# PTC 3360

## 2.2 Camada de transporte: princípios da transferência confiável de dados – Parte III

(Kurose, Seção 3.5)

Agosto 2025

# Capítulo 2 - Conteúdo

2.1 A camada de aplicação

2.2 A camada de transporte: princípios da transferência confiável de dados

2.3 A camada de rede

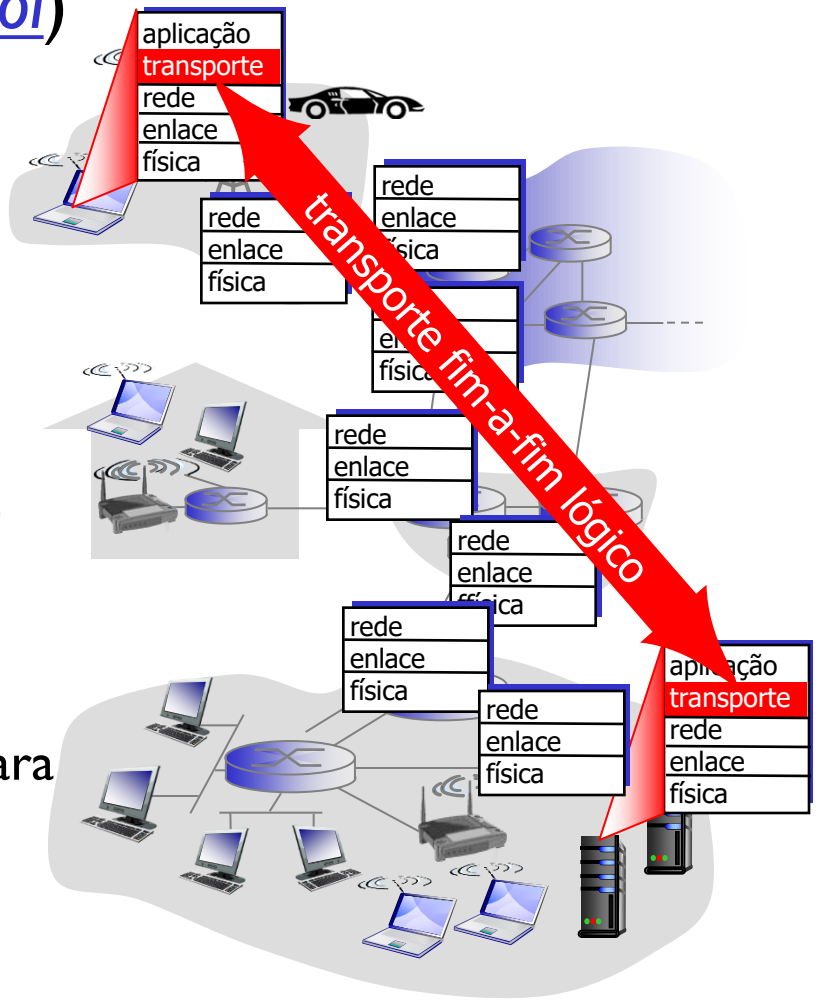
## Resumo: Mecanismos para transferência de dados confiável

- *Códigos para detecção de erros*
- *Acknowledgement* - Receptor informa que pacote ou conjunto de pacotes foi recebido corretamente. Pode ser individual ou acumulativo.
- *Negative acknowledgement* - Destinatário informa que pacote não foi recebido adequadamente.
- *Números sequenciais* – Detectar pacotes perdidos ou recebidos em duplicata.
- *Timer* - Usado para detectar perda de pacotes.
- *Janelas, paralelismo (pipelining)* – Permite *N* pacotes transmitidos mais ainda não *reconhecidos*, melhorando utilização do transmissor em relação ao *stop-and-wait*.

- *Quais são usados pelo TCP?? Vamos ver na sequência...*

# Protocolos da camada de transporte na Internet

- TCP (*Transmission Control Protocol*)
  - Versão atual: RFC 9293 (1980-2022)
  - entrega confiável dos pacotes
  - controle de congestionamento
  - controle de fluxo
  - *setup* de conexão
- UDP (*User Datagram Protocol*)
  - Versão original e ainda atual: RFC 768 (1980)
  - entrega não confiável dos pacotes
  - usuário cria quase diretamente um datagrama (pacote da camada de rede)
  - apenas recepção e entrega de pacotes para os aplicativos e detecção de erro
- Serviços não disponíveis em nenhum dos dois:
  - garantias de latência
  - garantias de capacidade




Obs: 3ª opção, QUIC, cada vez mais presente.

# TCP: Visão geral

- Ideia proposta por Cerf & Kahn em 1974 – IEEE Trans. on Communications Tech.
  - "Os pais da Internet"
- Ponto a ponto:
  - 1 transmissor, 1 receptor
- Fluxo de bytes confiável e em ordem
- Usa paralelismo (*pipelined*):
  - Controle de fluxo e congestionamento do TCP regulam comprimento da janela *N*
- Dados *full duplex*:
  - Fluxo de dados bidirecional na mesma conexão
  - ACK pode ser enviado "de carona" no próximo pacote de dados
- Orientado a conexão:
  - *Handshaking* (troca de mensagens de controle) inicializa estado do remetente e destinatário antes da troca de dados (variáveis, *buffers*)
- Fluxo controlado:
  - Remetente não sobrecarrega destinatário

# Transferência de dados confiável no TCP

- TCP cria transmissão de dados confiável sobre serviço não confiável do IP
  - Segmentos paralelizados (*pipelined*)
  - *ACKs* acumulativos *(parece GBN)*
  - *Único* temporizador de retransmissão *(parece GBN)*
- Transferência confiável do TCP é uma de suas principais características e algoritmo é bastante complicado, resultado de mais de 45 anos de evolução.
- Aqui vamos considerar apenas seus aspectos mais fundamentais.

# Eventos no transmissor TCP simplificado

*Dados recebidos da aplicação:*

- Cria segmento com número sequencial `seq`
- `seq` é número na cadeia de bytes do primeiro byte de dado no segmento (não número do pacote!)
- Inicializa temporizador se já não está rodando
  - Único temporizador para o segmento mais antigo sem ACK *(parece GBN)*
  - Intervalo de expiração: **`TimeOutInterval`**

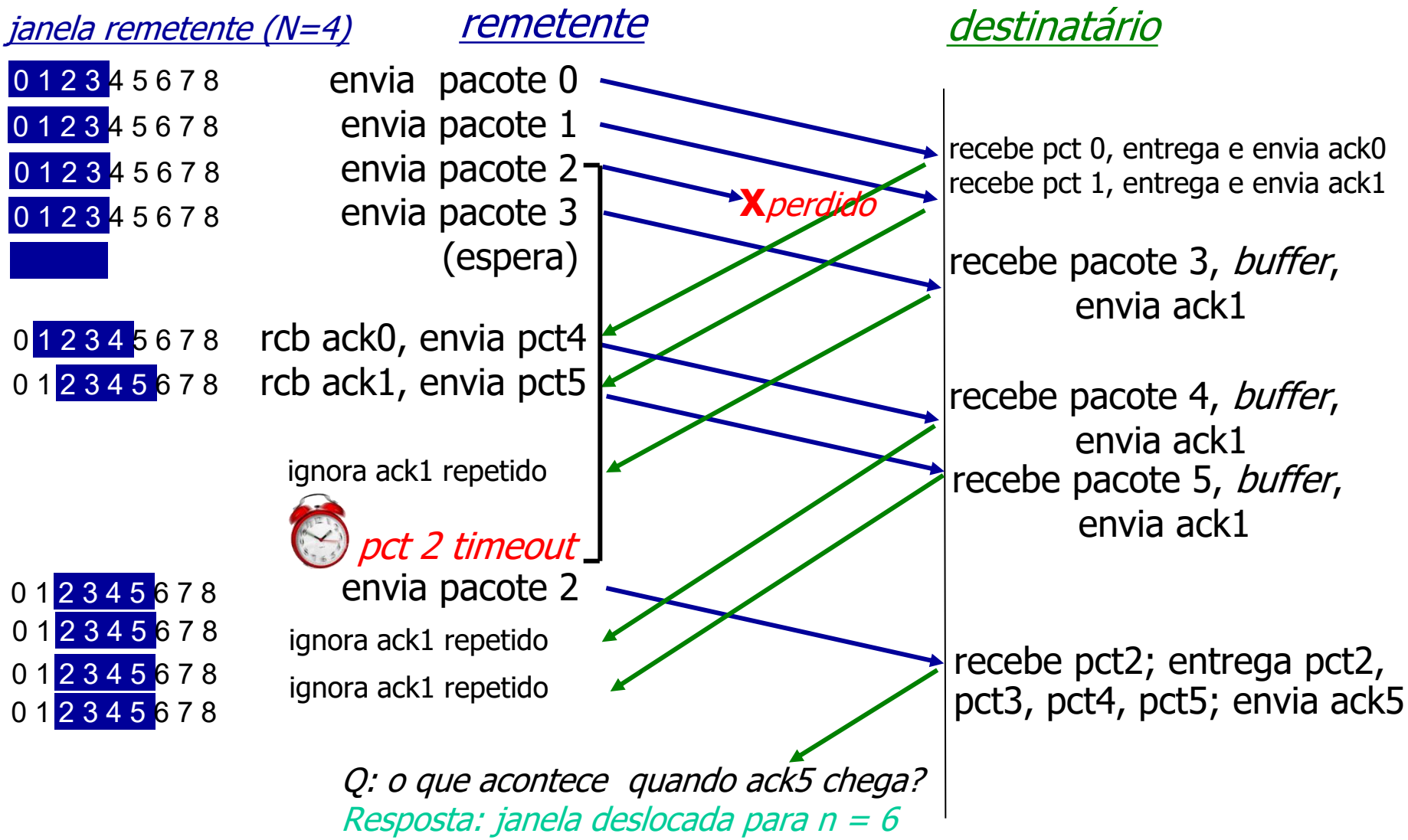
*Timeout:*

- Retransmite ***apenas*** segmento que causou *timeout (parece RS)*
- reinicializa temporizador

*ACK recebido (acumulativo):*

- Se *ACK* reconhece segmentos previamente sem *ACK*
  - Atualizar o que sabe-se transmitido com sucesso
  - Reiniciar temporizador se ainda existem segmentos sem ACK

# TCP (com *buffer*) simplificado em ação



# TCP: transferência de dados confiável

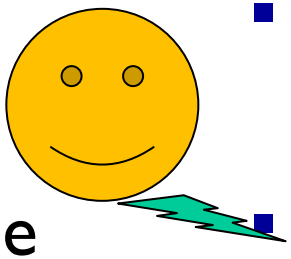
- Discussão: TCP é GBN ou Repetição Seletiva?

- Resposta: Versão híbrida
  - Um só temporizador e ACK cumulativo como GBN
  - Retransmissão apenas de pacote que gerou *timeout* como repetição seletiva.
  - Mais uma série de "truques" aprendidos ao longo de mais de 40 anos de pesquisas e experiências...

# TCP: *round trip time, timeout* (RFC 6298)

Q: como ajustar valor de *timeout do* TCP?

- Primeira condição: maior do que RTT
  - Mas RTT varia...
- *Se for muito curto*: *timeout* prematuro, retransmissões desnecessárias
- *Se for muito longo*: reação lenta a perda de segmento



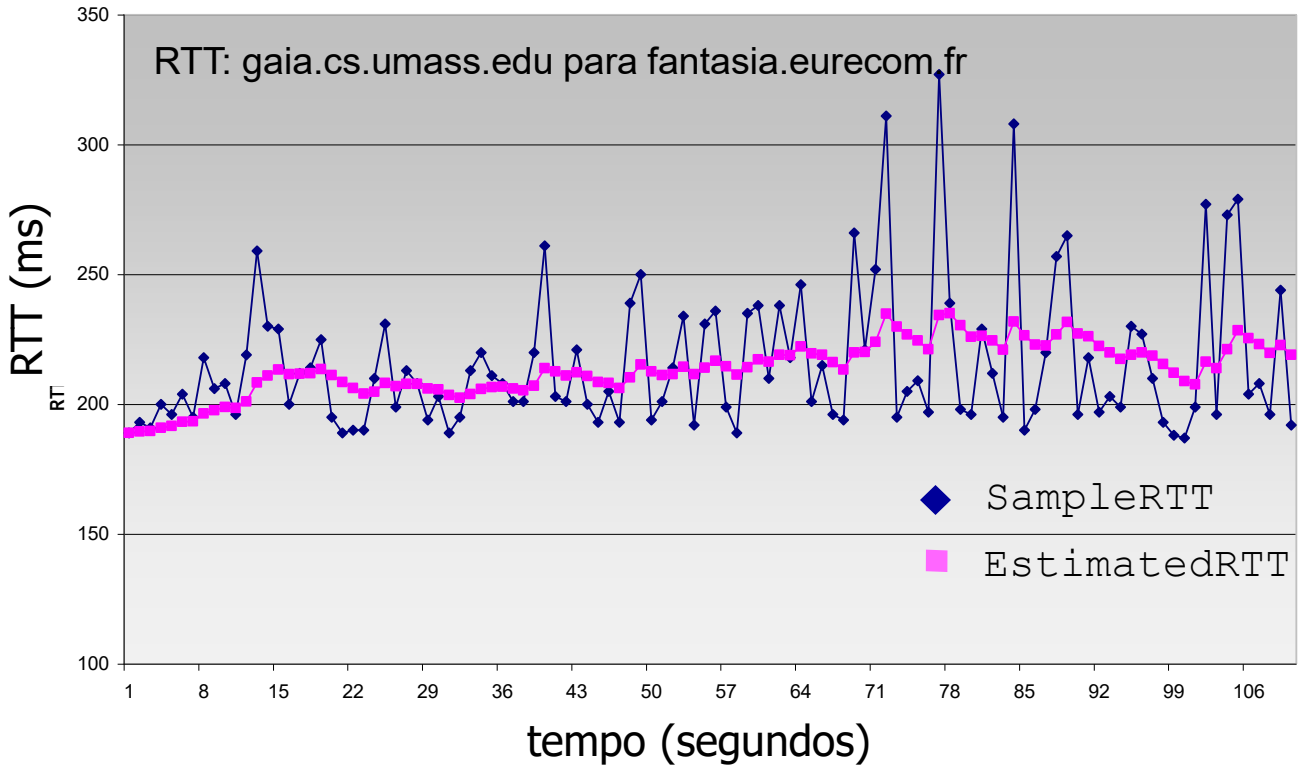
Q: Como estimar RTT?

- **`SampleRTT`**: tempo medido da transmissão de segmento até ACK recebido
  - ignora retransmissões
- **`SampleRTT`** irá variar, queremos RTT estimado "mais suave"
  - tomar média de medidas *recentes* não apenas o **`SampleRTT`** atual
  - Filto IIR passa-baixas!

# TCP: *round trip time, timeout – cálculo simplificado*

```
EstimatedRTT(t) = (1- α)*EstimatedRTT(t-1) + α*SampleRTT(t)
```

- *Média móvel com ponderação exponencial*
- Influência das amostras passadas decresce exponencialmente rápido
- valor típico : $\alpha = 1/8$ (RFC 6298)



# TCP: *round trip time, timeout – cálculo simplificado*

- Intervalo de *timeout*: **EstimatedRTT** mais "margem de segurança"
  - Maior variação em **EstimatedRTT** -> maior margem de segurança
- Estimar devio de **SampleRTT** de **EstimatedRTT**:

$$DevRTT(t) = (1-\beta)*DevRTT(t-1) + \beta*|SampleRTT(t)-EstimatedRTT(t)|$$

(com $\beta = 1/4$ (RFC 6298))

$$TimeoutInterval(t) = EstimatedRTT(t) + 4*DevRTT(t)$$

RTT estimado

"margem de segurança"

# Exercício

5) Considere que o RTT estimado (`EstimatedRTT`) seja calculado pela fórmula

$$\texttt{EstimatedRTT}(t) = 0.9 * \texttt{EstimatedRTT}(t-1) + 0.1 * \texttt{SampleRTT}(t),$$

similar à discutida em aula. Assuma que o `EstimatedRTT` num certo instante seja 200 ms para uma conexão TCP e que a conexão mede os três próximos RTTs em 200, 200 e 100 ms. Qual é o valor de `EstimatedRTT` depois de processados os novos dados?

(A) 190 ms

(B) 175 ms

(C) 200 ms

(D) 100 ms

(E) 150 ms

(A) $\texttt{EstimatedRTT}(t) = 0.9 * \texttt{EstimatedRTT}(t-1) + 0.1 * \texttt{SampleRTT}(t)$. `EstimatedRTT` não muda depois de se processar as duas primeiras amostras. Para a 3ª amostra, `EstimatedRTT` = 0.9*200 + 0.1*100 = 190 ms