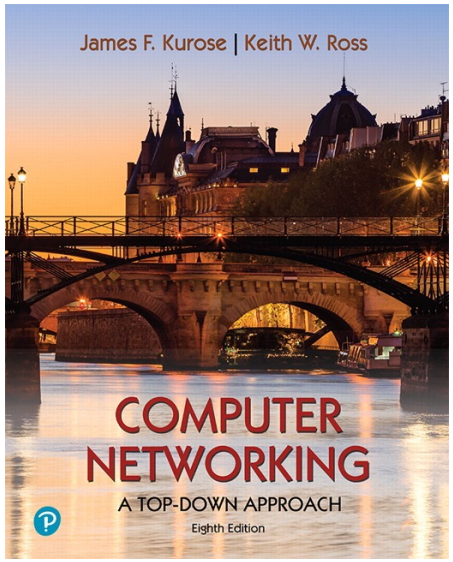
Nível de Transporte

Comunicações por Computador

Licenciatura em Engenharia Informática

3º ano/2º semestre

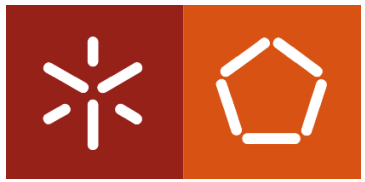
2025/2026



***Computer Networking: A Top Down Approach*, Capítulo 3**

Jim Kurose, Keith Ross, Addison-Wesley ©2021 .

Nível de Transporte

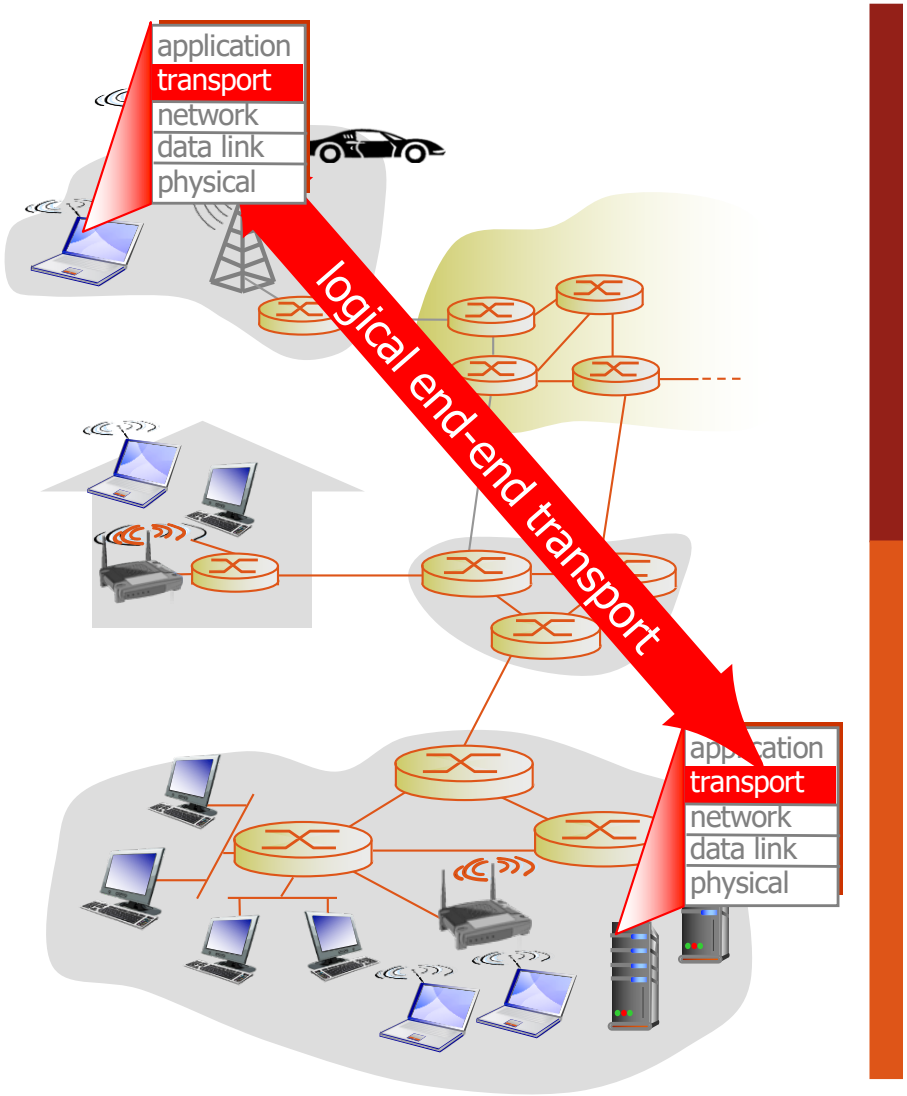


Objetivos:

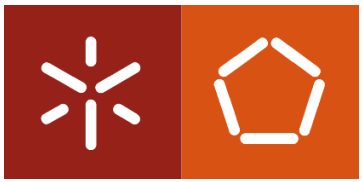
- **Compreender os princípios subjacentes aos serviços de camada de transporte:**
  - Transferência fiável de dados
  - Controlo de fluxo
  - Controlo de congestão
- **Conhecer os protocolos da camada de transporte da Internet**
  - UDP: transporte não orientado à conexão
  - TCP: transporte confiável e orientado à conexão
  - Controlo de congestão TCP

Serviço e Protocolos de Transporte

- Disponibiliza uma **ligação lógica entre aplicações** (*processos*) que estão a ser executadas em Sistemas Terminais diferentes
- Os protocolos de transporte são executados nos Sistemas Terminais
  - O emissor: parte a mensagem gerada pela aplicação em **segmentos** que passa à camada de rede
  - O recetor: junta os diferentes **segmentos** que constituem uma mensagem que passa à respetiva aplicação
  - Internet: TCP e UDP



Transporte *versus* Rede



- **Camada de Rede:**
  - fornece uma ligação lógica entre dois *sistemas terminais*
- **Camada de Transporte:**
  - fornece uma comunicação lógica entre *processos*

- Usa e melhora os serviços disponibilizados pela camada de Rede
- Troca de dados **fiável e ordenada** (TCP)
  - Controlo de **Fluxo**, Estabelecimento da Ligação
  - Controlo de **erros**
  - Controlo de **congestão**
- Troca de dados **não fiável e desordenada** (UDP)
- Serviços não disponíveis: garantia de atraso máximo e largura de banda mínima

Transporte

- **Discussão:**
  - É mesmo necessário termos uma camada de transporte?
  - Tudo o que a camada de transporte faz não pode ser feito pelas aplicações?
  - Não é possível desenvolver aplicações diretamente sobre o protocolo de rede IP?
  - Não é possível desenvolver aplicações diretamente sobre a camada lógica (MAC)?

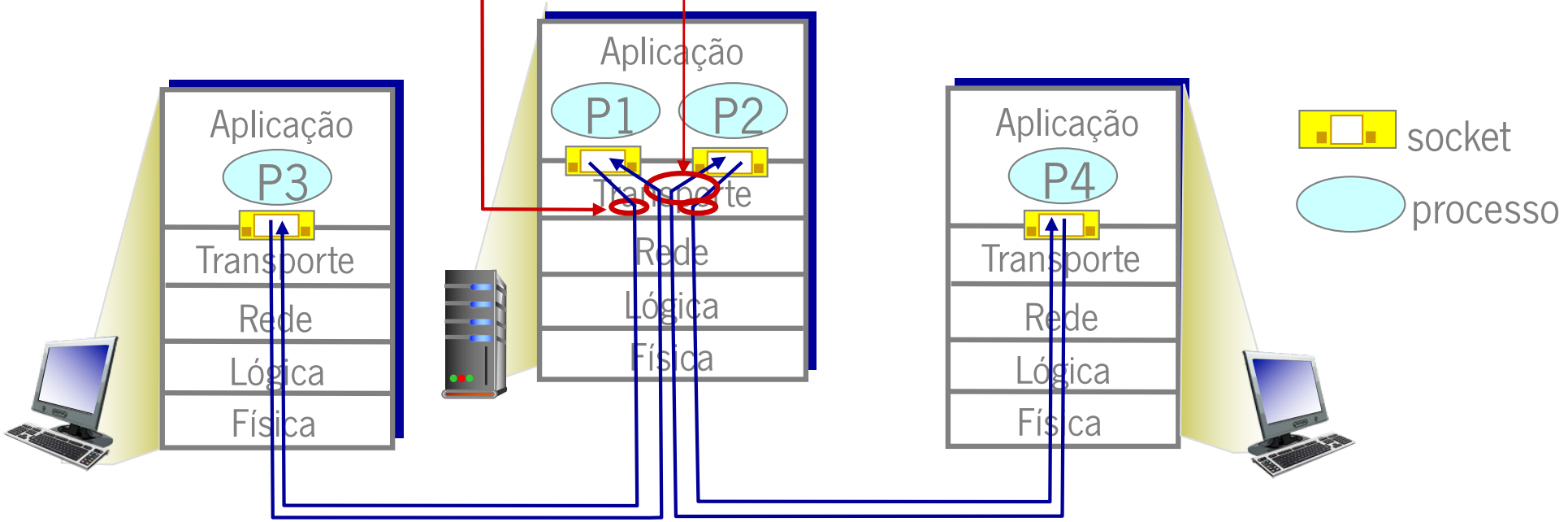
Multiplexagem / Desmultiplexagem

Multiplexagem no emissor:

Recolher os dados de diferentes *sockets* e delimitá-los com os respetivos cabeçalhos construindo os respetivos segmentos

Desmultiplexagem no recetor:

Entregar os diferentes segmentos ao *socket* correcto.

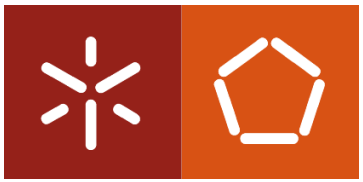


Desmultiplexagem

- **É efetuada pelo sistema terminal destino ao receber um *datagrama* IP**
  - Cada *datagrama* contém um segmento TCP ou UDP
  - Cada segmento possui a identificação da porta de origem e da porta destino.
  - O sistema terminal usa os **endereços IP** e os **números de porta** para encaminhar o segmento para o *socket* correto

Formato dos segmentos TCP/UDP

Desmultiplexagem
– *não orientado à conexão*



- **As aplicações criam um *socket* … e limitam-se a enviar datagramas para IP Destino, Porta destino**

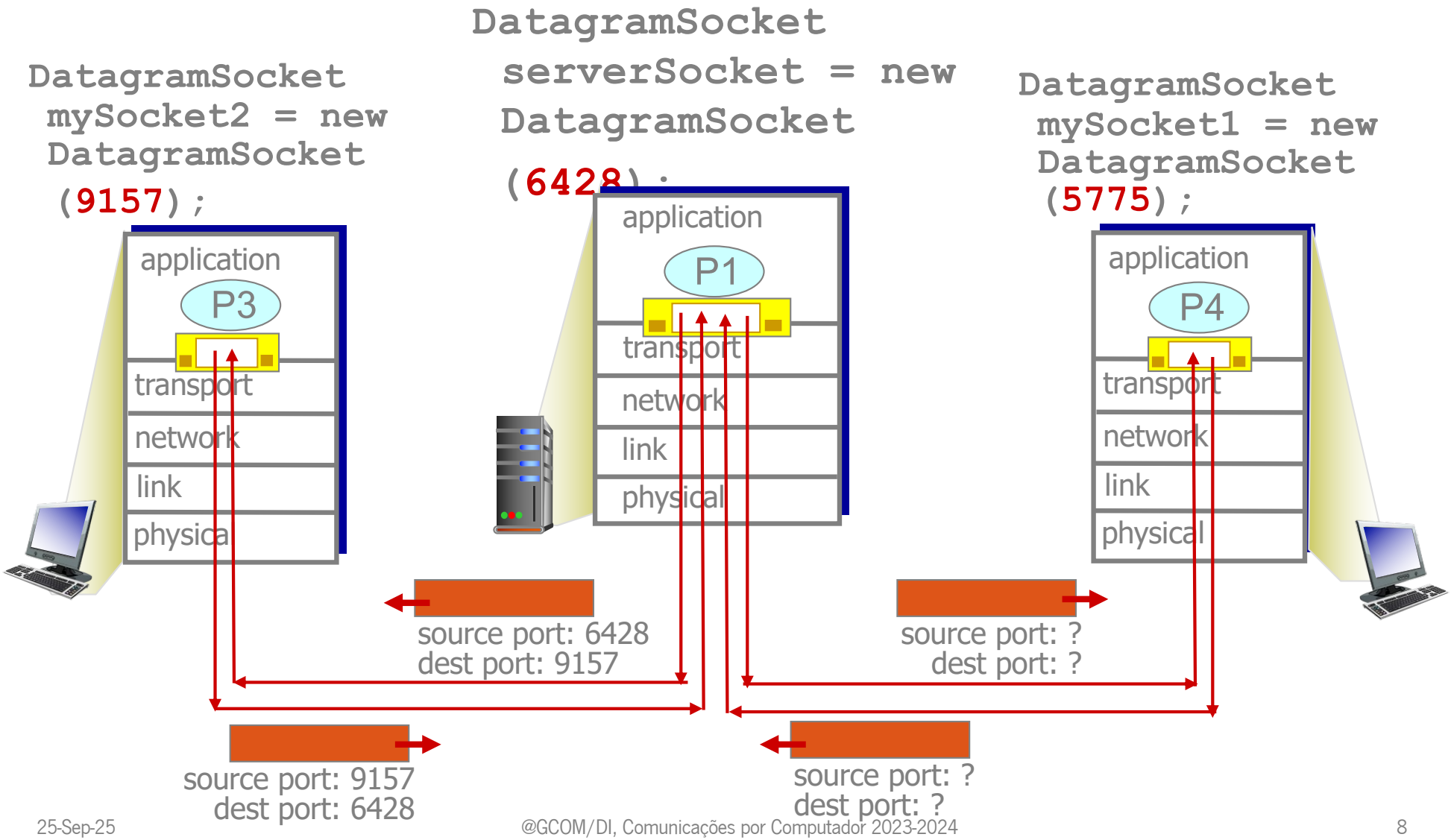
```
DatagramSocket s= new DatagramSocket();
DatagramPacket p = new DatagramPacket(aEnviar, aEnviar.length, IPAddress, 9999);
 s.send(p);
```

- **Quando o host recebe um segmento UDP:**
  - Verifica a ***porta#*** destino do segmento
  - direciona o segmento UDP para o *socket* com essa ***porta#***

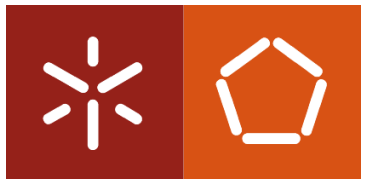
Datagramas IP com o mesmo *IP Destino, Porta destino,* mas com diferentes IP de origem e/ou portas de origem são dirigidos ao ***mesmo socket*** no destino!

Desmultiplexagem

– *não orientado à conexão*



Desmultiplexagem
– *orientado à conexão*



- **As aplicações criam um *socket* e <u>uma conexão com servidor</u> destino para enviar dados**

Opcionais!

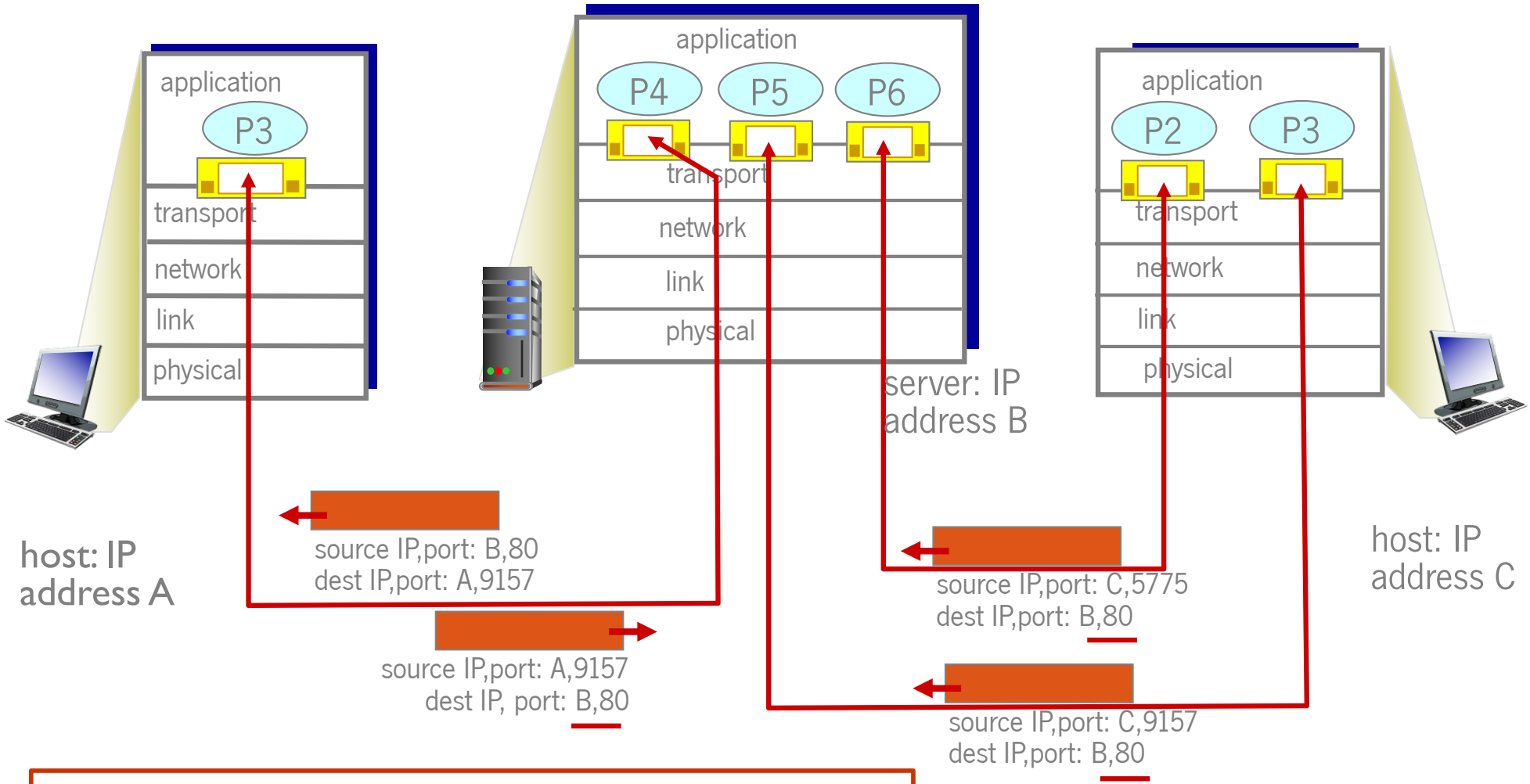
```
Socket socketCliente = new Socket(IPDestino, portaDestino, IPLocal, portaLocal);
```

- **Socket TCP identifica-se com 4 items:**
  - endereço IP origem
  - nº porta origem
  - endereço IP destino
  - nº porta destino

Recetor usa sempre **os 4 valores** para redirecionar para o *socket* correto!

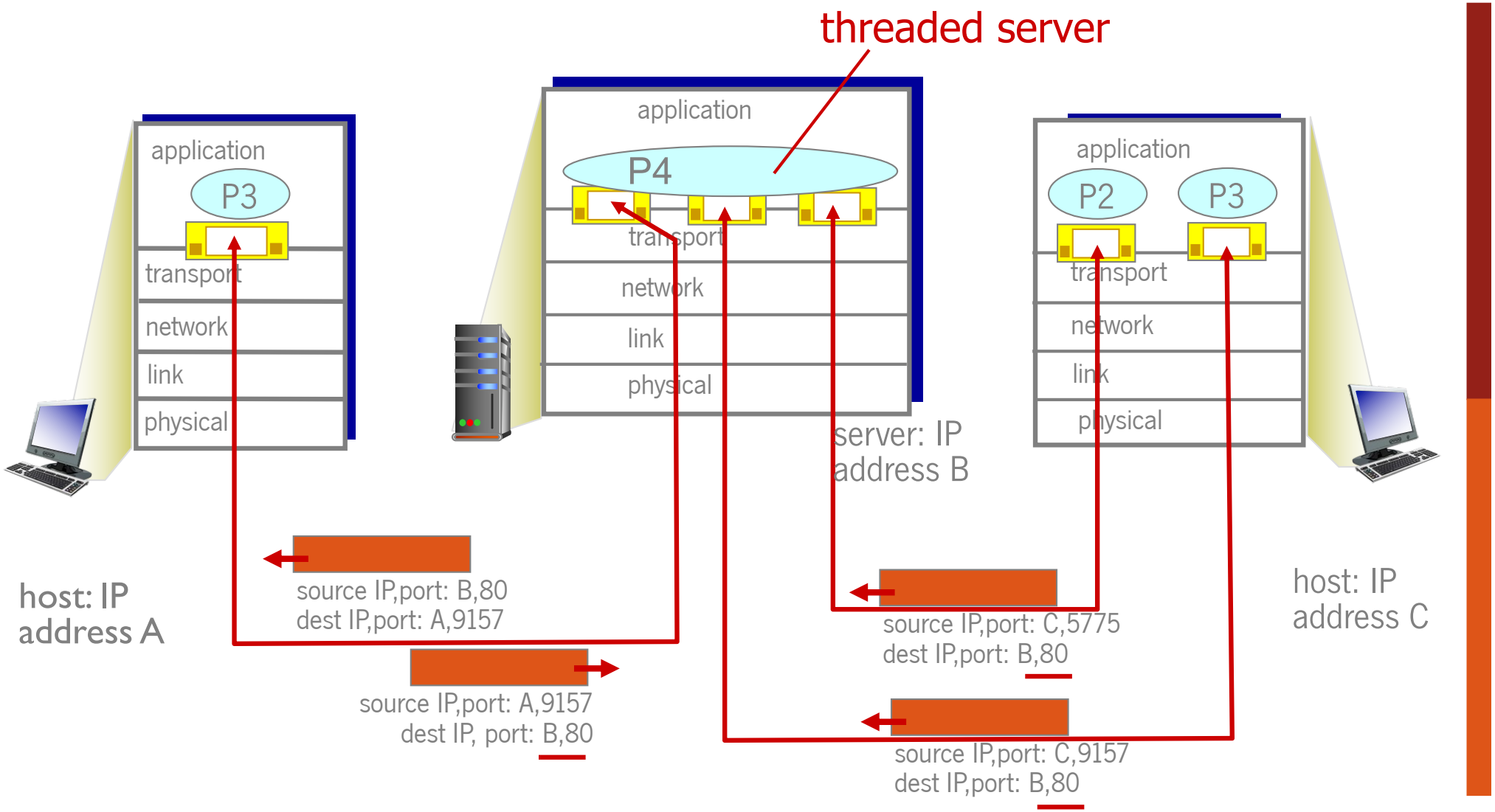
→ Servidor pode ter várias conexões TCP distintas em simultâneo, com uma **<u>socket distinto</u>** para cada uma delas!

Desmultiplexagem
– *orientado à conexão*



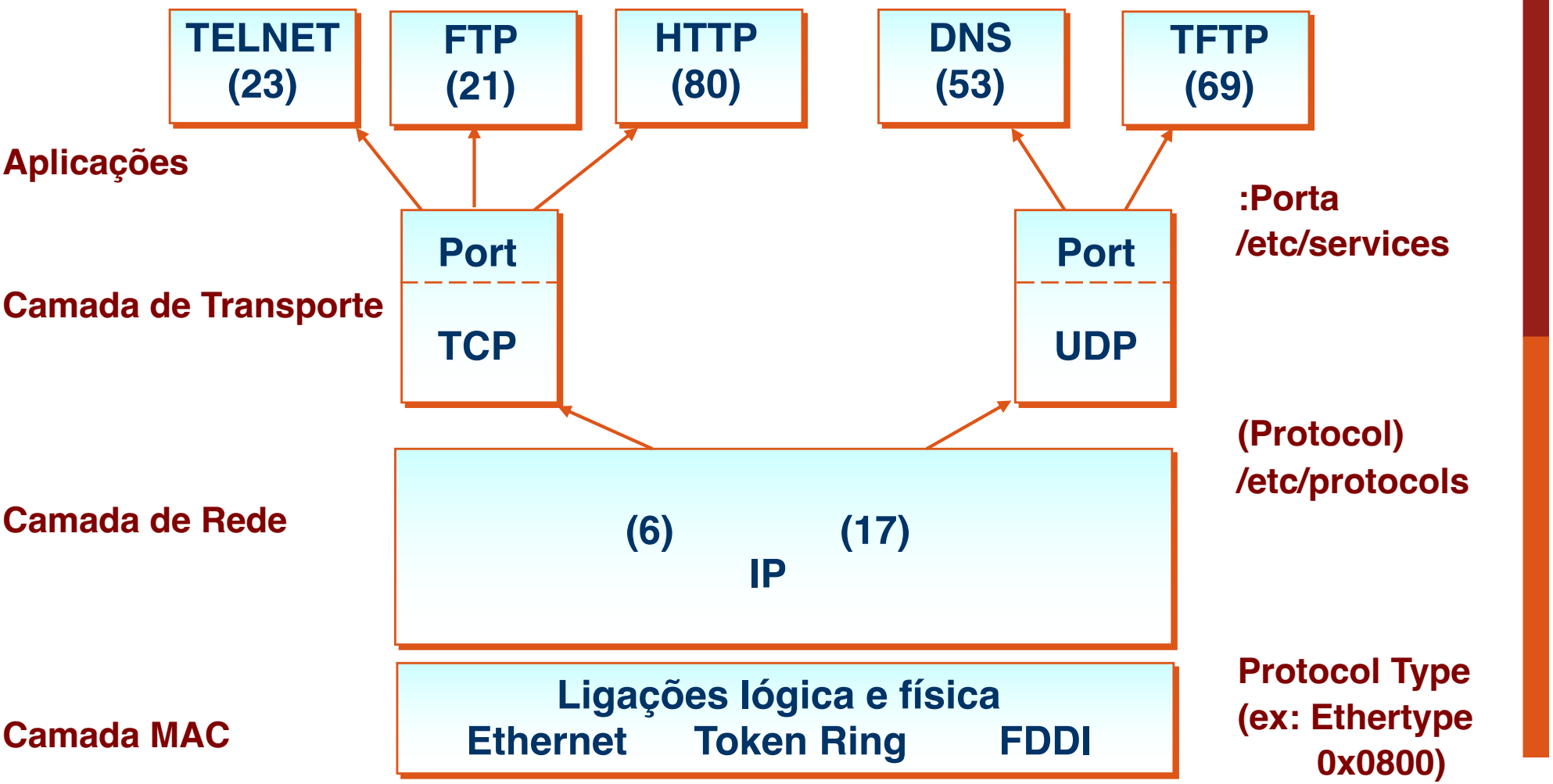
Três segmentos, todos destinados ao endereço destino: B, porta de destino: 80 são desmultiplexados em sockets *diferentes*

Desmultiplexagem
– *orientado à conexão*



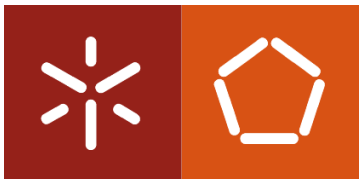
TCP/IP

Protocolos de Transporte: UDP e TCP



TCP/IP

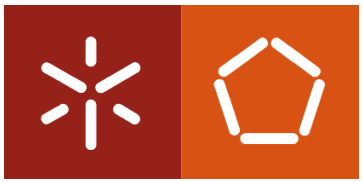
UDP - *User Datagram Protocol*



- **Funções do User Datagram Protocol**
  - protocolo de transporte fim-a-fim, não fiável
  - orientado ao datagrama (sem conexão)
  - actua como uma interface da aplicação com o IP para multiplexar e desmultiplexar tráfego
  - usa o conceito de porta / número de porta
    - forma de direccionar datagramas IP para o nível superior
    - portas reservadas: 0 a 1023, dinâmicas: 1024 a 65535
  - é utilizado em situações que não justificam o TCP
    - exemplos: TFTP, RPC, DNS
  - ... ou quando as aplicações querem controlar o fluxo de dados e gerir erros de transmissão diretamente

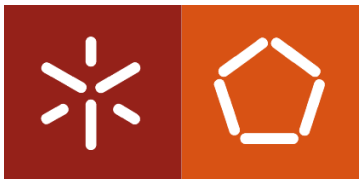
UDP

Desmuliplexagem



- O *socket* UDP é identificado através de dois números: endereço IP destino, e número de porta destino
- Quando um Sistema Terminal recebe um segmento UDP verifica qual o número da porta destino que consta do segmento UDP e redireciona o segmento para o *socket* com esse número de porta
- *Datagramas* com diferentes endereços IP origem e/ou portas origem podem ser redirecionados para o mesmo *socket*

UDP Sockets



- **Criar o socket:**

```
DatagramSocket s= new DatagramSocket(9876);
```

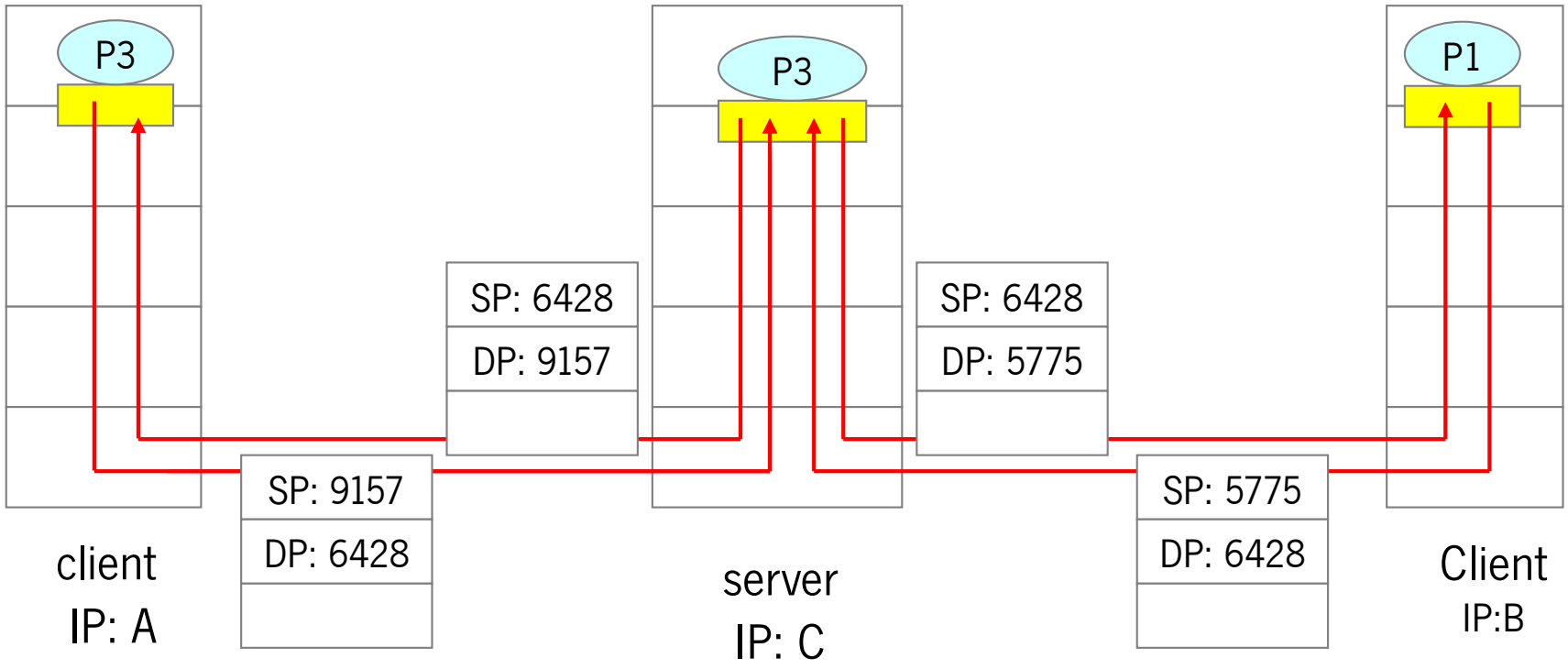
Fica em estado de escuta e pronto a receber datagramas

```
$ netstat –n –a
Proto  Local Address          Foreign Address        State
UDP    0.0.0.0:9876           *:*
```

- **E está pronto a receber dados:**

```
byte[] aReceber = new byte[1024];
DatagramPacket pedido =  new DatagramPacket(aReceber, aReceber.length);
s.receive(pedido);
```

UDP
Desmuliplexagem

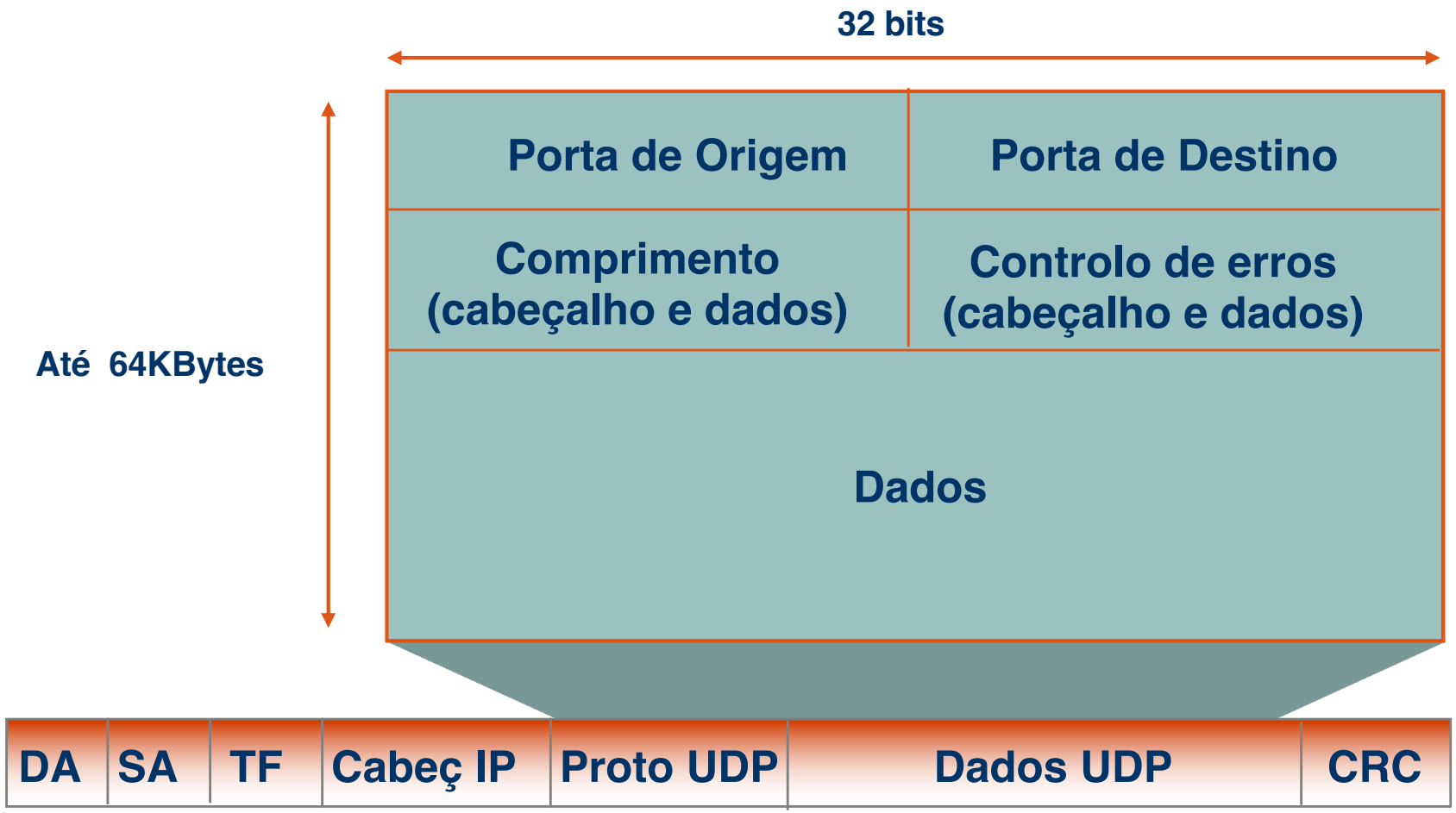


SP fornece o "return address"

TCP/IP

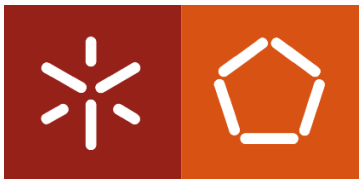
UDP - *User Datagram Protocol*

PDU



TCP/IP

UDP - *User Datagram Protocol*



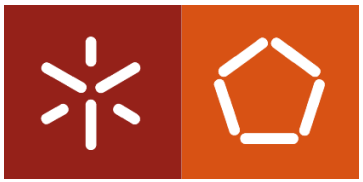
- **Controlo de erros (checksum) no UDP**
  - complemento para 1 da soma de grupos de 16 bits
  - cobre o datagrama completo (cabeçalho e dados)
  - o cálculo é facultativo mas a verificação é obrigatória
  - Checksum = **0** significa que o cálculo não foi efectuado
  - se Checksum ≠ **0** e o receptor detecta erro na soma:
    - o datagrama é ignorado (descartado);
    - não é gerada mensagem de erro para o transmissor;
    - a aplicação de recepção é notificada.

UDP - ***User Datagram Protocol***

- **Discussão:**
  - O que pode levar um "*developper*" a escolher o UDP como suporte à comunicação na sua App, sabendo à partida que não dá garantias nenhumas e fornece serviço mínimo de multiplexagem/desmultiplexagem e verificação de erros opcional? E tendo alternativas que dão todas as garantias!

TCP/IP

UDP - *User Datagram Protocol*

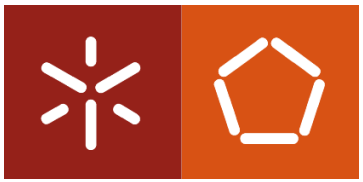


- **O que leva uma aplicação a escolher o UDP?**
  - Maior controlo sobre o envio dos dados por parte da aplicação;
    - aplicação controla quando deve enviar ou reenviar os dados sem deixar essa decisão ao transporte;

      → fuga ao controlo de congestão do TCP;
    - Aplicação decide quantos bytes envia realmente de cada vez
  - Não há estabelecimento e terminação da conexão;
  - Não é necessário manter informação de estado por conexão;
  - Menor *overhead* por pacote (cabeçalho UDP são apenas 8 bytes)

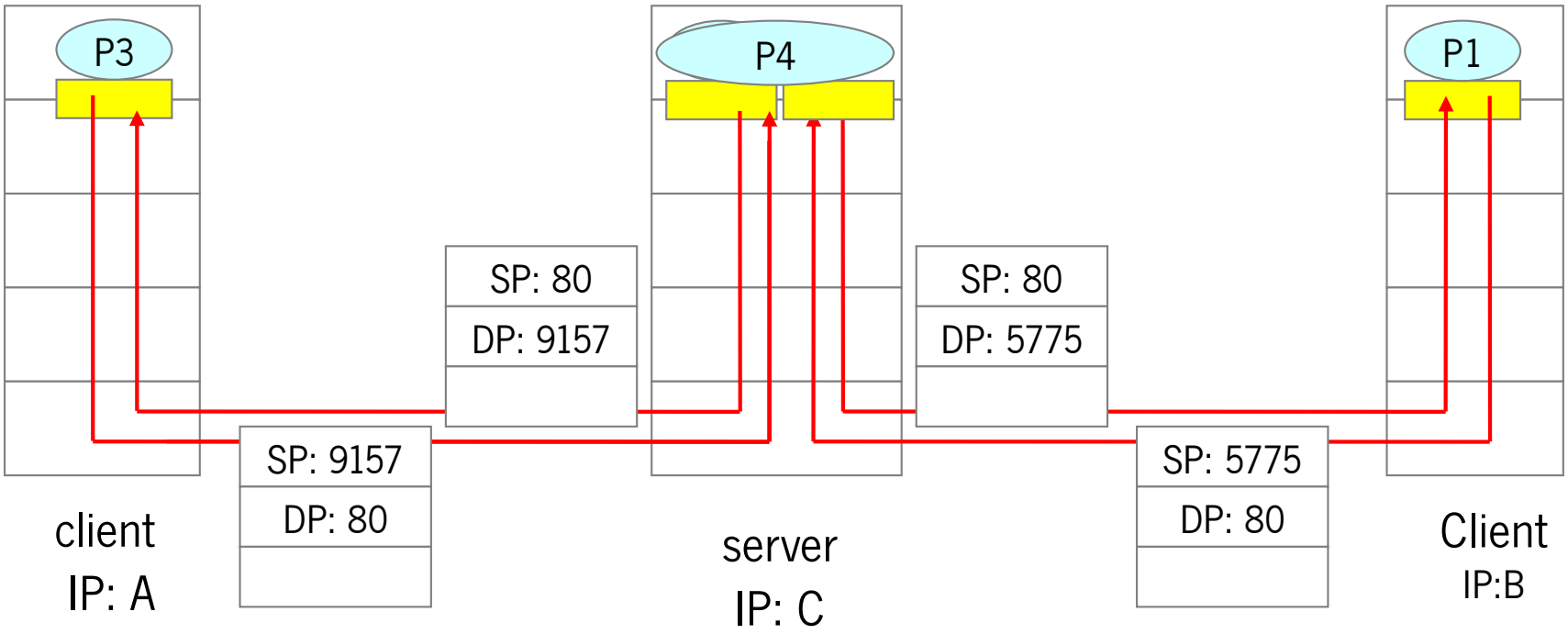
TCP/IP

TCP - *Transmission Control Protocol*

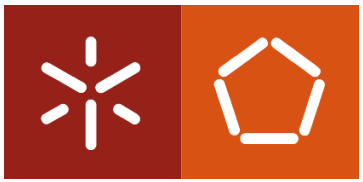


- **Funções do *Transmission Control Protocol***
  - transporte fiável de dados fim-a-fim (aplicações)
  - efetua associações lógicas fim-a-fim: conexões
  - cada conexão é identificada por um par de sockets: (IP_origem:porta_origem,IP_destino:porta_destino)
  - uma conexão é um circuito virtual entre portas de aplicações (também designadas portas de serviço)
  - multiplexa os dados de várias aplicações através de número de porta
  - efetua controlo de erros, controlo de fluxo e controlo de congestão

TCP
Desmuliplexagem

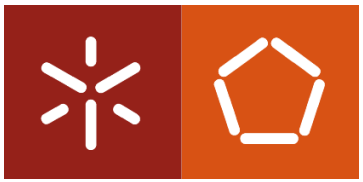


TCP
Desmuliplexagem



- Um *socket* TCP é identificado por quatro números: IP origem, número de porta da origem, IP destino e número de porta destino
- O sistema terminal ao receber um *datagrama* IP com um segmento TCP usa esses 4 números para redirecionar o segmento para o *socket* correcto.
- Um servidor suporta vários *sockets* TCP simultaneamente, cada um deles identificados pelos 4 números referidos
- Os servidores Web têm *sockets* diferentes para cada cliente (o http não persistente terá um *socket* diferente por cada pedido)

TCP



- **Criar o socket de atendimento principal**

```
ServerSocket welcomeSocket = new ServerSocket(9876);
```

```
$ netstat –n –a
Proto  Local Address        Foreign Address      State
TCP    0.0.0.0:9876         0.0.0.0:0            LISTENING
```

- **E lidar com cada conexão num sokect específico:**

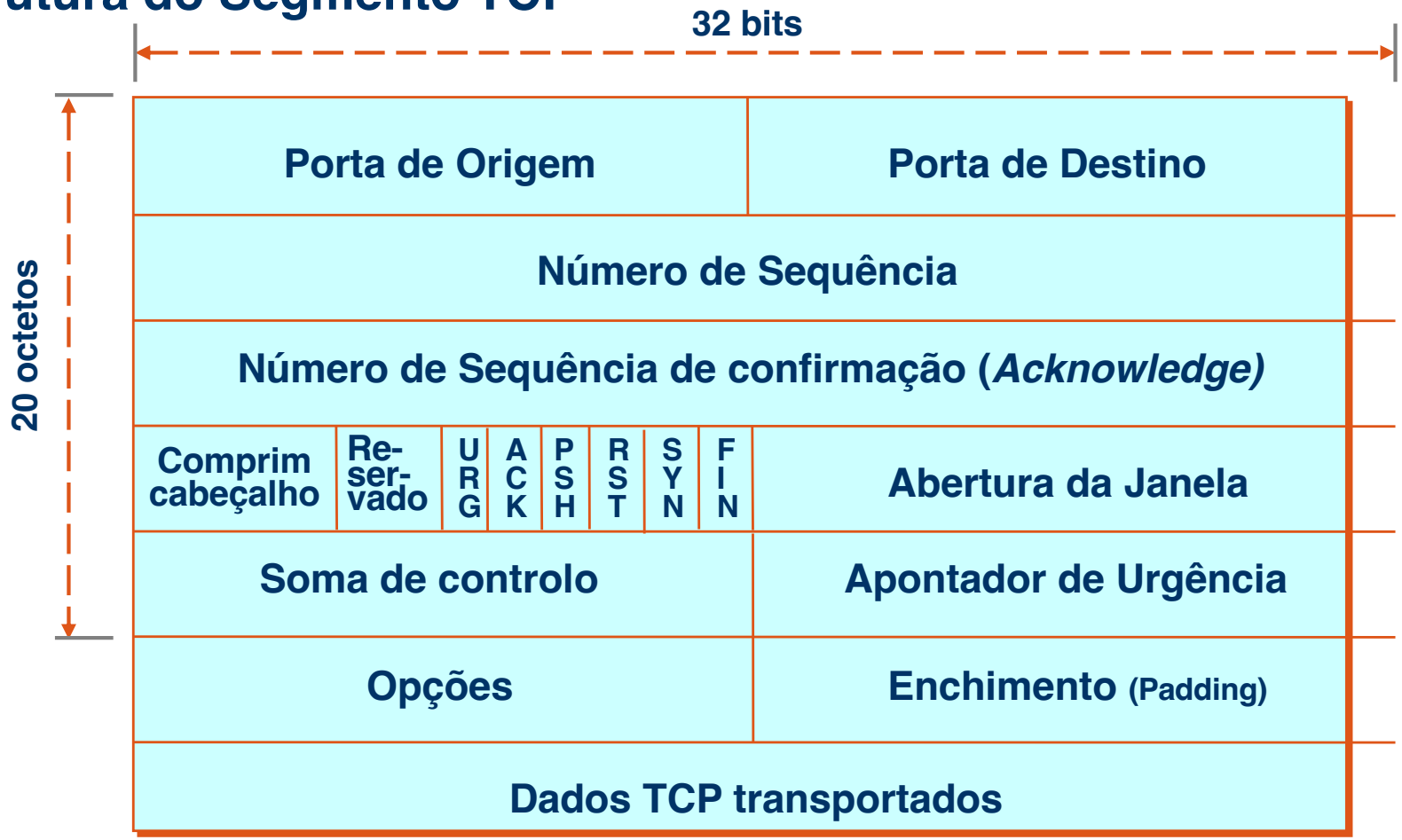
```
Socket connectionSocket = welcomeSocket.accept();
```

```
$ netstat –n –a
Proto  Local Address        Foreign Address      State
TCP    127.0.0.1:9876       127.0.0.1:5459       ESTABLISHED
```

TCP/IP

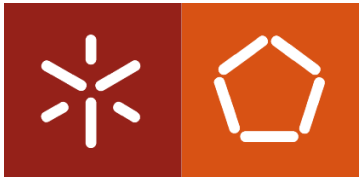
TCP - *Transmission Control Protocol*

Estrutura do Segmento TCP



TCP/IP

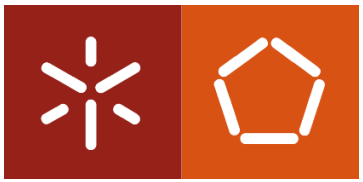
TCP - ***Transmission Control Protocol***



- *Porta Orig/Dest* – N° da porta TCP da aplicação de Origem/Destino
- *Número de Sequência* - ordem do primeiro octeto de dados no segmento (se SYN = 1, este número é o *initial sequence number*, ISN)
- *Número de Ack* (32 bits) - o número de ordem do octeto seguinte na sequência que a entidade TCP espera receber.
- *Comprimento Cabeçalho* (4 bits) - número de palavras de 32 bits no cabeçalho.
- *Flags* (6 bits) - indicações específicas.
- *Janela* – n° de octetos que o receptor é capaz de receber (controlo fluxo)
- *Soma de controlo* (16 bits) – soma para detecção de erros (segm)
- *Apontador de Urgência* (16 bits) – adicionado ao n° de sequência dá o n° de sequência do último octeto de dados urgentes.
- *Opções* (variável) - especifica características opcionais (ex. MSS, timestamp, factor de escala para a janela, etc.)

TCP/IP

TCP - ***Transmission Control Protocol***



Flags TCP (1 bit por flag):

URG - indica se o apontador de urgência é válido

ACK - indica se o nº de sequência de confirmação é válido

PSH - o recetor deve passar imediatamente os dados à aplicação (aparece nos seg de transferência de dados)

RST - indica que a conexão TCP vai ser reinicializada

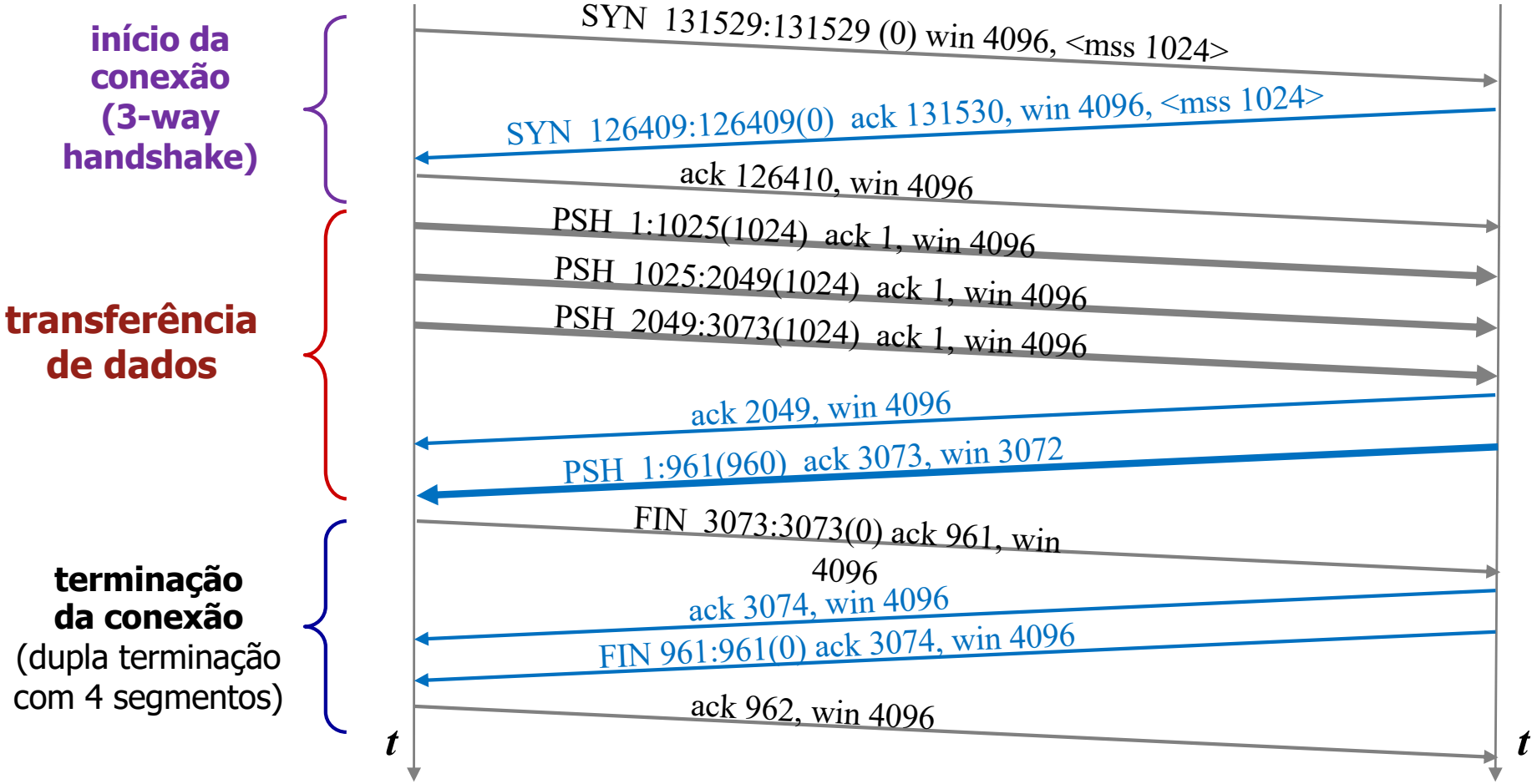
SYN - indica que os números de sequência devem ser sincronizados para se iniciar uma conexão

FIN - indica que o transmissor terminou o envio de dados

Os segmentos SYN e FIN consomem um número de sequência

TCP/IP

TCP - *Transmission Control Protocol* operação

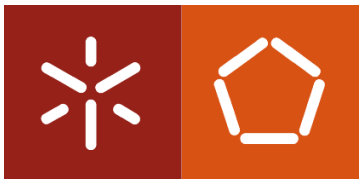


Controlo de conexão

- Que acontece se o servidor não responde ao SYN inicial?
- Que acontece se o servidor não enviar um FIN de volta?
  Ou o cliente não enviar o ACK final?
- Quais as questões de segurança associadas ao início e fim de conexão?
- O que acontece se em vez de *3-way* se usasse *2-way handshake* ?

TCP/IP

Estabelecimento de ligação



O emissor e o receptor TCP estabelecem uma ligação antes de iniciarem a troca de segmentos de dados.

- **Inicialização de variáveis**
  - números de sequência
  - *buffers*, controlo de fluxo (e.g. **`RcvWindow`**)
- ***Cliente: inicia a pedido de ligação***

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

- ***Servidor:* é contactado pelo cliente e aceita o pedido de ligação**

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

<u>Três passos:</u>

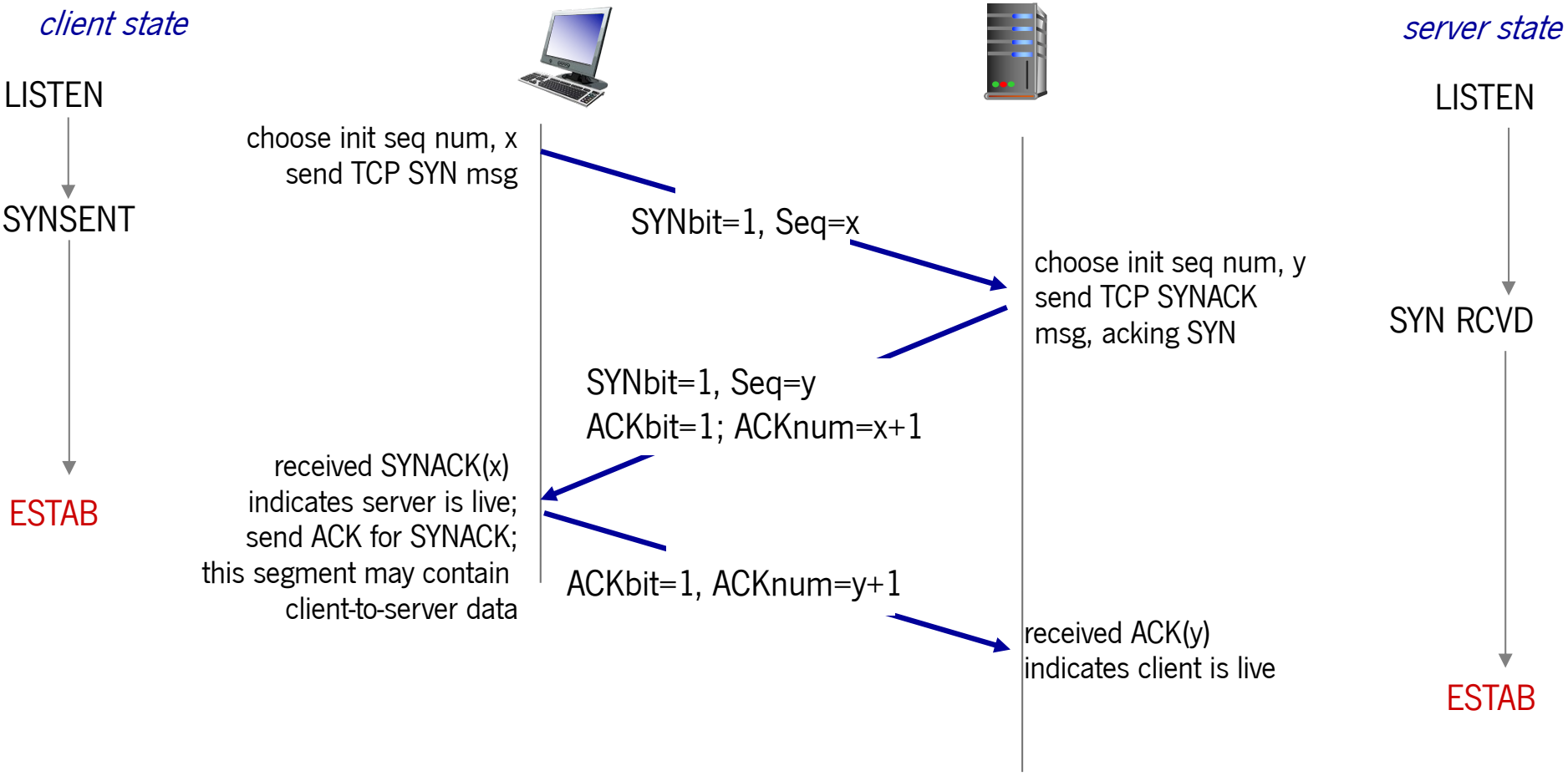
<u>1:</u> O cliente envia segmento SYN para o servidor
- especifica o número de sequência inicial
- sem dados

<u>2:</u> O servidor recebe o SYN e responde com um segmento SYNACK
- aloca espaço de armazenamento
- especifica o número de sequência inicial

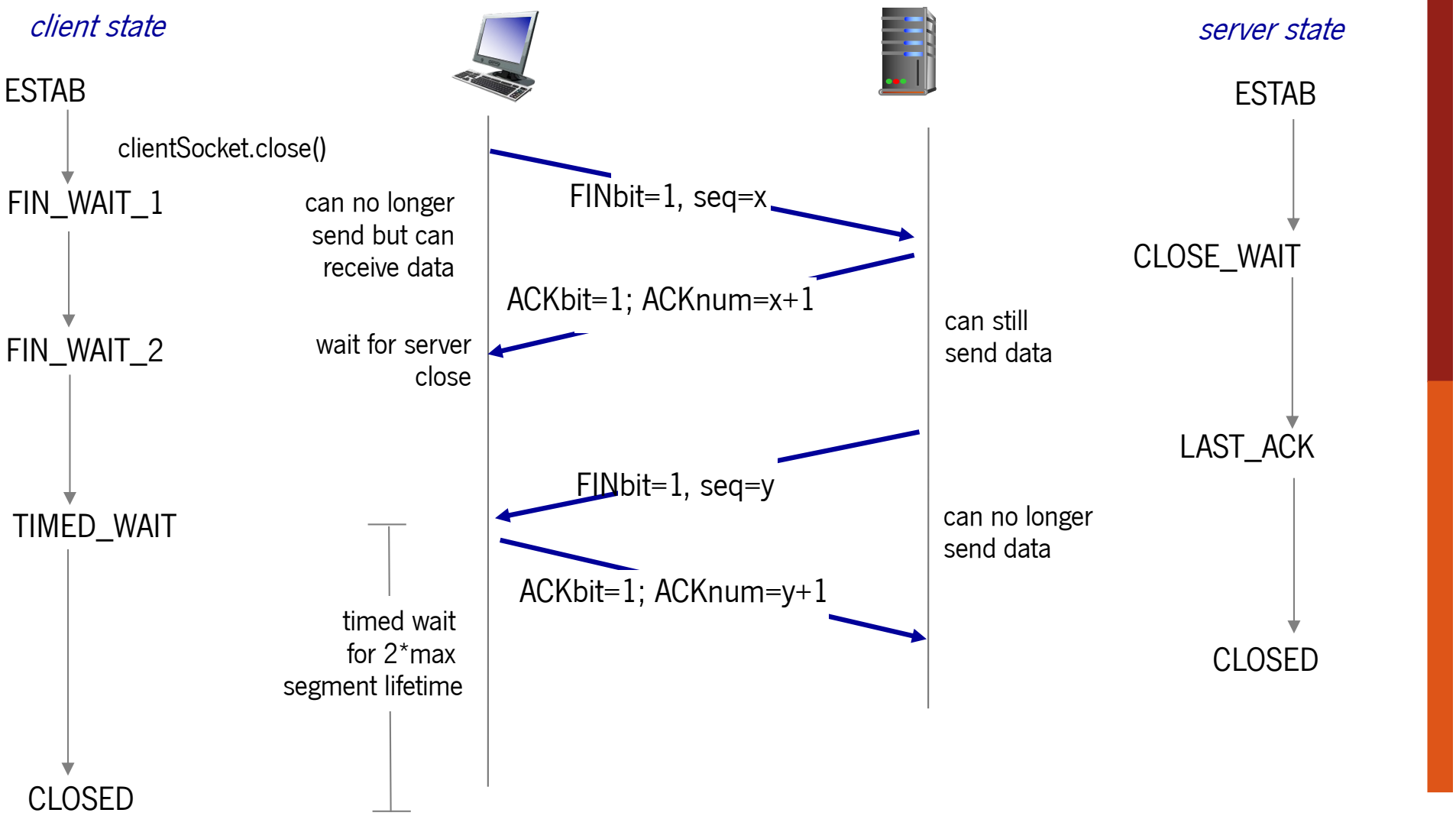
<u>3:</u> O cliente recebe o segmento SYNACK, e responde com um segmento ACK que pode conter dados

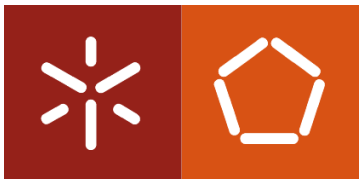
TCP/IP

Estabelecimento de ligação



Estabelecimento de ligação





Maximum Segment Size (MSS) do TCP

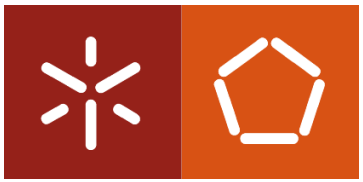
- opção TCP que apenas aparece em segmentos SYN
- o MSS é o maior bloco de dados da aplicação que o TCP enviará na conexão
- ao iniciar-se uma conexão, cada lado tem a opção de anunciar ao outro o MSS que espera receber
- o maior MSS possível é igual ao MTU do interface menos os comprimentos dos cabeçalhos TCP e IP:

  Exemplo:

  - sobre Ethernet o maior MSS é 1460 bytes

TCP/IP

Estabelecimento de ligação



```
Frame 1516: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface en6, id 0
Ethernet II, Src: Tp-LinkT_dd:71:32 (c4:e9:84:dd:71:32), Dst: Xensourc_83:7b:15 (00:16:3e:83:7b:15)
Internet Protocol Version 4, Src: 193.136.9.175, Dst: 193.136.9.241
Transmission Control Protocol, Src Port: 56843, Dst Port: 8080, Seq: 0, Len: 0
    Source Port: 56843
    Destination Port: 8080
    [Stream index: 54]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1125210519
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1011 .... = Header Length: 44 bytes (11)
    Flags: 0x0c2 (SYN, ECN, CWR)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0xfdbb [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), Timestamps, SACK permitted…
        TCP Option - Maximum segment size: 1460 bytes
        TCP Option - No-Operation (NOP)
        TCP Option - Window scale: 6 (multiply by 64)
        TCP Option - No-Operation (NOP)
        TCP Option - No-Operation (NOP)
        TCP Option - Timestamps: TSval 314438861, TSecr 0
        TCP Option - SACK permitted
        TCP Option - End of Option List (EOL)
    [Timestamps]
        [Time since first frame in this TCP stream: 0.000000000 seconds]
        [Time since previous frame in this TCP stream: 0.000000000 seconds]
```

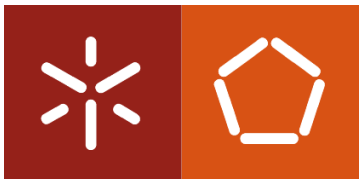
Segmento sem dados (LEN=0)
Mas conta um byte na stream que precisa ser confirmado com ACK

Inicio de conexão (SYN)

Opções negociadas no início da conexão: MSS e Window Scale

TCP

transporte fiável – controlo de erros

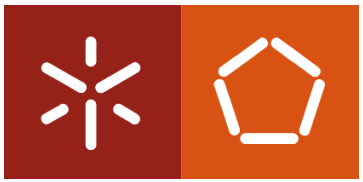


“A” pretende mandar – de forma fiável – uma mensagem “m” para “B”, usando uma ligação de “rede” não fiável.

- Como posso ter a certeza que B recebeu a mensagem “m”?
- O que pode correr mal no envio de “m”?
  - Tendo em atenção que estamos na camada de transporte
- Como lidar com os erros?

TCP

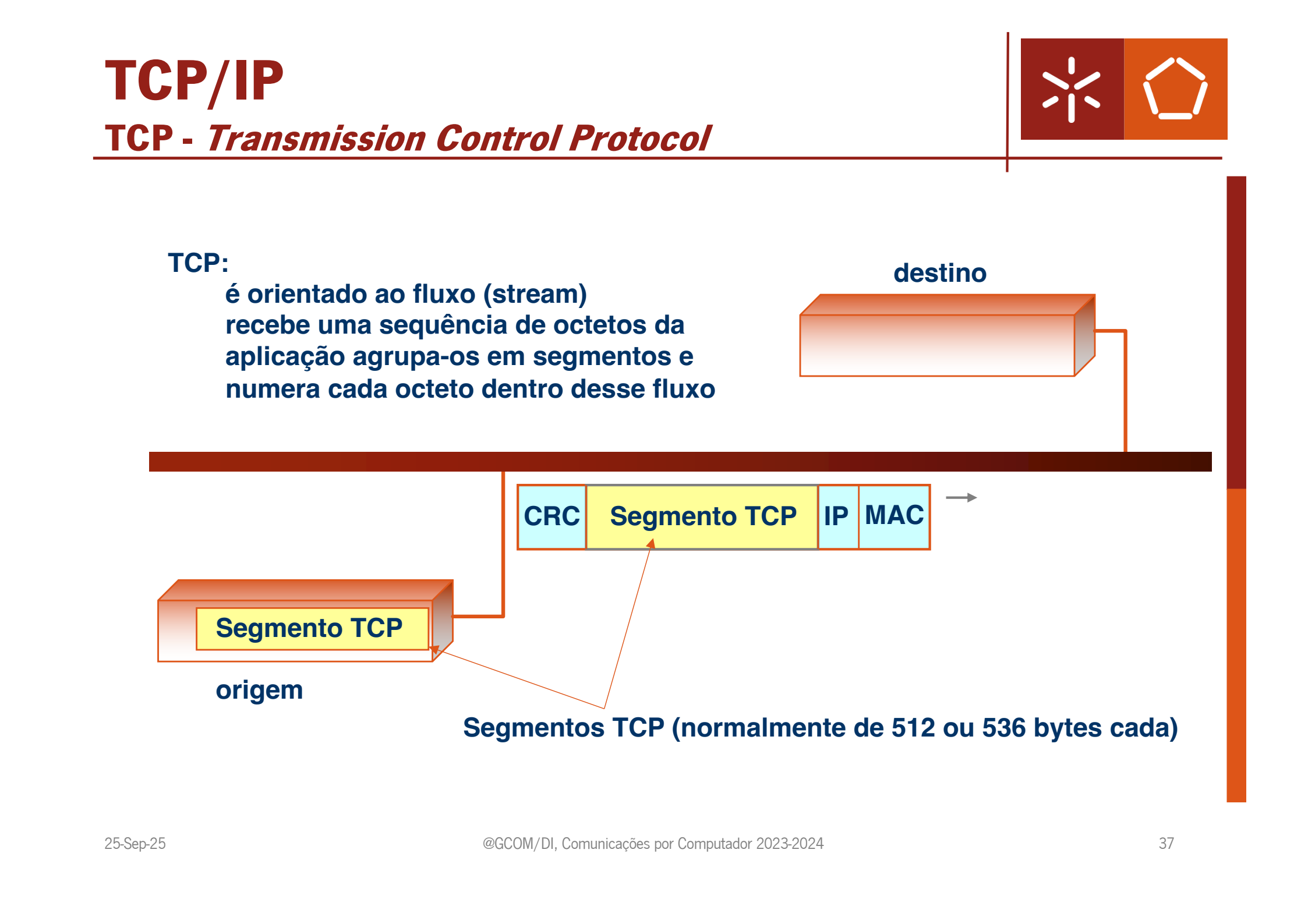
transporte fiável – controlo de erros



- **Nas redes o mecanismo preferencial é o ARQ (Automatic Repeat reQuest)**
  - Detecção de erros
  - Feedback do receptor
  - Retransmissão

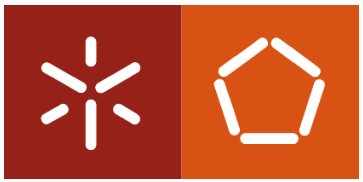
TCP/IP

TCP - *Transmission Control Protocol*



TCP:

- é orientado ao fluxo (stream)
- recebe uma sequência de octetos da aplicação agrupa-os em segmentos e numera cada octeto dentro desse fluxo

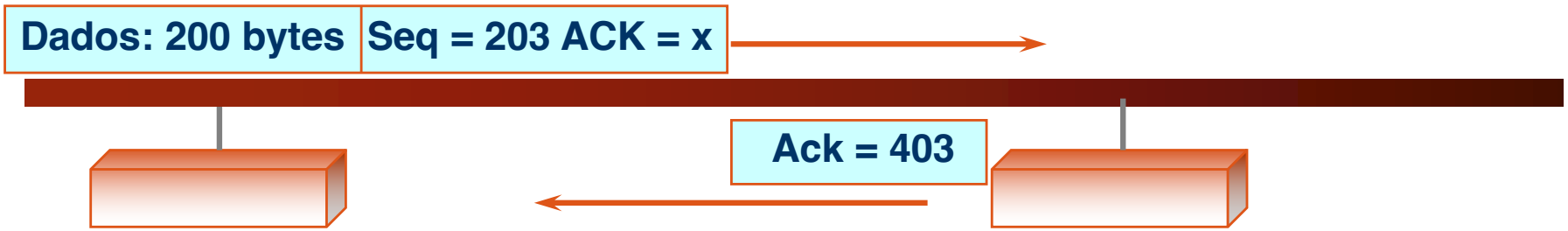


Segmentos TCP

- sequenciação necessária para ordenação na chegada
- o *número de sequência* é incrementado pelo número de bytes do campo de dados
- cada segmento TCP tem de ser confirmado (ACK), contudo é válido o ACK de múltiplos segmentos
- o campo ACK indica o próximo byte (*sequence*) que o receptor espera receber (*piggyback*)
- o emissor pode retransmitir por *timeout:* o protocolo define o tempo máximo de vida dos segmentos ou MSL (maximum segment lifetime)

TCP/IP

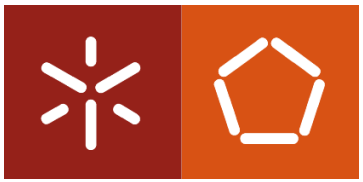
TCP - *Transmission Control Protocol*



- Cada *sistema-final (end-system)* mantém o seu próprio *Número de Sequência*: 0 .. 2^{32} -1
- N° de ACK = Número de Sequência + bytes correctos lidos no segmento.

TCP/IP

Controlo de erros

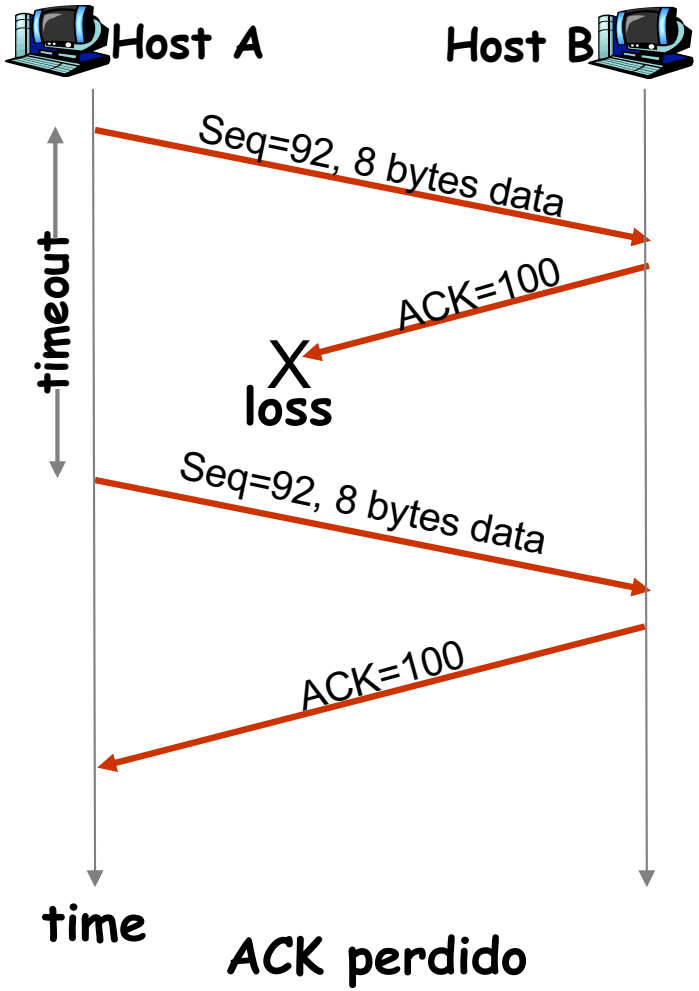


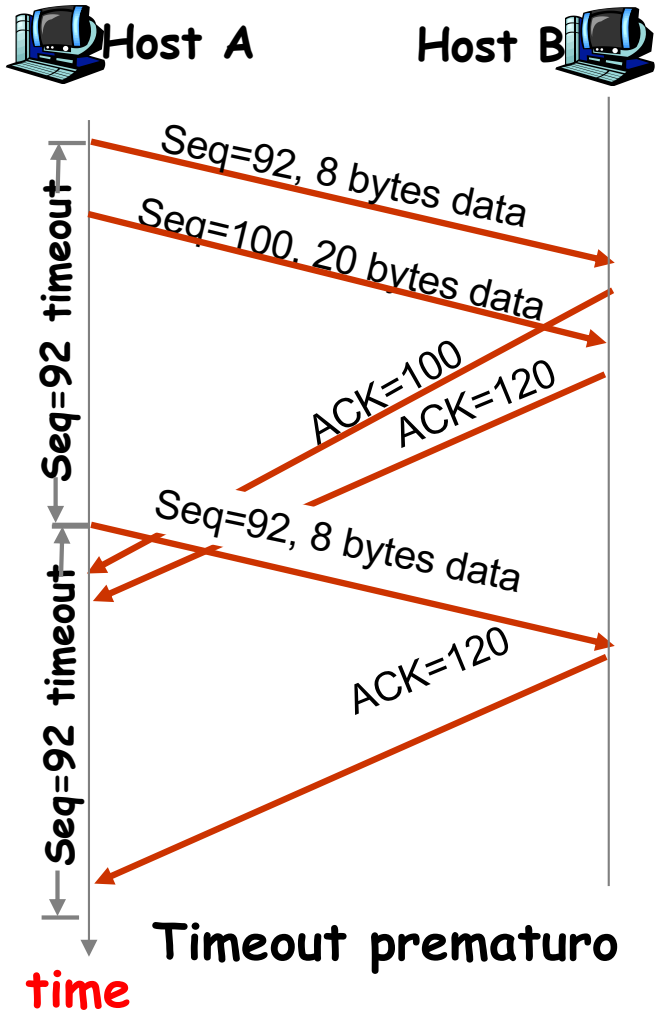
- **No TCP não há confirmações negativas**
  - Só há confirmações positivas (*Acks*) – decisão de design!
  - Por esse motivo o emissor pode apenas **desconfiar** que um determinado segmento enviado não chegou ao destino

QUESTÕES:

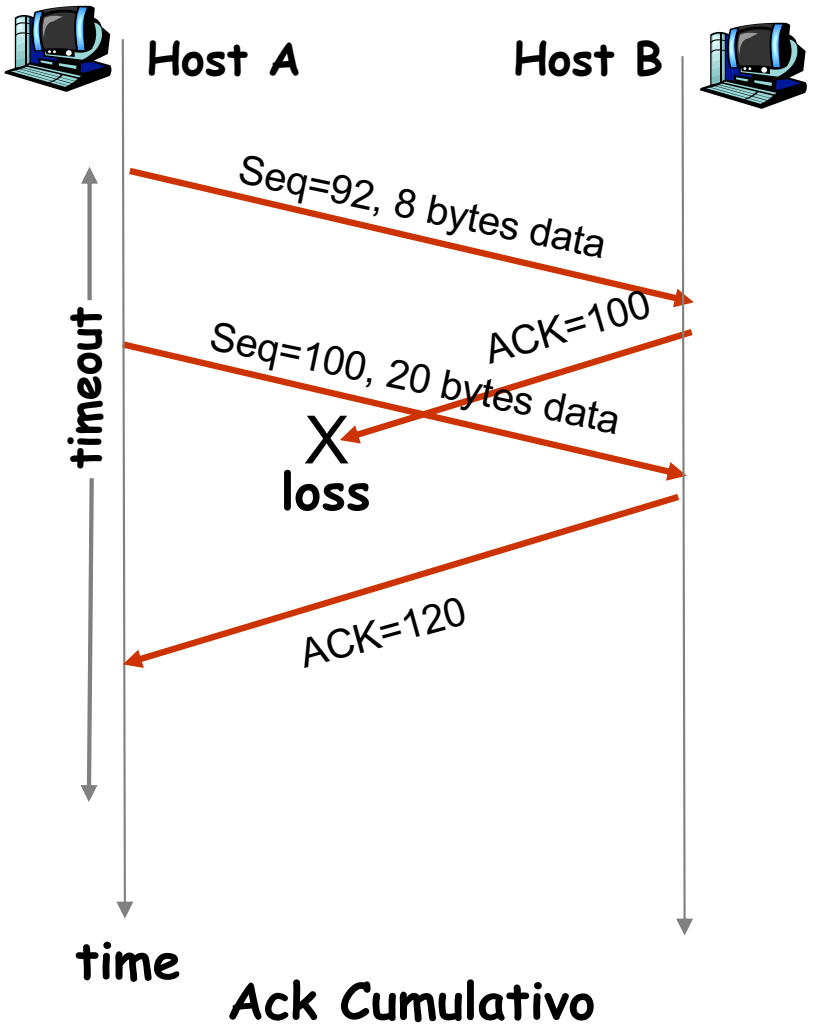
1) Que deve então fazer o "receptor" quando recebe um segmento em erro?
2) Como pode o "emissor" saber que o segmento estava em erro?
3) E se o segmento se perder mesmo?

TCP: Cenários de retransmissões



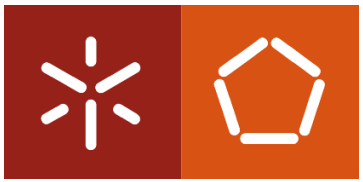


TCP: Cenários de retransmissões



TCP/IP

TCP Round Trip Time e Timeout

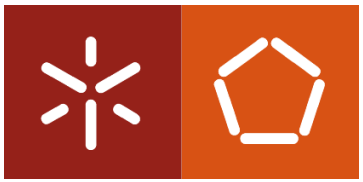


- **Como definir o valor do Timeout no TCP?**
  - Com base no RTT (mas o RTT varia)
  - Demasiado curto aumenta o número de retransmissões desnecessárias?
  - Demasiado longo atrasa a reacção a um segmento perdido

É necessário estimar o RTT

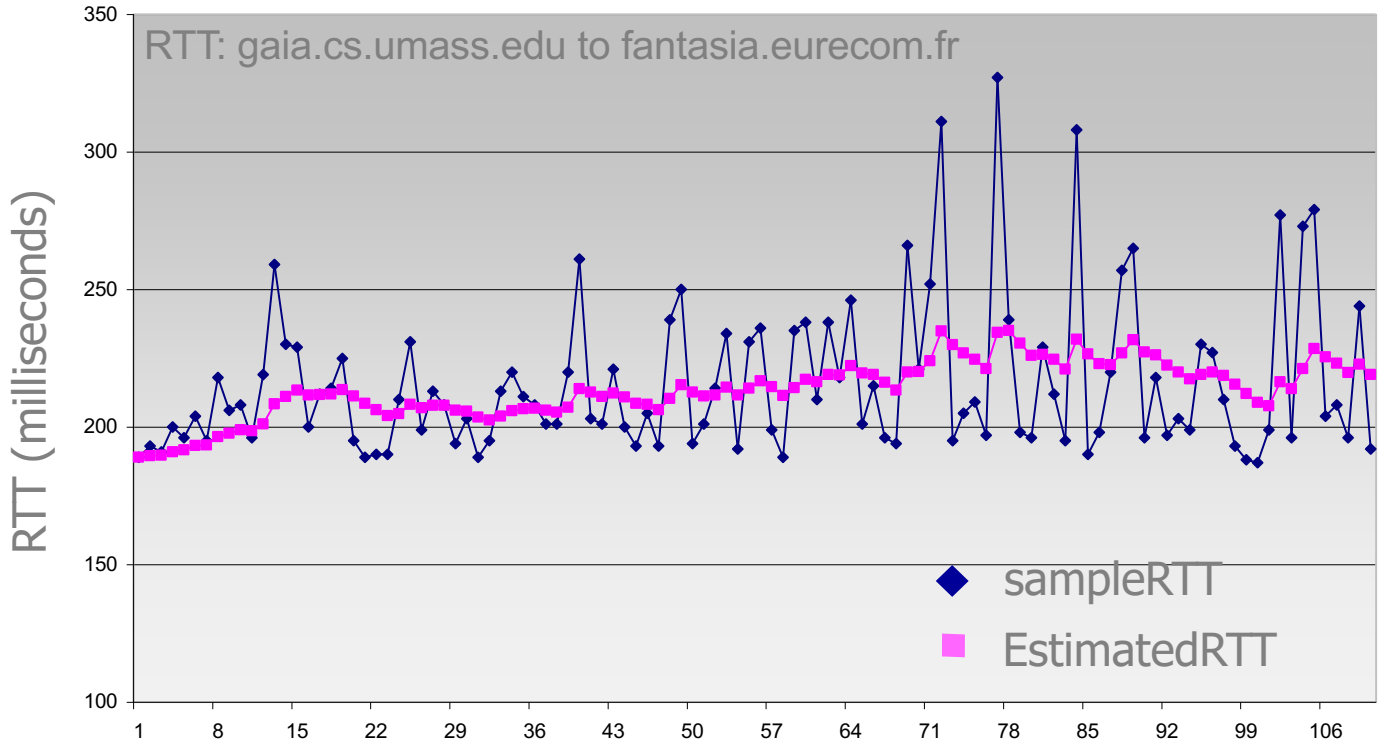
TCP/IP

TCP Round Trip Time e Timeout



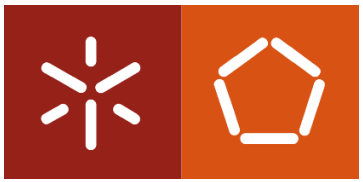
`EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT`

- média móvel de peso exponencial
- lo peso do passado decresce exponencialmente...
- valor tipico: **α =** 0.125



TCP/IP

TCP Round Trip Time e Timeout



$$EstimatedRTT = (1- \alpha)*EstimatedRTT + \alpha*SampleRTT$$
$$(typically,\ \alpha = 0.125)$$

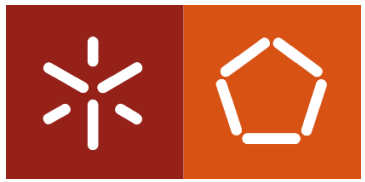
- `EstimatedRTT` **- Média móvel de peso exponencial onde a importância de uma amostra passada decresce exponencialmente**
- **O** `SampleRTT` **é medido desde a transmissão de um segmento até à recepção do Ack respectivo**
- **O** `timeout` **é definido com base nesta média (**`EstimatedRTT`**). Quanto maior for diferença entre os** `SampleRTT` **e o** `EstimatedRTT` **maior deverá ser o valor definido para o** `timeout`**.**

$$DevRTT = (1-\beta)*DevRTT + \beta*|SampleRTT - EstimatedRTT|$$
$$(typically,\ \beta = 0.25)$$

$$Timeout = EstimatedRTT + 4*DevRTT$$

Margem de Segurança

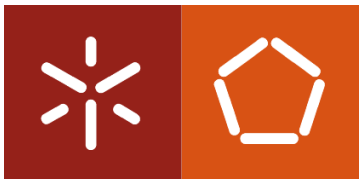
Gestão de ACKs [RFC 1122, RFC 2581]



Evento no Receptor	Acção da entidade TCP
Chegada de um segmento com o número de sequência esperado e tudo para trás confirmado.	Atrasa envio de ACK 500ms para Verificar se chega novo segmento. Senão chegar, envia ACK
Chegada de um segmento com o número de sequência esperado e um segmento por confirmar	Envia imediatamente um ACK cumulativo que confirma os dois Segmentos.
Chegada de um segmento com o número de sequência superior ao esperado. Buraco detectado	Envia imediatamente um ACK duplicado indicando o número de sequência esperado
Chegada de um segmento que preenche completa ou incompletamente um buraco	Se o número do segmento coincidir com o limite inferior do buraco envia ACK imediatamente.

TCP/IP

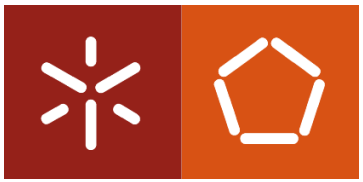
Fast Retransmit / Fast Recovery



- **A duração do *timeout* é por vezes demasiado longa, o que provoca atrasos na retransmissão de um pacote perdido**
- **Para minimizar esse problema, o emissor procura detectar perdas através da recepção de ACKs duplicados**
  - O emissor envia normalmente vários segmentos seguidos. No caso de algum deles se perder vai haver vários ACKs duplicados.
  - Se o emissor recebe **três** ACKs duplicados supõe que o segmento respectivo foi perdido e retransmiti-o (***Fast Restransmit***)

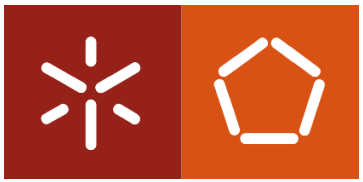
TCP/IP

Fast Retransmit / Fast Recovery



- **Fast recovery**
  - Implementado em conjunto com **Fast Retransmit**
  - Recomendado mas não obrigatório [RFC 5681]
  - Por cada Ack duplicado recebido, e enquanto não for recebido o Ack em falta, estica-se temporariamente a janela em 1 MSS
    - Para poder continuar a enviar, embora sem ajustar a janela
  - Quando chegar o Ack em falta, ou um Ack cumulativo que cubra o Ack em falta, abandona-se o estado de recuperação rápida

Exercício nº1



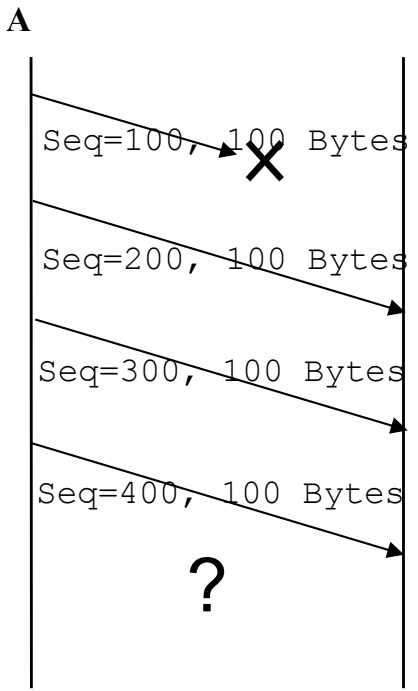
- **Suponha que o host A envia 2 segmentos TCP ao host B. O primeiro tem nº de sequência 90 e o segundo tem nº de sequência 120.**
  - Quantos bytes estão no primeiro segmento?
  - Se o primeiro segmento se perder, mas o segundo chegar em boas condições, qual será o nº da confirmação a ser enviada por B?

- **Na camada 2 foram analisados dois mecanismos de correcção de erros: Go-Back-N e Selective Reject. Tendo em conta a forma como o TCP recupera os segmentos em erro, como o classificaria? Como um protocolo Go-Back-N ou como um protocolo Selective Reject? Justifique.**

Exercício nº 2

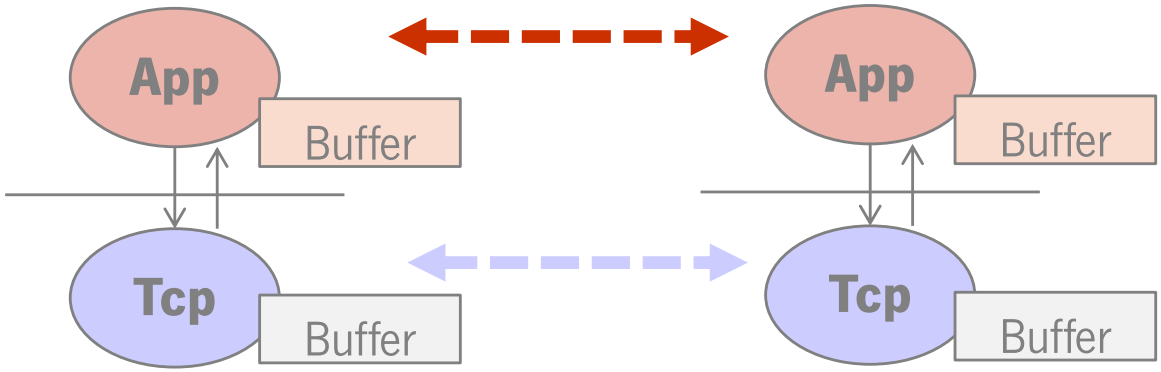
- **Os processos A e B estabeleceram uma ligação TCP. A figura seguinte representa um diagrama temporal de envio de alguns segmentos TCP entre A e B. No instante anterior ao nosso cenário, A recebeu de B o segmento (Ack=100, 0 bytes de dados) e pretende transmitir mais 400 bytes de informação.**

  a) Quantos segmentos tipo *acknowledgement* são transmitidos de B para A em resposta a cada um dos segmentos recebidos por B. Indique quais os valores dos campos Ack de cada um dos segmentos.
  b) Considere que, após A ter recebido todos os segmentos transmitidos em a), A retransmite o segmento (Seq=100, 100 bytes). B recebe este segmento e envia um segmento tipo acknowledgment. Indique qual o valor do campo Ack desse segmento.



Exercícios de reflexão

- Quem define qual o tamanho do segmento TCP a ser transmitido? A aplicação? O transporte? A rede IP?



- Supondo que a aplicação fornece sempre dados em contínuo, qual o tamanho adequado de um segmento TCP? Como determinar?

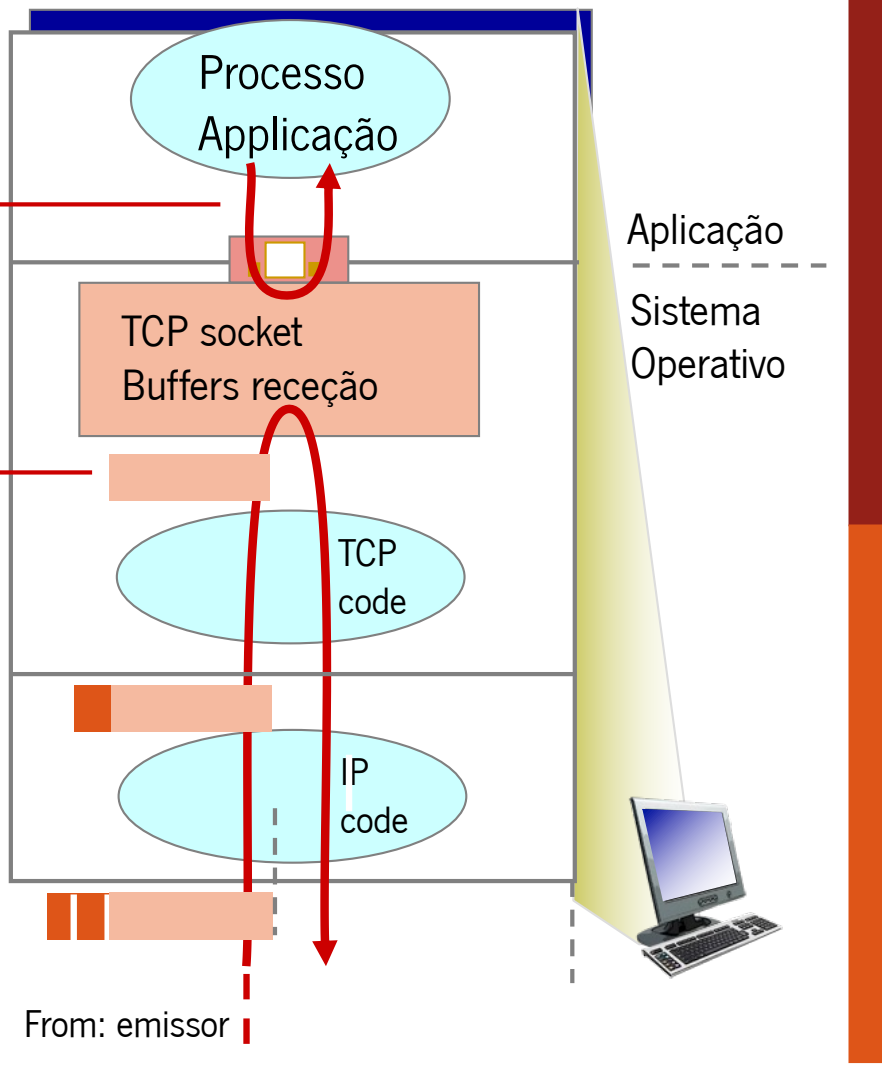
TCP/IP

TCP - ***Transmission Control Protocol***

Controlo de fluxo

Aplicação vai recebendo e retirando os dados dos buffers do socket TCP ....

... mais lentamente do que o recetor TCP vai colocando no buffer (emissor sempre a enviar)



Stack protocolar do recetor

Controlo de Fluxo

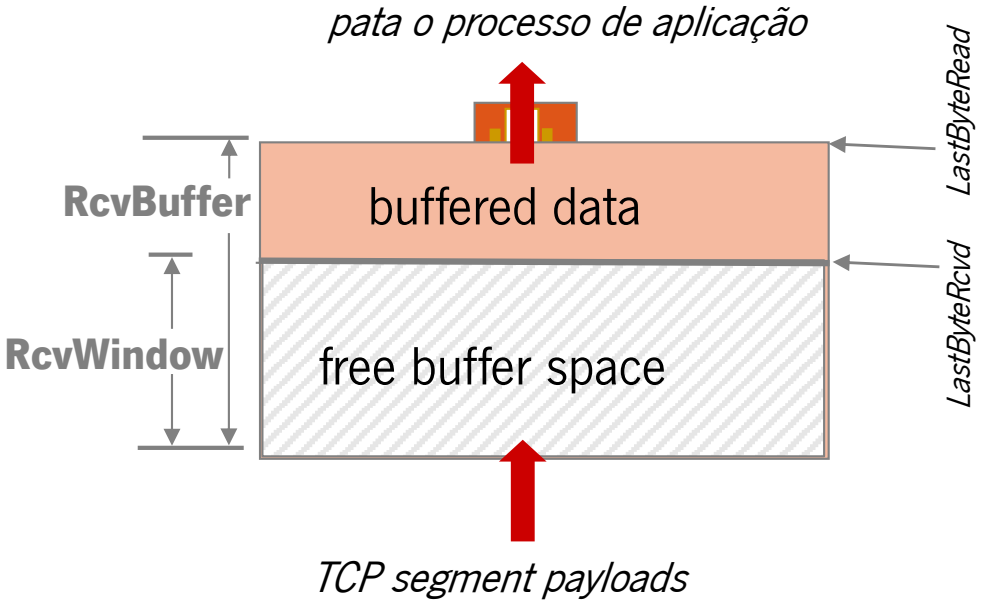
Recetor controla o emissor, para que o emissor não extravase o buffer de receção, enviando demasiados dados ou demasiado depressa

TCP/IP

TCP - ***Transmission Control Protocol***

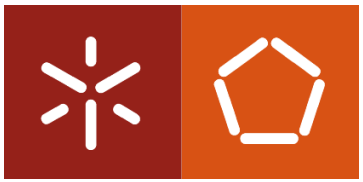
Controlo de fluxo

- O espaço no *buffer* do recetor é limitado.
  - ...definido pelo Sistema Operativo
  - ... tipicamente 4096 bytes
  - ... pode ser redefinido com "Soket Options" se o SO deixar!...
- O espaço livre é anunciado ao emissor num campo dos segmentos TCP enviados no sentido contrário "RcvWindow" (janela).
- O emissor sabe sempre o que pode mandar → nunca extravasa o buffer do recetor

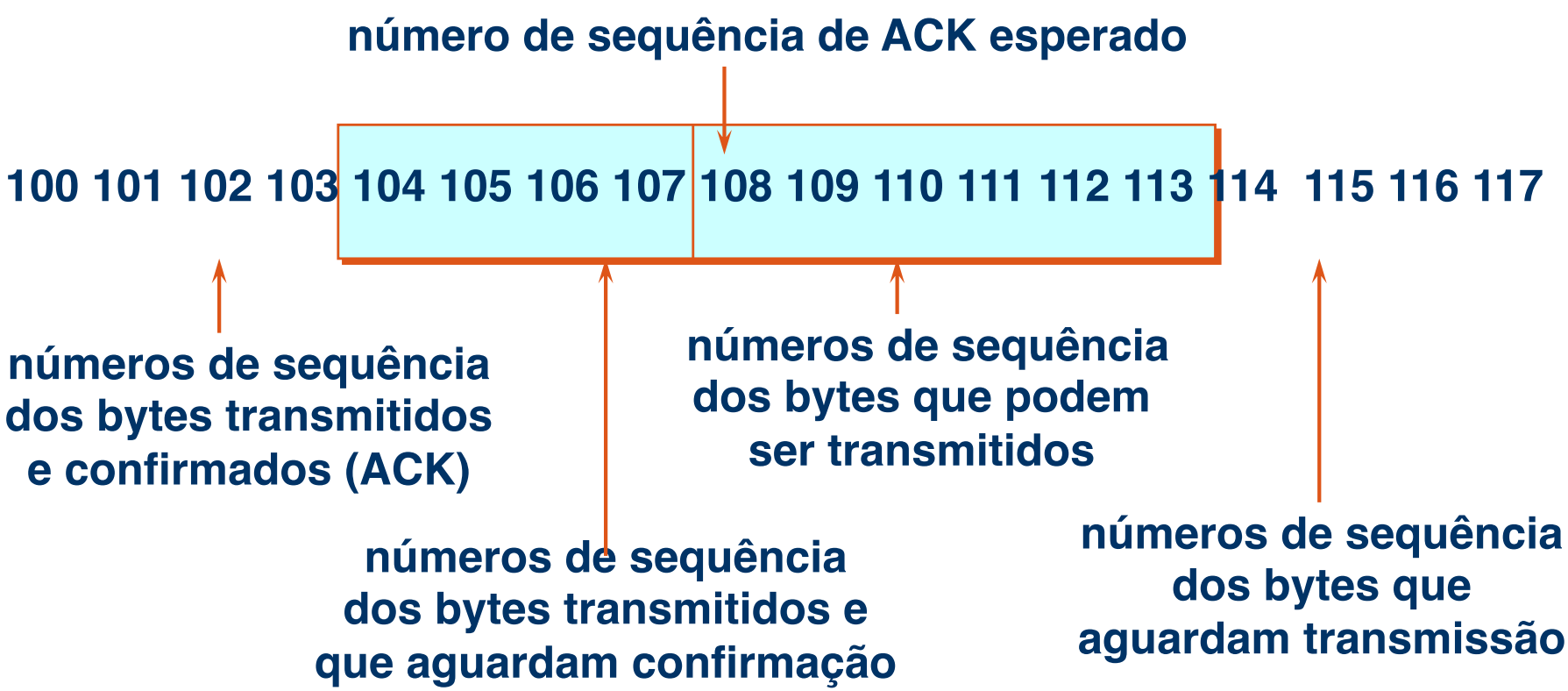


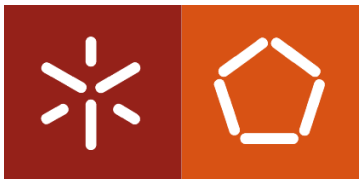
Buffer de recepção

RcvWindow = RcvBuffer- [LastByteRcvd - LastByteRead]



Controlo de fluxo baseado na abertura da janela anunciada no segmento recebido do parceiro



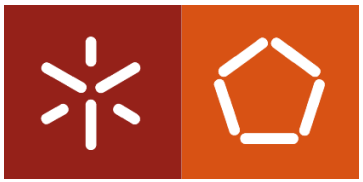


Uma entidade TCP:

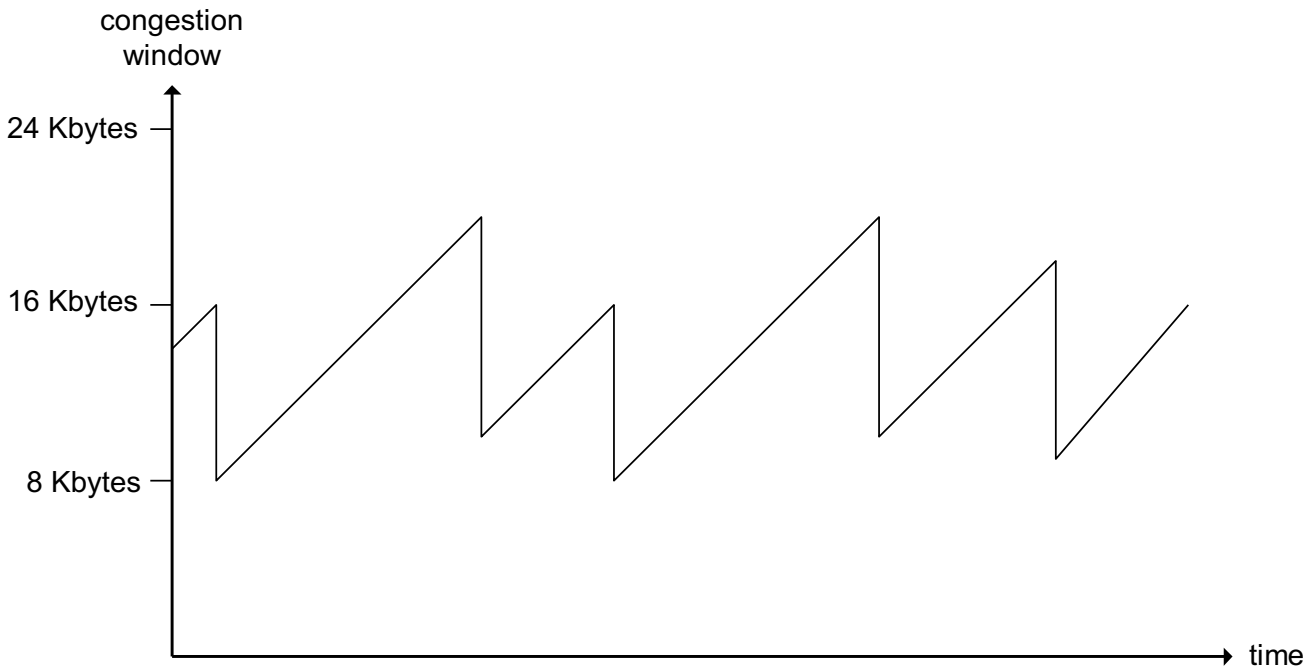
- Procura aperceber-se de situações de congestão através da recepção de ACKs duplicados e da ocorrência de *timeouts.*
- Utiliza três mecanismos para prevenir/minimizar situações de congestão:
  - **AIMD (Additive Increase/Multiplicative Decrease)**
  - **SlowStart**
  - **Conservativo depois de um *timeout***

- Mais uma variável: **Janela de Congestão (*CongWin*)**

```
LastByteSent - LastByteAcked <= MIN ( RecvWin,
  CongWin)
```



- **AIMD (Additive Increase/Multiplicative Decrease)**
  - **Sempre que chega um ACK esperado o tamanho da janela de congestão é incrementado**
  - **Quando chegam ACKs duplicados o tamanho da janela de congestão diminui para metade**



Slow Start

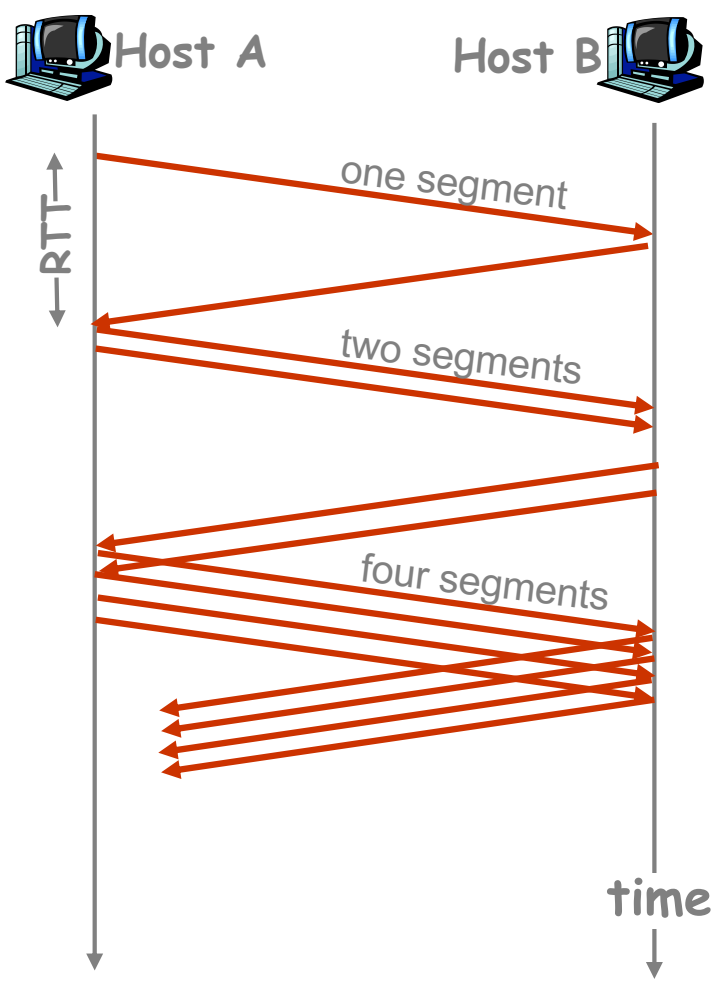
- No inicio da ligação, normalmente o tamanho da janela de congestão é igual a 1 MSS
- Sempre que é recebido um ACK, janela aumenta 1MSS (ou seja, cresce exponencialmente) até ser detectada a primeira perda ou até patamar congestão

TCP Reno (versão + recente)

- Se a perda corresponder a um timeout a janela de congestão volta a 1 MSS e re-inicia SlowStart
- Se corresponder a ACks duplicados é decrementada para metade, a partir daí a janela cresce de forma linear

TCP Tahoe

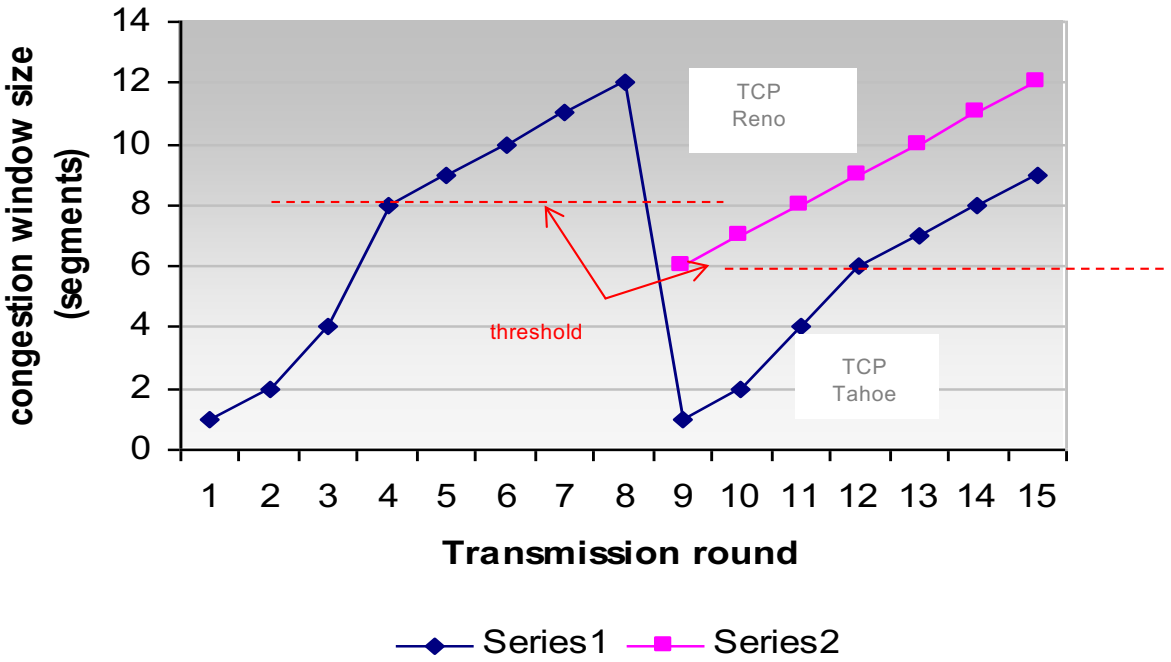
- A janela de congestão volta a 1 MSS em qualquer dos casos e re-inicia o SlowStart



TCP/IP

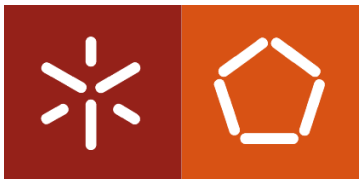
TCP - *Transmission Control Protocol* controlo de congestão

- Congestion Avoidance
  - O SlowStart progride até à detecção de uma perda ou até ter sido atingido um determinado threshold
  - Quando o threshold é atingido a janela de congestão passa a crescer de forma linear.
  - Quanto ocorre um timeout e o SlowStart é inicializado o thresold é decrementado para metade do tamanho da janela actual



TCP/IP

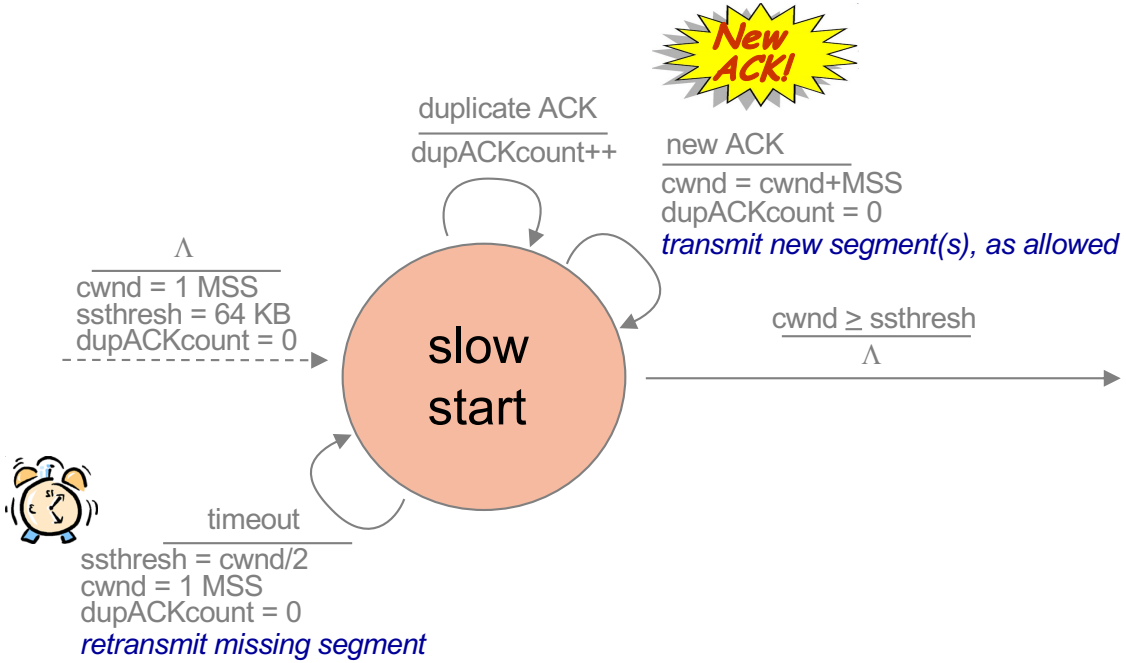
TCP - *Transmission Control Protocol* controlo de congestão



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

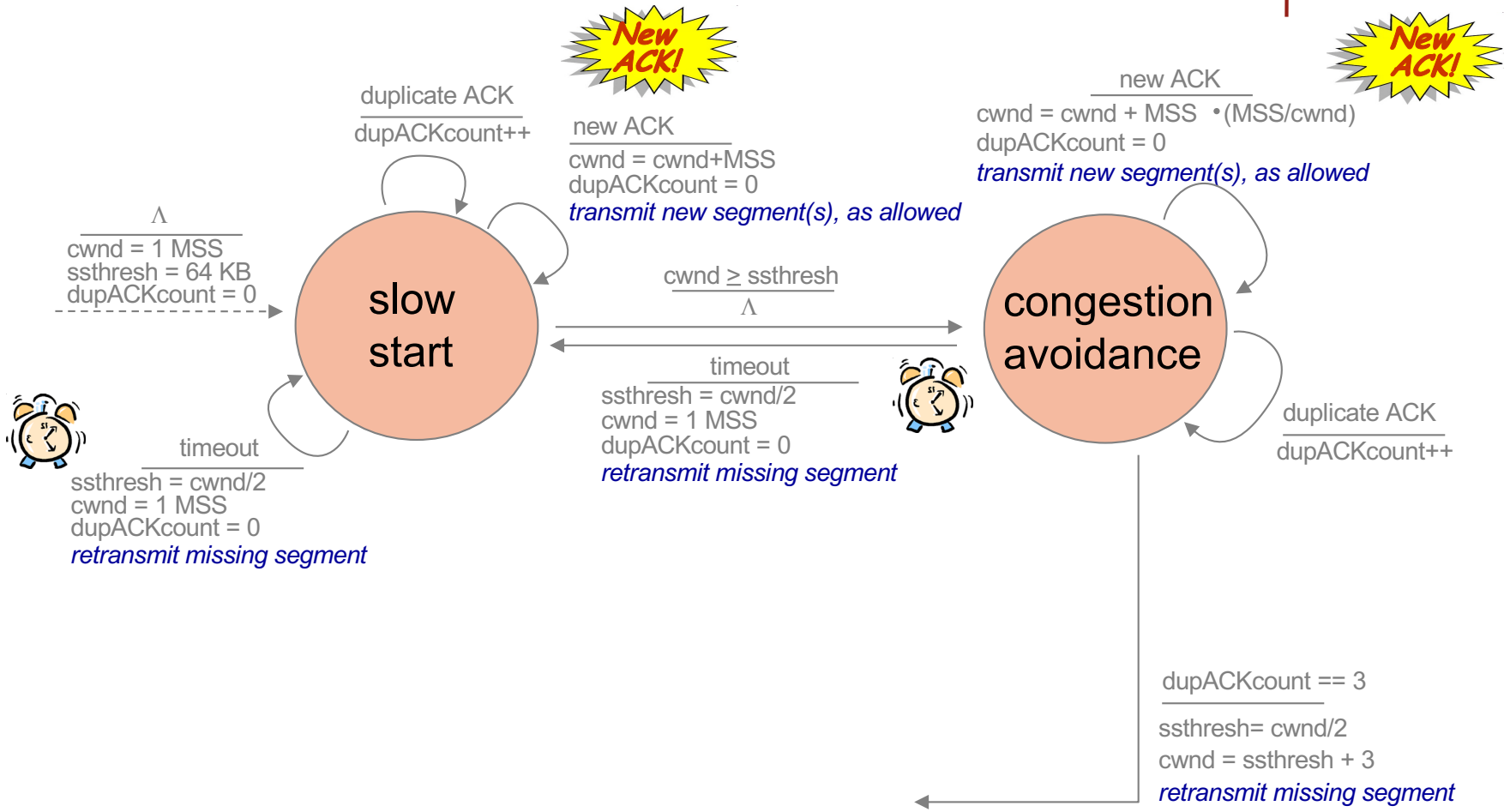
TCP/IP

TCP - *Transmission Control Protocol* controlo de congestão



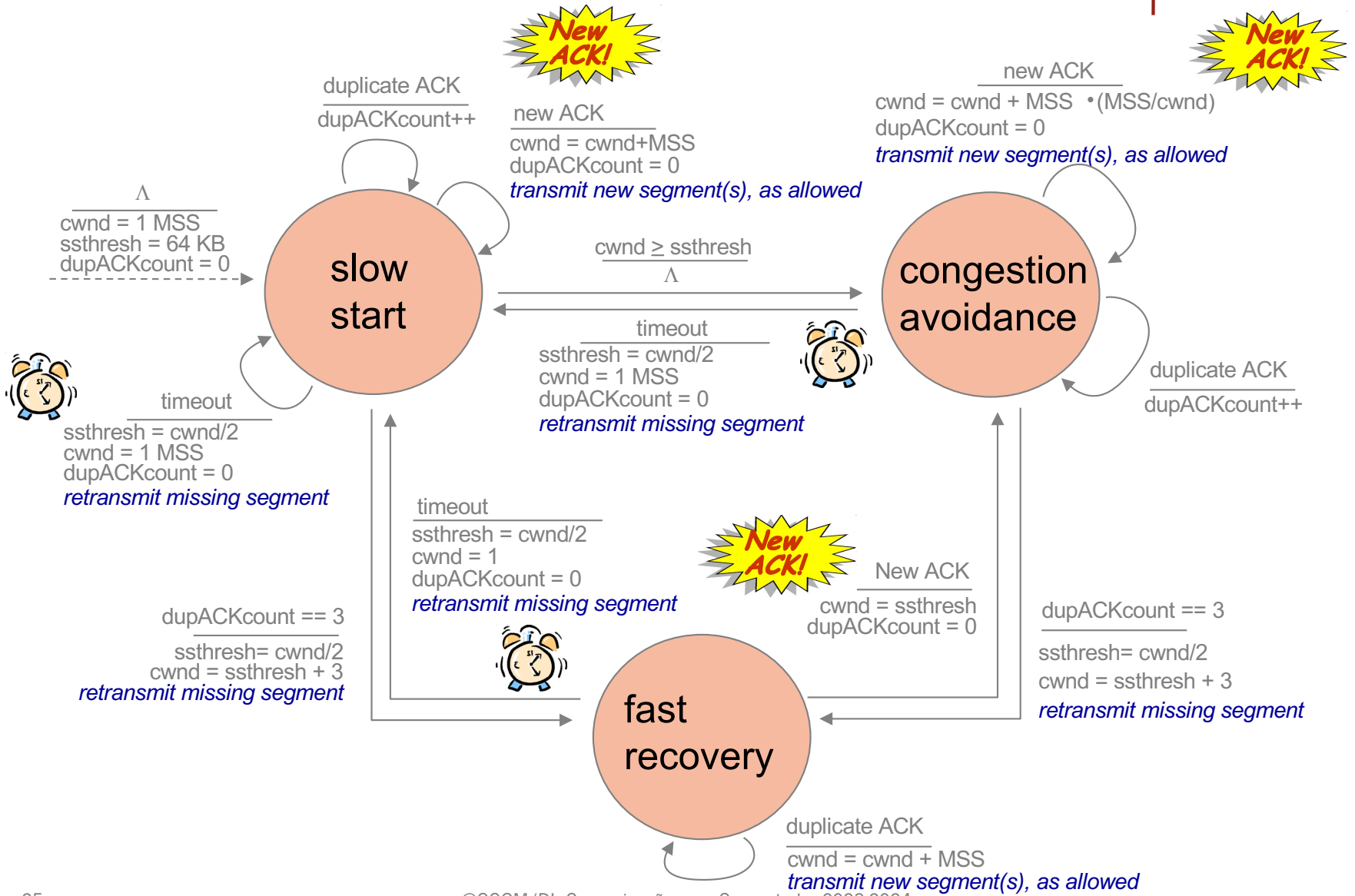
TCP/IP

TCP - *Transmission Control Protocol* controlo de congestão



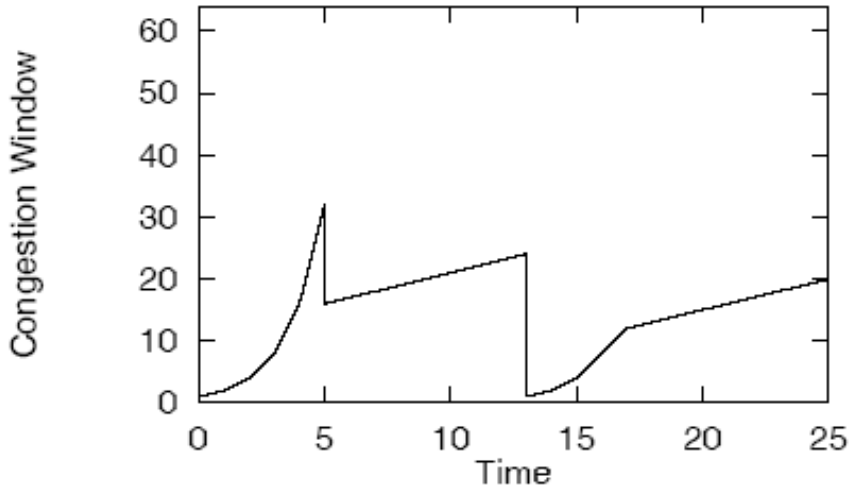
TCP/IP

TCP - *Transmission Control Protocol* controlo de congestão



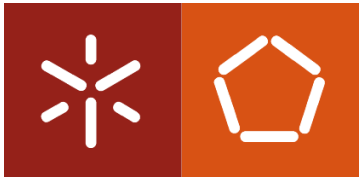
Exercício nº3

- **Considere o registo de tamanho da janela de congestão TCP mostrada na figura abaixo:**



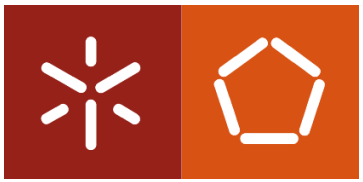
  - O que aconteceu nos instantes 5,13 e 17?
  - Atribua uma designação ao comportamento até o tempo 5, de 5 a 13, de 13 a 17 e de 17 em diante.

Exercício nº4



- **Suponha que uma aplicação no computador A estabelece uma ligação TCP com uma aplicação no computador B para enviar o conteúdo de um ficheiro. O ficheiro tem 8 000 bytes, tendo os segmentos TCP uma dimensão máxima de *S* = 500 bytes. Os computadores estão ligados por uma linha com débito *R* = 4 Mbps, com um atraso de ida e volta de *RTT* = 4 ms, onde não ocorrem erros nem perdas. Considere que todos os cabeçalhos, bem como os pacotes de pedido de ligação e confirmação de ligação, têm dimensão desprezável e admita que já são enviados dados no terceiro segmento do estabelecimento da ligação TCP.**
  - Qual a dimensão mínima da janela do emissor, em número de segmentos, para que a transmissão seja contínua?
  - Admita que a janela TCP de emissão é apenas limitada pelos mecanismos de controlo de congestionamento, isto é, o mecanismo de controlo de fluxo não intervém (os *buffers* na recepção são ilimitados). Admita que o TCP utilizado é uma versão experimental, em que apenas existe uma fase de arranque lento ("slow-start") que se inicia com uma janela de 1 segmento, mas que foi modificada por forma a que o factor de crescimento da janela de congestionamento por cada janela bem recebida seja 3 (e não 2 como nas versões habituais). Ilustrando a comunicação entre o computador A e o computador B com um diagrama temporal, determine o tempo necessário para o computador A enviar o ficheiro.

Exercício nº5



- **Suponha que uma aplicação no computador A pretende receber o conteúdo de um ficheiro de uma aplicação no computador B. O ficheiro tem 12 000 bytes. Os computadores estão ligados por uma linha com débito *R* = 4 Mbps, com um atraso de ida e volta de *RTT* = 5 ms, tendo os segmentos TCP uma dimensão máxima de *S* = 500 bytes. Considere que todos os cabeçalhos, bem como os pacotes de pedido de ligação e confirmação de ligação, têm dimensão desprezável. Admita que a janela TCP de emissão é apenas limitada pelos mecanismos de controlo de congestionamento, isto é, o mecanismo de controlo de fluxo não intervém (os *buffers* na recepção são ilimitados). Ilustrando cada situação com um diagrama temporal, qual o tempo mínimo para o ficheiro ser totalmente recebido em A, incluindo o estabelecimento e o fim da ligação nas seguintes condições:**
  - O TCP utiliza o mecanismo de arranque lento ("slow-start"), mas não usa o mecanismo de "congestion avoidance".
  - O TCP utiliza o mecanismo de arranque lento ("slow-start"), mudando para a fase de "congestion avoidance" quando a janela atinge os 4 segmentos.