Department of Computer Engineering   
Qatar University

**Advance IP services (DHCP and DNS)**

Report by

Mahmoud Abdelaziz 201704234

Ahmad Salem 201602464

Hasan Alsamra 201707182

Instructor

Eng. Heba Dawoud

Date

15/11/2020

# ABSTRACT

Advance IP services (DHCP and DNS) : In your local host, you are required to develop a multithreaded Server/Client application capable of serving different IP address services in parallel for different clients. The application has the following specifications: The server should be always listening for incoming connections on a separate thread , The server should have a predefined list of services including the DHCP

**TABLE OF CONTENTS**

Contents

[Contents ii](#_Toc56525953)

[1 Introduction 1](#_Toc56525954)

[2 Source files 2](#_Toc56525955)

[2.1 DHCP.go 2](#_Toc56525956)

[2.2 Client.go 3](#_Toc56525957)

[2.3 Server.go 5](#_Toc56525958)

[2.4 go.mod file main package 10](#_Toc56525959)

[2.5 go.mod file DHCP package 10](#_Toc56525960)

[3 Console output 11](#_Toc56525961)

[3.1 Client.go 11](#_Toc56525962)

[3.2 Server.go 11](#_Toc56525963)

[4 CONCLUSION 12](#_Toc56525964)

# LIST OF FIGURES

[Figure ‎1‑1 1](#_Toc56354435)

# Introduction

First of all, DHCP is a frequently used communication protocol that enables dynamic assignment of IP addresses to administrators of a dedicated network overview. On the Transport Layer, which is a central, unidirectional protocol in the Internet Protocol Set, DHCP uses UDP. UDP offers the ability to send datagrams with an IP address to other hosts, with no need to set up a data transmission channel for communication. This contributes to the lack of need to retransmit information, resulting in less delay. It does not require handshaking, but it informs the consumer if any unreliable transmission has occurred. Nevertheless the delivery is not guaranteed to be resolved, nor will there be insurance. UDP applies checksums and individual port numbers via the source and destination of the datagram to handle the various applications on the application layer. UDP is transaction driven, meaning, it is applicable for query requests (e.g. DNS servers). It is really easy to use. It is stateless, too which is beneficial when it comes to media streaming. DHCP is standardized and is used on a network by specialist servers to send computers IP addresses. A network administrator is not needed, but an overview of address allocation is possible.

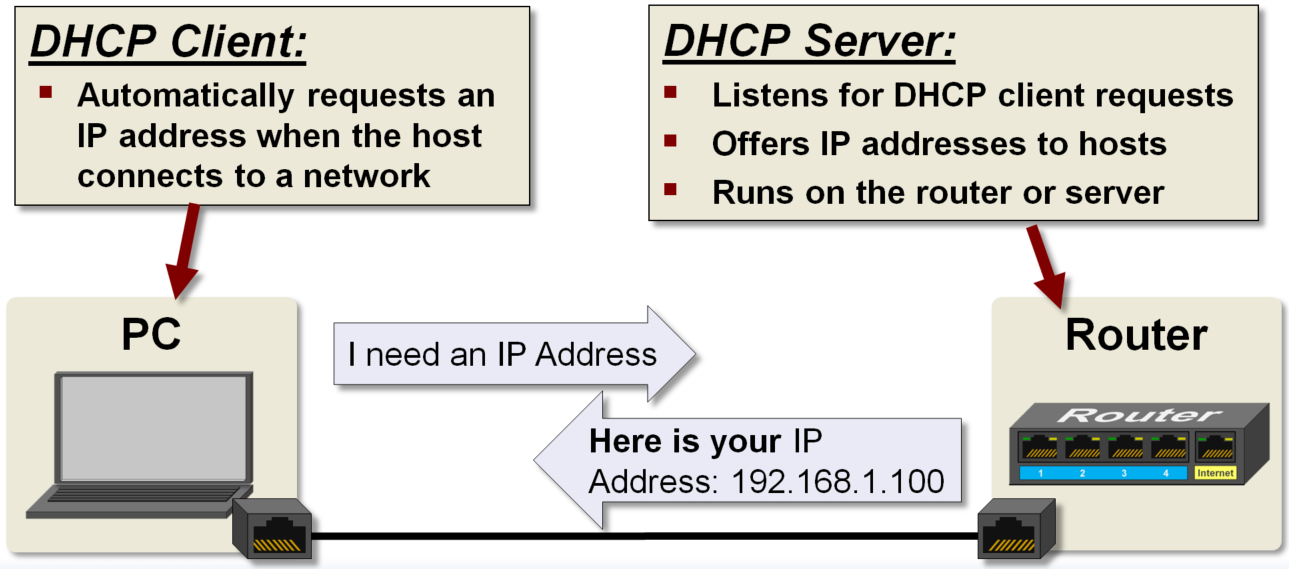


Figure ‎1‑1

So, to be able to connect to the internet, a specific IP address invigorates a PC device. DHCP is of high importance, without it in a factory or a large company, we will need to enter the IP address of each computer manually. This is not traditional in any way, and it could lead to a lot of time being lost. DHCP also provides the possibility to supervise the distribution of IP addresses from a central point, and when a PC connects to the network, it assigns IP addresses.

# Source files

We start write the code the DHCP server should be able to handle concurrent clients’ connection for IP address requests. The server can be acting as a router, as shown in Figure 1: Providing IP Addressing Services – DHCP.

## DHCP.go

package dhcp

// Creating a DHCP OFFER class

// Please note that the first name of a variable should be capitalized in order to accessed from outside the package

type DHCP struct{

Ip\_offer string

Default\_gateway string

Subnet\_mask string

Dns\_address string

}

## Client.go

package main

import (

"fmt"

"net"

"os"

"bufio"

"encoding/gob"

"osproject.qu/dhcp"

)

const (

discover = "DHCPDISCOVER"

request = "DHCPREQUEST"

)

func main() {

// Send a connection request to the server

conn, err := net.Dial("tcp", "localhost:8000")

if err != nil{

fmt.Println("Error Connection", err)

os.Exit(1)

}

defer conn.Close()

// printing the ip and port

fmt.Println("Successfully Connected with", conn.RemoteAddr().(\*net.TCPAddr).IP, "on port", conn.RemoteAddr().(\*net.TCPAddr).Port)

// Sending the discover message to the server

fmt.Fprintf(conn, discover + "\n")

// Getting Server Proposal

dec:= gob.NewDecoder(conn)

msg1 := &dhcp.DHCP{}

dec.Decode(msg1)

fmt.Printf("\nServer offered \n%+v\n\nI think I like it\n\n", msg1)

// Sending DHCP request

fmt.Fprintf(conn, request + "\n")

// Reading the ack message

buffer := bufio.NewReader(conn)

read,\_ := buffer.ReadBytes('\n')

msg2 := string(read[:len(read)-1])

fmt.Println("The server likes it too\n" + msg2)

// Handling lease questions

done := make(chan string)

go handleLease(done, buffer, conn)

// Sending and Receiving UDP DNS requests

dns()

dns2()

<-done

}

func handleLease(done chan<- string, buffer \*bufio.Reader, conn net.Conn){

for {

read, err := buffer.ReadBytes('\n')

if read != nil && err == nil{

fmt.Println("\nServer is checking on me")

fmt.Fprintf(conn, "present\n")

} else { fmt.Println("Error Reading",err); done<- "Done" }

}

}

// Simple DNS test

func dns(){

fmt.Println("\nRequesting ip address of PC 1")

// Setting up the destination

conn, \_ := net.Dial("udp", ":8001")

// Sending the domain

fmt.Fprintf(conn, "PC 1"+"\n")

// Receiving the Reply

buf,\_,\_ := bufio.NewReader(conn).ReadLine()

fmt.Println(string(buf))

}

// False domain DNS test

func dns2(){

fmt.Println("\nRequesting ip address of P 1")

// Setting up the destination

conn, \_ := net.Dial("udp", ":8001")

// Sending the domain

fmt.Fprintf(conn, "P 1"+"\n")

// Receiving the Reply

buf,\_,\_ := bufio.NewReader(conn).ReadLine()

fmt.Println(string(buf))

}

## Server.go

package main

import (

"fmt"// strings

"net" //sockets

"bufio" //buffers

"math/rand" //rand

"time"// time

"errors"//errors

"sync"//syncronization

"encoding/gob"//sending structs over tcp

"osproject.qu/dhcp"

)

const (

ack = "DHCPACK"

)

var ip\_table = make(map[string]string)

var mutex = &sync.Mutex{}

func main(){

ip\_table["192.168.0.1"] = "Default Gateway"

ip\_table["192.168.0.2"] = "DNS 1"

ip\_table["192.168.0.3"] = "DNS 2"

ip\_table["192.168.255.254"] = "Broadcast"

// Channel that we will use

done := make(chan string, 2)

go tcp\_listen(done)

go udp\_listen(done)

<-done

<-done

}

func tcp\_listen(done chan<- string){

// Creating a Listiner

in, err := net.Listen("tcp", ":8000")

if err != nil {

fmt.Println("Error Listening:", err)

done <- "done"

}

defer in.Close()

fmt.Println("Listener Started")

// Infinite loop to accept new clients and handle them concurrently

for {

conn, err := in.Accept()

if err != nil{

fmt.Println("Error Connecting:", err)

continue

}

// Handling each client in a separate thread

offer\_ip := make(chan string, 1)

go handleConnection(conn, offer\_ip)

fmt.Println("New Connection from IP address", conn.RemoteAddr().(\*net.TCPAddr).IP, "came\n")

}

done <- "done"

}

func handleConnection(conn net.Conn, offer\_ip chan string){

// Wrapping the connection with buferio reader and reading till the terminator

buffer, err := bufio.NewReader(conn).ReadBytes('\n')

if err != nil{

fmt.Println("The Client Left\n")

conn.Close()

return

}

// type cast the buffer to a string

msg := string(buffer[:len(buffer)-1])

fmt.Println("The client Sent This Message", msg, "\n")

if msg == "DHCPDISCOVER"{

dhcp\_offer, err := makeOffer()

if err != nil {

fmt.Println("Error while adding:", err)

}

// Sending the proposed ip to the client

encoder := gob.NewEncoder(conn)

encoder.Encode(dhcp\_offer)

fmt.Printf("Here is the offer sent to the client:\n%+v\n\n", dhcp\_offer)

// sending the ip to else if

offer\_ip <- dhcp\_offer.Ip\_offer

// Get the client reply

handleConnection(conn, offer\_ip)

} else if msg == "DHCPREQUEST" {

// receiving the ip from the channel

ip := <-offer\_ip

fmt.Printf("Client wants the ip \"%s\"\n\n", ip)

mutex.Lock()

addPC(ip)

mutex.Unlock()

// Sending the Acknowledgment to the client

fmt.Fprintf(conn, ack + "\n")

checkLease(conn, ip)

}

}

// An offer that the client can't refuse ;)

func makeOffer() (\*dhcp.DHCP, error) {

dhcp\_offer := dhcp.DHCP{

Default\_gateway : "192.168.0.1",

Subnet\_mask : "255.255.0.0",

Dns\_address : "192.168.0.2",

}

// If the number of pcs hasn't reached the limit

if len(ip\_table) < (2<<15 - 1) {

// Generate a randomly seeded number

random := rand.NewSource(time.Now().UnixNano())

// Cannot narrow down the range since ips are generated randomly

// max := rand.New(random).Intn( (2<<15 - 2) - len(ip\_table)) + len(ip\_table)

max := rand.New(random).Intn( 2<<15 - 2 )

// Formulating the ip address with the magic of math

fourth\_octet := max%256

third\_octet := (max/256) - (fourth\_octet/256)

ip := fmt.Sprintf("192.168.%d.%d", third\_octet, fourth\_octet)

// If the ip is not already in the table

if \_, found := ip\_table[ip]; !found{

// Adding the ip to the created struct

dhcp\_offer.Ip\_offer = ip

return &dhcp\_offer, nil

} else {

// Calling the function again until it gets an open ip

return makeOffer()

}

} else {return &dhcp\_offer, errors.New("The matrix is full")}

}

// Add the pc to the table

func addPC(ip string) {

pc := fmt.Sprintf("PC %d", len(ip\_table)-3)

ip\_table[ip] = pc

fmt.Println(ip\_table[ip] + " has been added to the DHCP table with ip " + ip)

fmt.Println("\nHere is the full table")

for k,v := range ip\_table {

fmt.Printf("%-20s%-20s\n", k, v)

}

}

// Sleeping for a day and then chekcing if the client is still alive

func checkLease(conn net.Conn, ip string){

for{

// Everyday

// time.Sleep(24 \* time.Hour)

time.Sleep(5 \* time.Second)

fmt.Println("\nChecking on", ip\_table[ip])

// Send this Message

fmt.Fprintf(conn, "Check" + "\n")

// If the bufio function returned an error then the client left

\_, err := bufio.NewReader(conn).ReadBytes('\n')

if err != nil{

fmt.Println("The Client Left\n")

mutex.Lock()

// Remove the entry

delete(ip\_table, ip)

mutex.Unlock()

return

} else { fmt.Println("He is there") }

}

}

// Listening to client UDP requests and replying to them

func udp\_listen(done chan<- string){

// Creating a Listiner

conn, err := net.ListenPacket("udp", ":8001")

if err != nil {

fmt.Println("Error Listening:", err)

done <- "done"

}

go dns\_reply(conn)

}

// Simple DNS implementation

func dns\_reply(conn net.PacketConn){

for {

buf := make([]byte, 1024)

length, source, \_ := conn.ReadFrom(buf)

fmt.Println("\nThis source sent a UDP message:", source)

fmt.Println("Here is the message:",string(buf[:length]))

// Checking if the domain is in the map

found := false

for k,v := range ip\_table {

if v == string(buf[:length-1]) {

buf = []uint8("The IP address is " + k + "\n")

conn.WriteTo(buf, source)

found = true

}

}

// If the "Domain" is not found

if !found{

buf = []uint8("The requested domain name is currently not available" + "\n")

conn.WriteTo(buf, source)

}

}

}

## go.mod file main package

module osproject.org/local

go 1.13

require osproject.qu/dhcp v0.0.0

replace osproject.qu/dhcp => ./dhcp

## go.mod file DHCP package

module osproject.qu/dhcp

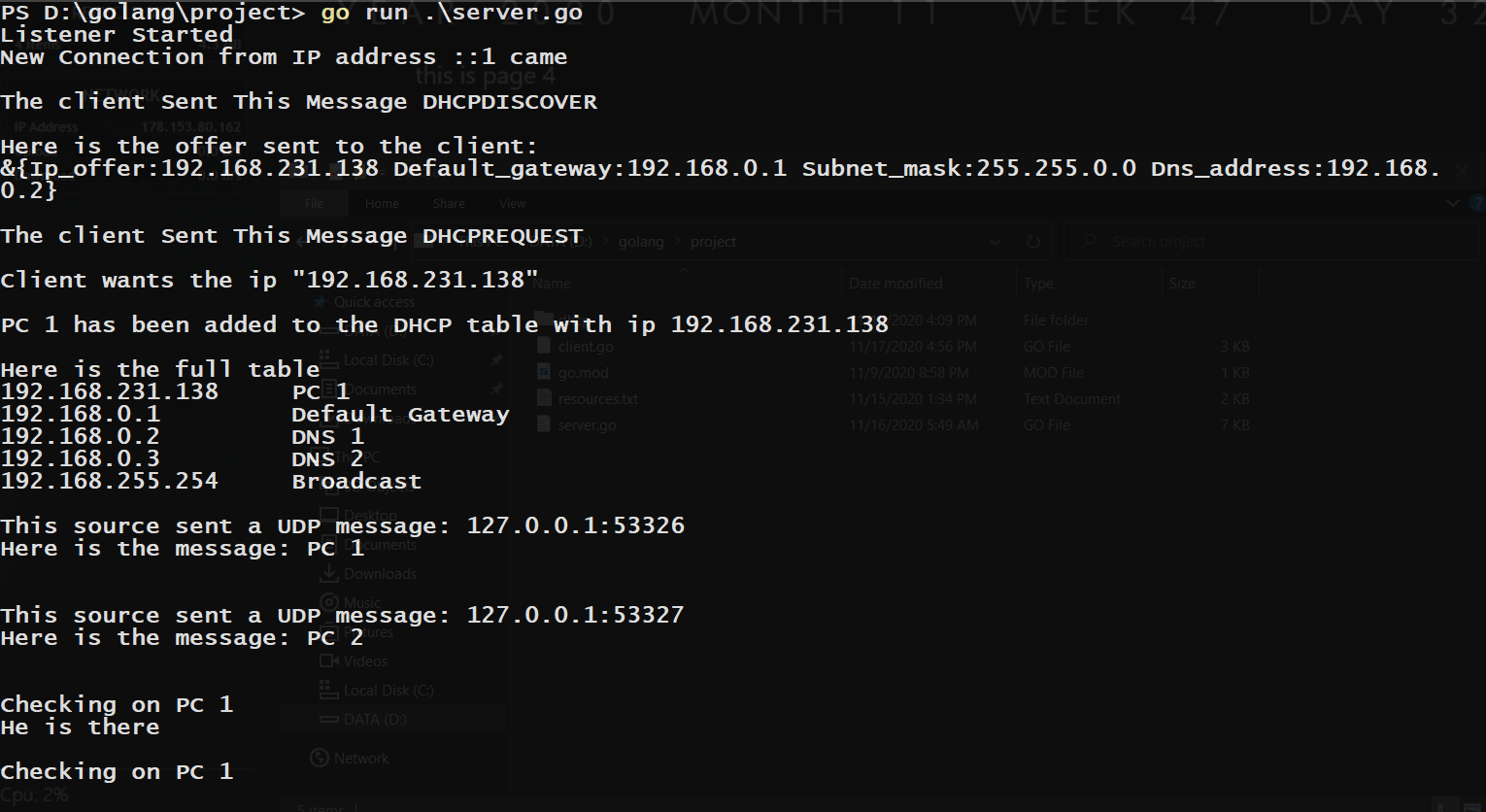
# Console output

In this part, we will show some of the output for the running code.

## Client.go



## Server.go



# CONCLUSION

In this project we were able to show that an operating system is highly complex. Network and socket programming, however has opened many doors to learn further and acquire more information about how to code network communications systems. We became more interested in implementing the principles that we learned from this course and attempting to incorporate more networking-related applications.

Fortunately, we were able to achieve all the objectives, but as always, we faced some challenges this has been open-ended but (at the same time) very rewarding project. As part of our ongoing project and with a strong intention to achieve the project objectives, a list of challenges from different perspectives was confronted. Some problematic issues showed up during disparate project progress stages. Nevertheless, in each challenging step, an alternative viable approach was taken to proceed further in the project schedule.

Any system can be development for the better this project can have more features, but time did not enough.

# REFERENCES

This is a list of all the resources used during the project:

golang Language Basics

gobyexample.com

Broadcast Message ip address (not used in the end):

https://en.wikipedia.org/wiki/Dynamic\_Host\_Configuration\_Protocol#:~:text=The%20DHCP%20client%20broadcasts%20a,broadcast%20address%20(directed%20broadcast).

Networking Library

https://golang.org/pkg/net/

Conn Objects

https://golang.org/pkg/net/?m=all#Conn:~:text=type%20Conn,-Conn

Type Assertions in golang

https://tour.golang.org/methods/15#:~:text=A%20type%20assertion%20provides%20access%20to%20an%20interface%20value%27s%20underlying%20concrete%20value.&text=This%20statement%20asserts%20that%20the,statement%20will%20trigger%20a%20panic.

Socket Programming

https://golangr.com/socket-server/

http://golangr.com/socket-client/

Packages:

https://www.youtube.com/watch?v=sf7f4QGkwfE

Go Path:

https://www.youtube.com/watch?v=QgHkCGkSoy4

using modules:

https://brokencode.io/how-to-use-local-go-modules-with-golang-with-examples/

Sending/Receiving structs over TCP

https://stackoverflow.com/questions/11202058/unable-to-send-gob-data-over-tcp-in-go-programming

String Formatting

https://golang.org/pkg/fmt/

Concurrency Patterns in GO

https://www.youtube.com/watch?v=f6kdp27TYZs

1. Contribution

|  |  |  |
| --- | --- | --- |
| Student Name | Student ID | Section Contributed to |
| Mahmoud Abdelaziz | 201704234 | 33% |
| Ahmad Salem | 201704234 | 33% |
| Hasan Alsamra | 201707182 | 33% |
| Laptops | 010101010 | 1% |