UDP Friendly

Gabriel Poça Maria Alves Tiago Ribeiro

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a56974,a54807,a54752}@alunos.uminho.pt

Resumo

Resumo do relatório.

1 Notas Iniciais

Este documento serve como resumo, intermédio, para acompanhamento do projecto prático de Comunicação por Computadores. São abordadas as metodologias e os conceitos adoptados.

2 Implementação

Esta secção trata com maior detalhe a componente da implementação do projecto. Aqui são abordados as diferentes entidades a funcionar em cada aplicação (cliente e servidor), a implementação dos cabeçalhos, etc.

3 Comunicação

3.1 Tipos de Mensagens

Cada mensagem tranpsorta diferentes elementos de informação, por exemplo, um mensagem *INFO* tem bytes da informação a enviar e um indicador da posição da mesma. Existem três **elementos de cabeçalho**:

Tipo Tipo da mensagem (os diferentes tipos são apresentados a seguir).

Posição Numero que representa a posição do pacote numa sequência que constitui a informação enviada.

Data Informação a enviar por pacote.

Diferentes tipos de mensagens preenchem diferentes elementos de informação, mensagens de *ACK* não enviam bytes de informação. Existem os seguintes tipos de mensagem:

SYN Mensagem inicial no establecer da comunicação com o servidor.

SYN_ACK Mensagem de confirmação de SYN.

INFO Mensagem que transporta informação sobre o documento a enviar.

ACK Mensagem de confirmação da recepção de uma mensagem INFO.

FIN Mensagem de final de comunicação.

FIN_ACK Mensagem de confirmação da recepção de FIN.

No processo de comunicação do cliente com o servidor o primeiro apenas comunica com mensagens *SYN*, *INFO* e *FIN* e o outro com as restantes. Mas tal será esclarecido na secção sobre protocolo.

3.2 Protocolo

A comunicação pode ser dividida em três componentes: establecer da comunicação, envio da informação e terminar da comunicação. As secções abaixo explicam as mesmas.

3.2.1 Establecer da comunicação

A comunicação tem inicio com o envio da mensagem *SYN* pelo client. O servidor recebe a mensagem e responde com *SYN_ACK*. Da mensagem *SYN_ACK* o cliente retira informação quanto à porta para a qual deverá continuar a comunicação, tal é necessário uma vez que se trata de aplicação para multiplos servidores, caso contrário não haveria necessidade de mudança de porta. Establecida a comunicação o cliente pode enviar informação e terminar a comunicação.

INSERIR GRÁFICO DAS PORTAS

3.2.2 Envio da informação

O *upload* de informação para o servidor é realizado através de mensagens *INFO*. Cada mensagem é constituida por informação (conjunto de bytes a enviar para o servidor) e um indicador da posição da mesma informação numa sequencia que permite reconstituir a informação no servidor.

3.2.3 Terminar

A comunicação termina quando o cliente envia a mensagem *FIN*. O servidor deve responder com *FIN_ACK* procedendo então à descodificação da informação recebida.

3.3 Controlo de congestão

Quando a carga oferecida a uma rede é superior à sua capacidade acontece um congestionamento. A soluçao para este problema passa por diminuir a taxa de transmissão de dados. Existe um protocolo que gere essa taxa bastante bem, o TCP. E uma vez que a sua implementação foi estudada exaustivamente, acreditamos que seja possivel obter bons resultados. Como tal decidimos que o nosso controlo de congestão seguiria o modelo que o TCP utiliza.

Uma das solução para fazer variar a transmissão de dados passa por implementar o mecanismo de janela dinâmica.

3.3.1 Aumentar o tamanho da janela

De cada vez que o servidor recebe um pacote, envia uma confirmação para o cliente. À medida que essas confirmações chegarem, caso não haja timeout's, e assim que o número de confirmações for igual ao tamanho da janela, esta aumentará. Aumentará para o dobro, caso o tamanho da janela seja inferior ao Threshold, caso contrário será incrementada em uma unidade.

3.3.2 Diminuir o tamanho da janela

Quando uma confirmação não chega, ou não chega a tempo ocorre um timeout. O cálculo do timeout é baseado numa soma entre a média ponderada do RTT (round trip time) novo (sampleRTT) e o anterior (estimatedRTT) e quatro vezes a média ponderada do desvio padrão anterior e o novo. As funções a seguir apresentadas correspondem ao cálculo desses três tempo, estimatedRTT, devRTT e timeout:

Listing 1: Implementação do algortimo de calculo do timeout

```
private long estimateRTT(long sampleRTT) {
        long newRTT = (long) ((1 - _alpha) *
                _estimatedRTT + _alpha * sampleRTT);
                _estimatedRTT = newRTT;
        return _estimatedRTT;
}

private long calculateDevRTT(long sampleRTT) {
        long newdevRTT = (long) ((1 - _beta) * _devRTT +
                _beta * (Math.abs(sampleRTT - _estimatedRTT)));
        _devRTT = newdevRTT;
        return _devRTT;
}

private void calculateTimeOut(long sampleRTT) {
        _timeout = estimateRTT(sampleRTT) + 4 * calculateDevRTT(sampleRTT);
}
```