Projeto de Redes de Comunicação

Relatório

Departamento de Engenharia Informática

Trabalho realizado por:

Miguel Barroso 2019219010

Edgar Duarte 2019216077

Introdução

Este projeto foi feito no âmbito da disciplina de Redes de Comunicação e tem como objetivo demonstrar quatro tipos de ligações entre vários computadores em redes diferentes usando diversas técnicas: **1)** Configurações do Servidor numa CLI via TCP; **2)** Comunicação com o Servidor como intermediário; **3)** Comunicação direta P2P; **4)** Comunicação em grupo usando endereços *Multicast*.

Foi desenvolvido para executar na rede do GNS3, dependendo, portanto, das configurações da mesma.

Antes de mais nada, queremos salientar que tivemos alguns problemas de *routing* nos diversos routers do GNS e que tomámos algumas decisões nas suas configurações que não estão na meta 1. Enviámos um e-mail a descrever os dois problemas que tivemos e duas propostas de solução, mas não obtivemos qualquer resposta. Tendo isso em conta, assumimos as nossas soluções como viáveis e aplicámo-las ao projeto.

Problema 1 – Multicasting

Os PCs davam *join* ao grupo *multicast* (*ADD_MEMBERSHIP*) mas os pacotes não eram reencaminhados pelo seu router respetivo. Procedemos a várias tentativas de resolução deste problema – entre as quais: aumentar o TTL dos pacotes para 64 (estavam a 1); mexemos nas *flags* no C; tentámos criar *multicast routes*... – e nenhuma funcionou.

Este problema ficou resolvido, à última da hora, através das configurações que nos enviaram por notificação no inforestudante.

Problema 2 – Router 3 não encaminha algum tráfego

Ora, este problema é muito estranho. Isto porque algum tráfego é encaminhado pelo *R3* (proveniente do server) e outro não.

Funcional: o client1 envia um pacote UDP para o server e o server responde ao client1 sem problemas.

Não funcional: o client1 envia uma mensagem ao client2 através do *server*, pelo que envia um pacote UDP para o *server* e este deveria encaminhá-lo para o client2 (o que não acontece, não saía do R3 sequer).

A nível de código, só existe uma diferença: no primeiro caso, usamos a estrutura devolvida pelo *recvfrom()* para responder ao client1; no segundo caso, criamos uma estrutura (*sockaddr_in*) do zero (porque temos de ir buscar o IP do client2 à tabela). Talvez aqui resida a fonte do problema.

Reparámos também, ao usar o *netcat* para enviar pacotes Server → client2, que tudo funcionava na perfeição. Posto isso, analisámos o pacote (do *netcat*) à saída do Server e comparámos com o pacote que não era reencaminhado e eles eram iguais.

Concluímos então que o problema provavelmente residia nas configurações do Router 3. Então, tentámos fazer um *debug* dos pacotes que chegavam ao router para perceber o que o router fazia com eles. Ao fazer o *debugging*, foi-nos aconselhado num fórum qualquer que teríamos de desativar o *fast-switching*, através deste comando: `no ip route-cache`

Curiosamente, ao desativar o *fast-switching*, o router começou a encaminhar esse pacote que não era encaminhado anteriormente. Provavelmente deveu-se a conflitos de cache...

Funcionalidades do Software

Adotámos um protocolo desenvolvido por nós que permite fazer *parsing* facilmente aos pacotes de forma a separar a informação e ser *machine-readable*. Abaixo ilustramos com alguns exemplos.

Autenticação

1. o client envia ao server um pacote no seguinte formato:
   `mode=1&user=miguel&password=pass`
2. o server responde com as comunicações possíveis:
   `1 - Client-Server | 2 - P2P | 3 - Group`

Client-Server

1. o client envia ao server um pacote no seguinte formato:
   `mode=4&user=miguel&destuser=edgar&data=mensagem`
2. o server faz o *parsing*, pega no IP do *user* destino e envia-lhe o conteúdo do *data* pela porta 4000 (padronizada)

P2P

1. o client envia ao server um pacote no seguinte formato:
   `mode=2&user=miguel&destuser=edgar`
2. o server responde com o IP do *user* destino
3. o *client* estabelece ligação com o IP do *user* destino e a mensagem *raw* para esse IP na porta 4000 (padronizada)

Multicast

1. o *client* pergunta ao utilizador se quer dar *join* a um grupo ou criar um novo.
2. se for para criar grupo, envia o seguinte pacote:
   `mode=3&groupmode=1`
   2.1. neste caso, o servidor cria o grupo com um dado IP *multicast*
3. se for para dar *join* a um grupo, envia o seguinte pacote:
   `mode=3&groupmode=2`
   3.1. neste caso, o servidor responde com os IPs dos grupos *online*
   3.2. o cliente pergunta ao utilizador qual o IP a entrar e procede ao *ADD_MEMBERSHIP*
4. envia mensagens *raw* para o IP multicast, na porta 4000 (padronizada)

Notas Finais

Os clientes perguntam ao utilizador qual modo de comunicação deseja realizar. Após isso, entra num *loop* infinito em que pede mensagens para serem enviadas.

Nos clientes, uma *thread* à parte é criada para receber as mensagens e dar *print* conforme necessário, visto que a função de *fgets* dá *block*, impedindo que o *recvfrom* seja executado. Já do lado do servidor, uma *thread* à parte também é criada para receber ligações TCP para a CLI das configurações. Provavelmente o ideal até seria criar um processo em vez de uma *thread*, mas não fizemos uma grande pesquisa nesse tema e, apenas por simplicidade, escolhemos *threads*, visto que partilham a *stack memory* evitando trabalho desnecessário.

Conclusão

O software funciona como pretendido. Os clientes comunicam com o servidor, com outros clientes e com os IP *multicast* sem problemas. O servidor recebe as *requests* e age conforme sem problemas também.