Computer Networks COL 334/672

Link Layer

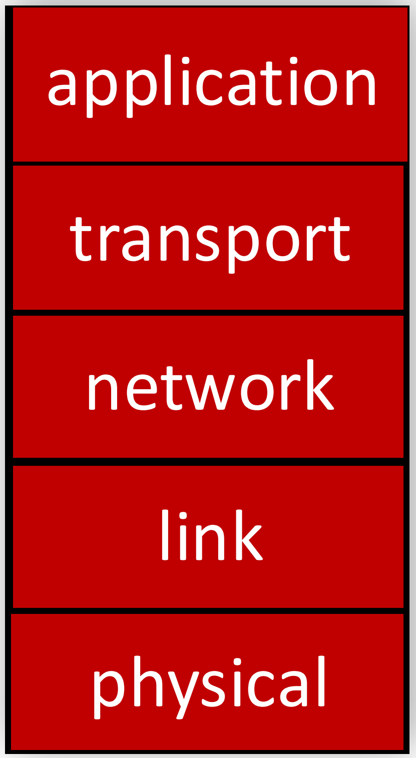
Tarun Mangla

Slides adapted from KR

Sem 1, 2024-25

Layered Internet protocol stack

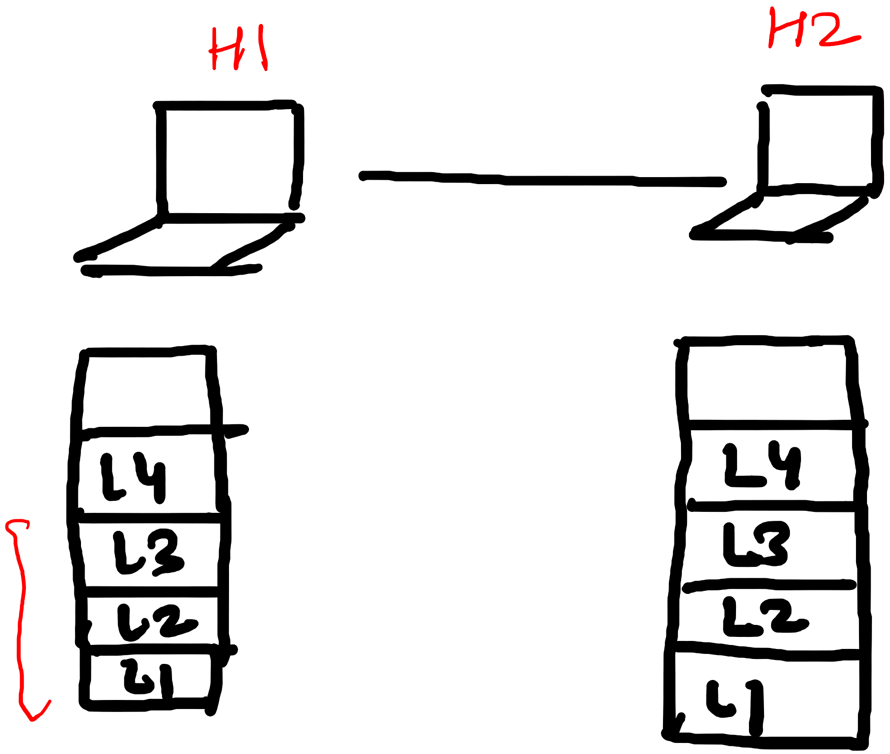
- *application:* supporting network applications
  - HTTP, IMAP, SMTP, DNS
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits "on the wire"



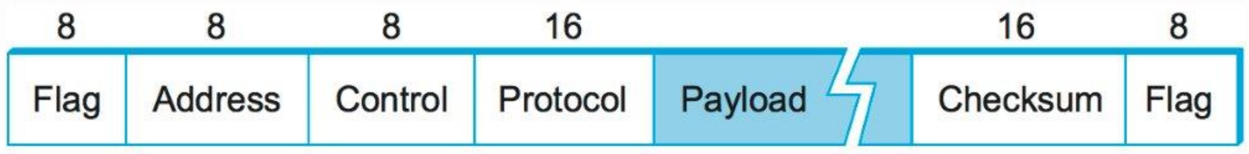
Link Layer: Services

layer-2 packet: *frame*, encapsulates datagram

- Framing
- Error detection
- Reliability
- Link access

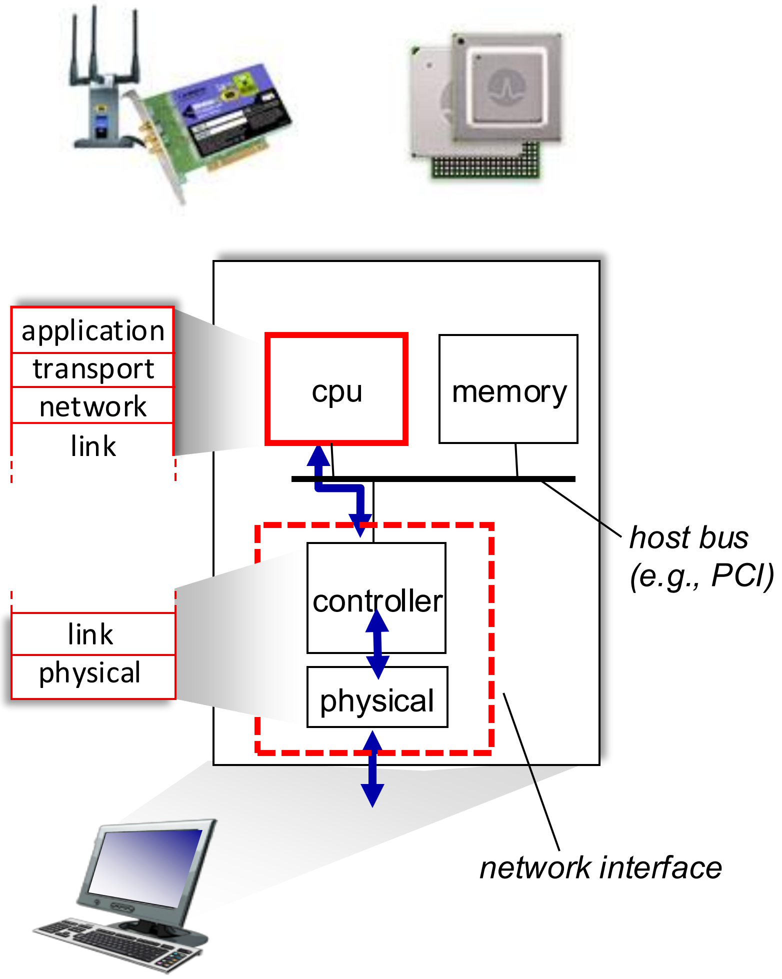


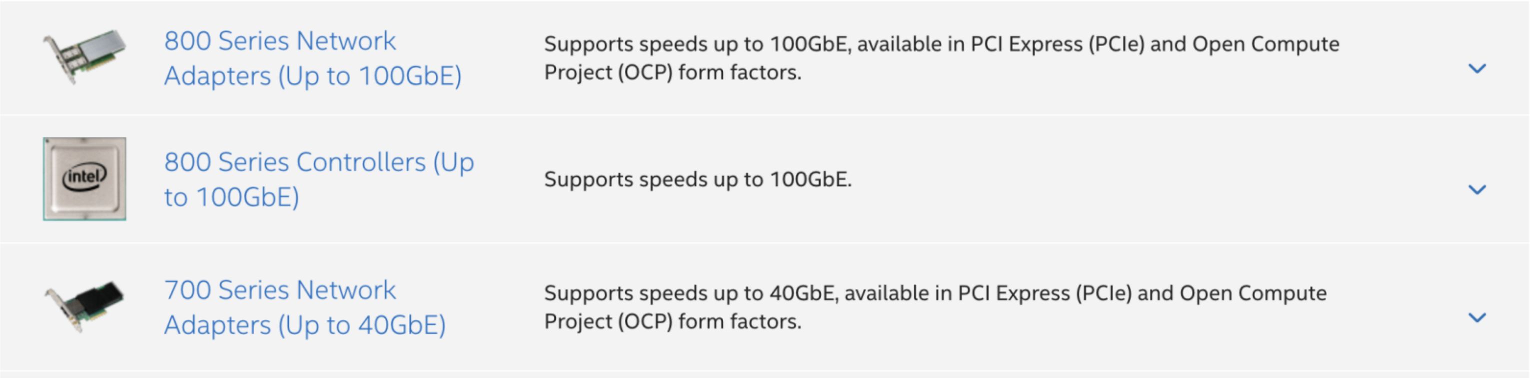
link layer has responsibility of transferring datagram from one node to physically adjacent node over a link



Where is the Link Layer?

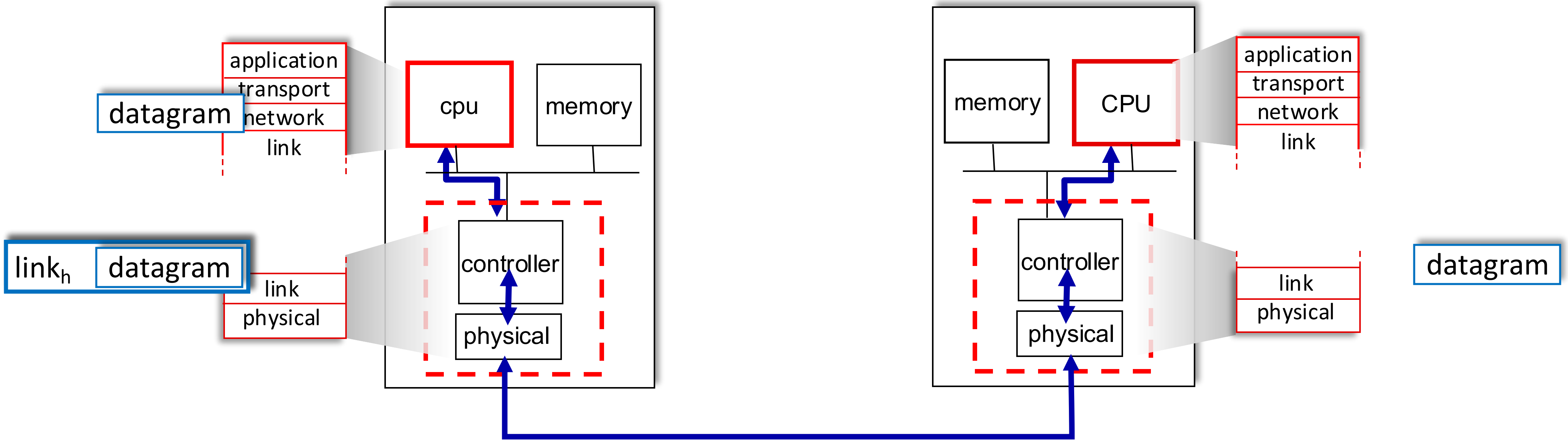
- in each-and-every node
- link layer implemented on-chip or in network interface card (NIC)
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware





800 Series Network Adapters (Up to 100GbE)	Supports speeds up to 100GbE, available in PCI Express (PCIe) and Open Compute Project (OCP) form factors.
800 Series Controllers (Up to 100GbE)	Supports speeds up to 100GbE.
700 Series Network Adapters (Up to 40GbE)	Supports speeds up to 40GbE, available in PCI Express (PCIe) and Open Compute Project (OCP) form factors.

Interfaces communicating



sending side:

- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

Framing

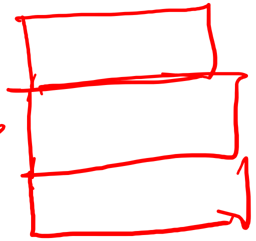
- Sender: Encapsulate datagram into frames
- Receiver: Assemble bitstream into frames
- ***Challenge: How to detect frame boundaries?***

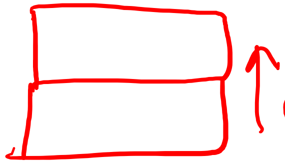
Hi

Datagram L3

Frame L2

Bitstream



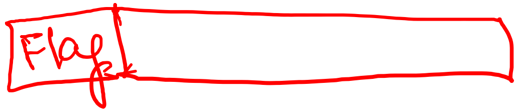


Frame

Bitstream

1101 . . . . 01 | 10 . . . .

F1 F2

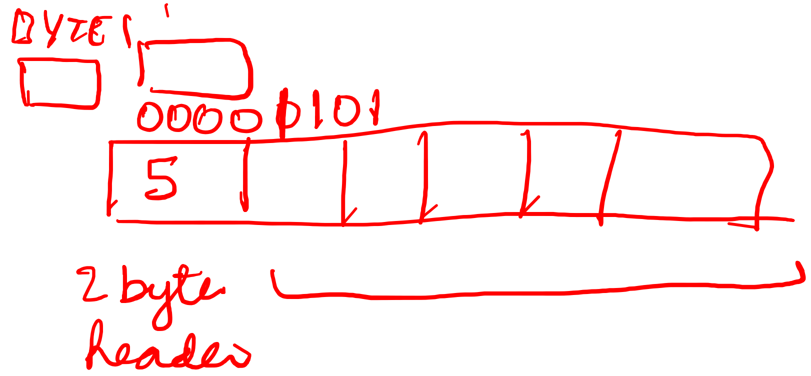


1111

① Special Token

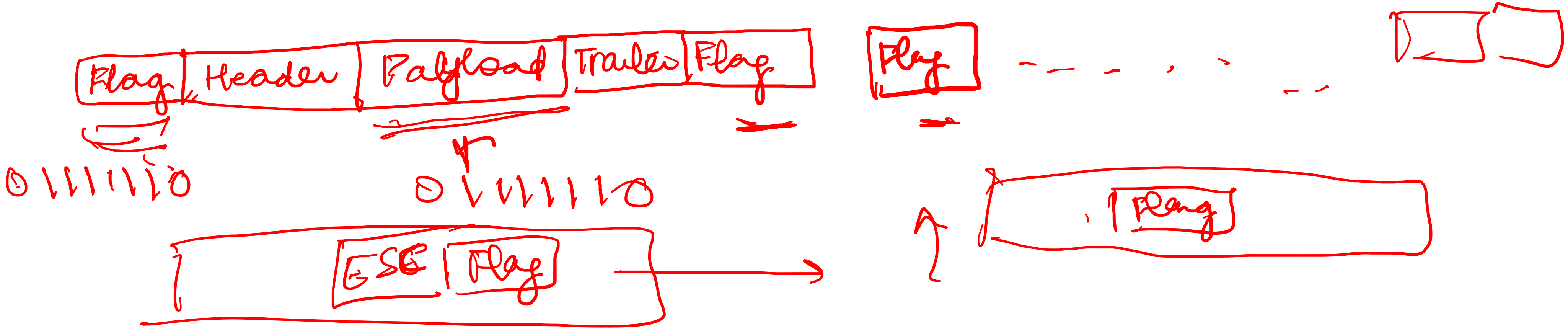
② Using a header that specifies the length of the frame

Frame Boundary Detection



- Including number of bytes in the header
  - Can lead to framing errors in case of bit errors

- Sentinel approach
  - Use special token to denote start and end of frame or sent (e.g., 01111110)
  - What happens when the token appears in the payload?
  - Use esc character or bit stuffing
  - Used in High-level data link control (HDLC) protocol



Bit stuffing algorithm

→ 01111110 → Flag

Preamble | SD

- **Sender side**
  - If see 5 consecutive 1s then insert a zero after them

FRAME

Flag | Header | Payload | Trailer

~~Flag~~

Data: 0111011111100

011111000 → 0111110000

Sent data

01111110 01110111110100

Receive

01111110 0111011111100

- **Receiver side**
  - If see 5 consecutive 1s then remove the stuffed bit 0 following them

Error Detection

- There can be bit errors as frames are transmitted
- **Challenge:** How to detect bit errors?

n bits + k bits

Redundant bits

n bits

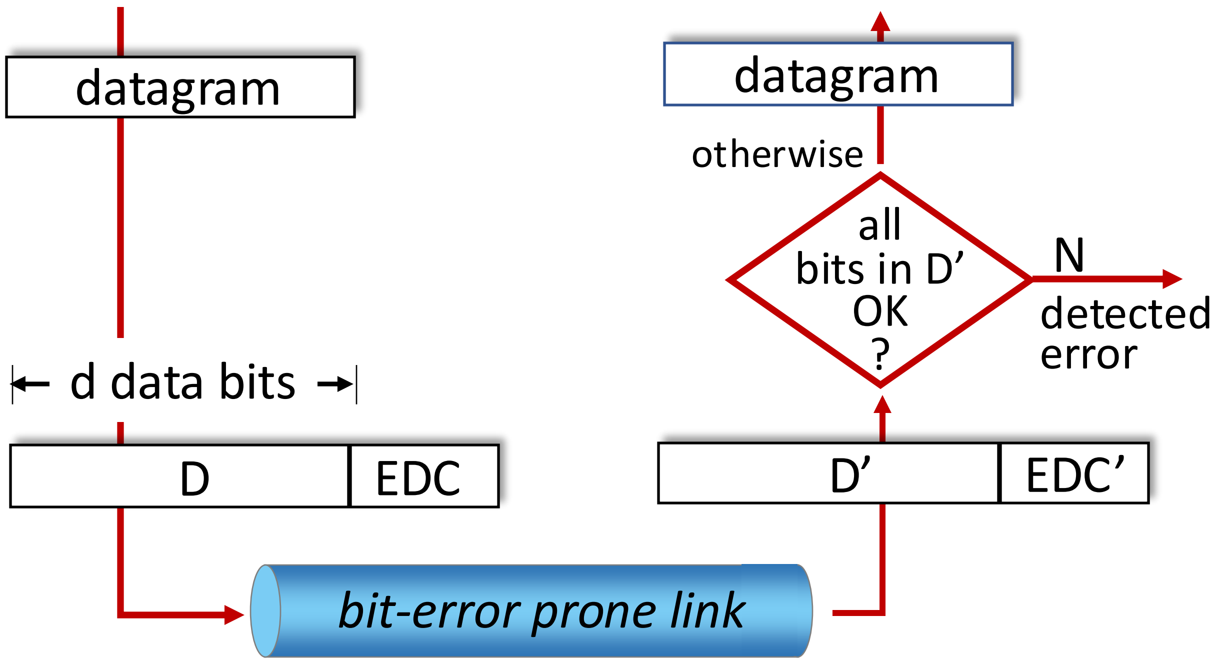
n bits

↳ Not efficient

Error detection

EDC: error detection and correction bits (e.g., 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 approaches

- Simplest approach
  - Send copy of data
  - In-efficient
  - Errors can go undetected
- Parity checking

Parity checking

single bit parity:

- detect single bit errors

0111000110101011	1

← *d* data bits →

parity bit

Even/odd parity: set parity bit so there is an even/odd number of 1's

At receiver:

- compute parity of *d* received bits
- compare with received parity bit – if different than error detected

Example

D: 1101 and even parity

What kind of errors does it work best for?

Even parity

Sender | 1 0 1 1 | 1 |

Receiver

| 1 1 1 0 | 1 | ∝ bit error

| 1 1 0 1 | 0 |

↳ Medium bit errors are small 0.2%

↳ Not good enough, bit errors occur in burst

Checksum

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment

sender:

- treat content as sequence of 16-bit integers
- checksum: addition (one's complement sum) of content
- checksum value put into checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected.

Checksum Example

4-bit check sum

Chksum

Data 1101 1001 0001 1000

sum: 1101

1001

0110

sum: 0111

0001

1000

100100

1001

1001

0100 → 0111

1100

1000