Instituto de computação

Programa de Pós-Graduação
Instituto de Computação
Universidade Federal Fluminense

Aulas 11, 12 e 13

Camada de Transporte

Conceitos, protocolos UDP e TCP

Igor Monteiro Moraes

Redes de Computadores

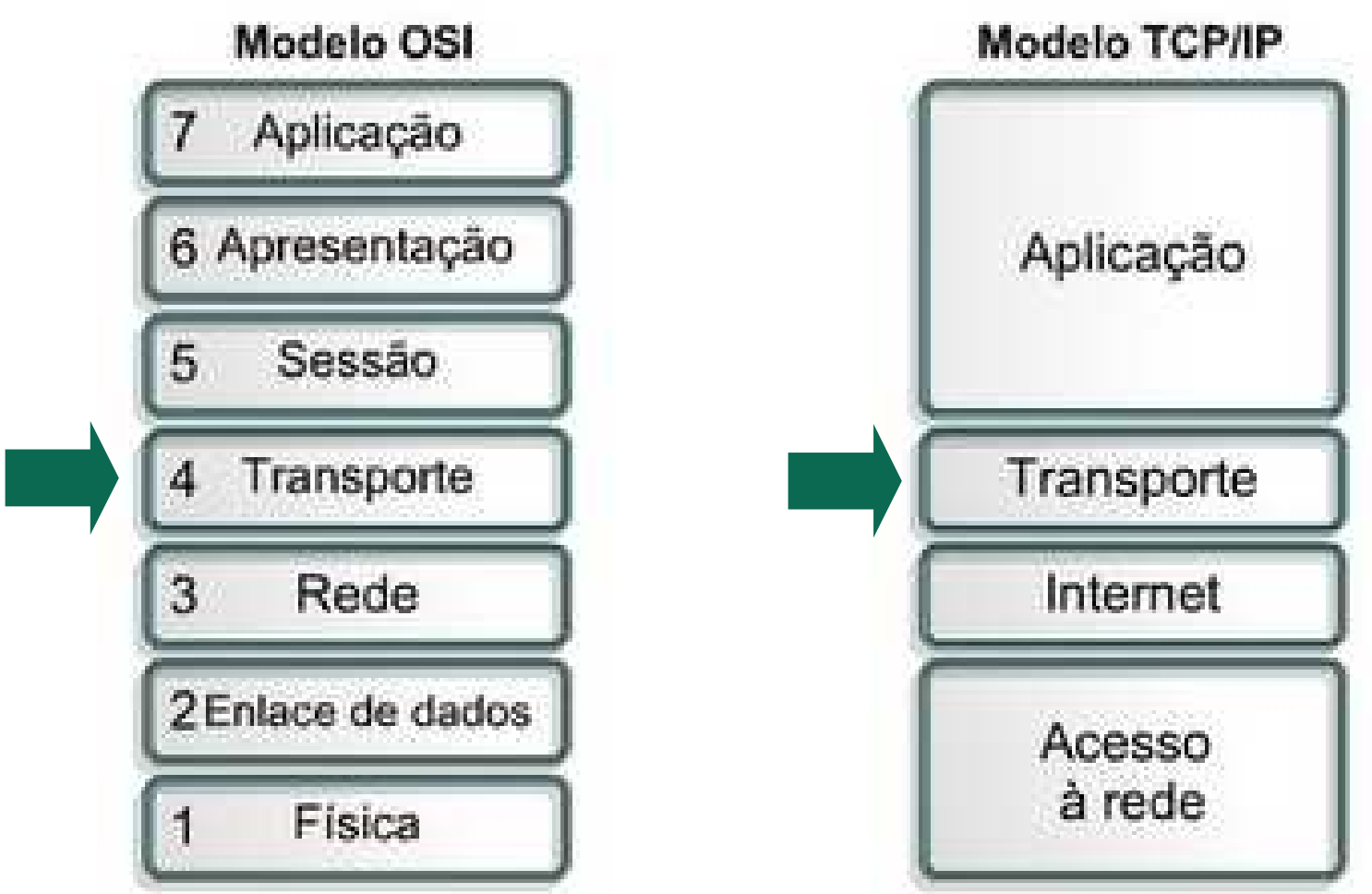
ATENÇÃO!

Instituto de computação

- Este apresentação é contém partes baseadas nos seguintes trabalhos
  - de Rezende, J. F., Costa, L. H. M. K. e Rubinstein, M. G. - "Avaliação Experimental e Simulação do Protocolo TCP em Redes de Alta Velocidade", em XXI Simpósio Brasileiro de Telecomunicações - SBrT'2005, setembro de 2005.
  - Rubinstein, M. G., Moraes, I. M., Campista, M. E. M., Costa, L. H. M. K. e Duarte, O. C. M. B. - "A Survey on Wireless Ad Hoc Networks", em Mobile and Wireless Communications Networks. ISBN: 0-387-34634-1, Springer-Verlag. p. 1-34, agosto de 2006.
  - Augusto, C. H. P., Silva, M. W. R., Cardoso, K. V., Mendes, A. C., Guedes, R. M., and de Rezende, J. F. - "Segmentação de Conexões TCP para Transferência Fim-a-Fim em Alta Velocidade", em WPerformance´2008, pp. 141-160, julho de 2008.
  - Notas de aula do Prof. José Augusto Suruagy Monteiro, disponíveis em http://www.nuperc.unifacs.br/Members/jose.suruagy/cursos

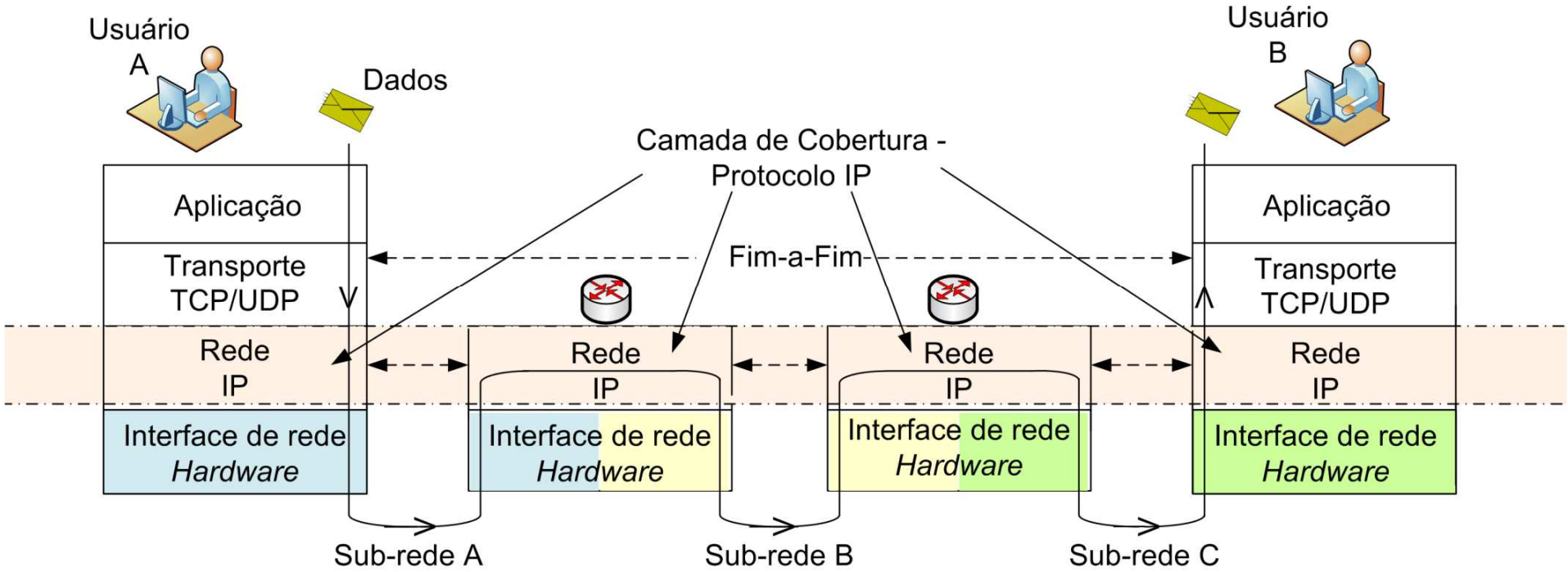
Camada de Transporte

Instituto de computação



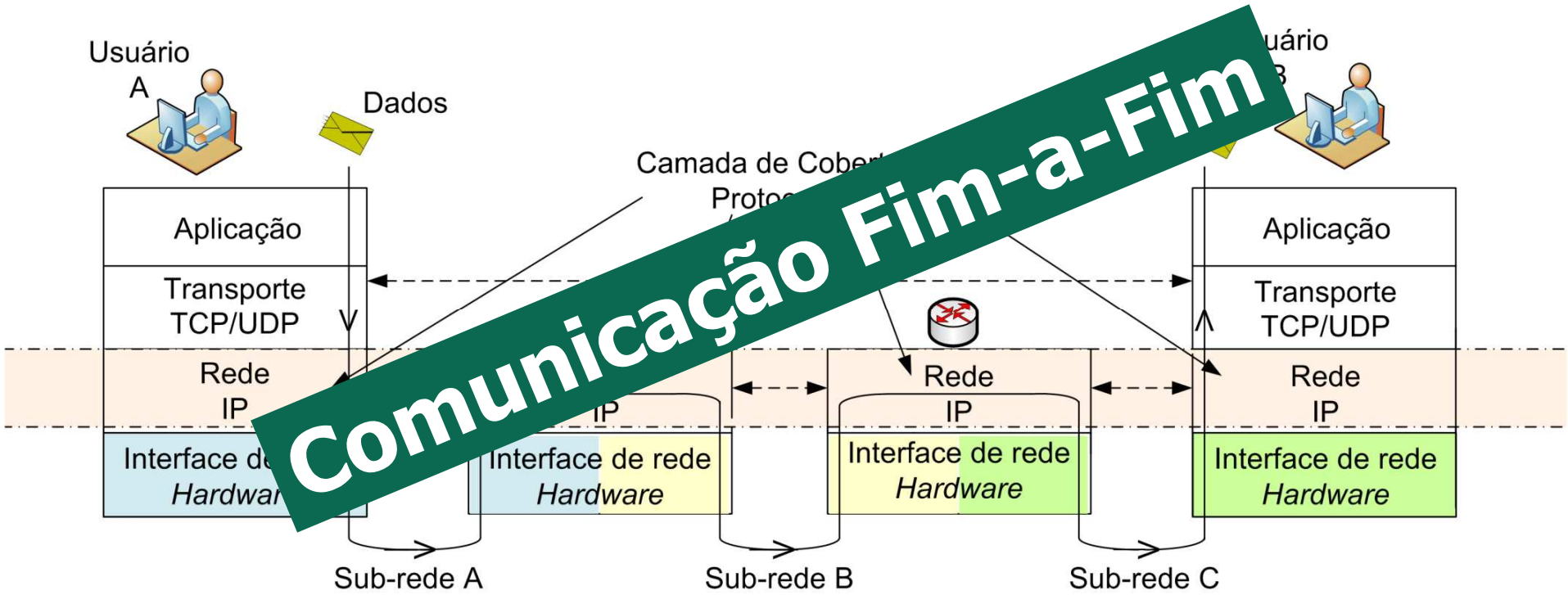
Camada de Transporte

- Provê um **canal lógico** de comunicação entre **processos** em **diferentes sistemas finais**
  - Para a aplicação, os sistemas finais estão diretamente conectados

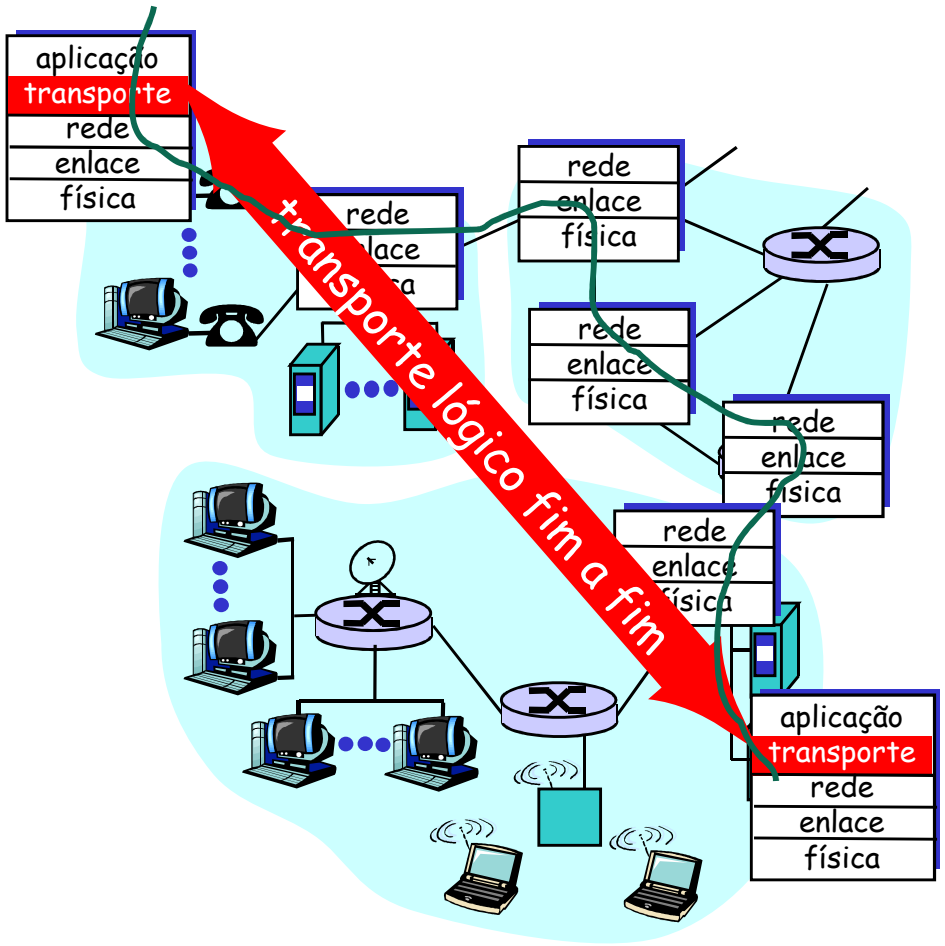


Camada de Transporte

- Provê um **canal lógico** de comunicação entre **processos** em **diferentes sistemas finais**
  - Para a aplicação, os sistemas finais estão diretamente conectados

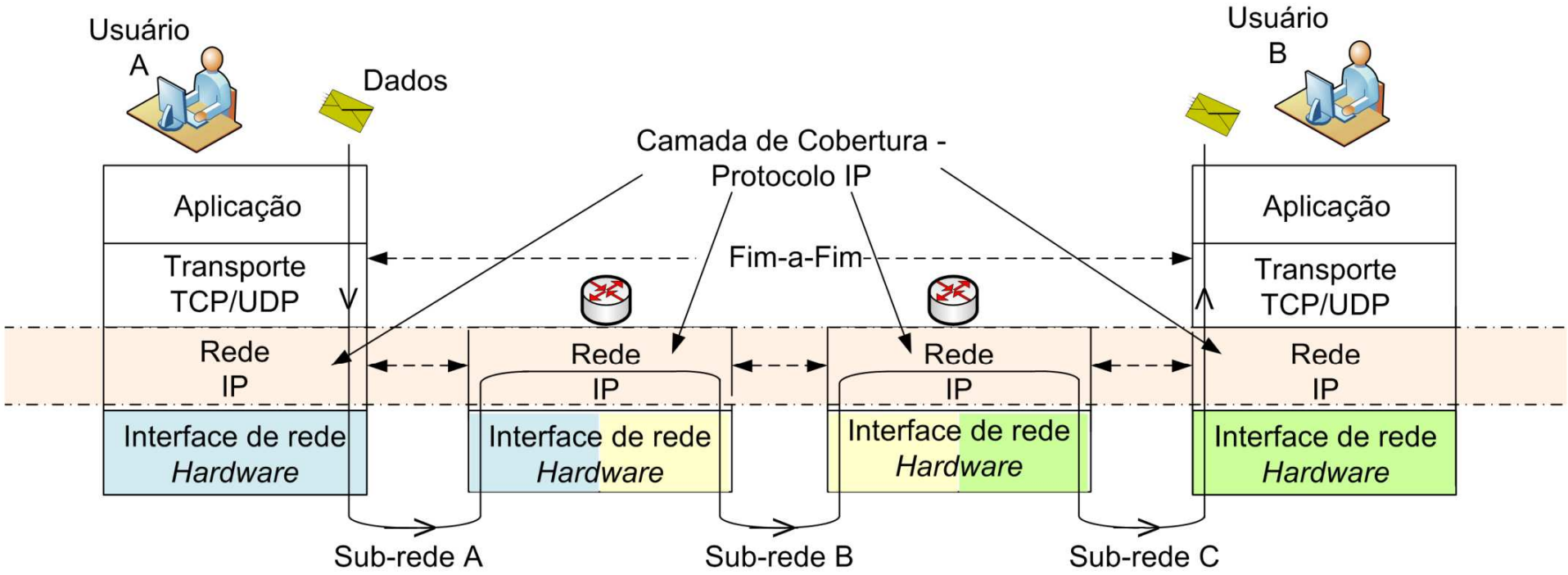


Comunicação Fim-a-Fim



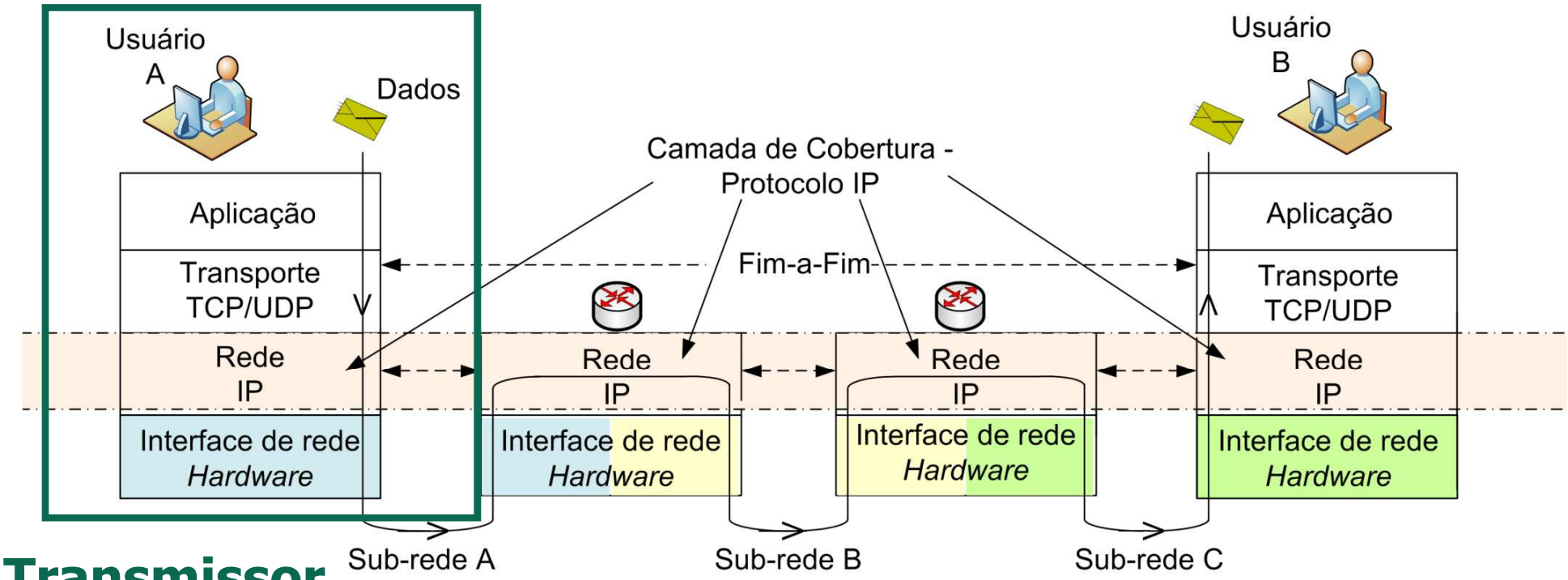
Camada de Transporte

- Protocolos de transporte
  - Executados nos **sistemas finais**



Camada de Transporte

- Protocolos de transporte
  - Executados nos **sistemas finais**

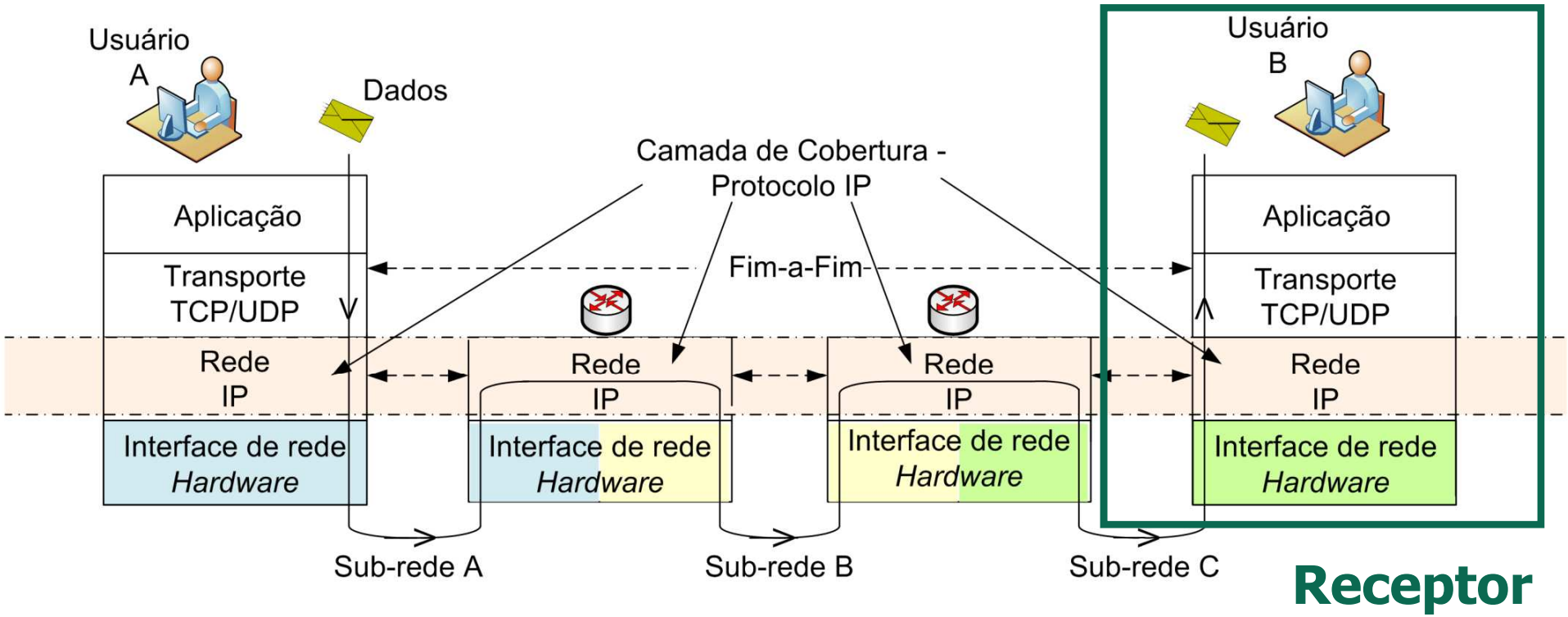


Transmissor

- Converte as mensagens da aplicação em **segmentos**
- Encaminha os segmentos para a camada de rede

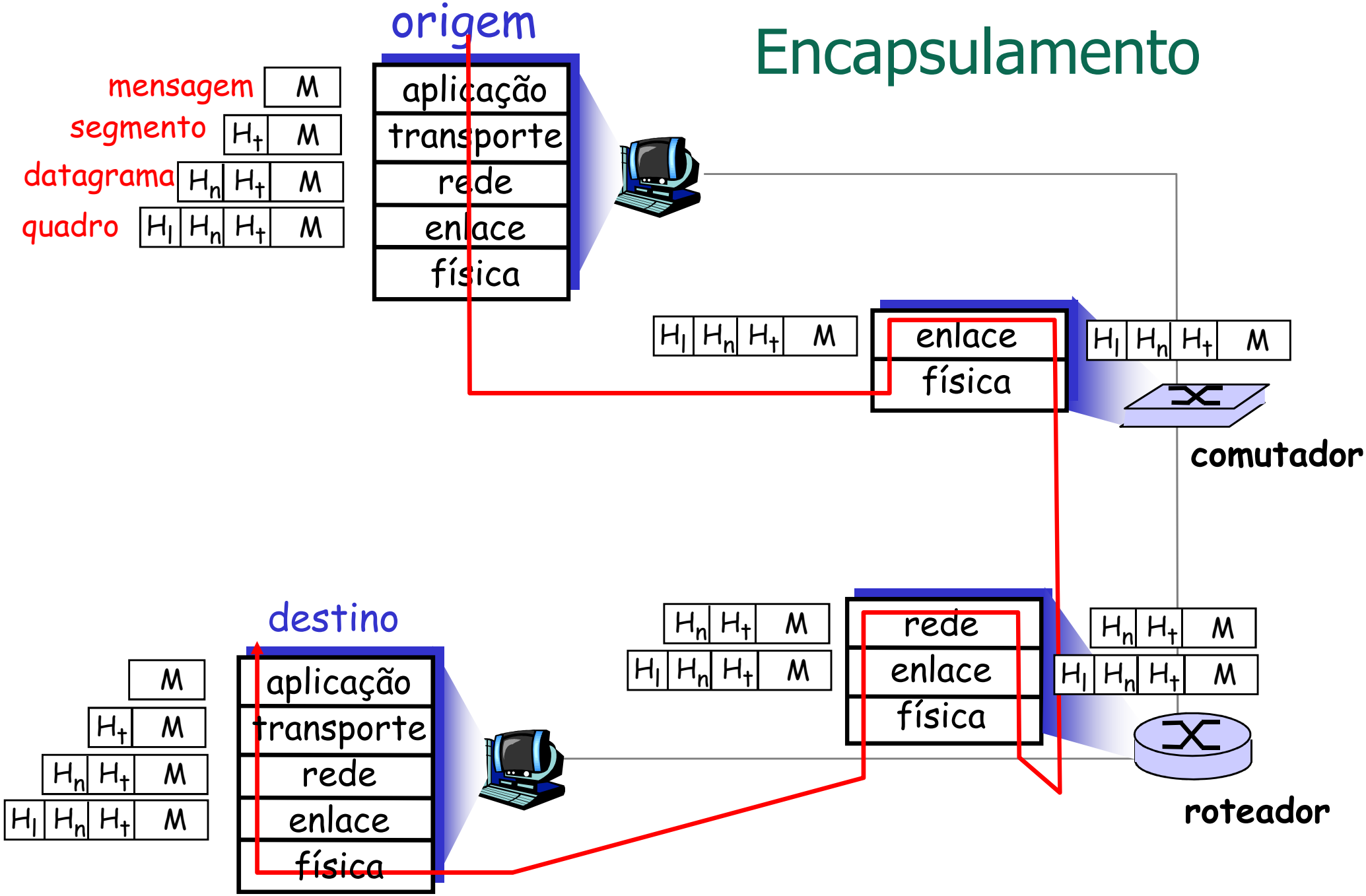
Camada de Transporte

- Protocolos de transporte
  - Executados nos **sistemas finais**



- Recebe os segmentos da camada de rede
- Remonta as mensagens e encaminha para a aplicação

Encapsulamento



Camada de Transporte

Instituto de computação

- Existem diferentes protocolos de transporte
  - Fornecem diferentes tipos de serviços
  - Aplicações usam o mais adequado ao seu propósito

- Na Internet
  - *User Datagram Protocol* (UDP)
  - *Transmission Control Protocol* (TCP)

Transporte x Rede

Instituto de computação

- Camada de transporte
  - Canal lógico de comunicação entre **processos**

depende dos serviços e pode estender os serviços

- Camada de rede
  - Canal lógico de comunicação entre **estações**

Transporte x Rede

Instituto de computação

- Serviço da camada de rede
  - Entrega de **melhor esforço**
  - Não garante a
    - Entrega dos segmentos
    - Ordenação dos segmentos
    - Integridade dos dados contidos nos segmentos

Serviço não-confiável

Transporte x Rede

Instituto de computação

- Serviços da camada de transporte
  - Estender o serviço de entrega da camada de rede
    - **Rede**: entre **sistemas finais**
    - **Transporte**: entre **processos** rodando em sistemas finais
      - Multiplexação e demultiplexação
  - Verificação de integridade
    - Campos de detecção de erros no cabeçalho

Serviços mínimos

Protocolos

Instituto de computação

- UDP
  - Somente os serviços mínimos
    - Entrega não-confiável e não-ordenada
- TCP
  - Mais do que os serviços mínimos
    - Entrega confiável e ordenada
      - Estabelecimento de conexão
      - Controle de congestionamento
      - Controle de fluxo

Protocolos

Instituto de computação

- UDP
  - Somente os serviços mínimos
    - Entrega não-confiável e não-ordenada
- TCP
  - Mais do que os serviços mínimos
    - Entrega confiável e ordenada
      - Estabelecimento de conexão
      - Controle de congestionamento
      - Controle de fluxo

Não garantem requisitos de atraso de banda passante

Comunicação entre Processos

Instituto de computação

- Processo
  - É um programa em execução em um sistema final
  - Em um **mesmo sistema final**
    - Dois processos se comunicam usando comunicação interprocessos definida pelo SO
  - Processos em **sistemas finais diferentes**
    - Se comunicam através da **troca de mensagens**
      - Processo cliente é o que inicia a comunicação
      - Processo servidor é que o espera para ser contatado
      - Em aplicações P2P, sistemas finais executam os dois processos

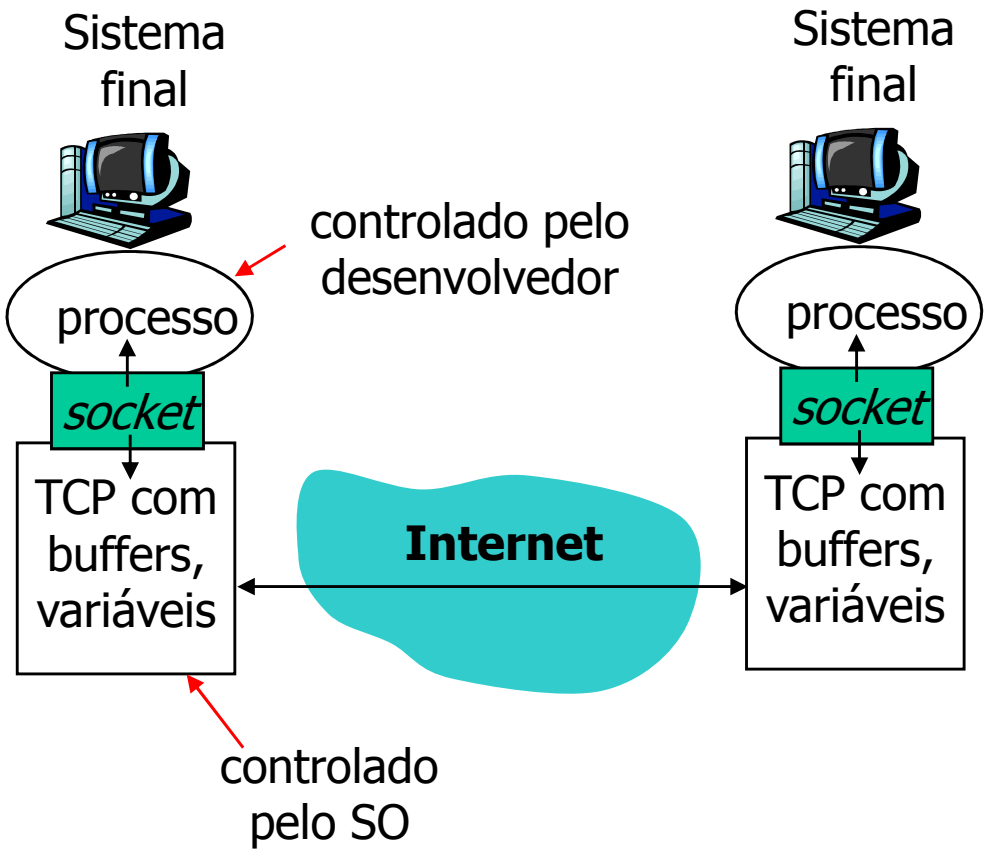
Sockets

Instituto de computação

- Um processo envia/recebe mensagens para/do seu ***socket***
  - É uma interface de software entre as camadas de aplicação e de transporte
    - API (*Application Programming Interface*) entre a aplicação e a rede
- Analogia: processo é uma casa → *socket* é a porta da casa
  - Envio: mensagem é empurrada pela porta
    - Emissor confia na infraestrutura de transporte do outro lado da porta para levar a mensagem até o destinatário
  - Recepção: mensagem passa através da porta
    - Receptor executa ações sobre a mensagem

Sockets

- API permite ao desenvolvedor
  - Controlar **tudo** do lado da camada de **aplicação**
  - Controlar **pouco** do lado da camada de **transporte**
    1. Escolha do protocolo
    2. Alguns parâmetros do protocolo



Endereçamento de Processos

Instituto de computação

- Processos devem ter um **identificador**
- Uma interface de rede de uma estação possui um identificador único
  - Endereço IP de 32 bits

- O endereço IP de uma estação na qual um processo está sendo executado é suficiente para identificar o processo?

Endereçamento de Processos

Instituto de computação

- Processos devem ter um **identificador**
- Uma interface de rede de uma estação possui um identificador único
  - Endereço IP de 32 bits

- O endereço IP de uma estação na qual um processo está sendo executado é suficiente para identificar o processo?

Não!
Vários processos podem estar em execução numa mesma estação

Endereçamento de Processos

Instituto de computação

- Identificador inclui tanto o **endereço IP** quanto **números de portas**
  - Associados ao processo em execução na estação
- Exemplos de números de portas
  - Servidor HTTP: 80
  - Servidor de email: 25
- Para enviar uma mensagem HTTP para o servidor **`www.ic.uff.br`**
  - Endereço IP: 200.20.15.38
  - Número de porta: 80

Programação com *Sockets*

Instituto de computação

- Socket API
  - Introduzida no Unix BSD 4.1 em 1981
  - *Sockets* são criados, usados e liberados explicitamente pelas aplicações
  - Baseados no paradigma cliente-servidor
  - Dois tipos de serviços de transporte através da API
    - Datagrama não-confiável (UDP)
    - Orientado a fluxo de bytes e confiável (TCP)

- Atualmente disponível em diferentes linguagens de programação

Programação com *Sockets*

Instituto de computação

- Mais detalhes
  - Seções 2.7 e 2.8 do Livro-base
    - Exemplos no site do livro: http://www.aw-bc.com/kurose-ross
  - Beej's Guide to Network Programming
    - Disponível em http://beej.us/guide/bgnet

(De)Multiplexação

Instituto de computação

- É um dos serviços mínimos
  - Identificar a qual processo pertence um segmento
  - Encaminhar para o processo correto

- *Socket*
  - Interface entre a camada de aplicação e a de transporte dentro de uma máquina

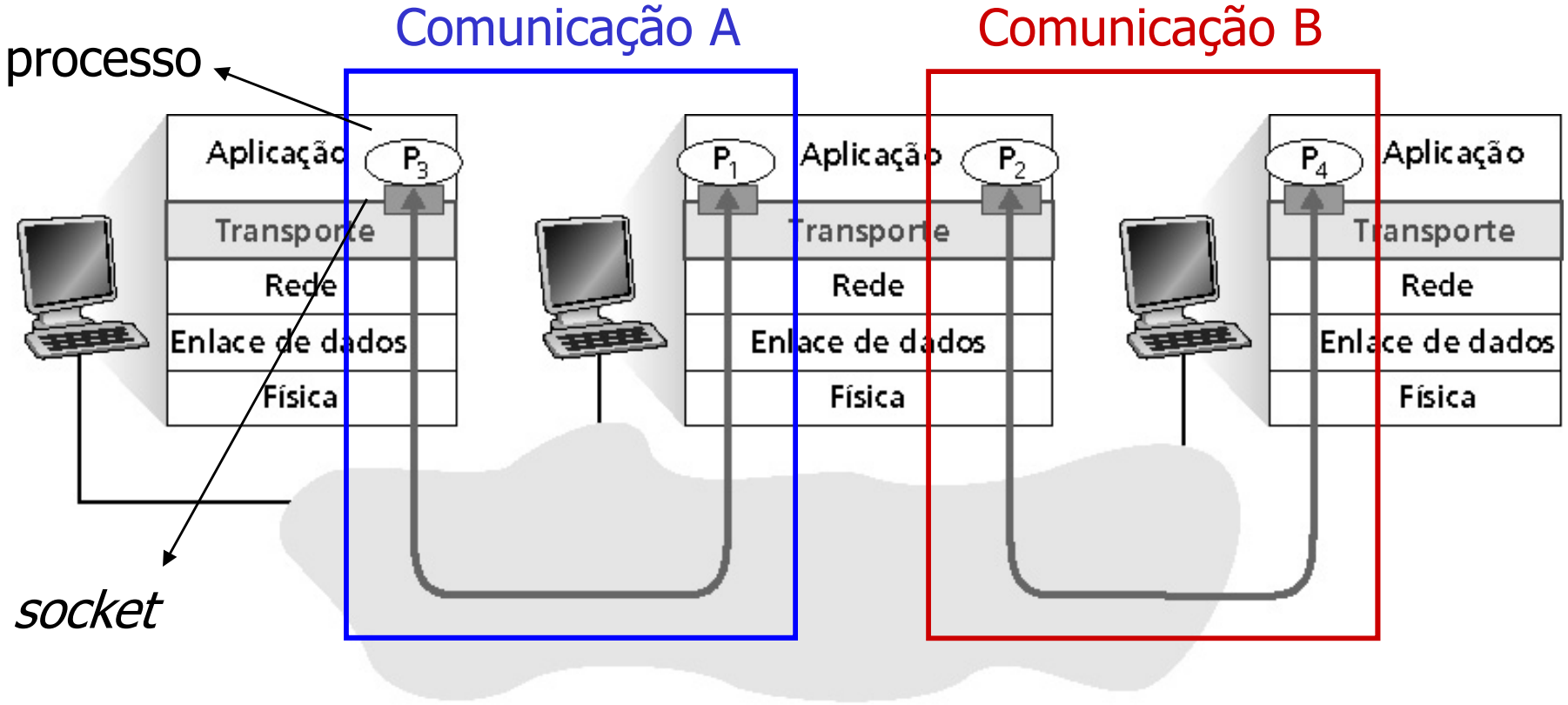
(De)Multiplexação

Instituto de computação

- Multiplexação
  - Receber as mensagens de vários *sockets*
  - Agrupar as mensagens de um mesmo *socket*
  - Adicionar um cabeçalho

- Demultiplexação
  - Encaminhar um segmento para o *socket* correto
    - Baseado nos dados do cabeçalho

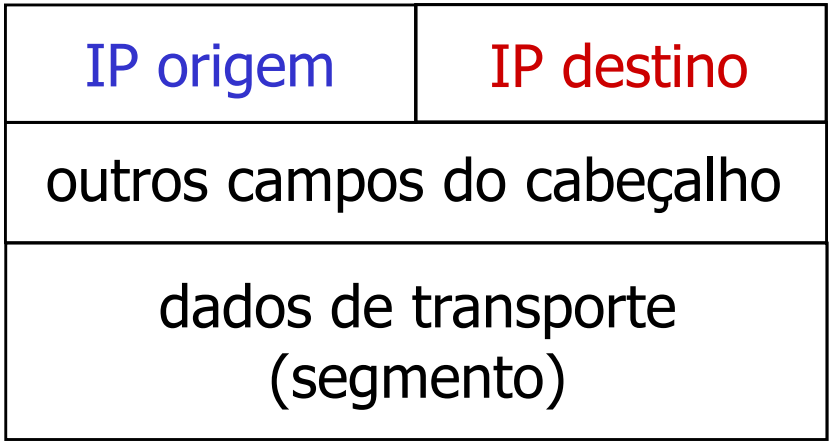
(De)Multiplexação



Demultiplexação

Instituto de computação

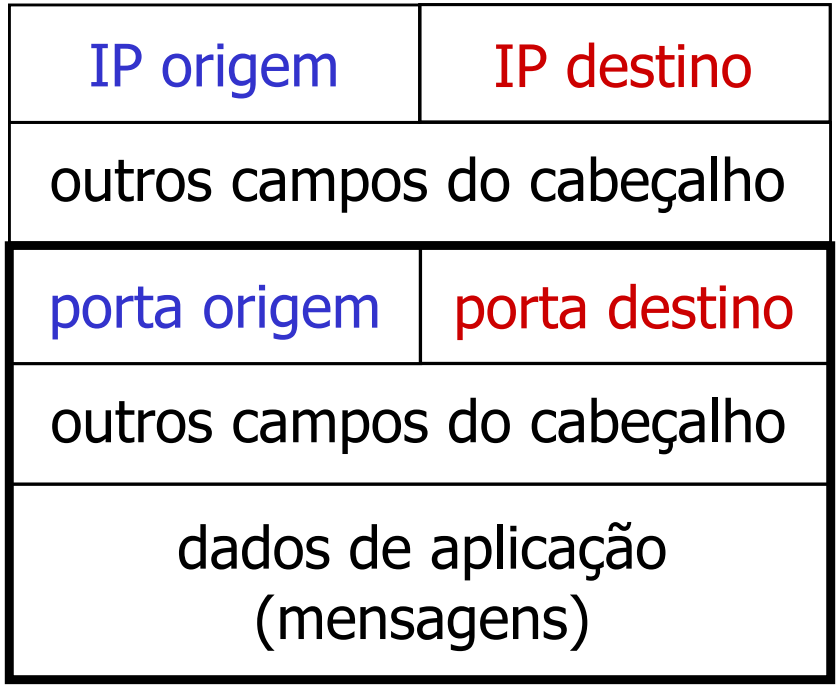
- Feita com base nos campo do cabeçalho dos segmentos e datagramas



Demultiplexação

Instituto de computação

- Feita com base nos campo do cabeçalho dos segmentos e datagramas



Demultiplexação

Instituto de computação

- Depende do tipo de serviço oferecido pela camada de transporte
  - Orientado à conexão
  - Não-orientado à conexão

Demultiplexação com UDP

Instituto de computação

- Não-orientada à conexão
- Identificação feita por
  - Endereço IP de **destino**
    - Chegar ao sistema final correspondente
  - Número da porta de **destino**

- Quando o sistema final recebe um segmento UDP
  1. Verifica o número da porta de destino no segmento
  2. Encaminha o segmento UDP para o *socket* com aquele número de porta

Demultiplexação com UDP

Instituto de computação

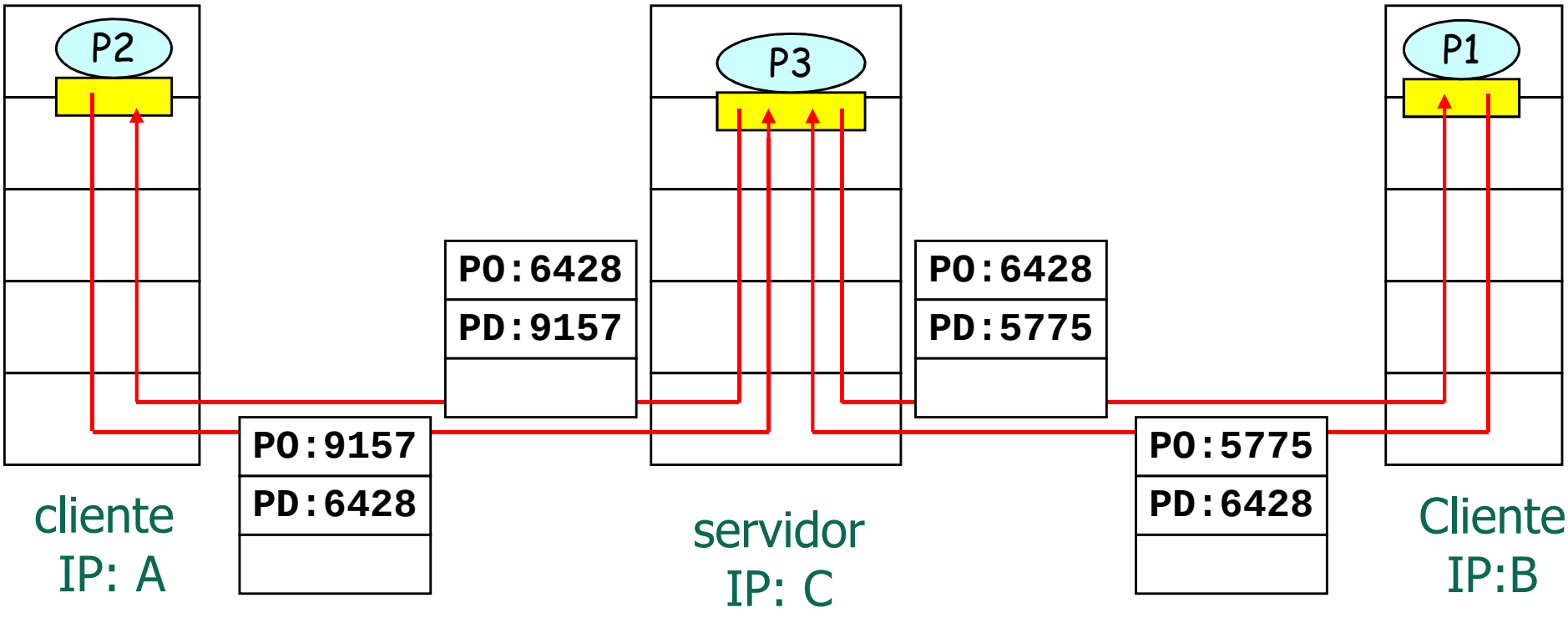
- Um *socket* pode receber datagramas com **diferentes** endereços **IP origem** e/ou # de **porta de origem**?

Sim!

Somente as informações do destino são usadas

Demultiplexação com UDP

- Como são usadas as informações de origem?



Endereço de retorno

Demultiplexação com TCP

Instituto de computação

- Orientada à conexão
- Identificação feita por
  - Endereço IP de **origem**
  - Número da porta de **origem**
  - Endereço IP de **destino**
  - Número da porta de **destino**
- Quando o hospedeiro recebe um segmento TCP
  1. Verifica o número das portas de origem e destino no segmento
  2. Encaminha o segmento TCP para o *socket* com aqueles números de porta

Demultiplexação com TCP

Instituto de computação

- Um *socket* pode receber datagramas com **diferentes** endereços **IP origem** e/ou # de **porta de origem**?

Não!

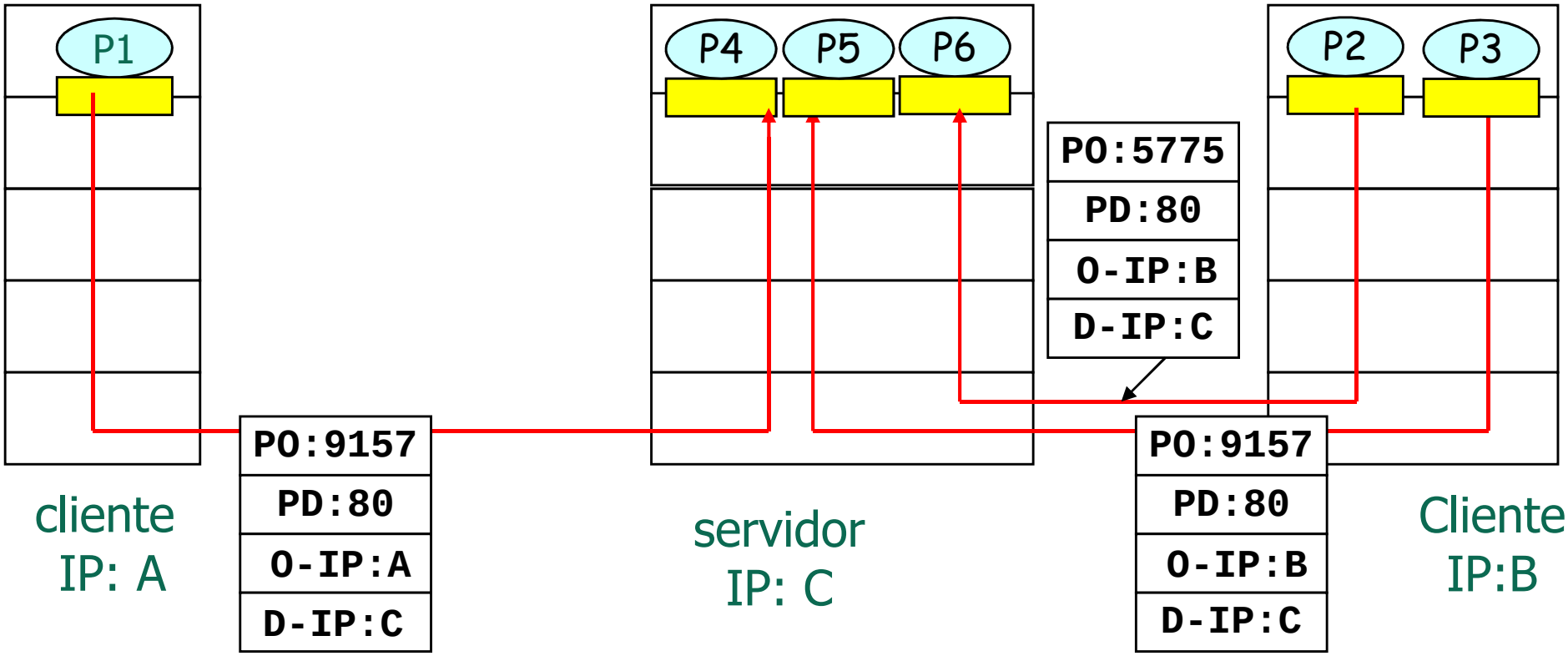
Cada segmento será direcionado para um *socket*

Demultiplexação com TCP

Instituto de computação

- Um servidor pode dar suporte a muitos *sockets* TCP simultâneos
  - Cada *socket* é identificado pela sua própria quádrupla
- Servidores Web têm *sockets* diferentes para cada conexão cliente
  - HTTP não persistente terá *sockets* diferentes para cada pedido

Demultiplexação com TCP



Instituto de Computação

User Datagram Protcol (UDP)

UDP

Instituto de computação

- Definido pela RFC 768
- Protocolo de transporte mínimo
  - Serviço de melhor esforço
- Segmentos UDP podem ser
  - Perdidos
  - Entregues à aplicação fora de ordem
- Sem conexão
  - Não há "setup" entre remetente e o receptor
  - Tratamento independente de cada segmento UDP

UDP

Instituto de computação

- Por quê é necessário?
  - Elimina o estabelecimento de conexão
    - Menor latência
  - É simples
    - Não mantém "estado" da conexão nem no remetente, nem no receptor
  - Cabeçalho de segmento reduzido
  - Não há controle de congestionamento
    - UDP pode transmitir tão rápido quanto desejado (e possível)

Requisitos das Aplicações

Aplicação	Perda	Banda passante	Atraso
Transferência de arquivos	sem perdas	elástica	tolerante
Email	sem perdas	elástica	tolerante
Web	sem perdas	elástica	tolerante
Áudio/vídeo em tempo real	tolerante*	áudio: 5kb-1Mb vídeo:10kb-5Mb	centenas de miliseg.
Áudio/vídeo gravado	tolerante*	Idem	poucos seg.
Jogos interativos	tolerante	até 10 kbps	centenas de miliseg.
Mensagens instantâneas	sem perdas	elástica	sim/não (?)

Protocolos por Aplicação

Aplicação	Protocolo de aplicação	Protocolo de transporte
Servidor de arquivos remoto	NFS	Tipicamente UDP
Gerenciamento de rede	SNMP	Tipicamente UDP
Protocolo de roteamento	RIP	Tipicamente UDP
Tradução de nomes	DNS	Tipicamente UDP

Protocolos por Aplicação

Aplicação	Protocolo de aplicação	Protocolo de transporte
Email	SMTP	TCP
Acesso remoto	Telnet, SSH	TCP
Web	HTTP	TCP
Transferência de arquivos	FTP	TCP
Distribuição multimídia	HTTP, RTP	TCP ou UDP
Telefonia na Internet	SIP, RTP, proprietário (Skype)	TCP ou UDP

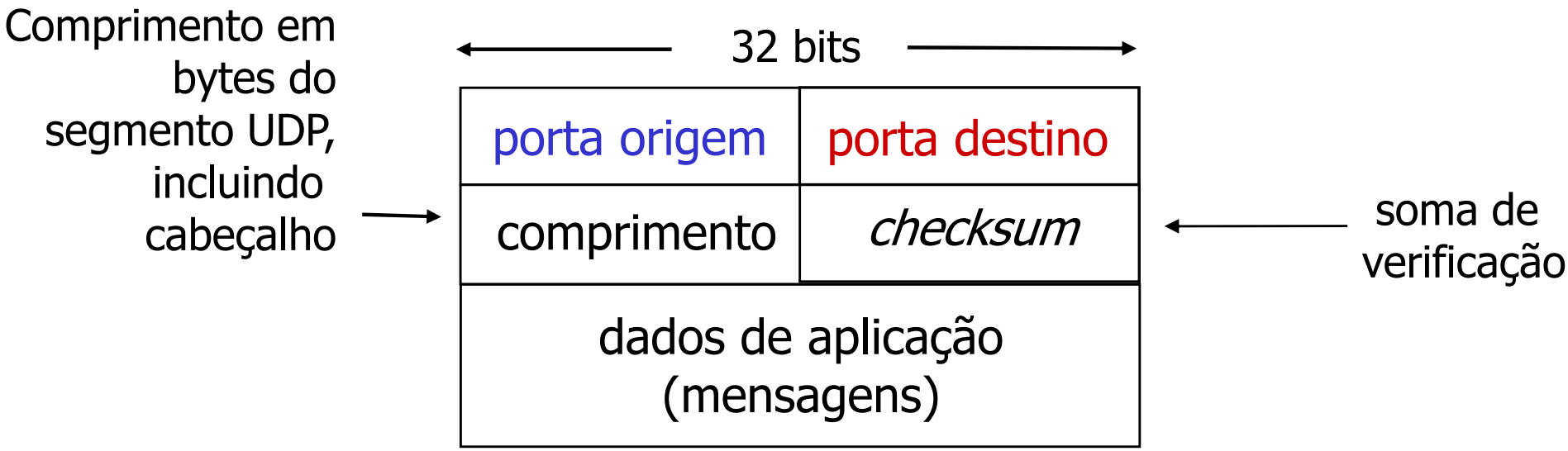
UDP

Instituto de computação

- Utilizado para aplicações multimídias
  - Tolerantes a perdas
  - Sensíveis à taxa de transmissão
- Outros usos
  - DNS → Reduzir a latência na requisição de páginas Web
  - SNMP → Reduzir o tempo de reação a um problema na rede
- Transferência confiável com UDP?
  - É necessário acrescentar confiabilidade na camada de aplicação
    - Recuperação de erro específica para cada aplicação

UDP

- Formato do segmento
  - Cabeçalho de 8 bytes



UDP

Instituto de computação

- Soma de verificação (*checksum*)
  - Usada para detectar "erros" no segmento transmitido
    - Ex.: bits trocados

Transmissor:

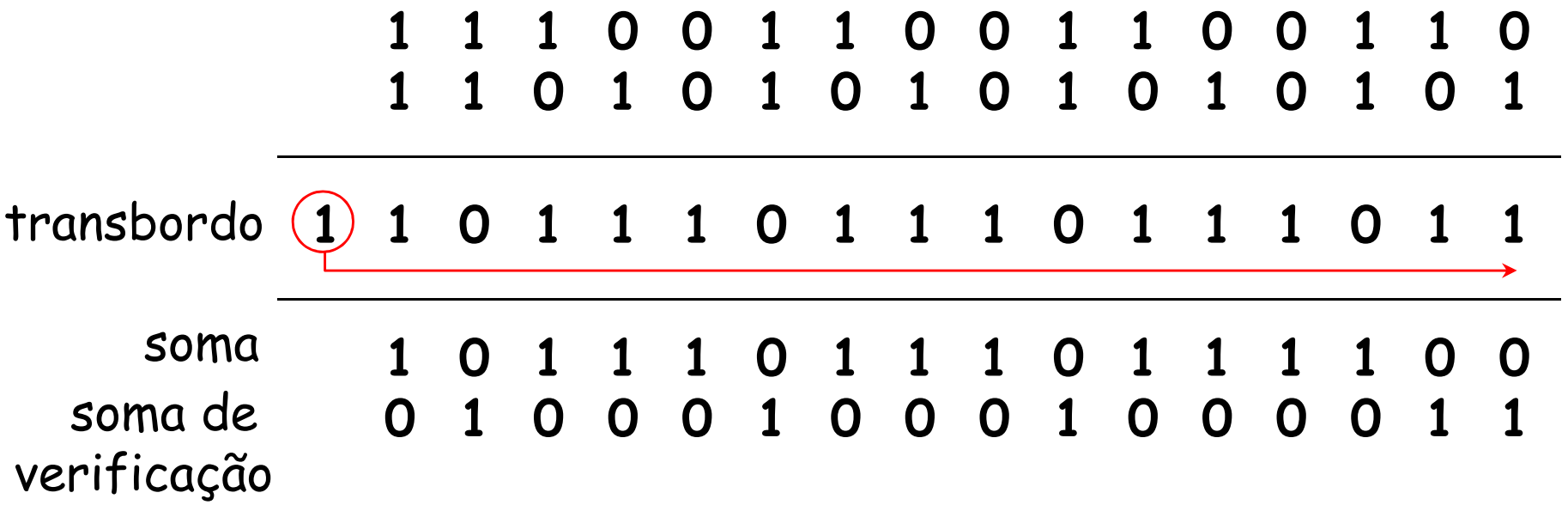
- Trata conteúdo do segmento como seqüência de inteiros de 16-bits
- Campo *checksum* zerado
- *Checksum*: soma (adição usando complemento de 1) do conteúdo do segmento
- Transmissor coloca **complemento do valor da soma** no campo *checksum* do cabeçalho UDP

Receptor:

- Calcula *checksum* do segmento recebido
- Soma do *checksum* computado com o enviado no segmento é igual a FFFF?
  - NÃO - erro detectado
  - SIM - nenhum erro detectado.
    - Mas ainda pode ter erros?
      - Veja depois...

Exemplo de *Checksum*

- Ao adicionar números, o transbordo (vai um) do bit mais significativo deve ser adicionado ao resultado
  - Exemplo: adição de dois inteiros de 16-bits



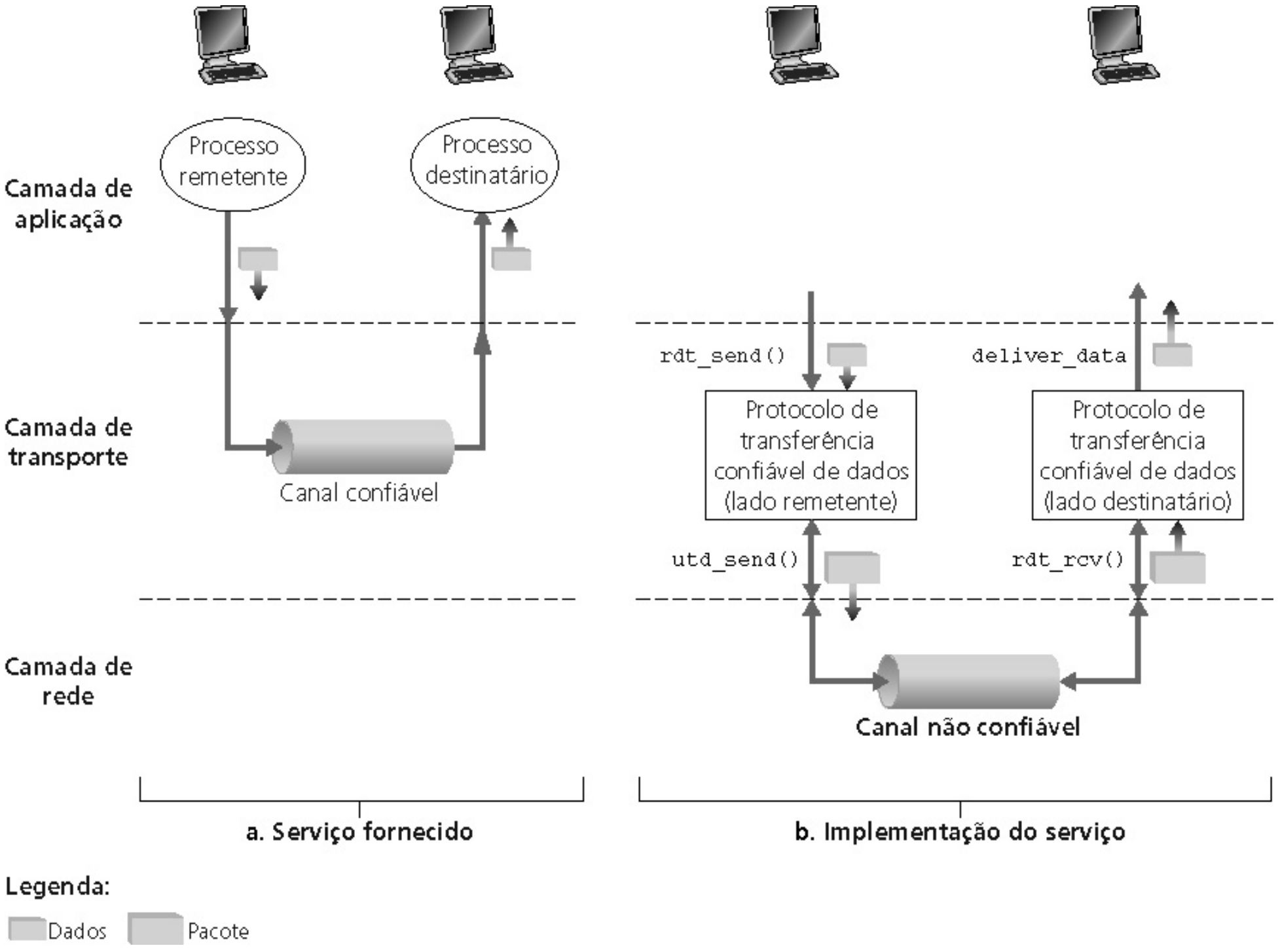
Instituto de computação

Transferência Confiável: Princípios

Transferência Confiável

Instituto de computação

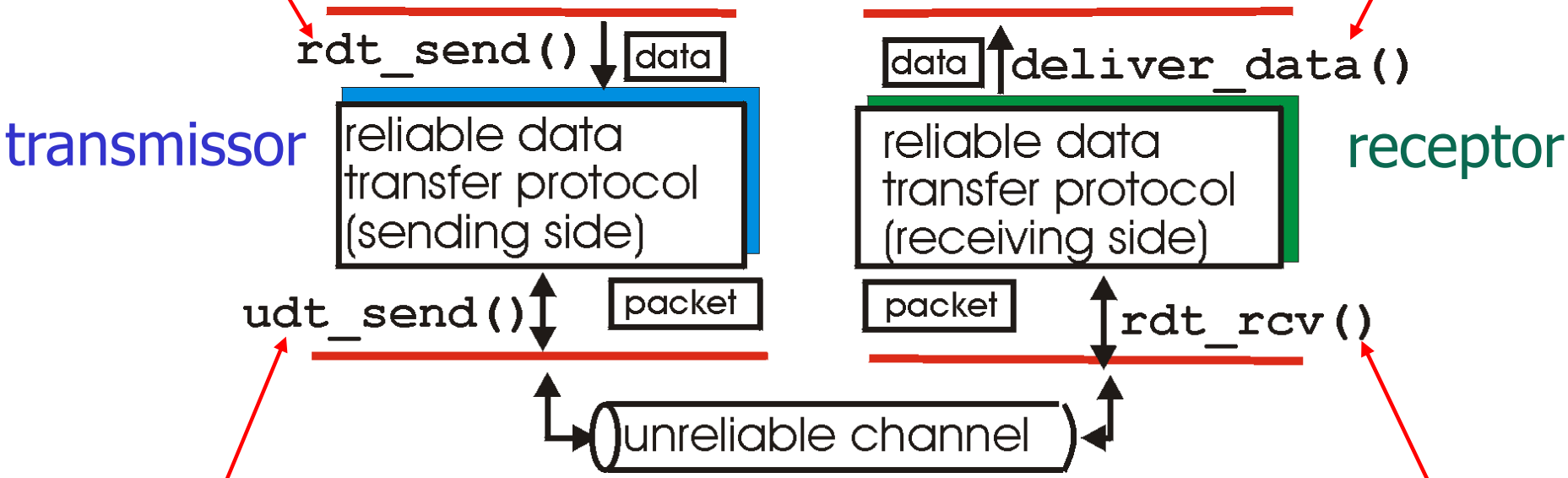
- Importante nas camadas de transporte, enlace, etc.
- Na lista dos 10 tópicos mais importantes em redes
- Características do canal não confiável determinam a complexidade de um protocolo de transferência confiável de dados (rdt)


Camada de aplicação
Camada de transporte
Camada de rede
Processo remetente
Processo destinatário
Canal confiável
rdt_send()
deliver_data
Protocolo de transferência confiável de dados (lado remetente)
Protocolo de transferência confiável de dados (lado destinatário)
utd_send()
rdt_rcv()
Canal não confiável
a. Serviço fornecido
b. Implementação do serviço
Legenda:
Dados
Pacote

Transferência Confiável

rdt_send(): chamada de cima, (ex.: pela apl.). Passa dados p/ serem entregues à camada sup. do receptor

deliver_data(): chamada pela entidade de transporte p/ entregar dados p/ camada superior



udt_send(): chamada pela entidade de transporte, p/ transferir pacotes para o receptor sobre o canal não confiável

rdt_rcv(): chamada quando pacote chega no lado receptor do canal

Transferência Confiável

Instituto de computação

- O que é um **canal confiável**?
  - Nenhum dado transmitido é **corrompido**
  - Nenhum dado transmitido é **perdido**
  - Todos os dados são entregues **ordenadamente**

- Protocolo de transferência confiável de dados
  - Responsável por implementar um canal confiável

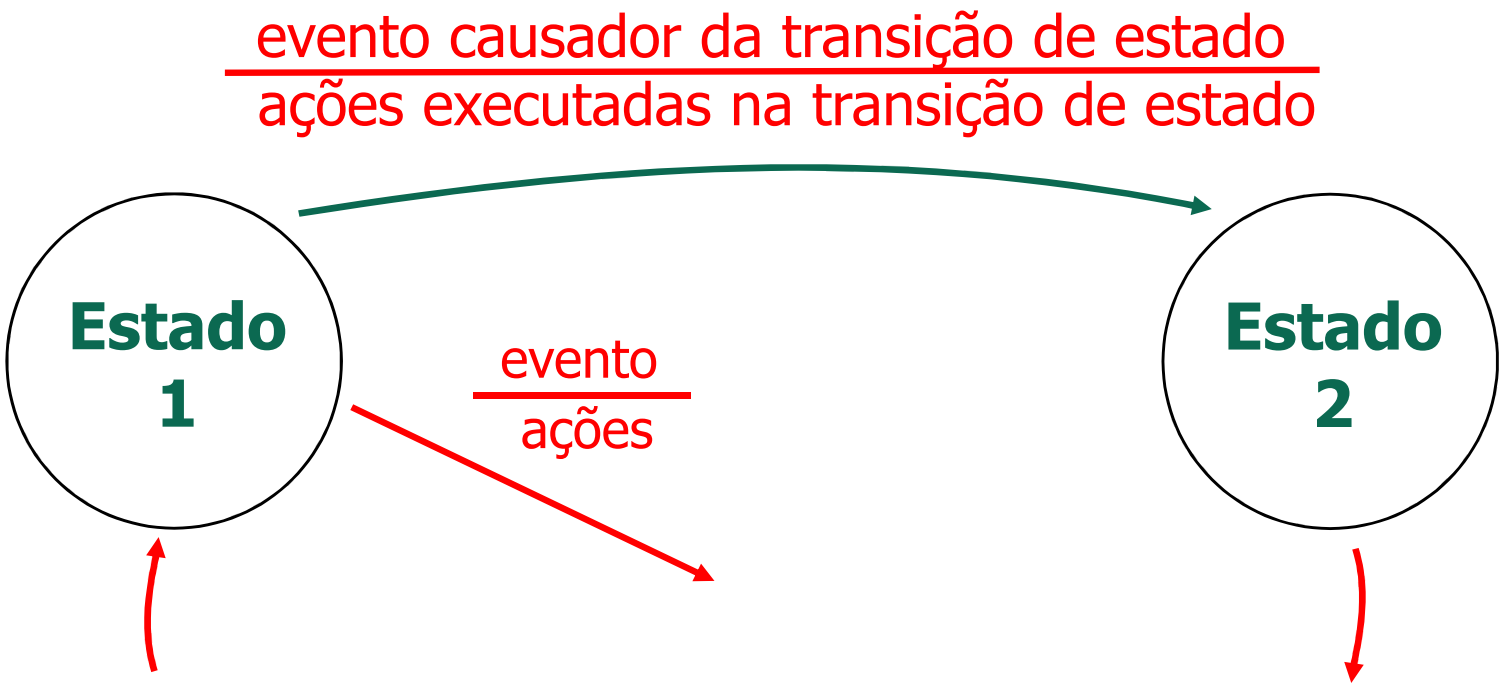
Transferência Confiável

Instituto de computação

- Quais os mecanismos usados para prover um canal confiável?
  - Serão vistos passo-a-passo

- Desenvolver incrementalmente os lados transmissor e receptor de um protocolo confiável de transferência de dados (rdt)
- Considerar apenas fluxo unidirecional de dados
  - Informações de controle fluem em ambos os sentidos
- Usar máquinas de estados finitos (FSM) para especificar os protocolos

Transferência Confiável



estado:

neste "estado" o próximo estado é determinado unicamente pelo próximo evento

Canal Confiável

- Protocolo **`rdt 1.0`**
- Canal de transmissão perfeitamente confiável
  - não há erros de bits
  - não há perda de pacotes
- FSMs separadas para transmissor e receptor
  - transmissor envia dados pelo canal subjacente
  - receptor lê os dados do canal subjacente

a. rdt1.0: lado remetente

b. rdt1.0: lado destinatário

Canal com Erros

Instituto de computação

- Protocolo **`rdt 2.0`**
- Canal pode trocar valores dos bits num pacote
  - É necessário detectar os erros: soma de verificação
- Como recuperar esses erros?
  - **Reconhecimentos positivos (ACKs)**
    - Receptor avisa explicitamente ao transmissor que o pacote foi recebido corretamente
  - **Reconhecimentos negativos (NAKs)**
    - Receptor avisa explicitamente ao transmissor que o pacote tinha erros
    - Transmissor reenvia o pacote ao receber um NAK

Canal com Erros

Instituto de computação

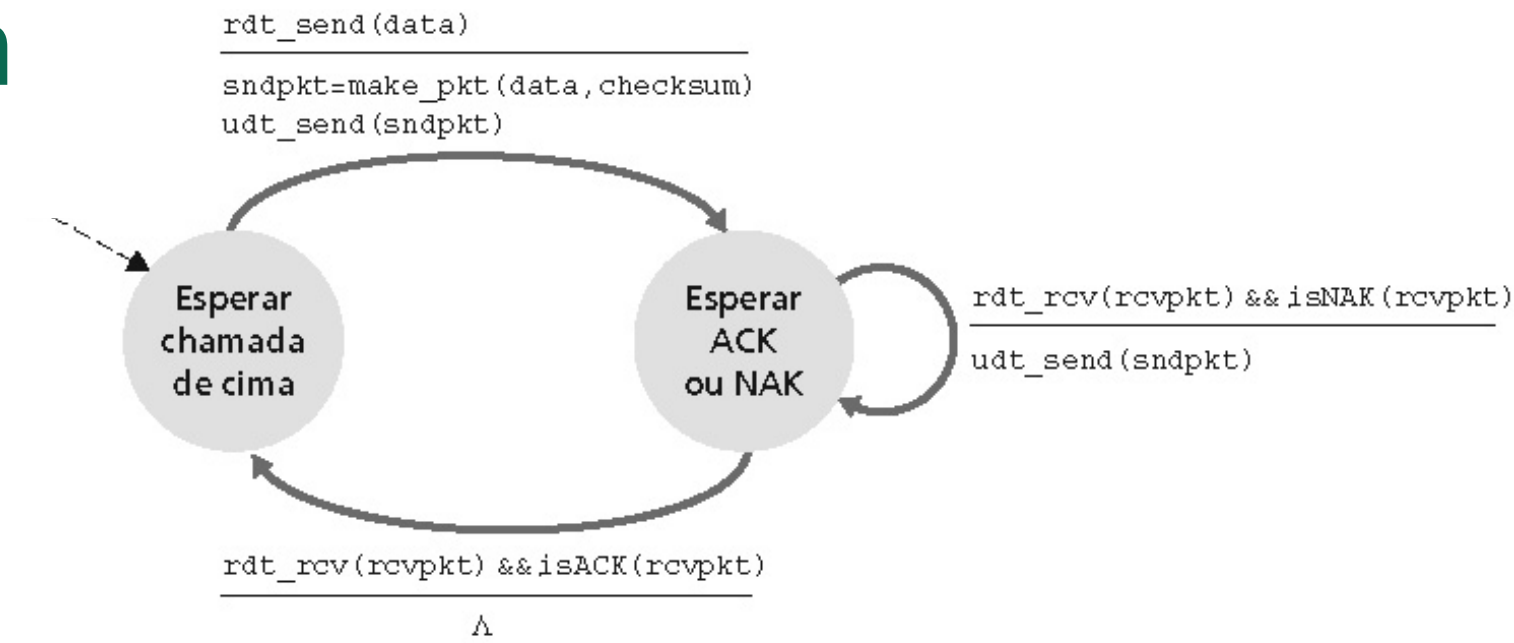
- Se o canal tem erros

Detecção de erros

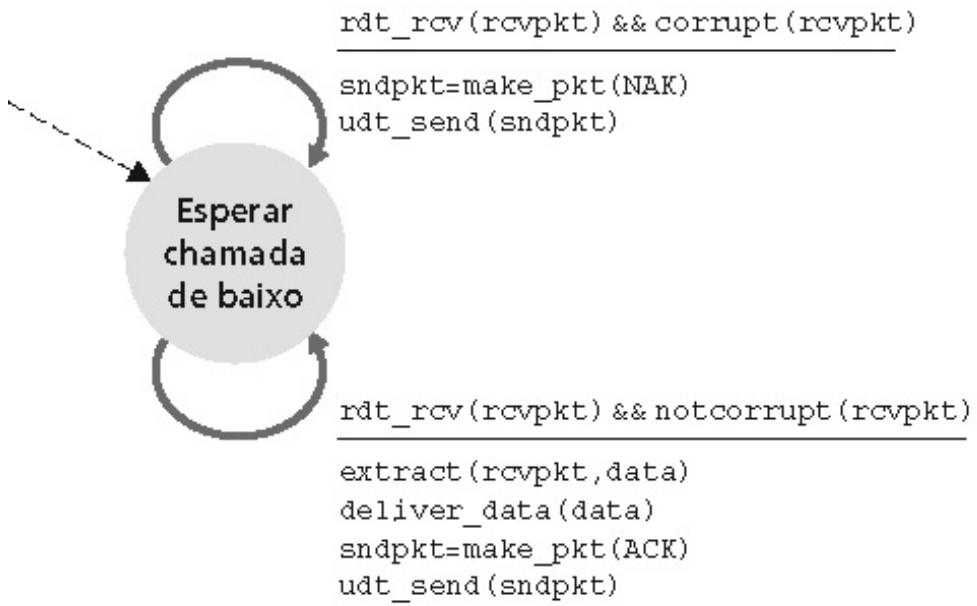
+

Mecanismos automáticos de repetição de requisição (ARQs)

Canal com Erros

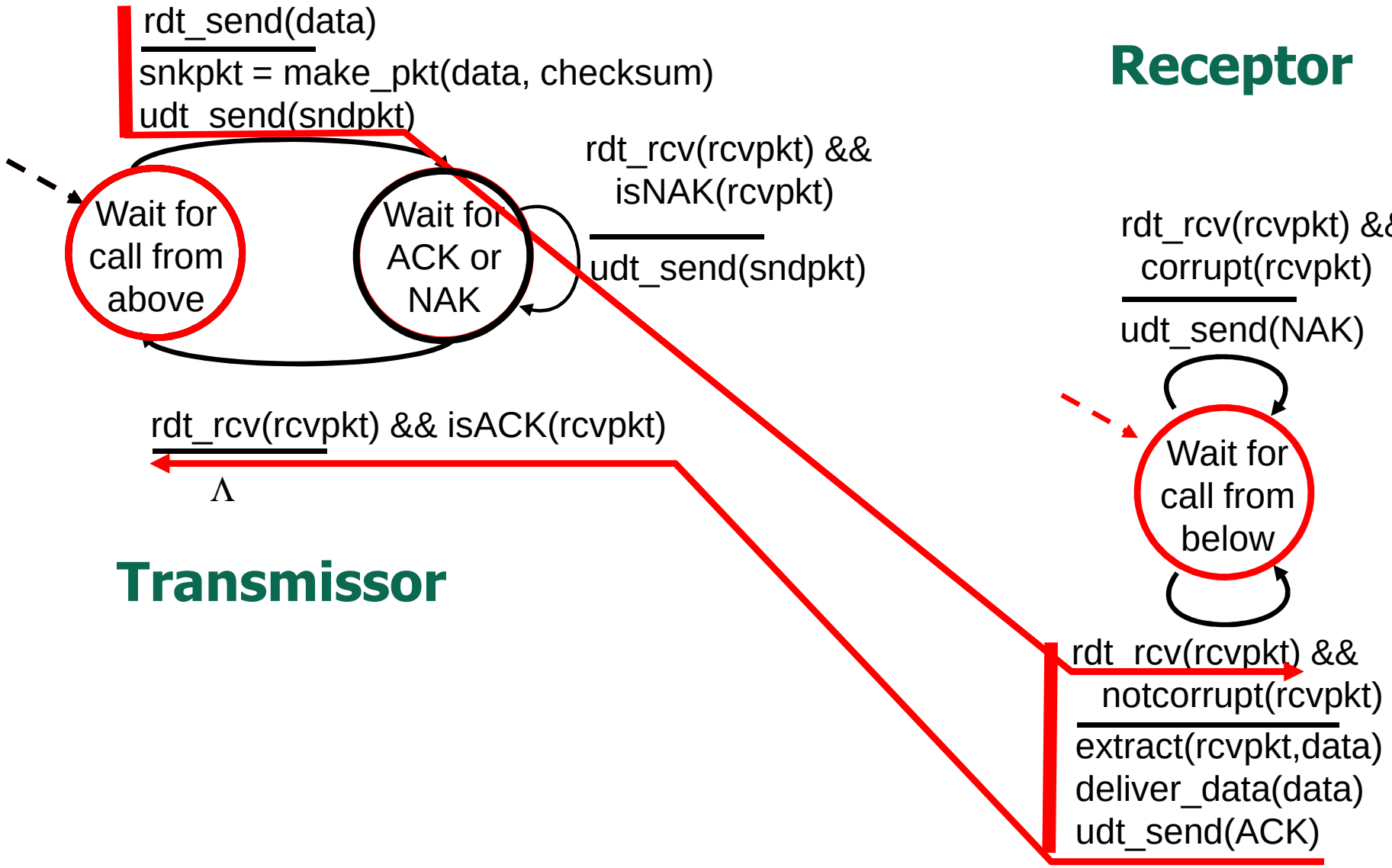


a. rdt2.0: lado remetente

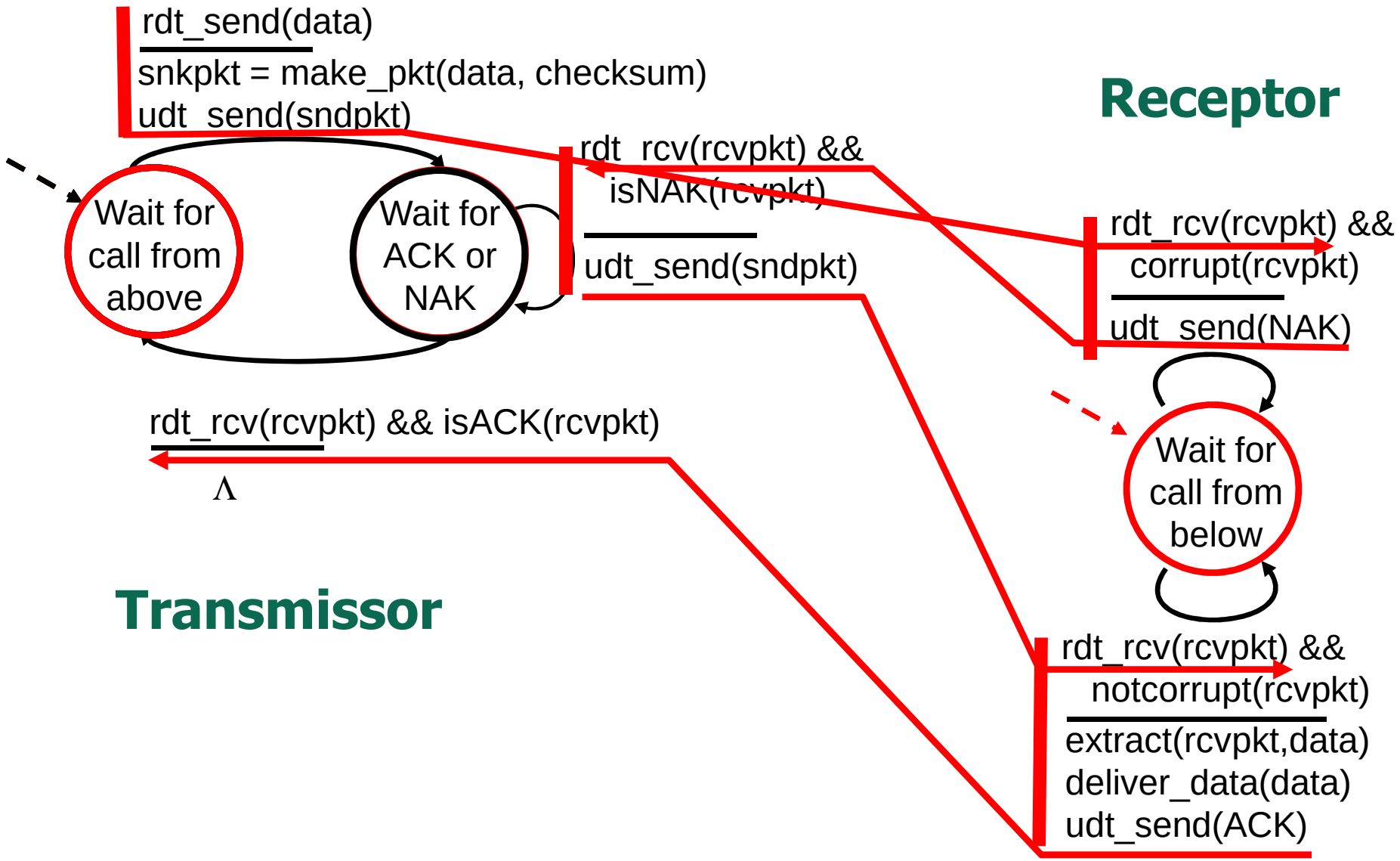


b. rdt2.0: lado destinatário

Canal com Erros: operação normal



Canal com Erros: operação com erro



Canal com Erros: Problema

Instituto de computação

- E se o ACK/NAK for corrompido?
  - Transmissor não sabe o que se passou no receptor
  - Não pode apenas retransmitir
    - Possibilidade de pacotes duplicados
- O que fazer?
  - Remetente usa ACKs/NAKs para cada ACK/NAK do receptor
    - E se perder ACK/NAK do remetente?
  - Retransmitir
    - Mas pode causar retransmissão de pacote recebido certo

Canal com Erros: Problema

Instituto de computação

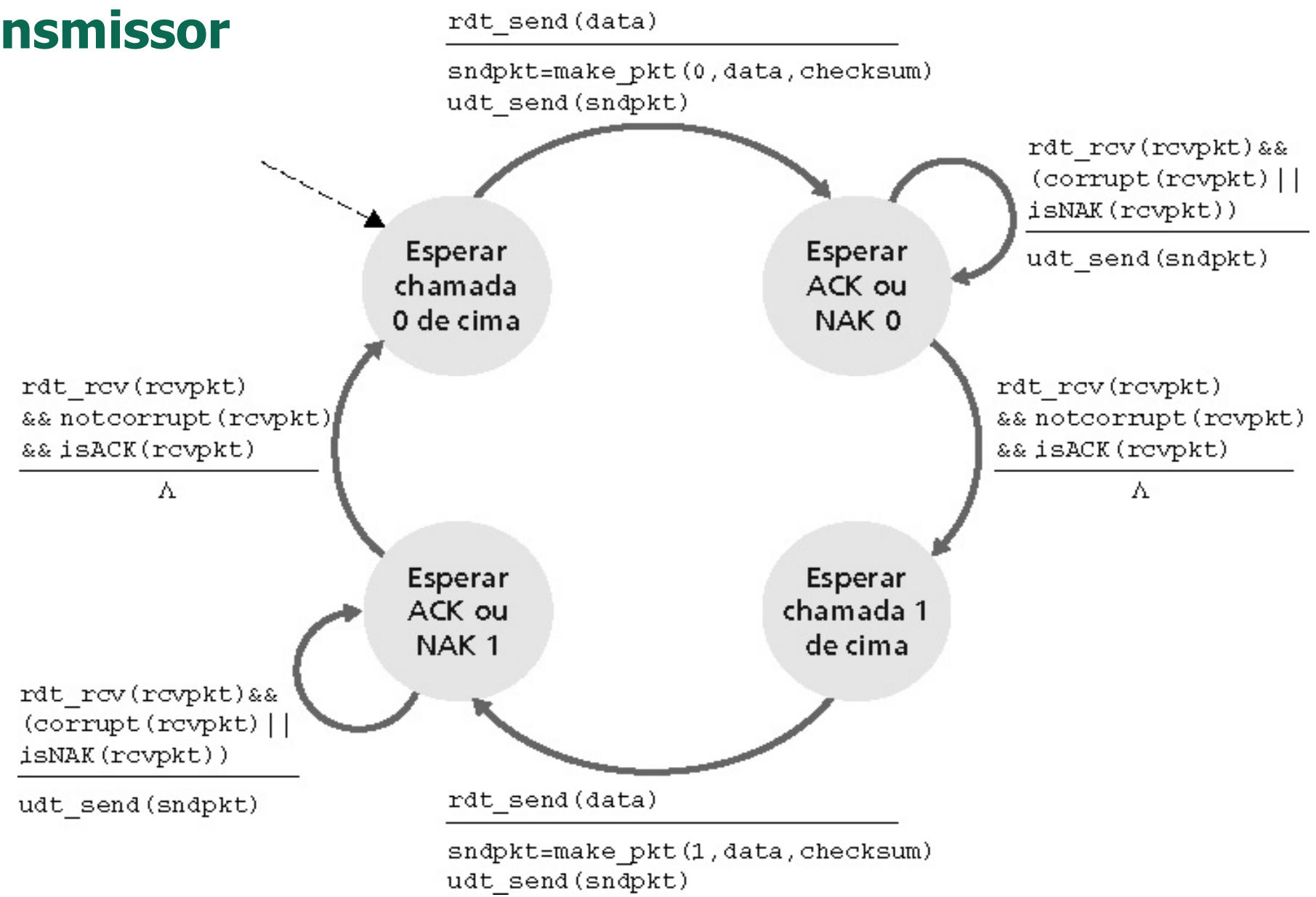
- Transmissor inclui um **número de seqüência** em cada pacote
- Transmissor retransmite o último pacote se ACK/NAK chegar com erro
- Receptor descarta pacotes duplicados
  - Não entrega a aplicação

Transmissor envia um pacote, e então aguarda resposta do receptor

Protocolos para-e-espera (*stop-and-wait*)

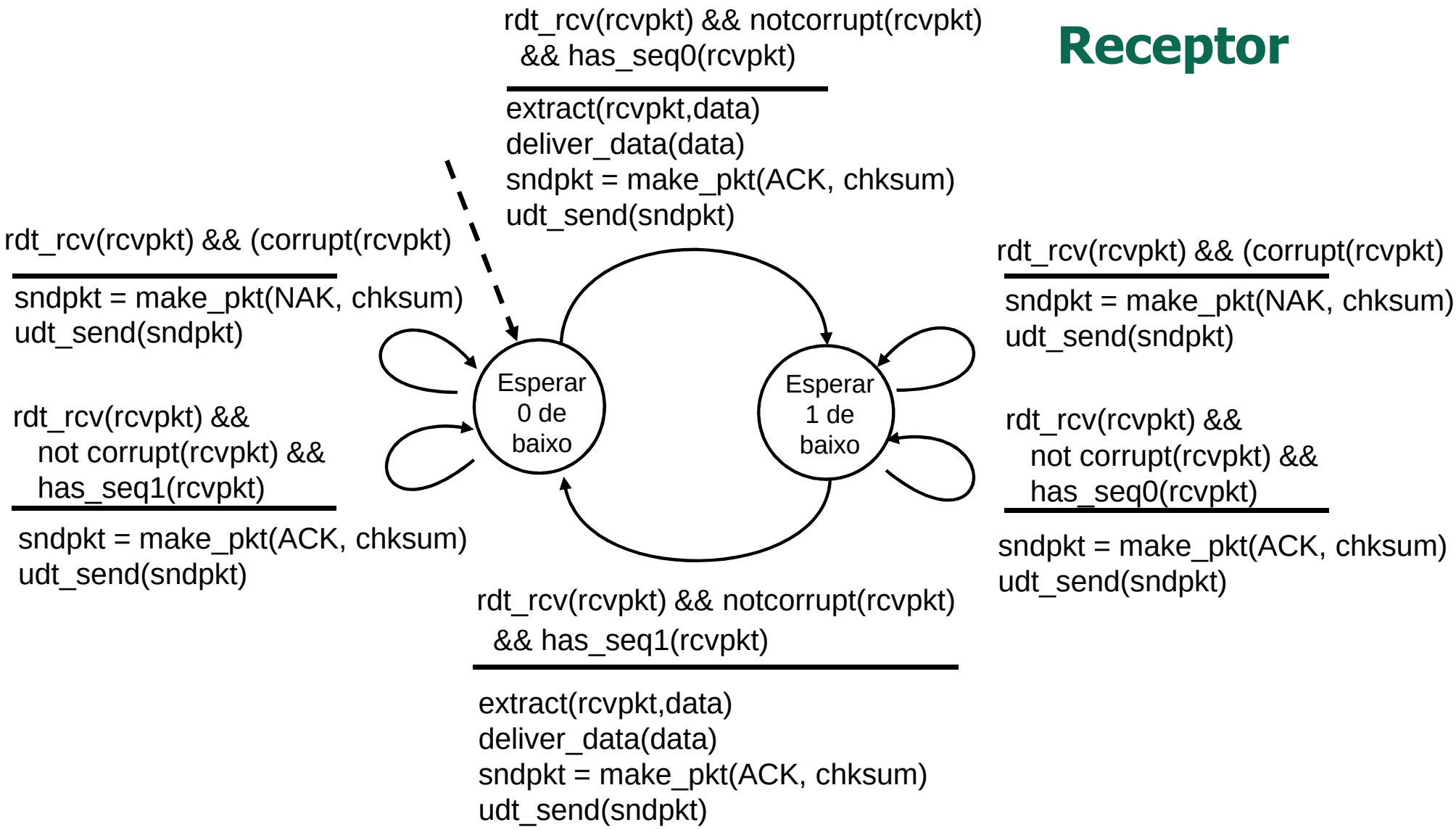
Canal com Erros: ACK/NAK corrompidos

Transmissor



Canal com Erros: ACK/NAK corrompidos

Receptor



Canal com Erros: ACK/NAK corrompidos

Transmissor

- Número de seq. no pacote
- Bastam dois números de sequência (0,1)
- Deve verificar se ACK/NAK recebidos estão corrompidos
- Duplicou o no. de estados
  - estado deve "lembrar" se o número de seq. do pacote atual é 0 ou 1

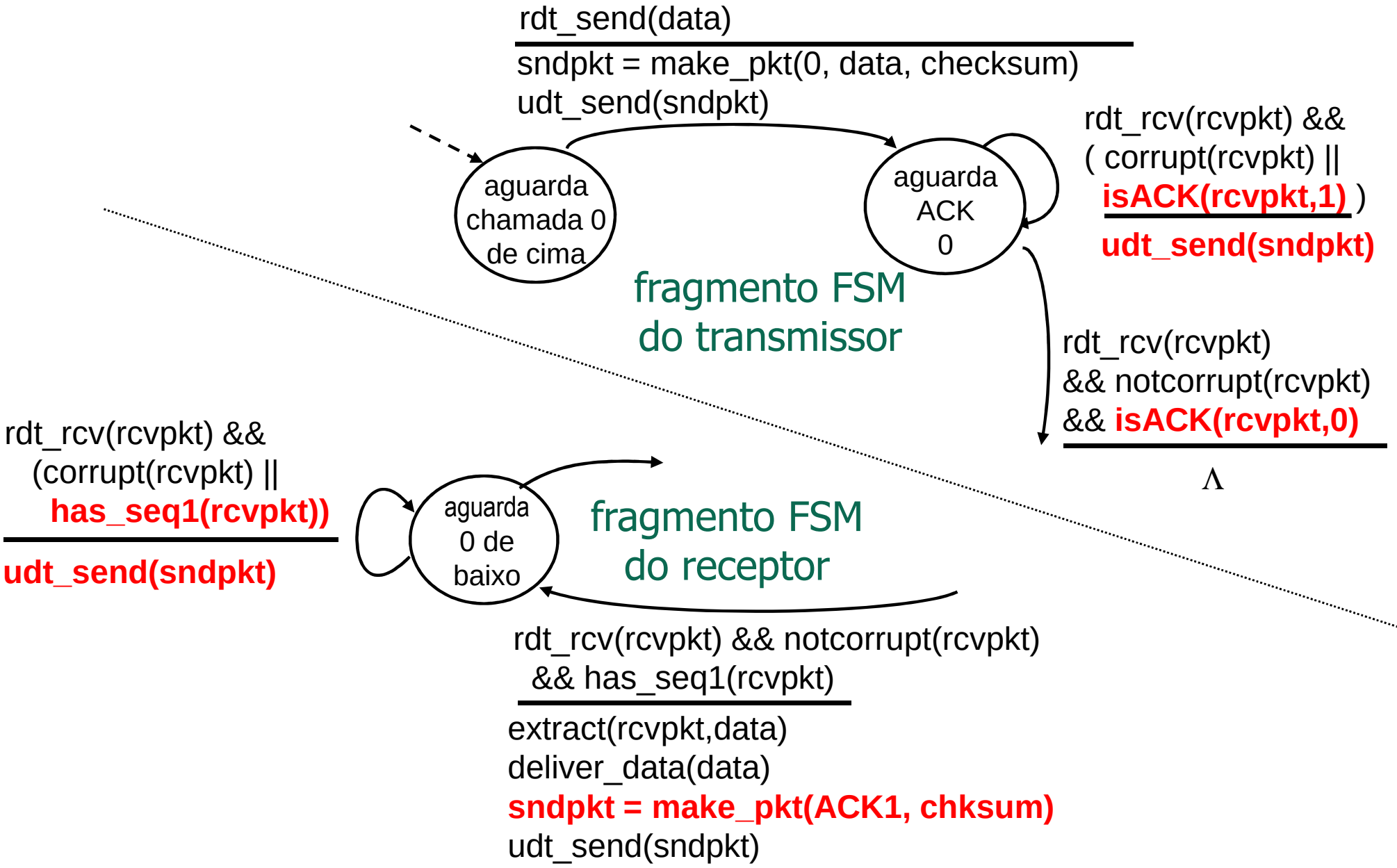
Receptor

- Deve verificar se o pacote recebido é duplicado
  - Estado indica se no. de seq. esperado é 0 ou 1
- Obs.:
  - Receptor não tem como saber se último ACK/NAK foi recebido bem pelo transmissor
    - Não são identificados

Canal com Erros: Sem NAK

- Mesma funcionalidade usando apenas ACKs
- Ao invés de NAK, receptor envia ACK para último pacote recebido sem erro
- Receptor deve incluir explicitamente no. de seq do pacote reconhecido
- **ACKs duplicados** no Transmissor resultam na **mesma ação do NAK**
  - **Retransmissão** do pacote corrente

Canal com Erros: Sem NAK



Canal com Erros e Perdas

Instituto de computação

- Canal de transmissão também pode perder pacotes
  - Dados ou ACKs
- *Checksum*, no. de seq., ACKs, retransmissões podem ajudar, mas não serão suficientes

- Como lidar com perdas?
  - Transmissor **espera** até ter certeza que se perdeu pacote ou ACK, e então retransmite

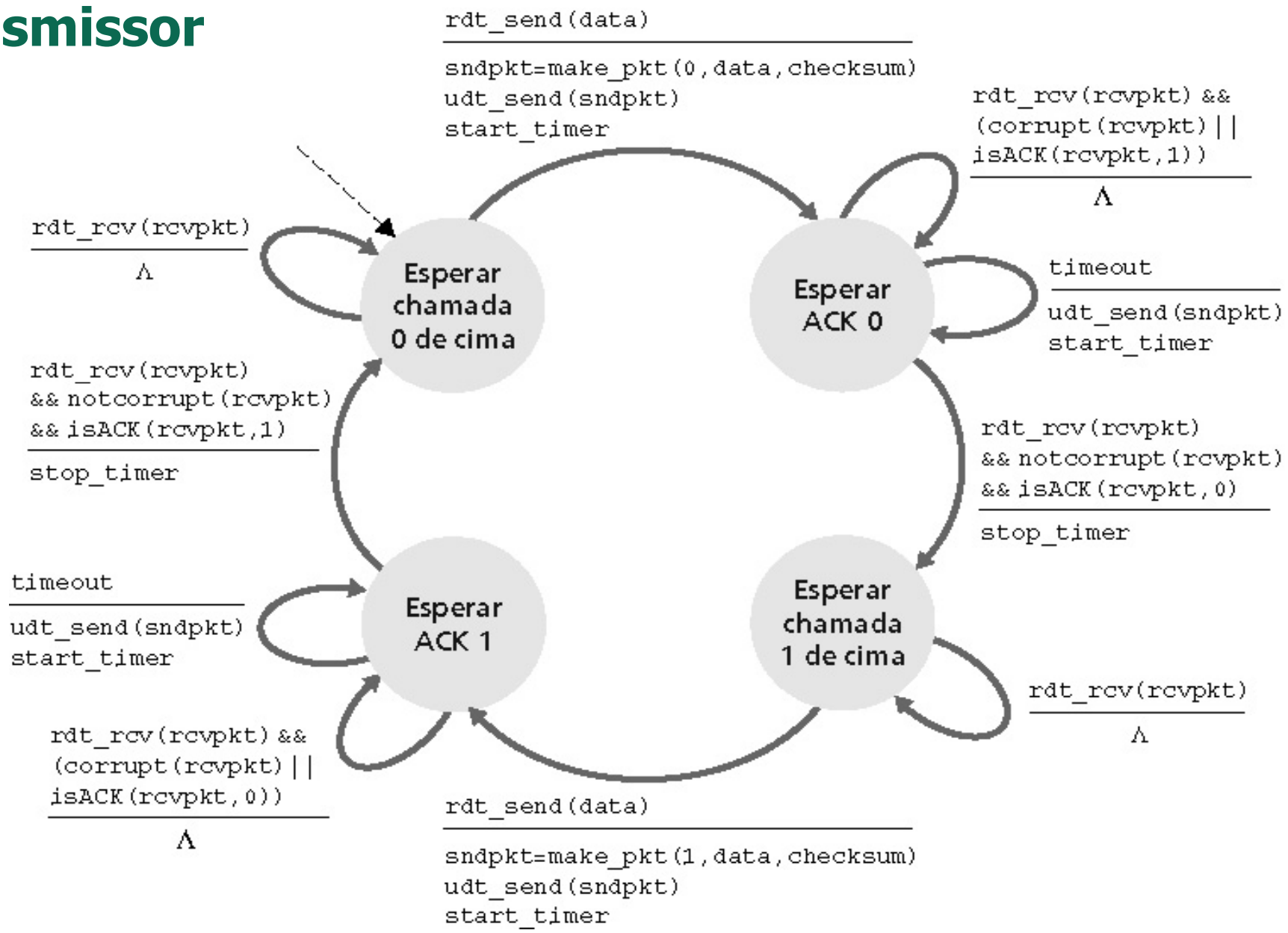
Canal com Erros e Perdas

Instituto de computação

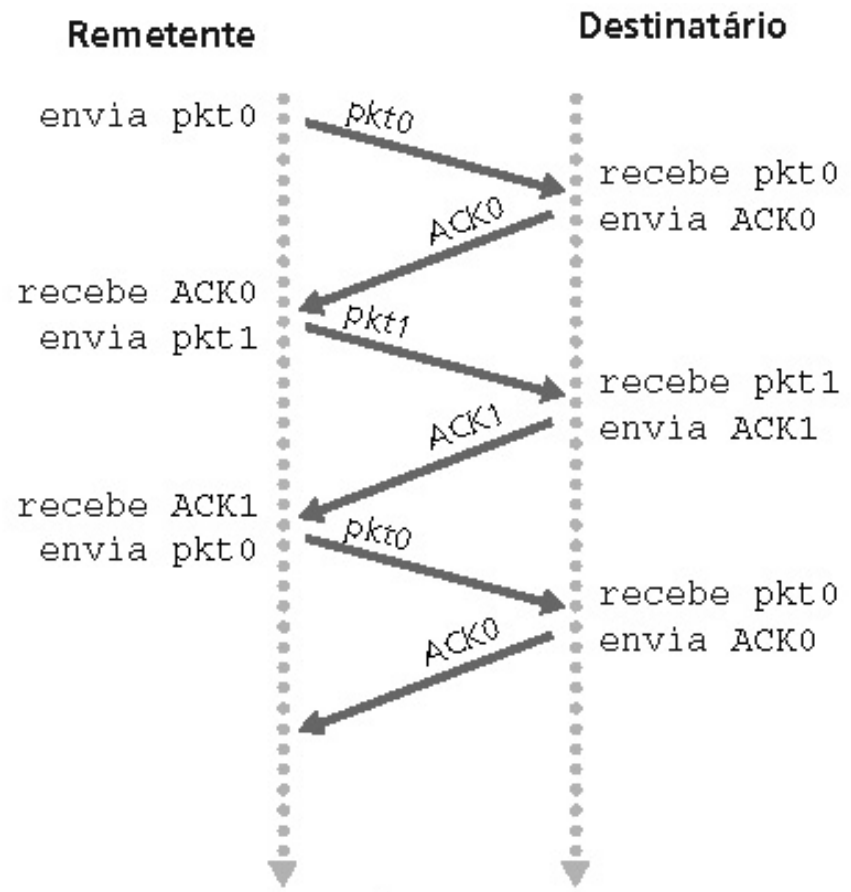
- Transmissor aguarda um tempo "razoável" pelo ACK
  - Retransmite se nenhum ACK for recebido neste intervalo
  - Se pacote (ou ACK) apenas atrasado (e não perdido)
    - Retransmissão será duplicada, mas uso de no. de seq. já cuida disto
    - Receptor deve especificar no. de seq do pacote sendo reconhecido
  - Requer temporizador

Canal com Erros e Perdas

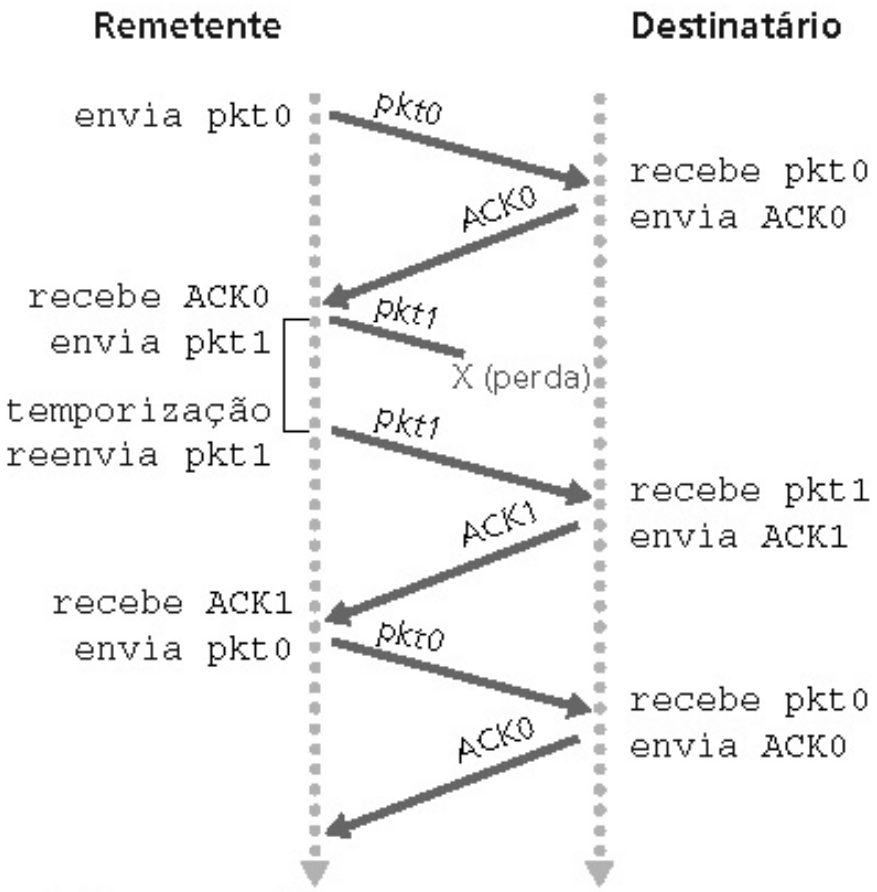
Transmissor



Canal com Erros e Perdas

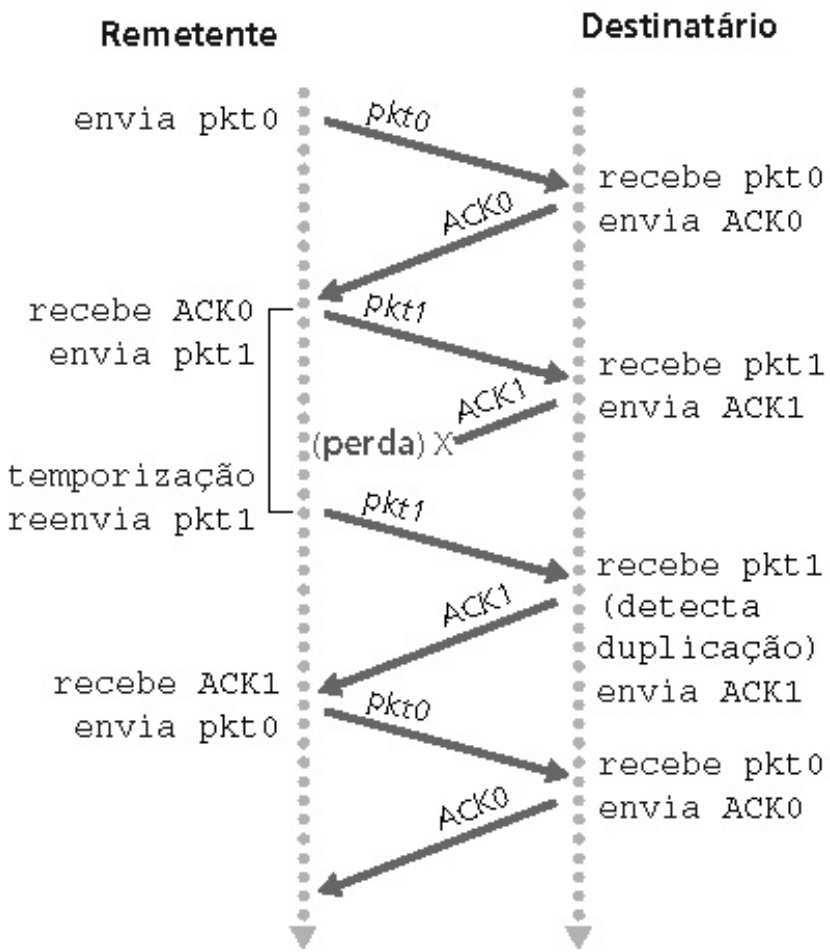


a. Operação sem perda

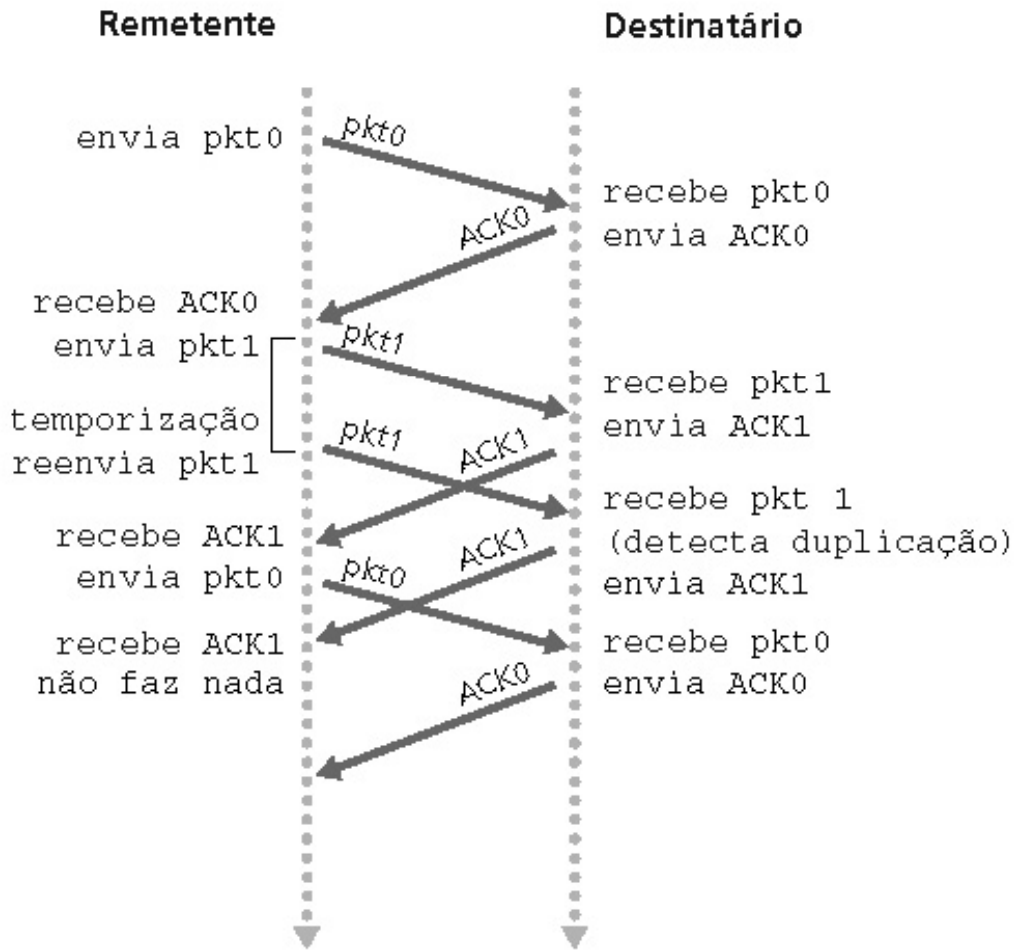


b. Pacote perdido

Canal com Erros e Perdas



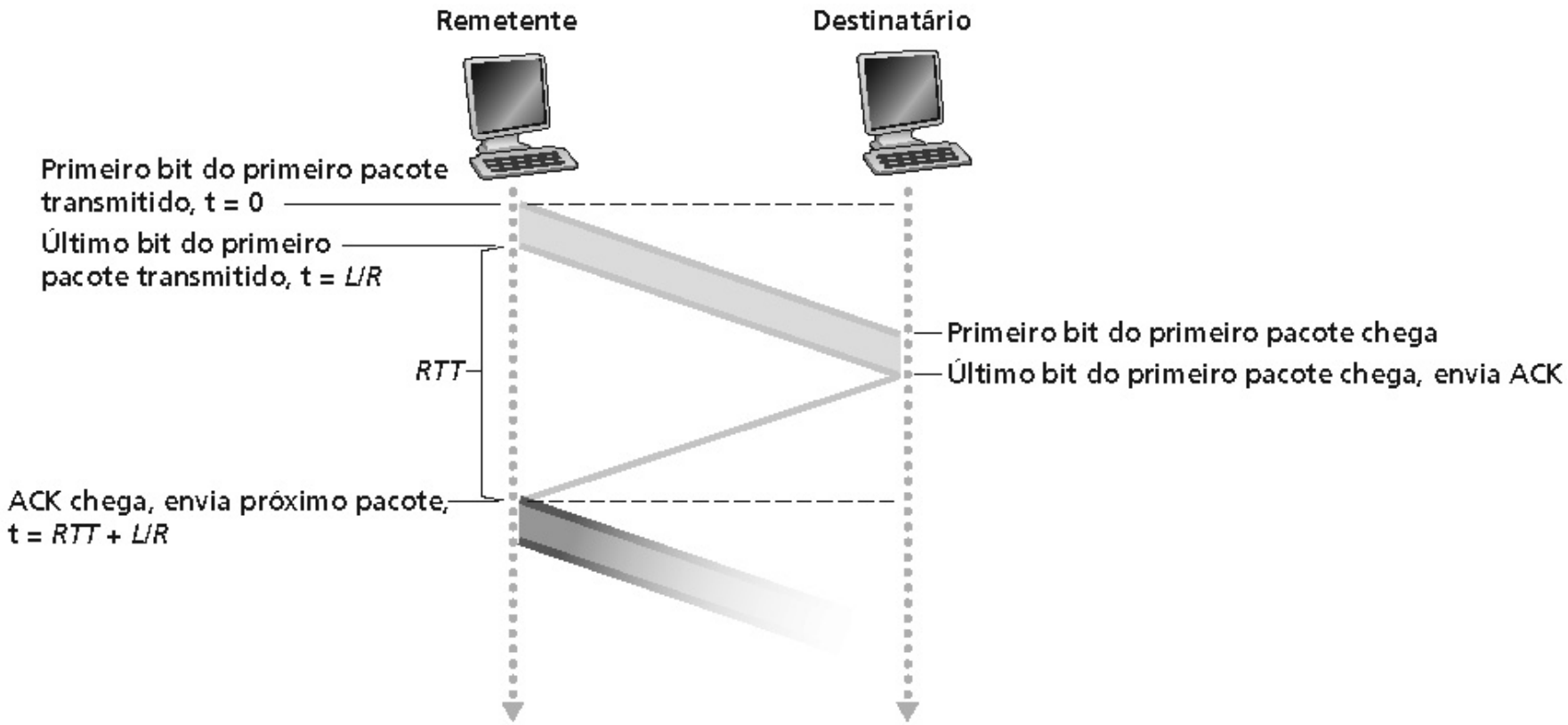
c. ACK perdido



retransmissão prematura

Canal com Erros e Perdas

- Canal confiável, mas **o desempenho é um problema**
  - Operação do pare-e-espere



a. Operação pare e espere

Canal com Erros e Perdas

Instituto de computação

- Canal confiável, mas **o desempenho é um problema**
  - Exemplo: enlace de 1 Gb/s, retardo de 15 ms, pacote de 1KB
  - Função utilidade
    - Fração de **tempo** em que o **emissor** está realmente ocupado **enviando bits**

$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\,\text{bps}} = 8\,\text{microsegundos}$$

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{0.008}{30.008} = 0.00027$$

pacotes de 1KB são enviados a cada 30 ms

vazão de 33kB/s num enlace de 1 Gb/s

Canal com Erros e Perdas

- Canal confiável, mas **o desempenho é um problema**
  - Exemplo: enlace de 1 Gb/s, retardo de 15 ms, pa[illegible]e 1KB
  - Função utilidade
    - Fração de **tempo** em que o **emiss**[illegible]do **enviando bits**

$$d_{trans} = \frac{L}{}$$ [illegible]segundos

$$\frac{R}{RTT + L / R} = \frac{0.008}{30.008} = 0.00027$$

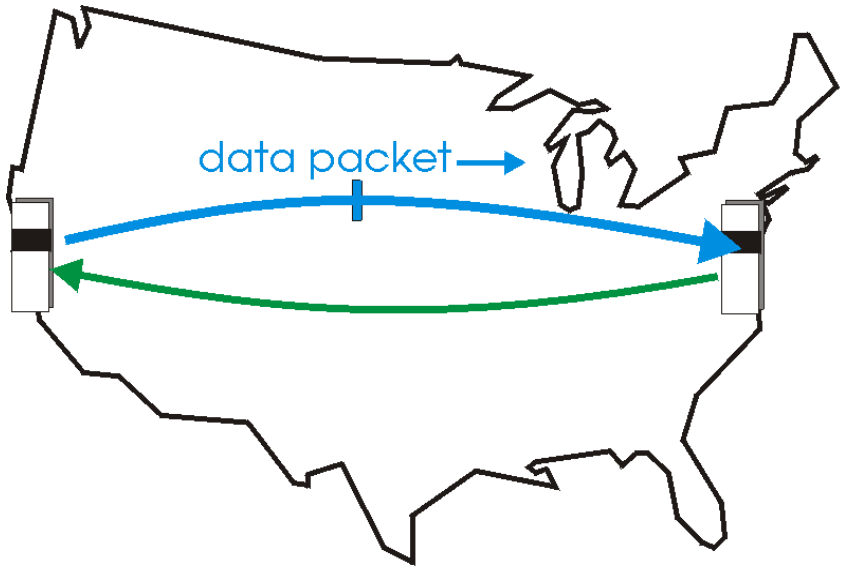
Protocolo limita o uso dos recursos físicos

pacotes de 1KB são enviados a cada 30 ms

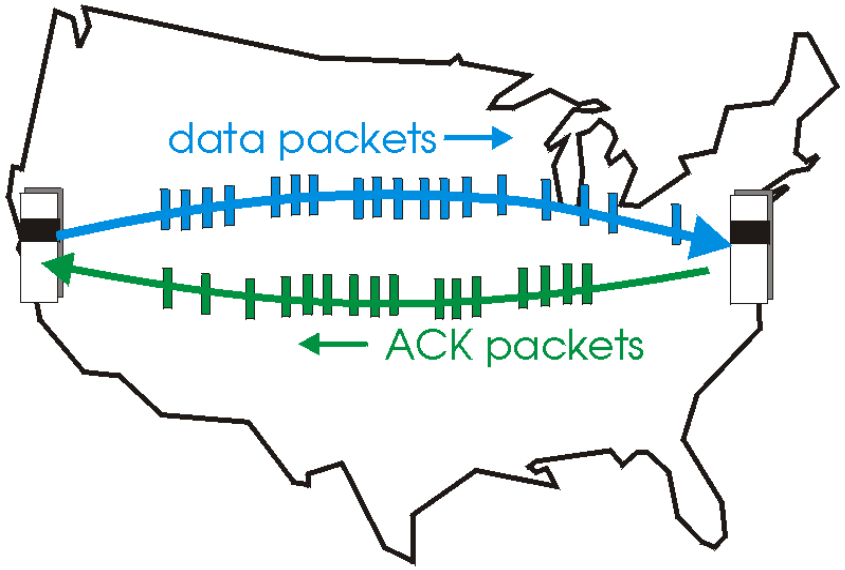
vazão de 33kB/s num enlace de 1 Gb/s

Paralelismo (*pipelining*)

- Transmissor envia vários pacotes em seqüência
  - **Todos** esperando para serem **reconhecidos**
- Faixa de números de seqüência deve ser aumentada
- Armazenamento no Transmissor e/ou no receptor



(a) operação do protocolo pare e espere



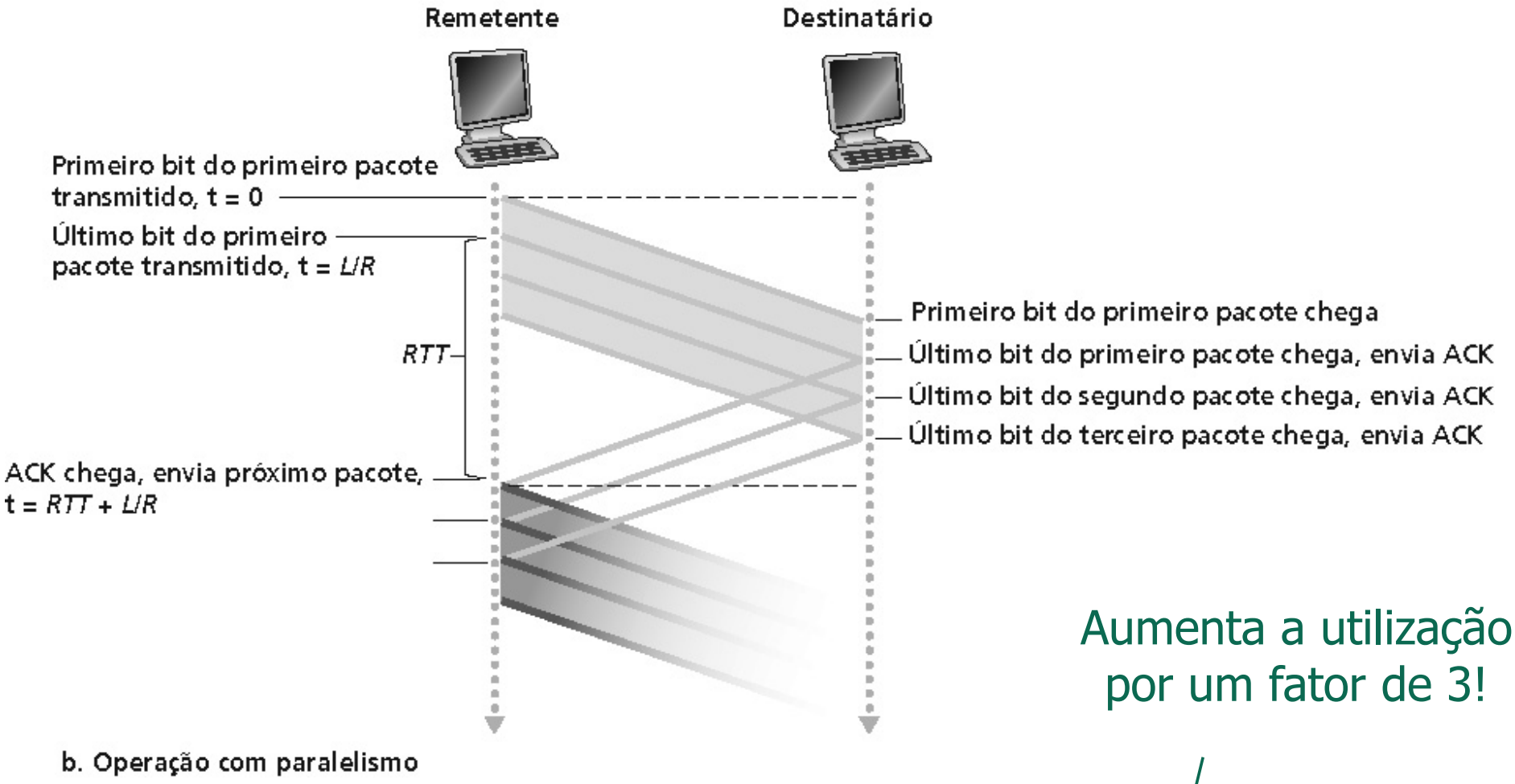
(a) operação do protocolo com paralelismo

Paralelismo (*pipelining*)

Instituto de computação

- Duas formas genéricas
  - *Go-back-N*
  - Retransmissão seletiva

Paralelismo (*pipelining*)



b. Operação com paralelismo

Aumenta a utilização por um fator de 3!

$$U_{\text{tx}} = \frac{3 \times L/R}{\text{RTT} + L/R} = \frac{0{,}024}{30{,}008} = 0{,}0008$$

Protocolos com Paralelismo

Instituto de computação

Go-back-N

- O transmissor pode ter até N pacotes não reconhecidos em trânsito
- Receptor envia apenas ACKs cumulativos
  - Reconhece o "passado"
  - Não reconhece pacote se houver falha de seq.
- Transmissor possui um temporizador para o pacote mais antigo ainda não reconhecido
  - Se o temporizador estourar, retransmite todos os pacotes ainda não reconhecidos

Retransmissão seletiva

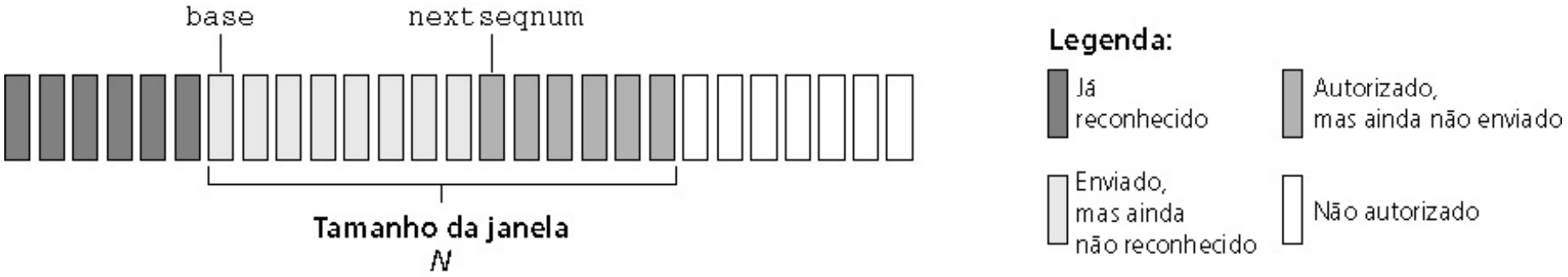
- O transmissor pode ter até N pacotes não reconhecidos em trânsito
- Receptor reconhece pacotes individuais
- Transmissor possui um temporizador para cada pacote ainda não reconhecido
  - Se o temporizador estourar, retransmite apenas o pacote correspondente

Go-back-N (GBN)

Instituto de computação

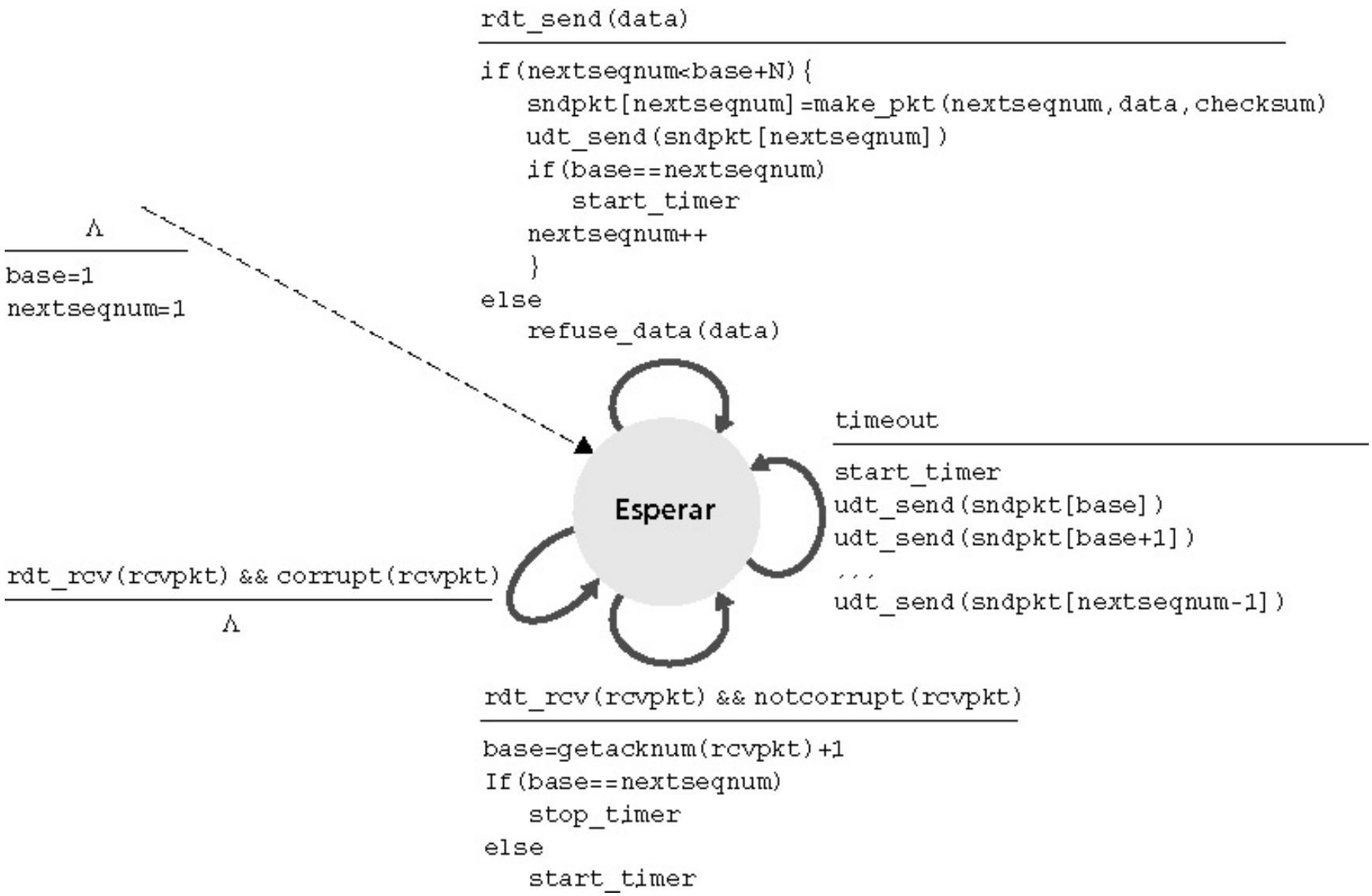
Transmissor

- no. de seq. de k-bits no cabeçalho do pacote
- admite "janela" de até N pacotes consecutivos não reconhecidos

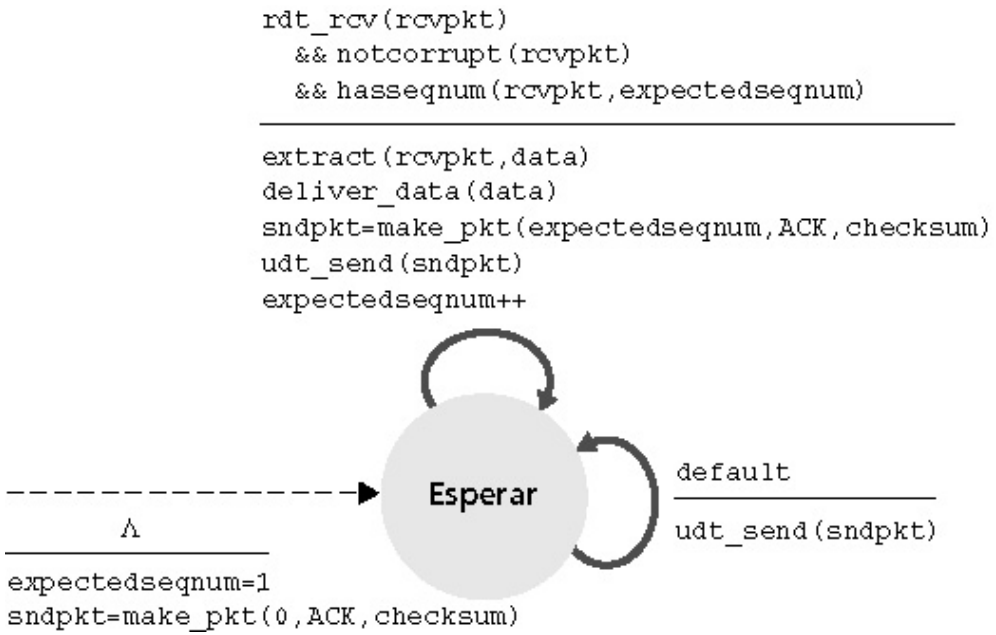


- ACK(n): reconhece todos pacotes, até e inclusive no. de seq n - "ACK cumulativo"
  - Pode receber ACKs duplicados
- Temporizador para cada pacote enviado e não confirmado
- *timeout(n):* retransmite pacote n e todos os pacotes com no. de seq maiores que estejam dentro da janela

GBN: FSM estendida para o transmissor



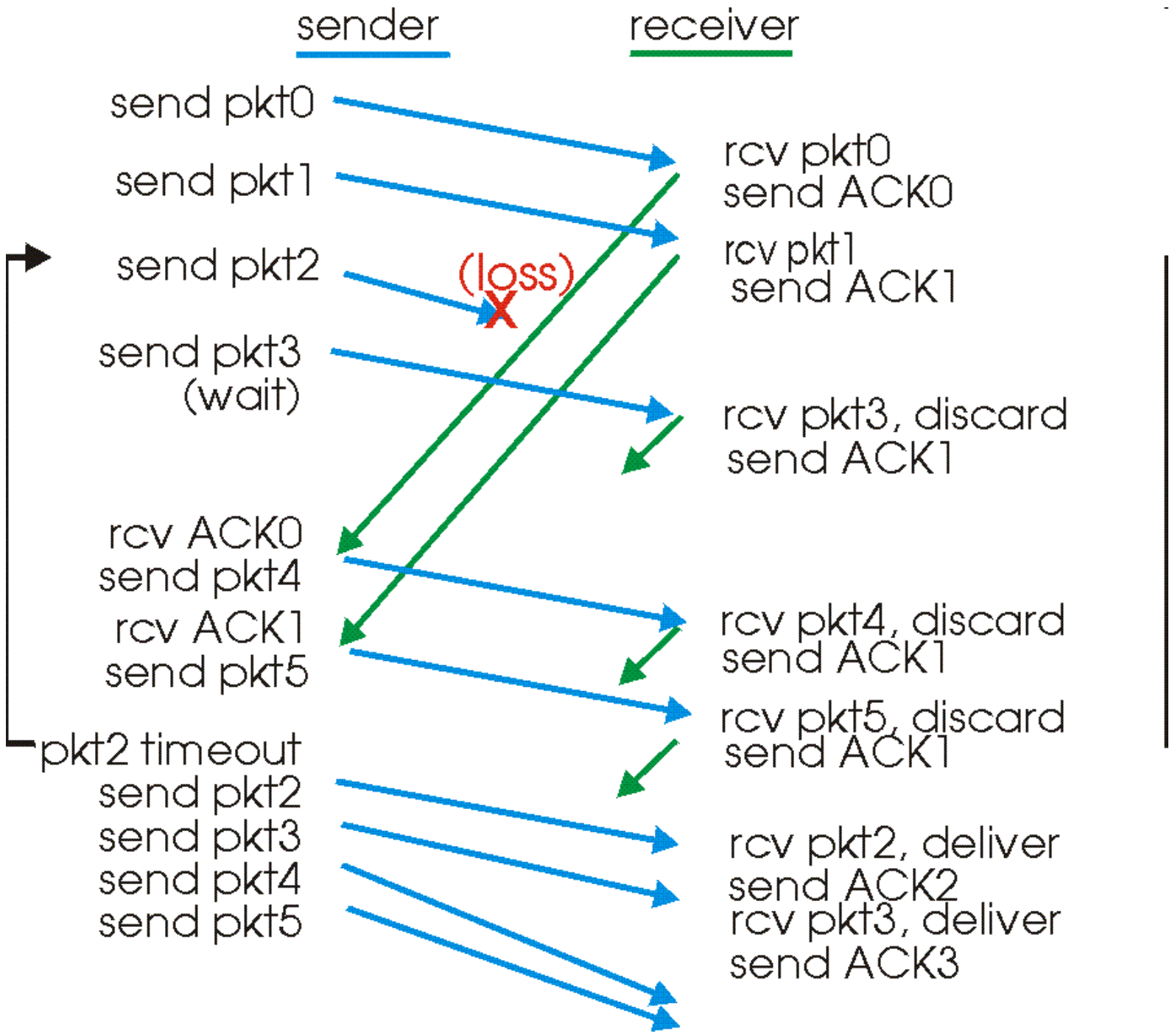
GBN: FSM estendida para o receptor



Receptor simples

- Usa apenas ACK
  - Sempre envia ACK para pacote recebido corretamente com o maior no. de seq. **em ordem**
  - Pode gerar ACKs duplicados
- Pacotes fora de ordem
  - Descarta (não armazena)
    - **Receptor não usa *buffers***
  - Reconhece pacote com o mais alto número de seqüência em ordem

GBN em ação

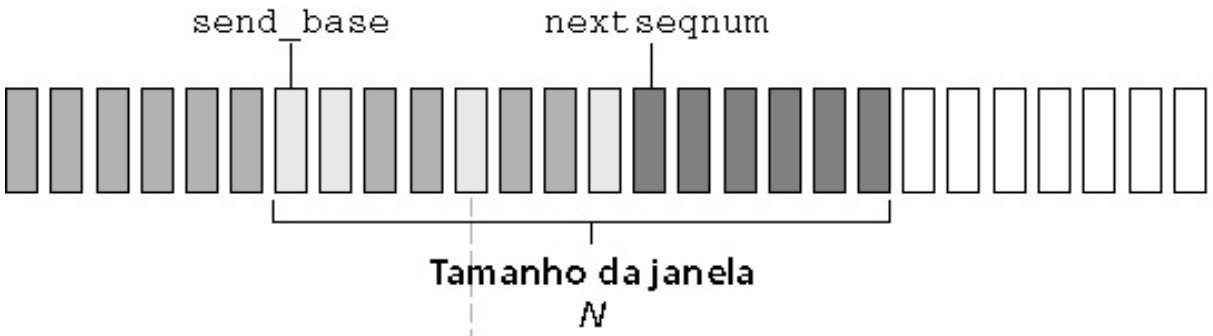

sender
receiver
send pkt0
send pkt1
send pkt2
(loss)
send pkt3
(wait)
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv ACK0
send pkt4
rcv ACK1
send pkt5
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3

Retransmissão Seletiva

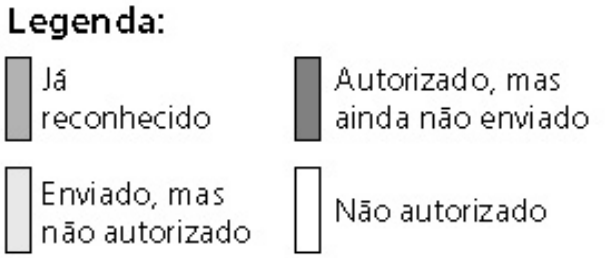
Instituto de computação

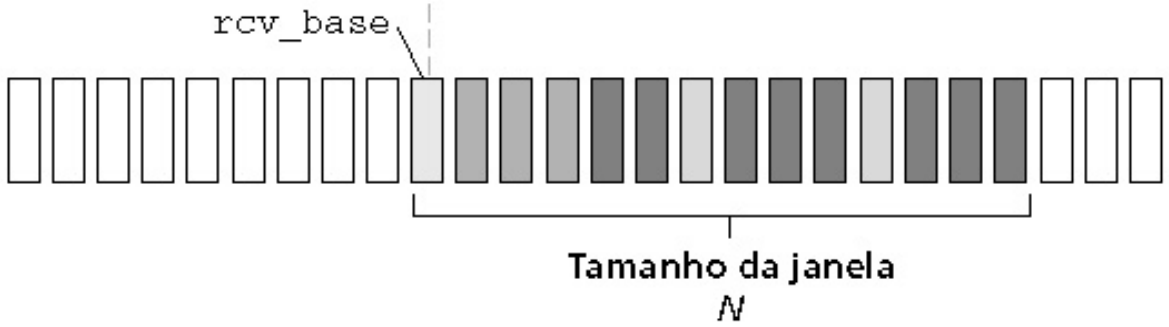
- Receptor reconhece individualmente todos os pacotes recebidos corretamente
  - Armazena pacotes no *buffer*, conforme necessário, para posterior entrega ordenada à camada superior
- Transmissor apenas reenvia pacotes para os quais um ACK não foi recebido
  - Temporizador de remetente para cada pacote sem ACK
- Janela do transmissão
  - *N* números de seqüência consecutivos
  - Outra vez limita números de seqüência de pacotes enviados, mas ainda não reconhecidos

Retransmissão Seletiva

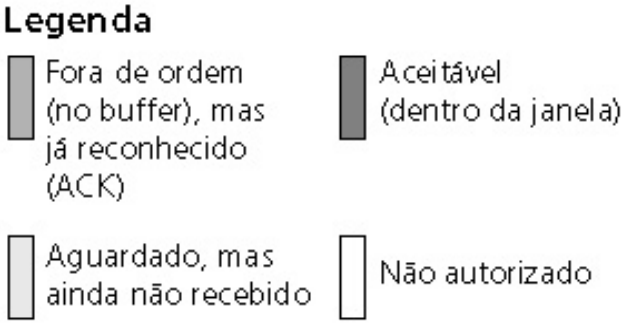


a. Visão que o remetente tem dos números de seqüência

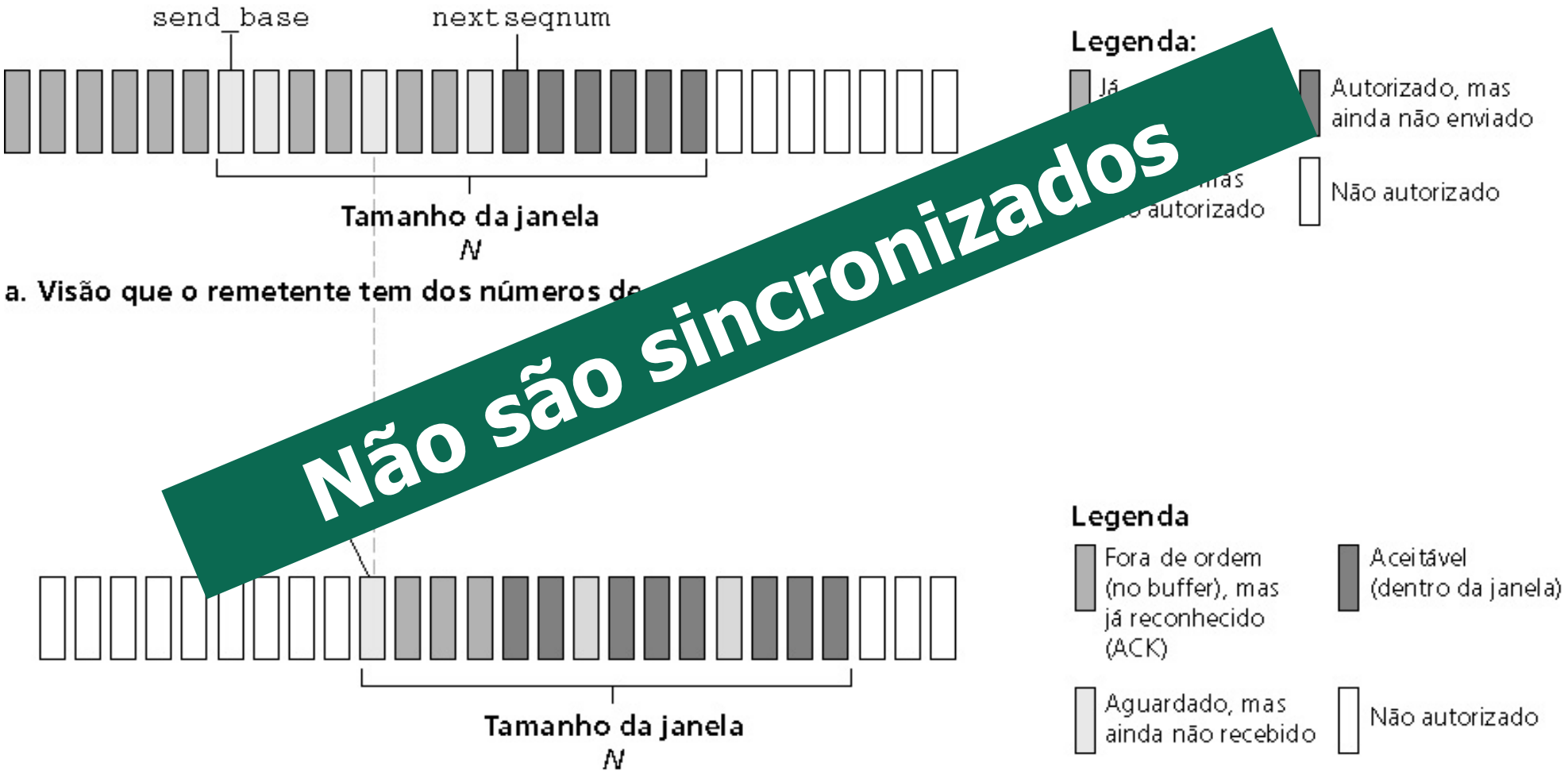




b. Visão que o destinatário tem dos números de seqüência



Retransmissão Seletiva



Retransmissão Seletiva

transmissor

Dados de cima

- Se próx. no. de seq (n) disponível está na janela, envia o pacote e liga temporizador(n)

Estouro do temporizador(n):

- reenvia pacote n, reinicia temporizador(n)

ACK(n) em [sendbase,nextseqnum-1]:

- marca pacote n "recebido"
- se n for menor pacote não reconhecido, avança base da janela ao próx. no. de seq não reconhecido

receptor

Pacote n em [rcvbase, rcvbase+N-1]

- Envia ACK(n)
- Fora de ordem: armazena
- Em ordem: entrega (tb. entrega pacotes armazenados em ordem), Avança janela p/ próxima pacote ainda não recebido

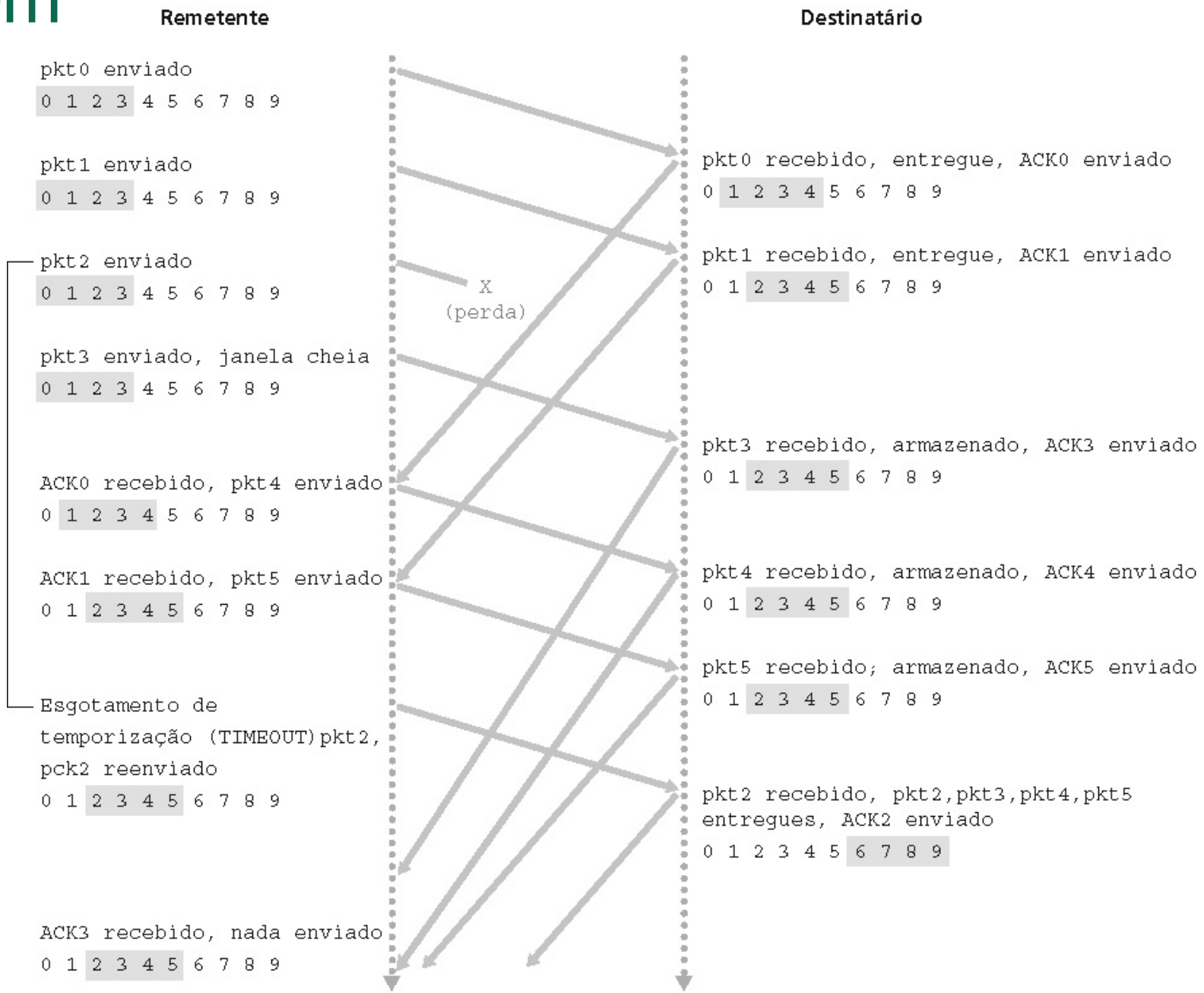
Pacote n em [rcvbase-N,rcvbase-1]

- ACK(n)

Senão:

- Ignora

Retransmissão seletiva em ação



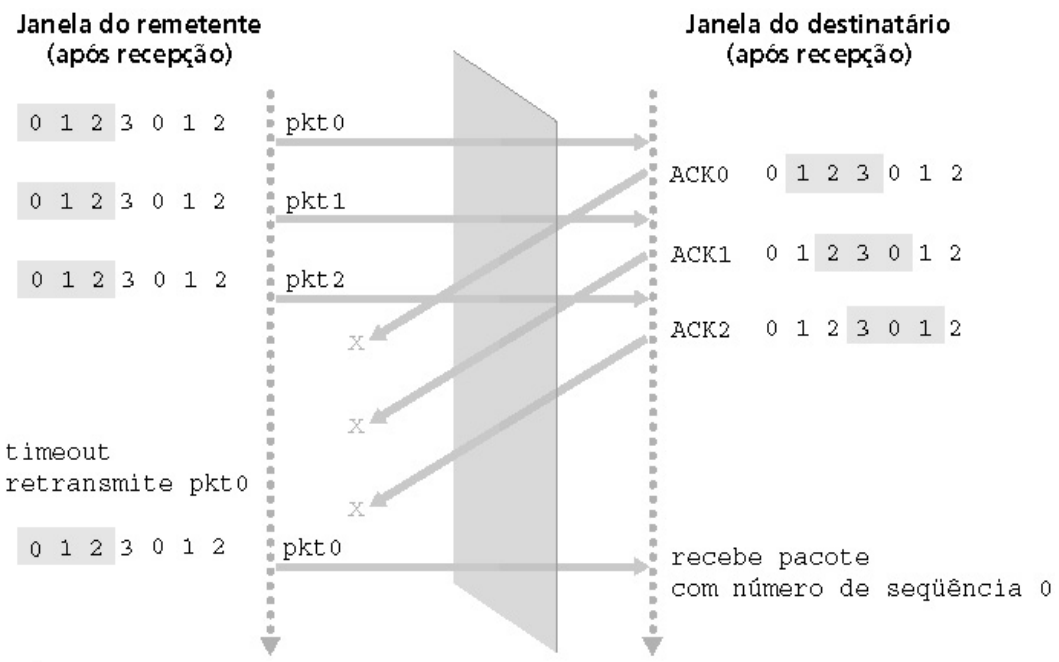
Retransmissão seletiva: dilema

Exemplo:

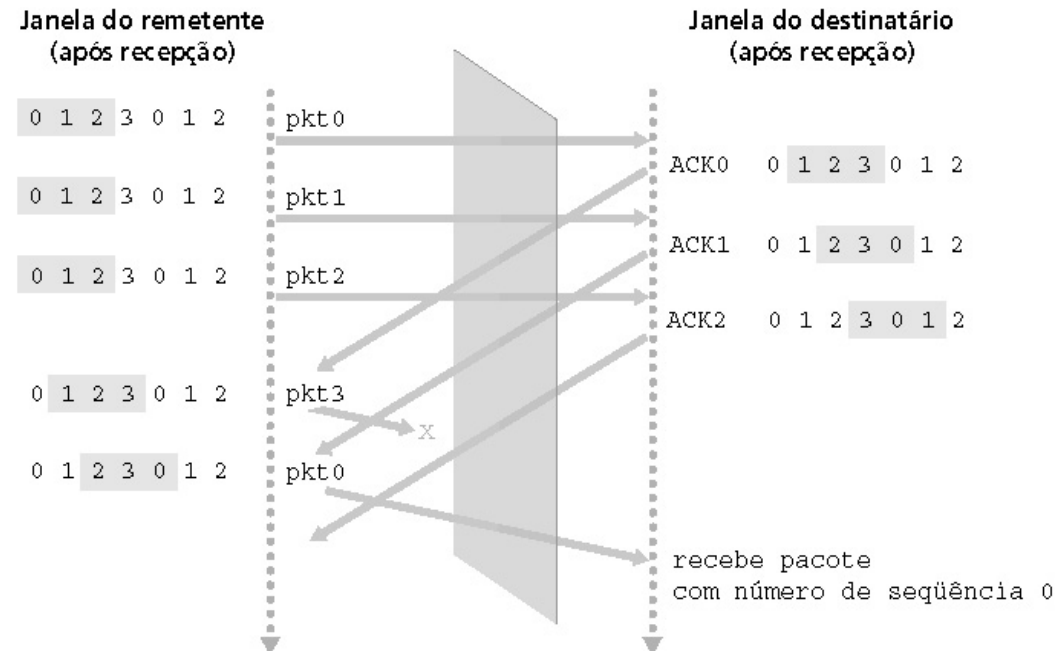
- nos. de seq : 0, 1, 2, 3
- tam. de janela =3

- receptor não vê diferença entre os dois cenários!
- incorretamente passa dados duplicados como novos em (a)

qual a relação entre tamanho de no. de seq e tamanho de janela?



a.



Instituto de computação

Transmission Control Protocol (TCP)

TCP

Instituto de computação

- Muito mais complexo que o UDP
  - UDP: RFC 768
  - TCP: RFCs 793, 1122, 1323, 2018 e 2581

- Orientado à conexão
  - Antes do início da transmissão existem um *handshake*
    - Dois processos trocam segmentos para definir parâmetros
  - É uma conexão lógica
    - Diferente da estabelecida na comutação de circuitos
      - Não há um caminho definido e nem reserva de recursos
    - "Só existe" nos sistemas finais
      - Elementos intermediários não armazenam nenhum estado

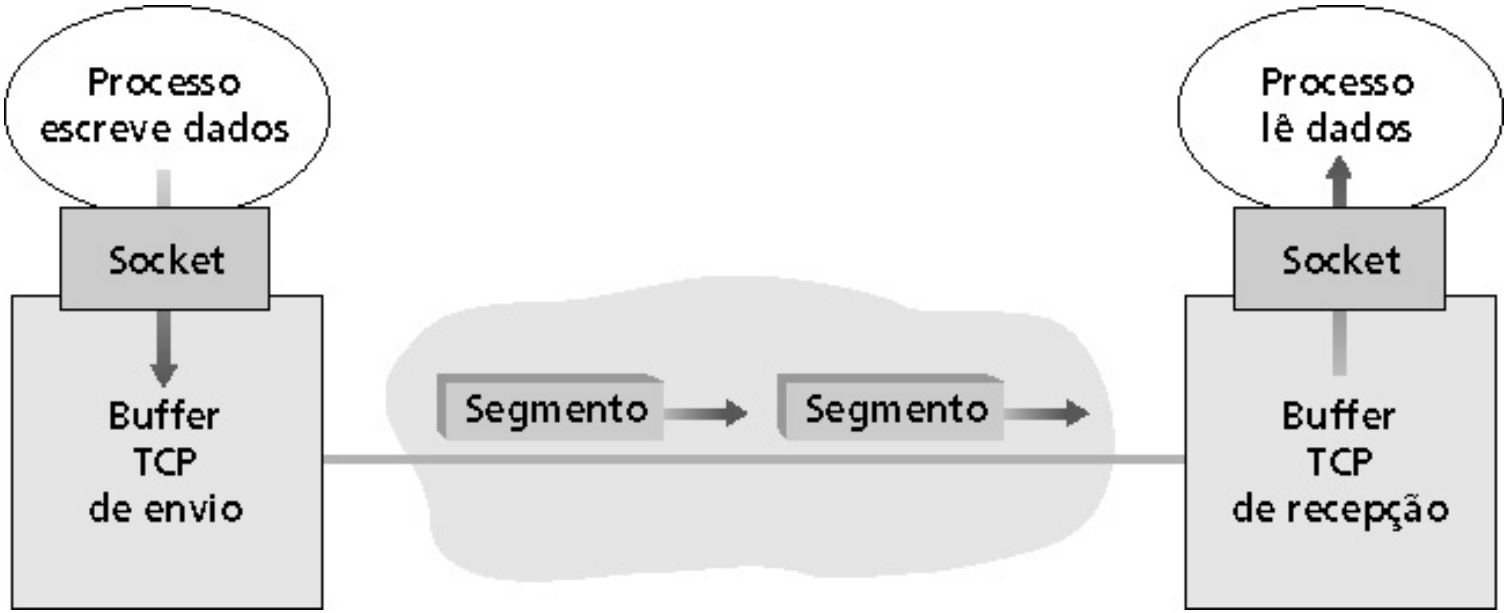
TCP

Instituto de computação

- É ponto-a-ponto
  - Um transmissor e um receptor
- Transmissão *full duplex*
  - Fluxo de dados bidirecional na mesma conexão
  - MSS: tamanho máximo de segmento
- Controle de fluxo
  - Receptor não será afogado pelo transmissor
- Controle de congestionamento
  - Evitar a saturação dos enlaces da rede

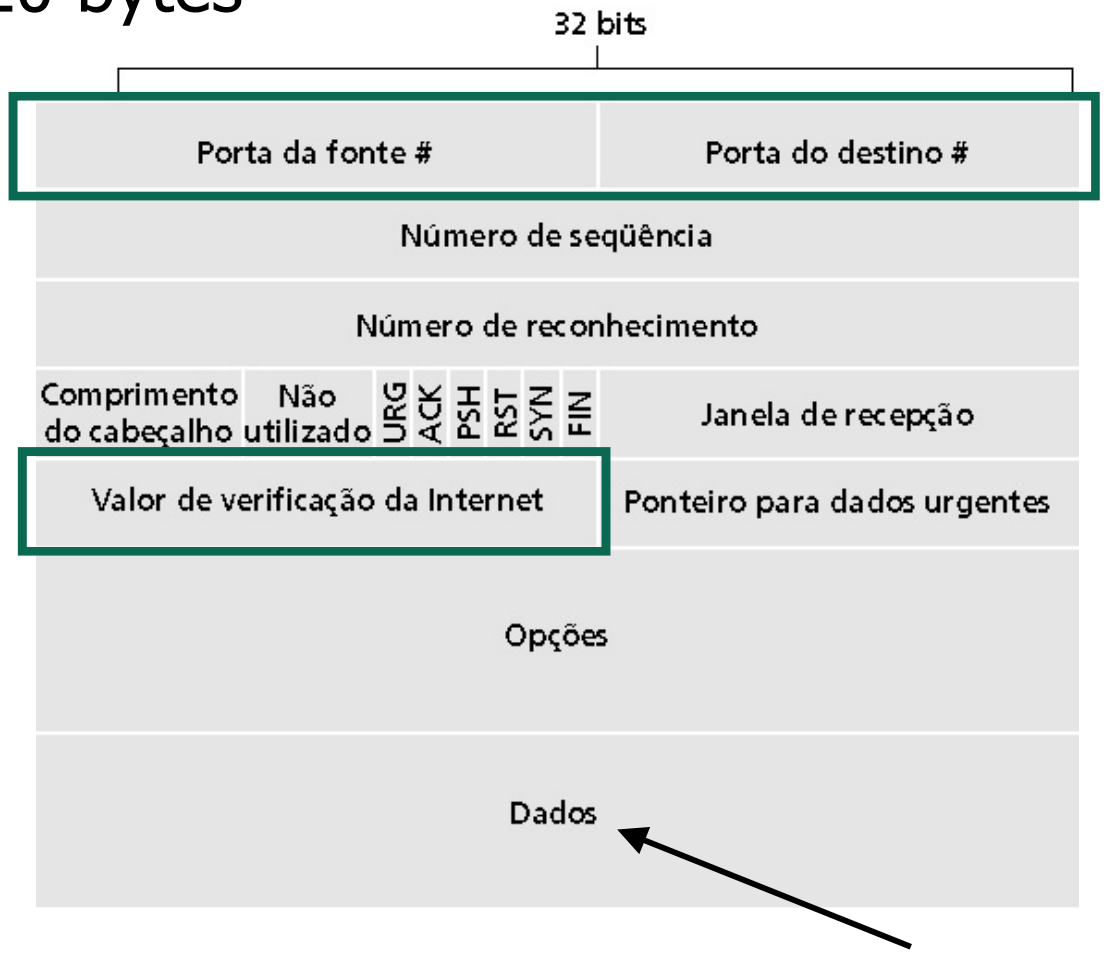
TCP

- *Buffers*
  - Transmissão e recepção
  - Tamanho definido durante a conexão



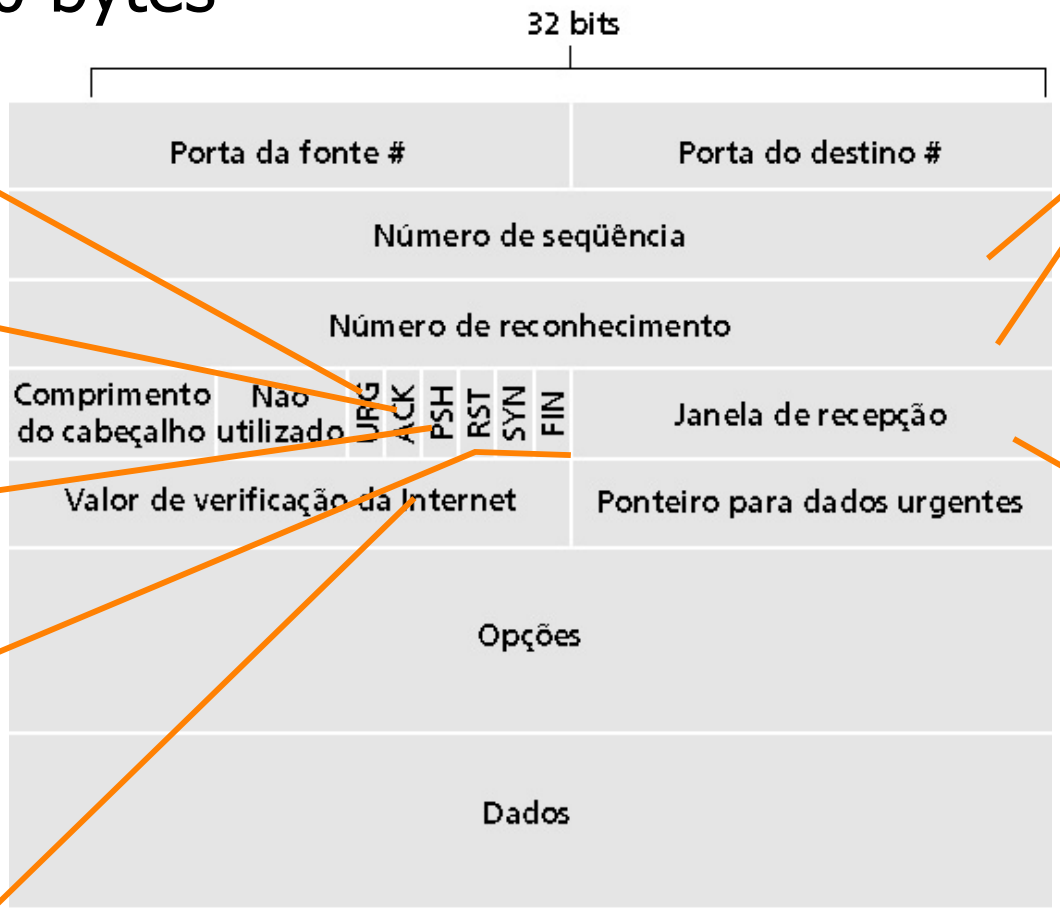
Segmento TCP

- Cabeçalho: 20 bytes



Segmento TCP

- Cabeçalho: 20 bytes



URG: dados urgentes (pouco usado)

ACK: campo de ACK é válido

PSH: envio imediato para a aplicação

RST, SYN, FIN: estabelec. de conexão (comandos de criação e término)

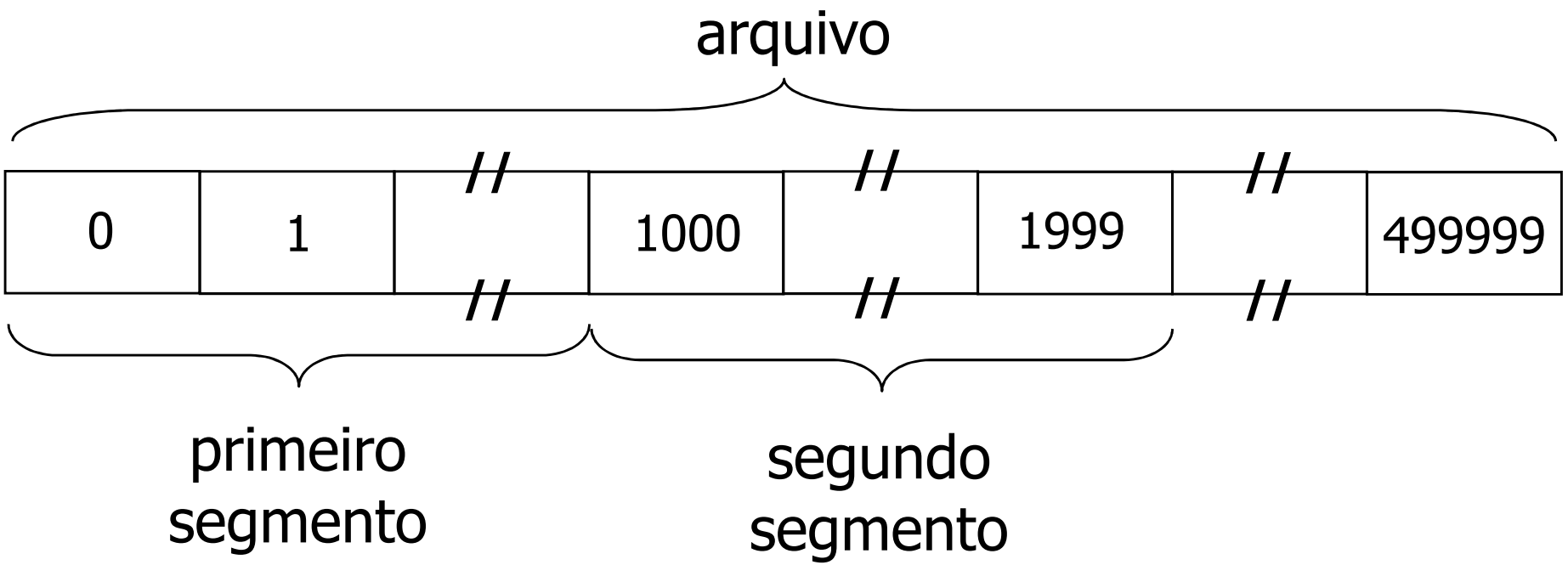
Internet checksum (como no UDP)

contagem por bytes de dados (não segmentos!)

número de bytes receptor está pronto para aceitar

Números de Sequência e ACKs

- Fundamentais para a transferência confiável
- Para o TCP, dados são um **fluxo** de bytes ordenados
  - Ex.: arquivo com 500 kB e MSS 1 kB



Números de Sequência e ACKs

Instituto de computação

- Número de sequência
  - Baseado no número de bytes e não no de segmentos
  - "Número" dentro do fluxo de bytes do primeiro byte de dados do segmento
  - Ex.: fluxo de dados com 500 kB e MSS 1 kB
    - 500 segmentos de 1000 bytes
    - Primeiro segmento: # seq → 0
    - Segundo segmento: # seq → 1000
    - Terceiro segmento: # seq → 2000
    - Etc.

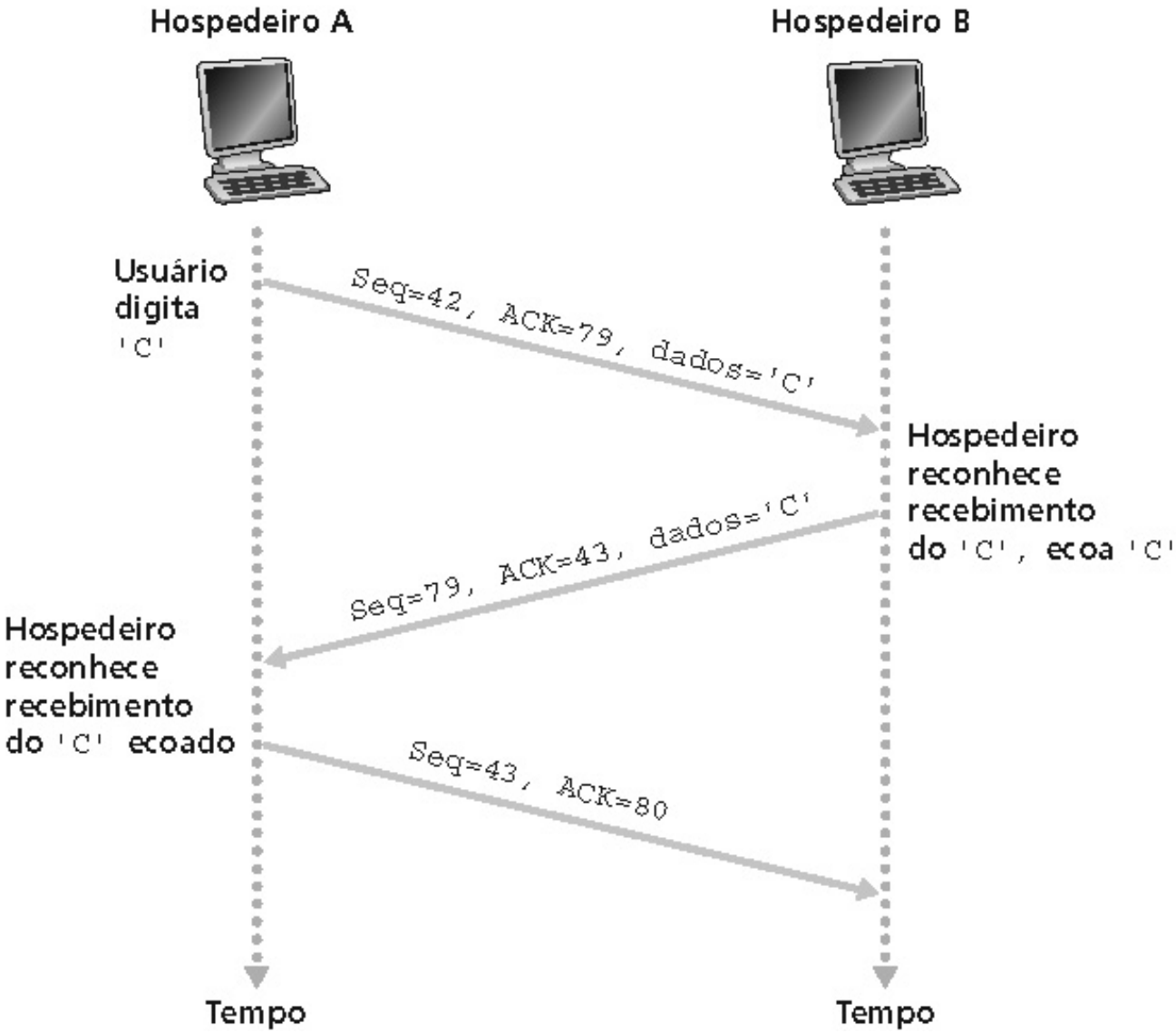
Números de Sequência e ACKs

Instituto de computação

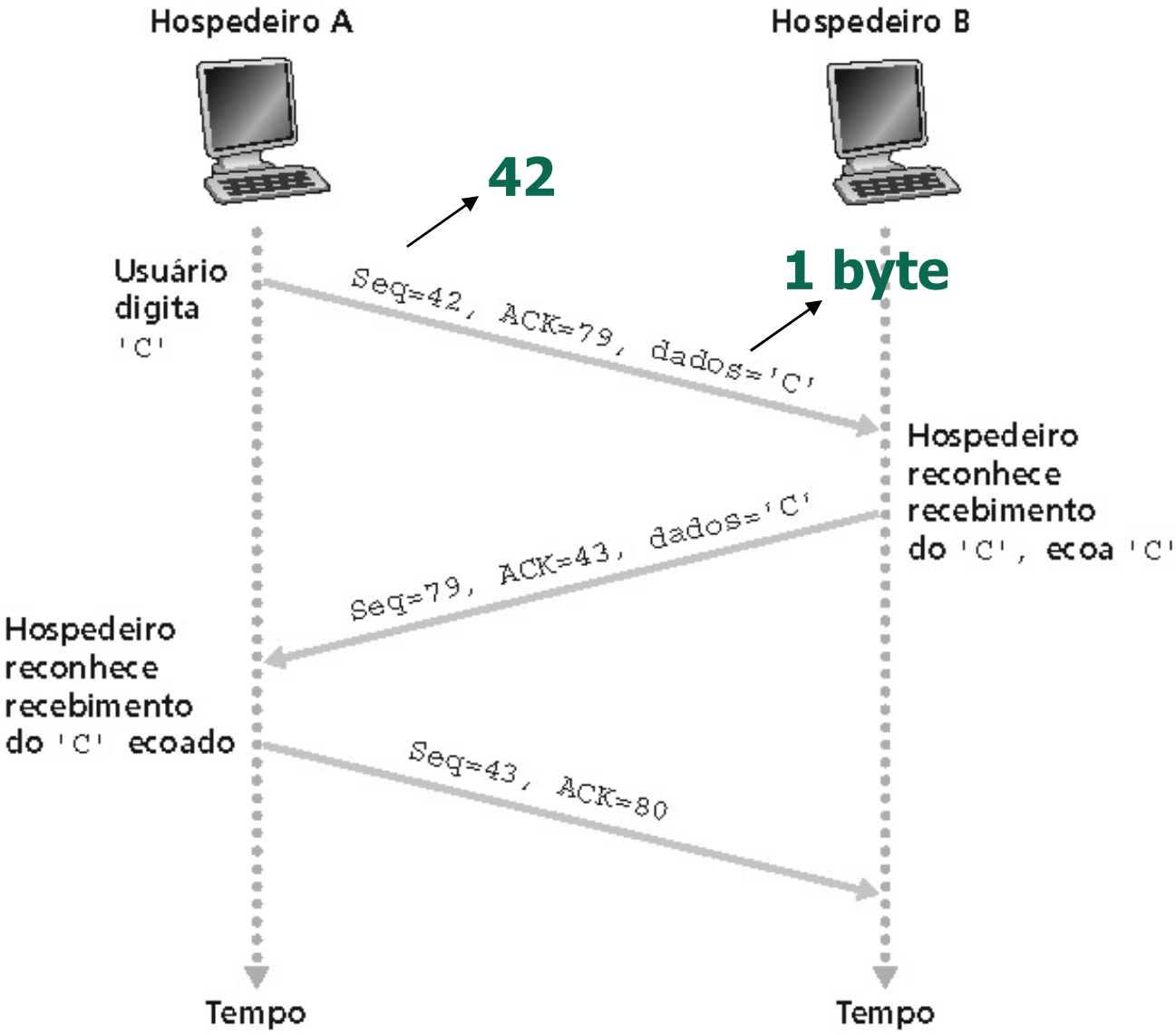
- Número de reconhecimento
  - Número de sequência do **próximo byte esperado** do "outro lado"
    - Identificador do seguimento a ser enviado
  - ACK cumulativo

- Como o receptor trata os segmentos fora da ordem?
  - Nada é especificado pela RFC
  - É definido por quem implementa o protocolo

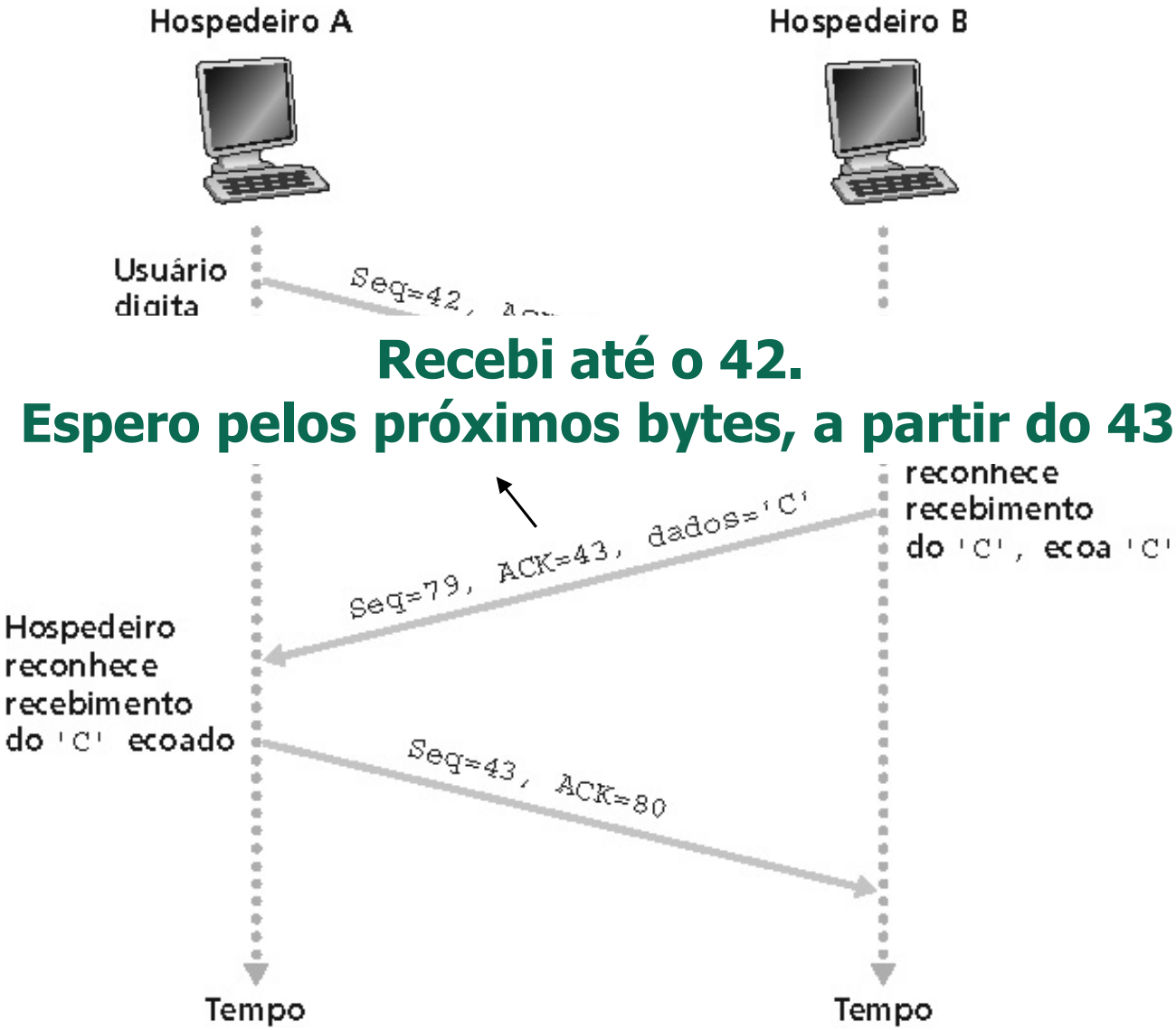
Números de Sequência e ACKs



Números de Sequência e ACKs



Números de Sequência e ACKs



Temporização

Instituto de computação

- Como escolher valor do temporizador TCP?
  - Maior que o RTT

RTT é variável!

- Muito curto
  - Estouro prematuro do temporizador
    - Retransmissões desnecessárias
- Muito longo
  - Reação demorada à perda de segmentos

Temporização

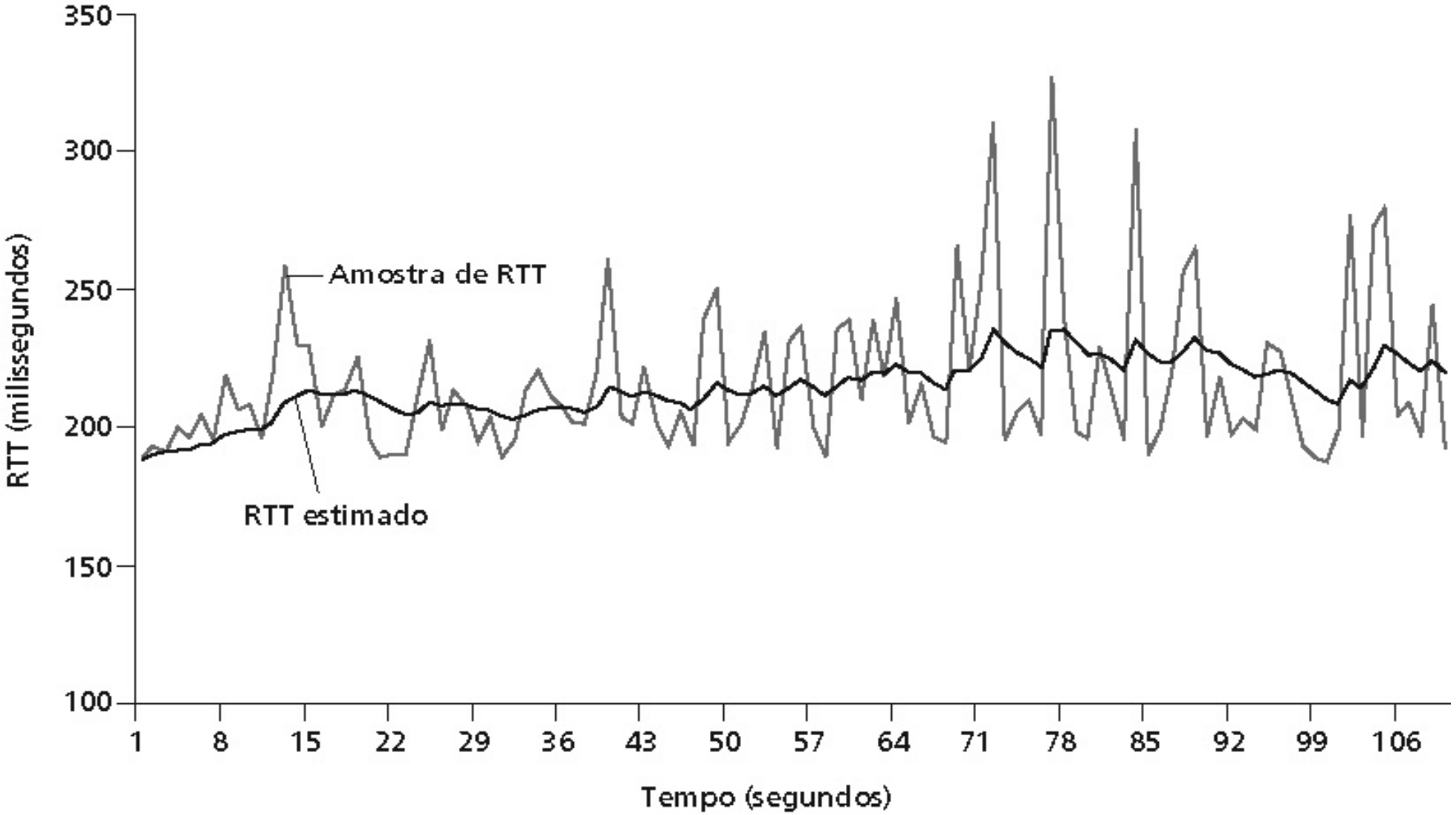
Instituto de computação

- Como estimar o RTT?
  - Medir o tempo entre a transmissão de um segmento e o recebimento do ACK correspondente
    - Ignorar retransmissões
- RTT de cada amostra pode ter grande variação
  - Solução: usar várias amostras recentes

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

- Média móvel exponencialmente ponderada
- Influência de uma amostra **diminui** exponencialmente com o tempo
- Valor típico de $\alpha = 0{,}125$

Temporização



Temporização

Instituto de computação

- Intervalo de temporização escolhido
  - `EstimatedRTT` mais uma "margem de segurança"
    - Definida pela **variação das amostras** em relação à estimativa
- Determinar o desvio da amostra em relação à estimativa

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

(valor típico de $\beta = 0{,}25$)

- Temporizador definido por

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

Transferência Confiável do TCP

Instituto de computação

- Provê um **serviço confiável sobre o serviço não confiável** do protocolo IP
- Segmentos transmitidos em "paralelo"
  - Princípio do *Go-Back-N*
- ACKs cumulativos
- O TCP usa um único temporizador para retransmissões

Transferência Confiável do TCP

Instituto de computação

- As retransmissões são disparadas por
  - Estouros de temporização
  - ACKs duplicados

Transmissor TCP

Instituto de Computação

- Transmissor TCP simplificado
  - Ignora ACKs duplicados
  - Ignora controles de fluxo e de congestionamento
- Ao receber os dados da aplicação
  - Cria segmento com no. de sequência (nseq)
    - nseq é o no. de seq. do primeiro byte de dados do segmento
  - Dispara o temporizador
    - Se já não estiver disparado
    - Relativo ao segmento mais antigo ainda não reconhecido
    - Valor calculado previamente

Transmissor TCP

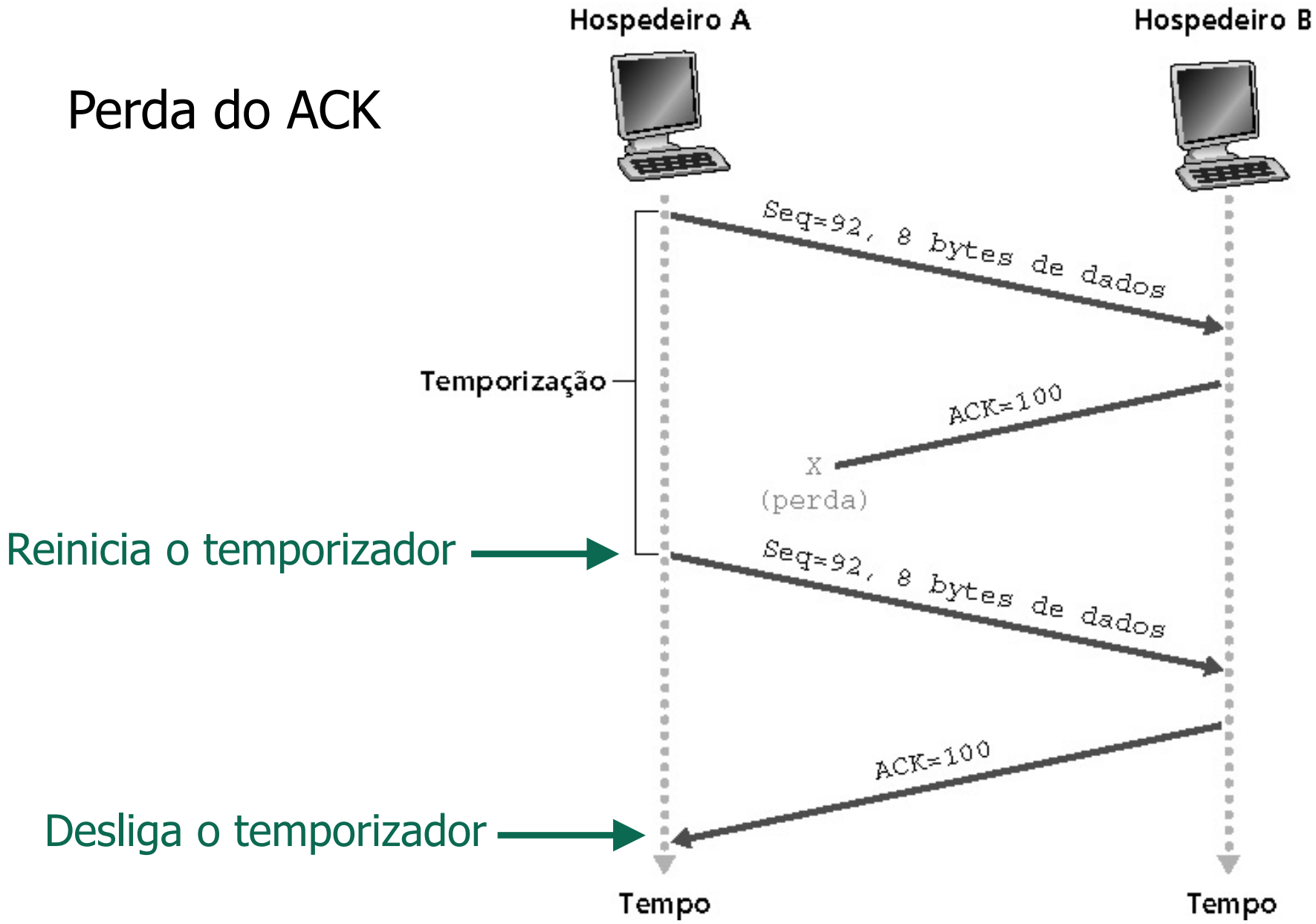
Instituto de computação

- Quando ocorre um estouro do temporizador
  - Retransmitir o segmento que causou o estouro do temporizador
  - Reiniciar o temporizador

- Quando um ACK é recebido
  - Se reconhecer segmentos ainda não reconhecidos
    - Atualizar informação sobre o que foi reconhecido
    - Disparar novamente o temporizador se ainda houver segmentos ainda não reconhecidos

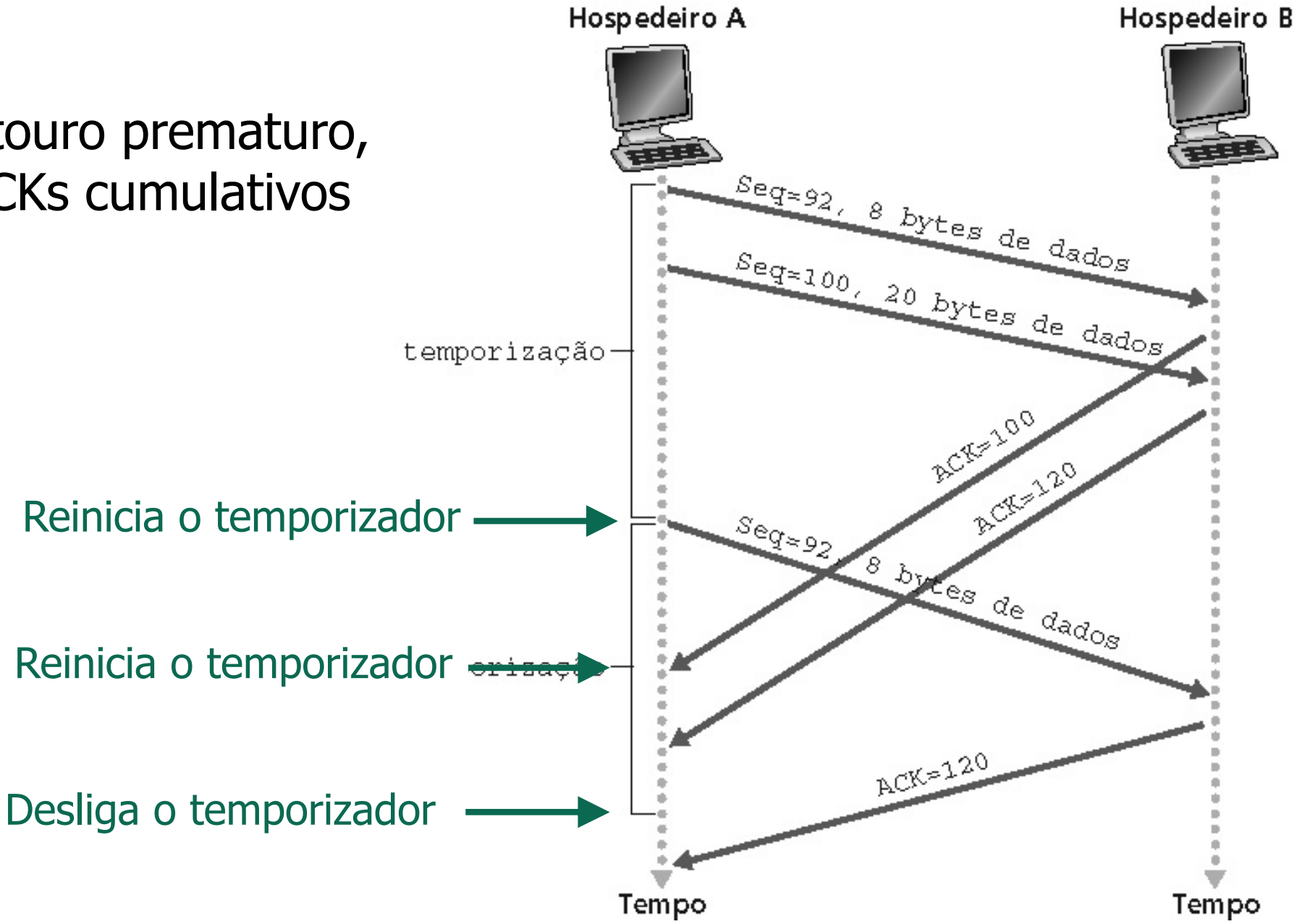
Cenário de Retransmissão

Perda do ACK



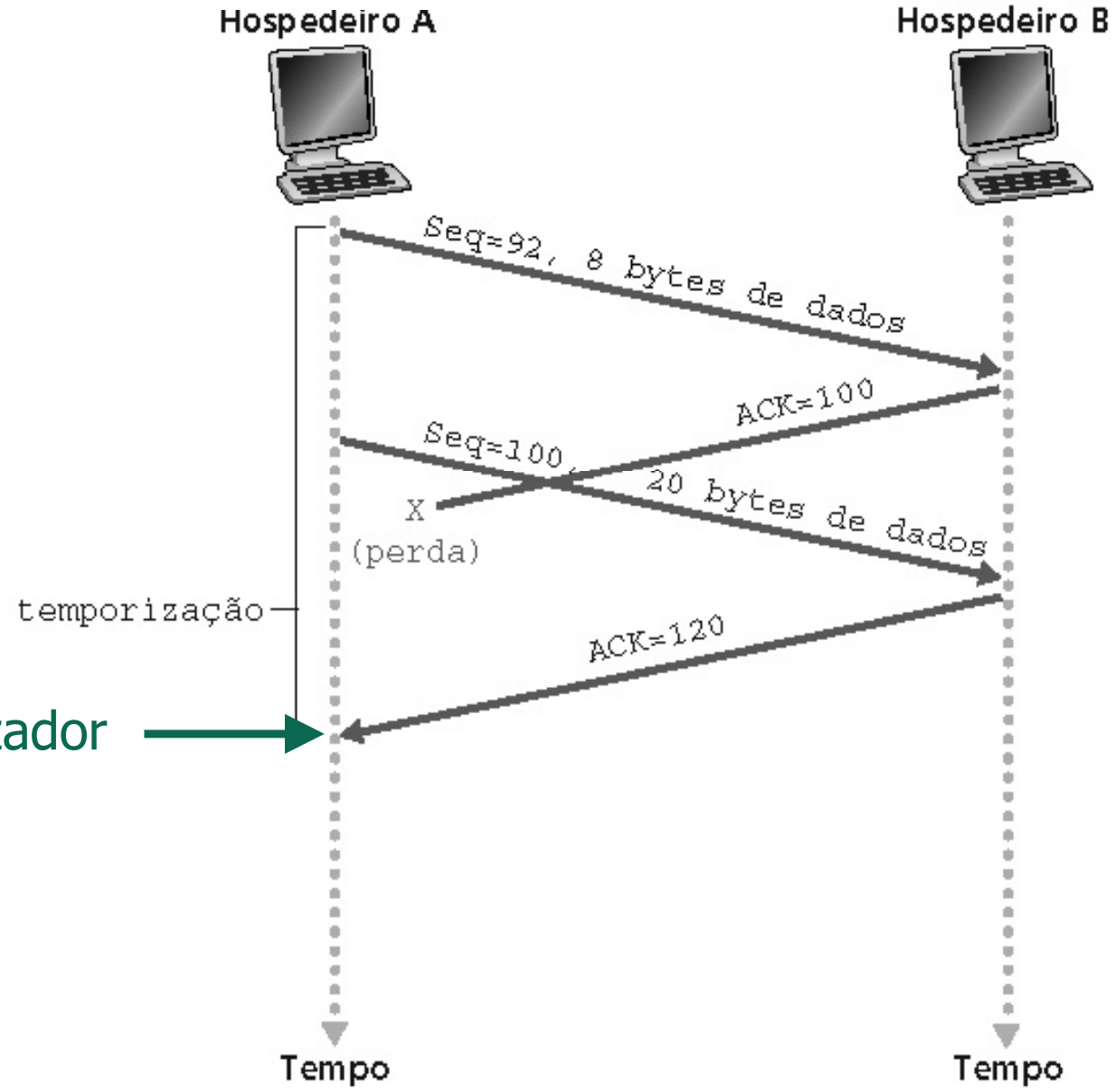
Cenário de Retransmissão

Estouro prematuro,
ACKs cumulativos



Cenário de Retransmissão

Perda de um
ACK cumulativo



Desliga o temporizador

Geração de ACKs

Instituto de computação

Evento no receptor	Ação do receptor
Chegada de segmento em ordem sem lacunas, anteriores já reconhecidos	ACK retardado. Espera até 500 ms pelo 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 começa no início da lacuna

Retransmissão Rápida

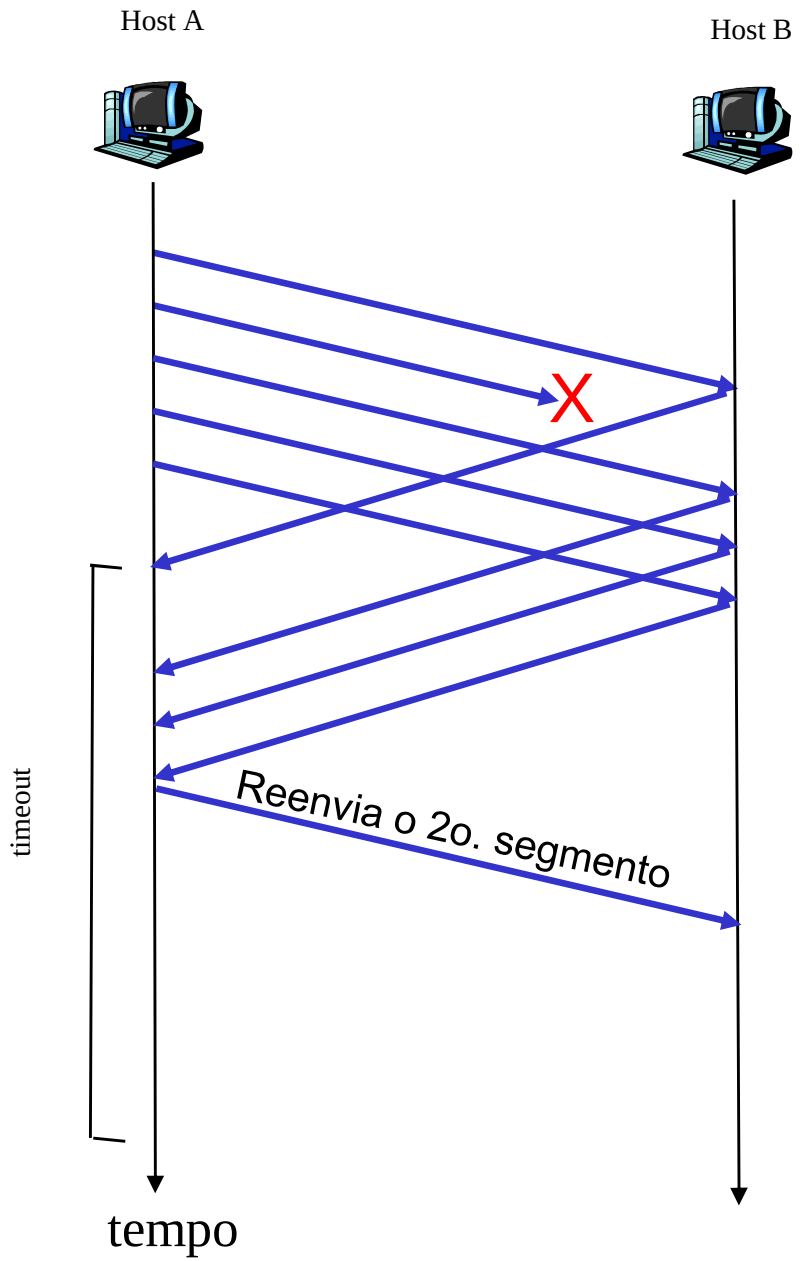
Instituto de computação

- O intervalo do temporizador pode ser grande
  - Espera demais para retransmitir um pacote perdido
- Detectar 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 duplicados para os mesmos dados
    - Supõe que o segmento após os dados reconhecidos se perdeu

Retransmite o segmento antes que o temporizador estoure

Retransmissão Rápida

Retransmissão de um segmento após três ACKs duplicados



Retransmissão Rápida

Instituto de computação

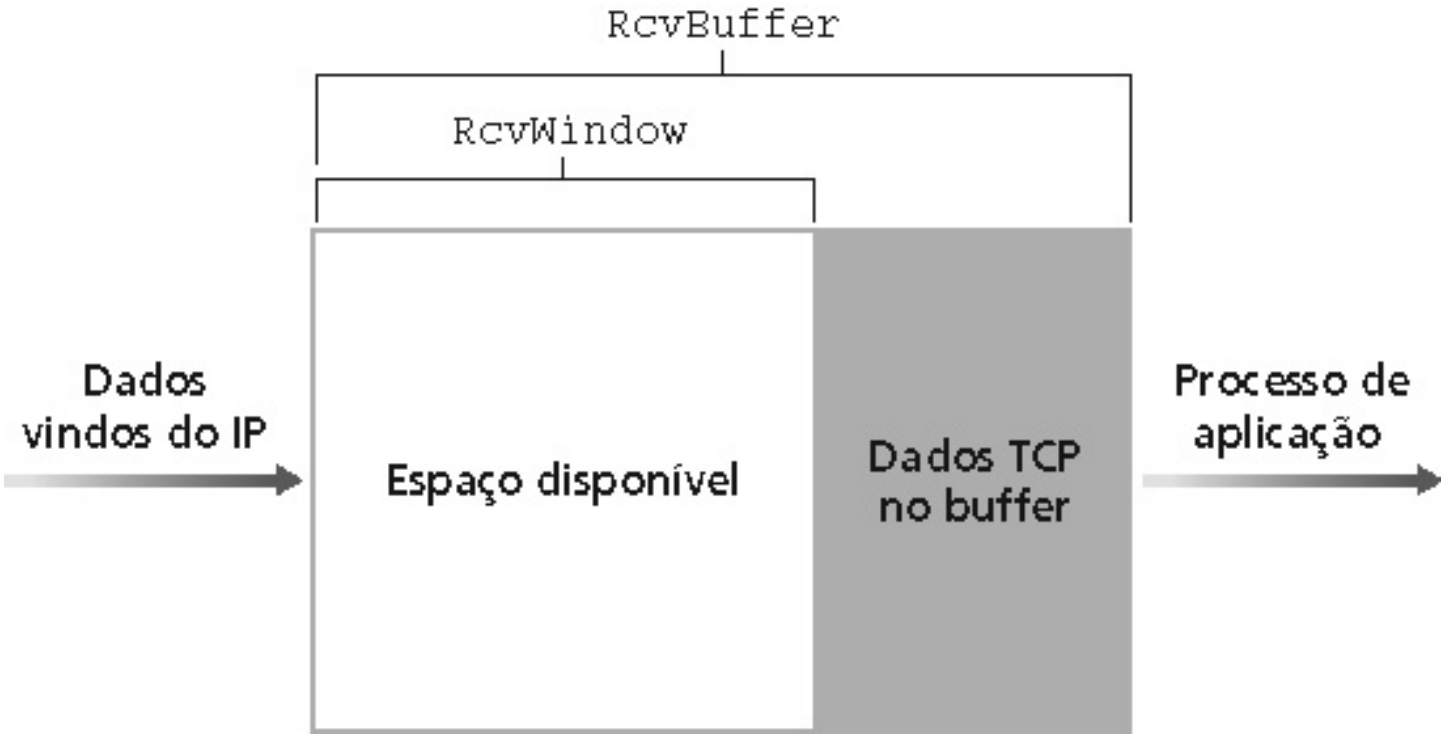
```
event: recebido ACK, com valor do campo ACK de y
            if (y > SendBase) {
                SendBase = y
                if (houver segmentos ainda não reconhecidos)
                    liga temporizador
                else desliga temporizador
                }
            else {
        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

um ACK duplicado para um segmento já reconhecido

Retransmissão rápida

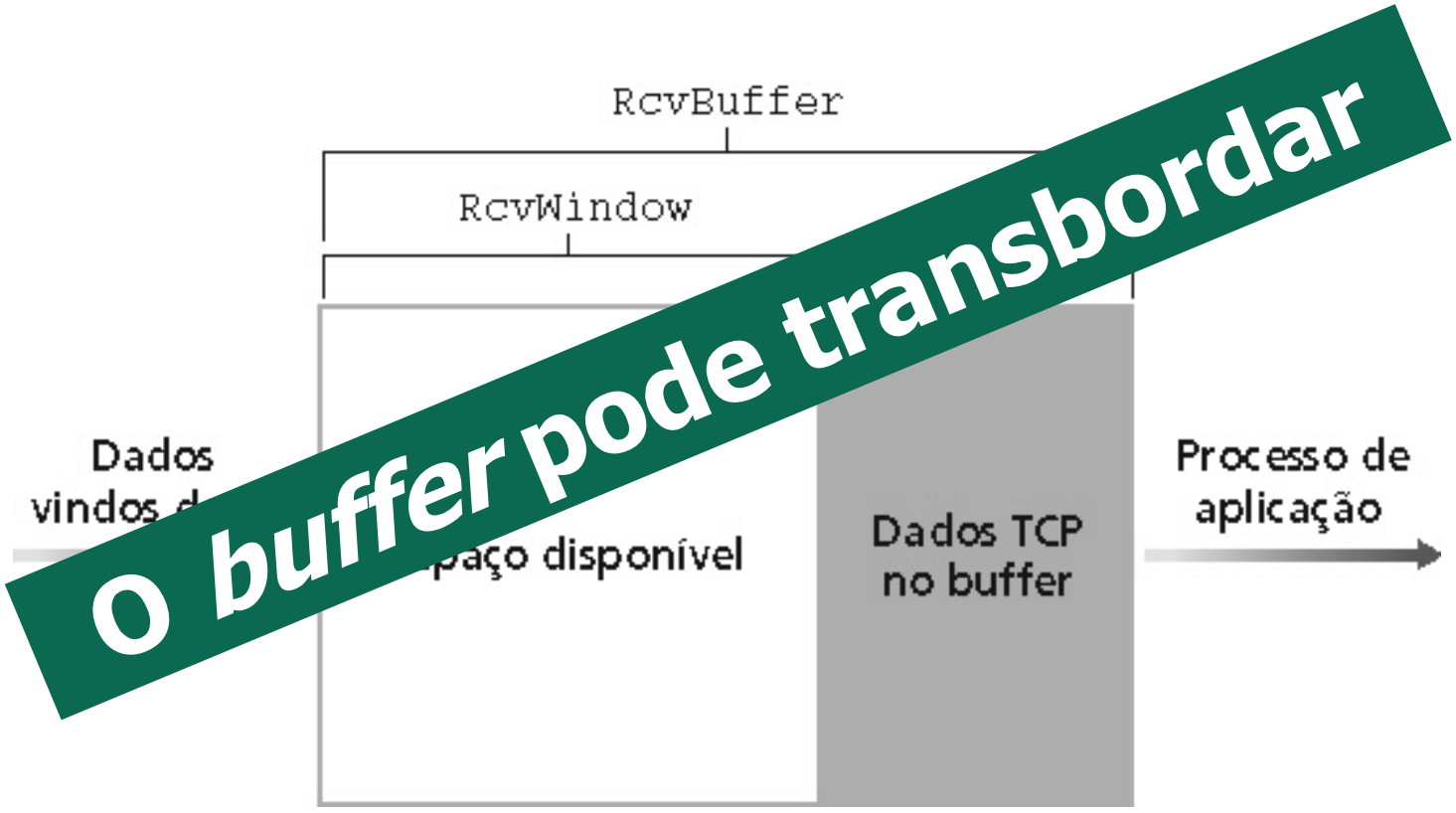
Controle de Fluxo do TCP

- Receptor possui um *buffer* de recepção
  - Processos da aplicações podem demorar a ler do *buffer*



Controle de Fluxo do TCP

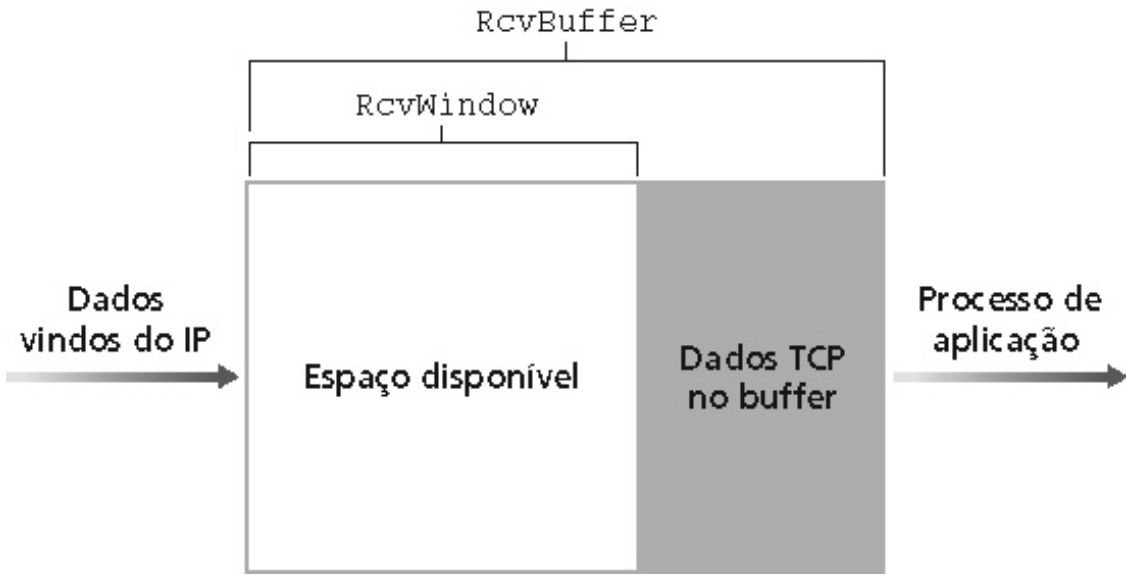
- Receptor possui um *buffer* de recepção
  - Processos da aplicações podem demorar a ler do *buffer*



Controle de Fluxo do TCP

- Funcionamento
  - Suposição
    - Receptor descarta segmentos recebidos fora de ordem
  - Espaço livre no *buffer*

```
RcvWindow = RcvBuffer-[LastByteRcvd - LastByteRead]
```



Controle de Fluxo do TCP

Instituto de computação

- Funcionamento
  - Suposição
    - Receptor descarta segmentos recebidos fora de ordem
  - Espaço livre no *buffer*

```
RcvWindow = RcvBuffer-[LastByteRcvd - LastByteRead]
```

  - O receptor anuncia o espaço livre no *buffer*
    - O valor da janela (**`RcvWindow`**) é informado nos segmentos
  - O transmissor limita os dados não reconhecidos ao tamanho da janela de recepção
    - Garante que o ***buffer* do receptor não transbordará**

Estabelecimento de Conexão

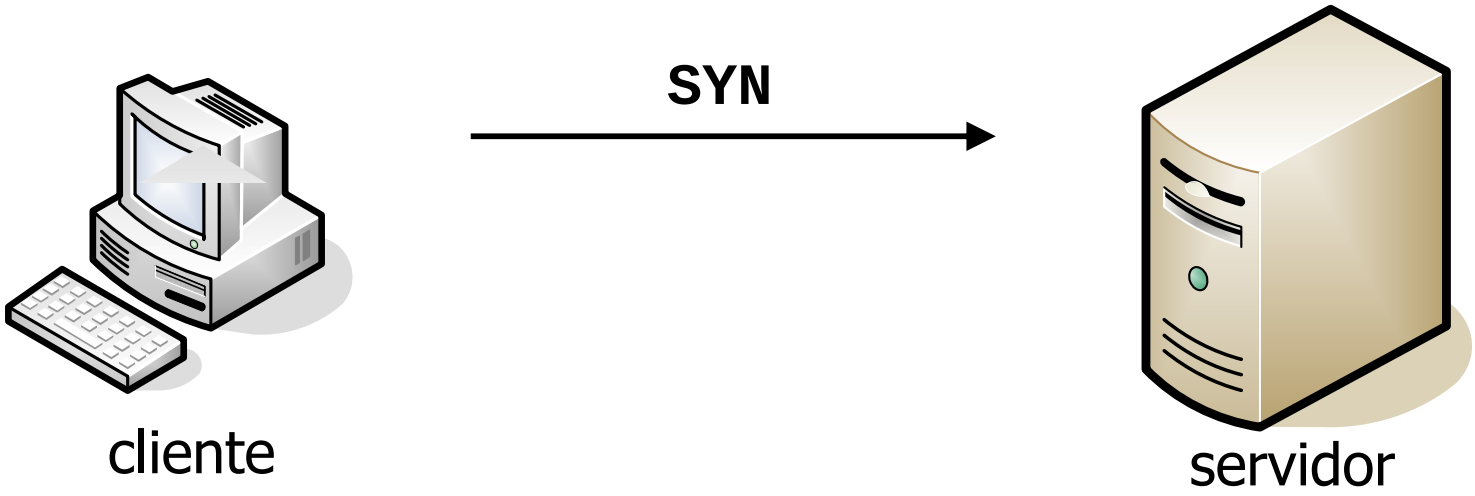
Instituto de computação

- É feita **antes da troca de dados**

- Inicialização de variáveis
  - Números de seqüência
  - Tamanho dos *buffers*,
  - Variáveis do mecanismo de controle de fluxo
    - Janela de recepção (**`RcvWindow`**)
  - Etc.

Estabelecimento de Conexão

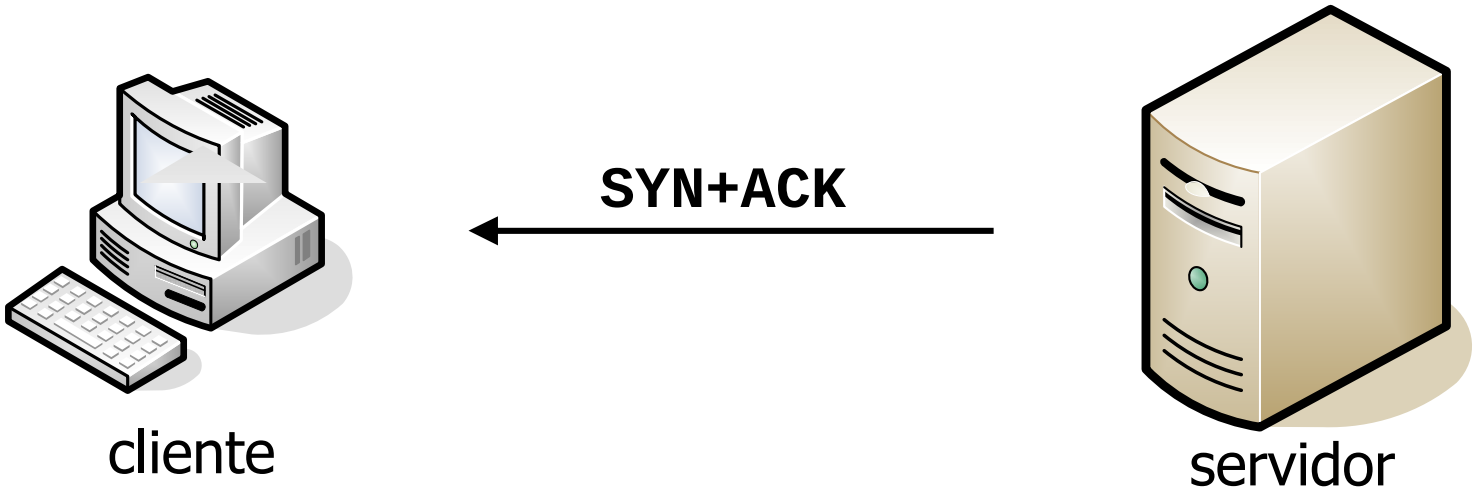
"*Three-way handshake*"



1. Cliente envia segmento de controle SYN para o servidor
   - Especifica o número de sequência inicial e não envia dados

Estabelecimento de Conexão

"*Three-way handshake*"

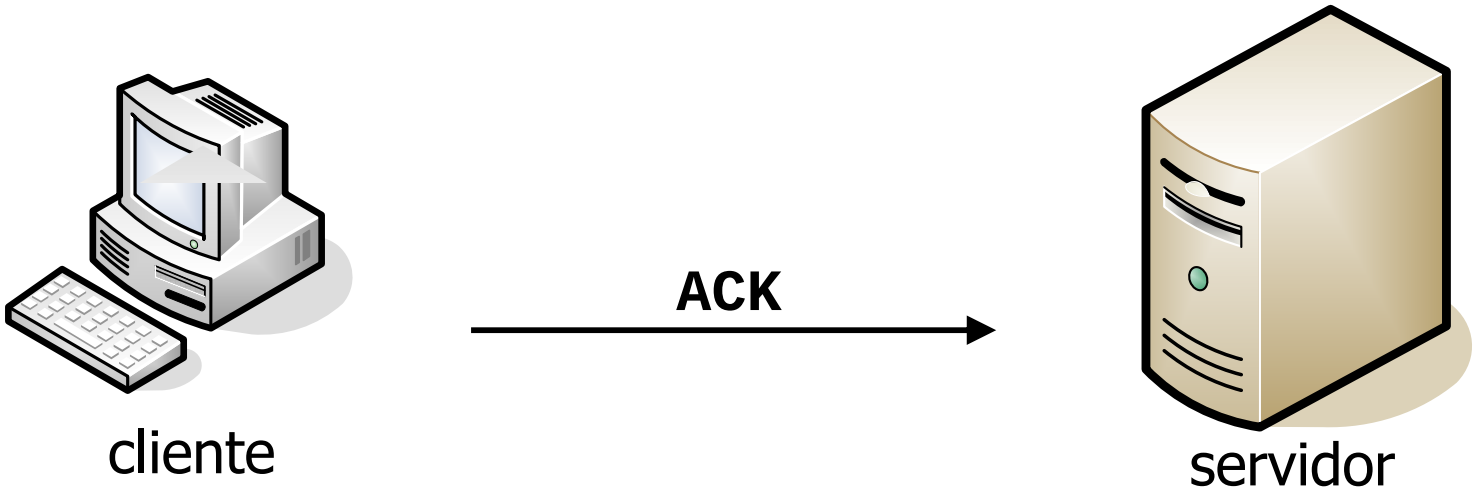


2. Ao receber o SYN, o servidor responde com segmento de controle SYN+ACK

Define o tamanho dos *buffers* e
especifica o número inicial de sequência do servidor para o receptor

Estabelecimento de Conexão

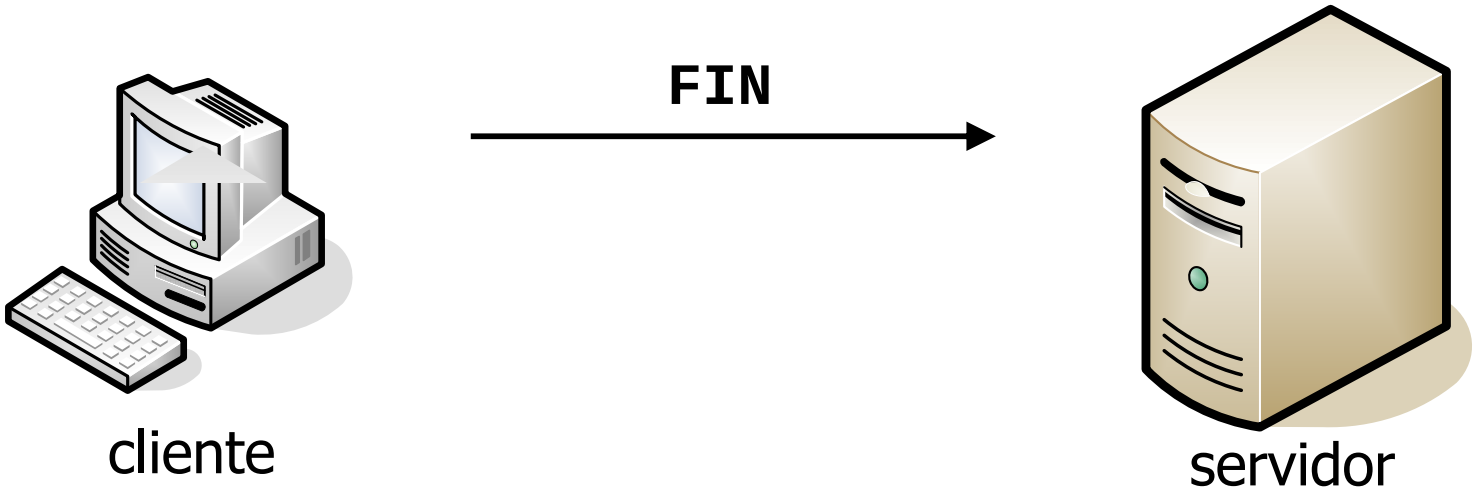
"*Three-way handshake*"



3. Ao receber SYN+ACK, o cliente responde com segmento ACK

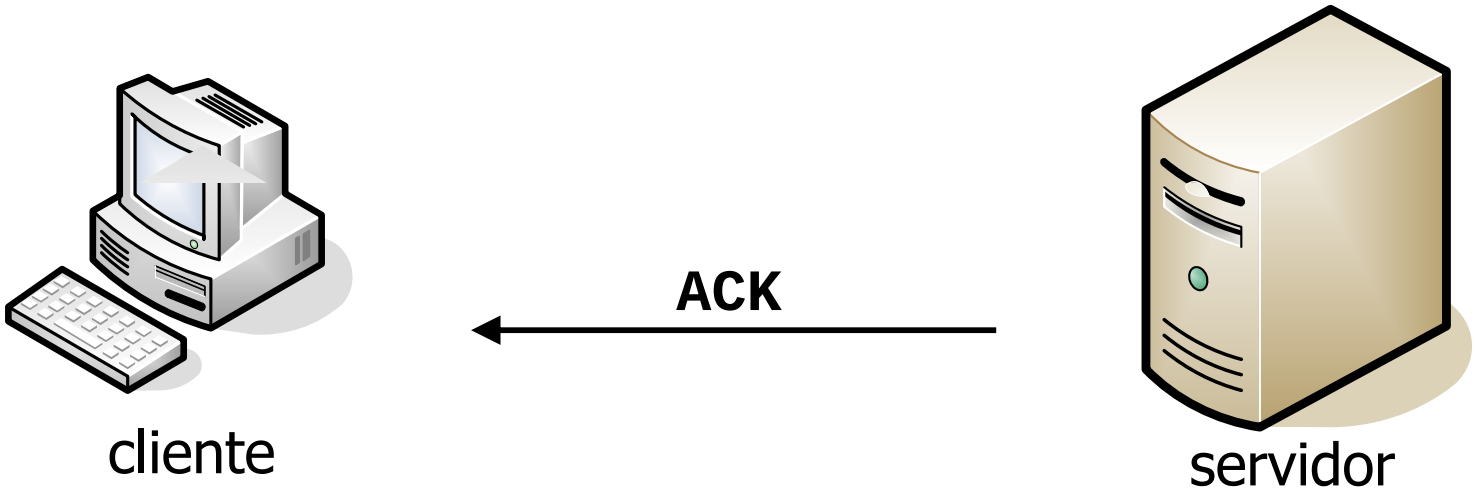
Pode conter dados

Encerramento de Conexão



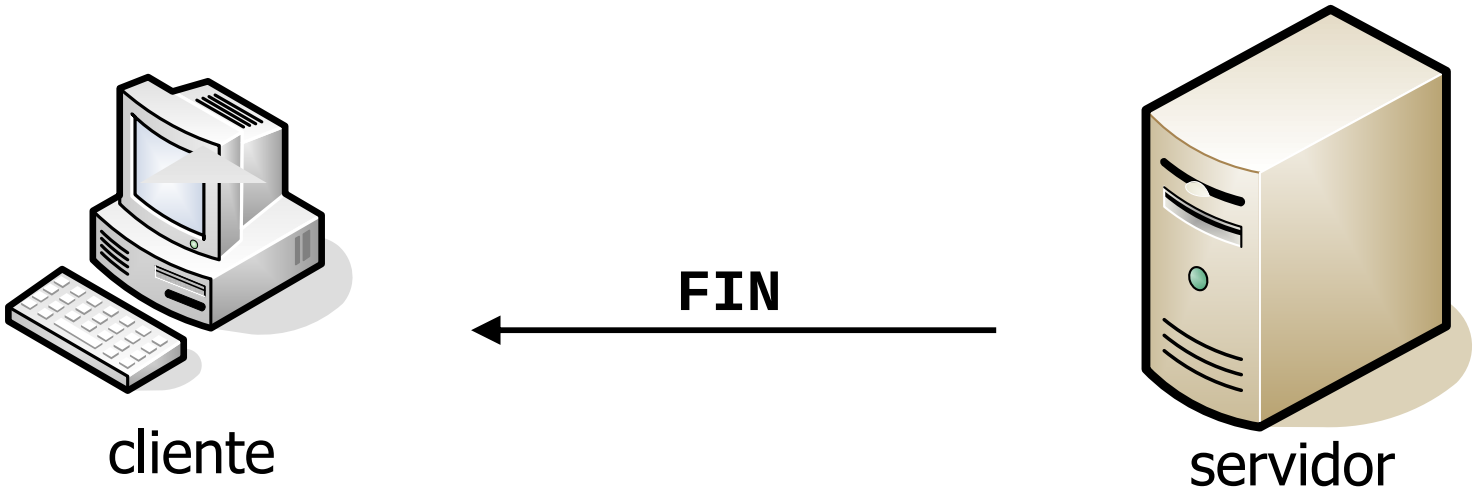
1. Cliente envia segmento de controle FIN ao servidor

Encerramento de Conexão



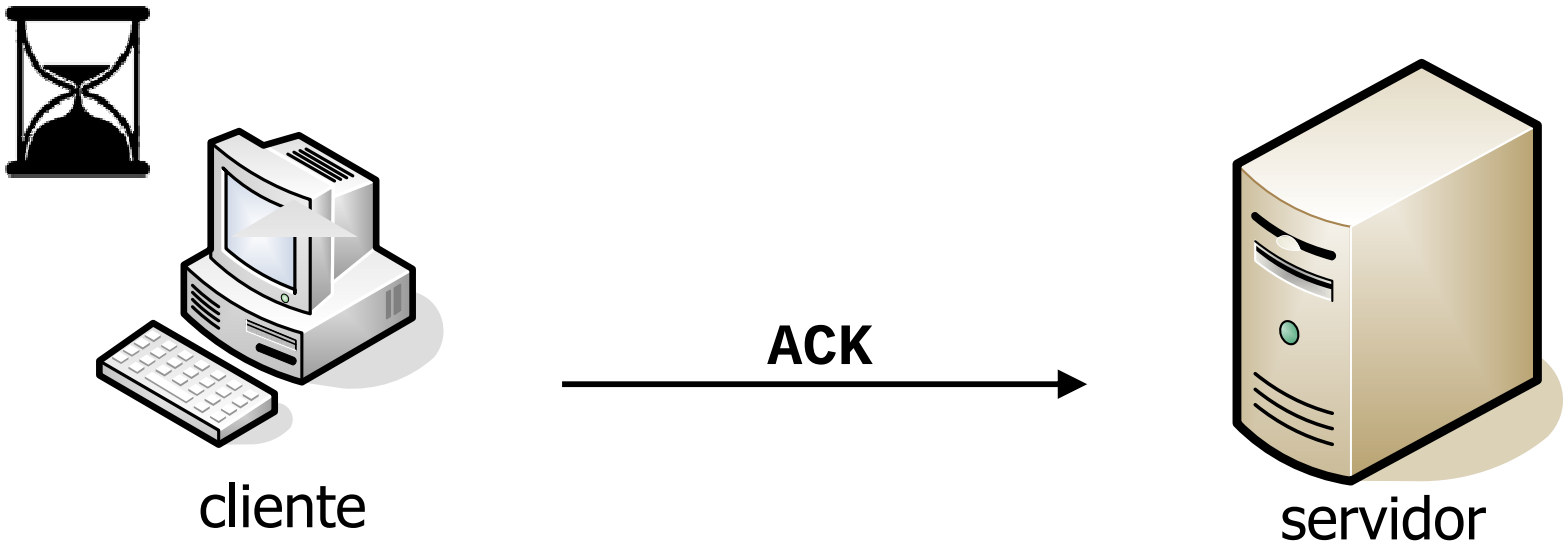
2. Ao receber FIN, o servidor responde com ACK

Encerramento de Conexão



3. Em seguida, o servidor envia FIN e encerra a conexão

Encerramento de Conexão

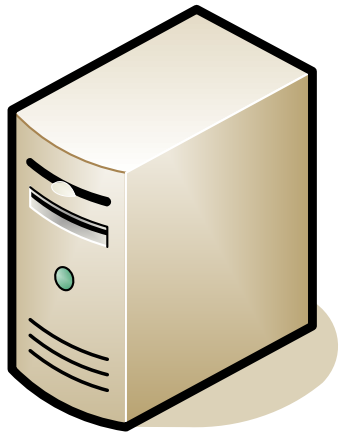


4. Ao receber FIN, o cliente responde com ACK

Entre em "espera temporizada" → responderá com ACK a FINs recebidos

Encerramento de Conexão

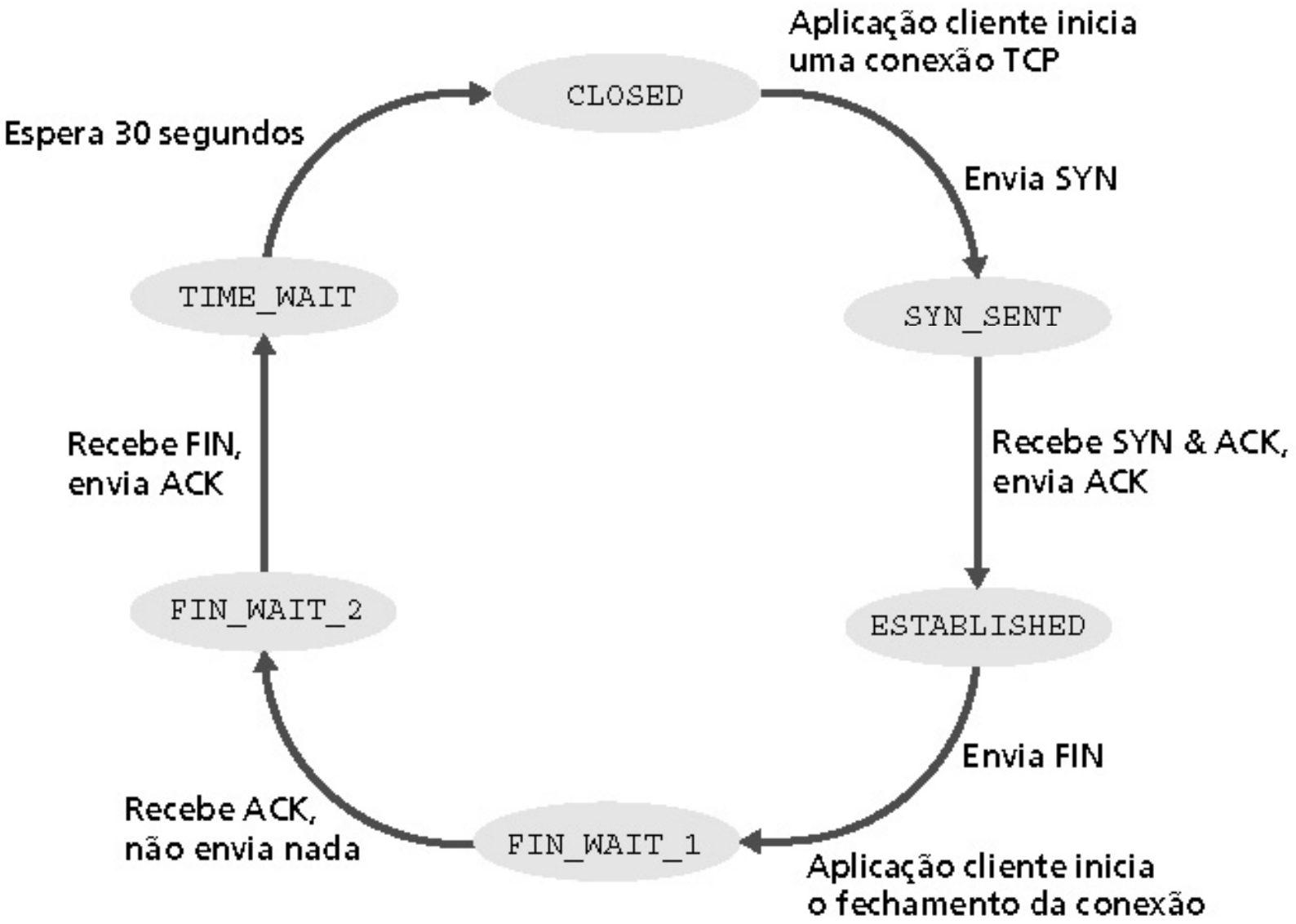
cliente



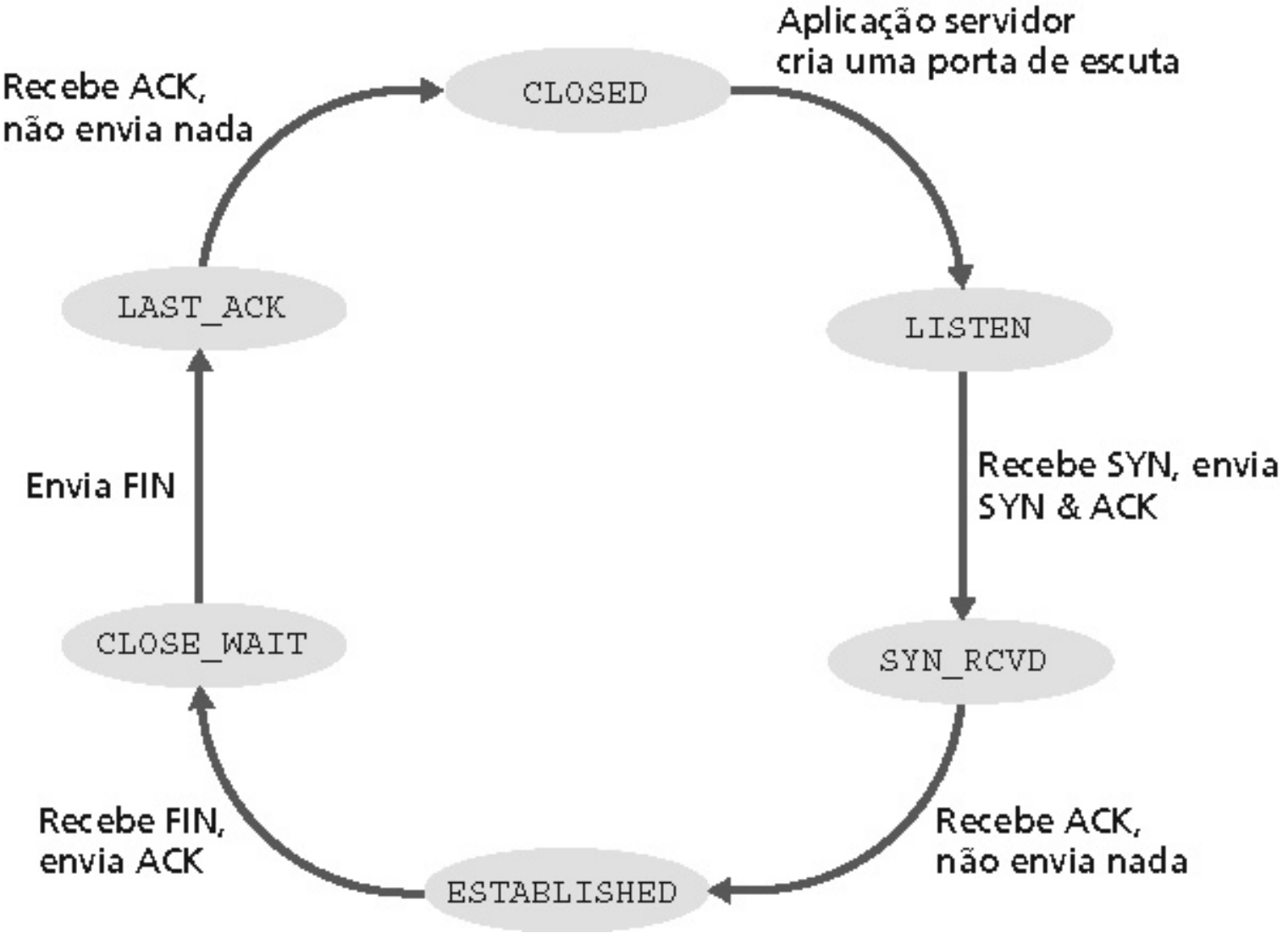
servidor

5. Quando o temporizador estoura a conexão é encerrada

Estados no Cliente TCP



Estados no Servidor TCP



Controle de Congestionamento

Instituto de computação

- Fontes enviam dados acima da capacidade da rede de tratá-los
  - Perda de pacotes
    - Saturação de *buffers* nos roteadores
  - Atrasos maiores
    - Espera nos *buffers* dos roteadores

A rede está congestionada!

Controle de Congestionamento

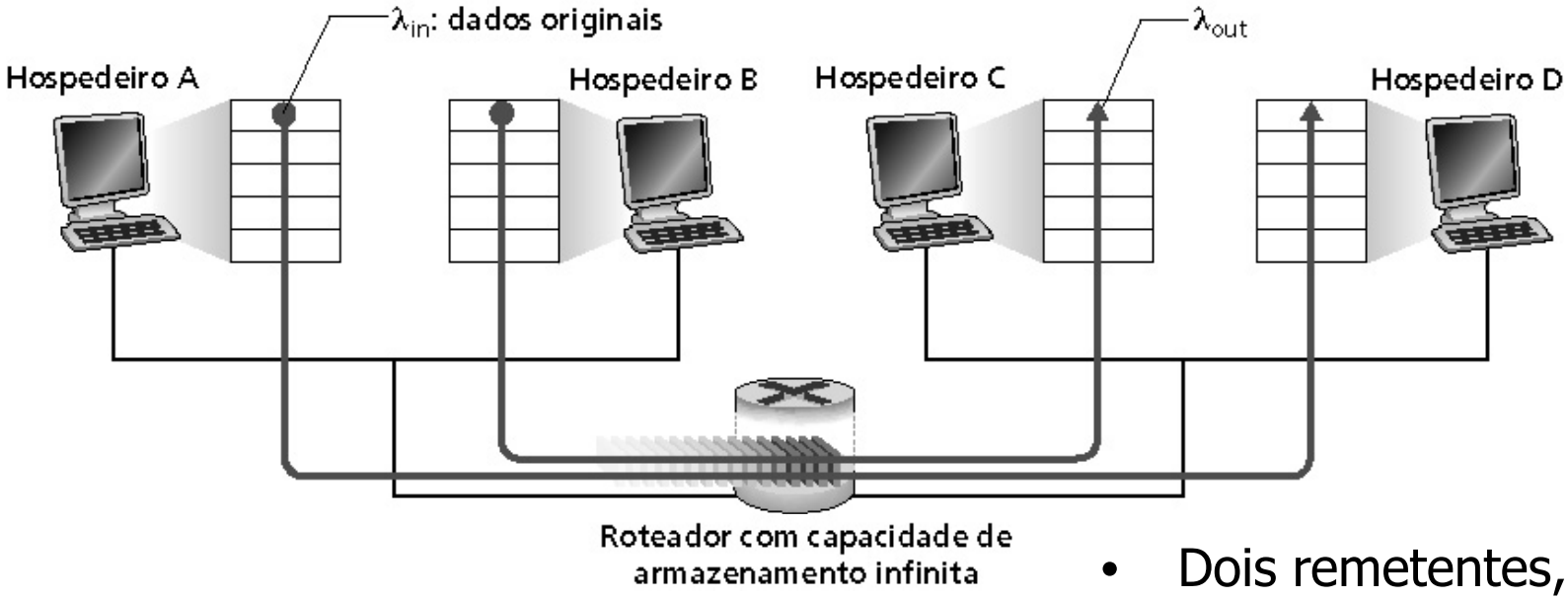
Instituto de computação

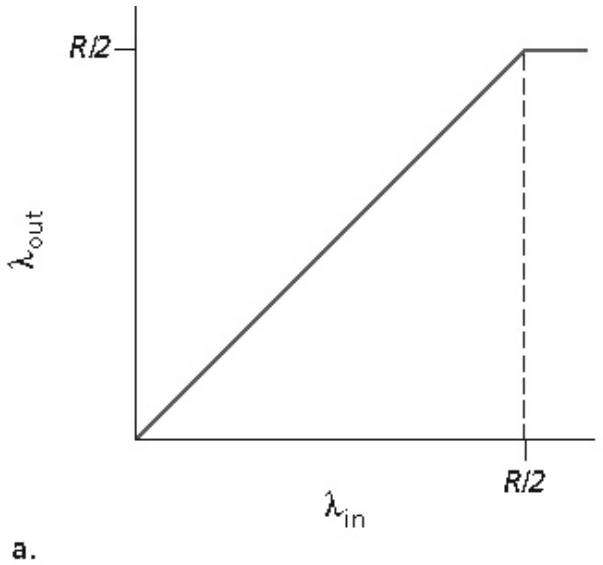
- Fontes enviam dados acima da capacidade da rede de tratá-los
  - Perda de pacotes
    - Saturação de *buffers* nos roteadores
  - Atrasos maiores
    - Espera nos *buffers* dos roteadores

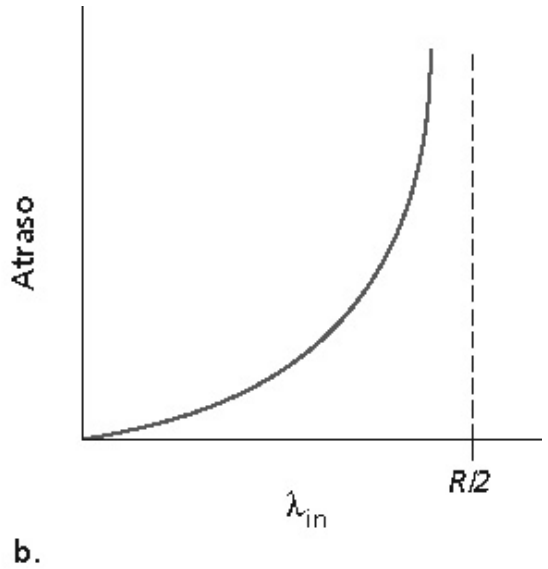
A rede está congestionada!

- **É diferente do controle de fluxo**
  - É um estado da rede e não dos sistemas finais

Congestionamento: *Buffers* Infinitos



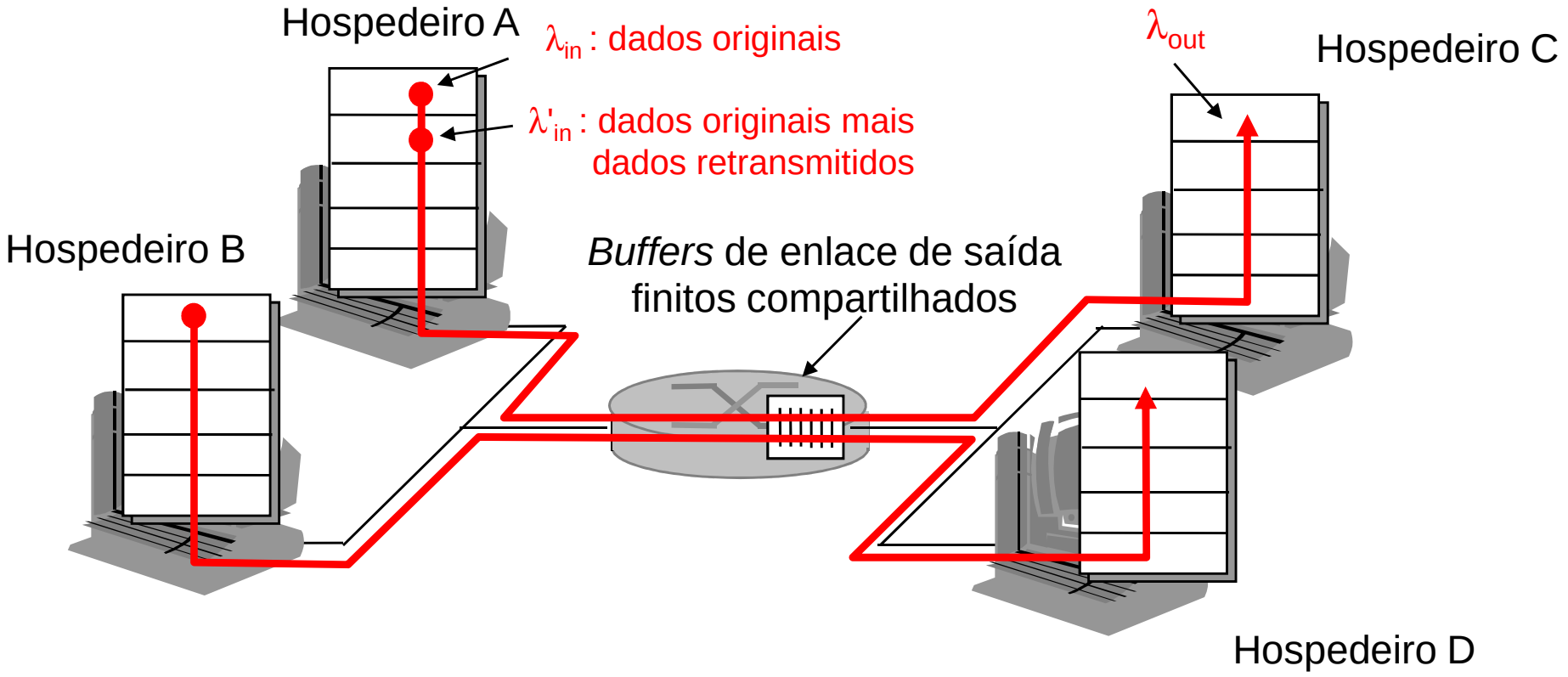




- Dois remetentes, dois receptores
- Um roteador, *buffers* infinitos
- Sem retransmissão
- Grandes retardos qdo. congestionada
- Máxima vazão alcançável

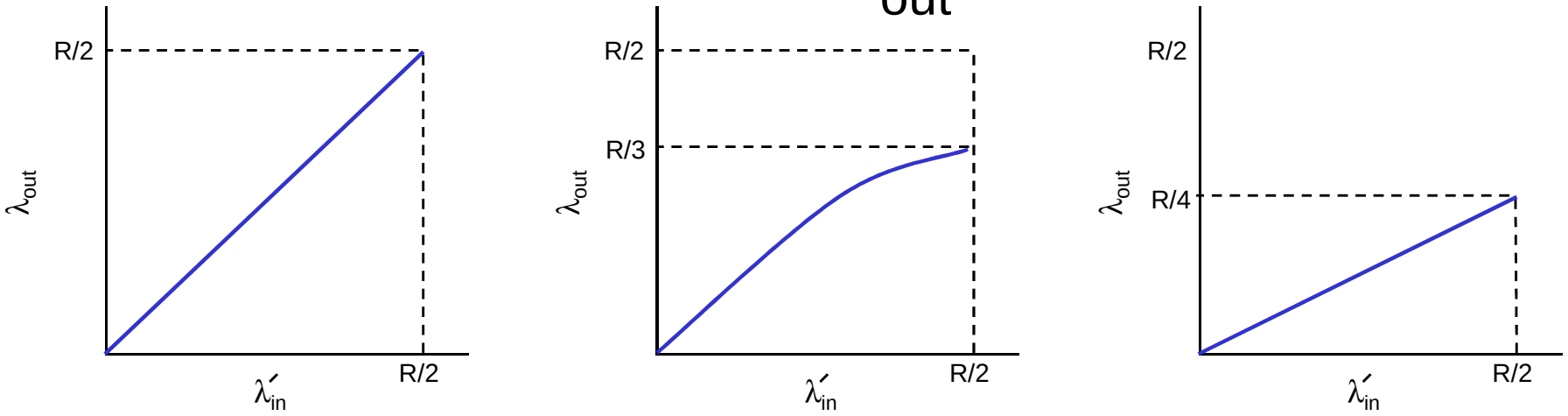
Congestionamento: *Buffers* Finitos

- Um roteador, *buffers* **finitos**
- Retransmissão pelo remetente de pacote perdido



Congestionamento: *Buffers* Finitos

- Sempre: $\lambda_{in} = \lambda_{out}$ (goodput)
- Retransmissão "perfeita" apenas com perdas: $\lambda'_{in} > \lambda_{out}$
- retransmissão de pacotes atrasados (não perdidos) torna λ'_{in} maior (do que o caso perfeito) para o mesmo λ_{out}



a. b. c.

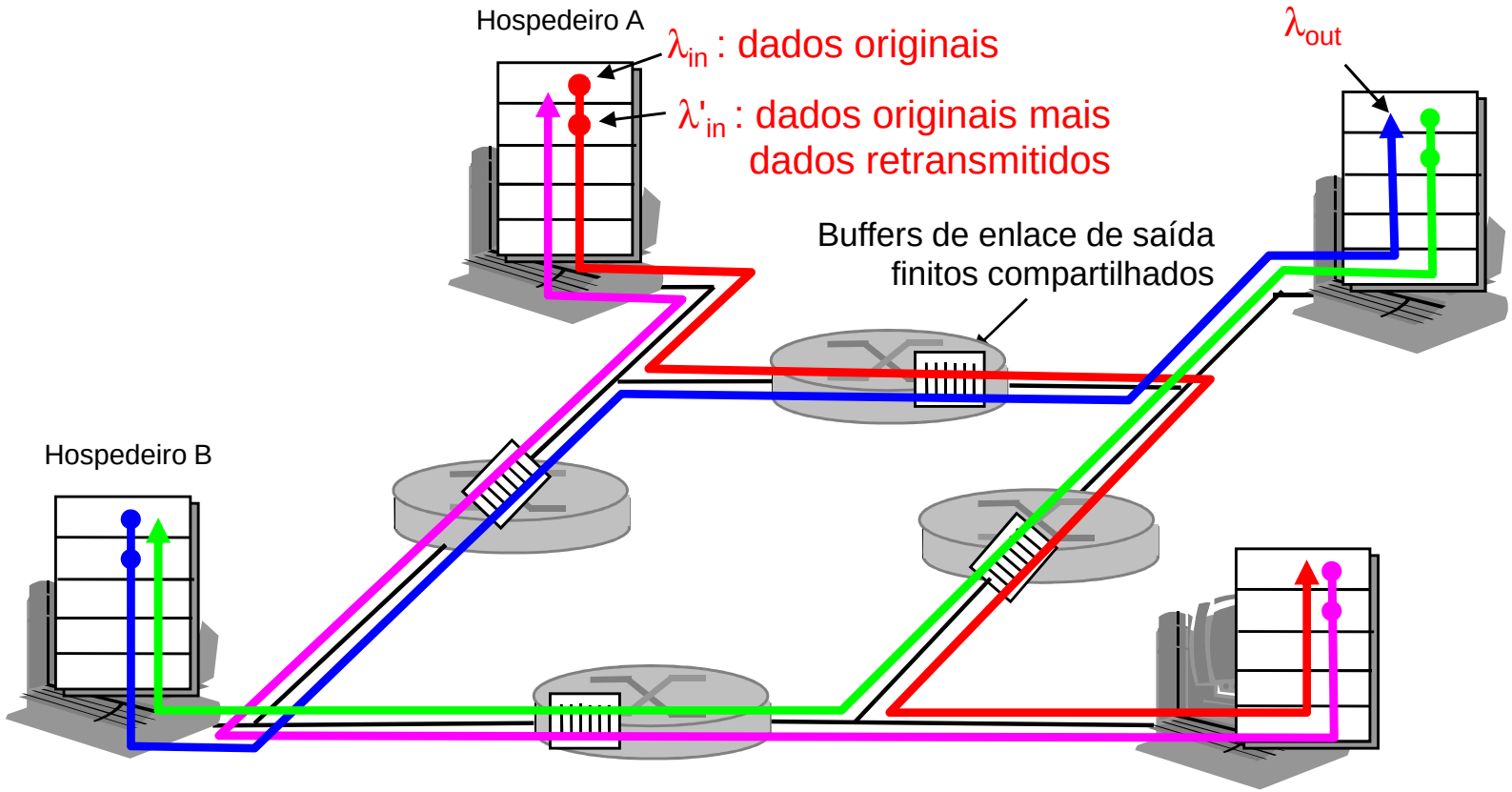
"Custos" de congestionamento:

- Mais trabalho (retransmissão) para um dado "*goodput*"
- Retransmissões desnecessárias: são enviadas múltiplas cópias do mesmo pacote

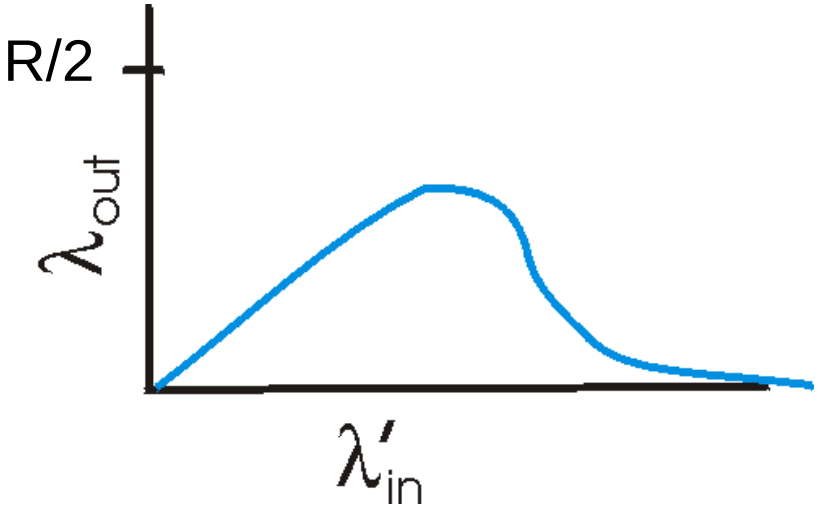
Congestionamento: Quatro Remetentes

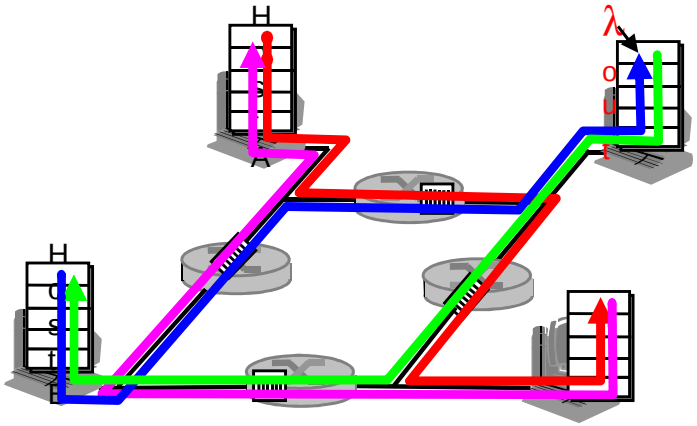
- Quatro remetentes
- Caminhos com múltiplos enlaces
- Temporização/retransmissão

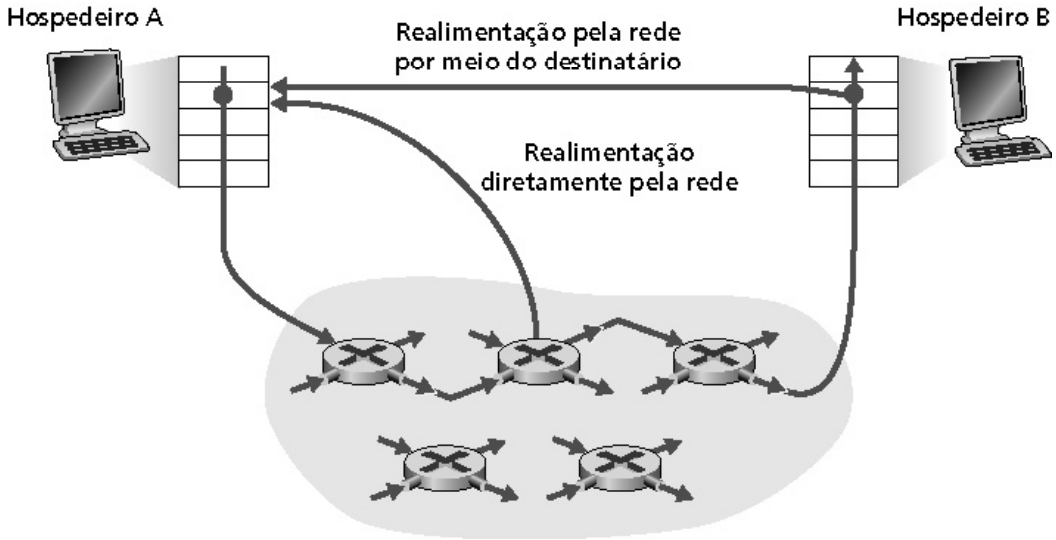
O que acontece à medida que λ_{in} e λ'_{in} crescem?



Congestionamento: Quatro Remetentes







Outro "custo" do congestionamento

- Quando o pacote é descartado, qq. capacidade de transmissão já usada (antes do descarte) para esse pacote foi desperdiçada

Controle de Congestionamento

Instituto de computação

- Controle de congestionamento fim-a-fim
  - Não usa realimentação explícita da rede
  - Congestionamento é inferido a partir das perdas e dos atrasos observados nos sistemas finais
  - **Abordagem usada pelo TCP**

- Controle de congestionamento assistido pela rede
  - Roteadores enviam informações para os sistemas finais
  - Bit indicando congestionamento (SNA, DECbit, TCP/IP ECN, ATM)
  - Taxa explícita para envio pelo transmissor

Controle de Congestionamento do TCP

Instituto de computação

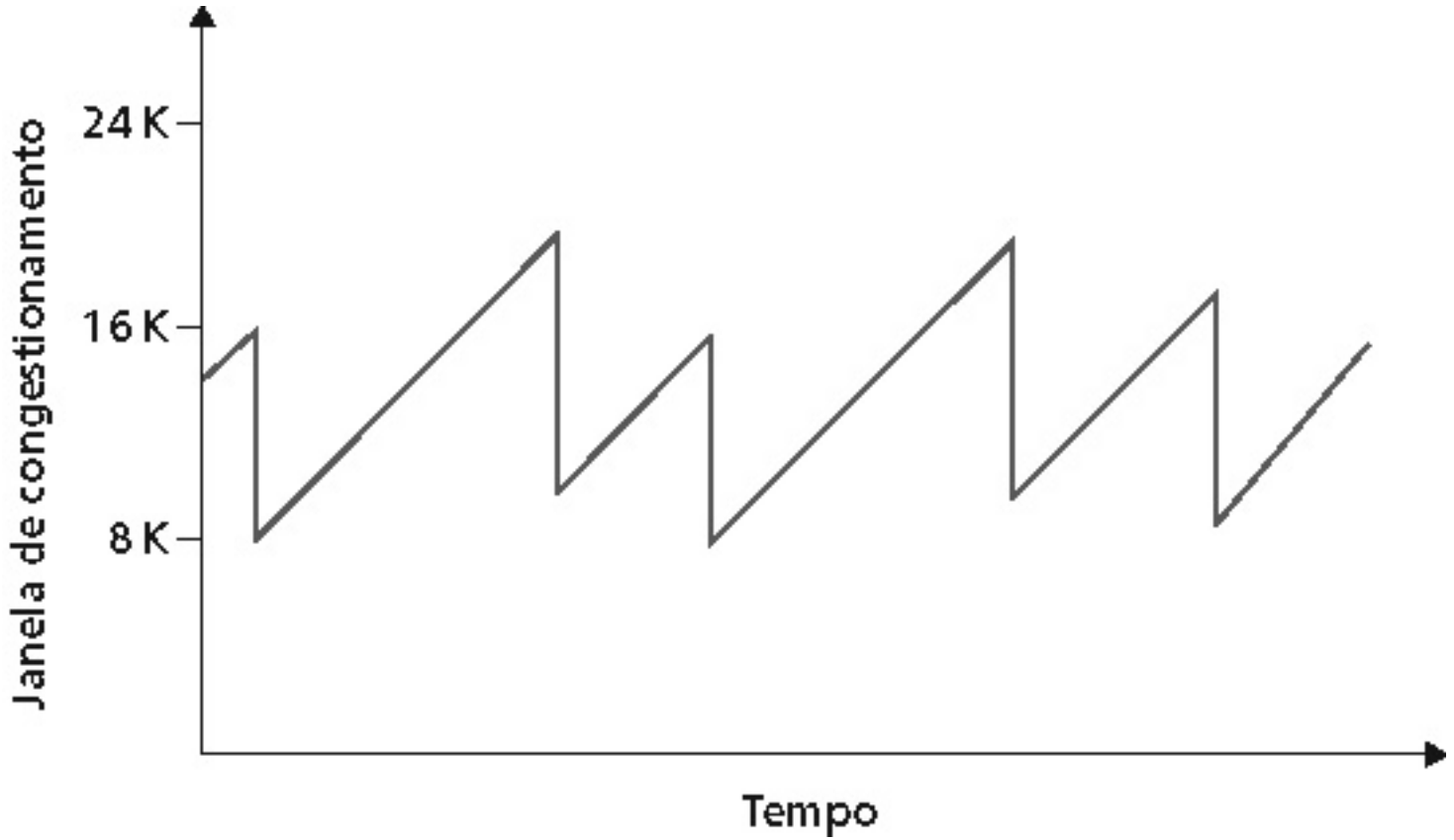
- Idéia
  - Aumentar a taxa de transmissão (tamanho da janela)
    - Testar a largura de banda utilizável até que ocorra uma perda
- Aumento aditivo
  - Incrementa **CongWin** de 1 MSS a cada RTT até detectar uma perda
- Diminuição multiplicativa
  - Corta **CongWin** pela metade após evento de perda

Controle de Congestionamento do TCP

Instituto de computação

Comportamento de dente de serra

"Testando" a largura de banda



Controle de Congestionamento do TCP

Instituto de computação

- Transmissor limita a transmissão

$$\texttt{LastByteSent-LastByteAcked} \leq \texttt{CongWin}$$

$$\texttt{taxa} = \frac{\texttt{CongWin}}{\texttt{RTT}} \text{ bytes/s}$$

- **CongWin** é dinâmica, em função do congestionamento detectado da rede

Controle de Congestionamento do TCP

Instituto de computação

- Como o transmissor detecta o congestionamento?
  - Evento de perda
    - **Estouro do temporizador ou 3 ACKs duplicados**
  - Transmissor reduz a taxa (**`CongWin`**) após evento de perda

- Três mecanismos
  - AIMD (*Additive-Increase, Multiplicative-Decrease*)
  - Partida lenta
  - Conservador após eventos de estouro de temporização

Partida Lenta do TCP

Instituto de computação

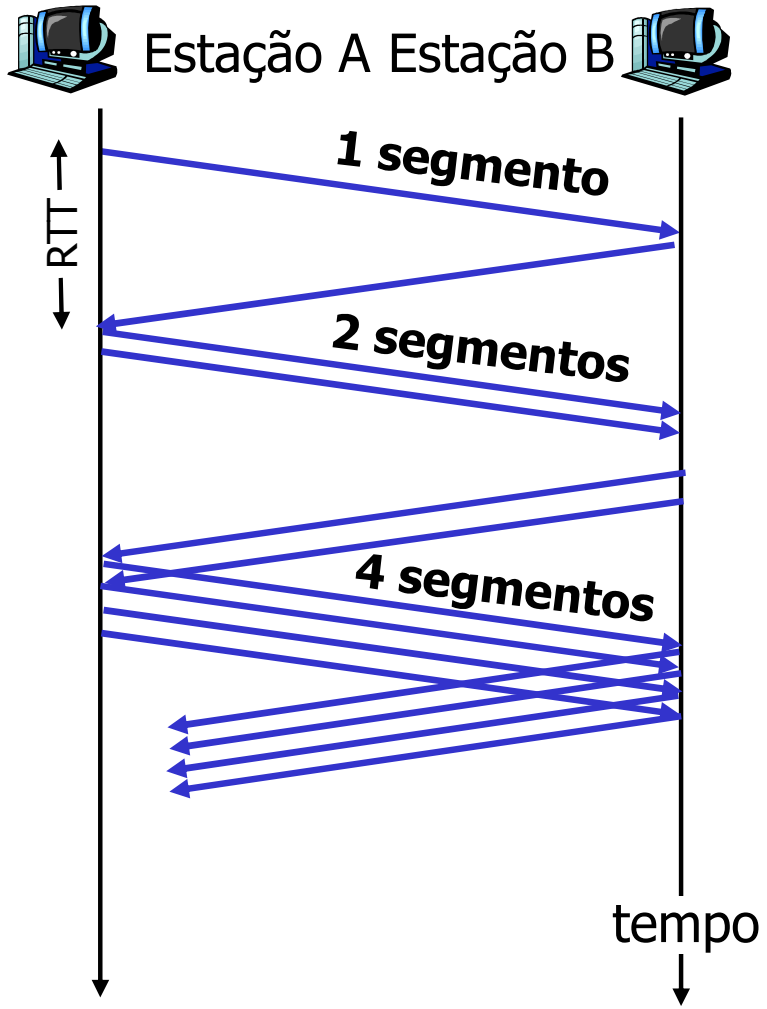
- No início da conexão: **`CongWin`** = 1 MSS
  - Exemplo: MSS = 500 bytes = 4000 bits e RTT = 200 ms
  - Taxa inicial = 20 kb/s
- Largura de banda disponível pode ser muito maior do que MSS/RTT
  - É desejável um crescimento rápido até uma taxa considerável

No início da conexão, a taxa aumenta exponencialmente até o primeiro evento de perda

Partida Lenta do TCP

- Duplica **`CongWin`** a cada RTT
- Através do incremento da **`CongWin`** para cada ACK recebido

taxa inicial é baixa, mas cresce rapidamente de forma exponencial



Controle de Congestionamento do TCP: Adaptações

Instituto de computação

- Após 3 ACKs duplicados
  - Corta **`CongWin`** pela metade
  - A janela depois cresce linearmente
- Porém, após estouro de temporizador
  - **`CongWin`** é reduzida a 1 MSS;
  - Janela cresce exponencialmente
  - Até um limiar, depois cresce linearmente

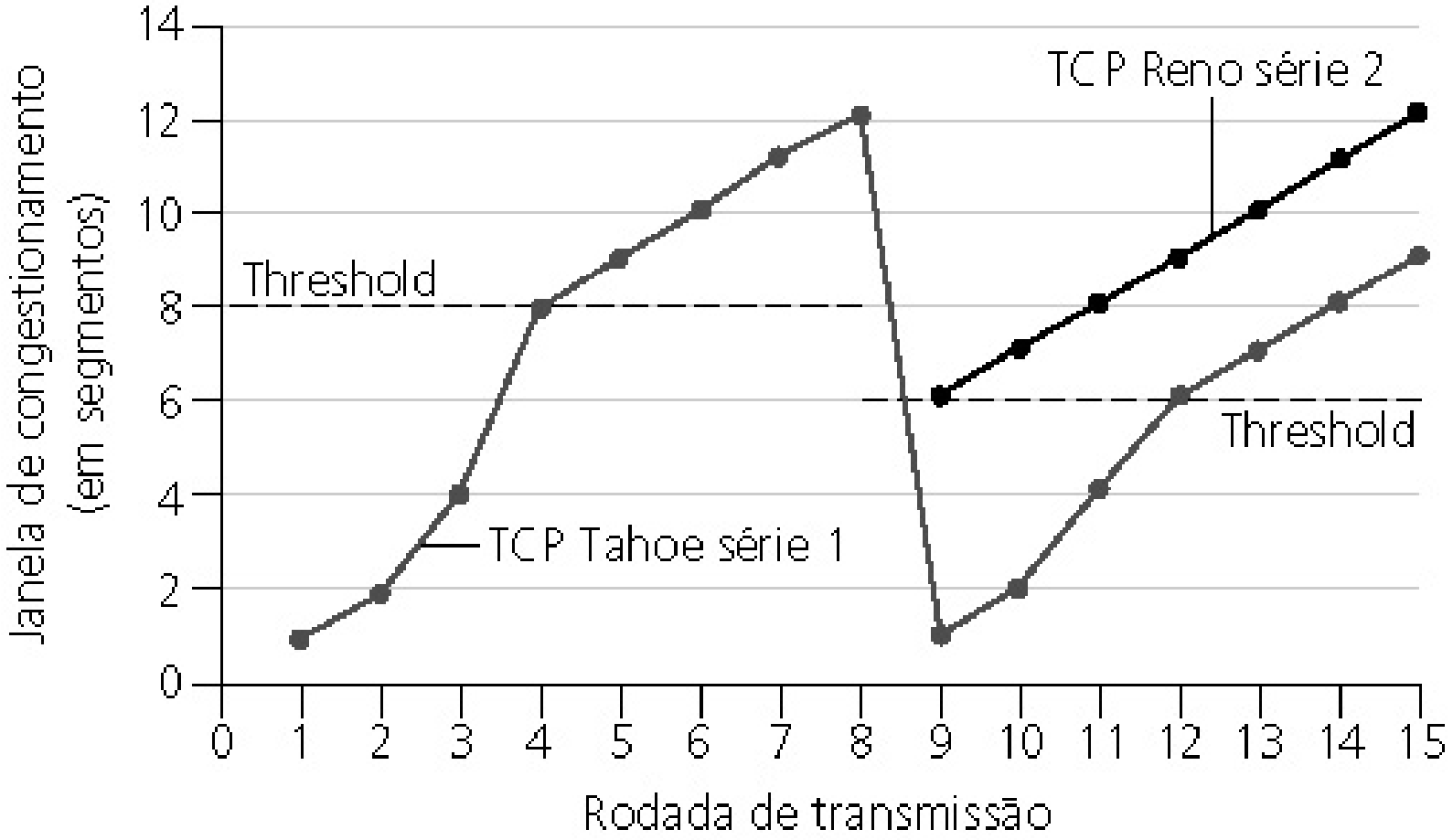
1. 3 ACKs duplicados indica que a rede é capaz de entregar alguns segmentos
2. Estouro de temporizador antes de 3 ACKs duplicados é mais "alarmante"

Controle de Congestionamento do TCP: Adaptações

Instituto de computação

- Quando o crescimento exponencial deve mudar para linear?
  - Quando **`CongWin`** atinge 1/2 do seu valor antes da detecção de perda
- Implementação
  - Limiar variável
  - Com uma perda o limiar é ajustado para 1/2 da **`CongWin`** imediatamente antes do evento de perda

Controle de Congestionamento do TCP: Adaptações



Controle de Congestionamento do TCP

Instituto de computação

- Quando a **`CongWin`** está abaixo do limiar
  - Transmissor está na fase de partida lenta
  - Janela cresce exponencialmente
- Quando a **`CongWin`** está acima do limiar
  - Transmissor está na fase de prevenção de congestionamento
  - Janela cresce linearmente
- Quando chegam três ACKs duplicados
  - Limiar passa a ser **`CongWin`**/2 e **`CongWin`** → valor do limiar
- Quando estoura o temporizador
  - Limiar é ajustado para **`CongWin`**/2 e **`CongWin`** → 1 MSS

Controle de Congestionamento do TCP

Evento	Estado	Ação do Transmissor TCP	Comentário
ACK recebido para dados ainda não reconhecidos	Partida lenta	CongWin = CongWin + MSS, Se (CongWin > Limiar) ajustar estado para "Prevenção de congestionamento"	Resulta na duplicação da CongWin a cada RTT
ACK recebido para dados ainda não reconhecidos	Prevenção de congestio-namento	CongWin = CongWin+MSS * (MSS/CongWin)	Aumento aditivo, resultando no incremento da CongWin de 1 MSS a cada RTT
Perda detectada por três ACKs duplicados	qualquer	Limiar = CongWin/2, CongWin = Limiar, Ajusta estado para "Prevenção de Congestionamento"	Recuperação rápida, implementa diminuição multiplicativa. CongWin não cai abaixo de 1 MSS.
Estouro de temporizador	qualquer	Limiar = CongWin/2, CongWin = 1 MSS, Ajusta estado para "Partida lenta"	Entra estado de "partida lenta"
ACK duplicado	qualquer	Incrementa contador de ACKs duplicados para o segmento que está sendo reconhecido	CongWin e Limiar não se alteram

Vazão do TCP

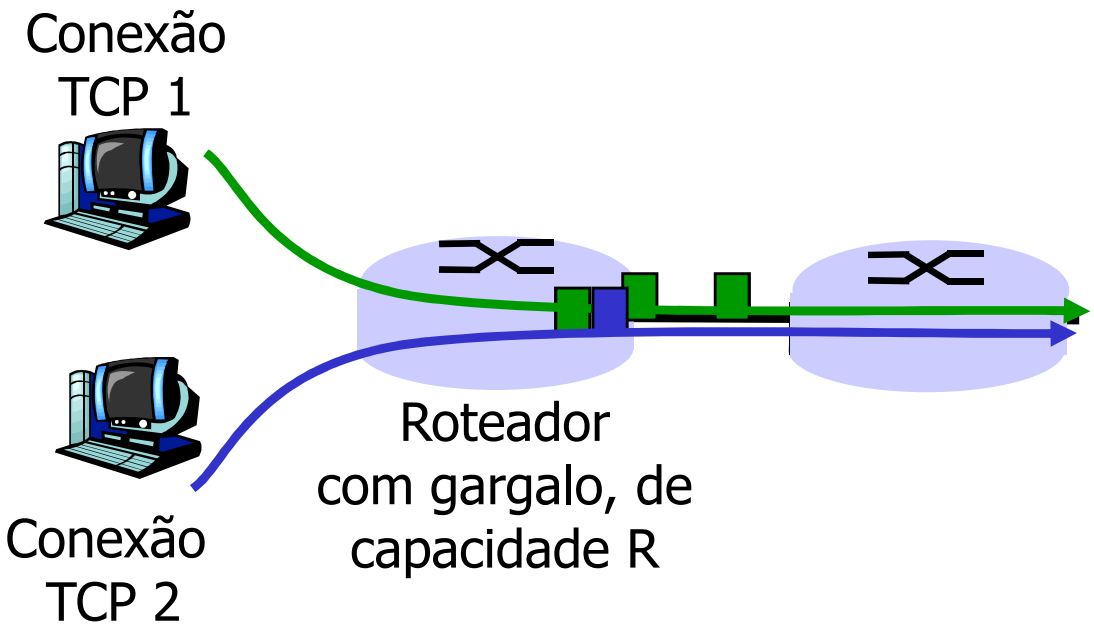
Instituto de computação

- Qual é a vazão média do TCP em função do tamanho da janela e do RTT?
  - Ignore a partida lenta

- Seja W o tamanho da janela quando ocorre a perda
- Quando a janela é W a vazão é W/RTT
- Logo após a perda, janela cai a W/2, e a vazão cai para W/2RTT
- Vazão média = 0,75 W/RTT

Justiça (*Fairness*) do TCP

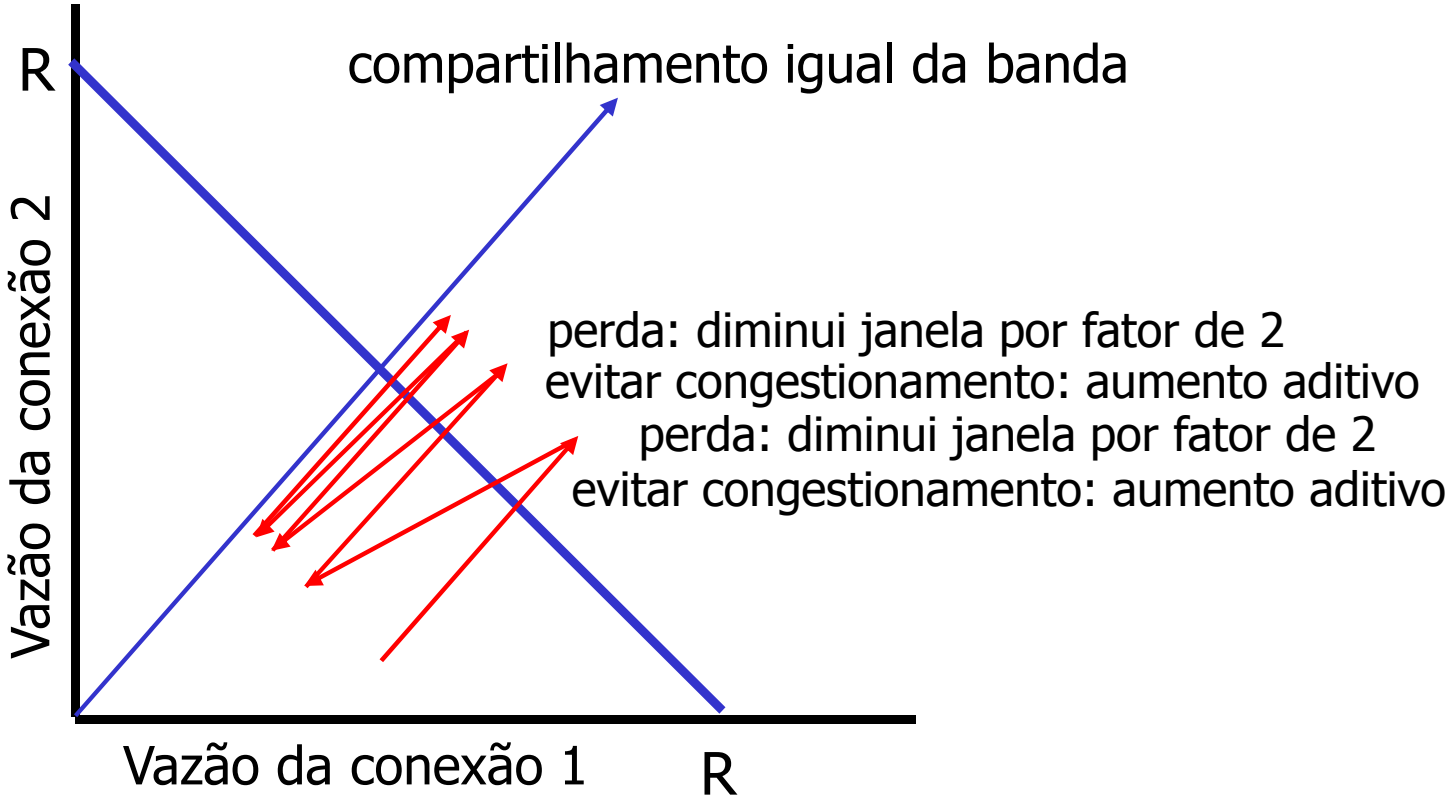
- Se *K* sessões TCP compartilham o mesmo enlace de gargalo com largura de banda *R*
  - Cada uma deve obter uma taxa média de *R*/*K*



Justiça do TCP

Duas sessões competindo pela banda:

- Aumento aditivo dá gradiente de 1, enquanto vazão aumenta
- Redução multiplicativa diminui vazão proporcionalmente



Justiça do TCP x UDP

Instituto de computação

- Aplicações multimídia freqüentemente não usam TCP
  - Não querem a taxa estrangulada pelo controle de congestionamento
  - Preferem usar o UDP
    - Injeta áudio/vídeo a taxas constantes, toleram perdas de pacotes
- Tornar as aplicações amigáveis ao TCP (*TCP friendly*)

Justiça x Conexões em Paralelo

Instituto de computação

- Aplicações podem abrir conexões paralelas entre dois sistemas finais
  - Os navegadores Web fazem isso

- Exemplo:
  - Canal com taxa R compartilhado por 9 conexões
    - Novas aplicações pedem 1 TCP → obtém taxa de R/10
    - Novas aplicações pedem 11 TCPs → obtém taxa R/2

Implementações

Instituto de computação

- Tahoe
  - Original
- Reno
- Vegas
- SACK
- NewReno (RFC 2582)
  - Usado no Windows Vista
- Bic
  - Usado pelo Debian (`/proc/sys/net/ipv4/tcp_congestion_control`)
- Etc.

Instituto de computação

Ambientes Desafiadores para o TCP

Redes de Alta Velocidade

Instituto de computação

- TCP
  - Permite o envio de W segmentos antes de receber um ACK
    - Janela de congestionamento
  - Cada segmento tem tamanho MSS

$$\text{vazão} = \frac{W \times MSS}{RTT} \text{ bytes/s}$$

Redes de Alta Velocidade

Instituto de computação

- Controle de congestionamento do TCP
  - Adaptar a taxa de envio de dados da fonte à carga da rede
  - Composto pelos mecanismos de
    - Partida lenta
    - Prevenção de congestionamentos

Redes de Alta Velocidade

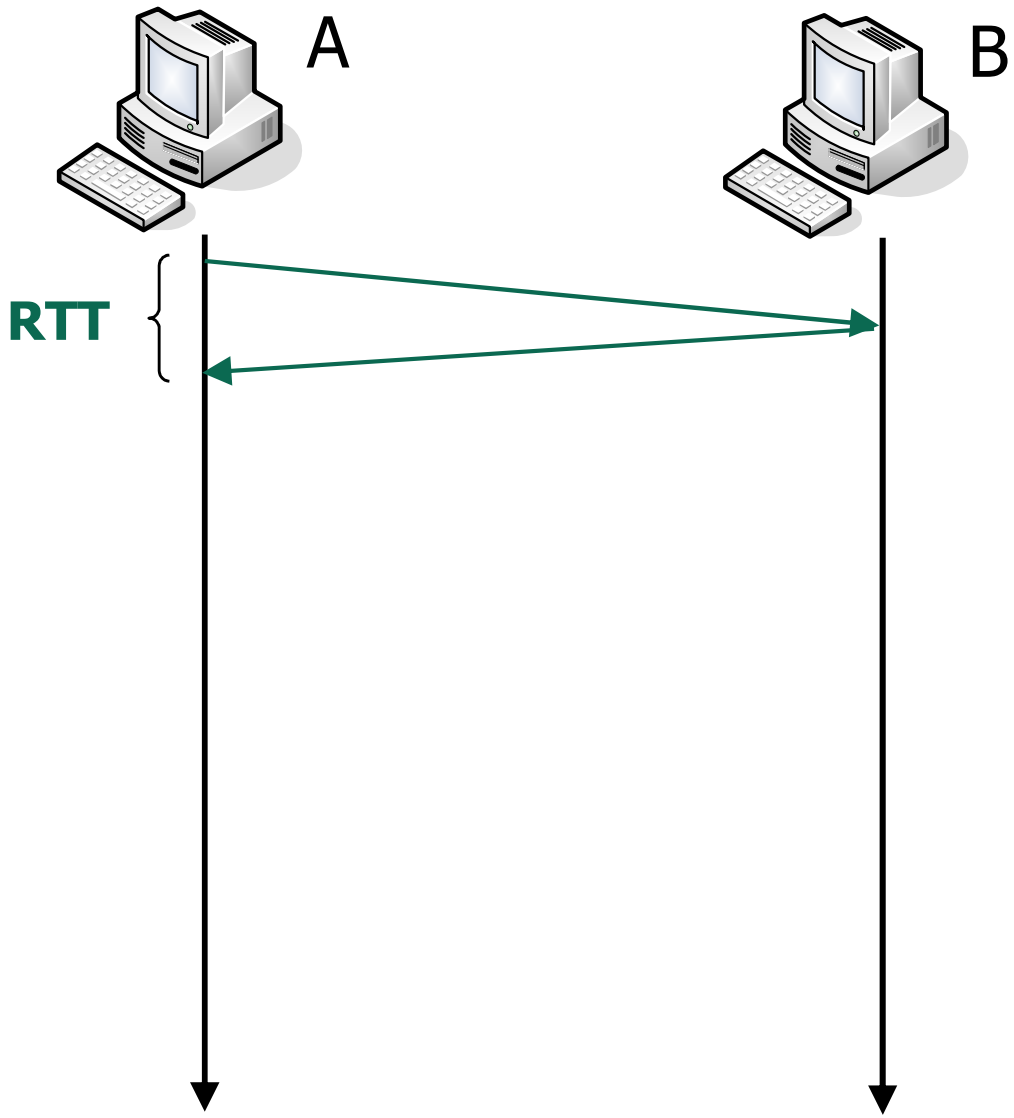
Instituto de computação

- Partida lenta
  - Usado no início de uma conexão
  - Fazer com que a taxa de transmissão da fonte convirja **lentamente** para a capacidade de transmissão da rede
  - Tamanho da janela
    - Inicialmente: W=1
    - Aumenta de 1 segmento a cada ACK recebido/RTT

crescimento exponencial

Partida Lenta

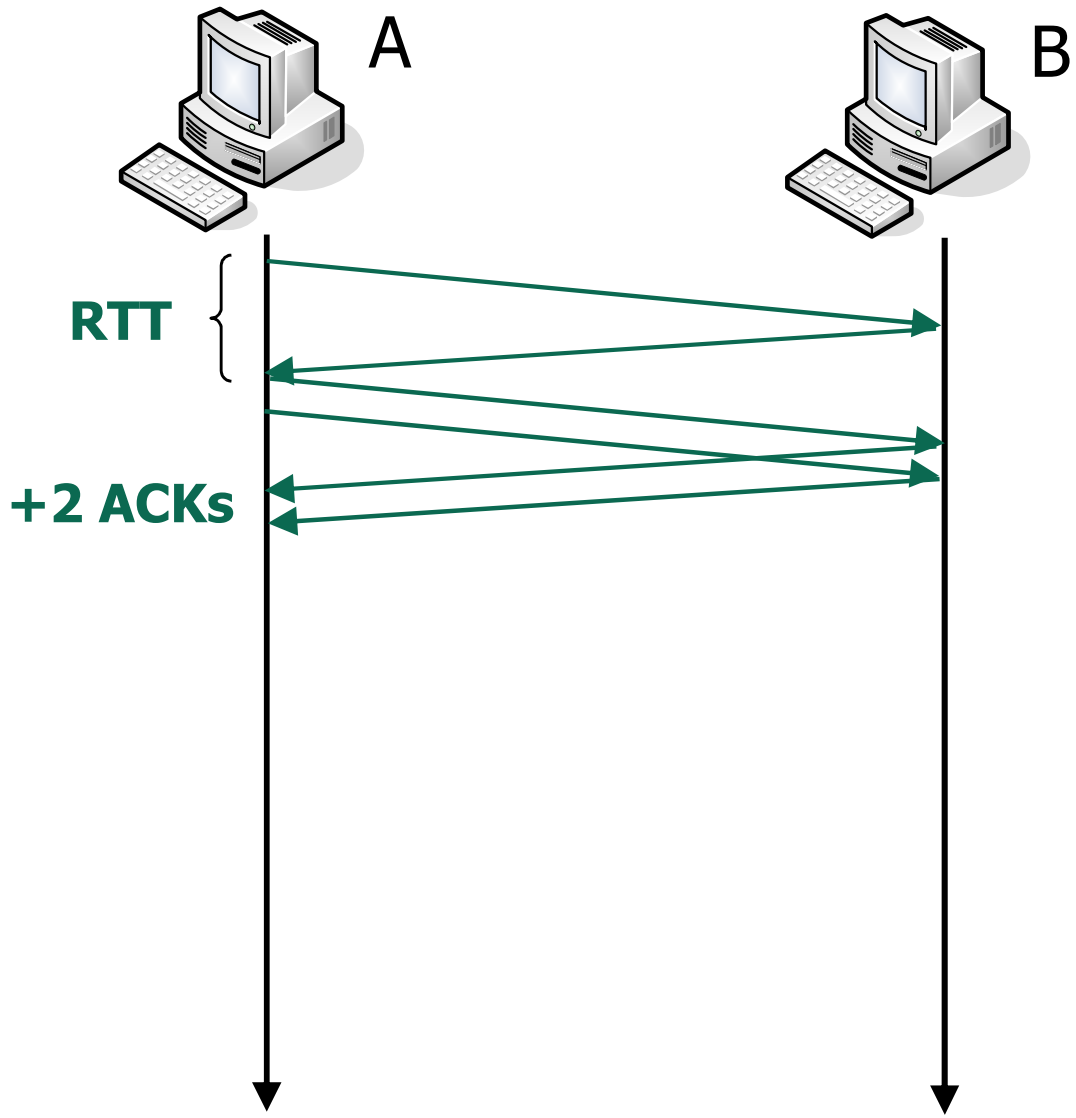
$W=1$



Partida Lenta

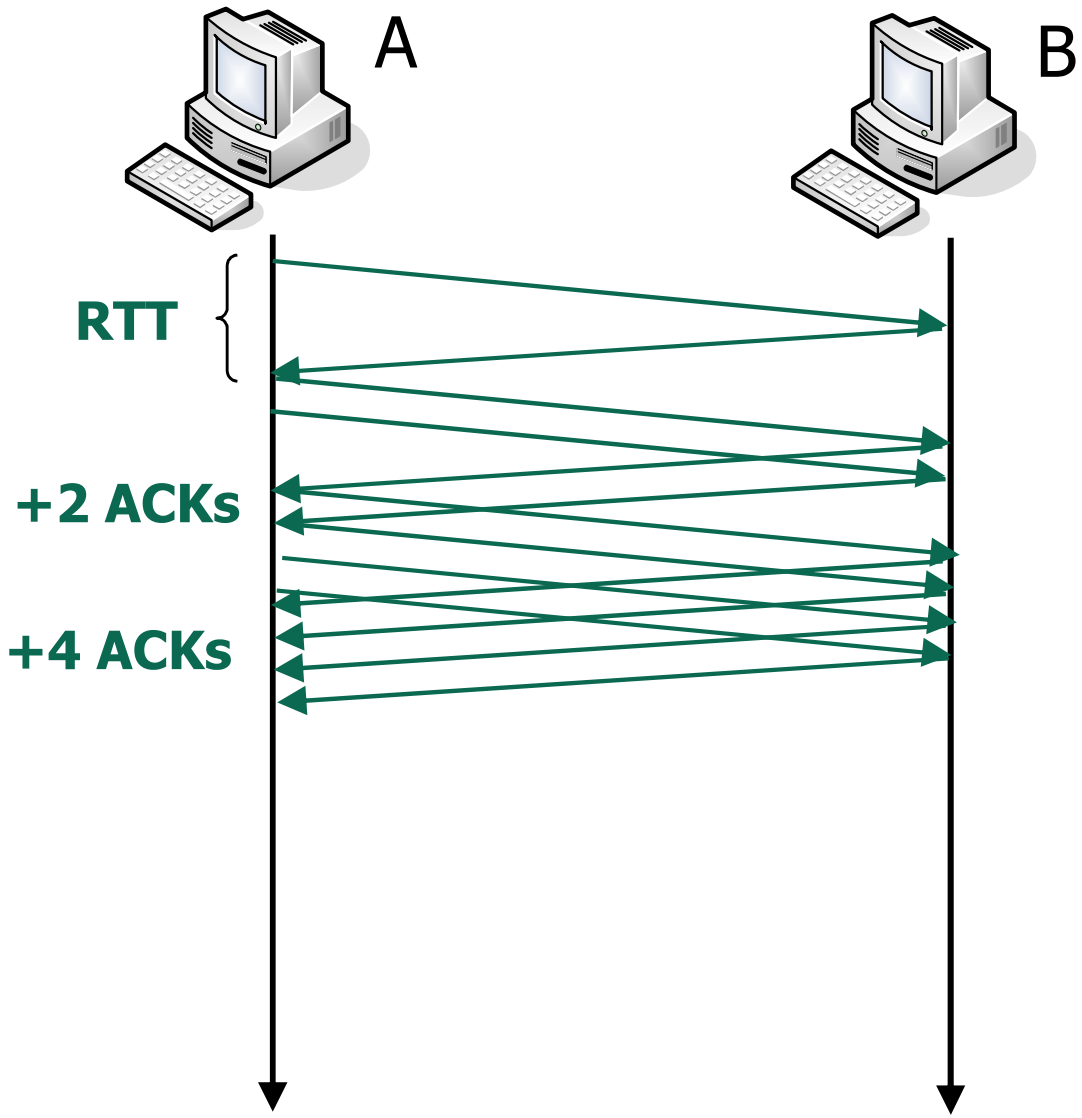
Instituto de computação

$W=2$



Partida Lenta

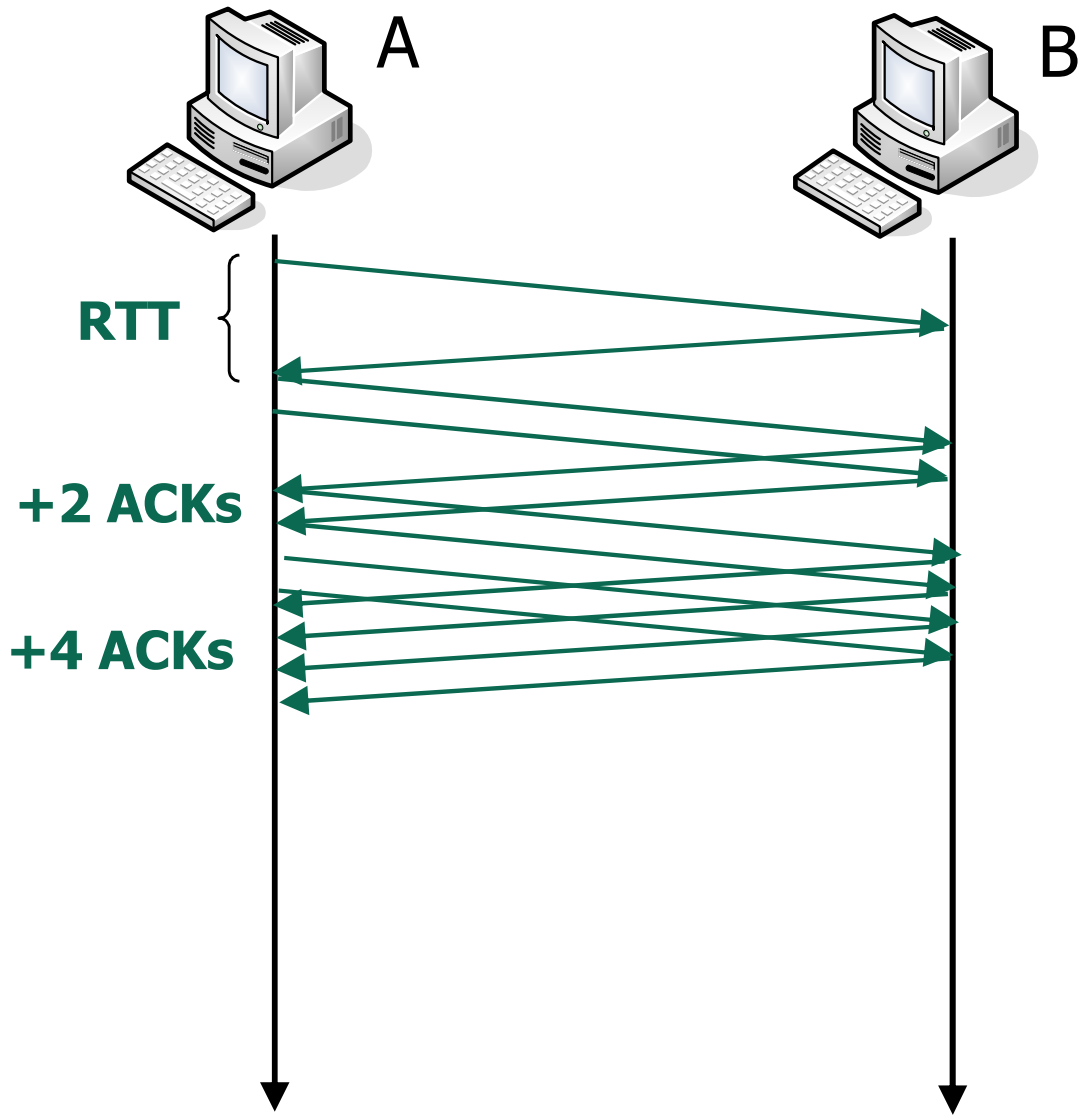
W=4



Partida Lenta

$W=8$

O crescimento da janela é exponencial até um limiar ou quando ocorre a perda de um segmento



Controle de Congestionamento do TCP

Instituto de computação

- Quando a janela está abaixo do limiar
  - Transmissor está na fase de partida lenta
  - Janela cresce exponencialmente

$$W = W + 1$$

- Quando a janela está acima do limiar
  - Transmissor está na fase de prevenção de congestionamento
  - Janela cresce linearmente

$$W = W + (a/W),\ a=1$$

Controle de Congestionamento do TCP

Instituto de computação

- Quando chegam três ACKs duplicados
  - Limiar passa a ser 0,5*janela e janela recebe o valor do limiar

$$W = W - (b*W),\ b=0,5$$

- Quando estoura o temporizador
  - Limiar passa a ser 0,5*janela e janela recebe o valor inicial (W=1)

Controle de Congestionamento do TCP

Instituto de computação

- Quando chegam três ACKs duplicados
  - Limiar passa a ser 0,5*janela e janela recebe o valor do limiar

$$W = W - (b*W),\ b=0,5$$

- Quando estoura o temp
  - Limiar passa a recebe o valor inicial (W=1)

Limita o tamanho máximo da janela

Redes de Alta Velocidade

Instituto de computação

- Exemplo
  - Segmentos de 1500 bytes e RTT=100 ms,
  - Vazão desejada de 10 Gb/s

$$\underbrace{\text{vazão*RTT}}_{\text{produto banda x latência}} = W \times MSS \text{ bytes}$$

  - Requer janela de W = 83.333 segmentos em trânsito

Redes de Alta Velocidade

Instituto de computação

- Vazão em termos de taxa de perdas

$$\frac{1{,}22 \cdot MSS}{RTT\sqrt{L}}$$

- Para o exemplo:
  - L= 2 x 10^(-10) → **taxa de perdas tem que ser muito baixa para "encher" o meio!**

Novas versões do TCP para alta velocidade

Protocolos de Alta Velocidade

Instituto de computação

- Novos protocolos de transporte para redes gigabit propostos
  - XCP (*Explicit Congestion Control Protocol*)
  - HSTCP (*HighSpeed TCP*)
  - STCP (*Scalable TCP*)
  - FAST TCP (*Fast Active-queue-management Scalable TCP*)

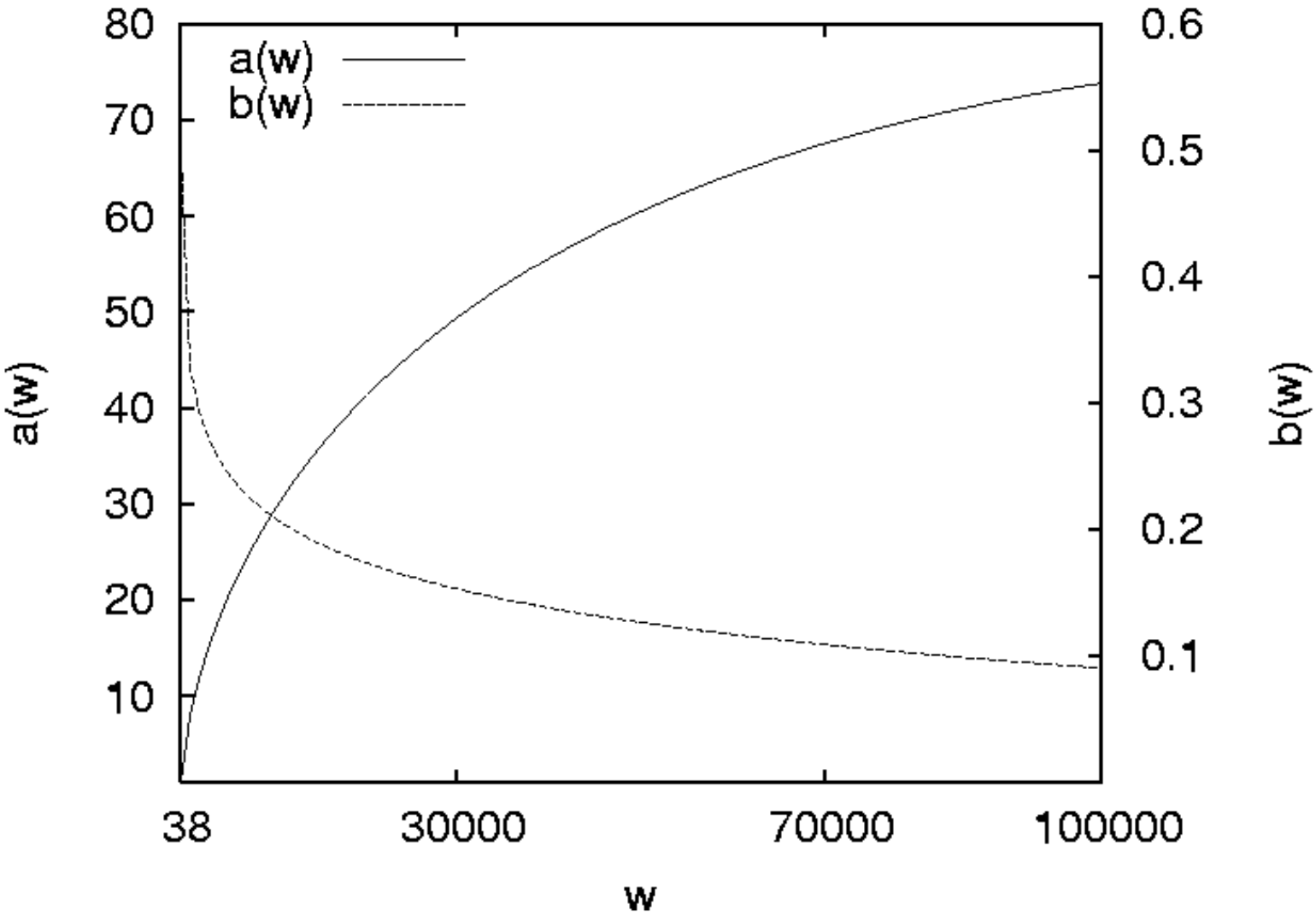
HSTCP

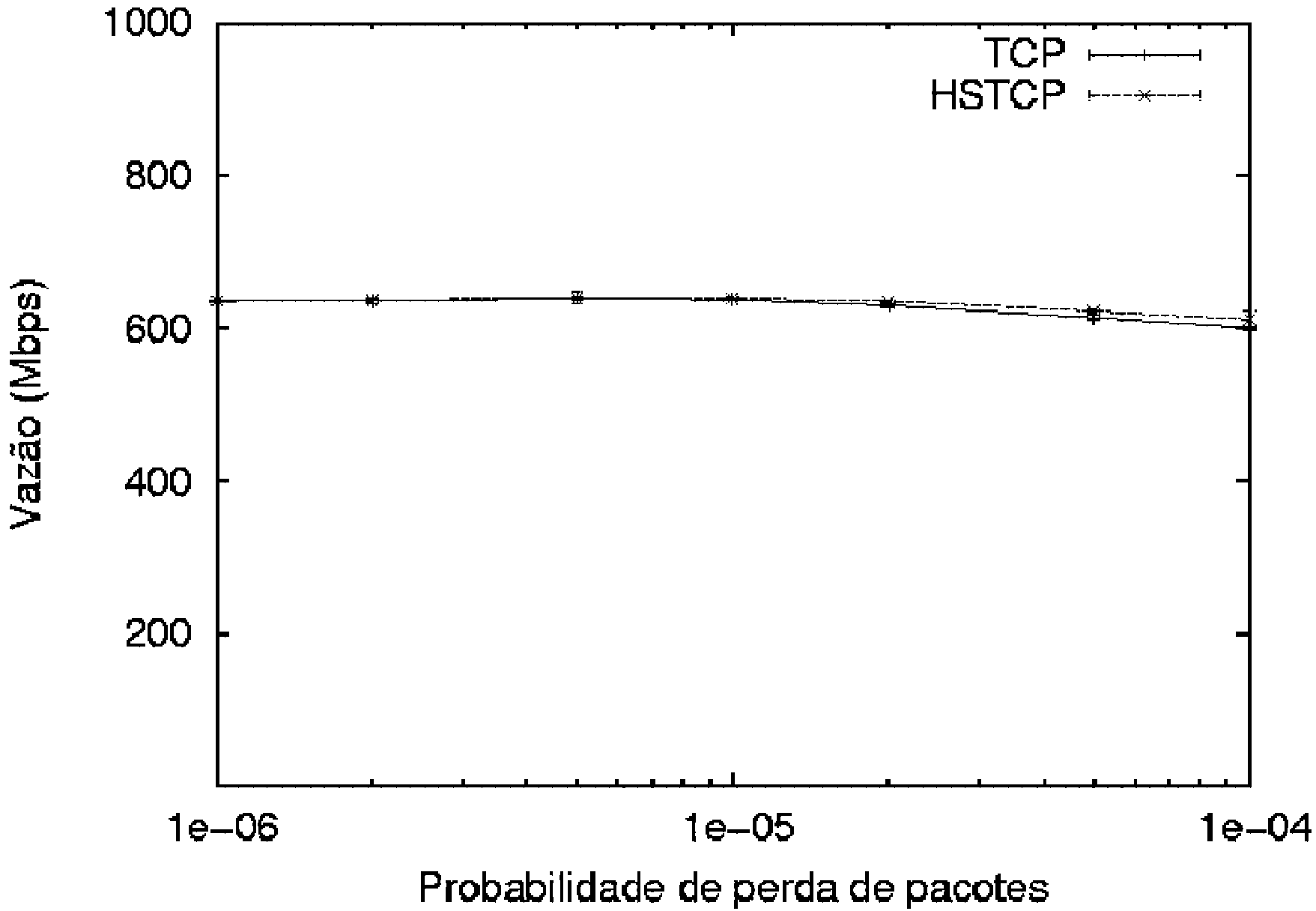
Instituto de computação

- Modificação do TCP
  - Muda o crescimento e a diminuição da janela de congestionamento para conexões com grandes janelas
    - Partida lenta é mantida
- Parâmetros *a* e *b* passam a ser função do tamanho da janela
  - Aumenta-se o valor de *a* para grandes valores de *w*
    - Maior vazão pode ser atingida mesmo para uma taxa de perdas significativa
  - Utiliza-se menores valores de *b* para grandes valores de *w*
    - Diminuição de *w* não seja muito grande com a perda de um segmento

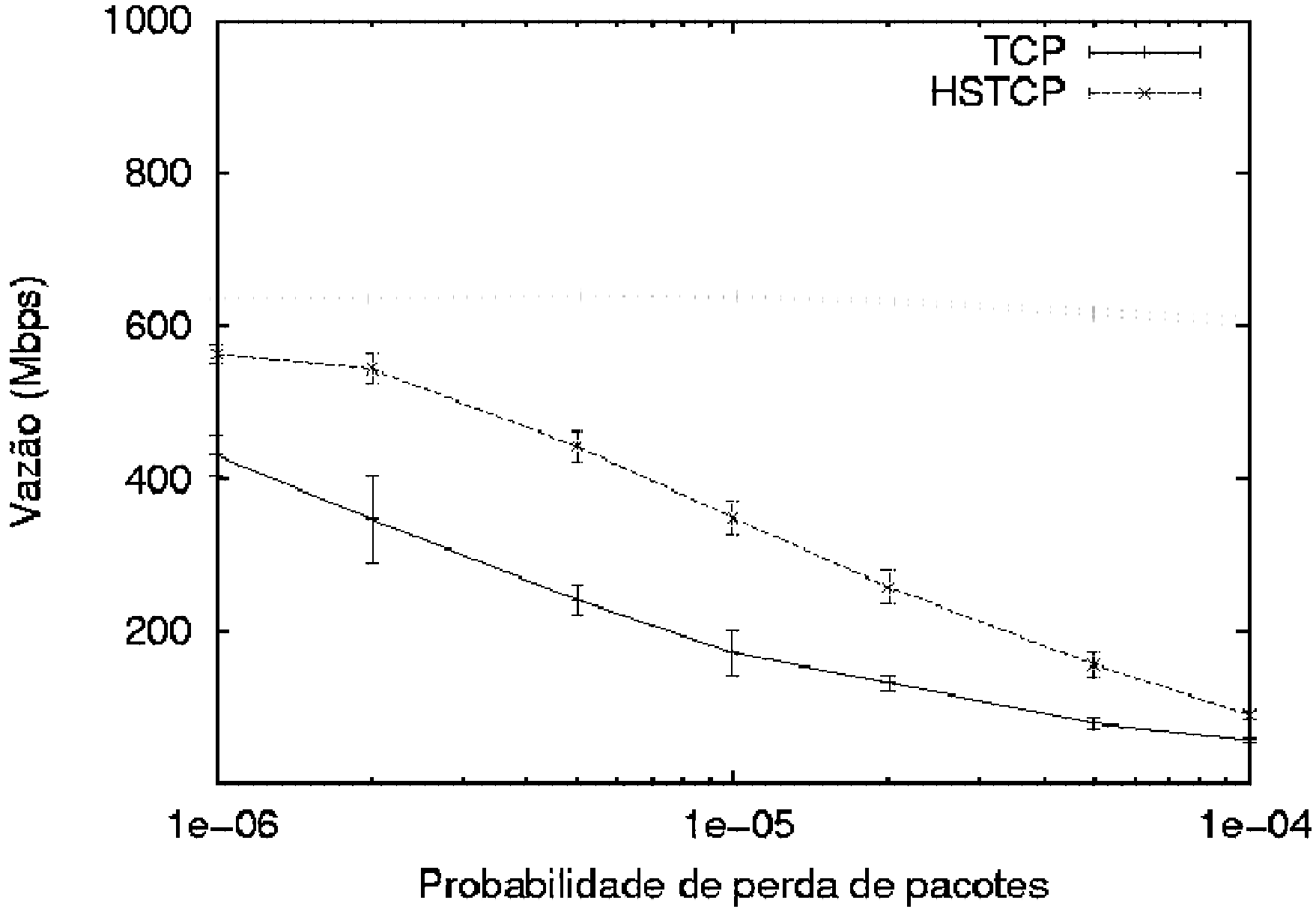
HSTCP

Exemplos de *a* e *b* no HSTCP

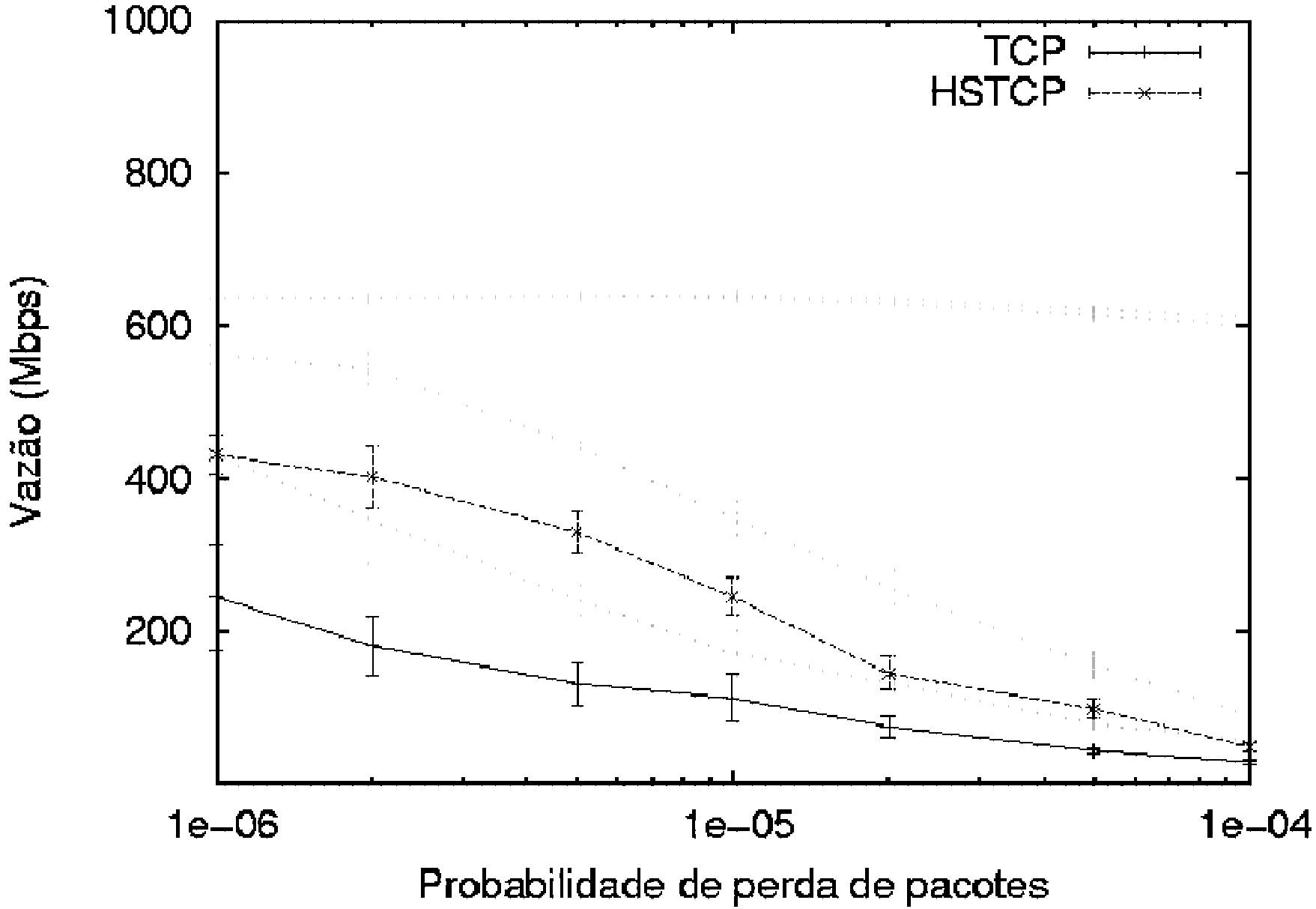



Experimentos - Latência igual a 0 ms
1000
800
600
400
200
Vazão (Mbps)
TCP
HSTCP
1e−06
1e−05
1e−04
Probabilidade de perda de pacotes

Experimentos - Latência igual a 20 ms



Experimentos - Latência igual a 40 ms



HSTCP

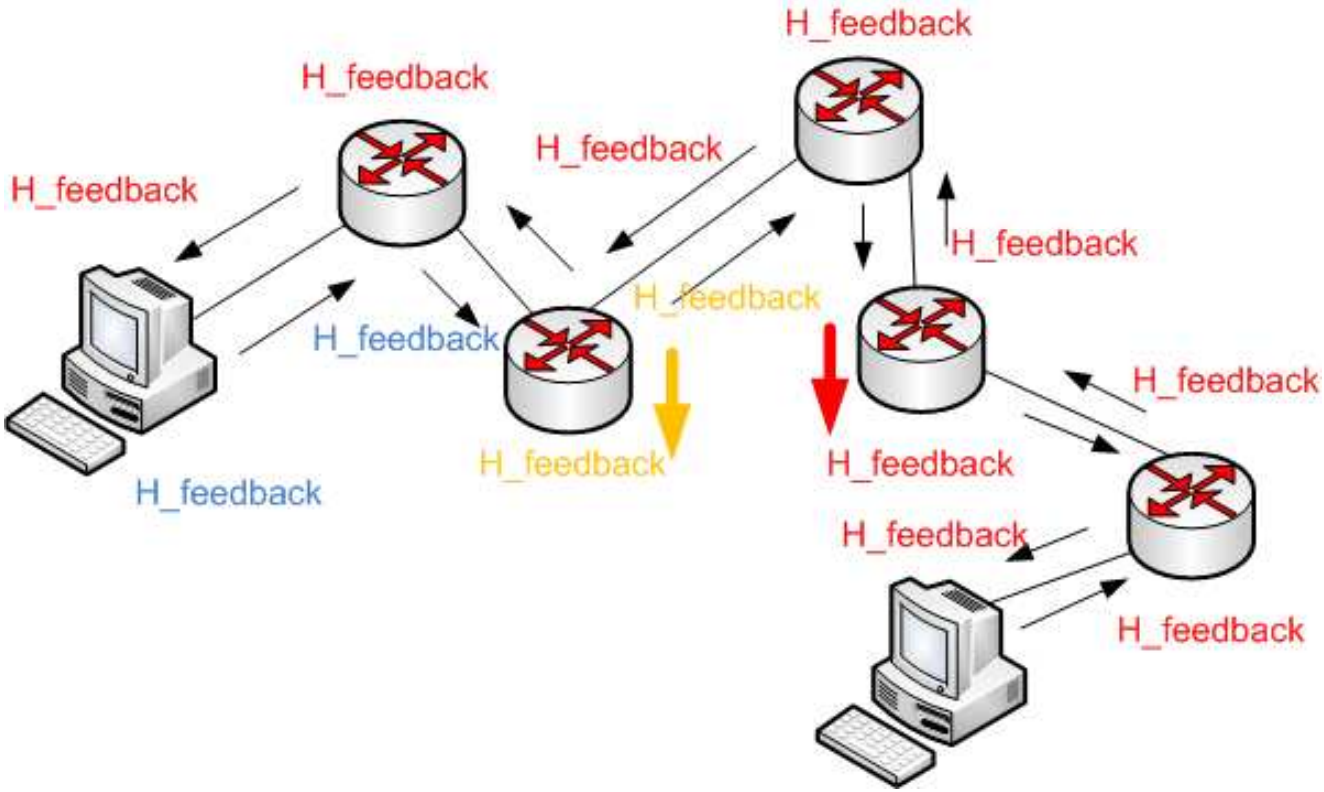
Instituto de computação

- Vantagem do HSTCP em relação ao TCP é maior para RTTs altos
- Para taxa de perdas maiores que 10^{-4} os dois protocolos tendem à mesma vazão independente do RTT
- Uso de várias conexões TCP faz com que o a vazão agregada se aproxime da vazão agregada do HSTCP

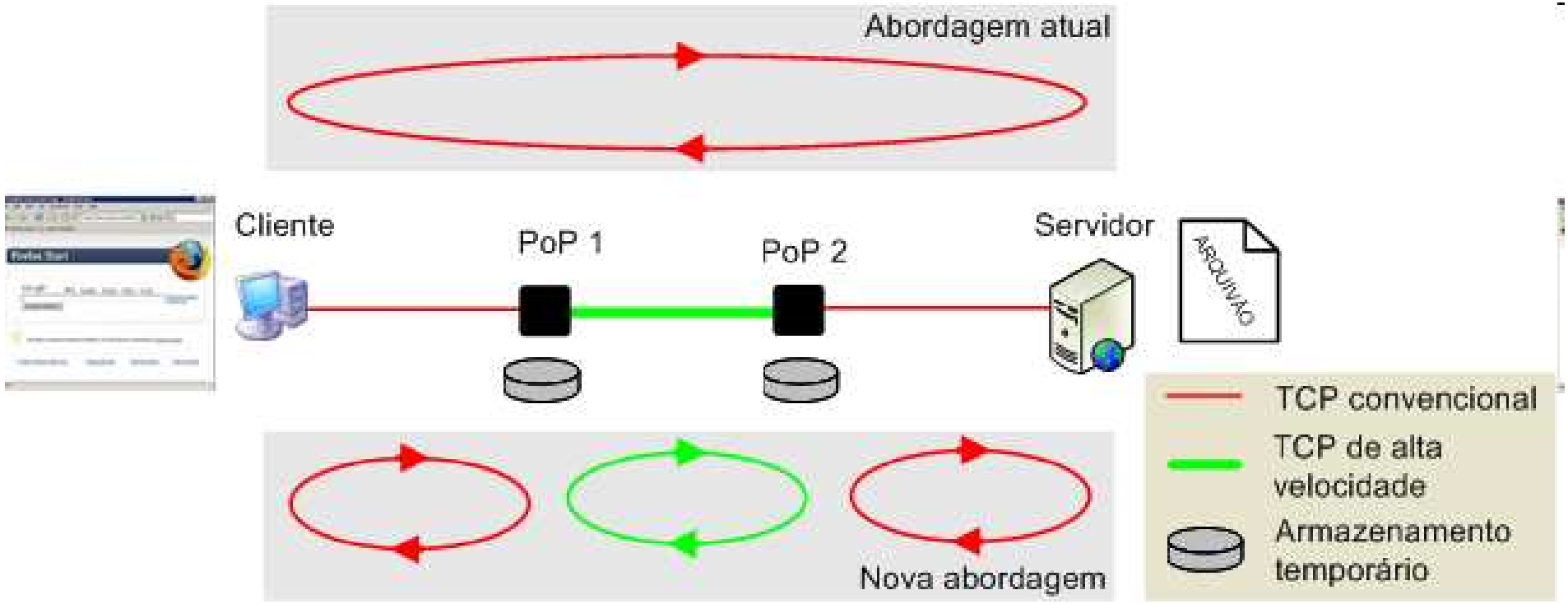
XCP

Instituto de Computação

- Exige modificações nos roteadores da rede



Quebra de Conexões



Quebra de Conexões

Instituto de computação

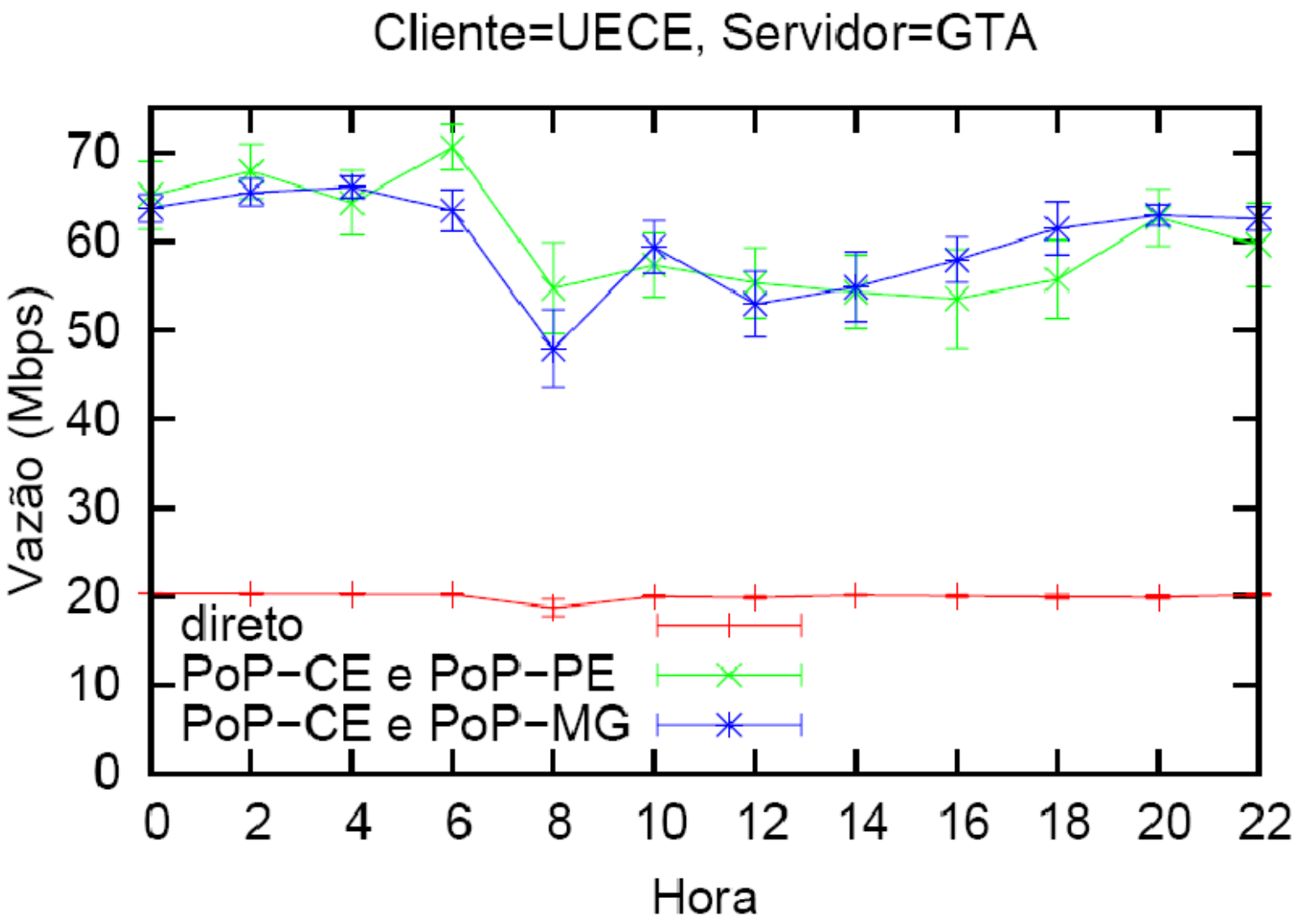
- Quebrar a semântica fim-a-fim da conexão TCP
  - Estabelecer conexões em série
  - Emprego de protocolos de transporte de alta velocidade
    - Em determinados enlaces

Quebra de Conexões

Instituto de computação

- Várias conexões
  - Cada conexão individual tem um RTT inferior ao da comunicação fim-a-fim
  - Cada uma ocupa de forma mais eficiente a capacidade dos enlaces que atravessam
    - Sinalização de congestionamento em malha fechada apresenta uma menor latência.
  - Transferência é realizada em paralelo entre as conexões que compõem a comunicação fim-a-fim
    - É possível aumentar ainda mais a vazão obtida

Quebra de Conexões



Redes Sem-Fio

Instituto de computação

- TCP supõe que as perdas são causadas por congestionamentos
  - Válido para redes cabeadas
- Em redes sem fio as perdas por variações da qualidade do meio de transmissão também são frequentes

Não há distinção entre as perdas

Janela de congestionamento é reduzida quando não deveria

Redes Sem-Fio

Instituto de computação

- TCP supõe que as perdas são causadas por congestionamentos
  - Válido para redes cabeadas
- Em redes sem fio as perdas por variaçõ dade do meio de transmissão também são

Como distinguir as perdas?

Não h ção entre as perdas

Janela de congestionamento é reduzida quando não deveria

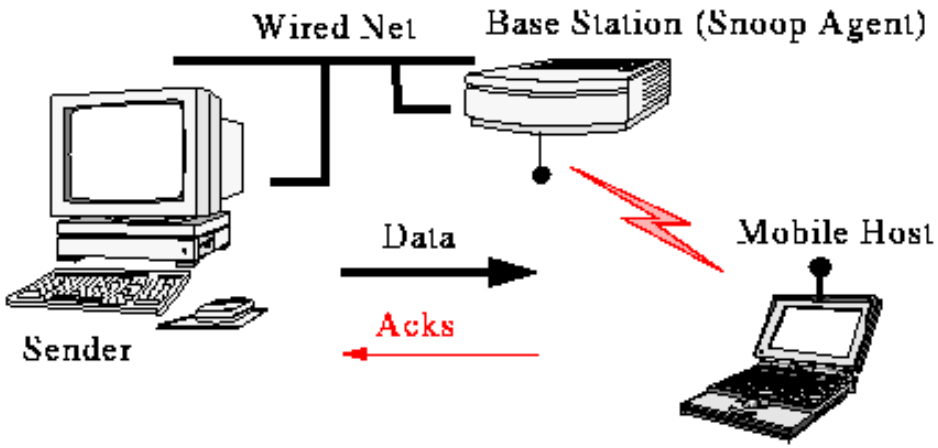
Redes Sem-Fio

Instituto de computação

- Duas abordagens
  - Quebra de conexão
  - Comunicação entre camadas (*cross layer*)

Snoop

- Introduz um agente
  - Localizado no ponto de acesso sem fio
  - Implementa um mecanismo de retransmissão local
    - Armazena os segmentos para os quais não foi enviado ACK
    - Não encaminha para a fonte ACKs duplicados
      - Fonte não reduz a taxa



Comunicação entre Camadas

Instituto de computação

- Usar as informações da camada de rede para notificar explicitamente a fonte sobre a natureza da perda
  - Usar o bit ECN já existente
  - TCP-Feedback
  - Ad-hoc TCP (ATCP)
  - Etc.

Instituto de computação

Programa de Pós-Graduação
Instituto de Computação
Universidade Federal Fluminense

Aulas 11, 12 e 13

Camada de Transporte

Conceitos, protocolos UDP e TCP

Igor Monteiro Moraes

Redes de Computadores