Nível de transporte

5 de janeiro de 2024 14:01

Transferência confiável de dados
Controlo de fluxo
Controlo de congestão
} TCP ≠ UDP

Emissor ⟶ Recetor
Parte em segmentos
Junta segmentos

- **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

[fiavel e ordenada]
controlo de fluxo
- handshake inicial -- estabelecer seq nums
- fexo de conecçao

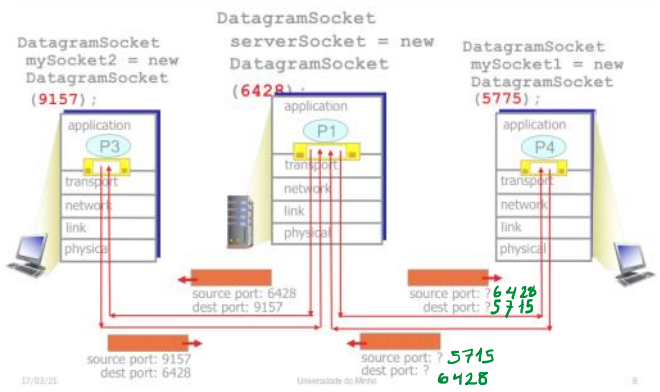
controlo de erros
controlo de congestao

O sistema terminal usa os endereços IP e os números de porta para encaminhar o segmento para o socket correto.

- **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);
```

Desmultiplexagem
– não orientado à conexão



– orientado à conexão

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

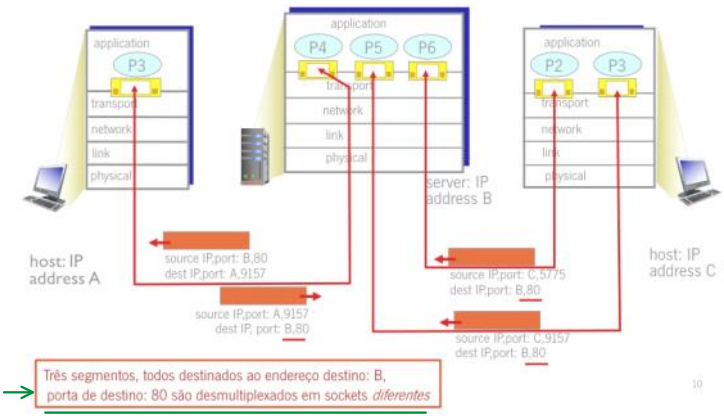
Opcionais!

```
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 **socket distinto** para cada uma delas!

servidor pode ter varias conexoes TCP distintas em simultaneo com um socket distinto



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

desmultiplexados em sockets diferentes

UDP

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

- **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.

- **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)

maior controlo sobre o envio de dados por parte da aplicaçao

não há conexao
não há estado
menor overhead

TCP

- 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.**
- Janela **– nº de octetos que o receptor é capaz de receber (controlo fluxo)**

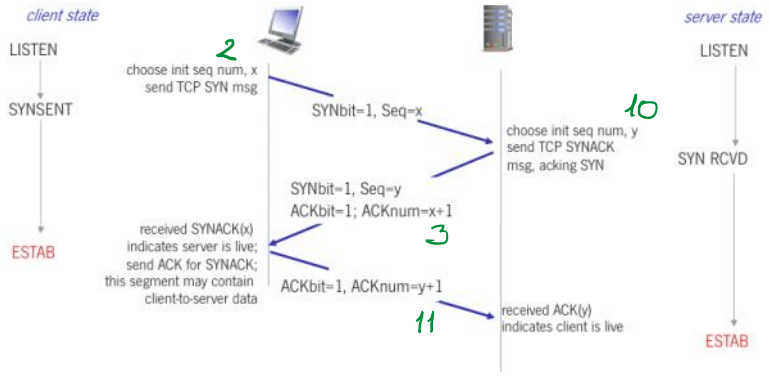
• Estabelecimento de ligação

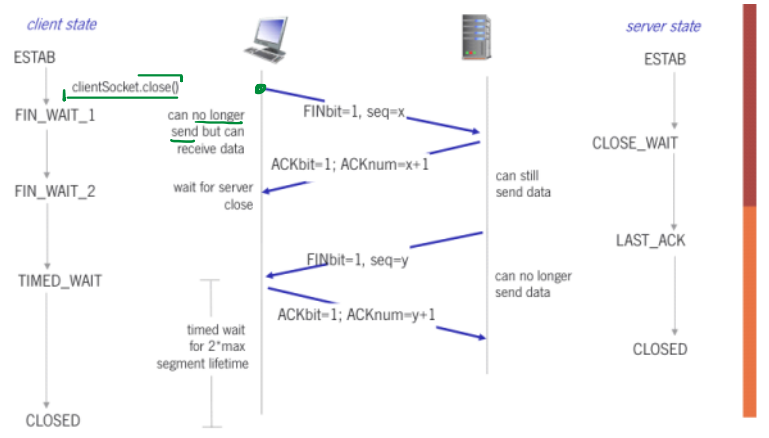
Três passos:

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

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

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





Transporte fiável
- controlo de erros
→ ARQ

| CRC | Segmento TCP | IP | MAC |

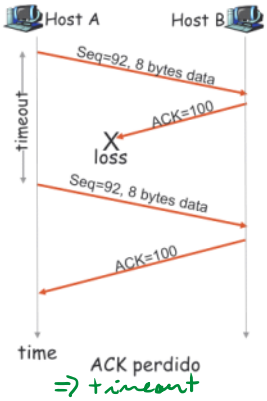
Segmentos TCP

- **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)**

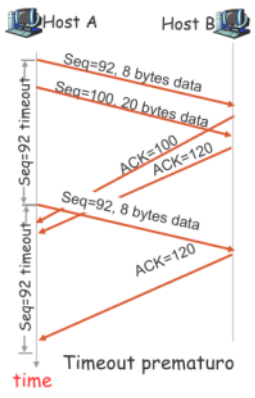
Nº ack = Nº sequência + bytes corretos lidos no segmento

Só há confirmações positivas
↳ emissor pode apenas desconfiar que um determinado segmento não chegou ao destino

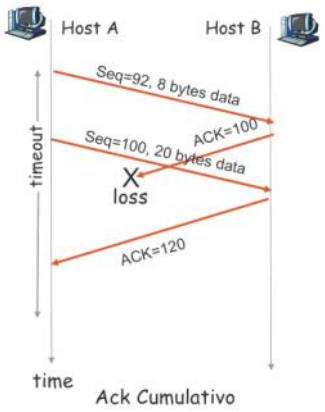
Cenários de retransmissões



ACK perdido
⇒ timeout



Timeout prematuro



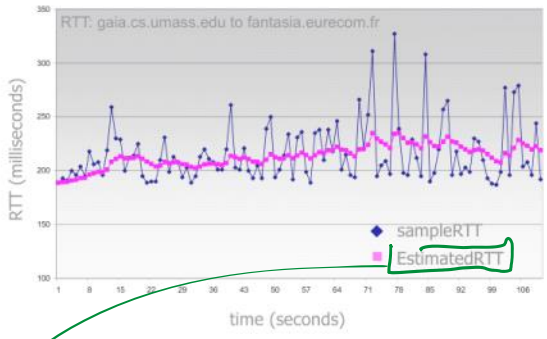
Ack Cumulativo

- **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

Estimated RTT
$= (1-\alpha) \times$ EstimatedRTT
$+ \alpha \times$ SampleRTT

Valor típico: $\alpha = 0.125$



Média móvel de peso exponencial onde a importância de uma amostra passada decresce exponencialmente

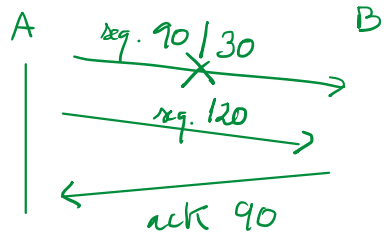
```
DevRTT = (1-β)*DevRTT +β*|SampleRTT - EstimatedRTT|
                (typically, β = 0.25)

Timeout = EstimatedRTT + 4* DevRTT
```

Margem de Segurança

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.

- **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***)

- **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



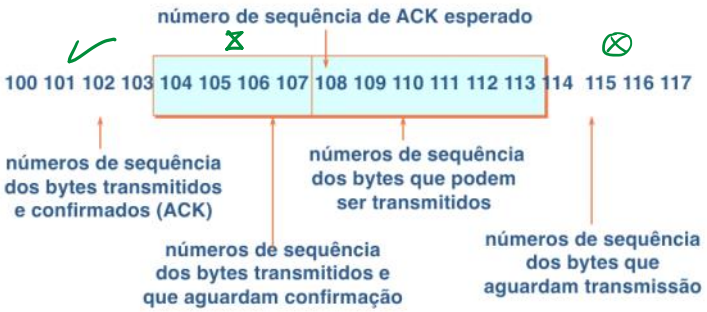
a) ack = 100 (× 3)
b) ack = 500 (× 1)
depois da retransmissão

Buffer do recetor
4096 bytes

Rcv Window = Rcv Buffer

Rcv Window = Rcv Buffer
- [LastByteRcvd - LastByteRead]

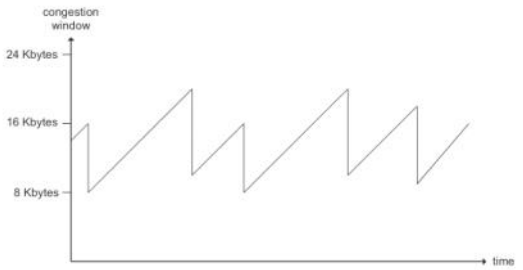
Controlo de fluxo baseado na abertura da janela anunciada no segmento recebido do parceiro



- AIMD (additive increase / multiplicative decrease)
- Slow start
- Conservativo depois de um timeout

→ Janela de congestão

- **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

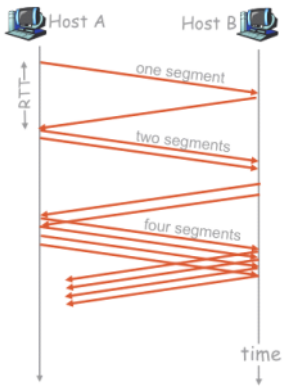
- No inicio da ligação, normalmente o tamanho da janela de congestão é igual a 1 MSS
- Sempre que é recebido um ACK, janela aumenta 1MSS (ou seja, cresce exponencialmente) até ser detectada a primeira perda ou até patamar congestão

TCP Reno (versão + recente)

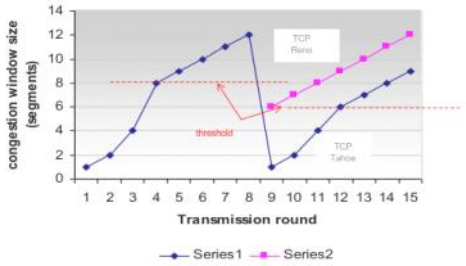
- Se a perda corresponder a um timeout a janela de congestão volta a 1 MSS e re-inicia SlowStart
- Se corresponder a ACKs duplicados é decrementada para metade, a partir daí a janela cresce de forma linear

TCP Tahoe

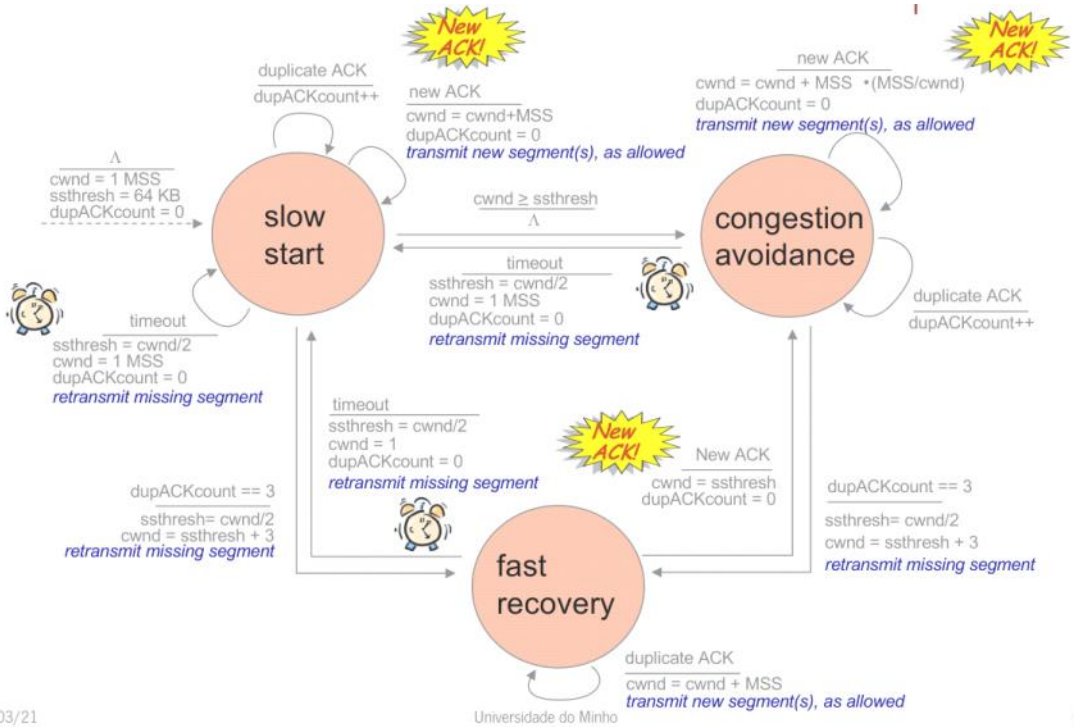
- A janela de congestão volta a 1 MSS em qualquer dos casos e re-inicia o SlowStart



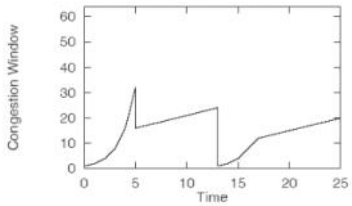
- Congestion Avoidance
  - O SlowStart progride até à detecção de uma perda ou até ter sido atingido um determinado threshold
  - Quando o threshold é atingido a janela de congestão passa a crescer de forma linear.
  - Quanto ocorre um timeout e o SlowStart é inicializado o thresold é decrementado para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



17/03/21 58



17/03/21 Universidade do Minho 62



- 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.

0..5 → slow start
5 → triple dupAck
5..13 → fast recovery
13 → timeout
13..17 → slow start
17 → cwnd ≥ ssthresh
17..25 → congestion avoidance

CA - congestion avoidance
dupAckCount = 3
⇒ fast recovery + retransmit missing segment
sstresh = w / 2
w = sstresh + 3
new ack
⇒ CA
w = ssthresh = w / 2
dupAckCount = 0

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

Exercício nº 4

A → B

File → 8000 bytes

S = 500 bytes

débito R = 4 Mbps = 4 000 000 bits/s

RTT = 4 ms

a) ?

Pah, não sei.

Exercício nº 5

File → F = 12 000 bytes

Segmento → S = 500 bytes

Débito → 4 Mbps

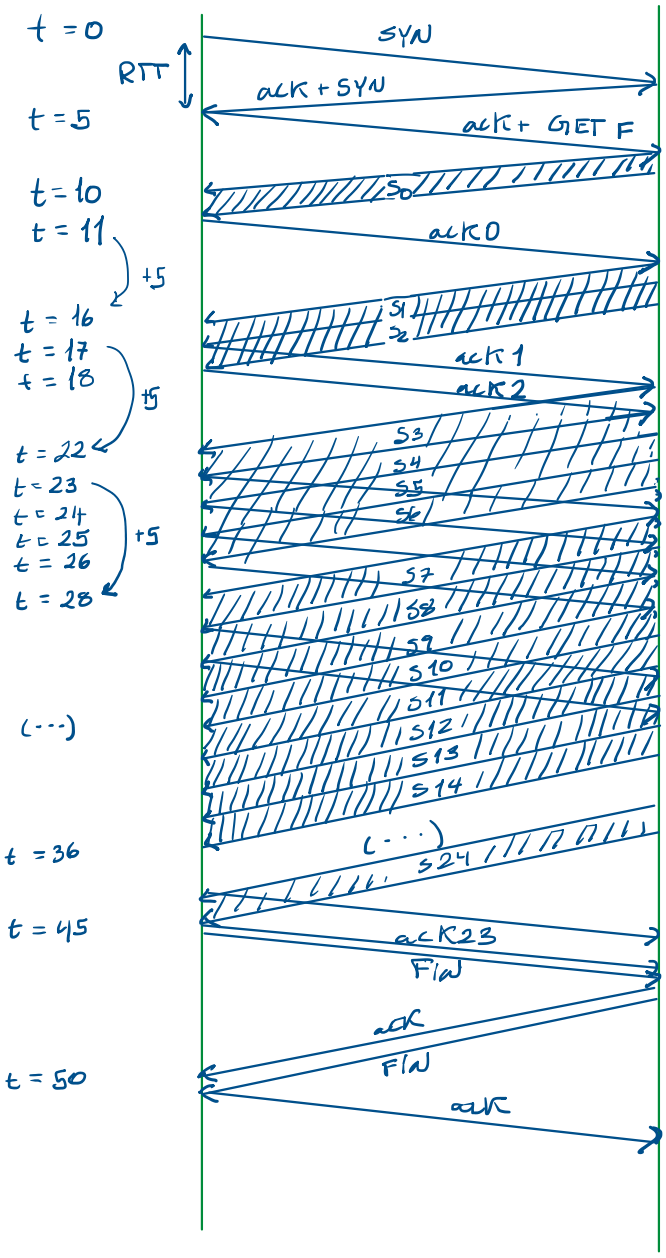
RTT → 5 ms

a) Slow start, sem congestion avoidance

A B

Cliente recebe S_0 e confirma que agora espera por S_1

Nota: o ack no TCP é feito com o número de sequência seguinte ao recebido, sendo o número de sequência o próximo byte na stream, no entanto neste exercício vamos simplificar a numeração dos segmentos, não numerando ao byte mas ao segmento, e confirmando a recebeção do segmento. Ack 0 significa que recebeu S0 e espera S1



$1\,s \;\text{——}\; 4\times10^6\ bit$

$t \;\text{——}\; 500\times8\ bit$

$t = \dfrac{500\times8}{4\times10^6} = 1\,ms$

Qual o W (tamanho de janela), a partir do qual se transmite em contínuo?

Depende do RTT... e do Tt

$$W * Tt \geq RTT + Tt$$

$$W \geq \frac{RTT+Tt}{Tt} \iff W \geq \frac{RTT}{Tt} + 1$$

$9 \geq 5/1 + 1 = 6$

True

Com a transmissão em contínuo, falta enviar os restantes segmentos: $N = \frac{F}{S} = \frac{12000\ bytes}{500\ bytes} = 24$

Transferência → 45 ms

$\Rightarrow R_{efetivo} = \dfrac{12000\times 8\,bit}{45\times10^{-3}\,s} = 2,13\ Mbps$

$Eficiência_{Coreia} = \dfrac{2.133}{4} = 53.3\,\%$

Exercício 5, alínea b) TCP com *slow start*, mas com *congestion avoidance* **TR=4**