UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Instituto de Informática

INF01154 - Redes de Computadores N
Prof. Valter Roesler

Laboratório 03

Guilherme Schievelbein
João Luiz Grave Gross

Porto Alegre, 26 de setembro de 2013.

Exercícios

1 UDP

1.1 Verificar as estruturas analisadas no diretório “esqueleto_udp”. Testar programa como está. Transformar o transmissor para enviar um número bits/s na rede de acordo com as seguintes linhas de comando:

“-r n”: onde n = multiplicação da idade da dupla em kbit/s. Por exemplo, se os dois possuem 20 anos, deve ser 400 kbit/s.

- **“-r 1000”: 1 Mbit/s**
- **“-r 2000”: 2 Mbit/s**

1.1.a) Descrição dos programas do diretório “esqueleto_udp”

No “esqueleto_udp”, temos dois códigos fonte, um pra o receptor e outro para o transmissor. No arquivo do receptor, inicialmente são tratadas as informações passadas ao programa pela linha de comando. A porta é indicada pela opção -p e deve ser uma porta de valor igual ou superior a 1024, já que portas de numeração inferior são reservadas ao sistema. O receptor cria um socket na familia AF_INET (Internet) e define o tipo de dados de transmissões em UDP (datagramas). Em seguida é estabelecida a porta por onde o receptor receberá as informações e os IPs de origem (qualquer um).

Após a configuração o receptor espera por mensagens e emite um “ACK” a cada mensagem recebida.

No arquivo do transmissor, inicialmente são tratados os parâmetros passados na inicialização do programa. O IP é indicado pela opção -h e a porta pela opção -p. O IP refere-se ao IP do servidor e a porta se refere a porta que o servidor está “escutando” para receber as informações, bem como a porta por onde o transmissor receberá a confirmação de entrega da sua mensagem enviada. Em seguida o transmissor enviada mensagens periodicamente ao receptor e, aguardando a cada mensagem por uma confirmação.

1.1.b) Alteração dos programas do diretório “esqueleto_udp” para permitir controle de taxa de envio

Para disponibilizar este recurso de controle de taxa de transmissão de transmissor para receptor, tivemos que fazer algumas alterações no transmissor. Adicionamos mais uma opção de passagem de parâmetros à aplicação pela opção -r, a qual indica a taxa de transmissão em kbits/s entre transmissor e receptor.

Para que este controle pudesse ser feito, estabelecemos que cada pacote de dados seria de 10.000 bits ou 1250 bytes. Além disso a taxa de transmissão passada como parâmetro determina o tempo de espera de envio entre uma mensagem e outra. O cálculo do tempo de espera é feito da seguinte forma:

```
kbits = atoi(argv[i]);

if(kbits % 10 != 0) {
	bytesSobra = ((kbits % 10) * 1000) / 8;
	quantSends = kbits/10 + 1;
	sleepTime = (float) 1000 / quantSends;
}
else {
	quantSends = -1;
	sleepTime = (float) 10000 / kbits;
}
```

Para taxas de transmissão que não são múltiplas de 10, são realizados n envios de 10.000 bits e um envio dos bits que restam para complear a taxa de transmissão. A variável bytesSobra calcula a quantidade de bytes que devem ser enviados nesta última transmissão. Já a variável quantSends determina o valor n de envios até que o último envio com os bytes que sobraram seja feito. A variável sleepTime, por sua vez, calcula o tempo de espera entre uma sequencia e outra de envios.

O controle dos envios é feito da seguinte forma:

```
while(1) {
	if(quantSends > -1)
		if(quantSends > 1) {
			sendto(s, buffer, 1250, 0, (struct sockaddr *)& peer, peerlen);
			quantSends--;
		}
		else {
			sendto(s, buffer, bytesSobra, 0, (struct sockaddr *)& peer, peerlen);
			quantSends = kbits/10 + 1;
		}
	else
		sendto(s, buffer, 1250, 0, (struct sockaddr *)& peer, peerlen);

	#ifdef _WIN32
		Sleep(sleepTime);
	#else
		usleep(sleepTime*1000);
```

```
        #endif
}
```

Se houver uma quantidade de envios definida, ou seja, para uma taxa de envio não múltipla de 10, são realizados n sends de mensagens de 1250 bytes e uma última mensagem dos bytes que sobraram para completar a taxa de transmissão. Caso a taxa de transmissão seja múltipla de 10, então cada envio é sempre de 1250 bytes. Após cada envio, independente da taxa de transmissão, é aguardado um tempo. Para fins de teste, desconsideramos o tempo de envio de cada mensagem, para facilitar o cálculo do tempo de espera entre o envio de uma mensagem e outra. Logo, depois observaremos que devido a isso a taxa de transmissão observada pelo canal de rede não foi exatamente igual a taxa de transmissão da aplicação transmissora, porém foi bastante semelhante.

1.1.c) Análise dos resultados obtidos

Para a execução dos testes foram utilizadas duas máquinas presentes na mesma redes, cada uma rodando uma versão do SO Ubuntu. As imagens a seguir foram capturadas da máquina transmissora.

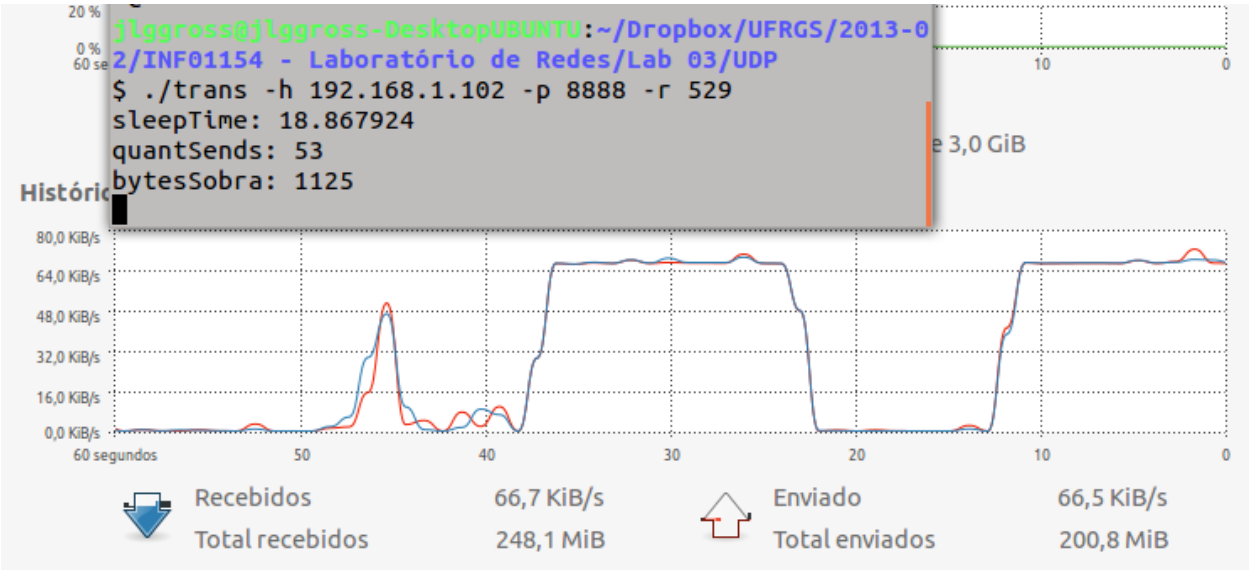


Figura 1: Transmissão em 529 kbps

Ambos os componentes do grupo tem 23 anos, logo a primeira taxa de transmissão realizada foi 23², ou 529. A taxa percebida no canal foi de 66,5 KiB/s. Cada KiB equivale a 1024 bytes, logo:

66,5KiB/s = 66,5 * 1024 bytes/s = 68,096 Kbytes/s = 68.096 * 8 bits/s = 544,768 kbps

Portanto a taxa de transmissão efetiva foi de 544,77 kbps, ao invés dos 529 kbps setados no transmissor.

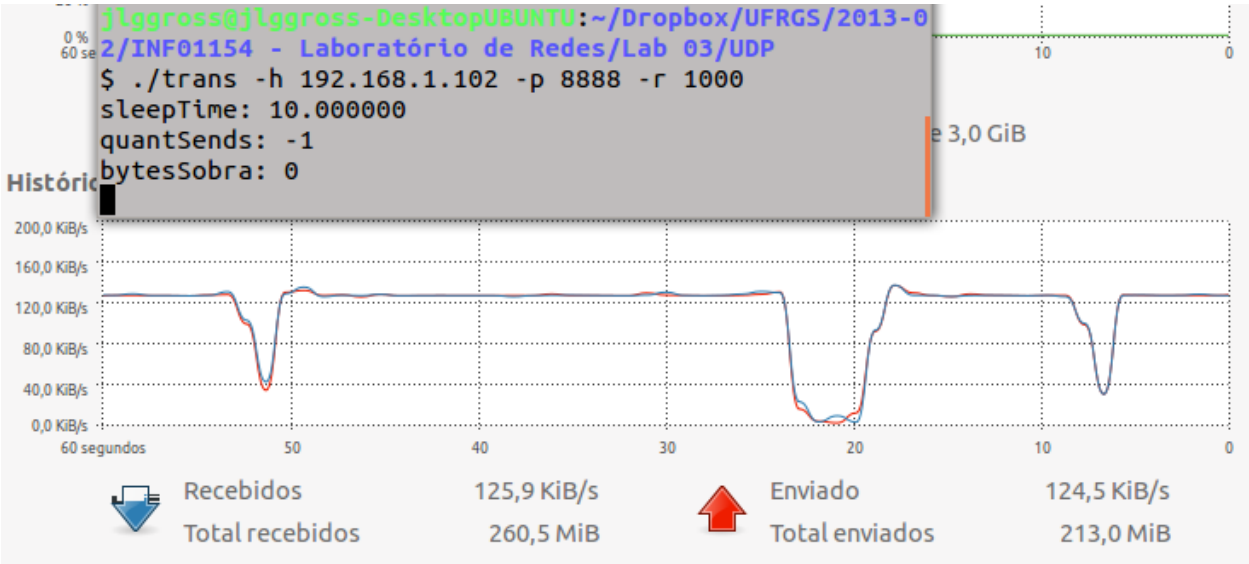


Figura 2: Transmissão em 1000 kbps

124,5 KiB/s = 124,5 * 1024 bytes/s = 127,488 Kbytes/s = 127.488 * 8 bits/s = 1019,9 kbps

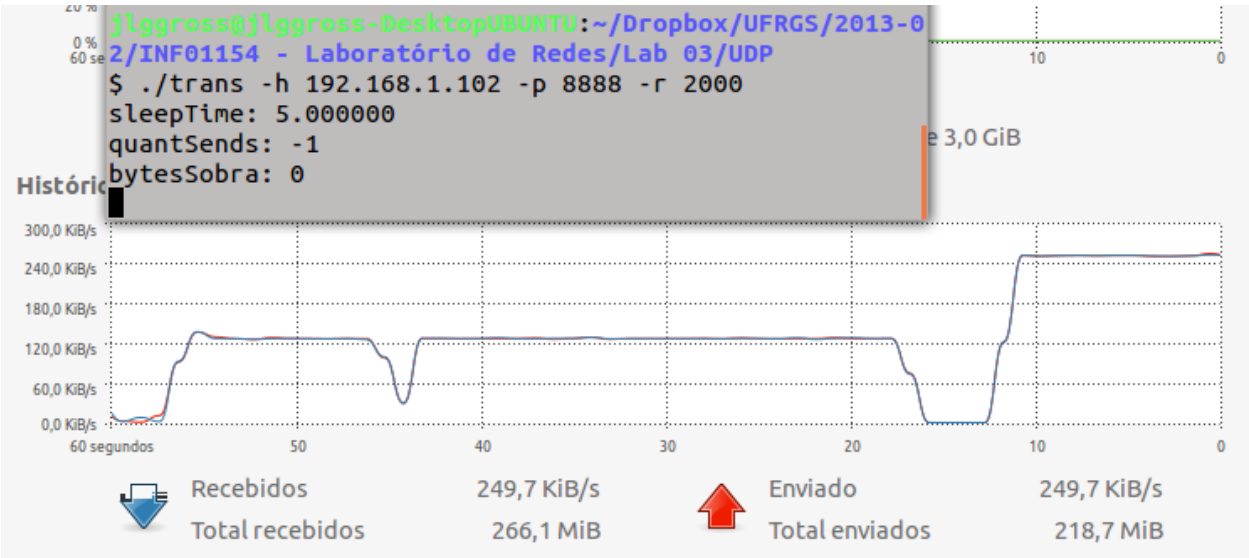


Figura 3: Transmissão em 2000 kbps

249,7 KiB/s = 249,7 * 1024 bytes/s = 255,693 Kbytes/s = 255.693 * 8 bits/s = 2045,5 kbps

O erro das transmissões ficou cerca de +3% do valor objetivado pela aplicação transmissora.

2 TCP

2.1 Verificar as estruturas analisadas no diretório "esqueleto_tcp". Testar programa como está. Analisar eqüidade do tráfego na rede, alterando o programa para ficar enviando dados na máxima velocidade possível, e testando com várias transmissões simultâneas (mínimo de 2 conexões –dois servidores (duas instâncias do servidor em portas diferentes) e dois clientes). Gerar log com a média de tráfego por segundo. Pode-se realizar o relatório com duas máquinas, mas se o grupo quiser, poderá utilizar 3 máquinas ou mais (para ver na ferramenta de redes, além do log, a variação de carga na rede)

Foi criado código para tratar os parâmetros de entrada no servidor e cliente:

```
//test parameters
if(argc<1) {
    printf("Use tcpserver -p <port_number>\n");
    exit(1);
}
int i;
for(i=1; i<argc; i++){
    if(argv[i][0]=='-'){
        switch(argv[i][1]){
            case 'p':
                i++;
                port = atoi(argv[i]);
                if(port < 1024){
                    printf("Invalid port number\n");
                    exit(1);
                }
                break;
            default:
                printf("Invalid parameter: %s\n",argv[i]);
                exit(1);
        }
    }else{
        printf("Invalid parameter: %s\n",argv[i]);
        exit(1);
    }
}
```

Servidor

```c
//test parameters
if(argc < 6){
    printf("Use: tcpclient -h <server_ip> -p <client_port> -q <server_port>\n");
    exit(1);
}
int i;
for(i=1; i<argc; i++){
    if(argv[i][0]=='-'){
        switch(argv[i][1]){
            //ip
            case 'h':
                i++;
                strcpy(ip, argv[i]);
                break;
            //client port
            case 'p':
                i++;
                client_port = atoi(argv[i]);
                if(client_port < 1024) {
                    printf("Invalid cliente port number\n");
                    exit(1);
                }
                break;

            case 'q':
                i++;
                server_port = atoi(argv[i]);
                if(server_port < 1024) {
                    printf("Invalid server port number\n");
                    exit(1);
                }
                break;

            default:
                printf("Invalid paramenter: %s\n",argv[i]);
        }
    }else{
        printf("Invalid paramenter: %s\n",argv[i]);
        exit(1);
    }
}
```

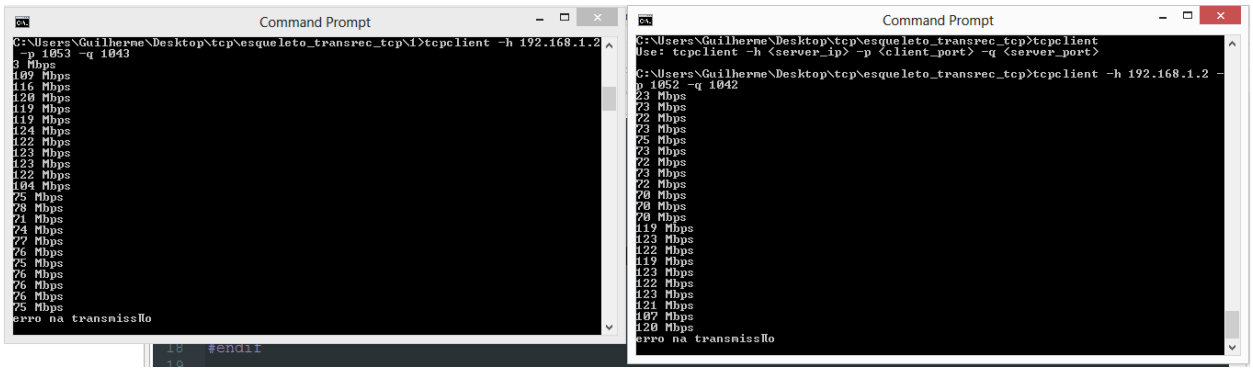
```
int printf (const char *__format, ...)
```

Cliente

O laço de envio de mensagens do código cliente foi modificado. Como mensagem a ser transferida, definimos um vetor de caracteres de tamanho 250, que é enviado indefinidamente durante a execução dos programas, fazendo com que se alcance a velocidade máxima de transmissão. Para a comparação das instâncias em execução, imprimimos no terminal a cada 1 segundo a quantidade de bits enviados nesse intervalo, como mostrado abaixo:

```c
// recebe do teclado e envia ao servidor
char str[1250];
char ch;
i = 0;

  time_t inicio;
  time_t atual;

  i = 0;
  inicio = time(NULL);

while(1)
{
      atual = time(NULL);
      i++;

      if (atual - inicio >= 1){
          printf("%d Mbps\n\n", (i*1250*8) / (1000000));
          i = 0;
          inicio = atual;
      }
```

Abaixo temos um exemplo da execução de 2 servidores e 2 clientes.



O cliente da esquerda(L) usa inicialmente toda a banda disponível (cerca de 110 Mbps), mas quando o cliente da direita(D) é acionado, a banda é dividida, e a taxa de transferência de ambos passa a ser cerca de 70 Mbps. Quando L é interrompido, D passa a usar toda a banda disponível.

2.2. O que é equidade de tráfego e em qual protocolo ela foi observada nativamente no laboratório?

É a divisão justa (igualitária) da banda total disponível entre as conexões existentes.

Ocorre nativamente no protocolo TCP.

2.3. Qual o nível do modelo OSI para fazer controle de fluxo quando o programador utiliza TCP e quando utiliza UDP?

UDP não possui controle de fluxo nativo na camada de transporte. Se necessário, pode ser feito na camada de aplicação.

TCP possui controle de fluxo nativo na camada de transporte. Utiliza o conceito de camada deslizante para adaptar-se a capacidade do transmissor e receptor.