

UDP Friendly

Gabriel Poça, Maria Alves, and Tiago Ribeiro

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

Resumo O UDP (User Datagram Protocol) é um protocolo simples da camada de transporte. Este protocolo faz a entrega de mensagens independentes, denominadas por datagramas, entre processos ou aplicações em sistemas de host. O protocolo UDP caracteriza-se por permitir uma transmissão de dados não fiável, uma vez que não possui qualquer mecanismo de controlo de fluxo e controlo de congestão.

1 Introdução

O UDP é uma escolha adequada para fluxos de dados em tempo real, especialmente aqueles que admitem perda ou corrompimento de parte de seu conteúdo, tais como vídeos ou voz. Aplicações sensíveis a atrasos na rede, mas poucos sensíveis a perdas de pacotes, como jogos de computadores, também podem se utilizar do UDP. O socket UDP é identificado através de dois números: endereço IP destino, e número de porta destino. Quando um Sistema Terminal recebe um segmento UDP verifica qual o número da porta destino que consta do segmento UDP e redirecciona o segmento para o socket com esse número de porta. Os Datagramas com diferentes endereços IP origem e/ou portas origem podem ser redireccionados para o mesmo socket. Caso o recurso a broadcasting e multicasting sejam necessários, o UDP deverá necessariamente ser utilizado. Este protocolo não perde tempo com criação ou destruição de conexões. Durante uma conexão, o UDP troca apenas 2 pacotes. Por isto, aplicações que encaixam num modelo de pergunta-resposta também são fortes candidatas a usar UDP. Entretanto, pode ser necessário implementar algoritmos de timeouts, acks e, no mínimo, retransmissão. Um maiores problemas deste protocolo é não ser amigável para o funcionamento da rede uma vez que tem um comportamento egoísta ao não ajustar o débito de envio face à largura de banda disponível. Para que nos seja permitido regular o débito dos dados a transmitir em função dos níveis de carga da rede foi necessário desenvolver uma camada que complementasse o protocolo UDP.

2 Objectivos

A construção da camada complementar ao protocolo UDP deve acentar nos seguintes propósitos definidos:

- Providenciar o estabelecimento e termino fiável de uma conexão;
- Oferecer um serviço de transporte de dados eficiente, em que os dados a transmitir são vistos como uma stream de pacotes (512Bytes), controlada por um mecanismo de janela orientado ao pacote;
- Implementar um mecanismo que ajuste a taxa de transmissão de envio de pacotes com base na perda de pacotes;
- Para simplificar a detecção da perda, cada pacote recebido pode ser individualmente confirmado;
- Continuar a ser um protocolo de transporte não fiável, i.e. sem incluir qualquer tipo de retransmissão;
- Continuar a ser um protocolo de transporte que não garante a entrega ordenada.

3 Comunicação

3.1 Tipos de Mensagens

Cada mensagem transporta 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 estabelecer 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.

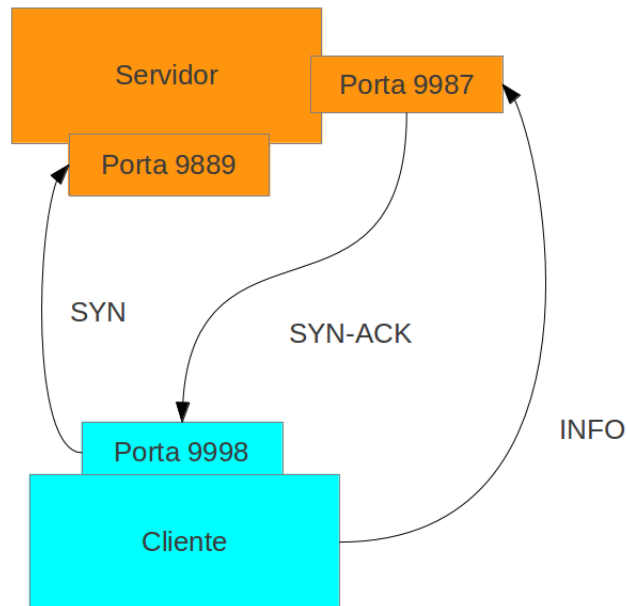
FIN_ACK Mensagem de confirmação da recepção de FIN.

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: estabelecer da comunicação, envio da informação e terminar da comunicação. As secções abaixo explicam as mesmas.

Estabelecer 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. Estabelecida a comunicação o cliente pode enviar informação e terminar a comunicação.



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

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.

4 Controlo de congestão

Estamos perante congestionamento quando a carga entregue a uma rede é superior à capacidade da mesma. Neste tipo de situação deve ser diminuída a taxa de transmissão. É sabido que a implementação do protocolo TCP consegue uma boa gestão de congestionamento, como tal, após estudo da mesma, acreditamos que permite os melhores resultados. Como tal adoptamos o modelo de controlo de congestão que o TCP utiliza. A ideia passa por trabalhar com uma janela dinâmica.

4.1 Aumentar o tamanho da janela

Por cada pacote recebido no servidor é devolvida uma confirmação ao cliente. São aguardados um número de confirmações igual ao tamanho da janela, ao fim das quais a mesma aumenta. Em caso de *timeout* ou 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.

4.2 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 1.1: Implementação do algoritmo 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);  
}
```

4.3 Conclusão

Neste projeto foi necessária a construção de uma camada adicional ao protocolo UDP para que nos fosse possível resolver o problema proposto. Após a elaboração deste projeto foi-nos possível observar como é possível regular o débito dos dados a transmitir em função dos níveis de congestão da rede. Ao longo deste trabalho fomos encontrando algumas dificuldades, sendo estas ultrapassadas. A maior dificuldade encontrada foi controlar a falta de ordem de chegada de confirmações. Pode-se então concluir que o projeto foi bem elaborado e foi concluído com sucesso.