Projeto Semestral de Computação Distribuída

O projeto consiste na construção de uma rede distribuída de partilha de foto entre os vários participantes, garantindo a consistência da rede durante toda a sua execução. A metodologia usada para o desenvolvimento do projeto foi dividi-lo em várias etapas de modo a implementar todas as funcionalidades necessárias.

Inicialização da rede

Primeiramente, era necessário decidir a topologia da rede a usar. Ao analisar as várias hipóteses possíveis foi decido que a solução ideal era usar a topologia *full mesh*, onde todos os nós (*Daemons*) se conhecem entre si. Apesar da grande complexidade de ligações, faz sentido usar esta topologia, visto que, a premissa do trabalho é uma rede de amigos que vão estar a usar em simultâneo logo, o número de utilizadores nunca vai ser grande o suficiente para justificar uma topologia mais eficiente.

Para a implementação, cada *Daemon* da rede tem uma *socket* de receção e *n*-1 *sockets* de envio, onde cada uma se liga aos outros nós da rede. Para isso o nó inicial (super) inicia sem tentar efetuar nenhuma ligação, apenas iniciando as suas estruturas necessárias para o seu funcionamento. Os nós que se ligarem depois do nó principal vão criar a *socket* responsável por ligar cada um de deles ao nó super e vão efetuar um pedido para se ligarem à rede através da chamada *send_join*. Ao efetuar o join vai ser criada a socket para o novo nó da rede e vão ser efetuadas as comunicações com os outros nós da rede a avisar que há um novo membro, forçando-os a tentar estabelecer ligação com este, com a chamada *send_peer*.

No caso de um dos nós sair da rede, todos os outros têm de ser avisados que a ligação morreu e que é preciso removê-lo da lista de *peers* e fechar a ligação com este.

Envio de imagens pela rede

O principal objetivo do projeto é ter uma coleção de imagens descentralizadas onde cada um pode aceder às imagens guardadas pelos outros, ou seja, as imagens têm de conseguir ser enviadas pela rede. Para isso, as imagens são lidas para o dicionário *personal_collection* onde cada chave é o nome da imagem e o valor é um tuplo com a *hash* da imagem e a imagem em binário. Durante a adição das imagens, são também filtradas as imagens localmente duplicadas pela *hash*, onde é escolhida sempre a imagem com melhor resolução. O dicionário *img_map* vai ser atualizado com uma nova entrada, onde a chave é o nome da imagem e como valor uma lista que contém os *sockets* onde esta imagem existe. Para o envio pela rede, a imagem é enviada pela socket e como as *sockets* usam o protocolo TCP toda a fragmentação da imagem é tratada automaticamente.

Cliente

É possível interagir com o *Daemon* de duas maneiras: através de um terminal ou de uma interface gráfica Web. No terminal, o utilizador tem vários comandos que pode executar para ver os *peers* que estão ligados ao seu *daemon* (*peers)*, a lista de imagens da rede(*list)* e pedir o mapa total da rede, *img_map* (map) e pedir uma imagem à rede (*get image_name)*.

A interface gráfica Web, foi construída com recurso à biblioteca Flask e mostra uma tabela de todas as imagens da rede com hiperligações que ao clicar vai mostrar a imagem pretendida. Para isto, a imagem pretendida é guardada na pasta *static* com o nome tmp.jpg e é mostrada no browser.

Redundância

Caso aconteça algum erro na rede, as imagens têm de permanecer nesta. Para isto, todas as imagens da rede têm de ser armazenadas duas vezes em nós diferentes. Para isso, quando um nó entra na rede as imagens são distribuídas pela rede, tendo como método de equilíbrio o tamanho atual da *personal_collection*. Por falta de tempo, não foi possível implementar a remoção total de imagens duplicadas pela rede toda, mas seria implementável se o dicionário *img_map* passasse a ter como chave o *hash* da imagem e fazer a mesma filtragem que é feita para duplicar as imagens pela rede, removendo as réplicas a mais e com menor qualidade de resolução.

Protocolo

O Protocolo assegura a consistência das mensagens passadas pela rede. Todas as mensagens efetuam o *marshalling* com recurso ao Pickle e começam com o tamanho total da mensagem.

A seguir é apresentado todas as mensagens utilizadas

Nome	Função	Argumentos
send_peer	Enviar pedido de Peer	{'type': 'peer', "args": {"addr": addr}}
send_join	Enviar pedido de join	{'type': 'join', "args": {"addr": addr}}
send_node_list	Enviar pedido de lista de peers	{'type': 'list_nodes', "args": {}}
send_nodes	Enviar lista de peers para o client	{'type': 'nodes', "args": {"peers": peers}}
send_register_client	Registar cliente de um Daemon	{'type': 'register', "args": {"addr": addr}}
send_image	Enviar imagem	{'type': 'image', "args": {"image": image, "addr": addr}}
send_get_image	Enviar um pedido de imagem	{'type': 'get_image', "args": {"key": key, "addr": addr}}
send_get_map	Enviar pedido para obter o img_map	{'type': 'get_map', "args": {"addr": addr}}
send_map	Enviar o img_map	{'type': 'map', "args": {"map": _map, "addr": addr}}
send_get_image_location	Enviar pedido pela lista da localização das imagens da rede	{'type': 'get_image_location', "args": {}}
send_image_location	Enviar lista da localização das imagens da rede	{'type': 'image_location', "args": {"locations": locations}}
send_get_folder_size	Enviar pedido pelo tamanho de cada coleção pessoal	{'type': 'get_folder_size', "args": {"addr": addr}}
send_folder_size	Enviar tamanho da coleção pessoal	{'type': 'folder_size', "args": {"size": size,"addr": addr}}
send_get_duplication_image	Enviar pedido para duplicação da imagem	{'type': 'get_duplication_image', "args": {"addr": addr,"key": key}}

send_duplication_image	Enviar imagem para o novo nó para efetuar a replicação	{'type': 'duplication_image', "args": {"addr": addr,"hash": _hash,"image": image,"key": key}}
send_get_image_by_node	Manda um pedido para receber img_list com a lista de nodes onde uma dada imagem se encontra	{'type': 'get_image_by_node', "args": {}}
send_image_by_node	Envia o img_list com a lista de nodes onde uma dada imagem se encontra	{'type': 'image_by_node', "args": {"map":_map}}