Camada de Enlace

Controle de Fluxo

Controle de Fluxo

- Quando a taxa de transmissão é superior à taxa em que o receptor pode processar os dados recebidos
  - Processo transmissor reside em um computador mais rápido ou menos carregado que o computador onde reside o receptor

Controle de Fluxo

- Receptor pode ficar “inundado” (flood) com quadros
  - buffer overflow
  - quadros começam a ser perdidos
    - mesmo que não haja erros de transmissão

Controle de Fluxo

- Solução genérica:
  - Regras para garantir o compasso entre o transmissor e o receptor
  - Exige alguma forma de feedback do receptor para o transmissor
  - Protocolo que permite ao receptor informar quando (e o quanto de) dados está preparado para receber

Controle de Fluxo

- Exemplo:
  - Receptor informa ao transmissor que pode transmitir n quadros, após os quais deve parar até que o receptor o autorize a enviar mais quadros

Protocolos de Enlace Elementares

- Três protocolos em ordem crescente de complexidade
- Um protocolo simplex irrestrito
  - uma série de suposições (não-realistas) que simplificam o projeto do protocolo

Protocolos de Enlace Elementares

- Um protocolo simplex do tipo stop-and-wait
  - controle de fluxo básico
- Um protocolo simplex com controle de erros
  - mais realista, reconhece que o canal de comunicação é sujeito a erros

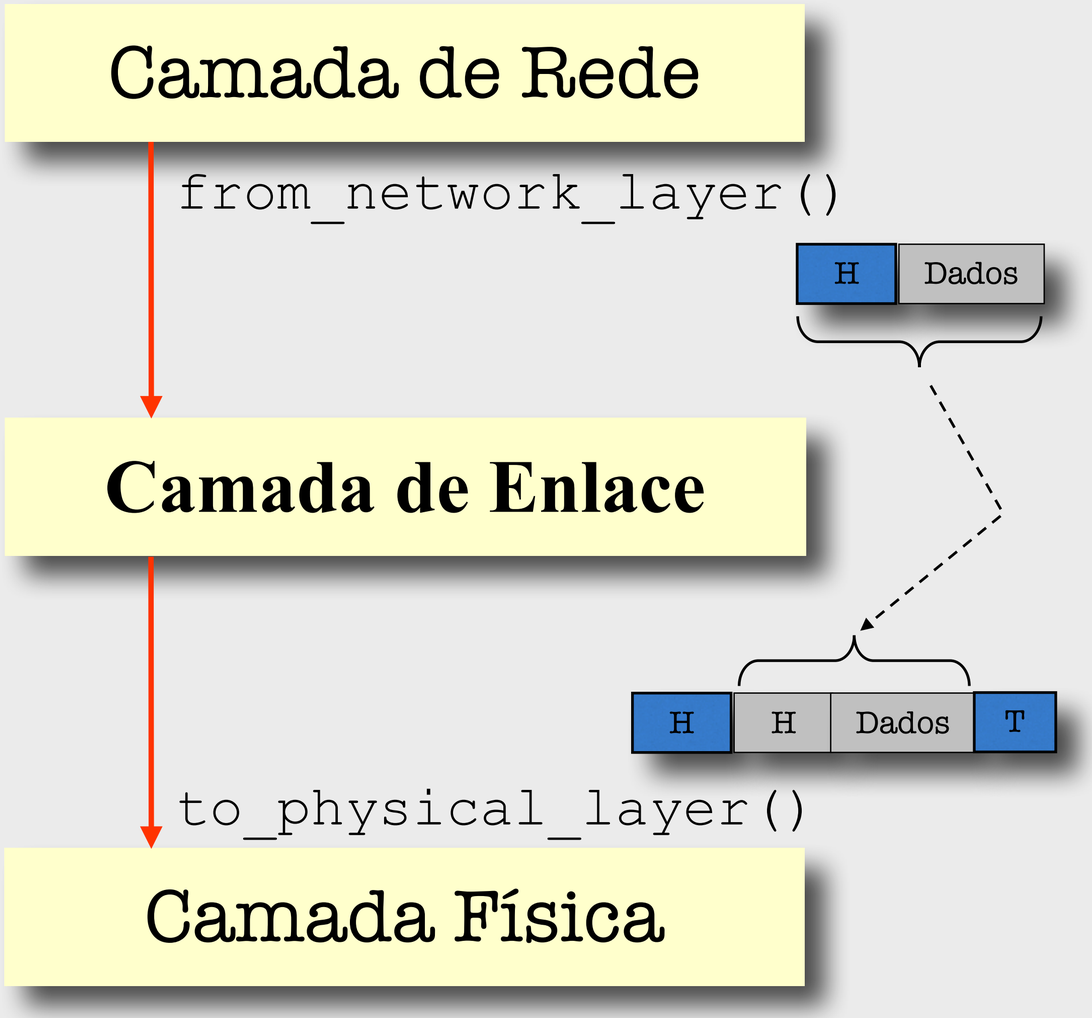
O Modelo de Comunicação

- Transmissão simplex
  - A comunicação é do computador A para o computador B apenas
  - Em protocolos mais sofisticados a comunição é duplex

O Modelo de Comunicação

- Camada de rede (em A) sempre tem dados a transmitir
  - Suprimento de dados infinito
  - Esta suposição será removida à medida em que protocolos mais sofisticados são apresentados
- Pacotes da camada de rede são tratados puramente como dados pela camada de enlace (inclusive o cabeçalho do pacote)

O Modelo de Comunicação



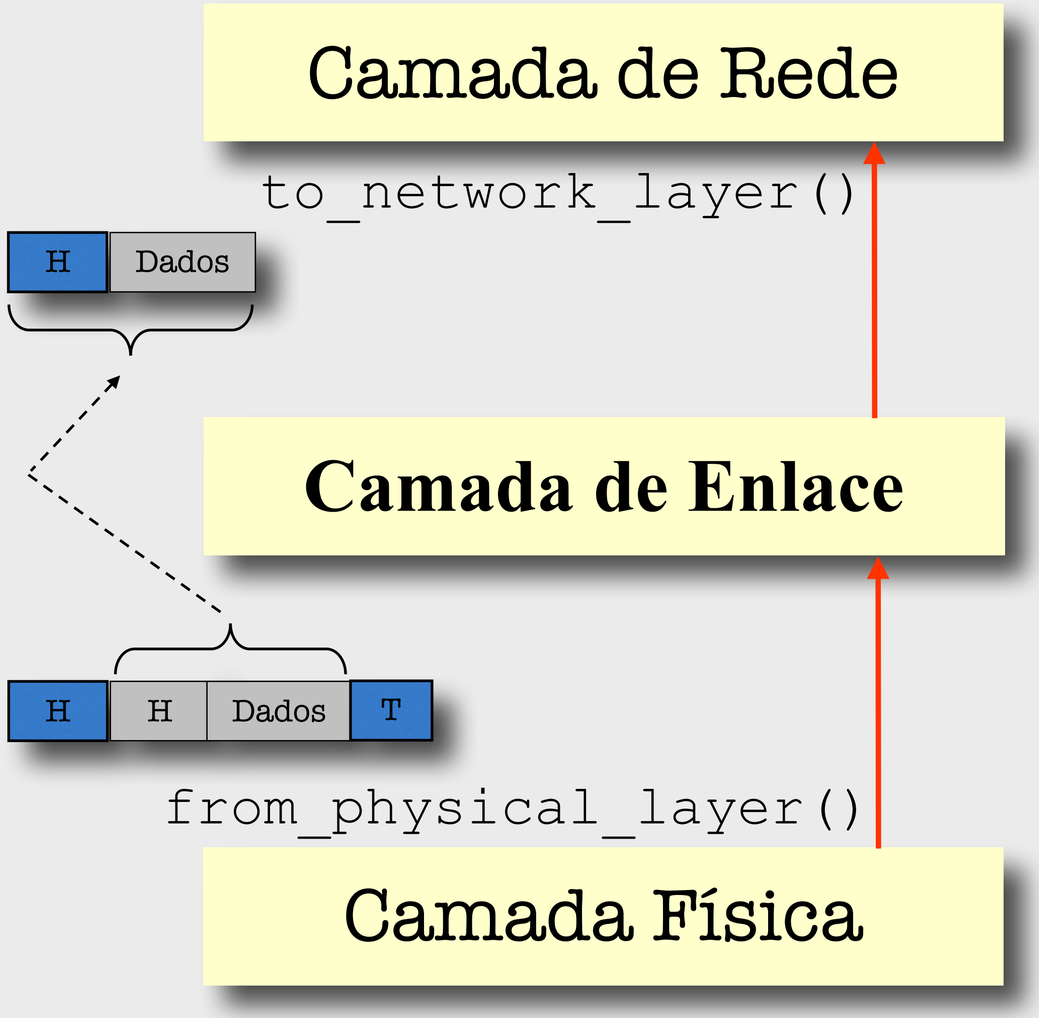
No Transmissor:

- Encapsula pacotes em quadros
- Adiciona cabeçalho e trailer
- Calcula checksum antes de transmitir o quadro

O Modelo de Comunicação

No Receptor:

- Checa cabeçalho para detectar qualquer problema
- Extrai o pacote e o repassa à camada de rede
- Verifica checksum
- Sinaliza chegada do quadro (evento)



O Modelo de Comunicação

- Loop infinito aguardando por eventos
- Procedure wait_for_event(&event)
  - Retorna quando algo acontece (ex.: chegada de um quadro)
- Vários tipos de eventos
  - Dependente de protocolo
  - Exemplos: chegada de quadro, erro de checksum, timeout, etc.

O Modelo de Comunicação

- Ao receber um evento (ex.: chegada de um quadro), a camada de enlace deve processá-lo
  - Ex.: chama from_physical_layer() para obter o quadro entrante do meio físico

Definições Básicas

- Os próximos slides apresentam um arquivo em linguagem C, protocol.h
- Esse arquivo vai ser base para os protocolos a serem estudados

Estruturas de Dados

```
/*  tamanho máximo do quadro */

#define MAX_PKT 1024

typedef enum {false, true} boolean;

/*  número de seqüência atribuído aos quadros 0 a
    MAX_SEQ (dependente de protocolo) contagem
    circular: (0, 1, 2,... MAX_SEQ, 0, 1, ...) */

typedef unsigned int seq_nr;

/*  unidade de dados trocada entre a camada de
    rede e a camada de enlace */

typedef struct {
  unsigned char data[MAX_PKT];
} packet;
```

Estruturas de Dados

```
typedef enum { /* tipo do quadro (dados ou controle) */
     data,
     ack,
     nak
} frame_kind;

typedef struct { /* quadro propriamente dito */
     frame_kind kind;
     seq_nr seq;
     seq_nr ack;
     packet info;
} frame;

/*   seq: número de seqüência do quadro
     ack: número do acknowledgement
     info: pacote encapsulado (vazio para controle) */
```

Rotinas Comuns

```
void wait_for_event(event_type &event);

/*  chamada para aceitar pacotes (da camada de rede) a
    serem transmitidos */
void from_network_layer(packet *p);

/*  chamada para passar pacotes recebidos para a camada
    de rede */
void to_network_layer(packet *p);

/*  interface com a camada física */

void to_physical_layer(frame *s);

void from_physical_layer(frame *s);
```

Rotinas Comuns

```
/*  dispara um temporizador para detectar a ocorrência de
    timeouts; um temporizador por quadro pendente */
void start_timer(seq_nr k);

/*  interrompe a contagem do temporizador quando o
    evento esperado ocorreu (ex.: quadro chegou antes do
    timeout) */
void stop_timer(seq_nr k);

/*  temporizadores com uso semelhante em situações
    especiais */
void start_ack_timer(void);

void stop_ack_timer(void);
```

Rotinas Comuns

```
/*  utilizada em protocolos mais sofisticados (com controle
    de fluxo e sem assumir fluxo de dados constante)
    quando habilitada, a camada de rede pode interromper
    a de enlace para avisar que há pacotes a serem
    transmitidos evento network_layer_ready a camada de
    enlace então invoca  from_network_layer para obter o
    pacote */
void enable_network_layer(void);

/*  desabilita a camada de rede (não permitindo novas
    interrupções para evitar que a camada de rede tente
    transmitir pacotes além da capacidade da camada de
    enlace */
void disable_network_layer(void);
```

Rotinas Comuns

```
/*  macro utilizada para incrementar números de
    seqüência circularmente

    MAX_SEQ é definido por cada protocolo */

#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Protocolo Simplex Irrestrito

- Dados são transmitidos apenas de A para B
- Camada de rede no transmissor sempre tem dados a transmitir
  - Ao ser invocada pela camada de enlace (através de from_network_layer)

Protocolo Simplex Irrestrito

- Camada de rede no receptor sempre está pronta para receber dados
  - Ao ser invocada pela camada de enlace (através de to_network_layer)
  - Tempos de processamento (nas camadas) é ignorado

Protocolo Simplex Irrestrito

- Buffers com capacidade infinita nas camadas
- Canal de comunicação 100% confiável
  - Os quadros nunca são corrompidos ou perdidos

Protocolo Simplex Irrestrito

- Suposições não-realistas, mas que simplificam a implementação deste primeiro protocolo estudado
  - Números de seqüência ou reconhecimentos (ACKs) não são necessários
  - único evento possível: chegada de quadro (sem erros)
  - apenas um tipo de pacote: de dados

Transmissor

```
#include “protocol.h”

void sender1(void) {
    frame s;
    packet buffer;


    while(true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
    }
}
```

Receptor

```
void receiver1(void) {
    frame r;
    event_type event;

    while(true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
    }
}
```

Protocolo Simplex

- Removendo a seguinte restrição:
  - receptor com capacidade infinita de processamento/ armazenamento de quadros
- Ainda assumindo um canal livre de erros e tráfego de dados em uma só direção

Protocolo Simplex

- Problema a ser resolvido:
  - prevenir que o transmissor inunde o receptor com uma taxa dados maior do que ele é capaz de consumir
- Solução: feedback do receptor para o transmissor indicando quando se pode transmitir mais quadros

Transmissor

```
#include “protocol.h”

void sender2(void) {

    frame s;
    packet buffer;
    event_type event;

    while(true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
        wait_for_event(&event);
    }
}
```

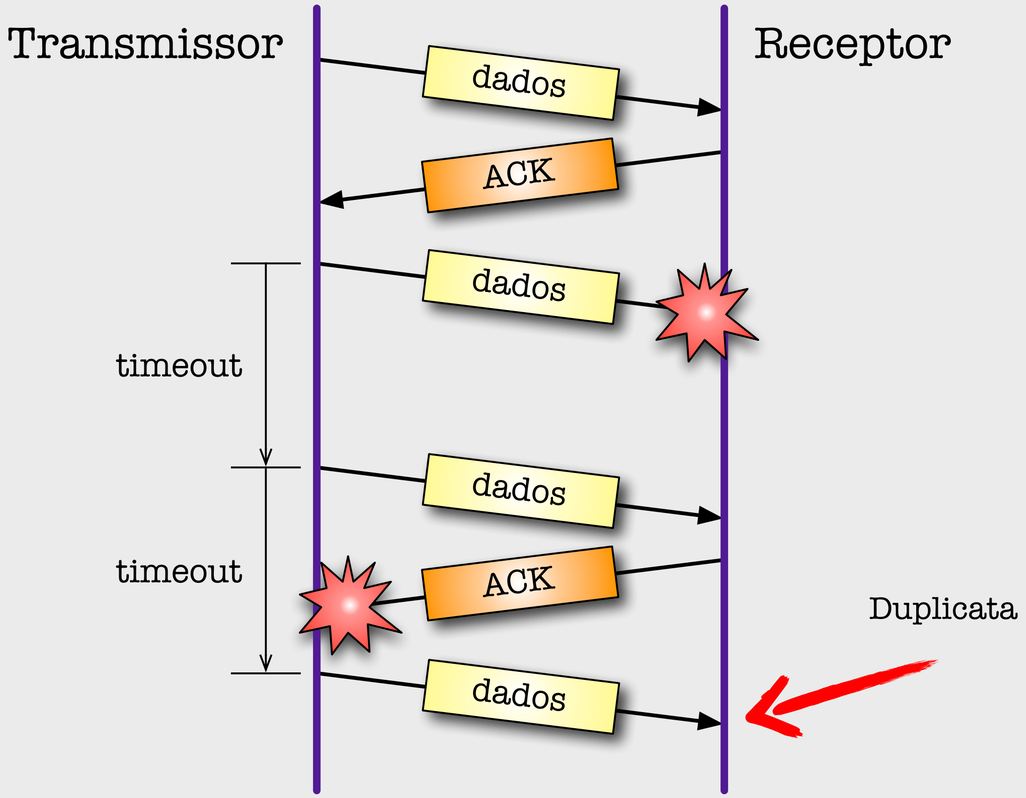
Receptor

```
void receiver2(void) {

    frame r, s;
    event_type event;

    while(true){
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```

Protocolo Simplex em Canal Sujeito a Erros

- Quadros podem ser danificados ou perdidos
  - Quadros danificados: detectados pelo hardware pelo checksum
  - Quadros perdidos: excede-se o tempo para receber o Acknowledgement
- Solução simplista:
  - Uso de timeout no protocolo anterior, não funciona

Protocolo Simplex em Canal Sujeito a Erros



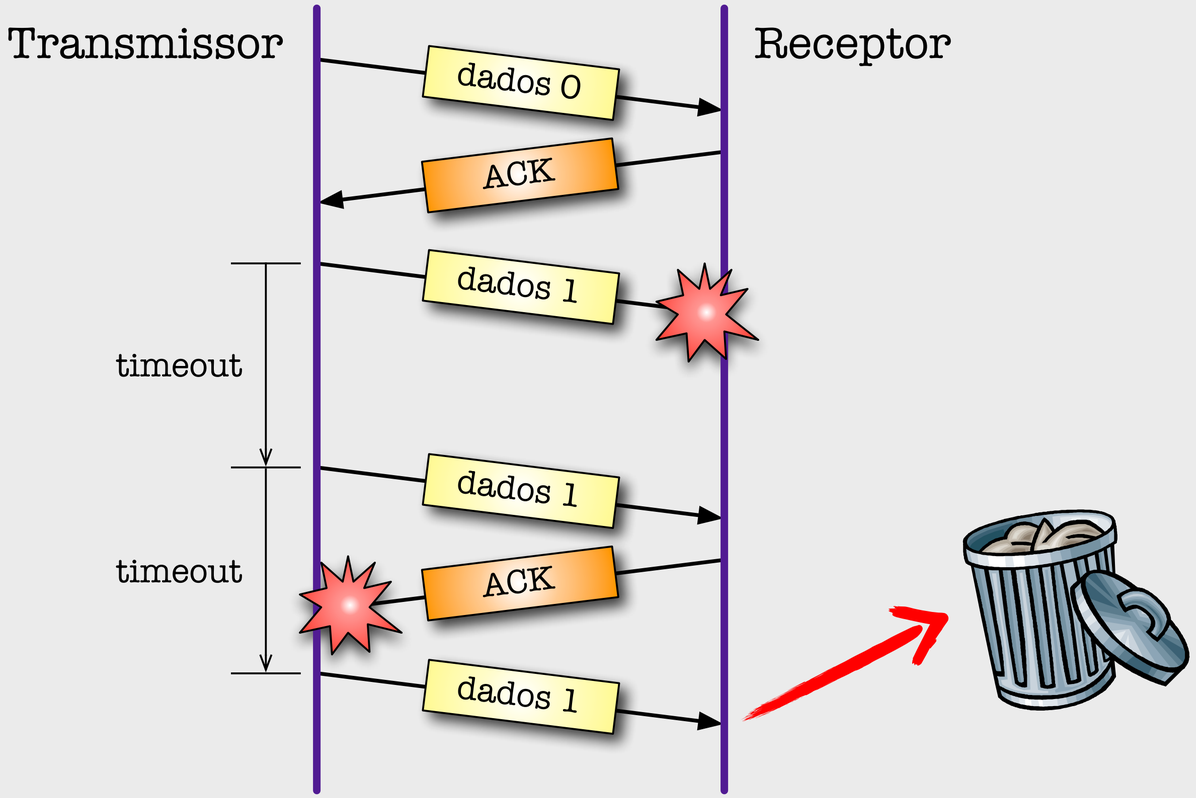
Protocolo Simplex em Canal Sujeito a Erros

- Solução mais elaborada:
  - Uso de números de seqüência no cabeçalho de cada quadro de dados
    - Stop-and-wait com timeout e números de seqüencia
    - Receptor pode distingüir quadros novos de retransmissões
    - Acknowledgements: quadros vazios (ver posteriormente)

Protocolo Simplex em Canal Sujeito a Erros

- Detalhe: Tamanho dos números de seqüência
  - Influencia no overhead carregado em cada quadro de dados
  - É necessário ao receptor distinguir apenas entre o quadro e atual e o próximo (no stop-and-wait há apenas um quadro a cada instante)
  - Portanto: 1 bit apenas neste caso:
    0, 1, 0, 1, ...

Protocolo Simplex em Canal Sujeito a Erros



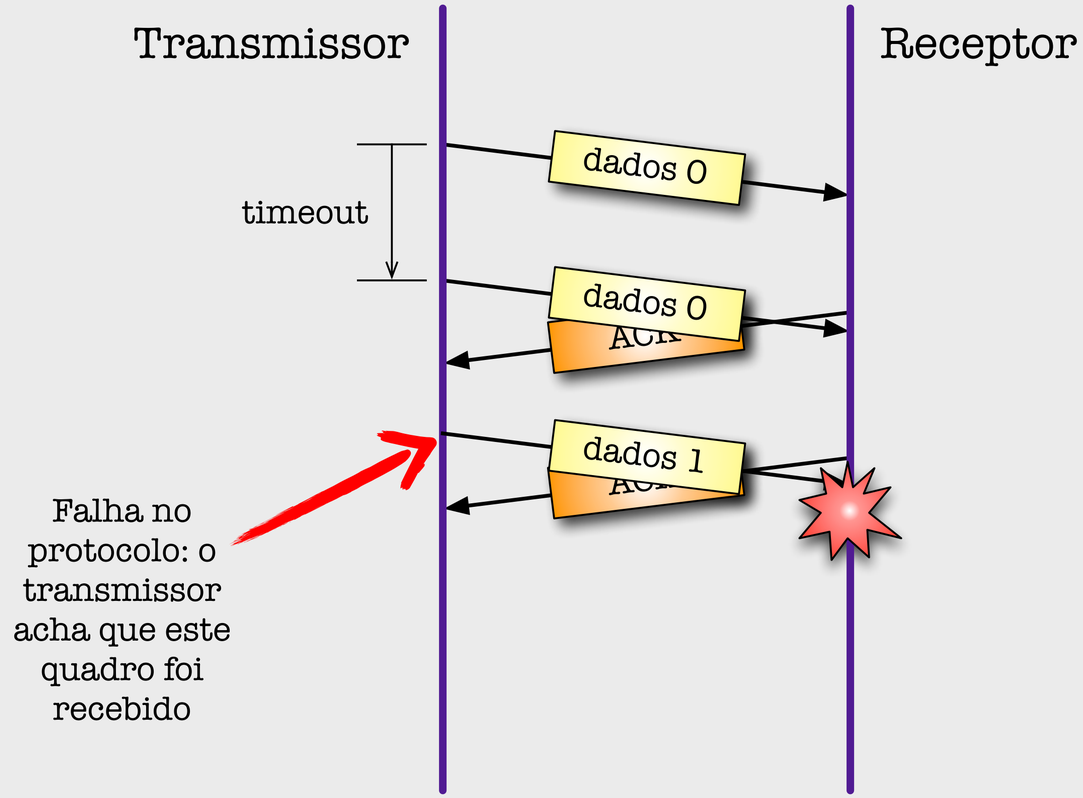
Protocolo Simplex em Canal Sujeito a Erros

- Protocolos deste tipo são conhecidos como:
  - ARQ – Automatic Repeat Request, ou
  - PAR – Positive Aknowledgement with Retransmission

Protocolo Simplex em Canal Sujeito a Erros

- Duração do intervalo de timeout deve ser corretamente ajustada, suficiente para:
  - Propagação do quadro até o receptor
  - Processamento do quadro e geração do ACK no receptor
  - Propagação do ACK (quadro de controle) até o transmissor

Timeout Subestimado



Protocolo ARQ Simplex: Definições

```
#define MAX_SEQ 1


typedef enum {
    frame_arrival,
    cksum_error,
    timeout
} event_type;


#include “protocol.h”
```

Protocolo ARQ Simplex Transmissor

```
void sender3(void) {

    seq_nr next_frame_to_send;
    frame s;
    packet buffer;
    event_type event;

    next_frame_to_send = 0;
    from_network_layer(&buffer);
```

Protocolo ARQ Simplex Transmissor

```
    while(true) {
        s.info = buffer;
        s.seq = next_frame_to_send;
        to_physical_layer(&s);
        start_timer(s.seq);
        wait_for_event(&event);

        if (event == frame_arrival) {  /* chegou ACK */
            from_network_layer(&buffer);
            inc(next_frame_to_send);
        }
    }
}
```

Protocolo ARQ Simplex Receptor

```
void receiver3(void) {

    seq_nr frame_expected;
    frame r, s;
    event_type event;

    frame_expected = 0;
```

Protocolo ARQ Simplex Receptor

```
    while(true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            to_physical_layer(&s); /* envia ACK */
        }
    }
}
```

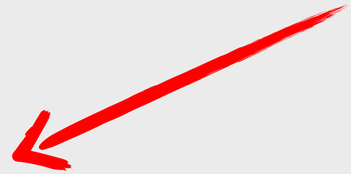
Transmissor com Número de Seqüência

```
void sender3a(void) {

    seq_nr next_frame_to_send;
    frame s;
    packet buffer;
    event_type event;

    next_frame_to_send = 0;
    from_network_layer(&buffer);
```

Transmissor com Número de Seqüência

```
while(true) {
    s.info = buffer;
    s.seq = next_frame_to_send;
    to_physical_layer(&s);
    start_timer(s.seq);
    wait_for_event(&event);
    if (event == frame_arrival) {
        from_physical_layer(&s);
        if (s.ack == next_frame_to_send) {
            from_network_layer(&buffer);
            inc(next_frame_to_send);
        }
    }
}
```

ACK com número de seqüência



Receptor com Número de Seqüência

```
void receiver3a(void) {

    seq_nr frame_expected;
    frame r, s;
    event_type event;

    frame_expected = 0;
```

Receptor com Número de Seqüência

```
while(true) {
    wait_for_event(&event);

    if(event == frame_arrival) {
        from_physical_layer(&r);

        if (r.seq == frame_expected) {
            to_network_layer(&r.info);
            inc(frame_expected);
        }

        s.ack = 1 - frame_expected;
        to_physical_layer(&s);
    }
}
}
```

ACK com número de seqüência

Piggybacking

- Problema: Reconhecimentos (ACKs) consomem recursos da rede
  - Um quadro transmitido para cada ACK
  - Tráfego desnecessário de quadros somente com confirmação de recebimento

Piggybacking

- Solução: Enviar reconhecimentos de “carona” em quadros de dados transmitidos no sentido oposto ao do quadro reconhecido
  - Aguarda-se até que haja um quadro de dados a ser transmitido para então enviar o ACK
  - Caso demore muito, enviar o ACK em quadro separado – para evitar timeout do transmissor

Eficiência de Utilização

- Stop-and-wait apresenta sérios problemas de eficiência de utilização da capacidade do enlace
- Stop-and-wait é inapropriado quando temos:
  - RTT muito alto
  - Alta largura de banda
  - Quadros de tamanho pequeno

Eficiência de Utilização
– Exemplo –

- Enlace de satélite usando protocolo 4
  - Taxa: 50 Kbps; RTT = 500 ms
  - Tempo de transmissão: 20 ms
  - Recepção do quadro: 270 ms
  - Recepção do ACK: 520 ms
  - Transmissor fica bloqueado 500/520 = 96% do tempo, utilizando 4% da capacidade disponível

Envio de Múltiplos Quadros sem Bloqueio

- Permite-se ao transmissor enviar até w quadros antes que o primeiro reconhecimento seja recebido
  - w calculado em função do RTT
  - Preenchendo o máximo da capacidade do enlace (lembrança: produto atraso × largura de banda)

Envio de Múltiplos Quadros sem Bloqueio

- No exemplo anterior:
  - w = 26 (RTT=520 dividido pelo tempo de transmissão = 20)
  - Após enviar o 26º quadro, ACKs chegarão a cada 20 ms, dando permissão para transmitir mais um quadro

Protocolos de Janela Deslizante

- Ou Sliding Window
- Removendo mais uma das suposições:
  - Protocolos full-duplex
    - Um circuito físico full-duplex ou dois circuitos simplex
- Mantém-se a suposição de que a camada de rede sempre tem pacotes a transmitir

Conceito de Janela Deslizante

- Cada quadro (e cada reconhecimento) contém um número de seqüência
- Janela de transmissão
  - Números de seqüência dos quadros que podem ser transmitidos
    - Ex.: 0 1 2 3 4 5 6 7 9 10 11
  - Quadros transmitidos mas com ACK pendente

Conceito de Janela Deslizante

- Janela de recepção
  - Números de seqüência dos quadros que o receptor pode aceitar
    - Ex.: 0 1 2 3 4 5 6 7 8 9 10 11
- As duas janelas são atualizadas (deslizadas) a cada quadro transmitido / recebido

Conceito de Janela Deslizante

- Janela de transmissão
  - Quadro transmitido: incrementa limite superior
  - ACK recebido: incrementa limite inferior
    - Quadros são mantidos em buffer até receber ACK

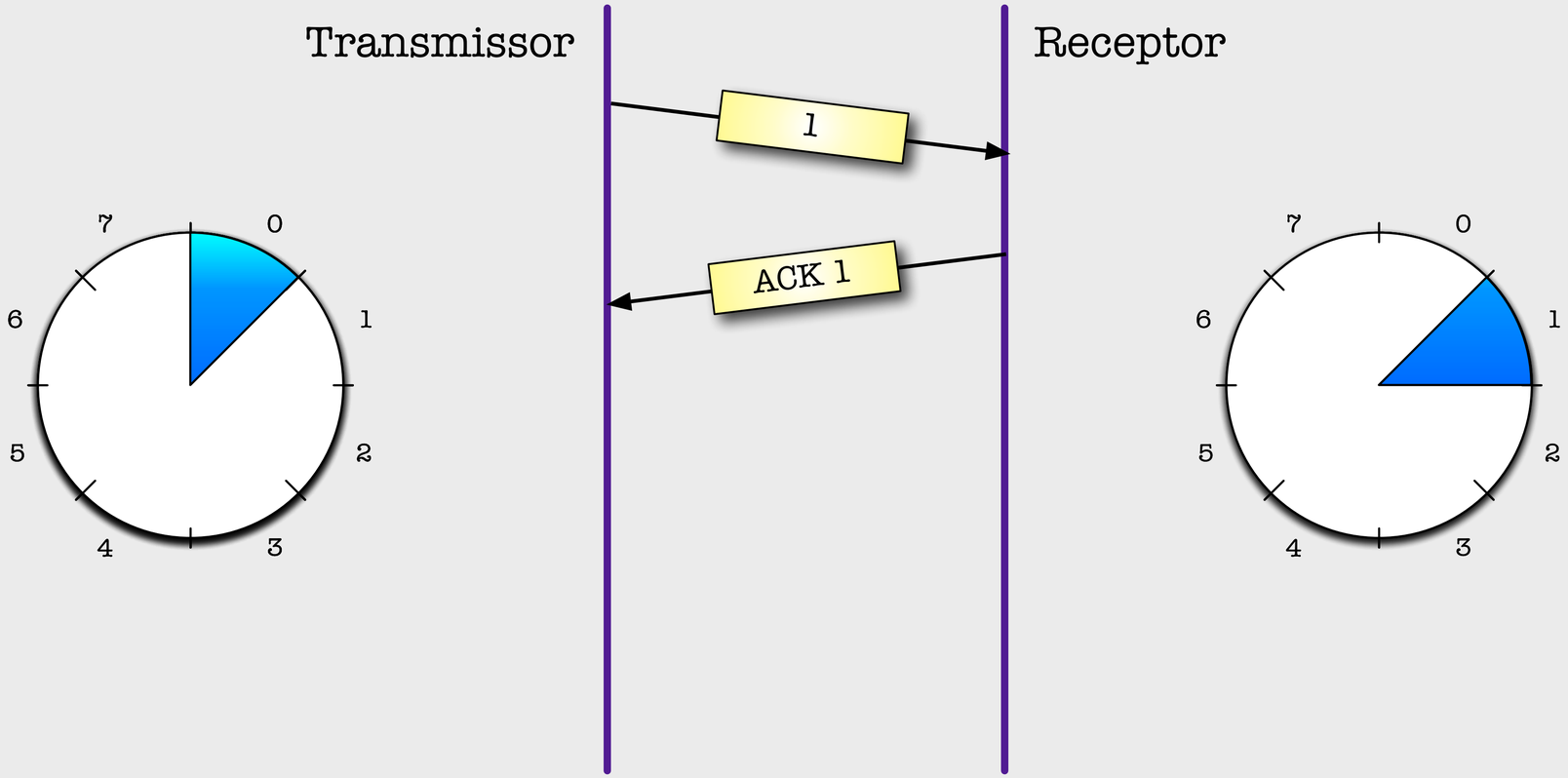
Conceito de Janela Deslizante

- Janela de recepção
  - Quadro recebido com número de seqüência dentro da janela:
    - Quadro é aceito, ACK é enviado, janela é deslizada, somente é passado para a camada de rede quando número for igual ao primeiro número de seqüência na janela

Conceito de Janela Deslizante

- Janela de recepção
  - Quadro recebido com número de seqüência fora da janela:
    - Quadro é simplesmente descartado

Janela Deslizante de Tamanho 1 bit



Janela Deslizante de Tamanho 1 bit

```
#define MAX_SEQ 1                       /* 1 para protocolo 4 */

typedef enum {
    frame_arrival,
    cksum_err,
    timeout
} event_type;

#include ”protocol.h“

void protocol4(void) {

    seq_nr next_frame_to_send;       /* 0 ou 1 apenas */
    seq_nr frame_expected;           /* 0 ou 1 apenas */
    frame   r, s;
    packet  buffer;                  /* pacote sendo enviado */
    event_type event;
```

Janela Deslizante de Tamanho 1 bit

```
next_frame_to_send = 0;    /* próximo quadro saída     */
frame_expected = 0;        /* num. do quadro entrada */

/* busca pacote na camada de rede */
from_network_layer(&buffer);

s.info = buffer;    /* prepara quadro inicial para envio */
s.seq = next_ frame_to_send;        /* insere seqüência */
s.ack = 1 – frame_expected;       /* ACK em piggyback */

to_physical_layer(&s);             /* transmite o quadro */

/* inicia os temporizadores para timeout */
start_timer(s.seq);
```

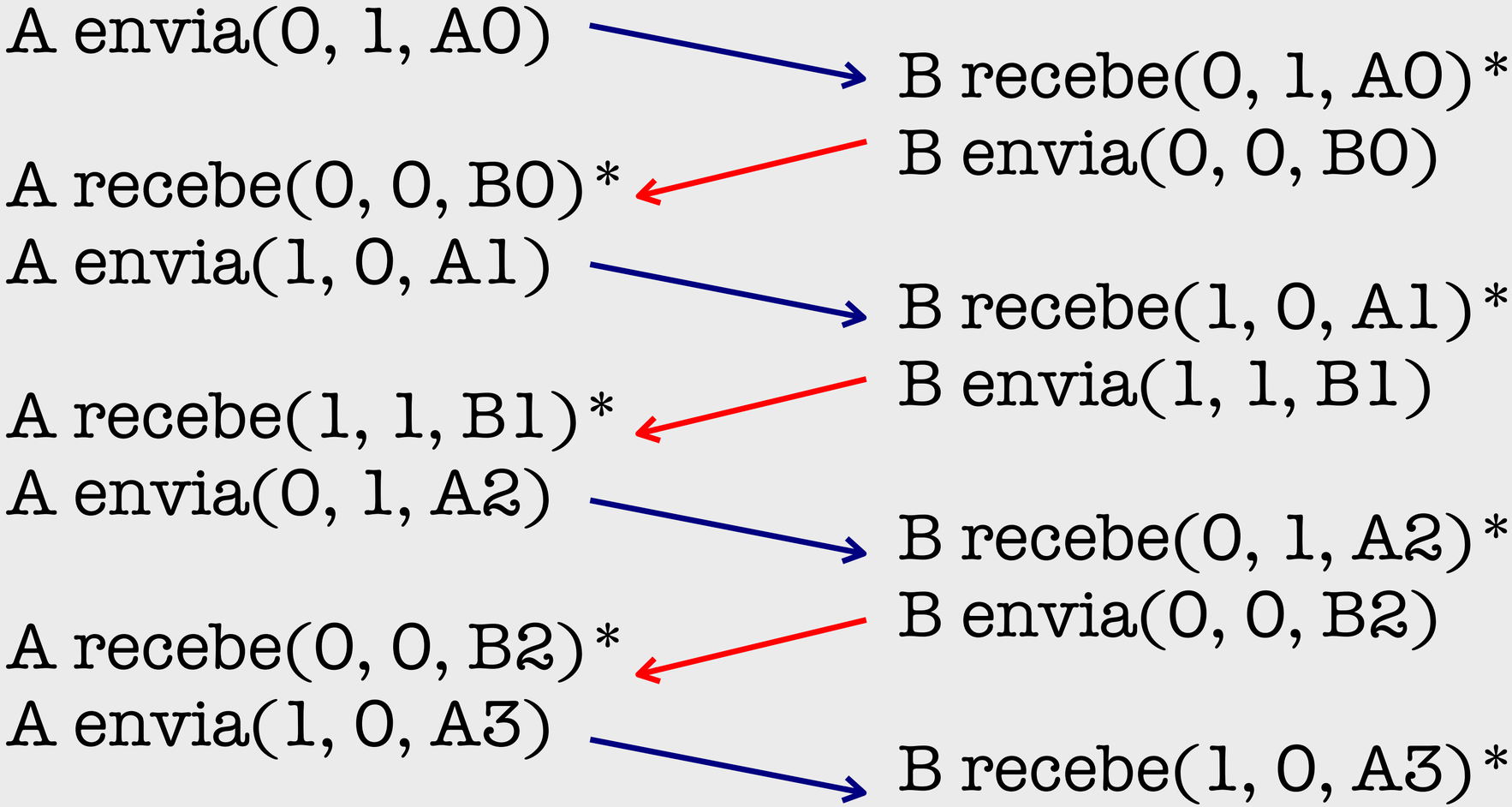
Janela Deslizante de Tamanho 1 bit

```
while (true) {
    wait_for_event(&event);
    /* frame_arrival, cksum_err, or timeout */

    if (event == frame_arrival) {
        from_physical_layer(&r); /* pega quadro */
        if (r.seq == frame_expected) {
            to_network_layer(&r.info);
            inc(frame_expected);       /* inverte 0, 1 */
        }

        if (r.ack == next_frame_to_send) {
            from_network_layer(&buffer);   /* pacote */
            inc(next_frame_to_send); /* inverte 0, 1 */
        }
    }
```

Janela Deslizante de Tamanho 1 bit

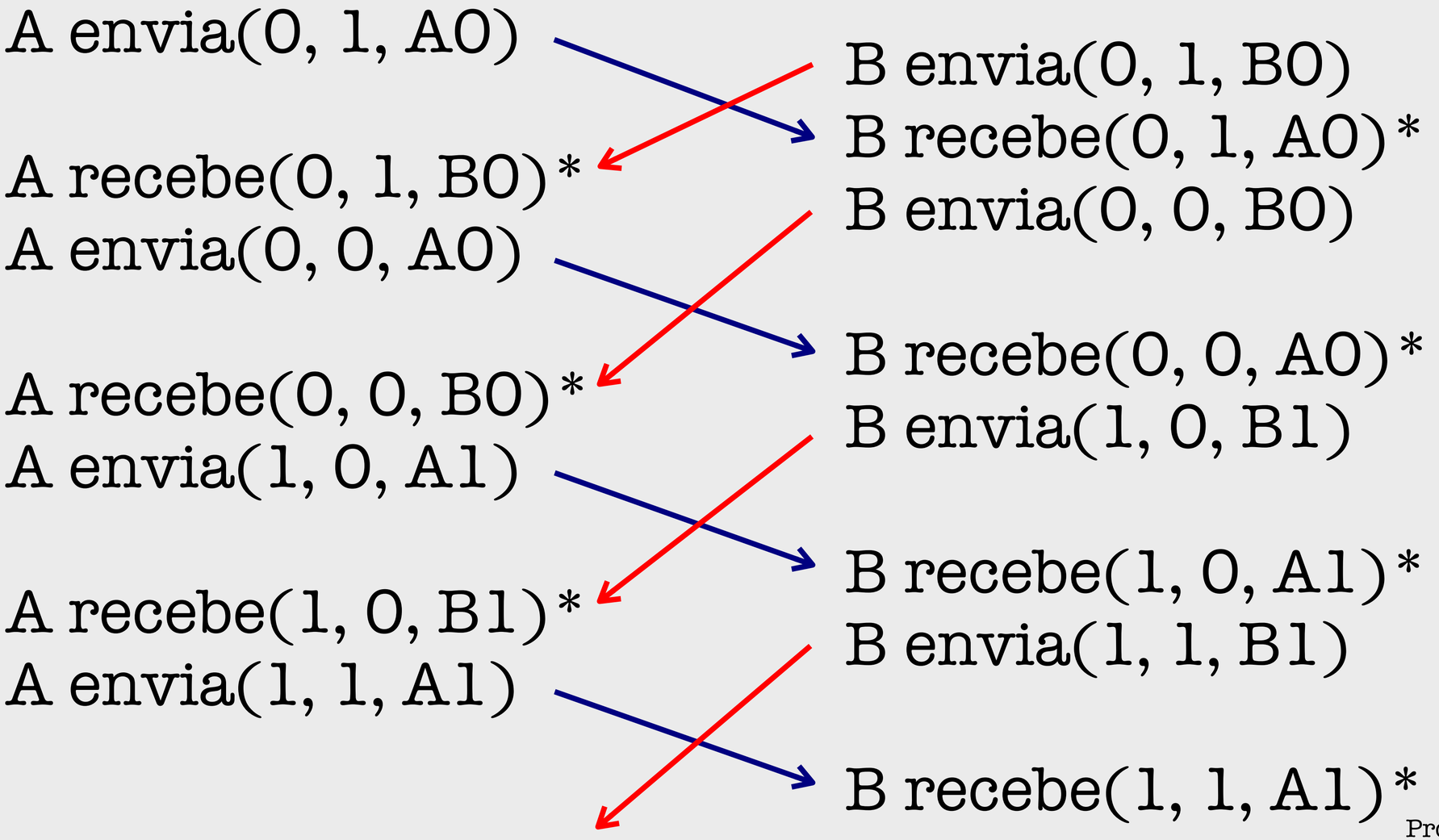
```
/*  envia novo quadro, no caso de:

    r.ack == next_frame_to_send

    ou repete o quadro, para eventos timeout e cksum_err */
        s.info = buffer;                /* monta quadro de saída */

        /* insere número de seqüência no quadro */
        s.seq = next_frame_to_send;

        /* seqüência do último quadro recebido */
        s.ack = 1 - frame_expected;
        to_physical_layer(&s);      /* transmite o quadro */
        start_timer(s.seq);         /* inicia temporizador */
    }
}
```

Cenário Ideal



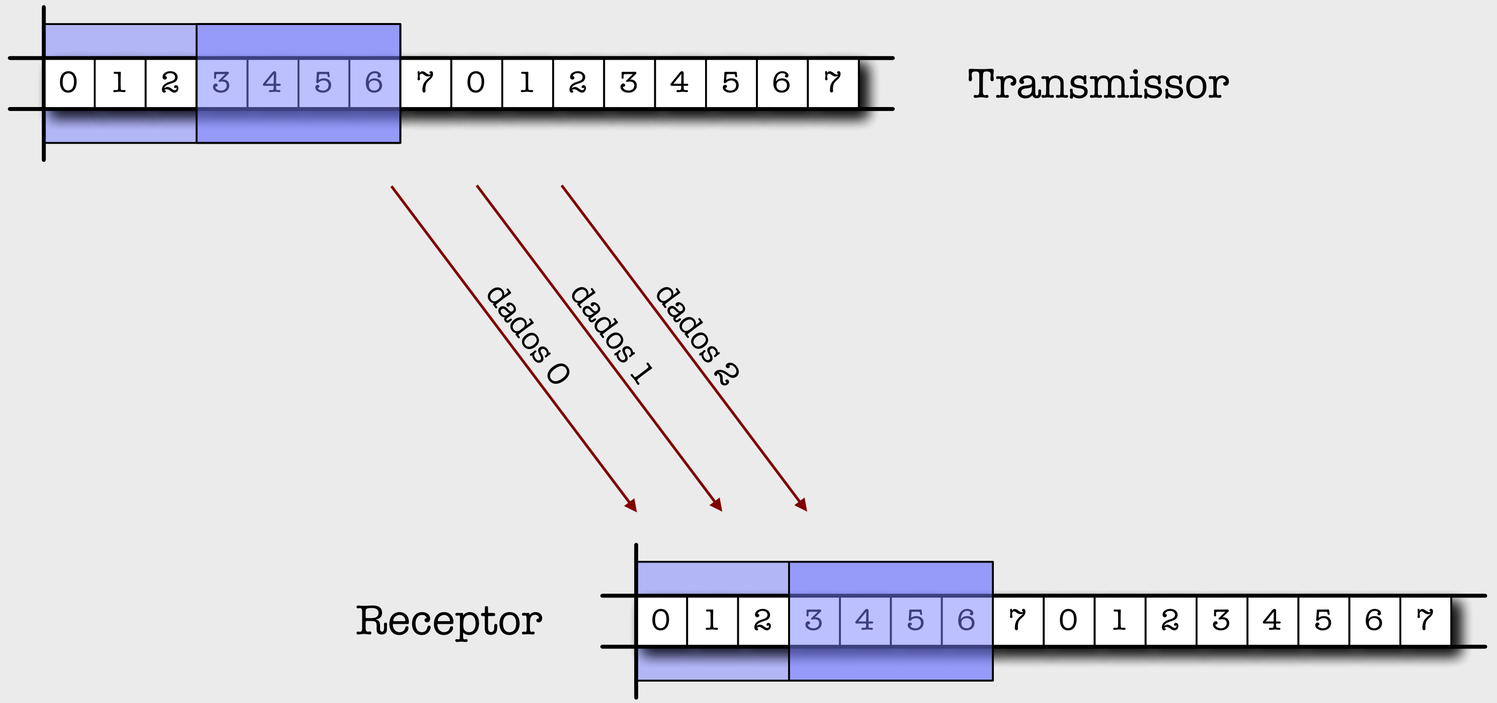
Notação:
(Seq, Ack, Número Pacote)

Cenário Anômalo

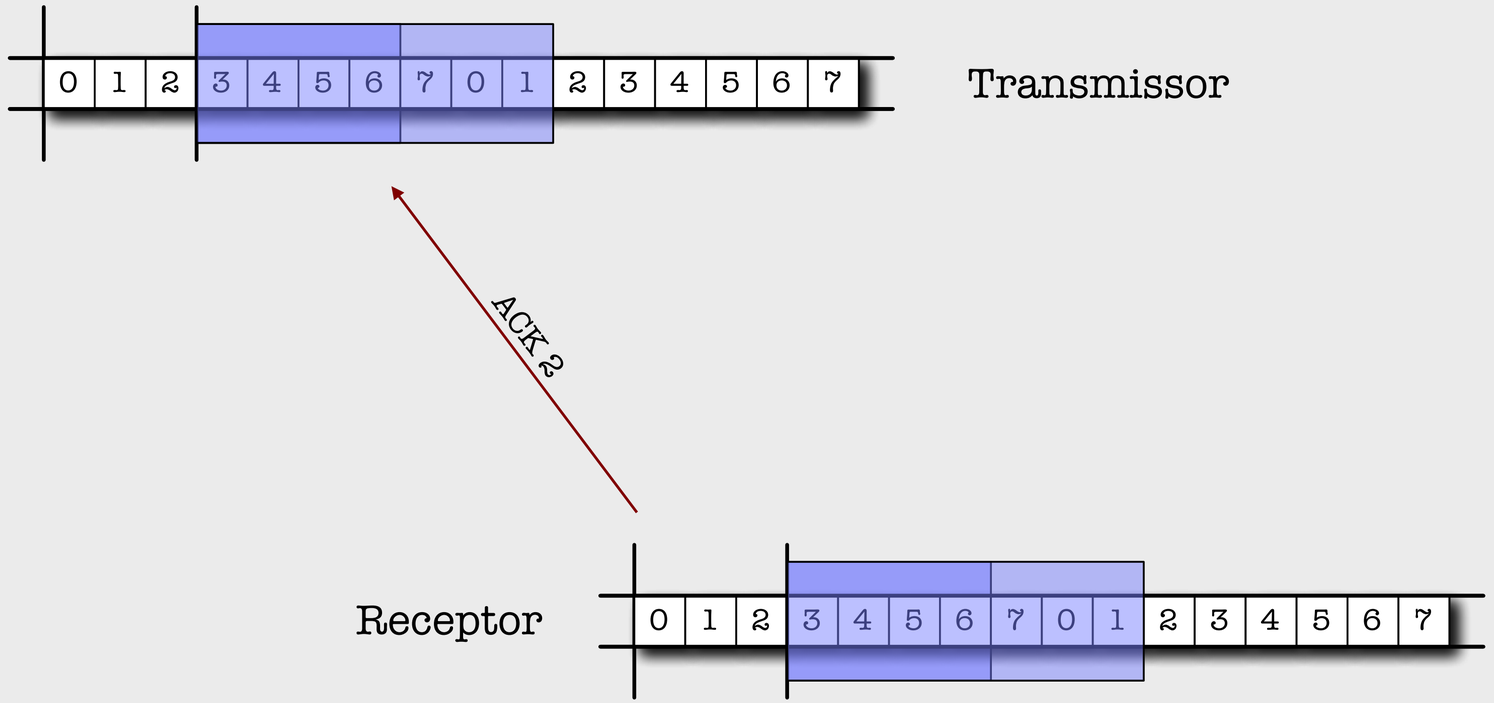


Protocolo não falha, mas transmite o dobro dos quadros

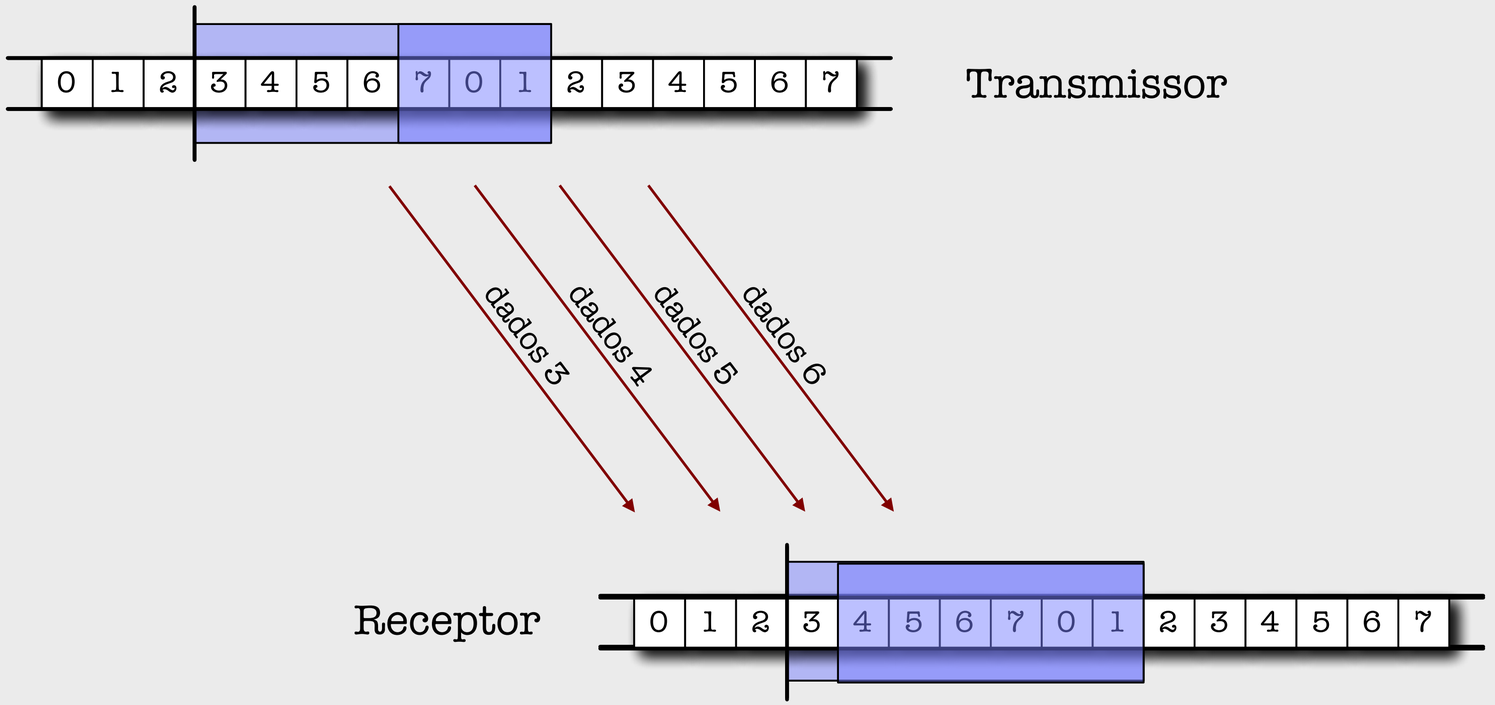
Janela Deslizante de Tamanho 3 bit



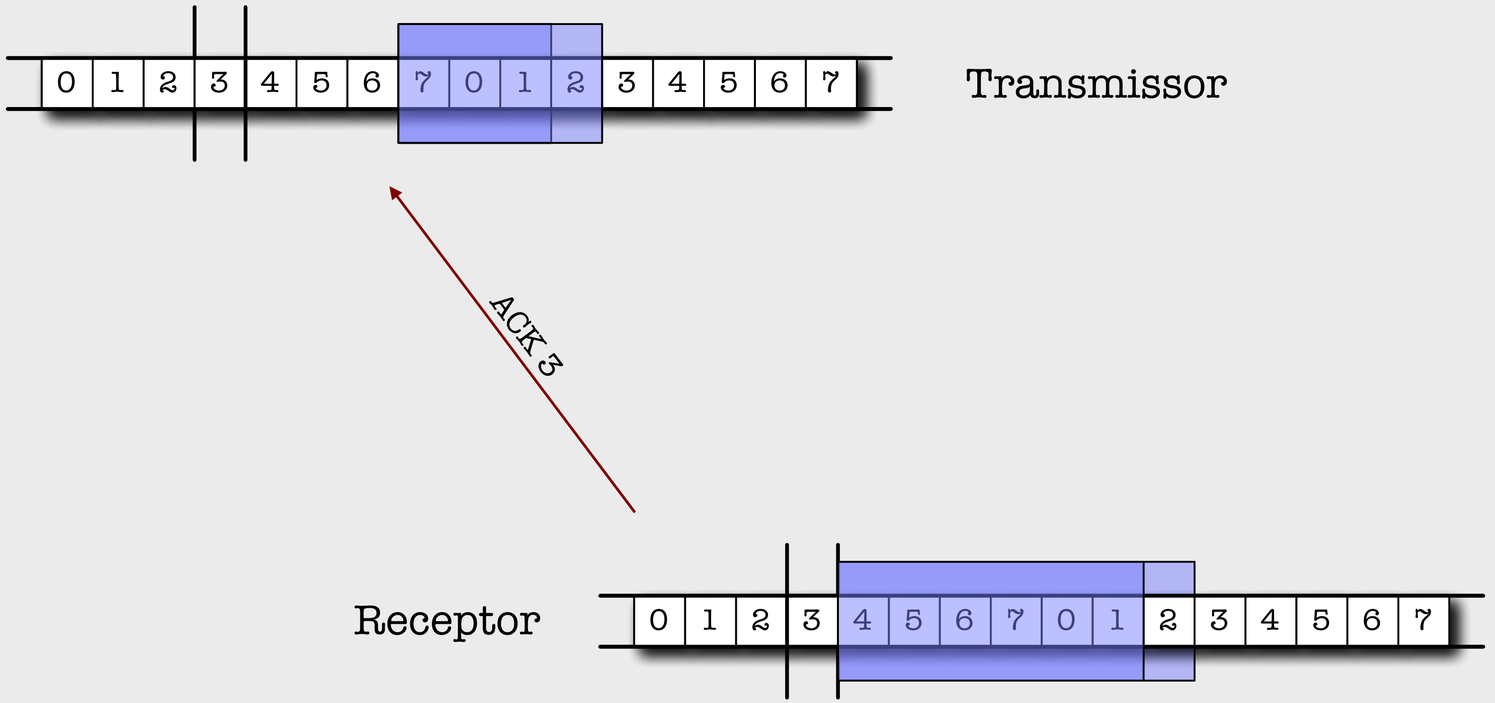
Janela Deslizante de Tamanho 3 bit

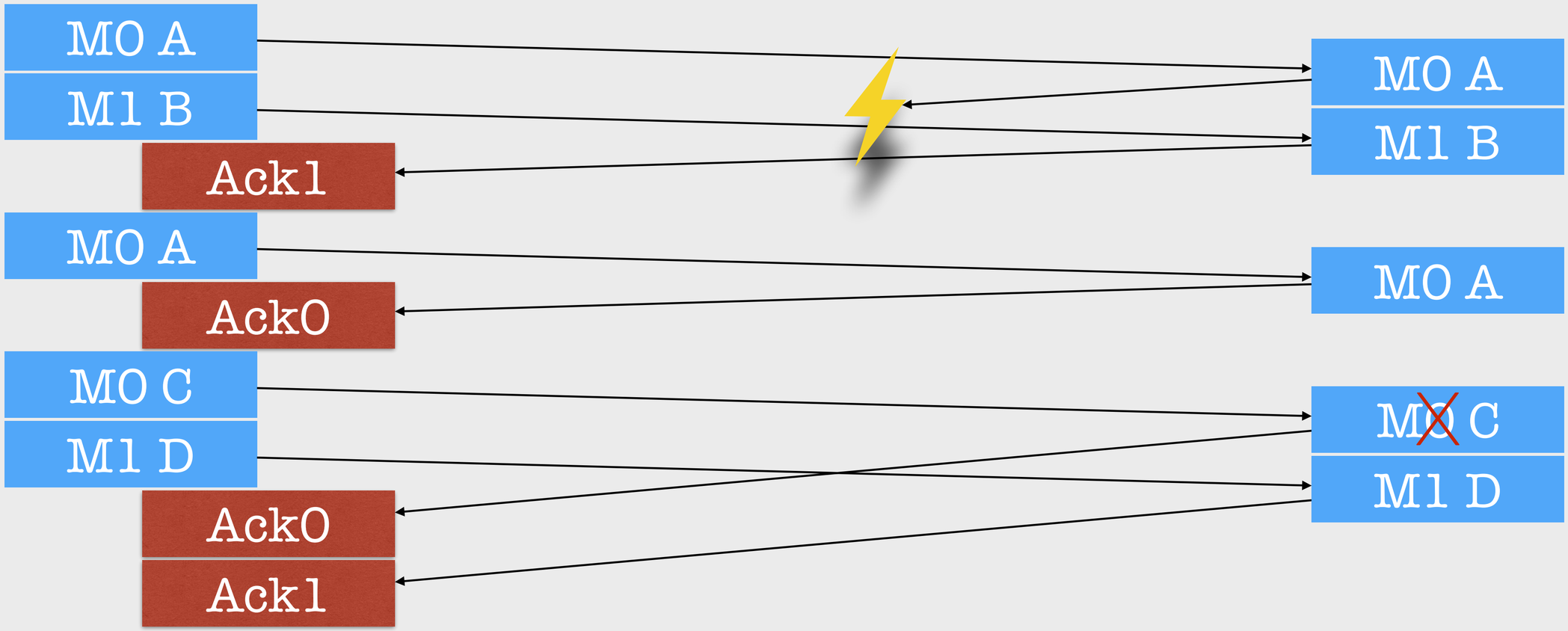


Janela Deslizante de Tamanho 3 bit



Janela Deslizante de Tamanho 3 bit

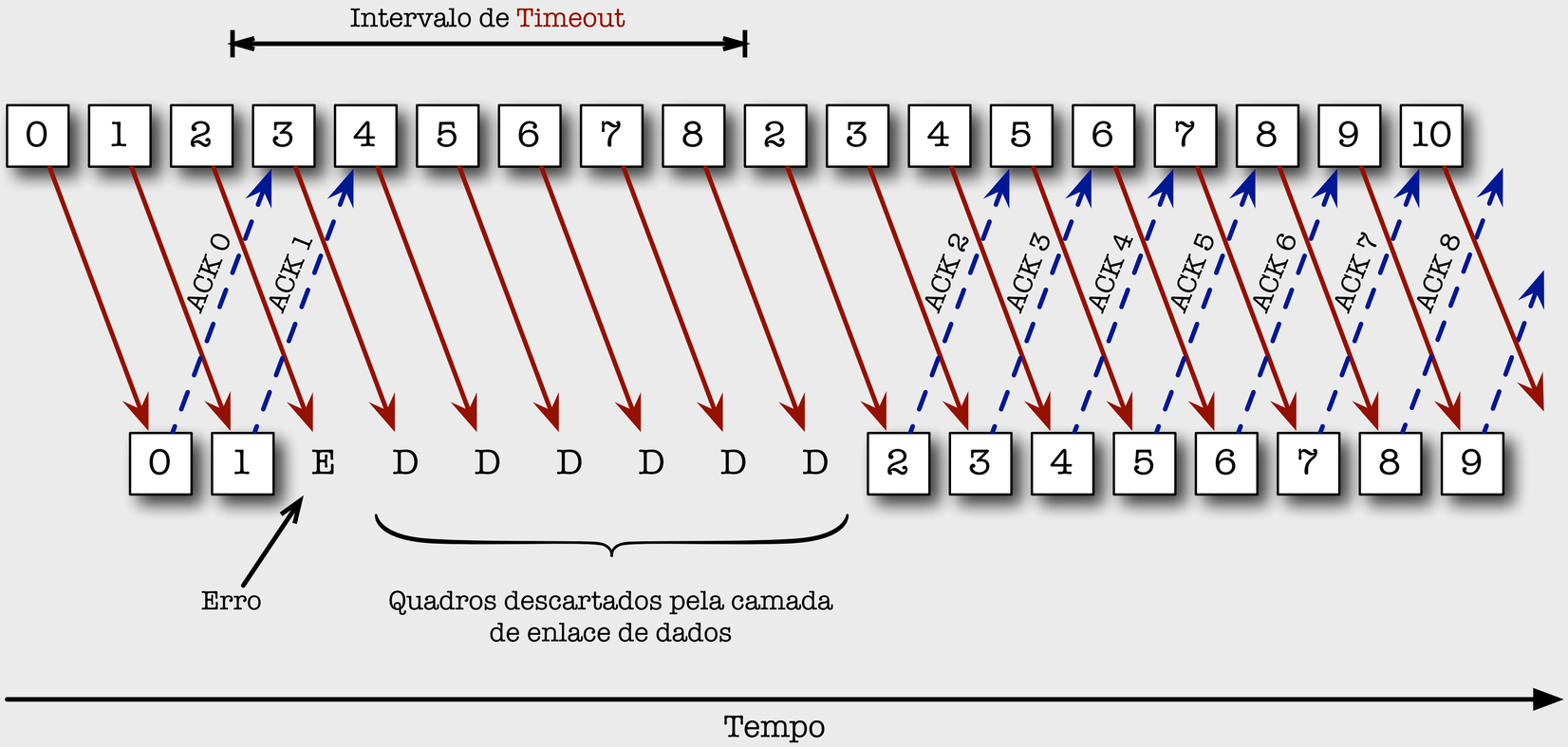



M0 A
M1 B
Ack1
M0 A
AckO
M0 C
M1 D
AckO
Ack1
M0 A
M1 B
M0 A
M0 C
M1 D
ABCD
ABAD

Protocolo “Go Back n”

- Que fazer quando se perde um dos quadros transmitidos? Go Back n
- Reenviar TODOS os quadros posteriores ao quadro perdido, inclusive
  - Após timeout do quadro perdido
- Receptor com janela de tamanho 1
  - Sem buffers para guardar quadros
- O receptor não pode passar quadros fora de ordem para a camada de rede

Protocolo “Go Back n”



Detalhes de Projeto

- Buffers de transmissão: quadros com ACK pendente devem ser armazenados temporariamente no transmissor (um buffer para cada quadro)
  - Buffers são liberados à medida em que ACKs são recebidos
  - Um ACK pode liberar um ou mais buffers

Detalhes de Projeto

- A camada de rede não possui um suprimento contínuo de pacotes
  - Ela interrompe a camada de enlace quando há pacotes
  - Camada de rede pode ser desabilitada quando a janela de transmissão está cheia
  - A cada ACK recebido, buffers podem ser liberados e novos pacotes podem ser aceitos da camada de rede

Detalhes de Projeto

- Números de seqüência dos quadros
  - De 0 a MAX_SEQ
  - No máximo MAX_SEQ quadros podem estar com ACK pendente em um dado instante
  - MAX_SEQ+1 números de seqüência para impedir que ACKs sejam mal interpretados

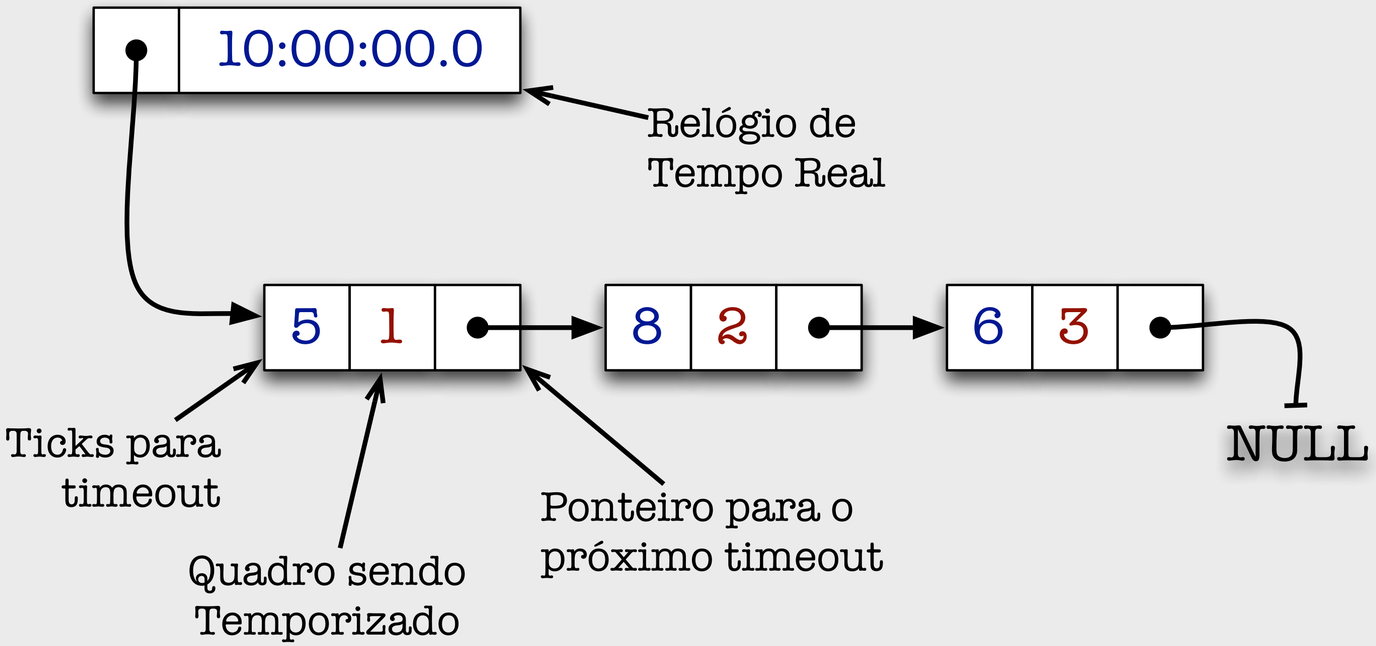
Exemplo

- Transmissor envia quadros 0 a 7
- Transmissor recebe ACK do quadro 7
- Transmissor envia os próximos 8 quadros (0 a 7)
- Outro ACK para o quadro 7 é recebido
- O que aconteceria se o segundo grupo de 8 quadros fosse perdido?

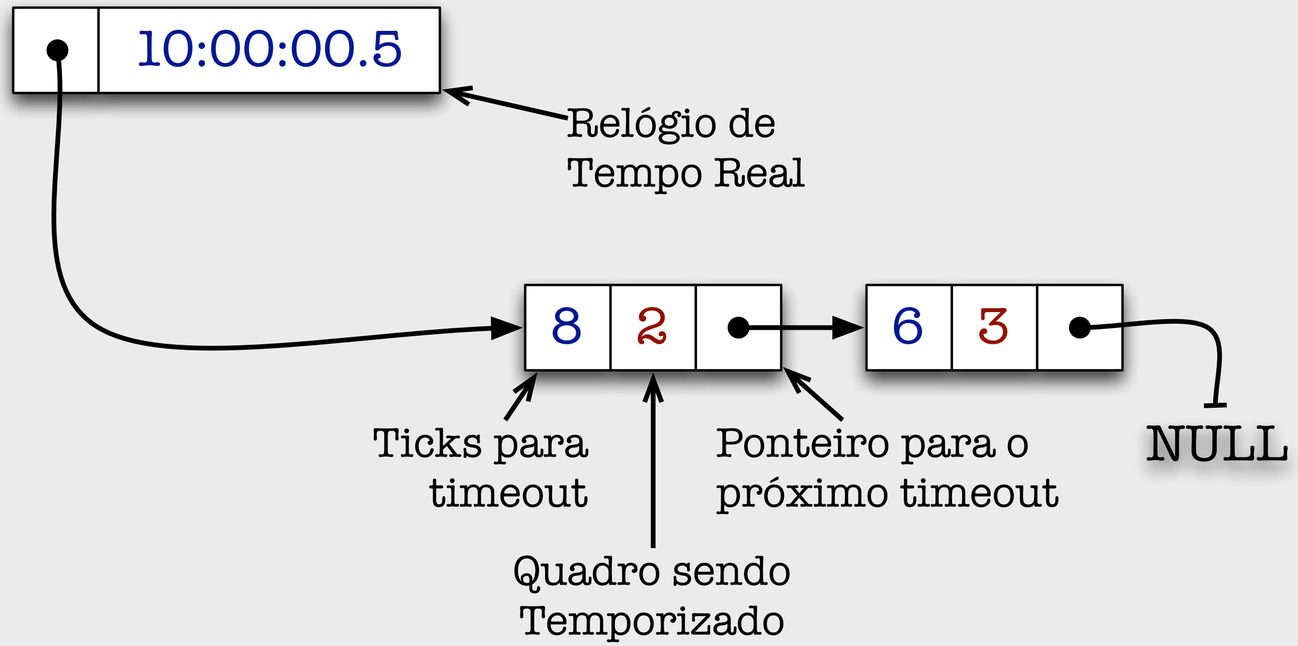
Detalhes de Projeto

- Temporizadores independentes devem ser associados a cada quadro transmitido
  - Cada quadro tem um período de timeout próprio
  - Temporizadores lógicos são usados
    - Com um único relógio físico

Detalhes de Projeto



Detalhes de Projeto



Protocolo 5 “Go Back n”

```
#define    MAX_SEQ    7
/* precisa ser igual a (2^n) - 1 */

typedef enum {
     frame_arrival,
     cksum_err,
     timeout,
     network_layer_ready
} event_type;

#include "protocol.h"
```

Protocolo 5 “Go Back n”

```
/*
    Return true if (a <=b < c circularly; false otherwise.
*/
static boolean between(seq_nr a, seq_nr b, seq_nr c)
{

    if ( ((a <= b) && (b < c)) ||
         ((c < a) && (a <= b)) ||
         ((b < c) && (c < a)) )
         return(true);
    else
         return(false);
}
```

Protocolo 5 “Go Back n”

```
/* Constrói e envia um quadro de dados */
static void send_data (seq_nr frame_nr,
                       seq_nr frame_expected,
                       packet buffer[])
{
    frame s;                                        /* quadro */
    s.info = buffer[frame_nr];  /* insere pacote no quadro */
    s.seq = frame_nr;           /* insere número de seqüência */

    /* piggyback ack */
    s.ack = (frame_expected+MAX_SEQ)%(MAX_SEQ+1);

    to_physical_layer(&s);      /* transmite o quadro  */
    start_timer(frame_nr);      /* inicia temporizador */
}
```

Protocolo 5 “Go Back n”

```
void protocol5(void)
{
     seq_nr next_frame_to_send;
     seq_nr ack_expected;      /* quadro a espera de ack */
     seq_nr frame_expected; /* próximo quadro  */
     frame r;                  /* quadro */
     packet buffer[MAX_SEQ + 1];/* buffers de pacotes */
     seq_nr nbuffered;         /* # de buffers de saída usados */
     seq_nr i;                 /* índice para vetor de buffers */
     event_type event;

     enable_network_layer();
     ack_expected = 0;
     next_frame_to_send = 0;
     frame_expected = 0;
```

Protocolo 5 “Go Back n”

```
nbuffered = 0;

while (true) {
    wait_for_event(&event);

    switch(event) {
        case network_layer_ready: /* pacote a enviar */
            from_network_layer(
                        &buffer[next_frame_to_send]);

            nbuffered = nbuffered + 1;

            send_data(next_frame_to_send,
                        frame_expected, buffer);

            /* avança borda superior da janela */
            inc(next_frame_to_send);
            break;
```

Protocolo 5 “Go Back n”

```
case frame_arrival:
     from_physical_layer(&r);
     if (r.seq == frame_expected) {
          /* aceitar apenas quadros em ordem */
          to_network_layer(&r.info);
          /* avançar borda inferior da janela */
          inc(frame_expected);
     }
     while (between(ack_expected, r.ack,
                              next_frame_to_send)) {
          nbuffered = nbuffered - 1;
          stop_timer(ack_expected);
          inc(ack_expected); /* contrai janela */
     }
     break;
```

Protocolo 5 “Go Back n”

```
            case cksum_err: break;      /* ignore quadro */

            case timeout: /* reenvia todos quadros */
                next_frame_to_send = ack_expected;
                for (i = 1; i <= nbuffered; i++) {
                    send_data(next_frame_to_send,
                                    frame_expected, buffer);
                    inc(next_frame_to_send);
                }
        } /* fim da estrutura switch */

        if (nbuffered < MAX_SEQ)
            enable_network_layer();
        else
            disable_network_layer();
    } /* fim da estrutura while */
}
```

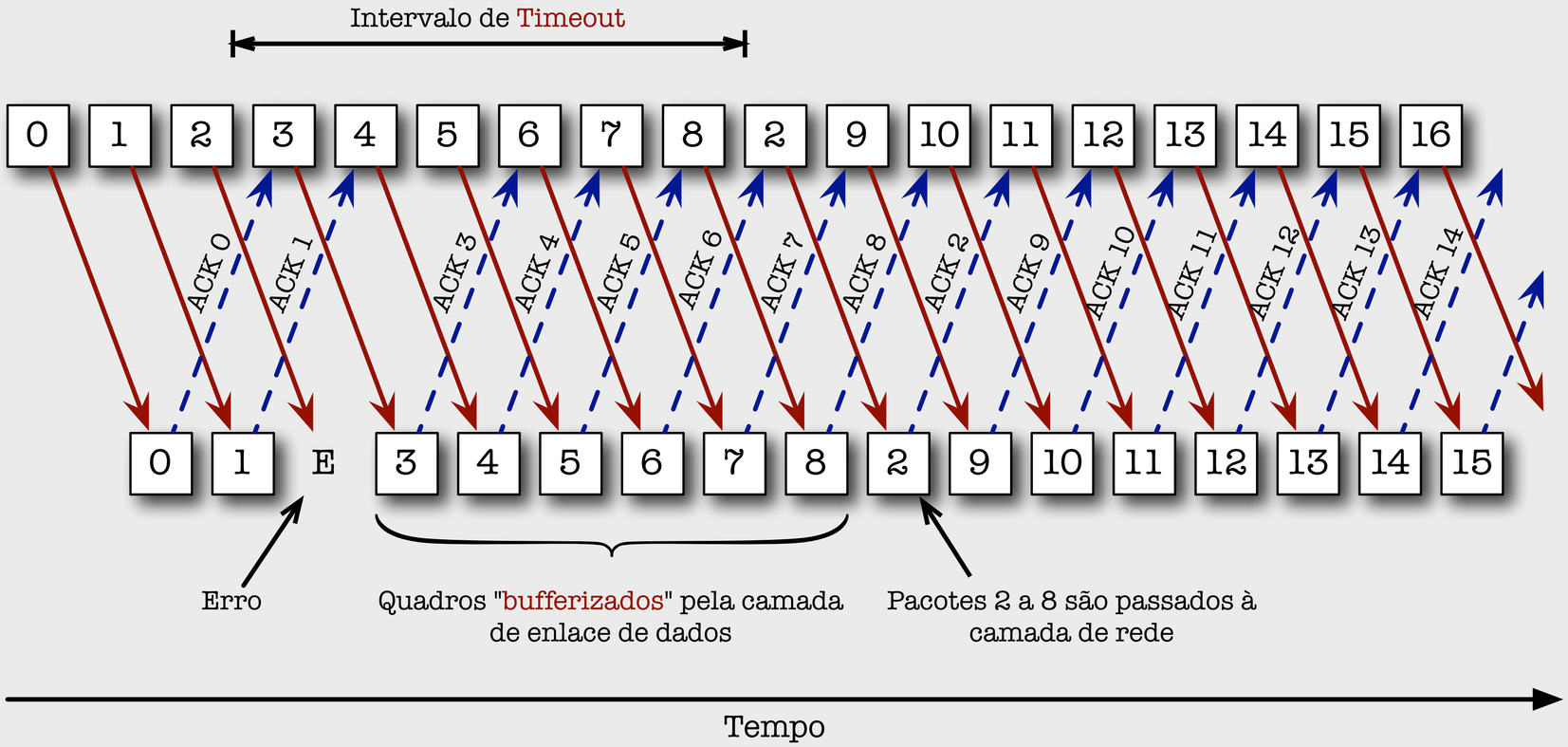
Protocolo Repetição Seletiva

- Alternativa para o protocolo 5 quando
  - Erros são freqüentes
  - Receptor possui espaço de buffer suficiente para janelas maiores que o tamanho 1
    - Um buffer para cada quadro que pode aceitar

Protocolo Repetição Seletiva

- Receptor aceita quadros recebidos fora de ordem, armazenando-os temporariamente
  - Até que possa entregá-los (em ordem) à camada de rede
  - Não descarta os quadros subseqüentes quando um quadro anterior for perdido ou danificado

Protocolo Repetição Seletiva



Detalhes de Projeto

- A faixa de números de seqüência deve ser grande o suficiente
  - O dobro do tamanho da janela
  - Evitar que duas janelas sucessivas se sobreponham
    - O que poderia causar erros no protocolo

Detalhes de Projeto

- Número de buffers necessários: equivale ao tamanho da janela
  - Um buffer para cada número de seqüência
  - bit para marcar se o buffer está cheio ou vazio
  - Um temporizador para cada buffer

Detalhes de Projeto

- Caso não haja um quadro de dados a ser transmitido, o ACK pode ser enviado em um quadro de controle independente
- Receptor só espera por um pacote da camada de rede por um certo tempo (ack_timeout)
- ack_timeout deve ser menor que o timeout para quadros de dados

Detalhes de Projeto

- Reconhecimentos negativos (NAK)
- Requisição para retransmissão de um quadro
- Quando o receptor suspeita de um erro: Quadro perdido ou recebido com erro de CRC
- Melhoram o desempenho global quando o tempo necessário para um quadro ser reconhecido não pode ser determinado com precisão

Detalhes de Projeto

- Determinação do quadro causou um timeout
  - Evento de timeout não diz explicitamente a que quadro ele se refere
  - Quadros são transmitidos em seqüência, um após o outro, um temporizador por quadro
  - Quadros mais “antigos” expiram antes
  - Logo, um evento de timeout refere-se ao quadro mais “antigo”

Atividade Complementar

- Applet animado ilustrando o protocolo de janelas deslizantes:
  - http://www.kom.e-technik.tu-darmstadt.de/projects/iteach/itbeankit/Applets/Sliding_Window/sliding_window.html