Trabalho Prático 2 - Transmissão de arquivos usando UDP com controle sobre TCP

Júlia Fonseca de Sena - 2018054508

1) Implementação

O projeto foi implementado em Python 3.6.9 com as bibliotecas sys, os, errno, select e socket. O objetivo foi simular um sistema de armazenamento na nuvem, sendo mensagens de controle enviadas pelo TCP e o arquivo pelo cliente via UDP.

Primeiramente, será explicada a abordagem escolhida para a janela deslizante. Foi implementada a Go Back N. O tamanho escolhido foi 5, e a janela avança à medida que recebe os ACKs, ou seja, se na janela que envia os pacotes 0, 1, 2, 3 e 4 (sendo esses os índices dos próprios pacotes e não o número de sequência na janela) não for recebido o ACK do pacote 2, avança a janela para abranger os pacotes 2, 3, 4, 5 e 6. Será reenviado somente o pacote 2 e os pacotes 5 e 6 serão enviados pela primeira vez, mas 3 e 4 não serão transmitidos visto que já tiveram seu ACK recebido. Do lado do cliente, vigia-se quantas retransmissões de cada pacote: caso algum chegue a 5 retransmissões, o é desconectado do servidor.

Partindo para o servidor, o primeiro passo é a inicialização do socket TCP, com o endereço IPv6 "::" e um port recebido pela linha de comando. O socket é configurado de forma que receba conexões de ambos clientes IPv6 e IPv4. O select seleciona os sockets com atividade (entrada, saída e except) de forma a aceitar conexões e *receives*, enviar mensagens e detectar erros em algum socket.

Cada cliente é representado pela estrutura de dados UDP_client, que guarda seu socket TCP, seu socket e port UDP, o nome do arquivo que será salvo, uma lista com os pacotes recebidos em ordem e o índice do início atual da janela deslizante.

As mensagens recebidas via TCP são enviadas para o método id_tcp_msg, que identifica o tipificador da mensagem e então detecta se é uma mensagem HELLO ou INFO FILE. Se for HELLO, abre um port UDP que será usado só pelo cliente específico. O primeiro port que será utilizado será o port do servidor somado de 1, e cada vez que o port é usado, a variável global (utilizada na inicialização) é incrementada de 1. Dessa forma, nenhum port será reutilizado. Cria-se então a estrutura do cliente que enviou o HELLO, e coloca-o na lista de clientes (para ter acesso a todas as suas informações a partir de um socket ou port UDP disponível). Após esse procedimento, envia-se, ainda via TCP, a resposta do tipo CONNECTION, informando o cliente para que port UDP enviar seus pacotes.

A próxima mensagem que será recebida é a INFO FILE, que provê para o servidor o nome e o tamanho do arquivo a ser salvo. Após separar os bytes de nome dos de tamanho (os últimos 8 representam o tamanho), procura-se a estrutura de dados do cliente na lista comparando seu socket TCP. A preparação para o recebimento do arquivo (feita dentro da estrutura de dados) é a seguinte: cadastra-se o nome recebido e aumenta-se a lista de pacotes (self.file_content) até o número de pacotes (de tamanho máximo 1000 bytes) que serão recebidos do cliente. Esse número é encontrado pela divisão inteira do tamanho do arquivo em bytes por 1000, somando-se 1 caso o resto tenha sido diferente de 0. Na lista de pacotes é colocado None para sinalizar que cada um dos pacotes ainda não foi recebido. Após o alocamento desses espaços, envia-se para o cliente a mensagem OK, sem cabeçalhos extras.

Depois, parte-se para o recebimento de pacotes de dados do arquivo via UDP. Com um select de timeout de 0.2 segundos, o servidor recebe 5 pacotes (o tamanho da janela). Depois de receber os 5 ou a espera de algum recv ultrapassar o timeout, processa-se cada mensagem, colocando o pacote na lista do arquivo do cliente (identificado pelo socket UDP em que a mensagem foi recebida) no índice correspondente ao começo da janela atual + número de sequência do pacote.

Checa-se, também, se o tamanho do pacote recebido é igual ao declarado no cabeçalho da mensagem (caso contrário, retorna -1 e o ACK não é enviado ao cliente).

Envia-se, então, o ACK com o número de sequência da mensagem, por TCP. O ACK é enviado logo após o processamento ao invés de esperar o próximo select como outras mensagens (as quais eram colocadas em uma fila de mensagens e seu socket na lista outputs), a fim de não prejudicar o funcionamento do timeout no cliente. Se ainda houver pacotes para receber (None na lista de conteúdo do arquivo), atualiza-se o começo da janela para o primeiro pacote que não foi recebido. Caso o arquivo inteiro já tenha sido recebido, salva-se o arquivo em uma pasta “output”, com o mesmo nome do arquivo original, envia-se FIM para o cliente fechar sua conexão com o servidor, e fecha-se o socket UDP do cliente. Não foi criado comando para finalização do servidor, basta dar Ctrl+C no prompt em que seu programa está aberto.

No cliente, a primeira checagem (depois do número de parâmetros recebidos na linha de comando) é se o nome de arquivo recebido é válido: todos os caracteres devem ser ASCII, só pode haver precisamente um ponto, e, após o ponto (o formato do arquivo), deve haver exatamente três caracteres. Tenta-se, então, abrir o arquivo. Se não conseguir, imprime que houve um erro. Lê-se o conteúdo do arquivo e mede-se o tamanho com len(file_content). Identifica-se o tipo de endereço recebido (IPv4 ou IPv6) e conecta-se via TCP com o servidor. Envia-se então o HELLO e espera-se o CONNECTION com o port UDP a ser utilizado. A próxima mensagem enviada é INFO FILE, com nome e tamanho do arquivo. Após receber o OK, conecta-se com o port UDP recebido no CONNECTION e prepara-se para enviar o arquivo.

O primeiro passo é dividir todo o arquivo em pacotes de 1000 bytes e um último pacote com o resto da string resultante da leitura do arquivo. Então, duas listas de controle são criadas, uma booleana que é inicialmente toda False (índices viram True se o ACK do pacote correspondente for recebido) e uma que possui a contagem de retransmissões de cada pacote, inicialmente 0 para todos. Ambas possuem o tamanho igual ao número de pacotes a ser enviado.

Em seguida, é iniciado um loop que se repete enquanto houver algum False na lista de booleanos. Primeiramente, seleciona-se a janela atual (e o *slice* da lista de booleanos correspondente) e envia seus pacotes. Antes de enviar de fato, verifica-se se um ACK já foi recebido para aquele pacote. Se sim, não há porquê transmiti-lo novamente, e passa para o próximo pacote da janela. Entra-se então em um loop que termina quando não há mais False no corte da lista de booleanos referente à janela, ou quando o timeout do select (1 segundo) é atingido. O único elemento nas listas do select é o socket TCP do cliente, então só há atividade quando uma mensagem é recebida por ele. Quando um ACK é recebido, identifica-se seu número de sequência e atualiza-se seu booleano para True. Se não houve atividade no socket em 1 segundo, um timeout foi detectado e o loop do select é quebrado. O mesmo ocorre se todos os pacotes da janela receberem um ACK.

Checa-se então se o número de retransmissões de um pacote foi ultrapassado. Para os pacotes que estão na janela, se não receberam ACK, adiciona-se um a seu número de retransmissões. Caso ultrapasse 5, o cliente é desconectado do servidor.

Após o recebimento da confirmação para todos os pacotes, o procedimento que falta é o recebimento da mensagem FIM, que indica que o servidor terminou de receber e salvar o arquivo e o cliente pode agora fechar sua conexão. Ao receber, é fechado o socket TCP e UDP, e o programa do cliente chega ao fim. O servidor continua esperando por outras conexões, mas percebe que o socket foi desconectado e o tira das listas vigiadas pelo select.

2) Desafios, dificuldades e imprevistos

Apesar da dificuldade inicial de adaptar o que já tinha sido aprendido de Programação em Redes em C para Python, o resto do trabalho se desenvolveu mais facilmente que no trabalho prático anterior. Talvez pela maior intimidade que tenho com a linguagem ou pelo fato de ela ser mais simples e intuitiva tenha contribuído.

Dessa vez, não houveram imprevistos que prejudicaram o andamento do projeto. A implementação ocorreu mais fluidamente.