



FCTUC

Computer Networks – 2016/2017

Assignment for evaluation

TestConnection

Class:PL3

José Castanheira
2013139490
a2013139490@alunos.deec.uc.pt

João Ferreira
2013139657
a2013139657@alunos.deec.uc.pt

30 de Dezembro de 2016

1 Notas

O problema explicito no enunciado foi resolvido de duas maneiras diferentes que se encontram nas suas respectivas pastas, o funcionamento dos programas é igual, mas numa das versoes os peers estão separados em dois programas diferentes e noutra estão juntos num só programa.

Entendemos 1 pacote como se fosse 1 sendto() de um inteiro.

Estes Programas devem ser corridos em ambiente Linux e podem ser compilados com um simples comando make a partir da directoria onde se encontram.

2 Manual de utilização

Neste trabalho era pedido que se simulasse uma ligação peer-to-peer. Para tal dividimos o trabalho em duas partes diferentes, uma é o peer onde são feitas as escolhas explíticas no enunciado e outra é o peer de teste de conexão.

A parte TestConnection, que na versão dois é um programa separado, apresenta um menu com uma serie de opções exigidas no enunciado entre elas a possibilidade de mudar o destino, escolher o numero de pacotes a enviar e escolher a porta de envio. O programa tem um funcionamento muito simples e passo agora a entrar em mais detalhe sobre como funciona. Logo ao entrar o utilizador vê um menu onde estão todas as opções.

A opção 0 é a opção de saída que, como o próprio nome indica, sai do programa.

A opção 1 é a opção de escolha de destino, isto é, o ip com quem é feita a ligação p2p. Nesta opção foi implementado um extra que permite que o utilizador insira o nome do host. Por defeito é feita uma ligação a localhost.

A opção 2 é a opção de envio de pacotes para o outro peer e depois mostra o resultado, isto é, o numero de pacotes enviados, o numero de pacotes recebidos e a percentagem. O funcionamento da comunicação p2p é simples e assim que é escolhida esta opção são criadas duas sockets, uma socket tcp e uma udp, é feita uma conexão inicial por tcp e são enviados os pacotes por udp, o outro peer recebe os pacotes e envia para o peer principal, por tcp ,quantos pacotes recebeu, tendo em conta estes dados são mostradas em ambos os computadores.

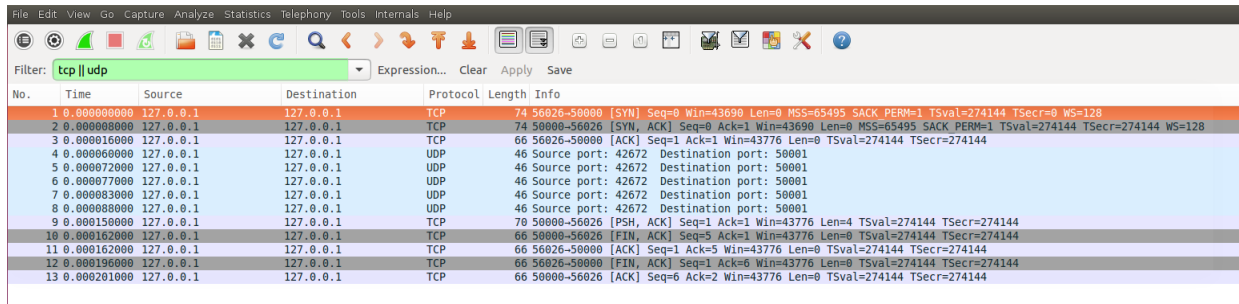
A opção 3 é a opção de seleccionar o numero de pacotes a enviar por defeito estão 5 pacotes .

A opção 4 permite a escolha da porta de teste entre o intervalo dito no enunciado.

A parte do outro peer ao executar, na versão onde estão separados em 2 programas, deve ser chamado da seguinte forma './Peer [porta]' sendo [porta] a porta de comunicação. O código e o funcionamento deste programa é muito simples e baseia-se basicamente em duas sockets, uma tcp e uma udp. Este programa começa por criar uma socket tcp que se conecta ao outro host com o comando accept. Seguidamente cria a socket udp e recebe os pacotes. No fim envia quantos pacotes lhe chegaram através da socket tcp e mostra-o ao utilizador, que poderá ler os resultados no programa original e neste.

Os pacotes udp enviados contêm sempre um numero inteiro que será 1143 excepto o ultimo que é 1337 e é usado para o segundo peer saber quando para de receber pacotes. Após a finalização das transferências, são feitos os closes e o segundo Peer desliga-se, necessitando de o utilizador o voltar a ligar de novo. Recomenda-se que se dê uma outra porta que não a mesma à execução deste se o tempo entre as duas for breve, porque o sistema operativo demora um bocado a fazer release da porta usada.

3 Captura em Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	50000->50000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=274144 TSecr=0 WS=128
2	0.000016000	127.0.0.1	127.0.0.1	TCP	66	50000->50000 [ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=274144 TSecr=274144 WS=128
3	0.000016000	127.0.0.1	127.0.0.1	UDP	46	Source port: 42672 Destination port: 50001
4	0.000060000	127.0.0.1	127.0.0.1	UDP	46	Source port: 42672 Destination port: 50001
5	0.000072000	127.0.0.1	127.0.0.1	UDP	46	Source port: 42672 Destination port: 50001
6	0.000077000	127.0.0.1	127.0.0.1	UDP	46	Source port: 42672 Destination port: 50001
7	0.000083000	127.0.0.1	127.0.0.1	UDP	46	Source port: 42672 Destination port: 50001
8	0.000088000	127.0.0.1	127.0.0.1	UDP	46	Source port: 42672 Destination port: 50001
9	0.000150000	127.0.0.1	127.0.0.1	TCP	70	50000->50026 [PSH, ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=274144 TSecr=274144
10	0.000162000	127.0.0.1	127.0.0.1	TCP	66	50000->50026 [FIN, ACK] Seq=5 Ack=1 Win=43776 Len=0 TSval=274144 TSecr=274144
11	0.000162000	127.0.0.1	127.0.0.1	TCP	66	50026->50000 [ACK] Seq=1 Ack=5 Win=43776 Len=0 TSval=274144 TSecr=274144
12	0.000196000	127.0.0.1	127.0.0.1	TCP	66	50026->50000 [FIN, ACK] Seq=1 Ack=6 Win=43776 Len=0 TSval=274144 TSecr=274144
13	0.000201000	127.0.0.1	127.0.0.1	TCP	66	50000->50026 [ACK] Seq=6 Ack=2 Win=43776 Len=0 TSval=274144 TSecr=274144

Figura 1: Captura no wireshark do programa em funcionamento

Como se pode ver pela figura 1 é inicialmente feita uma conexão TCP entre os peers.

Em seguida são enviados os pacotes UDP de teste. No programa utilizámos 5 pacotes e as portas foram: TCP-50000 e UDP-50001 pelo que os resultados obtidos são correctos.

Por fim são finalizadas as ligações que correspondem aos nossos closes(), evidenciadas pelos pares [FIN,ACK]/[ACK] no wireshark.