Chapter 3

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.

Data Link Layer (1/5)

This study deals the algorithms for achieving reliable, efficient communication between two adjacent machines at the data link layer

Data Link Layer (3/5)

- 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.

Data Link Layer (4/5)

- The protocols used for communication must take all these factors into consideration.
- These protocols are the subject of this chapter.

Data Link Layer (5/5)

Objectives:

1) An introduction to the key design issues present in the data link layer;
2) A study of protocols by looking at the nature of errors, their causes, and how they can be detected and corrected;
3) A study of increasingly complex protocols, each one solving more and more of the problems present in this layer;
4) An examination of protocol modeling and correctness.

Data Link Layer Design Issues (1/4)

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.

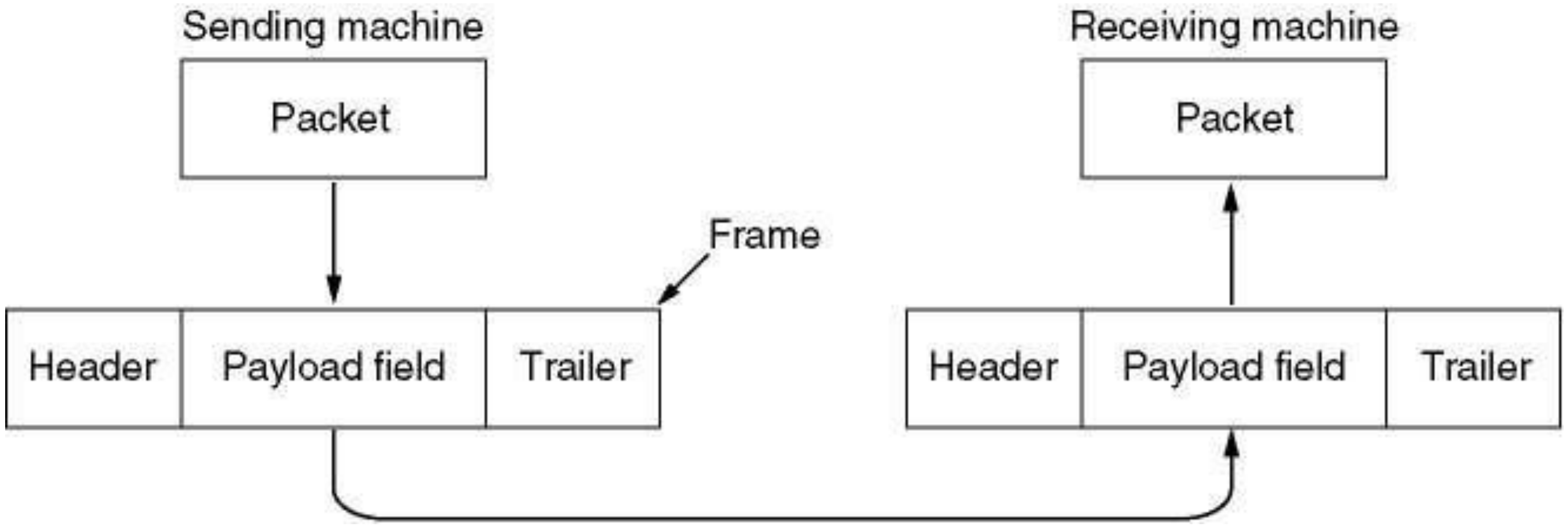
Data Link Layer Design Issues (2/4)

- 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 (3/4)



Relationship between packets and frames.

Data Link Layer Design Issues (4/4)

Subtopics:

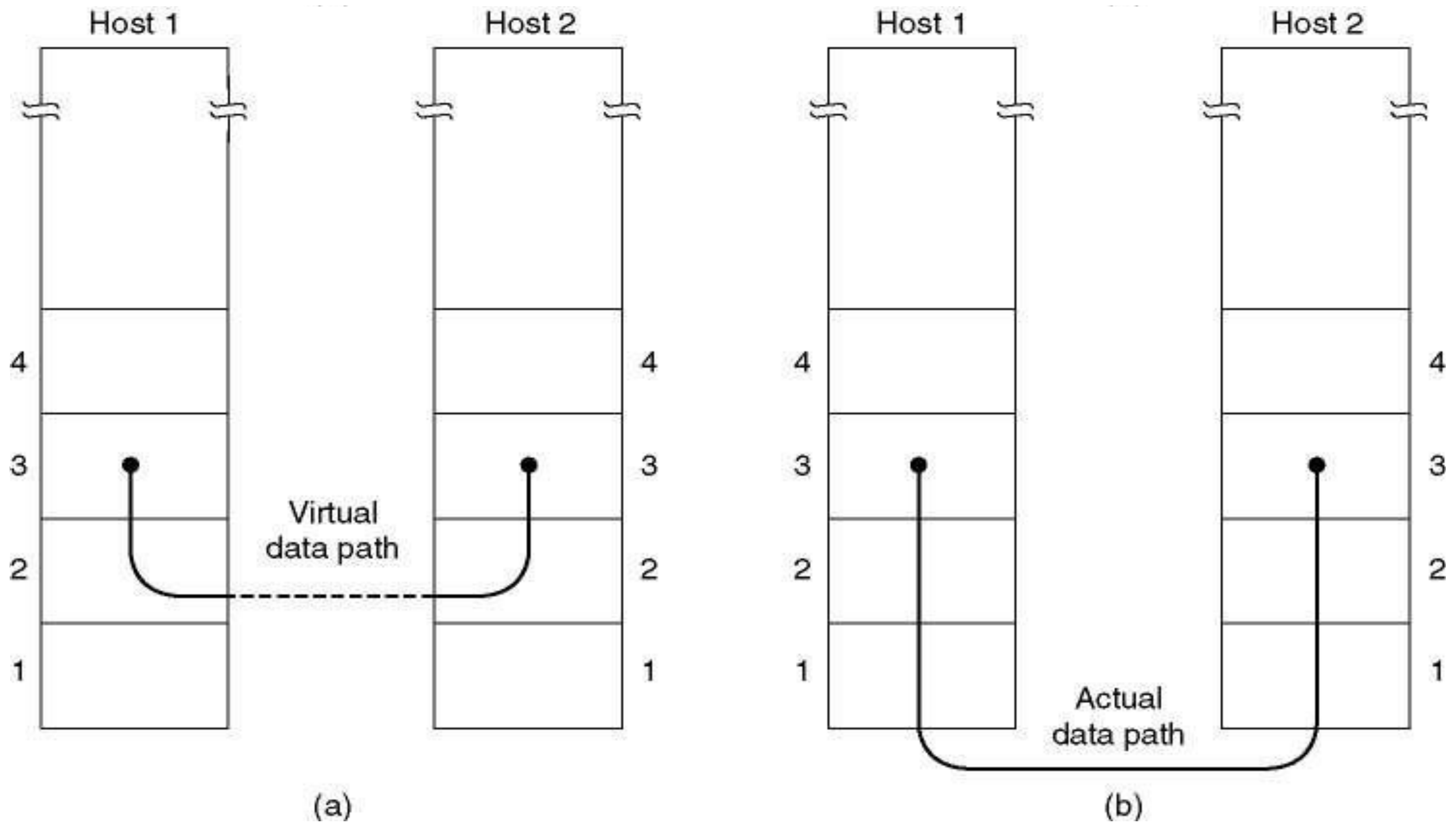
- Services Provided to the Network Layer
- Framing
- Error Control
- Flow Control

Services Provided to Network Layer (1/8)

- 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.

Services Provided to Network Layer (2/8)



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

Services Provided to Network Layer (3/8)

The data link layer can be designed to offer various services.

a) Unacknowledged connectionless service.

a) Acknowledged connectionless service.

a) Acknowledged connection-oriented service.

Services Provided to Network Layer (4/8)

a) 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.

Services Provided to Network Layer (5/8)

b) 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.

Services Provided to Network Layer (6/8)

c) 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.

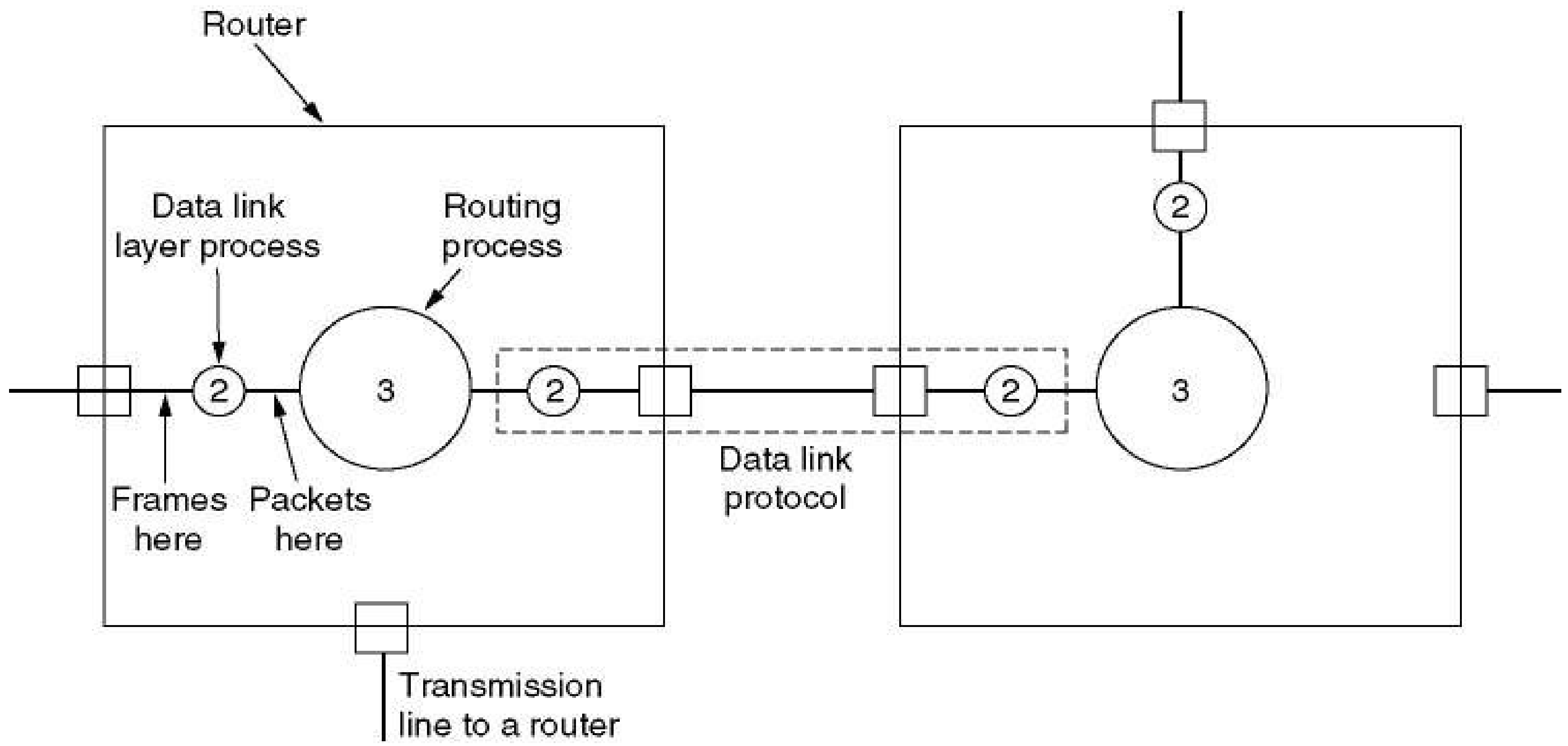
Services Provided to Network Layer (7/8)

c) Acknowledged connection-oriented service. There are three distinct phases in connection-oriented service.

- The connection is established;
- Frames are transmitted by this connection;
- The connection is released.

Services Provided to Network Layer (8/8)

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



Placement of the data link protocol.

Framing (1/12)

- 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.

Framing (2/12)

- 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.

Framing (3/12)

The methods to break the bit stream up into frames:

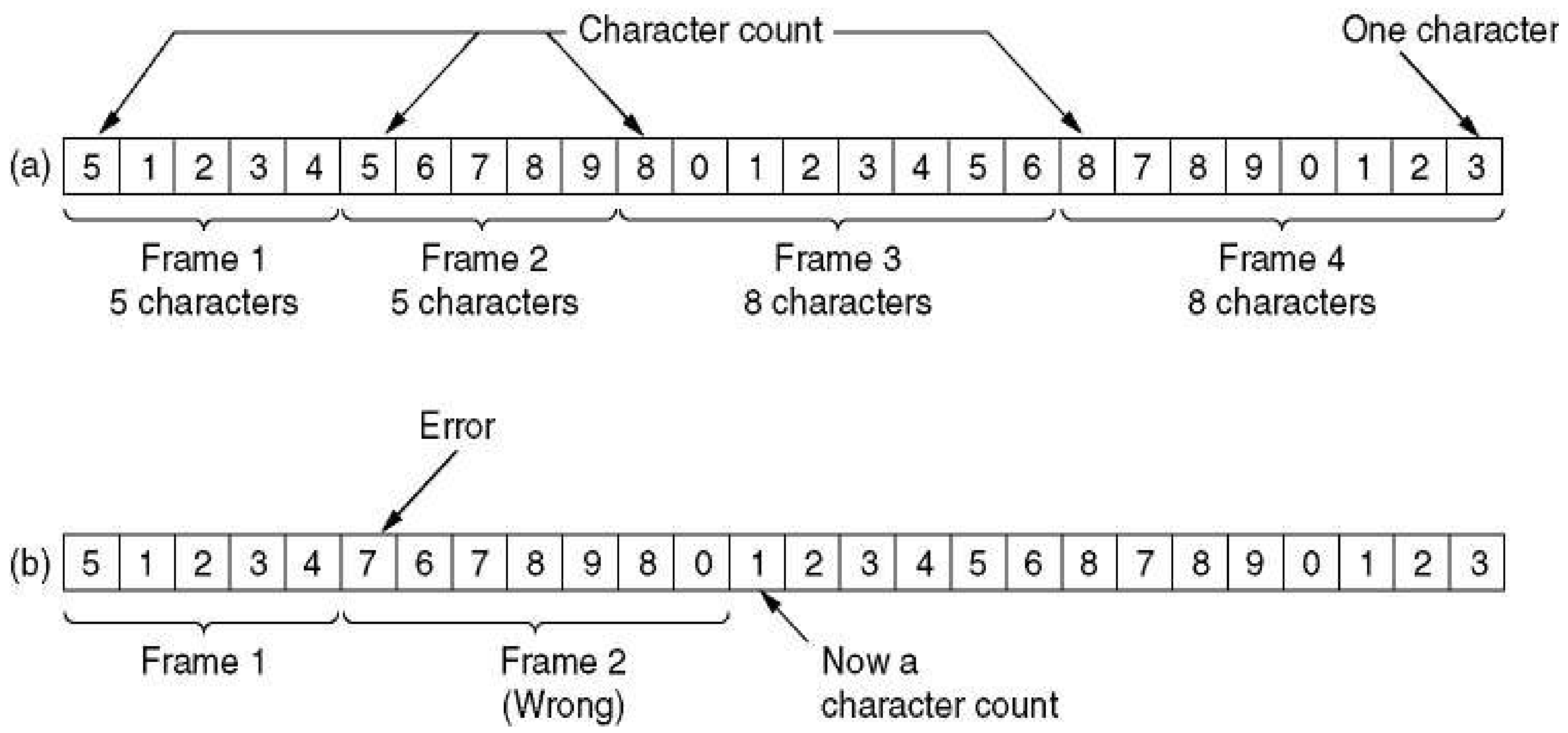
1) Character count;
2) Flag bytes with byte stuffing;
3) Starting and ending flags, with bit stuffing;
4) Physical layer coding violations.

Framing (4/12)

1) 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.

Framing (5/12)



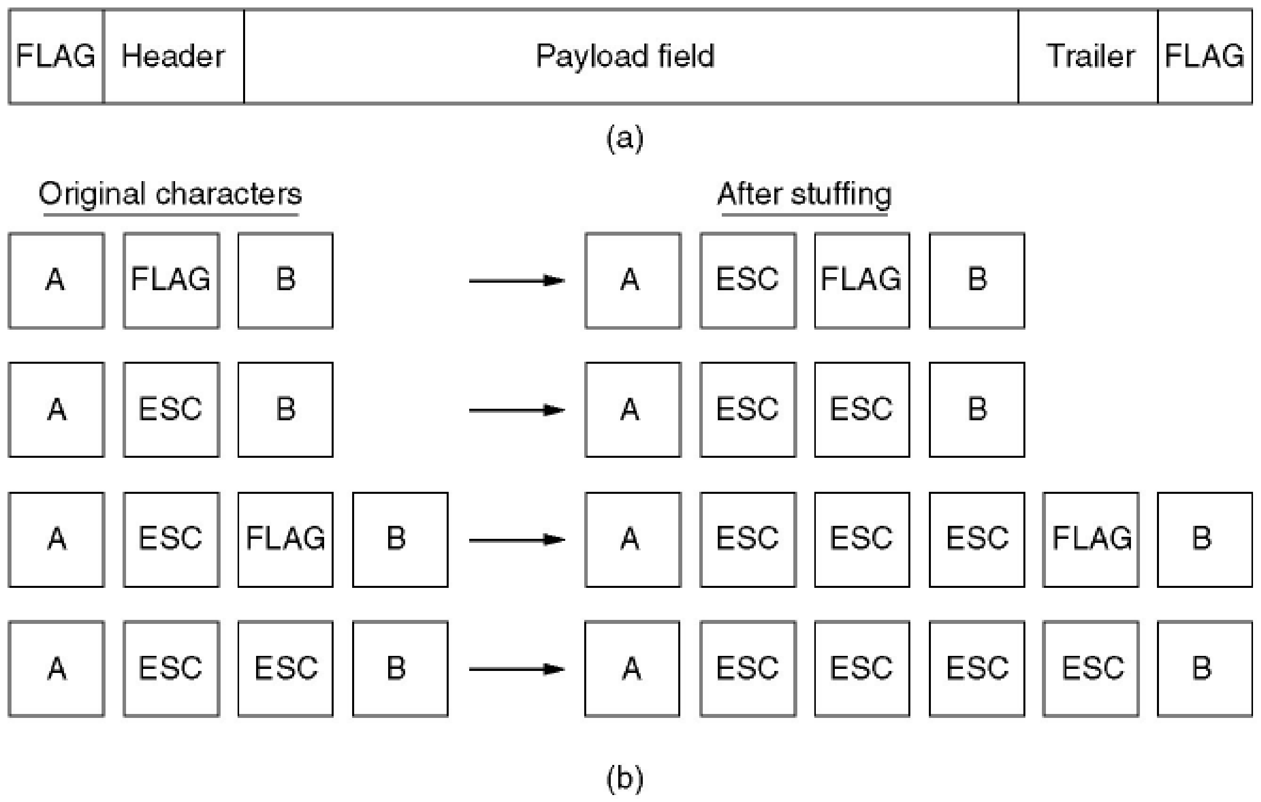
A character stream. (a) Without errors for four frames of sizes 5, 5,8, and 8 characters, respectively. (b) With one error.

Framing (6/12)

2) 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.

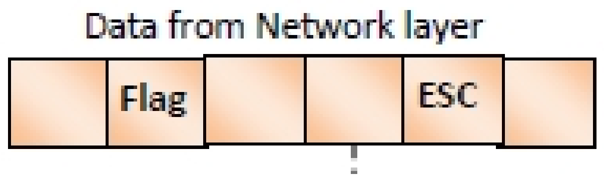
Framing (7/12)

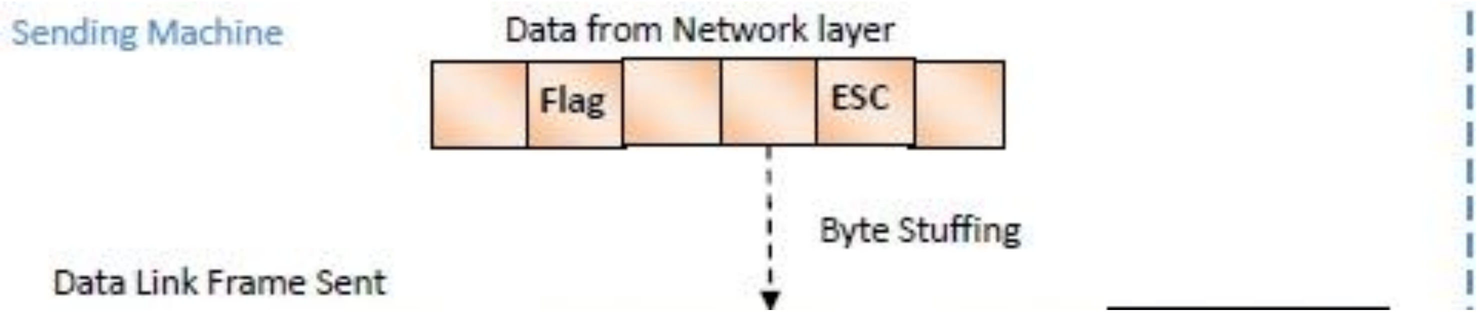


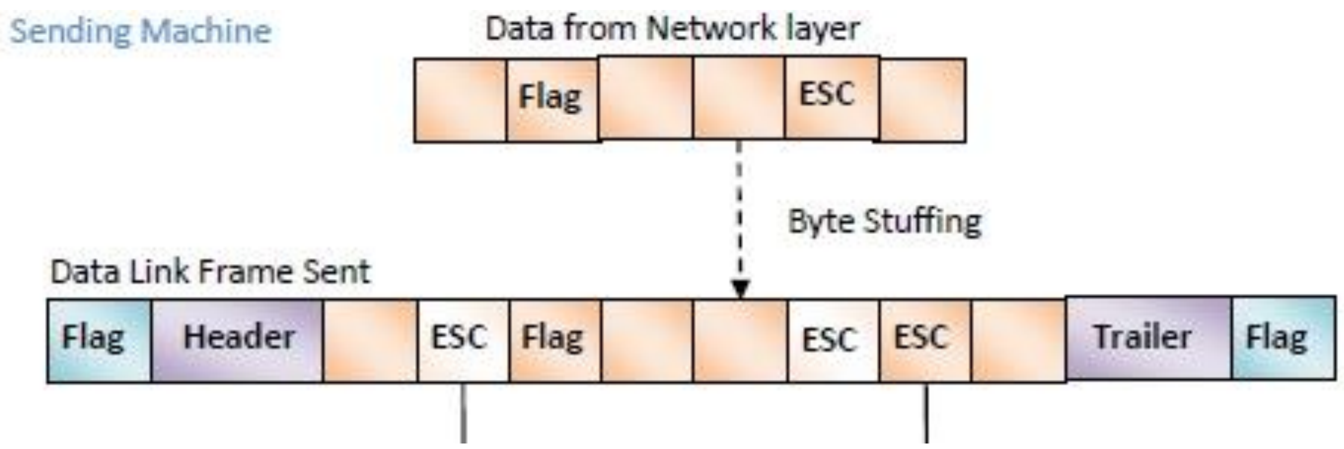
(a) A frame delimited by flag bytes.

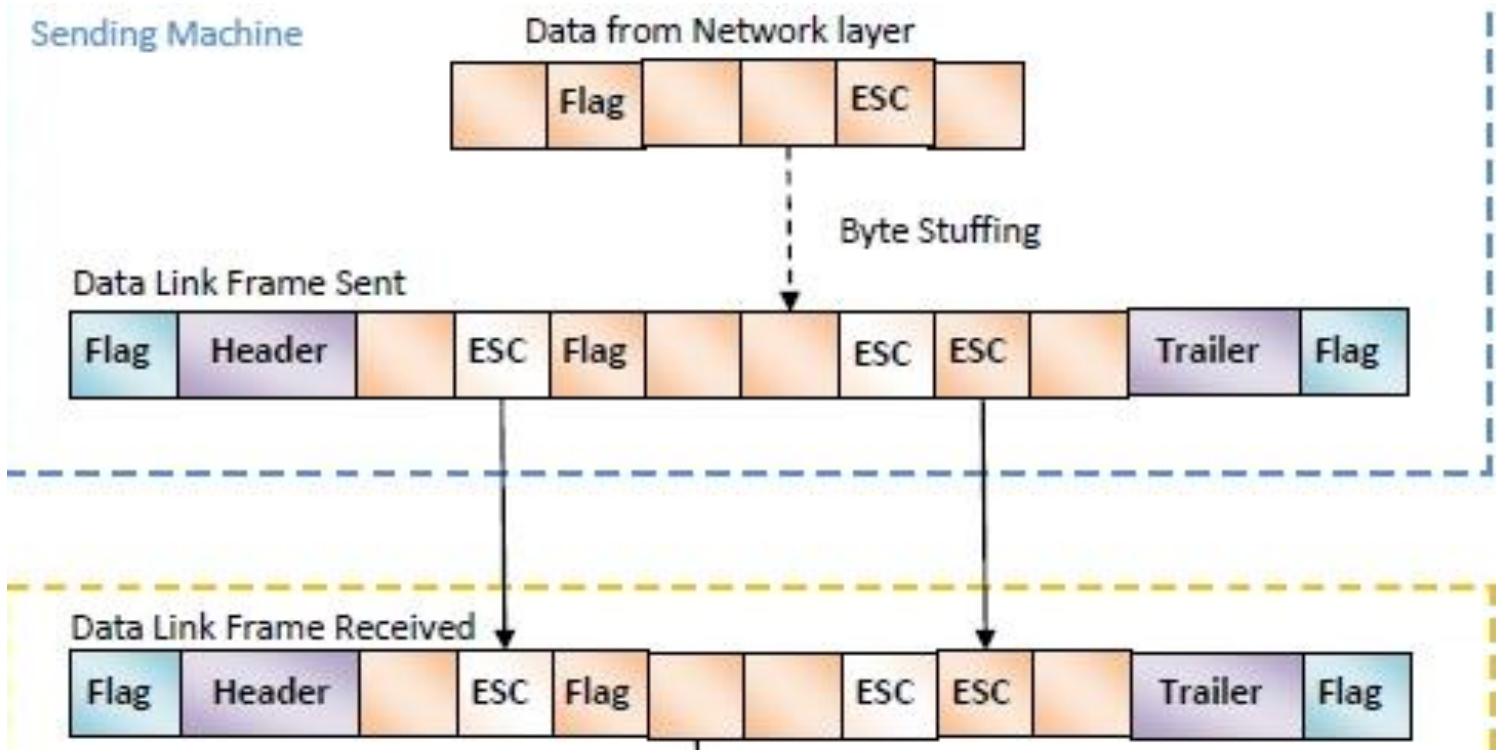
(b) Four examples of byte sequences before and after stuffing.

Sending Machine

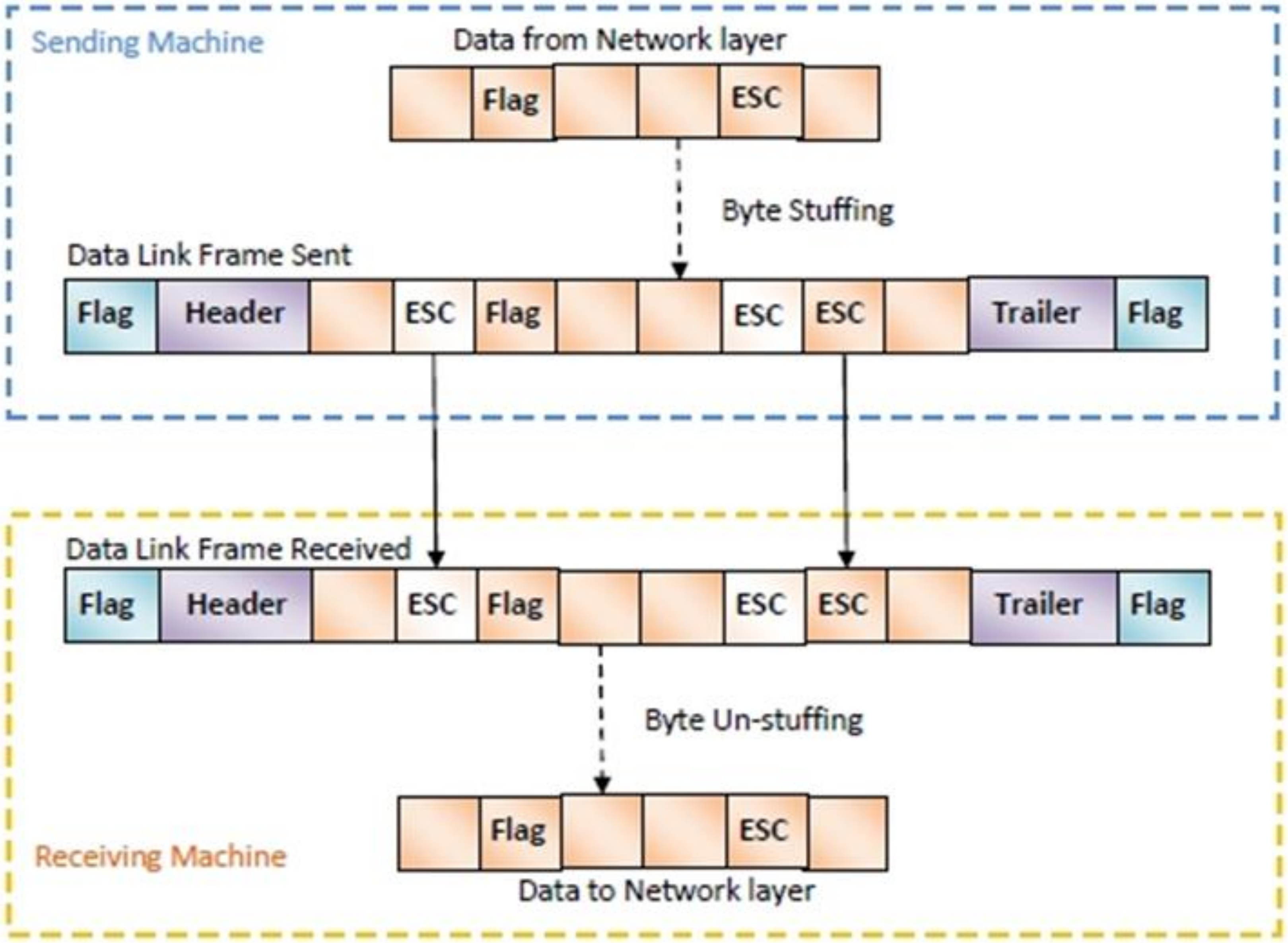



Sending Machine
Data from Network layer
Flag
ESC
Byte Stuffing
Data Link Frame Sent


Sending Machine
Data from Network layer
Flag
ESC
Byte Stuffing
Data Link Frame Sent
Flag
Header
ESC
Flag
ESC
ESC
Trailer
Flag


Sending Machine
Data from Network layer
Flag
ESC
Byte Stuffing
Data Link Frame Sent
Flag
Header
ESC
Flag
ESC
ESC
Trailer
Flag
Data Link Frame Received
Flag
Header
ESC
Flag
ESC
ESC
Trailer
Flag

Byte Stuffing Mechanism



Framing (8/12)

2) Flag bytes with byte stuffing – cont.

- 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

Framing (9/12)

2) Flag bytes with byte stuffing – cont.

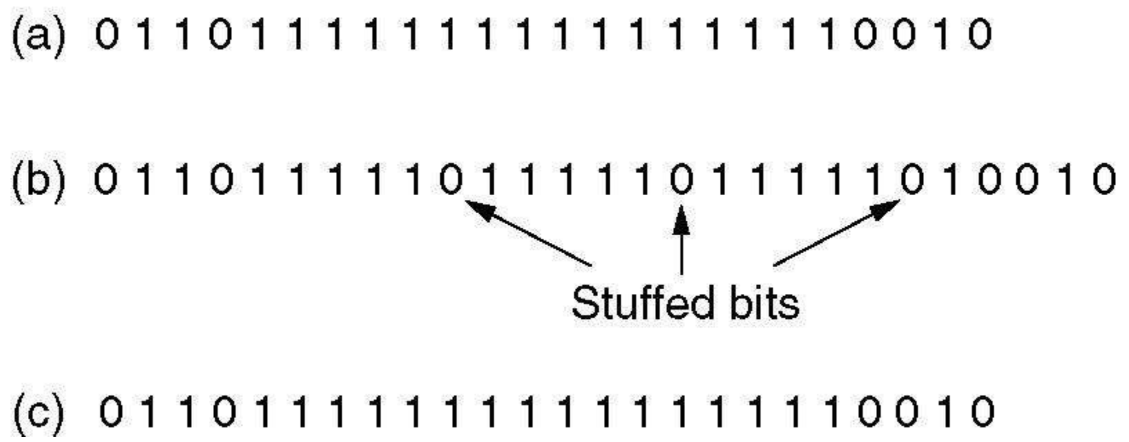
- A major disadvantage of using this framing method is that it is closely tied to the use of 8-bit characters.
- Not all character code use 8-bit characters.
- For example, UNICODE uses 16-bit characters.
- As networks developed, the disadvantages of embedding the character code length in the framing mechanism became more and more obvious, so a new technique had to be developed to allow arbitrary sized characters.

Framing (10/12)

3. 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.

Framing (11/12)


(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.

Problem

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

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

Show the bit sequence transmitted (in binary) for the four-character frame:

A B ESC FLAG when each of the following framing methods are used:

a) Character count.

b) Flag bytes with byte stuffing.

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

Problem

Solution:

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

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

a) Character count.

Problem

Solution:

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

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

a) Character count.

5	A	B	ESC	FLAG

Problem

Solution:

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

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

a) Character count.

5	A	B	ESC	FLAG

00000101	01000111	11100011	11100000	01111110

Problem

Solution:

1. b) flag bytes with byte stuffing.

A	B	ESC	FLAG

Problem

Solution:

1. b) flag bytes with byte stuffing.

FLAG	A	B	ESC	ESC	ESC	FLAG	FLAG

Problem

Solution:

1. b) flag bytes with byte stuffing.

FLAG	A	B	ESC	ESC	ESC	FLAG	FLAG

01111110	01000111	11100011	11100000	11100000	11100000	01111110	01111110

Problem

Solution:

1. c) flag bytes with bit stuffing.

A	B	ESC	FLAG

Problem

Solution:

1. c) flag bytes with bit stuffing.

A	B	ESC	FLAG

01111110	01000111	110100011	11100000	011111010	01111110

Problem

Q 2.

The following data fragment occurs in the middle of a data stream for which the byte-stuffing algorithm described in the text is used:

A B ESC C ESC FLAG FLAG D. What is the output after stuffing?

Problem

Q 2.

The following data fragment occurs in the middle of a data stream for which the byte-stuffing algorithm described in the text is used:

A B ESC C ESC FLAG FLAG D. What is the output after stuffing?

Solution:

flag bytes with byte stuffing.

A	B	ESC	C	ESC	FLAG	FLAG	D

Problem

Q 2.

The following data fragment occurs in the middle of a data stream for which the byte-stuffing algorithm described in the text is used:

A B ESC C ESC FLAG FLAG D. What is the output after stuffing?

Solution:

flag bytes with byte stuffing.

A	B	ESC	C	ESC	FLAG	FLAG	D

A	B	ESC	ESC	C	ESC	ESC	ESC	FLAG	ESC	FLAG	D

Problem

Q.3.

What is the maximum overhead in byte-stuffing algorithm?

Problem

Q.3.

What is the maximum overhead in byte-stuffing algorithm?

Solution:

The maximum overhead in byte stuffing algorithm is 100%(i.e. when the payload contains only ESC and Flag bytes).

Problem

Q 4.

A bit string, 0111101111101111110, needs to be transmitted at the data link layer. What is the string actually transmitted after bit stuffing?

Problem

Q 4.
A bit string, 0111101111101111110, needs to be transmitted at the data link layer. What is the string actually transmitted after bit stuffing?

Solution:

The actual transmitted bit string after bit stuffing is 011110111110011111010

Framing (12/12)

4) 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

Error Control (1/1)

- 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 frames must be transmitted again.

Flow Control (1/2)

- 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.

Flow Control (2/2)

- 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 (1/5)

- 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)

Error Detection and Correction (2/5)

- 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.
- We have to learn how to deal with them.

Error Detection and Correction (3/5)

- As a result of the physical processes that generate them, errors on some media (e.g., radio) tend to come in bursts rather than single.
- 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 Detection and Correction (4/5)

- 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.

Error Detection and Correction (5/5)

- 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 (1/11)

- 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*.

Error-Correcting Codes (2/11)

- 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-Correcting Codes (3/11)

- The number of bit positions in which two code words 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.

Error-Correcting Codes (4/11)

- 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.

Error-Correcting Codes (5/11)

- 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-Correcting Codes (6/11)

- 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).

Error-Correcting Codes (7/11)

- 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 (8/11)

- 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.

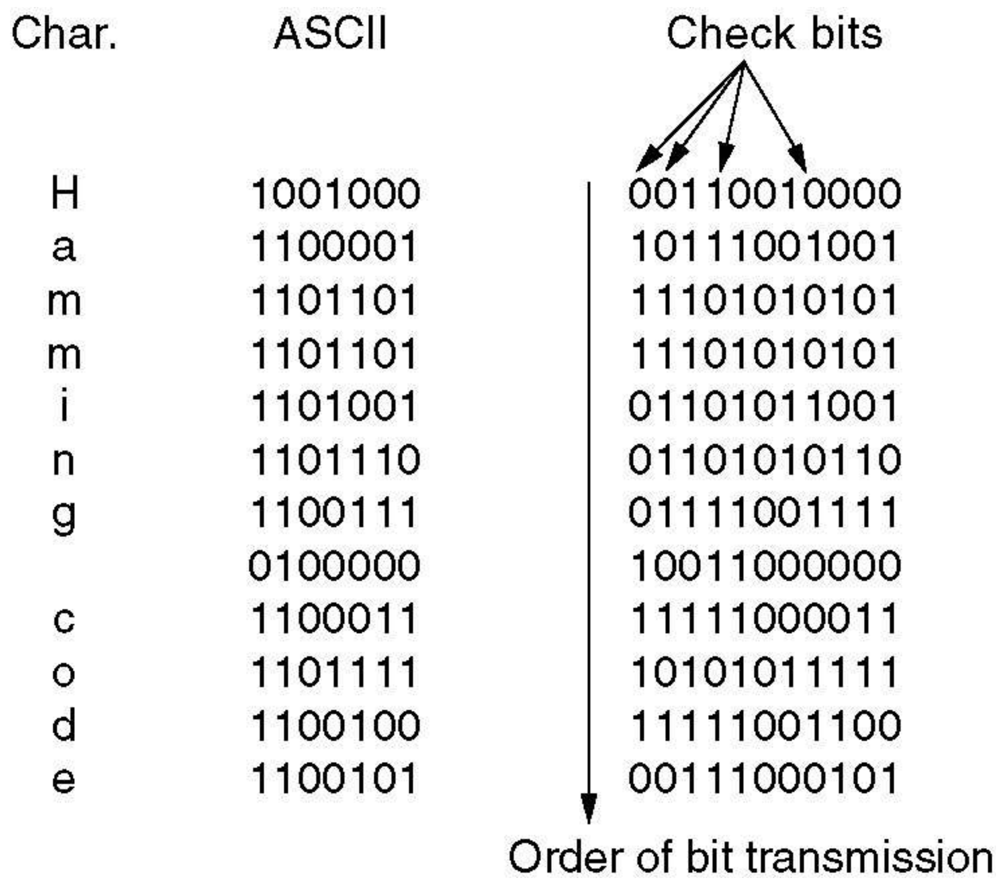
Error-Correcting Codes (9/11)

- 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 (10/11)

- 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.

Error-Correcting Codes (11/11)


Order of bit transmission

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

Use of a Hamming code to correct burst errors.

Error-Detecting Codes (1/12)

- 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.

Error-Detecting Codes (2/12)

- 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.

Error-Detecting Codes (3/12)

- Example (cont.):
- 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-Detecting Codes (4/12)

- 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 $x^5+x^4+x^0$

Error-Detecting Codes (5/12)

- 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 (6/12)

- 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-Detecting Codes (7/12)

The algorithm for computing the checksum is as follows:

1) Let r be the degree of $G(x)$. Append r zero bits to the low- order end of the frame so it now contains $m+r$ bits and corresponds to the polynomial $x^r M(x)$.

Error-Detecting Codes (8/12)

The algorithm for computing the checksum is as follows:

1) Let r be the degree of $G(x)$. Append r zero bits to the low- order 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.

Error-Detecting Codes (9/12)

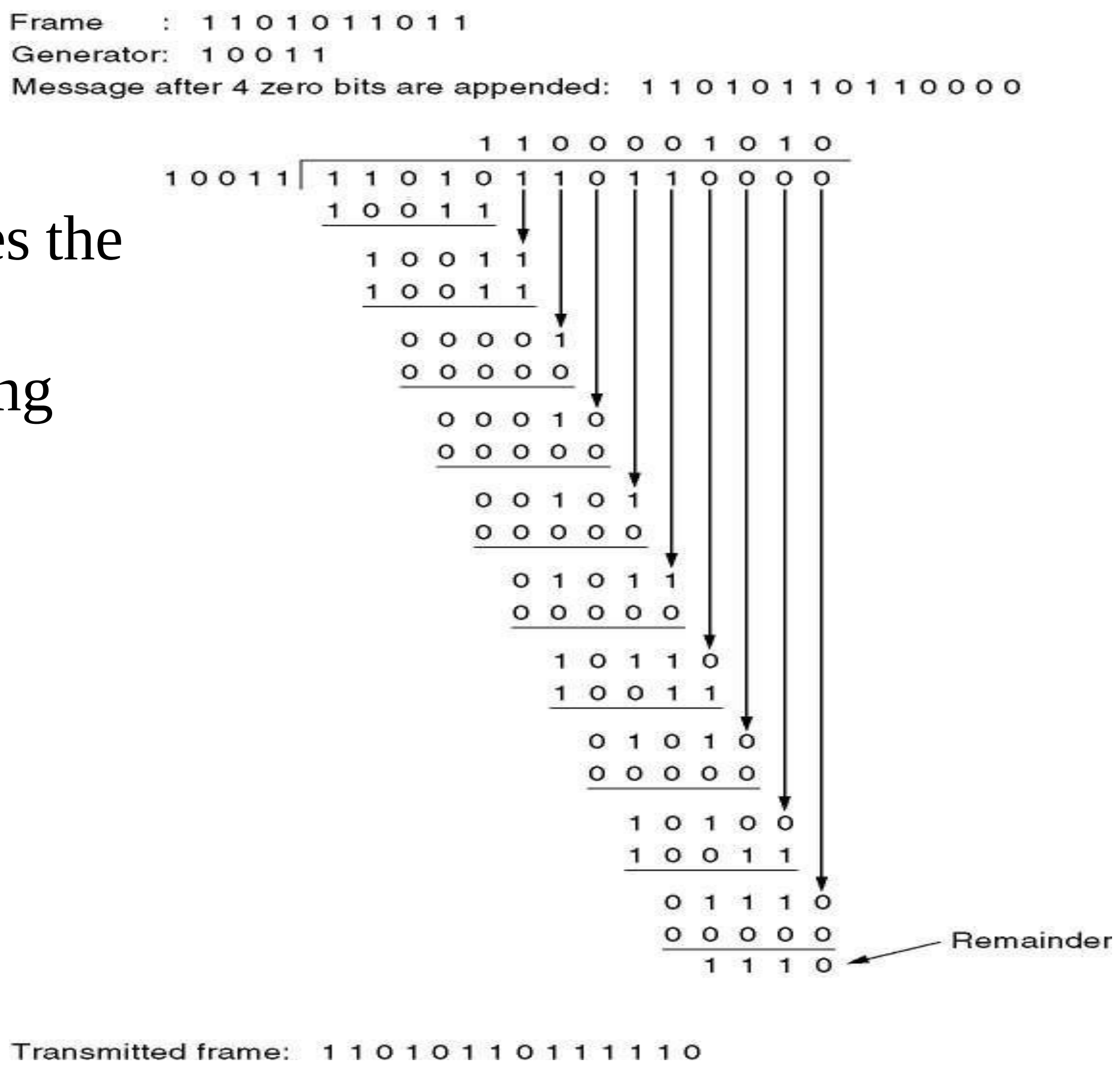
The algorithm for computing the checksum is as follows:

1) Let r be the degree of $G(x)$. Append r zero bits to the low- order 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 te checksummed frame to be transmitted. Call its polynomial $T(x)$.

Error-Detecting Codes (10/12)



This example illustrates the calculation for a frame *M(t)= 1101011011* using the generator *$G(x)=x^4+x+1$. r=4*

$x^rM(x)$=11010110110000

T(x)=11010110111110

E(x) is error polynomial

Calculation of the polynomial code checksum.

Error-Detecting Codes (11/12)

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 *0*s and *1*s, and a final *1*, with all other bits being *0*.

Error-Detecting Codes (12/12)

- 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.

Problem

Ex-5:

Let us assume that m = 3 and *n* = 4. Find the list of valid datawords and codewords assuming the check bit is used to indicate even parity in the code word.

Problem

Ex-5:

Let us assume that m = 3 and *n* = 4. Find the list of valid datawords and codewords assuming the check bit is used to indicate even parity in the code word.

Solution:

Valid datawords : 000, 001,010, 011,100,101,110,111

Valid codewords : 0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111

Problem

Ex-6:

What is the Hamming distance for each of the following codewords:

a. (10000, 00000)

b. (10101, 10000)

c. (11111,11111)

d.(000, 000)

Problem

Ex-6:

What is the Hamming distance for each of the following codewords:

a. (10000, 00000)

b. (10101, 10000)

c. (11111,11111)

d.(000, 000)

- Solution:
- a. **1**
- b. **2**
- c. **0**
- d. **0**

Problem

Ex-7:

Given the codeword of size 4 bit. If the size of dataword is 3 bit. What is the value of hamming distance for the codeword?

Solution :

Hamming distance = 2

Problem

Ex-8:

To provide more reliability than a single parity bit can give, an error-detecting coding scheme uses one parity bit for checking all the odd-numbered bits and a second parity bit for all the even-numbered bits. What is the Hamming distance of this code?

Solution:

Making one change to any valid character cannot generate another valid character due to the nature of parity bits. Making two changes to even bits or two changes to odd bits will give another valid character, so the distance is 2.

Problem

Ex-9:

Find the minimum Hamming distance to be implemented in codeword for the following cases:

a. Detection of two errors.

b. Correction of two errors.

c. Detection of 3 errors or correction of 2 errors.

d. Detection of 6 errors or correction of 2 errors.

Problem

Ex-9:

Solution:

a. For error detection → Hamming distance = d + 1 = 2 + 1 = **3**

b. For error correction → Hamming distance = 2d + 1 = 2 × 2 + 1 = **5**

c. For error detection → Hamming distance = d + 1 = 3 + 1 = **4**

For error correction → Hamming distance = 2d + 1 = 2 × 2 + 1 = **5**

Therefore minimum Hamming distance should be **5**.

d. For error detection → Hamming distance = d + 1 = 6 + 1 = **7**

For error correction → Hamming distance = 2d + 1 = 2 × 2 + 1 = **5**

Therefore minimum Hamming distance should be **7**.

Problem

- Ex-10:
- Given in the table a set of valid dataword and codeword.

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

- What is the dataword transmitted for the following codewords received assuming there is 1 bit error?
- 01010
- 11010

Problem

Ex-10:

Solution:

a. 01

b. 11

Problem

Ex-11:

Sixteen-bit messages are transmitted using a Hamming code. How many check bits are needed to ensure that the receiver can detect and correct single bit errors? Show the bit pattern transmitted for the message 1101001100110101. Assume that even parity is used in the Hamming code.

Problem

Ex-11:

Sixteen-bit messages are transmitted using a Hamming code. How many check bits are needed to ensure that the receiver can detect and correct single bit errors? Show the bit pattern transmitted for the message 1101001100110101. Assume that even parity is used in the Hamming code.

Solution:

5 check bits are needed at positions 1, 2, 4, 8, and 16.

The bit pattern transmitted for the message 1101001100110101 is

Codeword:011110110011001110101

Problem

Ex-12:

An 8 bit message using even-parity Hamming code is received as **101001001111**. Find the 8 bit message after getting decoded assuming no error during transmission?

Problem

Ex-12:

An 8 bit message using even-parity Hamming code is received as **101001001111**. Find the 8 bit message after getting decoded assuming no error during transmission?

Solution:

The 8 bit message after decoding is 10101111.

- Ex-13:

Problem

- A 12-bit Hamming code whose hexadecimal value is 0xE4F arrives at a receiver. What was the original value in hexadecimal? Assume that not more than 1 bit is in error.

- Solution:

- If we number the bits from left to right starting at bit 1, in this example bit 2 (a parity bit) is incorrect. The 12-bit value transmitted (after Hamming encoding) was 0xA4F. The original 8-bit data value was 0xAF.

Problem

- Ex-14:
- Suppose that data are transmitted in blocks of sizes 1000 bits. What is the maximum error rate under which error detection and retransmission mechanism (1 parity bit per block) is better than using Hamming code? Assume that bit errors are independent of one another and no bit error occurs during retransmission.
- Solution:
- From Eq. $\mathbf{(m+r+1)\leq 2^r}$, we know that 10 check bits are needed for each block in case of using Hamming code. Total bits transmitted per block are 1010 bits. In case of error detection mechanism, one parity bit is transmitted per block (i.e.1001). Suppose error rate is x per bit. Thus, a block may encounter a bit error $1000x$ times. Every time an error is encountered, 1001 bits have to be retransmitted. So, total bits transmitted per block are $1001+1000x \times 1001$ bits. For error detection and retransmission to be better, $1001 + 1000x \times 1001 <1010$. So, the error rate must be less than 9×10^{-6}.

Problem

Ex-15:

What is the remainder obtained by dividing $x^7 + x^5 + 1$ by the generator polynomial $x^3 + 1$?

Solution:

- The remainder is x^2+x+1.

Problem

Ex-16:

Given the dataword 101001111 and the divisor 10111. Show the generation of the CRC codeword at the sender site (using binary division).

Solution:

The codeword at the sender site is 1010011110001

Problem

Ex-17

A bit stream 10101010 is transmitted using the standard CRC method. The generator polynomial is x^3+x^2+1. Show the actual bit string transmitted. Suppose the second bit from the left is inverted during transmission. Show that this error is detected at the receiver's end.

Solution:

The frame is 10101010. The generator is 1101. We must append 3 zeros to the message (i.e. 10101010000).The remainder after dividing 10101010000 by 1101 is 110. So actual bit string transmitted is 10101010110.Since the second bit from left is inverted during transmission, the bits received are 11101010110. Dividing this by 1101 doesn't give remainder 0. So the received bits contain error.

Problem

Ex-18:

A bit stream 10011101 is transmitted using the standard CRC method described in the text. The generator polynomial is $x^3 + 1$. Show the actual bit string transmitted. Suppose the third bit from the left is inverted during transmission. Show that this error is detected at the receiver's end.

Solution:

The frame is 10011101. The generator is 1001. The message after appending three zeros is 10011101000. The remainder on dividing 10011101000 by 1001 is 100. So, the actual bit string transmitted is 10011101100. The received bit stream with an error in the third bit from the left is 10111101100. Dividing this by 1001 produces a remainder 100, not 0. So the received bits contain error and needs retransmission.

Problem

Ex-19:

A channel has a bit rate of 4 kbps and a propagation delay of 20 msec. For what range of frame sizes does stop-and-wait give an efficiency of at least 50%?

Solution:

Efficiency will be 50% when the time required to transmit the frame equals the round-trip propagation delay. At a transmission rate of 4 bits/msec, 160 bits takes 40 msec. For frame sizes above 160 bits, stop-and-wait is reasonably efficient.

Problem

Ex-20:

A 3000-km-long T1 trunk is used to transmit 64-byte frames using protocol 5. If the propagation speed is 6 μsec/km, how many bits should the sequence numbers be?

Solution:

To operate efficiently, the sequence space (actually, the send window size) must be large enough to allow the transmitter to keep transmitting until the first acknowledgement has been received. The propagation time is 18 ms. At T1 speed, which is 1.536 Mbps (excluding the 1 header bit), a 64-byte frame takes 0.300 msec. Therefore, the first frame fully arrives 18.3 msec after its transmission was started. The acknowledgement takes another 18 msec to get back, plus a small (negligible) time for the acknowledgement to arrive fully. In all, this time is 36.3 msec. The transmitter must have enough window space to keep going for 36.3 msec. A frame takes 0.3 ms, so it takes 121 frames to fill the pipe. Seven-bit sequence numbers are needed.

Elementary Data Link Protocols (1/14)

- Three protocols of increasing complexity:
- ASSUMPTION 1:
- In the physical layer (PhL), data link layer (DLL), and network layer (NL) are independent processes that comunicate 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.

Elementary Data Link Protocols (2/14)

- 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 (3/14)

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.

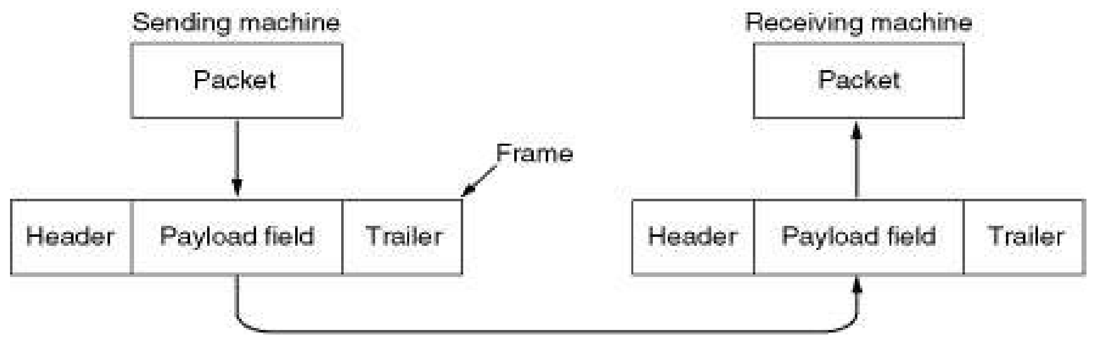
Elementary Data Link Protocols (4/14)

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 (5/14)

- 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 (6/14)

- 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.

Elementary Data Link Protocols (7/14)

- 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 (8/14)

- 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* bytes, but may be of variable length contains MAX_PKT

•

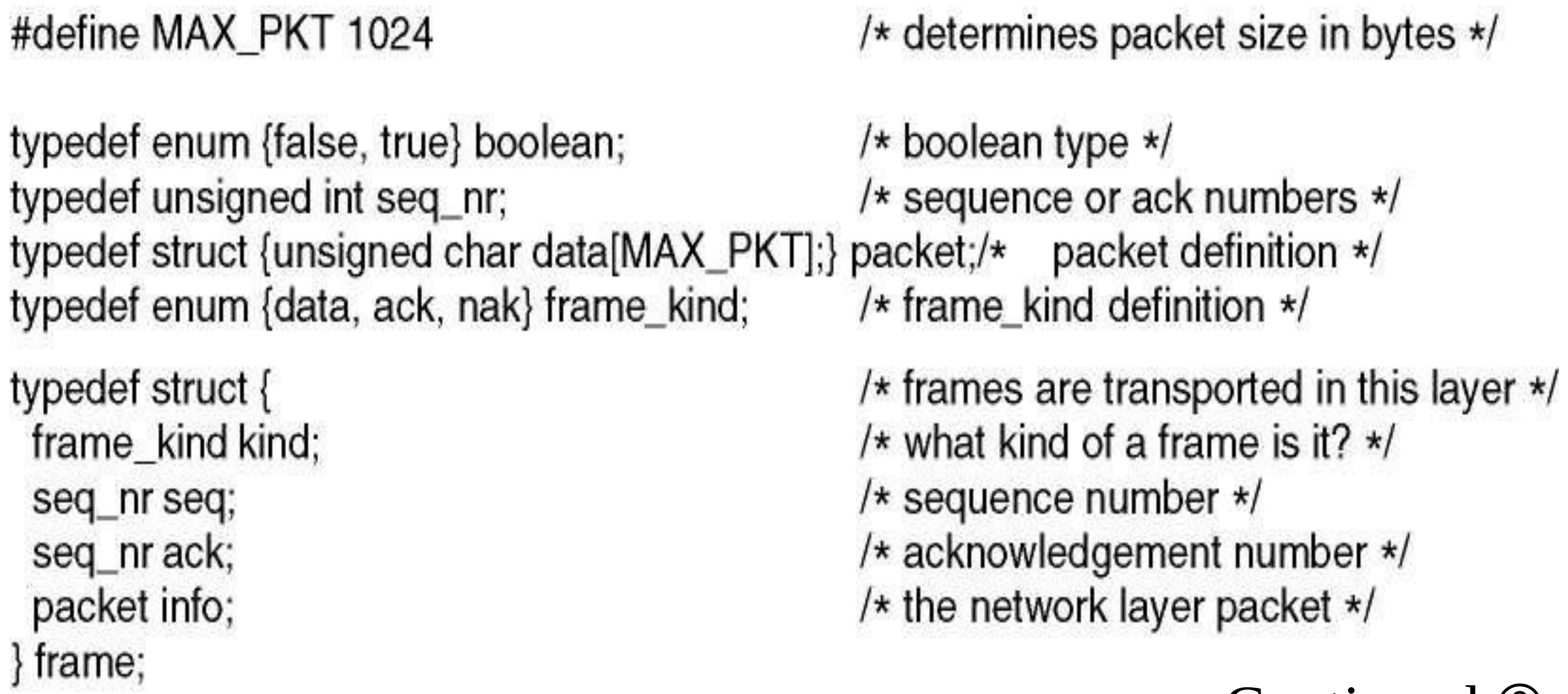
Elementary Data Link Protocols (9/14)

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.

c) 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 (10/14)



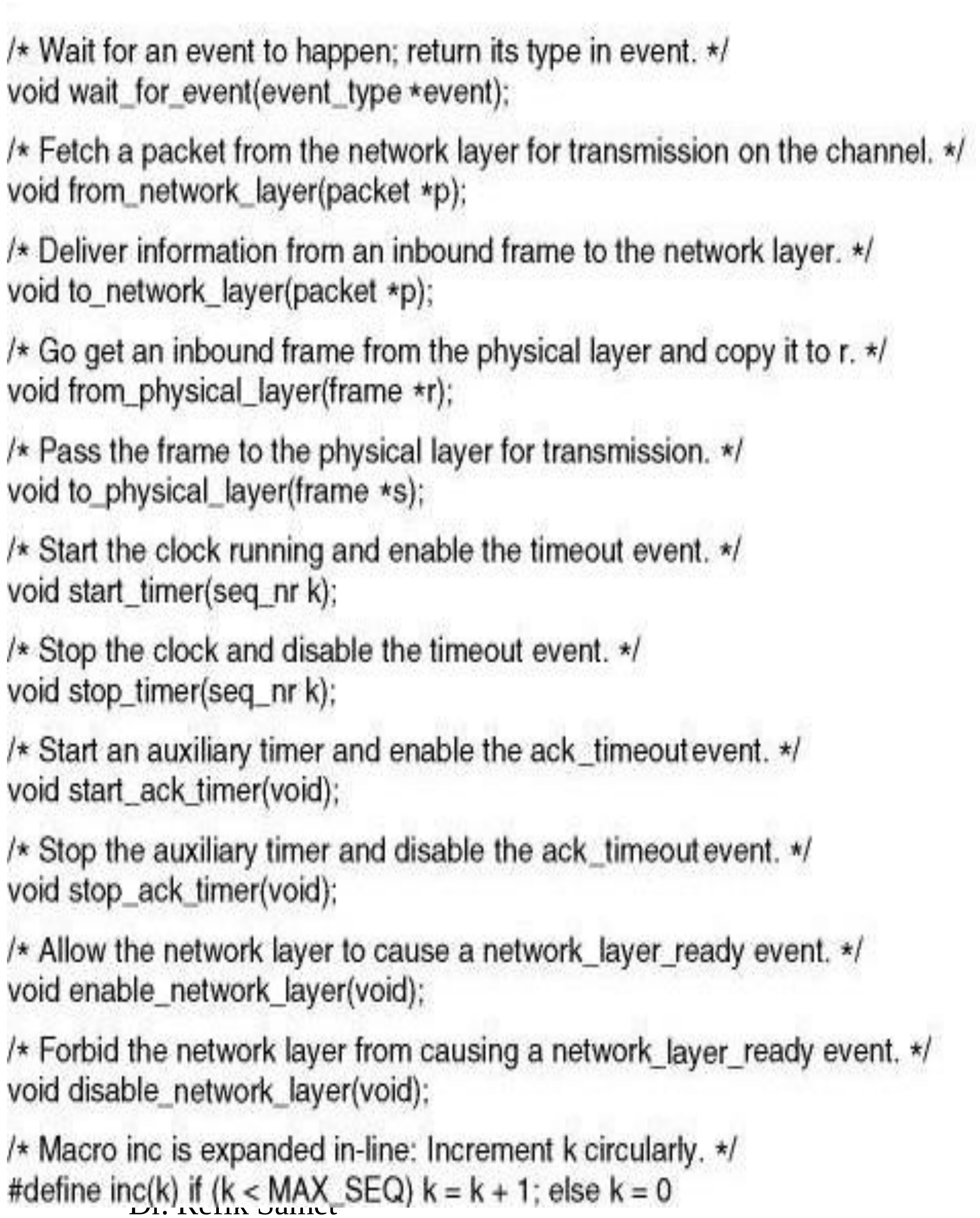
```
#define MAX_PKT 1024                                    /* determines packet size in bytes */

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;
```

Continued ⓪

Some definitions needed in the protocols to follow.

Protocol Definitions (ctd.) (11/14)

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



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

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

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

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

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

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

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

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

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

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

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

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

Elementary Data Link Protocols (12/14)

- 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.

Elementary Data Link Protocols (13/14)

- 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.

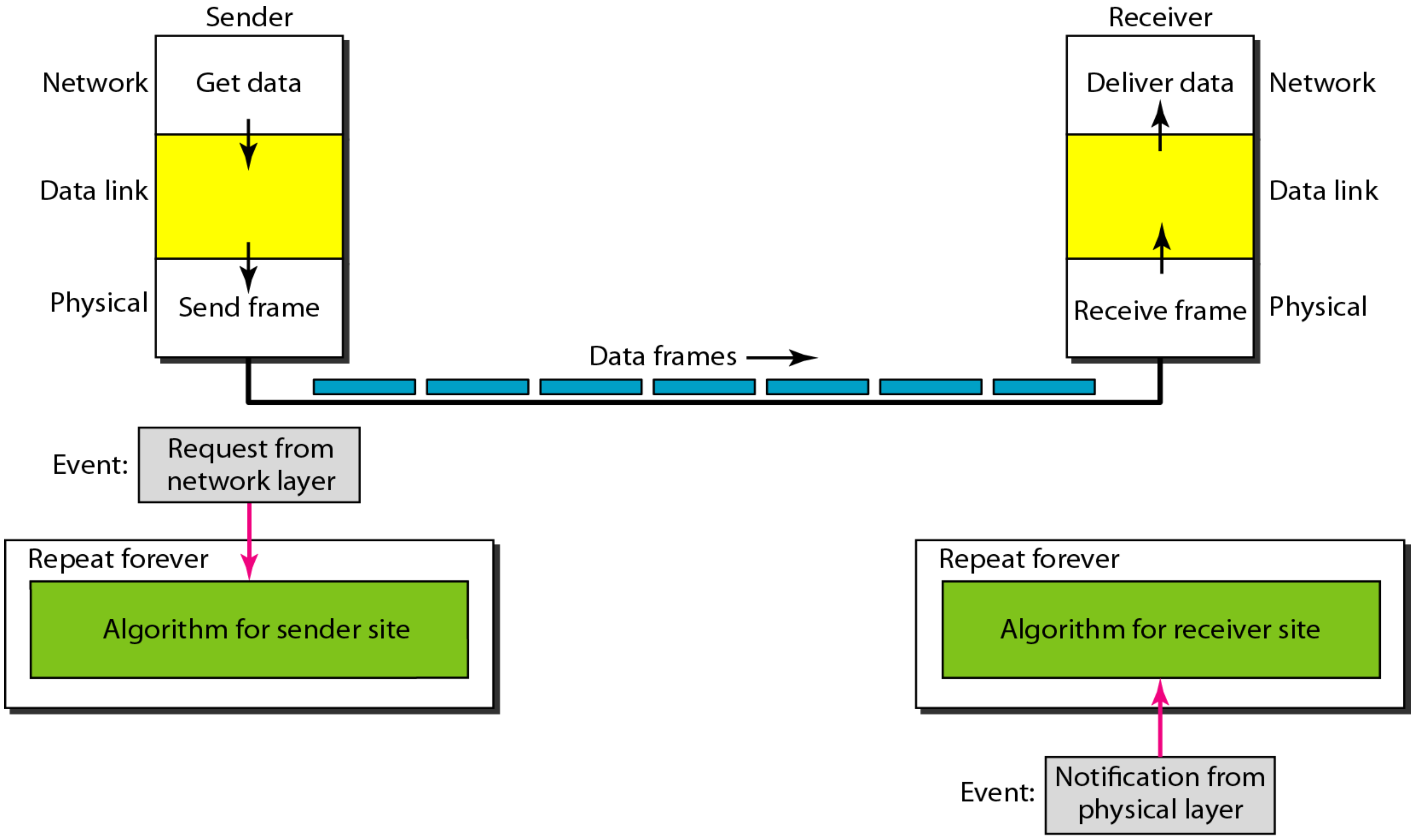
Elementary Data Link Protocols (14/14)

- 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: Unrestricted Simplex Protocol (1/2)

- 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”.

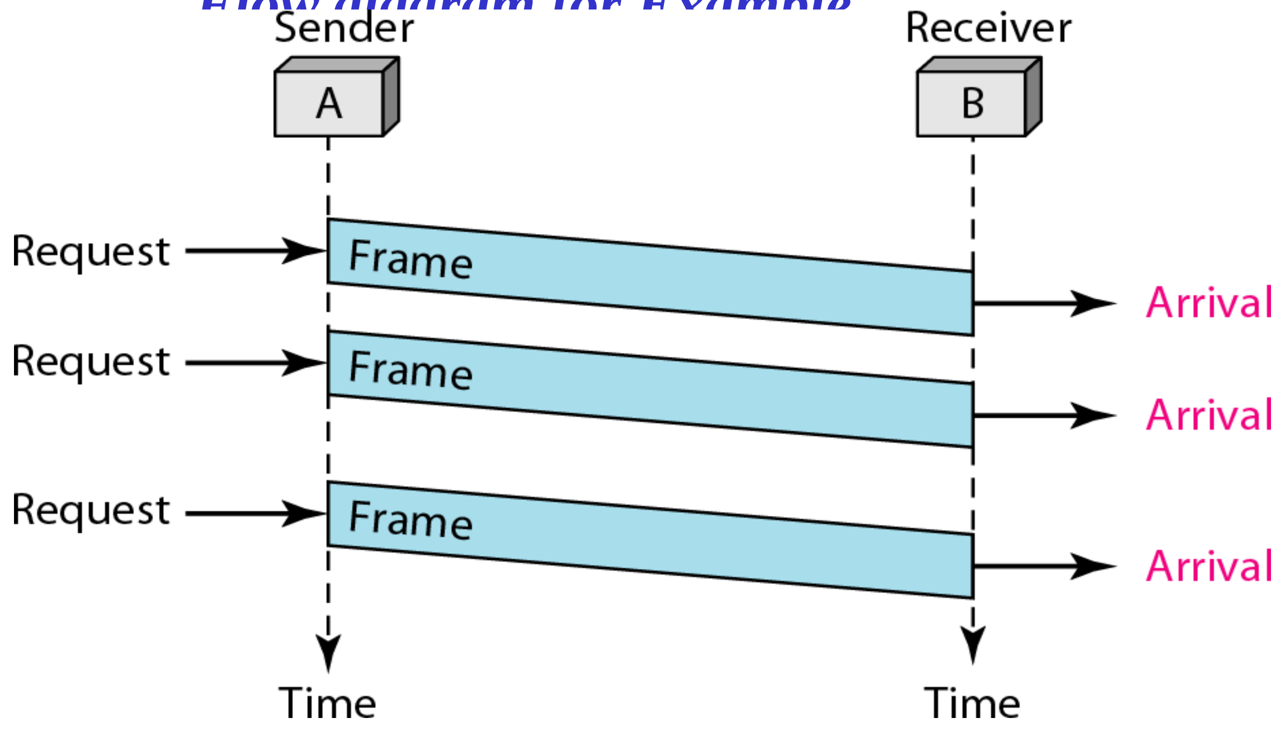
The design of the simplest protocol with no flow or error control



Example

Figure. shows an example of communication using this protocol. It is very simple. The sender sends a sequence of frames without even thinking about the receiver. To send three frames, three events occur at the sender site and three events at the receiver site. Note that the data frames are shown by tilted boxes; the height of the
box defines the transmission time difference between the first bit and the last bit in
the frame.

Flow diagram for Example



Unrestricted Simplex Protocol (2/2)

```
/* 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 */
}

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 (1/3)

- 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.

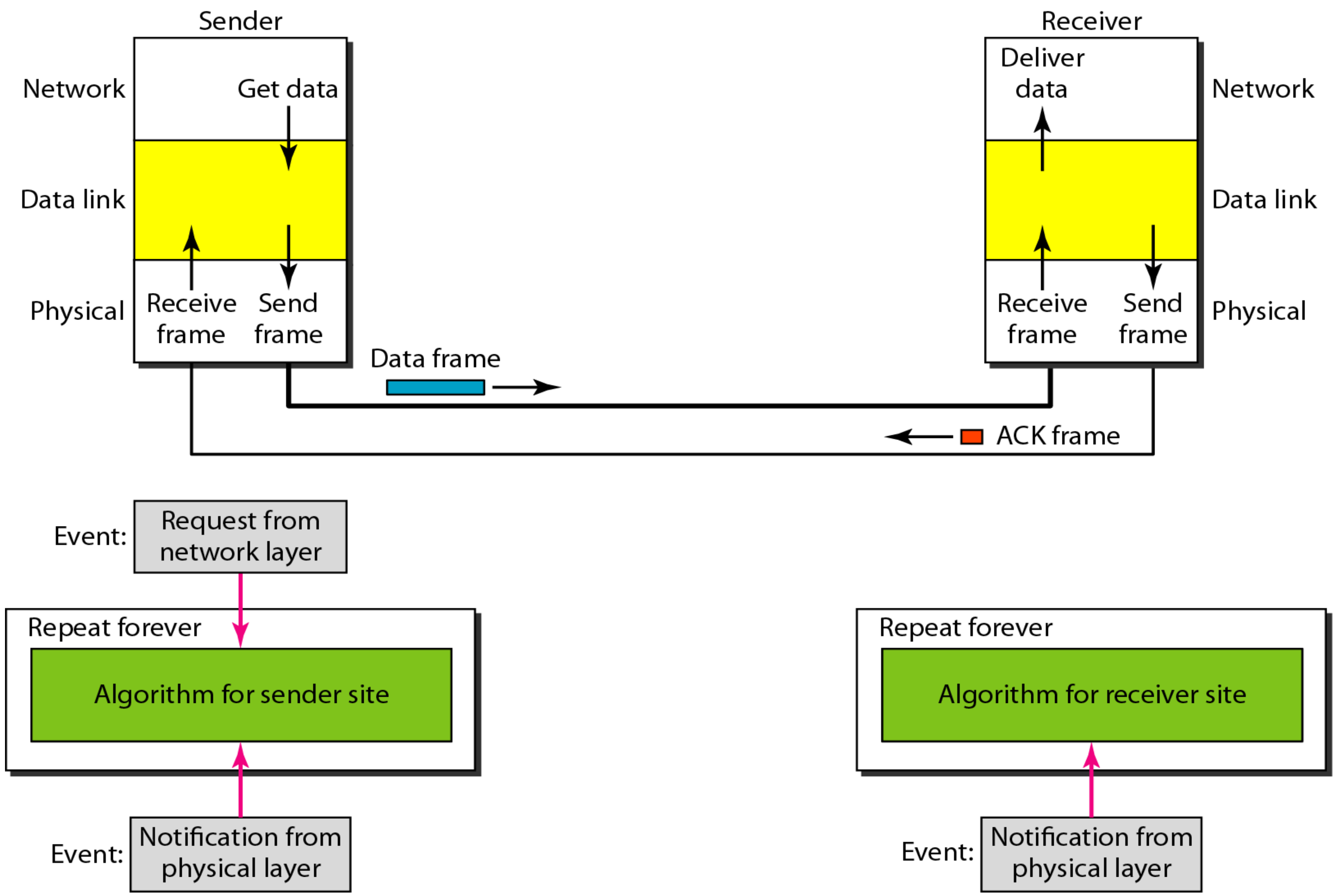
Stop-and-Wait Protocol

- If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources. We need to tell the sender to slow down. There must be feedback from the receiver to
  the sender.

  The sender sends one frame, stops until it receives agreement the receiver (okay to go ahead), and then sends the next frame.

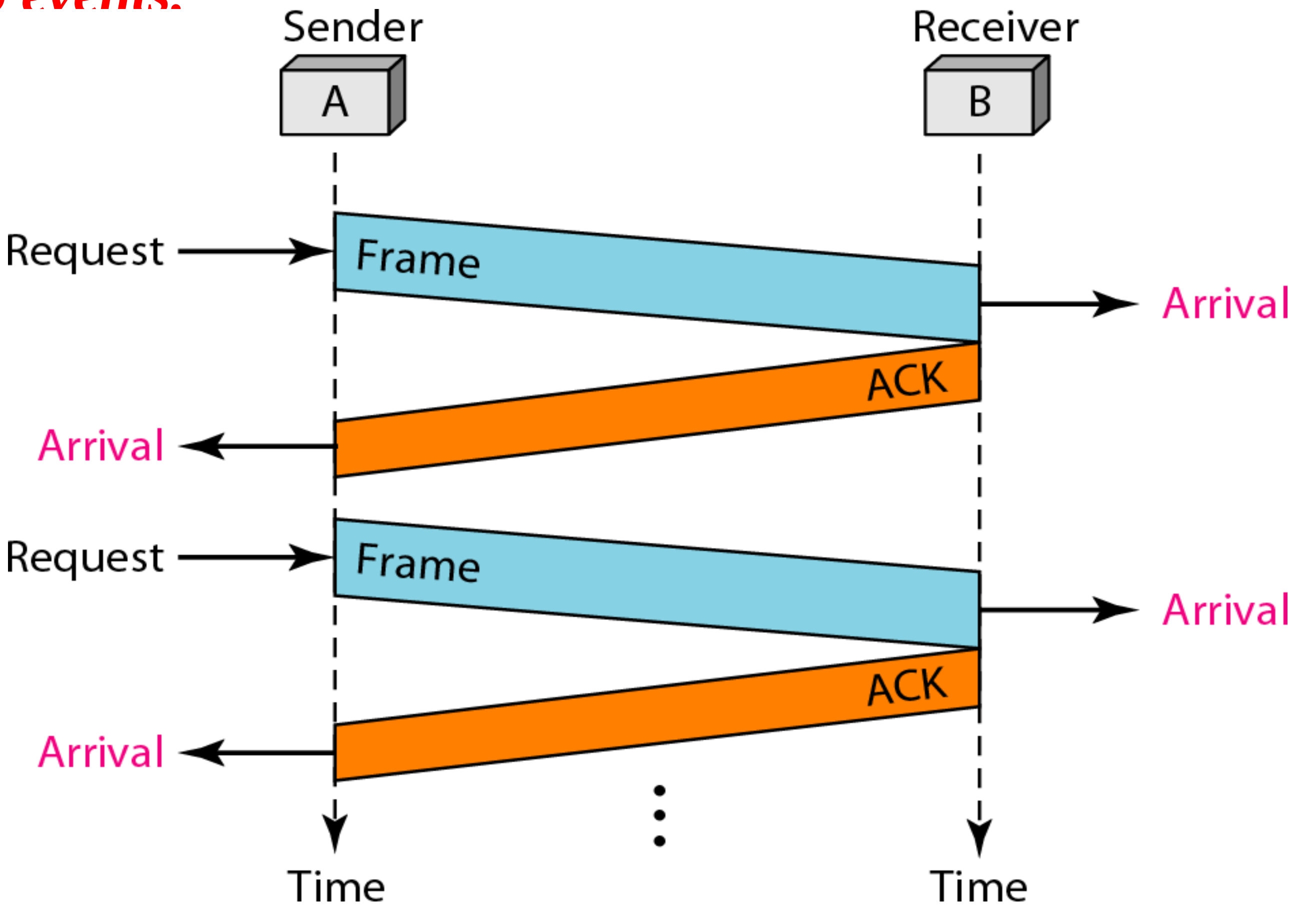
We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction.

Design of Stop-and-Wait Protocol



Example

Figure shows an example of communication using this protocol. It is still very simple.
The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame.
Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.



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 */
  }
}

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 */
  }
}
```