Instituto de Informática - UFRGS
Laboratório de Redes de Computadores
Prof. Valter Roesler
Alunos: Arateus Meneses - 242260
Luis Miguel Santos Batista - 265037

Relatório do Laboratório 3

Programação em Sockets visando verificar a diferença entre transmissão confiável (TCP) e não confiável (UDP).

Porto Alegre
2018/2

Observações

- Durante a execução dos códigos utilizamos o ambiente Linux.

- Nas páginas seguintes estão os prints dos códigos utilizados para os testes. Notar que foram adicionados comentários e melhorias para atingir com maior eficiência as propostas do laboratório de sockets.

- Também estão presentes as capturas de tela da taxa de envio no UDP, adaptação de rede e log com a média de tráfego por segundo.

```c
#include <stdio.h>
#include <string.h>

#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#define SOCKET int

int main(int argc, char **argv)
{
        struct sockaddr_in peer;        //Estrutura que define os campos para o socket
        SOCKET s;       //Descritor do socket
        int porta, peerlen;     //Porta e peerlen  serão utilizadas ao realizar o bind.
        int rc, i;      //Em "rc" será salvo o retorno do recvfrom e 'i' é uma variável auxiliar para o parser
        char ip[16], buffer[1250];      //Buffers para guardar o endereço IP e o conteudo do pacote. Serão 1250 bytes contendo lixo
        double rate;    //Taxa de transmissão

        if (argc < 7)  //Testa se o numero de argumentos para o parser está correto
        {
                printf("Utilizar:\n");
                printf("trans -h <numero_ip> -p <porta> -r <rate>\n");
                exit(1);
        }

        // Realiza o parser dos parametros
        for (i = 1; i < argc; i++)
        {
                if (argv[i][0] == '-')
                {
                        switch (argv[i][1])
                        {
                        case 'h': // Numero IP
                                i++;
                                strcpy(ip, argv[i]);
                                break;

                        case 'p': // porta
                                i++;
                                porta = atoi(argv[i]);
                                if (porta < 1024)
                                {
                                        printf("Valor da porta invalido\n");
                                        exit(1);
                                }
                                break;
                        case 'r':
                                i++;
                                rate = atof(argv[i]);
                                break;

                        default:
                                printf("Parametro invalido %d: %s\n", i, argv[i]);
                                exit(1);
                        }
                  }
                else
                {
                       printf("Parametro %d: %s invalido\n", i, argv[i]);
                       exit(1);
                }
        }

        // Cria o socket na familia AF_INET (Internet) e do tipo UDP (SOCK_DGRAM)
        if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        {
                printf("Falha na criacao do socket\n");
                exit(1);
        }
        //strcpy(ip,"127.0.0.1");
        printf("oi\n");
        printf("ip: %s\nporta: %d\n", ip, atoi(argv[4]));


        // Cria a estrutura com quem vai conversar
        peer.sin_family = AF_INET;      //Ajusta a estrutura peer para a familia internet
        peer.sin_port = htons(porta);   //Ajusta a porta do peer com a porta. A função htons coloca na ordem da rede //Ajusta o ip destino da estrutura peer
        peer.sin_addr.s_addr = inet_addr(ip);   //Ajusta o ip destino da estrutura peer
        peerlen = sizeof(peer);

        // Envia pacotes de 1250 bytes "infinitamente"
        while (1)
        {
                sendto(s, buffer, sizeof(buffer), 0, (struct sockaddr *)&peer, peerlen);
                printf("enviando\n");
                usleep(10000000 / rate); //sleep que evita rajadas na transmissão e leva em consideração a taxa de transmissão
        }
}

```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define SOCKET int
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char **argv)
{
        struct sockaddr_in peer;
        SOCKET s;
        int porta, peerlen, rc, i;
        char buffer[100];

        if (argc < 2)
        {
                printf("Utilizar:\nrec -p <porta>\n");
                exit(1);
        }

        for (i = 1; i < argc; i++)
        {
                if (argv[i][0] == '-')
                {
                        switch (argv[i][1])
                        {
                        case 'p':
                                i++;
                                porta = atoi(argv[i]);
                                if (porta < 1024)
                                {
                                        printf("Porta invalida\n");
                                        exit(1);
                                }
                                break;
                        default:
                                printf("Parametro %d: %s invalido\n", i, argv[i]);
                                exit(1);
                        }
                }
                else
                {
                        printf("Parametro %d: %s invalido\n", i, argv[i]);
                        exit(1);
                }
        }

        // Cria o socket na familia AF_INET (Internet) e do tipo UDP (SOCK_DGRAM)
        if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        {
                printf("Falha na criacao do socket\n");
                exit(1);
        }

        // Define domenio, IP e porta a receber dados
        memset((void *)&peer, 0, sizeof(struct sockaddr_in));
        peer.sin_family = AF_INET;
        peer.sin_addr.s_addr = htonl(INADDR_ANY); // Recebe de qualquer IP
        peer.sin_port = htons(porta);                   // Recebe na porta especificada na linha de comando
        peerlen = sizeof(peer);

        // Associa socket com estrutura peer
        if (bind(s, (struct sockaddr *)&peer, peerlen))
        {
                printf("Erro no bind\n");
                exit(1);
        }

        printf("Socket inicializado. Aguardando mensagens...\n\n");

        // Recebe pacotes do cliente
        while (1)
        {
                rc = recvfrom(s, buffer, sizeof(buffer),0,(struct sockaddr *) &peer, (socklen_t *)&peerlen);
                printf("Recebendo..\n");
        }
}
```

TCP - SERVIDOR

```
1  // testesrv.cpp : Servidor exemplo
2  //
3
4  #include <stdio.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include <unistd.h>
8  #define _WIN32 1 // 1 ou zero dependendo do SO utilizado
9
10 #include <winsock2.h>
11 #include <WS2tcpip.h>
12 #include <sys/types.h>
13 #include <sys/socket.h>
14 #include <netinet/in.h>
15 #include <arpa/inet.h>
16 #define SOCKET  int
17 #define INVALID_SOCKET  ((SOCKET)~0)
18
19
20 #define MAX_PACKET 1250
21 #define PORTA_SRV 2023 // porta TCP do servidor
22
23 enum erros {WSTARTUP, ABRESOCK, BIND, ACCEPT, LISTEN,RECEIVE};
24
25 void TrataErro(SOCKET, int);
26
27 int main(int argc, char* argv[])
28 {
29   SOCKET s=0, s_cli; //Descritores dos sockets do servidor e cliente
30   struct sockaddr_in  addr_serv, addr_cli; //Estrutura que define os campos
       para o socket
31   socklen_t addr_cli_len=sizeof(addr_cli);
32
33
34   char recvbuf[MAX_PACKET];
35
36   // Cria o socket na familia AF_INET (Internet) e do tipo TCP (SOCK_STREAM)
37   if ((s = socket(AF_INET, SOCK_STREAM, 0))==INVALID_SOCKET)
38       TrataErro(s, ABRESOCK);
39
40   // Define domínio, IP e porta a receber dados
41   addr_serv.sin_family = AF_INET; //Ajusta a estrutura peer para a familia
       internet
42   addr_serv.sin_addr.s_addr = htonl(INADDR_ANY); // recebe de qualquer IP
43   addr_serv.sin_port = htons(atoi(argv[1]); //Ajusta a porta do peer com a
       porta. A função htons coloca na ordem da rede
44
45   // Associa socket com estrutura addr_serv
46   if ((bind(s, (struct sockaddr *)&addr_serv, sizeof(addr_serv))) != 0)
47       TrataErro(s, BIND);
48
49   // Coloca socket em estado de escuta para as conexoes na porta especificada
50   if((listen(s, 8)) != 0) // permite ateh 8 conexoes simultaneas
51       TrataErro(s, LISTEN);
52
53   // permite conexoes entrantes utilizarem o socket
```

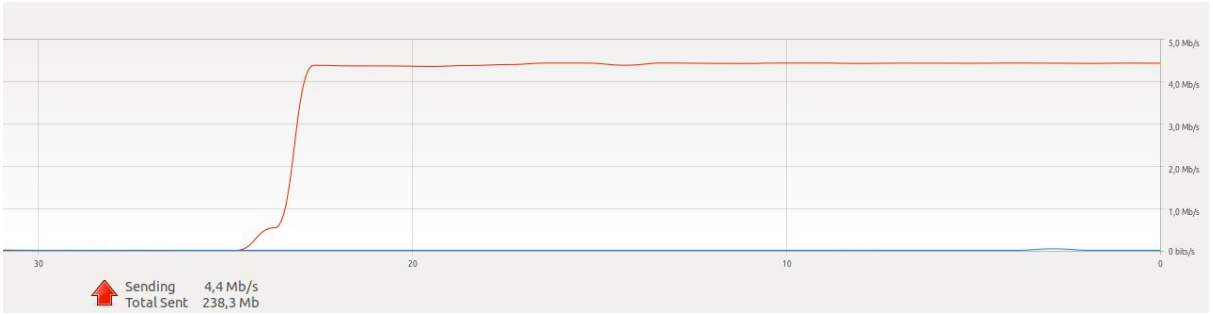
```
54    if((s_cli=accept(s, (struct sockaddr *)&addr_cli, (socklen_t *)
        &addr_cli_len)) < 0)
55        TrataErro(s, ACCEPT);
56
57    // fica esperando chegar mensagem
58    while(1)
59    {
60      if ((recv(s_cli, recvbuf, MAX_PACKET, 0)) < 0)
61      {
62        close(s_cli);
63        TrataErro(s, RECEIVE);
64      }
65
66      printf(" - msg recv - %s\n", recvbuf);
67    }
68
69    // fecha socket e termina programa
70    printf("Fim da conexao\n");
71    close(s);
72    close(s_cli);
73    exit(1);
74  }
75
76  void TrataErro(SOCKET s, int tipoerro)
77  {
78      char tipo[20];
79
80      switch(tipoerro) {
81          case WSTARTUP:
82              strcpy(tipo, "Windows Startup");
83              break;
84          case ABRESOCK:
85              strcpy(tipo, "Open Socket");
86              break;
87          case BIND:
88              strcpy(tipo, "Bind");
89              break;
90          case ACCEPT:
91              strcpy(tipo, "Accept");
92              break;
93          case LISTEN:
94              strcpy(tipo, "Listen");
95              break;
96          case RECEIVE:
97              strcpy(tipo, "Receive");
98              break;
99          default:
100             strcpy(tipo, "Indefinido. Verificar");
101             break;
102     }
103     printf("erro no %s", tipo);
104     close(s);
105     exit(1);
106 }
```

TCP - CLIENTE

```
1  // testecli.cpp : Defines the entry point for the console application.
2  //
3
4  #include <stdio.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include <time.h>
8  #include <unistd.h>
9
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <netinet/in.h>
13 #include <arpa/inet.h>
14 #define SOCKET  int
15 #define INVALID_SOCKET  ((SOCKET)~0)
16
17 //#define STR_IPSERVIDOR "192.168.0.146"
18
19 int main(int argc, char* argv[])
20 {
21   SOCKET s; //Descritor do socket
22   struct sockaddr_in  s_cli, s_serv; //Estrutura que define os campos para o
       socket
23
24
25   // abre socket TCP
26   if ((s = socket(AF_INET, SOCK_STREAM, 0))==INVALID_SOCKET)
27   {
28     printf("Erro iniciando socket\n");
29     return(0);
30   }
31
32   // seta informacoes IP/Porta locais
33   s_cli.sin_family = AF_INET; //Ajusta a estrutura peer para a familia internet
34   s_cli.sin_addr.s_addr = htonl(INADDR_ANY); //Ajusta o endereço do cliente
35   s_cli.sin_port = htons(atoi(argv[3]); //Ajusta a porta do peer com a porta. A
        função htons coloca na ordem da rede
36
37   // associa configuracoes locais com socket
38   if ((bind(s, (struct sockaddr *)&s_cli, sizeof(s_cli))) != 0)
39   {
40     printf("erro no bind\n");
41     close(s);
42     return(0);
43   }
44
45   // seta informacoes IP/Porta do servidor remoto
46   s_serv.sin_family = AF_INET; //Ajusta a estrutura peer para a familia
       internet
47   s_serv.sin_addr.s_addr = inet_addr(atoi(argv[1]); //Ajusta o ip destino da
       estrutura peer
48   s_serv.sin_port = htons(atoi(argv[2]); //Ajusta a porta do peer com a porta.
       A função htons coloca na ordem da rede
49
50   // connecta socket aberto no cliente com o servidor
51   if(connect(s, (struct sockaddr*)&s_serv, sizeof(s_serv)) != 0)
```

```
52    {
53      //printf("erro na conexao - %d\n", WSAGetLastError());
54      printf("erro na conexao");
55      close(s);
56      exit(1);
57    }
58
59  #if 0
60    // envia mensagem de conexao - aprimorar para dar IP e porta
61    if ((send(s, "Conectado\n", 11,0)) == SOCKET_ERROR);
62    {
63      printf("erro na transmissão - %d\n", WSAGetLastError());
64      closesocket(s);
65      return 0;
66    }
67  #endif
68
69    char str[1250]; //Buffer de 1250 bytes a ser enviado
70    char ch; //Caracetere utilizado para get/scan
71    int i; //Inteiro para auxiliar nesse scan
72
73    //Le do teclado e salva em str
74    for (i = 0; (i<80) && (ch = getchar()) != '\n'; i++)
75        str[i] = (char)ch;
76    str[i] = '\0';
77
78    //envia essa string infinitamente sobrecarregando o servidor
79    while(1)
80    {
81
82      if ((send(s, (const char *)&str, sizeof(str),0)) < 0)
83      {
84        close(s);
85        return 0;
86      }
87    }
88
89    // fecha socket e termina programa
90    printf("Fim da conexao\n");
91    close(s);
92    return 0;
93  }
94
```

- Com a execução do código UDP chegamos no gráfico abaixo:

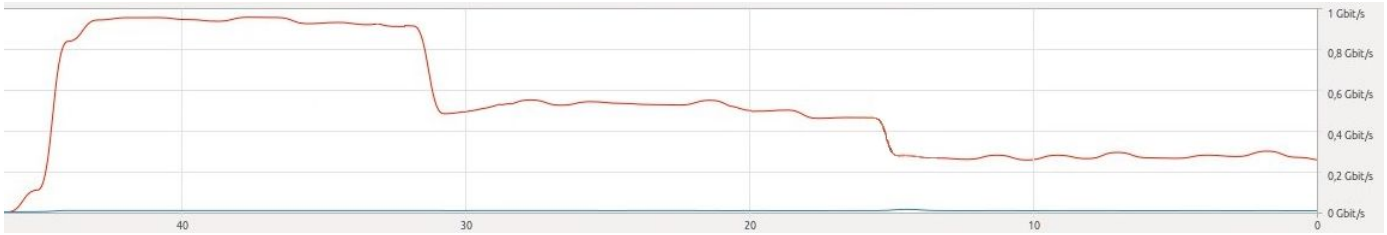


Levando em consideração a idade dos participantes do grupo (Arateus - 22 anos; Luis - 20 anos), é possível observar taxa de transmissão de 440 Kbits/s.

- Com a execução do código TCP, foi possível gerar o gráfico abaixo:

Essa é a visão da taxa de envio do cliente na máquina 1 para o servidor. No caso a comunicação entre eles estava a uma taxa de 1 Gbit/s porém diminuiu para 500 Mbits/s (pela metade) quando um outro cliente na máquina 2 também começou a enviar pacotes ao servidor. Isso ocorre devido à equidade de rede implementada pelo TCP.

- Gerando o gráfico com a média de tráfego para três conexões:



Quando há três conexões para um servidor, também há uma adaptação de rede. Pois inicialmente o primeiro processo estava enviando à taxa máxima disponível de 1Gbps, porém quando foi aberta uma segunda conexão, tal taxa diminui pela metade e posteriormente à ⅓ quando chegou um terceiro processo, havendo assim um compartilhamento da banda.