Universidade Federal de Minas Gerais – Belo Horizonte, MG – 2019/1
Instituto de Ciências Exatas – Departamento de Ciência da Computação

DCC023: Redes de Computadores
TP1 – DCCNET: Camada de Enlace

Aluna: Scarlet Gianasi Viana (2016006891)

1. Introdução

Este trabalho teve como objetivo implementar um emulador da camada de enlace da rede DCCNET, em que um cliente e um servidor trocam arquivos remotamente.

2. A Rede DCCNET

Como especificado, a rede **DCCNET** envia quadros codificados em **Base16** com tamanho máximo de 518 bytes (antes da codificação), e a cada quadro enviado, se espera receber um quadro de confirmação para se enviar o próximo (**Stop-and-wait**). Caso o quadro não seja confirmado após um segundo, ele será retransmitido. Caso haja retransmissão e o receptor detectá-la, outro quadro de confirmação será enviado, pois isso significa que o anterior não foi recebido.

Cada quadro possui um header com um **SOF** (‘0xcc’), **ID** (‘0x00’ ou ‘0x01’), **flags** (‘0x7f’ para dados e ‘0x80’ para confirmação) e **checksum**, e o final do quadro é dado por um **EOF** (‘0xcd’). O tamanho máximo de bytes de dados que podem ser enviados é 512 bytes, e o mínimo 1 byte. Quadros de confirmação não possuem dados.

O mecanismo de detecção de erros utilizado foi o **checksum** utilizado na Internet, e quadros com checksum inválido e IDs errados são descartados, pois implicam quadros com erros ou repetidos.

Também foi utilizado **byte stuffing** para que o quadro de dados seja lido corretamente até o final, pois o caractere **EOF** pode aparecer neste campo, sem ser o real fim de quadro.

As entradas e saídas do cliente e servidor, neste implementação, *devem ser arquivos diferentes*, para proteger a integridade da leitura e escrita dos dados.

3. Implementação

A implementação deste trabalho foi feita utilizando a linguagem **Python** (versão 3.6.7), utilizando a biblioteca “**socket**”.

O emulador recebe os argumentos de entrada seguindo o modelo especificado:

– Modo servidor: **./dccnet.py -s <PORTA> <entradaServidor> <saidaServidor>**
– Modo cliente: **./dccnet.py -c <IP>:<PORTA> <entradaCliente> <saidaCliente>**

Lembrando que nenhum dos arquivos podem ser repetidos, mas como não é possível comparar os arquivos do servidor com os do cliente, neste trabalho se assume que os mesmos sempre serão diferentes. Também se assume que os arquivos de entrada do servidor/cliente serão do mesmo tipo (texto ou imagem, por exemplo) que seus respectivos arquivos de saída, mas também é possível enviar um tipo de arquivo e receber outro, contanto que o outro nó receba e envie arquivos correspondentes aos seus tipos.

O protocolo utilizado nos soquetes é o **TCP**, portanto o nó servidor abre a porta e aceita conexões do nó cliente, nesta ordem. Se assume que a conexão será sempre somente entre um servidor e um cliente.

3.1 Envio dos dados

Assim que a conexão entre os dois nós for estabelecida, a troca de arquivos se iniciará. Para enviar os dados, cada nó envia quadros com os dados codificados. Para realizar a codificação, se utiliza a **Base16** da especificação. Cada nó abre seu arquivo de entrada e começa a leitura byte a byte dos dados. Como há um máximo de bytes, 512, e sabendo que é possível existir um **EOF** ou um **DLE** que precisará ser precedido por um **DLE** para que o quadro seja lido corretamente, o byte stuffing já é realizado. Cada haja um **EOF** ou **DLE** no último byte possível, então esse deverá ficar para o próximo quadro, pois o byte stuffing iria estourar o limite do quadro.

Os dados são lidos como bytes e são tratados como uma lista de suas respectivas representações em inteiros unicode, para facilitar seu manuseio, como no cálculo do checksum. Quando não houverem mais bytes a serem lidos, o arquivo de entrada é fechado, para manter sua integridade.

Com os bytes de dados já com byte stuffing, o nó pode construir um quadro de dados com essas informações. Como especificado, o quadro de dados segue o modelo, em que cada número abaixo representa “bytes”:

SOF	**ID**	**flags**	**Checksum**	**Dados**	**EOF**
1	1	1	2	1 a 512	1

O primeiro ID do quadro é por default 0, e cada quadro novo alterna seus IDs de 0 para 1, de forma a garantir que todos os quadros sejam confirmados.

Os bytes do checksum são zerados inicialmente, para permitir seu cálculo. Assim que o checksum for calculado, eles serão substituídos no quadro.

Com o quadro formado, o nó pode então realizar o envio do quadro para o nó receptor. Para realizar o envio, primeiro é necessário codificar o quadro em **Base16**, que retornará uma string de caracteres ASCII com a representação em hexa de cada byte a cada dois caracteres.

Um exemplo do processo de encodificação abaixo:

– Arquivo de texto, conteúdo: “**Hello, World!**”
– Após leitura dos dados e cálculo do checksum: **[204, 0, 127, 240, 93, 72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108, 100, 33, 10, 205]**
– Após *encode16*: "**cc007ff05d48656c6c6f2c20776f726c64210acd**"

Após o envio de um quadro, outros quadros, se ainda houverem dados a enviar, só poderão ser enviados quando o nó receber a confirmação do quadro a partir de um quadro de confirmação com o mesmo ID do quadro enviado, já que a rede utiliza o algoritmo **Stop-and-wait**. Caso o quadro de confirmação não seja recebido, o quadro de dados é retransmitido, e isso se repete a cada segundo até o quadro de confirmação ser recebido. Isso pode levar a um loop infinito, caso o nó receptor nunca envie o quadro nem encerre a conexão, e não foi especificado nenhum tratamento de erros para esse caso, então se assume que, caso ocorra, o usuário pode encerrar o programa utilizando a interrupção do teclado.

3.2 Recebimento dos dados

Cada nó buscará inicialmente um **SOF**, recebendo 2 bytes pelo soquete por vez (pois cada byte em **Base16** utiliza 2 bytes). Quando um início de quadro for encontrado, o nó passará a ler o restante do quadro. O **ID** e **flags** são lidos de uma vez, e a partir das flags pode se identificar se o quadro recebido é um quadro de dados ou um quadro de confirmação (e também se não é um real início de quadro, ou se possui algum erro, pois se as flags não corresponderem aos caracteres esperados o quadro é descartado).

Se for um quadro de dados, se iniciará a leitura do checksum e dos dados. No recebimento dos dados os caracteres de escape permitem ler o quadro integralmente, até se encontrar um **EOF** que não foi precedido de um **DLE**, inicando o fim do quadro.

Com o quadro inteiro recebido com sucesso, pode se testar o **checksum** do quadro. O checksum utilizado foi o da Internet, e para checar se o mesmo é válido, basta realizar o checksum do quadro recebido, e o resultado deve ser 0. Se o checksum do quadro for validado, o mesmo é salvo (para checar retransmissões) e o quadro pode ser decodificado e *destuffed*. Somente assim os dados recebidos no quadro podem ser escritos na saída do receptor.

O arquivo de saída é aberto para escritas binárias de forma a anexar os bytes, para que a cada quadro o arquivo possa ser fechado, dado que não é possível saber quando não haverá mais dados a escrever no mesmo.

Com um quadro de dados recebido com sucesso, o nó deverá transmitir então o quadro de confirmação, que deverá possuir o mesmo ID que o quadro de dados. O quadro de confirmação, como especificado, possui o mesmo cabeçalho e modelo que o outro, só que não possui o campo de dados e sua flag é diferente. No mais, sua construção e envio são iguais aos de dados, descritos no tópico anterior.

Caso o quadro recebido seja um quadro de confirmação, será checado seu ID e checksum do quadro anterior, para identificar retransmissões. Caso seja um quadro válido que confirme o quadro enviado, o receptor pode então enviar novos quadros e atualizar o ID e o checksum.

3.3 Troca de dados

Em resumo, cada nó enviará quadros, esperará a confirmação dos mesmos, receberá quadros de dados e confirmação do outro nó, e escreverá os dados ou retransmitirá quadros conforme o decorrer da troca de dados. Como podem haver arquivos de tamanhos diferentes entre os nós, cada nó continua sua execução mesmo ao terminar de enviar seus quadros de dados, sempre buscando receber mais quadros de dados, possivelmente. Quadros com erros de qualquer forma (ID errado, checksum inválido, tamanho que ultrapasse o limite, flag inválida, etc.) são descartados.

A execução do emulador pode ser encerrada com a interrupção do teclado, e, se todos os quadros forem transmitidos inteiramente, os arquivos de saída estarão preenchidos com todos os dados enviados pelos respectivos nós.

4. Pseudo-código

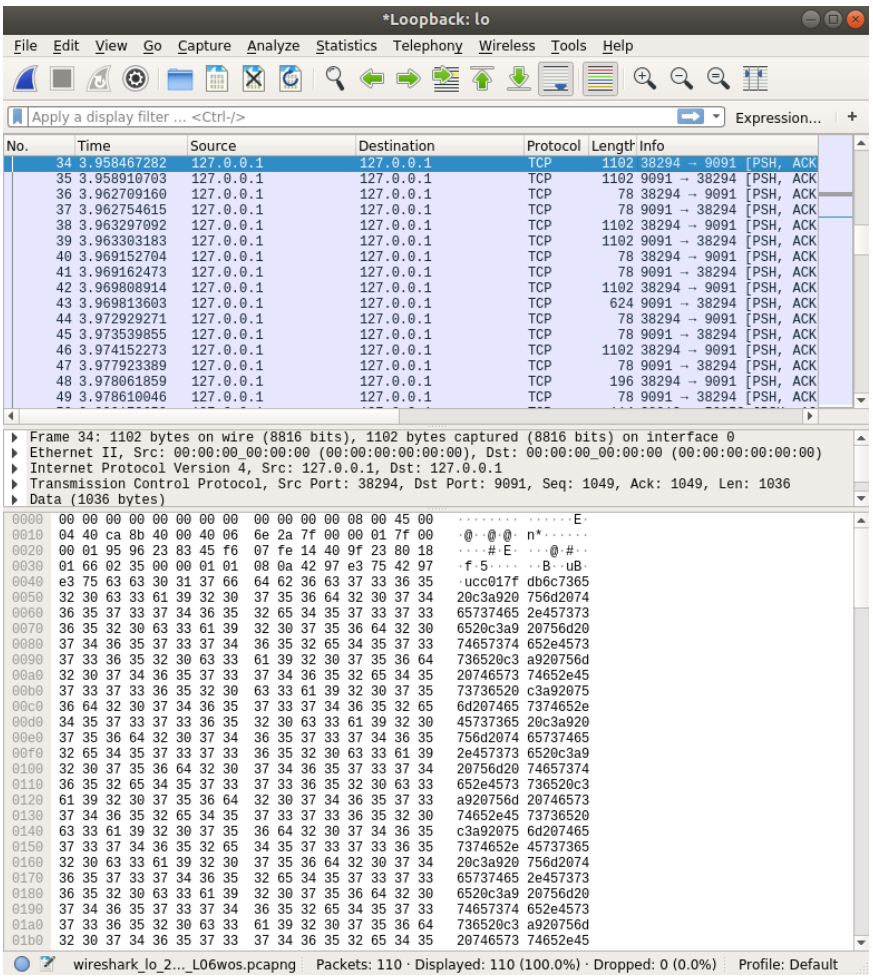
O pseudo-código abaixo demonstra a estrutura geral do emulador implementado:

```
-Inicia conexão entre os nós
-Lê entrada
-Constrói quadro de dados
-Envia quadro
-Enquanto o programa executar:
	-Enquanto não houver confirmação:
		-Recebe bytes até encontrar SOF:
			-Lê ID e flags
				-Quadro de Dados:
					-Lê restante do quadro
					-ID e checksum válidos:
						-Atualiza ID e checksum
						-Escreve dados na saída
						-Envia quadro de confirmação
					-ID e checksum inválidos:
						-Descarta quadro
				-Quadro de confirmação:
					-Lê restante do quadro
					-ID e checksum válidos:
						-Atualiza ID
						-Lê entrada
						-Se ainda há dados a enviar:
							-Constrói quadro de dados.
							-Envia quadro
					-ID e checksum inválidos:
						-Descarta quadro
	-Se passou 1 segundo e não houve confirmação:
		-Retransmite o quadro de dados enviado
```

5. Testes

Para testar o funcionamento do emulador, foram realizados vários testes de envio de arquivos de diferentes tamanhos e tipos, com sucesso. O emulador também conseguiu trocar arquivos com o emulador dado para testes.

A troca de dados pode ser observada utilizando o Wireshark, e nele é possível visualizar o envio dos quadros codificados. O envio de um quadro em um dos testes realizado pode ser observado abaixo:



A porta utilizada foi 9091 e o endereço local. É possível ler o quadro enviado na parte inferior da imagem, iniciando com “cc”.

6. Conclusão

A camada de enlace é de grande importância nas redes de computadores, já que liga a camada física com a camada de rede, detectando eventuais erros de transmissão e controle do fluxo de dados. O emulador criado neste trabalho proporcionou um melhor entendimento de seu funcionamento, apesar de funcionar em um nível superior, não realmente lidando com os sinais físicos. É um processo que exige sincronia e que deve se preparar para qualquer tipo de erro. A rede DCCNET proposta pôde ser implementada com êxito, levando em conta todos esses detalhes.

7. Referência

- https://tools.ietf.org/html/rfc1071