

CAP 5. CAMADA DE TRANSPORTE

AULA 1: SERVIÇOS DA CAMADA

INE5422 REDES DE COMPUTADORES II

PROF. ROBERTO WILLRICH (INE/UFSC)

ROBERTO.WILLRICH@UFSC.BR

[HTTPS://MOODLE.UFSC.BR](https://moodle.ufsc.br)

Capítulo 3

Camada de Transporte

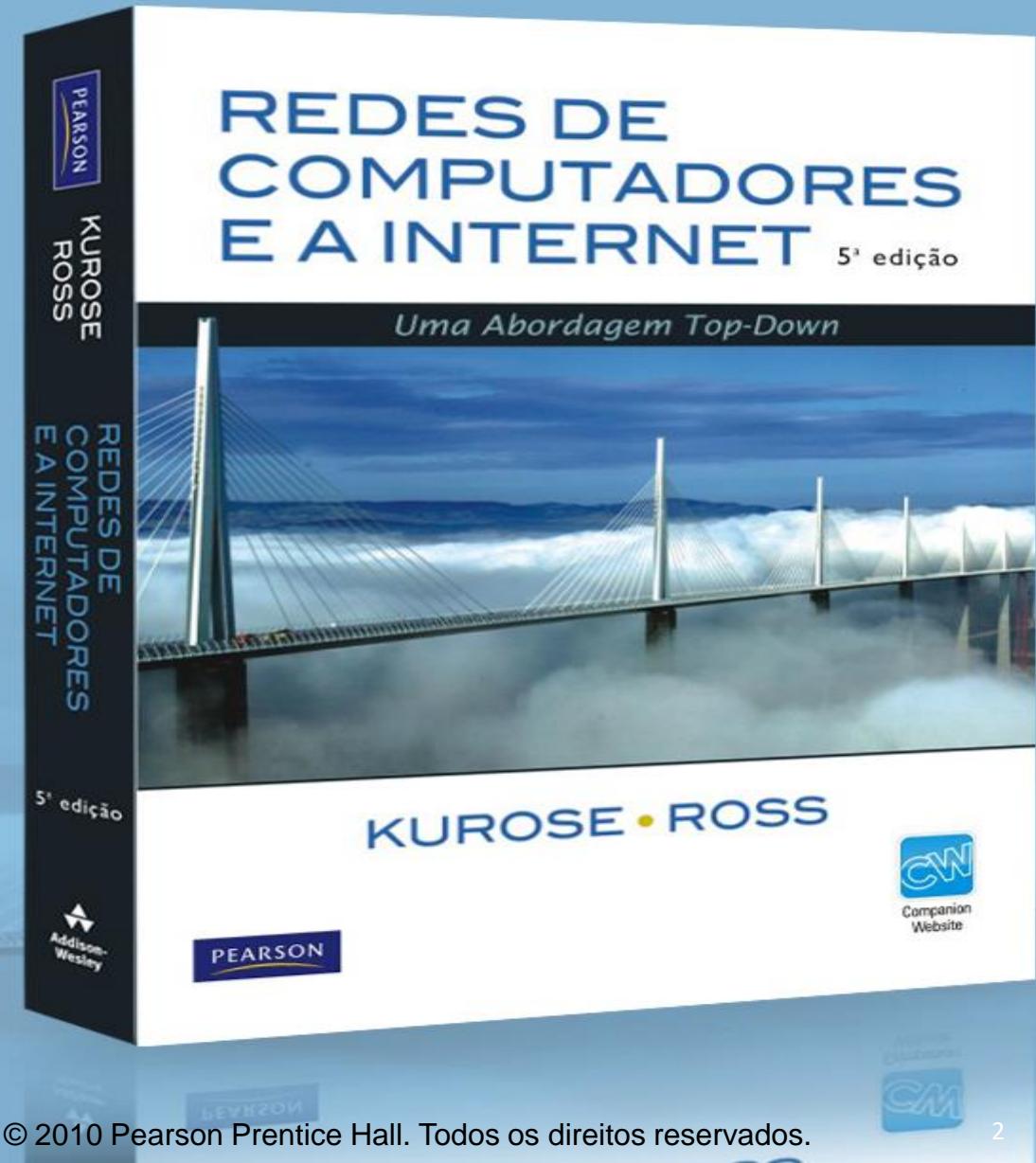
Nota sobre o uso destes slides ppt:

Estamos disponibilizando estes slides gratuitamente a todos (professores, alunos, leitores). Eles estão em formato do PowerPoint para que você possa incluir, modificar e excluir slides (incluindo este) e o conteúdo do slide, de acordo com suas necessidades. Eles obviamente representam *muito* trabalho da nossa parte. Em retorno pelo uso, pedimos apenas o seguinte:

- Se você usar estes slides (por exemplo, em sala de aula) sem muita alteração, que mencione sua fonte (afinal, gostamos que as pessoas usem nosso livro!).
- Se você postar quaisquer slides sem muita alteração em um site Web, que informe que eles foram adaptados dos (ou talvez idênticos aos) nossos slides, e inclua nossa nota de direito autoral desse material.

Obrigado e divirta-se! JFK/KWR

Todo o material copyright 1996-2009
J. F Kurose e K. W. Ross, Todos os direitos reservados.



CAPÍTULO 5: CAMADA DE TRANSPORTE

Metas do capítulo:

- Entender os princípios por trás dos serviços da camada de transporte:
 - multiplexação/demultiplexação
 - transferência confiável de dados
 - controle de fluxo
 - controle de congestionamento

Aprender os protocolos de transporte da Internet:

- Transporte sem conexão: UDP
- Transporte orientado a conexão: TCP
 - transferência confiável
 - controle de fluxo e de congestionamento
 - gerenciamento de conexões

SERVIÇOS DA CAMADA DE TRANSPORTE

Finalidade

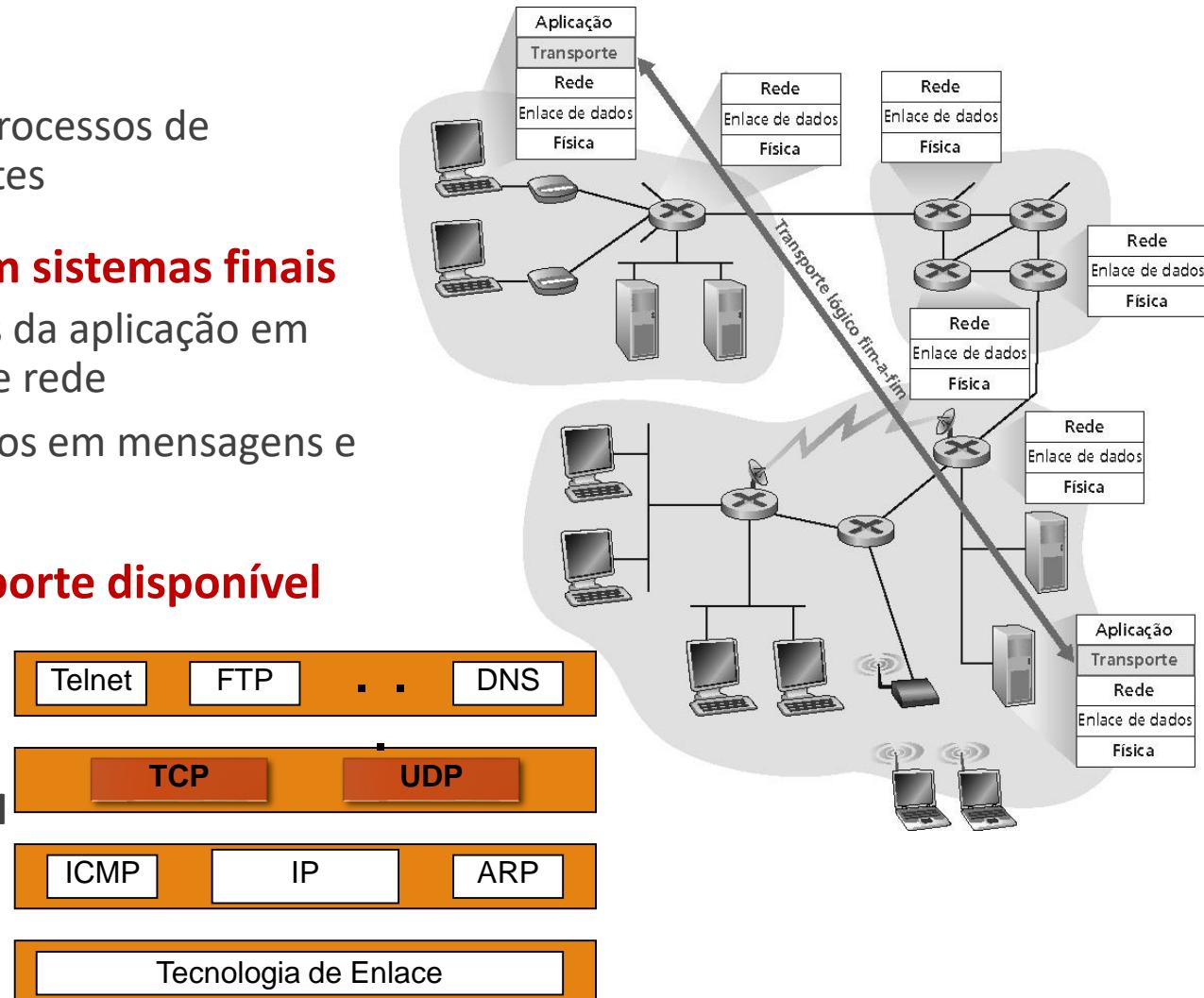
- Fornece comunicação lógica entre processos de aplicação rodando em hosts diferentes

Protocolos de transporte rodam em sistemas finais

- **Lado emissor:** quebra as mensagens da aplicação em segmentos e envia para a camada de rede
 - **Lado receptor:** remonta os segmentos em mensagens e passa para a camada de aplicação

Há mais de um protocolo de transporte disponível para as aplicações

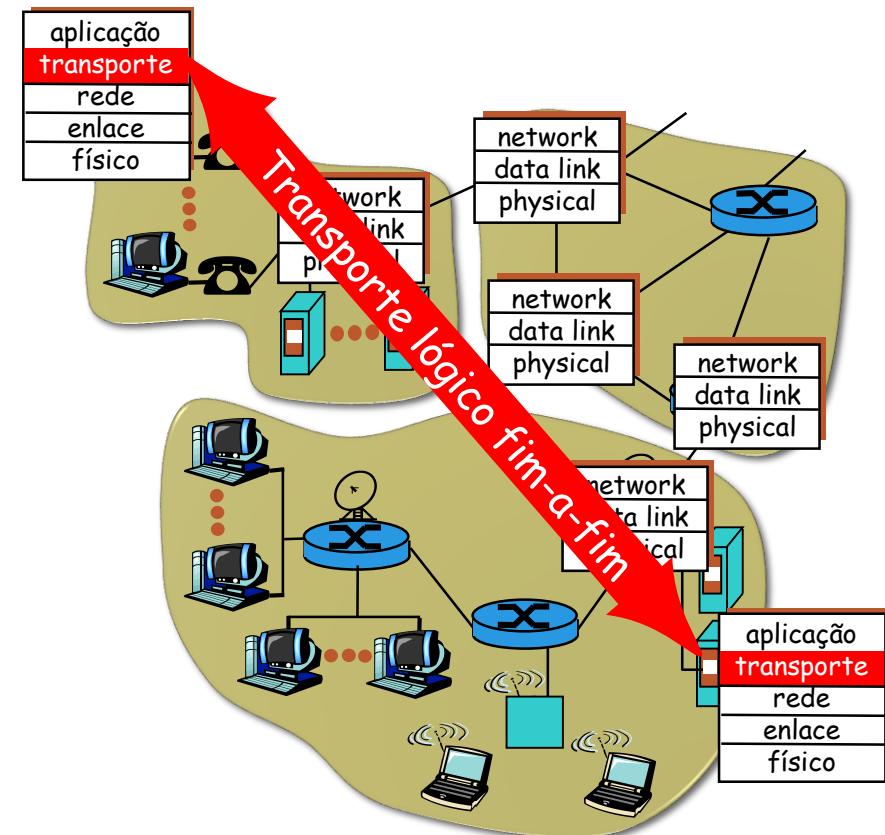
- **UDP – User Datagram Protocol**
 - Serviço sem conexão não confiável
 - **TCP – Transmission Control Protocol**
 - Serviço confiável e orientado a conexão



PROTOCOLOS DA CAMADA DE TRANSPORTE

Serviços de transporte da Internet

- **TCP:** Confiável, liberação em ordem unicast
 - Configuração de conexão
 - Sequenciamento de bytes
 - Controle de fluxo e de congestionamento
- **UDP:** Não confiável, liberação fora de ordem unicast ou multicast
- Serviços não disponíveis no TCP e UDP:
 - Tempo-real
 - Garantias de banda
 - Multicast confiável



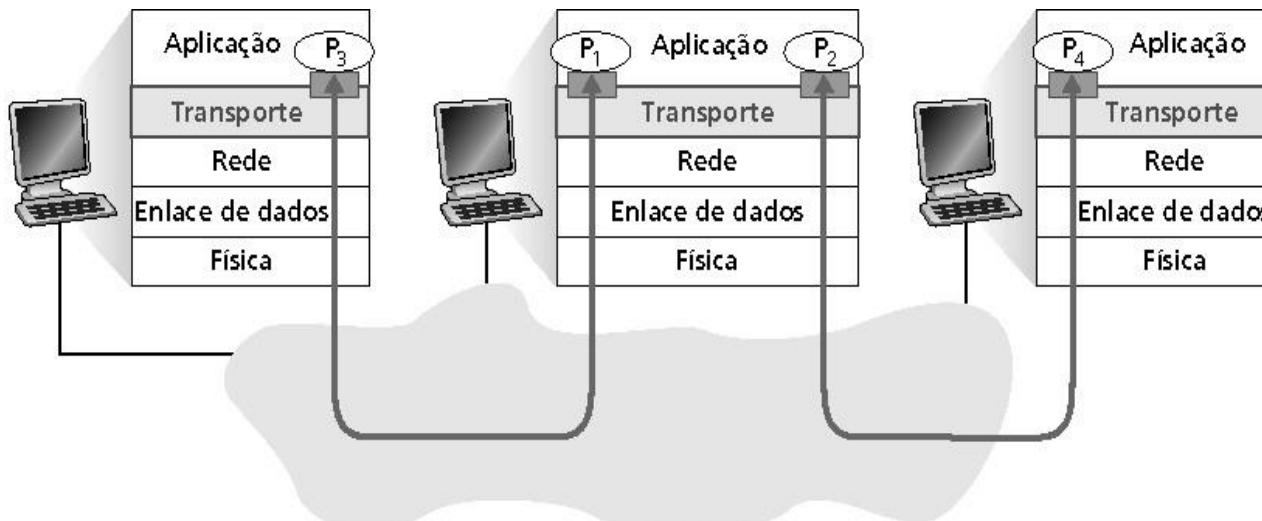
MULTIPLEXAÇÃO/DEMULITPLEXAÇÃO

Demultiplexação no hospedeiro receptor:

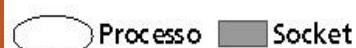
- entrega os segmentos recebidos ao socket correto

Multiplexação no hospedeiro emissor:

- coleta dados de múltiplos sockets, envelopa os dados com cabeçalho (usado depois para demultiplexação)



Legenda:



PORTAS DE PROTOCOLO

Último fonte/destino de/para uma mensagem é uma porta de protocolo

- Um processo envia via/ouve uma porta de protocolo (identificado por um inteiro)
- Muitos sistemas operacionais fornecem acesso síncrono as portas
 - Um processo pode se bloquear aguardando chegada de mensagens na porta
- Em geral, portas são buferizadas
 - Dados chegando antes da operação de leitura de um processo é colocada em uma fila (finita)

Para se comunicar com uma porta, um emissor necessita conhecer o Endereço IP e a Porta do processo receptor

- Combinação de endereço IP e porta é chamado de socket

NÚMERO DE PORTAS EM TRÊS GRUPOS

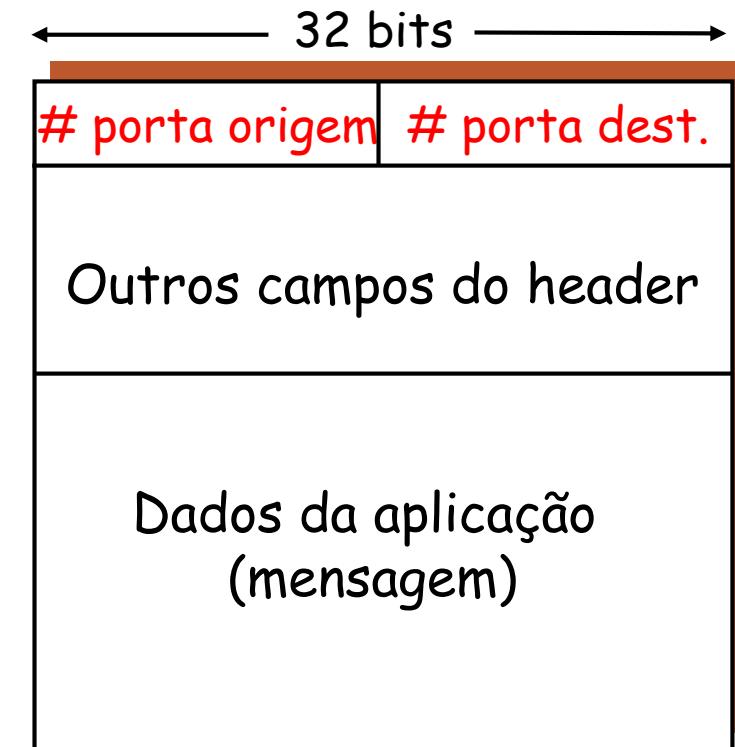
Faixa	Propósito
1 .. 1023	Portas bem conhecidas são atribuídas pela Internet Assigned Numbers Authority (IANA)
1024 .. 49151	Portas registradas
49152 .. 65535	Portas dinâmicas

- Servidores são normalmente conhecidos por portas bem conhecidas (por exemplo, 80 para HTTP)
- Portas dinâmicas podem ser usados por qualquer processos (normalmente usados por processos clientes)

MULTIPLEXAÇÃO/DEMULITPLEXAÇÃO

Baseado no número da porta do emissor e receptor, e endereços IP

- Números de portas origem e destino em cada segmento (16 bits: 0..65535)
- lembrete: número de portas bem conhecidas para aplicações específicas (0..1024)



Formato do segmento TCP/UDP

DEMULTIPLEXAÇÃO NÃO ORIENTADA À CONEXÃO

Cria sockets com números de porta:

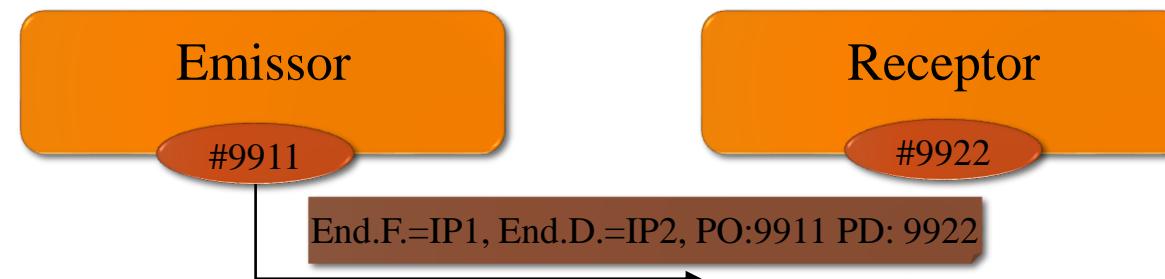
- Emissor: `mySocket1 = socket.socket(socket.AF_INET, SOCK_DGRAM)`
`mySocket1.bind((IP1, 9911))`
- Receptor: `mySocket2 = socket.socket(socket.AF_INET, SOCK_DGRAM)`
`mySocket2.bind((IP2, 9922))`

Socket UDP identificado pela porta que ele utiliza. No envio:

- `dest = (HOST, PORT) # Destino da mensagem`
- `mySocket1.sendto(message, dest)`

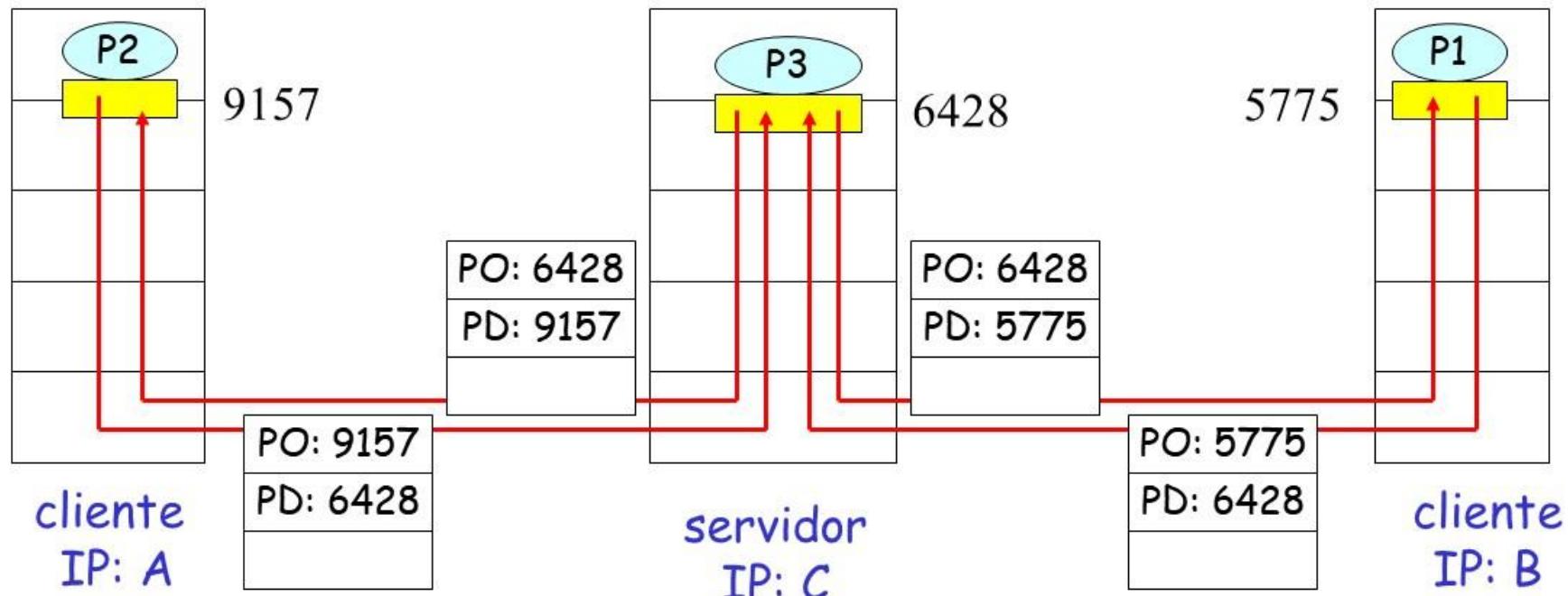
Na recepção o hospedeiro:

- Verifica o número da porta de destino no segmento
- Direciona o segmento UDP para o socket com este número de porta



DEMULTIPLEXAÇÃO NÃO ORIENTADA À CONEXÃO

```
mySocket3 = socket.socket(socket.AF_INET, SOCK_DGRAM)  
mySocket3.bind((IPc, 6428))
```



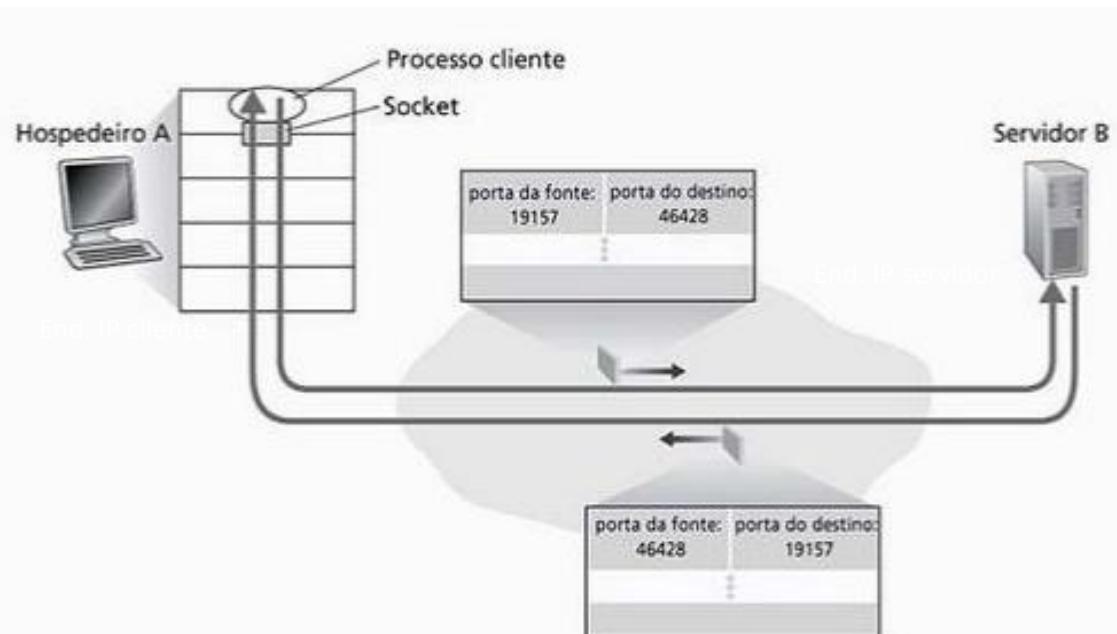
DEMUX ORIENTADA À CONEXÃO

No servidor (servidor.com.br) é instanciado um `ServerSocket`:

- `welcomeSocket=socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
- `welcomeSocket.bind(("servidor.com.br", 46428))`
- `welcomeSocket.listen()`
- `conn, addr = s.accept()`

No cliente se comunicando com servidor:

- `clientSocket= socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
- `clientSocket.connect((HOST, PORT))`

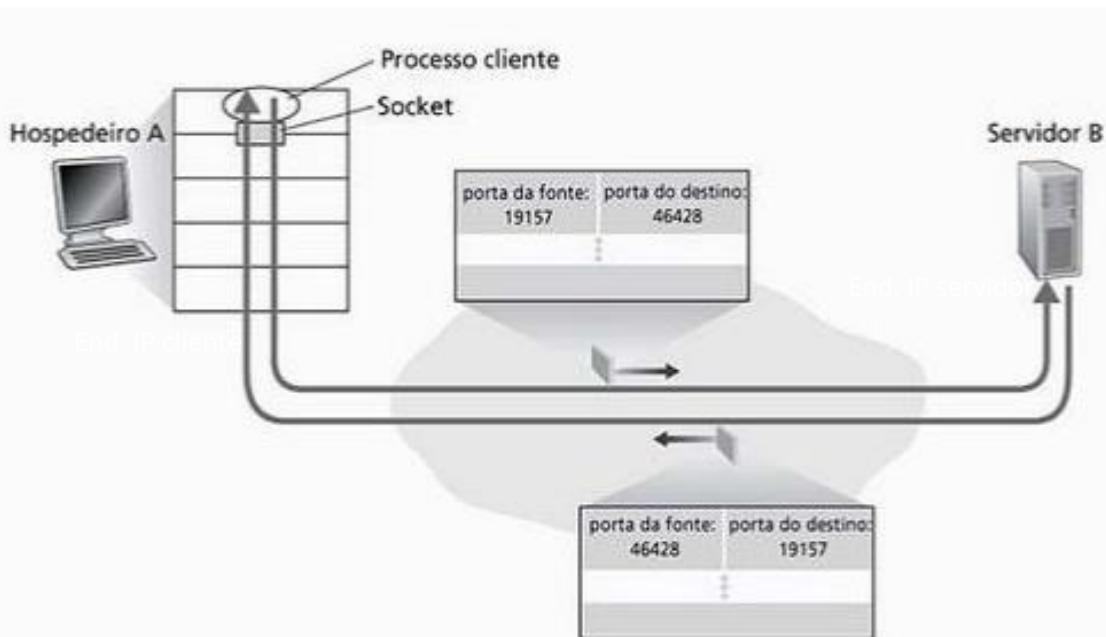


DEMUX ORIENTADA À CONEXÃO

Socket de conexão

- Instaciado no estabelecimento da conexão, é identificado por 4 valores: (End. IP de origem, End. porta de origem, Endereço IP de destino, End. porta de destino)
- Hospedeiro receptor usa os quatro valores para direcionar o segmento ao socket apropriado

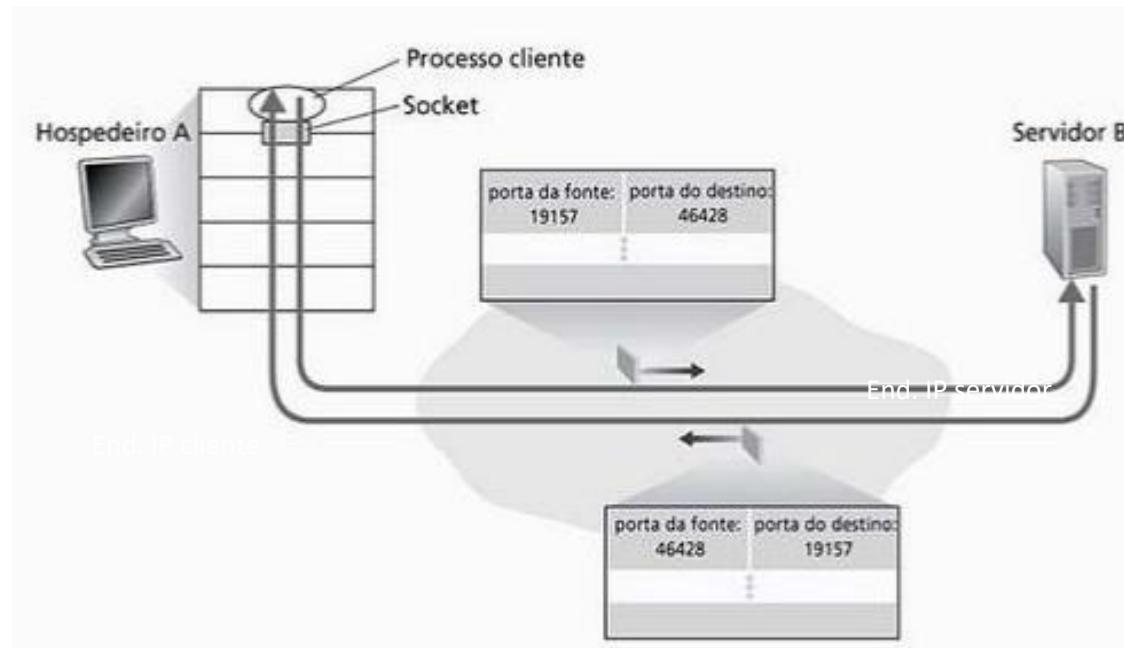
```
TCP      150.162.59.211:64347      162.125.34.129:https      ESTABLISHED      3020
```



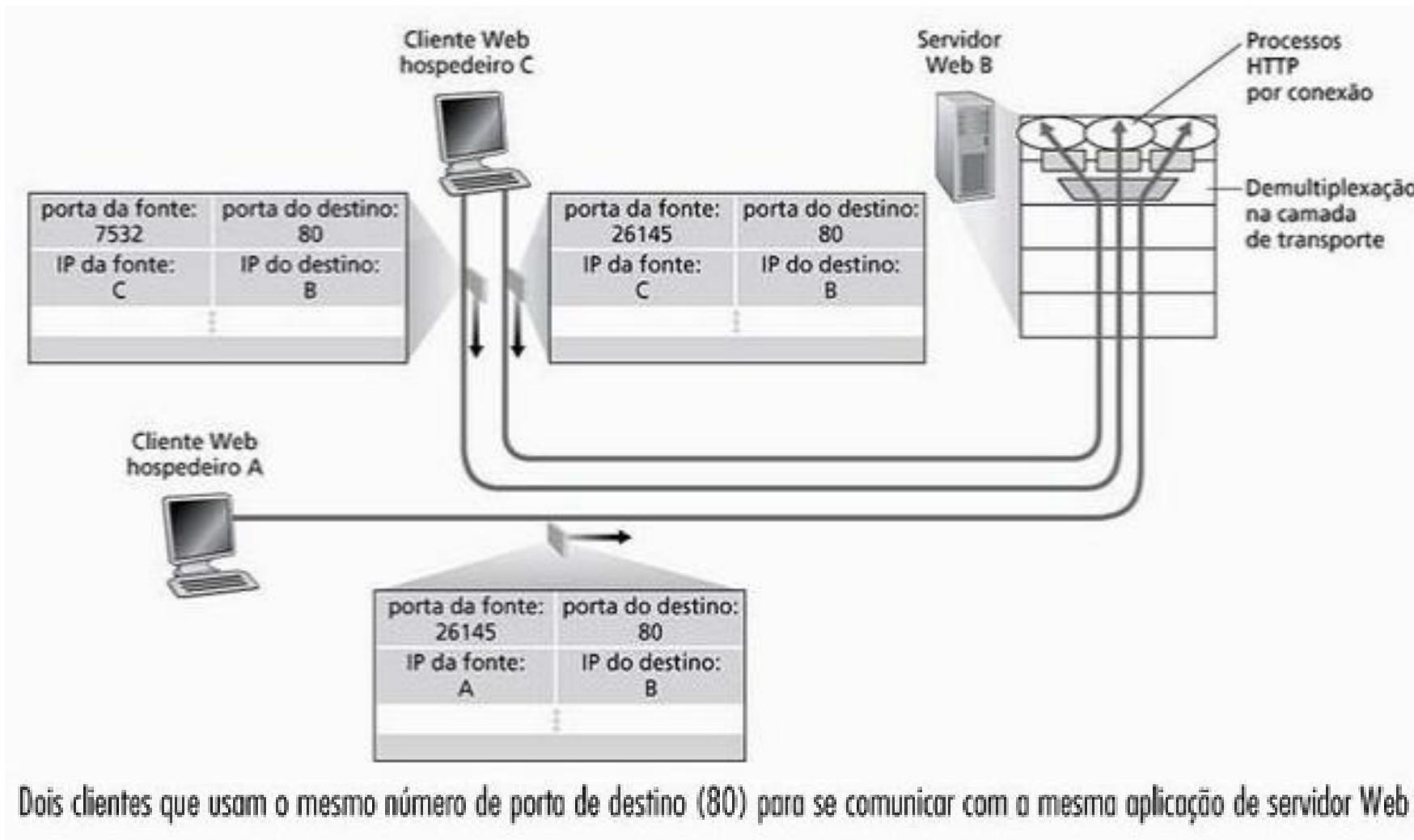
DEMUX ORIENTADA À CONEXÃO

Hospedeiro servidor pode suportar vários sockets TCP simultâneos:

- Cada socket de conexão é identificado pelos seus próprios 4 valores: Endereço Origem, Porta Origem, Endereço Destino, Porta Destino
- Servidores Web possuem sockets diferentes para cada cliente conectado



DEMUX ORIENTADA À CONEXÃO



CAP 5. CAMADA DE TRANSPORTE

AULA 2: PROTOCOLO UDP

INE5422 REDES DE COMPUTADORES II

PROF. ROBERTO WILLRICH (INE/UFSC)

ROBERTO.WILLRICH@UFSC.BR

[HTTPS://MOODLE.UFSC.BR](https://MOODLE.UFSC.BR)

UDP: USER DATAGRAM PROTOCOL [RFC 768]

Protocolo de transporte mínimo, “sem frescura”

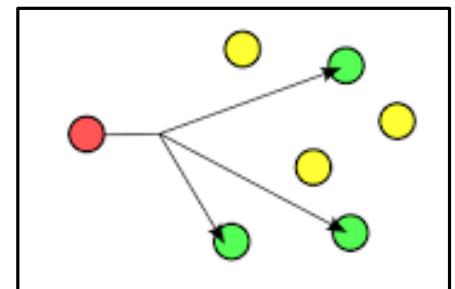
- Sem conexão:
 - não há “setup” UDP entre remetente e receptor
 - tratamento independente de cada datagrama UDP pela rede
- Oferecendo basicamente multiplexação/demultiplexação

Serviço “melhor esforço”

- Não garante taxa, atraso e taxa de perdas de pacotes
- Pacotes podem ser entregues à aplicação fora de ordem do envio

Permite transmissão em multicast

- Pacote pode ser enviado para um grupo de destinos



PROTOCOLO UDP

Formato do Datagrama UDP



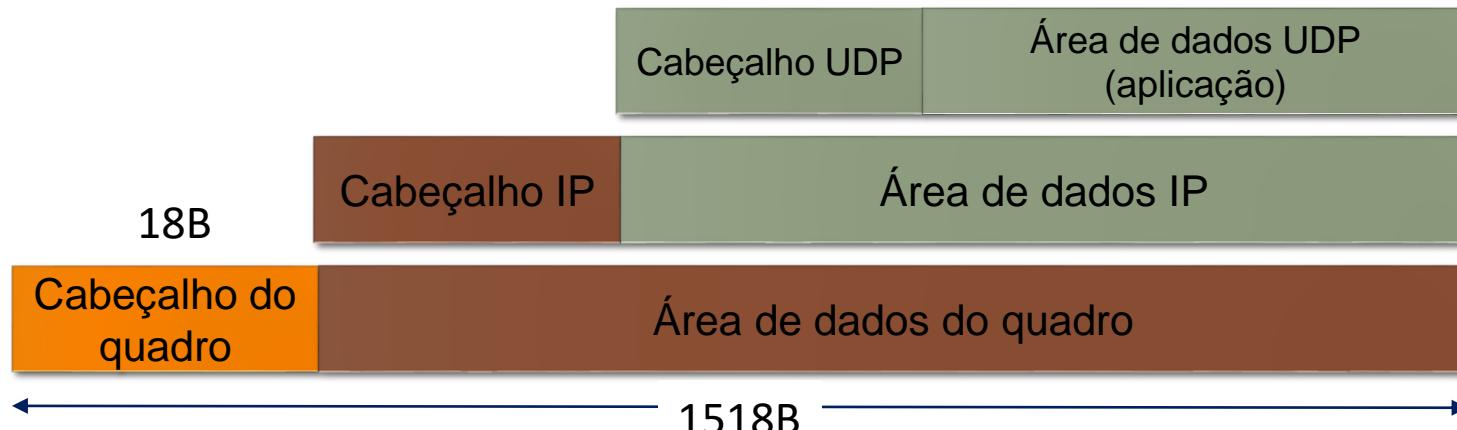
Onde:

- **Porta Origem e Porta Destino** identificam o processo de aplicação que está enviando dados e o processo de aplicação que irá receber os dados.
- **Tamanho** é representado o tamanho total do frame UDP
- **Checksum** é calculado usando o header UDP e também a área de dados, e destina-se a verificação de erros de transmissão.

TAMANHO MÁXIMO DO DATAGRAMA UDP

Teoricamente o tamanho máximo é de 64KB

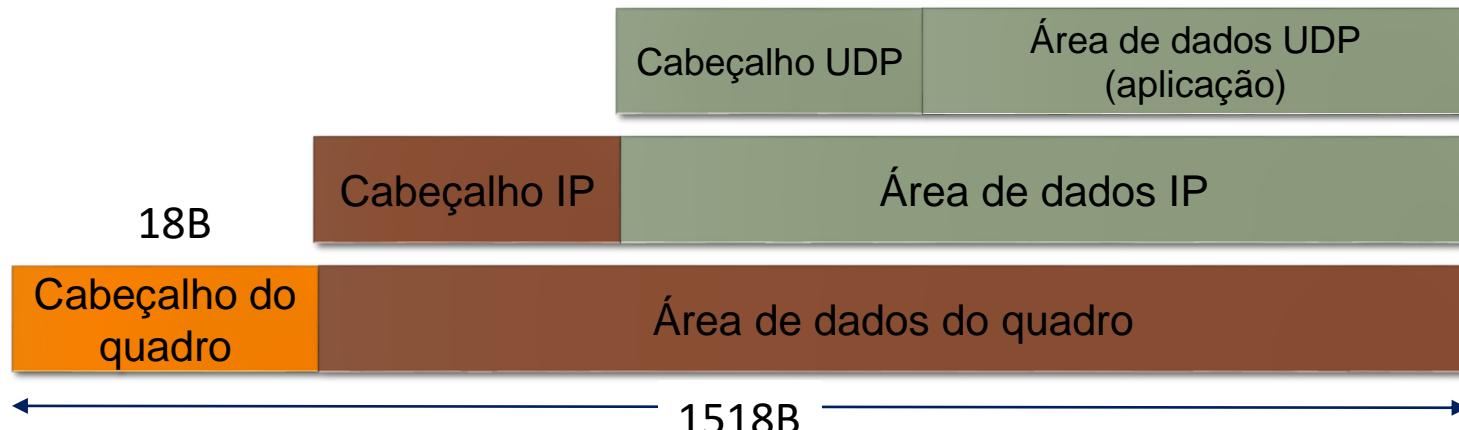
- Porque o campo tamanho é de 16 bits (2^{16} valores)
- Deve-se considerar que no IP o tamanho também é de 16 bits, que deve conter:
 - Cabeçalho IP (20 bytes)
 - Cabeçalho UDP (8 bytes)
- Assim, o tamanho máximo é de 65507 bytes
- Mas geralmente evita-se fragmentação do pacote UDP na fonte, assim o tamanho do quadro (camada 2) limita o tamanho do pacote UDP



TAMANHO MÁXIMO DO DATAGRAMA UDP

Ethernet 802.3

- Tamanho do payload do Ethernet é 1500 bytes
- Cabeçalho IP = 20B, Cabeçalho UDP= 8B
- Tamanho máximo de dados para não ocorrer fragmentação é $1500 - 28 = 1472$ B



CHECKSUM UDP

Meta: detectar erros no segmento transmitido

Emissor:

- Trata conteúdo do segmento como uma sequência de inteiros de 16 bits
- checksum: adição do conteúdo do segmento
 - Resultado é complementado de 1
- Coloca o valor checksum no campo UDP checksum

Receptor:

- Soma toda a sequência (incluindo checksum)
- Checa se checksum calculada é igual ao valor FFFFh:
 - NÃO – erro detectado
 - SIM – nenhum erro detectado.

CHECKSUM UDP

Exemplo numérico

- Transmissor calcula:
 - $1001101001010110 + 0000101110001110 + 0000110111001100 = 1011001110110000$
 - Complemento de 1: $1011001110110000 \Rightarrow 0100110001001111$
 - Se for tudo 0000h, checksum será fffffh
- Receptor calcula:
 - $1001101001010110 + 0000101110001110 + 0000110111001100 + 0100110001001111 = 1111111111111111$
 - Se não for FFFFh, há um erro!

Checksum no UDP é opcional

- Campo de checksum = 0, não executa verificação
- Campo de checksum $\neq 0$, executa verificação

UDP CHECKSUM E PSEUDOCABEÇALHO

UDP checksum cobre um pseudocabeçalho IP

- Checar se o pacote chegou no destino correto
- Checar se o IP entregou para o protocolo correto (UDP/TCP)



PROTOCOLO UDP

Por quê existe um UDP?

- Elimina estabelecimento de conexão (o que pode causar retardo)
- É um protocolo simples, não se mantém “estado” da conexão no remetente/receptor
 - Latência menor e uso menor de memória
- Usa mais eficientemente a banda da rede
 - Cabeçalho por segmento é menor
 - Sem controle de congestionamento: permite usar a banda de maneira mais eficiente
 - Mas pode provocar taxa de perdas altas
- Muito usado para aplicações multimídia de streaming
 - Tolerantes a perda e Sensíveis a taxa
- Outras aplicações que usam UDP
 - DNS, NFS, SNMP e Protocolo de roteamento (RIP)
- Transferência confiável sobre UDP:
 - adicionar confiabilidade na camada de aplicação para recobrimento de erro específico de aplicação

Pontos Importantes

Protocolo UDP

- Ter conhecimento dos serviços oferecidos e do protocolo em si

CAP 5. CAMADA DE TRANSPORTE

AULAS 3: PRINCÍPIOS DE TRANSFERÊNCIA CONFIÁVEL DE DADOS

INE5422 REDES DE COMPUTADORES II

PROF. ROBERTO WILLRICH (INE/UFSC)

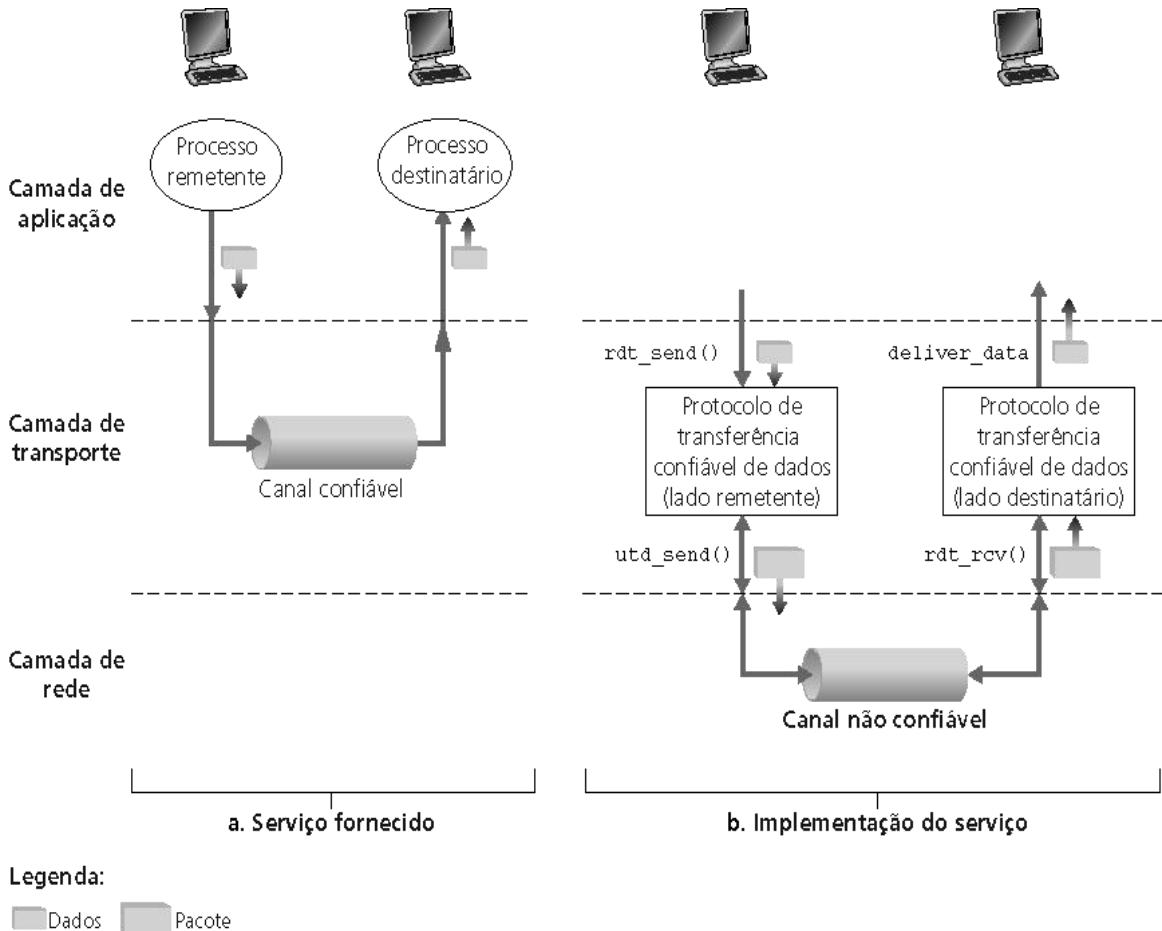
ROBERTO.WILLRICH@UFSC.BR

[HTTPS://MOODLE.UFSC.BR](https://moodle.ufsc.br)

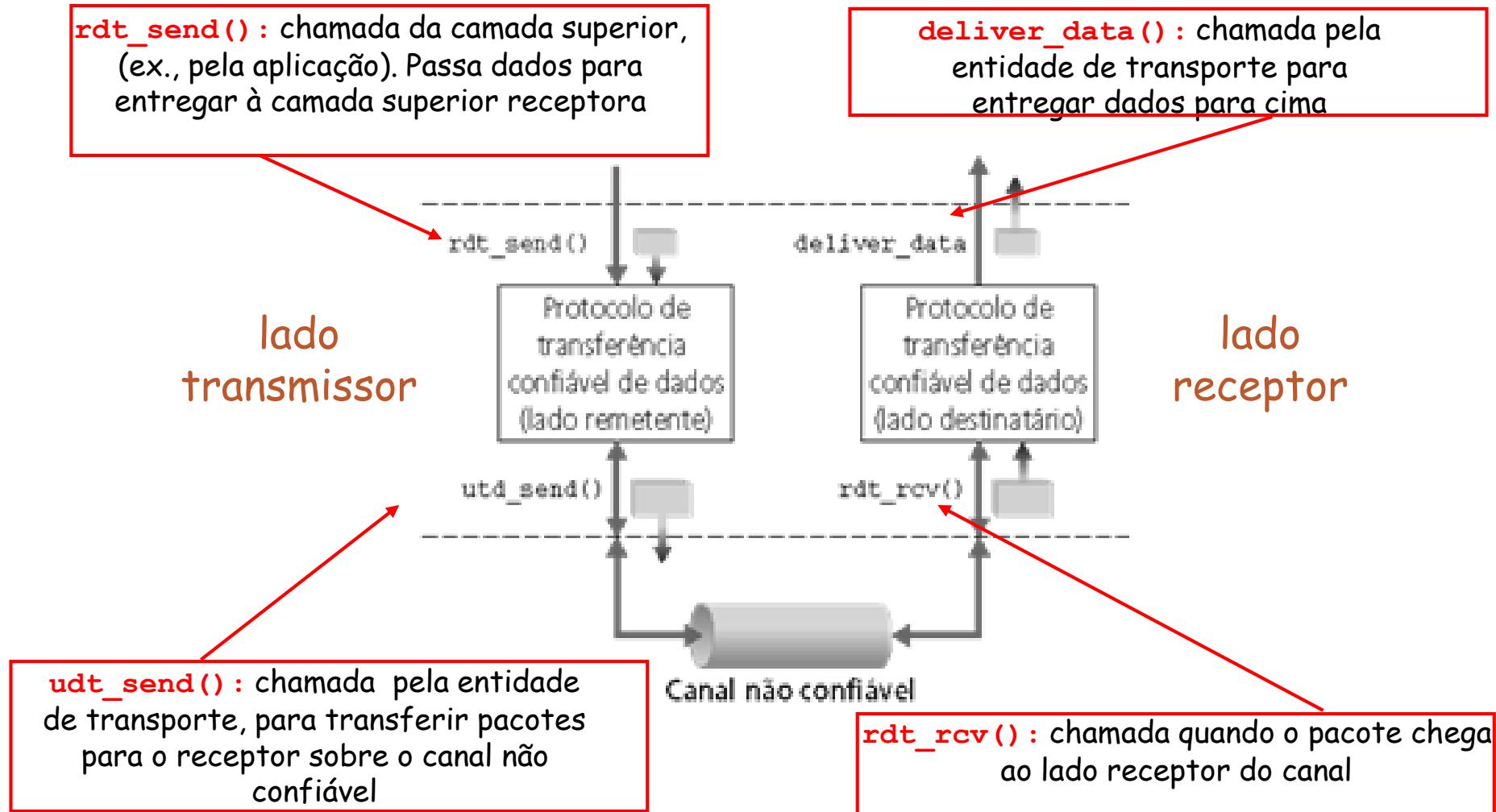
TEMA ABORDADO NA AULA

Princípios de transferência confiável de dados

- Importante nas camadas de aplicação, transporte e enlace
- Top 10 na lista dos tópicos mais importantes de redes!
- Características dos canais não confiáveis determinarão a complexidade dos protocolos confiáveis de transferência de dados



TRANSFERÊNCIA CONFIÁVEL: O PONTO DE PARTIDA

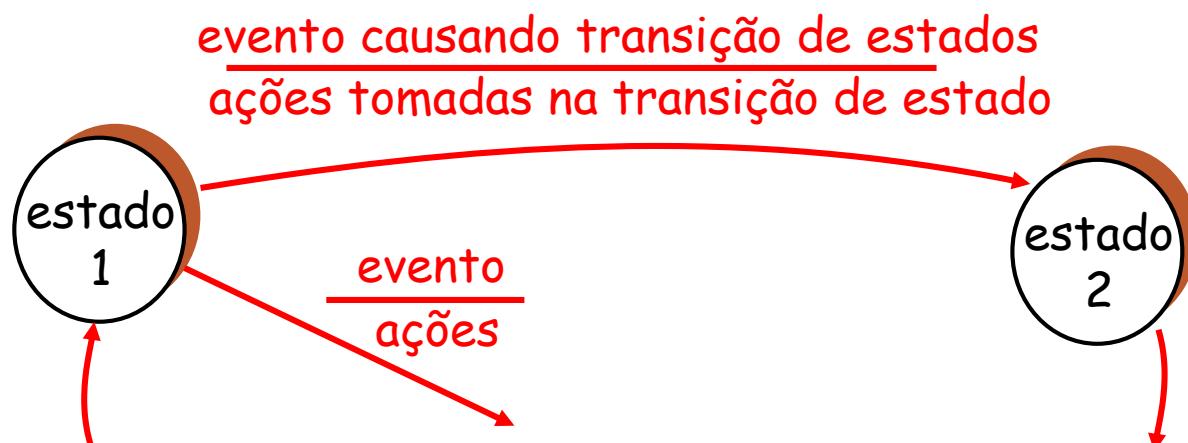


TRANSFERÊNCIA CONFIÁVEL: O PONTO DE PARTIDA

Estudo incremental de mecanismos para prover confiabilidade na transmissão de dados:

- Veremos algumas versões de um protocolo confiável (rdt)
- Consideraremos apenas transferências de dados unidirecionais
 - Mas informação de controle deve fluir em ambas as direções!
- Usaremos máquinas de estados finitos (FSM) para especificar o protocolo transmissor e o receptor

estado: quando neste “estado” o próximo estado fica unicamente determinado pelo próximo evento



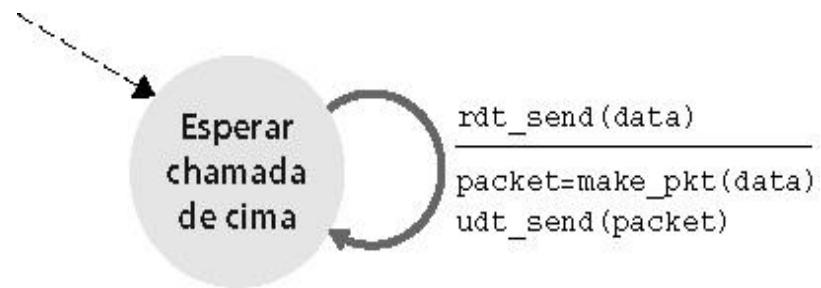
RDT1.0: TRANSFERÊNCIA CONFIÁVEL SOBRE CANAIS CONFIÁVEIS

Canal de transmissão perfeitamente confiável

- Não há erros de bits
- Não há perdas de pacotes
- Seria o serviço oferecido pelo TCP aos protocolos de aplicação (p.e., HTTP)

FSMs separadas para transmissor e receptor:

- Transmissor envia dados para o canal subjacente
- Receptor lê os dados do canal subjacente



a. rdt1.0: lado remetente



b. rdt1.0: lado destinatário

RDT2.0: CANAL COM ERROS DE BIT

Canal subjacente pode trocar valores dos bits num pacote

- Mecanismos de segurança: **Checksum** para detectar erros de bits

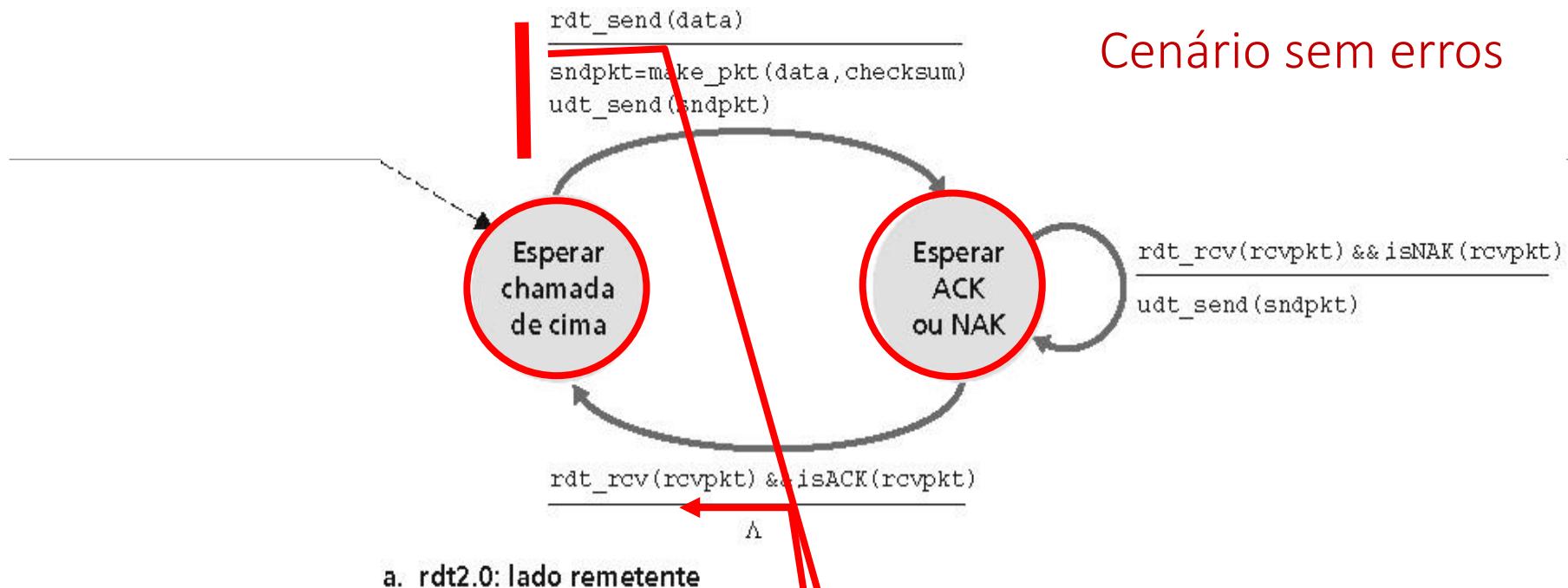
A questão: como recuperar esses erros?

- **Reconhecimentos (ACKs)**: receptor avisa explicitamente ao transmissor que o pacote foi recebido corretamente
- **Reconhecimentos negativos (NAKs)**: receptor avisa explicitamente ao transmissor que o pacote tem erros
- Transmissor reenvia o pacote quando da recepção de um NAK

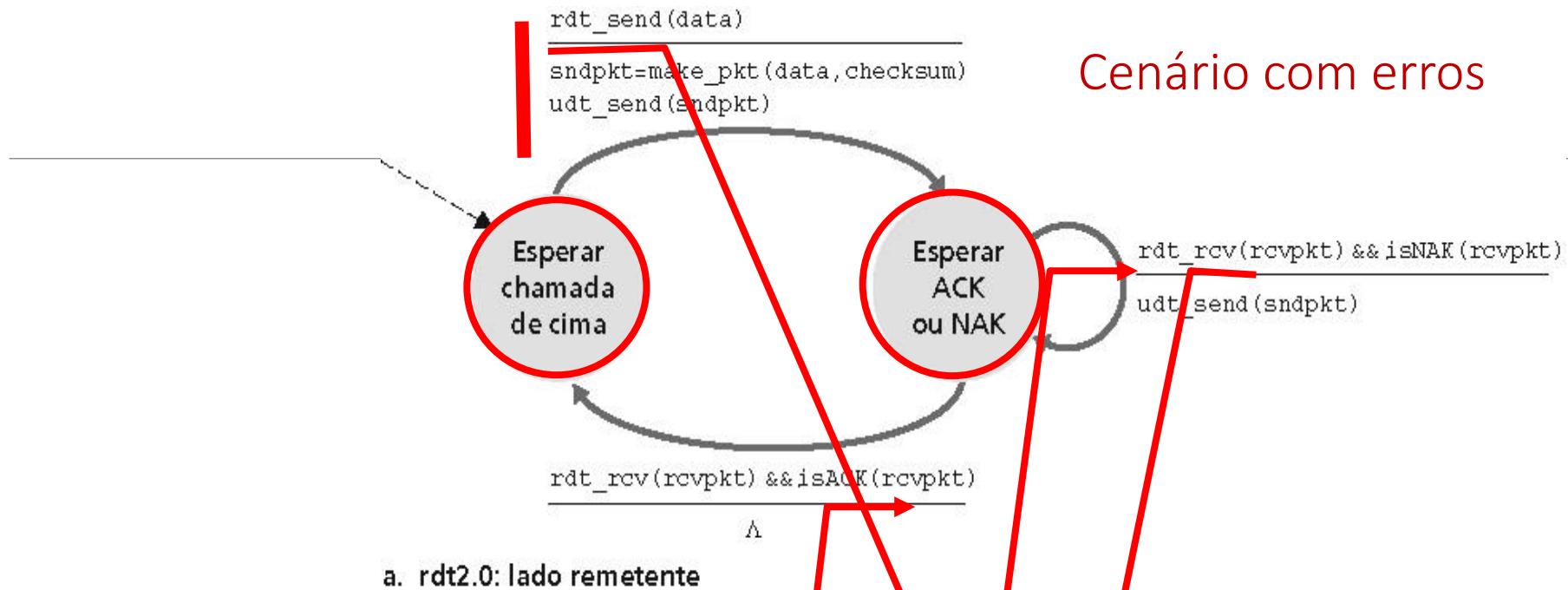
Novos mecanismos no rdt2.0 (além do rdt1.0):

- Detecção de erros
- Retorno do receptor: mensagens de controle (ACK, NAK) rcvr->sender

Cenário sem erros



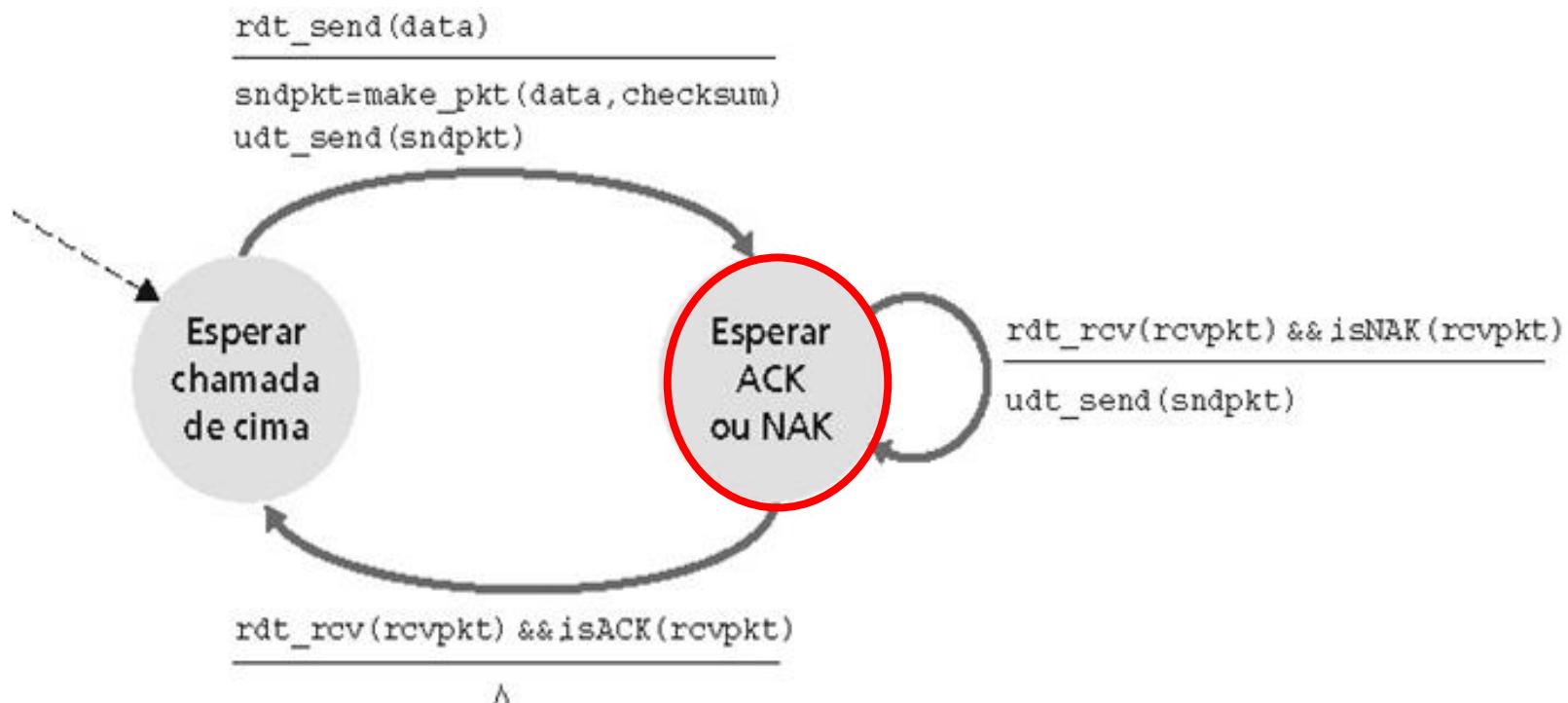
Cenário com erros



RDT2.0 TEM UM PROBLEMA FATAL!

O que ocorre se o ACK/NAK é corrompido?

- Assumimos que não há perda de pacote na rede
- Transmissor não sabe o que aconteceu no receptor!
- Não pode apenas retransmitir : possível duplicata



RDT2.0 TEM UM PROBLEMA FATAL!

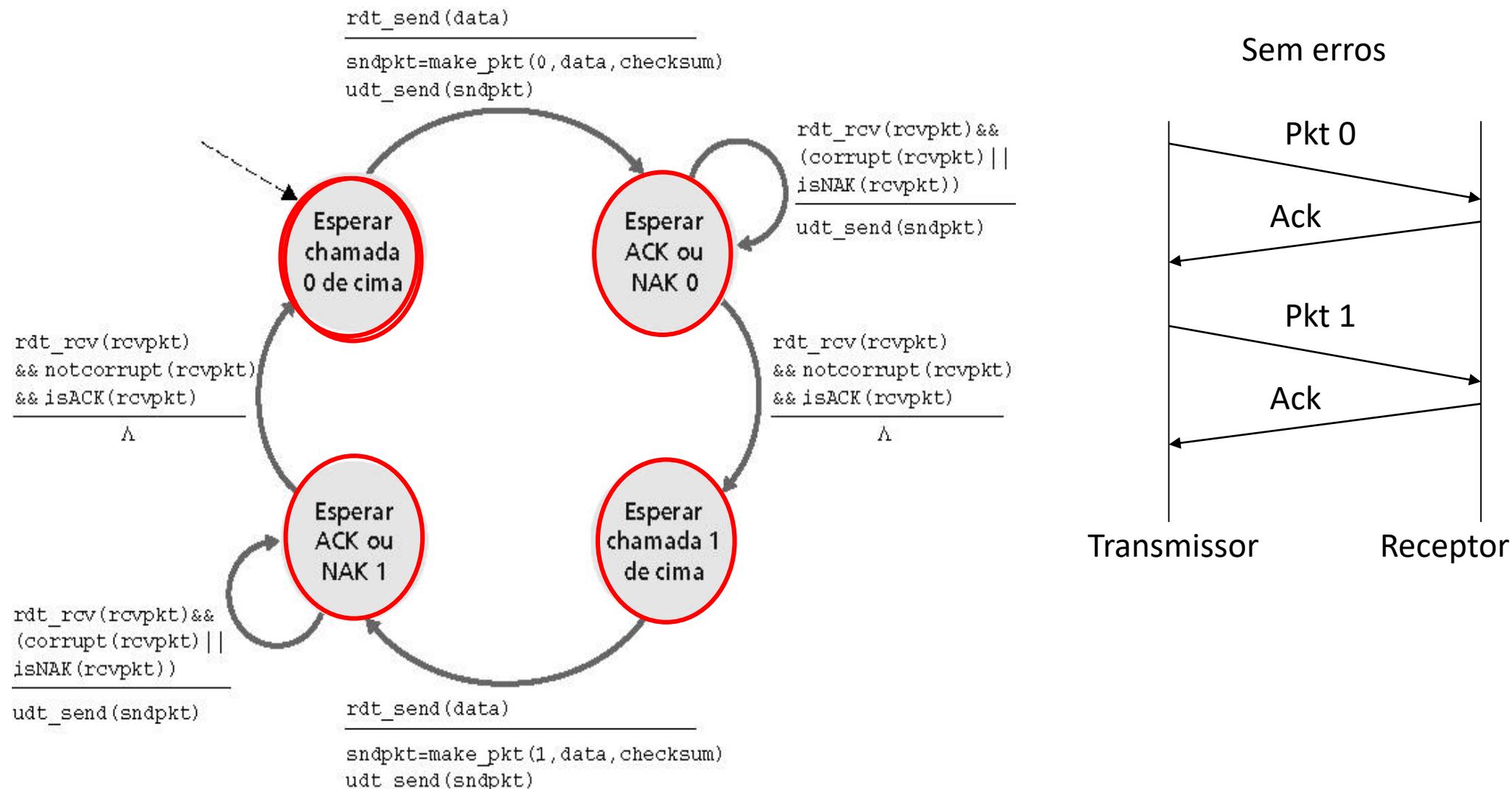
Tratando duplicatas:

- Transmissor acrescenta **número de sequência** em cada pacote
- Transmissor reenvia o último pacote se ACK/NAK for corrompido
- Receptor descarta (não passa para a aplicação) pacotes duplicados

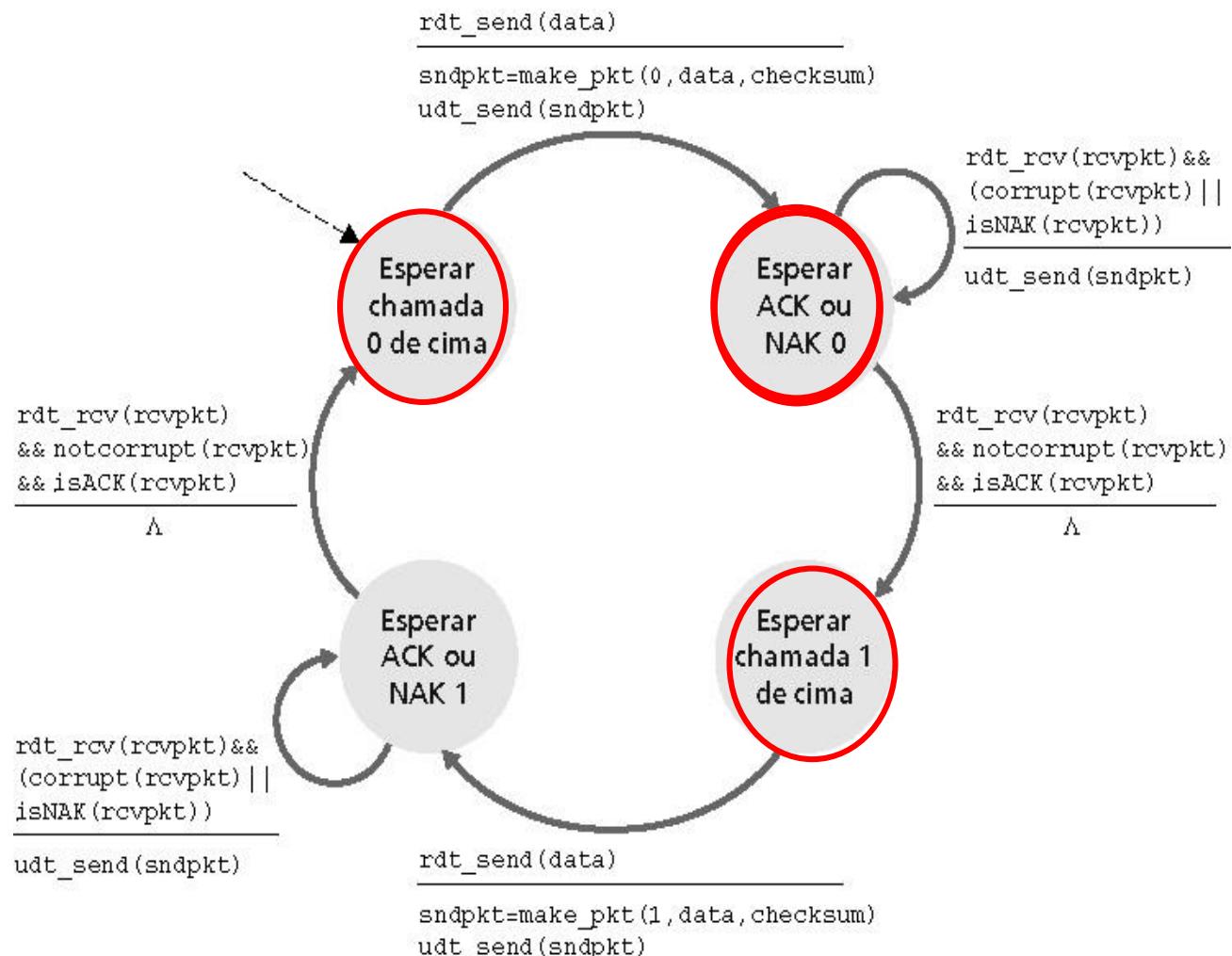
Consideramos o modo transmite e aguarde!!!

- Transmissor envia um pacote e então espera pela resposta do receptor

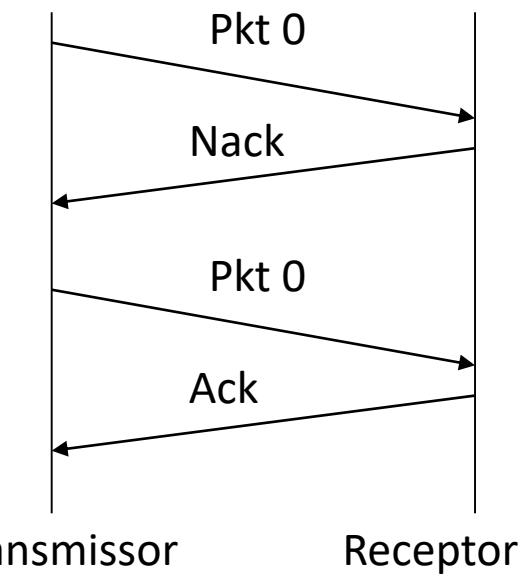
RDT2.1: TRANSMISSOR, TRATA ACK/NAKS PERDIDOS



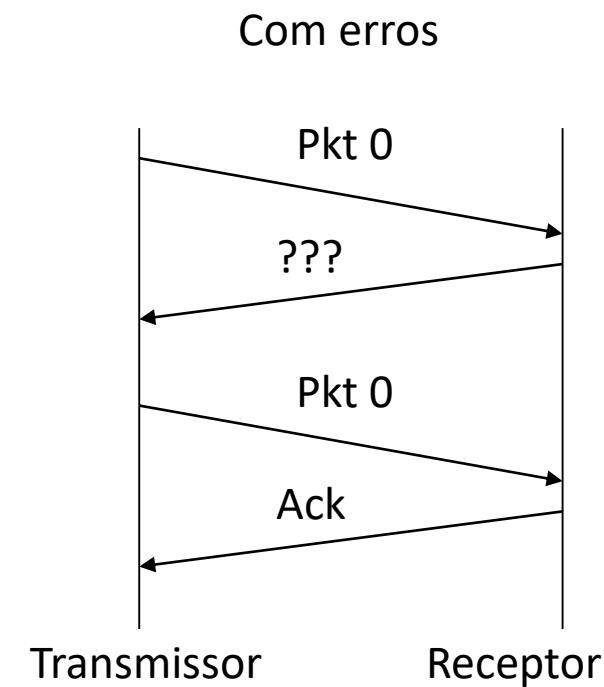
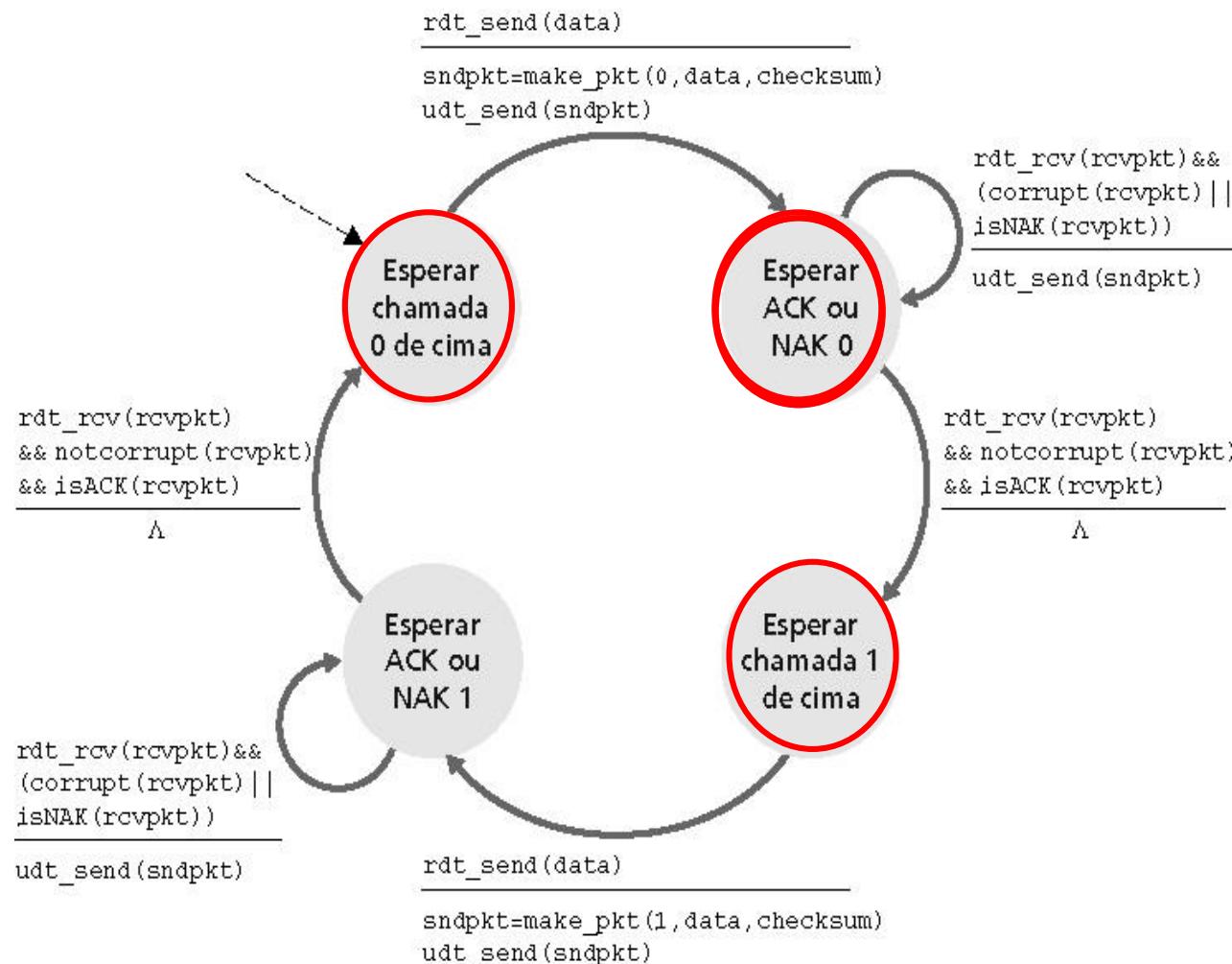
RDT2.1: TRANSMISSOR, TRATA ACK/NAKS PERDIDOS



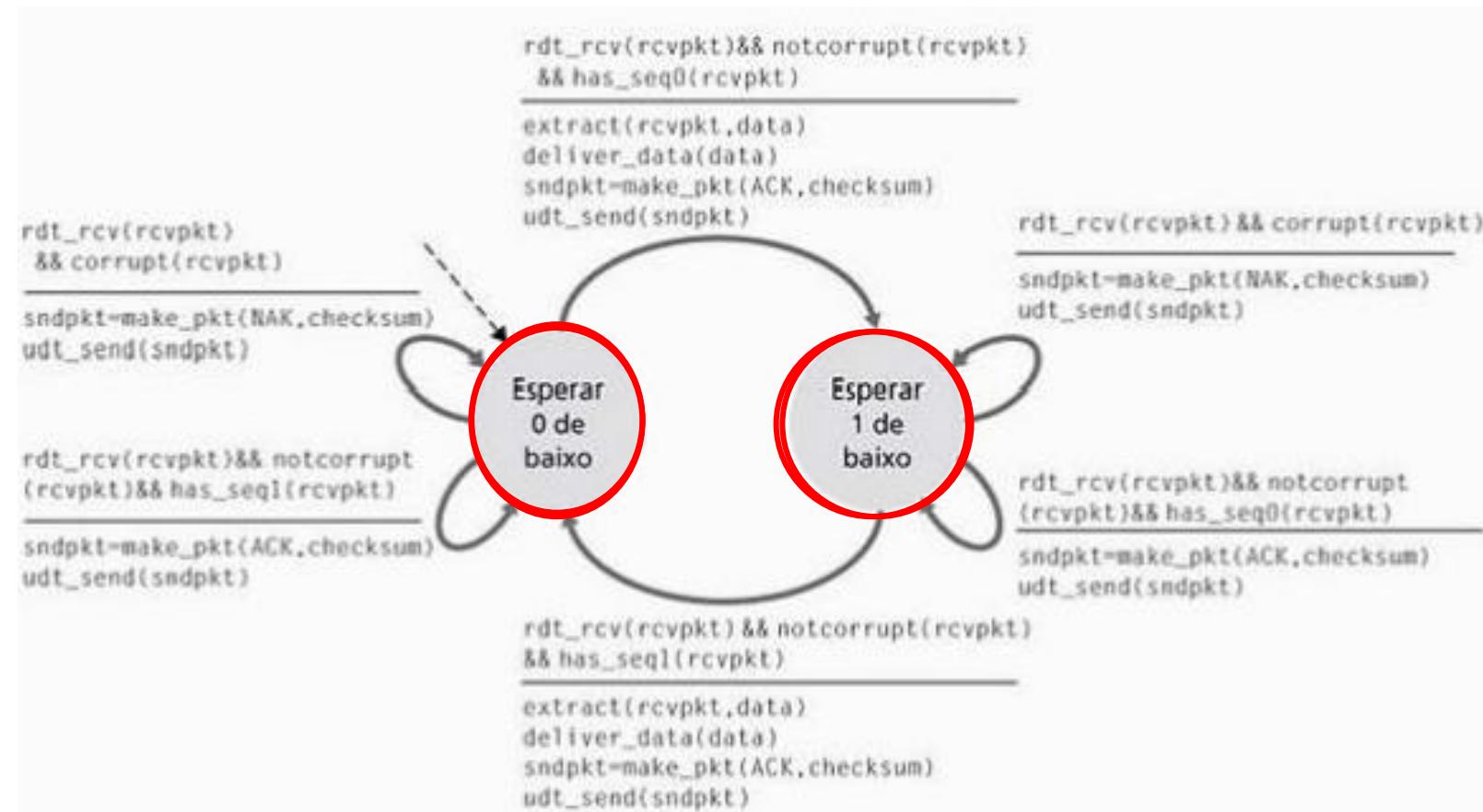
Com erros



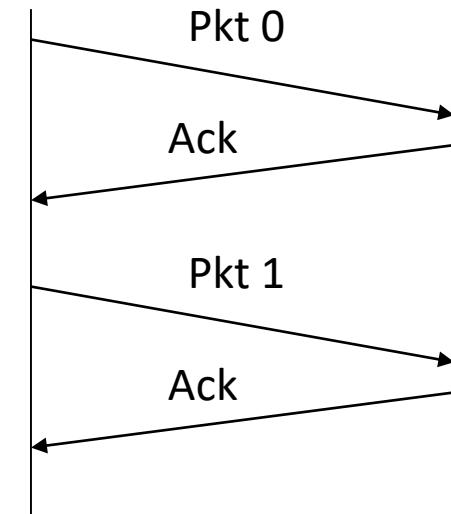
RDT2.1: TRANSMISSOR, TRATA ACK/NAKS PERDIDOS



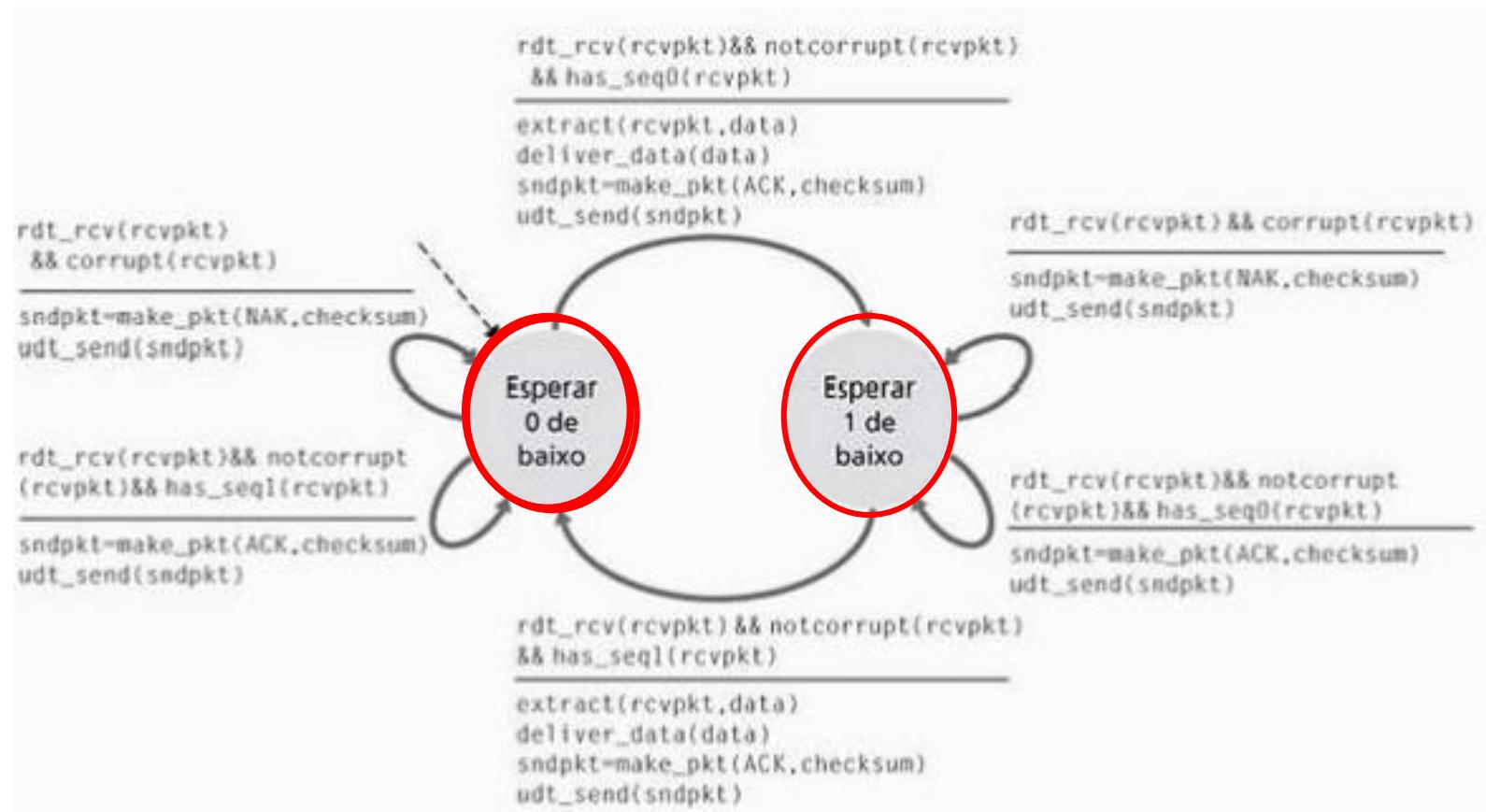
RDT2.1: RECEPTOR, TRATA ACK/NAKS PERDIDOS



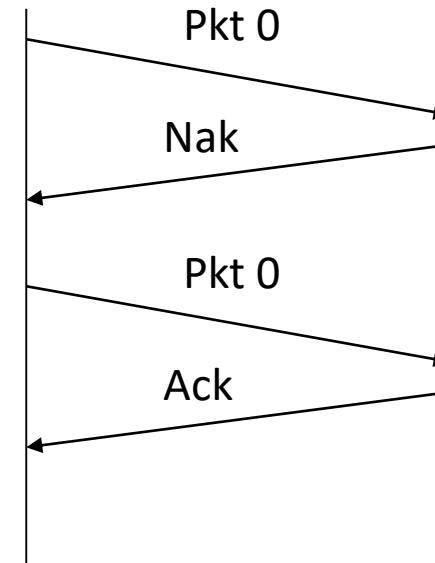
Sem erros



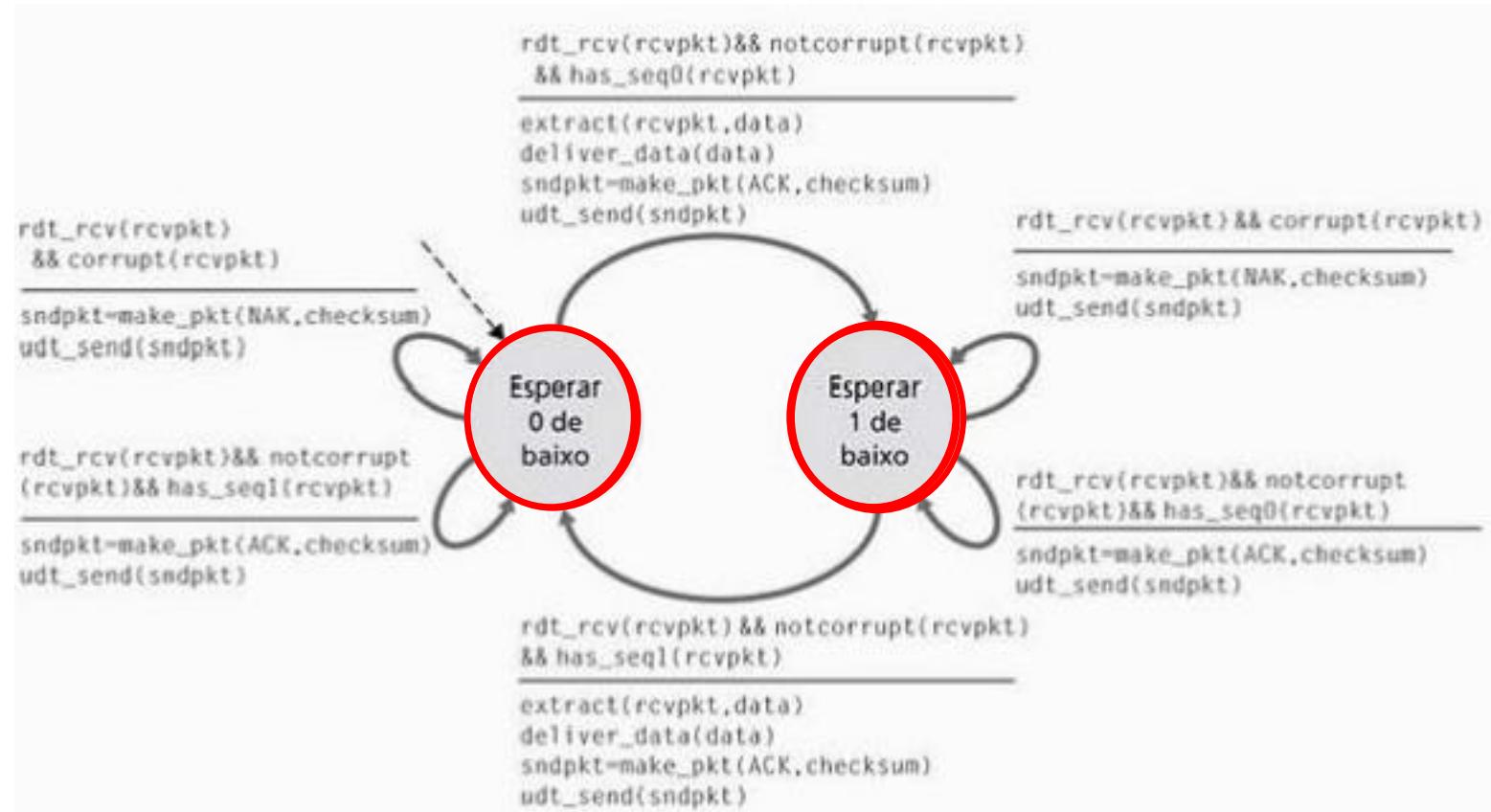
RDT2.1: RECEPTOR, TRATA ACK/NAKS PERDIDOS



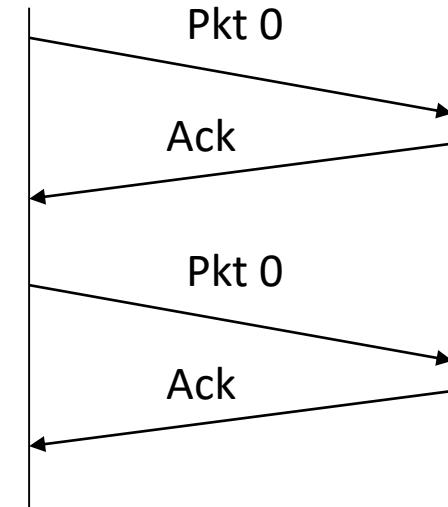
Com erros



RDT2.1: RECEPTOR, TRATA ACK/NAKS PERDIDOS



Duplicata



RDT2.1: DISCUSSÃO

Transmissor:

- Adiciona número de sequência ao pacote
- Dois números (0 e 1) bastam. Por quê?
- Duas vezes o número de estados
 - O estado deve “lembrar” se o pacote “corrente” tem número de sequência 0 ou 1

Receptor:

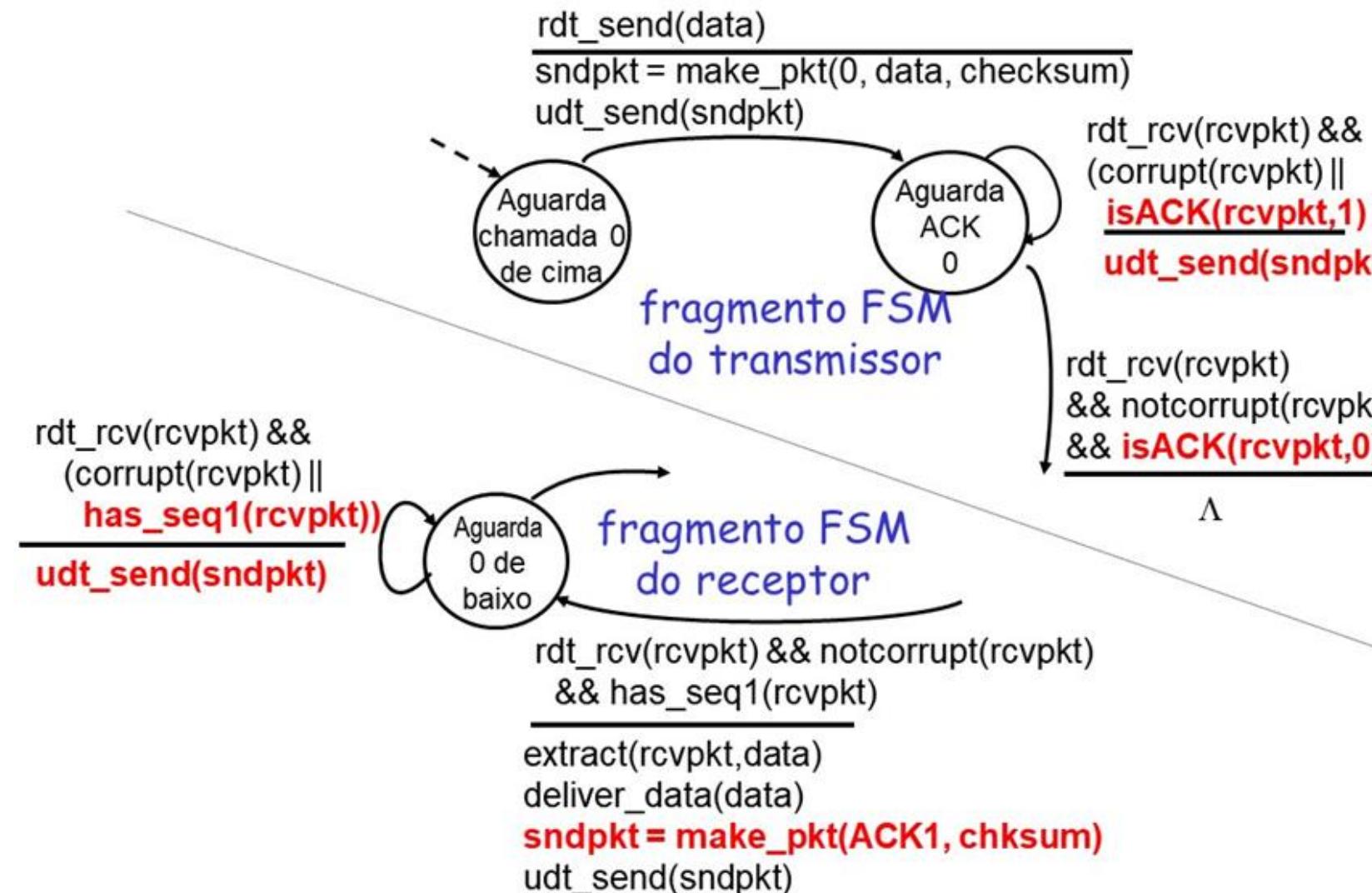
- Deve verificar se o pacote recebido é duplicado
 - Estado indica se o pacote 0 ou 1 é esperado

RDT2.2: UM PROTOCOLO SEM NAK

Mesma funcionalidade do rdt2.1, usando somente ACKs

- Em vez de enviar NAK, o receptor envia ACK para o último pacote recebido sem erro
- Receptor deve incluir explicitamente o número de sequência do pacote sendo reconhecido
- ACKs duplicados no transmissor resultam na mesma ação do NAK:
 - retransmissão do pacote corrente

RDT2.2: FRAGMENTOS DO TRANSMISSOR E DO RECEPTOR



RDT3.0: CANAIS COM ERROS E PERDAS

Nova hipótese: canal de transmissão pode também perder pacotes

- Checksum, números de sequência, ACKs, retransmissões serão de ajuda, mas não o bastante

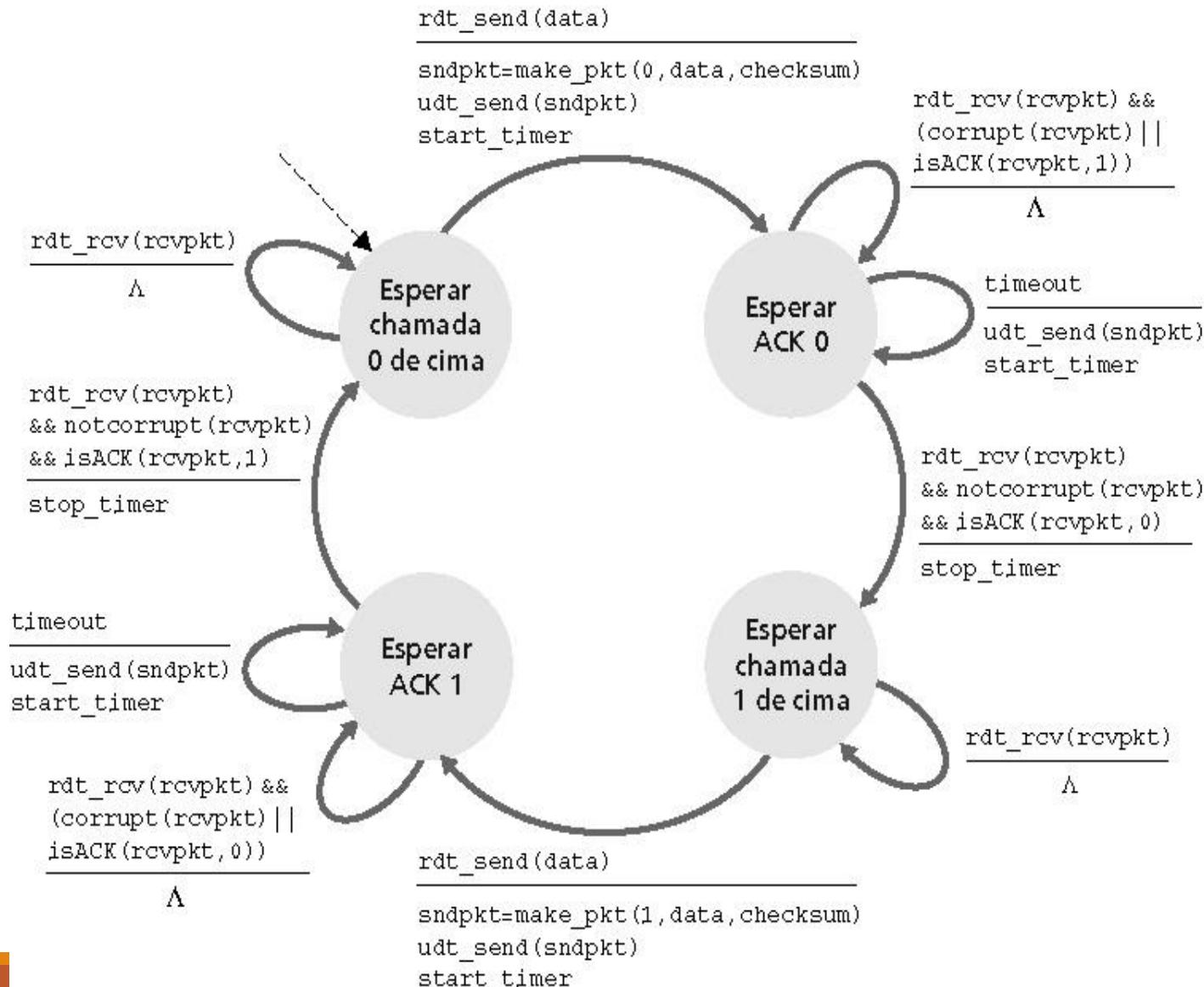
Abordagem: transmissor espera um tempo “razoável” pelo ACK

- Retransmite se nenhum ACK for recebido nesse tempo
- Se o pacote (ou ACK) estiver apenas atrasado (não perdido):
 - Retransmissão será duplicata, mas os números de sequência já tratam com isso

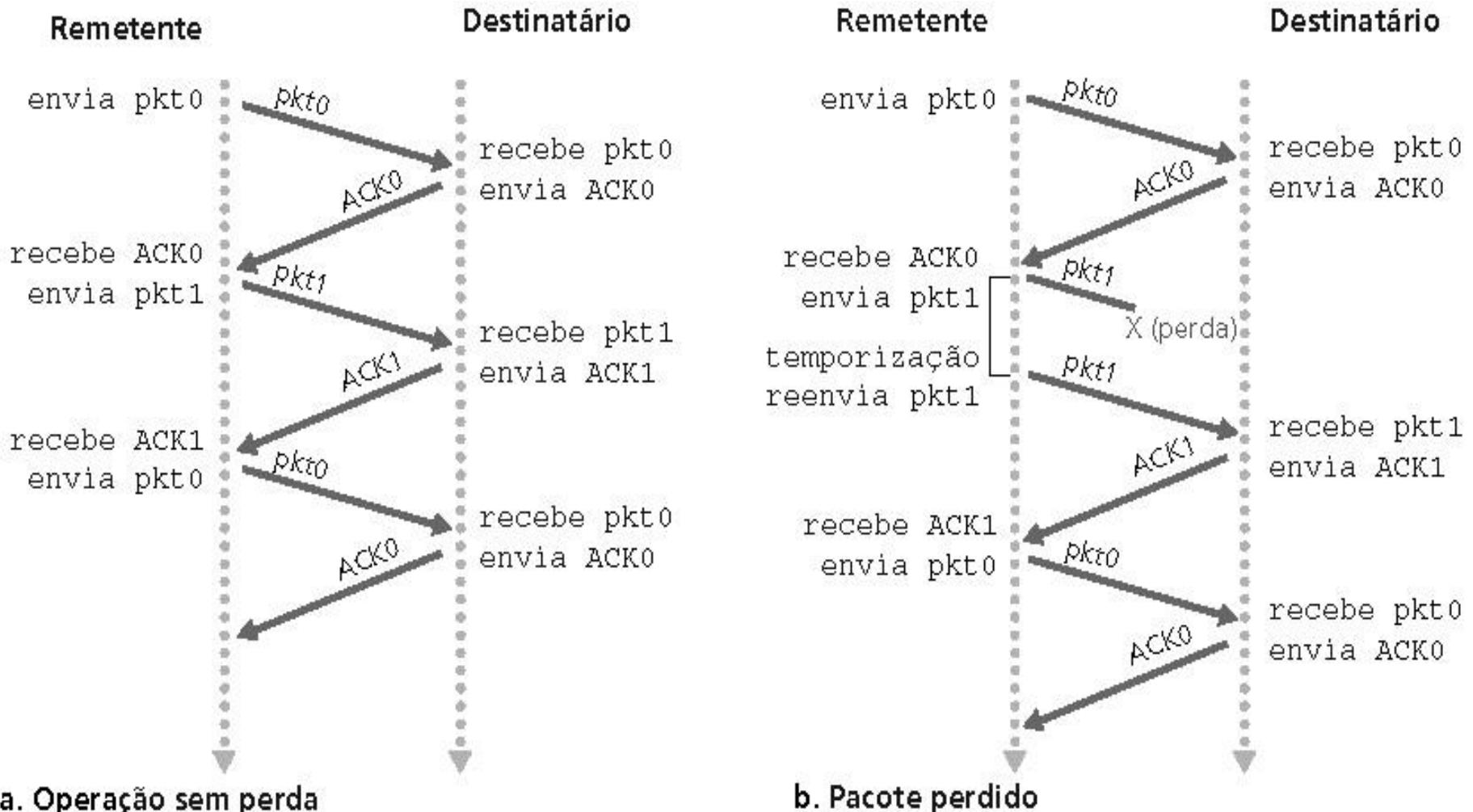
Receptor

- Deve especificar o número de sequência do pacote sendo reconhecido
- Exige um temporizador decrescente

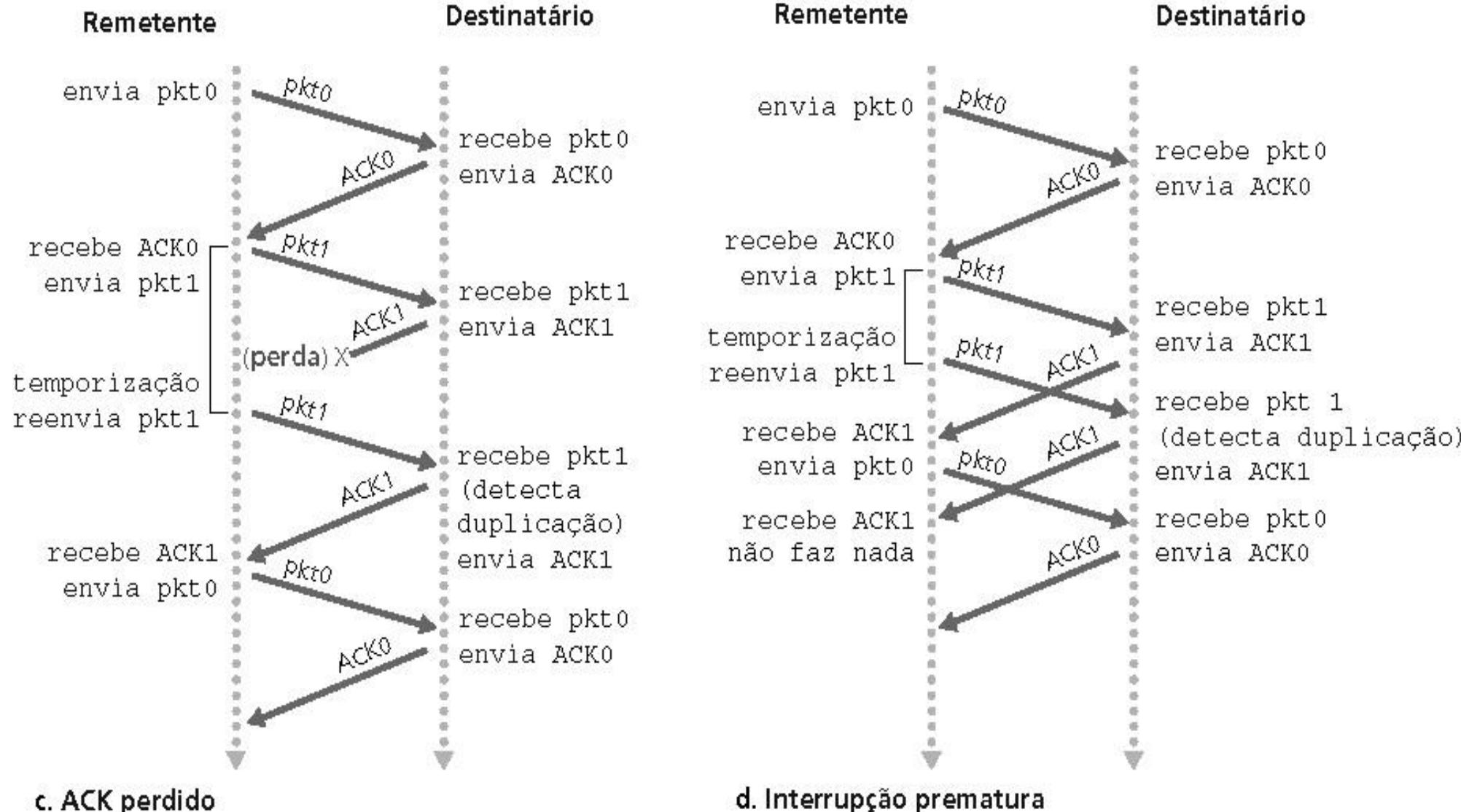
TRANSMISSOR RDT3.0



RDT3.0 EM AÇÃO



RDT3.0 EM AÇÃO



Pontos Importantes

Princípios da transferência confiável de dados

- Introdução à confiabilidade do TCP
- Entender conceitos de checksum e número de sequência e reconhecimento

CAP 5. CAMADA DE TRANSPORTE

AULA 4: PRINCÍPIOS DE TRANSFERÊNCIA CONFIÁVEL DE
DADOS (CONTINUAÇÃO)

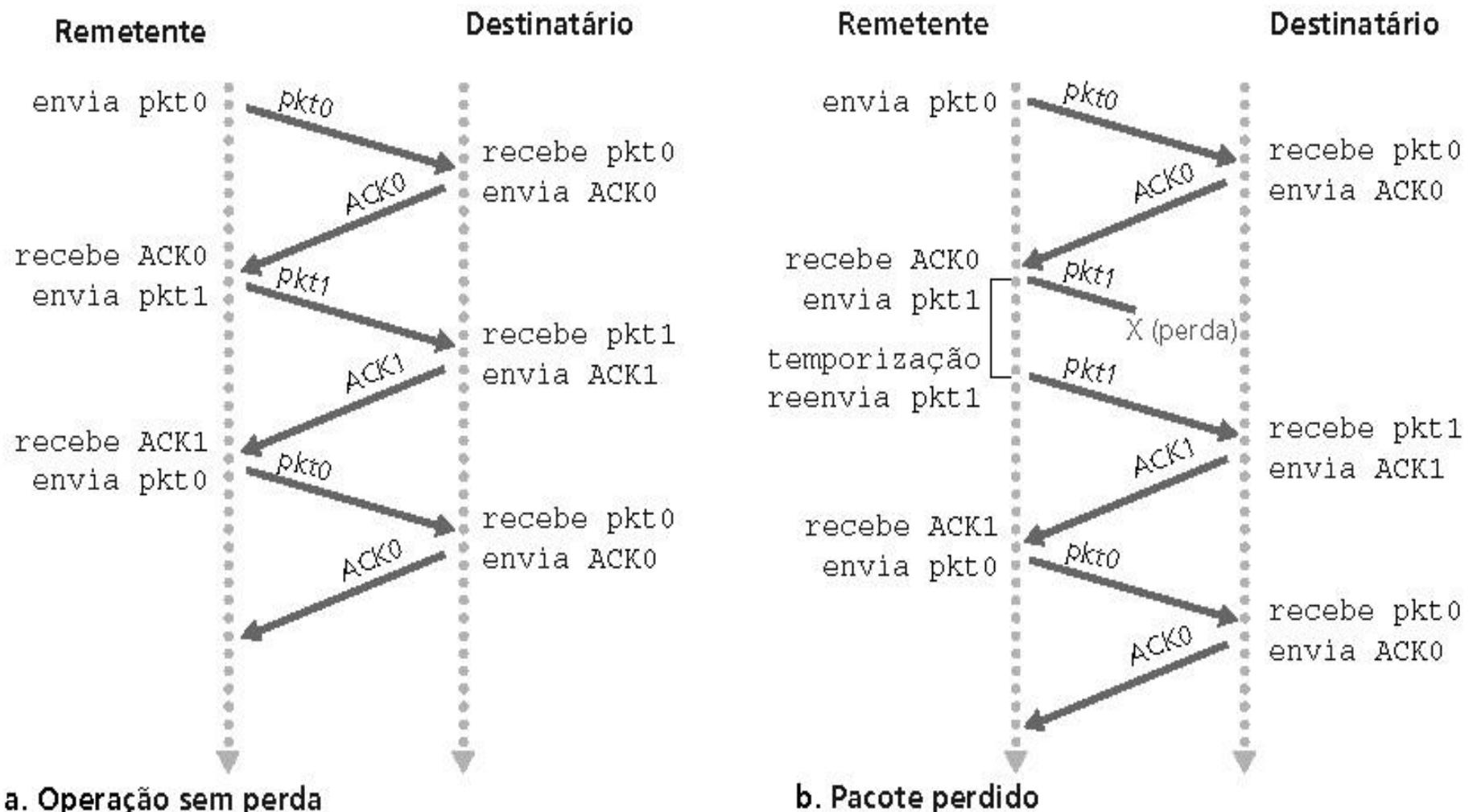
INE5422 REDES DE COMPUTADORES II

PROF. ROBERTO WILLRICH (INE/UFSC)

ROBERTO.WILLRICH@UFSC.BR

[HTTPS://MOODLE.UFSC.BR](https://moodle.ufsc.br)

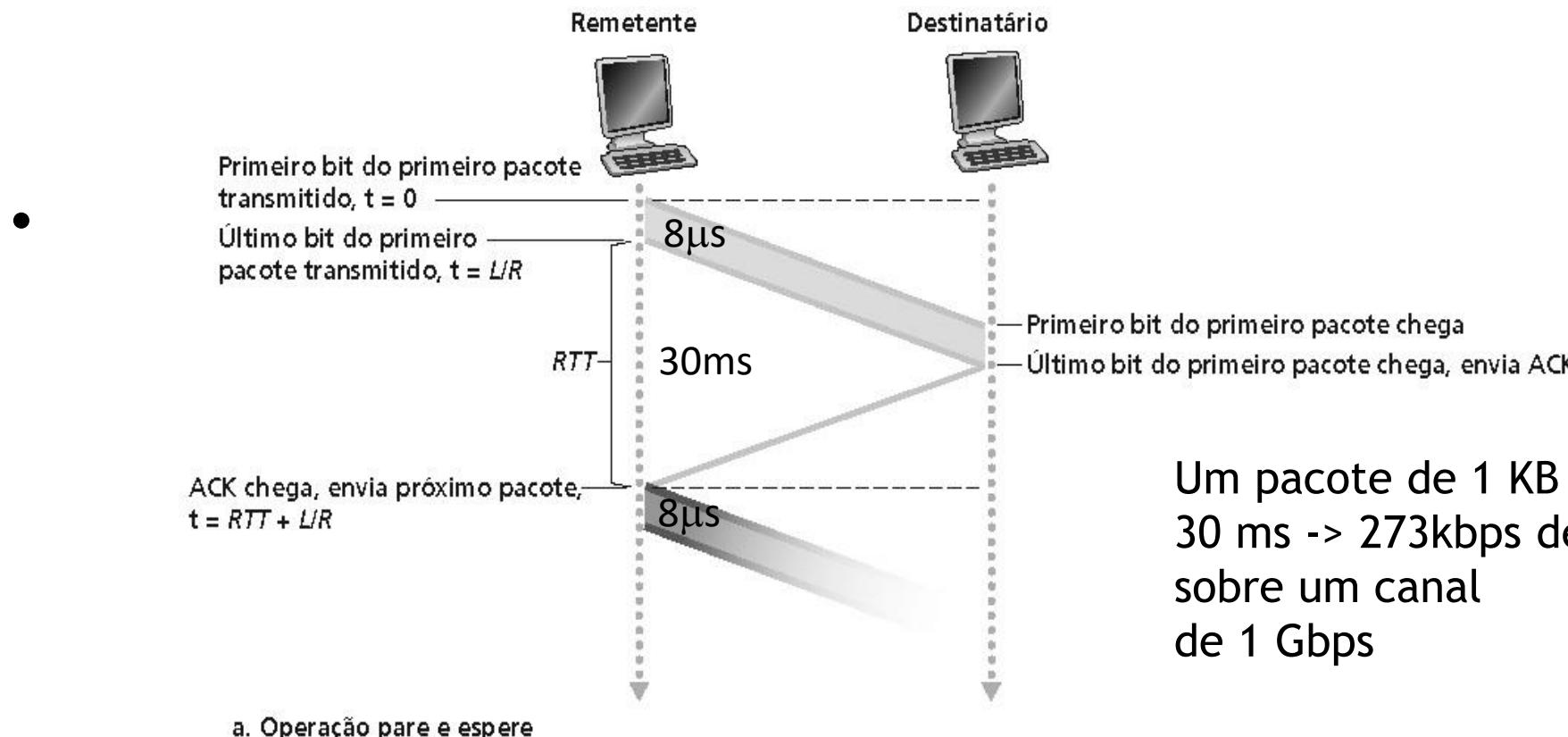
RDT3.0 EM AÇÃO



RDT3.0: OPERAÇÃO PARE E ESPERE

rdt3.0 funciona, mas o desempenho é sofrível

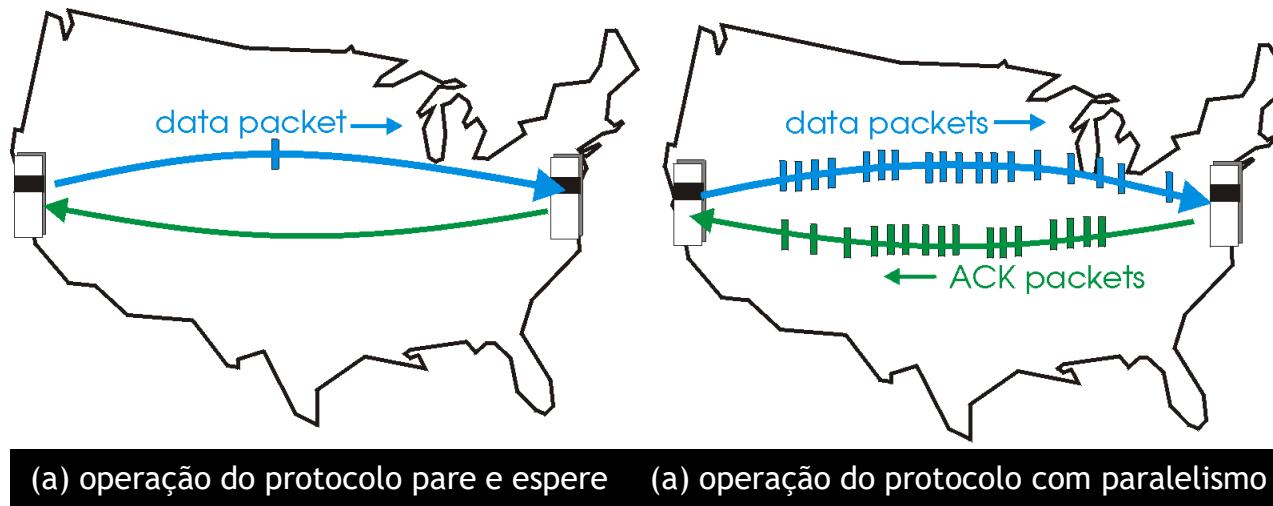
- Exemplo: enlace de 1 Gbps, 15 ms de atraso de ida ao destino, pacotes de 1 KB:



PROTOCOLOS COM PARALELISMO (PIPELINING)

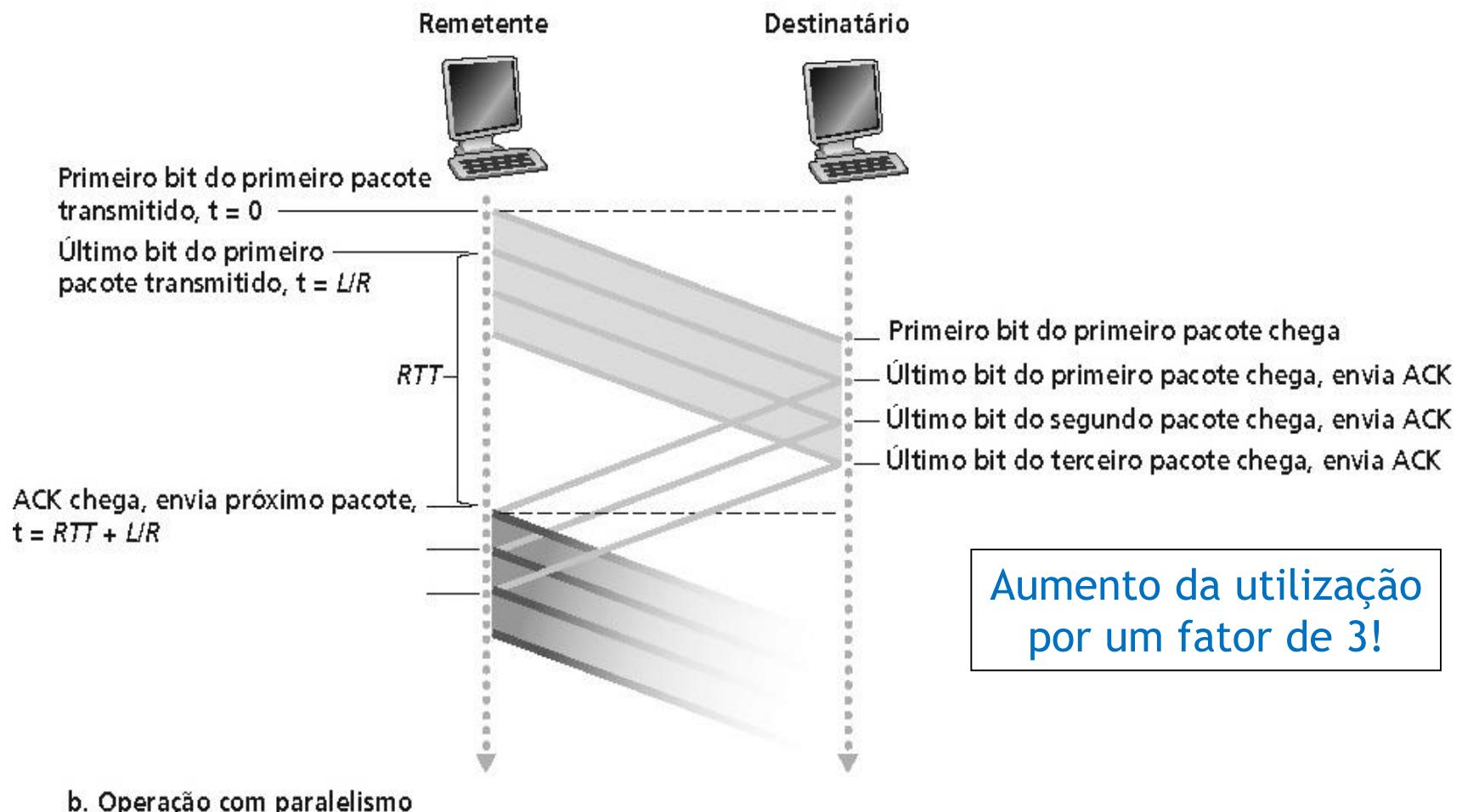
Paralelismo: transmissor envia vários pacotes ao mesmo tempo, todos esperando para serem reconhecidos

- Faixa de números de sequência deve ser aumentada



- Duas formas genéricas de protocolos com paralelismo: go-Back-N, retransmissão seletiva

PIPELINING: AUMENTO DA UTILIZAÇÃO



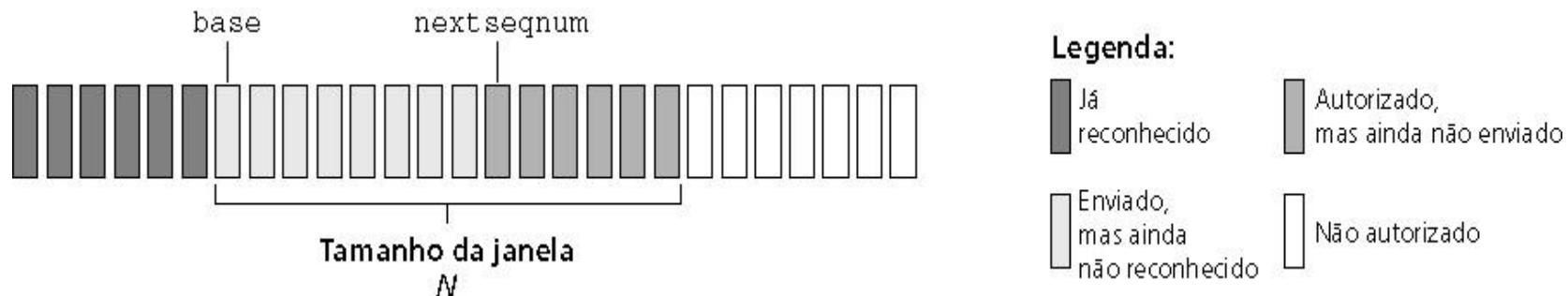
GO-BACK-N

Transmissor:

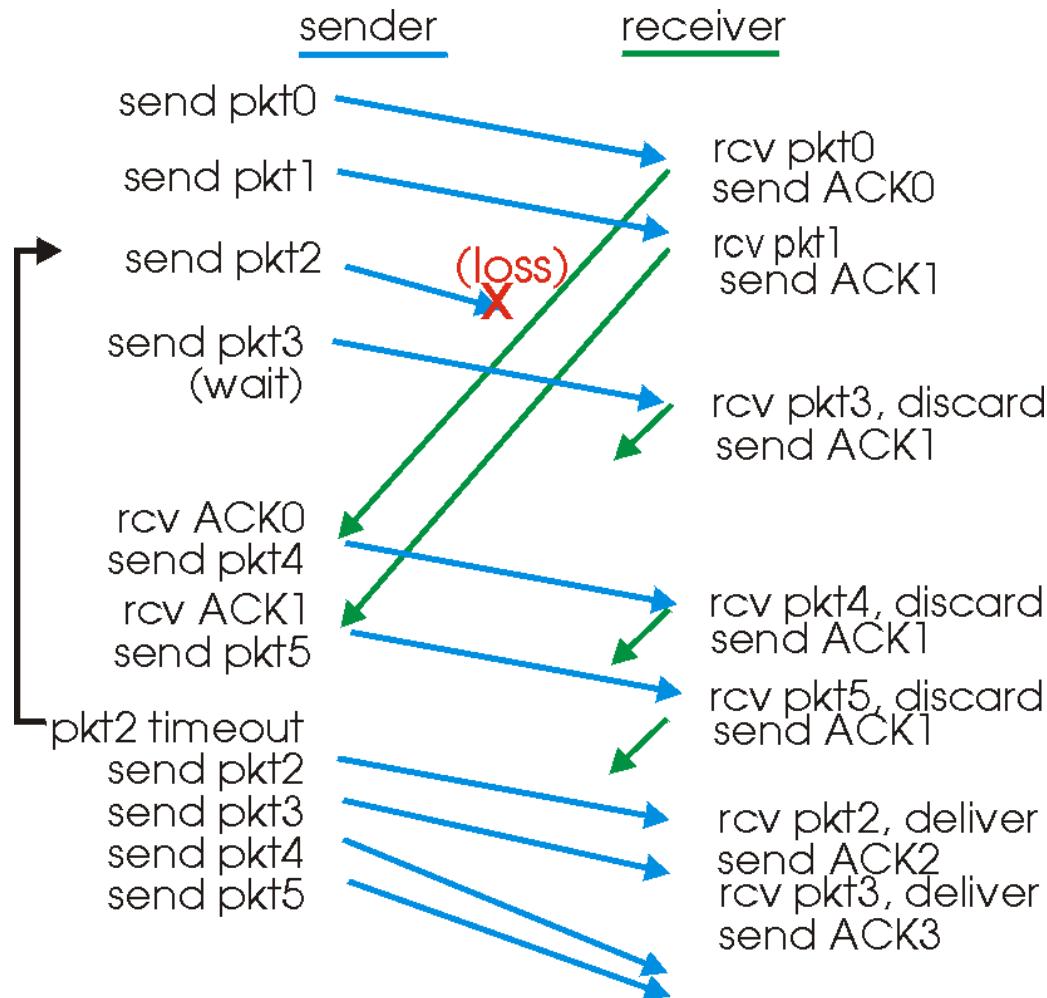
- Número de sequência com k bits no cabeçalho do pacote
- “janela” de até N pacotes que podem ser enviados sem esperar confirmação dos anteriores
- ACK(n) Cumulativo: reconhece todos os pacotes até o número de sequência confirmado
- Janela desliza na confirmação do(s) pacote(s)
- Lança temporizador para cada pacote enviado
 - Tempo de confirmação (do pacote n): retransmite pacote n e todos os pacotes com número de sequência maior que estejam dentro da janela

Receptor

- Pacotes fora de ordem:
 - Descarta (não armazena) -> não há buffer de recepção!
 - Reconhece pacote com o mais alto número de sequência em ordem



GBN EM AÇÃO (JANELA DE 4 PACOTES)



RETRANSMISSÃO SELETIVA

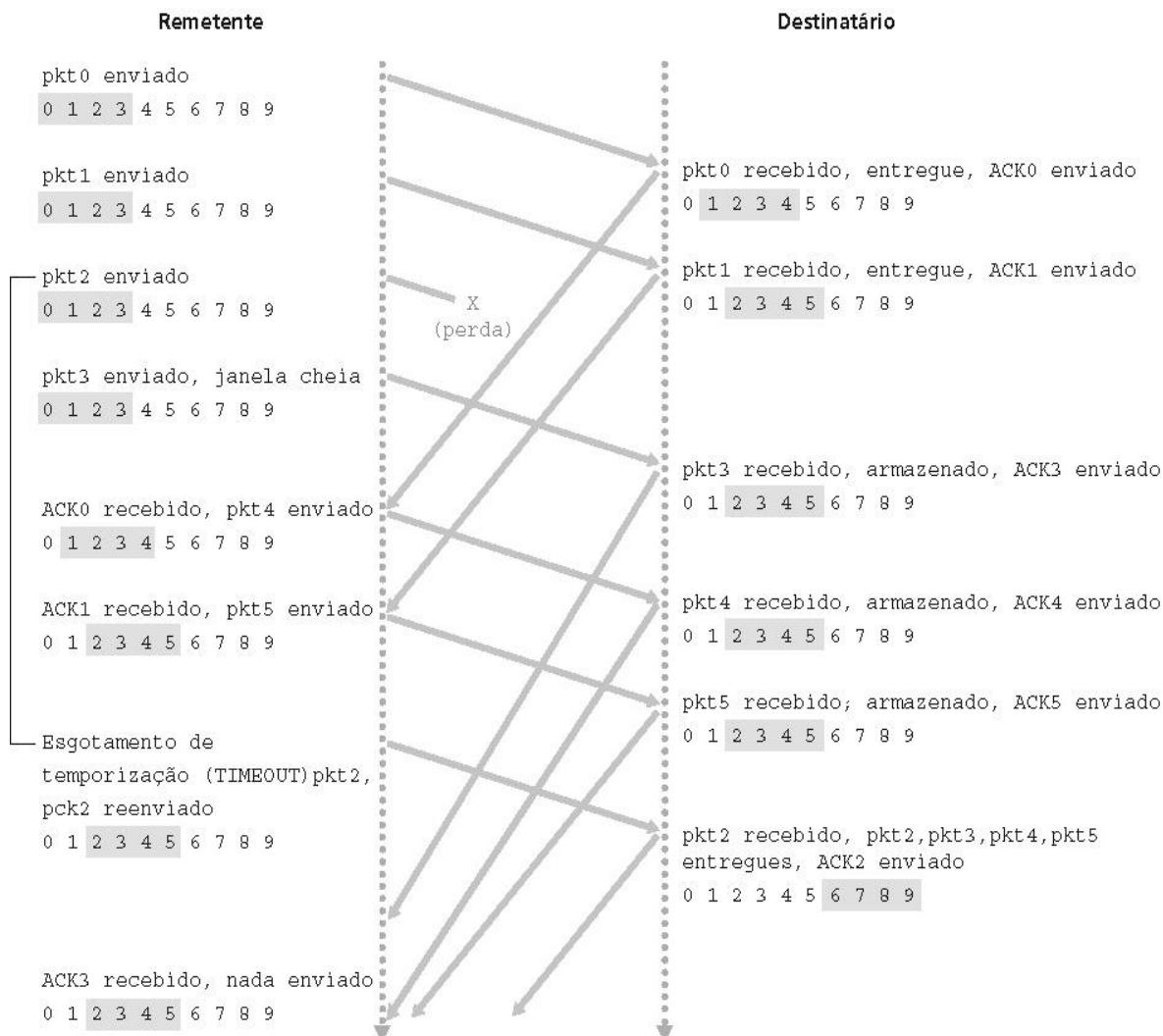
Transmissor

- Janela de transmissão
 - N números de sequência consecutivos : limita a quantidade de pacotes enviados sem confirmação
- Transmissor temporiza cada pacote não reconhecido
- Somente reenvia os pacotes para os quais um ACK não foi recebido

Receptor

- reconhece individualmente todos os pacotes recebidos corretamente
- Armazena pacotes, quando necessário, para eventual entrega em ordem para a camada superior

RETRANSMISSÃO SELETIVA EM AÇÃO



Pontos Importantes

Princípios da transferência confiável de dados

- Entender os tipos de retransmissão

CAP 5. CAMADA DE TRANSPORTE

AULA 5: PROTOCOLO TCP

INE5422 REDES DE COMPUTADORES II

PROF. ROBERTO WILLRICH (INE/UFSC)

ROBERTO.WILLRICH@UFSC.BR

[HTTPS://MOODLE.UFSC.BR](https://moodle.ufsc.br)

PROTOCOLO TCP

Orientado à conexão

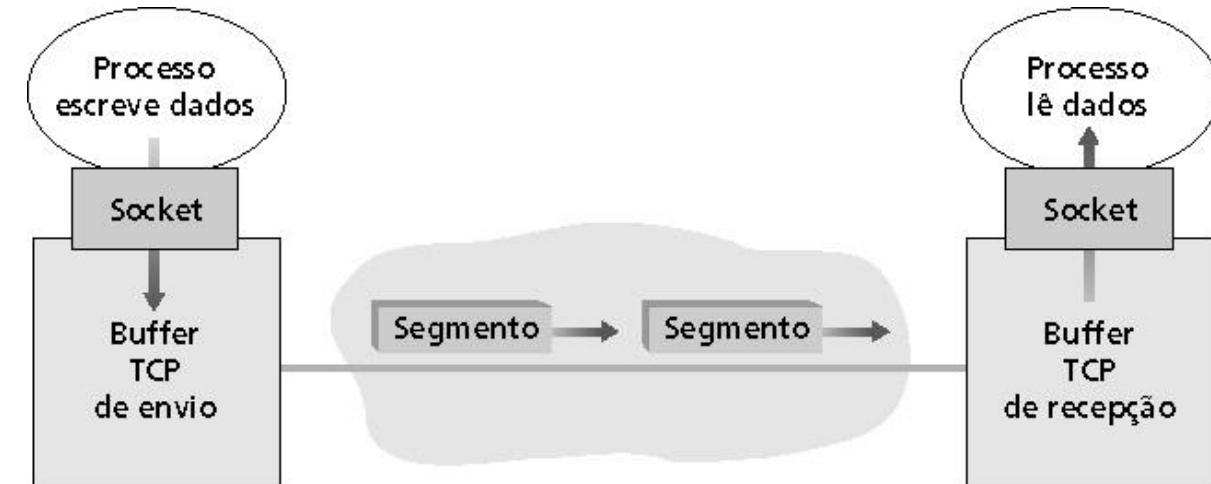
- Apresentação (troca de mensagens de controle)
- Inicia o estado do transmissor e do receptor antes da troca de dados
 - Buffers de transmissão e de recepção

Ponto-a-ponto

- Um transmissor, um receptor

Dados full-duplex:

- Transmissão bidirecional na mesma conexão



PROTOCOLO TCP

Pipelined (Dutado)

- Transporte confiável de pacotes via Ack
- Controle de congestionamento e de fluxo definem tamanho da janela

Fluxo de bytes sequencial

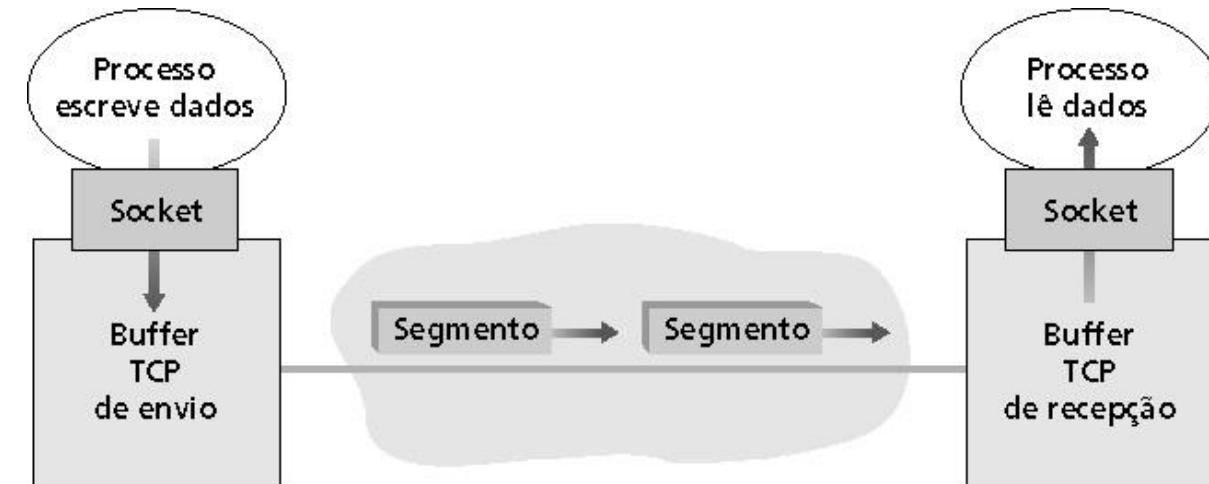
- Não há inversão de ordem de mensagens

Controle de fluxo:

- Transmissor não esgota a capacidade do receptor

Controle de congestionamento

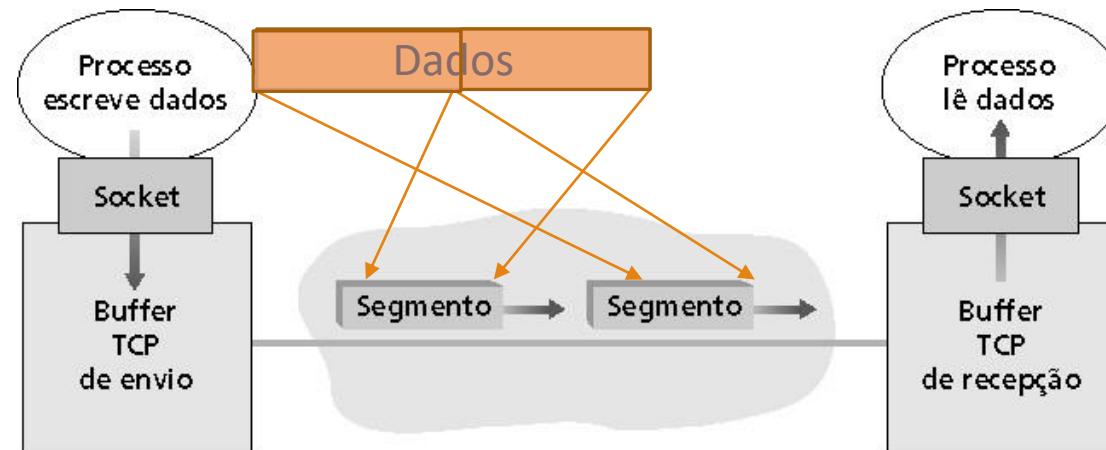
- Tenta evitar a sobrecarga da rede



PROTOCOLO TCP

Buffer de emissão e recepção

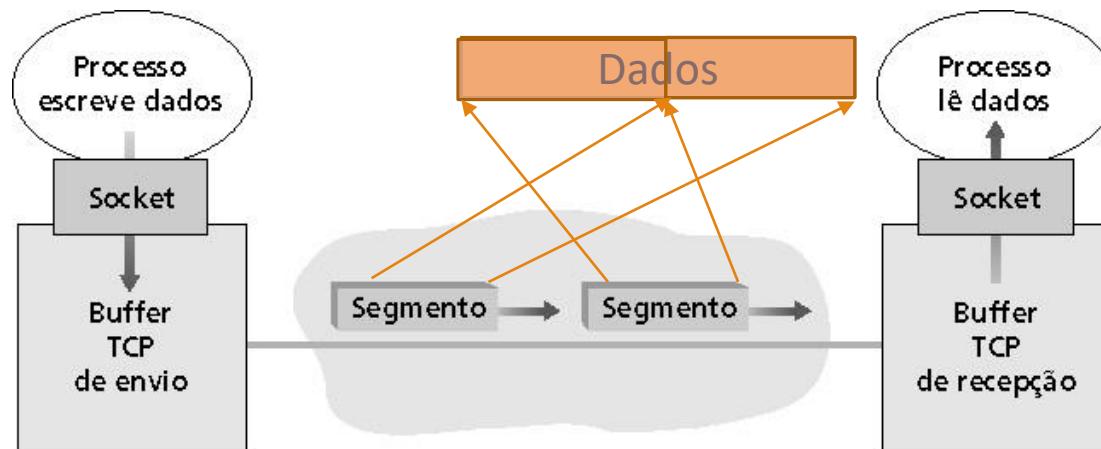
- Processo Cliente envia uma string de dados através da porta do processo
- String de dados é entregue ao processo TCP
 - Que envia a string para o buffer emissor TCP
 - De tempos-em-tempos segmentos de dados são extraídos do buffer
 - MSS: tamanho máximo do segmento - tamanho escolhido para evitar fragmentação no IP
 - TCP adiciona o cabeçalho formando o segmento TCP
 - Envia via rede para o buffer receptor TCP



PROTOCOLO TCP

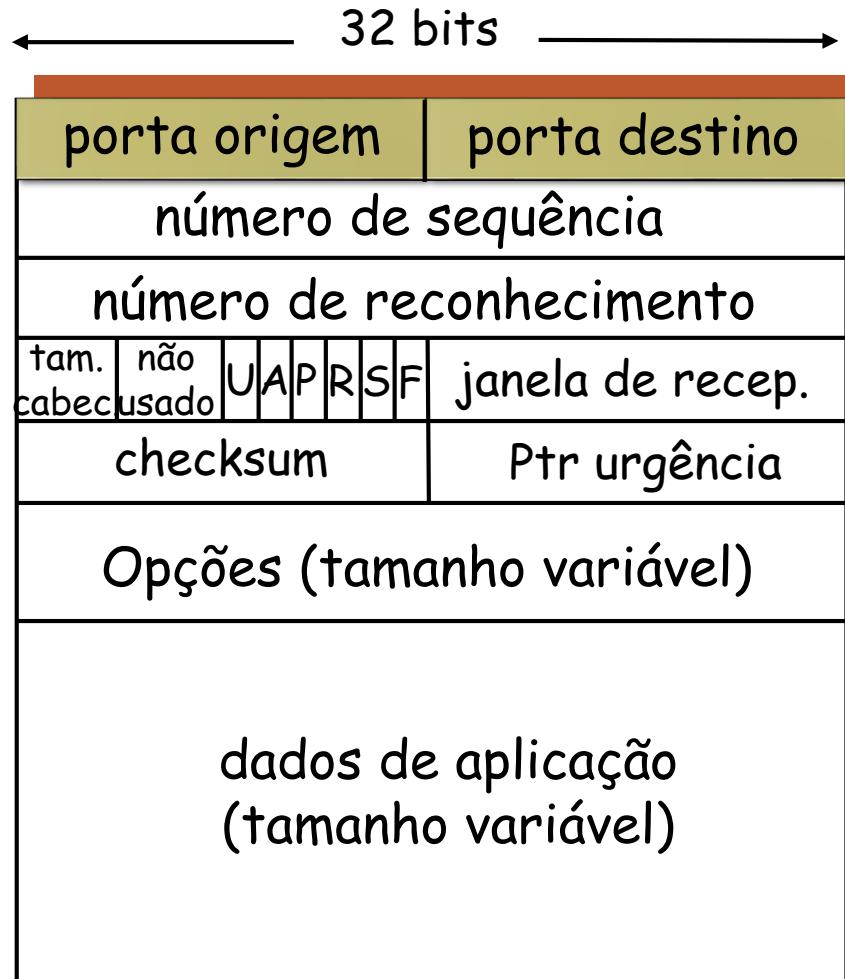
Buffer de emissão e recepção

- No host receptor
 - Segmento é colocado no buffer receptor TCP
 - Aplicação lê dados
- Tanto no emissor como receptor existem buffers de emissão e recepção
 - Fluxo de dados bidirecional na mesma conexão



ESTRUTURA DO SEGMENTO TCP

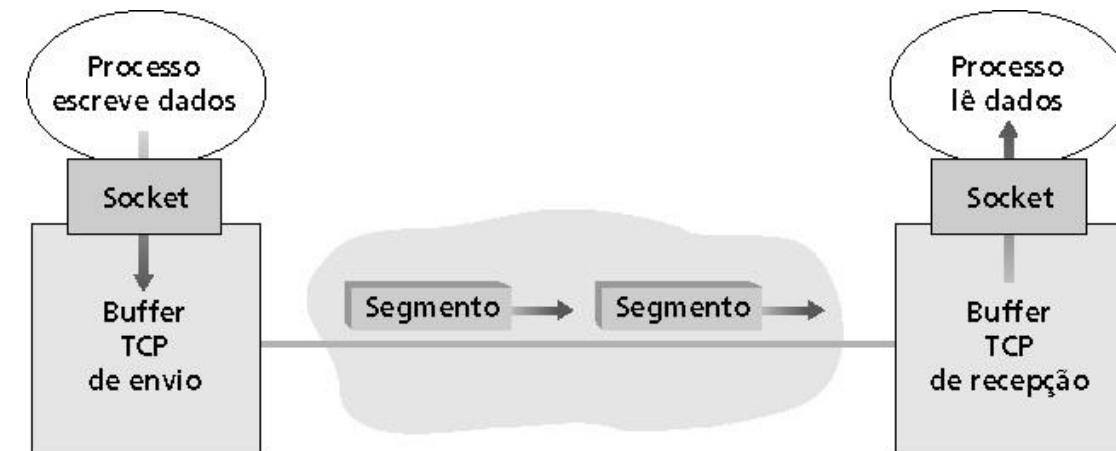
Multiplexação/Demultiplexação



TCP: NÚMERO DE SEQUÊNCIA

Tratamento do Tamanho dos Segmentos

- TCP deve manipular a diversidade de tamanhos das mensagens geradas pelas aplicações
- Blocos de dados muito grandes
 - TCP deve fragmentá-los em unidades menores (segmentos) de modo a que os protocolos de nível inferior possam tratá-los
- Blocos de dados pequenos
 - TCP “bufferiza” os segmentos para conduzi-los “juntos” num mesmo segmento TCP
 - Redução do “overhead” de transmissão



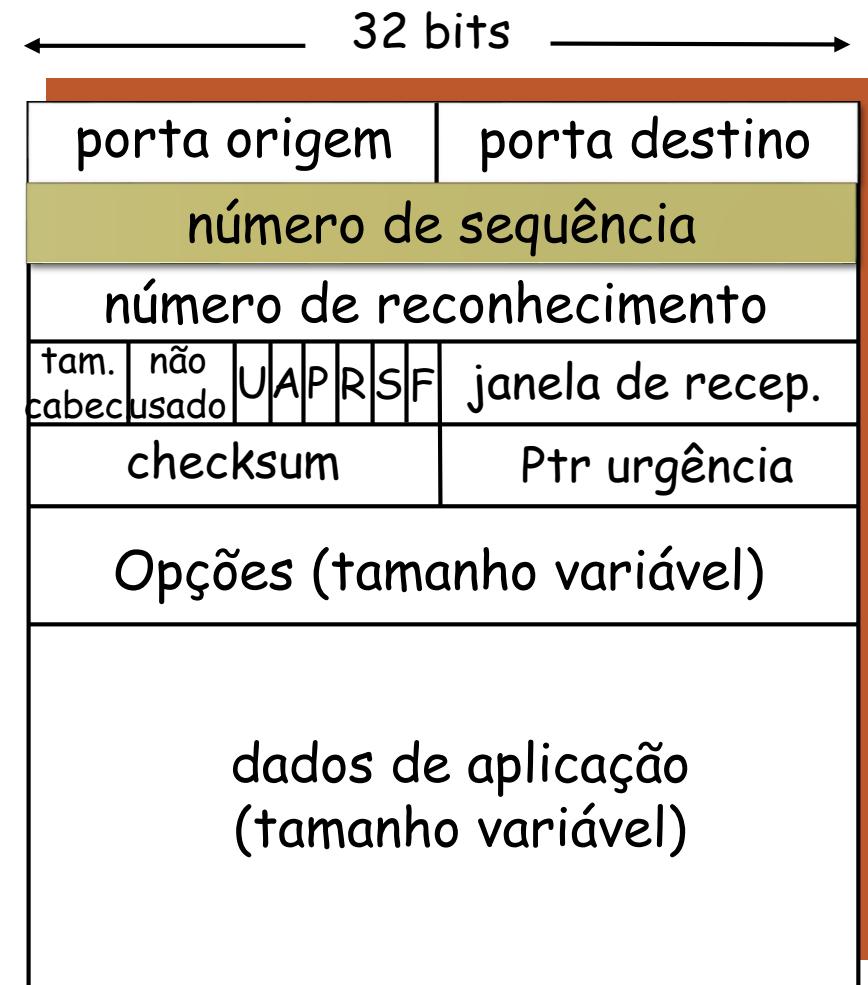
TCP: NÚMERO DE SEQUÊNCIA

Numeração dos segmentos

- Conceitualmente cada byte da mensagem é associado a um número de sequência
- Número de sequência do primeiro byte dos dados contidos em um segmento é transmitido junto com o segmento e é denominado número de sequência do segmento

Sequenciamento

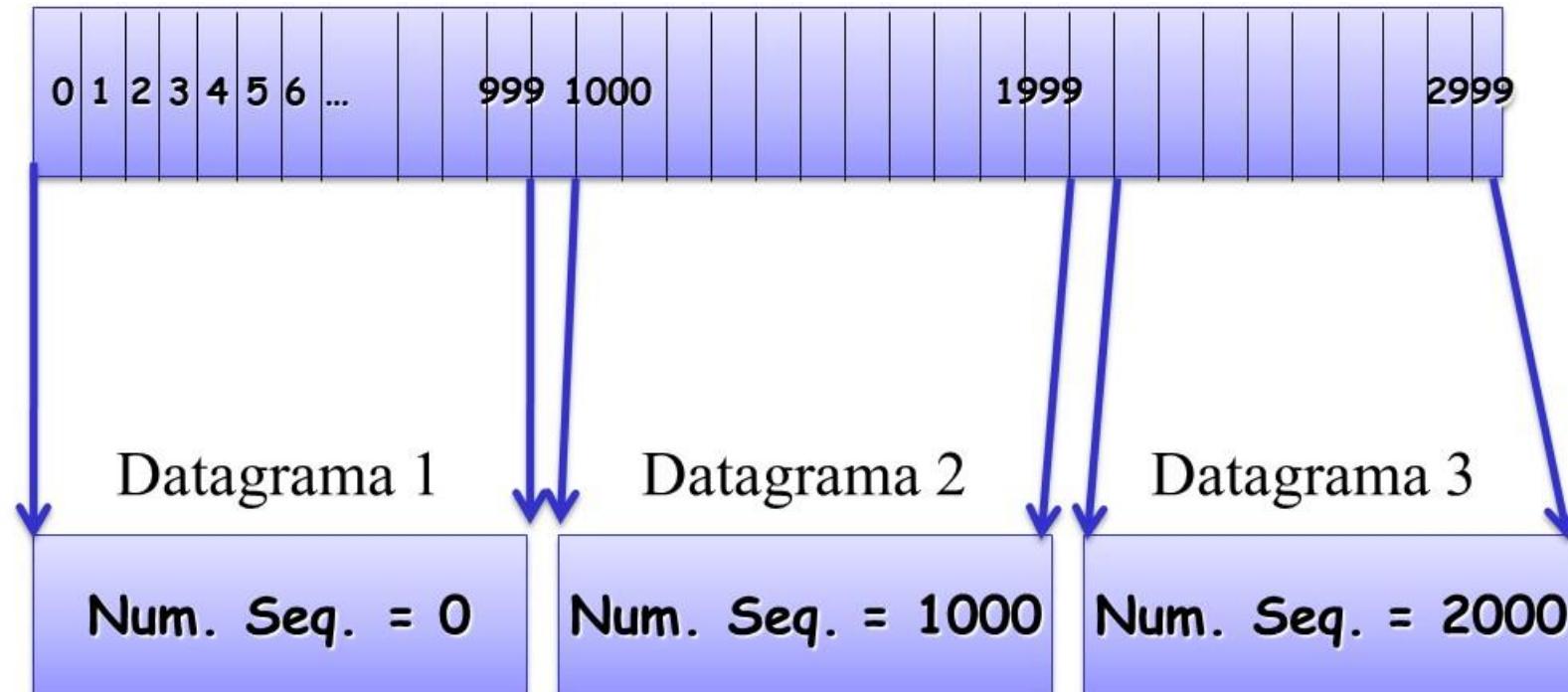
- Número de sequência é usado para ordenar os segmentos que porventura tenham sido recebidos fora de ordem e para eliminar segmentos duplicados



PROTOCOLO TCP

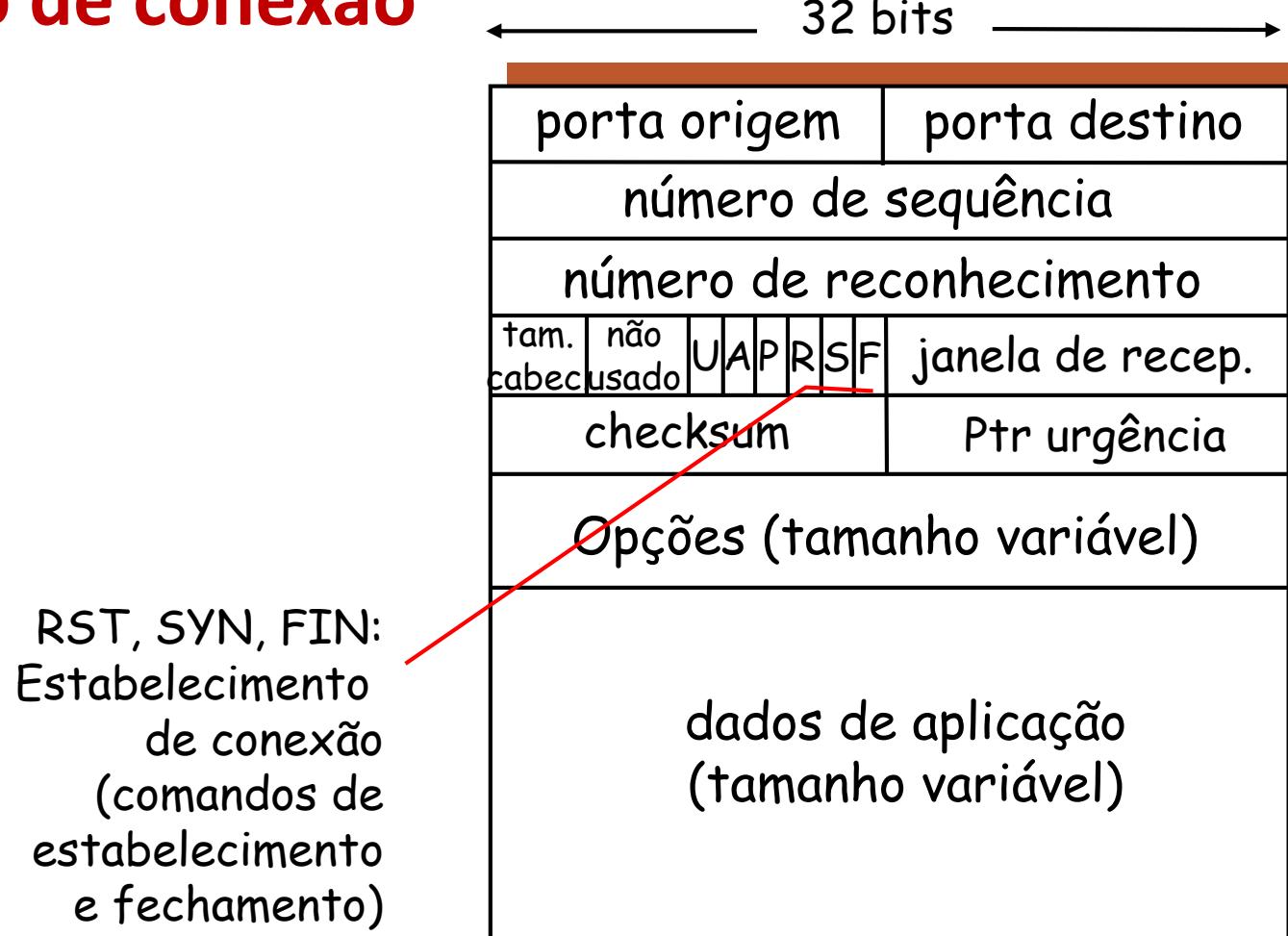
Sequenciamento

- Exemplo:
 - Segmento de 3000 bytes fragmentado em 3 datagramas de 1000 bytes (MSS)



ESTRUTURA DO SEGMENTO TCP

Estabelecimento de conexão



GERENCIAMENTO DA CONEXÃO TCP

Emissor e Receptor TCP estabelecem uma conexão antes da troca de segmentos

- Servidor: prepara socket para receber conexão e aguarda conexão

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    s.bind((HOST, PORT)) # Liga o socket ao endereço e a porta do servidor  
    s.listen() # Inicia escuta do socket  
    conn, addr = s.accept() # bloqueia aguardando pedidos de conexão
```

- Cliente: Inicializador da conexão

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    s.connect((HOST, PORT)) # estabelece a conexão
```

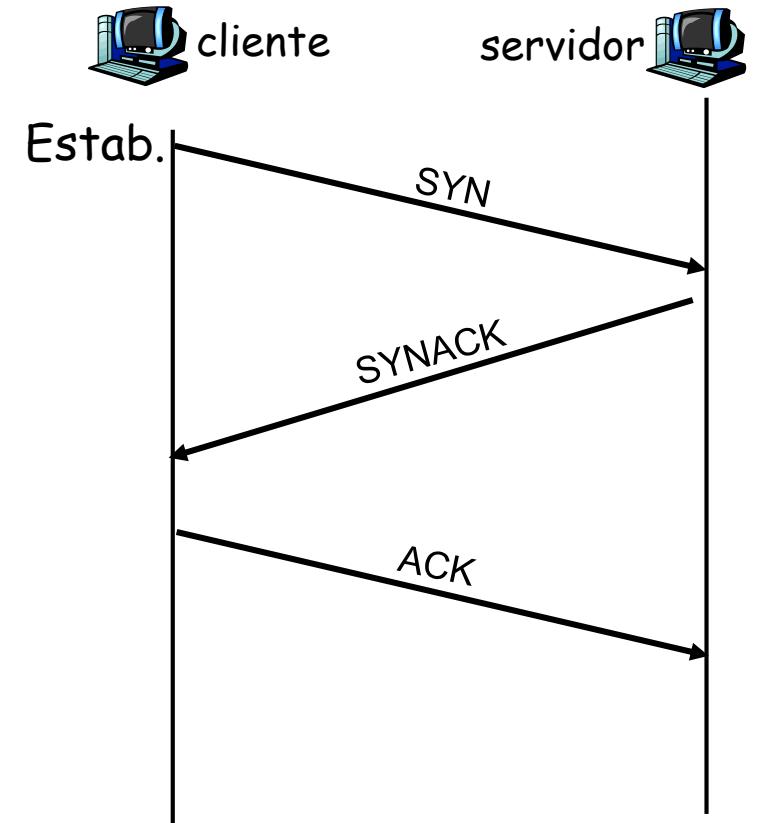
Ações após a conexão estabelecida

- Inicializa variáveis TCP:
- Números de sequência inicial
- Buffers, informações de controle de fluxo (e.g. RcvWindow)
- Etc.

GERENCIAMENTO DA CONEXÃO TCP

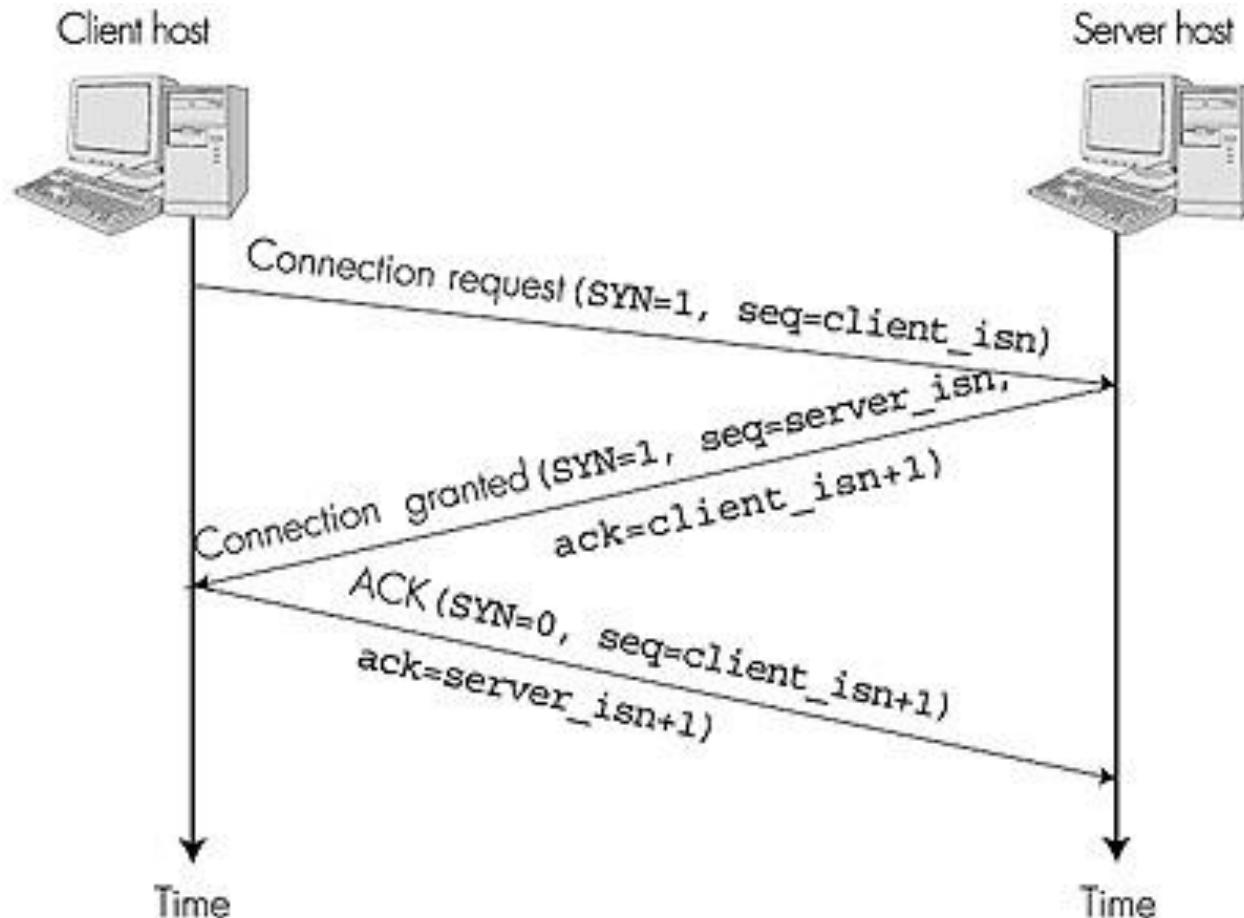
Algoritmo three-way handshake

- **Passo 1:** cliente envia segmento de controle TCP SYN (flag SYN setado a 1) para o servidor
 - Especifica o número inicial de sequência (c_nmseq)
- **Passo 2:** servidor recebe SYN, responde com o segmento de controle SYNACK
 - Aloca buffers e inicia variáveis da conexão
 - confirma SYN recebido
 - SYN bit é setado a 1
 - Campo ack do segmento TCP é setado com $c_nmseq+1$
 - Campo número de sequência é setado com o número de sequência inicial do servidor s_nmseq
- **Passo 3:** Cliente recebe SYN ACK
 - Cliente aloca buffers e variáveis para a conexão
 - Cliente envia ao servidor outro segmento confirmando o SYNACK
 - O cliente coloca o valor $s_nmseq+1$ no campo ACK do segmento TCP e o bit SYN é setado a 0



GERENCIAMENTO DA CONEXÃO TCP

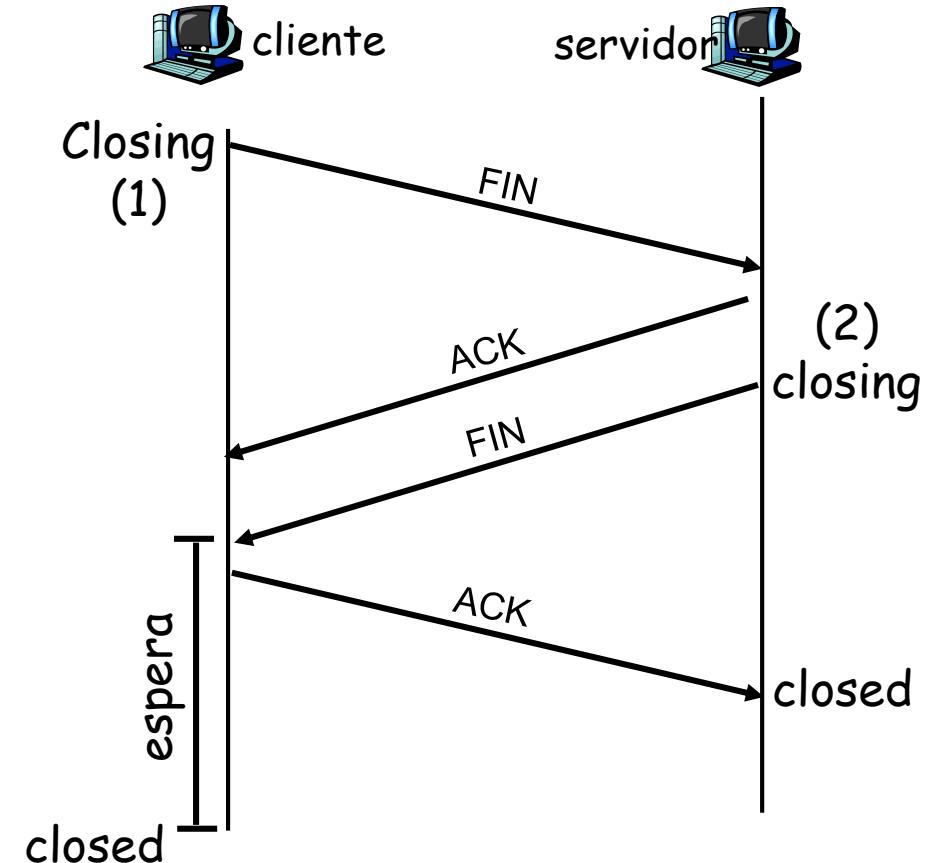
Conexão a três vias....



GERENCIAMENTO DA CONEXÃO (CONT.)

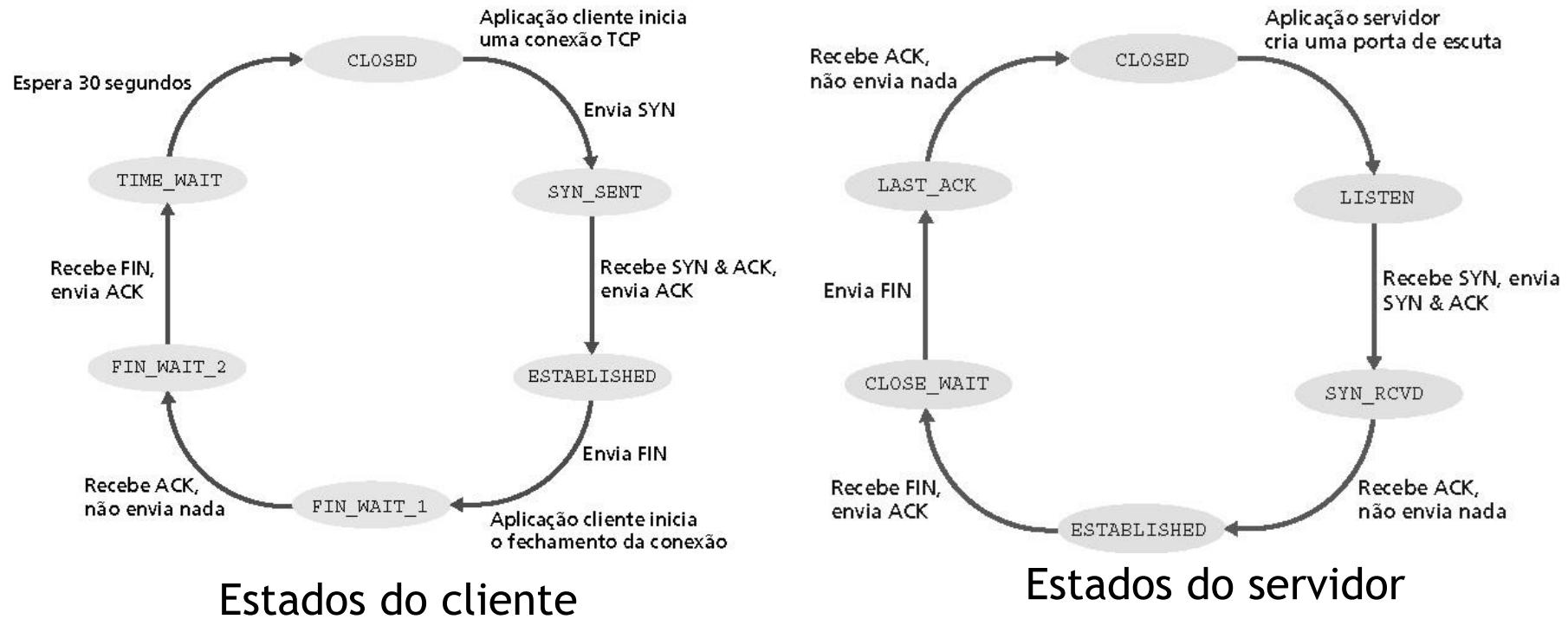
Fechando uma conexão:

- Cliente fecha socket: clientSocket.close();
- **Passo 1**: cliente envia segmento de controle TCP FIN para o servidor
- **Passo 2**: servidor recebe FIN, responde com ACK e envia FIN.
- **Passo 3**: cliente recebe FIN, responde com ACK.
 - Inicia “espera”
- **Passo 4**: receptor recebe ACK. Conexão fechada.



GERENCIAMENTO DA CONEXÃO (CONT.)

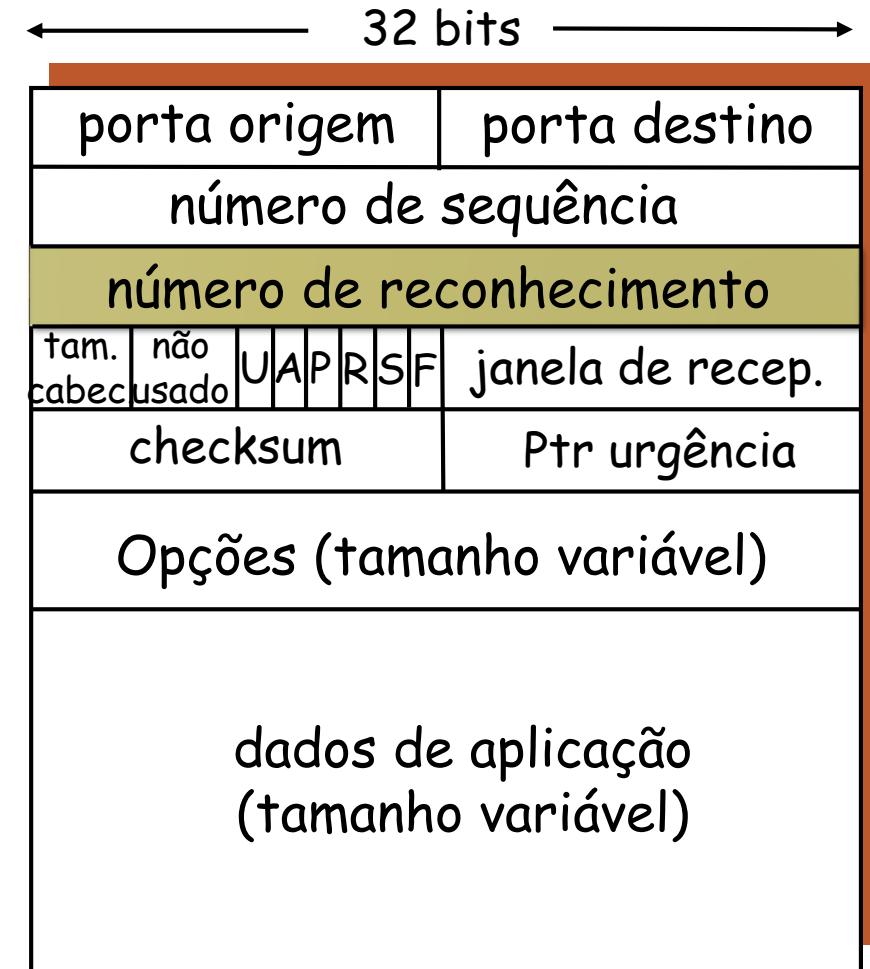
Estados da conexão TCP



ESTRUTURA DO SEGMENTO TCP

Reconhecimento (controle de erro)

- Número do próximo byte aguardado pelo receptor
 - Consiste do número de sequência do próximo byte que a entidade TCP receptora espera receber do TCP emissor
- Exemplo
 - Se número de reconhecimento X for transmitido no ACK
 - Indica que a entidade TCP receptora recebeu corretamente os bytes com número de sequência menores que X
 - Ela espera receber o byte X na próxima mensagem
 - Segmentos carregam de carona (piggybacking) um reconhecimento



TCP: RECONHECIMENTO

Número de sequência:

- Número do primeiro byte da string

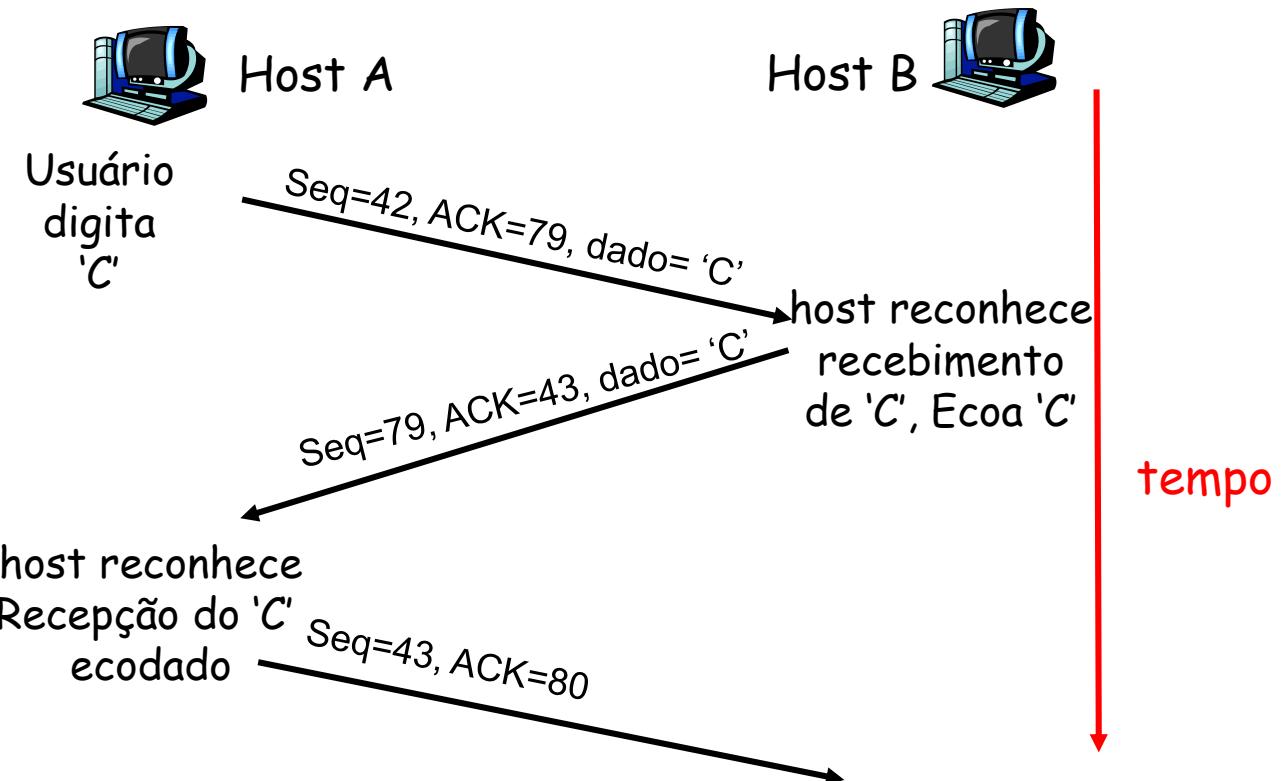
Reconhecimento:

- Número de sequência do próximo byte esperado

Sequenciamento:

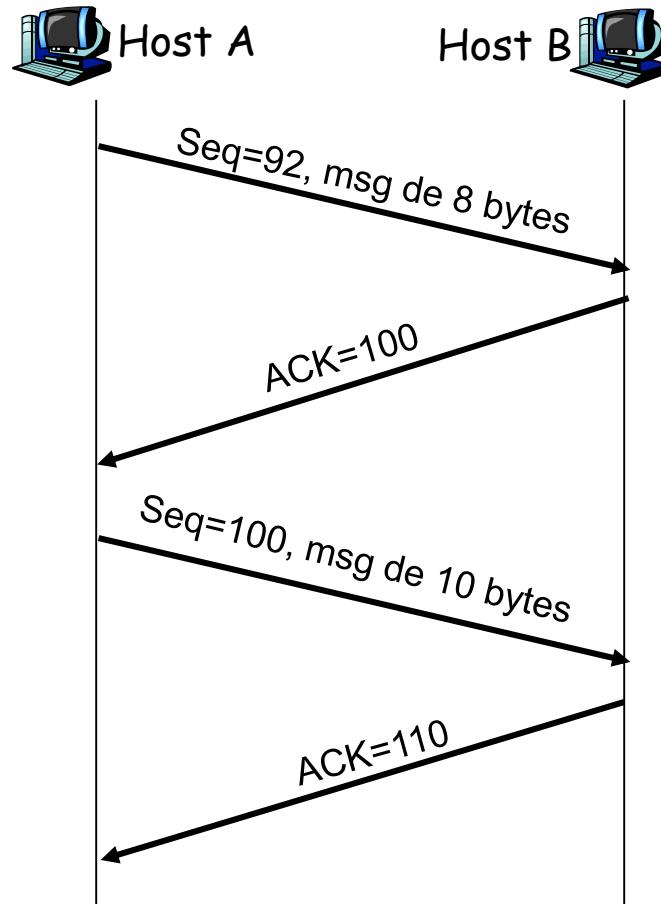
- especificação do TCP não define
- deixado ao implementador

Cenário telnet simples



TCP: RECONHECIMENTO

Outro cenário



PROTOCOLO TCP

Retransmissões

- Quando uma entidade TCP transmite um segmento
 - Ela coloca uma cópia do segmento em uma fila de retransmissão e dispara um temporizador
 - Caso o reconhecimento do segmento é recebido
 - o segmento é retirado desta fila
 - Caso o reconhecimento não ocorra antes do temporizador expirar
 - Segmento é retransmitido

TCP: TRANSFERÊNCIA DE DADOS CONFIÁVEL

Evento: dados recebidos
da aplicação acima

Cria, envia segmento com seq=y
Lança temporização

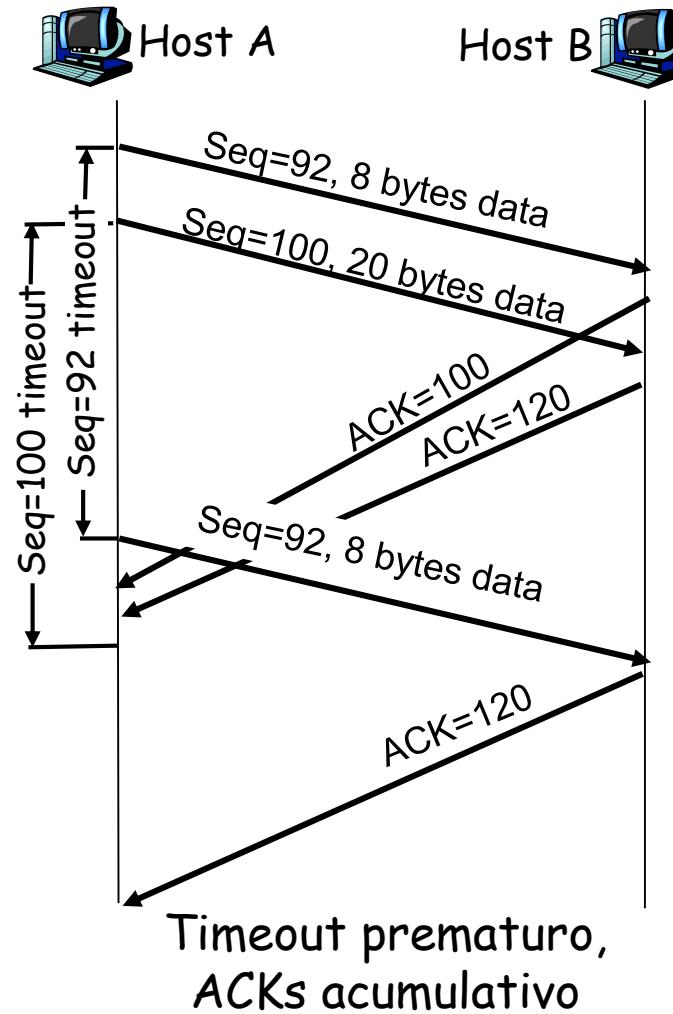
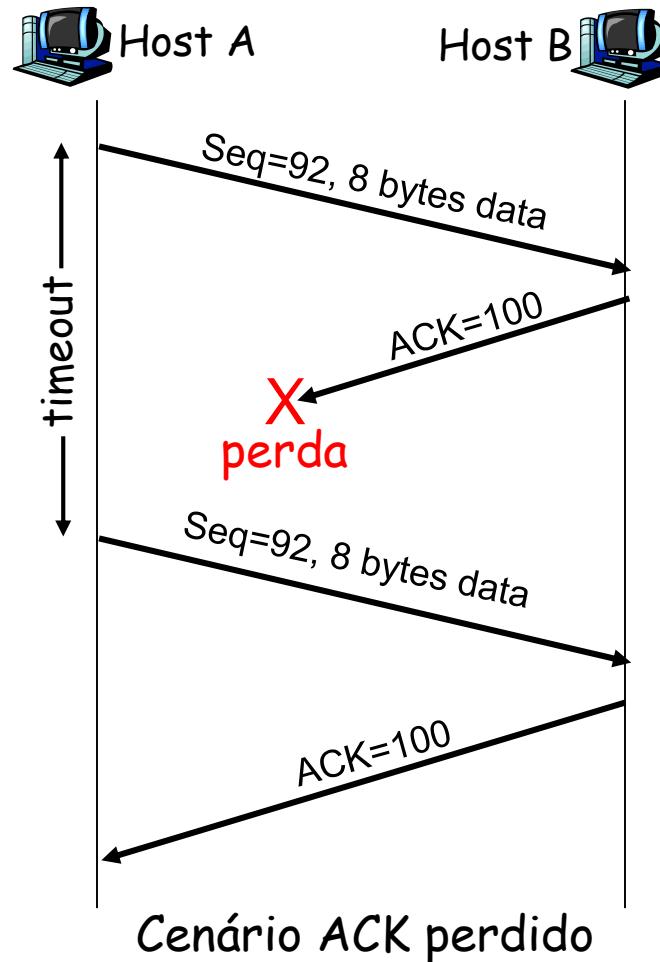


transmissor simplificado, assumindo
que não há controle de fluxo nem de
congestionamento

evento: temporização esgotada
para segmento com seq = y
retransmite segmento

evento: ACK recebido,
com número de ACK > y
processamento do ACK

TCP: CENÁRIOS DE RETRANSMISSÃO



NÚMERO DE SEQUÊNCIA E DE RECONHECIMENTO

Números iniciais são escolhidos aleatoriamente no estabelecimento da conexão

- De maneira a evitar quebras de criptografia
- Feito para minimizar a possibilidade que um segmento inválido seja confundido como válido
 - Que pertencente a uma conexão já terminada e chegue no receptor após estabelecida uma nova conexão entre os mesmos receptor e transmissor na mesma porta

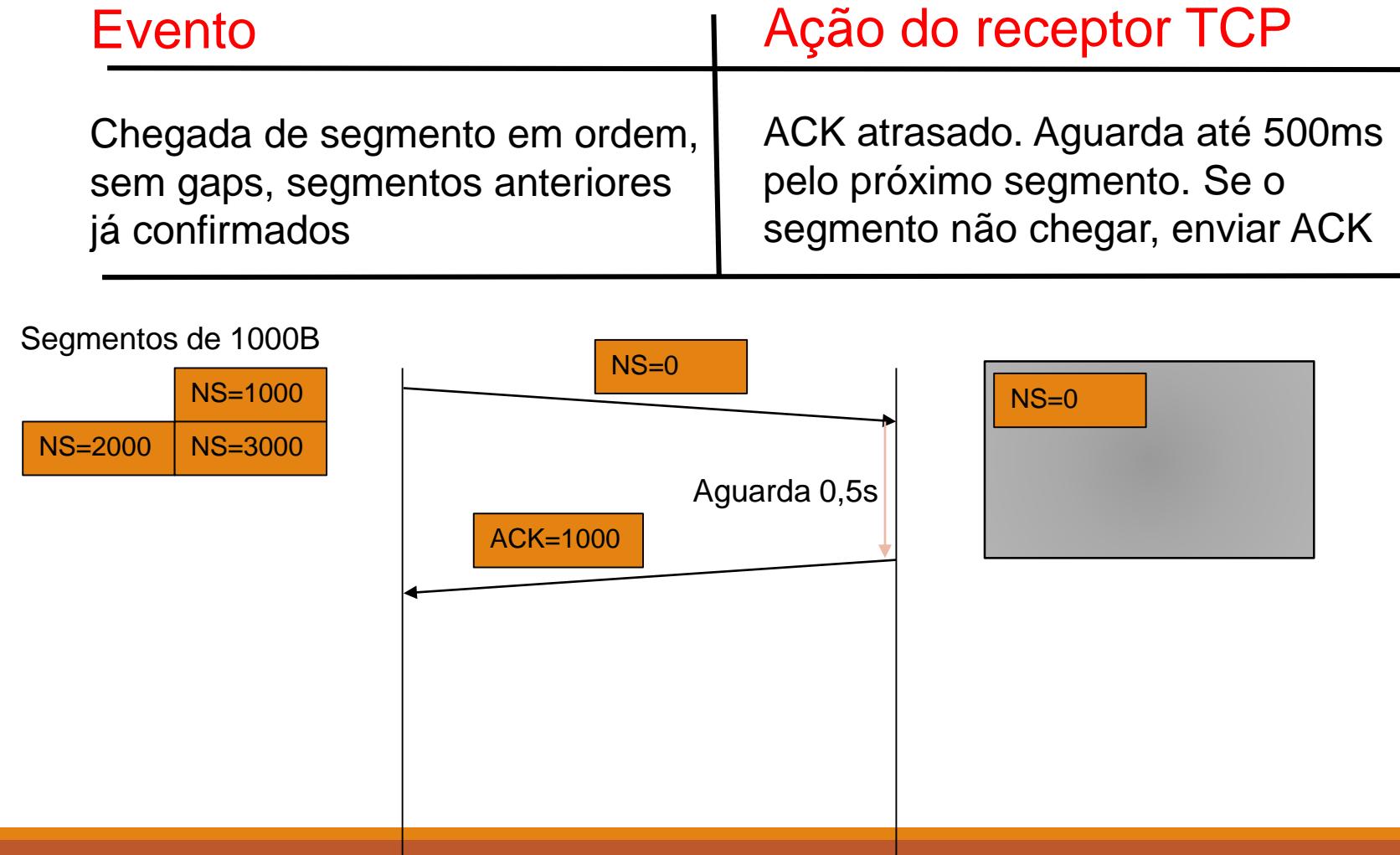
TCP GERAÇÃO DO RECONHECIMENTO [RFC 1122, RFC 2581]

Evento	Ação do receptor TCP
Chegada de segmento em ordem, sem gaps, segmentos anteriores já confirmados	ACK atrasado. Aguarda até 500ms pelo próximo segmento. Se o segmento não chegar, enviar ACK
Chegada de segmento em ordem, sem gaps, existe ACK atrasado	Enviar imediatamente um único ACK acumulativo
Chegada do segmento fora de ordem maior que o número de seq esperado (gap detectado)	Envia ACK duplicado, indicando o número de seq. do próximo byte esperado
Chegada de um segmento que preenche parcialmente ou complementamente um gap	ACK imediato se o segmento inicia na parte inicial mais baixa do gap

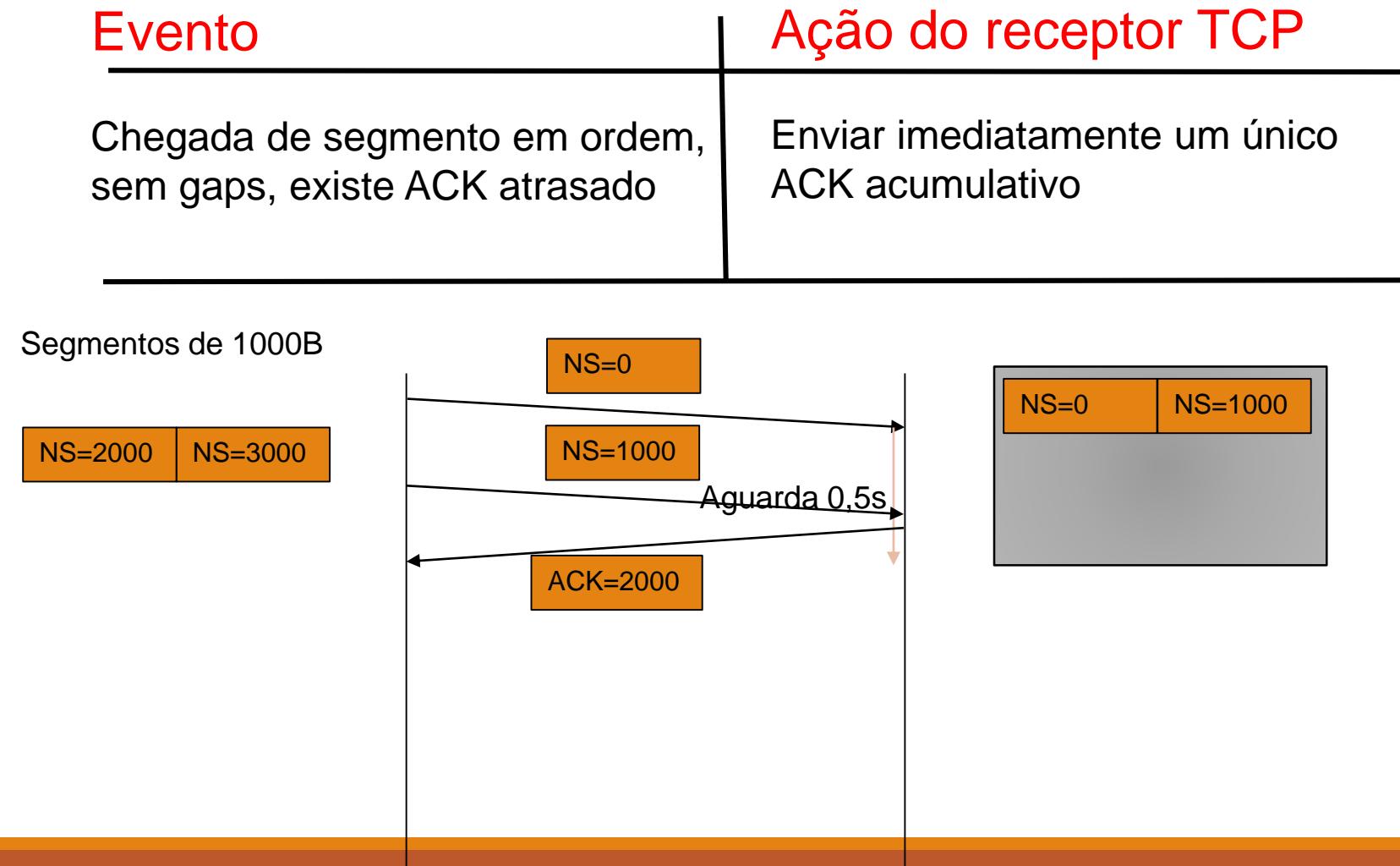
TCP GERAÇÃO DO RECONHECIMENTO [RFC 1122, RFC 2581]

Evento	Ação do receptor TCP				
Chegada de segmento em ordem, sem gaps, segmentos anteriores já confirmados	ACK atrasado. Aguarda até 500ms pelo próximo segmento. Se o segmento não chegar, enviar ACK				
Segmentos de 1000B <table border="1" data-bbox="404 779 770 904"><tr><td>NS=0</td><td>NS=1000</td></tr><tr><td>NS=2000</td><td>NS=3000</td></tr></table>	NS=0	NS=1000	NS=2000	NS=3000	
NS=0	NS=1000				
NS=2000	NS=3000				

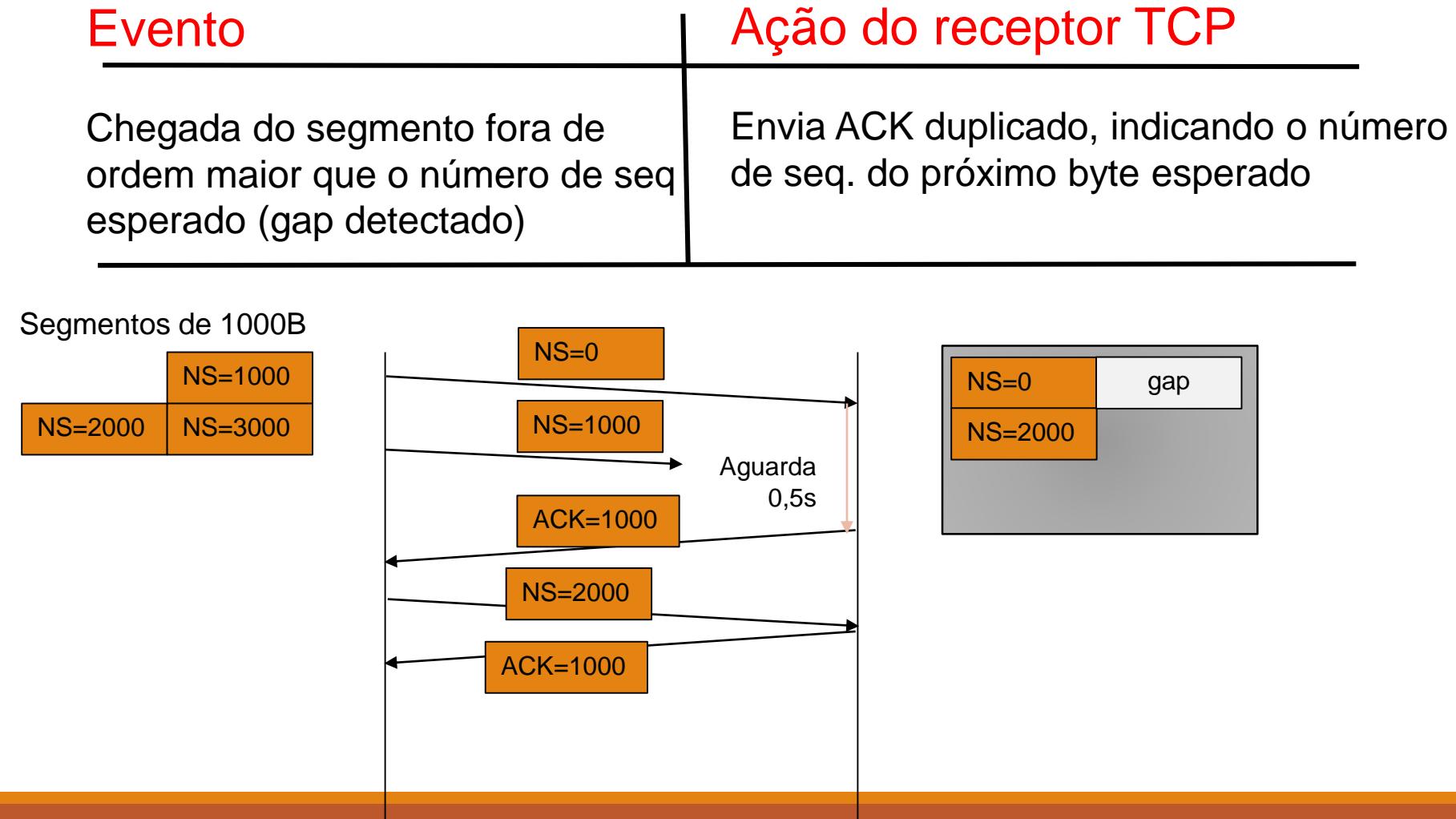
TCP GERAÇÃO DO RECONHECIMENTO [RFC 1122, RFC 2581]



TCP GERAÇÃO DO RECONHECIMENTO [RFC 1122, RFC 2581]



TCP GERAÇÃO DO RECONHECIMENTO [RFC 1122, RFC 2581]



TCP GERAÇÃO DO RECONHECIMENTO [RFC 1122, RFC 2581]

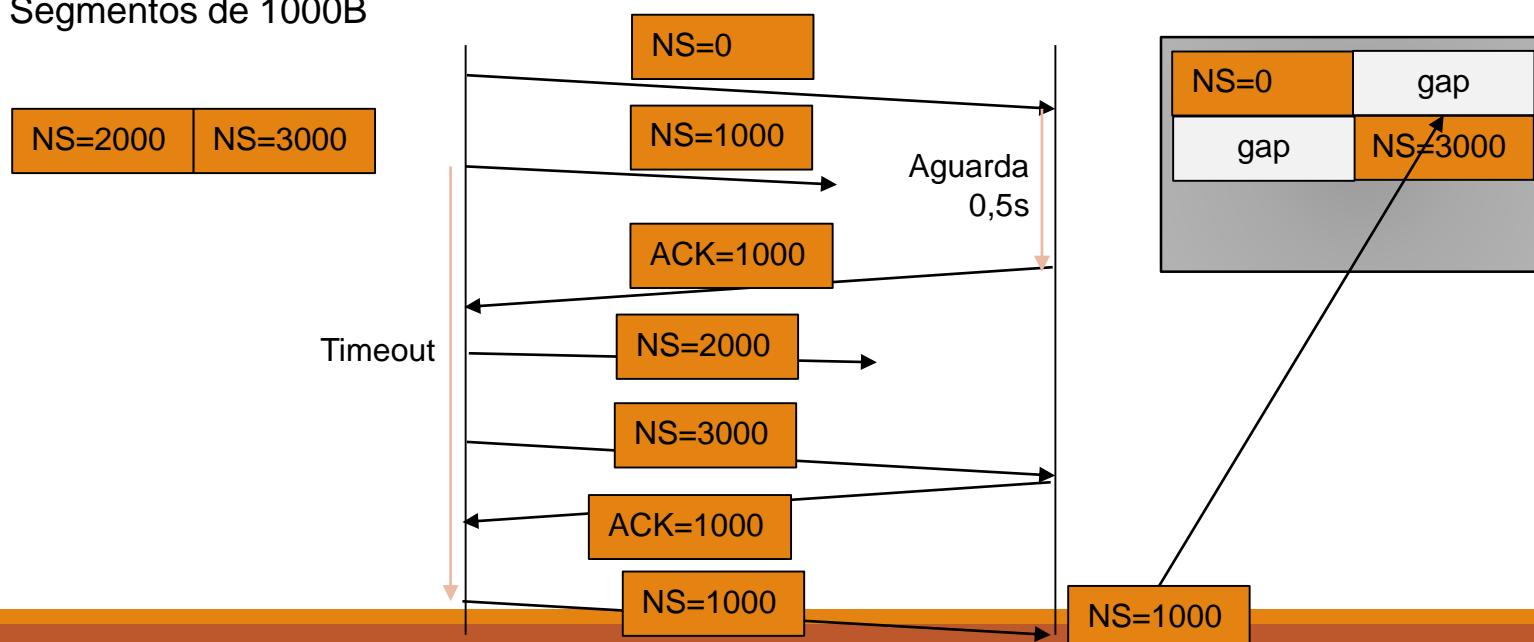
Evento

Chegada de um segmento que preenche parcialmente ou complementamente um gap

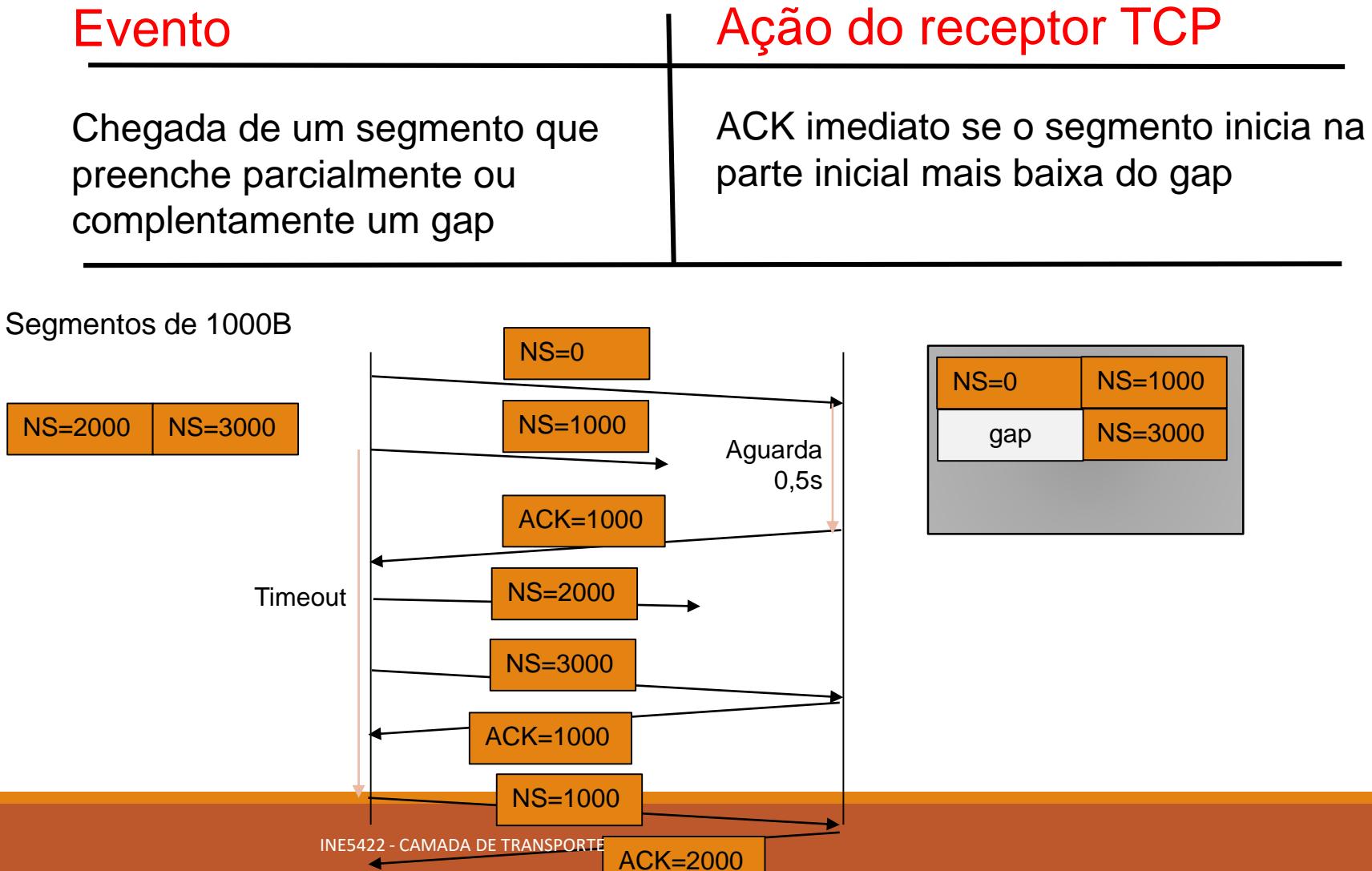
Ação do receptor TCP

ACK imediato se o segmento inicia na parte inicial mais baixa do gap

Segmentos de 1000B



TCP GERAÇÃO DO RECONHECIMENTO [RFC 1122, RFC 2581]



CAP 5. CAMADA DE TRANSPORTE

AULA 6: PROTOCOLO TCP(CONTINUAÇÃO)

INE5422 REDES DE COMPUTADORES II

PROF. ROBERTO WILLRICH (INE/UFSC)

ROBERTO.WILLRICH@UFSC.BR

[HTTPS://MOODLE.UFSC.BR](https://MOODLE.UFSC.BR)

TCP: TEMPO DE RESPOSTA (RTT) E TEMPORIZAÇÃO

Como escolher valor do temporizador TCP?

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

TCP: TEMPO DE RESPOSTA (RTT) E TEMPORIZAÇÃO

Como estimar o RTT

- *SampleRTT*: tempo medido entre a transmissão do segmento e o recebimento do ACK correspondente
- Cada segmento TCP terá seu próprio *sampleRTT*
 - Ignora retransmissões e segmentos reconhecidos de forma cumulativa

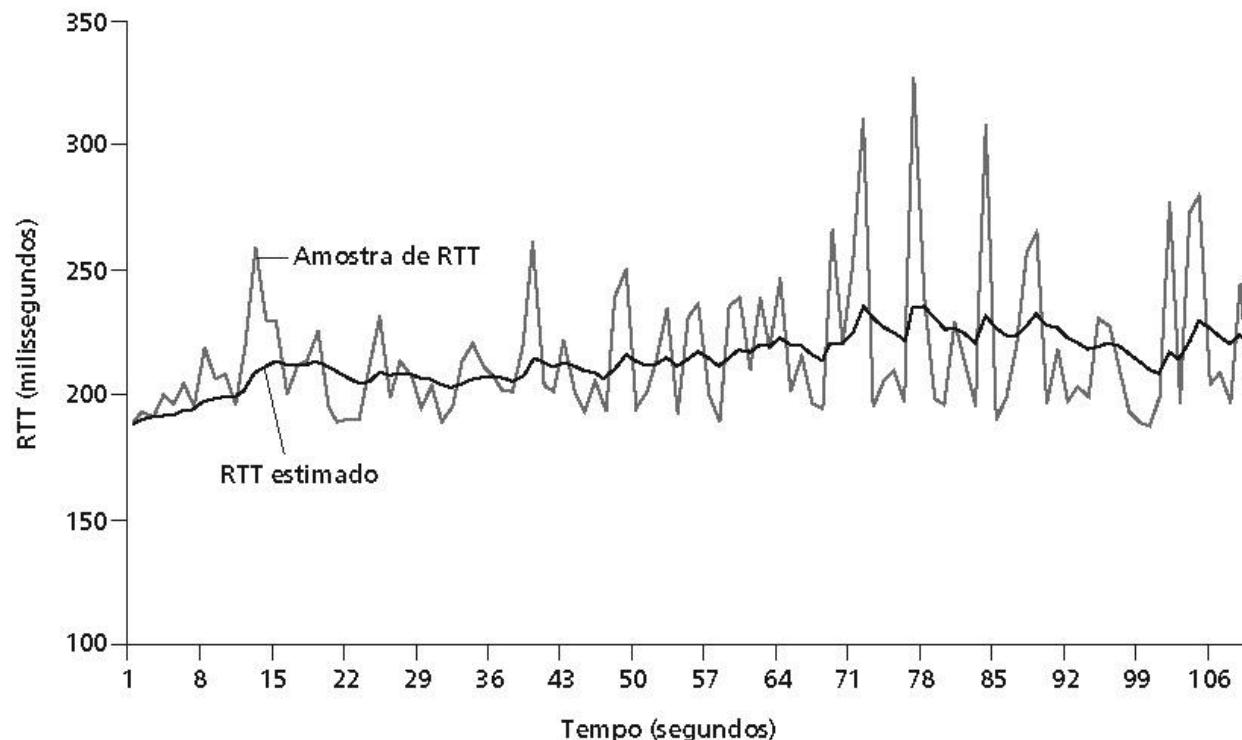
SampleRTT vai variar devido a congestionamento e sistemas finais

- TCP mantém média de *SampleRTT* (*EstimatedRTT*)

TCP: TEMPO DE RESPOSTA (RTT) E TEMPORIZAÇÃO

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

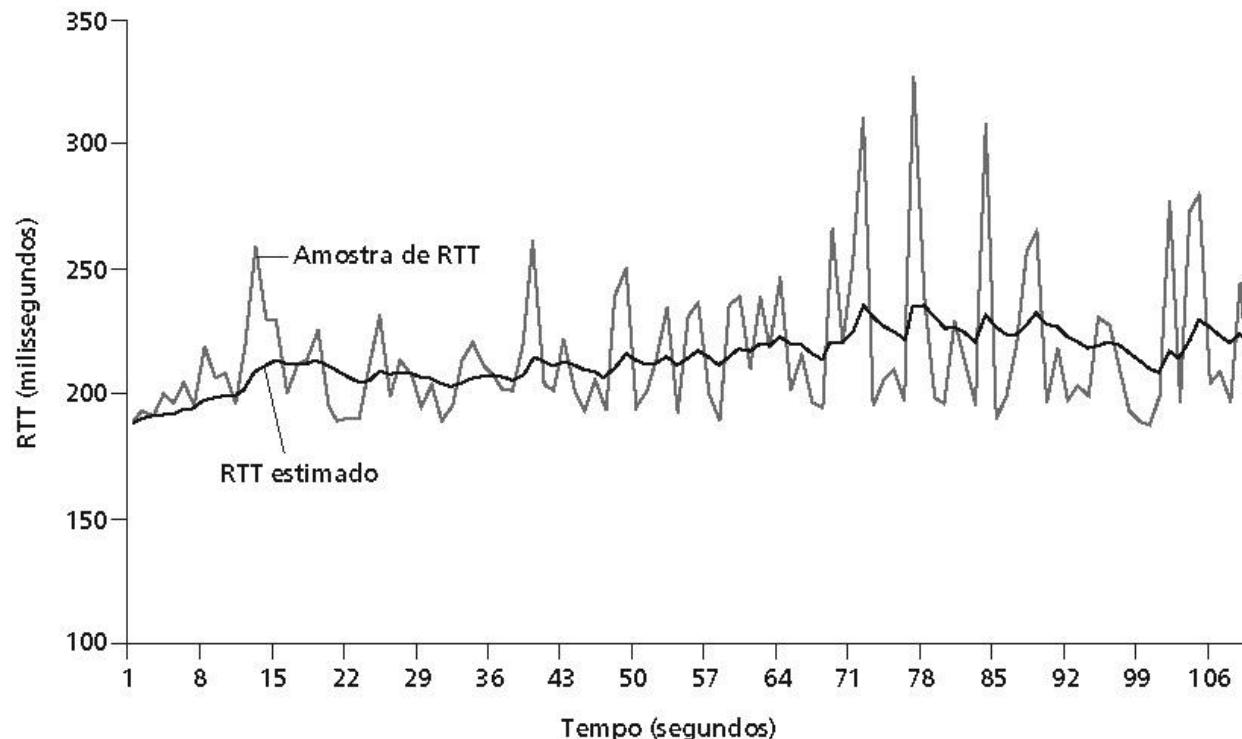
- média corrente exponencialmente ponderada (MMEP)
- valor típico de $\alpha = 0,125$ (isto é $1/8$)



TCP: TEMPO DE RESPOSTA (RTT) E TEMPORIZAÇÃO

$$\text{EstimatedRTT} = (1 - 0,125) * \text{EstimatedRTT} + 0,125 * \text{SampleRTT}$$

- RTT0 = 10ms => EstimatedRTT=10ms
- RTT1 = 12ms => EstimatedRTT= $0,975 * 10 + 0,125 * 12$ ms



TCP: TEMPO DE RESPOSTA (RTT) E TEMPORIZAÇÃO

Escolhendo o intervalo de temporização

- Intervalo de temporização = EstimatedRTT mais uma “margem de segurança”
- Se há uma variação grande em EstimatedRTT
 -> margem de segurança maior
- Desvio é uma estimativa de quando SampleRTT tipicamente se desvia de EstimatedRTT (MMPE)

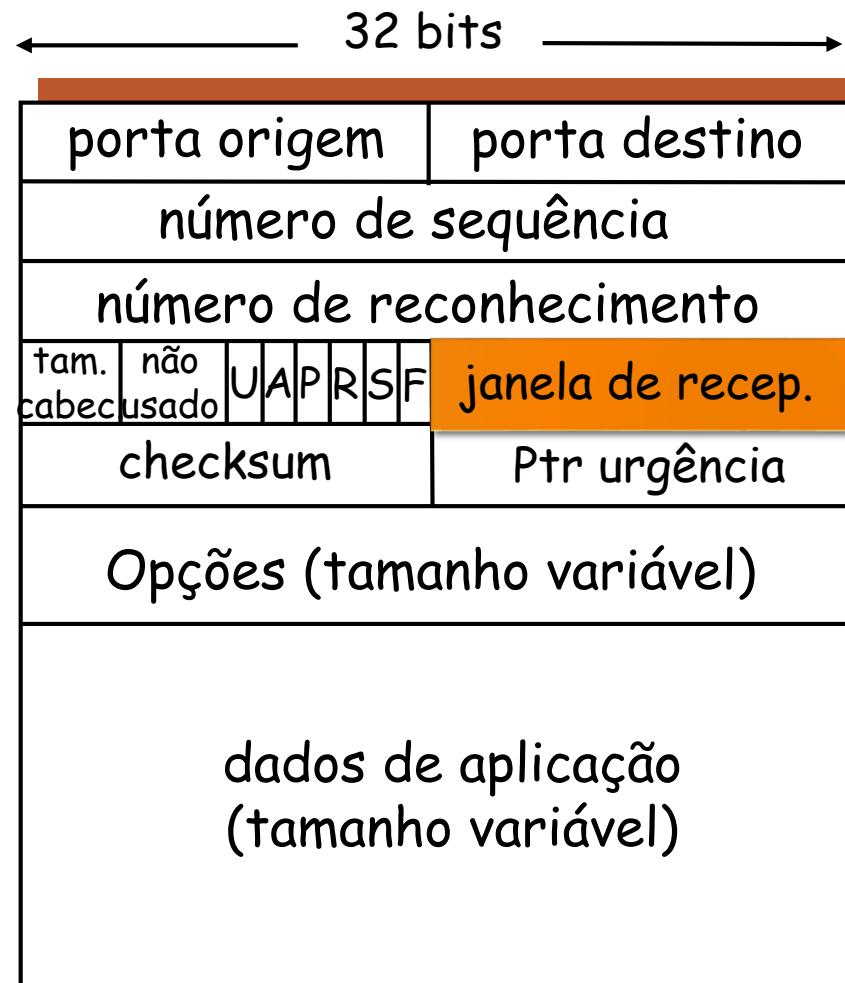
Temporização = RTT_estimado + 4*Desvio

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

$\beta = 0.25$ (valor típico)

ESTRUTURA DO SEGMENTO TCP

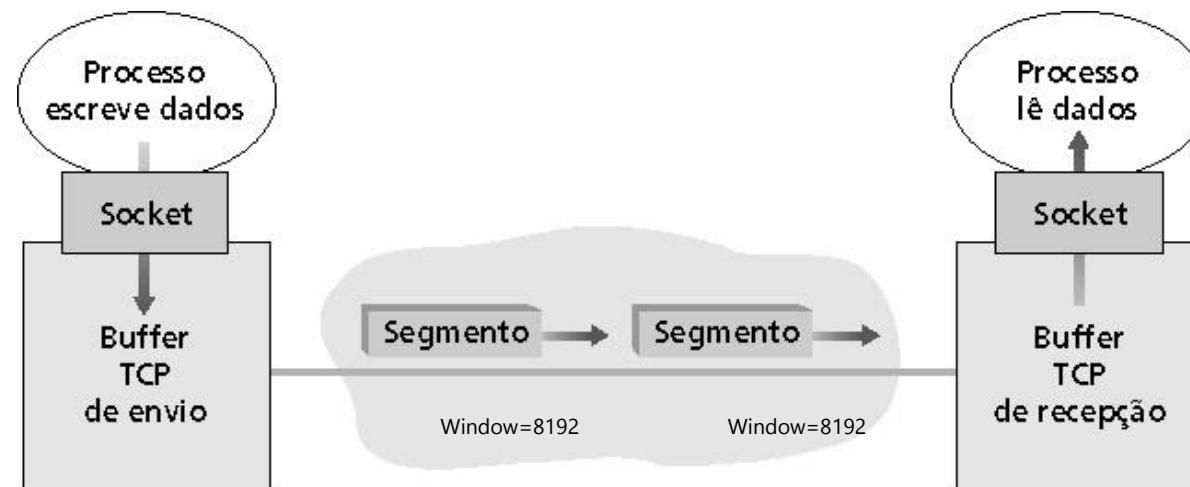
Controle de fluxo



PROTOCOLO TCP

Controle de Fluxo

- TCP provê mecanismo para que o transmissor possa determinar o volume de dados que o receptor pode receber
 - Baseia-se no campo Window do pacote: o transmissor do pacote informa o número de bytes que o transmissor tem condições de receber
 - O tamanho da janela de recepção (RcvWindow)



PROTOCOLO TCP

Controle de Fluxo

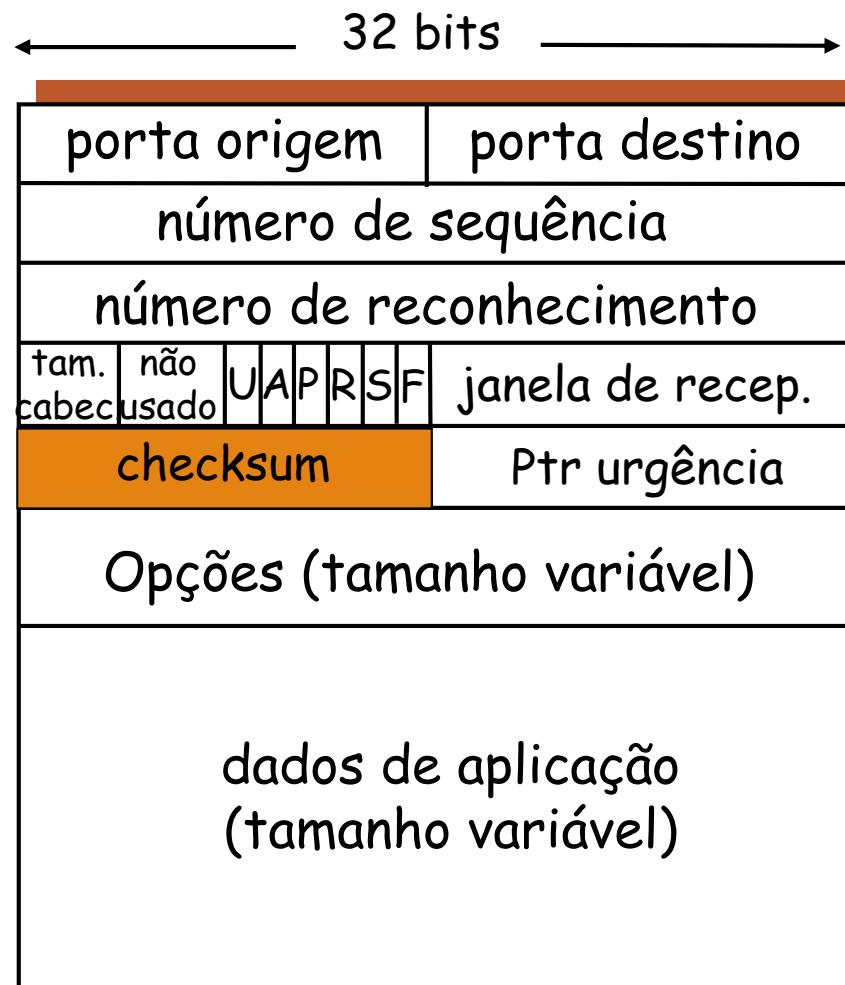
- TCP provê mecanismo para que o receptor possa determinar o volume de dados que o transmissor pode lhe enviar
 - Baseia-se no envio, junto com o reconhecimento, do número de octetos que o receptor tem condições de receber contados a partir do último octeto da cadeia de dados recebido com sucesso
 - O tamanho da janela de recepção (RcvWindow)



ESTRUTURA DO SEGMENTO TCP

Tratamento de erros

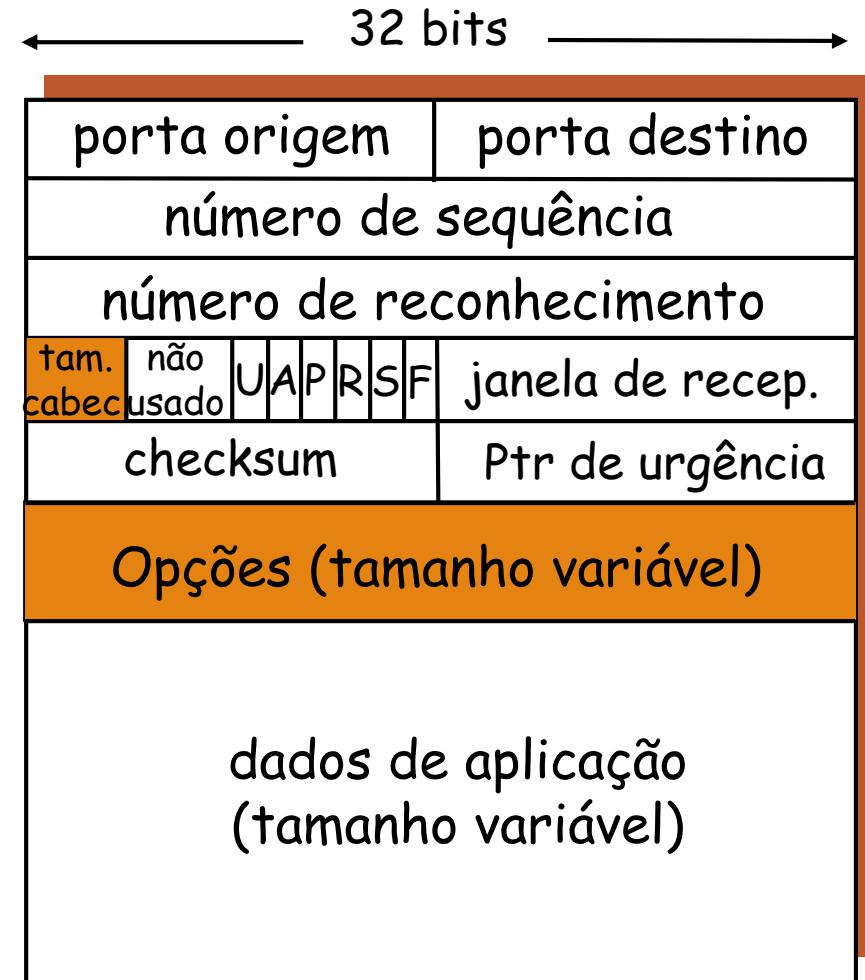
- é adicionado um checksum a cada segmento transmitidos
 - Receptor faz uma verificação e os segmentos danificados são descartados



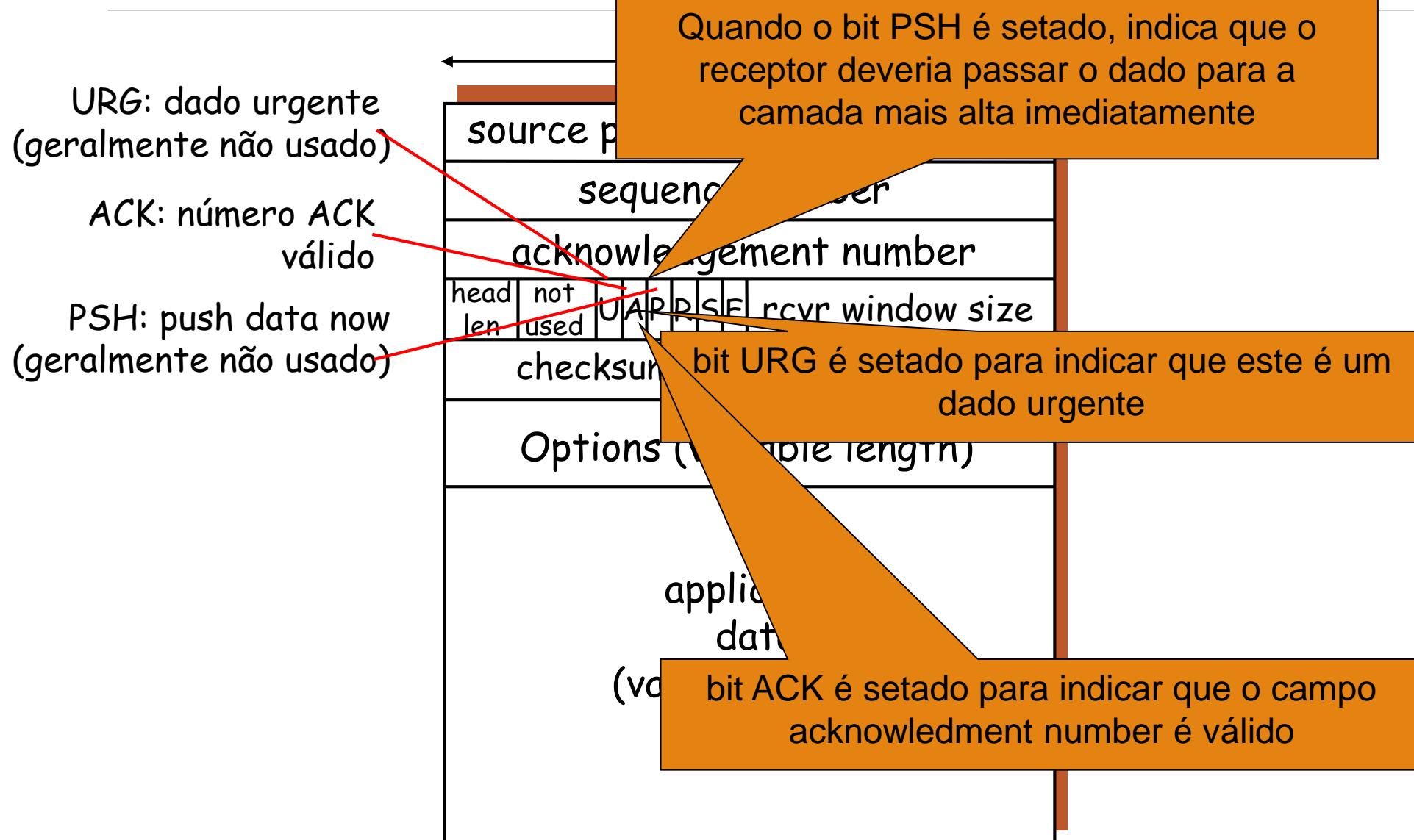
ESTRUTURA DO SEGMENTO TCP

Demais Campos

- Campo length (4 bits)
 - Especifica o tamanho do cabeçalho TCP em palavras de 32 bits
 - Cabeçalho tem tamanho variável devido ao campo TCP options
- Campo options
 - Usado quando o emissor e receptor negociam
 - um tamanho máximo de segmento (MSS)
 - fator de escalamento de janela para uso em redes de alta velocidade, para interpretação do campo Window, inicialmente teria um tamanho máximo de 65535 (por ser de 16 bits), sendo que o fator pode ser um multiplicador que permita janelas de até 1GB
 - Uma opção de timestamping também é definida
- Ponteiro de urgência
 - Aponta para o último byte de dados urgente do pacote
- Campo padding
 - Usado quando necessário para completar tamanho de segmento múltiplo de 32 bits



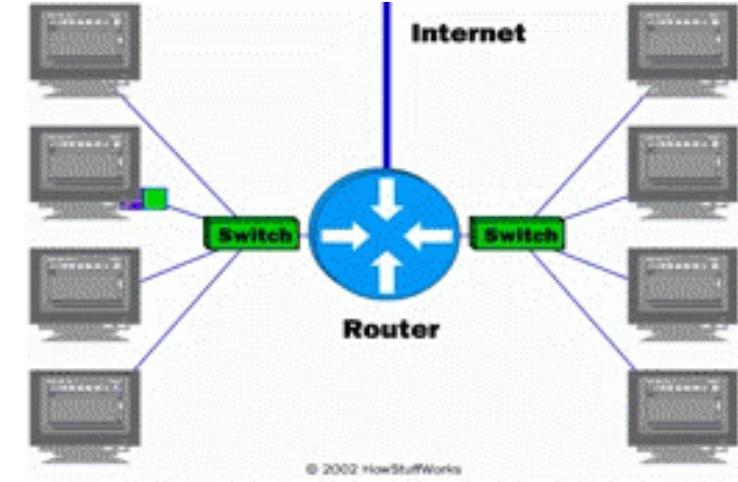
ESTRUTURA DO SEGMENTO TCP



CONTROLE DE CONGESTIONAMENTO

Congestionamento:

- Informalmente: “Excessivo número de fontes enviando grande quantidade de dados mais rápido que a rede possa manipular”
- Manifestações:
 - Pacotes perdidos (overflow dos buffers nos roteadores)
 - Grandes atrasos (enfileiramento nos buffers dos roteadores)
- Um grande problema de rede!



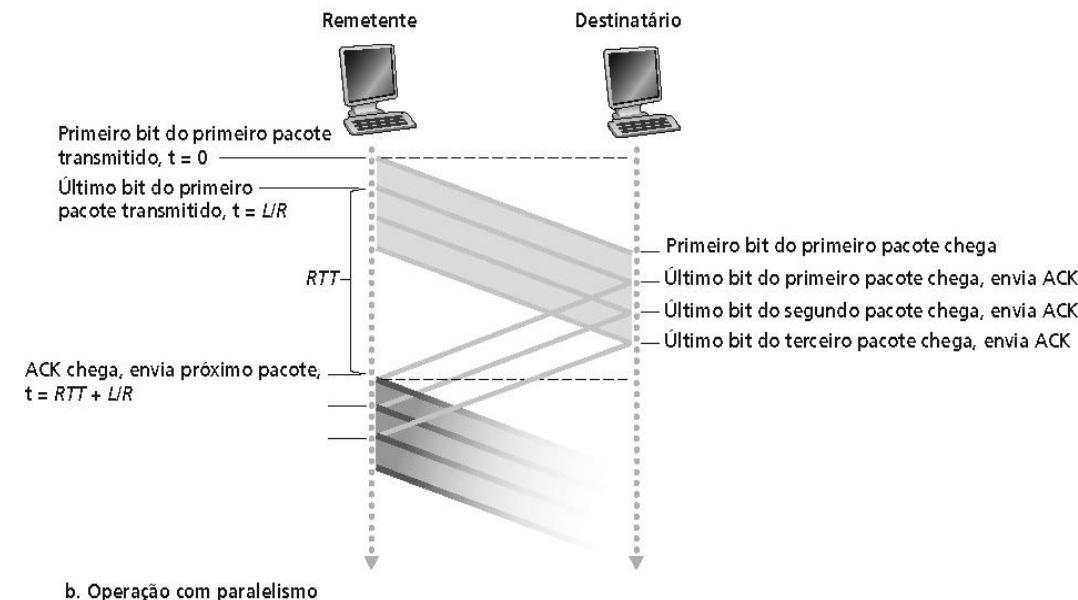
TCP: CONTROLE DE CONGESTIONAMENTO

Controle fim-a-fim (sem assistência da rede)

- Transmissor limita a transmissão:
 - $\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$
- Aproximadamente,

$$\text{Taxa} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- CongWin é dinâmico, função do nível de congestionamento da rede
- Como o transmissor detecta o congestionamento?
 - Evento de perda = timeout do temporizador ou 3 ACKs duplicados
- Transmissor TCP reduz a taxa (CongWin) após o evento de perda



CONTROLE DE CONGESTIONAMENTO TCP

Partida Lenta

- Quando a conexão começa, CongWin = 1 MSS
 - Exemplo: MSS = 500 bytes e RTT = 200 milissegundos
 - Taxa inicial = 20 kbps
- Largura de banda disponível pode ser >> MSS/RTT
 - Desejável aumentar rapidamente até a taxa respeitável
 - Quando a conexão começa, a taxa aumenta rapidamente de modo exponencial até a ocorrência do primeiro evento de perda
- Na partida lenta a banda oferecida cresce exponencialmente até chegar a um limiar ou ocorrência de perda
 - Na ocorrência da perda recomeça a partida lenta

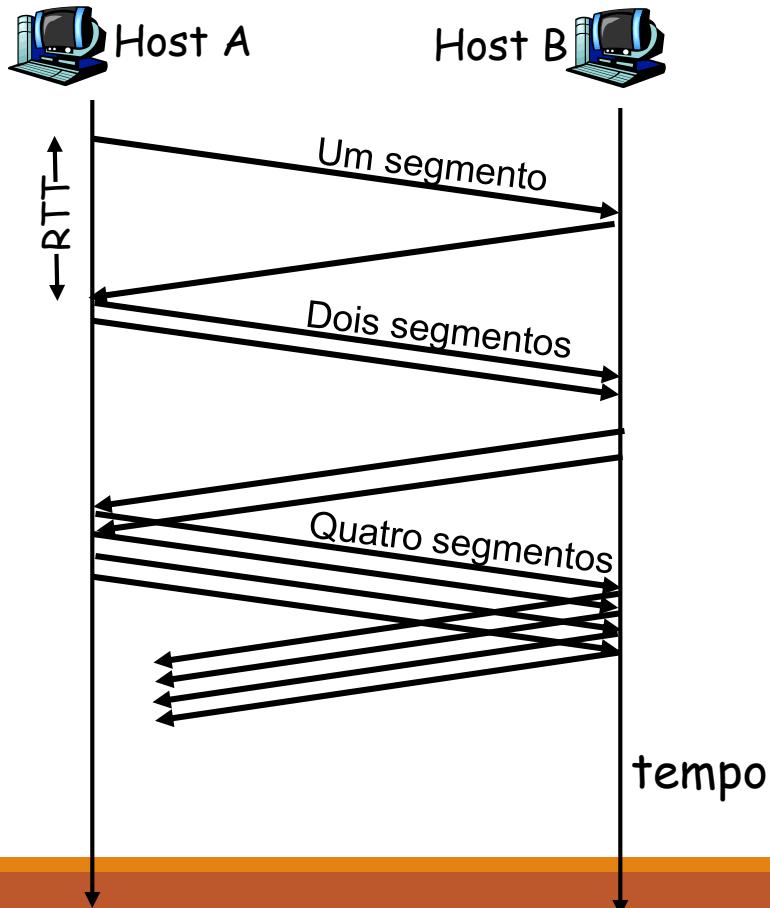
PARTIDA LENTA TCP

Incremento exponencial no tamanho da janela (não muito lenta!)

Evento de perda: timeout e/ou três ACKs duplicados

Algoritmo Partida lenta

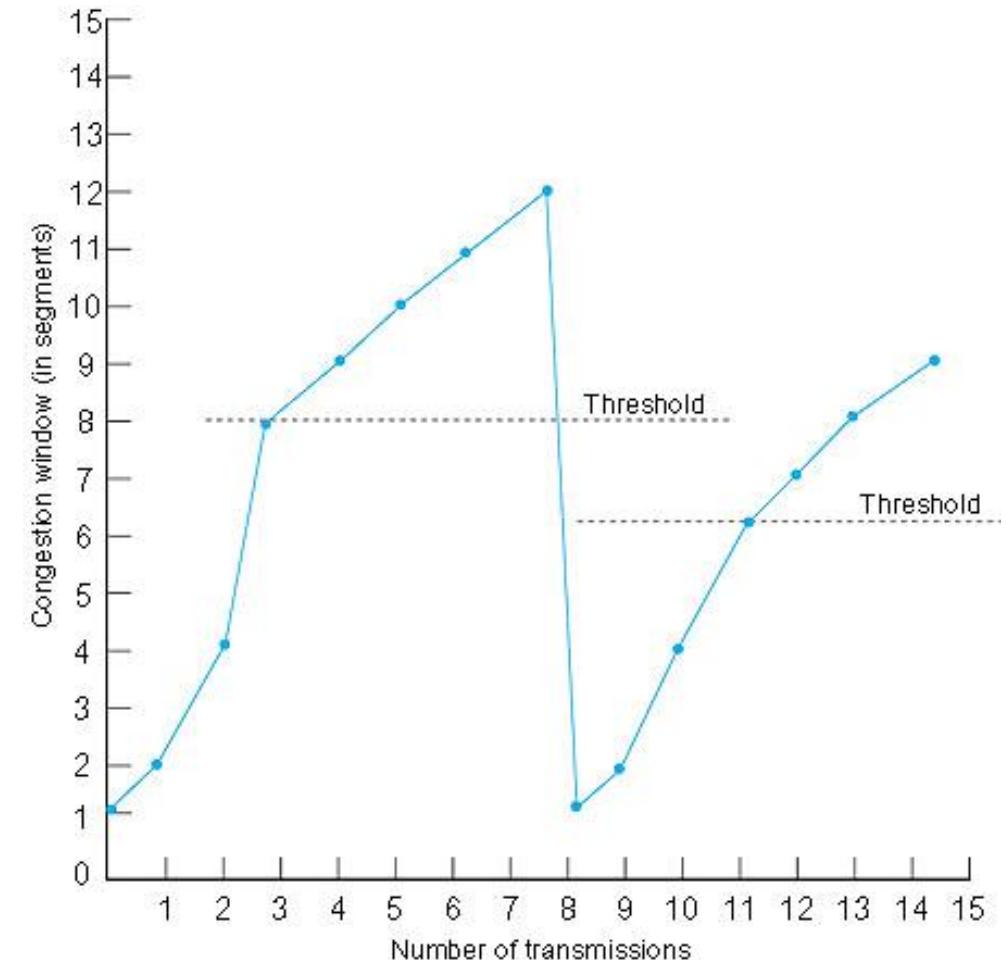
```
inicializa: Congwin = 1 MSS  
Para (cada segm com ack)  
    Congwin++  
Até (evento de perda OU  
    CongWin > threshold)
```



CONTROLE DE CONGESTIONAMENTO TCP

Fase de partida lenta

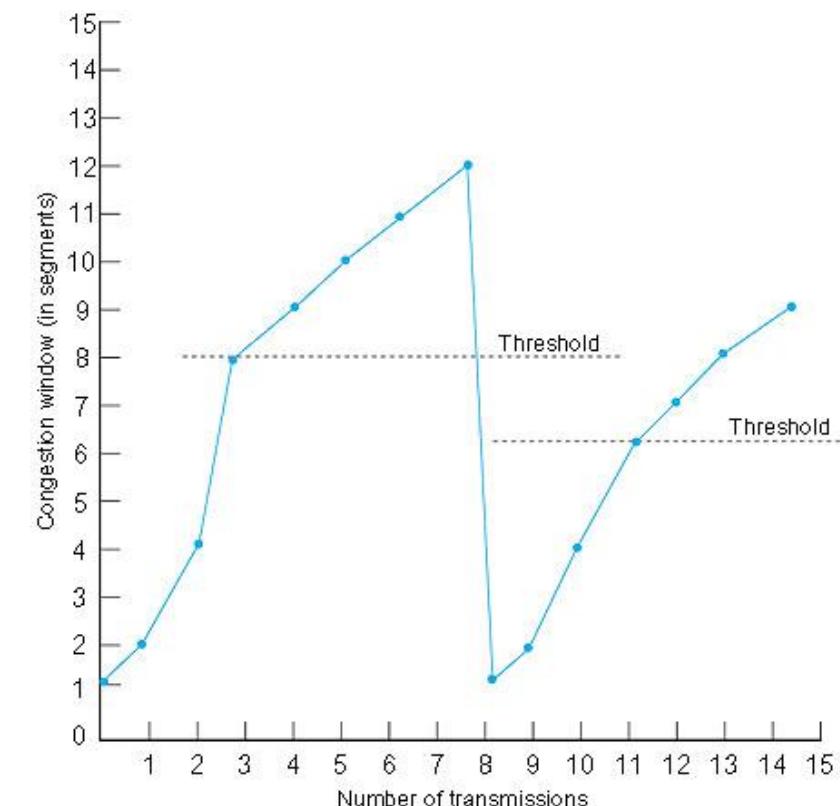
- Termina quando o tamanho da janela excede o valor do threshold
 - Uma vez que a janela de congestionamento é maior que o valor atual do threshold, a janela de congestionamento cresce linearmente (e não mais exponencialmente)
- Esta fase é chamada de prevenção do congestionamento



CONTROLE DE CONGESTIONAMENTO TCP

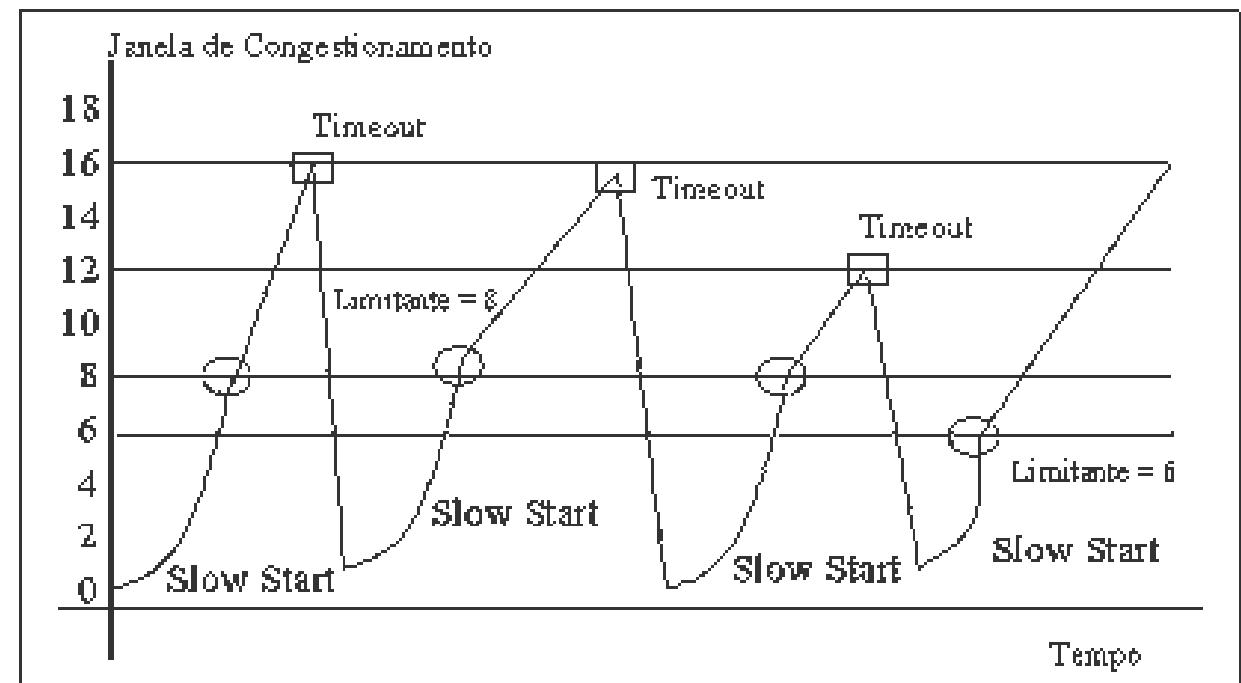
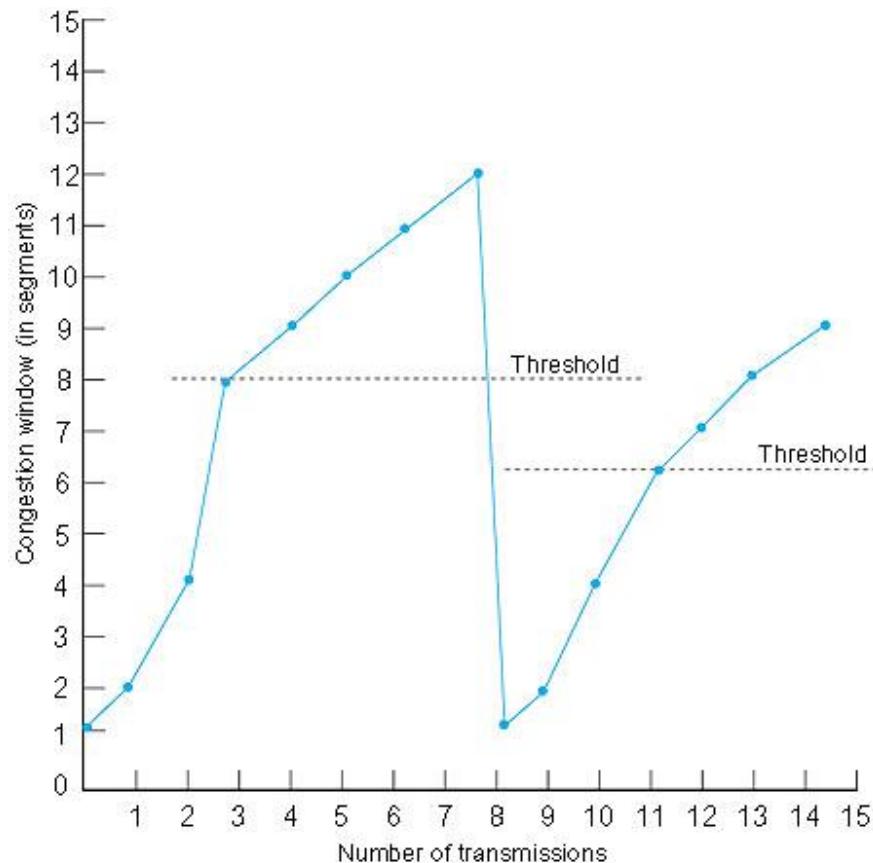
Fase de prevenção de congestionamento

- Mas o tamanho da janela não podem crescer para sempre
- Eventualmente, a taxa TCP será tal que um dos enlaces seja saturado
 - Em que perdas ocorrerão (e resultante timeout no emissor)
- Quando um timeout ocorrer
 - threshold é setado como a metade do valor da janela de congestionamento atual, e a janela de congestionamento é resetada para um MSS
 - Emissor então procede o incremento exponencial da janela de congestionamento usando o procedimento de partida lenta até a janela de congestionamento alcançar o threshold



CONTROLE DE CONGESTIONAMENTO TCP

Fase de prevenção de congestionamento



CONTROLE DE CONGESTIONAMENTO TCP

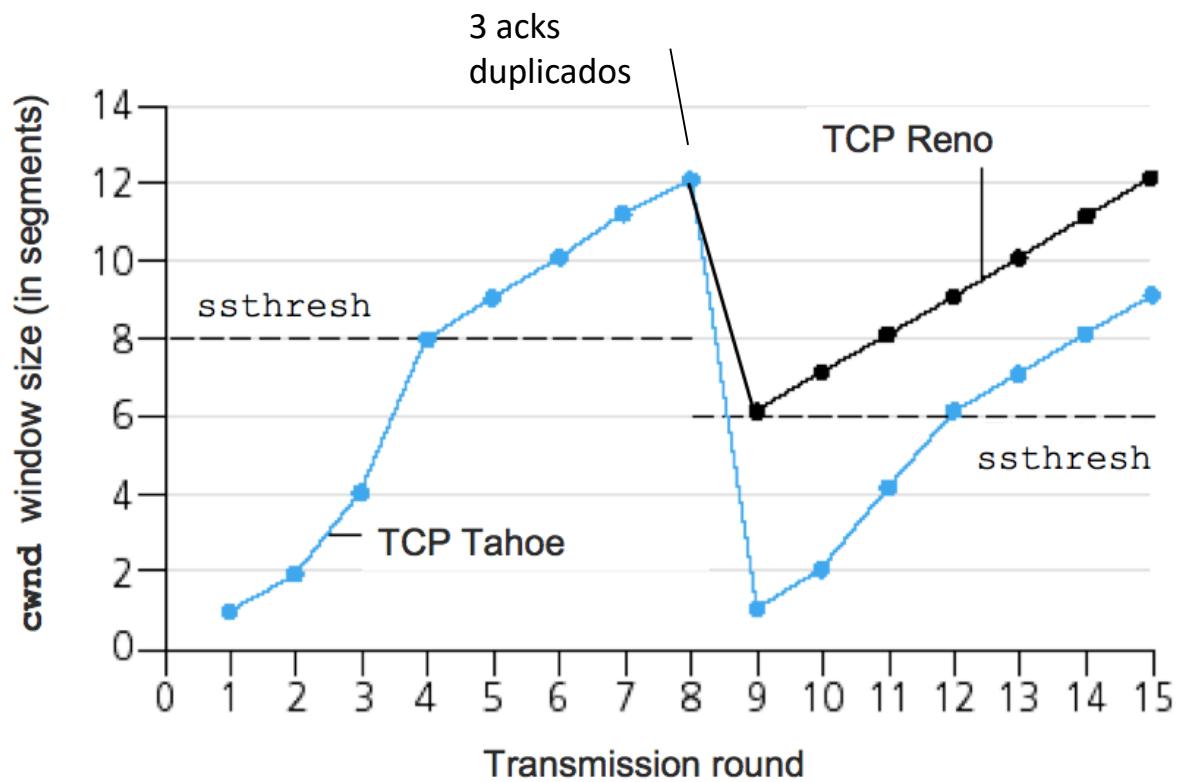
Algoritmo Tahoe

- Primeira implementação, com algoritmos de Partida Lenta, Prevenção de Congestionamento e retransmissão rápida
 - Retransmissão rápida: 3 acks duplicados ocasionam retransmissão do pacote e retorno a partida lenta.
 - Com muitas perdas provoca muitos retornos à partida lenta

Algoritmo Reno

- Opera melhor em redes com muitas perdas
- Tem mecanismo de recuperação rápida que cancela a fase de partida lenta após uma retransmissão rápida

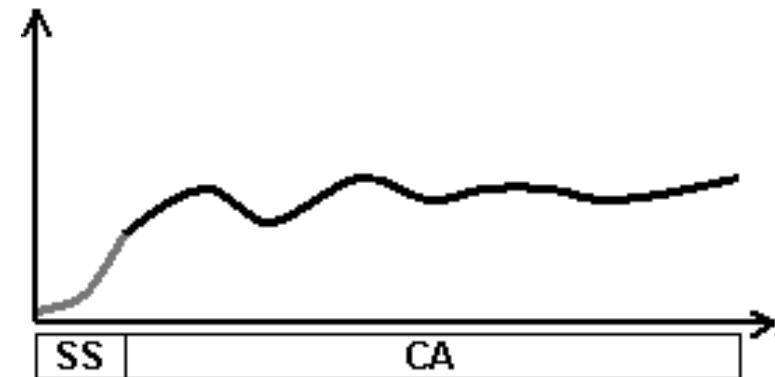
CONTROLE DE CONGESTIONAMENTO TCP



CONTROLE DE CONGESTIONAMENTO TCP

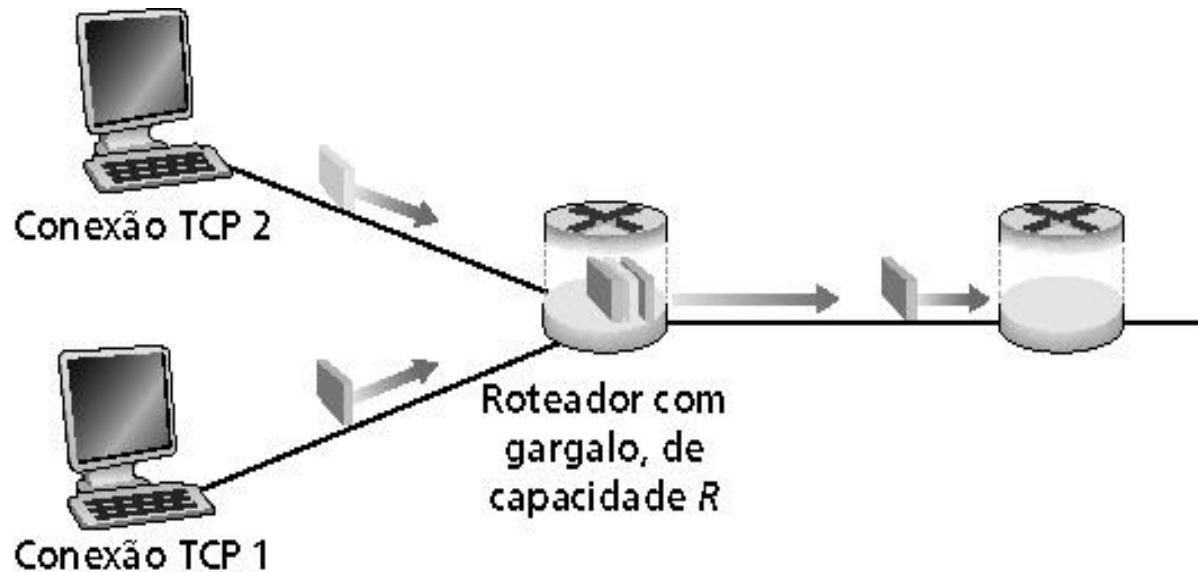
Algoritmo Vegas

- Aumenta o desempenho do Reno
- Tenta evitar o congestionamento mantendo uma boa vazão
- Ideia básica
 - detectar o congestionamento nos roteadores entre a fonte e destino antes da ocorrência da perda
 - reduz a taxa linearmente quando a perda eminent de pacotes é detectada
 - Perda eminent é prevista pela observação do atraso de ida-e-volta
 - Maior o atraso de ida-e-volta dos pacotes, maior o congestionamento nos roteadores



EQUIDADE DO TCP

Objetivo de equidade: se K conexões TCP compartilham o mesmo enlace de gargalo com largura de banda R , cada uma deve ter taxa média de R/K



EQUIDADE DO TCP

Equidade e conexões TCP paralelas

- Conexões no computador tendem a receber a mesma vazão
- Nada previne as aplicações de abrirem conexões paralelas entre 2 hospedeiros
 - Web browsers fazem isso
- Exemplo: enlace de taxa R suportando 9 conexões;
 - Nova aplicação pede 1 TCP, obtém taxa de $R/10$
 - Nova aplicação pede 11 TCPs, cada uma obtém $R/20!$

Equidade e UDP

- Aplicações multimídia normalmente não usam TCP
 - Não querem a taxa estrangulada pelo controle de congestionamento
- Em vez disso, usam UDP:
 - Trafega áudio/vídeo a taxas constantes, toleram perda de pacotes