DCCNET: Camada de Enlace

Última modificação: 2 de abril de 2018

1 Introdução

Neste trabalho iremos desenvolver um emulador de camada de enlace para uma rede fictícia chamada DCCNET. O emulador tratará da codificação, enquadramento, detecção de erro, sequenciamento e retransmissão dos dados.

2 Codificação

Na DCCNET, a codificação é sempre a última tarefa a ser realizada antes de transmitir dados e sempre a primeira tarefa a ser realizada após receber dados. A DCCNET transmite dados usando codificação Base16.[1] A codificação Base16 codifica dados binários nos caracteres ASCII `0-9` e `a-f`. Cada quatro bits (um *nibble*) a serem codificados são convertidos em oito bits (um *byte*) do caractere ASCII correspondente ao nibble. A codificação Base16 tem eficiência de 50%. Seu código deve implementar funções chamadas `encode16()` e `decode16()` para realizar as operações de codificação e decodificação de dados. *Dica:* A função `encode16()` deve ser a última função chamada antes de transmitir dados na rede e a função `decode16()` deve ser a primeira função chamada antes de receber dados da rede.

2.1 Exemplo

`DIAGRAMA 1: Sequência de 4 bytes em diferentes codificações`

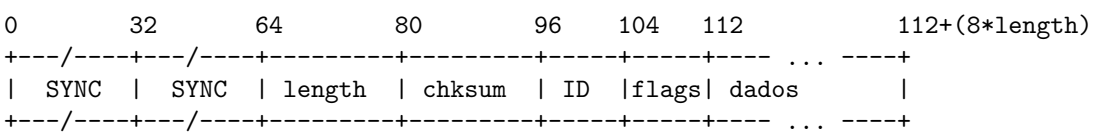
Bits	11011100	11000000	00100011	11000010
Decimal	220	192	35	194
Hexadecimal	0xDC	0xC0	0x23	0xC2
Base16	DC	C0	23	C2

3 Enquadramento

O formato de um quadro DCCNET é mostrado no diagrama 2. Todo quadro começa com uma sequência pré-determinada de sincronização [`SYNC`, 32 bits] seguida por uma repetição da sequência de sincronização. A sequência de sincronização entre os pontos finais é a mostrada no diagrama 1. O campo de tamanho [`length`, 16 bits] conta a quantidade de bytes de dados [`dados`, (`length` $\times$ 8) bits] transmitidos no quadro. O campo de tamanho *não* conta bytes do cabeçalho do quadro. O campo de tamanho deve ser transmitido sempre em *network byte order*.

[1] `https://en.wikipedia.org/wiki/Hexadecimal#Transfer_encoding`

DIAGRAMA 2: Formato de um quadro DCCNET



4 Detecção de Erros

Erros de transmissão são detectados usando o *checksum* presente no cabeçalho do pacote [`chksum`, 16 bits]. O *checksum* é o mesmo utilizado na Internet[2] e é calculado sobre todo o quadro, incluindo os 112 bits do cabeçalho e os dados. Durante o cálculo do *checksum*, os bits do cabeçalho reservados ao *checksum* devem ser considerados com o valor zero. Pacotes com *checksum* inválido não devem ser aceitos pelo destino, sendo simplesmente ignorados.

Para recuperar o enquadramento após a detecção de um erro, seu receptor deve esperar por duas ocorrências do padrão de sincronização (`SYNC`) que marcam o início de um quadro. O receptor deve então tentar receber o quadro, verificando o *checksum* ao final.

Note que o receptor pode ficar sem saber quando os quadros começam ou terminam, por exemplo, devido a erro de transmissão no campo `length` de um quadro bem como inserção de bytes corrompidos (lixo) no canal. Erros de transmissão podem ocorrer durante a transmissão de um quadro ou entre a transmissão de dois quadros. Por esse motivo, seu receptor deve permanecer procurando pelos padrões de sincronização (`SYNC`) para recuperar o enquadramento depois de um erro de transmissão.

5 Sequenciamento

A DCCNET usa o algoritmo *stop-and-wait* de controle de fluxo, isto é, as janelas de transmissão e recepção armazenam apenas um quadro. Todo quadro transmitido deve ser confirmado pelo receptor antes antes da transmissão do próximo quadro. O campo de identificação (`ID`, 8 bits) identifica o quadro transmitido ou confirmado. Os valores válidos do campo de identificação se restringem a zero ou um.

5.1 Transmissão

Para transmitir dados, o nó transmissor cria um quadro, como especificado no diagrama 2. Quadros de transmissão de dados possuem pelo menos 1 byte de dados e valor maior do que zero no campo de tamanho [`length`]. Após recebimento da confirmação de recepção do quadro, o transmissor deve trocar o valor do campo de identificação [`ID`] (de zero para um ou de um para zero) e transmitir o próximo quadro.

5.2 Recepção

Após o recebimento de duas ocorrências do padrão de sincronização, o receptor deve iniciar o recebimento de um quadro. Para isso deve esperar uma quantidade de bytes correspondente ao valor do campo de tamanho [`length`]. Ocorrências de padrões de sincronização durante o recebimento destes bytes devem ser ignoradas.

[2] https://tools.ietf.org/html/rfc1071

O receptor deve manter em memória o identificador (`ID`) do último quadro de dados recebido. Um novo quadro de dados só pode ser aceito se ele tiver identificador [`ID`] diferente do identificador do último quadro recebido[3] e se nenhum erro for detectado (seção 4). Ao aceitar um quadro de dados, o receptor deve enviar um quadro de confirmação com o campo de identificação [`ID`] idêntico ao do quadro confirmado.

Quadros de confirmação não carregam dados (têm tamanho [`length`] igual a zero) e possuem o bit de controle de confirmação [`ACK`] ligado. O diagrama 3 mostra os bits de controle utilizados na DCCNET.

```
DIAGRAMA 3: Bits de controle do campo flags

Máscara (hex)    | Bit | Nome | Função
1000 0000 (0x80) |  7  | ACK  | Confirmação de recebimento de dados.
0111 1111 (0x7f) | 6-0 | ---  | Reservado. Devem ser sempre zero.
```

Os enlaces DCCNET são *full-duplex*: dados podem ser transmitidos nas duas direções, e cada nó funciona como transmissor e receptor simultaneamente. Em particular, em qualquer instante, um nó DCCNET deve estar apto a receber e processar um quadro de dados ou um quadro de confirmação.

6 Retransmissão

Como transmissões podem sofrer erros, a DCCNET utiliza confirmação e retransmissão de quadros para garantir entrega. Em particular, erros podem acontecer na transmissão de quadros de dados e na transmissão de quadros de confirmação.

Enquanto um quadro de dados transmitido não for confirmado, o transmissor deve retransmiti-lo (periodicamente) a cada segundo. Se o receptor receber a retransmissão do último quadro de dados recebido, ele deve reenviar a confirmação. O receptor deve armazenar o identificador e *checksum* do último quadro recebido para detectar retransmissões.

7 Implementação e interface com usuário

Você deverá implementar o DCCNET sobre uma conexão TCP.[4] Seu emulador do DCCNET deve ser capaz de funcionar como transmissor e receptor simultaneamente. Seu emulador deve interoperar com outros emuladores (teste com emuladores dos colegas), inclusive com o emulador de referência implementado pelo professor. O trabalho pode ser implementado em Python, C, C++, Java, ou Rust, mas deve interoperar com emuladores escritos em outras linguagens.

7.1 Inicialização

O emulador na ponta passiva (servidor) do enlace DCCNET deve ser executado como segue:

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

[3] Inicialize o identificador do último quadro com o valor um, de forma que o primeiro quadro transmitido tenha o identificador zero.

[4] Utilize `socket(AF_INET, SOCK_STREAM, 0)`, ou equivalente, para criar o soquete.

Onde `PORT` indica em qual porta o emulador irá escutar por uma conexão. O emulador deve esperar a conexão no IP da máquina em que estiver executando. `INPUT` é o nome de um arquivo com os dados que devem ser enviados à outra ponta do enlace, e `OUTPUT` é o nome de um arquivo onde os dados recebidos da outra ponta do enlace devem ser armazenados.

O emulador na ponta ativa (cliente) do enlace DCC023C2 deve ser executado como segue:

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

Onde os parâmetros `PORT`, `INPUT` e `OUTPUT` têm o mesmo significado acima. O parâmetro `IP` é o endereço IP da máquina onde o emulador na ponta passiva está executando.[5] Note que no final o arquivo de saída do cliente (servidor) deve ter o mesmo conteúdo do arquivo de entrada do servidor (cliente).

7.2 Execução

Seu emulador deve ler os dados a serem transmitidos (possívelmente dentro de vários quadros) do arquivo `INPUT` especificado na linha de comando. Seu emulador deve escrever em no arquivo `OUTPUT` todos os dados recebidos da outra ponta do enlace DCCNET. Os dados devem ser escritos depois decodificados. (Os cabeçalhos não devem ser escritos no arquivo `OUTPUT`.)

7.3 Terminação

O emulador pode parar de executar sempre que a conexão for fechada pelo nó remoto (independente do emulador ter sido iniciado no modo cliente ou no modo servidor). O emulador também pode parar de executar quando receber um sinal de interrupção de teclado (`ctrl-c`), isto é, quando for terminado manualmente. O emulador *não* deve parar de executar quando terminar de transmitir o arquivo.

8 Avaliação

Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor; se confirmada a incoerência ou ambiguidade, o aluno que a apontou receberá um ponto extra.

8.1 Grupos

Este trabalho pode ser executado em grupos de até dois alunos (todos os alunos são responsáveis por dominar todos os aspectos do trabalho).

8.2 Entrega

O programa deve ser entregue como apenas um arquivo fonte na linguagem escolhida (`dcc023c3.c`, `dcc023c3.cpp`, `dcc023c3.py`, `dcc023c3.java`, `dcc023c3.rs`). Além disso, deve ser entregue também um arquivo PDF de documentação, descrito a seguir.

[5] Note que esse endereço pode até ser 127.0.0.1 se os programas estiverem executando na mesma máquina, mas pode também ser o endereço de uma outra máquina acessível pela rede. Para determinar o endereço da máquina onde o lado passivo vai executar você pode usar o comando `ifconfig`, ou fazer o seu programa escrever o endereço IP da máquina onde ele está executando [em Python, pode-se usar `socket.gethostbyname(socket.getfqdn())`].

8.3 Documentação

Cada grupo deverá entregar documentação em PDF de *até* 4 páginas (duas folhas), sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos de projeto, bem como as soluções adotadas para os problemas.

8.4 Testes

Pelo menos os testes abaixo *serão* realizados durante a avaliação do seu emulador:

- Transmissão e recepção de dados em paralelo.
- Recuperação do enquadramento após erros de transmissão. (Note que você deve usar uma versão alterada do programa para "criar" erros, já que a transmissão usará TCP.)

A versão do programa a ser entregue deve funcionar corretamente mesmo se houver erro em qualquer um dos quadros recebidos. Como os programas executarão sobre o protocolo TCP, não haverá erros na transmissão: os bytes recebidos com `recv()` corresponderão exatamente aos bytes enviados com `send()`. Entretanto, nos testes iremos usar programas que geram e enviam bytes errados intencionalmente, com o objetivo explícito de verificar se seu programa detecta erros de transmissão e consegue ressincronizar e continuar o recebimento de dados após um erro.

Para testar seu programa nesses casos você pode criar versões alteradas do programa que enviam intencionalmente quadros errados, para ver como o programa do outro lado se comporta. Não é necessário entregar esses programas usados para teste. O professor criará um programa com geração de diferentes tipos de erros para fins de teste. A princípio, esse programa não será publicado.

9 Exemplo

Para transmitir quatro bytes com valores 1, 2, 3 e 4 (nesta ordem), os seguintes bytes terão de ser transmitidos na camada de enlace (assumindo que o identificador (`ID`) do quadro é zero):

```
dcc023c2dcc023c20004faef000001020304
‘SYNC   ‘SYNC   |   |   | |   ‘dados
                |   |   | ‘flags
                |   |   ‘ID
                |   ‘chksum
                ‘length
```