CHAPTER 3 THE DATA LINK LAYER (数据链路层)

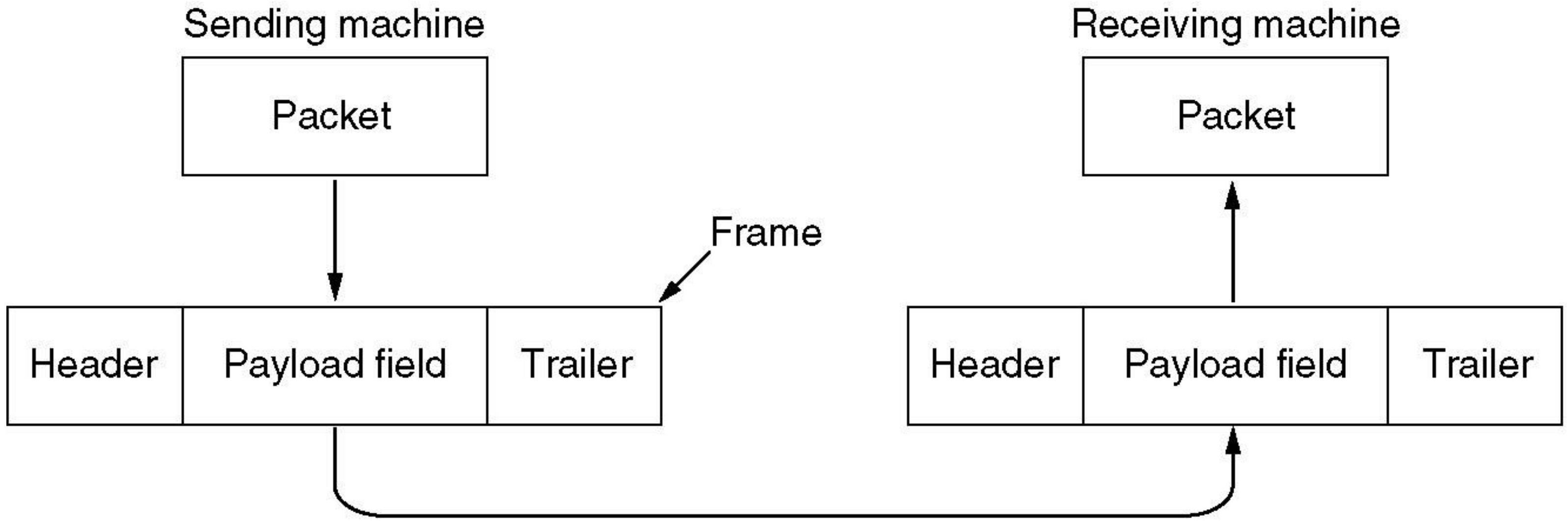
- Data Link Layer Design Issues
- Error Detection and Correction
- Elementary Data Link Protocols
- Sliding Window Protocols
- Example Data Link Protocols

DATA LINK LAYER DESIGN ISSUES

- Services provided to the network layer
- Framing
- Error control
- Flow control

Data Link Layer Design Issues

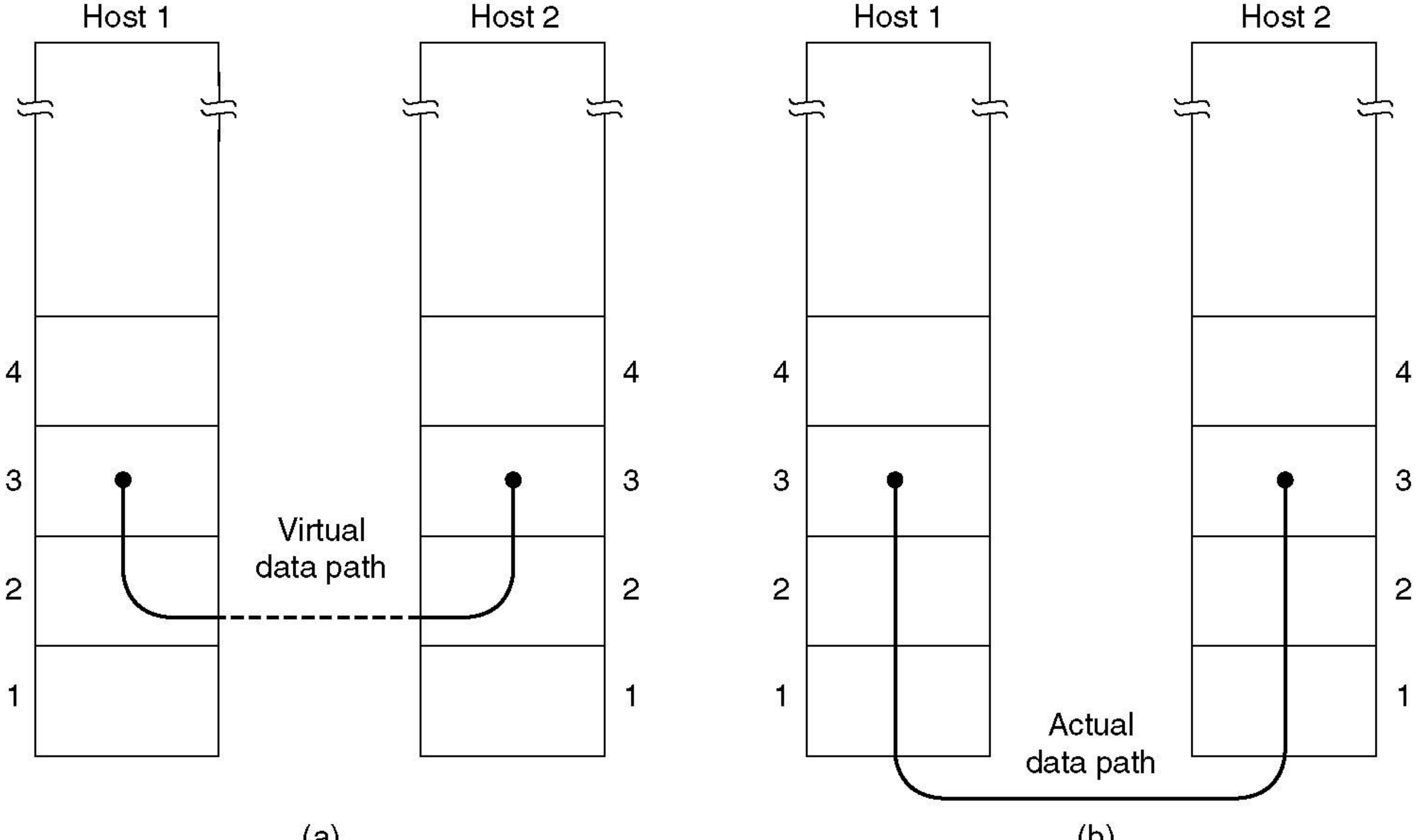
- Functions
  - Providing a well-defined service interface to the network layer;
  - Dealing with transmission errors;
  - Regulating data flow so that slow receivers are not swamped by fast senders.
- Relationship between packets and frames



Data Link Layer Design Issues

(a) Virtual communication.

(b) Actual communication.



Data Link Layer Design Issues: Services

- Three types of services:
  - Unacknowledged connectionless service,
  - Acknowledged connectionless service,
  - Acknowledged connection-oriented service.
- Unacknowledged connectionless service (无确认的无连接服务)
  - No connection is established beforehand or released afterward.
  - The source sends frames; the destination does not acknowledge. No attempt is made to recover any lost frames in the data link layer.
  - Appropriate when the error rate is very low. Appropriate for real-time traffic (Ethernet)

Data Link Layer Design Issues: Services

- Acknowledged connectionless service (有确认的无连接服务)
  - No connection
  - Each frame sent is acknowledged.
  - Useful for unreliable channels, such as wireless systems.   (WIFI)
- Acknowledged connection-oriented service (有确认的面向连接服务)
  - A connection is established before transmission.
  - Each frame sent over the connection is numbered and acknowledged.
  - Each frame is received exactly once and that all frames are received in the right order.

Data Link Layer Design Issues: Framing (成帧)

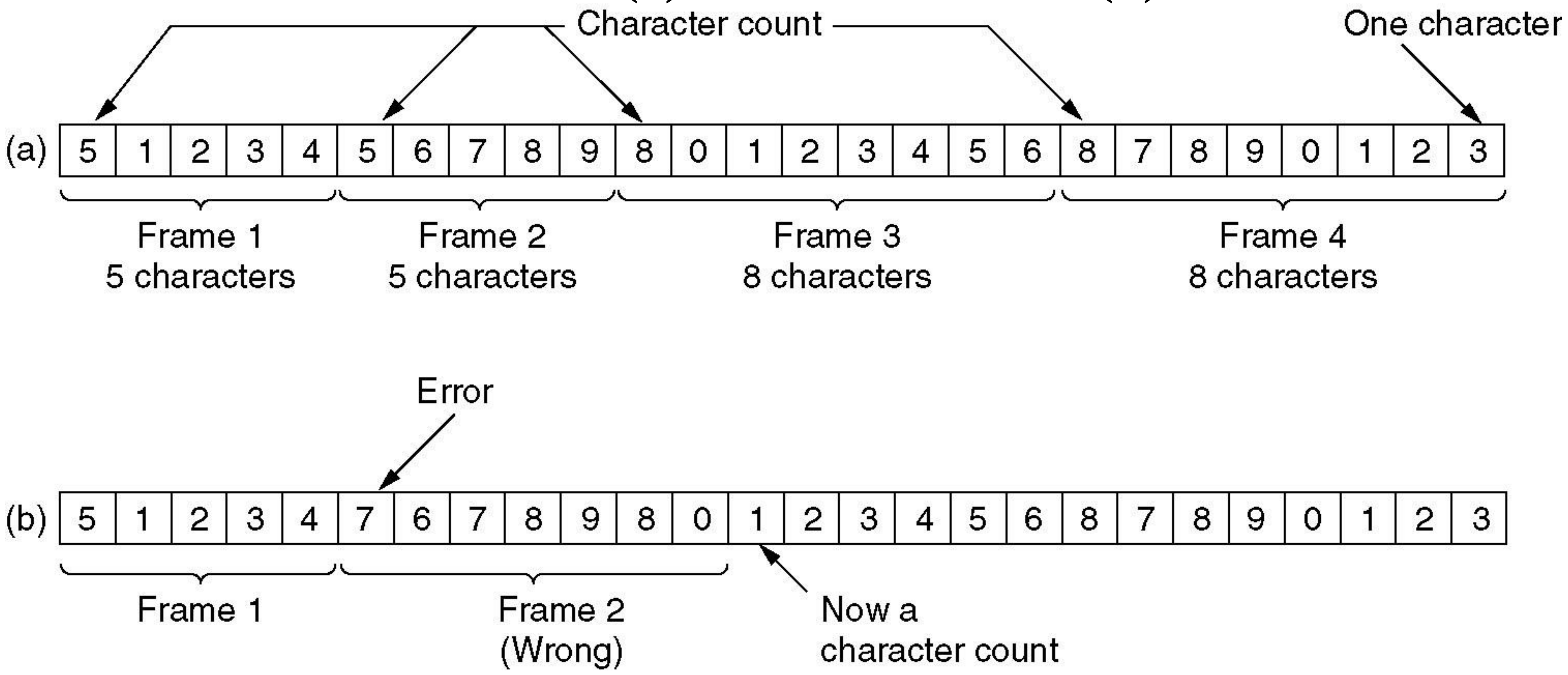
- The physical layer → **the data link layer** → the network layer
- The physical layer accepts a raw bit stream and attempts to deliver it to the destination. **This bit stream is not guaranteed to be error free.**
- The data link layer transforms an unreliable channel into a reliable one and do flow control
  - The data link layer breaks the bit stream up into discrete frames (帧) and compute the checksum for each frame.
  - When a frame arrives at the destination, the checksum is recomputed. If OK, fine; otherwise deal with errors.

Data Link Layer Design Issues: Framing

- How to break the bit stream up into frames
  - Byte count
  - Flag bytes with byte stuffing
  - Flag bits with bit stuffing
  - Physical layer encoding violation.

Data Link Layer Design Issues: Framing

- Byte count: to use a field in the header to specify the number of characters in the frame.
  - Count can be garbled by a transmission error.
  - Resynchronization problem.
  - A character stream. (a) Without errors. (b) With one error.

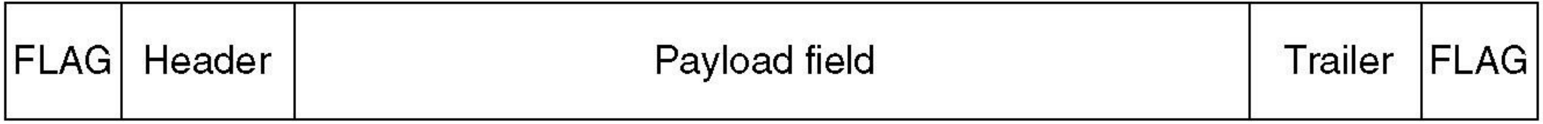


Data Link Layer Design Issues: Framing

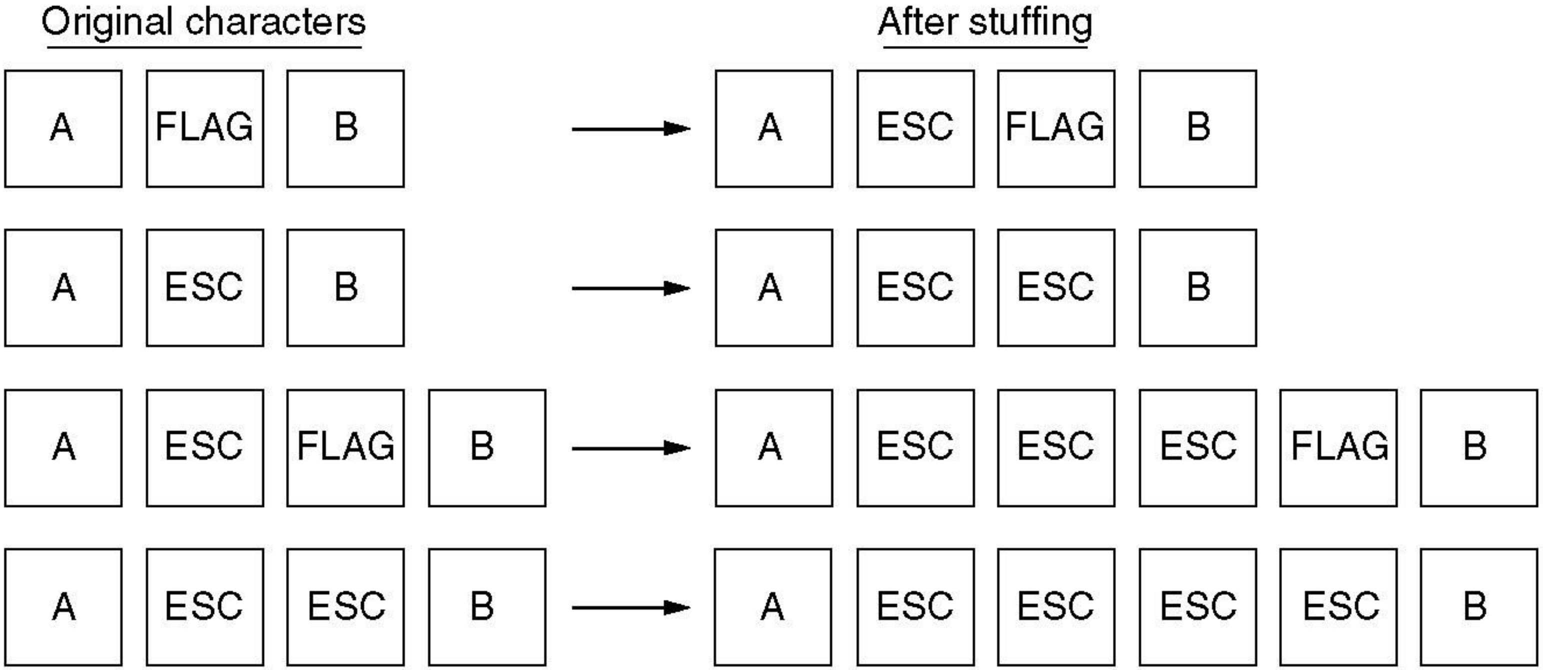
- Using flag bytes with byte stuffing

(a) A frame delimited by flag bytes.

(b) Examples of byte sequences before and after stuffing.



(a)



(b)

Data Link Layer Design Issues: Framing

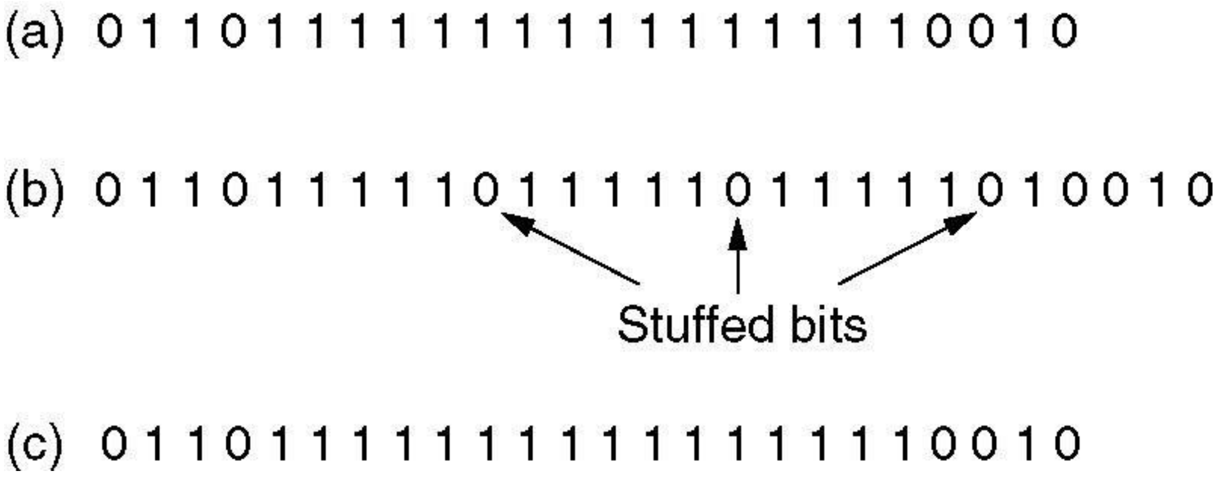
- Using flag bits with bit stuffing. Ex: flag bits 01111110

(a) The original data.

(b) The data as they appear on the line.

(c) The data as they are stored in receiver’s memory after destuffing.

(d) E.g., HDLC


(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

Data Link Layer Design Issues: Framing

- To use physical layer coding violations
  - For example, some LANs encode 1 bit of data by using 2 physical bits.
    - Normally, a 1 bit is a high-low pair and a 0 bit is low-high pair. → Every data bit has a transition in the middle, making it easy for the receiver to locate the bit boundaries.
    - The combinations high-high and low-low are not used for data but are used for delimiting frames in some protocols.

Data Link Layer Design Issues: Framing

- **To use a combination of a byte count with one of the other methods for extra safety.**
  - When a frame arrives, the count field is used to locate the end of the frame.
  - If the appropriate delimiter is present at that position and the checksum is correct, the frame is accepted as valid.
  - Otherwise, the input stream is scanned for the next delimiter.
  - Ex: preamble + byte count in Ethernet and 802.11

Data Link Layer Design Issues: Error control

- Ensure reliable frame delivery: every frame arrives without damaging, frames arrive in order.
  - To provide the sender with some feedback
    - Positive acknowledgement (ACK)
    - Negative acknowledgement (NAK)
  - To provide timeout timers
    - Resend as necessary
  - To number frames
    - To distinguish retransmissions from originals

Data Link Layer Design Issues: Flow control

What to do with a sender that systematically wants to transmit frames faster than the receiver can accept them.

- To introduce flow control to throttle the sender into sending no faster than the receiver can handle the traffic.
- Flow control protocol contains well-defined rules about when a sender may transmit the next frame.
- Two approaches
  - **Feedback-based flow control**
  - Rate-based flow control

ERROR DETECTION AND CORRECTION
差错检测与纠正

- Error-correcting codes
- Error-detecting codes

Error Detection and Correction

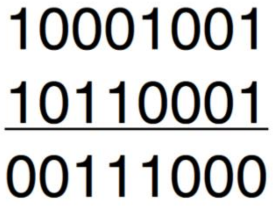
- Transmission errors are going to be a fact of life for many years to come:
  - The local loops in the PSTN (trunks and switching elements are digital)
  - Wireless communication.
- Error types:
  - isolated errors,
  - burst errors.
- Two approaches: to send the data and some extra data for detecting or correcting errors.
  - Error detection;
  - Error correction.

Error Detection and Correction: Hamming distance

- Key idea of dealing with errors: add redundancy
- m message bits => n-bit codeword (n=m+r)
- Hamming distance of 2 codewords:
  - # of bit positions in which the 2 codewords differ
- Hamming distance of complete code (all valid codewords)
  - The minimum Hamming distance of two valid codewords in the code.

Hamming distance: example

- **Ex1**: the Hamming distance of 2 codewords (10001001 and 10110001) is 3



10001001
10110001
00111000

- **Ex2**: consider a coding scheme:
  - 00 => 00000 00000
  - 01 => 00000 11111
  - 10 => 11111 00000
  - 11 => 11111 11111
  - Its Hamming distance is 5
- **Ex3**: A code with a single parity bit
  - 1011010→1011010 **0**; 1011000→1011000 **1**.
  - Hamming distance = 2 (Why?)

Error Detection and Correction: Hamming distance

The error-detection and error-correcting properties of a code depend on its Hamming distance.

To detect *d* errors:

- you need *d+1* Hamming distance code. (Because with such a code there is no way that *d* single-bit errors can change a valid code into another valid codeword).
- **Ex2**: can detect 4 bit errors.
- **Ex3**: can detect single errors.

Error Detection and Correction: Hamming distance

To use Hamming distance to correct *d* errors:

- you need *2d*+1 Hamming distance code. (Because that way the legal codewords are so far apart that even with *d* changes (A->A'), the original codeword *A* is still closer to *A'* than any other codeword *B*, so it can be uniquely determined.)
- **Ex2**:
  - – Can correct 2 errors: 00000 11111 => 0000000111, the receive know that the original code is 0000011111 (the closest valid codeword)
  - – Cannot correct 3 errors: 0000000000 => 0000000111, the error will not be corrected properly.

Error Detection and Correction:
Error correction codes

- To design a code with m message bits and r check bits that will allow all single errors to be corrected:
  - Each valid codeword of the 2^m legal messages has n illegal codewords at a distance 1 from it. Thus each of the 2^m legal messages requires $n+1$ bit patterns dedicated to it.
  - The total number of bit patterns is 2^n:

    $(m + r + 1)\ 2^m <= 2^n = 2^{m+r}$

    ➔ $(m+r+1) <= 2^r$
  - Given m, this puts a lower limit on the number of check bits needed to correct single errors.
  - This theoretical limit can be achieved using a method due to Hamming (1950).

Error Detection and Correction: Error correction codes

- Hamming code for single error:
  - The bits of the codeword are numbered consecutively, starting with bit 1 at the left end.
  - The bits that are powers of 2 (1,2,4,8, etc) are check bits. The rest (3, 5,6,7,etc) are filled up with the *m* data bits.
  - Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations.
  - To see which check bits the data bit in position *k* contributes to, rewrite *k* as a sum of powers of *2*. For example, *11=1+2+8* and *29=1+4+8+16*. **A bit is checked by just those check bits occurring in its expansion** (e.g., bit 11 is checked by bits 1, 2, and 8).

Error Detection and Correction: Error correction codes

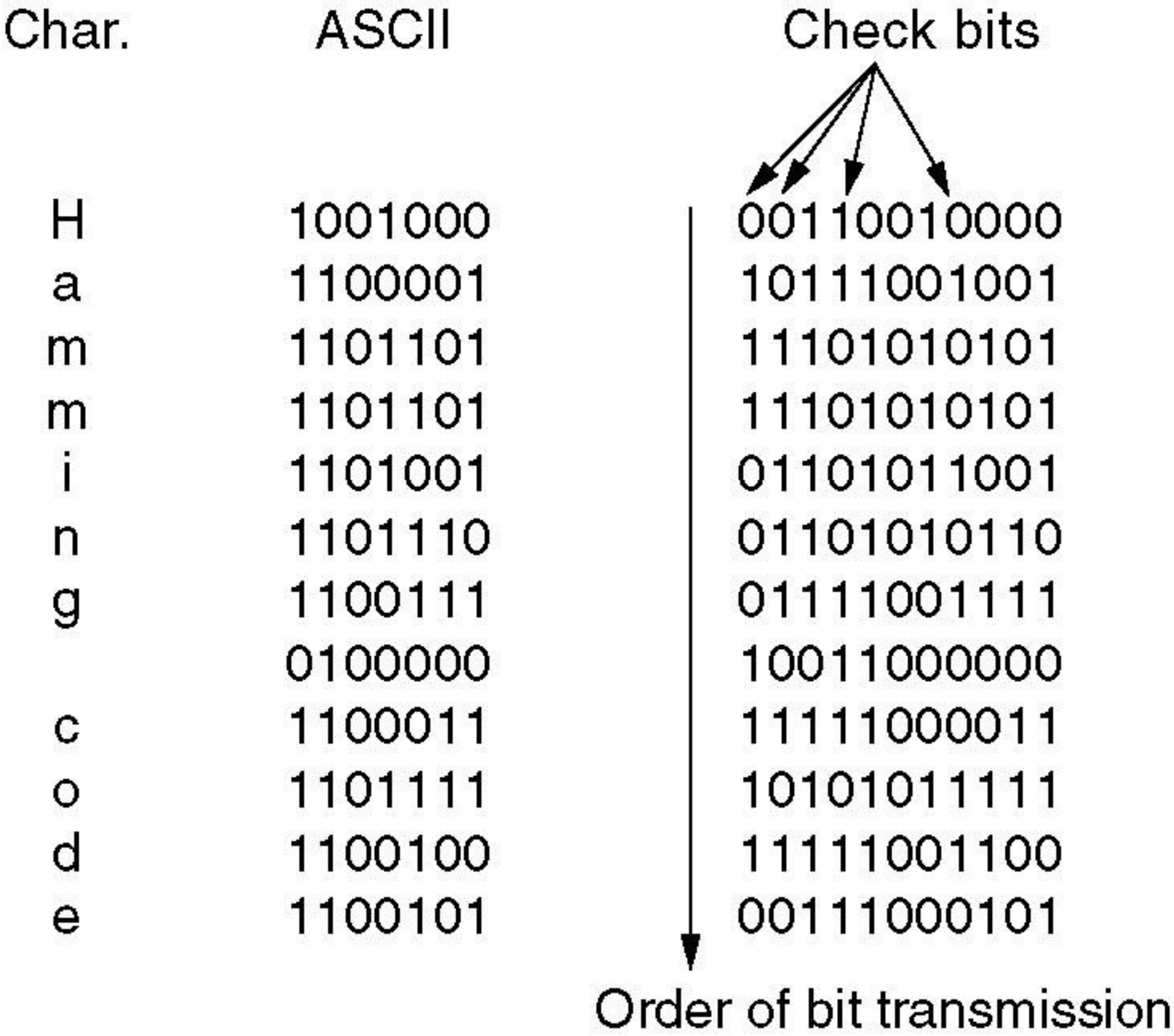
- Hamming codes for single error:
  - When a codeword arrives, the receiver initializes a counter to zero.
  - It then examines each check bit, *k (k=1,2,4,8,...)* to see if it has the correct parity. If not, it adds *k* to the counter.
  - If the counter is zero after all the check bits have been examined, the codeword is accepted as valid. If the counter is nonzero, it contains the position of the incorrect bit.
  - For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only one checked by bits 1, 2, 8.

Error Detection and Correction: Error correction codes

- Hamming codes can be used to correct burst errors indirectly.
  - A sequence of *k* consecutive codewords are arranged as a matrix, one codeword per row.
  - Transmit the matrix by one column at a time.
  - When the frame arrives at the receiver, the matrix is reconstructed, one column at a time.
  - If a burst error of length *k* occurs, at most 1 bit in each of the *k* codewords will have been affected, but the Hamming code can correct one error per codeword, so the entire block can be restored.
  - This method uses *kr* check bits to make blocks of *km* data bits immune to a single burst error of length *k* or less.

Error Detection and Correction: Error correction codes

- Hamming codes to correct burst errors



Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

Error Detection and Correction: Error correction codes

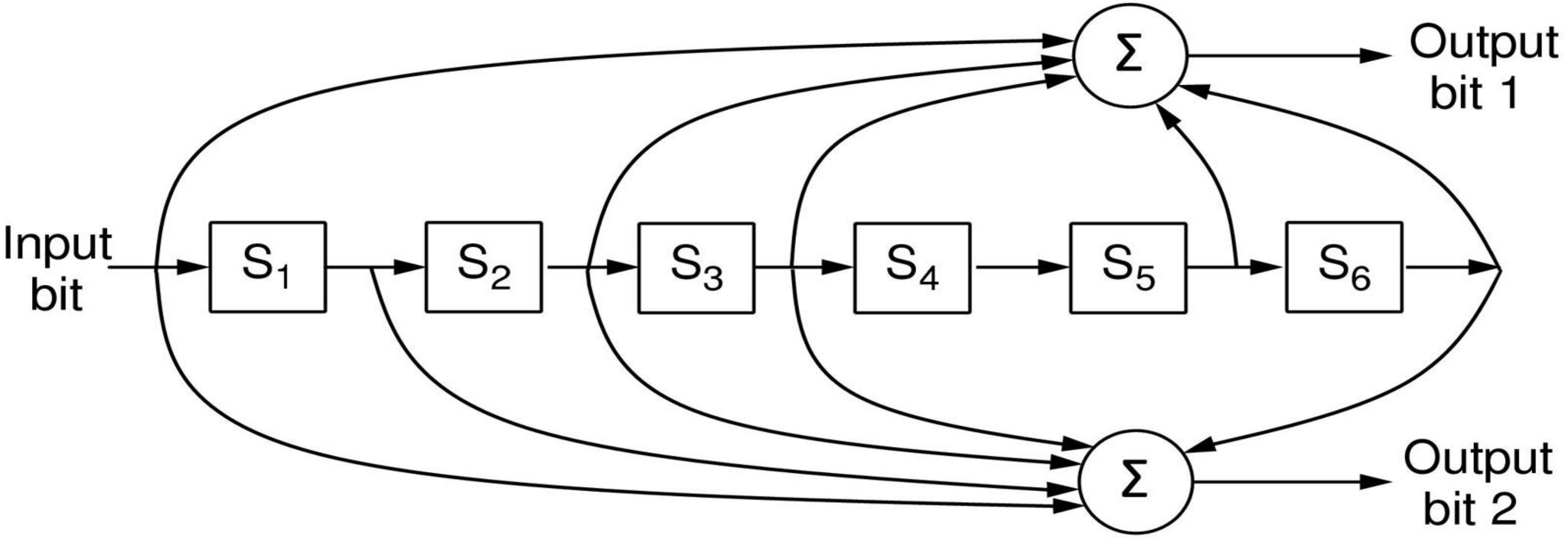
- Other codes
  - Binary Convolutional Codes
    - NASA，GSM，802.11
    - NB-IoT downlink
  - Reed-Solomon Codes
  - Low-Density Parity Check Codes (LDPC)

Convolutional Code

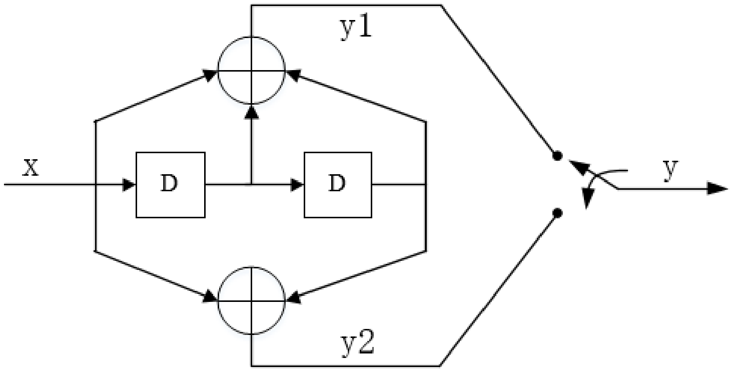
- An encoder processes a sequence of input bits and generates a sequence of output bits.
- The output depends on the current and previous input bits. That is, the encoder has memory. #previous input bits is called constraint length.
- Convolutional code is characterized by its coding rate and constraint length

Convolutional code example

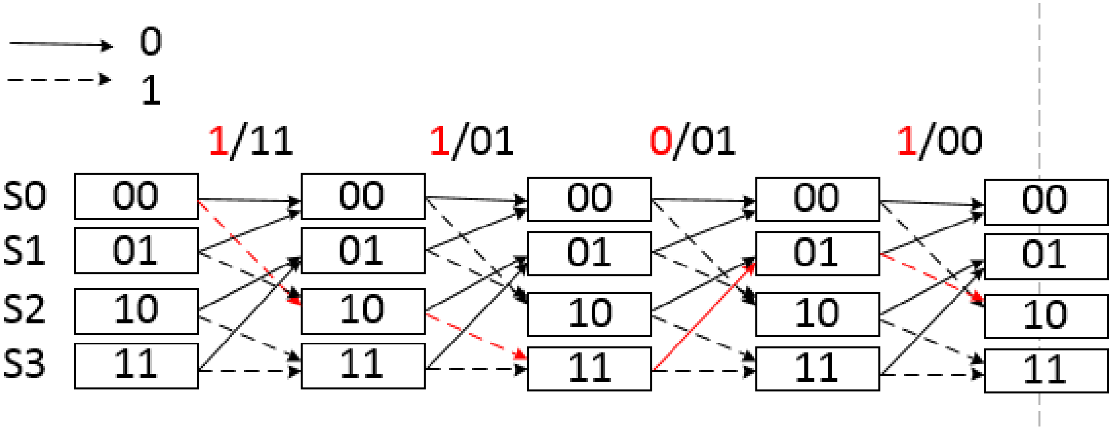
- Constraint length k=#registers+1=7, r=1/2
- Input bits: 111; Initial internal state $s_1..s_6$=000000
- Take the 1st input bit: output=11, state=100000
- Take the 2nd input bit: output=10, state=110000



Trellis diagram

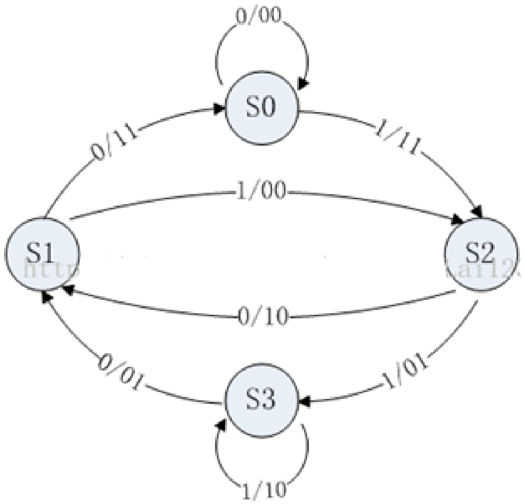


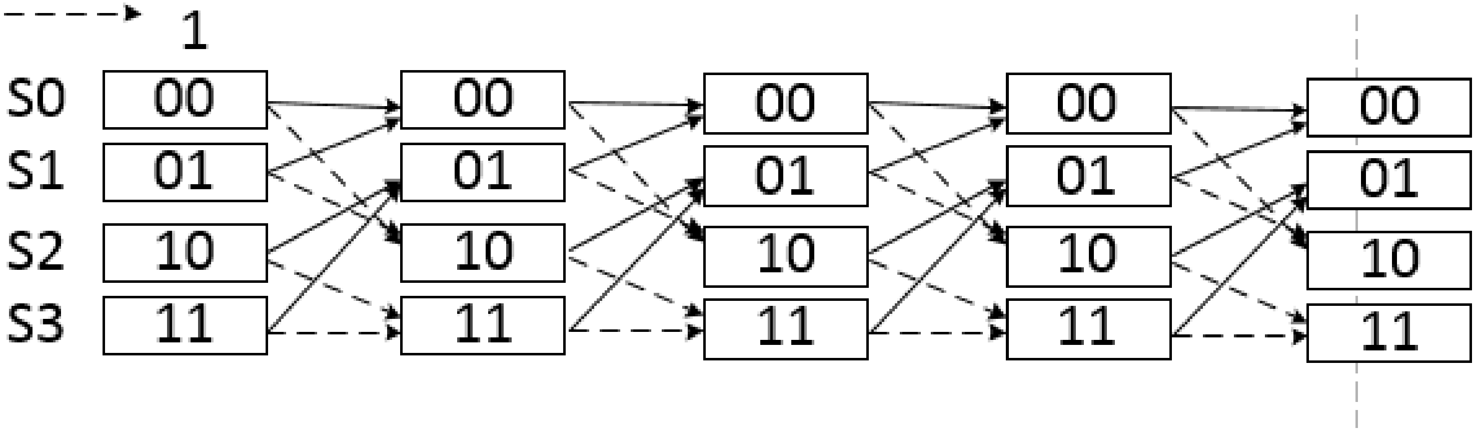
- 输入数据：u = [1 1 0 1 1]
  - 初始状态: 00
- 编码：V = [11 01 01 00 01]
- 正确接收：R = [11 01 01 00 01]
- 错误接收：R = [11 01 01 10 00]



思考题

- 给定接收数据: R = [00 10 01 01]
  - 初始状态: 00
- 问：原始数据是多少





Error Detection and Correction:
Error detection codes

- **Error detection is less expensive than error correction.**
  - Given a channel with the error rate as 10^{-6} per bit. Let the block size be 1000 bits.
  - **Error correcting**: To provide single error correcting, 10 check bits are need; a megabit of data would require 10,000 check bits.
  - **Error detection + retransmission**: To merely detect a block with a single 1-bit error, one parity bit per block will suffice. Once every 1000 blocks an extra block (1001 bits) will have to be transmitted. The total overhead for the error detection + retransmission method is 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

Error Detection and Correction:
Error detection codes

- **Parity**
- **Checksum**
- **CRC**

Error Detection and Correction: Error detection codes

- To detect single error by using parity bit
  - Append parity bit to the block to detect the possible single error
- To detect a single burst of length *k* by using parity bit.
  - To treat the data block as a matrix *n bits wide and k bits high*. To append every row with a parity bit;
  - To transmit the block one column at a time;
  - To check the parity bit.

Error Detection and Correction: Error detection codes (Checksum)

- Checksum treats data as N-bit words and adds N check bits that are the modulo 2^N sum of the words
  - Ex: Internet 16-bit 1 complement checksum
- Properties:
  - Improved error detection over parity bits
  - Vulnerable to systematic errors, e.g., added zeros

Internet Checksum Example

- ❐ Note
  - ❍ When adding numbers, a carryout from the most significant bit needs to be added to the result
- ❐ Example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Error Detection and Correction: Error detection codes

- **Polynomial code or CRC (Cyclic Redundancy Check)**
  - To treat bit strings as representations of polynomials with coefficients of 0 and 1 only.
  - A *k*-bit frame is regarded as the coefficient list for a polynomial with *k* terms, ranging from x^{k-1} *to* x^0.
  - For example, 110001 → $x^5 + x^4 + x^0$.
  - Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. There are no carries for addition or borrows for subtraction. Both addition and subtraction are **identical** to exclusive OR.

Error Detection and Correction: Error detection codes

- Polynomial code or CRC (Cyclic Redundancy Check)
  - The sender and receiver agree upon a generator polynomial, *G(x)*.
  - Let *M(x*) be the polynomial corresponding to some frame with *m* bits.
  - The polynomial *T(X)* represented by the checksummed frame is divisible by *G(x)*.
  - When the receiver gets *the checksummed frame*, it tries dividing it by *G(x)*. If there is a remainder, there has been a transmission error.

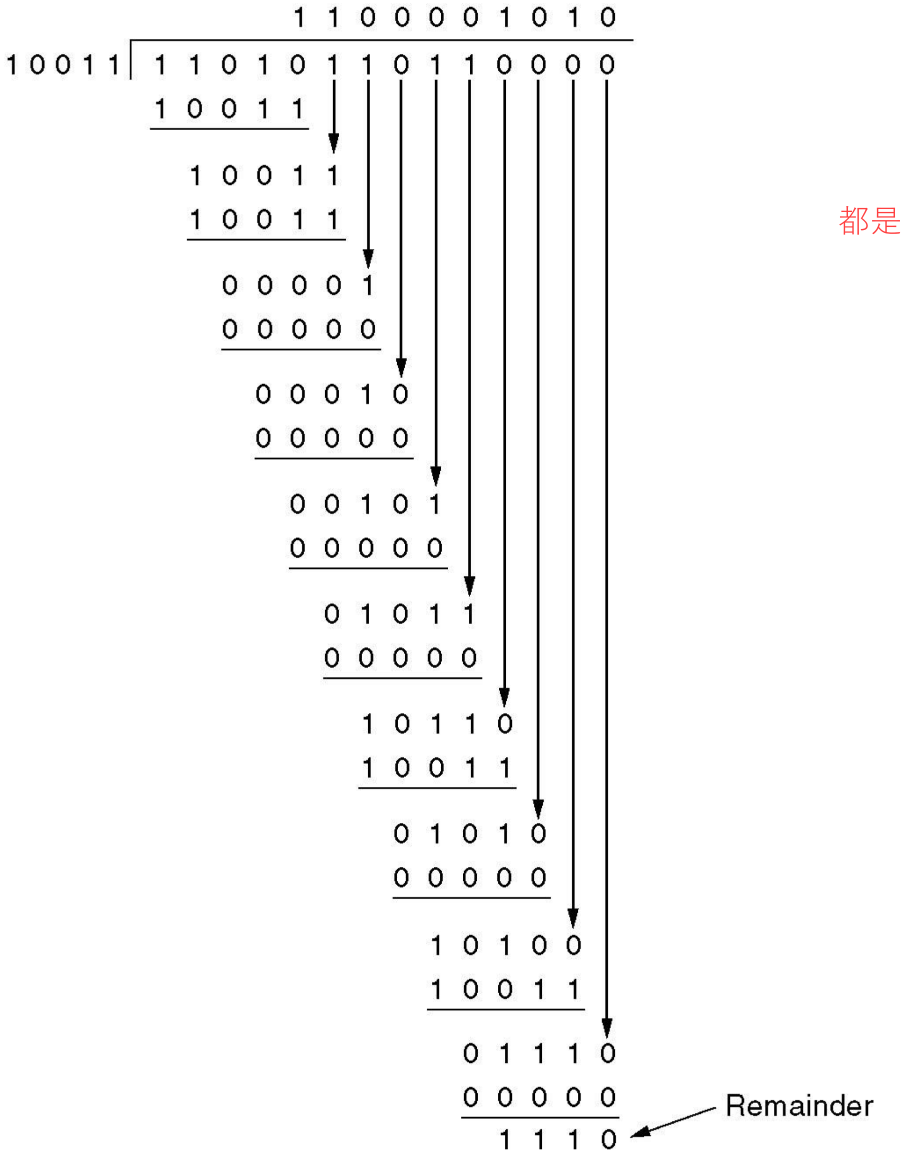
Error Detection and Correction: Error detection codes

- Polynomial code or CRC (Cyclic Redundancy Check)
  - Let *r* be the degree of *G(x)*.
  - Divide the bit string corresponding to *G(x)* into the bit string corresponding to $M(x)x^r$, using modulo 2 division, i.e. $M(x)x^r / G(x)$
  - Subtract the remainder from the bit string corresponding to $M(x)x^r$ using modulo 2 subtraction. ***The result is the checksummed frame to be transmitted.*** Call its polynomial *T(x)*.
  - Clearly, *T(x)* is divisible (modulo 2) by *G(x)*.

Frame : 1 1 0 1 0 1 1 0 1 1

Generator: 1 0 0 1 1

Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



都是异或操作, 非常高效.

Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

Error Detection and Correction: Error detection codes

- Polynomial code or CRC (Cyclic Redundancy Check)
  - Imagine that a transmission error occurs, so that instead of the bit string for *T(x)* arriving, *T(x) + E(x)* arrives.
  - Each *1* bit in *E(x)* corresponds to a bit that has been inverted.
    - If there are *k* *1* bits in *E(x)*, *k* single-bit errors have occurred.
    - A single burst error is characterized by an initial *1*, a mixture of *0*s and *1*s, and a final *1*, with all other bits being *0*.

Error Detection and Correction:
Error detection codes

- Polynomial code or CRC

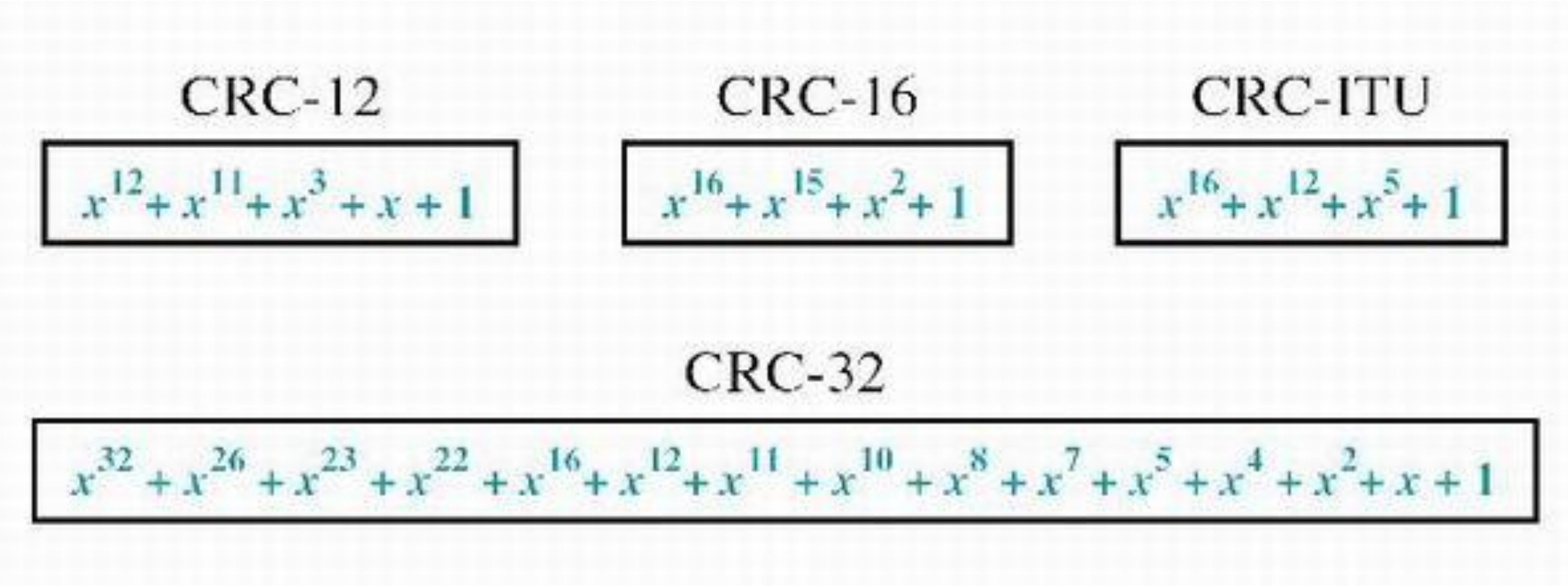
有余数就是错的

  - $[T(X)+E(X)] / G(X) = E(X)/G(X)$
  - Those errors that happen to correspond to polynomials containing G(X) as a factor will slip by; all other errors will be caught
  - All single errors can be detected as long as G(X) has more than one term
  - All double errors can be detected as long as G(X) does not divide x^k+1 for some k. $E(X) = x^i+x^j = x^j(x^{i-j} +1)$. **Ex**: $x^{15}+x^{14}+1$ will not divide x^k+1 for any value of k below 32768

Error Detection and Correction: Error detection codes

- Polynomial code or CRC
  - All errors with an odd # of bits can be detected as long as G(X) has x+1 as its factor
  - A polynomial code with r check bits will detect all burst errors of length <= r.
  - If the burst length is r+1, the remainder of the division by G(x) will be zero iff the burst is identical to G(x). This probability is $1/2^{r-1}$
  - When an error burst longer than r+1 bits occurs, or several shorter bursts occur, the probability of a bad frame getting through unnoticed is $1/2^{r}$

Error Detection and Correction: Error detection codes



CRC-12

$$x^{12} + x^{11} + x^{3} + x + 1$$

CRC-16

$$x^{16} + x^{15} + x^{2} + 1$$

CRC-ITU

$$x^{16} + x^{12} + x^{5} + 1$$

CRC-32

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

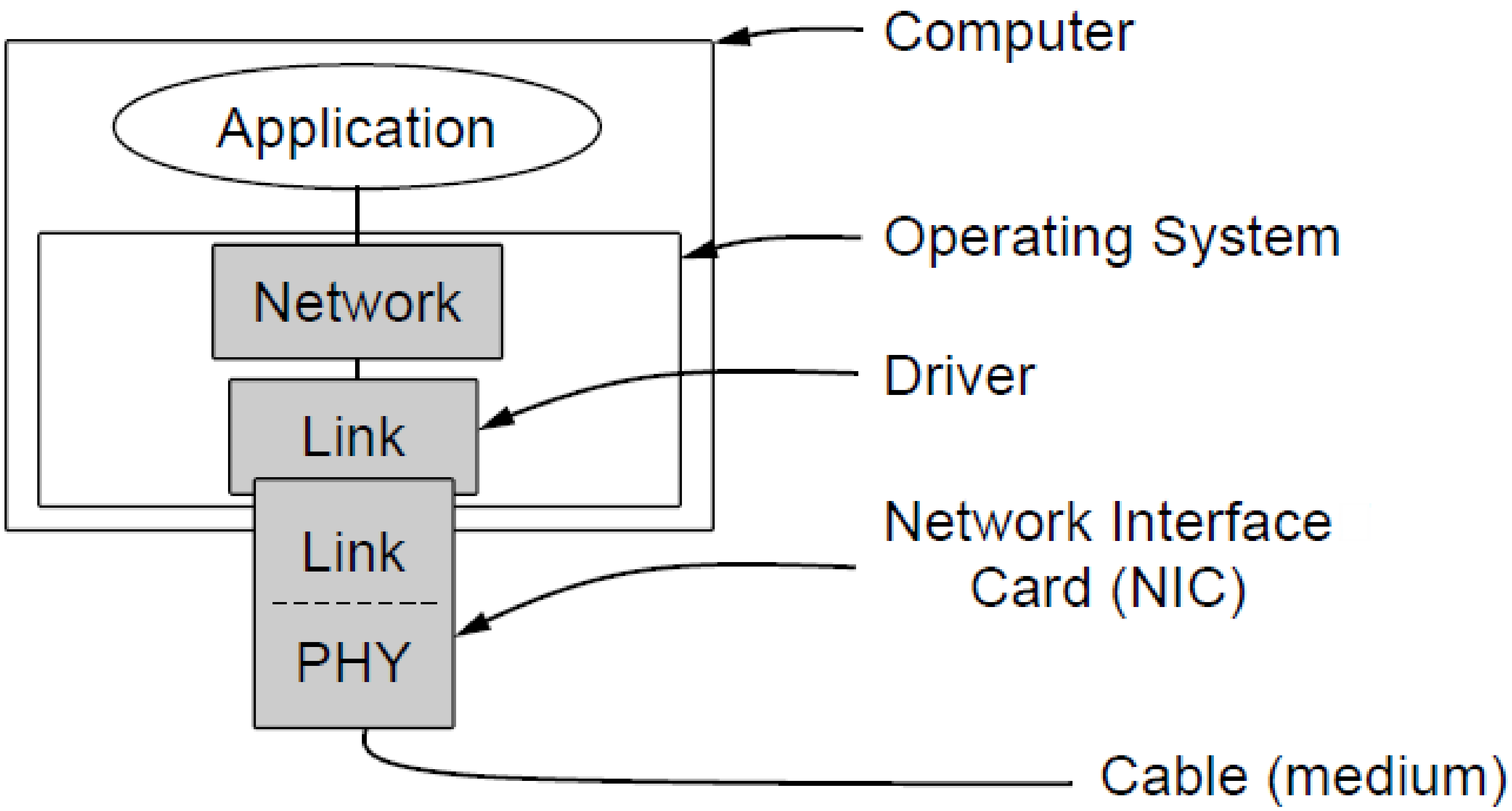
ELEMENTARY DATA LINK PROTOCOLS

- A Utopian Simplex Protocol

- A Simplex Stop-and-Wait Protocol

- A Simplex Protocol for a Noisy Channel

Elementary Data Link Protocols: Possible Implementation



Elementary Data Link Protocols:

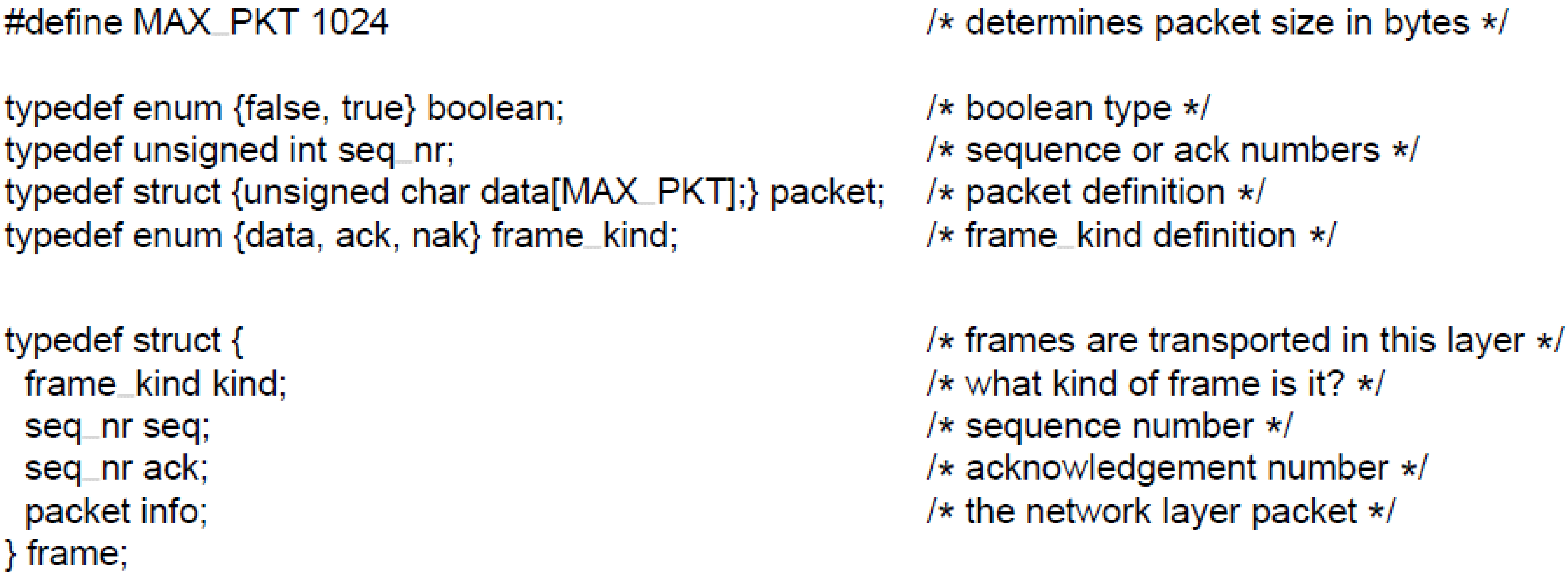
Some Assumptions

- The physical layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth.
- Machine A wants to send a long stream of data to machine B using a reliable, connection-oriented service.
- Machines do not crash.

Elementary Data Link Protocols: Sending and Receiving Frames

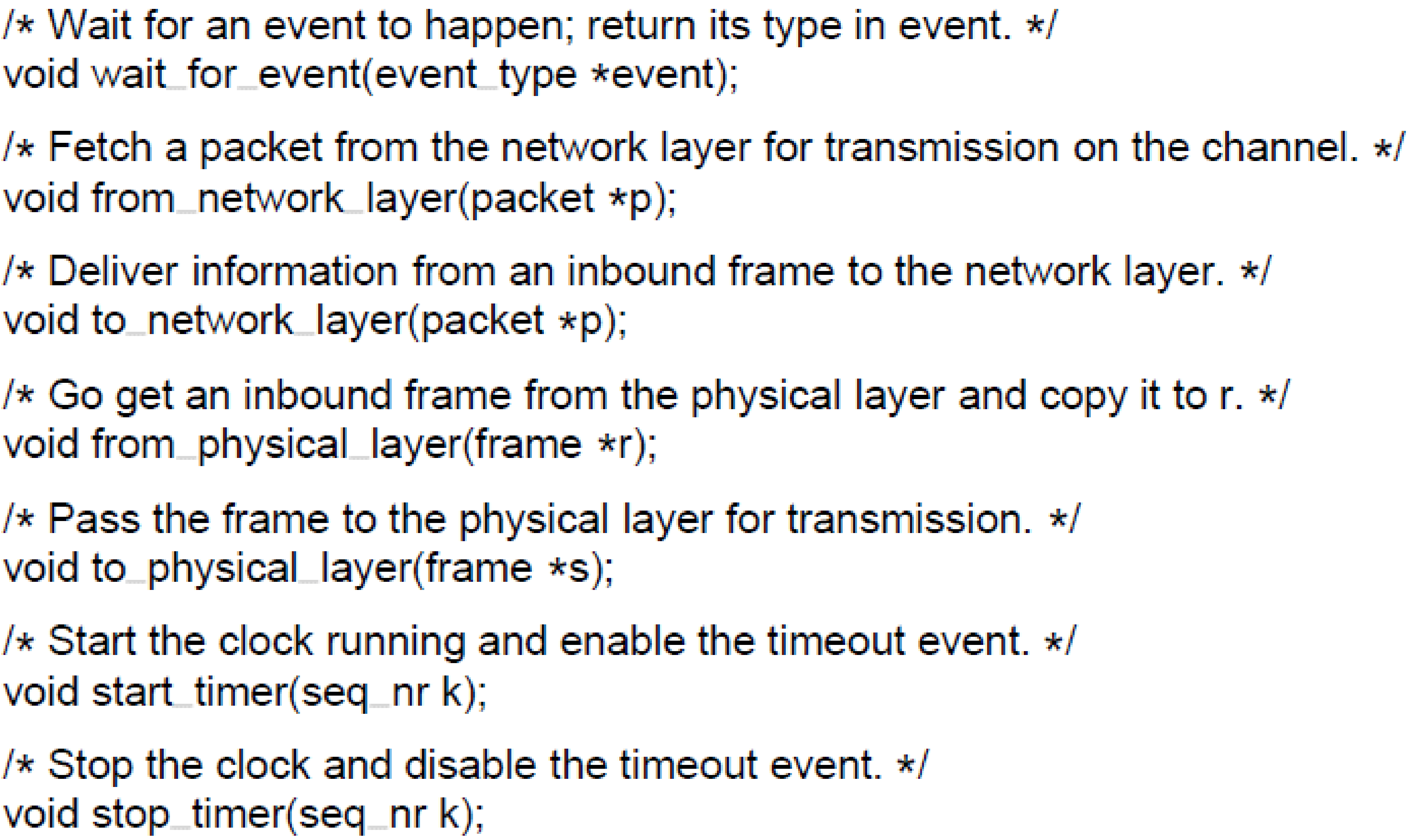
- To send a frame
  - To accept a packet passed from the network layer
  - To encapsulate the packet in a frame by adding data link header and trailer to it
  - To transmit it to the data link layer on the other machine
- To receive a frame
  - Initially, the receiver has just wait_for_event()
  - When something has happened, the procedure returns
  - If a frame arrives, the HW computes the checksum
  - If OK, it checks the control information in the header
  - If everything is all right, it passes the packet to the network layer

Elementary Data Link Protocols：protocol.h



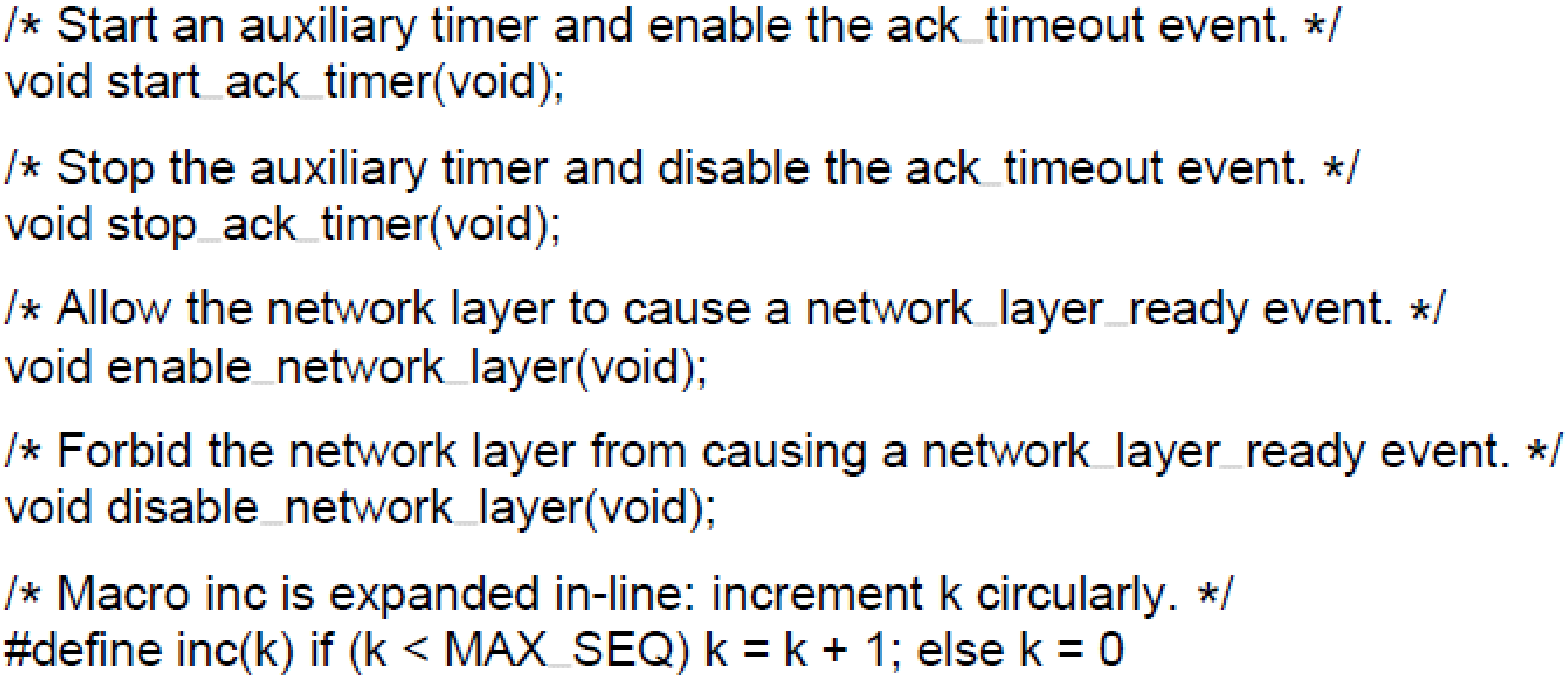
```
#define MAX_PKT 1024                                    /* determines packet size in bytes */

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: protocol.h



```
/* 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: 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
```

Elementary Data Link Protocols: Protocol 1

A Utopian simplex protocol

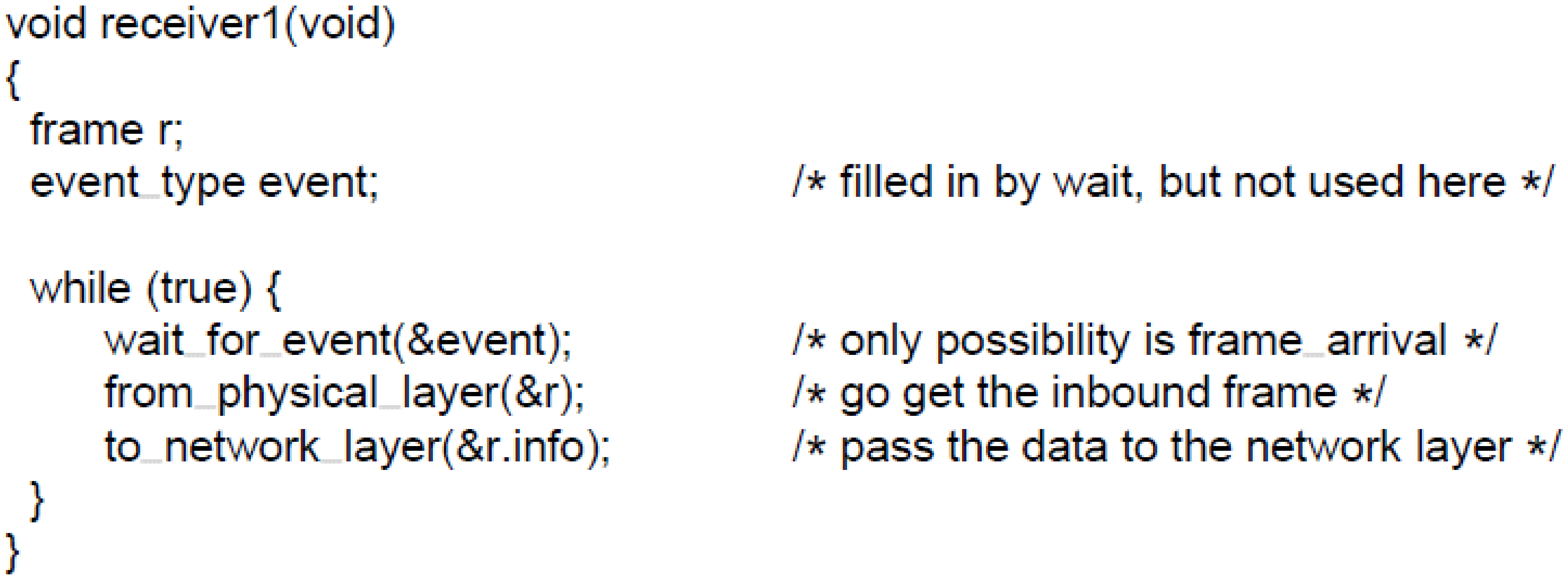
- Data are transmitted in one direction only.

- The communication channel never damages or loses frames.

- Both the transmitting and receiving network layers are always ready.
- Processing time can be ignored.
- Infinite buffer space is available.

```
/* 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 */
}
```

Elementary Data Link Protocols: Protocol 1



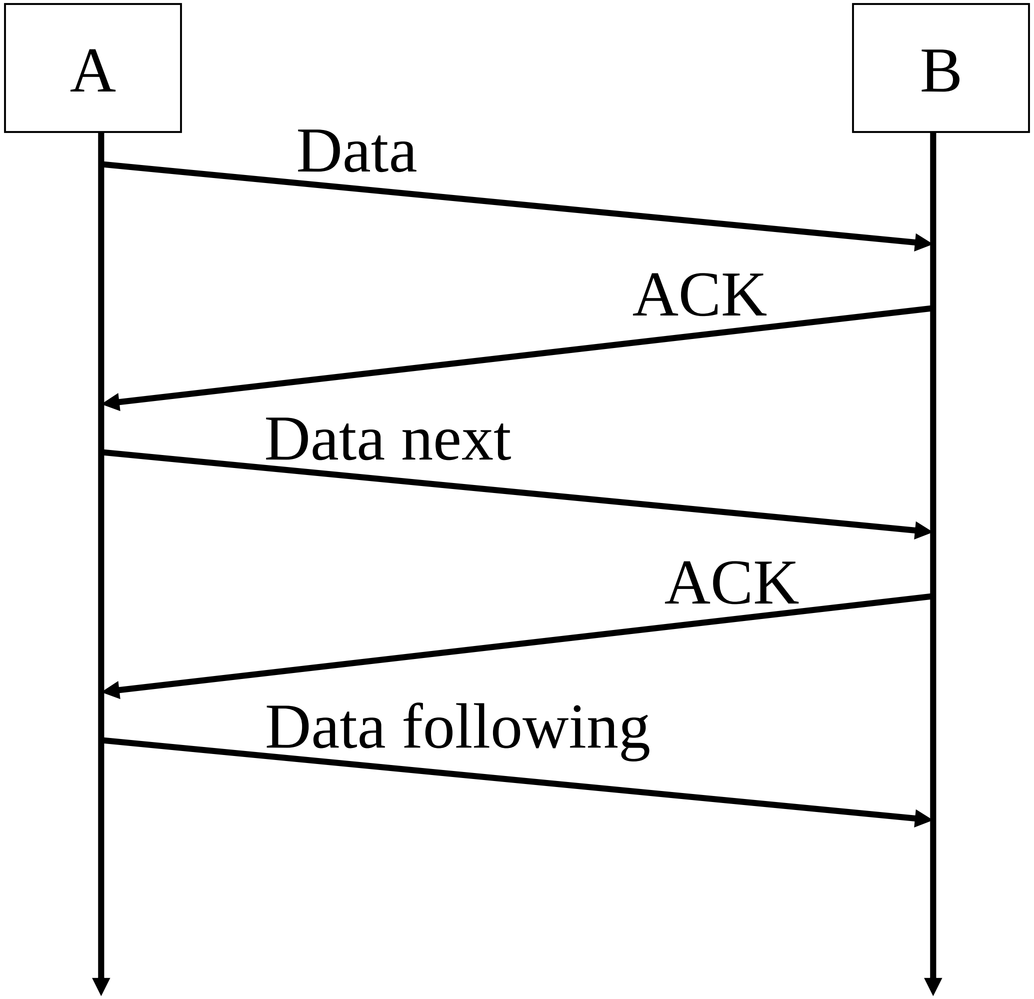
```
void receiver1(void)
{
  frame r;
  event_type event;                          /* filled in by wait, but not used here */

  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 */
  }
}
```

Elementary Data Link Protocols: Protocol 2

A simplex stop-and-wait protocol

- Data traffic is still simplex.
- The communication channel is assumed to **be error free**.
- The sender is always ready. The receiver is NOT always ready or the receiver has limited buffer space.
  - The sender simply inserts a delay into protocol 1 to slow it down sufficiently to keep from swamping the receiver. → low utilization of bandwidth.
  - The receiver provides feedback to the sender, permitting the sender to transmit the next frame.
- Protocol (p2) ensures sender can’t outpace receiver:
  - Receiver returns a dummy frame (ack) when ready
  - Only one frame out at a time – called stop-and-wait
  - We added flow control!

Stop-and-Wait(Normal)



A

B

Data

ACK

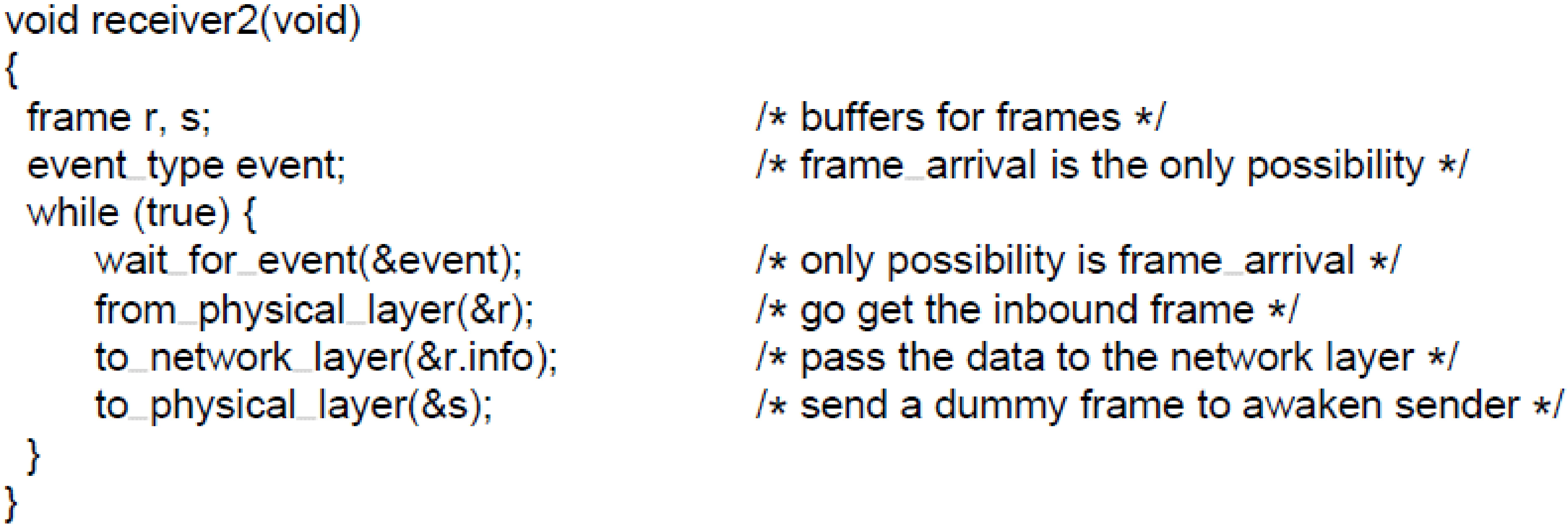
Data next

ACK

Data following

```c
/* 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 */
  }
}
```

Elementary Data Link Protocols: Protocol 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 */
  }
}
```

Elementary Data Link Protocols: Protocol 3

A simplex protocol for a noisy channel

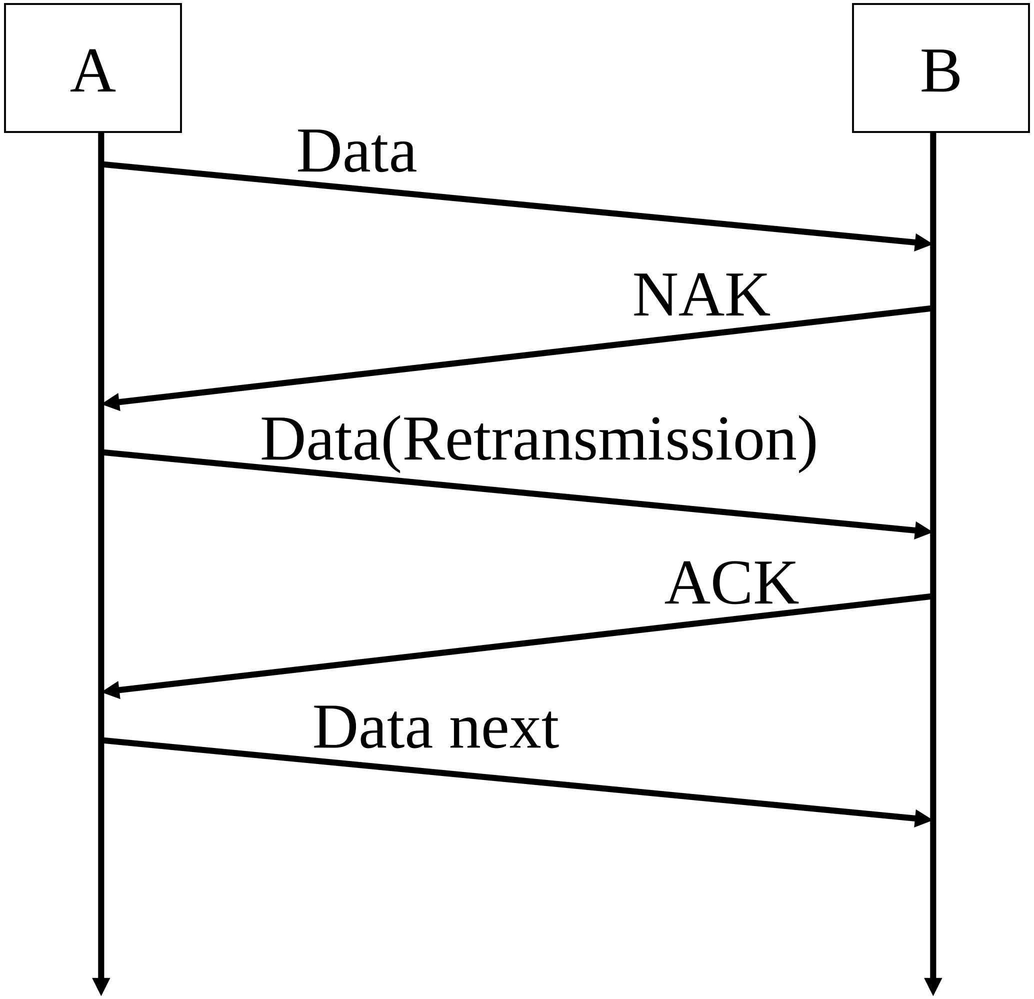
- Data traffic is still simplex.
- The communication channel is NOT free of errors.
- The receiver is NOT always ready.

- The possible solutions:
  - Protocol 2 + timer → duplicate packets
  - Protocol 2 + timer + to number the frame
- Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called PAR (Positive Acknowledgement with Retransmission) or **ARQ** (Automatic Repeat reQuest)

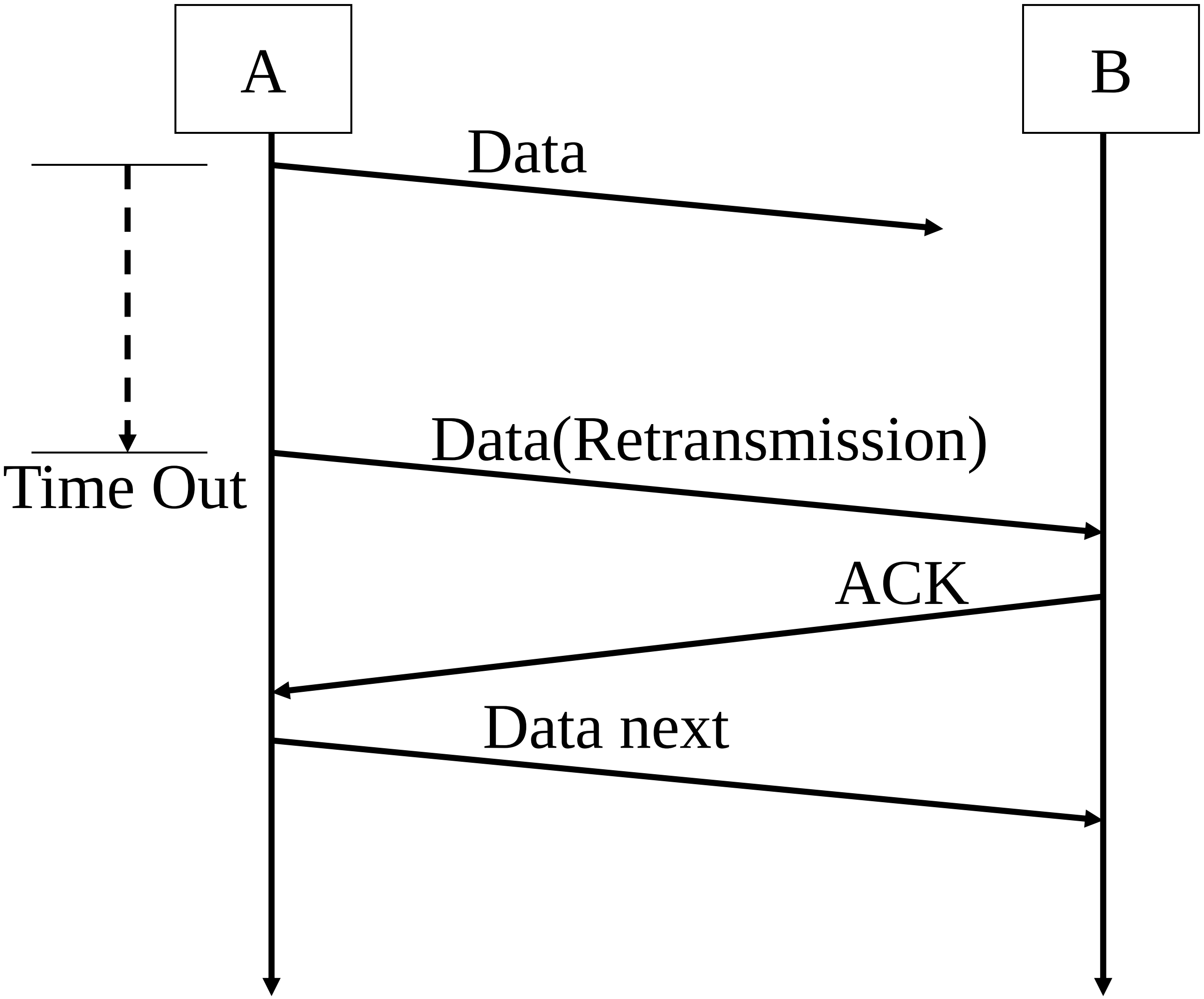
Elementary Data Link Protocols: Protocol 3

- ARQ (Automatic Repeat reQuest) adds error control
  - Receiver acks frames that are correctly delivered
  - Sender sets timer and resends frame if no ack
- For correctness, frames and acks must be numbered
  - Else receiver can’t tell retransmission (due to lost ack or early timer) from new frame
  - For stop-and-wait, 2 numbers (1 bit) are sufficient

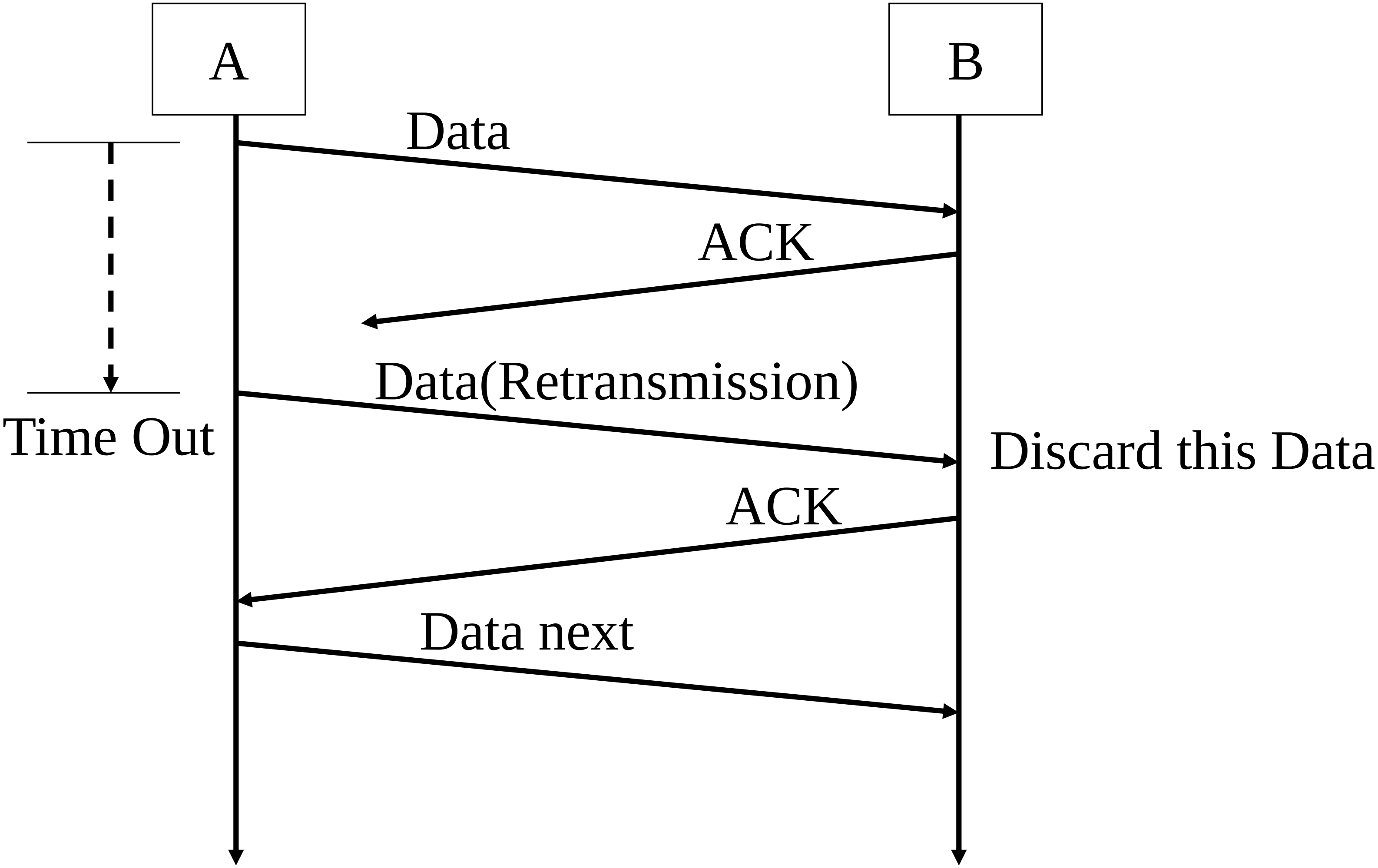
Stop-and-Wait(Data Error)



Stop-and-Wait(Data Lost)



Stop-and-Wait(ACK Lost)



Sender loop (p3):

Send frame (or retransmission)

Set timer for retransmission

Wait for ack or timeout

If a good ack then set up for the next frame to send (else the old frame will be retransmitted)

```
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);
   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) {
                  stop_timer(s.ack);
                  from_network_layer(&buffer);
                  inc(next_frame_to_send);
             }
        }
   }
}
```

Receiver loop (p3):

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

  frame_expected = 0;
  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);
      }
  }
}
```

Wait for a frame

If it's new then take it and advance expected frame

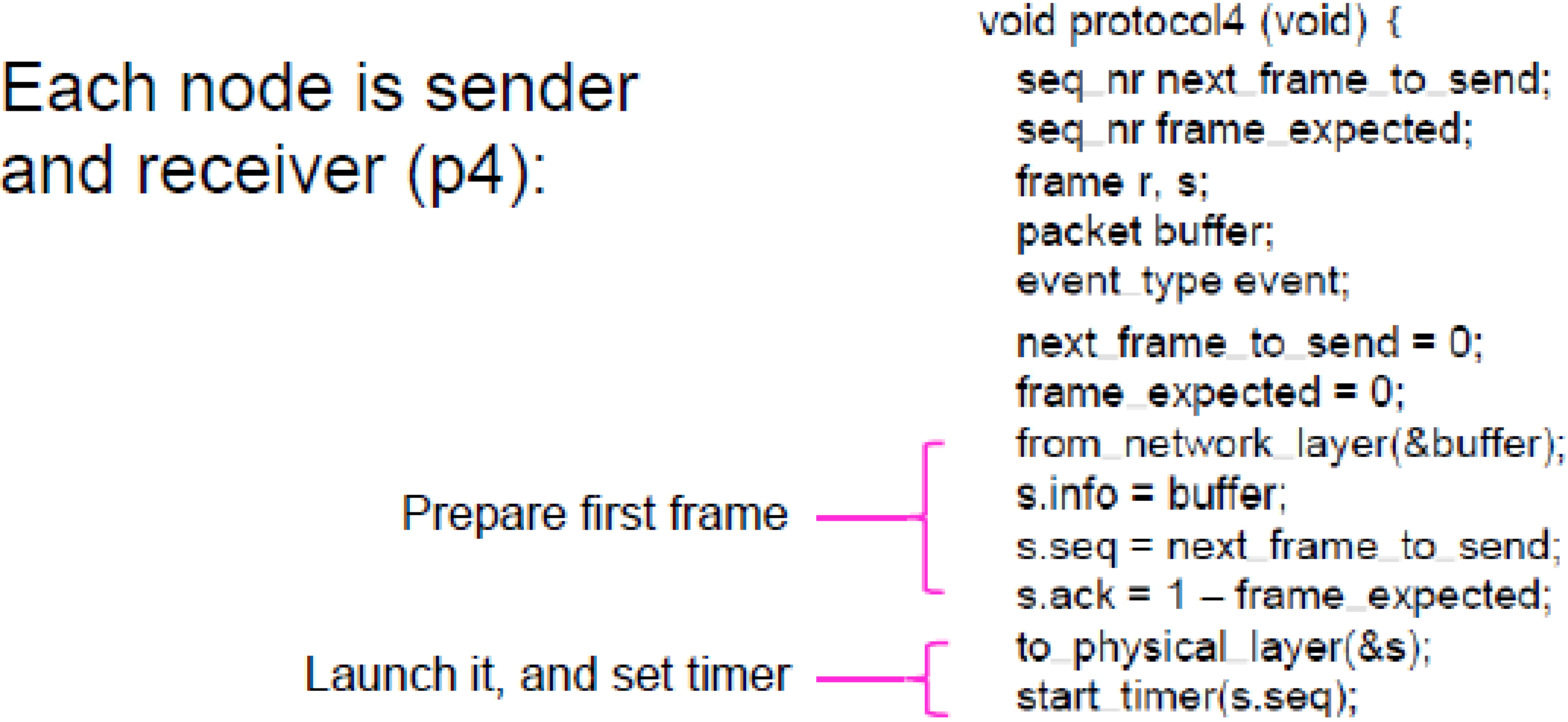
Ack current frame

Protocol 4

- Protocol 4
  - Duplex
  - Stop and wait

- No combination of lost frames or premature timeouts can cause the protocol to deliver duplicate packets to either network layer, or skip a packet, or to get into a deadlock.

Sliding Window Protocols: Protocol 4



Each node is sender and receiver (p4):

```
void protocol4 (void) {
    seq_nr next_frame_to_send;
    seq_nr frame_expected;
    frame r, s;
    packet buffer;
    event_type event;
    next_frame_to_send = 0;
    frame_expected = 0;
    from_network_layer(&buffer);
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 – frame_expected;
    to_physical_layer(&s);
    start_timer(s.seq);
```

Wait for frame or timeout

If a frame with new data then deliver it

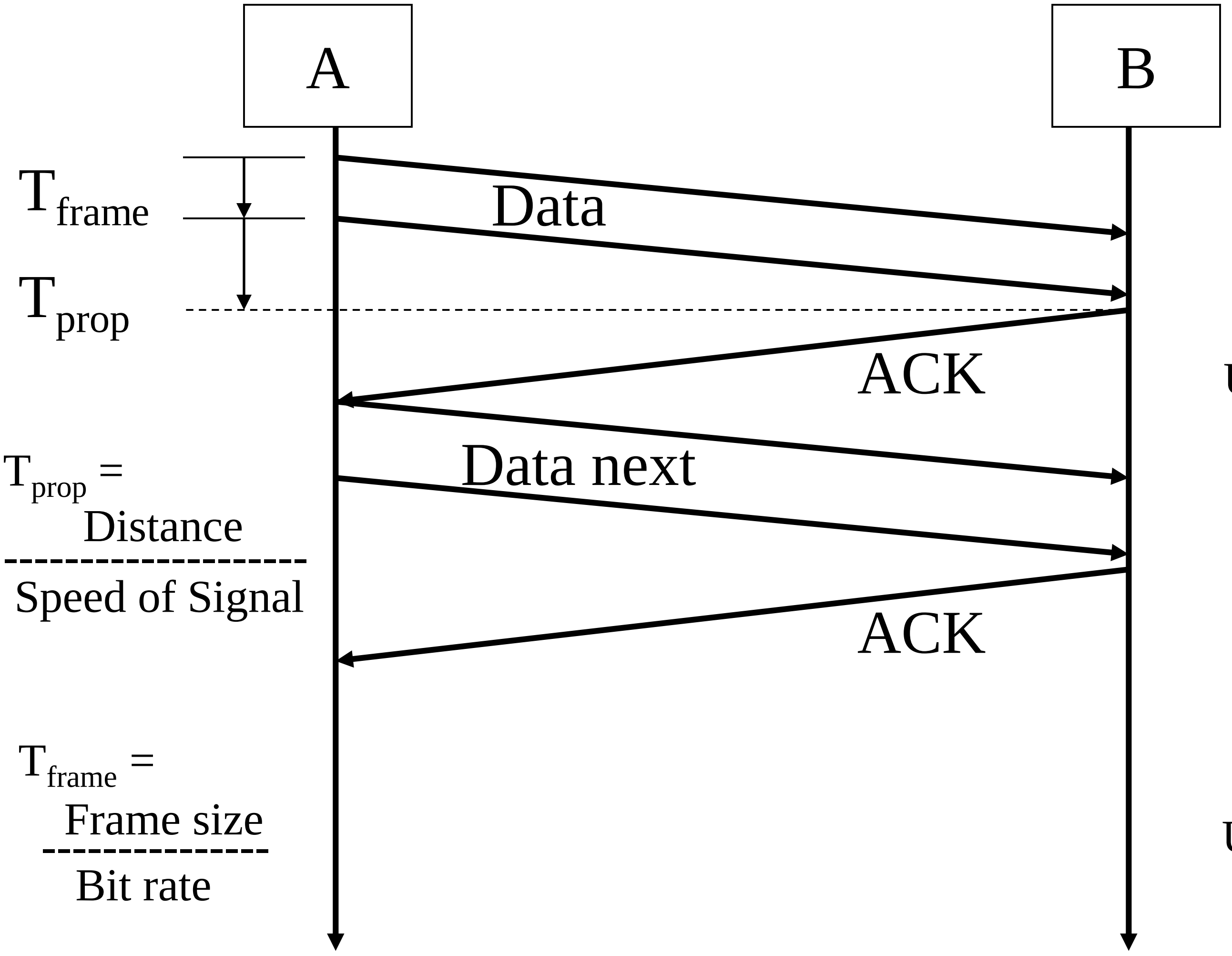
If an ack for last send then prepare for next data frame

Otherwise it was a timeout)

Send next data frame or retransmit old one; ack the last data we received

```
 . . .
 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);
          }
          if (r.ack == next_frame_to_send) {
               stop_timer(r.ack);
               from_network_layer(&buffer);
               inc(next_frame_to_send);
          }
     }
     s.info = buffer;
     s.seq = next_frame_to_send;
     s.ack = 1 – frame_expected;
     to_physical_layer(&s);
     start_timer(s.seq);
 }
}
```

Utlization of Stop-and-wait



T_{prop} =

$$\frac{\text{Distance}}{\text{Speed of Signal}}$$

T_{frame} =

$$\frac{\text{Frame size}}{\text{Bit rate}}$$

Utilization =

$$\frac{T_{frame}}{2\ T_{prop} + T_{frame}}$$

$\alpha = T_{prop} / T_{frame}$

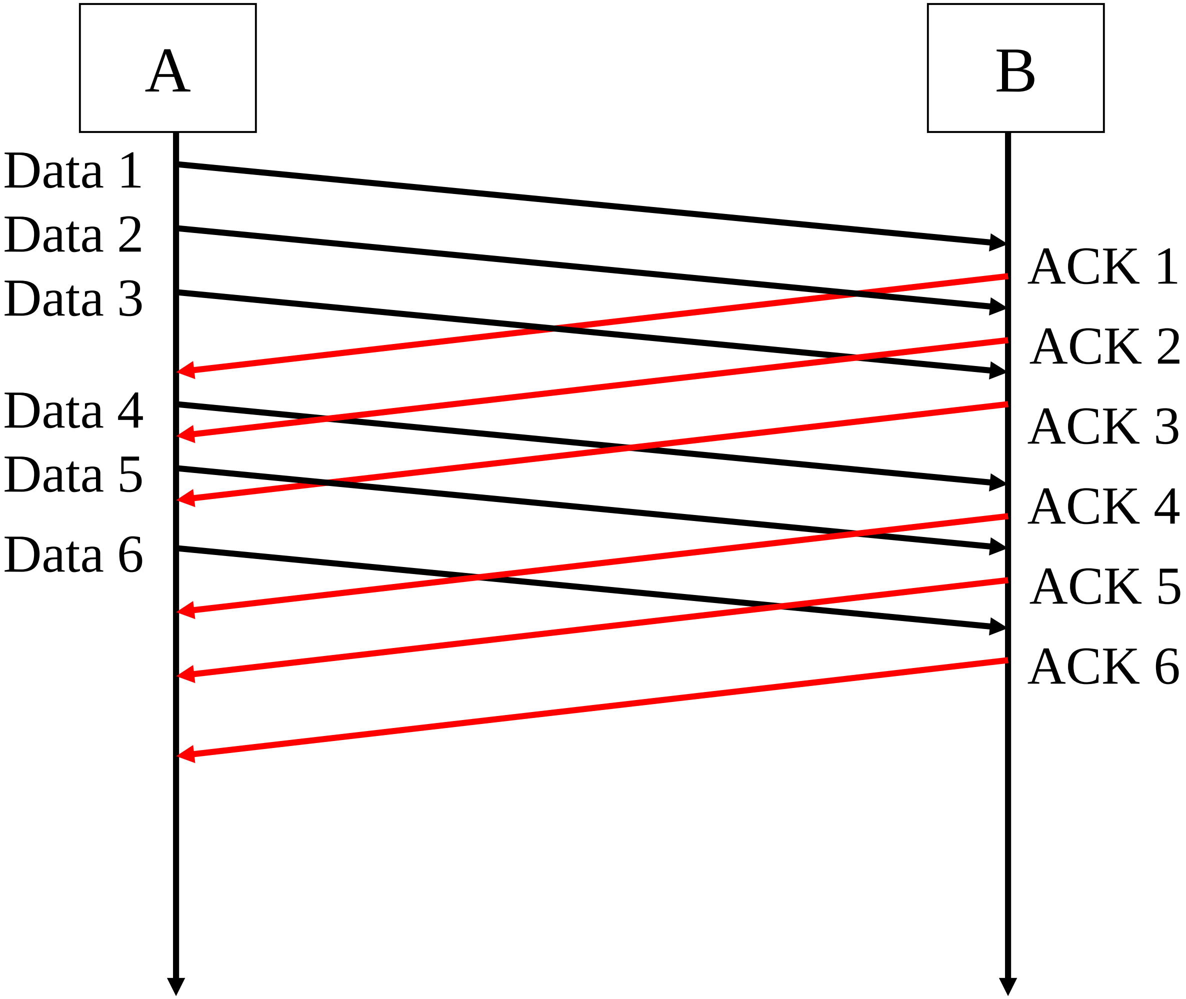
Utilization = $1 / (2\ \alpha + 1)$

Questions

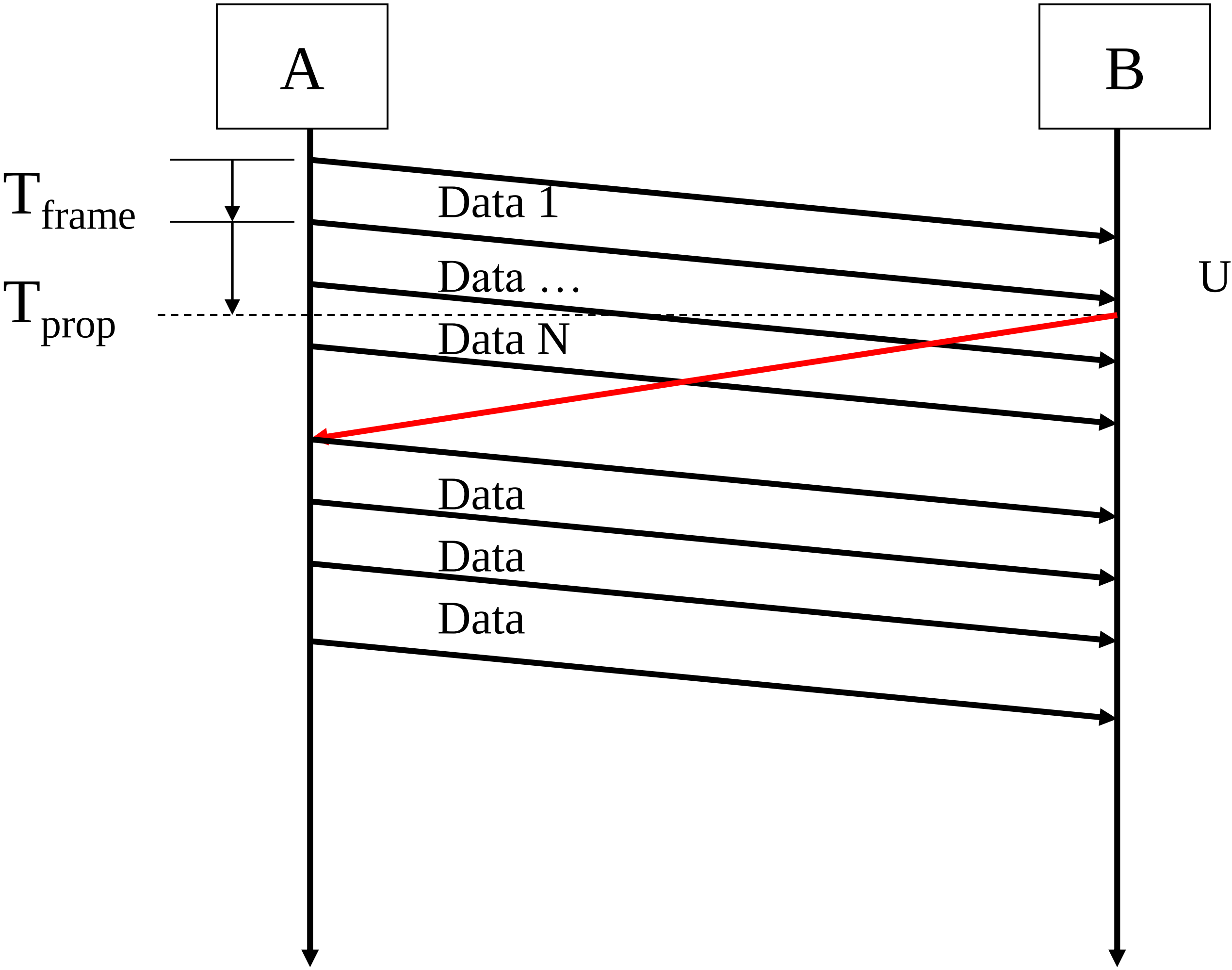
- Short Link(1 km) : t_{prop} = 5 μs
  - Frame Size= 4000 bits
  - Data rate=10 Mbps
  - Utilization = ???
  - If ACK size=100 bits, Utilization=???

- Satellite Link: Propagation Delay t_{prop} = 270 ms
  - Frame Size = 4000 bits
  - Data rate = 56 kbps
  - Utilization = ???

Sliding Window Protocols



Utlization of Sliding-window

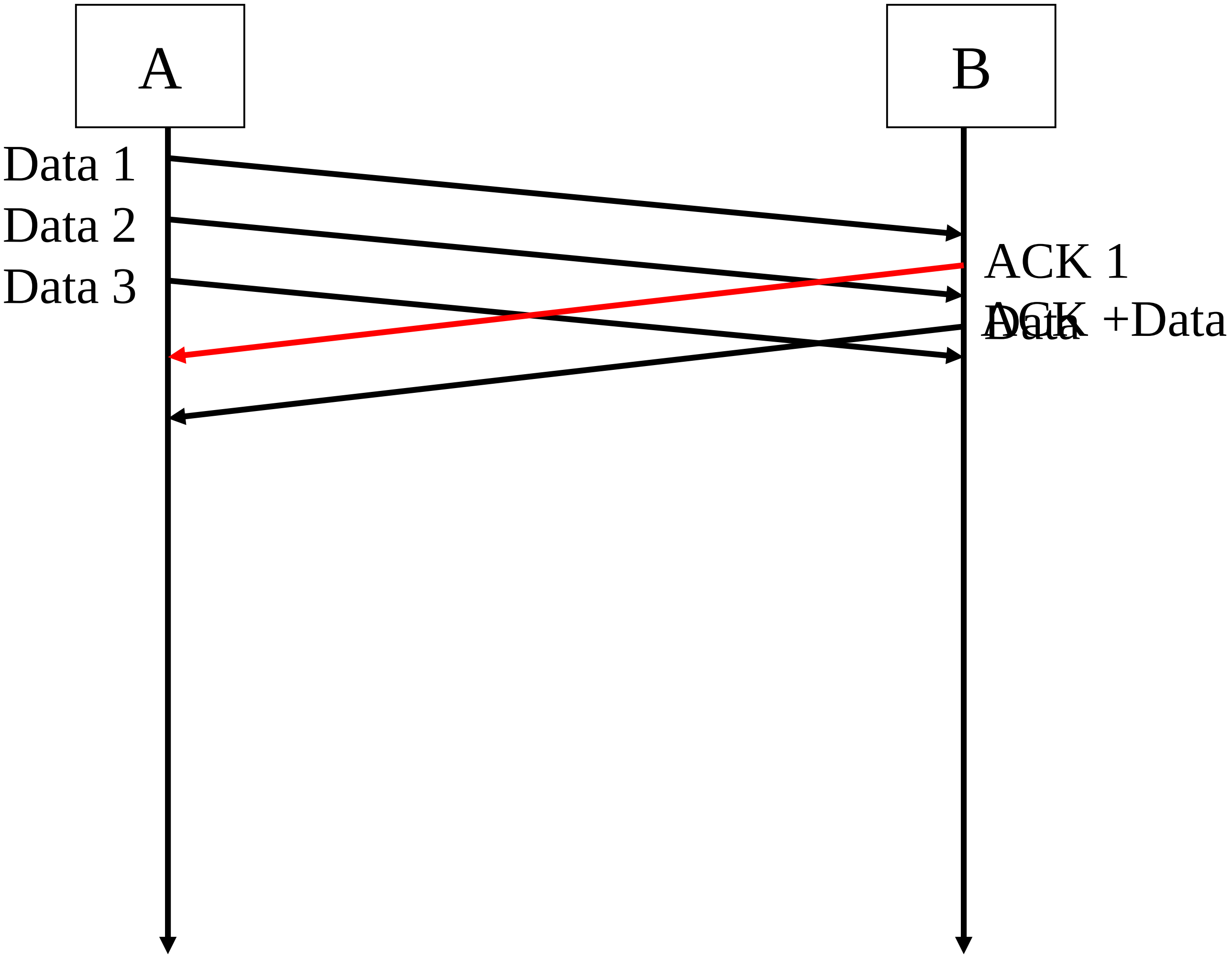


$$\text{Utilization} = \frac{N\ T_{\text{frame}}}{2\ T_{\text{prop}} + T_{\text{frame}}}$$

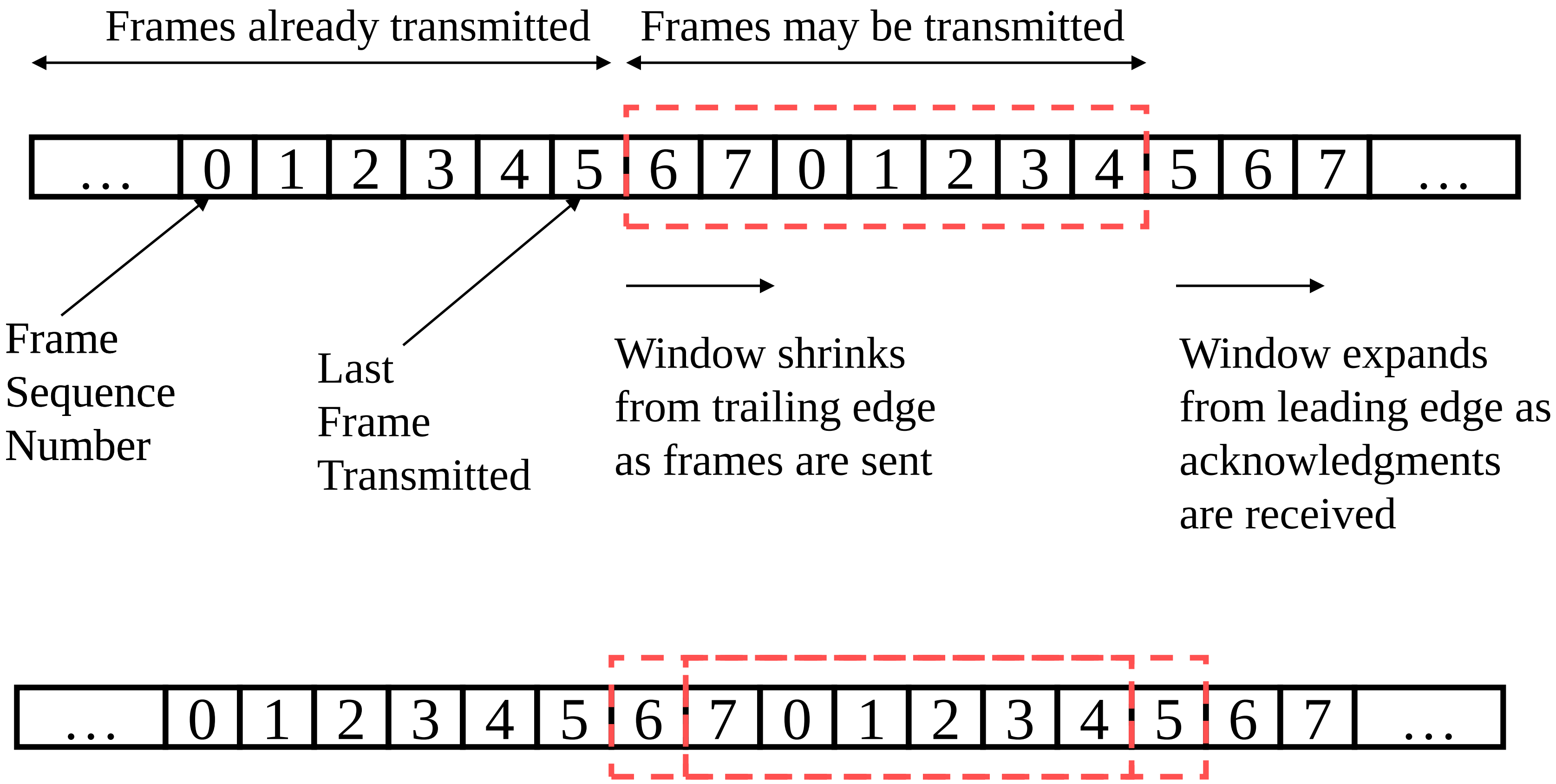
Sliding Window

- Window = Set of sequence numbers permitted to send/receive
- Sender window
  - Sender window increases when ack received
  - Packets in sender window must be buffered at Source
  - Sender window may grow in some protocols

Piggybacking



Sliding Window (Transmitter)



Frames already transmitted

Frames may be transmitted

… 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 …

Frame Sequence Number

Last Frame Transmitted

Window shrinks from trailing edge as frames are sent

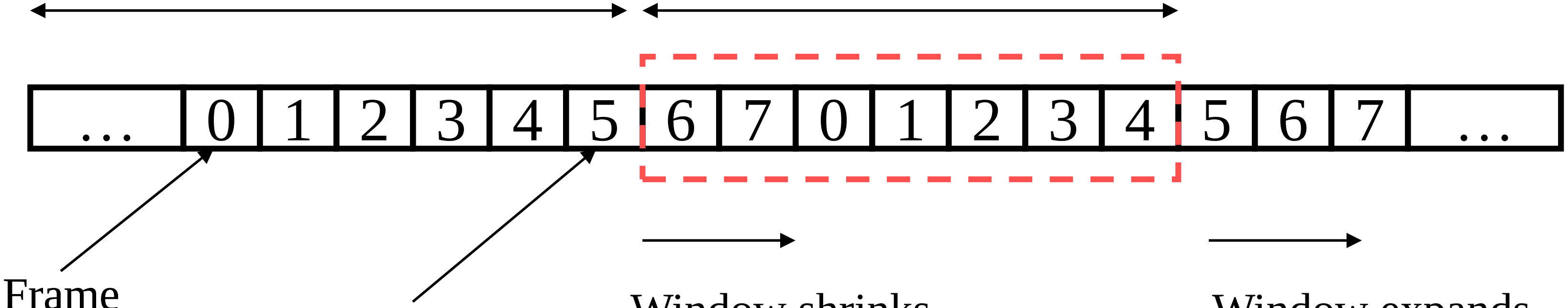
Window expands from leading edge as acknowledgments are received

… 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 …

Sliding Window (Receiver)

Frames already received

Frames may be received



… 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 …

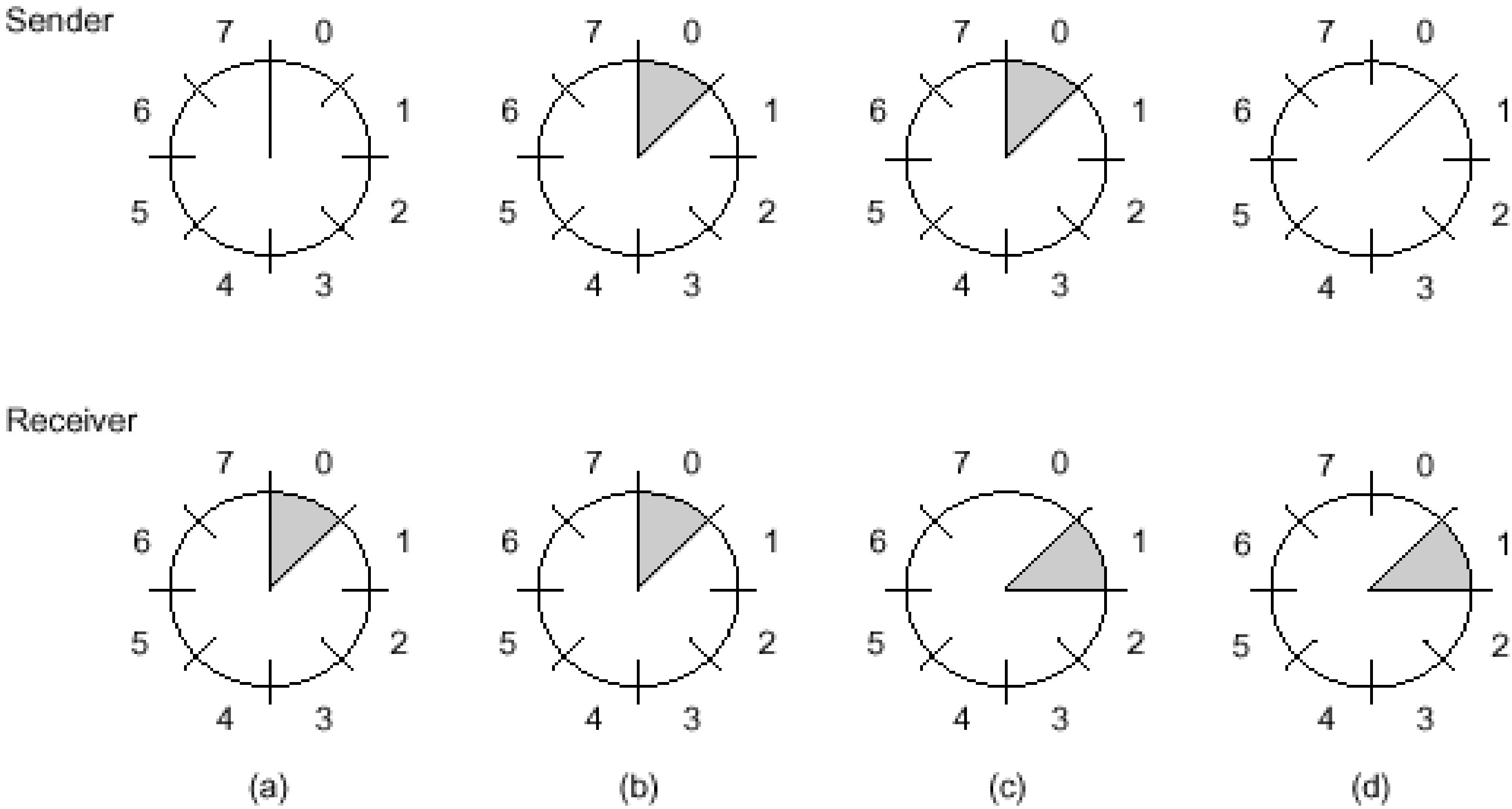
Frame
Sequence
Number

Last
Frame
acknowledged

Window shrinks
from trailing edge
as frames are received

Window expands
from leading edge as
acknowledgments
are sent

Sliding Window in Textbook



SW: upper +1 if frame sent;
lower + 1 if ack received

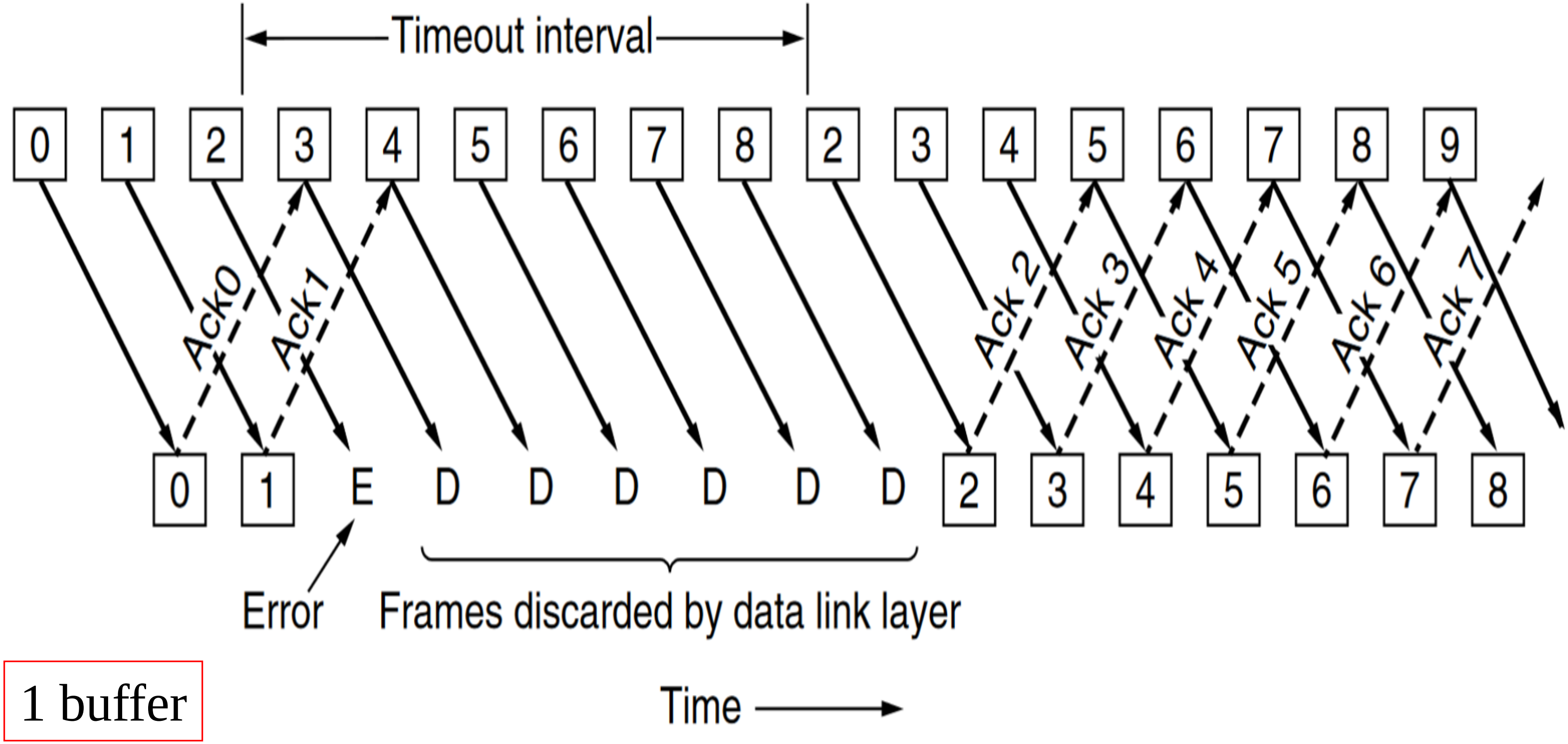
RW: lower+1 and upper +1
if frame received and ack sent

Fig. 3-12. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

SLIDING WINDOW PROTOCOLS (滑动窗口协议)

- Protocol 5: A protocol using **go back N**
- Protocol 6: A protocol using **selective repeat**

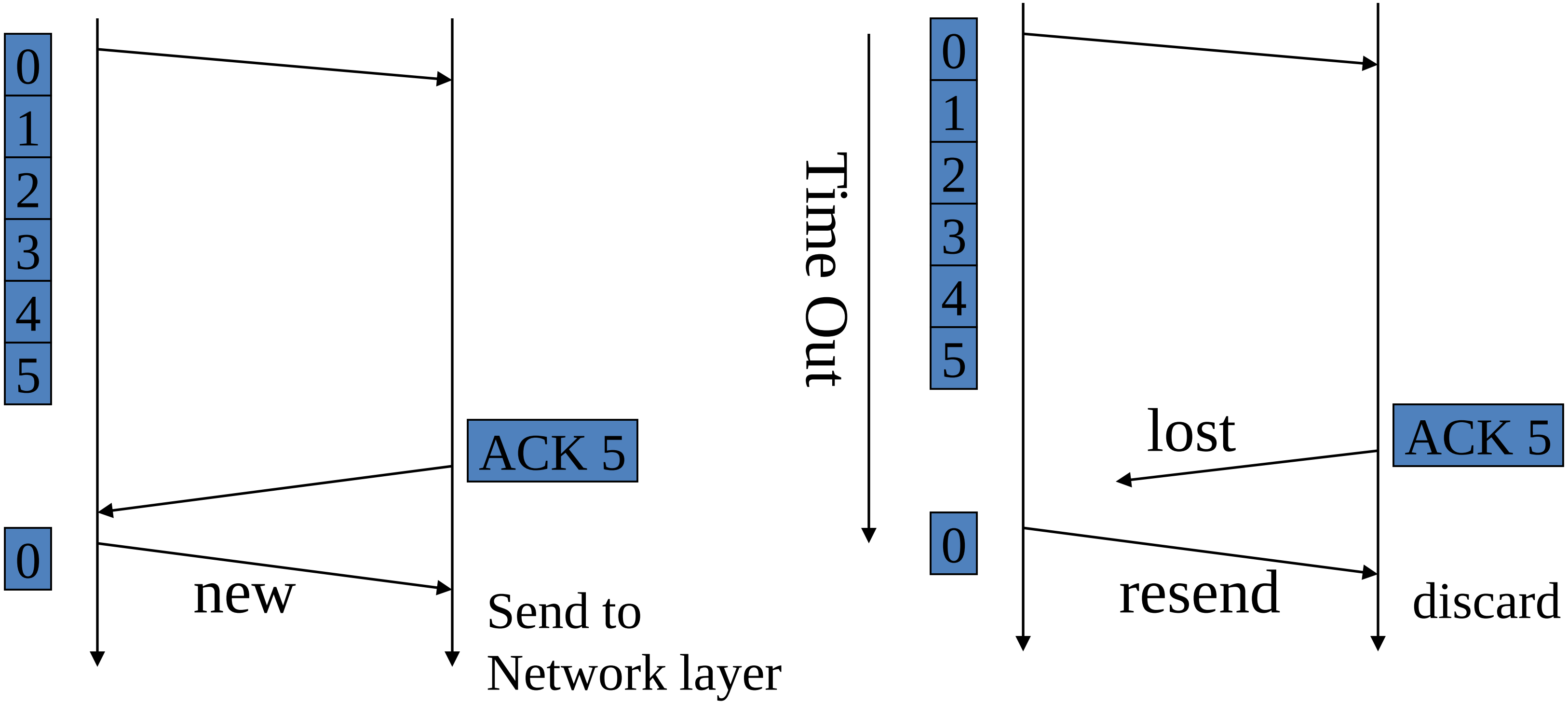
Go-Back-N ARQ



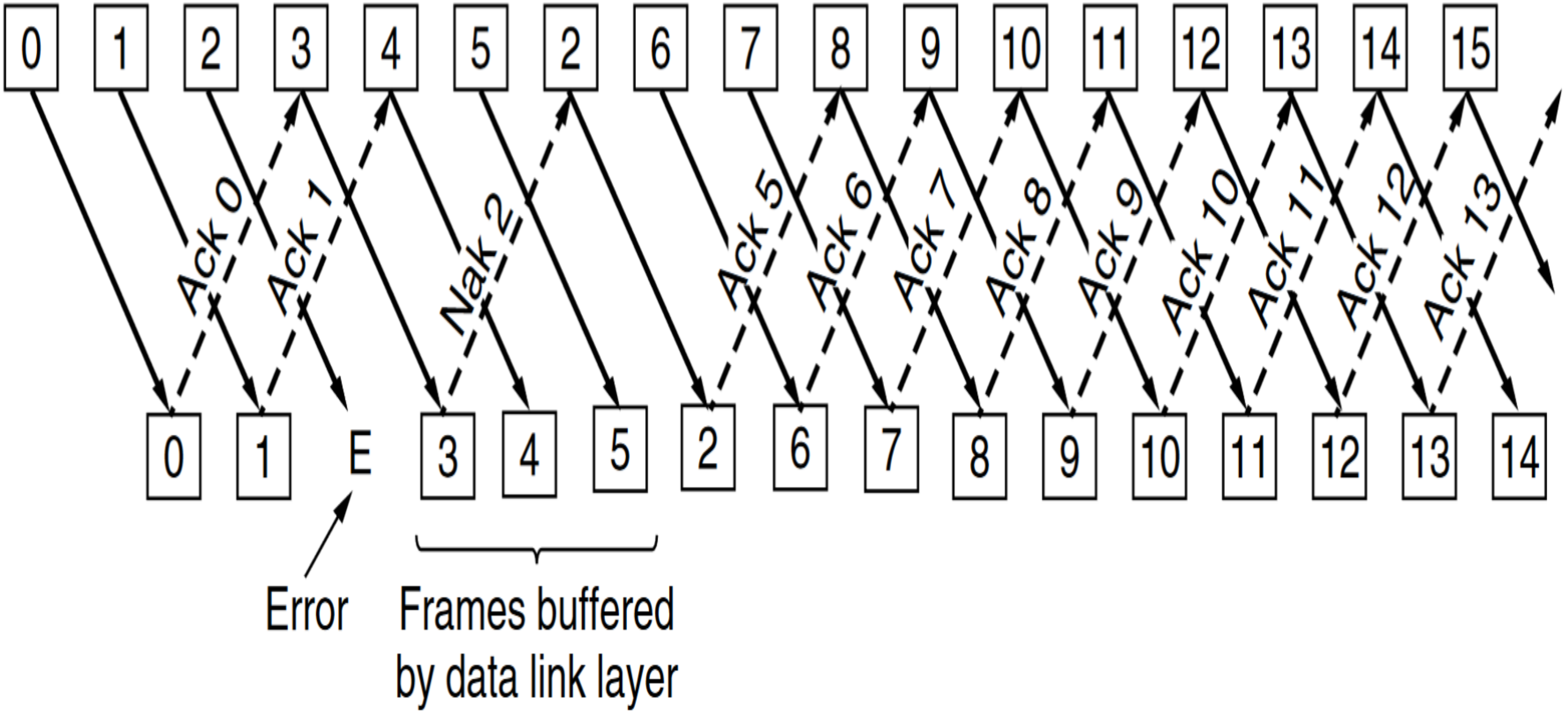
1 buffer

Window Size of Go-Back-N

- Receive Window Size=1
- **Send Window Size <= MAX_SEQ (0…N)**
- **Ex**: max seqno=5, send window=6 is not allowed since seqno 0 can be assigned to both new pkt or old pkt (retransmission)

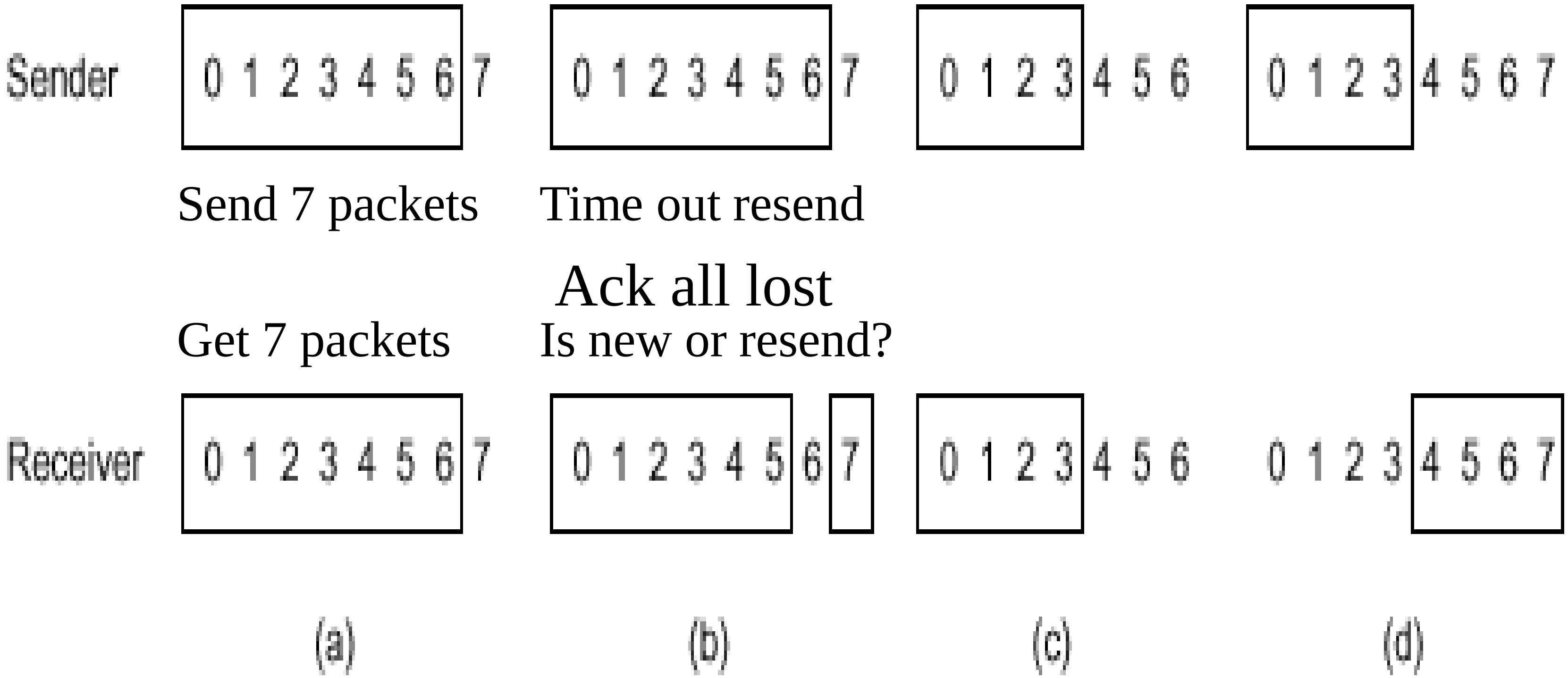


Selective Repeat ARQ



Window Size of Selective Repeat ARQ

- **Send Window Size <= (MAX_SEQ+1)/2**
- Receive Window Size = Send Window Size



Send 7 packets

Time out resend

Ack all lost

Get 7 packets

Is new or resend?

Init window

New window

EXAMPLE DATA LINK PROTOCOLS

- Packet over SONET (Synchronous Optical NETwork)
- ADSL (Asymmetric Digital Subscriber Loop)

PPP

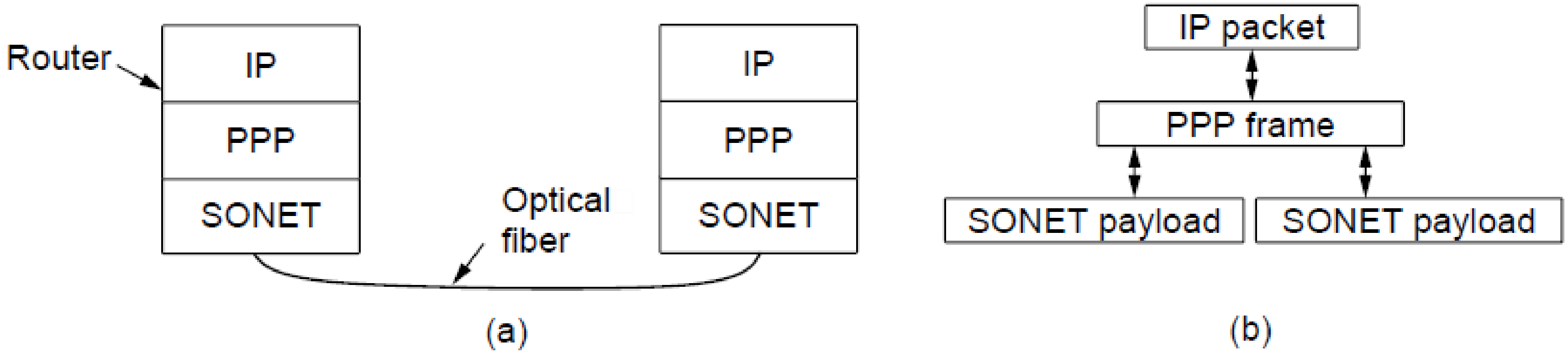
- A standard protocol called **PPP (Point to Point Protocol)** is used to send packets over the links, including the SONET fiber optic links and ADSL links

Example Data Link Protocols:
Packet over SONET

- Packet over SONET.

  (a) A protocol stack.

  (b) Frame relationships



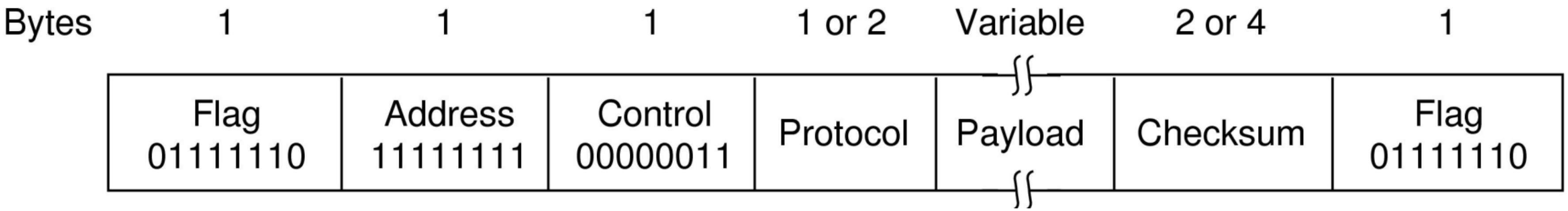
Example Data Link Protocols:

Packet over SONET

- PPP provides three features
  - A framing method that unambiguously delineates the end of one frame and the start of the next one. The frame format also handles error detection.
  - A link control protocol (链路控制协议, LCP) for bringing lines up, testing them, negotiation options, and bring them down again gracefully when they are no longer needed.
  - A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used. The method chosen is to have a different NCP (Network Control Protocol, 网络控制协议) for each network layer supported.

Example Data Link Protocols:
Packet over SONET

- The PPP full frame format for unnumbered mode operation

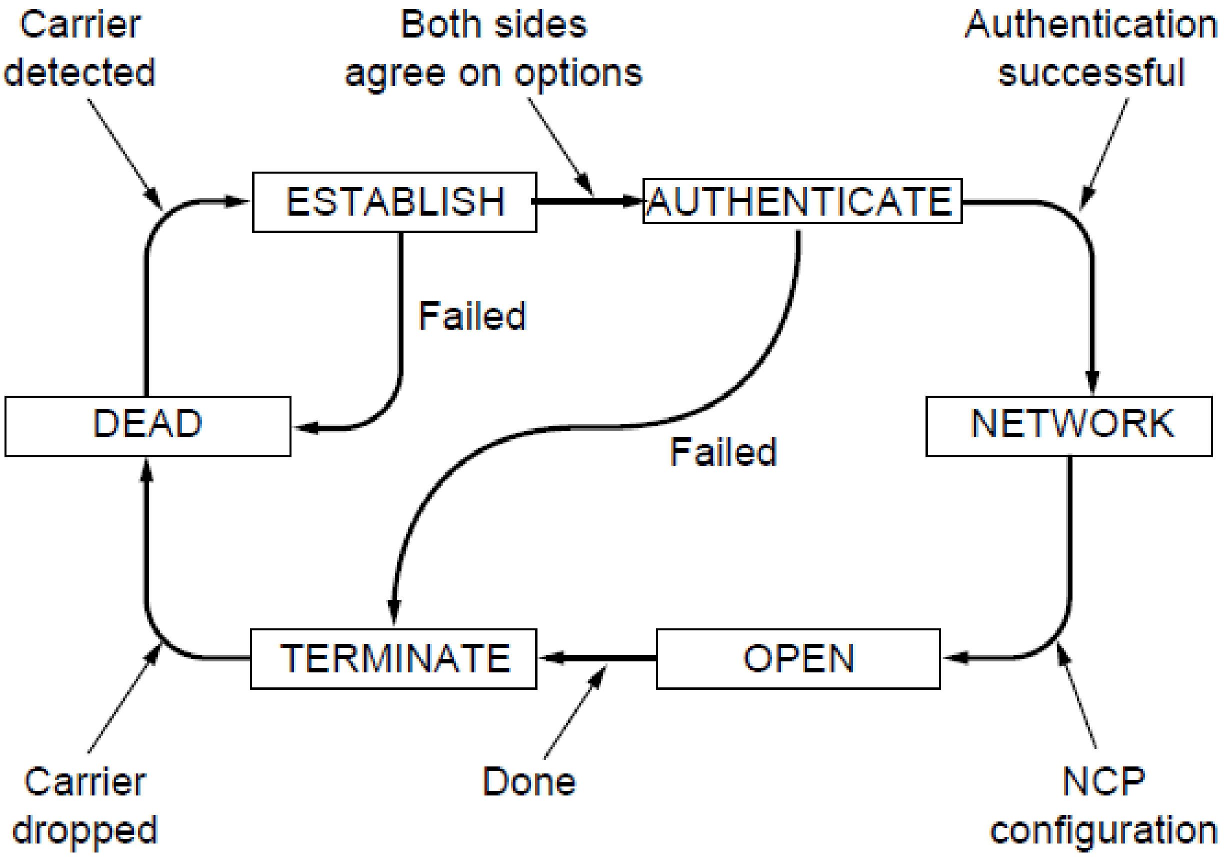


Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

- Delimiters: 01111110
- Address: 11111111
- Control: 00000011
- Protocol: To tell what kind of packet is in the Payload field.
- Payload: of variable length, up to some negotiated maximum.
- Checksum: CRC checksum.

Example Data Link Protocols:
Packet over SONET

State diagram for bringing a PPP link up and down



Example Data Link Protocols: PPP over SONET

- Typical Scenario (establishing a connection)
  - The PC first calls the ISP’s router via a modem. The router’s modem answers the phone and establish a physical connection.
  - The PC sends the router a series of LCP packets in the payload field of one or more PPP frames. These packets and their responses select the PPP parameters to be used.
  - Once the parameters have been agreed upon, a series of NCP packets are sent to configure the network layer. (DHCP, NAT)
  - Now, the PC is an Internet host and can send/receive IP packets.

Example Data Link Protocols: PPP over SONET

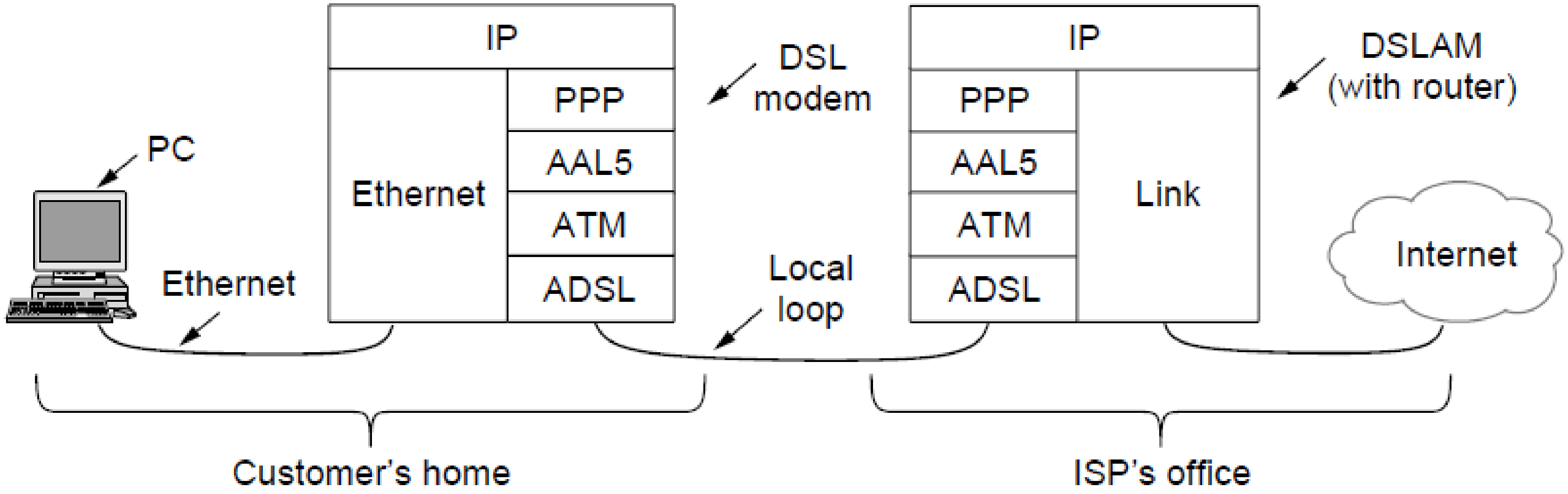
- Typical Scenario (releasing a connection)
  - When the user is finished, NCP tears down the network layer connection and frees up the IP address.
  - Then LCP shuts down the data link layer connection.
  - Finally, the computer tells the modem to hang up the phone, releasing the physical layer connection.

Example Data Link Protocols: ADSL

ADSL protocol stacks.

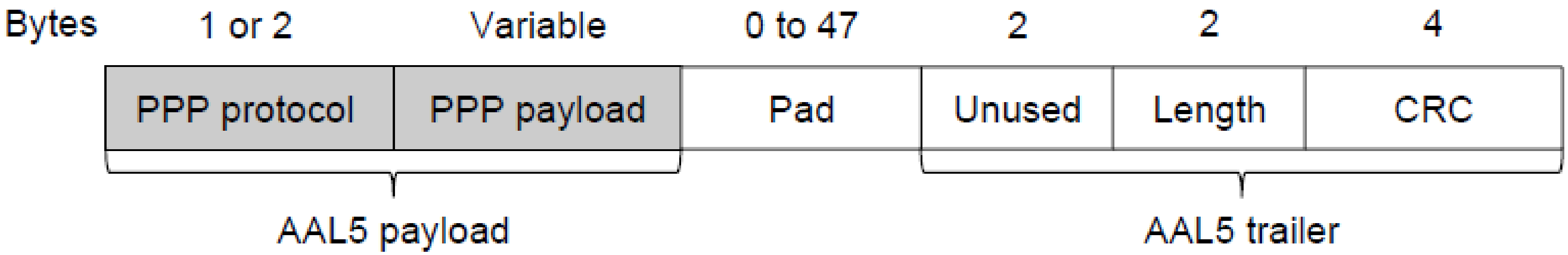
ATM: Asynchronous Transfer Mode

AAL5: ATM Adaptation Layer 5



Example Data Link Protocols: ADSL

AAL5 frame carrying PPP data



Bytes	1 or 2	Variable	0 to 47	2	2	4
	PPP protocol	PPP payload	Pad	Unused	Length	CRC

AAL5 payload

AAL5 trailer

Homework

1.The following character encoding is used in a data link protocol:

A: 01000111 B: 11100011 FLAG: 01111110 ESC: 11100000

Show the bit sequence transmitted(in binary) for the four-character frame A B ESC FLAG when each of the following framing methods is used:

(a) Byte count.

(b) Flag bytes with byte stuffing.

(c) Starting and ending flag bytes with bit stuffing.

2. Hamming code is an effective way for error correcting. Show that the # of check bits(i.e. r) in the Hamming codes described in the textbook(e.g., Fig.3-6) (almost) achieves the low bound of Eq (3-1).

Homework

3. Suppose you have the following 12-bit message: 010100111111

(a) Numbering bits from right to left (ie least-significant bit on the right), insert check bits according to to Hamming’s 1-bit error correction system. Indicate which bits are check bits and which are message bits.

(b) Hamming’s scheme only corrects 1-bit errors. Since it’ s a distance 3 code, it could also be used to detect 2-bit errors. Describe a 3-bit error (3 *1-bit errors) in the above codeword affecting only message bits
(not check bits) that would be undetected (and of course uncorrected). Be sure to describe how and why the algorithm fails.

Homework

4.Consider an original frame 110111011011. The generator polynomial x^4+x+1, show the converted frame after appending the CRC.

5.A 3000-km-long T1 trunk (with data rate 1.536Mbps) is used to transmit 64-byte frames. How many bits should the sequence numbers be for protocol 5 and protocol 6 respectively? The propagation speed is 6usec/km.

Homework

6.Frames of 1000 bits are sent over a 1-Mbps channel using a geostationary satellite whose propagation time from the earth is 270 msec. Acknowledgements are always piggybacked onto data frames. The headers are very short. Three-bit sequence numbers are used. What is the maximum achievable channel utilization for

(a) Stop-and-wait?

(b) Protocol 5?

(c) Protocol 6?

Homework

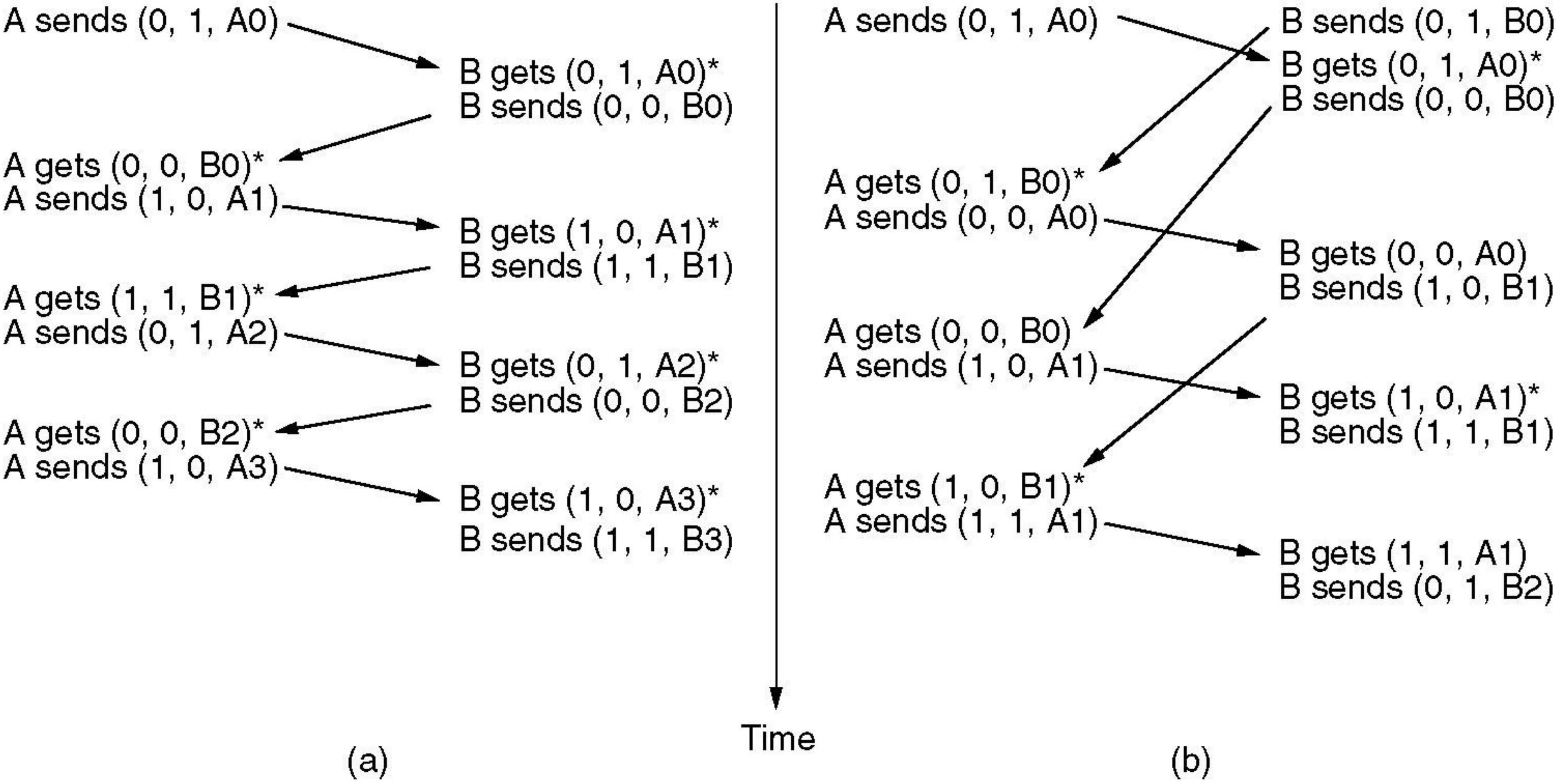
7. Compute the fraction of the useful data bandwidth for protocol 6 on a heavily loaded 50-kbps satellite channel with data frames consisting of 40 header and 3960 data bits. Assume that the signal propagation time from the earth to the satellite is 270 msec. ACK frames never occur. NAK frames are 40 bits. The error rate for data frames is 1%, and the error rate for NAK frames is negligible. The sequence numbers are 3 bits.

Sliding Window Protocols: Protocol 4

- A peculiar situation arises if both sides simultaneously send an initial packet.

Sliding Window Protocols: Protocol 4

The notation is **(seq, ack, info).** An asterisk ***** indicates where a network layer accepts a packet. (a) Normal case. (b)Abnormal case.



A sends (0, 1, A0)

B gets (0, 1, A0)*
B sends (0, 0, B0)

A gets (0, 0, B0)*
A sends (1, 0, A1)

B gets (1, 0, A1)*
B sends (1, 1, B1)

A gets (1, 1, B1)*
A sends (0, 1, A2)

B gets (0, 1, A2)*
B sends (0, 0, B2)

A gets (0, 0, B2)*
A sends (1, 0, A3)

B gets (1, 0, A3)*
B sends (1, 1, B3)

(a)

Time

A sends (0, 1, A0)

B sends (0, 1, B0)
B gets (0, 1, A0)*
B sends (0, 0, B0)

A gets (0, 1, B0)*
A sends (0, 0, A0)

B gets (0, 0, A0)
B sends (1, 0, B1)

A gets (0, 0, B0)
A sends (1, 0, A1)

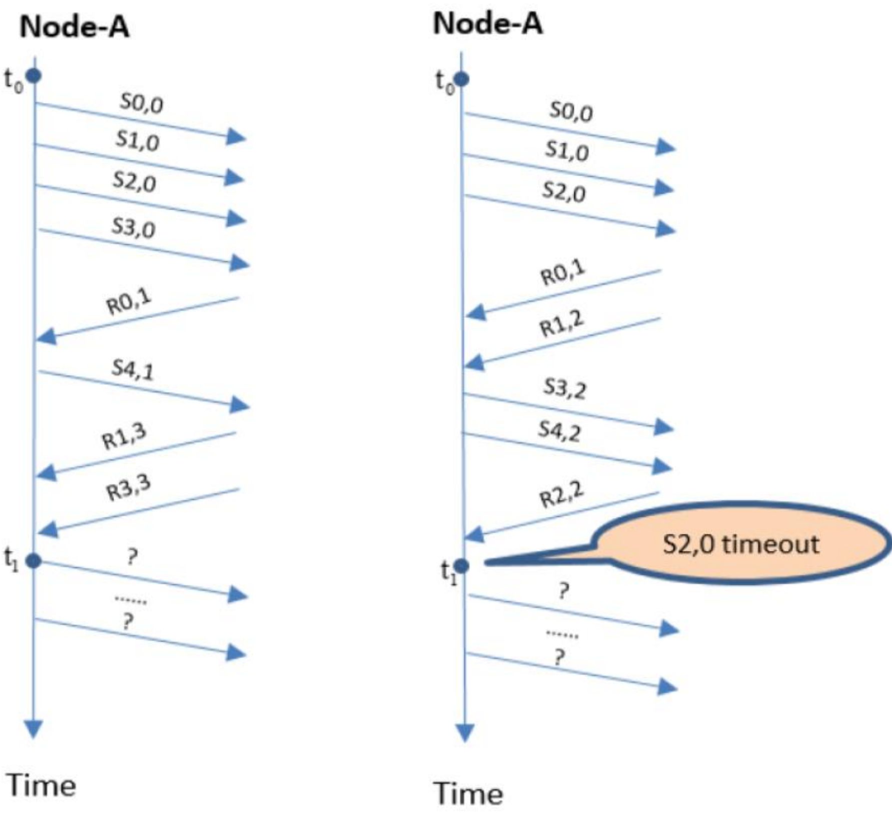
B gets (1, 0, A1)*
B sends (1, 1, B1)

A gets (1, 0, B1)*
A sends (1, 1, A1)

B gets (1, 1, A1)
B sends (0, 1, B2)

(b)

Node-A and Node-B use the Go-Back-N frame protocol (GBN) for continuous two-way data transmission, and both parties always use piggyback confirmation, frame length is always 1000B. Sx,y and Rx,y respectively represent the data frames sent by both Node-A and Node-B, where: x is the transmission number, y is the acknowledgment number (indicating the next frame number to be expected to receive). The field of the transmission sequence number and the acknowledgment sequence number of the data frame is 3 bits. The channel transmission rate is 100-Mbps and RTT (Round Trip Time) = 0.96ms. The following figures show two scenarios in which Node-A sends and receives data frame, where t0 is the initial time and at that time both of the sending and acknowledgment sequence number of Node-A is 0, and the node-A has enough data to be transmitted at time t1.



(a)   (b)

(1) For figure (a), starting from time t0 to t1, how many frames does Node-A can confirm that Node-B has correctly received? What is (are) the correct frame(s) received (List in Sx, y)?

(2) For figure (a), starting from time t1, how many data frames can Node-A send before a timeout doesn’t occur and no new Node-B data frame is received? What are the first frame and the last frame (List in Sx, y)?

(3) For figure (b), starting from time t1, how many data frames does Node-A need to retransmit before a timeout doesn’t occur and no new Node-B data frame is received? What is the first retransmission frame (List in Sx, y)?

(4) What is the maximum channel utilization that Node-A can achieve?