

# CSMC 417

## Computer Networks

### Prof. Ashok K Agrawala

© 2015 Ashok Agrawala  
Set 4

# The Data Link Layer

## Chapter 3

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

# The Data Link Layer

Responsible for delivering frames of information over a single link

- Handles transmission errors and regulates the flow of data

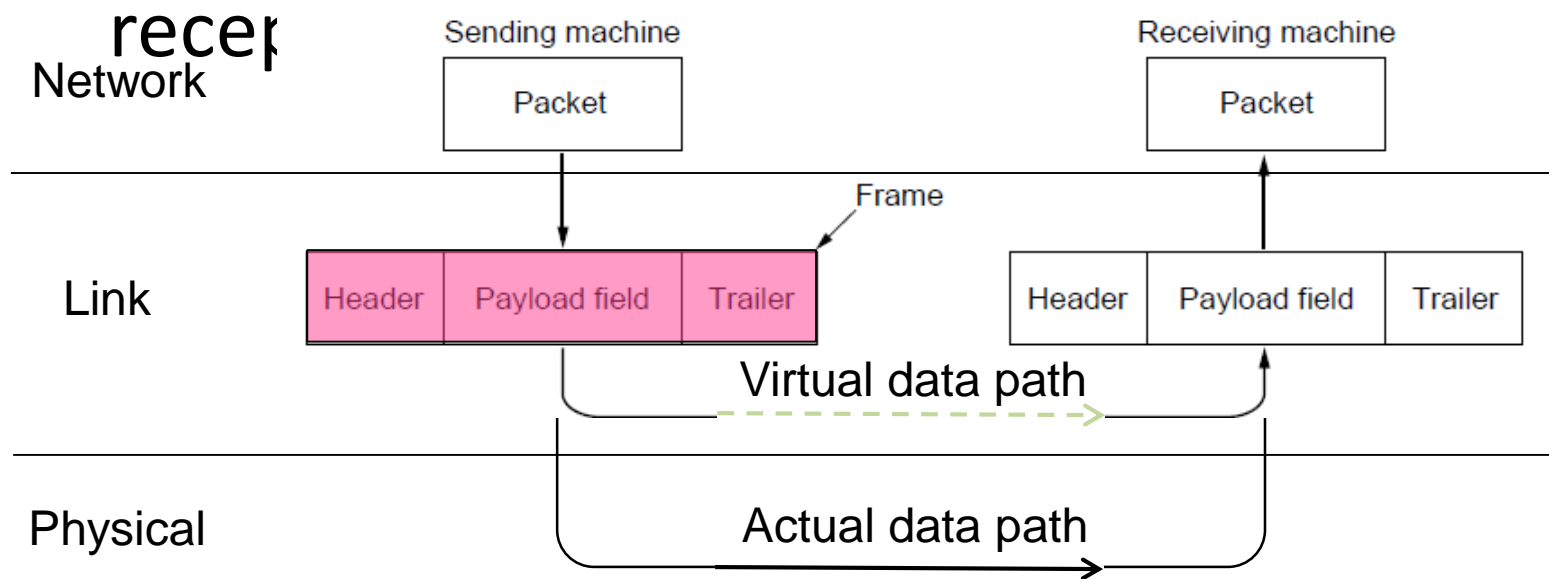
Application
Transport
Network
Link
Physical

# Data Link Layer Design Issues

- Frames »
- Possible services »
- Framing methods »
- Error control »
- Flow control »

# Frames

Link layer accepts packets from the network layer, and encapsulates them into frames that it sends using the physical layer;



# Functions of the Data Link Layer

- Provide service interface to the network layer
- Dealing with transmission errors
- Regulating data flow
  - Slow receivers not swamped by fast senders

# Possible Services

## Unacknowledged connectionless service

- Frame is sent with no connection / error recovery
- Ethernet is example

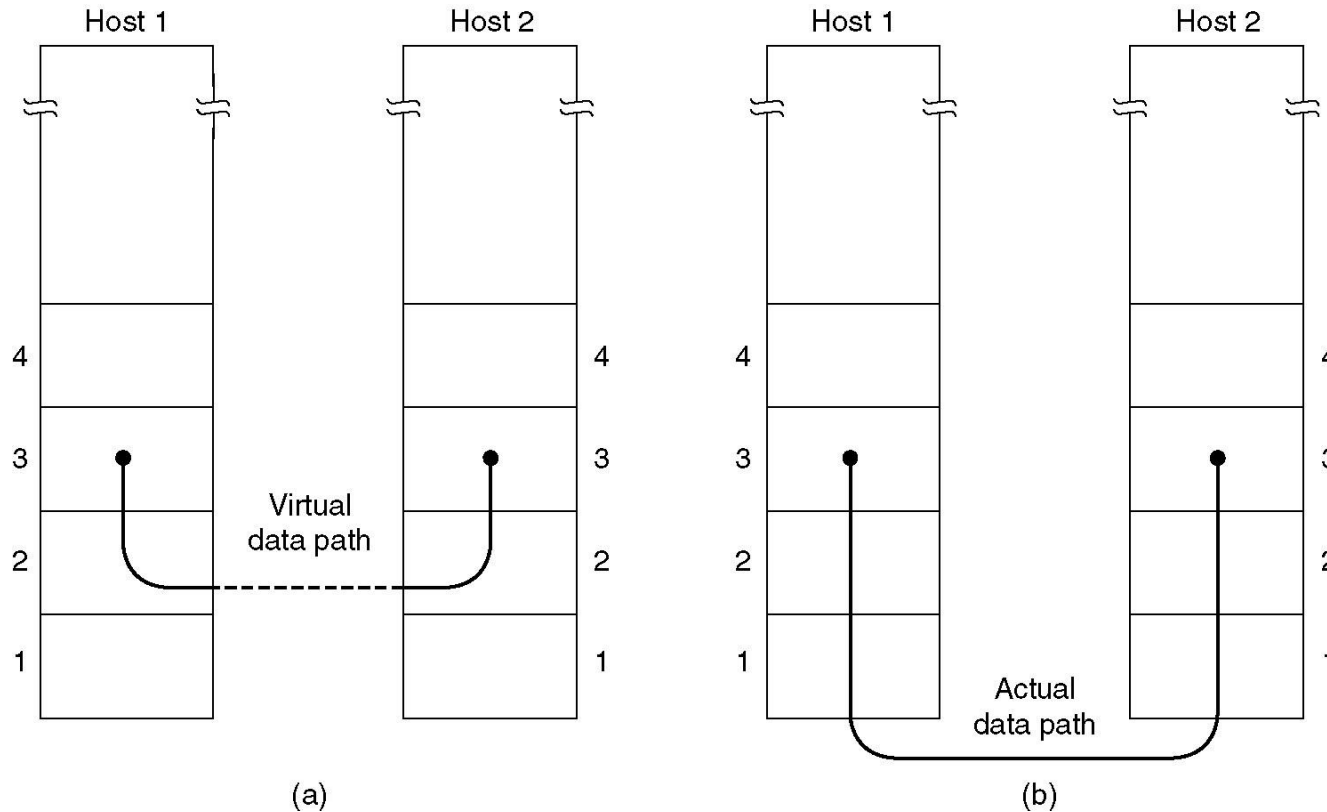
## Acknowledged connectionless service

- Frame is sent with retransmissions if needed
- Example is 802.11

## Acknowledged connection-oriented service

- Connection is set up; rare

# Services Provided to Network Layer

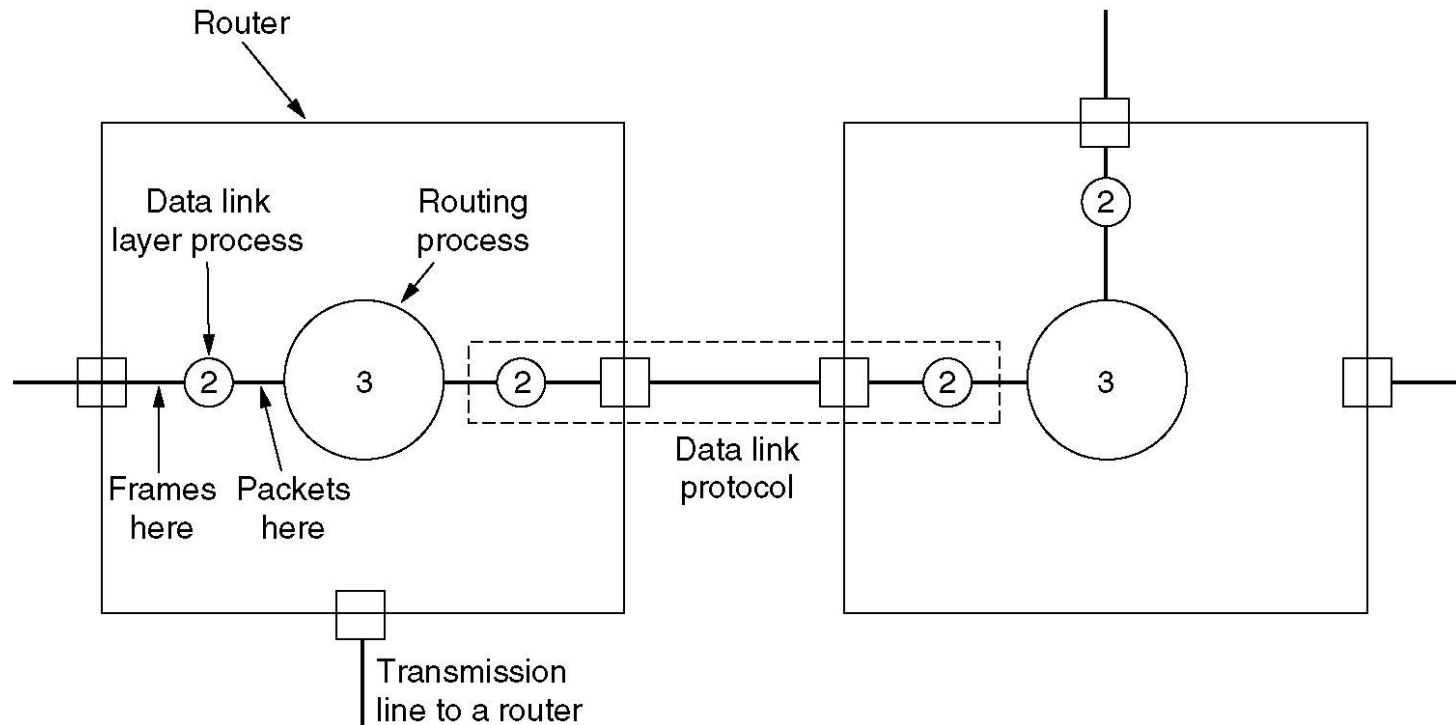


(a) Virtual communication.

(b) Actual communication.



# Services Provided to Network Layer (2)

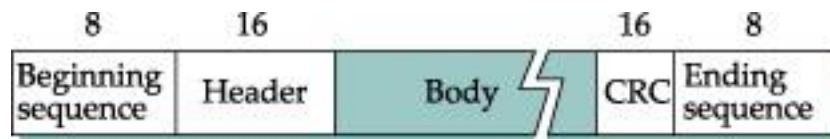


# Framing Methods

- Byte count »
- Flag bytes with byte stuffing »
- Flag bits with bit stuffing »
- Physical layer coding violations
  - Use non-data symbol to indicate frame

# Bit Oriented Protocols

- Frame – a collection of bits
  - No Byte boundary
- SDLC – Synchronous Data Link Control
  - IBM
- HDLC – High-Level Data Link Control
  - ISO Standard



HDLC Frame Format

# Framing

- Break sequence of bits into a frame
  - Typically implemented by the network adaptor
- Sentinel-based
  - Delineate frame with special pattern (e.g., 01111110)

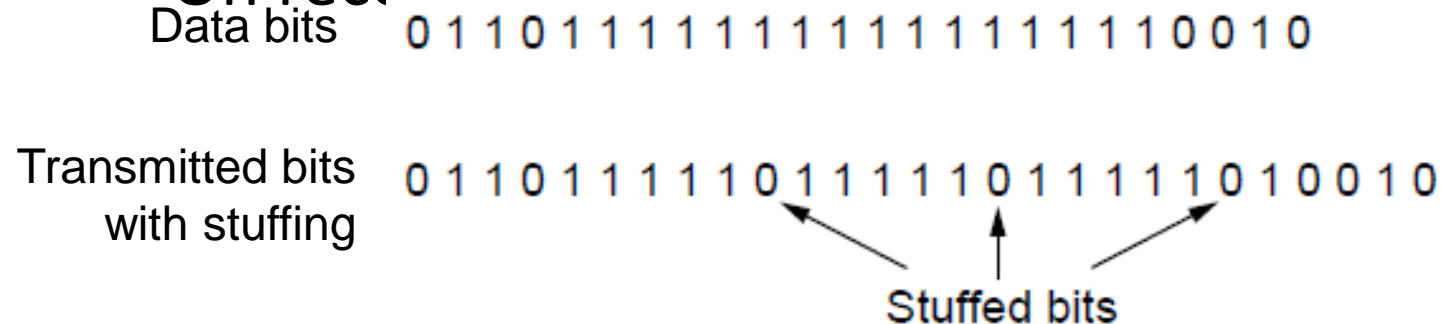


- Problem: what if special patterns occurs within frame?
- Solution: escaping the special characters
  - E.g., sender always inserts a 0 after five 1s
  - ... and receiver always removes a 0 appearing after five 1s
  - Bit Stuffing
- Similar to escaping special characters in C programs

# Framing – Bit stuffing

Stuffing done at the bit level:


- Frame flag has six consecutive 1s (not shown)
- On transmit, after five 1s in the data, a 0 is added
- On receive



# Framing – Bit Oriented

(a) 0 1 1 0 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

The diagram shows three arrows pointing to the stuffed bits in the bit stream (b). The first arrow points to the 10th bit (0), the second arrow points to the 15th bit (0), and the third arrow points to the 20th bit (0). The text 'Stuffed bits' is centered below the arrows.

(c) 0 1 1 0 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.

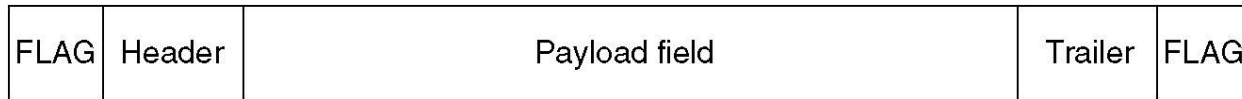
(b) The data as they appear on the line.

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

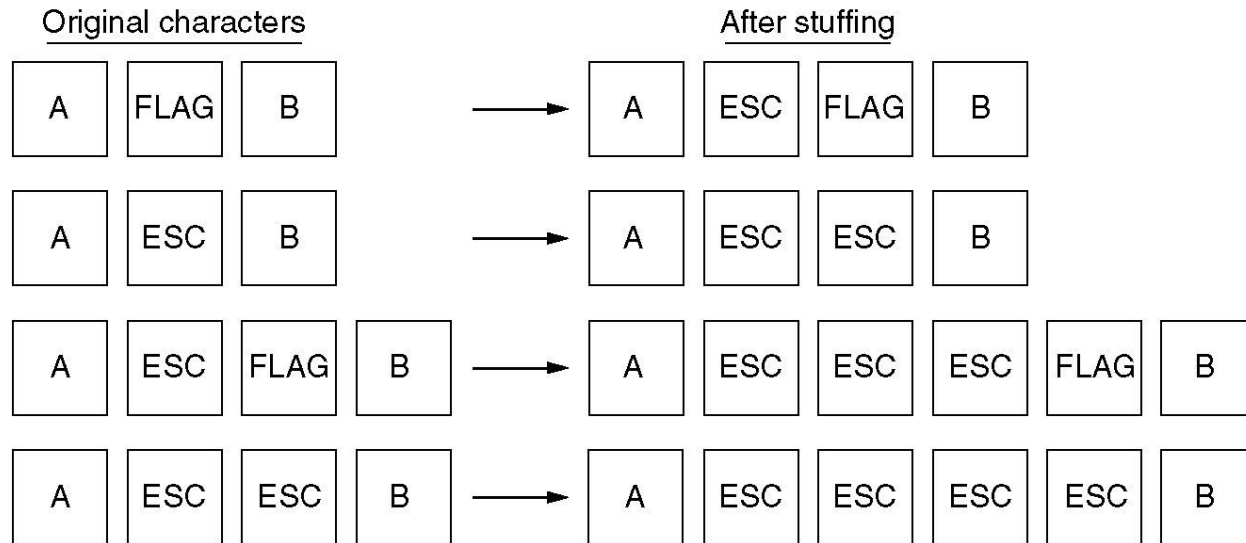
# Byte-Oriented Protocols

- Frame – a collection of bytes.
- Examples
  - BISYNC – Binary Synchronous Communication – IBM
  - DDCMP – Digital Data Communication Message Protocol
  - PPP – Point-to-Point
- Sentinel Based – Use special character as marker
  - BISYNC
    - SYN and SOH
    - STX and ETX
    - DLE as escape character. - Character Stuffing

# Framing



(a)



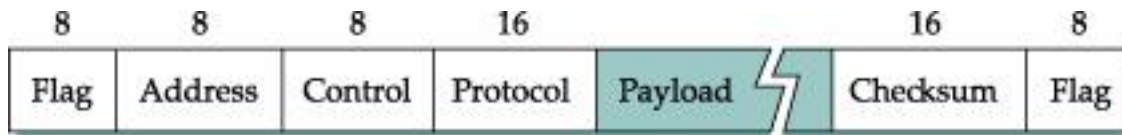
(b)

(a) A frame delimited by flag bytes.

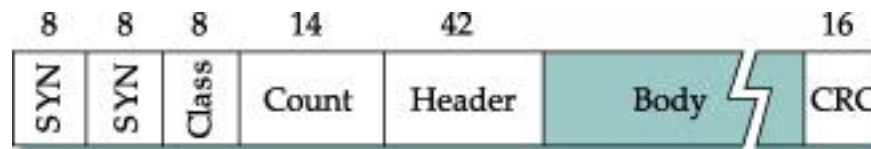
(b) Four examples of byte sequences before and after stuffing.



# Frame Structure



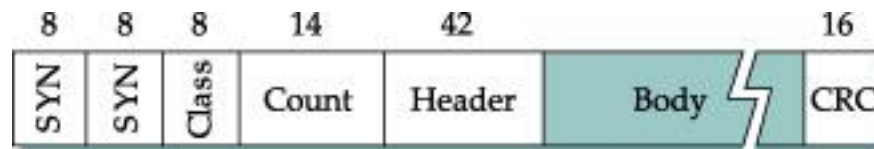
## PPP Frame Format



## BISYNC Frame Format

# Framing (Continued)

- Counter-based
  - Include the payload length in the header
  - ... instead of putting a sentinel at the end
  - Problem: what if the count field gets corrupted?
    - Causes receiver to think the frame ends at a different place
  - Solution: catch later when doing error detection
    - And wait for the next sentinel for the start of a new frame



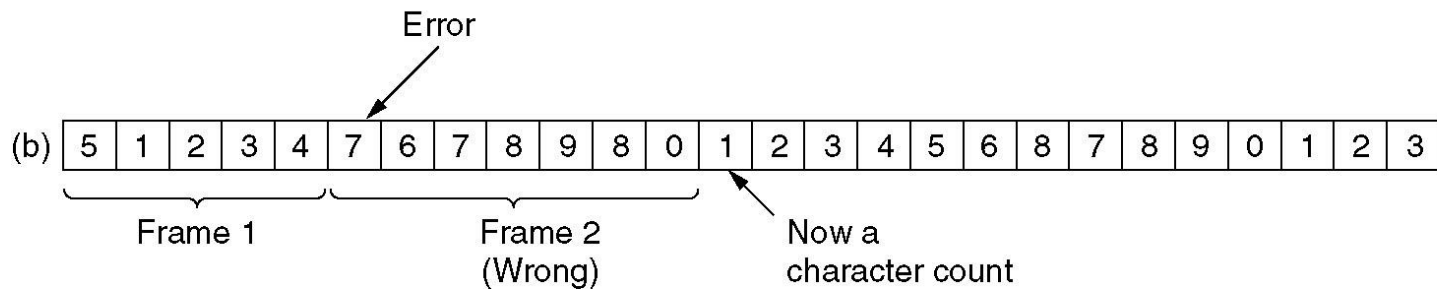
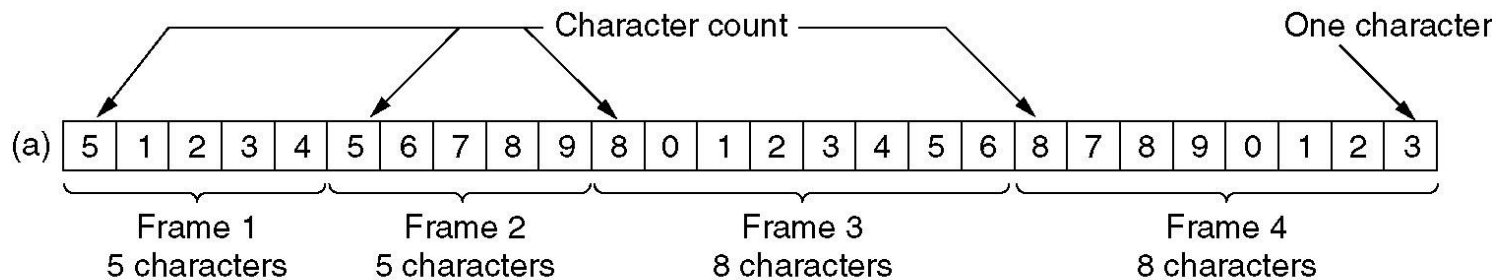
DDCMP Frame Format

# Framing

A character stream.

(a) Without errors.

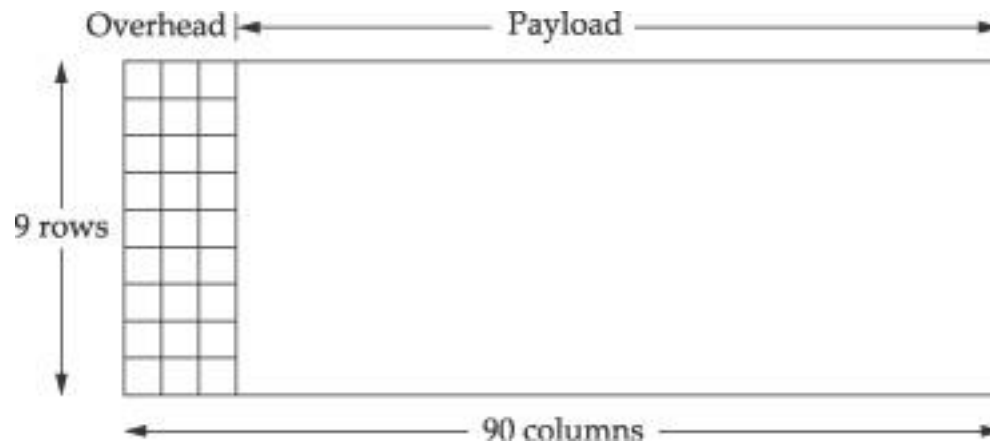
(b) With one error.



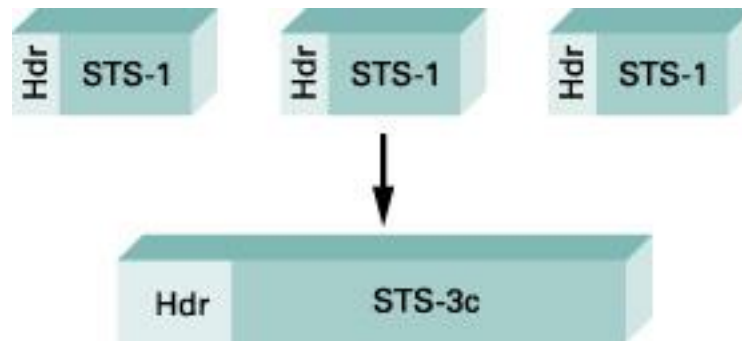
# Clock-Based Framing (SONET)

- Clock-based
  - Make each frame a fixed size
  - No ambiguity about start and end of frame
  - But, may be wasteful
- Synchronous Optical Network (SONET)
  - Slowest speed link STS-1 – 51.84 Mbps (  $810 \times 8 \times 8K$  )
  - Frame – 9 rows of 90 bytes
    - First 3 bytes of each row are overhead
    - First two bytes of a frame contain a special bit pattern – to mark the start of the frame – check for it every 810 bytes

# Sonet Frame



# Three STS-1 frames to one STS-3 frame



# Error Control

Error control repairs frames that are received in error

- Requires errors to be detected at the receiver
- Typically retransmit the unacknowledged frames
- Timer protects against lost acknowledgements

Detecting errors and retransmissions are next topics.

# Flow Control

Prevents a fast sender from out-pacing a slow receiver

- Receiver gives feedback on the data it can accept
- Rare in the Link layer as NICs run at “wire speed”
  - Receiver can take data as fast as it can be sent

Flow control is a topic in the Link and Transport layers.



# Error Detection and Correction

Error codes add structured redundancy to data so errors can be either detected, or corrected.

Error correction codes:

- Hamming codes »
- Binary convolutional codes »
- Reed-Solomon and Low-Density Parity Check codes
  - Mathematically complex, widely used in real systems

Error detection codes:

- Parity »
- Checksums »
- Cyclic redundancy codes »

# Error Detection

- Errors are unavoidable
  - Electrical interference, thermal noise, etc.
- Error detection
  - Transmit extra (redundant) information
  - Use redundant information to detect errors
  - Extreme case: send two copies of the data
  - Trade-off: accuracy vs. overhead
- Techniques for detecting errors
  - Parity checking
  - Checksum
  - Cyclic Redundancy Check (CRC)

# Error Detection Techniques

- Parity check
  - Add an extra bit to a 7-bit code
  - Odd parity: ensure an odd number of 1s
    - E.g., 0101011 becomes 0101011<sup>1</sup>
  - Even parity: ensure an even number of 1s
    - E.g., 0101011 becomes 0101011<sup>0</sup>
- Two Dimensional Parity

# Error Bounds – Hamming distance

Code turns data of  $n$  bits into codewords of  $n+k$  bits

Hamming distance is the minimum bit flips to turn one valid codeword into any other valid one.

- Example with 4 codewords of 10 bits ( $n=2$ ,  $k=8$ ):
  - 0000000000, 0000011111, 1111100000, and 1111111111
  - Hamming distance is 5

Bounds for a code with distance:

- $2d+1$  – can correct  $d$  errors (e.g., 2 errors above)
- $d+1$  – can detect  $d$  errors (e.g., 4 errors above)

# Error Detection – Parity (1)

Parity bit is added as the modulo 2 sum of data bits

- Equivalent to XOR; this is even parity
- Ex: 1110000  $\rightarrow$  1110000<sup>1</sup>
- Detection checks if the sum is wrong (an error)

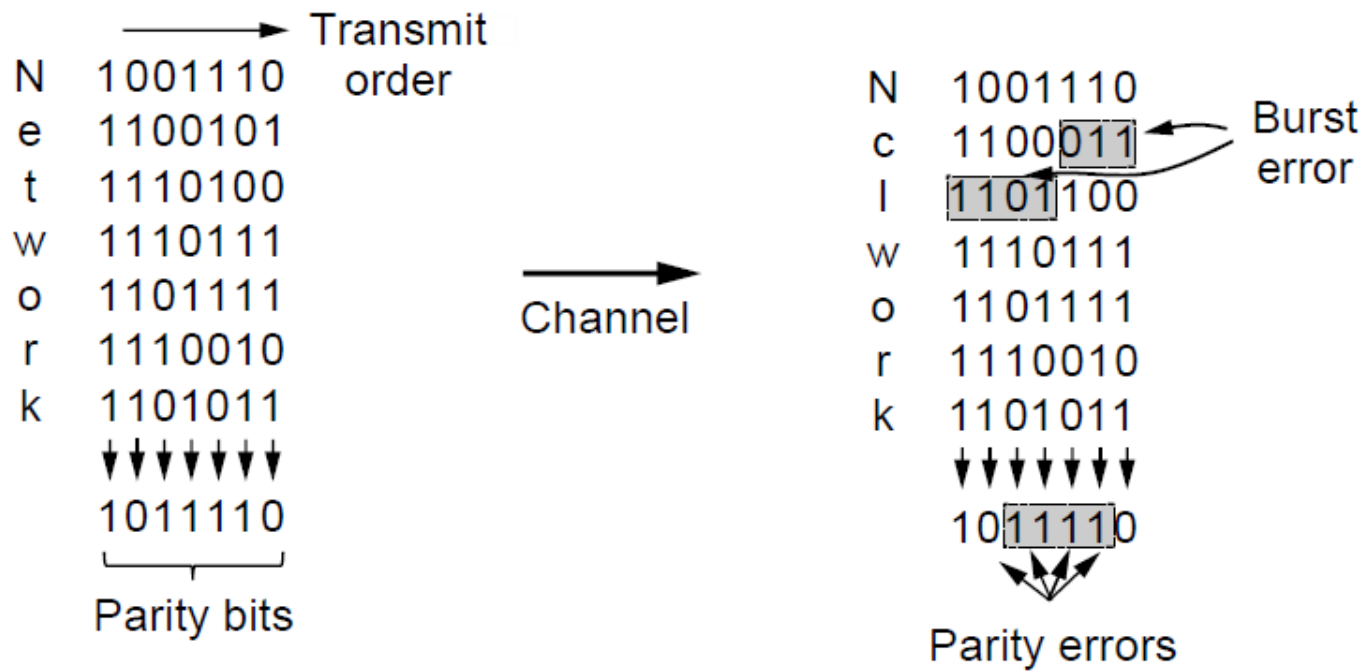
Simple way to detect an *odd* number of errors

- Ex: 1 error, 1110010<sup>1</sup>; detected, sum is wrong
- Ex: 3 errors, 1101100<sup>1</sup>; detected sum is wrong
- Ex: 2 errors, 1110110<sup>1</sup>; *not detected*, sum is right!
- Error can also be in the parity bit itself
- Random errors are detected with probability  $\frac{1}{2}$

# Error Detection – Parity (2)

Interleaving of N parity bits detects burst errors up to N

- Each parity sum is made over non-adjacent bits
- An even burst of up to N errors will not cause it



# Two Dimensional Parity

		Parity bits
Data	0101001	1
	1101001	0
	1011110	1
	0001110	1
	0110100	1
	1011111	0
Parity byte	1111011	0

# Error Detection – Checksums

Checksum treats data as  $N$ -bit words and adds  $N$  check bits that are the modulo  $2^N$  sum of the words

- Ex: Internet 16-bit 1s complement checksum

## Properties:

- Improved error detection over parity bits
- Detects bursts up to  $N$  errors
- Detects random errors with probability  $1-2^{-N}$
- Vulnerable to systematic errors, e.g., added zeros



# Checksum

- Checksum
  - Treat data as a sequence of 16-bit words
  - Compute a sum of all the 16-bit words, with no carries
  - Transmit the sum along with the packet

# Internet Checksum Algorithm

- Consider data as a sequence of 16-bit integers
- Add them together using 16-bit one's complement arithmetic
- Take 1's complement of the sum
- That is the checksum

# Cyclic Redundancy Check

- Have to maximize the probability of detecting the errors using a small number of additional bits.
- Based on powerful mathematical formulations – theory of finite fields
- Consider  $(n+1)$  bits as  $n$  degree polynomial
- Message  $M(x)$  represented as polynomial
- Divisor  $C(x)$  of degree  $k$
- Send  $P(x)$  as  $(n+1)$  bits +  $k$  bits such that  $P(x)$  is exactly divisible by  $C(x)$

$$C(x) = x^3 + x^2 + 1$$

$$M(x) = x^7 + x^4 + x^3 + x^1$$

# CRC Basis

- Use modulo 2 arithmetic
- Any Polynomial  $B(x)$  can be divided by a divisor polynomial  $C(x)$  if  $B(x)$  is of higher degree than  $C(x)$
- Any polynomial  $B(x)$  can be divided once by a divisor polynomial  $C(x)$  if they are of the same degree
- The remainder obtained when  $B(x)$  is divided by  $C(x)$  is obtained by subtracting  $C(x)$  from  $B(x)$
- To subtract  $C(x)$  from  $B(x)$  we simply perform the exclusive-OR operation on each pair of matching coefficients.

# CRC Basis

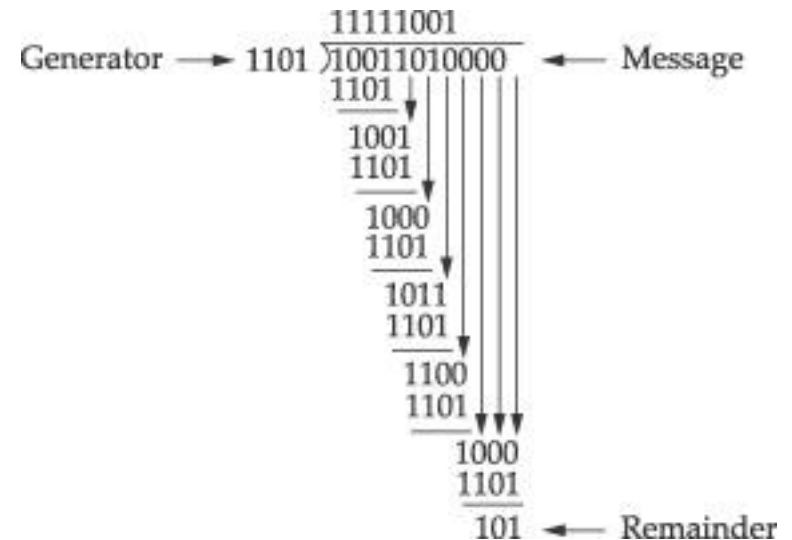
1. Multiply  $M(x)$  by  $x^k$ , i.e. add  $k$  zeros at the end of the message.  
Call this  $T(x)$

2. Divide  $T(x)$  by  $C(x)$

3. Subtract the remainder

from  $T(x)$

- Message sent –  
1001101010 101



# Cyclic Redundancy Check

- All single bit errors – if  $x^k$  and  $x^0$  terms are nonzero
- All double-bit errors – as long as  $C(x)$  has a factor with at least three terms
- Any odd number of errors as long as  $C(x)$  has  $(x+1)$  as a factor
- Any burst error of length  $k$  bits

# Common CRC Polynomials

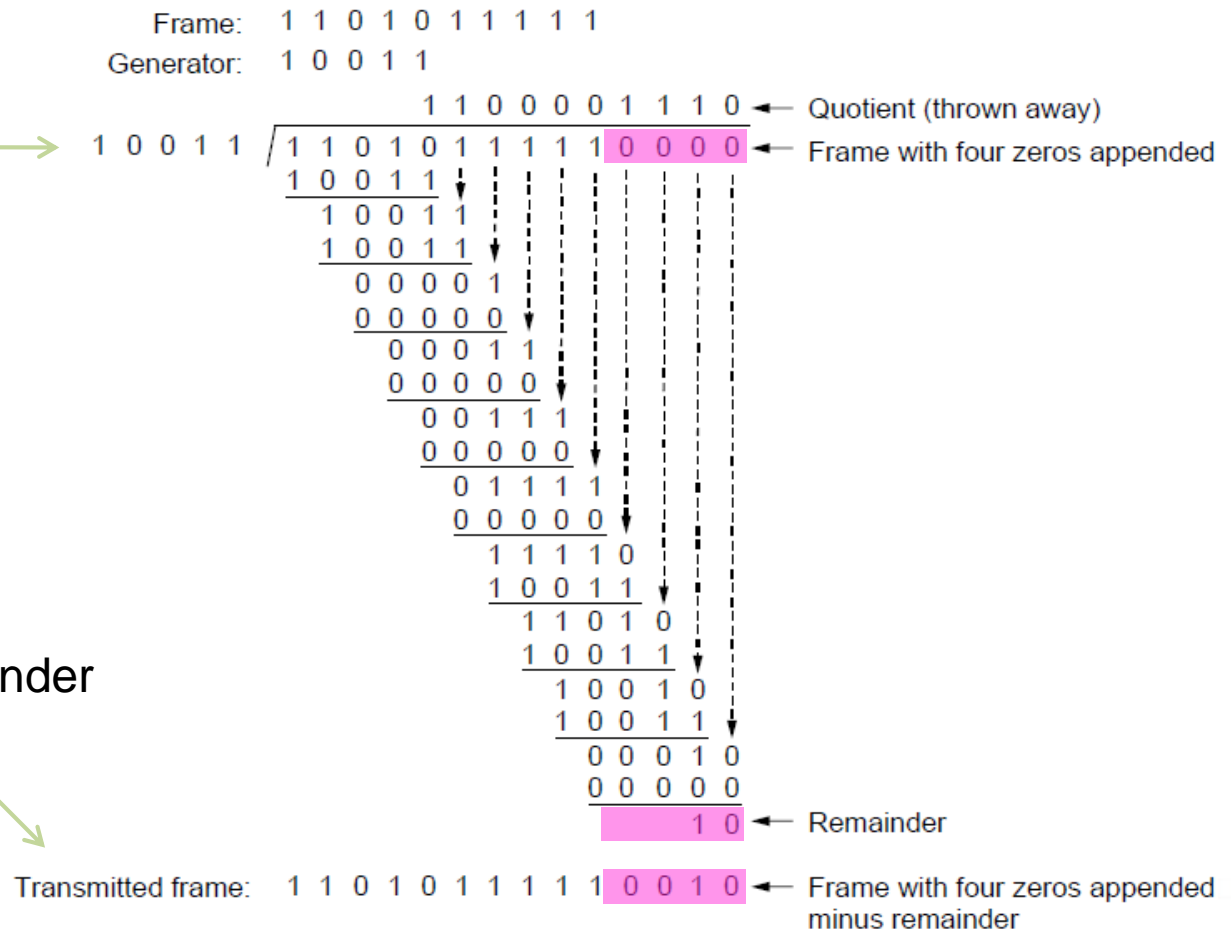
CRC	$C(x)$
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

# Error Detection – CRCs (1)

- Adds bits so that transmitted frame viewed as a polynomial is evenly divisible

Start by adding  
0s to frame  
and try dividing

Offset by any remainder  
to make it evenly  
divisible





# Error Detection – CRCs (2)

Based on standard polynomials:

- Ex: Ethernet 32-bit CRC is defined by:

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

- Computed with simple shift/XOR circuits

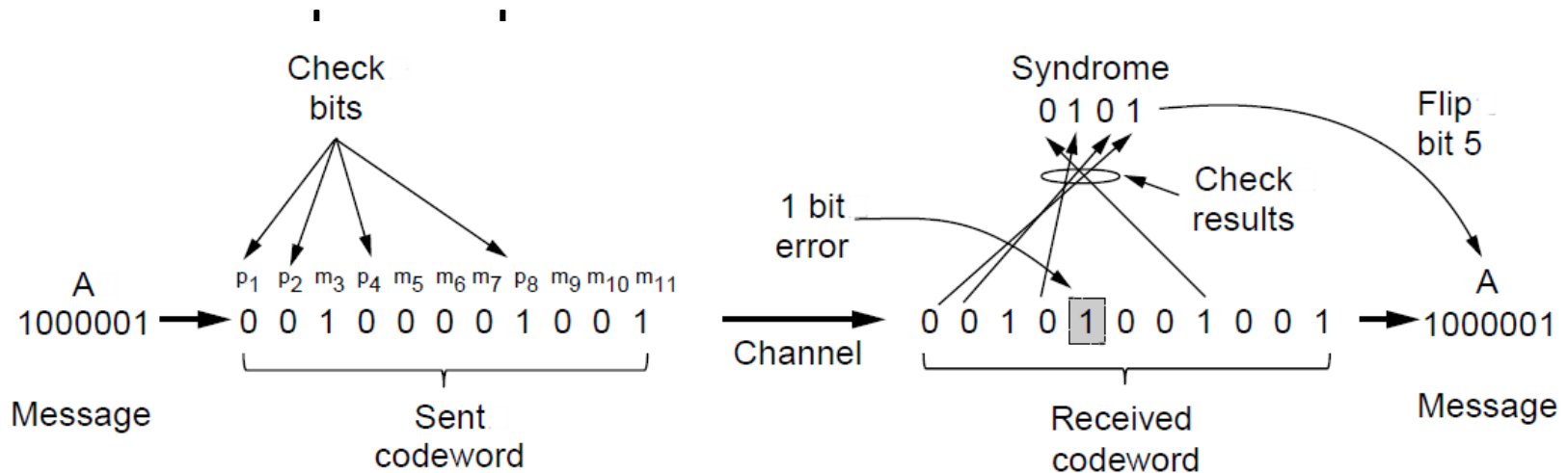
Stronger detection than checksums:

- E.g., can detect all double bit errors
- Not vulnerable to systematic errors

# Error Correction – Hamming code

Hamming code gives a simple way to add check bits and correct up to a single bit error:

- Check bits are parity over subsets of the



(11, 7) Hamming code adds 4 check bits and can correct 1 error

# Error-Correcting Codes

Use of a

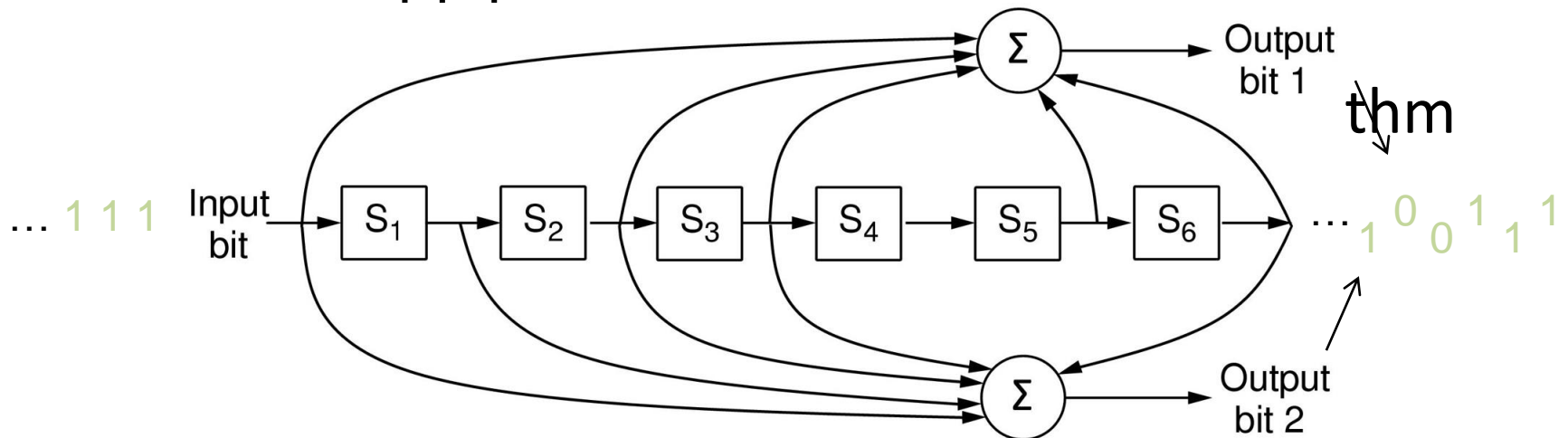
Char.	ASCII	Check bits	errors.
H	1001000	00110010000	
a	1100001	10111001001	
m	1101101	11101010101	
m	1101101	11101010101	
i	1101001	01101011001	
n	1101110	01101010110	
g	1100111	01111001111	
	0100000	10011000000	
c	1100011	11111000011	
o	1101111	10101011111	
d	1100100	11111001100	
e	1100101	00111000101	

Order of bit transmission

# Error Correction – Convolutional codes

Operates on a stream of bits, keeping internal state

- Output stream is a function of all preceding  
... ..

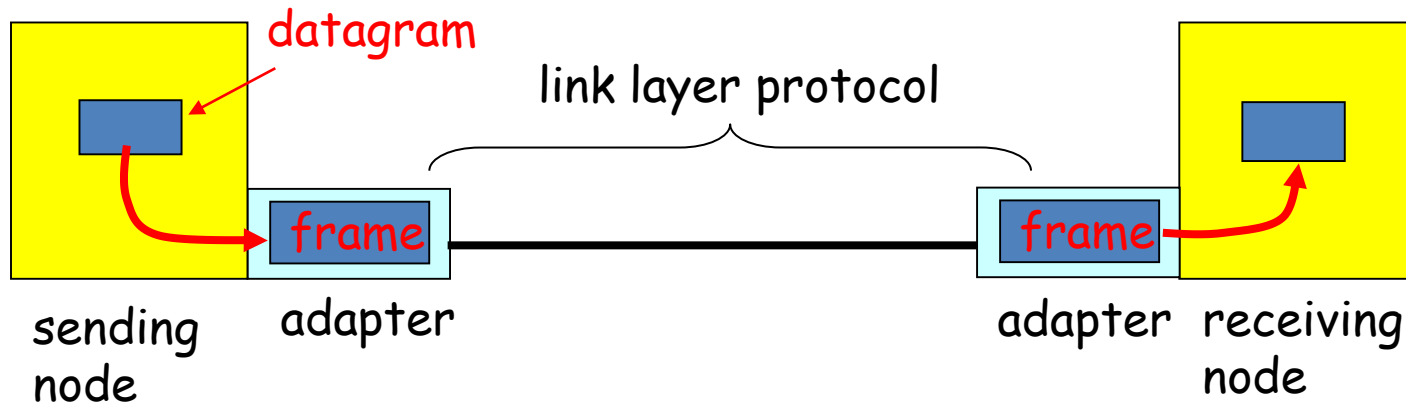


Popular NASA binary convolutional code (rate =  $\frac{1}{2}$ ) used in 802.11

# Link-Layer Services

- Encoding
  - Representing the 0s and 1s
- Framing
  - Encapsulating packet into frame, adding header, trailer
  - Using MAC addresses, rather than IP addresses
- Error detection
  - Errors caused by signal attenuation, noise.
  - Receiver detecting presence of errors
- Error correction
  - Receiver correcting errors without retransmission
- Flow control
  - Pacing between adjacent sending and receiving nodes

# Adaptors Communicating



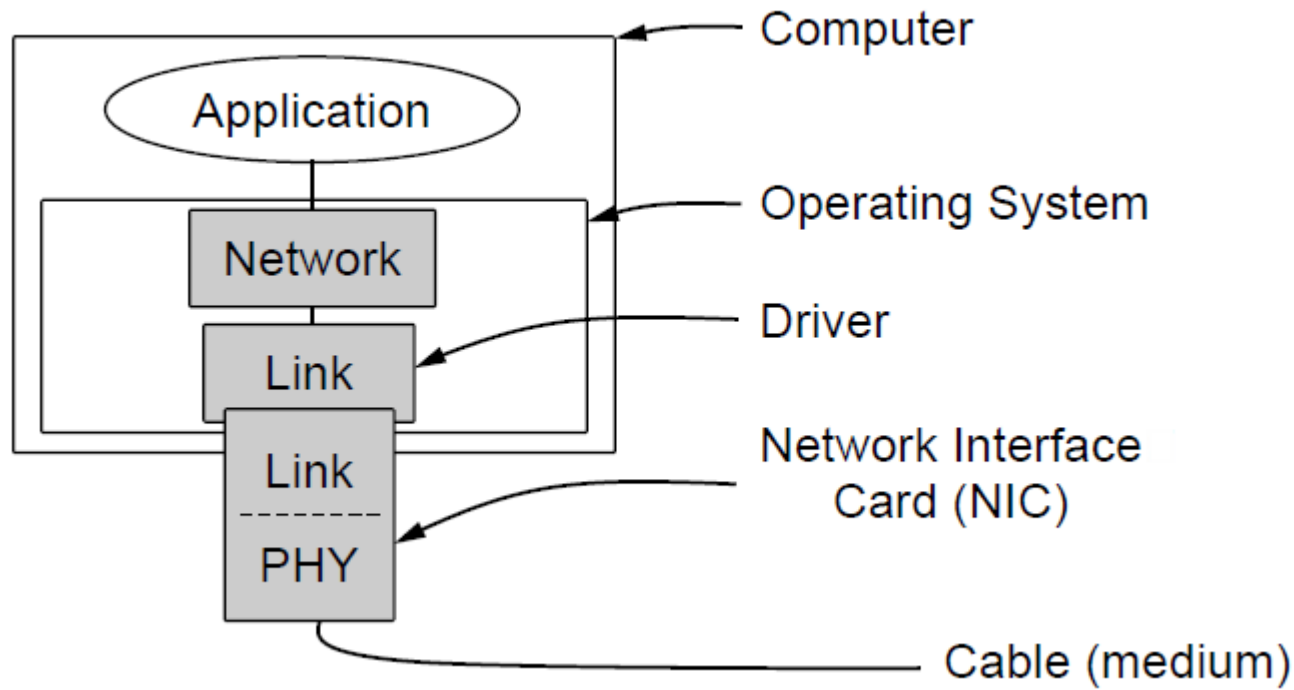
- Link layer implemented in adaptor (network interface card)
  - Ethernet card, PCMCIA card, 802.11 card
- Sending side:
  - Encapsulates datagram in a frame
  - Adds error checking bits, flow control, etc.
- Receiving side
  - Looks for errors, flow control, etc.
  - Extracts datagram and passes to receiving node

# Elementary Data Link Protocols

- Link layer environment »
- Utopian Simplex Protocol »
- Stop-and-Wait Protocol for Error-free channel »
- Stop-and-Wait Protocol for Noisy channel »

# Link layer environment (1)

Commonly implemented as NICs and OS drivers: network layer (IP) is often OS





# Link layer environment (2)

- Link layer protocol implementations use library functions
  - See code (`protocol.h`) for more details

Group	Library Function	Description
Network layer	<code>from_network_layer(&amp;packet)</code> <code>to_network_layer(&amp;packet)</code> <code>enable_network_layer()</code> <code>disable_network_layer()</code>	Take a packet from network layer to send Deliver a received packet to network layer Let network cause “ready” events Prevent network “ready” events
Physical layer	<code>from_physical_layer(&amp;frame)</code> <code>to_physical_layer(&amp;frame)</code>	Get an incoming frame from physical layer Pass an outgoing frame to physical layer
Events & timers	<code>wait_for_event(&amp;event)</code> <code>start_timer(seq_nr)</code> <code>stop_timer(seq_nr)</code> <code>start_ack_timer()</code> <code>stop_ack_timer()</code>	Wait for a packet / frame / timer event Start a countdown timer running Stop a countdown timer from running Start the ACK countdown timer Stop the ACK countdown timer

# Protocol Definitions

```
#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 →

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

# Protocol Definitions (ctd.)

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
```

# Transmission Sequence

A

B

A

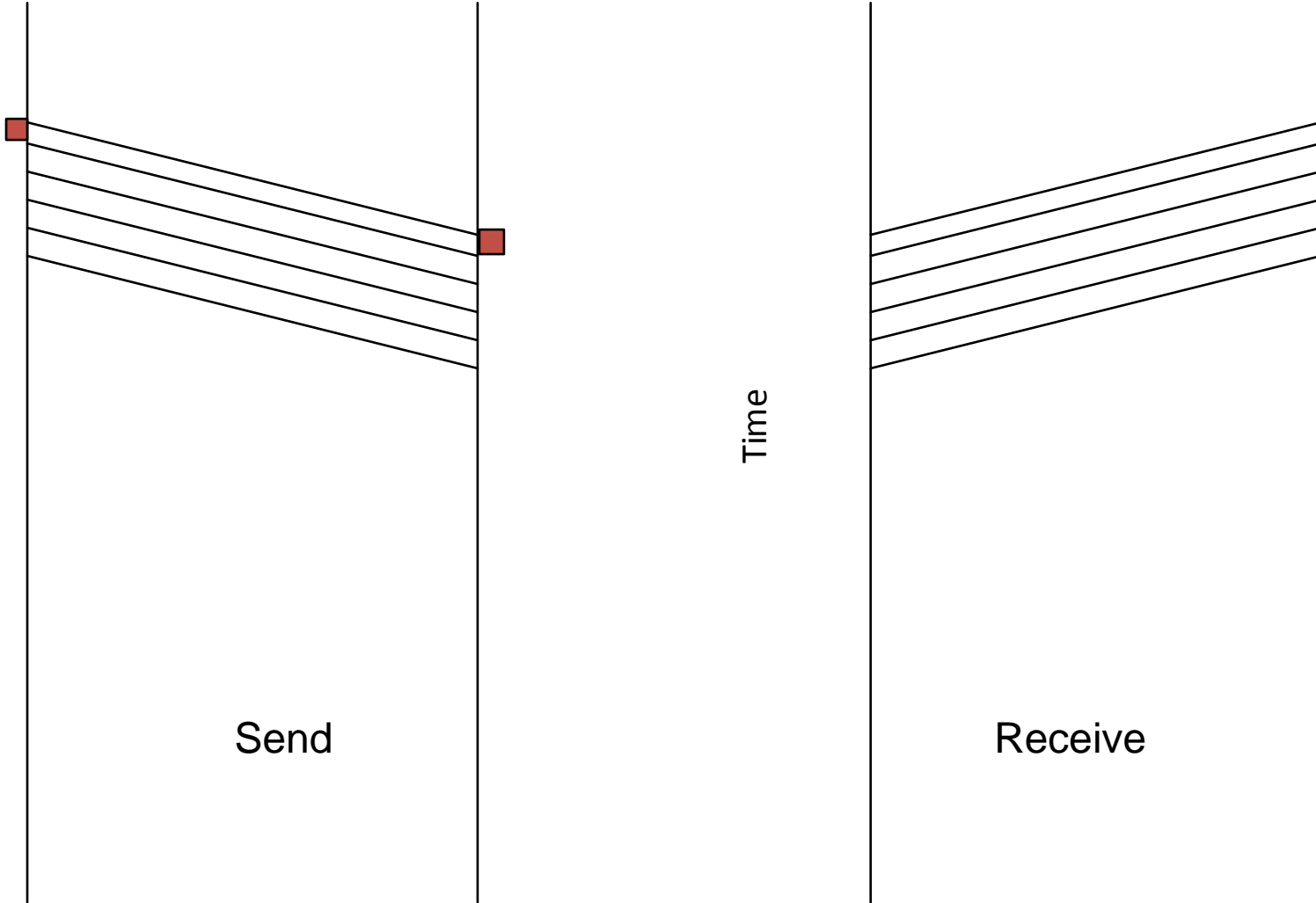
B

Time

Time

Send

Receive



# Utopian Simplex Protocol

An optimistic protocol (p1) to get us started

- Assumes no errors, and receiver as fast as sender
- Considers one-way data transfer

```
void sender1(void)
```

```
{  
    frame s;  
    packet buffer;  
  
    while (true) {  
        from_network_layer(&buffer);  
        s.info = buffer;  
        to_physical_layer(&s);  
    }  
}
```

Sender loops, blasting frames

```
void receiver1(void)
```

```
{  
    frame r;  
    event_type event;  
  
    while (true) {  
        wait_for_event(&event);  
        from_physical_layer(&r);  
        to_network_layer(&r.info);  
    }  
}
```

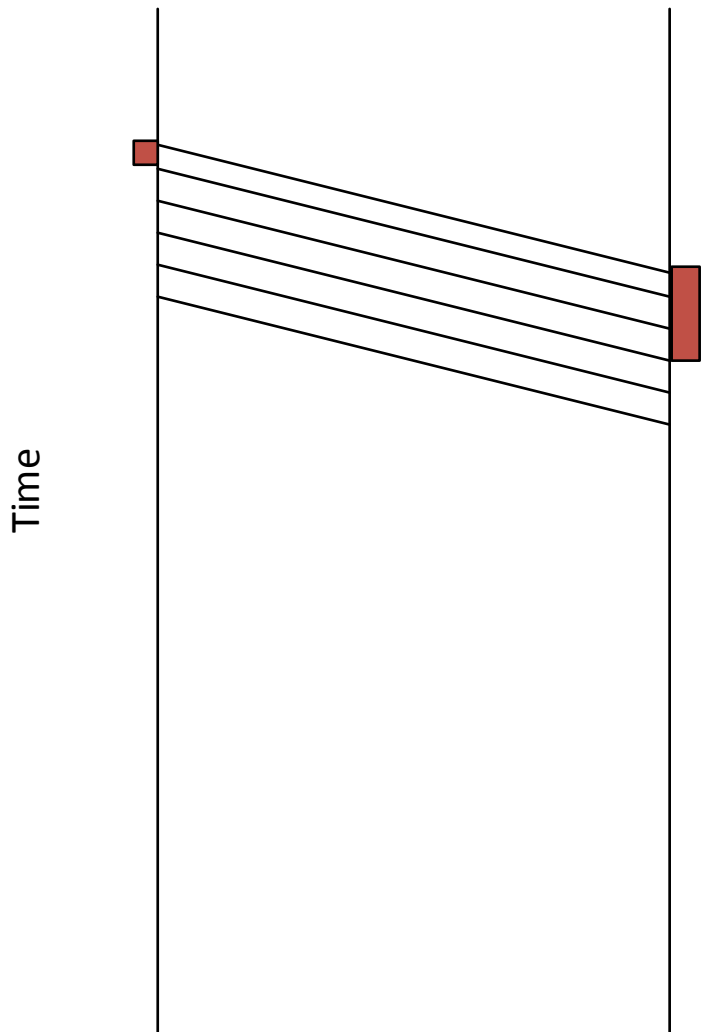
Receiver loops eating frames

- That's it, no error or flow control ...

# Flow Control

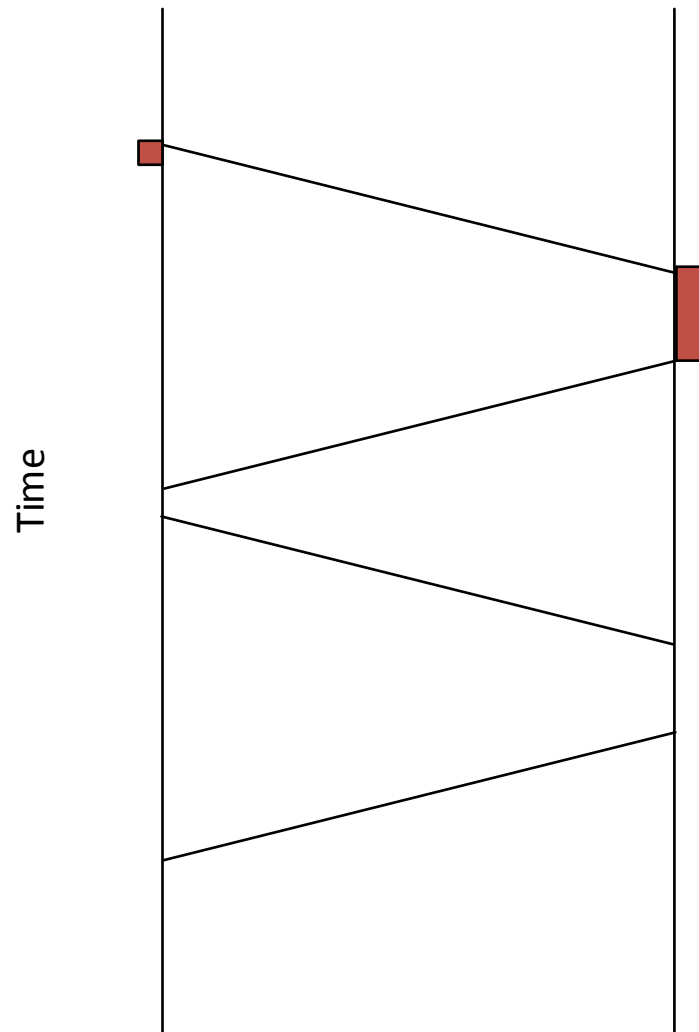
A

B



A

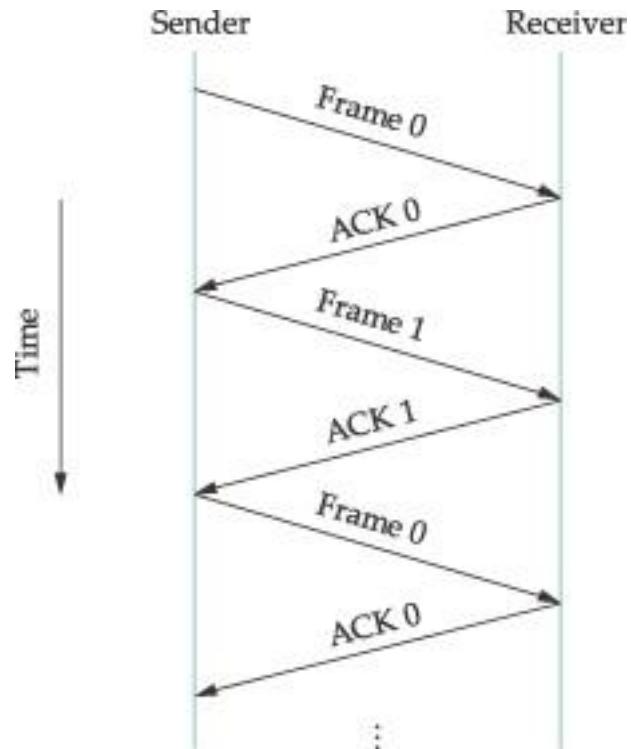
B



# Reliable Transmission

- Transfer frames without errors
  - Error Correction
  - Error Detection
  - Discard frames with error
- Acknowledgements and Timeouts
- Retransmission
- ARQ – Automatic Repeat Request

# Stop and Wait with 1-bit Seq No

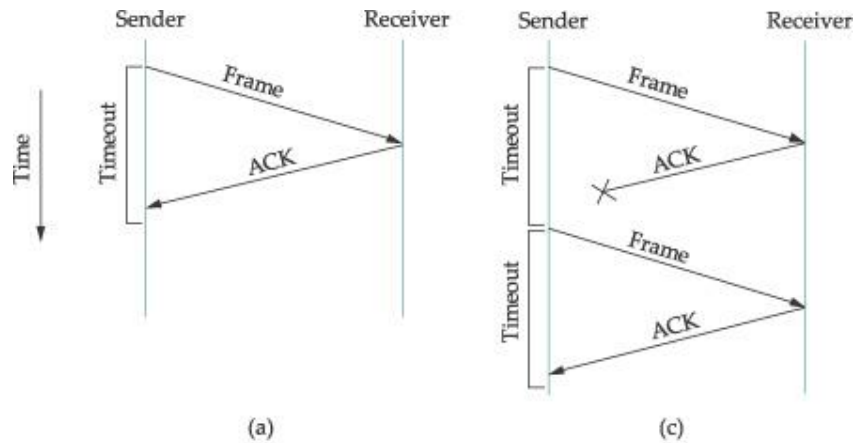




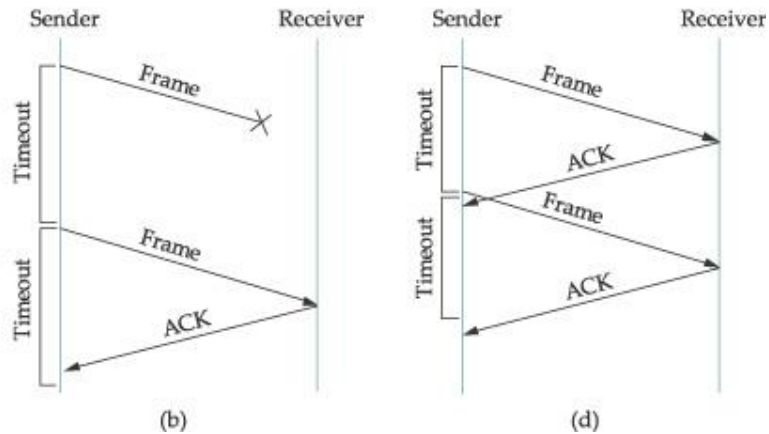
# Stop and Wait Protocols

- Simple
- Low Throughput
  - One Frame per RTT
- Increase throughput by having more frames in flight
  - Sliding Window Protocol

# Stop and Wait



Duplicate  
Frames



# Stop and Wait Protocol

- <http://www.cs.stir.ac.uk/~kjt/software/comms/jasper/ABP.html>
- <http://www.cs.stir.ac.uk/~kjt/software/comms/jasper/ABRA.html>

# Stop-and-Wait – Error-free channel

Protocol (p2) ensures sender can't outpace receiver:

- Receiver returns a dummy frame (ack) when ready
- Only one frame out at a time – called stop-and-wait
- We added flow control!

```
void sender2(void)
{
    frame s;
    packet buffer;
    event_type event;

    while (true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
        wait_for_event(&event);
    }
}
```

Sender waits to for ack after  
passing frame to physical layer

```
void receiver2(void)
{
    frame r, s;
    event_type event;
    while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```

Receiver sends ack after passing  
frame to network layer

# Stop-and-Wait – Noisy channel (1)

ARQ (Automatic Repeat reQuest) adds error control

- Receiver acks frames that are correctly delivered
- Sender sets timer and resends frame if no ack)

For correctness, frames and acks must be numbered

- Else receiver can't tell retransmission (due to lost ack or early timer) from new frame
- For stop-and-wait, 2 numbers (1 bit) are sufficient

# Stop-and-Wait – Noisy channel (2)

## Sender loop (p3):

Send frame (or retransmission)  
Set timer for retransmission  
Wait for ack or timeout

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

```
void sender3(void) {  
    seq_nr next_frame_to_send;  
    frame s;  
    packet buffer;  
    event_type event;  
  
    next_frame_to_send = 0;  
    from_network_layer(&buffer);  
    while (true) {  
        s.info = buffer;  
        s.seq = next_frame_to_send;  
        to_physical_layer(&s);  
        start_timer(s.seq);  
        wait_for_event(&event);  
        if (event == frame_arrival) {  
            from_physical_layer(&s);  
            if (s.ack == next_frame_to_send) {  
                stop_timer(s.ack);  
                from_network_layer(&buffer);  
                inc(next_frame_to_send);  
            }  
        }  
    }  
}
```

# Stop-and-Wait – Noisy channel (3)

Receiver loop (p3):

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

    frame_expected = 0;
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
        }
        s.ack = 1 - frame_expected;
        to_physical_layer(&s);
    }
}
```

Wait for a frame →

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

Ack current frame →

# Sliding Window Protocols

- Sliding Window concept »
- One-bit Sliding Window »
- Go-Back-N »
- Selective Repeat »



# Sliding Window concept (1)

Sender maintains window of frames it can send

- Needs to buffer them for possible retransmission
- Window advances with next acknowledgements

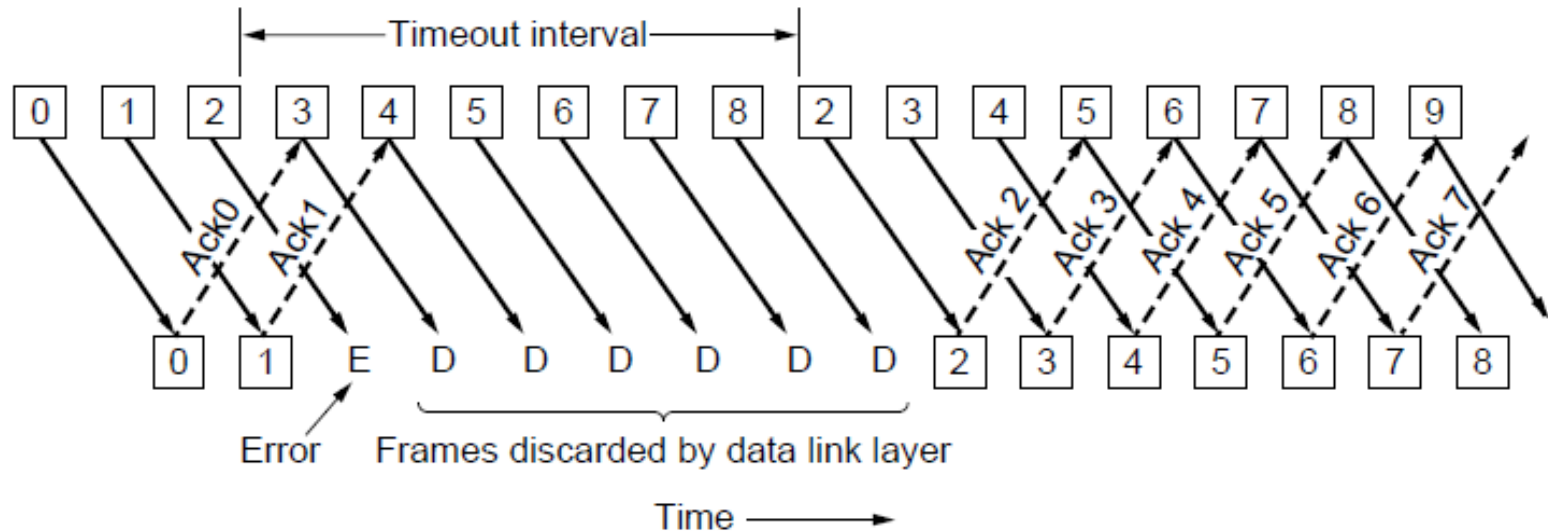
Receiver maintains window of frames it can receive

- Needs to keep buffer space for arrivals
- Window advances with in-order arrivals

# Go-Back-N (1)

Receiver only accepts/acks frames that arrive in order:

- Discards frames that follow a missing/errored frame
- Sender times out and resends all outstanding frames



# Go-Back-N (2)

Tradeoff made for Go-Back-N:

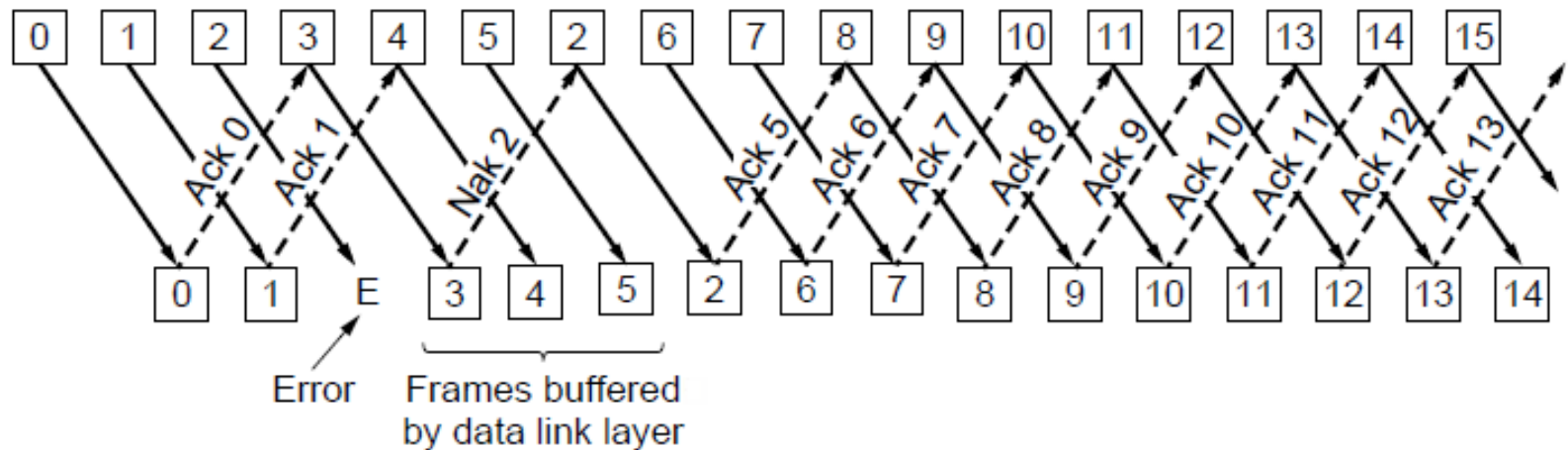
- Simple strategy for receiver; needs only 1 frame
- Wastes link bandwidth for errors with large windows; entire window is retransmitted

Implemented as p5 (see code in book)

# Selective Repeat (1)

Receiver accepts frames anywhere in receive window

- Cumulative ack indicates highest in-order frame
- NAK (negative ack) causes sender retransmission of a



# Selective Repeat (2)

Tradeoff made for Selective Repeat:

- More complex than Go-Back-N due to buffering at receiver and multiple timers at sender
- More efficient use of link bandwidth as only lost frames are resent (with low error rates)

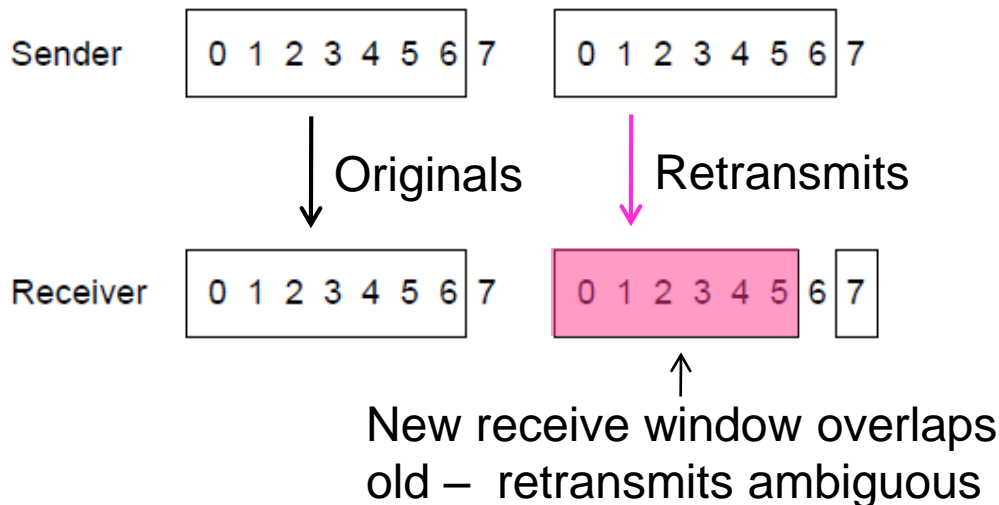
Implemented as p6 (see code in book)

# Selective Repeat (3)

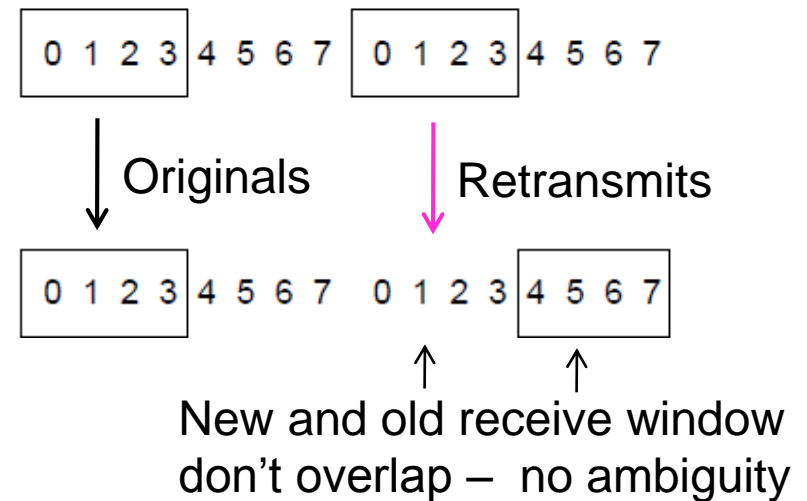
For correctness, we require:

- Sequence numbers (s) at least twice the window (w)

Error case ( $s=8, w=7$ ) – too few sequence numbers



Correct ( $s=8, w=4$ ) – enough sequence numbers



# Sliding Window Protocols

[http://www.site.uottawa.ca/~elsaddik/abedweb/applets/Applets/Sliding\\_Window/sliding\\_window.html](http://www.site.uottawa.ca/~elsaddik/abedweb/applets/Applets/Sliding_Window/sliding_window.html)

<http://www.cs.stir.ac.uk/~kjt/software/comms/jasper/SWP3.html>

<http://www.cs.stir.ac.uk/~kjt/software/comms/jasper/SWP5.html>

<http://www.eecis.udel.edu/~amer/450/TransportApplets/GBN/GBNindex.html>

# Example Data Link Protocols

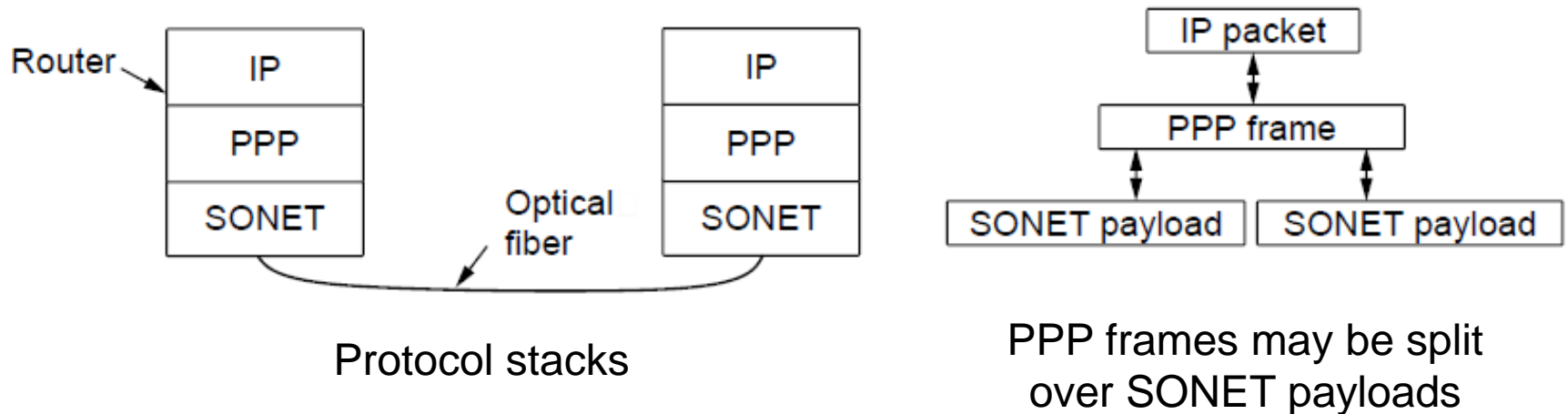
- Packet over SONET »
- PPP (Point-to-Point Protocol) »
- ADSL (Asymmetric Digital Subscriber Loop)  
»



# Packet over SONET

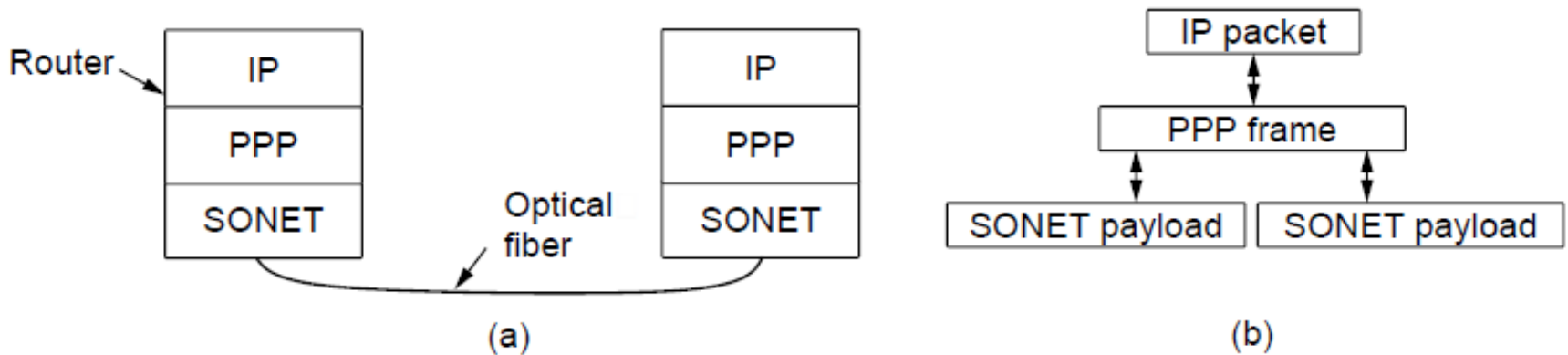
Packet over SONET is the method used to carry IP packets over SONET optical fiber links

- Uses PPP (Point-to-Point Protocol) for framing



# Packet over SONET (1)

Packet over SONET. (a) A protocol stack. (b)



# Packet over SONET (2)

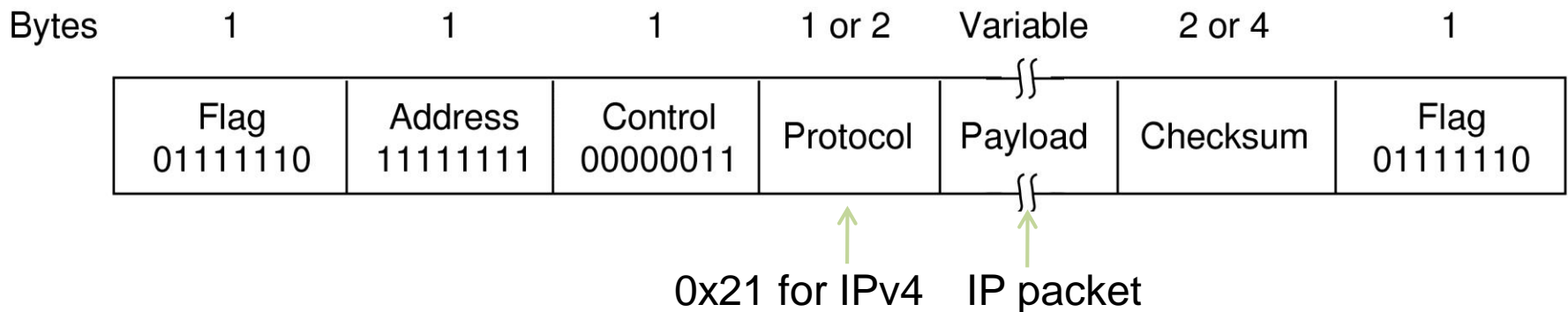
## PPP Features

1. Separate packets, error detection
2. Link Control Protocol
3. Network Control Protocol

# PPP (1)

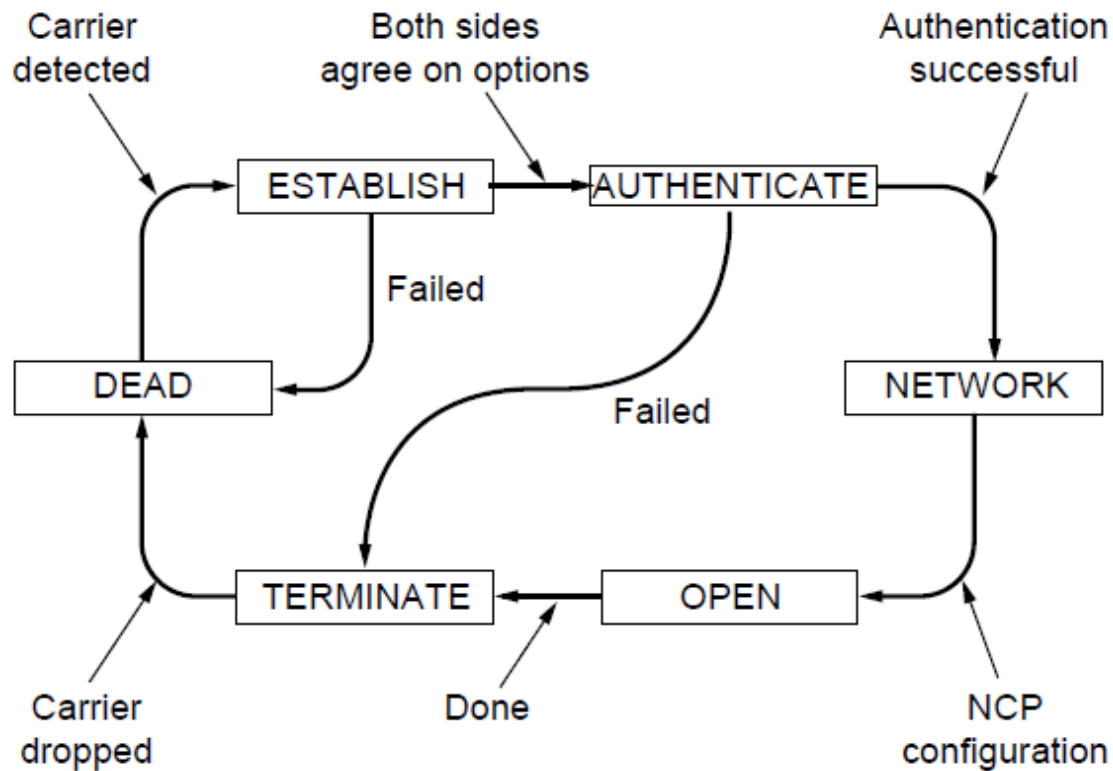
PPP (Point-to-Point Protocol) is a general method for delivering packets across links

- Framing uses a flag (0x7E) and byte stuffing
- “Unnumbered mode” (connectionless unacknowledged service) is used to carry IP packets
- Errors are detected with a checksum



# PPP (2)

A link control protocol brings the PPP link up and down



State machine for link control

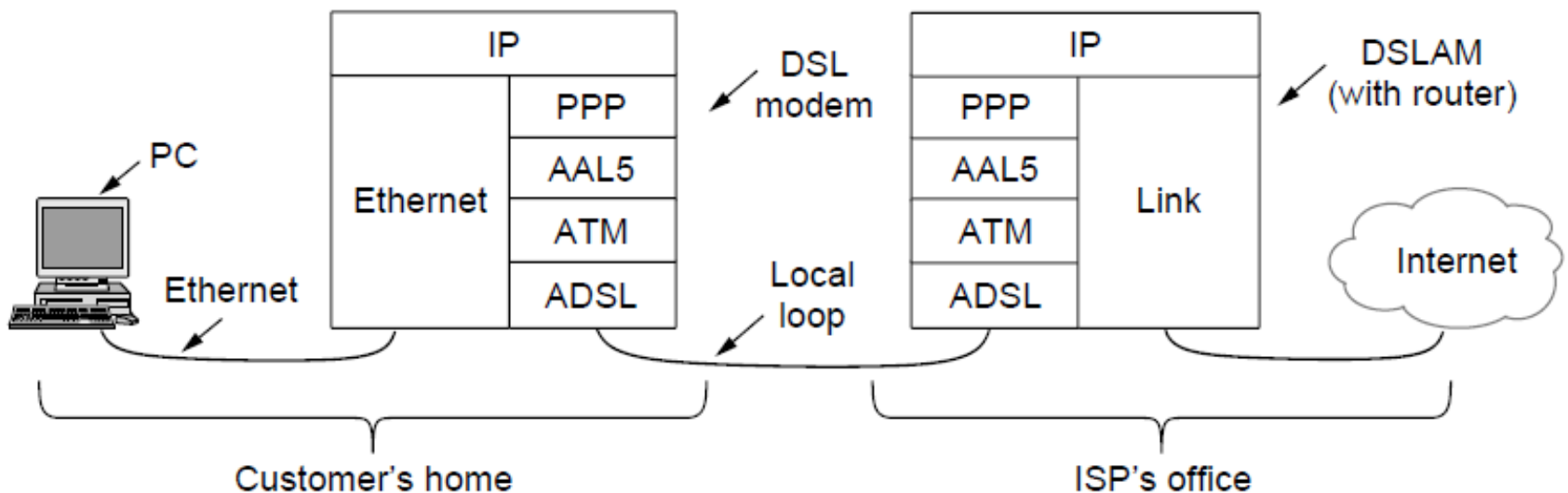
# PPP – Point to Point Protocol (3)

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

# ADSL (1)

Widely used for broadband Internet over local loops

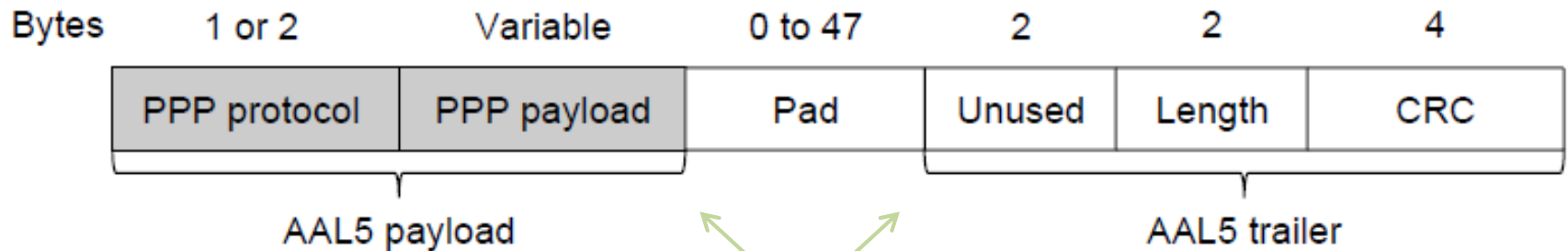
- ADSL runs from modem (customer) to DSLAM



# ADSL (2)

PPP data is sent in AAL5 frames over ATM cells:

- ATM is a link layer that uses short, fixed-size cells (53 bytes); each cell has a virtual circuit identifier

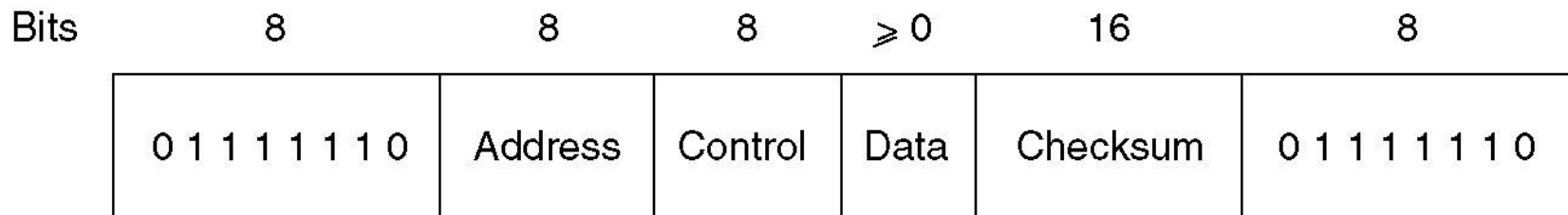


AAL5 frame is divided into 48 byte pieces, each of which goes into one ATM cell with 5 header bytes

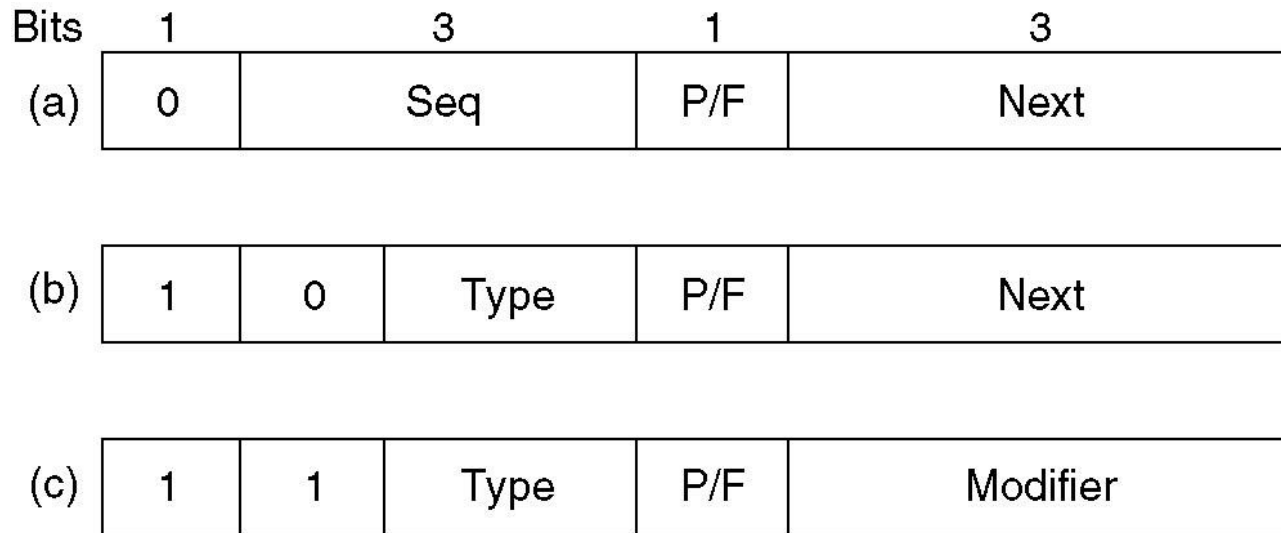


# High-Level Data Link Control

Frame format for bit-oriented protocols.



# High-Level Data Link Control (2)



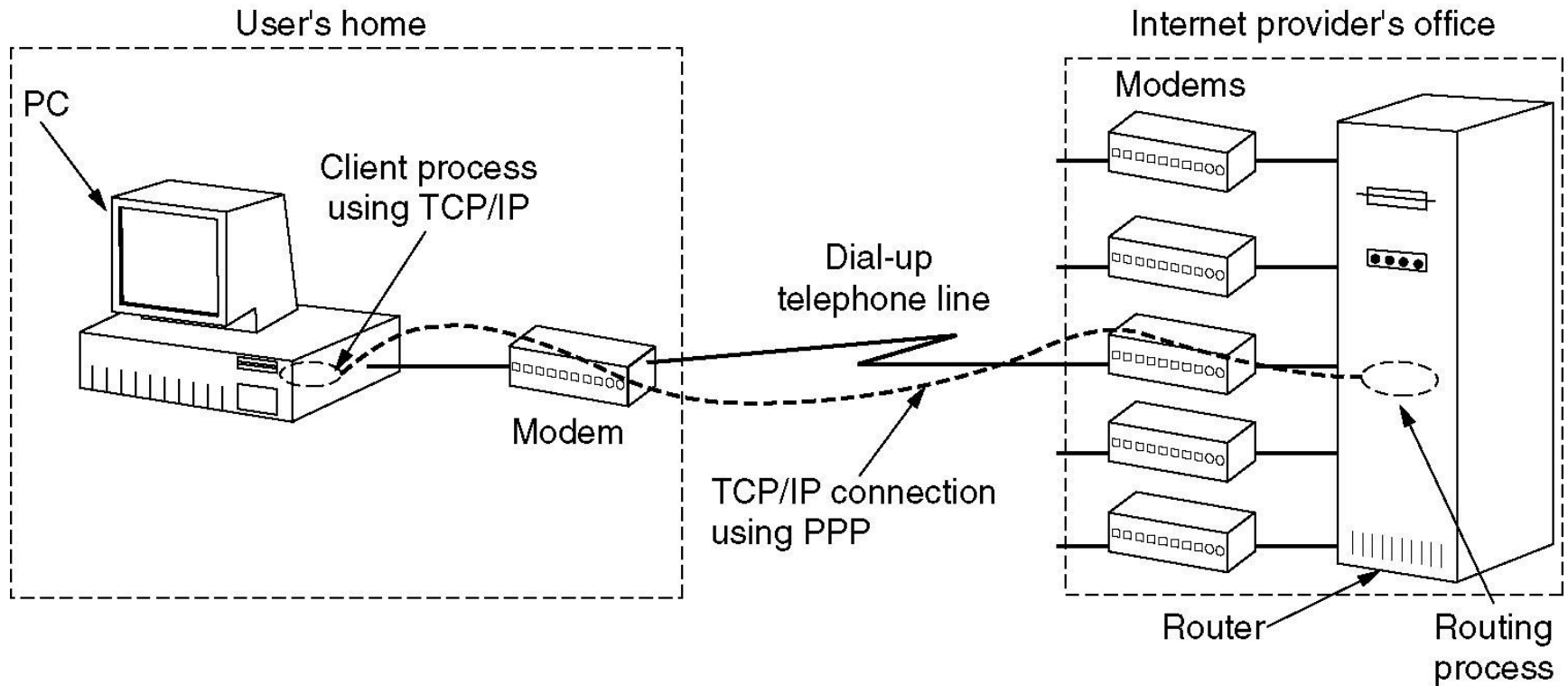
Control field of

(a) An information frame.

(b) A supervisory frame.

(c) An unnumbered frame.

# The Data Link Layer in the Internet



# End

## Chapter 3