PROJETO 02
REDES DE COMPUTADORES
TURMA-B 2021

Professora Profa. Priscila Solís Barreto

Protocolo de transferência de dados confiável nas versões:
a) Stop and Wait b) Go-Back-N

Victor Candeira: 170157636

Jefte Batista: 180057570

VItor Araruna: 202060980

Link para códigos no git: https://github.com/vscandeira/redes_trabalho2
Link para video: https://unbbr-my.sharepoint.com/:v:/g/personal/170157636_aluno_unb_br/EUZucgQs2KREqM1le58RHpcBkOczQ0YJ7LC-TBZmJKcuXw?e=1pm0JB

A partir dos estudos realizados na disciplina de Redes de Computadores, deve-se ter o entendimento sobre trabalhar com a ideia de camadas, uma vez que sistemas muito complexos buscam essa estrutura para se ter uma organização melhor e compreensível. Das sete camadas estipuladas pelo modelo OSI, a estrutura TCP/IP apresenta quatro delas: camada de aplicação, camada de transporte, camada de rede e camada de enlace.Com os ensinamentos sobre tais camadas da estrutura TCP/IP, sabe-se que cada uma tem seu objetivo e sua tarefa a ser realizada, assim, a execução de todas elas mantém a integridade dos dados que passam pela rede.

A camada de transporte tem como tarefa receber os dados vindos da camada de aplicação, dividi-los em pacotes menores, enviando-os à camada de rede e realizando uma comunicação lógica entre aplicações que estão funcionando em hosts diferentes. Uma vez que a camada de transporte realiza demultiplexação para entregar as mensagens, a camada de rede apenas direciona a informação (entre hosts).

Notam-se dois principais protocolos nesta camada, são eles: **TCP**: um protocolo confiável e entrega os segmentos de forma ordenada e **UDP**: um protocolo não confiável sujeito a perda de segmentos sem aviso ao receptor, utilizado em casos como em alguns casos de transmissão multimídia, DNS, entre outros que podem perder segmentos ou receber entregas fora de ordem à aplicação.

Este trabalho tem como objetivo desenvolver um código na camada de transporte utilizando um protocolo de transferência confiável. A implementação conta com duas versões de protocolos:

1) STOP AND WAIT : protocolo no qual o emissor envia um pacote e espera uma resposta do receptor para poder enviar o seguinte, ou seja, possui apenas uma janela de transmissão de tamanho igual a 1 e dois estados: 0 e 1. Caso o pacote tenha sido recebido com um NACK, então este contém erros e será retransmitido.

2) GO-BACK-N: protocolo semelhante com o citado acima, porém ele é capaz de transmitir múltiplos pacotes sem esperar por um reconhecimento, ou seja, com uma janela de transmissão de tamanho superior a 1. Possui um buffer de pacotes com função de armazenar os pacotes enviados, mas ainda não confirmados(ACK). Quando o buffer está cheio, só são aceitos novos pacotes após o recebimento de ACK do primeiro pacote do buffer, nesse momento a

janela de transmissão caminha uma posição, permitindo o envio de mais um pacote.

Para uma melhor visualização, o link a seguir exemplifica bem o caminho dos pactos para cada versão citada acima.
https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/

Para compilar o código, foi necessário comentar a linha que chamava a função exit() dentro de init(), pois a primeira não foi definida no programa. Também foi necessário comentar as linhas de código com "`char* malloc()`" .

Além disso, optou-se por fazer o uso de variáveis globais de forma a não alterar os parâmetros previamente definidos para cada função.

Parece lógico que no recebimento de um NACK, o emissor reenvie o pacote e inicie um novo timer. Porém, identificamos que a máquina de estados RDT 3.0 SENDER da vídeo aula orienta o reenvio de pacote apenas no caso de Timeout.

É importante ressaltar que nas máquinas de estados do livro texto e das aulas, ao receber um NACK não deve ser realizada nenhuma tarefa e o pacote só seria reenviado no timeout. Isso pareceu contraintuitivo, pois o que pareceu mais lógico seria reenviar o pacote e reiniciar o timer. Em conversa com o monitor, foi explicado que esse procedimento visa evitar inundar a rede com reenvios. Dessa forma, esse procedimento do protocolo seria, também, uma forma de buscar garantir a saúde da infraestrutura de rede como um todo. Por esse motivo, não é efetuada nenhuma ação no caso de NACK.

Rotinas desenvolvidas (stop and wait):

A_init():

- Rotina que é chamada uma só vez antes de qualquer outra rotina de A. Define o estado inicial de a "seqA" = 0.

B_init():

- Rotina que é chamada uma só vez antes de qualquer outra rotina de B. Define o estado inicial "seqB" = 0.

A_output(message):

- Essa rotina simula uma "message" que está chegando da camada de aplicação (Layer5) para a camada de transporte. Ao chegar tal mensagem na

entidade emissora A, esta é responsável por construir o pacote que será levado à camada de rede, layer3.

B_input(pacote):

- Ao passar pela camada de aplicação, o pacote gerado na rotina A_output é recebido nessa função. Caso o número de sequência “seqB” seja o mesmo que o do pacote recebido, ocorre a verificação do checksum. Nessa etapa é construído o pacote ACK NACK “packet_asw” (com seqNum=seqB) que será encaminhado para A.

  Se o checksum passar no teste, então o “packet_asw” é ACK (acknum = 1), o payload é entregue para a camada de aplicação no lado do receptor e é atualizado “seqB”. Caso o checksum não passe no teste, então “packet_asw” é NACK (acknum = 0) e é informado ao lado A que o pacote deve ser retransmitido.

  Caso o estado de B (seqB) seja diferente do estado do pacote recebido (seqNum), então o ACK correspondente ao estado anterior foi perdido/corrompido e deve ser enviado novamente.

A_input(packet_asw):

- O pacote ACK NACK enviado por B chega ao lado A. É verificado o checksum do pacote, se este é um ACK. No caso de NACK ou de ACK NACK corrompido, não realiza nenhuma tarefa.

A_timerinterrupt()

- Essa rotina é chamada quando o temporizador de A expirar. Ela é responsável por reenviar o pacote para a camada de transporte pela função A_OUTPUT, com o auxílio da variável “curr_msg” que sempre guarda os dados de pacotes que precisam ser reenviados futuramente, para não perdermos suas informações.

Rotinas desenvolvidas (Go-Back-N):

A_init():

- Agora além das funcionalidades do “stop and wait”, a rotina inicializa o vetor “`curr_msgs`” que representa o buffer do protocolo.

B_init():

- As mesmas funcionalidades do STOP AND WAIT.

A_output(message):

- É verificado se o estado de A (saqA), que agora representa o sequencial do último pacote enviado, é menor que a posição da janela (baseA) mais o tamanho da janela (WINDOWSIZE). Somente nesse caso é enviado um pacote e armazenada a mensagem no buffer. Caso contrário, a mensagem é ignorada.

B_input(pacote):

- As mesmas funcionalidades do STOP AND WAIT, com a ressalva de que agora o estado (seqB) é representado por um número inteiro que é incrementado a cada pacote confirmado.

A_input(packet_asw):

- Agora, além da verificação de integridade do pacote ACK NACK recebido, a posição da janela (baseA) é definida como o sequencial do pacote recebido (seqNum) + 1, apenas no caso de ACK. Isso porque B aguarda o envio do pacote de sequencial seguinte ao do último pacote confirmado (ACK) e, portanto, em caso de perda de pacotes, reenviará sempre o ACK correspondente a esse pacote. Nesse caso, o base pode ser definido como seqNum + 1, pois significa que a janela não avança. E no caso de não haver perda de pacotes de A para B, o baseA avançará uma posição, permitindo o envio de novos pacotes.
  No caso de NACK, não realiza nenhuma tarefa.

A_timerinterrupt()

- Agora, no evento timeout, serão reenviados todos os pacotes da janela.