Redes de Computadores

Conexões Ponto-a-Ponto:

Codificação
Enquadramento
Detecção de erros
Repetição de pacotes
Janela deslizante

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Objetivo

- Discutir os problemas para se conseguir transmitir dados entre computadores conectados diretamente

Observação/referências extras

- O livro fala pouco sobre tecnologias de transmissão e detalhes de hardware
- Para maiores informações sobre esses assuntos:

° Redes de Computadores, Tanenbaum, Ed. Campus, cap. 2

(Computer Networks, 3rd Ed.)

° Redes e Sistemas de Comunicação de Dados, Stallings, caps. 3-6

(Data and Computer Communication, 6th Ed.)

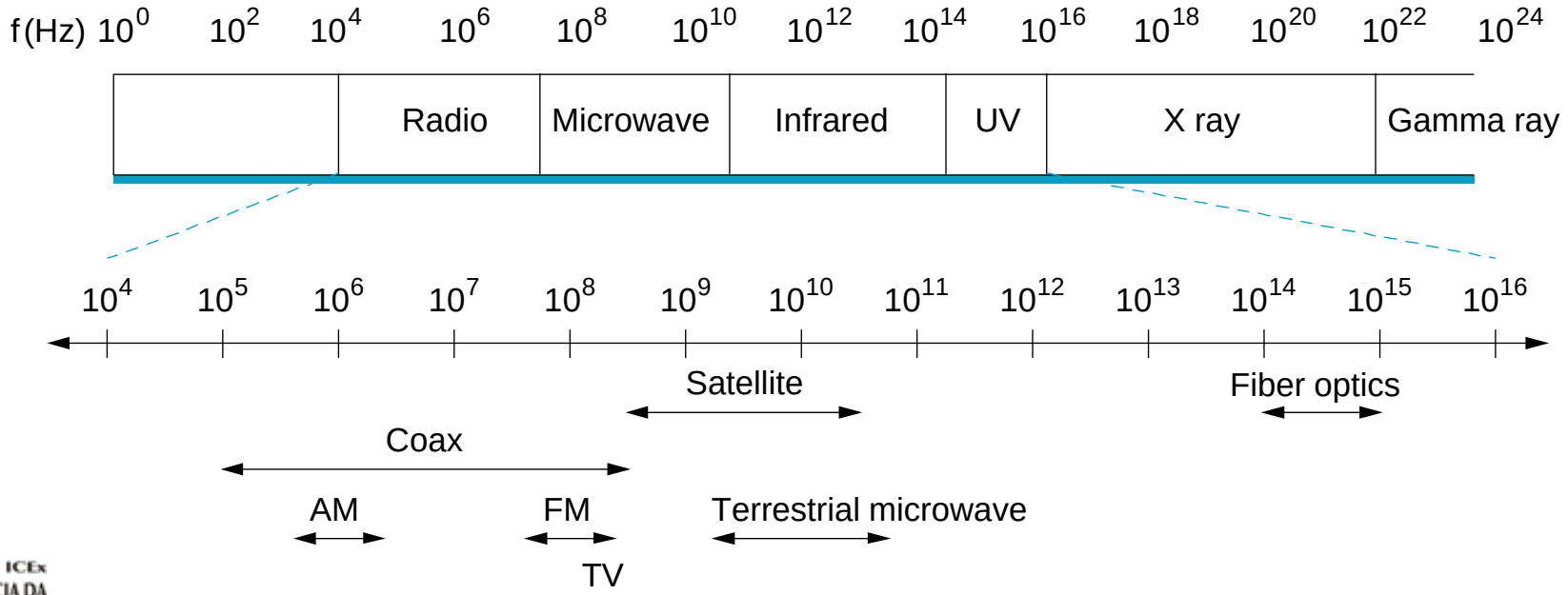
Conexões ponto-a-ponto

...na construção de redes é preciso conectar os nós e permitir que se comuniquem

- Ligação mais simples: conexões diretas
- Neste caso é preciso tratar de certos problemas:
  - Codificação: como os bits viajam pelo *link*?
  - Enquadramento: como identificar os limites entre pacotes?
  - Detecção e correção de erros:
    - como detectar quando algo saiu errado?
    - o que fazer nesse caso?
  - Controle de fluxo: minha Sun IPX 75 Mhz x seu P4 3,3 GHz
- Esses problemas são tratados na camada de **enlace**

Limitações de frequência

- Sinais se propagam sobre um meio físico
  - Modulação de ondas eletromagnéticas
  - P. ex., variação de tensão ou corrente
- Informação codificada sobre sinais eletromagnéticos



Limitações de frequência

- ... lá do cálculo: Série de Fourier
- Sinais elétricos podem ser representados pela combinação de infinitas senóides (harmônicos)

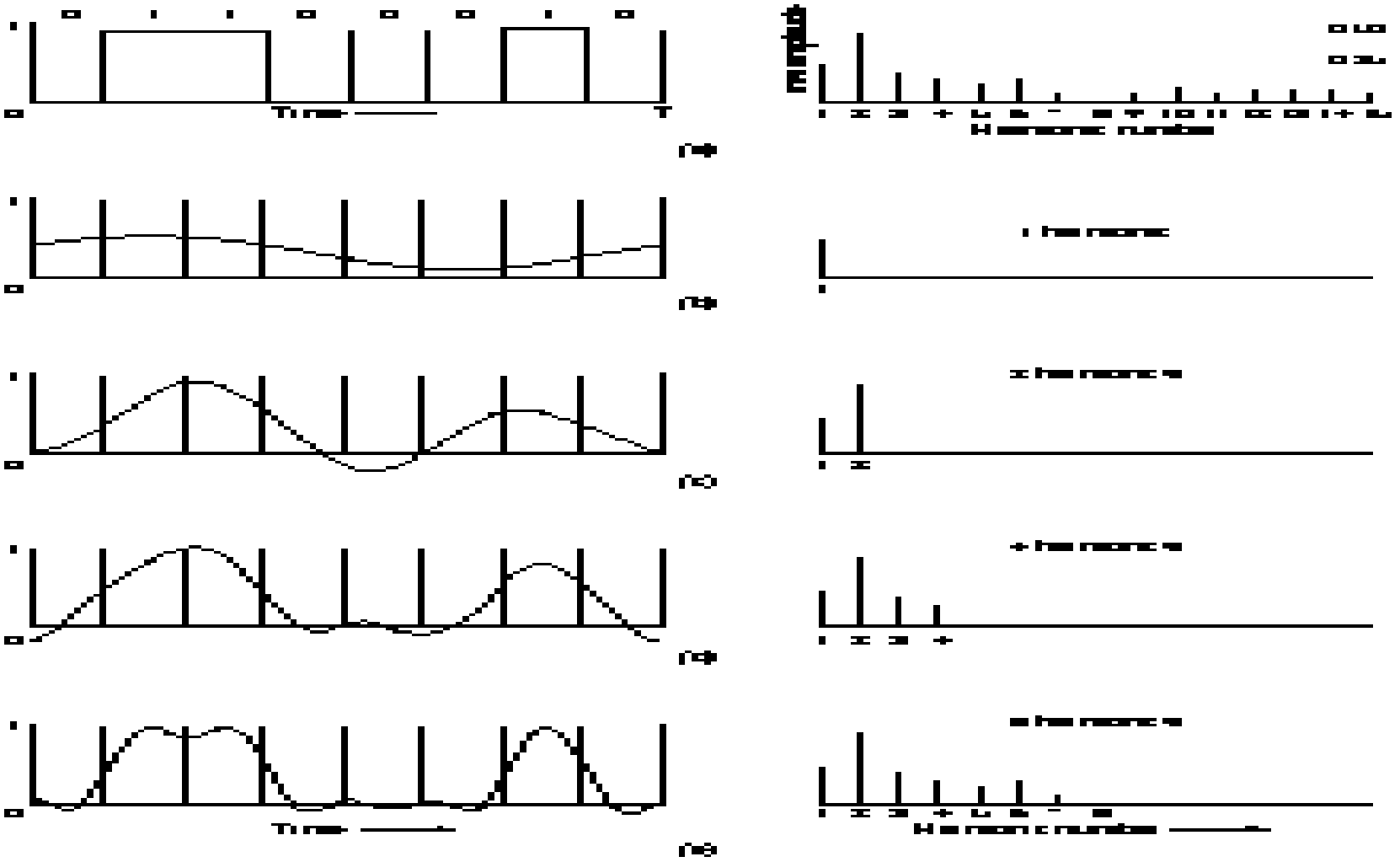
°Diferentes meios têm diferentes restrições em frequência

°Diferentes harmônicos sofrem atenuações diferentes no meio

°Alguns canais limitam intencionalmente a banda disponível

°Conteúdo harmônico alterado gera distorções no sinal

Sinais limitados por frequência

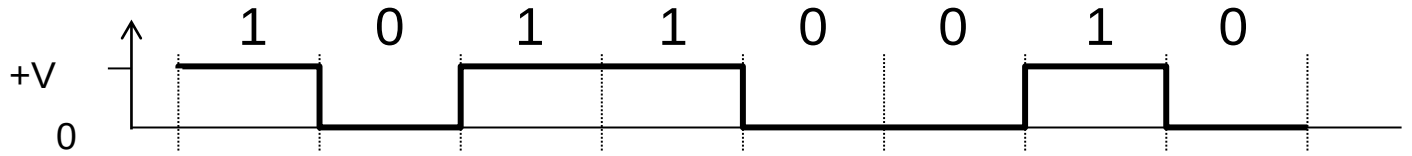


Codificação

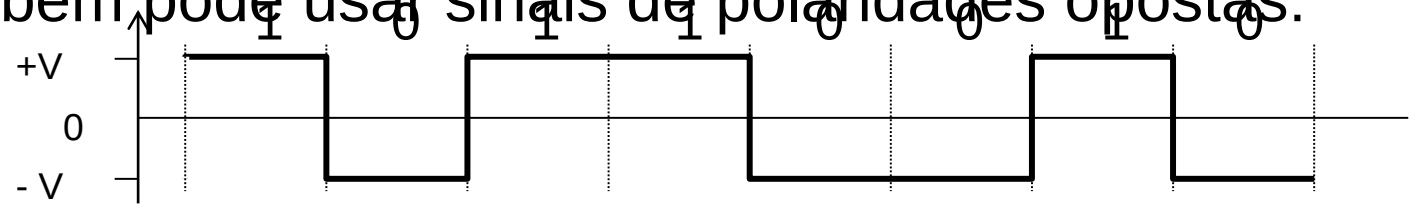
- Como o receptor sabe quando o transmissor está enviando zero ou um?

Codificação

- Solução mais simples:
  - 0 representado por sinal baixo e 1 por sinal alto
  - Conhecido por “Non-return to zero” (NRZ)

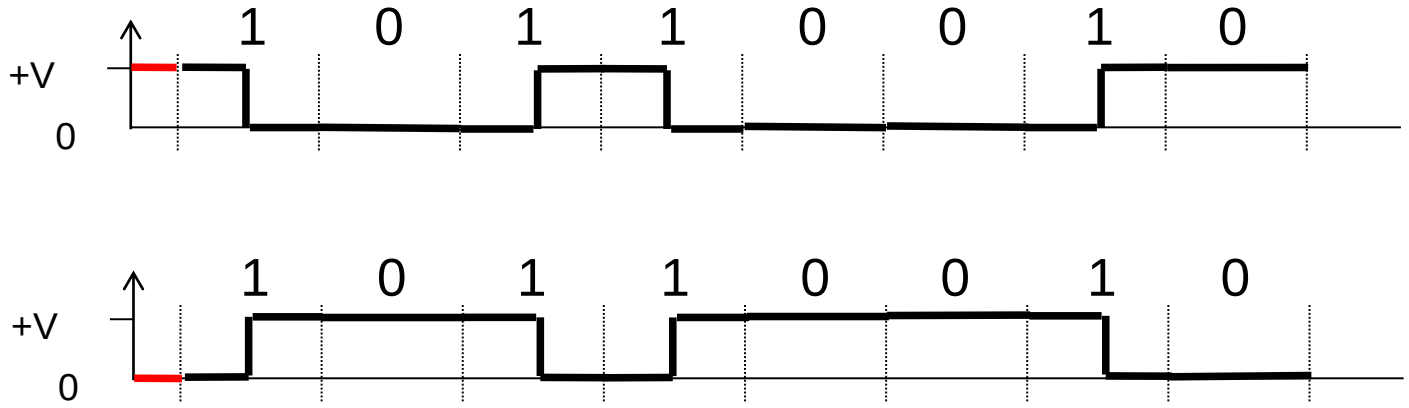


  - Também pode usar sinais de polaridades opostas:



Problema: 1s ou 0s consecutivos

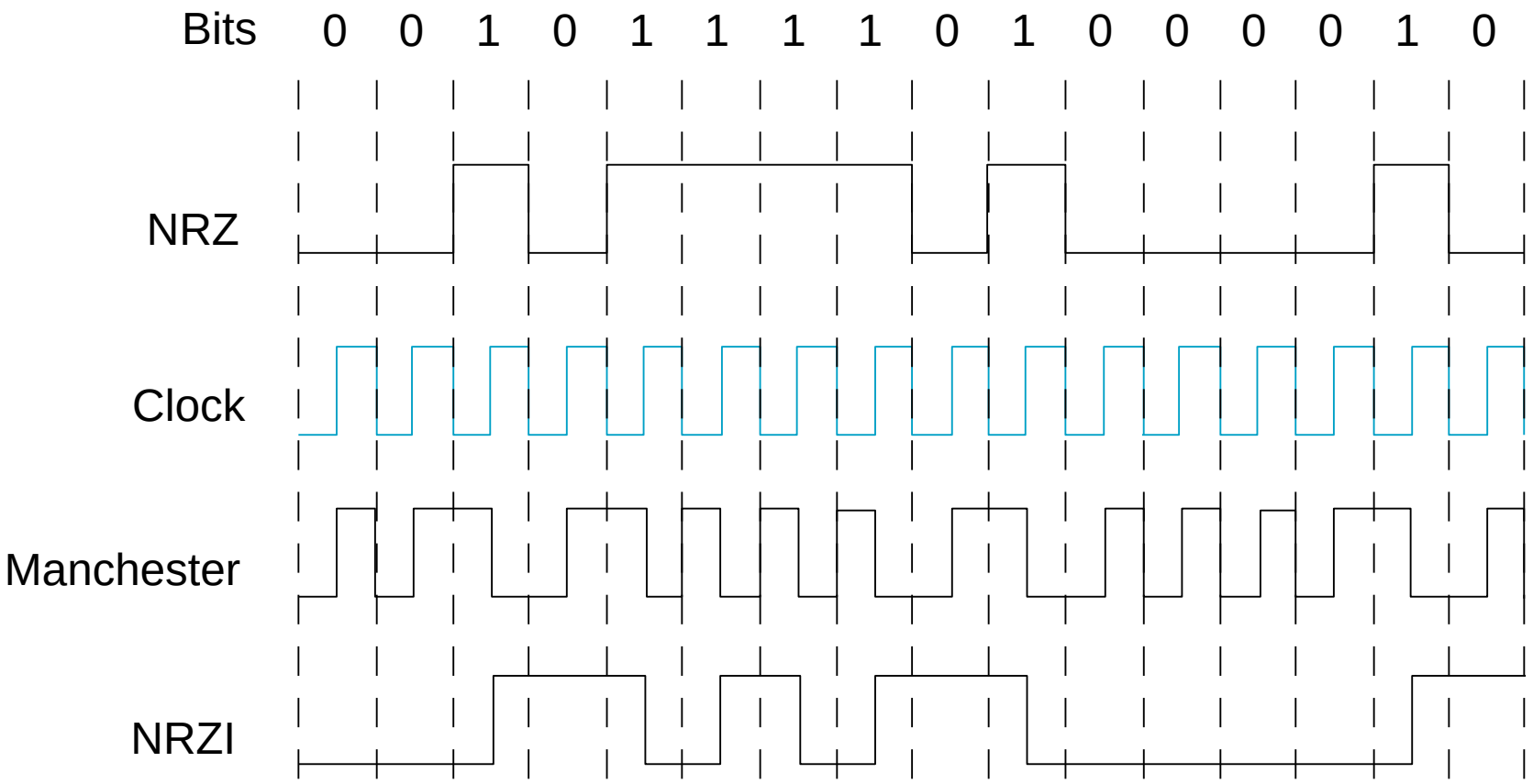
- Não há como recuperar o relógio da transmissão
  - Dois relógios sempre tem algum desvio relativo
- Tentativa de solução: forçar transições
  - *Non-return to Zero Inverted* (NRZI)
- um 0 mantém o sinal no nível anterior; um 1 inverte o sinal
- resolve o problema de 11111111, mas e 00000000?



Codificações alternantes (“alternativas”)

- Garantem uma transição por bit (no meio do bit)
- Manchester
  - bit 0: transição baixo->alto; bit 1: alto ->baixo
  - transição no início do bit se necessário
- Manchester diferencial
  - bit definido pela transição (ou não) no início do bit
  - bit 0: há transição; bit 1: mantém valor anterior
- Ambos têm eficiência de apenas 50% da banda

Codificações



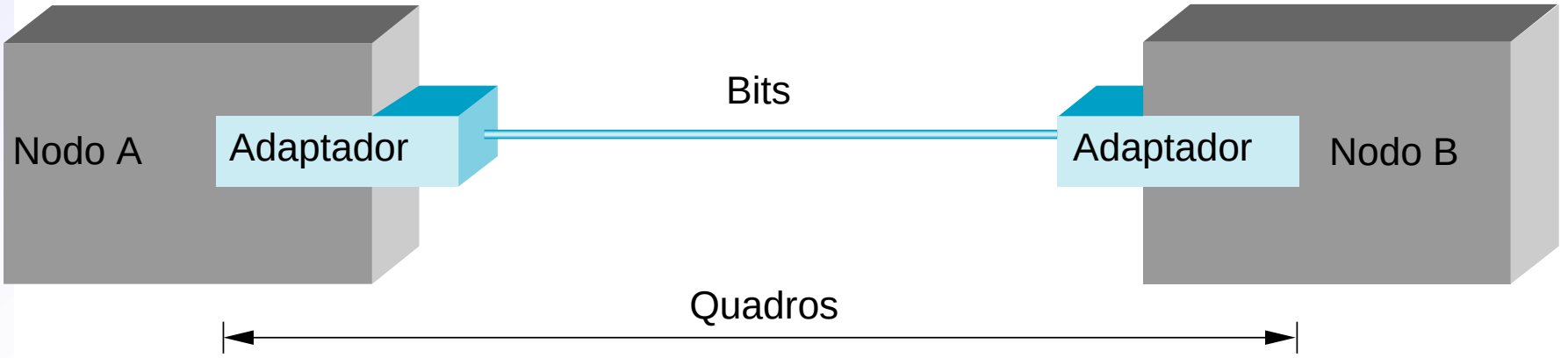
Codificação

- 4B/5B: 4 bits de dados codificados com 5 bits no canal
  - combinações de bits escolhidas:
    - não mais que um 0 no início e no máximo dois no final
    - logo, não há mais que três zeros consecutivos
  - sinal resultante é enviado com NRZI
  - algumas combinações não usadas são sinais de controle

0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

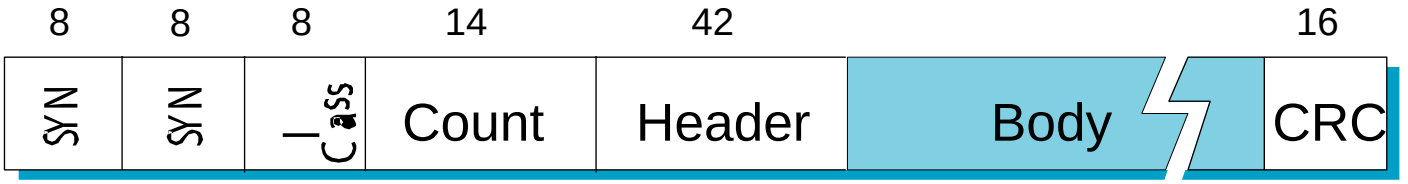
Framing (enquadramento)

- Identificação dos limites de cada pacote (quadro)
- Transforma sequências de bits em “quadros” (*frames*)
- Normalmente implementado na interface de rede
- Exige a inserção de marcas (molduras)



Enquadramento por contagem

- Exemplo: DDCMP
- Envia o comprimento do quadro no cabeçalho



°problema: campo de tamanho pode ser adulterado

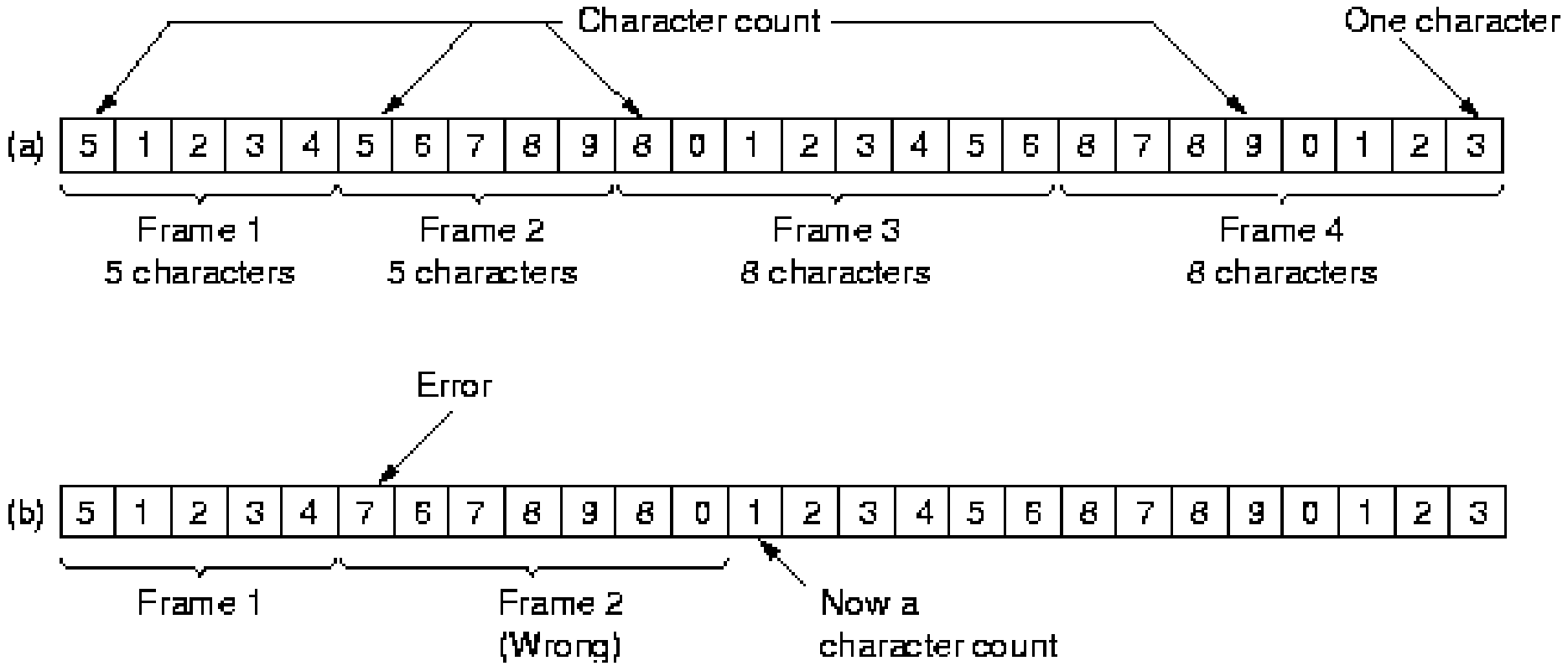
°solução: use um código de detecção de erros

- ATM usa esta solução (taxas de erro muito baixas)

- Mas como saber onde começar outro quadro?

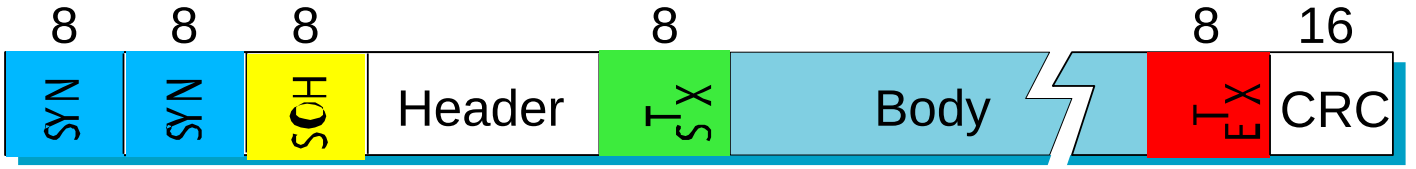
Contagem de caracteres

- Problema: erro causa perda da sincronização

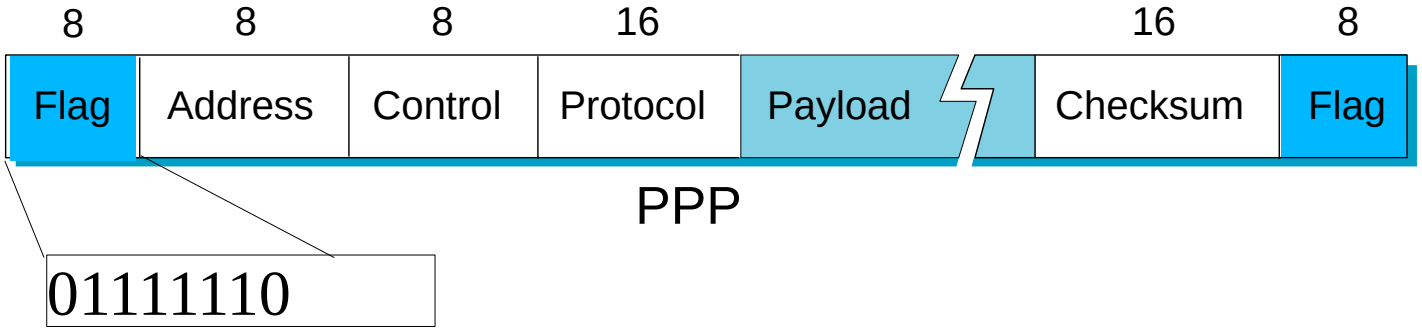


Enquadramento com sentinelas (bytes)

- Quadro é marcado com bytes com valor especial
  - fácil de ser implementado por *software*
  - e.g., BISYNC, PPP e SLIP



BISYNC



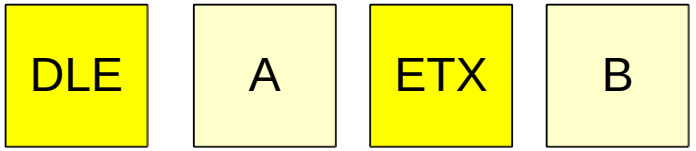
PPP

Enquadramento com sentinelas (bytes)

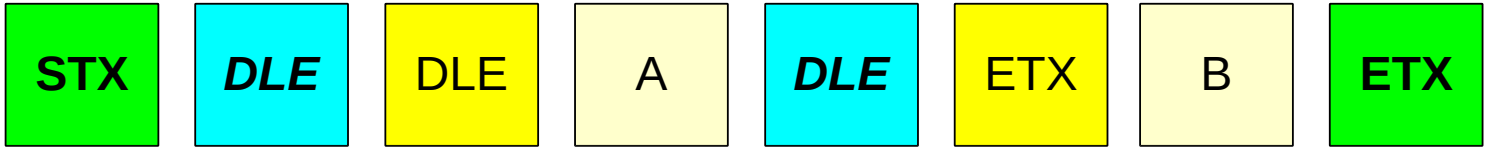
- Problema: os bytes especiais podem existir dentro dos dados a serem transmitidos!
- Solução: *byte stuffing*
  - transmissor: acrescenta um "escape" antes de qualquer byte **nos dados** que seja igual às marcas
  - receptor: remove todo primeiro "*escape*" dentro do pacote e nunca remove o byte seguinte
  - também conhecido como "*escape sequences*"

Enquadramento com sentinelas (bytes)

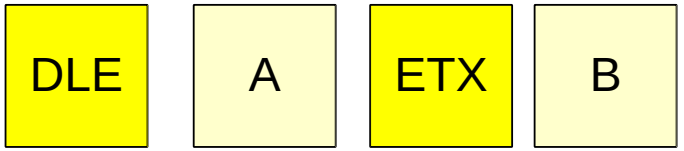
Dados enviados pela camada de rede



Sentinelas + character stuffing

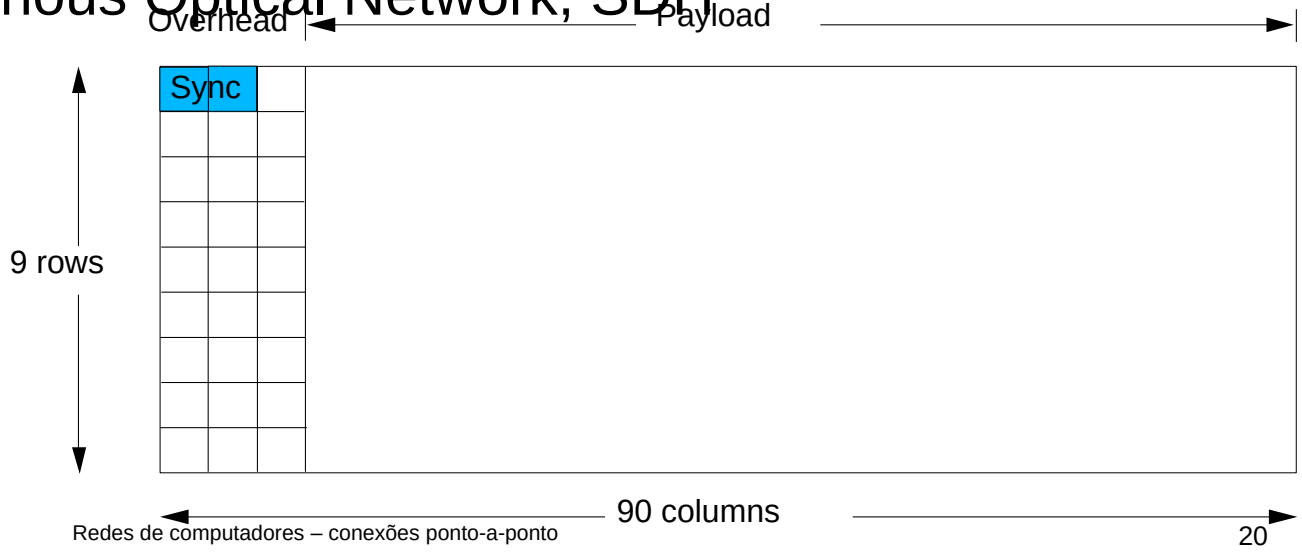


Dado recebido após a remoção de marcas e stuffing



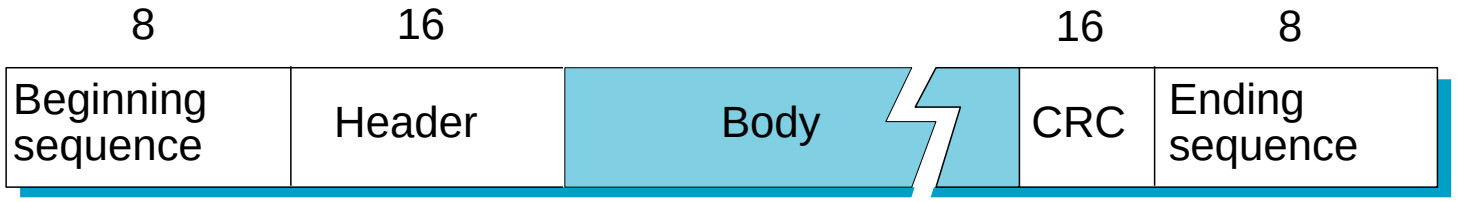
Enquadramento pelo relógio

- Na verdade, contagem – mas sempre valor fixo.
- Transmissor e receptor: relógios de grande precisão
- Quadros têm duração bem definida
  - Sinais precisam ter mínimo de transições para sincronizar
  - Melhor utilização da banda (não requer *stuffing*)
- ex.: SONET: Synchronous Optical Network, SDH
  - Mais complexa



Enquadramento com sentinelas (bits)

- Quadro é marcado com um padrão especial: 01111110
  - precisa ser implementado pelo *hardware;* e.g., HDLC, SDLC



Enquadramento com sentinelas (bits)

- Problema: o padrão especial pode existir nos dados!
- Solução: *bit stuffing*
  - transmissor: insere 0 após cinco 1s consecutivos
  - receptor: reage à chegada de cinco 1s consecutivos
    - se sexto bit é zero, remove-o e continua montando o quadro
    - se sexto bit é 1, decide que achou uma marca
  - enchimento garante transições em NRZI

Enquadramento com sentinelas (bits)

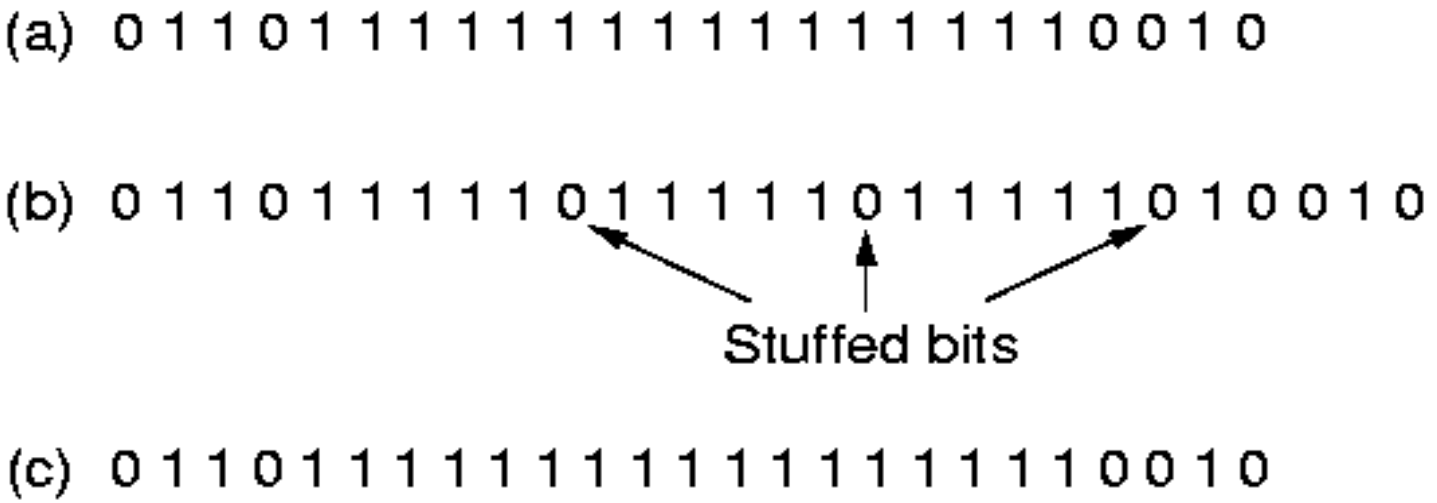


Fig. 3-5. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

Detecção e correção de erros

- Lei de Murphy aplicada a redes de dados:

Erros acontecem!!!

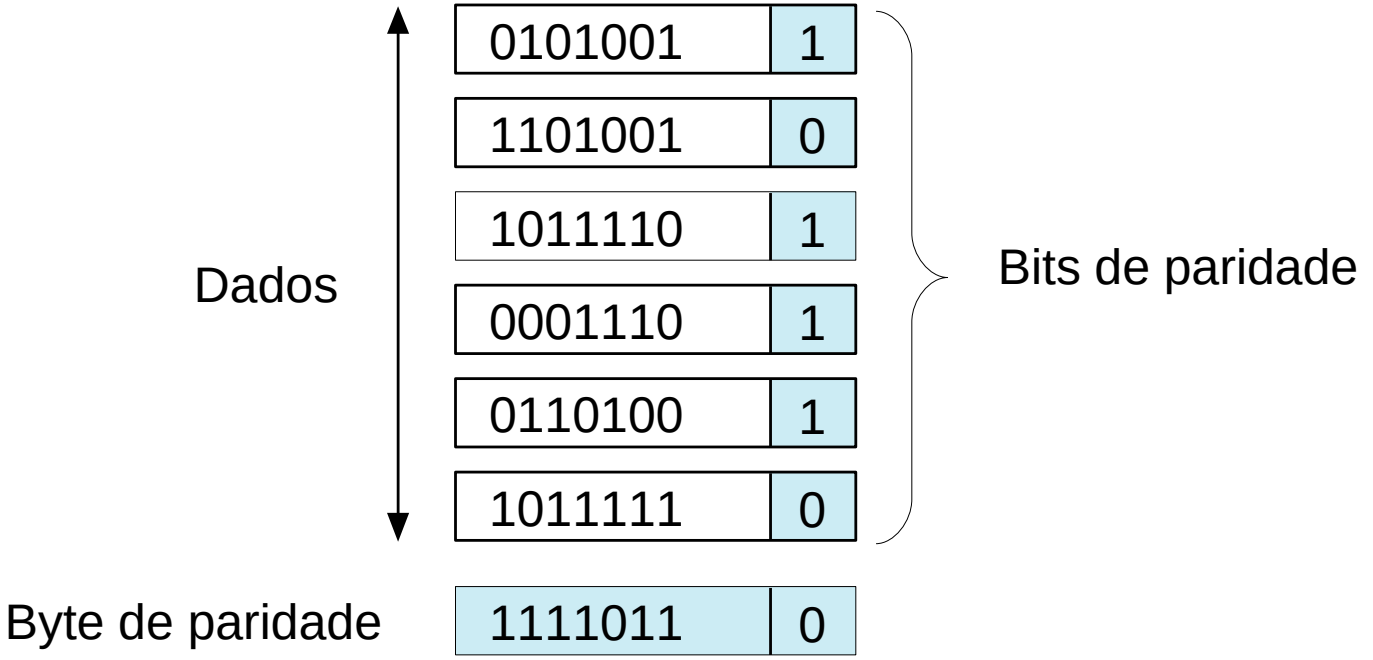
- **Detecção** de erros: bits de paridade
  - *checksum*
  - CRC
- Correção: informação é suficiente para recuperação
  - códigos de *hamming*
  - retransmissão!

Bits de paridade

- Tornam sempre par (ímpar) o número de 1s em um byte enviado
  - Simples, permite detecção de erros individuais
  - Normalmente implementado em *hardware*
- Paridade par: 0101101 -> 0101101 **0**
- Paridade ímpar: 0101101 -> 0101101 **1**

Bits de paridade

- Paridade pode ser também “bidimensional”:
  - dados são considerados como formando matrizes
  - calcula-se um bit de paridade para cada linha e coluna



Checksum

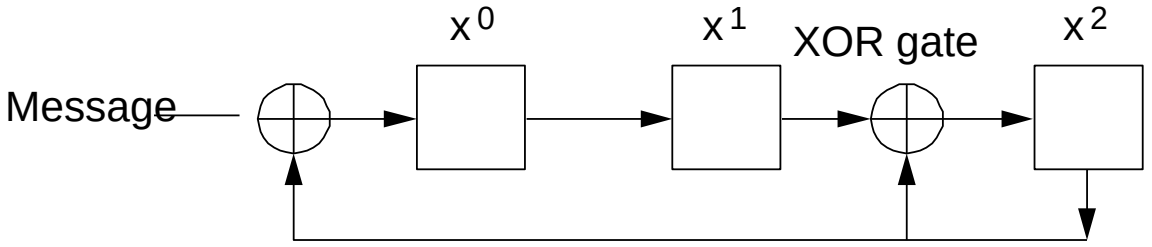
- Tomando os dados como uma sequência de inteiros, calcula-se a soma (módulo um) dos mesmos
- **Complemento** da soma é acrescentado à mensagem
  - Agora, se o receptor calcular a soma dos inteiros da mensagem recebida, essa deve ser 0!
- Permite detectar vários tipos de erros comuns
- Utilizado na Internet (16 bits)
  - Código em C para cálculo do *checksum* no livro

CRC (*Cyclic Redundancy Check*)

- Pacotes tratados como polinômios em base 2
  - "Bom" polinômio gerador é definido
  - Divide-se pacote pelo polinômio gerador
  - Resto da divisão é "subtraído" do pacote a ser enviado
- Receptor ao repetir a operação de divisão no pacote recebido deve encontrar resto zero!

CRC (*Cyclic Redundancy Check*)

- Permite detectar diversos tipos de erro desde que o polinômio gerador satisfaça certas regras
  - toda rajada de erros de comprimento < k
  - qualquer número ímpar de erros individuais
  - todos os erros de um ou dois bits
  - grande parte das demais possibilidades
- Em *hardware:*



Divisão por $x^3+x^2+x^o$

Cálculo do CRC

- Multiplicar mensagem M(x) por x^k
  - equivale a adicionar k zeros ao fim da mesma
- Dividir M(x).x^k por C(x) (polinômio gerador)
- Adicionar o **resto** a M(x).x^k
  - corresponde a substibuir os k zeros acrescentados pelo resto obtido
- Em complemento de 1, adição=subtração=XOR

Cálculo do CRC (exemplo)

- M(x) = 10011010, C(x) = 1101

```
10011010000 | 1101
 1101       -------
            111110001
 1001
 1101
 ----
 1000
 1101
 ----
 1011
 1101
 ----
 1100
 1101
 ----
 001000
    1101
 ------
  101
```

T'(x) = 10011010101

Código de Hamming

- Vários bits de paridade são acrescentados segundo regras especiais
- Essa redundância extra é tal que certos erros podem ser corrigidos
- Procedimento:
  - Numerar bits da palavra resultante
  - Posições = 2^x são bits de paridade
  - Demais são espaços para bits originais
  - Bit k controlado por bits de paridade correspondentes a cada componente de k

Código de Hamming (exemplo)

- Código de Hamming para 1101001

1	2	3	4	5	6	7	8	9	10	11
a	b	1	c	1	0	1	d	0	0	1

- a (bit_1) = $bit_3 + bit_5 + bit_7 + bit_9 + bit_{11}$ = 0
- b (bit_2) = $bit_3 + bit_6 + bit_7 + bit_{10} + bit_{11}$ = 1
- c (bit_4) = $bit_5 + bit_6 + bit_7$ = 0
- d (bit_8) = $bit_9 + bit_{10} + bit_{11}$ = 1

- Hamming = 0 1 1 0 1 0 1 1 0 0 1

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Código de Hamming (exemplo)

- Hamming = 0 1 1 0 1 0 1 1 0 0 1
- Erro no quinto bit: 0 1 1 0 0 0 1 1 0 0 1

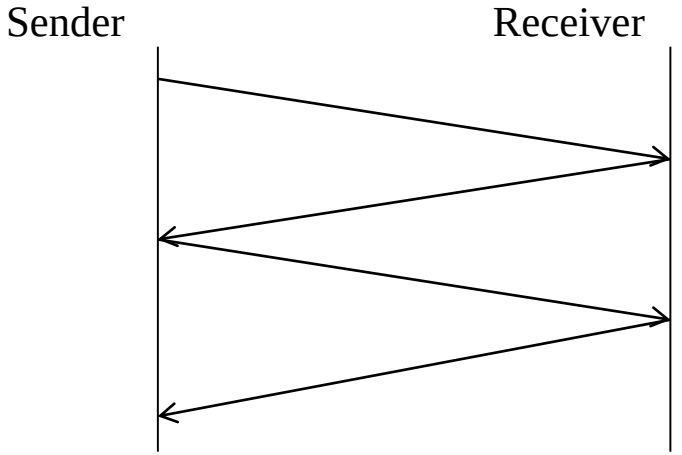
1	2	3	4	**5**	6	7	8	9	10	11
0	1	1	0	0	0	1	1	0	0	1

- a (bit_1) = $bit_1 + bit_3 + bit_5 + bit_7 + bit_9 + bit_{11}$ = 1
- b (bit_2) = $bit_2 + bit_3 + bit_6 + bit_7 + bit_{10} + bit_{11}$ = 0
- c (bit_4) = $bit_4 + bit_5 + bit_6 + bit_7$ = 1
- d (bit_8) = $bit_8 + bit_9 + bit_{10} + bit_{11}$ = 0

- Bit errado: 0 1 0 1

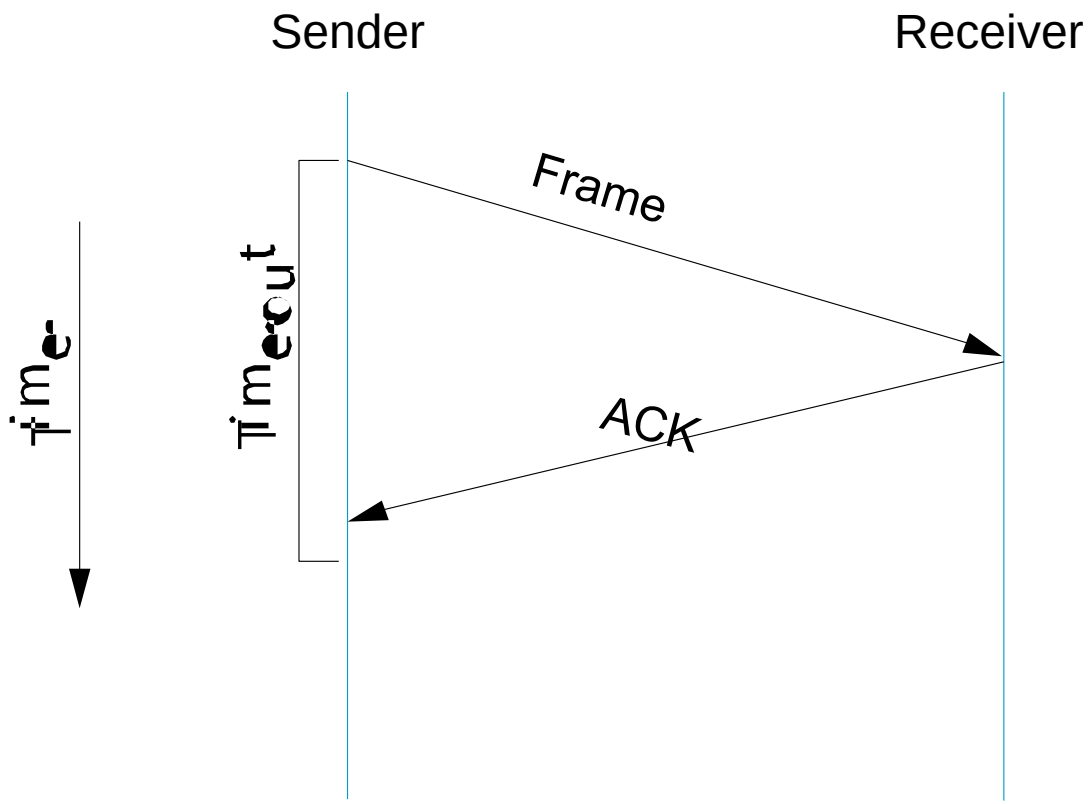
Transmissão confiável

- Se há erros e não é possível recuperar o dado diretamente, é preciso aguardar a confirmação do destinatário (*acknowledgement*, ACK)
- "Stop-and-wait":



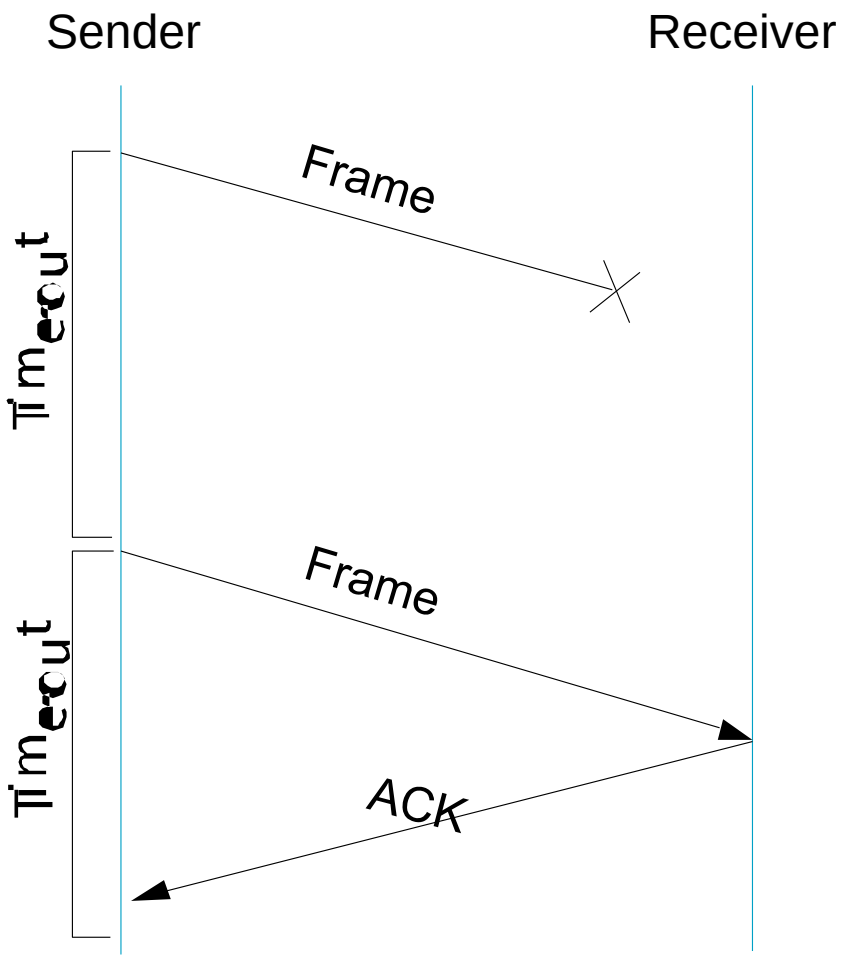
Pacotes são retransmitidos se confirmação não chega dentro de um determinado intervalo de tempo (*timeout*).

Stop-and-wait



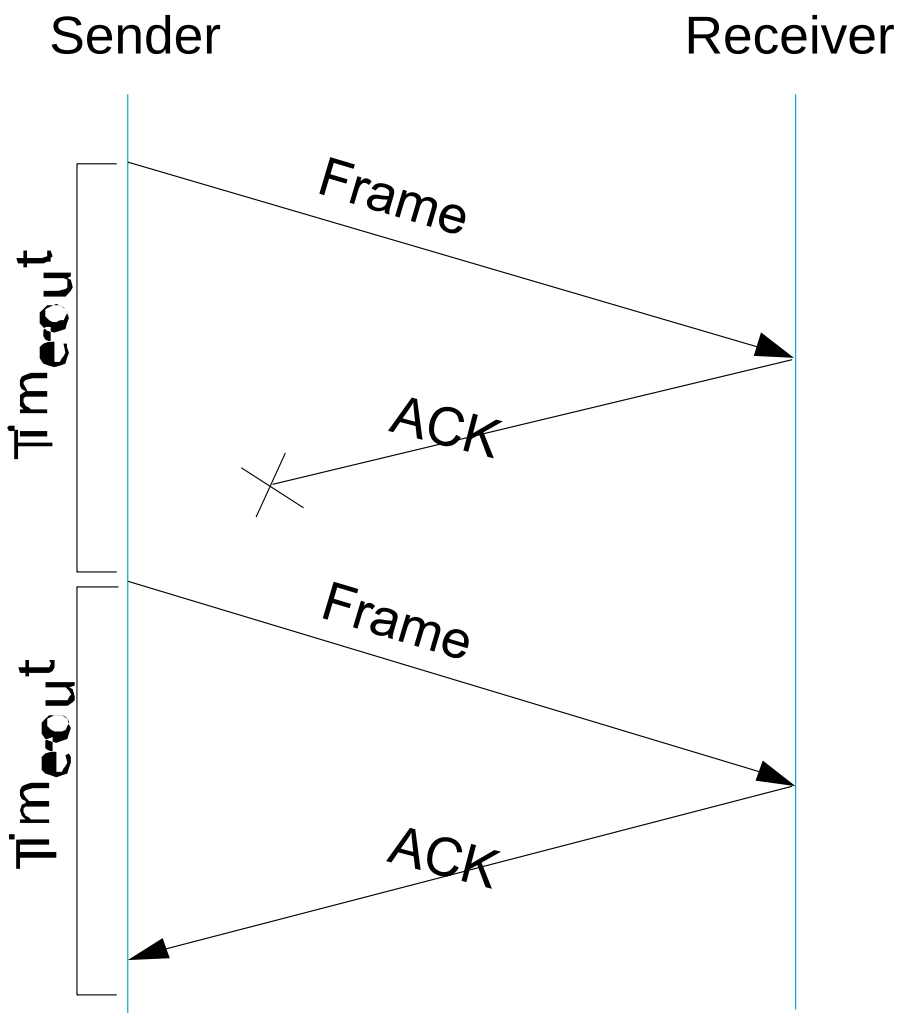
(a)

Stop-and-wait

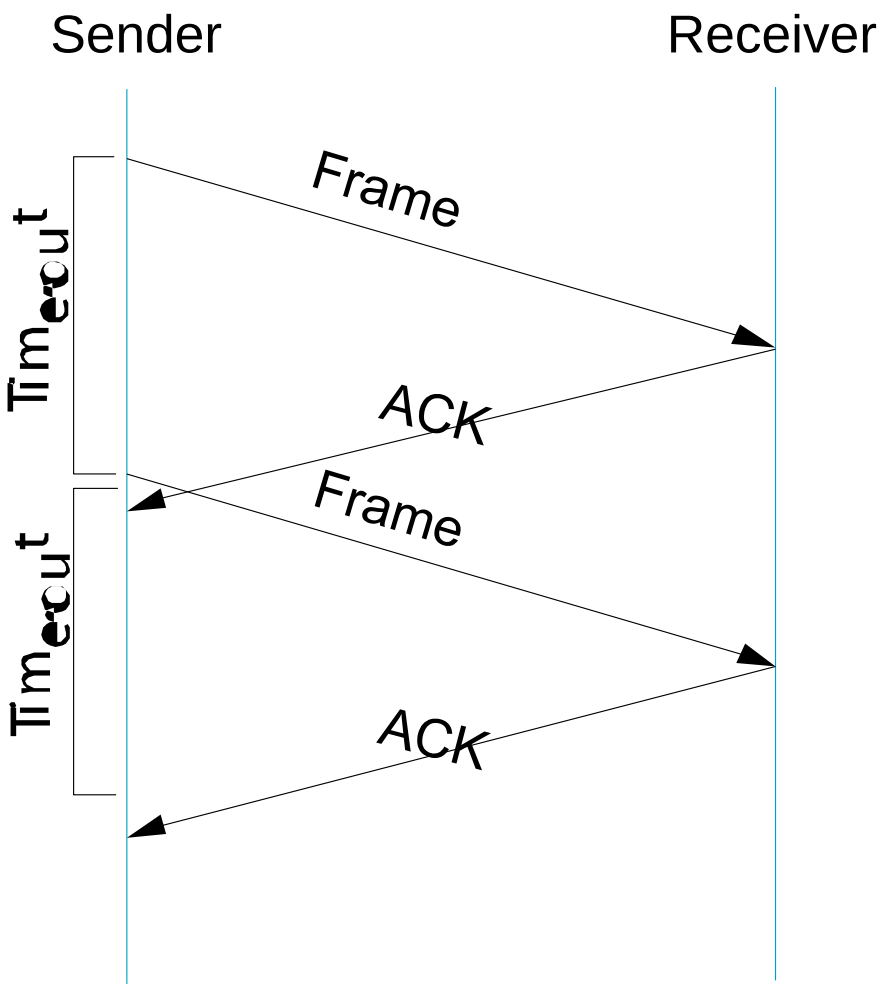


(b)

Stop-and-wait

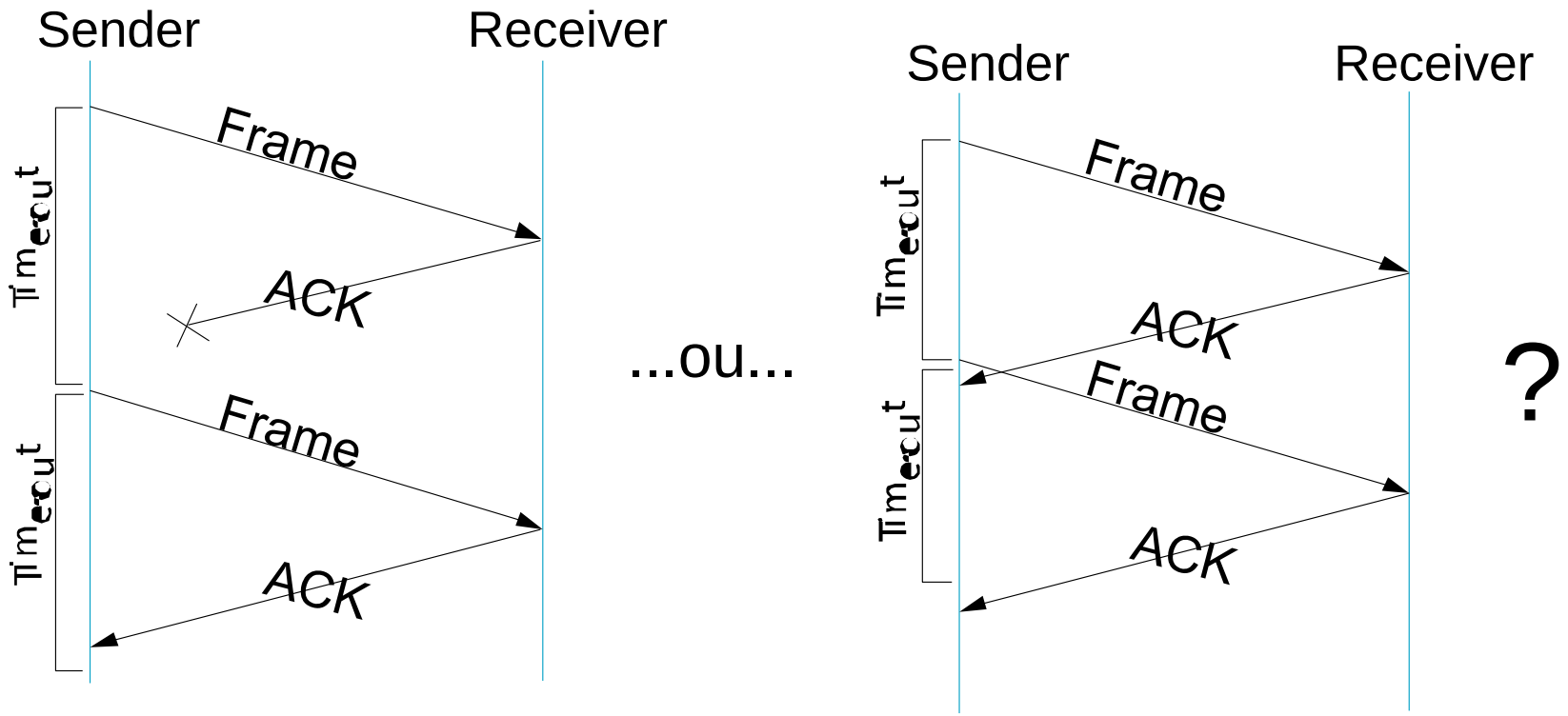


Stop-and-wait



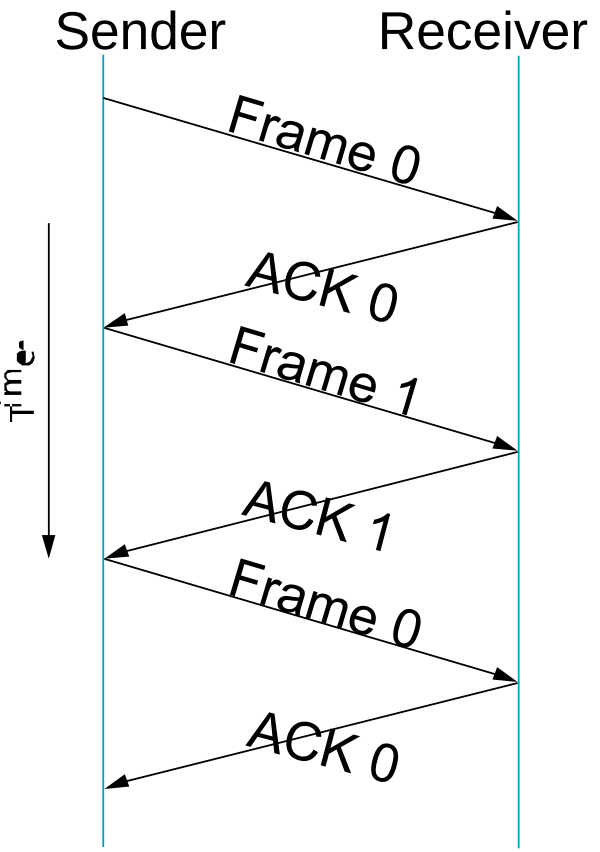
Stop-and-wait

- Problema para o receptor:



Stop-and-wait

- Uso de números de sequência:
  - °transmissor envia pacote 0;
  - °receptor envia ACK 0;
  - °se o ACK é perdido, transmissor temporiza e reenvia pacote 0;
  - •receptor sabe que acabou de receber 0, descarta o pacote (mas reenvia ACK 0)
  - °transm. recebe ACK 0, envia pacote 1;
- Apenas 0 e 1 são necessários.



Stop-and-wait

- Problema: manter a taxa de transmissão alta
  - apenas um pacote pode atravessar o canal de cada vez
  - canal fica ocioso do fim do envio até o retorno do ACK

Janela deslizante (*sliding window*)

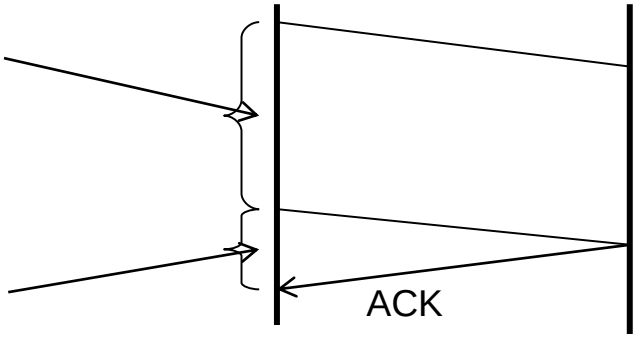
- Permite vários pacotes “em trânsito” (sem confirmação)
- Define limite para o número de pacotes nessa situação (“janela de transmissão”)
- Comportamento do receptor frente a perdas depende da sua “janela de recepção”

Janela deslizante (*sliding window*)

Desempenho: janela deslizante

Tempo de envio do pacote:
$T_p = p / Banda$

Tempo para receber o ACK:
$RTT = 2 . T_{prop}$

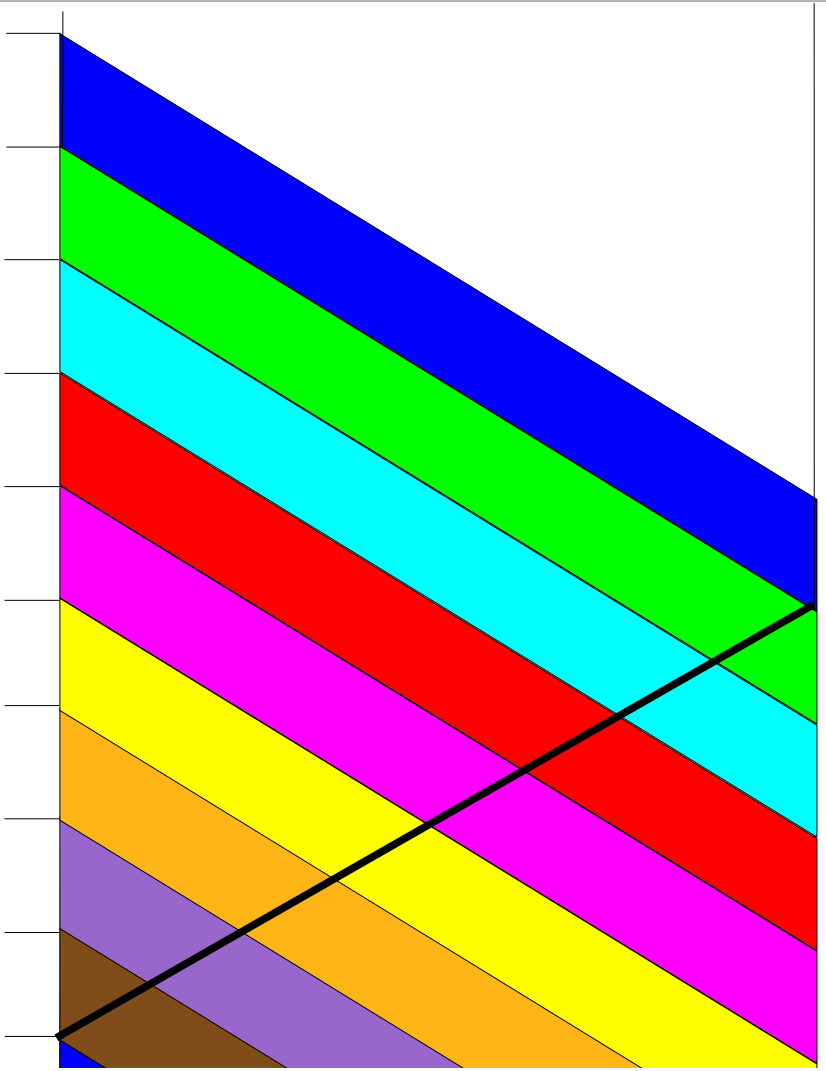


Tempo para transmitir uma janela de N pacotes: $T_w = N . T_p$

Tempo total até poder mover a janela: $T_m = T_p + RTT$

$$U = \begin{cases} T_w / T_m & \text{se } T_w < T_m \\ 1 & \text{se } T_w \varepsilon T_m \end{cases}$$

Desempenho: janela deslizante

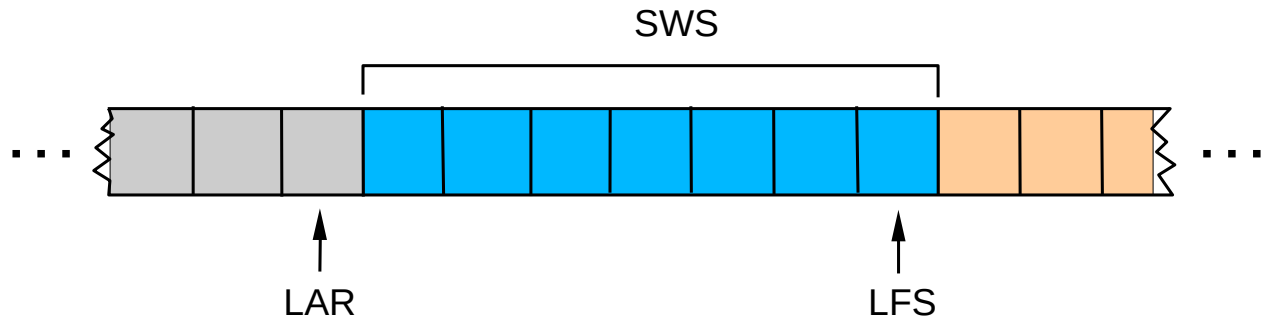


Tp = 1, RTT = 8 => Tm = 9

W	U
1	11%
2	22%
3	33%
4	44%
5	56%
6	67%
7	78%
8	89%
9	100%

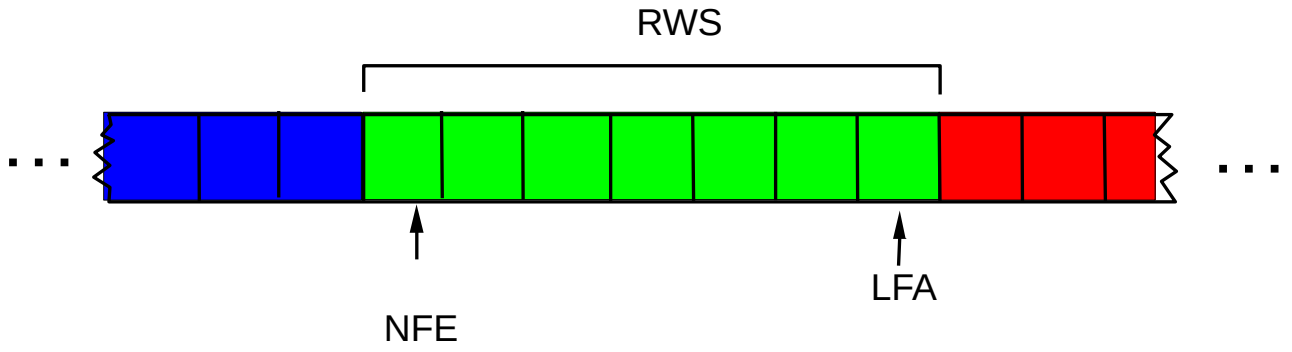
Janela deslizante: transmissor

- Determina número de sequência p/ cada quadro
- Mantém três variáveis:
  - Tamanho da janela de transmissão (SWS)
  - Última confirmação recebida (LAR)
  - Último quadro enviado (LFS)



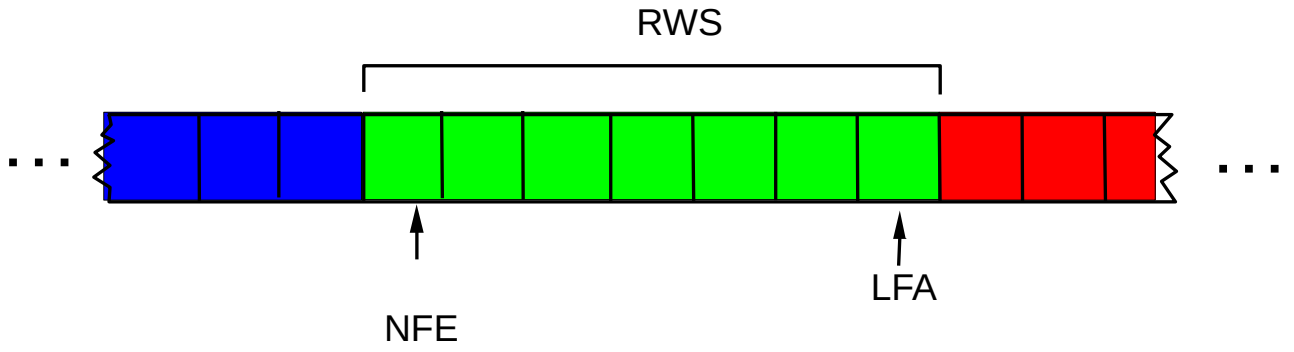
Janela deslizante: receptor

- Só aceita quadros dentro da sua janela
- Mantém três variáveis:
  - Tamanho da janela de recepção (RWS)
  - Maior quadro aceitável (LFA)
  - Próximo quadro esperado (NFE)

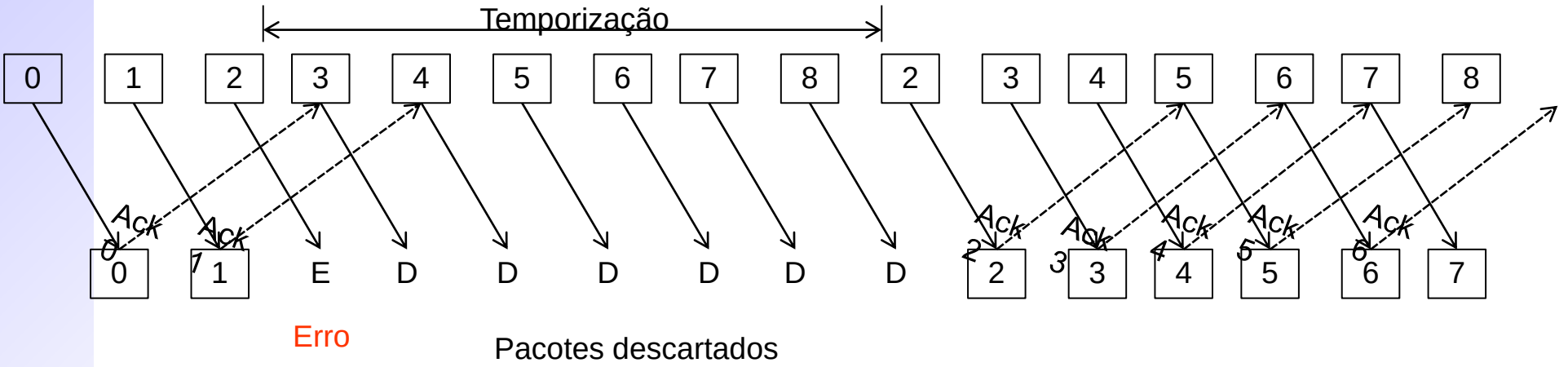


Janela deslizante: receptor

- Comportamento varia dependendo do tamanho da janela de recepção (2 casos mais comuns):
  - janela de tamanho 1 (*go-back-n*)
  - janela igual à do transmissor (armazena quadros)



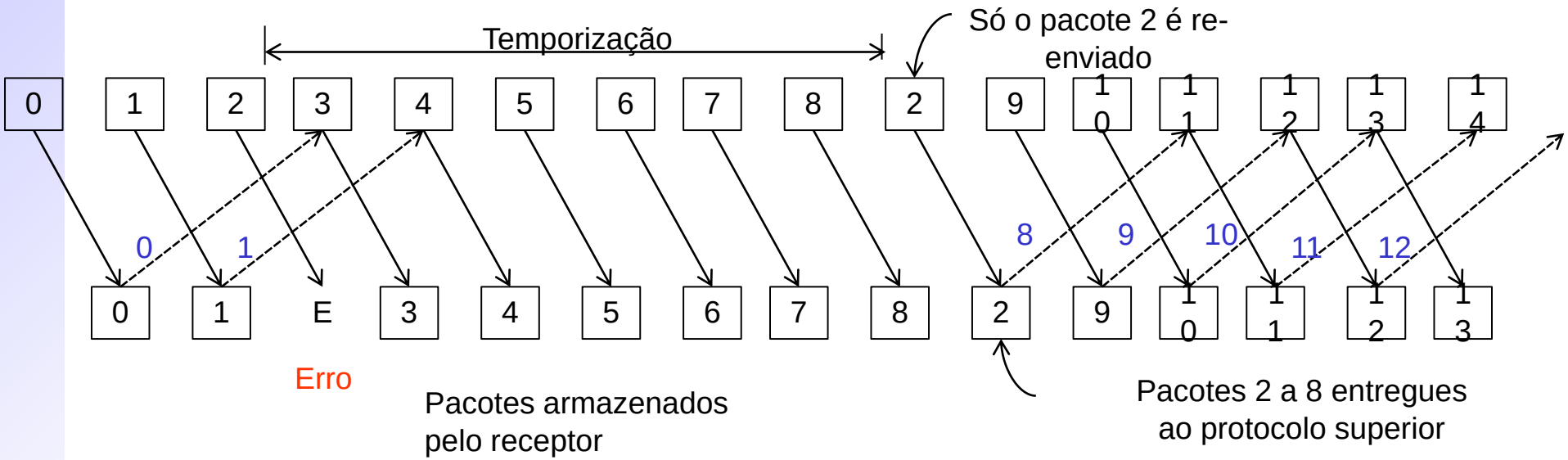
Janela deslizante (Go-back-n)



Problema:

- Muitos pacotes recebidos corretamente podem ter que ser reenviados

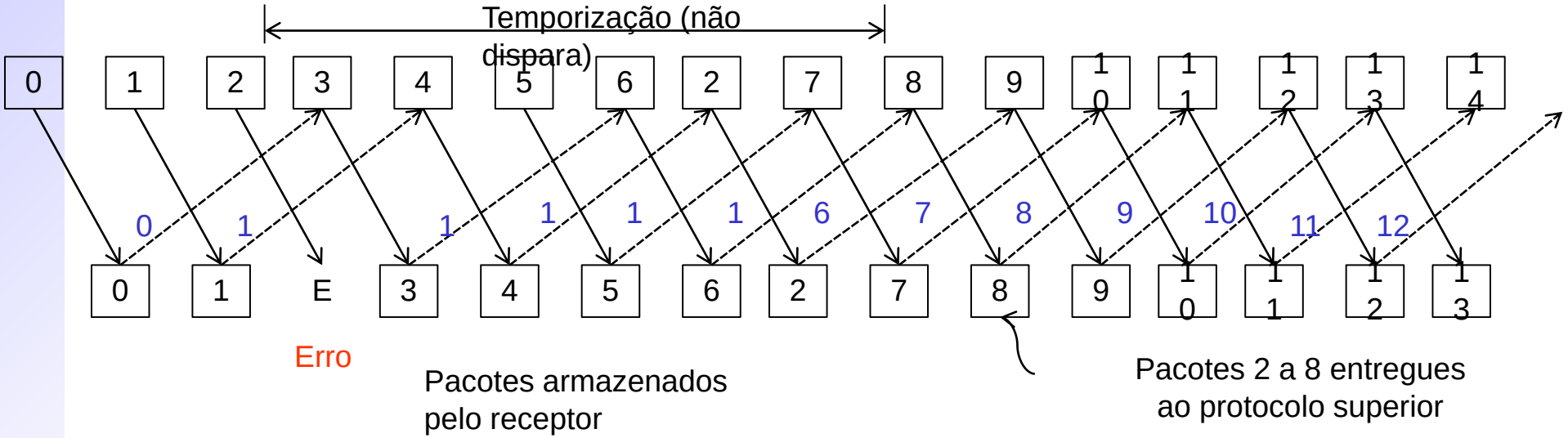
Janela deslizante (recepção seletiva)



Problema:

- Complexidade de se manter controle dos pacotes recebidos fora de ordem

Janela deslizante



Otimização (também para *go-back-n*):

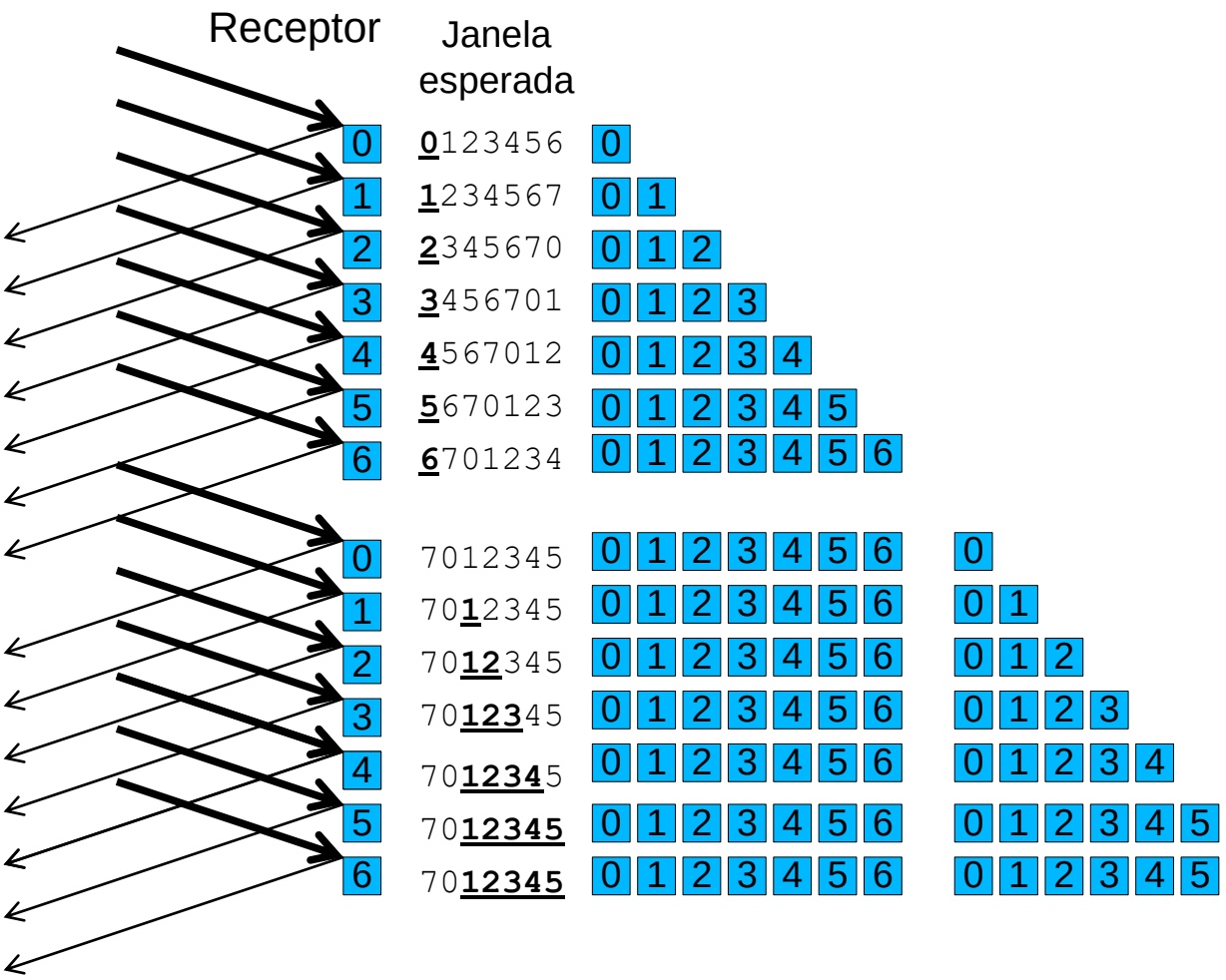
- Pacotes fora de ordem: ACKs duplicados
- ACKs duplicados: indicação de perda de um pacote
  - Retransmite-se se chegam *n* ACKs duplicados

Uso de números de sequência

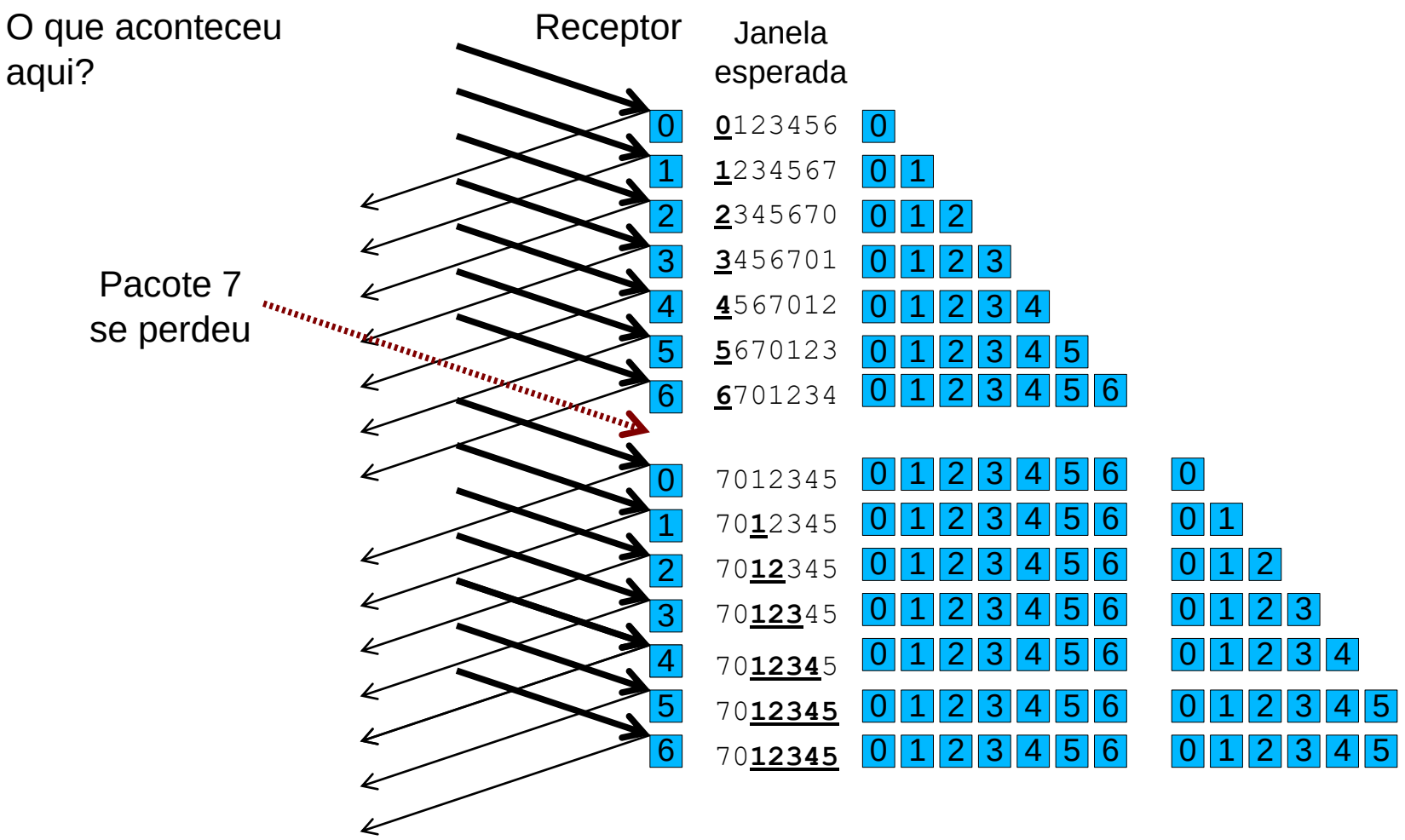
- Espaço é finito: números “dão a volta”, fila circular
  - Com 3 bits p/ SeqNum, SWS = RWS = 7 (Faixa possível - 1)

Uso de números de sequência

O que aconteceu aqui?



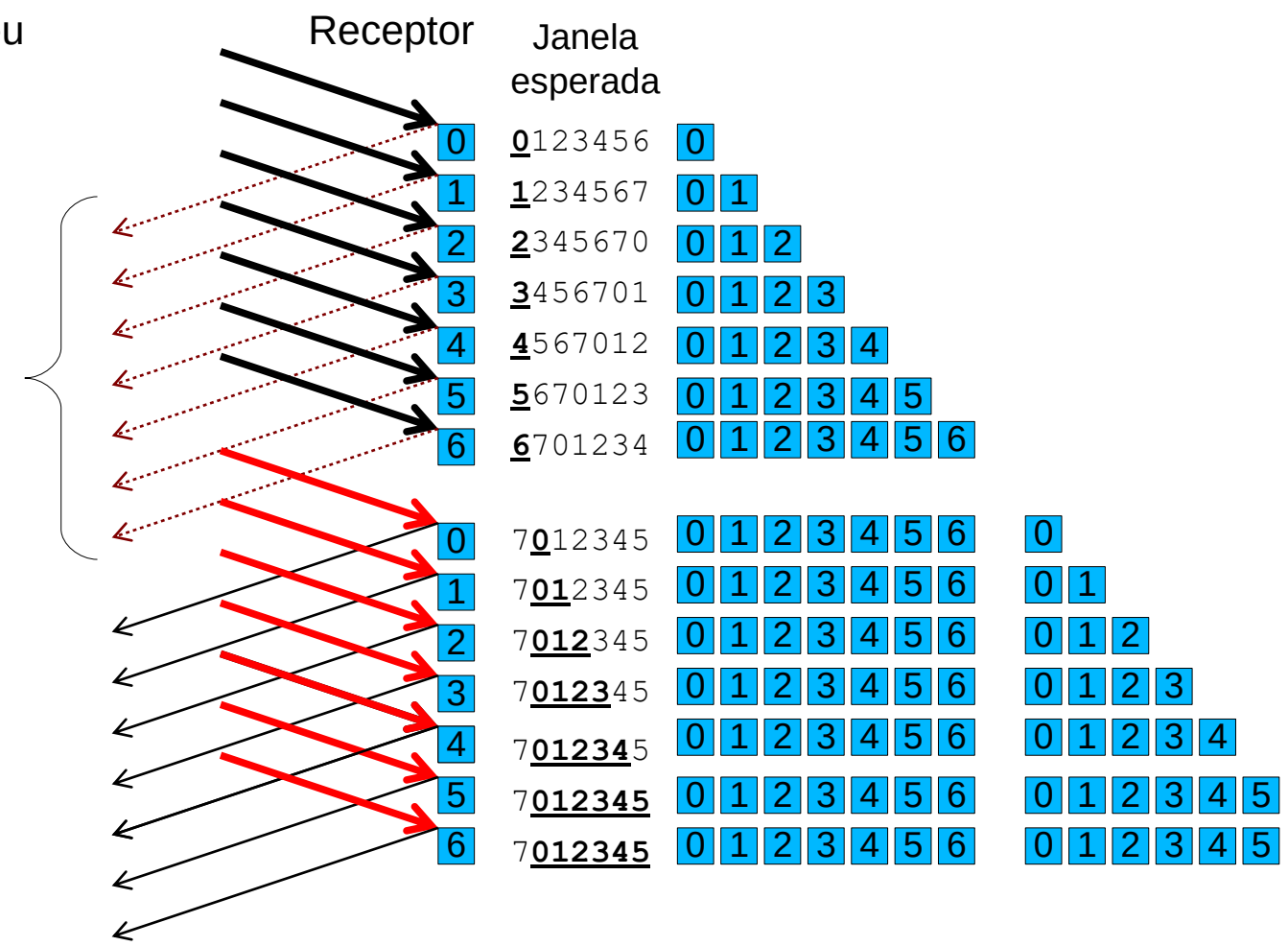
Uso de números de sequência



Uso de números de sequência

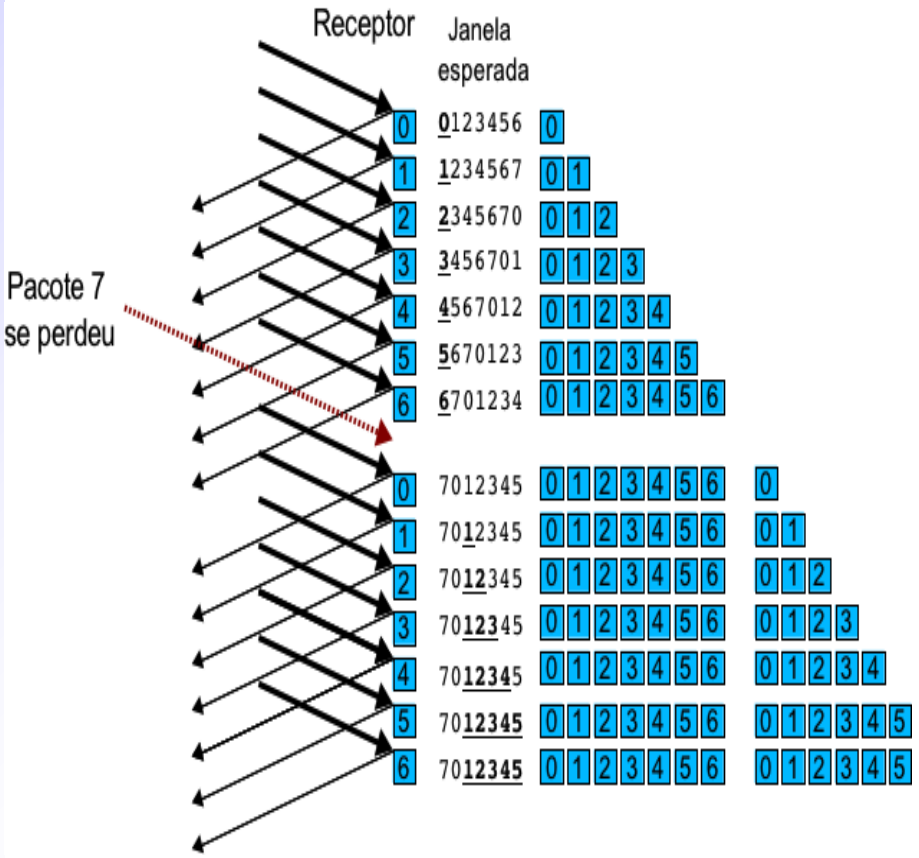
O que aconteceu aqui?

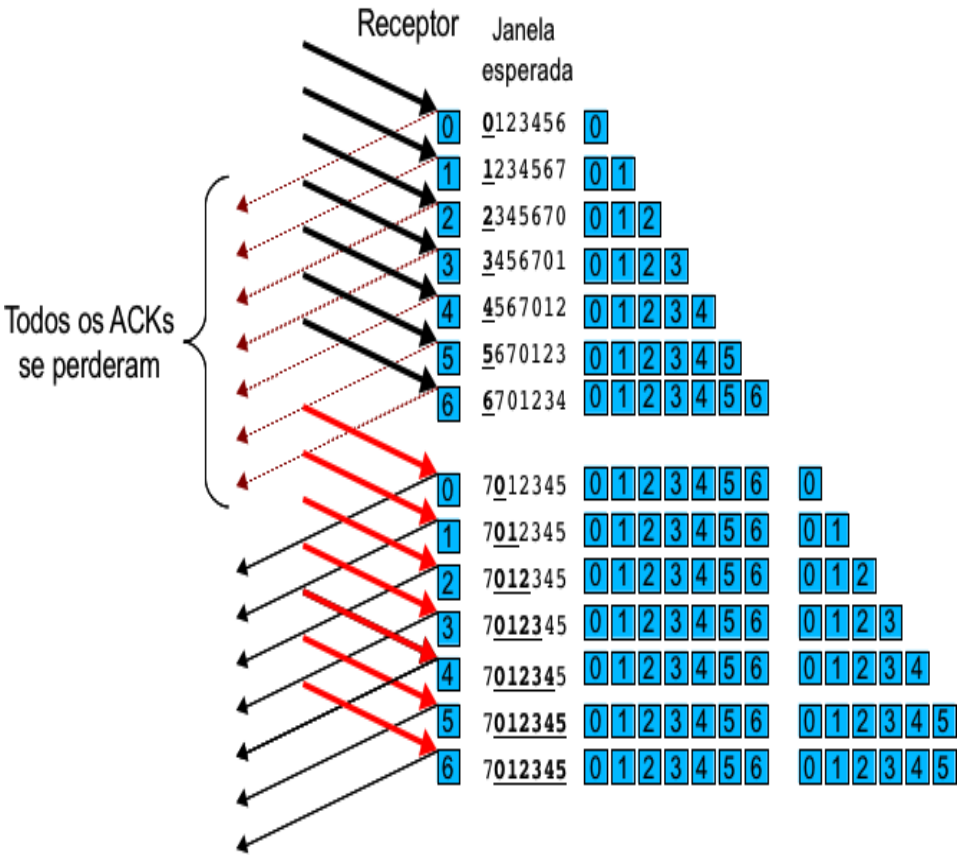
Todos os ACKs se perderam



Uso de números de sequência

Para o receptor, seria impossível diferenciar!





Solução: SWS < (MaxSeqNum+1)/2

Uso de números de sequência

- Espaço é finito: números “dão a volta”, fila circular
  - Com 3 bits p/ SeqNum, SWS = RWS = 7 (Faixa possível - 1)
  - transmissor envia quadros 0 a 6
  - todos chegam com sucesso, mas ACKs se perdem
- receptor agora espera por 7, 0, 1, 2, 3, 4, 5;
- transmissor retransmite 0 a 6
- receptor descarta 6
- mas aceita duplicatas de 0..5 como **novos pacotes**
- Correto: SWS < (MaxSeqNum+1)/2

Uso de números de sequência

- Espaço é finito: números “dão a volta”, fila circular
  - Com 3 bits p/ SeqNum, SWS = RWS = 7 (Faixa possível - 1)
  - transmissor envia quadros 0 a 6
  - todos chegam com sucesso, mas ACKs se perdem
- receptor agora espera por 7, 0, 1, 2, 3, 4, 5;
- transmissor retransmite 0 a 6
- receptor descarta 6
- mas aceita duplicatas de 0..5 como **novos pacotes**
- Correto: SWS < (MaxSeqNum+1)/2