UDP++

André Pimenta, João Gomes, and Daniel Santos

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a54745,a54738,a54825}@alunos.uminho.pt

Abstract. Este relatório apresenta todos os passos da construção da aplicação **UDP++** no âmbito da disciplina de Comunicação de Computadores.
O projecto **UDP++** consiste numa aplicação que implementa a camada protocolar sobre UDP, mas com o incremento da capacidade de regular o debito de dados a ser transmitidos de forma a controlar possíveis congestões na rede.
A aplicação **UDP++** deverá ser capaz de transferir ficheiros seguindo um modelo de cliente-servidor usando o protocolo UDP.
Ao longo deste relatório serão apresentados todos os passos e heurísticas da construção da aplicação, assim como o seu funcionamento.

1 Introdução

O User Datagram Protocol (UDP) é um protocolo simples da camada de transporte não orientado à conexão. Este permite que uma aplicação transmita datagramas entre diferentes maquinas, não garantindo no entanto se o pacote chega ao destino.
Ao contrario do Transmission Control Protocol (TCP), o UDP é um protocolo que não possui mecanismos de controlo de erros , não tem controlo de sequência dos pacotes de dados enviados nem procede a retransmissões de dados.
Podemos então dizer que é protocolo que não garante fiabilidade ao contrario do TCP, porém devido a alguns dos factos referidos, entre outros o protocolo UDP têm cabeçalhos menores o que o leva a um melhor desempenho e maior rapidez de transmissão consequentemente.
Analisando as características do UDP é fácil concluir que este é uma escolha adequada para a transferência de fluxos de dados em tempo real, principalmente aqueles que admitem a perda substancial de dados, sendo que esta não deverá afectar o seu bom funcionamento, exemplo disso são as streams de vídeo ou voz onde se pede rapidez de transferência.
Como já referimos o UDP contrariamente ao TCP não possui um mecanismo de controlo de fluxo o que pode levar por vezes a um congestionamento da rede, e apesar destes dois conceitos serem totalmente distintos estão muito relacionados.
Neste contexto foi pedida a implementação de uma aplicação (**UDP++**) que vai incutir um mecanismo de controlo de congestão de rede sobre o protocolo UDP, que seguindo uma heurística que por nós será apresentada neste relatório, que vai controlar a quantidade de dados a ser enviados de forma a tornar a aplicação mais amigável para a rede.
Vamos então procurar estabelecer um equilíbrio entre o serviço de conexão UDP e um equilíbrio na rede que irá trazer vantagens tanto na rede porque controla o congestionamento desta, mas também por tornar a aplicação minimamente mais fiável, pois sempre que se notarem grandes perdas de pacotes irá se fazer um controlo de congestão sobre a rede.
UDP++ será então uma aplicação de partilha de arquivos implementada sobre o protocolo UDP mas com um maior controlo de dados e de efeitos sobre a rede.

2 Requisitos e objectivos

Nesta secção apresentamos os requisitos que a aplicação **UDP++** deve cumprir e que foram propostos no enunciado deste trabalho prático, para além dos objectivos pessoais como a aprendizagem e consolidação das diferentes matérias e termos técnicos que são abordados ao longo deste relatório.
Os requisitos que deve apresentar a aplicação **UDP++** são os seguintes:

- Providenciar o estabelecimento e termino fiável de uma conexão
- Oferecer um serviço de transporte de dados, em que os dados a transmitir são vistos como uma stream de pacotes
- Cada pacote recebido deve ser individualmente confirmado
- Implementar um mecanismo de controlo de congestão que ajuste a taxa de transmissão de envio de pacotes com base numa heurística eficiente
- Incluir mecanismos para evitar conexões não terminadas
- Continuar a ser um protocolo de transporte não fiável e sem incluir qualquer tipo de retransmissão
- Continuar a ser um protocolo de trasnporte que não garanta entrega ordenada

3 Especificação do protocolo de comunicação

3.1 Primitivas de comunicação

Um dos requisitos da implementação do protocolo **UDP++** era a implementação de um mecanismo de controlo de congestão, de forma a ser possível controlar o débito dos pacotes do transmissor. Assim na implementação do transmissor decidimos criar três entidades que permitam controlar o débito de envio de pacotes:
A primeira entidade guarda todas as informações da ligação de um utilizador, como por exemplo a taxa de envio, o tempo de ida e volta de um pacote, etc, o que vai permitir às outras entidades a realização de operações como a alteração da taxa de envio. A segunda entidade é responsavel por enviar os dados para o cliente com base no débito disponibilizado pela primeira entidade. Por fim a terceira entidade é a responsavel por receber o feedback do utilizador, que irá permitir confirmar os pacotes recebidos pelo cliente, a confirmação do estabelecimento e término da comunicação de forma fiável, e ainda terminar a ligação se ocurreu timeout.
Estas três entidades formam a base do protocolo que tem na capacidade de adapação à rede a sua principal vantagem em relação a outras implementações de UDP.

3.2 Pacotes de dados - PDUs

Foram concebidos dois tipos de pacotes de dados, um para o transmissão de dados e outro para a gestão da ligação. A estes dois tipos de pacotes de dados foi adicionado um overhead adicional que identifica o tipo do pacote (dados, ACK dados, terminar conexão, etc).
O primeiro é composto pelo cabeçalho de um byte que identifica o pacote de dados e por 1023 bytes destinados a encapsular os dados provenientes do ficheiro a transmitir. Definimos este comprimento para que fosse evitada a fragmentação e para que a perda de um pacote não tivesse grande impacto na transmissão do ficheiro.
O segundo tipo de pacotes é por sua vez composto apenas pelo cabeçalho de um byte, que identifica o tipo do pacote.

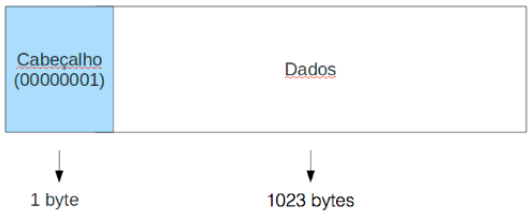


Fig. 1. Pacote de dados

Cada um dos diferentes tipos de pacotes de dados é identificado pela sua flag. A seguir apresentamos uma lista com as diferentes flags que identificam os tipos de pacotes de dados disponíveis no **UDP++** e as sua respectiva utilidade.

– **Providenciar o estabelecimento e termino fiável de uma conexão**

  - 00000001 -> Flag usada para informar intenção de iniciar uma conexão
  - 00000010 -> Flag usada para notificar o fim do envio de dados
  - 00000101 -> Flag usada para confirmar fim da conexão

– **Oferecer um serviço de transporte de dados, em que os dados a transmitir são vistos como uma stream de pacotes, sendo que cada pacote recebido deve ser individualmente confirmado.**

  - 00000011 -> Flag usada para identificar envio de pacote de dados
  - 00000110 -> Flag usada para confirmar a recepção de um pacotes de dados

– **Mecanismo de controlo de congestão que ajusta a taxa de transmissão de envio de pacotes.**

  - 00000100 -> Flag usada para a medição do round trip time
  - 00000111 -> Flag usada para comunicar a recepção de pacote de medição do RTT

Estas são as sete primitivas que consideramos fundamentais para o funcionamento dos mecanismos propostos para o **UDP++**, e que nos vais garantir o controlo de congestão da rede, mas ao mesmo tempo uma maior fiabilidade da aplicação.
Sendo que necessitamos de sete primitivas, estas podem ser representadas por 3 bits

$$(2^3 = 9) > 7$$

que permite representar todas estas, no entanto adicionamos mais 5 bits (total de 8 bits) por conveniência, diminuindo assim a percentagem de possíveis erros que possam ocorrer durante a transmissão , mas também porque 1 byte é constituído por 8 bits e facilitou a sua implementação.

3.3 Regras de funcionamento geral

Do lado do servidor é criada uma thread para a recepção de pacotes na porta 5000. Sempre que recebe um pedido de conexão o que o servidor faz é abrir um socket numa outra porta para que possa estar a enviar dados para o cliente e em simultaneo estar disponivel para receber novos pedidos.
Depois de criar um socket numa nova porta, cria as restantes primitivas que já indicamos e começa a enviar de imediato pacotes de dados para o cliente que se vai aperceber de qual é a nova porta para a qual tem de comunicar quando receber os pacotes. Cada pacote de dados a ser enviado levará um cabeçalho com a flag dos pacotes de dados e no restante espaço dados do ficheiro a transmitir como já referimos. Os pacotes de dados são enviados em grupos com um diferente espaçamento temporal calculado com base no RTT e na taxa de débito do servidor.
Sempre que um pacote de dados é recebido, são enviadas tramas de confirmação ACK pelo

receptor de dados, para posterior tratamento de estatísticas sobre pacotes confirmados. Do lado do servidor esse pacotes são recebidos e num intervalo de tempo de um segundo é renovada a nova taxa de débito de informação com base no numero de pacotes enviados e no número de confirmações dos pacotes de dados enviados pelo cliente.
Depois de enviados todos os pacotes de dados é necessário terminar a ligação, para isso são enviados pacotes a pedir o fim ligação, até que o cliente envie um pacote com a flag de confirmação de término da conexão.
Pode acontecer de o servidor deixar de receber resposta por parte do cliente. Nesse caso está definido um limite para timeout, que sempre que ocorre leva a que o servidor deixe de transmitir dados para o cliente e este seja forçado a criar uma nova ligação.
Com as primitivas de comunicação e regras de funcionamento apresentadas até este momento conseguimos controlar inicio e fim de ligações de forma fiável e segura, conseguimos também a confirmação dos pacotes recebidos, no entanto ainda não conseguimos controlar de forma eficaz a congestão da rede.

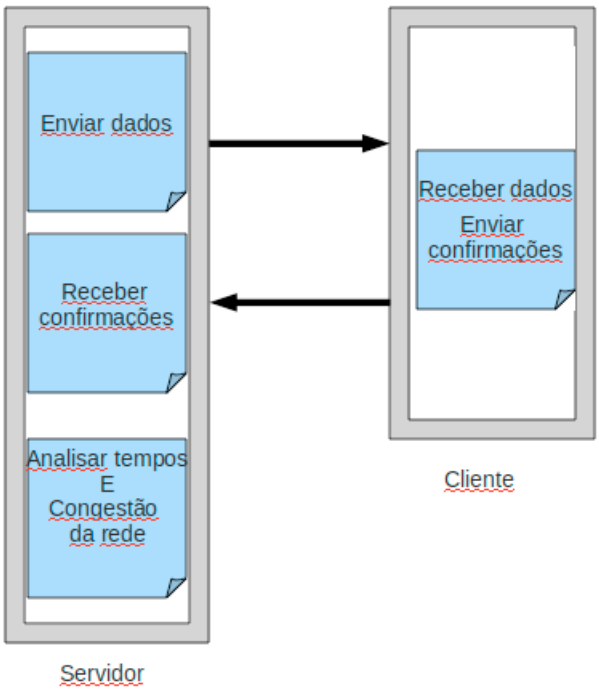
3.4 Mecanismo de controlo de congestão

Um mecanismo de controlo de congestão deve ter várias caracteristicas das quais podemos destacar a rápida adaptação às condições da rede. O mecanismos de controlo de congestão deve estar atento aos sinais que a rede lhe dá. Se a perda de pacotes começa a aumentar isso pode ser um sinal de que a rede começa a estar congestionada. Tendo em conta estes aspectos, definimos que o nosso mecanismo teria de ser baseado na percentagem de perdas e no taxa de débito actual.
O envio de pacotes é feito em grupo e o que o mecanismo de controlo de congestão faz é gerir o tempo entre o envio de cada um desses grupos. Assim em casos de congestão esse tempo será grande, por outrolo lado quando existir grande larguara de banda o tempo de envio entre grupos de pacotes será reduzido.
A cada segundo é recalculado o débito e para isso é usada a percentagem de perdas (Lr) calculada com base no número de pactoes enviados e no numero de confirmações recebidas.
É definido um limite mínimo de perdas (MIN_LOSS), que permite identificar o ponto apartir do qual o débito de envio deixa de ser incrementado, e um limite máximo de perdas (MAX_LOSS), que permite identificar a existência de congestão e então baixar o débito.
Baseado na percentagem de perdas e nos limites definidos o mecanismo de controlo de congestão vai ajustando o débito de forma continua. Quando o servido se apercebe da ocurrencia de congestão, ou seja, a percentagem de perdas excede o limite máximo de perdas, o débito baixa drásticamente para tentar "aliviar a rede". Depois da ocurrencia de congestão esse facto é registado e o servidor começa a aumentar o débito de forma progressiva com base num factor I, calculado com base num valor de débito pré-definido e na percentagem de perdas, enquanto a percentagem de perdas não exceder o valor mínimo. Quando a percentagem de perdas se encontra entre o valor mínimo e máximo de perdas os servidor mantém o débito pois este é um valor que permite o envio dos dados com uma percentagem de erro aceitável.

O numero de grupos de pacotes enviados por segundo é calculado com base no débito do servidor e também com base no RTT.

4 Implementação da aplicação

Para a implementação da aplicação **UDP++** decidimos usar a plataforma JAVA.
Dentro das bibliotecas oferecidas pelas linguagem encontrar a class "DatagramSocket" que permite criar sockets UDP para fazer transfereências de pacotes UDP.
Utilizamos então a class DatagramaSoctek para criar socket UDP e ainda a class DatagramPacket que nos permite criar os pacotes UDP.
Para tornar a aplicação mais eficaz , o servidor cria duas threads por cada ligação, uma para enviar os pacotes e outra para receber o feedback, e ainda uma entidade(class responsavel por guardar as estatisticas da ligação).
Do outro lado o cliente terá apenas uma responsável por receber os pacotes e enviar os ACKs.
Esta é a forma como se encontra estruturada a nossa aplicação e como podemos ver de forma mais detalhada na figura seguinte.



X

Com a estrutura apresentada em cima, esperamos ter uma aplicação eficaz, capaz de controlar da forma mais rápida e eficaz qualquer possível congestão na rede, mas também que permita o envio rápido de dados caso não cause congestionamento da rede.

5 Testes e Estatísticas

Depois de implementada a aplicação passamos à fase de testes para testar o mecanismos de controlo de congestão e todas os outros mecanismos implementados. O que observamos é que o mecanismo funcionava tal como esperado para a generalidade dos diiferentes débitos de transmissão.
Apresentamos de seguida laguns dos testes realizados:

```
Servidor is listening on port 5000
```

Fig. 2. Inicio do servidor à escuta na porta 5000

```
Servidor is listening on port 5000
Iniciei Feedback
renovar rate
Enviados: 5010.0 Recebidos: 4983.0 Rate: 40000.0 Perdas: 0.01
renovar rate
Enviados: 5000.0 Recebidos: 4680.0 Rate: 40000.0 Perdas: 0.06
RTT: 0
renovar rate
Enviados: 2000.0 Recebidos: 1966.0 Rate: 18000.0 Perdas: 0.02
RTT: 0
renovar rate
Enviados: 2000.0 Recebidos: 2000.0 Rate: 18000.0 Perdas: 0.0
renovar rate
Enviados: 2500.0 Recebidos: 2500.0 Rate: 19000.0 Perdas: 0.0
renovar rate
Enviados: 2510.0 Recebidos: 2320.0 Rate: 20000.0 Perdas: 0.08
```

Fig. 3. Acção do mecanismo de controlo de congestão

```
Servidor is listening on port 5000
Iniciei Feedback
renovar rate
Enviados: 5020.0 Recebidos: 4992.0 Rate: 40000.0 Perdas: 0.01
RTT: 0
renovar rate
Enviados: 4980.0 Recebidos: 4332.0 Rate: 40000.0 Perdas: 0.13
renovar rate
Enviados: 2010.0 Recebidos: 1236.0 Rate: 18000.0 Perdas: 0.39
renovar rate
Enviados: 1000.0 Recebidos: 0.0 Rate: 8100.0 Perdas: 1.0
renovar rate
Enviados: 460.0 Recebidos: 0.0 Rate: 3645.0 Perdas: 1.0
renovar rate
Enviados: 200.0 Recebidos: 0.0 Rate: 1640.0 Perdas: 1.0
renovar rate
Enviados: 100.0 Recebidos: 0.0 Rate: 738.0 Perdas: 1.0
Timeout
Terminou FeedBack
```

Fig. 4. Timeout

6 Conclusão

Findo o desenvolvimento da aplicação **UDP++** podemos tirar algumas conclusões, tanto a nível dos protocolos mencionados ao longo deste relatório como a nível dos conhecimento postos em causa ao longo do desenvolvimento deste.
Inicialmente expusemos as principais características do protocolo UDP e referenciamos as falhas deste relativamente ao protocolo TCP.
Falhas estas que a aplicação **UDP++** pretende colmatar tendo sempre em conta de que estamos perante uma entidade protocolar que deve continuar a ser um protocolo de transporte não fiável e que não garanta entrega ordenada.
Posto isto podemos concluir que o protocolo UDP não é o melhor protocolo, e que é possuidor de enumeras falhas, não deixando de ser o mais adequado para determinadas tarefas pelo qual este foi desenvolvidas. Tarefas estas que o protocolo TCP não desempenha com rigor , chegando mesmo a ser considerado uma má solução, e estamos a falar de transmissões rápidas e sem repetição de pacotes a enviar.
Posto isto a aplicação **UDP++** que suporta uma entidade protocolar baseado em UDP mas que garante melhorias na transmissão e melhor controlo de congestão de rede, pode ser vista como uma óptima solução para a transmissão de pacotes de dados em redes com elevada probabilidade de congestão, evitando assim uma grande perda de dados , apesar de uma pequena perda de velocidade de transmissão.
Falando agora um pouco a nível de conhecimentos e dificuldades que foram postos em causa ao longo do desenvolvimento deste trabalho, podes evidenciar alguma dificuldade na programação deste em JAVA, principalmente o facto de não ser nada fácil trabalhar com bits nesta mesma plataforma e o facto de a biblioteca "DatagramaSocteck" ser nova para nós. No entanto todas estas dificuldades foram ultrapassadas e o resultado final cumpre todos os objectivos propostos inicialmente.

References

1. Harold, R.: Java Network Programming. O'Reilly Media (2004)
2. Mueller's, S.: Upgrading and Repairing Networks. Que (2001)

Referências WWW
http://ieeexplore.ieee.org/