

## Aula 12

### Professores:

*Anna Dolejsi Santos (UFF)*

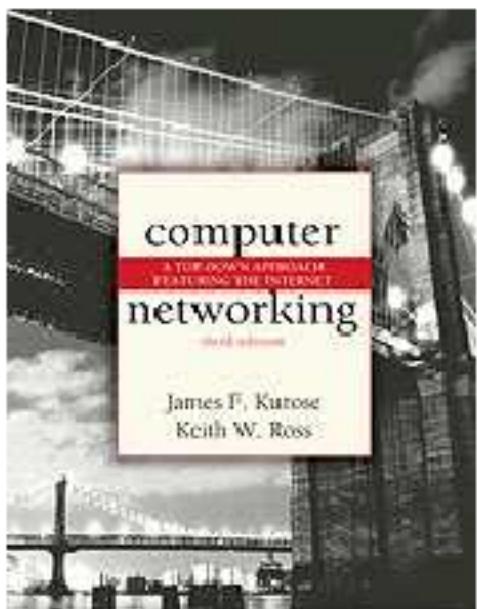
*Célio Vinicius Neves de Albuquerque (UFF)*

## TCP - Parte I

### Conteúdo:

- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável

## Livro Texto



### REDES DE COMPUTADORES E A INTERNET UMA ABORDAGEM TOP-DOWN

([www.aw.com/kurose\\_br](http://www.aw.com/kurose_br))

**James F. Kurose e Keith W. Ross**

**Copyright: 2006 - 3a. Edição**

**ISBN: 8588639181**

<http://www.pearson.com.br/>

### Referência Adicional:

2. **Computer Networks**, Andrew Tanenbaum - Capítulos 1-6.

Obs: As figuras que não têm referências pertencem ao material disponilizado pelo autor do livro texto ou foram produzidas pelo professor desta disciplina.

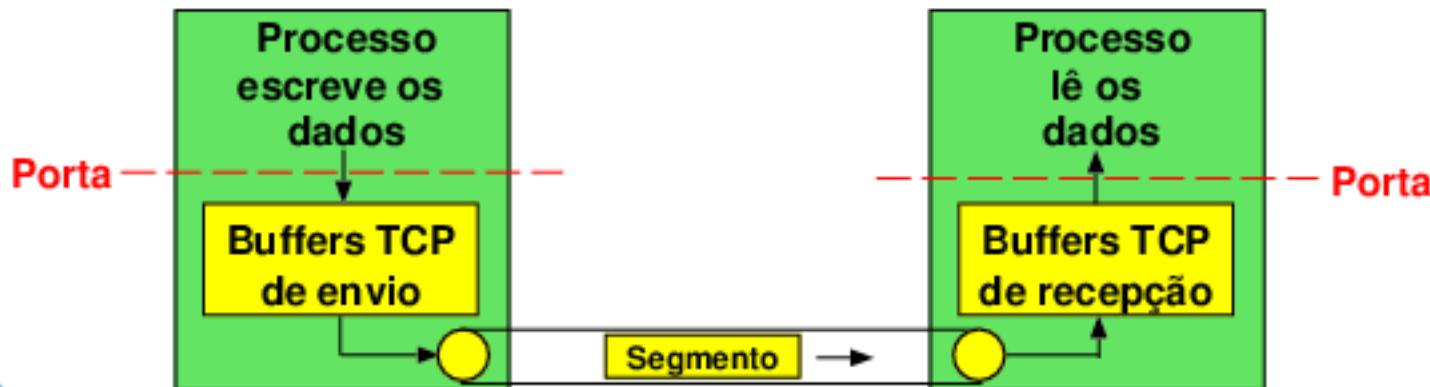
## Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado a conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável
  - controle de fluxo
  - gerenciamento de conexões
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

# TCP: Visão geral

RFCs: 793, 1122, 1323, 2018, 2581

- ponto a ponto:
    - 1 remetente, 1 receptor
  - fluxo de bytes, ordenados, confiável:
    - não estruturado em msgs
  - com paralelismo (*pipelined*):
    - tam. da janela ajustado por controle de fluxo e congestionamento do TCP
  - buffers de envio e recepção
- 
- transmissão full duplex:
    - fluxo de dados bi-direcional na mesma conexão
    - MSS: tamanho máximo de segmento
  - orientado a conexão:
    - handshaking (troca de msgs de controle) inicia estado de remetente, receptor antes de trocar dados
  - fluxo controlado:
    - receptor não será afogado



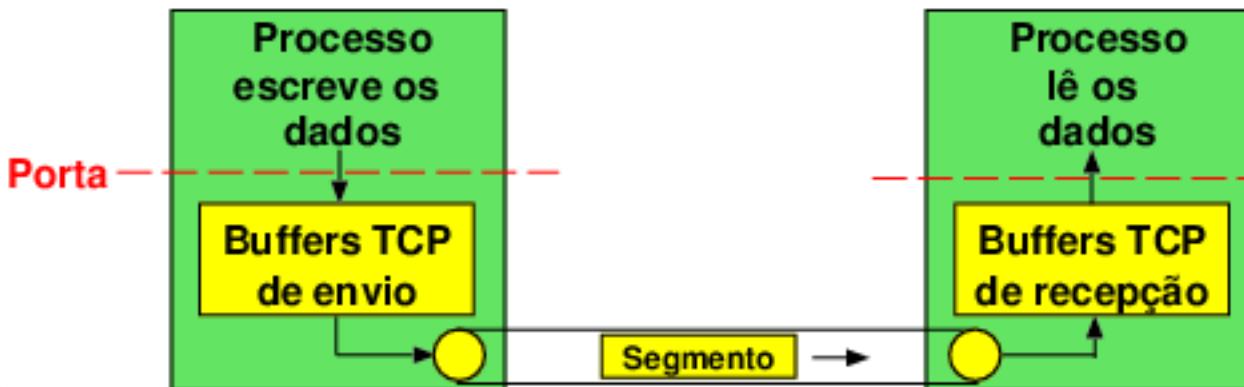
# TCP: Visão geral

RFCs: 793, 1122, 1323, 2018, 2581

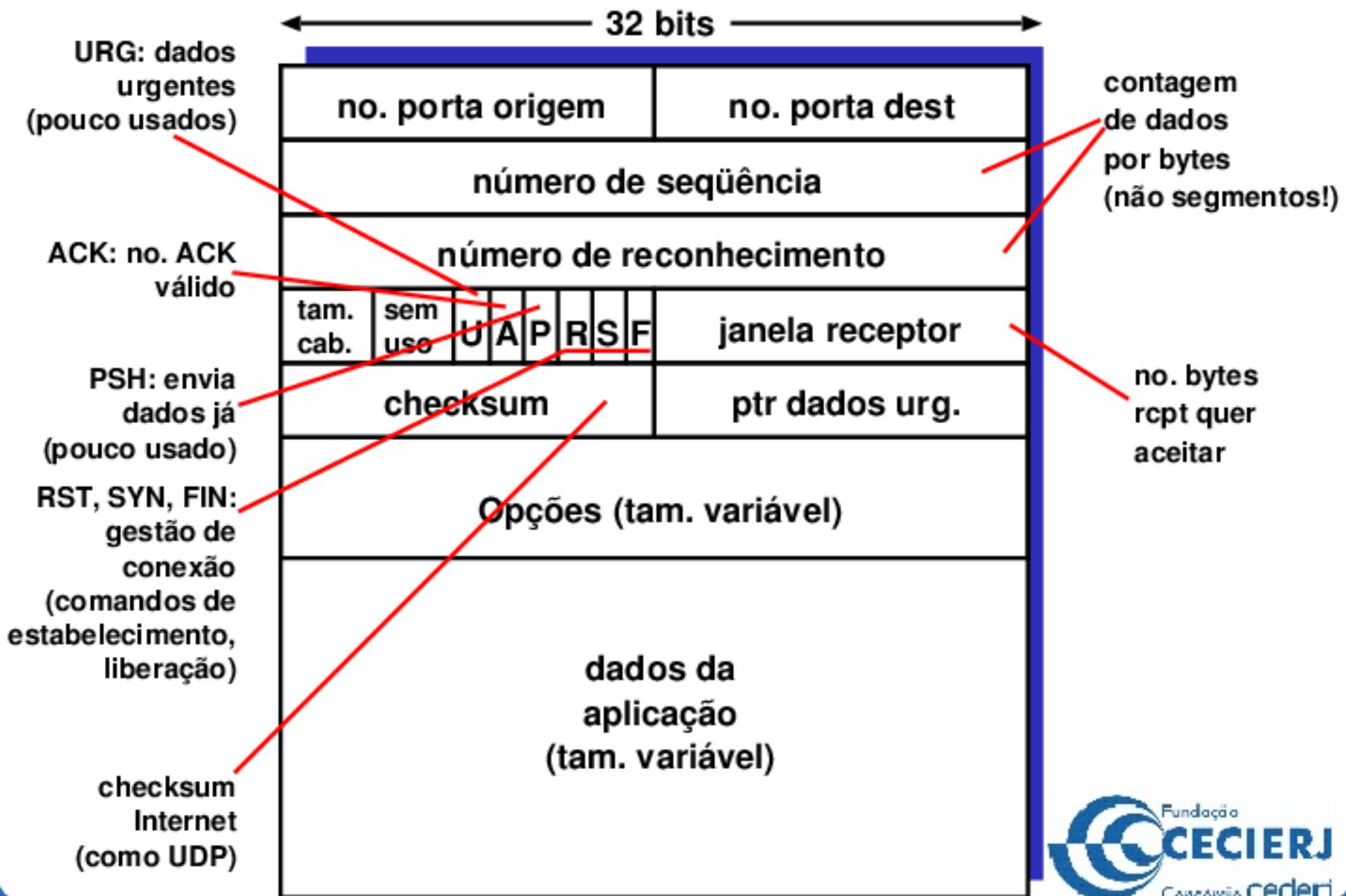
- ponto a ponto:
    - 1 remetente, 1 receptor
  - fluxo de bytes, ordenados, confiável:
    - não estruturado em msgs
  - com paralelismo (*pipelined*):
    - tam. da janela ajustado por controle de fluxo e congestionamento do TCP
  - buffers de envio e recepção
- 
- transmissão full duplex:
    - fluxo de dados bi-direcional na mesma conexão
    - MSS: tamanho máximo de segmento
  - orientado a conexão:
    - handshaking (troca de msgs de controle) inicia estado de remetente, receptor antes de trocar dados
  - fluxo controlado:
    - receptor não será afogado

OBS.: Na verdade, a mensagem passada pela aplicação é como se fosse uma seqüência de BYTES ordenados.

Porta



## TCP: estrutura do segmento



## TCP: nos. de seq. e ACKs

### Nos. de seq.:

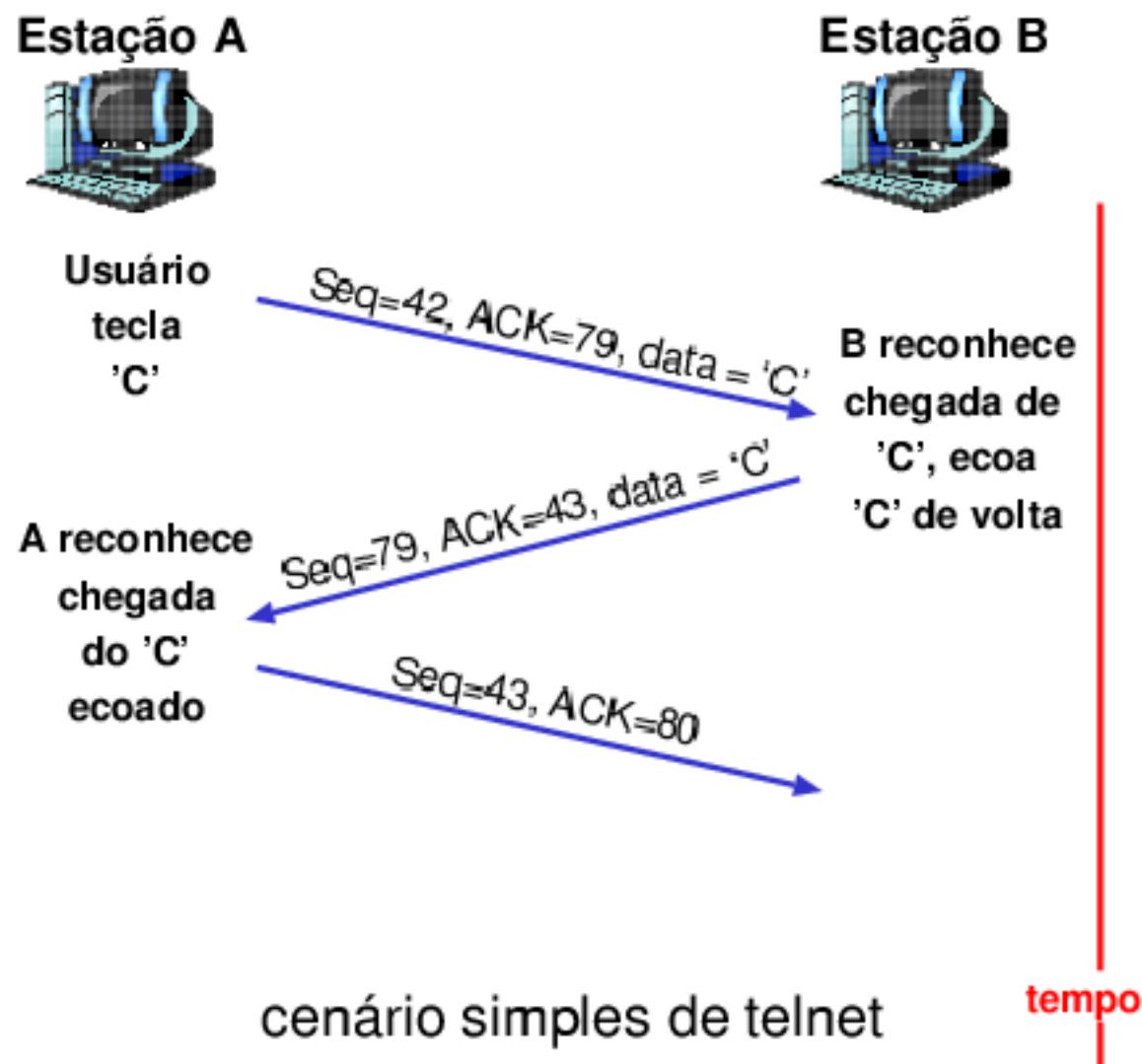
- "número" dentro do fluxo de bytes do primeiro byte de dados do segmento

### ACKs:

- no. de seq do próx. byte esperado do outro lado
- ACK cumulativo

P: como receptor trata segmentos fora da ordem?

- R: espec do TCP omissa - deixado ao implementador



## TCP: Tempo de Resposta (RTT - Round Trip Time) e Temporização

**P:** como escolher valor do temporizador TCP?

- maior que o RTT
  - note: RTT pode variar**
- muito curto: temporização prematura
  - retransmissões são desnecessárias**
- muito longo: reação demorada à perda de segmentos

**P:** como estimar RTT?

- RTT<sub>amostra</sub>:** tempo medido entre a transmissão do segmento e o recebimento do ACK correspondente
  - ignora retransmissões**
- RTT<sub>amostra</sub>** vai variar, queremos "amaciador" de RTT estimado
  - usa várias medições recentes, não apenas o valor corrente (RTT<sub>amostra</sub>)**

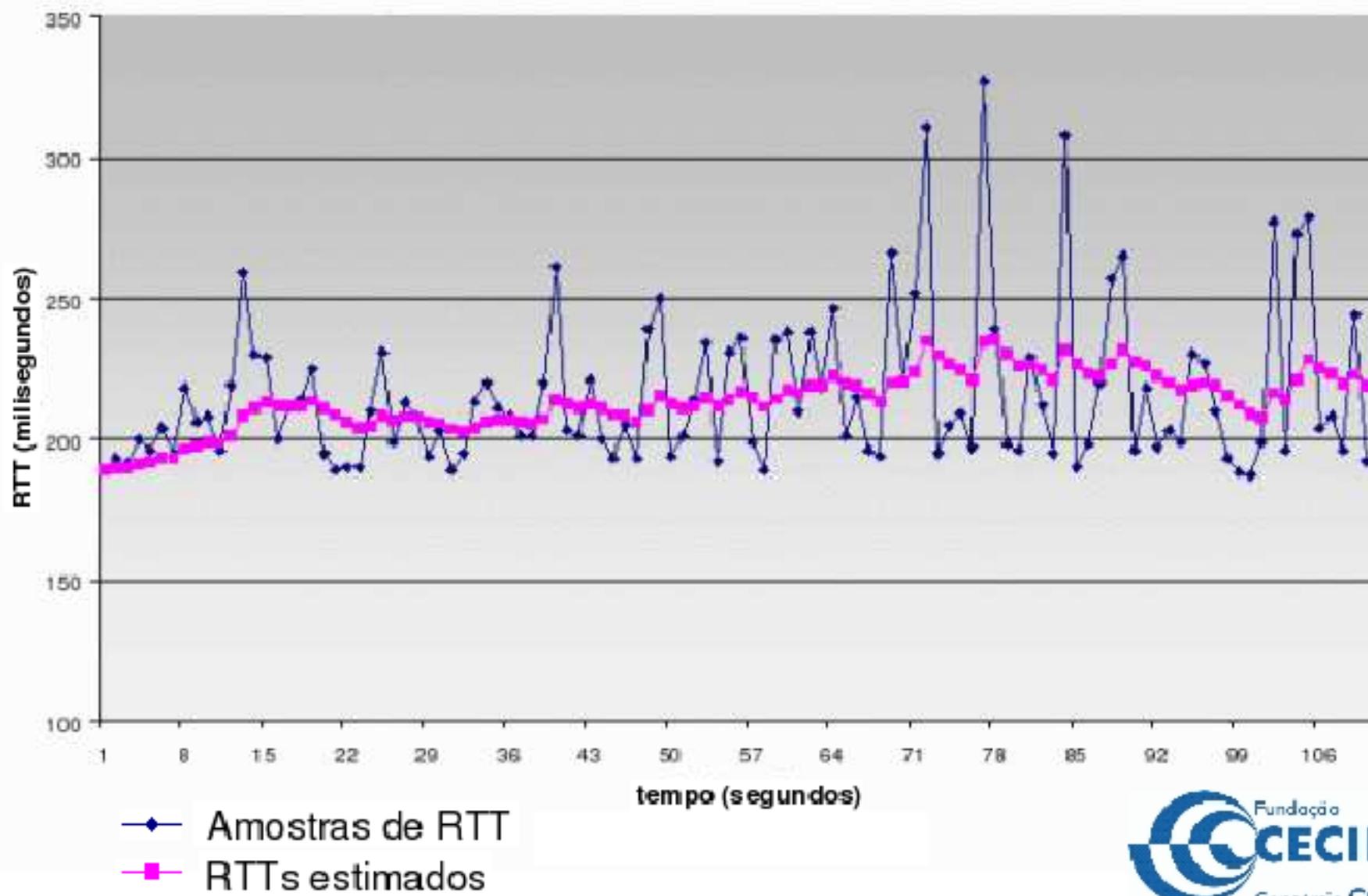
## TCP: Tempo de Resposta (RTT) e Temporização

$RTT_{estimado} = (1-\alpha) * RTT_{estimado} + \alpha * RTT_{amostra}$

- média corrente exponencialmente ponderada
- influência de cada amostra diminui exponencialmente com o tempo
- valor típico de  $\alpha = 0,125$

## Exemplo de estimativa do RTT:

RTT: [gaia.cs.umass.edu](http://gaia.cs.umass.edu) to [fantasia.eurecom.fr](http://fantasia.eurecom.fr)



## TCP: Tempo de Resposta (RTT) e Temporização

### Escolhendo o intervalo de temporização

- RTT\_estimado mais uma "margem de segurança"
  - grande variação no RTT\_estimado
    - > maior margem de segurança
- primeiro estima o quanto a RTTamostra desvia do RTT\_estimado:

$$\text{Desvio_RTT} = (1-\beta) * \text{Desvio_RTT} + \beta * |\text{RTT_amostra} - \text{RTT_estimado}|$$

- Então, seta o temporizador para:

$$\text{Temporização} = \text{RTT_estimado} + 4 * \text{Desvio_RTT}$$

## Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 UDP: Transporte não orientado a conexão
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável**
  - controle de fluxo**
  - gerenciamento de conexões**
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

## Transferência de dados confiável do TCP

- O TCP cria um serviço rdt sobre o serviço não confiável do IP
- Segmentos em série (*pipelined*)
- Ack's cumulativos
- O TCP usa um único temporizador para retransmissões
- As retransmissões são disparadas por:
  - estouros de temporização
  - acks duplicados
- Considere inicialmente um transmissor TCP simplificado:
  - ignora acks duplicados
  - ignora controles de fluxo e de congestionamento

## Eventos do transmissor TCP:

### Dados recebidos da apl.:

- Cria segmento com no. de seqüência (nseq)
- nseq é o número de seqüência do primeiro byte do segmento
- Liga o temporizador se já não estiver ligado (temporização do segmento mais antigo ainda não reconhecido)
- Valor do temporizador: calculado anteriormente

### estouro do temporizador:

- Retransmite o segmento que causou o estouro do temporizador
- Reinicia o temporizador

### Recepção de Ack:

- Se reconhecer segmentos ainda não reconhecidos
  - atualizar informação sobre o que foi reconhecido
  - religa o temporizador se ainda houver segmentos pendentes (não reconhecidos)

## Transmissor TCP (simplificado)

### Comentário:

- **SendBase-1**: último byte reconhecido cumulativamente

### Exemplo:

- **SendBase-1 = 71;**  
y= 73, portanto o receptor quer receber 73+;
- **y > SendBase,**  
portanto novos dados foram reconhecidos.

NextSeqNum = número de seqüência inicial  
SendBase = número de seqüência inicial

```
repita (sempre) {
    switch(event)
```

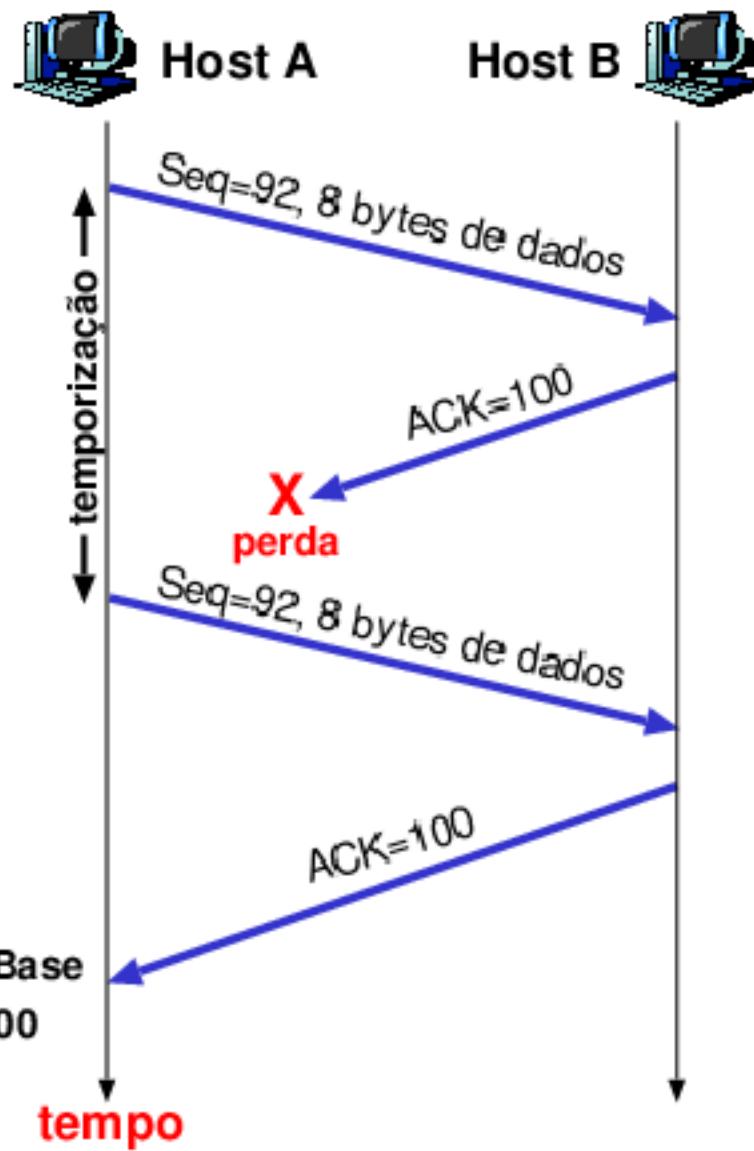
**event:** dados recebidos da aplicação acima  
cria segmento TCP com número de seqüência NextSeqNum  
**se** (temporizador estiver desligado)  
    liga o temporizador  
    passa segmento para IP  
    NextSeqNum = NextSeqNum + comprimento(dados)

**event:** estouro do temporizador  
retransmite segmento ainda não reconhecido com o menor número de seqüência  
reinicia o temporizador

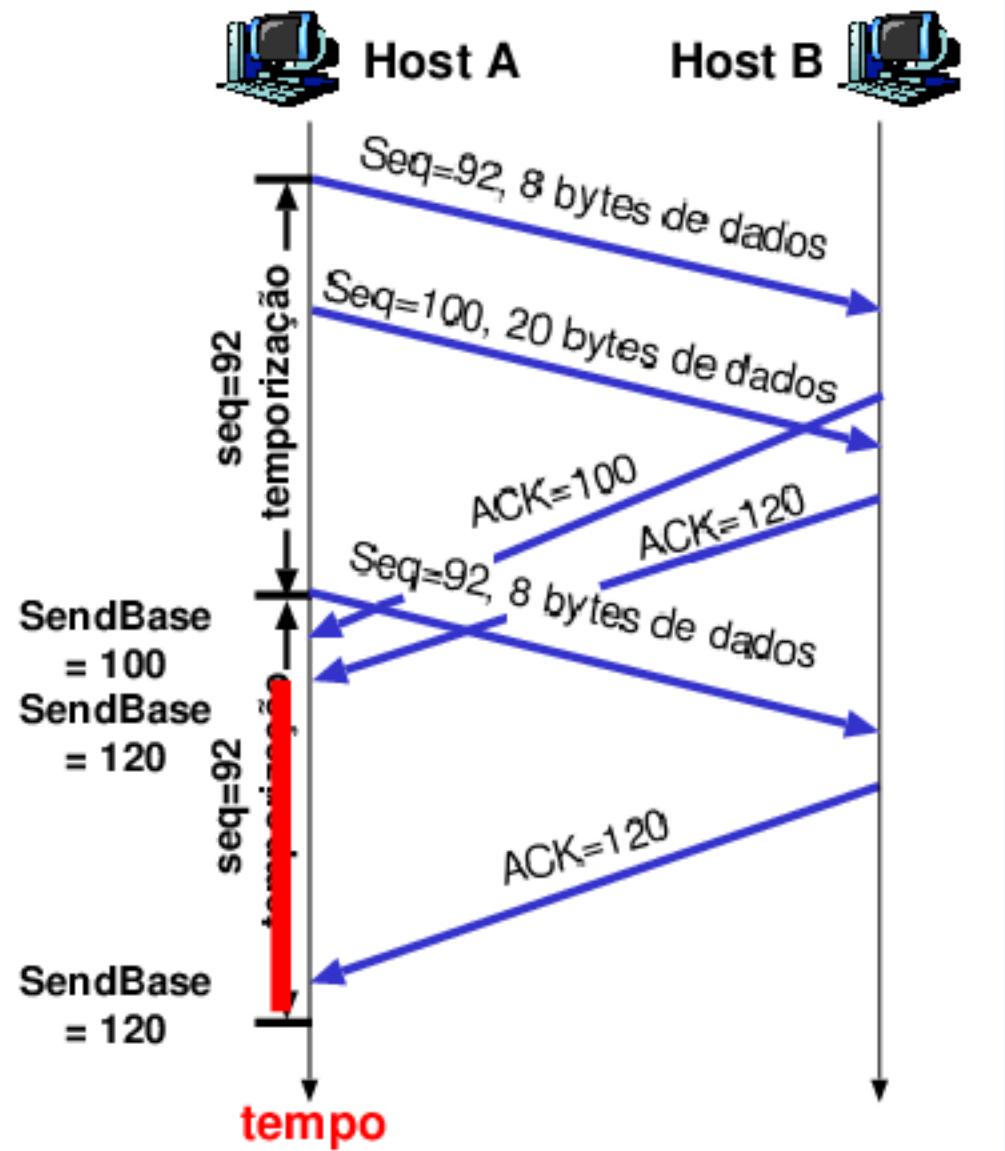
**event:** ACK recebido, com valor de campo ACK de y  
**se** (y > SendBase) { /\* ACK cumulativo de todos dados até y \*/  
    SendBase = y  
    **se** (houver segmentos ainda não reconhecidos)  
        liga o temporizador  
    } senão desliga o temporizador  
}

/\* fim do repita sempre \*/

## TCP: cenários de retransmissão

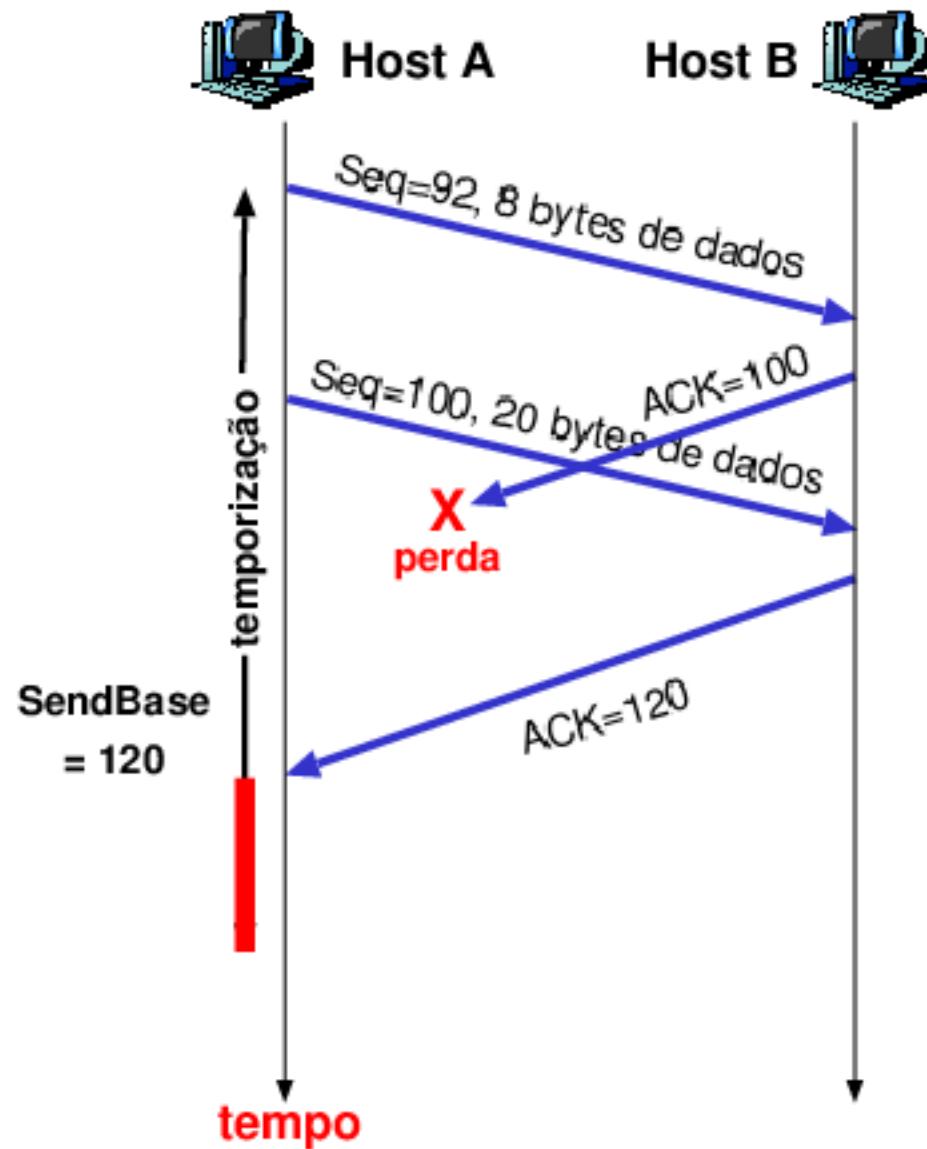


cenário de perda de ACK



estouro prematuro do temporizador

## TCP: cenários de retransmissão (mais)



Cenário de ACK cumulativo

## TCP geração de ACKs [RFCs 1122, 2581]

Evento no Receptor	Ação do Receptor TCP
chegada de segmento em ordem sem lacunas, anteriores já reconhecidos	ACK retardado. Espera até <b>500ms</b> p/ próx. segmento. Se não chegar segmento, envia ACK
chegada de segmento em ordem sem lacunas, um ACK retardado pendente	envia imediatamente um único ACK cumulativo
chegada de segmento fora de ordem, com no. de seq. maior que esperado -> lacuna	envia ACK duplicado, indicando no. de seq. do próximo byte esperado
chegada de segmento que preenche a lacuna parcial ou completamente	ACK imediato se segmento no início da lacuna

## Retransmissão rápida

- ❑ O intervalo do temporizador é freqüentemente bastante longo:
  - **longo atraso antes de retransmitir um pacote perdido**
- ❑ Detecta segmentos perdidos através de ACKs duplicados.
  - **O transmissor normalmente envia diversos segmentos**
  - **Se um segmento se perder, provavelmente haverá muitos ACKs duplicados.**
- ❑ Se o transmissor receber 3 ACKs para os mesmos dados, ele supõe que o segmento após os dados reconhecidos se perdeu:
  - **Retransmissão rápida:**  
**retransmite o segmento antes que estoure o temporizador**

## Retransmissão rápida

- ❑ O intervalo do temporizador é freqüentemente bastante longo:
  - **longo atraso antes de retransmitir um pacote perdido**
- ❑ Detecta segmentos perdidos através de ACKs duplicados.
  - **O transmissor normalmente envia diversos segmentos**
  - **Se um segmento se perder, provavelmente haverá muitos ACKs duplicados.**
- ❑ Se o transmissor receber 3 ACKs para os mesmos dados, ele supõe que o segmento após os dados reconhecidos se perdeu:
  - **Retransmissão rápida:** retransmite o segmento antes que estoure o temporizador

**OBS.:** Na verdade, o transmissor TCP simplificado não fazia controle de fluxo e também não ficava contando confirmações duplicadas que ele RECEBEU PARA O mesmo segmento.

## Algoritmo de retransmissão rápida:

```
event: recebido ACK, com valor do campo ACK de y
    if (y > SendBase) {
        SendBase = y
        if (houver segmentos ainda não reconhecidos)
            liga temporizador
        else desliga temporizador
    }
    else {
        incrementa contador de ACKs duplicados recebidos para y
        if (contador de ACKs duplicados recebido para y = 3) {
            retransmita segmento com número de seqüência y
        }
    }
}
```

um ACK duplicado para um  
segmento já reconhecido

Retransmissão rápida

## Conteúdo do Capítulo 3

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado a conexão: UDP
- 3.4 Princípios da transferência confiável de dados
- 3.5 Transporte orientado a conexão: TCP
  - transferência confiável
  - controle de fluxo
  - gerenciamento de conexões
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP