

Nível Transporte

Objectivos:

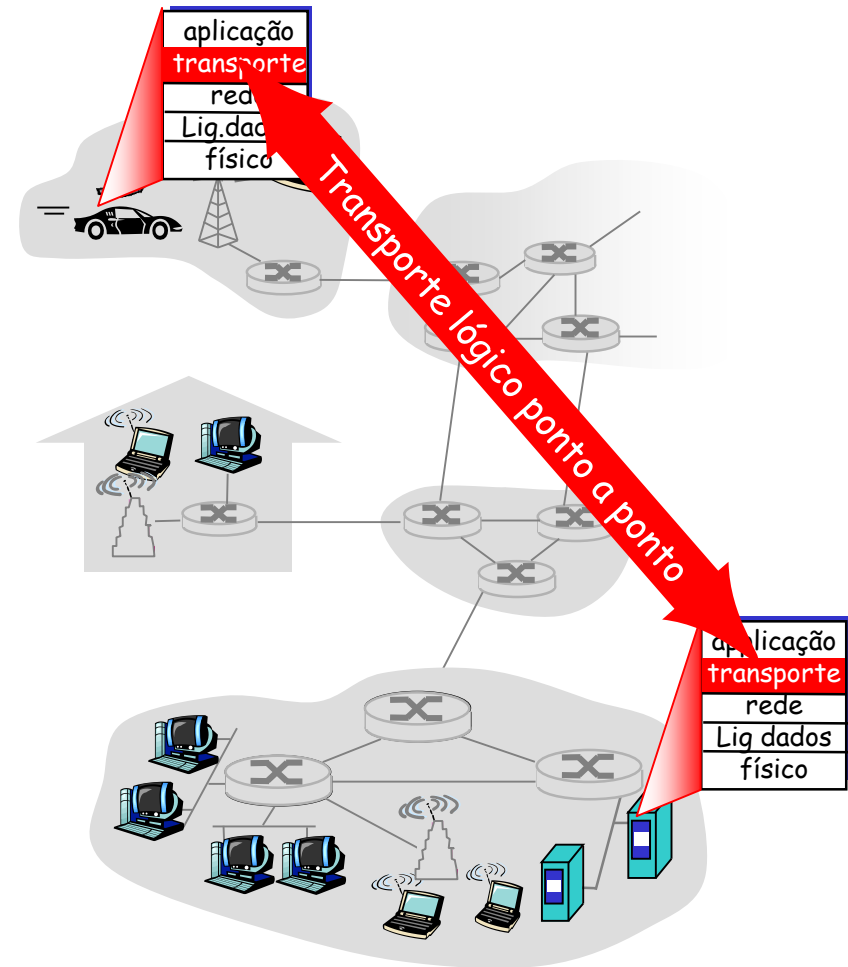
- ❑ Entender os princípios subjacentes ao serviço do Nível Transporte:
 - multiplexagem/desmultiplexagem
 - Transferência de dados fiável
 - Controlo de fluxo
 - Controlo de congestão
- ❑ Aprender os protocolos do Nível Transporte na Internet:
 - UDP: transporte sem ligação
 - TCP: transporte com ligação
 - Controlo de congestão em TCP

Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

Serviços de transporte e protocolos

- ❑ Fornece *comunicação lógica* entre processos de aplicação a funcionar em Sistemas Terminais diferentes
- ❑ Os protocolos de transporte são executados nos sistemas terminais
 - Lado emissor: divide a mensagem da aplicação em *segmentos*, que passa à camada de rede
 - Lado receptor: junta os segmentos em mensagens, que passa à camada de aplicação
- ❑ Mais que um protocolo de transporte disponível para as aplicações
 - Internet: TCP e UDP



Nível de transporte vs rede

- ❑ *Nível de rede:*

comunicação lógica entre Sistema terminais

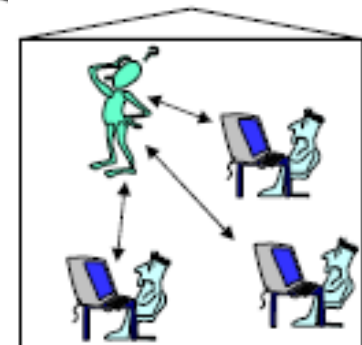
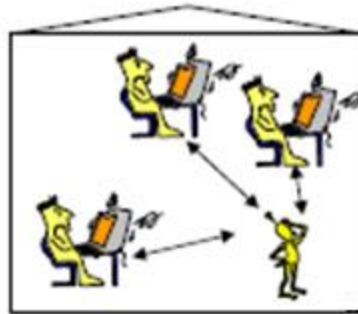
- ❑ *Nível Transporte:*

comunicação lógica entre processos

- Baseia-se nos serviços da camada de rede, melhorando a sua fiabilidade

Serviços de transporte e protocolos

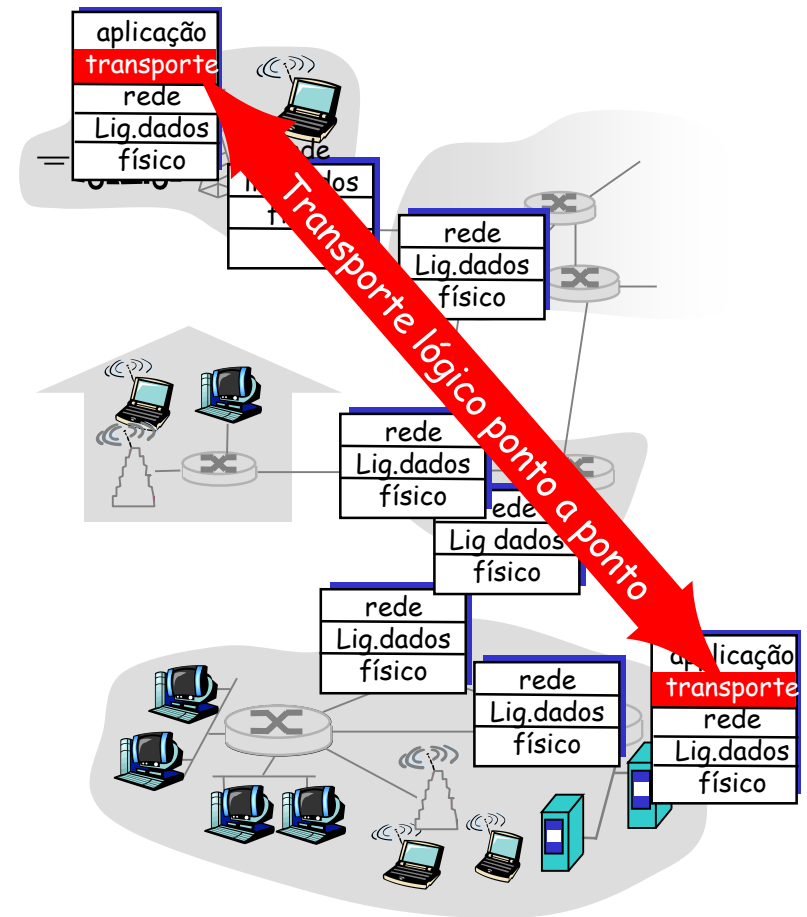
- ❑ Exemplo: serviço postal entre familiares de Lisboa e Porto
- ❑ Sistemas terminais = casas
- ❑ Processos = familiares
- ❑ Protocolo de transporte = carteiros de recolha e entrega
- ❑ Protocolo de rede = serviço postal



- ❑ Sistemas terminais?
- ❑ Processos?
- ❑ Mensagens de aplicação?
- ❑ Protocolos de transporte?
- ❑ Protocolos de rede?

Protocolos de nível de transporte na Internet

- ❑ Entrega fiável, ordenada unicast (TCP)
 - Controlo de congestão
 - Controlo de fluxo
 - Estabelecimento de ligação
- ❑ Entrega não fiável (melhor esforço, best effort) não ordenada, unicast ou multicast: UDP
- ❑ Serviços não disponíveis:
 - Tempo real
 - Garantias de largura de banda
 - Multicast fiável



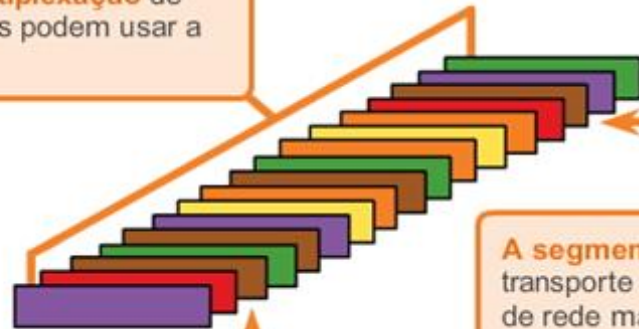
Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

Multiplexagem/demultiplexagem



A segmentação permite a **multiplexação** de conversação - várias aplicações podem usar a rede ao mesmo tempo.



A **segmentação** facilita o transporte de dados de camadas de rede mais baixas.

A **verificação** de erros pode ser executada nos dados no segmento para verificar se o segmento foi alterado durante a transmissão.

Multiplexagem/desmultiplexagem

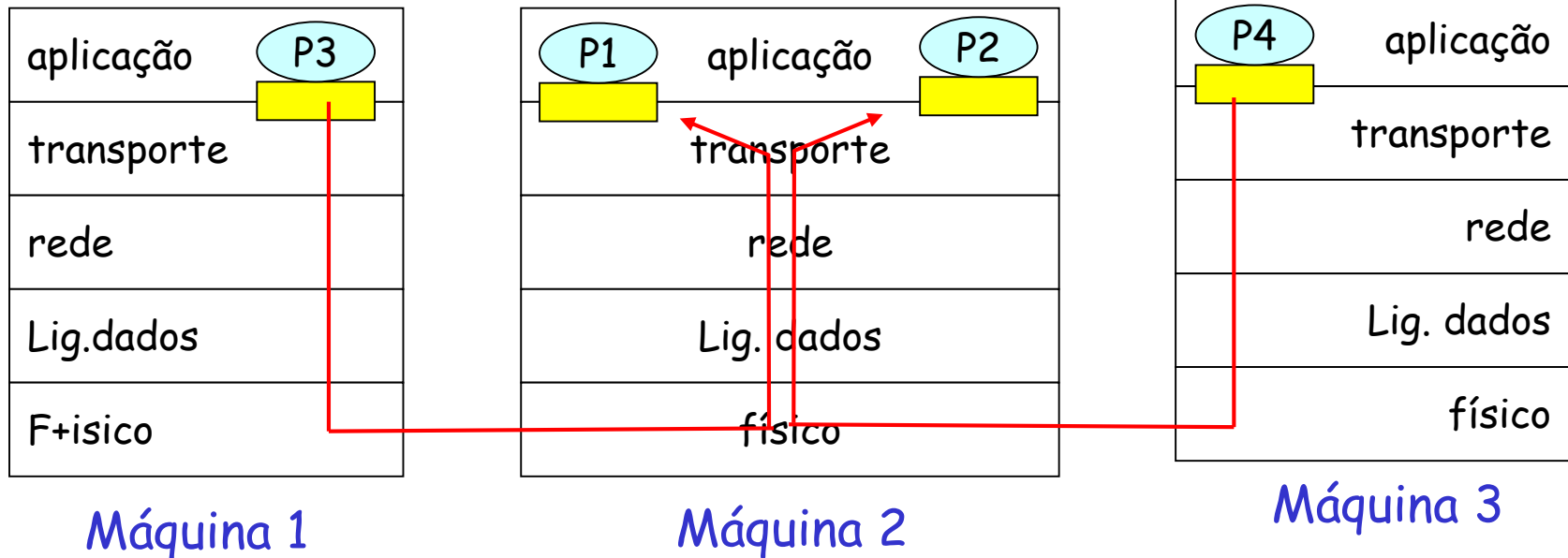
Desmultiplexagem na recepção:

Usa a informação do cabeçalho para entregar os segmentos recebidos ao socket correcto

 = socket  = processo

Multiplexagem no envio:

Recolhe dados de diferentes sockets, delimitar dados com cabeçalho (mais tarde usado para desmultiplexar)

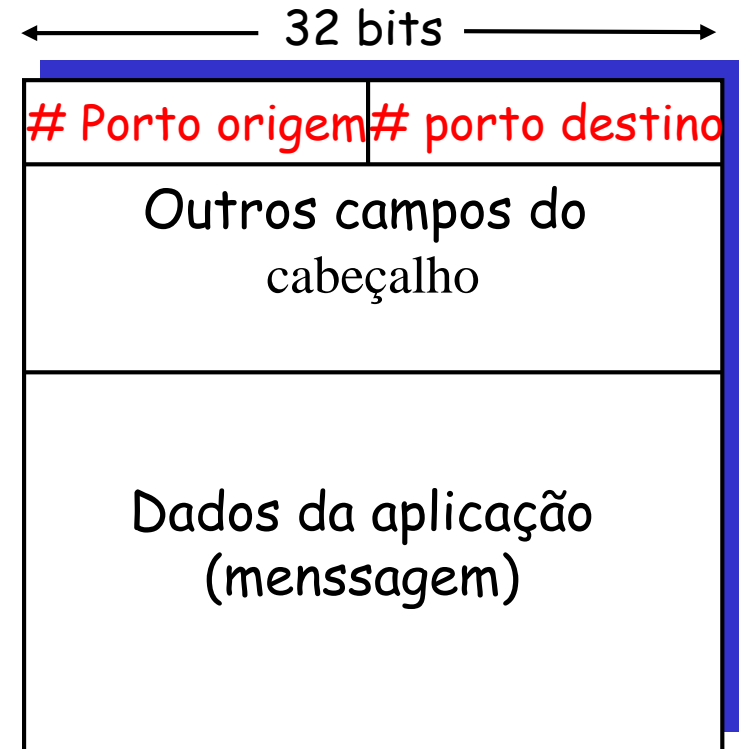


Como funciona a desmultiplexagem

❑ Máquina recebe datagrama IP

- Cada datagrama tem endereço IP de origem, endereço IP de destino
- Cada datagrama leva um segmento da camada de transporte
- Cada segmento tem números de porto de origem e de destino (relembre: nº de portos bem conhecidos para aplicações de rede)

❑ Máquina usa endereços IP e número de porto para enviar o segmento para o socket apropriado



Formato do segmento TCP/UDP

Desmultiplexagem sem ligação

- ❑ Criam-se sockets com números de portos:

```
DatagramSocket mySocket1 = new  
    DatagramSocket(12534);
```

```
DatagramSocket mySocket2 = new  
    DatagramSocket(12535);
```

- ❑ Socket **UDP** identificado por dois parâmetros:

IP address dest ,
port number dest

- ❑ Quando uma máquina recebe um datagrama UDP:
 - Verifica o nº do porto de destino no segmento
 - Envia o segmento UDP para o socket com esse nº de porto
- ❑ Envio independente do IP de origem e do Porto de origem - **apenas interessa o porto de destino!!!**

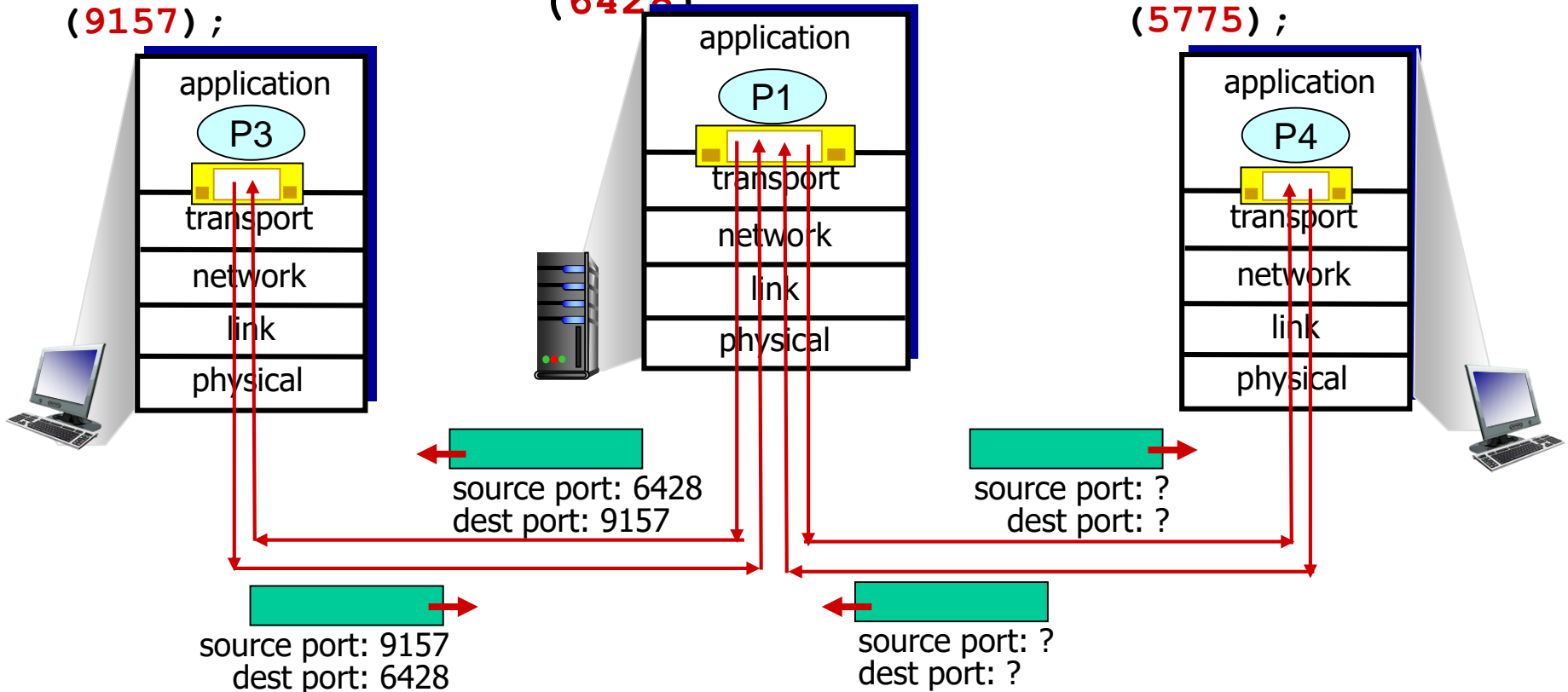
Desmultiplexagem sem ligação

DatagramSocket

```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```

```
serverSocket = new  
DatagramSocket  
(6428);
```

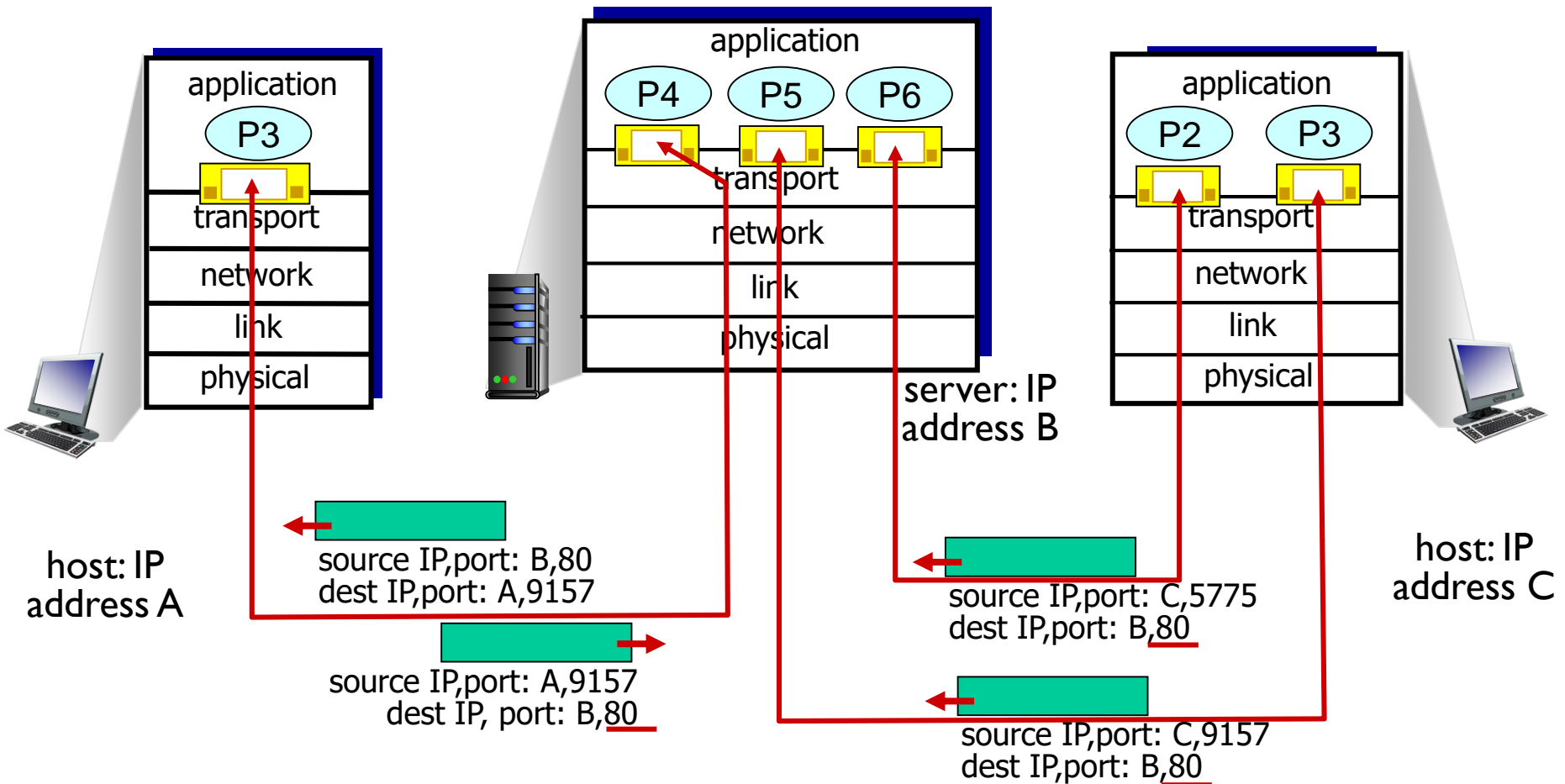
```
DatagramSocket  
mySocket1 = new  
DatagramSocket  
(5775);
```



Desmultiplexagem com ligação

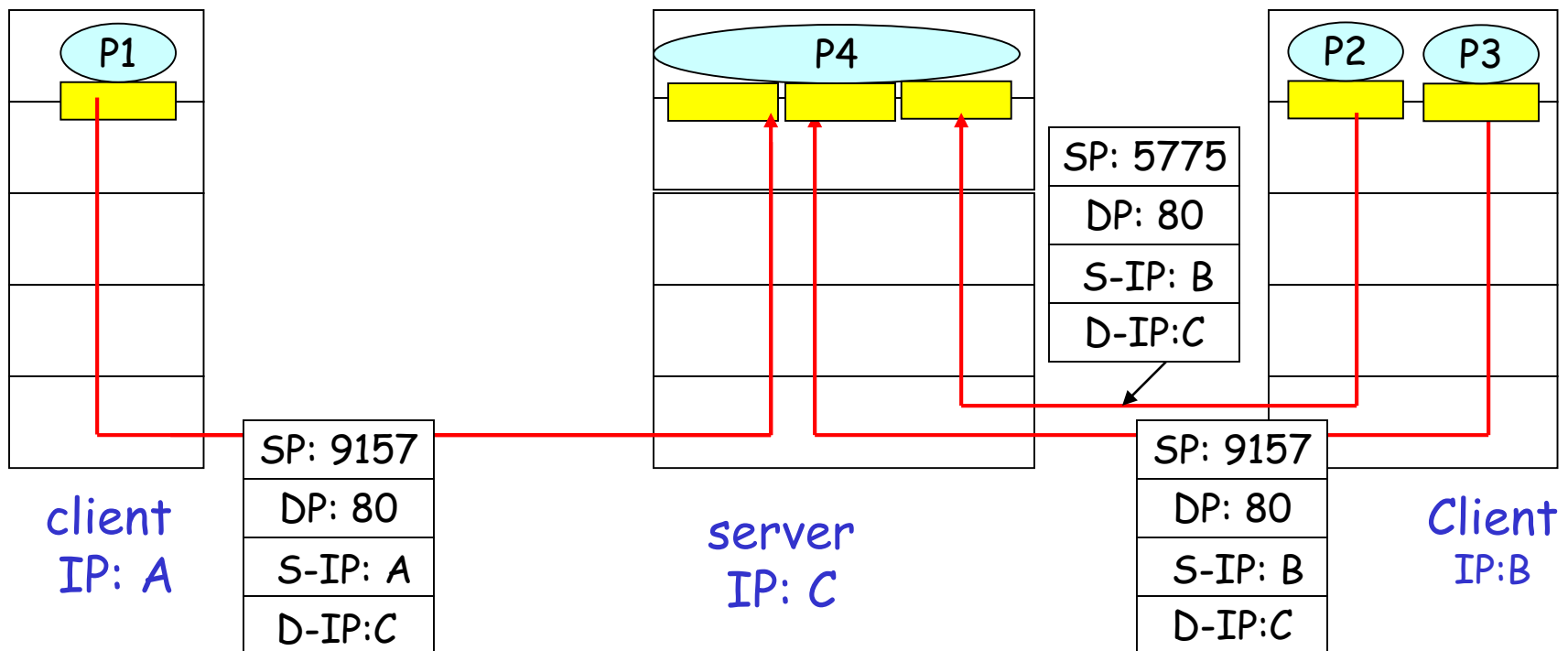
- ❑ Socket TCP identificado por 4 parâmetros:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- ❑ Receptor usa todos os 4 valores para enviar o segmento para o socket apropriado
- ❑ O servidor pode suportar muitos sockets TCP em simultâneo:
 - Cada socket é identificado pelos seus 4 parâmetros próprios
- ❑ Servidores Web têm sockets diferentes para cada cliente que se liga
 - HTTP não persistente terá sockets diferentes para cada pedido

Desmultiplexagem com ligação

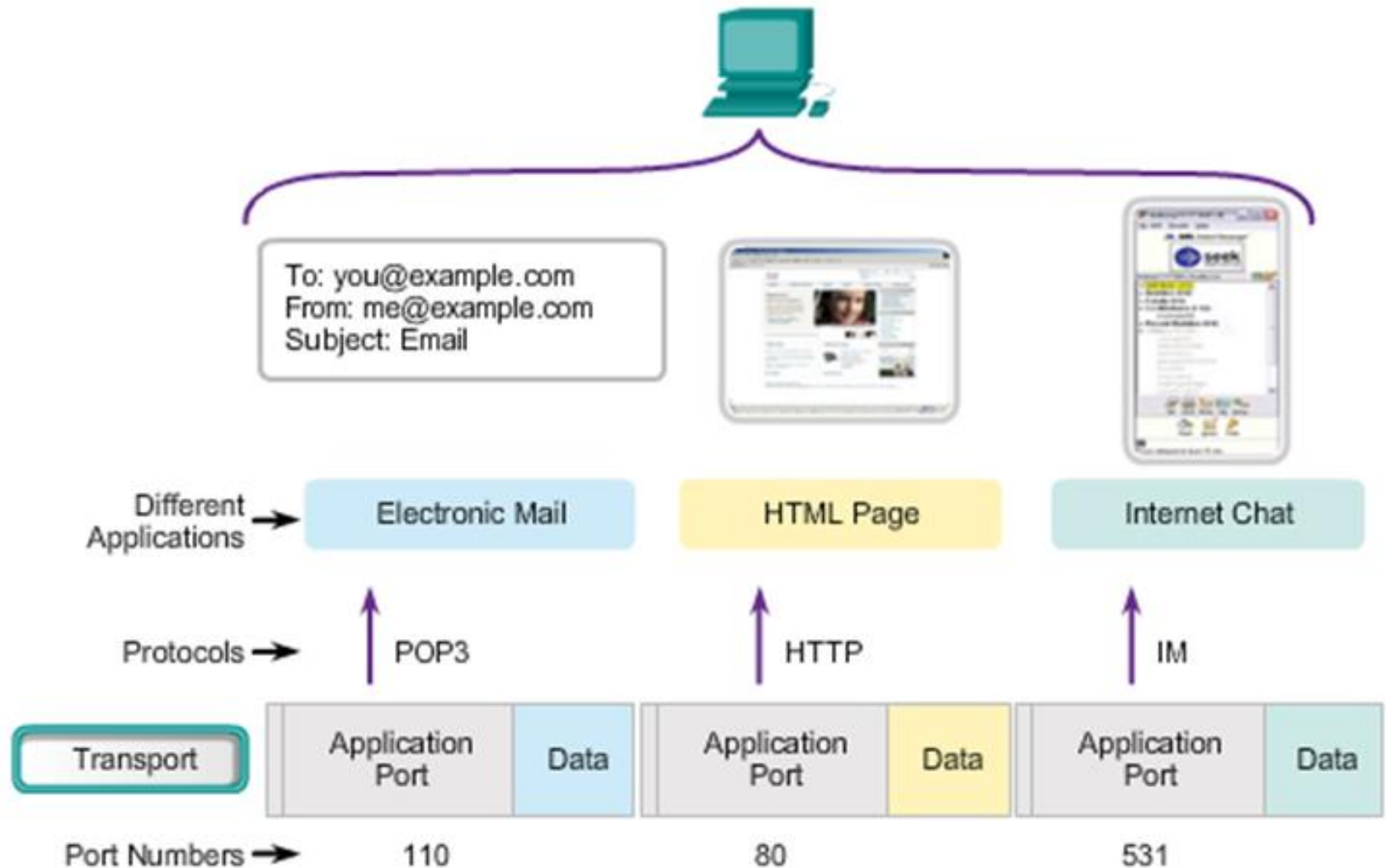


3 segmentos, todos com destino (IP address: B, dest port: 80) são desmultiplexados para *diferentes* sockets

Desmultiplexagem com ligação: Web Server



Portos de comunicação



Portos de comunicação

Port Number Range	Port Group
0 to 1023	Well Known (Contact) Ports
1024 to 49151	Registered Ports
49152 to 65533	Private and/or Dynamic Ports

Registered TCP Ports:

1863 MSN Messenger
2000 Cisco SCCP (VoIP)
8008 Alternate HTTP
8080 Alternate HTTP

Registered UDP Ports:

1812 RADIUS Authentication Protocol
5004 RTP (Voice and Video Transport Protocol)
5040 SIP (VoIP)

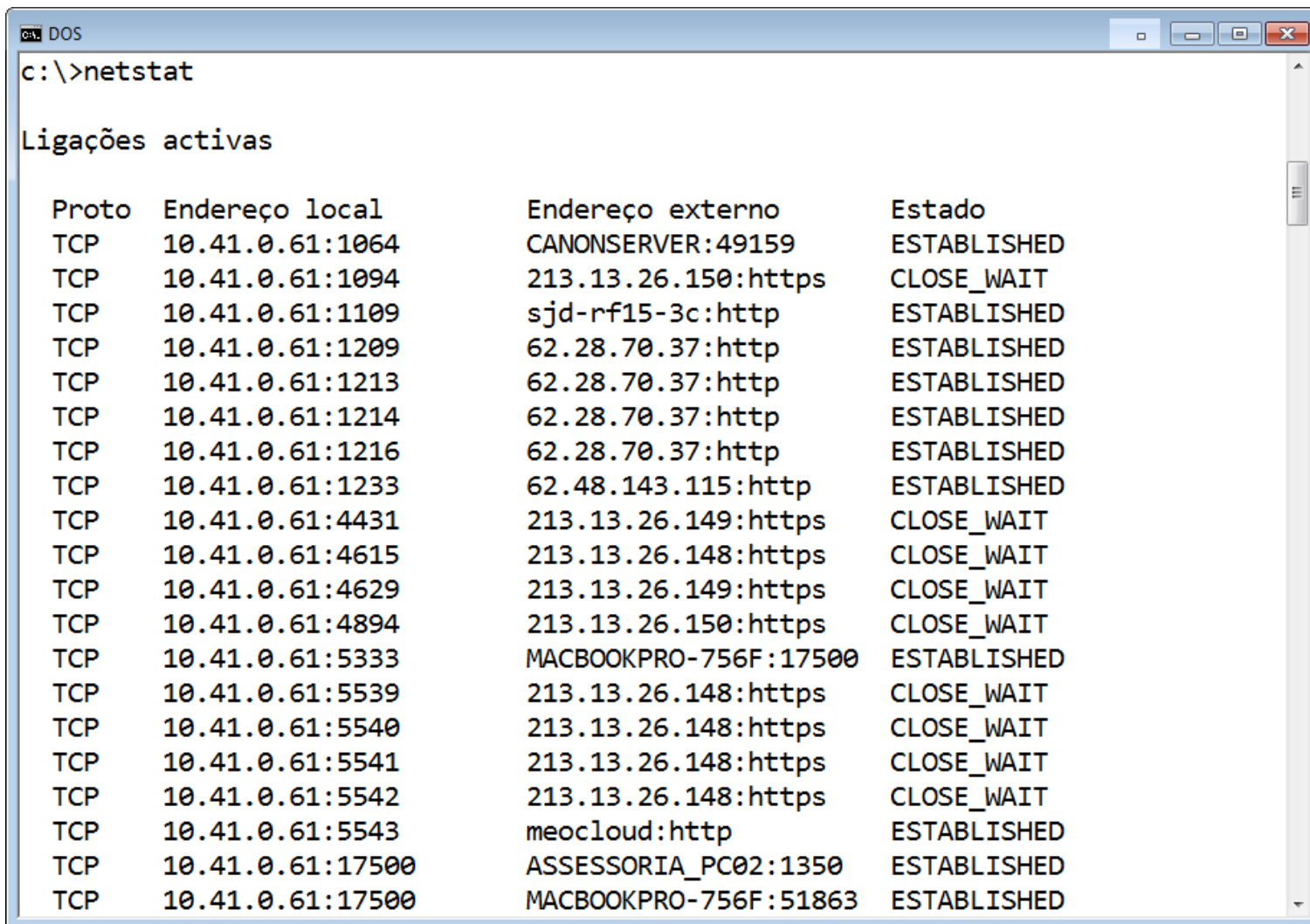
Well Known TCP Ports:

21 FTP
23 Telnet
25 SMTP
80 HTTP
110 POP3
194 Internet Relay Chat (IRC)
443 Secure HTTP (HTTPS)

Well Known UDP Ports:

69 TFTP
520 RIP

Portos TCP no Windows



```
c:\>netstat

Ligações activas

Proto  Endereço local          Endereço externo         Estado
TCP    10.41.0.61:1064          CANONSERVER:49159        ESTABLISHED
TCP    10.41.0.61:1094          213.13.26.150:https      CLOSE_WAIT
TCP    10.41.0.61:1109          sjd-rf15-3c:http         ESTABLISHED
TCP    10.41.0.61:1209          62.28.70.37:http         ESTABLISHED
TCP    10.41.0.61:1213          62.28.70.37:http         ESTABLISHED
TCP    10.41.0.61:1214          62.28.70.37:http         ESTABLISHED
TCP    10.41.0.61:1216          62.28.70.37:http         ESTABLISHED
TCP    10.41.0.61:1233          62.48.143.115:http       ESTABLISHED
TCP    10.41.0.61:4431          213.13.26.149:https      CLOSE_WAIT
TCP    10.41.0.61:4615          213.13.26.148:https      CLOSE_WAIT
TCP    10.41.0.61:4629          213.13.26.149:https      CLOSE_WAIT
TCP    10.41.0.61:4894          213.13.26.150:https      CLOSE_WAIT
TCP    10.41.0.61:5333          MACBOOKPRO-756F:17500    ESTABLISHED
TCP    10.41.0.61:5539          213.13.26.148:https      CLOSE_WAIT
TCP    10.41.0.61:5540          213.13.26.148:https      CLOSE_WAIT
TCP    10.41.0.61:5541          213.13.26.148:https      CLOSE_WAIT
TCP    10.41.0.61:5542          213.13.26.148:https      CLOSE_WAIT
TCP    10.41.0.61:5543          meocloud:http            ESTABLISHED
TCP    10.41.0.61:17500         ASSESSORIA_PC02:1350     ESTABLISHED
TCP    10.41.0.61:17500         MACBOOKPRO-756F:51863    ESTABLISHED
```

Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

UDP: User Datagram Protocol [RFC 768]

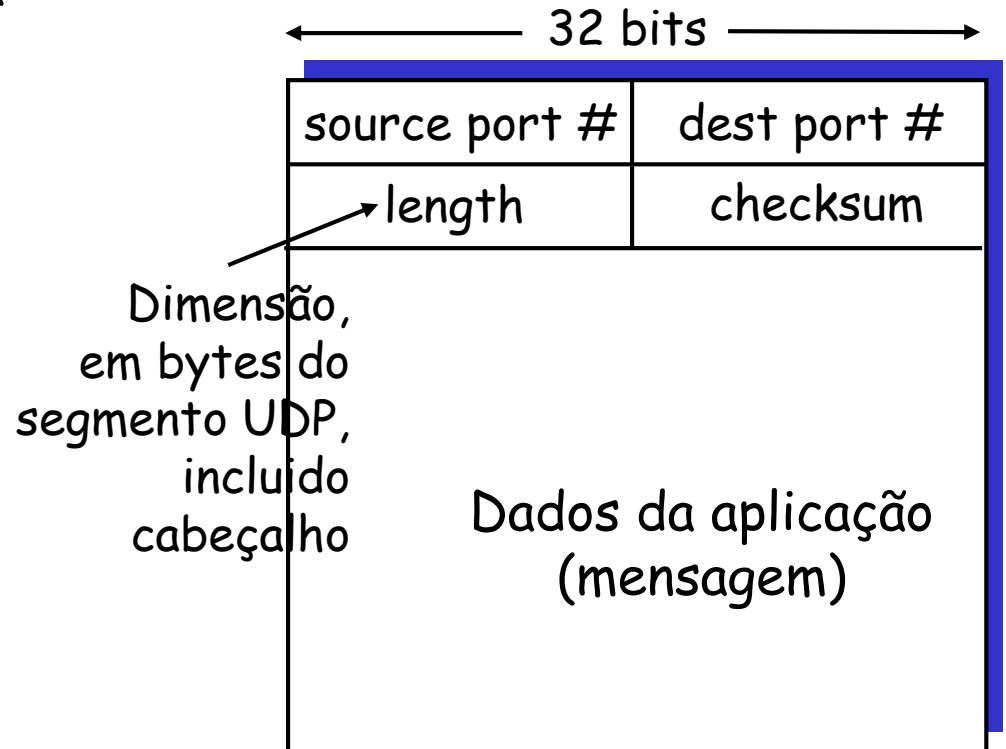
- ❑ Protocolo de transporte da Internet sem floreios, reduzido ao essencial
- ❑ Serviço de melhor esforço ("best effort")
- ❑ Segmentos UDP podem ser:
 - perdidos
 - Entregues fora de ordem à aplicação
- ❑ *Sem ligação:*
 - Sem handshaking emissor e receptor
 - Cada segmento UDP é independente dos demais

Porque existe UDP?

- ❑ Sem estabelecimento de ligação (que adiciona atraso)
- ❑ Simples: não há estado da ligação no emissor e receptor
- ❑ Cabeçalho do segmento é pequeno
- ❑ Não há controlo de congestão: UDP pode transmitir tão depressa quanto se queira

UDP: mais

- ❑ Usualmente utilizado para aplicações com fluxo multimédia, que são:
 - tolerantes a perdas
 - Sensíveis a atrasos
- ❑ Outras utilizações do UDP
 - DNS
 - SNMP
- ❑ A fiabilidade terá de ser garantida pela camada de aplicação
 - Recuperação de erros específica da aplicação!



Formato do segmento UDP

Checksum UDP

objectivo: detectar "erros" (e.g., bits trocados) no segmento transmitido

emissor:

- ❑ Trata o conteúdo do segmento como uma sequência de "inteiros" de 16 bits
- ❑ **checksum:** soma do conteúdo do segmento em complemento para 1
- ❑ Emissor coloca o valor da soma no campo checksum do segmento UDP

receptor:

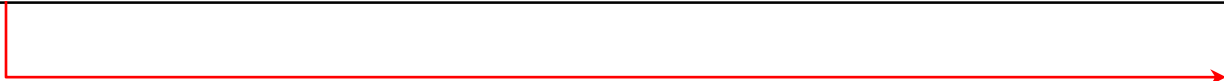
- ❑ Calcula a soma do conteúdo do segmento recebido
- ❑ Compara a soma efectuada com o valor do campo checksum recebido:
 - diferente- erro detectado
 - Igual- sem erro
 - *Mas poderão haver erros?*

Exemplo: Internet Checksum

□ Nota

- Ao somar 2 números, o transporte do bit mais significativo deve ser adicionado ao resultado

□ Exemple: soma de 2 inteiros de 16-bits

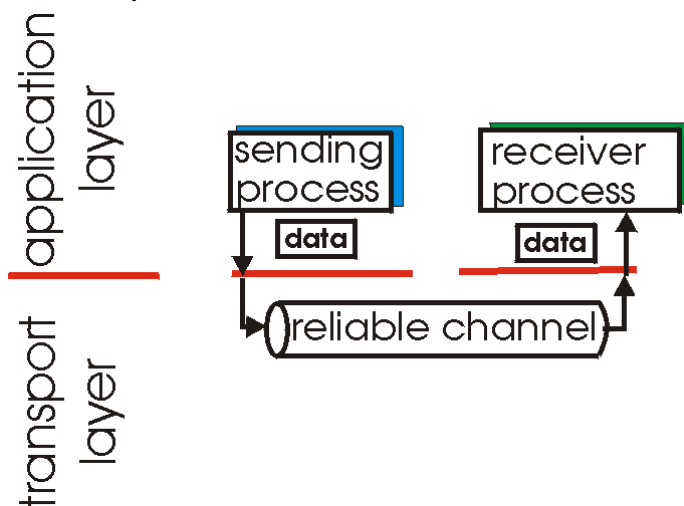
		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		<hr/>															
transporte	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
		<hr/>															
																	
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

Princípios da transmissão de dados fiável

- ❑ Importante nas aplicações, transporte e ligação de dados
- ❑ Faz parte da lista 10+ de assuntos de redes!!!



(a) provided service

- ❑ As características do canal não fiável determinam a complexidade do protocolo de transferência de dados fiável

rdt3.0: canais com erros e perdas

Novo pressuposto: canal utilizado **TAMBÉM** pode perder pacotes (dados ou ACK)

- Checksum, nº seq, ACK, retransmissão ajudam mas não são suficientes

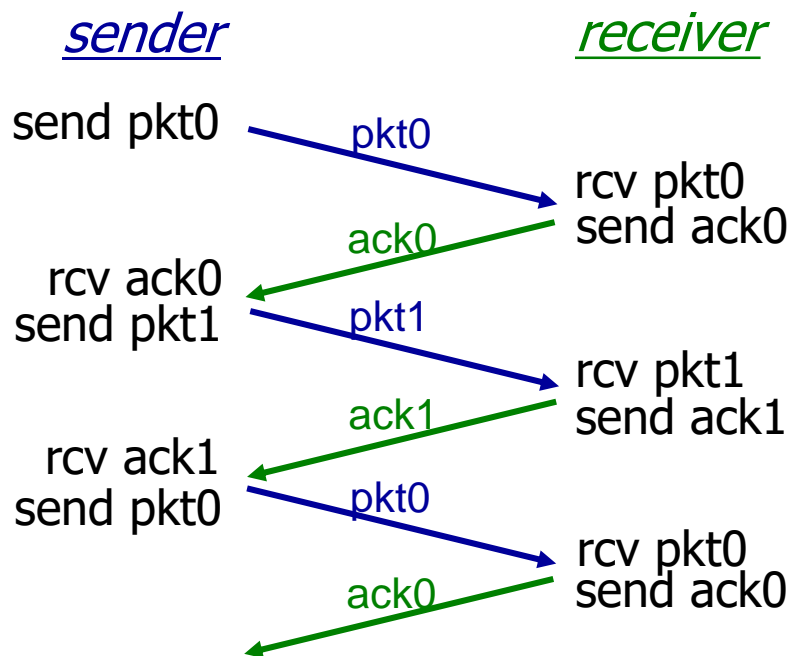
Como lidar com as perdas?

Emissor espera até ter a certeza que os dados ou ACK se perdeu, depois retransmite

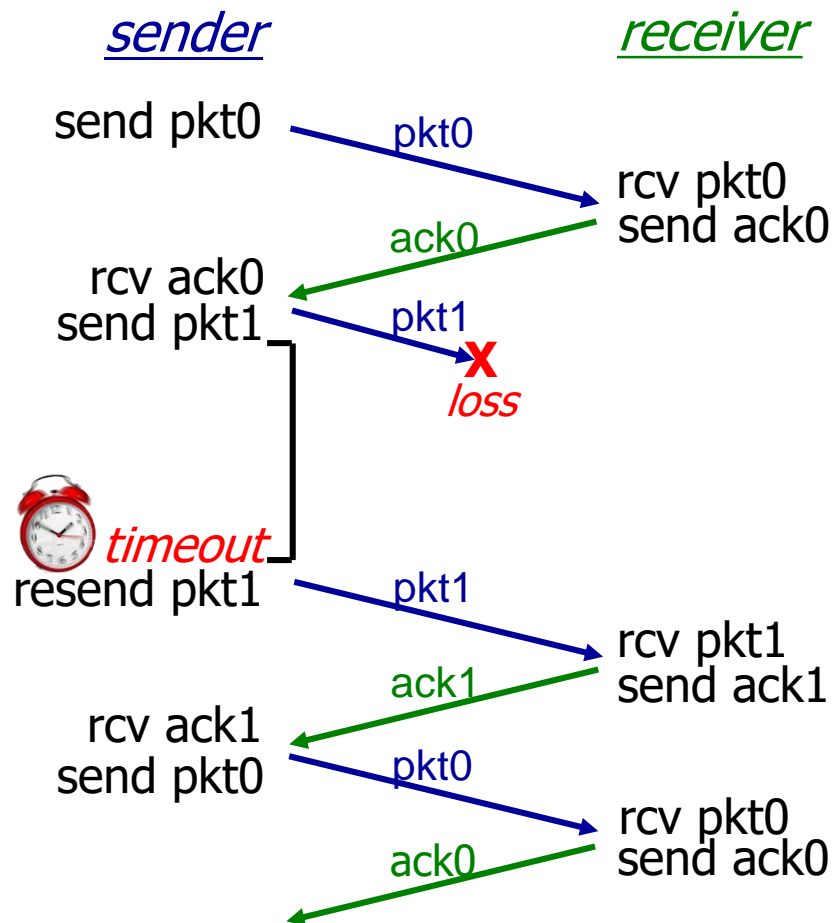
aproximação: emissor espera um tempo "razoável" por um ACK

- Retransmite se o ACK não for recebido nesse tempo
- Se o pacote de dados (ou ACK) se tiver atrasado (mas não perdido):
 - Retransmissão duplica o pacote, mas o nº de sequência trata disso
 - Receptor tem de especificar o nº de seq. do pacote que está a ser confirmado (ACKed)
- Necessário contador decrescente (countdown timer)

rdt3.0 em acção



(a) Sem perdas

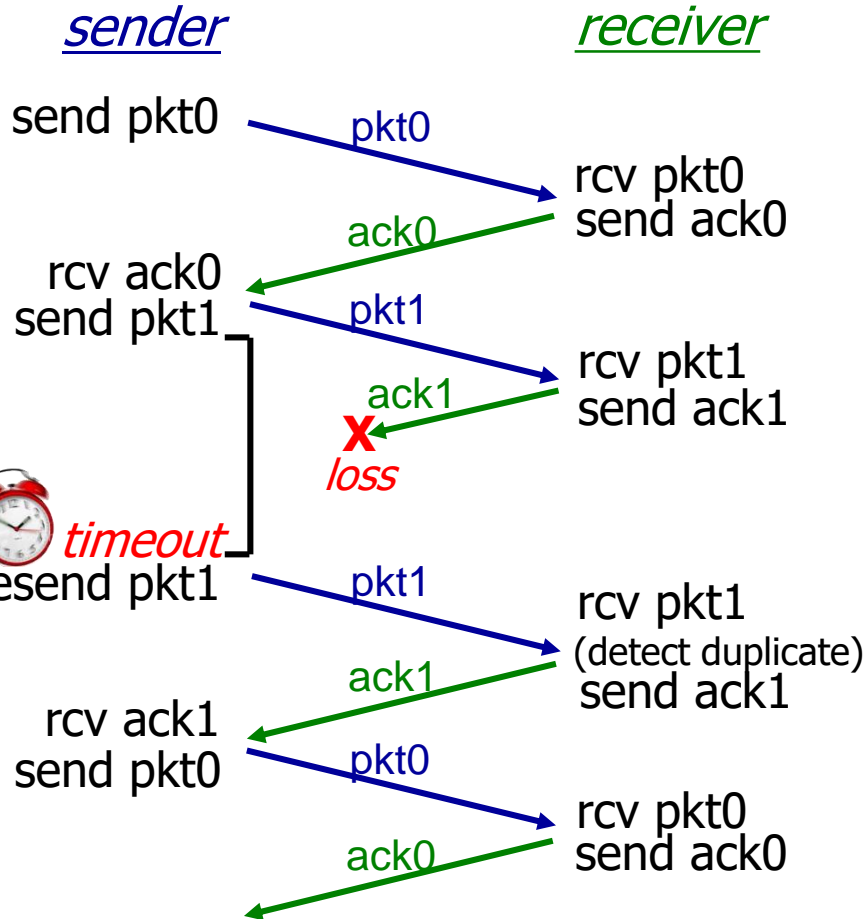


(b) Com perdas

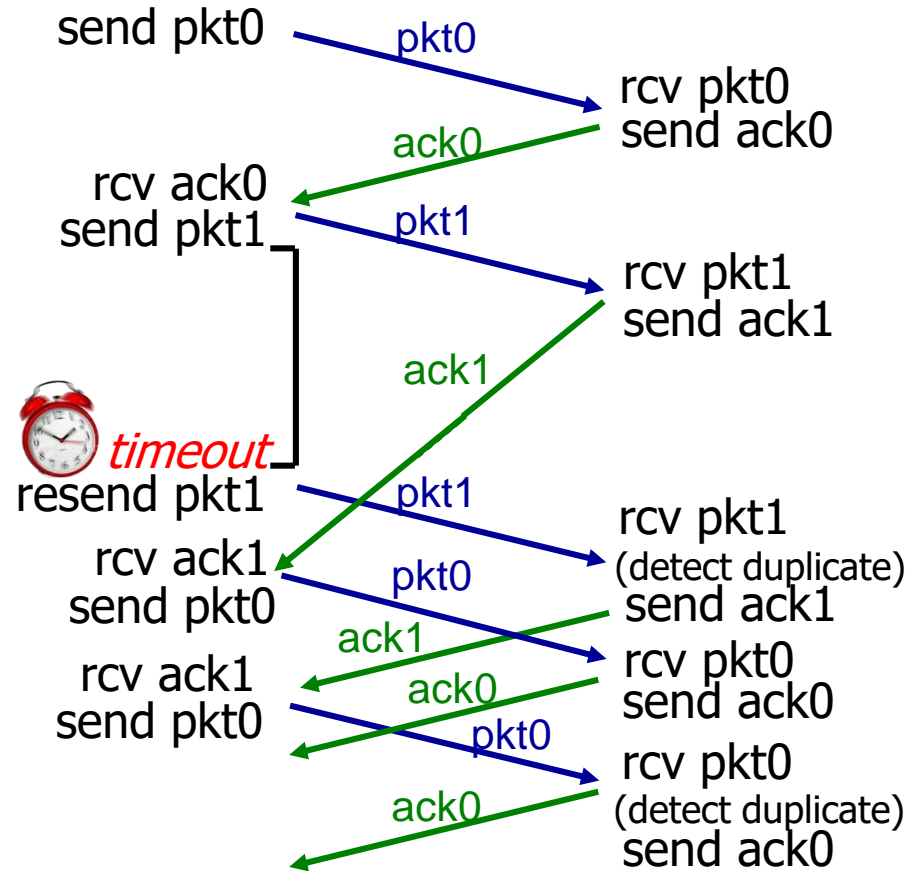
rdt3.0 em acção

sender

receiver



(c) ACK perdido



(d) timeout/ delayed ACK prematuro

Desempenho do rdt3.0

- ❑ rdt3.0 funciona, mas o seu desempenho deixa muito a desejar....
- ❑ ex: Ligação 1 Gbps,
- ❑ Tempo de propagação ponto a ponto = 15 ms,
- ❑ Pacote de 1 KB = 8000 bit packet:

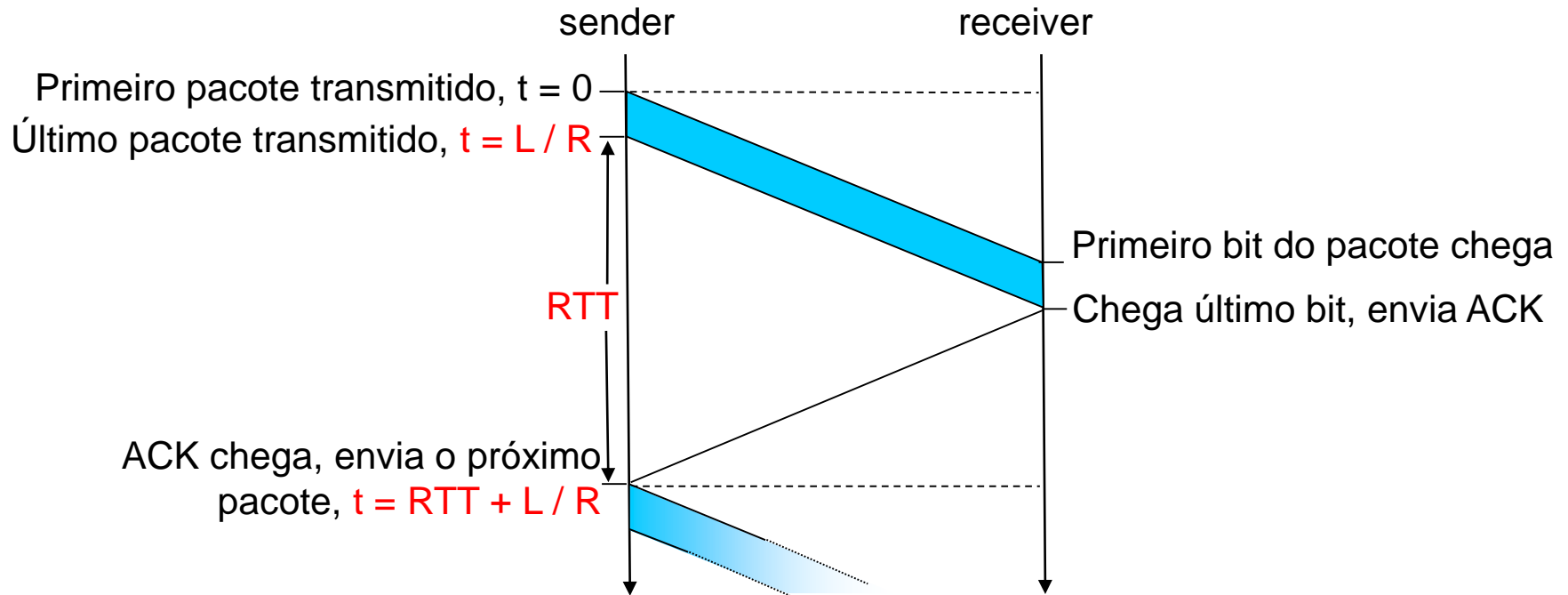
$$t_{trans} = \frac{L}{R} = \frac{8 \times 10^3 \text{ bits}}{10^9 \text{ bps}} = 8 \mu s$$

- U_{emissor} : **utilização/eficiência**- fracção de tempo em que o emissor está ocupado a enviar

$$U_{\text{emis}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- Um pacote de 1KB a cada 30 msec -> débito de 33kB/seg num link de 1 Gbps
- O protocolo de comunicação limita o uso dos recursos físicos!

rdt3.0: operação stop-and-wait

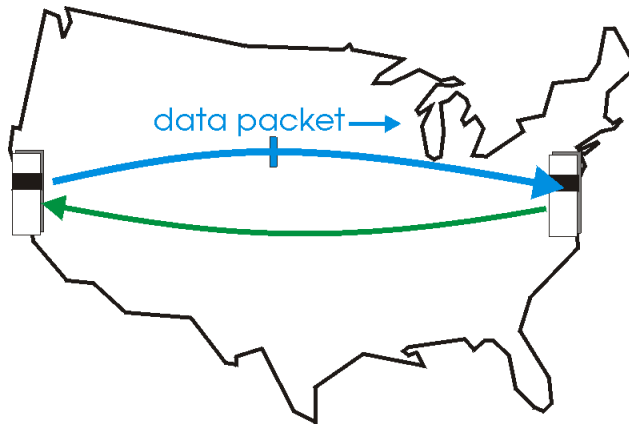


$$U_{\text{emiss}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

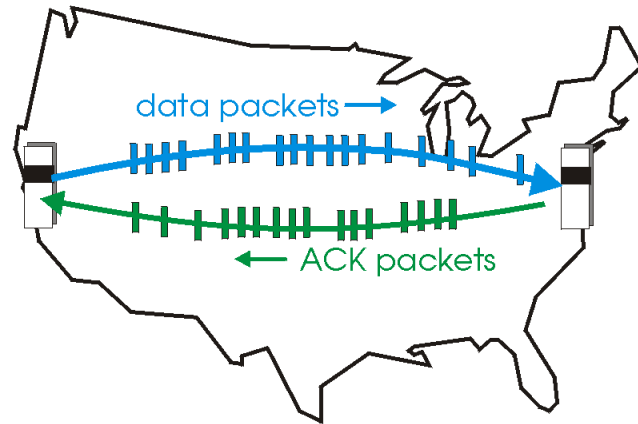
Protocolos em *pipeline*

Pipelining: o emissor permite o envio de múltiplos pacotes, ainda que não confirmados

- Intervalo dos n° de sequência tem de aumentar
- Armazenamento no emissor e no receptor



(a) um protocolo stop-and-wait em funcionamento

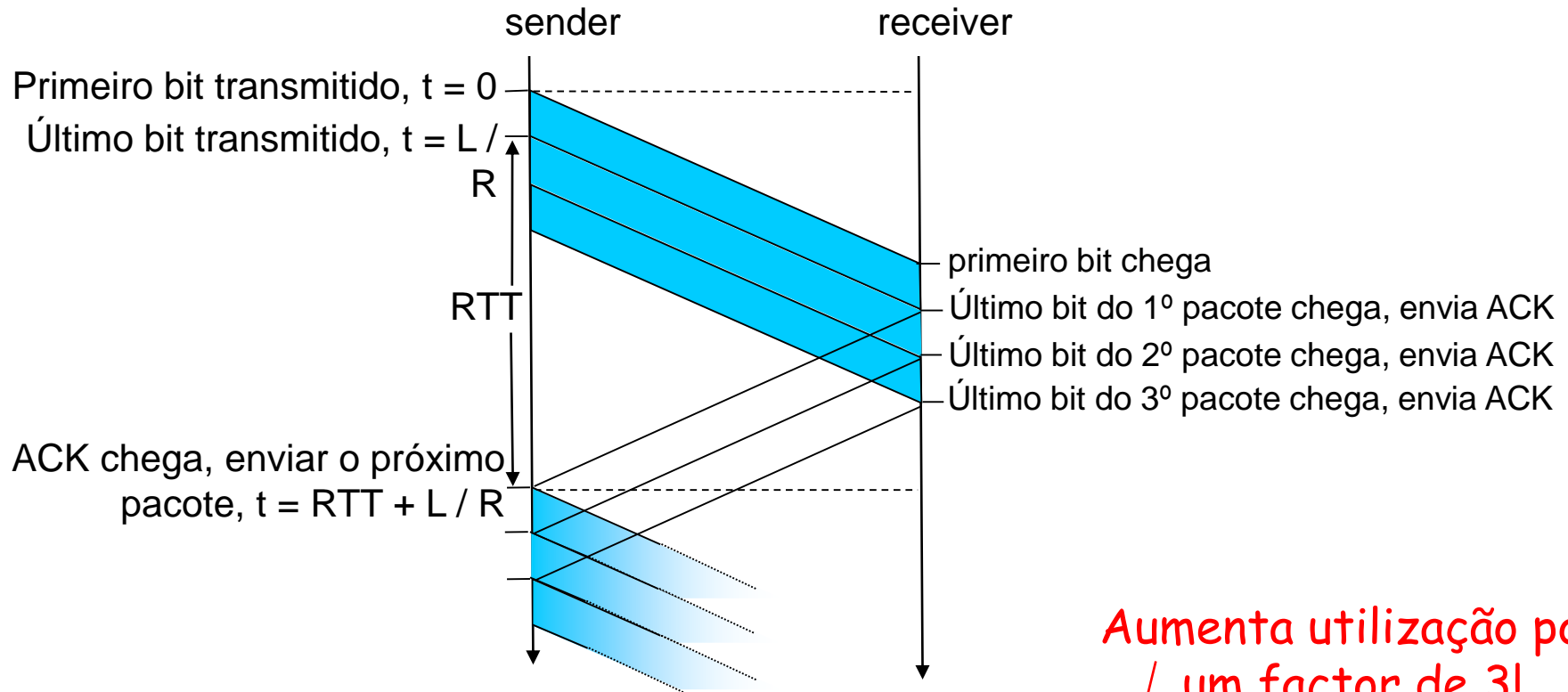


(b) um protocolo com pipeline em funcionamento

□ Duas formas genéricas de protocolos em pipeline:

- *Voltar atrás N (go-Back-N)*
- *Repetição selectiva (selective repeat)*

Pipelining: utilização melhorada



Aumenta utilização por
um factor de 3!

$$U_{\text{emiss}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Protocolos de Pipelining

Go-back-N:

- ❑ Emissor pode ter até N pacotes não confirmados em transito (no pipeline)
- ❑ Receptor apenas envia ACKs cumulativos
 - Não confirma pacotes se houver faltas no n° de seq.
- ❑ Emissor tem um timer para o 1° pacote não confirmado
 - Se o timer expira, retransmite todos os pacotes não confirmados

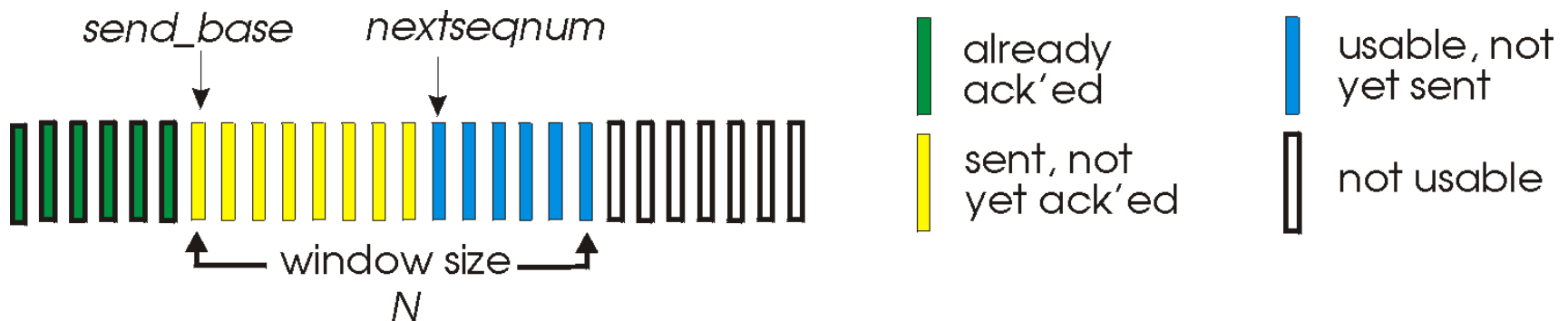
Selective Repeat:

- ❑ Emissor pode ter até N pacotes não confirmados em transito (no pipeline)
- ❑ Receptor confirma os pacotes individualmente
- ❑ Emissor mantém timer para cada pacote não confirmado
 - Quando o timer expira, retransmite apenas o pacote não confirmado

Go-Back-N

Emissor :

- ❑ Cabeçalho do pacote com k bits para o nº de sequência
- ❑ Janela de até N pacotes consecutivos, ainda não confirmados



- ❑ ACK(n): confirma todos os pacotes até ao nº de seq n- "ACK cumulativos"
 - Pode receber ACKs duplicados (ver receptor)
- ❑ timer for each in-flight pkt
- ❑ Timeout (retransmite pacote n e todos os pacotes de nº de seq. superior na janela)

GBN em ação

sender window (N=4)

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

sender

send pkt0
 send pkt1
 send pkt2
 send pkt3
 (wait)

rcv ack0, send pkt4
 rcv ack1, send pkt5

ignore duplicate ACK



pkt 2 timeout

send pkt2
 send pkt3
 send pkt4
 send pkt5

receiver

receive pkt0, send ack0
 receive pkt1, send ack1

receive pkt3, discard,
 (re)send ack1

receive pkt4, discard,
 (re)send ack1

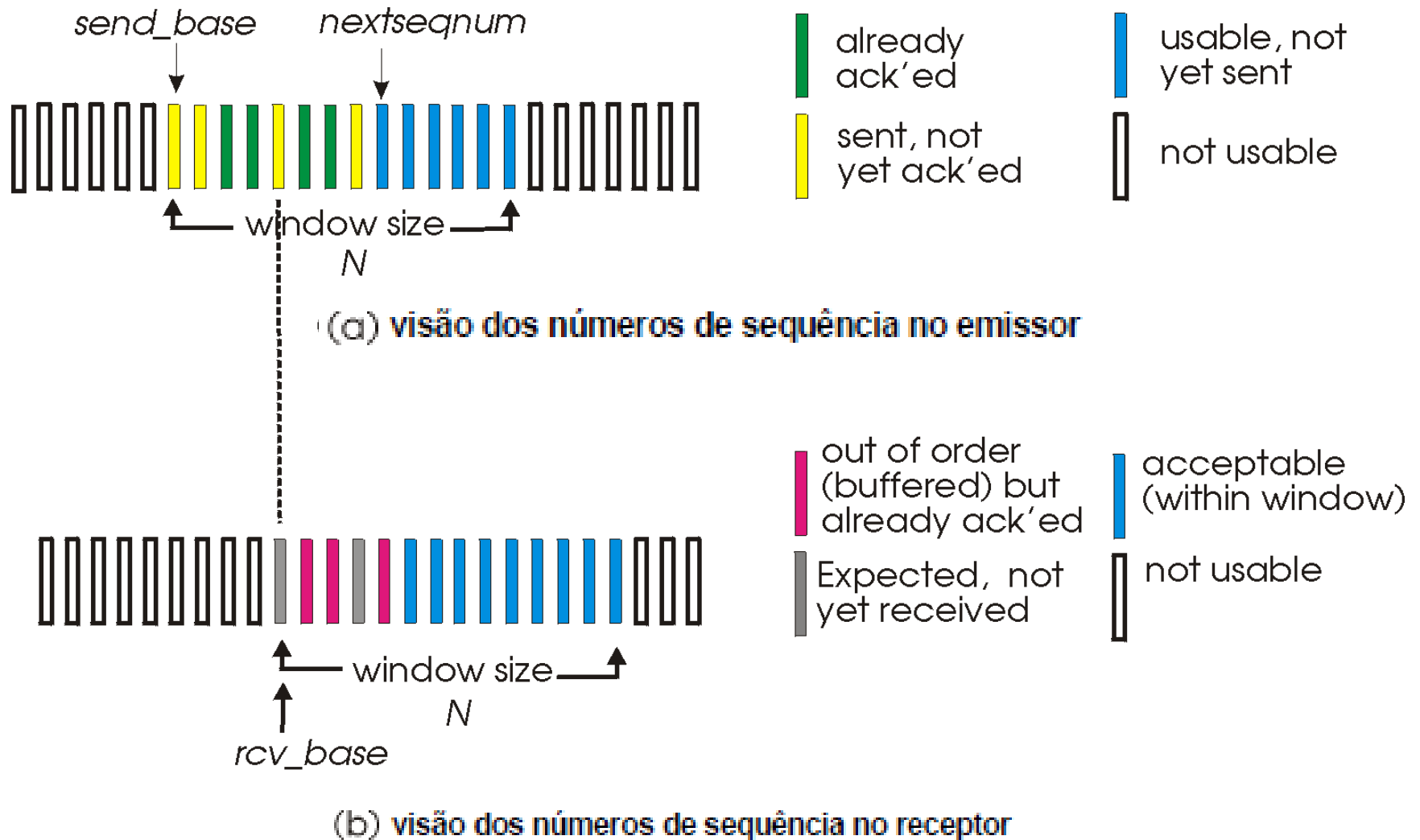
receive pkt5, discard,
 (re)send ack1

rcv pkt2, deliver, send ack2
 rcv pkt3, deliver, send ack3
 rcv pkt4, deliver, send ack4
 rcv pkt5, deliver, send ack5

Repetição selectiva

- ❑ Receptor faz o ACK individual de todos os pacotes correctamente recebidos
 - Armazena pacotes, quando necessário, para os poder entregar por ordem ao nível superior
- ❑ Emissor apenas reenvia pacotes para os quais o ACK não tenha sido recebido.
 - Temporizador no emissor para cada pacote por confirmar
- ❑ Janela do emissor
 - N números de sequência consecutivos
 - Novamente limita os números de sequência dos pacotes enviados, por confirmar

Repetição selectiva: janela do emissor e receptor



Repetição selectiva

emissor

Dados para enviar:

- ❑ Se o próximo n° de seq disponível cabe na janela

timeout(n):

- ❑ Reenvia pacote n

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

- ❑ Marca pacote n como recebido
- ❑ Se n é o pacote mais antigo por confirmar, avança a base da janela para o próximo pacote por confirmar

receptor

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

- ❑ Envia ACK(n)
- ❑ Fora de ordem? - armazena
 - Por ordem?: entrega (também entrega os pacotes armazenados que estejam na ordem)
 - Avança janela para o próximo pacote ainda não recebido

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

- ❑ ACK(n)

De outro modo:

- ❑ ignora

Selective repeat em ação

sender window (N=4)

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

sender

send pkt0

send pkt1

send pkt2

send pkt3

(wait)

rcv ack0, send pkt4

rcv ack1, send pkt5

record ack3 arrived



pkt 2 timeout

send pkt2

record ack4 arrived

record ack5 arrived

receiver

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, buffer,
send ack3

receive pkt4, buffer,
send ack4

receive pkt5, buffer,
send ack5

rcv pkt2; deliver pkt2,
pkt3, pkt4, pkt5; send ack2

Q: what happens when ack2 arrives?

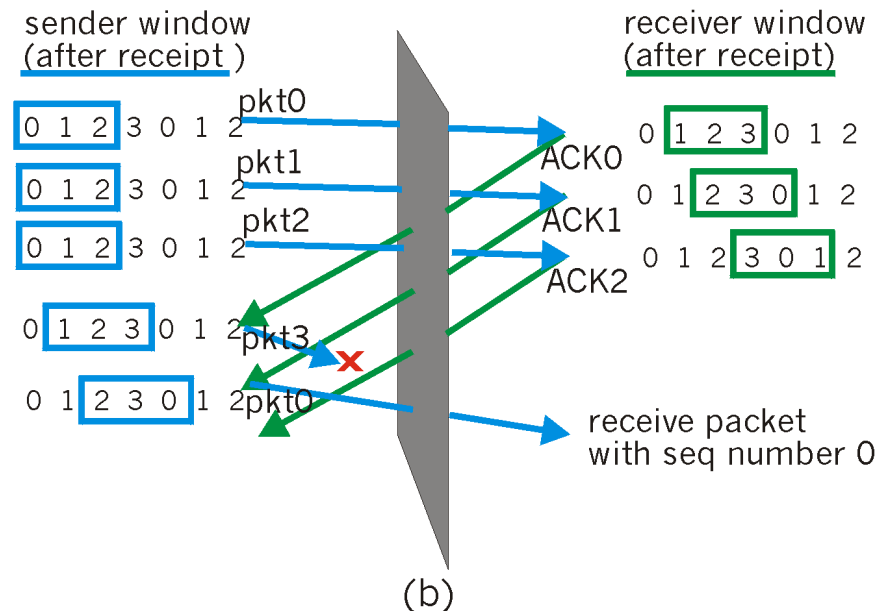
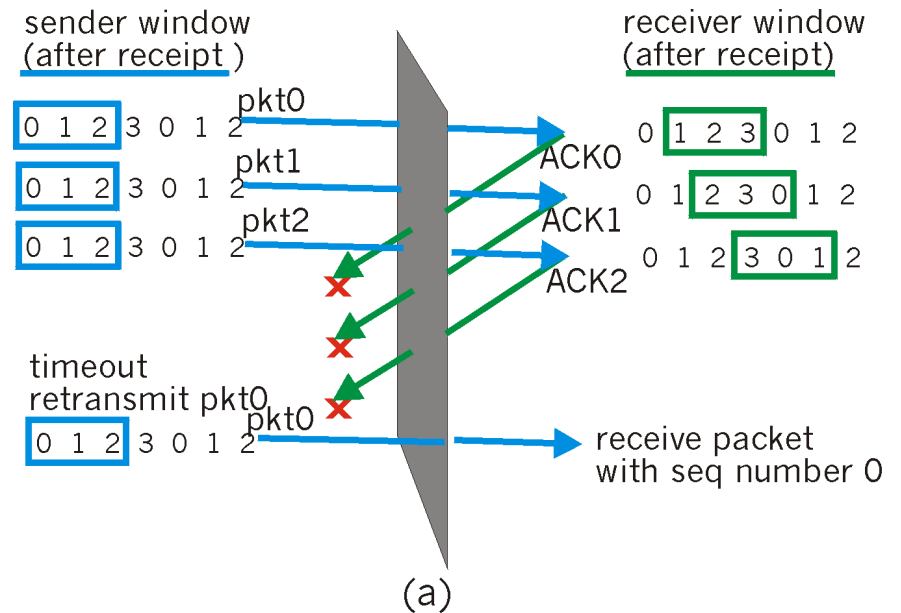
Selective repeat: dilema

Exemplo:

- nº seq: 0, 1, 2, 3
- Dimensão da janela = 3

- O receptor não vê diferença entre os 2 cenários!
- Dados duplicados são passados incorrectamente como novos em (a)

P: qual a relação entre o nº de seq. disponíveis e a dimensão da janela?



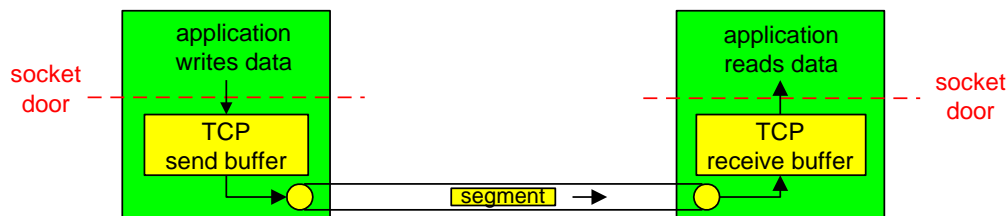
Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

TCP: Visão Geral

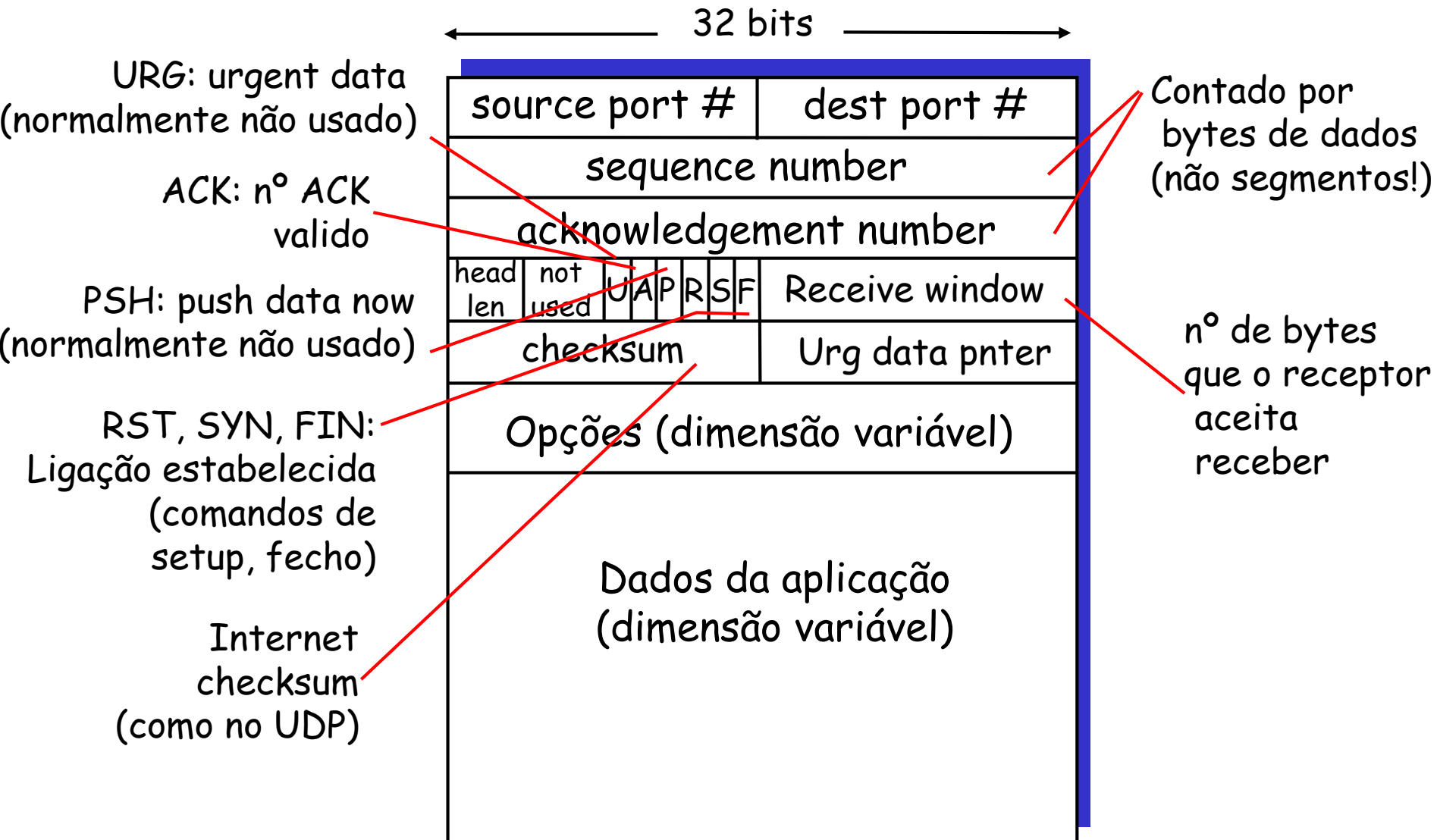
RFCs: 793, 1122, 1323, 2018, 2581

- ❑ **Ponto a ponto:**
 - Um emissor, um receptor
- ❑ **Fluxo de bytes ordenado e fiável:**
 - não há delimitação de mensagens"
- ❑ **pipelined:**
 - dimensão da janela definida pelo controlo de congestão e de fluxo do TCP
- ❑ **Buffers de emissão e recepção**



- ❑ **Transmissão full duplex:**
 - Transmissão de dados bidireccional na mesma ligação
 - MSS: maximum segment size
- ❑ **Orientada à ligação:**
 - handshaking (transferência de mensagens de controlo) inicia o estado do emissor e do receptor antes de transferir dados
- ❑ **Controlo de fluxo:**
 - Emissor não sobrecarrega receptor

TCP: estrutura do segmento



Nº de seq. TCP e ACKs

Nº Seq.:

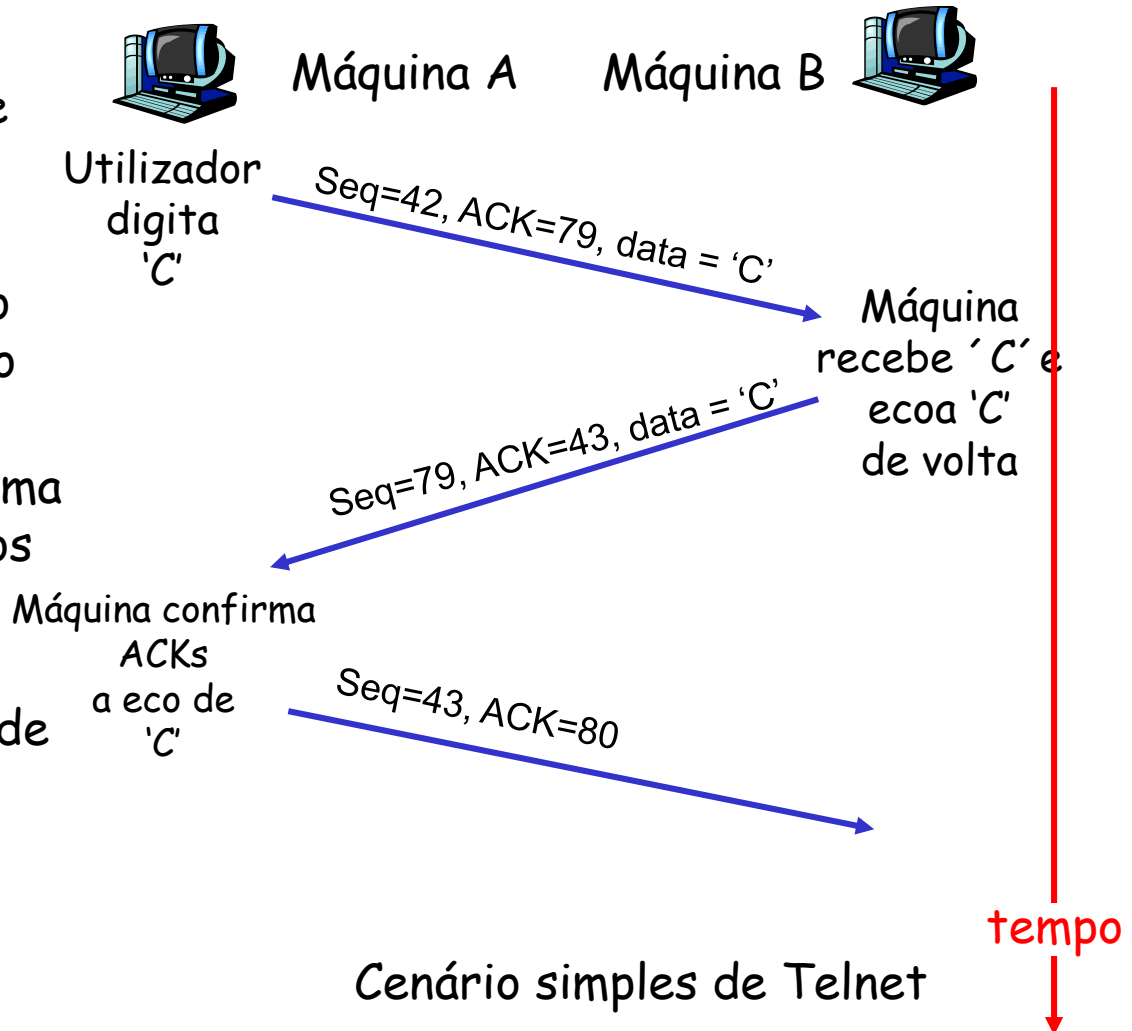
- Nº do primeiro byte de dados no segmento

ACKs:

- Nº de seq. do próximo byte esperado do outro lado
- ACK cumulativo: confirma a recepção correcta dos bytes anteriores

P: Como é que o receptor processa segmentos fora de ordem?

- R: TCP não especifica, deixa a questão para a implementação



TCP Round Trip Time e Timeout

Q: como definir a duração do temporizador do TCP?

- ❑ Maior que o RTT
 - mas RTT varia
- ❑ Demasiado curto: timeout prematuro
 - Retransmissões desnecessárias
- ❑ Demasiado longo: reacção lenta a perdas de segmentos

Q: como estimar o RTT?

- ❑ **SampleRTT**: tempo medido desde a transmissão de um segmento até à recepção do seu ACK
 - ignora retransmissões
- ❑ **SampleRTT** vai variar, quer-se uma estimativa do RTT com variações "suaves"
 - Fazer a média de várias medidas recentes, não apenas o **SampleRTT** actual

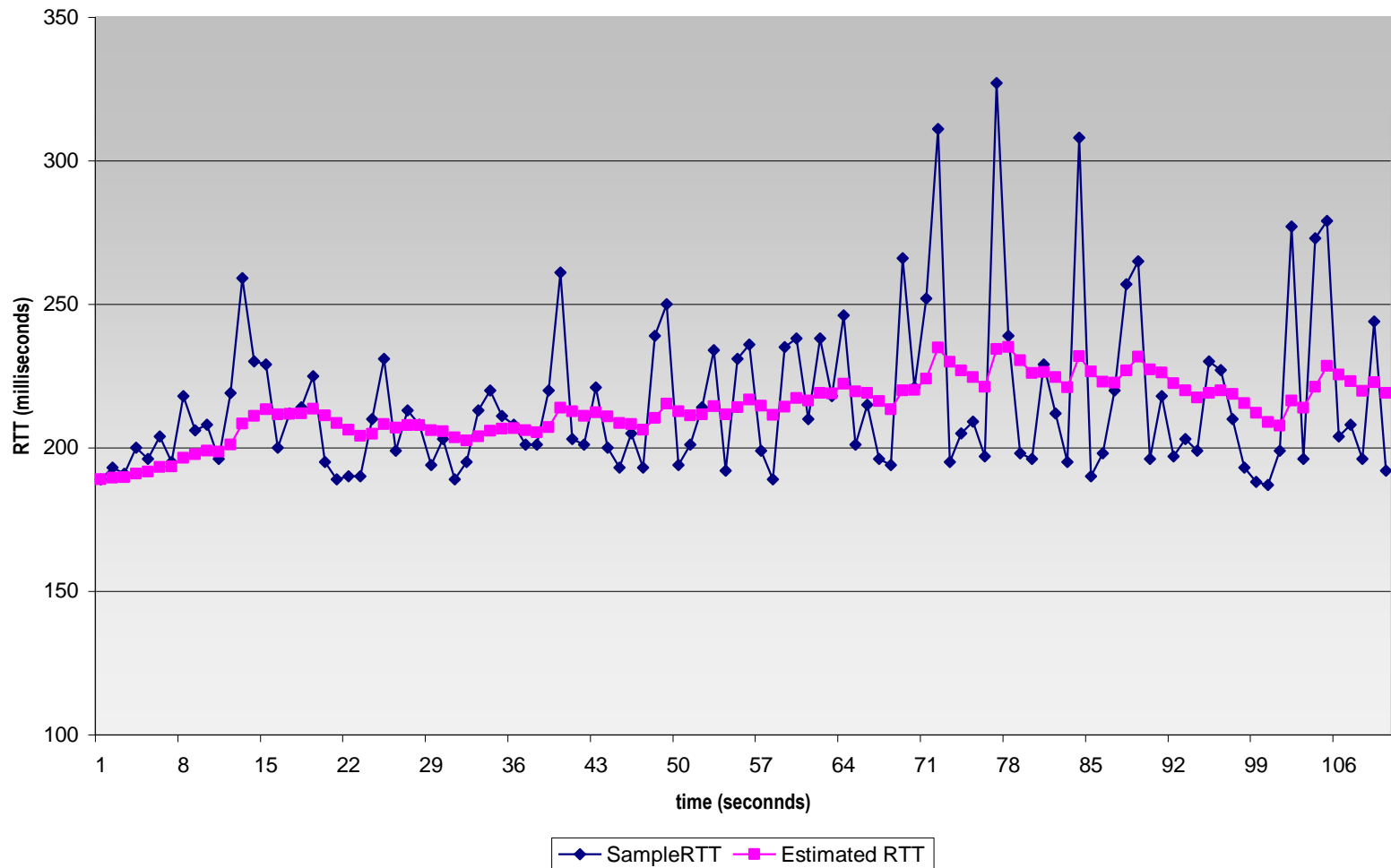
TCP Round Trip Time e Timeout

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

- ❑ Média móvel de peso exponencial (exponential wieghted moving average)
- ❑ Influência de uma amostra passada decresce com rapidez exponencial
- ❑ Valor típico: $\alpha = 0.125$

Exemplo de estimativa do RTT :

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



TCP Round Trip Time e Timeout

Definir a duração do temporizador (timeout)

- ❑ **EstimatedRTT** mais "margem de segurança"
 - Grande variação no **EstimatedRTT** -> maior margem de segurança
- ❑ Primeiro estima-se quanto é que as amostras **SampleRTT** se desviam do **EstimatedRTT**:

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

(tipicamente, $\beta = 0.25$)

definir o duração do temporizador como:

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

Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

TCP: transferência de dados fiável

- ❑ O TCP cria um serviço de transferência de dados fiável (RDT) por cima do serviço não fiável IP
- ❑ Transmissão de segmentos em pipeline
- ❑ ACKs Cumulativos
- ❑ O TCP usa um único temporizador de retransmissão
- ❑ Retransmissões desencadeadas por:
 - Eventos de timeout
 - ACKs duplicados
- ❑ Considere-se inicialmente um emissor TCP simplificado:
 - ignora ACKs duplicados
 - ignora controlo de fluxo e controlo de congestão

TCP: eventos do emissor

Dados recebidos da aplicação:

- ❑ Cria segmento com nº de seq.
- ❑ Nº de seq. é o nº do primeiro byte de dados do segmento
- ❑ Arma o temporizador, se já não estiver armado (o temporizador é para o segmentos mais antigo por confirmar)
- ❑ Duração do temporizador: `TimeoutInterval`

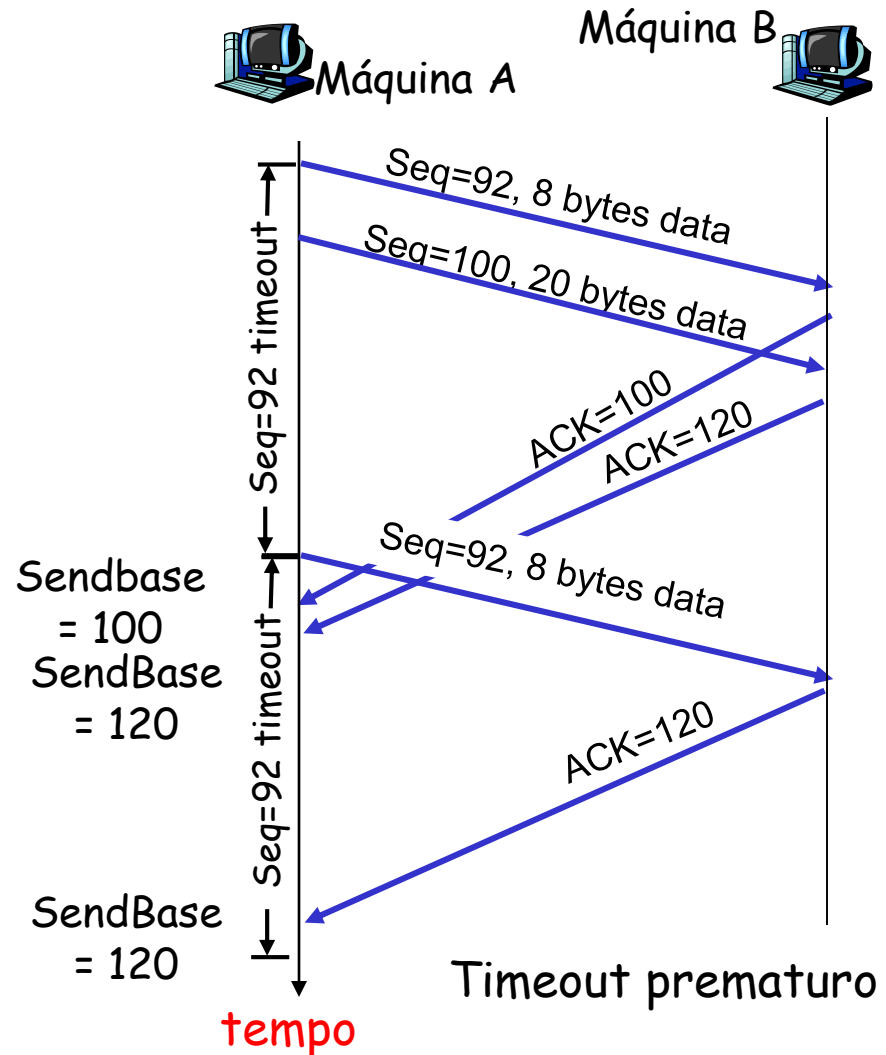
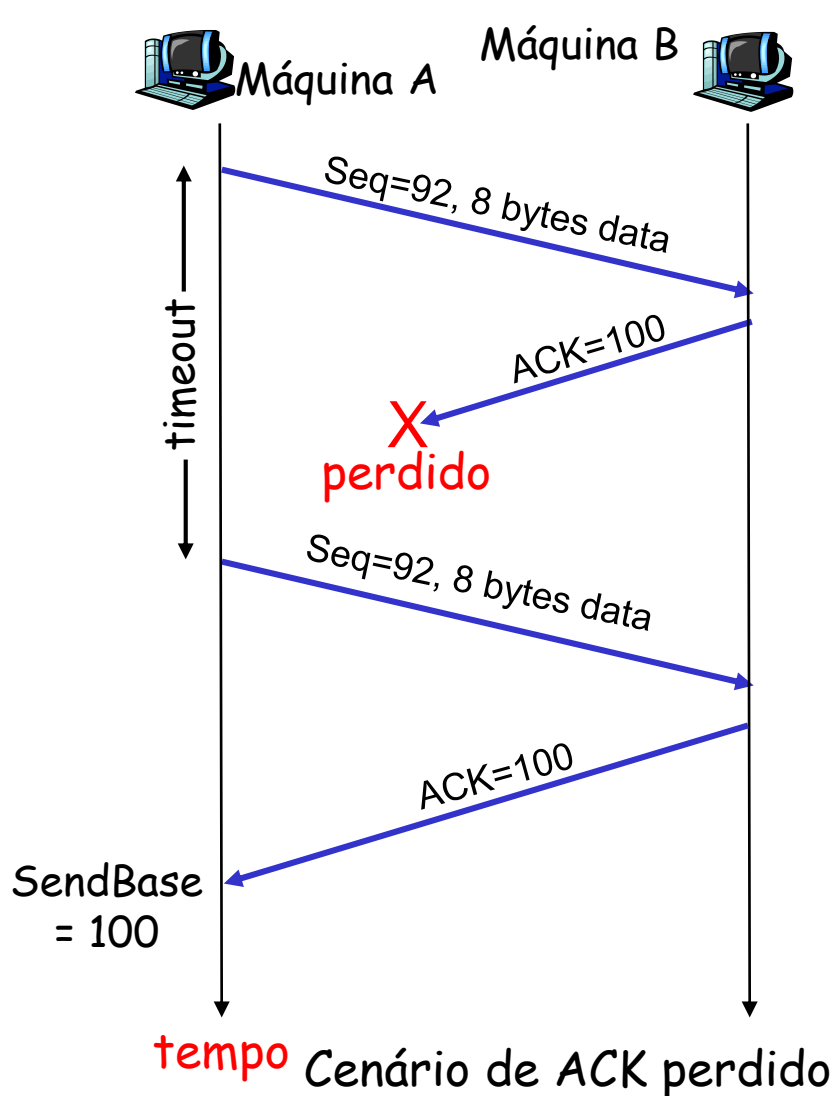
Temporizador expira:

- ❑ retransmitir segmento que causou timeout
- ❑ Rearma temporizador

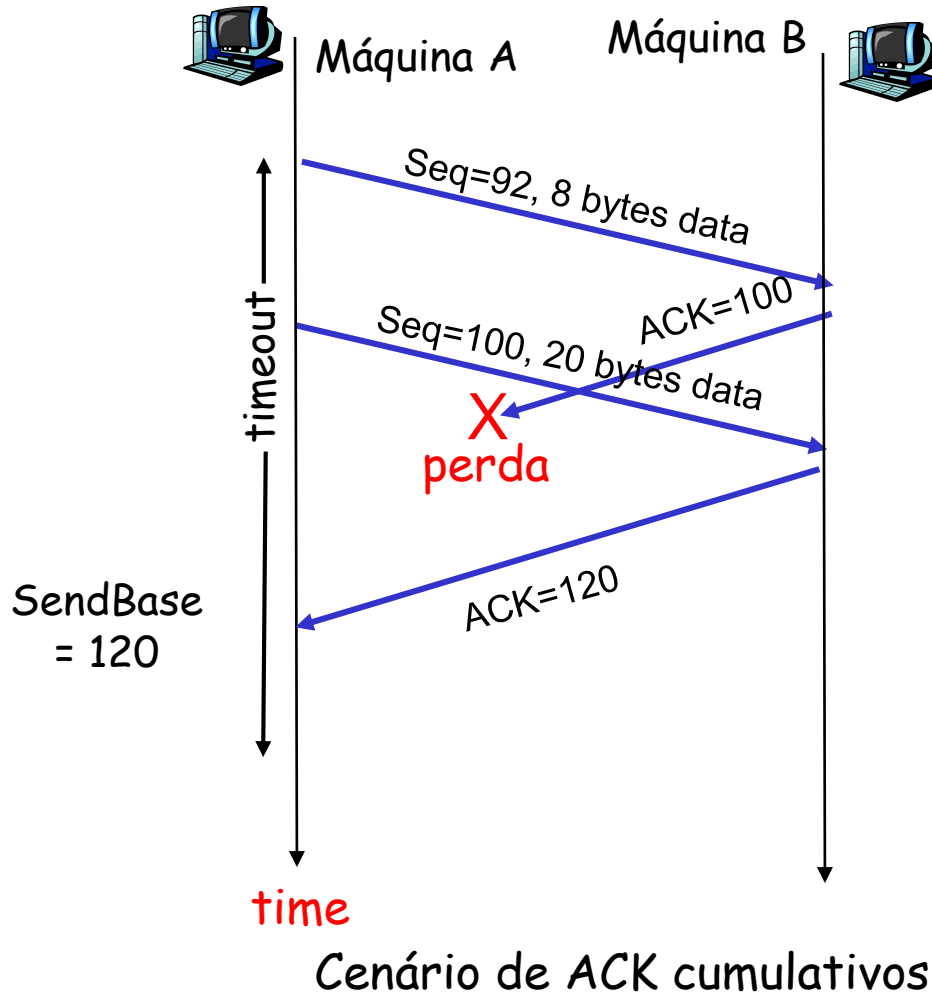
ACK recebido:

- ❑ Se confirma segmentos por confirmar
 - Actualizar o que foi confirmado
 - Arma temporizador se houver segmentos por confirmar

TCP: cenários de retransmissão



TCP: cenário de retransmissão



TCP: geração de ACK [RFC 1122, RFC 2581]

Eventos no receptor

Acção no receptor TCP

Chegada de segmentos na ordem,
com nº de sequência esperado.
Tudo para trás já confirmado

ACK atrasado. Espera máxima de 500 ms pelo próximo segmento. Se não chega o próximo segmento, envia ACK

Chegada de segmentos na ordem,
Con nº de seq. esperado.
Um segmento por confirmar.

Envia imediatamente um ACK cumulativo referente a todos os segmentos que chegaram na ordem

Chegada de segmento fora de ordem com nº seq. superior ao .
Esperado. Falta detectada

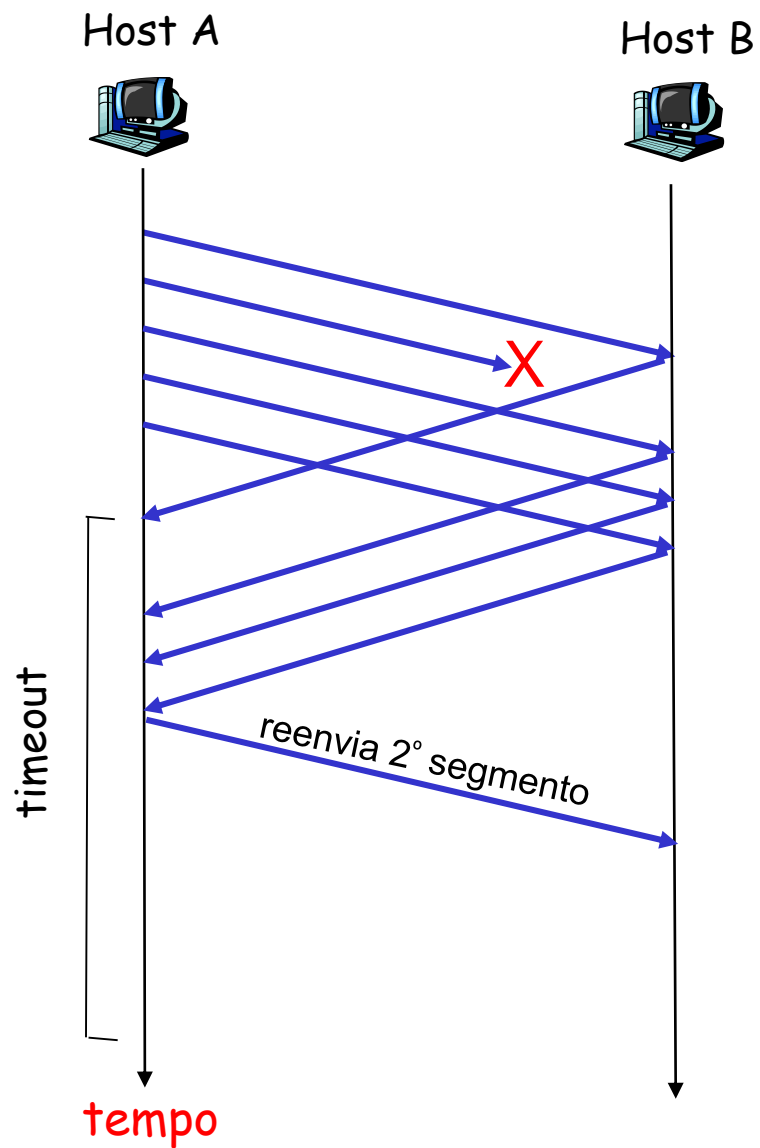
Envia ACK duplicado, indicando o nº de Seq. do próximo byte esperado

Chegada de segmento que preenche completa ou incompletamente a falha de nº seq

Envia ACK imediato se o segmento começa no limite inferior da falha

Retransmissão rápida

- ❑ Duração do temporizador (timeout) normalmente longa:
 - Atraso longo antes de reenviar o pacote perdido
- ❑ Detectar segmentos perdidos via ACKs.
 - Emissor envia frequentemente muitos segmentos seguidos
 - Se um segmento é perdido, provavelmente haverá muitos ACK duplicados.
- ❑ Se o emissor recebe 3 ACK para os mesmos dados, supõe que o segmento depois dos dados foi perdido:
 - fast retransmit: reenvia o segmento mesmo antes do temporizador expirar



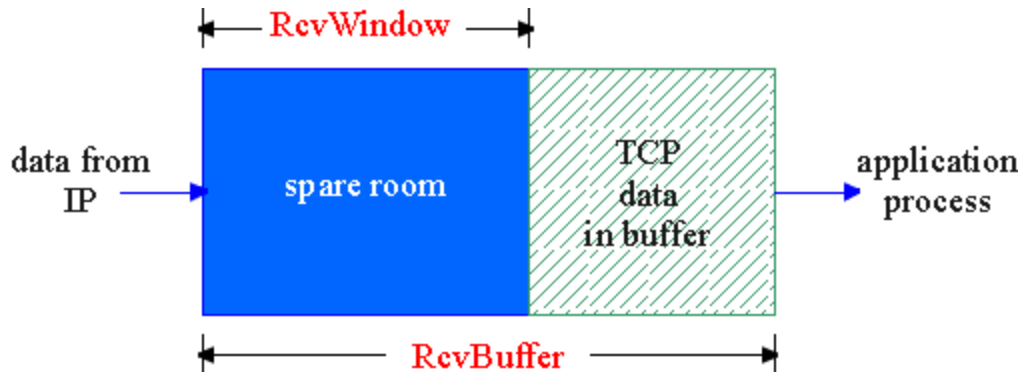
Reenvio de segmento depois de triplo ACK duplicados

Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - **Controlo de fluxo**
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

TCP: Controlo de fluxo

- ❑ O lado receptor da ligação tem um buffer de recepção:



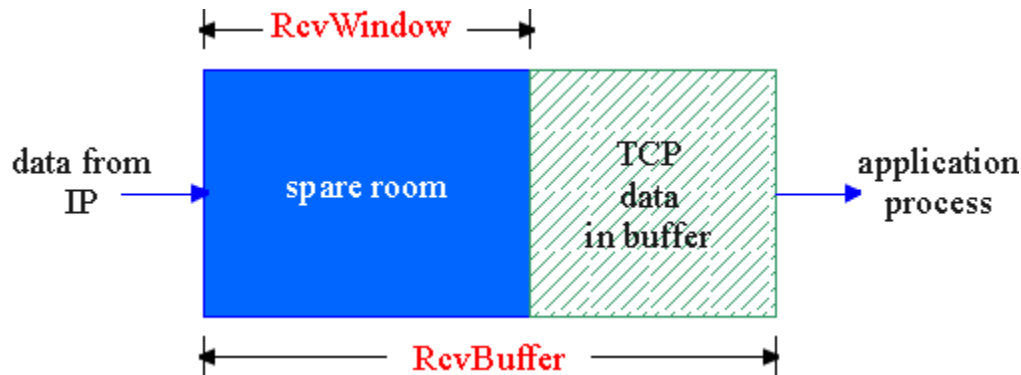
- ❑ O processo de aplicação pode ser lento a ler dados do buffer

Controlo de fluxo

Emissor não sobrecarrega o receptor por transmitir demasiado depressa

- ❑ Serviço de adaptação de velocidade: adapta o ritmo de transmissão

Controlo de fluxo: como funciona



(suponha que o receptor TCP descarta segmentos fora de ordem)

- ❑ Espaço livre no buffer
- = `RcvWindow`
- = `RcvBuffer - [LastByteRcvd - LastByteRead]`

- ❑ O receptor anuncia o espaço livre, incluindo o valor de `RcvWindow` nos segmentos
- ❑ O emissor limita os dados por confirmar a `RcvWindow`
 - Garante que o buffer de recepção não transborda
- ❑ Quando `RcvWindow=0` o emissor envia segmentos de 1 byte para obrigar o receptor a responder

Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - *Gestão de ligações*
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

TCP: gestão das ligações

Recordar: o emissor TCP estabelece uma ligação antes de iniciar a troca de segmentos de dados

❑ Iniciação das variáveis TCP:

- N° seq.
- buffers,
- informação de janela de controlo de fluxo(e.g. RcvWindow)

❑ *cliente:* inicia a ligação

```
Socket clientSocket = new  
Socket("hostname", "port  
number");
```

❑ *servidor:* contactado pelo cliente

```
Socket connectionSocket =  
welcomeSocket.accept();
```

Tripló handshake:

Passo 1: cliente envia segmento SYN=1 para servidor

- Define o n° de seq. inicial
- Não tem campo de dados

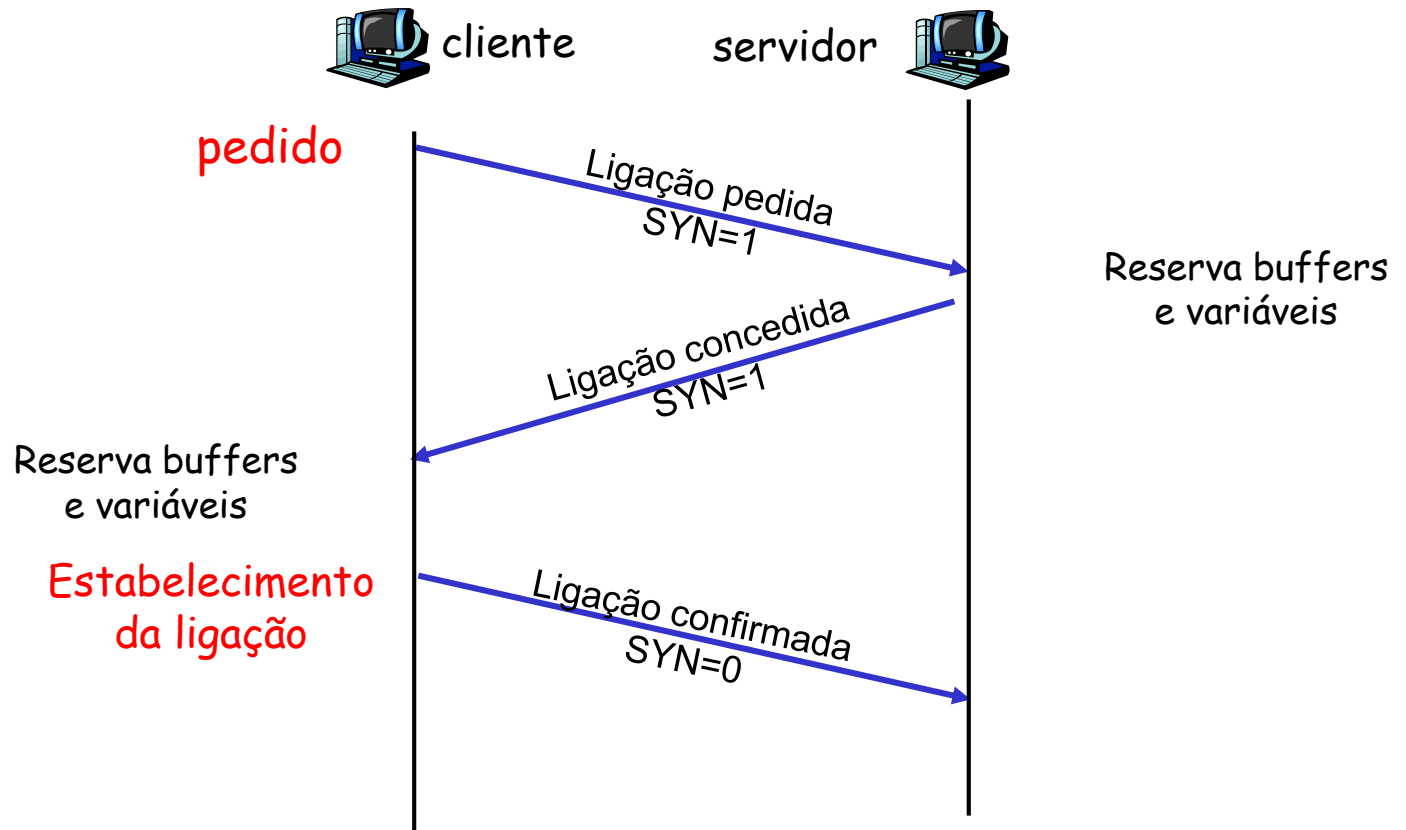
Passo 2: servidor recebe segmento de controlo SYN, responde com SYNACK

- servidor reserva buffers
- Especifica o seu n° seq inicial
- Confirma a recepção

Passo 3: cliente recebe SYNACK, responde com ACK que pode conter dados:

- Reserva buffer e variáveis
- Confirma a recepção

Gestão das ligações - estabelecimento



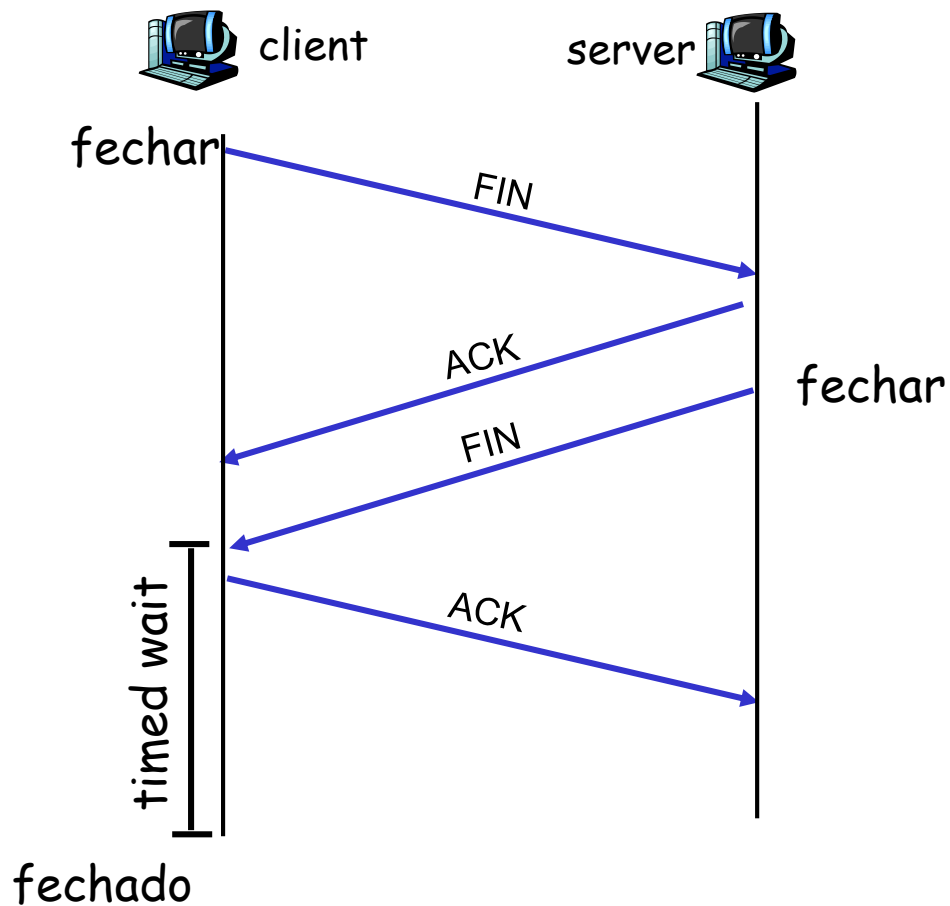
Gestão das ligações - fecho

Fechando uma ligação:

cliente fecha socket:
`clientSocket.close();`

Passo 1: cliente envia
segmento de controlo TCP
FIN para servidor

Passo 2: servidor recebe
FIN, responde com ACK.
Fecha ligação e envia FIN



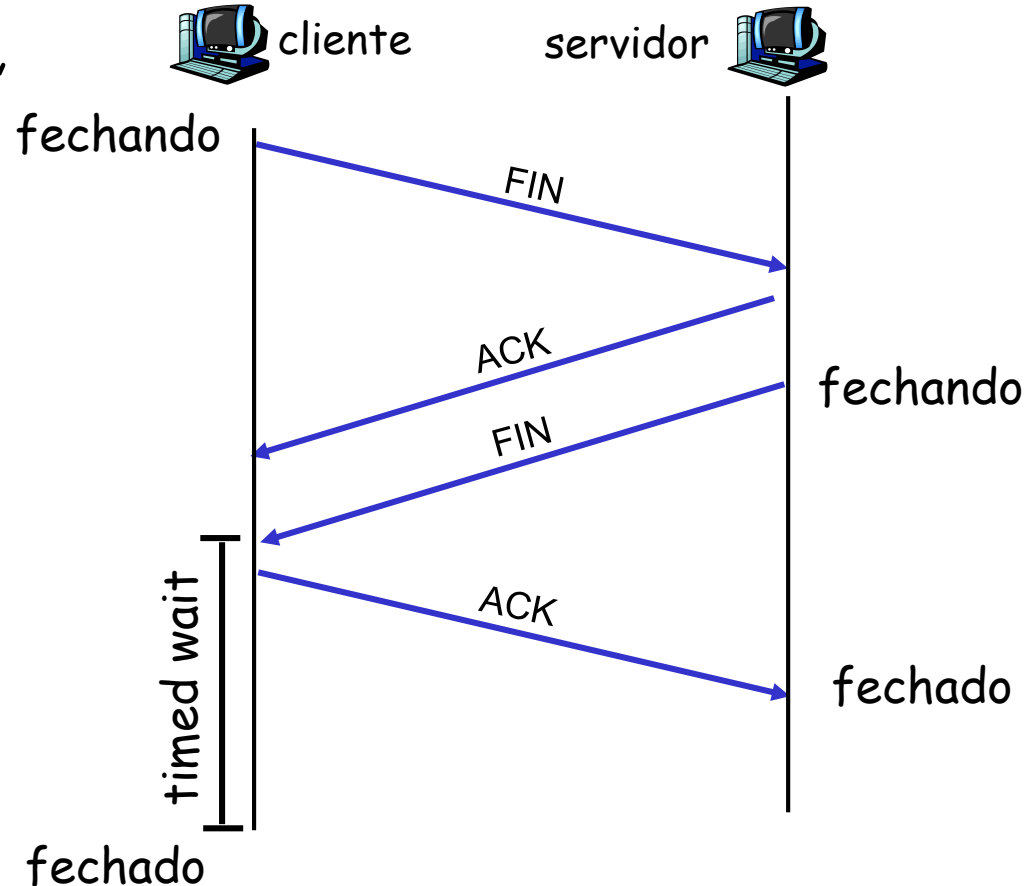
TCP: gestão das ligações - fecho

Passo 3: cliente recebe FIN, responde com ACK.

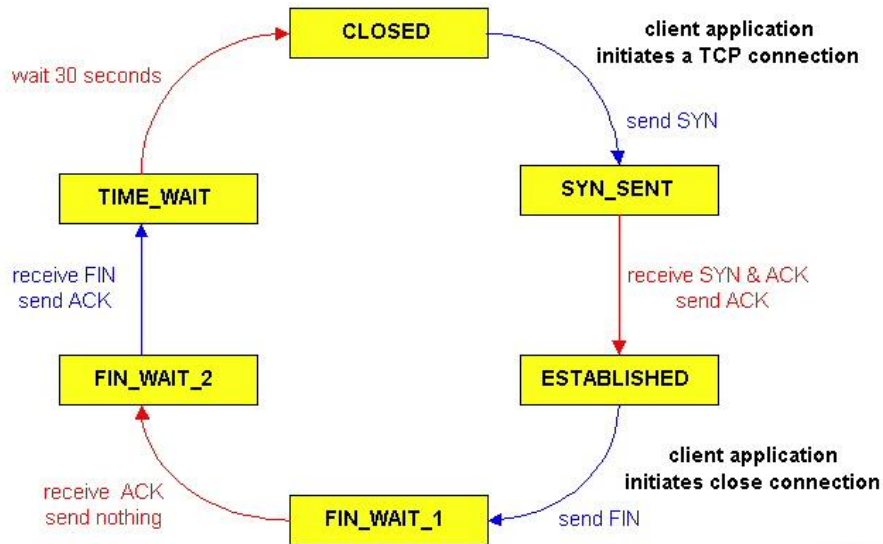
- Entra em "timed wait" - responderá com ACKs a FINs recebidos

Passo 4: servidor, recebe ACK. Ligação fechada.

Nota: com pequenas modificações, pode tratar FINs simultâneos.

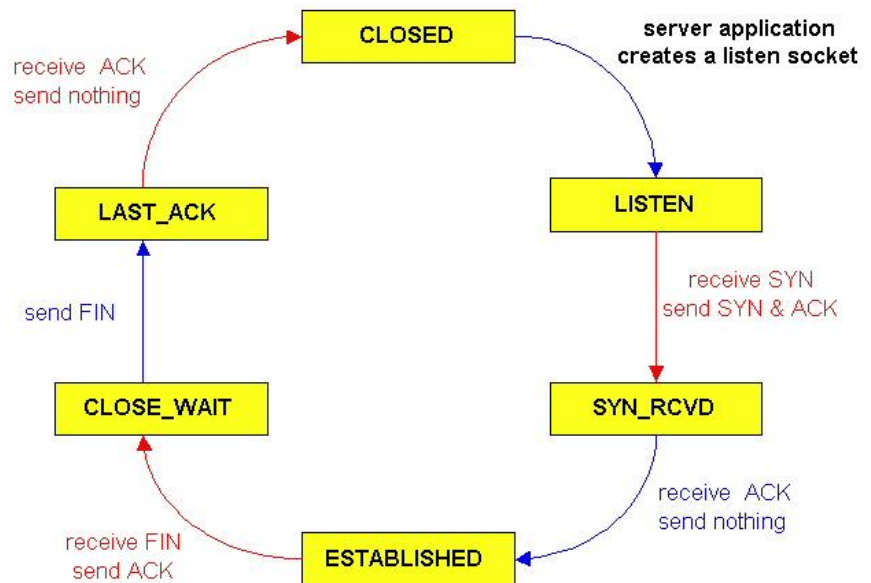


Gestão das ligações: ciclo de vida



Ciclo de vida do cliente TCP

Ciclo de vida do servidor TCP



Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

Princípios de controlo de congestão

Congestão:

❑ informalmente:

- “demasiadas fontes enviam demasiada informação a um ritmo demasiado superior ao que a rede é capaz de processar”

❑ Diferente do controlo de fluxo!

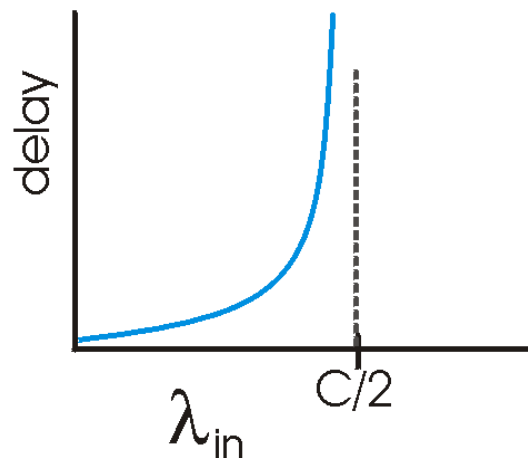
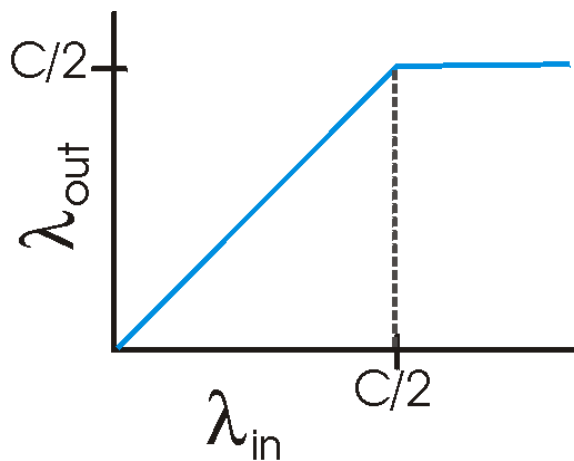
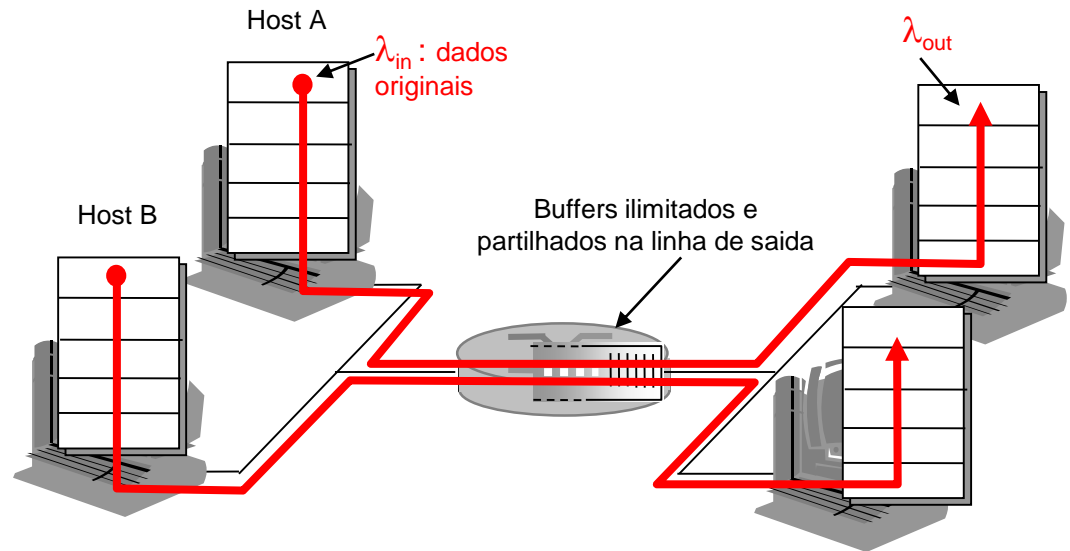
❑ Sintomas:

- **Perda de pacotes** (buffer overflow nos routers)
- **Atrasos elevados** (espera em fila nos buffers dos routers)

❑ Problema da lista dos top-10!

Causas e custos da congestão: cenário 1

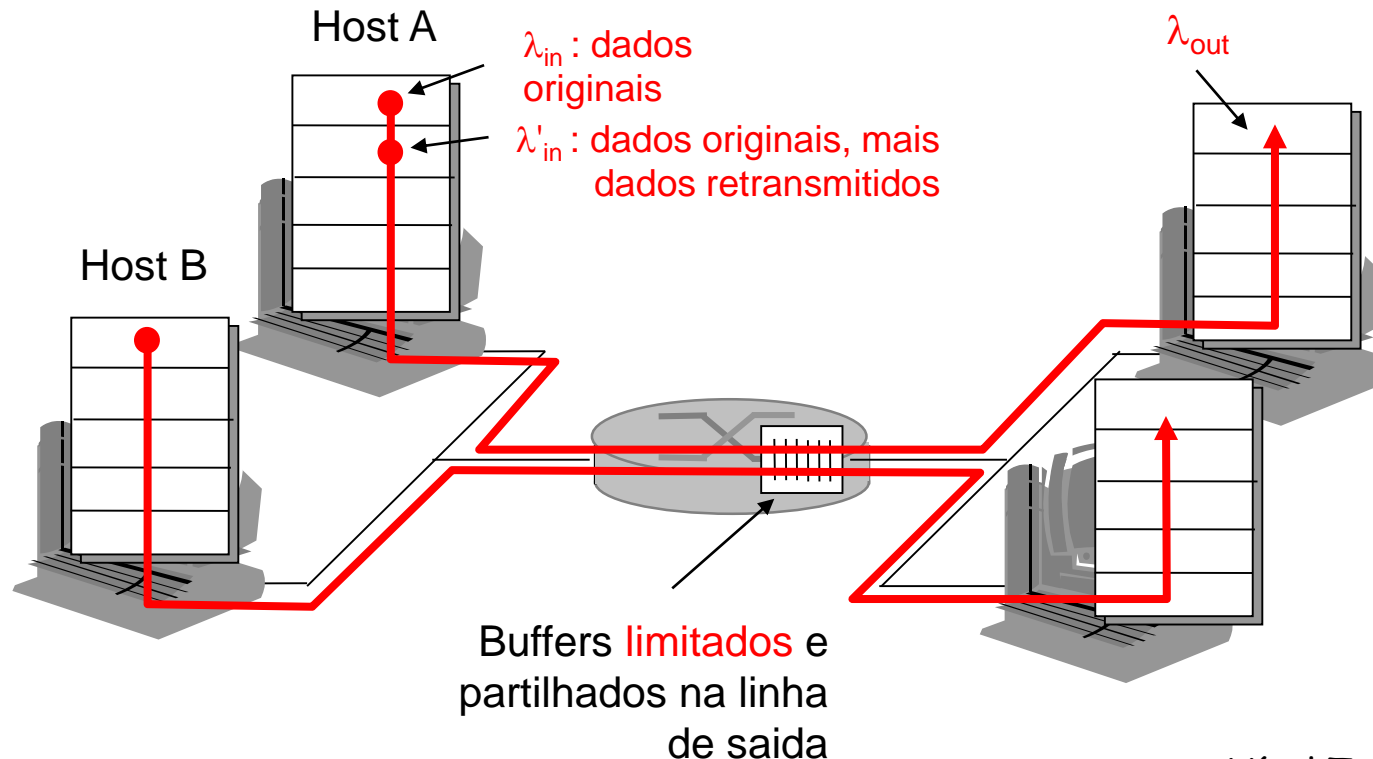
- ❑ Dois emissores, dois receptores
- ❑ Um router, com buffers infinitos
- ❑ Sem retransmissão
- ❑ Capacidade da linha C



- ❑ Atraso elevados quando congestionado
- ❑ Débito máximo atingível ($C/2$)

Causas e custos da congestão: cenário 2

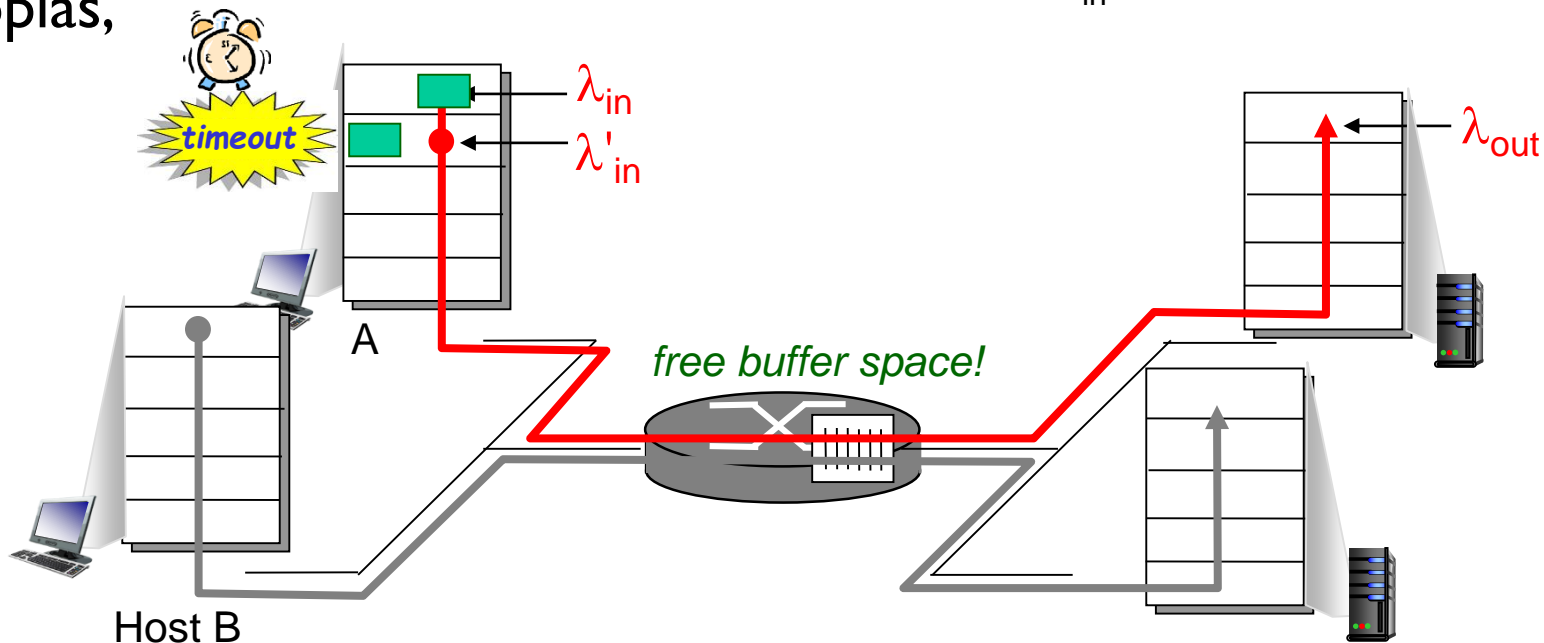
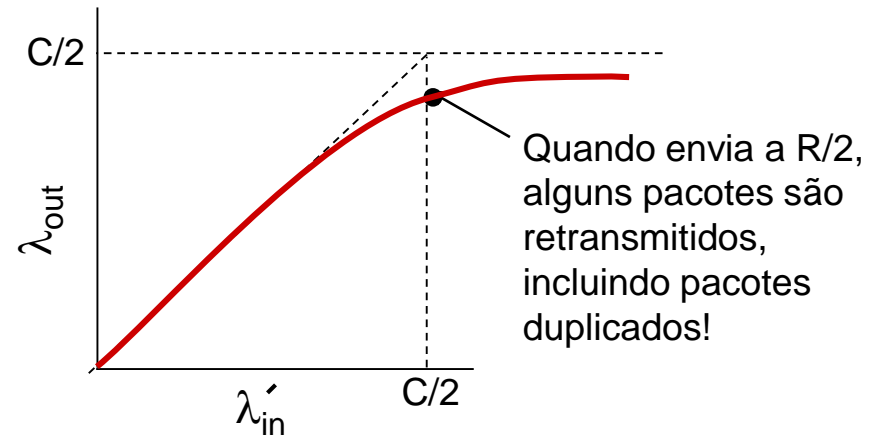
- ❑ um router, buffers *finito*
- ❑ Emissor retransmite pacote perdido



Causas/custos da congestão: cenário 2

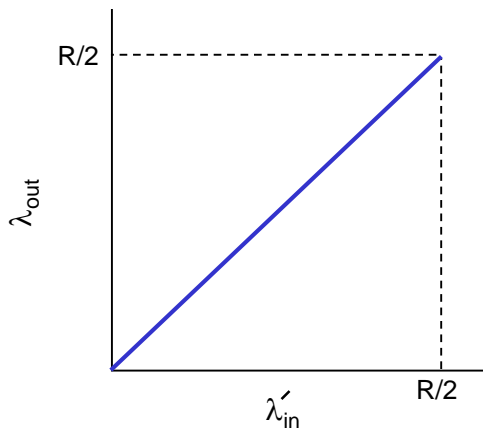
Realístico: *duplicação*

- ❖ Pacotes podem ser perdidos, descartados no router devido a buffers cheios
- ❖ Emissor declara times out prematuramente enviando *duas* cópias,

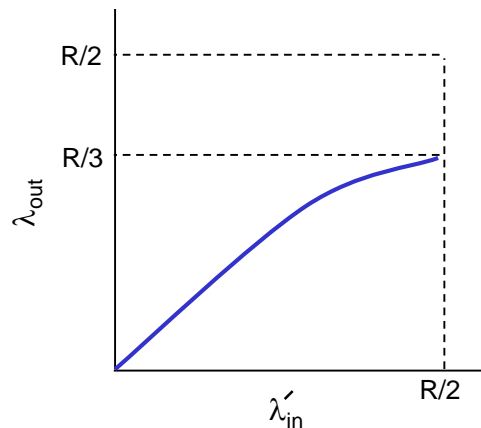


Causas e custos da congestão: cenário 2

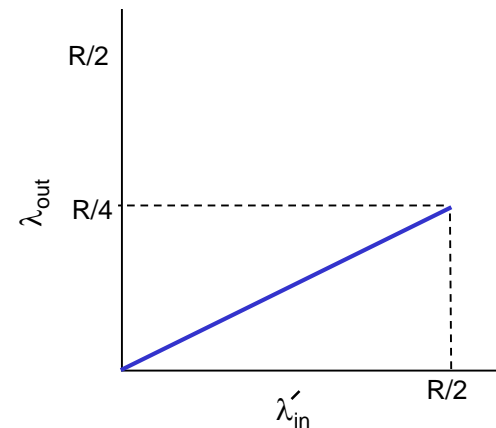
- ❑ Situação ideal: $\lambda_{in} = \lambda_{out}$ (goodput)
- ❑ Retransmissões ocorrem quando há perdas: $\lambda'_{in} > \lambda_{out}$
- ❑ Retransmissão de pacotes atrasados (não perdidos) faz λ'_{in} maior para o mesmo λ_{out}



a.



b.



c.

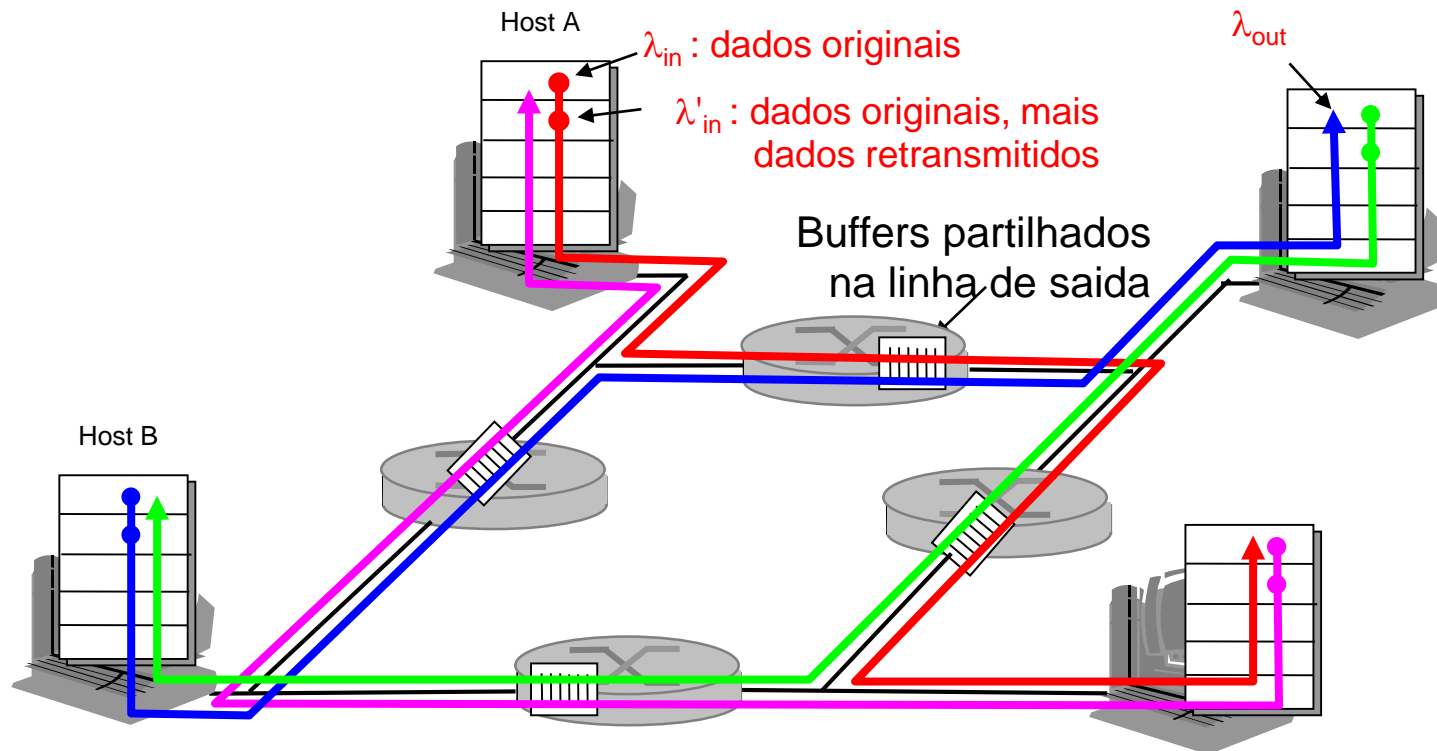
"custos" da congestão:

- ❑ Mais trabalho (retransmissões) para um determinado "goodput"
- ❑ Retransmissões desnecessárias: linha transporta múltiplas cópias do pacote

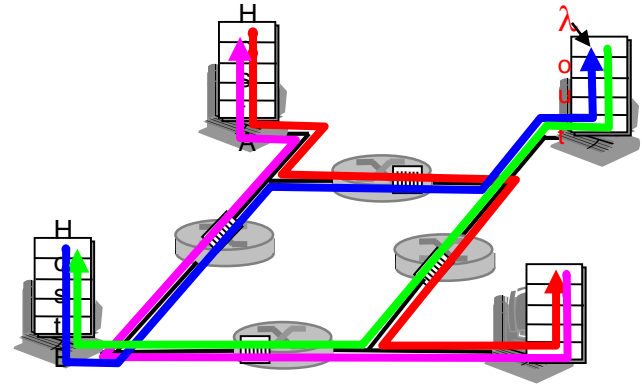
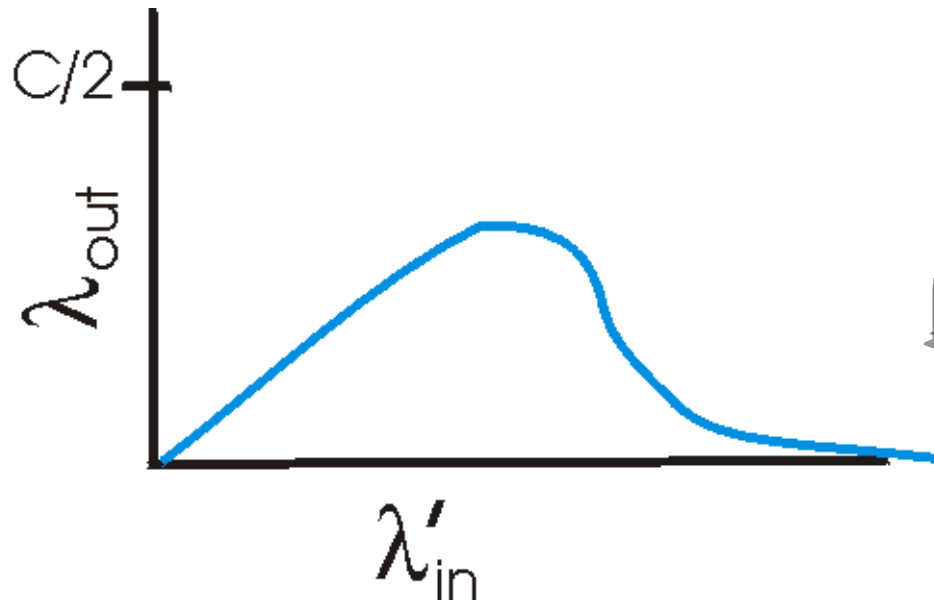
Causas e custos da congestão: cenário 3

- ❑ Quatro emissores
- ❑ Caminho com vários nós
- ❑ timeout/retransmissões

Q: O que acontece quando λ_{in} e λ'_{in} aumentam?



Causas e custos da congestão: cenário 3



Outro custo da congestão:

- Quando um pacote se perde, qualquer capacidade de transmissão que já tenha sido usado para o transmitir é perdida!

Abordagens ao controlo de congestão

Dois tipos de abordagem ao controlo de congestão:

Controlo de congestão extremo a extremo:

- ❑ Não há feedback explícito da rede
- ❑ Congestão inferida pela perda e atrasos
- ❑ Observada pelos sistemas terminais
- ❑ Aproximação do TCP

Controlo de gestão assistido pela rede:

- ❑ Routers fornecem feedback aos sistemas terminais
 - Um único bit indica a congestão (SNA, DECbit, TCP/IP ECN, ATM)
 - Envio explícito do ritmo a que o emissor pode enviar

Estudo de um caso: congestão de controlo ATM ABR

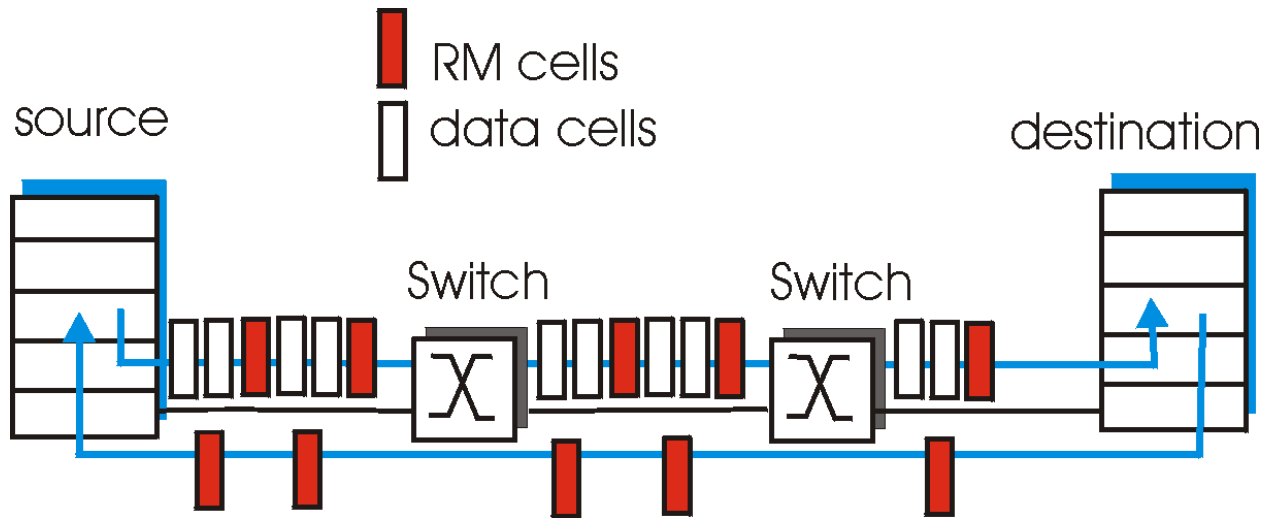
ABR: available bit rate:

- ❑ "serviço elástico"
- ❑ Se o caminho de emissor não estiver sobrecarregado:
 - emissor deve usar a largura de banda disponível
- ❑ Se o caminho do emissor estiver sobrecarregado:
 - Emissor envia apenas ao ritmo mínimo garantido

Células/pacotes RM (resource management):

- ❑ Enviadas pelo emissor, intercaladas com células de dados
- ❑ bits nas células RM activadas pelos switches (*"assistido pela rede"*)
 - **NI bit**: no increase - não aumentar o ritmo (congestão moderada)
 - **CI bit**: congestion indication - indicador de congestão
- ❑ Células RM devolvidas ao emissor pelo receptor com os bits intactos

Estudo de um caso: congestão de controlo ATM ABR



- ❑ Campo ER (Explicit Rate) de 2 bytes numa célula RM (Resource management)
 - O Switch congestionado pode diminuir o valor de ER da célula
 - Emissor envia ao ritmo mínimo suportado pelo caminho
- ❑ Bit EFCI nas células de dados: colocado a 1 no switch congestionado
 - Se uma célula de dados que precede a célula RM tem EFCI activo, o receptor activa o bit CI na célula RM devolvida

Sumário

- ❑ 3.1 Serviços do nível Transporte
- ❑ 3.2 Multiplexagem and desmultiplexagem
- ❑ 3.3 transporte sem ligação: UDP
- ❑ 3.4 princípios de transmissão de dados fiável
- ❑ 3.5 transporte com ligação: TCP
 - Estrutura dos segmentos
 - Transferência de dados fiável
 - Controlo de fluxo
 - Gestão de ligações
- ❑ 3.6 princípios de controlo de congestão
- ❑ 3.7 controlo de congestão em TCP

Controlo de congestão no TCP

- ❑ Controlo extremo a extremo (não assistido pela rede)
- ❑ Emissor limita a transmissão:

$\text{LastByteSent} - \text{LastByteAcked} < \text{CongWin}$

$$\text{ritmo} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/seg}$$

- ❑ CongWin é dinâmico, função da congestão da rede

Como o emissor detecta a congestão?

- ❑ Evento de perda = timeout ou 3 ACKs duplicados
- ❑ Emissor TCP reduz ritmo (CongWin) após evento de perda

Três mecanismos:

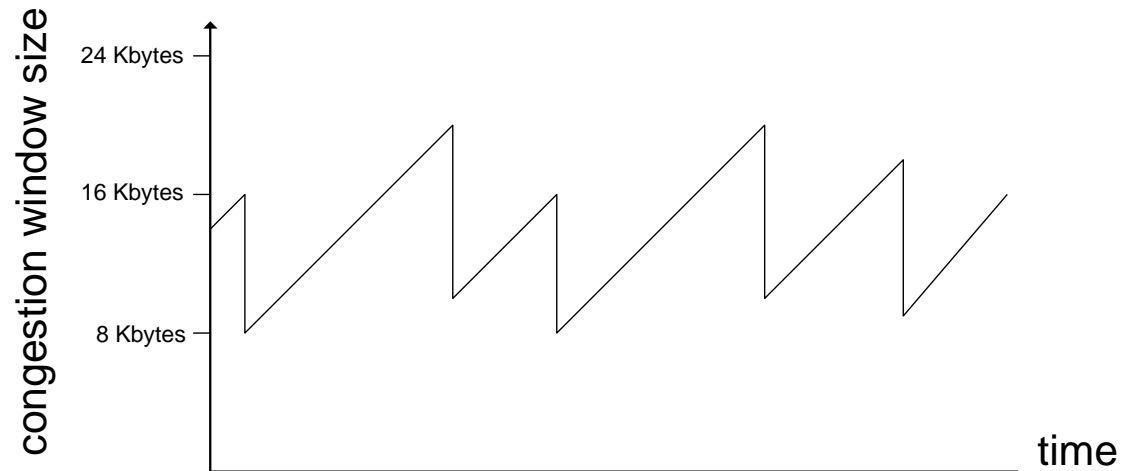
- AIMD
- Arranque lento (Slow Start)
- Conservativo após eventos de timeout

Congestão de controlo AIMD

additive increase, multiplicative decrease

- *aproximação*: aumenta ritmo de transmissão (window size), à procura de largura de banda, até ocorrerem perdas
 - *additive increase*: aumenta **CongWin** por 1 MSS a cada RTT até detectar perdas
 - *multiplicative decrease*: reduz **CongWin** para metade após perdas

Comportamento em
dente de serra:
À procura de
largura de banda

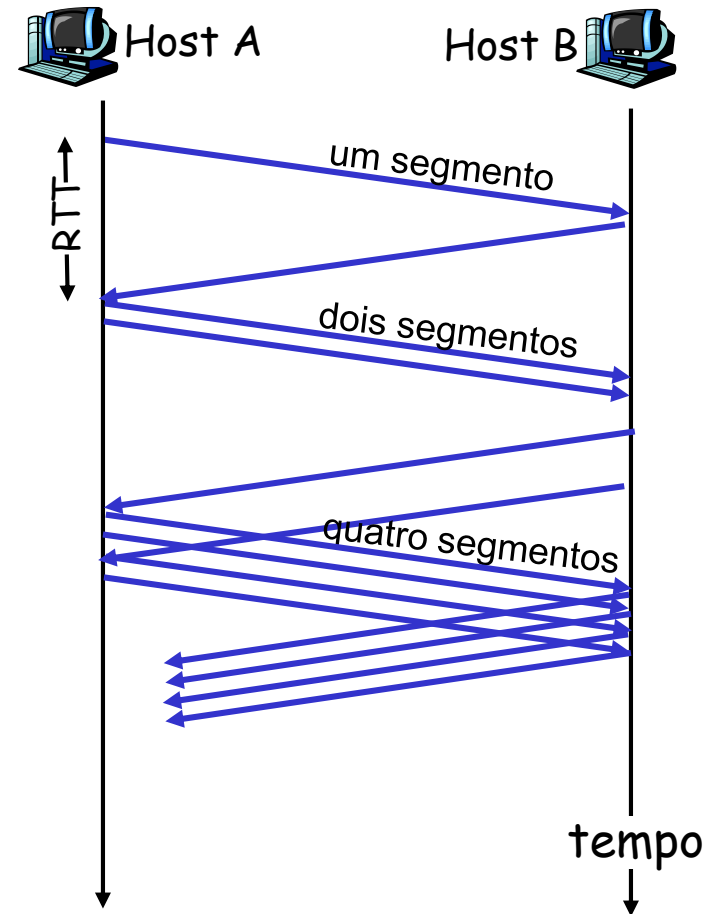


TCP: arranque lento (Slow Start)

- ❑ Quando a ligação começa, $\text{CongWin} = 1 \text{ MSS}$
 - Exemplo: $\text{MSS} = 500 \text{ bytes}$ & $\text{RTT} = 200 \text{ msec}$
 - Ritmo inicial = 20 kbps
- ❑ O ritmo possível pode ser $\gg \text{MSS}/\text{RTT}$
 - Vantajoso aumentar rapidamente o ritmo para um valor respeitável
- ❑ Quando a ligação começa, aumenta o ritmo com rapidez exponencial até ao primeiro evento de perda

TCP: Arranque lento (mais)

- ❑ Quando a ligação começa, aumenta o ritmo exponencialmente até ao primeiro evento de perda:
 - Duplica CongWin a cada RTT
 - Feito incrementando CongWin por cada ACK recebido
- ❑ Sumário: ritmo inicial lento, cresce exponencialmente



Refinamento: inferindo perdas

- ❑ Após 3 ACKs duplicados:
 - CongWin é reduzido a metade
 - Janela passa a crescer linearmente
- ❑ Mas após timeout:
 - Agora CongWin colocado a 1 MSS;
 - A janela passa a crescer exponencialmente
 - Até um limiar, depois cresce linearmente

Filosofia:

- ❑ 3 ACKs duplicados indicam que a rede é capaz de entregar alguns pacotes
- ❑ timeout antes de 3 ACKs duplicados é mais alarmante

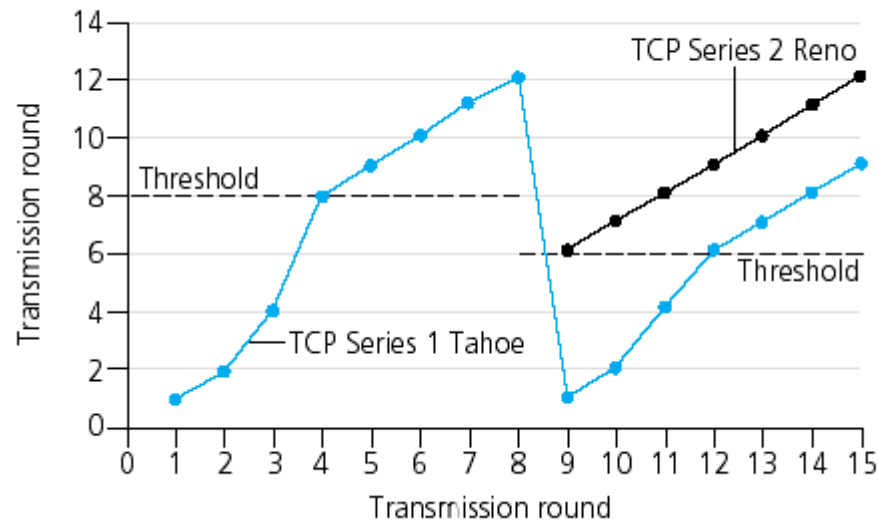
Refinamento

Q: quando deve o crescimento exponencial mudar para linear

A: Quando CongWin chegar a metade do seu valor antes do timeout.

Implementação:

- Limiar variável
- Num evento de perda, o limiar é colocado a metade de CongWin do valor que tinha antes da perda



Sumário: controlo de congestão TCP

- ❑ Quando CongWin está abaixo de Threshold (limiar), o emissor está na fase **slow-start**, a janela cresce exponencialmente
- ❑ Quando CongWin está acima do Threshold, o emissor está na fase **congestion-avoidance**, janela cresce linearmente
- ❑ Quando ocorre um **triplo ACK duplicado**, Threshold é colocado a $\text{CongWin}/2$ e CongWin é colocado a Threshold.
- ❑ Quando ocorre um **timeout**, Threshold é colocado a $\text{CongWin}/2$ e CongWin é colocado a 1 MSS.

Controlo de congestão do emissor

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	$\text{CongWin} = \text{CongWin} + \text{MSS}$, If ($\text{CongWin} > \text{Threshold}$) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	$\text{CongWin} = \text{CongWin} + \text{MSS} * (\text{MSS} / \text{CongWin})$	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	$\text{Threshold} = \text{CongWin} / 2$, $\text{CongWin} = \text{Threshold}$, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	$\text{Threshold} = \text{CongWin} / 2$, $\text{CongWin} = 1 \text{ MSS}$, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

Futuro TCP: TCP sobre "long, fat pipes"

- ❑ exemplo: segmentos de 1500 byte, 100ms RTT, quer desempenho de 10 Gbps
- ❑ requiere $W = 83,333$ segmentos "in-flight"
- ❑ Desempenho em termos da probabilidade de perdas, L [Mathis 1997]:

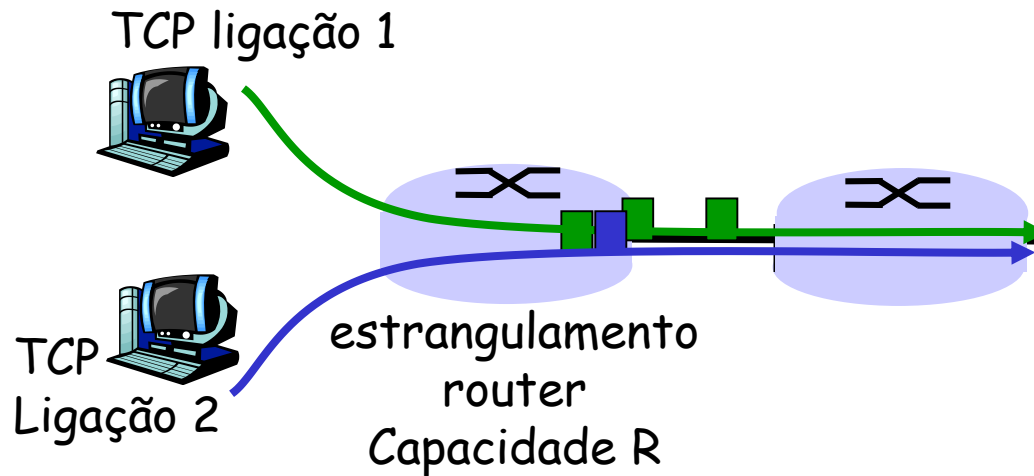
$$\text{Desempenho TCP} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

→ para obter desempenho de 10 Gbps, necesssita uma taxa de erro de $L = 2 \cdot 10^{-10}$
- uma taxa muito pequena!

- ❑ novas versões de TCP para alta velocidade

Justiça TCP

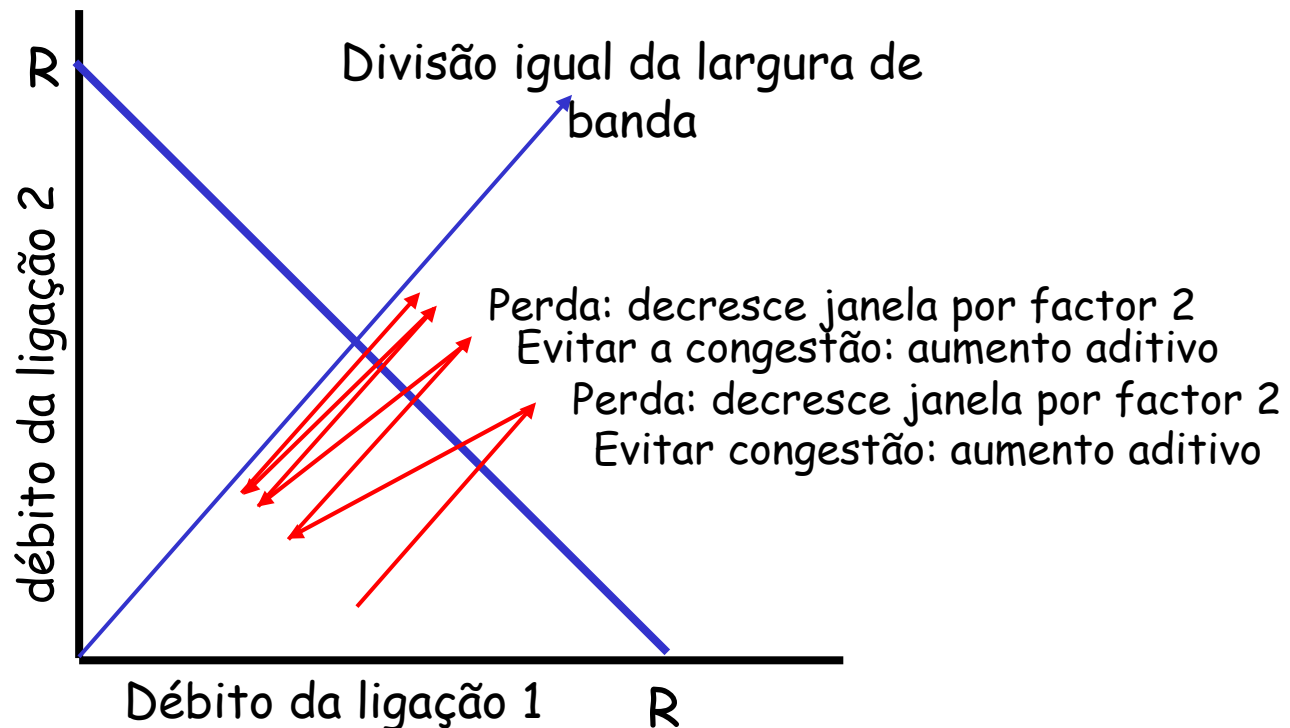
Objectivos da justiça: se K sessões TCP partilham um estrangulamento por uma mesma linha de ritmo R , cada uma deve obter um ritmo médio de R/K



Porque é que o TCP é justo?

Duas sessões em competição:

- Aumento aditivo origina um declive de 1, quando o débito aumenta
- Decréscimo multiplicativo decresce débito proporcionalmente



Justiça

Justiça e UDP

- ❑ As aplicações multimédia normalmente não usam TCP
 - Não querem o ritmo estrangulado pelo controlo de congestão
- ❑ Usam UDP:
 - Envia áudio/vídeo a ritmo constante, toleram perdas de pacotes

Justiça e ligações TCP paralelas

- ❑ Nada impede as aplicações de abrirem ligações paralelas para o mesmo destino.
- ❑ Web browsers fazem isto
- ❑ Exemplo: linha de ritmo R suportando 9 ligações;
 - Nova aplicação pede 1 ligação TCP, obtém ritmo $R/10$
 - Nova aplicação pede 11 ligações TCPs, obtém ritmo de $R/2$!

Sumário

- ❑ Princípios do serviço da camada de transporte:
 - multiplexagem, desmultiplexagem
 - Transferência fiável de dados
 - Controlo de fluxo
 - Controlo de congestão
- ❑ Instanciação e implementação na Internet
 - UDP
 - TCP

A seguir:

- ❑ Deixar a periferia da rede (aplicações, nível de transporte)
- ❑ Entrar no núcleo da rede