REDES DE COMPUTADORES 2021/2022

aula 0101 – O Protocolo TCP

17/03/2022

Pedro Patinho <pp@di.uevora.pt>

(Baseado nos slides do prof. José Legatheaux Martins)

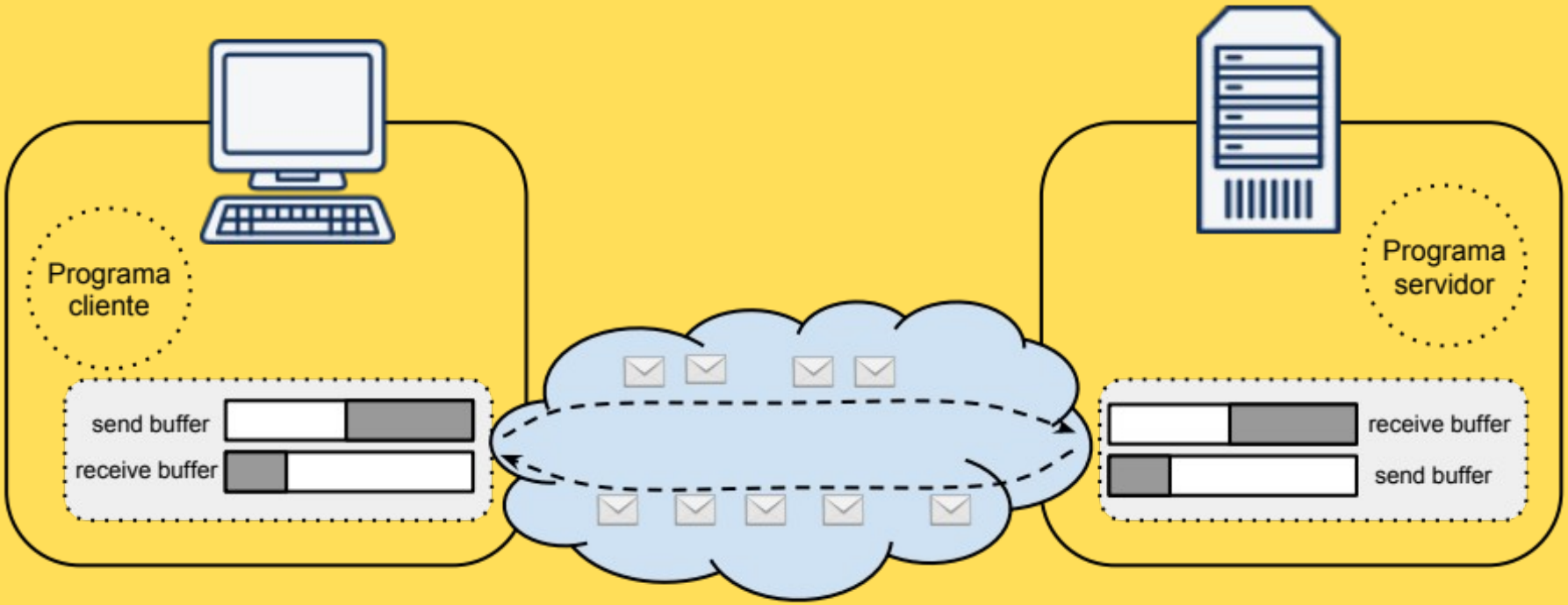
Objectivos do Capítulo

- O protocolo IP na Internet não garante a entrega de todos os pacotes, e também não garante que sejam entregues na ordem de emissão
- Ao nível transporte é necessário compensar essas deficiências providenciando um serviço de comunicações fiáveis
- É esse o papel do protocolo TCP cujas propriedades e funcionamento base serão estudadas nesta lição
- O protocolo TCP é um dos mais importantes da Internet

TCP - Transmission Control Protocol

- Serviço de comunicação entre dois computadores
  - Canal lógico dedicado entre duas aplicações
  - Sequência ordenada e fiável de bytes
  - Transmite simultaneamente nos dois sentidos
- Totalmente implementado pelos computadores, no sistema operativo (End-to-End)
  - Retransmissão dos pacotes perdidos
  - Colocação dos dados por ordem e supressão dos duplicados
  - Controlo de fluxos para evitar “afogar” o receptor
  - Controlo de saturação para adaptar a velocidade de transmissão à capacidade da rede

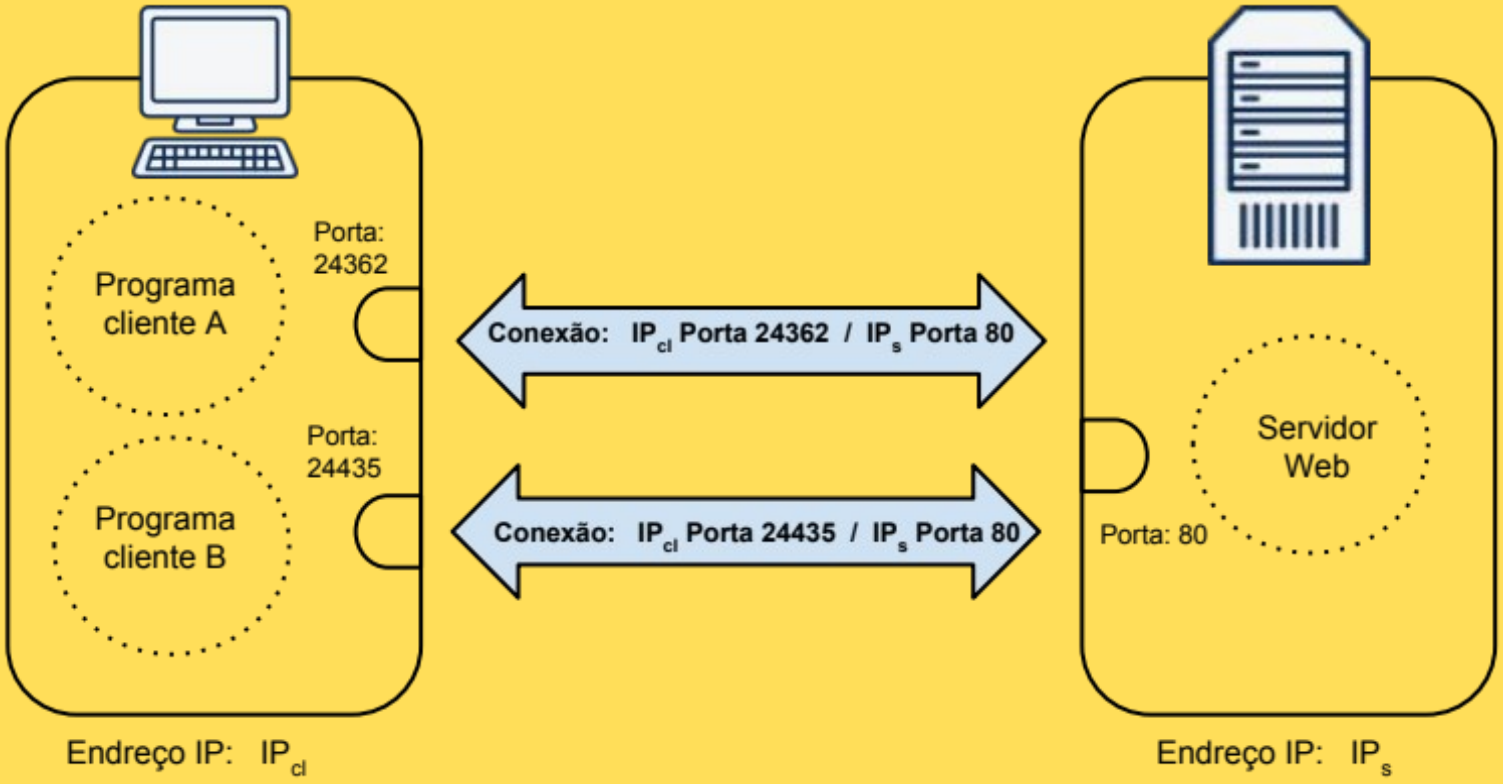
TCP é um Protocolo End-to-End



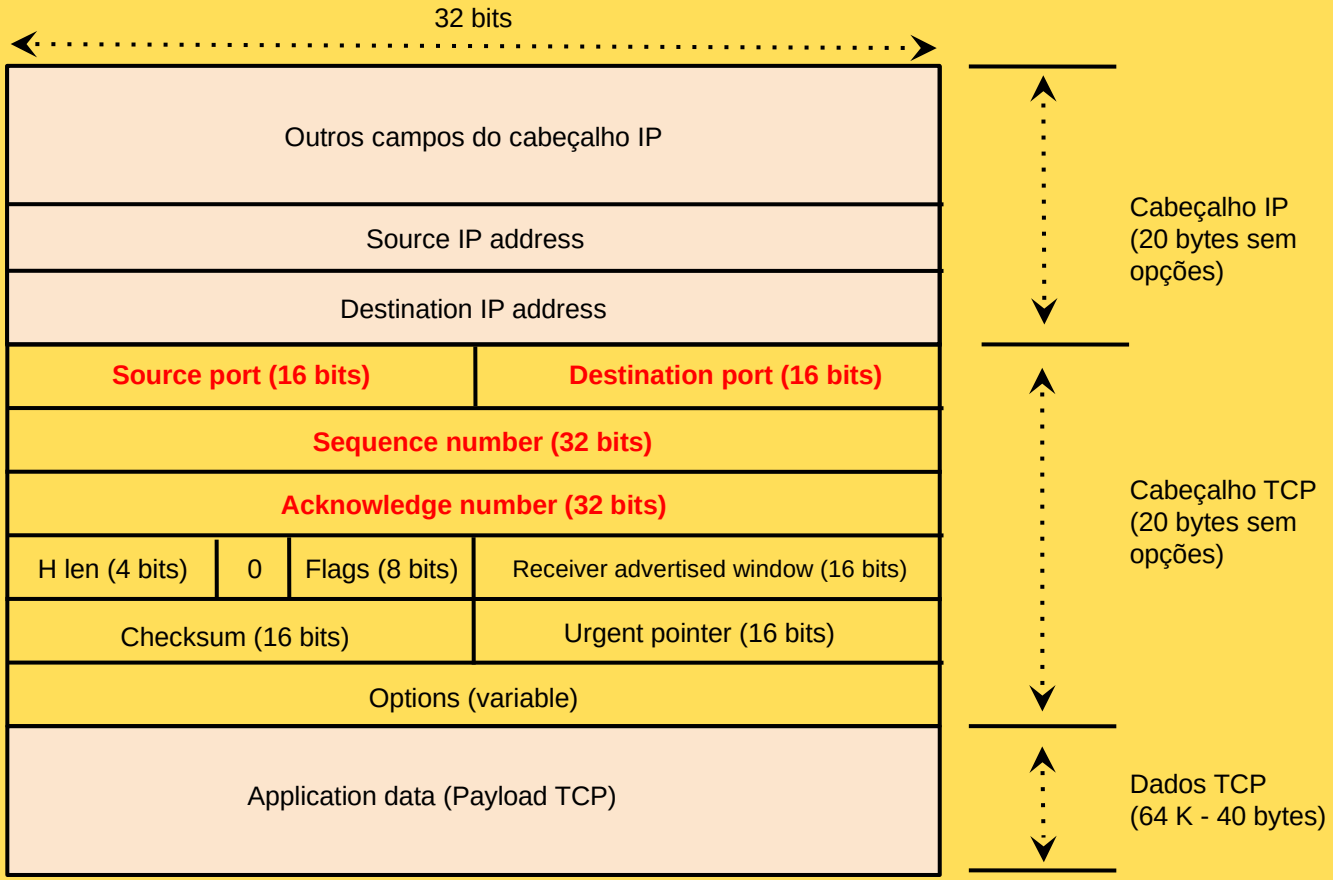
Semântica de uma Conexão TCP

- Conexão bi-direccional
- Transmite sequências de bytes de dimensão arbitrária, isto é, não tem noção de mensagem (mas a aplicação pode ter)
- Transmissão fiável dos dados
- Vistos pela aplicação, os *sockets* TCP são semelhantes a ficheiros ou *pipes*, mas a funcionalidade é distribuída a quaisquer dois computadores
- Abertura assimétricas das conexões: geralmente o servidor prepara-se para aceitar conexões, os clientes estabelecem as conexões
- Conexões TCP estabelecidas entre (endereço IP, porta) e (endereço IP, porta)

Conexão TCP = (IP_1,$Porta_1$, IP_2, $Porta_2$)



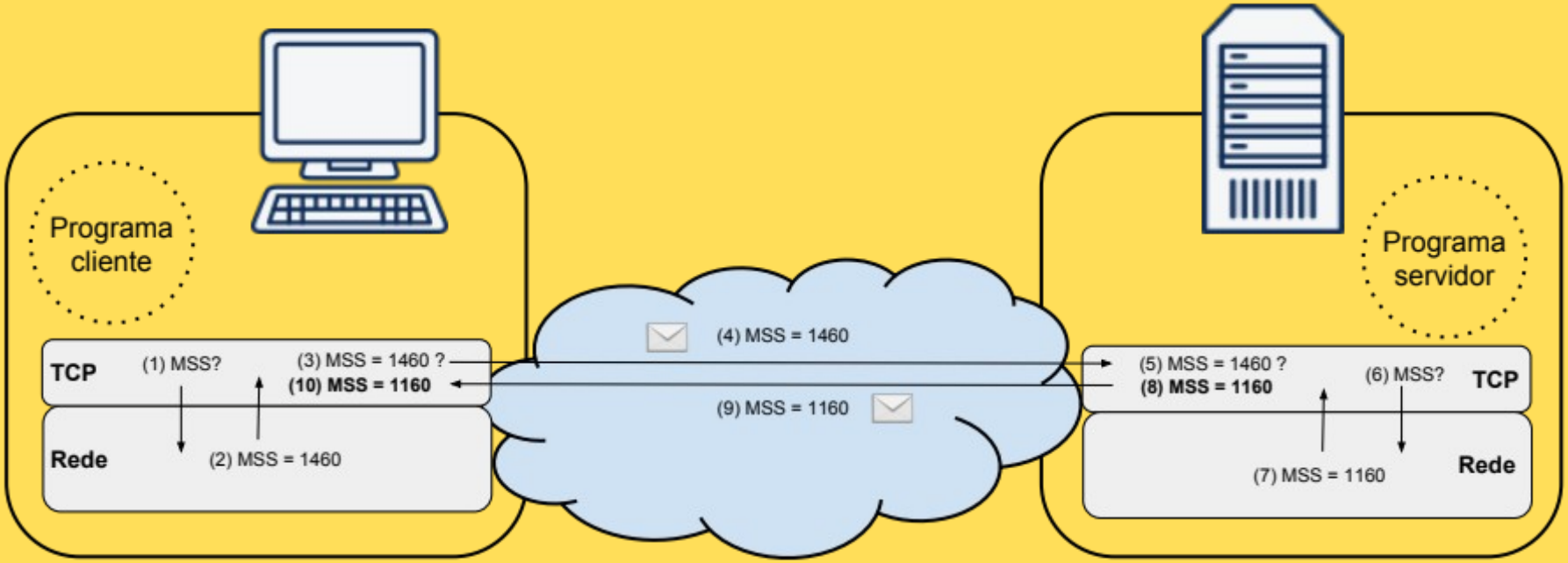
Segmento TCP



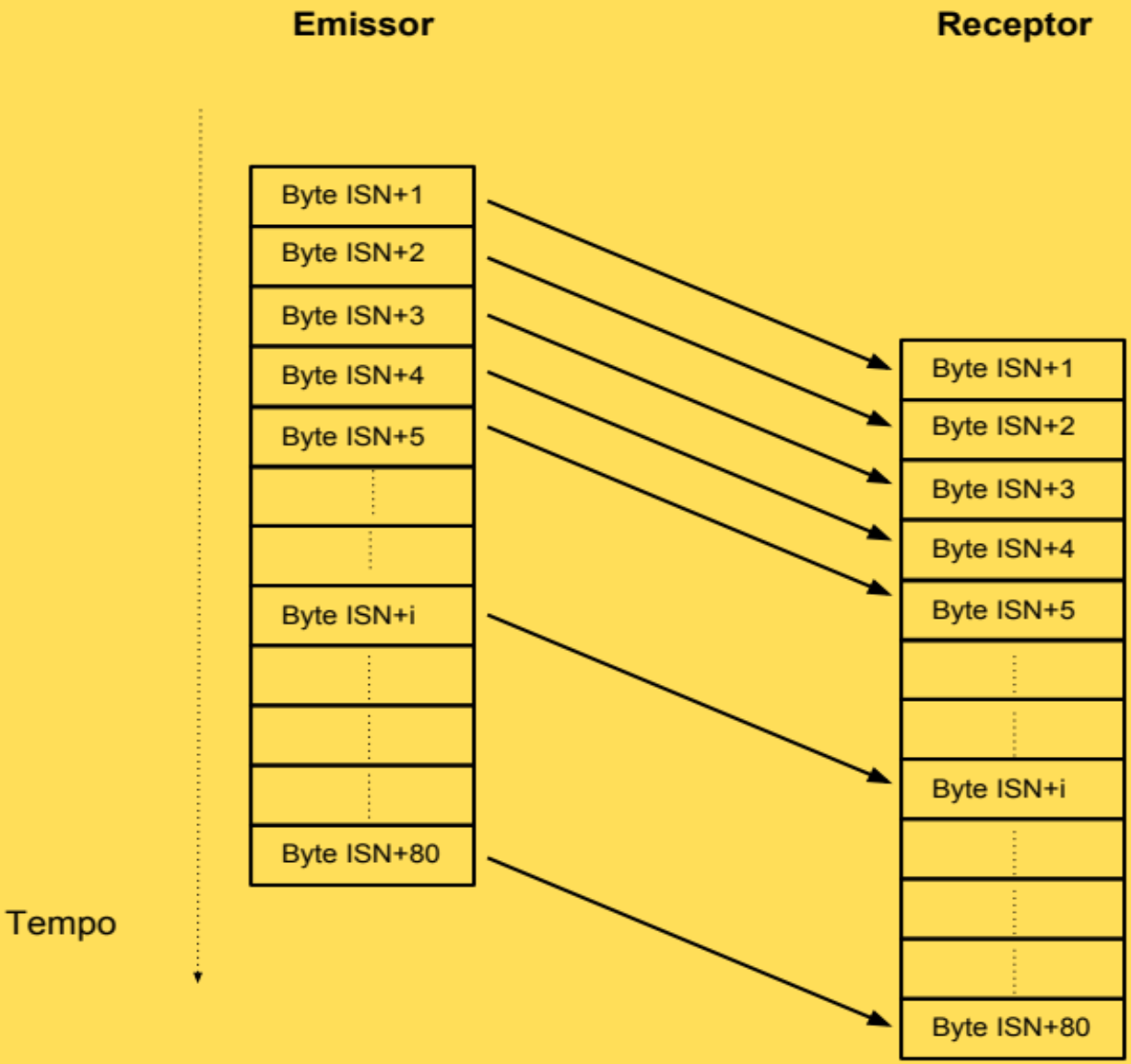
Dimensão dos segmentos TCP

- Um segmento pode ter uma dimensão arbitrária de 0 a MSS (MSS - Maximum Segment Size) bytes
- A dimensão máxima dos segmentos de uma conexão é negociada entre os dois extremos de forma a tentar que os segmentos caibam dentro de um pacote IP (sem fragmentação). Alguns valores típicos são 1460 bytes, 512, ...
- Geralmente cada extremo indica inicialmente aquele que julga ser o MSS mais adequado. Será selecionada a dimensão mais baixa das duas propostas
- Tal não garante que o MSS não provoque segmentação, dado não se saber o MTU (Maximum Transfer Unit) de todos os canais intermédios
- Existe um protocolo que permite a ambas as partes determinarem o MSS mais adequado com base no uso de ICMP

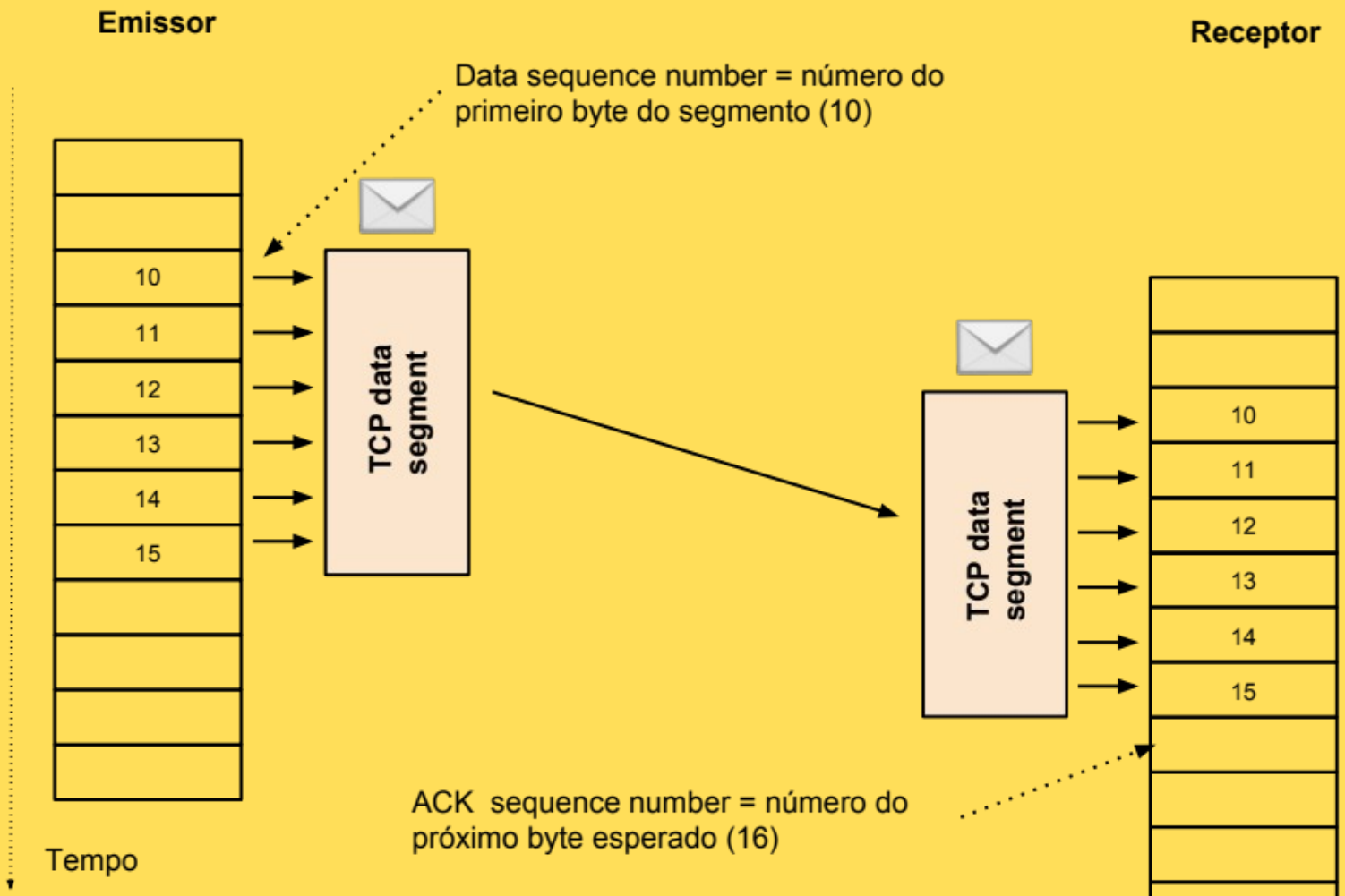
Maximum Segment Size (MSS)



Uma Conexão TCP Transporta Sequências de Bytes



Segmentos e Números de Sequência



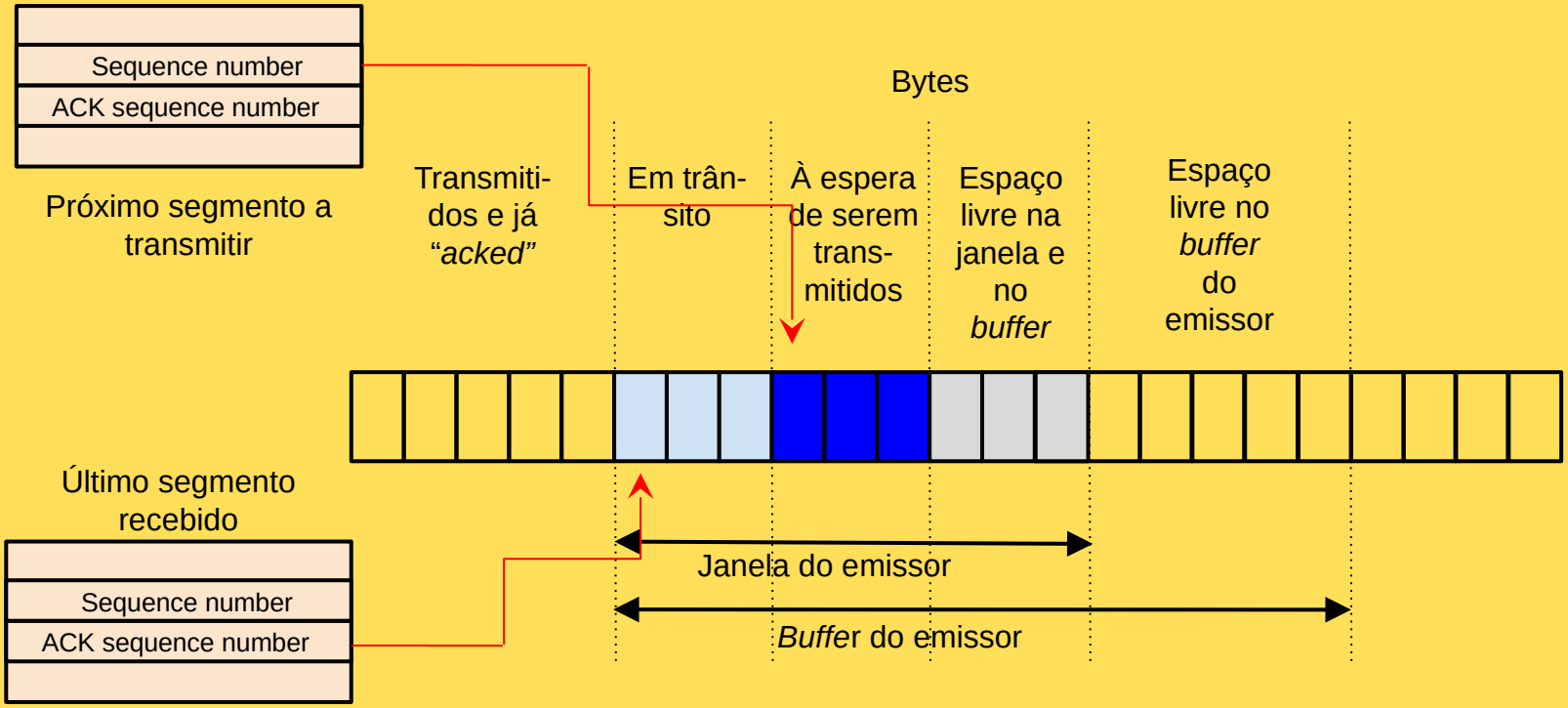
Initial Sequence Number (ISN)

- É o número de sequência do 1.º byte
  - Porque não sempre 0 ou 1?
  - Como garantir que não há confusão entre segmentos de conexões sucessivas?
- As conexões distinguem-se pelos números dos endereços IP e das portas
  - Mas ambos podem ser reutilizados e poderiam no limite introduzir confusão
  - Utilizar um valor fixo também ajudaria um atacante
- O TCP muda o ISN de cada nova conexão
  - Para isso é usado um gerador de números aleatórios

Sockets e *Buffers*

- A extremidade de cada socket TCP tem dois *buffers* associados
  - O *buffer* (parecido com a janela) de emissão
  - O *buffer* (parecido com a janela) de recepção
- O *buffer* de emissão pode ter muitos bytes ainda não enviados para permitir à aplicação progredir momentaneamente, mais depressa que o TCP
- O *buffer* de recepção pode ter muitos bytes já ACKed, mas que ainda não foram consumidos pela aplicação responsável pelo consumo dos dados

Buffer ≠ Janela do Emissor



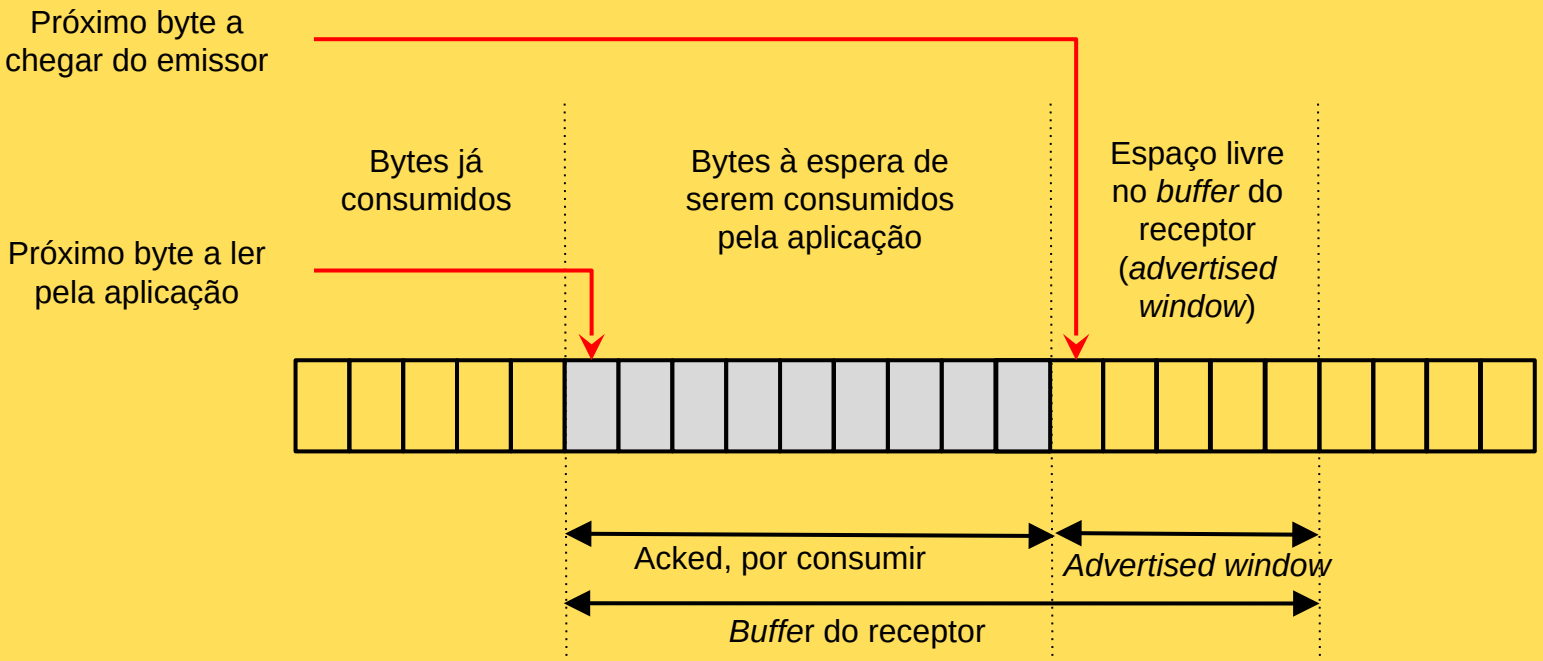
Gestão Inteligente do Envio de Segmentos

- Um novo segmento é transmitido:
  - Quando existem MSS bytes disponíveis para transmitir e a janela do receptor tem espaço (MSS = Max Segment Size)
  - Existem menos de MSS bytes disponíveis mas passou demasiado tempo (geralmente 200 ms) e há espaço na janela do receptor
  - Empurrados ("*pushed*") pela aplicação e há espaço na janela do receptor (chamada *flush*() ou opção SI_NODELAY)

ACKs e Retransmissões

- Por defeito todos os ACKs são cumulativos e o funcionamento do protocolo é do tipo GBN (Go-Back-N)
- Logo, em caso de alarme são retransmitidos todos os dados ainda não ACKed desde o mais velho segmento que desencadeou o *timeout* (GBN)
- O receptor pode guardar dados fora de ordem, isto é, o *buffer* de recepção pode ter buracos. A norma não obriga mas é a opção adoptada atualmente
- Os ACKs viajam nos segmentos enviados com os dados transmitidos no sentido contrário (*piggybacking*)
- Pode funcionar num modo semelhante ao SR em opção, enviando ACKs cumulativos e selectivos em simultâneo

Buffer do Receptor



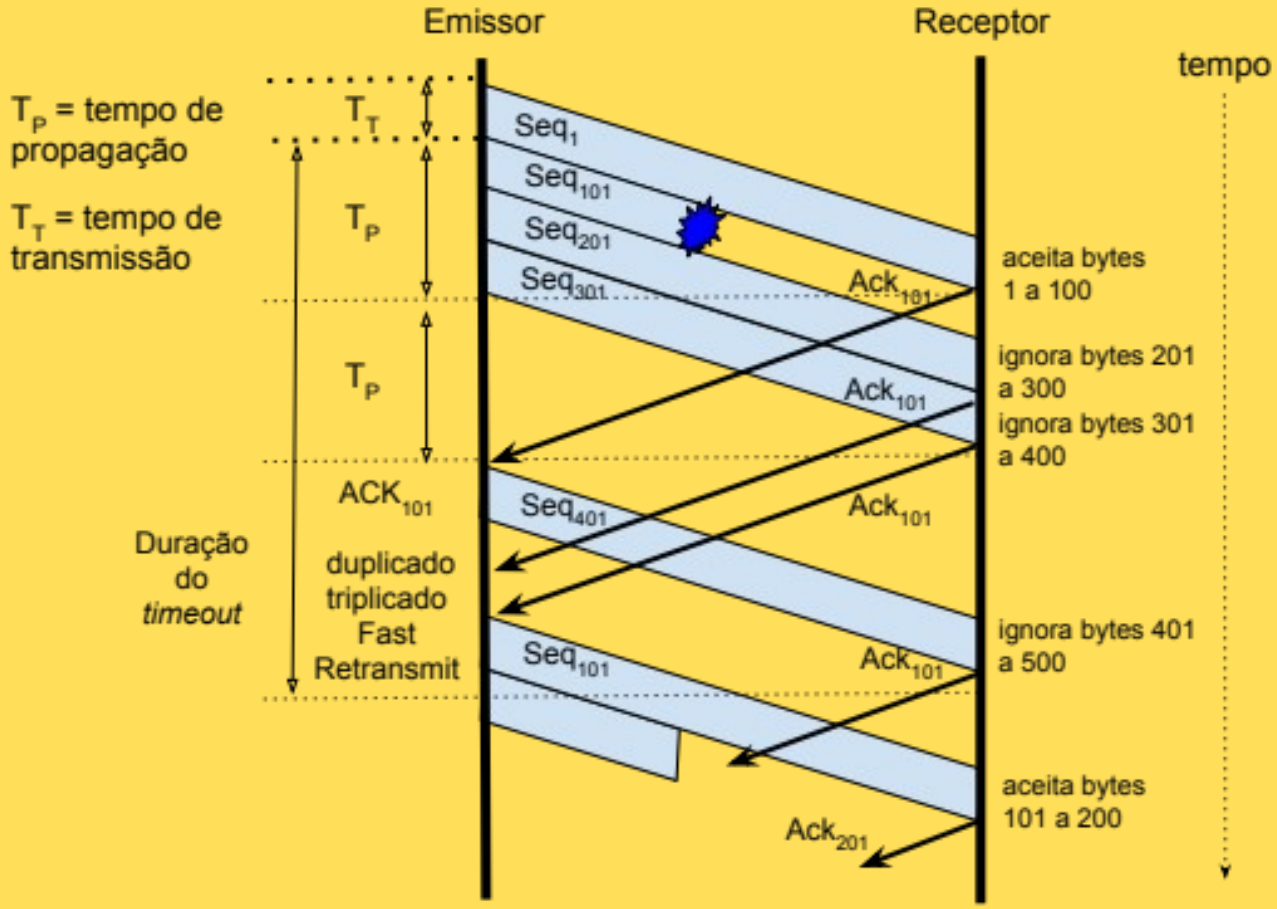
Gestão inteligente de ACKS cumulativos

Evento	Acção do receptor
Segmentos recebidos ordenados com números de sequência esperados, todos os segmentos anteriores já ACKed	*Delayed ACK*: esperar até 500ms por outro segmento na ordem. Se este não chegar enviar ACK
Segmentos recebidos na ordem esperada, todos os segmentos anteriores já ACKed menos 1	Enviar imediatamente um ACK cumulativo
Segmento fora de ordem (gerando um buraco na janela de recepção)	Enviar ACK duplicado, indicando o nº de sequência do próximo byte esperado (o 1º do 1º buraco)
Chegada de um segmento que preenche parcial ou totalmente um buraco	Envio imediato de um ACK cumulativo

Fast Retransmit

- Se os valores de *timeout* são altos quando comparados com o RTT (ver a seguir) a recuperação das falhas pode revelar-se muito lenta
- Como vimos o TCP ao receber segmentos que geram "buracos" no receptor (porque um segmento anterior se perdeu ou está atrasado e fora de ordem) envia por defeito ACKs cumulativos, correspondentes aos dados recebidos correctamente até momento
- Esses ACKs cumulativos têm necessariamente o mesmo número de sequência
- Quando o emissor recebe 3 ACKs seguidos com o mesmo número de sequência, comporta-se como se o *timeout* tivesse disparado e reemite imediatamente (GBN), ou seja, 3 ACKs repetidos idênticos são equivalentes a um NACK (Negative ACK)

Fast Retransmit (sem SR)



Fast Retransmit é Eficaz?

- A eficácia é máxima se
  - Com muitos pacotes transmitidos para a frente, isto é, com uma janela grande
  - Os *timeouts* >> RTT
  - Com transferências longas
- Implicações para o tráfego Web
  - Se as conexões para sites Web transmitirem relativamente poucos dados (e.g., 10 pacotes ≈ 15 Kbytes) não há oportunidade para serem enviados muitos pacotes emitidos de avanço
  - Logo, é de todo o interesse privilegiar conexões “longas”

Duração dos Alarmes

- Tem que ser maior que o RTT (e.g. RTT + delta)
  - Se demasiado longo, a recuperação dos erros é lenta
  - Se demasiado curto desperdiça a capacidade da rede
- Mas o RTT é variável, logo é necessário estimar o valor do RTT médio
- Mas adaptando-se às variações do RTT, isto é
- Cada vez que se estima o RTT, calcula-se o valor da variável estimatedRTT fazendo uma média pesada com o valor passado

$$estimatedRTT = estimatedRTT\ (1-\alpha) + \alpha\ .\ sampledRTT$$

Com $\alpha = 0{,}125$

$$estimatedRTT = estimateddRTT\ .\ (1-0{,}125) + 0{,}125\ .\ sampledRTT$$

Que Valor Usar?

- O valor de estimatedRTT é demasiado curto quando as variações momentâneas são significativas, logo temos de introduzir uma margem de segurança, multiplicando-o por um factor
- Esse factor deve ser proporcional ao desvio das amostras obtidas face ao estimado, isto é, quanto maior a variação, maior a margem de segurança

devRTT = (1-β) . devRTT + β . |sampledRTT-estimatedRTT|,

com β=0,25:

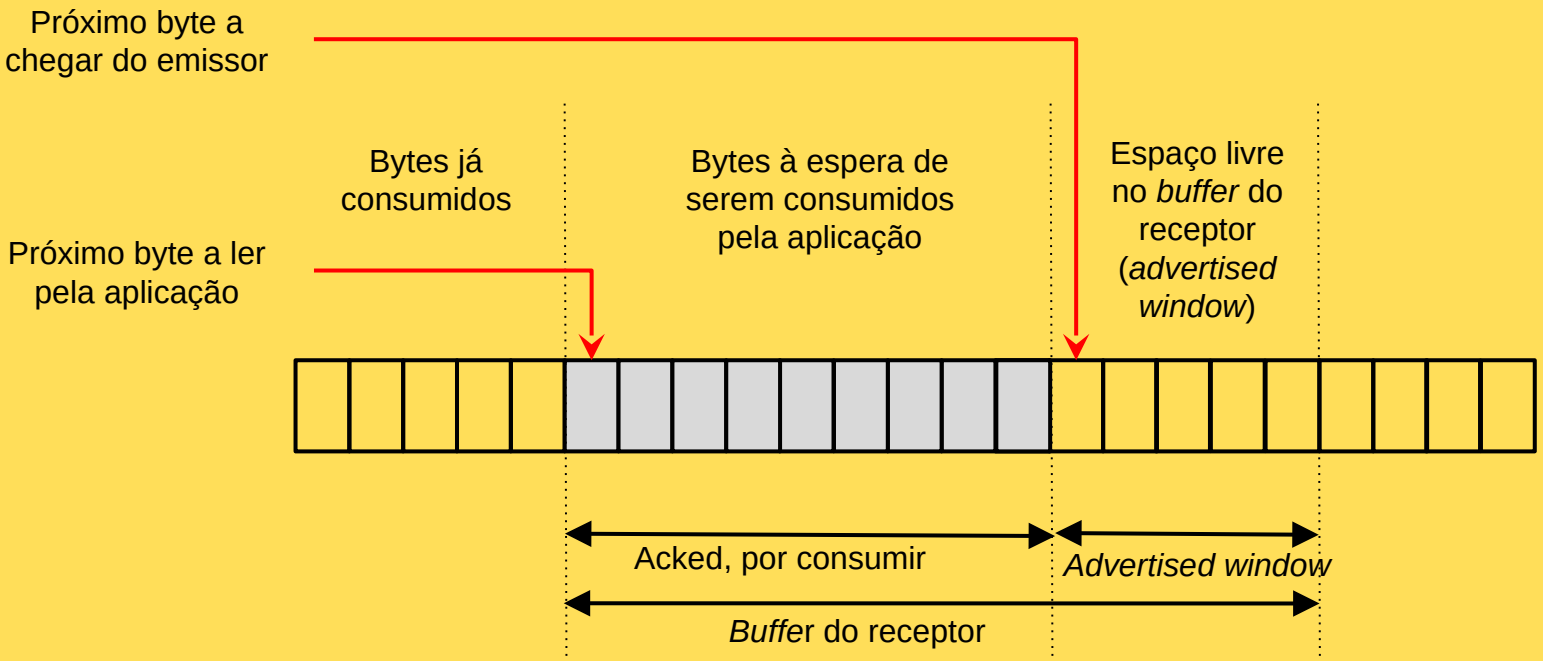
devRTT = (1-0,25) . devRTT + 0,25 . | sampledRTT-estimatedRTT |

timeoutValue = estimatedRTT + 4 . devRTT (RFC 2988)

Controlo de Fluxo

- Se a aplicação não consumir os dados recebidos a um ritmo maior ou igual ao que eles chegam, o *buffer* do receptor vai enchendo
- Até não haver espaço e os novos segmentos que chegam são desperdiçados
- O emissor notará isso mais tarde (pelos ACKs ou pelo *timeout*)
- Mas de que serve continuar a emitir se a aplicação não lê os dados? (e.g. o utilizador está a analisar os dados recebidos)

Buffer do Receptor



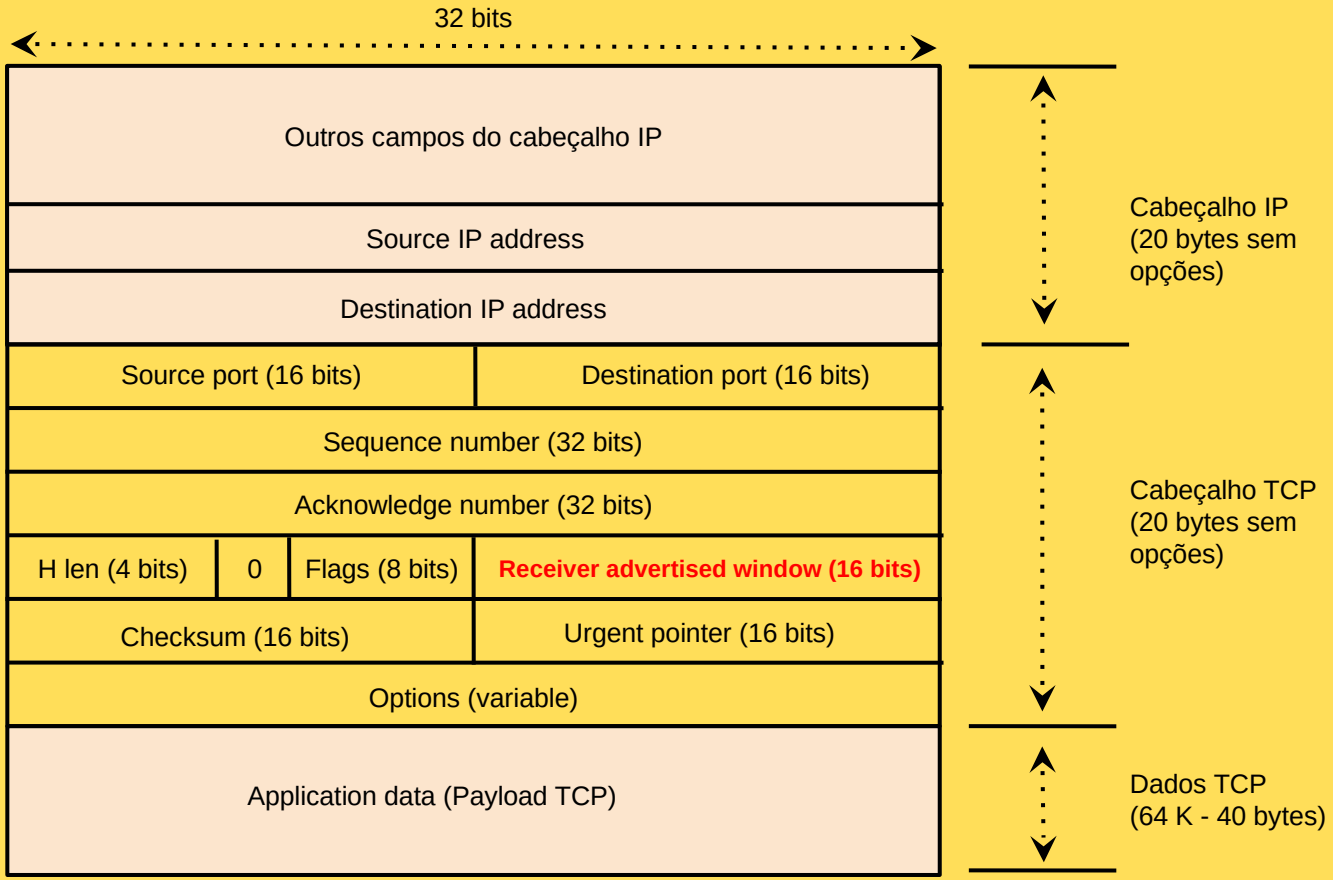
Advertised Window

- Sempre que um segmento é emitido num sentido da conexão, o receptor avisa o outro lado, o emissor, de quanto espaço tem livre no *buffer* de recepção (*Receiver Advertised Window*)
- O emissor refreia então o seu ritmo de emissão de tal forma que

LastByteSent – LastByteAck ≤ Last Receiver Advertised Window

- Dada a dimensão do campo, é interessante analisar o valor máximo da janela

Segmento TCP



Dimensão Mínima da Janela com RTT = 100 ms

Débito extremo a extremo	Dimensão mínima da janela
100 K bps	≈ 1250 bytes
1 M bps	≈ 12,5 K bytes
10 M bps	≈ 125 K bytes
100 M bps	≈ 1,25 M bytes
1000 M bps	≈ 12,5 M bytes

Estabelecimento da Conexão

- É baseado num protocolo especial designado por *three-way handshake*
- Durante o mesmo as partes comunicam o ISN (*Initial Sequence Number*) e negoceiam o MSS (*Maximum Segment Size*) e outras opções
- Outras opções típicas:
  - Factor de escala do campo *Receiver Window* para comportar janelas muito grandes (*window scaling factor*)
  - Uso opcional de *S*ACKs (Selective ACKs)
  - *Probing e Timestamps* (RFC 1323)

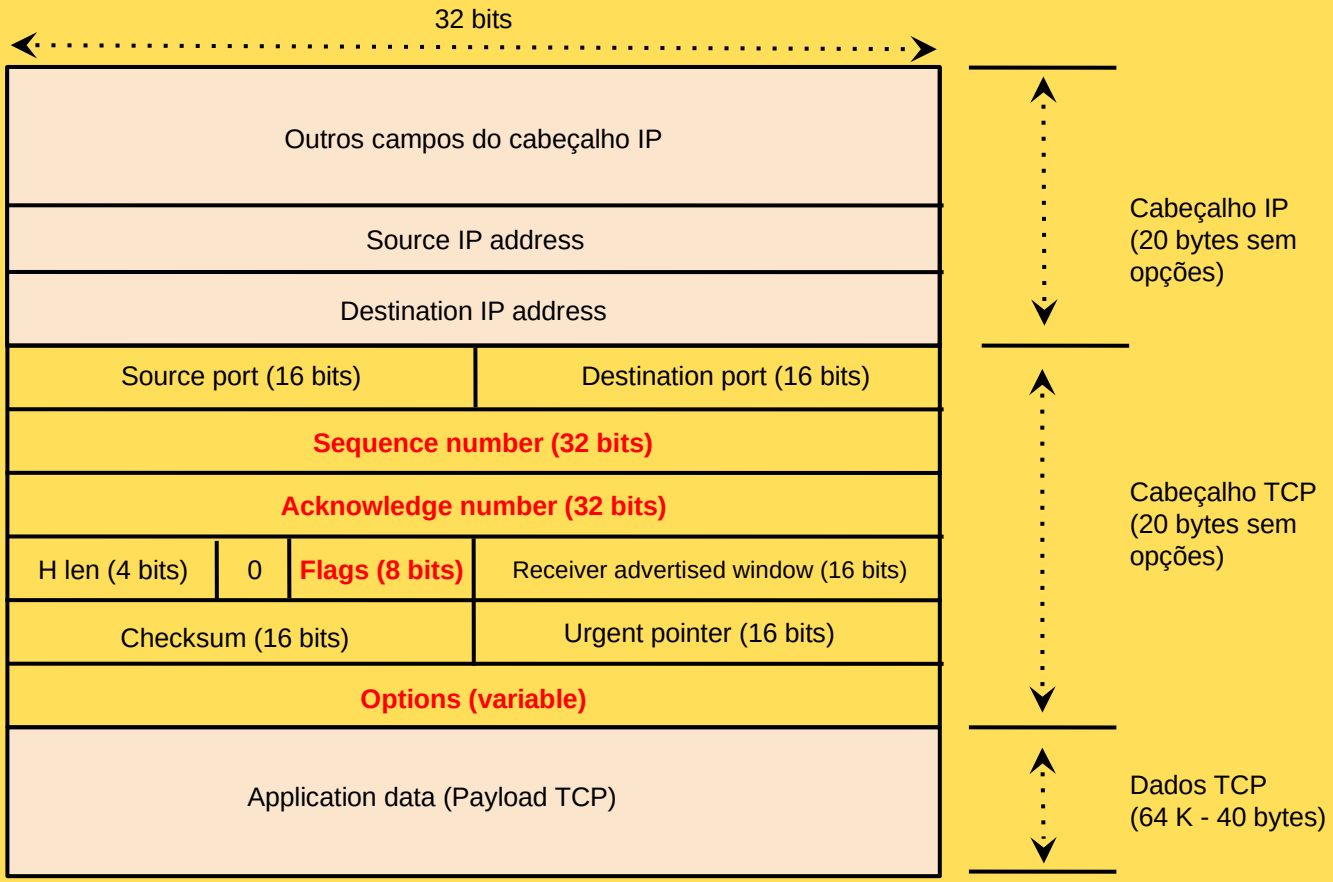
Segmento TCP

Flags:

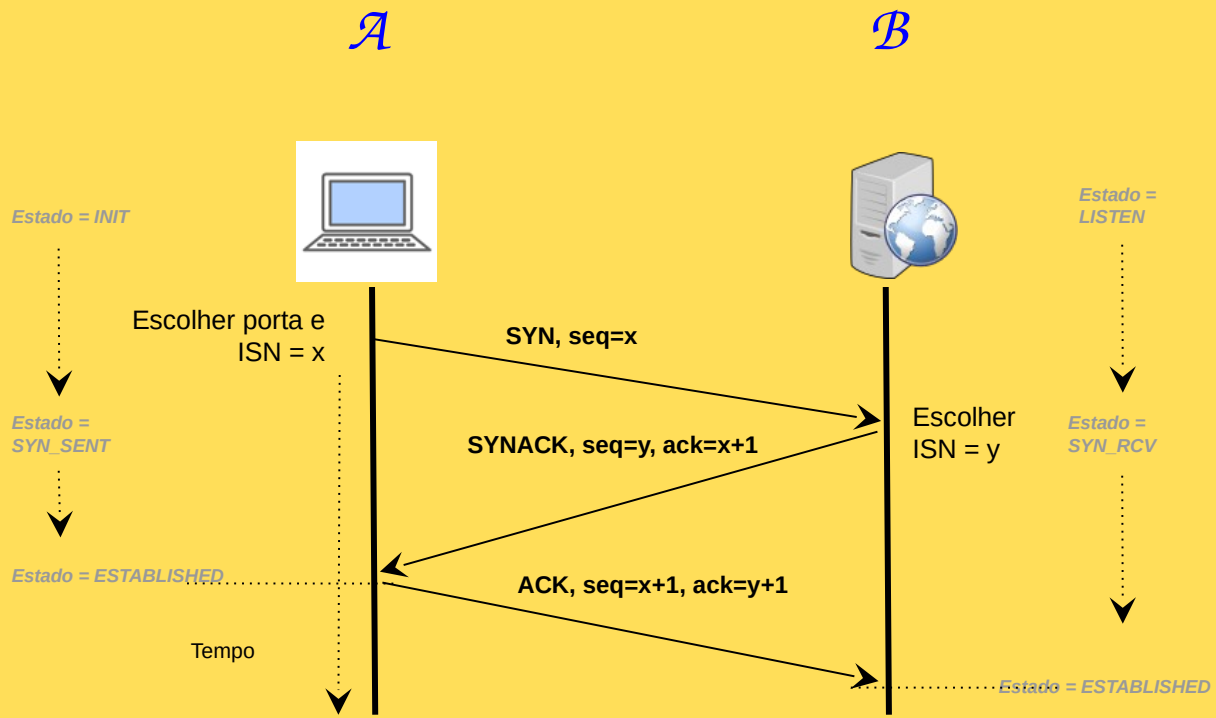
SYN
FIN
RST
PSH
URG
ACK

Options:

MSS
SACK
W. SCALE
T. STAMP



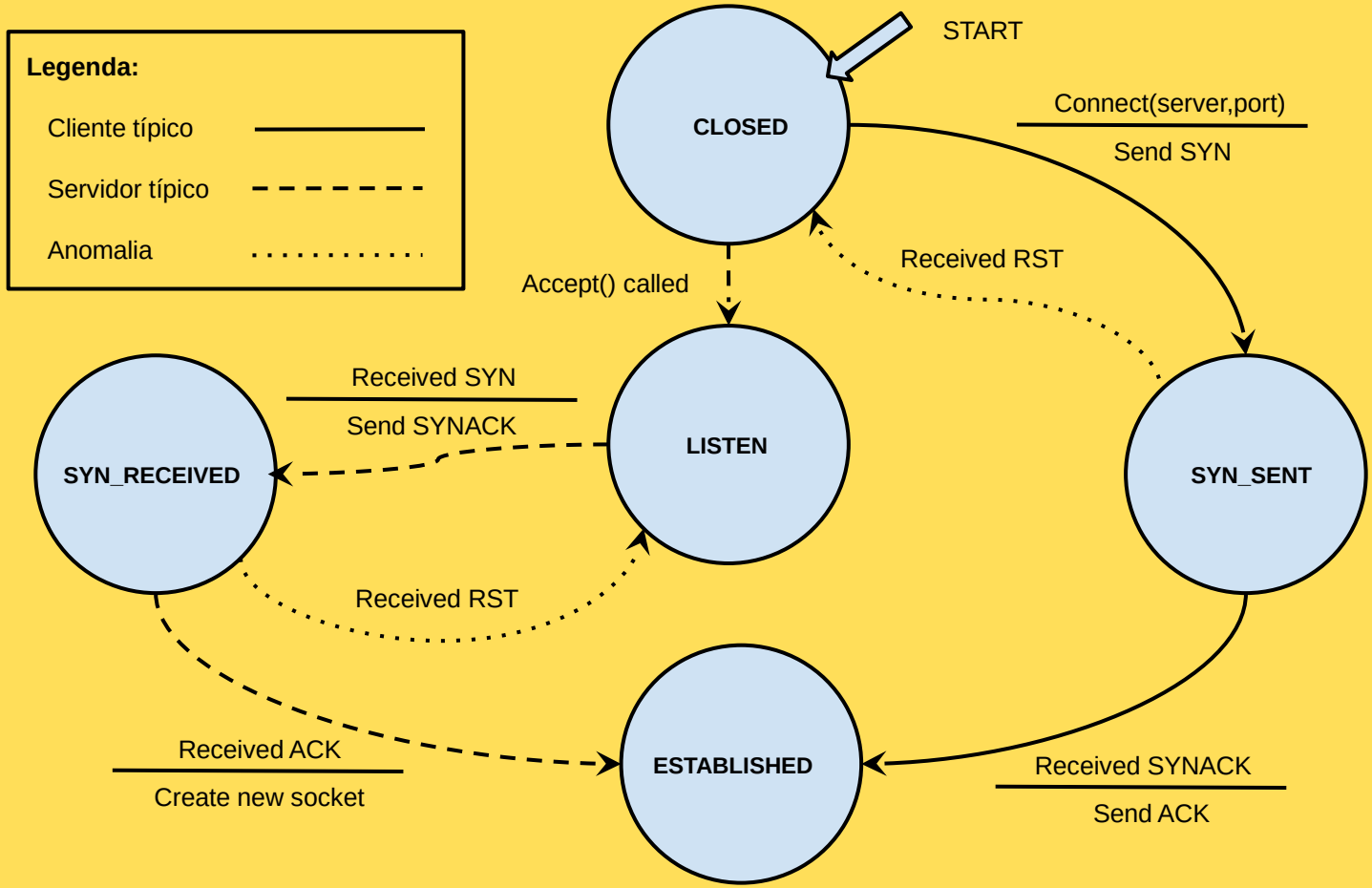
Estabelecimento da Conexão



Como Funciona

- A envia um segmento SYN com as suas propostas
- B responde com um segmento SYN+ACK com as suas alternativas
- A envia um ACK a fechar a negociação
- Se A não recebe o segmento SYN+ACK retransmite o segmento SYN

Evolução do Estado da Conexão



E Se o SYN Se Perde?

- O segmento SYN inicial pode perder-se
  - Na rede houve um problema ou
  - O servidor não aceita mais conexões (parâmetro *LISTEN for # new connections*, também acessível nos *sockets* Java)
- Nenhum SYN+ACK chegará ao cliente
  - O cliente espera um certo limite de tempo (*timeout*)
  - E reemite ou desiste (após algumas tentativas)
- Que valor usar para o *timeout* ?
  - Tipicamente alguns segundos, por exemplo 3 segundos, porque nada se sabe do RTT no início
  - Caso o SYN se perca, a abertura seria muito demorada pelo que modernamente se baixou o valor para 1 segundo

E Se o SYN+ACK Se Perde?

- O cliente acabará por voltar a emitir e tudo recomeça pois o servidor reconhecerá a conexão repetida
  - A conexão acabará por se estabelecer
- Mas que acontece se o cliente for batoteiro e não responder ao SYN+ACK com o ACK final (ataque *SYN flood*)
  - O servidor terá de esperar um certo tempo (*timeout,* com por exemplo o valor de 10 segundos) e depois desistir daquela conexão dita "meia aberta"
- Qual o valor do *timeout*?
  - O servidor espera um tempo fixo pois não conhece o RTT
  - O tempo que espera perante um ataque *SYN flood* determina o espaço desperdiçado na tabela de conexões abertas e *buffers*, etc.

Contra Medidas Para o SYN flood

- Usar SYN cookies, isto é, números de sequência iniciais (ISNs) especiais do lado do servidor
  - O servidor responde com ISN (especial), ver abaixo
  - Mas o servidor memoriza muito pouca informação sobre a conexão aberta (e.g. *client IP, client Port, server IP, server Port, ISN*), ou mesmo apenas o ISN
  - Um cliente correto responde com um ACK=ISN+1
  - O servidor analisa esse valor e constrói então o estado total da conexão pois tem a certeza que o cliente não é falso. Só nessa altura constrói todas as outras estruturas de dados sobre a conexão

ISN = Hash (client IP, client Port, server IP, server Port, segredo)

Perda do SYN ou SYN+ACK e Browsers Web

- O utilizador dá um URL ao browser e carrega em *enter*
  - O browser abre uma conexão TCP para o servidor
- O utilizador espera alguns segundos, mas vários segundos é muito tempo
- Impacienta-se e carrega em *reload*
- O browser abre uma nova conexão
  - o que se traduz em enviar um novo SYN e no abandono da antiga conexão
  - Isto é, o utilizador limita na prática um *timeout* muito alto
  - A antiga conexão não aberta acabará por desaparecer
  - Que repercussões haverá sobre o problema da confusão entre a nova e a antiga conexão ? Nenhuma se as portas não forem reutilizadas ou o ISN avançar adequadamente

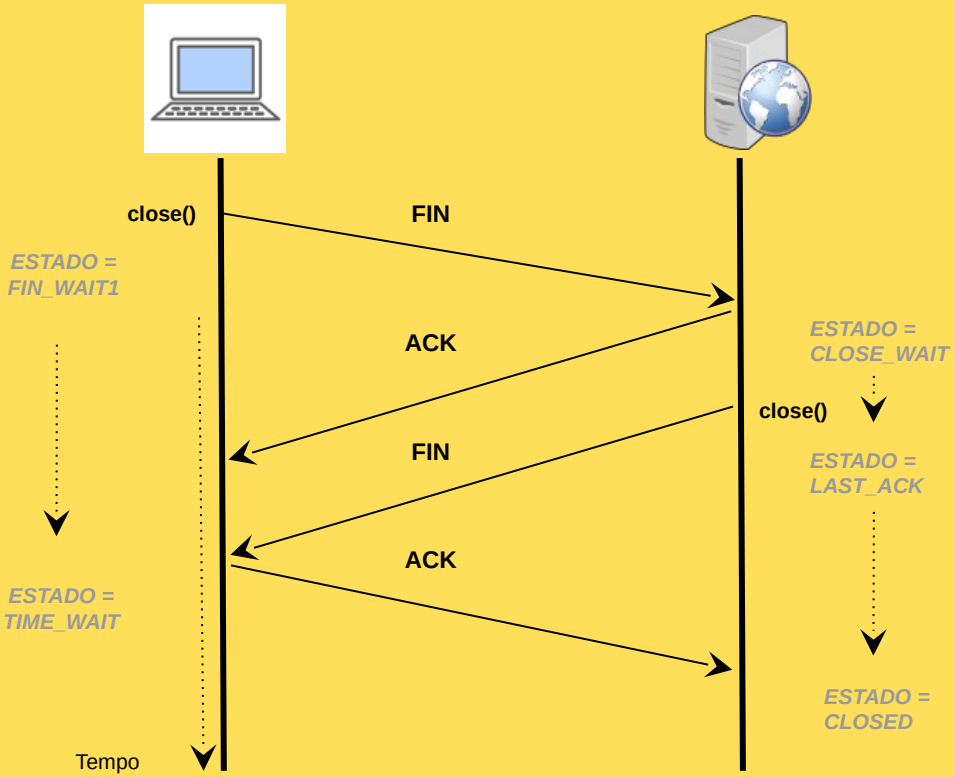
Fecho da Conexão

- Fechar uma conexão não é fácil se exigirmos o acordo de ambas as partes
- “Eu quero fechar” porque já não tenho nada para enviar
- Mas a outra parte tem também de estar de acordo e não ter também nada mais para enviar
- Chegar a este acordo numa situação em que se podem perder pacotes é complicado
- Solução do TCP: fechar cada uma das partes de forma independente

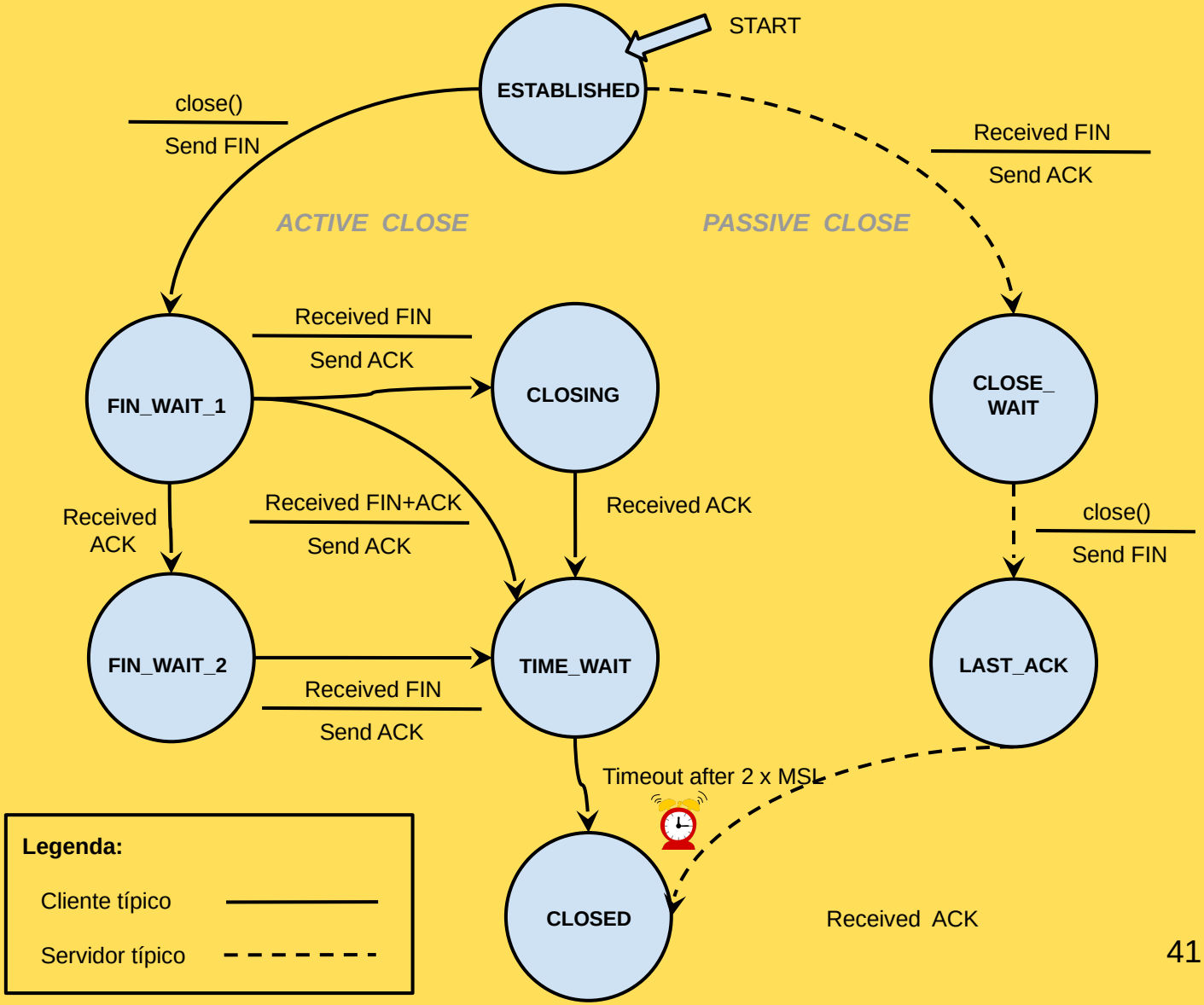
Fecho da Conexão

- O processo numa extremidade já não tem mais nada para escrever e fecha a conexão *(close()*)
  - O TCP assegura que os dados já escritos chegam ao outro lado e depois desencadeia o envio do segmento FIN
- O outro lado está a ler dados
  - Quando o TCP não tiver mais dados para fornecer
  - O leitor recebe *end-of-file (read() returns -1*)
- Por outro lado, cada lado da conexão pode fechar de forma independente a conexão
  - Invocando *shutdown()* ao invés de *close()*
- Por isso se diz que o TCP implementa um "*graceful tear-down*" das conexões

Fecho da conexão



Máquina de Estados

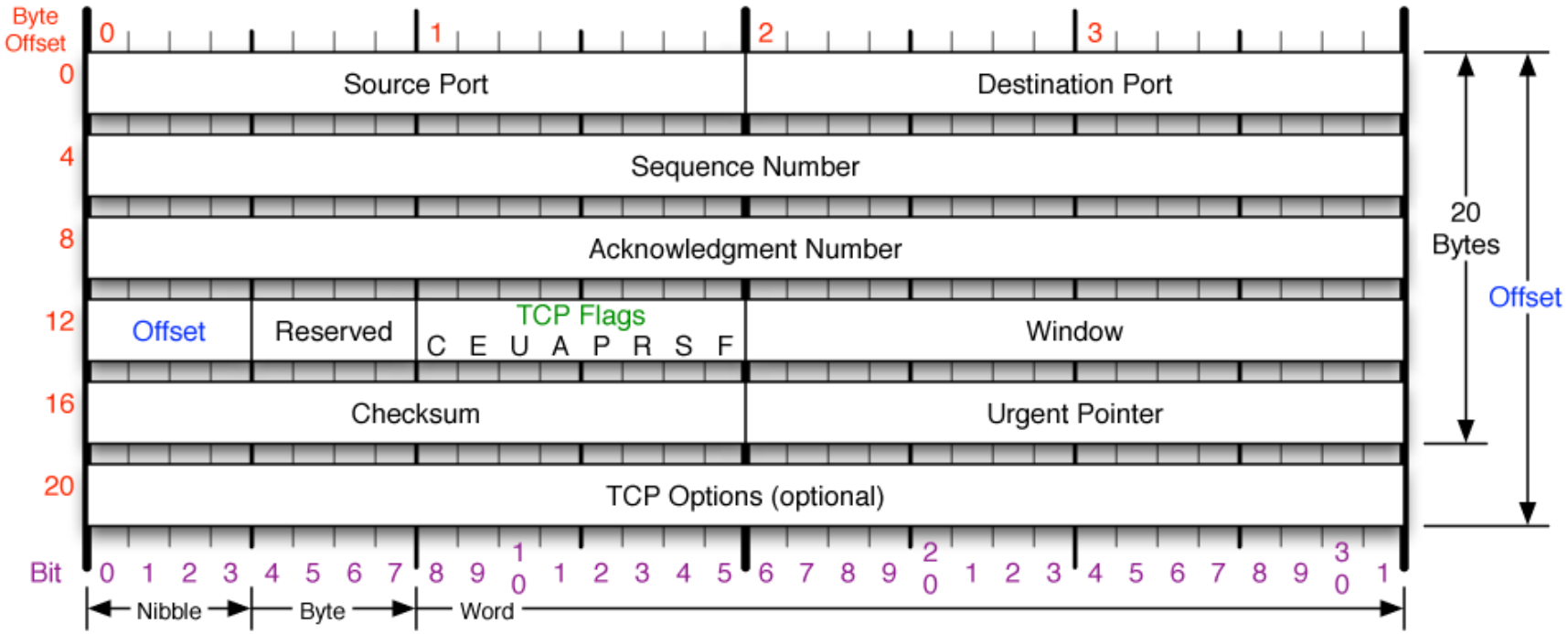


Débito Extremo a Extremo Teórico

Débito = Dimensão Média da Janela / RTT

$$V = W_{avr} / RTT$$

TCP Header



TCP Flags

C E U A P R S F

Congestion Window
C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Reciever Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

TCP = Turing Award

- O TCP foi desenvolvido a partir da experiência anterior e publicado no ano de 1974 pela primeira vez
- Vinton Cerf and Robert Khan, “A Protocol for Network Interconnection,” IEEE Transactions on Networking, Vol. 22 No. 5, May 1974, pp 637-648
- Os seus autores iniciais receberam um ACM Turing Award em 2004 relacionado com as contribuições que produzidas (IP + TCP = TCP/IP)

Conclusões

- A Internet tem características especiais
  - Pode perder pacotes
  - Pode fazê-los chegar fora de ordem
  - Pode fazê-los chegar muito atrasados face aos outros
  - Apresenta tempo de trânsito extremo a extremo muito variáveis

- Um protocolo de transferência fiável de dados como o TCP tem de lidar com todas essas características
  - O TCP foi sendo desenvolvido a partir da experiência adquirida, inicialmente durante a década de 1970
  - Tem sido melhorado incrementalmente desde então de forma contínua, mesmo ainda recentemente
  - É o principal protocolo de transporte usado na Internet e é muito sofisticado
  - O êxito da Internet está-lhe profundamente ligado