COMP 3331/9331: Computer Networks and Applications

Week 8
Data link Layer

Reading Guide: Chapter 6, Sections 6.1 – 6.2

Link layer and LANs

our goals:

- understand principles behind link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - local area networks: Ethernet

Link layer, LANs: outline

6.1 introduction, services

6.2 error detection, correction

6.3 multiple access protocols

6.4 Switched LANs
- addressing, ARP
- Ethernet
- Switches
- VLANS (**EXCLUDED**)

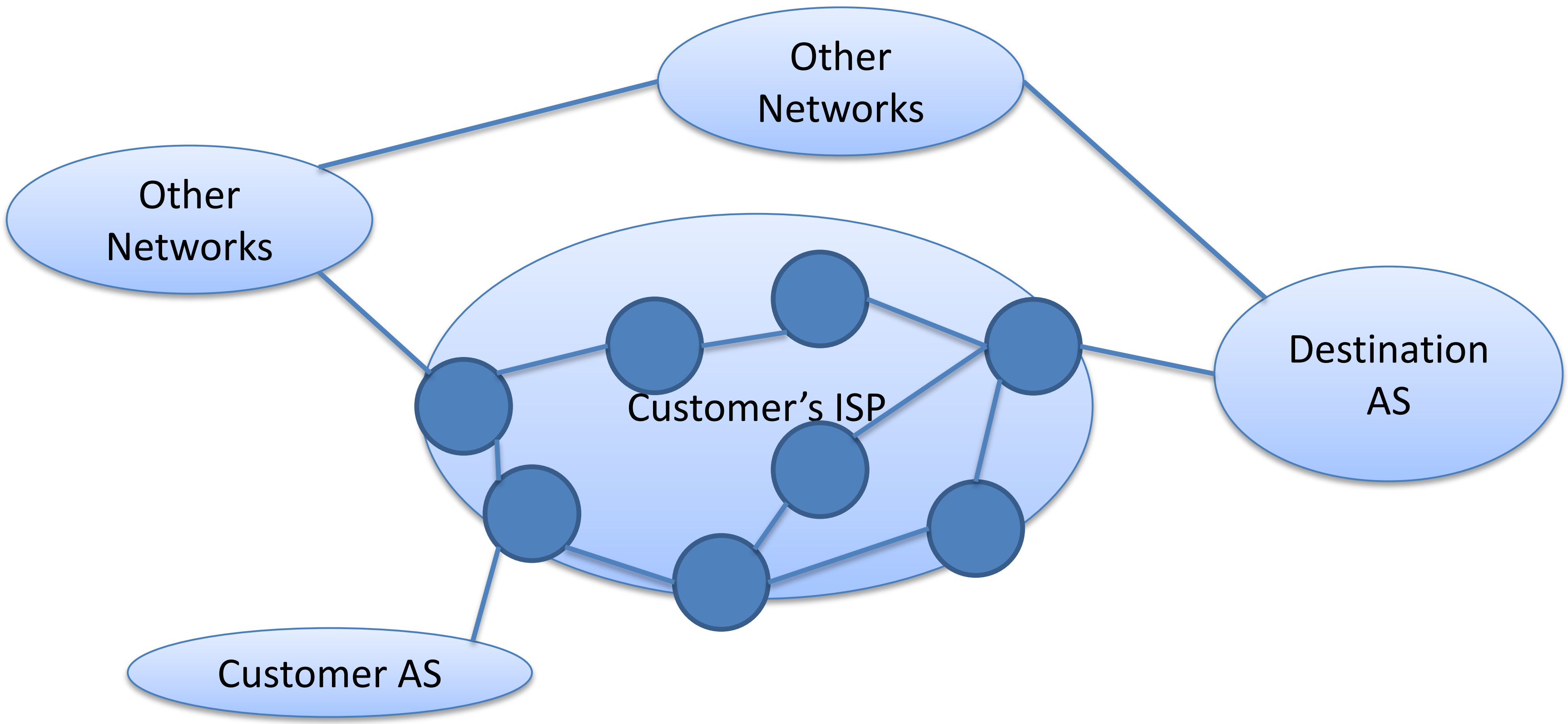
6.5 link virtualization: MPLS (**EXCLUDED**)

6.6 data center networking (**EXCLUDED**)

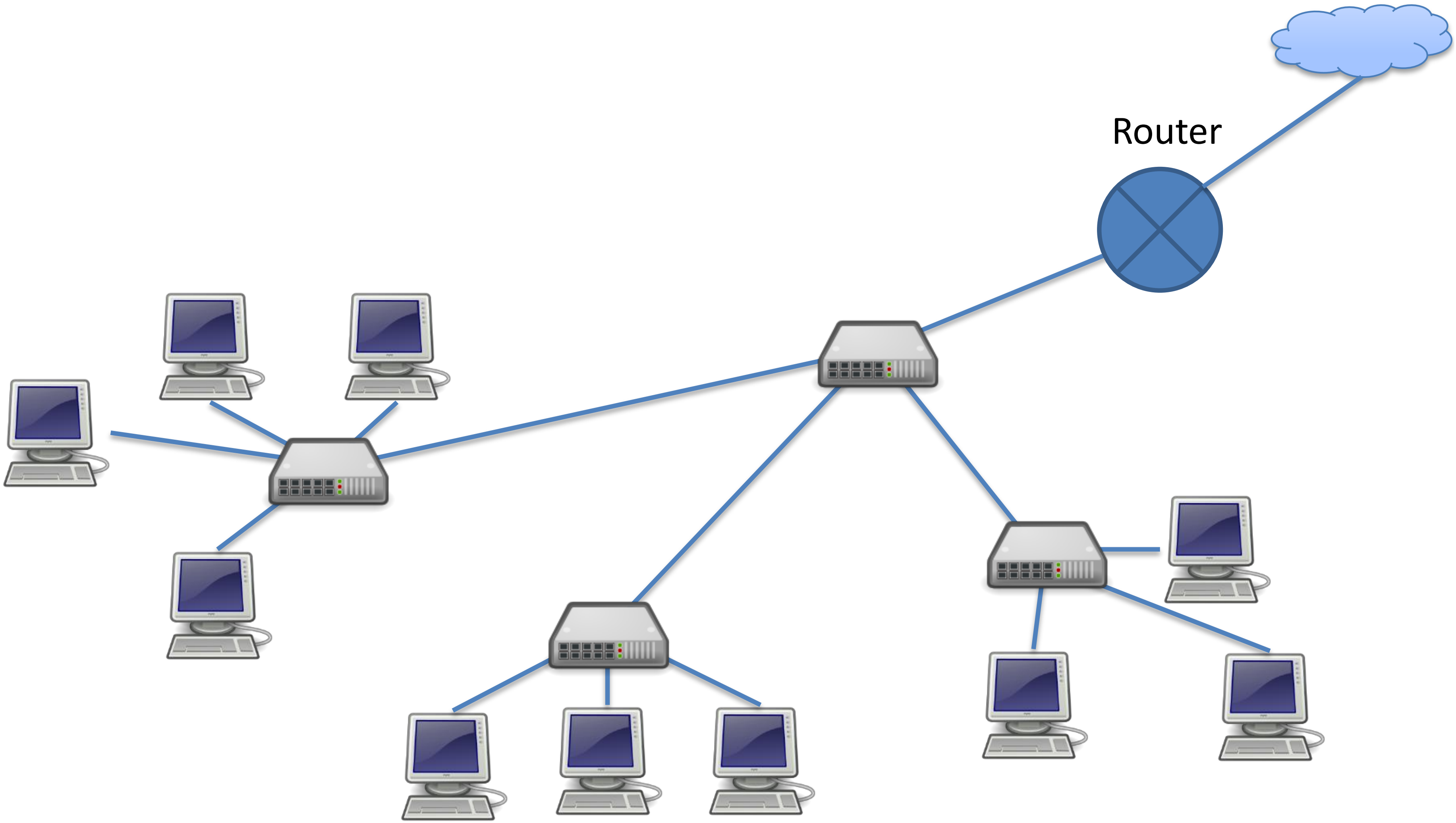
6.7 a day in the life of a web request

From Macro- to Micro-

- Previously, we looked at Internet scale…



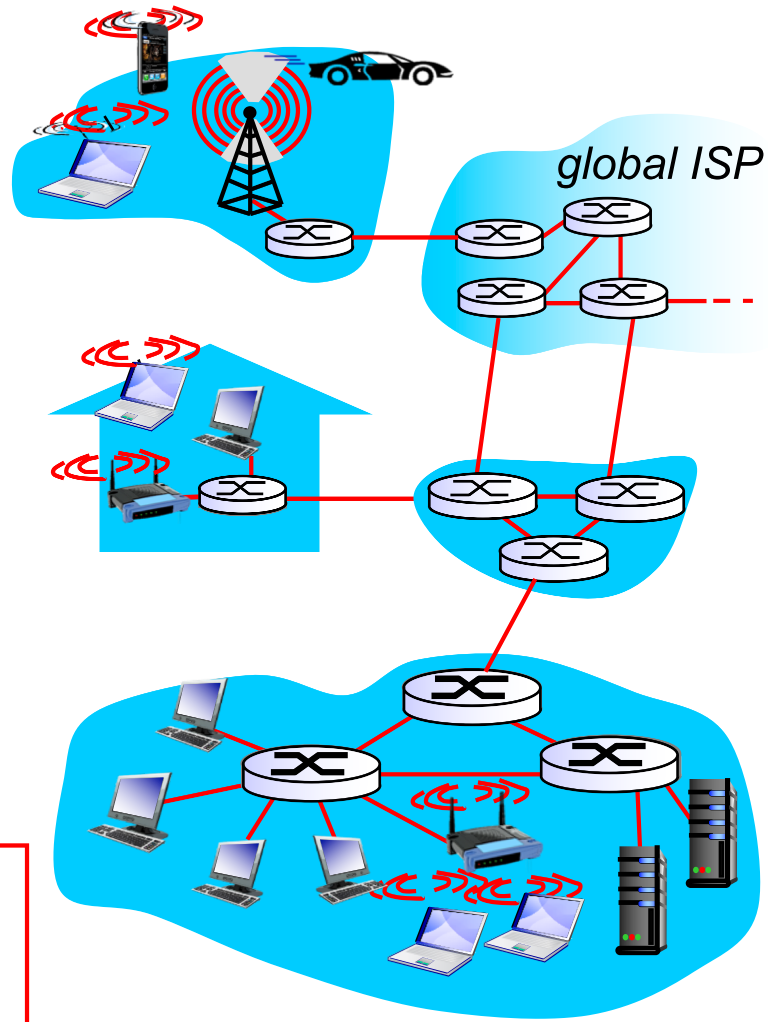
Link layer focus: Within a Subnet



Link layer: introduction

terminology:

- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired links
  - wireless links
  - LANs
- layer-2 packet: frame, encapsulates datagram



data-link layer has responsibility of transferring datagram from one node to *physically adjacent* node over a link

Link layer: context

- datagram transferred by different link protocols over different links:
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- each link protocol provides different services
  - e.g., may or may not provide rdt over link

transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link layer protocol
- travel agent = routing algorithm

Link layer services

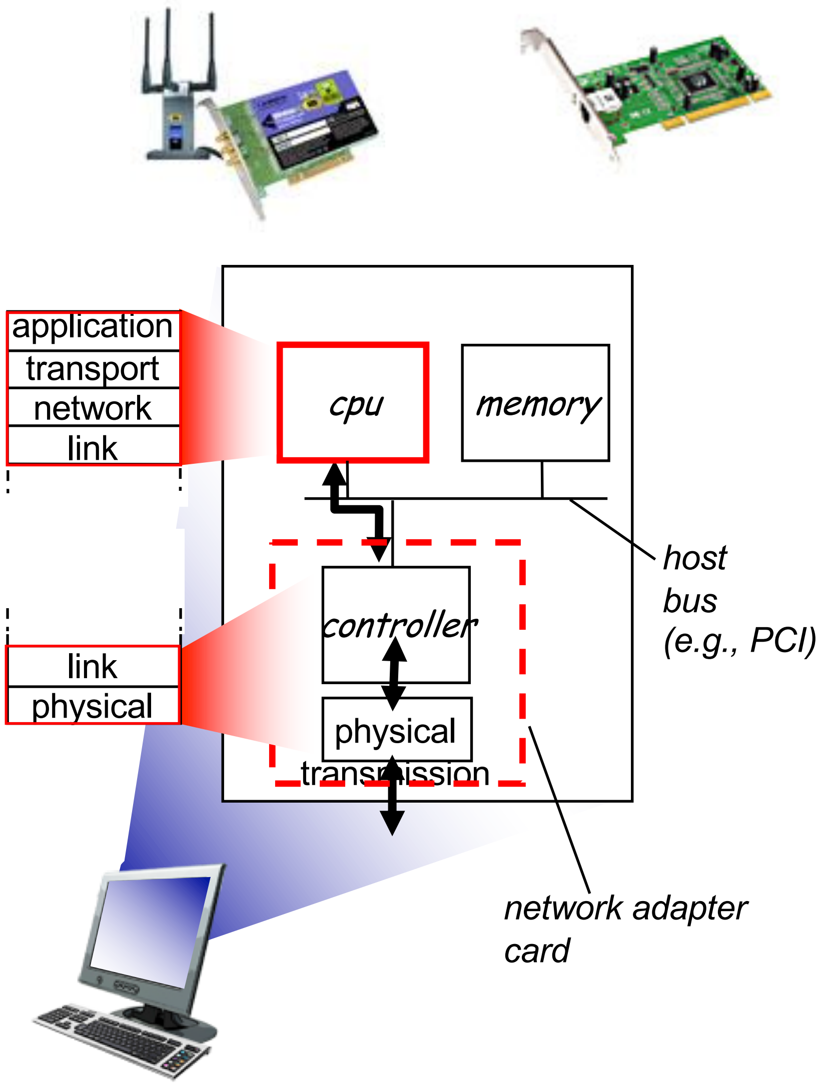
- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - we learned how to do this already (chapter 3)!
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - *Q:* why both link-level and end-end reliability?

Link layer services (more)

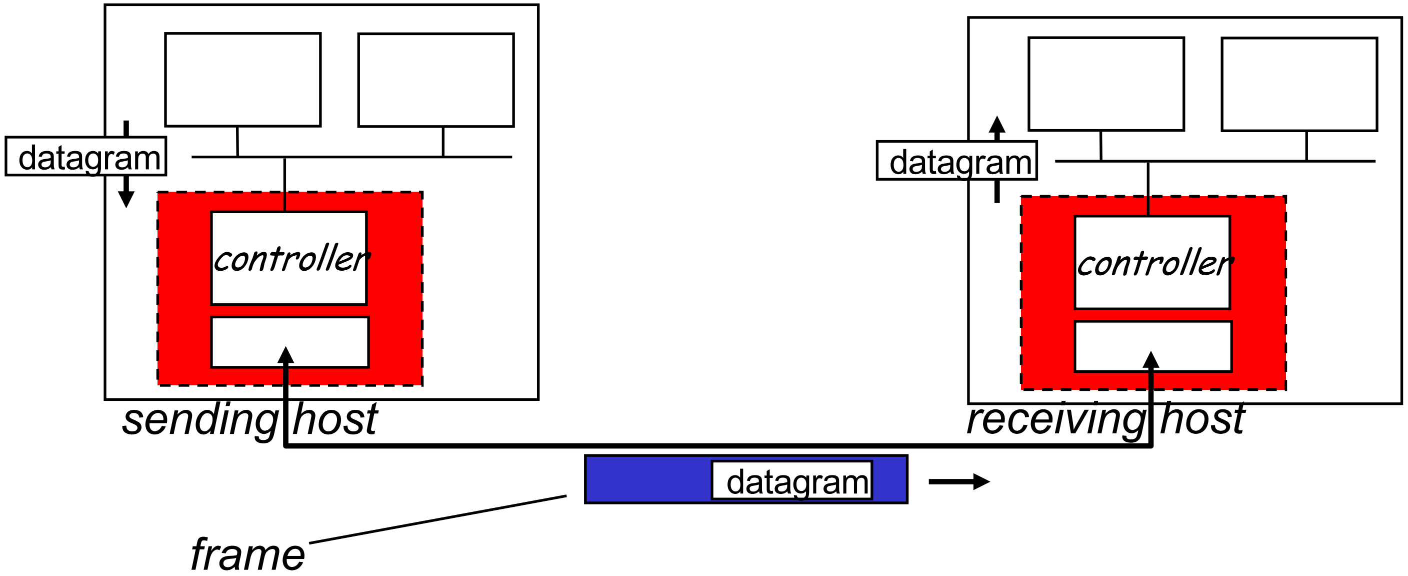
- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error detection:*
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- *error correction:*
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

Where is the link layer implemented?

- in each and every host
- link layer implemented in "adaptor" (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Adaptors communicating



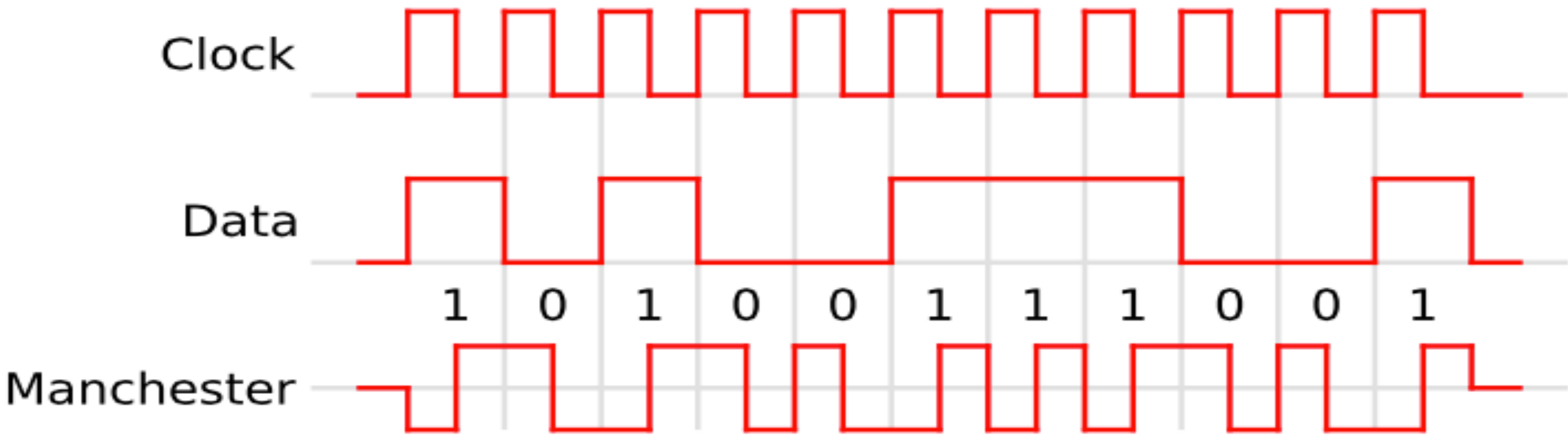
- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

What is framing?

- Physical layer talks in terms of bits.
- How do we identify frames within the sequence of bits?
- Need to do Framing
  - Delimit the start and end of the frame
- Ethernet Framing
  - Timing/Physical layer

Framing in Ethernet

- Start of frame is recognized by
  - Preamble : Seven bytes with pattern 10101010
  - Start of Frame Delimiter (SFD) : 10101011
- End of Frame: Absence of transition in Manchester encoded signal



Framing in Ethernet

- Start of frame is recognized by
  - Preamble : Seven bytes with pattern 10101010
  - Start of Frame Delimiter (SFD) : 10101011
- End of Frame: Absence of transition in Manchester encoded signal

Preamble 7 Bytes	**SFD 1 Byte**	**Dest MAC 6 Bytes**	**Source MAC 6 Bytes**	**Type/Length 2 Bytes**	**Payload 46-1500 Bytes**	**FCS/CRC 4 Bytes**	**Inter Frame Gap**

- Inter Frame Gap is 12 Bytes (96 bits) of idle state
  - 0.96 microsec for 100 Mbit/s Ethernet
  - 0.096 microsec for Gigabit/s Ethernet

Link layer, LANs: outline

6.1 introduction, services

6.2 error detection, correction

6.3 multiple access protocols

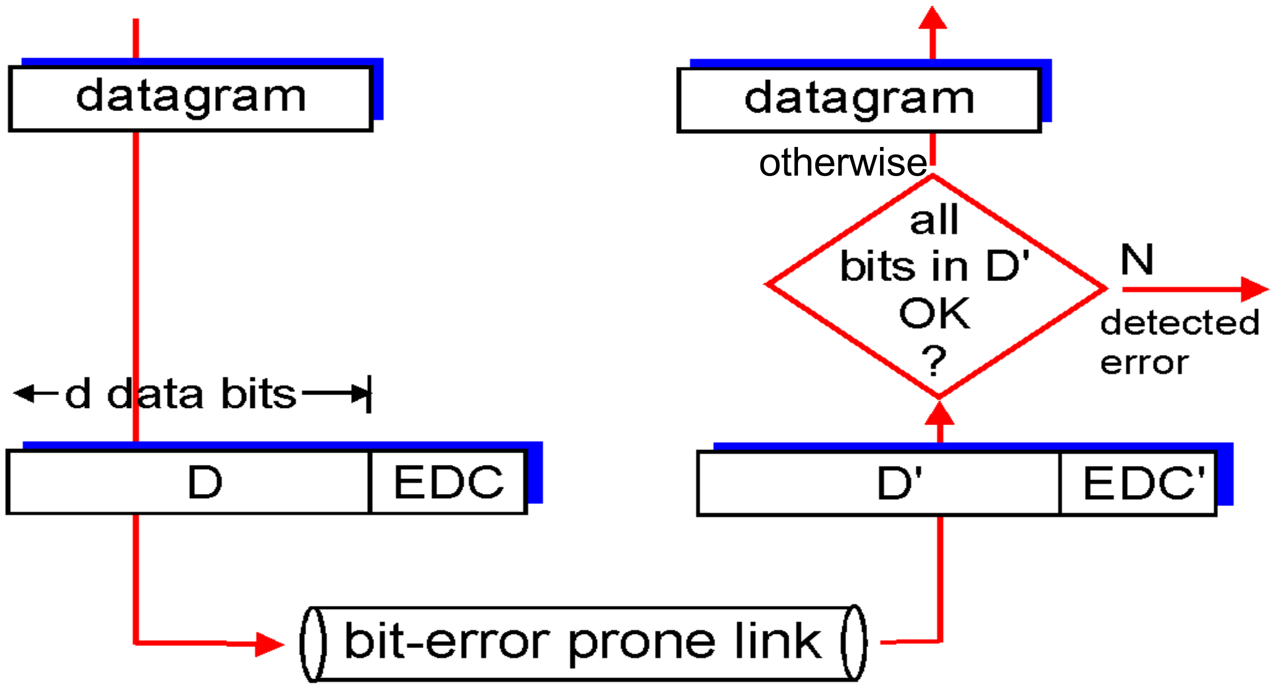
6.4 Switched LANs
- addressing, ARP
- Ethernet
- switches

6.7 a day in the life of a web request

Error detection

EDC= Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields

- Error detection not 100% reliable!
  - protocol may miss some errors, but rarely
  - larger EDC field yields better detection and correction



Error Detection

- Error coding
- Add check bits to the message bits to let some errors be detected and some be corrected
- How to structure the code to detect many errors with few check bits and modest computation?
- A simple code
  - Send two copies of the same message : 101101
  - Error if the copies are different : 101100
  - How many errors can it correct? 0
  - How many errors can it detect? Atmost 3
  - How many errors will make it fail? Specific 2 bits errors
  - What is the overhead? 50%

Simple Parity - Sender

- Suppose you want to send the message:
  - 0010110110110001100110010
- For every *d* bits (e.g., *d* = 7), add a parity bit:
  - 1 if the number of one's is odd
  - 0 if the number of one's is even

Message chunk	Parity bit
0010110	1
1101100	0
0110010	1

– 0010110<u>1</u>1101100<u>0</u>0110010<u>1</u>

Simple Parity - Receiver

- For each block of size *d*:
  - Count the number of 1's and compare with following parity bit.
- If an odd number of bits get flipped, we'll detect it (can't do much to correct it).
- Cost: One extra bit for every *d*
  - In this example, 21 -> 24 bits.

Two-Dimensional Parity

- Suppose you want to send the same message:
  - 0010110110110000110010
- Add an extra parity byte, compute parity on "columns" too.
- Can detect 1, 2, 3-bit (and some 4-bit) errors

	Message chunk	Parity bit
	0010110	1
	1101100	0
	0110010	1
Parity byte:	1001000	0

Forward Error Correction

- With two-dimensional parity, we can even *correct* single-bit errors.

Parity bits

	0	0	1	0	1	1	0	1
	1	0	1	0	0	0	1	0
	1	0	0	1	0	1	1	0
	1	1	1	0	1	1	0	1
Parity byte	1	1	1	1	1	1	0	0

Exactly one bit has been flipped. Which is it?

In practice

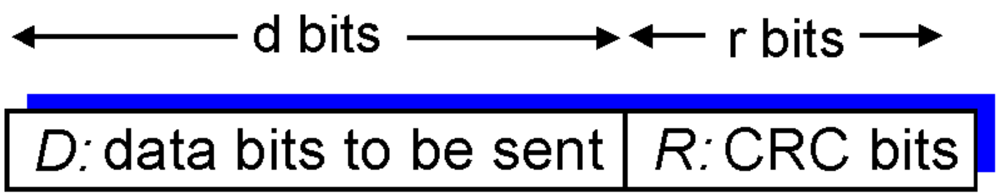
- Bit errors occur in bursts.
- We're willing to trade computational complexity for space efficiency.
  - Make the detection routine more complex, to detect error bursts, without tons of extra data
- Insight: We need hardware to interface with the network, do the computation there!

Error Detection and Correction

- Checksum
  - Sum up data in N-bit words
  - Internet Checksum uses 16 bit words
- How well checksum works?
  - How many errors can it detect/correct? 1, 0
- What have we gained as compared to parity bit?
  - Can now detect all burst errors up to 16

Cyclic redundancy check

- more powerful error-detection coding
- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi, ATM)



bit pattern

$D * 2^r$ XOR R

mathematical formula

Cyclic redundancy check

- Sender operation
  - Extend D data bits with R zeros
  - Divide by generator G
  - Keep remainder, ignore quotient
  - Adjust R check bits by the remainder
- Receiver Procedure
  - Divide and check for zero remainder

CRC example

want:

$$D \cdot 2^r \text{ XOR } R = nG$$

equivalently:

$$D \cdot 2^r = nG \text{ XOR } R$$

equivalently:

if we divide $D \cdot 2^r$ by G, want remainder R to satisfy:

$$R = \textit{remainder}\left[\frac{D \cdot 2^r}{G}\right]$$

G D $r = 3$

```
            101011
1001 ) 101110000
       1001
        101
        000
        1010
        1001
         110
         000
         1100
         1001
          1010
          1001
           011
```

R

Modulo-2 Arithmetic

- All calculations are modulo-2 arithmetic
- No carries or borrows in subtraction
- Addition and subtraction are identical and both are equivalent to XOR
  - 1011 XOR 0101 = 1110
  - 1011 - 0101 = 1110
  - 1011 + 0101 = 1110
- Multiplication by 2^k is essentially a left shift by *k* bits
  - 1011 x 2^2 = 101100