

**DCC023: Redes de Computadores**  
**Documentação Trabalho Prático 1 - DCCNET**

Daniel Oliveira Souza [2018019435] - Maria Fernanda Fávaro [2018019532]

Este relatório tem por objetivo apresentar os principais pontos relativos ao desenvolvimento e funcionalidades, assim como os principais desafios, do programa proposto no Trabalho Prático 1, da disciplina de Redes de Computadores da UFMG do primeiro semestre de 2021. Neste trabalho foi desenvolvido um emulador de camada de enlace para uma rede fictícia chamada DCCNET.

### Implementação

A construção do sistema foi realizada na linguagem C# (DotNet Core), utilizando da biblioteca System.Net.Sockets. A estratégia adotada foi a mesma sugerida na especificação do trabalho proposto. A mesma foi feita da seguinte forma:

- **Estabelecimento de conexão TCP com o nó remoto:** Ao iniciar o emulador, é definido se a comunicação será realizada por meio passivo (servidor) ou ativo (cliente), de tal forma a ser configurado o socket para cada nó nas respectivas funções.

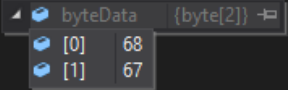
```
if (args[0] == "-c")
{
    FileInput = args[3];
    FileOutput = args[4];
    // Cliente configure
    AsynchronousClient.StartClient(args[1], args[2]);
}
else if (args[0] == "-s")
{
    FileInput = args[2];
    FileOutput = args[3];
    // Server configure
    AsynchronousSocketServer.StartListening(args[1]);
}
```

Em ambas as funções, o Socket foi configurado para IPv4 e IPv6. Após estabelecida a conexão, chama-se a função Master Loop.

Na região "Server" e na região "Client" do código foram definidas algumas funções assíncronas necessárias para a conexão. Para criar um novo nó, o server estabelece uma nova thread para o mesmo.

- **Codificação, envio, recebimento e decodificação usando o base16 unidirecionalmente:** A codificação e decodificação foi feita conforme a especificação do trabalho, utilizando de métodos *GetBytes()* e *GetString()* da biblioteca Encoding.ASCII. Assim como pode ser visto no exemplo abaixo, uma mensagem em hexadecimal "DC" foi convertida para um array de 68 67 em bits (01000100 01000011), conforme esperado:

```
var dataHexToSend = "DC";
byte[] byteData = Encoding.ASCII.GetBytes(dataHexToSend);
```



- **Construção de quadros, bem como correto envio e recebimento usando base 16:** Foi utilizada a classe Frame para definição do enquadramento:

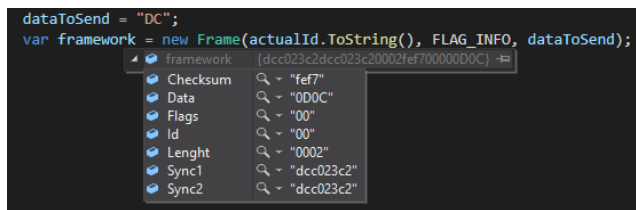
```
public class Frame
{
    public string Sync1;
    public string Sync2;
    public string Length;
    public string Checksum;
    public string Id;
    public string Flags;
    public string Data;
```

Os atributos da classe são os campos do quadro.

Nesta classe também foi criado um construtor que recebe como parâmetros a flag, o ID e o dado a ser enquadrado, de tal forma a ser calculado o checksum conforme o cabeçalho construído.

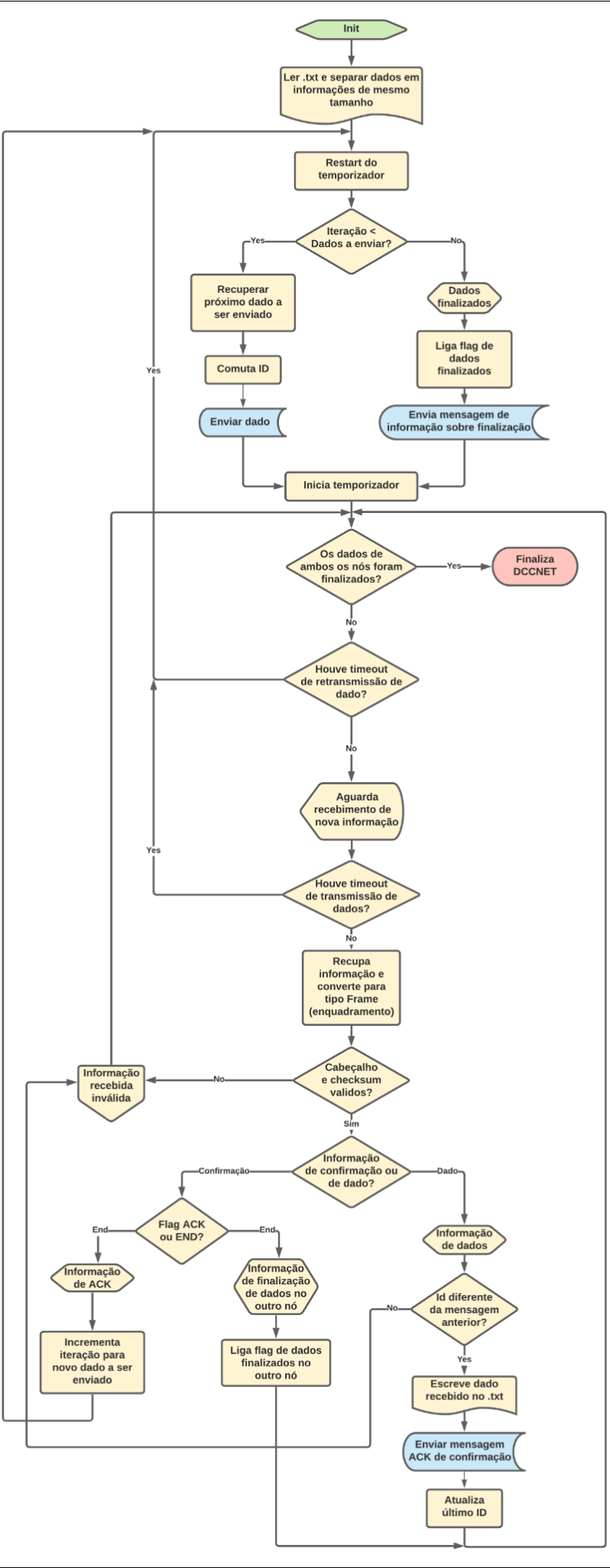
Neste construtor também é definido os campos de SYNC e concatenado a informação do campo 'dados' com o caractere '0'. Além disso é possível que seja definido as

flags e o checksum do frame em questão, de tal forma a ser utilizado em testes de erros do DCCNET. Abaixo é possível ver um exemplo da chamada de construção de um novo frame.



Nesse sentido, para enviar ou receber um novo dado, foram definidos métodos *SendDataInfoMessage()* e *WaitReceiveMessage()* que realizam o enquadramento ou então transformam uma string de conteúdo para o formato Frame.

- **Implementação da confirmação de quadros recebidos [ACK]:** Conforme será apresentado mais adiante, a lógica do DCCNET implementada na forma de loop verifica o tipo de informação recebida (frame ACK ou de dados). Após receber um dado, o programa envia um novo quadro para o outro nó da comunicação com as Flags definidas como "0x80" e sem dados. O método responsável por este processo foi nomeado como *SendConfirmationACK()*. O emulador só envia o próximo dado após receber um quadro de confirmação.
- **Cálculo e verificação do checksum:** O checksum é o campo do quadro utilizado para verificar se nenhum dos campos do cabeçalho foi enviado com erro. Ele foi feito exatamente conforme o utilizado na internet (IPv4 header checksum).  
A função *CalculateChecksum()* é utilizada tanto no emissor quanto no receptor. No emissor, o checksum é iniciado com o 0 e é calculado conforme o conteúdo dos outros campos do cabeçalho. No receptor, é utilizado o valor recebido e para que haja confirmação, o valor resultante deve ser "0000".
- **Implementação de temporizadores para re-transmitir periodicamente quadros caso a confirmação não seja recebida:** Após enviar uma nova mensagem, o loop da DCCNET inicia um novo contador que é verificado a cada iteração da repetição de recebimento de uma nova informação do outro nó. Quando o limite é estourado, o loop "while" é quebrado e o programa consequentemente envia uma nova mensagem com o mesmo dado. Também foram definidos os timeout do método de recebimento de uma nova mensagem em ambos nós da comunicação. Para isto, foi configurado a opção "*ReceiveTimeout*" dos sockets. Assim sendo, ao receber um quadro de confirmação ACK de uma mensagem enviada, o contador é zerado e o processo de validação é reiniciado.
- **Envio e recebimento de arquivos simultaneamente:** Conforme será apresentado a seguir, para implementar a comunicação simultânea entre os arquivos foram inseridos diversos loops no sistema de tal forma a ser possível sempre estar recebendo uma nova mensagem e enviando uma nova informação ao receber uma confirmação ACK. Dessa forma, a transferência de arquivos ocorre de forma paralela, de tal modo a enviar um novo dado assim que confirmado o anterior e sempre estar em um loop de repetição para receber um novo frame. É válido apontar que o cliente e o servidor utilizam do mesmo método principal.
- **Lógica para resincronização do enlace buscando pelas sequências de sincronização [SYNC] nos bytes transmitidos:** O código procura *substrings* dentro dos dados recebidos que tenham o padrão de sincronização definido. Se ele encontrar esse padrão, ele remonta o frame enviado e verifica o checksum para identificar se o que foi recebido é válido.



**MasterLoop:**

No diagrama à esquerda é possível visualizar o funcionamento do fluxo principal do método DCCNET, o qual realiza os envios e recebimentos de arquivos para ambos os nós. Após ler a informação do arquivo de input, é iniciado um loop principal responsável por enviar e receber os quadros, fazendo isso de forma paralela. Assim sendo, logo no início é construído o novo frame e é definido o novo ID deste. Então, é iniciado o temporizador responsável por contabilizar o timeout de não recebimento de confirmação, gerenciando assim a retransmissão de dados.

Logo a seguir, existe um loop interno e secundário que espera uma nova informação do outro nó. Após receber um novo buff de dados, é realizado diversas operações para testes de erros e conversão para a classe Frame. Nesse sentido, é realizada uma busca na string recebida procurando o padrão de sincronização. Para cada ocorrência deste, é verificado o checksum do cabeçalho e definido se o dado será aceito ou não. Caso nenhum erro seja detectado, é efetuado uma classificação do frame entre informação ou quadro de confirmação, verificando o valor do campo "Length" do Frame.

Em Frames de confirmação, é identificado a Flag passado, definindo entre ACK e END. Já em Frames de informação, caso o ID recebido seja diferente do anterior, é escrito no arquivo de output o dado recebido, enviado um novo quadro de confirmação ACK, atualizado em uma variável a identificação [ID] em questão é incrementado um iterador, de tal forma a estar configurado para ser enviado o próximo dado. Neste pronto, o loop interno é parado e o fluxo do DCCNET volta ao iniciando, enviando o próximo quadro de informação.

Com essa lógica, é realizado para cada informação recebida o teste de checksum, e é enviado um quadro de confirmação ACK apenas se o quadro recebido for aceito. Assim, caso não seja recebido um quadro ACK dentro do tempo máximo configurado, é realizada a retransmissão da última mensagem enviada, já que a verificação do tempo decorrido acontece a cada iteração de recebimento.

## Testes

Para testar a comunicação e validar os pontos de transmissão e recepção de dados em paralelo, detecção de erros, retransmissão periódica de quadros, terminação correta da execução e desligamento do enlace e recuperação do enquadramento após erros de transmissão, foi modificado o arquivo do emulador com o objetivo de injetar quadros fora do esperado no meio do processo. Assim como pode ser visto abaixo, em pontos específicos foi testado o comportamento para casos onde o checksum é enviado errado, o cabeçalho não possui os parâmetros de SYNC ou quando um dado é corrompido, simulando erros de transmissão que se assemelham ao início de um quadro.

```
if (!dataFinishedFlag)
{
    if ((data >= dataFiles.Count - 1) && flagTeste1)
    {
        flagTeste1 = false;
        actualId = SendDataInfoMessage(socket, actualId, dataToSend, "FFFF");
    }
    else if ((data >= dataFiles.Count / 2) && flagTeste2)
    {
        flagTeste2 = false; // (variável local) List<string> dataFiles
        actualId = SendDataInfoMessage(socket, actualId, dataToSend, null, "ErrorSyncErrorSync");
    }
    else if ((data == 0) && flagTeste3)
    {
        Console.WriteLine("-- Send fixed error frame ==");
        actualId = actualId == 0 ? 1 : 0;
        flagTeste3 = false;
        var checksumCalculatedError = CalculateChecksum("dcc023c2dcc023c20004" + "0000" + actualId.ToString().PadLeft(2, '0') + "00");
        var framework = "dcc023c2dcc023c2CCCC0001AAAA12345678901dcc023c2dcc023c20004" + checksumCalculatedError + actualId.ToString().PadLeft(2, '0') + "0001020304";
        Console.WriteLine("Send Framework: " + framework);
        byte[] byteData = Encoding.ASCII.GetBytes(framework);
        socket.Send(byteData);
    }
    else
    {
        actualId = SendDataInfoMessage(socket, actualId, dataToSend);
    }
}
```

Em todos, a re-transmissão e detecção de erros ocorreu com exatidão, assim como esperado no desenvolvimento. Um exemplo de execução destes testes pode ser visto abaixo:

```
Restarted time
Data: 0
Send Framework: dcc023c2dcc023c20212FFFF01000504010908010901060501080709040506070409080406050109080704050401090801090807010605040808090109080
108070901060504030507040504010908010901060501080709040506070409080406050109080704050401090801090807010605040808090109080108070901060504030507
040504010908010901060501080709040506070409080406050109080704050401090801090807010605040808090109080108070901060504030507040504010908010901060
501080709040506070409080406050109080704050401090801
Timer before check timeout: 15
Waiting receive message...
Content received: dcc023c2dcc023c20212FFFF01000504010908010901060501080709040506070409080406050109080704050401090801090807010605040808090109080108070901060504030507
040504010908010901060501080709040506070409080406050109080704050401090801090807010605040808090109080108070901060504030507040504010908010901060501080709040506070409080406050109080704050401090801
60501080709040506070409080406050109080704050401090801
Decode Received Framework: dcc023c2dcc023c20212FFFF010005041981916518794567498465198745419819871654891981879165435745419819165187945674984651
98745419819871654891981879165435745419819165187945674984651987454198198716548919818791654357454198191651879456749846519874541981
[Warning] Error checksum or SYNC - Data not accepted
Timer before check timeout: 22
Waiting receive message...
Timeout receive message - Resend - Time elapsed: 1s

Restarted time
Data: 0
Send Framework: ErrorSyncErrorSyncErrorSyncErrorSync0212fcd70000050401090801090106050108070904050607040908040605010908070405040109080109080
```

Neste print de exemplo, é possível ser visto como o quadro é enviado e recebido (após decodificado). No caso, o checksum foi corrompido para "FFFF", simulando um erro de cabeçalho, e o emulador conseguiu detectar com sucesso o erro, não aceitando o dado e consequentemente não enviando uma confirmação ACK. Consequentemente, o temporizador no nó emissor foi estourado, o que fez com que a última informação fosse re-transmitida. Também é possível ser notado que a comunicação acontece de forma simultânea, já que em uma única iteração é recebido uma fração dos dados em forma de quadro e também enviado um novo, seguindo a lógica apresentada no diagrama na página anterior.

## Para executar o emulador em C#

Exemplo em windows:

```
dotnet run -- -s 5151 inputServer.txt outputServer.txt
```

Exemplo em linux:

```
mcs -out:dcc023c2.exe Dcc023c2.cs
```

```
mono dcc023c2.exe -s 5151 inputServer.txt outputServer.txt
```

Observações: Em linux, é necessário ter instalado os pacotes 'chicken-bin' e 'mono-complete'.

- apt install chicken-bin
- sudo apt install mono-complete

Também foi desenvolvido um makefile para compilação do emulador.

### **Principais desafios, dificuldades e imprevisto**

Inicialmente, houve uma grande dificuldade para construir o emulador na linguagem C. Assim sendo, devido aos conhecimentos da dupla na linguagem C#, foi alterado para o desenvolvimento nesta. Também houve diversos problemas na sincronização e comunicação paralela dos arquivos, sendo considerado relativamente complexo o desenvolvimento da lógica de loops implementada. A detecção de erros e retransmissão de informação possui diversos detalhes e procedimentos a serem seguidos. Assim sendo, a verificação do cabeçalho via cálculo do checksum e os testes ao receber uma nova informação ou gerenciamento de timeout também aumentaram a complexidade do trabalho, resultando em diversos erros na elaboração do projeto. No entanto, o emulador foi desenvolvido com sucesso, de tal maneira a ser aplicado a todas as especificações passadas.