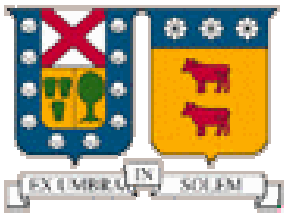
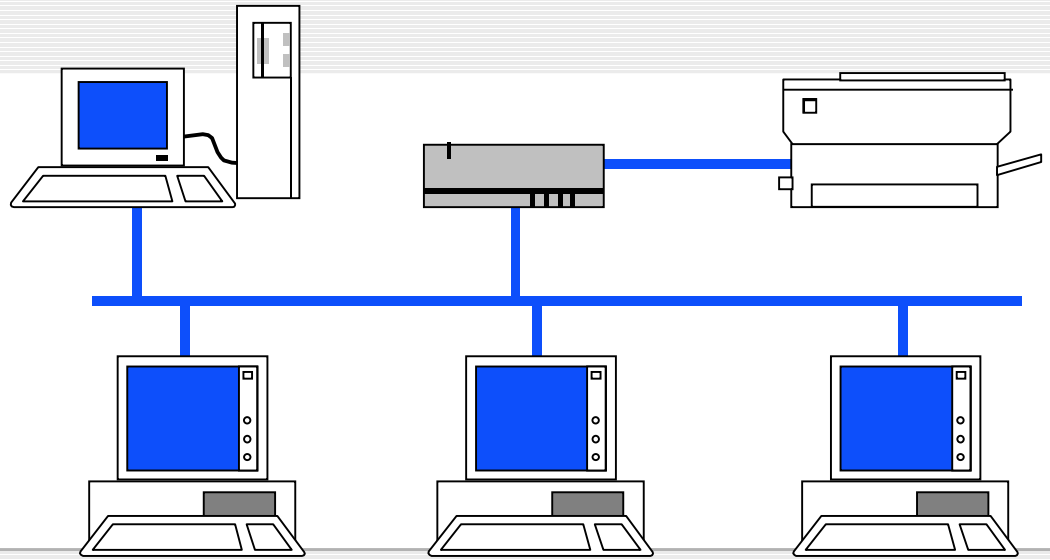


Redes de Computadores

Capa de Enlace de Datos



Control de Errores

- Básicamente existen dos técnicas de control de error:
  - ARQ **(Automatic Repeat Request)**
    - Se utiliza un protocolo de detección de error. Cada trama debe ser "reconocida" por el receptor (Ack, Nack)
  - FEC (**Forward Error Correction**)
    - Se utiliza un protocolo de corrección de error.

	VENTAJA	DESVENTAJA
ARQ	Tx pocos bits redundantes por trama	Por cada error debe retransmitir la trama
FEC	Muchos errores son corregidos en el receptor	Tx muchos bits redundantes por trama

Control de Errores y Control de Flujo

- Protocolo Stop & Wait (Pare y Espere)
- Protocolo Go Back-N
- Protocolo de Repetición Selectiva (Selective Repeat)
- Control de Flujo
  - Protocolo de Ventana Corrediza
- Ejemplo: Protocolo HDLC

Protocolo **Simplex**

- Situación 1: (con suposiciones..)
  - Comunicación Simplex
  - Capas de Red están siempre “listas”
  - Tiempo de Procesamiento despreciable
  - Buffer RX de tamaño infinito
  - Capa física no tiene problemas ni pierde marcos
  - No se enumeran los paquetes

¡¡ Situación Ficticia !!

Protocolo Simplex irreal

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                    /* buffer for an outbound frame */
  packet buffer;              /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;          /* copy it into s for transmission */
      to_physical_layer(&s);    /* send it on its way */
  }                             /* Tomorrow, and tomorrow, and tomorrow,
                                   Creeps in this petty pace from day to day
                                   To the last syllable of recorded time
                                       - Macbeth, V, v */
}

void receiver1(void)
{
  frame r;
  event_type event;           /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event); /* only possibility is frame_arrival */
      from_physical_layer(&r);/* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
  }
}
```

Protocolo **Half Duplex**

- Situación 2:
  - Canal es libre de errores (¿existirá?)
  - Buffer de tamaño limitado!!
    - TX no debe saturar al RX
    - Se debe usar Control de Flujo Simple
      - RX avisa al TX cuándo puede recibir otro marco
      - TX espera en silencio por este aviso
    - Es por ende un protocolo Half duplex.

Protocolo Half Duplex

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                    /* buffer for an outbound frame */
  packet buffer;              /* buffer for an outbound packet */
  event_type event;           /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);                    /* go get something to send */
      s.info = buffer;        /* copy it into s for transmission */
      to_physical_layer(&s);  /* bye bye little frame */
      wait_for_event(&event); /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                 /* buffers for frames */
  event_type event;           /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event); /* only possibility is frame_arrival */
      from_physical_layer(&r); /* go get the inbound frame */
      to_network_layer(&r.info);/* pass the data to the network layer */
      to_physical_layer(&s);  /* send a dummy frame to awaken sender */
  }
}
```

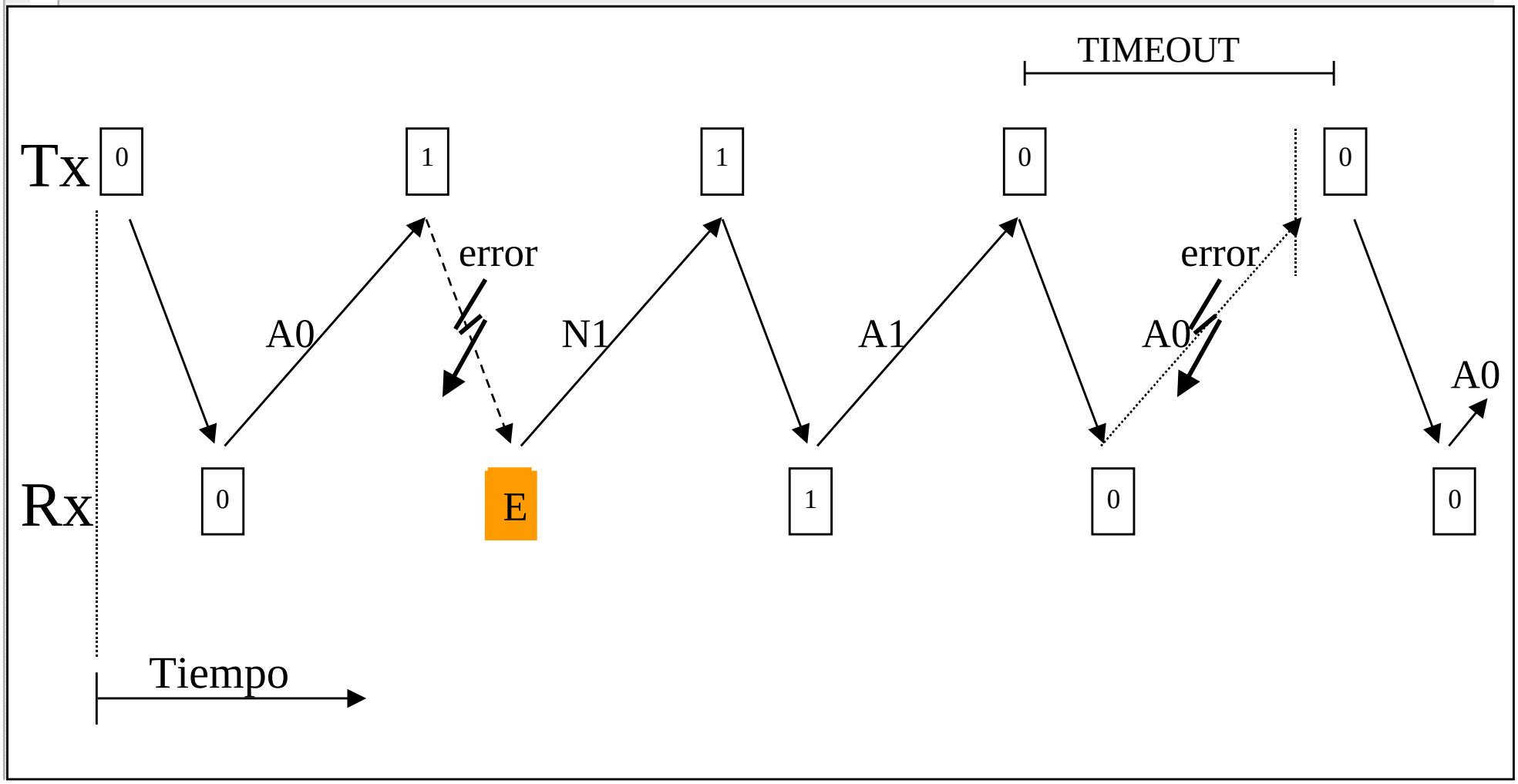
Protocolo ARQ **STOP & WAIT**

- Situación 3:
  - Canal ahora posee **ruido (errores)**
  - buffer limitado $\Rightarrow$ Control de Flujo
  - Idea: asegurar que cada trama transmitida es recibida correctamente antes de transmitir la siguiente
  - Se usa Control de Errores ARQ simple.

Protocolo ARQ **STOP & WAIT**

- ARQ Simple
  - el RX acusa recibo de cada frame “correcto” (no detecta error) con otro frame llamado ACK (Acknowledge)
  - ¿y si el RX recibió el marco con error? (se detecta error)
    - N-ACK (Negative Acknowledge)
  - ¿y si el marco no “alcanzó” a llegar al RX?
    - Temporizador en el TX causa re-TX
  - y si el acuse de recibo se perdió, se repetirán marcos !!
    - Se enumeran los marcos !
    - Basta enumeración módulo 2 (0 y 1)

Protocolo ARQ **STOP & WAIT**



Protocolo Simplex STOP & WAIT

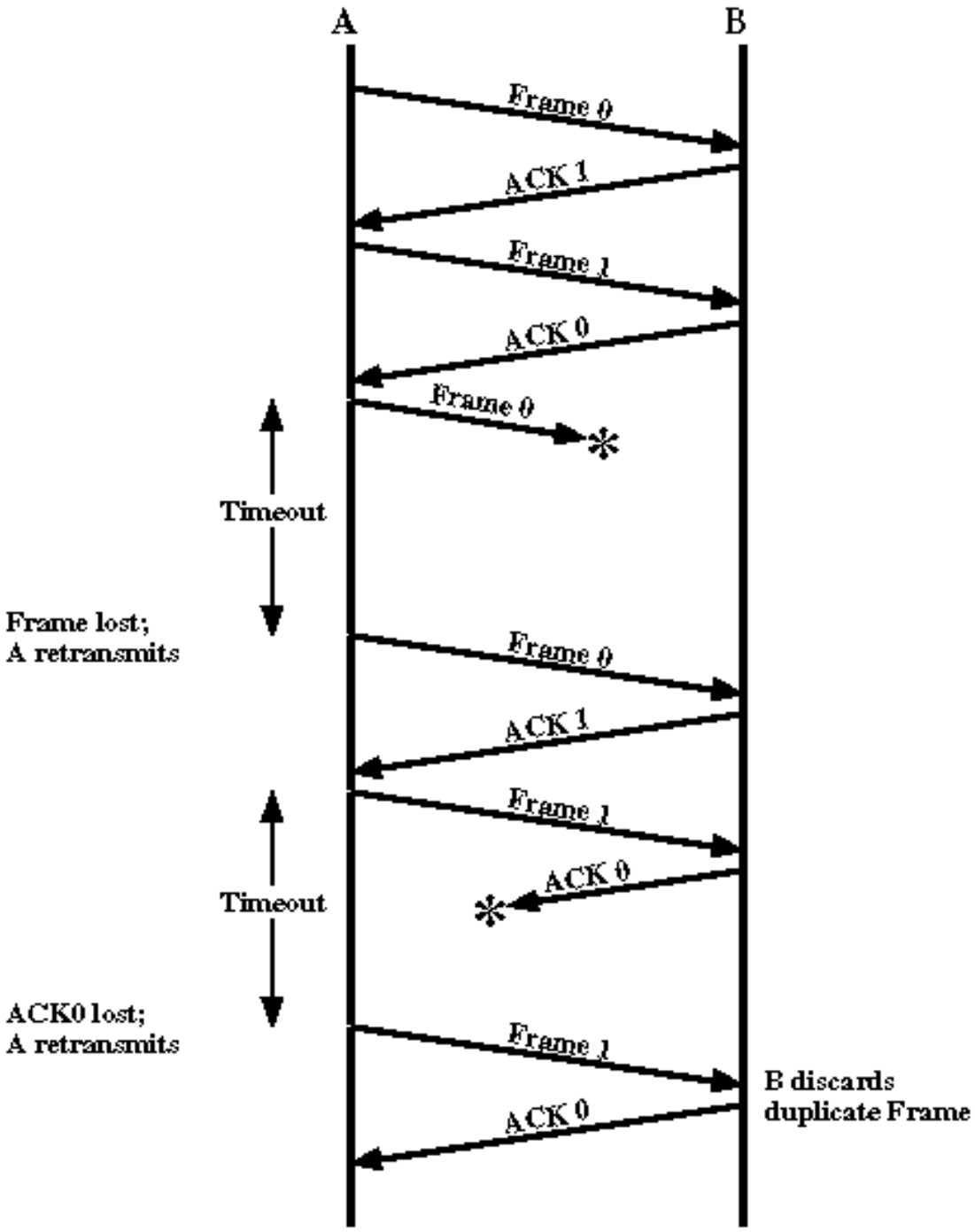


Figure 5.15 Stop-and-Wait ARQ

Protocolo ARQ STOP & WAIT

TX

```
#define MAX_SEQ 1                                   /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;                        /* seq number of next outgoing frame */
  frame s;                                          /* scratch variable */
  packet buffer;                                    /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;                           /* initialize outbound sequence numbers */
  from_network_layer(&buffer);                      /* fetch first packet */
  while (true) {
       s.info = buffer;                             /* construct a frame for transmission */
       s.seq = next_frame_to_send;                  /* insert sequence number in frame */
       to_physical_layer(&s);                       /* send it on its way */
       start_timer(s.seq);                          /* if (answer takes too long, time out */
       wait_for_event(&event);                      /* frame_arrival, cksum_err, timeout */
       if (event == frame_arrival) {
            from_physical_layer(&s);                /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                 from_network_layer(&buffer);/* get the next one to send */
                 inc(next_frame_to_send); /* invert next_frame_to_send */
            }
       }
  }
}
```

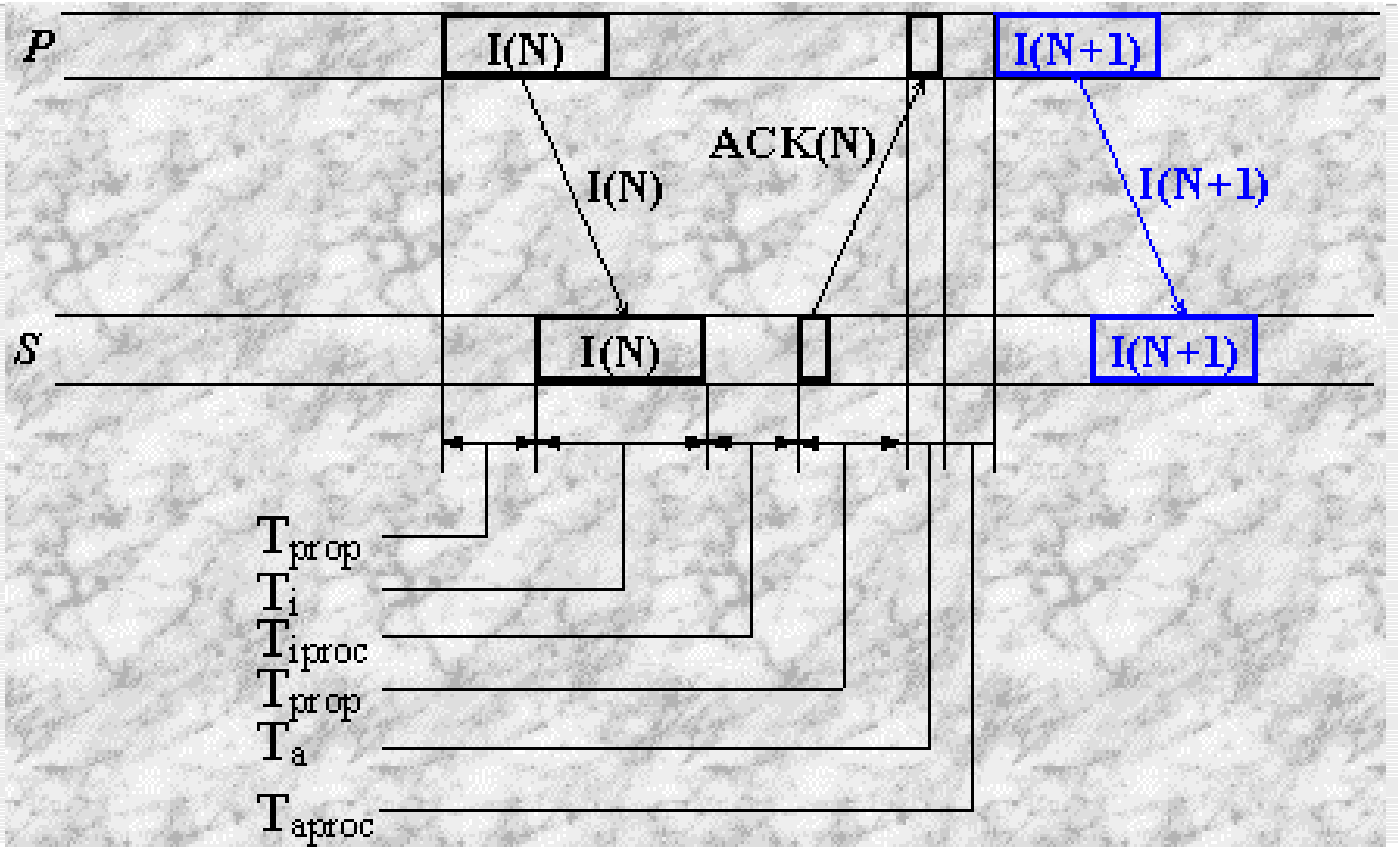
Protocolo ARQ STOP & WAIT RX

```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);                /* possibilities: frame_arrival, cksum_err */
       if (event == frame_arrival) {          /* a valid frame has arrived. */
            from_physical_layer(&r);          /* go get the newly arrived frame */
            if (r.seq == frame_expected) {    /* this is what we have been waiting for. */
                 to_network_layer(&r.info);   /* pass the data to the network layer */
                 inc(frame_expected);         /* next time expect the other sequence nr */
            }
            s.ack = 1 – frame_expected;       /* tell which frame is being acked */
            to_physical_layer(&s);            /* none of the fields are used */
       }
  }
}
```

Protocolo ARQ Stop & Wait

Variables Ocupadas



Protocolo ARQ Stop & Wait

Variables Ocupadas

- Tprop — Tiempo de propagación entre TX y RX
- Ti — Tiempo de TX de un marco (de TX a RX)
- Tiproc — Tiempo de procesamiento de un marco en el RX
- Ta — Tiempo de TX de un ACK (de RX a TX)
- Taproc — Tiempo de procesamiento de un ACK en el TX

- La Utilización del canal (desde el punto de vista del TX) está dado por :

$$U = Ti / Tt$$

donde

$Tt = Ti +$ tiempo de espera por un ACK

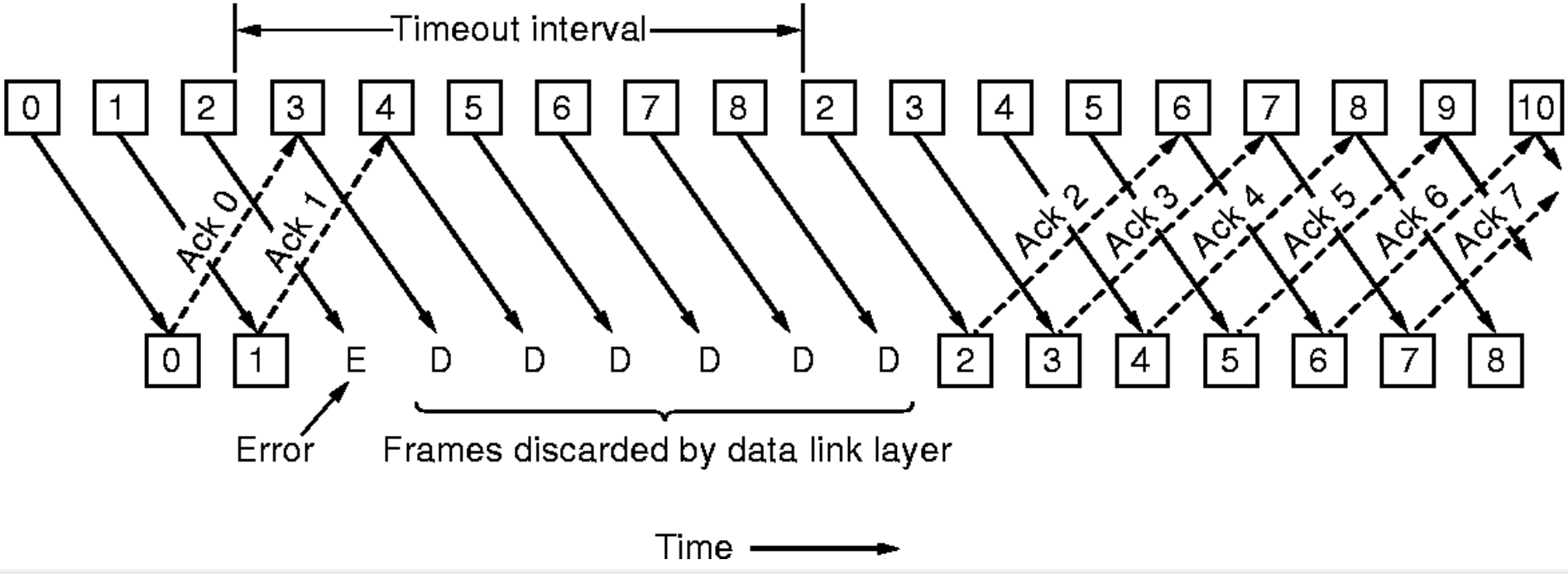
$Tt = 2Tprop + Ti + Tiproc + Ta + Taproc$

Protocolo ARQ Stop & Wait

- Este protocolo es adecuado para enlaces de corta distancia a baja tasa de bps
- El cálculo de U supone canal libre de errores (sin re-TX)
- Es posible que ocurra error en los marcos, los ACK y NACK al mismo tiempo
- Este protocolo depende fuertemente de la capacidad de detectar todos los errores
- El Control de Flujo está implicitamente dado que se TX en half-duplex y por ende el RX no se satura.

Go Back-N

- Ventana RX=1
- Transmisor posee un timeout por cada frame



Go Back-N

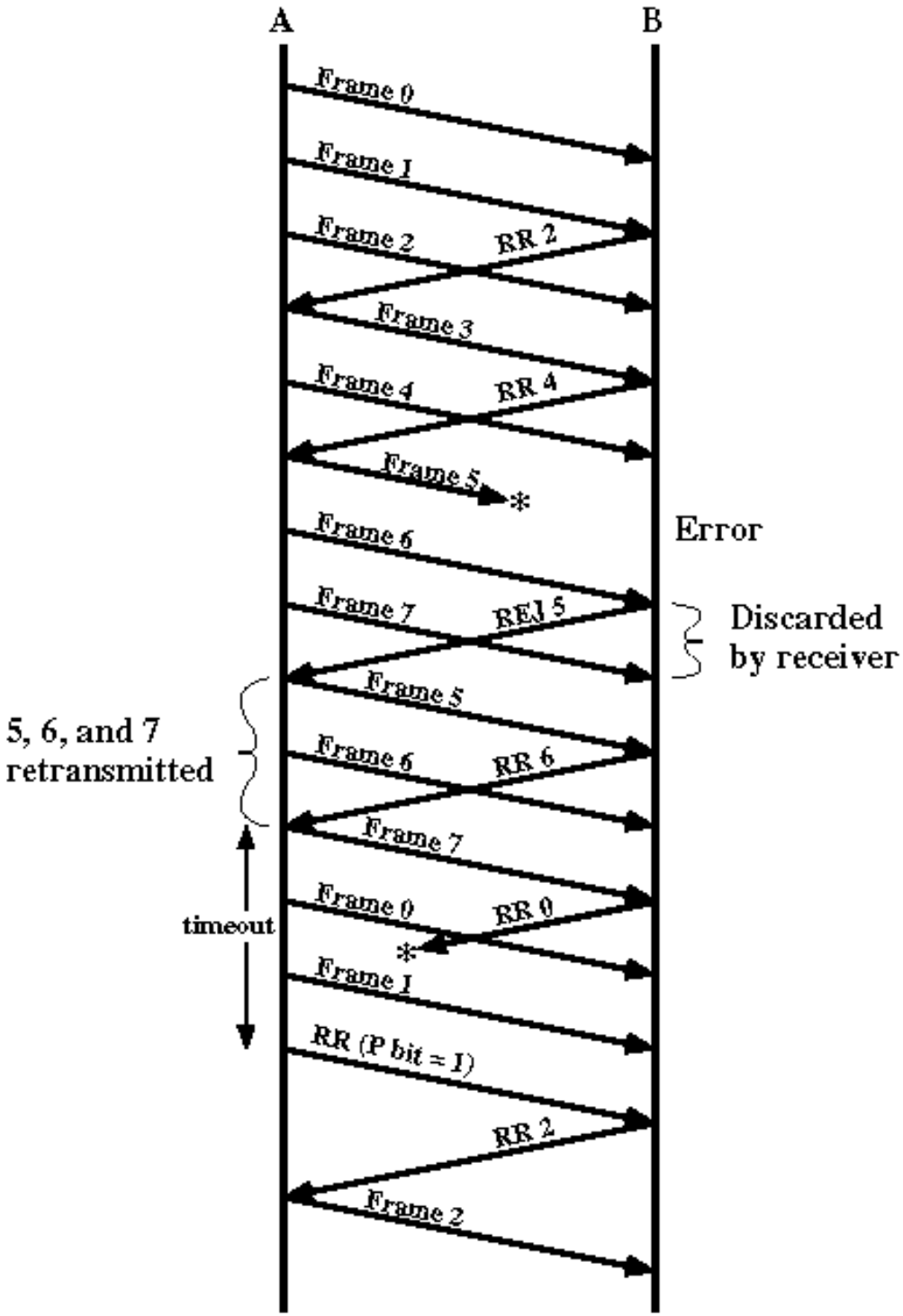
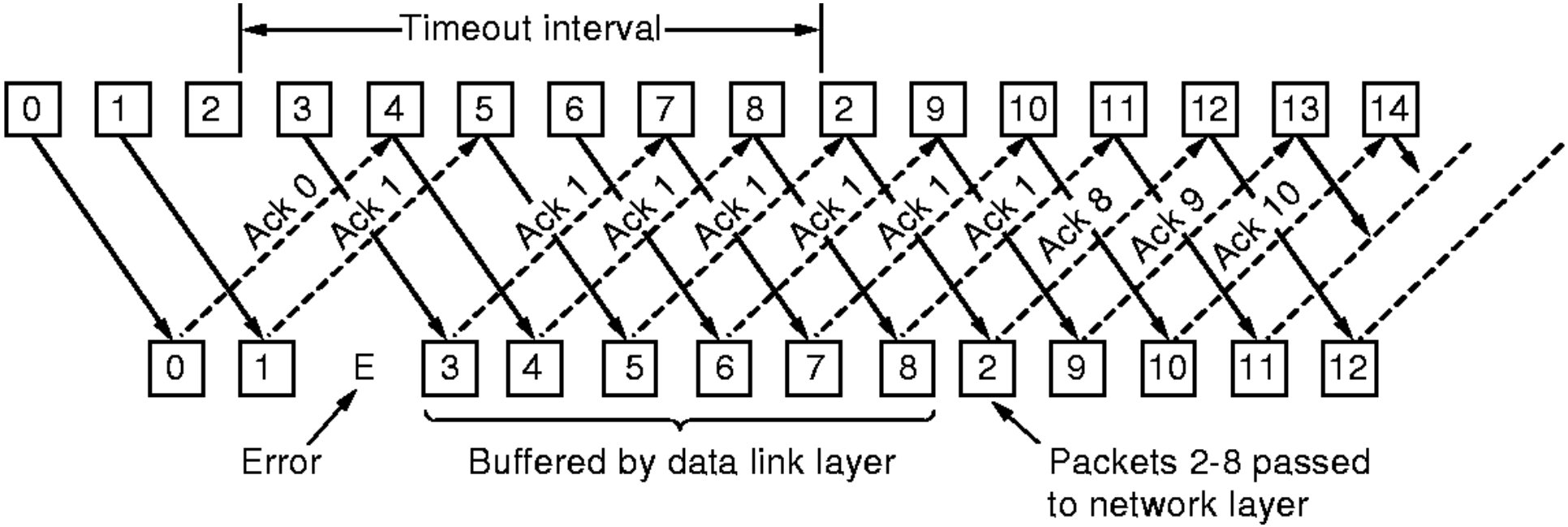


Figure 5.16 Go-back-N ARQ

Selective Repeat

- Ventana RX >1
- Transmisor posee un timeout por cada frame



Control de Flujo

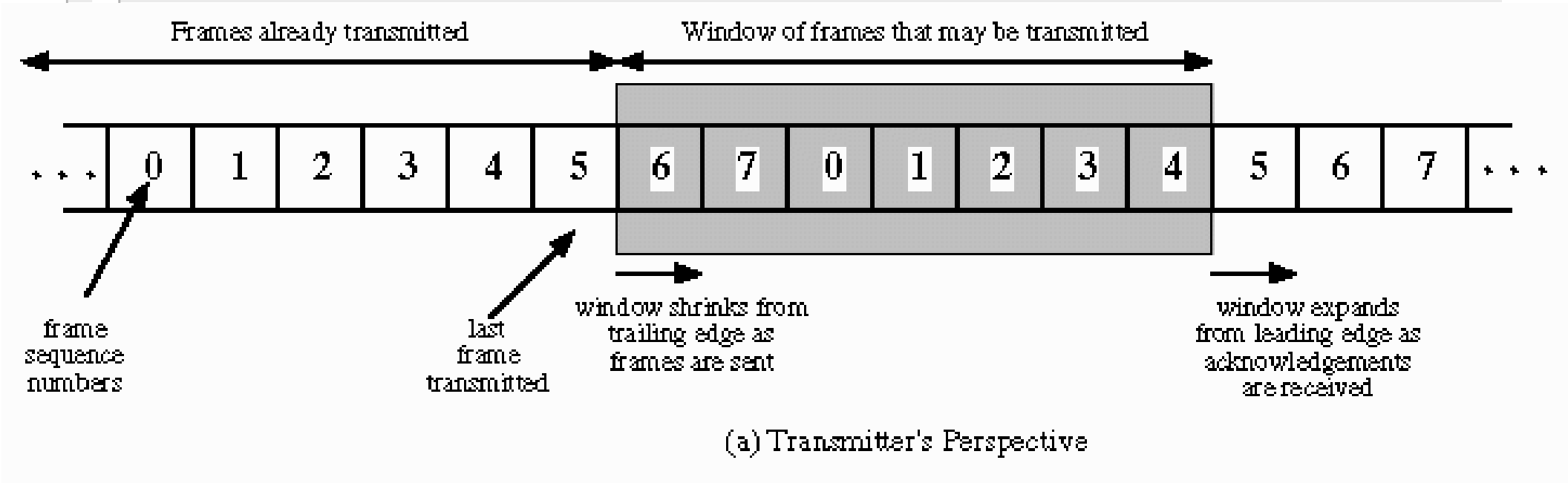
- Stop & Wait
  - ventana de RX = 1 y TX espera ACK para TX
  - control de flujo es natural y simple
- Go-Back-N
  - si timeout es grande no hay problema
- Selective Repeat
  - si timeout es grande hay problema
  - RX debe tener un buffer muy grande
  - RX no puede indicar al TX cuándo está en “problemas”

Control de Flujo

- El Control de Flujo
  - se aplica para no saturar la ventana de RX
  - para ello se limita la ventana de TX
- Control de Flujo ON/OFF
  - RX indica al TX que pare de Transmitir (pero puede ser demasiado tarde)
- Protocolo de Ventana Corrediza
  - TX y RX poseen una ventana de tamaño máximo
- TCP/IP
  - RX puede variar el tamaño de la ventana del TX en forma dinámica

Ejemplo de Protocolo de Ventana Corrediza

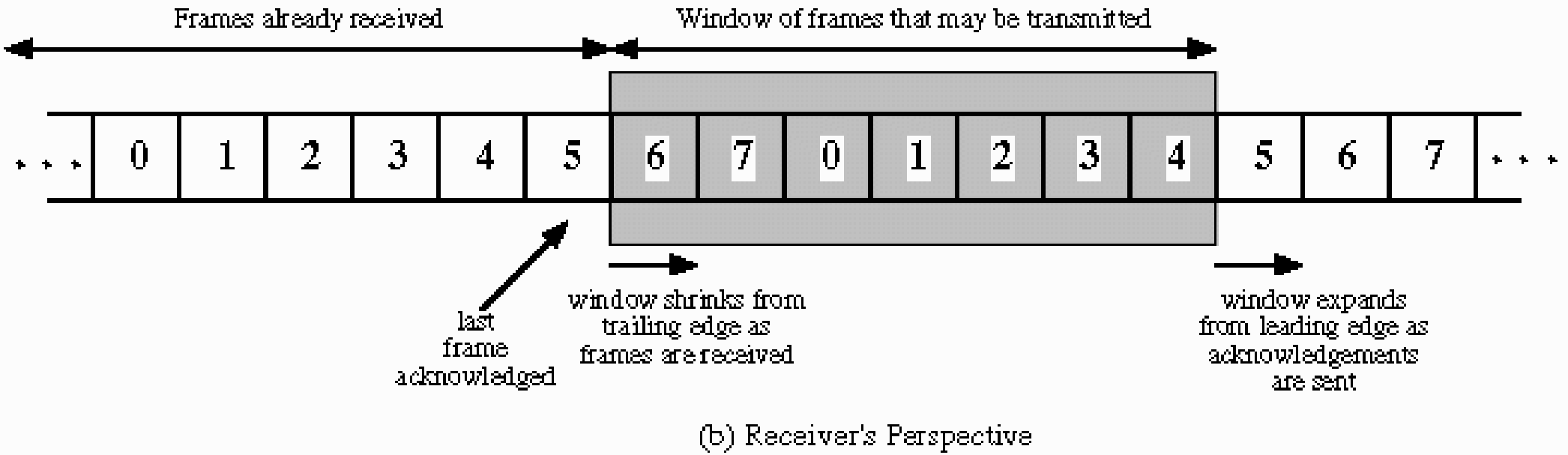
Transmisor



(a) Transmitter's Perspective

Ejemplo de Protocolo de Ventana Corrediza

Receptor



(b) Receiver's Perspective

Ejemplo de Protocolo de Ventana Corrediza

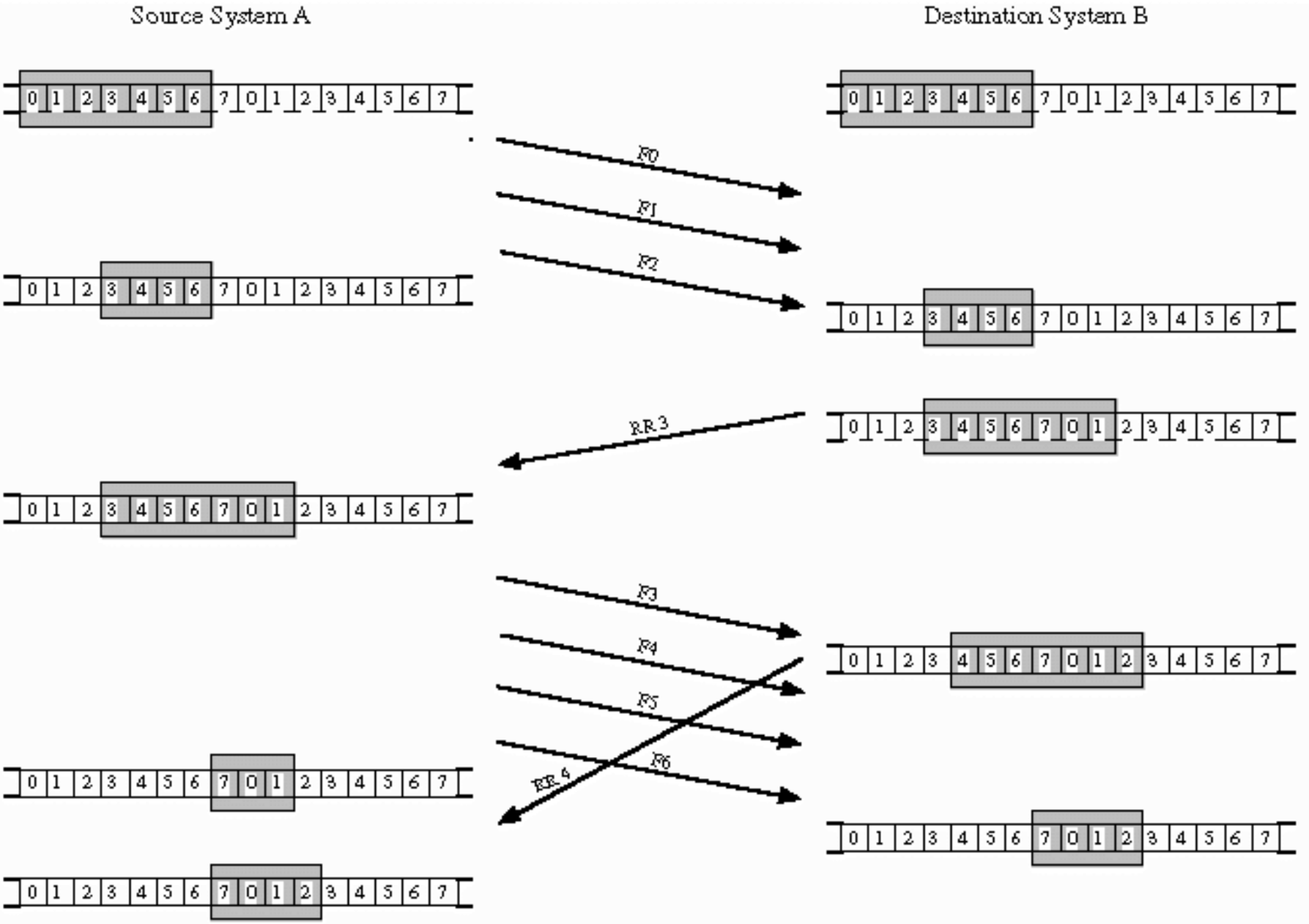


Figure 5.13 Example of a sliding-window protocol