Relatório de Redes de Computadores
2º Trabalho Laboratorial

Luiz Henrique Mamede Queiroz 202102362
David José Prata Barbedo Magalhães 201907075

Sumário

Esse trabalho laboratorial foi elaborado para o curso de Redes de Computadores e tem como objetivo o desenvolvimento de uma aplicação de download que utiliza o protocolo FTP e a configuração e análise de uma rede de computadores.
Ao realizar esse projeto, os conhecimentos adquiridos nas aulas teóricas do curso puderam ser aplicados para implementar o programa conforme o protocolo mencionado e para configurar a rede conforme requerido.

Introdução

Este projeto foi desenvolvido com o propósito de criar e testar um programa de download que utiliza o protocolo FTP, além de configurar uma rede de computadores conforme as diretrizes estabelecidas no guião fornecido. O relatório está organizado da seguinte forma:

1. Introdução
2. Parte 1 – Aplicação Download
3. Parte 2 – Configuração e Análise de rede
4. Conclusão
5. Referências
6. Anexos

Parte 1- Aplicação Download

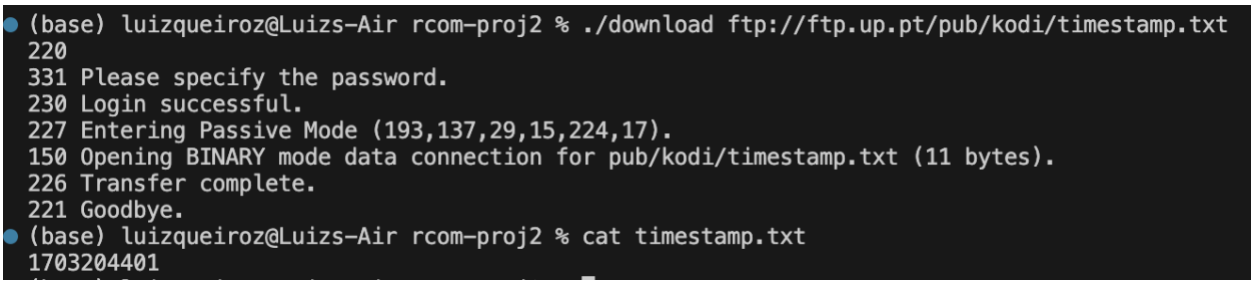
Arquitetura da aplicação Download:

A aplicação é projetada para fazer o download de um ficheiro a partir de um servidor FTP especificado como argumento da função. É suportado o modo passive para a transferência de dados e inclui funções para fazer parse da FTP URL, estabelecer tanto a conexão de controlo quando de dados, fazer a autenticação, entrar em modo passivo, requerer a transferência de ficheiro e lidar com as respostas do servidor.
Estrutura Geral: São utilizados 2 ficheiros, download.h e download.c. No download.h estão definidas as macros para tamanho de respostas e para códigos de resposta do servidor, estão definidas structs para o tipo de conexão (controlo ou dados), para o guardar informação da FTP URL (host, resource, filename, ip, user e password) e para o estado na obtenção da resposta do servidor (start, response para resposta única, resposes para múltiplas respostas e end). Também estão declaradas as funções implementadas no ficheiro download.c. No ficheiro download.c está a implementação das funções.
FTP URL Parsing: A função 'parse' extrai a informação (user, password, host, resource) da FTP URL fornecida como argumento.
Operações com Socket: Funções relacionadas com socket (`getSocket`, `closeSocket`) são responsáveis por criar e encerrar os sockets.
Estabelecimento de Conexão: A aplicação utiliza a funções 'connectToServer' para estabelecer conexão com o servidor através do socket. Ela utiliza 2 conexões. Uma de controlo (sockA) e outra para transferência de dados (sockB). A conexão de controlo é usada para fazer a autenticação no servidor através da função 'authToServer'. A conexão de dados é usada para a transferência dos dados.
Comandos FTP: A função `sendToServer` é responsável por enviar os comandos FTP para o servidor.

Passive Mode: A função que lida com o modo passivo é a 'passiveMode' onde é requerido o modo passivo e é analisada a resposta retornada através da função 'parsePasvResponse' obtendo assim a porta e o server IP para a conexão de dados.
Transferência de Ficheiro: A transferência de ficheiro é feita primeiro com o pedido através da função 'requestFileTransfer' onde o comando requerendo a transferência é enviado ao servidor e depois o ficheiro é obtido com a função 'getFileTrans'.
Tratando Respostas: A função `responseCode` analisa as respostas FTP do servidor e devolve através da variável response no argumento da função como também devolve o código da resposta como retorno da função. Essa função é implementada para lidar com múltiplas respostas também.
A aplicação funciona fazendo o parse da URL enviada como argumento na chamada do programa, obtém o IP com a função 'getIP' que precisa do hostname, cria dois sockets A e B, conecta o socketA ao servidor com o IP obtido utilizando a função 'connectToServer', depois faz a autenticação nesse server com a função 'authToServer', entra em modo passivo com a função 'passiveMode', conecta o socket B com o IP do servidor e porta obtidos na função 'passiveMode utilizando a função 'connectToServer', então ela faz o pedido de transferência de arquivo com a função 'requestFileTransfer' e obtém o arquivo com a função 'getFileTransfer', encerra a conexão do socket A com o servidor com a função 'closeConnectionToServer' e por fim encerra a ligação dos sockets A e B com a função 'closeSocket'.

Usando a aplicação download:

Foram feitos vários testes para verificar se a aplicação estava funcionando corretamente. Em um dos testes foi utilizado a URL: ftp://ftp.up.pt/pub/kodi/timestamp.txt. Neste teste foi possível perceber que a aplicação conseguiu se conectar ao servidor, se autenticar em modo anônimo, obter o modo passivo, requerer a transferência de ficheiro, obter o ficheiro concluindo a transferência e encerrar a conexão com o servidor.
Isso ocorreu conforme mostrado na figura abaixo onde é possível verificar o download bem-sucedido do ficheiro timestamp.txt.



Parte 2 – Configuração e Análise de Rede

Exp 1

1 - What are the commands required to configure this experience?

Os comandos requeridos são o ifconfig ethX up e ifconfig ethX <IP>

2 – What are the ARP packets and what are they used for?

São pacotes de rede que são usados para fazer o mapeamento de um endereço de IP com o endereço MAC dentro de uma rede local.

3 – What are the MAC and IP addresses of ARP packets and why?

Os endereços de IP são o 172.16.10.254 e 172.16.10.1 e os endereços MAC são 00:21:5a:61:2f:24 e 00:21:5a:61:2d:ef, respectivamente.

4 – What packets does the ping command generate?

Gera pacotes do tipo ICMP (Internet Control Message Protocol) para checar o alcance de um host numa rede.

5 – What are the MAC and IP addresses of the ping packets?

São os endereços de MAC e IP das máquinas TUX13 e TUX14, ou seja, IP -> 172.16.10.1 / MAC -> 00:21:5a:61:2f:24 e IP-> 172.16.10.254 / MAC -> 00:21:5a:61:2d:ef.

6 – How to determine if a receiving Ethernet frame is ARP, IP, ICMP?

No wireshark, na coluna protocol é designado o seu tipo. Isso pode ser visto pelo EtherType dentro do cabeçalho do frame para saber se é ARP ou IP e, se necessário, dentro do campo protocol para saber o tipo de protocolo.

7 – How to determine the length of a receiving frame?

No wireshark, na coluna length é possível. E no cabeçalho do frame, existe um campo para o tamanho.

8 – What is the loopback interface and why is it important?

É uma interface de rede virtual que permite que a máquina consiga se comunicar com ela mesma usando um endereço de IP. Ela é importante pois ajuda a validar o funcionamento da rede sem a necessidade de conexões de rede externas.

Exp 2

1 – How to configure bridgeY0?

Depois de adicionar a bridge com o comando /interface brigde add name=bridgeY0, é necessário remover as portas conectadas na bridge default para então adicionar as portas na bridgeY0. Isso é feito com os comandos /interface bridge port remove [find interface=etherX] e /interface bridge port add bridge=bridgeY0 interface=etherX.

2 – How many broadcast domains are there? How can you conclude it from the logs?

Existem 2 domínios de broadcast já que 2 bridges foram criadas. A partir dos logs, é possível ver que ao fazer ping -b 172.16.10.255 no TUX13, o ping do TUX13 obteve resposta do TUX14 mas não obteve

resposta do TUX12. E ao fazer ping -b 172.16.11.255 no TUX12, não se obtem resposta pois o TUX13 e TUX14 estão em outra bridge e logo em outro domínio de broadcast.

Exp 3

1 – What are the commands required to configure this experience?

Configurar eth1 do TUX14:
ifconfig eth1 up
ifconfig eth1 172.16.51.253/24

Adicionar a porta na bridge:
/interface bridge port remove [find interface=etherX]
/interface bridge port add bridge=bridge11 interface=etherX

Enable IP forwarding:
sysctl net.ipv4.ip_forward=1

Disable ICMP echo ignore broadcast:
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0

Reconfigurar TUX13 e TUX12 para que eles possam alcançar um ao outro:

route add -net 172.16.10.0/24 gw 172.16.11.253

route add -net 172.16.11.0/24 gw 172.16.10.254

2 – What routes are there in the tuxes? What are their meaning?

Existe rotas com destino para 172.16.10.0 e para 172.16.11.0. Isso quer dizer que os TUX conseguem se alcançar, a diferença está no caminho que será percorrido, pois o TUX14 está ligado as duas bridges, logo ele funcionará como ponte para o TUX13 alcançar o TUX12 (pelo gateway 172.16.10.254) e vice-versa (pelo gateway 172.16.11.253).

3 – What information does an entry of the forwarding table contain?

Contém o endereço de destino, o gateway, a genmask, flags, métrica, ref, use e interface.

4 – What ARP messages, and associated MAC addresses, are observed and why?

É feito um ping do TUX13 para o TUX12, entretando as ARP messages com os endereços MAC associados são do TUX13 e TUX14 e não do TUX12. Isso acontece por causa da rota feita que faz com que o TUX13 leve para o TUX14 (que é o gateway) e este leva para o TUX12, logo o TUX13 precisa do endereço MAC do TUX14 e não do TUX12.

5 – What ICMP packets are observed and why?

Os pacotes ICMP observados são os que tem como endereço de origem o TUX13 e como endereço de destino o TUX12 indicando que a rede está configurada corretamente.

6 – What are the IP and MAC addresses associated to ICMP packets and why?

O endereço de IP associado aos pacotes ICMP são os endereços de IP do TUX13 (origem) e do TUX12 (destino), entretanto o endereço MAC é do TUX14, pois é a máquina que faz a conexão entre as duas bridges.

Exp 4

1 – How to configure a static route in a commercial router?

Primeiro, se deve adicionar o router a rede interna através da bridge correspondente (como foi feito ao conectar o router a uma porta do Switch e configurar essa porta na bridge desejada), depois configurar o ip do router (tanto o interno quanto o externo) e, por fim, se necessário, adicionar a rota (no caso foi utilizado o comando /ip route add dst-address=? gateway=?)

2 – What are the paths followed by the packets in the experiments carried out and why?

Com a remoção da rota 172.16.10.0/24 pelo Tux14 no Tux12, os pacotes foram reencaminhados para o router, pois é a rota default do Tux12 e o router encaminhou os pacotes para o Tux13 passando pelo Tux14, pois ele tem essa rota.
Ao adicionar a rota 172.16.10.0/24 pelo Tux14 no Tux12, os pacotes foram encaminhados para o Tux14 e em seguida para o Tux13, pois a rota mais curta já estava disponível.

3 – How to configure NAT in a commercial router?

É possível utilizar no terminal do router o comando /ip firewall nat remove 0, para remover a NAT e utilizar o comando /ip firewall nat add chain=srcnat action=masquerade out-interface=ether1 para configurar a NAT. Caso a NAT esteja desabilitada, utilizar o comando /ip firewall nat enable 0 para habilitar a NAT.

4 – What does NAT do?

A NAT (Network Address Translation) é responsável por mapear os endereços de IP da rede privada para um único endereço de IP público. Dessa forma, se for necessário enviar um pacote para a rede externa, isso é feito usando o endereço público como origem. Desse modo, quando a máquina de destino responder, ela enviará a resposta para esse endereço público. Após isso, esse endereço público será traduzido de volta para o endereço privado que enviou o pacote. Assim, o número de endereços públicos utilizados pode ser reduzido.

Exp 5

1 – How to configure the DNS service in a host?

É possível configurar ao editar o ficheiro /etc/resolv.conf dos TUX com: “nameserver <IP>”

2 – What packets are exchanged by DNS and what information is transported?

São pacotes DNS que são trocados (DNS queries e DNS responses). A informação transportada é o mapeamento entre o hostname e o seu endereço de IP.

Exp 6

1 – How many TCP connections are opened by your FTP application?

2 conexões são abertas. Uma para controlo e outra para dados.

2 – In what connection is transported the FTP control information?

Na primeira conexão que é aberta, a conexão de controlo. Com ela é possível a troca de comandos e respostas do servidor.

3 – What are the phases of a TCP connection?

As fases são de estabelecimento de conexão, transferência de dados e encerramento de conexão.

4 – How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?

O ARQ (Automatic Repeat Request) garante de forma confiável e ordenada a entrega de dados usando uma sequência de números, reconhecimentos, retransmissões e controle de fluxo. Primeiro, há a divisão de dados em segmentos que são atribuídos uma sequência de números, depois é feito o reconhecimento para estabelecer a conexão, depois é feita a transmissão dos segmentos de dados e o reconhecimento pelo receptor (o remetente acompanha os seguimentos que foram recebidos com sucesso e retransmite só os segmentos que não foram reconhecidos) e, por fim, a conexão é encerrada. Os campos TCP relevantes são source port, destination port, sequence number, acknowledgment number, data offset e control flags. Pode ser observada informações relativas a perda de pacotes através da sequência e reconhecimento dos números atribuídos aos segmentos de dados, como também de retransmissões e congestionamentos. Também pode ser observado as TCP Flags para saber o estado da conexão.

5 – How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according to the TCP congestion control mechanism?

Ele funciona administrando o fluxo de dados para evitar congestionamento e responder as condições da rede de forma efetiva. Para isso, ele usa mecanismos como slow start, congestion avoidance, fast retransmit, fast recovery e window scaling. Os campos relevantes são congestion window, slow start threshold, duplicate acknowledgments, sequence and acknowledgment number e round-trip time.

6 – Is the throughput of a TCP data connections disturbed by the appearance of a second TCP connection? How?

Sim, pois as conexões TCP compartilham a largura de banda, assim quando há múltiplas conexões, elas podem sofrer redução no throughput.

Conclusão

Neste trabalho, consolidamos os aprendizados com a implementação e teste do programa de download via FTP, assim como com a configuração detalhada da rede de computadores, conforme as orientações do guião.
Assim, foi possível aprender mais sobre protocolos de rede e aprofundar os conhecimentos em protocolos como FTP e TCP. Além disso, devido a parte de configuração de rede do trabalho, foi possível entender como se configura um IP, como se implementa uma bridge num Switch, como se configura um router em linux, como se configura um commercial router, como funciona a NAT e o DNS.

Referências

1. Beej's Guide to Network Programming
2. Mikrotik RouterOS
3. RFC959-FTP
4. RFC1738

Anexos

Anexo 1- Código da Aplicação Download

Ficheiro Download.h

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

#define BUFFER_SIZE 1024
#define MAX_SIZE 512

#define SERVER_FTP_PORT 21
```

```
#define SERVER_DATA_CONNECTION_OPENED   125
#define SERVER_DATA_CONNECTION          150
#define SERVER_WELCOME                  220
#define SERVER_QUIT                     221
#define SERVER_TRANSFER_COMPLETE        226
#define SERVER_PASSIVE_MODE             227
#define SERVER_LOGIN_SUCCESSFUL         230
#define SERVER_PASSWORD                 331

enum Socket_type {
    controlo,
    dados,
};

enum State {
    START,
    RESPONSE,
    RESPONSES,
    END,
};

struct URL {
    char user[MAX_SIZE];
    char password[MAX_SIZE];
    char host[MAX_SIZE];
    char resource[MAX_SIZE];
    char filename[MAX_SIZE];
    char ip[MAX_SIZE];
};


int parse(char *parsing, struct URL *url);

int getSocket();

int closeSocket(int sockfd);
```

```
int closeConnectionToServer(int sockfd);

void handlingServerAddr(struct sockaddr_in *server_addr, char *server_address, int server_port);

int connectToServer(int sockfd, enum Socket_type type, char *server_address, int server_port);

int authToServer(int sockfd, char *user, char *password);

int sendToServer(int sockfd, char *buf);

int passiveMode(int sockfd, char *serverIP, int *port);

int responseCode(int sockfd, char *response);

int parsePasvResponse(char *response, char *serverIP, int *port);

int getHostEntry(char *host, struct hostent **h);

int getIP(char *host, char *ip);

int requestFileTransfer(int sockfd, char* resource);

int getFileTransfer(int sockfd1, int sockfd2, char *filename);
```

Ficheiro Download.c

```
#include "download.h"

int parse(char *parsing, struct URL *url) {
    if (sscanf(parsing, "ftp://%[^:]:%[^@]@%[^/]/%s",
               url->user,
               url->password,
               url->host,
               url->resource
               ) == 4) {
```

```
    } else if (sscanf(parsing, "ftp://%[^/]/%s",
                      url->host,
                      url->resource
                      ) == 2) {
                        strcpy(url->user, "anonymous");
                        strcpy(url->password, "anonymous");
    } else {
        return -1;
    }

    char *last = strrchr(url->resource, '/');
    if (last != NULL) {
        strcpy(url->filename, last + 1);
    } else {
        strcpy(url->filename, url->resource);
    }

    return 0;
}

int getSocket() {
    int sockfd;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        return -1;
    }

    return sockfd;
}

int closeSocket(int sockfd) {
    if (close(sockfd)<0) {
        perror("close()");
        return -1;
    }
```

```
    return 0;
}

int closeConnectionToServer(int sockfd) {
    if (sendToServer(sockfd, "quit\r\n")) return -1;

    char response[BUFFER_SIZE];
    if (responseCode(sockfd, response) != SERVER_QUIT) return -1;

    return 0;
}

void handlingServerAddr(struct sockaddr_in *server_addr, char *server_address, int server_port)  {
    bzero((char *) server_addr, sizeof(*server_addr));
    server_addr->sin_family = AF_INET;
    server_addr->sin_addr.s_addr = inet_addr(server_address);
    server_addr->sin_port = htons(server_port);
}

int connectToServer(int sockfd, enum Socket_type type, char *server_address, int server_port) {
    struct sockaddr_in server_addr;
    memset(&server_addr, 0, sizeof(server_addr));

    handlingServerAddr(&server_addr, server_address, server_port);

    if (connect(sockfd,
                (struct sockaddr *) &server_addr,
                sizeof(server_addr)) < 0) {
        perror("connect()");
        return -1;
    }

    if (type == controlo) {
        char response[BUFFER_SIZE];
        if (responseCode(sockfd, response) != SERVER_WELCOME) return -1;
    }
```

```
    return 0;
}

int authToServer(int sockfd, char *user, char *password) {
    char response[BUFFER_SIZE];
    char userCommand[5+strlen(user)+2];
    sprintf(userCommand, "USER %s\r\n", user);

    if (sendToServer(sockfd, userCommand)) return -1;

    if (responseCode(sockfd, response) != SERVER_PASSWORD) return -1;

    char passwordCommand[5+strlen(password)+2];
    sprintf(passwordCommand, "PASS %s\r\n", password);

    if (sendToServer(sockfd, passwordCommand)) return -1;
    if (responseCode(sockfd, response) != SERVER_LOGIN_SUCCESSFUL) return -1;

    return 0;
}

int sendToServer(int sockfd, char *buf) {
    size_t bytes;
    bytes = write(sockfd, buf, strlen(buf));

    if (bytes <= 0) {
        perror("write()");
        return -1;
    }

    return 0;
}

int passiveMode(int sockfd, char *serverIP, int *port) {
    char response[BUFFER_SIZE];
    if (sendToServer(sockfd, "pasv\r\n")) return -1;
    if (responseCode(sockfd, response) != SERVER_PASSIVE_MODE) return -1;
```

```
    if (parsePasvResponse(response, serverIP, port)) return -1;

    return 0;
}

int responseCode(int sockfd, char *response) {
        char serverResponse[BUFFER_SIZE], responseByte;
        ssize_t responseBytes = 0, byte;
        enum State state = START;

        while (state != END) {
            byte = recv(sockfd, &responseByte, 1, 0);
            if (byte < 0) {
                perror("Error receiving response");
                return -1;
            }
            serverResponse[responseBytes++] = responseByte;

            switch (state) {
                case START:
                    if (responseByte == ' ') {
                        state = RESPONSE;
                    } else if (responseByte == '-') {
                        state = RESPONSES;
                    } else if (responseByte == '\n') {
                        state = END;
                    }

                    break;
                case RESPONSE:
                    if (responseByte == '\n') {
                        state = END;
                    }

                    break;
                case RESPONSES:
                    if (responseByte == '\n') {
```

```
                    state = START;
                    memset(serverResponse, 0, sizeof(serverResponse));
                    responseBytes = 0;
                }

                break;
            case END:
                break;

            default:
                return -1;
        }
    }

    serverResponse[responseBytes] = '\0';
    strcpy(response, serverResponse);
    printf("%s", response);


  int responseCode;
  sscanf(response, "%3d", &responseCode);

  return responseCode;
}

int parsePasvResponse(char *response, char *serverIP, int *port) {
  unsigned int ip1, ip2, ip3, ip4, port1, port2;

  sscanf(response, "227 Entering Passive Mode (%u,%u,%u,%u,%u,%u).",
         &ip1, &ip2, &ip3, &ip4, &port1, &port2);

  sprintf(serverIP, "%u.%u.%u.%u", ip1, ip2, ip3, ip4);
  *port = port1*256 + port2;

  return 0;
}
```

```
int getHostEntry(char *host, struct hostent **h) {
    if ((*h = gethostbyname(host)) == NULL) {
        herror("gethostbyname()");
        return -1;
    }

    return 0;
}

int getIP(char *host, char *ip) {
    struct hostent *h = NULL;

    if (getHostEntry(host, &h)) return -1;

    strcpy(ip, inet_ntoa(*(struct in_addr *) h->h_addr));

    return 0;
}

int requestFileTransfer(int sockfd, char* resource) {
    char response[BUFFER_SIZE];
    char FileTransferCommand[5+strlen(resource)+2];
    sprintf(FileTransferCommand, "retr %s\r\n", resource);

    if (sendToServer(sockfd, FileTransferCommand)) return -1;

    int code = responseCode(sockfd, response);
    if (code != SERVER_DATA_CONNECTION && code != SERVER_DATA_CONNECTION_OPENED) return -1;

    return 0;
}

int getFileTransfer(int sockfd1, int sockfd2, char *filename) {
    char response[BUFFER_SIZE], data[BUFFER_SIZE];
    ssize_t bytes;

    FILE *fd = fopen(filename, "wb");
```

```
    if (fd == NULL) return -1;

    while ((bytes = recv(sockfd2, data, sizeof(data), 0)) > 0) {
        if (fwrite(data, 1, bytes, fd) != bytes) return -1;
    }

    if (bytes < 0) return -1;

    fclose(fd);

    if (responseCode(sockfd1, response) != SERVER_TRANSFER_COMPLETE) return -1;

    return 0;
}

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

    int sockA, sockB;
    struct URL url;
    memset(&url, 0, sizeof(url));

    if (parse(argv[1], &url)) {
        fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
        exit(-1);
    }

    if (getIP(url.host, url.ip)) {
        fprintf(stderr, "Erro ao obter o IP");
        exit(-1);
    }

    if ((sockA = getSocket()) == -1) {
        fprintf(stderr, "Erro ao obter o socket A");
```

```
    exit(-1);
}

if ((sockB = getSocket()) == -1) {
    fprintf(stderr, "Erro ao obter o socket B");
    exit(-1);
}

if (connectToServer(sockA, controlo, url.ip, SERVER_FTP_PORT)) {
    fprintf(stderr, "Erro ao conectar com o servidor A");
    exit(-1);
}

if (authToServer(sockA, url.user, url.password)) {
    fprintf(stderr, "Erro ao autenticar no servidor");
    exit(-1);
}

char serverIP[MAX_SIZE];
int port;
if (passiveMode(sockA, serverIP, &port)) {
    fprintf(stderr, "Erro ao entrar em modo passivo");
    exit(-1);
}

if (connectToServer(sockB, dados, serverIP, port)) {
    fprintf(stderr, "Erro ao conectar no servidor B");
    exit(-1);
}

if (requestFileTransfer(sockA, url.resource)) {
    fprintf(stderr, "Erro ao pedir a transferencia do ficheiro");
    exit(-1);
}

if (getFileTransfer(sockA, sockB, url.filename)) {
    fprintf(stderr, "Erro ao transferir o ficheiro");
```

```
        exit(-1);
    }

    if (closeConnectionToServer(sockA)) {
        fprintf(stderr, "Erro ao encerrar a conexão com o servidor");
        exit(-1);
    }

    if (closeSocket(sockA)) {
        fprintf(stderr, "Erro ao encerrar o socket A");
        exit(-1);
    }

    if (closeSocket(sockB)) {
        fprintf(stderr, "Erro ao encerrar o socket B");
        exit(-1);
    }

    return 0;
}
```

Anexo 2 – Comandos de Configuração

Exp 1

No console do Switch:
/system reset-configuration

Tux13:
ifconfig eth0 up
ifconfig eth0 172.16.10.1/24

Tux14:
ifconfig eth0 up
ifconfig eth0 172.16.10.254/24

Tux13 e Tux14:
ifconfig

Tux13:
ping 172.16.10.254

Tux14:
ping 172.16.10.1

Tux13 e Tux14:
route – n
arp -a

Tux13:
arp -d 172.16.10.254/24

Tux13:
ping 172.16.10.254

Exp 2

Tux12:
ifconfig eth0 up
ifconfig eth0 172.16.11.1/24

Tux12:
No console do Switch:
/interface bridge add name=bridge10
/interface bridge add name=bridge11
/interface bridge port remove [find interface=ether2]
/interface bridge port remove [find interface=ether8]
/interface bridge port remove [find interface=ether24]
/interface bridge port add bridge=bridge10 interface=ether2
/interface bridge port add bridge=bridge10 interface=ether24
/interface bridge port add bridge=bridge11 interface=ether8

Tux13:
ping 172.16.10.254
ping 172.16.11.1
ping -b 172.16.10.255

Tux12:
ping -b 172.16.11.255

Exp 3

Tux14:
ifconfig eth1 up
ifconfig eth1 172.16.11.253/24

No console do Switch:
/interface bridge port remove [find interface=ether12]
/interface bridge port add bridge=bridge11 interface=ether12

Tux14:
sysctl net.ipv4.ip_forward=1
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0

Tux14:
ifconfig

Tux12:
route add -net 172.16.10.0/24 gw 172.16.11.253

Tux13:
route add -net 172.16.11.0/24 gw 172.16.10.254

Tux13, Tux14, Tux12:
route -n

Tux13:
ping 172.16.10.254
ping 172.16.11.253
ping 172.16.11.1

Tux12:
arp -d 172.16.11.253

Tux13:
arp -d 172.16.10.254

Tux14:
arp -d 172.16.10.1
arp -d 172.16.11.1

Tux13:
ping 172.16.11.1

Exp 4

No console do Switch:
/interface bridge port remove [find interface=ether17]
/interface bridge port add bridge=bridge11 interface=ether17

No console do Router:
/system reset-configuration

/ip address add address=172.16.1.19/24 interface=ether1
/ip address add address=172.16.11.254/24 interface=ether2

Tux12:
route add default gw 172.16.11.254

Tux13:
route add default gw 172.16.10.254

Tux14:
route add default gw 172.16.11.254

No console do Router:
/ip route add dst-address=172.16.10.0/24 gateway=172.16.11.253
/ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254

Tux13:
ping 172.16.10.254
ping 172.16.11.1
ping 172.16.11.254

Tux12:
sysctl net.ipv4.conf.eth0.accept_redirects=0
sysctl net.ipv4.conf.all.accept_redirects=0
route del -net 172.16.10.0/24 gw 172.16.11.253
ping 172.16.10.1
traceroute -n 172.16.10.1
route add -net 172.16.10.0/24 gw 172.16.11.253
traceroute -n 172.16.10.1
sysctl net.ipv4.conf.eth0.accept_redirects=1
sysctl net.ipv4.conf.all.accept_redirects=1

Tux13:
ping 172.16.1.254

No console do Router:
/ip firewall nat disable 0

Tux13:
ping 172.16.1.254

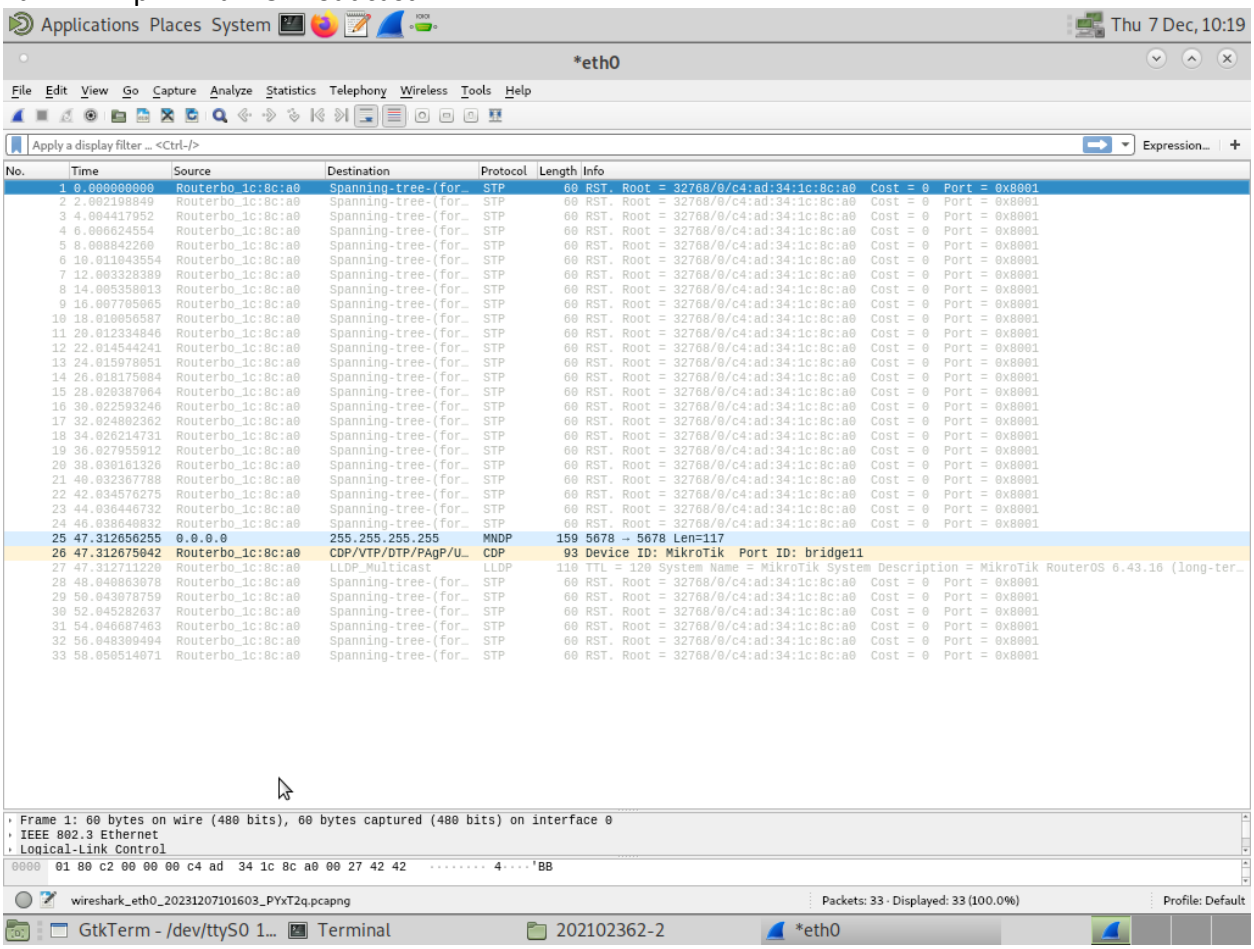
Router console:
/ip firewall nat enable 0

Exp 5

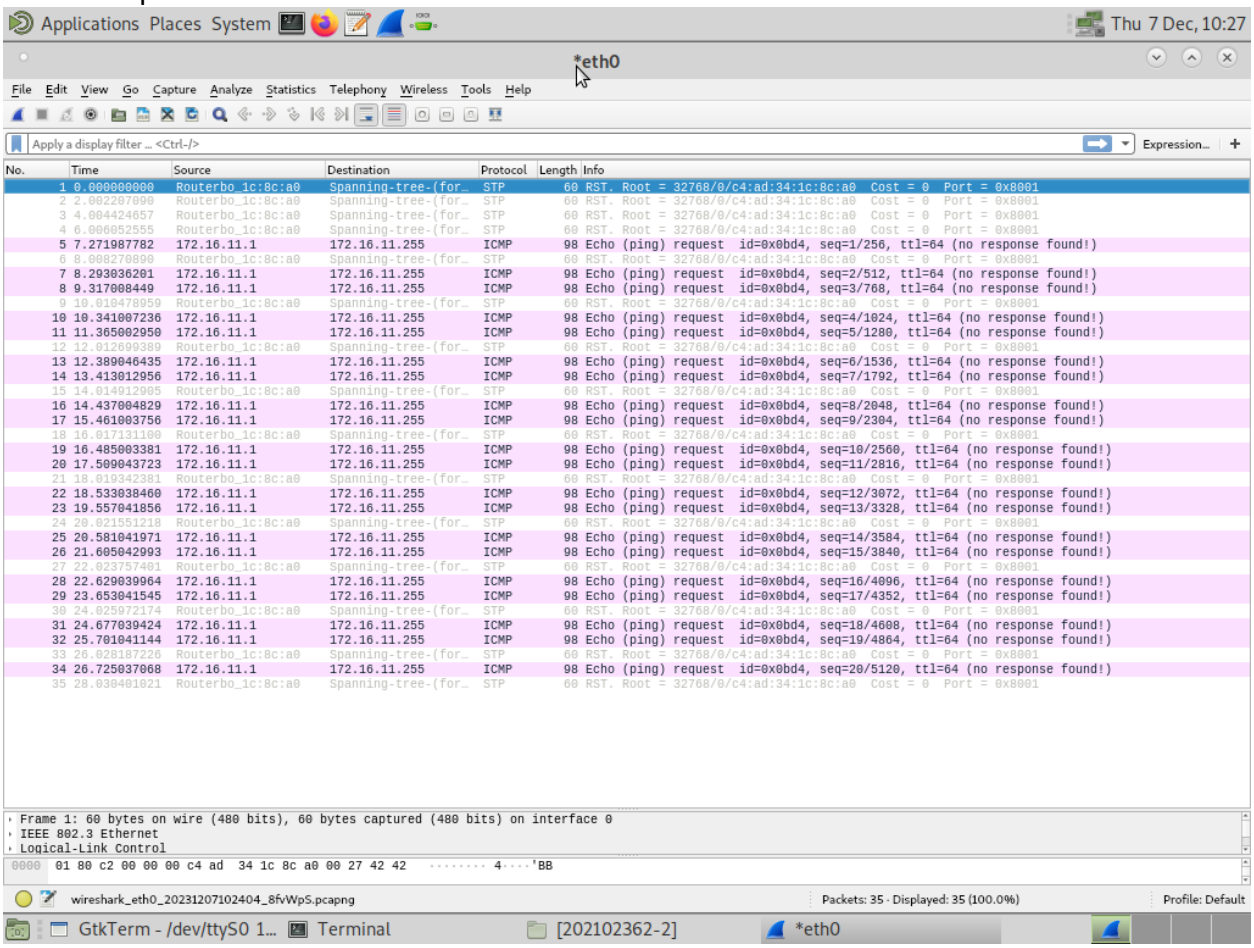
Tux12, Tux13, Tux14:
ping google.com

Anexo 3 – Logs Capturados

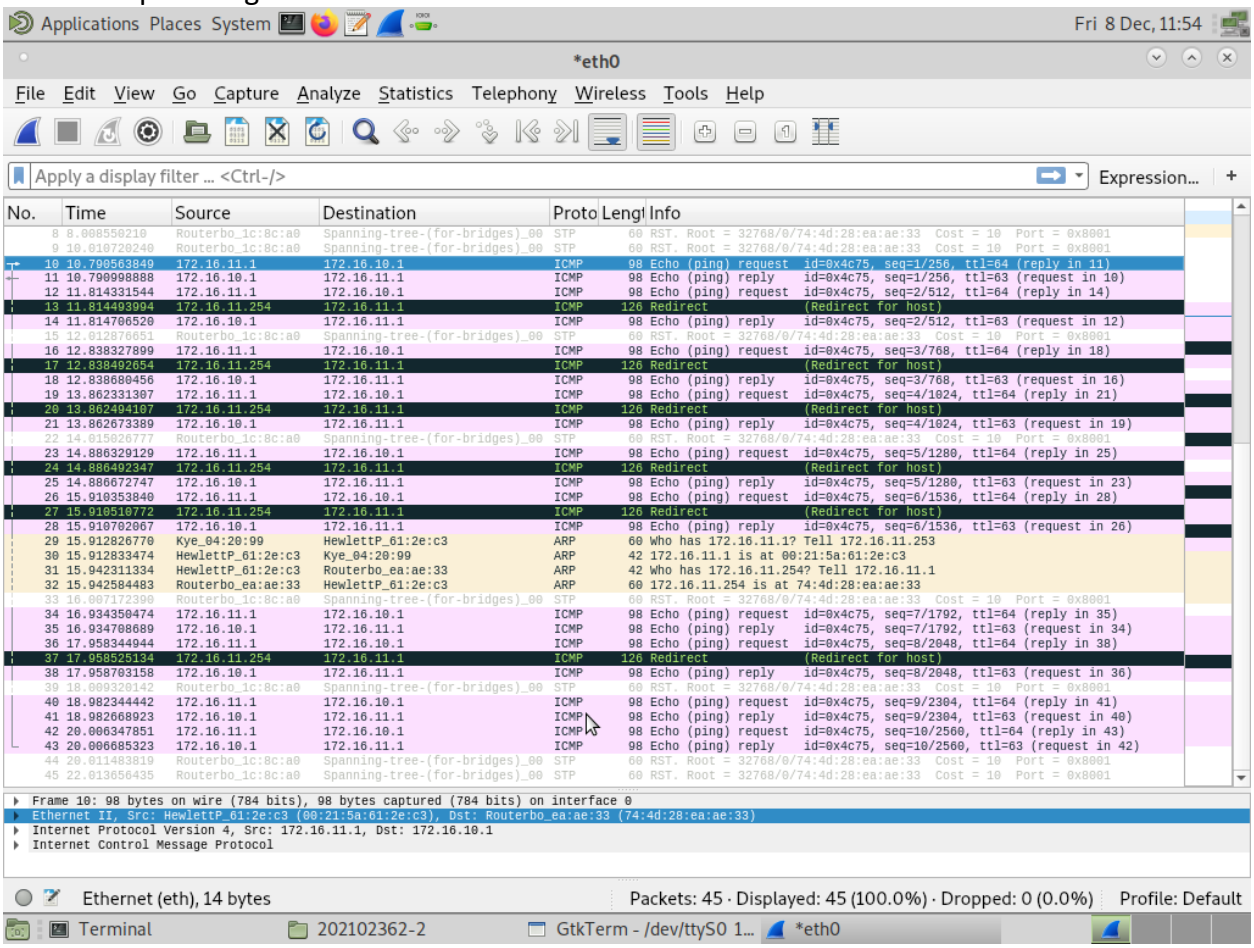
Tux12 - Exp 2 – Tux13 Broadcast



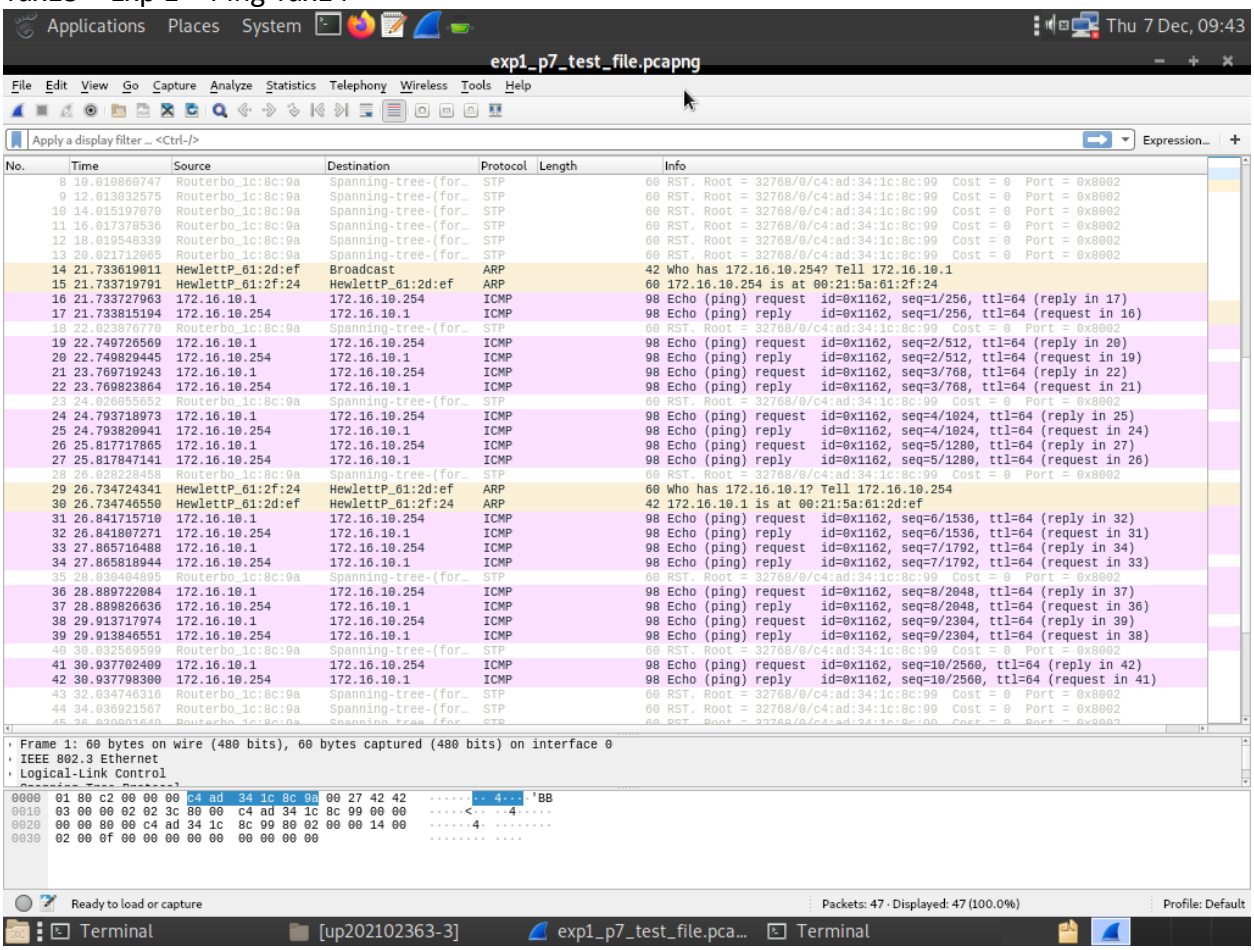
Tux12 – Exp 2 – Tux12 Broadcast



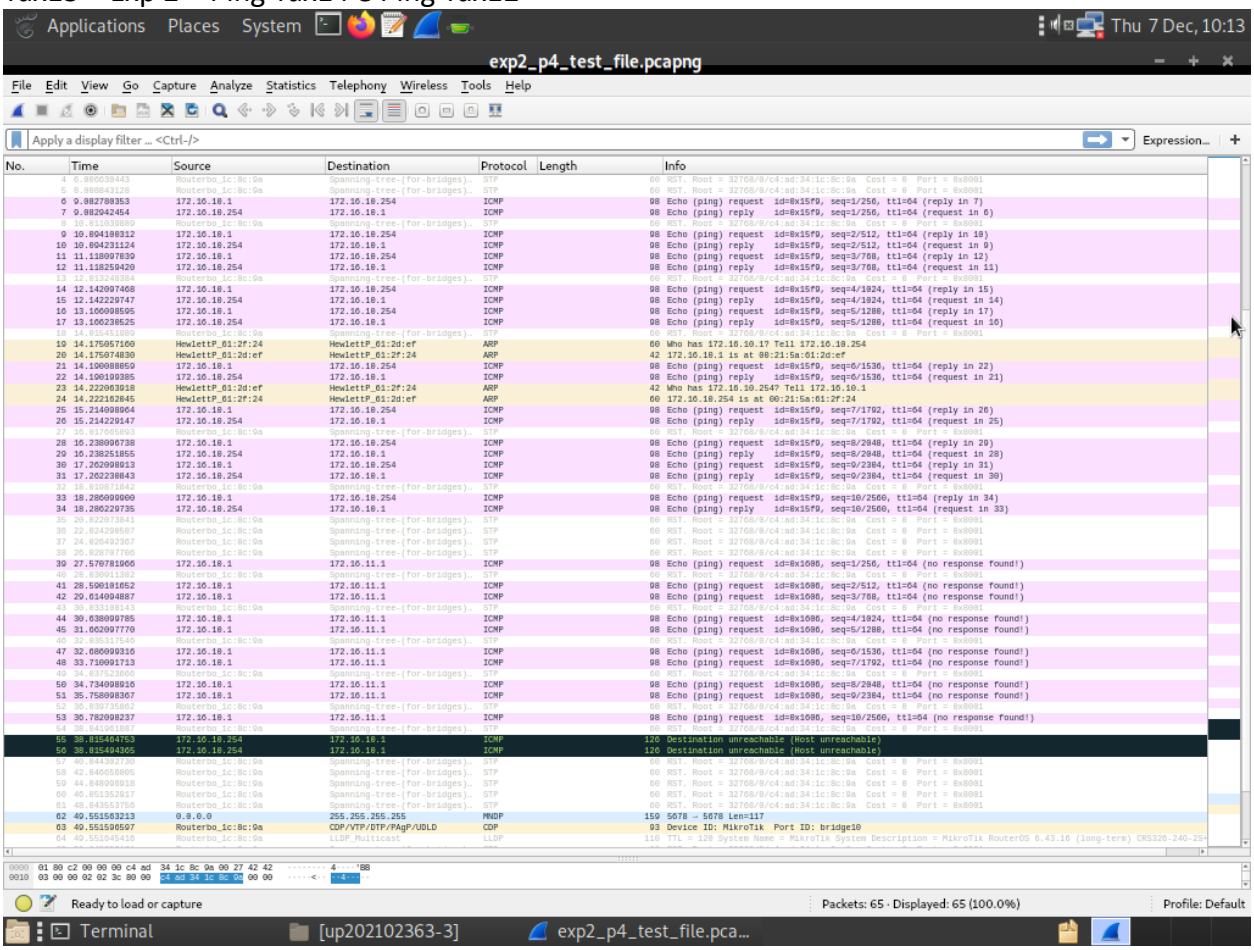
Tux12 – Exp 4 - Ping Tux13



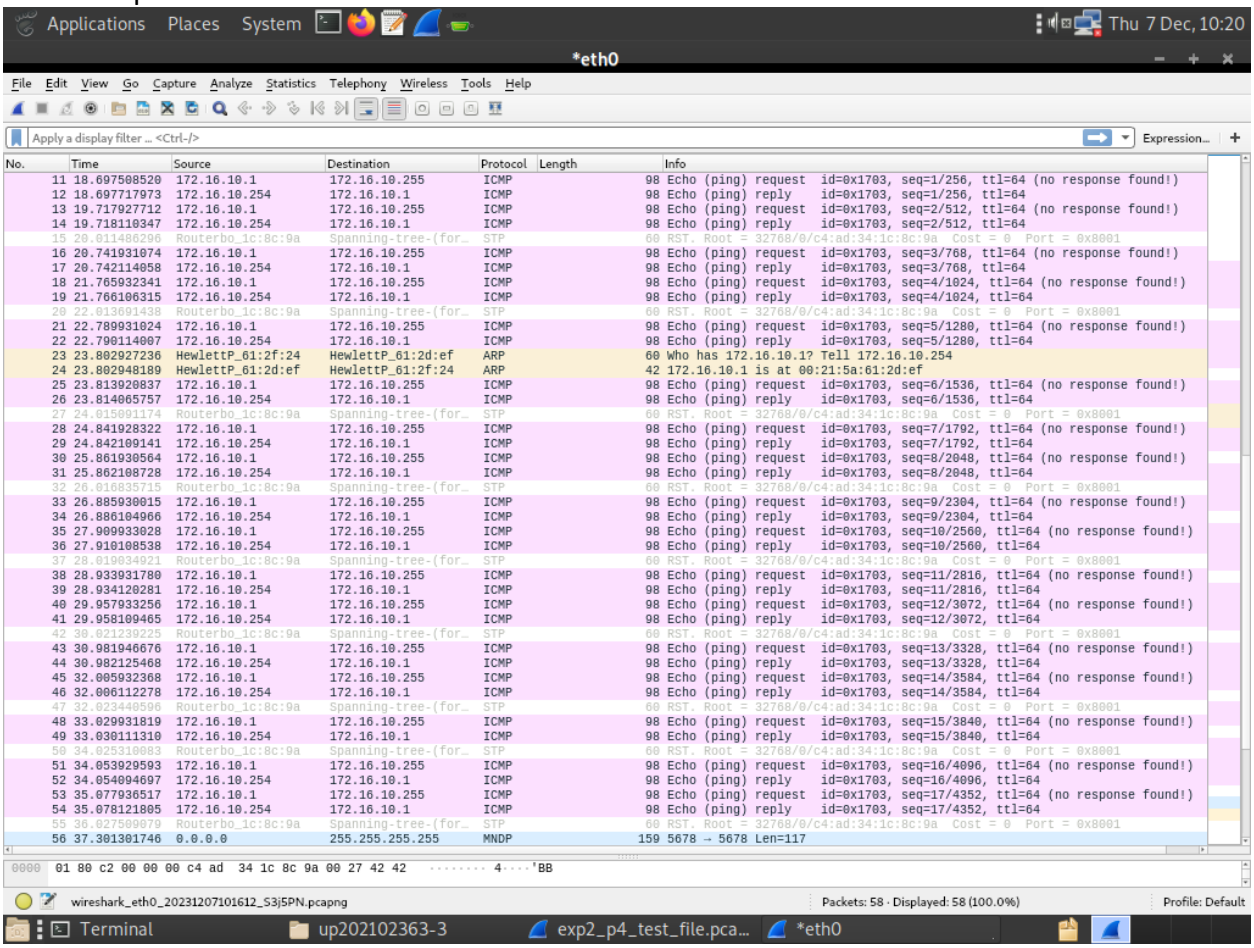
Tux13 – Exp 1 – Ping Tux14



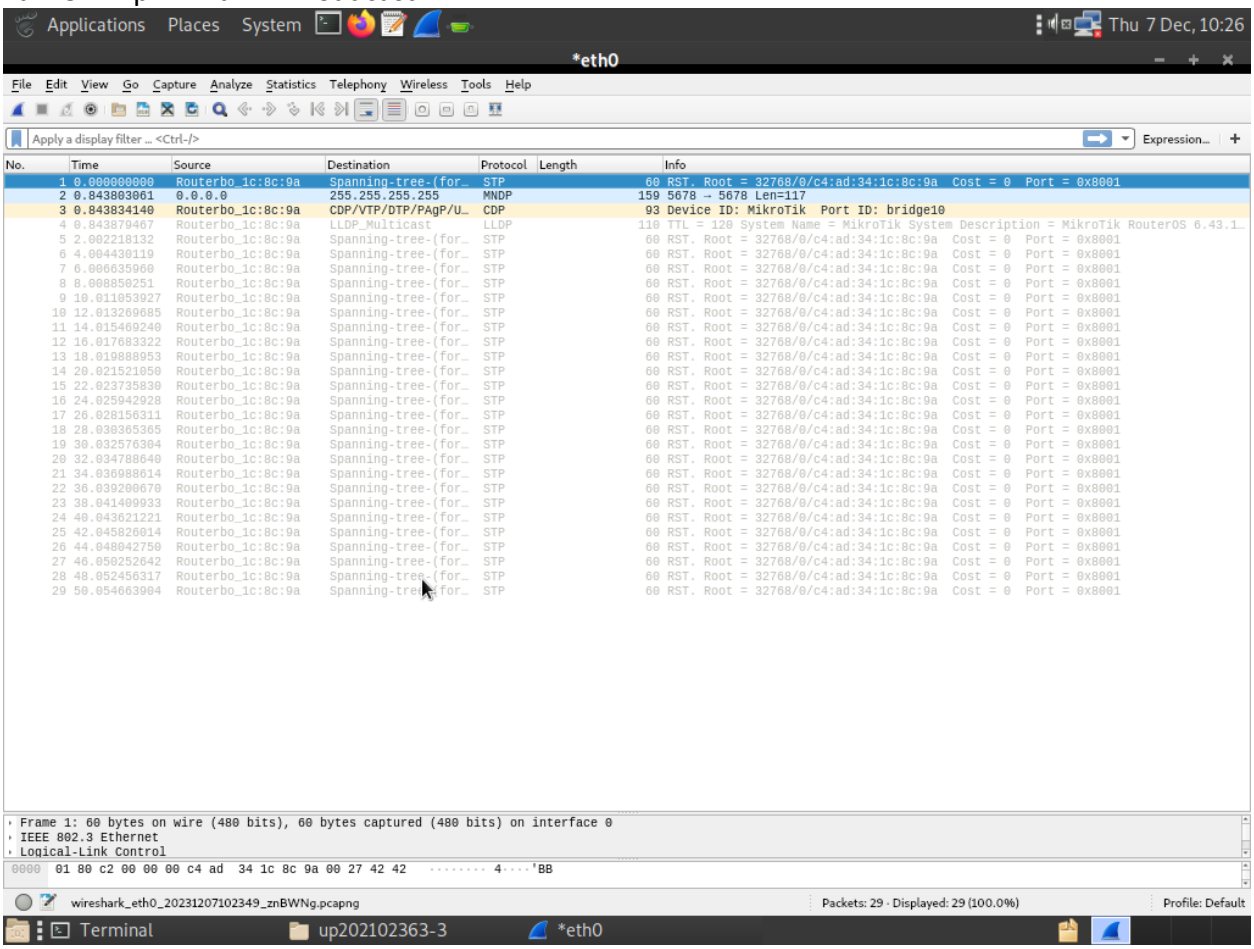
Tux13 – Exp 2 – Ping Tux14 e Ping Tux12



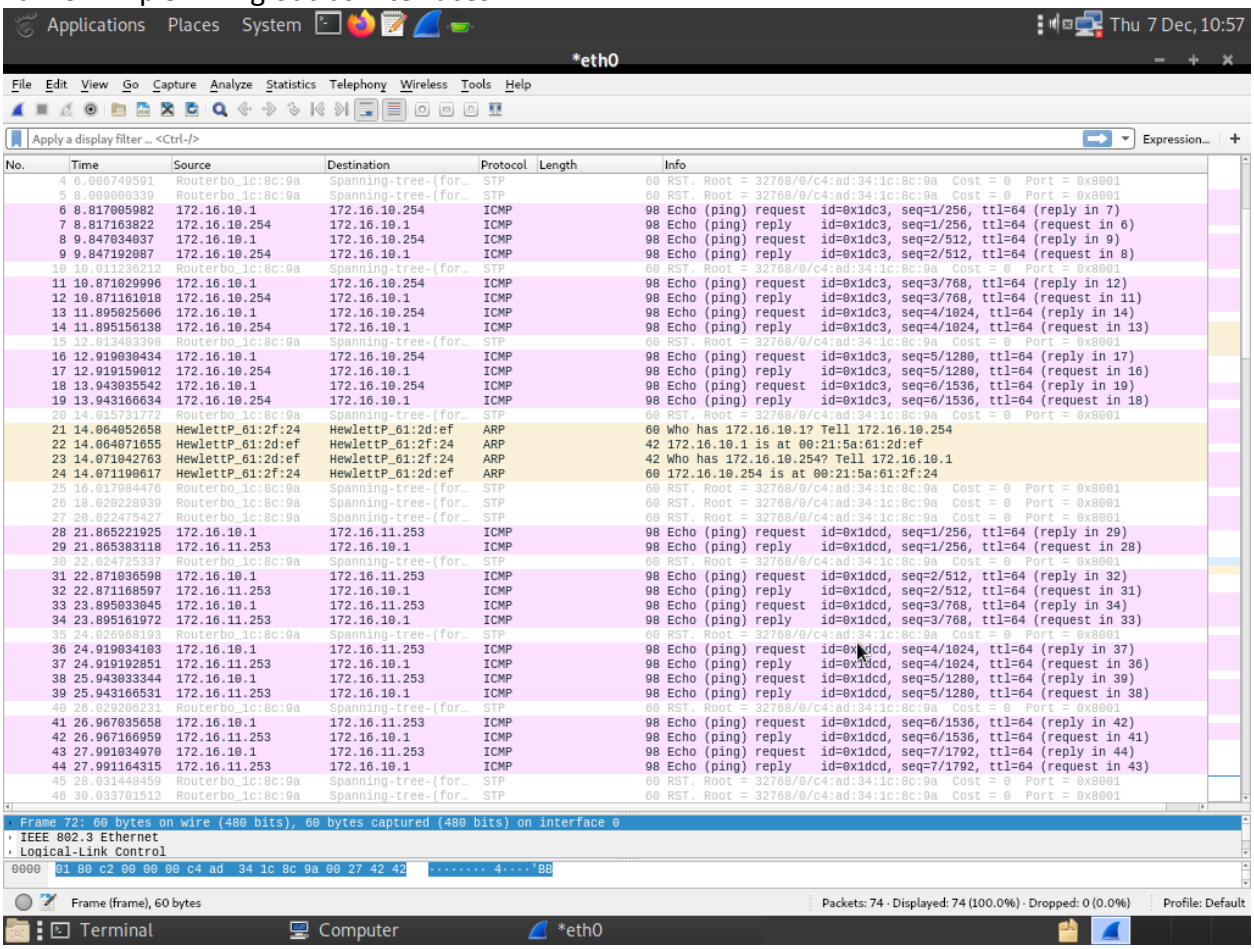
Tux13 - Exp 2 – Tux13 Broadcast

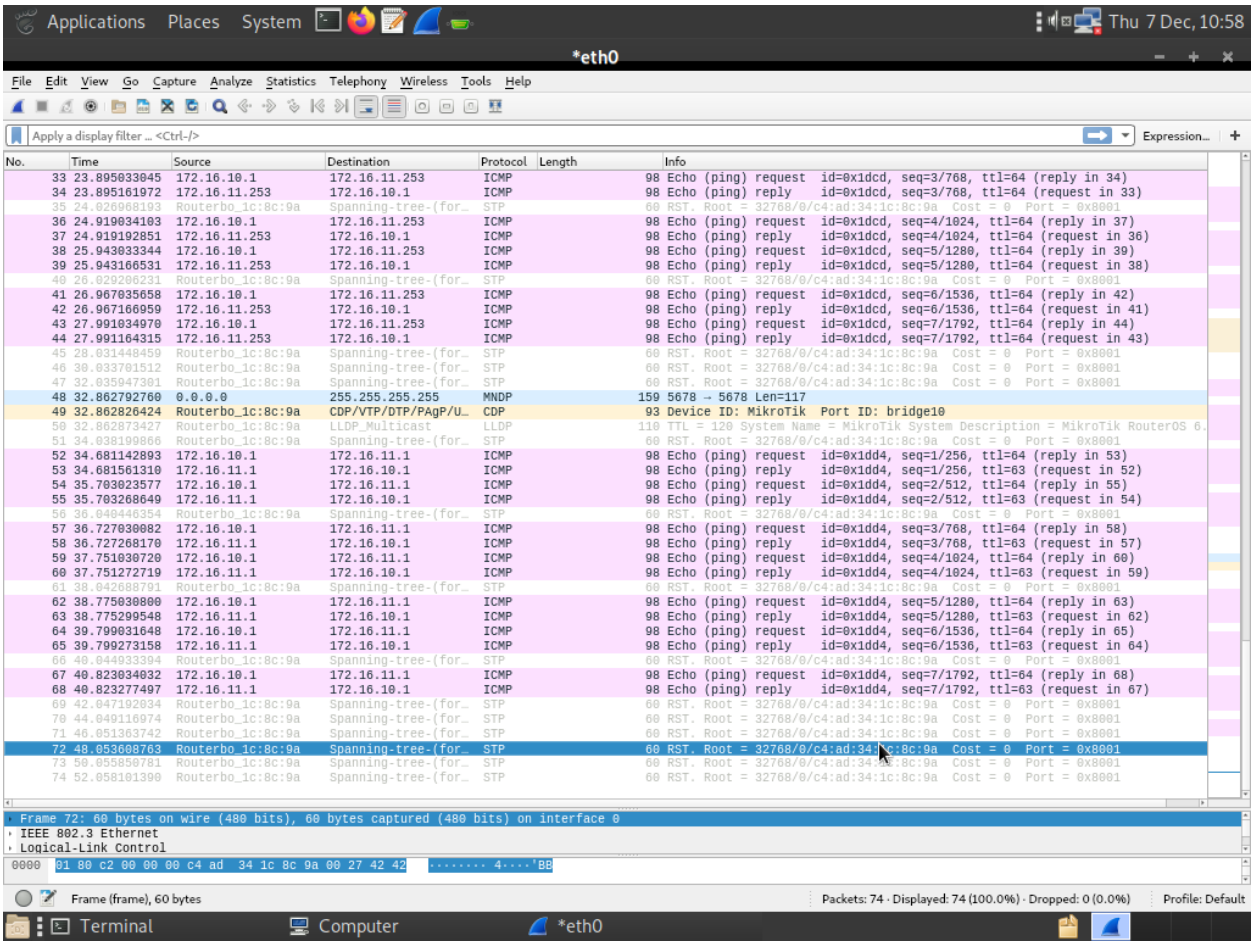


Tux13 - Exp 2 – Tux12 Broadcast

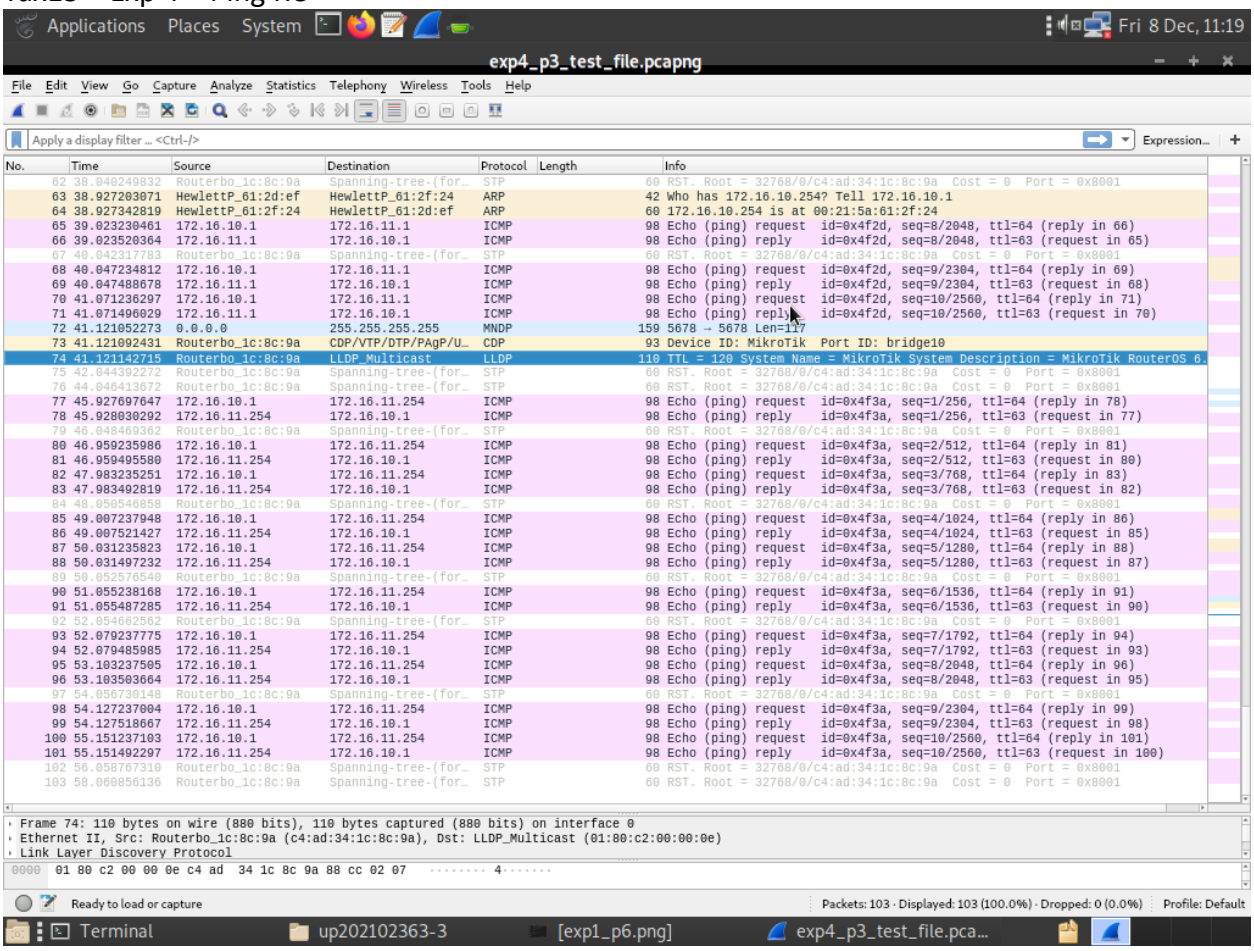


Tux13 – Exp 3 – Ping outras interfaces

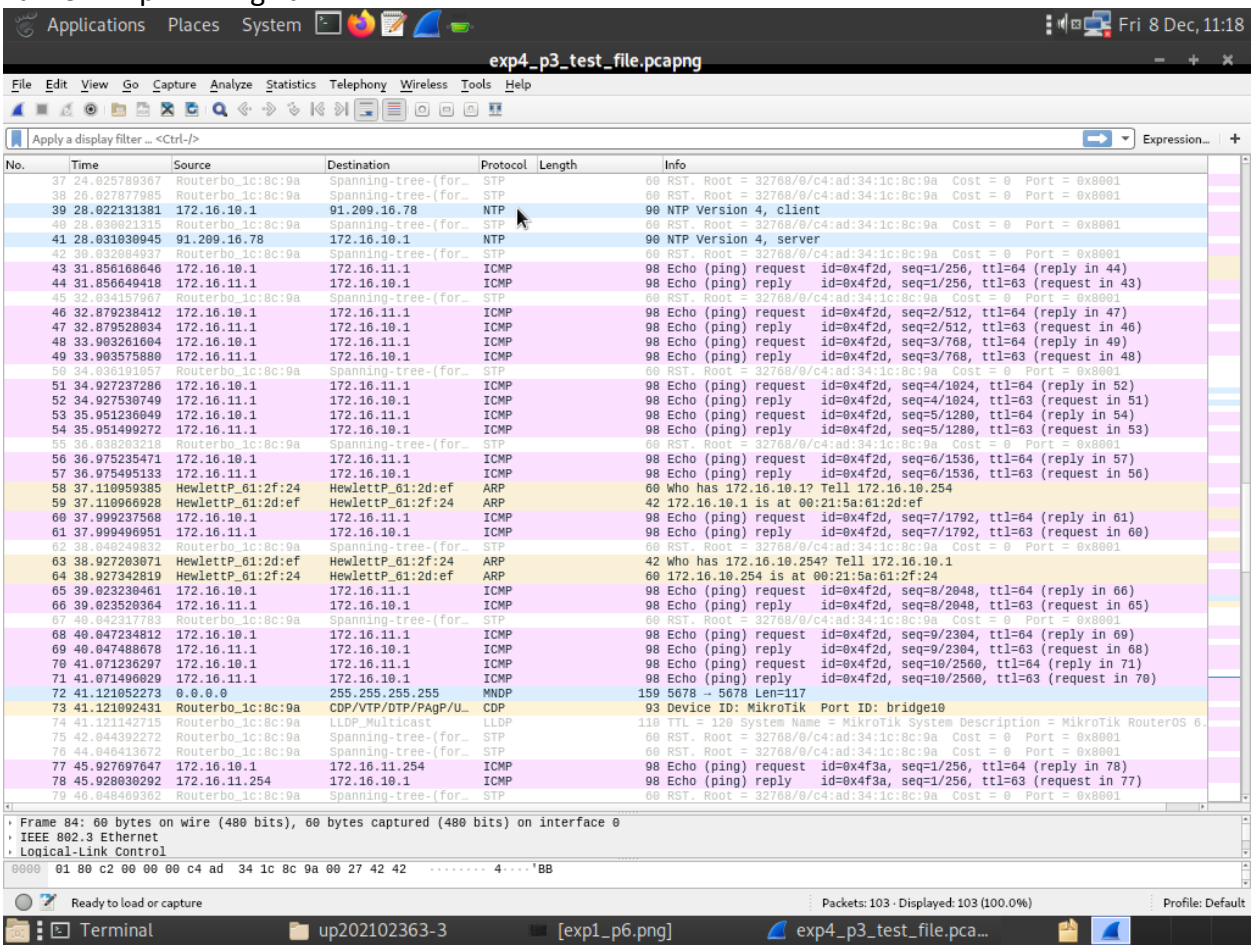



Applications Places System
Thu 7 Dec, 10:58
*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
33 23.895033045 172.16.10.1 172.16.11.253 ICMP 98 Echo (ping) request id=0x1dcd, seq=3/768, ttl=64 (reply in 34)
34 23.895161972 172.16.11.253 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dcd, seq=3/768, ttl=64 (request in 33)
35 24.026968193 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
36 24.919034103 172.16.10.1 172.16.11.253 ICMP 98 Echo (ping) request id=0x1dcd, seq=4/1024, ttl=64 (reply in 37)
37 24.919192851 172.16.11.253 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dcd, seq=4/1024, ttl=64 (request in 36)
38 25.943033344 172.16.10.1 172.16.11.253 ICMP 98 Echo (ping) request id=0x1dcd, seq=5/1280, ttl=64 (reply in 39)
39 25.943166531 172.16.11.253 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dcd, seq=5/1280, ttl=64 (request in 38)
40 26.029206231 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
41 26.967035658 172.16.10.1 172.16.11.253 ICMP 98 Echo (ping) request id=0x1dcd, seq=6/1536, ttl=64 (reply in 42)
42 26.967166959 172.16.11.253 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dcd, seq=6/1536, ttl=64 (request in 41)
43 27.991034970 172.16.10.1 172.16.11.253 ICMP 98 Echo (ping) request id=0x1dcd, seq=7/1792, ttl=64 (reply in 44)
44 27.991164315 172.16.11.253 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dcd, seq=7/1792, ttl=64 (request in 43)
45 28.031448459 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
46 30.033701512 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
47 32.035947301 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
48 32.862792760 0.0.0.0 255.255.255.255 MNDP 159 5678 → 5678 Len=117
49 32.862826424 Routerbo_1c:8c:9a CDP/VTP/DTP/PAgP/U… CDP 93 Device ID: MikroTik Port ID: bridge10
50 32.862873427 Routerbo_1c:8c:9a LLDP_Multicast LLDP 110 TTL = 120 System Name = MikroTik System Description = MikroTik RouterOS 6.
51 34.038199866 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
52 34.681142893 172.16.10.1 172.16.11.1 ICMP 98 Echo (ping) request id=0x1dd4, seq=1/256, ttl=64 (reply in 53)
53 34.681561310 172.16.11.1 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dd4, seq=1/256, ttl=63 (request in 52)
54 35.703023577 172.16.10.1 172.16.11.1 ICMP 98 Echo (ping) request id=0x1dd4, seq=2/512, ttl=64 (reply in 55)
55 35.703268649 172.16.11.1 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dd4, seq=2/512, ttl=63 (request in 54)
56 36.048446354 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
57 36.727030082 172.16.10.1 172.16.11.1 ICMP 98 Echo (ping) request id=0x1dd4, seq=3/768, ttl=64 (reply in 58)
58 36.727268170 172.16.11.1 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dd4, seq=3/768, ttl=63 (request in 57)
59 37.751030720 172.16.10.1 172.16.11.1 ICMP 98 Echo (ping) request id=0x1dd4, seq=4/1024, ttl=64 (reply in 60)
60 37.751272719 172.16.11.1 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dd4, seq=4/1024, ttl=63 (request in 59)
61 38.042688791 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
62 38.775030800 172.16.10.1 172.16.11.1 ICMP 98 Echo (ping) request id=0x1dd4, seq=5/1280, ttl=64 (reply in 63)
63 38.775299548 172.16.11.1 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dd4, seq=5/1280, ttl=63 (request in 62)
64 39.799031648 172.16.10.1 172.16.11.1 ICMP 98 Echo (ping) request id=0x1dd4, seq=6/1536, ttl=64 (reply in 65)
65 39.799273158 172.16.11.1 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dd4, seq=6/1536, ttl=63 (request in 64)
66 40.044933394 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
67 40.823034032 172.16.10.1 172.16.11.1 ICMP 98 Echo (ping) request id=0x1dd4, seq=7/1792, ttl=64 (reply in 68)
68 40.823277497 172.16.11.1 172.16.10.1 ICMP 98 Echo (ping) reply id=0x1dd4, seq=7/1792, ttl=63 (request in 67)
69 42.047192034 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
70 44.049116974 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
71 46.051363742 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
72 48.053608763 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
73 50.055850781 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
74 52.058101390 Routerbo_1c:8c:9a Spanning-tree-(for… STP 60 RST. Root = 32768/0/c4:ad:34:1c:8c:9a Cost = 0 Port = 0x8001
Frame 72: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
IEEE 802.3 Ethernet
Logical-Link Control
0000 01 80 c2 00 00 00 c4 ad 34 1c 8c 9a 00 27 42 42 .... 4....'BB
Frame (frame), 60 bytes
Packets: 74 · Displayed: 74 (100.0%) · Dropped: 0 (0.0%)
Profile: Default
Terminal
Computer
*eth0

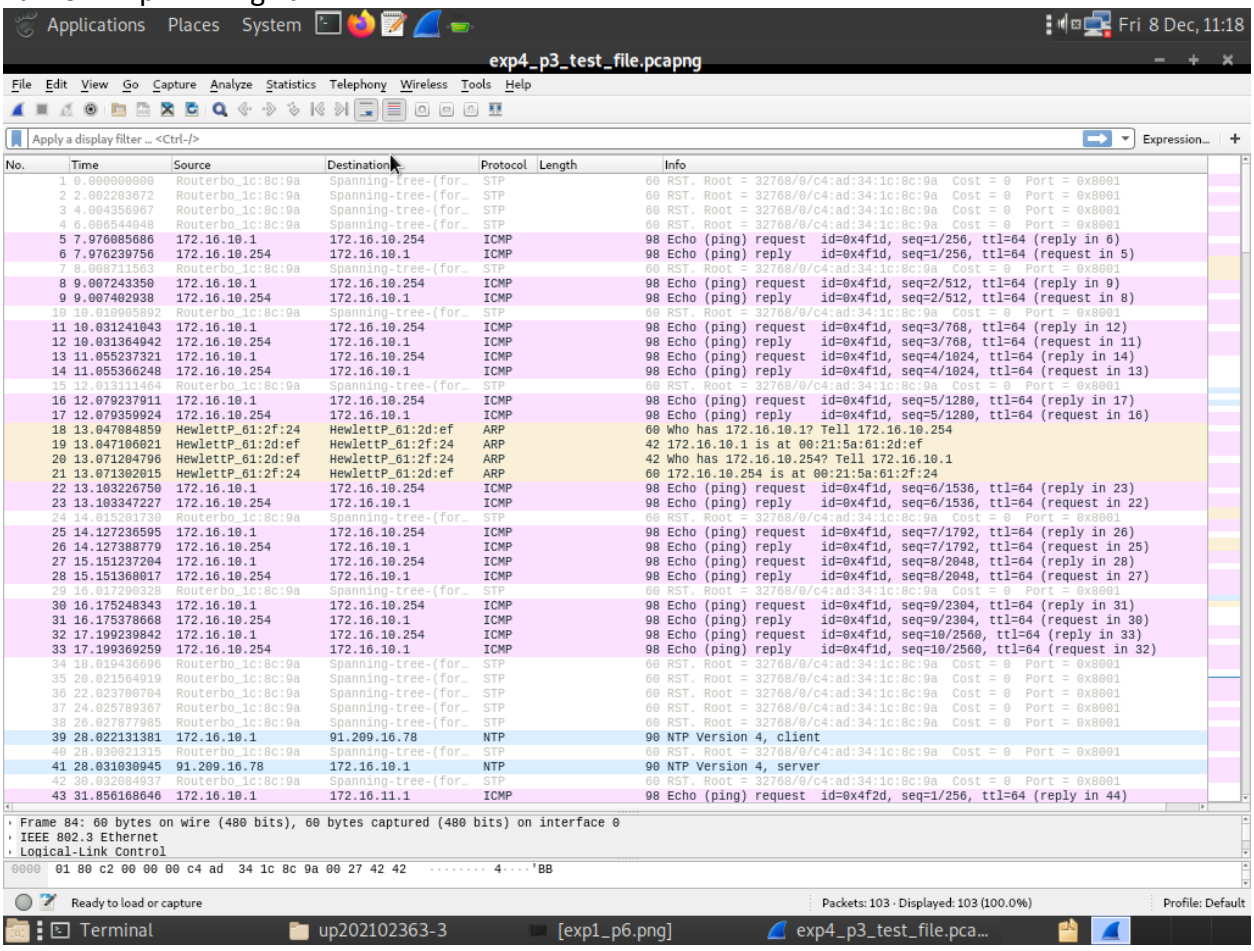
Tux13 – Exp 4 – Ping RC



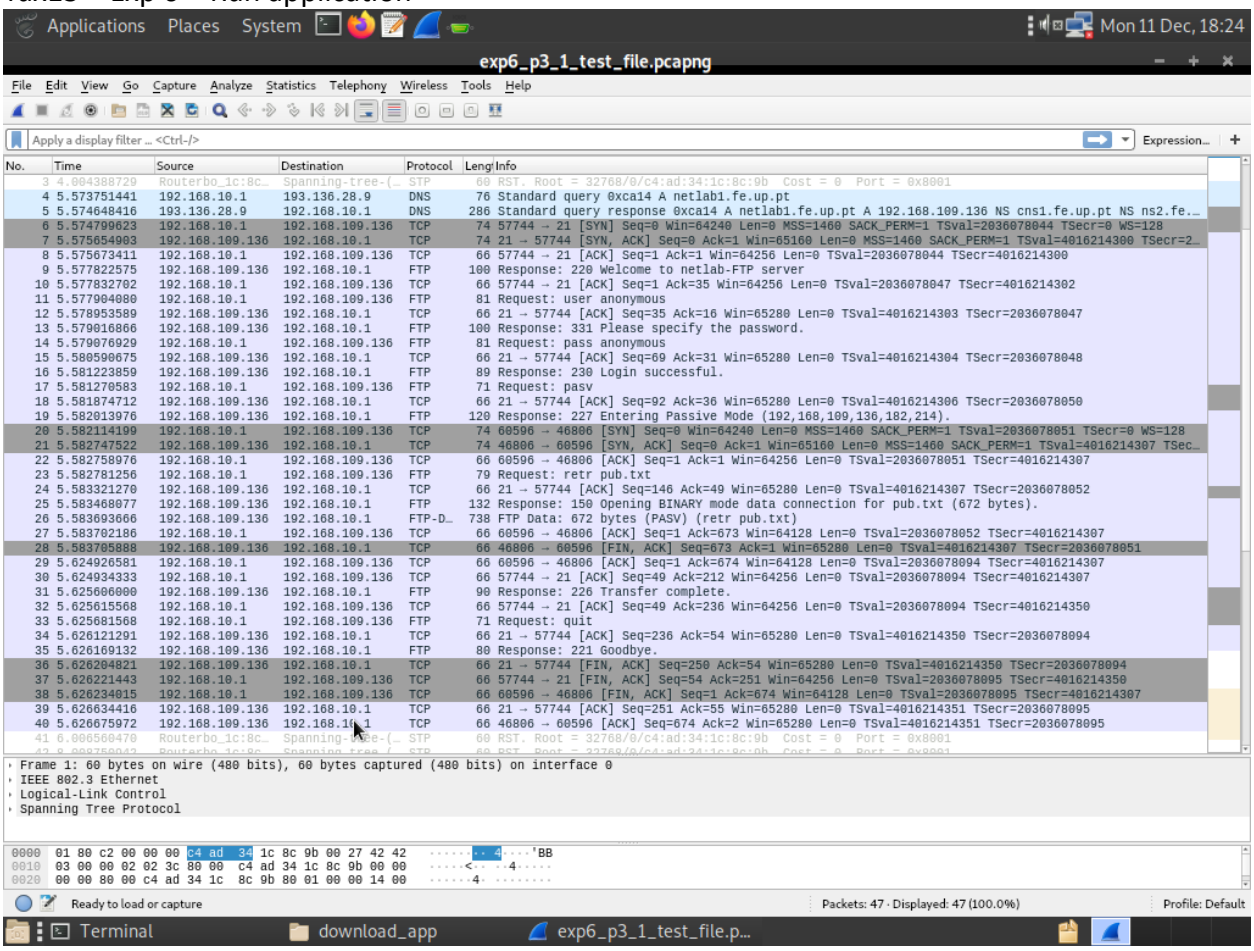
Tux13 – Exp 4 – Ping Tux12



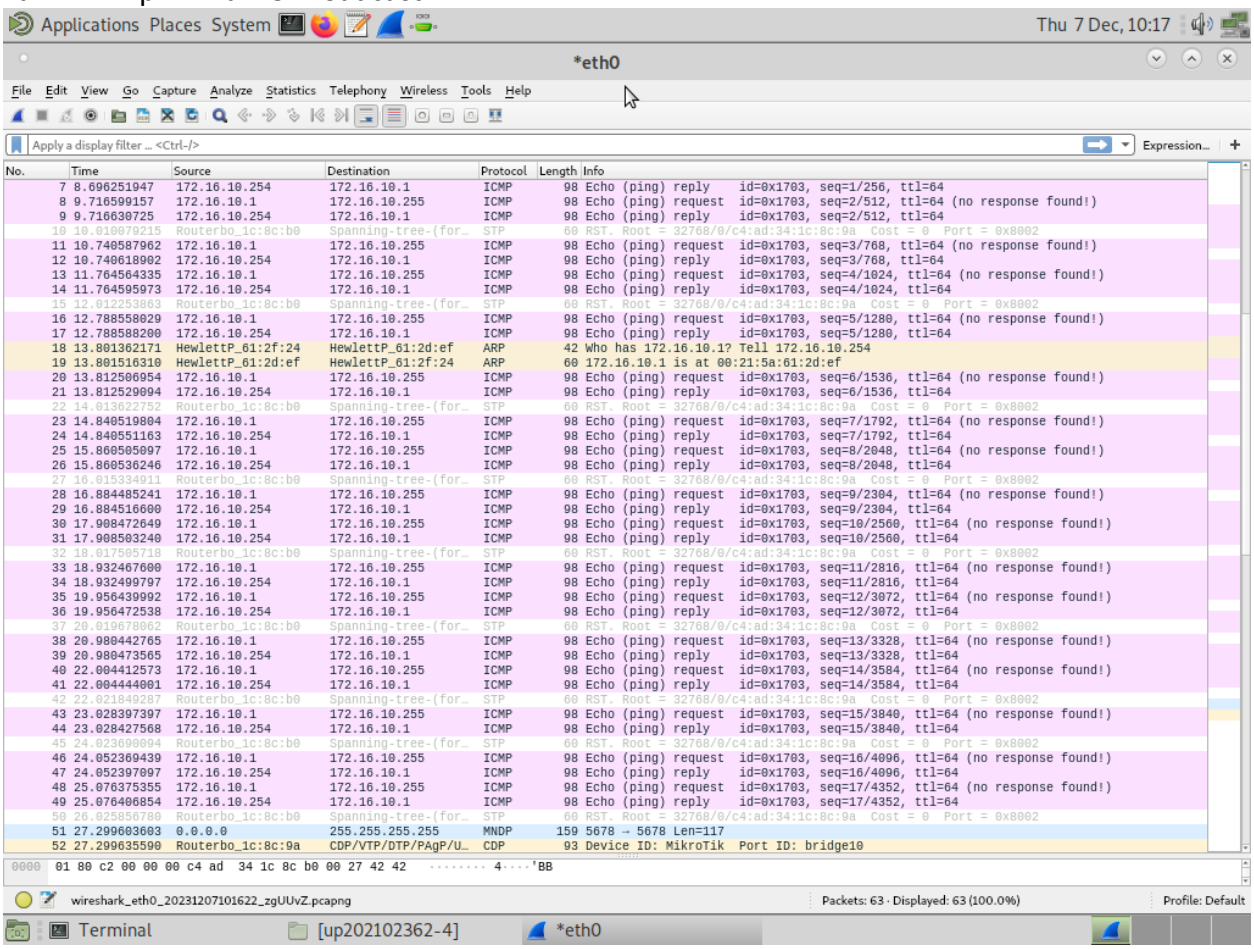
Tux13 – Exp 4 – Ping Tux14



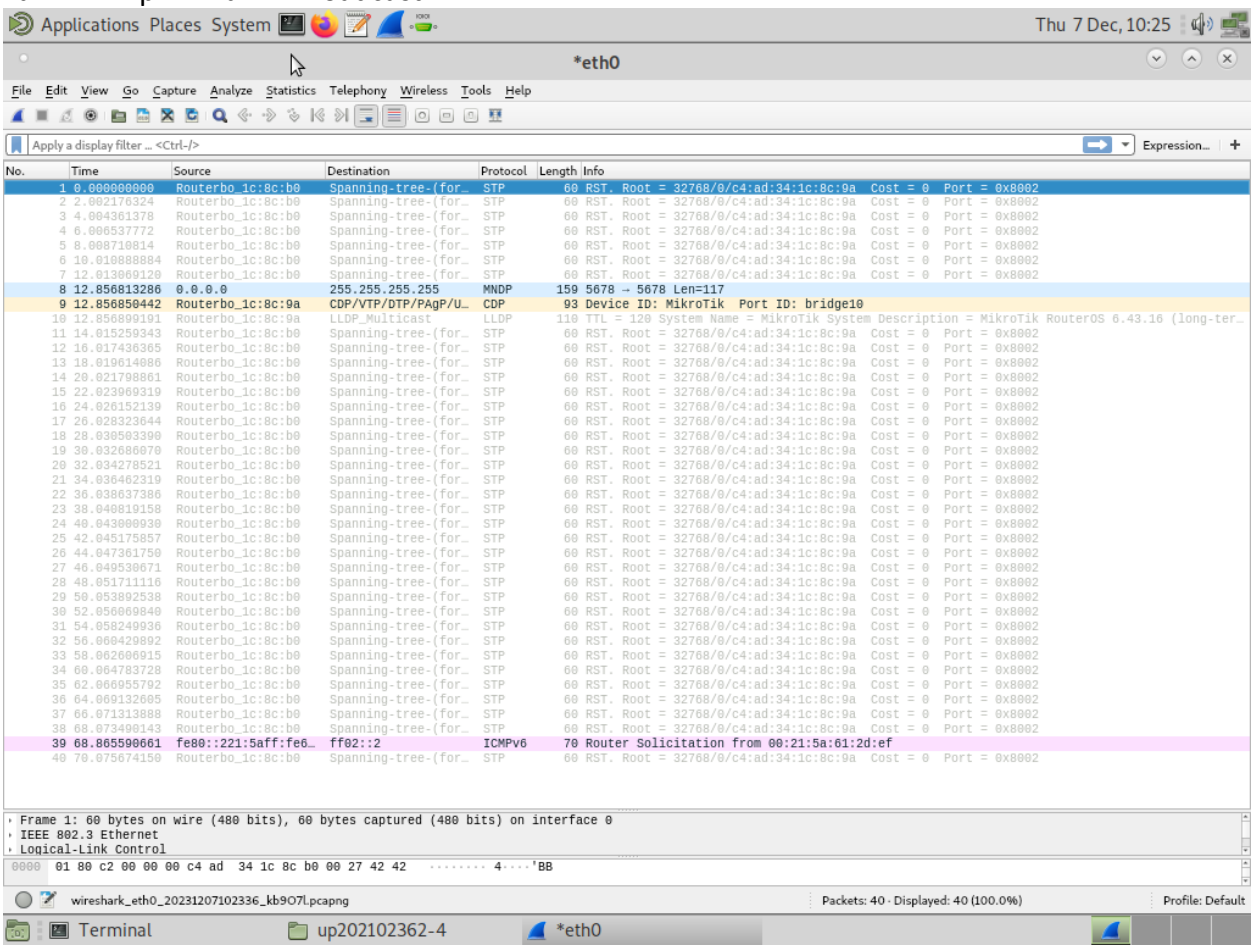
Tux13 – Exp 6 – Run application



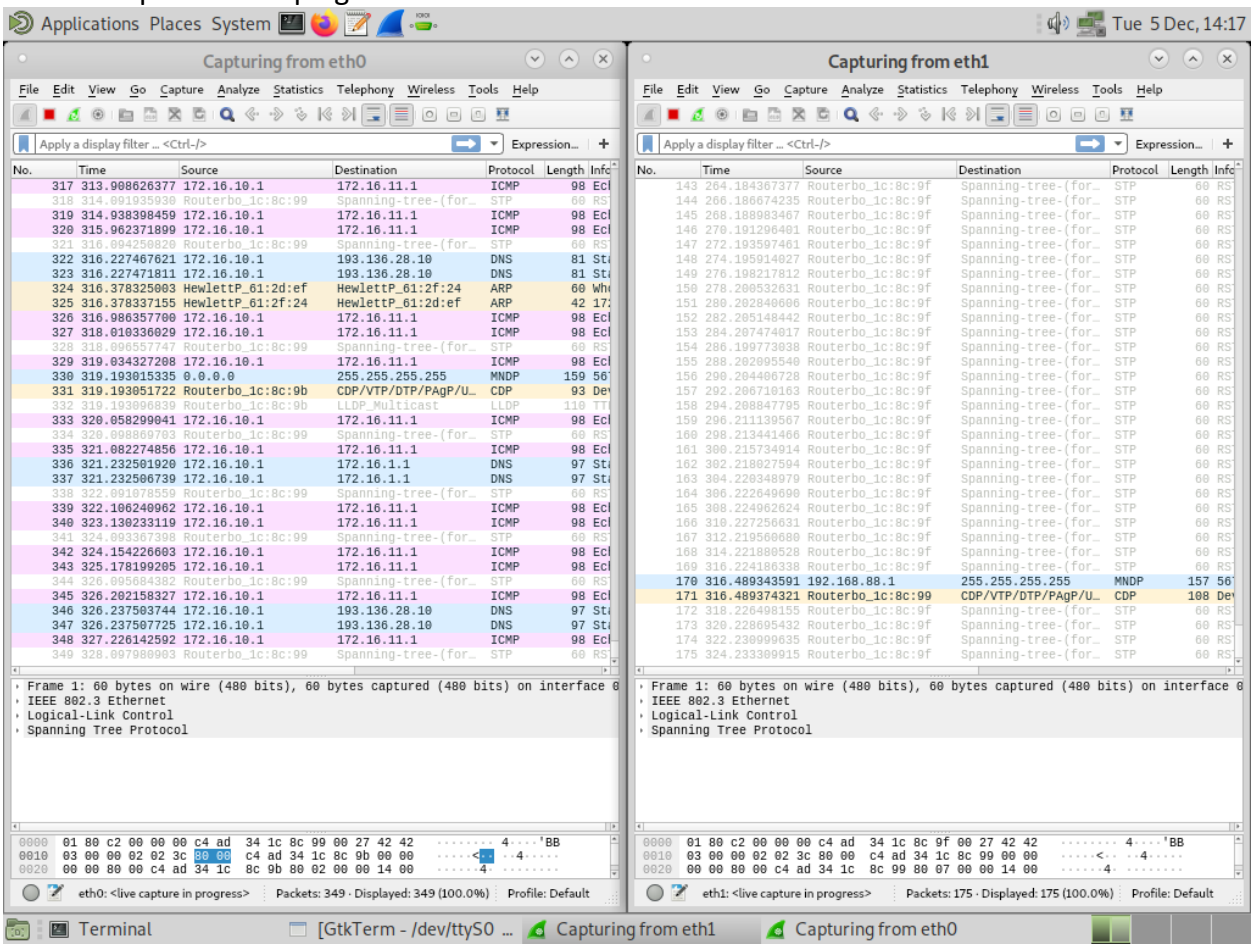
Tux14 – Exp 2 – Tux13 Broadcast



Tux14 – Exp 2 – Tux12 Broadcast



Tux14 – Exp 3 – Tux13 ping Tux12



Tux14 – Exp 4 – Tux13 Ping