Redes de computadores II

Aula 7 – TCP.

Prof. Felipe Leite Lobo
felipe.lobo@ufrr.br

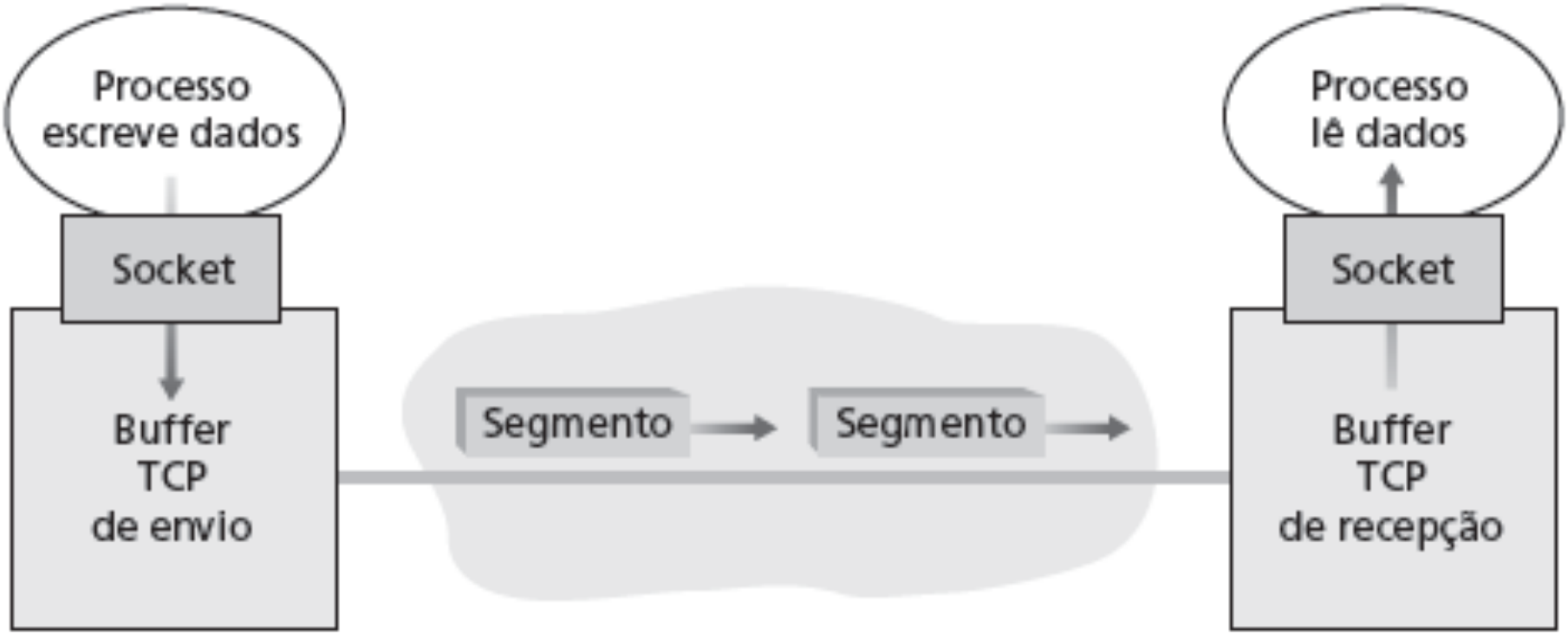
TCP: Visão geral

RFCs: 793, 1122, 1323, 2018, 2581

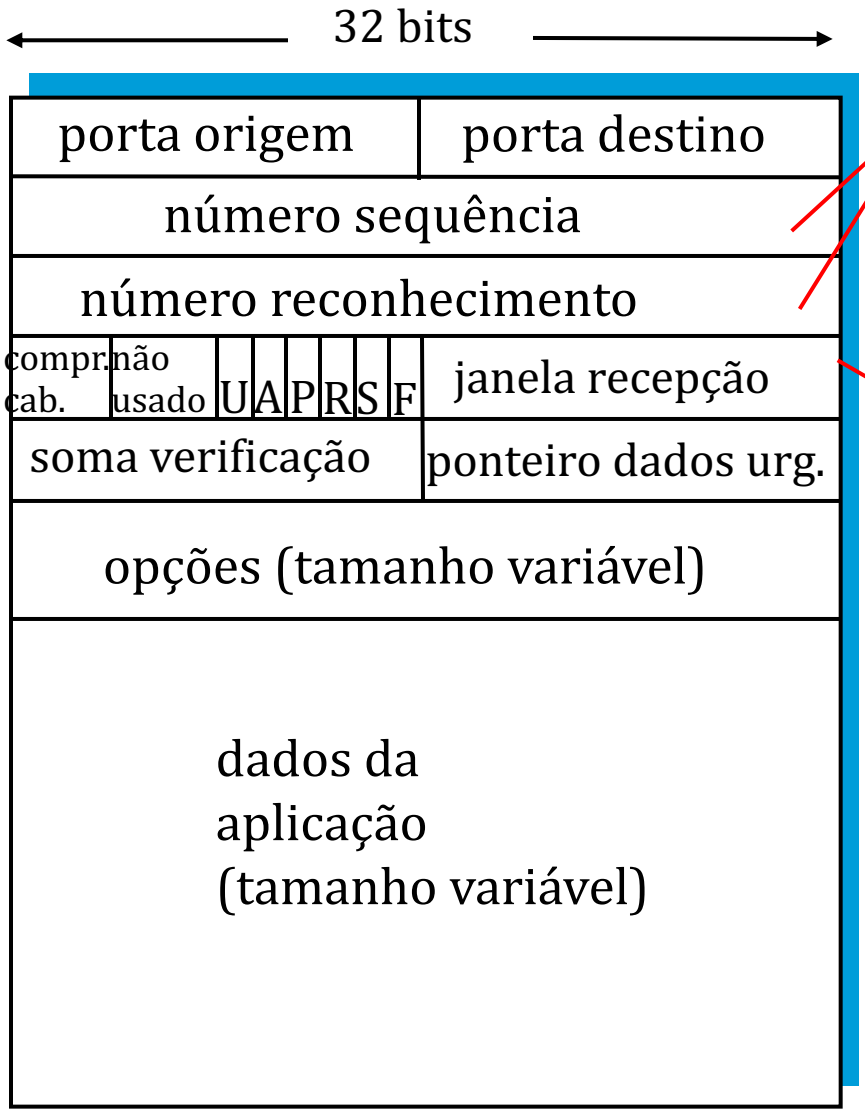
- Características do TCP:
  - orientado a conexão;
  - ponto a ponto;
  - cadeia de *bytes* confiável, em ordem*;*
  - paralelismo;
  - dados *full-duplex*;
  - fluxo controlado;

TCP

- *buffers* de envio e recepção;



Estrutura do segmento TCP



- URG: dados urgentes (quase não usado)
- ACK: # ACK válido
- PSH: empurrar dados agora (quase não usado)
- RST, SYN, FIN: estabelece conexão
- soma de verificação da Internet (como em UDP)

contagem por bytes de dados (não segmentos!)

bytes destinatário pode aceitar

#s sequência e ACKs do TCP

#’s de sequência:

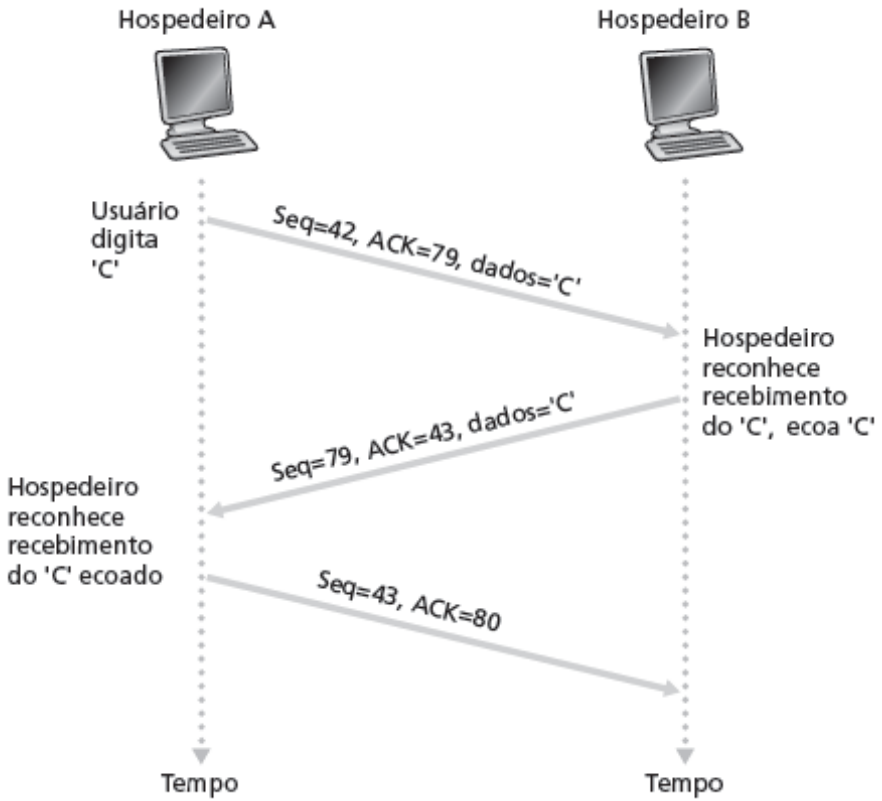
- “número” na cadeia de bytes do 1º byte nos dados do segmento

ACKs:

- # seq do próximo byte esperado do outro lado
- ACK cumulativo:

P: como o destinatário trata segmentos fora de ordem?

- R: TCP não diz – a critério do implementador



cenário telnet simples

Tempo de ida e volta e *timeout* do TCP

P: Como definir o valor de *timeout* do TCP?

- maior que RTT;
  - RTT varia;
- muito curto: *timeout* prematuro:
  - retransmissões desnecessárias;
- muito longo: baixa reação a perda de segmento;

P: Como estimar o RTT?

- **`SampleRTT`**: tempo medido da transmissão do segmento até receber o ACK;
  - ignora retransmissões;
- **`SampleRTT`** variará: RTT estimado "mais estável":
  - média de várias medições recentes, não apenas **`SampleRTT`** atual;

Tempo de ida e volta e *timeout* do TCP

definindo o *timeout*

- **EstimtedRTT** mais "margem de segurança"
  - grande variação em **EstimatedRTT ->** maior margem de segurança;
- primeira estimativa do quanto SampleRTT se desvia de EstimatedRTT:

$$\text{DevRTT} = (1-\beta)*\text{DevRTT} + \beta*|\text{SampleRTT}-\text{EstimatedRTT}|$$

(geralmente, $\beta = 0,25$)

depois definir intervalo de *timeout*

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4*\text{DevRTT}$$

Transferência confiável de dados no TCP

- TCP cria serviço rdt em cima do serviço não confiável do IP;
- segmentos em paralelo;
- ACKs cumulativos;
- TCP usa único temporizador de retransmissão;
- retransmissões são disparadas por:
  - eventos de *timeout;*
  - ACKs duplicados;
- inicialmente, considera remetente TCP simplificado:
  - ignora ACKs duplicados;
  - ignora controle de fluxo, controle de congestionamento;

Eventos de remetente TCP:

dados recebidos da aplicação:

- cria segmento com # seq;
- # seq # é número da cadeia de bytes do primeiro byte de dados no segmento;
- inicia temporizador, se ainda não tiver iniciado (pense nele como para o segmento mais antigo sem ACK);
- intervalo de expiração: `TimeOutInterval`

timeout:

- retransmite segmento que causou *timeout;*
- reinicia temporizador;

ACK recebido:

- Reconhecem-se segmentos sem ACK anteriores;
  - atualiza o que sabidamente tem ACK;
  - inicia temporizador se houver segmentos pendentes;

RemetenteTCP (simplificado)

```
 NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)

  event: data received from application above
      create TCP segment with sequence number NextSeqNum
      if (timer currently not running)
          start timer
      pass segment to IP
      NextSeqNum = NextSeqNum + length(dados)

  event: timer timeout
      retransmit not-yet-acknowledged segment with
          smallest sequence number
      start timer

  event: ACK received, with ACK field value of y
      if (y > SendBase) {
          SendBase = y
          if (there are currently not-yet-acknowledged segments)
              start timer
          }

} /* end of loop forever */
```

Comentário:

- SendBase-1: último byte cumulativo com ACK

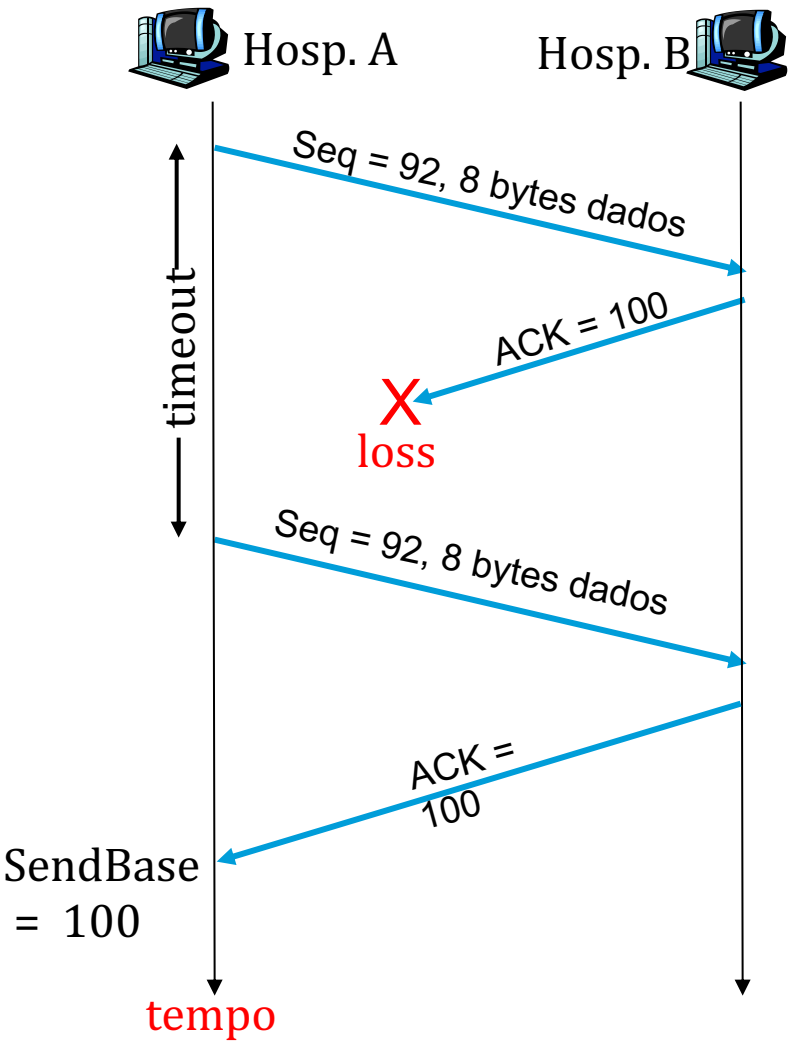
Exemplo:

- SendBase-1 = 71;

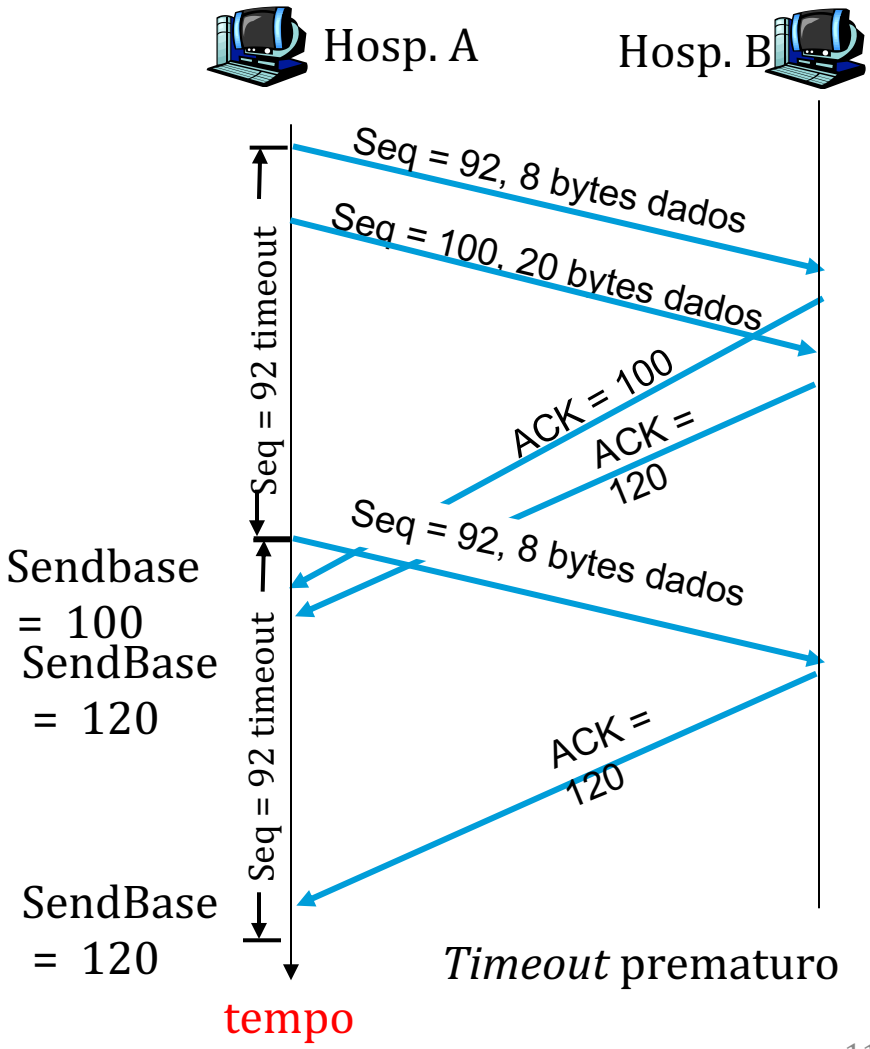
y = 73, de modo que destinatário deseja 73+ ;

y > SendBase, de modo que novos dados têm ACK

TCP: cenários de retransmissão

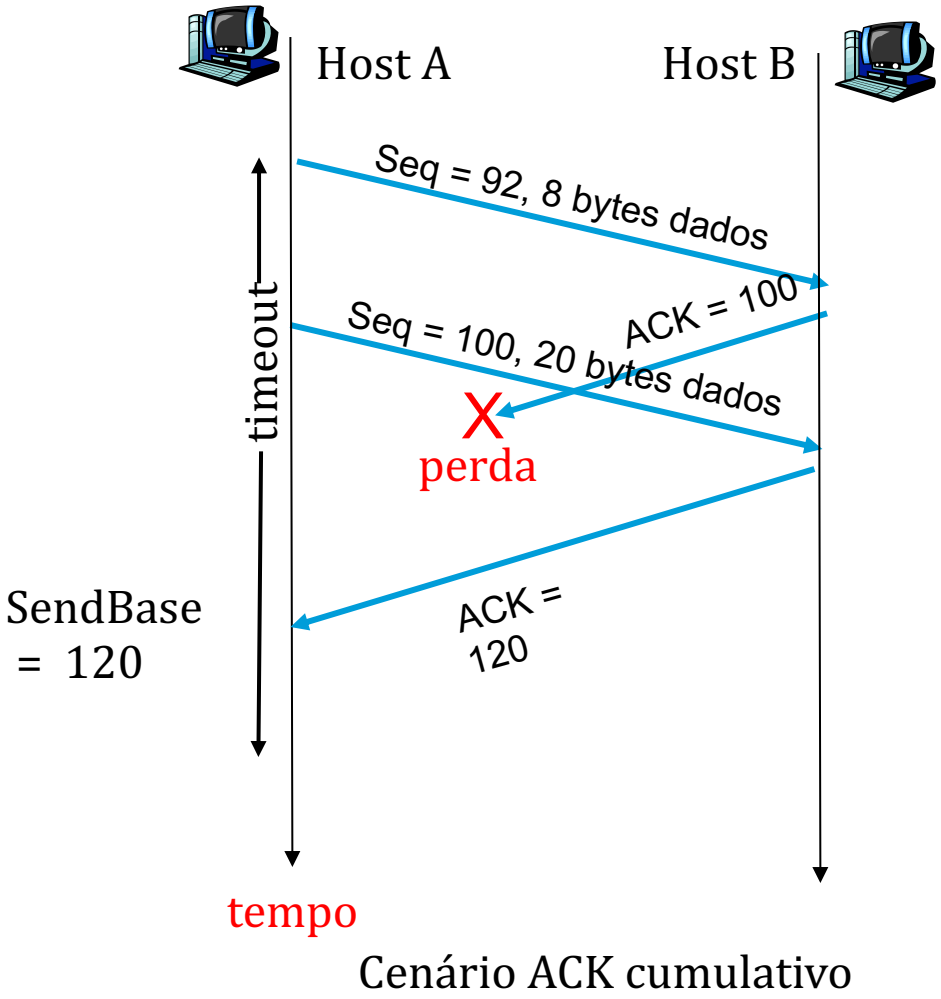


Cenário de ACK perdido



Timeout prematuro

TCP: cenários de retransmissão



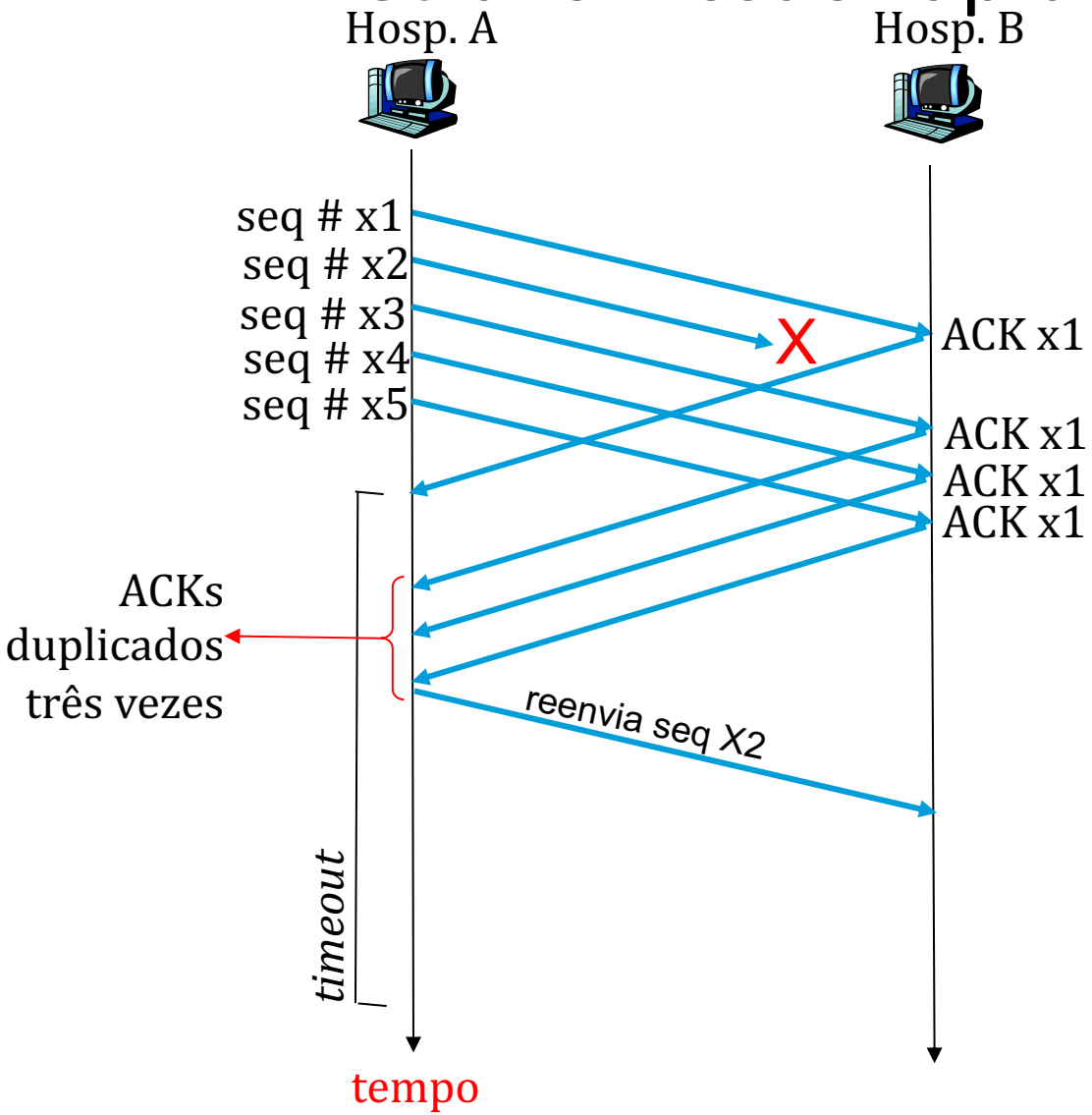
Cenário ACK cumulativo

Retransmissão rápida

- Período de *timeout* relativamente grande:
  - longo atraso antes de reenviar pacote perdido;
- Detecta segmentos perdidos por meio de ACKs duplicados:
  - remetente geralmente envia muitos segmentos um após o outro;
  - se segmento for perdido, provavelmente haverá muitos ACKs duplicados para esse segmento;

- Se o remetente recebe 3 ACKs para os mesmos dados, ele supõe que segmento após dados com ACK foi perdido:
  - retransmissão rápida: reenvia segmento antes que o temporizador expire;

Retransmissão rápida



Algoritmo de retransmissão rápida:

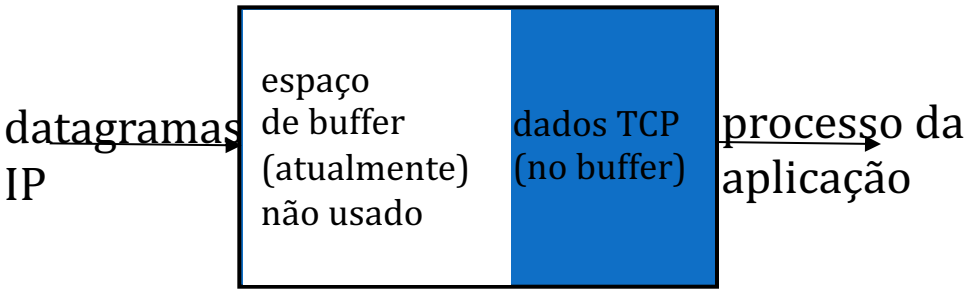
```
event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase = y
                if (there are currently not-yet-acknowledged segments)
                    start timer
              }
            else {
                increment count of dup ACKs received for y
                if (count of dup ACKs received for y = 3) {
                    resend segment with sequence number y
                 }
```

ACK duplicado para segmento já com ACK

retransmissão rápida

Controle de fluxo TCP

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



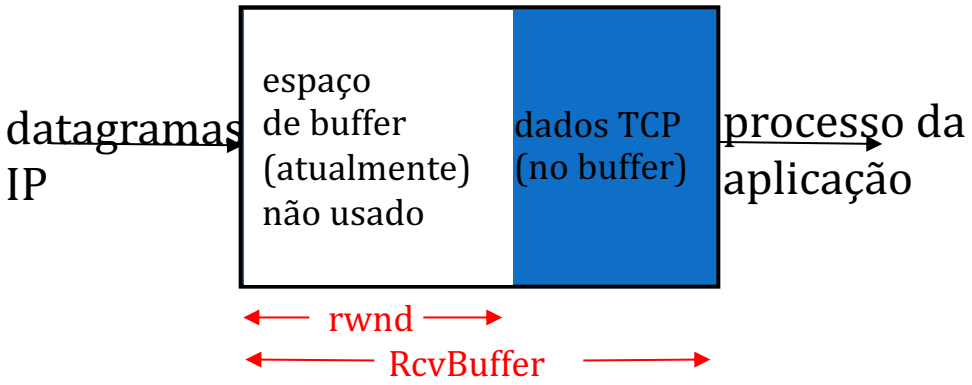
- processo da aplicação pode ser lento na leitura do buffer;

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



(suponha que destinatário TCP descarte segmentos fora de ordem)

- espaço de buffer não usado:

= rwnd

= RcvBuffer-[LastByteRcvd - LastByteRead]

- destinatário: anuncia espaço de buffer não usado incluindo valor de `rwnd` no cabeçalho do segmento
- remetente: limita # de bytes com ACK `rwnd`
  - garante que buffer do destinatário não estoura

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 (Ex: **`RcvWindow`**)
- *cliente:* inicia a conexão:

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

- *servidor:* contactado pelo cliente

```
Socket connectionSocket =
welcomeSocket.accept();
```

Gerenciamento da conexão TCP

apresentação de 3 vias (*three-way handshake*) :

etapa 1: cliente envia segmento SYN do TCP ao servidor

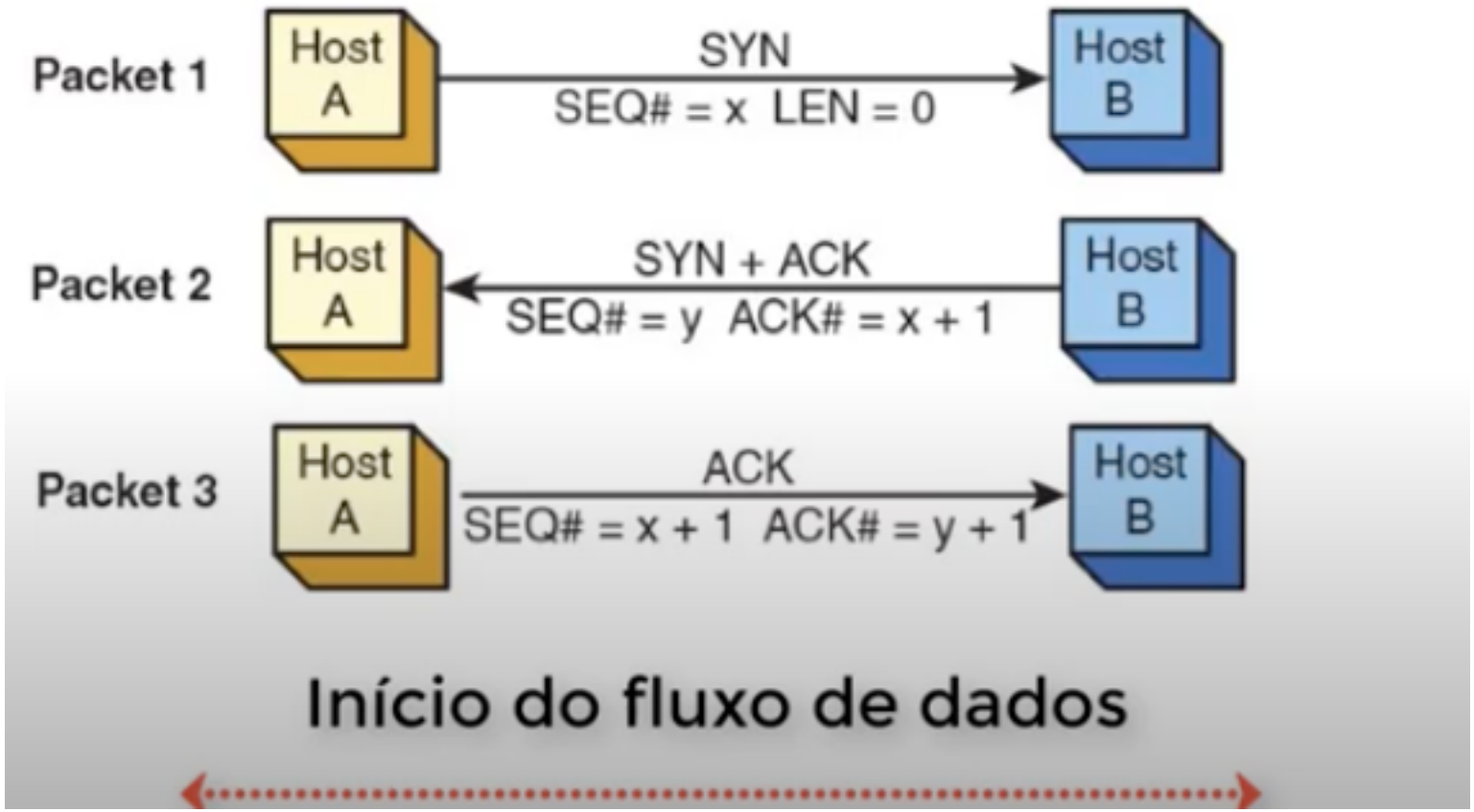
- especifica # seq. inicial
- sem dados

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.

Three-Way Handshake



Gerenciamento da conexão TCP

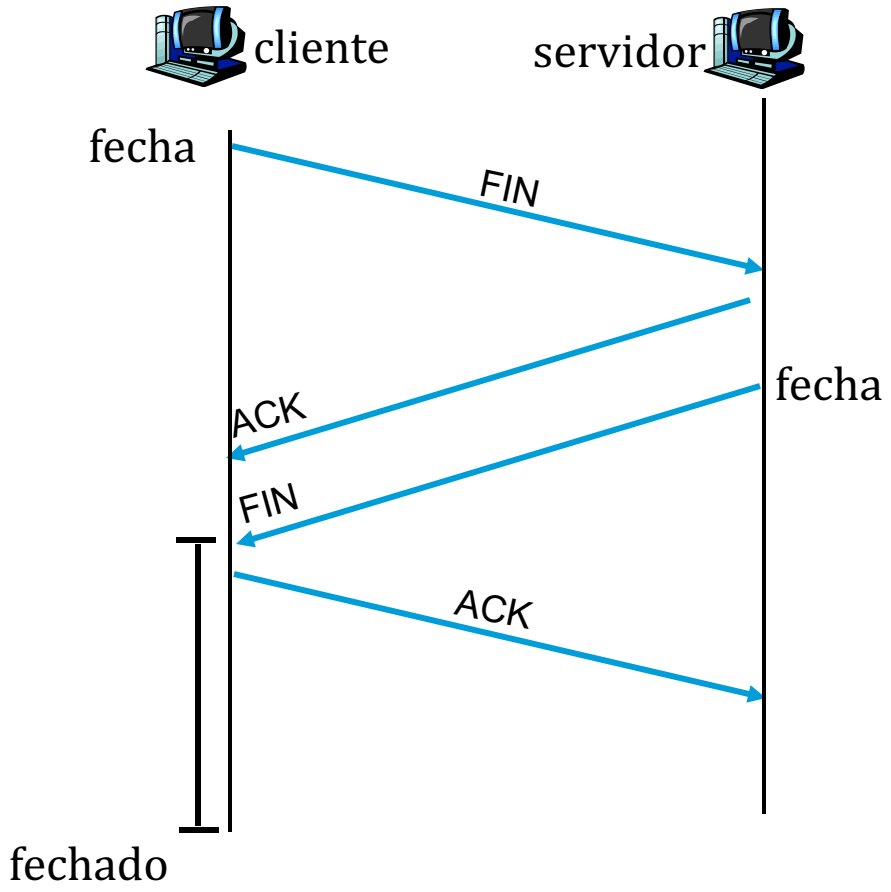
fechando uma conexão:

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.



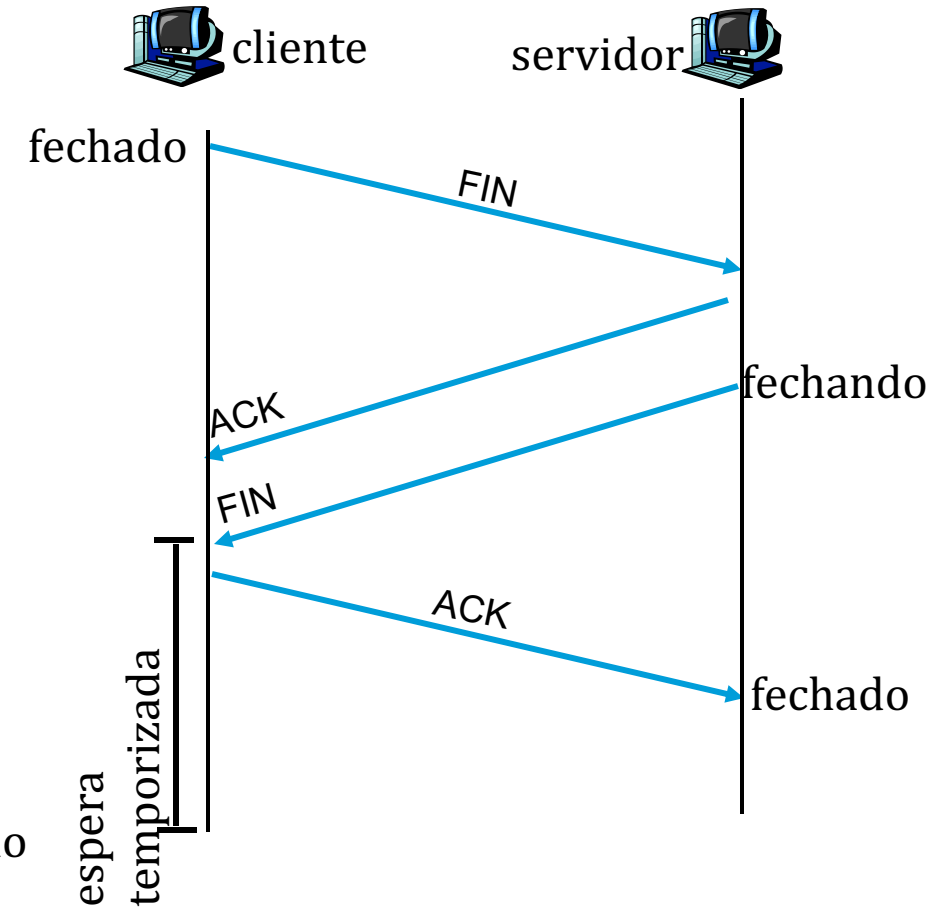
Gerenciamento da conexão TCP

etapa 3: cliente recebe FIN, responde com ACK

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

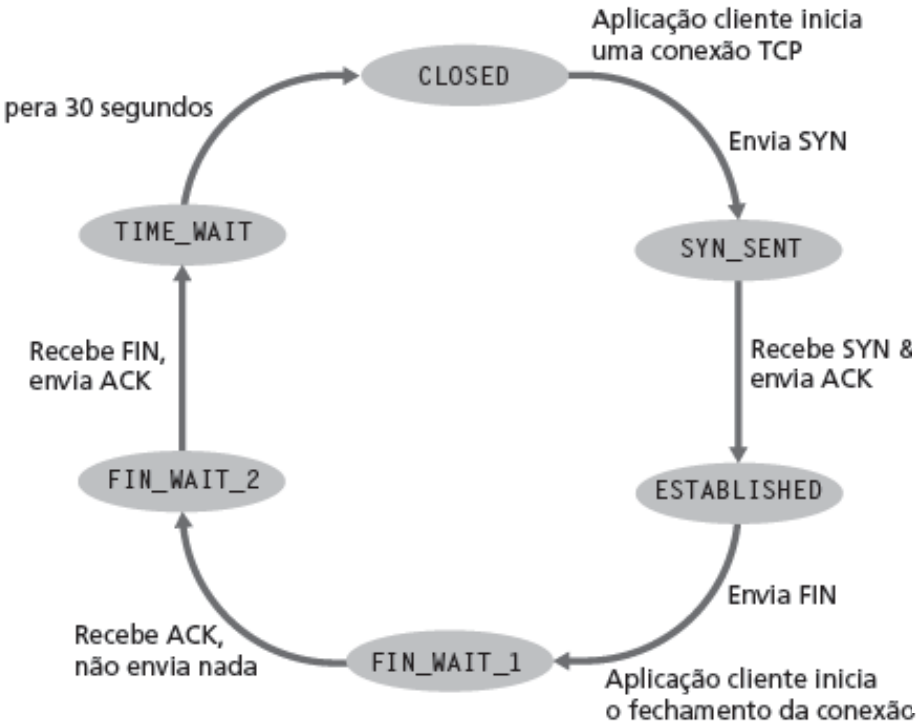
etapa 4: servidor recebe ACK - conexão fechada

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



Gerenciamento da conexão TCP

ciclo de vida do cliente TCP



ciclo de vida do servidor TCP