Implementação de Pilha de Protocolos

Arthur de Freitas Abeilice[1], **Daniel Silva da Fonseca**[1]
Rodrigo Silva Borges[1], **Taís Rocha Silva**[1]

[1]Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Belo Horizonte – MG – Brasil

{afa7789,daniel0547,rodrigo.sil.borges,silva.taisrocha}@gmail.com

1. Introdução

O propósito deste trabalho foi simular a camada física de um serviço de rede. Para tanto, foi implementada em *shell script* uma aplicação de cliente/servidor que realiza a comunicação entre duas máquinas utilizando o protocolo TCP.

A partir do que foi construído nesta etapa, outras camadas serão criadas e adicionadas à pilha de protocolos para comunicarem entre si. De modo a testar a camada física isoladamente, realizou-se a transmissão de um arquivo existente na máquina cliente para a máquina servidor.

No ato da transmissão em uma camada física, é adicionado um cabeçalho ao pacote, com o endereço MAC tanto de origem quanto de destino. Além disso, foi simulada a ocorrência de colisão na rede. Sendo o caso, o cliente espera um tempo aleatório e tenta a retransmissão.

2. Camada Física

2.1. Solução prática: Netcat

O Netcat é um utilitário capaz de ler e escrever dados em conexões de rede utilizando os protocolos TCP ou UDP. Segundo definição do próprio desenvolvedor [1], o comando foi projetado para ser uma ferramenta "*back-end*" confiável, que pode ser usada direta e facilmente por outros programas e scripts. Em fóruns de discussão sobre programação e tecnologia, o Netcat é conhecido como o "canivete suíço das conexões TCP/IP", expressão que denota o quanto esta ferramenta é considerada versátil como solução em aplicações de rede.

Em uma aplicação que utiliza o Netcat, a transmissão de mensagens segue os princípios da comunicação entre processos do paradigma cliente-servidor. Sendo assim, é necessário configurar um *host* servidor (processo destino) para "escutar" uma porta específica, enquanto o *host* cliente (processo origem) realiza a transmissão da mensagem através da conexão com a porta especificada pelo servidor.

Para executar o Netcat, utiliza-se o comando `nc` acompanhado pelas diretivas adequadas a cada aplicação. As principais opções empregadas no comando Netcat podem ser visualizadas a seguir [2]:

[1]Disponível em `http://netcat.sourceforge.net/`

[2]Para detalhamento do Netcat, com descrição de todas as opções disponíveis, consultar o comando `nc -help`.

- `-e`: Permite especificar o nome do programa a ser executado após uma conexão.
- `-l`: Habilita o módulo "escuta" do Netcat, ou seja, a ferramenta passará a escutar as conexões ocorridas na porta especificada.
- `-p`: Especifica qual porta será utilizada pelo Netcat ao fechar uma conexão.
- `-u`: Habilita o uso do protocolo UDP, por padrão o Netcat trabalha utilizando o padrão TCP/IP de conexão.
- `-v`: Permitirá restringir a quantidade de mensagens mostrada em tela, na prática poderemos criar conexões com maior nível de detalhamento no retorno das informações.
- `-w`: Permite limitar o tempo de uma conexão com um valor numérico passado em segundos.

A sintaxe fundamental para execução de uma requisição pelo Netcat cliente é apresentada pelo comando a seguir, sendo possível estabelecer a conexão com uma porta arbitrária **`[port]`** a partir do **`[TargetIPaddr]`**, que é simplesmente o endereço IP ou o nome de domínio do destinatário.

```
$ nc [TargetIPaddr] [port]
```

A sintaxe fundamental para execução de uma requisição pelo Netcat servidor é apresentada pelo comando a seguir, em que **`[LocalPort]`** representa uma porta local arbitrária onde o Netcat *listener* será criado.

```
$ nc -l -p [LocalPort]
```

2.2. Implementação

A implementação da camada física foi realizada utilizando a linguagem de script *shell*. Foram escritos dois códigos, um dedicado a descrever as funcionalidades de um processo cliente e o outro destinado a cumprir com os propósitos de um processo servidor, ambos considerando o transporte orientado para conexão TCP (*Transmission Control Protocol*). As seções que seguem buscam apresentar um detalhamento de cada um dos códigos.

2.2.1. Cliente TCP

O código referente ao cliente TCP da camada física implementada compreende algumas etapas, listadas como:

1. Carregar o *payload* recebido da camada superior, incluindo a conversão de hexadecimal para string.
2. Preencher o cabeçalho do quadro com os endereços MAC dos *hosts* origem e destino.
3. Montar a PDU com o cabeçalho e *payload*.
4. Enviar o quandro para o endereço e porta de destino especificados, incluindo a conversão da PDU para hexadecimal.
5. Tratar a ocorrência de colisões.

Listing 1. Etapa 1

```
payload=$(cat $2 | xxd -r -p);
```

Listing 2. Etapa 2

```
mac_orig=$(ifconfig | grep -o -m 1 "..:..:..:..:..:..");
mac_dest=$(arp $1 | grep -o -m 1 "..:..:..:..:..:..");
head=$(echo "MAC ORIGEM: $mac_orig MAC DESTINO: $mac_dest");
```

Listing 3. Etapa 3

```
frame=$(echo "$head $payload");
```

Listing 4. Etapa 4

```
echo -n $frame | xxd -c 256 -ps | nc $1 $PORT;
```

Listing 5. Etapa 5

```
#sorteia um numero aleatorio de 0 a 10 (inclusive)
DIV=$((10+1));
R=$(($RANDOM%$DIV));

#enquanto o numero sorteado for maior ou igual a "x",
#considera uma colisao
while [ $R -ge 7 ]
do
        echo "colision detected...";

        DIV=$((10+1));
        R=$(($RANDOM%$DIV));

        echo "waiting for $R seconds...";

        #aguarda de 0 a 10 segundos para reenviar o quadro
        sleep $R;

        echo -n $frame | xxd -c 256 -ps | nc $1 $PORT;

        DIV=$((10+1));
        R=$(($RANDOM%$DIV));
done
```

Destaca-se que a decisão de implementar a conversão do conteúdo das PDUs para hexadecimal foi meramente prática, tendo-se em vista a correspondência direta entre os sistemas de numeração de base 2 e de base 16, respectivamente binário e hexadecimal. Alguns programas empregados na análise do tráfego de pacotes de rede, tais como o Wireshark [3], apresenta todo o conteúdo das mensagens em hexadecimal.

2.2.2. Servidor TCP

O código referente ao servidor TCP da camada física implementada compreende algumas etapas, listadas como:

[3]Disponível em `https://www.wireshark.org/`

1. Criar um processo em *background* (coprocesso) para escutar o canal de comunicação em determinada porta.
2. Redirecionar o quadro recebido para o arquivo descritor.
3. Imprimir o número de bytes do quadro recebido e guardar a PDU recebida em um arquivo.
4. Redirecionar a resposta para o canal de comunicação.

Listing 6. Etapa 1

```
coproc nc -l $PORT;
```

Listing 7. Etapa 2

```
while read -r cmd
do
[...]
done >&${COPROC[1]}
```

Listing 8. Etapa 3

```
echo $(echo $cmd | wc -c)
echo $cmd > $1
```

Listing 9. Etapa 4

```
while read -r cmd
do
[...]
done <&${COPROC[0]}
```

3. Exemplo de execução

A máquina servidor executa a função *server.sh* definindo um arquivo para onde direcionar o conteúdo recebido do cliente. Uma opção de chamada seria:

```
./server.sh recebidos.txt
```

A definição da porta a ser escutada encontra-se na própria função. A máquina cliente, então, executa a função *client.sh* explicitando o endereço IP do servidor e o arquivo que pretende transmitir, conforme:

```
./client.sh 10.0.125.146 mensagem.txt
```

Novamente, a porta do serviço já está definida na função chamada. A transmissão é realizada e, depois de concluída, o arquivo *recebidos.txt* na máquina servidor tem uma cópia do conteúdo do arquivo *mensagem.txt* da máquina cliente.

Referências