

## A Camada de Transporte

Links desta videoaula - P1: <https://youtu.be/1PE2zAqTv60>  
P2: <https://youtu.be/48jbFhzU6TY>  
P3: <https://youtu.be/kmp1w2r5J0Q>

### Referências:

- Redes de Computadores. **A. S. Tanenbaum**. 5ª ed. Pearson, 2011 - Capítulo 6
- Redes de Computadores e a Internet. **J. Kurose, K. Ross**. Pearson, 2010 - Capítulo 3

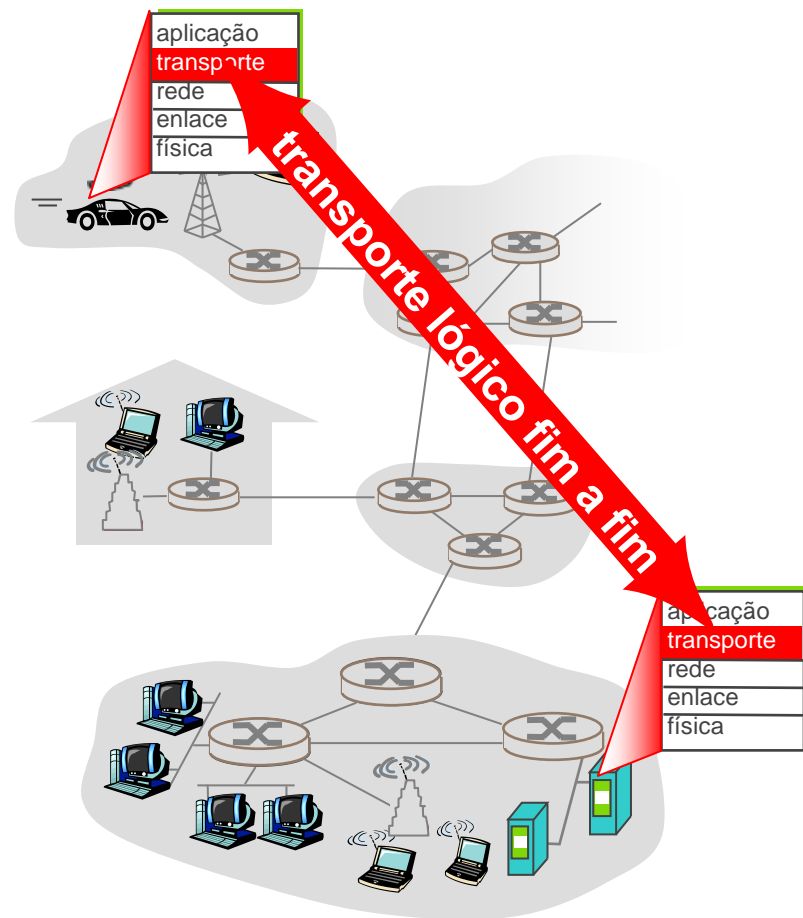
## Objetivos do capítulo:

- **entender princípios dos serviços da camada de transporte:**
  - Multiplexação / demultiplexação
  - transferência de dados confiável
  - controle de fluxo
  - controle de congestionamento
- **aprender sobre os protocolos da camada de transporte na Internet:**
  - UDP: transporte sem conexão
  - TCP: transporte orientado a conexão
  - controle de congestionamento TCP

- **5.1 Serviços da camada de transporte**
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- **5.3 Transporte não orientado para conexão: UDP**
- **5.4 Princípios da transferência confiável de dados**
- **5.5 Transporte orientado para conexão: TCP**
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
- **5.6 Princípios de controle de congestionamento**
- **5.7 Controle de congestionamento no TCP**

# Serviços e protocolos de transporte

- oferecem *comunicação lógica* entre processos de aplicação rodando em hospedeiros diferentes.
- protocolos de transporte rodam em sistemas finais.
  - **lado remetente:** divide as msgs da aplicação em *segmentos*, passa à camada de rede
  - **lado destinatário:** remonta os segmentos em msgs, passa à camada de aplicação
- **mais de um protocolo de transporte disponível às aplicações**
  - Internet: TCP e UDP



# Camada de Transporte vs Rede



- ***camada de rede:*** comunicação lógica entre hosts
  - uso de datagramas
  - Executada em roteadores e hosts (endereçamento)
- ***camada de transporte:*** comunicação lógica entre processos
  - amplia os serviços da camada de rede
  - Converte os dados recebidos da camada de aplicação em segmentos
  - Executada somente nos hosts

# Camada de Transporte vs Rede

## analogia com 2 famílias:

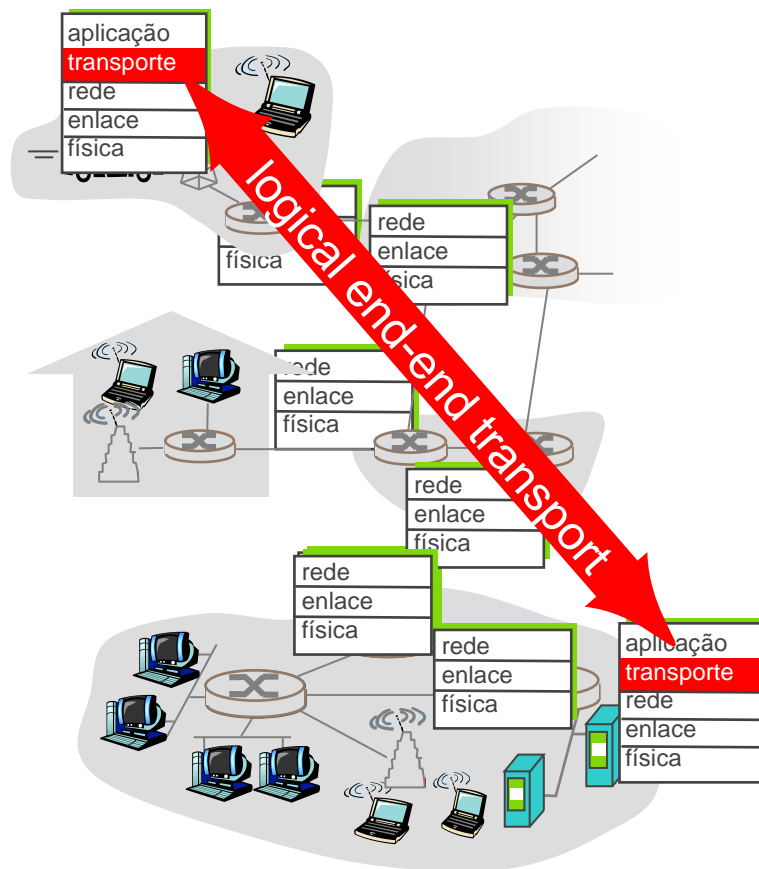


- Primos adoram escrever cartas uns para outros. Cada um dos 12 primos escreve ao outro uma vez por semana e a carta é entregue aos correios.
  - Total 144 cartas por semana (iriam economizar dinheiro com e-mail)
- Ana e Bill são os responsáveis por distribuir as cartas aos seus irmãos
  - **processos** = crianças
  - **msgs da aplicação** = cartas nos envelopes
  - **hosts** = casas
  - **protocolo de transporte** = Ana e Bill
  - **protocolo da camada de rede** = serviço postal

# Protocolos da camada de transporte

- **remessa confiável e em ordem (TCP)**
  - controle de congestionamento
    - Não direcionado à aplicação solicitante, mas à Internet como um todo.
  - controle de fluxo
  - estabelecimento da conexão
- **remessa não confiável e desordenada: UDP**
  - extensão sem luxo do IP pelo “melhor esforço”
- **serviços não disponíveis:**
  - garantias de atraso
  - garantias de largura de banda

**Limitados pelo modelo de serviço da camada de rede.**



# Protocolos da camada de transporte

| Serviço                     | Protocolo de aplicação | Protocolo de transporte |
|-----------------------------|------------------------|-------------------------|
| Transferência de arquivos   | FTP                    | TCP                     |
| Terminal remoto             | Telnet                 | TCP                     |
| Correio eletrônico          | SMTP                   | TCP                     |
| Serviço Web                 | HTTP                   | TCP                     |
| Trivial FTP                 | TFTP                   | UDP                     |
| Endereçamento dinâmico      | DHCP                   | UDP                     |
| Gerência remota             | SNMP                   | UDP                     |
| Serviço de nomes            | DNS                    | UDP/TCP                 |
| Serviço de arquivos remotos | NFS                    | UDP/TCP                 |



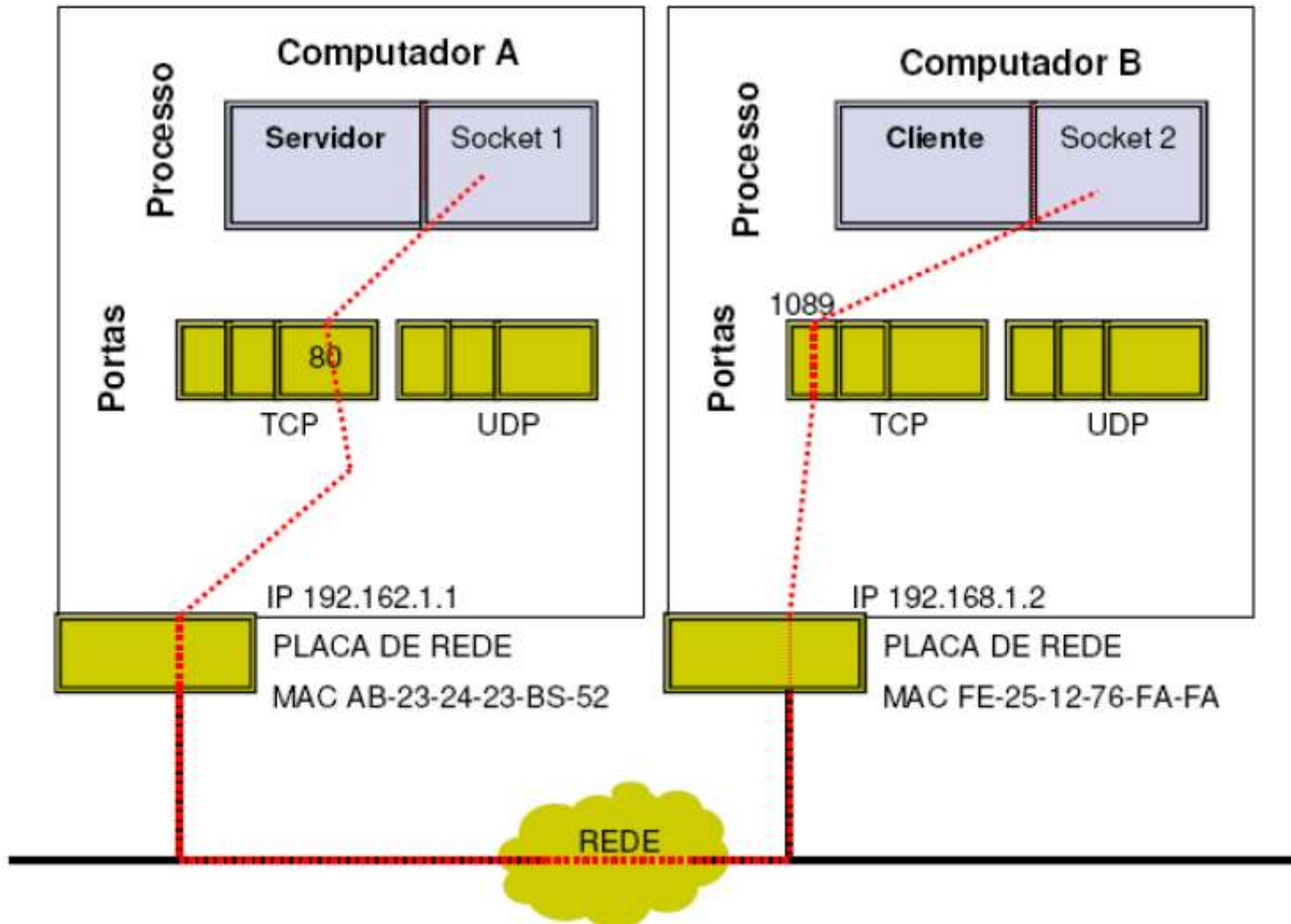
- 5.1 Serviços da camada de transporte
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- 5.3 Transporte não orientado para conexão: UDP
- 5.4 Princípios da transferência confiável de dados
- 5.5 Transporte orientado para conexão: TCP
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
- 5.6 Princípios de controle de congestionamento
- 5.7 Controle de congestionamento no TCP

- **Ampliação do serviço de entrega host a host para um serviço de entrega processo a processo.**
- **Suponha a seguinte situação:**
  - Você está com seu smartphone acessando 3 páginas web e executando outros 4 apps simultaneamente.
  - A camada de transporte deve entregar o segmento à aplicação correta
- **Sockets:** portas por onde os dados passam da rede para o processo e do processo para a rede.

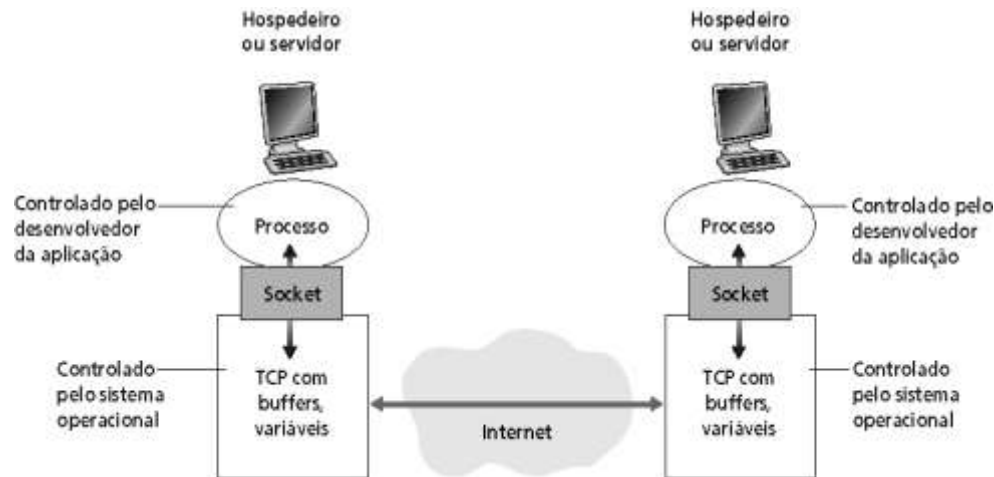
- **Mecanismo básico de comunicação sobre IP.**
- **Histórico da API** (*application programming interface*) **sockets:** desenvolvida na época da implementação dos protocolos TCP/IP pelo grupo da UC, Berkeley.
- **Representação interna do S.O. para um ponto de comunicação**
- Permite que os processos acessem os serviços de rede;
- A troca de mensagens é feita entre dois processos usando vários mecanismos de transporte.
- Programador controla tudo que existe no lado da camada de aplicação do socket, mas tem pouco controle do lado da camada de transporte do socket.

# Sockets

## Estrutura De Funcionamento de um Socket



- Um processo envia/recebe mensagens para/de seu **socket**
- O socket é análogo a uma porta
  - O processo de envio empurra a mensagem para fora da porta.
  - Processo de envio confia na infra-estrutura de transporte no outro lado da porta que leva a mensagem para o socket no processo de recepção.
- API: (1) escolha do protocolo de transporte; (2) habilidade para fixar poucos parâmetros (será explicado mais tarde)



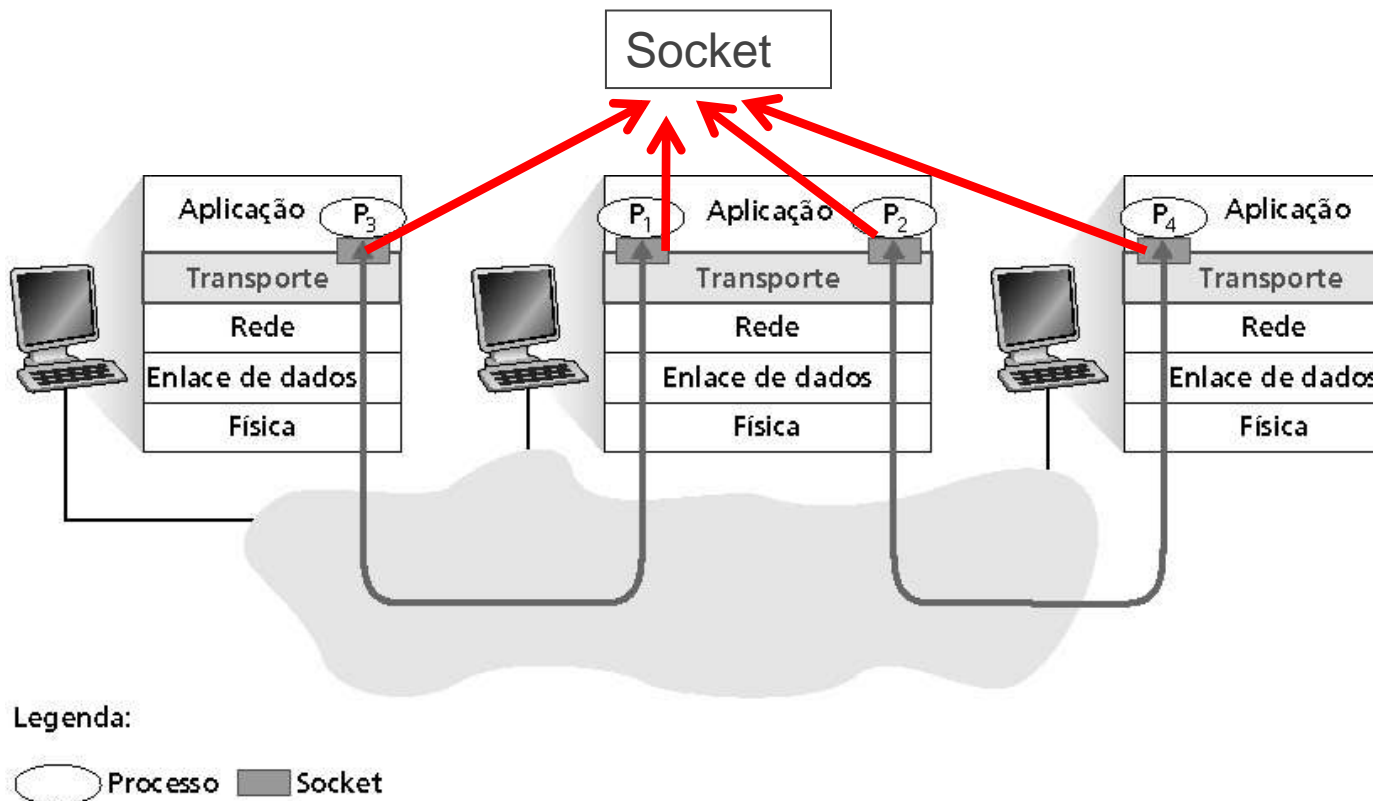
# Multiplexação / demultiplexação

## Multiplexação no host emissor:

Coleta dados de múltiplos sockets, envelope os dados com cabeçalho (usado depois para demultiplexação)

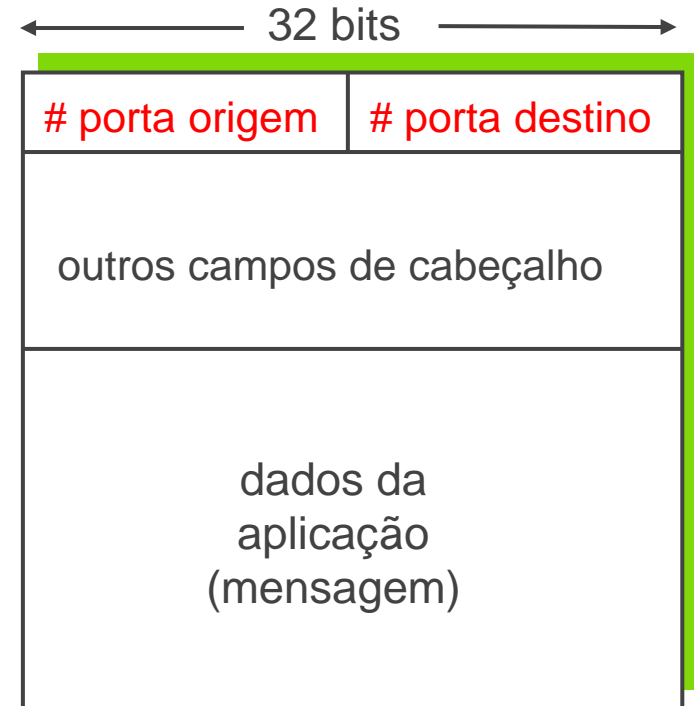
## Demultiplexação no receptor:

Entrega os segmentos recebidos ao socket correto



# Como funciona a demultiplexação

- **host recebe datagramas IP**
  - cada datagrama tem end. IP de origem, endereço IP de destino;
  - cada datagrama carrega 1 segmento da camada de transporte;
  - cada segmento tem número de porta de origem, destino.
- **host usa end. IP e nº de porta para direcionar segmento ao socket apropriado**



formato do segmento TCP/UDP

- **A multiplexação requer:**
  - Portas com identificadores exclusivos - 16 bits (0 a 65535)
- **Portas bem conhecidas: 0 a 1023** (RFC 3232 - <http://www.iana.org>)
  - Reservadas para uso em protocolos de aplicações bem conhecidas.

| Porta | Protocolo de aplicação | Descrição                 |
|-------|------------------------|---------------------------|
| 20/21 | FTP                    | Transferência de arquivos |
| 23    | Telnet                 | Terminal remoto           |
| 25    | SMTP                   | Correio eletrônico        |
| 53    | DNS                    | Serviço de nomes          |
| 80    | HTTP                   | Serviço Web               |

- **Portas registradas: 1024 a 49151**
  - Usadas por serviços proprietários.
- **Portas privadas (dinâmicas): 49152 a 65535**
  - Usadas por clientes e serviços não padronizados.

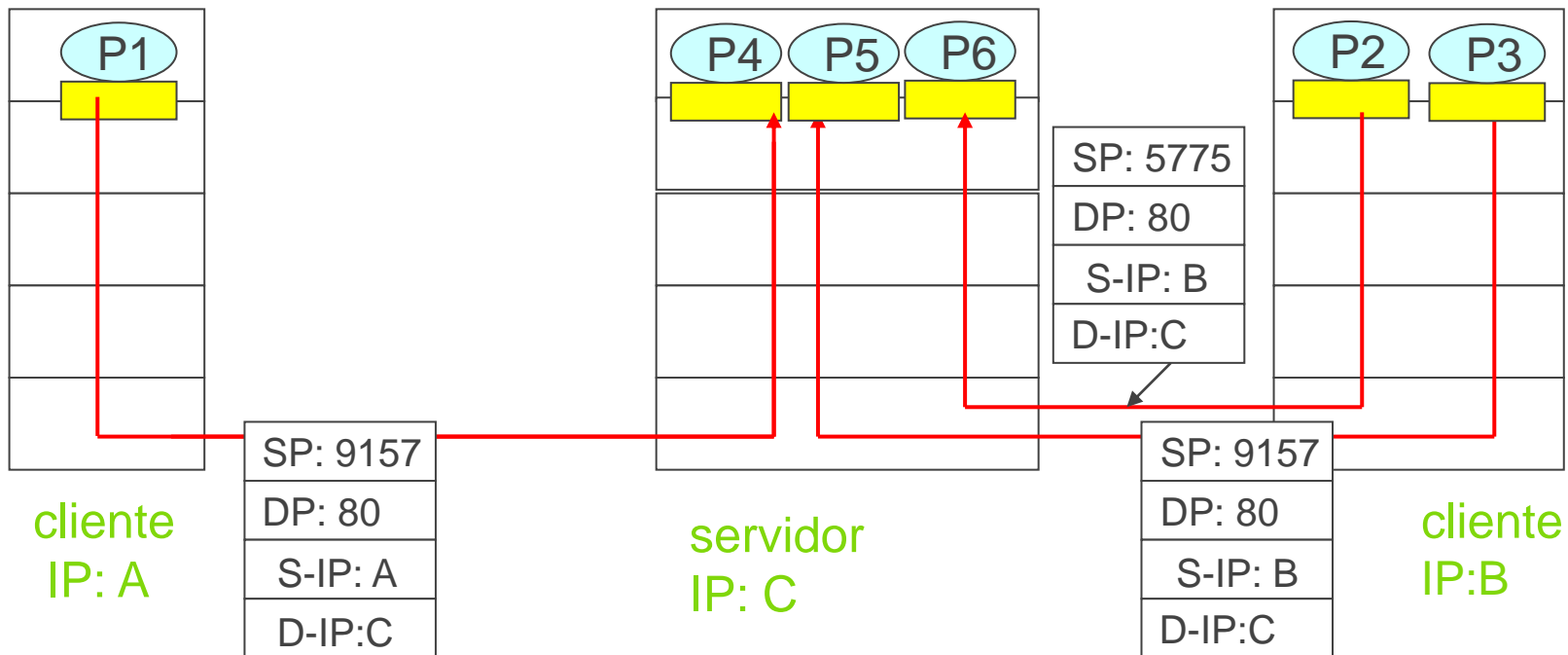


# Demultiplexação TCP (orientada a conexão)



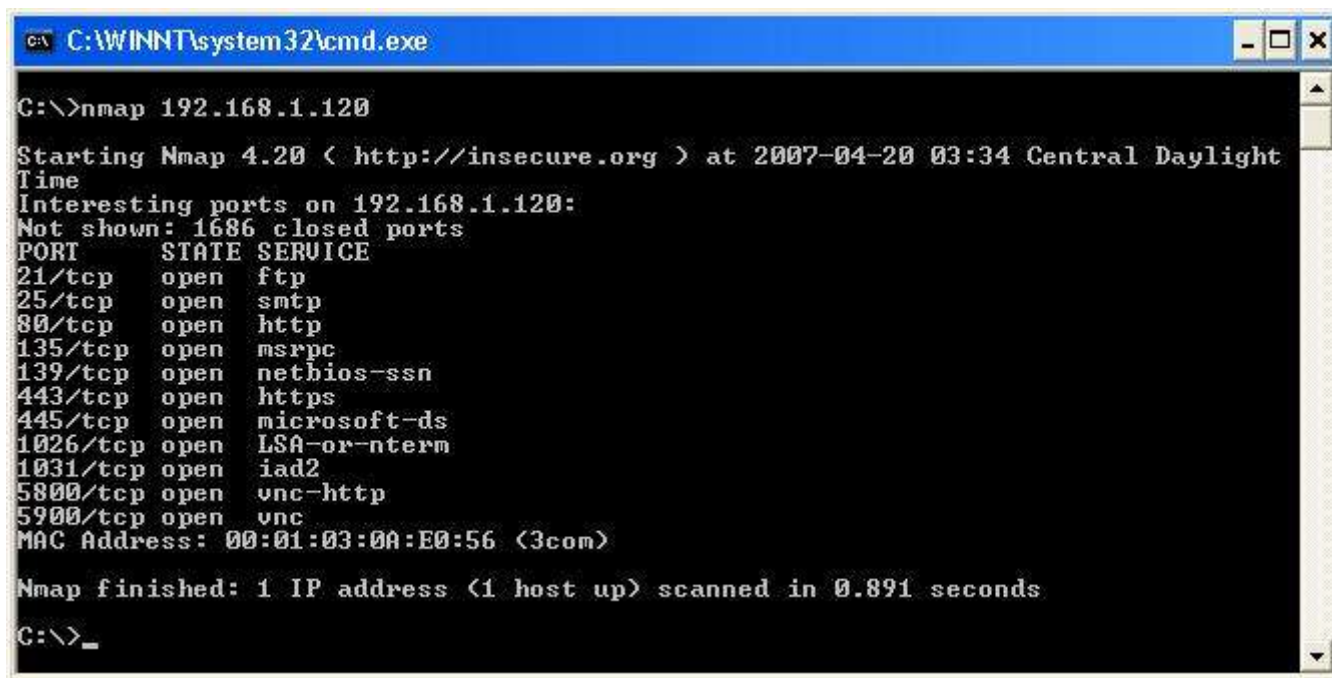
- **socket TCP identificado por tupla de 4 elementos:**
  - endereço IP de origem
  - número de porta de origem
  - endereço IP de destino
  - número de porta de destino
- **hospedeiro destinatário usa todos os quatro valores para direcionar segmento ao socket apropriado**
- **hospedeiro servidor pode admitir muitos sockets TCP simultâneos:**
  - cada socket identificado por usa própria tupla de 4
- **servidores Web têm diferentes sockets para cada cliente conectando**
  - HTTP não persistente terá diferentes sockets para cada requisição

# Demultiplexação orientada para conexão



# Varredura de portas

- **Processo servidor espera pacientemente em um porta aberta para contato por um cliente remoto.**
  - É possível mapear as portas abertas em servidores e hosts
  - Muitos malwares exploram falhas de aplicações através de portas abertas.
- **Técnica utilizada para invasão e para análise de vulnerabilidades.**



```
C:\WINNT\system32\cmd.exe

C:\>nmap 192.168.1.120

Starting Nmap 4.20 ( http://insecure.org ) at 2007-04-20 03:34 Central Daylight
Time
Interesting ports on 192.168.1.120:
Not shown: 1686 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
25/tcp    open  smtp
80/tcp    open  http
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
443/tcp   open  https
445/tcp   open  microsoft-ds
1026/tcp  open  LSA-or-nterm
1031/tcp  open  iad2
5800/tcp  open  vnc-http
5900/tcp  open  vnc
MAC Address: 00:01:03:0A:E0:56 (3com)

Nmap finished: 1 IP address (1 host up) scanned in 0.891 seconds

C:\>_
```

- **5.1 Serviços da camada de transporte**
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- **5.3 Transporte não orientado para conexão: UDP**
- **5.4 Princípios da transferência confiável de dados**
- **5.5 Transporte orientado para conexão: TCP**
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
- **5.6 Princípios de controle de congestionamento**
- **5.7 Controle de congestionamento no TCP**

# UDP: User Datagram Protocol [RFC 768]

- **Projetado para ser um protocolo + simples e básico possível**

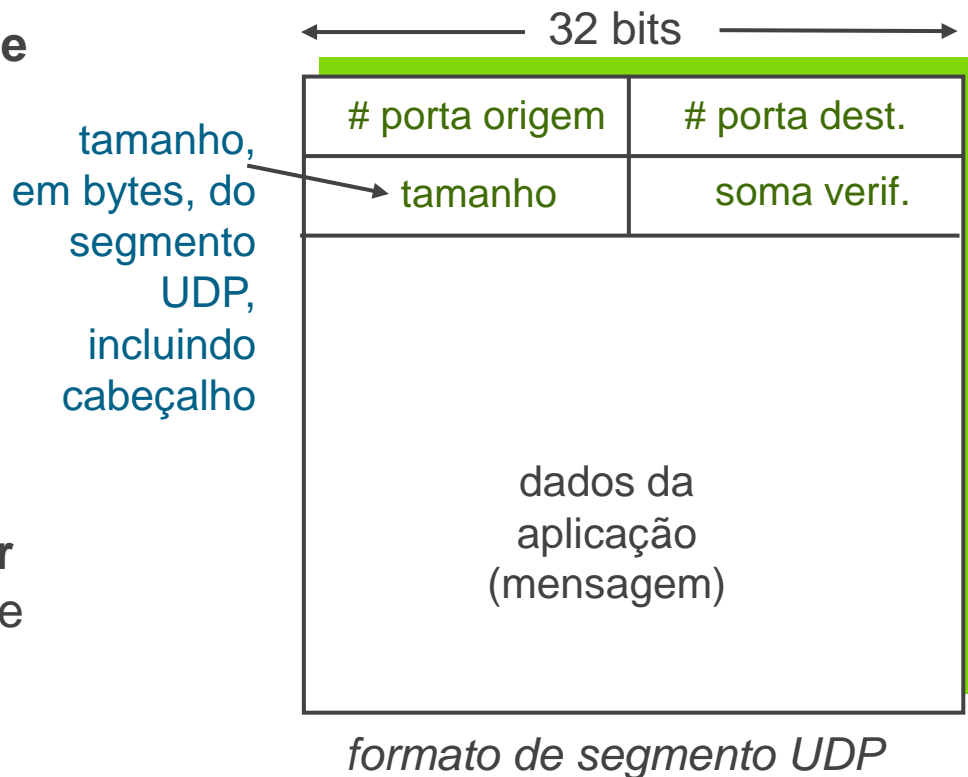
- Aplicação quase “falando” diretamente com o IP.

- protocolo de transporte da Internet “sem luxo”, básico
- serviço de “melhor esforço”, segmentos UDP podem ser:
  - Perdidos;
  - entregues à aplicação fora da ordem.
- **sem conexão:**
  - sem handshaking entre remetente e destinatário UDP
  - cada segmento UDP tratado independente dos outros

## Por que existe um UDP?

- **sem estabelecimento de conexão** (que pode gerar atraso)
- **simples:** sem estado de conexão no remetente, destinatário
- **cabeçalho de segmento pequeno**
- **sem controle de congestionamento:** UDP pode transmitir o mais rápido possível

- normalmente usado para streaming de aplicações de multimídia
  - tolerante a perdas
  - sensível à taxa
- outros usos do UDP
  - DNS
  - SNMP
- transferência confiável por UDP: aumenta confiabilidade na camada de aplicação
  - recuperação de erro específica da aplicação!



# TCP x UDP

TCP

data

UDP

data

# Soma de verificação UDP



**objetivo:** detectar erros (ex: bits invertidos) no segmento transmitido

## **remetente:**

- trata conteúdo de segmento como sequência de inteiros de 16 bits
- soma de verificação (*checksum*): adição (soma por complemento de 1) do conteúdo do segmento
- remetente coloca valor da soma de verificação no campo de soma de verificação UDP

## **destinatário:**

- calcula soma de verificação do segmento recebido
- verifica se soma de verificação calculada igual ao valor do campo de soma de verificação:
  - NÃO – erro detectado
  - SIM – nenhum erro detectado.



# Ex. soma de verificação da Internet

✓ Cálculo do *checksum* – Definido no RFC 1071

✓ Ex: 3 palavras 16 bit: **A**:0110011001100110 ; **B**:0101010101010101 ; **C**:0000111100001111

## Emissor:

1 – Soma uma a uma com "carry"

**A** 0110011001100110

**B** 0101010101010101

-----

**R1** 1011101110111011

1011101110111011

**C** 0000111100001111

-----

**R2** 1100101011001010

2 – *Checksum* é o complemento para 1 do resultado (inverter)

**Checksum:** 0011010100110101

## Receptor:

1 – As 4 palavras de 16 bit são somadas (incluindo o checksum)

**R2** 1100101011001010  
**Checksum:** 0011010100110101

-----  
1111111111111111

2 – Todos os bits a 1 : **Não detecta erro**  
Um ou mais bits a 0 : **Detecta erro**

É possível que uma aplicação tenha transferência confiável de dados usando UDP?

- **Sim, desde que a confiabilidade esteja embutida na aplicação.**
- Assim as aplicações podem se comunicar de maneira confiável sem ter de se sujeitar às limitações da taxa de transmissão impostas pelo mecanismo de controle de congestionamento do TCP.
- Muitas aplicações proprietárias de áudio e vídeo fazem exatamente isto – rodam sobre UDP, mas dispõem de reconhecimentos e retransmissões embutidos na aplicação para reduzir a perda de pacotes.

- **5.1 Serviços da camada de transporte**
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- **5.3 Transporte não orientado para conexão: UDP**
- **5.4 Princípios da transferência confiável de dados**
- **5.5 Transporte orientado para conexão: TCP**
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
- **5.6 Princípios de controle de congestionamento**
- **5.7 Controle de congestionamento no TCP**

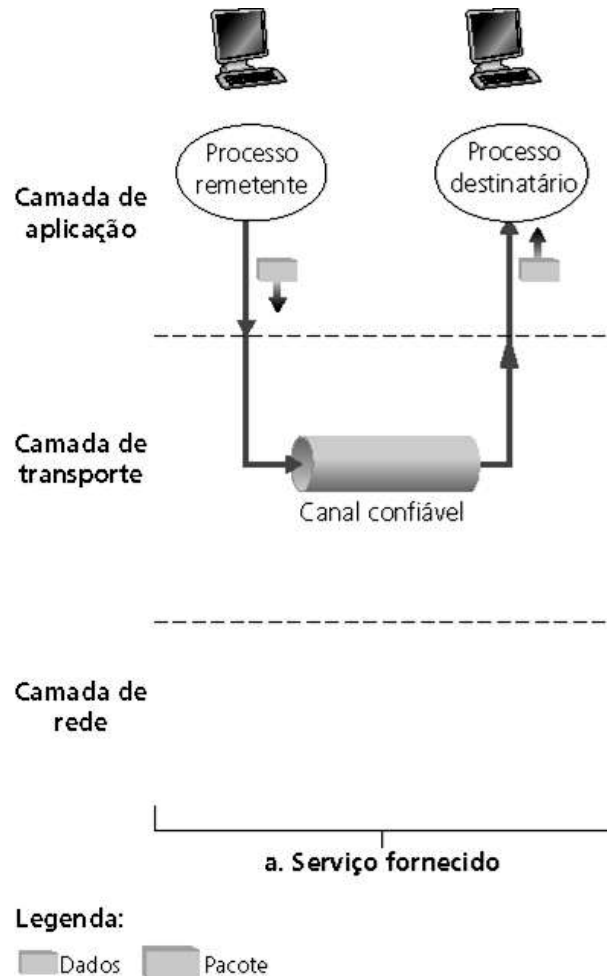
- Top 10 na lista das funções mais importantes em Redes;
  - Pode estar presente nas camadas de aplicação, transporte e enlace

## Princípios da transmissão confiável de dados:

- **Controle de erros:**
  - Têm por objetivo assegurar a retransmissão de dados não corretamente recebidas (*após a detecção do erro*);
  - O método automático que os sistemas “pedem” a retransmissão dos dados ao emissor é chamado **ARQ – Automatic repeat request**;
- **Controle de congestionamento:**
  - É necessário assegurar que o emissor não sobrecarrega o receptor com quantidade de dados superior à que este consegue suportar;

# Princípios de transferência confiável de dados

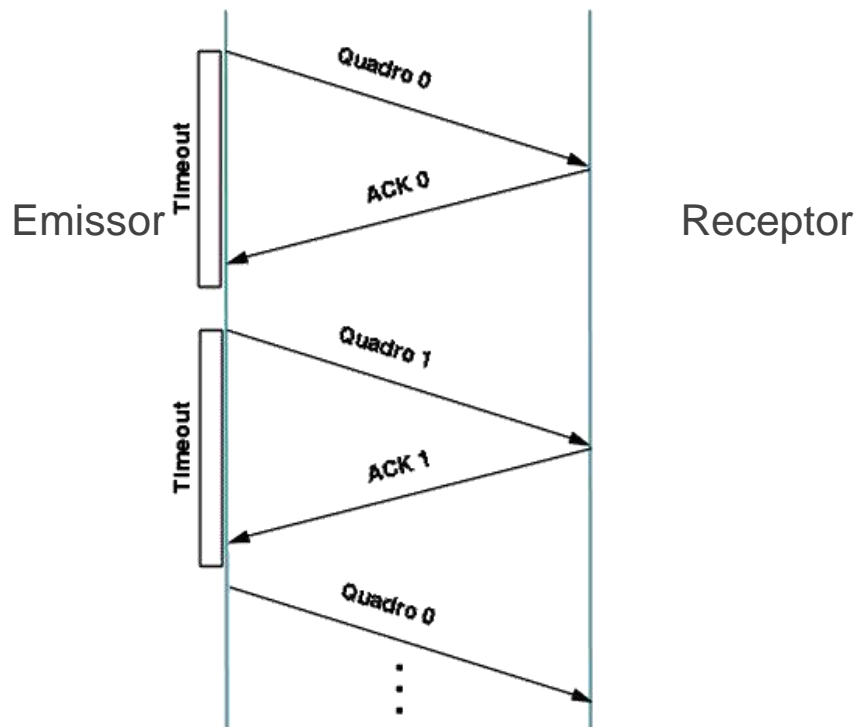
- **Transmissão confiável:** confiabilidade pelo canal



- Estratégia que implementa a entrega confiável: ARQ - Automatic Repeat Request (Solicitação automática de repetição)
  - técnica mais utilizada;
  - exige detecção de erro (CRC, p. ex.);
  - retransmite o quadro que deu problema;
  - combina confirmação (ACK - acknowledgment) e temporização (timeout).
- Existem diferentes algoritmos ARQ:
  - Stop and wait
  - Janela deslizante:
    - Go Back N
    - Selective Repeat

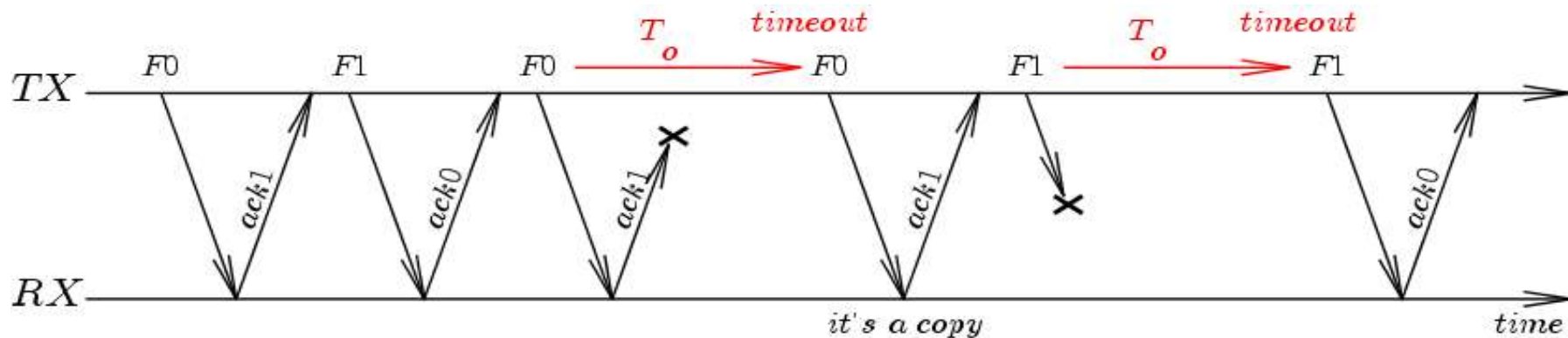
# ARQ “Parar e Esperar” (Stop and Wait)

- Esquema de ARQ mais simples:
  - Depois de transmitir um quadro, o emissor espera uma confirmação antes de transmitir o próximo quadro.
  - Se a confirmação não chegar após um período, o emissor reinicia seu tempo limite e retransmite o quadro original.



# ARQ “Parar e Esperar” (Stop and Wait)

- Problemas do ARQ “Parar e Esperar”:
  - Um pacote pode ser perdido;
  - Um ACK pode ser perdido ou chegar atrasado;
  - Pode ocorrer a situação em que um timeout é disparado antes do ACK chegar ao emissor;

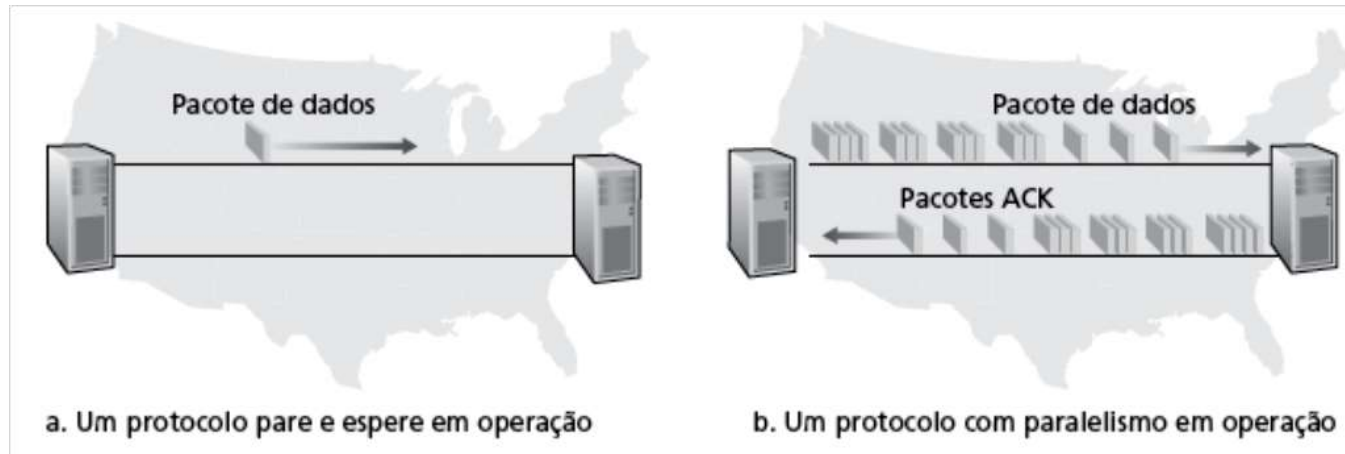


- **Problema: manter a tubulação cheia.**
  - Esse ARQ é limitado, pois permite que o emissor tenha apenas 1 quadro pendente no enlace de cada vez... e isso pode ser muito abaixo da capacidade do enlace.



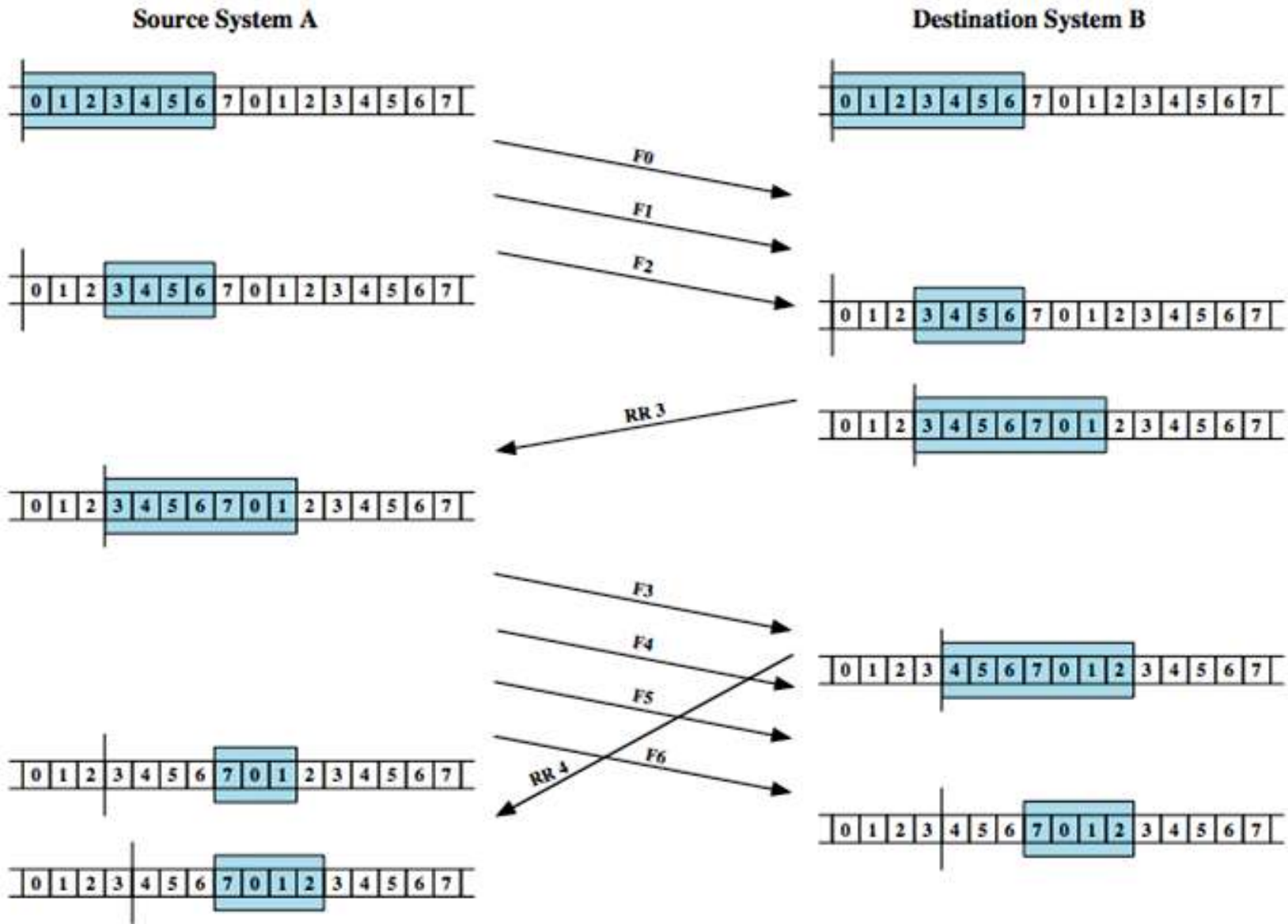
**paralelismo:** remetente permite múltiplos pacotes “no ar”, ainda a serem reconhecidos

- intervalo de números de sequência deve ser aumentado
- buffering no remetente e/ou destinatário

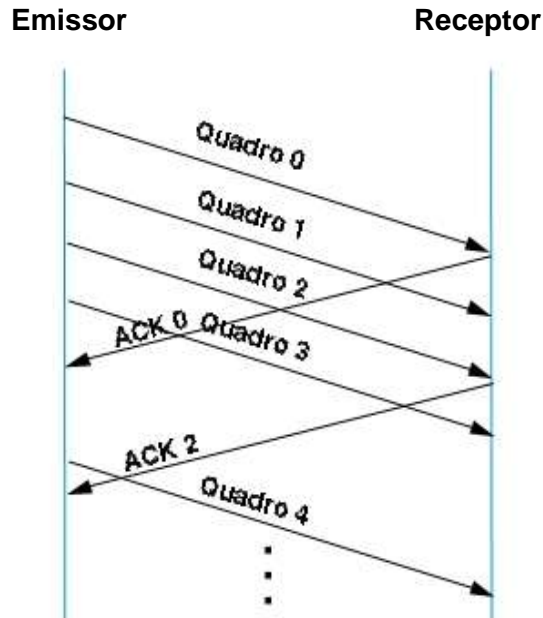


- duas formas genéricas de protocolo com paralelismo: ***Go-Back-N, repetição seletiva***

# Sliding Window Example



## ARQ “Janela Deslizante” (Sliding Window)

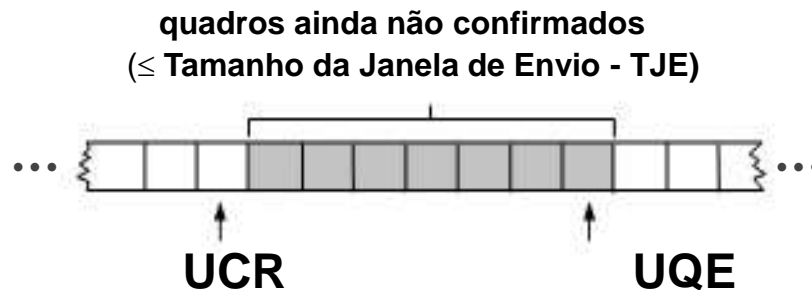


- Stop-and-Wait só permitiria 100% de utilização se RTT fosse zero !
- Solução: manter a tubulação cheia enviando mais quadros antes de receber todos os ACKs
- Receptor tem buffer para armazenar até  $W$  quadros.
- Emissor pode enviar até  $W$  quadros sem ACK ( $W = \text{janela}$ ).
- Cada quadro recebe um número de seqüência.
- Este número tem  $k$  bits (cobre de **0 a  $2^k - 1$** ).
- O ACK pode ser individual ou cumulativo.

# ARQ “Janela Deslizante” (emissor)

- Funcionamento do algoritmo de janela deslizante:
  - Emissor atribui um nº de sequência a cada quadro (NumSeq);
  - Emissor mantém 3 variáveis:
    - Tamanho da janela de envio (TJE) – nº de quadros não confirmados que o emissor pode transmitir;
    - Última confirmação recebida (UCR)
    - Último quadro enviado (UQE)
  - **Emissor** deve manter o seguinte invariável:

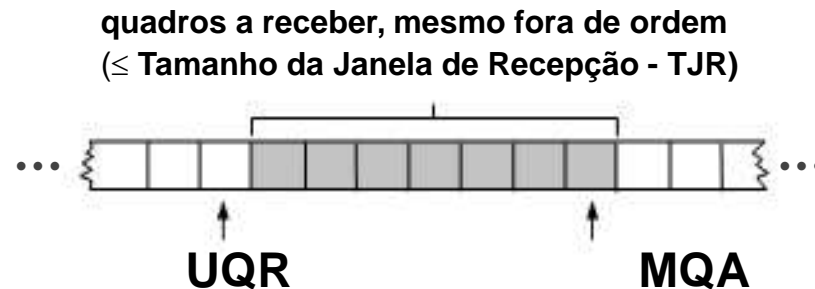
$$\text{UQE} - \text{UCR} \leq \text{TJE}$$



# ARQ “Janela Deslizante” (receptor)

- Funcionamento do algoritmo de janela deslizante:
  - Receptor mantém 3 variáveis:
    - Tamanho da janela de recepção (TJR) – nº de quadros fora de ordem que o receptor deseja aceitar;
    - Maior quadro aceitável (MQA)
    - Último quadro recebido (UQR)
  - **Receptor** também deve manter o seguinte invariável:

$$\text{MQA} - \text{UQR} \leq \text{TJR}$$



# ARQ “Janela Deslizante”

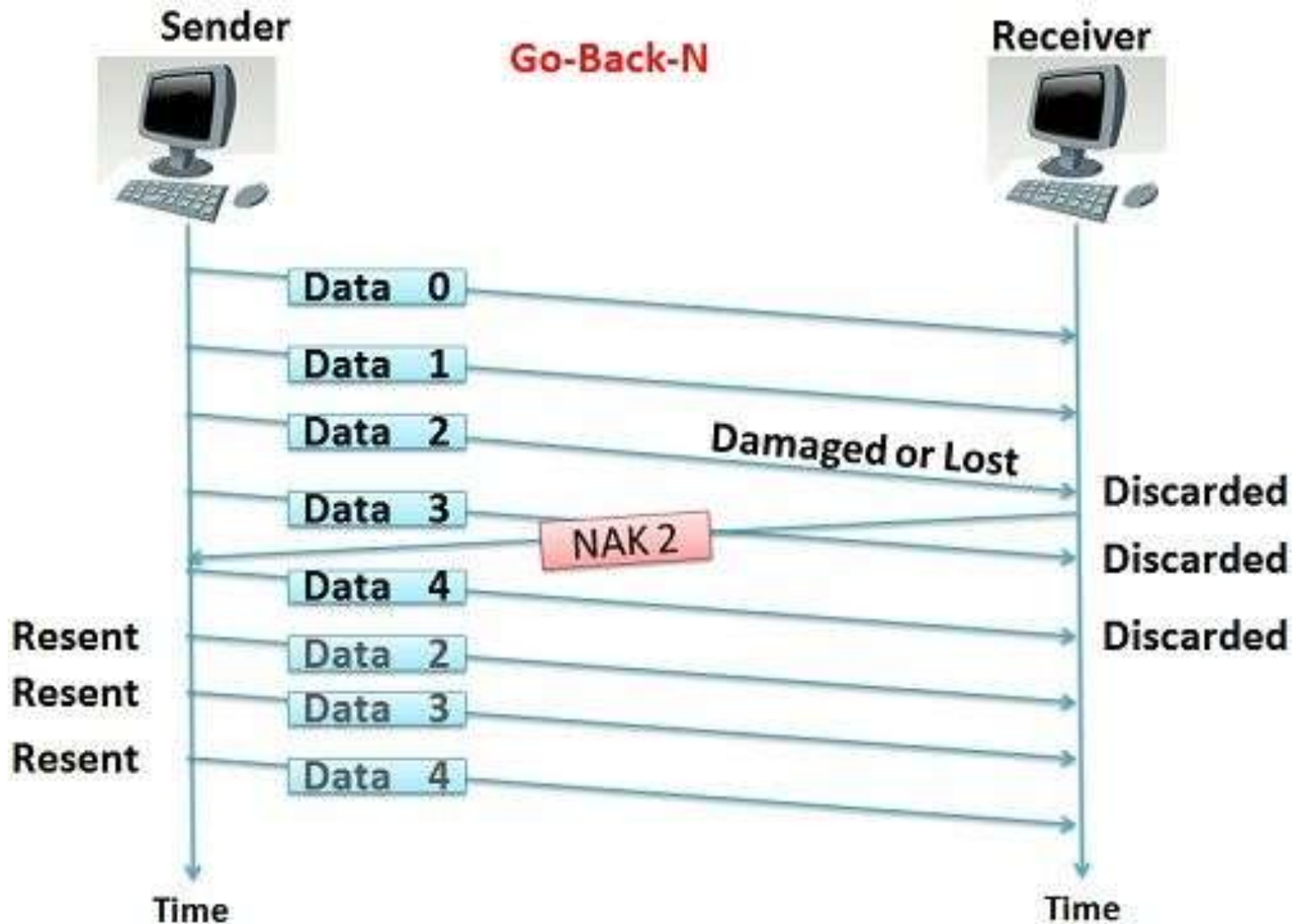
- Quando ocorre um timeout, quantidade de dados em trânsito diminui, pois emissor é incapaz de avançar sua janela, até um quadro x seja confirmado.
  - Quando ocorre perdas de pacotes, esse esquema **não consegue manter o canal cheio**.
  - Quanto + tempo para notar a perda de pacotes, + severo o problema.
- O tam. janela de envio é selecionado de acordo com o nº de quadros que pode-se ter pendentes no enlace.
  - TJE → fácil de calcular (produto retardo x BW)
  - Receptor pode definir TJR que quiser:
    - TJR = 1** : receptor não coloca em buffer nenhum quadro fora de ordem;
    - TJR = TJE** : receptor pode manter em buffer qualquer um dos quadros que emissor transmitir.
    - TJR > TJE** : não faz sentido. É impossível que mais de TJE cheguem fora de ordem.

# “Sliding-Windows” - Go-Back-N

## « *Go-Back-N* »

- Janela tem um tamanho máximo de N pacotes ainda não “ACKados”;
- Receptor pode enviar uma confirmação por vários pacotes;
- ACK(M) : confirma a recepção dos pacotes com um nº de sequência inferior ou igual a M;
- Emissor usa um timeout por os pacotes não “ACKados”;
- **Timeout(M)** : permite a retransmissão de todos os pacotes que são superior a M e na “sending-windows”;
- Problemas :
  - O receptor pode enviar “ACK” varias vez por um mesmo pacote

# Go Back N



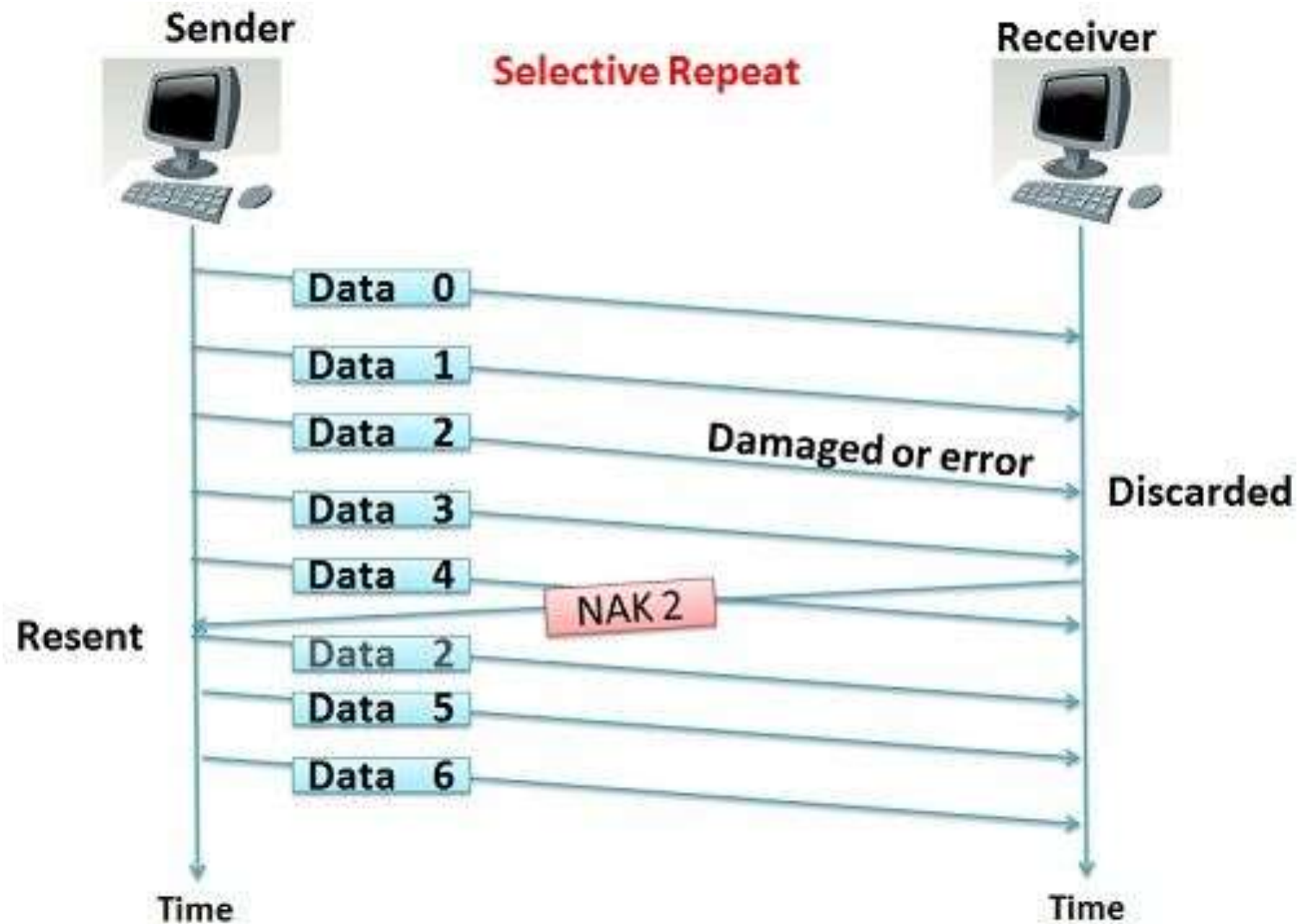


# “Sliding-Windows” - Repetição seletiva

## « Selective Repeat »

- Destinatário reconhece individualmente todos os pacotes recebidos de modo correto
  - mantém pcts em buffer, se for preciso, para eventual remessa em ordem para a camada superior
- Remetente só reenvia pcts para os quais o ACK não foi recebido
  - temporizador no remetente para cada pct sem ACK
- Janela do remetente
  - N # seq. consecutivos
  - novamente limita #s seq. de pcts enviados, sem ACK

# Repetição seletiva



## Go-back-N: visão geral

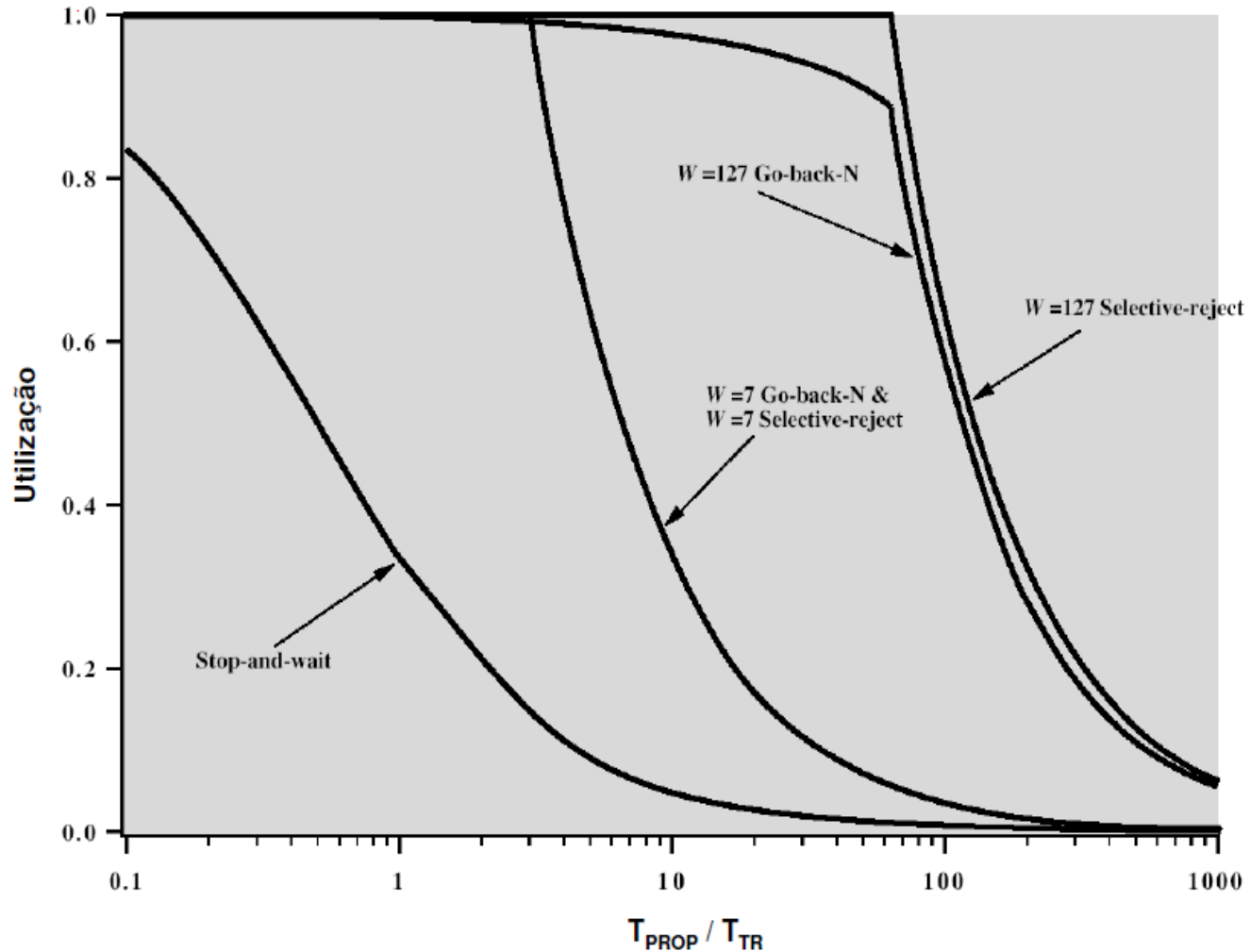
- **remetente:** até N pacotes não reconhecidos na pipeline
- **destinatário:** só envia ACKs cumulativos
  - não envia ACK se houver uma lacuna
- **remetente:** tem temporizador para pct sem ACK mais antigo
  - se o temporizador expirar: retransmite todos os pacotes sem ACK
- Na prática, o Go-back-N é mais utilizado devido a:
  - simplicidade de implementação;
  - Frequentemente, erros ocorrem em rajadas.

## Repetição seletiva: visão geral

- **remetente:** até pacotes não reconhecidos na pipeline
- **destinatário:** reconhece (ACK) pacotes individuais
- **remetente:** mantém temporizador para cada pct sem ACK
  - se o temporizador expirar: retransmite apenas o pacote sem ACK

# Desempenho em função de $T_{PROP}/T_{TR}$

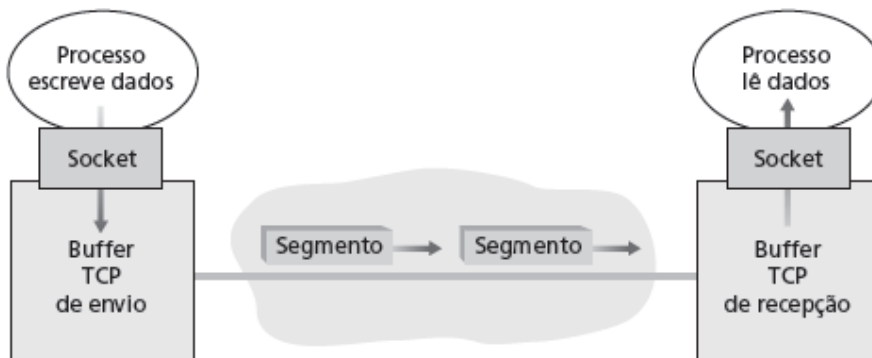
(com probabilidade de 1 quadro errado em cada 1000 quadros)



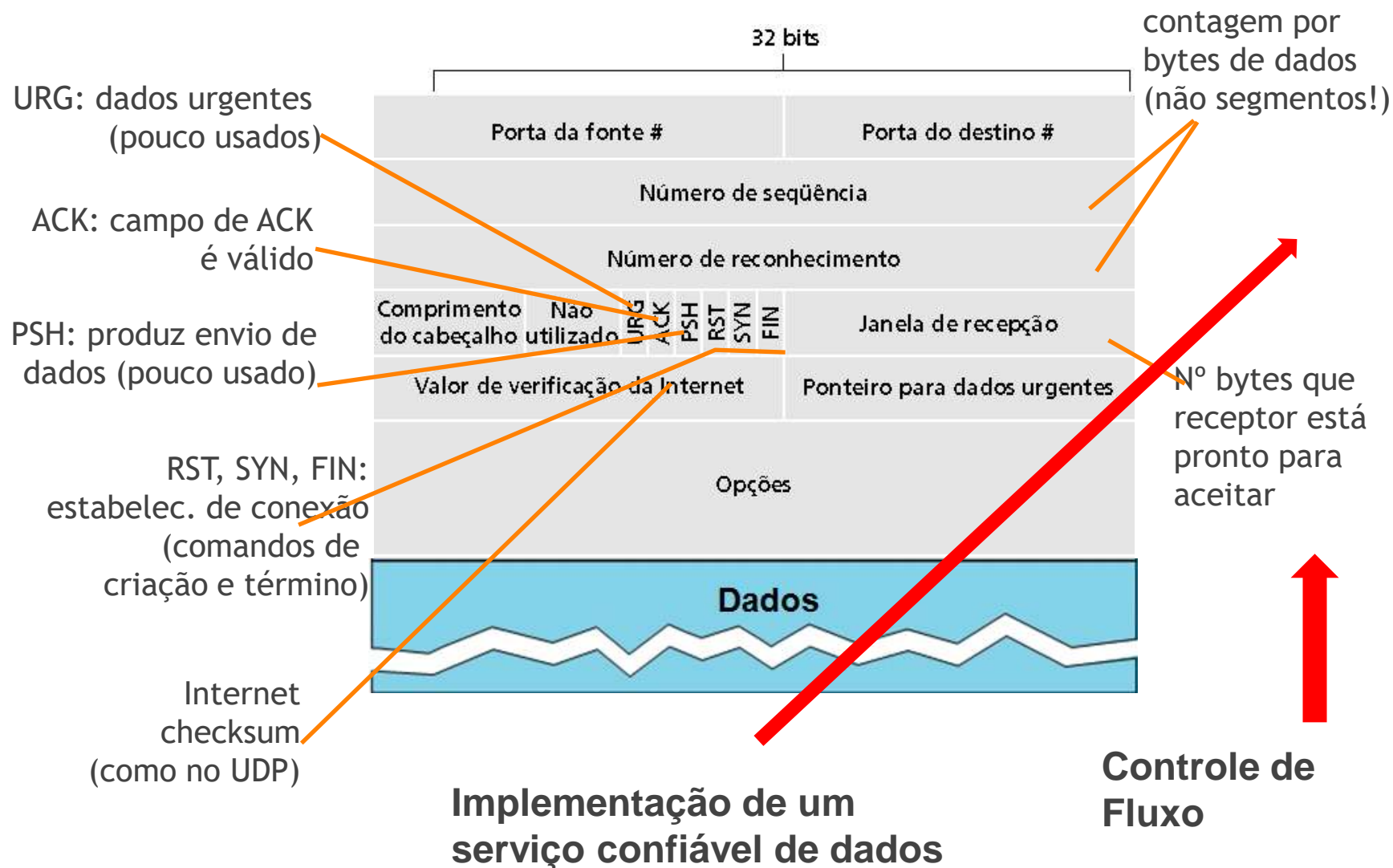
- **5.1 Serviços da camada de transporte**
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- **5.3 Transporte não orientado para conexão: UDP**
- **5.4 Princípios da transferência confiável de dados**
- **5.5 Transporte orientado para conexão: TCP**
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
- **5.6 Princípios de controle de congestionamento**
- **5.7 Controle de congestionamento no TCP**

# TCP: Visão geral

- **A conexão é ponto a ponto:**
  - um remetente, um destinatário
- **cadeia de bytes confiável, em ordem:**
  - sem “limites de mensagem”
- **paralelismo:**
  - congestionamento TCP e controle de fluxo definem tamanho da janela
- **buffers de envio & recepção**
- **dados full duplex:**
  - dados bidirecionais fluem na mesma conexão
  - MSS: tamanho máximo do segmento
- **orientado a conexão:**
  - apresentação (troca de msgs de controle) inicia estado do remetente e destinatário antes da troca de dados
- **fluxo controlado:**
  - remetente não sobrecarrega destinatário



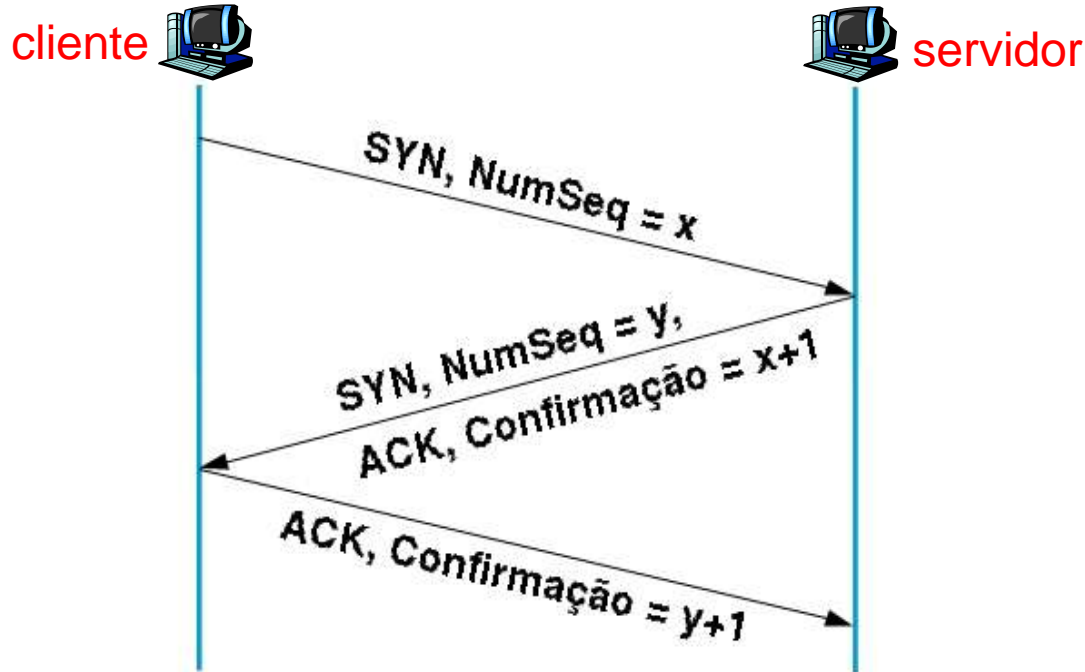
# Estrutura do segmento TCP



# Conexão TCP - estabelecimento

Participante ativo

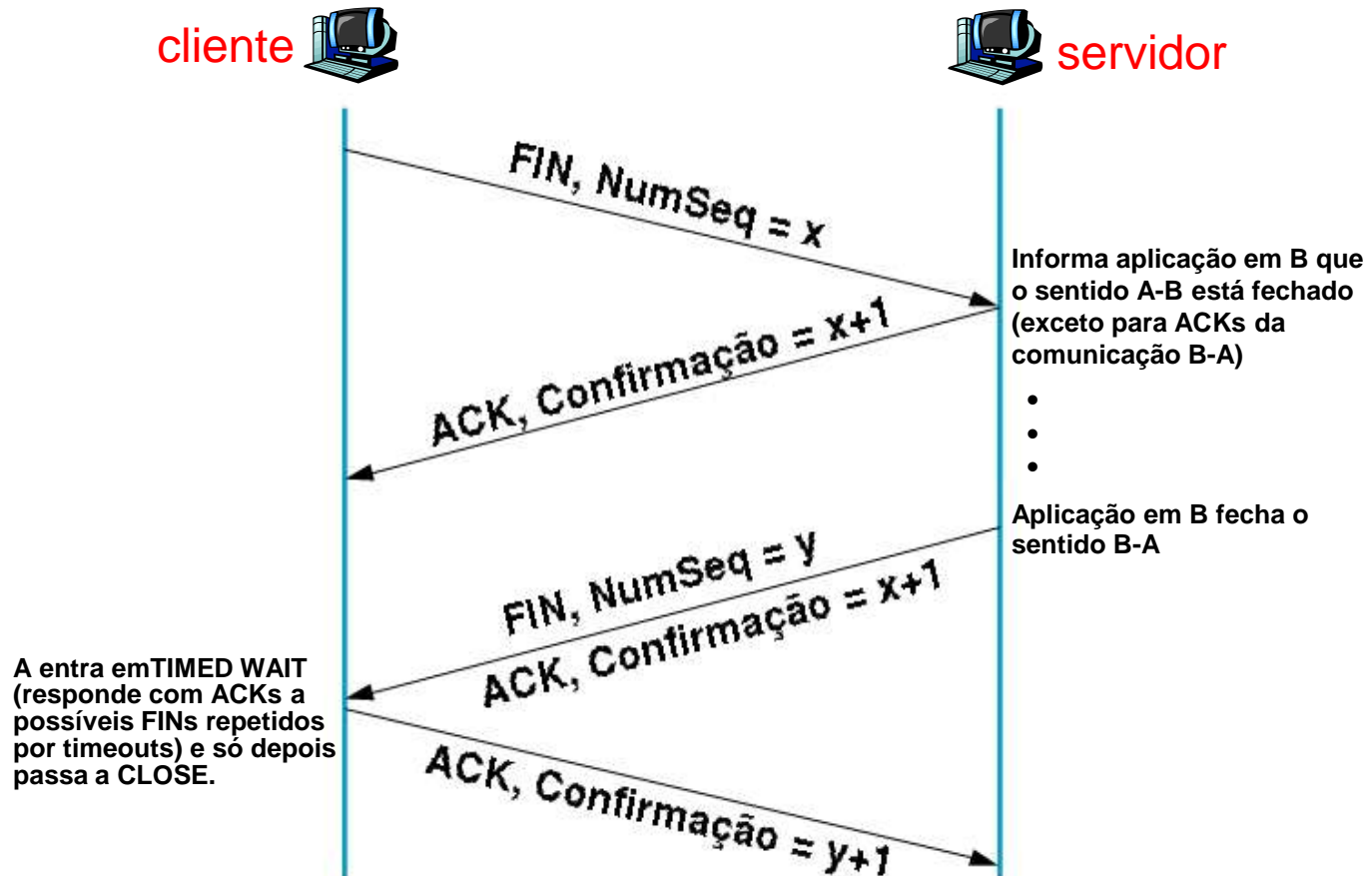
Participante passivo



**Handshake de Três Vias (Three-Way Handshake)  
para estabelecimento da conexão TCP**



# Conexão TCP - término



Handshake de Três Vias para término da conexão TCP

# Exibindo conexões ativas / portas

- Exibindo conexões ativas e as portas utilizadas .

```

C:\Windows\system32\cmd.exe

C:\Users\Marlon>netstat -n

Conexões ativas

Proto Endereço local      Endereço externo    Estado
TCP    177.30.211.137:53055  74.125.45.94:80      ESTABLISHED
TCP    177.30.211.137:53056  74.125.45.94:80      ESTABLISHED
TCP    177.30.211.137:53058  67.195.186.237:80    TIME_WAIT
TCP    177.30.211.137:53059  74.125.45.94:80      TIME_WAIT
TCP    177.30.211.137:53065  74.125.45.94:80      TIME_WAIT
TCP    177.30.211.137:53079  200.192.176.110:80   TIME_WAIT
TCP    177.30.211.137:53081  74.125.45.94:80      TIME_WAIT
TCP    177.30.211.137:53087  200.154.56.237:80    TIME_WAIT
TCP    177.30.211.137:53101  173.194.42.0:80      TIME_WAIT
TCP    177.30.211.137:53102  200.154.56.14:80     FIN_WAIT_1
TCP    177.30.211.137:53108  200.123.198.178:80   TIME_WAIT
TCP    177.30.211.137:53110  173.194.42.27:80     TIME_WAIT
TCP    177.30.211.137:53114  173.194.42.27:80     TIME_WAIT
TCP    177.30.211.137:53116  173.194.42.25:80     TIME_WAIT
TCP    177.30.211.137:53119  200.192.176.87:80    TIME_WAIT
TCP    177.30.211.137:53122  173.194.42.15:80     TIME_WAIT
TCP    177.30.211.137:53124  209.191.69.109:80    TIME_WAIT
TCP    177.30.211.137:53125  209.191.69.109:80    TIME_WAIT
TCP    177.30.211.137:53134  69.171.247.69:80     TIME_WAIT
TCP    177.30.211.137:53148  74.125.45.99:80      TIME_WAIT
TCP    177.30.211.137:53151  74.125.45.104:80     TIME_WAIT
TCP    177.30.211.137:53158  74.125.45.104:80     TIME_WAIT
TCP    177.30.211.137:53160  67.195.186.237:80    ESTABLISHED
TCP    177.30.211.137:53161  67.195.186.237:80    ESTABLISHED
TCP    177.30.211.137:53162  209.191.69.109:80    ESTABLISHED
TCP    177.30.211.137:53163  209.191.69.109:80    ESTABLISHED

C:\Users\Marlon>_
  
```

# #s sequência e ACKs do TCP

## #s de sequência:

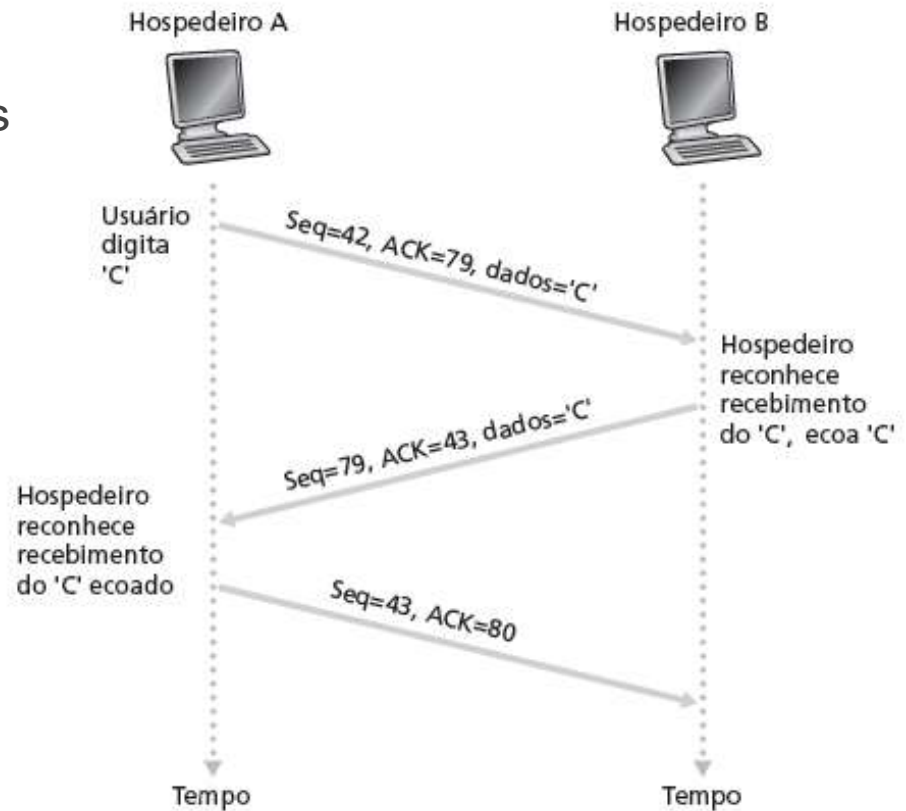
- “número” na cadeia de bytes do 1º byte nos dados do segmento

## ACKs:

- # seq do próximo byte esperado do outro lado
- ACK cumulativo

## P: como o destinatário trata segmentos fora de ordem

- R: TCP não diz – a critério do implementador



cenário telnet simples

## **P: Como definir o valor de *timeout* do TCP?**

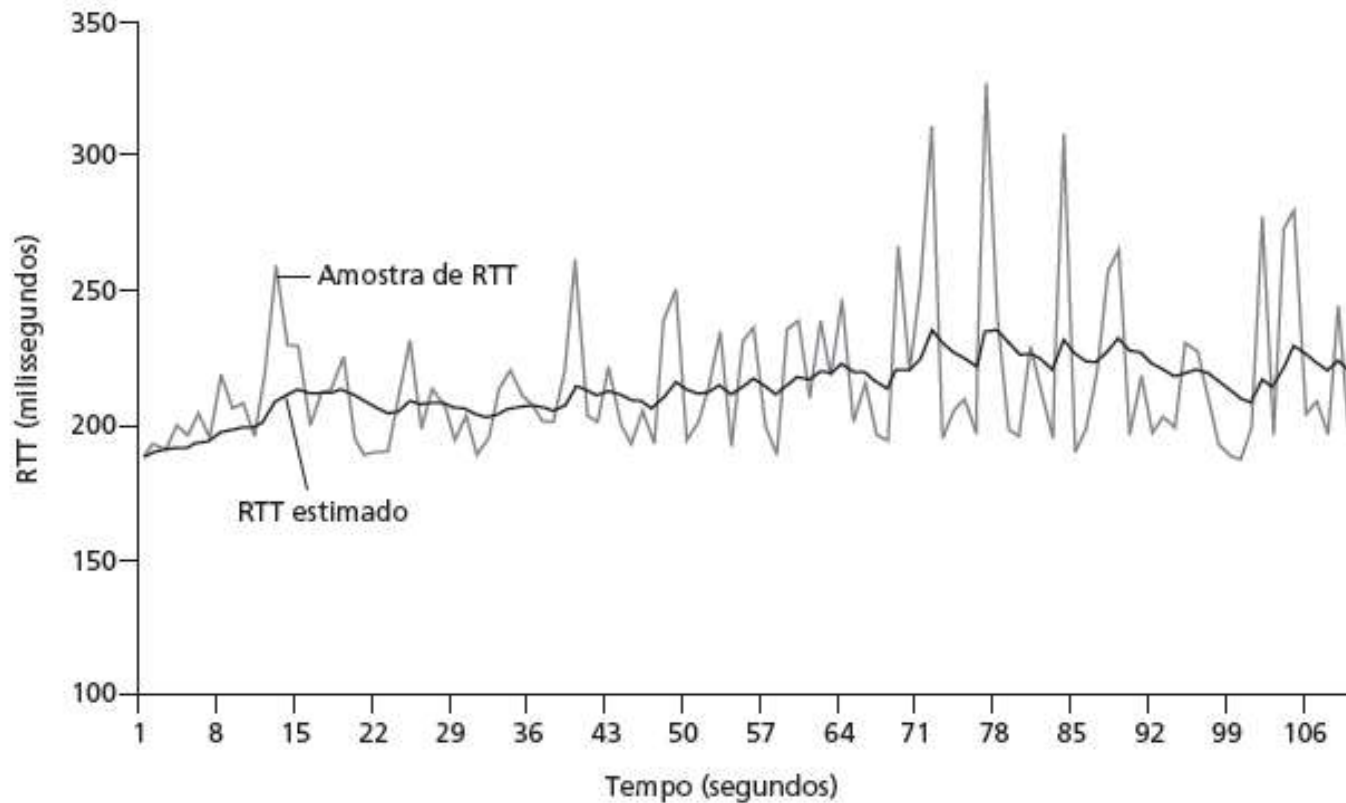
- maior que RTT -> **porém, RTT varia...**
  - **muito curto:** *timeout* prematuro
    - retransmissões desnecessárias
  - **muito longo:** baixa reação a perda de segmento

## **P: Como estimar o RTT?**

- **SampleRTT:** tempo medido da transmissão do segmento até receber o ACK
  - ignora retransmissões
- **SampleRTT variará; queremos RTT estimado “mais estável”**
  - média de várias medições recentes, não apenas **SampleRTT** atual

# Amostras de RTTs estimados:

média de várias medições recentes, não apenas `SampleRTT` atual



- **5.1 Serviços da camada de transporte**
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- **5.3 Transporte não orientado para conexão: UDP**
- **5.4 Princípios da transferência confiável de dados**
- **5.5 Transporte orientado para conexão: TCP**
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
- **5.6 Princípios de controle de congestionamento**
- **5.7 Controle de congestionamento no TCP**

## Janela deslizando pode ser usado para 3 papéis diferentes:

- Entregar quadros de modo confiável em um enlace não confiável
  - função básica do algoritmo;
- Preservar a ordem em que os quadros são transmitidos
  - Fácil de se fazer no receptor, visto que cada quadro possui um nº de sequência;
  - Receptor certifica que não passará um quadro para o próxima camada de nível superior até que tenha recebido todos quadros com nº seq menor;
    - Receptor mantém em buffer os quadros fora de ordem.
- Admitir o controle de fluxo

- Mesmo respeitando a taxa de transferência acordada, emissor pode enviar quadros um após o outro, de uma forma que um receptor lento para processá-los acaba perdendo vários quadros.
- **SOLUÇÃO:** Controle de Fluxo
  - Mecanismo evita que o emissor ultrapasse a capacidade do receptor;
    - **compatibiliza as taxas de envio e de leitura dos hosts;**
  - Um mecanismo de realimentação, usado pelo receptor, capaz de cadenciar o transmissor, não estourando o buffer do destinatário;
- Permite vários frames numerados em trânsito;
  - Receptor com buffer de tamanho W:
  - Transmissor pode enviar até W frames sem ACK;
  - Número de seqüência é limitado
- Podem ser utilizados piggyback ACKs.



- **5.1 Serviços da camada de transporte**
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- **5.3 Transporte não orientado para conexão: UDP**
- **5.4 Princípios da transferência confiável de dados**
- **5.5 Transporte orientado para conexão: TCP**
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
- **5.6 Princípios de controle de congestionamento**
- **5.7 Controle de congestionamento no TCP**

# Princípios de controle de congestionamento

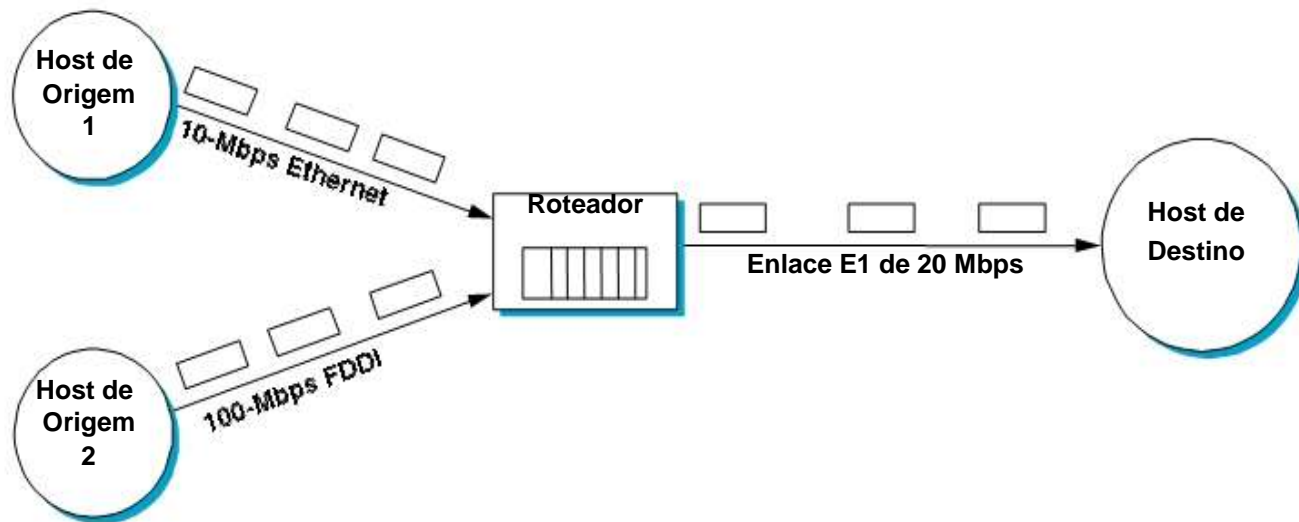
## Congestionamento:

- informalmente: “muitas fontes enviando muitos dados muito rápido para a **rede** tratar”
- diferente de controle de fluxo!
- manifestações:
  - pacotes perdidos (estouro de buffer nos roteadores)
  - longos atrasos (enfileiramento nos buffers do roteador)
- um dos maiores problemas da rede!

# Congestionamento

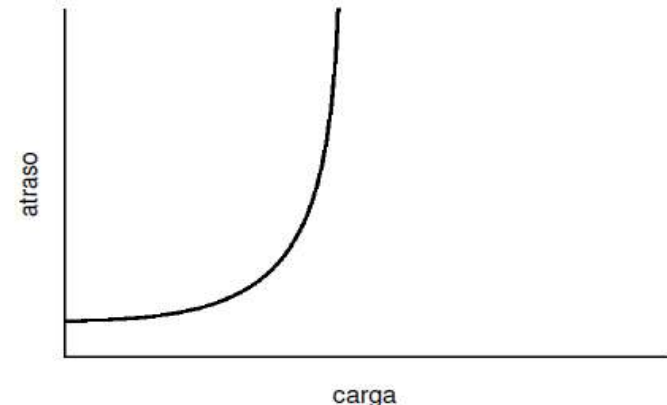
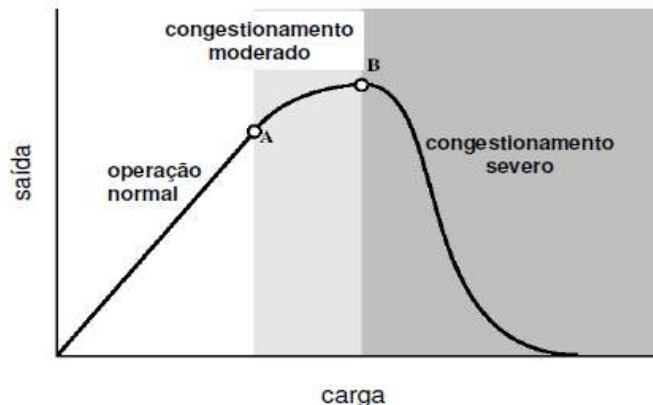
## Congestionamento:

- Ocorre quando o número de pacotes que atravessa uma rede se aproxima da capacidade máxima dessa rede.
- Ocorre então uma perda significativa de desempenho, tendendo ao colapso total de funcionamento da rede.



# Alocação de Recursos da Rede

- **Cenário:** diversas aplicações disputam simultaneamente os recursos da rede (para que seus pacotes cheguem aos destinos).
- **Recursos da rede (principais):** largura de banda dos enlaces e espaço de buffer em roteadores e switches.
- **Preocupação 1- congestionamento da rede:**
  - a rede não consegue atender à demanda total apresentada, devido ao esgotamento de seus recursos internos.



- **Preocupação 2 - Qualidade de Serviço (QoS) da rede:**
  - quais níveis de atendimento a requisitos específicos das aplicações a rede pode atender.

\* 2 técnicas amplas para controle de congestionamento:

## **controle de congestionam. fim a fim:**

- nenhum feedback explícito da rede
- congestionamento deduzido da perda e atraso observados do sistema final
- técnica tomada pelo TCP

## **controle de congestionam. assistido pela rede:**

- roteadores oferecem feedback aos sistemas finais
  - único bit indicando congestionamento (SNA, DECbit, TCP/IP ECN, ATM)
  - taxa explícita que o remetente deve enviar no enlace de saída

- **Centrado no roteador versus centrado no host:**
  - **roteador:** controle dos recursos (buffers e linhas de saída; política de descarte)
  - **host:** observação da rede; controle da injeção de tráfego
  - papéis complementares
- **Baseado em reserva versus baseado em feedback:**
  - **reserva:** pré-aloca recursos (buffers; parcela do BW)
  - **feedback:** feedback (explícito ou implícito) regula dinamicamente o volume de envio dos hosts
  - obs: o feedback pode ser apenas binário ou mais elaborado
- **Baseado em janela versus baseado em taxa:**
  - **janela:** um feedback de créditos regula dinamicamente o volume de envio dos hosts
  - **taxa:** um feedback de taxa (em bps) regula dinamicamente o volume de envio dos hosts

- **5.1 Serviços da camada de transporte**
- **5.2 Multiplexação e demultiplexação**
  - Sockets
- **5.3 Transporte não orientado para conexão: UDP**
- **5.4 Princípios da transferência confiável de dados**
- **5.5 Transporte orientado para conexão: TCP**
  - estrutura de segmento
  - transferência confiável de dados
  - controle de fluxo
  - gerenciamento da conexão
- **5.6 Princípios de controle de congestionamento**
- **5.7 Controle de congestionamento no TCP**

# Aumento aditivo/diminuição multiplicativa



- **Additive Increase/Multiplicative Decrease (AIMD):**
  - objetivo: ajustar-se às mudanças na capacidade disponível da rede.
- **Nova variável de estado por conexão:**  
**CongestionWindow**  
limita quantos dados a origem tem em trânsito:  
$$\text{MaxWin} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$$
$$\text{EffWin} = \text{MaxWin} - (\text{LastByteSent} - \text{LastByteAcked})$$
- **Idéia:**
  - aumentar **CongestionWindow** quando o congestion. diminuir;
  - diminuir **CongestionWindow** quando o congestion. aumentar;
  - usar timeout como sinal de congestionamento:
    - um timeout sinaliza uma perda de pacote;
    - pacotes raramente são perdidos por erro de transmissão;
    - logo, pacote perdido sinaliza congestionamento.



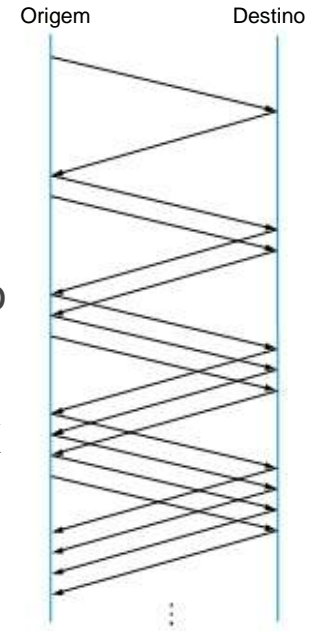
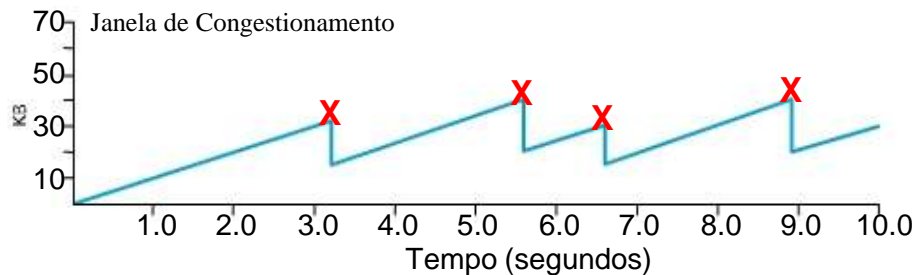
# AIMD (cont.)

- **Algoritmo AIMD:**

- incremente **CongestionWindow** em um pacote por RTT (aumento linear)
- divida **CongestionWindow** por dois sempre que houver um timeout (diminuição multiplicativa)

- **Na prática:** incrementa “um pouco” para cada ACK  
 $\text{Increment} = (\text{MSS} * \text{MSS}) / \text{CongestionWindow}$   
 $\text{CongestionWindow} += \text{Increment}$

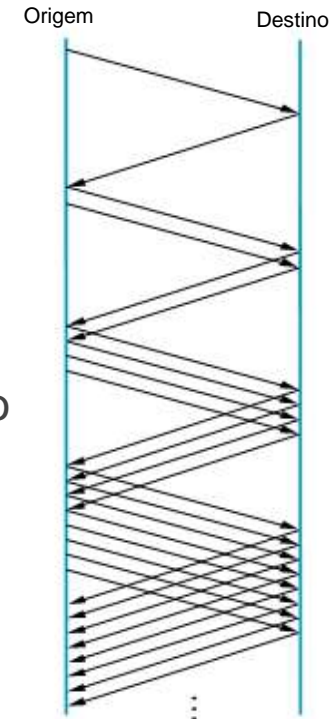
- **Rastreamento:**



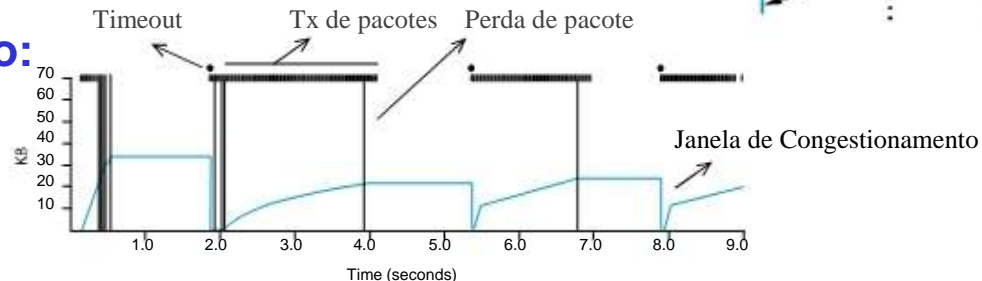
**X** perda e diminuição de taxa

# Partida lenta (slow start)

- **Objetivo:** determinar, no início, a capacidade disponível da rede.
- **Idéia:**
  - comece com **CongestionWindow** = 1 pacote;
  - duplique **CongestionWindow** a cada RTT (incremente em 1 pacote para cada ACK).
- O crescimento é exponencial, porém é melhor do que começar injetando um “pico” na rede.
- **Usada...**
  - quando se inicia a conexão;
  - quando a conexão termina esperando por timeout.

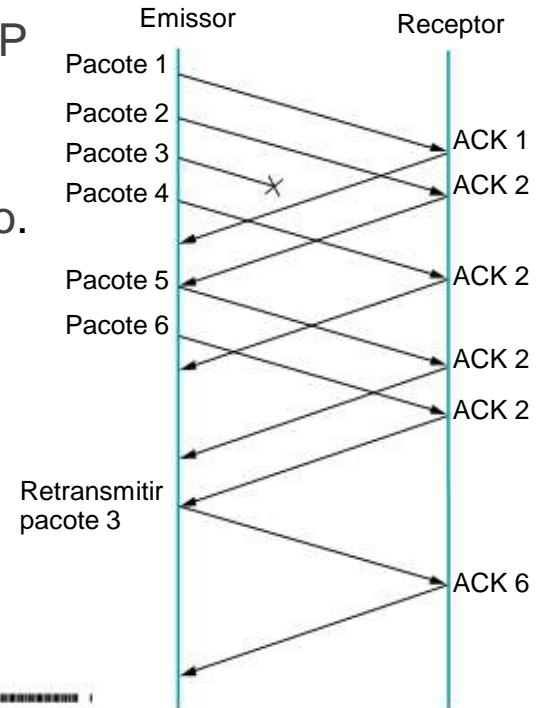
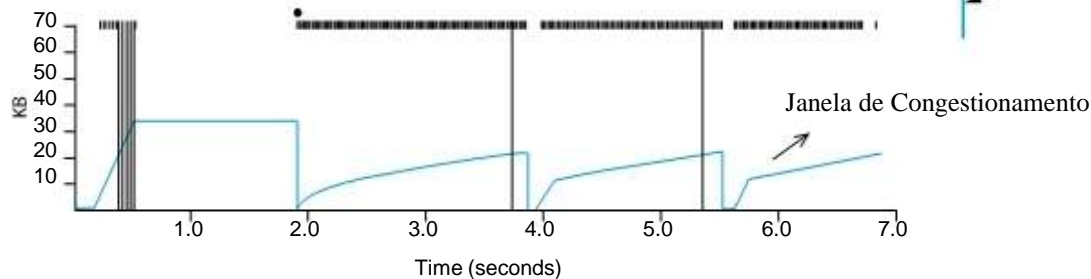


- **Rastreamento:**



# Retransmissão rápida e recuperação rápida

- **Problema:** timeouts espaçados no TCP levam a períodos de ociosidade.
- **Retransmissão rápida:** usa ACKs duplicados para disparar retransmissão.
- **Recuperação rápida:**
  - pula a fase de partida lenta;
  - vai diretamente para metade do último **CongestionWindow** (**ssthresh**) bem sucedido.
- **Rastreamento:**



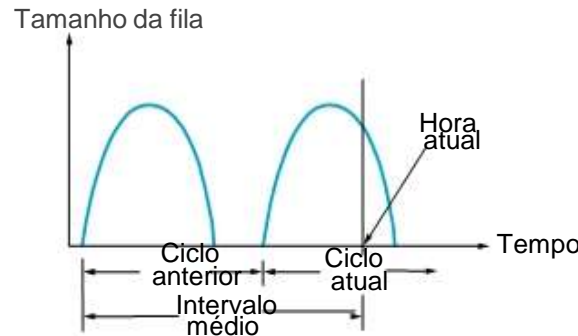
# Prevenindo o Congestionamento

# Prevenção de congestionamento

- **Estratégia do TCP:**
  - controle o congestionamento quando ele acontecer;
  - aumente a carga repetidamente em um esforço para encontrar o ponto em que o congestionamento ocorre, e depois recue.
- **Estratégia alternativa:**
  - preveja quando o congestionamento está para acontecer;
  - reduza a velocidade antes que os pacotes comecem a ser descartados;
  - chame isso de prevenção de congestionamento, em vez de controle de congestionamento.
- **Duas possibilidades:**
  - centrada no roteador: DECbit e RED;
  - centrada no host: TCP Vegas.

# DECbit

- Inclua um bit de congestionamento em cada cabeçalho de pacote.
- Roteador: monitora o tamanho médio da fila durante último ciclo ocupado+ocioso e marca o bit de congestionamento se o tamanho médio da fila  $> 1$ .



- Destino ecoa o bit de volta para a origem, que registra quantos pacotes chegaram com o bit marcado:
  - se menos de 50% do tamanho da última janela  
aumenta **CongestionWindow** em 1 pacote;
  - se 50% ou mais da última janela com bit marcado  
diminua **CongestionWindow** em 0,875 vezes.

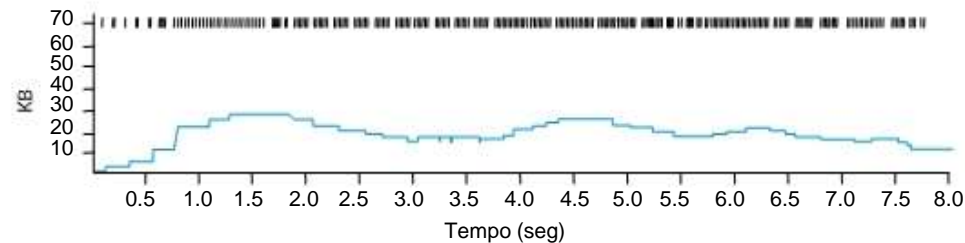
# Random Early Detection (RED)

- **Random Early Detection - RED:** descarte aleatório antecipado:
  - em vez de esperar que a fila se encha, descarte cada pacote que chega com uma probabilidade de descarte, sempre que o tamanho da fila ultrapassar determinado nível de descarte.
- **A notificação é implícita:**
  - apenas descarta o pacote (TCP esgotará tempo limite);
  - poderia se tornar explícita marcando o pacote.

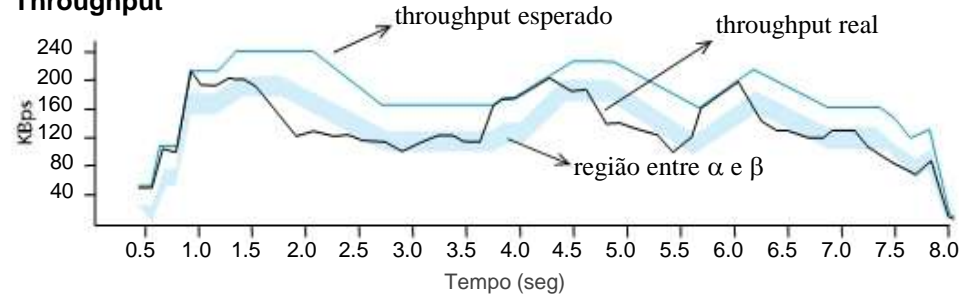
# TCP Vegas

- **Idéia:** origem observa algum sinal de que a fila do roteador está aumentando e que o congestionamento ocorrerá, por exemplo:
  - RTT cresce;
  - taxa de envio nivela.

**Janela de Congestionamento**



**Throughput**





# TCP - Limitações

- ***Nem tudo são flores no TCP***

## **TCP é mais lento**

- Perde-se tempo para estabelecer conexão
- Quando utilizado com segurança (HTTPS+TLS), gastam-se 3 RTTs antes de enviar dados.
- A Google desenvolveu o QUIC



## **Oferece suporte apenas à comunicação ponto a ponto**

- Multicast IP somente com UDP
- Aplicação tem que se virar...