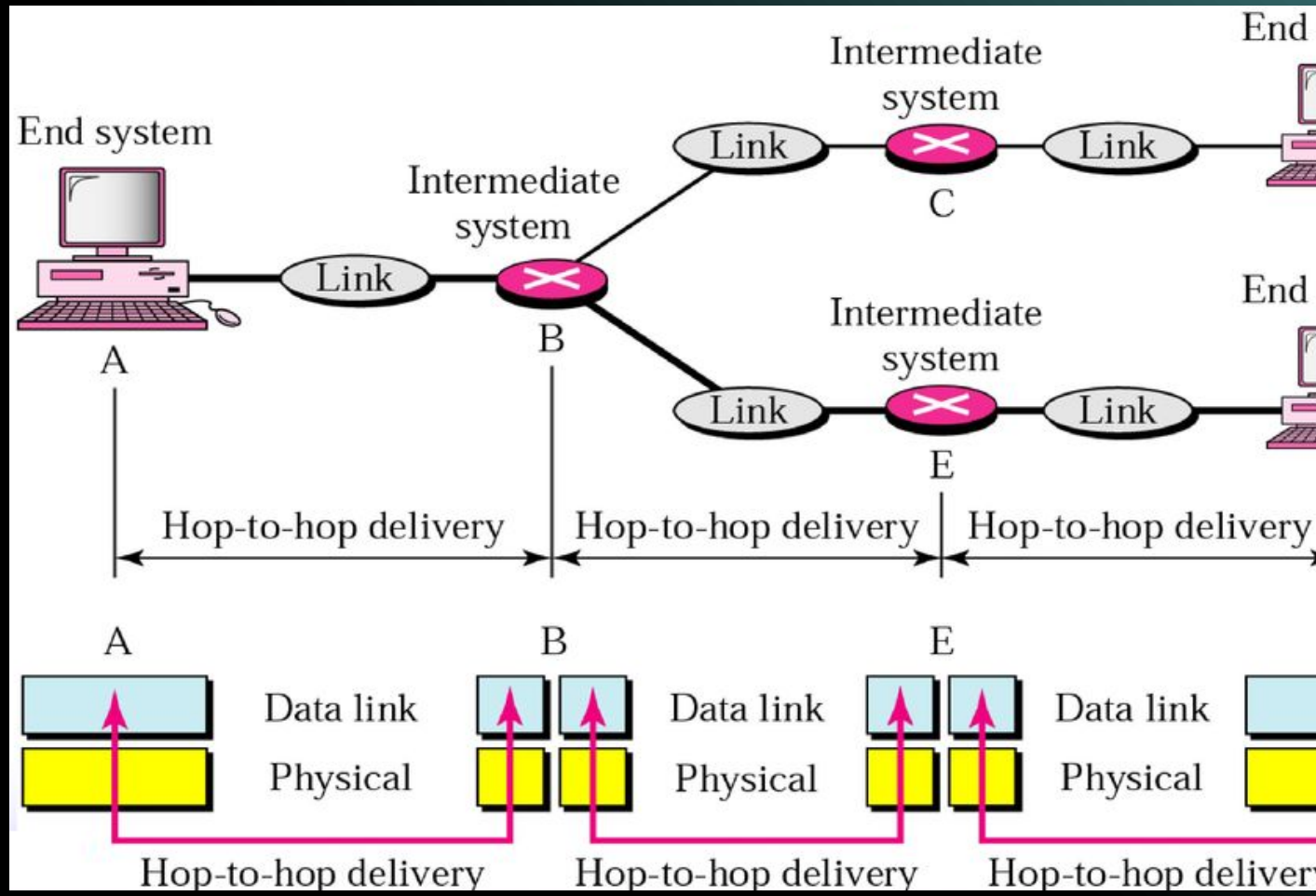
Computer Networks

Unit-3: Data Link Layer

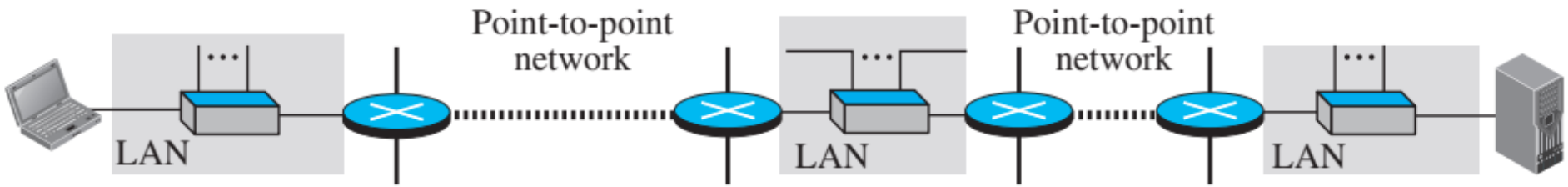


Day	Course Outcome	Intended Learning Outcomes (ILO)	Knowledge Level of ILO	No. of Hours	Pedagogy
1	CO 3	Describe The Data Link Layer - Services Provided to the Network Layer	K1	1	Lecture
2		Explain Data Link Layer Design Issues	K2	1	Lecture
3 & 4		Illustrate Framing – Error Control – Flow Control, Error Detection and Correction – Error-Correcting Codes – Error Detecting code	K3	2	Lecture with Discussion and in class Assignment
5 &6		Discuss MAC Protocols: ALOHA, CSMA	K2	2	Lecture with Discussion
7		Explain Elementary Data Link Protocols- A Utopian Simplex Protocol	K2	1	Lecture with Discussion
8		Discuss A Simplex Stop and Wait Protocol for an Error free channel-A Simplex Stop and Wait Protocol for a Noisy Channel.	K2	1	Lecture
9, 10, & 11		Demonstrate Sliding Window Protocols	K3	3	Lecture with Discussion
12 & 13		Discuss HDLC, PPP and Piggybacking	K2	2	Lecture

Data Link Layer (From CO-1)

Transform a raw transmission facility into a line that appears free of undetected transmission errors.

- DLL will achieve this by
  - having the sender break up the input data into **data frames** and transmitting them sequentially**.**
  - Make the receiving device send an acknowledgement in reliable services.
- Also make sure that a fast sender will not overrun a slow receiver.
- In case of broadcasting channels (multi-point), also decides, who can access and transmit (Medium Access Control).
- Uses physical (MAC) Address for delivery of information to next node (node-to-node delivery).

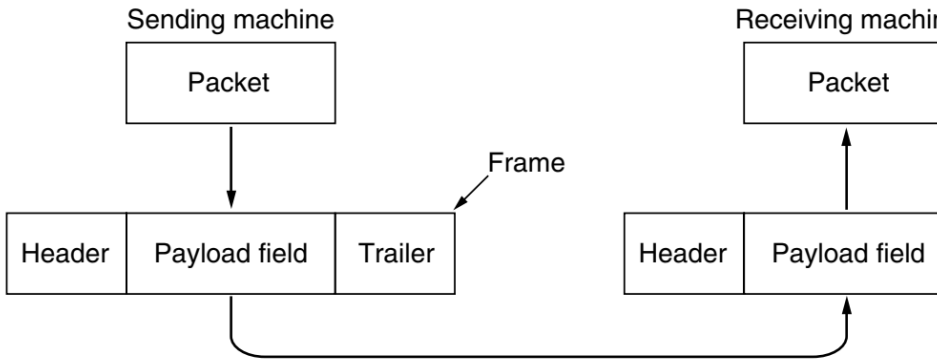


a. A small part of the Internet

Nodes and Links

DATA LINK LAYER DESIGN ISSUES

- The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:
  - 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.

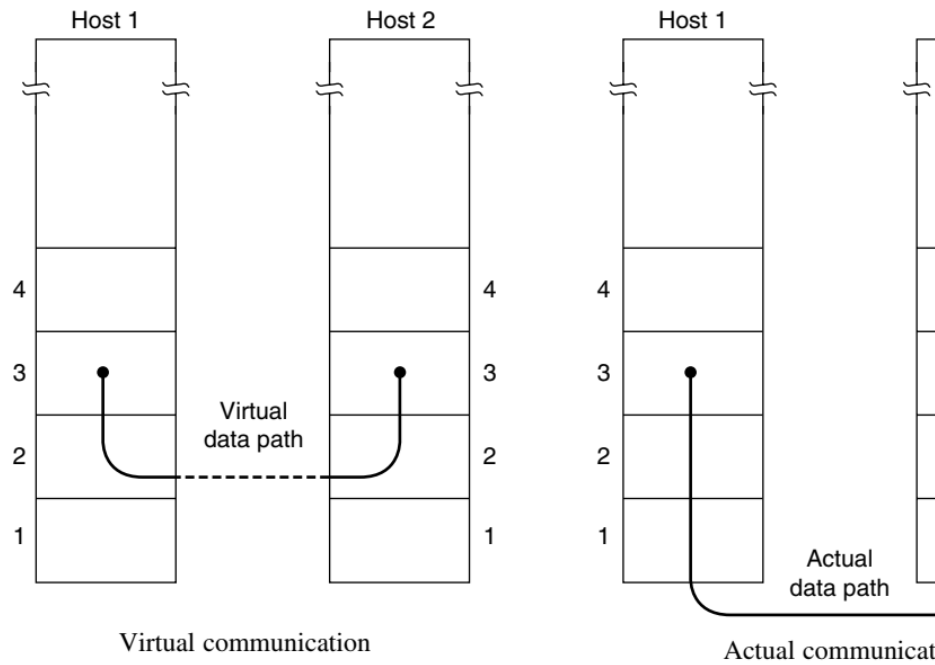


Providing a well-defined service interface to the network layer

- What are the services?
- How the services are achieved?

Services Provided to the Network Layer

- The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine.



Services Provided to the Network Layer (Cont.)

- Designed to offer various services.
- Three reasonable possibilities:
  - Unacknowledged connectionless service
  - Acknowledged connectionless service
  - Acknowledged connection-oriented service

Sublayers - Data Link Layer

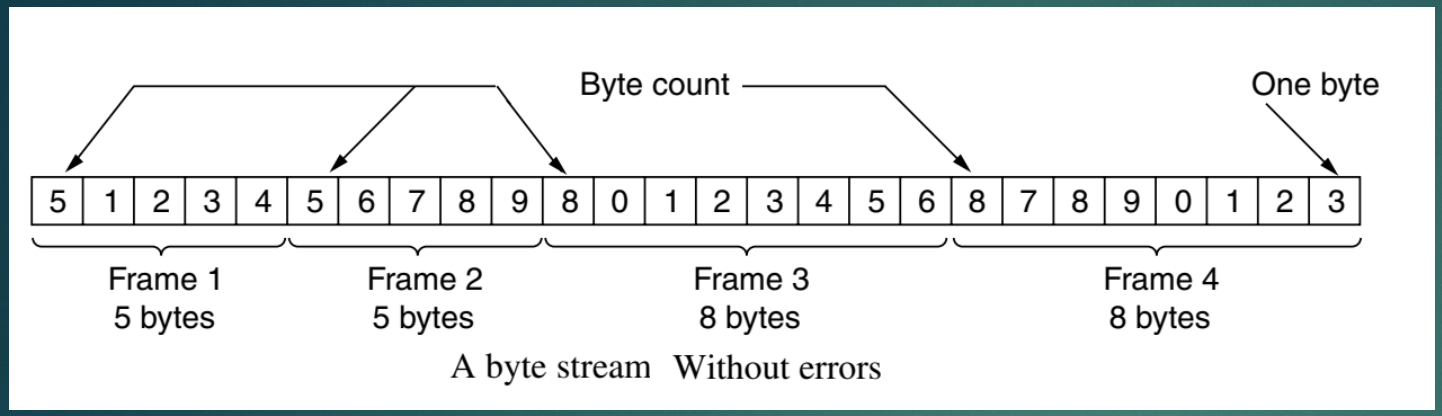
- Data Link Layer functionalities are divided into 2 groups with 2 sublayers taking care of each group:
  - Logical Link Control Sub-Layer / Data Link Control Sub-Layer
    - All the logical services
  - Medium Access Control Sub-Layer
    - All other services related to accessing medium during shared medium cases.

How the services are achieved?

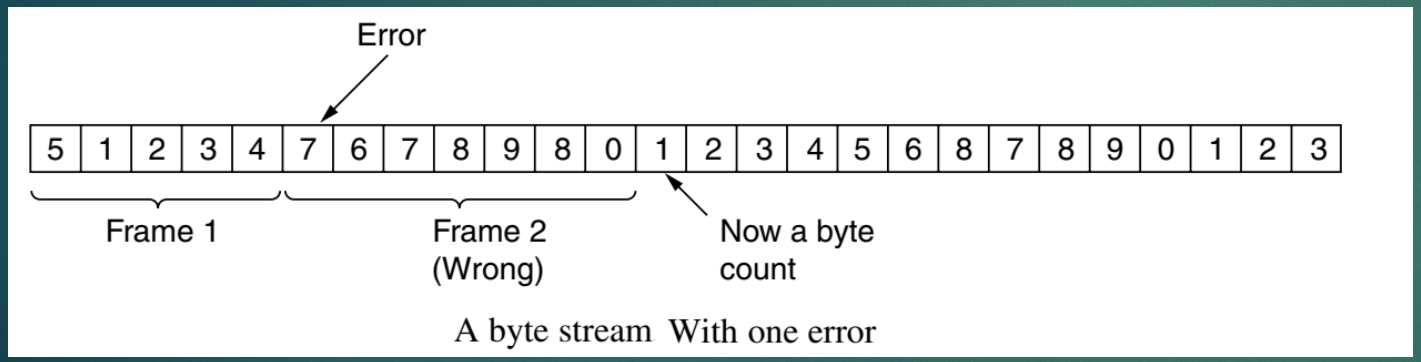
- Framing
  - Communication link is susceptible to errors.
  - Hence, “Break up the input data into **data frames** and transmitting them sequentially”, will suffice better communication.
  - Problem – Frame Boundaries
  - Solution
    - Fixed Length Framing : Not Efficient
    - Variable Length Framing : Very Efficient

Variable Length Framing

- Each Frame may be having different length
  - Byte Count
  - Flag bytes with byte Stuffing
  - Flag bits with bit stuffing
  - Physical Layer coding violation



A byte stream Without errors



A byte stream With one error

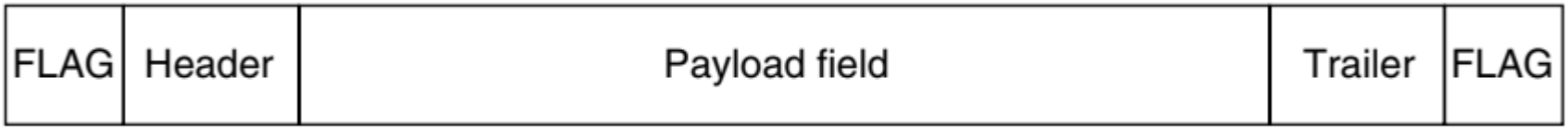
Byte Cou

ADDING NUMBER OF
IN EACH FRAME AT
BEGINNING OF FRA

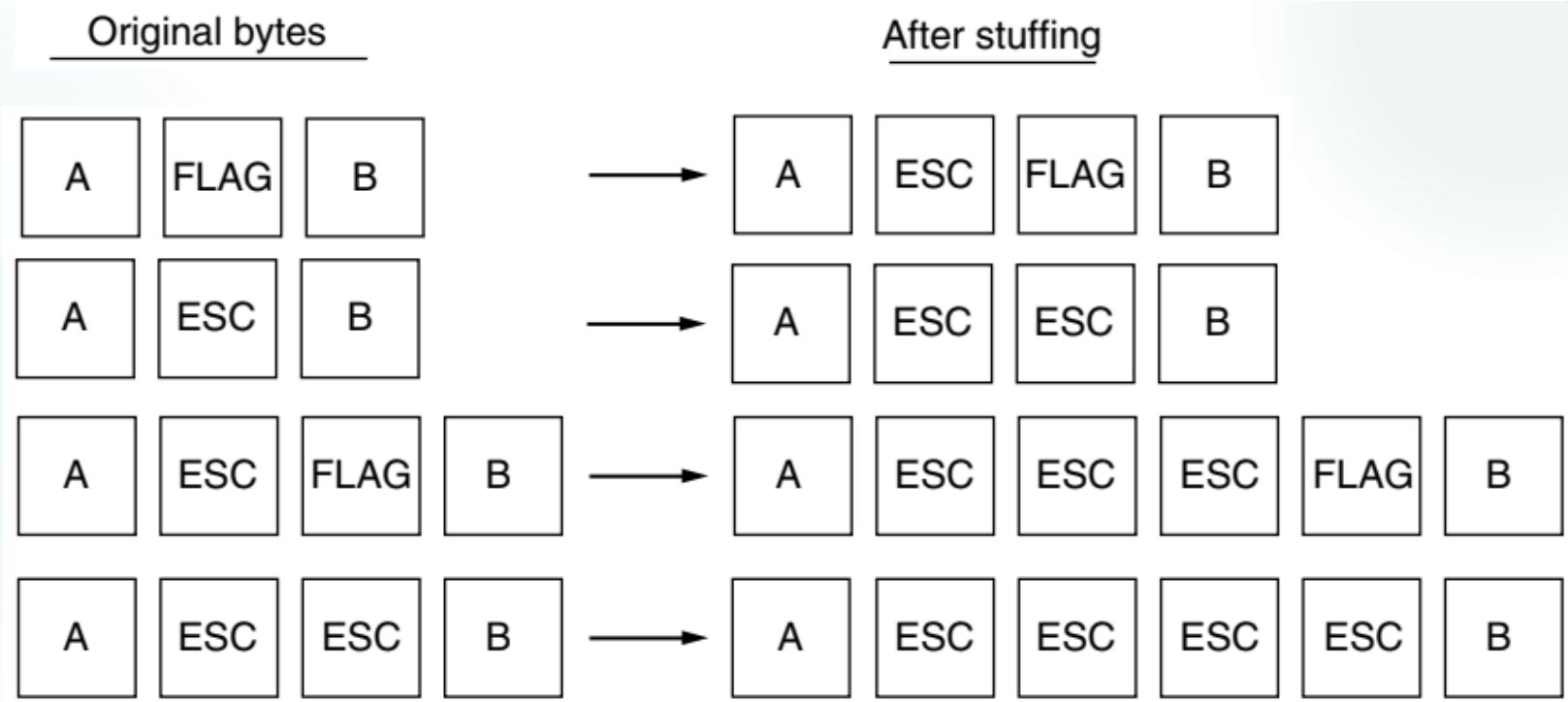
Byte oriented vs Bit Oriented Protocols

- A machine implemented with “Bit Oriented Protocols” process information bit by bit.
- A machine implemented with “Byte Oriented Protocols” process information byte by byte, i.e., once the system starts receiving first bit unless it receives remaining 7 bits, it cant proceed.
- DLL layer provides two framing methods based on whether the system process information bit by bit or byte by byte.

Byte Stuffing for Byte Oriented Systems



A frame delimited by flag bytes



Bit Stuffing for Bit Oriented Protocols

- The Flag pattern used is : “01111110”
- So, when ever this pattern appears in received stream means the frame end here and new frame begins.
- So to avoid getting this pattern in actual frame itself, the sender stuffs one 0 after every “011111” to avoid the flag pattern and the receiver do the opposite.

0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

The original data

0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0

Stuffed bits

The data as they appear on the li

0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

The data as thev are stored in the receiver’s memory after destuffing

Physical Layer coding violation

- Using a shortcut from the physical layer.
- When the physical layer uses encoding mechanism with redundant bits, using those redundant bits for representing beginning or ending of a frame.

How the services are achieved?

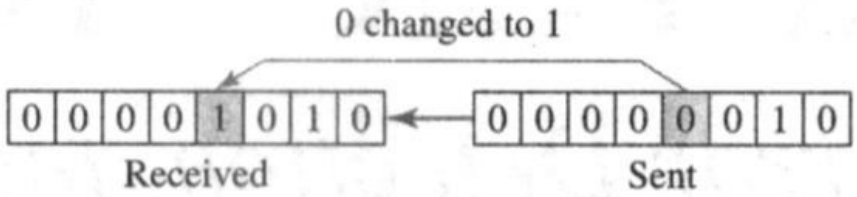
- Error Control
  - Detecting and if possible correcting errors.
- With just the actual information we will not be able tell whether received data is right or wrong.
- Hence, we will send additional bits that will help to do this task.
- “Redundant Bits” will be generated by encoding actual data bits.
- Number of redundant bits will decide how efficient a encoding mechanism is in terms of error control.

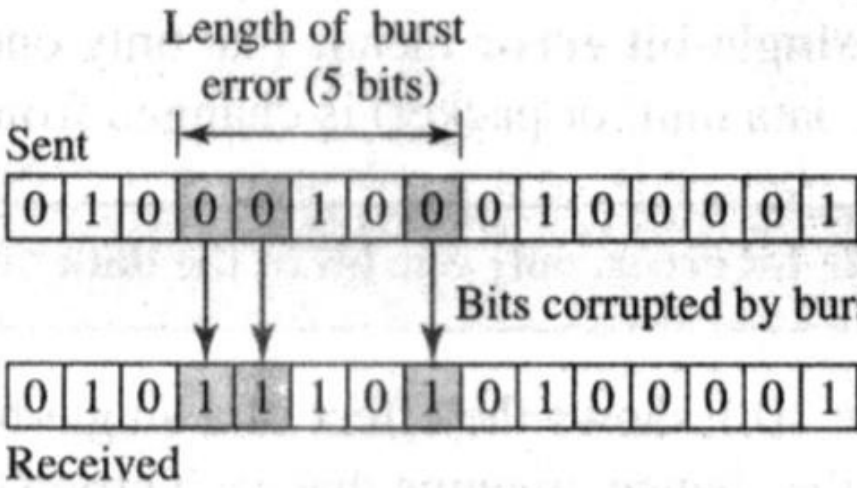
Error Control

- Some channels (Fibers) less prone to errors than other channels(Wireless).
- Two types of Error Controlling
  - Error correction (Forward Error Correction) : For more noisy channels
  - Error detection : For less noisy channels
- Error types
  - Single bit errors
  - Burst errors

Types Of Errors

- Single-Bit Error
- Burst Error





Block Codes

- So, How the receiver will know that he received a wrong data?
- We divide our message into blocks, each of m bits, each block is called a data words.
- For example:
  - The received data is divided into blocks of 4 bits.
  - The sender sent 0011 and due to a single bit error at position 2, the receiver received 0001.
  - Can the receiver can guess that it is wrong? (Yes/No)

Dataword	*Datawor*
0000	1000
0001	1001
0010	1010
0011	1011
0100	1100
0101	1101
0110	1110
0111	1111

Block Codes

- Solution is adding redundant bits and creating valid and invalid words.
- m= no of message bits, i.e., Number of bits in data word.
- r= no of redundant bits (deducted by applying some function o word)
- The resulting output word of size n= m + r is called codeword.
- For example:
  - Now we had added one redundant bit to each data word creating va codewords and invalid codewords.
  - So the user sent the codeword 00110 of data word 0011.
  - Receiver got, 00010 with a single bit error at 2nd position.
  - Can he guess that the received codeword is wrong? (Yes/No)

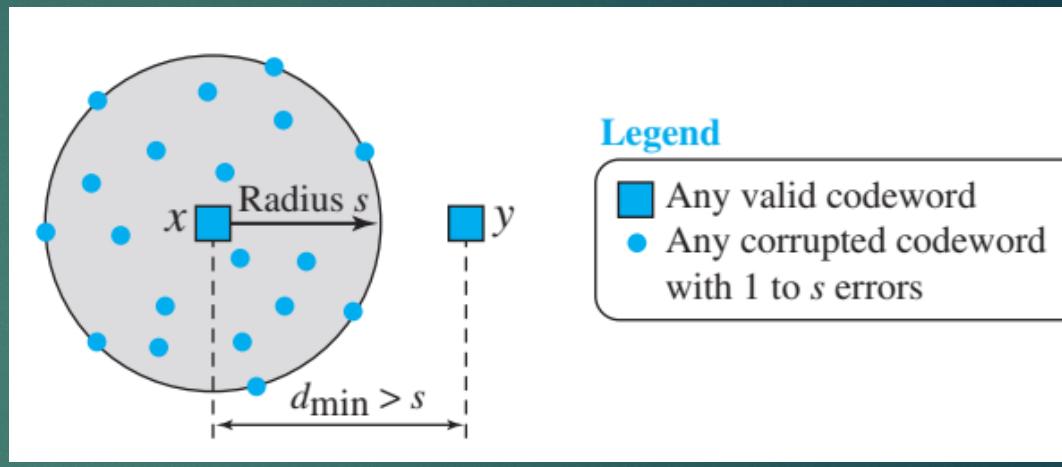
Dataword	*Codeword*	*Dataword*	*Codeword*
0000	**00000**	1000	**10001**
0001	**00011**	1001	**10010**
0010	**00101**	1010	**10100**
0011	**00110**	1011	**10111**
0100	**01001**	1100	**11000**
0101	**01010**	1101	**11011**
0110	**01100**	1110	**11101**
0111	**01111**	1111	**11110**

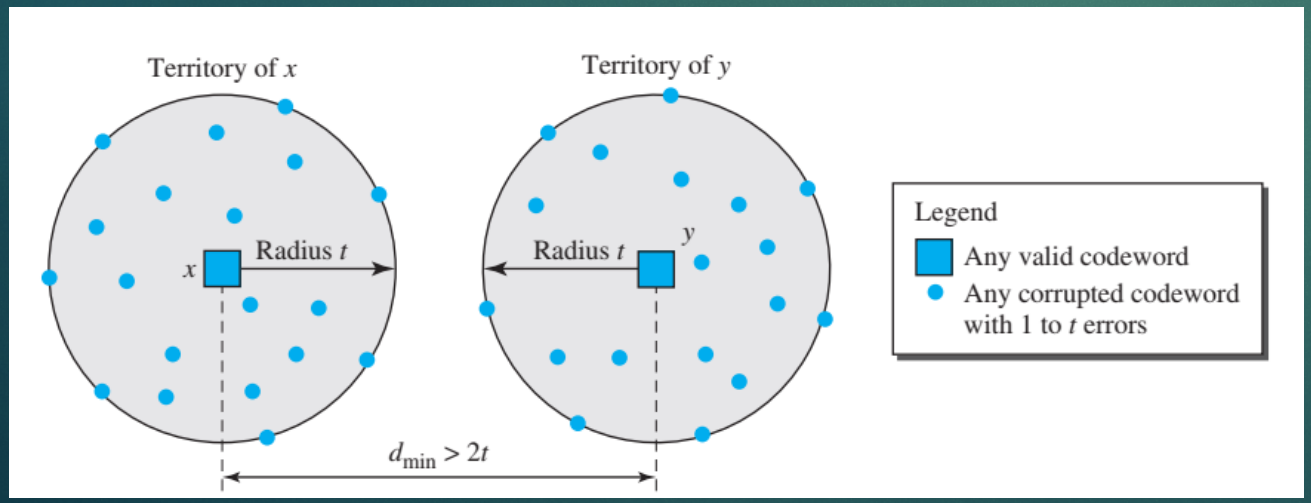
Block Codes

- How many redundant bits to be added?
- Answer: Hamming Distance
- Hamming Distance
  - between two words (of the same size) is the number of differences between the corresponding bits.
  - example d(01011, 01101) = 2
  - Minimum Hamming Distance of a encoding mechanism is minimum of all the hamming distances calculated between all valid codewords.

Minimum Hamming Distance & Error Control

- To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.





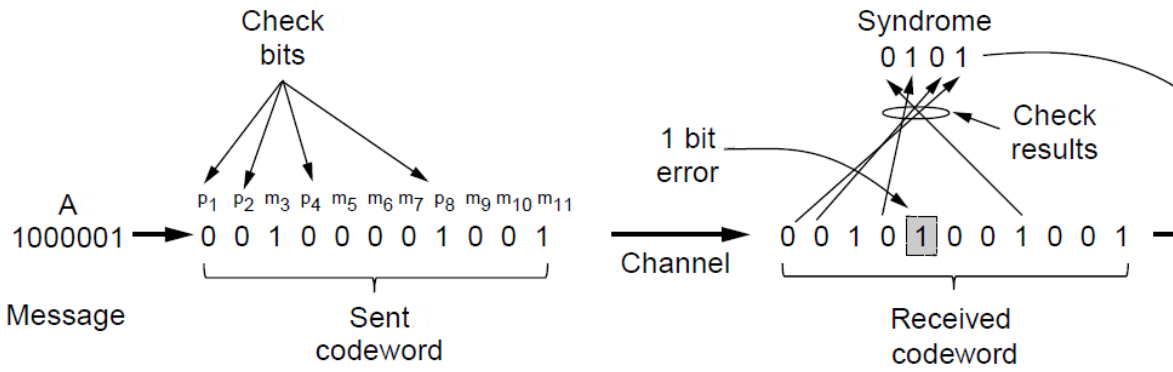
To guarantee the correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

Error correction

- EC Codes
  - **Hamming codes**.
  - Binary convolutional codes.
  - Reed-Solomon codes.
  - Low-Density Parity Check codes.

Hamming code

- To correct single bit error, the number of redundant bits c calculated using the following formula: $2^r \geq m + r + 1$
- In hamming codes the bits in code word is numbered consecutively from 1 to n from left to right or right to left.
- The bits at powers of 2 ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$,..) are check redundant bits.
- The remaining (3,5,6,7,9…) are message bits.
- A (11,7) hamming code looks, as shown bellow.
- Here hamming distance is 3.



Hamming code (Cont..)

- Each check bit forces the modulo 2 sum, or parity, of some collection of bits, including itse be even (or odd).
- R1, i.e., P1 bit is calculated finding parity of all the bits at positions whose bi representation includes a 1 in the least significant position (1st Position in bit pattern) .P1 1, 3, 5, 7, 9, 11.

Binary Positions	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	101
Decimal Positions	1	2	3	4	5	6	7	8	9	10	11
Bits Representation	**R1 (P1)**	**R2 (P2)**	**M1**	**R3 (P4)**	**M2**	**M3**	**M4**	**R4 (P8)**	**M5**	**M6**	**M7**
Message Bits			**1**		**0**	**0**	**1**		**1**	**0**	**1**
R1 calculation=0	**R1**		**1**		**0**		**1**		**1**		**1**
R2 calculation=1		**R2**	**1**			**0**	**1**			**0**	**1**
R3 calculation=1				**R3**	**0**	**0**	**1**				
R4 calculation=0								**R4**	**1**	**0**	**1**
Generated Code Word	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**

Hamming code (Cont..)

- In the same manner R2 (P2), R3 (P4), R4 (P8) is calculated and placed in the corresponding position.

- When a codeword arrives, the receiver redoes the check bit computations including the values of the received check bits.
- If the check bits are correct then, for even parity sums, each check result should be zero. In this case the codeword is accepted as valid.
- If the check results are not all zero, however, an error has been detected. The set of check results forms the "**error syndrome (s)**" that is used to pinpoint and correct the error.

Hamming Code(Cont.)

- Suppose in the above example, the received code word is not corrupted then.
- S=S4,S3,S2,S1=0000=0

Decimal Positions	1	2	3	4	5	6	7	8	9	10
Tx Code Word	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**
Rx Code Word	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**
S1 calculation=0	**0**		**1**		**0**		**1**		**1**	
S2 calculation=0		**1**	**1**			**0**	**1**			**0**
S3 calculation=0				**1**	**0**	**0**	**1**			
S4 calculation=0								**0**	**1**	**0**

Hamming Code(Cont..)

- Suppose in the above example the 3rd bit is changed from 1 to 0 during data transmission, then it gives new parit
the binary number.
- S=S4,S3,S2,S1=0011=3
- The bits give the binary number as 0011 whose decimal representation is 3. Thus, the bit 3 contains an error. To
error the 3rd bit is changed from 0 to 1.

Decimal Positions	1	2	3	4	5	6	7	8	9	10
Tx Code Word	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**
Rx Code Word	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**
S1 calculation=1	**0**		**0**		**0**		**1**		**1**	
S2 calculation=1		**1**	**0**			**0**	**1**			**0**
S3 calculation=0				**1**	**0**	**0**	**1**			
S4 calculation=0								**0**	**1**	**0**

Hamming Code(Cont..)

- Suppose in the above example the 6th bit is changed from 0 to 1 during data transmission, then it gives new parit
the binary number.
- S=S4,S3,S2,S1=0110=6
- The bits give the binary number as 0110 whose decimal representation is 6. Thus, the bit 6 contains an error. To
error the 6th bit is changed from 1 to 0.

Decimal Positions	1	2	3	4	5	6	7	8	9	10
Tx Code Word	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**
Rx Code Word	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**
S1 calculation=0	**0**		**1**		**0**		**1**		**1**	
S2 calculation=1		**1**	**1**			**1**	**1**			**0**
S3 calculation=1				**1**	**0**	**1**	**1**			
S4 calculation=0								**0**	**1**	**0**

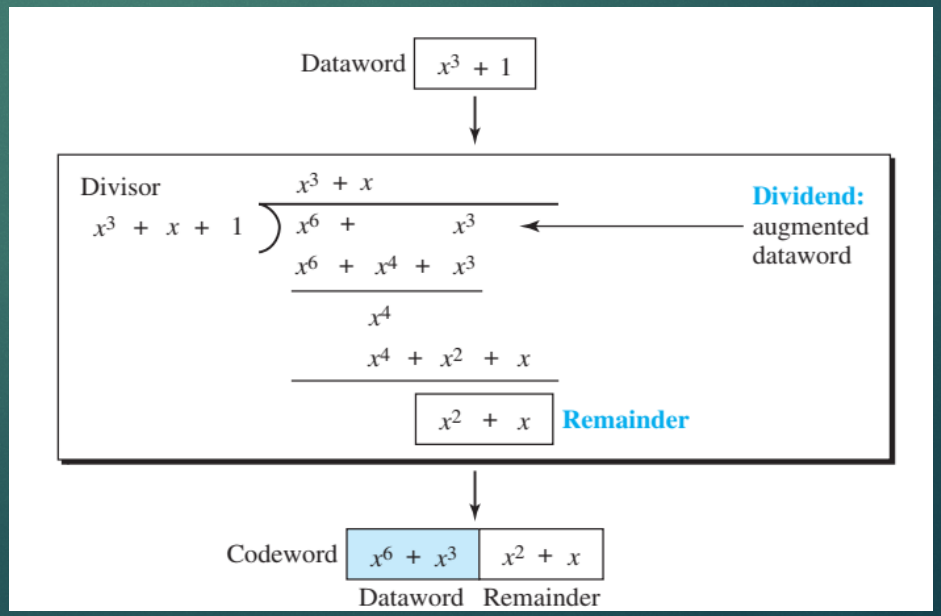
Hamming Code(Cont..)

- Suppose in the above example the 5th and 6th bit is changed from 0 to 1 during data transmission, then it parity values in the binary number.
- S=S4,S3,S2,S1=0011=3
- The bits give the binary number as 0011 whose decimal representation is 3. Thus, the bit 3 contains a correct the error the 3rd bit is changed from 0 to 1.
- Hence here we can see that the designed hamming code can only correct only single bit errors.

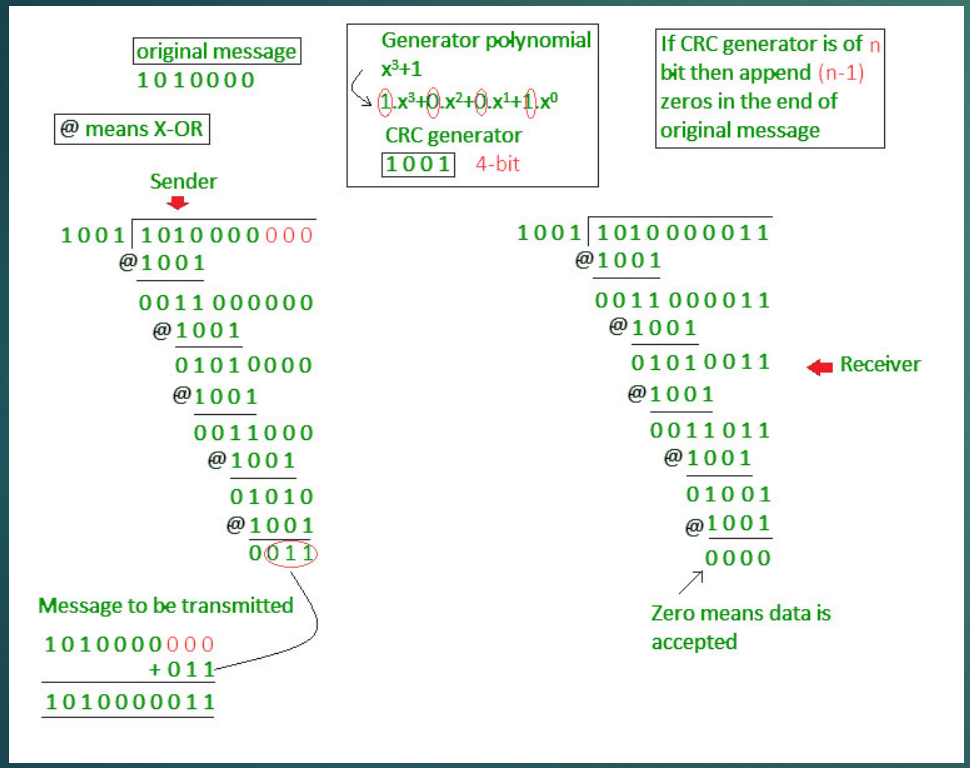
Decimal Positions	1	2	3	4	5	6	7	8	9	10
Tx Code Word	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**
Rx Code Word	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
S1 calculation=1	**0**		**1**		**1**		**1**		**1**	
S2 calculation=1		**1**	**1**			**1**	**1**			**0**
S3 calculation=0				**1**	**1**	**1**	**1**			
S4 calculation=0								**0**	**1**	**0**

CRC – Polynomial code

- Based on treating bit strings as representations of coefficients of polynomial.
- 11001 is representing $1 * x^4 + 1 * x^3 + 0 * x^2 + 0 * x^1 + 1 * x^0$
- Generator polynomial G(x), Message Polynomial M(x), Transmitted Polynomial T(x)=CRC(G,M)



CRC



- Let r be the degree of $G(x)$. Append r zero bits to the low-order end of the message frame $M(x)$ to get $x^r M(x)$.
- Divide the bit string corresponding to $G(x)$ into the bit string corresponding to $x^r M(x)$, using Mod 2 division.
- Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction.

CRC

- The first term at the right-hand side of the equality has a remainder of zero.
- So the syndrome is actually the remainder of the second term on the right-hand side.
- So, syndrome=0 means either $e(x) = 0$, i.e., no error or $e(x)$ is divisible by $g(x)$. So such errors are not caught.

$$\text{Received codeword} = c(x) + e(x)$$

$$\frac{\text{Received codeword}}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

A good polynomial generator needs to have the following char

1. **It should have at least two terms.**
2. **The coefficient of the term x^0 should be 1.**
3. **It should not divide $x^t + 1$, for t between 2 and $n - 1$.**
4. **It should have the factor $x + 1$.**

CRC

- Standard CRC Polynomials are:

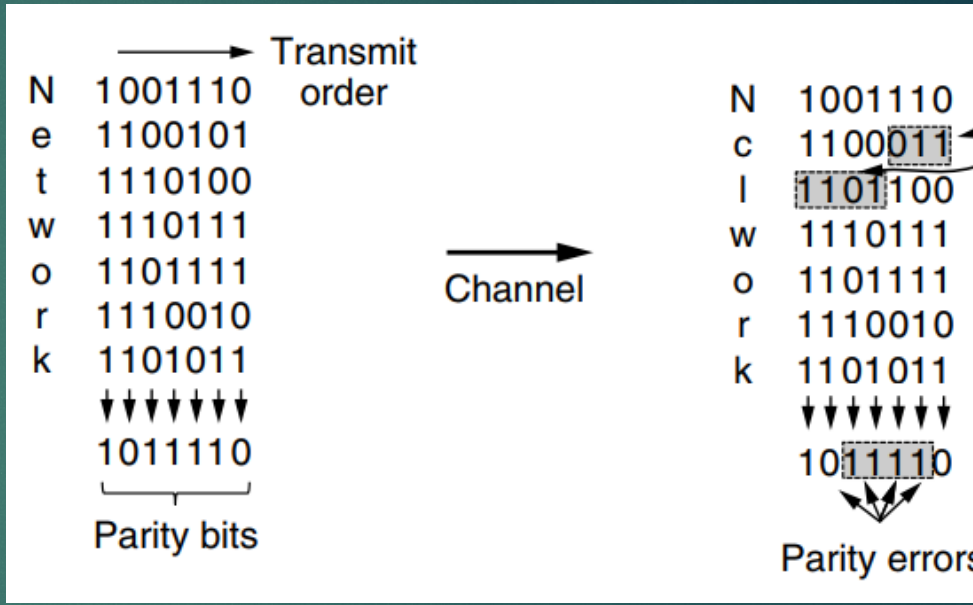
Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ **100000111**	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ **11000110101**	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ **10001000000100001**	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **100000100110000010001110110110111**	LANs

Parity checking

- Min hamming distance is 2.
- Suitable for detecting single bit errors, but can detect odd number of errors.
- Even parity: Number of 1's even – parity bit "0"
- Odd parity: Number of 1's odd – parity bit "0"
- Interleaving will allow parity checking to detect burst errors.

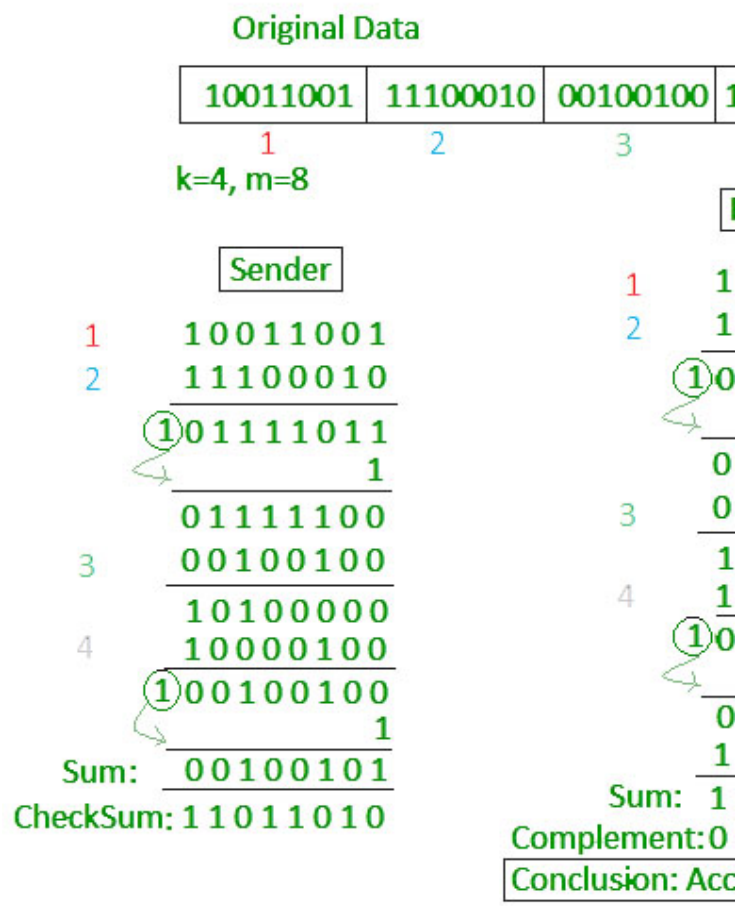
Parity Checking

- To find a burst error we can use a simple parity checking process as shown in figure.

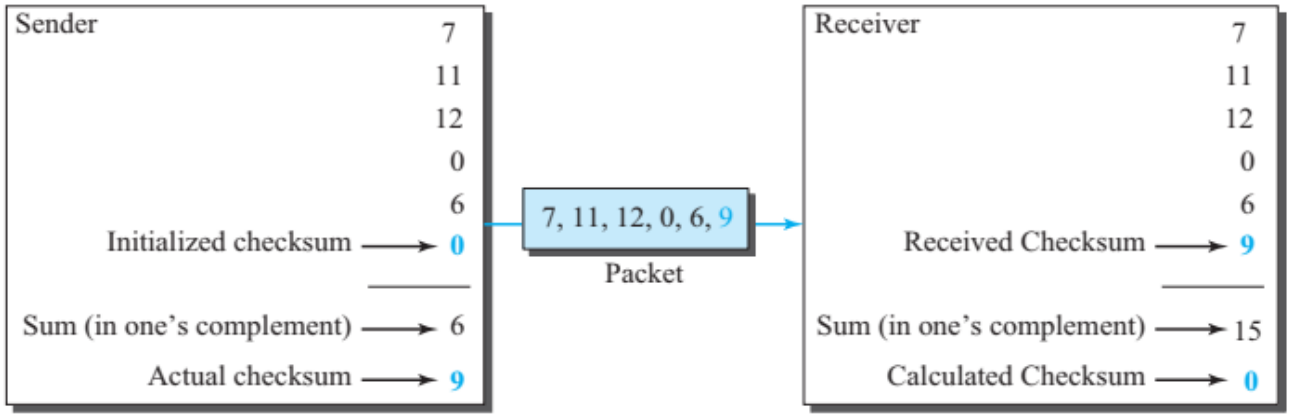


Checksum : 1's Complement

- Related to group of parity bits or group of check bits associated with message.
- Multiple message blocks are added together to get the checksum and the 1's complement of the checksum is transmitted to receiver.
- The receiver will calculate the checksum for total received data (message + checksum) if the result is 0 then no error otherwise there is an error.



Checksum 1's Compleme

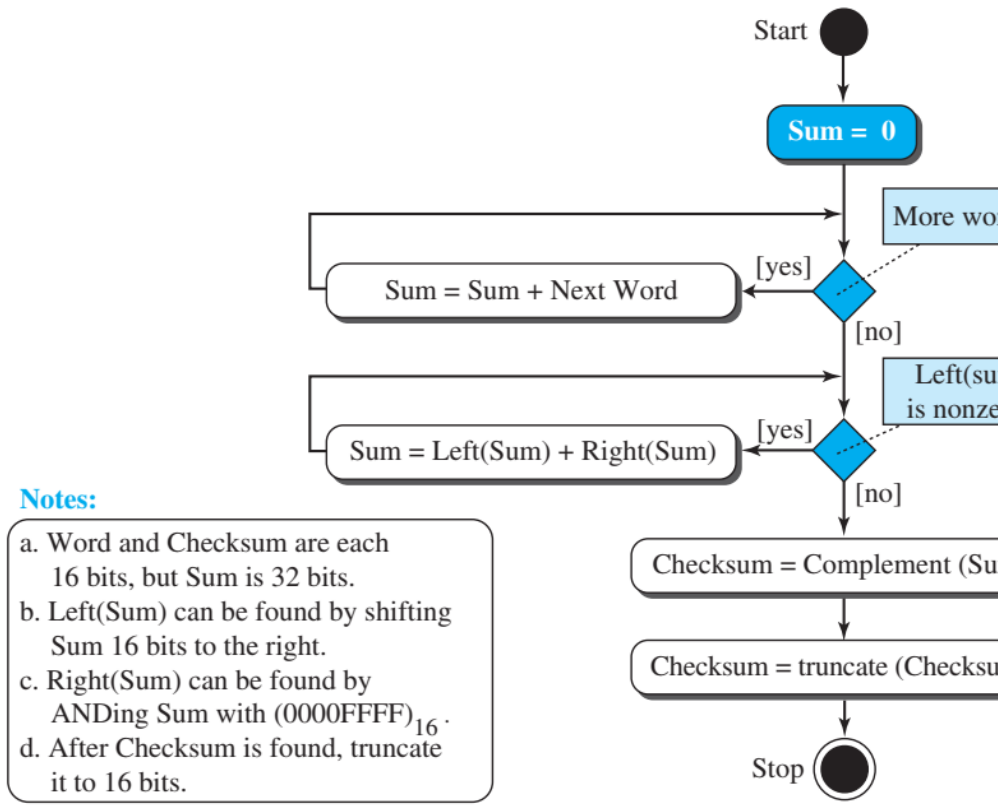


Internet Checksum

- Traditionally, the Internet has used a 16-bit checksum. The sender and the receiver follow the steps depicted in Table, the sender or the receiver uses five steps.

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checks
2. The value of the checksum word is initially set to zero.	2. The message is divided into
3. All words including the checksum are added using one's complement addition.	3. All words are added using c ment addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented a new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is accepted; otherwise, it is r

Internet Checksum

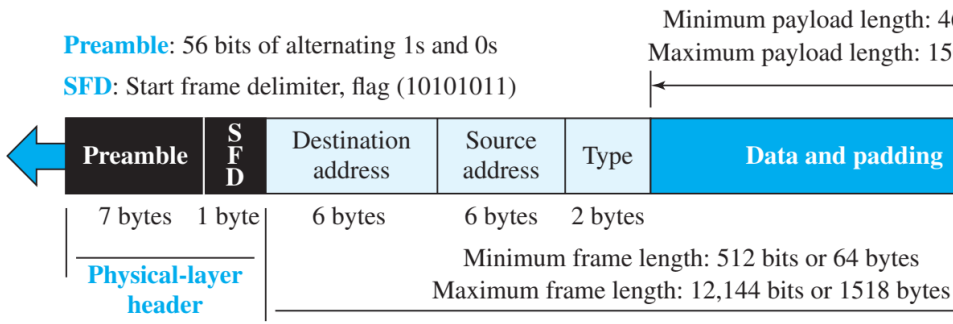


Notes:

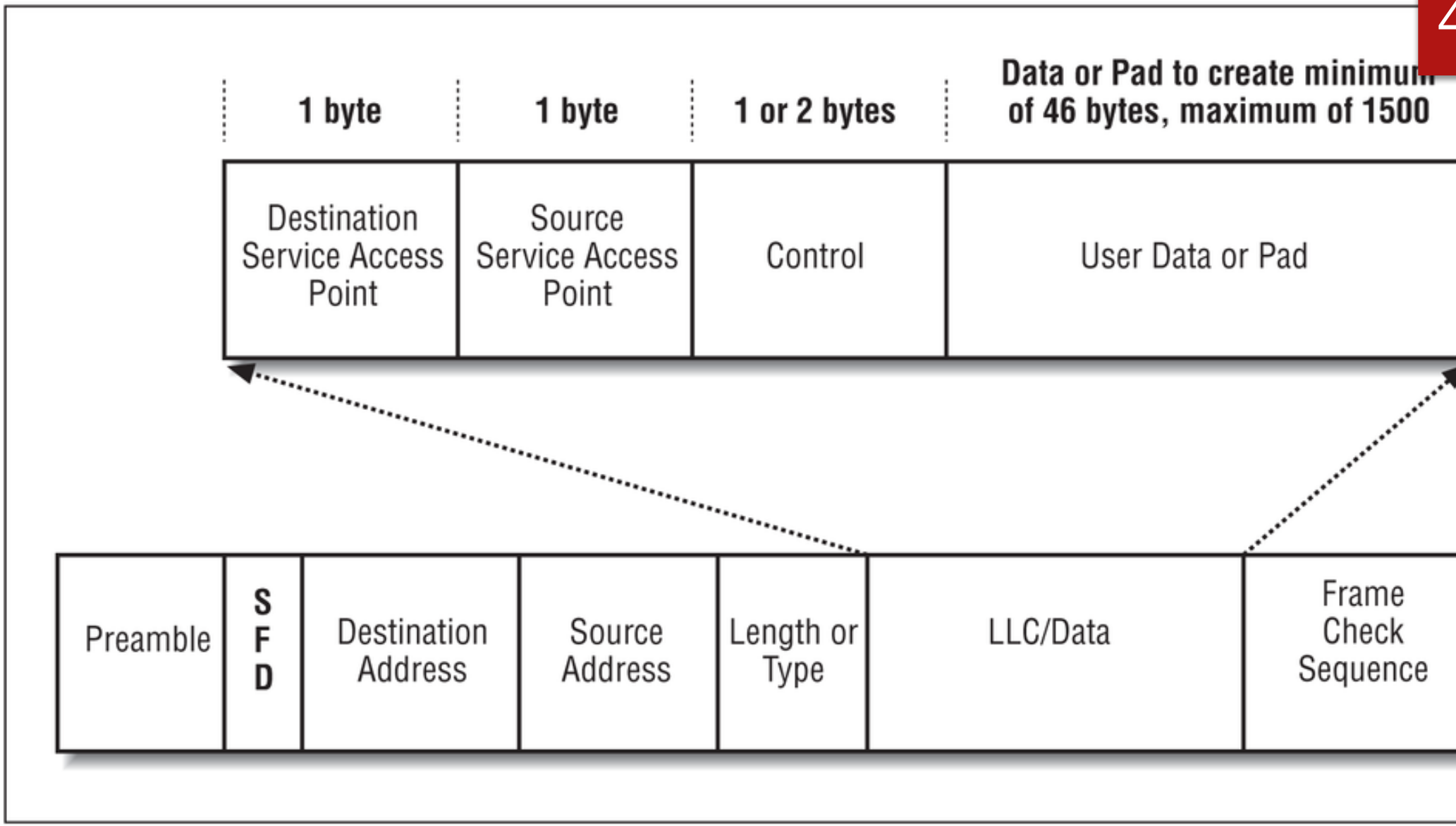
a. Word and Checksum are each 16 bits, but Sum is 32 bits.
b. Left(Sum) can be found by shifting Sum 16 bits to the right.
c. Right(Sum) can be found by ANDing Sum with $(0000FFFF)_{16}$.
d. After Checksum is found, truncate it to 16 bits.

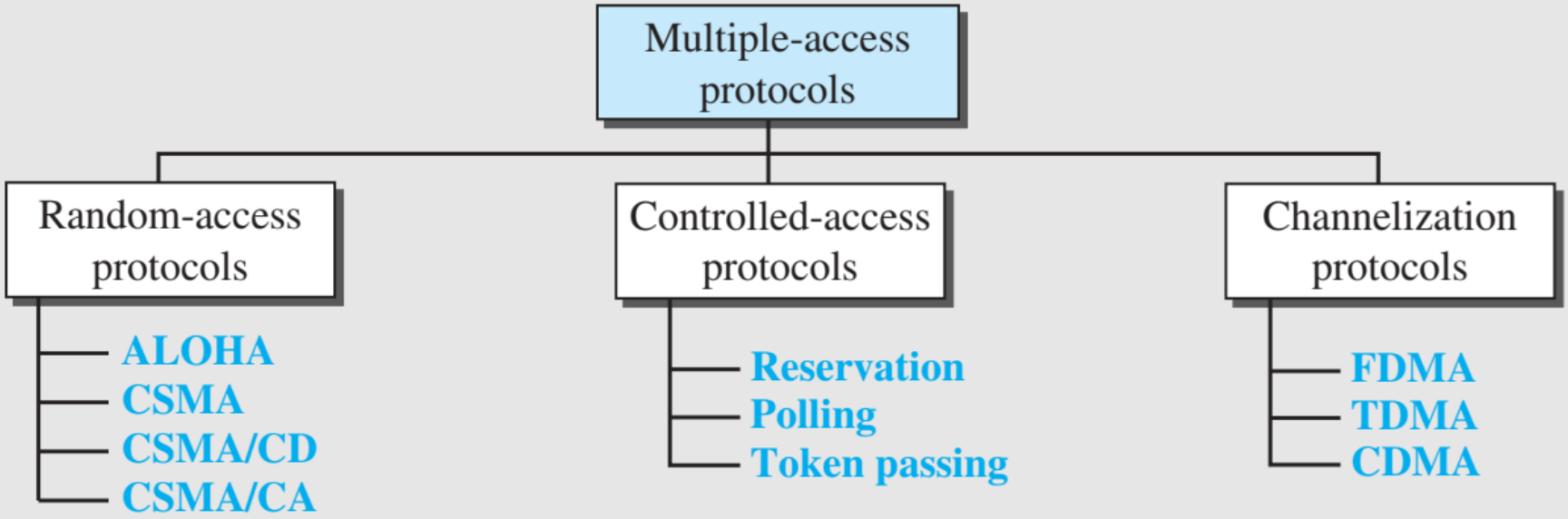
How the services are achieved?

- Flow Control
  - Regulating the communication between faster sender and slower receiver.
  - Will be discussed later in MAC- Elimentary Data Link Control.



Ethernet Frame Structure (Just For Under


1 byte
1 byte
1 or 2 bytes
Data or Pad to create minimum of 46 bytes, maximum of 1500
Destination Service Access Point
Source Service Access Point
Control
User Data or Pad
Preamble
SFD
Destination Address
Source Address
Length or Type
LLC/Data
Frame Check Sequence



Medium Access Control

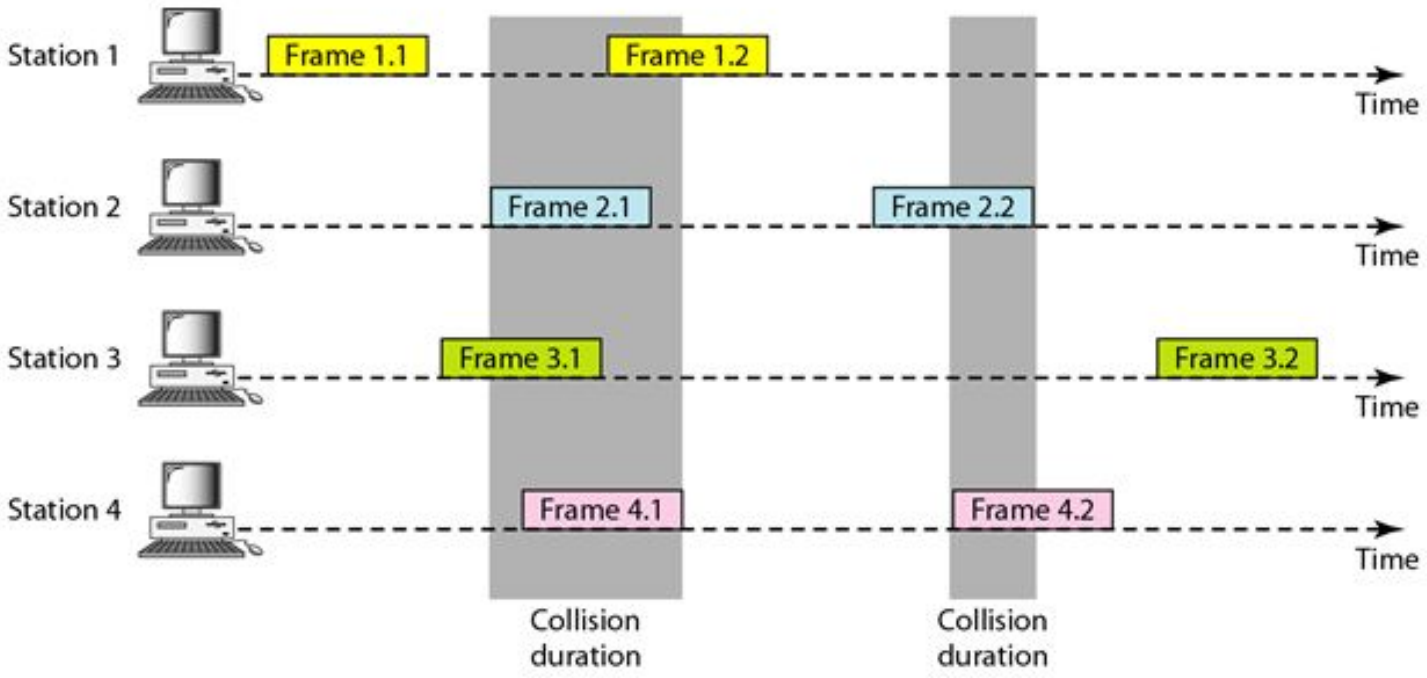
Random Access Protocols

- In random-access or contention methods, no station is superior to another station and none is assigned control over another. At each instance, a station that has data to send uses a procedure defined by the protocol to make a decision on whether or not to send.
- Two features give this method its name.
  - First, there is no scheduled time for a station to transmit. Transmission is random among the stations. That is why these methods are called random access.
  - Second, no rules specify which station should send next. Stations compete with one another to access the medium. That is why these methods are also called contention methods.

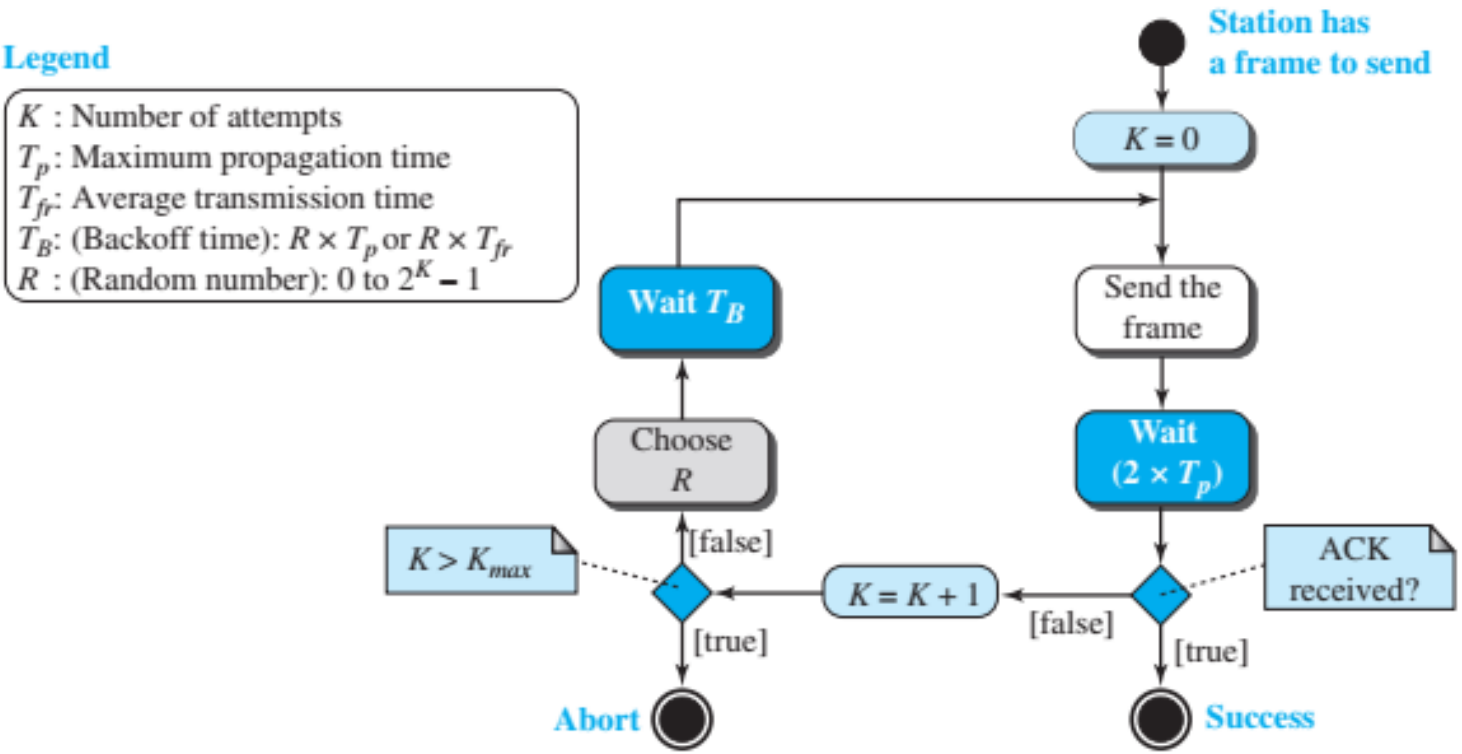
ALOHA

- ALOHA, the earliest random-access method, was developed at the University of Hawaii in early 1970. It was designed for a radio (wireless) LAN, but it can be used on any shared medium.
- Divided into two types based on whether time is considered as continuous factor discrete:
  - Pure ALOHA
  - Slotted ALOHA

Pure ALOHA

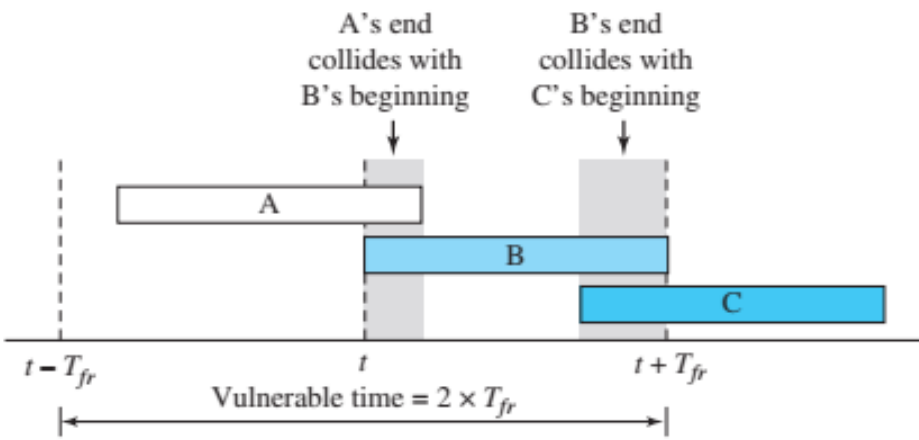

Station 1
Frame 1.1
Frame 1.2
Time
Station 2
Frame 2.1
Frame 2.2
Time
Station 3
Frame 3.1
Frame 3.2
Time
Station 4
Frame 4.1
Frame 4.2
Time
Collision duration
Collision duration

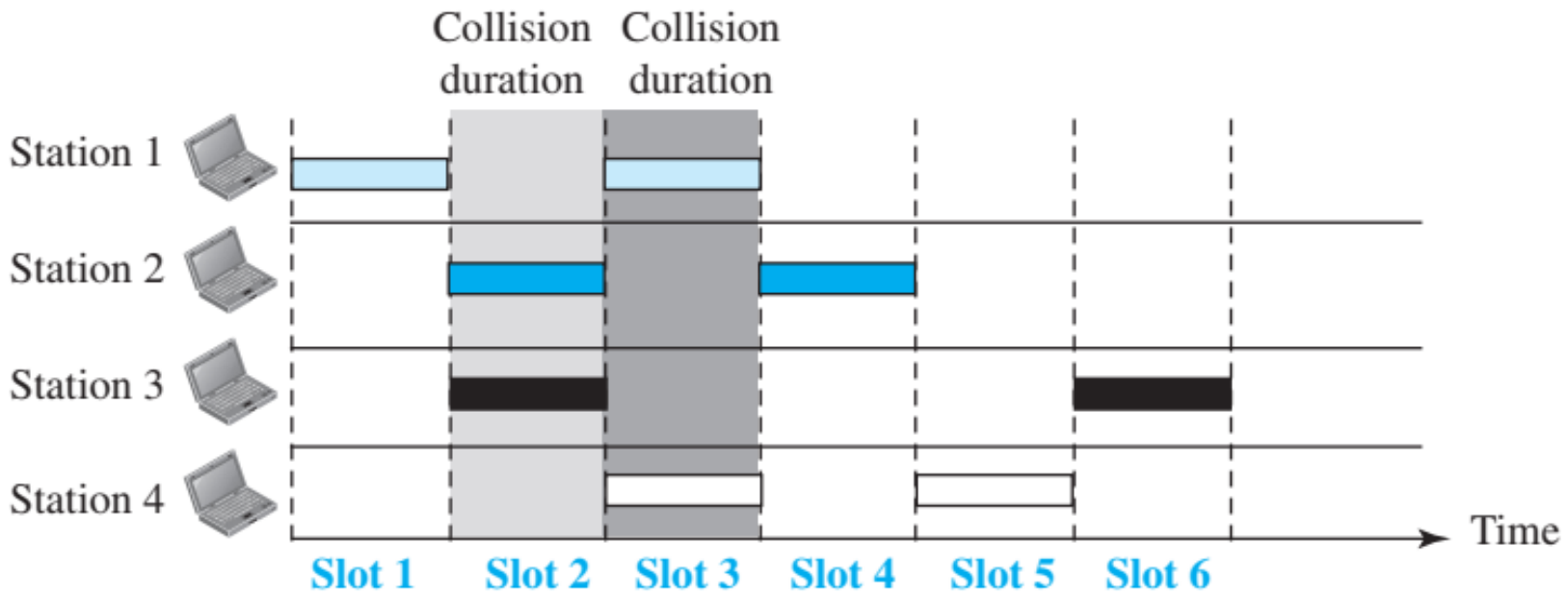
Pure ALOHA


Legend
K : Number of attempts
Tp: Maximum propagation time
Tfr: Average transmission time
TB: (Backoff time): R × Tp or R × Tfr
R : (Random number): 0 to 2^K − 1
Station has a frame to send
K = 0
Send the frame
Wait (2 × Tp)
ACK received?
[false]
[true]
K = K + 1
Wait TB
Choose R
K > Kmax
[false]
[true]
Abort
Success

Pure ALOHA

- Vulnerable period for the frame= $2T_{fr}$
- The throughput for pure ALOHA is $S = G * e^{-2G}$, where G is the average number of frames generated and transmitted by a system in T_{fr} sec.
- The maximum throughput $S_{max} = {}^{1}/_{2e} = 0.184$, when $G = 1/2$.

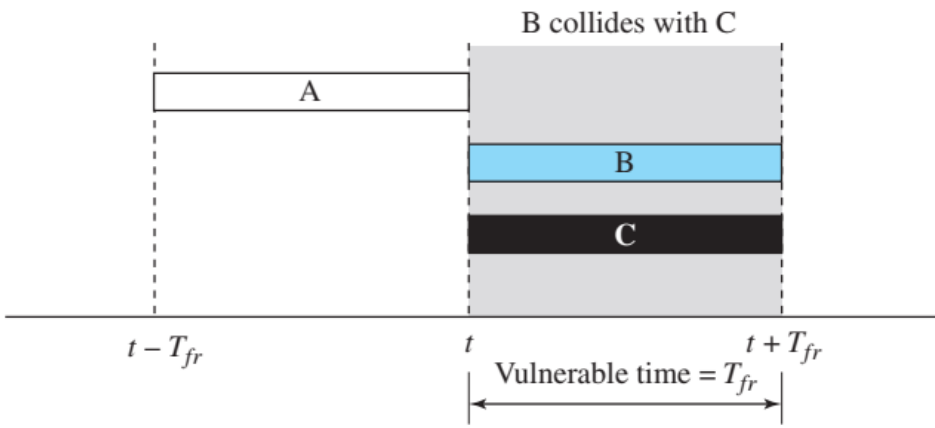




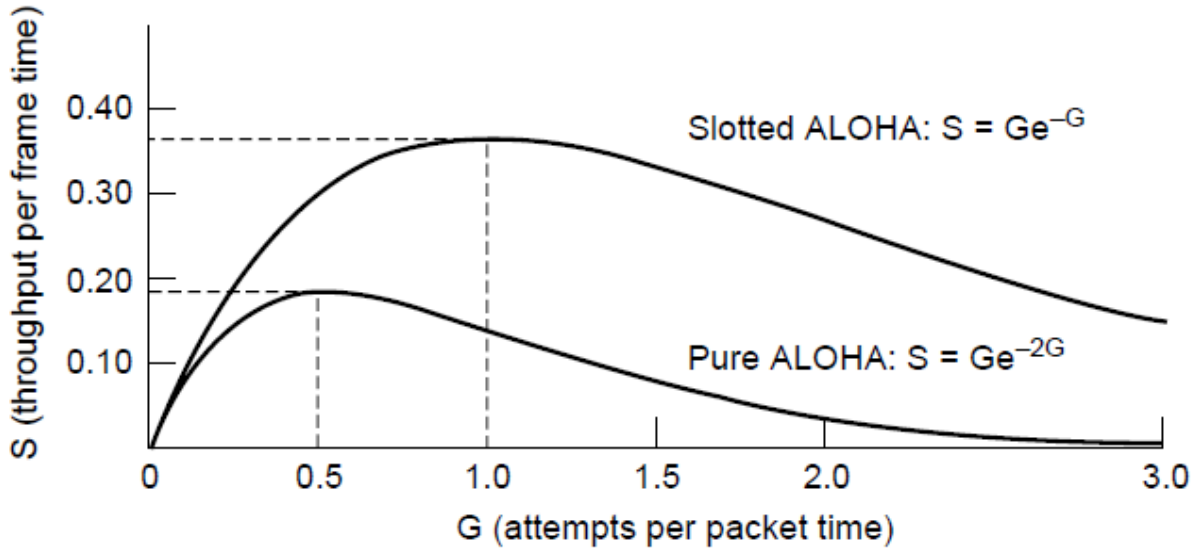
Slotted ALOHA

Slotted ALOHA

- Vulnerable period for the frame= T_{fr}
- The throughput for pure ALOHA is $S = G * e^{-G}$.
- The maximum throughput $S_{max} = {}^{1}/_{e} = 0.368$, when $G = 1$.



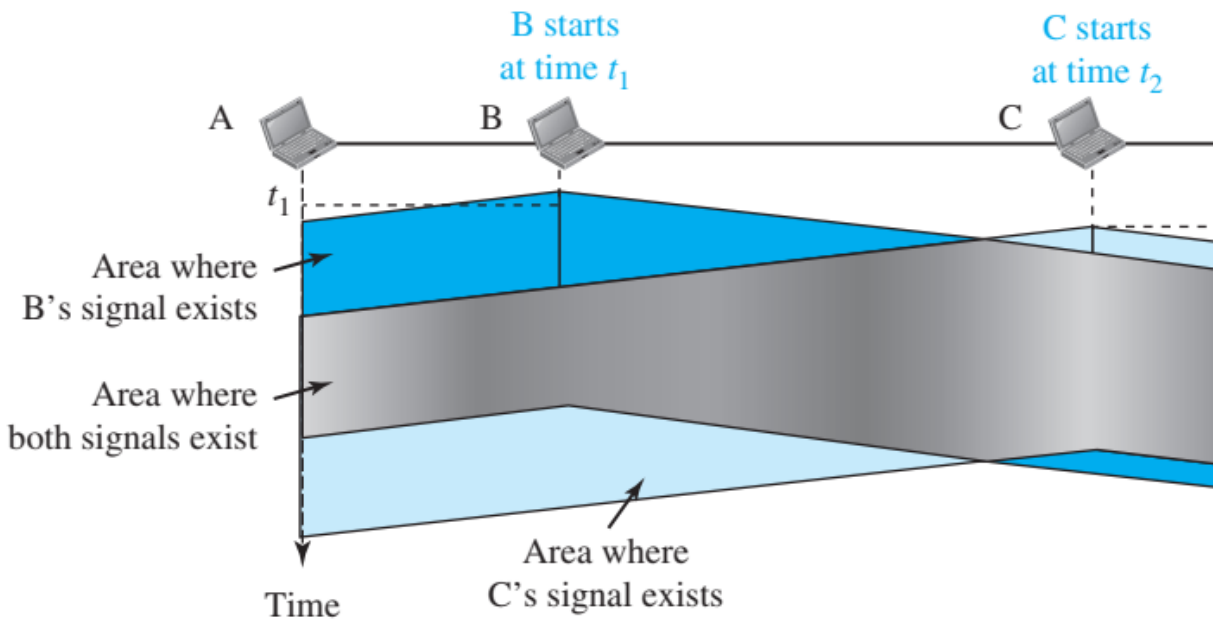
ALOHA

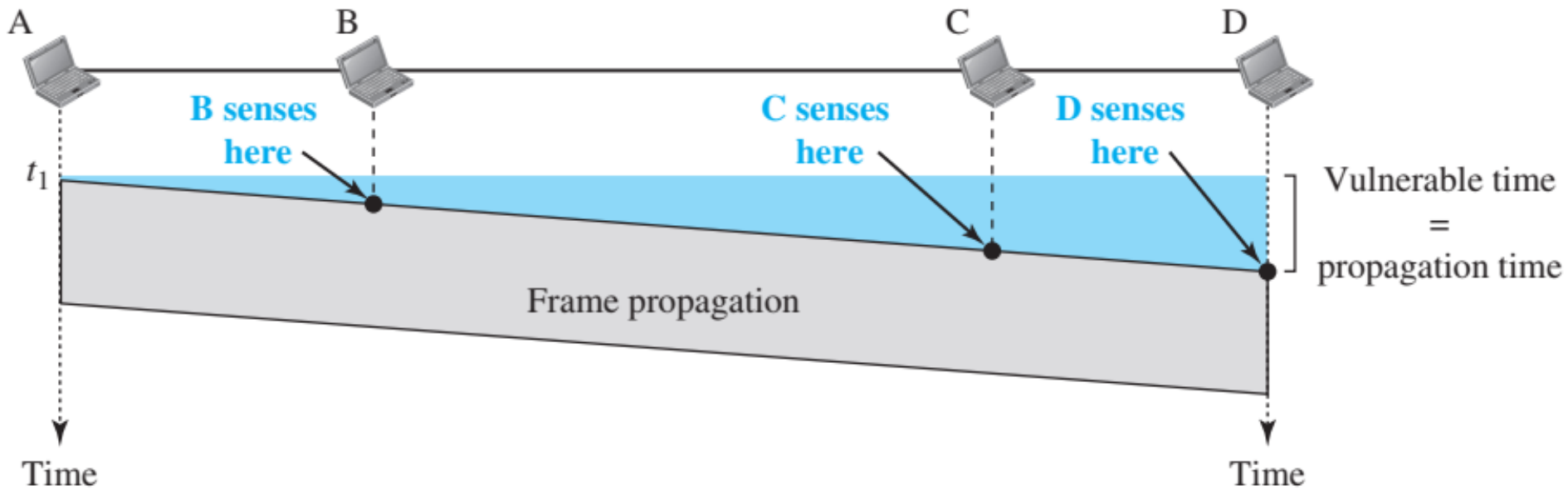

S (throughput per frame time)
0.40
0.30
0.20
0.10
Slotted ALOHA: S = Ge^−G
Pure ALOHA: S = Ge^−2G
0
0.5
1.0
1.5
2.0
3.0
G (attempts per packet time)

Carrier sense multiple access (CSMA)

- Requires that each station first listen to the medium before sending.

"sense before transmit" or "listen before talk"



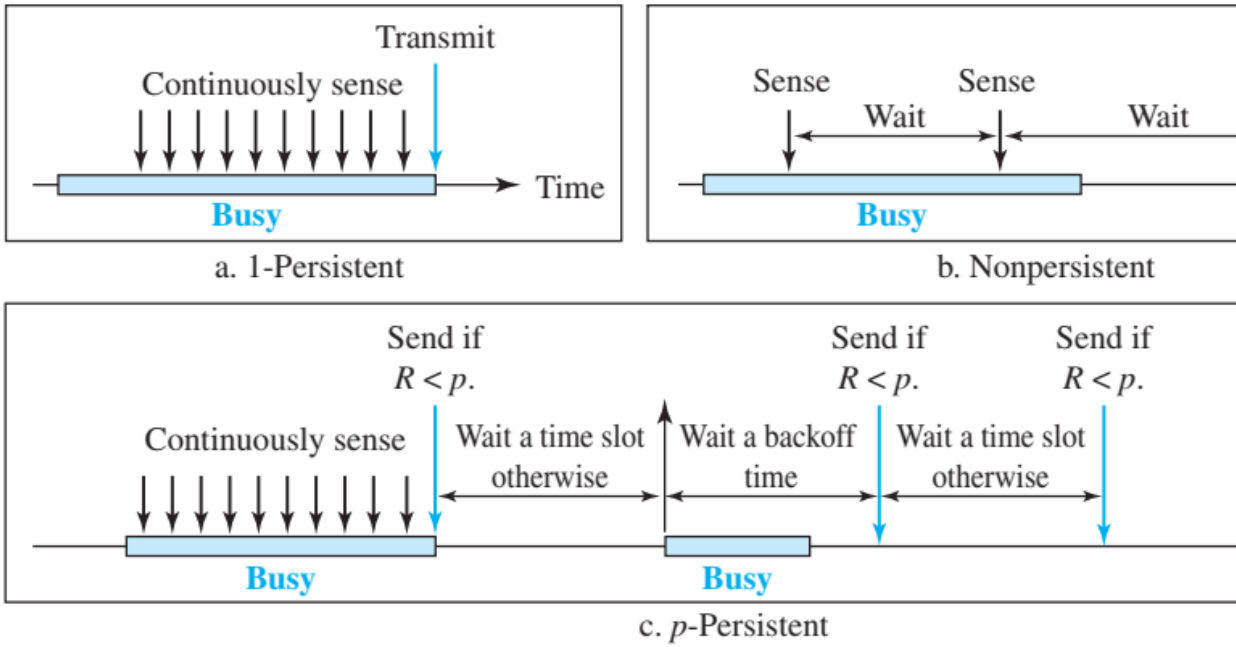


CSMA

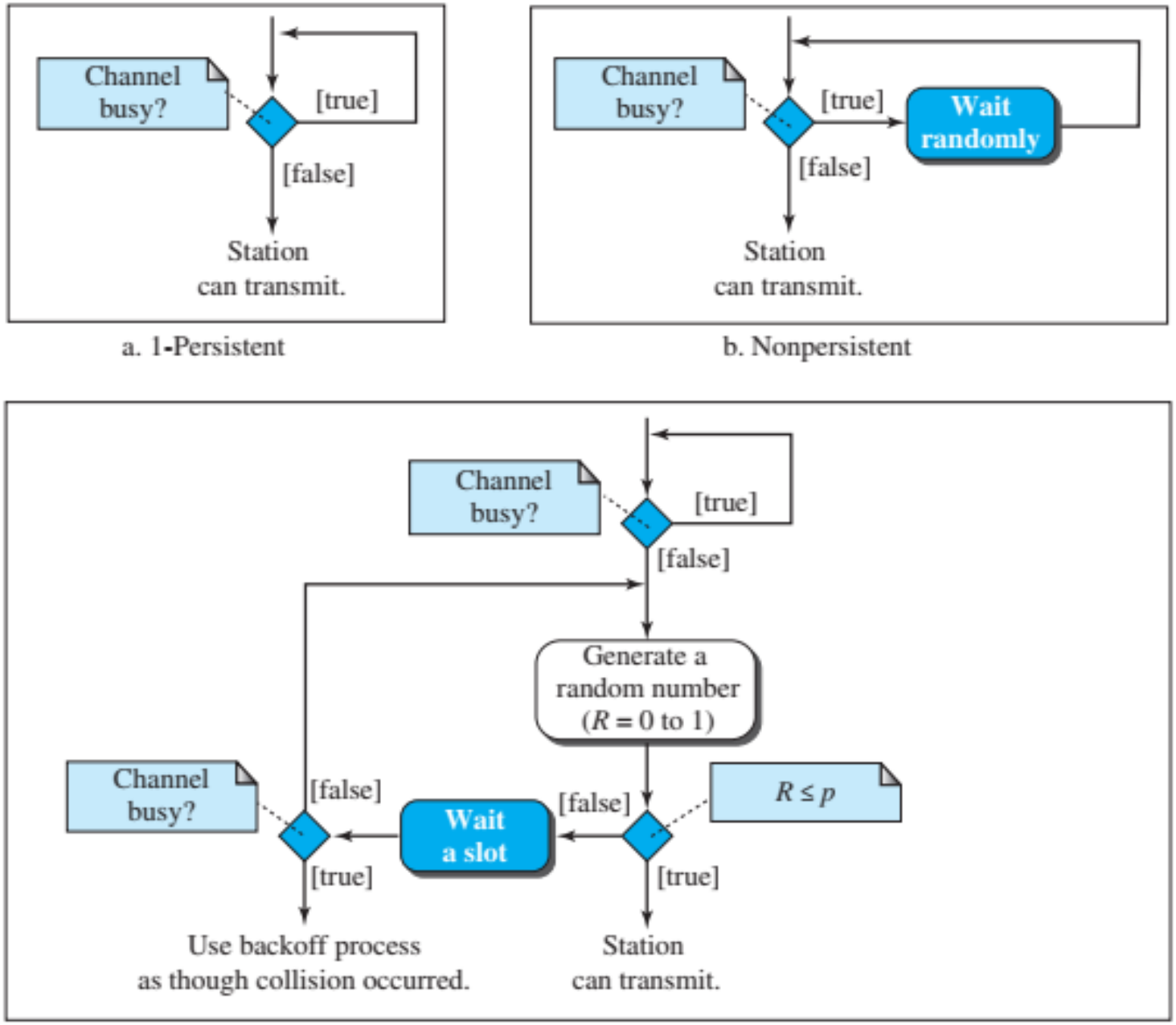
- THE VULNERABLE TIME FOR CSMA IS THE PROPAGATION TIME T_P.

CSAM

- Persistence Methods: What should a station do if the channel is busy? Or What should a station do if the channel is idle?



CSM



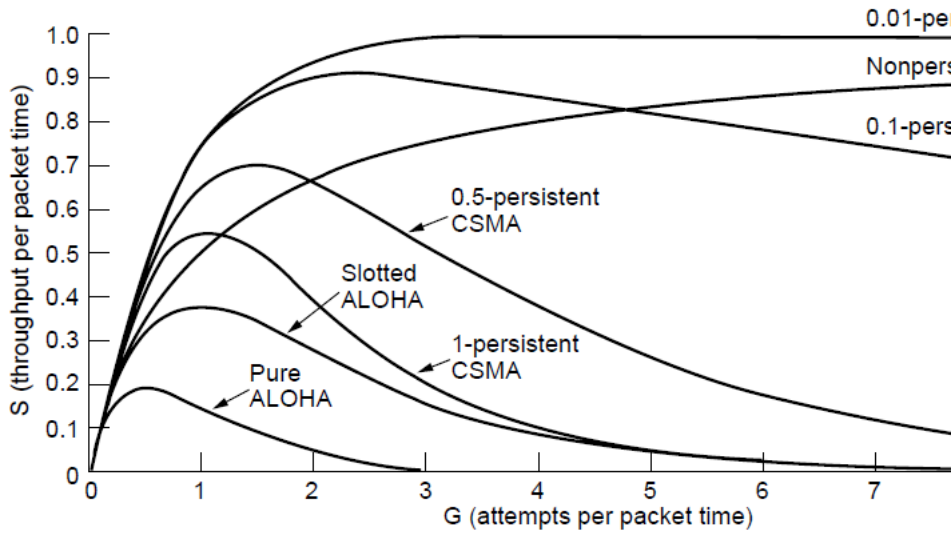
a. 1-Persistent

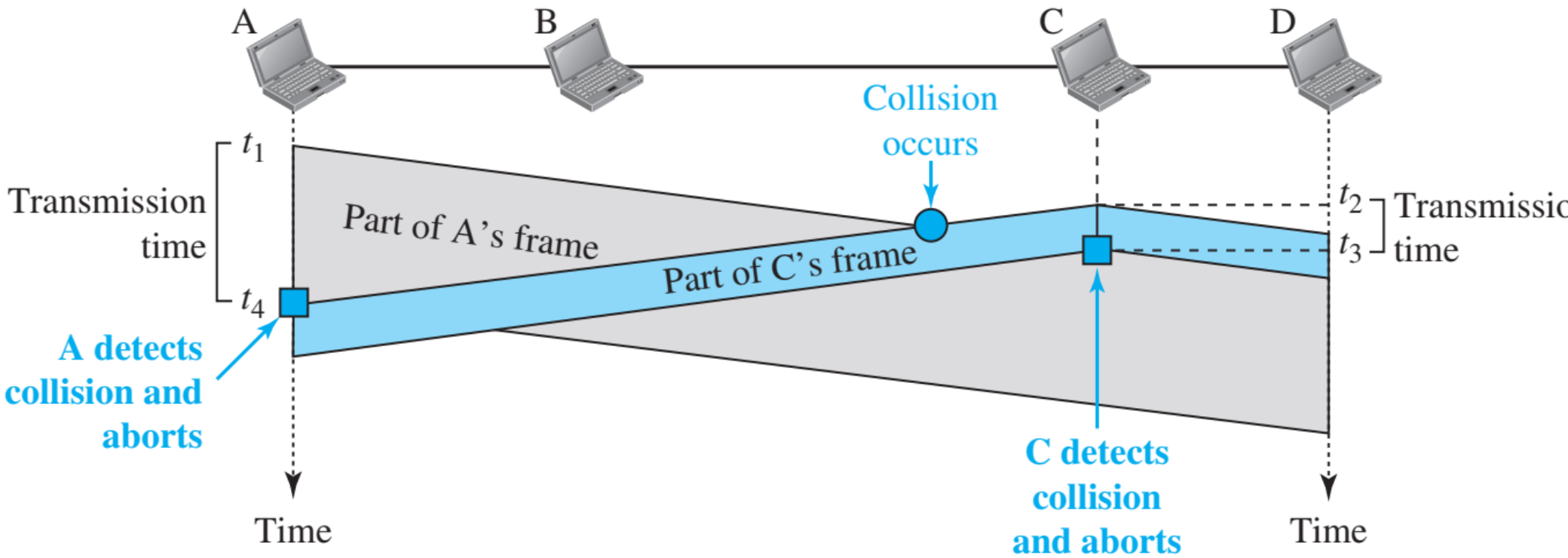
b. Nonpersistent

c. p-Persistent

CSMA

- The protocol, known as CSMA/CD (CSMA with Collision Detection), is the basis of the classic Ethernet LAN.
- The protocol, known as CSMA/CA (CSMA with Collision Avoidance), is the basis of the wireless networks.

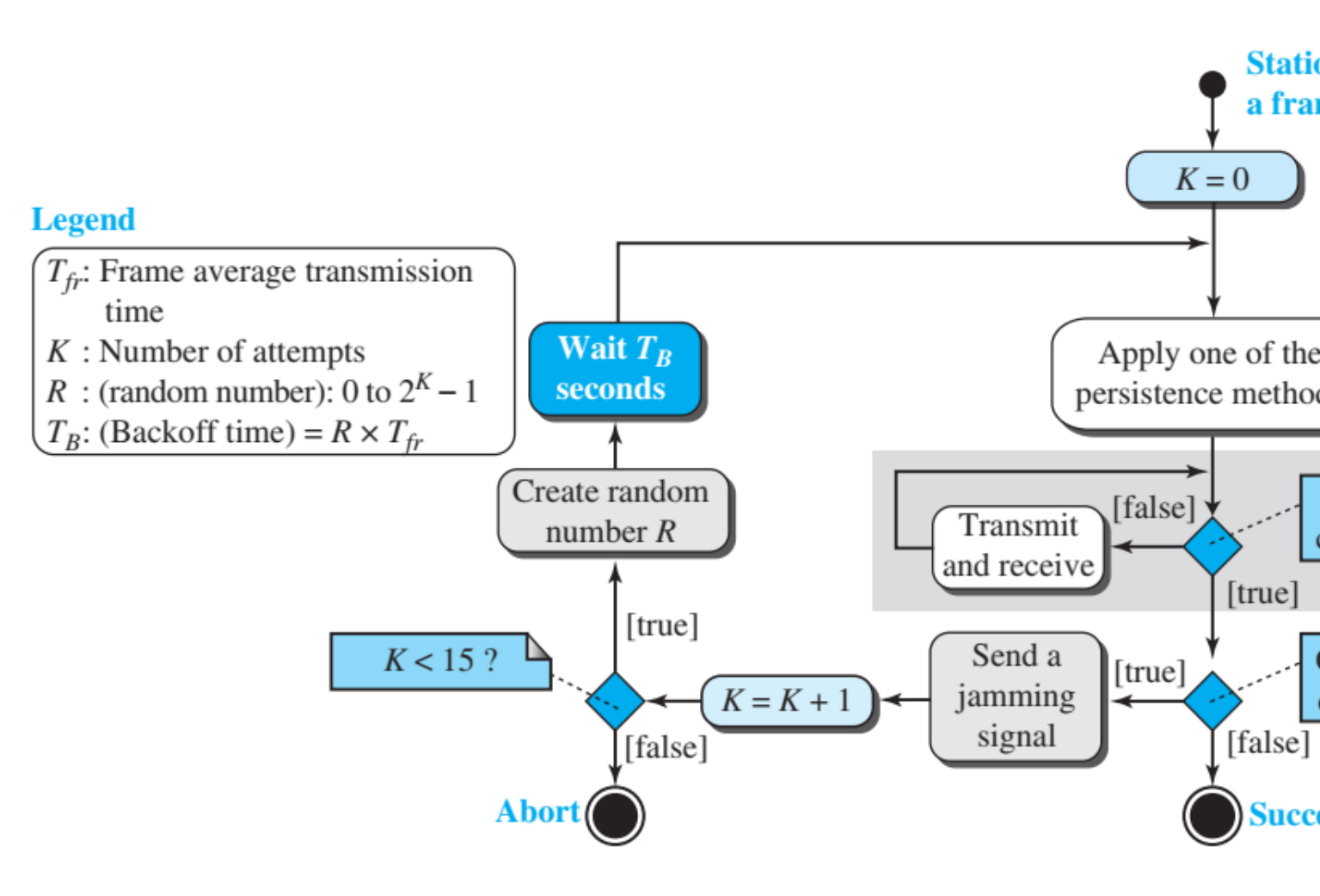



A
B
C
D
Collision occurs
t_1
Transmission time
t_4
Part of A's frame
Part of C's frame
t_2
t_3
Transmissio
time
A detects collision and aborts
C detects collision and aborts
Time
Time

CSMA/CD

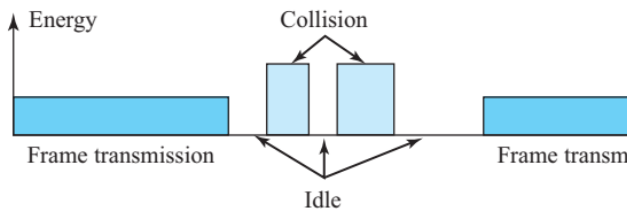
CSMA/CD

- For CSMA/CD to work, we need a restriction on the frame size.
- Minimum frame size is $T_{fr} = 2T_P$

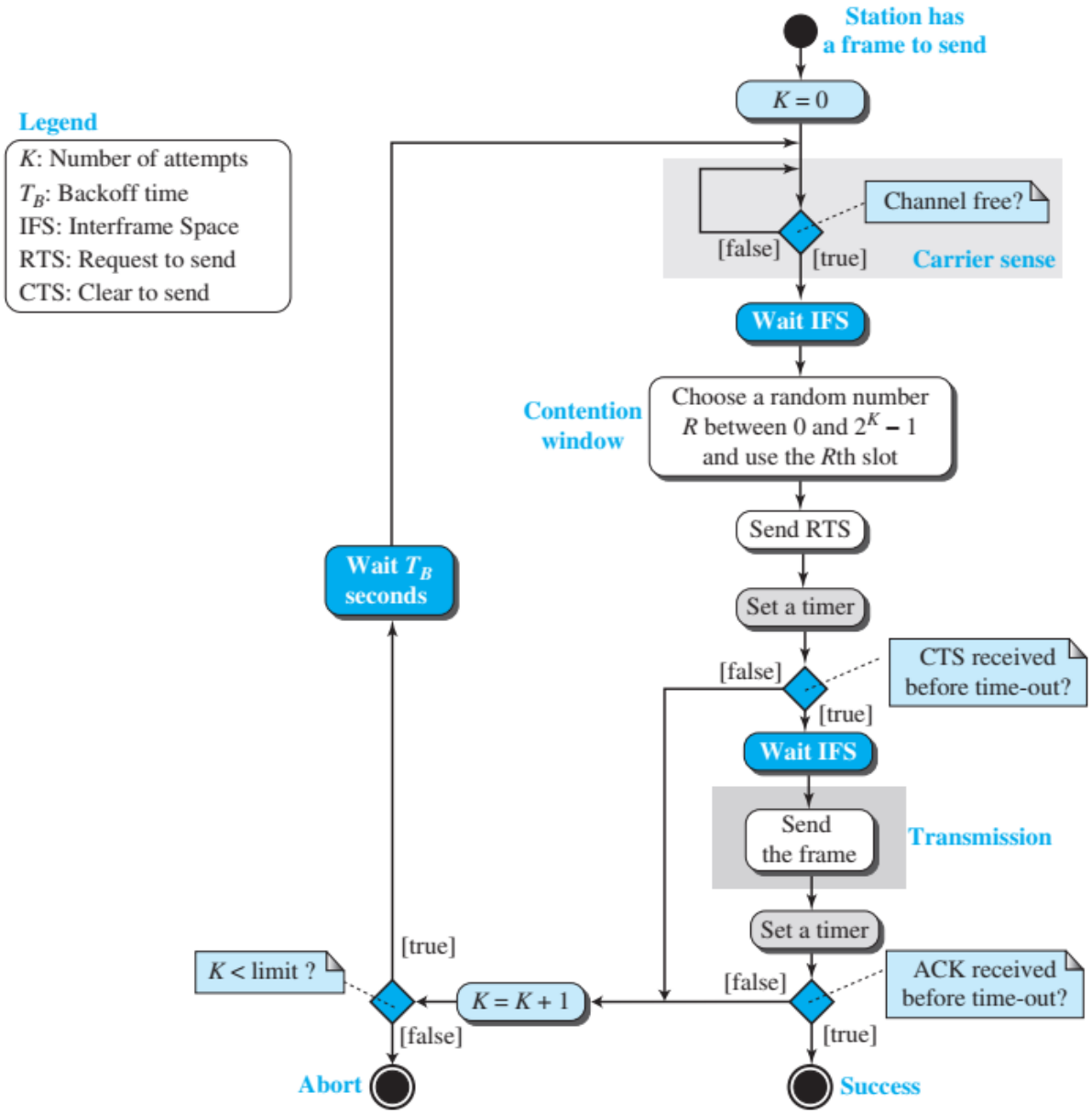


CSMA/CD Performance

- Throughput: The maximum throughput occurs at a different value of G and is based on the persistence method and the value of p in the p-persistent approach.
- For the 1-persistent method, the maximum throughput is around 50 percent when G = 1.
- For the nonpersistent method, the maximum throughput can go up to 90 percent when G is between 3 and 8.



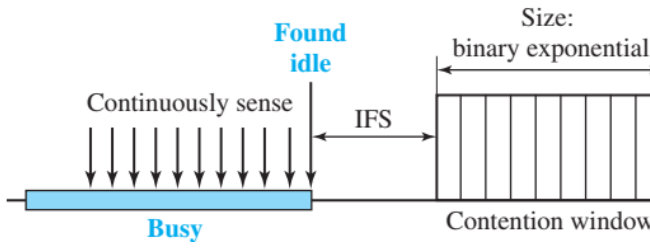
CSMA/C



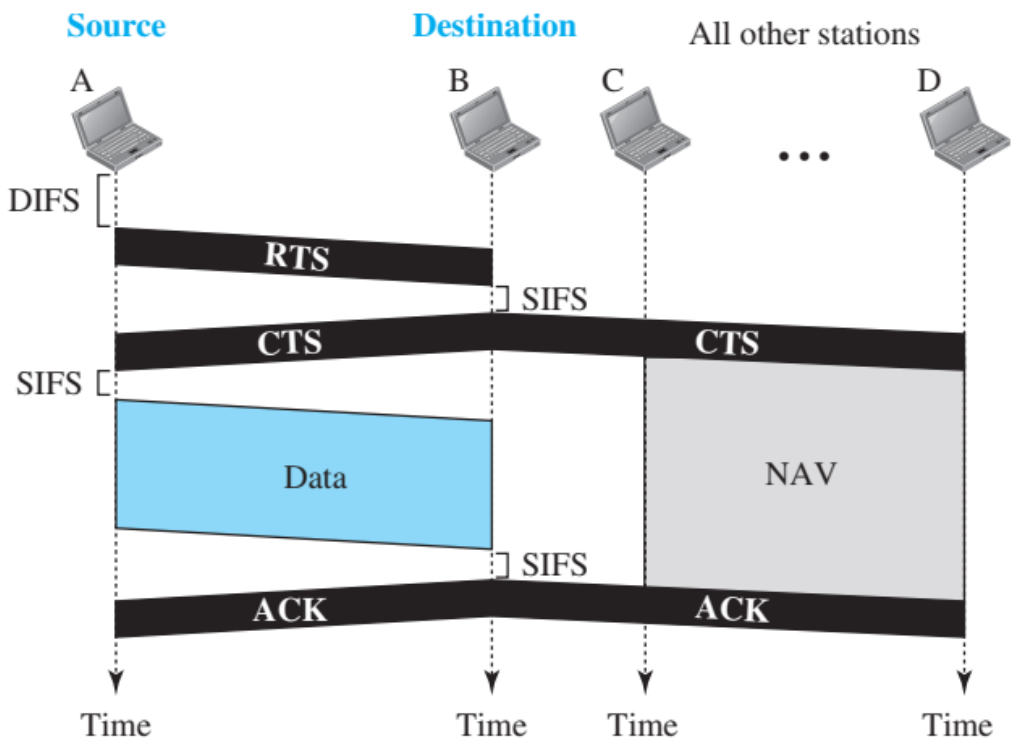
In wireless networks
abnormal energy detec
not practically employable
collisions happen. Henc
use CSAM/CD.

CSMA/CA

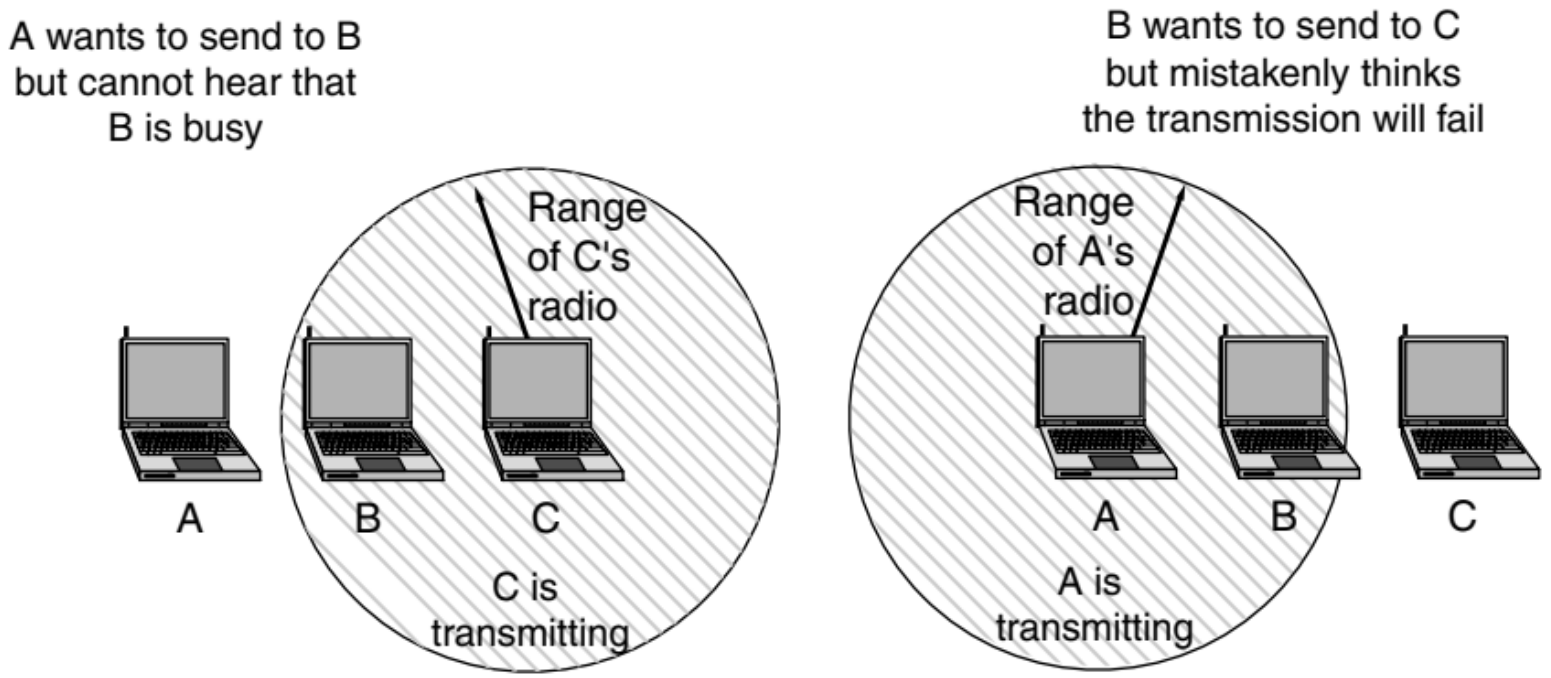
- Interframe Space (IFS): First, collisions are avoided by deferring transmission even if the channel is found idle. When an idle channel is found, the station does not send immediately. It waits for a period of time called the interframe space or IFS.
- Contention Window: A station that is ready to send chooses a random number of slots as its wait time. The number of slots in the window changes according to the binary exponential back off strategy.
- Acknowledgment: The positive acknowledgment and the time-out timer can help guarantee that the receiver has received the frame.
- When a station sends an RTS frame, it includes the duration of time that it needs to occupy the channel. The stations that are affected by this transmission create a timer called a network allocation vector (NAV).



CSMA/CA Frame Exchange Timeline



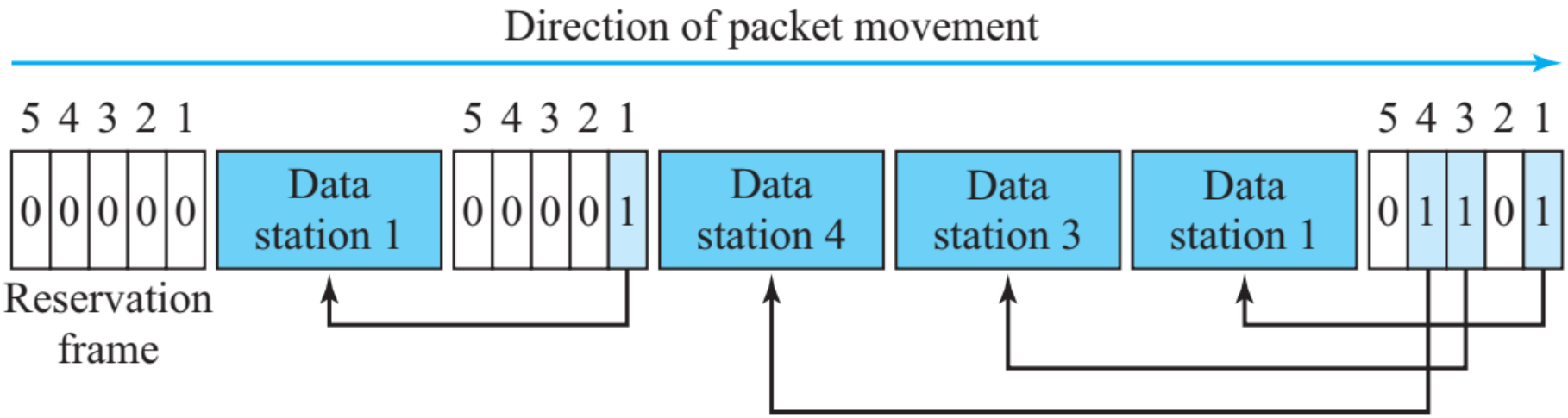
Hidden Station and Exposed Station Problem



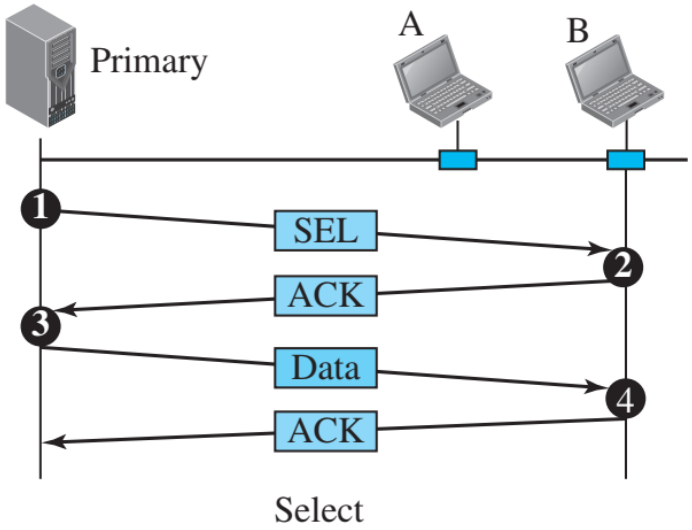
The hidden terminal problem. The exposed terminal problem.

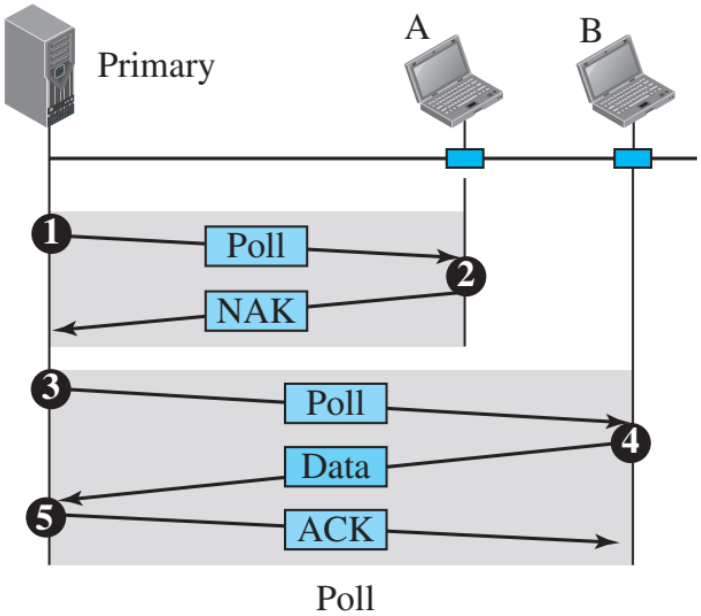
Controlled Access Protocols (Not Part of Syllb)

- Some times also called as collision free protocols
- 3 Protocols
  - Reservation
  - Polling
  - Token Passing



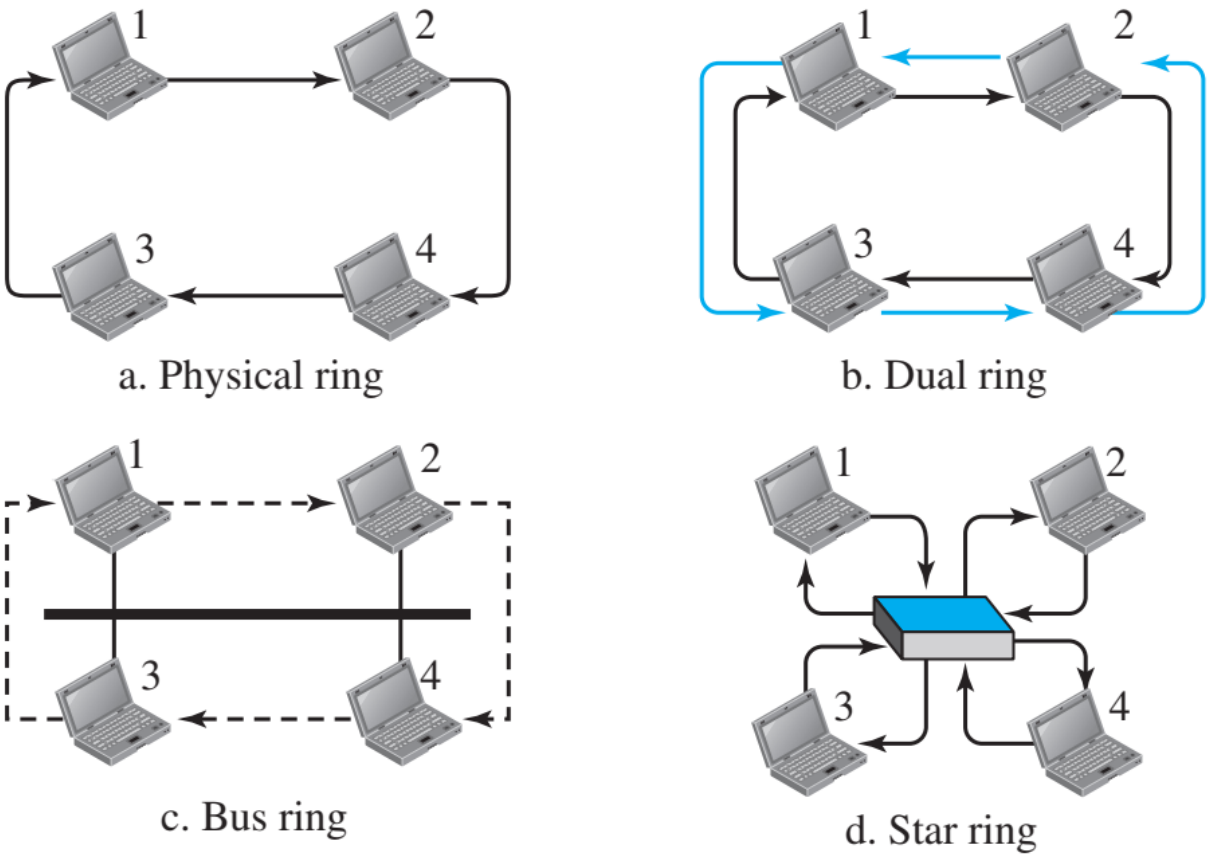
Reservation


Primary
A
B
1
SEL
2
ACK
3
Data
4
ACK
Select


Primary
A
B
1
Poll
2
NAK
3
Poll
4
Data
5
ACK
Poll

Polling

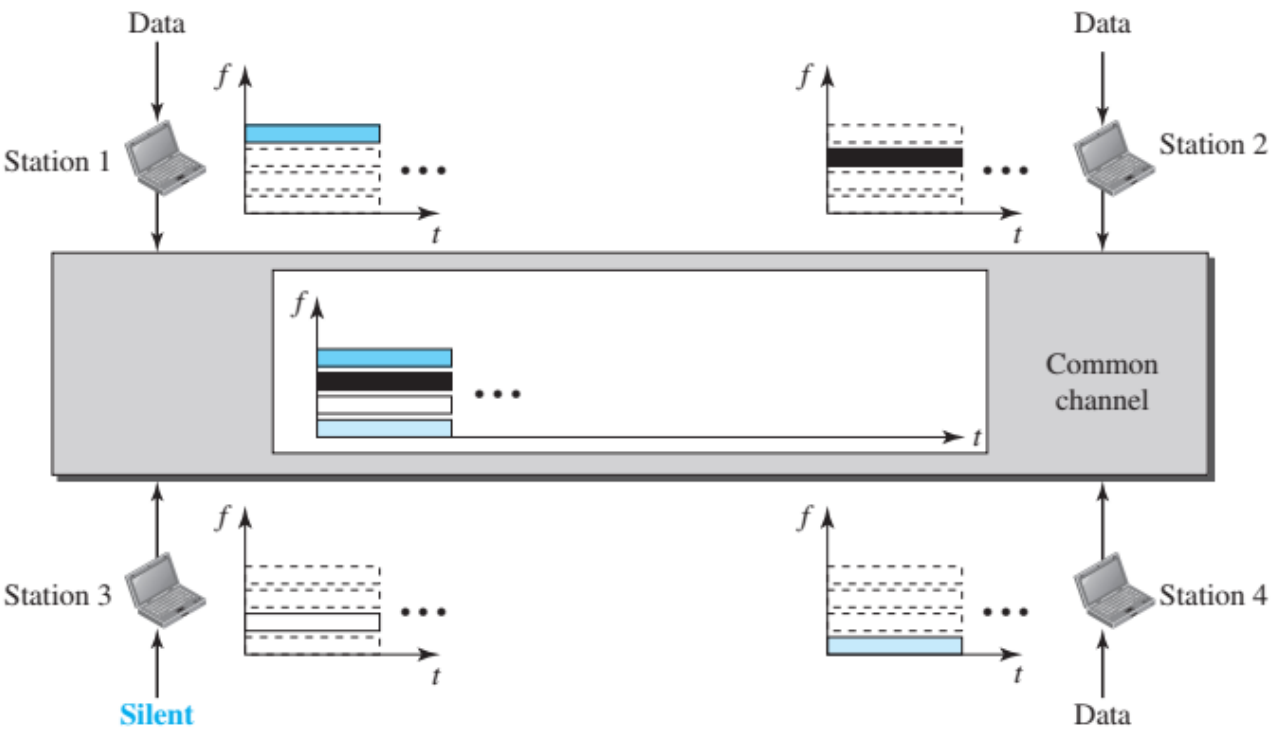
Token Passing



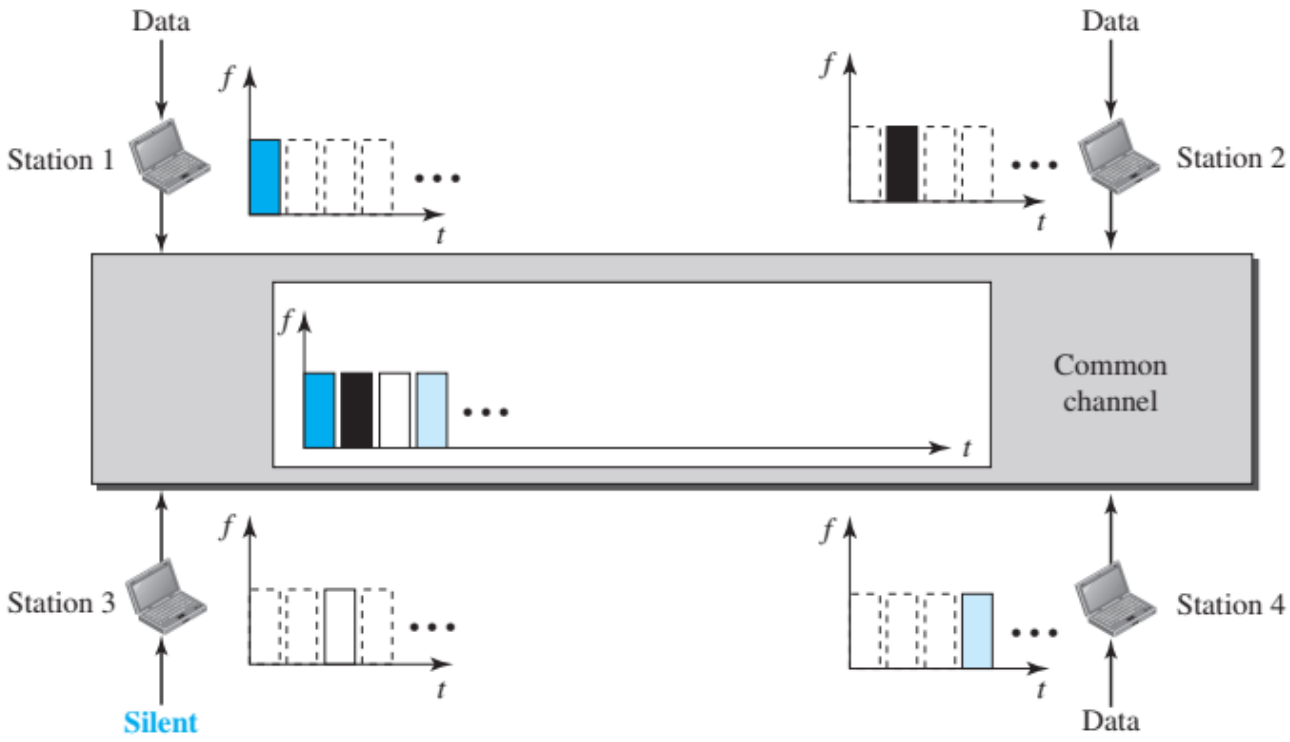
Channelization (Not Part of Syllb)

- Channelization (or channel partition, as it is sometimes called) is a multiple-access method in which the available bandwidth of a link is shared in time, frequency, or through code, among different stations. In this section, we discuss three channelization protocols: FDMA, TDMA, and CDMA.

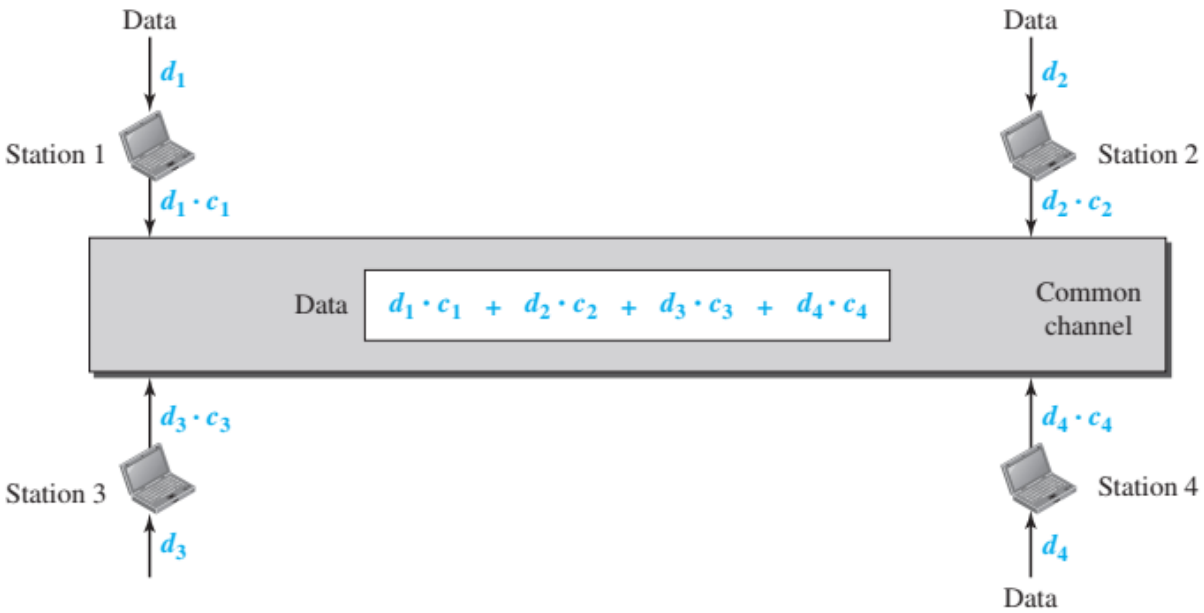
FDMA


Data
Station 1
Data
Station 2
Common channel
Station 3
Silent
Station 4
Data

TDMA

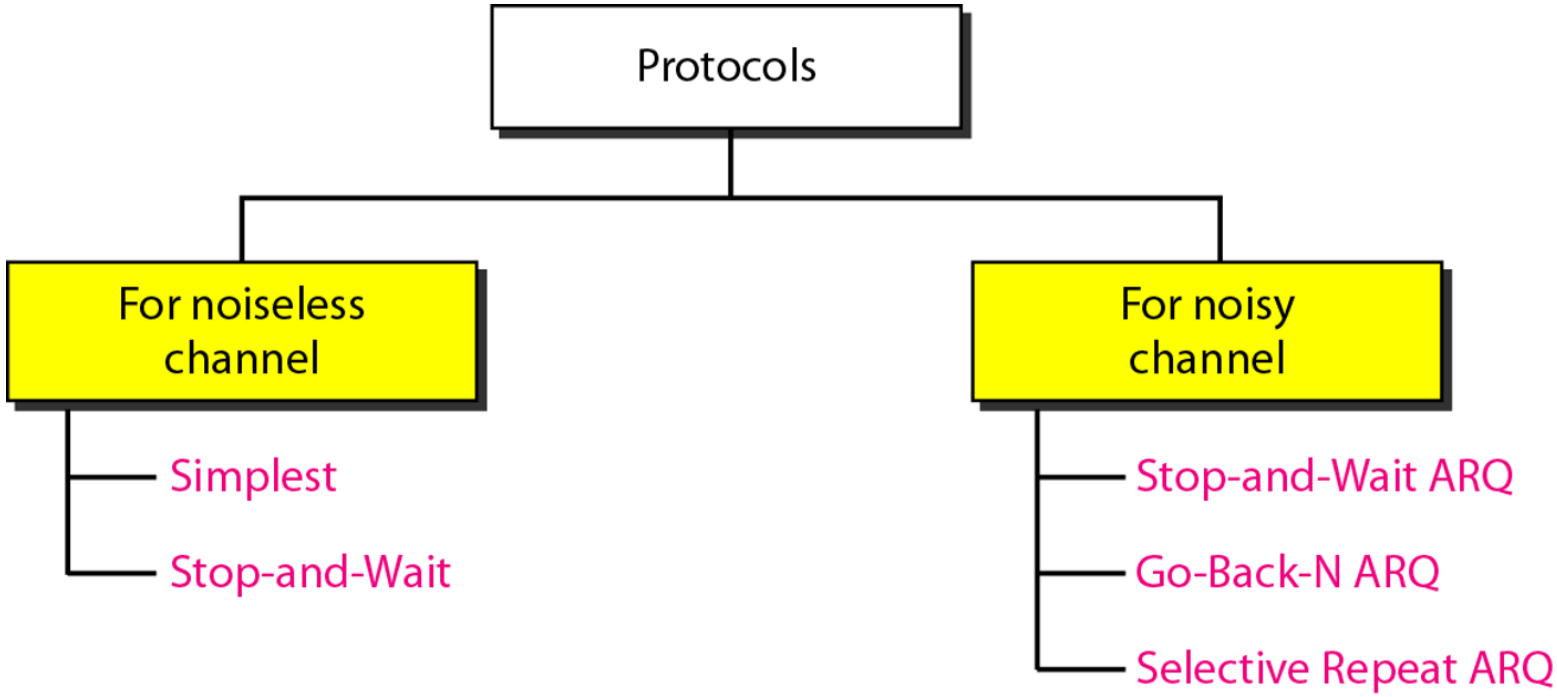

Data
Station 1
f
t
Data
Station 2
f
t
Common
channel
f
t
Station 3
f
t
Silent
Station 4
f
t
Data

CDMA



$$\text{data} = (d_1 \cdot c_1 + d_2 \cdot c_2 + d_3 \cdot c_3 + d_4 \cdot c_4) \cdot c_1$$
$$= d_1 \cdot c_1 \cdot c_1 + d_2 \cdot c_2 \cdot c_1 + d_3 \cdot c_3 \cdot c_1 + d_4 \cdot c_4 \cdot c_1 = 4 \times d_1$$

C_1	C_2	C_3	C_4
[+1 +1 +1 +1]	[+1 −1 +1 −1]	[+1 +1 −1 −1]	[+1 −1 −1 +1]



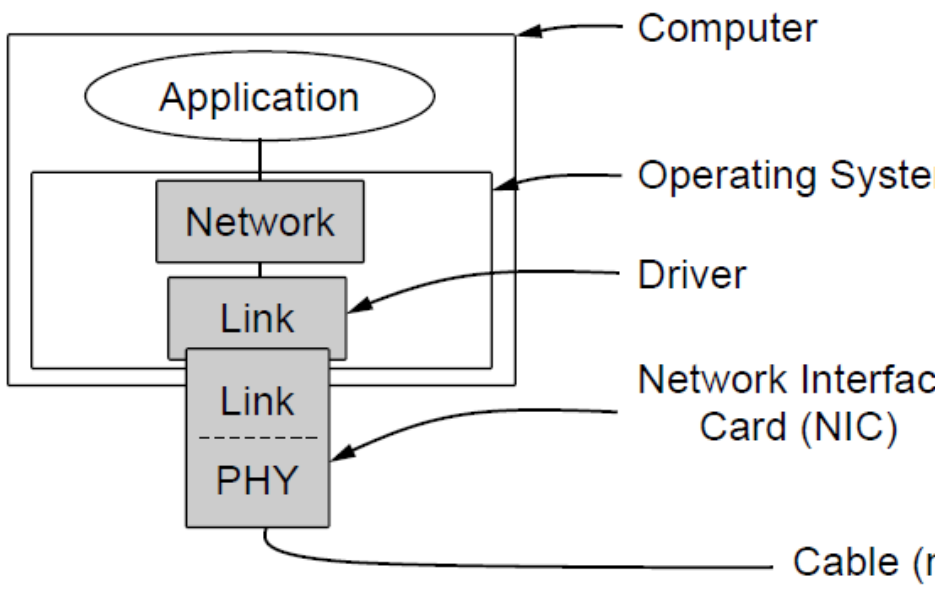
Elementary Data Link Protocols

Elementary Data Link Protocols

- A Utopian Simplex Protocol
- A Simplex Stop and Wait Protocol
  - For error free channel
  - For noisy channel
- Sliding Window Protocols
  - One-bit sliding window protocol
  - Go-back-N
  - Selective Repeat

Elementary Data Link Protocol

- The physical layer process and some of the data link layer process run on dedicate hardware called a **NIC** (**Network Interface Card**).
- The rest of the link layer process and the network layer process run on the main CPU as part of the operating system, with the software for the link layer process often taking the form of a **device driver**.



Elementary Data Link Protocols : Terminology and Functions

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

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

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

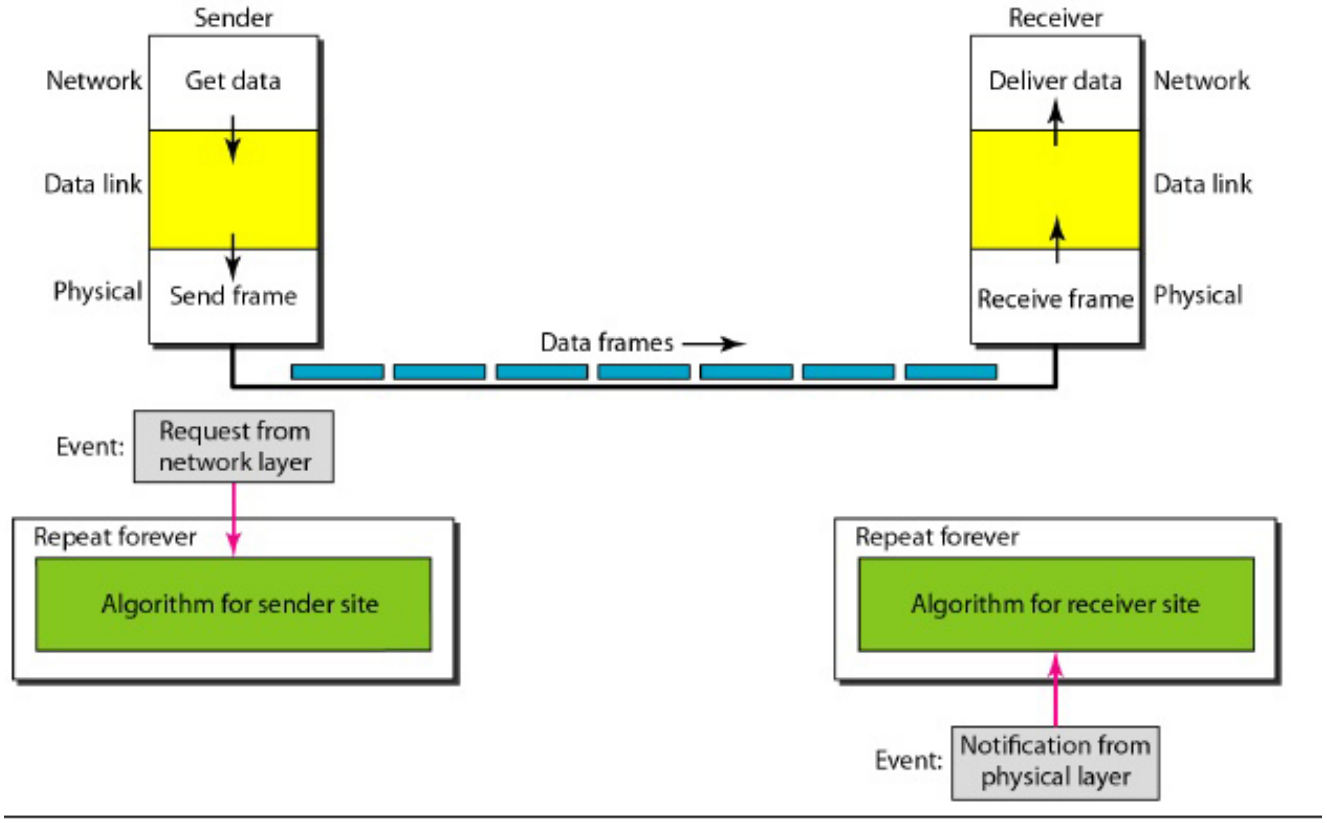
```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

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

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

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

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

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

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

Elementary Data Link Protocols : Terminology and Functio

```
/* 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 : Terminology and Functio

A Utopian Simplex Protoco



A Utopian Simplex Protocol Sender

- The utopia protocol is unrealistic because it does not handle either flow control or error correction. Its processing is close to that of an unacknowledged connectionless service that relies on higher layers to solve these problems.

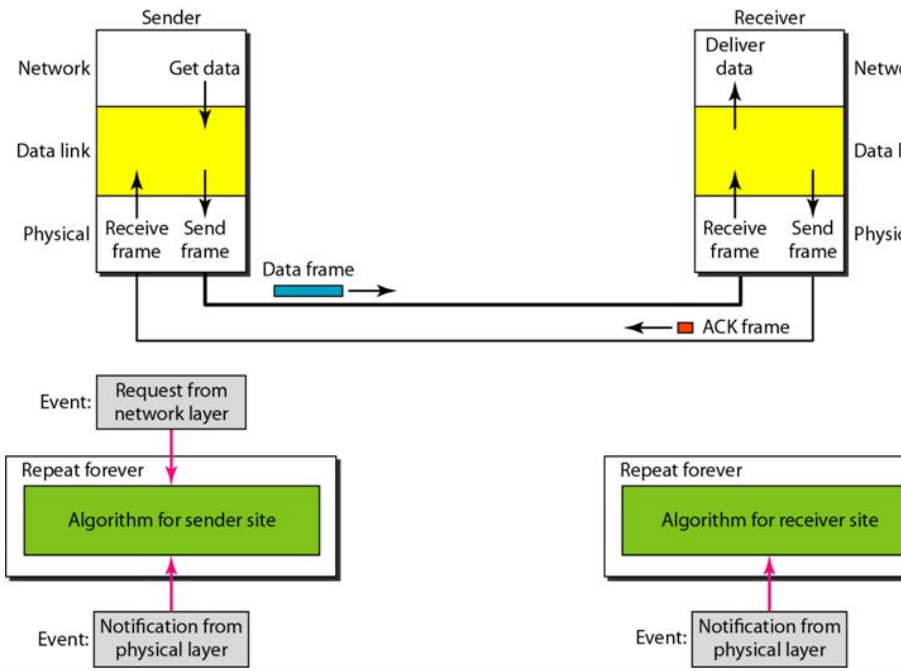
```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

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

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

A Utopian Simplex Protocol Receiver

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

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

Stop-and-Wait Protocol

- Uses Acknowledgments
- For Noiseless Channel:
  - We assume that the communication link is error free.
  - But must provide flow control to stop the faster sender to over power the slow receiver.
- For Noisy Channel:
  - We will do error control along with flow control.
  - Ack will be sent only if there is no error.

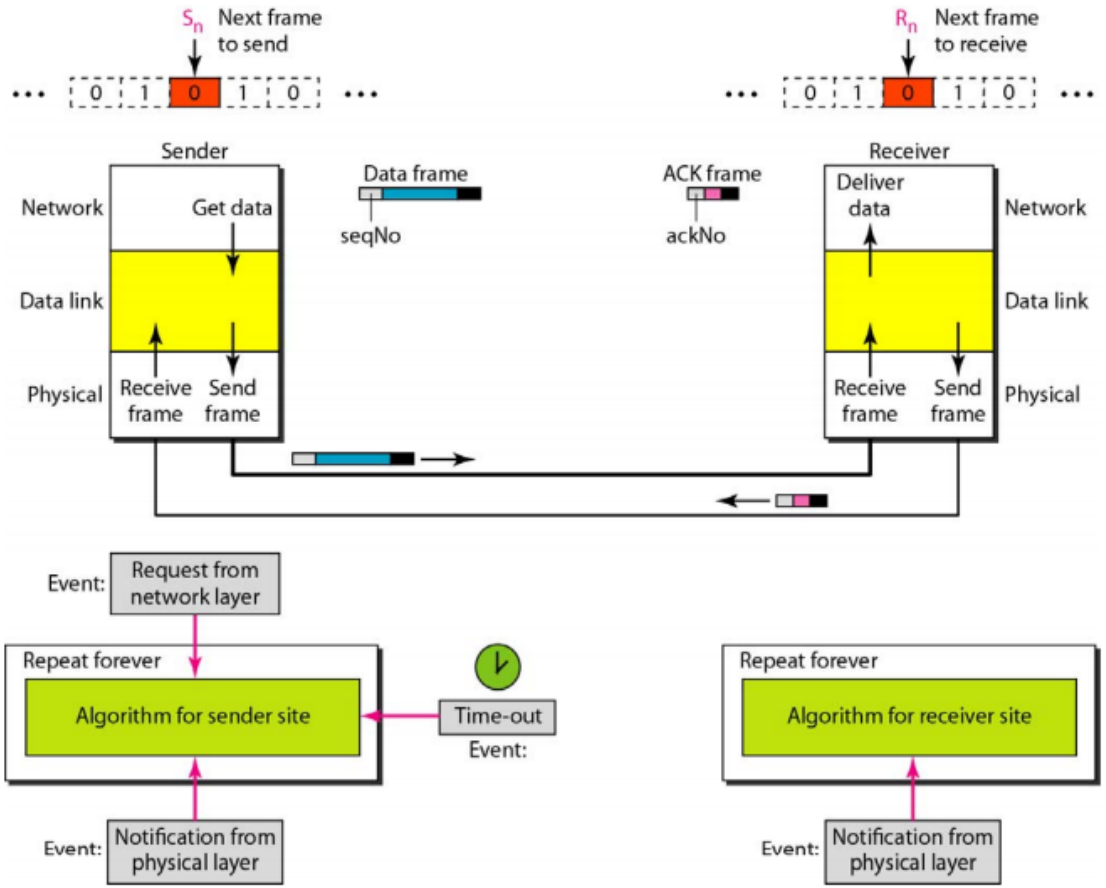


Simplex Stop-and-Wait Protocol for Noiseless Channel

```
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 */
  }
}
```

Simplex Stop-and-Wait Protocol for Noiseless Channel



Simplex Stop-and-Wait Protocol for Noisy Channel (Stop-and-Wait ARQ)

Simplex Stop-and-Wait Protocol for Noisy Channel

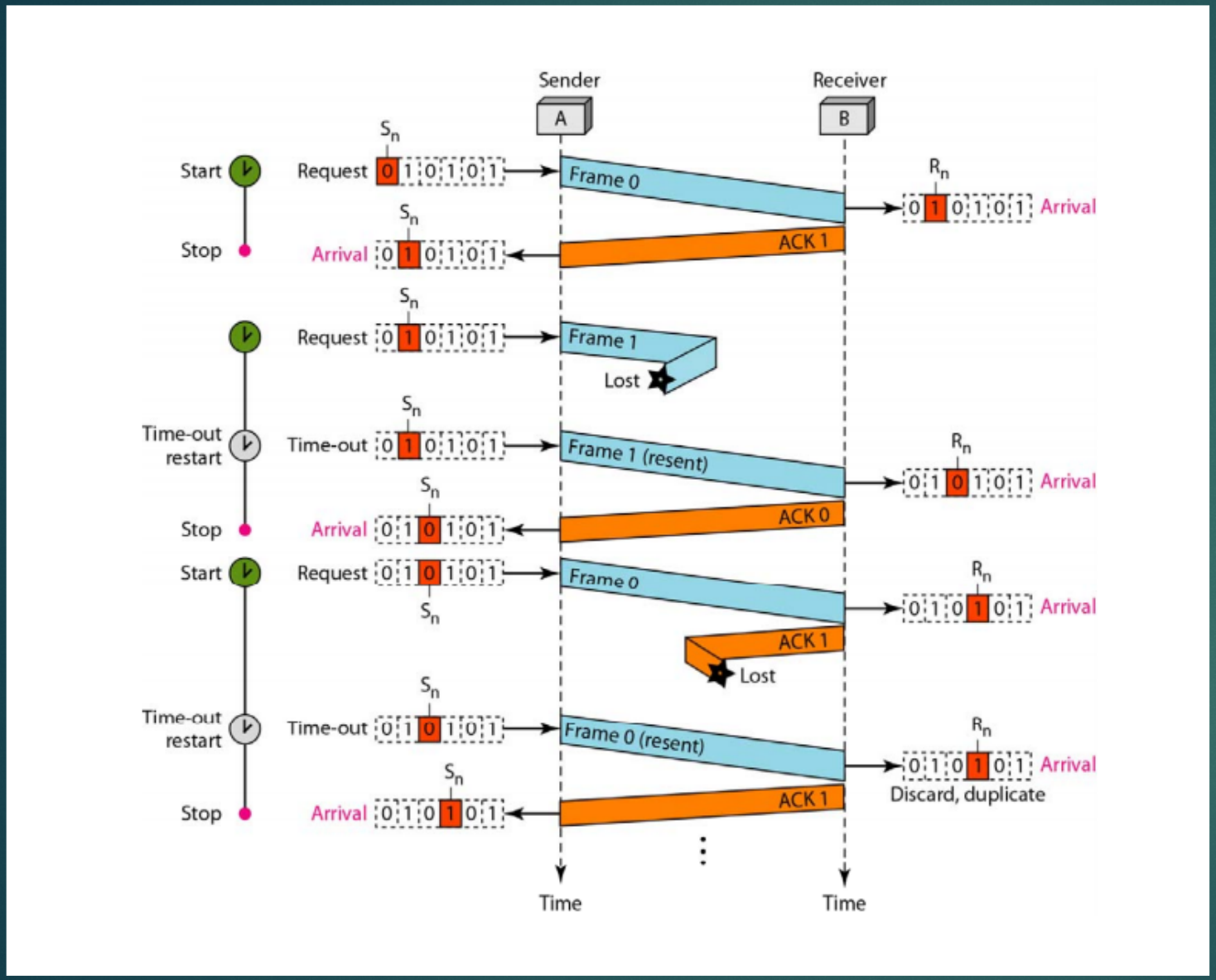
```
/* Protocol 3 (PAR) allows unidirectional data flow over an unreliable chann

#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 outgoi
  frame s;                                 /* scratch variable */
  packet buffer;                           /* buffer for an outbound pac
  event_type event;
```

Simplex Stop-and-Wait Protocol for Noisy Channel

```
next_frame_to_send = 0;                    /* initialize outbound sequence nur
from_network_layer(&buffer);               /* fetch first packet */
while (true) {
    s.info = buffer;                       /* construct a frame for transmissic
    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 ou
    wait_for_event(&event);                /* frame_arrival, cksum_err, timeou
    if (event == frame_arrival) {
        from_physical_layer(&s);           /* get the acknowledgement */
        if (s.ack == next_frame_to_send +1 ) {
            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 */
        }
    }
 }
}
```

Simplex Stop-and-Wait Protocol for 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 */
                  !check_sum_error    ) {
                 to_network_layer(&r.info);    /* pass the data to the network layer */
                 inc(frame_expected);          /* next time expect the other sequence nr */
                 s.ack = frame_expected;       /* tell which frame is being acked */
                 to_physical_layer(&s);        /* send acknowledgement */

            }
      }
  }
}
```



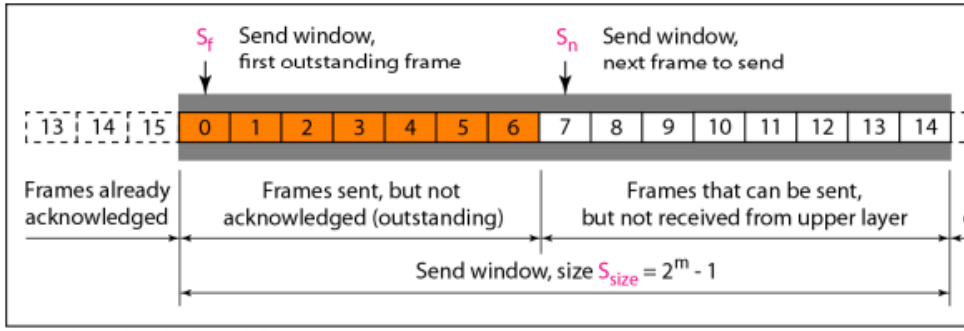
Simplex Stop-and-Wait Protocol for Noisy Channel

Sliding Window Protocols

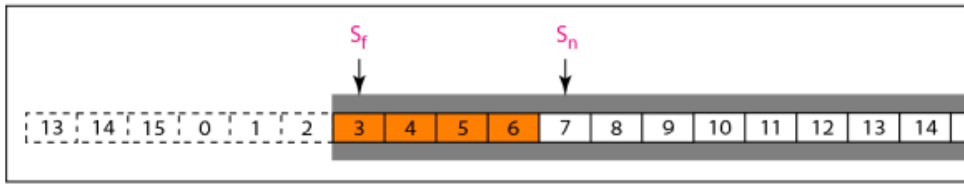
- The next three protocols are bidirectional protocols.
- The three differ among themselves in terms of efficiency, complexity, and buffer requirements.
- each outbound frame contains a sequence number, ranging from 0 up to some maximum $2^n - 1$, where n is size of sequence number field.
- The stop-and-wait sliding window protocol uses $n = 1$, restricting the sequence numbers 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 Protocol

- A one-bit sliding window protocol with size 1 is same as stop-and-wait protocol.
- Till this point we have assumed only one frame will be sent before getting an ack.



a. Send window before sliding



b. Send window after sliding

One-bit Sliding Window Protocol

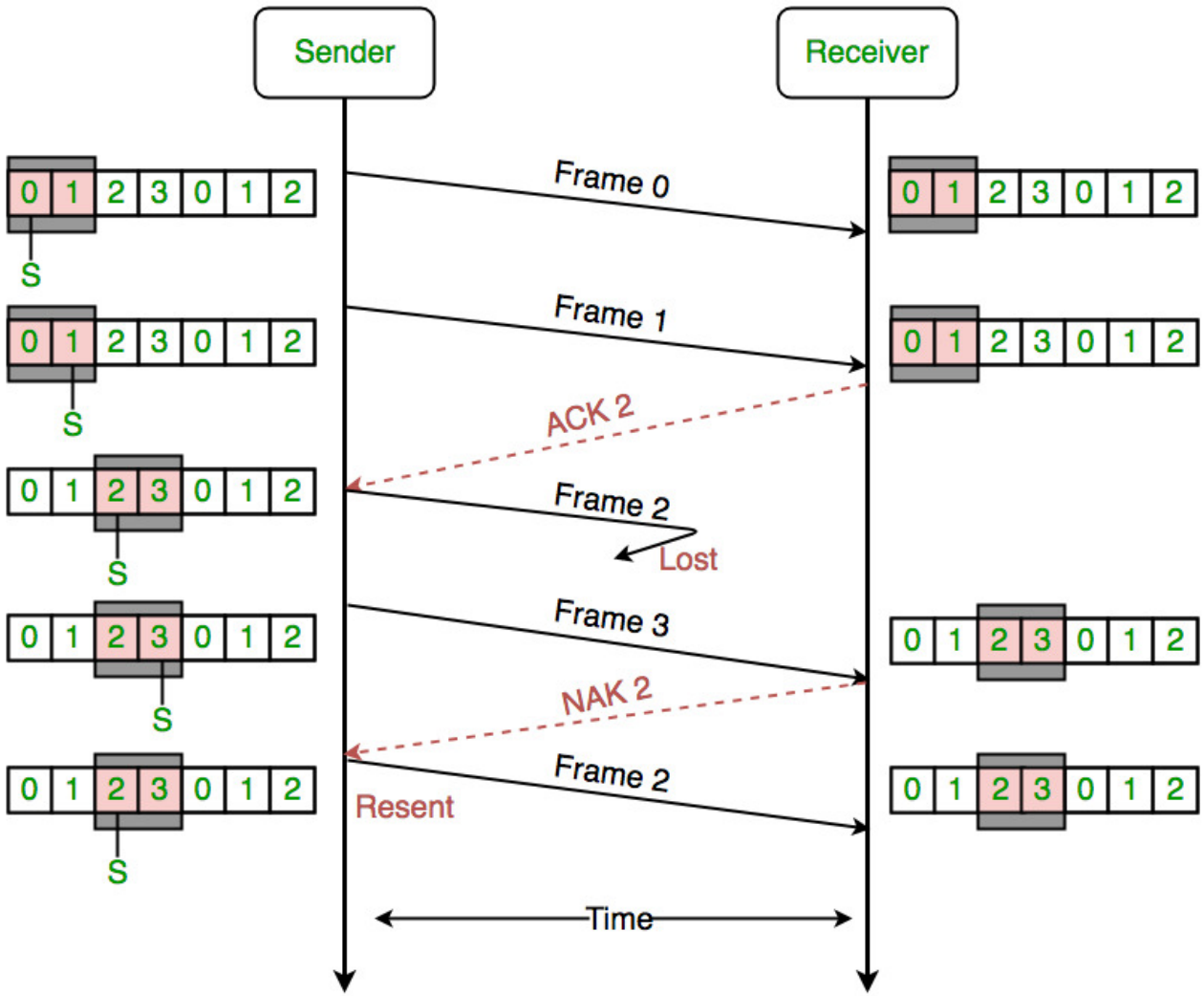
```
/* Protocol 4 (Sliding window) is bidirectional. */

#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 stre
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network
  s.info = buffer;                                  /* prepare to send the initial frame
  s.seq = next_frame_to_send;                       /* insert sequence number into fra
  s.ack = frame_expected;                           /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
```

One-bit Sliding Window Protocol

```
while (true) {
    wait_for_event(&event);                    /* frame_arrival, cksum_err, or
    if (event == frame_arrival) {              /* a frame has arrived undama
        from_physical_layer(&r);               /* go get it */
        if (r.seq == frame_expected
                !Check_sum_error ) {           /* handle inbound frame strear
            to_network_layer(&r.info);         /* pass packet to network layer
            inc(frame_expected);               /* invert seq number expected
        }
        if (r.ack == next_frame_to_send+1) {   /* handle outbound frame strea
            stop_timer(r.ack);                 /* turn the timer off */
            from_network_layer(&buffer);       /* fetch new pkt from network l
            inc(next_frame_to_send);           /* invert sender's sequence nu
        }
    }
```

One-bit Sliding Window Protocol

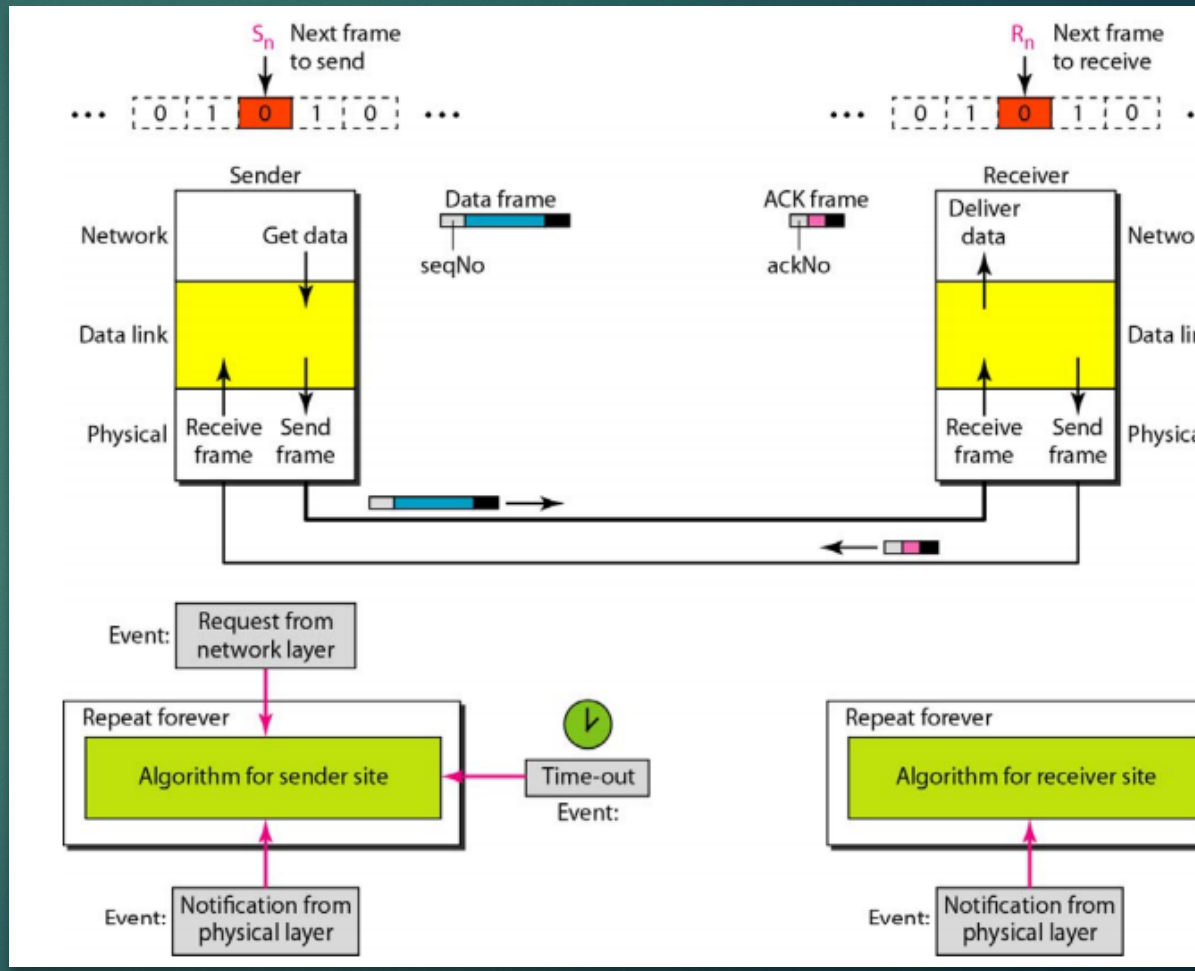
```
        s.info = buffer;                    /* construct outbound frame */
        s.seq = next_frame_to_send;         /* insert sequence number into it */
        s.ack = frame_expected;             /* seq number of last received fram
        to_physical_layer(&s);              /* transmit a frame */
        start_timer(s.seq);                 /* start the timer running */
    }
}
```



Sliding Window Protoco

Draw Back of Stop-and-Wait

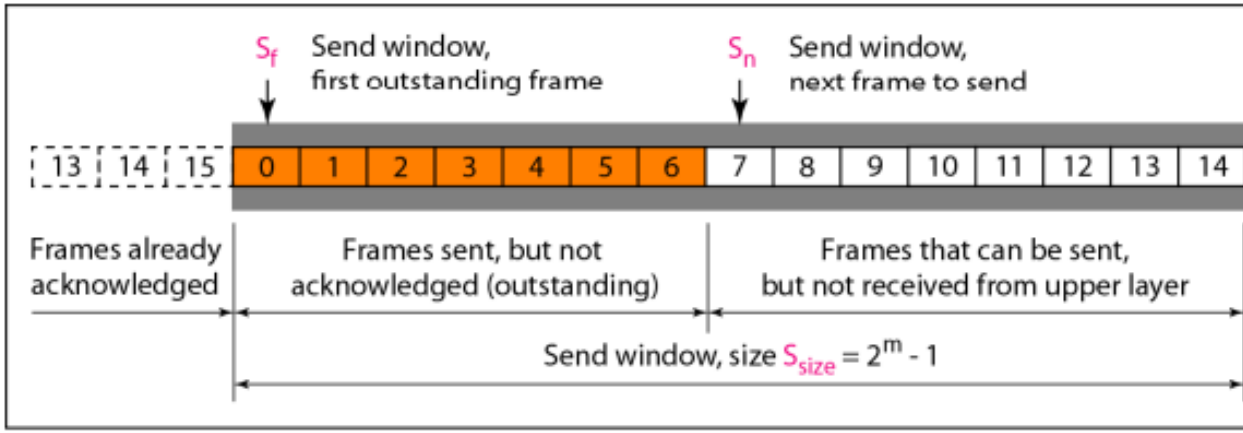
- Stopping the sender from sending even though the receiver can work at higher speed and the medium is free till sender receives an ACK.



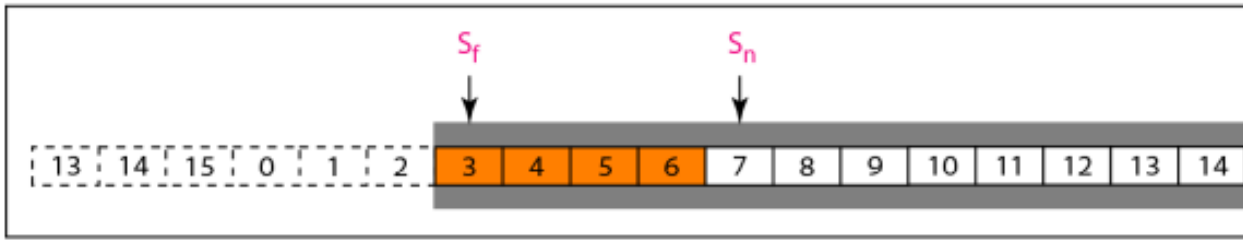
Go-Back-N Protocol (Sending Window)

The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}.

The send window can slide one or more slots when a valid acknowledgment arrives.



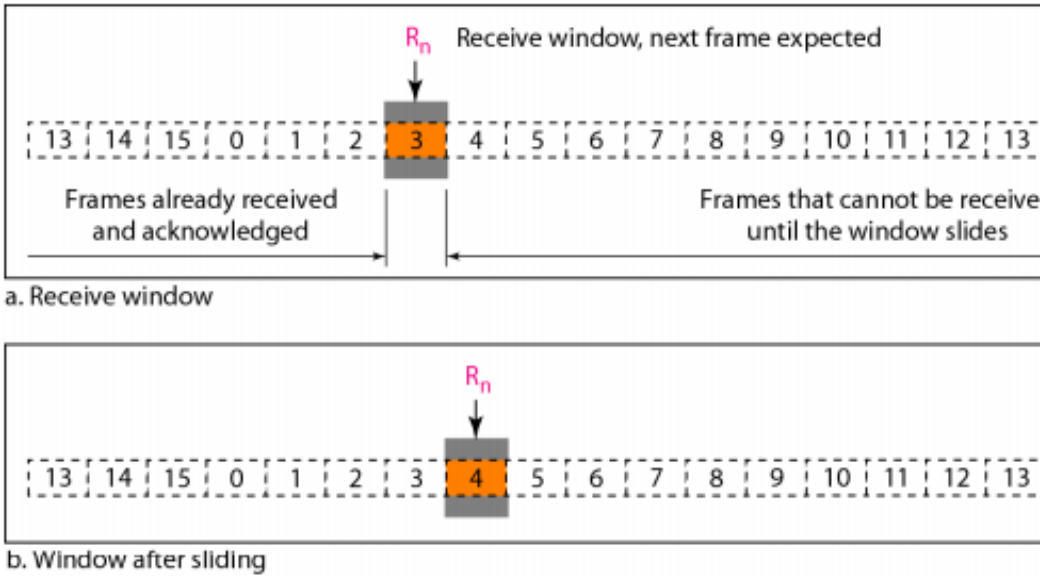
a. Send window before sliding



b. Send window after sliding

Go-Back-N Protocol (Receiver Window)

The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n. The window slides when a correct frame has arrived; sliding occurs one slot at a time.

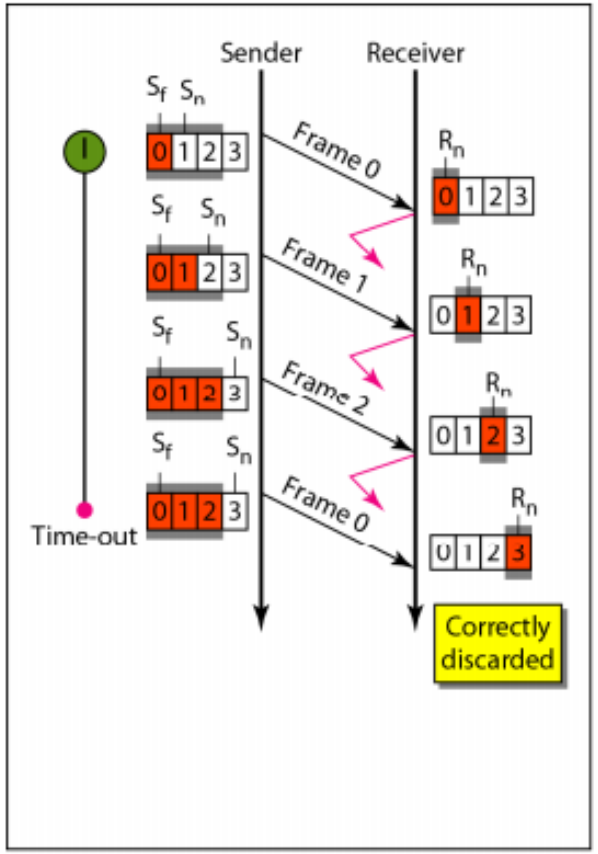


Go-Back-N Protocol

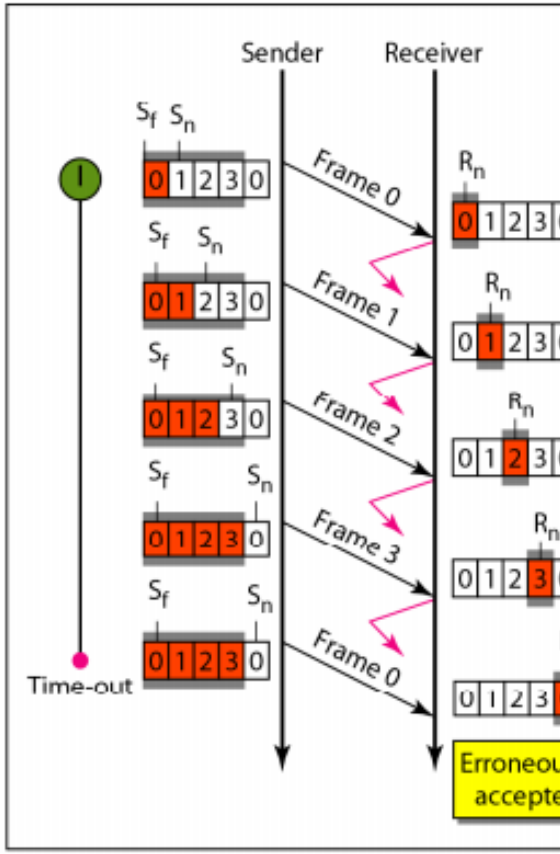
- When the sender receives an ack with exp_no > sf, all the frames sf will move to the position exp_no and all frames <seq_no are considered received correctly.

Go-Back-N Protocol

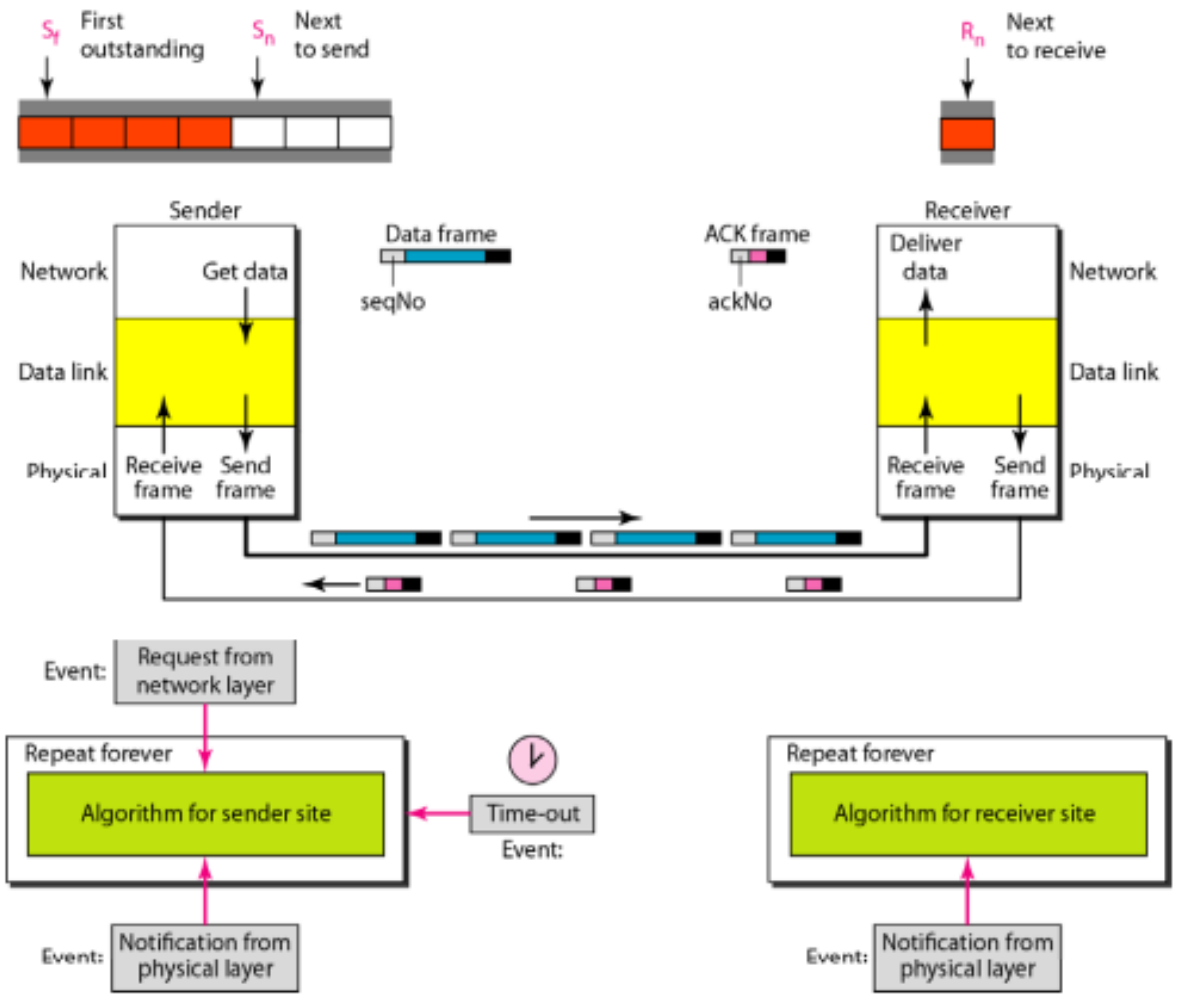
In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.



a. Window size < 2^m



b. Window size = 2^m



Go-Back N Protoco

Go-Back-N Protocol

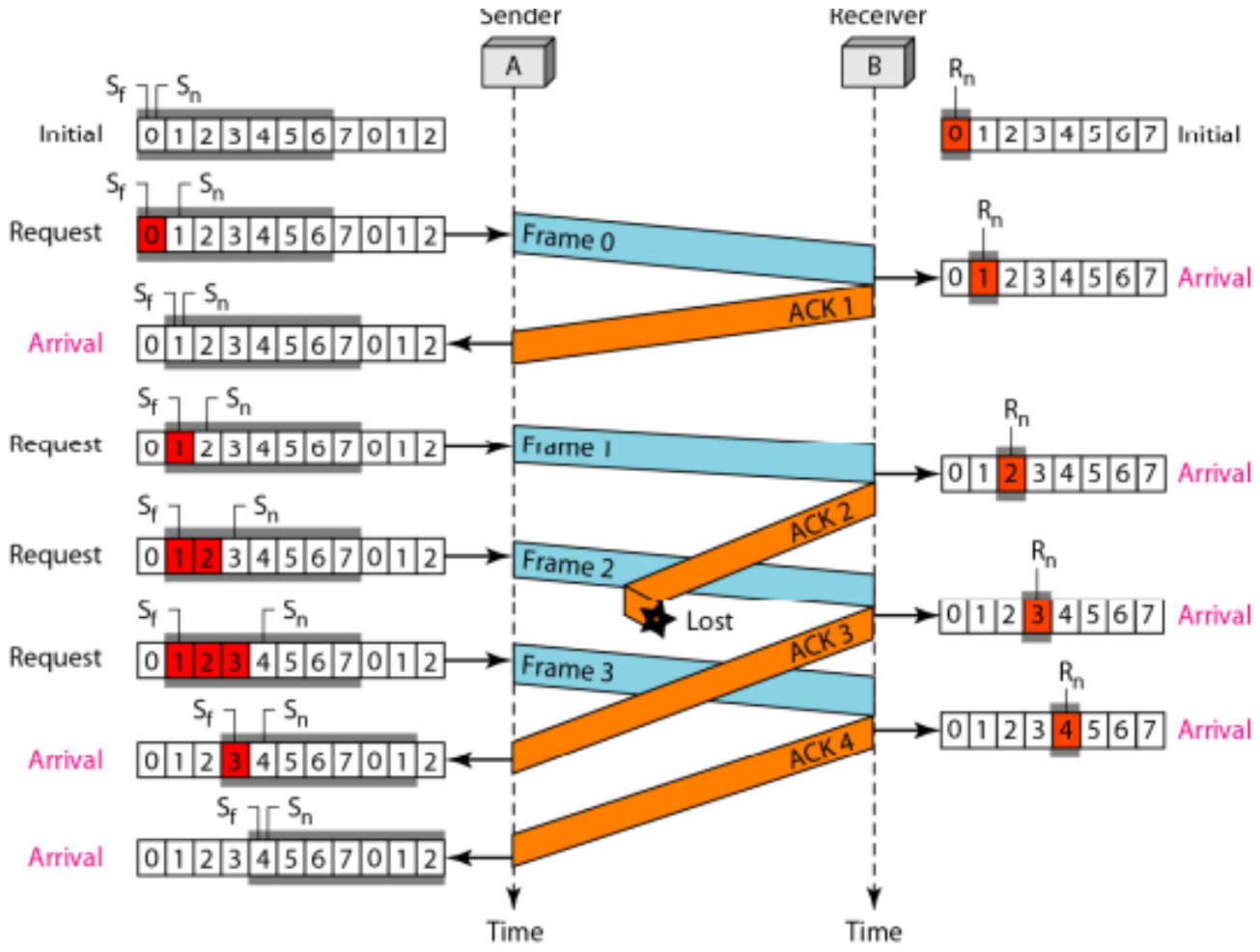
```
/* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike in the previous
   protocols, the network layer is not assumed to have a new packet all the time. Instead,
   the network layer causes a network_layer_ready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

#define MAX_SEQ 7
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 */
}
```

```
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;                           /* number of output buffers currently in use */
  seq_nr i;                                   /* used to index into the buffer array */
  event_type event;
```

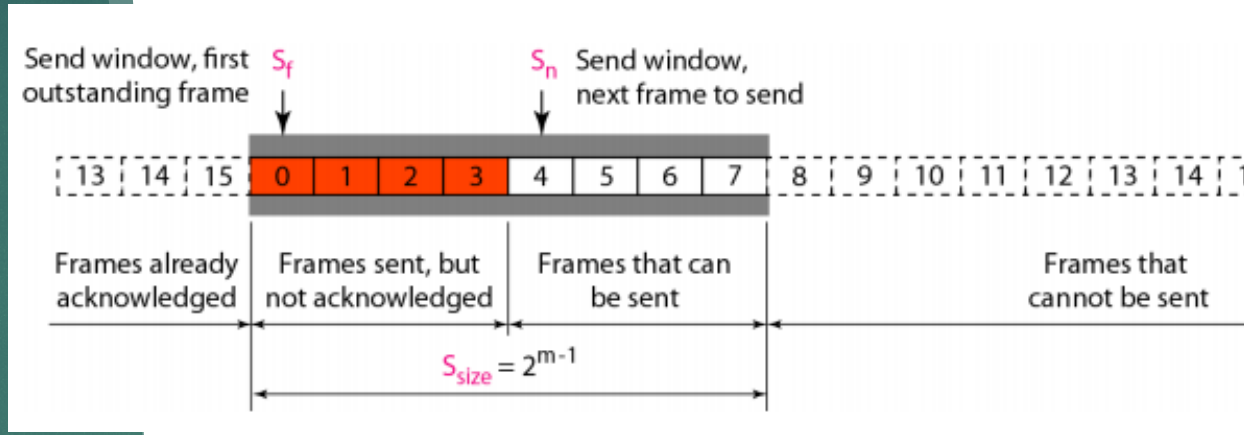
```
enable_network_layer();                    /* allow network_layer_ready events */
ack_expected = 0;                          /* next ack expected inbound */
next_frame_to_send = 0;                    /* next frame going out */
frame_expected = 0;                        /* number of frame expected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

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 */
               }
```

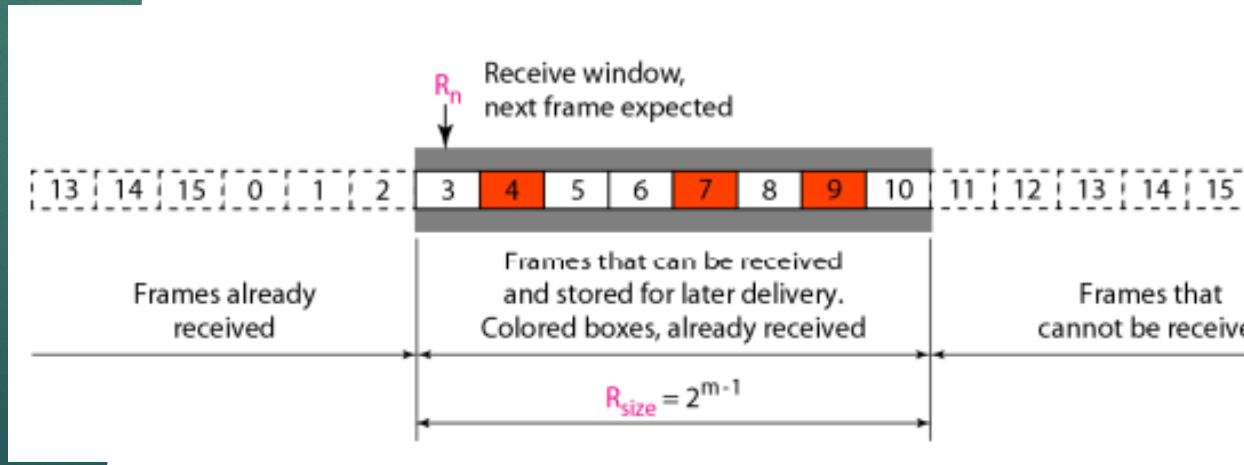
```
            /* Ack n implies n – 1, n – 2, etc.  Check for this. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                /* Handle piggybacked ack. */
                nbuffered = nbuffered – 1;          /* one frame fewer buffered */
                stop_timer(ack_expected);           /* 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 frame */
                inc(next_frame_to_send);            /* prepare to send the next one */
            }

    }

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

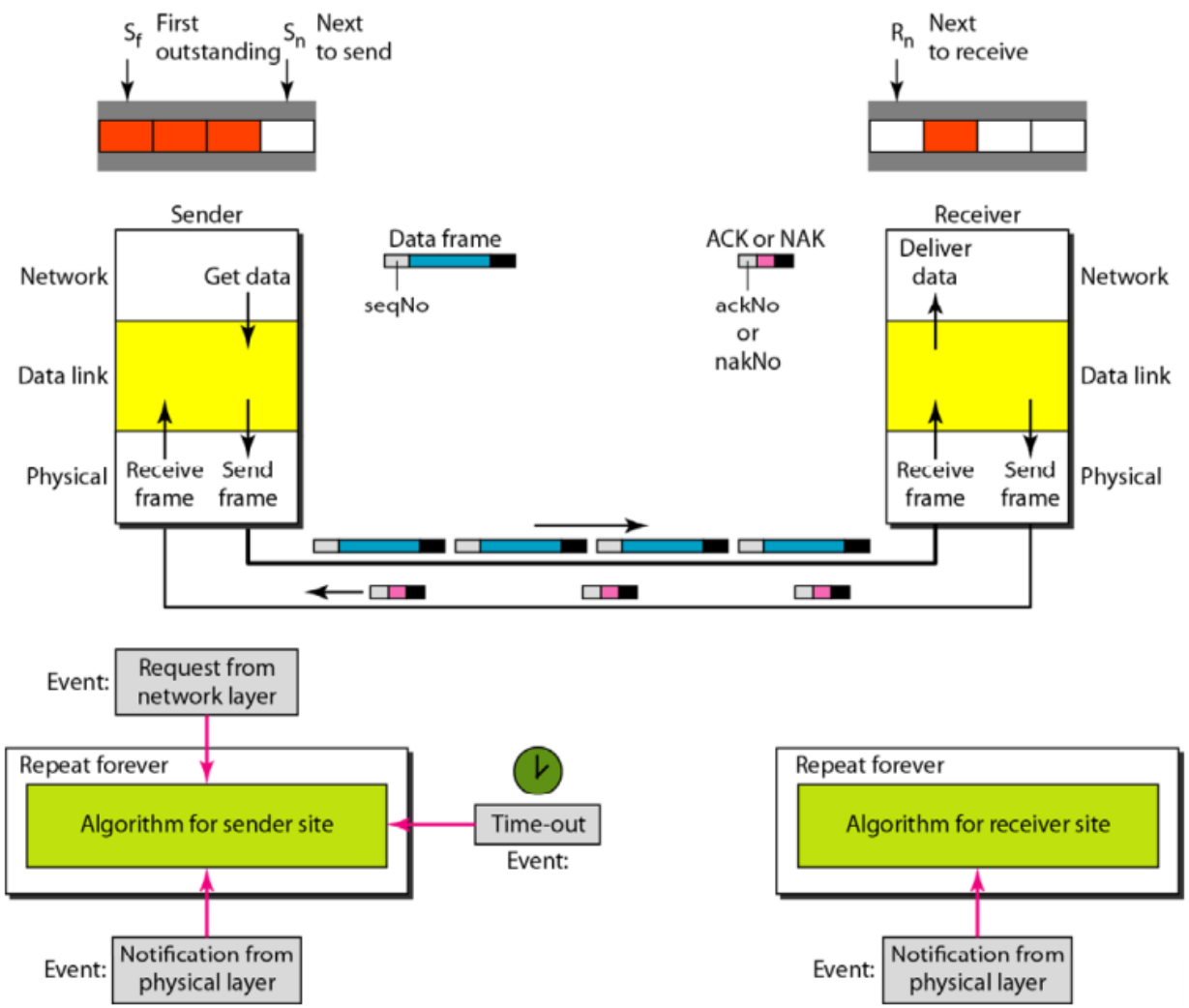


Go-Back N Protoco

Selective Repeat Protocol

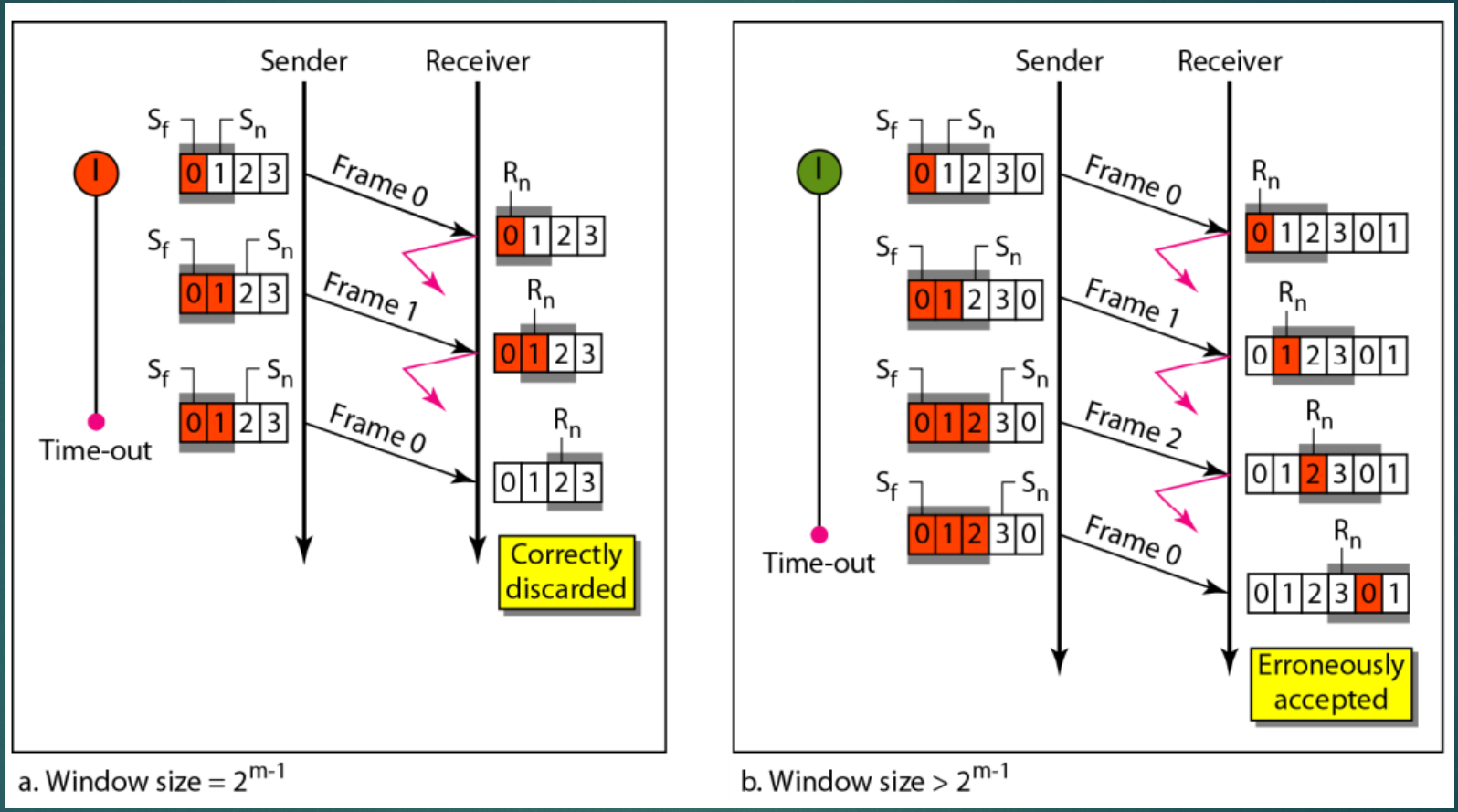






Selectiv Repeat Protoco

Selective Repeat Protocol

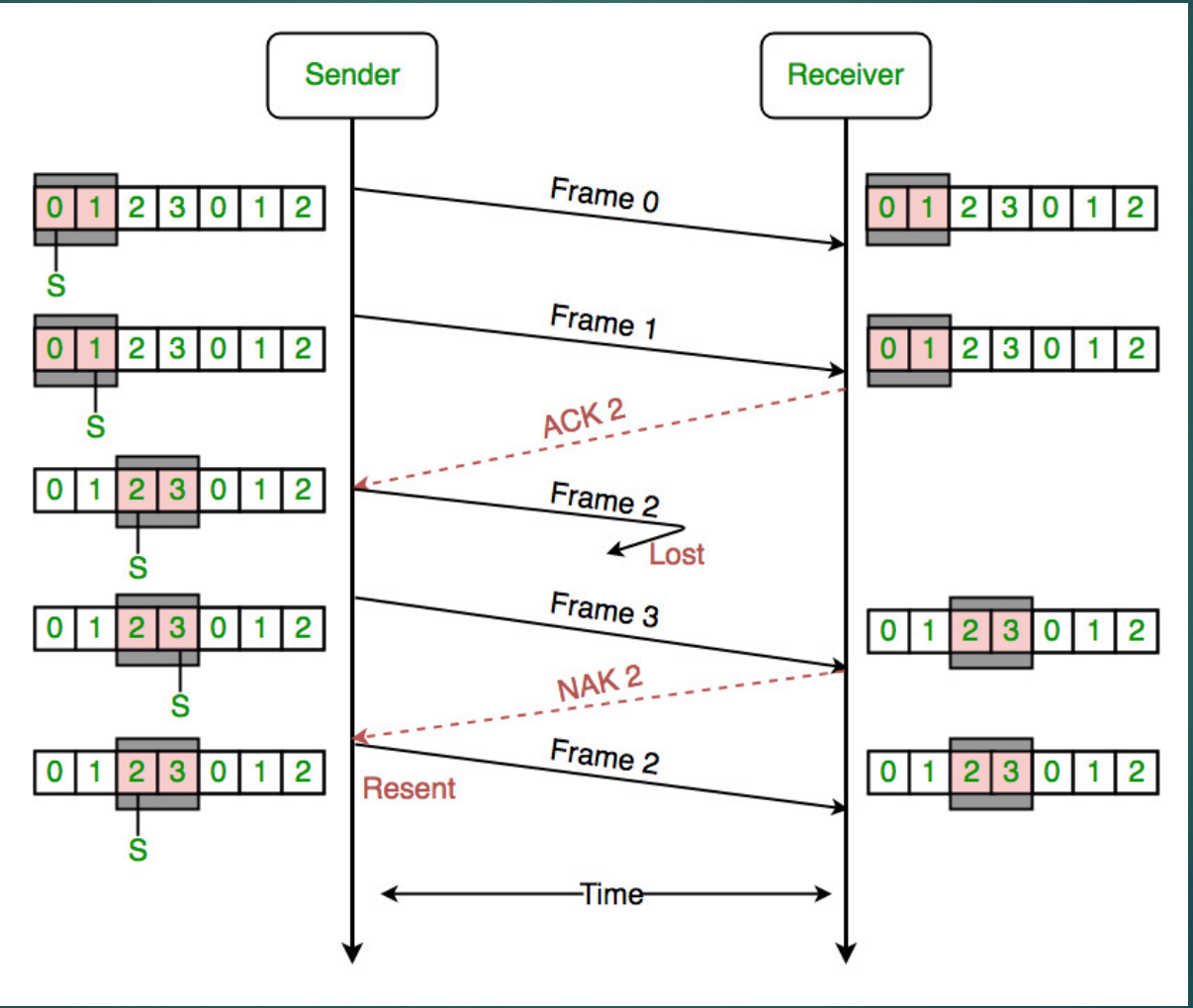


a. Window size = 2^{m-1}

b. Window size > 2^{m-1}

In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.

Selective Repeat



Selective Repeat Protocol

```
/* Protocol 6 (Selective repeat) 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 protocol 5, 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 */
}
```

```
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;
```

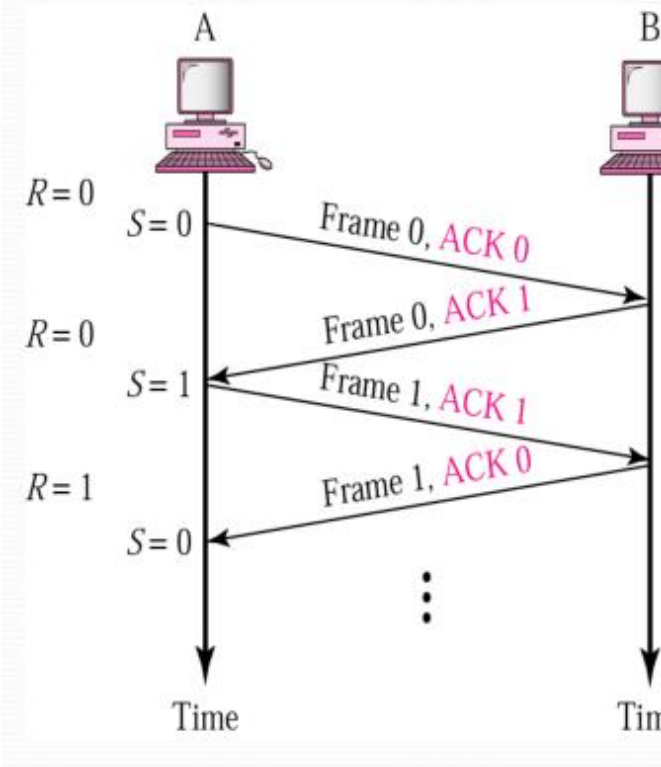
```
enable_network_layer();                          /* initialize */
ack_expected = 0;                                /* next ack expected on the inbound stream */
next_frame_to_send = 0;                          /* number of next outgoing frame */
frame_expected = 0;
too_far = NR_BUFS;
nbuffered =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

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 */
            }
        }
    }
```

```
        if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to
            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();
 }
}
```

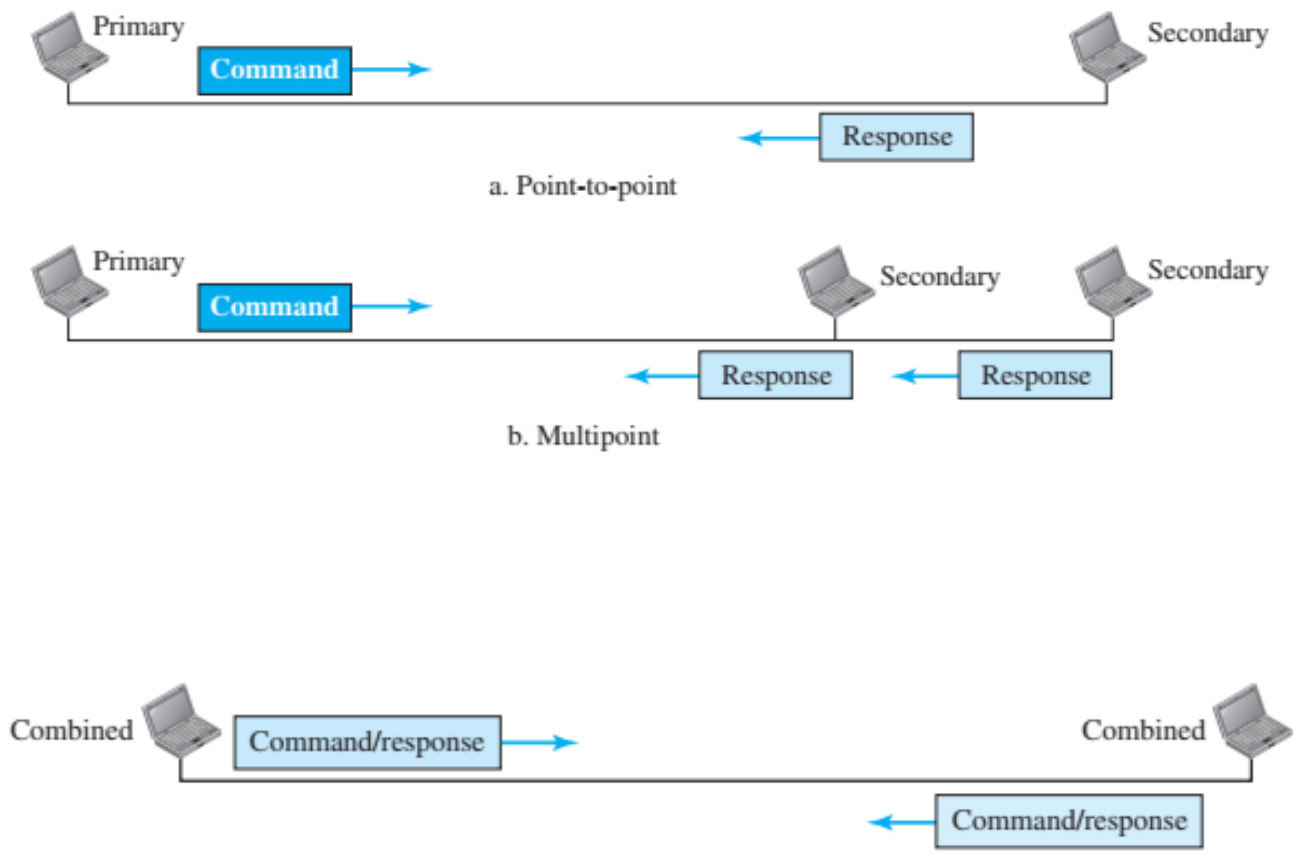
Piggybacking

- In a reliable communication data travels from one system to another, i.e., sender to receiver, and ack travels in the reverse.
- Protocols have been designed in the past to allow data to flow in both directions.
- However, to make the communication more efficient, the data in one direction is piggybacked with the acknowledgment in the other direction.
- In other words, when node A is sending data to node B, Node A also acknowledges the data received from node B.



High-level Data Link Control (HDLC)

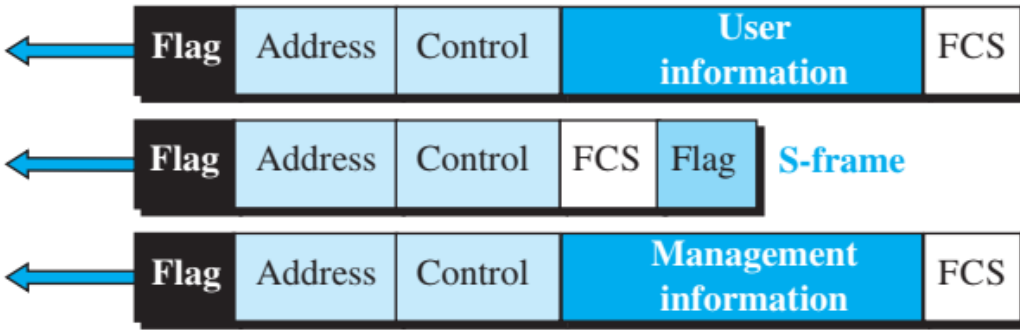
- A bit-oriented protocol for communication over point-to-point and multipoint links.
- Implements the Stop-and-Wait protocol.
- HDLC provides two common transfer modes that can be used in different configurations
  - Normal response mode (NRM)
    - The station configuration is unbalanced, i.e., one primary and multiple secondary.
    - Primary sends commands, secondary responds.
    - Used for both point-to-point and multipoint links.
  - Asynchronous balanced mode (ABM)
    - The configuration is balanced.
    - Used in point-to-point, where each station can function as a primary and a secondary, i.e., peer-to-peer



High-lev
Data Lin
Control
(HDLC)

HDLC: Framing

- To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames:
  - information frames (I-frames)
    - user data and control information relating to user data (piggybacking
  - supervisory frames (S-frames)
    - control information
  - unnumbered frames (U-frames)
    - system management, i.e., link management

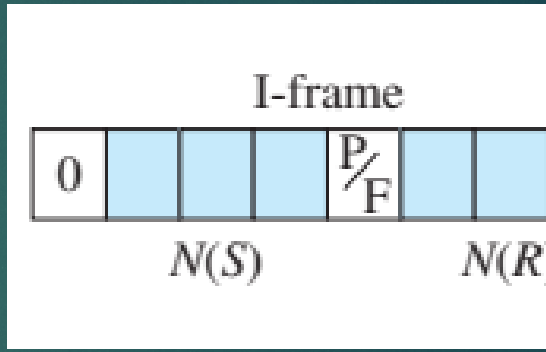


HDLC: Framing

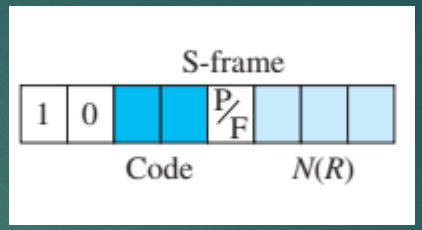
- Flag field: Contains synchronization pattern 01111110.
- Address field: Address of the secondary station.
- Control field: One or two bytes used for flow and error control.
- Information field: user’s data from the network layer or management information.
- FCS field: error detection field that contain either a 2- or 4-byte CRC.

HDLC: Framing-Control Field

- First 1(I-Frame) or 2(S & U Frames) bits tells what type of frame it is.
- I-Frame
  - User data from the network layer, flow & error-control information(PGB)
  - The 3 bits after type field, called N(S), define the sequence number.
  - P/F is set to 1 when the frame is a poll or final.
  - The 3 bits after P/F field, called N(R), define the acknowledge number.



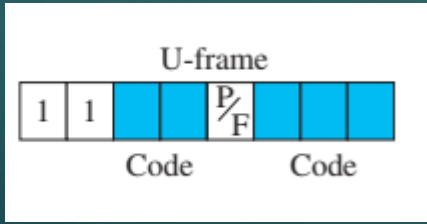
HDLC: Framing-Control Field



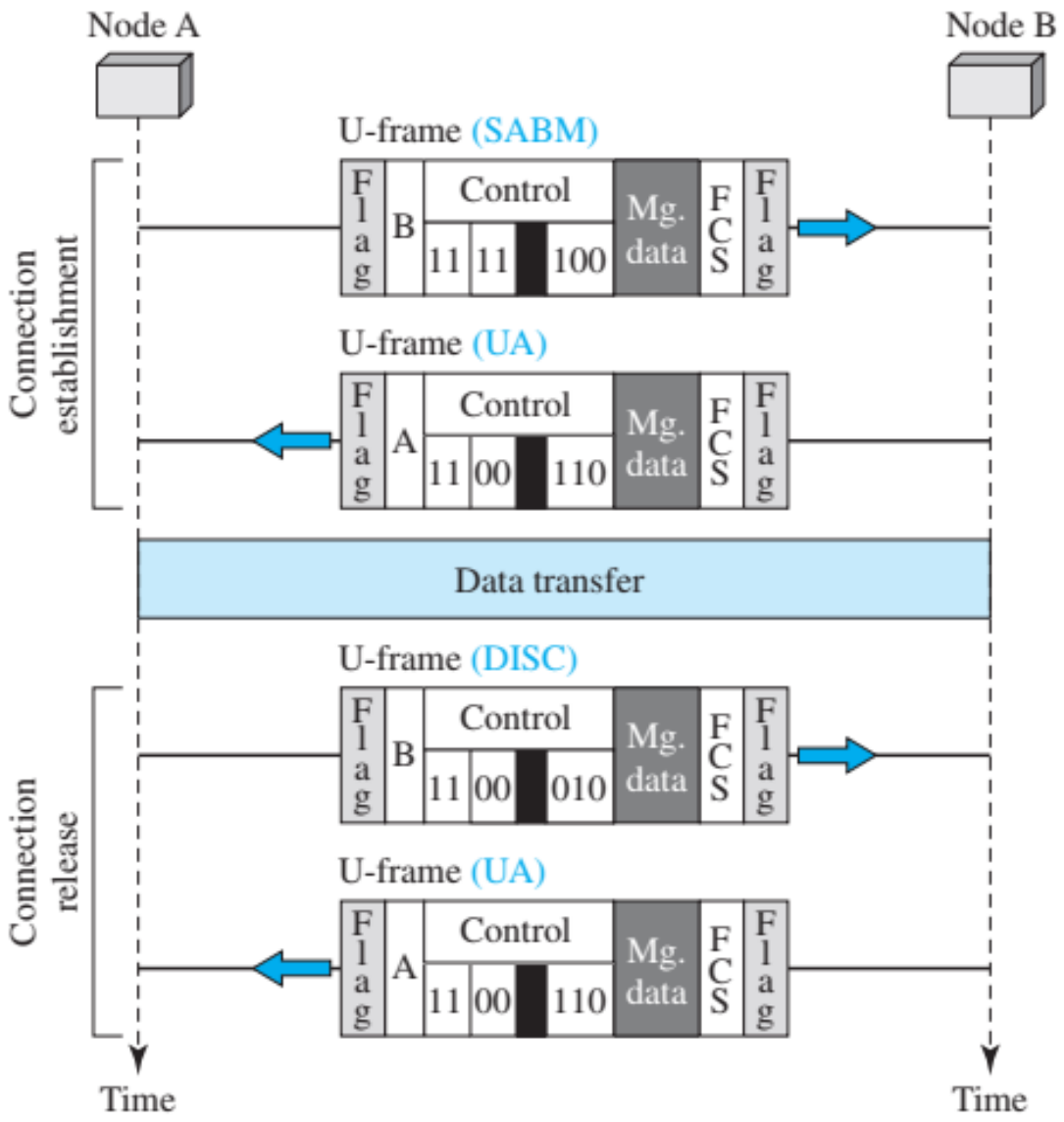
- S-Frame
  - Flow & error control when piggybacking is impossible or inappropriate.
  - The last 3 bits, called N(R), correspond to the ACK or NACK number.
  - 2 bits called code are used to define the type of S-frame
    - 00: Receive ready (RR), used for telling that receiver received the frames and it is ready for next frame.
    - 10: Receive not ready (RNR), used for telling that receiver received the frames but it is not ready for next frame.
    - 01: Reject (REJ), It is a NAK that can be used in Go-Back-N ARQ to improve the efficiency of the process by informing the sender
    - 11: Selective reject (SREJ), is a NAK frame used in Selective Repeat ARQ

HDLC: Framing-Control Field

- U-Frame
  - Used to exchange session management and control information between connected devices.
  - U-frame codes are divided into two sections:
    - a 2-bit prefix before the P/F bit
    - a 3-bit suffix after the P/F bit



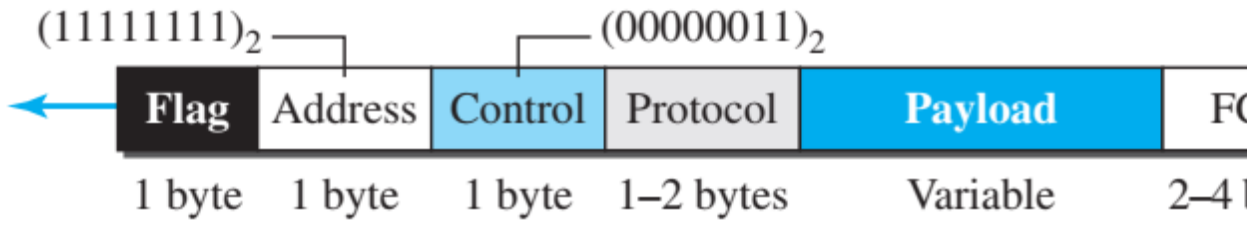
Code	*Command*	*Response*	*Meaning*
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	**DM**	Set asynchronous balanced mode or **d**
11 110	SABME		Set asynchronous balanced mode, ex
00 000	UI	**UI**	Unnumbered information
00 110		**UA**	**Unnumbered acknowledgment**
00 010	DISC	**RD**	Disconnect or **request disconnect**
10 000	SIM	**RIM**	Set initialization mode or **request inf**
00 100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	**XID**	Exchange ID
10 001	FRMR	**FRMR**	Frame reject


Node A
Node B
U-frame (SABM)
Flag
B
Control
11 11 100
Mg. data
FCS
Flag
U-frame (UA)
Flag
A
Control
11 00 110
Mg. data
FCS
Flag
Connection establishment
Data transfer
U-frame (DISC)
Flag
B
Control
11 00 010
Mg. data
FCS
Flag
U-frame (UA)
Flag
A
Control
11 00 110
Mg. data
FCS
Flag
Connection release
Time
Time

Point-to-Point Protocol (PPP)

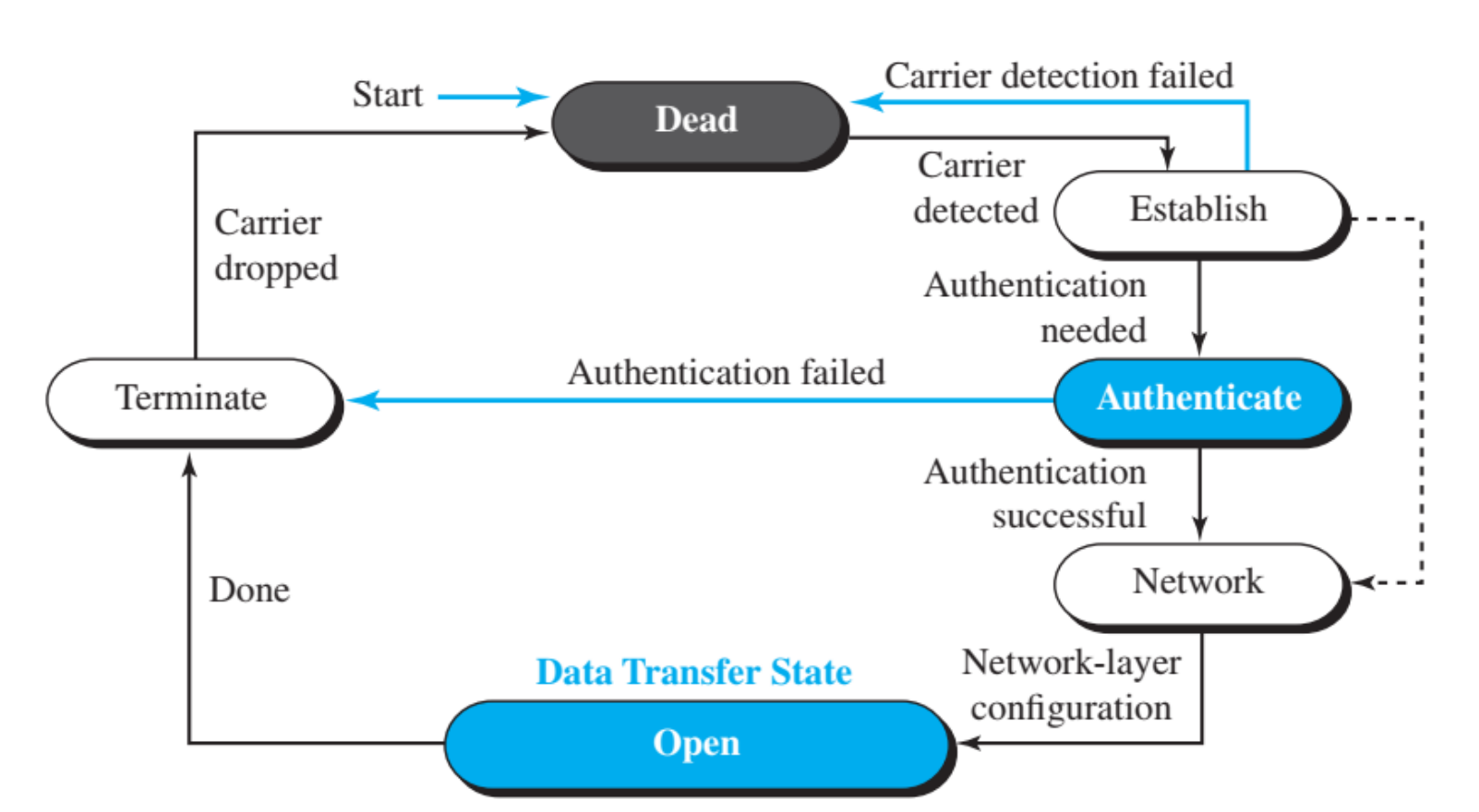
- Today, millions of Internet users who need to connect their home computers to the server of an Internet service provider use PPP.
- Is a byte-oriented protocol
- Services Provided by PPP
  - Defines the format of the frame to be exchanged between devices.
  - Defines how two devices can negotiate the establishment of the link and the exchange of data.
  - Authentication is also provided in the protocol, but it is optional.
- Services Not Provided by PPP
  - Does not provide flow control.
  - If the frame is corrupted, it is silently discarded. Upper layers has to take care of it.
  - Lack of error control and sequence numbering may cause a packet to be received out of order.
  - Does not provide a sophisticated addressing mechanism to handle frames in a multipoint configuration

PPP-Framing



- Flag. A PPP frame starts and ends with a 1-byte flag wit bit pattern 01111110.
- Address. The address field in this protocol is a constant and set to 11111111.
- Control. This field is set to the constant value 00000011
- Protocol. The protocol field defines what is being carrie data field: either user data or other information.
- Payload field. This field carries either the user data or c information.
- Payload field. This field carries either the user data or c information.

PPP-Transition Phase



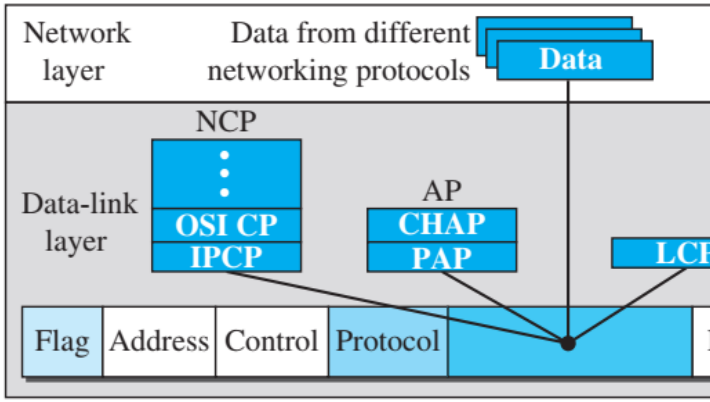
PPP-Multiplexing Protocols

- Although PPP is a link-layer protocol, it uses another se protocols to establish the link, authenticate the parties involved, and carry the network-layer data.
- Three sets of protocols are defined to make PPP power
  - Link Control Protocol (LCP)
  - Two Authentication Protocols : CHAP, PAP
  - Several Network Control Protocols (NCPs): OSICP,IP

Legend

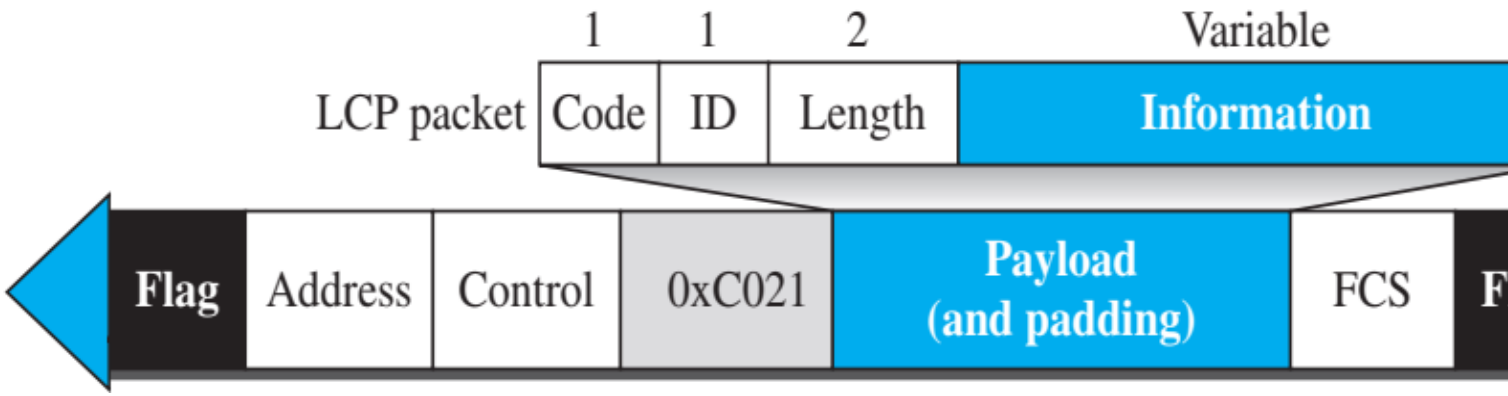
LCP : Link control protocol
AP: Authentication protocol
NCP: Network control protocol

Protocol values:
LCP : 0xC021
AP: 0xC023 and 0xC223
NCP: 0x8021 and ....
Data: 0x0021 and ....



PPP-LCP

- Responsible for establishing, maintaining, configuring, and terminating links.
- Responsible for establishing, maintaining, configuring, and terminating links.



PPP-LCP

- The ID field holds a value that matches a request with a reply
- The length field defines the length of the entire LCP packet.
- The length field defines the length of the entire LCP packet. .

Code	Packet Type	Description
0x01	Configure-request	Contains the list of proposed options and their
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptabl
0x04	Configure-reject	Announces that some options are not recognize
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other er
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet

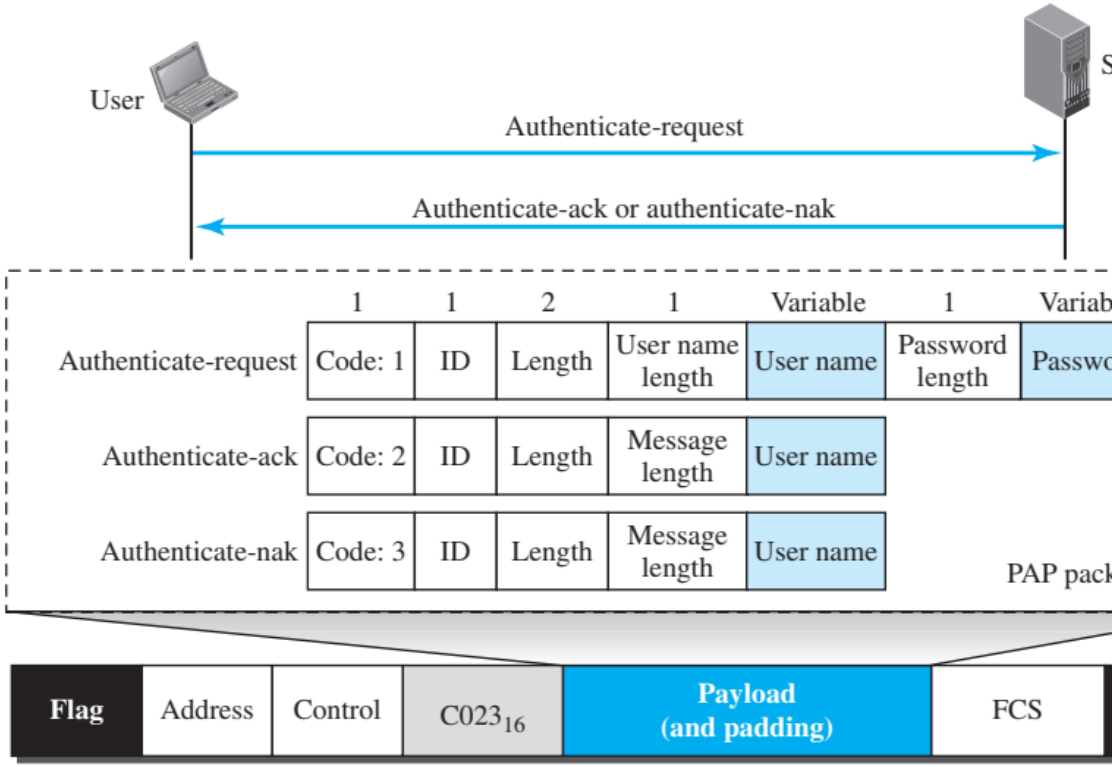
Option	*Default*
Maximum receive unit (payload field size)	1500
Authentication protocol	None
Protocol field compression	Off
Address and control field compression	Off

PPP-APs

- Authentication means validating the identity of a user who needs to access a set of resources.
- PPP has created two protocols for authentication
  - Password Authentication Protocol (PAP)
  - Challenge Handshake Authentication Protocol (CHAP)

PPP-PAP

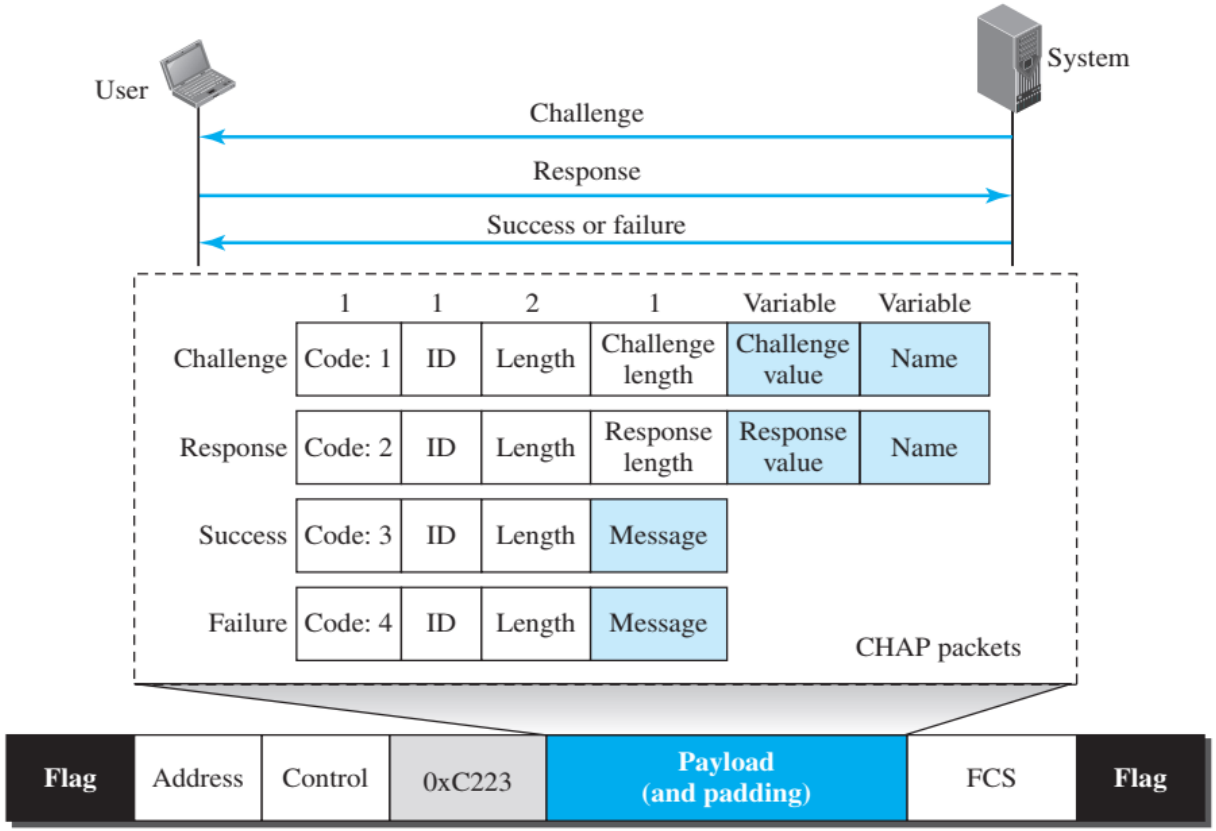
- A simple authentication procedure with a two-step process:
  - The user who wants to access a system sends an authentication identification(usually the username) and a password.
  - The system checks the validity of the identification and password and either accepts or denies connection

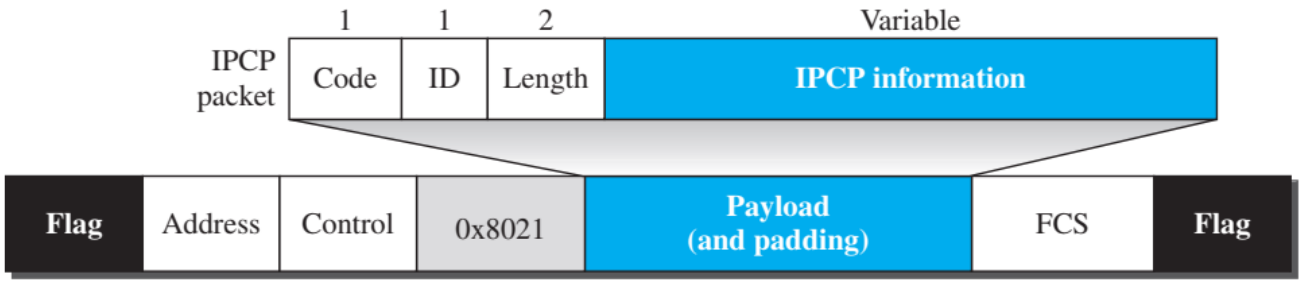


PPP-CHAP

- A three-way handshaking authentication protocol that provides greater security than PAP.
- In this method, the password is kept secret; it is never sent online.
- 3-way Handshaking
  - The system sends the user a challenge packet containing a challenge value, usually a few bytes.
  - The user applies a predefined function that takes the challenge value and the user's own password and creates a result. The user sends the result in the response packet to the system.
  - The system does the same. It applies the same function to the password of the user (known to the system) and the challenge value to create a result.
  - If the created and received is same sends ACK else NACK.

PPP-CHAP


User
System
Challenge
Response
Success or failure
1
1
2
1
Variable
Variable
Challenge
Code: 1
ID
Length
Challenge length
Challenge value
Name
Response
Code: 2
ID
Length
Response length
Response value
Name
Success
Code: 3
ID
Length
Message
Failure
Code: 4
ID
Length
Message
CHAP packets
Flag
Address
Control
0xC223
Payload (and padding)
FCS
Flag

PPP-NCP

- PPP is a multiple-network-layer protocol. It can carry a network-layer data packet from pro defined by the Internet, OSI, Xerox, DECnet, AppleTalk, Novel, and so on.
- To do this, PPP has defined a specific Network Control Protocol for each network protoc example, IPCP (Internet Protocol Control Protocol) configures the link for carrying IP data pa Xerox CP does the same for the Xerox protocol data packets, and so on.
- Note that none of the NCP packets carry network-layer data; they just configure the link network layer for the incoming data.