Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação
Redes de Computadores 2017/1
Professor: Dorgival Olavo Guedes Neto

TRABALHO PRÁTICO 1 – DCCNET: Camada de Enlace

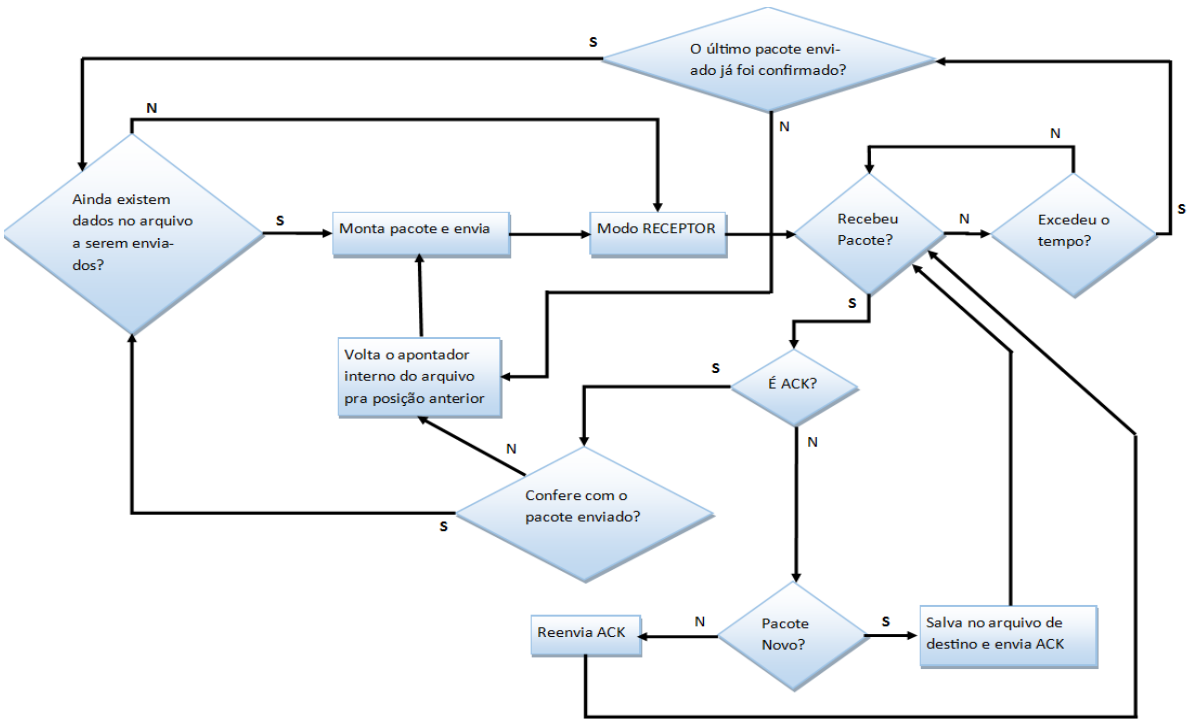
Aluno: Alison de Oliveira Souza – 2012049316
Aluno: Lucas Pedro da Silva Machado - 2012049740

Neste trabalho, implementamos um emulador da camada de enlace para uma rede fictícia chamada DCCNET, tratando do sequenciamento, enquadramento, detecção de erros e retransmissão de dados. A nossa camada de enlace funciona enviando quadros de tamanho máximo pré-determinado, no formato indicado abaixo. Os quadros são enviados usando o algoritmo *stop-and-wait* de controle de fluxo, ou seja, cada pacote é enviado apenas após o pacote anterior ter sido enviado e confirmado.

SYNC (0-31)	SYNC (32-63)	chksum (64-79)	length (80-95)	ID (96-103)	FLAGS (104-111)	DADOS (112-)

Implementação

O programa apresentado é *full-duplex*, ou seja, recebe e transmite dados simultaneamente. Para tal, optamos por implementar a seguinte máquina de estados:



Optamos por não salvar uma cópia do pacote enviado. Caso o ACK não seja recebido, confirmando o envio, o programa decrementa o ponteiro interno do arquivo de origem através da função *fseek()* e procede normalmente, como se estivesse enviando um novo pacote.

Para identificar erros de transmissão, usamos o algoritmo de checksum encontrado no endereço: https://codereview.stackexchange.com/questions/154007/16-bit-checksum-function-in-c.

Desafios

A primeira dificuldade foi a abstração da máquina de estados. Como o programa recebe e envia dados simultaneamente, foi meio difícil definir a ordem das coisas.

Com a ordem já definida, tivemos que lidar com mais alguns imprevistos. Na especificação mais atual do trabalho, os campos de identificação e de flags tinham apenas 8 bits, o que era incompatível com a função htons, que retorna uint16_t. Esse problema foi particularmente desafiador, uma vez que passava facilmente despercebido.

Além disso, como o fluxo de execução era definido por dados transmitidos pela rede e convertidos de um formato para outro, o programa facilmente entrava em loop ou não entrava em seções que deveria. Problema que costuma ser chato de se deputar.

Por fim, como tivemos problemas com a função htons, modificamos o código e erramos na hora de identificar qual era o pacote ao qual o segundo *ACK* estava se referindo. Por um breve momento, o nosso programa só funcionava para arquivos que coubessem em um só pacote.

Instruções de uso

Para iniciar o programa no modo servidor, use o comando:

```
./dcc023c2 -s <PORT> <INPUT> <OUTPUT>
```

Para o modo cliente, use o seguinte comando:

```
./dcc023c2 -c <IPPAS>:<PORT> <INPUT> <OUTPUT>
```

- Os parâmetros <INPUT> e <OUTPUT> são os nomes do arquivo que o programa deve enviar e onde ele deve armazenar os dados vindos da conexão.
- O parâmetro <PORT> indica em qual porta o servidor deverá esperar por uma conexão e o cliente deverá se conectar. Deve ser um número entre 51000 e 55000.
- O parâmetro <IPPAS> deve conter o endereço IP do servidor.

Considerações finais

Apesar de C não ser considerada uma linguagem de tão alto nível, ainda é relativamente desafiador trabalhar com bits e bytes. A depuração desse tipo de problema costuma ser meio complicada.

Felizmente, como já havíamos feito o primeiro trabalho, já estávamos familiarizados com o conceito de conexão entre processos, através de socketes.

Referências:

- TUTORIAL BASIC SOCKET IN C
http://www.bogotobogo.com/cplusplus/sockets_server_client.php

- CHECKSUM DA INTERNET
https://codereview.stackexchange.com/questions/154007/16-bit-checksum-function-in-c

- Programação em sockets em C para linux
http://www-usr.inf.ufsm.br/~giovani/sockets/sockets.txt