Universidade Federal de Minas Gerais
DCC023: Redes de Computadores
Trabalho Prático

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 erros, sequenciamento e retransmissão de dados.

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. 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.

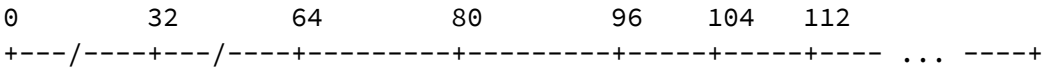
Tabela 1: Exemplo de sequência de quatro bytes em diferentes codificações

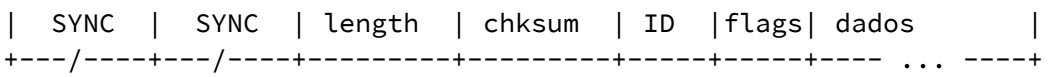
Bits	`11011100`	`11000000`	`00100011`	`11000010`
Decimal	`220`	`192`	`35`	`194`
Hexadecimal	`0xDC`	`0xC0`	`0x23`	`0xC2`
Base16 (em ascii)	`DC`	`C0`	`23`	`C2`
Base16 (em bin)	`01000100` `01000011`	`01000011` `00110000`	`00110010` `00110011`	`01000011` `00110010`

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 na tabela 1. O campo de tamanho [length, 16 bits] conta a quantidade de bytes de dados [dados, length * 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 (*big endian*).

DIAGRAMA 1: Formato de um quadro DCCNET





Dica: Verificar como montar sequências de bytes com dados específicos (p.ex., inteiros de 16 bits) na linguagem escolhida. Por exemplo, no Python, use o pacote struct (funções pack e unpack).

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 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.

Transmissão

Para transmitir dados, o nó transmissor cria um quadro, como especificado no diagrama 1. 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. Todos os quadros transmitidos devem possuir valor maior do que zero no campo de tamanho [length]. Quadros são transmitidos de forma assíncrona e unidirecional, isto é, o transmissor pode enviar um quadro a qualquer momento (p.ex., devido a um temporizador).

Recepção

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

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[1] e se nenhum erro for detectado (seção sobre o checksum). 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.

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

Tabela 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`	Informação de término de transmissão.
`0011 1111 (0x3f)`	`5-0`	`---`	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.

Término da Comunicação

Quando um nó termina de transmitir dados, ele informa o nó remoto enviando um *quadro de terminação*. O quadro de terminação não carrega dados (i.e., [length] igual a zero) e possui a [flag] de terminação (END, 0x40) ligada. Um nó não pode enviar quadros com dados após enviar um quadro de terminação. Ao receber um quadro de terminação, o nó deve confirmá-lo com uma mensagem de confirmação [ACK] normalmente. Quando os dois nós tiverem recebido confirmações dos seus respectivos quadros de terminação, a execução termina e o enlace DCCNET é desligado.

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 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] e eventualmente recuperar o enquadramento depois de um erro de transmissão.

Retransmissão

Como as transmissões podem sofrer erros, a DCCNET utiliza confirmação e retransmissão de quadros para garantir a 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.

De forma idêntica aos quadros de dados, o quadro de terminação deve ser retransmitindo periodicamente até que seja confirmado, e o receptor deve reenviar a confirmação caso receba o quadro de terminação múltiplas vezes.

Implementação e Interface de Usuário

Você deverá implementar o DCCNET sobre uma conexão TCP [SOCK_STREAM]. 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. Discuta com o professor a possibilidade de implementar o trabalho em outras linguagens.

Inicialização

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

```
./dcc023c2 -s <port> <input> <output>
```

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. A transmissão e recepção de dados acontece após a conexão de um cliente. O servidor pode terminar depois que a execução terminar (não é preciso tratar mais de um cliente).

O emulador na ponta ativa (cliente) do enlace DCCNET 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.[2] Note que no final o arquivo de saída do cliente (servidor) deve ter o mesmo conteúdo do arquivo de entrada do servidor (cliente).

Execução

Seu emulador deve ler os dados a serem transmitidos (possivelmente 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 de decodificados. (Os cabeçalhos *não* devem ser escritos no arquivo [output].)

Terminação

O emulador deve parar de executar quando receber um sinal de interrupção de teclado [ctrl-c], isto é, quando for terminado manualmente. O emulador também deve 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 (p.ex., caso o emulador do nó remoto feche devido a algum erro). Caso nenhuma destas situações ocorra, o emulador deve executar até que os quadros de terminação sejam trocados como descrito acima.

Sugestão de Estratégia de Implementação

Abaixo segue uma passo a passo sugerido para implementação do trabalho:

1. Estabelecimento de conexão TCP com o nó remoto.
2. Codificação, envio, recebimento e decodificação usando o base16 unidirecionalmente. [Tabela 1]
    a. Implemente primeiro a comunicação unidirecional (um programa envia e o outro recebe).
    b. Teste a corretude do seu código imprimindo dados em várias etapas do processamento (em ordem): imprima o dado original, o dado codificado, o dado recebido e o dado decodificado.
3. Construção de quadros [Diagrama 1], bem como correto envio e recebimento usando base 16.
    a. Teste a corretude do enquadramento imprimindo os valores de cada campo após recebimento e decodificação.
4. Implementação da confirmação de quadros recebidos [ACK].
    a. Neste ponto seu programa já consegue transferir arquivos unidirecionalmente usando o DCCNET *quando não ocorrem erros*.

[2] 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 os comandos [ifconfig] ou [ip addr] no Linux, [ipconfig /a] no windows, ou fazer o seu programa escrever o endereço IP da máquina onde ele está executando.

    b. Teste a corretude do seu código enviando arquivos entre os nós e verificando que os arquivos recebidos estão íntegros (i.e., são idênticos aos originais).
5. Cálculo e verificação do [checksum].
    a. Teste a corretude do [checksum] utilizando calculadoras online e compare com o exemplo abaixo.
6. Implemente temporizadores para retransmitir periodicamente quadros caso a confirmação não seja recebida.
    a. Neste ponto seu programa já consegue transferir arquivos unidirecionalmente usando o DCCNET *quando erros não corrompem o comprimento de quadros*.
    b. Teste a corretude de seu código enviando quadros corrompidos (e.g., com o [checksum] calculado incorretamente (estes quadros devem ser descartados e retransmitidos).
7. Ajuste seu programa para permitir envio e recebimento de arquivos simultaneamente.
    a. Teste seu programa enviando e recebendo arquivo em cada programa. Verifique que a transferência está ocorrendo em paralelo. Em particular, uma patologia comum é o programa A enviar o arquivo completo enquanto o programa B recebe, e depois os programas trocam de papel (B envia e A recebe); este comportamento não é válido.
8. Implemente a lógica para ressincronização do enlace buscando pelas sequências de sincronização [SYNC] nos bytes transmitidos.
    a. Neste ponto seu programa consegue transferir arquivos usando o DCCNET mesmo quando ocorrem erros.
    b. Teste a corretude de seu código enviando quadros com o campo [length] corrompido ou inserindo lixo no enlace (p.ex., bytes aleatórios).

Avaliação

Este trabalho deve ser realizado individualmente. Seu programa deve rodar no sistema operacional Linux e, em particular, não deve utilizar bibliotecas do Windows, como o winsock. Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor.

Entrega

Deve ser entregue o código fonte na linguagem escolhida (**Python, C, C++, Java ou Rust**) e **um arquivo README com as instruções para compilação (se houver) e execução**. Não é permitido o uso de bibliotecas externas. Além disso, deve ser entregue também um arquivo PDF de documentação, descrito a seguir. **Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.**

Documentação

Cada 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.
- Detecção de erros.
- Retransmissão periódica de quadros.
- Terminação correta da execução e desligamento do enlace.
- Recuperação do enquadramento após erros de transmissão.

Usaremos programas que geram bytes errados intencionalmente no envio para emular erros e exercitar o código de detecção de erros e recuperação de enquadramento. (Note que para realizar este teste 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 bem como 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 (**esse programa não será publicado**).

Entrega Parcial

A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente. A funcionalidade correspondente aos passos 1-4 da sugestão de estratégia de implementação vale 30% dos pontos do trabalho. A funcionalidade correspondente aos passos 5 e 6 da sugestão de estratégia de implementação vale 30% dos pontos do trabalho. A funcionalidade correspondente ao passo 7 da sugestão de estratégia de implementação vale 20% dos pontos do trabalho.

Desconto de nota por atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades. A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = 2^{d-1} \div 0.32\ \%$$

onde d é o atraso em dias úteis. Note que após 5 dias, o trabalho não pode ser mais entregue.

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):

```
DIAGRAMA 2: Exemplo de quadro DCCNET

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

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.

```
DIAGRAMA 3: Exemplo de recepção após erro de transmissão:

/--- erro de transmissão ---\/- lixo -\/--- quadro válido ----------------\
dcc023c2dcc023c2CCCC0001AAAA12345678901dcc023c2dcc023c20004faef000001020304



0123456789abcdefghik0123456789abcdefghik
01234567
```

```
 11011100         11000000      00100011              11000010
```