DCCNET: Camada de Enlace

Introdução

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

Enquadramento

O formato de um quadro DCCNET é mostrado no diagrama 1. A sequência utilizada para sincronização entre os pontos finais (`SYNC`, 32 bits) é sempre `0xDCC023C2`. O campo de detecção de erro (`chksum`, 16 bits) utiliza o algoritmo de *checksum* da Internet para detecção de erros. O campo de tamanho (`length`, 16 bits) conta a quantidade de bytes de *dados* transmitidos no quadro; o campo de tamanho *não* conta bytes do cabeçalho do quadro. O campo `ID` (16 bits) identifica o quadro transmitido ou confirmado. Os campos `length` e `ID` devem se transmitidos usando *network byte order* (*big-endian*).

DIAGRAMA 1: Formato de um quadro DCCNET

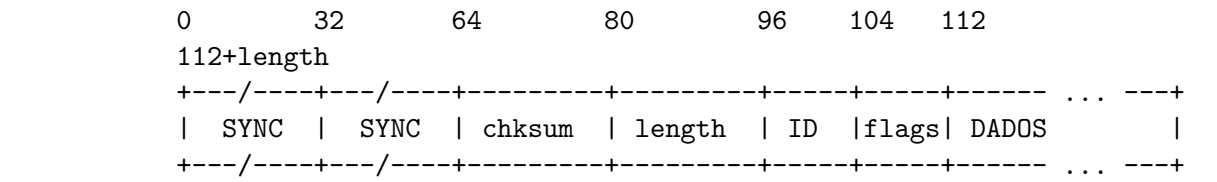

DIAGRAMA 2: Bits de controle do campo flags

Máscara (hex)	Bit	Nome	Função
1000 0000 (0x80)	7	ACK	Confirmação de recebimento de dados
0100 0000 (0x40)	6	END	Bit indicativo de fim da transmissão
0011 1111 (0x3f)	5-0	---	Reservado. Devem ser sempre zero.

Sequenciamento

Nesta versão do trabalho iremos emular a camada de enlace DCCNET usando o algoritmo *stop-and-wait* de controle de fluxo, isto é, as janelas de transmissão e recepção terão tamanho um. Consequentemente, os valores do

campo `ID` se restringem a zero ou um. Abaixo detalhamos o funcionamento de um nó para transmitir e receber dados. Note que, como os enlaces DCC-NET são *full-duplex* (dados podem ser transmitidos nas duas direções), cada nó funciona como transmissor e receptor simultaneamente.

Transmissor

Para transmitir dados, o nó transmissor cria um quadro como especificado no diagrama 1. Quadros de transmissão de dados possuem valor maior do que zero no campo `length`, a não ser que um arquvio seja vazio. O campo `ID` contém o identificador do quadro. Enquanto o quadro transmitido não for confirmado, ele deverá ser retransmitido (periodicamente) a cada segundo. Após recebimento da confirmação, o transmissor deve trocar o valor do campo `ID` (de zero-para-um ou de um-para-zero) e transmitir o próximo quadro.

Receptor

Um quadro de dados só pode ser aceito se ele tiver identificador (`ID`) diferente do identificador do último quadro recebido[1] e se nenhum erro for detectado (ver abaixo). Ao aceitar um quadro de dados, o receptor deve criar um quadro de confirmação. Quadros de confirmação não carregam dados, têm o flag ACK ligado e `length = 0`. O campo `ID` do quadro de confirmação deve ser idêntico ao do quadro confirmado.

O receptor deve manter em memória o identificador (`ID`) e o *checksum* (`chksum`) do último quadro de dados recebido. Se um quadro recebido for uma retransmissão do último quadro recebido, o receptor deve re-enviar o quadro de confirmação. Um quadro é considerado uma retransmissão do último quadro recebido se tiver o mesmo identificador (`ID`) e o mesmo *checksum* (`chksum`).

Deteção de Erros

Erros de transmissão são detectados usando o *checksum* presente no cabeçalho do pacote. O *checksum* é o mesmo utilizado na Internet e é calculado sobre todo o quadro, incluindo cabeçalho e dados. Durante o cálculo do *checksum*, os bits do cabeçalho reservados ao *checksum* devem ser considerados com o valor zero. Pacotes com erro não devem ser aceitos pelo destino nem confirmados.

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

Para detectar o início de cada quadro, sempre, seu emulador deve esperar por duas ocorrências do padrão de sincronização (`SYNC`) que marcam o início de um quadro. (Já que o quadro no qual o erro aconteceu não será confirmado, o transmissor irá retransmiti-lo periodicamente.)

Indicativo do fim da transmissão

Quando o transmissor envia o último quadro de dados ele deve ligar o bit `END` do campo `flags` para indicar que não haverão mais quadros de dados após este quadro, ou enviar um quadro vazio com aquele flag ligado e um novo número de sequência. O receptor deve confirmar o quadro que tem um flag END ligado normalmente e guardar a informação de que o outro lado não tem mais dados para transmitir. Note que *apenas o último quadro de dados* deve ter o `flag END` ligado; em particular, quadros de confirmação (contendo `length = 0` e `flag` de `ACK` ligada) *nunca* devem ter o `flag END` ligado. Quando um processo recebe (e confirma) o último quadro de dados (com `flag END` ligado) e o programa não tem mais o que transmitir, ele deve fechar a conexão e terminar a execução.

Implementação

Você deverá implementar o DCCNET sobre uma conexão TCP.[2] Seu emulador do DCCNET deve ser capaz de funcionar como transmissor e receptor simultaneamente. Seu emulador deve escrever em um arquivo todos os dados recebidos. Seu emulador deve ler os dados a serem transmitidos de um arquivo.

Seu emulador deve interoperar com outros emuladores (teste com emuladores dos colegas), inclusive com o emulador de referência implementado pelo professor.

Avaliação

Este trabalho deve ser realizado em grupo de até dois alunos. O trabalho pode ser implementado em Python, C ou C++, mas deve interoperar com emuladores escritos em outras linguagens. Seu programa deve usar apenas as bibliotecas padrão (com interface similar à de soquetes UNIX).

Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor; se confirmada a incoerência ou ambiguidade, o aluno que

[2] Utilize `socket(AF_INET, SOCK_STREAM, 0)` para criar o *socket*. Isto simplifica o desenvolvimento do trabalho.

a apontou receberá um ponto extra. O aluno 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.

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.)

Interface com o usuário

O emulador na ponta passiva (servidor) do enlace DCC023C2 deve ser executado como segue:

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

Onde `PORT` é um número entre 51000 e 55000 indicando 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 <IPPAS>:<PORT> <INPUT> <OUTPUT>
```

Onde os parâmetros `PORT`, `INPUT` e `OUTPUT` tem o mesmo significado acima. O parâmetro `IPPAS` é o endereço IP da máquina onde o emulador na ponta passiva está executando[3]. Note que no final o arquivo de saída

[3]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 da máquina onde ele está executando (em Python, pode-se usar `socket.gethostbyname(socket.getfqdn())`.

do cliente (servidor) deve ter o mesmo conteúdo do arquivo de entrada do servidor (cliente).

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):

```
dcc023c2dcc023c2faef0004000001020304
```

Sobre o teste com erros

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 nunca bytes errados na chegada de uma conexão TCP (os bytes recebidos com recv corresponderão exatamente aos bytes enviados com send). Entretanto, nos testes podemos usar programas que geram bytes errados intencionalmente no envio, para verificar se seu programa detecta corretamente erros de sincronização, checksum, etc.

Para testar seu programa nesses casos você pode criar versões alteradas do programa que geram intencionalmente quadros errados no envio, para ver como o programa do outro lado se comporta. Não é necessário entregar esses programas usados para teste, apenas o program que funciona exatamente conforme esta especificação. O professor criará seu programa com geração de diferentes tipos de erros para usar durante os testes.