Rede de Computadores

Relatório do 2º trabalho laboratorial

Mestrado Integrado em Engenharia Informática e Computação

Rede de Computadores

Grupo 1

Daniel Garrido – up201403060

Nuno Castro – up201406990

Sara Santos – up201402814

23 de dezembro de 2016

Conteúdo

1. Sumário

Este relatório tem como objetivo abordar e analisar os resultados obtidos ao longo da realização do segundo projeto laboratorial no âmbito da disciplina de Redes e Computadores.

2. Introdução

Este projeto laboratorial dividiu-se em duas partes: implementação de uma aplicação de download e configuração e analise de uma rede de computadores.

Na primeira parte foi desenvolvido uma aplicação para download de um ficheiro obtido a partir de um URL, focando na implementação do protocolo FTP. Por outro lado, na segunda parte deste trabalho laboratorial foram realizadas diversas experiências com o objetivo de implementar e de compreender e analisar uma rede de computadores.

De seguida abordam-se de forma mais profunda e analisam-se todos os tópicos relativos a este projeto.

3. Parte I – Aplicação de Download

O objetivo da primeira parte deste trabalho é criar uma aplicação que transfere ficheiros utilizando o protocolo FTP. Para tal desenvolvemos a aplicação ftp, que funciona da seguinte forma:

A aplicação tem dois modos de funcionamento. O modo normal em que recebe um nome de utilizador e palavra-passe, e o modo anônimo. Neste, a aplicação acede ao servidor FTP com o utilizador “anonymous” palavra-passe indefinida.

A struct urlData guarda a informação introduzida pelo utilizador ao nos parâmetros da aplicação.

```
typedef struct{
    char * user;
    char * password;
    struct hostent * h;
    char * urlPath;
    char * hostIp;
}urlData;
```

Figura 1 *Struct para URL*

Para tal, no inicio da aplicação, e chamada a função getUrlInfo que traduz e valida o input do utilizador para informação que a aplicação consegue usar e subsequentemente guarda essa informação na struct acima.

```
void getUrlInfo(char * completeUrl, urlData * url);
```

Figura 2 *Chamada da funçao getUrlInfo*

Depois de toda a informação do utilizador ter sido lida e validada, a função startConnection é chamada. Esta e responsável por conectar o cliente FTP ao servidor por meio de um socket. Para terminar a ligação falta enviar os comandos de login e PASV. No login são enviados o nome de utilizador e palavra-passe e o comando PASV habilita a comunicação nos dois sentidos.

```
int getControl(FTP * ftp, urlData * url, FTP * receiverFtp){
    if(sendAndReceiveControl(USER, ftp, receiverFtp, url) != 0)
        return -1;

    if(sendAndReceiveControl(PASS, ftp, receiverFtp, url) != 0)
        return -1;

    if(sendAndReceiveControl(PASSIVE, ftp, receiverFtp, url) != 0)
        return -1;

    return 0;
}
```

Figura 3 *Função getControl*

Para ser possível receber um ficheiro é necessário criar uma ligação do servidor para o cliente. A função startReceiverCon faz isso mesmo. Assim estão concluídas as ligações necessárias para receber o ficheiro pedido pelo utilizador. Para tal é chamada a função receiveFile que envia o comando RETR, lê do socket a informação e guarda-a num ficheiro.

```
typedef struct {
    int fdSocket;
    int fdDataSocket;
    int passiveAnswer[6];
    int port;
    char ip[100];
} FTP;
```

Figura 4 *Struct onde é guardada a informação da ligação*

Concluída a transferência do ficheiro é chamada a função closeConnection que fecha as ligações e a aplicação termina.

4. Parte II – Configuração e analise de uma rede de computadores

4.1 Configuração de um IP de rede

O principal objetivo desta experiência era conectar duas máquinas através da configuração do IP de rede. Para tal, através do comado *ifconfig*, configurou-se o IP na interface eth0 das máquinas tux11 e o tux14 para 172.16.10.1 e 172.16.10.254, respetivamente.

De seguida, tal como indicado, verificou-se a conectividade entre os dois computadores recorrendo ao comando *ping*, podendo-se analisar nos registos do *Wireshark* o envio de pacote ICMP e a resposta correspondente.

Ao inspecionar a tabela de rotas verificamos que se encontra, em ambas as máquinas, o endereço para o seu próprio domínio e na tabela ARP a entrada que reconhece o endereço IP com o respetivo endereço MAC. Como tal, quando apagamos as entradas da tabela ARP e efetuamos novamente o *ping* é necessário reconhecer qual o endereço MAC associado ao IP ao qual se tenta fazer *ping*. Para tal, é enviado em modo *broadcast* um pacote ARP que recebe como resposta o endereço MAC correspondente.

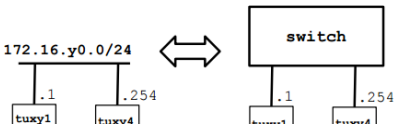


Figura 5 Arquitetura Experiência 1

4.2 Configuração de duas redes LAN virtuais num switch

Esta experiência teve como finalidade a configuração de duas LAN virtuais (VLAN) num switch. Desta forma, foram criadas duas VLAN (VLAN10 e VLAN11) recorrendo à aplicação *gkterm* e aos comandos especificados em anexo, que têm como destino associar as portas que conectam as máquinas à respetiva VLAN. À VLAN10 pertencem o tux10 e o tux14 e à VLAN11 pertence o tux12.

Após as configurações estarem devidamente efetuadas, executou-se novamente o comando *ping*, primeiramente, do tux11 para tux14 e após para o tux12. Verificou-se, assim, que é possível conectar com o tux14, visto que se encontram ligados na mesma rede, porém o mesmo não acontece com o tux12, uma vez que estão ligados a redes diferentes e não existe nenhum tipo de conexão entre as duas. Na experiência seguinte, pretende-se estabelecer essa ligação de forma a ser possível conectar duas máquinas em redes virtuais diferentes (tux11 e tux12).

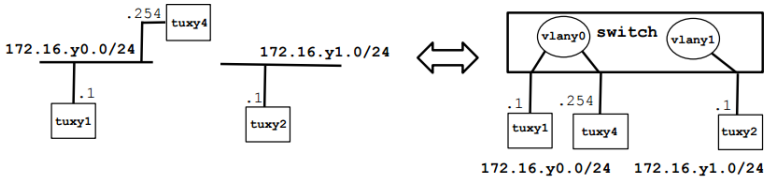


Figura 6 *Arquitetura Experiência 2*

4.3 Configuração de um router em Linux

O objetivo desta experiência é configurar o computador tux4 como router de forma a conectar tux1 e tux2, que se encontram em vlans diferentes. Tux1 (172.16.10.0) pertence à vlan 10 e o tux2 (172.16.11.0) pertence à vlan 11.

Primeiro é necessário adicionar a porta eth1, do tux4 ao switch, para conectar o tux4 com a vlan 11. O endereço de ip desta porta será 172.16.11.253. Para ativar a função de router corre-se o seguinte comando: “echo 1 > /proc/sys/net/ipv4/ip_foward”.

Após adicionar as rotas necessárias no tux1 e tux2 para estes acederem o tux4, com o comando “route add -net”, os dois tuxs já conseguem aceder a redes fora do seu domínio.

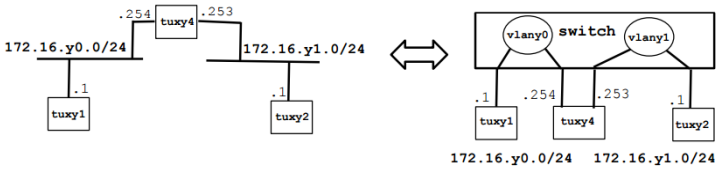


Figura 7 *Arquitetura Experiência 3*

4.4 Configuração de um router comercial implementado com NAT

Esta experiencia consiste em configurar o router CISCO da rede 11 de forma a que os computadores do domínio 10 e 11 tenham acesso à Internet.

Primeiro é necessário aceder a linha de comandos do router através da sua porta serie. Depois de fazer login, executam-se os comandos do anexo //TODO. Estes comandos começam por configurar o router para ter duas interfaces, atribuindo-lhes as configurações de NAT. O NAT (Network Address Translation) traduz os endereços de sub-rede de cada tux para o endereço do router comercial. Sem a correta implementação do NAT, os tuxs não teriam acesso à Internet.

Depois de ser permitido o acesso dos tux às redes criadas, é adicionada uma rota predefinida, para o endereço da Internet. De igual modo é adicionado nos computadores uma rota predefinida para o endereço do router (172.16.11.254).

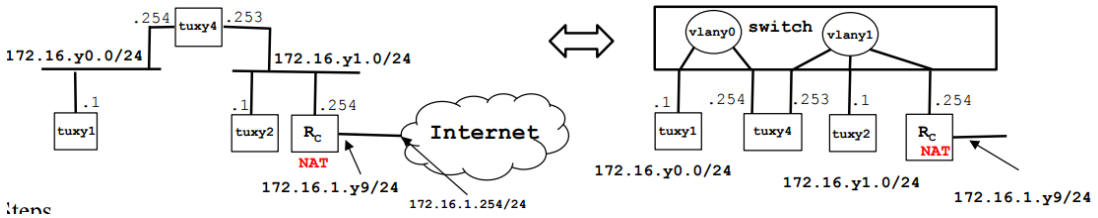


Figura 8 *Arquitetura Experiência 4*

4.5 DNS

Nesta experiência era esperado que fosse configurado o serviço DNS. Para tal, foi alterado o ficheiro **resolv.conf**, no diretório **/etc**, para ter a mesma configuração que o exemplo acima mencionado. Ao executar o comando **ping hostname** (ex **ping google.pt**), verificou-se que ocorria a transferência de pacotes **DNS**, **ICMP**, **STP** e **ARP**, tal como mostra a figura que se encontra em anexo.

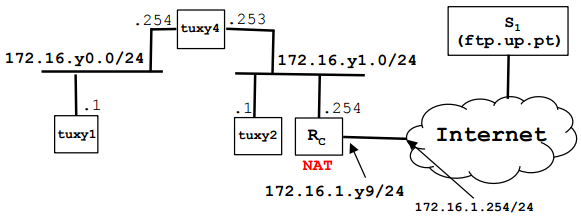


Figura 9 *Arquitetura Experiência 5*

4.6 Conexões TCP

Para esta experiência foi necessário conciliar as duas partes de trabalho: utilizar a aplicação de *download* desenvolvida na primeira parte do trabalho na rede configurada por nós ao longo das experiências. A aplicação possui modo anónimo e não anónimo e foi testada com ficheiros de imagem, texto e pastas comprimidas, que após descompressão não revelaram qualquer tipo de problema com os ficheiros presentes.

O gráfico de pacotes transferidos pelo *tux1* pode ser consultado nas imagens em anexo.

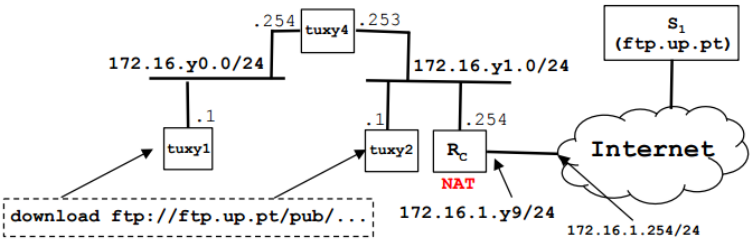


Figura 10 *Arquitetura Experiência 6*

5. Conclusões

O desenvolvimento de todas as experiências juntamente com a aplicação de *download* permitiu-nos compreender como funcionam as transferências de ficheiros, bem como a configuração de uma rede.

Sendo as experiências e a aplicação de fácil compreensão e tendo o grupo concluído todas as atividades propostas, concluímos que o projeto foi terminado com sucesso.

Anexos

A. main.c

```
#include "connection.h"

int main(int argc, char * argv[]){
    if(argc != 2){
        printf("Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
        printf("Usage: %s ftp://<host>/<url-path>\n", argv[0]);
        exit(1);
    }

    urlData * url = malloc(sizeof(urlData));
    urlData * urlReceiver = malloc(sizeof(urlData));

    getUrlInfo(argv[1], url);

    printf("User: %s\n", url->user);
    printf("Password: %s\n", url->password);
    printf("path: %s\n", url->urlPath);
    printf("HostIp: %s\n", url->hostIp);

    FTP ftp;
    FTP ftpReceiver;

    if(startConnection(url, &ftp) != 0){
        printf("Error connecting\n");
        exit(1);
    }

    if(getControl(&ftp, url, &ftpReceiver) != 0){
        printf("Error getting control\n");
        exit(1);
```

```
    }

    if(startReceiverCon(urlReceiver, &ftpReceiver) != 0){
        printf("Error starting receiving connection\n");
        exit(1);
    }

    if(receiveFile(url, &ftp, &ftpReceiver) != 0){
        printf("Error receiving file\n");
        exit(1);
    }

    if(closeConnection(&ftp, &ftpReceiver) != 0){
        printf("Error closing connection\n");
        exit(1);
    }

    free(url->user);
    free(url->password);
    free(url->urlPath);
    free(url);

    return 0;
}
```

B. connection.h

```
#include "url.h"

typedef struct {
    int fdSocket;
    int fdDataSocket;
    int passiveAnswer[6];
    int port;
```

```
   char ip[100];
} FTP;

int startConnection(urlData * url, FTP * ftp);
int getControl(FTP * ftp, urlData * url, FTP * receiverFtp);
int receivePassiveAnswer(FTP * ftp);
int sendAndReceiveControl(int cmd, FTP * ftp, FTP * ftpReceiver, urlData * url);
int startReceiverCon(urlData * urlReceiver, FTP * ftpReceiver);
int receiveFile(urlData * url, FTP * ftp, FTP * ftpReceiver);
int closeConnection(FTP * ftp, FTP * ftpReceiver);
```

C. connection.c

```
#include "connection.h"

int startConnection(urlData * url, FTP * ftp){
   int     sockfd;
           struct    sockaddr_in server_addr;

           /*server address handling*/
           bzero((char*)&server_addr,sizeof(server_addr));
           server_addr.sin_family = AF_INET;
           server_addr.sin_addr.s_addr = inet_addr(url->hostIp);    /*32 bit Internet address
network byte ordered*/
           server_addr.sin_port = htons(SERVER_PORT);                    /*server TCP port must
be network byte ordered */

           /*open an TCP socket*/
           if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
      perror("socket()");
      return -1;
   }
           /*connect to the server*/
   if(connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
      perror("connect()");
```

```
                return -1;
            }

    ftp->fdSocket = sockfd;

    return 0;
}

int getControl(FTP * ftp, urlData * url, FTP * receiverFtp){
    char buffer[1000];
    read(ftp->fdSocket, buffer, 1000);

    if(sendAndReceiveControl(USER, ftp, receiverFtp, url) != 0)
        return -1;

    if(sendAndReceiveControl(PASS, ftp, receiverFtp, url) != 0)
        return -1;

    if(sendAndReceiveControl(PASSIVE, ftp, receiverFtp, url) != 0)
        return -1;

    return 0;
}

int sendAndReceiveControl(int cmd, FTP * ftp, FTP * ftpReceiver, urlData * url){
    char command[256];

    switch(cmd){
        case USER:
            strcpy(command, "user \0");
            strcat(command, url->user);
            break;
```

```
    case PASS:
        strcpy(command, "pass \0");
        strcat(command, url->password);
        break;
    case PASSIVE:
        strcpy(command, "pasv \0");
        break;
    default:
        break;
}


strcat(command, "\n");
if(write(ftp->fdSocket, command, strlen(command)) < 0){
    printf("Error writing to socket\n");
    return -1;
}
sleep(1);

if(cmd == PASSIVE){
    if(receivePassiveAnswer(ftp) == 0){
        ftpReceiver->port = ftp->passiveAnswer[4] * 256 + ftp->passiveAnswer[5];
        memset(ftpReceiver->ip, 0, 256);
        sprintf(ftpReceiver->ip, "%d.%d.%d.%d", ftp->passiveAnswer[0],
            ftp->passiveAnswer[1], ftp->passiveAnswer[2], ftp->passiveAnswer[3]);

    }
}else{
    char answer[256] = "";
    if(read(ftp->fdSocket, answer, 256) <= 0){
        printf("Error reading from socket\n");
        return -1;
```

```
        }
    }
    return 0;
}

int receivePassiveAnswer(FTP * ftp){
    char passiveAnswer[256];
    if(read(ftp->fdSocket, passiveAnswer, 256) <= 0){
        printf("Error reading from socket\n");
          return -1;
    }else{
        int r = sscanf(passiveAnswer, "%*[^(](%d,%d,%d,%d,%d,%d)\n", &(ftp->passiveAnswer[0]),
                &(ftp->passiveAnswer[1]), &(ftp->passiveAnswer[2]), &(ftp->passiveAnswer[3]),
                &(ftp->passiveAnswer[4]), &(ftp->passiveAnswer[5]));
       if(r != 6){
            printf("Error reading answer\n");
             return -1;
        }
    }
    return 0;
}

int startReceiverCon(urlData * urlReceiver, FTP * ftpReceiver){
  int      sockfd;
  struct  sockaddr_in server_addr;
  char * host_ip;

  urlReceiver->h = gethostbyname(ftpReceiver->ip);

  if(urlReceiver->h == NULL){
      printf("Could not find host\n");
      return -1;
```

```
  }

  host_ip = inet_ntoa(*((struct in_addr *)urlReceiver->h->h_addr));

  /*server address handling*/
  bzero((char*)&server_addr,sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = inet_addr(host_ip);        /*32 bit Internet address network byte ordered*/
  server_addr.sin_port = htons(ftpReceiver->port);                    /*server TCP port must be network byte ordered */

  /*open an TCP socket*/
  sockfd = socket(AF_INET,SOCK_STREAM,0);
  if (sockfd < 0) {
        perror("socket()");
        return -1;
      }
  /*connect to the server*/
      if(connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
          perror("connect()");
          return -1;
  }
  ftpReceiver->fdSocket = sockfd;
  ftpReceiver->fdDataSocket = sockfd;

  return 0;
}

void getName(char * url, char ** filename){
    char * temp = url;
    int i = 0;
```

```
    while(temp != NULL){
        if(i > 0)
            *filename = temp + 1;
        else
            *filename = temp;
        temp = strchr(*filename, '/');
        i++;
    }
}

int receiveFile(urlData * url, FTP * ftp, FTP * ftpReceiver){
        write(ftp->fdSocket, "TYPE L 8\r\n", strlen("TYPE L 8\r\n"));

    char command[256] = "";
    strcpy(command, "retr ");
    strcat(command, url->urlPath);
    strcat(command, "\n");

    if(write(ftp->fdSocket, command, strlen(command)) < 0){
      printf("Error sending command\n");
      return -1;
    }

    char * filename;
    FILE* file;
    int res;
    char temp[256];

    getName(url->urlPath, &filename);
    if(!(file = fopen(filename, "wb"))){
        printf("Error opening file\n");
        return -1;
```

```
    }

    sprintf(temp,"%d",ftpReceiver->fdDataSocket);

    char buf[1024];
    while((res = read(ftpReceiver->fdDataSocket, buf, sizeof(buf)))){
        if(res < 0){
            printf("Error reading\n");
            return -1;
        }
        if((res = fwrite(buf, res, 1, file)) < 0){
            printf("Error writing to file\n");
            return -1;
        }
    }
    fclose(file);
    return 0;
}

int closeConnection(FTP * ftp, FTP * ftpReceiver){
    int res;
    res = close(ftpReceiver->fdSocket);
    if(res < 0){
        printf("Error closing FTP receiver\n");
        return -1;
    }

    res = close(ftp->fdSocket);
    if(res < 0){
        printf("Error closing FTP\n");
        return -1;
    }
```

```
    return 0;
```

D. }url.h

```
#include "utilities.h"

void getUrlInfo(char * completeUrl, urlData * url);
```

E. url.c

```
#include "url.h"

void getUrlInfo(char * completeUrl, urlData * url){
    //verifica o inicio do url
    if(strncmp(completeUrl, "ftp://", 6)){
        printf("Wrong url: expected ftp://...\n");
        exit(1);
    }

    char * mode = strchr(completeUrl, '@');
    if(mode == NULL){
        printf("Anonymous mode\n");
    }else{
        printf("Standard mode\n");
    }

    int userPassLength, userLength, passLength, hostLength, urlPathLength;
    char * colon = strchr(completeUrl + 6, ':');
    char * slash = strchr(completeUrl + 7, '/');

    if(slash == NULL){
        printf("Wrong url\n");
        exit(1);
    }

    if(mode == NULL){
```

```
    userPassLength = 0;
    passLength = 0;
    userLength = 0;
    hostLength = (int) (slash - completeUrl - 6);
    urlPathLength = strlen(completeUrl) - (7 + userLength + passLength + hostLength);
}else{
    userPassLength = (int) (mode - completeUrl - 6);
    userLength = (int) (colon - completeUrl -6);
    passLength = (int) (userPassLength - userLength - 1);
    hostLength = (int) (slash - mode - 1);
    urlPathLength = strlen(completeUrl) - (9 + userLength + passLength + hostLength);
}

if(hostLength <= 0 || urlPathLength <= 0){
    printf("Wrong url: host and Path length can't be 0\n");
    exit(1);
}



url->urlPath = malloc(sizeof(char) * urlPathLength);

char hostTemp[100];

if(mode == NULL){
    url->user = malloc(sizeof(char) * strlen("anonymous"));
    strncpy(url->user, "anonymous", strlen("anonymous"));
    url->password = malloc(sizeof(char) * strlen("bill9gates@"));
    strncpy(url->password, "bill9gates@", strlen("bill9gates@"));
    strncpy(hostTemp, completeUrl + 6, hostLength);
}else{
    url->user = malloc(sizeof(char) * userLength);
    strncpy(url->user, completeUrl + 6, userLength);
```

```
        url->password = malloc(sizeof(char) * passLength);
        strncpy(url->password, completeUrl + userLength + 7, passLength);
        strncpy(hostTemp, mode + 1, hostLength);
    }
    strncpy(url->urlPath, slash + 1, urlPathLength);
    hostTemp[hostLength] = '\0';

    if((url->h = gethostbyname(hostTemp)) == NULL){
        herror("gethostbyname");
        exit(1);
    }

    int l = strlen(inet_ntoa(*((struct in_addr *)url->h->h_addr)));
    url->hostIp = calloc(l, sizeof(char) * l);
    strncpy(url->hostIp, inet_ntoa(*((struct in_addr *)url->h->h_addr)), l);
}
```

F. utilities.h

```
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <errno.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/socket.h>
#include <arpa/inet.h>

typedef struct{
    char * user;
    char * password;
```

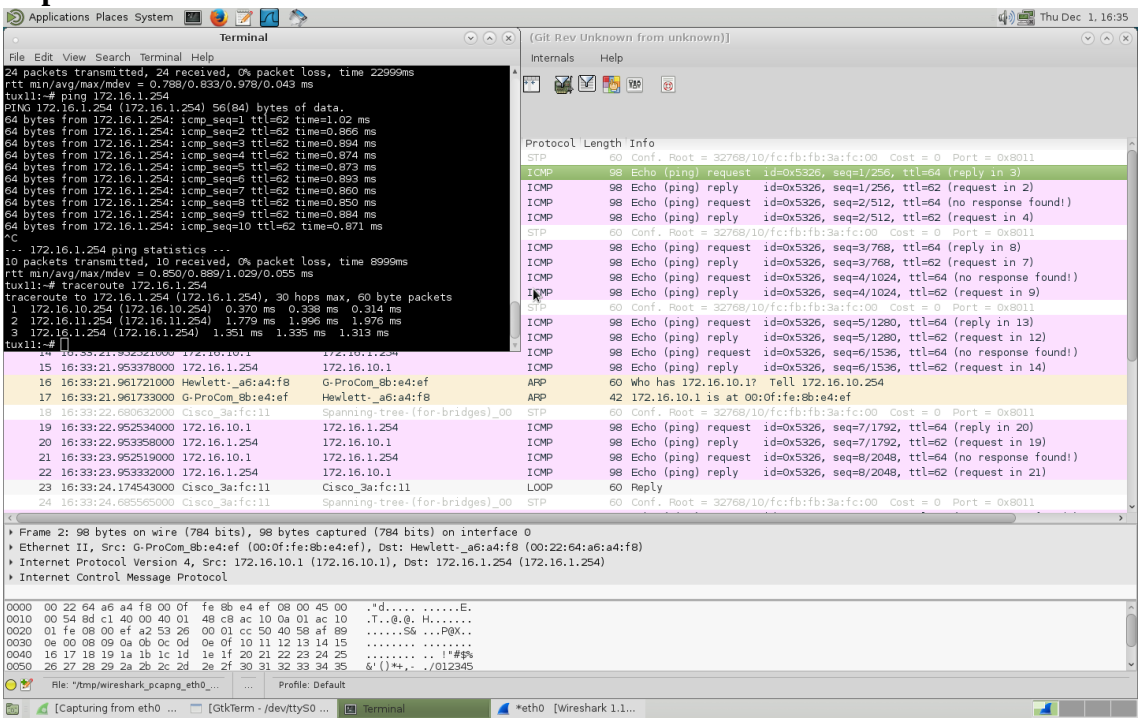
```
    struct hostent * h;
    char * urlPath;
    char * hostIp;
}urlData;


#define SERVER_PORT 21


#define USER 0
#define PASS 1
#define PASSIVE 2
```

G. Makefile

```
CC = gcc
CFLAGS= -Wall
all:ftp.o
ftp.o: url.c connection.c main.c
        $(CC) $(CFLAGS) -o ftp url.c connection.c main.c
```

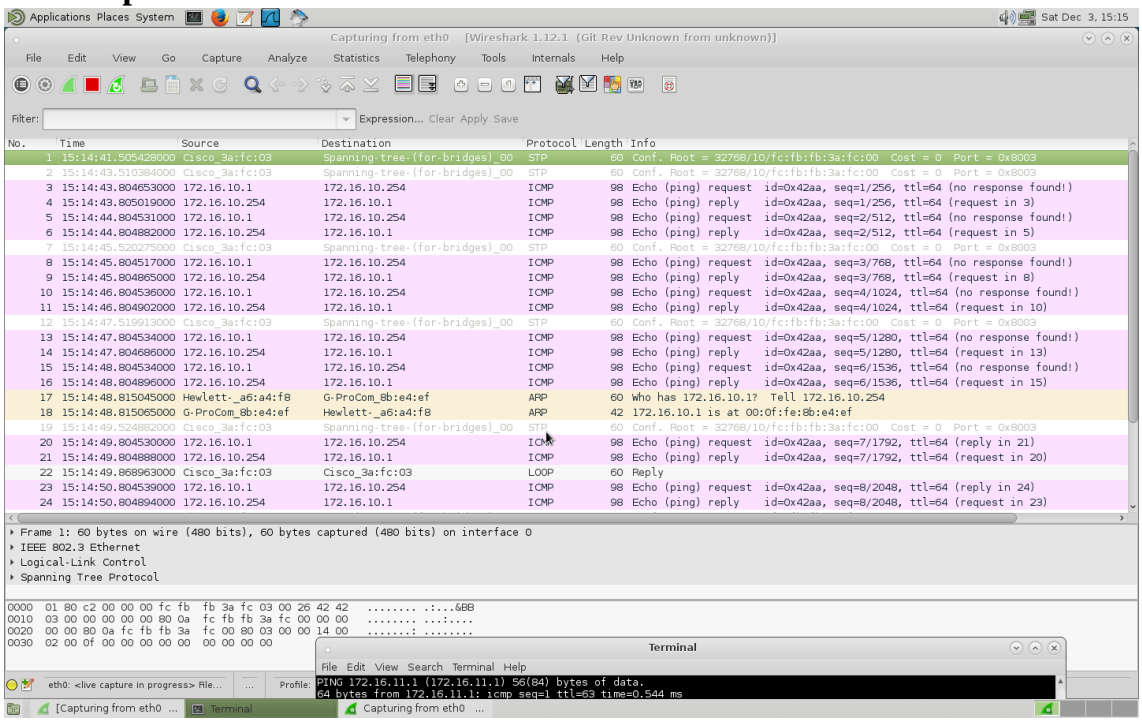
H. Wireshark logs

Experiência 1

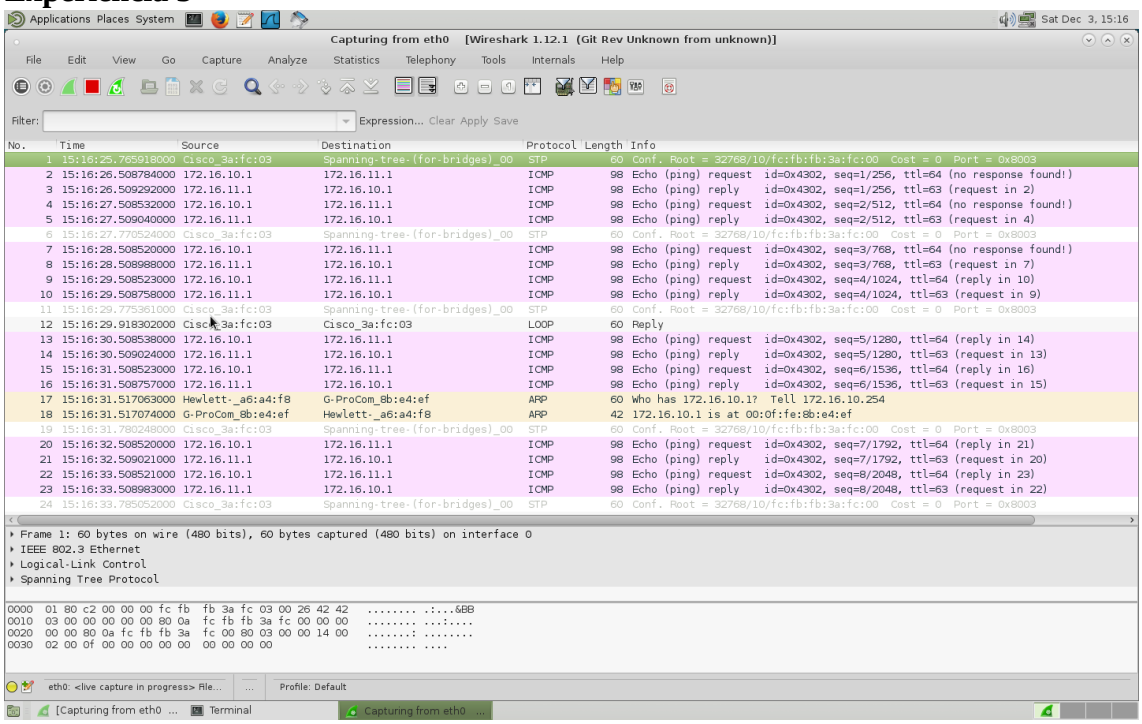


```
```

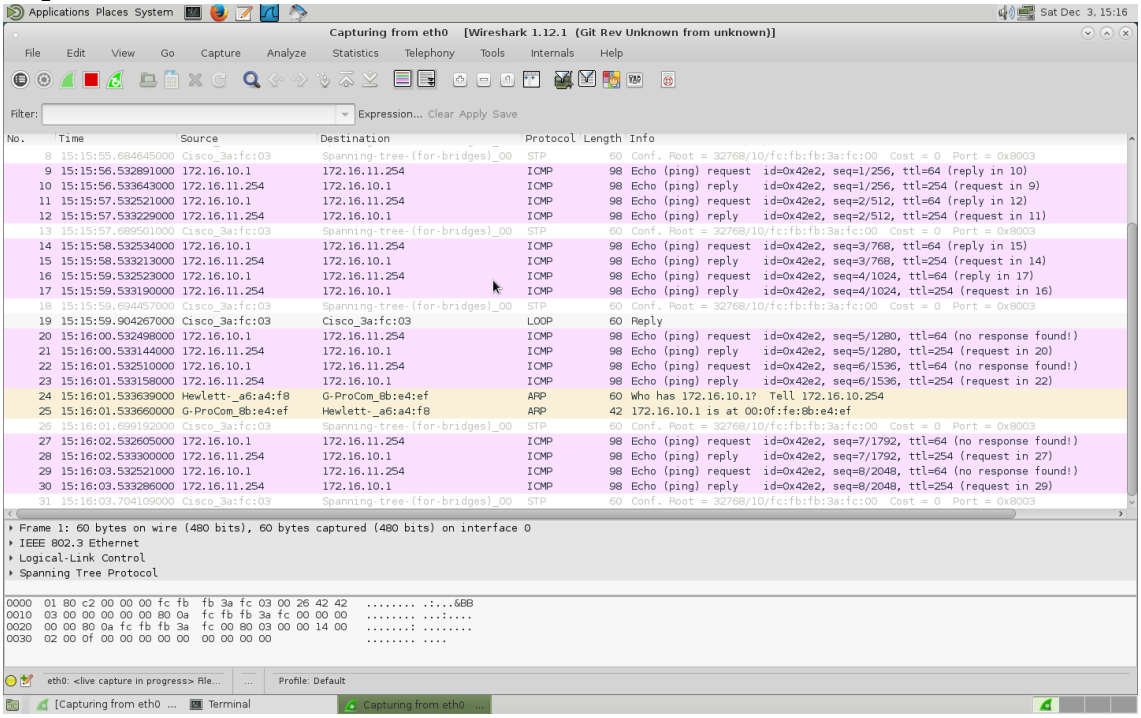
Experiência 2

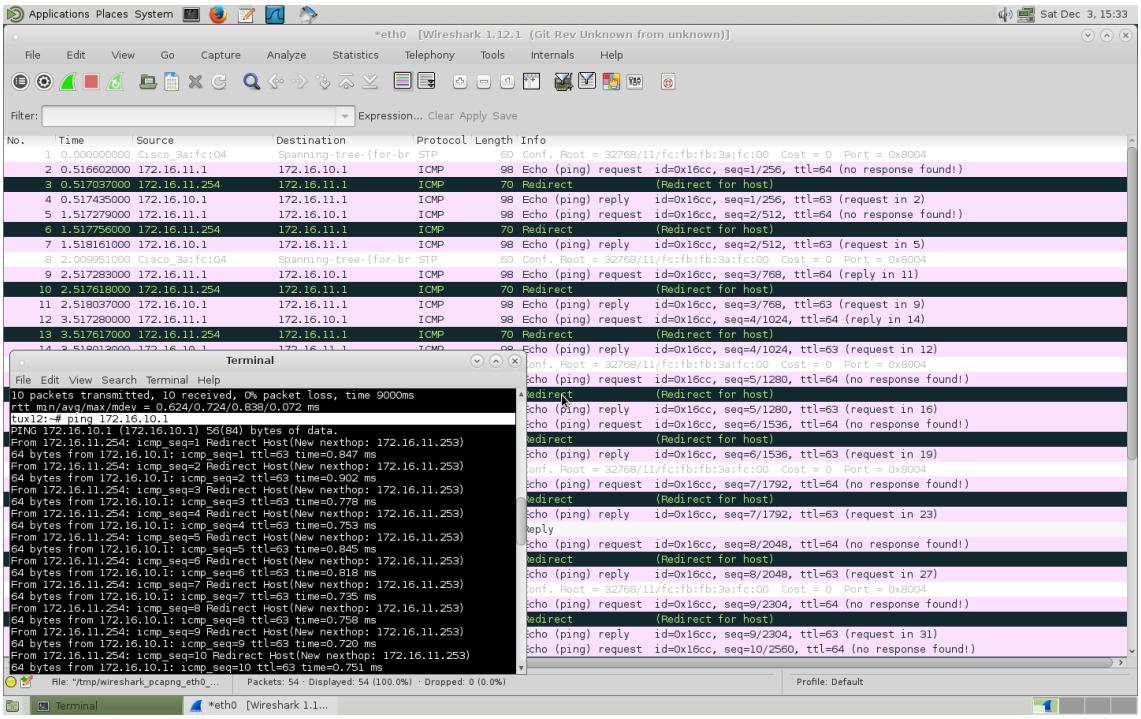


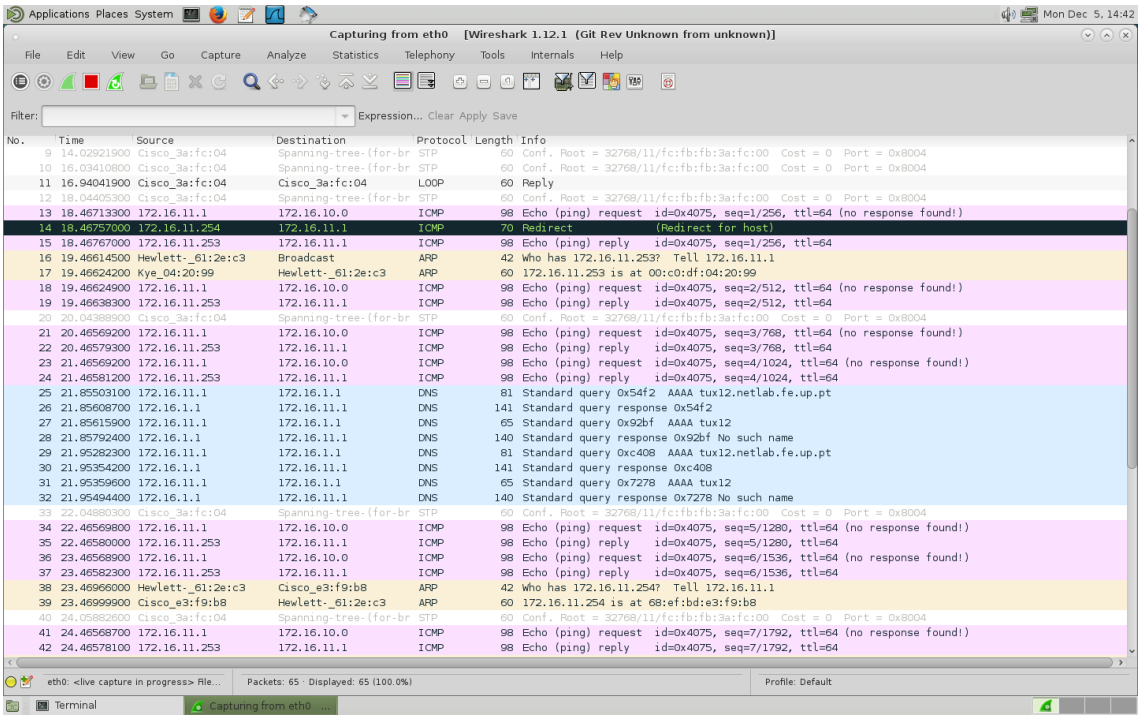
Experiência 3



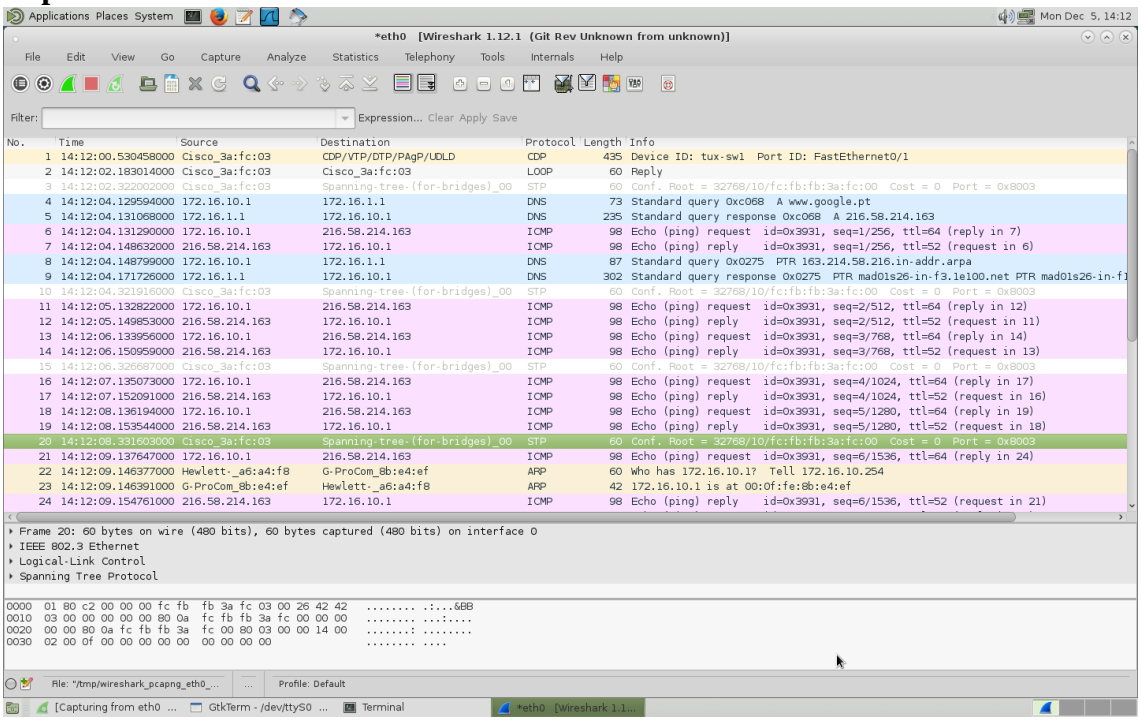
Experiência 4



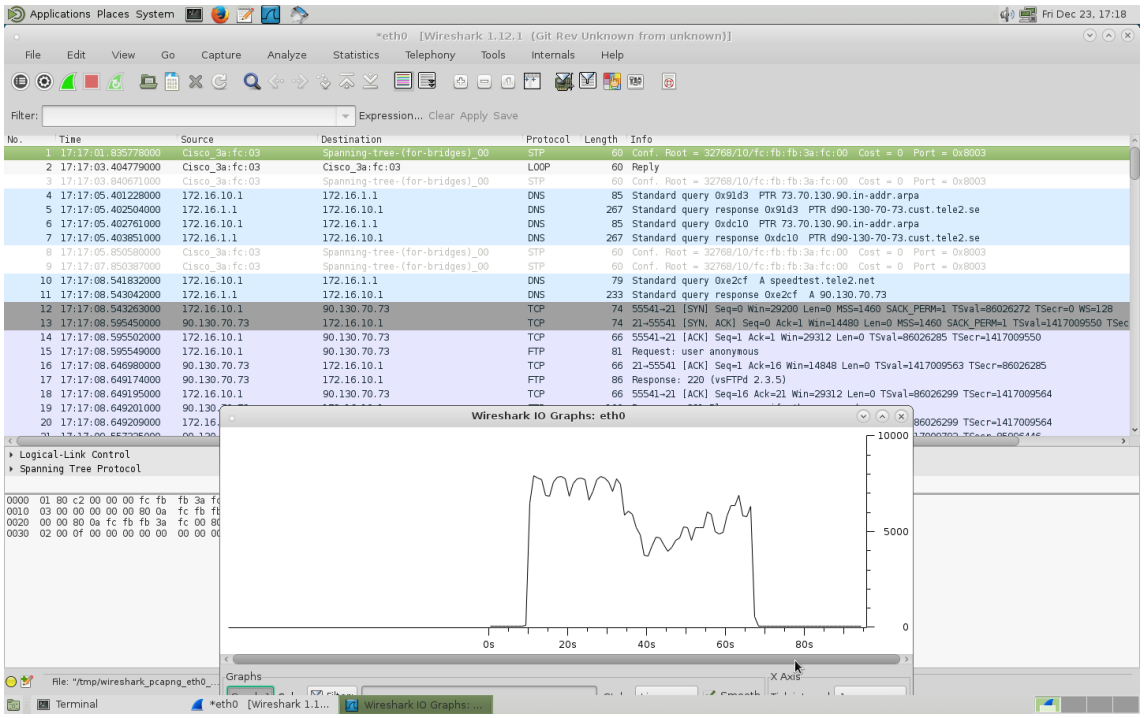


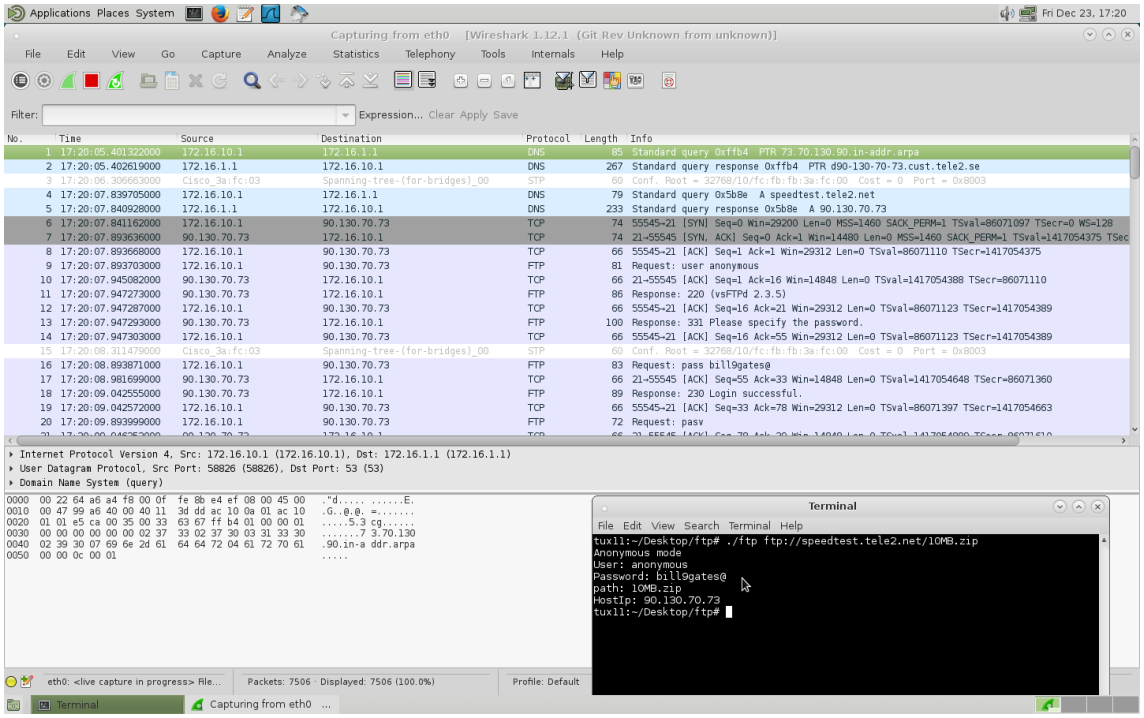


Experiência 5



Experiência 6





I. Comandos

Configuração do Switch

configure terminal

vlan x0

```
 end
 show vlan id x0

configure terminal
 interface fastethernet 0/1
 switchport mode access
 switchport access vlan x0
end
 show running-config interface fastethernet 0/1
 show interfaces fastethernet 0/1 switchport
```

Configuração do Router

Adicionar Rotas

```
configure terminal
router rip
 version 2
 network 192.168.1.1
 network 10.10.7.1
 no auto-summary
end
 show ip route
```

Configurar NAT

```
conf t
 interface gigabitethernet 0/0
ip address 172.16.11.254 255.255.255.0
no shutdown ip nat inside
exit
interface gigabitethernet 0/1
ip address 172.16.1.19 255.255.255.0
```

```
no shutdown
ip nat outside
exit
ip nat pool ovrld 172.16.1.19 172.16.1.19 prefix 24
ip nat inside source list 1 pool ovrld overload
 access-list 1 permit 172.16.10.0 0.0.0.7
access-list 1 permit 172.16.11.0 0.0.0.7
 ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.10.0 255.255.255.0 172.16.11.253 end
```