Relatório de Projeto IRC 19/20

Nota inicial

Este projeto aborda uma forma muito direta à comunicação de diferentes interfaces por intermédio dois protocolos de transporte base. Iremos lidar com os protocolos **UDP** e **TCP**. Estes protocolos perfazem a generalidade dos transportes de dados entre diferentes interfaces de redes locais ou externas. No caso do *Transmission Control Protocol* é estabelecida uma conexão entre cliente e servidor, em que não há perdas de dados. Isto resulta numa latência ligeiramente superior à do protocolo concorrente. No TCP, enviado um pedido, o servidor ao responder, só termina a transmissão quando receber um ACK (acknoledge) proveniente do Cliente. No caso do *User Datagram Protocol* não é estabelecida uma conexão e contrariamente ao TCP, o UDP não garante que o pedido seja corretamente recebido, ou seja, existe uma possibilidade do dados serem perdidos na retransmissão. Isto torna este protocolo mais rápido do que o TCP. A escolha entre os dois cabe meramente ao objetivo do Software. Por exemplo, streaming pode ser feito com requisito ao protocolo UDP visto que a perda de um pacote ou outro não afeta o processo final.

Relativamente a este projeto, é necessário criar um servidor, proxy e cliente.

O **servidor** tem como objetivo gerir os dados recebidos e responder conforme o pedido. Este tem que ter a capacidade de responder tanto a pedidos UDP e TCP, concorrentemente. A operação principal é a transferência de um ficheiro para um cliente.

O **proxy** irá funcionar como um intermediário, recebendo o pedido do cliente e reencaminhado-o para o servidor. Posteriormente fica à escuta da resposta e assim que a recebe, transmite para o cliente em questão. Contudo, o proxy também deve fornecer a capacidade de gerir as conexões ativas no momento, escrever em ficheiro os últimos pedidos de transferência e, finalmente, introduzir uma simulação à perda de pacotes. Esta perda deve estar somente ligada ao **datagramas** (pedidos UDP).

Finalmente, o que diz respeito ao cliente, este pode aceder à lista de ficheiros disponíveis na base de dados do servidor, transferir um ficheiro que esteja presente no mesmo, verificar os ficheiros já transferidos e finalmente sair do serviço.

Servidor

O servidor tem de estar preparado para lidar com pedidos concorrentes, ou seja, não poderá haver somente uma interface de receção de pedidos. Nisto está implícito que o servidor deverá estar capaz de criar um socket por cada cliente que entra no serviço e esse trata somente do cliente em questão. Assim que o cliente sair do serviço, este socket é fechado. No entanto, esta abordagem é dispendiosa em termos de recursos. A abordagem utilizada neste projeto foca a criação de um socket e somente um, para cada tipo de protocolo. Ou seja, um servidor terá um socket UDP e um socket TCP. Contudo, estes sockets são geridos por vários processos, que trabalham de forma concorrente, dependendo dos pedidos que chegam ao servidor.

A estratégia então está focada num servidor Multi-Threaded, isto implica que a cada nova conexão, teremos uma thread a tratar do pedido e a utilizar o socket para comunicar com o cliente.

Relativamente à transferência de dados, quando este servidor recebe um pedido para transferência, vai transmitindo byte a byte para o socket no proxy, que está à escuta. Assim que terminar a transferência (Chegada ao byte final do ficheiro) volta a ficar à escuta do socket UDP ou TCP.

Proxy

A abordagem à receção e resposta de pedidos é tanto ou quanto semelhante à do servidor. No entanto, como o proxy funciona como um intermédio entre o cliente e o próprio servidor, tanto para garantir fidelidade na comunicação como para mediar a mesma.

Na programação do mesmo, caso um cliente mande um pedido desconformado por alguma razão, o proxy, com base nos dados recebidos do cliente, irá corrigi-los de forma a que o pedido prossiga de forma normal, sem erros.

É seguro dizer que a maior parte das operações no sistema ocorrem servidor, contudo, o proxy mantém o serviço normalizado.

Para que se mantenha a segurança no pedido, os dados podem ser encriptados, sem que este consiga visualizar o conteúdo. Somente os polos da comunicação (Cliente e Servidor) podem observar o resultado.

Tanto como no servidor, o proxy irá ter um socket socket para cada protocolo, bem como um socket referente às resposta provenientes do servidor e um para responder ao cliente. Isto constitui então uma interface de quatro sockets, sendo que cada um é gerido por múltiplos processos. Na realidade, tanto processos quantos clientes existam no sistema. Permitindo concorrência, sem desmobilizar as comunicações dos restantes clientes.

A transferência de ficheiros passa de forma transparente a esta interface, querendo com isto dizer que, o proxy retransmite apenas o que o servidor lhe mandou, em resposta ao pedido do cliente. Desta forma, o proxy tanto recebe como envia diretamente para o cliente, não havendo qualquer processo de leitura intermédia ou tratamento de dados. Da forma como a informação chega ao proxy, é da forma como o cliente irá receber.

Em referência às funcionalidade internas do proxy temos a exposição de todas as conexões ativas com o comando SHOW. Utilizando o comando **SHOW** o proxy irá mostrar na consola todas as conexões ativas naquela instante, com o formato "***CLIENT IP : CLIENT PORT || SERVER IP: SERVER PORT***".

Por conseguinte, existem outros dois comandos: SAVE e LOSSES.

Quanto ao comando **SAVE** funciona apenas como um switch, permitindo ou não a escrita em ficheiro, dos pedidos de transferência.

Finalmente, com o comando **LOSSES** iremos implementar um fator para efeitos de simulação. Como o software irá correr (possivelmente) numa rede local, não iremos ter capacidade de notar quaisquer perdas de pacotes, relativamente ao UDP. Por isto, é necessário introduzir um fator que simule esta perda. Então, procedemos à implementação do comando LOSSES seguido da percentagem de perda de pacotes. Esta percentagem é verificada sempre que o proxy quer responder a um cliente. Com recurso a um número aleatoriamente gerado é decidido se mandamos a resposta ou não.

Cliente

O cliente é talvez a interface mais simples. É provido da capacidade de transmitir um pedido em qualquer protocolo e receber o pedido. Contudo, um cliente que tenha iniciado a sessão com requisito ao protocolo UDP pode utilizar o protocolo TCP para transferir um ficheiro e vice-versa. Isto permite alguma heterogeneidade ao serviço do cliente. Poderia também haver um mecanismo que, consoante o tipo de ficheiro,

decidisse qual protocolo usar, de forma a minimizar o esforço do servidor e otimizar o tempo de resposta.

Um cliente deve poder pedir a lista de atuais ficheiros em memória (do servidor). Deve também poder transferir ficheiros. Esta transferência recorre a um protocolo em formato de string (char *) com o formato "**DOWNLOAD|TCPorUDP|ENCorNOT|FILENAME**". Neste, é imposto qual protocolo a utilizar, bem como se é necessária encriptação de dados ou não. Este protocolo é usado tanto por clientes UDP como por clientes TCP.

Nesta interface foi também implementada a funcionalidade extra que permite a um cliente ter a noção de quais ficheiros já transferiu. Isto processa-se de forma simples. Assim que o pedido é gerado, é feito internamente, sem requisito ao proxy ou ao servidor. O sistema percorre a diretoria Downloads do cliente e mostra-lhe os ficheiros presentes. Espelhando o processo do servidor para resposta de um pedido de listagem (**LIST**).

Finalmente, o cliente deve poder terminar a sua sessão, fechando não só os seus sockets, como também todos os processos criados pela sua atividade. Este processo é feito com requisito ao ID que percorre o proxy e servidor, dentro do pacote. Cada pedido vais juntamente com o ID do cliente, para que se saiba precisamente qual o cliente em questão. Então, assim que um cliente invoca o comando **QUIT**, o proxy e o servidor terminam os processos referentes a esse ID.

Da autoria de:

André Leite, 2015250489, uc2015250489@student.uc.pt