PTC 2550 - Aula 03

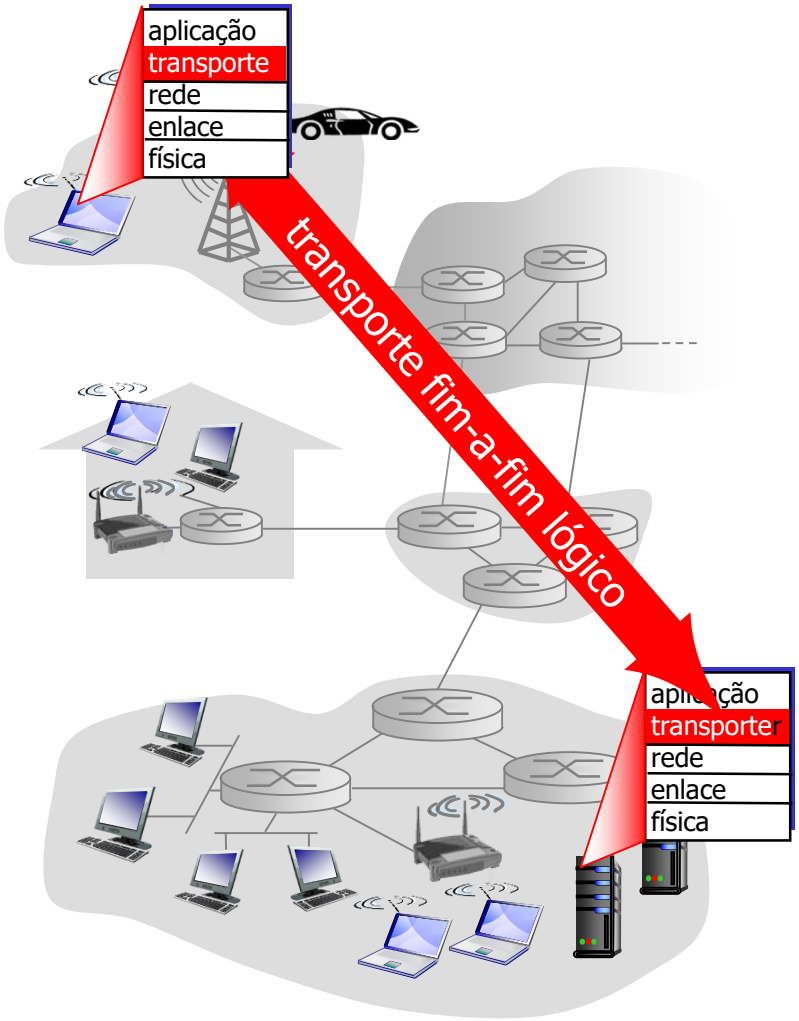
1.3 A camada de transporte

(Kurose, p. 135 - 209)

15/03/2017

Serviços e protocolos de transporte

- provê *comunicação lógica* entre *aplicativos* rodando em *hosts* diferentes
- protocolos de transporte rodam em sistemas finais
  - lado enviar: converte (quebrar + inserir cabeçalho) mensagem do *aplicativo* em *segmentos*, passa para a camada de rede
  - lado receber: reconverte segmentos em mensagens, passa para camada de aplicação
- mais de um protocolo de transporte pode estar disponível para aplicativo
  - Internet: TCP e UDP



Camada de transporte vs. camada de rede

- *camada de rede :* comunicação lógica entre *hosts*
- *camada de transporte:* comunicação lógica entre processos
  - conta com e aprimora serviços da camada de rede

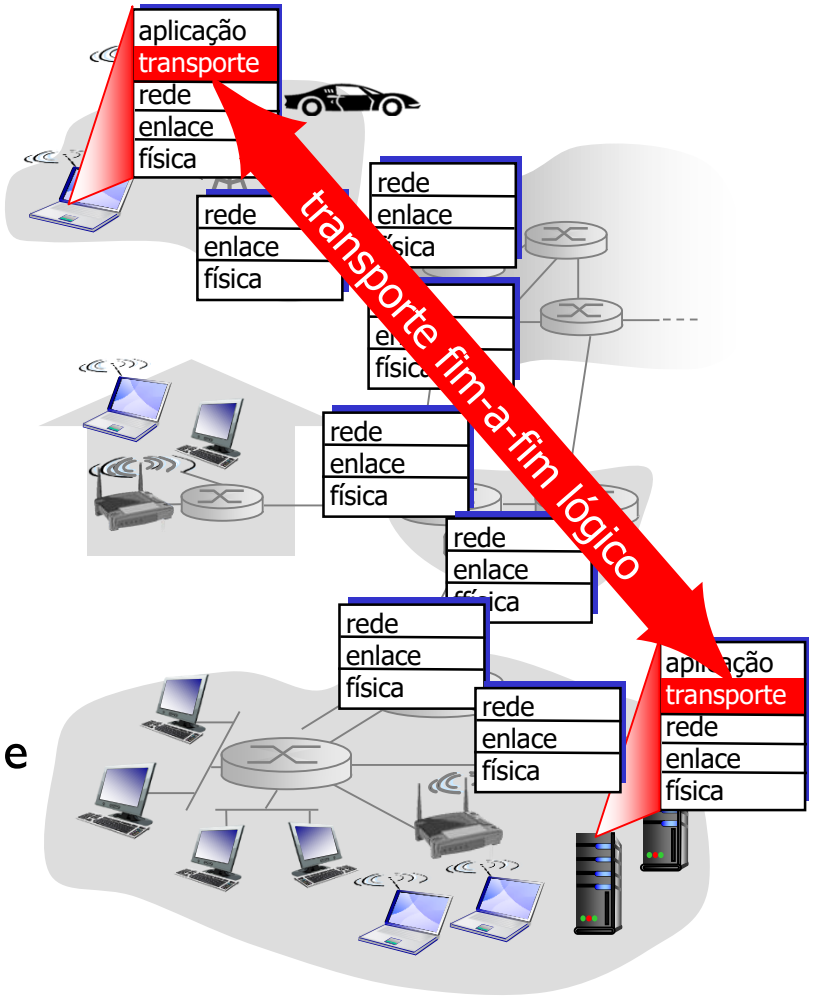
analogia familiar:

12 crianças na casa de Ana enviando cartas para 12 crianças na casa de Bruno:

- *hosts* = casas
- processos = crianças
- mensagens *app* = cartas em envelopes
- protocolo de transporte = Ana e Bruno que fazem demux interno entre os irmãos
- protocolo da camada de rede = serviço de correio

Protocolos da camada de transporte na Internet

- entrega confiável e em ordem (TCP)
  - controle de congestionamento
  - controle de fluxo
  - *setup* de conexão
- entrega não confiável e desordenada: UDP
  - extensão "sem frescuras" do "melhor-esforço" do IP
  - Apenas multiplexação/desmultiplexação e detecção de erro
- serviços não disponíveis:
  - garantias de latência
  - garantias de capacidade



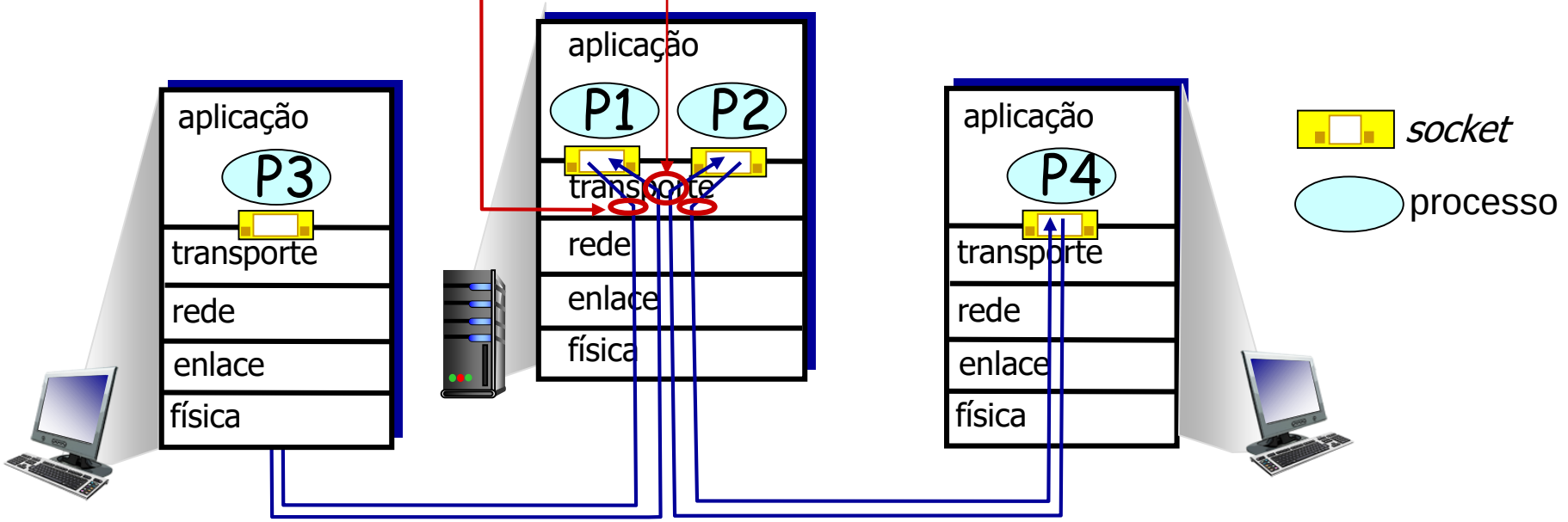
Multiplexação/desmultiplexação

multiplexando no remetente:

manipula dados de múltiplos *sockets*, adiciona cabeçalho de transporte (usado depois para desmultiplexar)

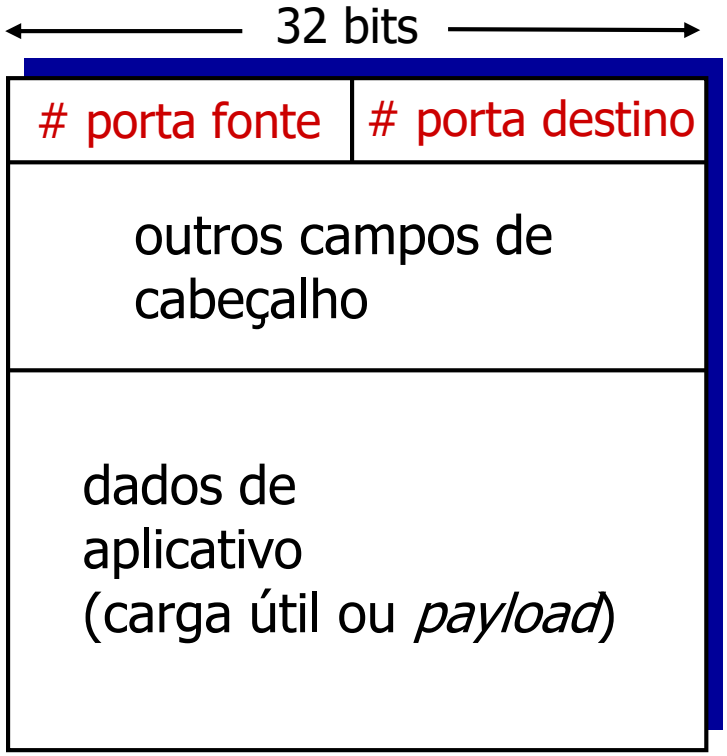
desmultiplexando no destinatário:

usa info do cabeçalho para entregar segmentos recebidos para *socket* correto



Como a *desmultiplexação* funciona

- *host* recebe datagramas IP
  - cada datagrama tem endereço IP da fonte e endereço IP do destino
  - cada datagrama carrega um segmento da camada de transporte
  - cada segmento tem números de porta destino e fonte (16 bits cada – de 0 a 1023 – reservado – RFC 1700)
- *host* usa *endereços IP & números de portas* para direcionar segmento para o *socket* apropriado



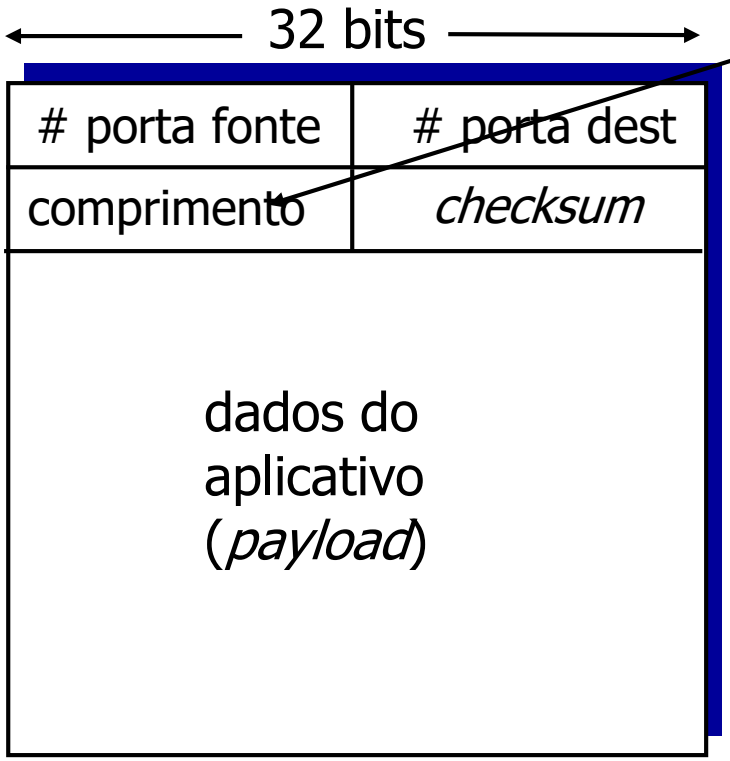
formato do segmento TCP/UDP

UDP: *User Datagram Protocol* [RFC 768]

- protocolo de transporte Internet "sem frescura", "minimalista", faz o mínimo possível
- serviço "melhor esforço", segmentos UDP podem ser:
  - perdidos
  - entregues fora de ordem para *app*
- *sem conexão:*
  - não há *handshaking* entre remetente e destinatário UDP
  - cada segmento UDP tratado independentemente de outros

- usam UDP:
  - alguns apps de *streaming* multimídia (tolerantes a perdas, sensíveis a taxa)
  - DNS
  - SNMP (*Simple Network Management Protocol*)
- transferência confiável sobre UDP:
  - adicionar confiabilidade na camada de aplicação
  - recuperação de erro específica da aplicação!

Cabeçalho do segmento UDP (RFC 768)



comprimento, em bytes do segmento UDP, incluindo cabeçalho

formato do segmento UDP

por que existe um UDP?

- sem estabecimento de conexão (que pode adicionar atraso)
- simples: sem estado da conexão no remetente e no destinatário
- pequeno cabeçalho(UDP = 8 bytes; TCP = 20 bytes)
- melhor controle no nível da aplicação sobre quais dados são enviados e quando

Checksum UDP

Objetivo: detectar "erros" (e.g., bits alterados) em segmento transmitido

remetente:

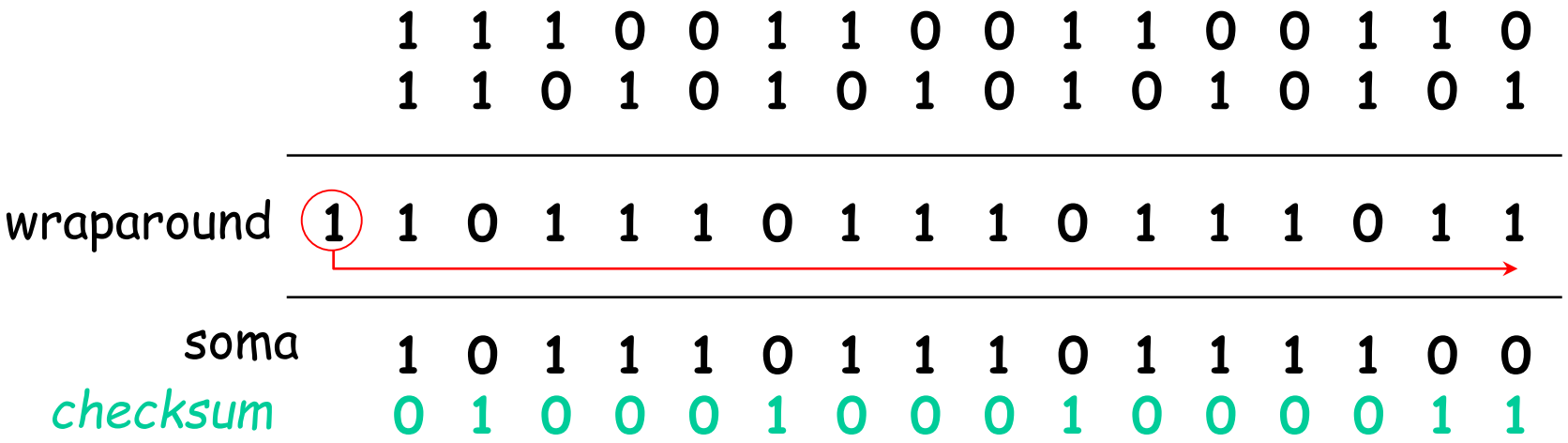
- trata conteúdo do segmento, incluindo campos de cabeçalho, como sequência de inteiros de 16-bit
- *checksum*: complemento de 1 da soma do conteúdo do segmento
- remetente coloca valor *checksum* no campo UDP *checksum*

destinatário:

- calcula *checksum* do segmento recebido
- verifica se *checksum* calculado é igual ao valor no campo *checksum*:
  - NÃO - erro detectado
  - SIM – nenhum erro detectado. Mas podem haver erros mesmo assim? Mais adiante…

Checksum Internet : exemplo

exemplo: soma de inteiros de 16-bit



Notas:

1) quando somamos números, o "vai um" do bit mais significativo precisa ser adicionado ao resultado
2) Se não houver erros, soma de todas as palavras de 16 bits no destino deve ser 1111111111111111.

RDT – *Reliable Data Transfer*

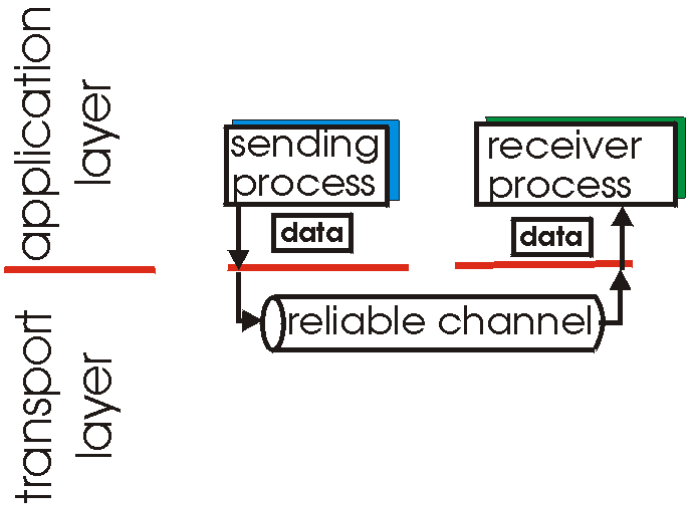
- Um dos aspectos que permeiam todas a pilha de protocolos é o de *transferência confiável de dados* (RDT – *reliable data transfer*)

- Entre fonte e destino, pacotes podem passar por diversos enlaces diferentes, com características e probabilidade de erros muito diferentes
  - Podem ser descartados em cada um dos roteadores no meio da rota
  - Podem ser corrompidos
  - Podem percorrer rotas diferentes, tendo atrasos diferentes

- Mesmo assim, muitas aplicações são viáveis apenas se garante-se que todos os pacotes sejam entregues e em ordem:
  - comércio eletrônico
  - e-mail
  - transferência de arquivos de dados
  - etc.

Princípios da transferência confiável de dados

- RDT usada nas camadas de aplicação, transporte e enlace
  - Tópico de importância central na área de redes!

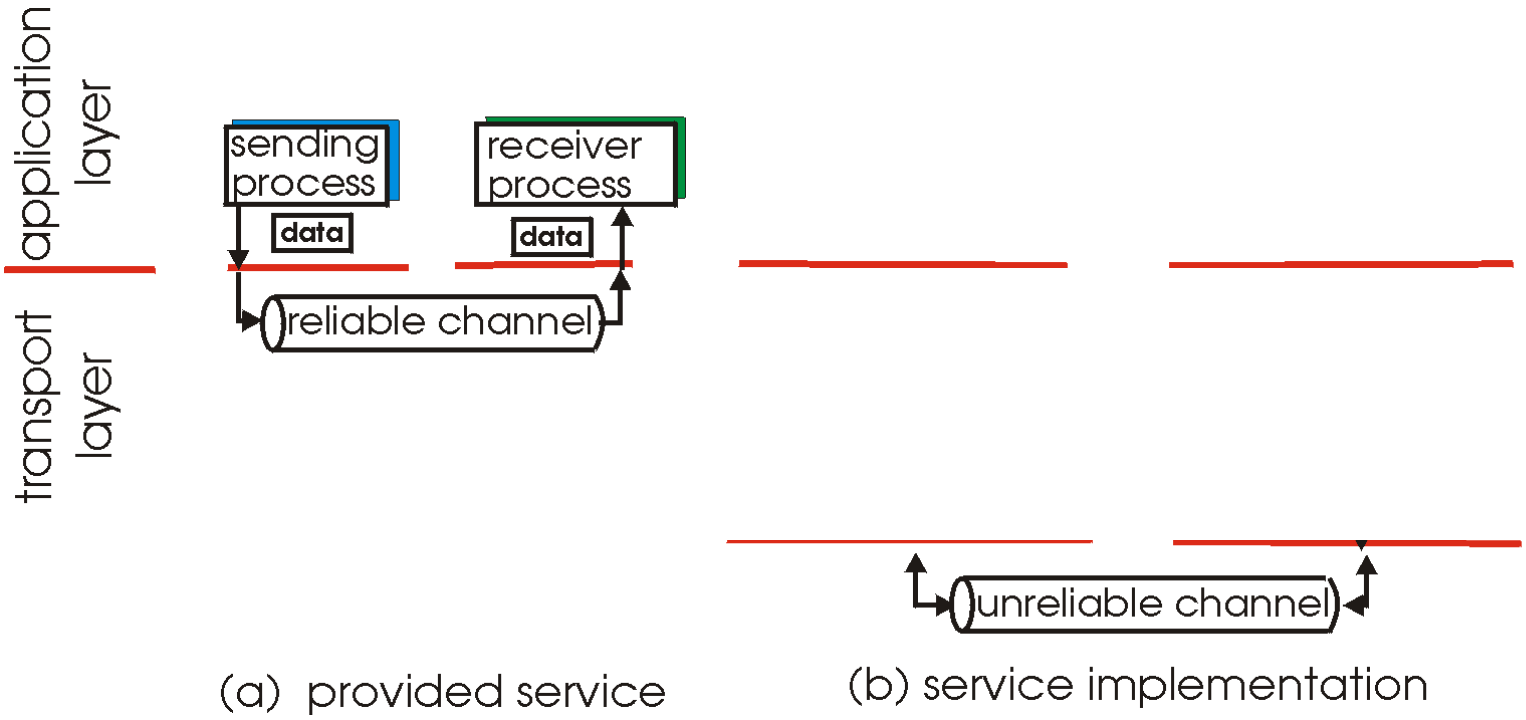


(a) provided service

- características do canal não confiável determinarão complexidade do protocolo de transferência de dados confiável (rdt - *reliable data transfer protocol*)

Princípios da transferência confiável de dados

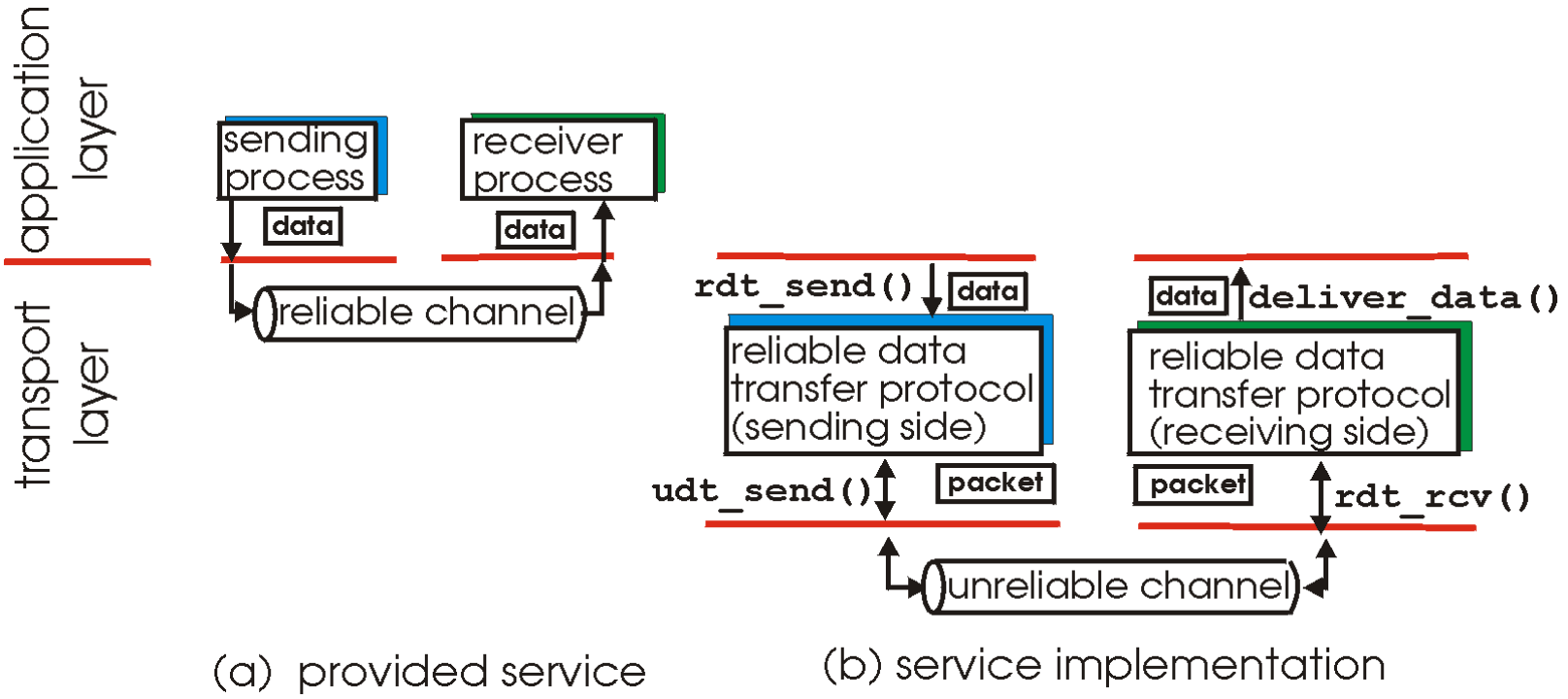
- importante nas camadas de aplicação, transporte e enlace
  - na lista top-10 dos tópicos mais importantes em redes!



- características do canal não confiável determinarão complexidade do protocolo de transferência de dados confiável (RDT - *reliable data transfer protocol*)

Princípios da transferência confiável de dados

- importante nas camadas de aplicação, transporte e enlace
  - na lista top-10 dos tópicos mais importantes em redes!



- características do canal não confiável determinarão complexidade do protocolo de transferência de dados confiável (RDT - *reliable data transfer protocol*)

Resumo: Mecanismos para transferência de dados confiável

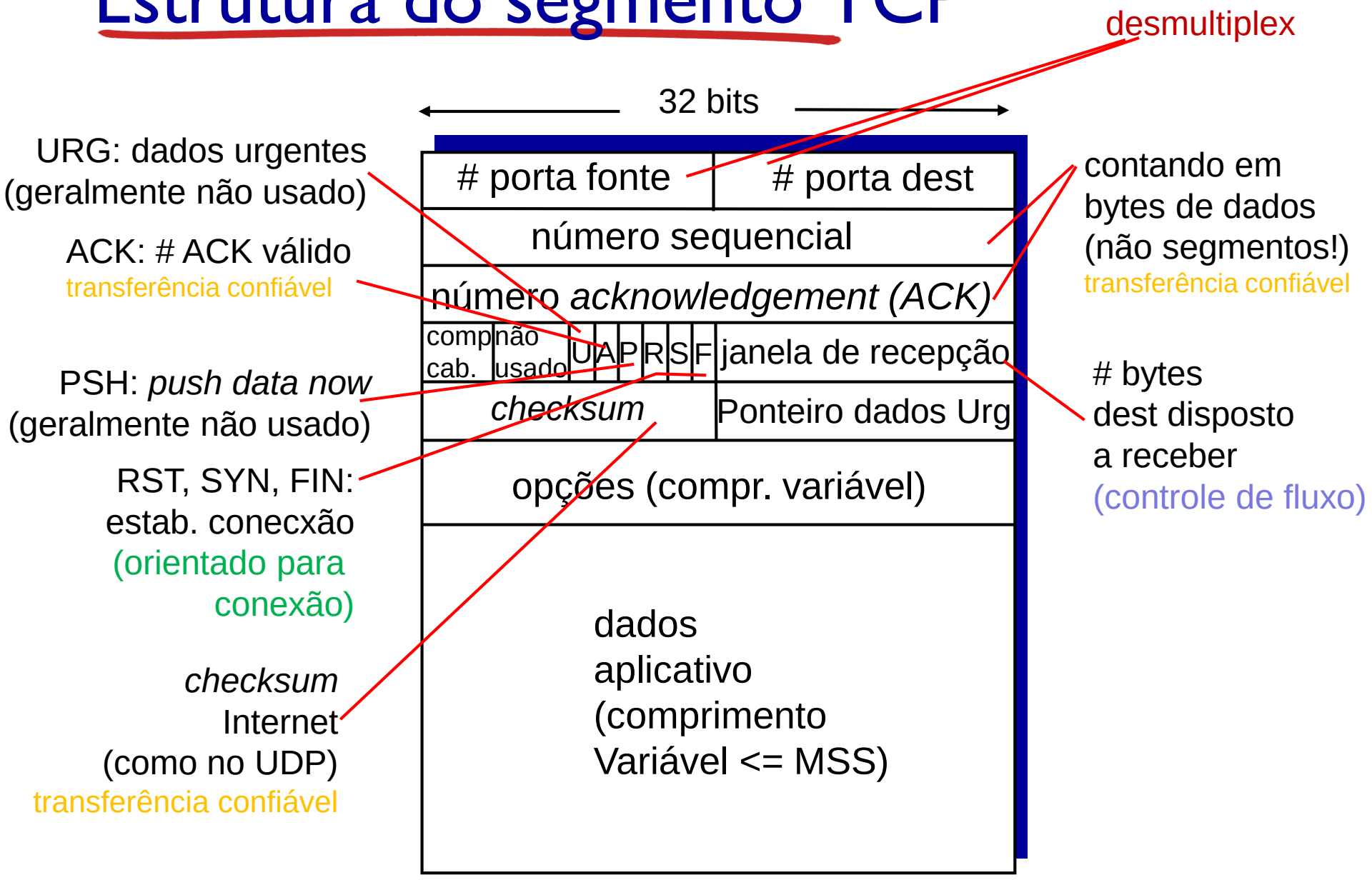
- *Códigos para detecção ou correção de erros* – detecção de erros
- *Acknowledgement* - Receptor informa que pacote ou conjunto de pacotes foi recebido corretamente. Pode ser individual ou acumulativo
- *Números sequenciais* – Detectar pacotes perdidos ou recebidos em duplicata
- *Timer* - usado para 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*

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



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.

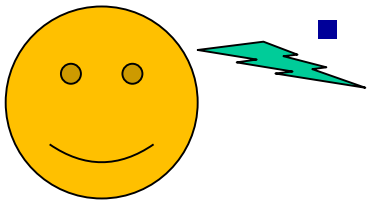
TCP: *round trip time, timeout* (RFC 6298)

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?

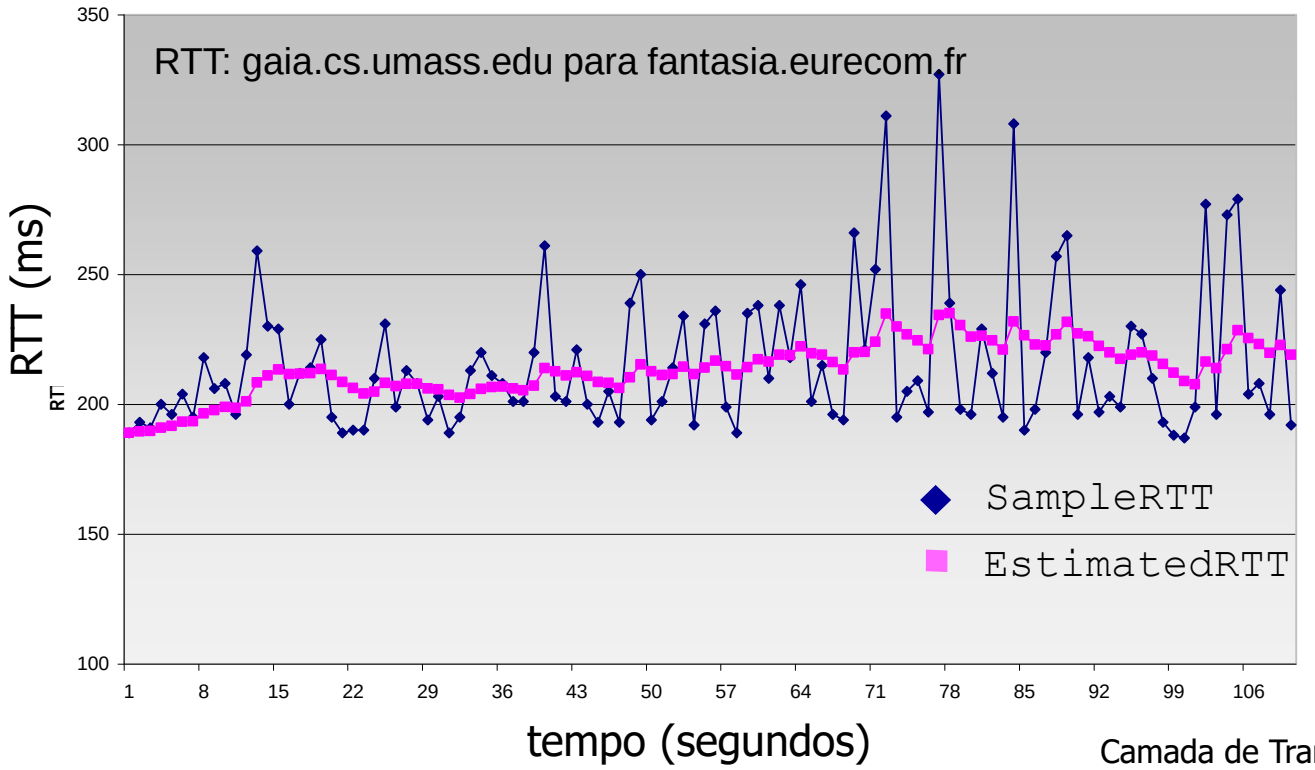
- **`SampleRTT`**: tempo medido da transmissão de segmento até ACK recebido
  - ignora retransmissões
- **`SampleRTT`** irá variar, queremos RTT estimado "mais suave"
  - tomar média de medidas *recentes* não apenas o **`SampleRTT`** atual
  - Filto IIR passa-baixas!



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**:

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

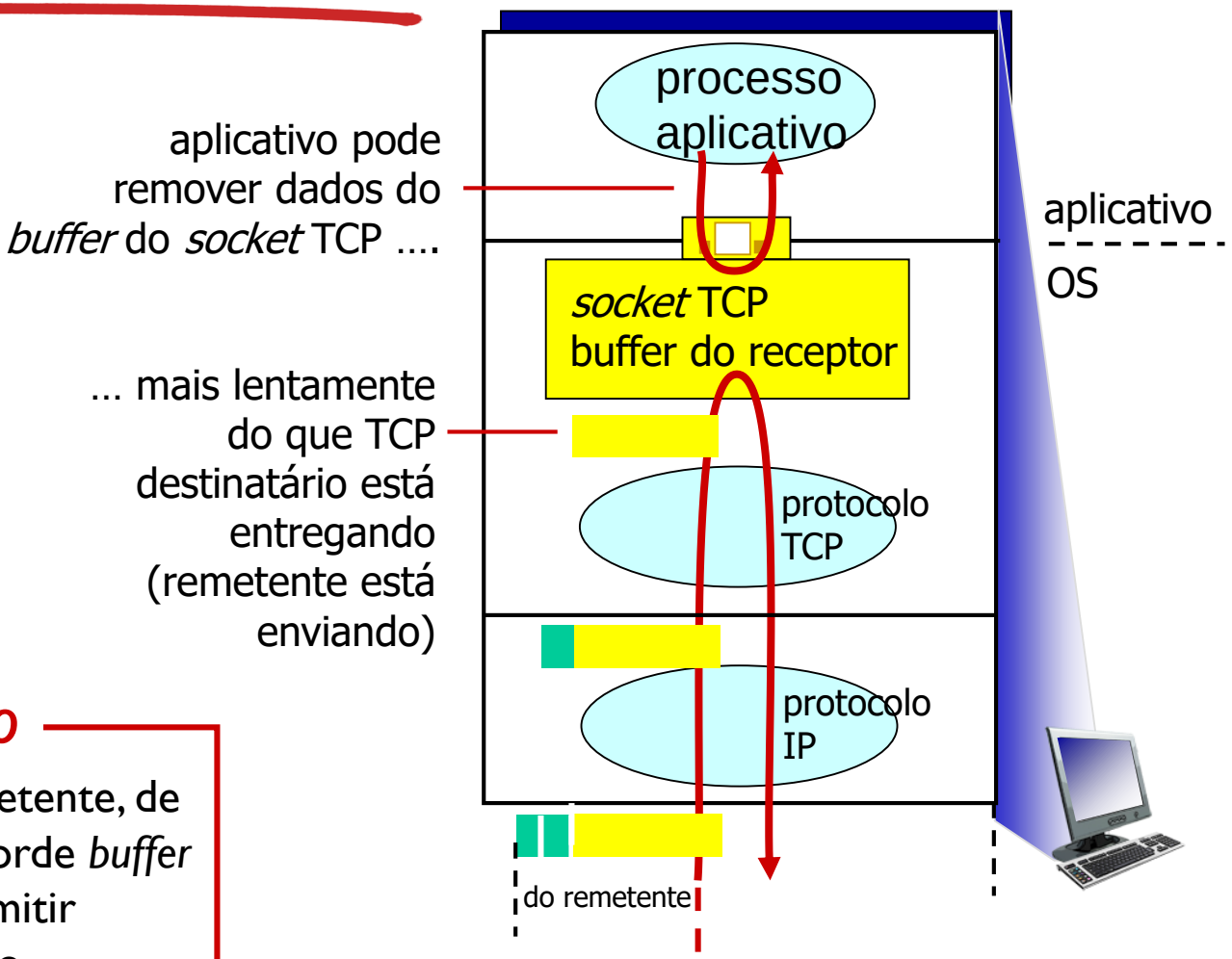
(tipicamente, $\beta = 0,25$)

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

RTT estimado

"margem de segurança"

TCP: controle de fluxo



pilha de protocolos do destinatário

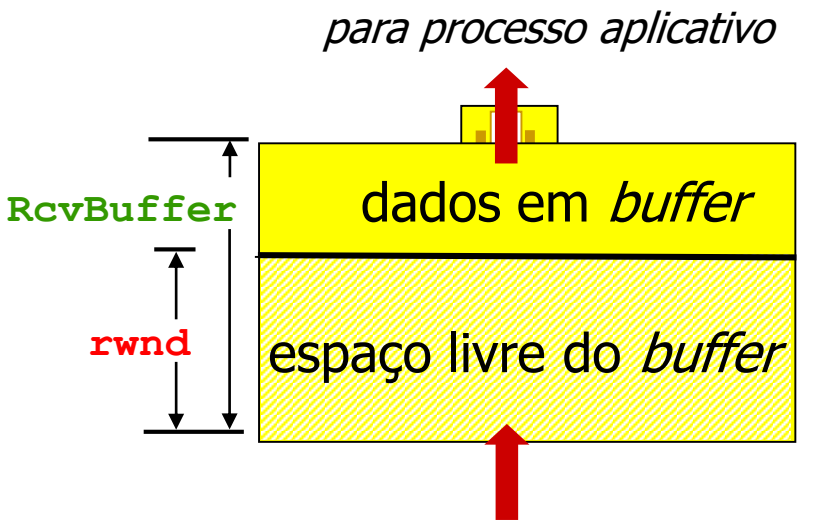
controle de fluxo

destinatário controla remetente, de forma que ele não transborde *buffer* do destinatário por transmitir muitos dados muito rápido

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

TCP: controle de fluxo

- destinatário "informa" espaço de *buffer* livre incluindo o valor **`rwnd`** no campo "janela de recepção" do cabeçalho TCP dos segmentos destinatário-para-remetente
  - **`RcvBuffer`** determinado via opção na criação do *socket* (default típico é 4096 bytes)
  - muitos sistemas operacionais autoajustam **`RcvBuffer`**
- remente limita quantidade de dados *unacked* ("*in-flight*") ao valor **`rwnd`** do destinatário
- garante que *buffer do destinatário não* transborda



buffer do lado destinatário

- *Lembrando que TCP é full-duplex, ambos os lados terão buffer de recepção...*
- *OBS: cuidado com* **`rwnd = 0`**

Princípios de controle de congestionamento

congestionamento:

- informalmente: "muitas fontes enviando muitos dados mais rapidamente do que com que a *rede* consegue lidar"
- diferente do controle de fluxo!
- *um problema top-10!*
- manifestações:
  - perda de pacotes (transbordamento de *buffer* nos roteadores)
  - longos atrasos (fila em *buffers* de roteadores)

Abordagens para controle de congestionamento

2 abordagens gerais para controle de congestionamento:

controle de congestionamento fim a fim:

- sem realimentação explícita da rede
- congestionamento inferido de perdas e atrasos nos sistemas finais
- abordagem usada pelo TCP

controle de congestionamento assistido pela rede:

- roteadores provêem realimentação para sistemas finais
  - bit único indicando congestionamento (SNA, DECbit, TCP/IP ECN, ATM)
  - taxa explícita para remetente enviar

Camada de transporte: resumo

- princípios por trás dos serviços da camada de transporte:
  - multiplexação, desmultiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento
- exemplificação, implementação na Internet
  - UDP
  - TCP
- Outros protocolos estão em estudo: DCCP (*Datagram Congestion Control Protocol* – *RFC 4340*), SCTP (Stream Control Transmission Protocol – RFC 4960), ...

a seguir:

- deixamos a "periferia da rede" (camadas de aplicação e transporte)
- vamos para o "núcleo" da rede