Nome: João Pedro Moreira Antunes

Introdução

Neste trabalho será feito a implementação de dois códigos cliente/servidor, um TCP e um UDP. O chat irá permitir a comunicação entre usuários e possibilitar o envio de arquivos. Serão apontadas as principais diferenças de funcionamento entre os dois protocolos, observando suas reações a diferentes condições de rede, como perda de pacotes e latência.
Através do uso do Wireshark e da ferramenta netem, será possível monitorar o tráfego gerado pela aplicação e aplicar simulações de redes com falhas e atrasos, avaliando o impacto dessas condições no desempenho da comunicação.

Servidor:

O servidor é responsável pela maior parte do funcionamento do chat. Como hub, o servidor recebe mensagens, arquivos, comandos e faz a distribuição deles para os demais clientes. Para isso, foi feito o uso de uma thread para cada cliente, garantindo a sincronização com a exclusão mútua em recursos compartilhados. Em relação as diferenças dos protocolos, é mais acessível trabalhar com o servidor TCP, pois ele estabelece uma conexão com o cliente, facilitando a implementação.

```c
if(pthread_create(&thread_id, NULL, client_main, (void *)socket_alloc) < 0){ //cria uma thread para cuidar de cada cliente
    die("pthread_create");
}
```

```c
void *client_main(void *socket_alloc){
    int socket = *(int *)socket_alloc;
    char buf[BUFLEN];
    int recv_len;
    char sender_name[50] = "Stranger"; //nome padrão do cliente

    while(1){
        memset(buf, 0, sizeof(buf));

        recv_len = read(socket, buf, BUFLEN);
        if(recv_len <= 0){
            close(socket);
            break;
        }

        if (strncmp(buf, DISCONNECT, strlen(DISCONNECT)) == 0) { //Se comando "/quit" desconecta cliente
            remove_client(socket);
            break;
        }
        if (strncmp(buf, REG_PREFIX, strlen(REG_PREFIX)) == 0) {  //Se comando "/reg" registra o nome do cliente
            char *name = buf + strlen(REG_PREFIX);
            register_client(socket, name);

            pthread_mutex_lock(&clients_mutex);
            for(int i=0; i < num_clients; i++){
                if(clients[i].sock == socket){
                    strncpy(sender_name, clients[i].name, sizeof(sender_name) - 1);
                    sender_name[sizeof(sender_name) - 1] = '\0';
                    break;
                }
            }
            pthread_mutex_unlock(&clients_mutex);
            continue;
        }

        char full_msg[BUFLEN + 50];
        snprintf(full_msg, sizeof(full_msg), "%s: %s\n", sender_name, buf); //mensagem inteira (nome + mensagem)
        send_message(socket, full_msg);
    }

    free(socket_alloc); //desaloca o socket
    return 0;
}
```

Nome: João Pedro Moreira Antunes

Aqui é organizado as funções de remoção e registro dos clientes, e o envio adequado das mensagens.

Cliente:

O cliente apenas envia as mensagens e os arquivos de texto. Diferente da conexão do TCP, no protocolo UDP o cliente conecta-se registrando seu nome no servidor.

```
printf("Registre-se com /reg <nome>\n");

fgets(command, sizeof(command), stdin);
if (strncmp(command, REG_PREFIX, strlen(REG_PREFIX)) == 0) {
    sendto(s, command, strlen(command), 0, (struct sockaddr *)&si_other, slen);
}
```

1) As portas dos clientes geradas aleatoriamente sempre que se conectam, enquanto a do servidor é escolhida ao ser criado. Nesse caso, seu número é 8080.

4	14.870769095	127.0.0.1	127.0.0.1	UDP	53 54233 → 8080 Len=9
7	19.318976252	127.0.0.1	127.0.0.1	UDP	57 52174 → 8080 Len=13
11	23.358957648	127.0.0.1	127.0.0.1	UDP	53 52174 → 8080 Len=9
12	23.358990127	127.0.0.1	127.0.0.1	UDP	63 8080 → 54233 Len=19
13	26.104878906	127.0.0.1	127.0.0.1	UDP	52 52174 → 8080 Len=8
14	26.104906247	127.0.0.1	127.0.0.1	UDP	62 8080 → 54233 Len=18
15	30.302994545	127.0.0.1	127.0.0.1	UDP	54 52174 → 8080 Len=10

2) O protocolo TCP utiliza pacotes adicionais para controle de fluxo e confiabilidade. Para estabelecer uma conexão, ele utiliza o processo de three-way handshake, que envolve três etapas: primeiro, o cliente envia um pacote de sincronização (SYN), seguido pela resposta do servidor com um pacote de sincronização e confirmação (SYN-ACK), e, por fim, o cliente envia um pacote de confirmação (ACK). Além disso, o TCP possui mecanismos para retransmissão de pacotes perdidos, garantindo uma entrega confiável dos dados. E por outro lado, o protocolo UDP não implementa controle de fluxo ou retransmissão, porém ele é mais simples.

3) O protocolo UDP tende a ser mais rápido por não ter as etapas de verificação do TCP, porém a falta de controle sobre os pacotes pode resultar em mensagens ou arquivos corrompidos.

4) Para arquivos menores que 1500 bytes, um único pacote UDP pode ser o suficiente, enquanto o protocolo TCP usa mais pacotes para sincronização da conexão.

Envio de um arquivo de texto de 512bytes:

58	156.452653536	127.0.0.1	127.0.0.1	UDP	556 58987 → 8888 Len=512
59	156.452717865	127.0.0.1	127.0.0.1	UDP	556 8888 → 58987 Len=512

9	50.037515905	127.0.0.1	127.0.0.1	TCP	76 36626 → 8888 [SYN] Seq=0
10	50.037525233	127.0.0.1	127.0.0.1	TCP	76 8888 → 36626 [SYN, ACK]
11	50.037534273	127.0.0.1	127.0.0.1	TCP	68 36626 → 8888 [ACK] Seq=1
12	50.037817755	127.0.0.1	127.0.0.1	TCP	580 36626 → 8888 [PSH, ACK]
13	50.037822446	127.0.0.1	127.0.0.1	TCP	68 8888 → 36626 [ACK] Seq=1
14	50.038214613	127.0.0.1	127.0.0.1	TCP	580 8888 → 36626 [PSH, ACK]
15	50.038220327	127.0.0.1	127.0.0.1	TCP	68 36626 → 8888 [ACK] Seq=5
16	50.038380470	127.0.0.1	127.0.0.1	TCP	68 36626 → 8888 [FIN, ACK]
17	50.079563951	127.0.0.1	127.0.0.1	TCP	68 8888 → 36626 [ACK] Seq=5

5) Para arquivos maiores, como 15000 bytes, haverá fragmentações em vários pacotes UDP, enquanto o TCP gerenciará essa divisão internamente.

Nome: João Pedro Moreira Antunes

6) Primeiro é necessário verificar a interface de rede do seu dispositivo utilizando o seguinte comando***: ip a (no meu caso é enp0s3)***
E para aplicar perda de pacotes usasse o comando***: sudo tc qdisc add dev eth0 root netem loss 30% (30% de perda)***
Para a remoção de qualquer tipo de controle de fluxo utiliza-se: ***sudo tc qdisc del dev eth0 root netem***

O protocolo TCP garante a entrega confiável dos pacotes, retransmitindo os pacotes quando perdidos. Enquanto, no protocolo UDP os pacotes são simplesmente perdidos.

7) Para incluir latência na interface de rede utilizamos o seguinte comando: ***sudo tc qdisc add dev eth0 root netem delay 200ms (atraso de 200ms)***
Também podemos adicionar uma latência variável: ***sudo tc qdisc add dev eth0 root netem delay 200ms 10ms (a latência irá variar aleatoriamente entre 190ms e 210ms)***

O protocolo TCP garante a entrega dos pacotes, mas pode ser afetado por conta da espera por confirmações ACKs. Caso isso ocorra, o TCP diminuirá a taxa de dados a serem retransmitidos até que eles sejam confirmados. O número de pacotes do protocolo UDP não é afetado, pois não precisa de confirmações para que continue enviando pacotes, mas ainda assim terá atraso devido a latência.