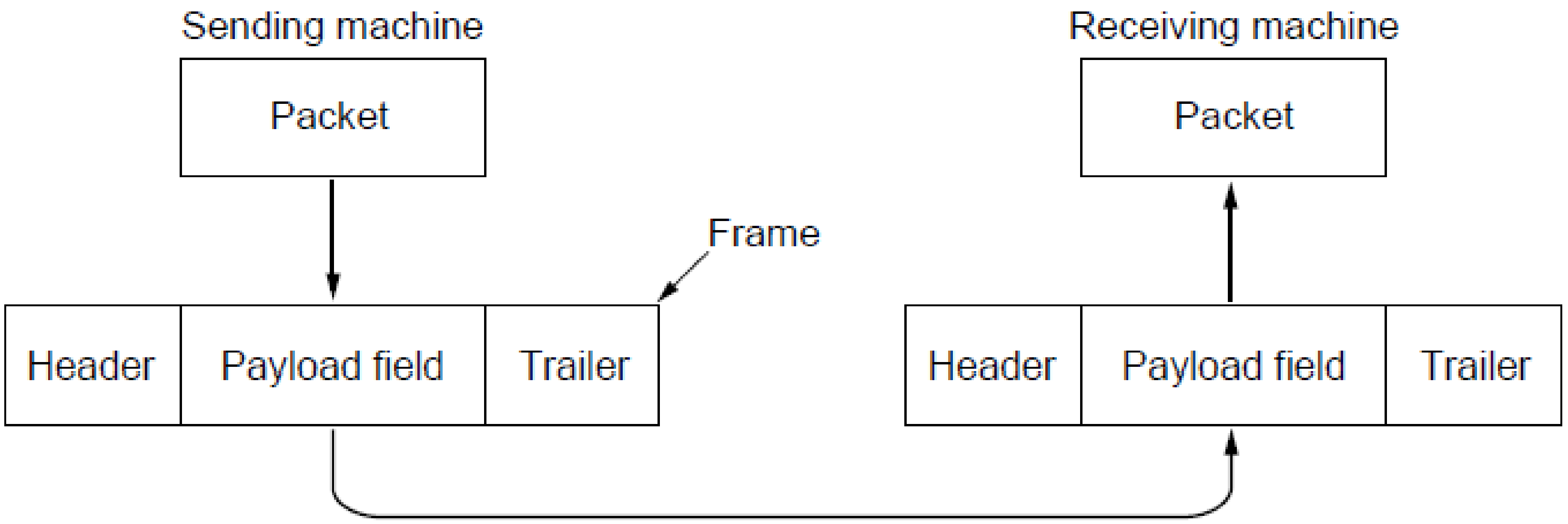
THE DATA LINK LAYER

DATA LINK LAYER DESIGN ISSUES

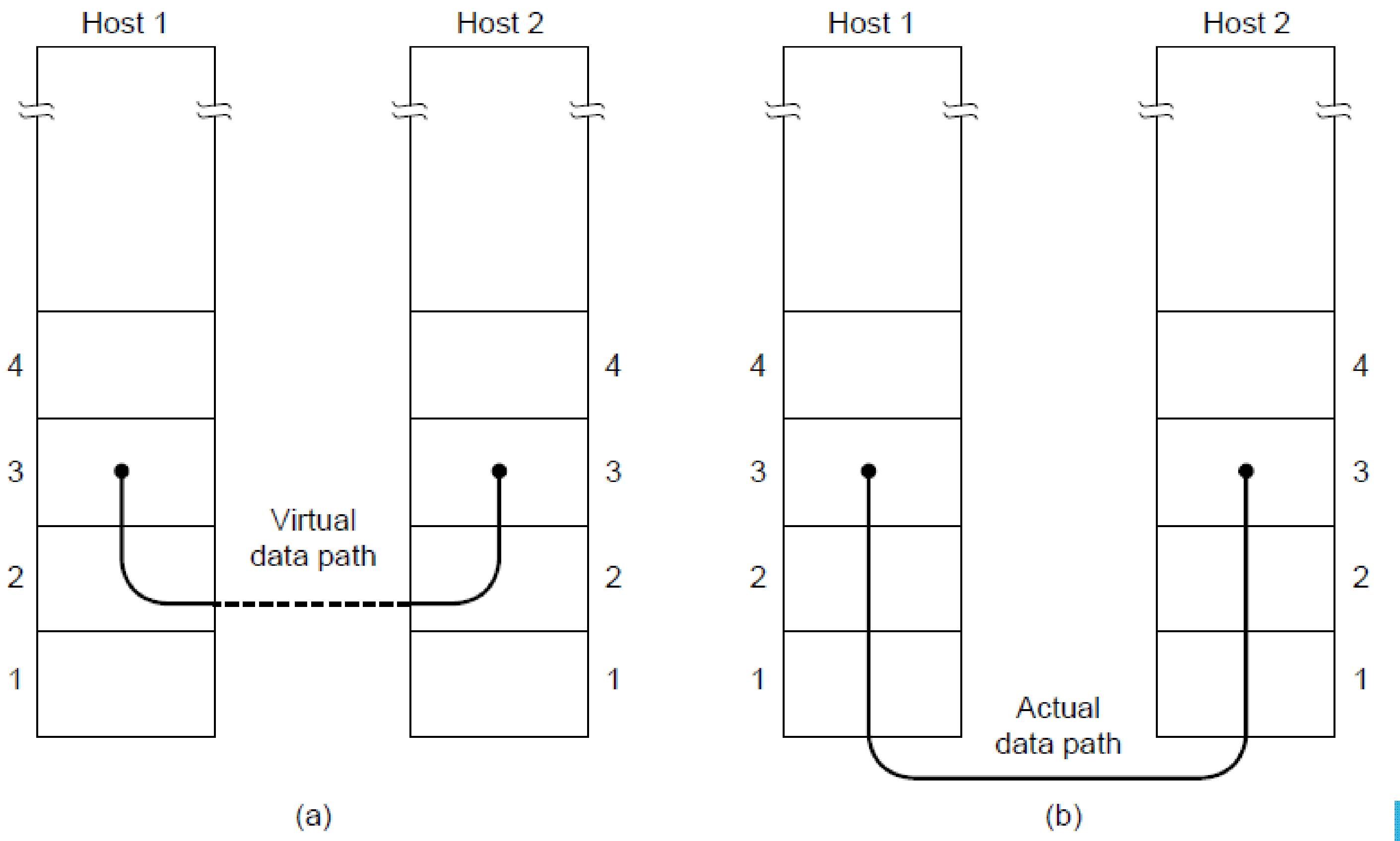
- **Network layer services**
- **Framing**
- **Error control**
- **Flow control**

PACKETS AND FRAMES

Relationship between packets and frames.



NETWORK LAYER SERVICES



POSSIBLE SERVICES OFFERED

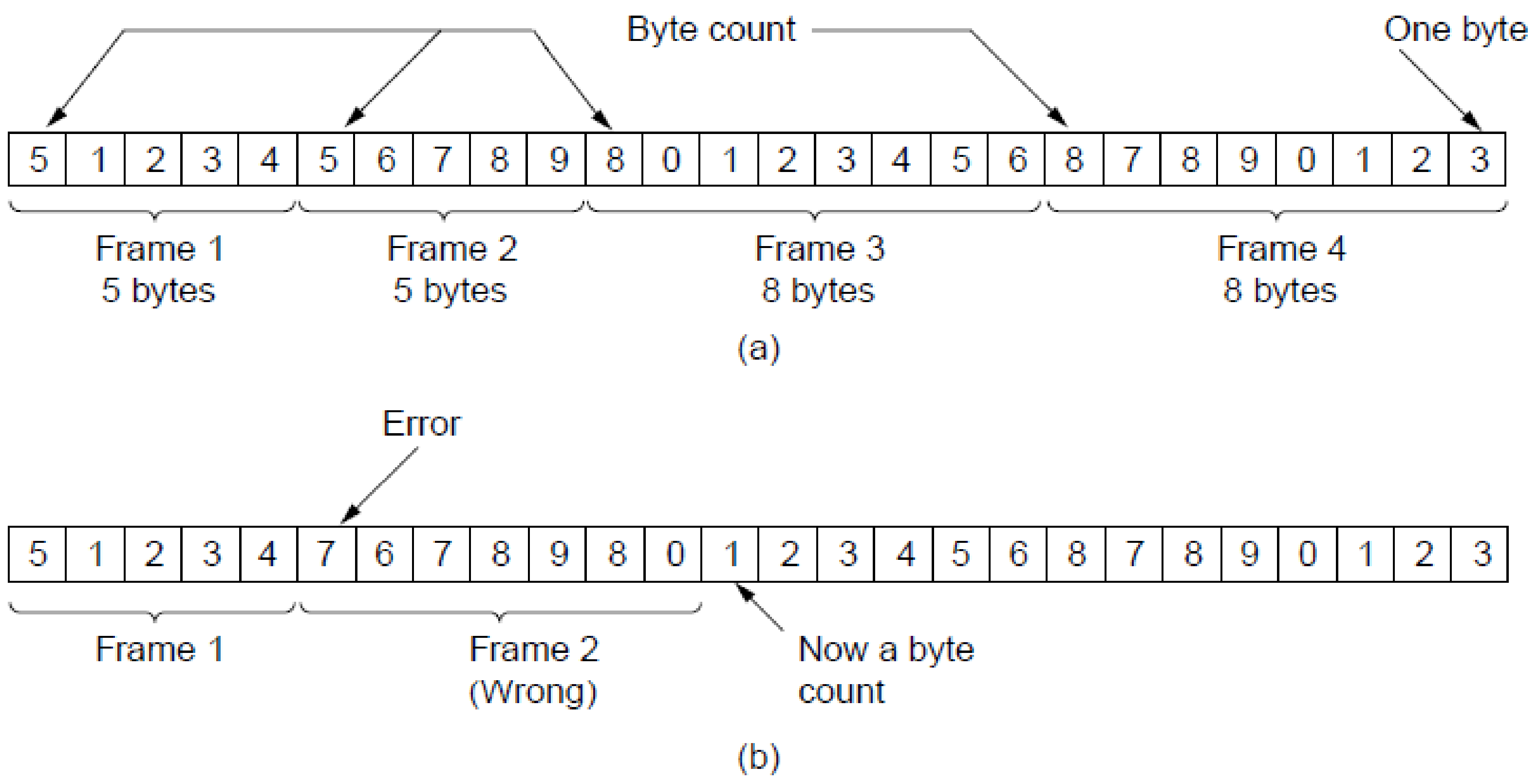
1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

FRAMING METHODS

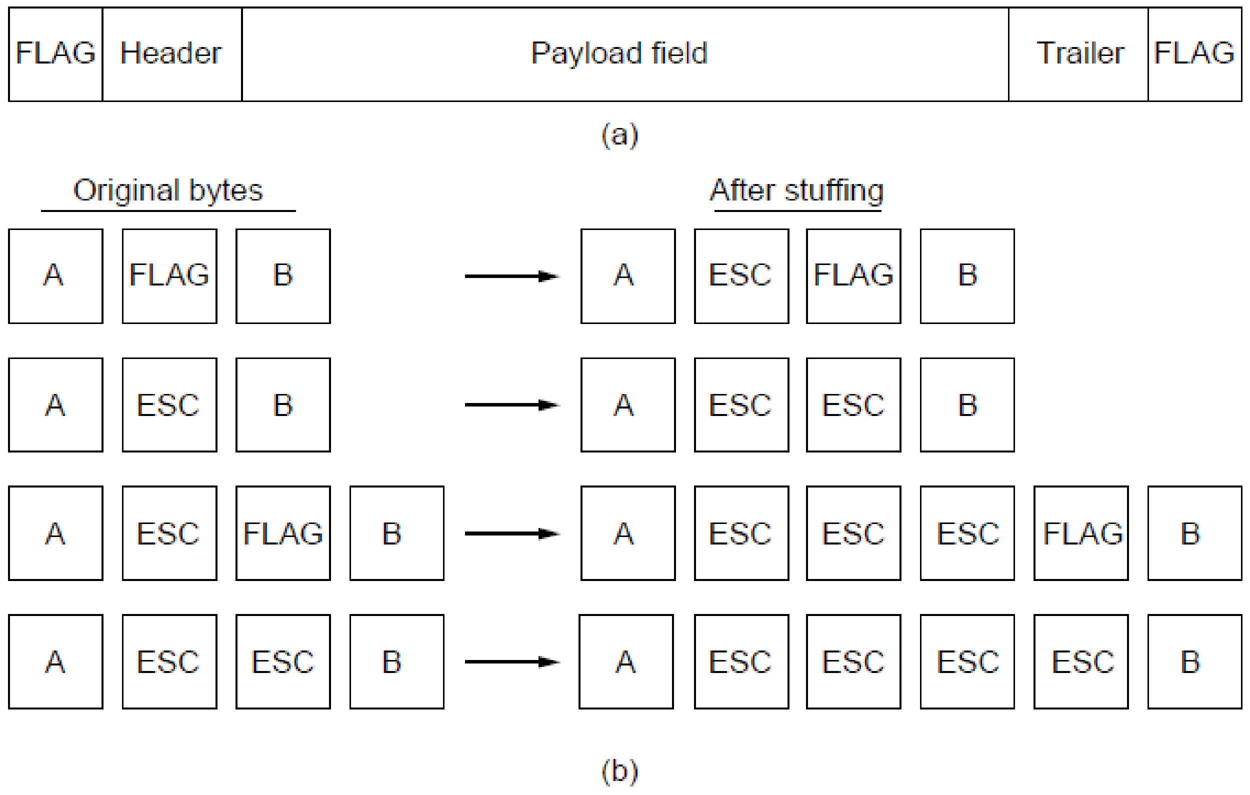
1. **Byte count.**
2. **Flag bytes with byte stuffing.**
3. **Flag bits with bit stuffing.**
4. **Physical layer coding violations.**

FRAMING (1)

A byte stream. (a) Without errors. (b) With one error.



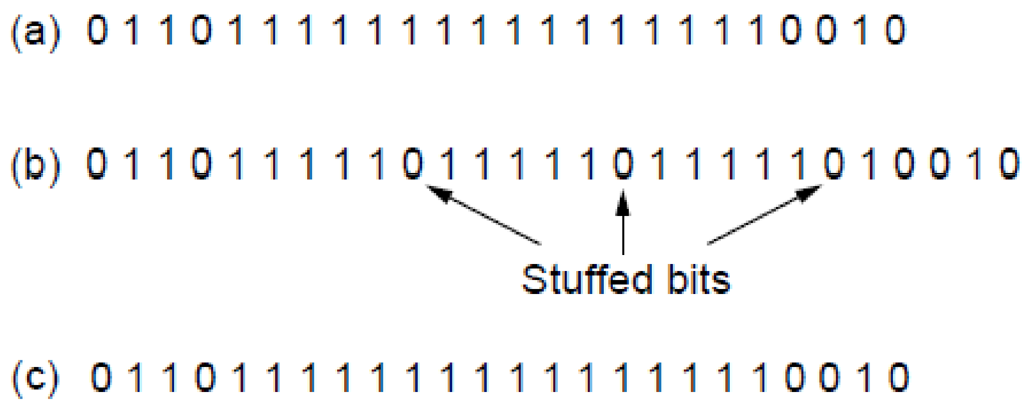
FRAMING (2)



A frame delimited by flag bytes.

Four examples of byte sequences before and after byte stuffing.

FRAMING (3)

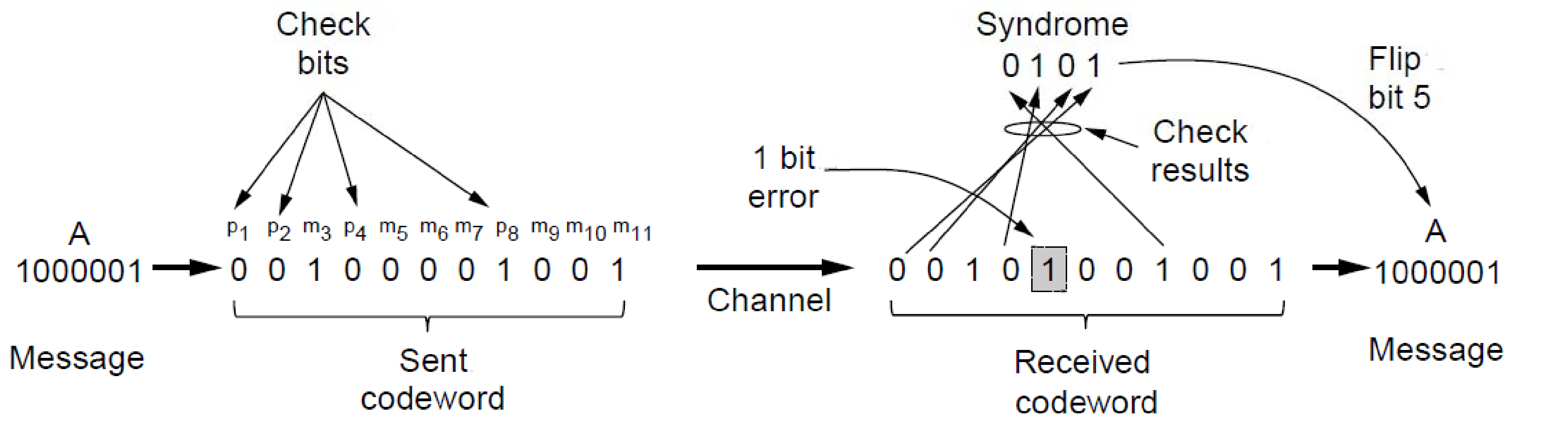

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0
Stuffed bits
(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

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.

ERROR CORRECTING CODES (1)

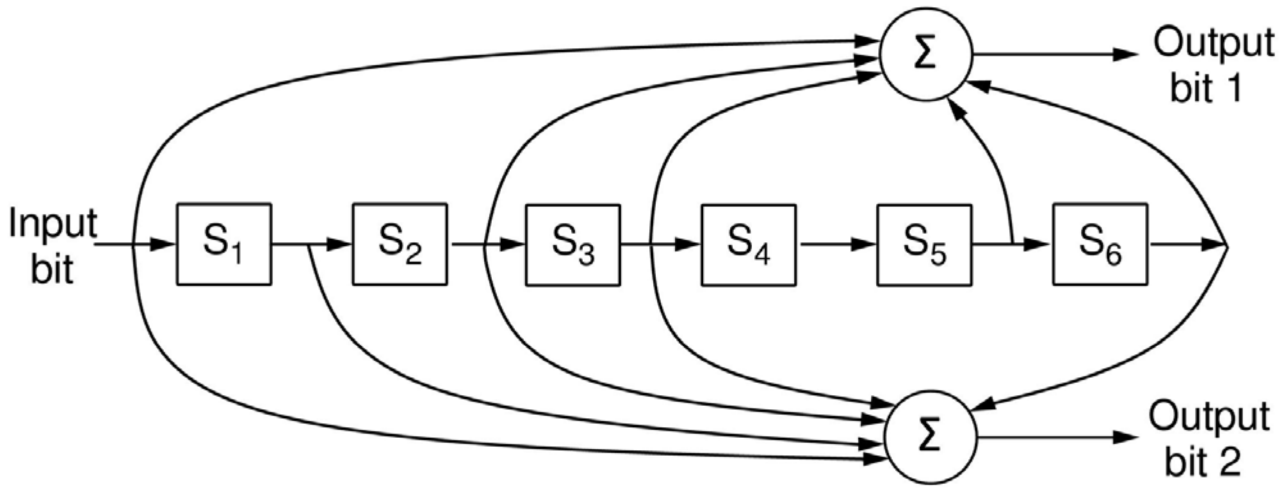
1. **Hamming codes.**
2. **Binary convolutional codes.**
3. **Reed-Solomon codes.**
4. **Low-Density Parity Check codes.**

ERROR CORRECTING CODES (2)



Example of an (11, 7) Hamming code correcting a single-bit error.

ERROR CORRECTING CODES (3)



The NASA binary convolutional code used in 802.11.

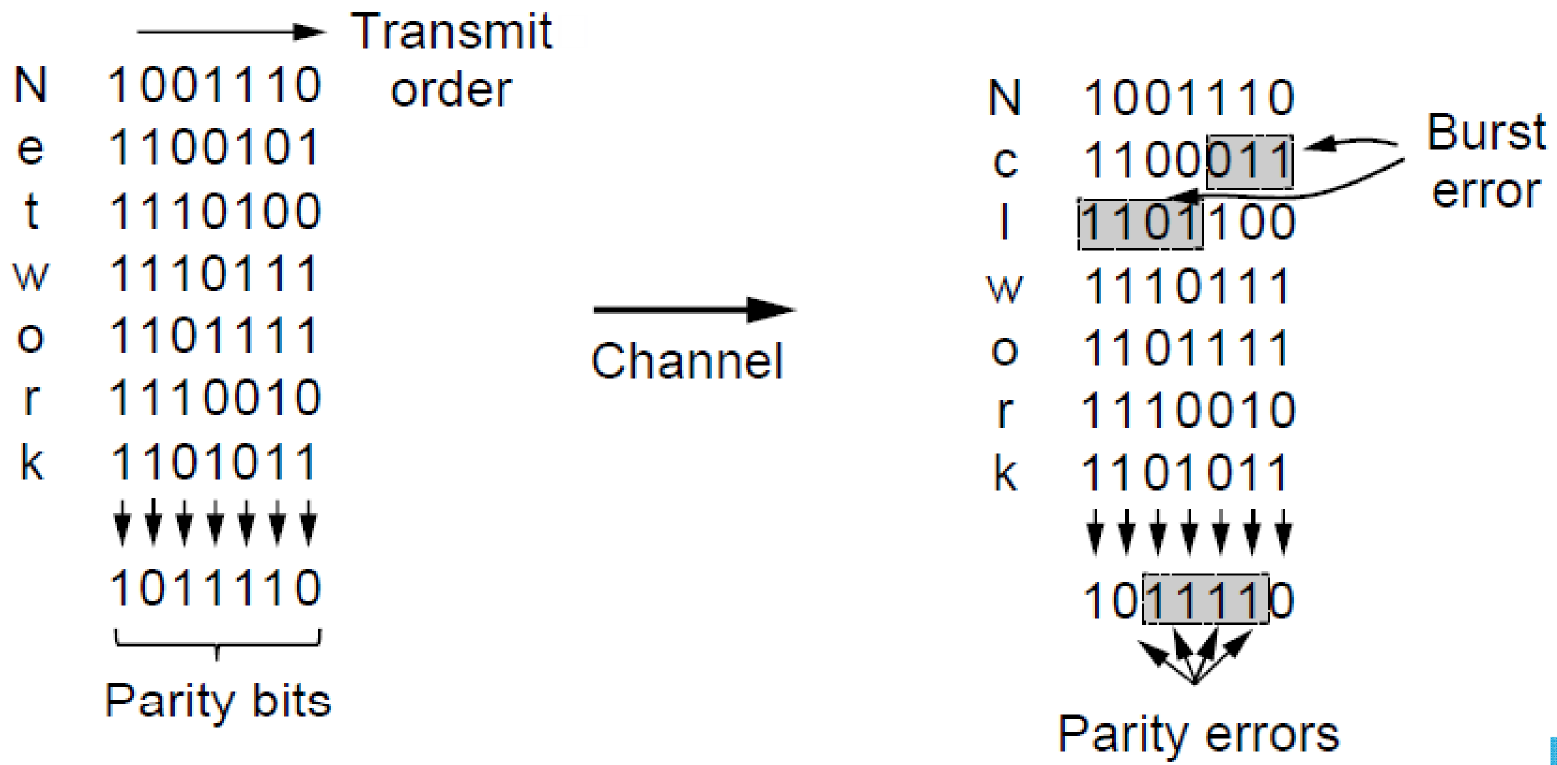
ERROR-DETECTING CODES (1)

Linear, systematic block codes

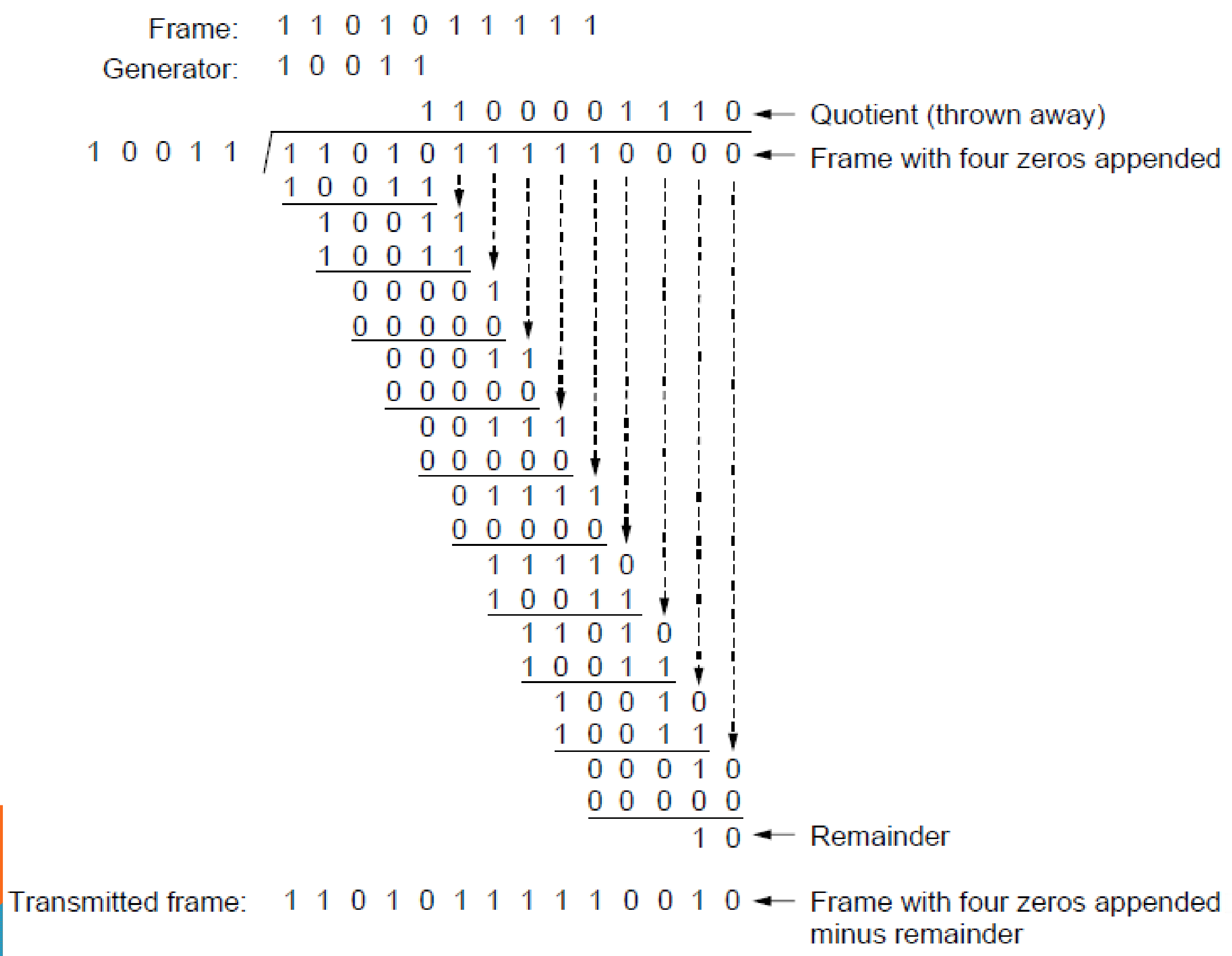
1. **Parity.**
2. **Checksums.**
3. **Cyclic Redundancy Checks (CRCs).**

ERROR-DETECTING CODES (2)

Interleaving of parity bits to detect a burst error.



ERROR-DETECTING CODES (3)



```
          Frame:  1 1 0 1 0 1 1 1 1 1
      Generator:  1 0 0 1 1

                              1 1 0 0 0 0 1 1 1 0  ←  Quotient (thrown away)
1 0 0 1 1  /  1 1 0 1 0 1 1 1 1 1 0 0 0 0  ←  Frame with four zeros appended
              1 0 0 1 1
                1 0 0 1 1
                1 0 0 1 1
                  0 0 0 0 1
                  0 0 0 0 0
                    0 0 0 1 1
                    0 0 0 0 0
                      0 0 1 1 1
                      0 0 0 0 0
                        0 1 1 1 1
                        0 0 0 0 0
                          1 1 1 1 0
                          1 0 0 1 1
                            1 1 0 1 0
                            1 0 0 1 1
                              1 0 0 1 0
                              1 0 0 1 1
                                0 0 0 1 0
                                0 0 0 0 0
                                      1 0  ←  Remainder

Transmitted frame:  1 1 0 1 0 1 1 1 1 1 0 0 1 0  ←  Frame with four zeros appended
                                                     minus remainder
```

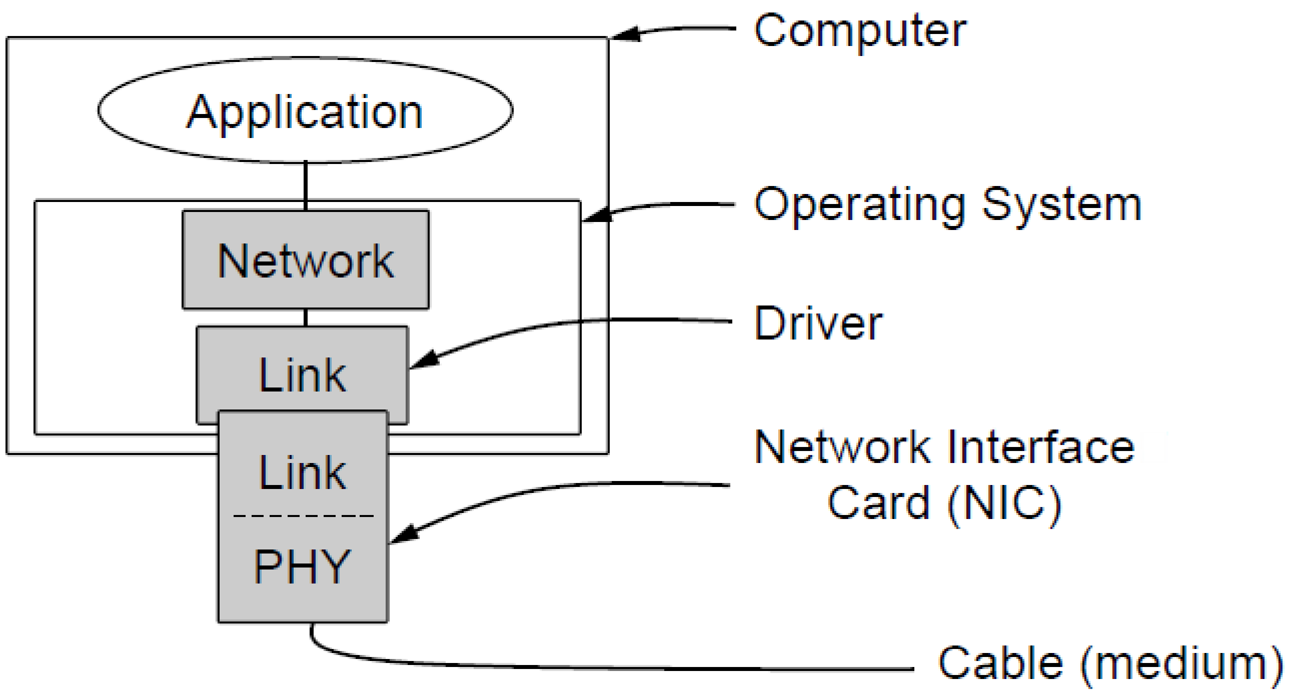
Example calculation of the CRC

ELEMENTARY DATA LINK PROTOCOLS (1)

- **Utopian Simplex Protocol**
- **Simplex Stop-and-Wait Protocol**
  - Error-Free Channel
- **Simplex Stop-and-Wait Protocol**
  - Noisy Channel

ELEMENTARY DATA LINK PROTOCOLS (2)

Implementation of the physical, data link, and network layers.

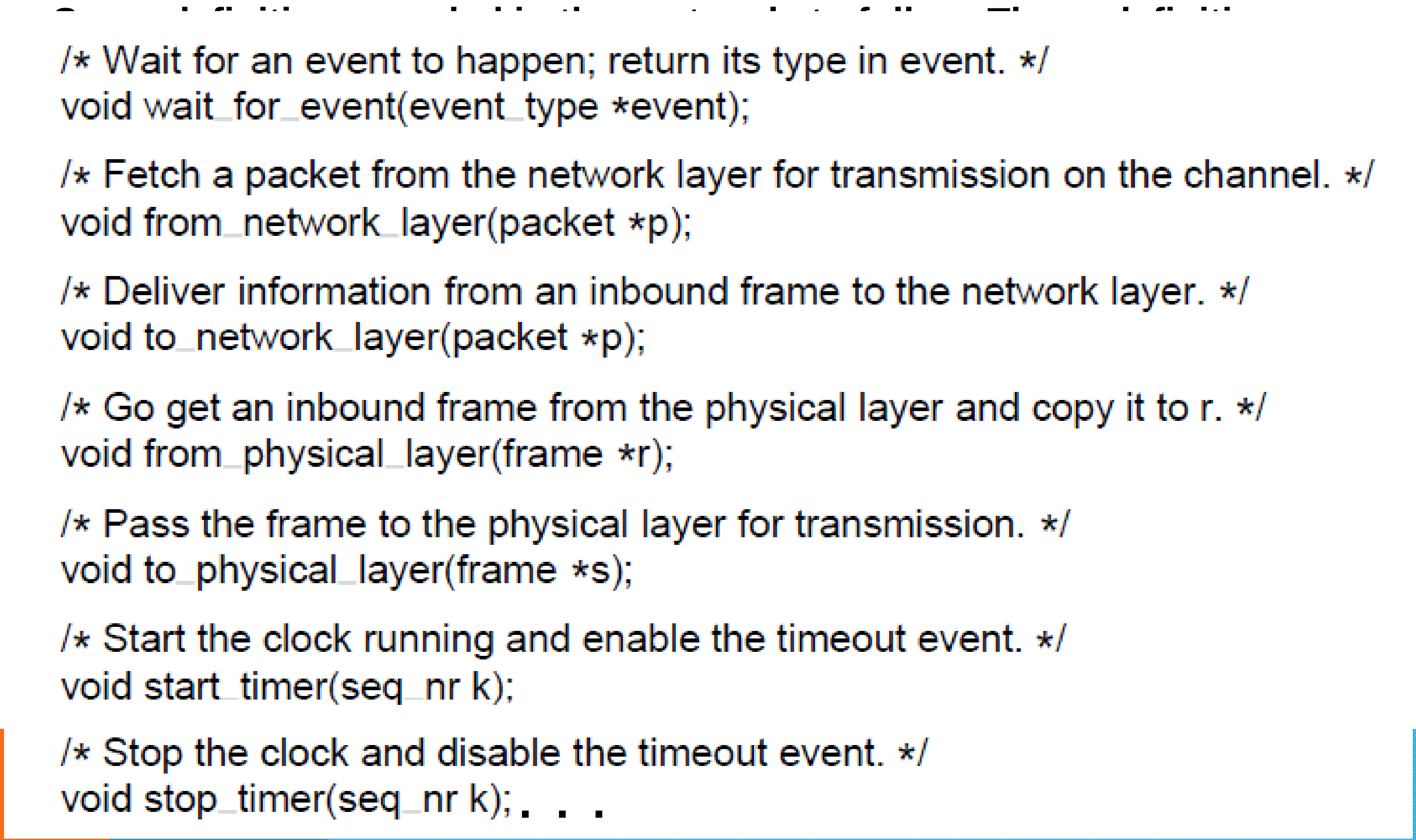


ELEMENTARY DATA LINK PROTOCOLS (3)

Some definitions needed in the protocols to follow. These definitions are located in the file protocol.h.

```
#define MAX_PKT 1024                                          /* determines packet size in bytes */

typedef enum {false, true} boolean;                           /* boolean type */
typedef unsigned int seq_nr;                                  /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;         /* packet definition */
typedef enum {data, ack, nak} frame_kind;                     /* frame_kind definition */


typedef struct {                                              /* frames are transported in this layer */
  frame_kind kind;                                            /* what kind of frame is it? */
  seq_nr seq;                                                 /* sequence number */
  seq_nr ack;                                                 /* acknowledgement number */
  packet info;                                                /* the network layer packet */
} frame;
```

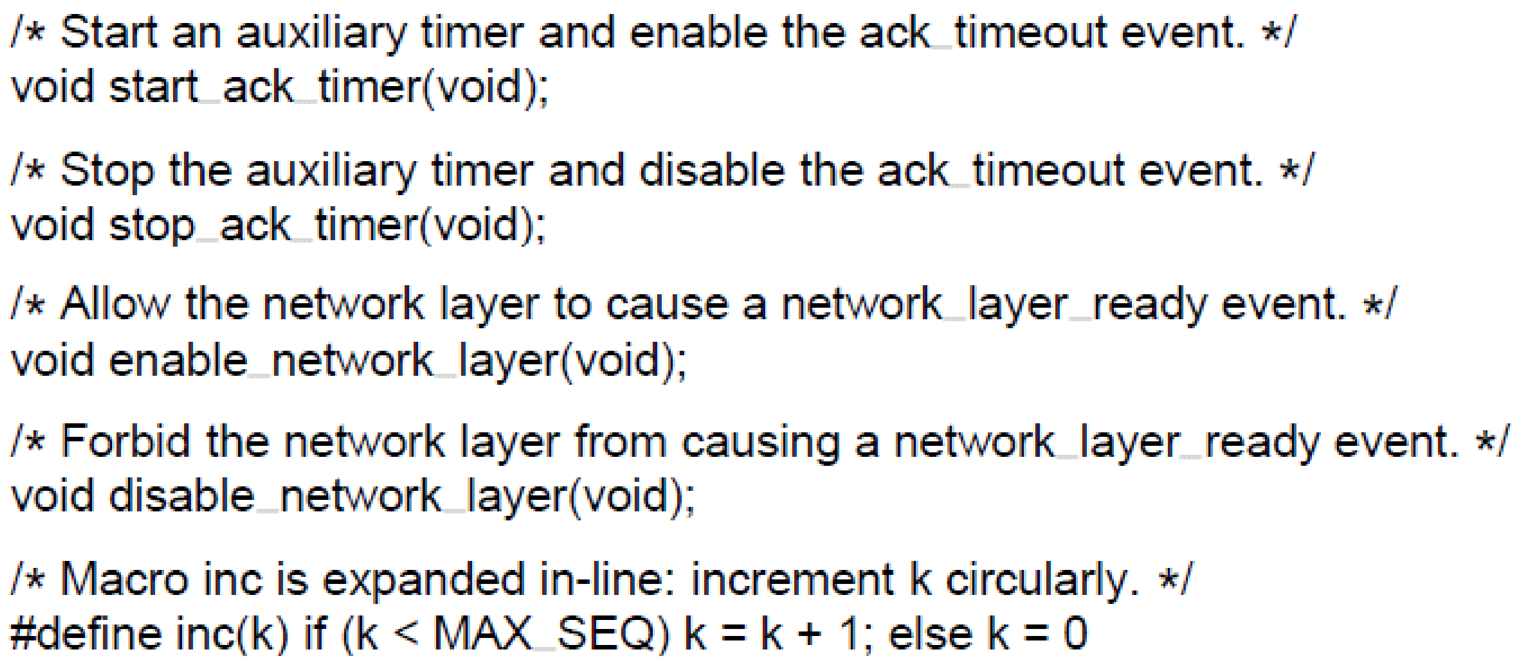
ELEMENTARY DATA LINK PROTOCOLS (4)



```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k); . . .
```

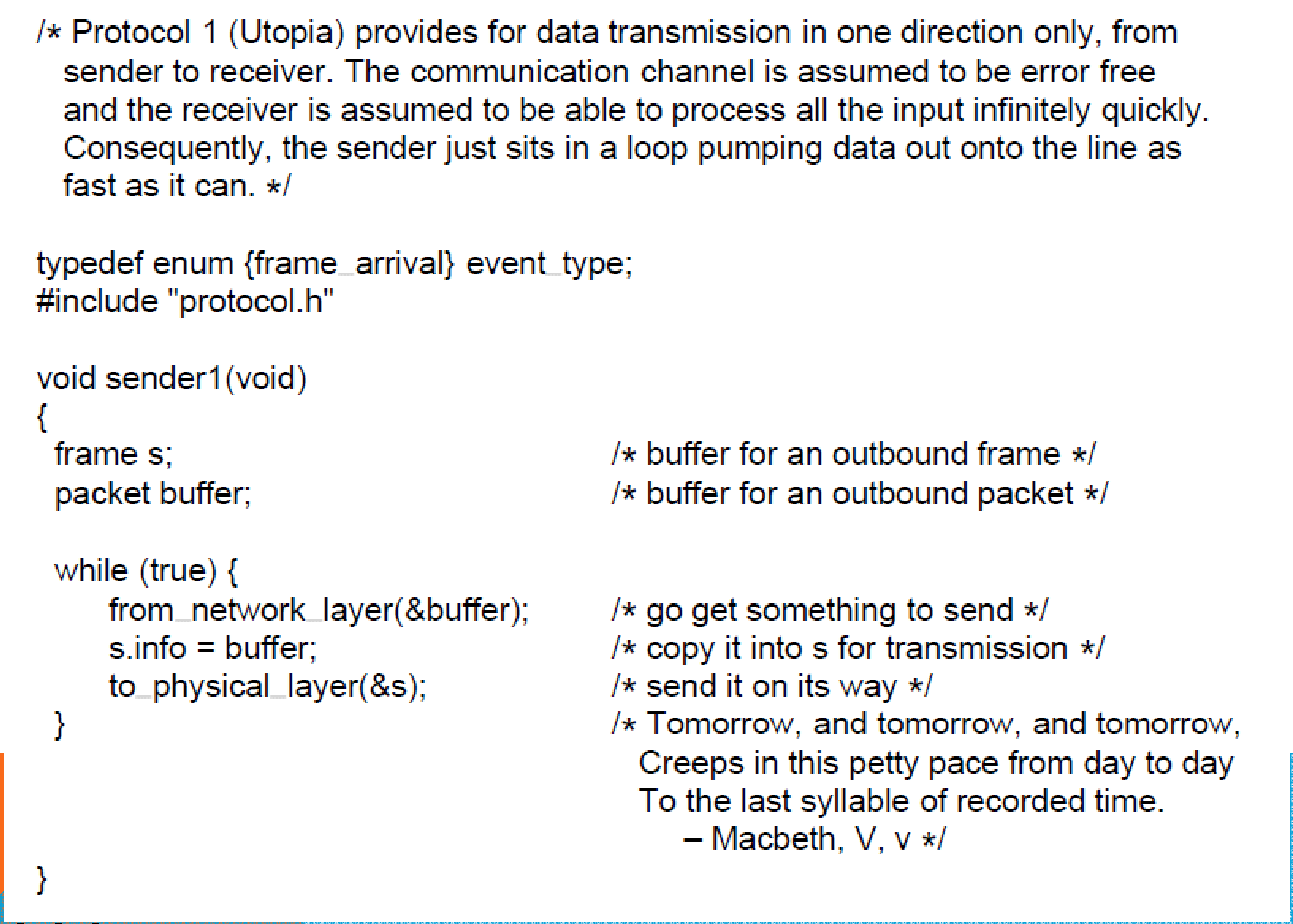
ELEMENTARY DATA LINK PROTOCOLS (5)

Some definitions needed in the protocols to follow. These definitions are located in the file protocol.h.



```
/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

UTOPIAN SIMPLEX PROTOCOL (1)

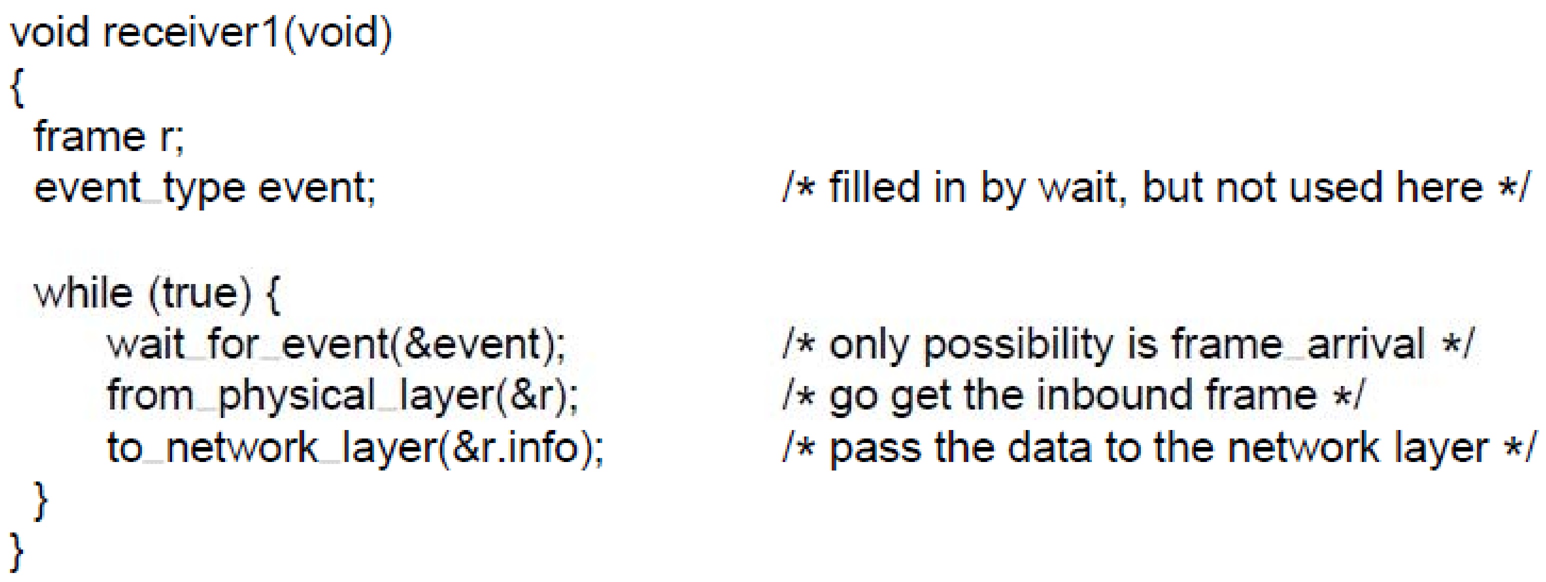


```
/* Protocol 1 (Utopia) provides for data transmission in one direction only, from
   sender to receiver. The communication channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

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 */
}
```

A utopian simplex protocol.

UTOPIAN SIMPLEX PROTOCOL (2)

A utopian simplex protocol.



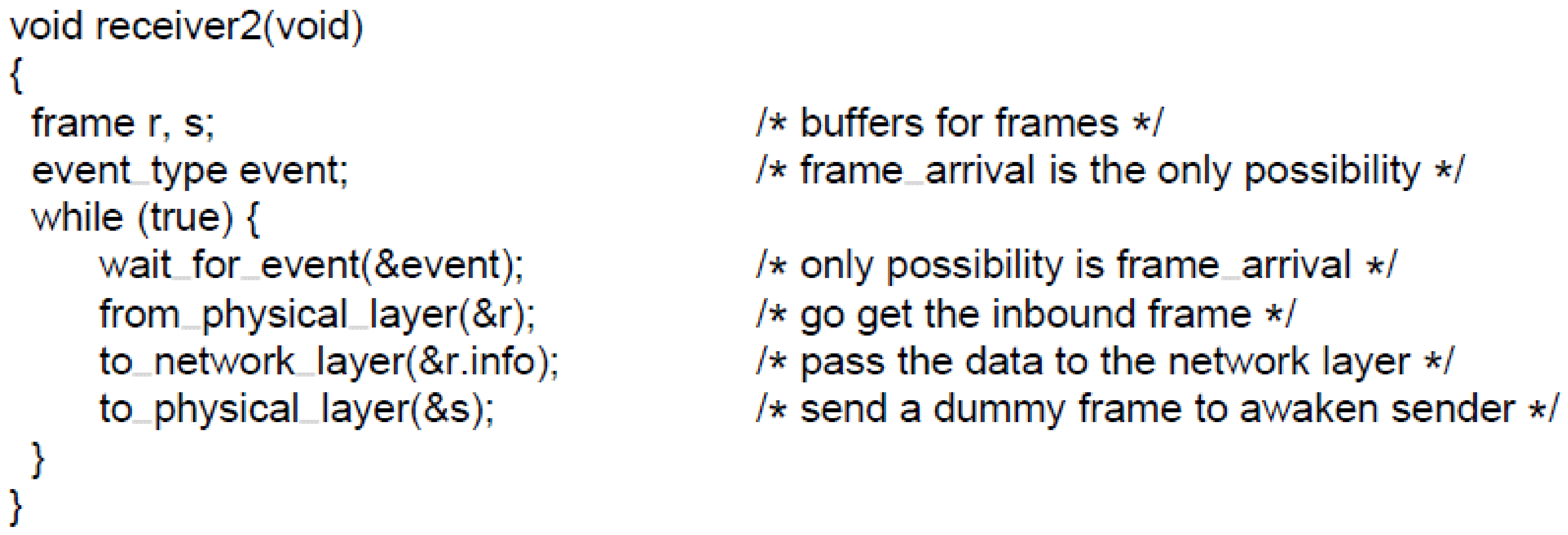
```
void receiver1(void)
{
  frame r;
  event_type event;                          /* filled in by wait, but not used here */

  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 */
  }
}
```

SIMPLEX STOP-AND-WAIT PROTOCOL FOR A NOISY CHANNEL (1)

```
/* Protocol 2 (Stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

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 */
  }
}   . . .
```

A simplex stop-and-wait protocol.

SIMPLEX STOP-AND-WAIT PROTOCOL FOR A NOISY CHANNEL (2)



```
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 */
  }
}
```

A simplex stop-and-wait protocol.