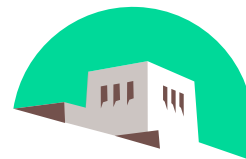


# Chapter 3: The Data Link Layer

## *Computer Networks*

Maccabe

Computer Science Department  
The University of New Mexico



August 2002

# Data Link Layer

---

Reliable bitstream between adjacent computers

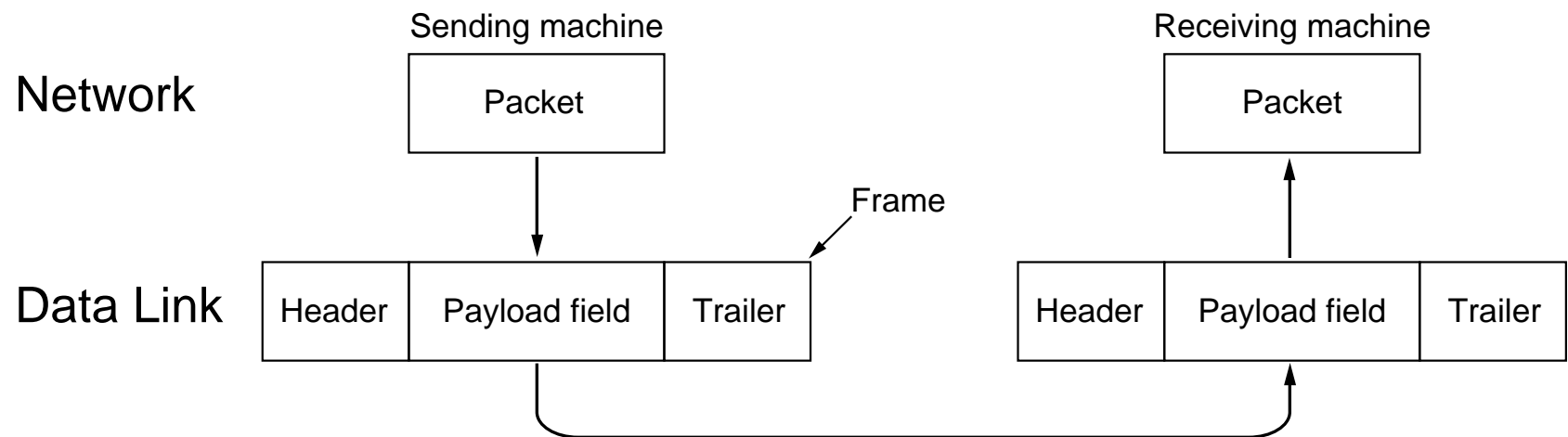
- Design Issues
- Error Detection and Correction
- Elementary Protocols
- Sliding Window Protocols
- Protocol Verification
- Example Data Link Protocols

# Design Issues

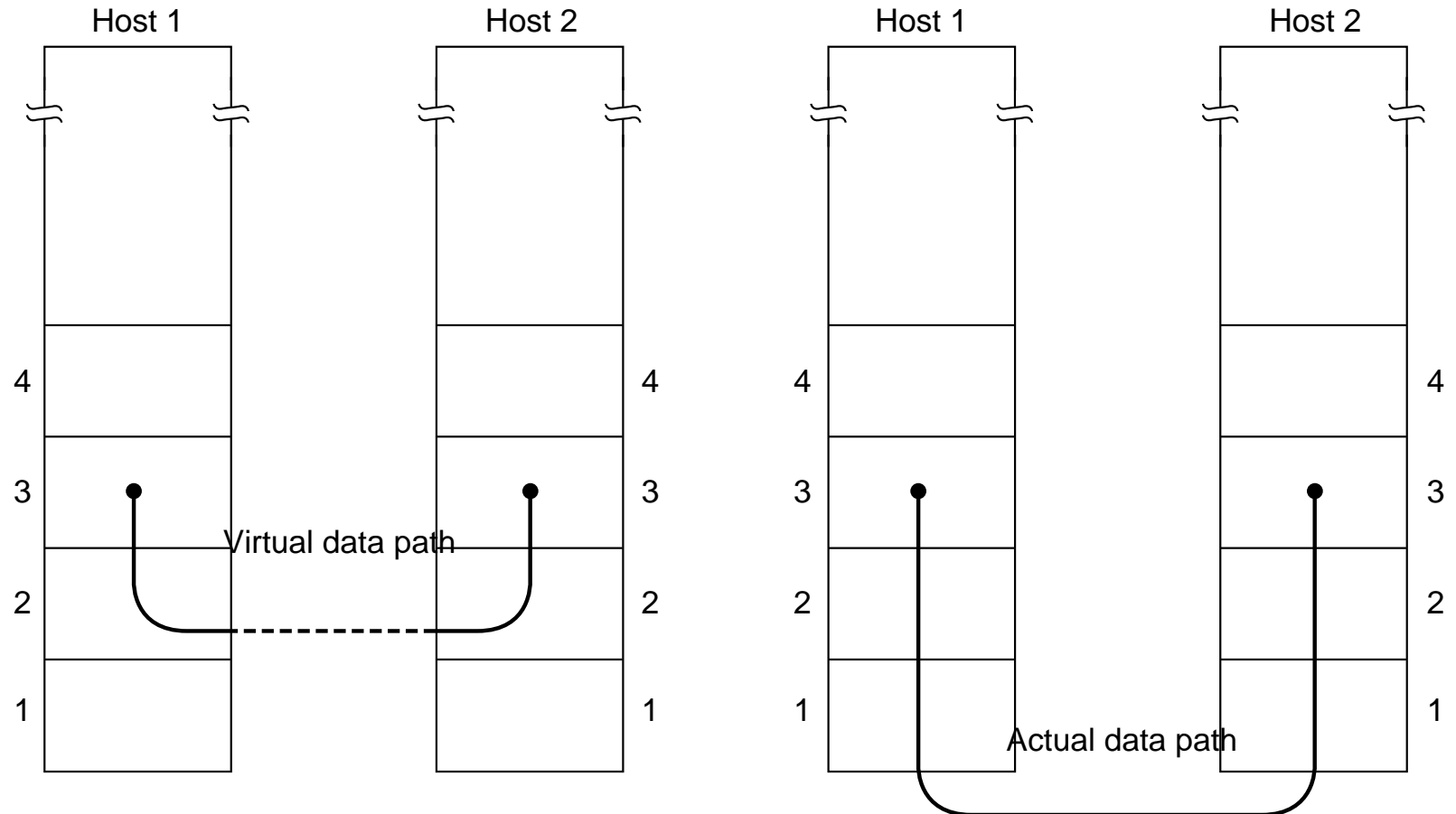
---

- Provides an interface to network layer
- Deals with transmission errors
- Regulates flow of data so slow receivers are not swamped
- Topics
  - Services
  - Framing
  - Error control
  - Flow control

# Terminology: Packets and Frames



# Layered Communication

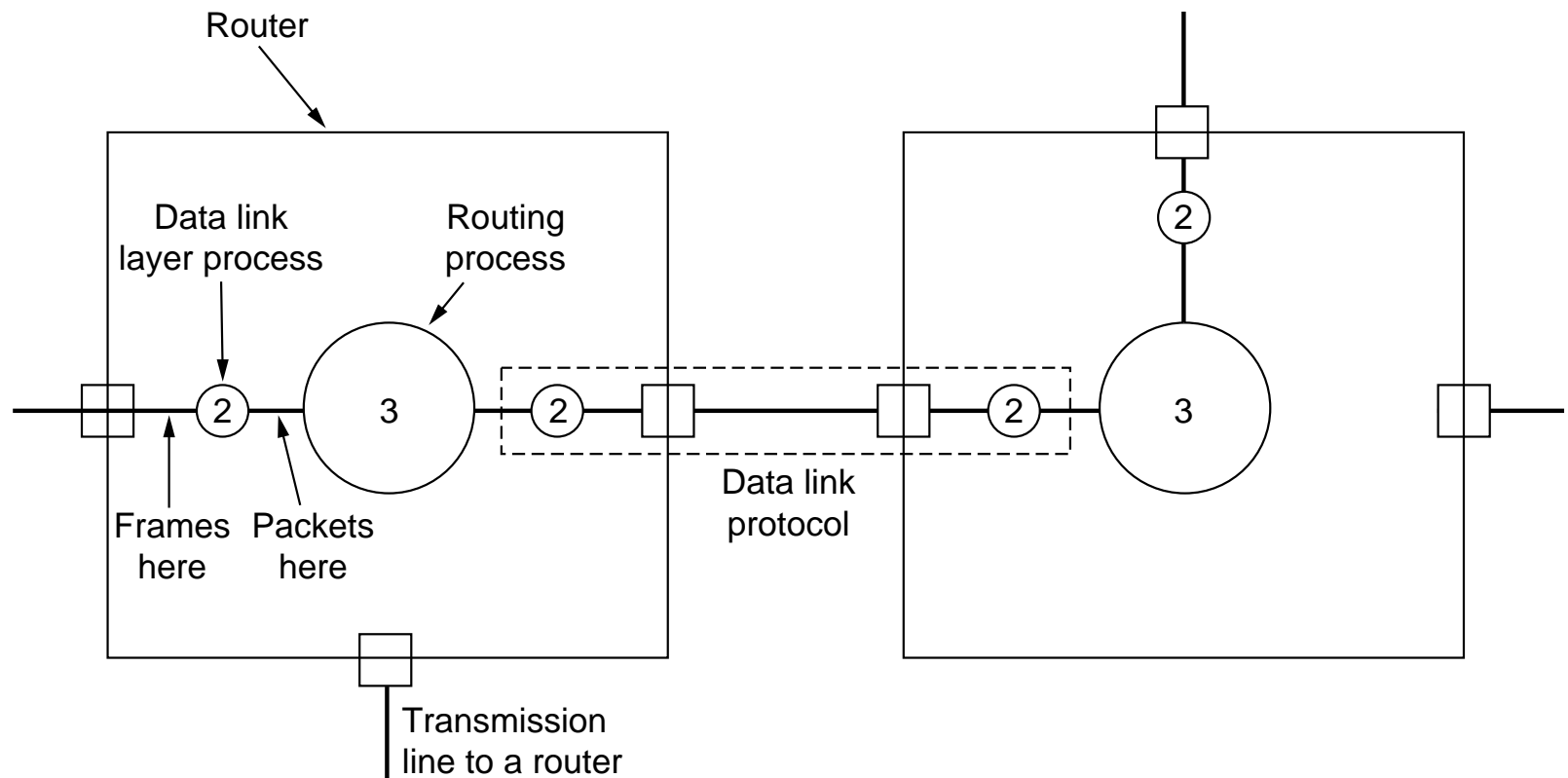


# Kinds of Services

---

- **Unacknowledged connectionless service**
  - no error detection or recovery
  - appropriate when error rate is very low
  - real-time – late data are worse than bad data
- **Acknowledged connectionless service**
  - each packet is acknowledged
  - retransmit after timeout expires
  - optimization when errors are frequent
- **Acknowledged connection-oriented service**
  - number frames to detect duplicates
  - connection setup, transmission, connection release

## Example: The Data Link Layer in Context



# Framing (1 of 3)

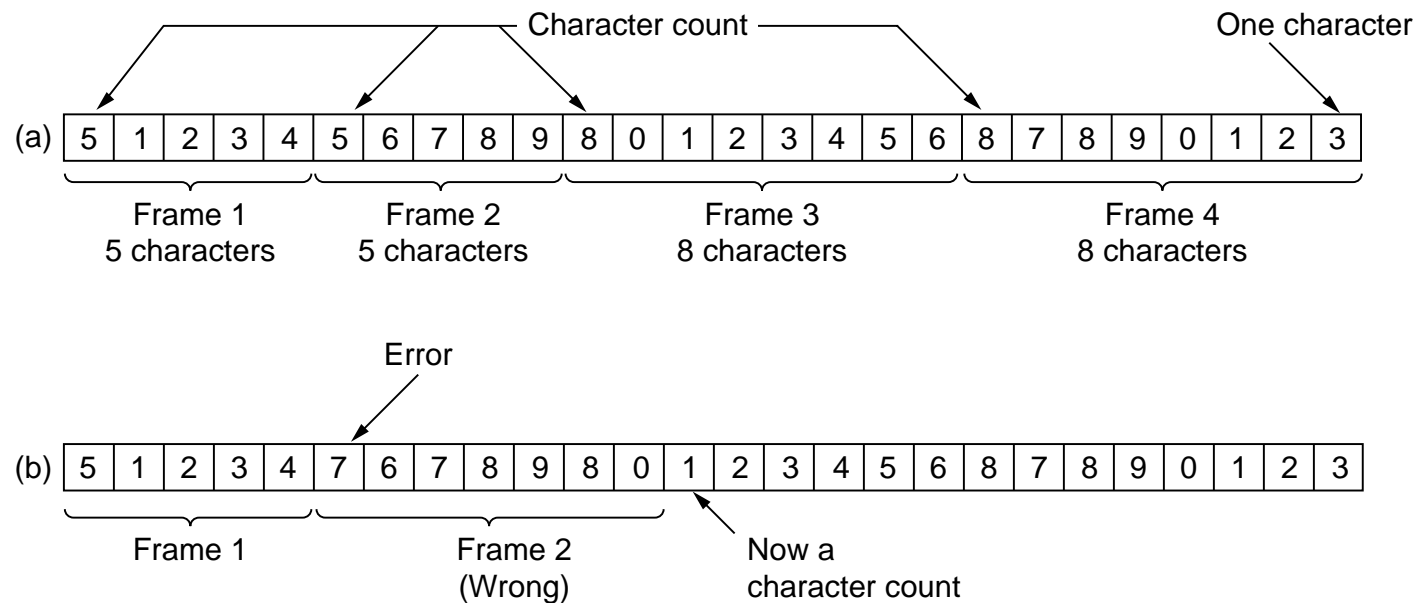
How to mark the start and end of each frame?

- 1. Time gaps

network may squeeze out the gaps during transmission

- 2. Character count: field in header for frame size

transmission errors are problematic

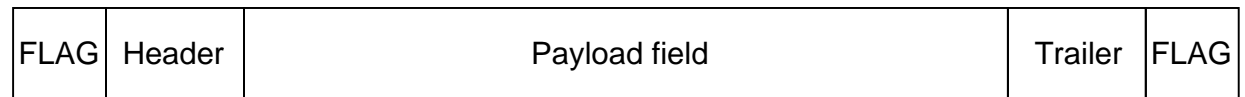




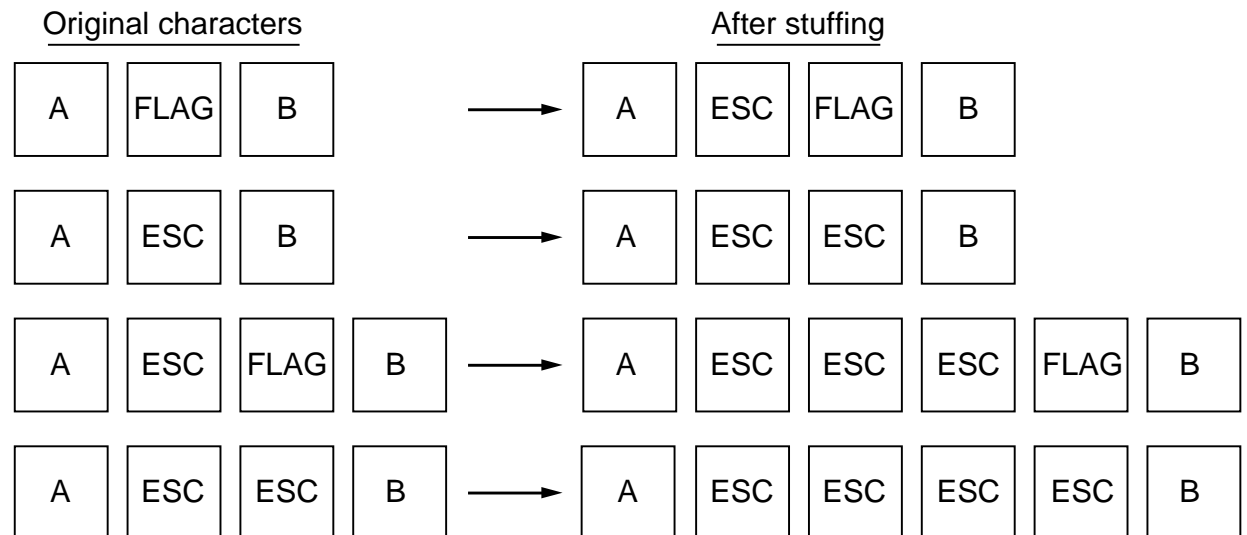
# Framing (2 of 3)

## ■ 3. Flag bytes: two consecutive flags indicates end and start of frame

### ■ flag bytes



### ■ escape conventions (stuffing)




# Framing (3 of 3)

## ■ 4. Flags with bit stuffing

Flag is 01111110

(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 0 1 1 1 1 1 0 1 0 0 1 0



Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

## ■ 5. Physical layer coding violations

physical layer provides a “frame” service

## ■ 6. Combination of count and flag

# Error Control

---

- Receiver provides feedback in the form of positive or negative acknowledgements (ACKs)
- Dropped frames are handled using timeouts
- Dropped transmissions or ACKs are handled by retransmitting
- Duplicate frames are handled using sequence numbers

# Flow Control

---

## Slowing down a fast sender

- feedback-based flow control
- rate-based flow control

# Data Link Layer

---

Reliable bitstream between adjacent computers

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

# Error Detection and Correction

## ■ Future importance

- errors are rare in digital devices
- local loops and wireless are analog and error prone

## ■ Bursty versus independent errors

- 1000 bits / frame
- bit error rate of 0.001

## ■ Topics

- error correcting
  - AKA, forward error correction
  - used on unreliable links
- error detecting
  - enough to know something went wrong
  - used on reliable links

# Hamming Codes, 1950 (1 of 8)

- An example illustrating error-correcting codes
- Codeword

$n$  bits =  $m$  message (data) bits +  $r$  redundant (check) bits

- Hamming distance

- count of corresponding bits that differ
- e.g., 10001001 and 10110001 have distance 3

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

- if distance is  $d$ ,  $d$  single bit errors are needed to convert one to the other

# Hamming Codes (2 of 8)

## Definitions

---

- A *code* is a set of codewords
- The Hamming distance for a set of code words is the *minimum* Hamming distance for any two codewords in the code
- To detect  $d$  single bit errors, the code must have distance  $d + 1$
- To correct  $d$  single bit errors, the code must have distance  $2d + 1$ , to ensure that the original codeword is closer than any other



# Hamming Codes (3 of 8)

## Simple Examples

- Single parity bit –distance 2

00 000

01 011

10 101

11 110

- A code with distance 3

00 000000

01 000111

10 111000

11 111111

# Hamming Codes (4 of 8)

## Correcting single bit errors

- $m$  message bits,  $n$  code bits,  $r$  check bits
- $m$  bits,  $2^m$  patterns
- each pattern needs  $n + 1$  patterns dedicated to it
- $(n + 1)2^m \leq 2^n$
- substitute  $m + r$  for  $n$  yields  $(m + r + 1) \leq 2^r$
- E.g.,  $m = 4$  implies  $r \geq 3$

# Hamming Codes (5 of 8)

Attaining a code with minimal  $n$

- Bits 1, 2, 4, 8, 16, ... are used as parity bits
- Bits 3, 5, 6, 7, 9, 10, 11, ... are used for the message
- For  $k = \sum c_i 2^i$ , bit in position  $k$  is checked by bit in positions  $2^i$  where  $c_i \neq 0$

# Hamming Codes (6 of 8)

Example:  $m = 7$ , thus  $r \geq 4$

bit	1	2	4	8
3	X	X		
5	X		X	
6		X	X	
7	X	X	X	
9	X			X
10		X		X
11	X	X		X

# Hamming Codes (7 of 8)

---

## Examples (even parity)

- Calculate the encoding for 1101101, e.g., what are bits 1, 2, 4, and 8.
- What does 11101011111 correspond to? Which bit was changed?

# Hamming Codes (8 of 8)

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

Order of bit transmission

**Burst errors – send column-by-column**

# Cost of Error-Correction

## ■ Error Correction

- $m = 1000$  implies  $r = 10$

- $N + \frac{N}{100}$

## ■ Error detection

- 1 parity bit / 1,000 bits

- retransmit when error is detected

- if bit error rate is  $10^{-6}$ :

$$N + \frac{N}{1,000} + 1,000 \times (N/1,000) \times (1,000 \times 10^{-6}) = N + \frac{2N}{1,000}$$

- Can use rectangle trick to deal with burst errors

# CRC (1 of 7)

- Cyclic Redundancy Check
- An example of error-detection
- In the  $n \times k$  parity scheme, probability that a bad block will be accepted is  $2^{-n}$ 

probability that a column has the correct parity is .5
- In the CRC scheme, this probability is  $2^{-r}$  where  $r$  is the number of check bits (a common value is 32)



# CRC (2 of 7)

- **Theory based on polynomial arithmetic, module 2**
  - polynomials provide algebraic structure for reasoning about error detection
  - 110010 is viewed as expressing a polynomial of degree 5, e.g.,  $x^5 + x^4 + x$
  - both addition and subtraction use bitwise
  - long division is done just like binary division, except that XOR is used for subtraction
- **Use of CRC**
  - sender and receiver agree on **generator polynomial**
  - sender ensures that message + **checksum** is divisible by the generator polynomial

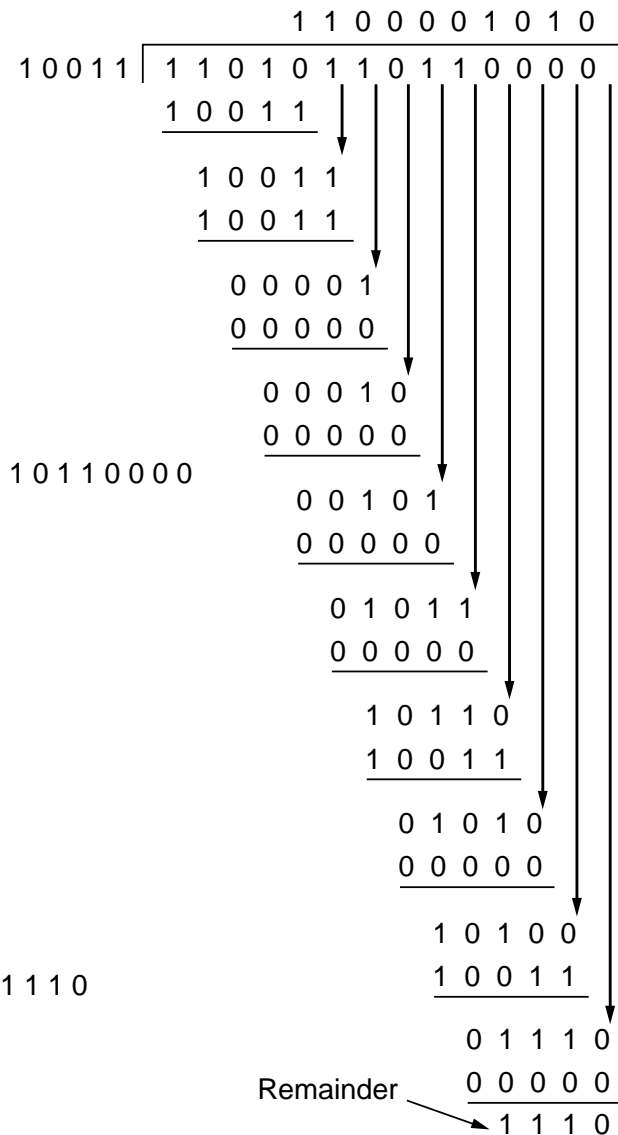
## CRC (3 of 7) Example

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

Generator: 1 0 0 1 1

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

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



# CRC (4 of 7)

A bit more of the theory

- $M(x)$  is the message (as a polynomial)
- $G(x)$  is the generator polynomial with degree  $r$
- Divide  $x^r M(x)$  by  $G(x)$ , call the remainder  $R(x)$
- Transmit  $T(x) = x^r M(x) - R(x)$
- $T(x)$  will be divisible by  $G(x)$

# CRC (5 of 7)

## Detecting Errors

- Let  $E(x)$  be the error polynomial (each 1 in  $E(x)$  represents a bit changed)
- The received value is  $T(x) + E(x)$
- The number of 1's in  $E(x)$  represents the number of errors in a burst
- In an undetected error  $[T(x) + E(x)]/G(x) = 0$
- That is,  $E(x)/G(x) = 0$
- I.e., if  $E(x)$  is a multiple of  $G(x)$ , the error will not be detected

# CRC (6 of 7)

## Detecting Errors

- Single bit errors  $E(x) = x^i$ 
  - if  $G(x)$  has two or more terms, all single bit errors will be detected
- Two bit errors,  $E(x) = x^i + x^j$  where  $i > j$  is  $x^j(x^{i-j} + 1)$ 
  - assume  $G(x)$  is not divisible by  $x$
  - no double errors will go undetected if  $G(x)$  does not divide  $X^k + 1$  (for  $k$  up to the maximum of  $i - j$ )
  - $x^{15} + x^{14} + 1$  will not divide  $x^k + 1$  for any  $k < 32,768$

# CRC (7 of 7)

- Lots more interesting facts about polynomials

- IEEE 802 standard

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

- Shift register implementation

- Analysis assumes that frames contain random bits

Recent evidence indicates that this may be an invalid assumption

# Data Link Layer

---

Reliable bitstream between adjacent computers

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

# Elementary Protocols

---

- Utopia
- Simplex Stop-and-Wait
- Positive Acknowledgement with Retransmission



# Utopia

```
void sender1( void )
```

```
{
```

```
    frame s;
```

```
    packet buffer;
```

```
    while( true ) {
```

```
        from_net_layer( &buffer );
```

```
        s.info = buffer;
```

```
        to_phy_layer( &s );
```

```
    }
```

```
}
```

```
void receiver1( void )
```

```
{
```

```
    frame r;
```

```
    event_type event;
```

```
    while( true ) {
```

```
        wait_for_event( &event );
```

```
        from_phys_layer( &r );
```

```
        to_net_layer( &r.info );
```

```
    }
```

```
}
```

# Stop-and-Wait

```
void sender2( void )
```

```
{  
    frame s;  
    packet buffer;  
    event_type event;
```

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

```
}
```

```
void receiver2( void )
```

```
{  
    frame r,s;  
    event_type event;
```

```
    while( true ) {  
        wait_for_event( &event );  
        from_phys_layer( &r );  
        to_net_layer( &r.info );  
        to_phy_layer( &s );  
    }
```

```
}
```

# Positive Acknowledgement with Retransmission

```
void sender3( void )
{
    seq_nr next_frame_to_send;
    frame s;
    packet buffer;
    event_type event;

    next_frame_to_send = 0;
    from_net_layer( &buffer );
    while( true ) {
        s.info = buffer;
        s.seq = next_frame_to_send;
        to_phy_layer( &s );
        set_timer( s.seq );
        wait_for_event( &event );
        if( event == frame_arrival ) {
            from_phys_layer( &s );
            if( s.ack == next_frame_to_send ) {
                stop_timer( s.ack );
                from_net_layer( &buffer );
                inc( next_frame_to_send );
            }
        }
    }
}
```

```
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_phys_layer( &r );
            if( r.seq == frame_expected ) {
                to_net_layer( &r.info );
                inc( frame_expected );
            }
            s.ack = 1 - frame_expected;
            to_phy_layer( &s );
        }
    }
}
```

# Data Link Layer

---

Reliable bitstream between adjacent computers

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

# Sliding Window Protocols

---

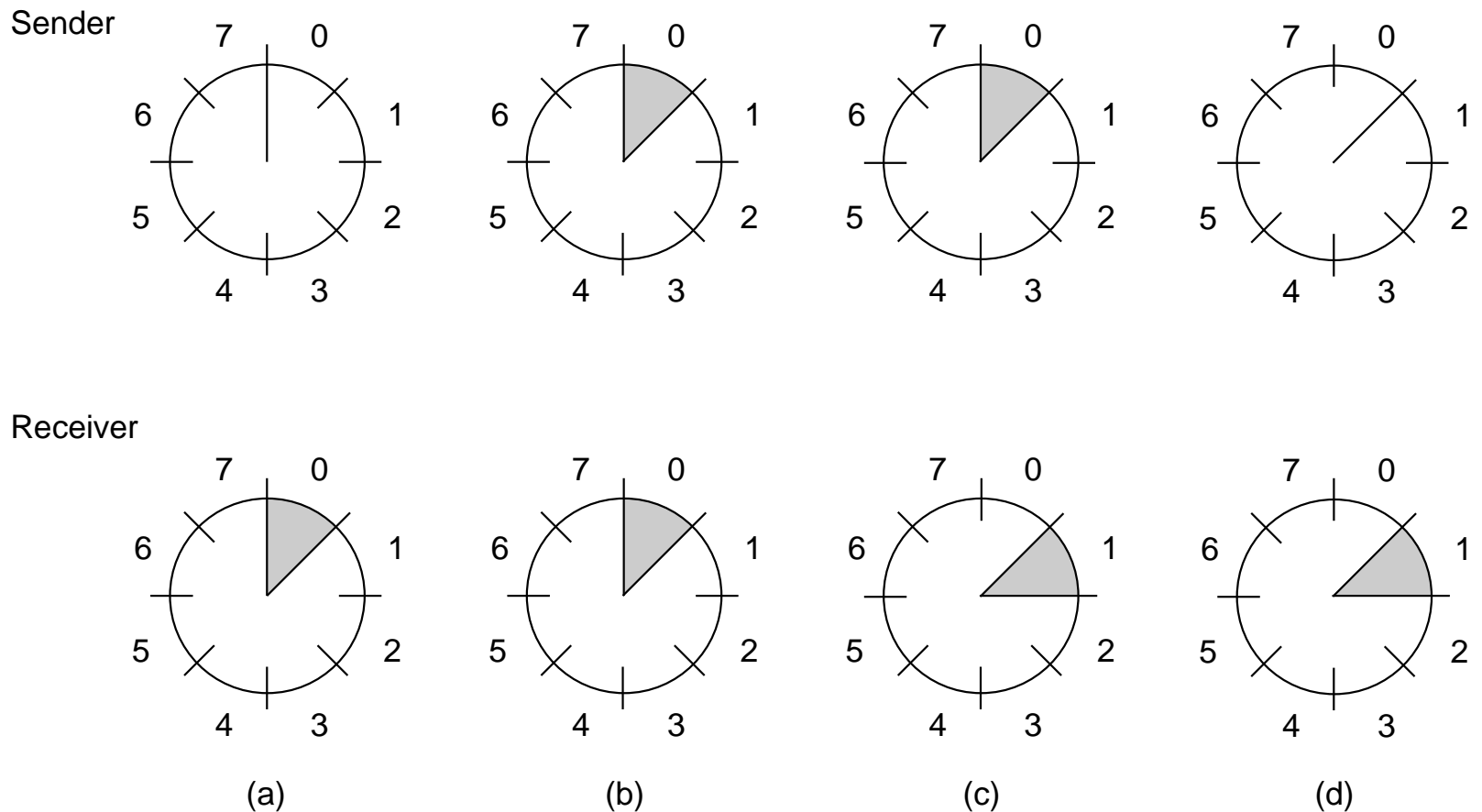
- Piggybacking
- One-Bit Sliding Window
- Go Back N
- Selective Repeat

# Piggybacking

---

- Wasted bandwidth on ACK channel
- Interleave ACKs in reverse data channel (it's probably there)
- Piggybacking—add ACKs in the header of outgoing data packets
- Issue: how long to wait for an outgoing data packet

# Sliding Window of Size 1



(a) initial; (b) after first frame sent; (c) first frame received; (d) first frame acknowledged

# One-Bit Sliding Window Initialization

```
void protocol4( void )
{
    seq_nr next_frame_to_send, frame_expected;
    frame r, s;
    packet buffer;
    event_type event;

    next_frame_to_send = 0;
    frame_expected = 0;
    from_net_layer( &buffer );
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 - frame_expected;
    to_phys_layer( &s );
    start_timer( s.seq );
}
```



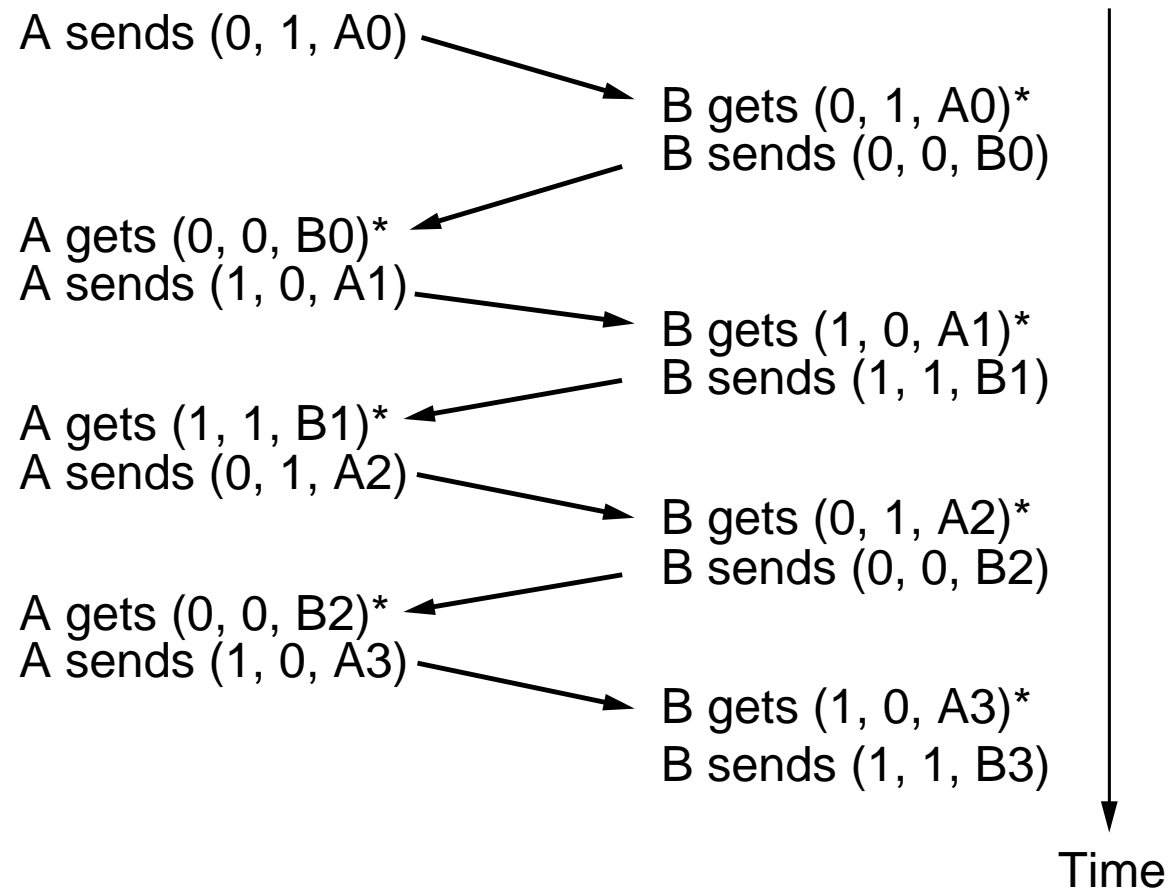
# One-Bit Sliding Window

## Main Loop

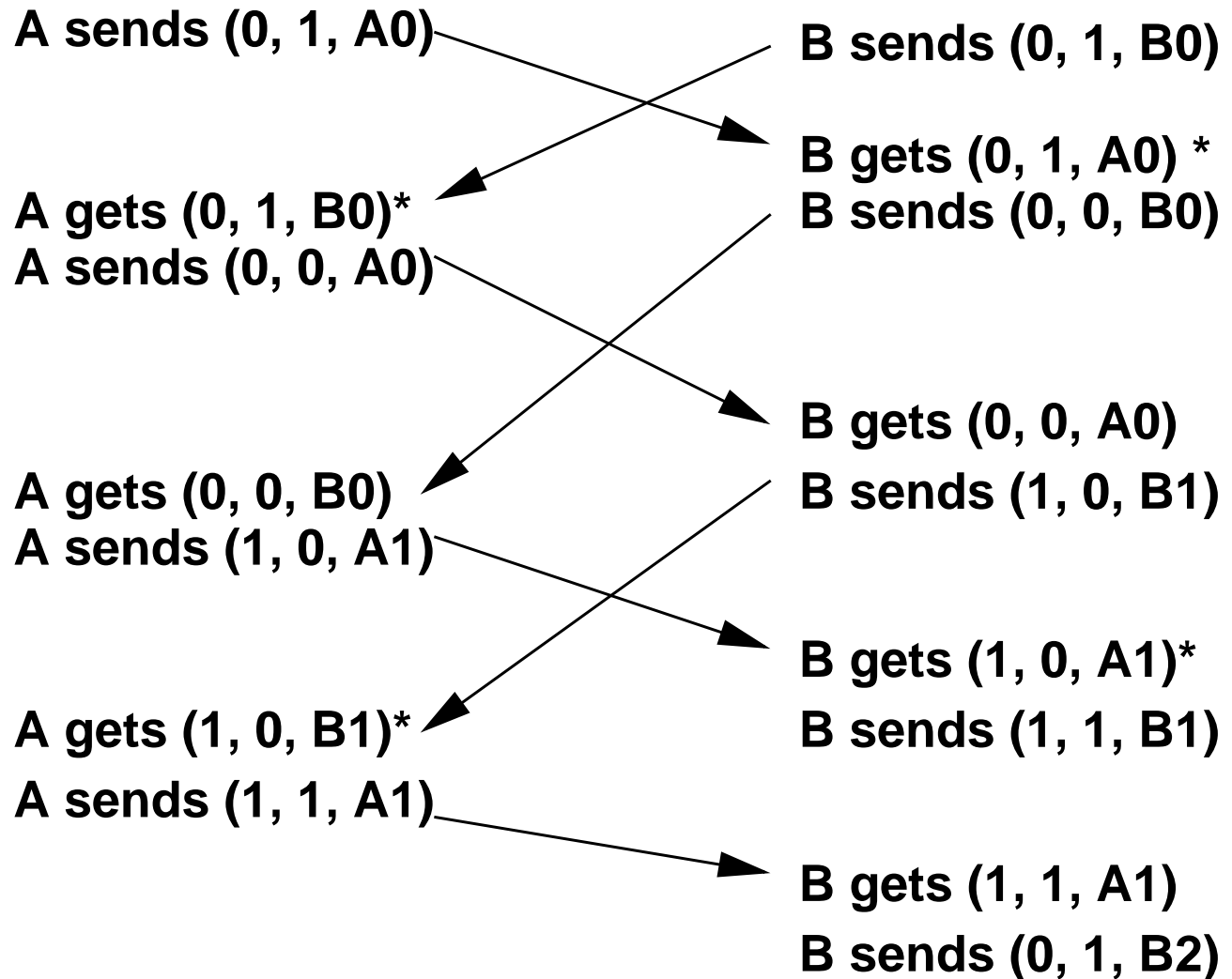
```
while( true ) {  
    wait_for_event( &event );  
    if( event == frame_arrival ) {  
        from_physical_layer( &r );  
        if( r.seq == frame_expected ) {  
            to_net_layer( &r.info ); inc( frame_expected );  
        }  
        if( r.ack == next_frame_to_send ) {  
            stop_timer( r.ack );  
            from_net_layer( &buffer ); inc( next_frame_to_send );  
        }  
    }  
    s.info = buffer;  
    s.seq = next_frame_to_send;  
    s.ack = 1 - frame_expected;  
    to_phys_layer( &s ); start_timer( s.seq );  
}
```

# One-Bit Sliding Window Normal Case

Notation: (sequence, ack, packet)



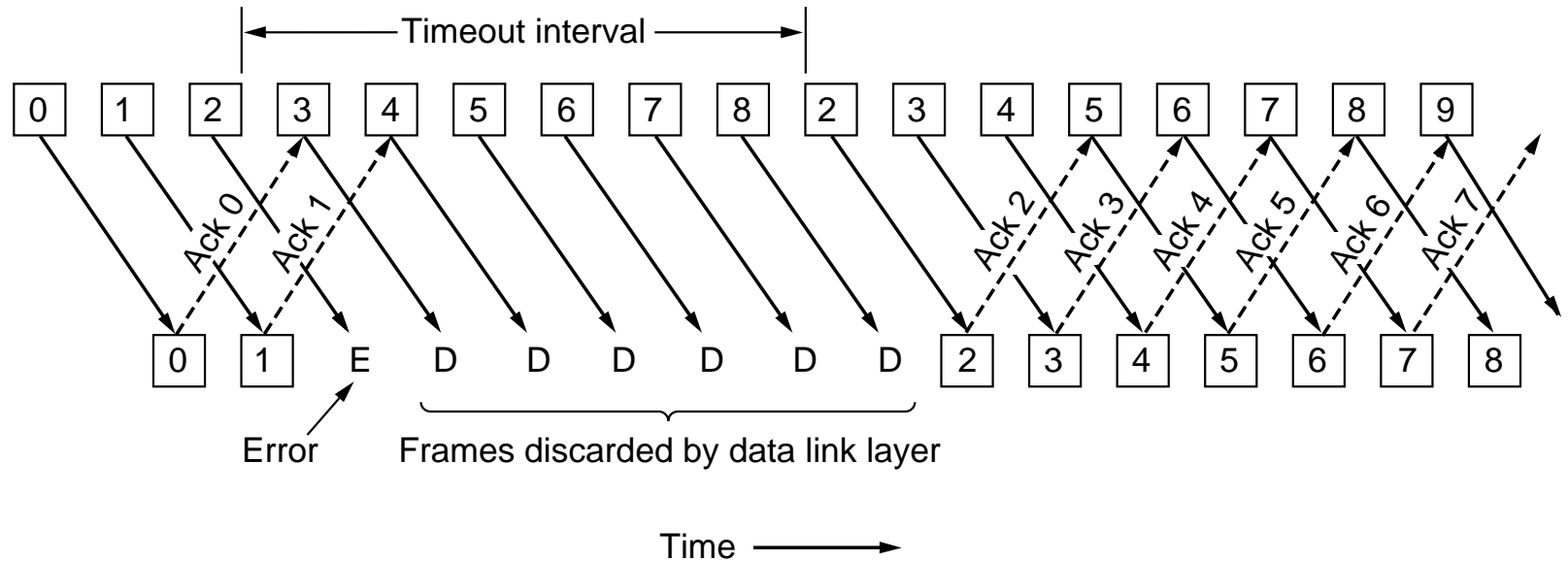
# One-Bit Sliding Window Degenerate Case



# Pipelining

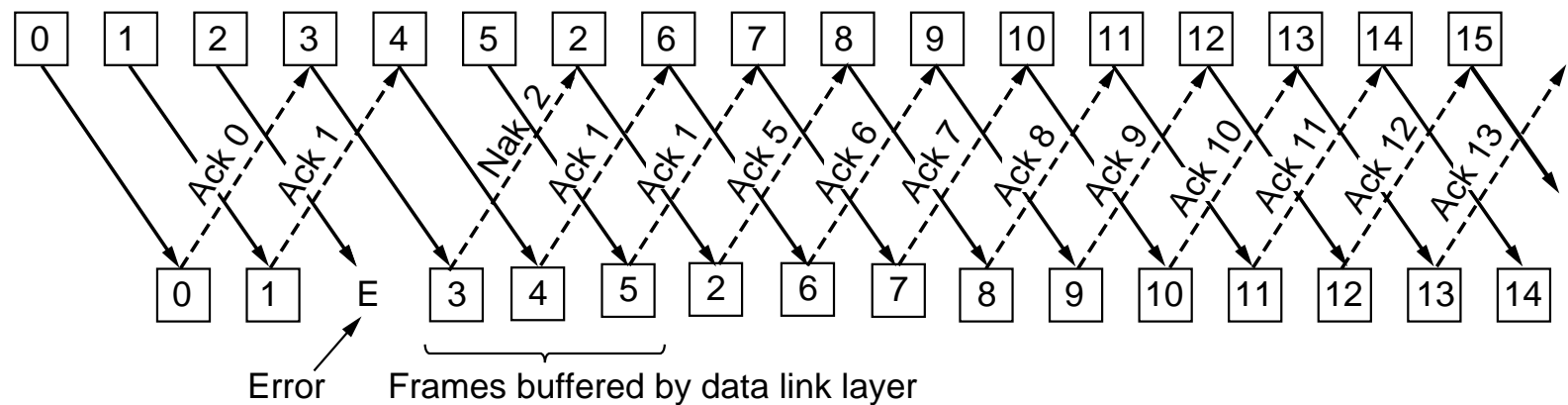
- Want to avoid stalls in transmission
  - in one-bit sliding window, we wait a full round trip between packet sends
- Make sender window large when:
  - bandwidth is high or
  - the round-trip time is high
  - bandwidth  $\times$  round-trip time
- line utilization  $= l / (l + bR)$ 
  - $l$  is the frame size (in bits)
  - $b$  is the channel capacity (in bits/sec)
  - $R$  is the round-trip time (in sec)
- If  $l < bR$  the efficiency is less than 50%

# Receiver Window Size is One



Leads to Go-back  $n$

# Receiver's Window is Large



Leads to selective repeat

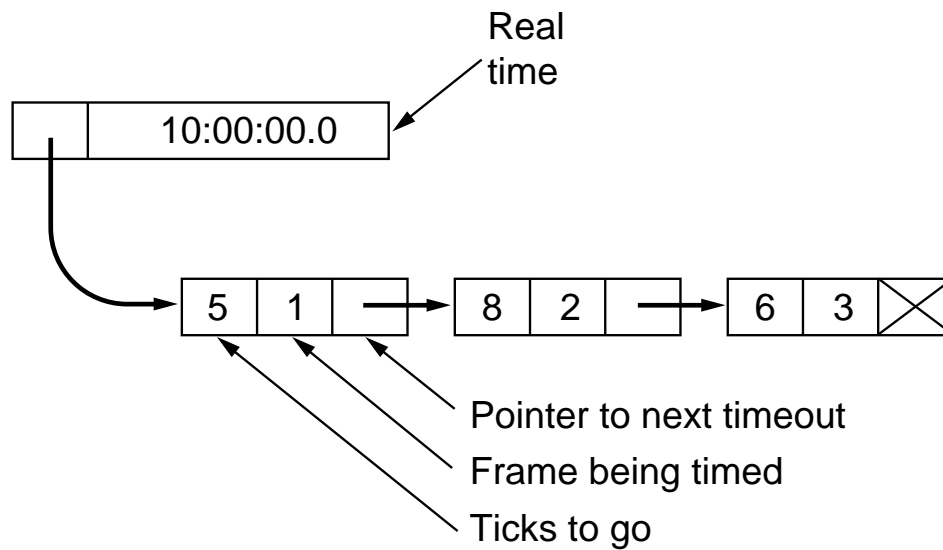
- tradeoff bandwidth and buffer capacity

# Go Back n Off by one

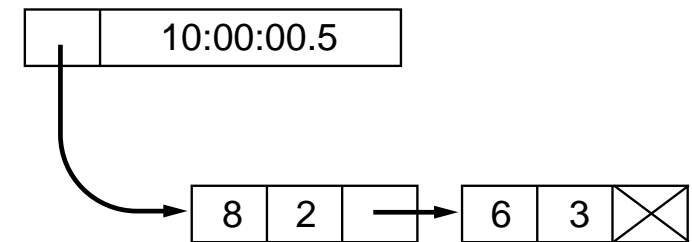
---

- Can only have MAX\_SEQ Frames in flight (not MAX\_SEQ + 1)
- Consider MAX\_SEQ = 7
  - Sender sends frames 0 through 7
  - Receiver acknowledges frame 7
  - Sender sends the next 8 frames with sequence numbers 0 through 7
  - Another acknowledgement for frame 7 arrives
  - Is this an acknowledgement for the first group or the second?

# Managing Multiple Timers



(a)

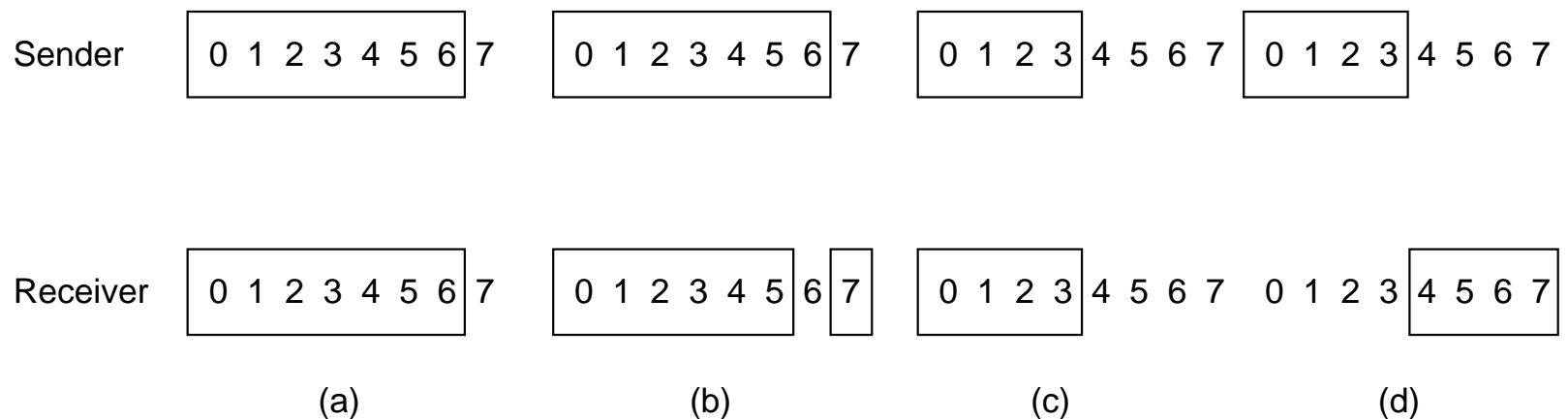


(b)



# Problem for Selective Repeat

Do not allow sender to overlap sequence numbers in consecutive windows



# Data Link Layer

---

Reliable bitstream between adjacent computers

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

# Protocol Verification

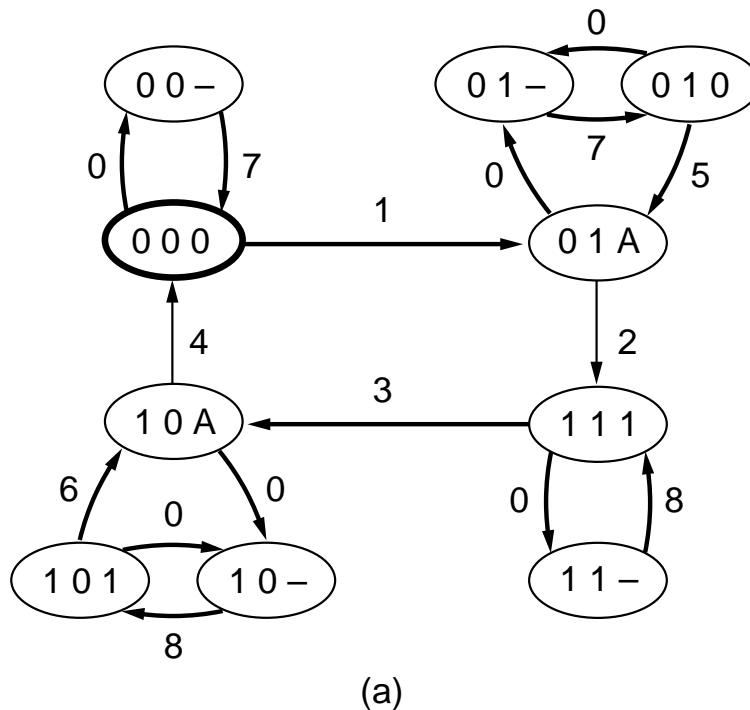
---

- Finite state machines

- Petri nets

# Finite State Machines

## State diagram and transitions for protocol 3

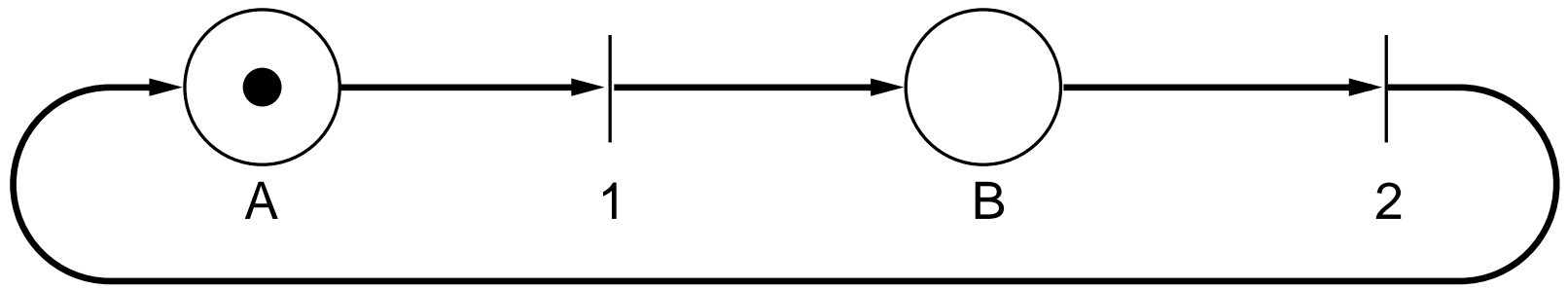


(b)

Transition	Who runs?	Frame accepted	Frame emitted	To network layer
0	–	(frame lost)		–
1	R	0	A	Yes
2	S	A	1	–
3	R	1	A	Yes
4	S	A	0	–
5	R	0	A	No
6	R	1	A	No
7	S	(timeout)	0	–
8	S	(timeout)	1	–

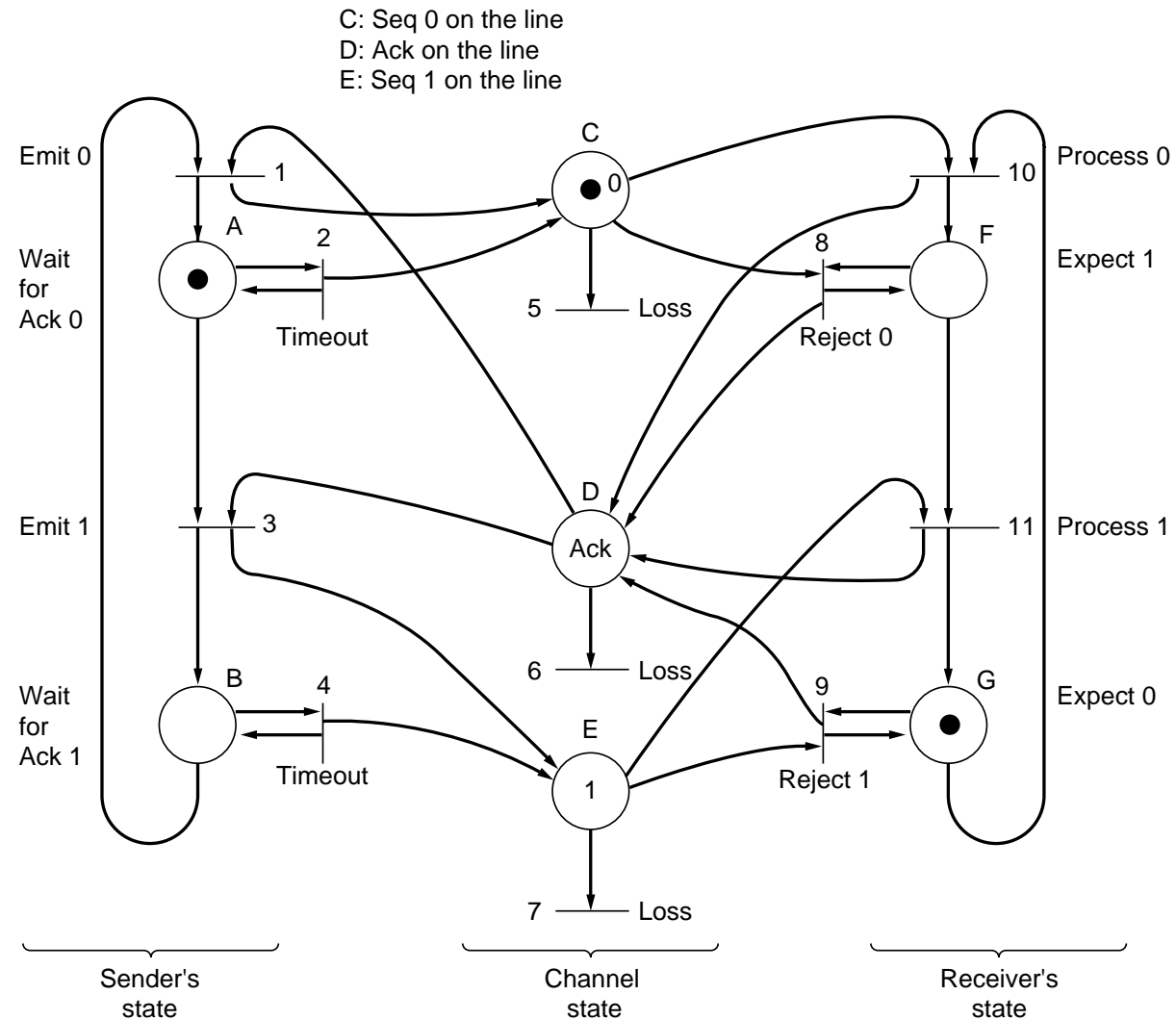
# Petri Nets (1 of 3)

- Places, transitions, arcs and tokens
- Bipartite graph
- Input places, enabled transition
- Transition firing, output places



# Petri Nets (2 of 3)

## Protocol 3



# Petri Nets (3 of 3)

## Grammar

- 1:  $BD \rightarrow AC$
- 2:  $A \rightarrow A$
- 3:  $AD \rightarrow BE$
- 4:  $B \rightarrow B$
- 5:  $C \rightarrow$
- 6:  $D \rightarrow$
- 7:  $E \rightarrow$
- 8:  $CF \rightarrow DF$
- 9:  $EG \rightarrow DG$
- 10:  $CG \rightarrow DF$
- 11:  $EF \rightarrow DG$

# Data Link Layer

---

Reliable bitstream between adjacent computers

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



# Example Protocols

---

- HDLC (High-Level Data Link Control)
- Data Link Layer on the Internet
  - PPP—The Point-to-Point Protocol

# HDLC (1 of 6)

---

- Lots of protocols

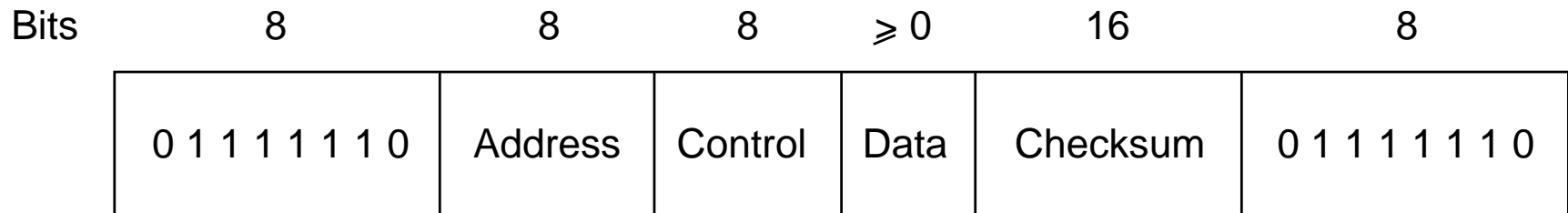
- SDLC (Synchronous Data Link Control) protocol – IBM
  - ADCCP (Advanced Data Communication Control Procedure – ANSI
  - HDLC (High-level Data Link Control) – ISO
    - LAP (Link Access Procedure) – CCITT

- All basically the same

- all are bit oriented
- all use bit stuffing
- differences are minor but irritating

# HDLC (2 of 6)

## Frame Format



**Flag value** 01111110

**Address** used to identify a terminal or distinguish commands from responses

**Control** sequence numbers, acknowledgements, etc.

**Data** any information, arbitrary length—efficiency of checksum falls off for very large data

**Checksum** CRC checksum

# HDLC (3 of 6)

## Kinds of Frames (Control Field)

Bits	1	3	1	3
(a)	0	Seq	P/F	Next

(b)	1	0	Type	P/F	Next
-----	---	---	------	-----	------

(c)	1	1	Type	P/F	Modifier
-----	---	---	------	-----	----------

- a** Information frame
- b** Supervisory frame
- c** Unnumbered frame

# HDLC (4 of 6)

## Control Protocol

---

**Seq** Sliding window with 3 bit sequence number

**Next** Piggybacked acknowledgement (next frame expected)

**P/F** Poll/Final, used for concentrator to terminals

**P** lets terminal send data

**F** terminal sending final data

# HDLC (5 of 6)

## Supervisory Frames

---

- Type 0** RECEIVE READY (ACK), used when no reverse traffic for piggybacking
- Type 1** REJECT (NACK); transmission error detected; *Next* field indicates frame to retransmit
- Type 2** RECEIVE NOT READY; ACK, but don't send more; eventually, receiver sends RECEIVE READY or REJECT
- Type 3** SELECTIVE REJECT

# HDLC (6 of 6)

## Unnumbered Frames

---

- Not standard across all protocols
- Some common commands

**DISC** DISConnect

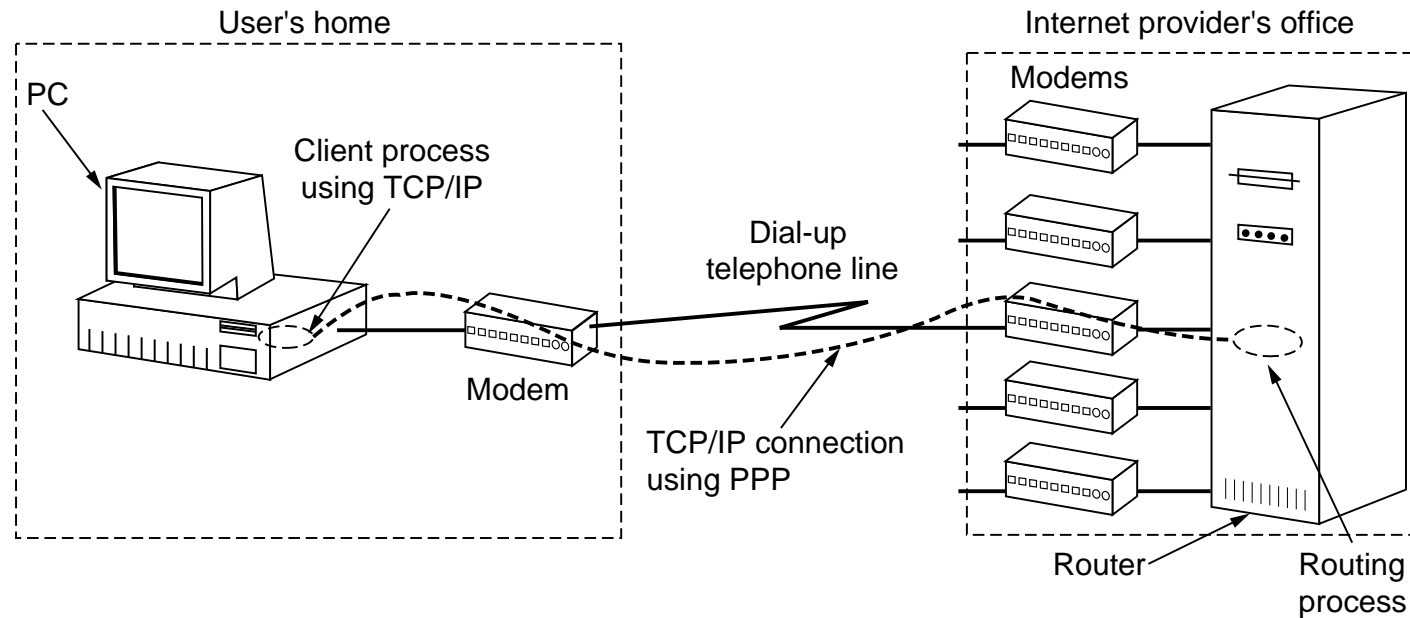
**SNRM** Set Normal Response Mode (asymmetric)

**FRMR** FRaMe Reject – correct data, but impossible semantics

**UA** Unnumbered Acknowledgement – for control frames

# PPP (1 of 5)

- LAN router to Internet router
- Dial-up host to LAN router



- RFC 1661 (1662 and 1663)

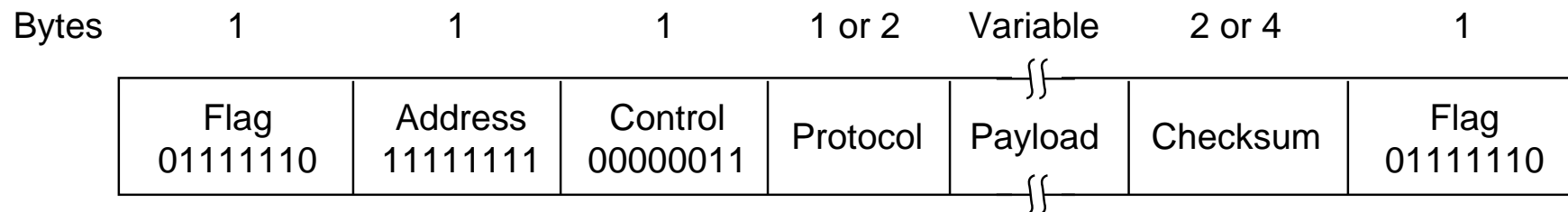


# PPP (2 of 5)

- Framing identifies frame boundaries and handles error detection
- Link Control Protocol (LCP)
  - bringing up and testing lines
  - negotiate parameters
  - bringing down lines
  - services
    - synchronous and asynchronous circuits
    - byte and bit-oriented encodings
- Network Control Protocol (NCP)
  - unique to the network layer being supported
  - IP version negotiates IP address

# PPP (3 of 5)

## Framing

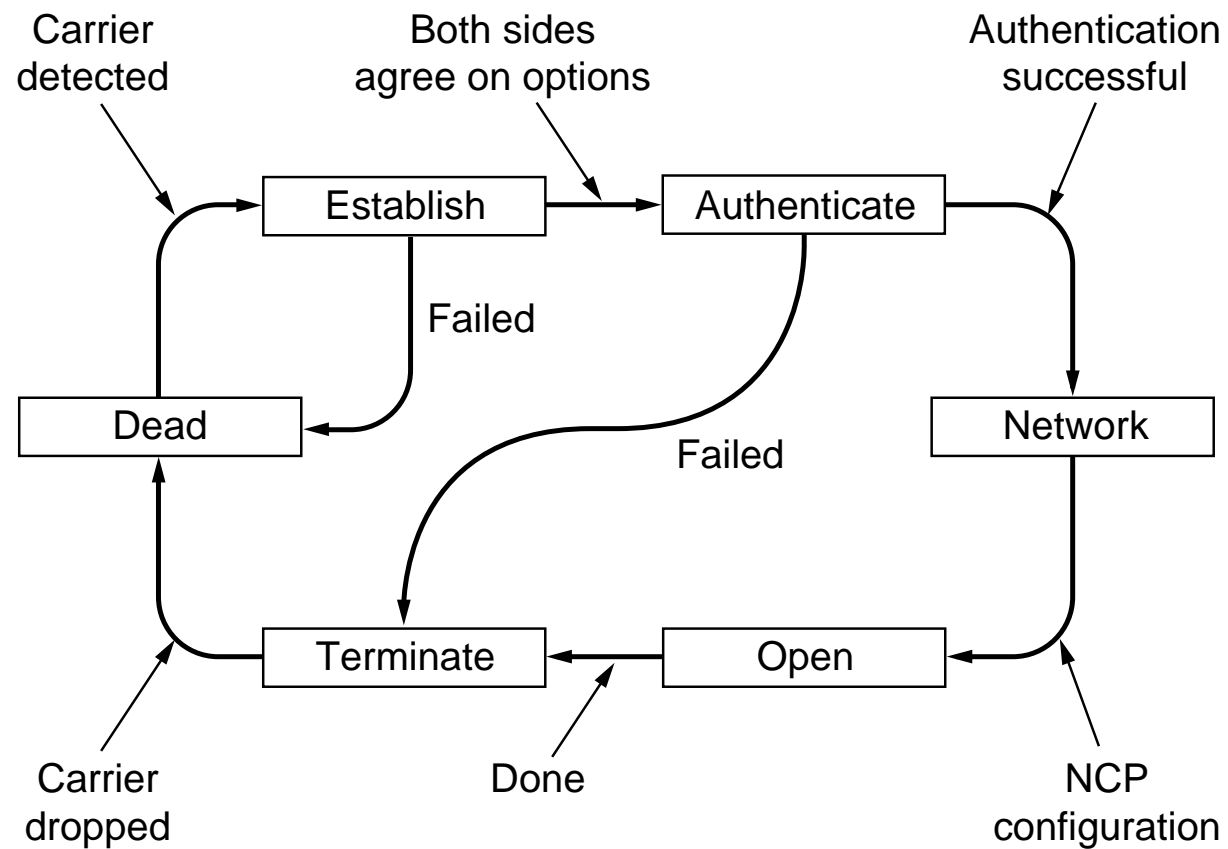


- Borrowed from HDLC
- LCP provides a way to negotiate away the Address and Control fields
- Protocol field
  - identifies upper level protocol
  - LCP, NCP, IP, IPX, AppleTalk, etc
- Payload
  - holds data
  - variable length up to a negotiated maximum
- Checksum (default is 2 bytes)

# PPP (4 of 5)

## LCP States (Simplified)

Starts in Dead state



# PPP (5 of 5)

## LCP Frame Types

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