Princípios de controle de congestionamento

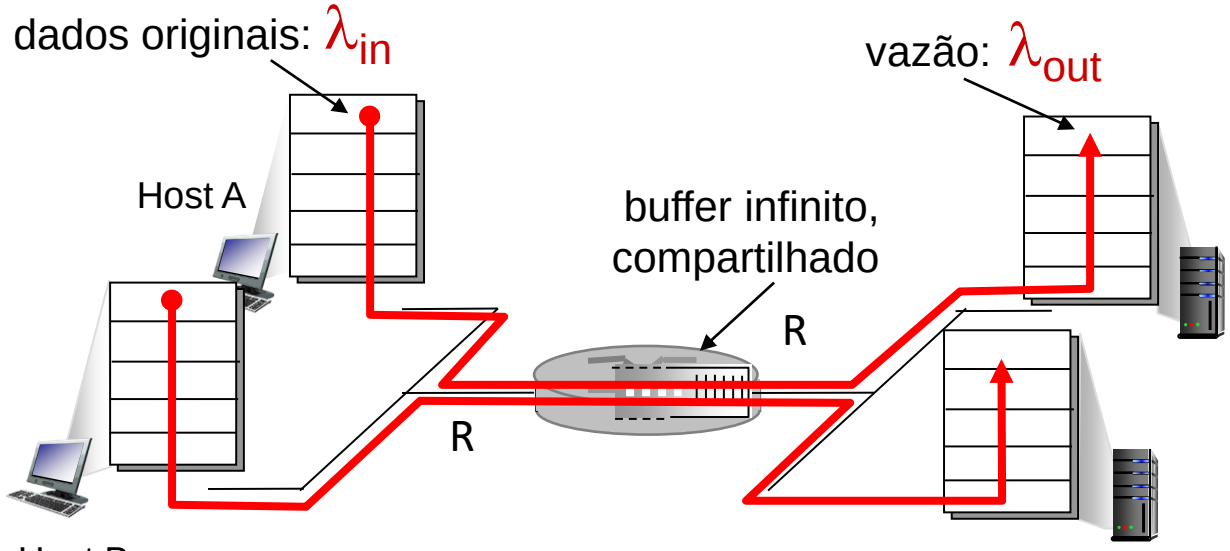
Congestionamento:

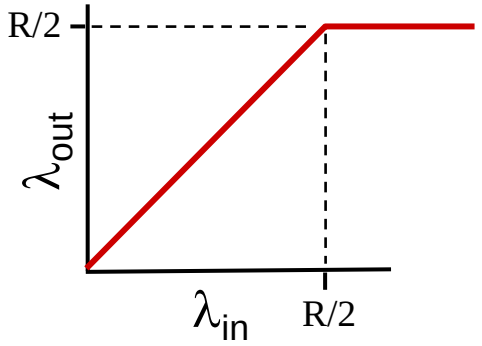
- informalmente: "muitas fontes enviando dados a uma taxa alta, acima da capacidade da ***rede*** de tratá-los"
- diferente de controle de fluxo!
- sintomas:
  - perda de pacotes (saturação de buffer nos roteadores)
  - atrasos grandes (filas nos buffers dos roteadores)
- um dos 10 problemas mais importantes na Internet!

Causas/custos do congestionamento: cenário 1

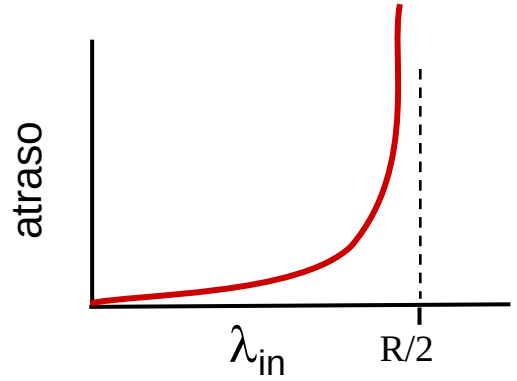
- 2 transmissores, 2 receptores
- Um roteador c/ buffer infinito
- Capacidade dos enlaces de entrada e saída: R
- Sem retransmissão

O que acontece quando a taxa λ_{in} se aproxima de R/2?





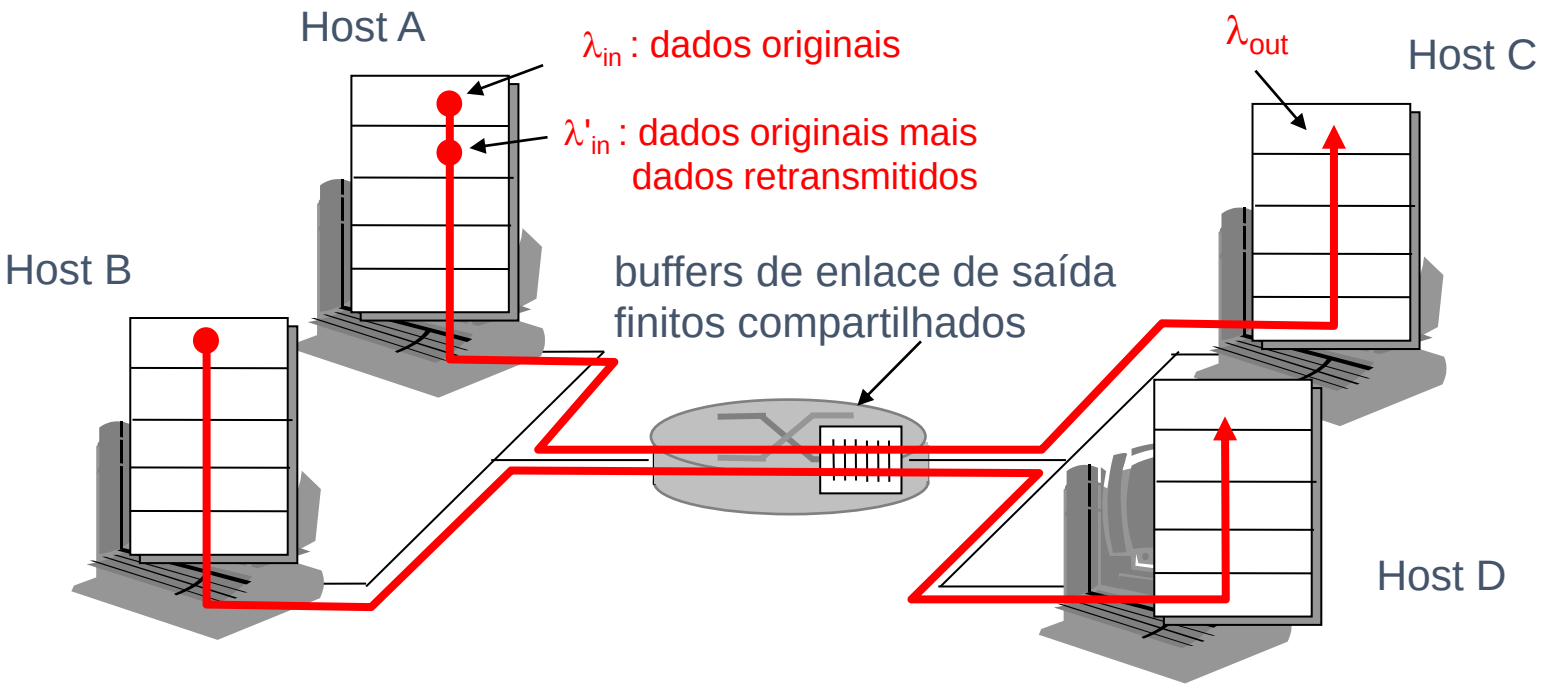
Vazão máxima por conexão: R/2



Grandes atrasos qdo λ_{in} se aproxima da capacidade do enlace

Causas/custos de congestionamento: cenário 2

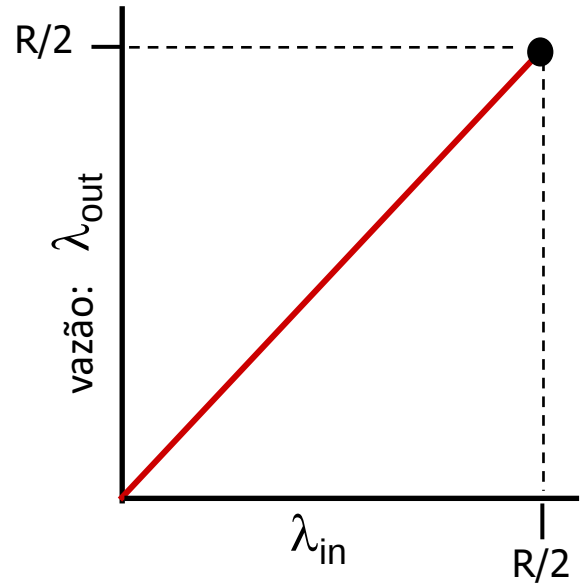
- um roteador, buffers *finitos*
- retransmissão pelo remetente de pacote perdido
  - entrada da camada de aplicação = saída camada apl.: $\lambda_{in} = \lambda_{out}$
  - entrada da camada de transporte inclui retransmissões.: $\lambda'_{in} \geq \lambda_{out}$

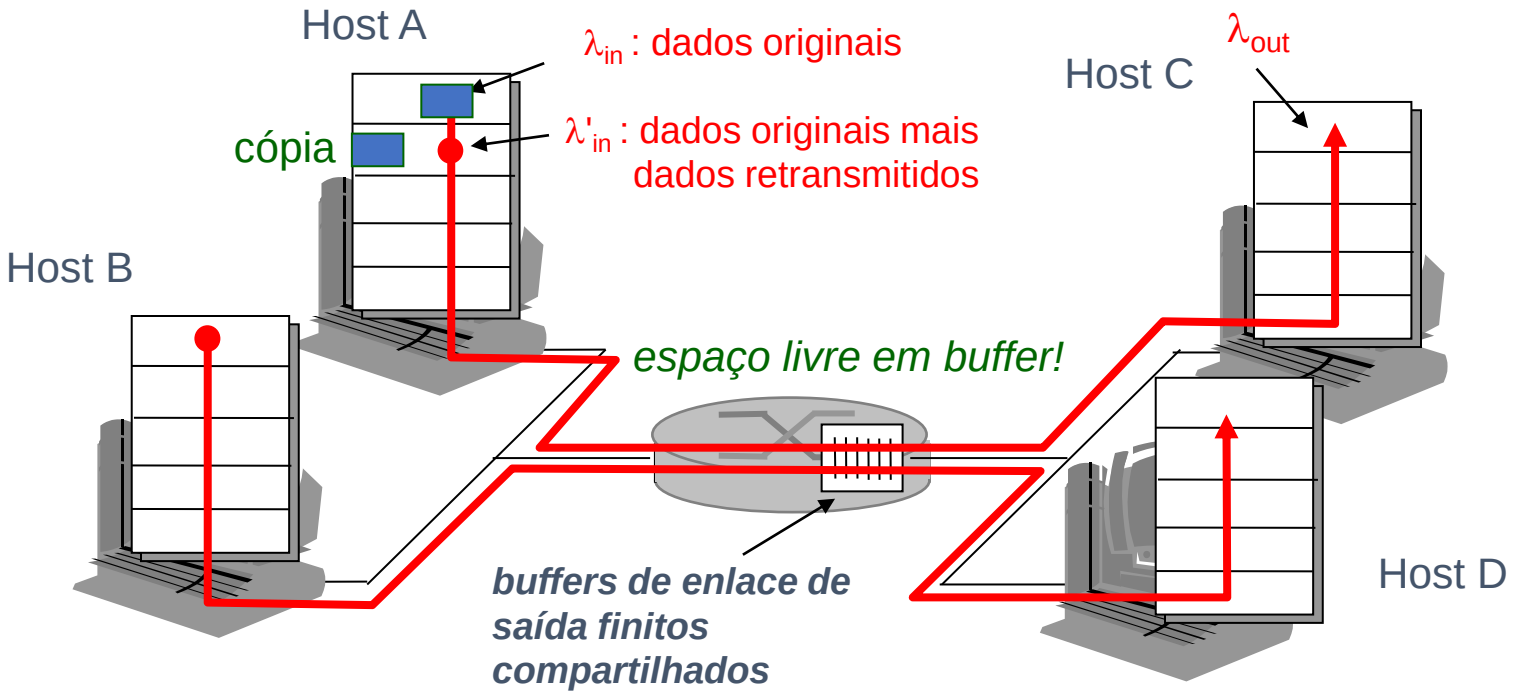


Causas/custos de congestionamento: cenário 2

Idealização: conhecimento perfeito

- transmissor envia apenas quando houver buffer disponível no roteador

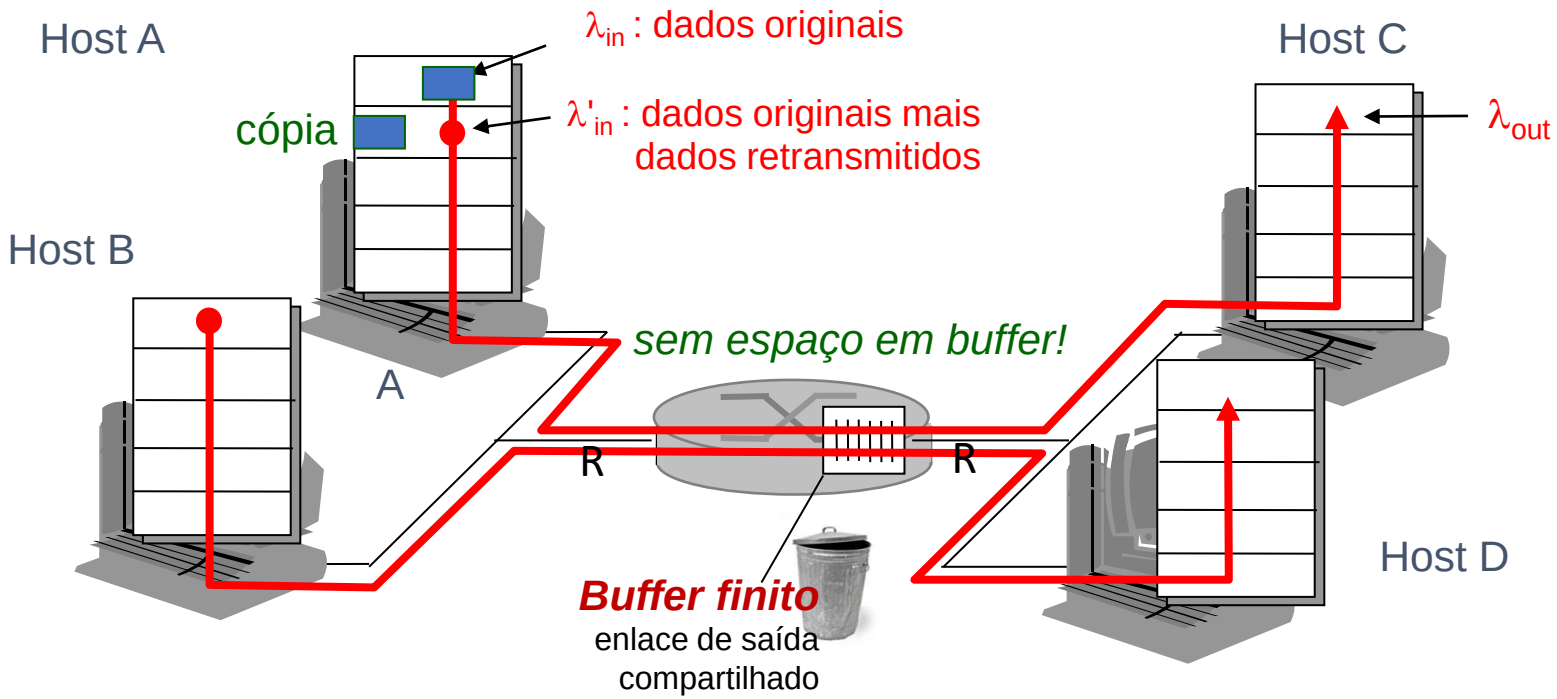




Causas/custos de congestionamento: cenário 2

Idealização: *perda conhecida*

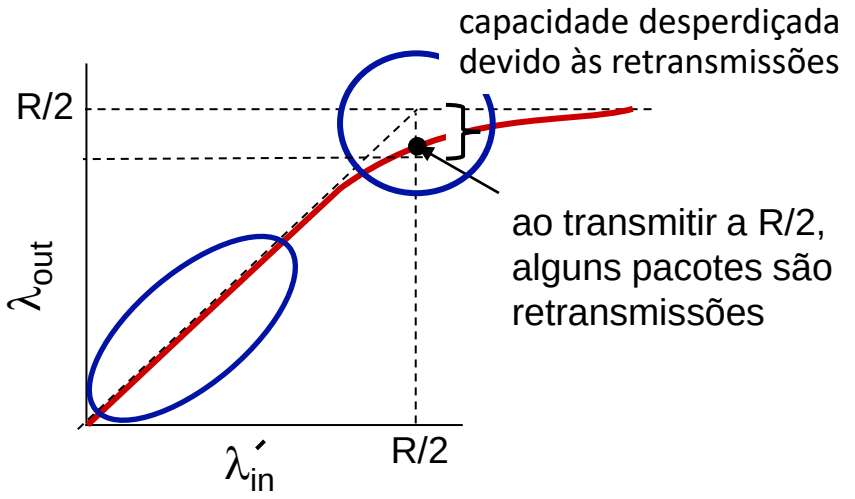
- pacotes podem ser perdidos, descartados no roteador devido a buffers cheios
- transmissor sabe quando o pacote foi descartado: apenas retransmite se o pacote sabidamente se perdeu.

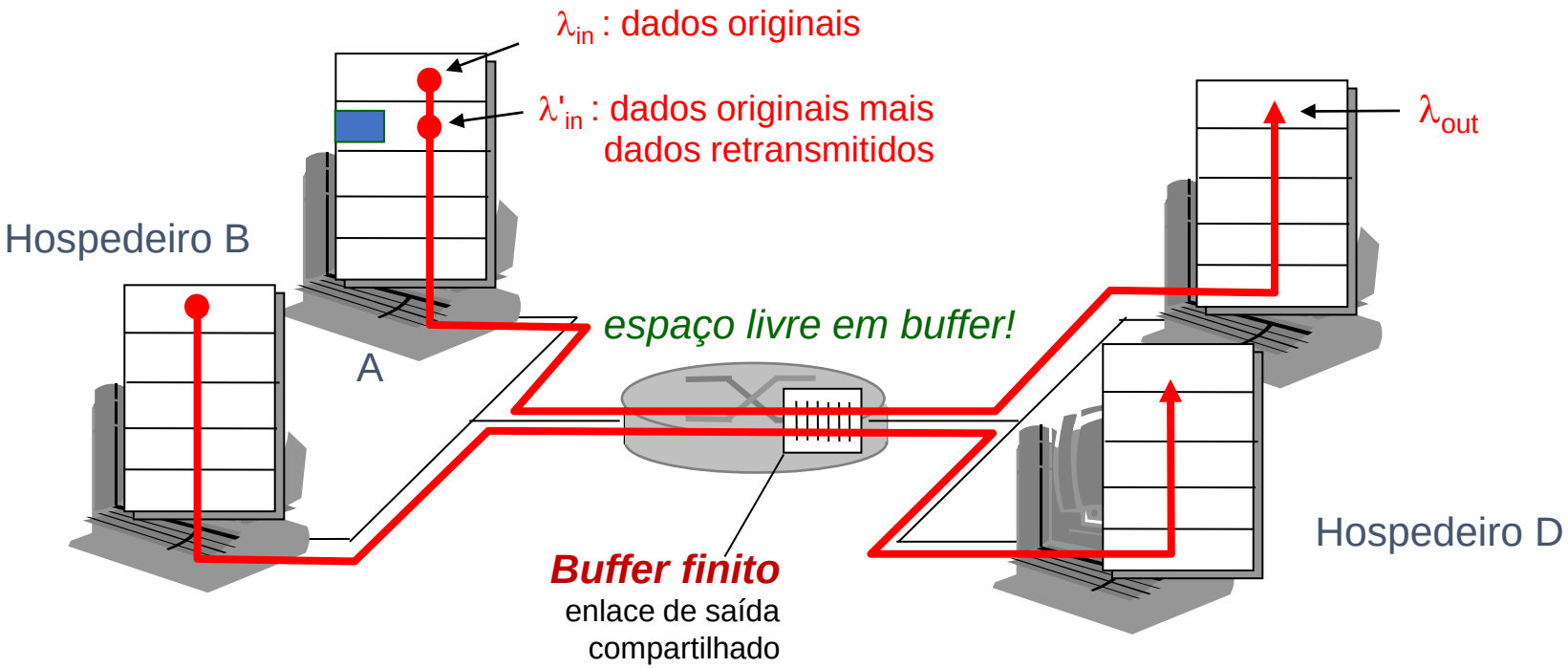


Causas/custos de congestionamento: cenário 2

Idealização: *perda conhecida*

- pacotes podem ser perdidos, descartados no roteador devido a buffers cheios
- transmissor sabe quando o pacote foi descartado: apenas retransmite se o pacote sabidamente se perdeu.

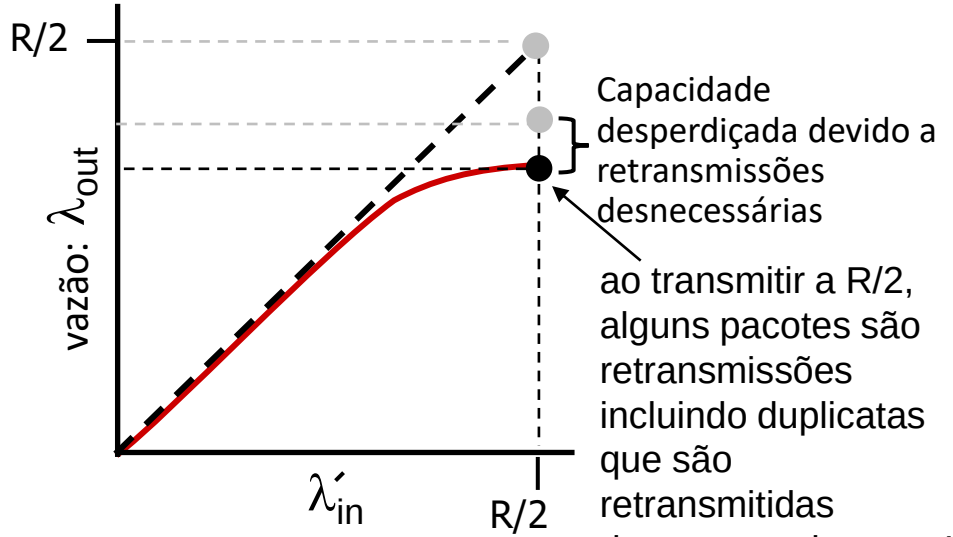


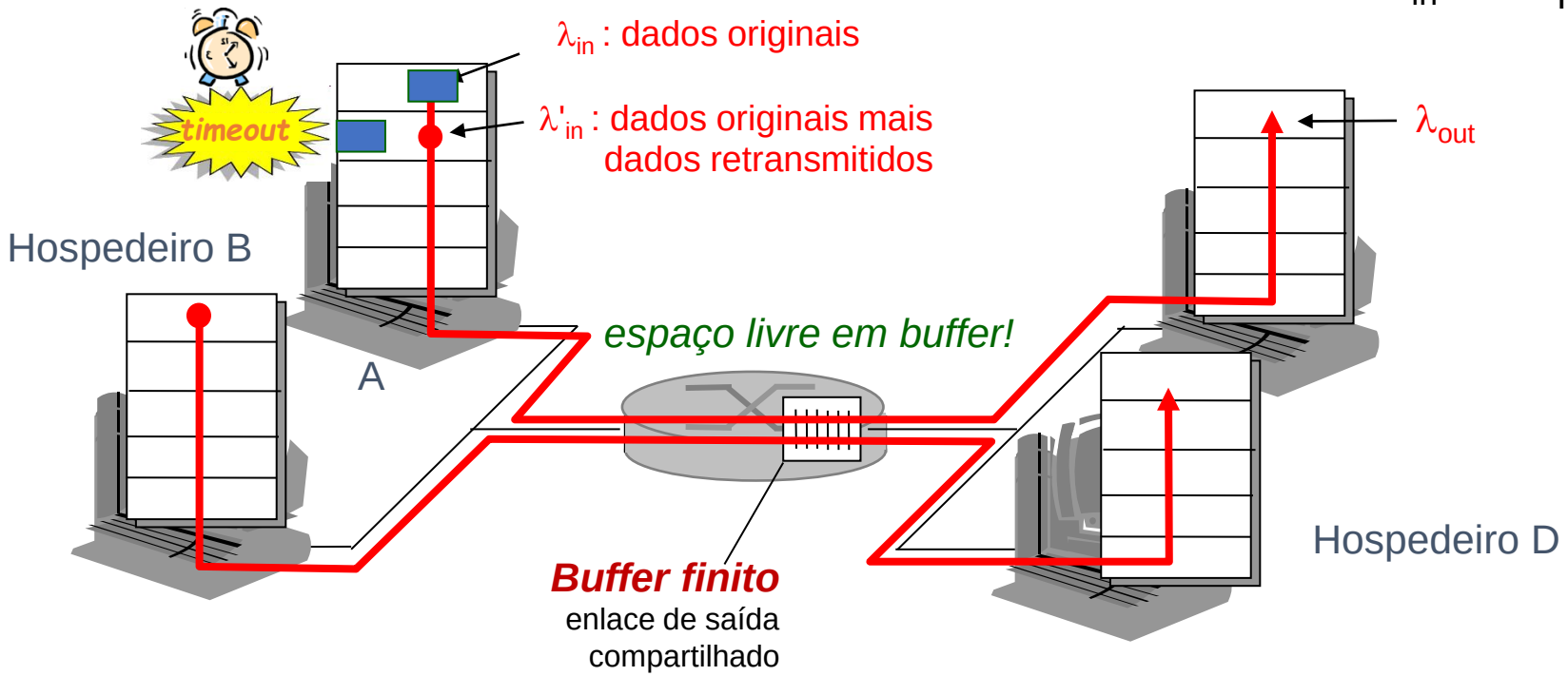


Causas/custos de congestionamento: cenário 2

Cenário real: *duplicatas*

- pacotes podem ser perdidos, descartados no roteador devido a buffers cheios: necessidade de retransmissões
- retransmissão prematura (timeout): envio de ***duas*** cópias, uma desnecessária





Causas/custos de congestionamento: cenário 2

Cenário real: *duplicatas*

- pacotes podem ser perdidos, descartados no roteador devido a buffers cheios
- retransmissão prematura (timeout), envio de ***duas*** cópias

"custos" do congestionamento:

- Mais trabalho (retransmissões) devido ao esgotamento do buffer no roteador
- Retransmissões desnecessárias: enlace transporta múltiplas cópias do pacote devido a uma temporização prematura

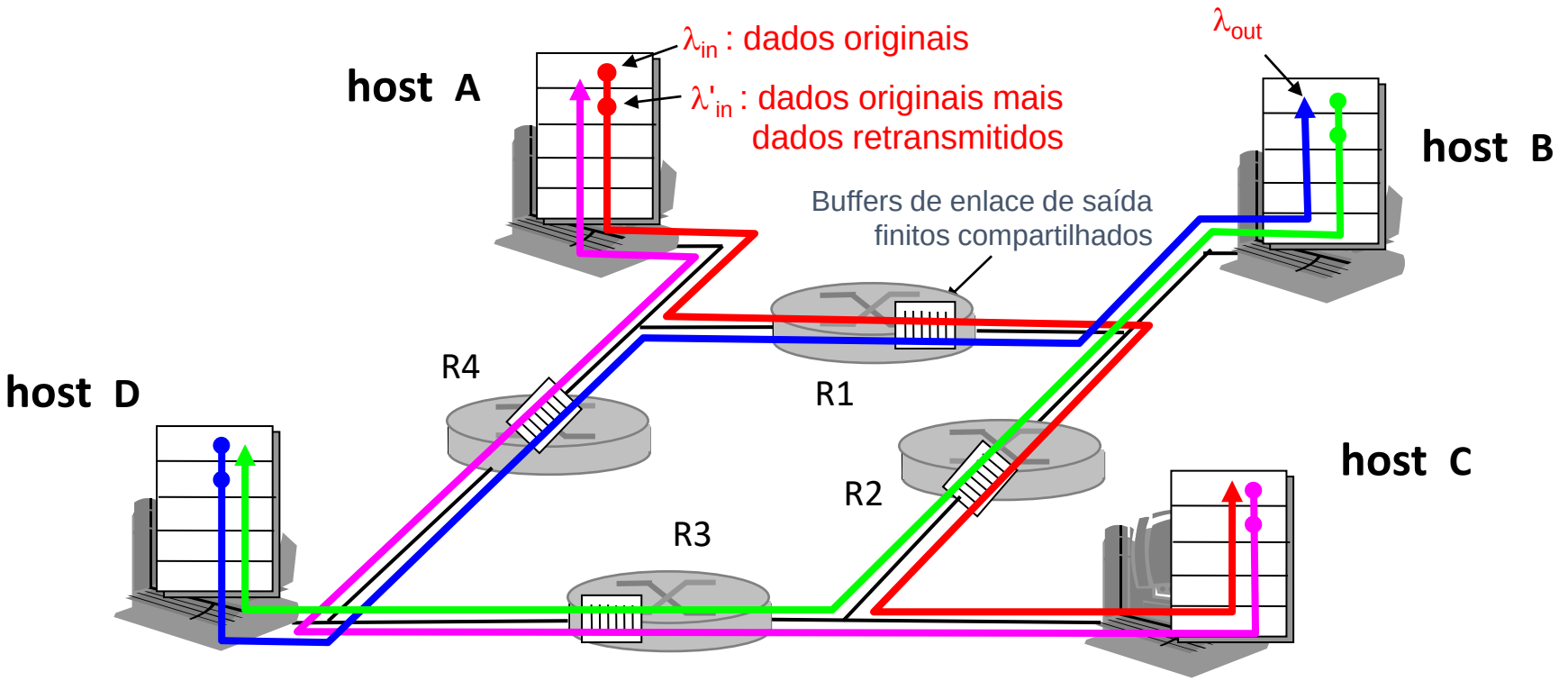
  ⇒ ***diminuindo a vazão máxima alcançável***

Causas/custos de congestionamento: cenário 3

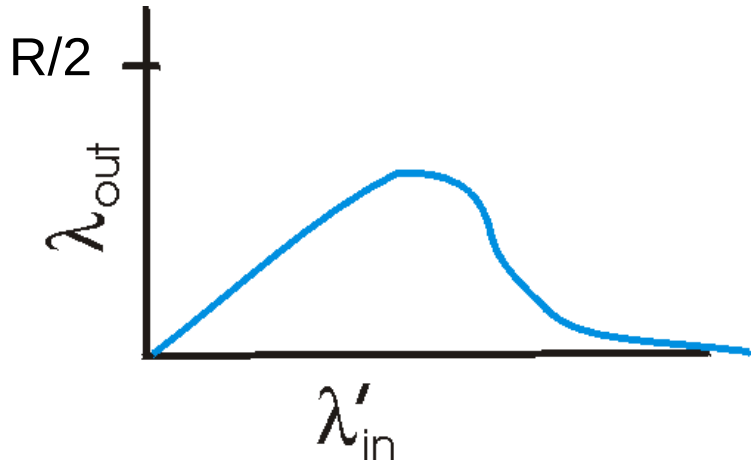
- quatro remetentes
- caminhos com múltiplos enlaces
- temporização/retransmissão

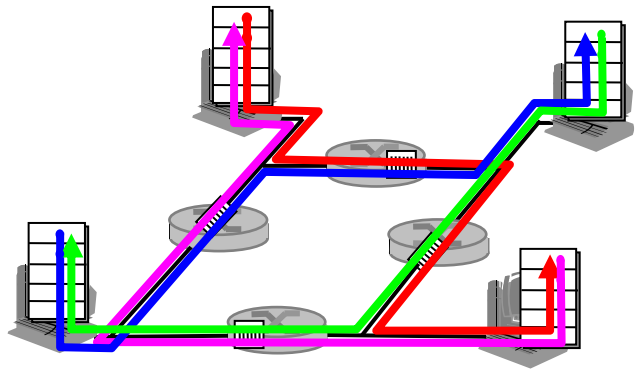
Q: O que acontece quando λ_{in} e λ'_{in} crescem ?

R: qdo λ'_{in} cresce, todos os pacotes da conexão azul na fila de R1 são descartados e a vazão desta conexão → 0



Causas/custos de congestionamento: cenário 3



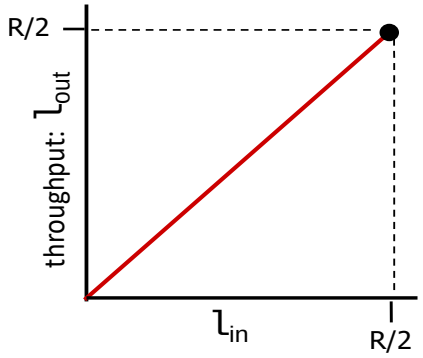


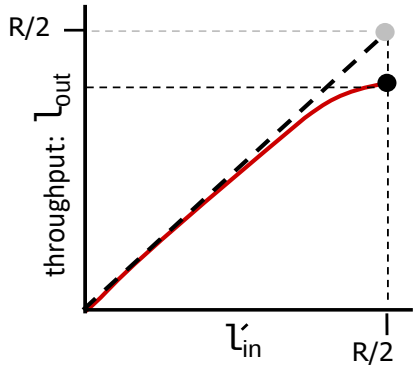
Outro "custo" de congestionamento:

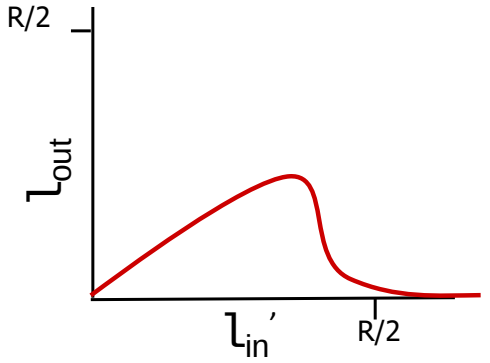
quando pacote é descartado, qualquer capacidade de transmissão e de armazenamento que já foi usada para este pacote, antes do descarte, é desperdiçada!

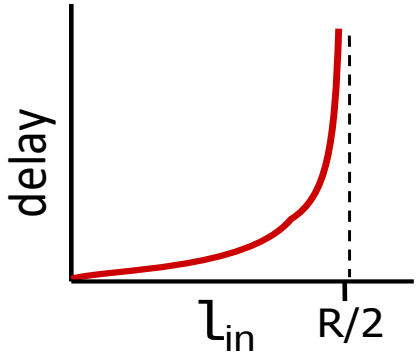
Causas/custos de congestionamento: resumo

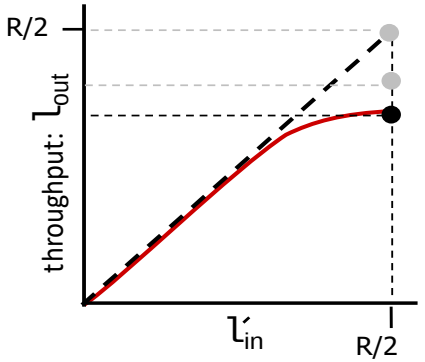
- Vazão nunca pode exceder a capacidade do enlace
- Atraso aumenta qdo nos aproximamos da capacidade do enlace
- Perda/retransmissões reduzem a vazão efetiva
- Duplicatas reduzem ainda mais a vazão efetiva
- Capacidade de transmissão ou armazenamento em roteadores anteriores é desperdiçada quando um pacote é perdido em roteadores mais adiante











Abordagens de controle de congestionamento

Controle de congestionamento fim-a-fim:

- Rede não fornece nenhuma realimentação
- Congestionamento intuído pelos *sistemas finais* com base nas perdas e atrasos observados
- Adotado pelo TCP

Controle de congestionamento assistido pela rede:

- Roteadores enviam informações aos *sistemas finais*
  - bit indicando o nível de congestionamento (Ex.: protocolos TCP ECN, ATM, DECbit)
  - pode indicar explicitamente a taxa de transmissão
- Roteador informa explicitamente ao remetente sobre o estado de congestionamento da rede

TCP: controle de congestionamento

- TCP obriga cada remetente a limitar a taxa à qual enviam tráfego para sua conexão como uma função do congestionamento da rede percebido.

- Se um remetente TCP perceber que há pouco congestionamento no caminho entre ele e o destinatário, aumentará sua taxa de transmissão.

- Se perceber que há congestionamento, reduzirá sua taxa de transmissão.

TCP: controle de congestionamento

Mas essa abordagem levanta três questões:

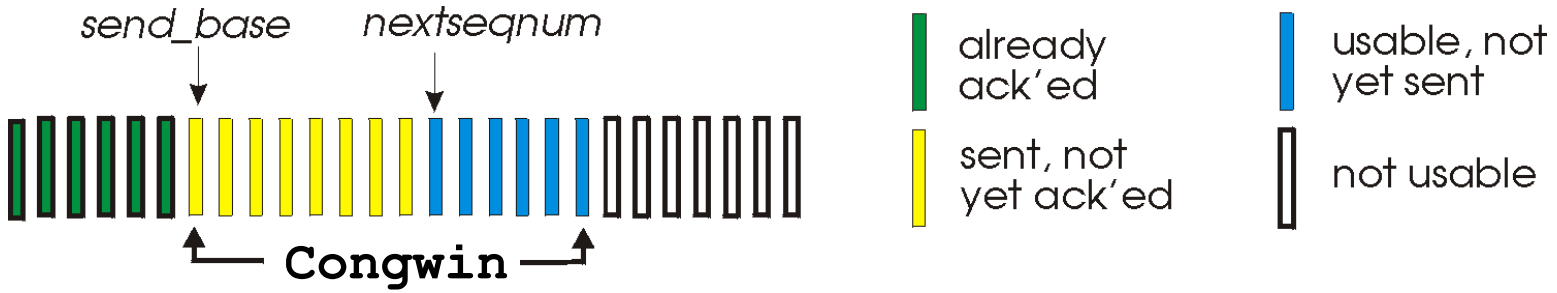
1. Como um remetente TCP limita a taxa pela qual envia tráfego para sua conexão?

2. Como um remetente TCP percebe que há congestionamento entre ele e o destinatário?

3. Que algoritmo o remetente deve utilizar para modificar sua taxa de transmissão como uma função do congestionamento fim a fim percebido?

TCP: controle de congestionamento

- Taxa de transmissão limitada pelo tamanho da janela de congestionamento, **CongWin**
- **CongWin** é ajustada dinamicamente em resposta ao congestionamento observado



- **CongWin** bytes enviados em um RTT, espera Acks e envia mais bytes:

$$\text{taxa permitida} = \frac{\texttt{Congwin}}{\text{RTT}} \text{ Bytes/seg}$$

TCP: controle de congestionamento

Como o transmissor detecta o congestionamento?

Resposta:

Evento de perda ⇒ esgotamento de temporização ou 3 ACKs duplicados

Transmissor TCP reduz a taxa (**CongWin**) após o evento de perda

Recebimento de ACKs não-duplicados ⇒ aumento da CongWin

ALGORITMO DE CONTROLE DE CONGESTIONAMENTO DO TCP

Três mecanismos:

- Reação a eventos de esgotamento de temporização (timeout) ou 3 ACKs
- Partida lenta
- AIMD (aumento aditivo, diminuição multiplicativa)

TCP: controle de congestionamento

"sondagem" para banda utilizável:

- idealmente: transmitir o mais rápido possível (**Congwin** no máximo possível) sem perder pacotes
- aumentar **Congwin** até perder pacotes (congestionamento)
- ocorreu perda: *diminui* **Congwin**, depois volta à "sondagem" aumentando novamente **Congwin**

duas "fases"

- *partida lenta*
- *congestion avoidance (recuperação rápida)*

variáveis importantes:

- **Congwin**
- **threshold**: define limiar entre fases de partida lenta e *congestion avoidance*

TCP Partida lenta (Slow Start)

Durante a fase inicial:

- quando a conexão começa, **CongWin** = 1 MSS (tamanho máximo do segmento)

  Exemplo: MSS = 500 bytes e RTT = 200 milissegundos

  Taxa inicial = 20 kbps

- largura de banda disponível pode ser muito maior que 1 MSS/RTT

  ⇒ Desejável aumentar rapidamente até uma taxa que não ocasione perdas

Quando a conexão começa, a *taxa aumenta rapidamente de modo exponencial* até a ocorrência do primeiro evento de perda

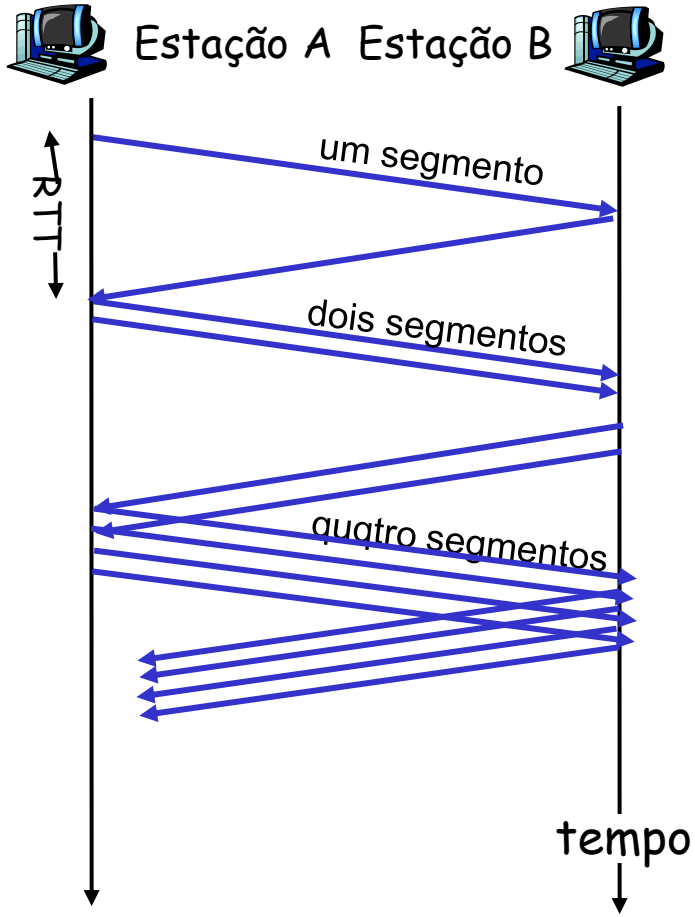
Taxa inicial lenta (por isso o nome "partida lenta"), mas o aumento é exponencial

TCP Partida lenta (Slow Start)

Algoritmo Partida Lenta

inicializa: Congwin = 1
for (cada segmento c/ ACK)
    Congwin++
until (evento de perda OU
    CongWin > threshold)

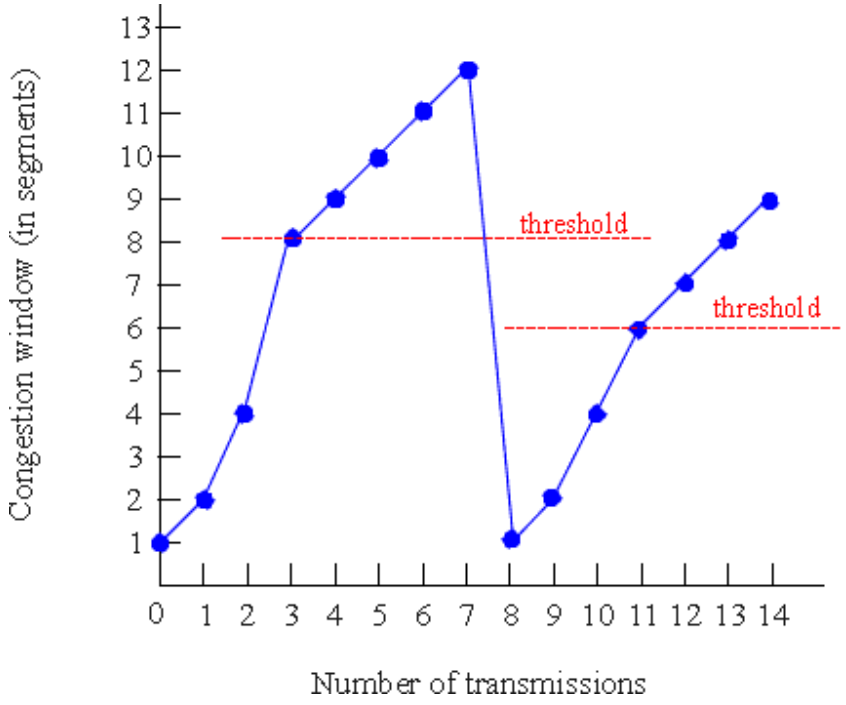
- aumento exponencial (por RTT) no tamanho da janela, apesar da taxa inicial ser *"lenta"*
- evento de perda: temporizador (Tahoe TCP) e/ou três ACKs duplicados (Reno TCP)



TCP Cogestion Avoidance (recuperação rápida)

Congestion avoidance

```
/* partida lenta acabou */
/* Congwin > threshold */
Repita {
  cada w segmentos
reconhecidos:
       Congwin++
  } até ocorrer uma perda
threshold = Congwin/2
Congwin = 1
faça partida lenta
```



Refinamento

Após 3 ACKs duplicados:

- **CongWin** é cortado pela metade
- Janela então cresce linearmente

Após evento de esgotamento de temporização:

- **CongWin** é ajustado para 1 MSS;
- A janela então cresce exponencialmente **até CongWin/2 (threshold)** e, a partir daí, cresce linearmente

Filosofia:

- 3 ACKs duplicados indicam que a rede é capaz de entregar alguns segmentos
- Esgotamento de temporização antes dos 3 ACKs duplicados é “mais alarmante”

Resumo: controle de congestionamento TCP

- Quando **CongWin** está abaixo do limite (**Threshold)**, transmissor em fase de ***partida lenta***, janela cresce exponencialmente.

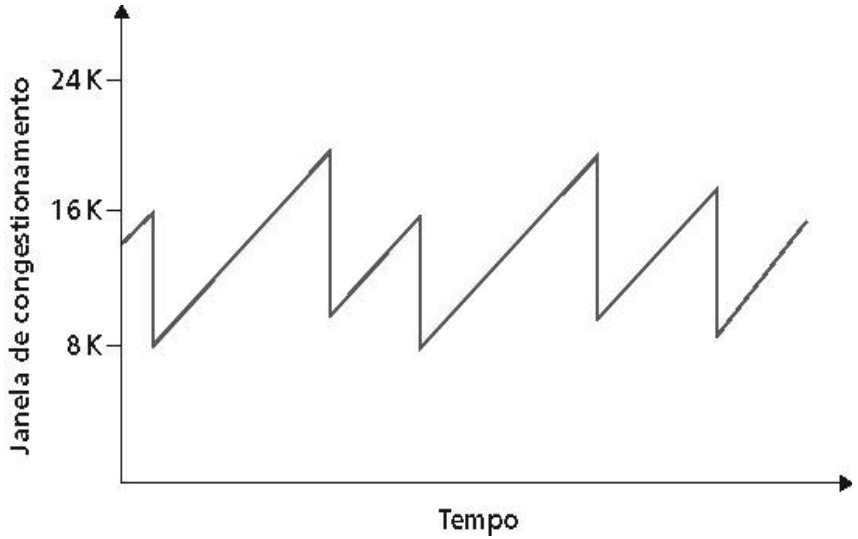
- Quando **CongWin** está acima do limite (**Threshold)**, o transmissor entra em fase de ***congestion-avoidance***, janela cresce linearmente.

- Quando ocorrem ***três ACK duplicados***, o limiar (**Threshold**) é ajustado em **CongWin/2** e **CongWin** é ajustado para **Threshold**.

- Quando ocorre ***esgotamento de temporização***, o **Threshold** é ajustado para **CongWin/2** e o **CongWin** é ajustado para 1 MSS e inicia-se a fase de partida lenta.

TCP: AIMD

- Aumento aditivo: aumenta o **`CongWin`** com 1 MSS a cada RTT na ausência de eventos de perda
- Redução multiplicativa: diminui o **`CongWin`** pela metade após o evento de perda



Conexão TCP de longa-vida:

O controle de congestionamento AIMD faz surgir o comportamento semelhante a "dentes de serra":

TCP: AIMD

Redução multiplicativa:

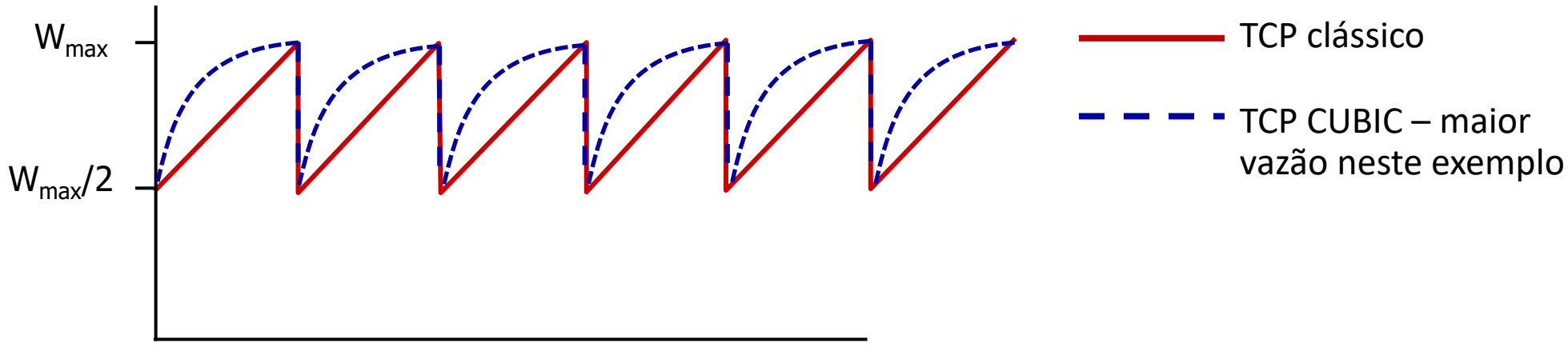
- Taxa de transmissão é diminuída pela metade quando detectada uma perda por **3 ACKs duplicados** (TCP Reno pula a fase de partida lenta)
- Taxa de transmissão é diminuída para 1 MSS quando a perda é detectada por ***timeout*** (TCP Tahoe)

Por que AIMD?

- AIMD – algoritmo distribuído e assíncrono:
  - Otimiza as taxas de transmissão numa rede congestionada!
  - Tem propriedades de estabilidade

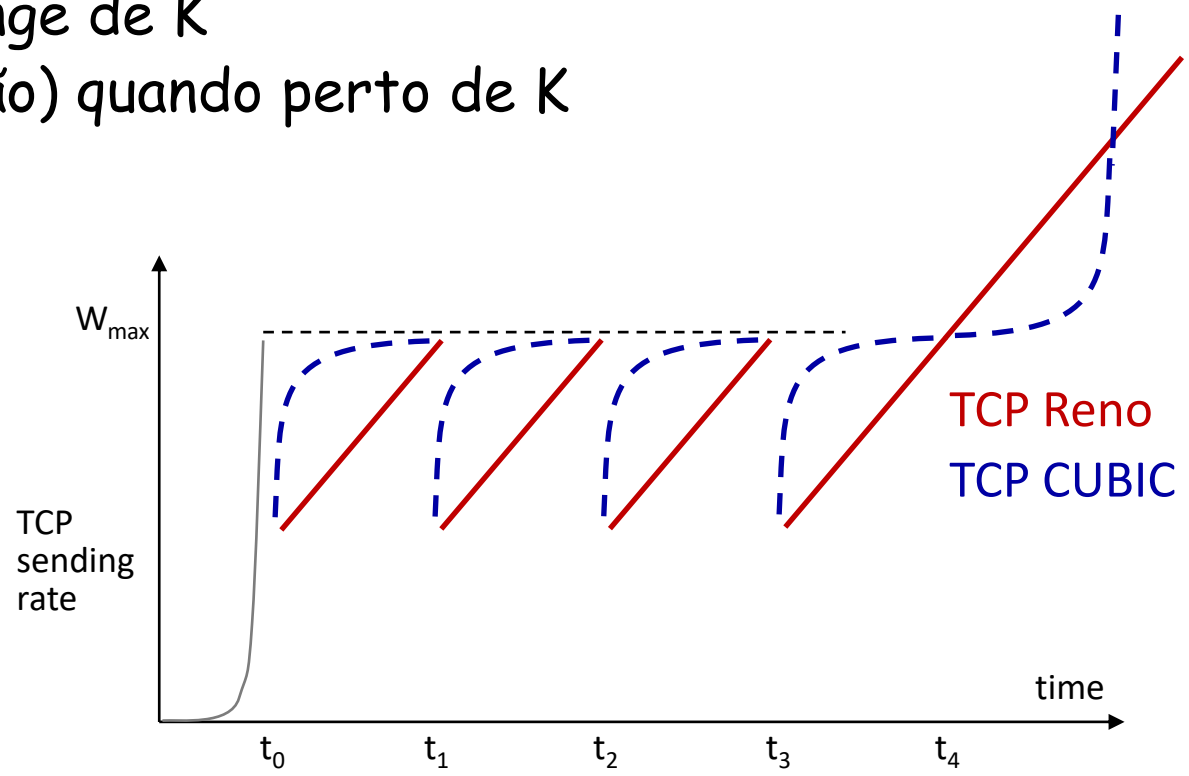
TCP CUBIC

- Existe uma maneira melhor do que o AIMD para "sondar" qual a taxa de transmissão apropriada?
- Intuição:
  - W_{max}: taxa de transmissão para a qual uma perda foi detectada
  - Após cortar o tam. da janela pela metade no momento da perda, aumenta-se mais rapidamente até próx. de W_{max} e mais lentamente ao se aproximar de W_{max}



TCP CUBIC

- K: instante de tempo no qual o tamanho da janela vai atingir W_{max}
  - K é ajustável
- Aumenta-se W como função do *cubo* da diferença entre o tempo atual e K
  - Maiores aumentos qdo longe de K
  - Menor aumento (precaução) quando perto de K
- Padrão no Linux e nos servidores Web mais populares



TCP - Justiça

Lembrar que:

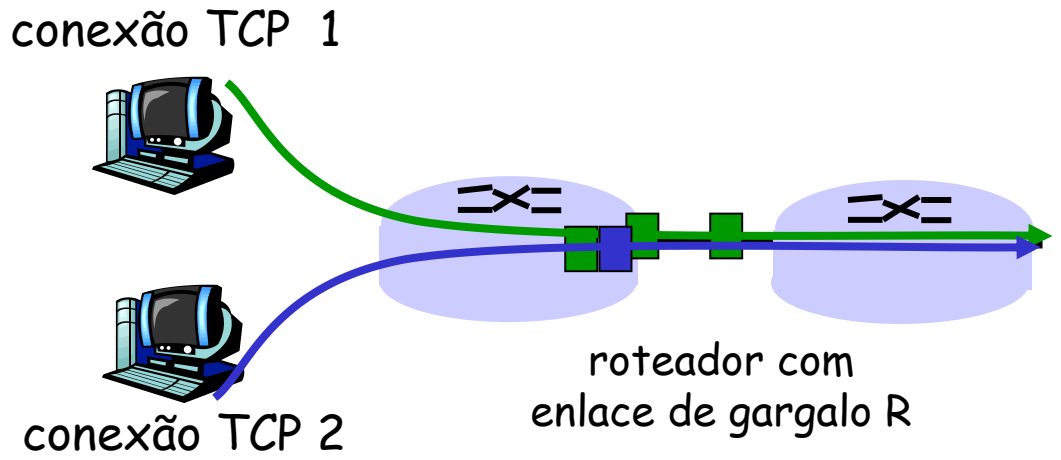
TCP Congestion Avoidance:

AIMD: *aumento aditivo, redução multiplicativa*

- aumenta a janela de 1 a cada RTT
- diminui a janela por um fator de 2 em caso de evento perda

TCP - Justiça

Objetivo: se N sessões TCP devem passar pelo mesmo gargalo, cada uma deve obter 1/N da capacidade do enlace

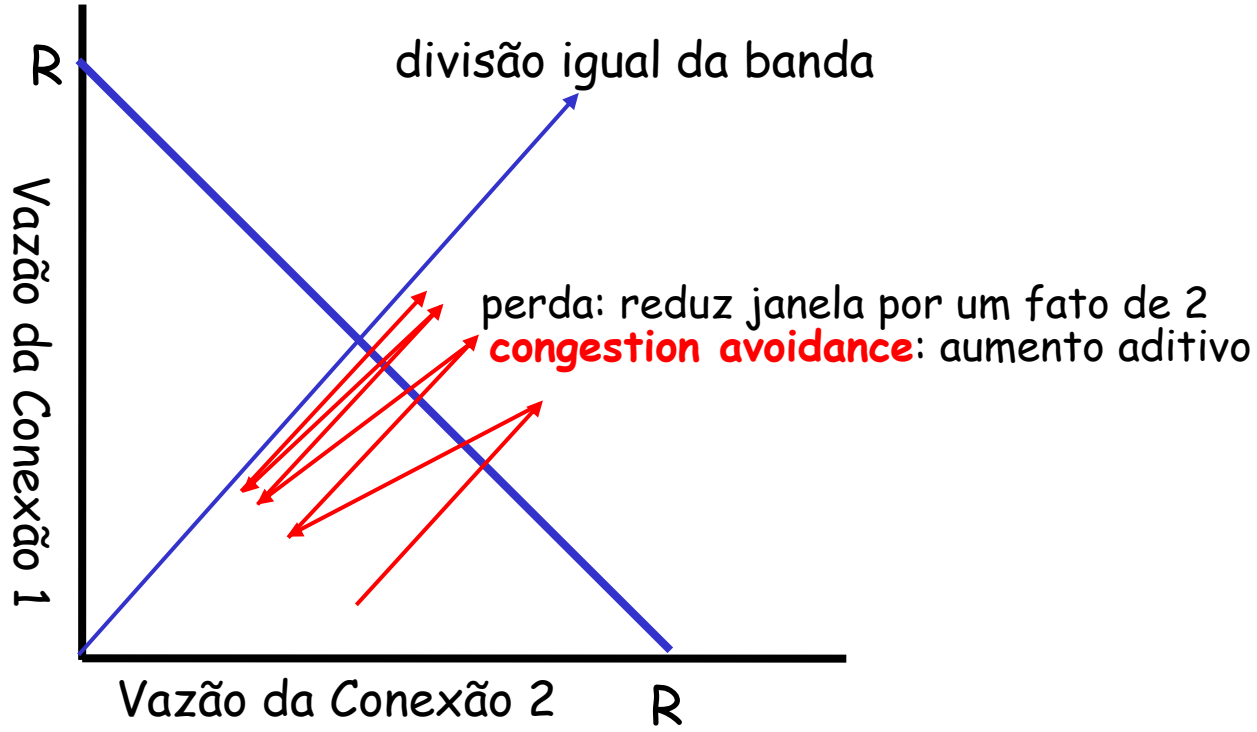


Suposições: 1. conexões têm o mesmo MSS e RTT
2. ambas com grande quantidade de dados a enviar
3. nenhuma outra conexão divide o enlace
4. ignorar partida lenta

Porque o TCP é justo?

Duas sessões competindo pela banda:

- O aumento aditivo fornece uma inclinação de 1, quando a vazão aumenta
- redução multiplicativa diminui a vazão proporcionalmente



Justiça (mais)

Justiça e UDP:

- aplicações multimídia frequentemente não usam TCP
  - não querem a taxa estrangulada pelo controle de congestionamento
- Preferem usar o UDP:
  - transmite áudio/vídeo a taxas constantes
  - toleram perdas de pacotes

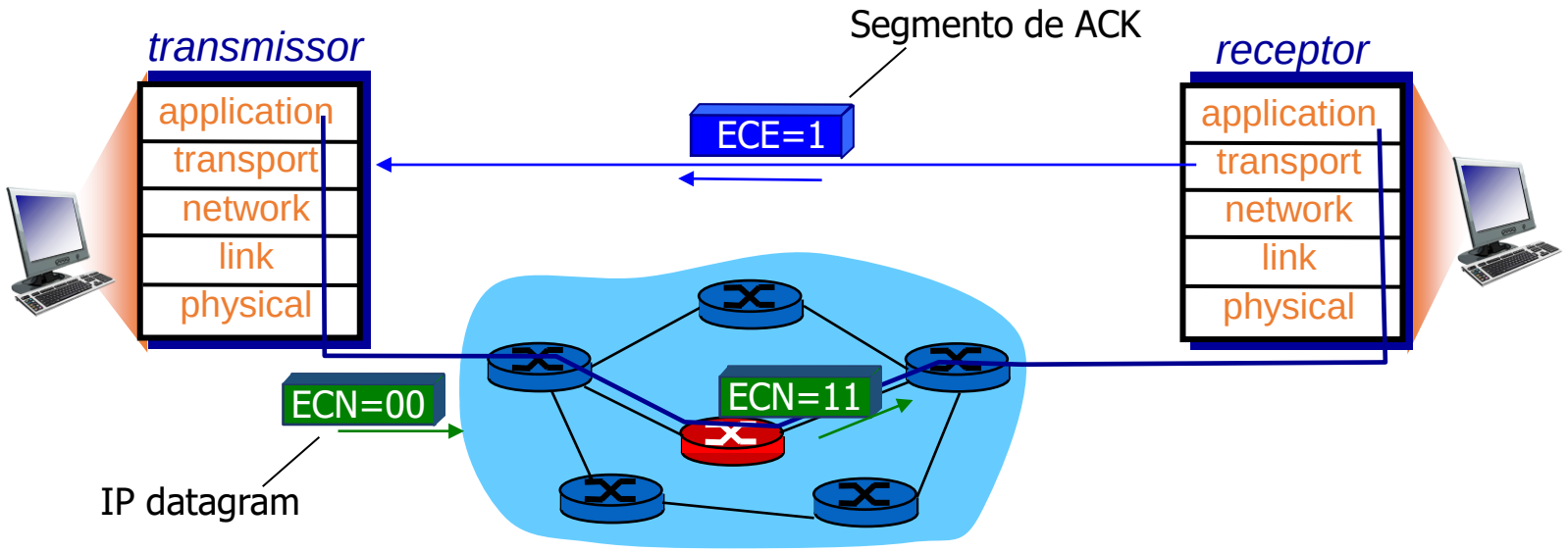
Justiça e conexões TCP em paralelo:

- nada impede que as aplicações abram conexões paralelas entre 2 hosts
- Os *browsers* Web fazem isto
- Exemplo: enlace com taxa R compartilhado por 9 conexões:
  - novas aplicações pedem 1 TCP, obtém taxa de R/10
  - novas aplicações pedem 11 TCPs, obtém taxa R/2!

Notificação Explícita de Congestionamento (ECN)

Controle de congestionamento assistido pela rede

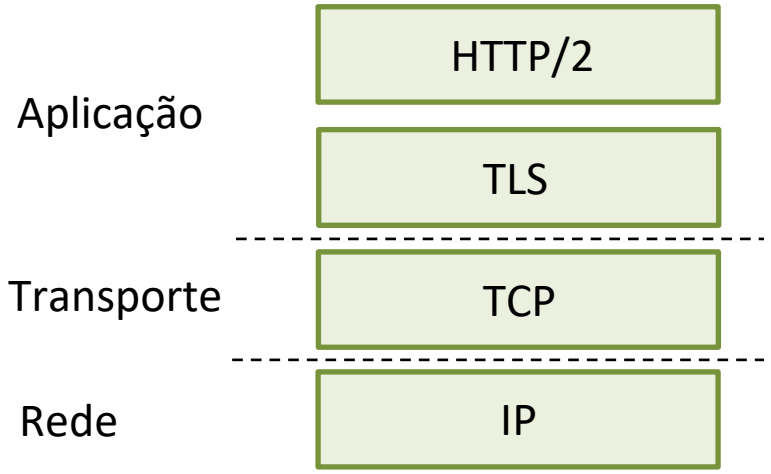
- dois bits no cabeçalho do datagrama IP (campo ToS) são marcados pelo roteador para indicar congestionamento
- indicação de congestionamento é carregada para o recptor
- receptor (vendo a indicação de congestionamento no datagrama IP) marca o bit ECE no segmento de ACK enviado ao transmissor para notificá-lo do congestionamento
- envolve tanto o datagrama IP (bit ECN marcado no datagrama IP) quanto o segmento TCP (bits C, E marcados no cabeçalho do segmento TCP)



QUIC: Quick UDP Internet Connections

Protocolo da camada de aplicação, em cima do UDP

- Aumenta o desempenho dos serviços de transporte para o HTTP Seguro
- Implantado em muitos servidores da Google, apps (Chrome, mobile YouTube, busca p/ Android)
- Mais de 7% do tráfego da Internet (2020)
- Usa transferência confiável de dados, controle de congestionamento, gerenciam. de conexão

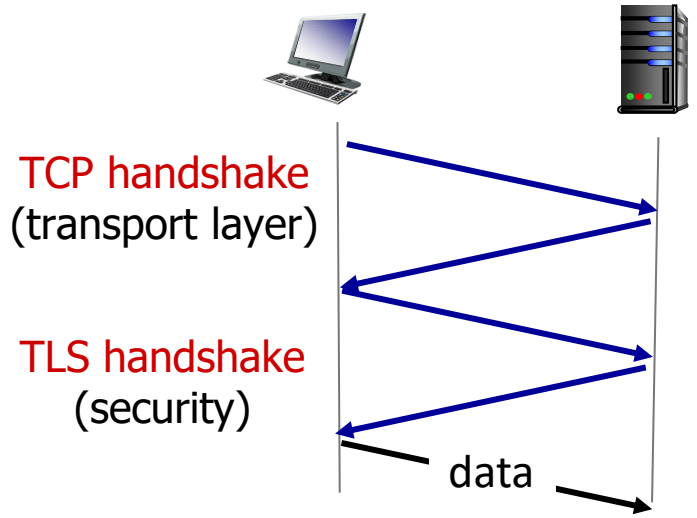


HTTP/2 sobre TCP

QUIC

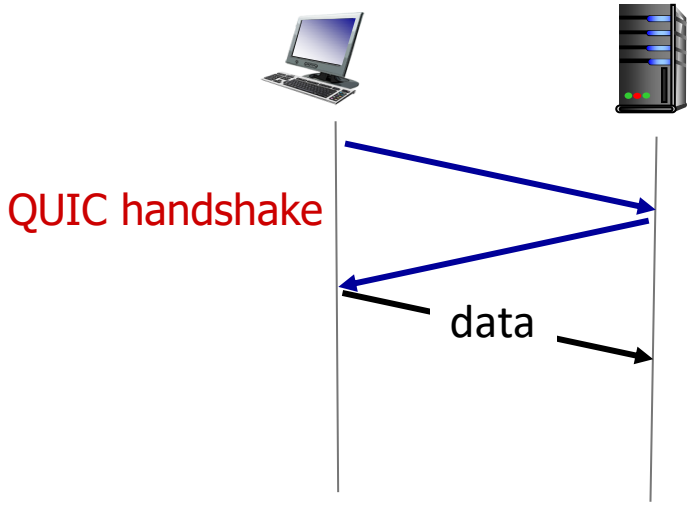
- **Transferência confiável de dados e controle de congestionamento:**
  - adota técnicas semelhantes às utilizadas no TCP (algoritmos de controle de congestionamento similares aos do TCP)
- **Estabelecimento de conexão:**
  - confiabilidade, controle de congestionamento, autenticação, criptografia, estado estabelecido em um RTT
- **Múltiplos fluxos de aplicações diferentes são multiplexados sobre uma única conexão QUIC**
  - transferência confiável de dados separadas, segurança separada
  - controle de congestionamento comum

QUIC: estabelecimento de conexão



TCP **(confiabilidade, controle de congestionamento)** + TLS **(autenticação, criptografia)**

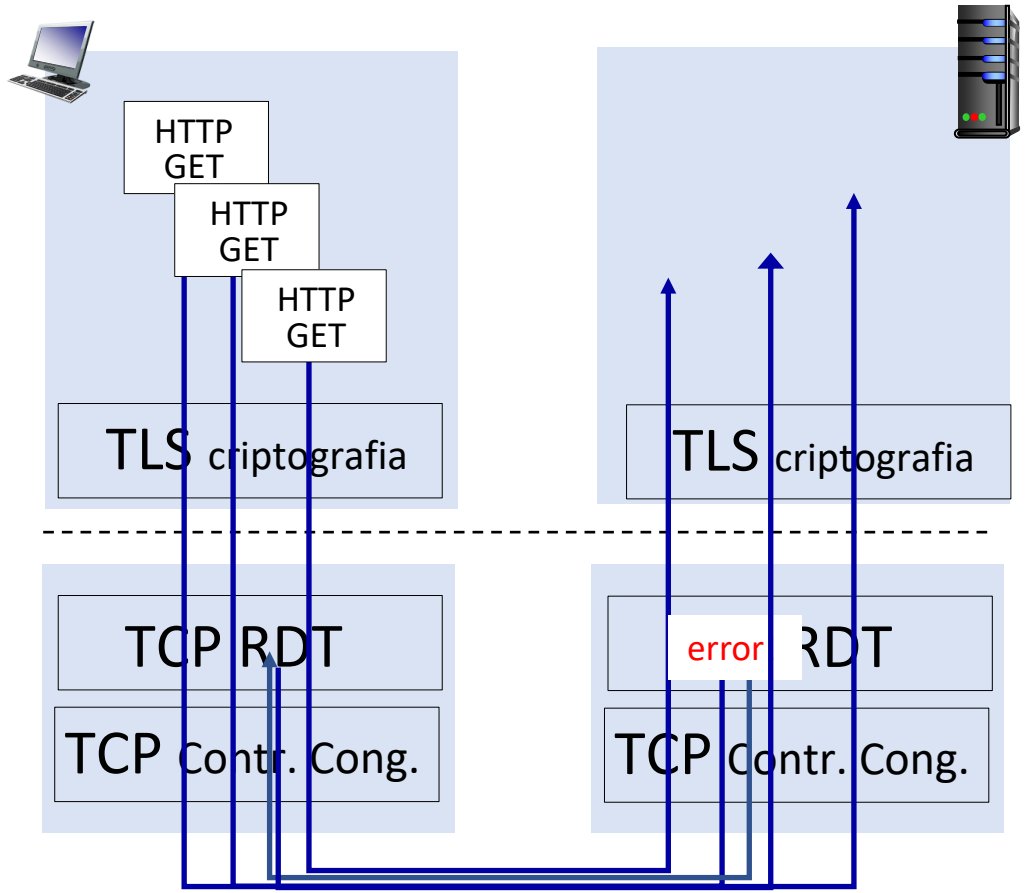
- **2 "apresentações" seriais**



QUIC: **confiabilidade, controle de congestionamento, autenticação e criptografia são estabelecidos em apenas**

- **1 apresentação**

QUIC: fluxos - paralelismo, sem bloqueio HOL



(a) HTTP 1.1

Capítulo 3: Resumo

Princípios por trás dos serviços da camada de transporte:

- multiplexação/demultiplexação
- transferência de dados confiável
- controle de fluxo
- controle de congestionamento

Principais protocolos:

- UDP
- TCP
- QUIC

A seguir:

- saímos da “borda” da rede (camadas de aplicação e de transporte)
- vamos para o “núcleo” da rede