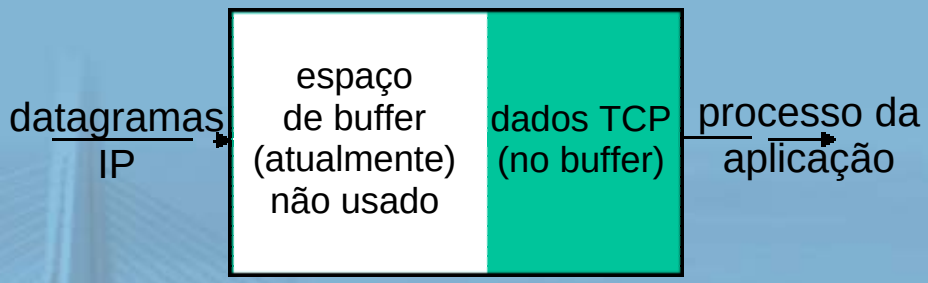
Camada de Transporte (cont.)

- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado para conexão: UDP
- Princípios da transferência confiável de dados
- Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- Princípios de controle de congestionamento
- Congestionamento no TCP

Controle de fluxo TCP

- lado receptor da conexão TCP tem um buffer de recepção:



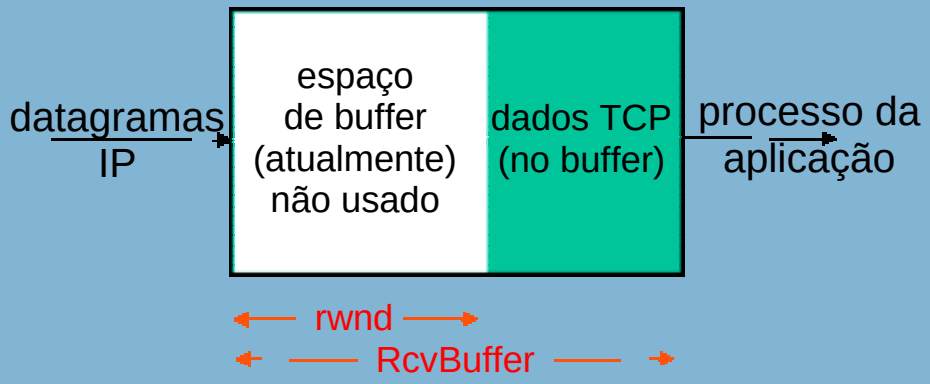
- processo da aplicação pode ser lento na leitura do buffer

controle de fluxo

remetente não estourará buffer do destinatário transmitindo muitos dados muito rapidamente

- *serviço de compatibilização de velocidades:* compatibiliza a taxa de envio do remetente com a de leitura da aplicação receptora

Controle de fluxo TCP: como funciona



- espaço de buffer não usado (janela recepção):

rwnd = RcvBuffer - (LastByteRcvd - LastByteRead)

- destinatário: anuncia espaço de buffer não usado incluindo valor de rwnd no cabeçalho do segmento
- remetente: limita # de bytes sem ACK a rwnd
  - LastByteSent - LastByteAcked
  - garante que buffer do destinatário não estoura
- E qdo rwnd = 0?
  - emissor fica enviando segmentos de 1 byte

Gerenciamento da conexão TCP

lembre-se: Remetente e destinatário TCP estabelecem "conexão" antes que troquem segmentos dados

- inicializa variáveis TCP:
  - #s seq.:
  - buffers, informação de controle de fluxo (p. e. **RcvWindow**)
- *cliente:* inicia a conexão

Socket clientSocket = new Socket("hostname","port #");

- *servidor:* contactado pelo cliente

Socket connectionSocket = welcomeSocket.accept();

apresentação de 3 vias:

etapa 1: cliente envia segmento SYN do TCP ao servidor

- especifica # seq. inicial
- sem dados

etapa 2: servidor recebe SYN, responde com segmento SYNACK

- servidor aloca buffers
- especifica # seq. inicial do servidor

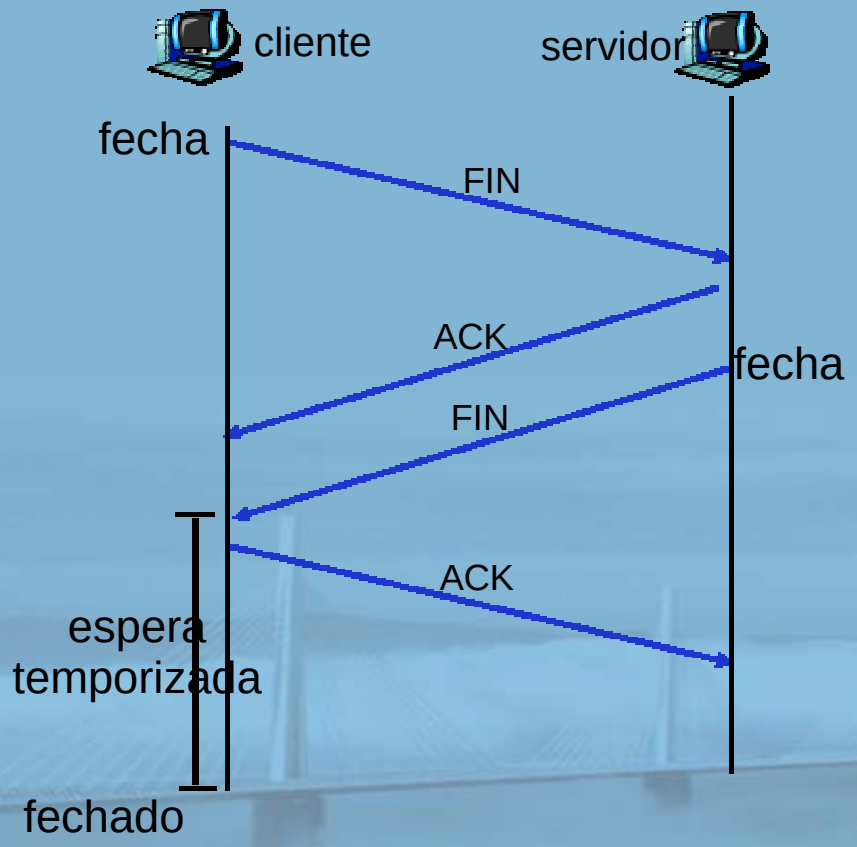
etapa 3: cliente recebe SYNACK, responde com segmento ACK, que pode conter dados

fechando uma conexão:

cliente fecha socket:
clientSocket.close();

etapa 1: sistema final do cliente envia segmento de controle TCP FIN ao servidor

etapa 2: servidor recebe FIN, responde com ACK. Fecha conexão, envia FIN.



etapa 3: cliente recebe FIN, responde com ACK

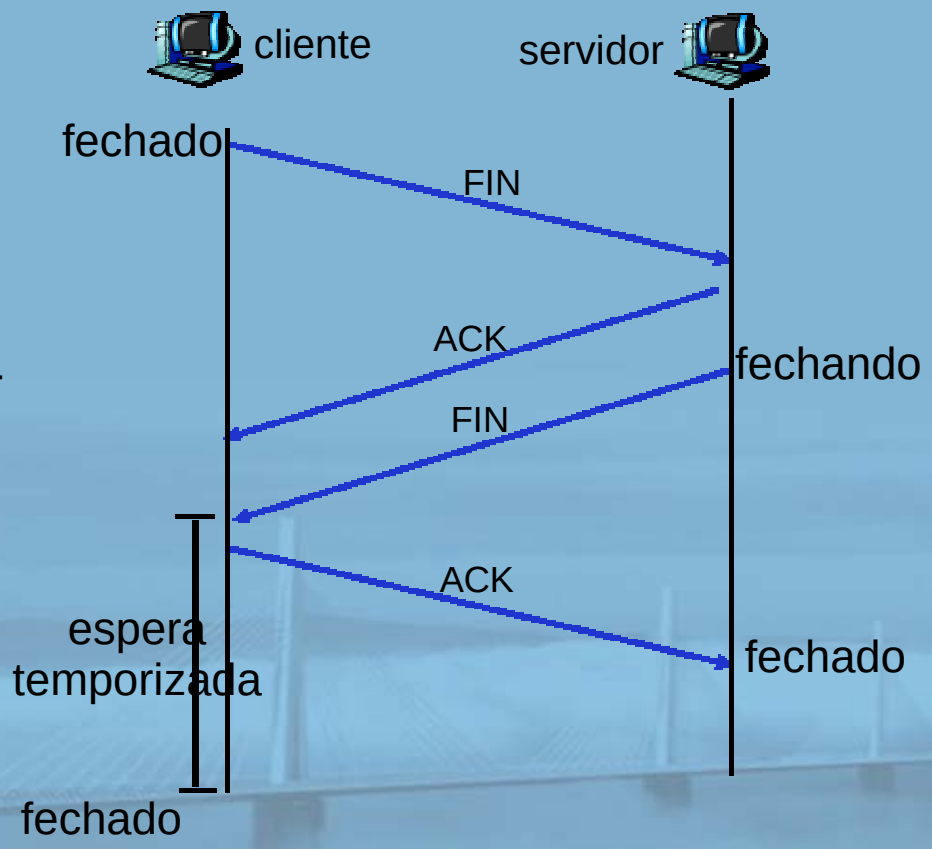
- entra em "espera temporizada" – responderá com ACK aos FINs recebidos

etapa 4: servidor recebe ACK - conexão fechada

Notas: Com pequena modificação, pode tratar de FINs simultâneos.

Espera é para evitar que pacotes atrasados atrapalhem nova conexão na mesma porta

rst = reset



Esboço

- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado para conexão: UDP
- Princípios da transferência confiável de dados
- Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- Princípios de controle de congestionamento
- Congestionamento no TCP

Princípios de controle de congestionamento

Congestionamento:

- informalmente: “muitas fontes enviando muitos dados muito rápido para a *rede* tratar”
  - buffers dos roteadores
- diferente de controle de fluxo!
- manifestações:
  - pacotes perdidos (estouro de buffer nos roteadores)
  - longos atrasos (enfileiramento nos buffers do roteador)
- um dos maiores problemas da rede!

Técnicas para controle de congestionamento

duas técnicas amplas para controle de congestionamento:

controle de congestionamento fim a fim:

- nenhum feedback explícito da rede
- congestionamento deduzido da perda e atraso observados do sistema final
- técnica tomada pelo TCP

controle de congestionamento assistido pela rede:

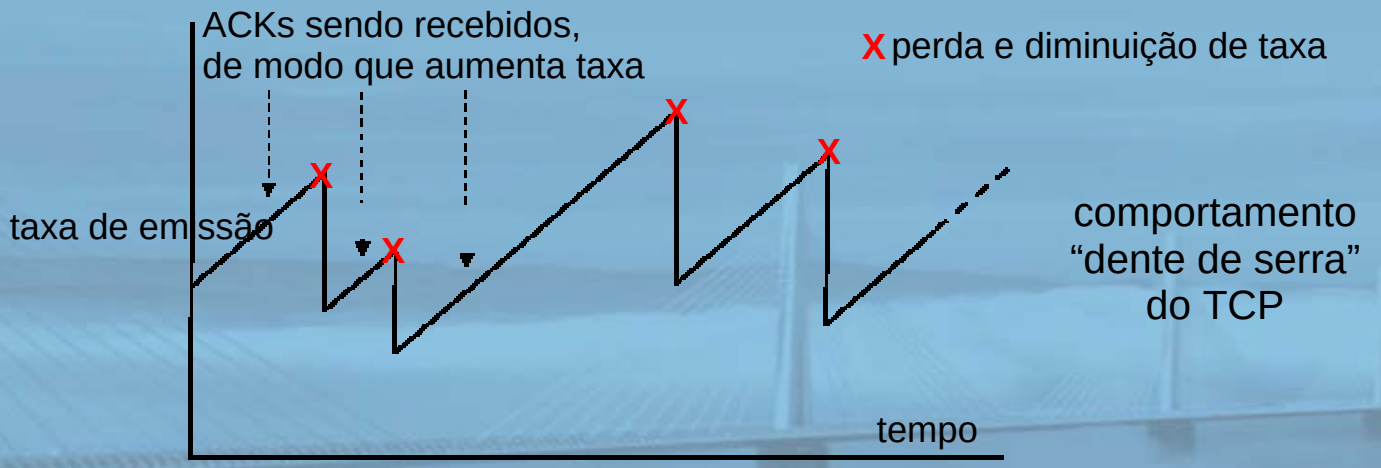
- roteadores oferecem feedback aos sistemas finais
  - desde um único bit indicando congestiona-mento (ex.: TCP/IP ECN-Explicit Congestion Notification - ECE flag)
  - até taxa explícita que o remetente deve enviar no enlace de saída

Esboço

- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado para conexão: UDP
- Princípios da transferência confiável de dados
- Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- Princípios de controle de congestionamento
- Congestionamento no TCP

Controle de congestionamento TCP: busca por largura de banda

- "procura por largura de banda": aumenta taxa de transmissão no recebimento do ACK até por fim ocorrer perda; depois diminui taxa de transmissão
  - continua a aumentar no ACK, diminui na perda (pois largura de banda disponível está mudando, dependendo de outras conexões na rede)



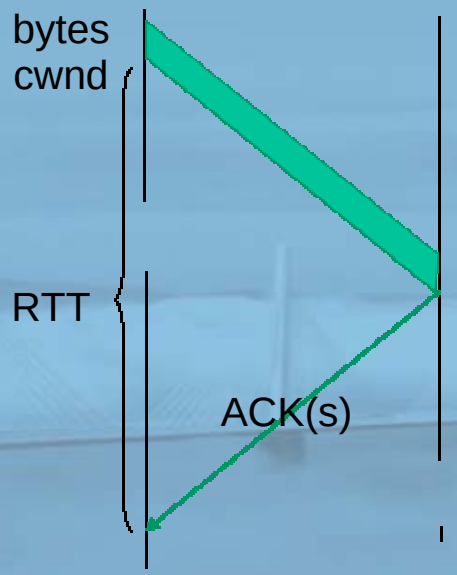
- P: Com que velocidade aumentar/diminuir?
  - detalhes a seguir

Controle de congestionamento TCP: detalhes

- remetente limita taxa limitando número de bytes sem ACK "na pipeline":
  LastByteSent-LastByteAcked < cwnd
- **cwnd:** difere de **rwnd**
- remetente limitado por **min(cwnd,rwnd**)
- aproximadamente,

$$\text{taxa} = \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/seg}$$

- **cwnd** é dinâmico, função do congestionamento de rede percebido



Controle de congestionamento TCP: mais detalhes

evento de perda de segmento: reduzindo **cwnd**

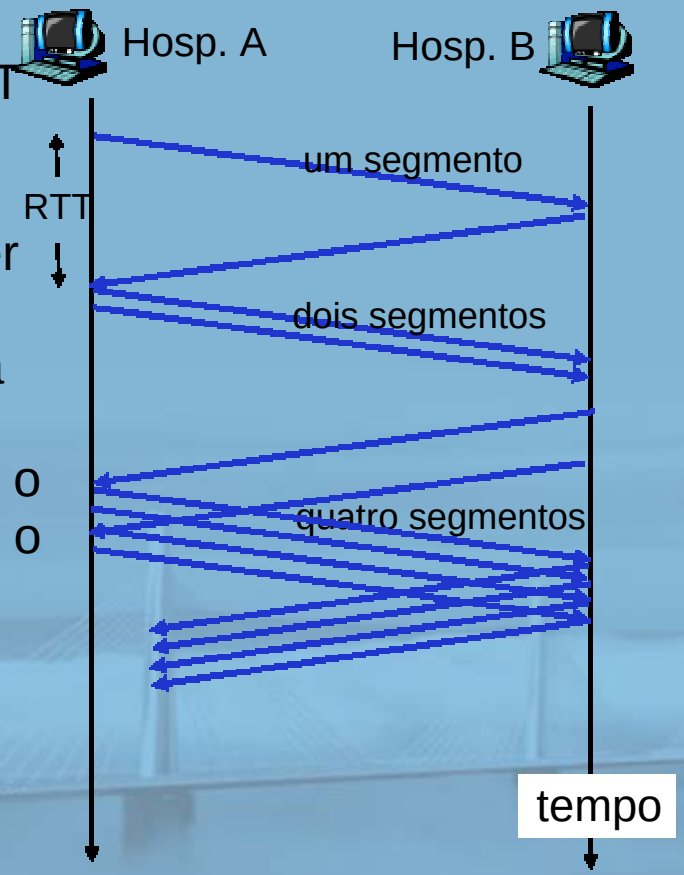
- *timeout*: sem resposta do destinatário
  - corta **cwnd** para 1
- 3 ACKs duplicados: pelo menos alguns segmentos passando (lembre-se da retransmissão rápida)
  - corta **cwnd** pela metade, menos agressivamente do que no *timeout*

ACK recebido: aumenta **cwnd**

- fase de partida lenta:
  - aumento exponencialmente rápido (apesar do nome) no início da conexão, ou após o *timeout*
- prevenção de congestionamento:
  - aumento linear

Partida lenta do TCP

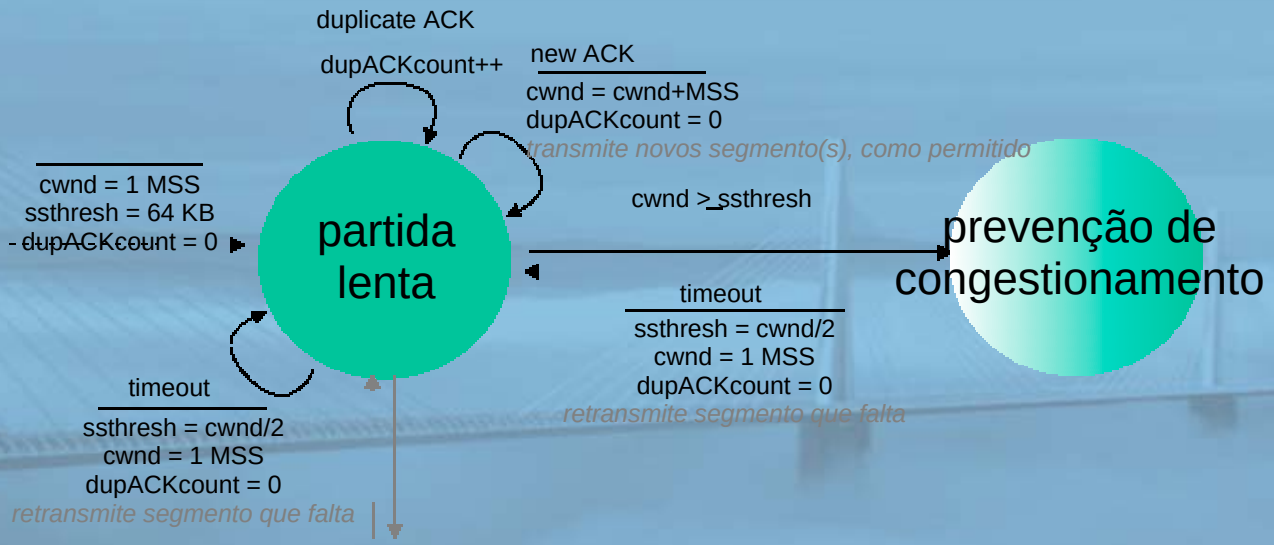
- quando conexão começa, **cwnd** = 1 MSS
  - exemplo: MSS = 500 bytes & RTT = 200 ms
  - taxa inicial = 20 kbps
- largura de banda disponível pode ser >> MSS/RTT
  - desejável subir rapidamente para taxa respeitável
- aumenta taxa exponencialmente até o primeiro evento de perda ou quando o patamar é alcançado
  - **cwnd** duplo a cada RTT
  - feito incrementando **cwnd** por 1 para cada ACK recebido



Transição dentro/fora da partida rápida

ssthresh: patamar de **cwnd** mantido pelo TCP

- um evento de perda: define **ssthresh** como **cwnd/2**
  - lembre-se (metade) da taxa TCP quando ocorreu perda de congestionamento
- quando transição de **cwnd** > = **ssthresh**: da partida lenta para fase de prevenção de congestionamento



TCP: prevenção de congestionamento

- quando **cwnd > ssthresh** cresce **cwnd** de forma linear
  - aumenta **cwnd** em 1 MSS por RTT
  - aborda possível congestionamento mais lento que na partida lenta
  - implementação: **cwnd = cwnd + MSS/cwnd** para cada ACK recebido

AIMD

- ACKs: aumenta **cwnd** em 1 MSS por RTT: aumento aditivo
- perda: corta **cwnd** ao meio (perda sem *timeout* detectado): diminuição multiplicativa

AIMD: Additive Increase Multiplicative Decrease