Camada de Transporte – Parte II

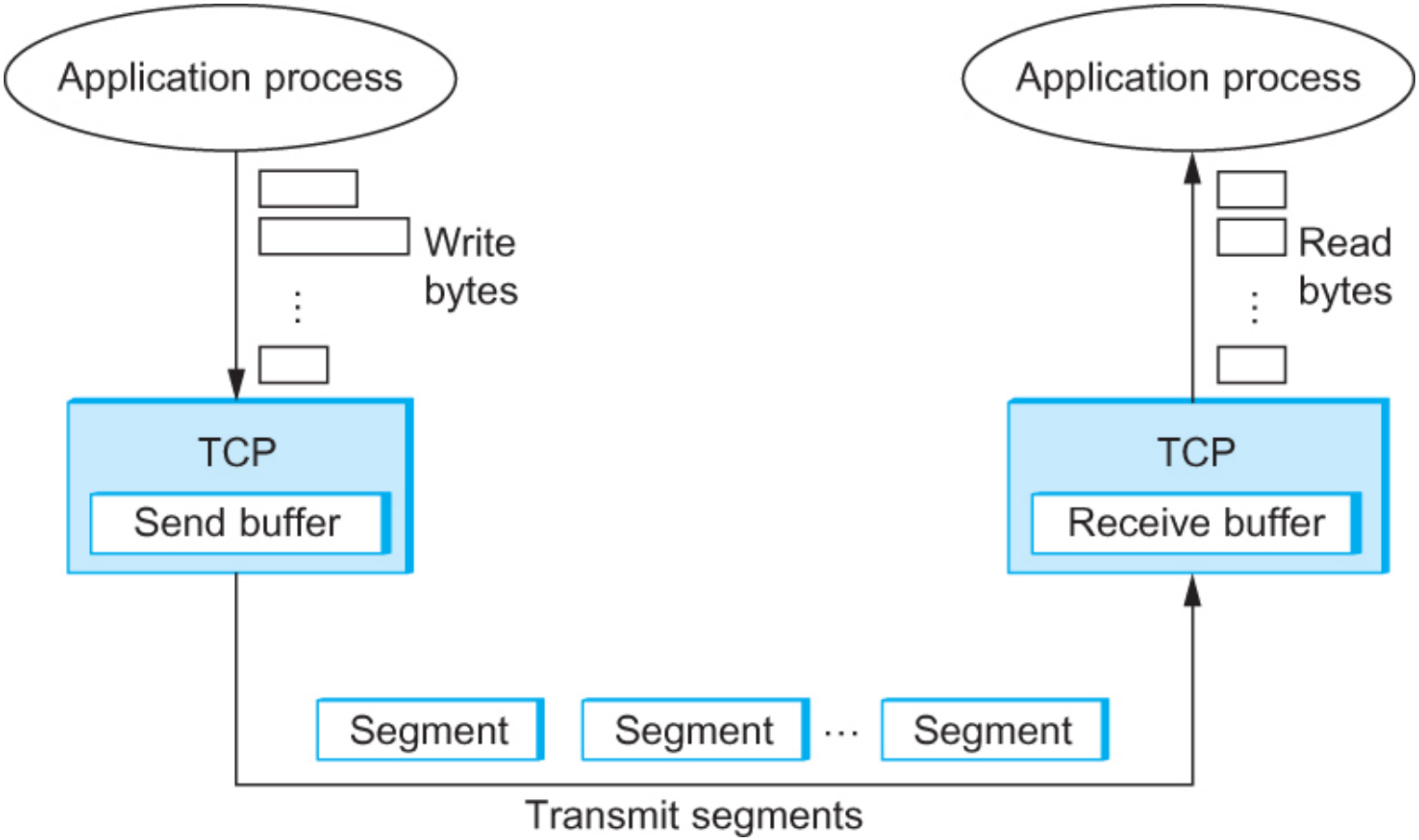
Prof. Jaime Cohen

2018

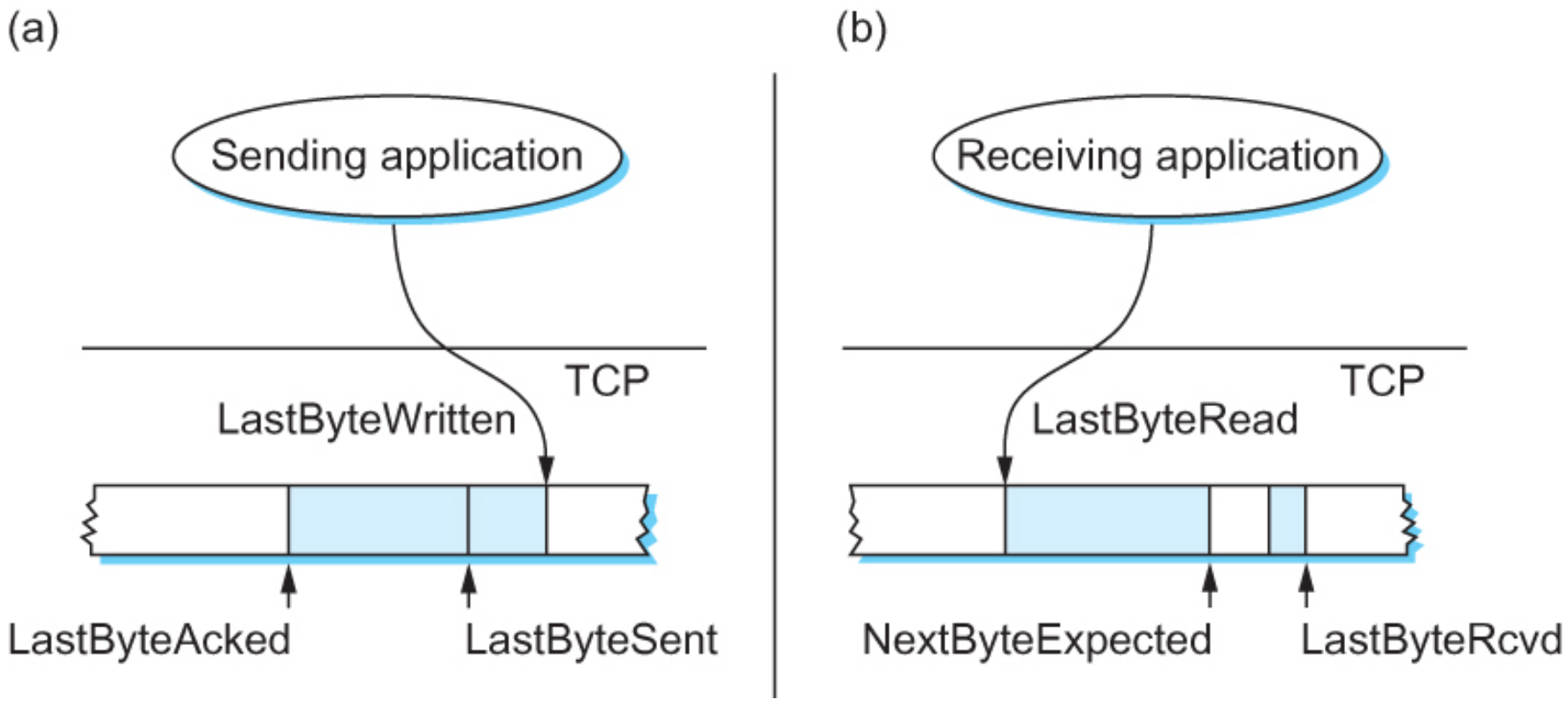
Sumário

Camada de Transporte (parte II)

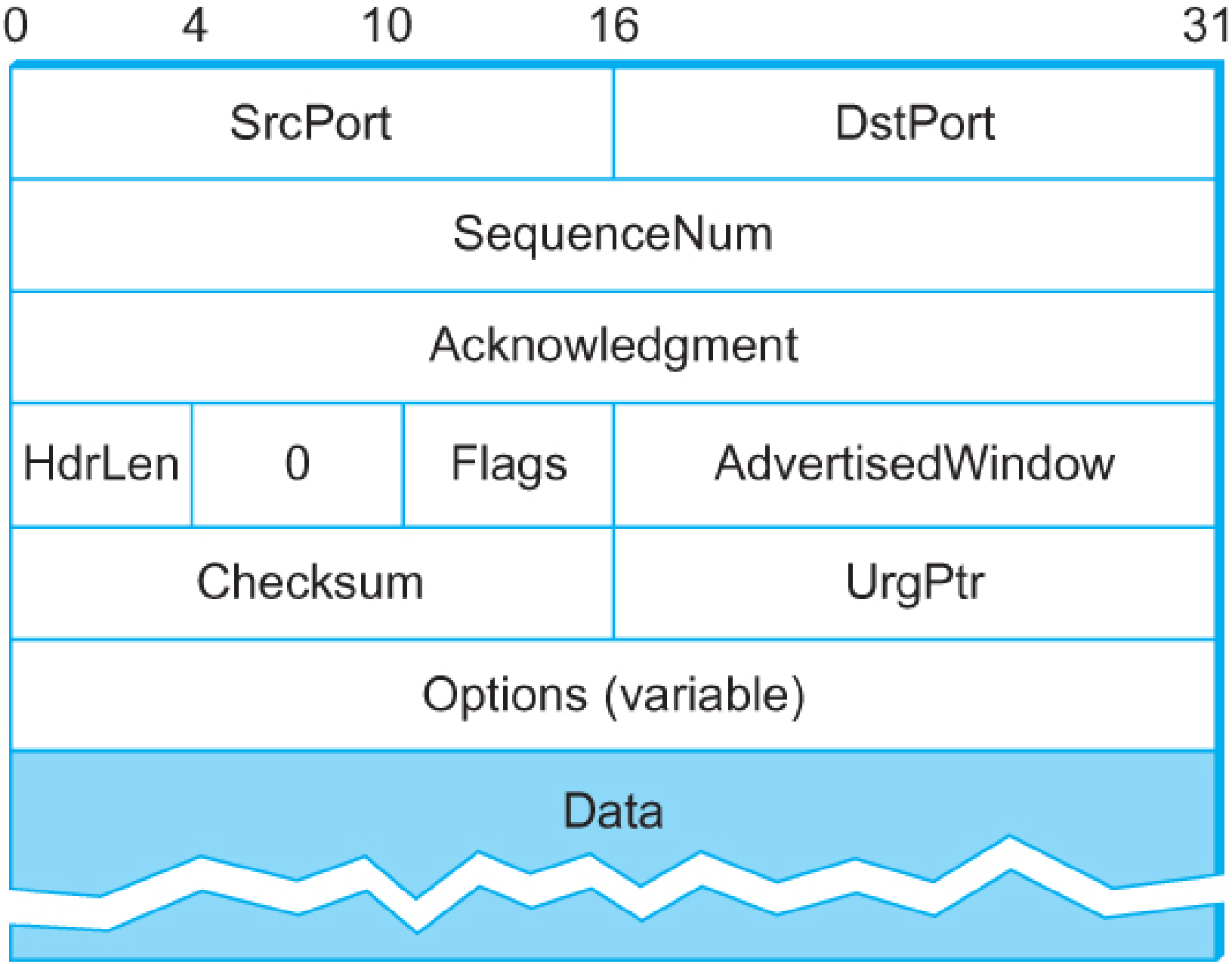
Janela Deslizante do TCP



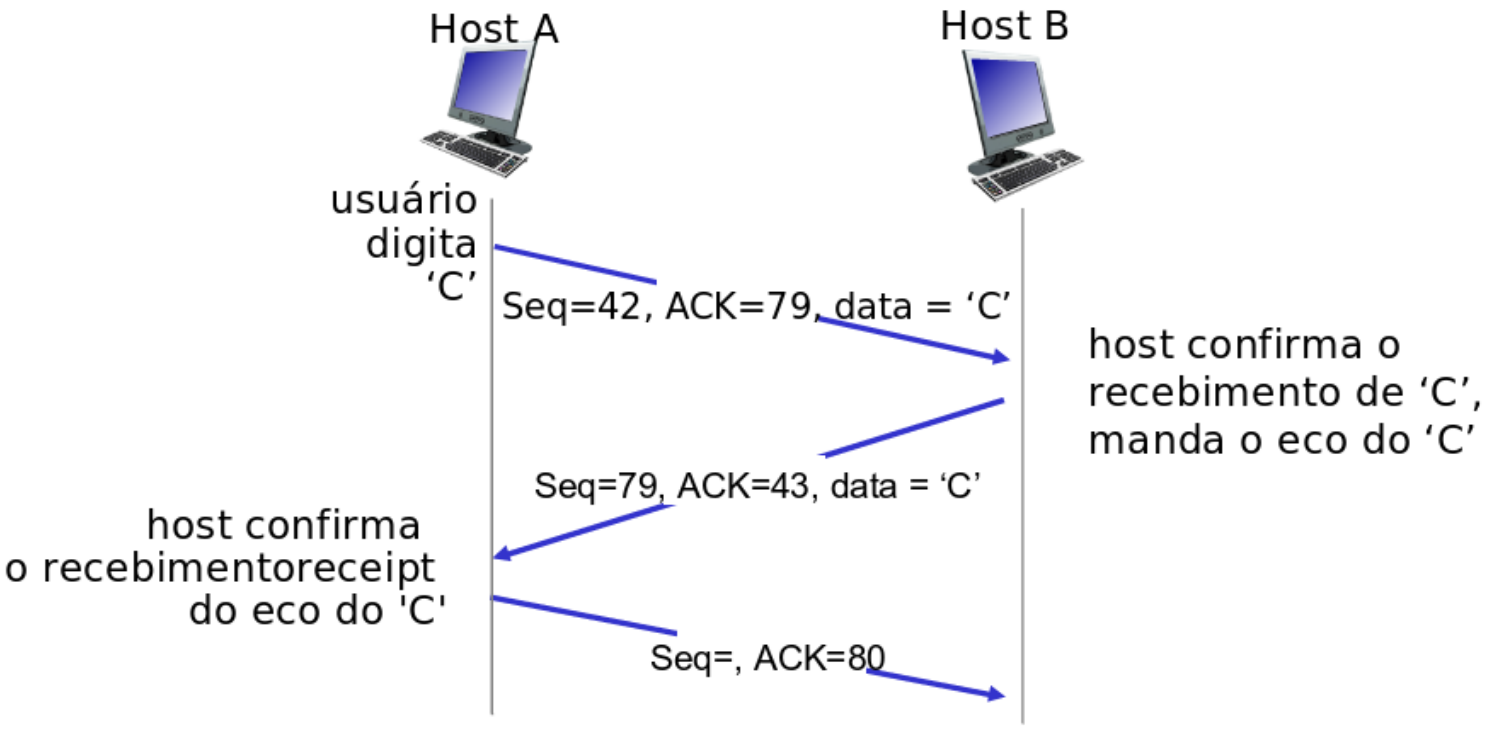
TCP Fluxo de Bytes



Cabeçalho TCP



Confirmações



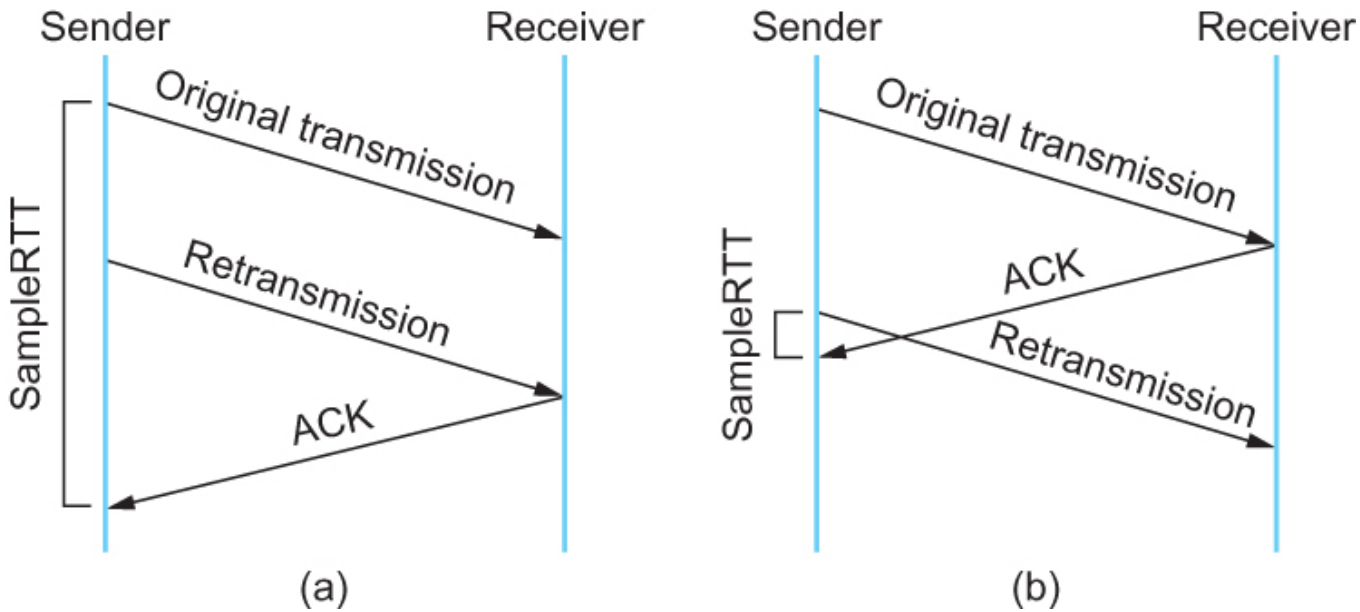
Exemplo: TELNET (caractere por caractere)

Timeouts e RTT

- Como definir o tempo de *timeout*?
- Deve ser maior do que o RTT
- Como lidar com as variações do RTT?
- Muito pequeno $\Rightarrow$ retransmissões desnecessárias
- Muito grande $\Rightarrow$ demora para retransmitir

Estimativa do RTT

- Mede o tempo entre o envio de um segmento e o recebimento da confirmação.
- Desconsidera dados retransmitidos.

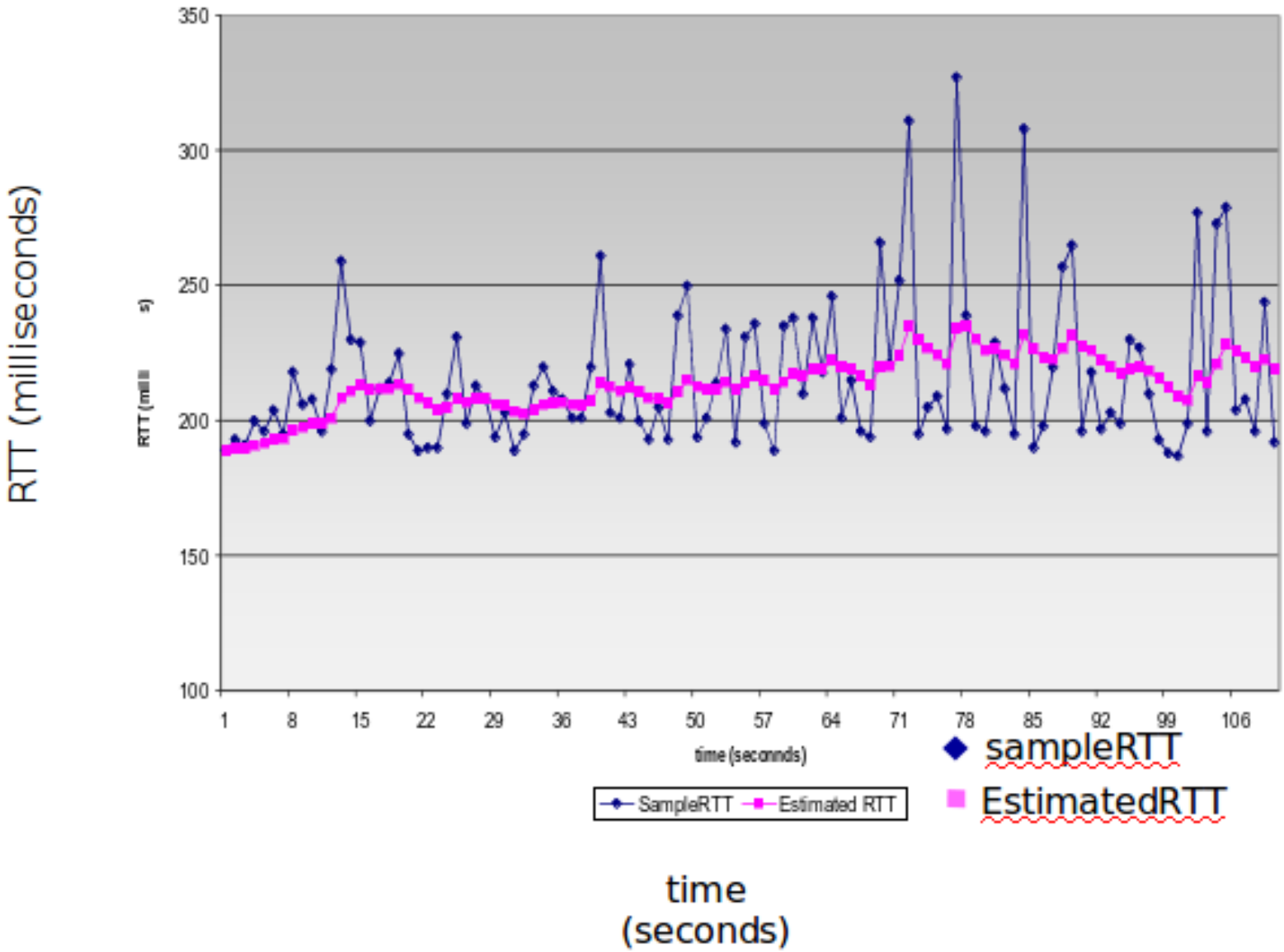


Estimativa do RTT

- Como o RTT varia, precisa adaptar
- Usa média móvel com decaimento exponencial
- $\mathit{EstimatedRTT} = (1 - \alpha) * \mathit{EstimatedRTT} + \alpha * \mathit{SampleRTT}$
- o peso das amostras antigas diminuiu exponencialmente
- α típico: 0.125

Estimativa do RTT

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



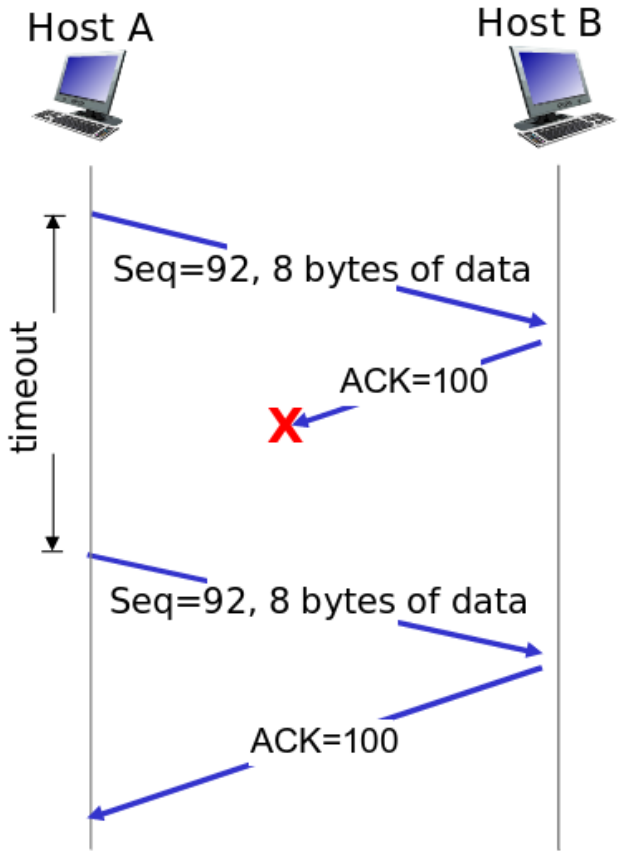
Cálculo do *timeout*

- RTT estimado mais uma margem de segurança
- Se a variação do RTT estimado é grande, então a margem de segurança é maior
- $DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$
- β típico: 0.25

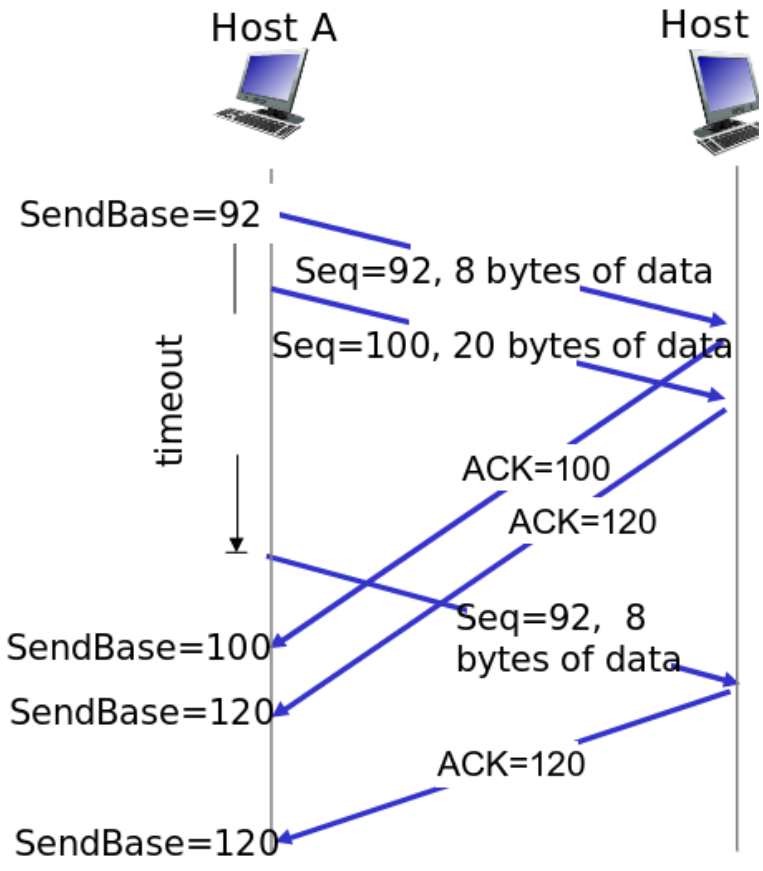
Timeout estimado

$Timeout = EstimatedRTT + 4 * DevRTT$

TCP + Timeouts

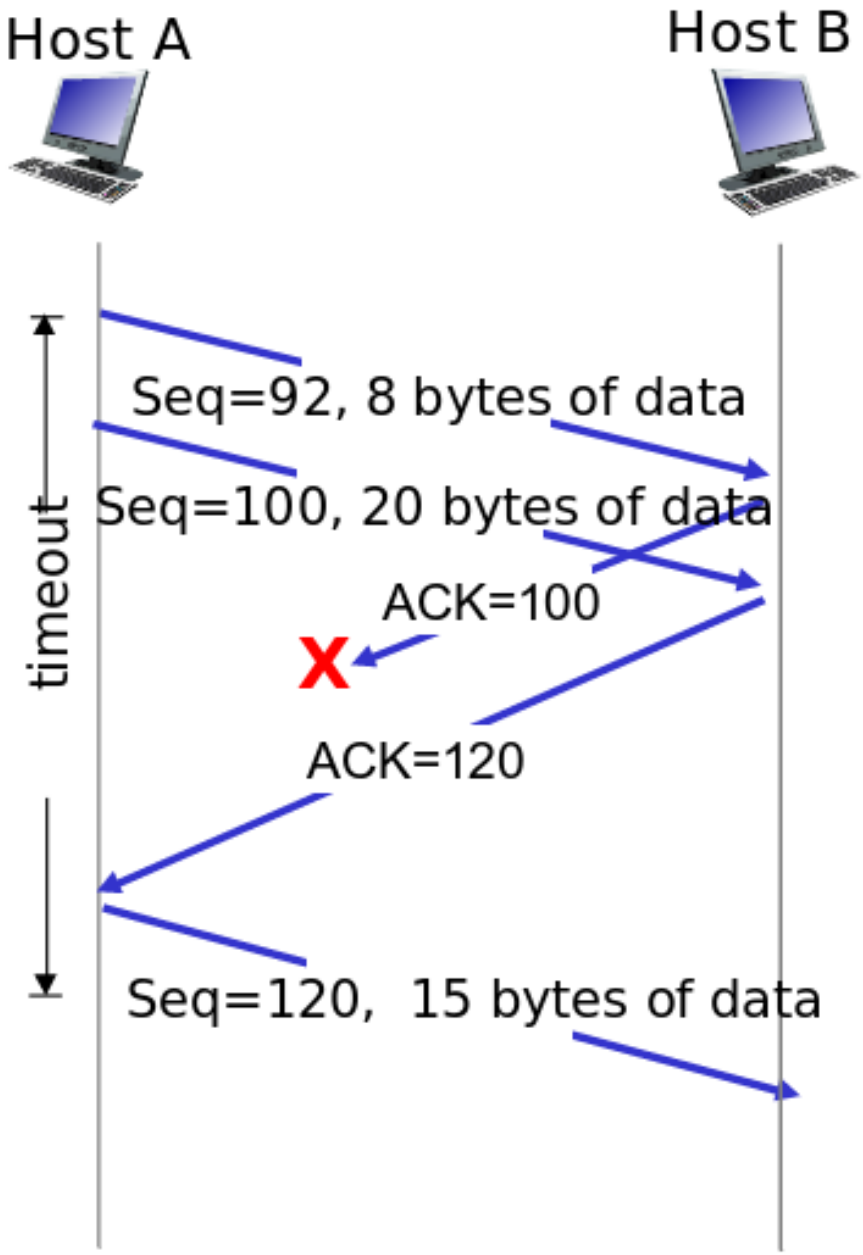


confirmação perdida



timeout prematuro

TCP + Timeouts



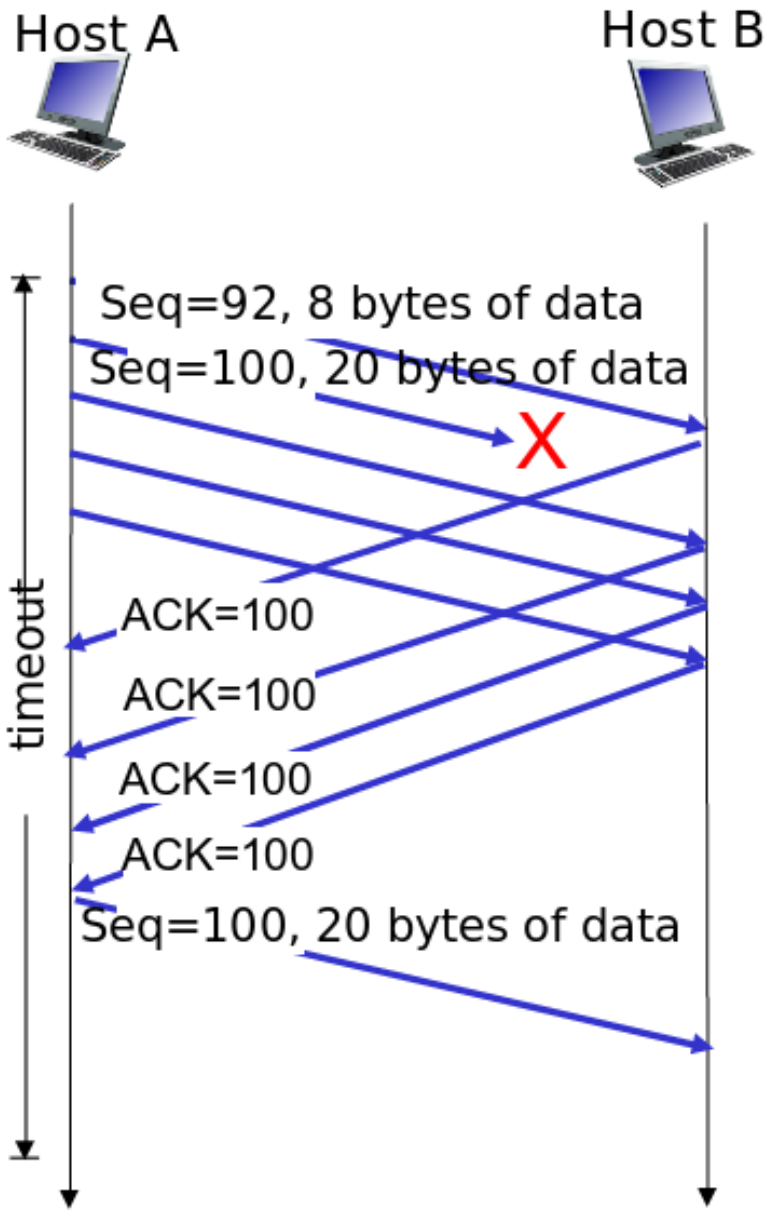
TCP - geração de confirmações

1. recebimento em ordem com núm. de seq. esperado, dados antigos já confirmados $\Rightarrow$ atrasa a confirmação, espera até 500ms pelo próximo segmento, depois, se não chegar um segmento, envia a confirmação.
2. recebimento em ordem, mas já havia um segmento com confirmação pendente (item 1) $\Rightarrow$ envia imediatamente a confirmação (cumulativa).
3. chegada de segmento fora de ordem; *gap* detectado $\Rightarrow$ imediatamente envia uma confirmação (duplicada)
4. chegada de um segmento no início de um *gap* $\Rightarrow$ imediatamente envia uma confirmação

Retransmissão Rápida

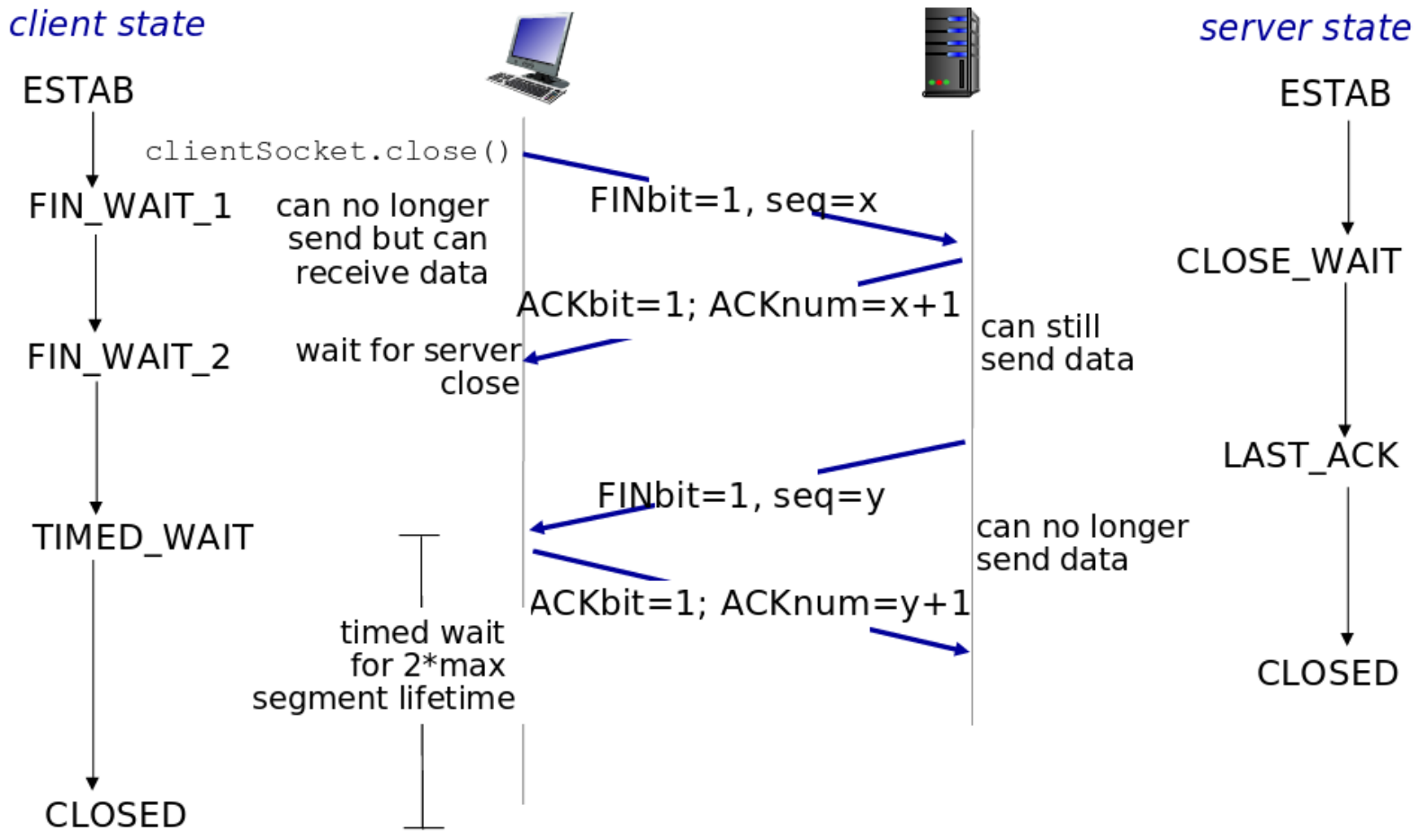
- *timeouts* são longos
- mas pode detectar segmentos perdidos pelas confirmações duplicadas
- se receber 3 confirmações duplicadas, então retransmite o segmento com menor número de sequência ainda não confirmado.

Retransmissão Rápida



Retransmissão rápida após 3 ACKs repetidos

Finalização da Conexão



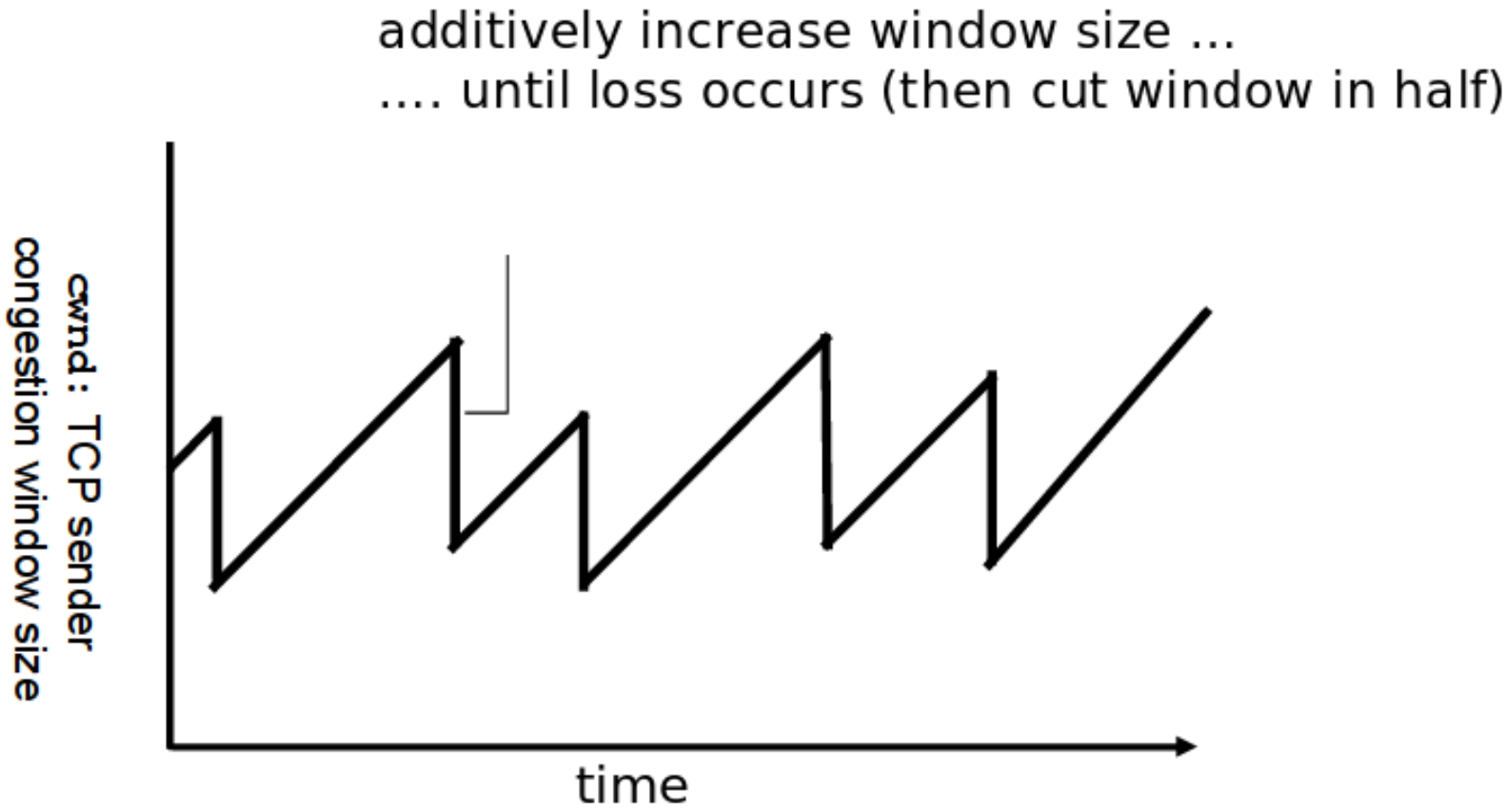
Controle de congestionamento

- Tamanho da Janela de Congestionamento (*congestion window* ou *cwnd*)
- Mantém $LastByteSent - LastByteAcked \leq cwnd$
- *cwnd* é modificado pela percepção de congestionamento

Controle de congestionamento

- aumenta o tamanho da janela de transmissão até detectar perdas de pacotes
- aumento aditivo: aumenta de 1 MSS (tamanho máximo de segmento) a cada RTT.
- diminui pela metade quando detecta perda.
- $rate = \frac{cwnd}{RTT}\ bytesseg$

Controle de congestionamento



Controle de congestionamento - *Slow start*

- Início 'lento' (TCP *slow start*)
- Início: $cwnd = 1$
- $cwnd = 2 * cwnd$ a cada RTT
- Na verdade, aumenta de 1 MSS a cada ACK recebido
- A janela começa pequena, mas cresce exponencialmente

até atingir um limiar ou até detectar perda de pacotes.

Versões do TCP

- Diferentes implementações do TCP implementam diferentes mecanismos de controle de congestionamento.
- Exemplos:
  - Tahoe e Reno: começam com *start slow* depois entram no estado de *congestion avoidance*
  - TCP Tahoe: em caso de *timeout* ou 3 ACKs iguais faz a retransmissão rápida e define o limiar do *slow start* para a metade do tamanho da janela (*cwnd*), define o tamanho da janela em 1 MSS e entra em *slow start*.
  - TCP Reno: em caso de 3 ACKs iguais, ele reduz pela metade o tamanho da janela de congestionamento e atribui ao limiar do *slow start* o tamanho da janela de congestionamento e entra em um estado de recuperação rápida onde aguarda os ACKs dos pacotes em trânsito. Caso não cheguem, entra em *slow start*.
  - Outras implementações: TCC Vegas, new Reno, CUBIC.