Serviço de transferência rápida e fiável de dados sobre UDP

Relatório do trabalho prático

Grupo 9
Joana Cruz(A76270)
Etienne Costa (A76089)
Hugo Moreira (A43148)

Maio 2019

Universidade do Minho
Escola de Engenharia

Comunicações por Computador
Mestrado Integrado em Engenharia Informática
Universidade do Minho

1 Introdução

Este projeto surge no âmbito da Unidade Curricular de Comunicações por Computador e tem como objetivo a implementação de um serviço de transferência rápida e fiável de dados sobre UDP. Sendo que a comunicação deverá ser feita em unidades de dados que caibam dentro de um datagrama UDP e que sejam enviados e recebidos via Sockets UDP. Este serviço de transferência é caracterizado por três fases durante uma conexão: início da conexão, transferência de dados e término da conexão.

2 Especificação do protocolo

Sendo um protocolo uma convenção que controla e possibilita uma conexão, comunicação e transferência de dados entre dois sistemas computacionais, dedicou-se esta secção para explicar o formato das mensagens protocolares bem com as suas interações.

2.1 Proposta do PDU

O serviço de transferência rápida e fiável de dados foi implementado, a nível aplicacional, com recurso ao UDP. Sendo assim houve a necessidade de definir um datagrama UDP que possui as seguintes características :

- **Sequence Number**:Valor utilizado para garantir a entrega ordenada e robustez durante a transferência.
- **Acknowledge**: Valor utilizado para confirmar a entrega de um segmento.
- **Flag Type**: Valores utilizados para indicar um estado particular da conexão ou para fornecer informação adicional. Portanto, podem ser usados para fins de solução de problemas ou para controlar como uma determinada conexão é tratada.
  De seguida apresenta-se os possíveis valores para estas flags:
  1. **SYN** - início de uma conexão
  2. **ACK** - confirmação da recepção de um segmento
  3. **PSH** - segmento com dados de um ficheiro
  4. **FIN** - término de uma conexão
  5. **SYN+ACK** - início de uma conexão
  6. **JOE DOWN** - download de um ficheiro
  7. **JOE UP** - upload de um ficheiro
  8. **EXIT** - objetivo de terminar uma conexão
  9. **LIST** - listar os ficheiros
  10. **END TRANSF** - término de uma transferência

- **Port Number**:Valor que indica a nova porta que o cliente irá responder após ser efetuado o início de uma conexão.
- **Window Size**:Indica em bytes a quantidade de informação que este pode enviar nos próximos pacotes, durante uma conexão.
- **Length Data**: Corresponde em bytes ao tamanho do pacote a ser enviado.
- **CheckSum**: Valor utilizado para verificar a integridade de dados transmititos.
- **File Data**: Corresponde em bytes a informação a ser transferida.

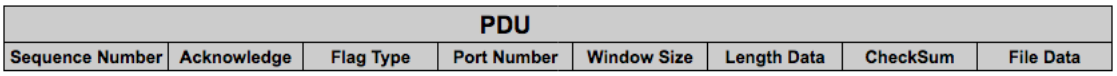

Figura 1: Protocol Data Unit

2.2 Interação

O nosso sistema de transferência é caracterizado por três fases distintas durante uma conexão:

1. **Início da conexão** Para o início de uma conexão o nosso sistema foi baseado no 3-Way Handshake do TCP.
   Sendo que esta conexão procede do seguinte modo:

Início da Conexão
Host A sends **SYN**chronize packet to Host B
Host B receives **SYN**
Host B sends **SYN**chronize-**ACK**nowledgment
Host A receives **SYN-ACK**
Host A sends **ACK**nowledgment
Host B receives **ACK**

Tabela 1: Início da conexão

2. **Transferência de Dados** Para a transferência de dados de forma fiável adoptamos o seguinte mecanismo:
   - O Transmissor vai enviando os segmentos de dados, em que a cada segmento enviado o número do sequência corresponde ao número do segmento
   - O Receptor vai enviando as confirmações dos segmentos recebidos e que não estão corrompidos

Uma transferência de dados termina quando o transmissor recebe todos os Acknowledges dos segmentos enviados. Adoptamos um mecanismo de retransmissão seletiva, em que apenas serão retransmitidos os segmentos de dados do ficheiro que não foram entregues ou que estavam corrompidos. Posteriormente esta explicação encontra-se mais detalhada. Agora apresentamos um pequeno exemplo de execução da nossa aplicação de um início de conexão, transferência de um ficheiro(download) e término de uma conexão.

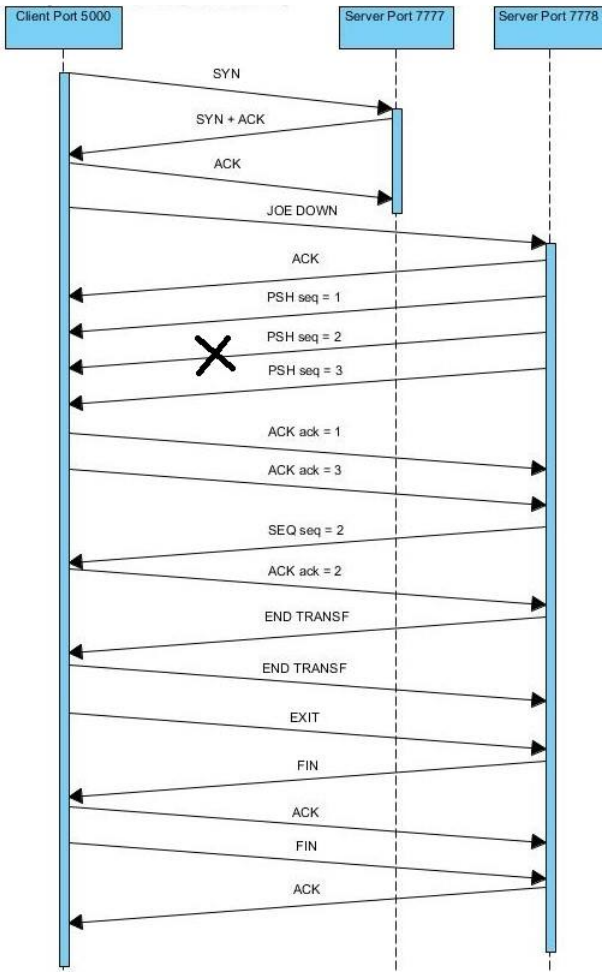

Figura 2: Diagrama temporal de uma transferência

3. **Término da conexão** Para o término de uma conexão ,o nosso sistema foi baseado no TCP,sendo que este término é um processo de quatro fases, em que cada sistema computacional é responsável pelo encerramento do

seu lado da ligação.

Término da Conexão
Host A sends a **FIN**ished packet to Host B
Host B receives **FIN**ished
Host B sends **ACK**nowledgment
Host A receives **ACK**nowledgment
Host B sends **FIN**ished packet to Host A
Host A receives **FIN**ished
Host A sends **ACK**nowledgment
Host B receives **ACK**nowledgment

Tabela 2: Término da conexão

3 Implementação

3.1 Arquitetura da solução

A arquitetura da nossa solução difere um pouco da pedida no enunciado, dado que estruturámos o nosso problema como sendo Servidor e Cliente. O Servidor encontra-se sempre à escuta na porta 7777 de pedidos de conexão de clientes. Quando um cliente se pretende conectar, o servidor terá que aceitar a conexão pedida, e aí dá-se o mecanismo de início de conexão descrito anteriormente. Após este mecanismo, o Servidor abre um novo Socket que será responsável de satisfazer os pedidos do cliente, podendo aceitar pedidos de outros clientes pois a porta 7777 assim continua à escuta de outros pedidos de conexão. O cliente após estar conectado poderá listar os ficheiros disponíveis no Servidor, fazer download ou upload de um ficheiro, e terminar uma conexão.

3.2 Estrutura de dados

Para a implementação destes conceitos, foi utilizada a linguagem de programação orientada a objectos *Java*, e foram definidas as seguintes classes.

Classes relacionadas com o Cliente:

- **ClientMain** - Classe main do cliente
- **ClientAgentUDP** - Classe principal do cliente que contém métodos de início e término de conexão, assim como download e upload de ficheiros.

Classes relacionadas com o Servidor:

- **ServerMain** - Classe main do Servidor
- **ServerAgentUDP** - Thread responsável por estar sempre à escuta de pedidos de conexões de clientes distintos
- **ClientHandler** - Thread responsável por estar à escuta de diversos pedidos de um cliente

Classes comuns ao Servidor e Cliente:

- **PDU** - classe que corresponde à estrutura de PDU que irá ser enviada nos Sockets UDP
- **Resources** - classe com os recursos de uma conexão, contém variáveis como o socket, a porta de envio, o endereço IP e todos os métodos associados ao envio e recepção de um segmento
- **FileSender** - Thread responsável por mandar um ficheiro
- **AckReceiver** - Thread responsável por receber os ACKs

- **FileReceiver** - Thread responsável por receber segementos correspondentes a um ficheiro
- **AckSender** - Thread responsável por mandar ACKs dos segmentos do ficheiro que já recebeu

3.3 Modo de funcionamento

Como já foi referido anteriormente o Servidor encontra-se sempre à escuta na porta 7777. Após efetuar o início de uma conexão, abre um novo Socket, e a Thread ClientHandler estará sempre à escuta de pedidos desse cliente. Conforme o pedido efetuado o procedimento vai ser diferente. Aqui iremos apresentar o procedimento caso o cliente peça um download de um ficheiro. Então quando o cliente pede um download 4 novas threads irão começar a trabalhar, duas no lado do Servidor e duas no Cliente. Cada par trabalha em conjunto, como um Receptor e um Transmissor.

No lado do Servidor:
Então no Servidor que neste caso será o Transmissor, a Thread fileSender estará responsável por enviar PDUs com a informação do ficheiro no máximo 1024 bytes de cada vez, enquanto a thread AckReceiver estrá responsável por estar à escuta, esperando por segmentos que sejam Acknowledges recebidos pelo Cliente. Estas 2 Threads partilham uma estrutura de dados(Thread safe), sendo que sempre que a FileSender envia um segmento com o respetivo número de sequência adiciona ao HashMap(o número de sequência e o respetivo PDU), ao mesmo tempo que a AckReceiver irá retirar desse HashMap sempre que recebe um Acknowledge. Após a FileSender ter enviado todos os segmentos do ficheiro, irá apenas retransmitir os segmentos dos quais não recebeu ACKs. Quando esta estrutura estiver vazia quer dizer que todos os segmentos foram entregues ao Receptor, e será enviado um PDU com a informação que a transferência foi efetuada com sucesso.

No lado do Cliente:
No Cliente estará à escuta(esperando por receber segmentos) a Thread FileReceiver, e responsável por enviar PDUs de confirmação será a AckSender. O mecanismo é o mesmo utilizado no Servidor, ambas as Threads partilham de uma estrutura, e a FileReceiver vai adicionando à estrutura, enquanto a AckSender retira da estrutura os ACKs que enviou. Quando a FileReceiver receber um PDU com a flag END TRANSF, saberá que a transferência foi efetuada com sucesso, e enviará uma confirmação.

Observações:

- Para controle de erros: A FileSender calcula sempre o checksum antes de ser enviado o segmento, incluindo esse valor no PDU enviado. Enquando que a FileReceiver, quando recebe o segmento faz esse mesmo

exato cálculo, e verifica se o pacote não está corrompido, e apenas adiciona à estrutura partilhada se o pacote não estiver corrompido. Para o cálculo do checksum usou-se uma classe já pré-definida pelo Java e a sua respetiva API - CRC32.

- Para controlo de fluxo: Para que o emissor não extravase o buffer do emissor enviámos 10 segmentos de cada vez, adormecendo a FileSender durante milisegundos, e reenviando o resto dos segmentos outra vez.

De seguida apresentamos uma imagem ilustrativa do processo descrito.

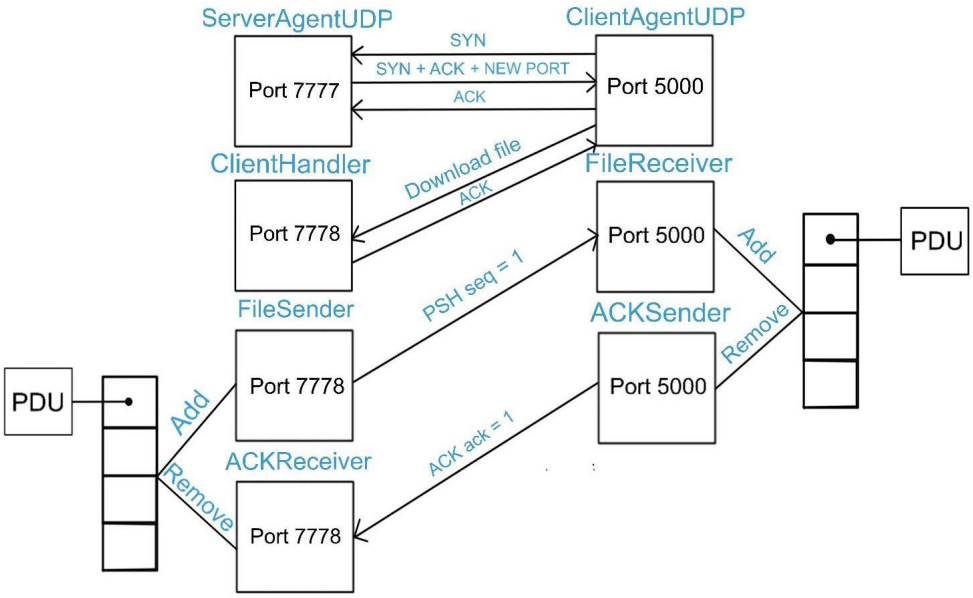

Figura 3: Arquitetura da solução implementada

Caso o pedido fosse um upload do ficheiro, o mecanismo seria o inverso, dado que o Cliente funcionava como Transmissor e o Servidor como Receptor.

3.4 Métodos

De modo a não tornar esta leitura exaustiva , decidiu-se simplesmente apresentar os métodos mais relevantes e procurar explicar o impacto que cada um tem na implementação do nosso serviço.
Na classe PDU, estes métodos foram necessários para troca de segmentos UDP:

1. **ByteToPDU** - converte um array de bytes num objeto PDU
2. **PDUToByte** - converte um objeto PDU num array de bytes

Na classe Resources:

1. **receive** - Método que recebe um DatagramPacket, e converte os bytes recebidos num PDU
2. **receive(int timeout)** - Igual ao método anterior, mas agora com um timeout pré-estabelecido para receber um segmento
3. **send(PDU packet)** - Método que converte um PDU num array de bytes, e envia um DatagramPacket
4. **sendAndExpect(PDU packet, int timeout, int attemps)** - Método que envia um PDU, e caso não receba um segmento de volta durante um perído de tempo, tenta enviar o segmento X vezes até receber um segmento de volta.

Na classe ClientAgentUDP:

1. **connect** - Método de conexão do cliente
2. **disconnect** - Método de término de conexão do cliente
3. **downloadFile** - Método em que o cliente funciona como receptor
4. **uploadFile** - Método em que o cliente funciona como transmissor

Consíderamos o método sendAndExpect o mais importante, pois dado que o UDP não garante a recepção de segmentos, também teríamos que considerar que poderiam haver perdas de pacotes no ínicio de uma conexão, num pedido de um cliente, no término de uma conexão. O nosso serviço está preparado para perda de qualquer segmento, e não apenas durante uma transferência.

3.5 Bibliotecas de suporte

Sendo java uma linguagem de programação bastante versátil foi possível importar todas as bibliotecas que permitissem a realização deste projecto,sendo elas as seguintes:

- java.net.InetAddress;
- java.util.concurrent.ConcurrentHashMap;
- java.util.concurrent.atomic.AtomicBoolean;
- java.util.Scanner;
- java.net.UnknownHostException;
- java.net.SocketException;
- java.io.IOException;

- java.net.SocketTimeoutException;
- java.io.FileOutputStream;
- java.io.FileInputStream;
- java.util.Arrays;
- java.util.zip.CRC32;
- java.nio.ByteBuffer;
- java.net.DatagramPacket;
- java.net.DatagramSocket;

3.6 Comandos

O cliente na nossa aplicação poderá utilizar os seguintes comandos:

- list - lista os ficheiros disponíveis no Servidor
- download <fileLocalName> ou download <fileLocalName> <fileRemoteName> - download de um ficheiro, com possibilidade de escolha de um nome remoto
- upload <fileLocalName> - upload de um ficheiro para o Servidor
- help - menu de ajuda para visualizar os comandos disponíveis
- exit - término de uma conexão, e consequentemente fecho do socket

4 Testes e Resultados

Nesta secção apresentamos alguns testes efetuados no core que achamos relevantes. Primeiramente o início de conexão de 2 Clientes, sendo que um encontra-se numa ligação segura, e outro com perda e duplicação de pacotes.

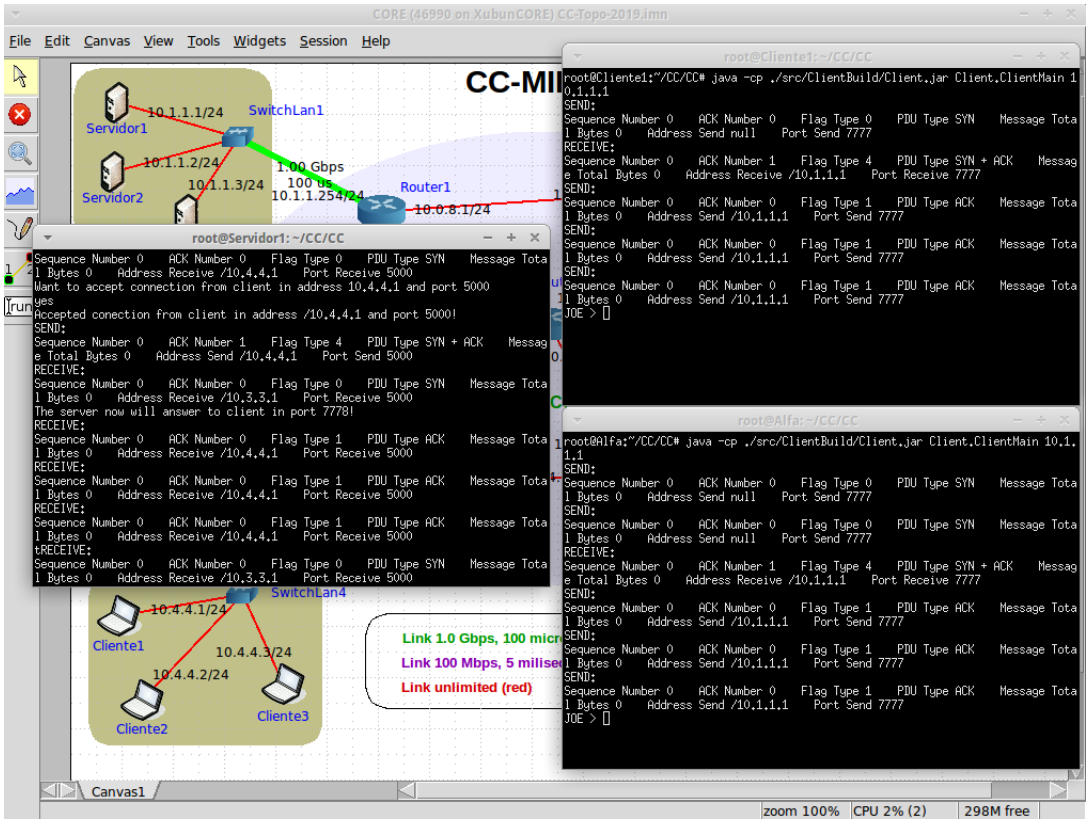

Figura 4: Conexão de clientes

Posteriormente apresentamos um dos clientes a pedir a lista de ficheiros disponíveis no Servidor.

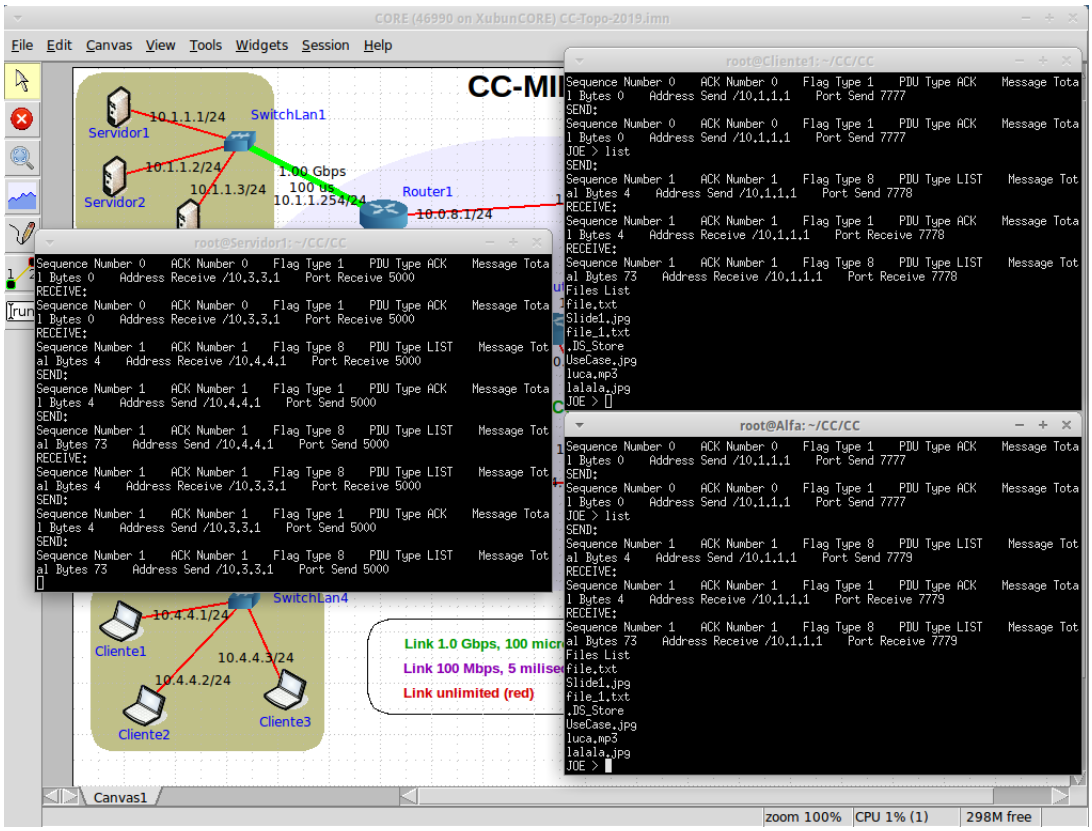

Figura 5: Pedido da lista de ficheiros

Por último, a execução do download de um ficheiro MP3 pelos 2 clientes no mesmo instante.

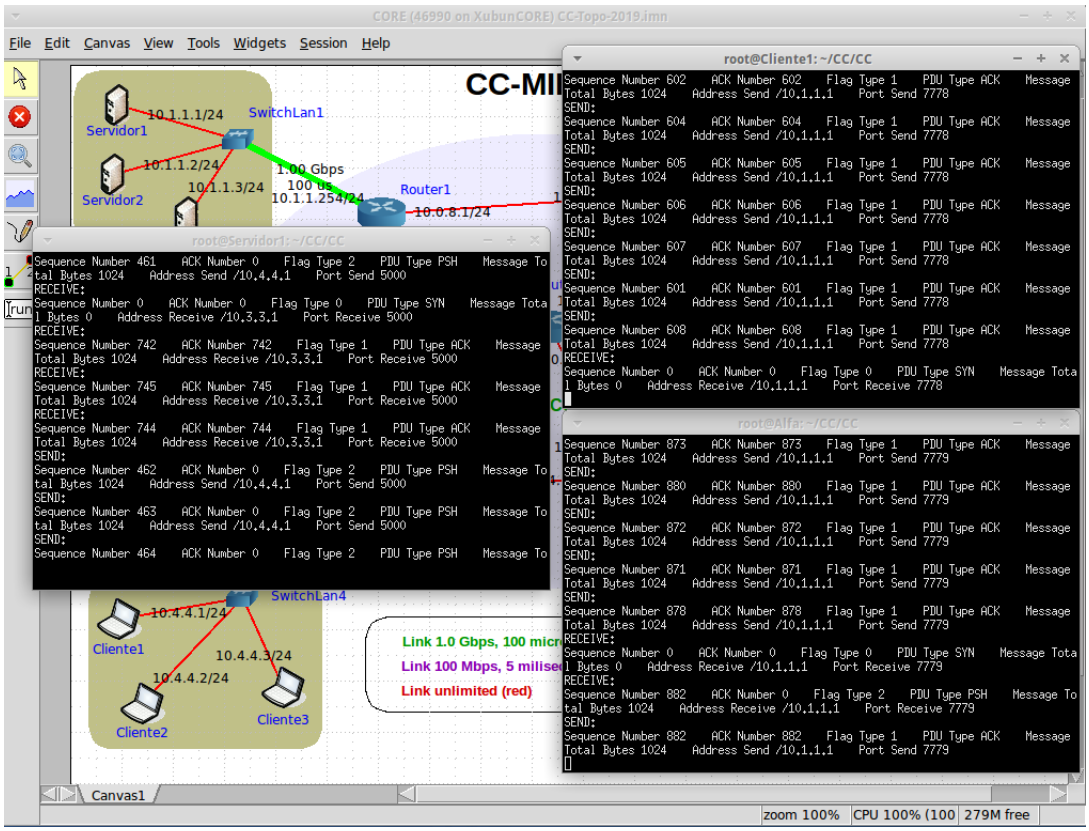

Figura 6: Pedidos de transferência de ficheiro

Para uma fase de desenvolvimento e de testes decidimos demostrar os prints de todos os segmentos enviados e recebidos tanto pelo Servidor e pelos Clientes que funciona como a nossa Tabela de Estado. Sendo que estes prints apresentam sempre a informação do número de sequência, número de ACK, tipo de PDU, tamanho dos dados transferidos, porta e o endereço IP.

5 Conclusões e trabalho futuro

Inicialmente tivemos algumas dificuldades na estruturação do trabalho, nomeadamente qual o mecanismo que adoptaríamos para que a recepção de qualquer segmento enviado fosse de forma fiável, assim como efetuar as retransmissões, que após uma análise detalhada do problema foram ultrapassadas. Implementámos todas as funcionalidades obrigatórias propostas, dado que a solução de transferência de um ficheiro não foi a ideal dado o espaço em memória utilizado pelas estruturas partilhadas pelas Threads. Por outro lado, consíderamos a nossa solução eficiente pois em qualquer transferência, tanto no Transmissor como no Receptor, temos uma Thread responsável pela recepção de segmentos e outra pelo envio. Quanto às funcionalidades adicionais apenas as múltiplas conexões em simultâneo foi implementada, sendo assim numa perspectiva de trabalho futuro serão implementas mais funcionalidades de modo a tornar o sistema mais complexo.