Lista 8 - Unidade III - Camada de Transporte

Nome: Victor Ferraz de Moraes
Matrícula: 802371

Exercício 1:

O endereço de rede (IP), utilizado na camada de rede, identifica unicamente um dispositivo na rede, ou seja, para qual máquina os dados devem ser enviados. Já a porta, usada na camada de transporte (TCP/UDP), identifica qual processo ou aplicação dentro daquele dispositivo deve receber os dados. Portanto, o IP leva o pacote até o destino correto na rede, e a porta garante que ele chegue ao aplicativo certo dentro do dispositivo. A seleção da porta só ocorre nos dispositivos finais (hosts), pois são eles que executam os processos.

Exercício 2:

A camada de transporte dá sentido à pilha de protocolos porque ela conecta diretamente as aplicações dos dispositivos finais, permitindo que dados sejam trocados de processo para processo, e não apenas de máquina para máquina. Enquanto as camadas inferiores (como enlace e rede) cuidam de entregar os dados fisicamente e logicamente entre dispositivos, é a camada de transporte que garante que os dados cheguem corretamente ao programa certo, e, se necessário, na ordem certa, sem perdas ou duplicações (como no caso do TCP).

Exercício 3:

Uma aplicação one-shot é aquela que realiza uma única troca de mensagens, como um envio seguido de uma resposta simples. O protocolo TCP pode ser usado nesse tipo de aplicação, mas não é o mais eficiente, pois exige o estabelecimento e encerramento de uma conexão (com o three-way handshake e o four-way termination), gerando overhead desnecessário para algo tão breve. Já o protocolo UDP é mais indicado, pois é leve, sem conexão, e permite enviar e receber dados com menor latência, ideal para esse tipo de comunicação.

Exercício 4:

As primitivas são executadas na seguinte ordem:

Do lado do Servidor:
Socket: Cria um ponto final de comunicação, especificando o formato do endereço, tipo de serviço e protocolo.
Bind: Atribui um endereço local (IP e porta) ao soquete.
Listen: Anuncia disposição para aceitar conexões e aloca espaço para a fila de chamadas recebidas.
Close: É fechada a conexão com o Cliente.

Accept: Bloqueia o responsável pela chamada até que o servidor receba uma tentativa de conexão. Quando o servidor recebe uma conexão, ele pode desviá-la para outro processo (thread) e iniciar a espera por outra conexão.

Do lado do Cliente:
Socket: Como no servidor, esta primitiva cria um ponto final de comunicação, fazendo as devidas especificações.
Connect: Tenta estabelecer uma conexão com o servidor.
Send/Receive: As partes devem ter uma política para definir de quem é a vez de enviar/receber dados.
Close: É fechada a conexão com o Servidor.

Exercício 5:

Após o estabelecimento da conexão pelo three-way handshake, o emissor (host A) envia dados ao receptor (host B) com um número de sequência que representa o número do primeiro byte daquele segmento. O receptor (host B), ao receber o segmento, envia um pacote com a flag ACK ativada e um número de reconhecimento (ACK Number) que indica o próximo byte que ele espera receber. Isso confirma o recebimento dos dados anteriores. Por exemplo, se o host A envia 1000 bytes começando com o SEQ = 1, o host B, ao receber corretamente, responderá com ACK = 1001, indicando que espera a partir dali. A transferência continua dessa forma, com ambos os lados acompanhando e confirmando os dados recebidos e enviando novos segmentos com seus próprios SEQ e ACK.

Exercício 6:

As flags FIN, ACK e SYN são muito importantes no estabelecimento e término das conexões TCP, pois elas são as principais confirmações utilizadas no three-way handshake e no four-way termination.

Para o three-way handshake, o host A encaminha um pacote com a flag SYN = 1, enquanto as outras flags permanecem com 0 para o host B. Após isso, caso o host B receba a mensagem, ele encaminha para o host A outra mensagem contendo as flags SYN e ACK iguais a 1, indicando que ele permite a conexão e que recebeu a mensagem do host A. Por fim, o host A envia mais uma mensagem com a flag ACK = 1 e as outras iguais a 0, reconhecendo que recebeu a resposta do host B.

Um processo como esse no término também, com a diferença que está sinalizado somente nas flags FIN e ACK:

1) Um dos hosts envia uma mensagem com a flag FIN = 1, indicando que deseja finalizar a conexão por sua parte
2) O outro host sinaliza que recebeu essa mensagem com a flag ACK = 1.

3) Após um tempo, quando o segundo host não realizar mais transferências, ele também realiza os passos 1 e 2 e, portanto, são realizadas 4 confirmações para o término da conexão no total.

Exercício 7:

O controle de fluxo no TCP é um mecanismo que garante que um remetente não envie mais dados do que o receptor é capaz de processar e armazenar. Isso evita que o receptor fique sobrecarregado caso tenha um buffer pequeno ou esteja processando lentamente os dados recebidos. Esse controle é feito por meio do campo Window Size no cabeçalho dos segmentos TCP. Esse campo indica a quantidade de bytes que o receptor ainda consegue receber no momento sem risco de perda de dados — ou seja, o tamanho da "janela de recepção" disponível. Quando o receptor envia um ACK ao remetente, ele inclui o valor atualizado da sua janela (Window Size). Esse valor informa ao emissor quantos bytes ele ainda pode enviar antes de precisar aguardar um novo ACK. Se a janela ficar cheia (valor igual a 0), o emissor pausa o envio até que receba um Window Update, com um novo valor de janela.

Exercício 8:

Na figura A, está sendo representado o controle de fluxo, em que a torneira representa a fonte de dados (emissor). A rede de transmissão leva os dados até um receptor de pequena capacidade, como mostra o copo pequeno. A vazão da torneira (transmissão) é ajustada para não sobrecarregar o receptor.
Já na figura B, está sendo representado o controle de congestionamento, em que a rede de transmissão está sobrecarregada, mesmo que o receptor tenha grande capacidade de recepção (o balde grande). O gargalo está dentro da rede, com acúmulo de dados causando congestionamento interno.

Exercício 9:

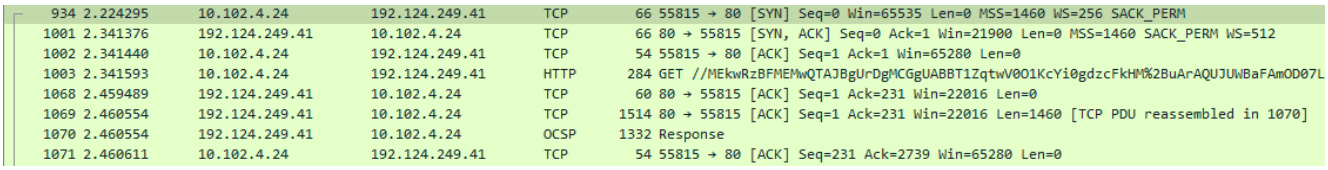
O controle de congestionamento é um mecanismo do TCP que visa evitar sobrecarregar a rede com dados demais. Mesmo que o receptor consiga receber mais dados, a rede pode estar congestionada. Para evitar isso, podemos utilizar o algoritmo de iniciação lenta da seguinte forma:

1) O TCP manda dados através da rede do tamanho do segmento
2) A cada ACK recebido do outro host, o TCP dobra o tamanho do segmento a ser mandado pela rede até que tenha um timeout ou que atinja o limiar desejado.
3) Ao atingir o limiar, o TCP passa a aumentar o tamanho do segmento a ser enviado de forma linear.
4) Quando um timeout ocorrer, o tamanho da janela que será enviada volta ser do segmento inicial e o limiar é atualizado para ser metade do valor ao qual ocorreu o timeout.
5) Os passos 1 a 4 ocorrem até o fim da conexão.

Exercício 10:

A figura demonstra de forma visual como funciona o algoritmo de Iniciação lenta utilizado para lidar com o controle de congestionamento. Nela, podemos ver que a Janela de Congestionamento começa com o valor de 1 segmento e é aumentado de forma exponencial até atingir seu limiar em 32. Após esse crescimento exponencial, a janela passa crescer de forma linear até que esteja no valor de 40, onde ocorre um timeout.
Assim que ocorre o timeout, o valor da janela de congestionamento passa a valer 1 novamente e seu limiar é atualizado para valer metade da última janela (40/2 = 20).
O valor da janela de congestionamento passa a crescer de forma exponencial mais uma vez até chegar em seu novo limiar (20), onde passa a crescer de forma linear.

Exercício 11:



Primeiro precisamos definir os conceitos de número ACK e SEQ:

SEQ (Sequence Number): identifica o número do primeiro byte de dados naquele segmento TCP.
ACK (Acknowledgment Number): indica o próximo byte que o emissor espera receber do outro lado (ou seja, já recebeu tudo até o ACK−1).

A partir disso, podemos demonstrar como os pacotes estão sendo enviados e recebidos de cada parte. Em que:

Cliente ao Servidor: SEQ=1, ACK=231
O Cliente começa a enviar dados a partir do byte 1 e já recebeu até o byte 230 do servidor, esperando o byte 231.

Servidor ao Cliente: SEQ=1, ACK=231
Começa também em byte 1, e confirma ter recebido tudo até o byte 230 do cliente.
Depois, quando o servidor efetivamente envia dados (segmento de 1 460 bytes), o cliente responde com ACK=2739, indicando “já recebi todos os bytes do servidor até o 2738, meu próximo esperado é o 2739”.