Enquadramento de dados

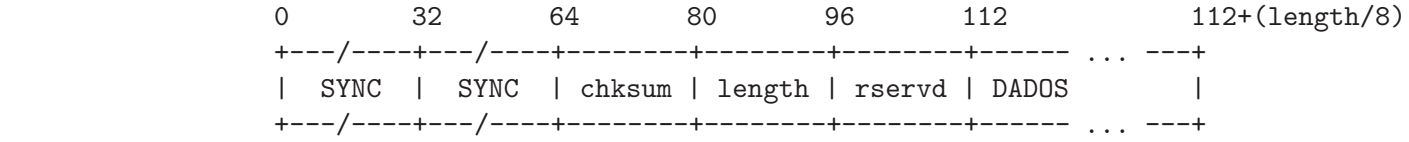
Introdução

Neste trabalho iremos desenvolver um emulador de enquadramento de dados para comunicação em redes usando a camada de enlace fictícia DCCNET. O software irá implementar sequenciamento, enquadramento e detecção de erros.

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.[1] O campo de tamanho (`length`, 16 bits) conta a quantidade de *dados* transmitidos no quadro; o campo de tamanho *não* conta bytes do cabeçalho do quadro. O campo `rservd` (16 bits) é reservado para uso futuro e não será utilizado neste trabalho. O campo `length` deve ser transmitido usando *network byte order* (*big-endian*).

DIAGRAMA 1: Formato de um quadro DCCNET



Transmissor

Para transmitir dados, o nó transmissor cria um quadro como especificado no diagrama 1, os campos `checksum` e `length` devem ser inicializados, respectivamente, com o valor do *checksum* do quadro e com com a quantidade de bytes de dados enviados no quadro. Todos os quadros transmitidos devem possuir valor maior do que zero no campo de tamanho. Quadros são transmitidos de forma assíncrona e unidirecional, isto é, o transmissor não espera nenhum sinal ou confirmação do receptor

[1] RFC1624, `https://tools.ietf.org/html/rfc1624`.

para transmitir. Note que não existem confirmação ou retransmissão de quadros.

Receptor

Um quadro de dados só pode ser aceito se ele for válido. A validade de um quadro inclui verificação do valor do campo `length` (que deve ser maior do que zero) e do `checksum`. Ao aceitar um quadro de dados, o receptor deve escrever os dados recebidos (o *payload*) em um arquivo. Note que não há retransmissão de dados em caso de erro na recepção.

Após o recebimento de duas ocorrências do padrão de sincronização, o receptor deve iniciar o recebimento de um quadro. Para isso o programa deve:

1. Esperar uma quantidade de bytes correspondente ao valor do campo `length`. Ocorrências de padrões de sincronização deve durante o recebimento destes bytes devem ser ignoradas.

2. Calcular o *checksum* do pacote para detectar erros como descrito abaixo.

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. Apesar do campo `rservd` não ser utilizado neste trabalho, ele deve ser considerado no cálculo do *checksum* (note que o campo `rservd` pode ter valor arbitrário). 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, 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 um o quadro iniciando com os marcadores, verificando o *checksum* de todo o *payload*. Como descrito acima, o *payload* é definido pelo valor do campo `length`.

Note que o receptor pode ficar sem saber quando os quadros começam ou terminam, por exemplo, devido a (1) erro de transmissão no campo `length` de um quadro bem como (2) inserção (ou remoção) de bytes corrompidos (lixo) no canal. Erros de transmissão, inserção ou remoção de

bytes 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.

O receptor deve receber corretamente e imprimir no arquivo de saída os *payloads* de todos quadros válidos recebidos.

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 imprimir em um arquivo, em hexadecimal, os dados nos *payloads* de todos os quadros recebidos.

Seu emulador deve ler os dados a serem transmitidos de um arquivo. O conteúdo do arquivo deverá ser enquadrado e transmitido ao receptor na outra ponta do enlace emulado.

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

Execução

Seu programa deve receber cinco parâmetros como abaixo (na mesma ordem):

```
./emulador <in> <out> <ip> <port> <mode>
```

Onde `<in>` é o nome do arquivo de entrada a ser enquadrado e transmitido; `<out>` é o nome do arquivo onde os *payloads* de quadros válidos devem ser armazenados; `<ip>` e `<port>` indicam um endereço IP e porta; e `<mode>` descreve o modo de operação do emulador. O modo de operação deve ser indicado por extenso na linha de comando; existem dois modos de operação:

- `ativo`: No modo `ativo`, o programa conecta-se ao IP e porta indicados para começar a transmitir e receber dados.
- `passivo`: No modo `passivo`, o programa abre um soquete para recebimento de conexões no IP e porta indicados. A transmissão e

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

recepção de dados acontece após a conexão de um cliente. O servidor pode terminar depois que a conexão com o primeiro cliente for terminada (não é preciso tratar mais de um cliente).

Terminação

O emulador pode parar de executar sempre que uma 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.

Avaliação

Este trabalho é em dupla. O trabalho pode ser implementado em Python, C, C++ ou Java, mas deve interoperar com emuladores escritos em outras linguagens. Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor; se confirmada a incoerência ou ambiguidade, o aluno que a apontou receberá dois pontos extras. 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.

O arquivo `<out>` gerado pelo seu programa deve conter *apenas* os *payloads* dos pacotes recebidos corretamente. O arquivo `<out>` não deve contar comentários, mensagens de depuração ou qualquer outra saída. *Todos* os dados contidos no arquivo `<out>` serão considerados como recebidos pela rede e serão considerados na avaliação da corretude do seu programa.

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 (usaremos programas que geram bytes errados intencionalmente no envio).

Para verificar se seu programa detecta corretamente erros de sincronização, 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 programa que funciona exatamente conforme esta especificação.

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 campo `rservd` tem valor zero):

```
dcc023c2dcc023c2faef0004000001020304
```

Note que se o seu emulador deve ser capaz de receber corretamente o quadro acima mesmo após erros de transmissão que se assemelham ao início de um quadro. Por exemplo, no exemplo abaixo, seu emulador deve ser capaz de receber corretamente o quadro válido.

```
/--- erro de transmissão ---\/- lixo -\/--- quadro válido ----------------\
dcc023c2dcc023c2CCCC0001AAAA12345678901dcc023c2dcc023c2faef0004000001020304
```