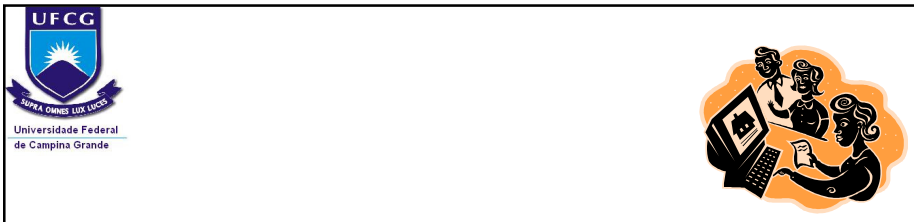


Redes de Computadores

Parte III: Camada de Transporte

Março, 2012

Professor: Reinaldo Gomes
reinaldo@dsc.ufcg.edu.br

Camada de transporte

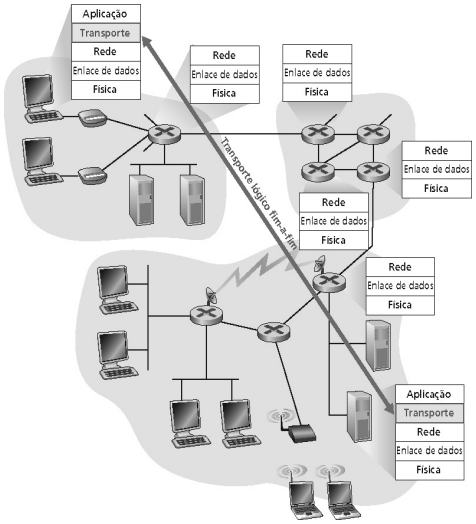
- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado à conexão: UDP
- 3.4 Princípios de transferência confiável de dados
- 3.5 Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

Protocolos e serviços de transporte

- Fornecem comunicação lógica entre processos de aplicação em diferentes hospedeiros

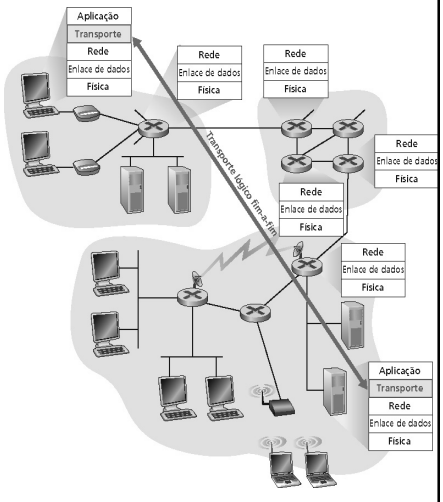
- Os protocolos de transporte são executados nos sistemas finais
  - Lado emissor: quebra as mensagens da aplicação em segmentos e envia para a camada de rede
  - Lado receptor: remonta os segmentos em mensagens e passa para a camada de aplicação

- Há mais de um protocolo de transporte disponível para as aplicações
  - Internet: TCP e UDP



Protocolos da camada de transporte da Internet

- TCP:
  - Confiável garante ordem de entrega
  - Controle de congestionamento
  - Controle de fluxo
  - Orientado à conexão
- UDP:
  - Não confiável, não garante ordem na entrega
  - Extensão do "melhor esforço" do IP
- Serviços não disponíveis:
  - Garantia a atrasos
  - Garantia de banda
  - Por quê?



Principais funções

- Controle de erros
- Controle de fluxo
- Multiplexação de aplicações
- Nem todas as camadas de transporte implementam o mesmo conjunto de serviços
- TCP - Controle de congestionamento
- UDP - apenas multiplexação

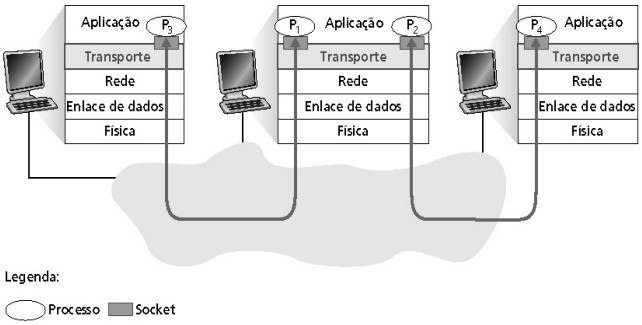
Camada de transporte

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado à conexão: UDP
- 3.4 Princípios de transferência confiável de dados
- 3.5 Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

Demultiplexação/Multiplexação

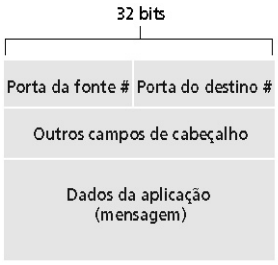
Demultiplexação no hospedeiro receptor:
entrega os segmentos recebidos ao socket correto

Multiplexação no hospedeiro emissor:
coleta dados de múltiplos sockets, envelopa os dados com cabeçalho (usado depois para demultiplexação)



Demultiplexação/Multiplexação

- Computador recebe datagramas IP
  - Cada datagrama possui endereço IP de origem e IP de destino
  - Cada datagrama carrega 1 segmento da camada de transporte
  - Cada segmento possui números de porta de origem e destino (lembre-se: números de porta bem conhecidos para aplicações específicas)

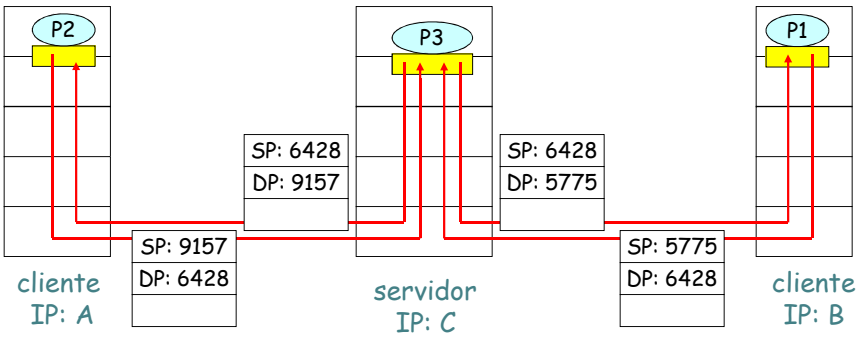


- O hospedeiro usa endereços IP e números de porta para direcionar o segmento ao socket apropriado

Demultiplexação/Multiplexação

- Socket UDP identificado por dois valores:
  (endereço IP de destino, número da porta de destino)
- Quando o hospedeiro recebe o segmento UDP:
  - Verifica o número da porta de destino no segmento
  - Direciona o segmento UDP para o socket com este número de porta
- Datagramas com IP de origem diferentes e/ou portas de origem diferentes são direcionados para o mesmo socket

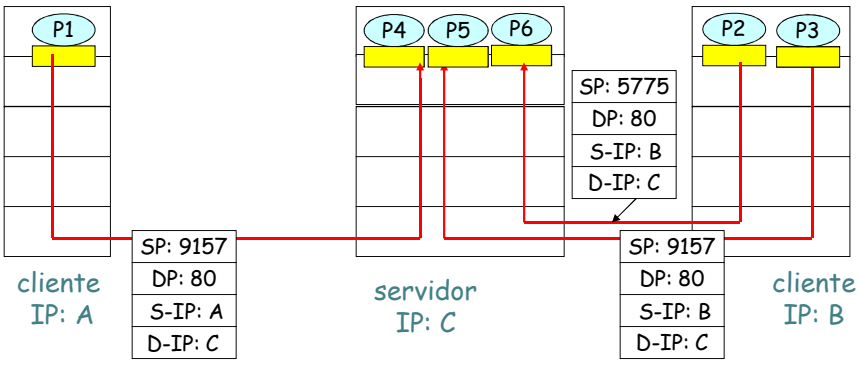
Demultiplexação/Multiplexação



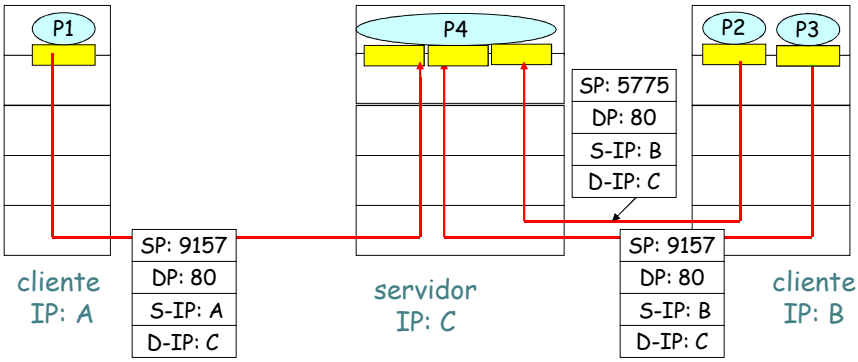
Demultiplexação/Multiplexação

- Socket TCP identificado por 4 valores:
  - Endereço IP de origem
  - End. porta de origem
  - Endereço IP de destino
  - End. porta de destino
- Hospedeiro receptor usa os quatro valores para direcionar o segmento ao socket apropriado
- Hospedeiro servidor pode suportar vários sockets TCP simultâneos:
  - Cada socket é identificado pelos seus próprios 4 valores
  - Servidores Web possuem sockets diferentes para cada cliente conectado
  - HTTP não persistente terá um socket diferente para cada requisição

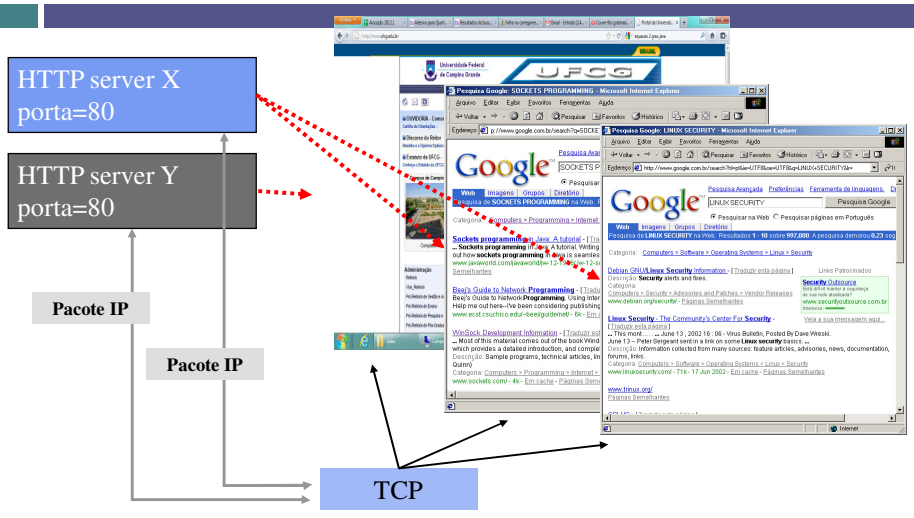
Demultiplexação/Multiplexação



Demultiplexação/Multiplexação



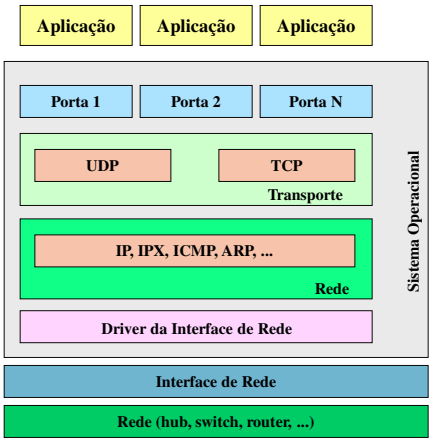
Identificação da aplicação no host



As aplicações não confundem os pacotes que chegam ?

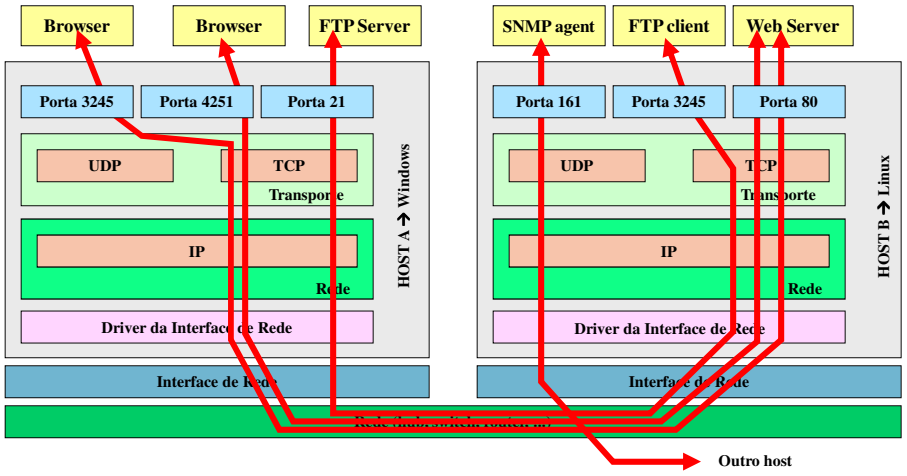
Pilha de Protocolos, na prática

15



Pilha de Protocolos, na prática

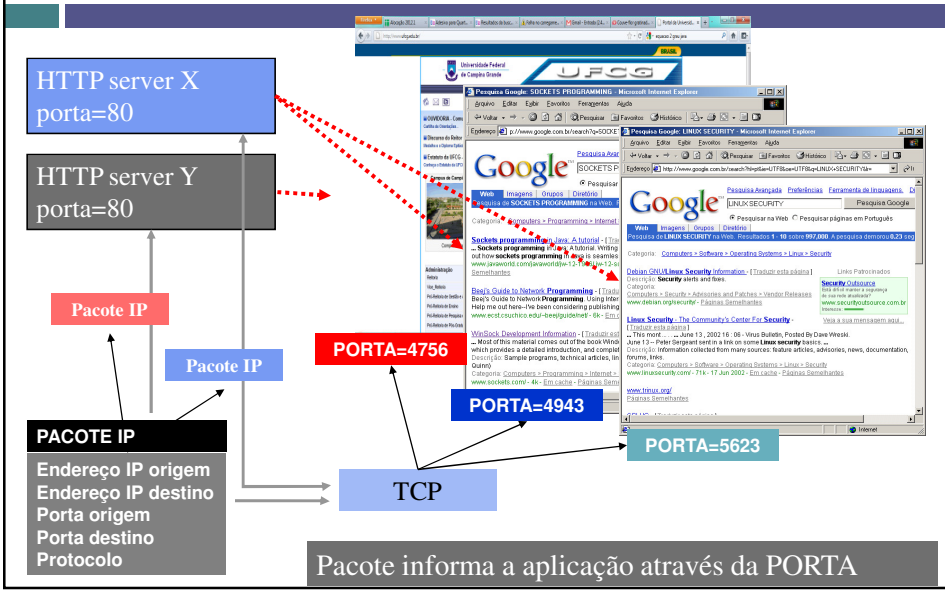
16



O TCP/IP sabe para qual aplicação entregar o pacote olhando a TUPLA:

Endereço IP origem, Endereço IP destino, Porta origem, Porta destino, Protocolo

Identificação da aplicação no host



Camada de transporte

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado à conexão: UDP
- 3.4 Princípios de transferência confiável de dados
- 3.5 Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

UDP: User Datagram Protocol

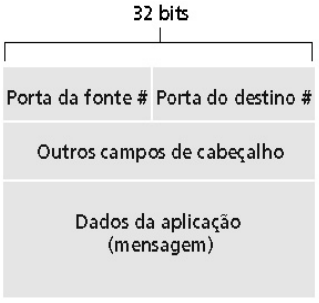
- Protocolo de transporte da Internet "sem gorduras", "sem frescuras"
- Serviço "best effort", segmentos UDP podem ser:
  - Perdidos
  - Entregues fora de ordem para a aplicação
- Sem conexão:
  - Não há apresentação entre o UDP transmissor e o receptor
  - Cada segmento UDP é tratado de forma independente dos outros

Por que existe um UDP?

- Não há estabelecimento de conexão (que possa redundar em atrasos)
- Simples: não há estado de conexão nem no transmissor, nem no receptor
- Cabeçalho de segmento reduzido
- Não há controle de congestionamento: UDP pode enviar segmentos tão rápido quanto desejado (e possível)

UDP: User Datagram Protocol

- Muito usado por aplicações de multimídia contínua (streaming)
  - Tolerantes à perda
  - Sensíveis à taxa
- Outros usos do UDP (por quê?):
  - DNS
  - SNMP
- Transferência confiável sobre UDP: acrescentar confiabilidade na camada de aplicação
  - Recuperação de erro específica de cada aplicação



UDP Checksum

Objetivo: detectar "erros" (ex.: bits trocados) no segmento transmitido

Transmissor:

- Trata o conteúdo do segmento como seqüência de inteiros de 16 bits
- Checksum: soma (complemento de 1 da soma) do conteúdo do segmento
- Transmissor coloca o valor do checksum no campo de checksum do UDP

Receptor:

- Computa o checksum do segmento recebido
- Verifica se o checksum calculado é igual ao valor do campo checksum:
  - NÃO - erro detectado
  - SIM - não há erros. **Mas talvez haja erros apesar disso**? Mas depois…

Exemplo – Internet Checksum

- Note que:
  - Ao se adicionar números, um *vai um* do bit mais significativo deve ser acrescentado ao resultado
- Exemplo: adicione dois inteiros de 16 bits

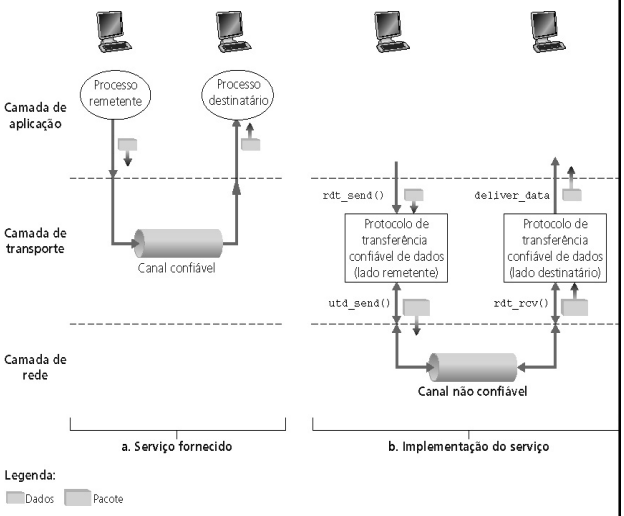
		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Camada de transporte

- 3.1 Serviços da camada de transporte
- 3.2 Multiplexação e demultiplexação
- 3.3 Transporte não orientado à conexão: UDP
- 3.4 Princípios de transferência confiável de dados
- 3.5 Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- 3.6 Princípios de controle de congestionamento
- 3.7 Controle de congestionamento do TCP

Princícios de transferência confiável de dados

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



Transferência confiável usando um canal com erro de bits

- Canal subjacente pode trocar valores dos bits num pacote
  - Checksum para detectar erros de bits
- *A* questão: como recuperar esses erros:
  - **Reconhecimentos (ACKs):** receptor avisa explicitamente ao transmissor que o pacote foi recebido corretamente
  - **Reconhecimentos negativos (NAKs):** receptor avisa explicitamente ao transmissor que o pacote tem erros
  - Transmissor reenvia o pacote quando da recepção de um NAK
- Mecanismos necessários:
  - Detecção de erros
  - Retorno do receptor: mensagens de controle (ACK, NAK) rcvr->sender

Transferência confiável usando um canal com erro de bits e perdas

O que acontece se o ACK/NAK é corrompido ou perdido?

- Transmissor não sabe o que aconteceu no receptor!
- Transmissor deve esperar durante um tempo razoável pelo ACK e se não recebe-lo deve retransmitir a informação
  - Não pode apenas retransmitir: possível duplicata

Tratando duplicatas:

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

Estratégias de Retransmissão

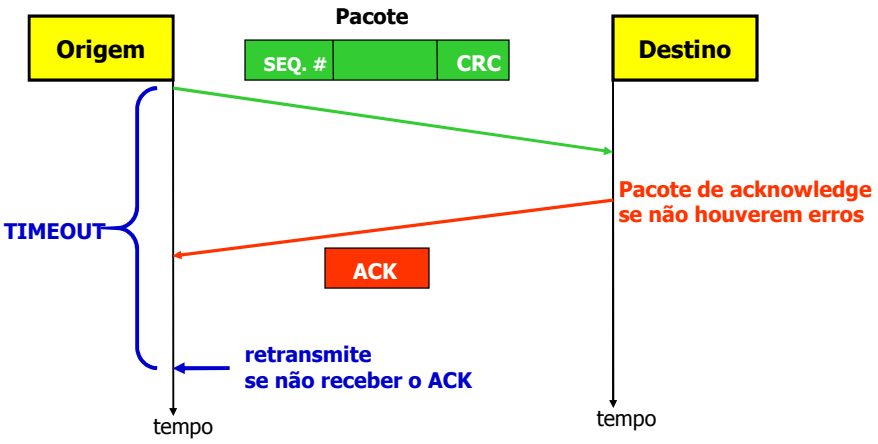
- Conhecidos como algoritmos ou protocolos ***Automatic Repeat Request*** (ARQ)
- Questões de projeto:
  - Como o receptor requisita uma retransmissão?
  - Como a fonte sabe quando retransmitir?
- Desempenho e exatidão
- Para simplificar as explicações assumiremos comunicações do tipo ponto-a-ponto

ARQ: hipóteses

- Comunicações ponto-a-ponto
  - Uma fonte e um destino
- Enlaces e nós seguem a ordem FIFO
- Os nós executam o protocolo ARQ corretamente
- A sessão entre fonte e destino já está inicializada e é permanente
- Todos os erros são detectados corretamente e o enquadramento é perfeito

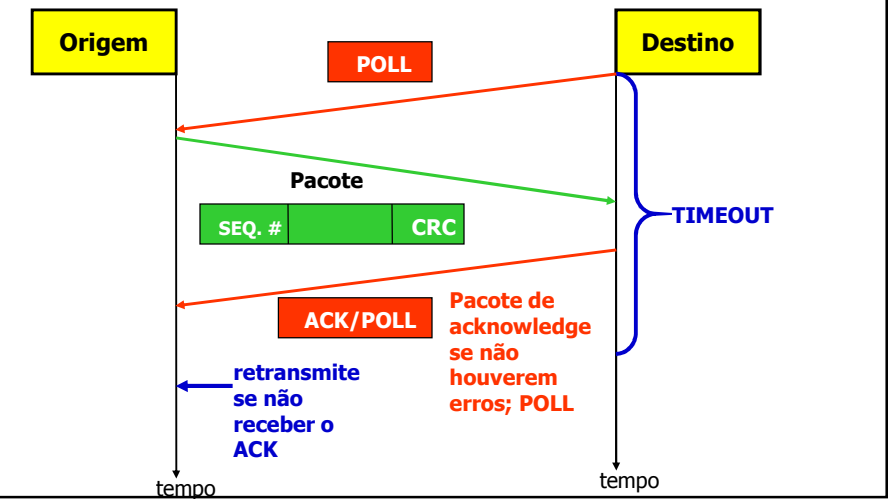
Esquema ARQ genérico em canais FIFO

Iniciado pela Origem



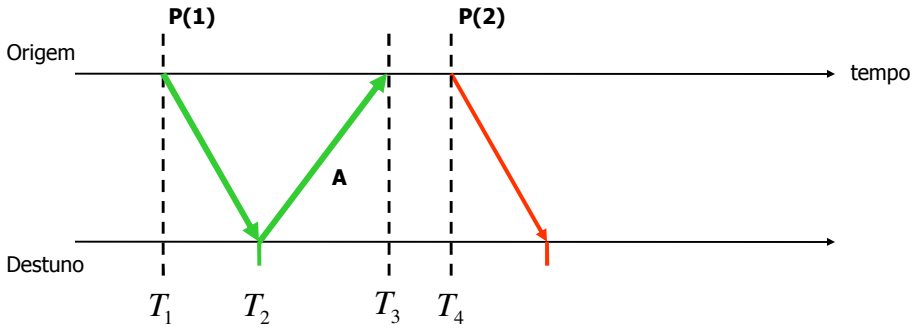
Esquema ARQ genérico em canais FIFO

Iniciado pelo Destino



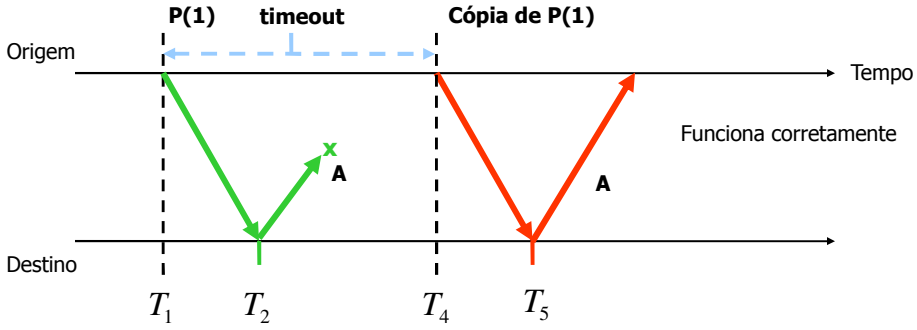
Exemplo de um ARQ simples

- Assuma que o destino rotule todo ACK com o mesmo rótulo (A) independentemente de qual pacote esteja sendo reconhecido.

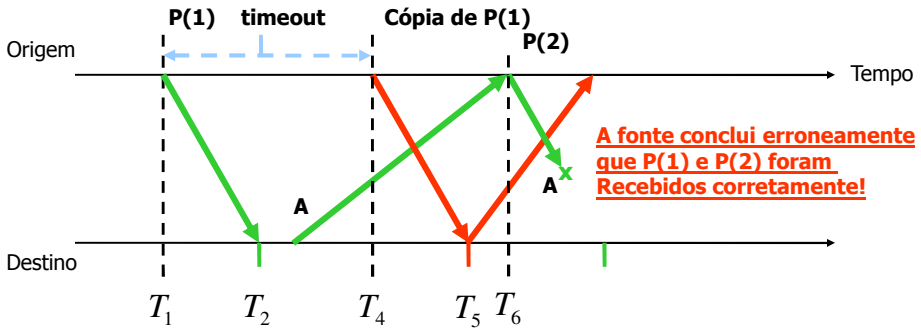


Erros podem confundir o destino

- Assuma que a fonte retransmite um pacote quando não recebe um ACK dentro de um determinado tempo fixo (i.e., ***timeout***).



Erros podem confundir o destino



Para evitar confusão na fonte, cada ACK deve referenciar qual pacote está sendo reconhecido!

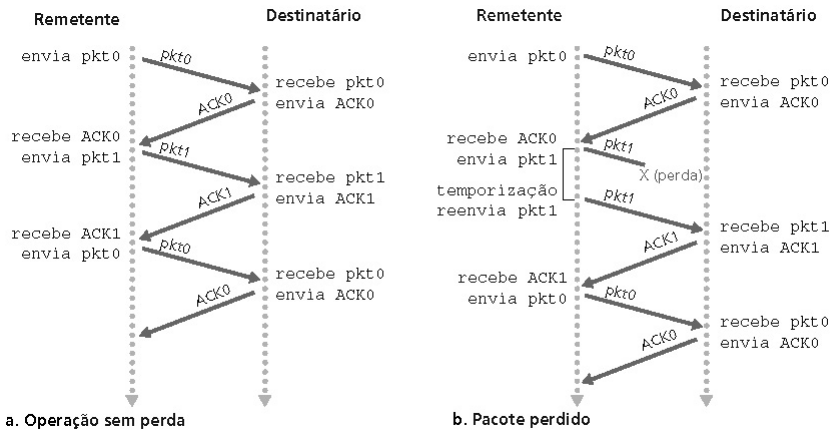
Requisitos em ARQ

- A fonte rotula cada pacote enviado utilizando um espaço de numeração sequencial.
- O destino envia um ACK para cada pacote que ele recebe sem erros e numera cada ACK com o número sequencial do pacote correspondente.
- Caso expire o tempo de espera na fonte sem receber a confirmação da recepção pelo destino isto resulta na retransmissão do pacote pela fonte.

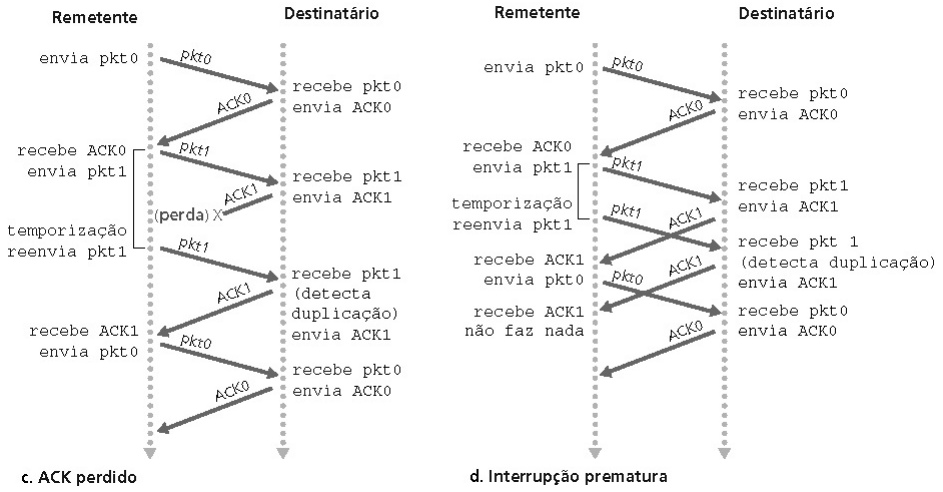
ARQ do tipo *Stop-and-Wait*

- Fonte transmite pacotes rotulados 1, 2,….. (ou apenas 1 bit: valores 0 ou 1)
- Destino envia ACK para todo pacote recebido corretamente sendo que o ACK especifica o próximo pacote esperado.
- Destino passa cópia do pacote recebido corretamente para o nível de rede descartando pacotes com erro.
- Fonte retransmite pacote sem confirmação após um tempo fixo de espera (i.e., *timeout*).
- Fonte e destino são incializados para enviar e receber pacote com número de sequência 1.

Stop-and-Wait Protocol (SWP)



Stop-and-Wait Protocol



Stop-and-Wait Protocol

SWP funciona, mas o desempenho é sofrível

- Exemplo: enlace de 1 Gbps, 15 ms de atraso de propagação, pacotes de 1 KB:

$$\text{Transmissão} = \frac{\text{L (tamanho do pacote em bits)}}{\text{R (taxa de transmissão, bps)}} = \frac{\text{8 kb/pkt}}{\text{10**9 b/s}} = 8 \text{ microsseg}$$

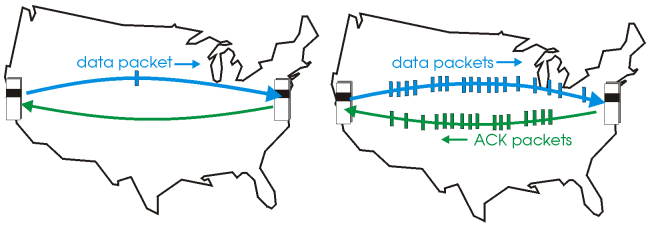
$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

- U_{sender}: utilização – fração de tempo do transmissor ocupado
- Um pacote de 1 KB cada 30 ms -> 33 kB/s de vazão sobre um canal de 1 Gbps

Solução

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

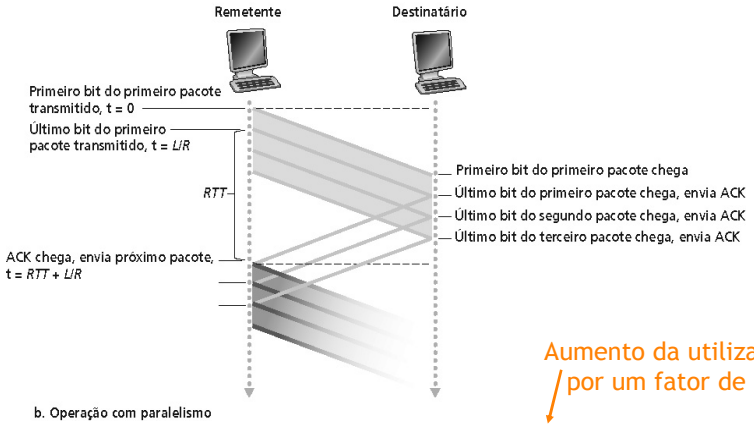
- Faixa de números de seqüência deve ser aumentada
- Armazenamento no transmissor e/ou no receptor



(a) operação do protocolo pare e espere (a) operação do protocolo com paralelismo

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

Solução



b. Operação com paralelismo

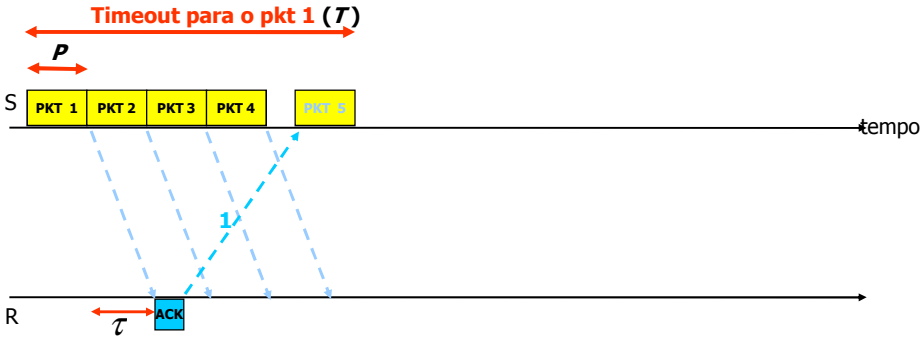
Aumento da utilização por um fator de 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{0{,}024}{30{,}008} = 0{,}0008$$

Go-Back-N (GBN) ARQ

- Com GBN, o destino descarta qualquer pacote fora de ordem; portanto, não necessita de um *buffer*.
- Destino confirma (i.e., ACK) um pacote recebido corretamente com o número de sequência do último pacote recebido em ordem.
- A fonte inicializa um tempo de espera para cada pacote transmitido. Caso não receba confirmação dentro deste tempo, a fonte retransmite o pacote expirado ***e todos os pacotes enviados após aquele pacote.***
- A fonte pode ter até ***W*** pacotes esperando por confirmação.

GBN ARQ Exemplo 1

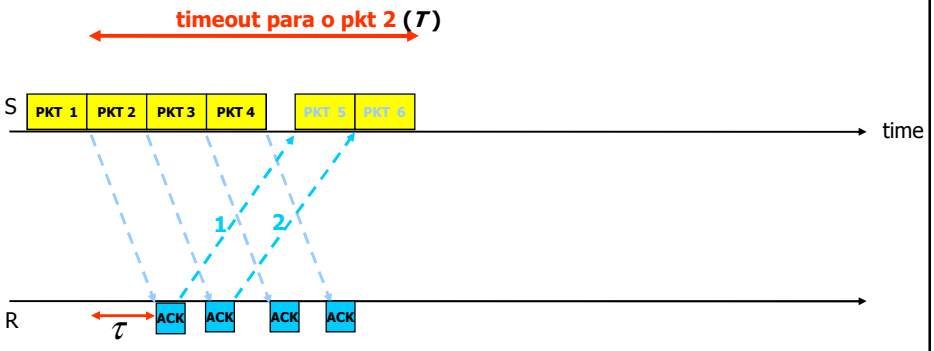


Packet 1 é recebido corretamente; destino envia ACK

Origem pode transmitir 4 pacotes antes que o timeout do pkt 1 expire

ACK do pkt 1 chega antes do seu timeout; origem pode enviar o pacote 5.

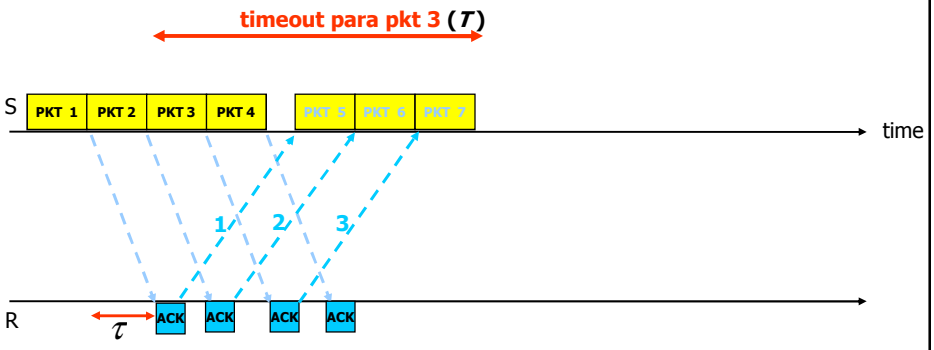
GBN ARQ Exemplo 1



Pacotes 2 a 4 chegam na ordem no receptor; receptor envia os ACKs para eles

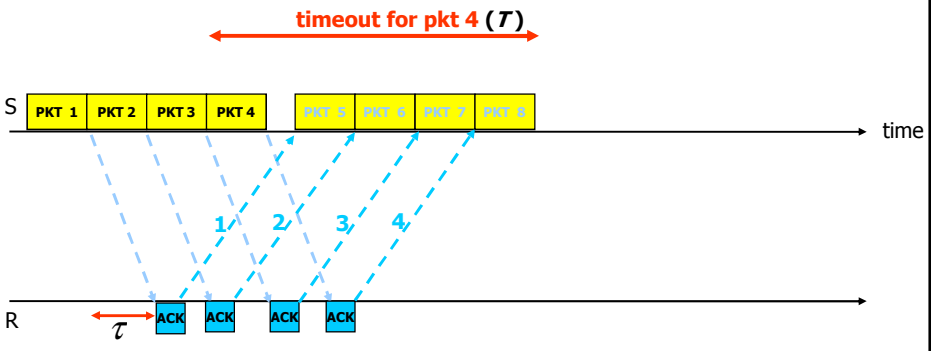
ACK do pkt 2 chega antes do seu timeout expirar; origem pode enviar o pacote 6.

GBN ARQ Exemplo 1



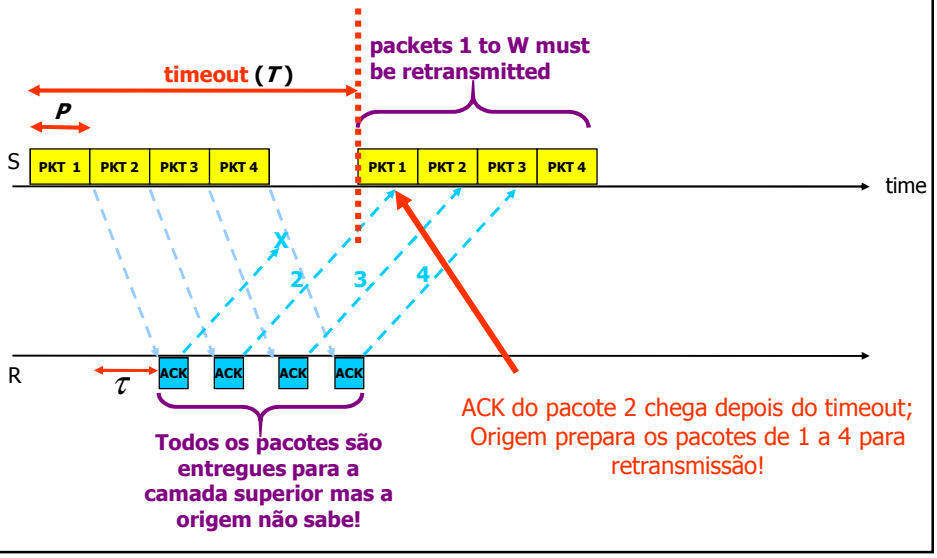
ACK do pkt 3 chega antes do seu timeout expirar; origem pode enviar o pacote 7.

GBN ARQ Exemplo 1



ACK do pkt 4 chega antes do seu timeout expirar; origem pode enviar o pacote 8, etc.

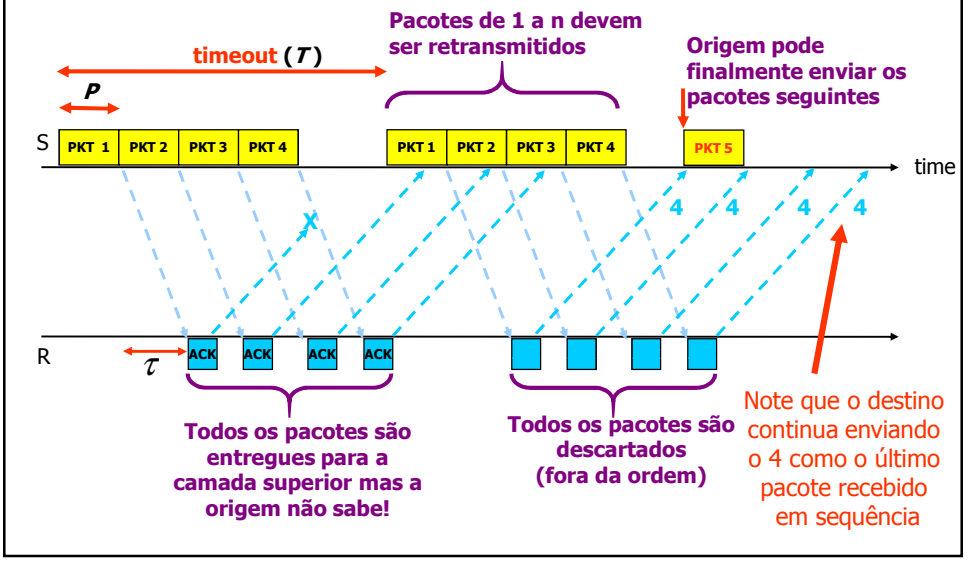
GBN ARQ Exemplo 2



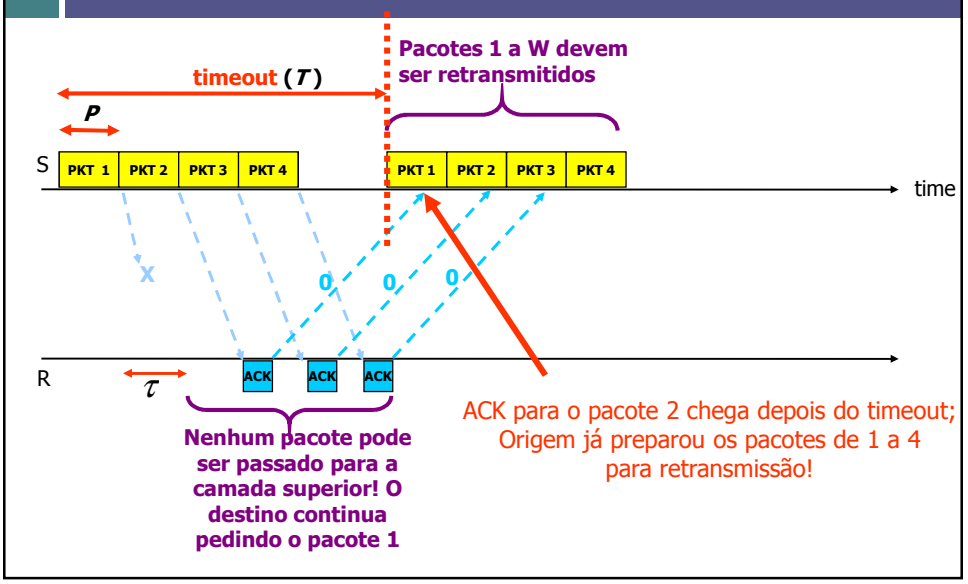
Todos os pacotes são entregues para a camada superior mas a origem não sabe!

ACK do pacote 2 chega depois do timeout; Origem prepara os pacotes de 1 a 4 para retransmissão!

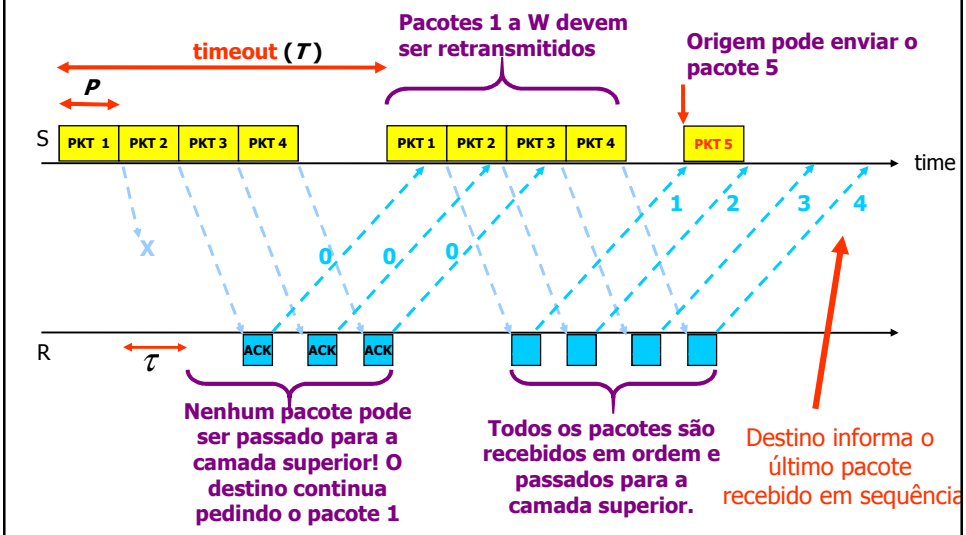
GBN ARQ Exemplo 2



GBN ARQ Exemplo 3



GBN ARQ Exemplo 3



Repetição Seletiva

- ***Motivação:*** SWP deixa a fonte ociosa por períodos longos esperando por ACKs e o GBN descarta pacotes que poderiam ser aproveitados.

- ***Solução:*** Permitir que o destino receba múltiplos pacotes e os armazene enquanto para preencher "buracos" nas transmissões.

Repetição Seletiva

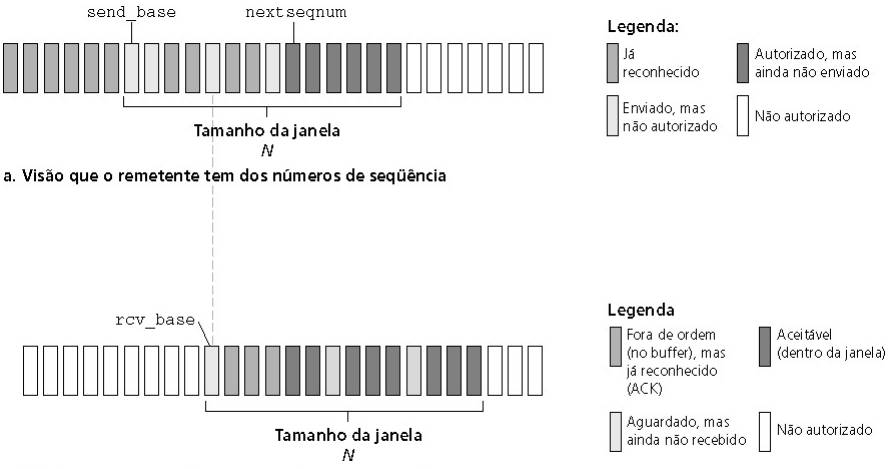
- ***Requisitos:***
  - Fonte e destino podem armazenar ***W*** pacotes
  - A fonte rotula os pacotes utilizando números consecutivos 1, 2,….
  - O destino armazena os pacotes recebidos sem erros, confirma os mesmos com ACKs, e os entrega ordenados ao nível superior.

  (e.g., se o pacote P1 tem erros e P2 e P3 estão corretos, o destino os segura até receber P1 corretamente)
  - A fonte mantém cópias dos pacotes transmitidos até que ela receba confirmação da recepção dos mesmos.
  - A fonte retransmite um pacote quando o ***timeout*** do mesmo expira sem ter recebido ACK
  - ACK referencia o número de sequência do pacote sendo confirmado

Repetição Seletiva

- ***Diversas variantes possíveis, assumiremos a seguinte:***
  - **ACK** como descrito anteriormente (confirmação de um determinado pacote com número de seq. *n*)
  - **ACK** também especifica qual o próximo pacote *NP* esperado pelo destino (i.e., qual o próximo pacote na sequência esperado pelo destino)
    - **Todos os pacotes no intervalo [*NP – W, NP – 1*] estão automaticamente confirmados. Por exemplo, com W=4 e números de sequência de 0 a 7, quando NP=2 significa que as mensagens com números de sequência 6, 7, 0 e 1 foram recebidas pelo destino.**

Repetição Seletiva



a. Visão que o remetente tem dos números de seqüência

b. Visão que o destinatário tem dos números de seqüência

Repetição Seletiva

TRANSMISSOR

Dados da camada superior:

- Se o próximo número de seqüência disponível está na janela, envia o pacote

Tempo de confirmação(n):

- Reenvia pacote n, restart timer

ACK (n) em [sendbase,sendbase+N]:

- Marca pacote n como recebido
- Se n é o menor pacote não reconhecido, avança a base da janela para o próximo número de seqüência não reconhecido

RECEPTOR

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

- Envia ACK(n)
- Fora de ordem: armazena
- Em ordem: entrega (também entrega pacotes armazenados em ordem), avança janela para o próximo pacote ainda não recebido

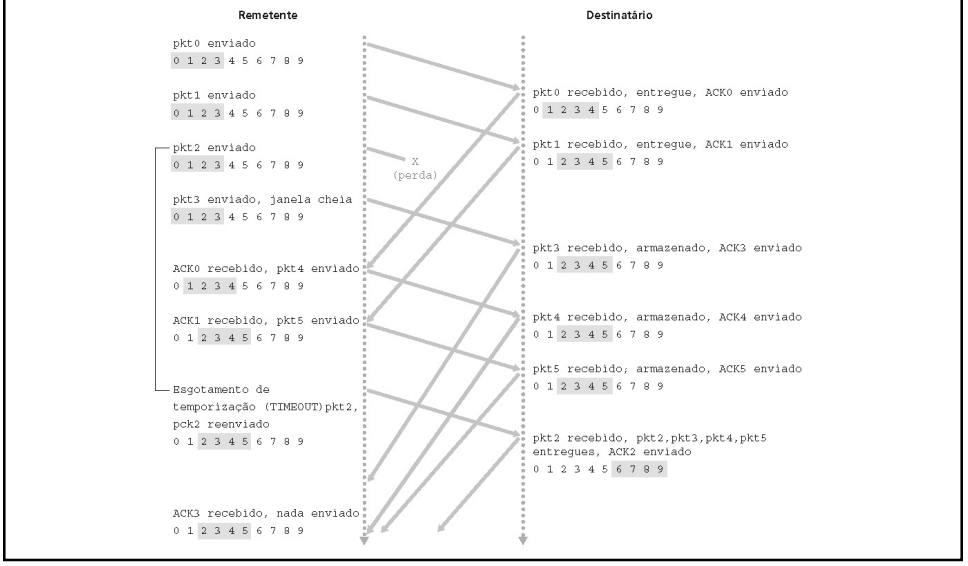
pkt n em [rcvbase-N,rcvbase-1]

- ACK(n)

Caso contrário:

- Ignora

Repetição Seletiva

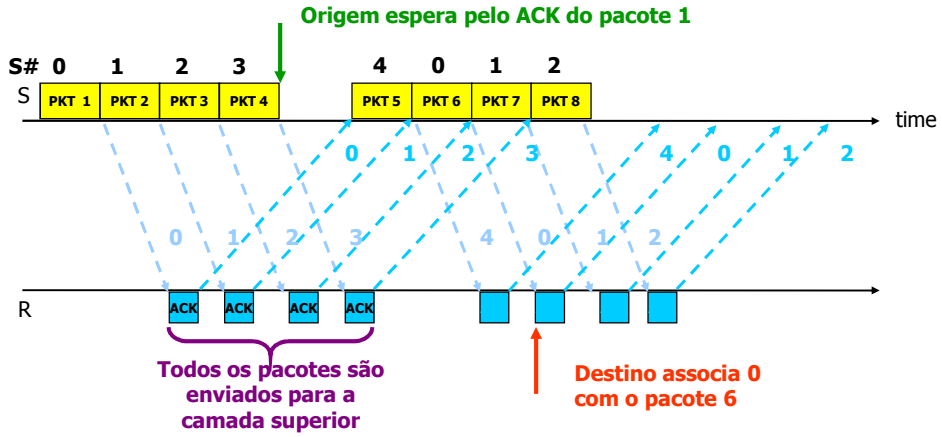


Correção de Repetição Seletiva

- O que provar:
  - ***Espaço finito de números de sequência:*** Qual é dimensão mínima do espaço de números de sequência de forma que a fonte e o destino nunca se confundam (i.e., número de sequência no ACK e pacotes).
    - *Deve-se assegurar que após o receptor ter ajustado a sua janela não haja nenhuma sobreposição com a janela original!!!*
  - ***Safety:*** O destino passa os pacotes em sequência para o nível superior, sem intervalos ou réplicas.
  - ***Liveness:*** Deadlocks nunca acontecem.

Números de sequência em SRP

- Assuma W=4 e 5 números de sequência (i.e., 0 a 4)



Números de sequência no SRP

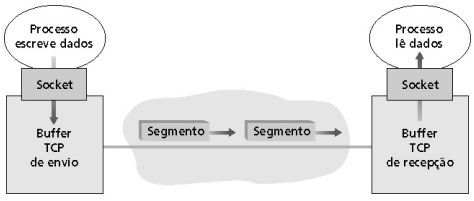
- **W>(#s de sequência)/2 pode confundir o receptor**
- Exemplo: considere W=3 e #Seq=4 (i.e., 0..3)
- Fonte envia pacotes com #seq 0, 1 e 2
- Assuma que o receptor recebe todos os pacotes e envia confirmacao ***mas todas as confirmacoes são perdidas***
- Neste ponto, o ***receptor avancou sua janela de recepcao*** para os #seq 3, 0 e 1
- A fonte não recebendo confirmacao vai retransmitir os pacotes #seq 0, 1 e 2
- O receptor vai aceitar os pacotes com #seq 0 e 1 porque está dentro da janela de recepção! ***Erroneamente aceita pacotes repetidos como pacotes novos!***
- Confirma a recepção de 0 e 1.
- A fonte recebendo ACK para #seq 0 e 1 avança a sua janela para #seq 2, 3 e 0...

Camada de transporte

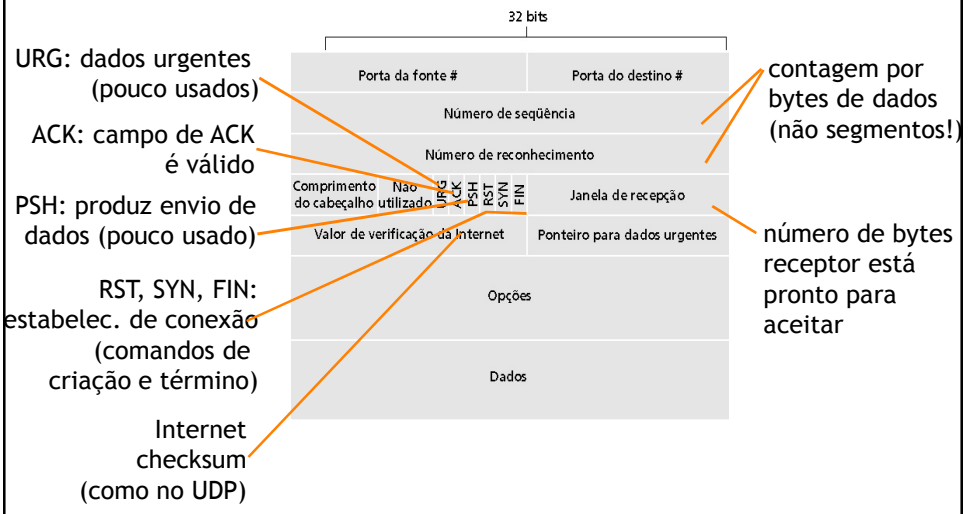
- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não-orientado à conexão: UDP
- Princípios de transferência confiável de dados
- Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- Controle de congestionamento do TCP

O Protocolo TCP

- **Ponto-a-ponto:**
  - Um transmissor, um receptor
- **Confiável, seqüencial byte stream:**
  - Não há contornos de mensagens
- **Pipelined:** (transmissão de vários pacotes sem confirmação)
  - Controle de congestionamento e de fluxo definem tamanho da janela
- **Buffers de transmissão e de recepção**
- Dados full-duplex:
  - Transmissão bidirecional na mesma conexão
  - MSS: maximum segment size
- Orientado à conexão:
  - Apresentação (troca de mensagens de controle) inicia o estado do transmissor e do receptor antes da troca de dados
- Controle de fluxo:
  - Transmissor não esgota a capacidade do receptor



Estrutura do Segmento TCP



Número de Sequência e ACKs no TCP

Números de seqüência:

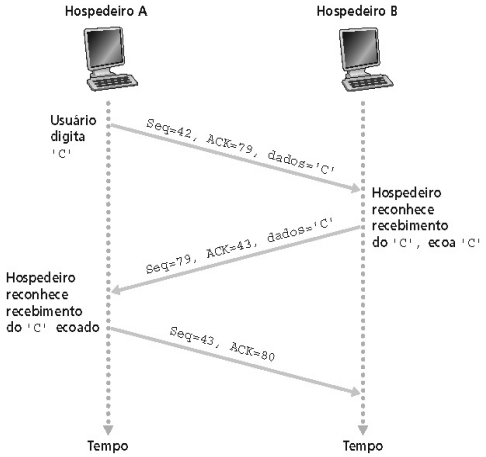
- Número do primeiro byte nos segmentos de dados

ACKs:

- Número do próximo byte esperado do outro lado
- ACK cumulativo

P.: Como o receptor trata segmentos fora de ordem?

- A especificação do TCP não define, fica a critério do implementador



TCP Round Trip Time e temporização

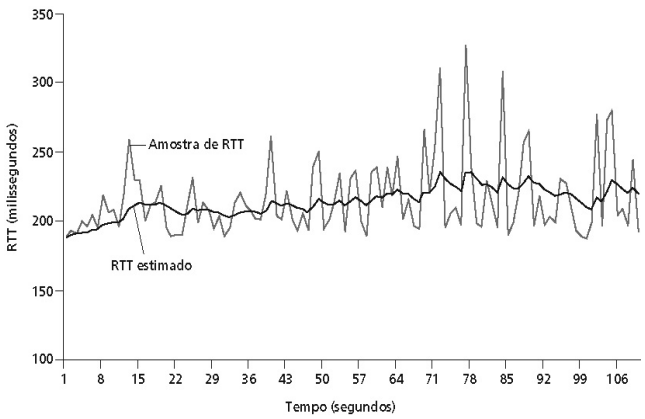
P.: como escolher o valor da temporização do TCP?

- Maior que o RTT
  - Nota: RTT varia
- Muito curto: temporização prematura
  - Retransmissões desnecessárias
- Muito longo: a reação à perda de segmento fica lenta

P.: Como estimar o RTT?

- SampleRTT: tempo medido da transmissão de um segmento até a respectiva confirmação
  - Ignora retransmissões e segmentos reconhecidos de forma cumulativa
- **SampleRTT** varia de forma rápida, é desejável um amortecedor para a estimativa do RTT
  - Usar várias medidas recentes, não apenas o último **SampleRTT** obtido

Exemplos de estimativa do RTT



- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não-orientado à conexão: UDP
- Princípios de transferência confiável de dados
- Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- Controle de congestionamento do TCP

TCP: transferência de dados confiável

- TCP cria serviços de trasferência confiável de dados em cima do serviço não-confiável do IP
- Transmissão de vários segmentos em paralelo (*Pipelined segments*)
- ACKs cumulativos
- TCP usa tempo de retransmissão simples
- Retransmissões são disparadas por:
  - Eventos de tempo de confirmação
  - ACKs duplicados
- Inicialmente, considere um transmissor TCP simplificado:
  - Ignore ACKs duplicados
  - Ignore controle de fluxo, controle de congestionamento

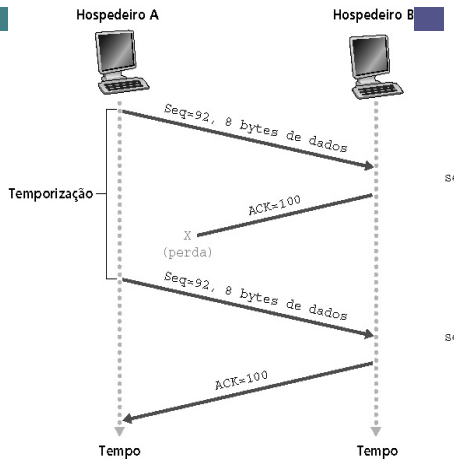
Eventos do transmissor TCP

Dado recebido da app:
- Crie um segmento com número de seqüência
- # seq é o número do byte-stream do 1º byte de dados no segmento
- Inicie o temporizador se ele ainda não estiver em execução (pense no temporizador para o mais antigo segmento não-confirmado)
- Tempo de expiração: `TimeOutInterval`

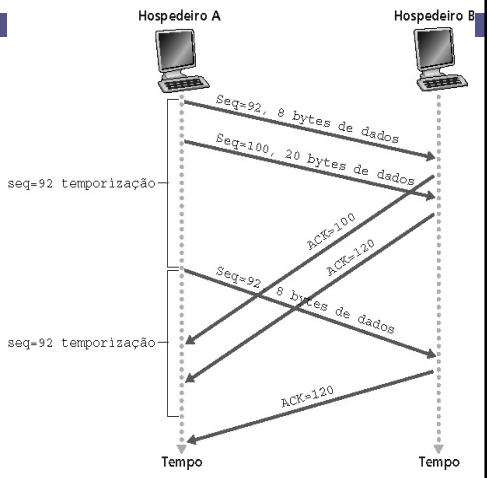
Tempo de confirmação:
- Retransmite o segmento que provocou o tempo de confirmação
- Reinicia o temporizador

ACK recebido:
- Quando houver o ACK de segmentos anteriormente não confirmados
  - Atualizar o que foi confirmado
  - Iniciar o temporizador se houver segmentos pendentes

TCP: cenários de retransmissão

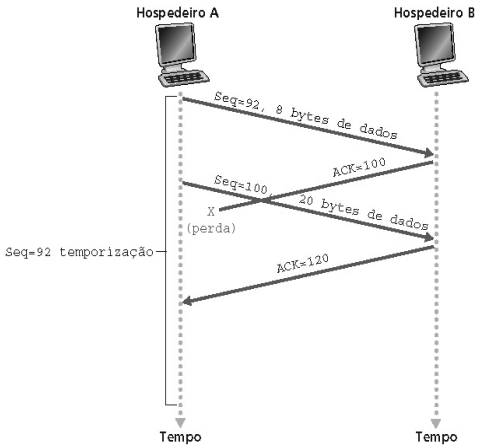


Cenário com perda do ACK



Temporização prematura, ACKs cumulativos

TCP: cenários de retransmissão



Cenário de ACK cumulativo

Geração de ACK [RFC 1122, RFC 2581]

Evento no receptor	Ação do receptor TCP
Segmento chega em ordem, não há lacunas, segmentos anteriores já aceitos	ACK retardado. Espera até 500 ms pelo próximo segmento. Se não chegar, envia ACK
Segmento chega em ordem, não há lacunas, um ACK atrasado pendente	Imediatamente envia um ACK cumulativo
Segmento chega fora de ordem, número de seqüência chegou maior: gap detectado	Envia ACK duplicado, indicando número de seqüência do próximo byte esperado
Chegada de segmento que parcial ou completamente preenche o gap	Reconhece imediatamente se o segmento começa na borda inferior do gap

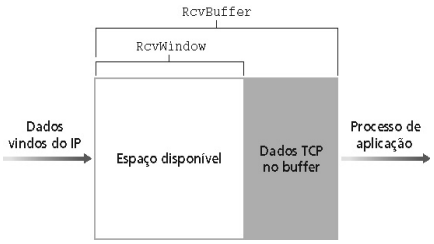
Retransmissão rápida

- Com freqüência, o tempo de expiração é relativamente longo:
  - Longo atraso antes de reenviar um pacote perdido
- Detecta segmentos perdidos por meio de ACKs duplicados
  - Transmissor freqüentemente envia muitos segmentos
  - Se o segmento é perdido, haverá muitos ACKs duplicados
- Se o transmissor recebe 3 ACKs para o mesmo dado, ele supõe que o segmento após o dado confirmado foi perdido:
  - Retransmissão rápida: reenvia o segmento antes de o temporizador expirar

- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado à conexão: UDP
- Princípios de transferência confiável de dados
- Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- Controle de congestionamento do TCP

TCP: controle de fluxo

- Lado receptor da conexão TCP possui um buffer de recepção:



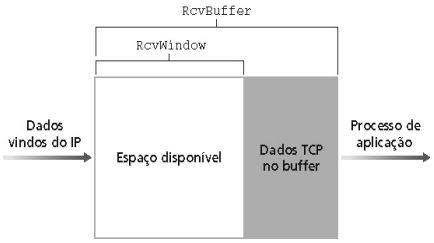
- Processos de aplicação podem ser lentos para ler o buffer

Controle de fluxo

Transmissor não deve esgotar os buffers de recepção enviando dados rápido demais

- Serviço de **speed-matching**: encontra a taxa de envio adequada à taxa de vazão da aplicação receptora

Controle de fluxo TCP: como funciona



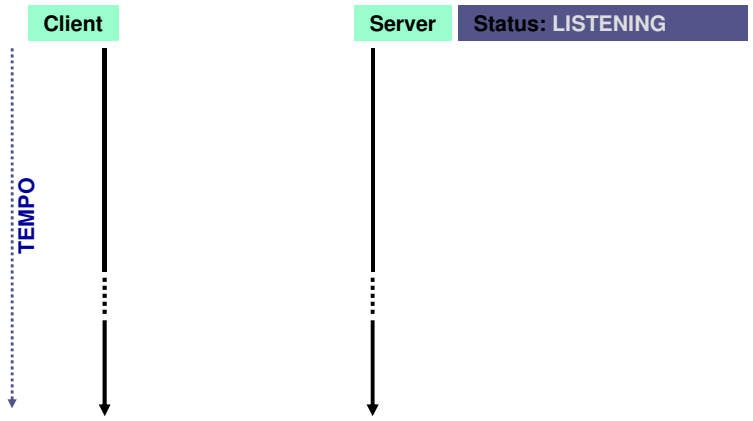
- Receptor informa a área disponível incluindo valor **`RcvWindow`** nos segmentos
- Transmissor limita os dados não confinados ao **`RcvWindow`**
  - Garantia contra overflow no buffer do receptor

- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado à conexão: UDP
- Princípios de transferência confiável de dados
- Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- Controle de congestionamento do TCP

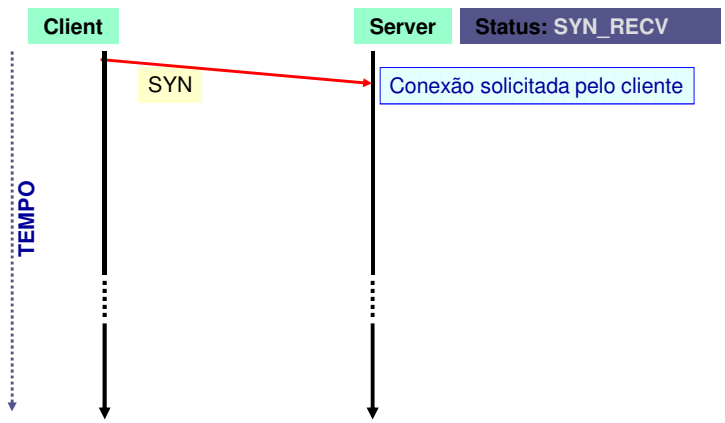
O Protocolo TCP

- Estabelecimento de Conexão
  - Protocolo
    - Passo 1: o cliente envia um segmento SYN especificando a porta do servidor ao qual deseja se conectar e seu número de sequência inicial
    - Passo 2: o servidor responde enviando outro segmento SYN com o ACK do segmento recebido e o seu próprio número de sequência
    - Passo 3: o cliente retorna um ACK e a conexão se estabelece
  - O tamanho máximo de segmento (MSS) que cada lado se propõe a aceitar também é definido no momento do estabelecimento da conexão
  - Pode acontecer um "half open"

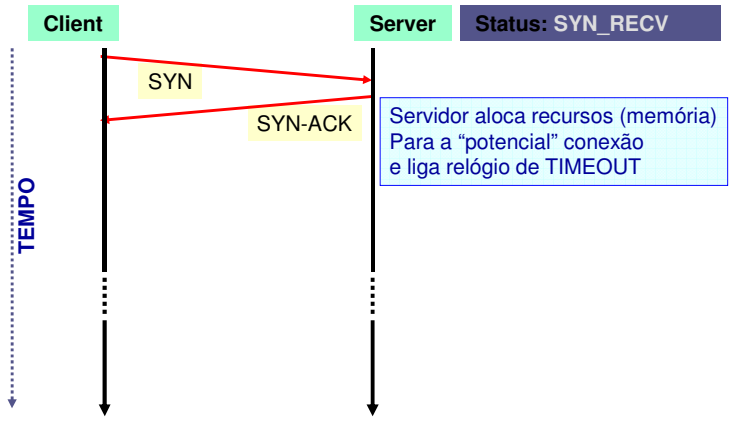
TCP – Como Funciona



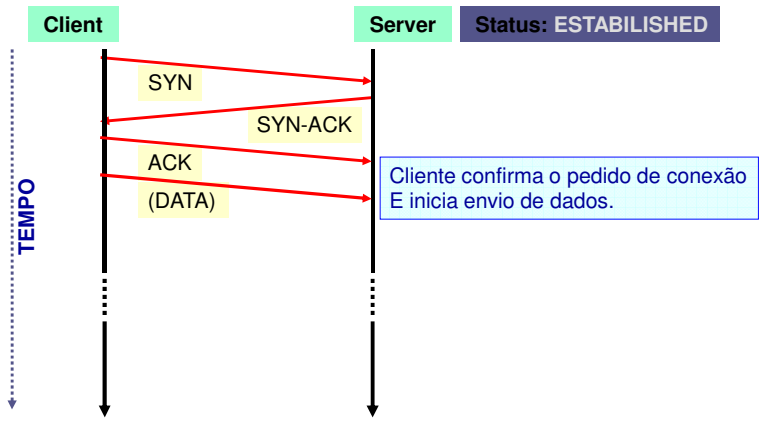
TCP – Como Funciona



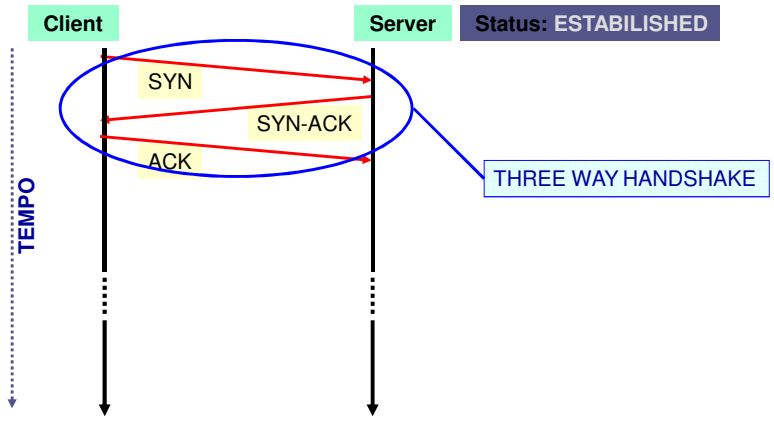
TCP – Como Funciona



TCP – Como Funciona

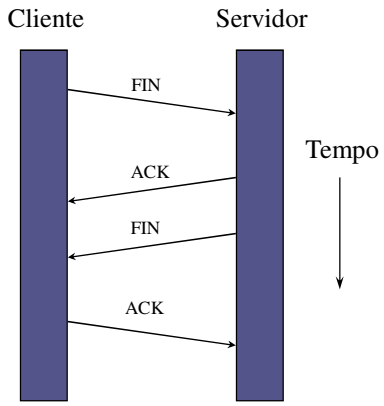


TCP – Como Funciona

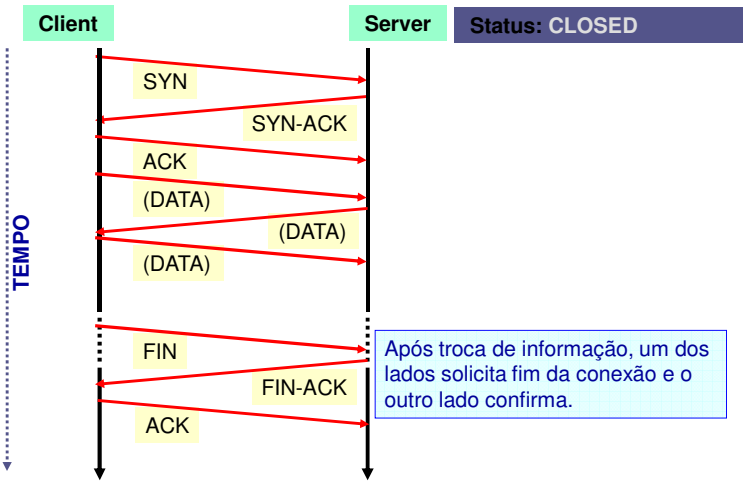


O Protocolo TCP

- Término de Conexão
  - Cada direção da conexão é encerrada independentemente
  - Protocolo
    - Passo 1: o cliente envia um segmento FIN
    - Passo 2: o servidor retorna um FIN e um ACK para o cliente
    - Passo 3: o cliente envia um ACK e a conexão se encerra
  - É possível efetuar um "half close", mantendo-se apenas uma conexão simplex



TCP – Como Funciona



- Serviços da camada de transporte
- Multiplexação e demultiplexação
- Transporte não orientado à conexão: UDP
- Princípios de transferência confiável de dados
- Transporte orientado à conexão: TCP
  - Estrutura do segmento
  - Transferência confiável de dados
  - Controle de fluxo
  - Gerenciamento de conexão
- Controle de congestionamento do TCP

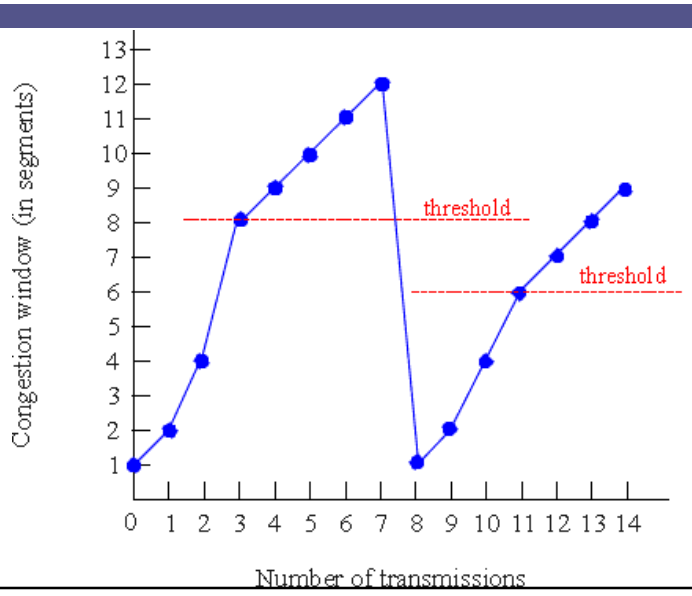
Janela de Congestionamento

- Uma conexão TCP controla sua taxa de transmissão limitando o seu número de segmentos que podem ser transmitidos sem que uma confirmação seja recebida
- Esse número é chamado o tamanho da janela do TCP (w)
- Uma conexão TCP começa com um pequeno valor de *w* e então o incrementa arriscando que exista mais largura de banda disponível
- Isso continua a ocorrer até que algum segmento seja perdido
- Nesse momento, a conexão TCP reduz *w* para um valor seguro, e então continua a arriscar o crescimento

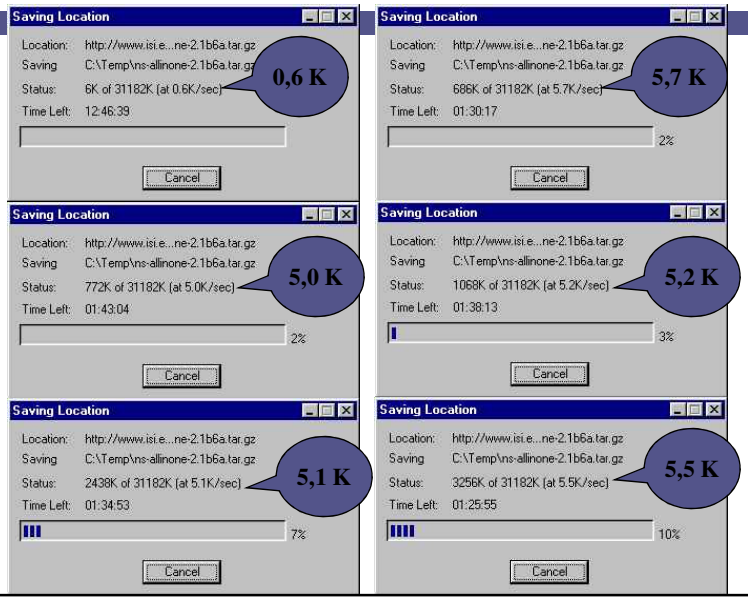
Controle de Congestionamento

- O controle é feito através de duas variáveis adicionadas em cada lado da conexão:
  - ***Janela de Congestionamento***
    - Janela do TCP explicada anteriormente
  - ***Limiar***
    - Serve para controlar o crescimento da janela de congestionamento

Graficamente ...



ontrole de Congestionamento TCP



Janela do Receptor

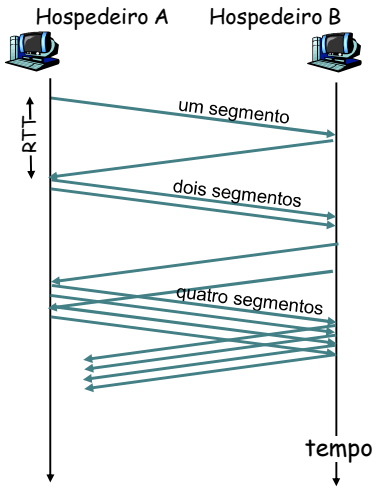
- O número máximo de segmentos não confirmados é dado pelo mínimo entre os tamanhos das janelas de congestionamento e do receptor.
  - ***Ou seja, mesmo que haja mais largura de banda, o receptor também pode ser um gargalo.***

Evolução de uma Conexão TCP

- No início, a janela de congestionamento tem o tamanho de um segmento.
  - Tal segmento tem o tamanho do maior segmento suportado.
- O primeiro segmento é enviado e então é esperado seu reconhecimento.
  - Se o mesmo chegar antes que ocorra o timeout, o transmissor duplica o tamanho da janela de congestionamento e envia dois segmentos.
  - Se esses dois segmentos também forem reconhecidos antes de seus timeouts, o transmissor duplica novamente sua janela, enviando agora quatro segmentos.

Evolução de uma Conexão TCP

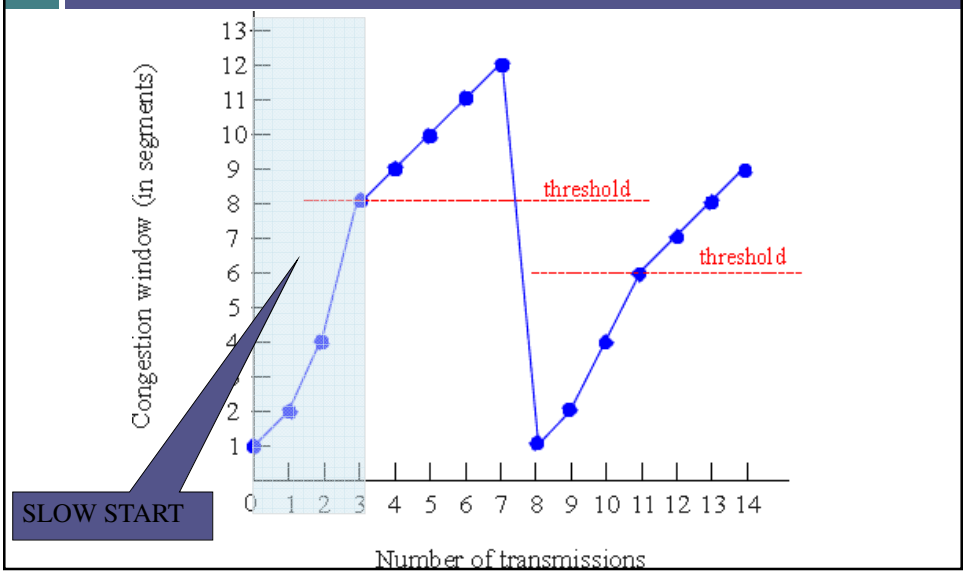
- Esse processo continua até que:
  - O tamanho da janela de congestionamento seja maior que o limiar, ou maior que o tamanho da janela do receptor;
  - Ocorra algum timeouts antes da confirmação.



Duas Fases dessa Evolução

- A primeira fase, em que a janela de congestionamento cresce exponencialmente é chamada de inicialização lenta (slow start), pelo fato de começar com um segmento
  - A taxa de transmissão começa pequena porém cresce muito rapidamente

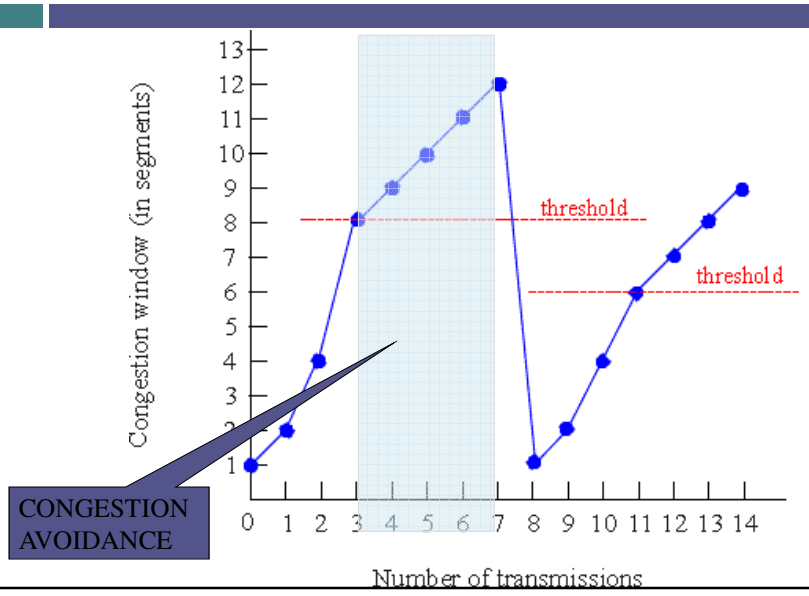
Graficamente ...



Duas Fases dessa Evolução

- Uma vez ultrapassado o limiar, e a janela do receptor ainda não seja um limitante, o crescimento da janela passa a ser linear.
- Essa *segunda fase* é chamada de prevenção de congestionamento (congestion avoidance).
  - Sua duração também depende da não ocorrência timeouts, e da aceitação do fluxo por parte do receptor.

Graficamente ...



E quando ocorrer um problema?

Evolução de uma Conexão TCP

- Na ocorrência de um timeout o TCP irá configurar:
  - O valor do limiar passa a ser a metade do tamanho atual da janela de congestionamento
  - O tamanho da janela de congestionamento volta ser do tamanho de um segmento
  - O tamanho da janela de congestionamento volta a crescer exponencialmente
- Caso ocorram 3 ACKs duplicados:
  - O valor do limiar é ajustado para metade tamanho atual da janela de congestionamento
  - O tamanho da janela de congestionamento passa igual ao valor do limiar (metade da janela de congestionamento atual)
  - O tamanho da janela de congestionamento cresce linearmente

Resumo

- Quando o tamanho da janela de congestionamento está abaixo do limiar, seu crescimento é exponencial
- Quando este tamanho está acima do limiar, o crescimento é linear
- Todas as vezes que ocorrer um timeout, o limiar é modificado para a metade do tamanho da janela e o tamanho da janela passa a ser 1
  - A rede não consegue entregar nenhum dos pacotes (“congestionamento pesado”)
- Quando ocorrem ACKs repetidos a janela cai pela metade
  - A rede ainda é capaz de entregar alguns pacotes (“congestionamento leve”)

Graficamente ...