PTC 3450 - Aula 14

3.4 Princípios da transferência de dados confiável
3.5 Transporte orientado para conexão: TCP

(Kurose, p. 164 - 177)
(Peterson, p. 242-264)

02/05/2017

Muitos slides adaptados com autorização de J.F Kurose and K.W. Ross, All Rights Reserved

Capítulo 3: conteúdo

Protocolos com paralelismo: visão geral

Go-back-N (GBN):

- Transmissor pode ter até *N* pacotes que ainda não retornaram ACKs
- Receptor envia *ACK acumulativo*
  - ACK *x* implica que todos os pacotes até *x* foram recebidos
- Transmissor tem apenas um temporizador associado ao pacote não reconhecido mais antigo
  - quando tempo expira, retransmite *todos* pacotes não reconhecidos

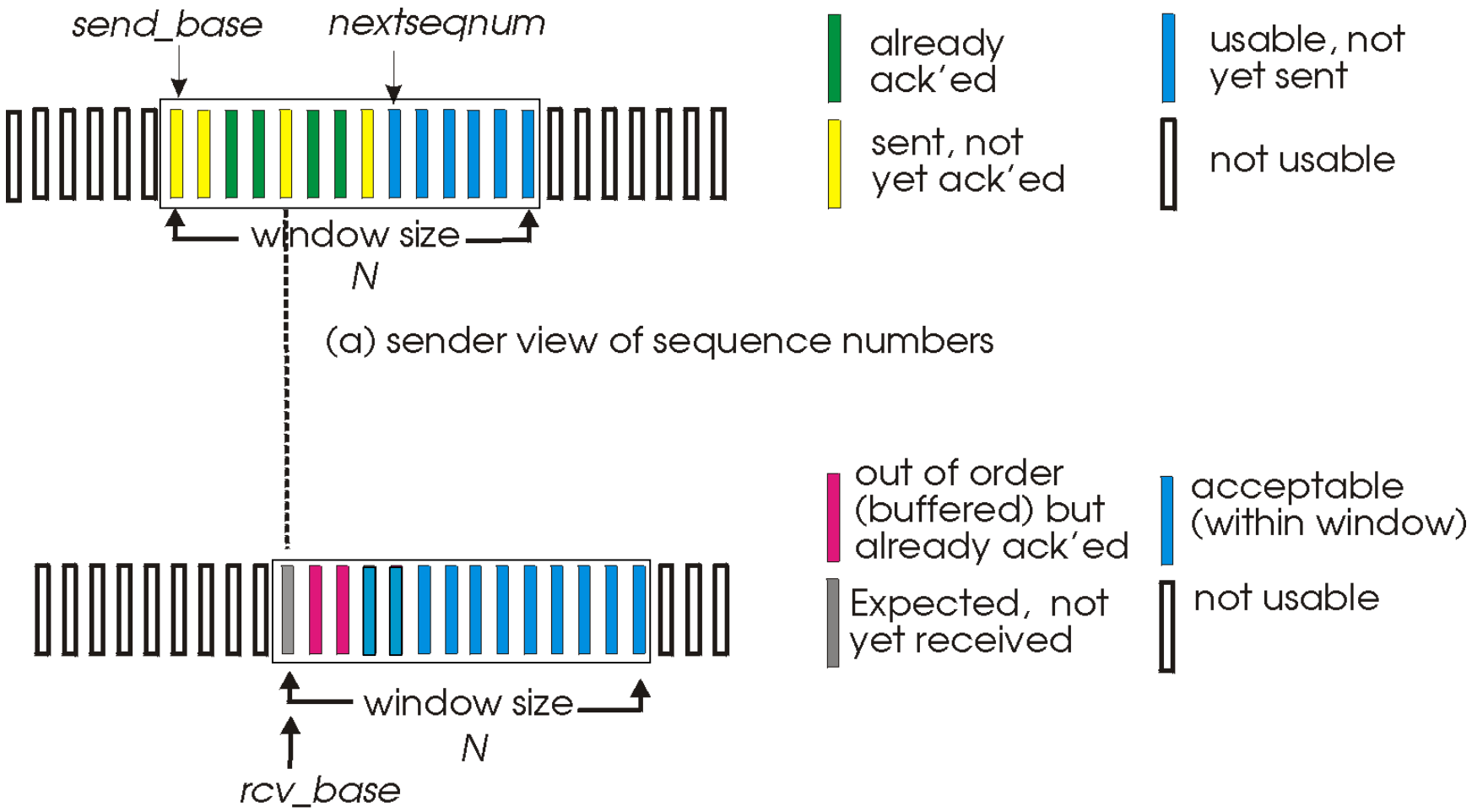
Repetição Seletiva (RS):

- Transmissor pode ter até *N* pacotes que ainda não retornaram ACKs
- Receptor envia *ACK individual* para cada pacote
- Transmissor mantém temporizador para cada pacote ainda não reconhecido
  - quando tempo expira, retransmite apenas aquele pacote associado com o temporizador expirado

Repetição Seletiva

- receptor reconhece *individualmente* cada pacote recebidos corretamente
  - pacotes colocados em *buffer*, conforme necessário, para eventual entrega em ordem para camada superior
- transmissor apenas reenvia pacotes para os quais ACK não foi recebido
  - temporizador para cada pacote não ACK
- janela deslizante
  - janelas podem estar deslocadas entre remetente e destinatário
  - mais números sequenciais necessários

Repetição seletiva : janelas transmissor, receptor



Repetição seletiva

transmissor

dados de cima:

- se `nextseqnum` dentro da janela, envia pacote

timeout(n):

- reenvia pacote *n*, reinicializa temporizador *n*

ACK(n) em [`send_base`,`send_base+N-1`]:

- marca pacote *n* como recebido
- se *n* é menor pacote que não havia recebido ACK, avança `send_base` para próximo # seq. ainda sem ACK

receptor

recebe pacote com número sequencial *n* em [`rcv_base`, `rcv_base+N-1`]

- envia ACK(*n*)
- fora de ordem: *buffer*
- na ordem: entrega (também entrega pacotes no *buffer* em ordem), avança `rcv_base` para próximo pacote ainda não recebido

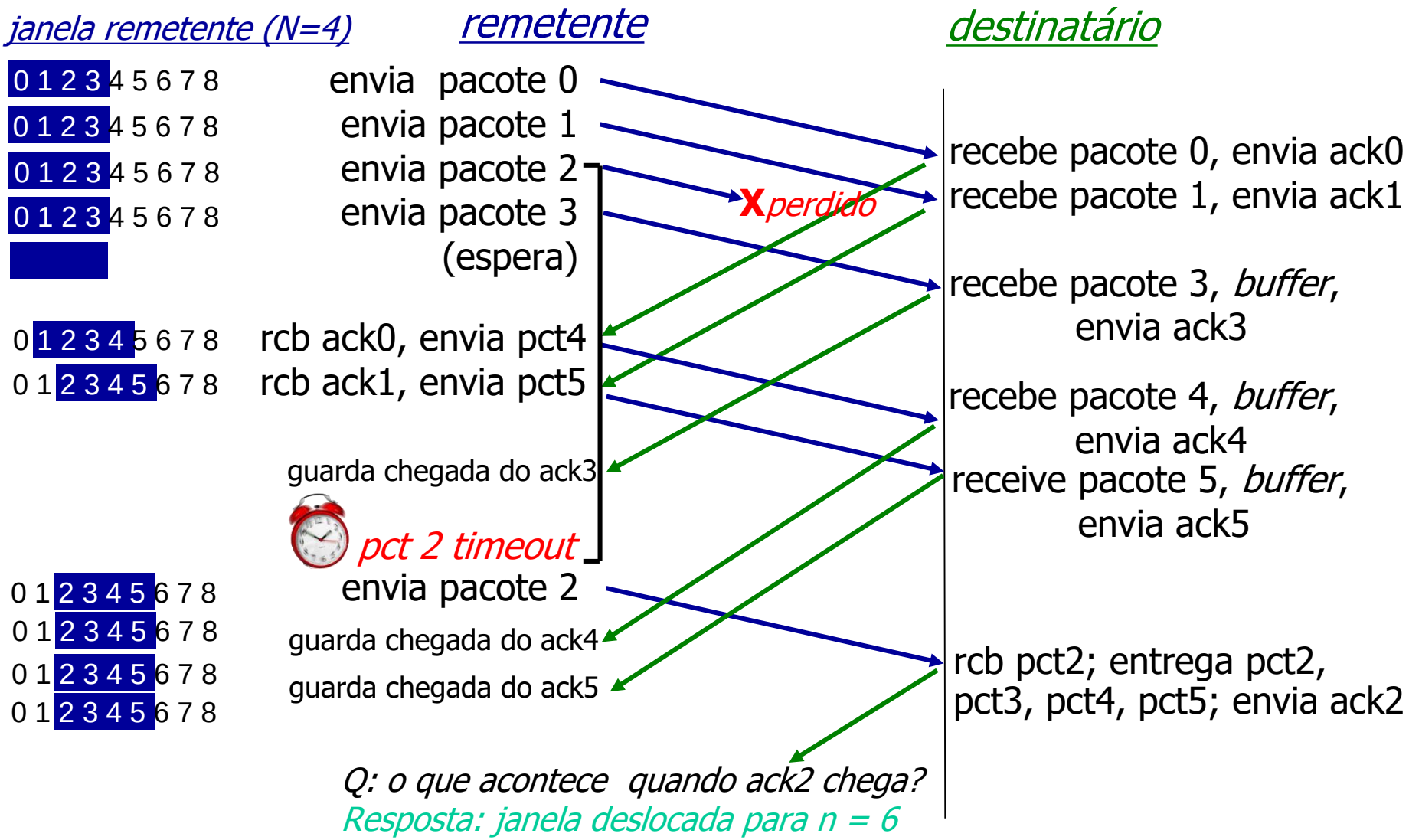
pacote n em [`rcv_base-N`,`rcv_base-1`]

- ACK(*n*)

caso contrário:

- ignora

Repetição seletiva em ação



Repetição seletiva: dilema

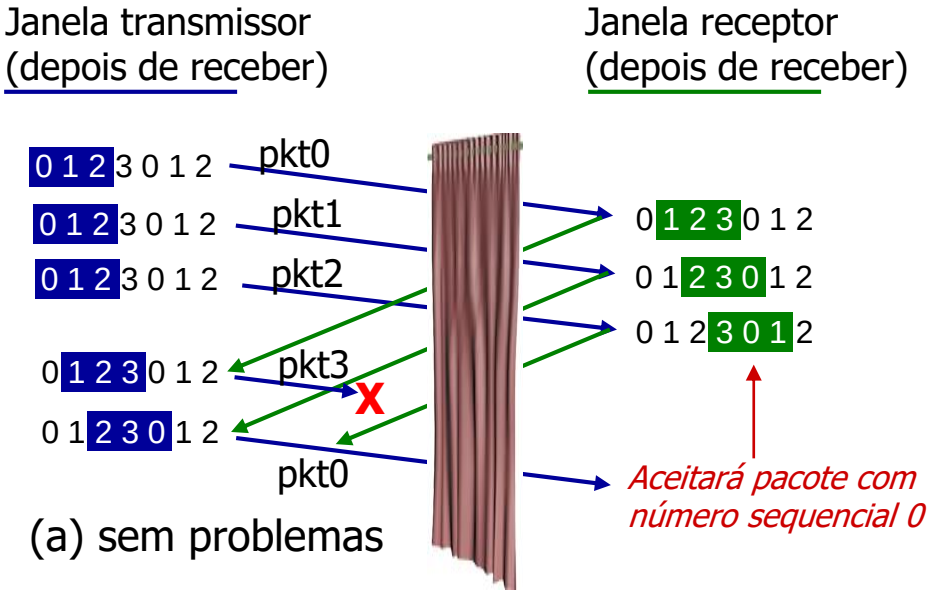
exemplo:

- #'s seq : 0, 1, 2, 3
- *N=3*

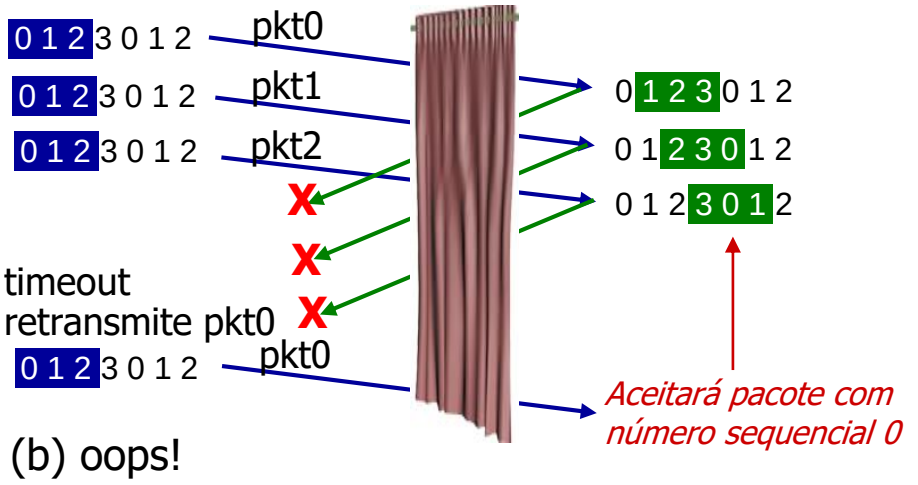
- Receptor não vê diferença entre dois cenários!
- Dados duplicados aceitos como novos em (b)

Q: Qual a relação entre #'s sequenciais e *N* para evitar problema em (b)?

Resposta: intervalo de comprimento 2N



Receptor não consegue ver lado transmissor
Receptor se comporta de forma igual nos dois casos! Alguma coisa está (muito) errada!



Comparação: Vantanges do GBN e da Repetição Seletiva

- GBN
  - Menor complexidade – apenas um temporizador; não há necessidade de janelas no receptor ☺
  - Não são necessário *buffers* ☺
  - Muitas retransmissões desnecessárias ☹

- Repetição seletiva
  - Menos retransmissões – apenas pacotes de fato necessários são retransmitidos (pacote ou ACK perdidos) ☺
  - Maior complexidade – um temporizador por pacote ☹
  - *Buffers* necessários para armazenar pacotes fora de ordem no receptor e ACKs no transmissor ☹

- Qual é usado nos protocolos em geral?

Resposta: São casos extremos de simplicidade e complexidade, respectivamente. Veremos que o TCP usa uma combinação deles...

Resumo: Mecanismos para transferência de dados confiável

- *Códigos para detecção ou correção de erros* – 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??*

Capítulo 3: conteúdo

3.1 serviços da camada de transporte

3.2 multiplexação e desmultiplexação

3.3 transporte sem conexão: UDP

3.4 princípios da transferência de dados confiável

3.5 transporte orientado para conexão: TCP

- estrutura dos segmentos
- transferência de dados confiável
- controle de fluxo
- gerenciamento de conexão

3.6 princípios do controle de congestionamento

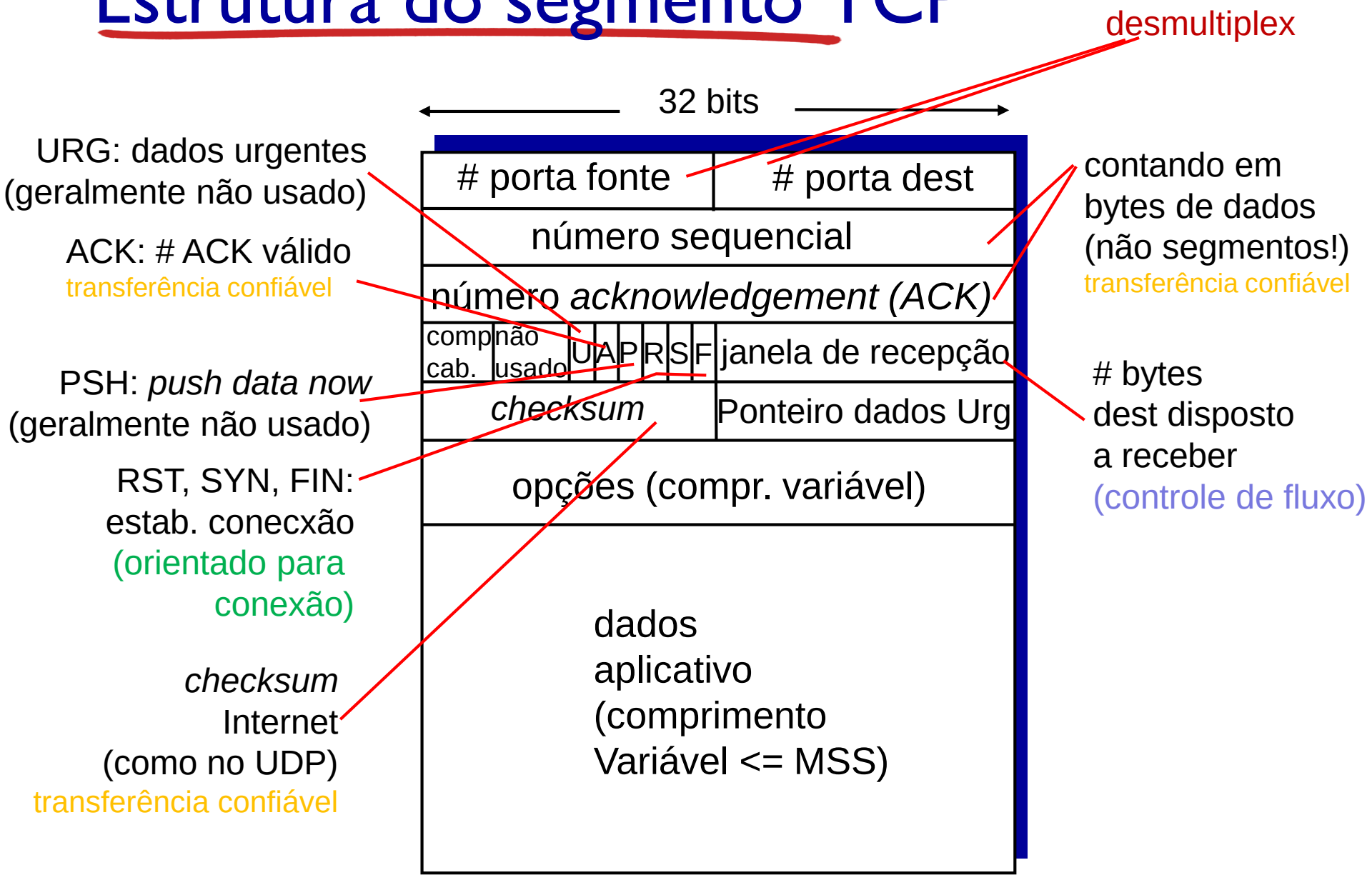
3.7 controle de congestionamento no TCP

TCP: Visão geral RFC 761 (1980),..., RFC 8095(2017)

- [Cerf & Kahn1974] – IEEE Trans. on Com. Tech.
- ponto a ponto:
  - 1 remetente, 1 destinatário
- fluxo de bytes confiável e em ordem:
  - Variáveis e *buffers* apenas no remetente e destinatário
- usa paralelismo (*pipelined*):
  - controle de fluxo e congestionamento do TCP regulam comprimento da janela
- dados *full duplex*:
  - fluxo de dados bidirecional na mesma conexão
  - MSS: máximo comprimento de segmento (sem cabeçalho); típico 1500 bytes
- orientado para conexão:
  - *handshaking* (troca de mensagens de controle) inicializa estado do remetente e destinatário antes da troca de dados (varíaveis, *buffers*)
- fluxo controlado:
  - remetente não sobrecarrega destinatário

Veja: http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/flow/FlowControl.htm

Estrutura do segmento TCP



TCP: números seq. e ACKs

números sequenciais:

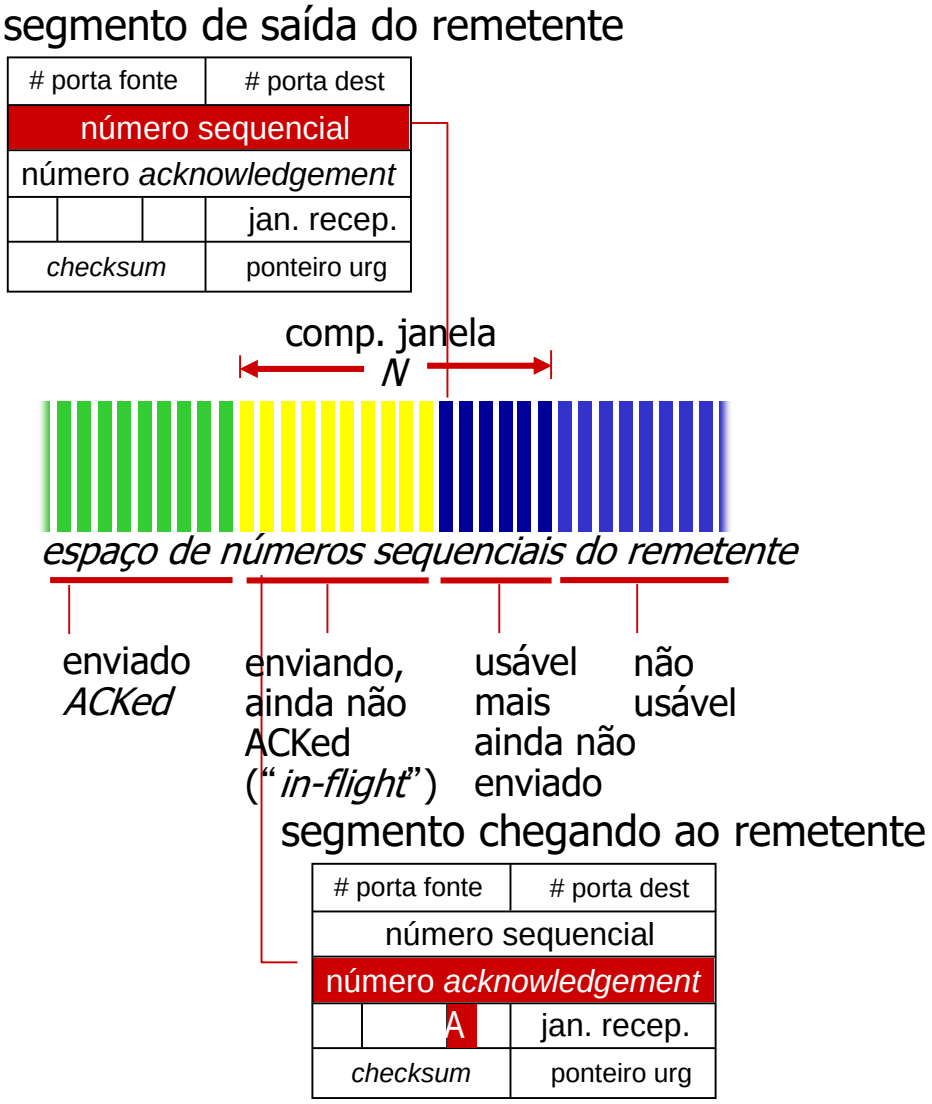
- "número" no fluxo de bytes do 1º byte nos dados do segmento

acknowledgements:

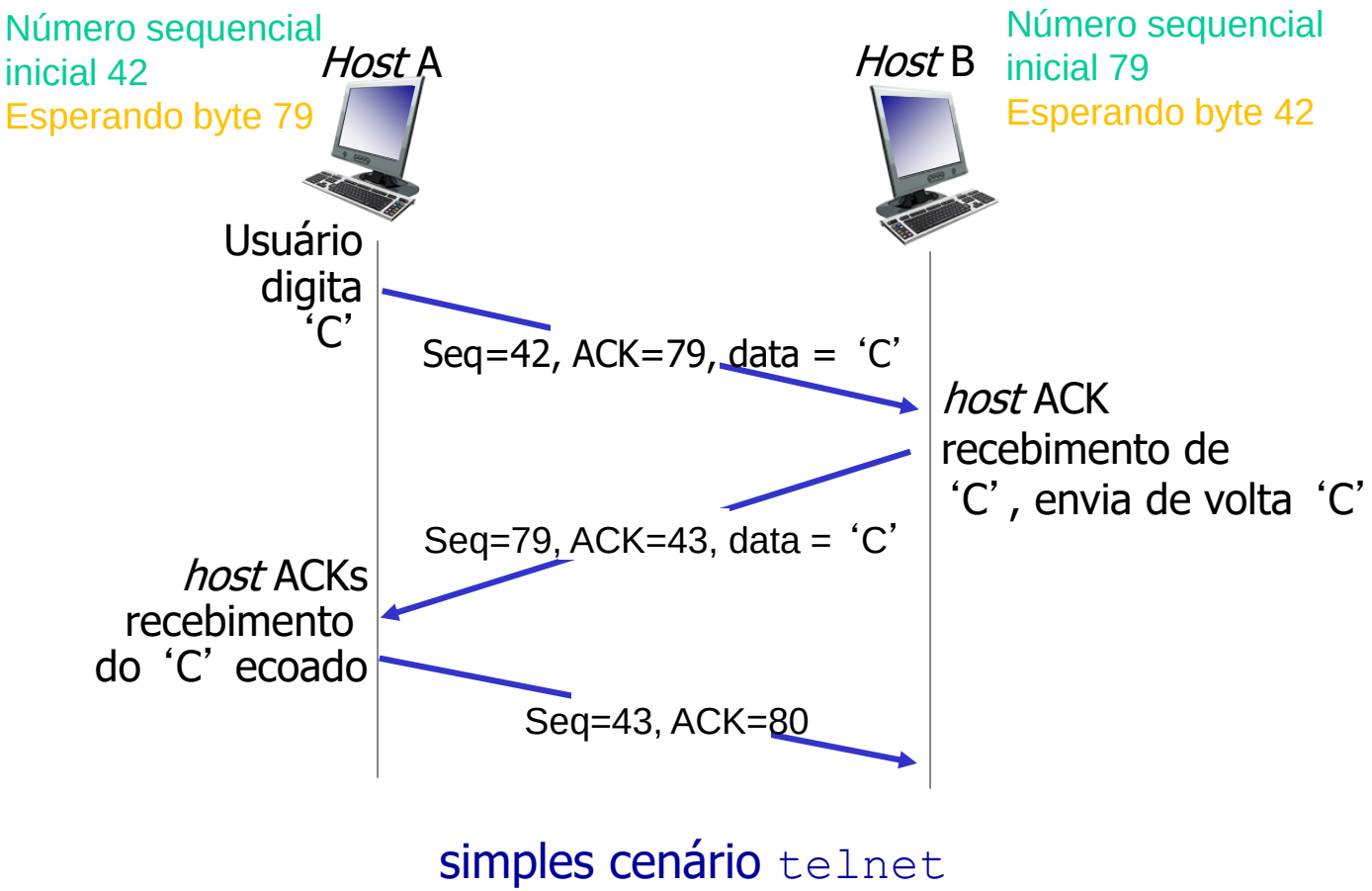
- # seq do próximo byte esperado do outro lado
- ACK acumulativo
- Levado no segmento de dados seguinte destinatário-remetente (pega carona)

Q: como destinatário lida com segmentos fora de ordem?

- R: especificação TCP não define, - a critério da implementação – Abordagem prática: coloca em *buffer*



TCP: números seq. e ACKs



simples cenário `telnet`

Exercício

1. (Kurose e Ross, 2013, p. 215) (2,0) Considere a transferência de um arquivo enorme de L bytes do *host* A para o *host* B. Suponha um MSS de 536 bytes.

(a) Qual é o máximo valor de L tal que não sejam esgotados os números sequenciais do TCP? Lembre-se que o campo de número sequencial no TCP tem 4 bytes.

(b) Para o L que obtiver em (a), descubra quanto tempo demora para transmitir o a arquivo. Admita que um total de 66 bytes de cabeçalho de transporte, de rede e de enlace de dados seja adicionado antes que o pacote seja enviado por um enlace de 155 Mbits/s. Ignore controle de fluxo e controle de congestionamento de modo que A possa enviar os segmentos um atrás do outro e continuamente.

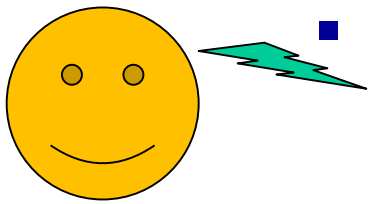
TCP: *round trip time, timeout* (RFC 6298)

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

- maior do que RTT
  - mas RTT varia
- *muito curto*: *timeout* prematuro, retransmissões desnecessárias
- *muito longo*: reação lenta a perda de segmento

Q: como estimar RTT?

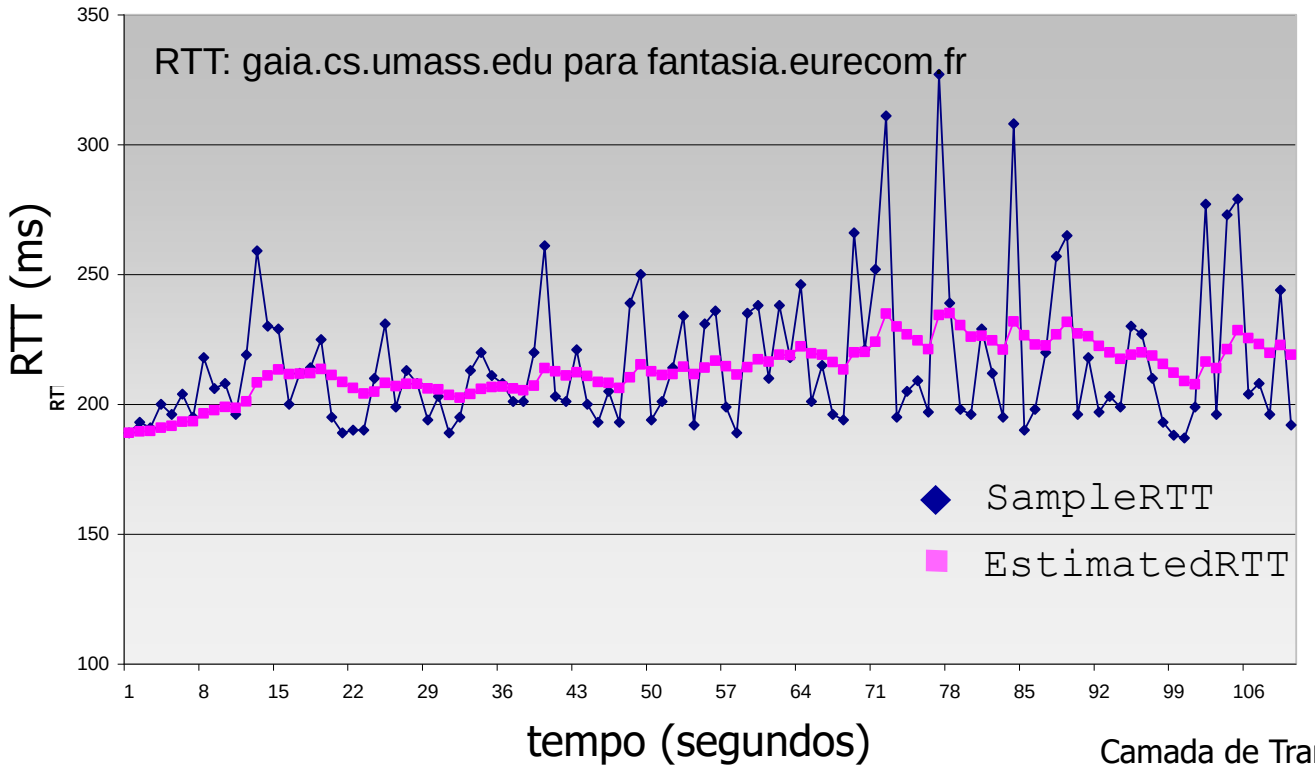
- **`SampleRTT`**: tempo medido da transmissão de segmento até ACK recebido
  - ignora retransmissões
- **`SampleRTT`** irá variar, queremos RTT estimado "mais suave"
  - tomar média de medidas *recentes* não apenas o **`SampleRTT`** atual
  - Filto IIR passa-baixas!



TCP: *round trip time, timeout*

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

- *média móvel com ponderação exponencial*
- influência das amostras passadas decresce exponencialmente rápido
- valor típico : $\alpha = 0.125$ (RFC 6298)



TCP: *round trip time, timeout*

- 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-β)*DevRTT(t-1) + β*|SampleRTT(t)-EstimatedRTT(t)|
```

(tipicamente, β = 0.25)

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

RTT estimado

"margem de segurança"