CIS 551 / TCOM 401

Computer and Network Security

Spring 2008
Lecture 9

Announcements

- Project 2: on the web Today
  - Due March 7th
  - Network intrusion detection
- Midterm I will be held in class next week
  - February 19th
  - Covers all material in class so far
  - Example exams from past instances are on the web (note: material was covered in a different order…)
- Plan for today:
  - Talk about formal verification a bit
  - Start to switch gears: network security

In-class exercise

- Formal specification of a common function.

- High level specification:
  - A Java method that sorts an array of integer values.
  - Java method signature:

```
// Sorts the input array
int[] sort(int[] input) { … }
```

- What is a good specification for this function?
  - Should be capture intended behavior/invariants
  - Should be expressed in mathematical / rigorous language

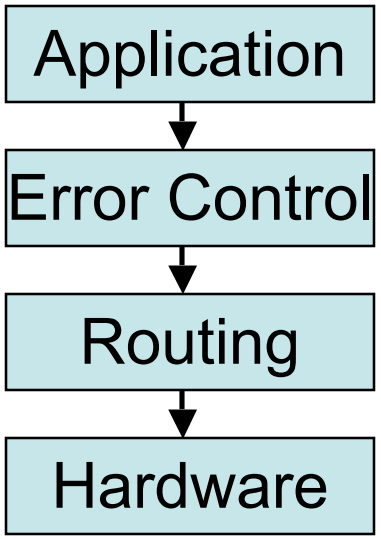
Network Architecture

- General blueprints that guide the design and implementation of networks
- Goal: to deal with the complex requirements of a network
- Use *abstraction* to separate concerns
  - Identify the useful service
  - Specify the interface
  - Hide the implementation

Layering

- A result of abstraction in network design
  - A stack of services (layers)
  - Hardware service at the bottom layer
  - Higher level services are implemented by using services at lower levels

- Advantages
  - Decompose problems
  - Modular changes

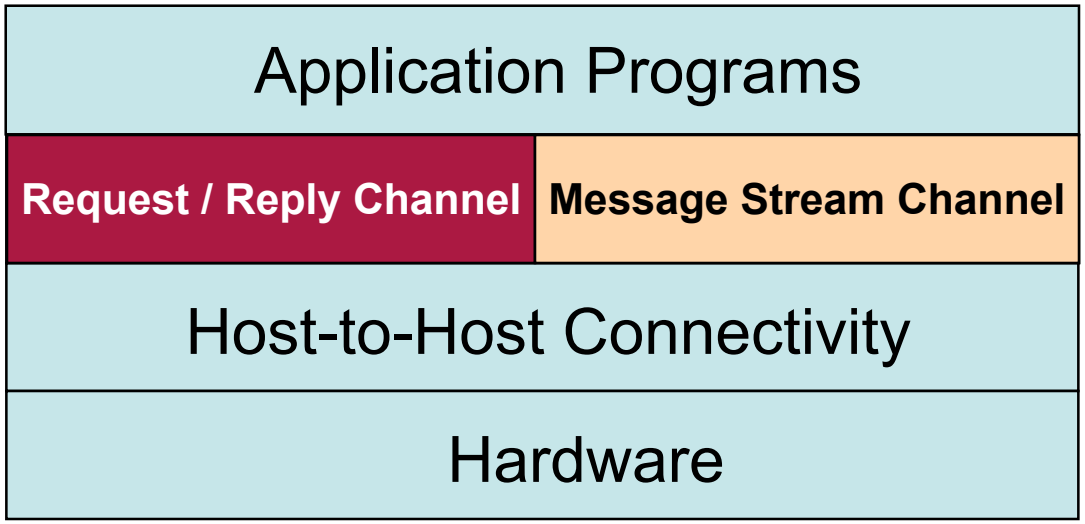


Protocols

- A *protocol* is a specification of an interface between modules (often on different machines)

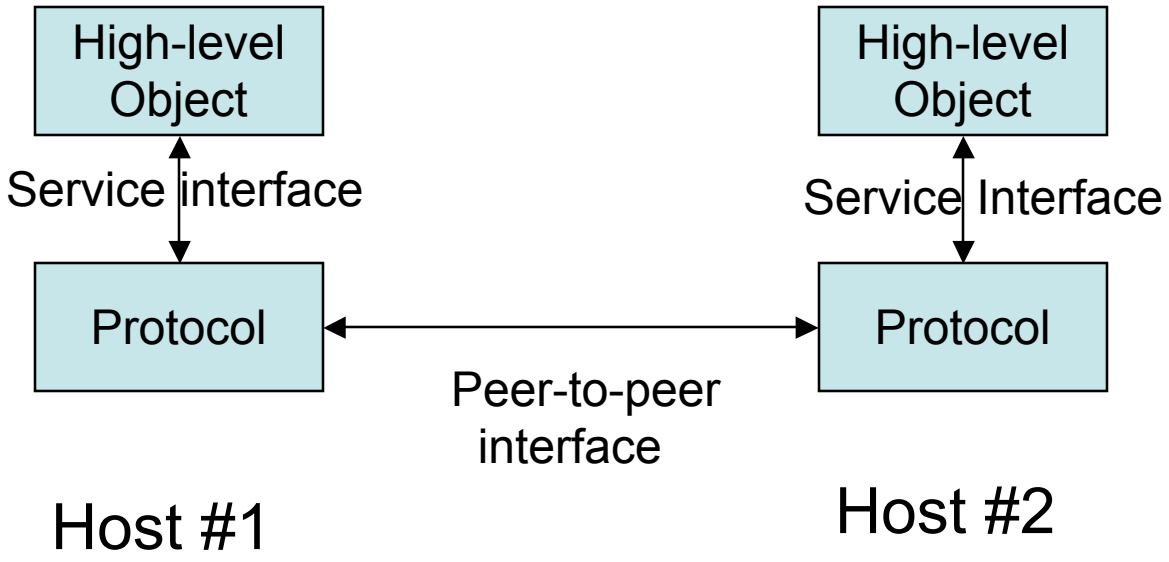
- Sometimes “protocol” is used to mean the implementation of the specification.

Example Protocol Stack

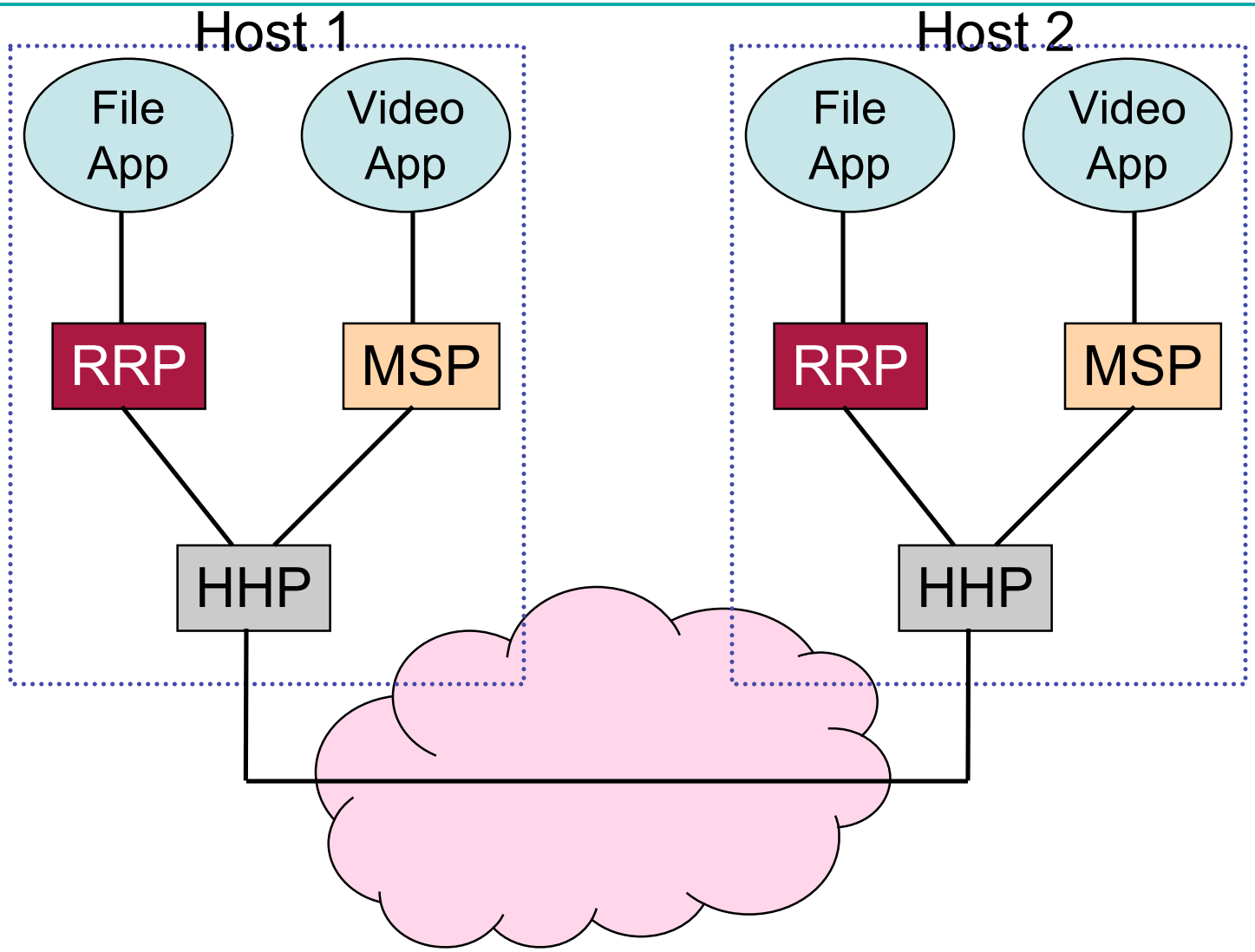


Protocol Interfaces

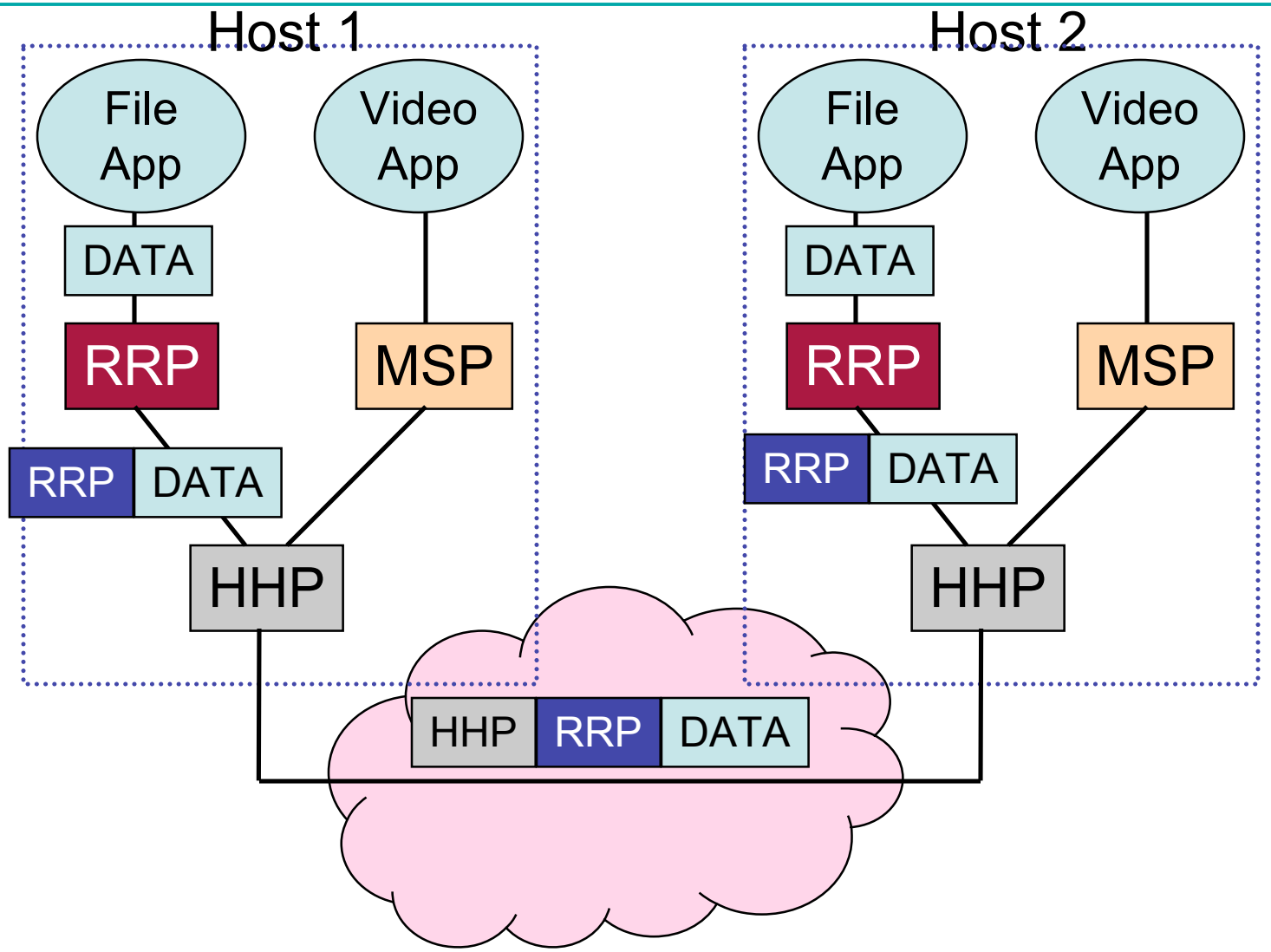
- Service Interfaces
  - Communicate up and down the stack
- Peer Interfaces
  - Communicate to counterpart on another host



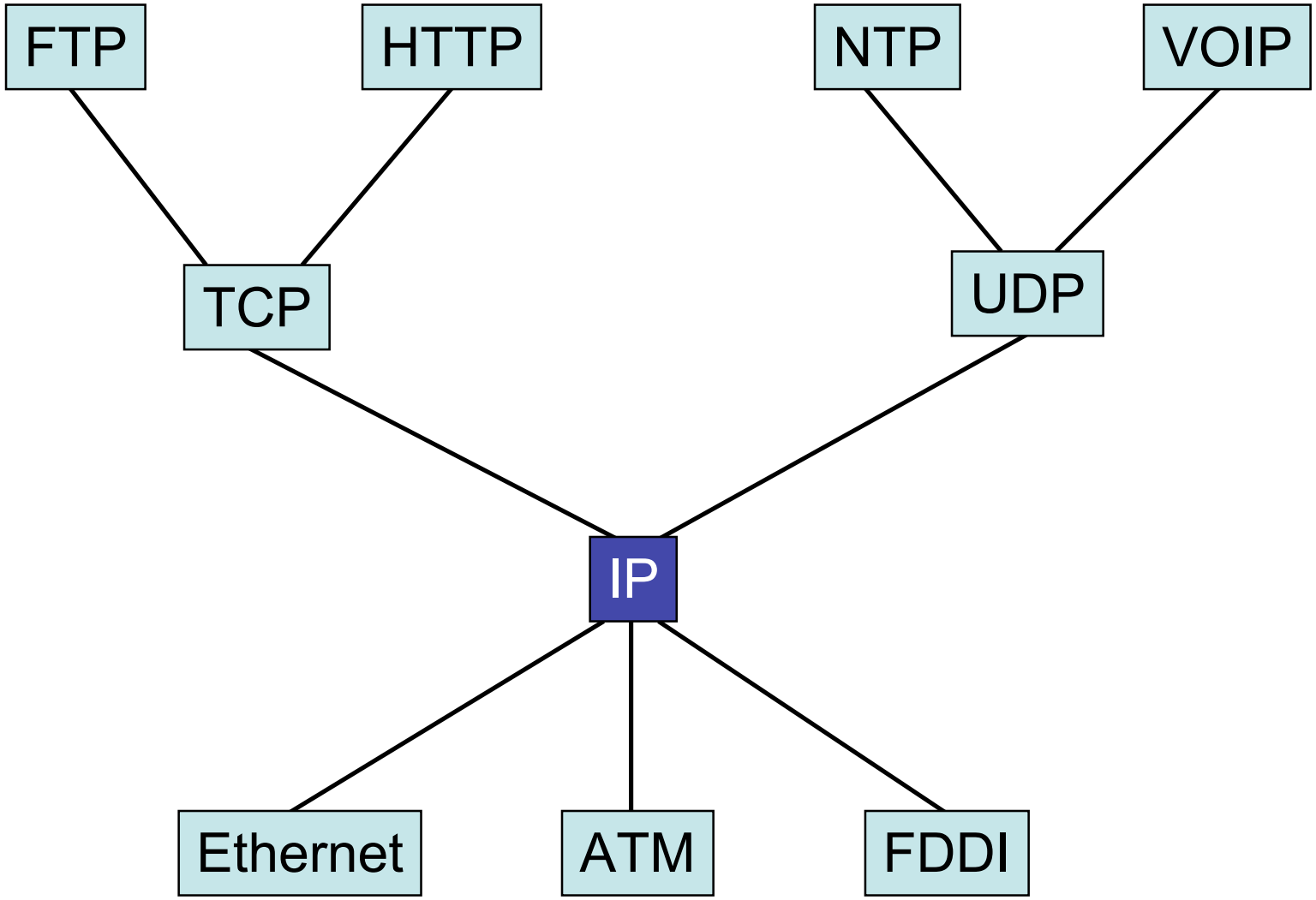
Example Protocol Graph



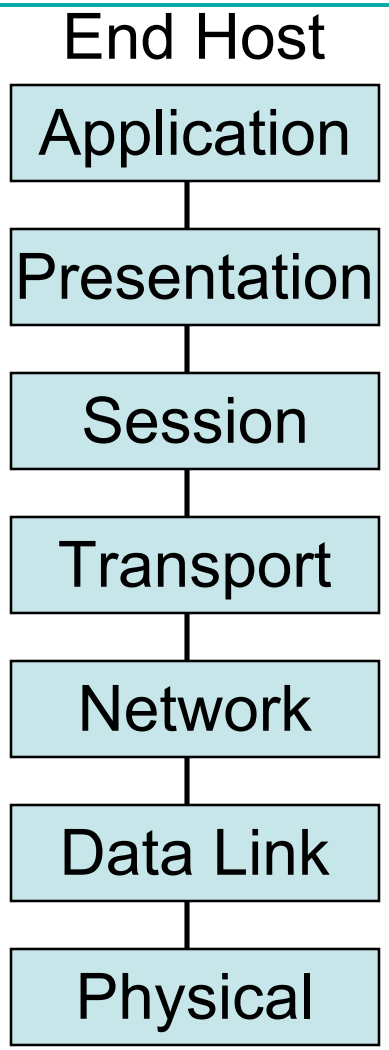
Encapsulation



Internet Protocol Graph



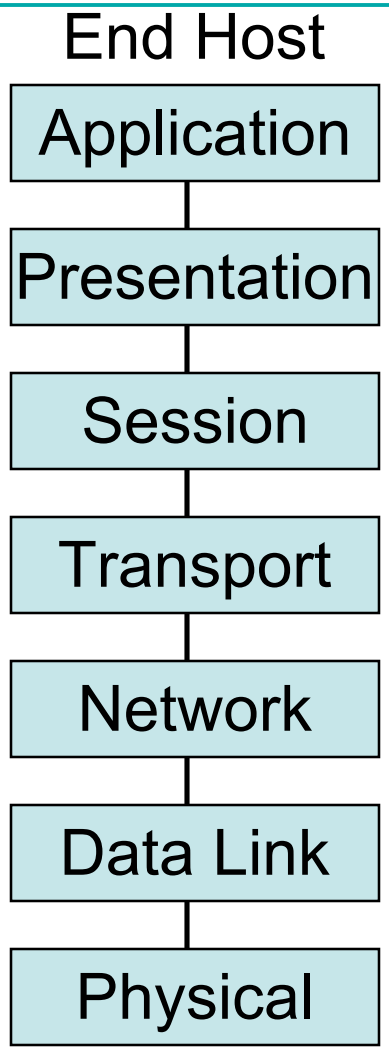
Open Systems Interconnection (OSI)



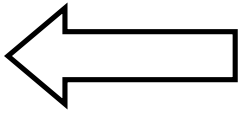
Reference model – not actual implementation.

- Application: Transmits *messages* (e.g. FTP or HTTP)
- Presentation: Data format issues (e.g. big- vs. little-endian)
- Session: Manages multiple streams of data
- Transport: Process to process protocols
- Network: Routes *packets* among nodes in network
- Data Link: Packages bit streams into *frames*
- Physical: Transmits raw bits over link

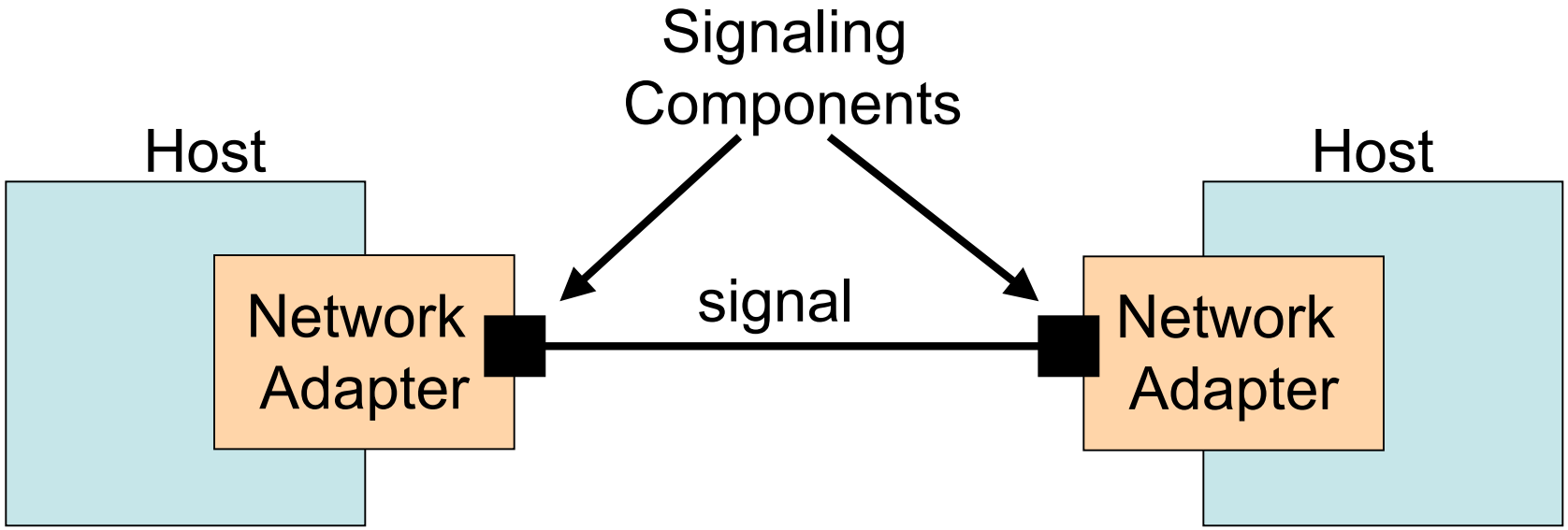
Open Systems Interconnection (OSI)



Reference model – not actual implementation.

- Application: Transmits *messages* (e.g. FTP or HTTP)
- Presentation: Data format issues (e.g. big- vs. little-endian)
- Session: Manages multiple streams of data
- Transport: Process to process protocols
- Network: Routes *packets* among nodes in network
- Data Link: Packages bit streams into *frames*
- Physical: Transmits raw bits over link 

Signaling Components



Network adapters encode streams of bits into signals.

Simplification: Assume two discrete signals—high and low.

Practice: Two different voltages on copper link or different brightness of light on fiber link.
(leads to some interesting encoding issues)

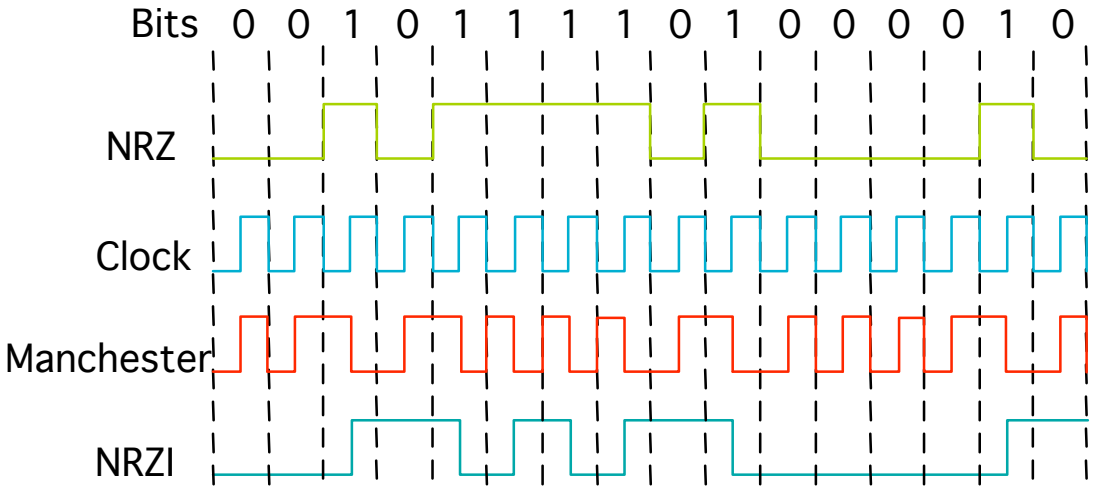
Not in this course

$$\nabla \bullet \vec{E} = \varepsilon_0 \rho$$

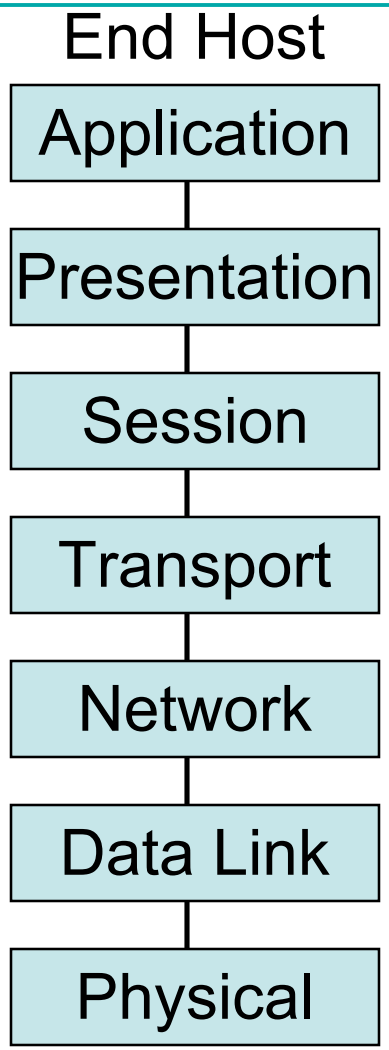
$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$$

$$\nabla \bullet \vec{B} = 0$$

$$\nabla \times \vec{B} = \mu_0 \vec{J} + \mu_0 \varepsilon_0 \frac{\partial \vec{E}}{\partial t}$$



Open Systems Interconnection (OSI)



Reference model – not actual implementation.

Transmits *messages* (e.g. FTP or HTTP)

Data format issues (e.g. big- vs. little-endian)

Manages multiple streams of data

Process to process protocols

Routes *packets* among nodes in network

Packages bit streams into *frames*

Transmits raw bits over link

Framing

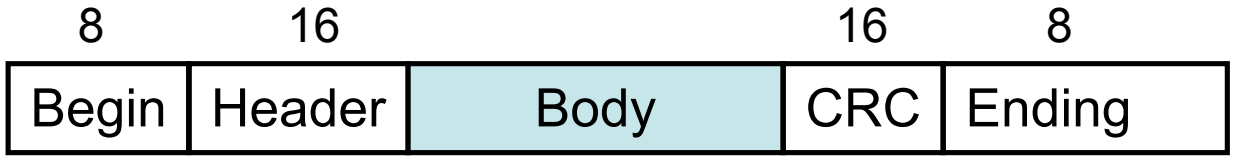
- Need a way to send blocks of data.
  - How does the network adapter detect when the sequence begins and ends?
  - Are there transmission errors in the data?

- *Frames* are link layer unit of data transmission
  - Byte oriented vs. Bit oriented
  - Point-to-point (e.g. PPP) vs. Multiple access (Ethernet)

A Multi-access, Bit-oriented Protocol

- Frames contain sequences of bits
  - Could be ASCII
  - Could be pixels from an image
- Frames read by many nodes
  - Address distinguishes intended recipient

- HDLC (High-level Data Link Control)
  - Begin and ending = 01111110
  - Uses *bit stuffing:* suffix five 1's with a 0



HDLC frame format

Problem: Error Detection & Correction

- Bit errors may be introduced into frames
  - Electrical interference
  - Thermal noise
- Could flip one bit or a few bits independently
- Could zero-out or flip a sequence of bits (*burst error*)

- How do you detect an error?

- What do you do once you find one?

Error Detection

- General principal: Introduce redundancy
- Trivial example: send two copies
  - High overheads: 2n bits to send n
  - Won't detect errors that corrupt same bits in both copies
- How can we do better?
  - Minimize overhead
  - Detect many errors
  - General subject: error detecting codes

Simple Error Detection Schemes

- Parity
  - 7 bits of data
  - 8^{th} bit is sum of first seven bits mod 2
  - Overhead: 8n bits to send 7n
  - Detects: any odd number of bit errors

- Internet Checksum algorithm
  - Add up the words of the message, transmit sum
  - 16 bit ones-complement addition
  - Overhead: 16 bits to send n
  - Does not detect all two bit errors

Cyclic Redundancy Check

- Reading: Wikipedia entry on CRC
- Used in link-level protocols
  - CRC-32 used by Ethernet, 802.5, PKzip, …
  - CRC-CCITT used by HDLC
  - CRC-8, CRC-10, CRC-32 used by ATM

- Better than parity or checksum
  - (e.g. 32 bits to send 12000)
- Simple to implement

Cyclic Redundancy Check (CRC)

- Consider (n+1)-bit message as a n-degree polynomial
  - Polynomial arithmetic modulo 2
  - Bit values of message are coefficients
  - Message = 10011010
  - Polynomial

$$M(z) = (1 \cdot z^7) + (0 \cdot z^6) + (0 \cdot z^5) + (1 \cdot z^4) + (1 \cdot z^3) + (0 \cdot z^2) + (1 \cdot z^1) + (0 \cdot z^0)$$
$$= z^7 + z^4 + z^3 + z^1$$

Cyclic Redundancy Check

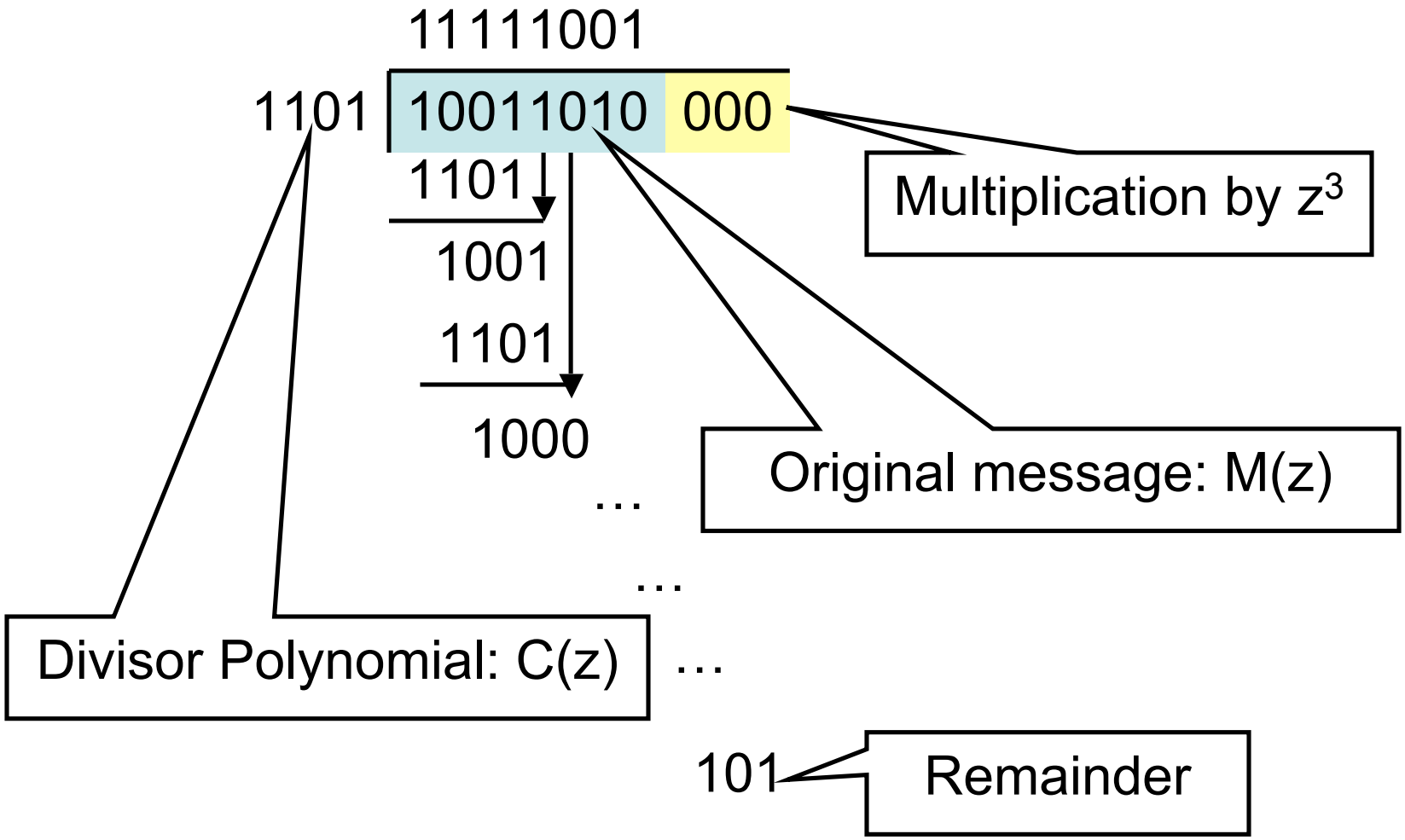
- Sender and receiver agree on a *divisor polynomial* C(z) of degree k
  - Example k = 3
  - $C(z) = z^3 + z^2 + 1$
  - Coefficients are 1101

- Error correction bits are remainder of

  $(M(z) \cdot z^k)$ divided by C(z)

- This yields a n+k bit transmission polynomial P(z) that is *exactly* divisible by C(z)

Example CRC Calculation



Example CRC Calculation

$Z^3 \cdot$ Original Message M(z) = 10011010 000

Remainder = + 101

Transmitted message P(z) = 10011010 101

- Recipient checks that C(z) evenly divides the received message.

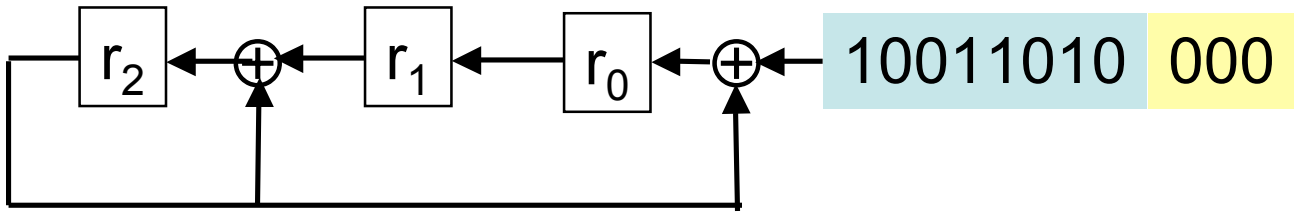
CRC Error Detection

- Must choose a good divisor C(z)
  - There are many standard choices:
    CRC-8, CRC-10, CRC-12, CRC-16, CRC-32
  - CRC-32: 0x04C11DB7
- All 1-bit errors as long as z^k and z^0 coefficients are 1
- All 2-bit errors as long as C(z) has three terms
- Any odd number of errors if (z+1) divides C(z)
- Any burst errors of length ≤ k

CRC Implementations

- Easy to implement in hardware
  - Base 2 subtraction is XOR
  - Simple k-bit shift register with XOR gates inserted before 1's in C(z) polynomial
  - Message is shifted in, registers fill with remainder

- Example C(z) = 1101



Error Correction Codes

- Redundant information can be used to *correct* some errors
- Typically requires more redundancy

- Tradeoffs:
  - Error detection requires retransmission
  - Error correction sends more bits all the time
- Forward Error Correction is useful:
  - When errors are likely (e.g. wireless network)
  - When latency is too high for retransmission (e.g. satellite link)