Transmissão de arquivos usando UDP com controle sobre TCP

Redes de Computadores

Gabriel Victor Carvalho Rocha

gabrielcarvalho@dcc.ufmg.br - 2018054907

1. Introdução

O projeto se resume à construção de uma conexão cliente/servidor, com objetivo do cliente enviar um arquivo por meio do protocolo UDP através de uma janela deslizante, sendo a escolhida neste trabalho a *Go-Back-N*. As primeiras mensagens TCP estabelecem a conexão e enviam informações que são necessárias para o servidor, após isso são transferidos os dados do arquivo através do UDP. Esse envio é controlado pelas mensagens de confirmações transmitidas por meio do protocolo TCP, que caminha pela janela e prossegue nos envios até que se termine todos os bytes do arquivo.

2. Implementação

No primeiro passo, o servidor cria duas threads, uma para cada socket (IPv4 e IPv6) que após utilizar o método *listen*, ficará ambas em um *loop* aguardando que algum cliente se conecte.

Quando um cliente inicia a conexão, ele possui alguns valores iniciais padrões, como o tamanho da janela sendo 4, logo em seguida é verificado se o endereço IP fornecido é IPv4 ou IPv6 para gerar seu socket corretamente e então conecta-se ao servidor. As informações do arquivo são verificadas pela função *check_file_name* chamada pela *get_info_file*, interrompendo a execução do programa caso seja um arquivo inválido. Do contrário, é iniciado o estabelecimento da conexão, enviando a mensagem *HELLO*, que quando recebida pelo servidor, irá gerar uma porta UDP (que o cliente usará para transmitir o arquivo), enviada por meio da mensagem *CONNECTION*.

Agora, o cliente envia as informações do arquivo que deseja fazer *upload* através da mensagem *INFO_FILE*, sendo recebidas e armazenadas no servidor, que são confirmadas ao cliente como recebidas por meio da mensagem *OK,* permitindo-o iniciar o *upload.*

Antes de enviar, é criada uma lista contendo os frames do arquivo, ou seja, cada frame nessa lista será um *dict* contendo a sequência do frame, os *bytes* do arquivo e o tamanho desse frame. O tamanho máximo de cada frame é 1000 *bytes*, respeitando o cabeçalho da mensagem. Também é feita uma verificação do tamanho da janela, caso o tamanho do arquivo seja pequeno o

suficiente para não precisar das 4 janelas (ou seja, menor do que 4000 *bytes*) o novo valor da janela será o número de frames que existem.

Com a execução da função *go_back_n*, temos a implementação da janela deslizante *Go-Back-N*, iniciando o envio dos frames. Aqui temos um *loop* que se mantém até que a mensagem de *FIM* seja recebida, a janela é controlada pelas duas variáveis que servem como apontadores: sendo *next_seq_num* o próximo frame disponível para enviar e *send_base* o primeiro frame da janela que foi enviado, mas não confirmado. Logo no início, é verificado se o próximo a enviar está dentro da janela e se há mais frames disponíveis, enviando então esse frame para o servidor e incrementando *next_seq_num*. Após isso é verificado também se foi recebida alguma confirmação de alguma sequência, que se retornado a *flag* de *FIM*, significa que o envio terminou. Senão, caso a sequência que foi recebida for igual ao valor que se espera, irá incrementar *send_base*, movendo a janela. Entretanto, se a sequência recebida for diferente do *send_base*, significa que houve um atraso no frame que deveria ter chegado, iniciando um *timer* que só se reinicia se a sequência esperada chegue, caso contrário irá acionar o *timeout* que reenviará todos os frames a partir do *send_base*.

Já no servidor, há um *loop* que ficará recebendo os frames pelo socket até que o tamanho do arquivo seja alcançado. Os *bytes* recebidos do arquivo são armazenados em um *dict* que irá mapear esses *bytes* pelo valor da sua sequência, e após receber uma sequência que está dentro da janela definida com tamanho 4, envia ao cliente a confirmação *ack* da respectiva sequência, e se a sequência for igual a esperada (*next_seq_num*) irá incrementar este apontador, movendo a janela. Caso haja alguma perda de pacote ou receba uma sequência diferente das esperadas, o servidor retorna a última sequência confirmada (ou retorna -1, no caso do frame 0) a fim de possibilitar que o cliente perceba e gere um *timeout* para reenviar os frames. Quando o arquivo enviado for totalmente recebido, existe um segundo *loop* que escreverá todos os bytes na ordem da sequência em um novo arquivo que será salvo na pasta *uploads/*.

Diferente do *Go-Back-N* tradicional que reenvia e salva todos os frames novamente, nessa implementação o *dict* irá adicionar uma nova sequência apenas se ela não estiver salva, ou seja, quando se envia os frames *n* e *n+1* e houver apenas a confirmação do *n+1*, *n* e *n+1* são reenviados, mas não será necessário salvar novamente *n+1*, precisando salvar apenas o frame *n*. Isso pode ser visualizado quando há perda na última janela, que após reenviar os frames, o servidor pode finalizar a conexão antes que o cliente termine o envio, pois o arquivo já foi totalmente recebido e salvo.

Por último, é enviada a mensagem *FIM* para o cliente, informando que o *upload* foi concluído, encerrando assim a execução do mesmo.

3. Conclusão

Dessa forma, o trabalho se mostrou muito mais complexo do que o anterior, gerando muitas dúvidas na questão da implementação da janela do lado do cliente e principalmente do servidor, como também o entendimento das variações, devido a falta de base oferecida nas aulas.

No fim, foi possível implementar uma variação da janela deslizante *Go-Back-N* com tamanho 4 de ambos lados, funcionando como esperado em casos de perdas de frames, realizando o *upload* do arquivo passado como argumento.