Trabalho Prático 2 - Redes de Computadores

Luiz A. D. Berto

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

luizberto@dcc.ufmg.br

Abstract. *This work aims to implement a reliable file transfer over UDP, with control over TCP. The main goal is to allow files to be transfered over unreliable networks. The Go-Back-N protocol was used with N=10. We discuss the various difficulties encountered with the implementation, and details of the Go-Back-N implementation.*

Resumo. *Este trabalho visa implementar uma transferência de arquivos confiável sobre UDP, com controle sobre TCP. O objetivo principal é permitir que os arquivos sejam transferidos por redes não confiáveis. O protocolo Go-Back-N foi usado com N=10. Discutimos as várias dificuldades encontradas com a implementação e os detalhes da implementação do Go-Back-N.*

1. Introdução

A transferência de arquivos em redes não confiáveis é um assunto de muita relevância na atualidade. Hoje, com a grande quantidade de dados que é gerada a cada segundo, aliada à grande popularidade de redes que são pouco confiáveis como Wi-Fi faz com que a necessidade do uso dessas redes para transmissões confiáveis e otimizadas seja de suma importância. Nesse trabalho, foi implementado o algoritmo *Go-Back-N*, com N=10, para a janela deslizante. Com ele, transferências em redes com altas taxas de perda de pacote foram possíveis, com performance que seria inalcançável pelo algoritmo *Stop-And-Wait*.

2. Implementação

Os programas foram implementados na linguagem de programação *Rust*. Essa escolha foi feita, dentre outros motivos, pelos mecanismos da linguagem que evitam *leaks* de memória e *data-races*, que são fatores bastante relevantes nesse contexto, uma vez que os programas lidam com múltiplas *threads*, e potencialmente podem lidar com arquivos muito grandes.

2.1. Common

Na *crate common*, foram implementadas algumas funcionalidades em comum ao cliente e ao servidor, como:

- O *enum Message*, que codifica os sete tipos de mensagens definidos na especificação;
- Funções para envio e recebimento de instâncias do *enum Message*;
- Funções para converter números em array de bytes;
- Tipos de dados abstratos que encapsulam algumas operações usadas em ambos os programas;
- Erros customizados, que permitem a generalização do tratamento de erros dos programas.

2.2. Client

Na *crate client*, foi feita a implementação de toda a lógica do cliente. O cliente é um programa que recebe como argumentos o ip e a porta do servidor, além do nome do arquivo a ser transferido. Após fazer algumas validações dos dados de entrada, o cliente inicia a troca de mensagens com o servidor, conforme detalhado na especificação. Após receber a mensagem do tipo *Ok*, o cliente inicia o processo de transmissão do arquivo, usando o algoritmo *Go-Back-N*, com N=10. Para tal, o cliente instancia uma nova *thread* ("*sender*") para lidar com o envio dos *chunks* de 1000 bytes via udp, e na *thread* principal ("*receiver*"), o cliente fica aguardando os *acks* do servidor.

Em alto nível, a implementação do envio acontece da seguinte maneira: o cliente sempre mantém a invariante de que no máximo 10 *chunks* podem ser enviados ao mesmo tempo sem receber *acks*. Com isso, o cliente é capaz de fazer um uso mais otimizado da rede, fazendo um uso melhor da capacidade de banda. Enquanto a thread *sender* envia os *chunks*, ela também verifica pela chegada de *acks* (que nesse algoritmo são *acks* cumulativos), que são recebidos pela thread *receiver*. Sempre que um *ack* chega, a variável de estado *send_base* é atualizada, deslocando a janela deslizante para frente, e permitindo o envio de novos *chunks*. O cliente também verifica se não houve *timeout*, ou seja, se passou um determinado período de tempo (na implementação, 200 milissegundos) que o pacote foi enviado e seu *ack* não foi recebido. Caso isso aconteça, o cliente reenvia toda a janela.

2.3. Server

Na *crate server*, foi feita a implementação da lógica do servidor. O servidor recebe como argumentos a porta TCP, e passa a escutar por conexões na porta especificada (tanto em IPv4 quanto em IPv6, fazendo uso de dual stack, sempre fazendo bind em *::*). Sempre que uma nova conexão é iniciada, o servidor instancia uma nova *thread* para atendê-la, possibilitando que múltiplos clientes se conectem simultaneamente. Após fazer a troca de mensagens inicial com o cliente, o servidor começa a receber os datagramas do arquivo, também empregando uma janela deslizante de tamanho 10. Para tal, o servidor mantém duas variáveis de estado para controlar a janela: *last_chunk_read*, que controla qual foi o último *chunk* lido que já foi reconhecido (limite inferior da janela deslizante), e *last_acceptable_chunk*, que controla qual é o último chunk aceitável (limite superior da janela deslizante). Com isso, a cada datagrama recebido, o servidor verifica se seu número de sequência está dentro dos limites da janela, e toma a ação adequada:

- Se o servidor verificar que todos os *chunks* foram recebidos, ele envia um *ack* cumulativo para o último *chunk*, efetivamente indicando o final da transmissão;
- Caso o número de sequência do datagrama seja o próximo *chunk* que o servidor ainda não mandou *ack*, então o servidor envia um *ack* cumulativo, verificando quantos *chunks* foram recebidos de forma contígua antes do *chunk* atual (pois UDP não garante a ordem de recebimento). Por exemplo, supondo uma janela de tamanho 3, se a ordem de recebimento dos três primeiros *chunks* de uma transmissão for 1, 2 e 0, ao receber o *chunk* com número de sequência 0, o servidor enviará um *ack* cumulativo para o *chunk* 2, indicando para o cliente que todos os *chunks* até o 2 (inclusive) foram recebidos com sucesso;

- Caso o número de sequência do datagrama não seja o próximo *chunk* que o servidor ainda não mandou *ack*, ele simplesmente vai ser armazenado na posição correta, e nenhum *ack* será enviado;
- Caso o servidor receba um *chunk* que está fora da janela, será enviado um *ack* para o último pacote já reconhecido. Essa ação é necessária pois o *ack* pode ter sido perdido pelo cliente, que com isso fica fora de sincronia com o servidor.

Ao final do recebimento dos datagramas e envio do último *ack*, o servidor salva todo o conteúdo recebido num arquivo na pasta *output*, envia a mensagem de fim de transmissão para o cliente, e encerra a conexão.

2.4. Dificuldades

Algumas dificuldades foram encontradas durante a implementação do trabalho, sendo elas:

- A simulação de falhas de rede, que fez com que a verificação da corretude da implementação da janela deslizante fosse bastante desafiadora
- Problemas de inanição da conexão TCP, que muitas das vezes não consegue utilizar a interface de rede devido ao alto tráfego de datagramas UDP (problema conhecido como *TCP Stavation / UDP dominance*). Para resolver parcialmente o problema, foi adicionado um *sleep* na thread UDP no cliente, visando liberar o canal para que a thread TCP consiga fazer a leitura de *acks*.
- Situações onde a taxa de perda de pacotes é muito alta causam alguns problemas como a falha para estabelecimento de conexões, ou perda dos *acks* finais, uma vez que o servidor finaliza a conexão após enviar o último *ack*. Nesse caso, o cliente usa uma estratégia otimista, assumindo que se a conexão tenha sido fechada quando faltam poucos *acks*, o servidor já conseguiu receber o arquivo por completo. Além disso, para taxas de perda superiores à um terço, é observada uma perda de performance altíssima, fazendo com que a transferência do arquivo passe a ser impraticável.

3. Execução

Para compilar o programa, basta rodar o comando *cargo build –release*. Esse comando gerará os executáveis *cliente* e *servidor* na pasta *target/release*. Os programas podem ser executados normalmente, recebendo os argumentos denotados na especificação.

4. Conclusão

Conclui-se que os algoritmos de janela deslizante são extremamente interessantes para transmissão eficiente de dados em redes não confiáveis, tendo performance muito superior ao *Stop-And-Wait*.

References

(2021). Linux man pages. `https://linux.die.net/man/`. Acessado em: 2021-02-15.