COMPUTER NETWORKS

By

Dr. Kadiyala Ramana

Associate Professor

Department of Artificial Intelligence and Data Science

Chaitanya Bharathi Institute of Technology, Hyderabad

E-mail: kramana_aids@cbit.ac.in

Mobile Number: +91-9666635150

CONTENTS

- Data Link Layer: Design issues
- Framing
- Error detection and correction
- Elementary data link protocols: simplex protocol, A Simplex Stop and Wait Protocol for an Error-free channel, A Simplex Stop and Wait Protocol for Noisy Channel
- Sliding Window protocols: A One-Bit Sliding Window Protocol, A protocol using Go-Back-N, A Protocol using Selective Repeat
- Example data link protocols
- Medium Access Sub Layer: The Channel allocation problem,
- Multiple Access Protocols: ALOHA, Carrier Sense Multiple Access Protocols, Collision Free Protocols
- Ethernet.

INTRODUCTION

Previous Layer: The Physical Layer

- The physical layer is concerned with transmitting raw bits over a communication channel.

Next Layer: The Data Link Layer

- The main task of the data link layer is to transform a raw transmission facility into a line that appears free of undetected transmission errors to the network layer.

Why Data Link Layer?

- You might think this problem is so trivial that there is no software to study – machine A just puts the bits on the wire, and machine B just takes them off.
- Unfortunately, communication circuits make errors occasionally.

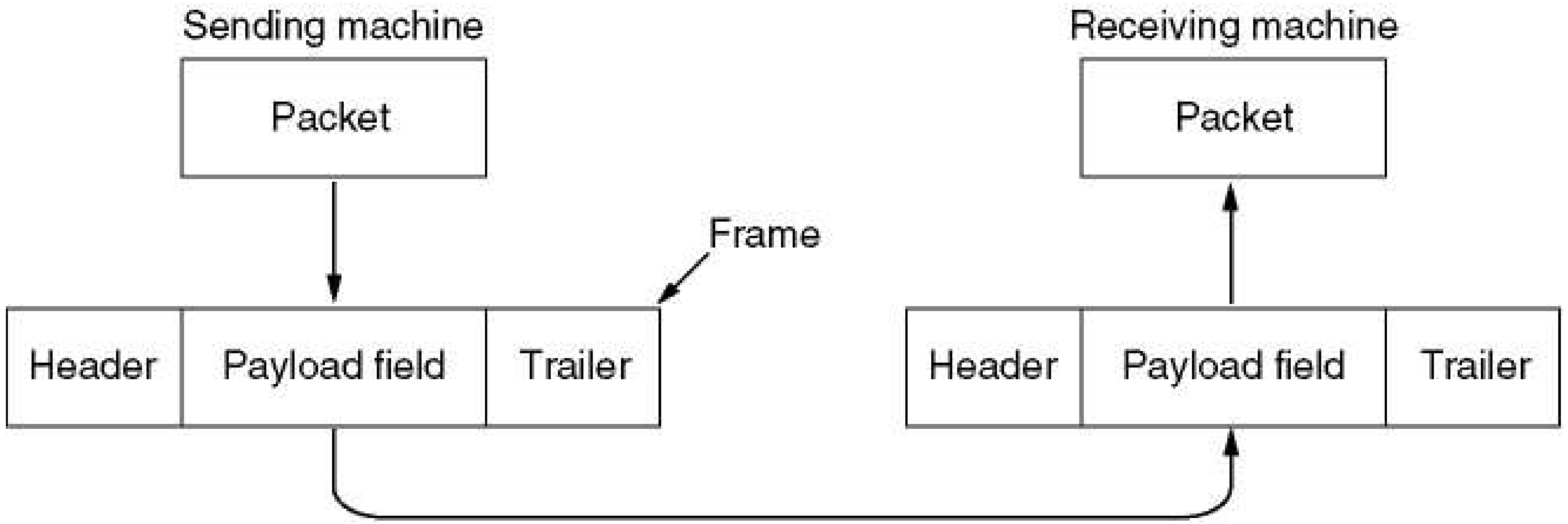
INTRODUCTION

- Machines have only a finite data rate, and there is a nonzero propagation delay between the time a bit is sent and the time it is received.
- These limitations have important implications for the efficiency of the data transfer.
- The protocols used for communication must take all these factors into consideration.

DATA LINK LAYER DESIGN ISSUES

- The data link layer's specific functions:
  - ➢ Providing a well-defined service interface to the network layer
  - ➢ Dealing with transmission errors
  - ➢ Regulating the flow of data so that slow receivers are not swamped by fast senders.
  - ➢ The data link layer takes the packets it gets from the network layer and encapsulates them into the frames for transmission.
  - ➢ Each frame consists of a frame header, a payload field for holding the packet, and a frame trailer.
  - ➢ Frame management forms the heart of what the data link layer does.

DATA LINK LAYER DESIGN ISSUES

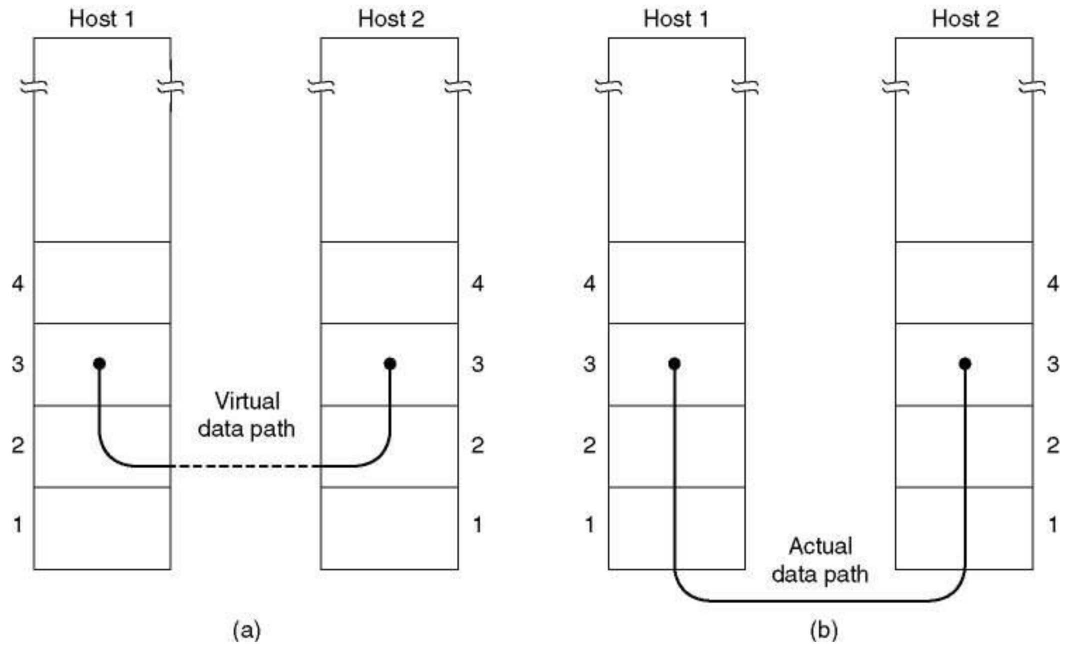


Relationship between packets and frames.

DATA LINK LAYER DESIGN ISSUES

Services Provided to Network Layer:

- The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine.
- The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer.



(a) Virtual communication. (b) Actual communication.

Services Provided to Network Layer:

- The data link layer can be designed to offer various services.
  a) Unacknowledged connectionless service.
  b) Acknowledged connectionless service.
  c) Acknowledged connection-oriented service.

DATA LINK LAYER DESIGN ISSUES

Services Provided to Network Layer:

Unacknowledged connectionless service:

- The source machine sends independent frames to the destination machines without having the destination machine acknowledge them.
- No logical connection is established beforehand or released afterward.
- If a frame is lost due to noise on the line, no attempt is made to detect the loss or recover from it in the data link layer.
- It is used where error rate is very low, for real-time traffic such as voice, most LANs use it.

DATA LINK LAYER DESIGN ISSUES

Services Provided to Network Layer:

Acknowledged connectionless service:

- Reliable connectionless service
- When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged.
- In this way, the sender knows whether a frame has arrived correctly.
- If it has not arrived within a specified time interval, it can be sent again.
- For unreliable channels, such as wireless systems.

DATA LINK LAYER DESIGN ISSUES

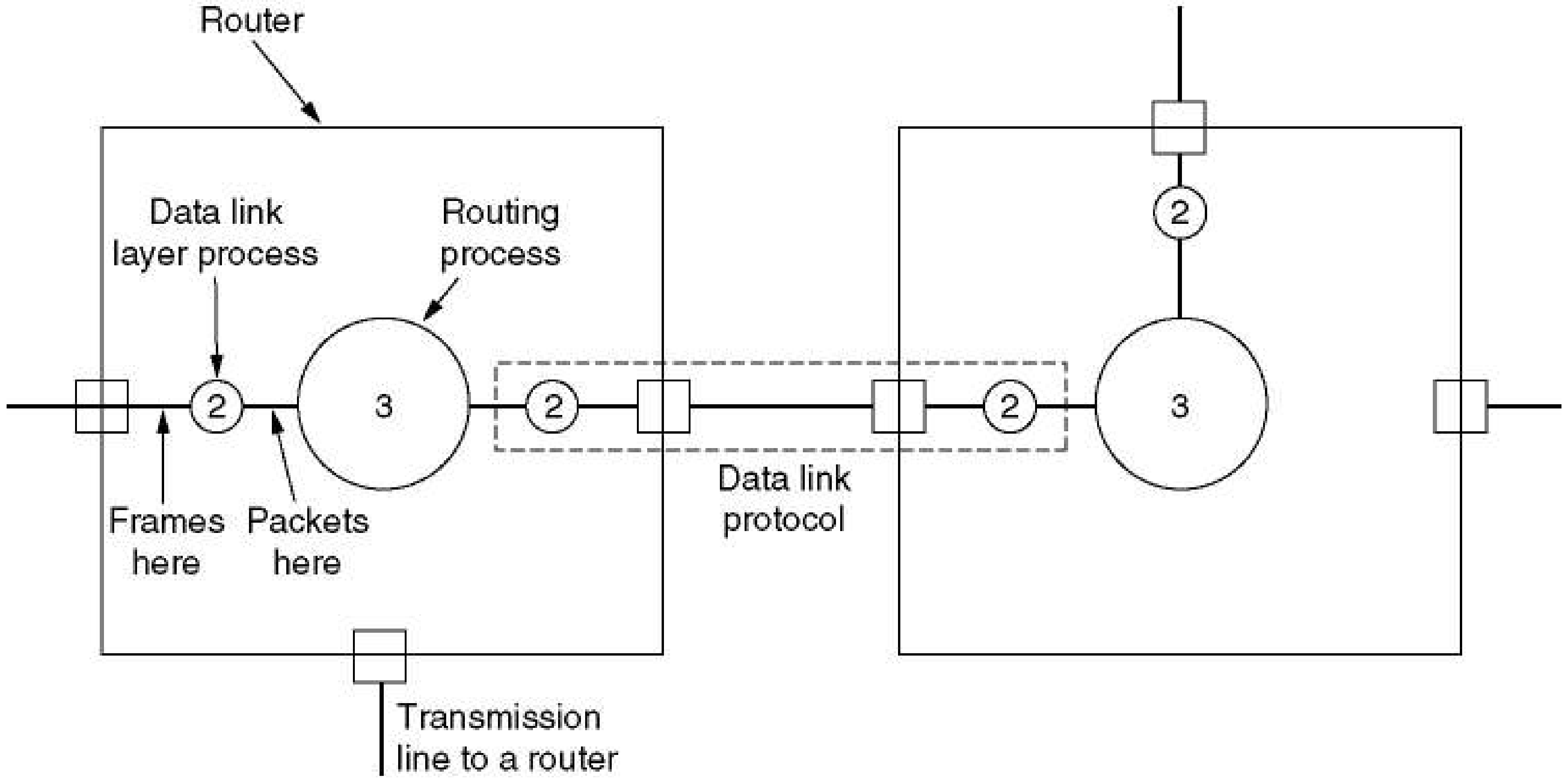
Services Provided to Network Layer:

Acknowledged connection-oriented service:

- The source and destination machines establish a connection before and data are transferred.
- Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received.
- Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order.
- There are three distinct phases in connection oriented service.
    - ➢ The connection is established;
    - ➢ Frames are transmitted by this connection;
    - ➢ The connection is released.

DATA LINK LAYER DESIGN ISSUES

Example: A WAN subnet consisting of routers connected by point-topoint leased telephone lines..



Placement of the data link protocol.

DATA LINK LAYER DESIGN ISSUES: FRAMING

- To provide service to the network layer, the data link layer must use the service provided to it by the physical layer.
- What the physical layer does is accept a row bit stream and attempt to deliver it to the destination.
- The number of bits received may be less than, equal to, or more than the number of bits transmitted, an they may have different values.
- It is up to the data link layer to detect and, if necessary, correct errors.
- The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame.
- When a frame arrives at the destination, the checksum is recomputed.
- If the newly – computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it.

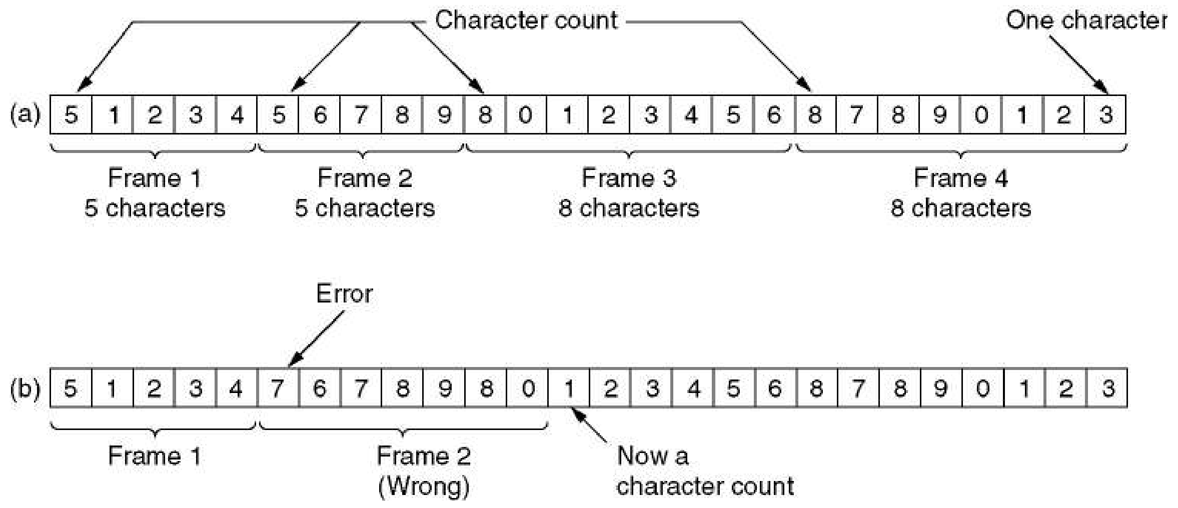
DATA LINK LAYER DESIGN ISSUES: FRAMING

- The methods to break the bit stream up into frames:
  1) Character count
  2) Flag bytes with byte stuffing
  3) Starting and ending flags, with bit stuffing
  4) Physical layer coding violations

DATA LINK LAYER DESIGN ISSUES: FRAMING

Character count:

- This framing method uses a field in the header to specify the number of characters in the frame.
- When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is.
- The trouble with this algorithm is that the count can be garbled by a transmission error.



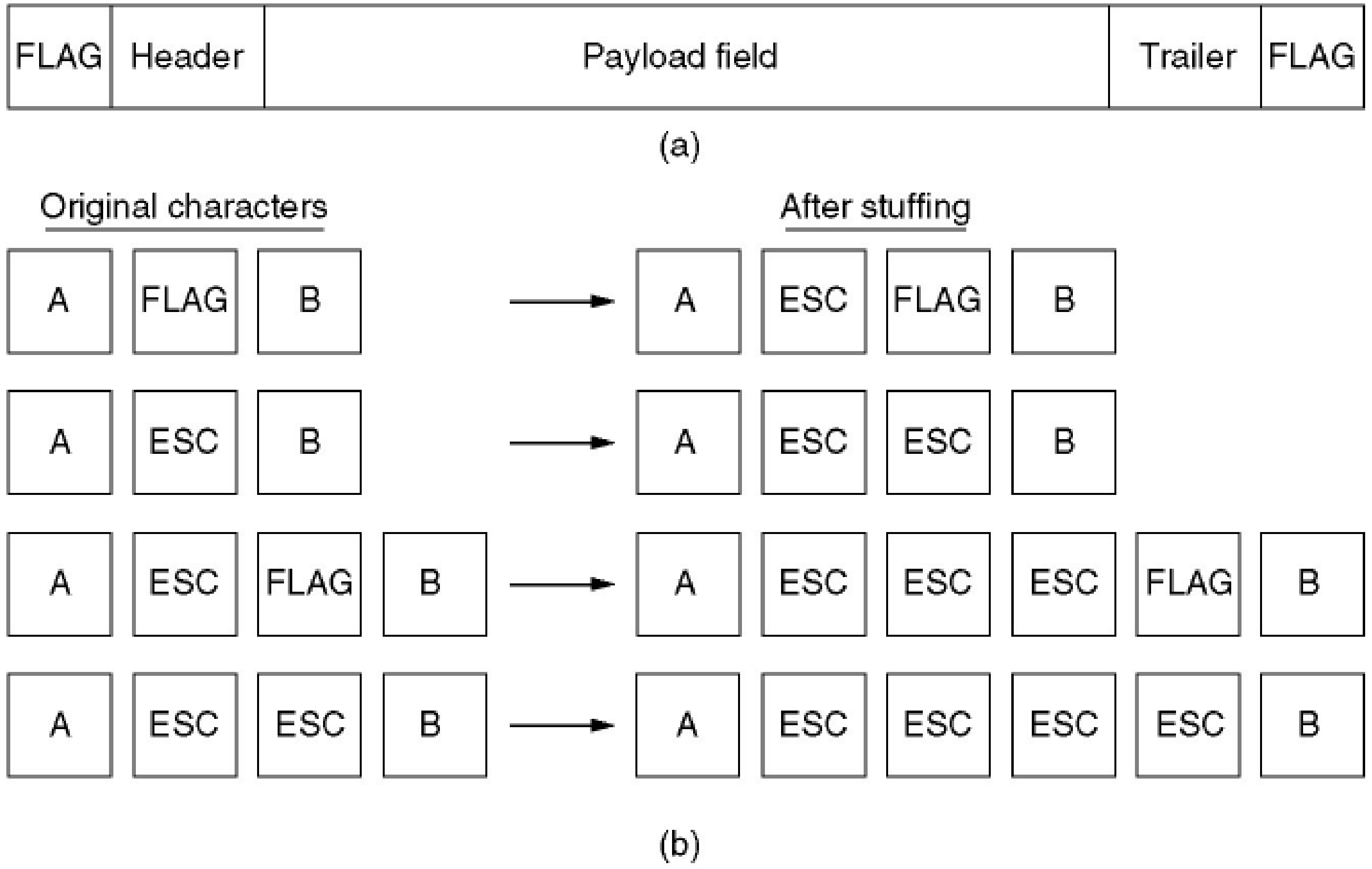
A character stream. (a) Without errors for four frames of sizes 5, 5,8, and 8 characters, respectively. (b) With one error.

DATA LINK LAYER DESIGN ISSUES: FRAMING

Flag bytes with byte stuffing:

- This method gets around the problem of resynchronization after an error by having each frame start and end with special bytes.
- The starting and ending Flag Bytes are same in recent years.
- If the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame.
- 2 consecutive flag bytes indicate the end of one frame and start of next one.

Flag bytes with byte stuffing:



(a) A frame delimited by flag bytes.
(b) Four examples of byte sequences before and after stuffing.

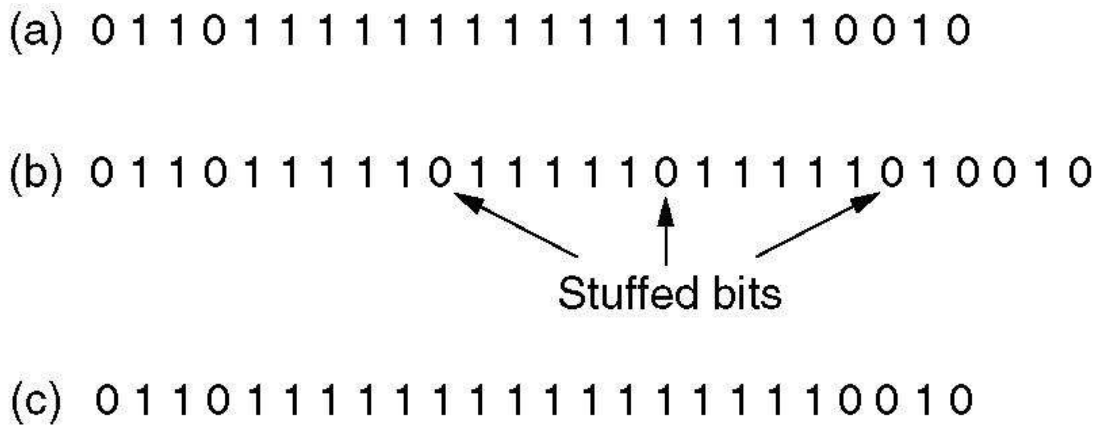
DATA LINK LAYER DESIGN ISSUES: FRAMING

Flag bytes with byte stuffing:

- A serious problem occurs with this method when binary data, such as object programs or floating point numbers, are being transmitted.
- It may easily happen that the flag byte's bit pattern occurs in the data.
- This situation will usually interfere with the framing.
- One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each "accidental" flag byte in the data.
- A major disadvantage of using this framing method is that it is closely tied to the use of 8-bit characters.
- Not all character code use 8-bit characters.
- For example, UNICODE uses 16-bit characters.
- As networks developed, the disadvantages of embedding the character code length in the framing mechanism became more and more obvious, so a new technique had to be developed to allow arbitrary sized characters.

Starting and ending flags, with bit stuffing:

- The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character.
- Each frame begins and ends with a special bit pattern, 01111110 (in fact, a flag byte).
- Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream.


(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 receiver's memory after destuffing.

DATA LINK LAYER DESIGN ISSUES: FRAMING

Physical layer coding violations:

- This method is only applicable to networks in which the encoding on the physical medium contains some redundancy.
- 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 a low-high pair.
- 11 or 00 are not used for data but are used for delimiting frames in some protocols.

DATA LINK LAYER DESIGN ISSUES: ERROR CONTROL

- The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line.
- Typically, the protocol calls for the receiver to send back special control frames.
- If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely.
- On the other hand, a negative acknowledgement means that something has gone wrong, and the frame must be transmitted again.

DATA LINK LAYER DESIGN ISSUES: FLOW CONTROL

- Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them.
- To solve this problem feedback-based flow control and rate-based flow control are used.
- **Feedback-based flow control:** the receiver sends back information to the sender giving it permission to send more data or at least telling the sender how the receiver is doing.
- **Rate-based flow control:** the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.
- Rate-based flow control is never used in the data link layer.

ERROR DETECTION AND CORRECTION

- The telephone system has three parts:
  1) the switches – almost entirely digital
  2) the interoffice trunks - almost entirely digital
  3) the local loops – still analog (twisted copper pairs)
- While errors are rare on the digital part, they are still common on the local loops.
- Furthermore, wireless communication is becoming more common, and the error rates
- here are orders of magnitude worse than on the interoffice fiber trunks.
- The conclusion is: transmission errors are going to be with us for many years to come.

ERROR DETECTION AND CORRECTION

- As a result of the physical processes that generate them, errors on some media (e.g., radio) tend to come in bursts rather than singly.
- The advantage of bursts errors is that the computer data are always sent in blocks of bits.
- The disadvantage is that they are much harder to correct than are isolated errors.
- **Error-Correcting Codes**
  - ➢ To include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been.
- **Error-Detecting Codes**
  - ➢ To include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission.
- On channels that are highly reliable, such as fiber, it is cheaper to use an error detecting
- code and just retransmit the occasional block found to be faulty.
- On channels such as wireless links that make many errors, it is better to add enough redundancy to each block for the receiver to be able to figure out what the original block was, rather than relying on a retransmission, which itself may be in error.

Error Correcting Codes:

- What is an error?
- Normally, a frame consists of **m** data bits and **r** redundant, or check, bits.
- The total length is **n (n=m+r)**
- An **n**-bit unit containing data and check bits is often referred to as an **n-bit codeword**.
- Given any two codeword (say, 10001001 and 10110001), it is possible to determine how many corresponding bits differ. As we see in this case 3 bits differ.
- To determine how many bits differ, just exclusive OR the two codewords and count the number of 1 bits in the result, for example:

```
10001001
10110001
--------------
00111000
```

ERROR DETECTION AND CORRECTION

Error Correcting Codes:

- The number of bit positions in which two codewords differ is called the **Hamming distance**.
- Its significance is that if two codewords are a Hamming distance **d** apart, it will require **d** single bit errors to convert one into the other.
- **2^m** possible data message are legal;
- But due to the way the check bits are computed, not all of the **2^n** possible codewords are used.
- It is possible to construct a complete list of legal codewords by given algorithm for computing the check bits, and from this find the two codewords whose Hamming distance is minimum.
- This distance is the Hamming distance of the complete code.
- The error-detecting and error –correcting properties of a code depend on its Hamming distance.
- To detect d errors, you need a distance d+1 code because with such a code there is no way that d single-bit errors can change a valid codeword.
- When the receiver sees an invalid codeword, it can tell that a transmission error has occurred.

ERROR DETECTION AND CORRECTION

Error Correcting Codes:

- Similarly, to correct d errors, you need a distance 2d+1 code because that way the legal codewords are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined.
- As a simple example of an error-detecting code, consider a code in which a single parity bit is appended to the data.
- The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd).
- For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100.
- With odd parity 1011010 becomes 10110101.
- A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity.
- It can be used to detect single errors.

Error Correcting Codes:

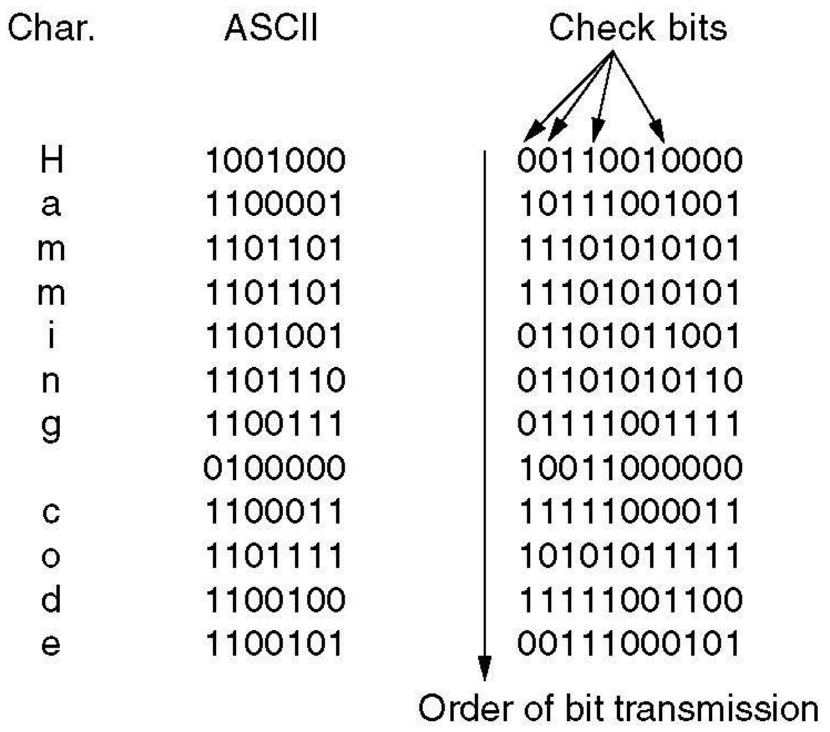
Example:

- Consider a code with only four valid codewords: 0000000000, 0000011111, 1111100000, 1111111111
- This code has a distance 5, which means that it can correct double errors, because of 5=2d+1.
- If codeword 0000000111 arrives, the receiver knows, that the original must have been 0000011111.
- If however a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.
- Imaging that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected.
- Each of the 2^m legal message has n illegal codewords at a distance 1 from it.
- These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it.
- Thus each of 2^m legal messages requires n+1 bit patterns dedicated to it.

Error Correcting Codes:

Example:

- Since the total number of bit patterns is 2^n, we must have $(n+1)2^m \leq 2^n$.
- Using $n=m+r$, this requirement becomes $(m+r+1) \leq 2^r$.
- Given m, this puts a lower limit on the number of check bits needed to correct single 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

Use of a Hamming code to correct burst errors.

ERROR DETECTION AND CORRECTION

Error Detecting Codes:

- Error correcting codes are widely used on wireless links, which are noisy and error prone when compared to copper wire or optical fibers.
- Without error correcting codes, it would be hard to get anything through.
- However over copper wire or fiber, the error rate is much lower, so error detection and retransmission is usually more efficient there for dealing with the occasional error.

Example:

- Consider a channel on which errors are isolated and the error rate is 10^{-6} per bit.
- Let the block size be 1000 bits.
- To provide error correction for 1000-bit block, 10 check bits are needed: a megabit of data would require 10,000 check bits.
- 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 only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

ERROR DETECTION AND CORRECTION

Error Detecting Codes:

- The polynomial code or CRC (Cyclic Redundancy Check) is in widespread use;
- Polynomial codes are based upon treating 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.
- Such a polynomial is said to be of degree k-1.
- For example: 110001 is $x^5+x^4+x^0$
- 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.
- For example:

```
  10011011    00110011     11110000    01010101
 +11001010   +11001101   -101001101   -10101111
 ---------   ---------   ----------   ---------
  01010001    1111110      01010110    11111010
```

Error Detecting Codes:

- When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, G(x), in advance.
- Both high- and low- order bits of the generator must be 1.
- To compute the checksum for some frame with m bits, corresponding to the polynomial M(x), the frame must be longer than the generator polynomial.
- The idea is to append a checksum to the end of the frame in such a way that the polynomial 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

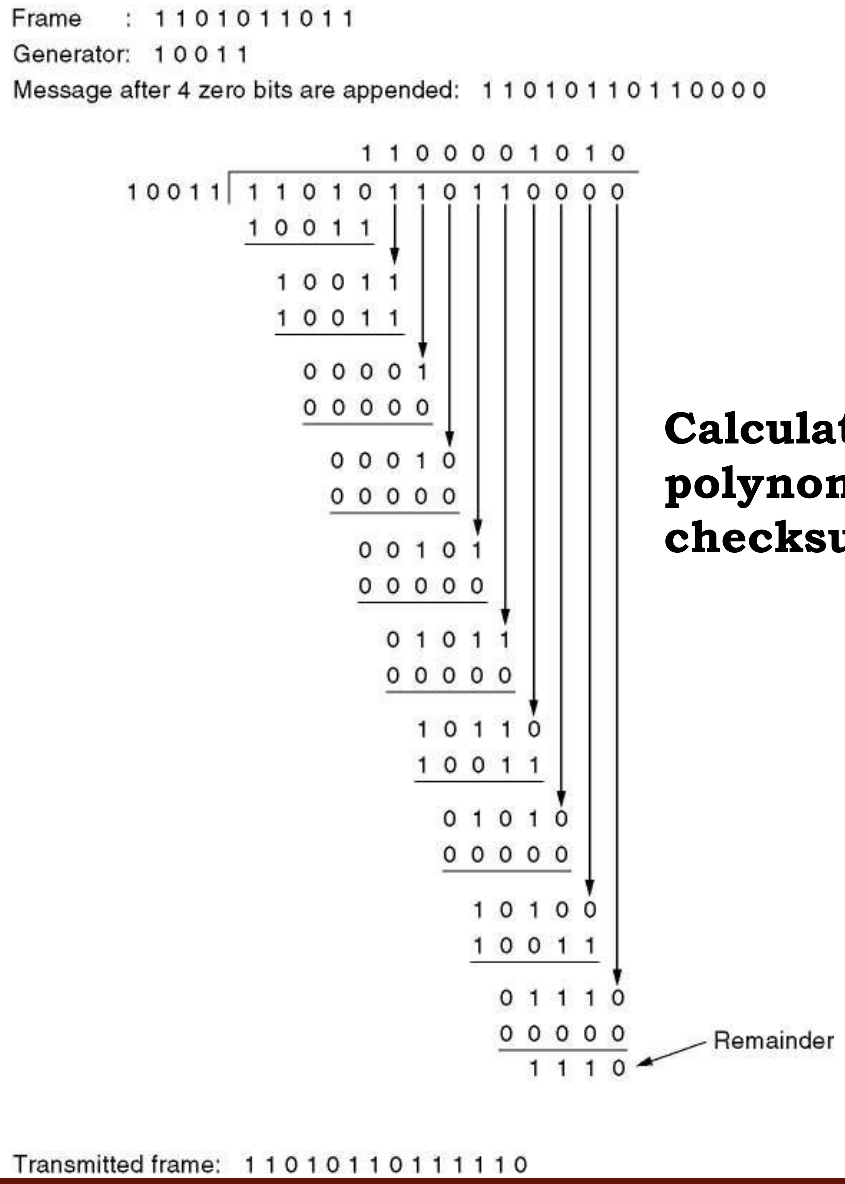
Error Detecting Codes:

- The algorithm for computing the checksum is as follows:
  1) Let r be the degree of G(x). Append r zero bits to the loworder end of the frame so it now contains m+r bits and corresponds to the polynomial $x^rM(x)$.
  2) Divide the bit string corresponding to G(x) into the bit string corresponding to $x^rM(x)$, using modulo 2 division.
  3) Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^rM(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).

ERROR DETECTION AND CORRECTION

Error Detecting Codes:

This example illustrates the calculation for a frame *M(t)= 1101011011* using the generator *G(x)=x4+x+1. r=4*
$x^r M(x)$*=11010110110000*
T(x)=11010110111110
E(x) is error polynomial


Frame : 1101011011
Generator: 10011
Message after 4 zero bits are appended: 11010110110000

Transmitted frame: 11010110111110

Calculation of the polynomial code checksum.

ERROR DETECTION AND CORRECTION

Error Detecting Codes:

- What kinds of errors will be detected?
- Imaging 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 0s and 1s, and a final 1, with all other bits being 0.
- Upon receiving the checksummed frame, the receiver divides it by G(x); that is, it computes [T(x)+E(x)]/G(x).
- T(x)/G(x) is 0, so the result of the computation is simply 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.

ELEMENTARY DATA LINK PROTOCOLS

- Three protocols of increasing complexity:
- A simulator for protocols is available via Web

  http://www.prenhall.com/tanenbaum

ASSUMPTION 1:

- In the physical layer (PhL), data link layer (DLL), and network layer (NL) are independent processes that communicate by passing messages back and forth.
- In many cases, PhL and DLL processes will be running on a processor inside a special network I/O chip and NL code will be running on the main CPU.
- However, other implementations are also possible:
- three processes inside a single I/O chip;
- or PhL and DLL as procedures called by NL process.
- In any event, treating the three layers as separate processes makes the discussion conceptually cleaner and also serves to emphasize the independence of the layers.

ELEMENTARY DATA LINK PROTOCOLS

ASSUMPTION 2:

- Machine A wants to send a long stream of data to machine B, using reliable, connection-oriented service.
- Later, we will consider the case where B also wants to send data to A simultaneously.
- A is assumed to have an infinite supply of data ready to send and never has to wait for data to be produced.
- Instead, when A's DLL asks for data, NL is always able to comply immediately. This restriction will be dropped later.

ELEMENTARY DATA LINK PROTOCOLS

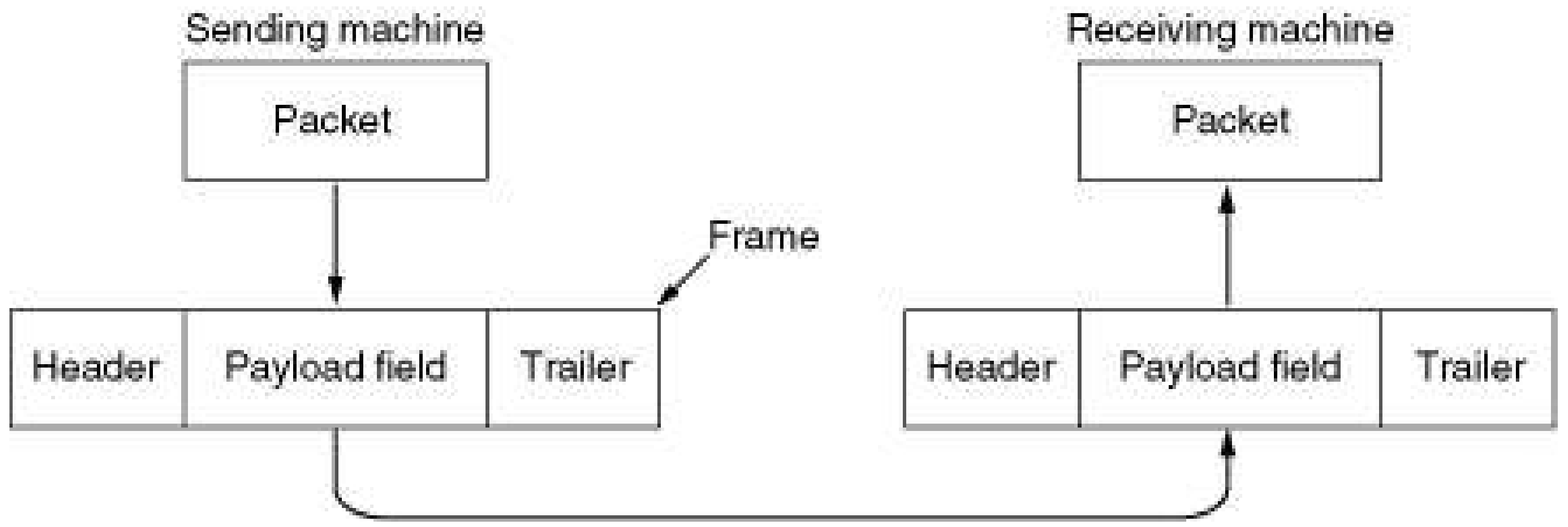
ASSUMPTION 3:

- Machines do not crash. That is these protocols deal with communication errors, but not the problems caused by computers crashing and rebooting.
- The packet passed across the interface to DLL from NL is pure data, whose every bit is to be delivered to the destination's NL.
- The fact that the destination's NL may interpret part of the packet as a header is of no concern to DLL.

ELEMENTARY DATA LINK PROTOCOLS

ASSUMPTION 3:

- When DLL accepts a packet, it encapsulates the packet in a frame by adding a data link header and trailer to it.



- Thus, a frame consists of an embedded packet, some control information (in the header), and a checksum (in the trailer).
- The frame is then transmitted to DLL on the other machine.

ELEMENTARY DATA LINK PROTOCOLS

Library procedures:

- to_physical_layer is for sending a frame
- from_physical_layer is for receiving a frame
- The transmitting hardware computes and appends the checksum (thus creating the trailer), so that DLL software need not worry about it.
- The polynomial algorithm discussed earlier in this chapter might be used, for example.
- Initially, the receiver just sits around waiting for something to happen (e.g., a frame has arrived).
- wait_for_event(&event) is for receiver to act
- Variable event tells what happened.
- For example, event=cksum_err means that the checksum is incorrect, there was a transmission error.
- event=frame_arrival means the inbound frame arrived undamaged.
- The set of possible events differs for the various protocols.

ELEMENTARY DATA LINK PROTOCOLS

Library procedures:

- Following slide shows some declarations (in C) common to many of protocols to be discussed later.
- Five data structures are defined there:
  a) boolean – is an enumerated type and can take on the values true and false.
  b) seq_nr is a small integer (0 - MAX_SEQ) used to number the frames so that we can tell them apart.
  c) packet is the unit of information exchanged between NL and DLL on the same machine, or between NL peers. In our model packet contains MAX_PKT bytes, but may be of variable length.
  d) frame_kind is wether there are any data in frame, because some of protocols distinguish frames containing only control information from those containing data as well.

ELEMENTARY DATA LINK PROTOCOLS

Library procedures:

e) frame is composed of four fields: kind, seq, ack, and info. The first three contain control information. These control fields are collectively called the frame header. A last one may contain actual data to be transferred.

- — kind – whether there are any data in the frame.
- — seq – for sequence numbers
- — ack – for acknowledgements
- — info – in data frame it contains a single packet

A number of procedures are also listed in figure.

ELEMENTARY DATA LINK PROTOCOLS

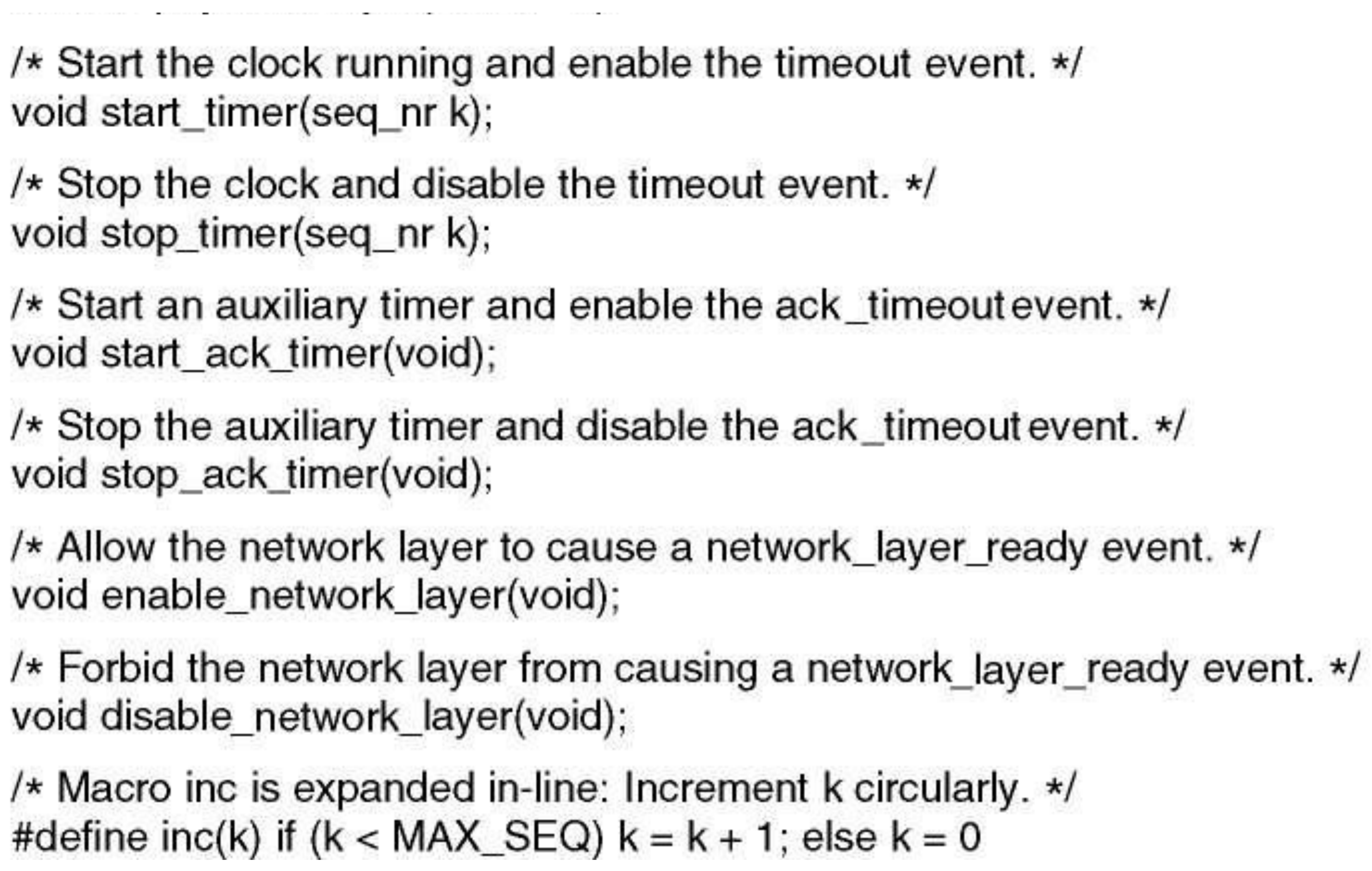
Library procedures:

```
#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 a frame is it? */
 seq_nr seq;                                      /* sequence number */
 seq_nr ack;                                      /* acknowledgement number */
 packet info;                                     /* the network layer packet */
} frame;

/* 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);
```

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

ELEMENTARY DATA LINK PROTOCOLS

Library procedures:



```
/* 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);

/* 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
```

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

ELEMENTARY DATA LINK PROTOCOLS

Library procedures:

- In most of the protocols, we assume that the channel is unreliable and loses entire frames upon occasion.
- To be able to recover from such calamities, the sending DLL must start an internal or clock whenever it sends a frame.
- If no reply has been received within a certain predetermined time interval, the clock times out and DLL receives an interrupt signal.
- In our protocols this is handled by allowing the procedure wait_for_event to return event=timeout.
- The procedures start_timer and stop_timer turn the timer on and off, respectively.
- The procedures start_ack_timer and stop_ack_timer control an auxiliary timer used to generate acknowledgements under certain conditions.

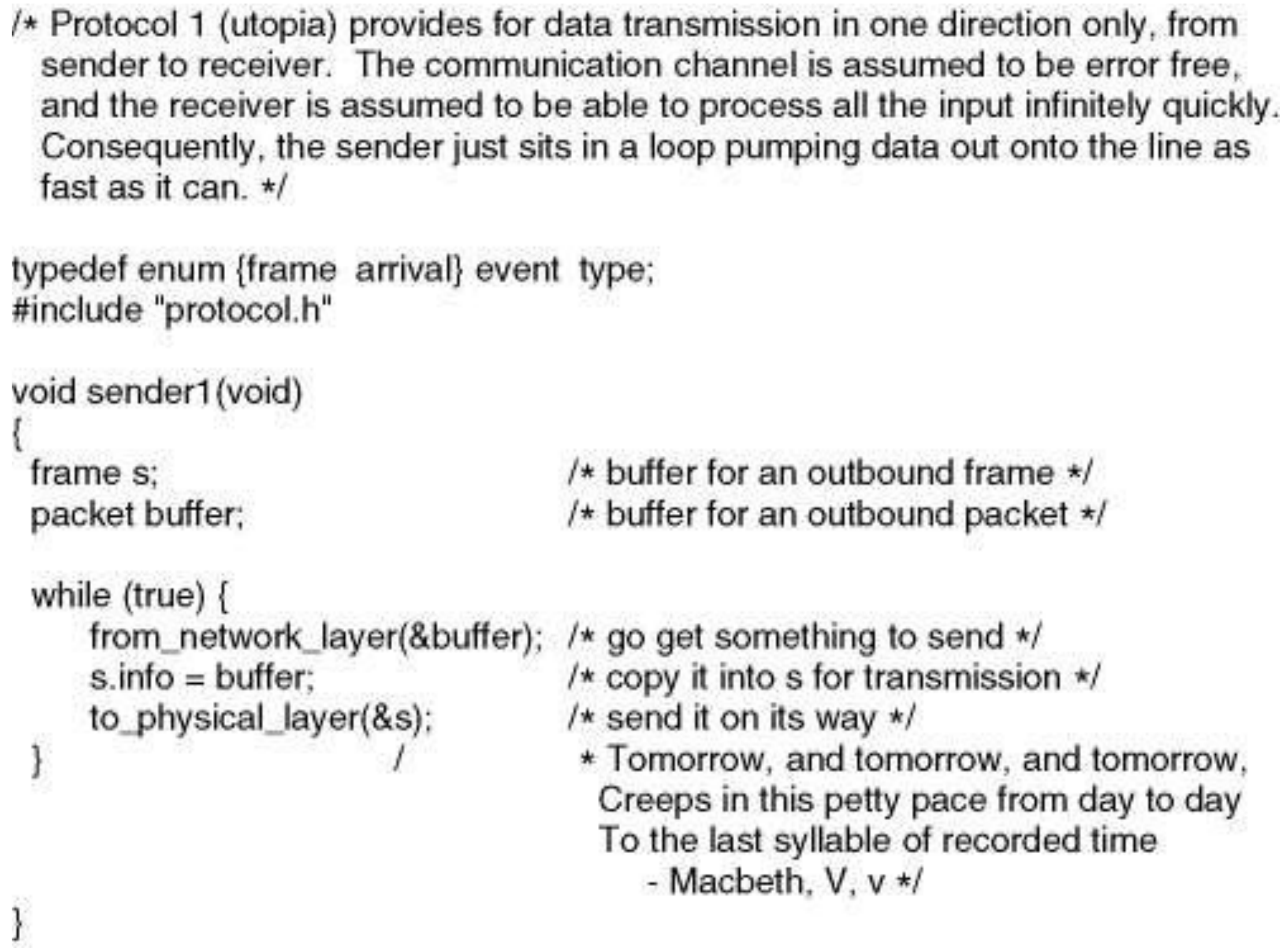
ELEMENTARY DATA LINK PROTOCOLS

- **PROTOCOL 1: An Unrestricted Simplex Protocol**
- **PROTOCOL 2: A Simplex Stop-and-Wait Protocol**
- **PROTOCOL 3: A Simplex Protocol for a Noisy Channel**

PROTOCOL 1: An Unrestricted Simplex Protocol

- Data are transmitted in one direction only.
- Both the transmitting and receiving network layers are always ready.
- Processing time can be ignored.
- Infinite buffer space is available.
- The communication channel between the data link layers never damages or loses frames.
- This is thoroughly unrealistic protocol and we nickname it as “utopia”.

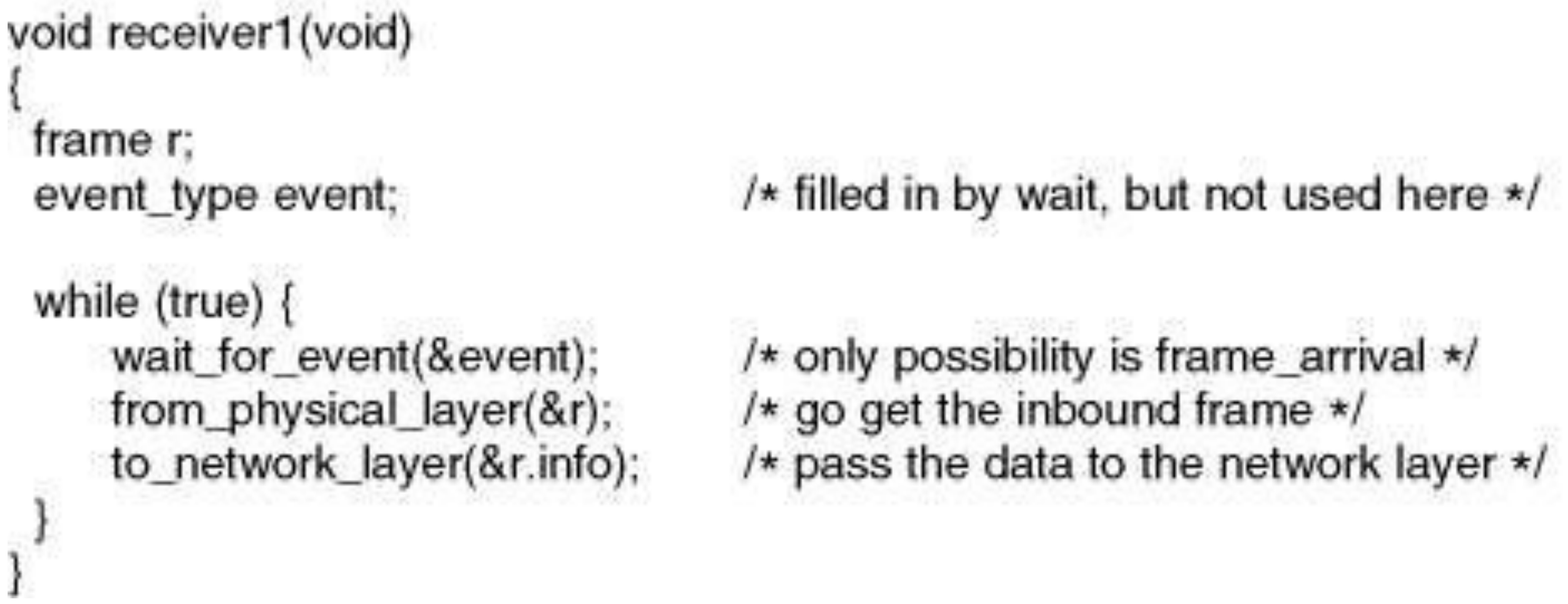
PROTOCOL 1: An Unrestricted Simplex Protocol



```
/* 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: An Unrestricted Simplex Protocol

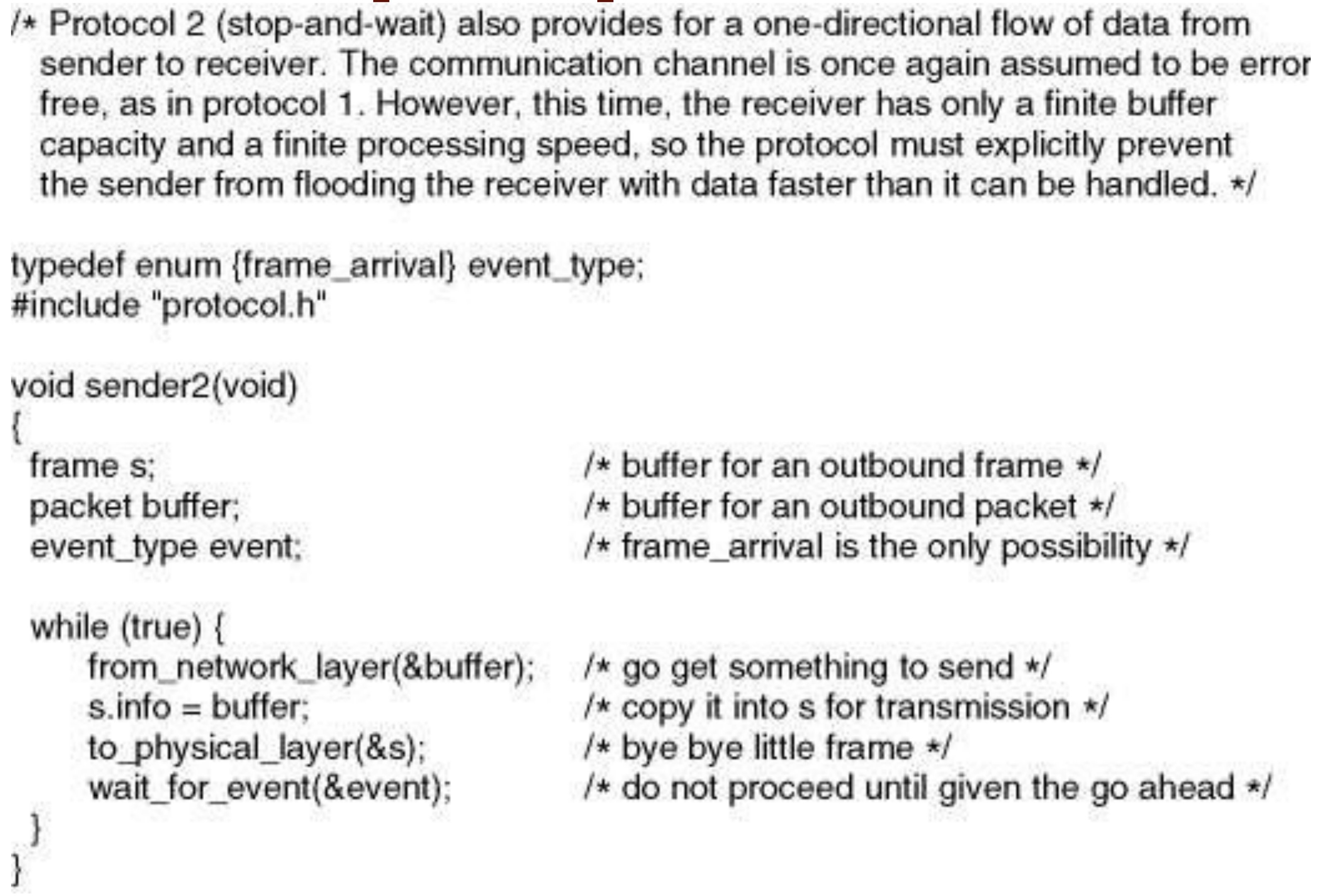


```
void receiver1(void)
{
 frame r;
 event_type event;                    /* filled in by wait, but not used here */

 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 */
 }
}
```

PROTOCOL 2: Simplex Stop-and-Wait Protocol

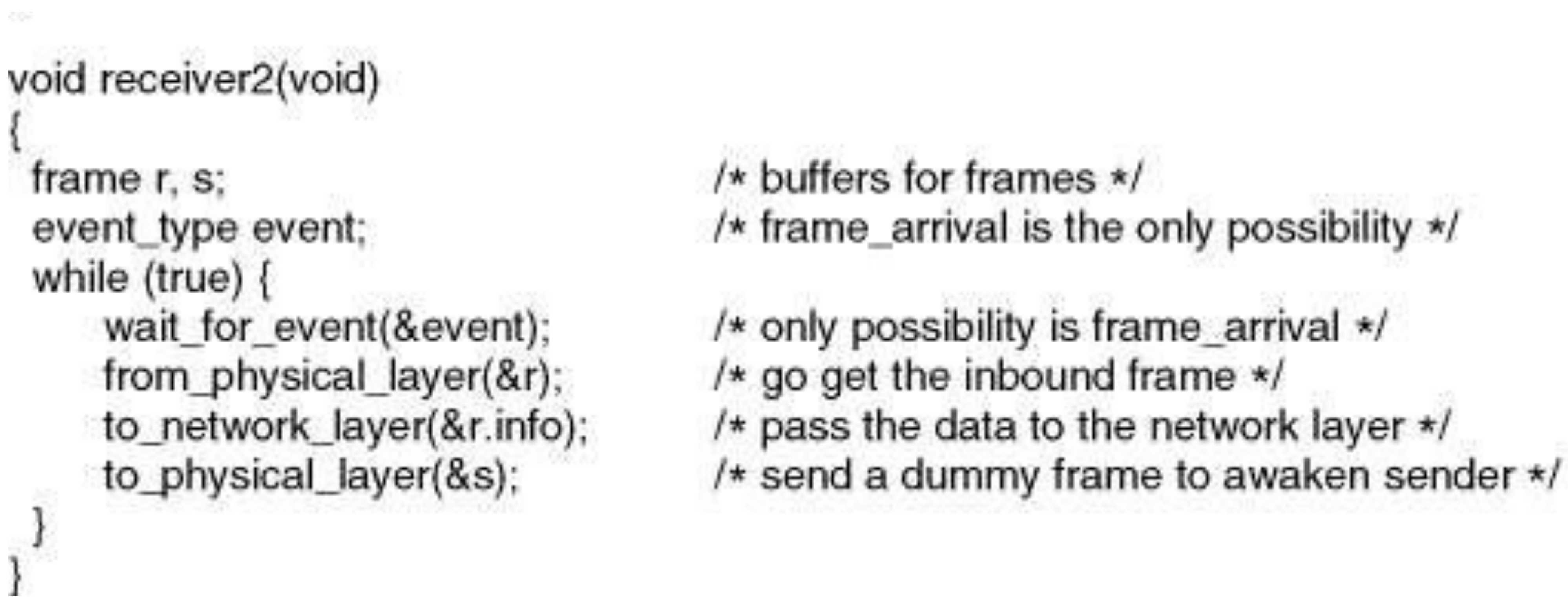
- Now we will drop the most unrealistic restriction used in Protocol 1: the ability of the receiving NL to process incoming data infinitely quickly.
- The communication channel is still assumed to be error free however, and the data traffic is still simplex.
- The main problem we have to deal with here is how to prevent the sender from flooding the receiver with data faster than the latter is able to process them.

PROTOCOL 2: Simplex Stop-and-Wait Protocol



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

PROTOCOL 2: Simplex Stop-and-Wait Protocol



```
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 */
  }
}
```

PROTOCOL 2: Simplex Stop-and-Wait Protocol

- If the receiver requires a time Δt to execute from_physical_layer plus to_network_layer, the sender must transmit at an average rate less than one frame per time Δt.
- If we assume that no automatic buffering and queuing are done within the receiver's hardware, the sender must never transmit a new frame until the old one has been fetched by from_physical_layer, let the new one overwrite the old one.
- Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait.

PROTOCOL 3: A Simplex Protocol for a Noisy Channel

- Frames may be either damaged or lost completely.
- However, we assume that if a frame is damaged in transit, the receiver hardware will detect this when it computes the checksum.
- If the frame is damaged in such a way that the checksum is nevertheless correct, an unlikely occurrence, this protocol (and all other protocols) can fail (i.e., deliver an incorrect packet to the network layer).
- A simple solution is to change Protocol 1 by adding timer such a way:
- The sender could send a frame, but the receiver would only send an acknowledgement frame if the data were correctly received.
- If a damaged frame arrived at the receiver, it would be discarded.
- After a while the sender would time out and send the frame again.
- This process would be repeated until the frame finally arrived intact.

PROTOCOL 3: A Simplex Protocol for a Noisy Channel

- The above scheme has a fatal flaw in it.
- The task of DLL processes is to provide error-free, transparent communication between NL processes.
- NL on machine A gives a series of packets to its DLL, which must ensure that an identical series of packets are delivered to NL on machine B by its DLL.
- In particular, NL on B has no way of knowing that a packet has been lost or duplicated, so DLL must guarantee that no combination of transmission errors, however unlikely, can cause a duplicate packet to be delivered to NL.

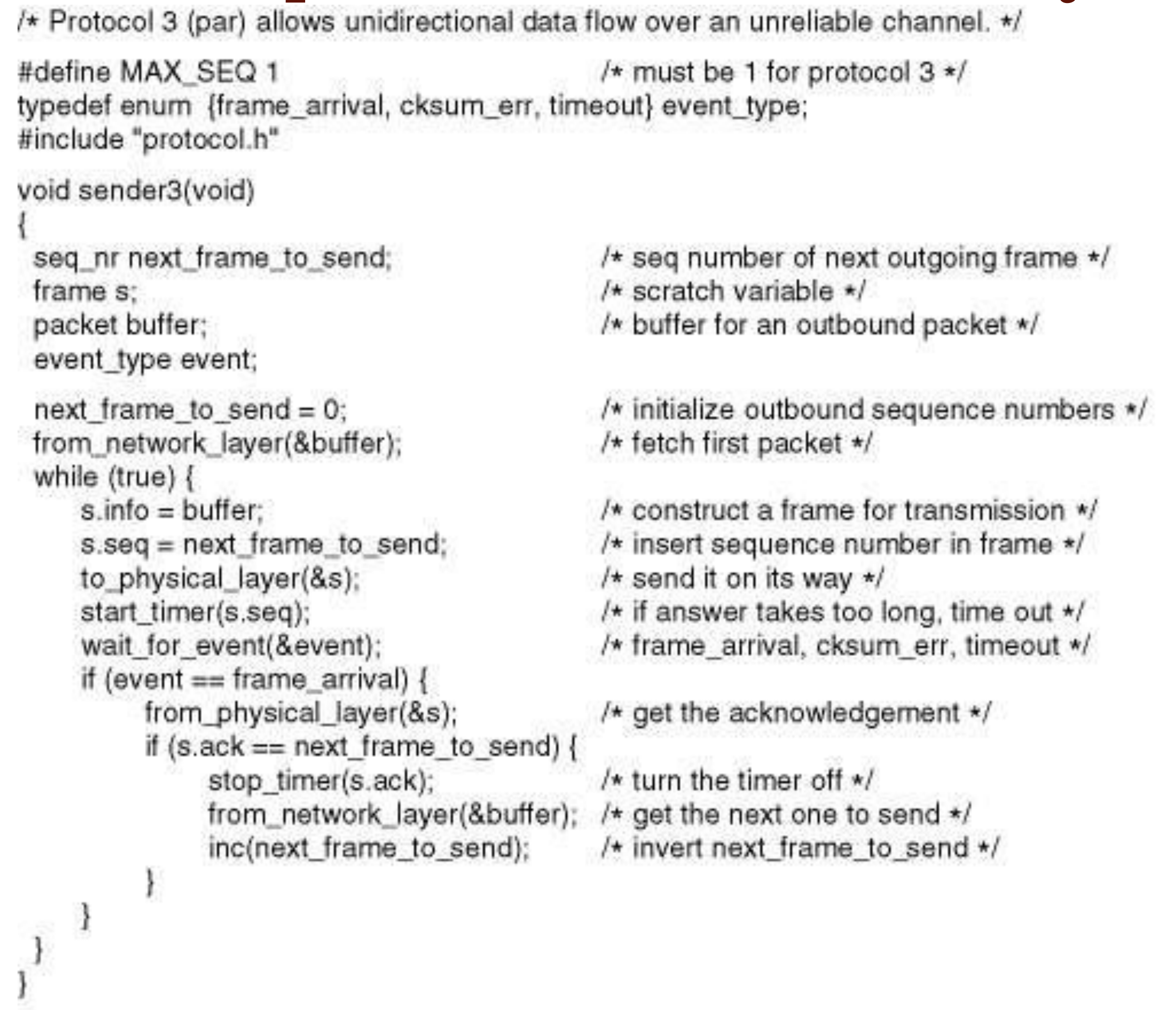
PROTOCOL 3: A Simplex Protocol for a Noisy Channel

Consider the following scenario:

1) NL on A gives packet 1 to its DLL. The packet is correctly received at B and passed to NL on B. B sends an acknowledgement frame back to A.
2) The acknowledgement frame gets lost completely. It just never arrives at all. Not only data frames but also control frames may be lost.
3) DLL on A eventually times out. Not having received an acknowledgement, it (incorrectly) assumes that its data frames was lost or damaged and sends the frame containing packet 1 again.
4) The duplicate frame also arrives at DLL on B perfectly and is unwittingly passed to NL there. If A is sending a file to B, part of the file will be duplicated (i.e., the copy of the file made by B will be incorrect and the error will not have been detected). In other words, the protocol will fail.

One solution: By sequence number in the header of each frame the receiver can check if it is a new frame or a duplicate to be discarded.

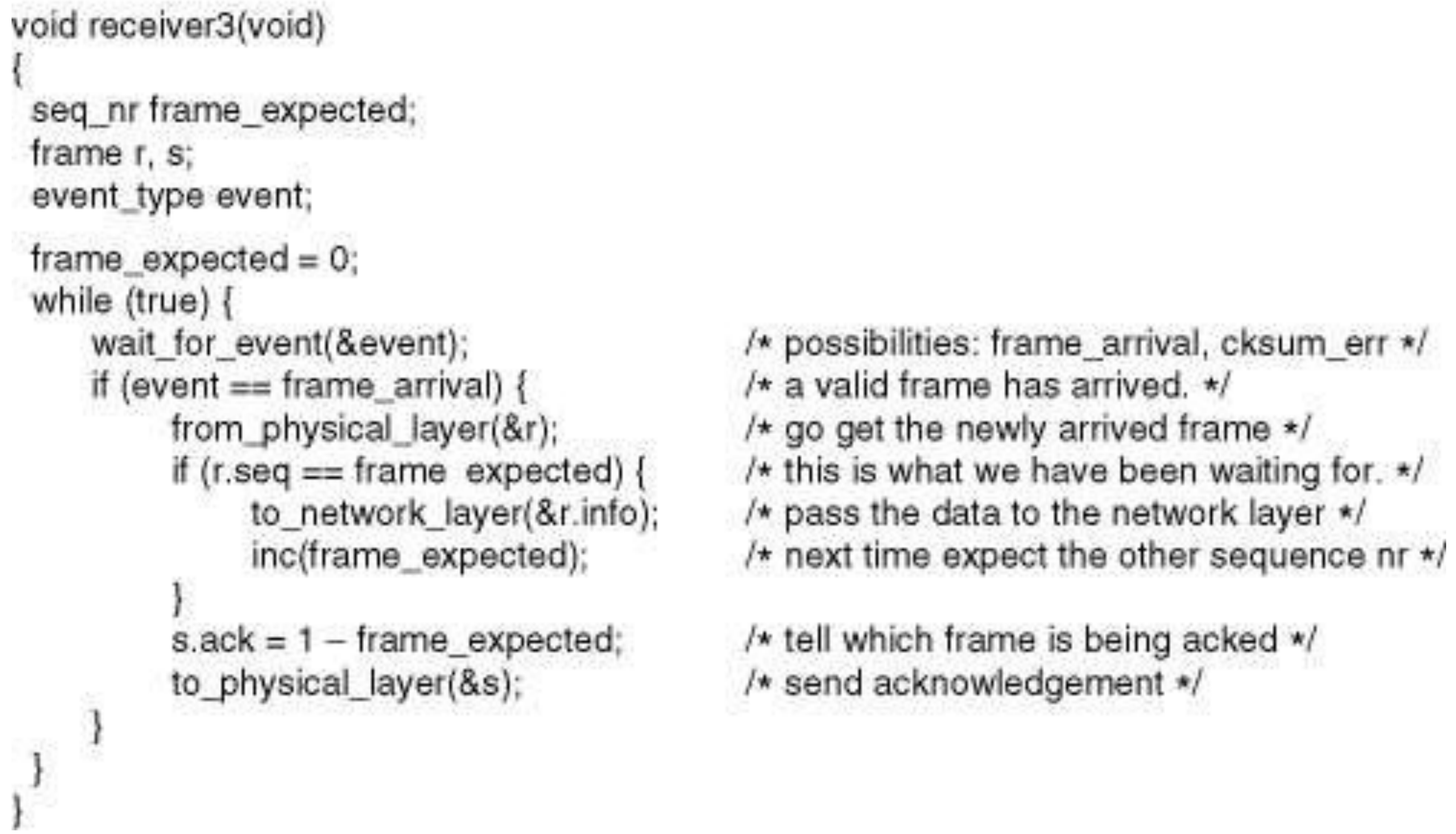
PROTOCOL 3: A Simplex Protocol for a Noisy Channel



```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */

#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) {
                    stop_timer(s.ack);             /* turn the timer off */
                    from_network_layer(&buffer);   /* get the next one to send */
                    inc(next_frame_to_send);       /* invert next_frame_to_send */
             }
      }
 }
}
```

A positive acknowledgement with retransmission protocol.

PROTOCOL 3: A Simplex Protocol for a Noisy Channel



```
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);             /* send acknowledgement */
      }
 }
}
```

A positive acknowledgement with retransmission protocol.

PROTOCOL 3: A Simplex Protocol for a Noisy Channel

- 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).
- Like protocol 2, this one also transmits data only in one direction.
- Protocol 3 differs from Protocol 2 in that both sender and receiver have a variable whose value is remembered while DLL is in the wait state.

SLIDING WINDOW PROTOCOLS

- In the previous protocols, data frames were transmitted in one direction only.
- In most practical situations, there is a need for transmitting data in both directions.
- One way of achieving full-duplex data transmission is to have two separate communication channels and use each one for simplex data traffic (in different directions).
- If this is done, we have two separate physical circuits, each with a “forward” channel (for data) and a “reverse” channel (for acknowledgements).
- In both cases the bandwidth of the reverse channel is almost entirely wasted.
- In effect, the user is paying for two circuits but using only the capacity of one.

SLIDING WINDOW PROTOCOLS

- Another way of achieving full-duplex data transmission is to use the same circuit for data in both directions.
- In protocols 2 and 3 “forward” and “reverse” channels were used and both of them have the same capacity.
- Now we will discuss the model in which the data frames from A to B are intermixed with the acknowledgement frames from A to B.
- By looking at the kind field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement.

SLIDING WINDOW PROTOCOLS

Piggybacking:

- Although interleaving data and control frames on the same circuit is an improvement over having two separate physical circuits, yet another improvement is possible.
- When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet.
- The ACK is attached to the outgoing data frame (using the ACK field in the frame header).
- In effect, ACK gets free ride on the next outgoing data frame.
- The technique of temporarily delaying outgoing ACK so that they can be hooked onto the next outgoing data frame is known as Piggybacking.

SLIDING WINDOW PROTOCOLS

Piggybacking:

- The principal advantage of using piggybacking over having distinct ACK frames is a better use of the available channel bandwidth.
- The ACK field in the frame header costs only a few bits, whereas a separate frame would need a header, the ACK, and a checksum.
- In addition, fewer frames sent means fewer "frame arrival" interrupts, and perhaps fewer buffers in the receiver, depending on how the receiver's software is organized.
- In the next protocol to be examined, the piggyback field costs only 1 bit in the frame header.
- It rarely costs more than a few bits.
- However, piggybacking introduces a complication not present with separate ACSs.
- How long should the data link layer wait for a packet onto which to piggyback the acknowledgement?

SLIDING WINDOW PROTOCOLS

Piggybacking:

- If the data link layer waits longer than the sender's timeout period, the frame will be retransmitted, defeating the whole purpose of having ACKs.
- If the data link layer were an oracle and could foretell the future, it would know when the next network layer packet was going to come in and could decide either to wait for it or send a separate ACK immediately, depending on how long the projected wait was going to be.
- Of course, the data link layer cannot foretell the future, so it must resort to some ad hoc scheme, such as waiting a fixed number of milliseconds.
- If a new packet arrives quickly, the ACK is piggybacked onto it; otherwise, if no packet has arrived by the end of this time period, the data link layer just sends a separate ACK frame.

SLIDING WINDOW PROTOCOLS

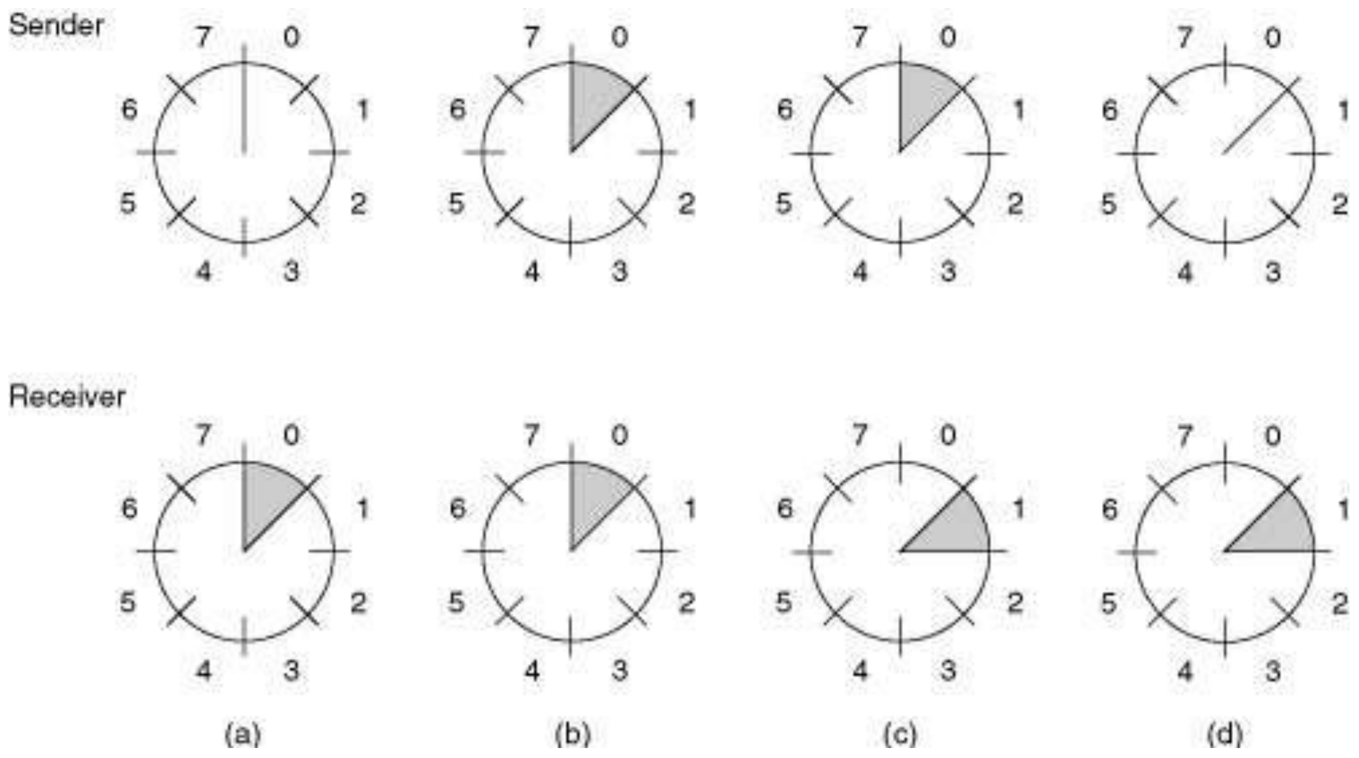
- The next three protocols are bidirectional protocols that belong to a class called sliding window protocols.
  - **PROTOCOL 4: A One-Bit Sliding Window Protocol**
  - **PROTOCOL 5: A Protocol Using Go Back N**
  - **PROTOCOL 6: A Protocol Using Selective Repeat**

SLIDING WINDOW PROTOCOLS

- The three differ among themselves in term of efficiency, complexity, and buffer requirements.
- In all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum.
- The maximum is usually 2^n-1 so the sequence number fits exactly in an n-bit field.
- The stop-and-wait sliding window protocol uses n=1, restricting the sequence number to 0 and 1.
- The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send.
- These frames are said to fall within the sending window.
- Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept.

SLIDING WINDOW PROTOCOLS

- The sender's window and the receiver's window need not have the same lower and upper limits or even have the same size.
- In some protocols they are fixed in size, but in others they can grow or shrink over the course of time as frames are sent and received.



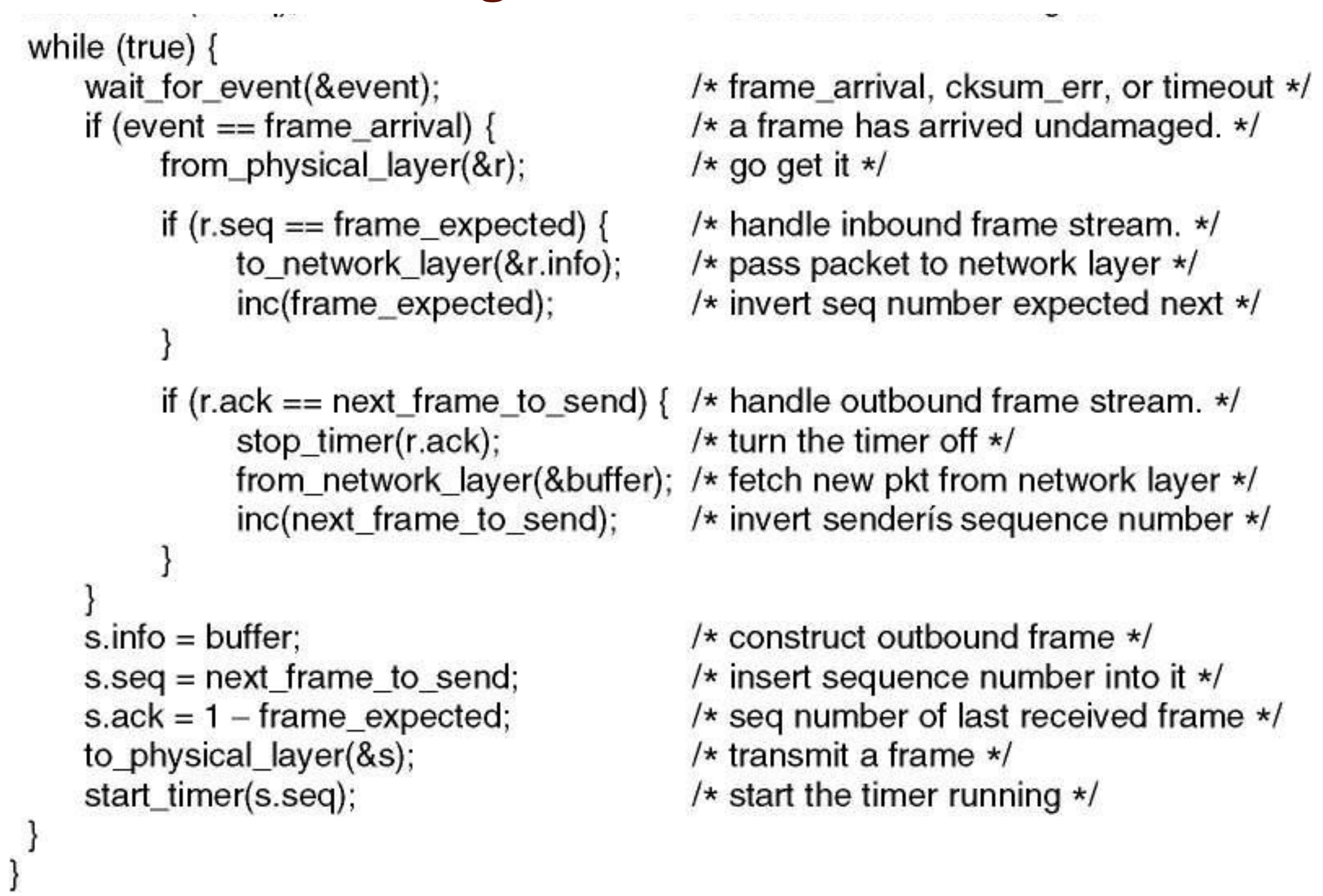
A sliding window of size 1, with a 3-bit sequence number.
(a) Initially.
(b) After the first frame has been sent.
(c) After the first frame has been received.
(d) After the first acknowledgement has been received.

PROTOCOL 4: A One-Bit Sliding Window Protocol

```
/* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1                                  /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                       /* 0 or 1 only */
  seq_nr frame_expected;                           /* 0 or 1 only */
  frame r, s;                                      /* scratch variables */
  packet buffer;                                   /* current packet being sent */
  event_type event;
  next_frame_to_send = 0;                          /* next frame on the outbound stream */
  frame_expected = 0;                              /* frame expected next */
  from_network_layer(&buffer);                     /* fetch a packet from the network layer */
  s.info = buffer;                                 /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                      /* insert sequence number into frame */
  s.ack = 1 – frame_expected;                      /* piggybacked ack */
  to_physical_layer(&s);                           /* transmit the frame */
  start_timer(s.seq);                              /* start the timer running */
```

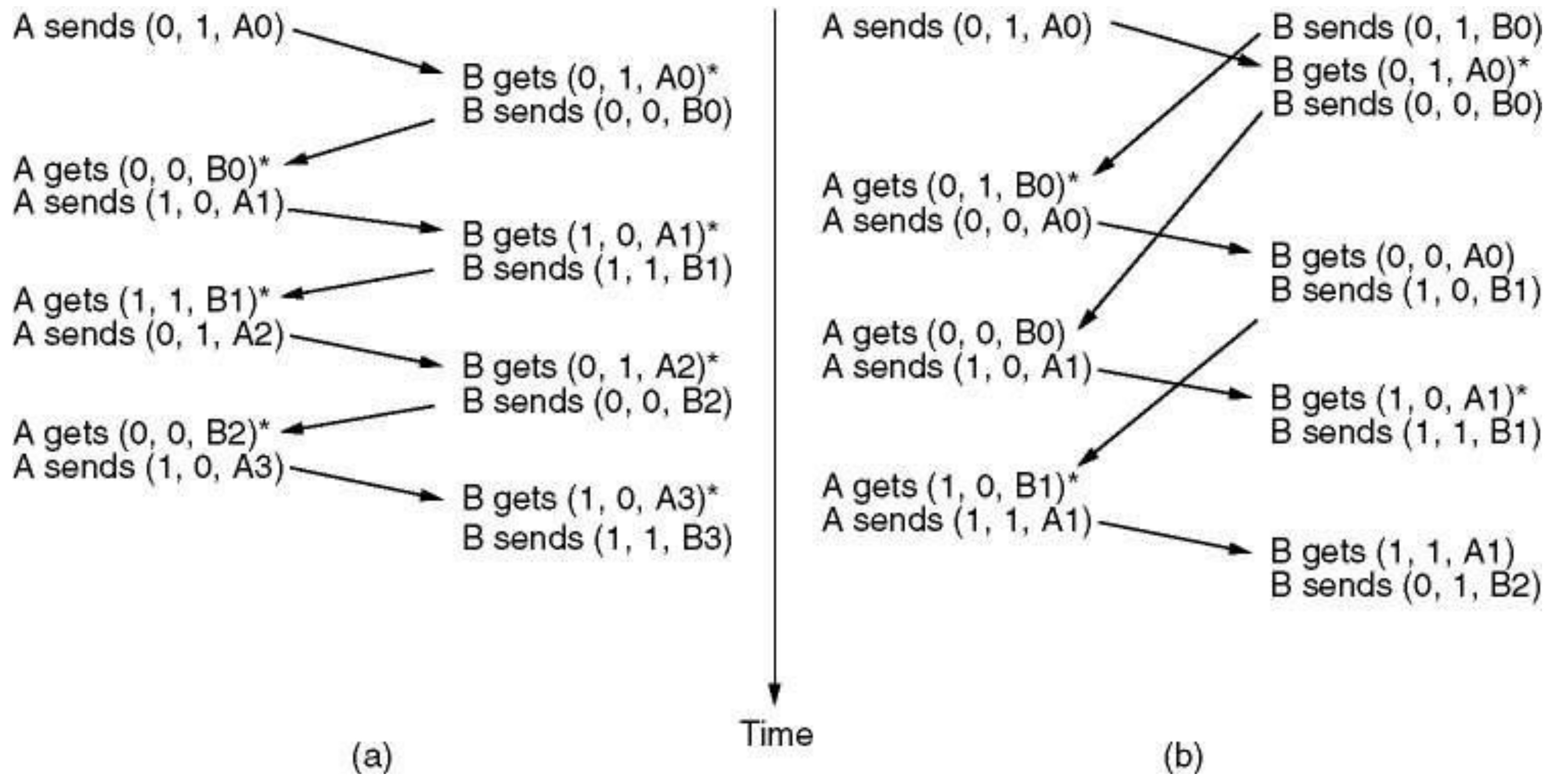
SLIDING WINDOW PROTOCOLS

PROTOCOL 4: A One-Bit Sliding Window Protocol



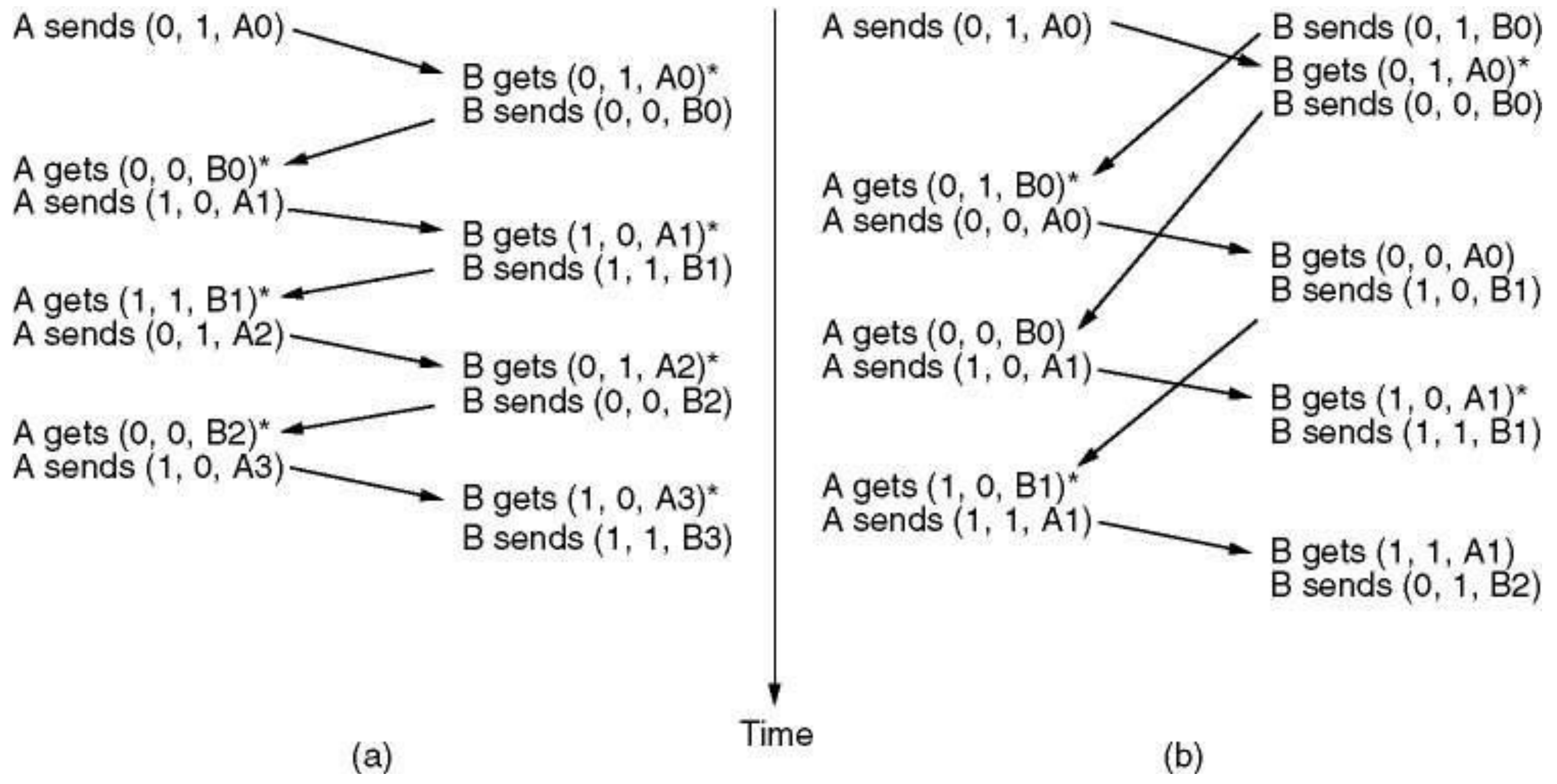
```
while (true) {
     wait_for_event(&event);              /* frame_arrival, cksum_err, or timeout */
     if (event == frame_arrival) {        /* a frame has arrived undamaged. */
          from_physical_layer(&r);        /* go get it */

          if (r.seq == frame_expected) {  /* handle inbound frame stream. */
               to_network_layer(&r.info); /* pass packet to network layer */
               inc(frame_expected);       /* invert seq number expected next */
          }

          if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
               stop_timer(r.ack);         /* turn the timer off */
               from_network_layer(&buffer); /* fetch new pkt from network layer */
               inc(next_frame_to_send);   /* invert senderís sequence number */
          }
     }
     s.info = buffer;                     /* construct outbound frame */
     s.seq = next_frame_to_send;          /* insert sequence number into it */
     s.ack = 1 – frame_expected;          /* seq number of last received frame */
     to_physical_layer(&s);               /* transmit a frame */
     start_timer(s.seq);                  /* start the timer running */
}
}
```

PROTOCOL 4: A One-Bit Sliding Window Protocol



Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

PROTOCOL 4: A One-Bit Sliding Window Protocol



Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

SLIDING WINDOW PROTOCOLS

Pipelining

- Imagine in the above example, if the sender could send more frames without waiting for the ack for the first one, the channel could have been utilized more efficiently. This technique is called pipelining.
- Imagine the scenario, when the sender always has something to send until the ack for the first frame arrives .. Best scenario, right?
- Clearly this is not Stop-n Wait.

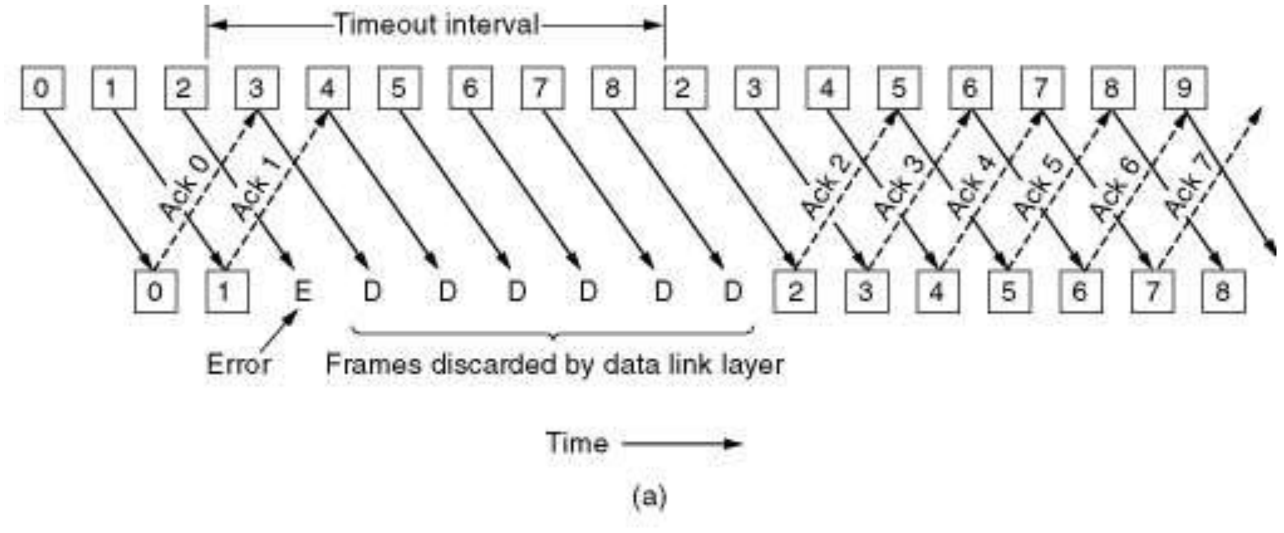
Pipelining

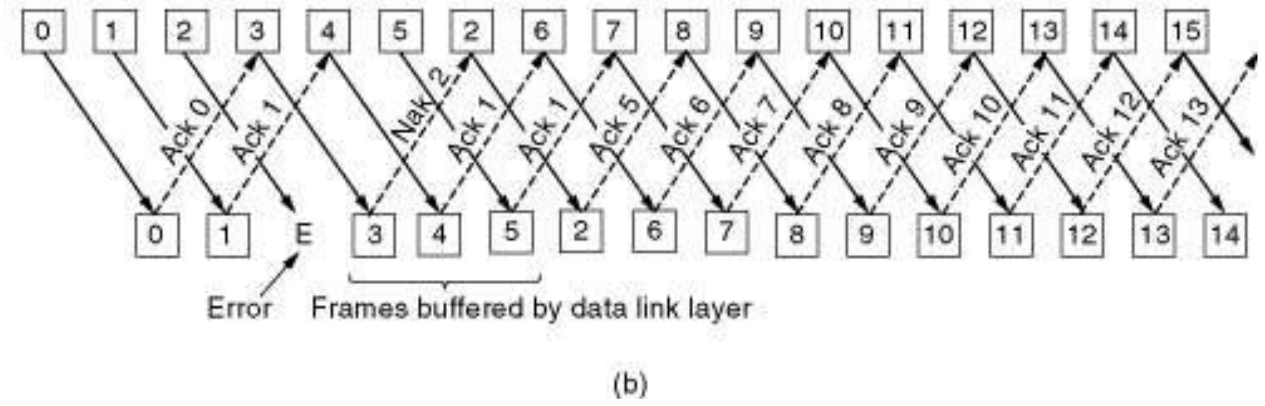
- Let
- Channel capacity is b bits/sec
- Frame size l bits,
- Round trip propagation time is R
- The time required to transmit a single frame is l/b.
- So, in stop n wait, the line is busy for l/b time and idle for R time, thus the line utilization is
- = l/b divided by l/b + R = l/(l + bR).
- If l < bR, the efficiency will be less than 50%.

Pipelining over noisy channels

- What if one or more frames is errored.
- Two options :
  - ✓ Throw away all the subsequent frames and go back and start resending from the damaged frame. This corresponds to size 1 window in the receiver. - Go back N Protocol
  - ✓ Buffer the subsequent frames and wait for the damaged frame to be resent, once that comes handover the frames in proper order to the NL-Selective Repeat Protocol

PROTOCOL 5: A Protocol Using Go Back N





Pipelining and error recovery. Effect on an error when
(a) Receiver's window size is 1.
(b) Receiver's window size is large.

SLIDING WINDOW PROTOCOLS

PROTOCOL 5: A Protocol Using Go Back N

Issues

- For a maxseq = 7, Number of outstanding frames can only be 7 and not 8. To see why, let us consider what happens when 8 frames are allowed to be outstanding .. That is the scene is
- Sender has sent out 8 frames
- The piggybacked ack for frame 7 finally comes to the sender.
- Now suppose the sender has 8 outstanding frames, 0…7
- Now sender sends next frame 0, if it is lost, if B has a frame to send, B will send the piggybacked ack for 7.

PROTOCOL 5: A Protocol Using Go Back N

Issues

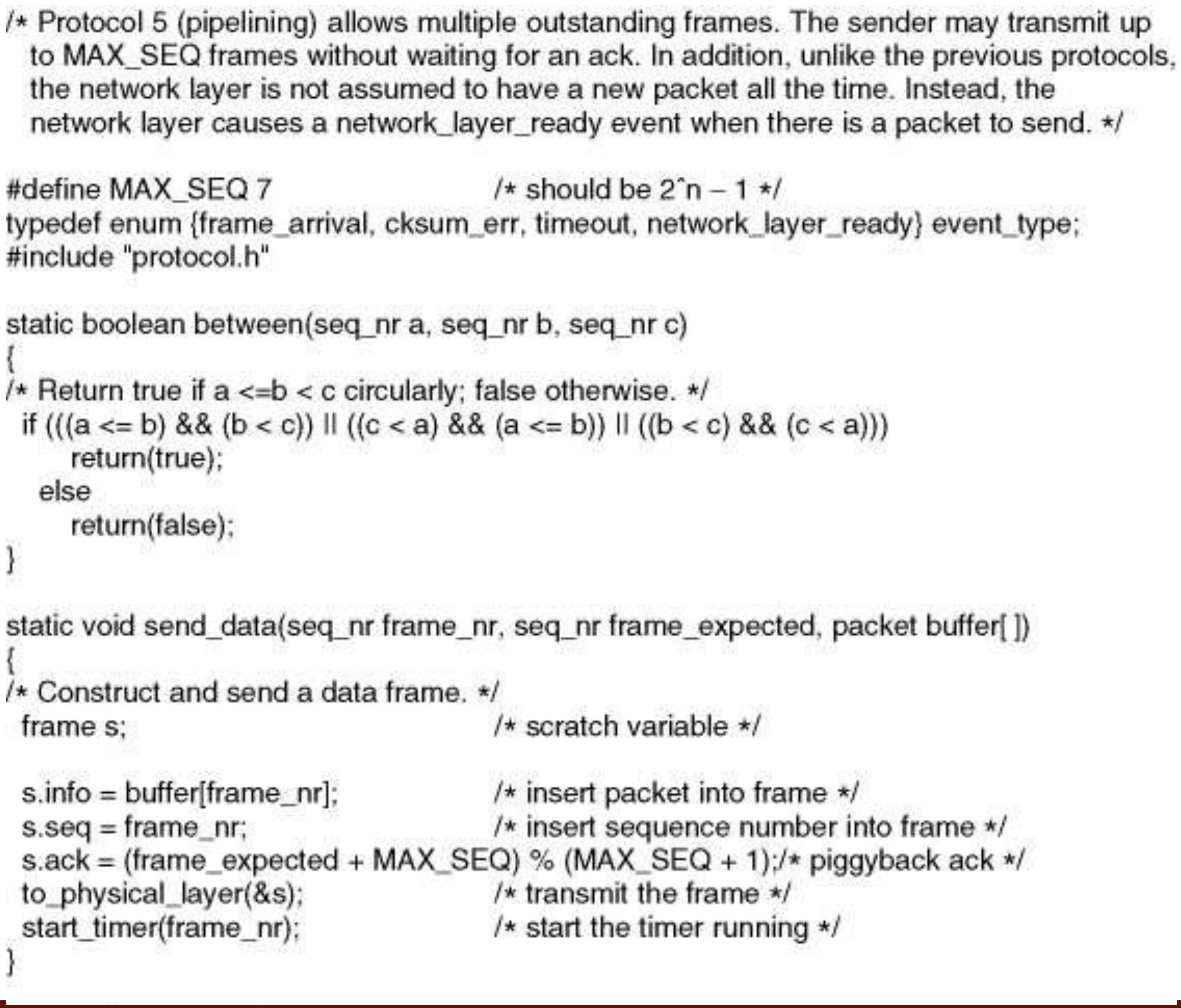
- Next, the sender sends the next frame 1 without waiting for the ack for 0, now suppose this is also lost, again B has a frame to send,then B will again send ack for 7.
- And so on .. Sender sends all the outstanding frames, 0…7
- Suppose all of them are lost, the receiver will still send the ack of the last frame received I.e. 7.
- Now sender has no way of knowing, whether all the frames were lost or since it has received ack for 7..all the frames reached intact.

PROTOCOL 5: A Protocol Using Go Back N

Solution

- The problem is solved by keeping at most 7 instead of 8 outstanding frames, say
  - ✓ 0-6 in the first batch, and
  - ✓ 7,0-5 in the second
- If all the frames in the second batch are lost, sender may keep on receiving ack for frame 6 which is not in sender's window and hence it will know that frames were lost.

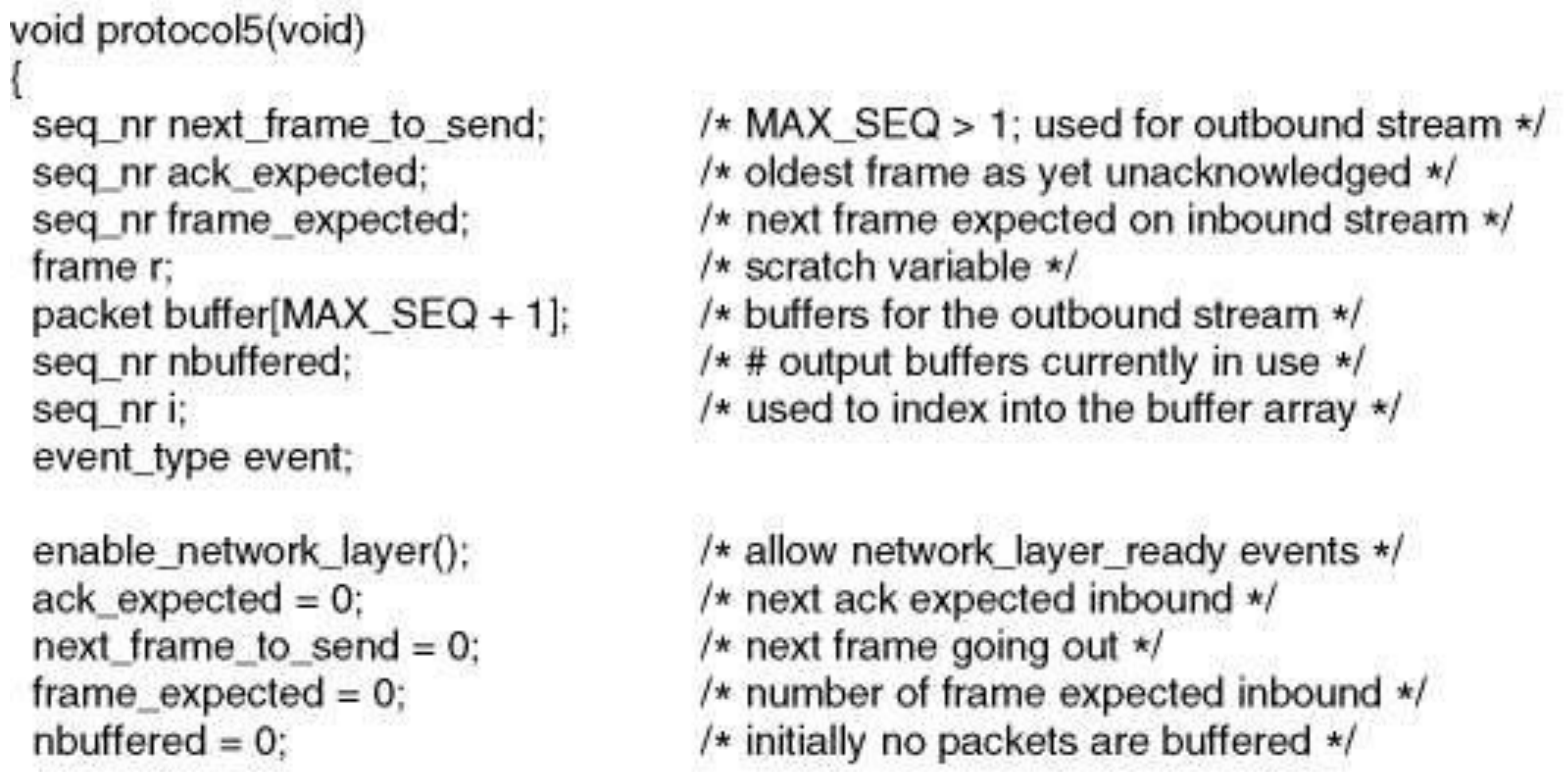
PROTOCOL 5: A Protocol Using Go Back N



```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike the previous protocols,
   the network layer is not assumed to have a new packet all the time. Instead, the
   network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7                          /* should be 2^n – 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <=b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
      return(true);
    else
      return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data frame. */
  frame s;                                 /* scratch variable */

  s.info = buffer[frame_nr];               /* insert packet into frame */
  s.seq = frame_nr;                        /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);                   /* transmit the frame */
  start_timer(frame_nr);                   /* start the timer running */
}
```

SLIDING WINDOW PROTOCOLS

PROTOCOL 5: A Protocol Using Go Back N



```
void protocol5(void)
{
  seq_nr next_frame_to_send;          /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;              /* next frame expected on inbound stream */
  frame r;                            /* scratch variable */
  packet buffer[MAX_SEQ + 1];         /* buffers for the outbound stream */
  seq_nr nbuffered;                   /* # output buffers currently in use */
  seq_nr i;                           /* used to index into the buffer array */
  event_type event;

  enable_network_layer();             /* allow network_layer_ready events */
  ack_expected = 0;                   /* next ack expected inbound */
  next_frame_to_send = 0;             /* next frame going out */
  frame_expected = 0;                 /* number of frame expected inbound */
  nbuffered = 0;                      /* initially no packets are buffered */
```

SLIDING WINDOW PROTOCOLS

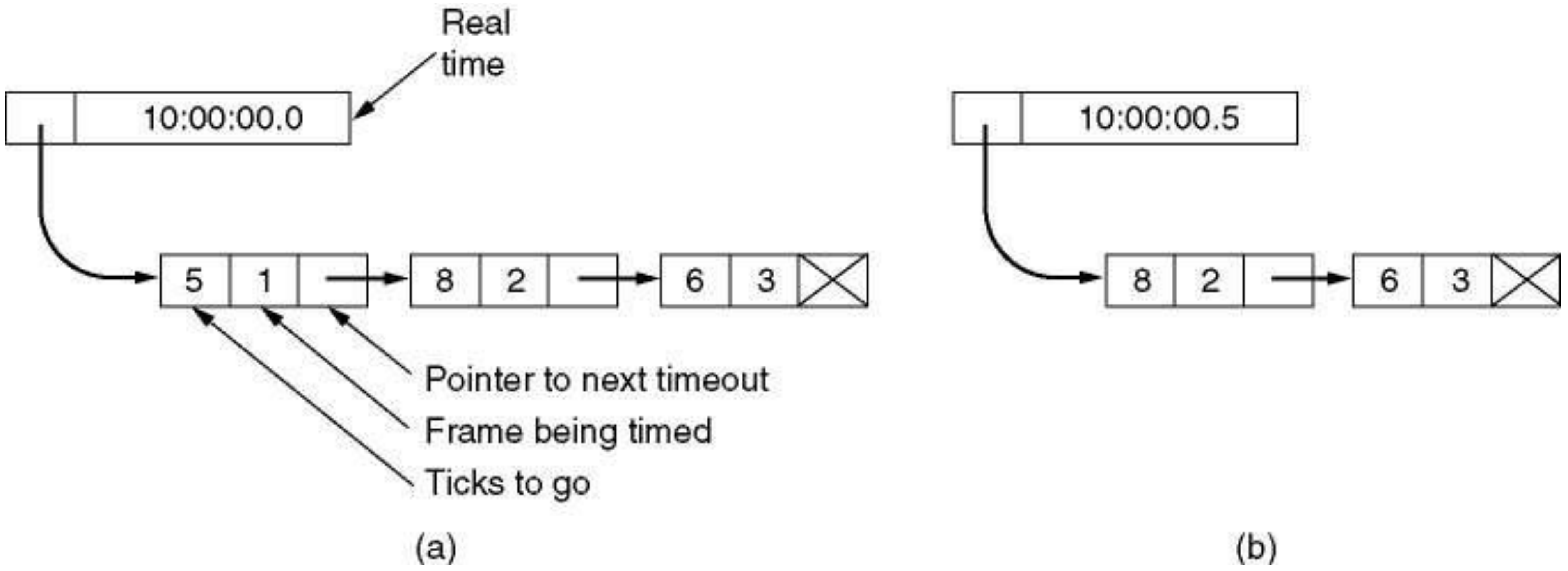
PROTOCOL 5: A Protocol Using Go Back N

```
while (true) {
  wait_for_event(&event);              /* four possibilities: see event_type above */

  switch(event) {
    case network_layer_ready:          /* the network layer has a packet to send */
         /* Accept, save, and transmit a new frame. */
         from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
         nbuffered = nbuffered + 1;    /* expand the sender's window */
         send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
         inc(next_frame_to_send);      /* advance sender's upper window edge */
         break;

    case frame_arrival:                /* a data or control frame has arrived */
         from_physical_layer(&r);      /* get incoming frame from physical layer */

         if (r.seq == frame_expected) {
              /* Frames are accepted only in order. */
              to_network_layer(&r.info);  /* pass packet to network layer */
              inc(frame_expected);     /* advance lower edge of receiver's window */
         }
```

PROTOCOL 5: A Protocol Using Go Back N

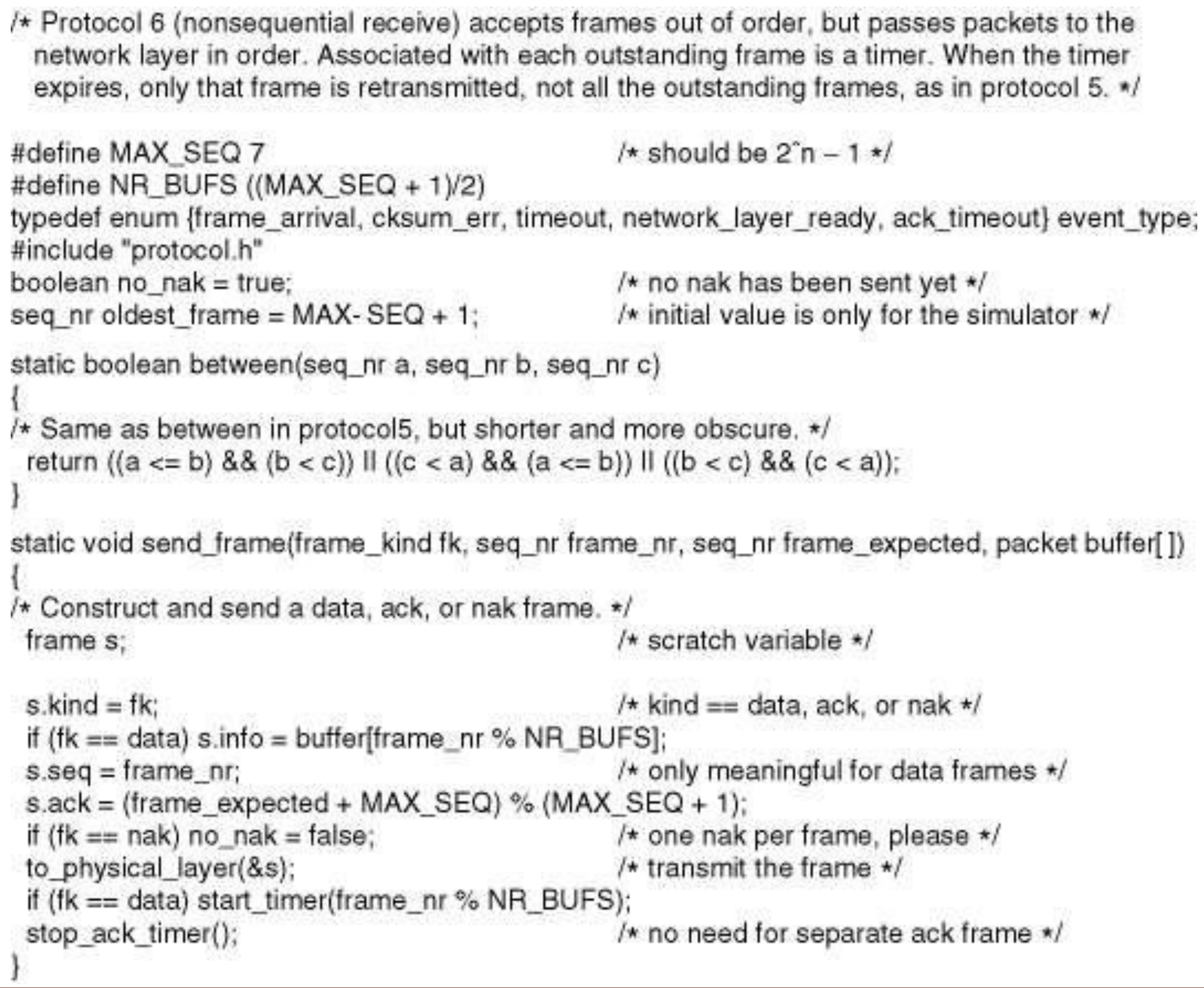
```
                /* Ack n implies n – 1, n – 2, etc.  Check for this. */
                while (between(ack_expected, r.ack, next_frame_to_send)) {
                     /* Handle piggybacked ack. */
                     nbuffered = nbuffered    1; /* one frame fewer buffered */
                     stop_timer(ack_expected); /* frame arrived intact; stop timer */
                     inc(ack_expected);      /* contract sender's window */
                }
                break;

           case cksum_err: break;            /* just ignore bad frames */

           case timeout:                     /* trouble; retransmit all outstanding frames */
                next_frame_to_send = ack_expected;     /* start retransmitting here */
                for (i = 1; i <= nbuffered; i++) {
                     send_data(next_frame_to_send, frame_expected, buffer);/* resend 1 frame */
                     inc(next_frame_to_send);  /* prepare to send the next one */
                }

         }

         if (nbuffered < MAX_SEQ)
                enable_network_layer();
         else
                disable_network_layer();
      }
    }
```

PROTOCOL 5: A Protocol Using Go Back N



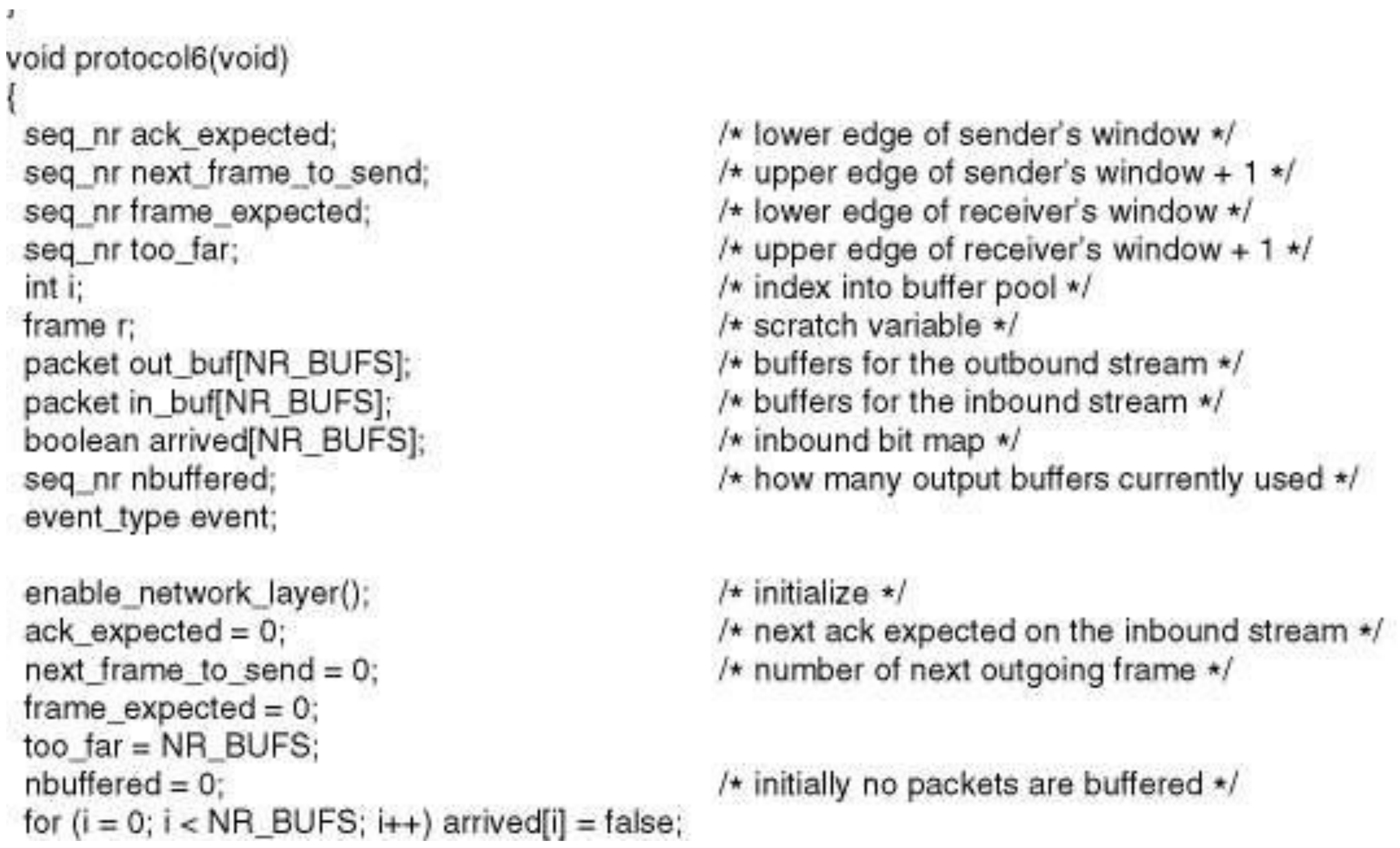
Simulation of multiple timers in software.

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat



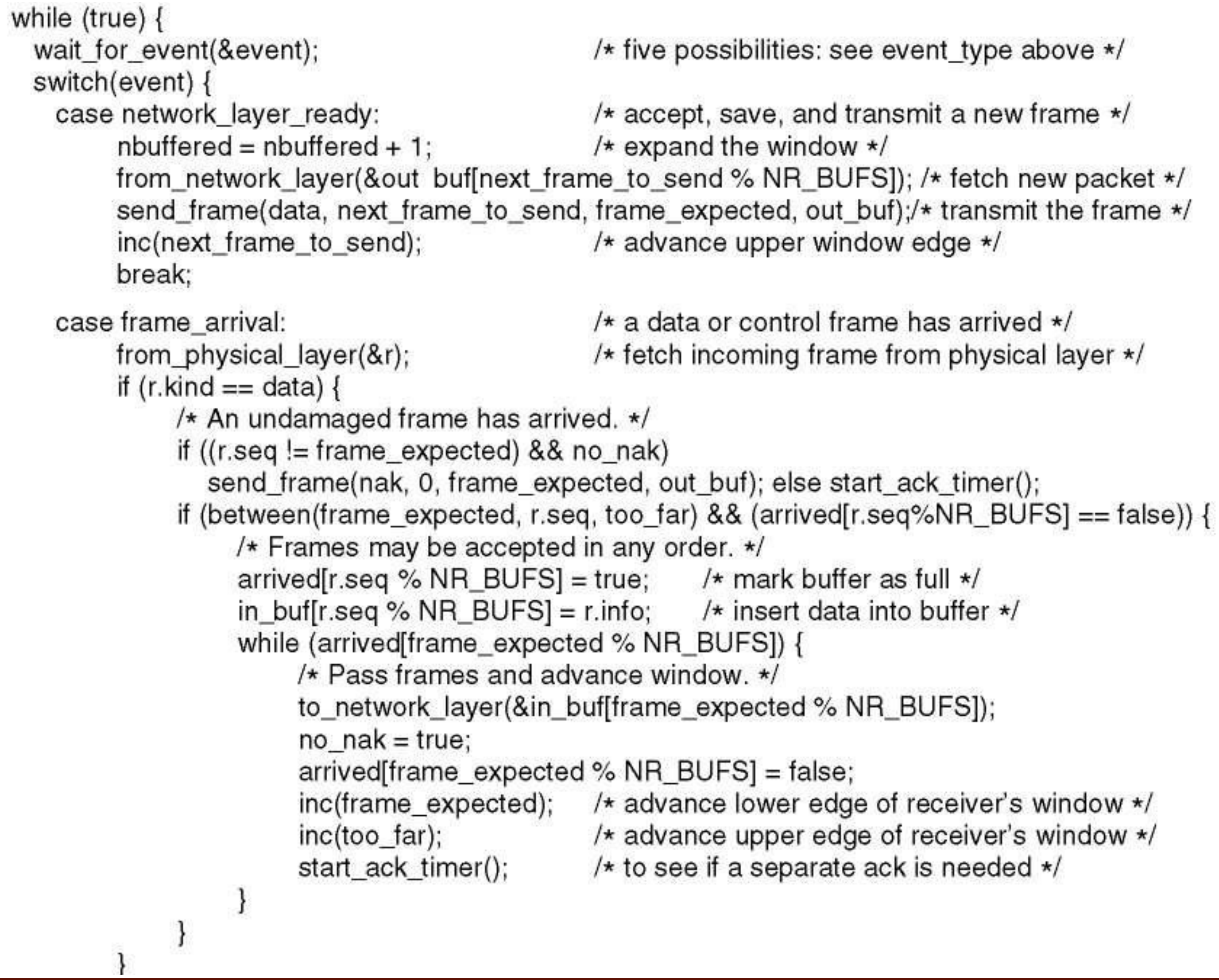
```
/* Protocol 6 (nonsequential receive) accepts frames out of order, but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                                  /* should be 2^n – 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                             /* no nak has been sent yet */
seq_nr oldest_frame = MAX- SEQ + 1;                /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                                         /* scratch variable */

  s.kind = fk;                                     /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                                /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;                   /* one nak per frame, please */
  to_physical_layer(&s);                           /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                                /* no need for separate ack frame */
}
```

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat



```
}
void protocol6(void)
{
  seq_nr ack_expected;                        /* lower edge of sender's window */
  seq_nr next_frame_to_send;                  /* upper edge of sender's window + 1 */
  seq_nr frame_expected;                      /* lower edge of receiver's window */
  seq_nr too_far;                             /* upper edge of receiver's window + 1 */
  int i;                                      /* index into buffer pool */
  frame r;                                    /* scratch variable */
  packet out_buf[NR_BUFS];                    /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];                     /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];                   /* inbound bit map */
  seq_nr nbuffered;                           /* how many output buffers currently used */
  event_type event;

  enable_network_layer();                     /* initialize */
  ack_expected = 0;                           /* next ack expected on the inbound stream */
  next_frame_to_send = 0;                     /* number of next outgoing frame */
  frame_expected = 0;
  too_far = NR_BUFS;
  nbuffered = 0;                              /* initially no packets are buffered */
  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat



```
while (true) {
  wait_for_event(&event);                  /* five possibilities: see event_type above */
  switch(event) {
    case network_layer_ready:              /* accept, save, and transmit a new frame */
         nbuffered = nbuffered + 1;        /* expand the window */
         from_network_layer(&out buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
         send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
         inc(next_frame_to_send);          /* advance upper window edge */
         break;

    case frame_arrival:                    /* a data or control frame has arrived */
         from_physical_layer(&r);          /* fetch incoming frame from physical layer */
         if (r.kind == data) {
              /* An undamaged frame has arrived. */
              if ((r.seq != frame_expected) && no_nak)
                  send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
              if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
                   /* Frames may be accepted in any order. */
                   arrived[r.seq % NR_BUFS] = true;      /* mark buffer as full */
                   in_buf[r.seq % NR_BUFS] = r.info;     /* insert data into buffer */
                   while (arrived[frame_expected % NR_BUFS]) {
                        /* Pass frames and advance window. */
                        to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                        no_nak = true;
                        arrived[frame_expected % NR_BUFS] = false;
                        inc(frame_expected);     /* advance lower edge of receiver's window */
                        inc(too_far);            /* advance upper edge of receiver's window */
                        start_ack_timer();       /* to see if a separate ack is needed */
                   }
              }
         }
```

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat

```
            if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next frame to send))
                 send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

            while (between(ack_expected, r.ack, next_frame_to_send)) {
                 nbuffered = nbuffered    1;          /* handle piggybacked ack */
                 stop_timer(ack_expected % NR_BUFS);     /* frame arrived intact */
                 inc(ack_expected);                  /* advance lower edge of sender's window */
            }
            break;
       case cksum_err:
            if (no_nak) send_frame(nak, 0, frame_expected, out_buf);/* damaged frame */
            break;
       case timeout:
            send_frame(data, oldest_frame, frame_expected, out_buf);/* we timed out */
            break;
       case ack_timeout:
            send_frame(ack,0,frame_expected, out_buf);      /* ack timer expired; send ack */
     }
     if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
  }
}
```

SLIDING WINDOW PROTOCOLS

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat

Issues

- 7 frames 0-6 are sent,
- Received, ack-ed, and window of the receiver advanced to 7,0-5
- All ack lost
- Sender times out and resends frame 0-6
- At receiver, 0-5 accepted, buffered(since next frame it is expecting is 7) as new frames and ack-ed (Ack 6) and 6 discarded and acked as a duplicate
- Sender advances its window to 7,0-5
- Sends packets 7,0-5

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat

Issues

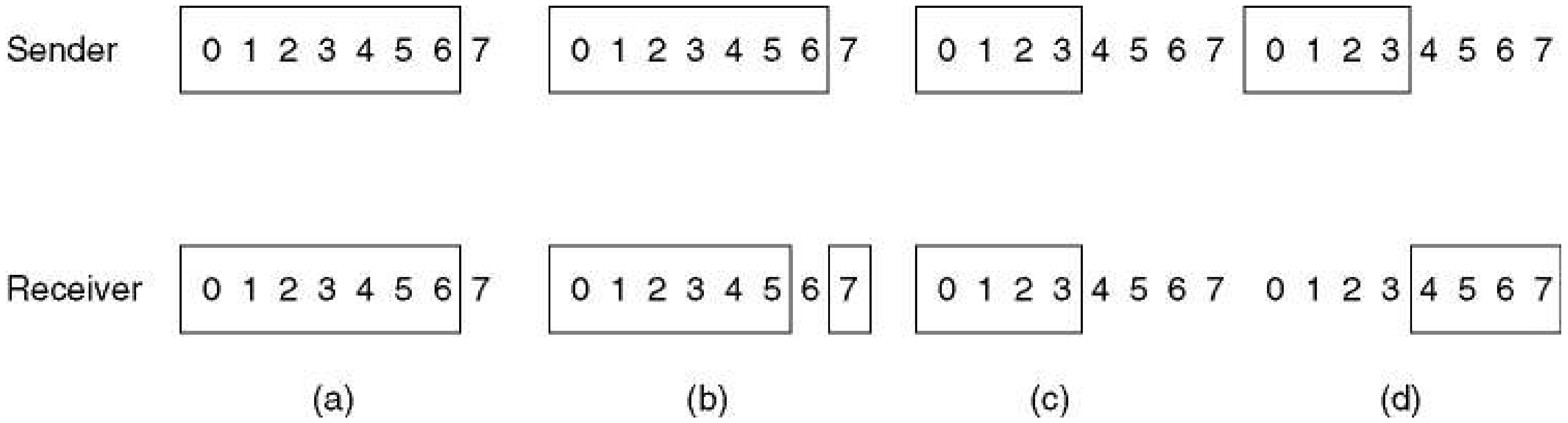
- 7 is accepted and passed onto the NL
- 0 and others are rejected as duplicate and the previous buffered frames are passed on to the NL - wrong packets!!!!
- Order in which packets are sent:
- P0…p6, p0…p6, p7, p0(new)…p5(new)
- Order in which packets are handed over to NL of the receiver:
- P0…p6,p7,p0…p5(duplicate) and rest are discarded including the new frames

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat

Solution

- Keep the window size 4
- That is half the range of permitted sequence numbers.

PROTOCOL 6: A Sliding Window Protocol Using Selective Repeat



(a) Initial situation with a window size seven.
(b) After seven frames sent and received, but not acknowledged.
(c) Initial situation with a window size of four.
(d) After four frames sent and received, but not acknowledged.

EXAMPLE DATA LINK PROTOCOLS

- Many networks use one of the bit-oriented protocols – SDLC (Synchronous Data Link Control), HDLC (High-level Data Link Control), ADCCP (Advanced Data Communication Control Procedure), or LAPB (Link Access Procedure) – at data link level.
- All of these protocols use flag bytes to delimit frames, and bit stuffing to prevent flag bytes from occurring in the data.
- HDLC – High-Level Data Link Control
  - ➢ This is a classical bit-oriented protocol whose variants have been in use for decades in many applications.
- The Data Link Layer in the Internet
  - ➢ PPP is the data link protocol used to connect home computers to the Internet.

EXAMPLE DATA LINK PROTOCOLS

High-Level Data Link Control

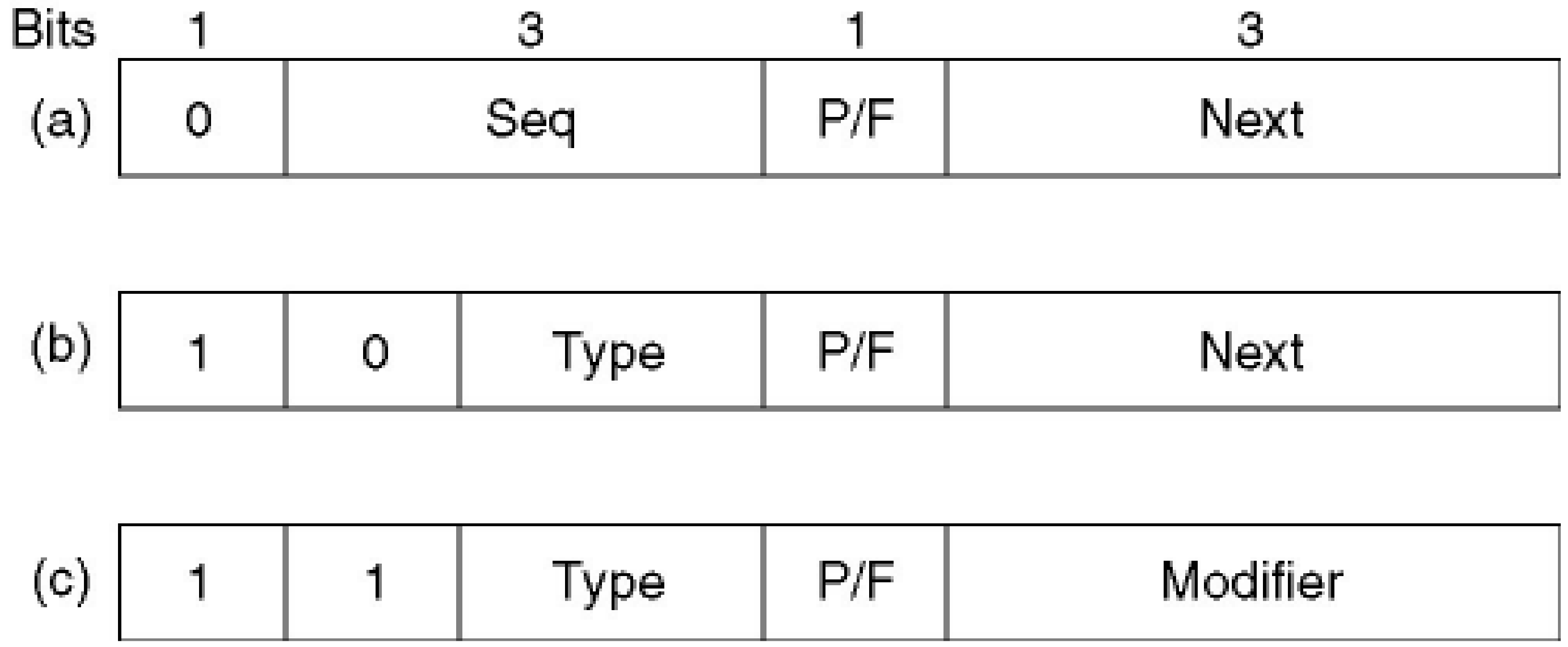
Bits	8	8	8	⩾ 0	16	8
	0 1 1 1 1 1 1 0	Address	Control	Data	Checksum	0 1 1 1 1 1 1 0

Frame format for bit-oriented protocols.

- Uses bit stuffing for data transparency.
- Address field is to identify one of the terminals;
- Control field is used for sequence numbers, acknowledgements, and other purpose.
- Data field may contain any information.
- Checksum field is a cyclic redundancy code.

EXAMPLE DATA LINK PROTOCOLS

High-Level Data Link Control



Control field of
(a) An information frame.
(b) A supervisory frame.
(c) An unnumbered frame.

- Seq is the frame sequence number.
- Next is piggybacked acknowledgement.
- Type is for distinguishing various kinds of Supervisory frames.
- P/F is Poll/Final.

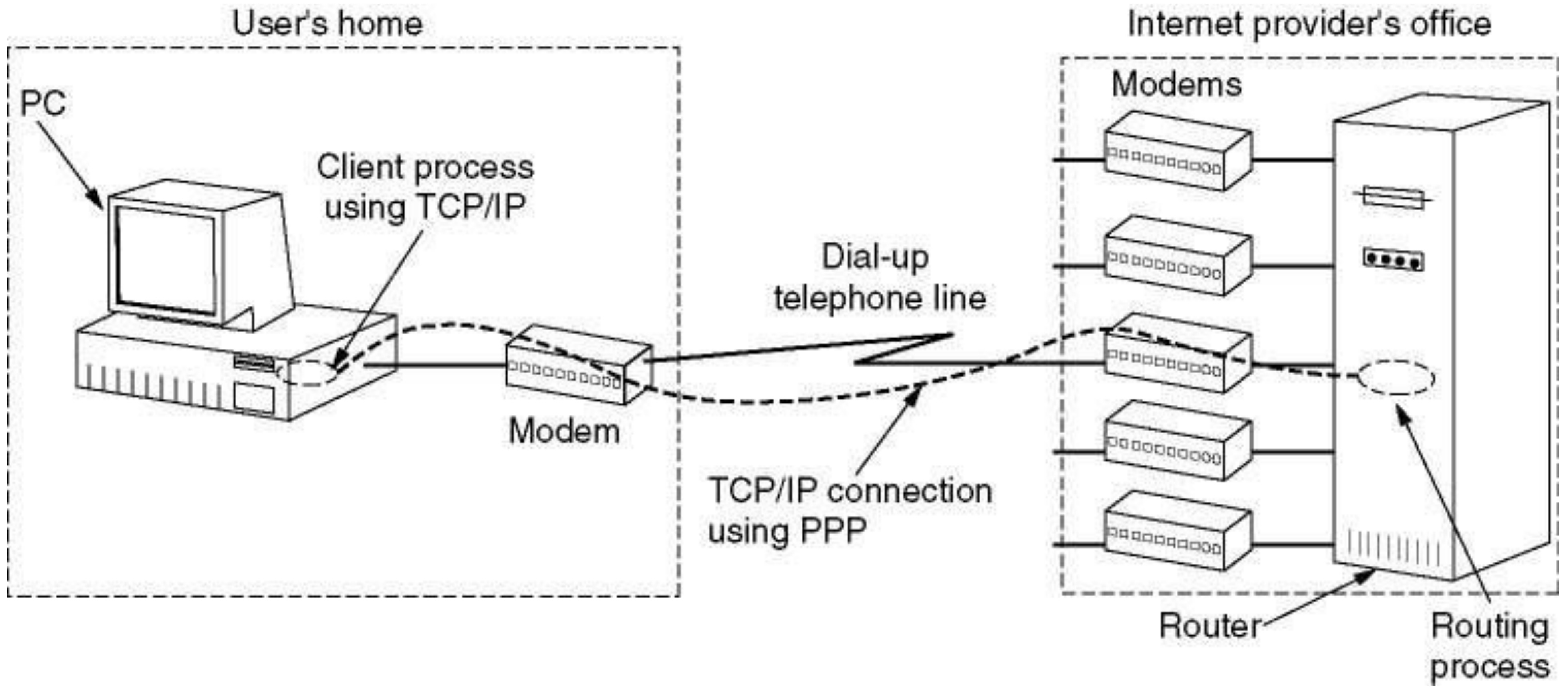
EXAMPLE DATA LINK PROTOCOLS

The Data Link Layer in the Internet

- Internet consists of individual machines (hosts and routers) and the communication infrastructure that connects them.
- LANs are widely used for interconnection, but most of the wide area infrastructure is built up from point-to-point leased lines.
- In this section we will examine the data link protocols used on point-to-point lines in the Internet.
- In practice, point-to-point communication is primarily used in two situations:
- All routers in subnet are communicated by point-to-point leased lines (or router-router leased line connection).
- Many users have home connections to the Internet using modems and dial-up telephone lines (dial-up host-router connection).
- For both router-router leased line connection and dial-up host-router connection, some point-to-point data link protocol is required on line for framing, error control, etc.
- The one used in Internet is called PPP.

EXAMPLE DATA LINK PROTOCOLS

The Data Link Layer in the Internet


User's home
PC
Client process using TCP/IP
Modem
Dial-up telephone line
TCP/IP connection using PPP
Internet provider's office
Modems
Router
Routing process

A home personal computer acting as an internet host.

EXAMPLE DATA LINK PROTOCOLS

The Data Link Layer in the Internet

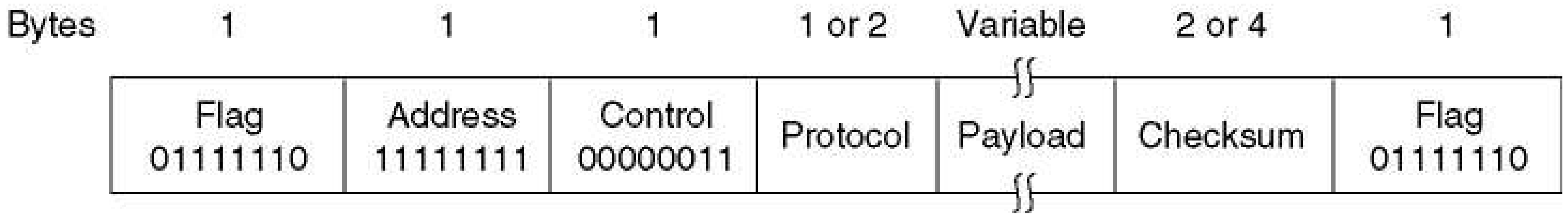
Point to Point Protocol

- PPP handles error detection, supports multiple protocols, allows IP addresses to be negotiated at connection time, permits authentication, and has many other features.
- PPP provides three features:
  1) 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.
  2) LCP – Link Control Protocol for bringing lines up, testing them, negotiating options, and bringing them down again gracefully when they are no longer needed.
  3) NCP – Network Control Protocol . A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used.

EXAMPLE DATA LINK PROTOCOLS

The Data Link Layer in the Internet

Point to Point Protocol



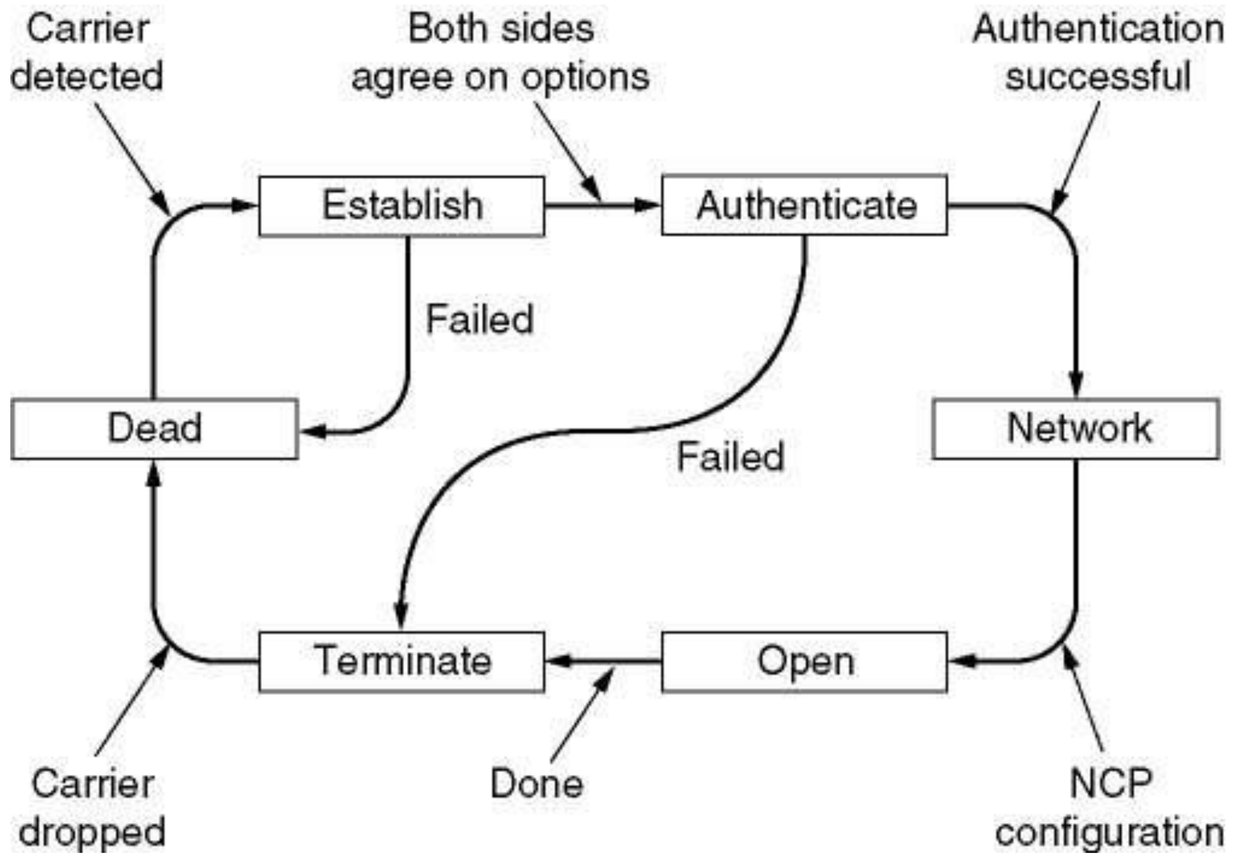
The PPP full frame format for unnumbered mode operation.

- Address is always set to the binary value 1111111 to indicate that all stations are to accept the frame.
- Control is always set to the 00000011 which indicates an unnumbered frame.
- Protocol tells what kind of packet is in the Payload field.

EXAMPLE DATA LINK PROTOCOLS

The Data Link Layer in the Internet

Point to Point Protocol



A simplified phase diagram for bring a line up and down.

EXAMPLE DATA LINK PROTOCOLS

The Data Link Layer in the Internet

Point to Point Protocol

Name	Direction	Description
Configure-request	I $\rightarrow$ R	List of proposed options and values
Configure-ack	I $\leftarrow$ R	All options are accepted
Configure-nak	I $\leftarrow$ R	Some options are not accepted
Configure-reject	I $\leftarrow$ R	Some options are not negotiable
Terminate-request	I $\rightarrow$ R	Request to shut the line down
Terminate-ack	I $\leftarrow$ R	OK, line shut down
Code-reject	I $\leftarrow$ R	Unknown request received
Protocol-reject	I $\leftarrow$ R	Unknown protocol requested
Echo-request	I $\rightarrow$ R	Please send this frame back
Echo-reply	I $\leftarrow$ R	Here is the frame back
Discard-request	I $\rightarrow$ R	Just discard this frame (for testing)

The LCP (Link Control Protocol) frame types.

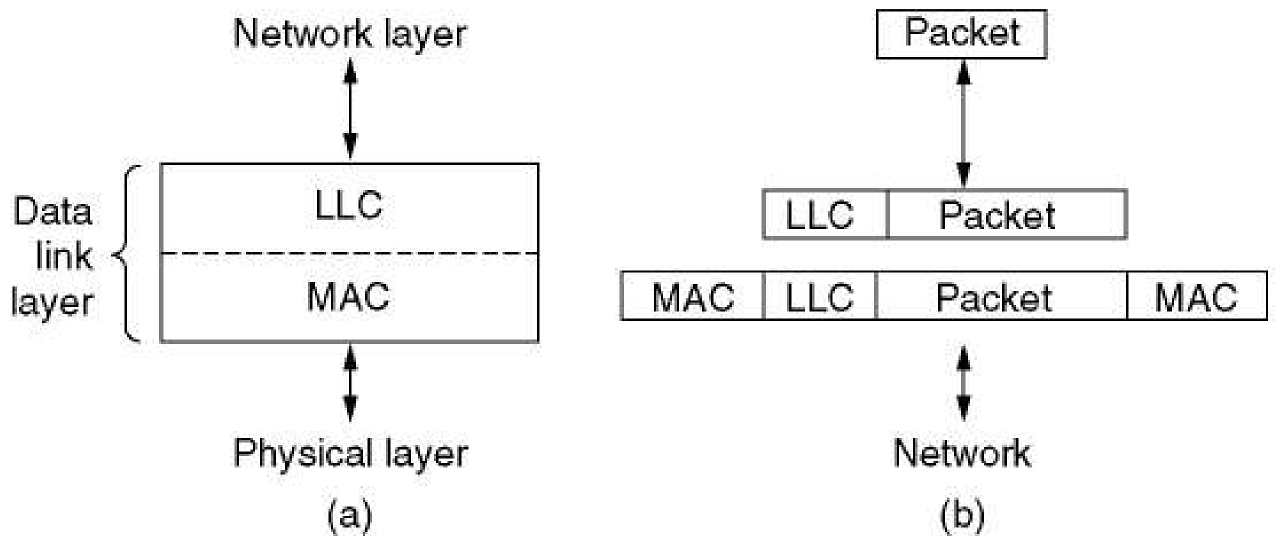
I –INITIATOR **R - RESPONDER**

MEDIUM ACCESS SUB LAYER

- Networks can be divided into two categories:
  1) Those using point-to-point connections and
  2) Those using broadcast channels.
- In any broadcast network, the key issue is how to determine who gets to use the channel when there is competition for it.
- To make this point clearer, consider a conference call in which six people, on six different telephones, are all connected together so that each one can hear and talk to all the others.
- It is very likely that when one of them stops speaking, two or more will start talking at once, leading to chaos.
- In a face-to-face meeting, chaos is avoided by external means, for example, at a meeting, people raise their hands to request permission to speak.

MEDIUM ACCESS SUB LAYER

- When only a single channel is available, determining who should go next is much harder.
- Many protocols for solving the problem are known and form the contents of this chapter.
- In the literature, broadcast channels are sometimes referred to as multi-access channels or random access channels.
- The protocols used to determine who goes next on a multi-access channel belong to a sublayer of the data link layer called the MAC (Medium Access Control) sublayer.
- Technically, the MAC sublayer is the bottom part of the data link layer.



MEDIUM ACCESS SUB LAYER

- The MAC sublayer is especially important in LANs, nearly all of which use a multiaccess channel as the basis of their communication.
- WANs, in contrast, use point-to-point links, except for satellite networks.
- Because multi-access channels and LANs are so closely related, in this chapter we will discuss LANs in general, as well as satellite and some other broadcast networks

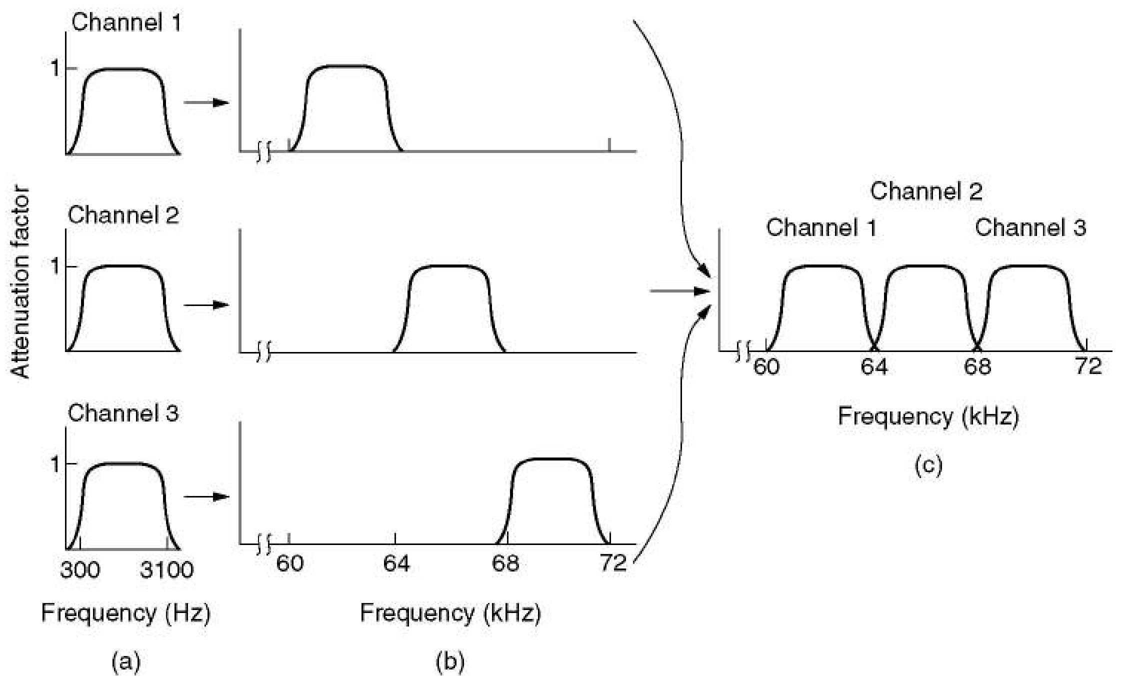
THE CHANNEL ALLOCATION PROBLEM

- The central theme of this chapter is how to allocate a single broadcast channel among competing users.
- We will first look at static and dynamic schemes in general. Then we will examine a number of specific algorithms.

THE CHANNEL ALLOCATION PROBLEM

Static Channel Allocation in LANs and MANs

- The traditional way of allocation a single channel, such as a telephone trunk, among multiple competing users is Frequency Division Multiplexing (FDM).
- If there are N users, the bandwidth is divided into N equal sized portions, each user being assigned one portion.



(a) The original bandwidths.
(b) The bandwidths raised in frequency.
(b) The multiplexed channel.

THE CHANNEL ALLOCATION PROBLEM

Static Channel Allocation in LANs and MANs

- Since each user has a private frequency band, there is no interference between users
- When there is only a small and fixed number of users, each of which has a heavy (buffered) load of traffic (e.g., carriers' switching offices), FDM is a simple and efficient allocation mechanism.
- However, when the number of senders is large and continuously varying, or the traffic is bursty, FDM presents some problems.
- If the spectrum is cut up into N regions, and fewer than N users are currently interested in communicating, a large piece of valuable spectrum will be wasted.

THE CHANNEL ALLOCATION PROBLEM

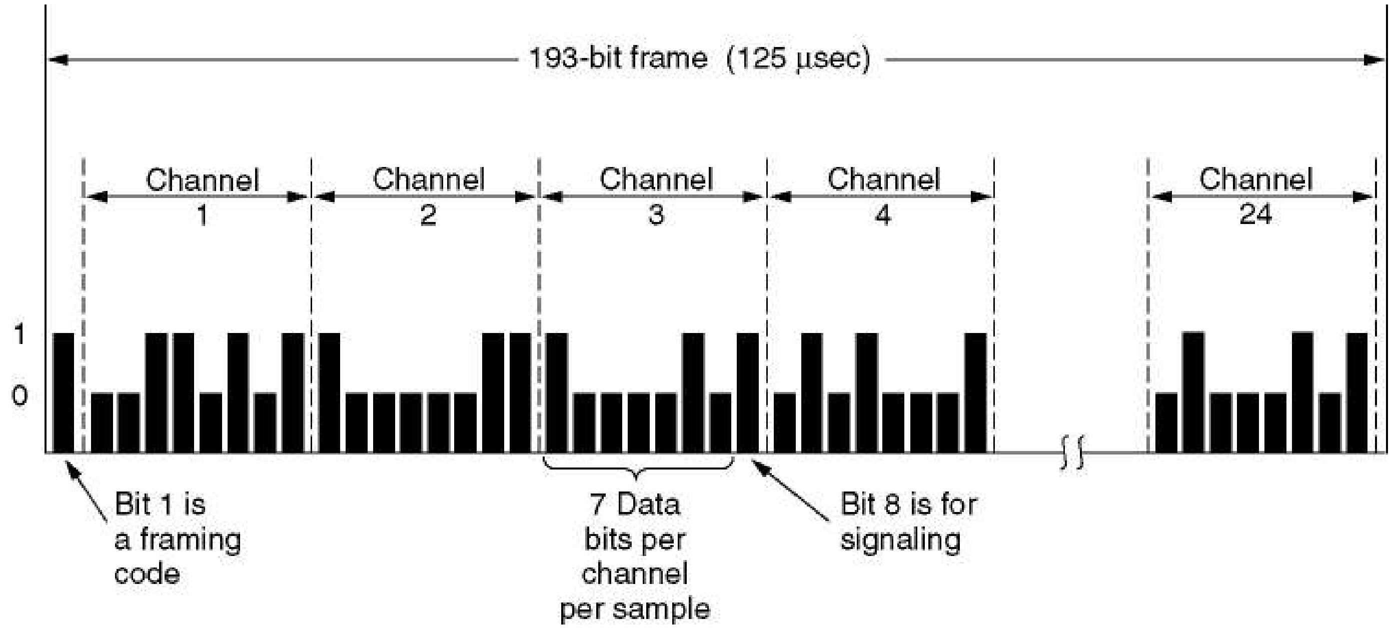
Static Channel Allocation in LANs and MANs

- If more than N users want to communicate, some of them will be denied permission, for lack of bandwidth, even if some of the users who have been assigned a frequency band hardly ever transmit or receive anything.
- However, even assuming that the number of users could somehow be held constant at N, dividing the single available channel into static subchannels is inherently inefficient.
- The basic problem is that when some users are quiescent, their bandwidth is simply lost. They are not using it, and no one else is allowed to use it either.
- Furthermore, in most computer systems, data traffic is extremely bursty (peak traffic to mean traffic ratios of 1000:1 are common).
- Consequently, most of the channels will be idle most of the time.

THE CHANNEL ALLOCATION PROBLEM

Static Channel Allocation in LANs and MANs

- Precisely the same arguments that apply to FDM also apply to time division multiplexing (TDM).
- Each user is statically allocated every Nth time slot.
- If a user does not use the allocated slot, it just lies fallow.



Time Division Multiplexing
The T1 carrier (1.544 Mbps).

THE CHANNEL ALLOCATION PROBLEM

Static Channel Allocation in LANs and MANs

- Since none of the traditional static channel allocation methods work well with bursty traffic, we will now explore dynamic methods.

THE CHANNEL ALLOCATION PROBLEM

Dynamic Channel Allocation in LANs and MANs

- Before we get into the first of the many channel allocation methods to be discussed in this chapter, it is worthwhile carefully formulating the allocation problem.
- Underlying all the work done in this area are five key assumptions, described below.

1. **Station Model.**
   - ➢ The model consists of N independent stations (e.g., computers, telephones, or personnel communicators), each with a program or user that generates frames for transmissions. Stations are sometimes called terminals.
   - ➢ Once a frame has been generated, the station is blocked and does nothing until the frame has been successfully transmitted.

THE CHANNEL ALLOCATION PROBLEM

Dynamic Channel Allocation in LANs and MANs

2. **Single Channel Assumption.**
   - ➢ A single channel is available for all communication. All stations can transmit on it and all can receive from it.
3. **Collision Assumption.**
   - ➢ If two frames are transmitted simultaneously, they overlap in time and the resulting signal is garbled. This event is called a collision. All stations can detect collisions.

THE CHANNEL ALLOCATION PROBLEM

Dynamic Channel Allocation in LANs and MANs

4.(a) Continuous Time.

- ➢ Frame transmission can begin at any instant. There is no master clock dividing time into discrete intervals.

4.(b) Slotted Time.

- ➢ Time is divided into discrete intervals (slots). Frame transmissions always begin at the start of a slot. A slot may contain 0, 1, or more frames, corresponding to an idle slot, a successful transmission, or a collision, respectively.

THE CHANNEL ALLOCATION PROBLEM

Dynamic Channel Allocation in LANs and MANs

5.(a) Carrier Sense.

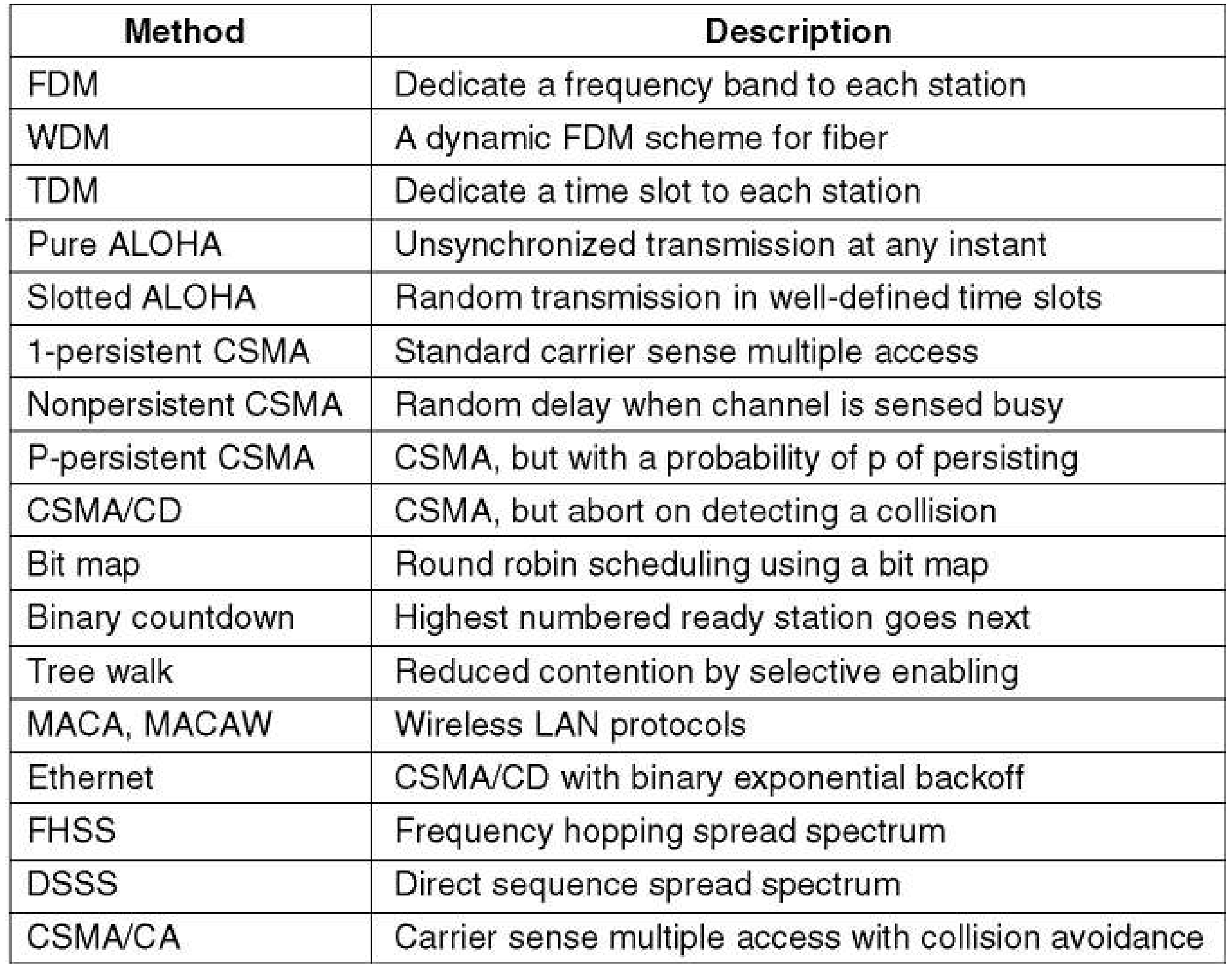
- Stations can tell if the channel is in use before trying to use it. If the channel is sensed as busy, no station will attempt to use it until it goes idle.

5.(b) No Carrier Sense.

- Stations cannot sense the channel before trying to use it. They go ahead and transmit. Only later can determine whether the transmission was successful.

MULTIPLE ACCESS PROTOCOLS

- Many Algorithms for Allocating a Multiple Access Channel are Known.



Method	Description
FDM	Dedicate a frequency band to each station
WDM	A dynamic FDM scheme for fiber
TDM	Dedicate a time slot to each station
Pure ALOHA	Unsynchronized transmission at any instant
Slotted ALOHA	Random transmission in well-defined time slots
1-persistent CSMA	Standard carrier sense multiple access
Nonpersistent CSMA	Random delay when channel is sensed busy
P-persistent CSMA	CSMA, but with a probability of p of persisting
CSMA/CD	CSMA, but abort on detecting a collision
Bit map	Round robin scheduling using a bit map
Binary countdown	Highest numbered ready station goes next
Tree walk	Reduced contention by selective enabling
MACA, MACAW	Wireless LAN protocols
Ethernet	CSMA/CD with binary exponential backoff
FHSS	Frequency hopping spread spectrum
DSSS	Direct sequence spread spectrum
CSMA/CA	Carrier sense multiple access with collision avoidance

Channel allocation methods and systems for a common channel.

THE CHANNEL ALLOCATION PROBLEM

ALOHA

- Although Abramson’s work, called ALOHA system, used ground-based radio broadcasting, the basic idea is applicable to any system in which uncoordinated users are competing for the use of a single shared channel.
- We will discuss two versions of ALOHA here: pure (1970) and slotted (1972). They differ with respect to whether time is divided into discrete slots into which all frames must fit. Pure ALOHA does not require global time synchronization; slotted ALOHA does.

THE CHANNEL ALLOCATION PROBLEM

ALOHA

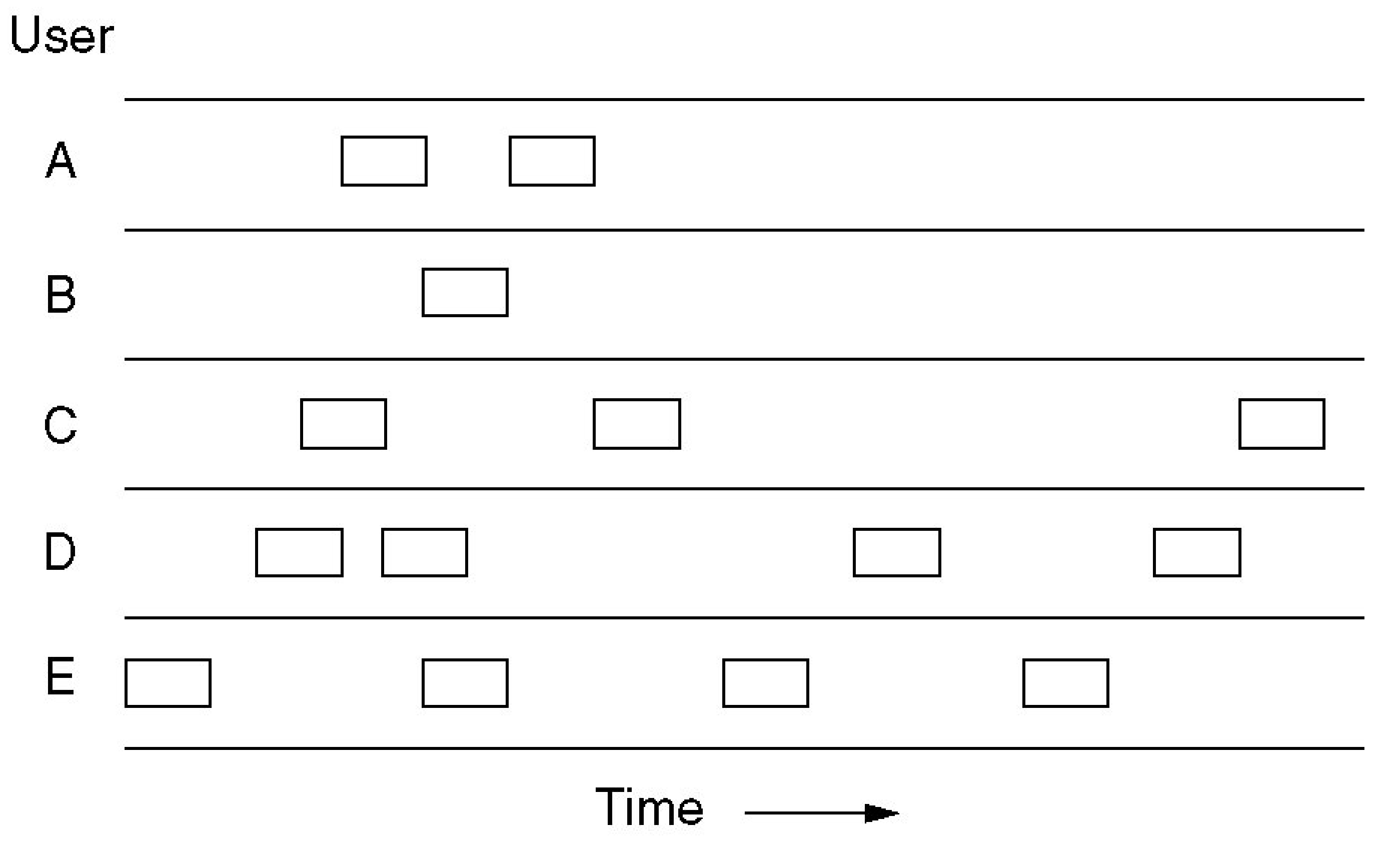
Pure ALOHA

- The basic idea of an ALOHA system is simple: let users transmit whenever they have data to be sent.
- There will be collisions, of course, and the colliding frames will be damaged.
- However, due to the feedback property of broadcasting, a sender can always find out whether its frame was destroyed by listening to the channel, the same way other users do.
- If listening while transmitting is not possible for some reason, acknowledgements are needed.
- If the frame was destroyed, the sender just waits a random amount of time and sends it again.
- Systems in which multiple users share a common channel in a way that can lead to conflicts are widely known as contention systems.

THE CHANNEL ALLOCATION PROBLEM

ALOHA

Pure ALOHA

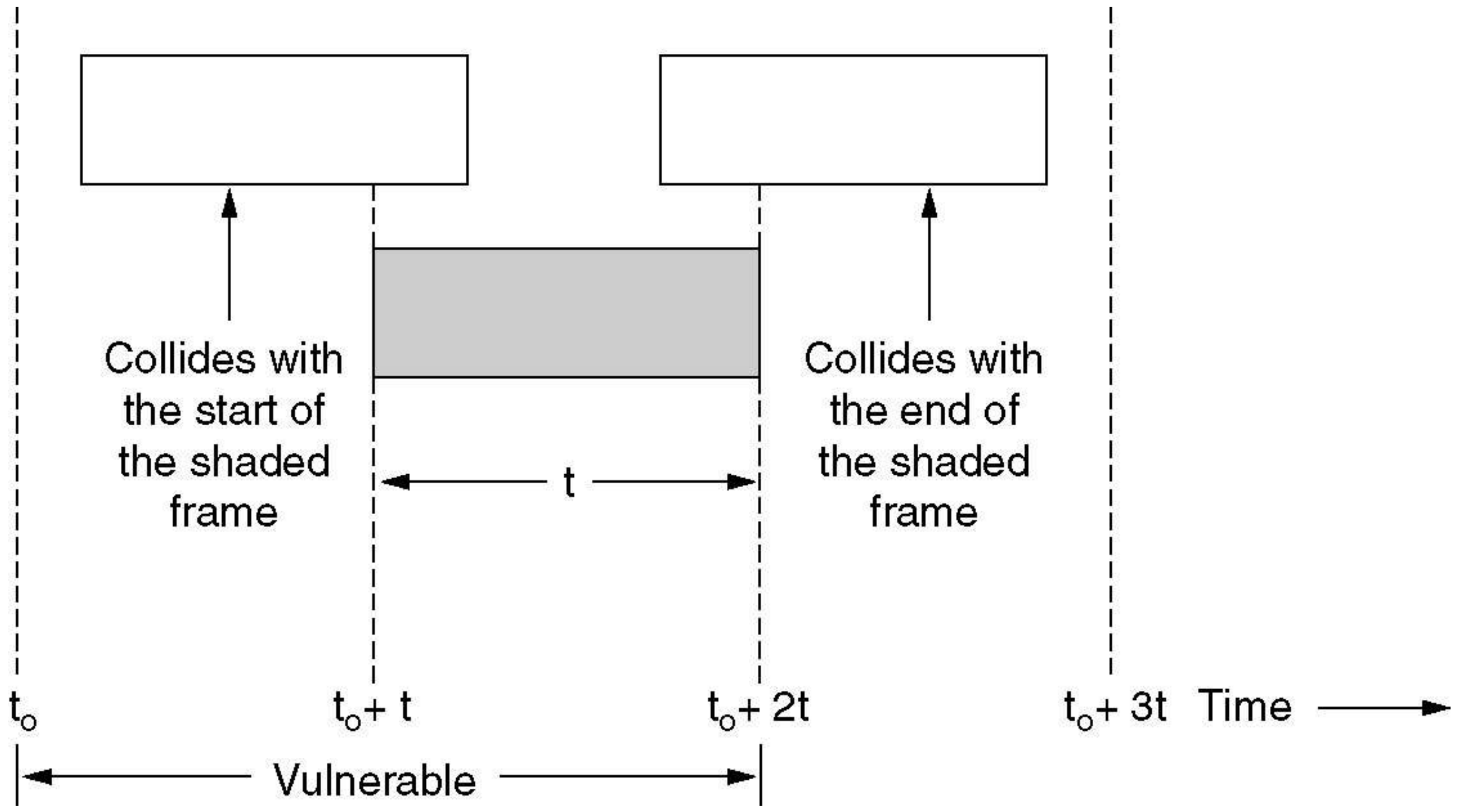


In pure ALOHA, frames are transmitted at completely arbitrary times.

THE CHANNEL ALLOCATION PROBLEM

ALOHA

Pure ALOHA


Collides with the start of the shaded frame
Collides with the end of the shaded frame
t
t_o
t_o+ t
t_o+ 2t
t_o+ 3t
Time
Vulnerable

Vulnerable period for the shaded frame.

THE CHANNEL ALLOCATION PROBLEM

ALOHA

Pure ALOHA

- With pure ALOHA the best channel utilization that can be achieved is

$$S=Ge^{-2G}$$

- for pure ALOHA

$$G=0.5$$

- and

$$S=1/2e \text{ or } 18 \text{ percent}$$

THE CHANNEL ALLOCATION PROBLEM

ALOHA

Slotted ALOHA

- In 1972, Roberts published a method for doubling the capacity of an ALOHA system. His proposal was to divide time into discrete intervals, each interval corresponding to one frame.
- This approach requires the users to agree on slot boundaries. One way to achieve synchronization would be to have one special station emit a pip at the start of each interval, like a clock.
- In Roberts' method, which has come to be known as slotted ALOHA, in contrast to Abramson's pure ALOHA, a computer is not permitted to send whenever a carriage return is typed. Instead, it is required to wait for the beginning of the next slot
- Thus, the continuous pure ALOHA is turned into a discrete one.

ALOHA

Slotted ALOHA

- With Slotted ALOHA the best channel utilization that can be achieved is

$$S=Ge^{-G}$$

- for slotted ALOHA

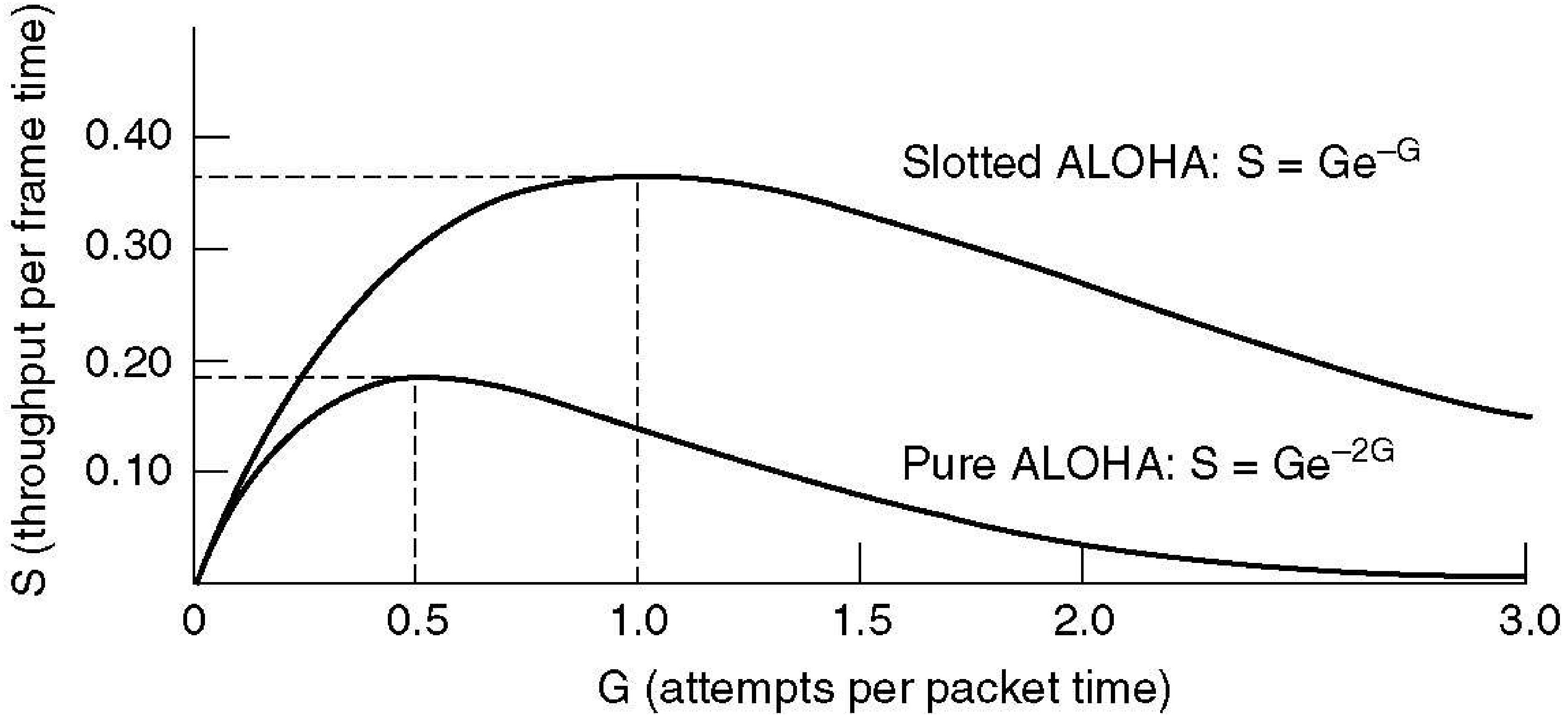
$$G=1$$

- and

$$S=1/e \text{ or } 36 \text{ percent}$$

ALOHA

Pure and Slotted ALOHA



Throughput versus offered traffic for ALOHA systems.

THE CHANNEL ALLOCATION PROBLEM

Carrier Sense Multiple Access Protocols

- With slotted ALOHA the best channel utilization that can be achieved is 1/e. This is hardly surprising, since with stations transmitting at will, without paying attention to what the other station are doing, there are bound to be many collisions.
- In local area networks, however, it is possible for station to detect what other station are doing, and adapt their behavior accordingly. This networks can achieve a much better utilization than 1/e.
- In this section we will discuss some protocols for improving performance.
- Protocols in which stations listen for a carrier (i.e., a transmission) and act accordingly are called carrier sense protocols.

THE CHANNEL ALLOCATION PROBLEM

Carrier Sense Multiple Access Protocols

- A number of them have been proposed.
  1) 1-persistent CSMA (Carrier Sense Multiple Access)
  2) nonpersistent CSMA
  3) p-persistent CSMA
  4) CSMA with Collision Detection (CSMA/CD)

Carrier Sense Multiple Access Protocols

1-persistent CSMA

- When a station has data to send, it first listens to the channel to see if anyone else is transmitting at that moment.
- If the channel is busy, the station waits until it becomes idle. When the station detects an idle channel, it transmits a frame.
- If a collision occurs, the station waits a random amount of time and starts all over again.
- The protocol is called 1-persistent because the station transmits with a probability of 1 when it finds the channel idle.

THE CHANNEL ALLOCATION PROBLEM

Carrier Sense Multiple Access Protocols

Nonpersistent CSMA

- Before sending, a station senses the channel. If no one else is sending, the station begins doing so itself.
- However, if the channel is already in use, the station does not continually sense it for the purpose of seizing it immediately upon detecting the end of the previous transmission.
- Instead, it waits a random period of time and then repeats the algorithm.
- Consequently, this algorithm leads to better channel utilization but longer delays than 1-persistent CSMA.

Carrier Sense Multiple Access Protocols

p-persistent CSMA

- It applies to slotted channel and works as follows.
- When a station becomes ready to send, it sense the channel. If it is idle, it transmits with a probability p.
- With a probability q=1-p, it defers until the next slot.
- If that slot is also idle, it either transmits or defers again, with probabilities p and q.
- This process is repeated until either the frame has been transmitted or another station has begun transmitting.

THE CHANNEL ALLOCATION PROBLEM

Carrier Sense Multiple Access Protocols

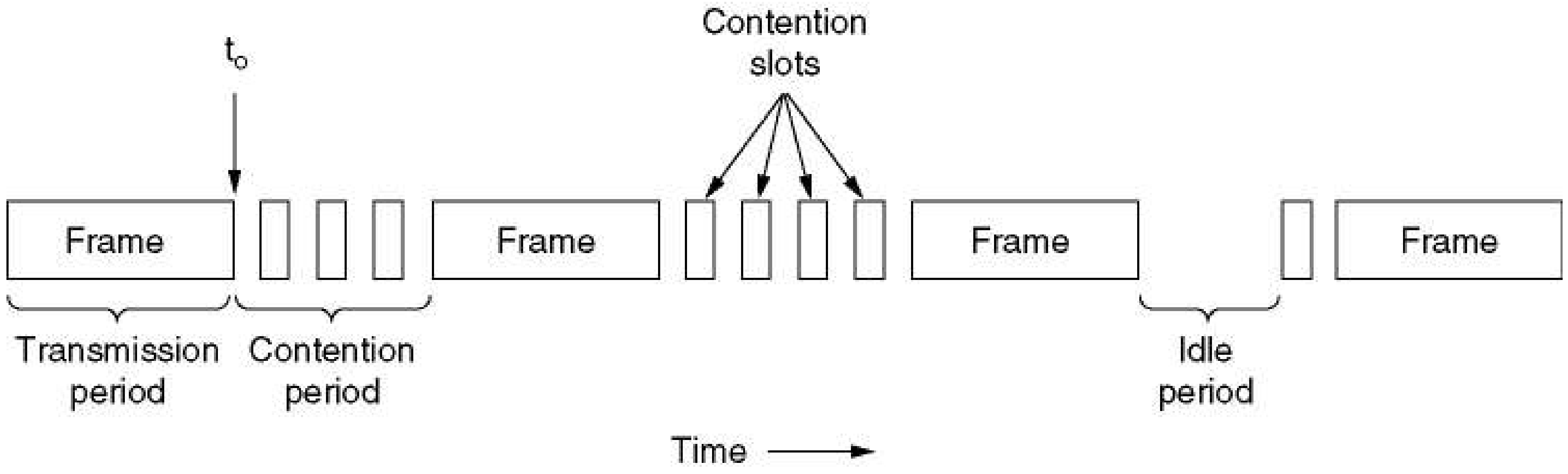
CSMA with Collision Detection

- Another improvement over ALOHA is for station to abort their transmissions as soon as they detect a collision.
- In other words, if two stations sense the channel to be idle and begin transmitting simultaneously, they will both detect the collision almost immediately.
- Rather than finish transmitting their frames, which are irretrievably garbled anyway, they should abruptly stop transmitting as soon as the collision is detected.
- Quickly terminating damaged frames saves time and bandwidth.
- This protocol is widely used on LANs in the MAC sublayer.

THE CHANNEL ALLOCATION PROBLEM

Carrier Sense Multiple Access Protocols

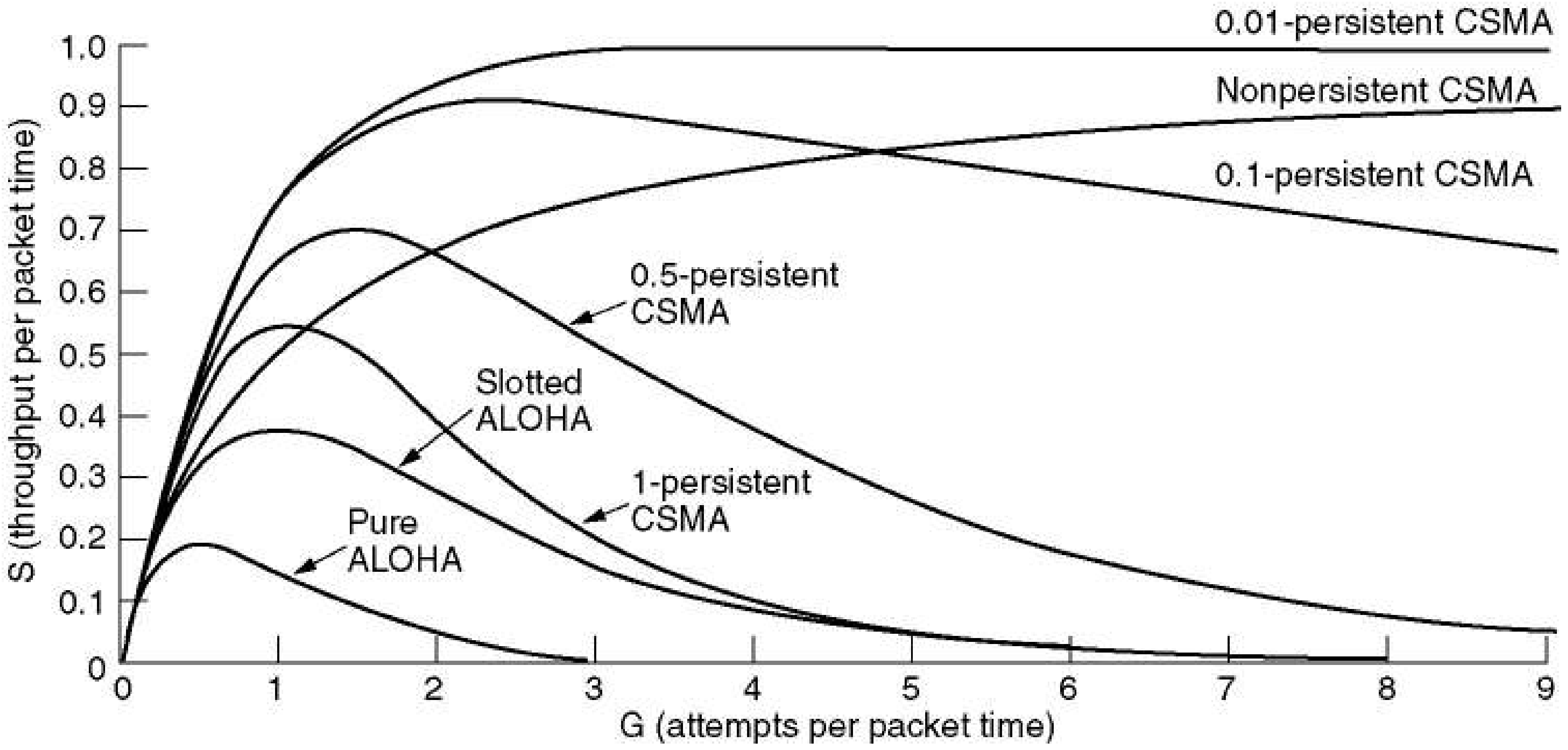
CSMA with Collision Detection



CSMA/CD can be in one of three states: contention, transmission, or idle.

Carrier Sense Multiple Access Protocols

CSMA Protocols Performance



Comparison of the channel utilization versus load for various random access protocols.

COLLISION-FREE PROTOCOLS

- Although collisions do not occur with CSMA/CD once a station has unambiguously captured the channel, they can still occur during the contention period.
- These collisions adversely affect the system performance, especially when the cable is long and the frames are short. And CSMA/CD is not universally applicable.
- In this section, we will examine two protocols that resolve the contention for channel without any collisions at all, not even during the contention period.

  1) a bit-map protocol

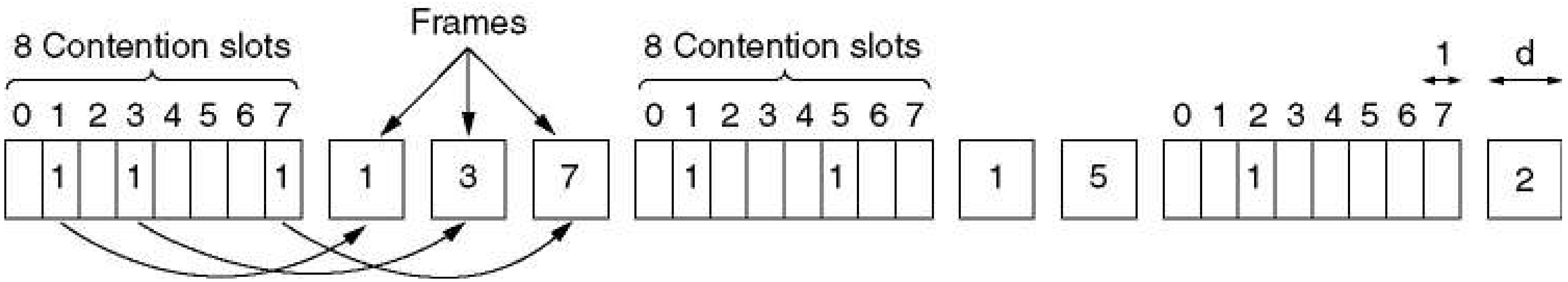
  2) binary countdown

COLLISION-FREE PROTOCOLS

- Assumptions for these two protocols:
  1) there are exactly n stations, each with a unique address from 0 to n-1 “wired” into it.
  2) propagation delay is negligible

Basic Bit-Map Protocol:

- In this protocol, each contention period consists of exactly n slots.
- If station 0 has a frame to send, it transmits a 1 bit during the zeroth slot.
- No other station is allowed to transmit during this slot.
- In general, station j may announce that it has a frame to send by inserting a 1 bit into slot j.



The basic bit-map protocol.

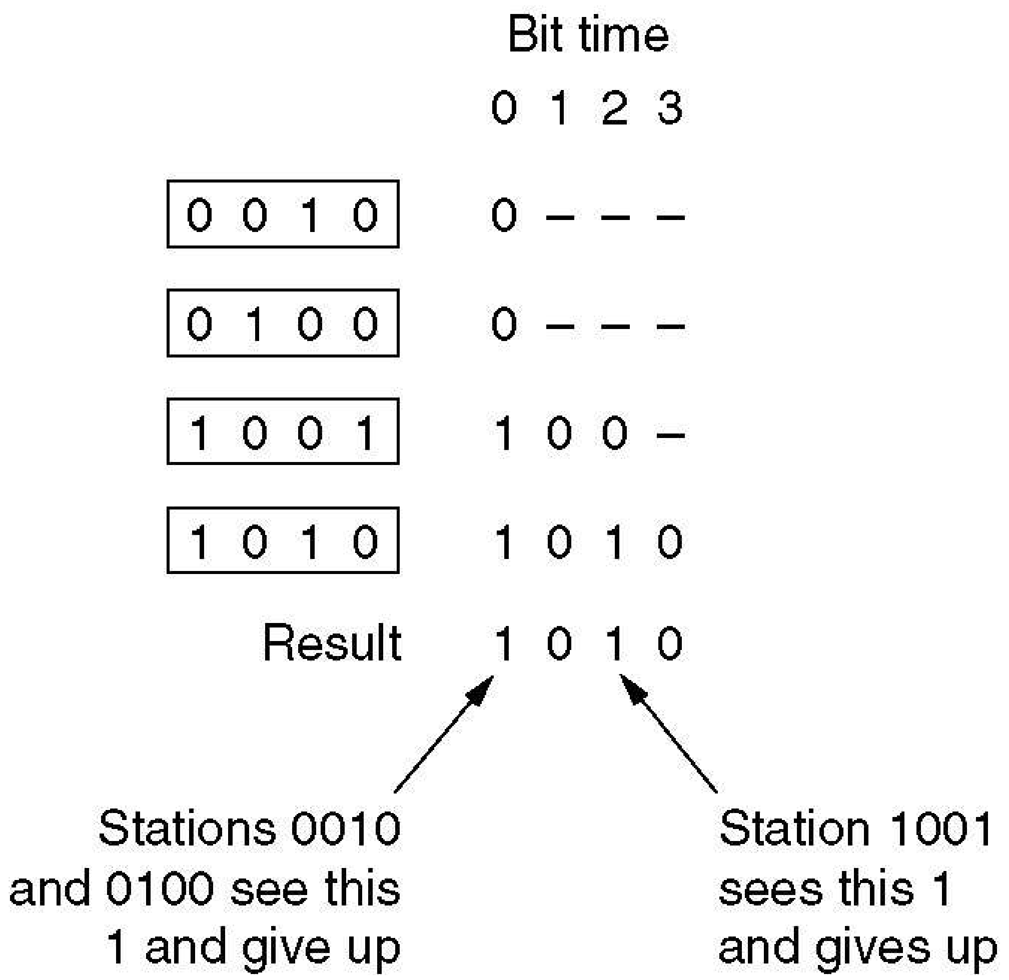
COLLISION-FREE PROTOCOLS

Binary Countdown:

- A problem with the basic bit-map protocol is that the overhead is 1 bit per station, so it does not scale well to networks with thousands of stations.
- We can do better than that by using binary station addresses.
- A station wanting to use the channel now broadcasts its address as a binary bit string, starting with the high-order bit.
- All addresses are assumed to be the same length.
- The bits in each address position from different stations are Boolean ORed together.

COLLISION-FREE PROTOCOLS

Binary Countdown:



The binary countdown protocol. A dash indicates silence.

ETHERNET

- Both the INTERNET and ATM were designed for Wide Area Networking.
- However, many companies universities, and other organizations have large numbers of computers that must be connected.
- This need gave rise to the Local Area Network.
- In this section we will say a little bit about the most popular LAN, ETHERNET

ETHERNET

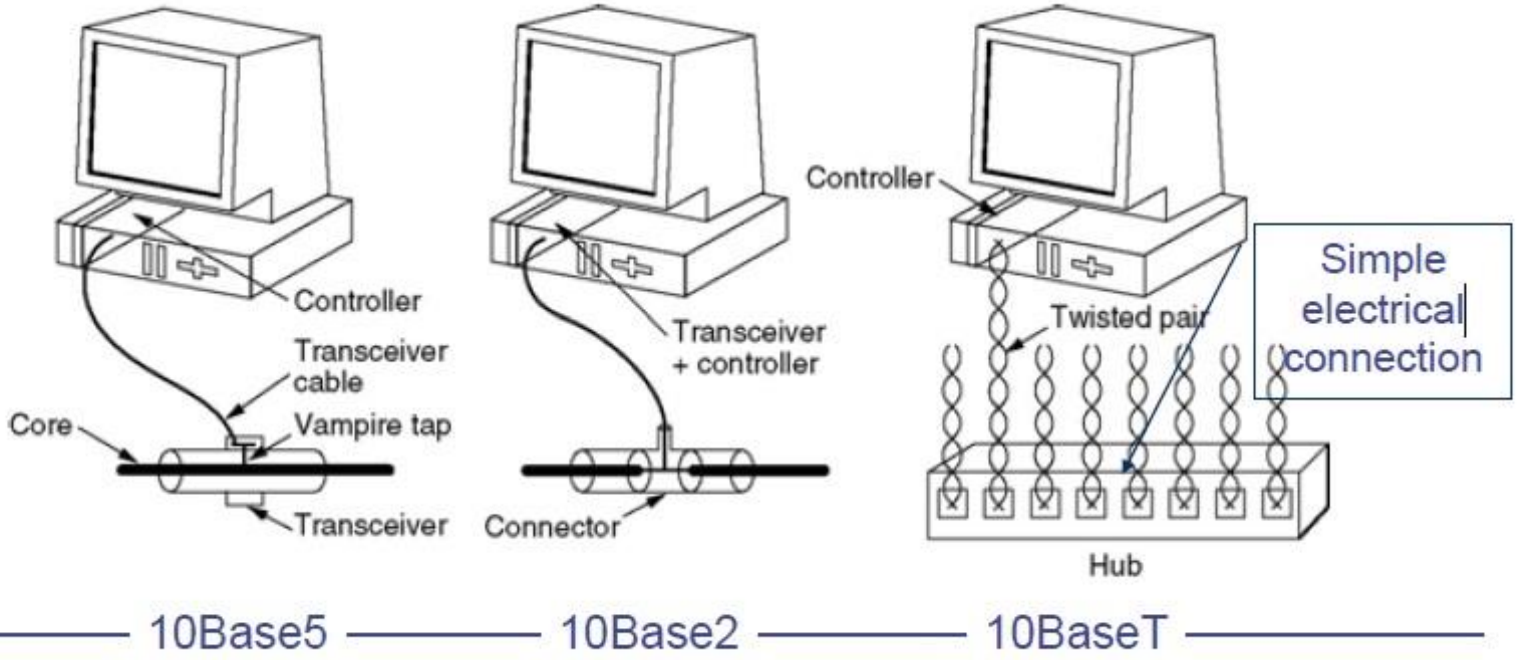
- A practical example, dealing (mostly) with MAC: Ethernet
  - ✓ Standardized by IEEE as standard 802.3
  - ✓ Part of the 802 family of standards dealing with MAC protocols
  - ✓ Also contains PHY and DLC specifications
- **Issues**
  - ✓ Cabling
  - ✓ Physical layer
  - ✓ MAC sublayer
  - ✓ Switched Ethernet
  - ✓ Fast & gigabit Ethernet

ETHERNET

Cabling

"Yellow cable"

Name	Cable	Max. seg.	Nodes/seg.	Advantages
10Base5	Thick coax	500 m	100	Original cable; now obsolete
10Base2	Thin coax	185 m	30	No hub needed
10Base-T	Twisted pair	100 m	1024	Cheapest system
10Base-F	Fiber optics	2000 m	1024	Best between buildings

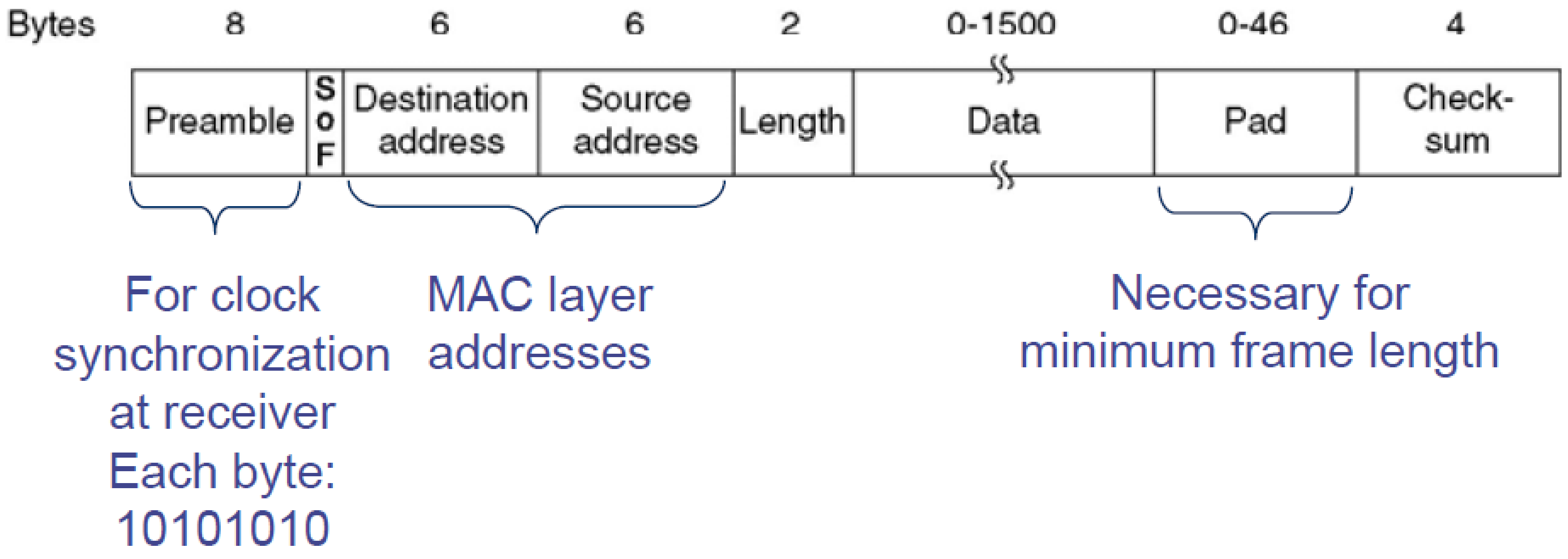


Ethernet Physical Layer

- Details depend on medium
- Common: Manchester encoding
  - ✓ At +/- 0.85 V (typically) to ensure DC freeness
- With option for signal violations
  - ✓ Used to demarcate frames

Ethernet MAC Sub Layer

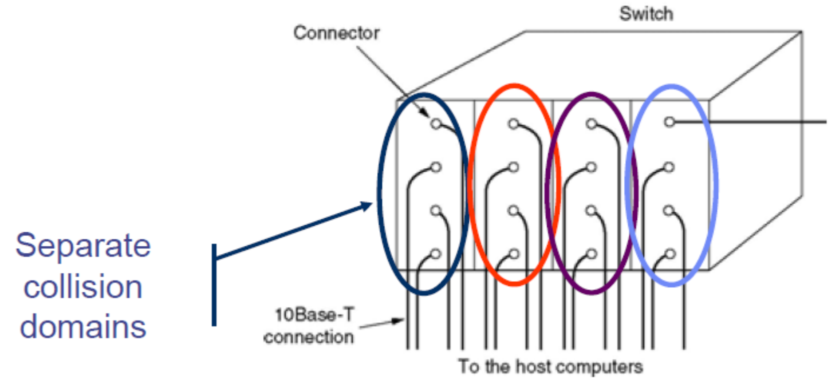
- Essentially: CSMA/CD with binary exponential backoff
- Frame format:



Switched Ethernet

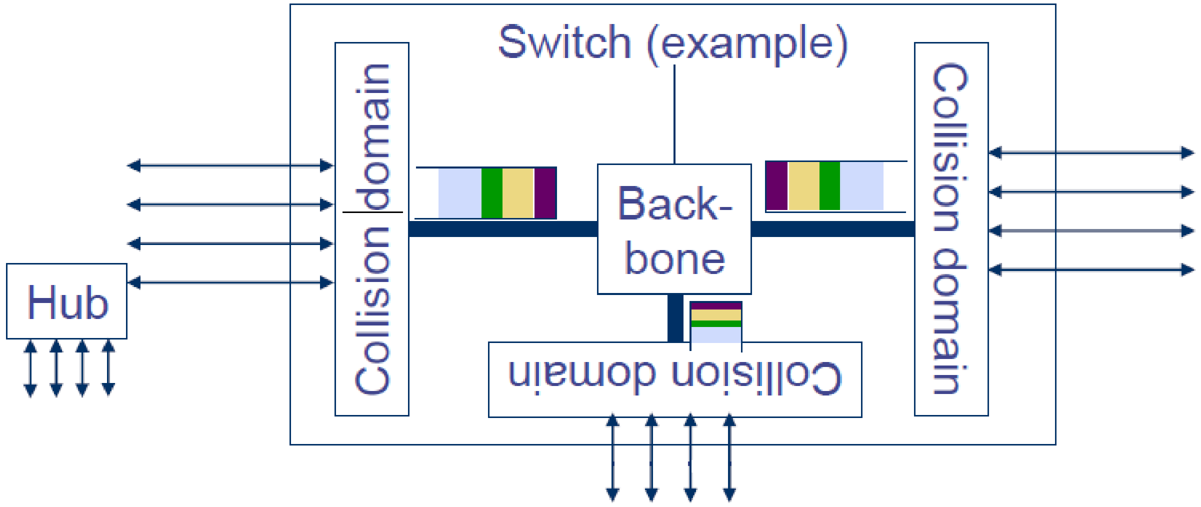
- With conventional 10Base5/10Base2 Ethernet, all stations attached to a single cable form a collision domain
  - ✓ Packets from all these stations might potentially collide
  - ✓ Big collision domains stress the CSMA/CD mechanism, reducing performance
- How to reduce collision domains but still maintain connectivity of local stations?
  - ✓ Use smaller collision domains!
  - ✓ To ensure connectivity, put a switch in

Recall: A *hub* is electrically connected, thus a single collision domain



An Ethernet Switch

- Unlike a hub, not a simple electrical connection for a starwired topology
- How to exchange packets between different collision domains?
  - ✓ Switch contains buffers to intermediately store incoming packets before forwarding them towards their destination
  - ✓ Different buffer structures possible: one per incoming link, one per group of links,…
  - ✓ Cost issue, mainly

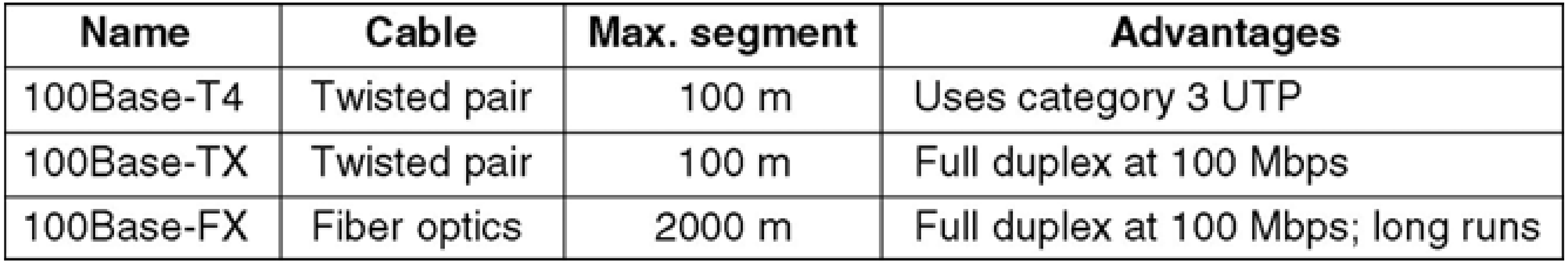


Fast Ethernet

- "Normal" (even switched) Ethernet "only" achieves 10 MBit/s
- 1992: Build a faster Ethernet!
  - ✓ Goals: Backward compatible, stick with the old protocol to avoid hidden traps, get job done quickly
  - ✓ Result: 802.3u – aka "Fast Ethernet"
- Fast Ethernet
  - ✓ Keep everything the same (frame format, protocol rules)
  - ✓ Reduce bit time from 100 ns to 10 ns
  - ✓ Consequences for maximum length of a wiring segment, minimum packet sizes? (Recall unavoidable collisions in CSMA!)

Fast Ethernet - Cabling

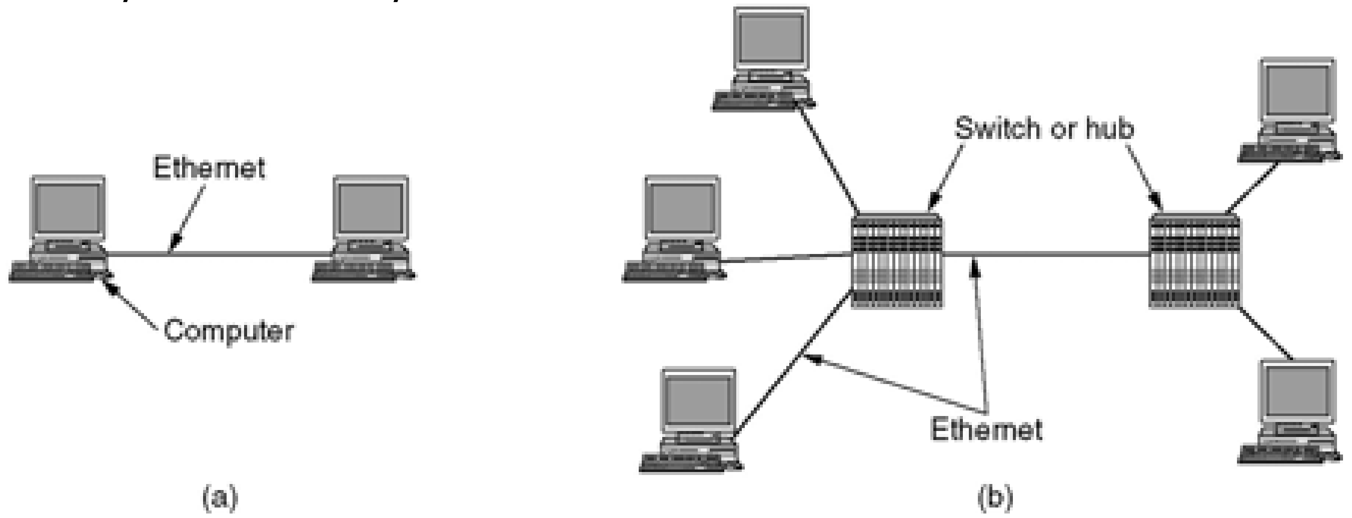
- Standard category 3 twisted pairs (telephony cables) cannot support 200 MBaud over 100 m cable length
  - ✓ Solution: use 2 pairs of wires in this case, reduce baud rate
- Also, Fast Ethernet/cat 5 cabling does not use Manchester, but 4B/5B



Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

Gigabit Ethernet

- Ok: can we go another factor of 10 faster?
  - ✓ 1995 – gigabit Ethernet
  - ✓ Goal: again, keep basic scheme as it is
- Works, but price to pay: No more multi-drop configurations as in classic Ethernet
  - ✓ In gigabit Ethernet, each wire has exactly two machines attached to it
    - ➢ Terminal and/or switch/hub



Gigabit Ethernet

- With a switch
  - ✓ No shared collision domains → no collision → no need for CSMA/CD
  - ✓ Allows full-duplex operation of each link
- With a hub
  - ✓ Collisions, half duplex, CSMA/CD
  - ✓ Maximum cable distance is reduced to 25 m
  - ✓ Actually: not very sensible combination from a cost/performance perspective

Cabling

Name	Cable	Max. segment	Advantages
1000Base-SX	Fiber optics	550 m	Multimode fiber (50, 62.5 microns)
1000Base-LX	Fiber optics	5000 m	Single (10 µ) or multimode (50, 62.5 µ)
1000Base-CX	2 Pairs of STP	25 m	Shielded twisted pair
1000Base-T	4 Pairs of UTP	100 m	Standard category 5 UTP

And how does traffic on an Ethernet look like?

- How many packets are there, per time unit, transmitted over a typical Ethernet?
- Assumptions:
  - ✓ Many sources connected to a single Ethernet
  - ✓ Sources independently generate traffic (=try to transmit a packet)
- Intuition:
  - ✓ Average number of transmitted packets might be bursty over short time windows
  - ✓ The longer the considered time window, the smoother the number of transmissions should become, right?