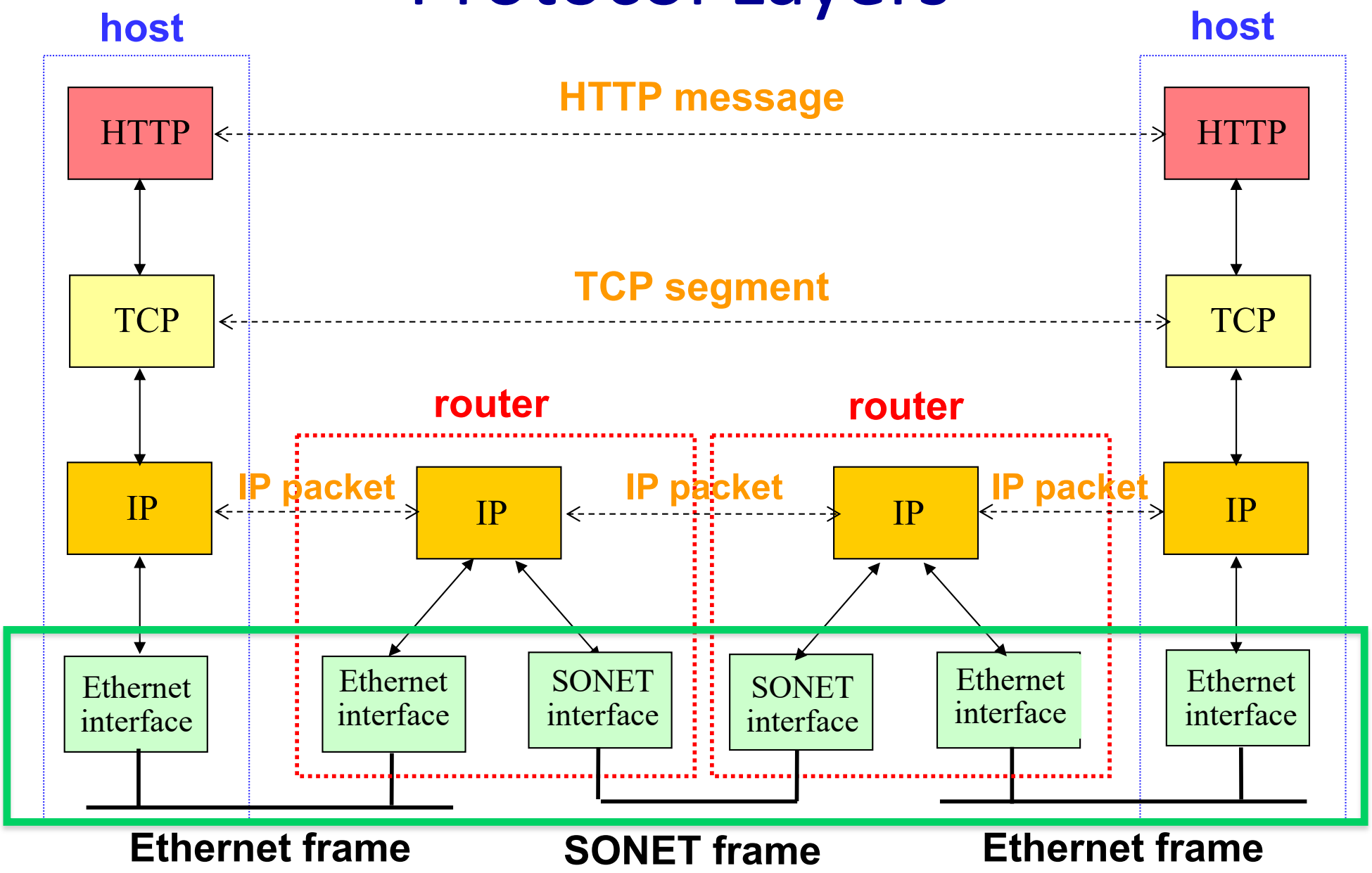
Protocol Layers



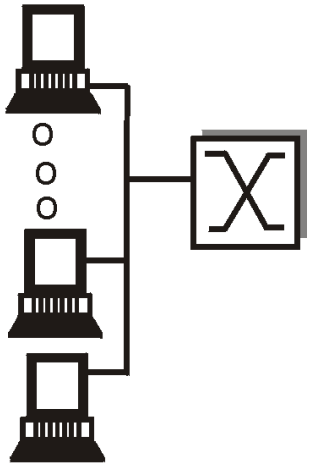
Link = Medium + Adapters

What is a Link?

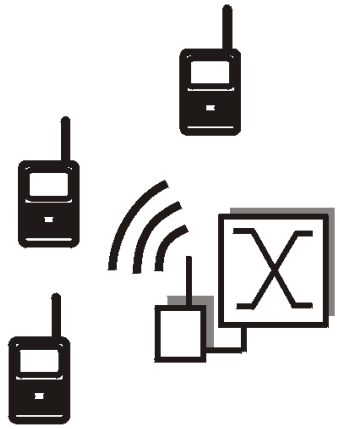
Communication Medium

Network Adapter

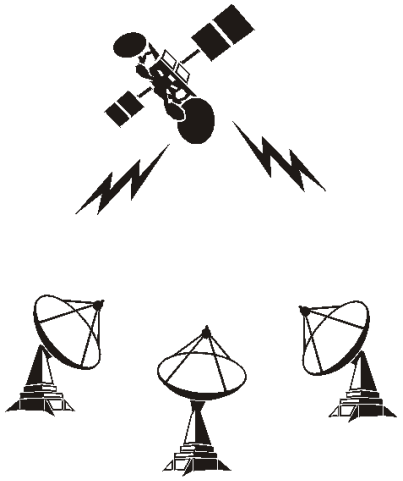
Broadcast Links: Shared Media



shared wire
(e.g. Ethernet)



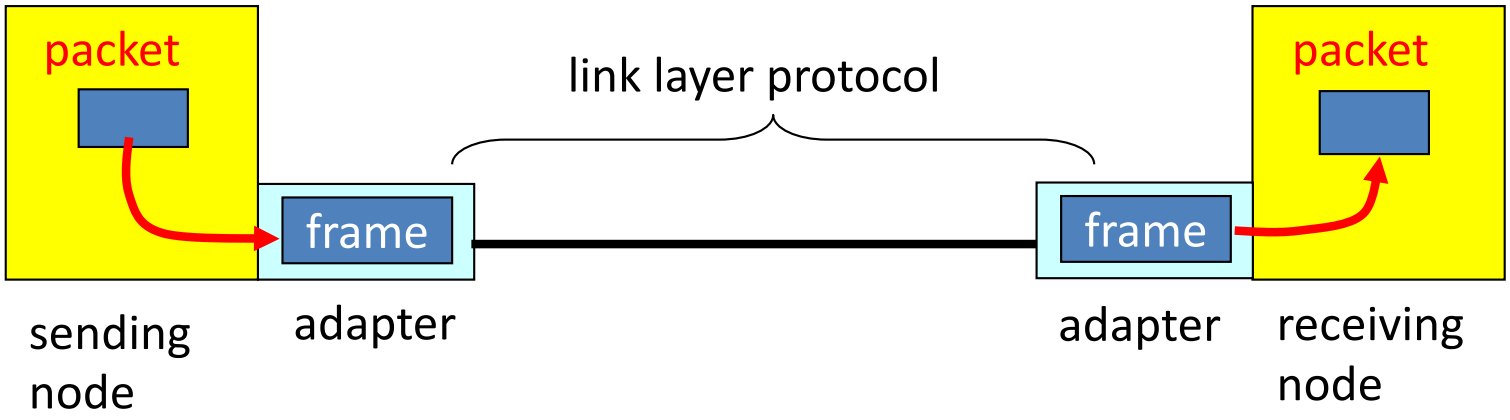
shared wireless
(e.g. Wavelan)



satellite

cocktail party

Adaptors Communicating



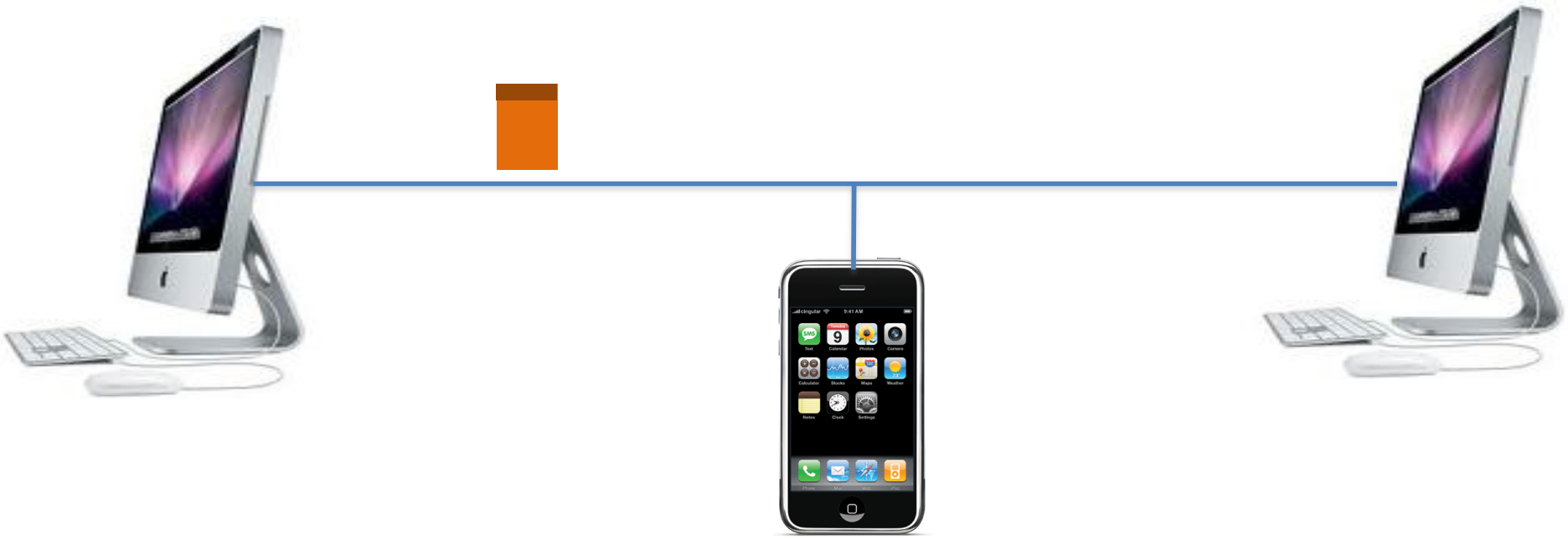
- Sending side
  - Encapsulates packet in a frame
  - Adds error checking bits, flow control, etc.
- Receiving side
  - Looks for errors, flow control, etc.
  - Extracts datagram and passes to receiving node

Link-Layer Services

- Encoding
  - Represent the 0s and 1s
- Framing
  - Encapsulate packet into frame, adding header/trailer
- Error detection
  - Receiver detecting errors with checksums
- Error correction
  - Receiver optionally correcting errors
- Flow control
  - Pacing between sending and receiving nodes

Addresses

Medium Access Control Address



- Identify the sending and receiving adapter
  - Unique identifier for each network adapter
  - Identifies the intended receiver(s) of the frame
  - … and the sender who sent the frame

Medium Access Control Address

- MAC address (e.g., 00-15-C5-49-04-A9)
  - Numerical address used within a link
  - Unique, hard-coded in the adapter when it is built
  - Flat name space of 48 bits
- Hierarchical allocation: Global uniqueness!
  - Blocks: assigned to vendors (e.g., Dell) by the IEEE
  - Adapters: assigned by the vendor from its block
- Broadcast address (i.e., FF-FF-FF-FF-FF-FF)
  - Send the frame to *all* adapters

As an Aside: Promiscuous Mode

- Normal adapter: receives frames sent to
  - The local MAC address
  - Broadcast address FF-FF-FF-FF-FF-FF
- Promiscuous mode
  - Receive *everything*, independent of destination MAC
- Useful for packet sniffing
  - Network monitoring
  - E.g., wireshark, tcpdump

Why Not Just Use IP Addresses?

- Links can support *any* network protocol
  - Not just for IP (e.g., IPX, Appletalk, X.25, …)
  - Different addresses on different kinds of links

Because links can support different protocols, not all devices or network setups will use IP addresses. Therefore, using only IP addresses would be limiting and wouldn't accommodate all possible network scenarios.

- An adapter may move to a new location
  - So, cannot simply assign a static IP address
  - Instead, must reconfigure the adapter's IP address

If the adapter has a static IP address (a fixed IP address manually assigned), it might not work correctly in the new network because IP addresses are typically network-specific. For example, a static IP address assigned in one network might conflict with the IP addressing scheme in another network.

- Must identify the adapter during bootstrap
  - Need to talk to the adapter to assign it an IP address

The device or adapter does not have an IP address when it first connects to a network (in most cases), so the network needs to communicate with it in some other way (e.g., using MAC addresses) to assign it an IP address.

Who Am I: Acquiring an IP Address

71-65-F7-2B-08-53

????

1A-2F-BB-76-09-AD

DHCP server

- Dynamic Host Configuration Protocol (DHCP)
  - Broadcast “I need an IP address, please!”
  - Response “You can have IP address 1.2.3.4.”

Who Are You: Discovering the Receiver

71-65-F7-2B-08-53
1.2.3.4

0C-C4-11-6F-E3-98
1.2.3.6

1A-2F-BB-76-09-AD
1.2.3.5

- Address Resolution Protocol (ARP)
  - Broadcast “who has IP address 1.2.3.6?”
  - Response “0C-C4-11-6F-E3-98 has 1.2.3.6!”

Sharing the Medium

Collisions

71-65-F7-2B-08-53

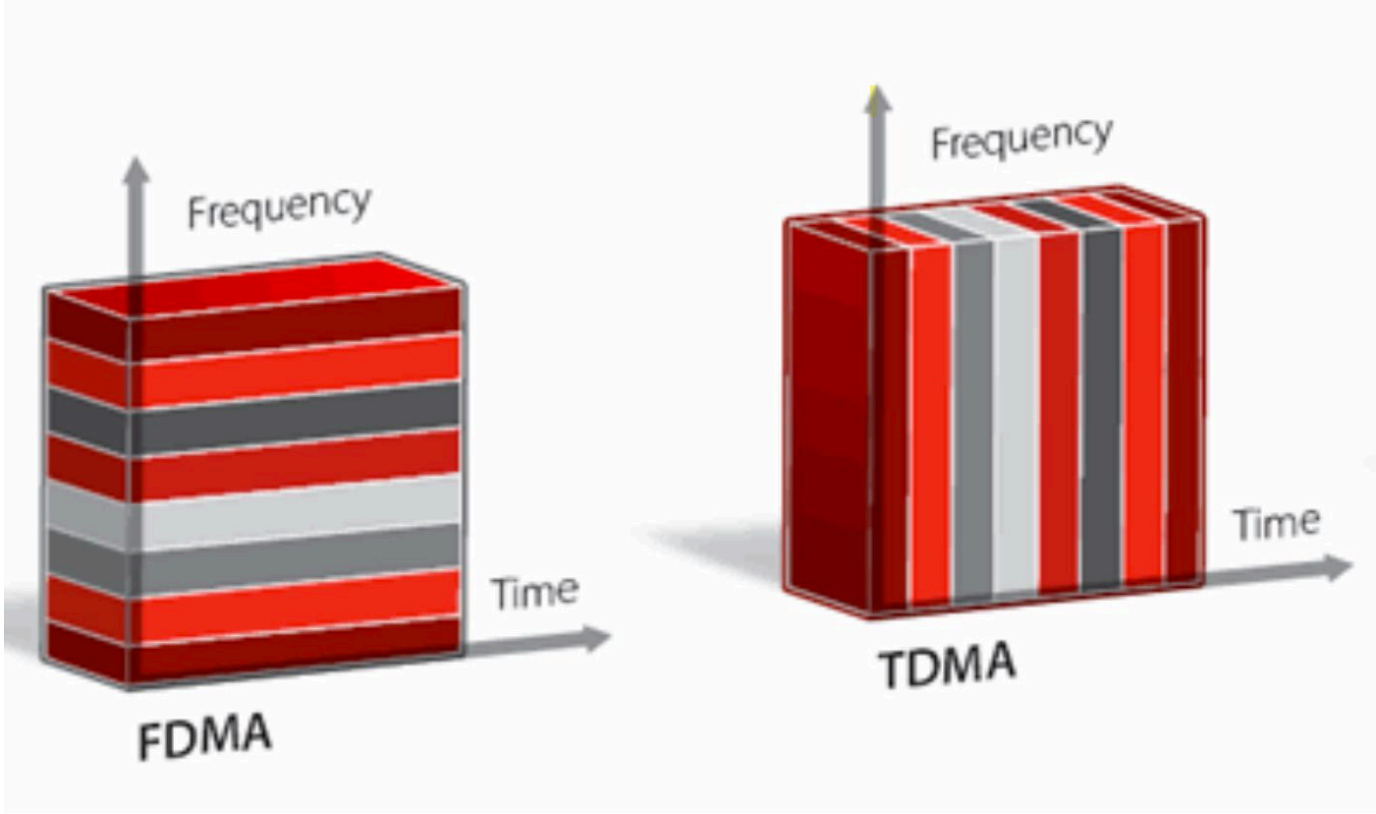
0C-C4-11-6F-E3-98

1A-2F-BB-76-09-AD

- Single shared broadcast channel
  - Avoid having multiple nodes speaking at once
  - Otherwise, collisions lead to garbled data

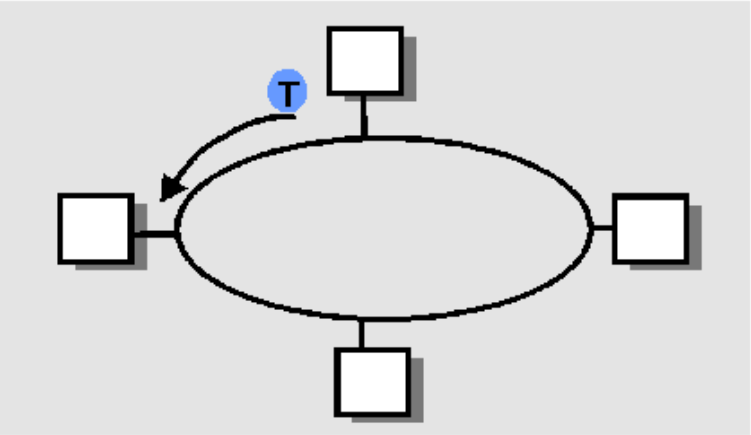
Multi-Access Protocol

1. Divide the channel into pieces
   - in frequency vs. in time



Multi-Access Protocol

- Take turns
  - Do not transmit w/o token
  - With token, transmit for up to some max time/length
  - Pass token to right



- Punt
  - Let collisions happen
  - … and detect and recover from them

Like Human Conversation…

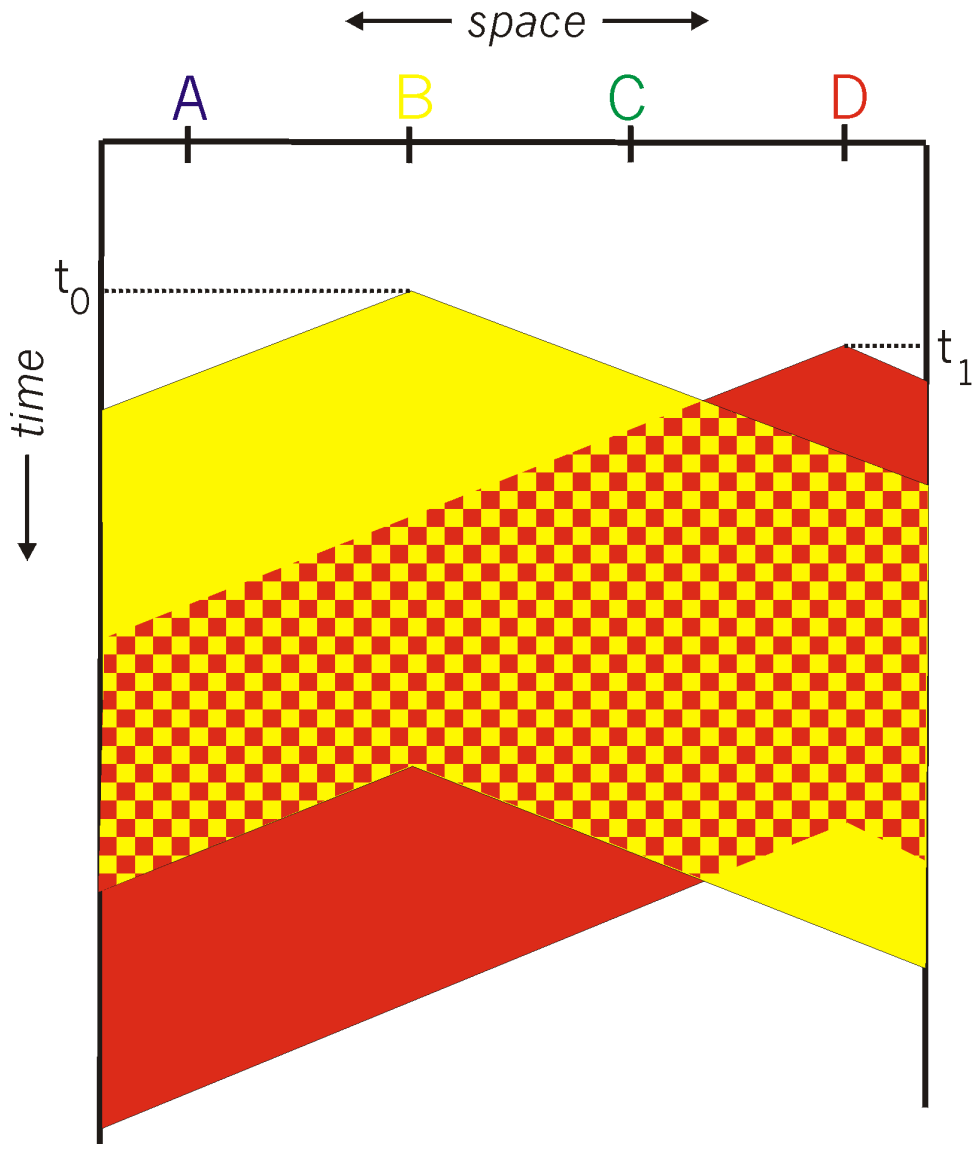
- Carrier sense
  - Listen before speaking
  - …and don't interrupt!

- Collision detection
  - Detect simultaneous talking
  - … and shut up!

- Random access
  - Wait for a random period of time
  - … before trying to talk again!

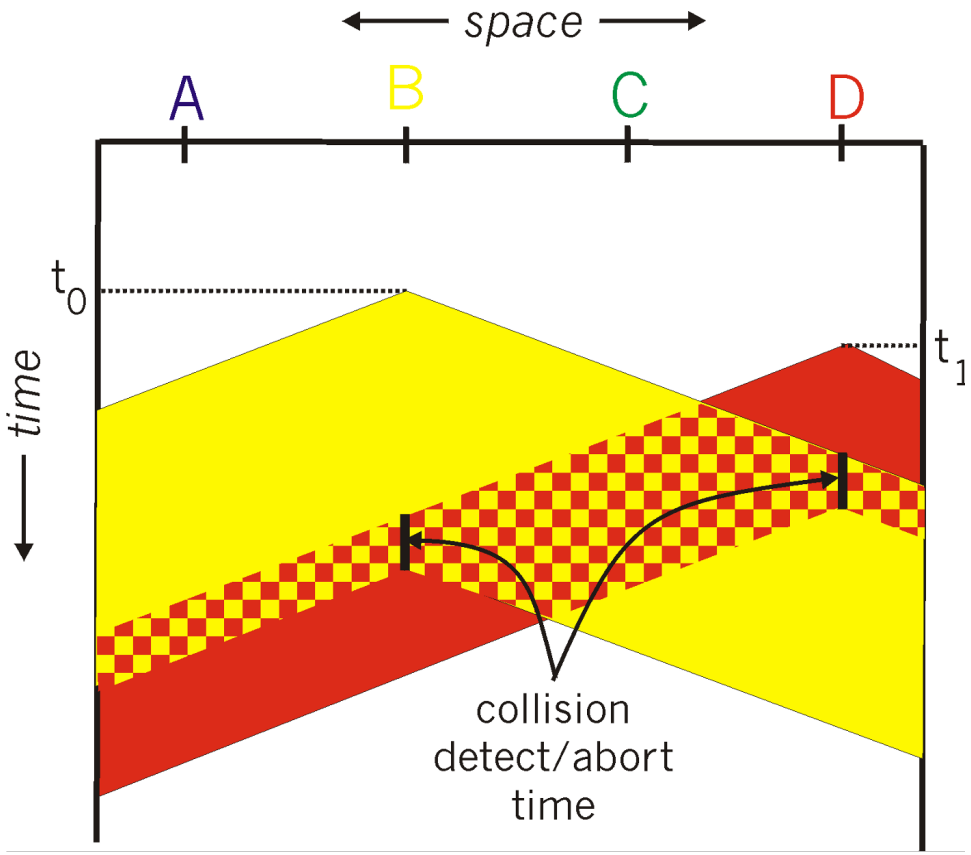
Carrier Sense Multiple Access

- Listen for other senders
  - Then transmit your data
- Collisions can still occur
  - Propagation delay
  - Wasted transmission



CSMA/CD Collision Detection

- Detect collision
  - Abort transmission
  - Jam the link
- Wait random time
  - Transmit again
- Hard in wireless
  - Must receive data while transmitting



Comparing the Three Approaches

- Channel partitioning is
  - (a) Efficient/fair at high load, inefficient at low load
  - (b) Inefficient at high load, efficient/fair at low load

When few users or devices are active, channel partitioning is inefficient because it still dedicates portions of the channel to each user, even if they don't need it, leading to underutilization of the available bandwidth.

- "Taking turns"
  - (a) Inefficient at high load
  - (b) Efficient at all loads
  - (c) Robust to failures

At high load: These methods are efficient because they avoid collisions and ensure that each device gets a turn to transmit without interference.

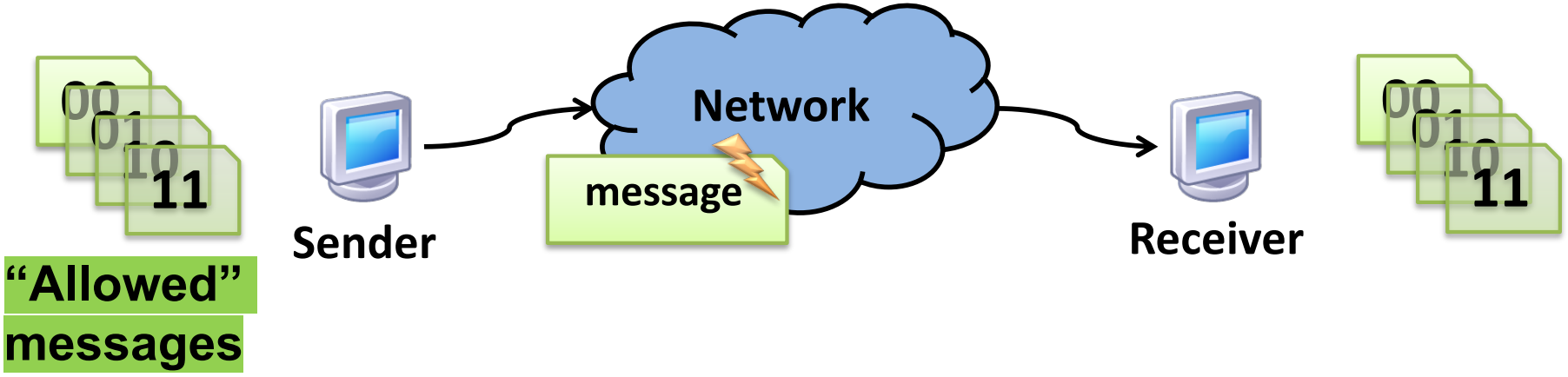
At low load: They remain efficient because even if there are few devices, each device can still transmit as soon as its turn comes up, and the channel remains fully utilized without idle time.

Robust to Failures: However, they are also robust to certain types of failures (like token loss in token-passing), but the primary focus here is on their efficiency.

- Random access
  - (a) Inefficient at low load
  - (b) Efficient at all load
  - (c) Robust to failures

Error Control

Error control: Motivation

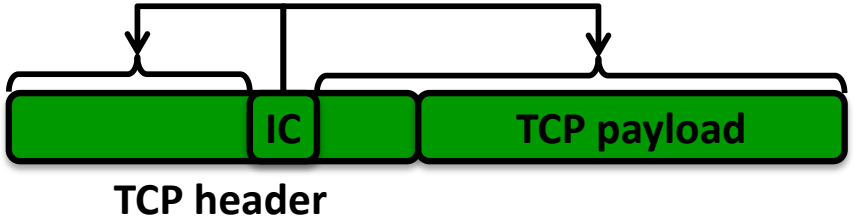


- *A priori,* any string of bits is an "allowed" **message**
  - Hence any **changes to the bits** (***bit errors***) the sender transmits produce "**allowed**" messages

- **Therefore without error control, receiver wouldn't know errors happened!**

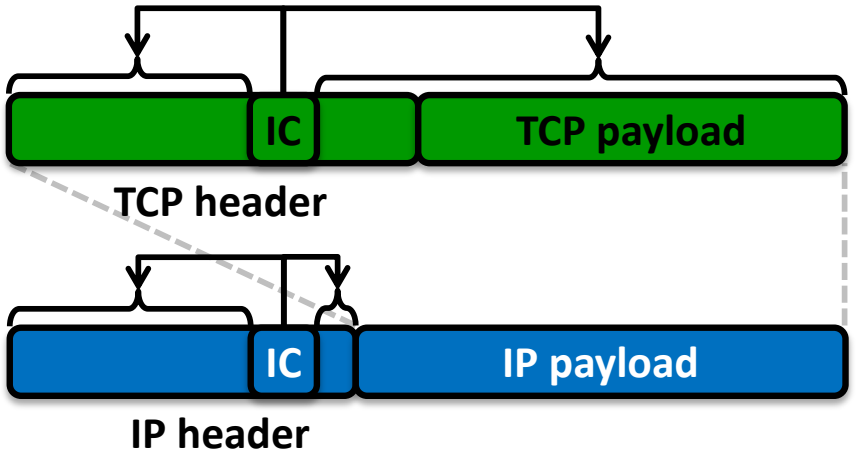
Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data



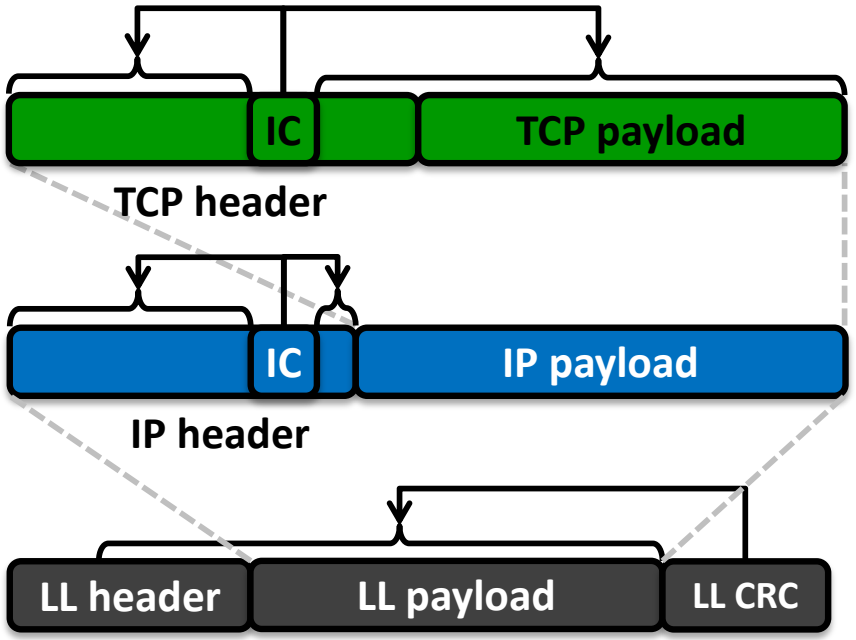
Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
  - **IC** over IP header only



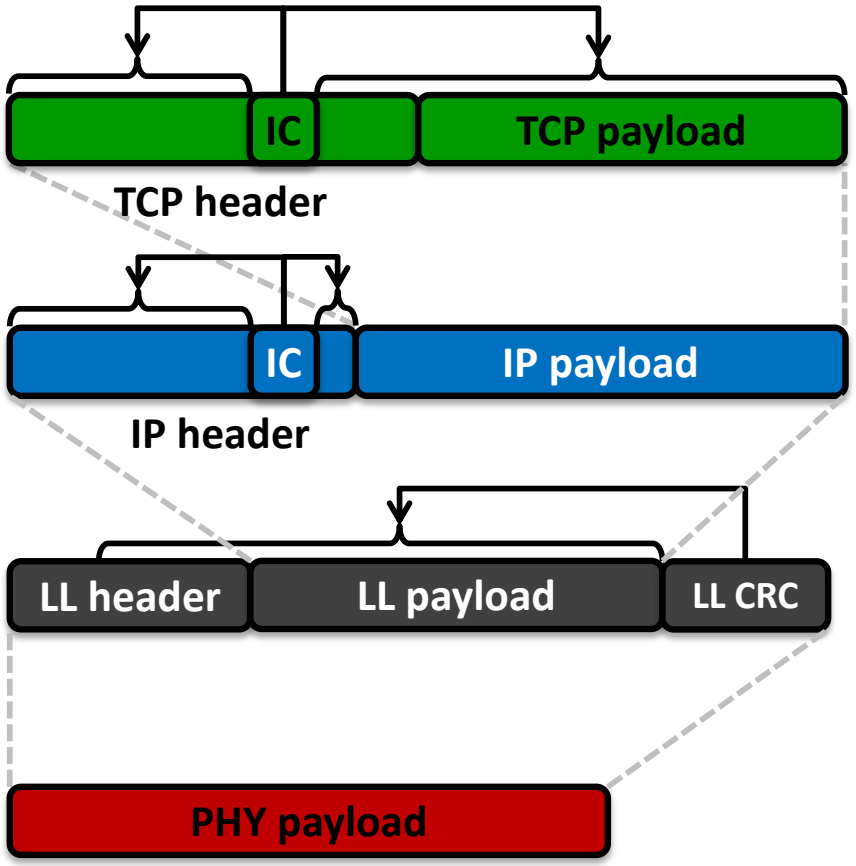
Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
  - **IC** over IP header only
- **Link layer (L2)**
  - **Cyclic Redundancy Check (CRC)**



Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
  - **IC** over IP header only
- **Link layer (L2)**
  - **Cyclic Redundancy Check (CRC)**
- **Physical layer (PHY)**
  - **Error Control Coding (ECC), or**
  - **Forward Error Correction (FEC)**



Error control: Key Ideas

- **Reduce the set of "allowed" messages**
  - **Not every string of bits** is an "allowed" message
  - Receipt of a **disallowed** string of bits means that the **message was garbled** in transit over the network

- We call an allowable message (of *n* bits) a ***codeword***
  - **Not all** *n*-bit strings are codewords!
  - The remaining *n*-bit strings are **"space" between codewords**

- **Plan:** Receiver will **use that space** to both **detect** and **correct** errors in transmitted messages

Encoding and decoding

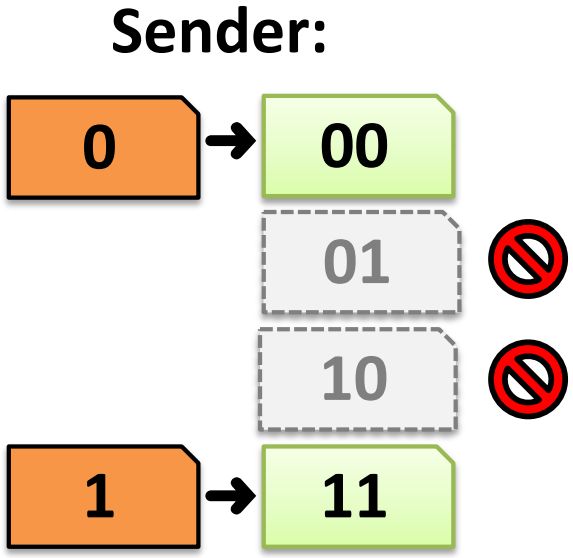
- **Problem: Not every string of bits is "allowed"**
  - But we want to be **able to send any** message!
  - *How can we send a "disallowed" message?*

- **Answer: Codes,** as a sender-receiver **protocol**
  - The sender must ***encode*** its messages ➜ codewords
  - The receiver then ***decodes*** received bits ➜ messages

- The **relationship between messages and codewords** isn't always obvious!

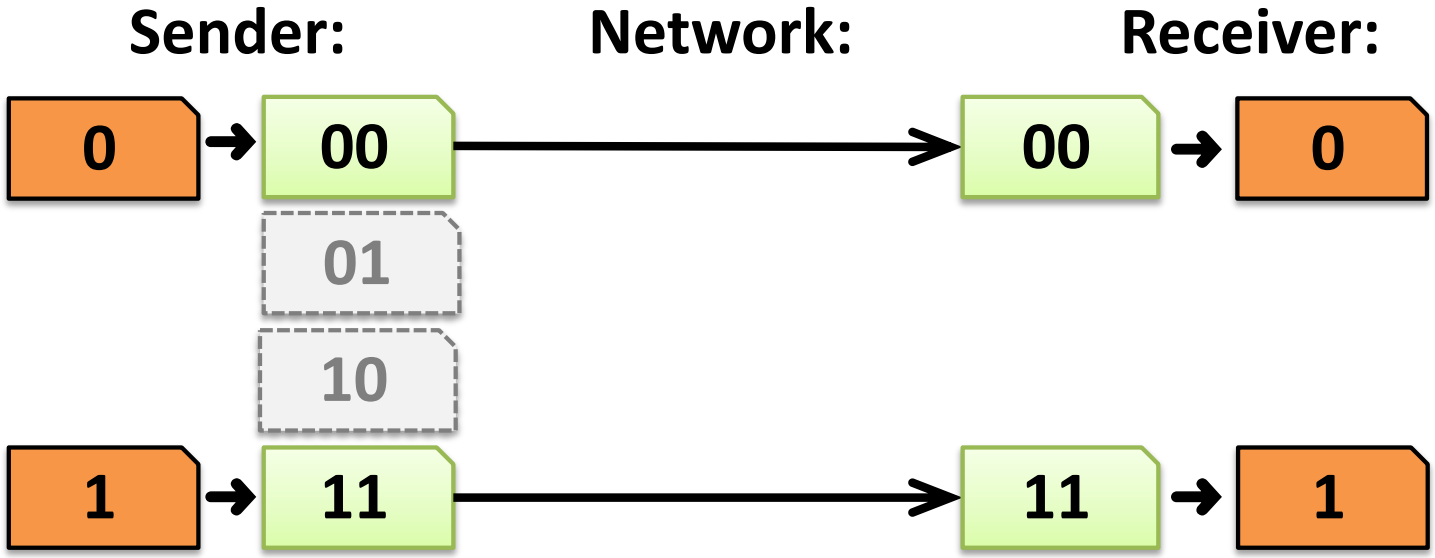
A simple error-detecting code

- Let's start simple: suppose messages are one bit long
- Take the message bit, and **repeat** it once
  - This is called a ***two-repetition code***



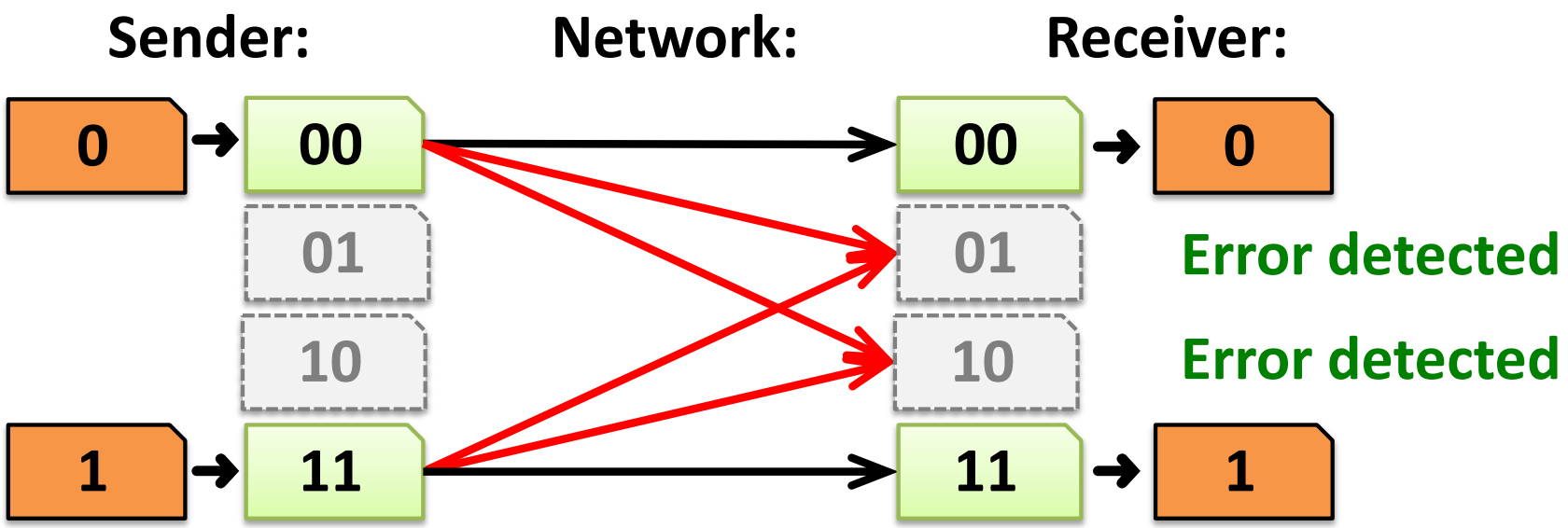
Receiving the two-repetition code

- Suppose the network causes **no bit error**
- Receiver **removes repetition** to **correctly *decode*** the message bits



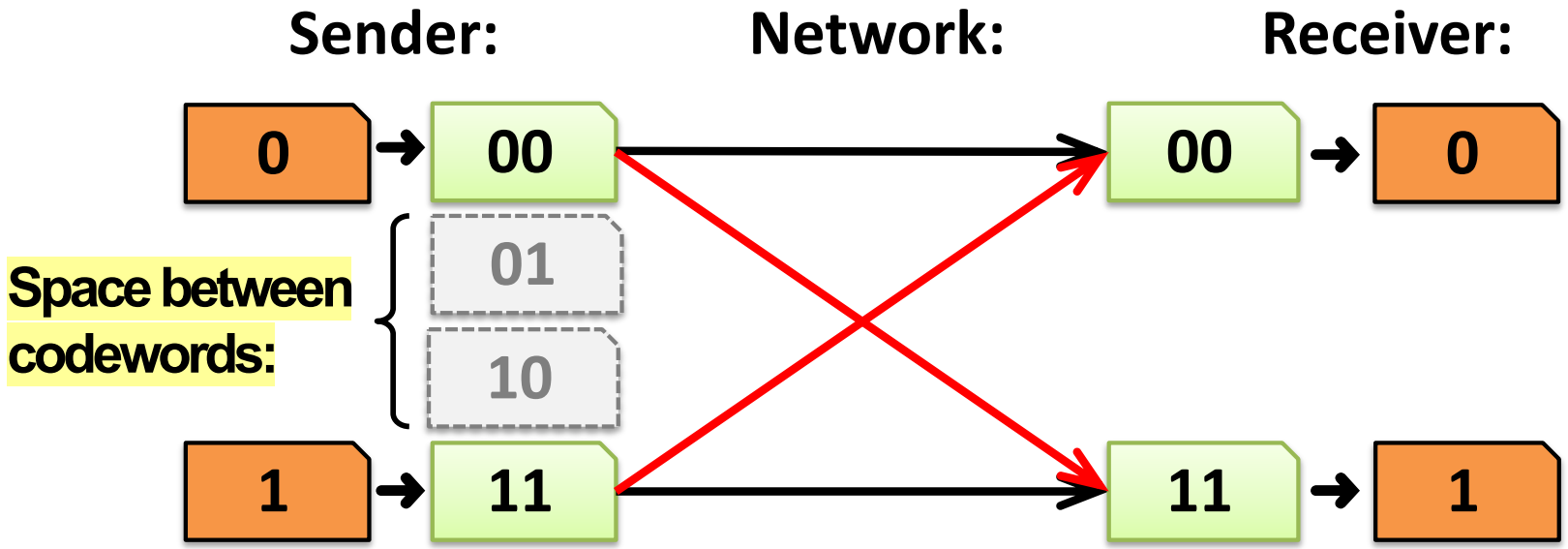
Detecting one bit error

- Suppose the network causes up to **one bit error**
- The receiver **can detect** the error:
  - It received a **non-codeword**
- Can the receiver **correct** the error?
  - **No!** The **other** codeword could have been sent as well



Reception with two bit errors

- Can receiver **detect** presence of **two bit errors?**
  - **No:** It has no way of telling which codeword was sent!
    - Enough bit errors that the sent codeword **"jumped over" the space between** codewords



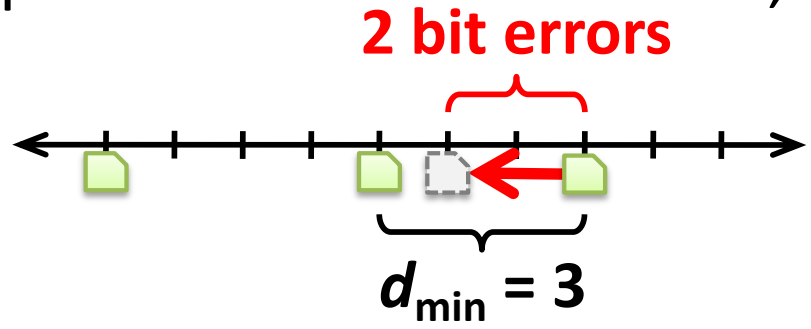
Hamming distance

- Measures the **number of bit flips** to change **one codeword into another**
- ***Hamming distance*** between two messages m_1, m_2: The number of bit flips needed to change m_1 into m_2
- **Example:** Two bit flips needed to change codeword 00 to codeword 11, so they are Hamming distance of **two** apart:

How many bit errors can we detect?

- Suppose the **minimum Hamming distance** between **any pair** of codewords is d_{min}

- Then, we **can detect at most $d_{min} - 1$ bit errors**
  - Will land in space between codewords, as we just saw



  - Receiver will flag message as **"Error detected"**

Decoding error **detecting** codes

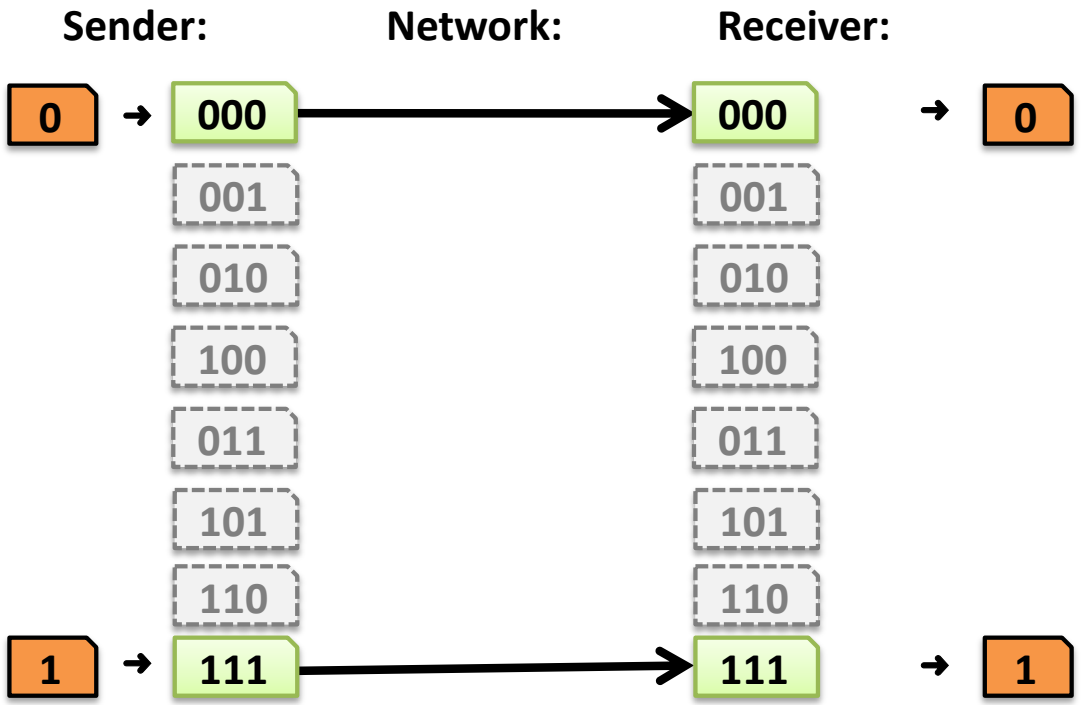
- **The receiver decodes** in a **two-step** process:

1. Map **received bits → codeword**
   - **Decoding rule:** Consider all codewords
     - **Choose** the one that **exactly matches** the received bits
     - **Return "error detected"** if none match

2. Map **codeword → source bits** and **"error detected"**
   - Use the **reverse map** of the sender

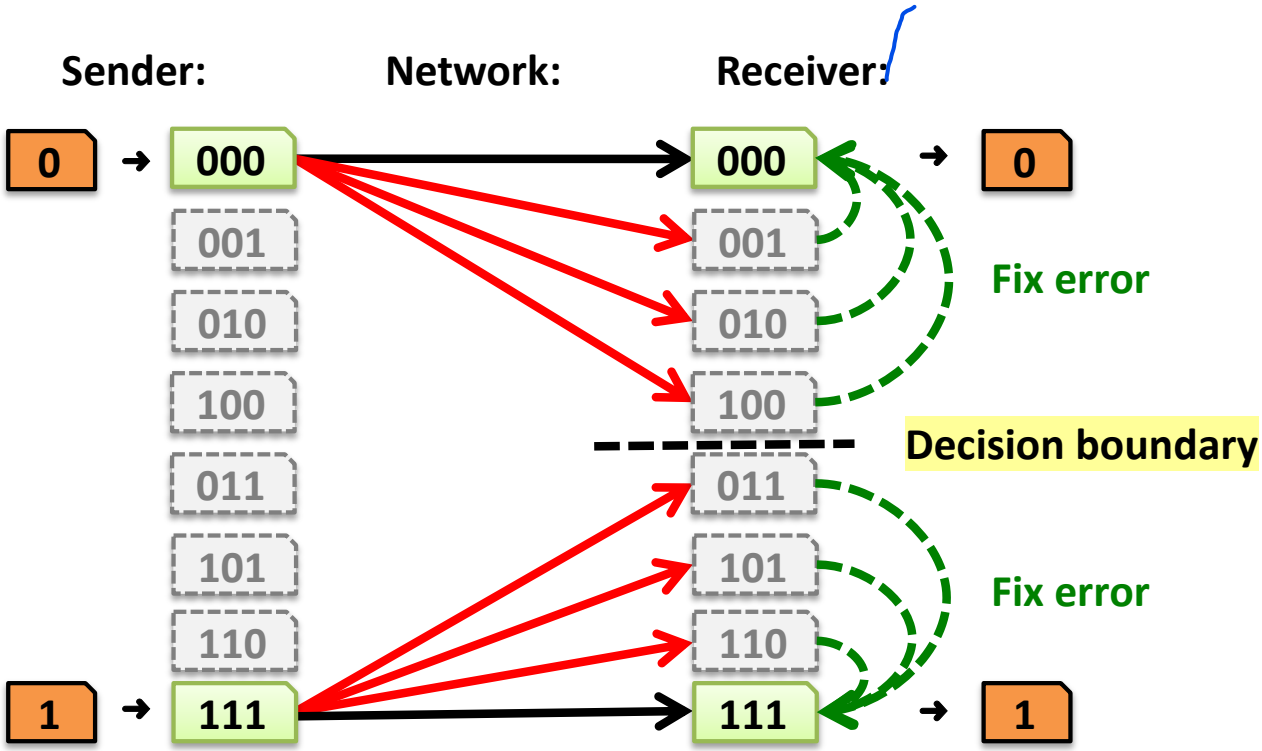
A simple error-correcting code

- Let's look at **a three-repetition code**
- If **no errors,** it works like the two-repetition code:



Correcting one bit error

- Receiver **chooses the closest codeword** (measured by Hamming distance) **to the received bits**
  - A ***decision boundary*** **exists** halfway between codewords



Decoding error correcting codes

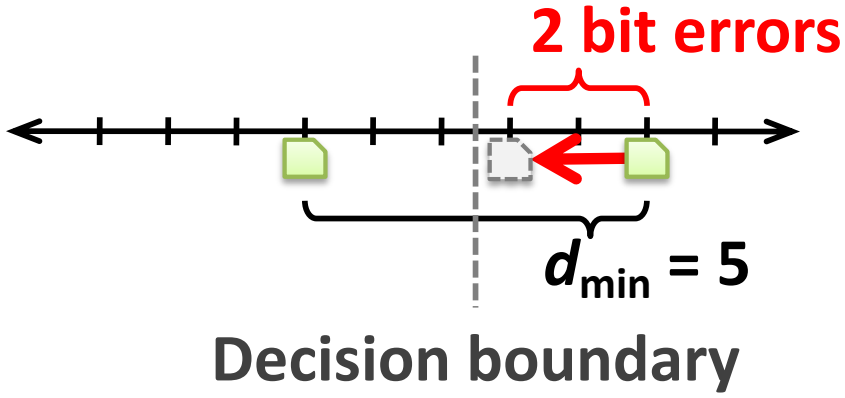
- **The receiver decodes** in a two-step process:

1. Map **received bits → codeword**
   - **Decoding rule:** Consider all codewords
     - **Choose one** with the **minimum Hamming distance** to the received bits

2. Map **codeword → source bits**
   - Use the **reverse map** of the sender

How many bit errors can we correct?

- There is **≥ d_{min} Hamming distance** between any two codewords
- So we can **correct ≤ $\lfloor (d_{min}-1)/2 \rfloor$ bit flips:**
  - This many bit flips can't move received bits closer to another codeword, **across the decision boundary:**



Code rate

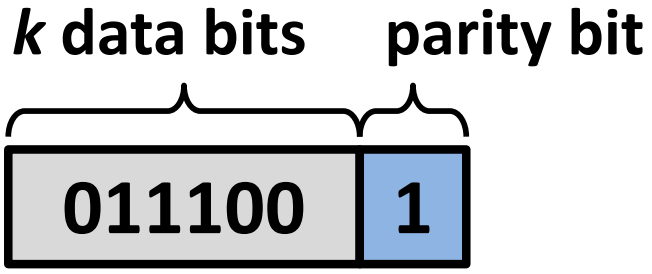
- Suppose **codewords** of length ***n*, messages** length ***k*** ($k < n$)
- The **code rate *R = k/n*** is a fraction between 0 and 1
- So, we have a **tradeoff:**
  - **High-rate codes** (*R* approaching one) generally **correct fewer errors,** but **add less overhead**
  - **Low-rate codes** (*R* close to zero) generally **correct more errors,** but **add more overhead**

Parity bit

- Given a message of k data bits $D_1, D_2, \ldots, D_k$, append a ***parity bit P*** to make a codeword of length $n = k + 1$

  – P is the exclusive-or of the data bits:
    - $P = D_1 \oplus D_2 \oplus \cdots \oplus D_k$

  – Pick the parity bit so that **total number of 1's is even**



Checking the parity bit

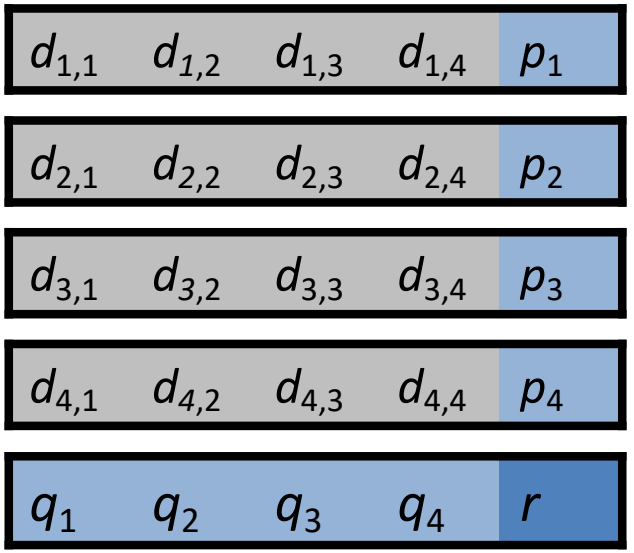
- **Receiver: counts number of 1s** in received message
  - **Even:** received message is a codeword
  - **Odd:** isn't a codeword, and **error detected**
    - But receiver doesn't know where, so **can't correct**
- What about d_{min}?
  - Change one data bit → change parity bit, so $d_{min} = 2$
    - So parity bit **detects 1 bit error, corrects 0**
- Can we **detect and correct more errors,** in general?

Two-dimensional parity

- Break up data into multiple rows
  - Parity bit **across** each row (p_i)
  - Parity bit **down each column** (q_i)
  - Add a parity bit r covering row parities

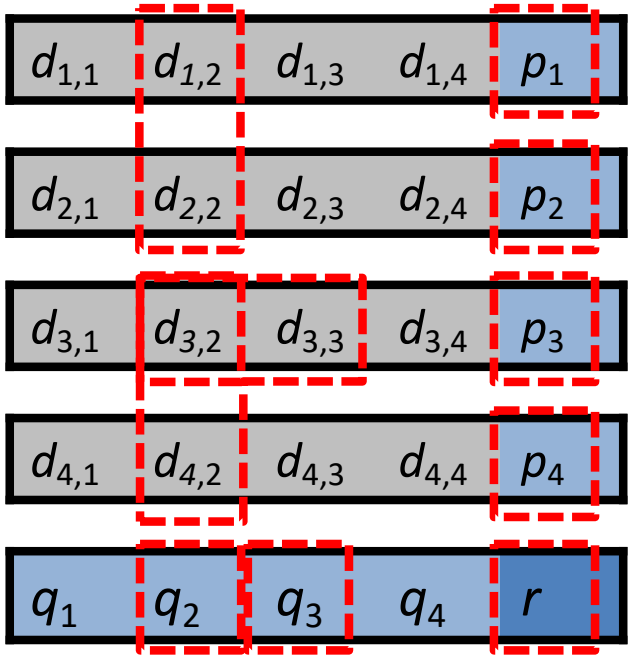
$$p_j = d_{j,1} \oplus d_{j,2} \oplus d_{j,3} \oplus d_{j,4}$$
$$q_j = d_{1,j} \oplus d_{2,j} \oplus d_{3,j} \oplus d_{4,j}$$
$$r = p_1 \oplus p_2 \oplus p_3 \oplus p_4$$

- This example has rate 16/25:



Two-dimensional parity: Properties

- Flip **1 data bit, 3 parity bits** flip
- Flip **2 data bits, ≥ 2 parity bits** flip
- Flip **3 data bits, ≥ 3 parity bits** flip



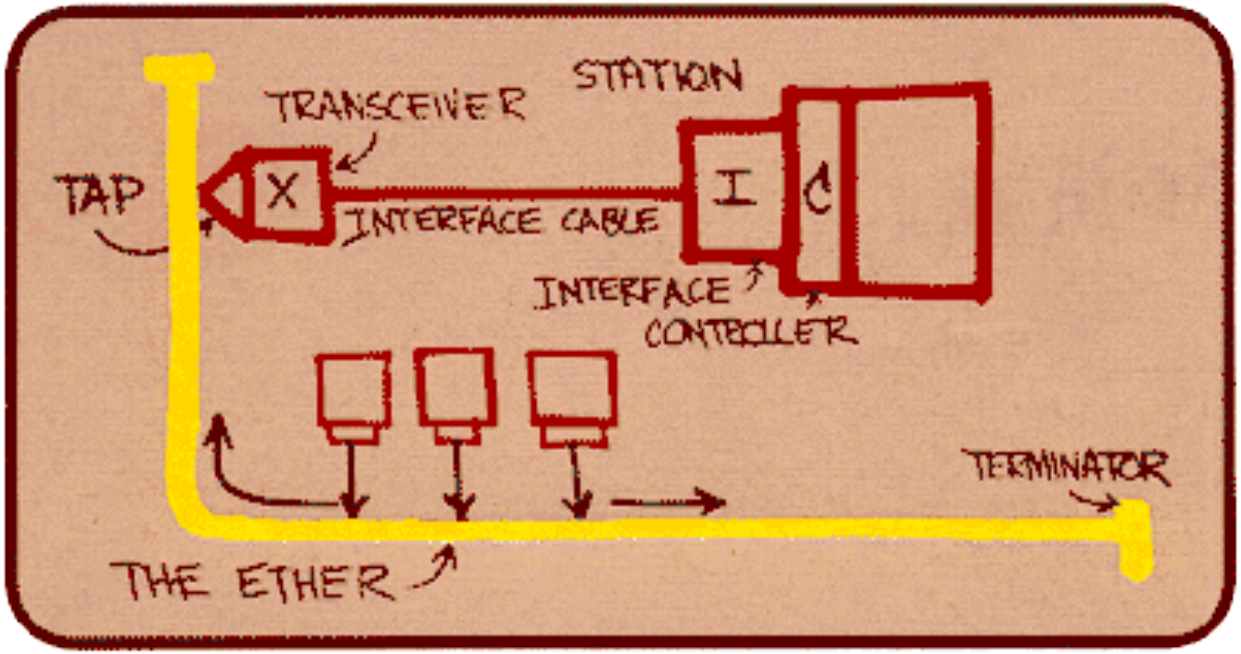
- Therefore, $\boldsymbol{d_{min} = 4}$, so
  - Can detect ≤ 3 bit errors
  - Can correct single-bit errors (*how?*)

- 2-D parity detects **most** four-bit errors

Ethernet

Ethernet

- Dominant wired LAN technology
- First widely used LAN technology
- Kept up with speed race: 10 Mbps – 40 Gbps

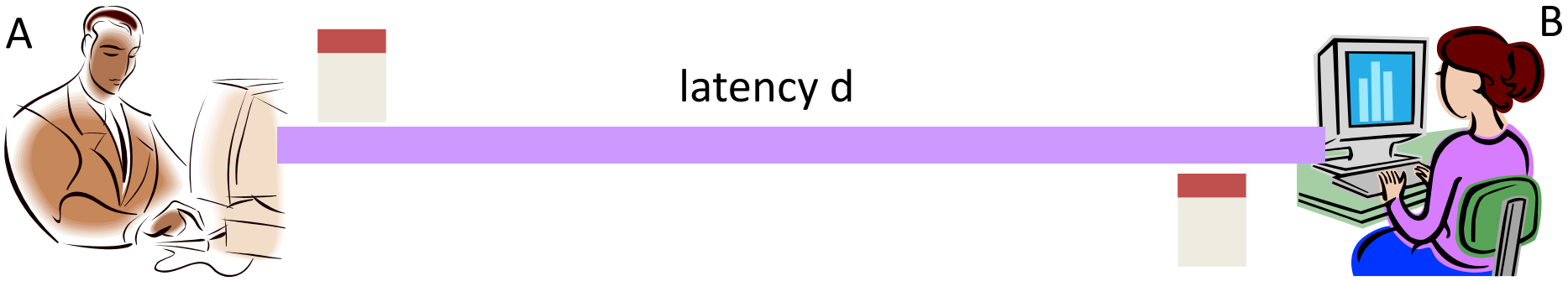


Metcalfe's Ethernet sketch

Ethernet Uses CSMA/CD

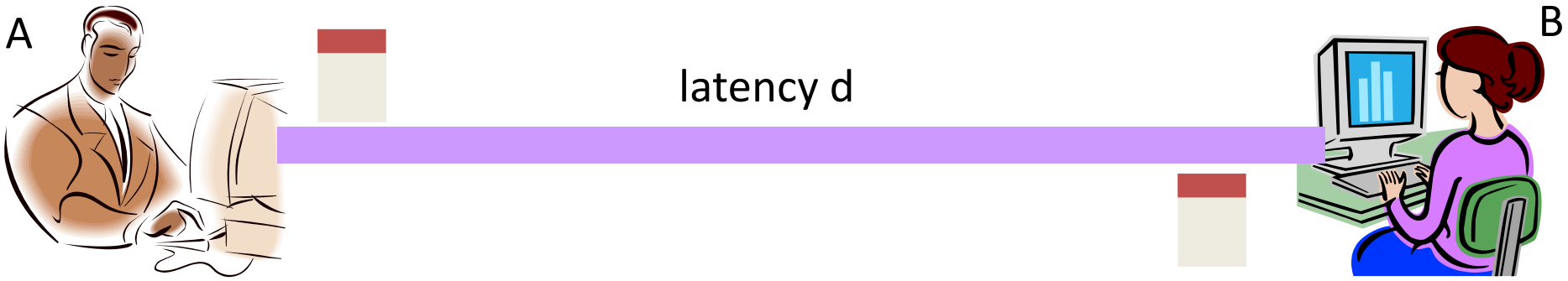
- Carrier Sense: wait for link to be idle
  - Channel idle: start transmitting
  - Channel busy: wait until idle
- Collision Detection: listen while transmitting
  - No collision: transmission is complete
  - Collision: abort transmission, and send jam signal
- Random Access: exponential back-off
  - After collision, wait random time before trying again
  - After m^{th} collision, choose K randomly from {0, …, 2^m-1}
  - … and wait for K*512 bit times before trying again

Limitations on Ethernet Length



- Latency depends on physical length of link
  - Time to propagate a packet from one end to other
- Suppose A sends a packet at time t
  - And B sees an idle line at a time just before t+d
  - … so B happily starts transmitting a packet
- B detects a collision, and sends jamming signal
  - But A doesn't see collision till t+2d

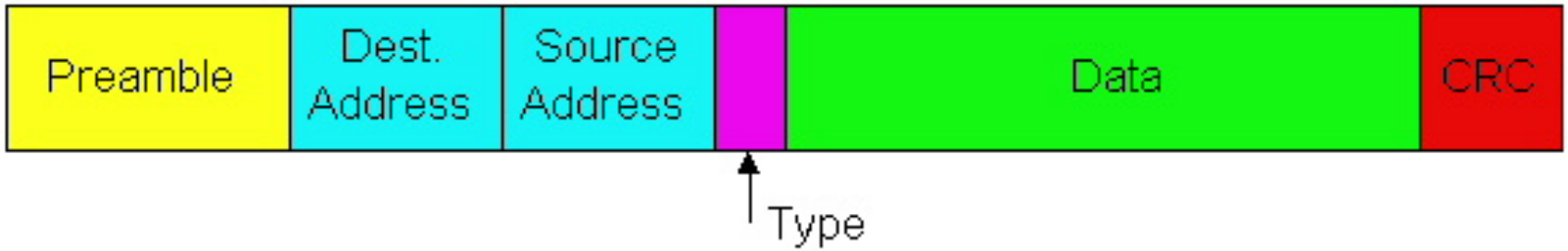
Limitations on Ethernet Length



- A needs to wait for time 2d to detect collision
  - So, A should keep transmitting during this period
  - … and keep an eye out for a possible collision
- Imposes restrictions on Ethernet
  - Maximum length of the wire: 2500 meters
  - Minimum length of the packet: 512 bits (64 bytes)

Ethernet Frame Structure

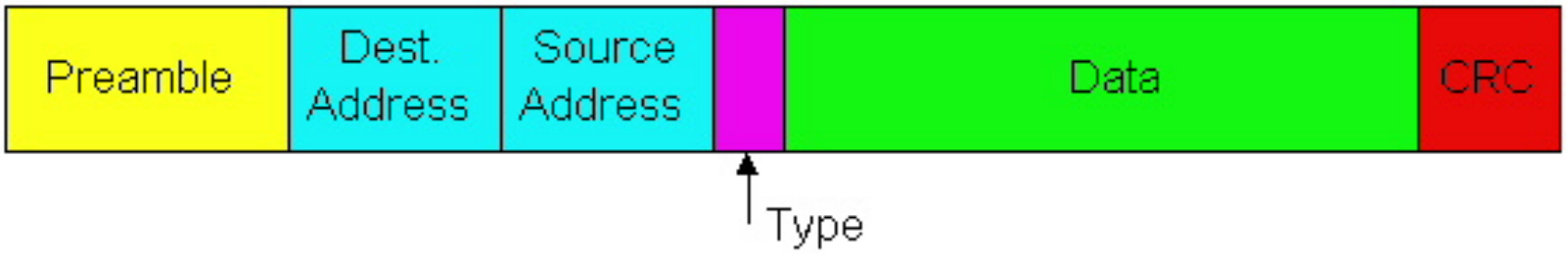
- Sending adapter encapsulates packet in frame



- Preamble: synchronization
  - Seven bytes with pattern 10101010, followed by one byte with pattern 10101011
  - Used to synchronize receiver, sender clock rates

Ethernet Frame Structure

- Addresses: source and destination MAC addresses
  - Adaptor passes frame to network-level protocol
    - If destination is local MAC address or broadcast address
  - Otherwise, adapter discards frame
- Type: indicates the higher layer protocol
  - Usually IP
  - But also Novell IPX, AppleTalk, …
- CRC: cyclic redundancy check

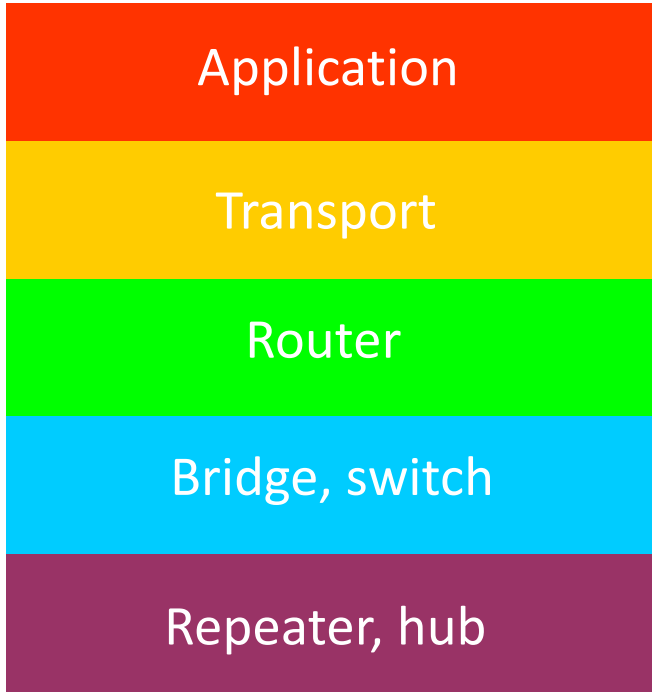


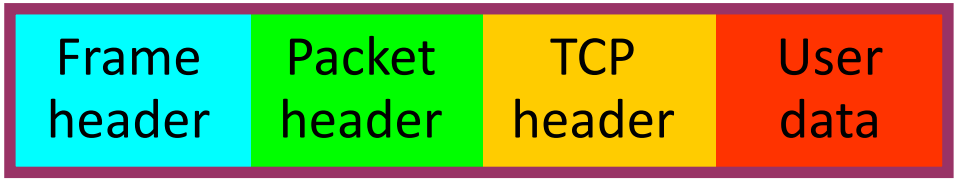
Unreliable, Connectionless Service

- Connectionless
  - No handshaking between send and receive adapter
- Unreliable
  - Receiving adapter doesn't send ACKs or NACKs
  - Packets passed to network layer can have gaps
  - Gaps can be filled by transport protocol (e.g., TCP)
  - Otherwise, the application will see the gaps

Summary: Multiple Layers

- Different devices switch different things
  - Network layer: packets (routers)
  - Link layer: frames (bridges and switches)
  - Physical layer: electrical signals (repeaters and hubs)





Conclusion

- Links
  - Connect two or more network adapters
  - … each with a unique address
  - … over a shared communication medium

- Coming next
  - Network layer (IP)