are consisting of seven layers of Open Systems Interconnection Basic Reference Model in today's Internet world. To implement a generic and modern TCP/IP stack all the layers of the stack must be designed according to this model even though that does not satisfy all the needs to realization but considered an excellent place to begin the study of network architecture.

On the basis of explanation of a TCP/IP stack model, there are two approaches which explains the needs and working principles of a layer in the layered-network model. First one is called the top-down approach which starts to describe the layers from top to down and the second one is called bottom-up approach which starts to describe the layers from bottom to up in the OSI model. According to the development of this thesis, it is more suitable to explain things with the second approach.

3.1 OSI Model

The Open Systems Interconnection Basic Reference Model (*OSI Reference Model* or *OSI Model* for short) is a layered, abstract description for communications and computer network protocol design. It was developed as part of the Open Systems Interconnection (OSI) initiative and is sometimes known as the OSI seven layer model. From top to bottom, the OSI Model consists of the Application, Presentation, Session, Transport, Network, Data Link, and Physical layers. A layer is a collection of related functions that provides services to the layer above it and receives service from the layer below it. For example, a layer that provides error-free communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that make up the contents of the path.

In 1977, work on a layered model of network architecture, which was to become the OSI model, started in the American National Standards Institute (ANSI) working group on Distributed Systems (DISY). With the DISY work and worldwide input, the International Organization for Standardization (ISO) began to develop its OSI networking suite. The International Organization for Standardization (ISO) is a worldwide federation of national standards bodies from some 130 countries, one from each country. According to Bachman, the term "OSI" came into use on 12 October 1979. OSI has two major components: an abstract model of networking (the Basic Reference Model, or seven-layer model) and a set of concrete protocols [12, 15].

Parts of OSI have influenced Internet protocol development, but none more than the concrete operational system model itself, documented in ISO 7498 and its various agenda. In this model, a networking system is divided into layers. Within each layer, one or more entities implement its functionality. Each entity interacts directly only with the layer immediately beneath it, and provides facilities for use by the layer above it.

In particular, Internet protocols are deliberately not as rigorously designed as the OSI model, but a common version of the TCP/IP model splits it into four layers. The Internet Application Layer includes the OSI Application Layer, Presentation Layer, and most of the Session Layer. Its End-to-End Layer includes the graceful close function of the OSI Session Layer as well as the Transport Layer. Its Internetwork Layer is equivalent to the OSI Network Layer, while its Interface layer includes the OSI Data Link and Physical Layers. These comparisons are based on the original seven-layer protocol model as defined in ISO 7498, rather than refinements in such things as the Internal Organization of the Network Layer document [3, 4].

3.2 TCP/IP Model

The TCP/IP Model is a specification for computer network protocols created in the 1970s by DARPA, an agency of the United States Department of Defense. It laid the foundations for ARPANET, which were the world's first wide area network

and a predecessor of the Internet. The TCP/IP Model is sometimes called the Internet Reference Model, the DoD Model or the ARPANET Reference Model [7].

TCP/IP defines a set of rules to enable computers to communicate over a network. TCP/IP provides end to end connectivity specifying how data should be formatted, addressed, shipped, routed and delivered to the right destination. The specification defines protocols for different types of communication between computers and provides a framework for more detailed standards [4].

TCP/IP is generally described as having five 'layers' if you include the bottom physical layer. The layer view of TCP/IP is based on the seven-layer OSI Reference Model written long after the original TCP/IP specifications, and is not officially recognized. Regardless, it makes a good analogy for how TCP/IP works and comparison of the models is common.

The TCP/IP Model and related protocols are currently maintained by the Internet Engineering Task Force (IETF) [3, 4].

3.2.1 Architectural principles

In this model, layers are not separated so rigid such in OSI model, therefore it maintains an easier structure to fit in the real world protocols. Rather than emphasizing on layers on this model, it simply defines two layer classes called Internetworking layer and upper layers. Actually this is the main difference between OSI model and the TCP/IP model.

There are versions of this model with four layers and with five layers. It simply defines a four-layer version, with the layers having names, not numbers.

Process Layer or Application Layer is where the higher level protocols such as SMTP, FTP, SSH, HTTP, etc. operate [7, 9].

Host-To-Host (Transport) Layer is where flow-control and connection protocols exist, such as UDP. This layer deals with opening and maintaining connections, ensuring that packets are in fact received.

Internet or Internetworking Layer defines IP addresses, with many routing schemes for navigating packets from one IP address to another.

Network Access Layer describes both the protocols (i.e., the OSI Data Link Layer) used to mediate access to shared media, and the physical protocols and technologies necessary for communications from individual hosts to a medium [14].

The Internet protocol suite (and corresponding protocol stack) and its layering model were in use before the OSI model was established. Since then, the TCP/IP model has been compared with the OSI model numerous times in books and classrooms, which often results in confusion because the two models use different assumptions, including about the relative importance of strict layering [2, 4].

3.2.2 Layers in the TCP/IP model

The layers near the top are logically closer to the user application (as opposed to the human user), while those near the bottom are logically closer to the physical transmission of the data. Viewing layers as providing or consuming a service is a method of abstraction to isolate upper layer protocols from the detail of transmitting bits over Ethernet, while the lower layers avoid having to know the details of each and every application and its protocol [2]. A sample network connection between two hosts over stack layers is shown in Figure 3.1 and a sample packet encapsulation over the network is shown in Figure 3.2.

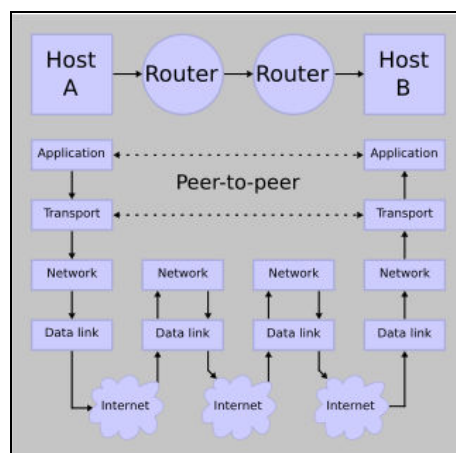


Figure 3.1 Network connections over two routers and corresponding stack layers [18]

This abstraction also allows upper layers to provide services that the lower layers cannot, or choose not to, provide. Again, the original OSI Reference Model was extended to include connectionless services (OSIRM CL). For example, IP is not designed to be reliable and is a best effort delivery protocol. This means that all transport layers must choose whether or not to provide reliability and to what degree. UDP provides data integrity but does not guarantee delivery; TCP provides both data integrity and delivery guarantee [19].

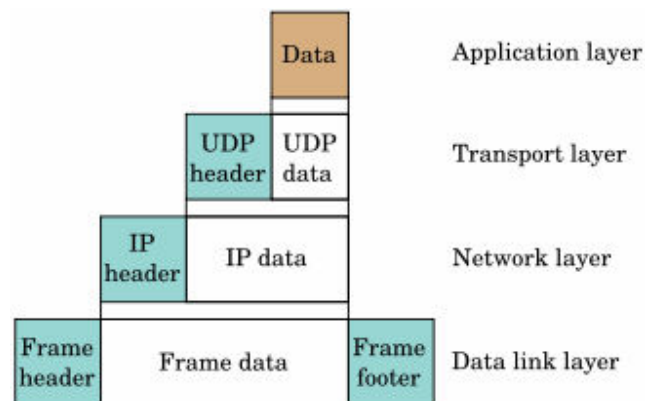


Figure 3.2 Packet encapsulation over layers for a UDP packet [18]

3.3 Physical Layer

The physical layer is the first level in the seven-layer OSI model of computer networking as well as in the five-layer TCP/IP reference model. It performs services requested by the data link layer [13].

The physical layer is the most basic network layer, providing only the means of transmitting raw bits rather than packets over a physical data link connecting network nodes. Neither packet headers nor trailers are consequently added to the data by the physical layer. The bit stream may be grouped into code words or symbols, and converted to a physical signal, which is transmitted over a physical transmission

medium. The physical layer provides an electrical, mechanical, and procedural interface to the transmission medium. An analogy of this layer in a physical mail network would be the roads along which the vans carrying the mail drive. [1, 3]

In this thesis a RTL8139 type Ethernet card is used as the physical and link layer of the protocol stack. The Ethernet physical layer evolved over a considerable time span and encompasses quite a few physical media interfaces and several magnitudes of speed. The speed ranges from 3 Mbit/s to 10 Gbit/s in speed while the physical medium can range from bulky coaxial cable to twisted pair to optical fiber. In general, network protocol stack software will work identically on most of the cable types [2, 12].

3.4 Implementation of the Packet Structures and Functions In Stack

To be able to hold packet as streams of bits, some packet structures are defined. Those structures show how to handle the data and where to put specific information in the header.

For this purpose a general packet structure is defined with the following struct:

```

struct oPacket
{
    char * mPacket;
    unsigned ll;
    unsigned nl;
    unsigned tl;
    unsigned al;
    unsigned tail;
    struct oPacket * mNext;
};

```

This structure holds a pointer, “mPacket”, to where actual data starts. “ll”, “nl”, “tl” and “al” are offset values for the link layer, network layer, transport layer, application layer headers respectively. “tail” points to the end of the packet. Finally, “mNext” points to the next packet.

There are also some basic functions to handle for the incoming and outgoing network packets such as creation of a new packet, destroying a packet, pushing and popping a packet, and copying the packet into a buffer. The details of these functions are given below.

```
struct oPacket * CreateNewPacket(int size)
{
    struct oPacket * newPack = (struct oPacket *)
Malloc(sizeof(struct oPacket));

    newPack->ll = 0;
    newPack->nl = 0;
    newPack->tl = 0;
    newPack->al = 0;
    newPack->tail = 0;
    newPack->mNext = NULL;
    newPack->mPacket = (char *) Malloc(size);
    return newPack;
}
```

```
void DestroyPacket(struct oPacket * pack)
{
    if (pack->mPacket)
        Free(pack->mPacket);
    pack->mNext=NULL;
    Free(pack);
}
```

```
void PushPacket(struct oPacket * newPack)
{
    if (gPacketListTail == NULL) {
        gPacketListHead = gPacketListTail = newPack;
    }
    else {
        gPacketListTail->mNext = newPack;
        gPacketListTail = newPack;
    }
}
```

```
struct oPacket * PopPacket()
{
    struct oPacket * firstPack = gPacketListHead;
    if (gPacketListHead) {
        gPacketListHead = gPacketListHead->mNext;
        if (gPacketListHead == NULL)
            gPacketListTail = NULL;
        firstPack->mNext = NULL;
    }
    return firstPack;
}
```

```
void LoadPacketData(struct oPacket* pack, char * data, size_t len)
{
    memcpy(pack->mPacket, data, len);
}
```

3.5 Data Link Layer

The data link layer is the second layer of the seven-layer OSI model as well as of the five-layer TCP/IP reference model. It responds to service requests from the network layer and issues service requests to the physical layer.

This is the layer which transfers data between adjacent network nodes in a wide area network or between nodes on the same local area network segment. The data link layer provides the functional and procedural means to transfer data between network entities and might provide the means to detect and possibly correct errors that may occur in the Physical layer. Examples of data link protocols are Ethernet for local area networks and PPP, HDLC and ADCCP [7] for point-to-point connections.

The data link provides data transfer across the physical link. That transfer might or might not be reliable; many data link protocols do not have acknowledgments of successful frame reception and acceptance, and some data link protocols might not even have any form of checksum to check for transmission errors. In those cases, higher-level protocols must provide flow control, error checking, and acknowledgments and retransmission [2, 3].

3.6 Implementation of the Data Link Layer

Implementation of the link layer is done by implementing a structure to hold link layer header and two functions called LLSend and LLRecv to have a service interface to network layer and physical layer, respectively.

The link layer header structure is implemented with a struct as follows:

```
struct LL_Hdr{
    unsigned char DestMAC[6];
    unsigned char SrcMAC[6];
    short Protocol;
};
```


First 6 bytes of this structure hold the destination MAC address; next 6 bytes hold source MAC and last 2 bytes hold the protocol number to identify the upper layer protocol, which can be either IP or ARP in our case.

In function `void LLSend(struct oPacket * pack)`, a packet structure is taken, source and destination MAC address fields are filled and protocol number field is set according to the protocol numbers of IP and ARP protocols. Then 32-bit CRC is calculated and appended to the tail of the packet for error detection and copied to the next available transmit buffer. Then next available transmit buffer is set by incrementing once and taking modulo 4 since there are four distinct transmit buffers on RTL8139.

In function `void LLRecv(struct oPacket * pack)`, a network packet is received. The last 4 bytes are taken from the end of the packet as being calculated CRC value at the sender side and a new CRC is calculated in the receiver side again. If both values are equal and the received packet's destination MAC is either receiver node's MAC address or broadcast address, i.e., `0xFFFFFFFFFFFF`, then one of two network layer protocols' receive function is called according to the protocol number field in the link layer header after setting the network layer header offset.

3.7 Network Layer

The network layer is the third layer in the seven layer OSI model, and the third layer in the five layer TCP/IP model. In the TCP/IP reference model it is called the Internet layer. In all of the models, the network layer responds to service requests from the transport layer and issues service requests to the data link layer.

In essence, the network layer is responsible for end-to-end (source to destination) packet delivery, whereas the data link layer is responsible for node to node (hop to hop) frame delivery [2, 3].

The network layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination via one or more networks while maintaining the quality of service, and error control functions.

The network layer deals with transmission of information all the way from its source to its destination - transmission from anywhere to anywhere. Some of the things that the network layer needs to address are: (1) Is the network connection-oriented or connectionless, (2) how to manage global addresses, and (3) how to forward a message.

3.8 Implementation of the Network Layer

There are two network layer protocols implemented in this thesis: (1) Internet Protocol (IPv4), and (2) Address Resolution Protocol (ARP). The implementation details of these protocols are given in sections 3.8.1 and 3.8.2.

3.8.1 Address Resolution Protocol (ARP) and Its Implementation

In computer networking, the Address Resolution Protocol (ARP) is the standard method for finding a host's hardware address when only its network layer address is known [7]. The packet structure of the ARP is shown in Table 3.1

ARP is not an IP-only or Ethernet-only protocol; it can be used to resolve many different network-layer protocol addresses to hardware addresses, although, due to the overwhelming prevalence of IPv4 and Ethernet, ARP is primarily used to translate IP addresses to Ethernet MAC addresses [1].

The packet structure given in Table 3.1 is used for ARP requests and replies. On Ethernet networks, these packets use an EtherType of 0x0806, and are sent to the broadcast MAC address of FF:FF:FF:FF:FF:FF.

- **Hardware type (HTYPE)**
Each data link layer protocol is assigned a number used in this field. For example, Ethernet is 1.
- **Protocol type (PTYPE)**
Each protocol is assigned a number used in this field. For example, IPv4 is 0x0800.

Table 3.1 Address resolution protocol packet structure [19]

	Bits 0-7	Bits 8-15	Bits 16-31
0	Hardware Type(HTYPE)		Protocol Type (PTYPE)
32	Hardware Length(HLEN)	Protocol Length (PLEN)	Operation (OPER)
64	Sender hardware address (SHA)		
?	Sender protocol address (SPA)		
?	Target hardware address (THA)		
?	Target protocol address (TPA)		

- Hardware length (HLEN)
Length in bytes of a hardware address. Ethernet addresses are 6 bytes long.
- Protocol length (PLEN)
Length in bytes of a logical address. IPv4 addresses are 4 bytes long.
- Operation
Specifies the operation the sender is performing: 1 for request, and 2 for reply.
- Sender hardware address (SHA)
Hardware address of the sender.
- Sender protocol address (SPA)
Protocol address of the sender.
- Target hardware address (THA)
Hardware address of the intended receiver. This field is ignored in requests.
- Target protocol address (TPA)
Protocol address of the intended receiver.

In the implementation of ARP, a specific structure is defined to handle the header of this protocol. This header is given in the following structure:

```
struct ARP_Hdr
{
    unsigned char HardwareType[2];
    unsigned char ProtocolType[2];
    unsigned char HardwareAddrLen;
    unsigned char ProtocolAddrLen;
    unsigned char Opcode[2];
    unsigned char SourceHardwareAddr[6];
    unsigned char SourceProtocolAddr[4];
    unsigned char DestinationHardwareAddr[6];
    unsigned char DestinationProtocolAddr[4];
};
```

Also the implementation has a function to receive an ARP packet called `ARPRecv()`. This function takes an ARP packet and according to the operation field either takes the needed IP address from the body of the packet or prepares a reply packet and sends it back to the sender.

3.8.2 Internet Protocol (IPv4) and Its Implementation

The Internet Protocol (IPv4) is a data-oriented protocol used for communicating data across a packet-switched Internetwork.

IPv4 is a network layer protocol in the Internet protocol suite and is encapsulated in a data link layer protocol, such as Ethernet). As a lower layer protocol, IP provides the service of communicable unique global addressing amongst computers [4, 15]. The packet structure of the IP is shown in Table 3.2

An IP packet consists of a header section and a data section.

Table 3.2 Internet protocol packet structure [19]

	Bits 0-3	Bits 4-7	Bits 8-15	Bits 16-18	Bits 19-31
0	Version	Header Length	Type of Service	Total Length	
32	Identification			Flags	Fragment Offset
64	Time To Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options				
160 or 192+	Data				

Header

The header consists of 13 fields, of which only 12 are required. The 13th field is optional (red background in table) and aptly named: options. The fields in the header are packed with the most significant byte first (big endian), and for the diagram and discussion, the most significant bits are considered to come first. The most significant bit is numbered 0, so the version field is actually found in the four most significant bits of the first byte [1, 4].

- Version

The first header field in an IP packet is the four-bit version field. For IPv4, this has a value of 4 (hence the name IPv4).

- Internet Header Length (IHL)

The second field is a four-bit Internet Header Length (IHL) telling the number of 32-bit words in the header. Since an IPv4 header may contain a variable number of options, this field specifies the size of the header (this also coincides with the offset to

the data). The minimum value for this field is 5, which is a length of $5 \times 32 = 160$ bits. Being a four-bit field, the maximum length is 15 words or 480 bits.

- Type of Service (TOS)

bits 0–2: precedence (111 - Network Control, 110 - Internetwork Control, 101 - CRITIC/ECP, 100 - Flash Override, 011 - Flash, 010 - Immediate, 001 - Priority, 000 - Routine)

bit 3: 0 = Normal Delay, 1 = Low Delay

bit 4: 0 = Normal Throughput, 1 = High Throughput

bit 5: 0 = Normal Reliability, 1 = High Reliability

bits 6–7: Reserved for future use

This field is now used for DiffServ and ECN. The original intention was for a sending host to specify a preference for how the datagram would be handled as it made its way through an Internet. For instance, one host could set its IPv4 datagrams' TOS field value to prefer low delay, while another might prefer high reliability. In practice, the TOS field has not been widely implemented. However, a great deal of experimental, research and deployment work has focused on how to make use of these eight bits. These bits have been redefined, most recently through DiffServ working group in the IETF and the Explicit Congestion Notification code point. New technologies are emerging that requires real-time data streaming and therefore will make use of the TOS field. An example is Voice over IP (VoIP) that is used for interactive data voice exchange [11, 15].

- Total Length

This 16-bit field defines the entire datagram size, including header and data, in bytes. The minimum-length datagram is 20 bytes (20-byte header + 0 bytes data) and the maximum is 65,535, i.e., the maximum value of a 16-bit word. The minimum size datagram that any host is required to be able to handle is 576 bytes, but most modern hosts handle much larger packets.

- Identification

This field is an identification field and is primarily used for uniquely identifying fragments of an original IP datagram. Some experimental work has

suggested using the ID field for other purposes, such as for adding packet-tracing information to datagrams in order to help trace back datagrams with spoofed source addresses.

- Flags

A three-bit field follows and is used to control or identify fragments. They are (in order, from high order to low order):

Don't Fragment (DF)

More Fragments (MF)

If the DF flag is set and fragmentation is required to route the packet then the packet will be dropped. This can be used when sending packets to a host that does not have sufficient resources to handle fragmentation.

When a packet is fragmented all fragments have the MF flag set except the last fragment, which does not have the MF flag set. The MF flag is also not set on packets that are not fragmented — an unfragmented packet is its own last fragment.

- Fragment Offset

The fragment offset field, measured in units of eight-byte blocks, is 13 bits long and specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram. The first fragment has an offset of zero. This allows a maximum offset of 65,528 which would exceed the maximum IP packet length of 65,535 with the header length included.

- Time To Live (TTL)

An eight-bit time to live (TTL) field helps prevent datagrams from persisting (e.g. going in circles) on an Internet. Historically the TTL field limited a datagram's lifetime in seconds, but has come to be a hop count field. Each packet switch (or router) that a datagram crosses decrements the TTL field by one. When the TTL field hits zero, the packet is no longer forwarded by a packet switch and is discarded. Typically, an ICMP message (specifically the time exceeded) is sent back to the sender that it has been discarded. The reception of these ICMP messages is at the heart of how traceroute works.

- Protocol

This field defines the protocol used in the data portion of the IP datagram. The Internet Assigned Numbers Authority maintains a list of Protocol numbers and were originally defined in RFC 790.

- Header Checksum

The 16-bit checksum field is used for error-checking of the header. At each hop, the checksum of the header must be compared to the value of this field. If a header checksum is found to be mismatched, then the packet is discarded. Note that errors in the data field are up to the encapsulated protocol to handle — indeed, both UDP and TCP have checksum fields.

Since the TTL field is decremented on each hop and fragmentation is possible at each hop then at each hop the checksum will have to be recomputed.

- Source address

An IPv4 address is a group of four eight-bit octets for a total of 32 bits. The value for this field is determined by taking the binary value of each octet and concatenating them together to make a single 32-bit value.

For example, the address 10.9.8.7 (00001010.00001001.00001000.00000111 in binary) would be 00001010000010010000100000000111.

This address is the address of the sender of the packet. Note that this address may not be the "true" sender of the packet due to network address translation. Instead, the source address will be translated by the NATing machine to its own address. Thus, reply packets sent by the receiver are routed to the NATing machine, which translates the destination address to the original sender's address.

- Destination address

Identical to the source address field but indicates the receiver of the packet.

- Options

Additional header fields may follow the destination address field, but these are not often used. Note that the value in the IHL field must include enough extra 32-bit words to hold all the options (plus any padding needed to ensure that the header contains an integral number of 32-bit words). The list of options may be terminated

with an EOL (End of Options List, 0x00) option; this is only necessary if the end of the options would not otherwise coincide with the end of the header. The possible options that can be put in the header are given in Table 3.3.

Table 3.3 Option bits in IP header [19]

Field	Size (bits)	Description
Copied	1	Set to 1 if the options need to be copied into all fragments of a fragmented packet.
Option Class	2	A general options category. 0 is for " <i>control</i> " options, and 2 is for " <i>debugging and measurement</i> ". 1, and 3 are reserved.
Option Number	5	Specifies an option.
Option Length	8	Indicates the size of the entire option (including this field). This field may not exist for simple options.
Option Data	Variable	Option-specific data. This field may not exist for simple options.

The use of the LSRR and SSRR options (Loose and Strict Source and Record Route) is discouraged because they create security concerns; many routers block packets containing these options [4].

Data

The last field is not a part of the header and, consequently, not included in the checksum field. The contents of the data field are specified in the protocol header field and can be any one of the transport layer protocols [1, 4].

In this thesis, IP protocol is handled by a structure and two functions to send and receive IP packets over the network. The header structure of the IP protocol is handled by the following struct.

```
struct IP_Hdr
{
    unsigned char VersionAndIHL;
    unsigned char TypeOfService;
    unsigned char TotalLength[2];
    unsigned char Identification[2];
    unsigned char FlagsAndFragmentOffset[2];
    unsigned char TimeToLive;
    unsigned char Protocol;
    unsigned char HeaderChecksum[2];
    unsigned char SourceIP[4];
    unsigned char DestinationIP[4];
};
```

Also two functions are implemented to send/receive IP packets. In `IPReceive()` function, an incoming packet's header is analyzed and if the destination IP address of the packet is equal to the receiver host's IP address, the packet is delivered to the upper layer after setting the transport layer offset in the packet structure. Otherwise it is dropped.

In `IPSend()` function, the packet is received from the transport layer and network layer header is added in front of it. To set the destination IP address field in the network layer header, ARP is used to determine the MAC address of the destination host. Until the reply comes back from ARP, it waits for the response. When the response comes, this function constructs the IP protocol header and sends it thorough the lower layer, i.e., data link layer.

3.9 Transport Layer

The transport layer is the second highest layer in the five layer TCP/IP reference models, where it responds to service requests from the application layer and

issues service requests to the network layer. It is also the name of layer four of the seven layer OSI model, where it responds to service requests from the session layer and issues service requests to the network layer.

A transport protocol is a protocol on the transport layer. The two most widely used transport protocols on the Internet are the connection oriented Transmission Control Protocol (TCP), and connectionless User Datagram Protocol (UDP). TCP is more complicated and most common one. Other options are the Datagram Congestion Control Protocol (DCCP) and Stream Control Transmission Protocol (SCTP) [8, 9].

The transport layer is typically handled by processes in the host computer operating system, and not by routers and switches. The transport layer usually turns the unreliable and very basic service provided by the network layer into a more powerful one.

In the TCP/IP model, the transport layer is responsible for delivering data to the appropriate application process on the host computers. This involves statistical multiplexing of data from different application processes, i.e. forming data packets, and adding source and destination port numbers in the header of each transport layer data packet. Together with the source and destination IP address, the port numbers constitutes a network socket, i.e. an identification address of the process-to-process communication.

Some transport layer protocols, for example TCP but not UDP, support virtual circuits, i.e. provide connection oriented communication over an underlying packet oriented datagram network [8]. A byte-stream is delivered while hiding the packet mode communication for the application processes. This involves connection establishment, dividing of the data stream into packets called segments, segment numbering and reordering of out-of order data.

Finally, some transport layer protocols, for example TCP but not UDP, provides end-to-end reliable communication, i.e. error recovery by means of error detecting code and automatic repeat request (ARQ) protocol. The ARQ protocol also provides flow control, which may be combined with congestion avoidance [2].

UDP is a very simple service, and does not provide virtual circuits, or reliable communication, leaving these to the application. UDP packets are called datagrams rather than segments [1].

TCP is used by many protocols, including web browsing (HTTP) [9] and email transfer (SMTP). UDP may be used for multicasting and broadcasting, since retransmissions are not possible to a large amount of hosts. UDP typically gives higher throughput and shorter latency, and is therefore often used for real-time multimedia communication, where packet loss occasionally can be accepted, for example IP-TV and IP-telephony, and for online computer games.

In many non-IP-based networks, for example X.25, Frame Relay and ATM, the connection oriented communication is implemented at network layer or data link layer rather than the transport layer. In X.25, in telephone network modems and in wireless communication systems, reliable node-to-node communication is implemented at lower protocol layers [2, 3].

3.10 Implementation of Transport Layer

In this thesis, User Datagram Protocol (UDP) is implemented as the transport layer protocol. The implementation details of this protocol are explained in Section 3.10.1.

3.10.1 User datagram protocol (UDP) and its implementation

User Datagram Protocol is one of the core protocols of the Internet protocol suite. Using UDP, programs on networked computers can send short messages sometimes known as datagrams (using Datagram Sockets) to one another. UDP is sometimes called the Universal Datagram Protocol. The protocol was designed by David P. Reed in 1980.

UDP does not guarantee reliability or ordering in the way that TCP does. Datagrams may arrive out of order, appear duplicated, or go missing without notice.

Avoiding the overhead of checking whether every packet actually arrived makes UDP faster and more efficient, for applications that do not need guaranteed delivery. Time-sensitive applications often use UDP because dropped packets are preferable to delayed packets. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers) .

Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and online games [7, 11].

In the Internet protocol suite, UDP provides a very simple interface between a network layer below (e.g., IPv4) and a session layer or application layer above.

UDP provides no guarantees to the upper layer protocol for message delivery and a UDP sender retains no state on UDP messages once sent (for this reason UDP is sometimes called the Unreliable Datagram Protocol). UDP adds only application multiplexing and checksumming of the header and payload. If any kind of reliability for the information transmitted is needed, it must be implemented in upper layers [4]. The packet structure of the UDP is shown in Table 3.4.

Table 3.4 User datagram protocol header [19]

	Bits 0-15	Bits 16-31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

The UDP header consists of only 4 fields [19].

- Source port
This field identifies the sending port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero.

- Destination port
This field identifies the destination port and is required.
- Length
A 16-bit field that specifies the length in bytes of the entire datagram: header and data. The minimum length is 8 bytes since that's the length of the header. The field size sets a theoretical limit of 65,535 bytes for the data carried by a single UDP datagram. The practical limit for the data length which is imposed by the underlying IPv4 protocol is 65,507 bytes.
- Checksum
The 16-bit checksum field is used for error-checking of the header and data.

In this thesis, UDP is implemented by defining a structure to construct the UDP header, and two functions to send and receive UDP packets over the network. . The UDP header structure is handled by the following struct.

```
struct UDP_Hdr{  
    unsigned char SourcePort[2];  
    unsigned char DestinationPort[2];  
    unsigned char Length[2];  
    unsigned char CheckSum[2];  
};
```

To send/receive UDP packets over the network, two distinct functions are implemented. These are UDPReceive() and UDPSend() functions.

In UDPReceive() function a UDP packet structure is received from the network and UDP header is analyzed. According to the header information, the packet is sent to the application waiting for this packet from a specific port after setting the application layer offset. The distinction between different applications is made with port number values in the incoming packet. There would be 2^{16} , i.e.,

65536, possible port number values, and multiplexing of applications are done with those distinct port values.

In `UDPSend()` function a UDP packet is prepared using the source and destination port numbers. The length of the datagram including the UDP header is calculated and written into the length field of the header. Finally, a 16-bit checksum is added for error detection.

3.11 Application Layer

The application layer is the seventh level of the seven-layer OSI model, and the fifth layer of the five-layer TCP/IP model. It interfaces directly to and performs common application services for the application processes; it also issues requests to the presentation layer.

The application layer of the five-layer TCP/IP model corresponds to the application layer, the presentation layer and session layer in the seven layer OSI model [1, 3].

3.12 Implementation of Application Layer

To demonstrate that our device driver and the TCP/IP stack implementation works and interoperates with existing standards compliant TCP/IP implementations, we have developed a simple distributed game called tic-tac-toe. The game is developed as a client/server application and works by exchanging messages between the applications over the network.

3.12.1 Tic-tac-toe game

Tic-tac-toe, also called doughnuts and crosses, hugs and kisses, and many other names, is a pencil-and-paper game for two players, O and X, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three respective

marks in a horizontal, vertical or diagonal row wins the game [18]. The game shown in Figure 3.3 is won by the first player, X:



Figure 3.3 A sequence of tic-tac-toe game [18]

3.12.2 Structure of client/server application

Game is served by one of the players. The client application joins the game and the game is started by putting an X by the server. Then game continues by putting Xs and Os respectively by client and server players. At the end, according to the sequence of the game either one of the players wins the game or a draw happens.

In this thesis, the application server is run in a computer running Microsoft Windows XP and the client application is run on OSman. Using different platforms, i.e., two different operating systems, shows that not only the implemented device driver works perfectly, but also the implemented TCP/IP protocol stack is able to communicate with another standards-compliant TCP/IP protocol stack implementation.

In Figures 3.4 and 3.5 two screenshots are demonstrated. In these figures OSman is emulated with an emulator (qemu) having RTL8139 type Ethernet card emulation. Communication between Windows XP and OSman is done in the following way: A virtual private network is established on Windows XP, which creates a virtual local area network called Tap. This local area network's parameters are configured to communicate with emulated OSman and packets sent from Windows XP passes through Tap and reach OSman. Also the packets generated and sent from OSman by RTL8139 emulation are received by Tap. With this emulation system, some screenshots are taken. Also the system is able to do the same job by running on two different real computers as well.


```

QEMU
TxOK
IP layer SI 192.168.10.254 DI 192.168.10.121 - Protocol 17 TOS 0 L:30
UDP : SP8eb DP1234 Len0a CS1cc1
UDP datagram size : 2 (CHC 1CC1 o:1CC1 diff 0)
UDP data : 1 2
-----
UDP SEND : SP433 DP1234 Len0c CSec42 4
TxOK
IP layer SI 192.168.10.254 DI 192.168.10.121 - Protocol 17 TOS 0 L:32
UDP : SP8eb DP1234 Len0c CS8615
UDP datagram size : 4 (CHC 8615 o:8615 diff 0)
UDP data : T e s t
-----
UDP SEND : SP433 DP1234 Len0c CSec42 4
TxOK
IP layer SI 192.168.10.254 DI 192.168.10.121 - Protocol 17 TOS 0 L:40
UDP : SP8eb DP1234 Len014 CS257e
UDP datagram size : 12 (CHC 257E o:257E diff 0)
UDP data : T e s t M e s s a g e
-----
UDP SEND : SP433 DP1234 Len0c CSec42 4
TxOK
This packet is not mine (Drop)
-----

```

Figure 3.4 UDP packet receiving and printing in OSman

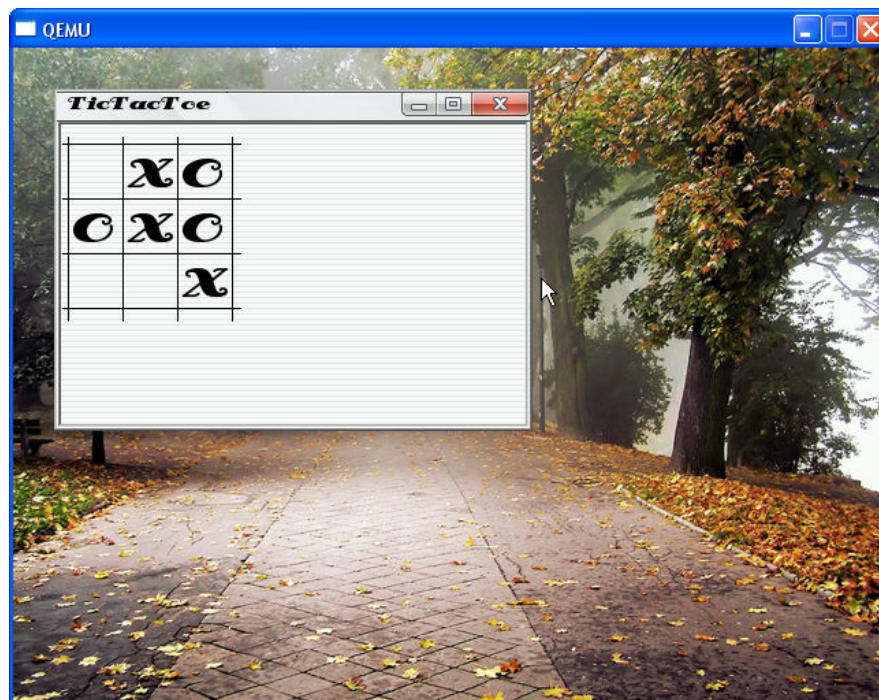


Figure 3.5 A game sequence in GUI platform of OSman

4. CONCLUSIONS AND FUTURE WORK

In this thesis, a device driver for a network interface card is implemented for a newly developed operating system, which has various capabilities such as real multi-programming, high resolution VESA graphics card driver, virtual x86 Mode support, PCI driver system base, PCI device detection, MFC like event table based GUI system, and cross platform gcc and g++ port. Also a TCP/IP protocol stack is built on top of the implemented Ethernet device driver. Thus our operating system, called OSman, is able to establish network connections.

Driver implementation of the system is done for a specific network interface card. Since all PCI based cards have different configuration and programming specifications, the implementations for some other cards must be done according to their specifications. But PCI configuration space has a standard structure. According to this structure, program design of the device driver is written and packets coming from the network are received.

A standards-compliant TCP/IP protocol stack has been implemented to interpret the incoming bits to the card and also send packets to the network. Finally, a distributed game application has been developed on top of the TCP/IP stack to demonstrate that the implemented device driver and the TCP/IP protocol stack can interoperate with another standards-compliant TCP/IP protocol suite, that of the Windows XP Operating System. This game is a simple client/server application to show that system works perfectly.

As for future work, other widely used Ethernet cards' driver implementations can be developed for OSman to support a variety of Ethernet cards. Similarly, drivers for other PCI based cards, e.g., sound cards, video cards, can be developed on this system based on the PCI driver implementation of this thesis.

As an enhancement to the developed protocol stack implementation, other transport layer protocols such as TCP, SCTP can be added for reliable, in-order, and connection oriented communications. This way, other important applications such as a web explorer can be developed on OSman.

REFERENCES

- [1] Kurose J., F., Ross, K., W., *Computer Networking*, Addison Wesley Longman, Inc., USA, 2001.
- [2] Taylor, E., *TCP/IP Complete*, McGraw-Hill Professional Book Group, USA, 1998.
- [3] Wetteroth, D., *OSI Reference Model for Telecommunications*, McGraw-Hill Professional Publishing, USA, 2001.
- [4] Blank, A., G., *TCP/IP JumpStart: Internet Protocol Basics*, Sybex, Inc., USA, 2002.
- [5] Herbert, T., *Linux TCP/IP Stack: Networking for Embedded Systems*, Charles River Media, USA, 2004.
- [6] Muller, N., J., *IP from A to Z*, McGraw-Hill Professional Publishing, 2002.
- [7] Stevens, W., R., *TCP/IP Illustrated, Volume 1: The Protocols*, Addison Wesley, 1994.
- [8] Stevens, W., R., *TCP/IP Illustrated, Volume 2: The Implementation*, Addison Wesley, 1995.
- [9] Stevens, W., R., *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*, Addison-Wesley, 1996.
- [10] Carne, B., *A Professional's Guide to Data Communication in a TCP/IP World*, Artech House, Inc., 2004.
- [11] Feit, S., *TCP/IP: Architecture Protocols & Implementation with IPV6 & IP Security*, McGraw-Hill Professional Book Group, 1998.
- [12] Miller, M., A., *Internet Technologies Handbook: Optimizing the IP Network*, John Wiley & Sons, Inc., 2005.
- [13] Keogh, J., *Ethernet in the First Mile*, McGraw-Hill Professional Publishing, 2002.
- [14] Axelson, J., *Embedded Ethernet and Internet Complete*, Lakeview Research, 2003.

- [15] Russell, T., *Telecommunications Basics*, McGraw-Hill Professional Publishing, 1997.
- [16] Datasheet, *Realtek 3.3V Single Chip Fast Ethernet Controller With Power Management RTL8139C(L)*, Realtek Semiconductor Corp., Rev 1.4, 2002.
- [17] Datasheet, *RTL8139 Programming guide: (V0.1)*, Realtek Semiconductor Corp., 1999.
- [18] Wikipedia, the Online Free Encyclopedia, <http://www.wikipedia.org>, 2008.
- [19] RFC web site, <http://www.ietf.org>, 2008.
- [20] GeekOS project web site, <http://geekos.sourceforge.net>, 2008.
- [21] Video Electronics Standard Association web site, <http://www.vesa.org>, 2008.