

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.

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #include <stdlib.h>
5 #include <unistd.h>
6 #include <sys/types.h>
7 #include <sys/socket.h>
8 #include <netinet/in.h>
9 #include <unistd.h>
10 #include <stdlib.h>
11 #define SOCKET int
12
13 int main(int argc, char **argv)
14 {
15     struct sockaddr_in peer; //Estrutura que define os campos para o socket
16     SOCKET s; //Descritor do socket
17     int porta, peerlen; //Porta e peerlen serão utilizadas ao realizar o bind.
18     int rc, i; //Em "rc" será salvo o retorno do recvfrom e 'i' é uma variável auxiliar para o parser
19     char ip[16], buffer[1250]; //Buffers para guardar o endereço IP e o conteúdo do pacote. Serão 1250 bytes contendo lixo
20     double rate; //Taxa de transmissão
21
22     if (argc < 7) //Testa se o numero de argumentos para o parser está correto
23     {
24         printf("Utilizar:\n");
25         printf("trans -h <numero_ip> -p <porta> -r <rate>\n");
26         exit(1);
27     }
28
29     // Realiza o parser dos parametros
30     for (i = 1; i < argc; i++)
31     {
32         if (argv[i][0] == '-')
33         {
34             switch (argv[i][1])
35             {
36                 case 'h': // Numero IP
37                     i++;
38                     strcpy(ip, argv[i]);
39                     break;
40
41                 case 'p': // porta
42                     i++;
43                     porta = atoi(argv[i]);
44                     if (porta < 1024)
45                     {
46                         printf("Valor da porta invalido\n");
47                         exit(1);
48                     }
49                     break;
50
51                 case 'r':
52                     i++;
53                     rate = atof(argv[i]);
54                     break;
55
56                 default:
57                     printf("Parametro invalido %d: %s\n", i, argv[i]);
58                     exit(1);
59             }
60         }
61         else
62         {
63             printf("Parametro %d: %s invalido\n", i, argv[i]);
64             exit(1);
65         }
66     }
67
68     // Cria o socket na familia AF_INET (Internet) e do tipo UDP (SOCK_DGRAM)
69     if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
70     {
71         printf("Falha na criacao do socket\n");
72         exit(1);
73     }
74     //strcpy(ip, "127.0.0.1");
75     printf("oi\n");
76     printf("ip: %s\nporta: %d\n", ip, atoi(argv[4]));
77
78
79     // Cria a estrutura com quem vai conversar
80     peer.sin_family = AF_INET; //Ajusta a estrutura peer para a familia internet
81     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
82     peer.sin_addr.s_addr = inet_addr(ip); //Ajusta o ip destino da estrutura peer
83     peerlen = sizeof(peer);
84
85     // Envia pacotes de 1250 bytes "infinitamente"
86     while (1)
87     {
88         sendto(s, buffer, sizeof(buffer), 0, (struct sockaddr *)&peer, peerlen);
89         printf("enviando\n");
90         usleep(1000000 / rate); //sleep que evita rajadas na transmissão e leva em consideração a taxa de transmissão
91     }
92 }
93

```

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define SOCKET int
6 #include <sys/types.h>
7 #include <sys/socket.h>
8 #include <netinet/in.h>
9
10 int main(int argc, char **argv)
11 {
12     struct sockaddr_in peer;
13     SOCKET s;
14     int porta, peerlen, rc, i;
15     char buffer[100];
16
17     if (argc < 2)
18     {
19         printf("Utilizar: \nnrec -p <porta>\n");
20         exit(1);
21     }
22
23     for (i = 1; i < argc; i++)
24     {
25         if (argv[i][0] == '-')
26         {
27             switch (argv[i][1])
28             {
29                 case 'p':
30                     i++;
31                     porta = atoi(argv[i]);
32                     if (porta < 1024)
33                     {
34                         printf("Porta invalida\n");
35                         exit(1);
36                     }
37                     break;
38                 default:
39                     printf("Parametro %d: %s invalido\n", i, argv[i]);
40                     exit(1);
41             }
42         }
43         else
44         {
45             printf("Parametro %d: %s invalido\n", i, argv[i]);
46             exit(1);
47         }
48     }
49
50     // Cria o socket na familia AF_INET (Internet) e do tipo UDP (SOCK_DGRAM)
51     if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
52     {
53         printf("Falha na criacao do socket\n");
54         exit(1);
55     }
56
57     // Define dominio, IP e porta a receber dados
58     memset((void *)&peer, 0, sizeof(struct sockaddr_in));
59     peer.sin_family = AF_INET;
60     peer.sin_addr.s_addr = htonl(INADDR_ANY); // Recebe de qualquer IP
61     peer.sin_port = htons(porta); // Recebe na porta especificada na linha de comando
62     peerlen = sizeof(peer);
63
64     // Associa socket com estrutura peer
65     if (bind(s, (struct sockaddr *)&peer, peerlen))
66     {
67         printf("Erro no bind\n");
68         exit(1);
69     }
70
71     printf("Socket inicializado. Aguardando mensagens...\n\n");
72
73     // Recebe pacotes do cliente
74     while (1)
75     {
76         rc = recvfrom(s, buffer, sizeof(buffer), 0, (struct sockaddr *)&peer, (socklen_t *)&peerlen);
77         printf("Recebendo..\n");
78     }
79 }

```

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

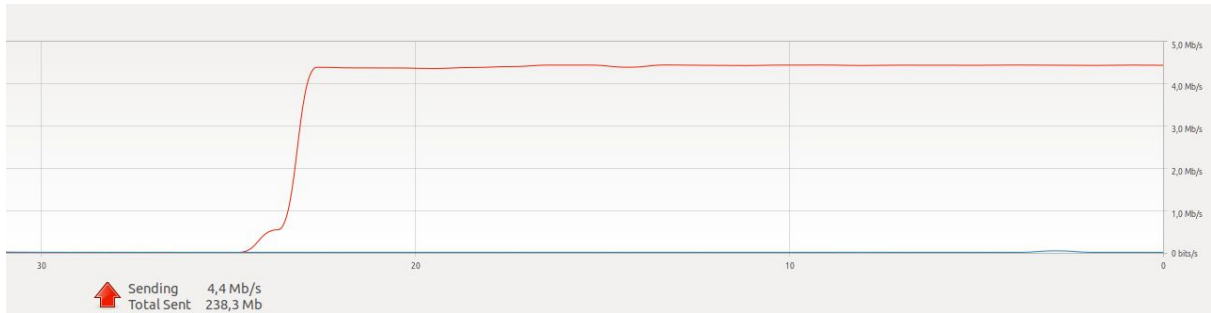
```
1 // testecli.cpp : Defines the entry point for the console application.
2 //
3
4 #include <stdio.h>
5 #include <string.h>
6 #include <stdlib.h>
7 #include <time.h>
8 #include <unistd.h>
9
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <netinet/in.h>
13 #include <arpa/inet.h>
14 #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; //Descriptor 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     // conecta 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; //Caracetera 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 a primeira conexão está enviando à taxa máxima disponível de 1Gbps, porém quando foi aberta uma segunda conexão, tal taxa diminui pela metade e posteriormente à $\frac{1}{3}$ quando chegou um terceiro processo, havendo assim um compartilhamento da banda.