# CSE331: Introduction to Networks and Security

Lecture 5

Fall 2006

# Announcements

- First project: Due: 25 Sept. 2006
- Prof. Zdancewic will be away Sept. 18 & 20.
  – Class will be taught by Peng Li
- First homework is available online:
  – Due: 29th Sept. 2006

# Open Systems Interconnection (OSI)

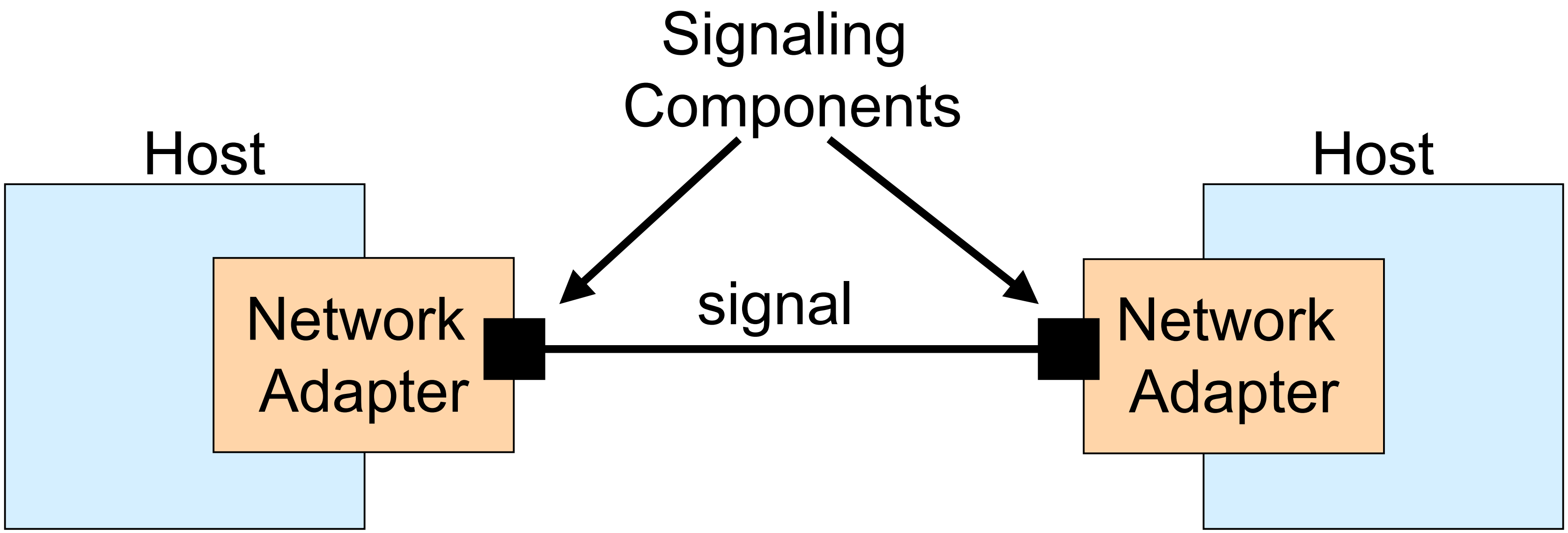
Reference model – not actual implementation.

| End Host | |
|---|---|
| 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 |

# Problem: Physical connection

- Transmitting signals
- Encoding & decoding bits
- Error detection and correction
- Reliable transmission

# 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.
(leads to some interesting encoding issues)

# Framing

- Need a way to send blocks of data.
  - How does the network adapter detect when the sequence begins and ends?
- *Frames* are link layer unit of data transmission
  - Byte oriented vs. Bit oriented
  - Point-to-point (e.g. PPP) vs. Multiple access (Ethernet)

# Byte-oriented Protocols

- View each frame as a sequence of bytes
- BISYNC
  - Binary Synchronous Communication protocol
  - Developed by IBM in late 1960's
- DDCMP
  - Digital Data Communication Message Protocol
  - Used in Digital Equipment Corporation's DECNET
- Primary question: which bytes are in the frame?

# Sentinel Approach

| 8 | 8 | 8 | | 8 | | 8 | 16 |
|---|---|---|---|---|---|---|---|
| SYN | SYN | SOH | HEADER | STX | BODY | ETX | CRC |

BISYNC frame format

- SYN – synchronization
- SOH – start of header
- STX – start of text
- ETX – end of text
- CRC – cyclic redundancy check

} Sentinels

# Character Stuffing

- What happens if ETX code occurs in BODY?

- Use an "escape character"
- DLE – Data-link-escape

- Used just as \ in C- or Java-style strings
  - **`"quotes in \"quotes\""`**
  - **`"slash is \\"`**

# (PPP) Point-to-Point Protocol

| 8 | 8 | 8 | 16 | | 16 | 8 |
|---|---|---|---|---|---|---|
| Flag | Addr | Ctrl | Protocol | Payload | Checksum | Flag |

PPP frame format

- Used for dial-up connections (modem)
- Flag – sentinel 01111110
- Protocol – demux identifies high-level protocol such as IP or LCP
- Payload size is negotiated
  - 1500 bytes default
  - Link Control Protocol (LCP)

# Byte-counting Approach

| 8 | 8 | 8 | 14 | 42 | | 16 |
|---|---|---|---|---|---|---|
| SYN | SYN | Class | Count | Header | Body | CRC |

DDCMP Frame Format

- Instead of sentinels, include *byte count* in frame.
- What happens if count is corrupted?

# Bit-oriented Protocols

- Frames are just sequences of bits
- Could be ASCII
- Could be pixels from an image

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

| 8 | 16 | | 16 | 8 |
|---|---|---|---|---|
| Begin | Header | Body | CRC | Ending |

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

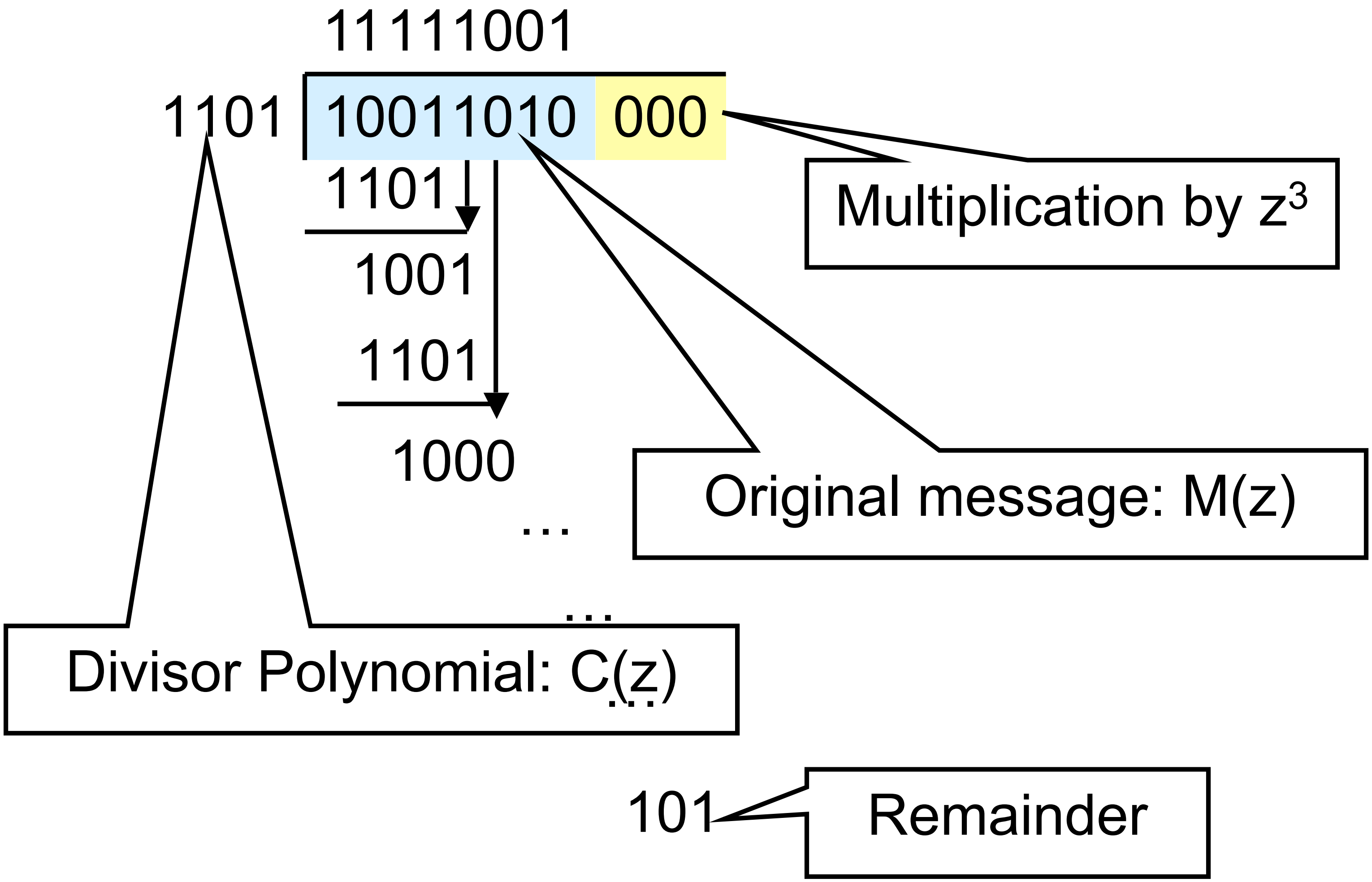
# Cyclic Redundancy Check

- 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 $(M(z) \times z^k)$ 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 calculuation

Transmitted message 10011010 101

Original message: M(z)

Remainder

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

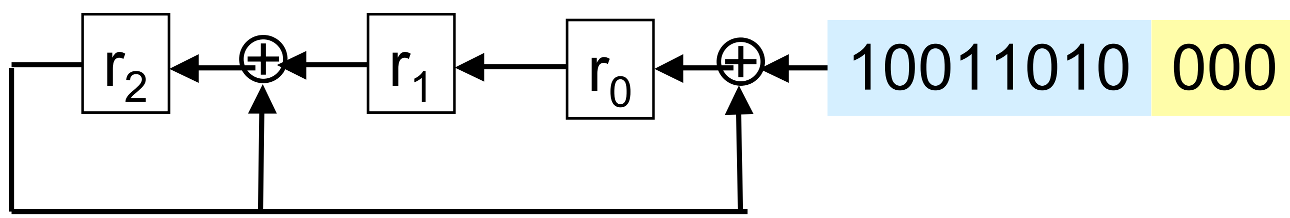
# CRC Error Detection

- 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)