PTC 3450 - Aula 16

3.6 Princípios do controle de congestionamento
3.7 Controle de congestionamento no TCP

(Kurose, p. 190 - 205)
(Peterson, p. 105-124 e 242-264)

26/05/2017

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

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 de controle de congestionamento

3.7 controle de congestionamento no TCP

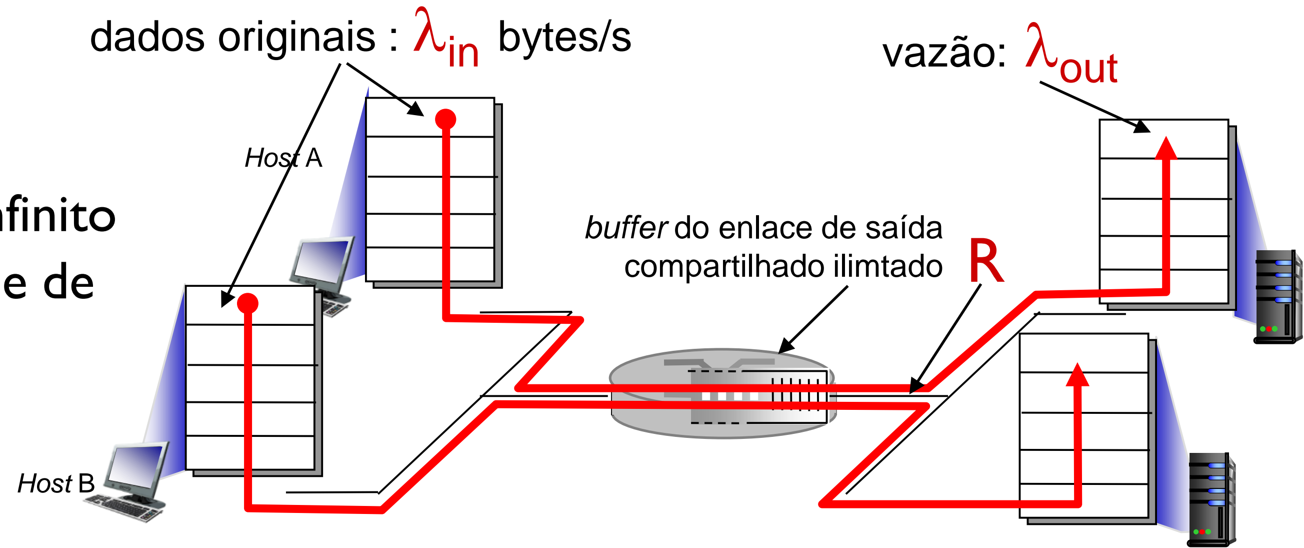
Princípios de controle de congestionamento

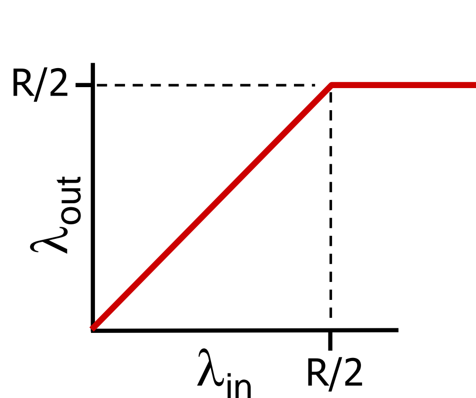
congestionamento:

- 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)

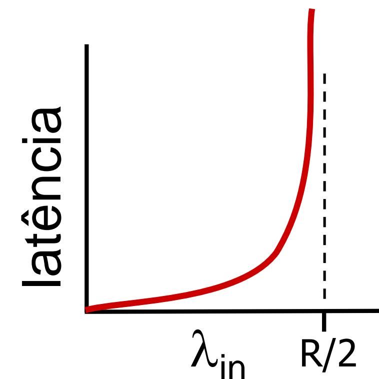
Causas/custos de congestionamento: cenário 1

- 2 remetentes, 2 destinatários
- 1 roteador, *buffer* infinito
- capacidade do enlace de saída: *R*
- sem retransmissão





- máxima vazão por conexão: *R/2*

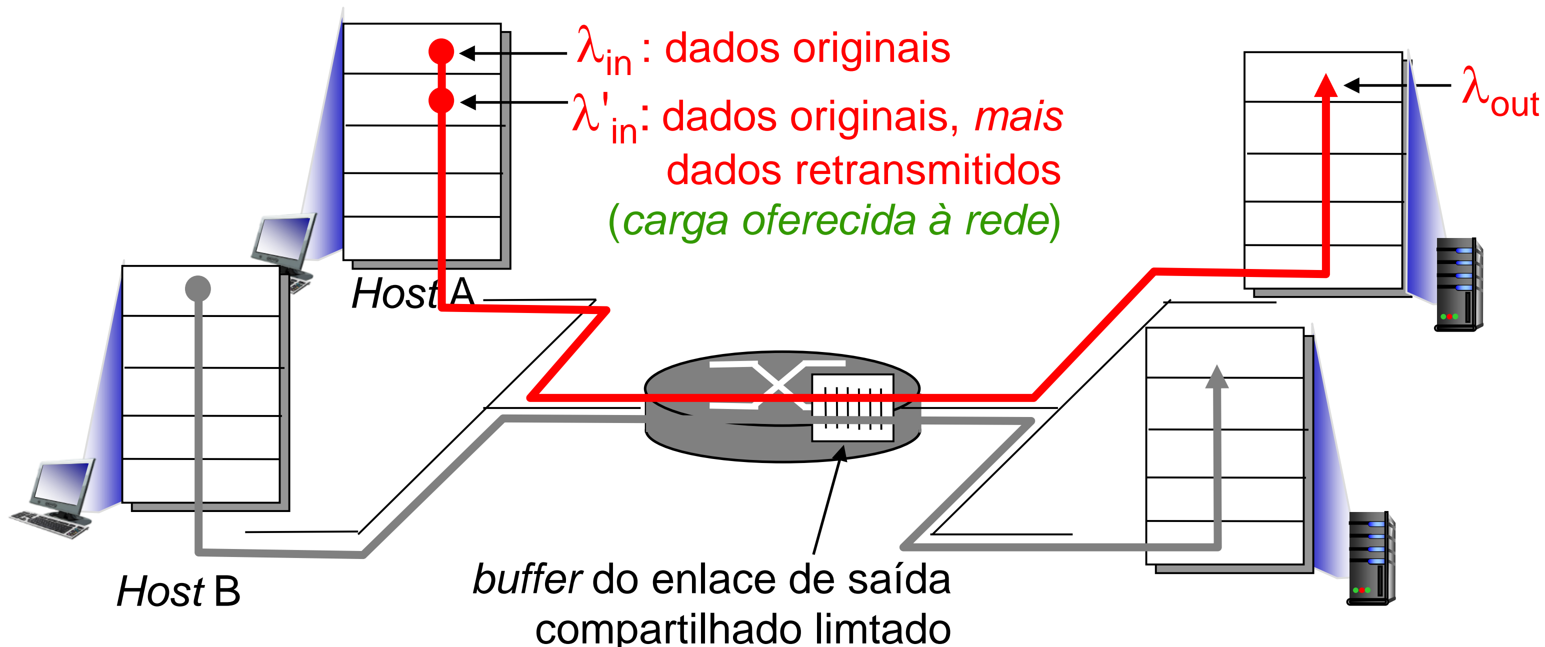


- grandes latências (atrasos) conforme taxa de chegada, λ_{in}, aproxima-se da capacidade

Veja: http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/queuing/queuing.html

Causas/custos de congestionamento: cenário 2

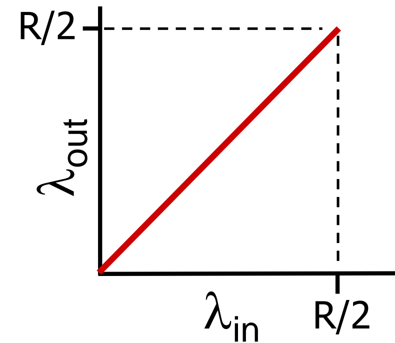
- 1 roteador, *buffer finito*
- remetente retransmite pacotes com *timeout*
  - entrada da camada de aplicação = saída para camada de aplicação: $\lambda_{in} = \lambda_{out}$
  - entrada da camada de transporte inclui retransmissão: $\lambda'_{in} \geq \lambda_{in}$



Causas/custos de congestionamento: cenário 2

idealização: conhecimento perfeito

- remetente envia apenas quando *buffer* do roteador está disponível



λ_{in} : dados originais

cópia

λ'_{in}: dados originais, *mais* dados retransmitidos

λ_{out}

A

espaço livre no buffer!

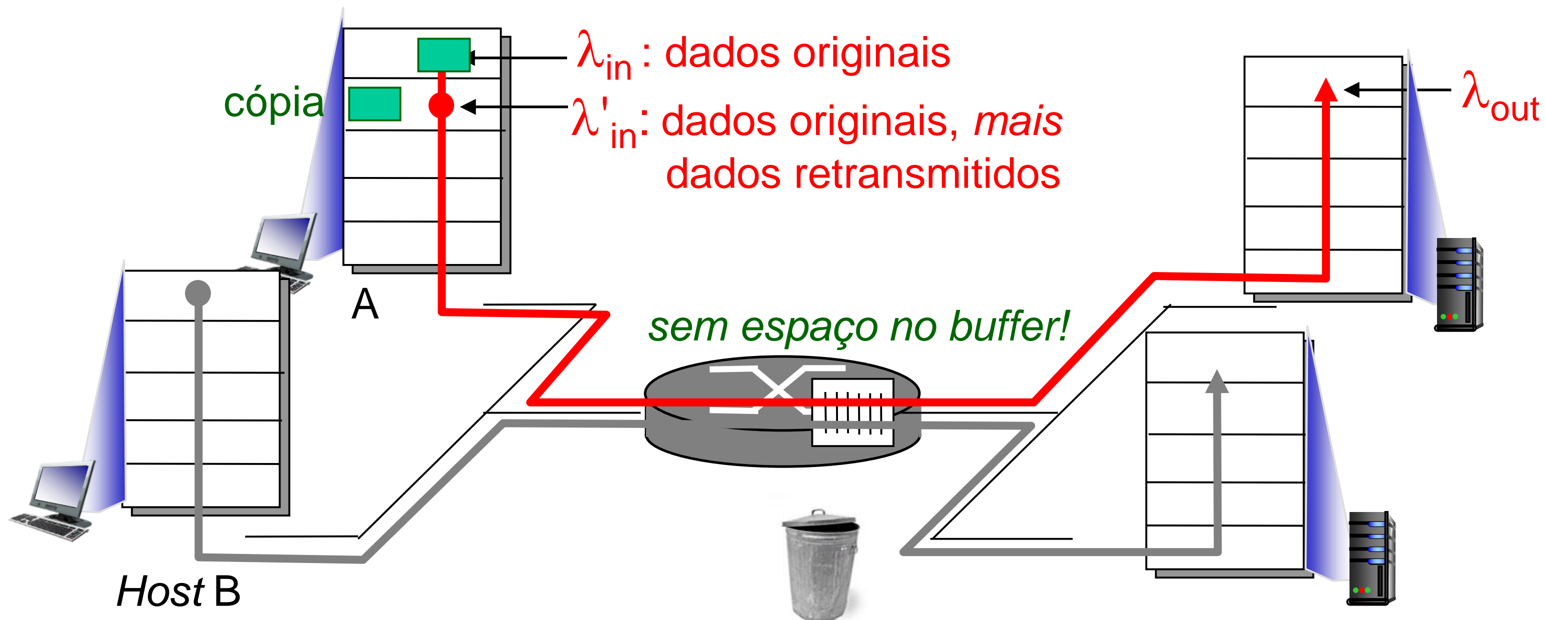
Host B

buffer do enlace de saída compartilhado limtado

Causas/custos de congestionamento: cenário 2

Idealização: perda conhecida
pacotes podem ser perdidos, descartados no roteador devido a *buffer* cheio

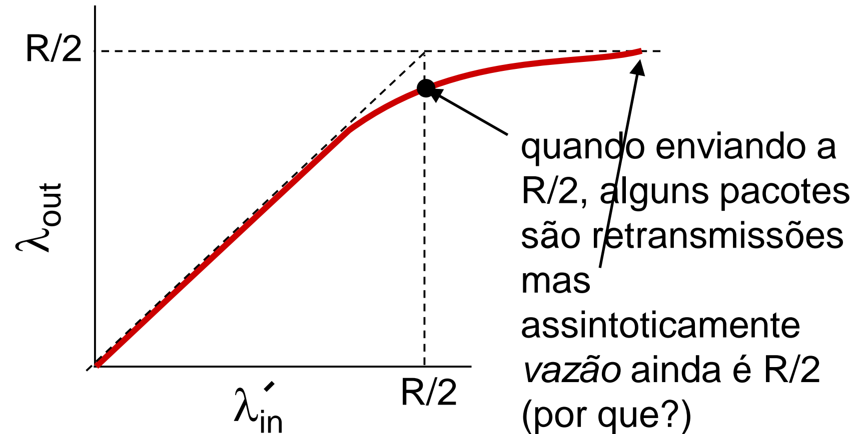
- remetente apenas reenvia se *sabe* que pacote se perdeu

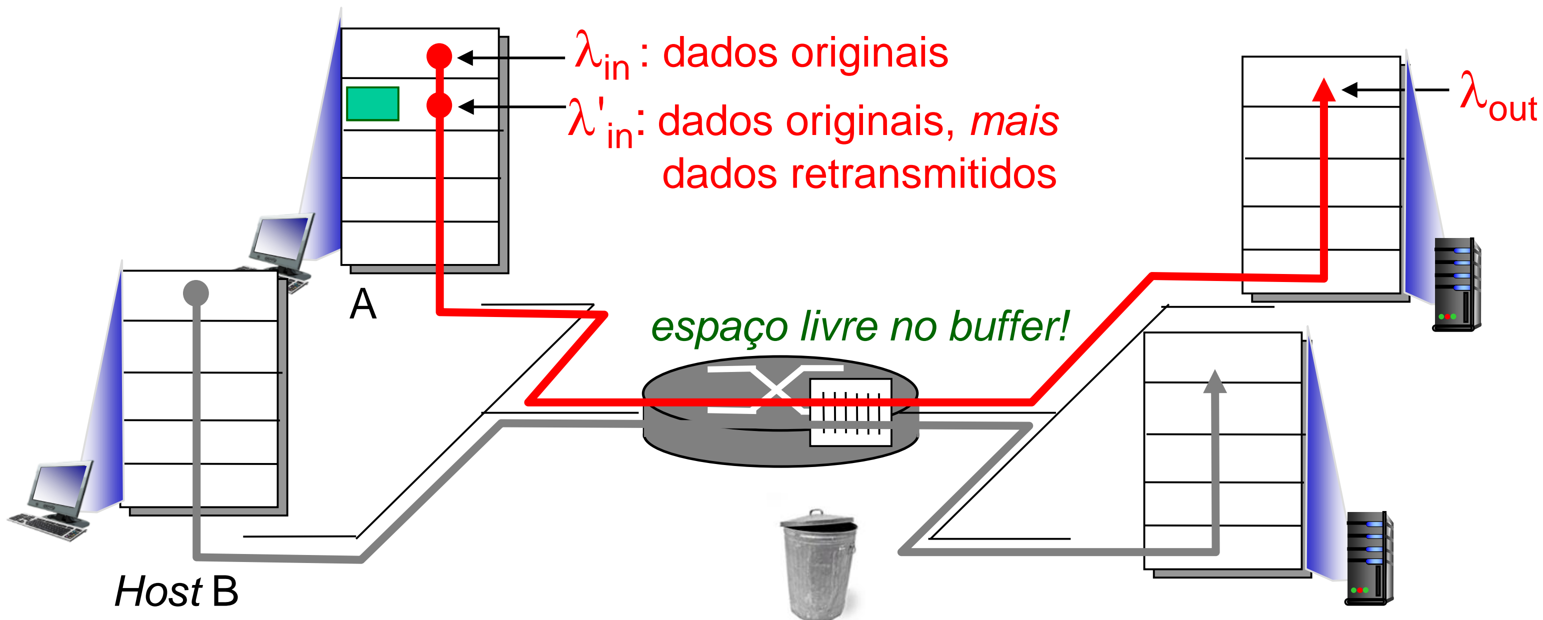


Causas/custos de congestionamento: cenário 2

Idealização: perda conhecida pacotes podem ser perdidos, descartados no roteador devido a *buffer* cheio

- remetente apenas reenvia se *sabe* que pacote se perdeu

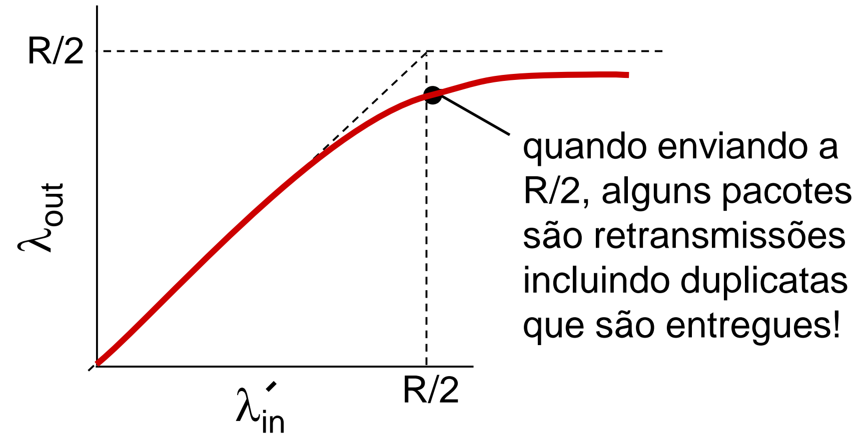


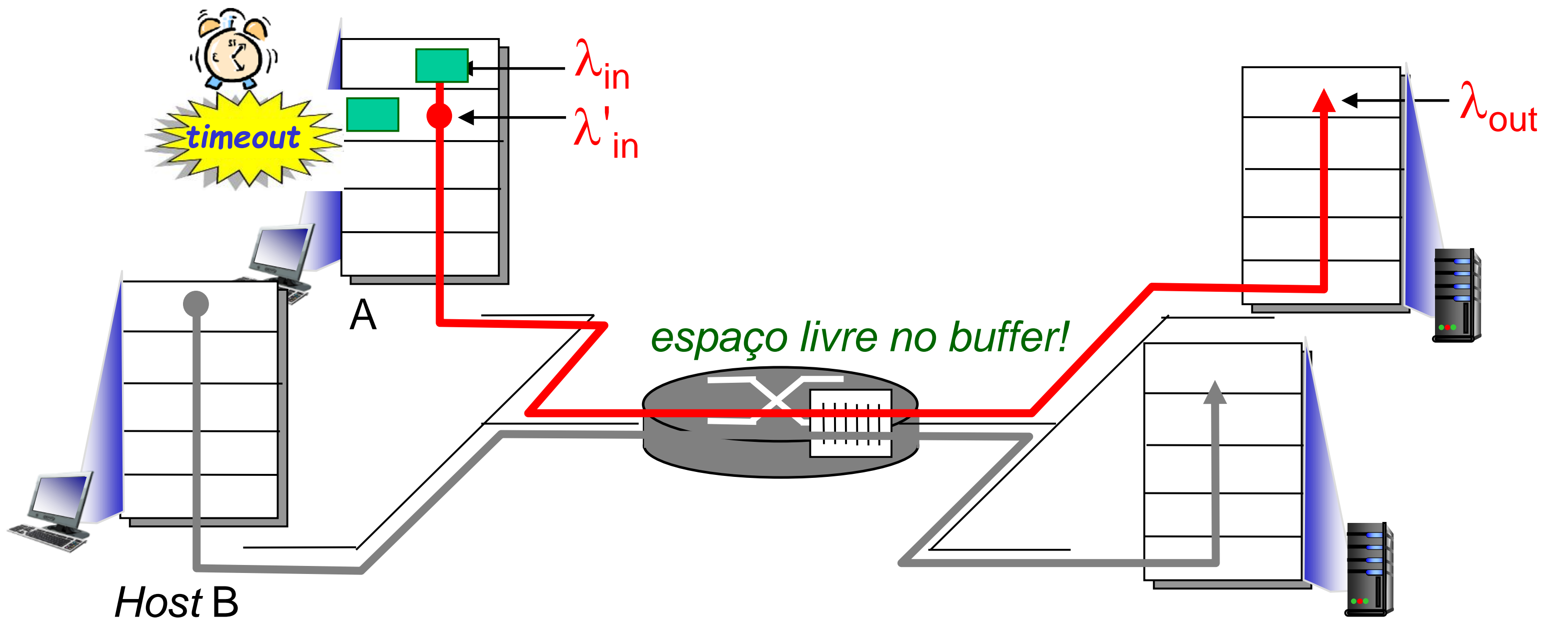


Causas/custos de congestionamento: cenário 2

Realístico: duplicatas

- pacotes podem ser perdidos, descartados no roteador devido a *buffer* cheio
- *timeout* no remetente pode ser prematuro, enviando 2 cópias ambas entregues

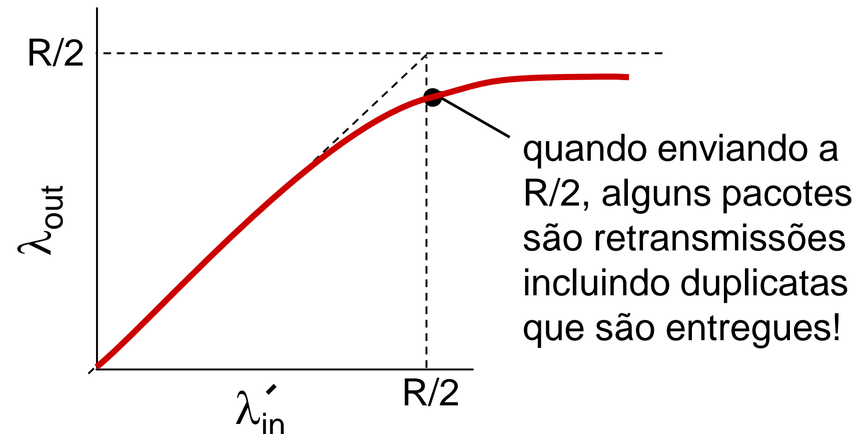




Causas/custos de congestionamento: cenário 2

Realístico: duplicatas

- pacotes podem ser perdidos, descartados no roteador devido a *buffer* cheio
- *timeout* no remetente pode ser prematuro, enviando 2 cópias ambas entregues



"custos" do congestionamento:

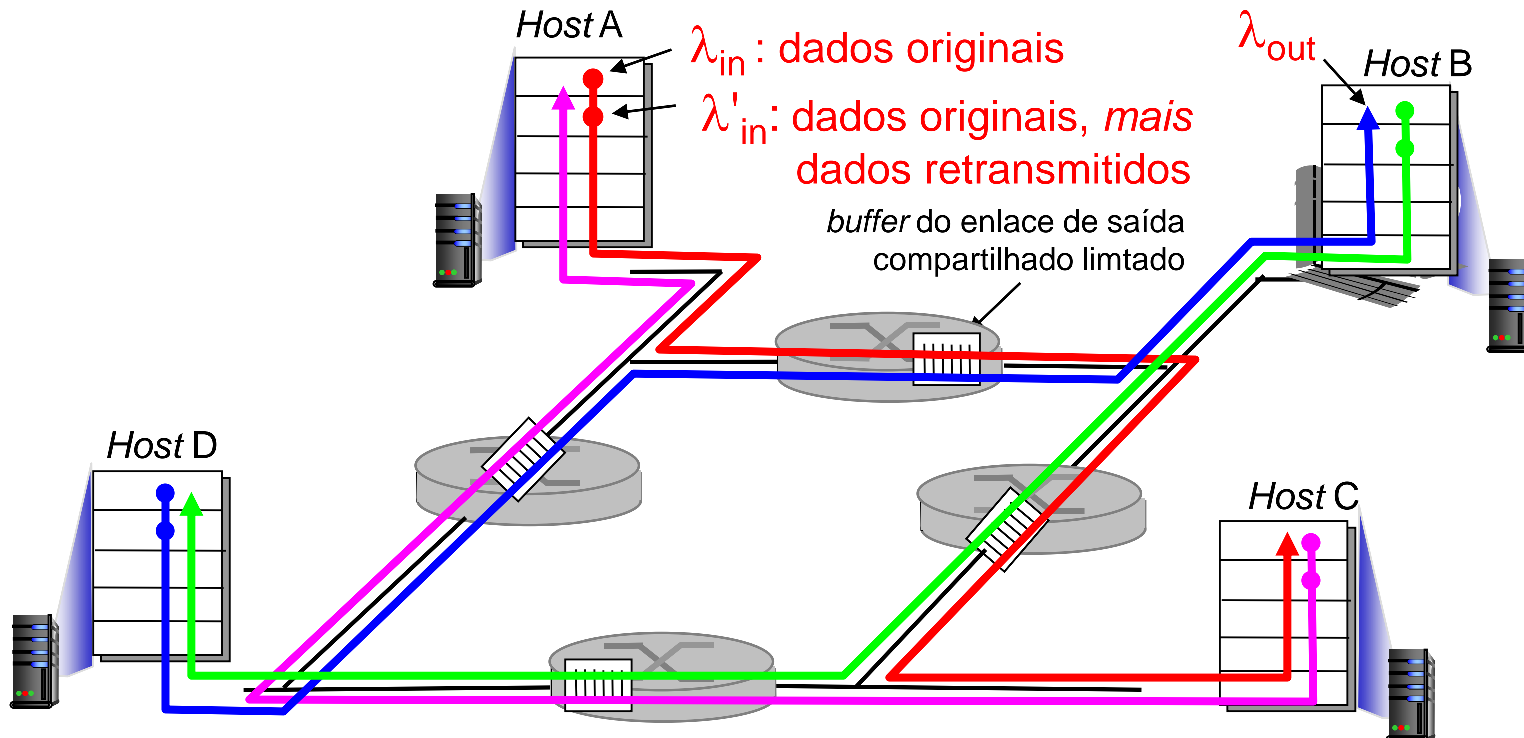
- mais trabalho do roteador e do transmissor (retransmissões) para dada vazão
- retransmissões desnecessárias: enlace carrega múltiplas cópias de pacotes
  - diminuição da vazão

Causas/custos de congestionamento: cenário 3

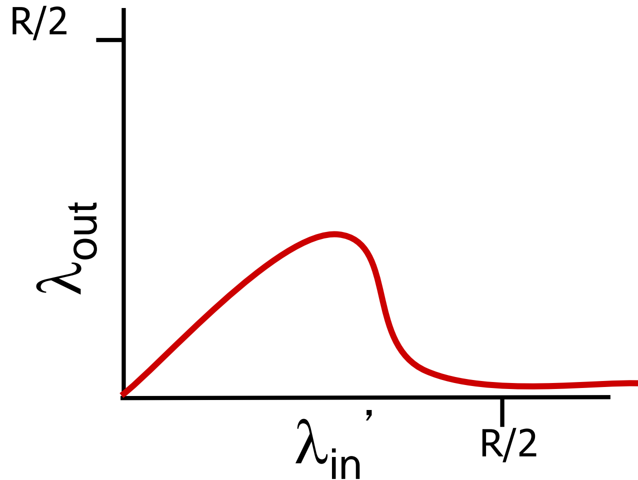
- 4 remetentes
- caminhos com múltiplos saltos
- *timeout*/retransmição

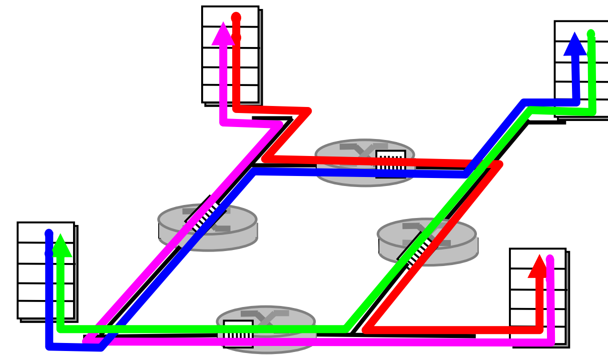
Q: o que acontece quando λ_{in} e λ_{in}' aumentam?

R: quando λ_{in}' (vermelho) aumenta, todos pacotes azuis na fila superior são descartados, vazão azul → 0



Causas/custos de congestionamento: cenário 3





outro "custo" do congestionamento:

- quando pacotes são descartados, toda capacidade de transmissão *até aquele ponto* usada para aquele pacote é perdida!

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

Estudo de caso: controle de congestionamento do ATM ABR

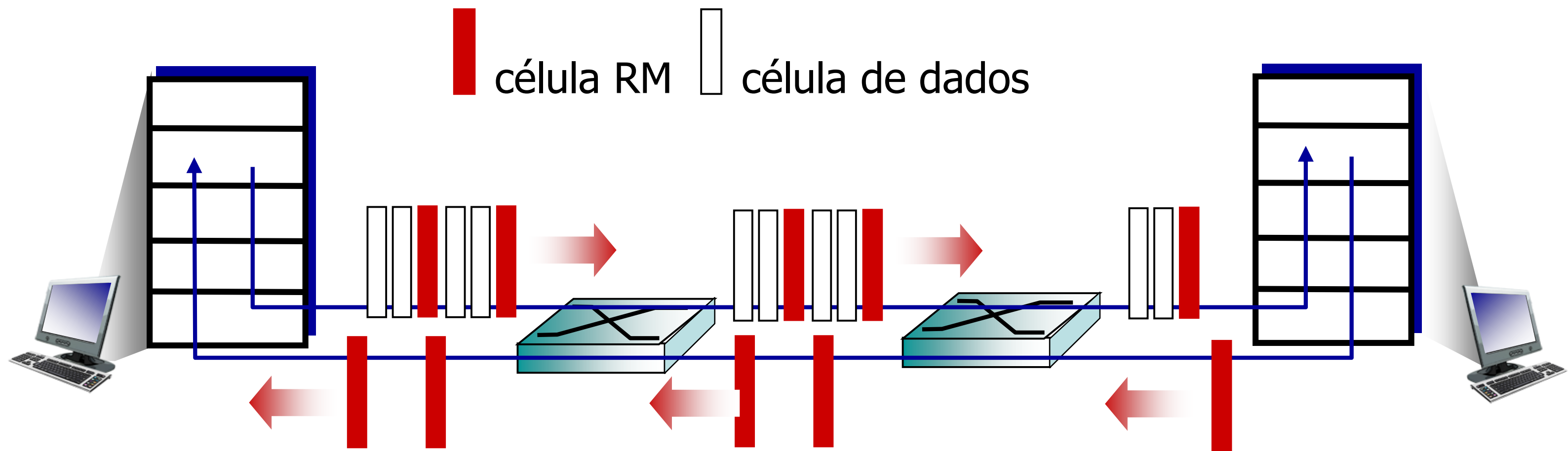
ABR: *Available Bit Rate:*

- ATM: comutação de *circuito virtual (VC)*
- *Cada switch* no caminho fonte-destino mantém estado sobre VC
- Permite que *switch* rastreie comportamento de fontes individuais
- ABR: "serviço elástico"
- se caminho do remetente "subutilizado":
  - remetente deve usar capacidade disponível
- se caminho do remetente congestionado:
  - remetente estrangulado para mínima taxa garantida

células (pacotes) RM (*resource management*) :

- enviada pelo remetente, intercaladas com células de dados (*default* a cada 32 células de dados)
- Destino envia células RM de volta, podendo modificá-las
- *Switches* também podem enviar e modificar células RM
- bits nas células RM modificados por *switches* ("*assistido pela rede*")
  - *bit NI:* não aumentar taxa (congestionamento moderado)
  - *bit CI:* indicador de congestionamento

Estudo de caso: controle de congestionamento do ATM ABR



- campo ER (*explicit rate*) de 2 bytes nas células RM
  - *switch* congestionado pode baixar valor ER na célula
  - remetente envia assim na máxima taxa suportada pelo caminho
- bit EFCI *(explicit forward congestion indication)* nas células de dados: modificado para 1 em *switch* congestionado
  - se célula de dados precedendo célula RM tem EFCI assinalado, destinatário ativa bit CI em célula RM devolvida

Capítulo 3: conteúdo

3.1 serviços da camada de transporte

3.2 multiplexação e desmultiplexação

3.3 transporte sem conexão: UDP

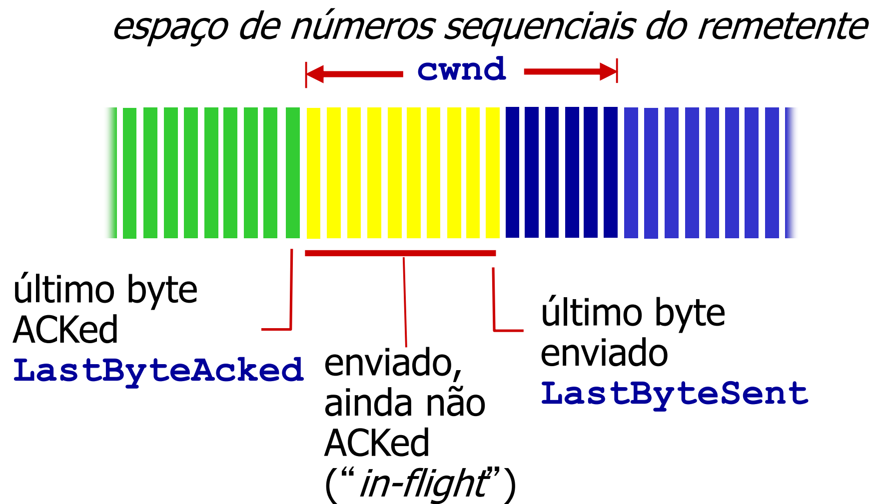
3.4 princípios da transferência de dados confiável

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: Controle de Congestionamento : detalhes



- remetente limita transmissão:

$$\texttt{LastByteSent-LastByteAcked} \le \texttt{cwnd}$$

- **cwnd** é dinâmico, função do congestionamento da rede notado

Taxa aprox. de envio TCP :

- Envia **cwnd** bytes, espera RTT por ACKS, então envia mais bytes

$$\text{taxa} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/s}$$

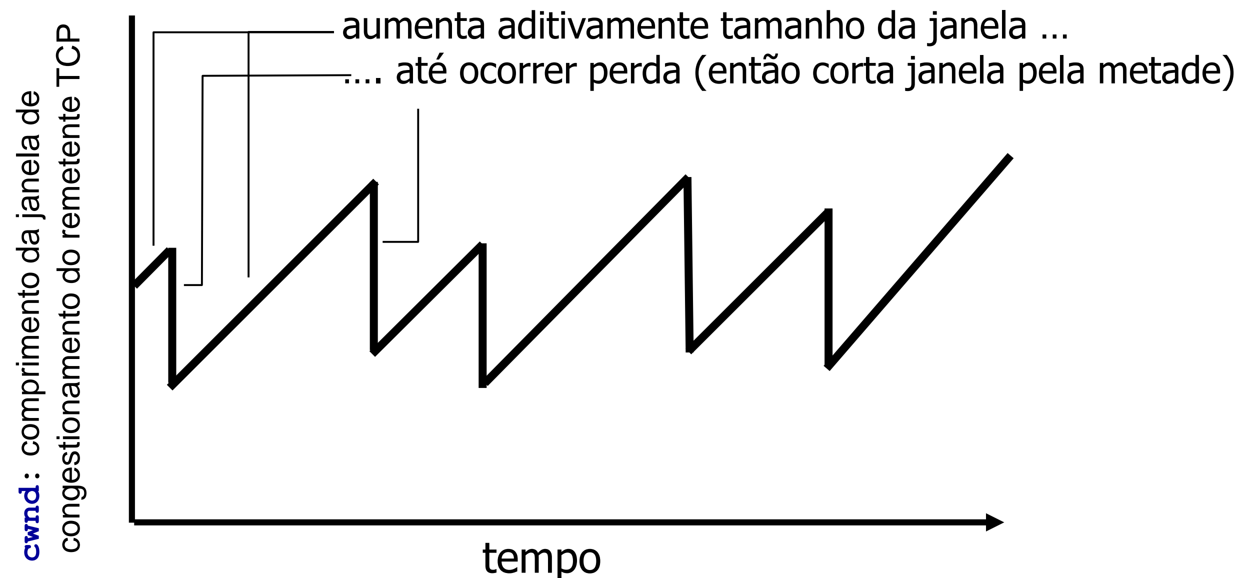
- Obs: Levando-se em conta também o controle de fluxo o comprimento da janela é **min{cwnd,rwnd}**. Vamos considerar aquir que o buffer do destino é suficientemente grande.

TCP: controle de congestionamento

incremento aditivo, decremento multiplicativo (AIMD)

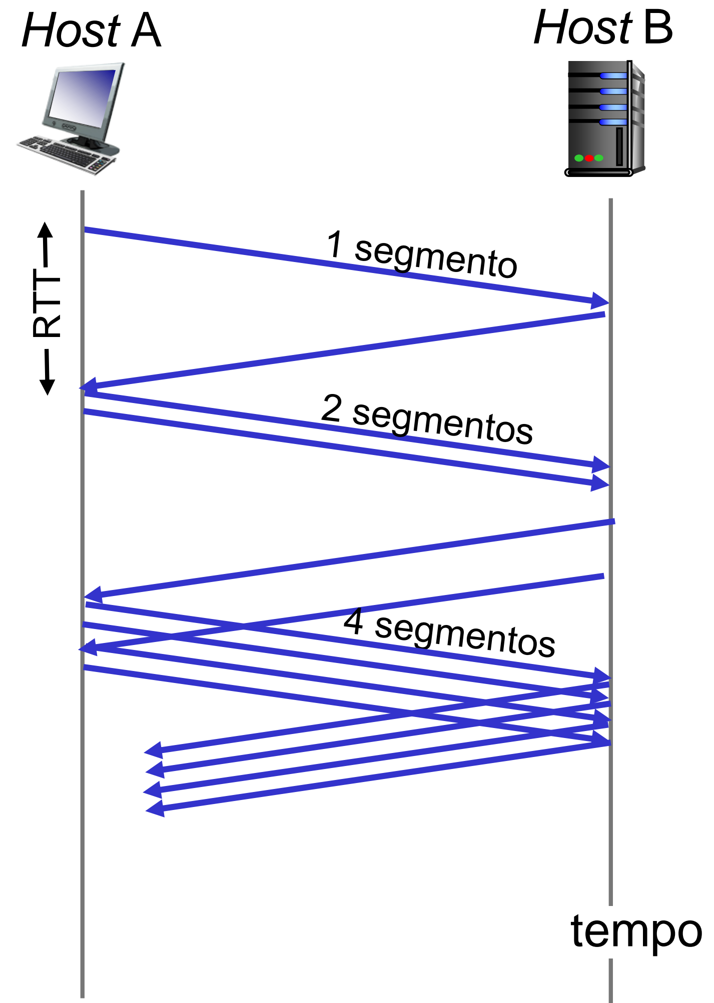
- *Controle fim a fim:* IP não fornece realimentação
- *abordagem:* remetente aumenta taxa de transmissão (compr. janela), sondando por capacidade utilizável, até ocorrer perda [Jacobson, 1988; RFC 5681 - 2009]
  - *aumento aditivo:* aumenta **cwnd** em 1 MSS a cada ACK válido (tudo vai bem! ☺) até que perda é detectada (indício de congestionamento ☹)
  - *diminuição multiplicativa:* corta **cwnd** pela metade após perda

comportamento dente de serra do AIMD: sondando por capacidade



TCP: partida lenta

- componente obrigatório, assim como contenção de congestionamento (CA)
- quando conexão inicia, aumenta taxa exponencialmente até primeiro evento de perda:
  - inicialmente **cwnd** = 1 MSS (RFC3390)
  - dobra **cwnd** a cada RTT
  - feito incrementando **cwnd** de 1 MSS para cada ACK recebido
- *resumo:* taxa inicial é lenta mas aumenta exponencialmente rápido



TCP: detectando e reagindo a perda

- perda indicada por *timeout:*
  - Faz `ssthresh = cwnd/2` (*slow start threshold)*
  - `cwnd` ajustado para 1 MSS;
  - janela então cresce exponencialmente (com na partida lenta) até `ssthresh`, então cresce linearmente (*congestion avoidance mode* – CA)

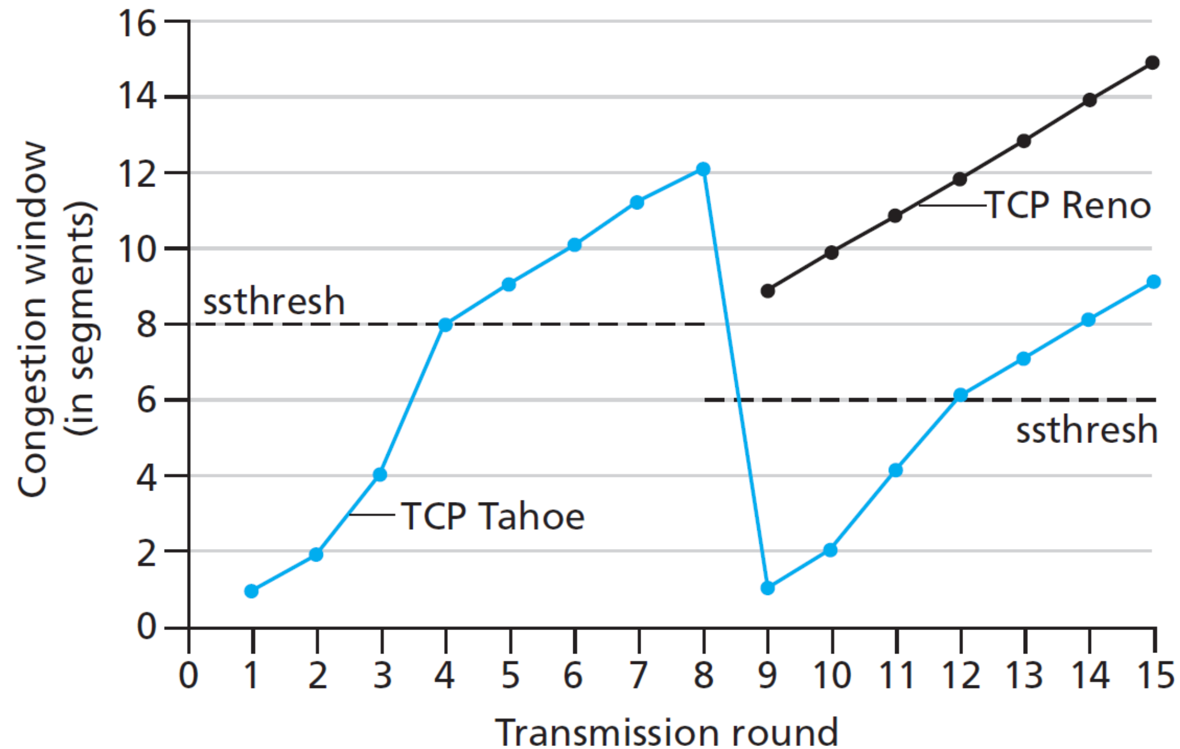
- perda indicada por 3 ACKs duplicados: TCP RENO
  - ACKs dup indicam capacidade da rede de entregar alguns segmentos
  - `cwnd = cwnd/2` + 3 e então cresce linearmente – recuperação rápida

- TCP Tahoe sempre ajusta `cwnd` para 1 (*timeout* ou 3 ACKs duplicados)

TCP: mudando de partida lenta para CA

Q: quando o aumento exponencial deve mudar para linear?

R: quando **cwnd** chega a 1/2 do seu valor antes do *timeout*.



Implementação:

- variável **ssthresh** (slow start threshold)
- em evento de perda, **ssthresh** é ajustada para 1/2 de **cwnd** logo antes do evento de perda

Resumo: Controle de Congestionamento do TCP

Λ
cwnd = 1 MSS
ssthresh = 64 KB
dupACKcount = 0

ACK duplicado
dupACKcount++

Novo ACK!

novo ACK
cwnd = cwnd+MSS
dupACKcount = 0
transmite novo(s) segmento(s), se permitido

Novo ACK!

novo ACK
cwnd = cwnd + MSS • (MSS/cwnd)
dupACKcount = 0
transmite novo(s) segmento(s), se permitido

partida lenta

cwnd ≥ ssthresh
Λ

Contenção de congestionamento

timeout
ssthresh = cwnd/2
cwnd = 1 MSS
dupACKcount = 0
retransmite segmento perdido

ACK duplicado
dupACKcount++

timeout
ssthresh = cwnd/2
cwnd = 1 MSS
dupACKcount = 0
retransmite segmento perdido

timeout
ssthresh = cwnd/2
cwnd = 1
dupACKcount = 0
retransmite segmento perdido

Novo ACK!

Novo ACK
cwnd = ssthresh
dupACKcount = 0

dupACKcount == 3
ssthresh= cwnd/2
cwnd = ssthresh + 3
retransmite segmento perdido

dupACKcount == 3
ssthresh= cwnd/2
cwnd = ssthresh + 3
retransmite segmento perdido

recuperação rápida

ACK duplicado
cwnd = cwnd + MSS
transmite novo(s) segmento(s), se permitido

TABLE I
FEATURES OF TCP VARIANTS THAT SOLVE THE CONGESTION COLLAPSE PROBLEM

TCP Variant	Section	Year	Base	Added/Changed Modes or Features	Mod[1]	Status	Implementation			
							BSD[2]	Linux	Win	Mac
TCP Tahoe [14]	II-A	1988	RFC793	Slow Start, Congestion Avoidance, Fast Retransmit	S	Obsolete Standard	>4.3	1.0		
TCP-DUAL [15]	II-B	1992	Tahoe	Queuing delay as a supplemental congestion prediction parameter for Congestion Avoidance	S	Experimental				
TCP Reno [16], [17]	II-C	1990	Tahoe	Fast Recovery	S	Standard	>4.3 >F2.2	> 1.3.90	>95/NT	
TCP NewReno [18], [19]	II-D	1999	Reno	Fast Recovery resistant to multiple losses	S	Standard	>F4	> 2.1.36		>10.4.6 (opt)
TCP SACK [20]	II-E	1996	RFC793	Extended information in feedback messages	P+S+R	Standard	>S2.6, >N1.1, >F2.1R	> 2.1.90	> 98	> 10.4.6
TCP FACK [21]	II-F	1996	Reno, SACK	SACK-based loss recovery algorithm	S	Experimental	>N1.1	>2.1.92		
TCP-Vegas [22]	II-G	1995	Reno	Bottleneck buffer utilization as a primary feedback for the Congestion Avoidance and secondary for the Slow Start	S	Experimental		> 2.2.10		
TCP-Vegas+ [23]	II-H	2000	NewReno, Vegas	Reno/Vegas Congestion Avoidance mode switching based of RTT dynamics	S	Experimental				
TCP-Veno [24]	II-I	2002	NewReno, Vegas	Reno-type Congestion Avoidance and Fast Recovery increase/decrease coefficient adaptation based on bottleneck buffer state estimation	S	Experimental		> 2.6.18		
TCP-Vegas A [25]	II-J	2005	Vegas	Adaptive bottleneck buffer state aware Congestion Avoidance	S	Experimental				

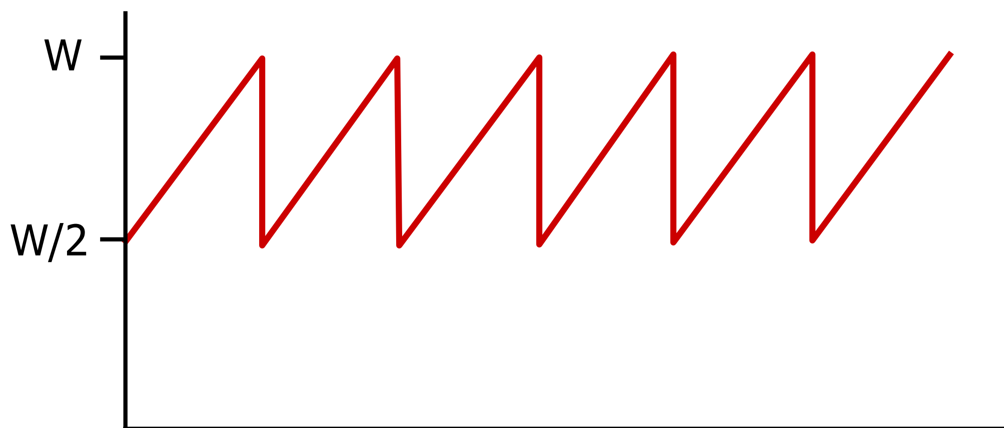
[1] TCP specification modification: S = the sender reactions, R = the receiver reactions, P = the protocol specification
[2] S for Sun, F for FreeBSD, N for NetBSD

Exemplos de experimentos: aqui, aqui e aqui.
Implementação no Windows NT

TCP: vazão

- vazão TCP média em função do comprimento da janela, RTT?
  - hipóteses: ignorar partida lenta (TCP sai dela exponencialmente rápido), assumir que sempre existe dados para enviar, comprimento da janela em que ocorre perda é constante
- W: comprimento da janela (medido em bytes) em que ocorre perda
  - tamanho médio da janela (número de bytes *in-flight*) é ¾ W
  - vazão média é 3/4W por RTT

$$\text{vazão média TCP} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/s}$$



Futuro do TCP : TCP em "canos longos e grossos"

- exemplo: segmentos de 1 500 bytes, RTT 100ms, queremos vazão de10 Gbps
- Quantos segmentos *in-flight* em média?
- requer em média 83 333 segmentos *in-flight*
- vazão em termos de probabilidade de perda de segmentos, L [Mathis 1997]:

$$\text{vazão TCP} = \frac{1{,}22 \cdot MSS}{RTT\sqrt{L}}$$

- Qual a taxa de perda para alcançar 10 Gbps?
- *$L = 2 \cdot 10^{-10}$ – taxa de perda muito baixa!*
- novas versões do TCP para altas velocidades
- TCP Vegas, CUBIC, etc. – Muita pesquisa interessante!