Universidade Federal de Ouro Preto

Transmissão Confiável

Link desta videoaula - youtu.be/G9tEQmImPYs

Referências:

- Redes de Computadores**. L. L. Peterson e B. S. Davie. 5ª edição, LTE, 2013. Seção: 2.5.**
- Data and Computer Communications**, 9th edition. W. Stallings. Prentice Hall, 2010. Appendix 7A.**

Transmissão Confiável

•Durante a transmissão de dados em uma rede de datagramas, diversos problemas podem ocorrer:

- Datagramas podem chegar atrasados;
- Datagramas podem chegar fora de ordem;
- Pacotes às vezes são modificados enquanto estão em trânsito;
- Pacotes podem não chegar ao destino.

• Devido à estes problemas e aos requisitos de confiabilidade por parte da aplicação, o TCP foi desenvolvido.

- Orientado à conexão
- Confiabilidade
- Full duplex
- Entrega ordenada
- Controle de fluxo
- Controle de congestionamento

Transmissão Confiável

•**Controle de Erro**: técnicas usadas para detectar e corrigir erros ocorridos durante a transmissão;

- Muitas vezes a correção de erros não é viável:
  - Na prática a correção de erros causa uma sobrecarga muito grande para tratar todos os erros de bits e rajadas de um enlcace.
  - Há alguns erros sérios demais para serem corrigidos.

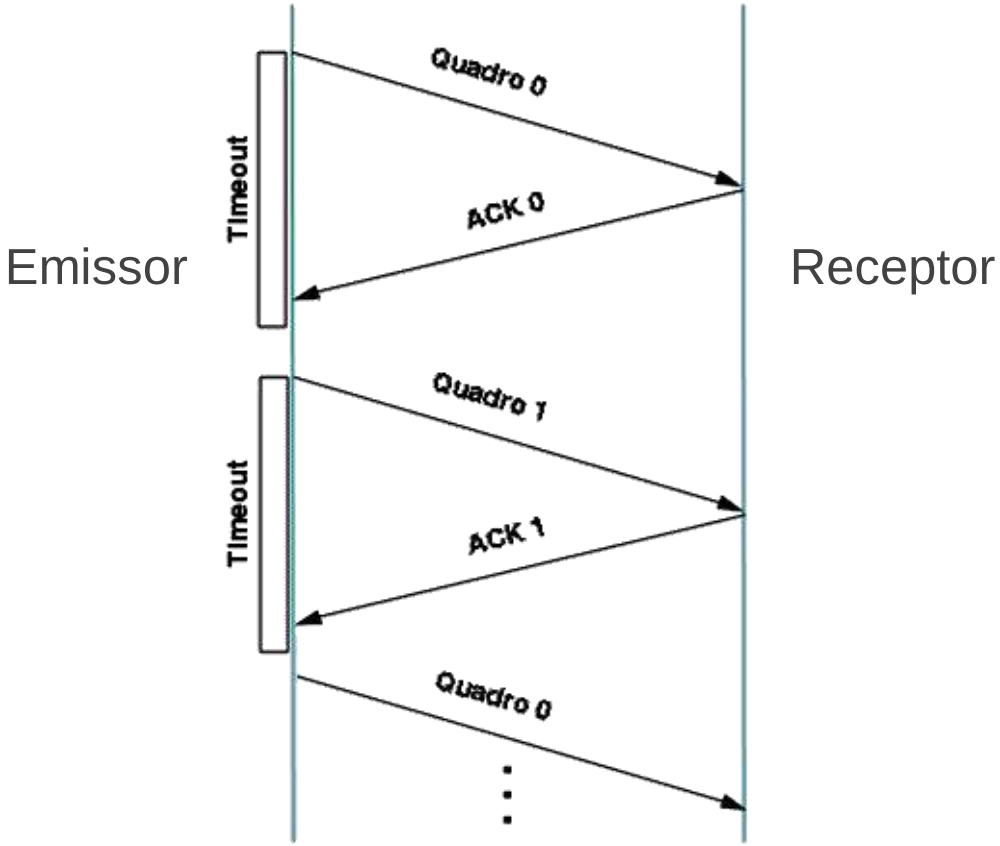
- Assim, alguns quadros com erros precisam ser descartados.
- Um protocolo no nível de transporte que seja confiável precisa recuperar-se de alguma forma desses quadros descartados / perdidos.
  - Isso é feito pelo TCP por uma combinação de dois mecanismos:
    - Confirmações (ACK) e timeouts.

Transmissão Confiável

- Estratégia que implementa a entrega confiável: ARQ - Automatic Repeat Request (Solicitação automática de repetição)
  - técnica mais utilizada;
  - exige detecção de erro (CRC, p. ex.);
  - retransmite o quadro que deu problema;
  - combina confirmação (ACK - acknowledgment) e temporização (timeout).
- Existem diferentes algoritmos ARQ:
  - Stop and wait
  - Janela deslizante:
    - Go Back N
    - Selective Repeat

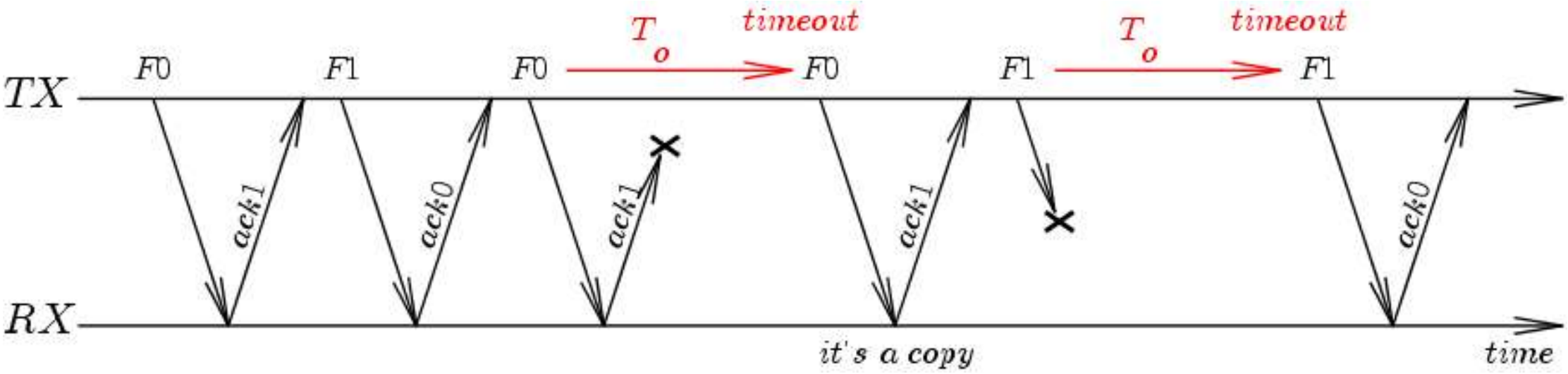
ARQ “Parar e Esperar” (Stop and Wait)

- Esquema de ARQ mais simples:
  - Depois de transmitir um quadro, o emissor espera uma confirmação antes de transmitir o próximo quadro.
  - Se a confirmação não chegar após um período, o emissor reinicia seu tempo limite e retransmite o quadro original.



ARQ “Parar e Esperar” (Stop and Wait)

- Problemas do ARQ “Parar e Esperar”:
  - Um pacote pode ser perdido;
  - Um ACK pode ser perdido ou chegar atrasado;
  - Pode ocorrer a situação em que um timeout é disparado antes do ACK chegar ao emissor;



- Problema: manter a tubulação cheia.
  - Esse ARQ é limitado, pois permite que o emissor tenha apenas 1 quadro pendente no enlace de cada vez... e isso pode ser muito abaixo da capacidade do enlace

ARQ “Parar e Esperar” (Stop and Wait)

- Considere o seguinte exemplo:
  - Um enlace com BW = 1,5 Mbps e RTT = 45 ms
  - Esse enlace possui produto retardo x BW = 67,5 kb (ou 8 KB)
  - T_{TR} = 8192 bits/1,5 Mbps = **5,46 ms**
  - T_{TOTAL} = (5,46 + 45) ms = **50,46 ms**
  - Como o emissor só pode enviar um quadro por RTT e considerando tamanho de quadro de 1 KB, temos:

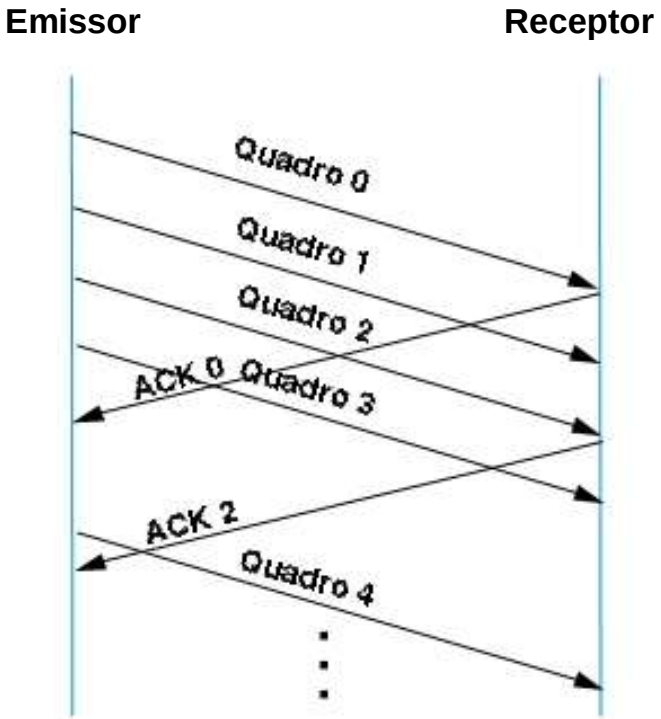
    VAZÃO = Bits enviados (por quadro) **/ Tempo** (por quadro)

    Vazão = 8192 bits/ 50,46 ms

    VAZÃO = 162 kbps **(apenas 11% de BW !!!)**

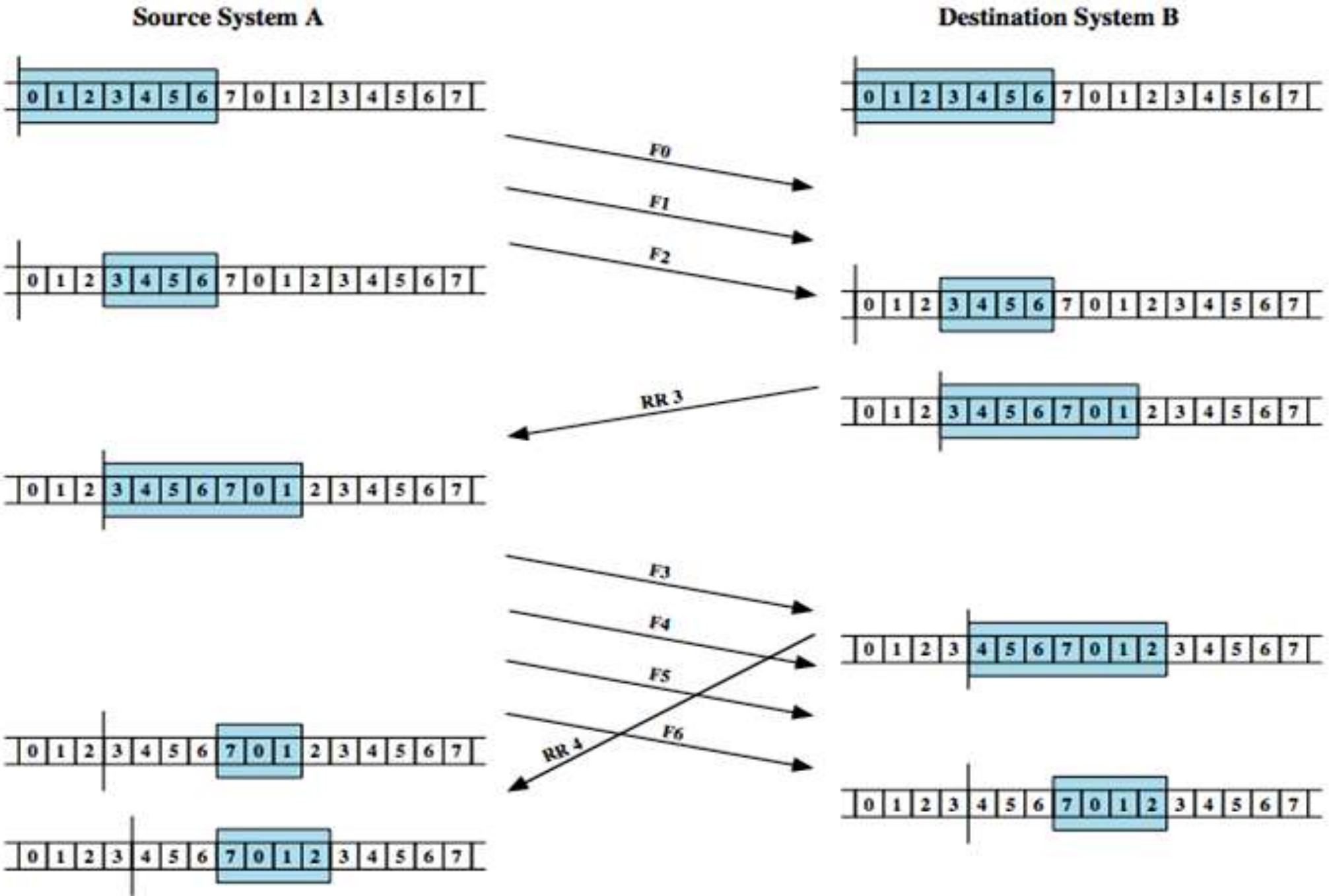
  - Logo, para usar o enlace totalmente, seria necessário que o transmissor pudesse enviar até nove quadros antes de ter que esperar uma confirmação.

ARQ “Janela Deslizante” (Sliding Window)



- Stop-and-Wait só permitiria 100% de utilização se RTT fosse zero !
- Solução: manter a tubulação cheia enviando mais quadros antes de receber todos os ACKs
- Receptor tem buffer para armazenar até W quadros.
- Emissor pode enviar até W quadros sem ACK (W = janela).
- Cada quadro recebe um número de seqüência.
- Este número tem k bits (cobre de 0 a $2^k - 1$).
- O ACK pode ser individual ou cumulativo.

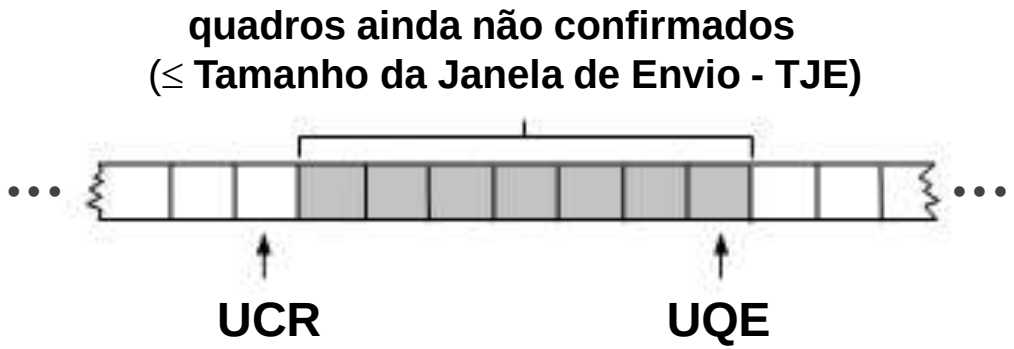
Sliding Window Example



ARQ “Janela Deslizante” (emissor)

- Funcionamento do algoritmo de janela deslizante:
  - Emissor atribui um nº de sequência a cada quadro (NumSeq);
  - Emissor mantém 3 variáveis:
    - Tamanho da janela de envio (TJE) – nº de quadros não confirmados que o emissor pode transmitir;
    - Última confirmação recebida (UCR)
    - Último quadro enviado (UQE)
  - Emissor deve manter o seguinte invariável:

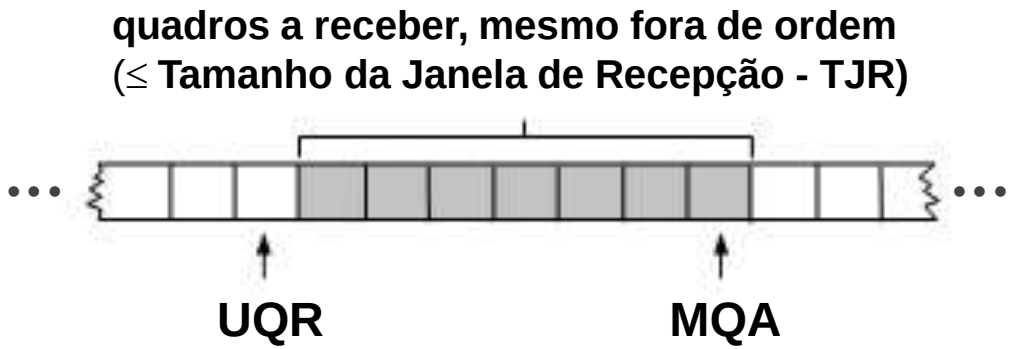
UQE – UCR ≤ TJE



ARQ “Janela Deslizante” (receptor)

- Funcionamento do algoritmo de janela deslizante:
  - Receptor mantém 3 variáveis:
    - Tamanho da janela de recepção (TJR) – nº de quadros fora de ordem que o receptor deseja aceitar;
    - Maior quadro aceitável (MQA)
    - Último quadro recebido (UQR)
  - Receptor também deve manter o seguinte invariável:

MQA – UQR ≤ TJR



ARQ “Janela Deslizante”

- Quando ocorre um timeout, a quantidade de dados em trânsito diminui, pois o emissor é incapaz de avançar sua janela, até um quadro x seja confirmado.
  - Quando ocorre perdas de pacotes, esse esquema **não consegue manter o canal cheio**. Quanto + tempo para notar a perda de pacotes, + severo o problema.

- O tam. janela de envio é selecionado de acordo com a quantidade de quadros que podemos ter pendentes no enlace.
  - TJE –> fácil de calcular (produto retardo x BW)
  - Receptor pode definir TJR que quiser:

    TJR = 1 : receptor não coloca em buffer nenhum quadro fora de ordem;

    TJR = TJE : receptor pode manter em buffer qualquer um dos quadros que emissor transmitir.

    TJR > TJE : não faz sentido. É impossível que mais de TJE cheguem fora de ordem.

ARQ “Janela Deslizante” - Eficiência

- **Melhor situação possível:**
  - emissor tem estoque infinito de quadros a enviar;
- **Emissor começa a transmitir o 1º quadro da janela.**
- **1º ACK (correspondente ao 1º quadro da janela) chega após:**

$$T_{1º\ ACK} = T_{TR\ 1º\ Quadro} + RTT$$

- **Sendo ($T_W = W \bullet T_{TR}$) o tempo de transmissão de toda a janela:**

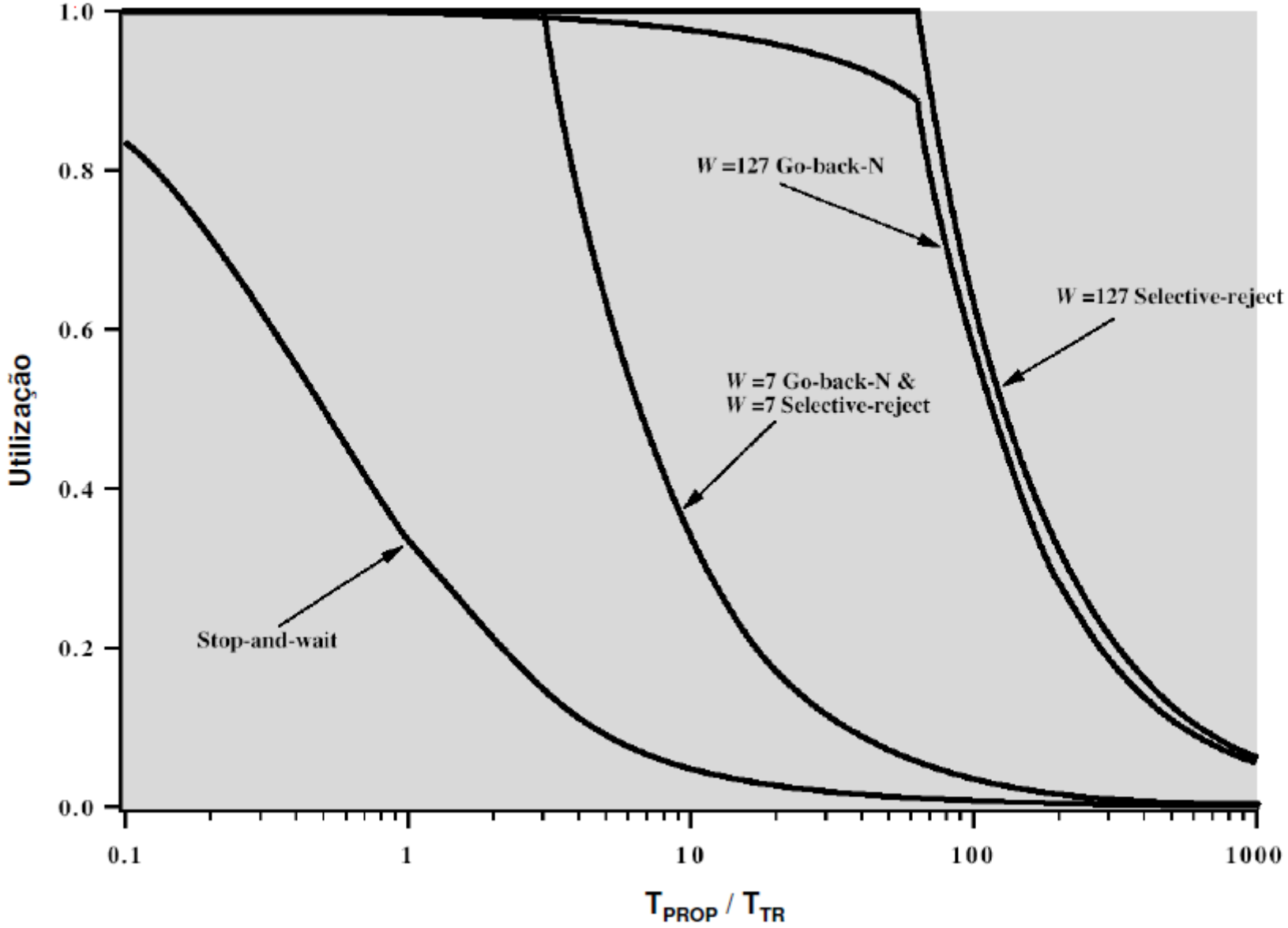
  se $T_W < T_{1º\ ACK}$ então Vazão = TotalBitsJanela / $T_{1º\ ACK}$

  se $T_W \geq T_{1º\ ACK}$ então Vazão = BW (100% de utilização !!!)

- **Exemplo:**
  - BW = 1,5 Mbps, RTT = 45 ms e quadros de 1 kB (8192 bits)
  - se W = 2 ⇒ T_W = 10,9 ms < $T_{1o\ ACK}$ = 50,5 ms ⇒ **Vazão ≈ 324 kbps**
  - se W = 5 ⇒ T_W = 27,3 ms < $T_{1o\ ACK}$ = 50,5 ms ⇒ **Vazão ≈ 812 kbps**
  - se W = 10 ⇒ T_W = 54,6 ms > $T_{1o\ ACK}$ = 50,5 ms ⇒ **Vazão = 1,5 Mbps**

Desempenho em função de T_{PROP}/T_{TR}

(com probabilidade de 1 quadro errado em cada 1000 quadros)



Ordem dos quadros e Controle de Fluxo

Janela deslizante pode ser usado para 3 papéis diferentes:

- Entregar quadros de modo confiável em um enlace não confiável
  - Essa é a função básica do algoritmo;
- Preservar a ordem em que os quadros são transmitidos
  - Fácil de se fazer no receptor, visto que cada quadro possui um nº de sequência;
  - O receptor certifica-se de que não passará um quadro para o próximo protocolo de nível superior até que tenha recebido todos quadros com nº sequência menor;
  - Ou seja, receptor mantém em buffer os quadros fora de ordem.
- Admitir o controle de fluxo

Controle de Fluxo

- Mesmo respeitando a taxa de transferência acordada, o emissor pode enviar quadros um após o outro, de uma forma que um receptor lento para processá-los acaba **perdendo vários quadros**.

- **SOLUÇÃO**: Controle de Fluxo
  - Mecanismo evita que o emissor ultrapasse a capacidade do receptor.
  - Um mecanismo de realimentação, usado pelo receptor, capaz de cadenciar o transmissor.
- Permite vários frames numerados em trânsito;
- Receptor com buffer de tamanho W:
  - Transmissor pode enviar até W frames sem ACK;
  - Número de seqüência é limitado pelo tamanho do campo (k)
  - Frames são numerados modulo 2k, dando uma janela de no máximo 2k - 1

Universidade Federal de Ouro Preto

Transporte
- Protocolo TCP -

Referência:
Redes de Computadores**. L. L. Peterson e B. S. Davie. Ed Campus, 2013.**
Seção: 5.2

Transporte TCP x Enlace de Dados

- TCP trabalha pela Internet e não por um enlace ponto a ponto;
  - *Essa pequena diferença complica um pouco o TCP.*
- TCP conecta-se a muitos hosts diferentes.
  - precisa de estabelecimento e término explícitos para cada “conexão.
- Enlace físico, 2 hosts possuem RTT fixo.
- TCP: **RTT variável !!!**
- **Tempos de RTT muito diferentes:**
  - RTT entre 2 hosts no mesmo prédio → 1 ms
  - RTT entre um host no Brasil e um servidor no Alaska (satélite) → 600 ms
- **Variações no decorrer do dia:**
  - RTT de 100 ms às 3 hs da madrugada e 500 ms às 3 da tarde

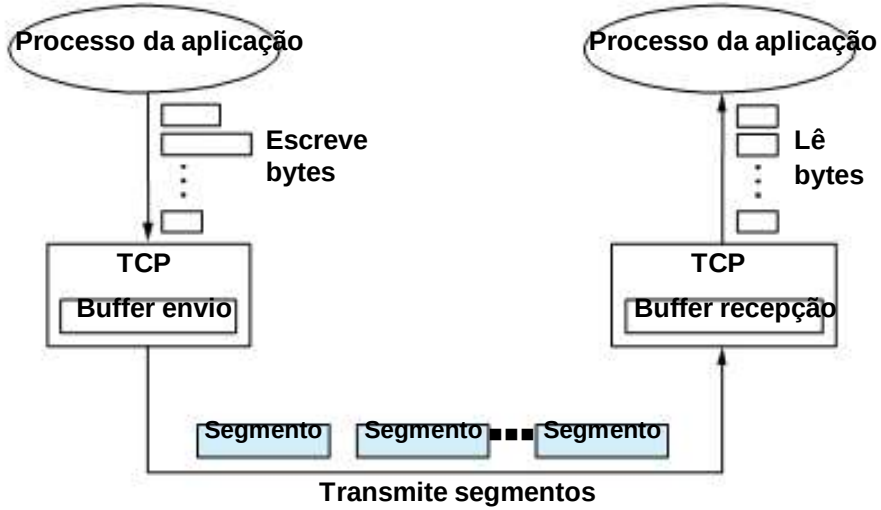
Transporte TCP x Enlace de Dados

- TCP: **RTT variável !!!**
- **Variações na mesma conexão:**
  - Pacotes podem seguir caminhos diferentes
- Mecanismo de timeout que dispara retransmissões deve ser **adaptável !**
- No TCP pacotes chegam fora de ordem, no mesmo enlace não.
- Capacidade possivelmente diferente no destino:
  - precisa acomodar capacidades diferentes dos nós.
- Capacidade de rede possivelmente variável:
  - precisa estar preparado para congestionamentos na rede.

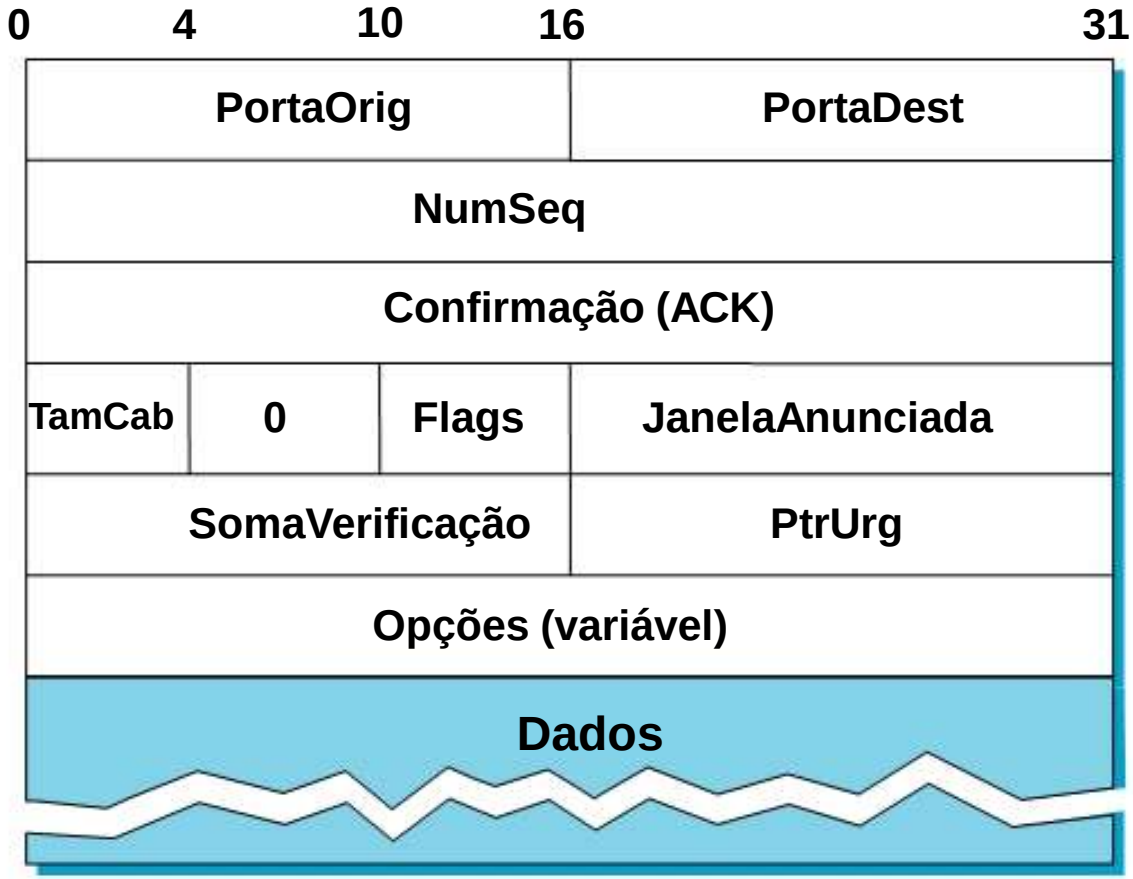
TCP - visão geral

- **Orientado a conexão.**
- **Full duplex.**
- **Fluxo de bytes:**
  - **aplicação escreve bytes;**
  - **TCP envia** segmentos;
  - **aplicação lê bytes.**

- **Controle de erro (confiável).**
- **Controle de fluxo (evita que emissor sobrecarregue o receptor).**
- **Controle de congestionamento (evita que o emissor sobrecarregue a rede).**



Formato do segmento TCP



- Flags: ACK, URG, PUSH, SYN, FIN, RESET.
- Soma de verificação: cobre "pseudo-cabeçalho + cabeçalho TCP + dados".

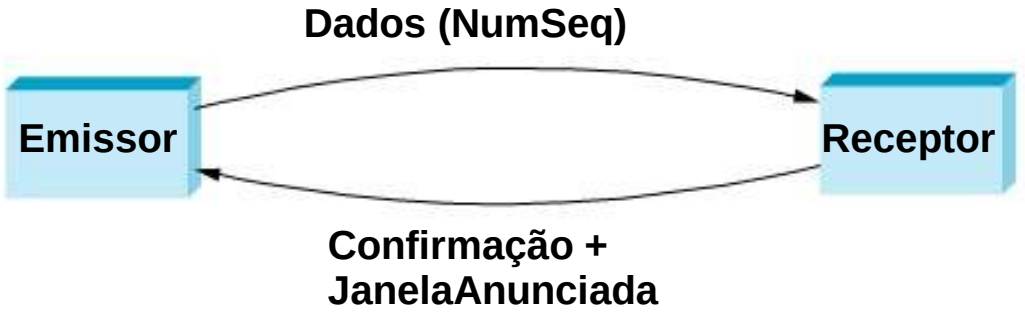
Formato do segmento TCP

- Flags:
  - ACK, URG, PUSH, SYN, FIN, RESET
- NumSeq:
  - nº de sequencia para o 1º byte de dados transportados neste segmento
  - **Qual tamanho da transmissão que este campo pode definir, sem que haja repetição?**
- Confirmação e JanelaAnunciada:
  - Transportam informações sobre o fluxo de dados indo na outra direção

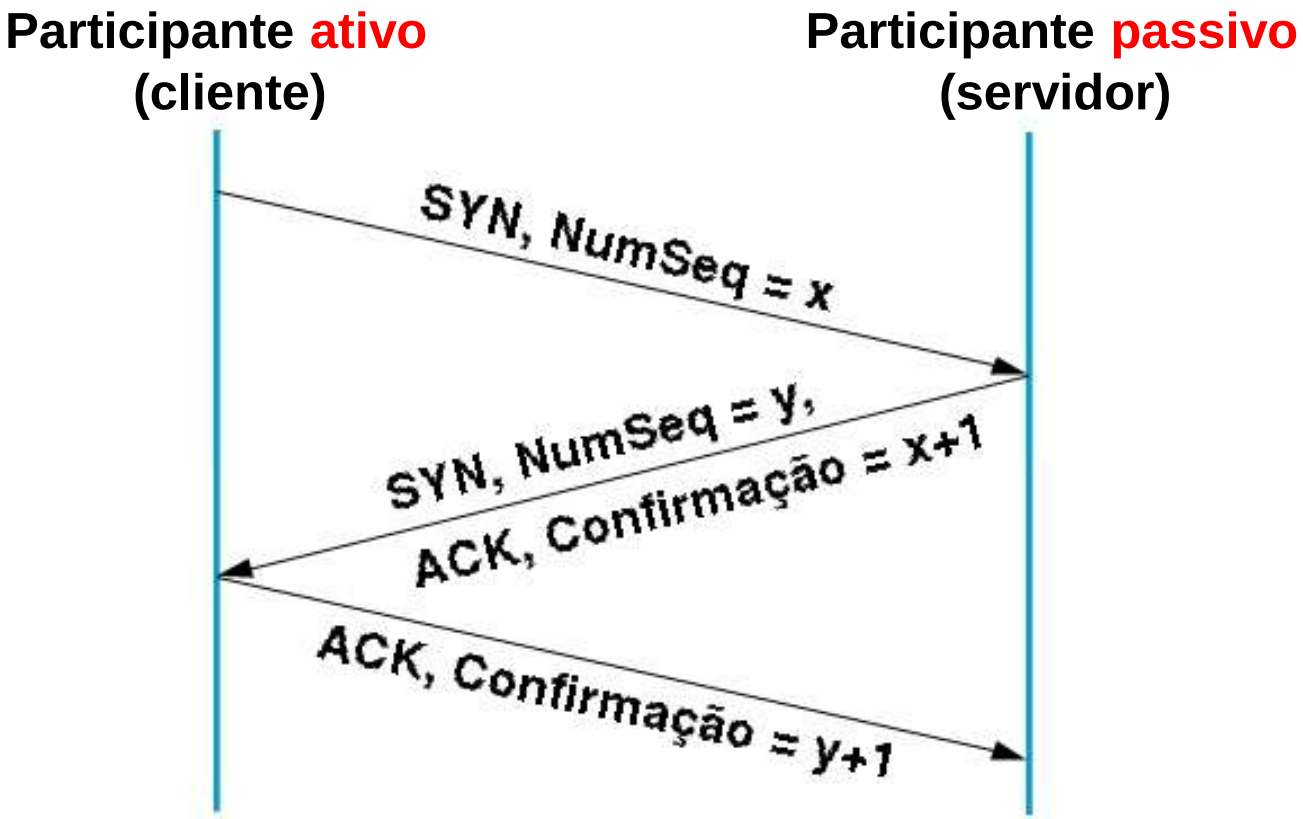
Segmento TCP

- **Cada conexão é identificada por 4 valores:**
  (PortaOrig, EndIPOrig, PortaDest, EndIPDest)
  - mudança em um ou mais destes valores caracteriza uma diferente conexão;
  - de uma mesma porta de origem podem sair diferentes conexões;
  - em uma mesma porta de destino podem chegar diferentes conexões.

- **Janela deslizante + controle de fluxo:**
  NumSeq, Confirmação, JanelaAnunciada
  - a confirmação de recebimento (ACK) é separada do processo de restituição de créditos para transmissão (tamanho da janela - W);
  - só existe o ACK de confirmação - retransmissão ocorre por timeout.

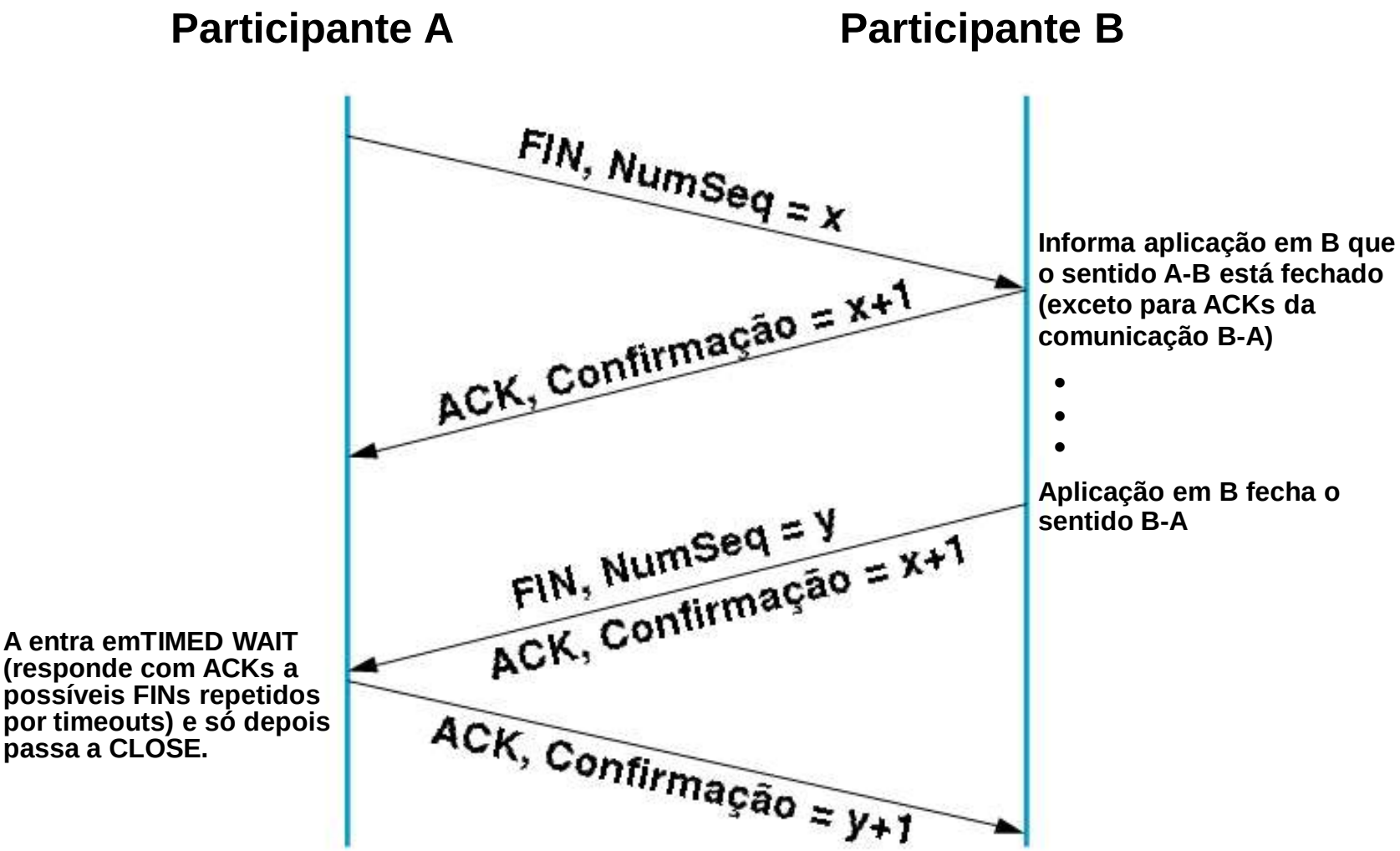


Conexão - estabelecimento e término



Handshake de Três Vias (Three-Way Handshake**)**
para estabelecimento da conexão TCP

Conexão - estabelecimento e término



Handshake de Três Vias para término da conexão TCP

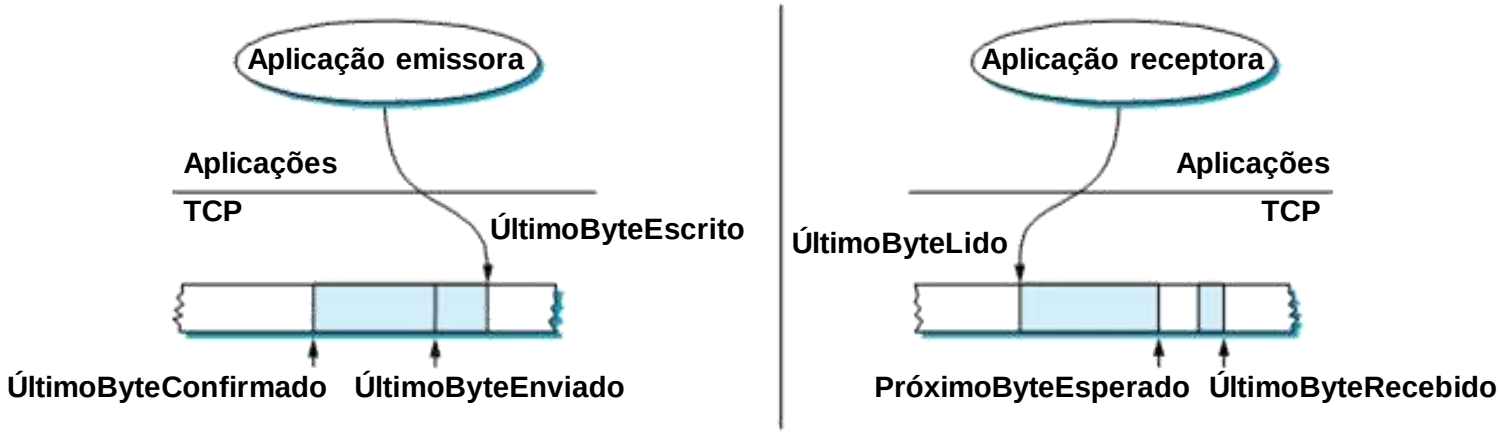
A TCP joke...

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out. ... Hello, would you like to hear a TCP joke?"

Janela Deslizante no TCP

- 3 finalidades:
  - Controle de Fluxo / Entrega ordenada / Controle de congestionamento

- **Entrega confiável e ordenada:**



Lado emissor

- ÚltimoByteConfirmado <= ÚltimoByteEnviado
- ÚltimoByteEnviado <= ÚltimoByteEscrito

Lado receptor

- ÚltimoByteLido < PróxByteEsperado
- PróxByteEsperado <= ÚltimoByteRecebido + 1

Controle de Fluxo no TCP

- Mecanismo que evita que transmissor inunde receptor/rede com dados que não podem ser lidos.
  - Assim, receptor estrangula transmissor anunciando uma janela que possa armazenar dados em seu buffer.

    EX: Transmissor enviou 40KB, janela = 65KB. → Transmissor só pode enviar mais 25 KB até que receba uma confirmação de aumento de janela.

- Receptor confirma recebimento de um pacote com um ACK, mesmo sem aumentar o tamanho da janela de recepção.

- Quando receptor informa J*anelaAnunciada = 0*, transmissor não tem permissão para enviar dados. Em algum momento J*anelaAnunciada* ≠ 0. Como transmissor saberá disso?

Controle de Fluxo no TCP

• Quando receptor informa J*anelaAnunciada* = *0*, transmissor não tem permissão para enviar dados. Em algum momento J*anelaAnunciada* ≠ 0. Como transmissor saberá disso?

  • Receptor não vai enviar um pacote dizendo: “me envie mais dados!”

• Quando *JanelaAnunciada* = 0, transmissor continua enviando um segmento com 1 byte de dados com certa frequencia.

  • Provavelmente estes segmentos não serão aceitos

  • Uma destas sondas de 1 byte dispara uma resposta que informa uma ***JanelaAnunciada* > 0**.

TCP - Política de Transmissão

- A janela de transmissão está aberta...

- Como o TCP pode transmitir um segmento?

Questão:

Quantos bytes acumular para compor e enviar um segmento?

- O TCP mantém uma variável MSS (maximum segment size) e envia assim que tiver coletado MSS Bytes do processo transmissor.

Política original:

se aplicação comandou um **PUSH** ⇒ envie o que estiver pendente;

- **Quando um timer dispara.**

TCP - Política de Transmissão

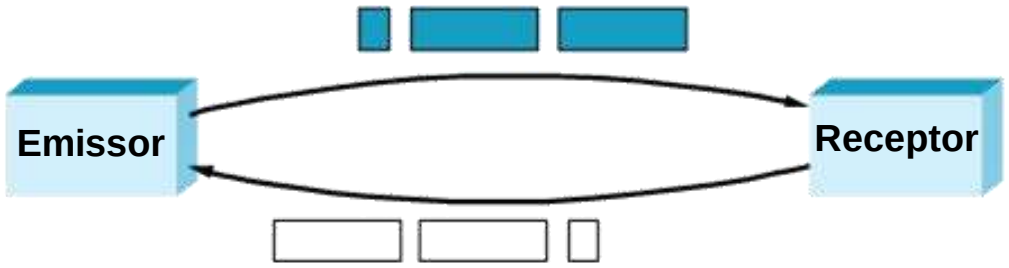
- **Problema: Silly Window Syndrome (SWS):**

• Considere a situação:

• Transmissor acumulou bytes para enviar, mas a janela esta atualmente fechada.

• Um ACK abre a janela o suficiente para transmissor enviar MSS/2 bytes. Transmite esta quantidade ou não???

•Pense em uma correia transportadora com recipientes cheios (segmentos) e recipientes vazios (ACKs) seguindo na direção contrária.



•Se transmissor preenche agressivamente recipiente vazio, então qualquer recipiente pequeno é introduzido no sistema indefinidamente...

TCP - Política de Transmissão

- **MSS (**maximum segment size**):**
  - máximo em bytes que um segmento pode carregar sem causar fragmentação no IP local;
  - default = 536 bytes, mas há negociação do MSS via SYNs.
- **Evitando SWS no receptor:**
  - só anuncie ao emissor um novo valor de janela quando acumular espaço livre igual a MSS ou igual à metade do buffer de recepção, o que for menor.
- **Evitando SWS no transmissor:** use o **Algoritmo de Nagle** :
  - se não houverem ACKs pendentes, a aplicação produzir bytes e a janela permitir, envie o que puder;
  - se houverem ACKs pendentes e a aplicação produzir bytes e/ou houverem bytes a enviar, espere por todos os ACKs acumulados ou espere até completar MSS bytes, para então enviar (se janela permitir);
  - ignore qualquer **PUSH**.
- **Algoritmo de Nagle:**
  - auto-clock;
  - pode ser desligado (opção **TCP_NODELAY** no soquete).

Cálculo do Timeout no TCP

- TCP retransmite segmento se ACK não chegar dentro de um tempo (*TimeOut*).
  - *TimeOut* calculado em cima do RTT.

- **Problema:** RTT é variável durante a conexão (!).
  - **Solução:** mecanismo de retransmissão adaptável.
  - **Algoritmo Original:**
    - Mede **RTTAmostra**, para cada par segmento/ACK.
    - Calcula média ponderada de RTT:

      $$\mathbf{RTTEst} = \alpha \times \mathbf{RTTEst}_{\text{anterior}} + (1 - \alpha) \times \mathbf{RTTAmostra}$$

      onde α entre 0,8 e 0,9 (*suaviza variações pontuais*)

  - **α peq:** influenciado por flutuações na rede
  - **α gde:** não se adapta às mudanças na redes

Cálculo do Timeout no TCP

• TCP usa *RTTestimado* para calcular o timeout :

– Define timeout com base em **RTTEst**:

TimeOut = β x RTTEst

onde β = 2 (*conservador*)

•*Se ACK não chega dentro do timeout, então dispara retransmissão*

• **Problema com esta técnica???**

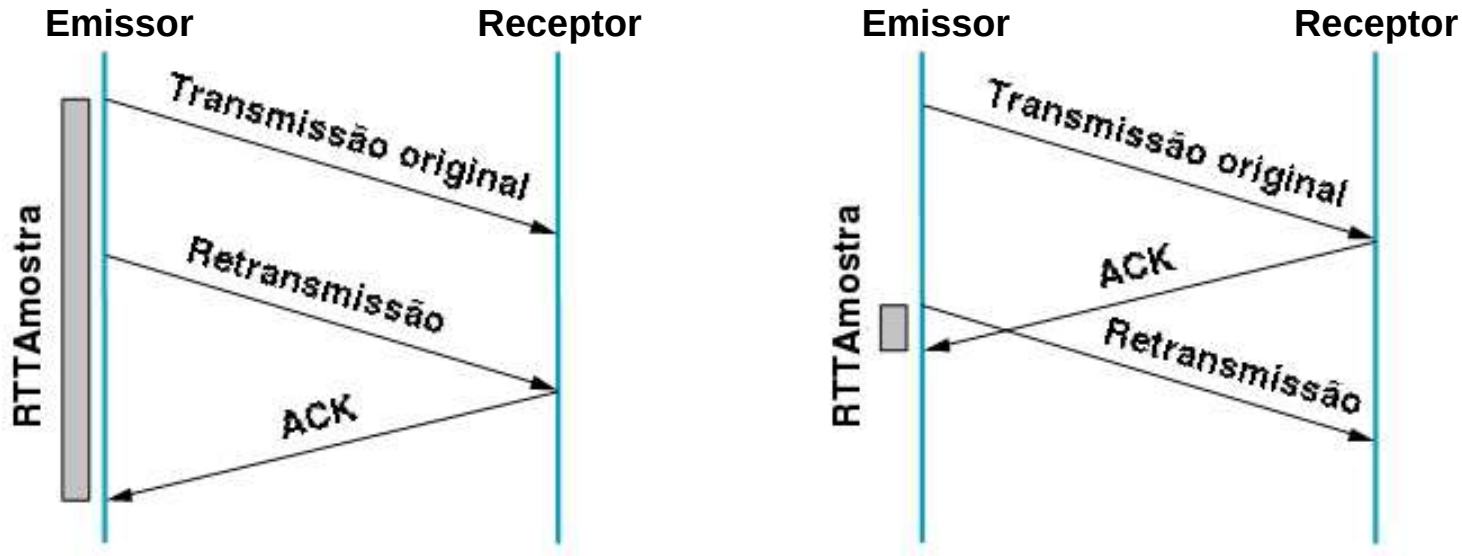
• ACK não confirma realmente uma transmissão, mas sim o recebimento dos dados (!).

• Impossível determinar se ACK é da 1ª ou 2ª transmissão para fins de medição de RTT ☹.

Timeout - Algoritmo de Karn/Partridge

- **Problema:** após retransmissão, o ACK que chega é referente ao segmento original ou ao segmento retransmitido?

  ⇒ Pode levar a falsos valores de RTT:



- **Solução (Karn/Partridge):** após cada retransmissão:
  - ignora amostra de **RTT**;
  - dobra o valor de **TimeOut** (exponential backoff).

Congestionamento → segmentos perdidos.

Origem não deve reagir muito agressivamente a um TimeOut

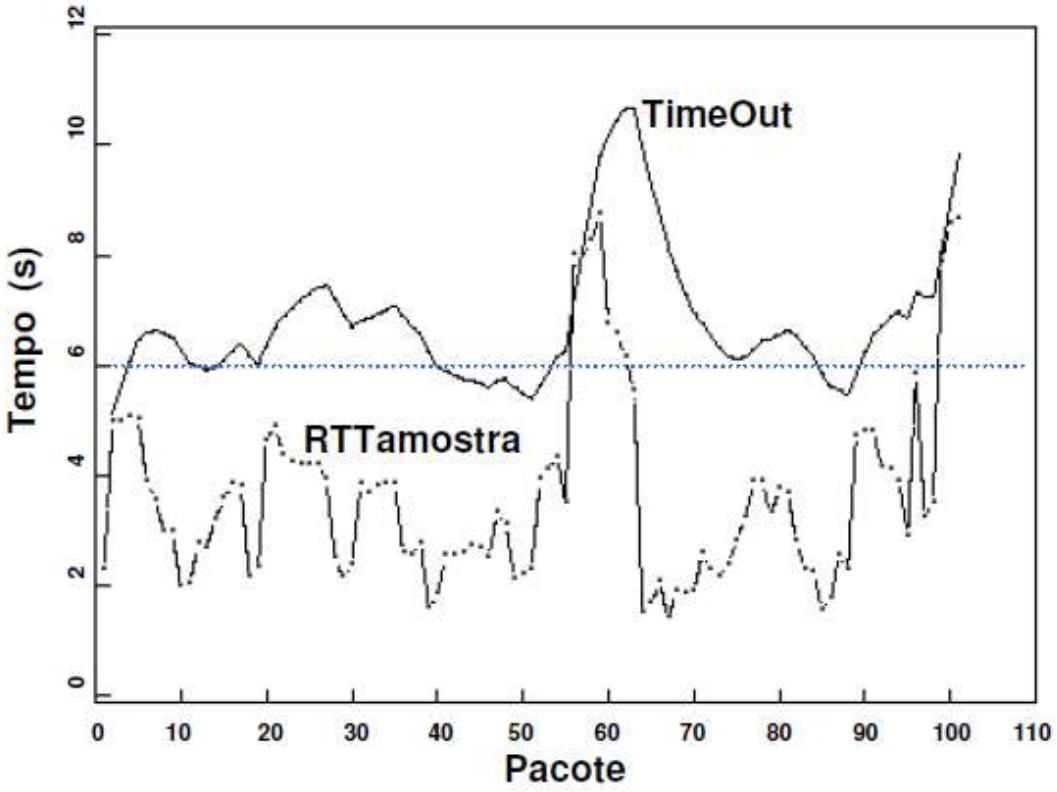
Timeout - Algoritmo de Jacobson/Karels

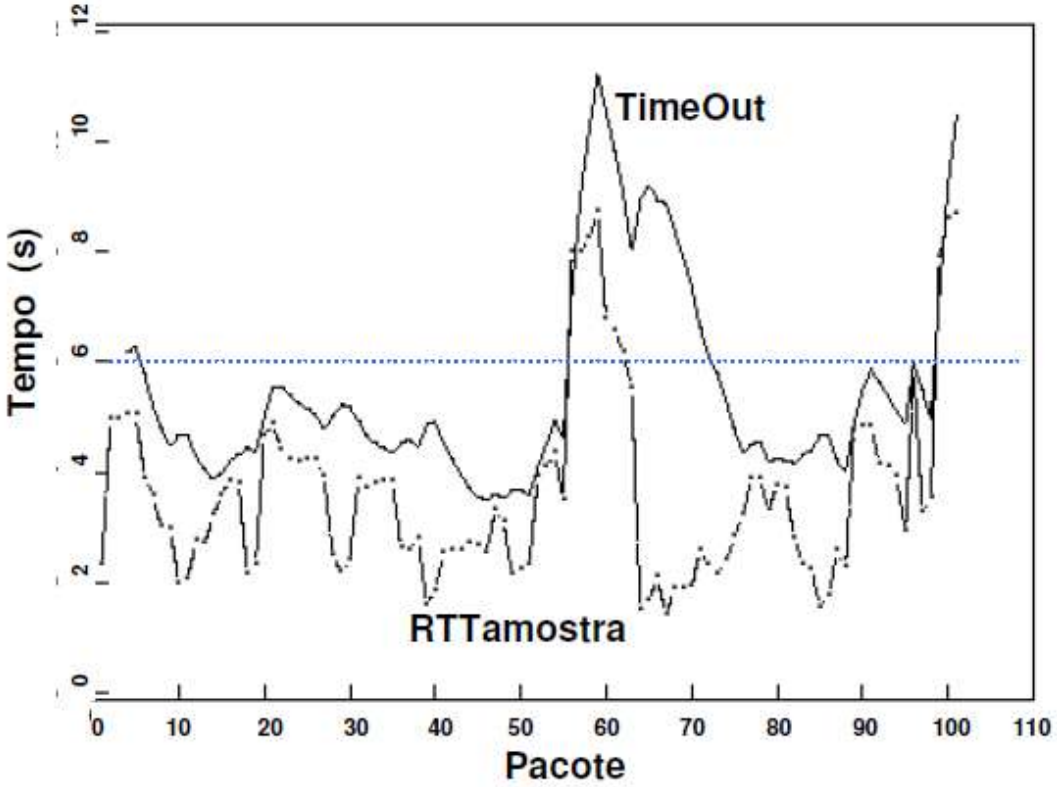
- **Problema:** Karn/Partridge melhorou, mas não resolveu o congestionamento.
- **Solução (Jacobson/Karels):** levar em conta a variância das amostras de RTT:

• Se variação entre as amostras for pequena → OK , pode confiar no RTT estimado.

• Se variação entre as amostras for grande, o valor do timeout não deve estar muito ligado ao RTT estimado.

Timeout - Algoritmo de Jacobson/Karels





Cálculo de timeout segundo RFC 793:

$RTTEst = \alpha \times RTTEst_{anterior} + (1 - \alpha) \times RTTAmostra$

$TimeOut = \beta \times RTTEst$

$\alpha = 0{,}9$, $\beta = 2$

Cálculo de timeout segundo Jacobson/Karels:

$Diferença = RTTAmostra - RTTEst_{anterior}$

$RTTEst = RTTEst_{anterior} + (\delta \times Diferença)$

$Desvio = Desvio_{anterior} + \delta (|Diferença| - Desvio_{anterior})$

$TimeOut = \mu \times RTTEst + \phi \times Desvio$

$\delta = 1/8$, $\mu = 1$ e $\phi = 4$

Fonte: JACOBSON, V. Congestion avoidance and control. In: Proceedings of SIGCOMM '88**, Stanford, CA, August 1988, ACM.**