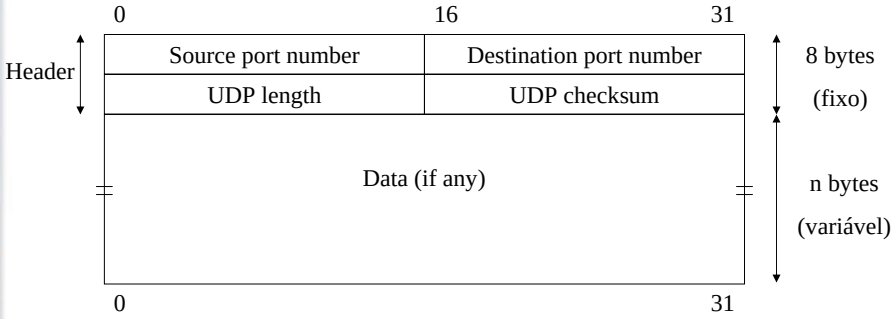
UDP - User Datagram Protocol

Serviço disponibilizado

- Fornece um serviço não orientado à ligação e não fiável
- Protocolo muito simples
- Cada pedido de envio gera um único datagrama UDP colocado num pacote IP (ou mais se houver fragmentação)
- Acrescenta a identificação do processo de aplicação remetente e do processo de aplicação destinatário (o IP identifica o sistema remetente e o sistema destinatário)

UDP - User Datagram Protocol

- Formato do datagrama



- Pseudo header

Source IP Address		
Destination IP Address		
0	Protocol (17)	UDP length

UDP - User Datagram Protocol

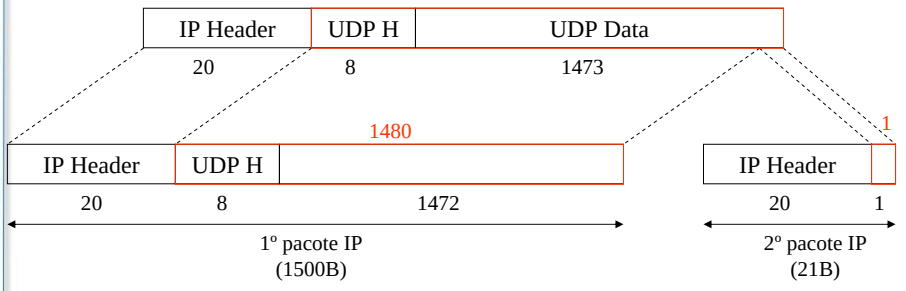
- Formato do datagrama (cont.)
  - Source port (16b): número do porto que identifica o processo de aplicação remetente
  - Destination port (16b): número do porto que identifica o processo de aplicação destinatário
  - UDP length (16b): Nº de bytes de todo o datagrama
  - UDP checksum (16b): (opcional) verifica integridade de todo o datagrama - cabeçalho mais dados – e ainda o pseudo cabeçalho
  - Data: dados de aplicação a transportar

UDP - User Datagram Protocol

- Números de porto
  - Os portos são usados pelo UDP e pelo TCP para identificarem os processos de aplicação que estão a servir
  - Uma vez que ao nível do IP se faz a distinção entre os dois protocolos, pode-se usar o mesmo número de porto para dois processos distintos, um que utilize o UDP e outro que utilize o TCP
  - Números de porto “bem conhecidos” (*well-known*):
    - Entre 1 e 1023 são usados por servidores para fornecerem serviços bem conhecidos, geridos pela IANA: 25-SMTP, 80-HTTP, ...
  - Números de porto “clientes” (depende da implementação):
    - Entre 1024 e 5000 são usados por processos clientes
  - Restantes (depende da implementação):
    - Entre 5001 e 65535 são usados por servidores para fornecerem serviços pouco conhecidos

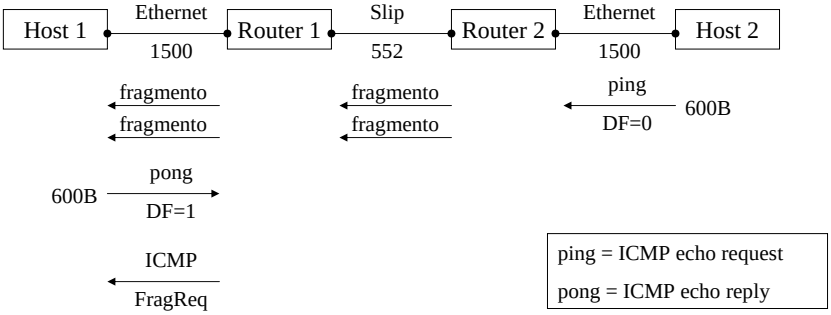
UDP - User Datagram Protocol

- Fragmentação IP
  - Antes do IP enviar o pacote questiona a interface para obter o MTU (Maximum Trasmission Unit)
  - Se o datagrama UDP, com o cabeçalho IP, for maior que o MTU é necessário fragmentar em vários pacotes IP
  - Exemplo (ethernet: MTU = 1500B):



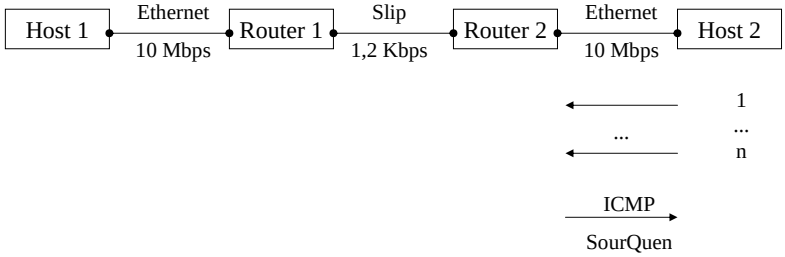
UDP - User Datagram Protocol

- ICMP unreachable error (fragmentation required)
  - Se a flag *don't fragment* (DF) estiver activa no pacote IP e seja necessário fragmentar é gerada uma mensagem ICMP para o remetente a indicar que o destino é inacessível (tipo=3) porque é necessário fragmentar mas a flag não deixa (código=4)



UDP - User Datagram Protocol

- ICMP source quench error (tipo=4, código=0)
  - Se um sistema estiver a receber datagramas a uma velocidade superior à sua capacidade de processamento é gerada uma mensagem ICMP para o remetente a avisar que deve diminuir a velocidade de envio



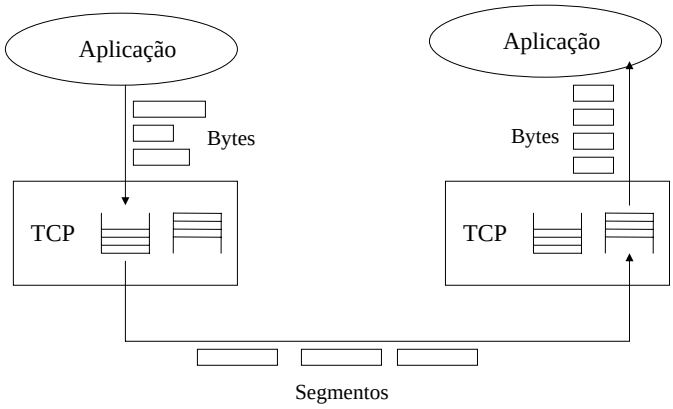
TCP – Transmission Control Protocolc

Serviço disponibilizado

- Serviço orientado à ligação, fiável e *byte stream*
  - Orientado à ligação: necessidade de estabelecer uma ligação antes do envio de dados e posteriormente quebrar a ligação (ex: telefone)
  - Fiável: garante que os dados chegam ao destino e na ordem pela qual saíram. Necessita de mecanismo de confirmação de recepção e de numeração dos dados
  - *Byte stream*: o fluxo de bytes escrito para a ligação é exactamente o fluxo de bytes lido na outra extremidade. A operação de leitura é independente da operação de escrita (uma leitura pode obter os dados de três escritas consecutivas)

TCP – Transmission Control Protocol

Serviço disponibilizado (*byte stream*)

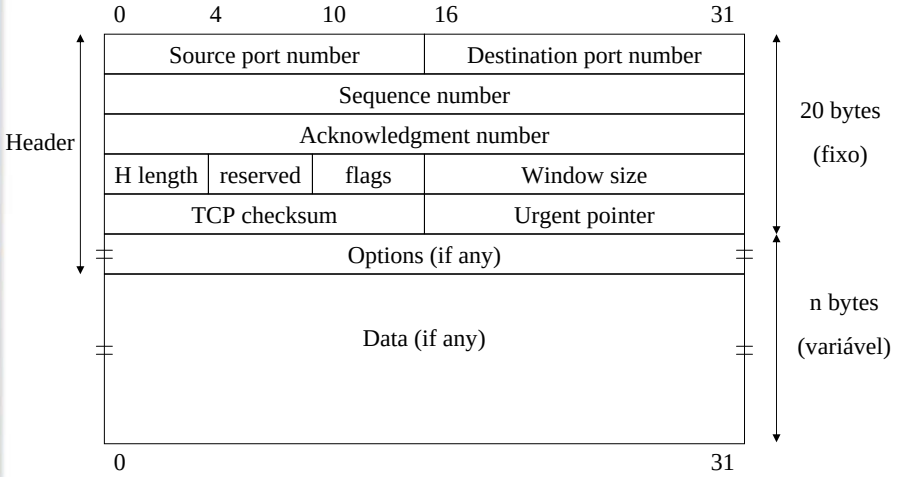


TCP – Transmission Control Protocol

- Características da ligação TCP
  - identificada univocamente por 4 parâmetros:
    - endereço IP do remetente;
    - número de porto do remetente;
    - endereço IP do destinatário;
    - número de porto do destinatário.
  - Funciona em modo *full-duplex*
  - Permite ao TCP fornecer um circuito virtual directo entre duas aplicações
  - As aplicações não precisam de se preocupar com possíveis perdas de dados que possam ocorrer na rede

TCP – Transmission Control Protocol

- Formato do segmento



TCP – Transmission Control Protocol

- Formato do segmento (cont.)
  - Source port (16b) e Destination port (16b): número do porto que identificam univocamente a ligação TCP
  - SN: Sequence number (32b): identifica o byte da stream de dados que está a ser enviada
  - AN: Acknowledgement number (32b): contém o próximo número de sequência que o remetente espera receber. Este número só é válido se a flag ACK estiver activa
  - Header length (4b): comprimento do cabeçalho em palavras de 32 bits → Max. H len = 60 bytes
  - Reserved (6b): campo reservado (colocado a 0)
  - Flags (6b):
    - URG – urgent pointer é válido
    - ACK – acknowledgement number é válido
    - PSH – destinatário deve passar os dados à aplicação imediatamente
    - RST – efectuar reset da ligação
    - SYN – sincronizar o sequence number para iniciar a ligação (estabelecimento)
    - FIN – o remetente não tem mais dados a enviar (quebra da ligação)

TCP – Transmission Control Protocol

- Formato do segmento (cont.)
  - Window size (16b): número de bytes que o remetente está disponível a receber no próximo segmento (controlo de fluxo)
  - TCP checksum (16b): verifica integridade de todo o segmento - cabeçalho mais dados – e ainda o pseudo cabeçalho
  - Urgent pointer (16b): offset que indica a posição do último byte de dados urgentes enviados no início dos dados
  - Option: opções do TCP (ex: Maximum Segment Size: indica o tamanho máximo do segmento que o remetente pode receber – normalmente enviado no segmento de estabelecimento da ligação)
  - Data: dados de aplicação a transportar

TCP – Transmission Control Protocol

- Modo de funcionamento
  - Estabelecimento de ligação
    - Decomposto em três fases: pedido, aceitação e confirmação
  - Troca de dados
    - Divide os dados da aplicação em segmentos
    - Active um despertador cada vez que envia um segmento
    - Confirma a recepção de dados
    - Ordena os dados recebidos fora de ordem
    - Deita fora dados duplicados
    - Fornece controlo de fluxo extremo-a-extremo
    - Fornece controlo de congestionamento
    - Obrigatória a detecção de erros de transmissão
  - Quebra de ligação
    - Normalmente em quatro fases: indicação e confirmação, indicação e confirmação

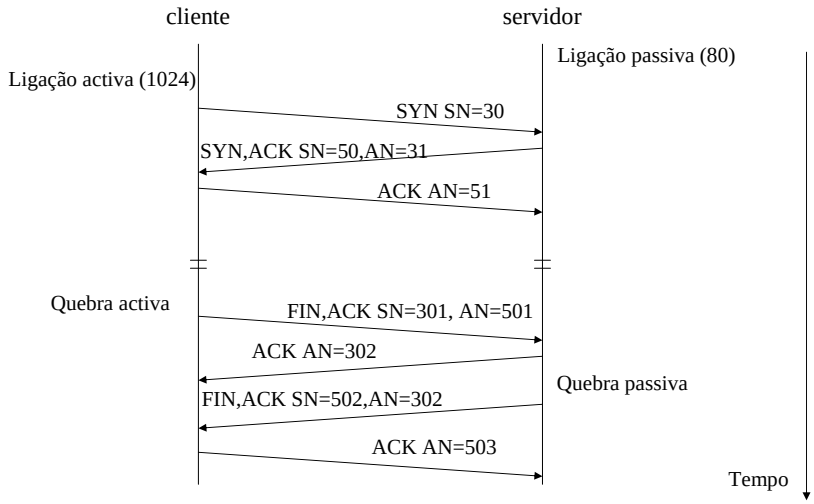
TCP – Transmission Control Protocol

- Protocolo de estabelecimento de ligação
  - Nesta fase é usado o modelo cliente-servidor (um dos sistemas funciona como cliente e o outro como servidor)
  - Protocolo
    1. O cliente envia um segmento SYN (todas as outras flags estão inactivas) com o número de sequência inicial do cliente (ISNcl)
    2. O servidor responde com o seu segmento SYN com o número de sequência inicial do servidor (ISNsrv) e ao mesmo tempo confirma a recepção do segmento SYN cliente activando a flag ACK e colocando no campo número de confirmação (AN) o valor de ISNcl+1
    3. O cliente envia confirmação num segmento ACK com AN=ISNs+1.
  - Este processo é designado por aperto de mão em três fases (*three-way handshake*)

TCP – Transmission Control Protocol

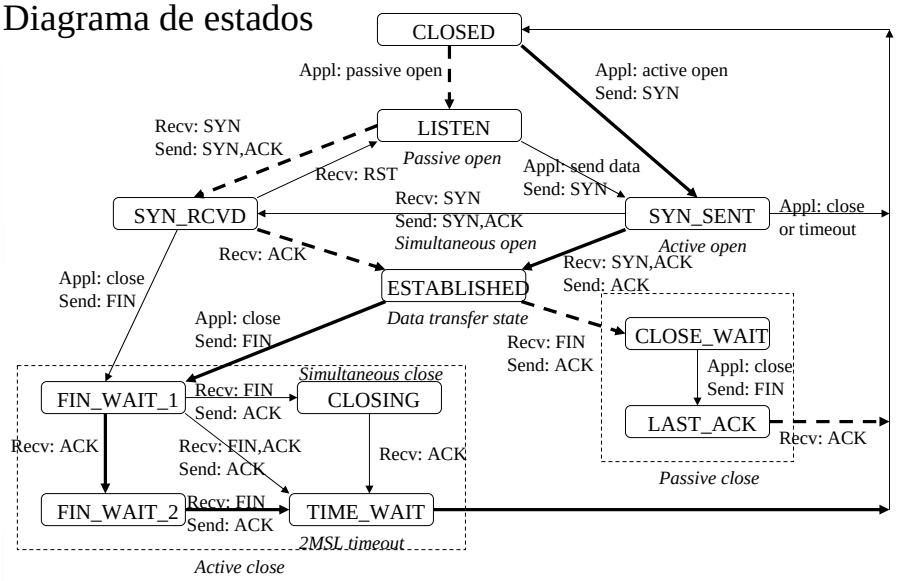
- Protocolo de quebra de ligação
  - Como a ligação é *full-duplex*, cada uma das partes necessita de fazer um *half-close*
  - Quando um sistema já não tem dados para enviar faz o *half-close*. Qualquer um dos sistemas pode começar esta fase
  - Protocolo
    1. O cliente envia um segmento FIN com o seu número de sequência (SNcl) actual e com o respectivo número de confirmação
    2. O servidor envia um segmento ACK com AN=SNcl+1
    3. O servidor envia um segmento FIN com o seu número de sequência (SNsrv) actual e com o respectivo número de confirmação
    4. O cliente envia um segmento ACK com AN=SNsrv+1
    - Entre os pontos 2 e 3 o servidor pode enviar dados, no entanto, não é muito comum

TCP – Transmission Control Protocol



TCP – Transmission Control Protocol

Diagrama de estados

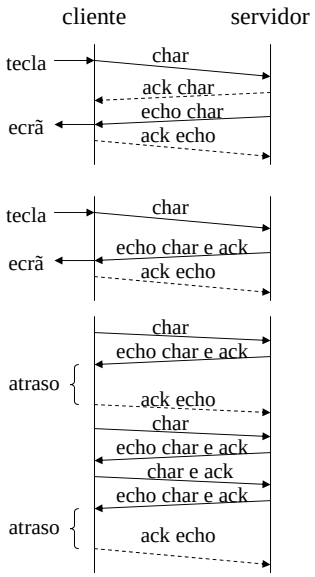


TCP – Transmission Control Protocol

- Tamanho máximo dos segmentos (MSS)
  - Número máximo de bytes que podem ser enviados num segmento
  - Anunciados por cada parte nos segmentos SYN (536 B se omitido)
  - Normalmente, quanto maior melhor (até ocorrer fragmentação)
- Segmentos *Reset*
  - Enviado sempre que chega um segmento numa altura indevida:
    - Pedido de estabelecimento para uma porta inexistente
    - Abortar uma ligação (quebra imediata da ligação sem envio dos restantes dados nem espera confirmação da outra parte)
- Estabelecimento simultânea da ligação
  - Ambas os sistemas efectuam uma ligação activa (pouco provável)
  - Resulta numa única ligação
- Quebra simultânea de ligação
  - Ambos os sistemas efectuam uma quebra activa (nº segmentos igual)
- TCP *options*
  - Existem outras opções mas a mais comum é a MSS

TCP – Transmission Control Protocol

- Fluxo de dados interactivo
  - Existem aplicações (Rlogin, Telnet, ...) que enviam um segmento por cada byte (caracter) escrito
  - No entanto o 2º e 3º segmento costumam ser combinados
  - Confirmações atrasadas:
    - Envio de ack não é imediato
    - Despertador (tipica/ de 200ms)
    - Objectivo: esperar por dados a enviar
    - ACK *piggyback*



TCP – Fluxo de dados interactivo

- Algoritmo de Nagle
  - Evitar pacotes demasiado pequenos (*tinygrams*); ex:41 bytes
  - TCP não pode enviar *tinygrams* enquanto não receber confirmação dos segmentos na rede
  - Os dados pequenos vão sendo acumulados e enviados num só segmento quando a confirmação chegar
  - Quanto mais depressa chegarem os ACK mais depressa os dados são enviados
  - A utilização de aplicações do tipo Rlogin e Telnet em LANs é difícil ver este algoritmo a funcionar pois seria necessário escrever no mínimo 60 caracteres por segundo
  - Em ligações WANs lentas já se consegue ver o algoritmo a funcionar, melhorando o desempenho da rede

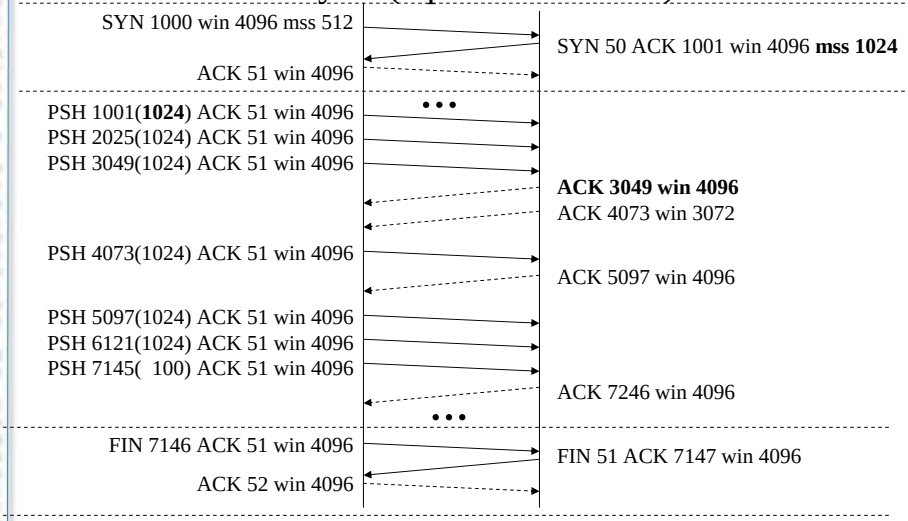
TCP – Fluxo de dados volumoso

- Fluxo de dados volumoso (*bulk*)
  - Comum em muitas aplicações (FTP, HTTP, ...) onde os dados estão disponíveis (em ficheiros) para serem enviados
  - Consiste no envio de vários segmentos antes de receber confirmação de recepção
  - As próprias confirmações podem anunciar, de uma só vez, a chegada de vários segmentos
  - Deste modo, a transferência dos dados é mais rápida e o débito diminui (carga de processamento dos *tinygrams*)
  - O comportamento do TCP depende de muitos factores:
    - Modo de implementação do remetente;
    - Modo de implementação do destinatário;
    - Escalonamento do sistema operativo;
    - Comportamento da rede.

    Não existindo um único modo correcto para transferir os dados

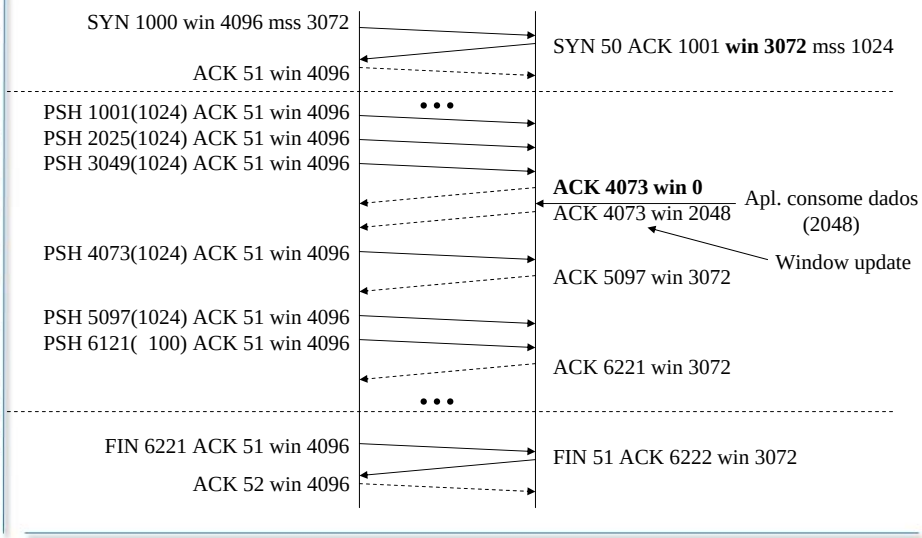
TCP – Fluxo de dados volumoso

- Envio de 6 244 Bytes (Apl 6244 → TCP):



TCP – Fluxo de dados volumoso

- Remetente rápido, destinatário lento

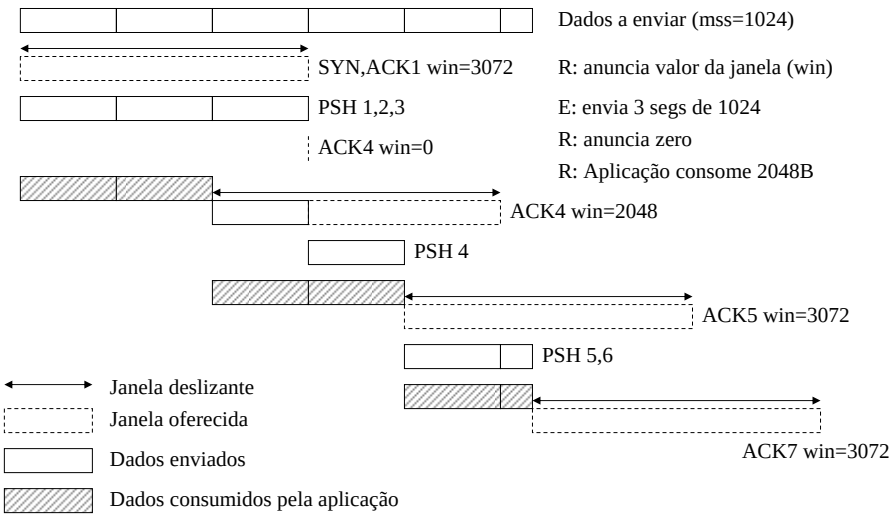


TCP – Fluxo de dados volumoso

- Janelas deslizantes
  - O TCP reserva memória de recepção (*buffer*) para a aplicação
  - O tamanho do *buffer* determina o valor máximo da janela (*win*)
  - Ambos as partes anunciam o tamanho da janela
  - Um sistema não pode enviar mais segmentos se o tamanho dos dados enviados atingir o valor anunciado pelo outro
  - Conforme chegam dados, o TCP coloca-os no *buffer* na posição correspondente (utiliza o *offset* do SN)
  - Antes de ser enviado o ACK, é verificado se a aplicação consumiu dados:
    - Não, o valor da janela no ACK é diminuído (pode chegar a ser 0)
    - Sim, a janela desliza para um novo *offset*
  - Assim, se o sistema que recebe estiver sobrecarregado, a aplicação demora mais a consumir os dados, o TCP avisa a outra parte para diminuir o débito

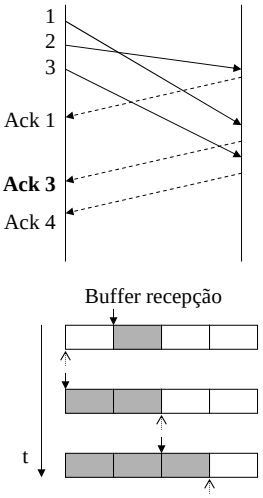
TCP – Fluxo de dados volumoso

- Janelas deslizantes (ver segmento de dados da pág. 25 - receptor)



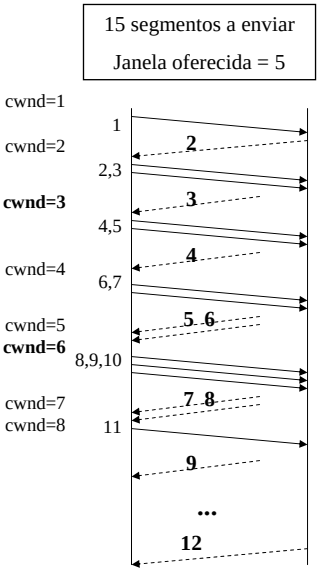
TCP – Fluxo de dados volumoso

- Segmentos fora de ordem
  - Uma vez que os segmentos TCP são transportados por pacotes IP, podem chegar na ordem errada
  - Como o *buffer* está preparado para receber todos os segmentos enviados, basta colocar os dados na posição respectiva
  - As confirmações de recepção continuam a apontar para o primeiro *offset* dos dados
  - Ao contrário da recepção de segmentos na ordem correcta, quando chega um fora de ordem é necessário enviar logo uma confirmação



TCP – Fluxo de dados volumoso

- Começo lento
  - Evitar sobrecarregar a rede quando está congestionada
  - Necessário janela de congestionamento (cwnd)
  - Controlo de fluxo imposto pelo remetente
  - Inicialmente cwnd=1,envia 1 segmento e espera por ACK (na realidade cwnd é expresso em bytes)
  - Incrementa a janela cwnd sempre que chega um ACK
  - O nº de segmentos que podem estar na rede (ainda não confirmados) é o mínimo entre o valor de cwnd e o valor da janela oferecida
  - Quando houver perda de pacotes volta ao começo lento (posteriormente o aumento do cwnd será controlado por um algoritmo para evitar o congestionamento)
  - O débito de envio de segmento é determinado pelo débito de recepção de confirmações

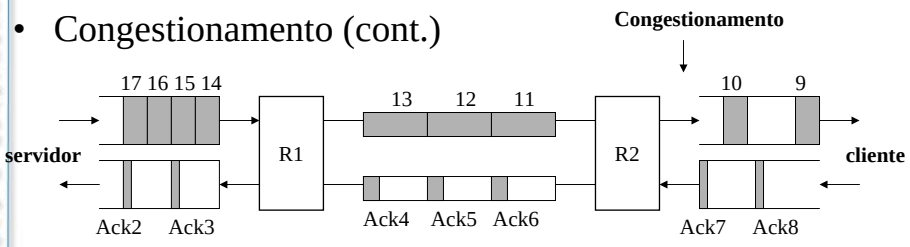


TCP – Fluxo de dados volumoso

- Congestionamento
  - Normalmente ocorre na intercepção de uma LAN com uma WAN
  - A transmissão dos dados é rápida mas a chegada das confirmações é lenta
  - O *buffer* de recepção do router enche e os pacotes são deitados fora
  - Uma vez que a janela oferecida costuma ser relativamente grande, o começo lento ocorreria periodicamente
  - Situação com pouco dinamismo para o TCP
  - Solução: sempre que ocorre um erro é definido um novo patamar (ssthresh) para o valor do cwnd → ssthresh = 1,5 * min(cwnd; janela oferecida)
  - Quando o patamar é atingido, entra em funcionamento o algoritmo de controlo de congestionamento:
    - cwnd < ssthresh; cwnd=cwnd+1 por ACK ← começo lento
    - cwnd >= ssthresh; cwnd=cwnd+1/cwnd por ACK ← congestionamento
  - Tipo de erros:
    - *timeout* ← perda de pacote ou despertador com pouco tempo (requer começo lento)
    - ACK duplicado ← recepção fora de ordem ou *buffer* do destinatário cheio

TCP – Fluxo de dados volumoso

- Congestionamento (cont.)



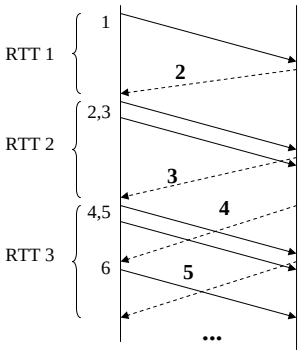
- Dados urgentes
  - Envio de dados especiais ao mesmo tempo que dados normais
  - Implementação depende da aplicação
  - Exemplo: envio de tecla de interrupção (Telnet, FTP)
  - Envio de dados urgentes:
    - Activar flag URG
    - Colocar offset do último byte de dados urgentes
    - Normalmente, colocar os dados urgentes no início dos dados

TCP – Timeout e retransmissão

- *Timeout* e retransmissão
  - Transmissão fiável → recepção de confirmação
  - Perda da pacote → retransmissão, controlada por:
    - Destinatário → *Negative* ACK (NACK) ← difícil (pacotes desordenados)
    - Remetente → *timeout* ← solução
  - Valor do despertador:
    - Pequeno → mais rápido ← retransmissão antes de chegar confirmação
    - Grande → mais lento ← demasiado numa LAN
    - Variável consoante o estado da rede ← solução
  - Tempo entre envio e recepção do ACK (*round-trip time - RTT*):
    - Fundamental para calcular o valor do despertador de retransmissão (RTO)
    - RTT calculado por segmento → RTO actualizado proporcionalmente:
      - Chega ACK → RTO diminuí
      - Ocorre *timeout* → RTO aumenta
    - Segmento re-transmitido → RTO não actualizado
    - Implica mais um cálculo por segmento enviado

TCP – Timeout e retransmissão

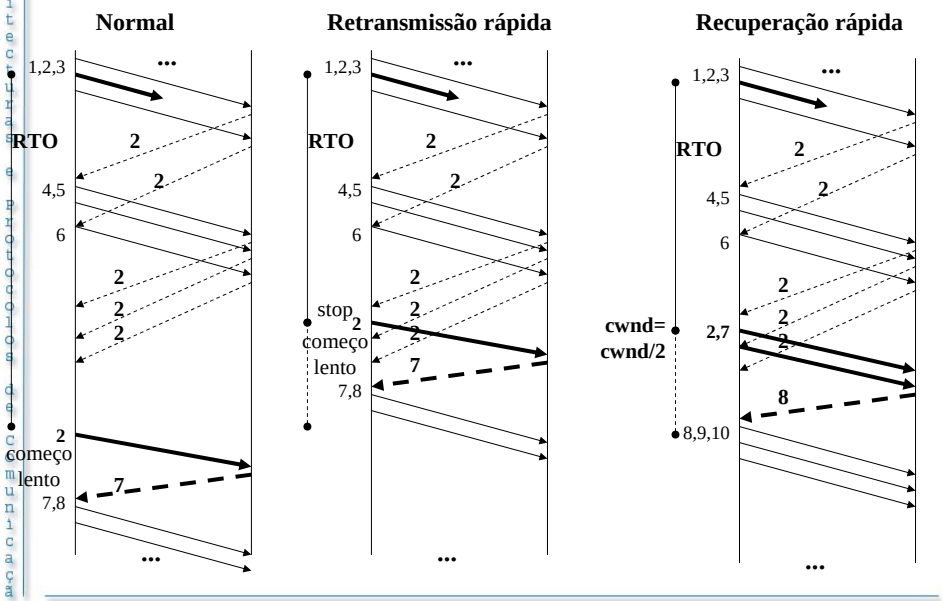
- Implementações típicas
  - Os tempos dos despertadores são medidos em intervalos de 500ms
  - Só existe um cálculo do RTT num dado instante (só é calculado o RTT do 1º segmento de uma série)
  - O valor do RTO inicial é de 6s
  - O cálculo de um novo RTT leva a um desvio proporcional do valor do RTO corrente
  - Em caso de retransmissão:
    - Na 1ª vez o RTO não é actualizado
    - Para as seguintes o RTO aumenta exponencialmente (algoritmo *backoff*)



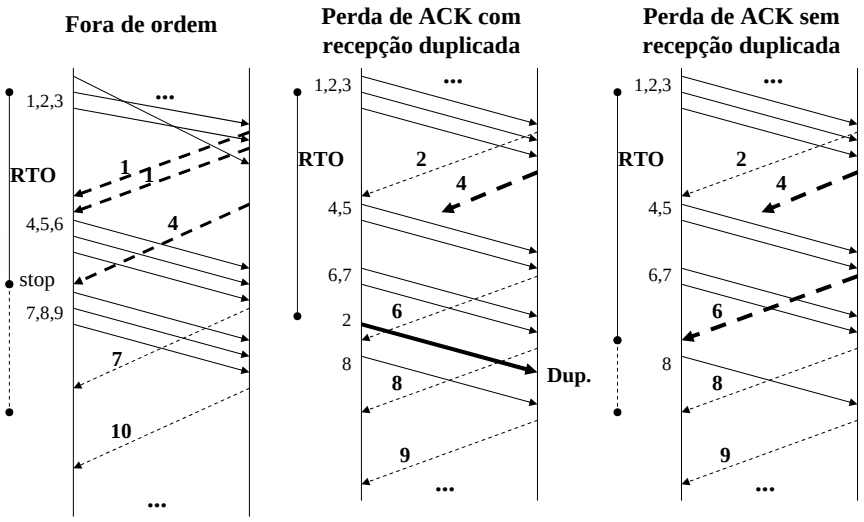
TCP – Timeout e retransmissão

- Retransmissão rápida e recuperação rápida
  - Relembrar que o TCP envia imediatamente um ACK sempre que chega um segmento fora de ordem
  - A chegada de ACKs iguais pode ser por 2 motivos:
    - Segmento fora de ordem (1 ou 2 ACKs iguais seguidos)
    - Pacote perdido (série de ACKs iguais seguidos)
  - A retransmissão de segmento antes do despertador tocar quando é detectada uma série de ACKs iguais chama-se **retransmissão rápida**
  - A seguir, em vez de usar o começo lento, o cwnd é actualizado por um algoritmo de **recuperação rápida**

TCP – Timeout e retransmissão



TCP – Timeout e retransmissão



RTO demasiado baixo
TCP deita duplicado fora

TCP – Outros despertadores

- Despertador persistente (*Persist Timer*)
  - Se *buffer* enche é enviado ACK com uma janela de 0
  - Remetente não envia enquanto não receber ACK com janela diferente de zero
  - Se este último ACK se perder ambos ficam à espera (*deadlock*)
  - Remetente coloca despertador para envio de segmento de sonda da janela (*window probe*). Segmento com 1 byte que deve ser aceite pelo destinatário mas que pode deitar fora se a sua janela continuar cheia
- Despertador acordado (*Keepalive Timer*)
  - O TCP não quebra a ligação em caso de inactividade
  - Pode-se efectuar uma ligação, ir uma semana de férias, os router podem ter ido abaixo, mas a ligação continua activa
  - O destinatário não sabe se o remetente foi de férias ou fez *reboot*
  - Existem servidores que implementam despertadores para quebrarem ligações inactivas
  - A solução é controversa pois o cliente pode ter ficado temporariamente sem conectividade

TCP – Outros despertadores

- Aplicação não consome dados (mss=1024)