



Universidade de Brasília
Departamento de Ciências da Computação
Engenharia de Computação
Teleinformática e Redes II

TR2

João Pedro Gomes Covalski Marin Antonow, 221006351

Professor: Jacir Luiz Bordim

Brasília
2023

1. Objetivo

1. Trabalhar com uma arquitetura de rede em camadas e implementar funções que permitam uma melhor utilização do enlace utilizando técnicas de pipelining vistas no capítulo 4 (Camada de Transporte) do livro.
2. Seu trabalho é implementar o `rdt_4_0`, estendendo/modificando o código inicial fornecido (stop-and-wait) para que múltiplos pacotes possam fluir entre cliente e servidor.
3. Implementações possíveis:
 - (a) Go-back-N (mais simples)
 - (b) Selective Repeat (mais complexo, e por isso tem um bônus extra na nota final para o grupo que implementar de forma correta).
4. Após a execução/simulação, seu código deve fornecer as seguintes estatísticas:
 - (a) Vazão (camada de rede, incluindo cabeçalhos)
 - (b) Goodput (vazão na camada de aplicação)
 - (c) Total de pacotes transmitidos
 - (d) Total de retransmissões (para cada tipo de pacote utilizado)
 - (e) Total de pacotes corrompidos (para cada tipo de pacote utilizado)
 - (f) Tempo de simulação (tempo desde o início do envio até o último pacote enviado)
5. O código deve permitir o envio de múltiplas mensagens entre o cliente e servidor. O número de mensagens deve ser definido como argumento de linha do cliente.

2. Funcionamento

1. O código fornecido implementa o `rdt_3_0` de forma similar ao que vimos na FSM do livro, com pequenas alterações. Basicamente:
 - (a) O cliente envia mensagens para o servidor
 - (b) O servidor converte as mensagens recebidas para caixa alta e as transmite de volta ao cliente.
 - (c) Cliente e servidor enviam mensagens um para o outro por meio da camada de transporte fornecida por uma implementação RDT. O RTD utiliza as funções `rdt_3_0_send` e `rdt_3_0_receive`.
 - (d) Obs 1: O RDT possui a opção de debug que poderá ser utilizada para mostrar em tela as mensagens e estados do cliente e servidor.
2. O protocolo RDT utiliza as funções `udt_send` e `udt_receive` fornecidos pela camada de rede, implementados no protocolo `network.py`. Estas funções permitem transferir bytes entre as máquinas cliente e servidor
3. Obs 2: A camada de rede pode corromper pacotes, perder pacotes ou mesmo reordenar estes pacotes, a depender da configuração dos parâmetros da camada de rede controlados pelas variáveis abaixo listadas.
 - (a) `prob_pkt_loss = 0`
 - (b) `prob_byte_corr = 0`
 - (c) `prob_pkt_reorder = 0`
4. Obs 3: Talvez seja necessário modificar/estender a classe `Packet` para transmitir as informações necessárias para que essas funções funcionem corretamente. Teste seu código para garantir que estas opções funcionem corretamente no seu código.

3. Conceitos

3.1. Go-Back-N ou Selective Repeat?

Vamos encerrar nosso estudo do mecanismo de recuperação de erros do TCP considerando a seguinte pergunta: o TCP é um protocolo GBN ou SR? Lembre-se de que, no TCP, os reconhecimentos são cumulativos e segmentos recebidos de modo correto, mas fora da ordem, não são reconhecidos (ACK) individualmente pelo destinatário. Em consequência, como mostrou a Figura 3.33 (veja também a Figura 3.19), o TCP remetente precisa tão somente lembrar o menor número de sequência de um byte transmitido, porém não reconhecido (SendBase) e o número de sequência do byte seguinte a ser enviado (NextSeqNum). Nesse sentido, o TCP se parece muito com um protocolo ao estilo do GBN. Porém, há algumas diferenças surpreendentes entre o TCP e o GBN. Muitas execuções do TCP armazenarão segmentos recebidos corretamente, mas fora da ordem [Stevens, 1994]. Considere também o que acontece quando o remetente envia uma sequência de segmentos 1, 2, ..., N e todos os segmentos chegam ao destinatário na ordem e sem erro. Além disso, suponha que o reconhecimento para o pacote $n + 1$ se perca, mas que os $N - 1$ reconhecimentos restantes cheguem ao remetente antes do esgotamento de suas respectivas temporizações. Nesse exemplo, o GBN retransmitiria não só o pacote n , mas também todos os subseqüentes $n + 1$, $n + 2$, ..., N . O TCP, por outro lado, retransmitiria no máximo um segmento, a saber, n . E mais, o TCP nem ao menos retransmitiria o segmento n se o reconhecimento para $n + 1$ chegasse antes do final da temporização para o segmento n . Uma modificação proposta para o TCP, denominada reconhecimento seletivo [RFC 2018], permite que um destinatário TCP reconheça seletivamente segmentos fora de ordem, em vez de apenas reconhecer de modo cumulativo o último segmento recebido corretamente e na ordem. Quando combinado com retransmissão seletiva — isto é, saltar a retransmissão de segmentos que já foram reconhecidos de modo seletivo pelo destinatário —, o TCP se parece muito com nosso protocolo SR genérico. Assim, o mecanismo de recuperação de erros do TCP talvez seja mais bem caracterizado como um híbrido dos protocolos GBN e SR.

4. Projeto

A nossa implementação consistiu em modificar o código-base proposto e, além de implementar o mecanismo de selective repeat para a transmissão de dados, elaborar um gráfico automático através dos recursos da biblioteca *matplotlib*, comparando com os resultados vistos no wire-shark.

4.1. Resultados:

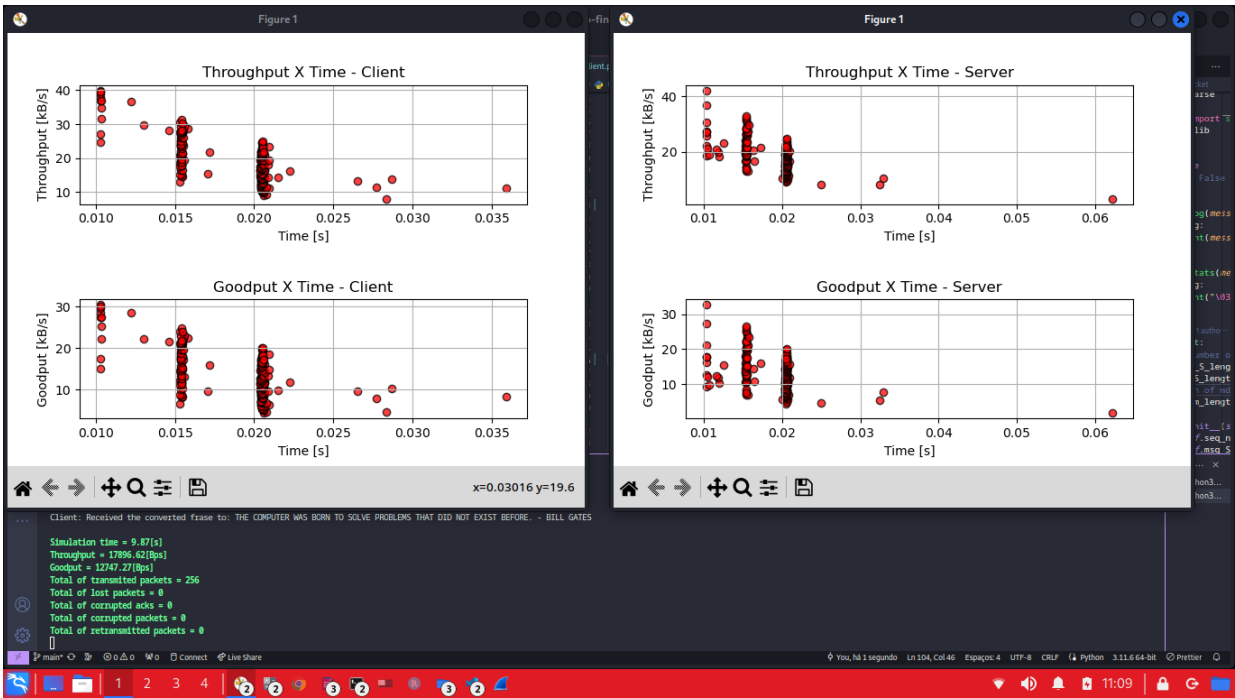


Figura 1. Simulação sem erro

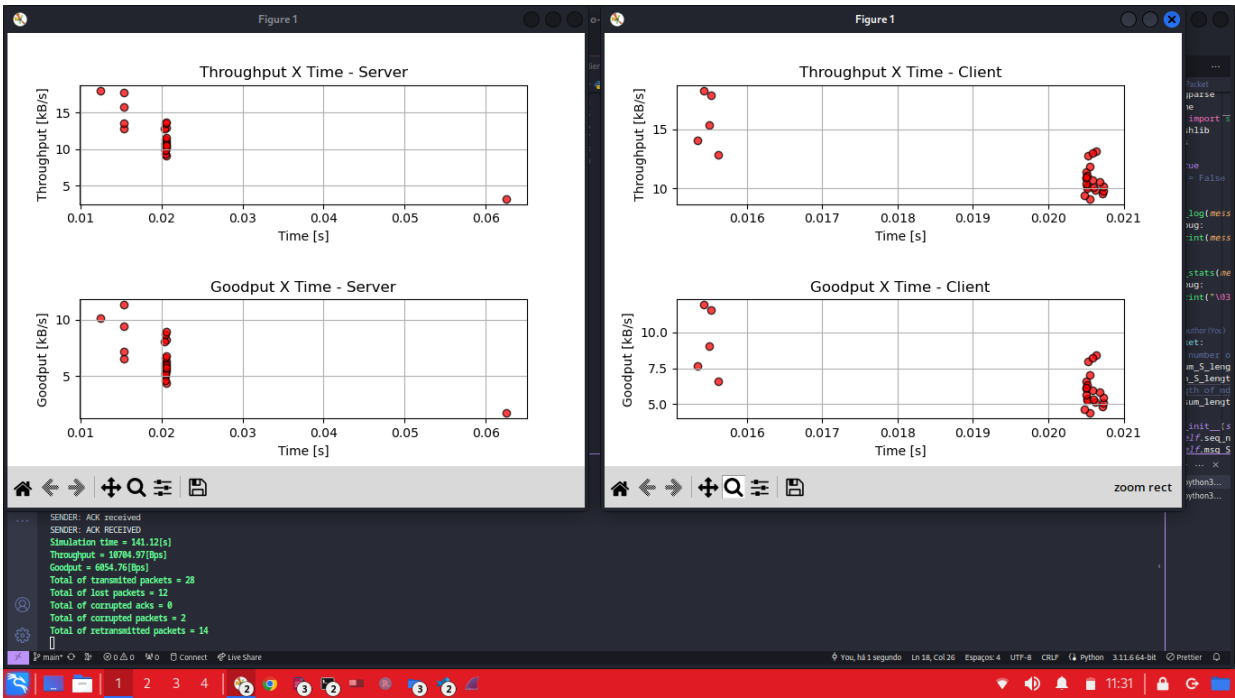


Figura 2. Simulação com erro

5. Client-Server-Interaction - Stop and Wait

Network: role **is** client

Client asking to change case: The art of debugging **is** figuring out what you really told your program to do rather than what you thought you told it to do. -- Andrew Singer

SENDER:

00000000530000000000dee1d923c35ace28544ef49e879ff31710000000193000000000101

ART OF DEBUGGING IS FIGURING OUT WHAT YOU REALLY TOLD YOUR PROGRAM TO DO RATHER THAN WHAT YOU THOUGHT YOU TOLD IT TO DO. -- ANDREW SINGER

SENDER: Received ACK, move on to **next**.

SENDER: Incrementing seq_num **from** 0 to 1

RECEIVER: Received new. Send ACK **and** increment seq.

RECEIVER: Incrementing seq_num **from** 1 to 2

Client: Received the converted frase to: THE ART OF DEBUGGING IS FIGURING OUT WHAT YOU REALLY TOLD YOUR PROGRAM TO DO RATHER THAN WHAT YOU THOUGHT YOU TOLD IT TO DO. -- ANDREW SINGER

Client asking to change case: The good news about computers **is** that they do what you tell them to do. The bad news **is** that they do what you tell them to do. -- Ted Nelson

SENDER: 000000005300000000028527f664eb2892eac11172a94e71d2991

SENDER: Received ACK, move on to **next**.

SENDER: Incrementing seq_num **from** 2 to 3

RECEIVER: Received new. Send ACK **and** increment seq.

RECEIVER: Incrementing seq_num **from** 3 to 4

Client: Received the converted frase to: THE GOOD NEWS ABOUT COMPUTERS IS THAT THEY DO WHAT YOU TELL THEM TO DO. THE BAD NEWS IS THAT THEY DO WHAT YOU TELL THEM TO DO. -- TED NELSON

Client asking to change case: It **is** hardware that makes a machine fast. It **is** software that makes a fast machine slow. -- Craig Bruce

SENDER: 000000005300000000049bd3fdd84fb4ab8212beafb637a8ffc21

SENDER: Received ACK, move on to **next**.

SENDER: Incrementing seq_num **from** 4 to 5

RECEIVER: Received new. Send ACK **and** increment seq.

RECEIVER: Incrementing seq_num **from** 5 to 6

Client: Received the converted frase to: IT IS HARDWARE THAT MAKES A MACHINE FAST. IT IS SOFTWARE THAT MAKES A FAST MACHINE SLOW. -- CRAIG BRUCE

Client asking to change case: The art of debugging **is** figuring out what you really told your program to do rather than what you thought you told it to do. -- Andrew Singer

SENDER: 000000005300000000067df0e21232975b1e5ab8851cc513d8e91

SENDER: Received ACK, move on to **next**.

SENDER: Incrementing seq_num **from** 6 to 7

RECEIVER: Received new. Send ACK and increment seq.
RECEIVER: Incrementing seq_num from 7 to 8
Client: Received the converted frase to: THE ART OF DEBUGGING
IS FIGURING OUT WHAT YOU REALLY TOLD YOUR PROGRAM TO DO
RATHER THAN WHAT YOU THOUGHT YOU TOLD IT TO DO. -- ANDREW
SINGER

Client asking to change case: The computer was born to solve
problems that did not exist before. - Bill Gates
SENDER: 0000000053000000000858e02b3236c1e3f820981a9659a591711
SENDER: Received ACK, move on to next.
SENDER: Incrementing seq_num from 8 to 9
RECEIVER: Received new. Send ACK and increment seq.
RECEIVER: Incrementing seq_num from 9 to 10
Client: Received the converted frase to: THE COMPUTER WAS BORN
TO SOLVE PROBLEMS THAT DID NOT EXIST BEFORE. - BILL GATES

Connection ended.

6. Selective Repeat Simulation

Link: <https://nikhilsahu.me/Sliding-window-simulator/>

Referências