Redes de Computadores

Camada de Transporte

Prof. Everthon Valadão

INSTITUTO FEDERAL
MINAS GERAIS

Material baseado nos slides de: Dorgival G. (UFMG), Fábio C. (UFG), Kurose & Ross (Pearson)

(última modificação: 01/10/2020)

Camada de transporte

Nosso objetivo:

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

Tópicos abordados:

- serviços e protocolos
- transporte sem conexão: UDP
- princípios de transferência confiável de dados
- transporte orientado à conexão: TCP
- princípios de controle de congestionamento

Camada de transporte

Nosso objetivo:

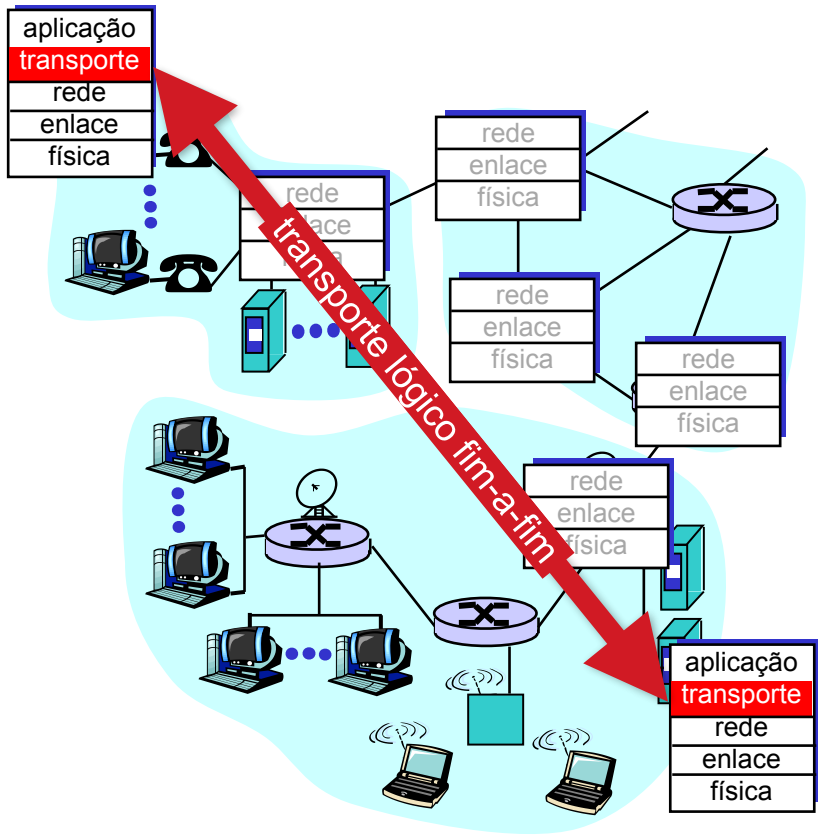
- entender os princípios por trás dos serviços da camada de transporte:
  - multiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento

Tópicos abordados:

- serviços e protocolos
- transporte sem conexão: UDP
- princípios de transferência confiável de dados
- transporte orientado à conexão: TCP
- princípios de controle de congestionamento

Serviços e protocolos de transporte

- Comunicação lógica entre aplicações em diferentes hosts
- Serviços executados nos sistemas finais
- Utilizam serviço da camada de rede:
  - transferência de dados entre computadores (passo-a-passo)



Serviços e protocolos de transporte

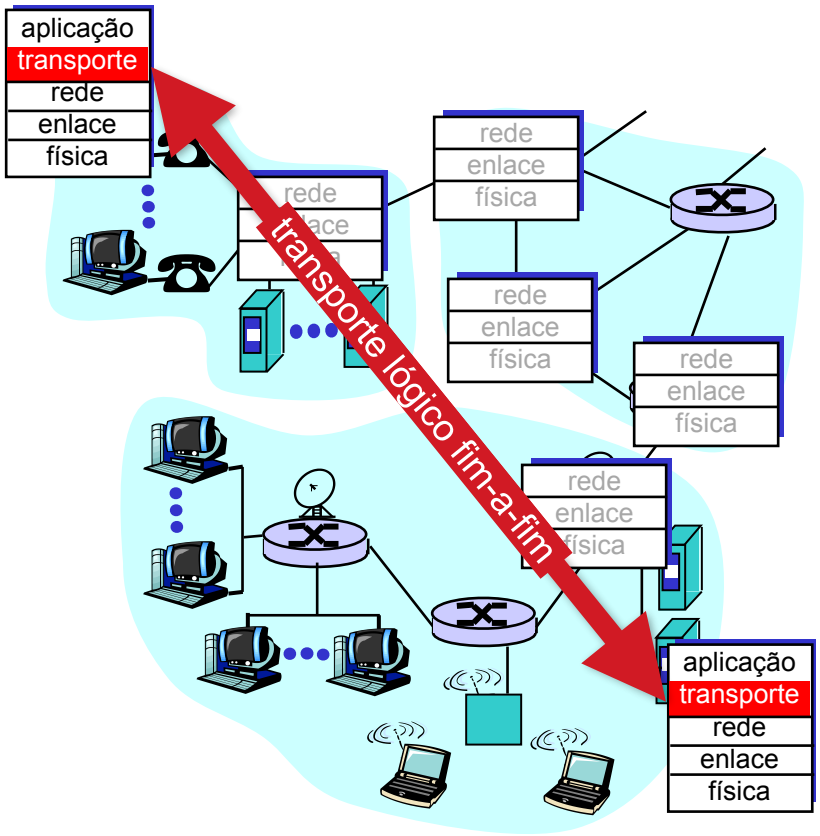
Serviços de transporte na internet:

- **TCP**: confiável, sequencial e *unicast*
  - *orientado a conexões*
  - *controle de fluxo*
  - congestionamento (“congestão”)

- **UDP**: não confiável (“*melhor esforço*”), não seqüencial, entrega *unicast* ou *multicast*

Serviços não disponíveis na internet:

- tempo-real
- garantia de banda
- *multicast* confiável



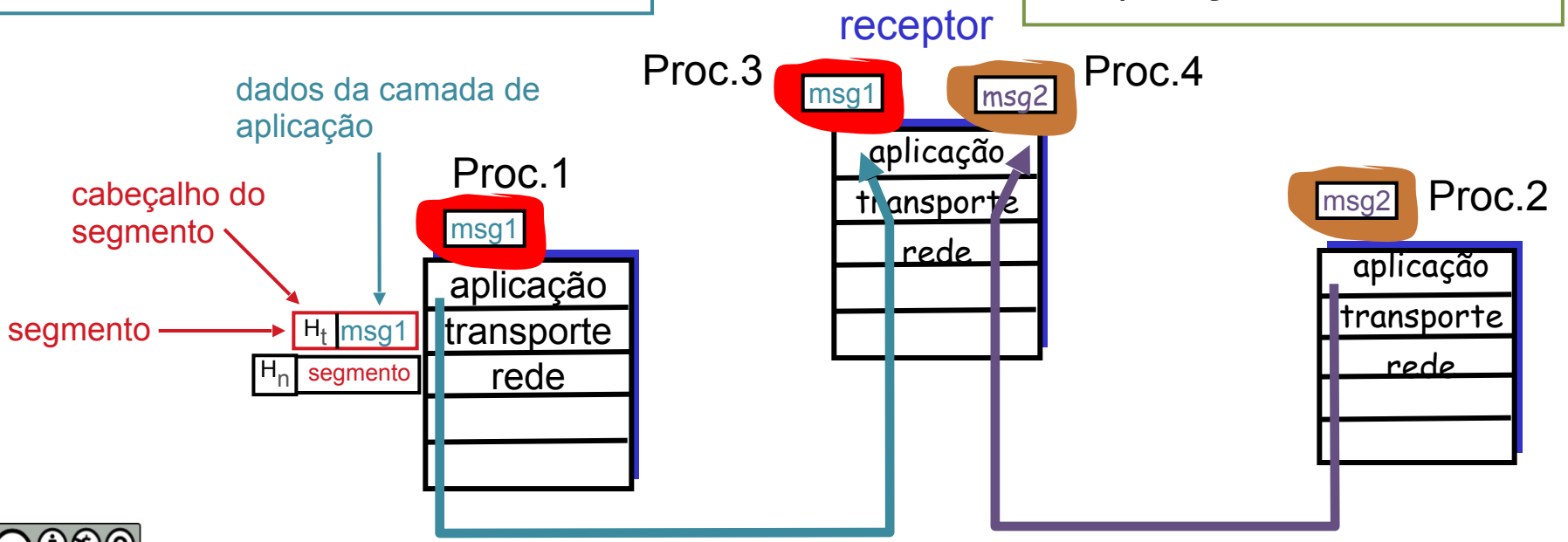
Multiplexação de aplicações

Multiplexação:

reunir dados de múltiplos processo de aplicação, juntar cabeçalhos com informações

Demultiplexação:

analisar cabeçalhos, entregar segmentos recebidos aos processos de aplicação corretos



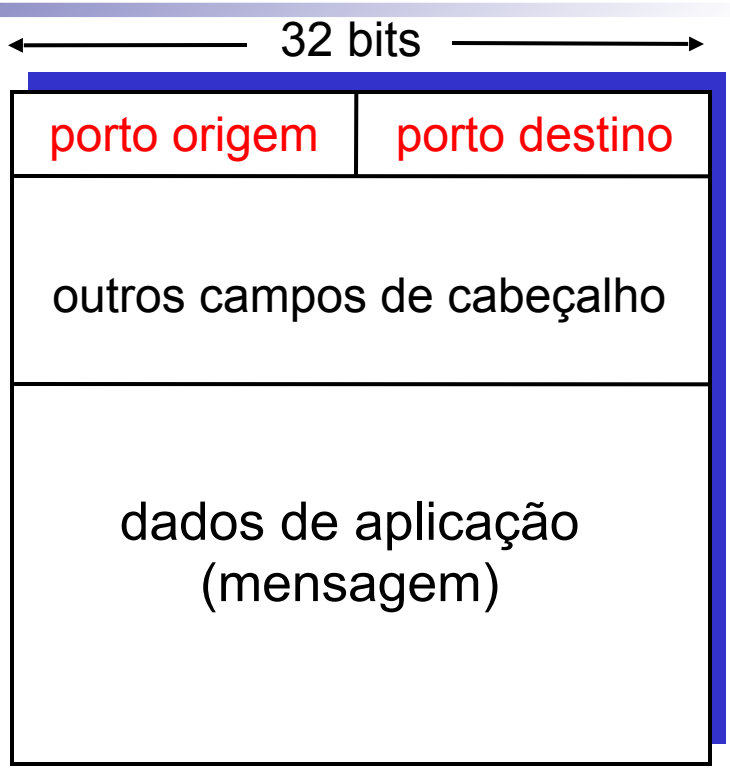
Multiplexação de aplicações

- Multiplexação é baseada em:
  - nº de porto do transmissor
  - nº de porto do receptor
  - respectivos endereços IP

OBS.: todo e cada segmento contém os números de porto da origem e do destino

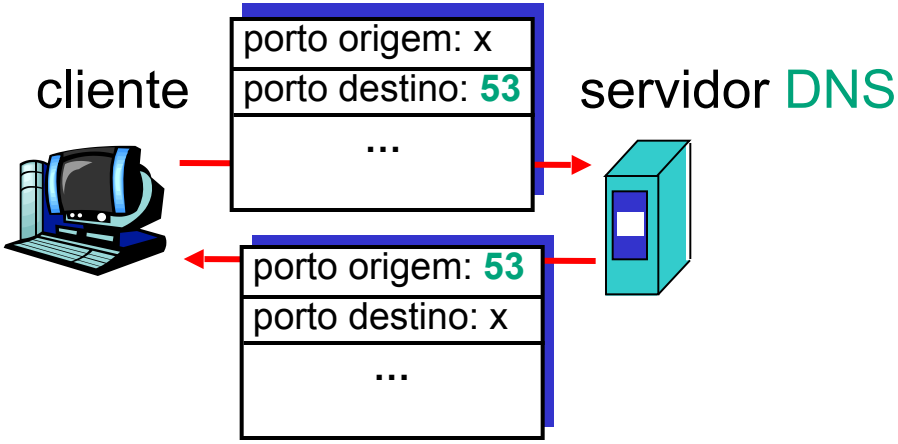
Atenção: portos com números bem-conhecidos são reservados para aplicações específicas

- os sistemas operacionais não permitem ao usuário comum utilizar porto com valor abaixo de 1024, ex.: 80 (HTTP), 53 (DNS), 25 (SMTP), 22 (SSH), 21 (FTP), etc.



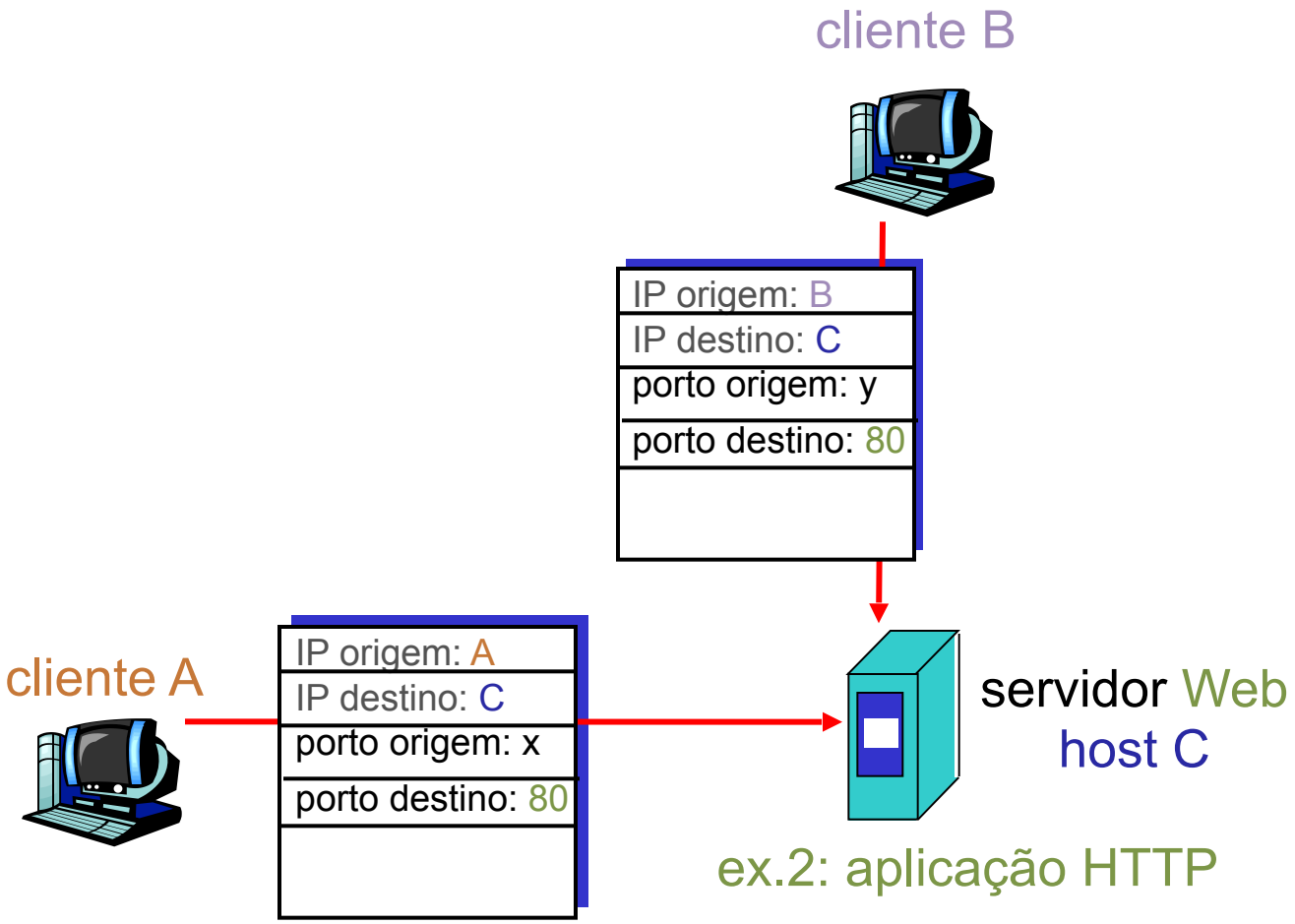
formato comum
de pacote TCP/UDP

Exemplos de multiplexação

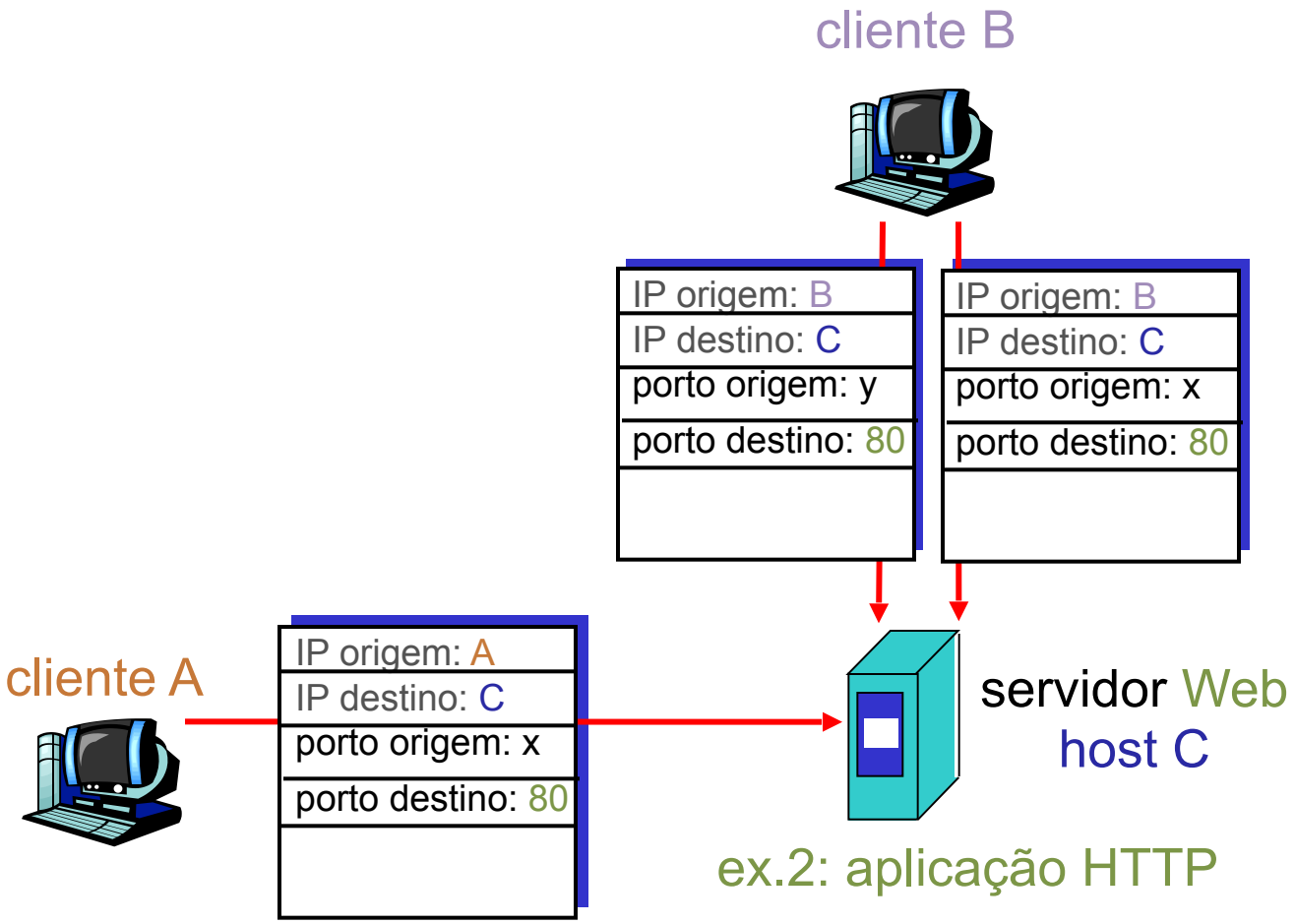


ex.1: aplicação DNS

Exemplos de multiplexação



Exemplos de multiplexação



Camada de transporte

Nosso objetivo:

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

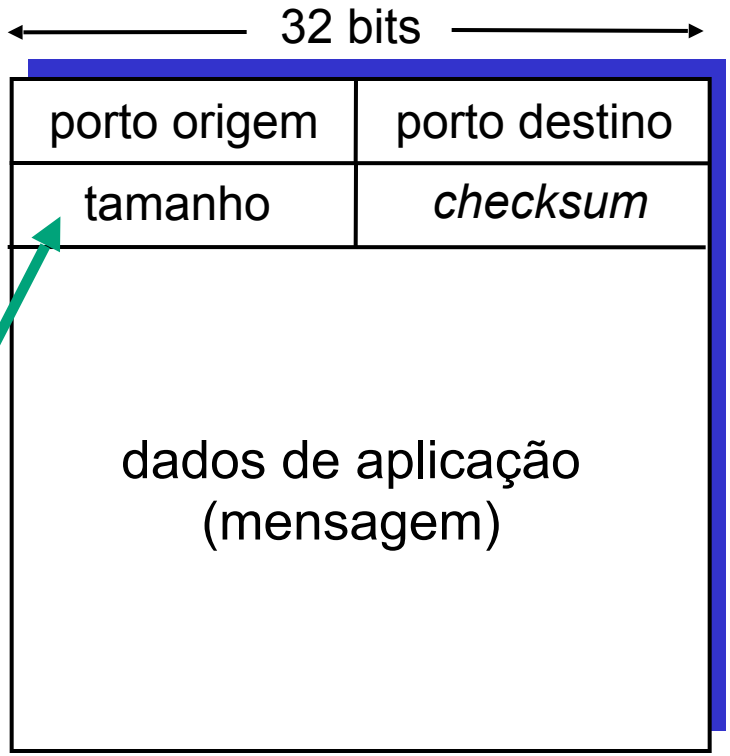
Tópicos abordados:

- serviços e protocolos
- transporte sem conexão: UDP
- princípios de transferência confiável de dados
- transporte orientado à conexão: TCP
- princípios de controle de congestionamento

UDP (*User Datagram Protocol*)

UDP é um protocolo de transporte básico, simples e rápido

- sem conexão
  - não há *"aperto de mão"* inicial
  - cada datagrama é independente dos demais
- serviço de "*melhor esforço*": tenta entregar, mas datagramas podem ser
  - perdidos
  - entregues fora de ordem
  - entregues com erros
- usos: multimídia, request-reply (ex.: DNS), UI interativas, etc



formato do datagrama UDP

OBS.: tamanho total em bytes do datagrama, incluindo o cabeçalho

Checksum UDP

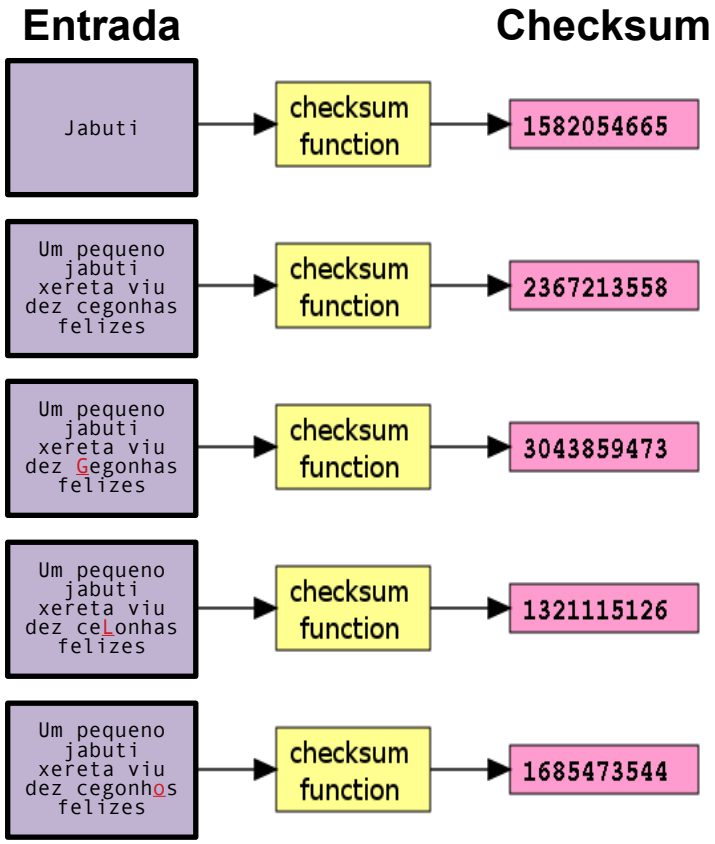
Objetivo: detectar erros nos dados transmitidos

Transmissor:

- trata o conteúdo do segmento como inteiros de 16 bits
- calcula a soma do conteúdo do datagrama e faz o complemento de 1 no resultado (inverte os bits)
- coloca o valor calculado no campo de *checksum*

Receptor:

- computa o *checksum* do conteúdo do datagrama recebido
- soma o *checksum* ***re-calculado*** com o *checksum* ***original*** do datagrama e, se o resultado de C_1+C_0 for:
  - ZERO → não há erros detectados
  - OUTRO → algum erro foi detectado



Camada de transporte

Nosso objetivo:

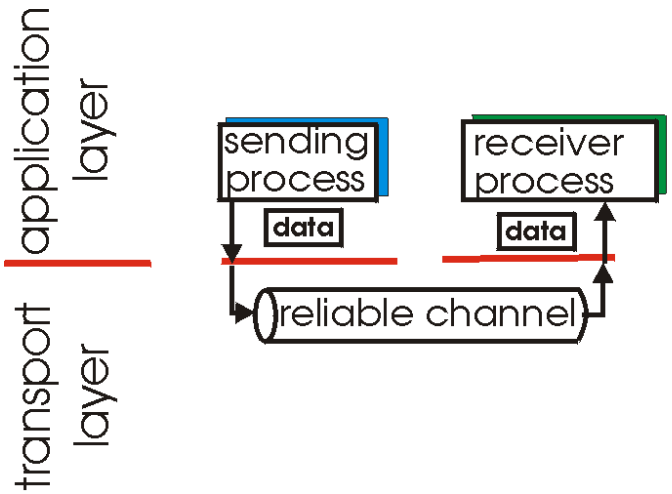
- entender os princípios por trás dos serviços da camada de transporte:
  - multiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento

Tópicos abordados:

- serviços e protocolos
- transporte sem conexão: UDP
- princípios de transferência confiável de dados
- transporte orientado à conexão: TCP
- princípios de controle de congestionamento

Transferência confiável: abstração

- Serviço de transferência confiável
- Porém, sobre um *canal não-confiável!*



(a) provided service

1 – se o canal fosse confiável...

- Vamos estudar incrementalmente como prover um ***protocolo confiável de transferência de dados*** (rdt)

- Numa primeira versão, se o canal é confiável, então:
  - transmissor envia dados para o canal subjacente
  - receptor lê os dados do canal subjacente (sem erros)

2 – problema: canal com erros de bit

- O canal pode trocar valores dos bits em um pacote
  - lembrete-se que a *checksum* **pode** detectar erros de bits
  - o transmissor poderia retransmitir um pacote "perdido" por erros de bits

- A questão é: **como detectar** que erros aconteceram:
  - *reconhecimento positivo (ACK):* receptor avisa explicitamente ao transmissor que o pacote foi **recebido corretamente**
  - *reconhecimento negativo (NACK):* receptor avisa explicitamente ao transmissor que o pacote **chegou com erros**

2 – problema: corrupção de ACKs

Mas o que acontece se o próprio ACK ou NACK for **corrompido**?

- transmissor não saberá diferenciar o que aconteceu no receptor!
- o transmissor não poderá simplesmente retransmitir o pacote em questão, pois geraria uma possível **duplicata** dele no receptor:

⇒ o receptor receberia o pacote (retransmitido) mas não saberia diferenciar se é um novo pacote ou uma duplicata do pacote anterior…

para e espera

transmissor envia um pacote e então espera pela resposta do receptor (ACK ou NACK)

2 – melhoria: um protocolo sem NACK

- MELHORIA: para simplificar, ao invés de enviar NACK
  - receptor re-envia ACK para o último pacote *recebido sem erro*
  - receptor inclui explicitamente o **número de sequência** do pacote sendo reconhecido

- RESULTADO:
  - ACKs duplicados no transmissor resultam na mesma ação do NACK
  - retransmissão do pacote posterior ao número de sequência informado no ACK

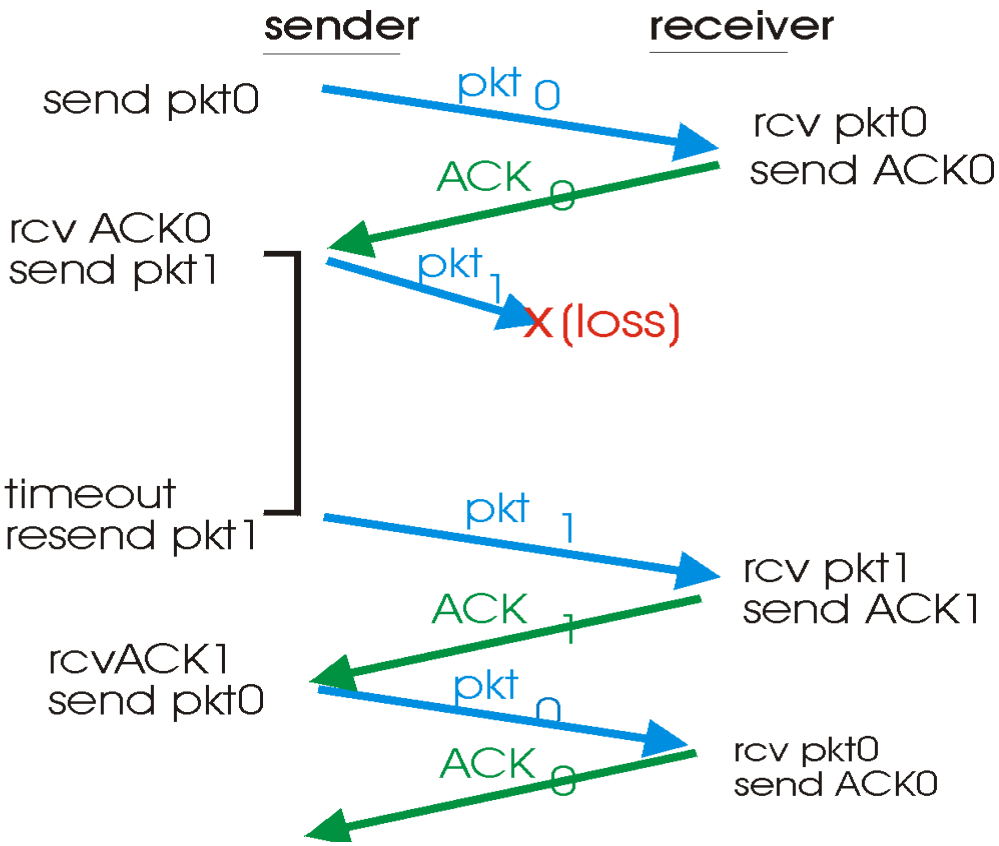
3 – problema: canais c/ erros e perdas

Mas como tratar perdas?

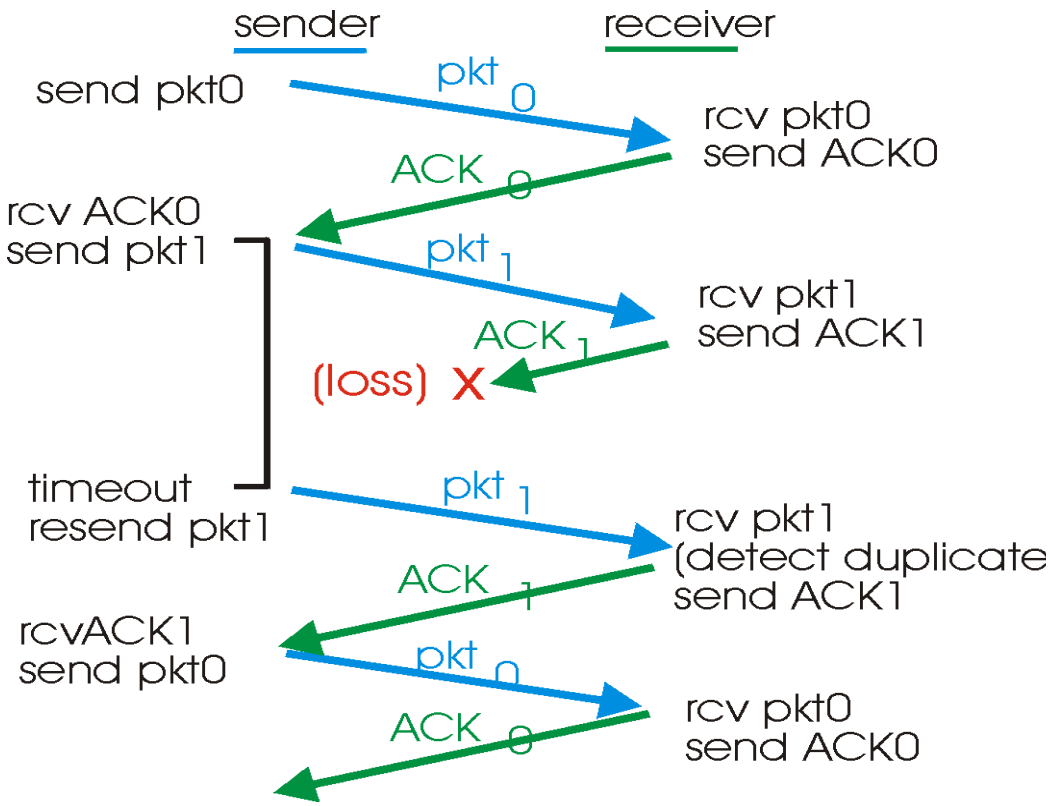
- transmissor *espera um tempo* "razoável" pelo ACK
  - retransmite se nenhum ACK for recebido nesse tempo

- mas se o pacote (ou ACK) estiver apenas atrasado (não perdido):
  - retransmissão será duplicata, mas os números de sequência já lidam com isso e a duplicata será descartada
  - por isso o receptor deve especificar o número de sequência do último pacote sendo reconhecido

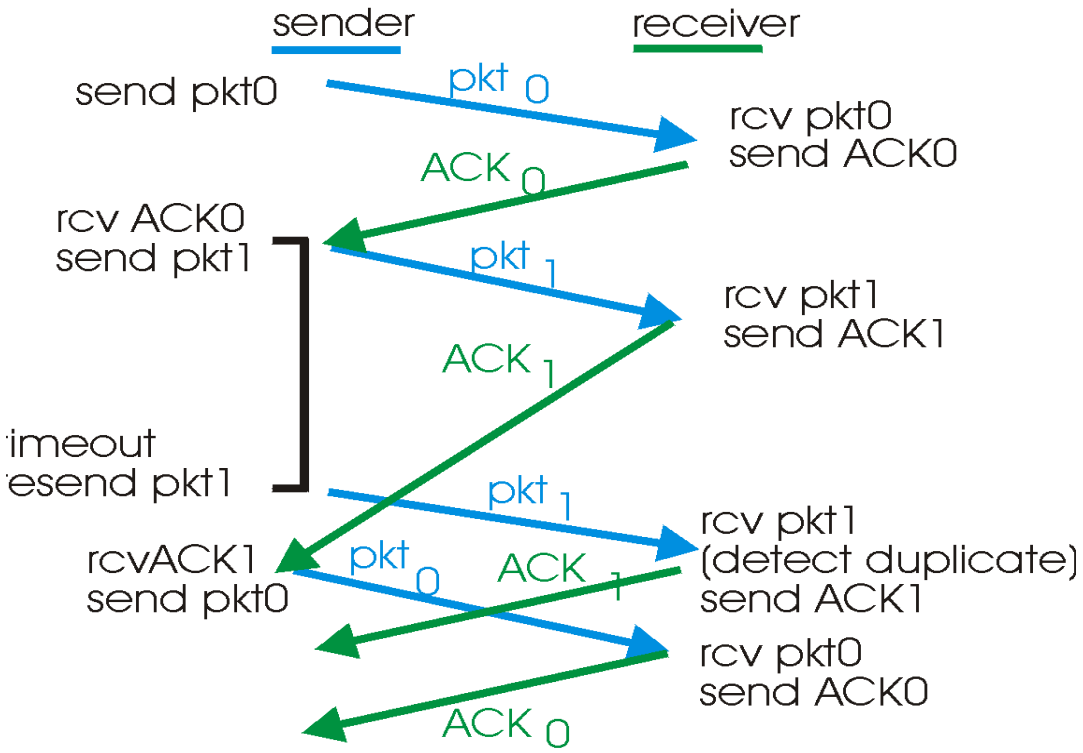
3 – exemplo: pacote perdido



3 – exemplo: ACK perdido



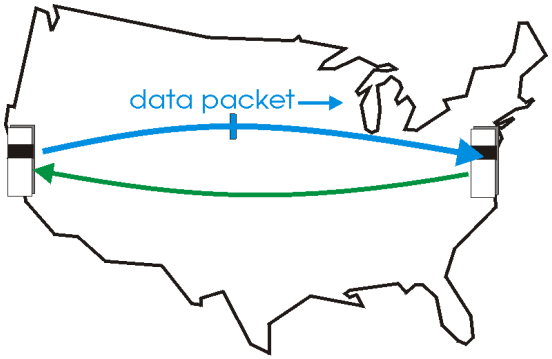
3 – exemplo: *timeout* precoce



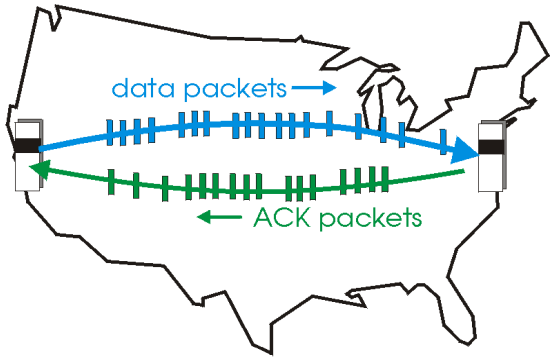
Protocolos com paralelismo (*pipelining*)

Transmissor envia vários pacotes “ao mesmo tempo” (em seguida, sem esperar que cada um seja reconhecido)

- uma “janela” define e limita o grau de paralelismo
- faixa de números de sequência deve ser aumentada
- armazenamento no transmissor e/ou no receptor



(a) operação do protocolo para-e-espera

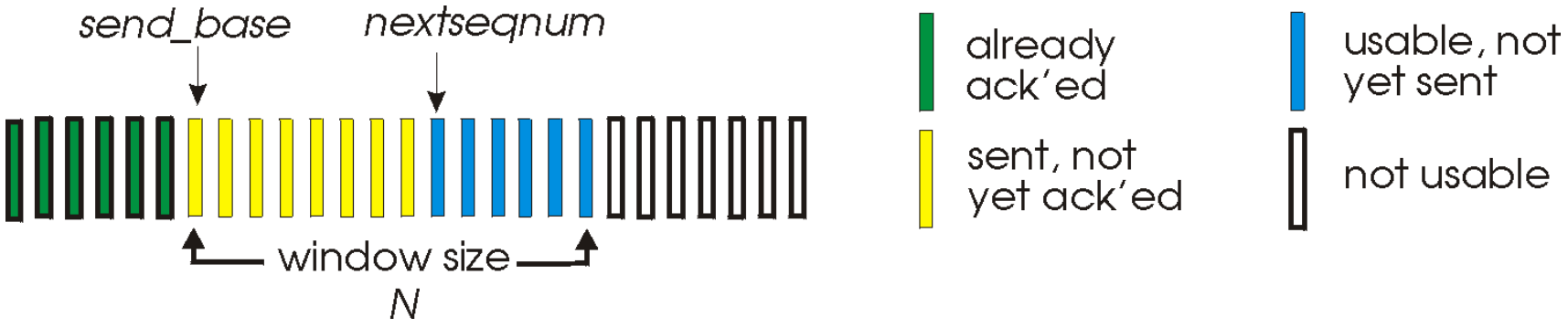


(b) operação do protocolo com paralelismo

Protocolos com paralelismo (*pipelining*)

Transmissor:

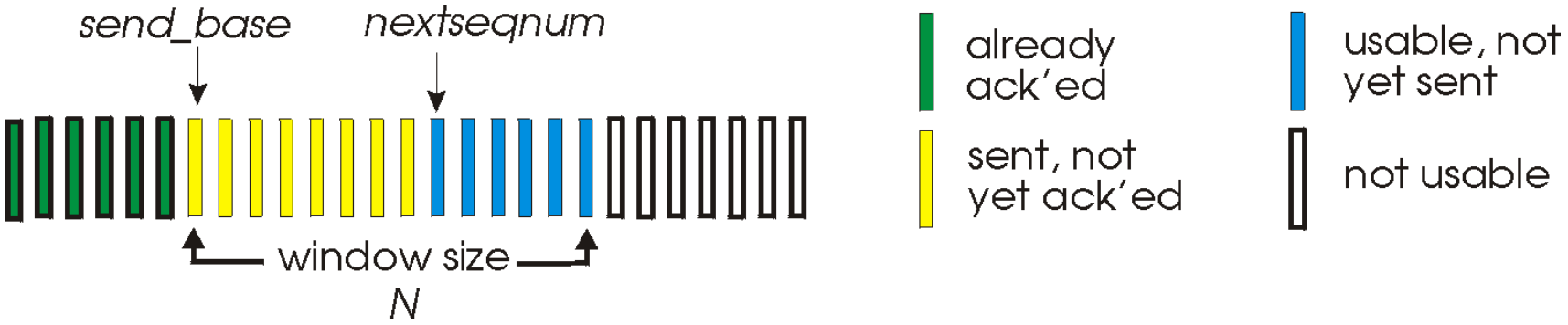
- Número de sequência com *k* bits no cabeçalho do pacote
- "Janela" de **até *N*** pacotes consecutivos não confirmados



Solução 1: *Go-Back-N* ("Volta-N")

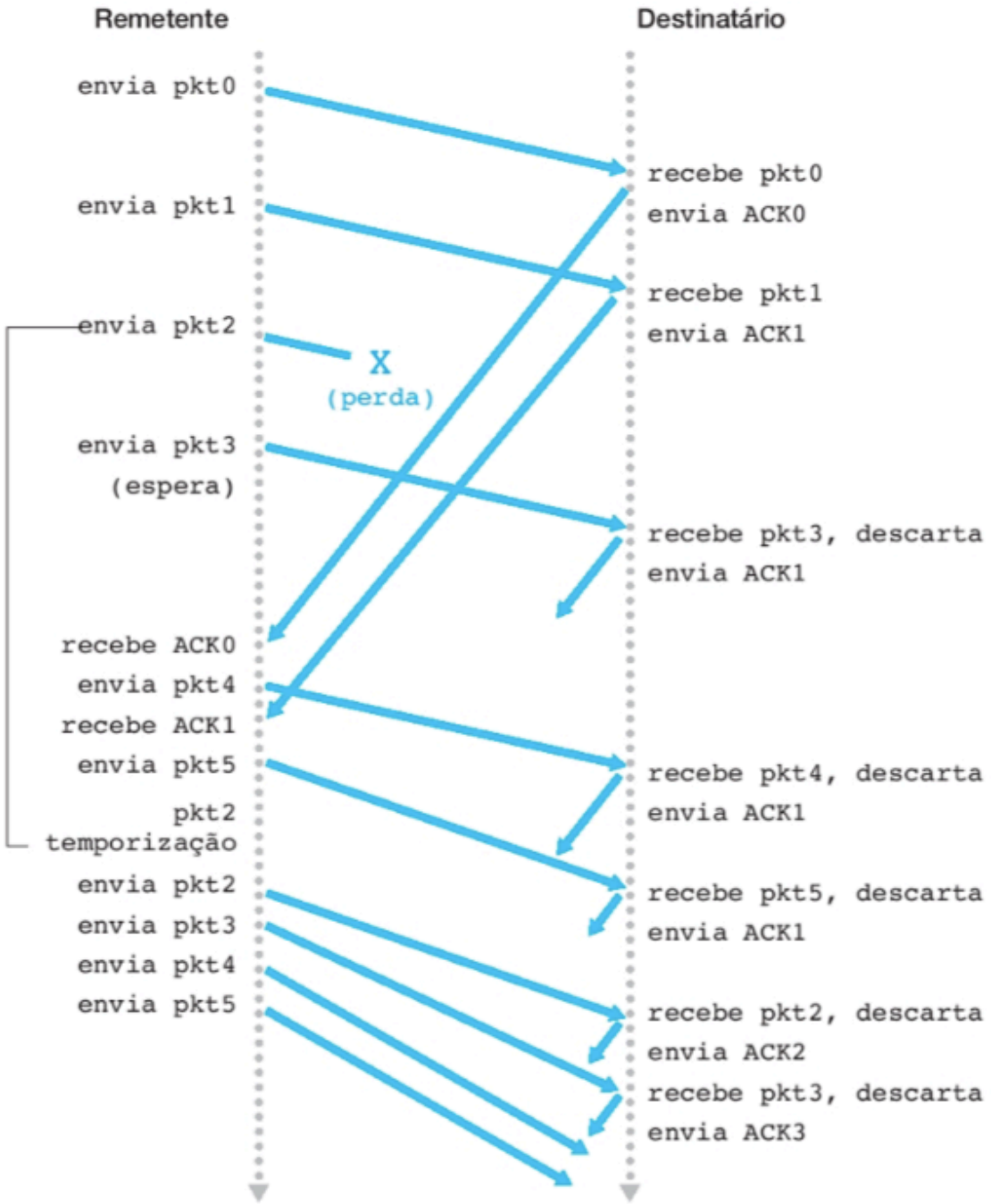
Transmissor:

- Número de sequência com *k* bits no cabeçalho do pacote
- "Janela" de **até *N*** pacotes consecutivos não confirmados



- Confirmação **cumulativa**: o *ACK*(x) confirma recebimento dos pacotes com número sequência "**até X**" (pacotes anteriores a ele)
- Há temporização para cada pacote enviado e não confirmado
  - *timeout*(x) retransmitiria o pacote nº X e todos os seguintes!

FIGURA 3.22 GO-BACK-N EM OPERAÇÃO

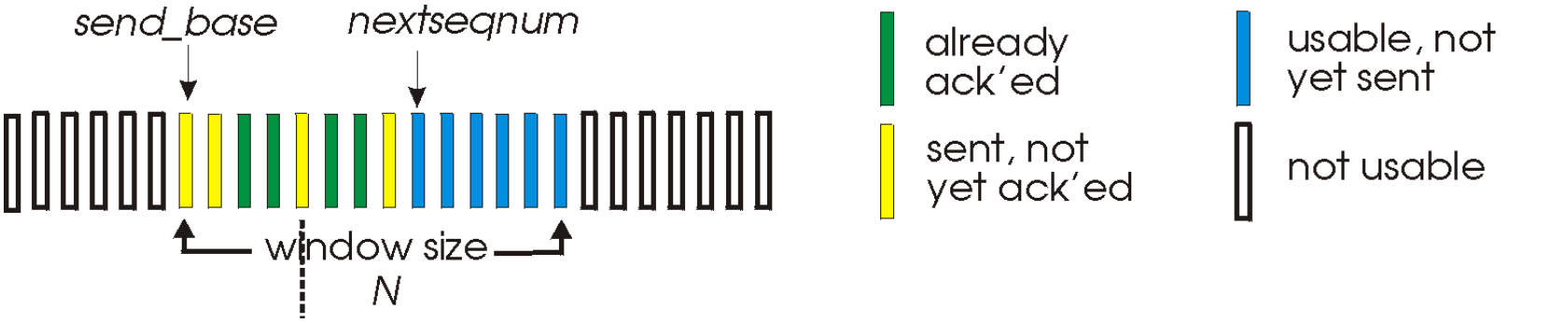


Link para animação

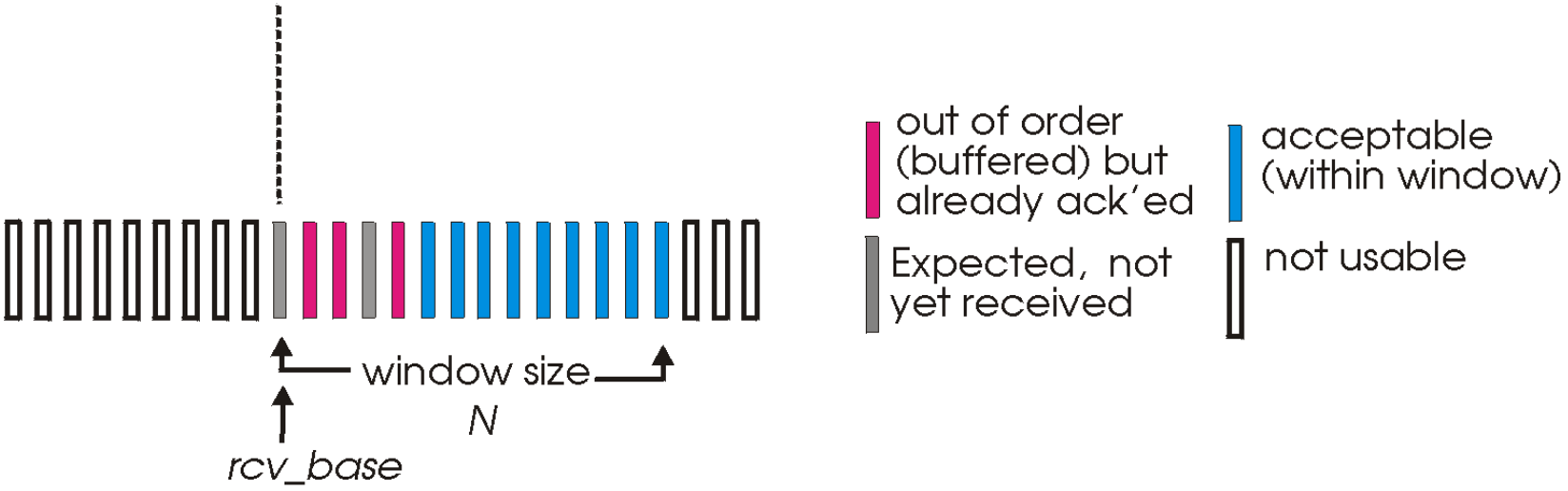
Solução 2: Retransmissão seletiva (SR)

- receptor confirma *individualmente* cada pacote
  - **armazena**, quando necessário, para entrega em ordem
- transmissor só reenvia os pacotes para os quais um ACK não foi recebido
  - transmissor temporiza cada pacote não reconhecido

SR: janelas no transmissor e no receptor

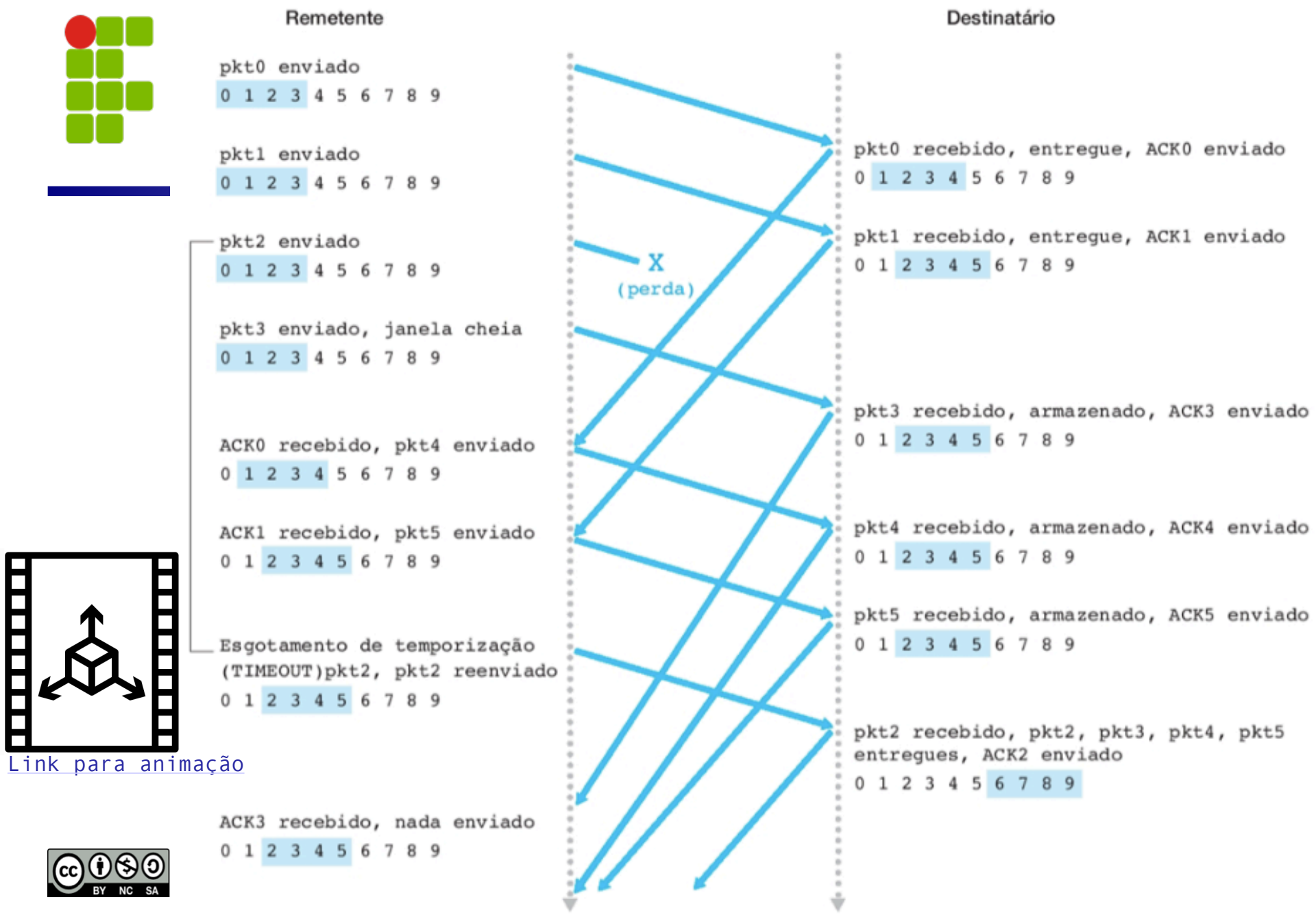


(a) visão dos números de seqüência pelo transmissor



(b) visão dos números de sequência pelo receptor

FIGURA 3.26 OPERAÇÃO SR



Link para animação

Janelas deslizantes e nº.s de sequência

Como no caso do *para-e-espera*, números de sequência são finitos e precisam ser reutilizados

Ex.: para dois bits (2^2 = 4 nº.s), a seq. seria 0 1 2 3 0 1 2 3 ...

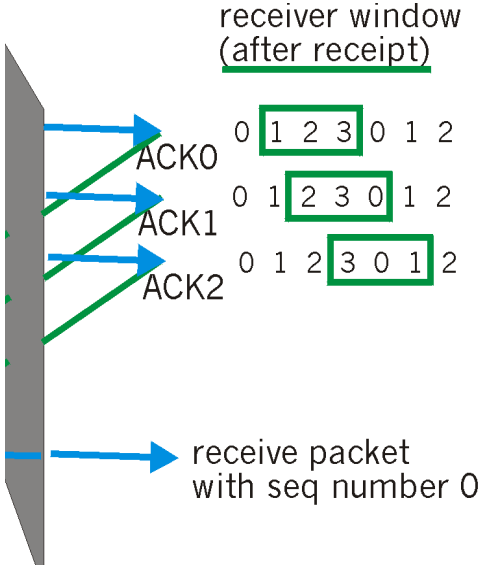
Mas isso precisa ser feito com cuidado!

Ex.: seja uma janela de tamanho 3, o receptor recebe e confirma os seguintes pacotes: 0 1 2 **0**

→ O que aconteceu? ACK 0 ou ACK 0 ?

→ Quantos pacotes foram recebidos?

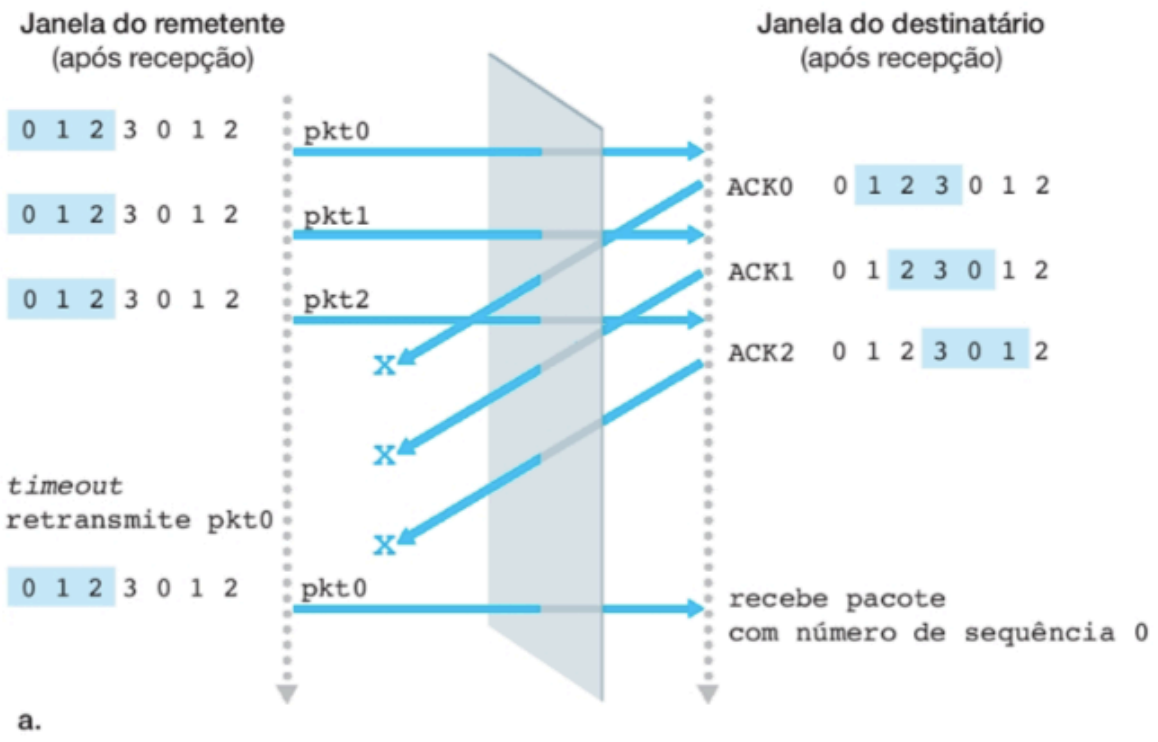
3? 4?



Janelas deslizantes e nº.s de sequência

Todos os ACKs foram perdidos MAS os pacotes nº 0,1 e 2 chegaram:

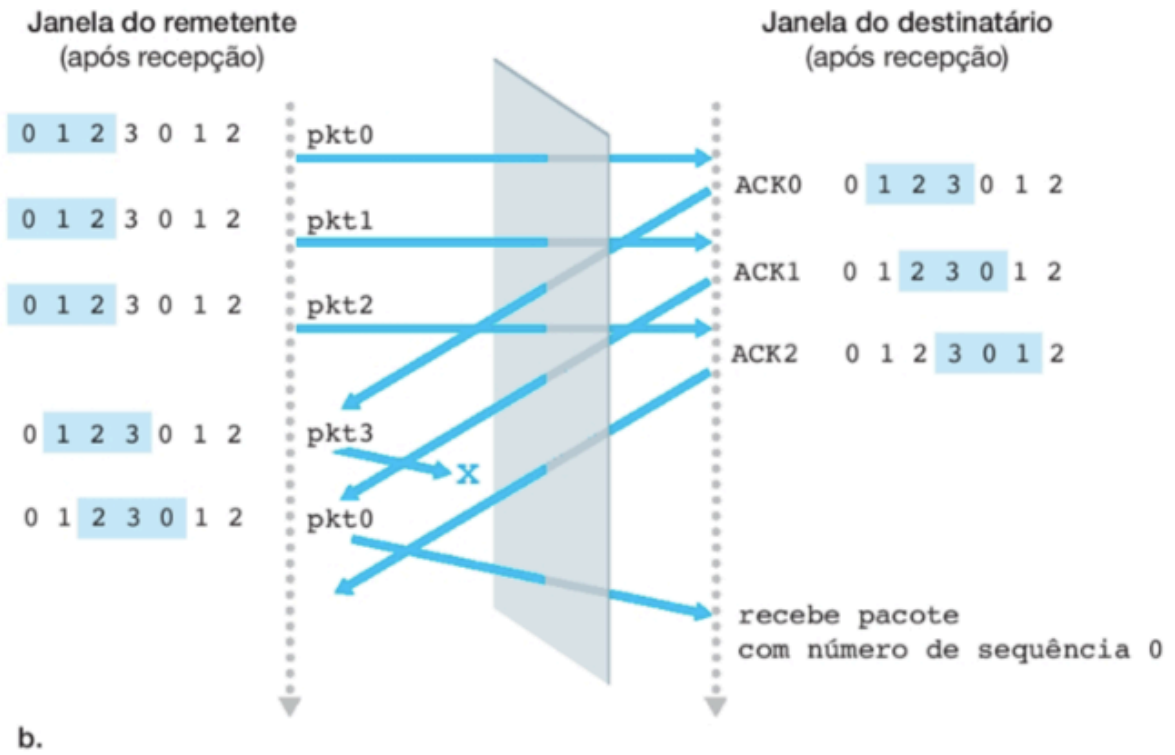
FIGURA 3.27 **DILEMA DO REMETENTE SR COM JANELAS MUITO GRANDES: UM NOVO PACOTE OU UMA RETRANSMISSÃO?**



Janelas deslizantes e nº.s de sequência

Todos os ACKs chegaram MAS o pacote nº 3 foi perdido:

FIGURA 3.27 **DILEMA DO REMETENTE SR COM JANELAS MUITO GRANDES: UM NOVO PACOTE OU UMA RETRANSMISSÃO?**



Saiba mais!

- Pesquise sobre as seguintes implementações TCP:
  - Tahoe
  - Reno
  - Vegas
  - NewReno
  - SACK

- No que estas implementações do TCP diferem?
- Em que cenários uma implementação se sai melhor e pior que as outras?

Camada de transporte

Nosso objetivo:

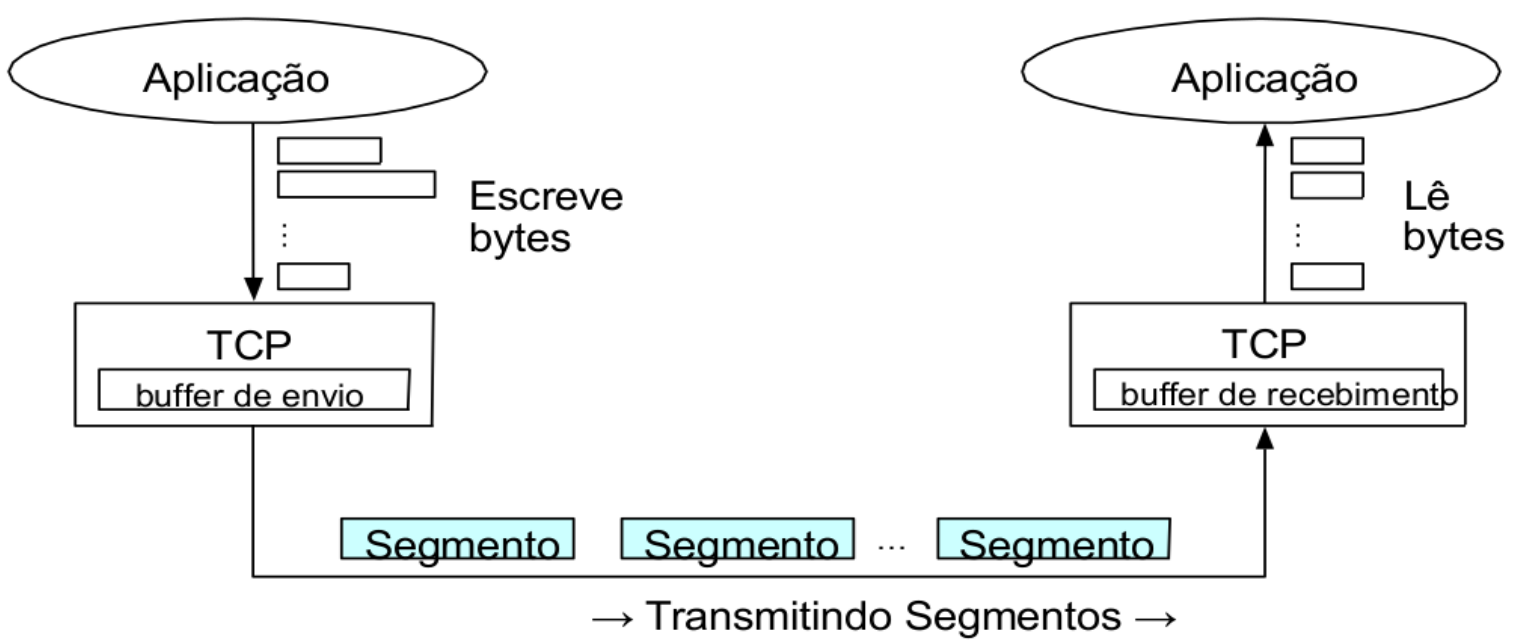
- entender os princípios por trás dos serviços da camada de transporte:
  - multiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento

Tópicos abordados:

- serviços e protocolos
- transporte sem conexão: UDP
- princípios de transferência confiável de dados
- transporte orientado à conexão: TCP
- princípios de controle de congestionamento

TCP (*Transmission Control Protocol*)

- Sequência de bytes não estruturada (*byte stream*)
  - aplicação transmissora escreve bytes
  - sequência é dividida em **segmentos** para o envio
  - aplicação receptora lê bytes



TCP: identificação de conexão

Portos, conexões e endpoints:

TCP identifica os dois pontos que se comunicam como terminações da conexão (*endpoints*)

Um *endpoint* é um par de inteiros da forma:

Host, Port

A conexão é identificada por um par de *endpoints*

(128.9.0.32, 1184) (128.10.2.3, 25)

TCP: identificação de conexão

- Portos, conexões e *endpoints*:
  - *Endpoints* permitem que um determinad porto possa ser compartilhado por múltiplas conexões

(128.9.0.32, 1184) (128.10.2.3, 25)

(128.2.1.27, 1184) (128.10.2.3, 25)

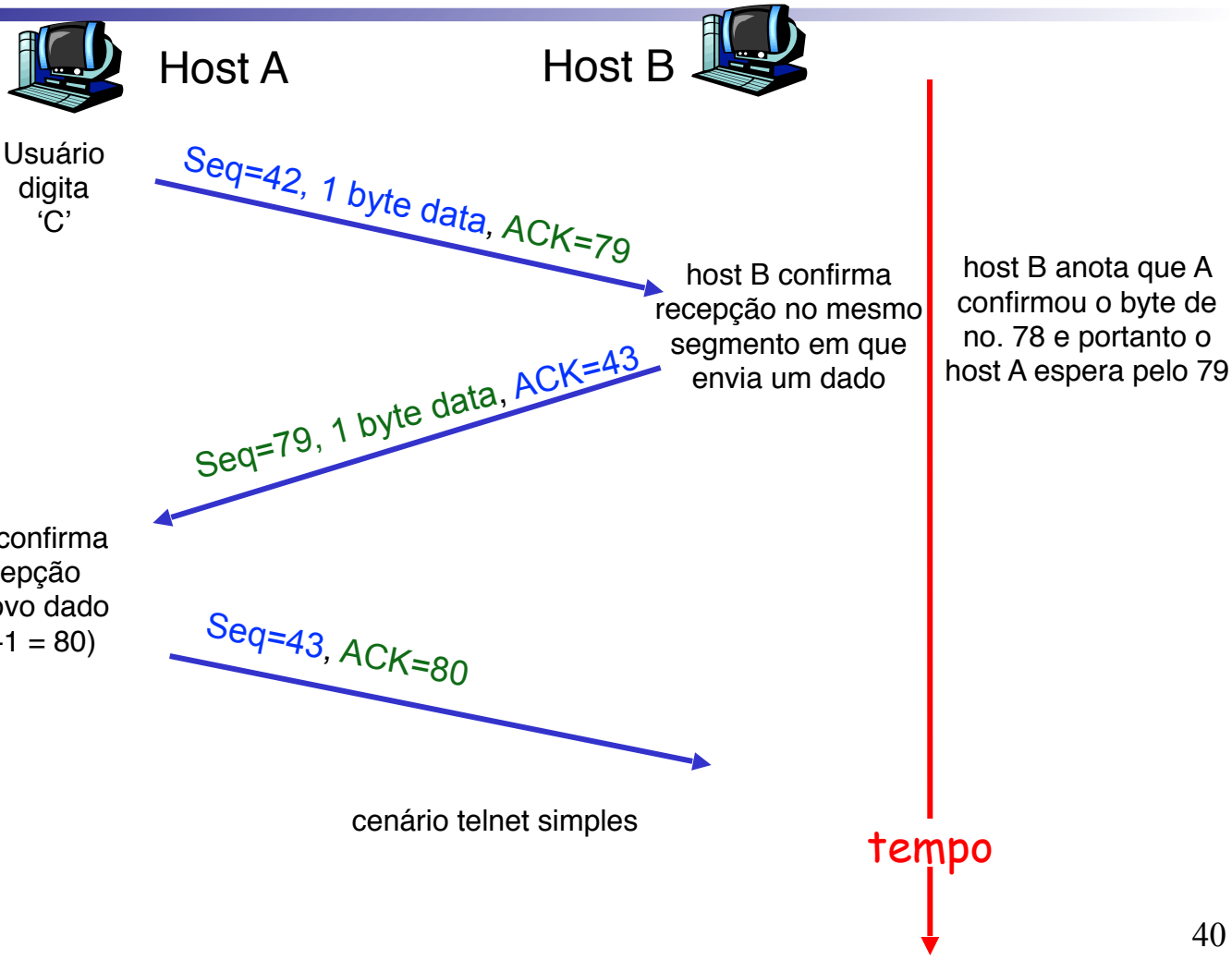
(128.2.1.27, 2167) (128.10.2.3, 25)

  - Neste exemplo, o endpoint (128.10.2.3, 25) seria um cliente ou um servidor?

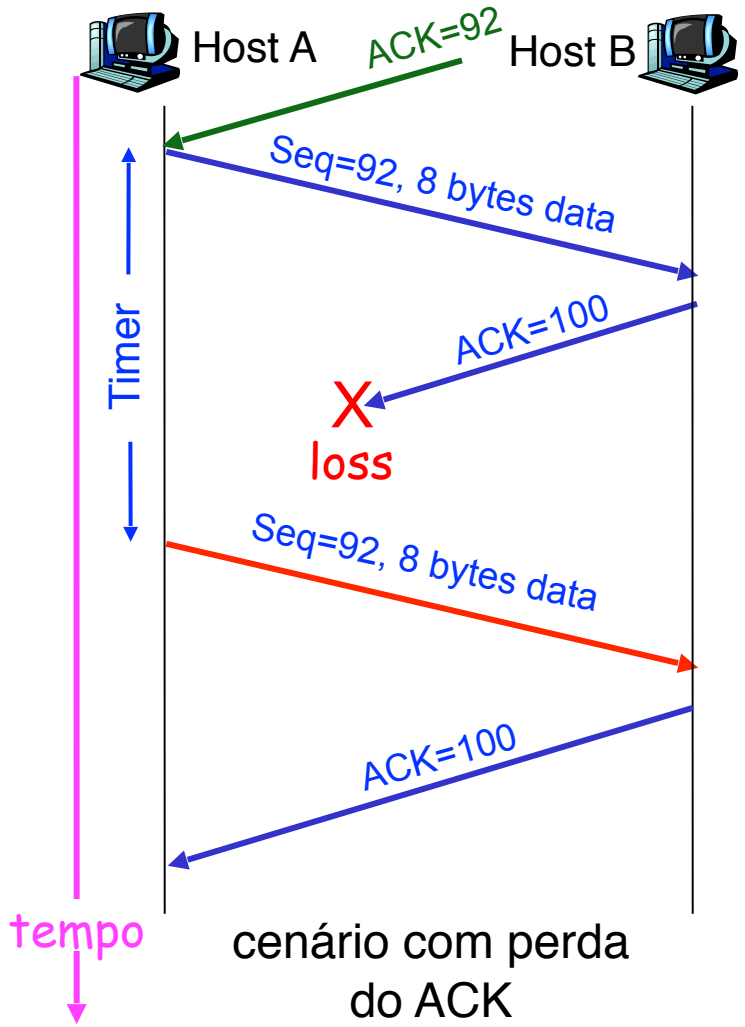
TCP: correção de erros na transmissão

- Usa confirmação positiva (ACK, *acknowledgement*)
  - OBS.: utiliza *piggybacking* (carona) no reconhecimento, confirmações são enviadas com os dados no sentido oposto
- Dados podem ser recebidos fora de ordem
- Qualquer segmento recebido gera uma **confirmação do último byte recebido em ordem**

Números de Sequência (Seq) e ACKs



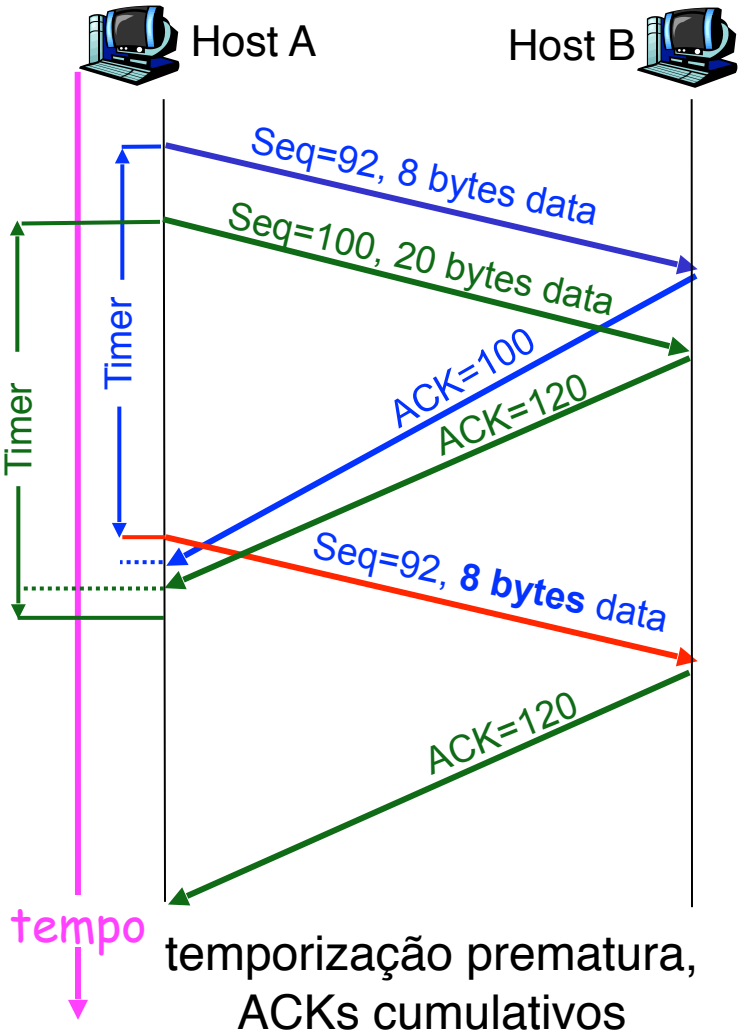
TCP: cenário com perda de ACK



cenário com perda do ACK

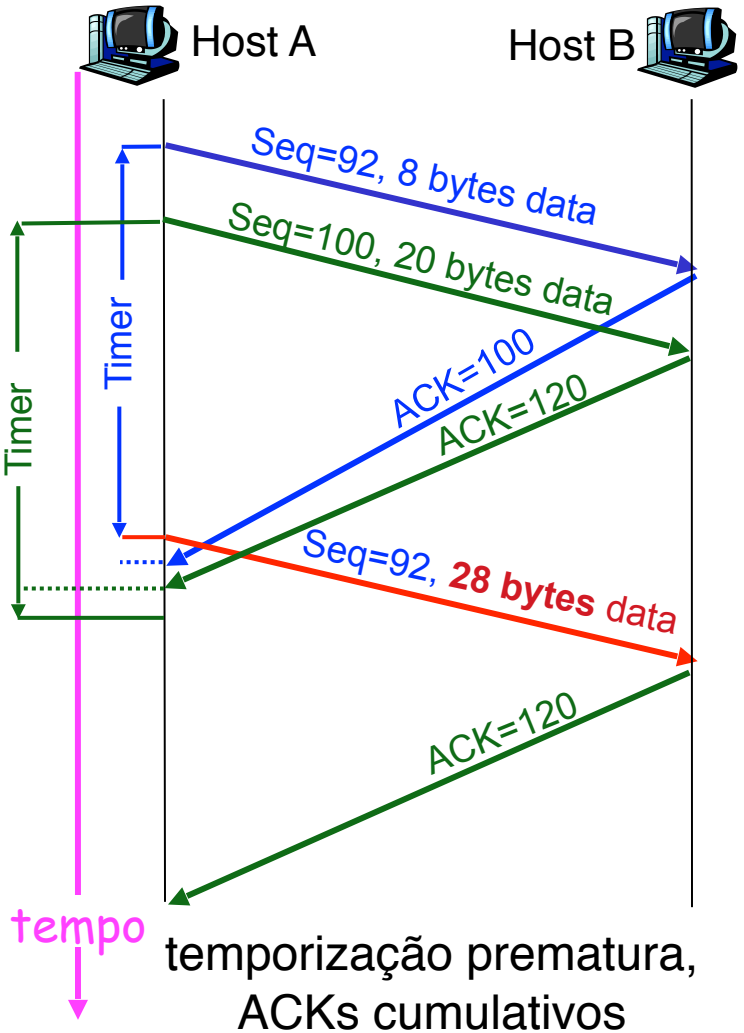
DICA:
ACK = Seq. + Tam.

TCP: cenário com retransmissão prematura e ACKs cumulativos



temporização prematura, ACKs cumulativos

GBN: cenário com retransmissão prematura e ACKs cumulativos



temporização prematura, ACKs cumulativos

TCP: retransmissão adaptativa
(ou temporização adaptável)

Problema a ser resolvido:

Como configurar temporizadores para comunicações em LANs e WANs?

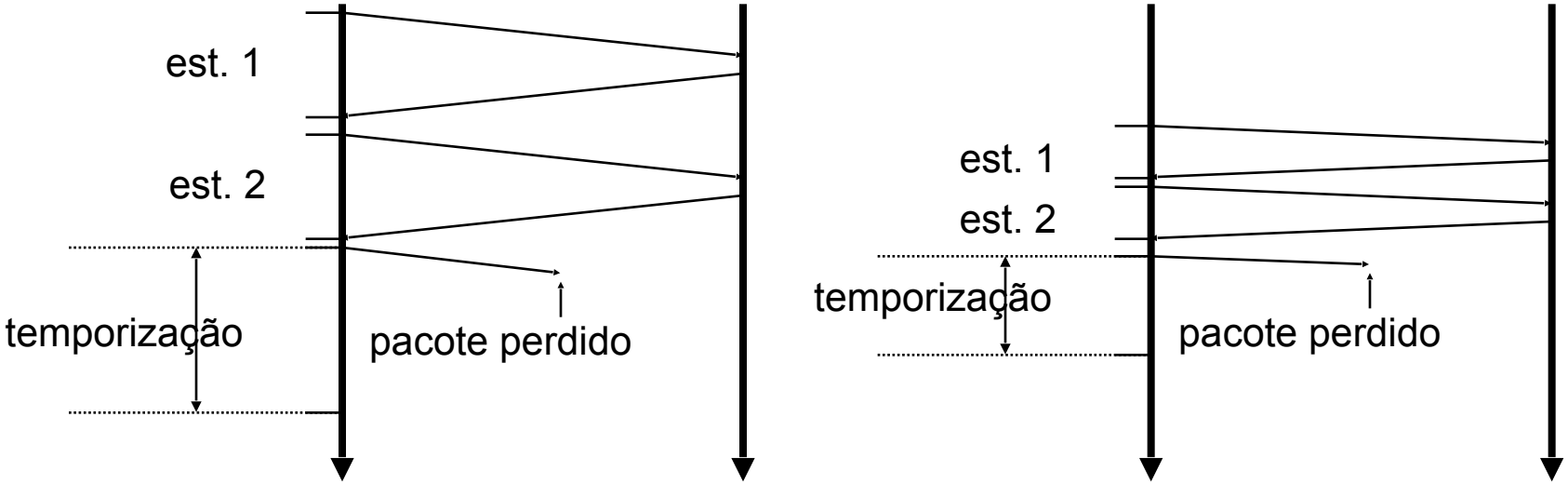
- LANs: deve esperar pouco (ex.: atraso < 10 ms)
- WANs: deve esperar mais (ex.: atraso > 100 ms)

TCP: retransmissão adaptativa
(ou temporização adaptável)

TCP adota temporizações variáveis

Monitora o atraso de alguns pacotes em cada conexão (`SampleRTT`) e modifica o temporizador para acomodar mudanças

Mudança é feita em função de análise estatística das mensagens transmitidas **com sucesso**

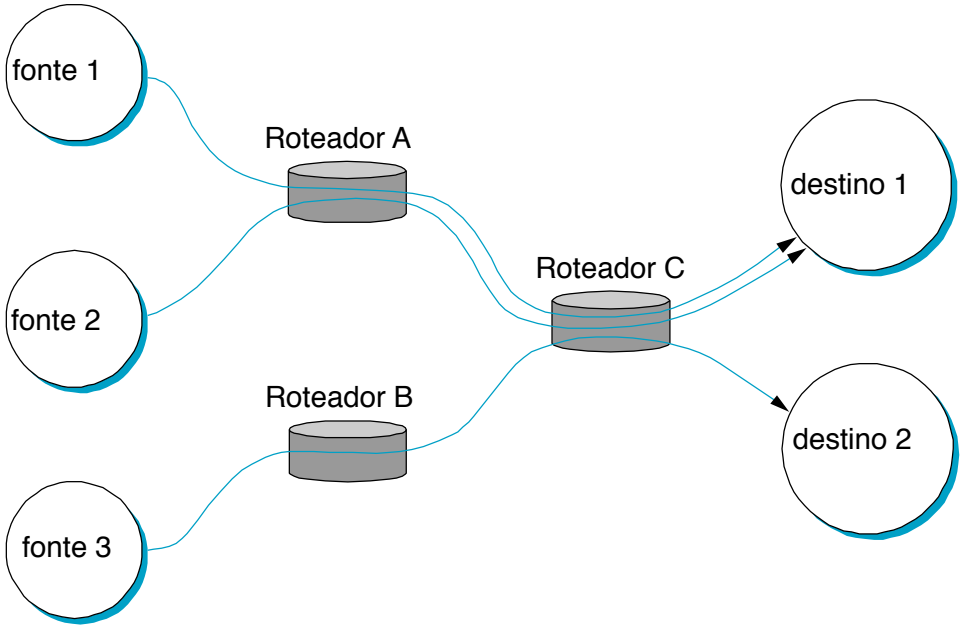


Controle de fluxo

- Evita que o transmissor sobrecarregue o receptor
- TCP é um protocolo de transporte, não aplicação
  - se um dado confirmado ainda não foi usado pela aplicação, este dado precisa ficar no buffer até a aplicação lê-lo
  - um ACK significa somente que foi recebido até aquele byte
  - se os ACKs chegarem, pode-se enviar mais dados infinitamente?
- Solução: um “anúncio de janela” indica se o byte foi lido pelo destino
  - ex.: “posso receber mais 4096 bytes” (tamanho da janela)
  - transmissor pode enviar dados somente se o anúncio “abre nova janela”

Controle de congestionamento

- Evita que os transmissores sobrecarreguem a rede
- Desafios:
  - Determinar a capacidade para cada fonte
  - Adaptar fluxos a variações da capacidade



Camada de transporte

Nosso objetivo:

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

Tópicos abordados:

- serviços e protocolos
- transporte sem conexão: UDP
- princípios de transferência confiável de dados
- transporte orientado à conexão: TCP
- princípios de controle de congestionamento

Há duas abordagens para este problema

Controle de congestionamento **fim-a-fim**:

- usa realimentação **implícita**
- congestionamento é **inferido** a partir das perdas e dos atrasos observados nos sistemas finais
- abordagem usada pelo TCP

Controle de congestionamento **assistido pela rede**:

- usa realimentação **explícita** da rede
- **roteadores** enviam informações para os sistemas finais
  - no cabeçalho, há um bit único indicando o congestionamento
  - taxa explícita do transmissor poderia ser enviada

Controle de congestionamento de TCP

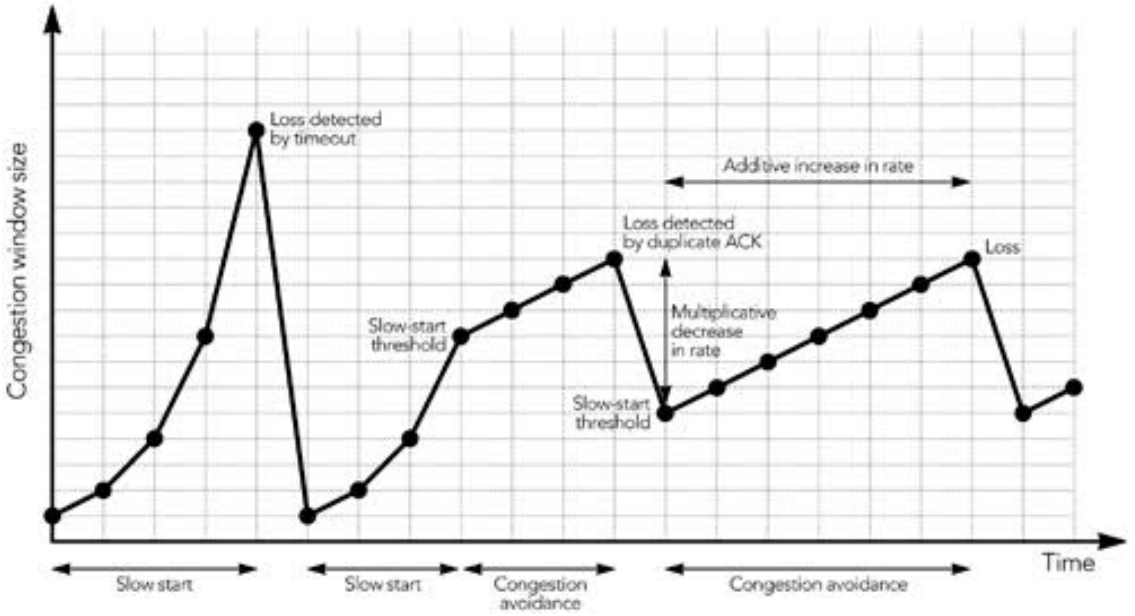
- Considera a rede como "*melhor-esforço*"
  - Transmissão acima da capacidade é descartada
- Usa realimentação implícita
  - Perda de pacotes é considerada sinal de contenção (redução de taxa)
- ACKs controlam a transmissão ("auto-temporização")
  - Anúncio de nova janela vem junto com ACKs e permite novos envios
- Janela de transmissão: anúncio & janela de congestionamento

Controle de congestionamento de TCP

- TCP provê um controle de *feedback* para evitar congestionamento
- 1ª fase: **início lento** (*slow start*)
- 2ª fase: **algoritmo "AIMD"** (incremento aditivo / decremento multiplicativo)
- Múltiplos fluxos TCP usando o controle de congestionamento AIMD
  - eventualmente irão convergir para usar quantidades similares de um enlace congestionado

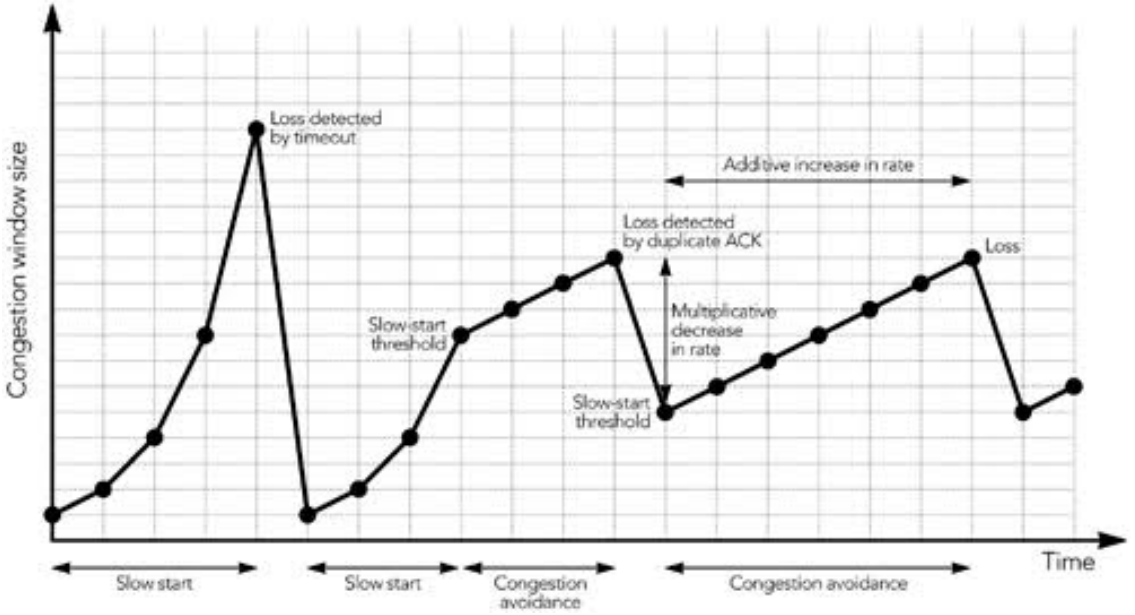
Slow Start (RFC 5681)

- Nesta 1ª fase, chamada início "lento" (*slow start*), a janela de congestionamento (***c**wnd*) inicial tem um valor de 1, 2 ou 10.
- Este valor vai ser incrementado a cada ACK recebido, efetivamente dobrando o tamanho da janela a cada RTT (*round-trip-time*)
  - OBS.: não é exponencial pois o receptor pode atrasar seus ACKS



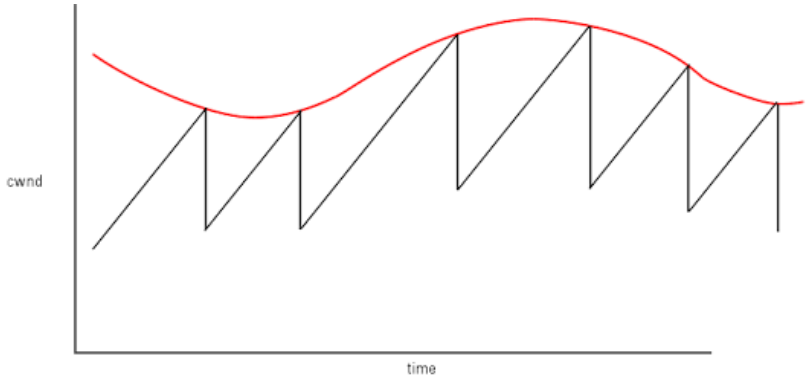
Slow Start (RFC 5681)

- A taxa de transmissão será incrementada com o algoritmo *slow start* até que uma das situações abaixo seja atingida:
  - uma perda seja detectada
  - até o anúncio de janela do receptor (***rwnd***) ser o fator limitador
  - até o limiar do *slow start* seja alcançado (***ssthresh***)



Algoritmo AIMD

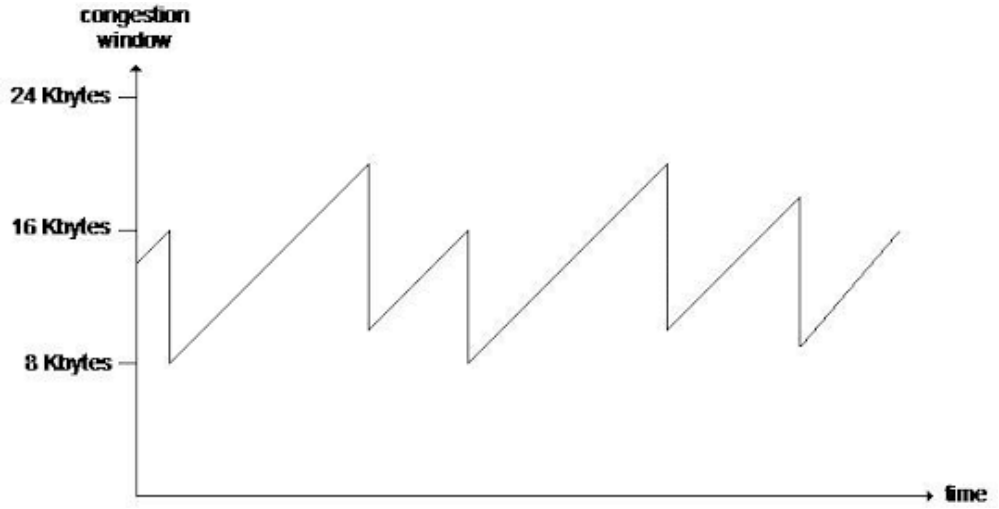
- AIMD (incremento aditivo / decremento multiplicativo):
  - crescimento linear da janela de congestionamento
  - redução exponencial quando ocorre congestionamento
- Abordagem: incrementar a taxa de transmissão (tamanho da janela) **até que uma perda ocorra**
  - realiza uma "sondagem" da largura de banda utilizável, ilustrada pelo comportamento serrilhado (*saw-tooth*)



TCP Sawtooth, red curve represents the network capacity

Algoritmo AIMD

- Aumento aditivo:
  - ex.: aumenta a janela de congestionamento em uma quantidade fixa (tipicamente 1 MSS) a cada RTT (*round-trip-time*)
- Decremento multiplicativo:
  - ex.: após uma perda reduz a taxa de transmissão num fator multiplicativo, tipicamente 1/2 (reduz pela metade)



TCP: conexão com três fases

- Estabelecimento (SYN)
  - Aplicações devem reconhecer a nova conexão como única e inconfundível
  - Pacotes de conexões anteriores não são podem ser tomados como pacotes válidos
- Transferência de dados
  - Comunicação *full-duplex* (pode enviar e receber ao mesmo tempo)
  - Envia os dados como uma sequência de bytes
- Término da conexão (FIN)
  - Garante a entrega de todos os dados antes de fechar uma conexão a pedido da aplicação

TCP (*Transmission Control Protocol*)

- Identificação de **portos** (multiplexação)
- Serviço orientado a **conexão** (circuito virtual)
- **Confiável** (detecção e correção de erros)
- Sequência de bytes não estruturada (*byte* ***stream***)
- Controle automático de ***buffers***
- Controle de fluxo (**janela** deslizante)
  - evita que o transmissor sobrecaregue o receptor
- Controle de congestionamento (**SlowStart**, **AIMD**)
  - evita que o transmissor sobrecarregue a rede

Resumo

- princípios por trás dos serviços da camada de transporte:
  - multiplexação/demultiplexação
  - transferência confiável
  - controle de fluxo
  - controle de congestionamento
- protocolos:
  - UDP
  - TCP

A seguir:

- saímos da “borda” da rede (camadas de aplicação e de transporte)
- vamos para o “núcleo” da rede (cama de rede e enlace)