

Aula 10 - TCP: Confiabilidade, Controle de Fluxo, Gerenciamento

Diego Passos

Universidade Federal Fluminense

Redes de Computadores

Material adaptado a partir dos slides
originais de J.F Kurose and K.W. Ross.

Timeout do TCP

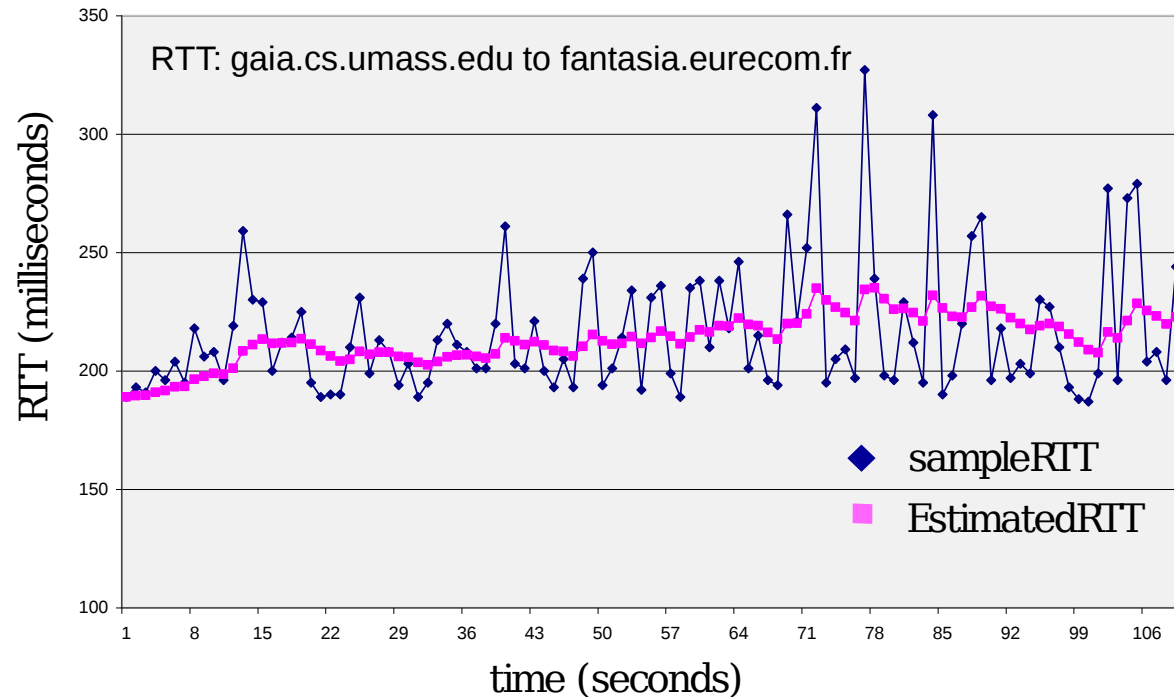
TCP: RTT, Timeout (I)

- **Pergunta:** como o *timeout* do temporizador do TCP é configurado?
 - Precisa ser mais longo que o RTT.
 - Mas RTT varia.
 - **Curto demais:** *timeout* prematuro, retransmissões desnecessárias.
 - **Longo demais:** reação lenta às perdas de segmentos.
- **Pergunta:** como estimar o RTT?
 - **SampleRTT:** tempo medido desde a transmissão do segmento até recepção do ACK.
 - Ignora retransmissões.
 - SampleRTT varia muito, é melhor ter uma estimativa mais “suave”.
 - Calcular média sobre várias amostras recentes, não usar apenas a última.

TCP: RTT, Timeout (II)

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

- Média Movente Exponencialmente Ponderada.
- Influência das amostras passadas decai exponencialmente.
- Valor típico de α : 0,125.




TCP: RTT, Timeout (III)

- **Timeout do temporizador:** EstimatedRTT mais “margem de segurança”.
 - Quanto maior o EstimatedRTT, maior a margem.
- Estima-se o **desvio** do SampleRTT em relação ao EstimatedRTT:

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

- Tipicamente, $\beta = 0,25$.

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

 ↑
estimated RTT ↑
“safety margin”

TCP: Transmissão Confiável de Dados

TCP: Transmissão Confiável de Dados

- TCP cria serviço de transmissão confiável sobre serviço não confiável do IP.
 - Utiliza pipeline.
 - ACKs cumulativos.
 - Único temporizador de retransmissão.
- Retransmissões disparadas por:
 - Eventos de *timeout*.
 - ACKs duplicados.
- Vamos considerar inicialmente um transmissor TCP simplificado.
 - Ignorando ACKs duplicados.
 - Ignorando controle de fluxo, controle de congestionamento.

Eventos do Transmissor TCP

- **Dados recebidos da aplicação:**

- Cria segmento com # de sequência.
- # de sequência é **índice da posição do primeiro byte no fluxo de dados da aplicação**.
- Inicia o temporizador, se ele já não estiver rodando.
 - Pense no temporizador como para o segmento mais antigo com ACK pendente.
 - Intervalo de expiração: TimeoutInterval.

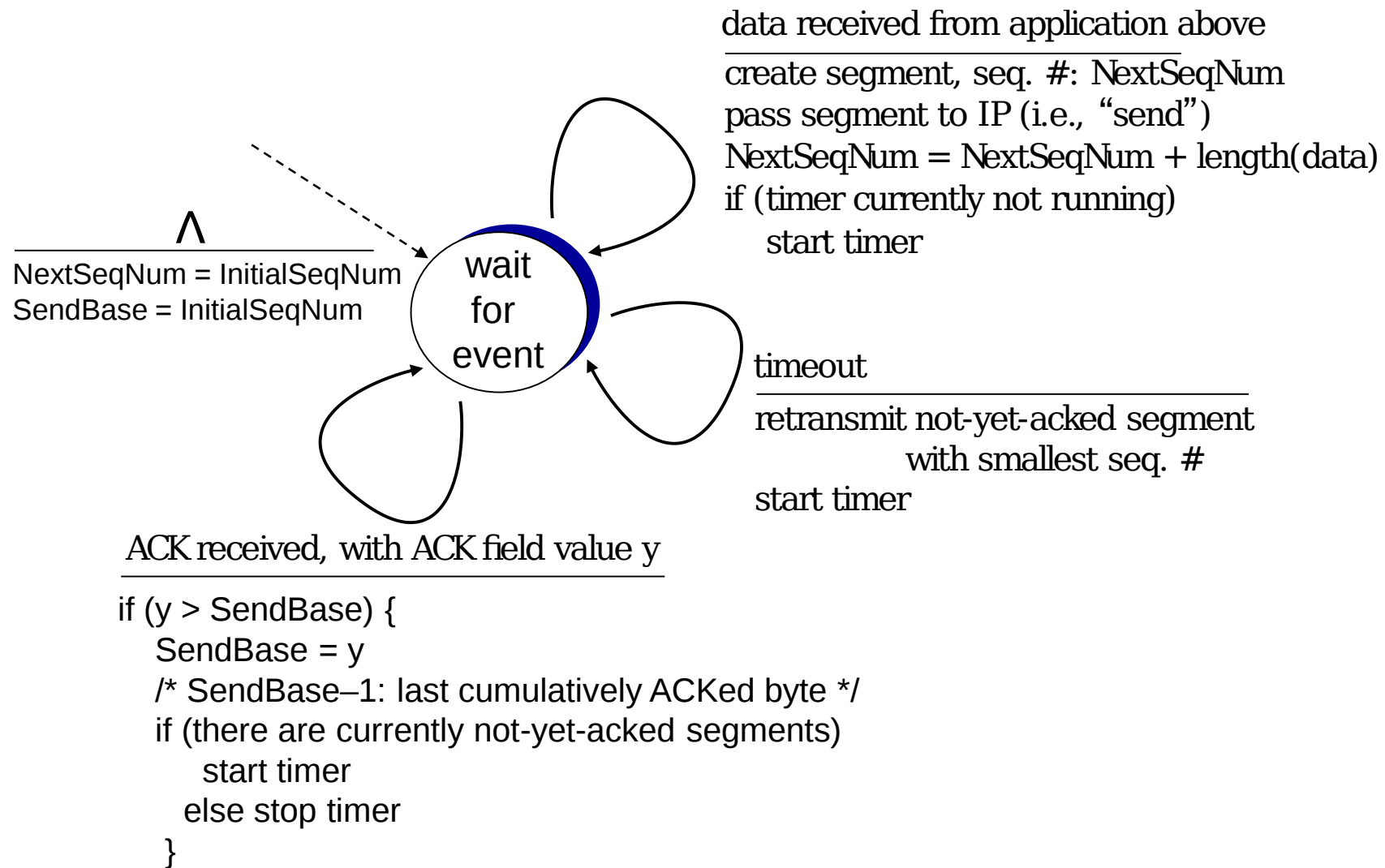
- **Estouro de temporizador:**

- Retransmita segmento que causou o estouro.
- Reinicie o temporizador.

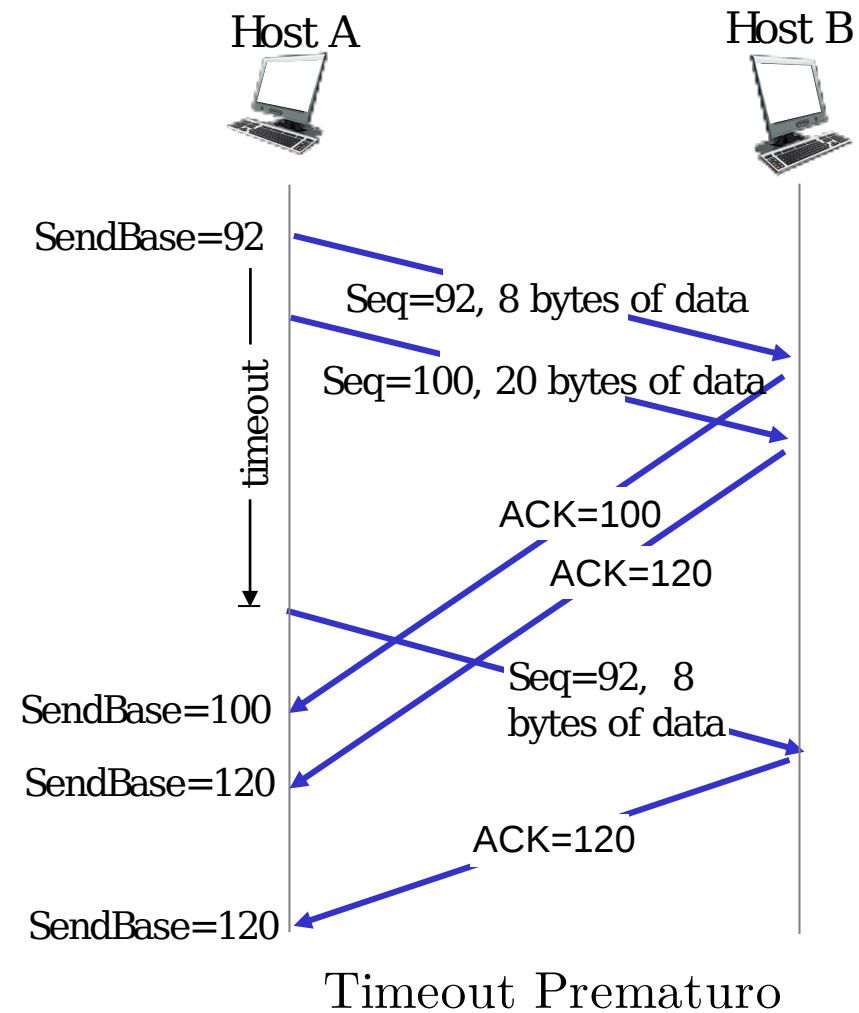
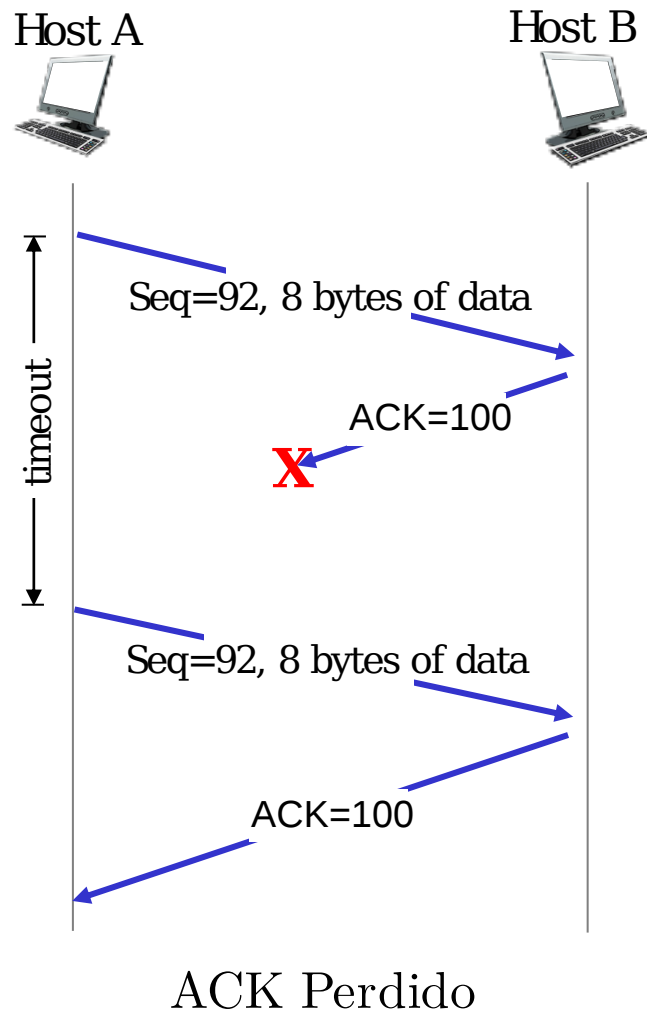
- **Recepção de ACK:**

- Se ACK reconhece segmentos ainda pendentes:
 - Atualize os números de sequência já reconhecidos.
 - **Reinicie o temporizador se ainda há segmentos não reconhecidos.**

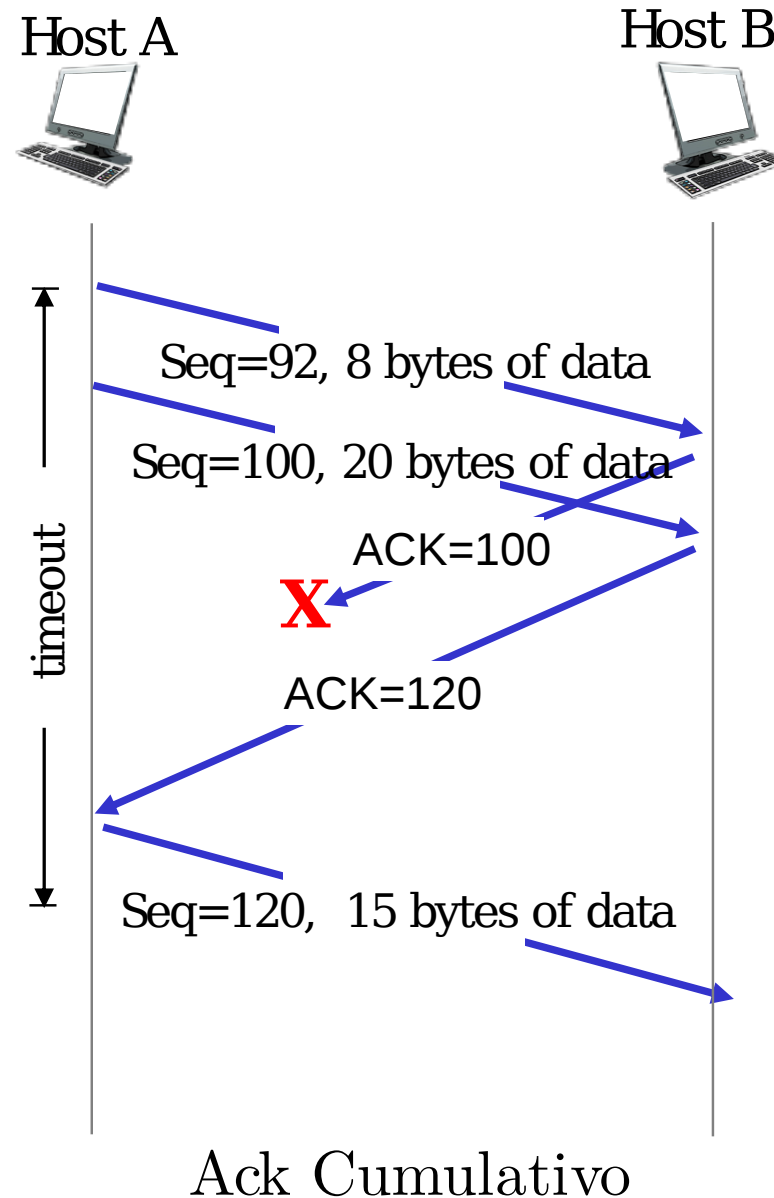
Transmissor TCP (Simplificado)



TCP: Cenários de Retransmissão (I)



TCP: Cenários de Retransmissão (II)



Geração de ACKs pelo TCP [RFC 1122, RFC 2581]

Evento no Receptor	Ação Realizada
Chegada de segmento em ordem com # de seq. esperado. Já foram enviados ACKs para todos os números de sequência menores que o recebido.	ACK atrasado. Esperar até 500 ms pelo próximo segmento. Se nenhum novo segmento chegar, enviar o ACK.
Chegada de segmento em ordem com # de seq. Segmento anterior teve seu ACK atrasado (envio pendente).	Imediatamente enviar ACK cumulativo reconhecendo ambos os segmentos.
Chegada de segmento fora de ordem, com # de seq. maior que o esperado. Lacuna detectada.	Imediatamente enviar um ACK duplicado , indicando o # de seq. do próximo byte esperado.
Chegada de segmento que parcialmente ou completamente preenche uma lacuna.	Imediatamente enviar um ACK, desde que o segmento comece no início da lacuna.

Fast Retransmit

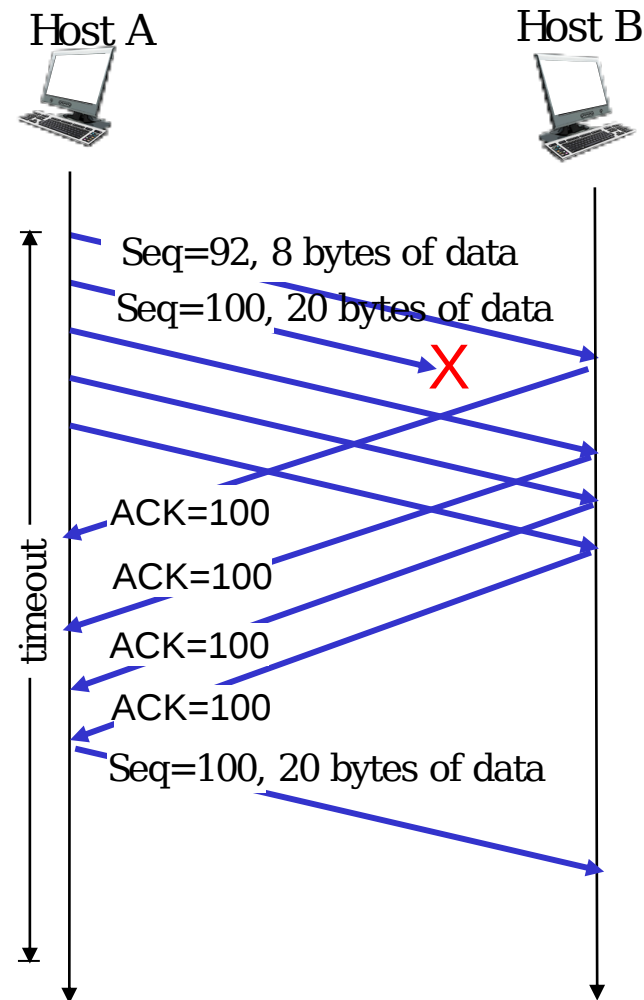
- Tempo de expiração do temporizador normalmente é relativamente longo.
 - Atraso longo antes do reenvio do pacote perdido.
- Detecção de segmentos perdidos através de ACKs duplicados.
 - Transmissor geralmente envia vários segmentos em sequência.
 - Se um segmento é perdido, provavelmente haverá múltiplos ACKs duplicados.

TCP Fast Retransmit

Se transmissor recebe 3 ACKs repetidos para um mesmo número de sequência (“ACK duplicado triplo”), reenvie segmento com menor # de seq.

- Provavelmente, segmento foi perdido. Então, não esperamos pelo *timeout*.

Fast Retransmit: Exemplo



Fast Retransmit é disparado
depois da recepção de 3 ACKs
duplicados para o # de seq. 100

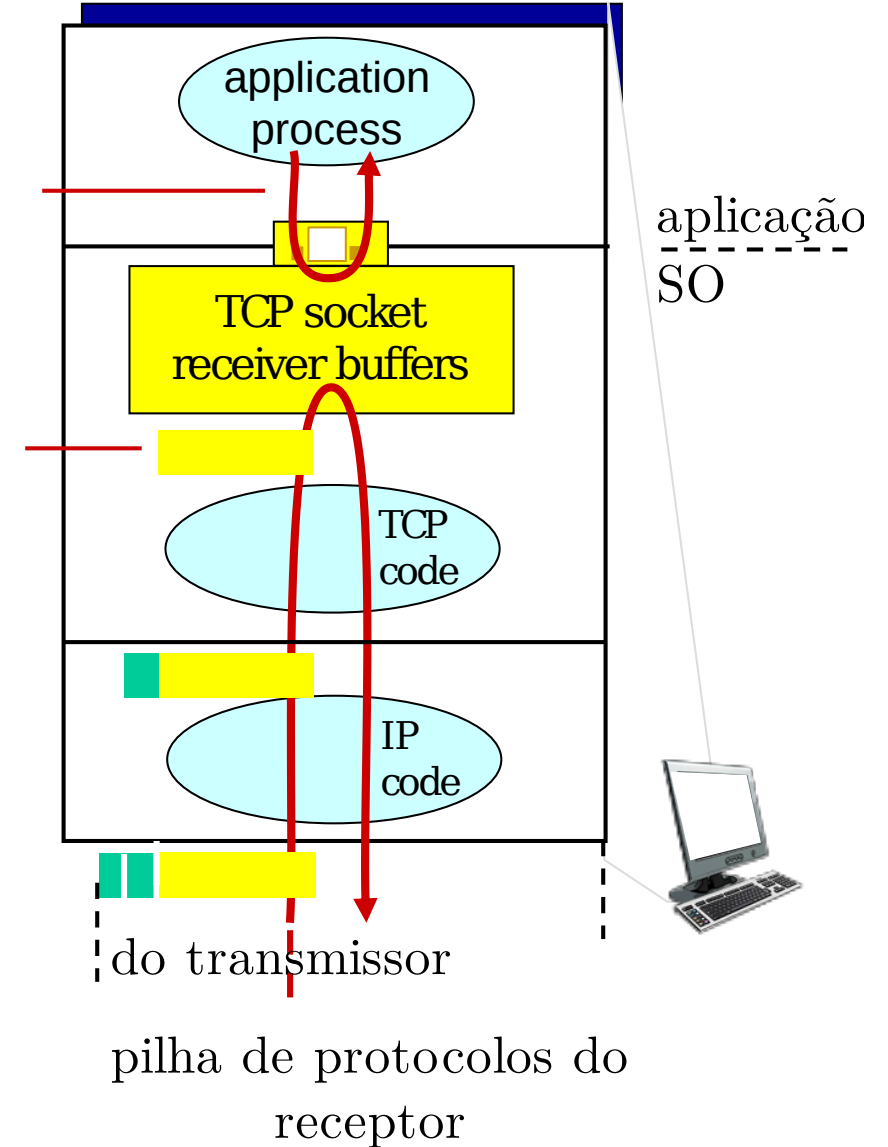
TCP: Controle de Fluxo

TCP: Controle de Fluxo (I)

- Aplicação pode ler dados do *buffer* do socket TCP mais devagar que estes chegam ao *host* de destino.

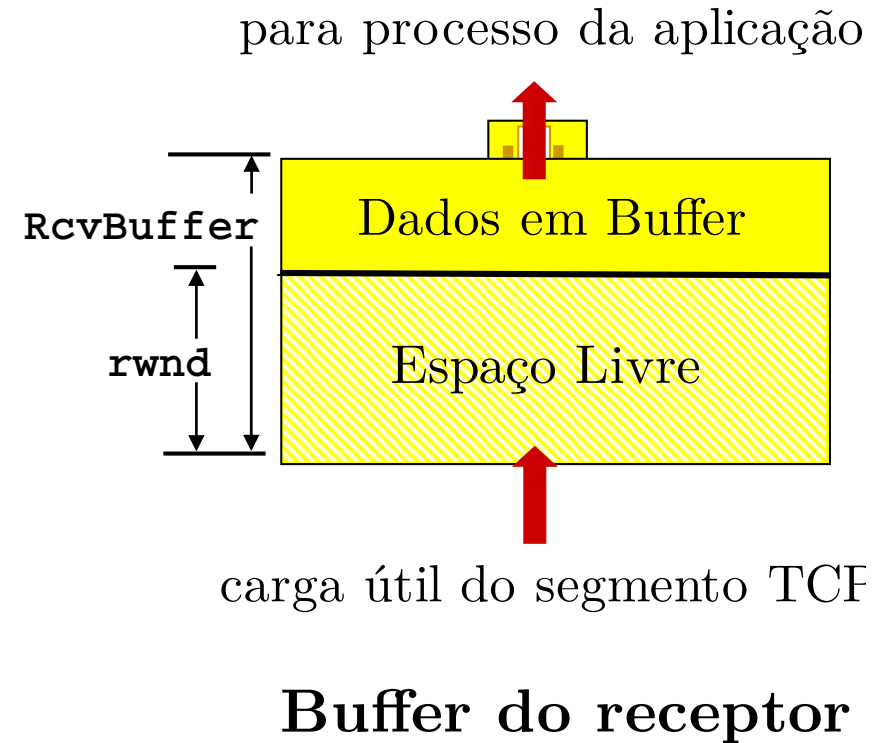
Controle de Fluxo

Receptor controla transmissor, de forma que transmissor não sobrecarregue o *buffer* do receptor transmitindo muitos dados, rápido demais.



TCP: Controle de Fluxo (II)

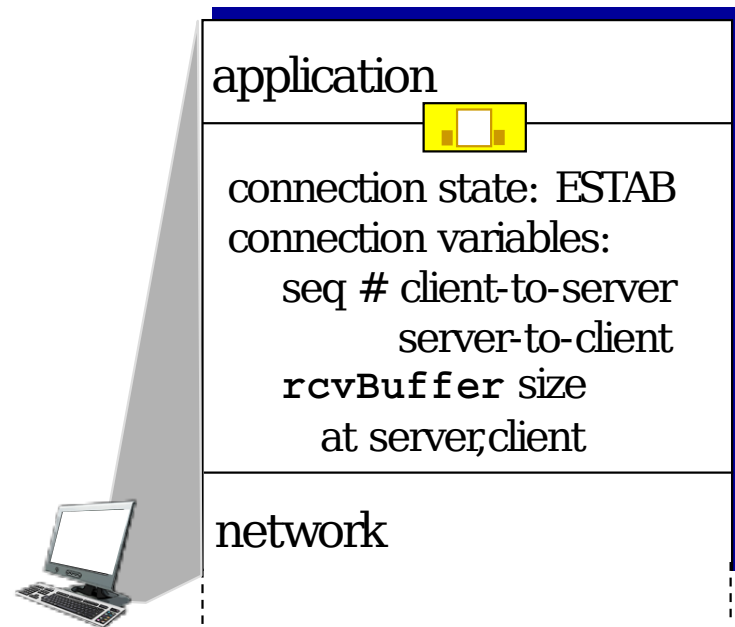
- Receptor “anuncia” espaço livre no seu *buffer*, incluindo este valor no campo *rwnd* do cabeçalho TCP dos segmentos enviados do receptor para o transmissor.
 - Tamanho do *buffer* de recepção do TCP pode ser alterado via API de sockets (valor padrão típico é 4096 bytes).
 - Muitos sistemas operacionais ajustam o valor automaticamente.
- Transmissor limita quantidade de segmentos pendentes (“em trânsito”) ao *rwnd* do receptor.
- Garante que não haverá *overflow* no *buffer* do receptor.



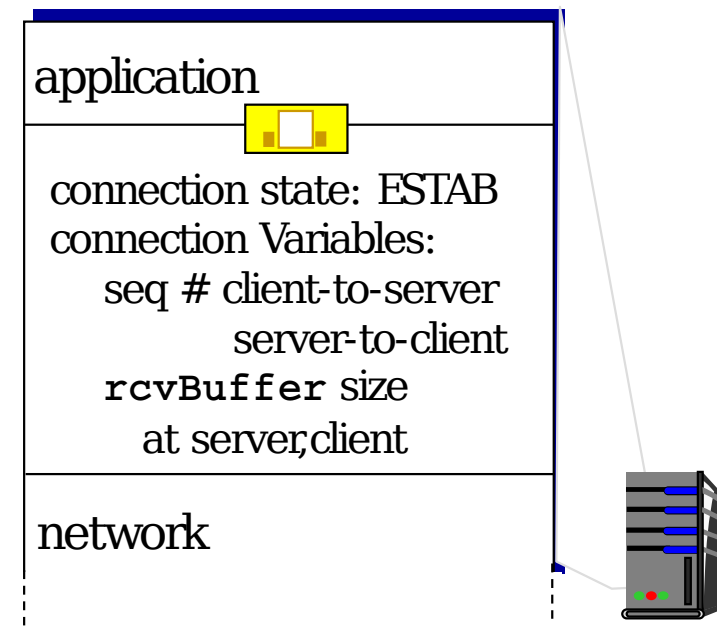
TCP: Gerenciamento da Conexão

Gerenciamento da Conexão

- Antes de trocar dados, transmissor e receptor fazem um “handshake”.
 - Concordam em estabelecer conexão.
 - Acordam parâmetros da conexão.



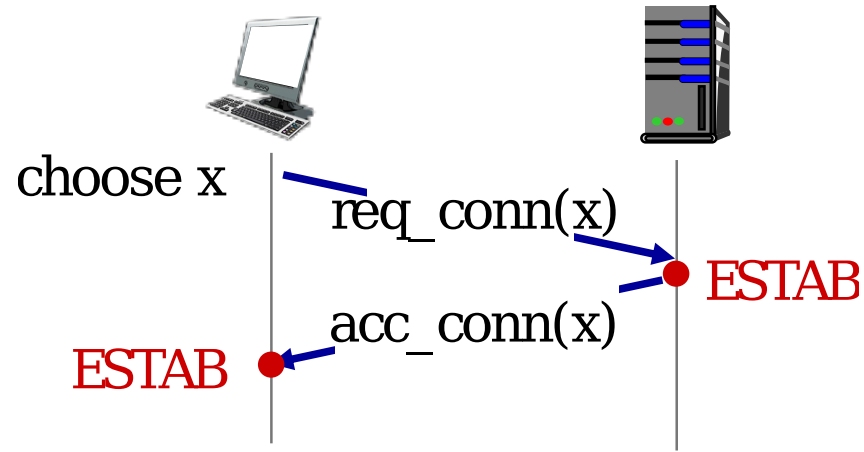
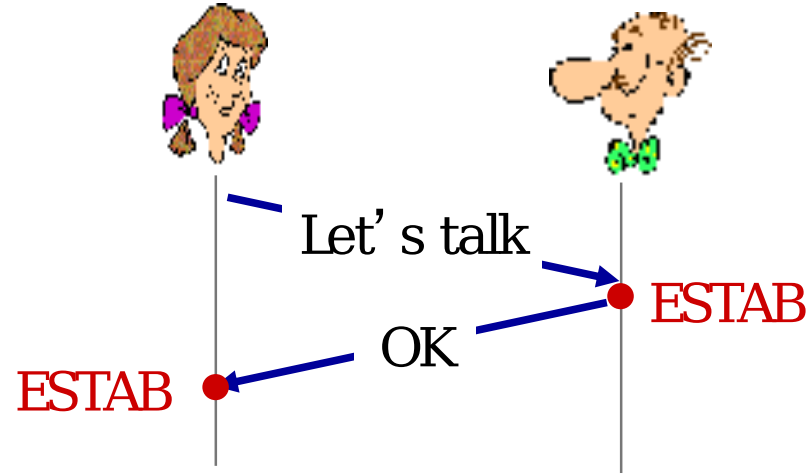
```
Socket clientSocket =  
    newSocket ("hostname", "port  
    number");
```



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

Concordando em Estabelecer uma Conexão (I)

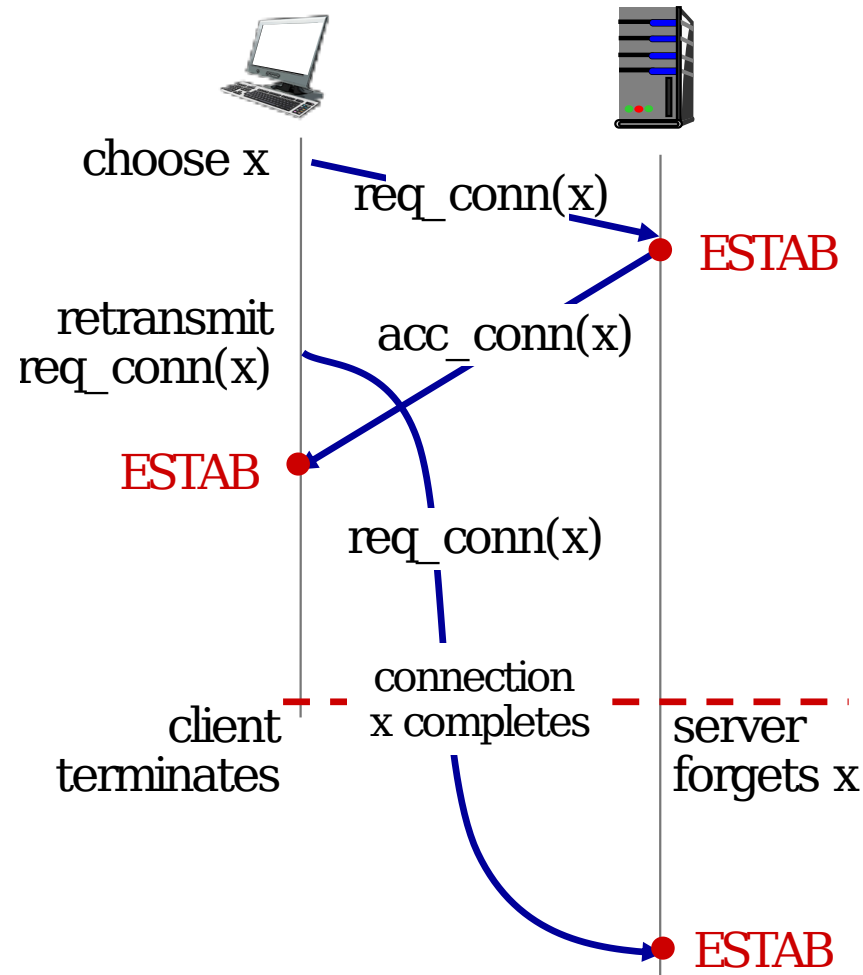
Handshake de duas fases:



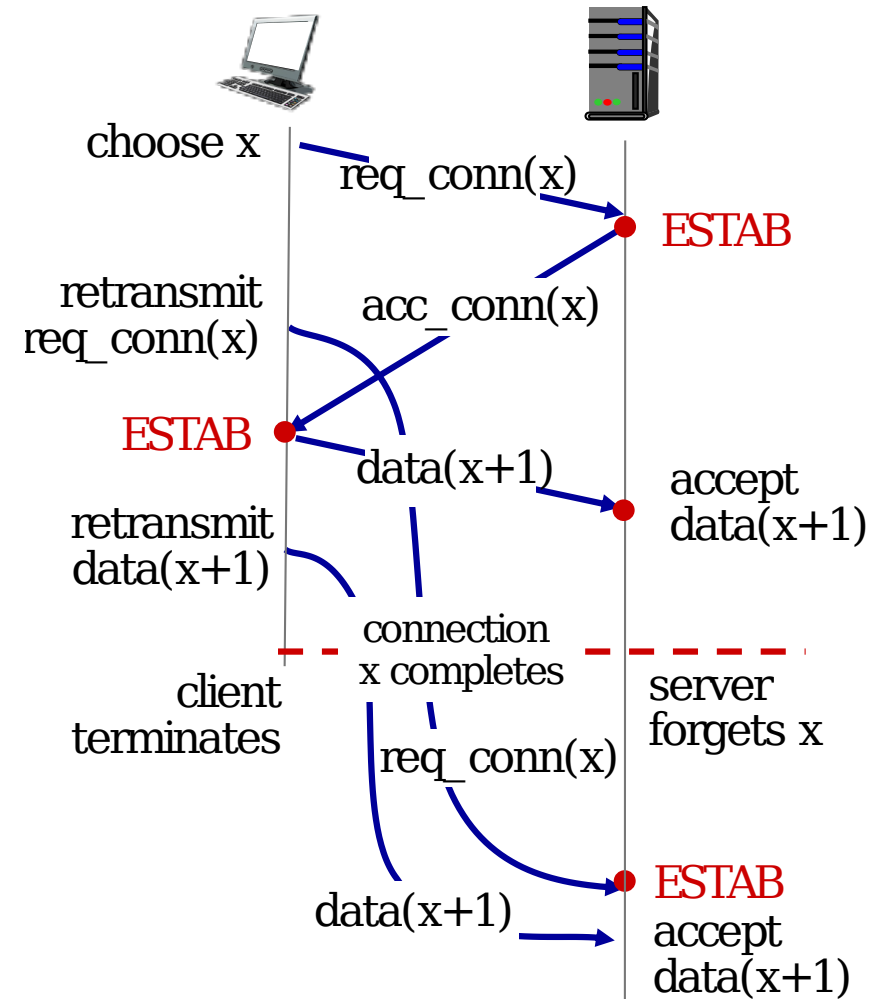
- **Pergunta:** um *handshake* de duas fases sempre irá funcionar em uma rede?
 - Atrasos variáveis.
 - Mensagens retransmitidas.
 - e.g., `req_conn(x)`, devido a perda de mensagem.
 - Reordenação de mensagem.
 - É impossível “ver” o outro lado.

Concordando em Estabelecer uma Conexão (II)

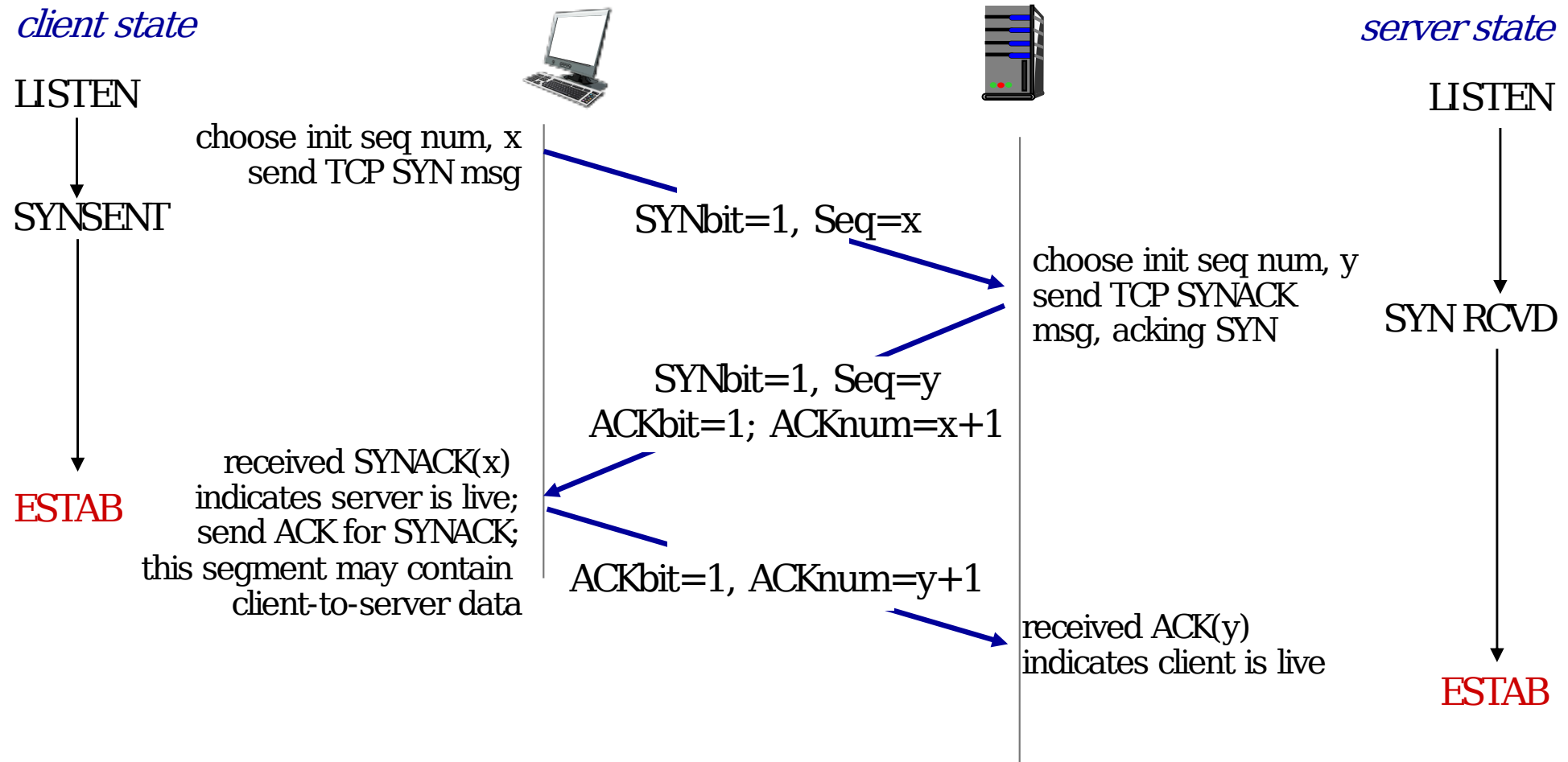
- Cenários de falha do 2-way handshake:



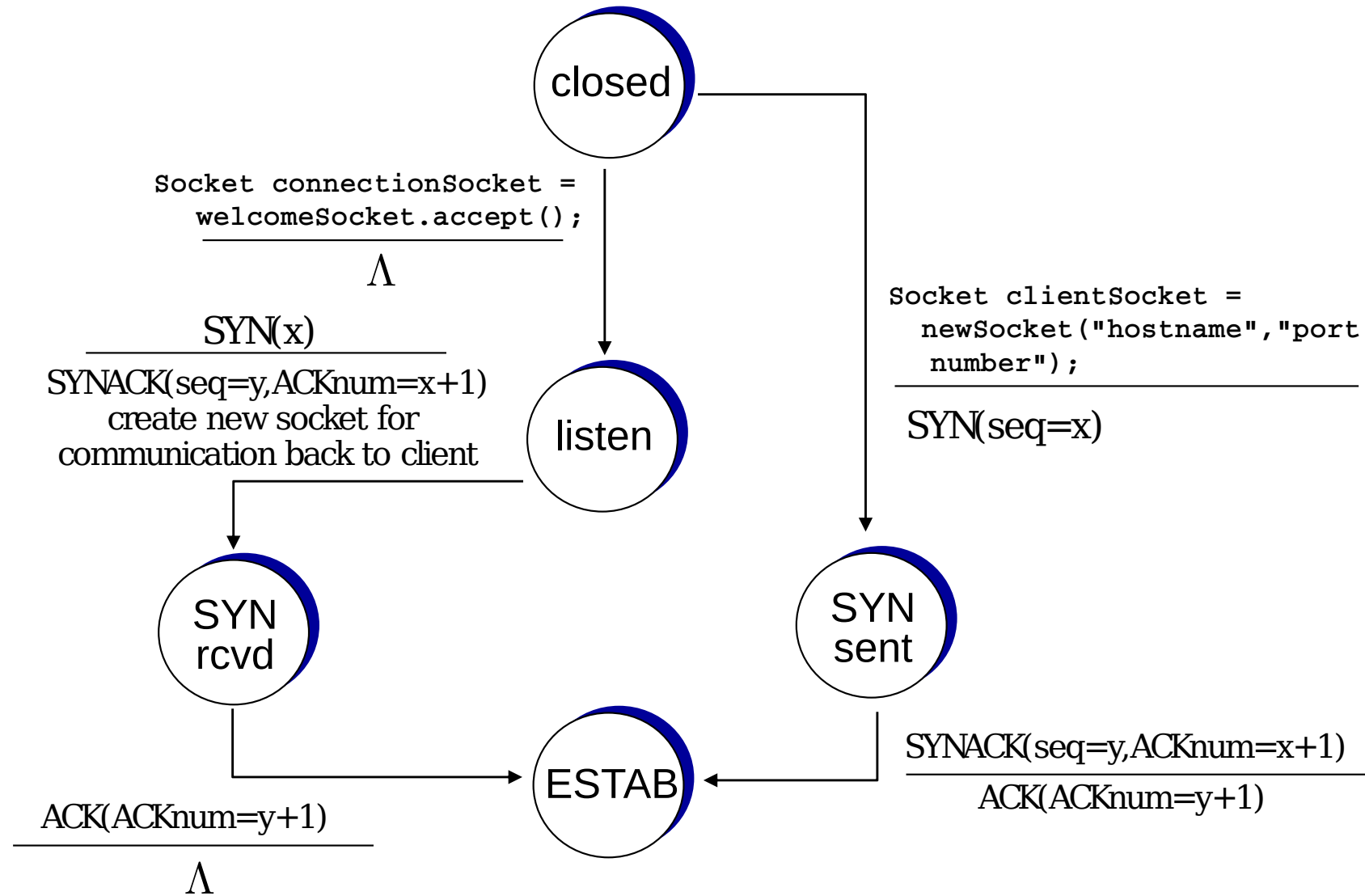
meia conexão aberta!
(sem cliente!)



TCP: 3-way handshake



TCP: Máquina de Estados do 3-way handshake



TCP: Fechando uma Conexão (I)

- Cliente, servidor fecham cada um seu lado da conexão.
 - Envia segmento TCP com bit FIN = 1.
- Sempre respondem segmento com FIN = 1 com um ACK.
 - Ao receber um FIN, ACK pode ser combinado também com um FIN.
- Protocolo também lida com troca simultânea de FINs.

TCP: Fechando uma Conexão (II)

