Lecture 2: Layers and Framing
Our Problem

- Communications is complicated
  - Modulation and encoding bits
  - Splitting sequences of bits into packets
  - Fixing errors
  - Controlling access to network
  - Routing
  - Recovering from lost messages
  - Etc….

- Really hard to think about all of this and get it right
- Not all applications need all of it
- How to achieve interoperability?
Layering: A Modular Approach

- Sub-divide the problem
  - Each layer relies on services from layer below
  - Each layer exports services to layer above

- Interface between layers defines interaction
  - Hides implementation details (encapsulation)
  - Layers can change without disturbing other layers (modularity)

- Interface among peers in a layer is a protocol
  - If peers speak same protocol, they can interoperate
Key Design Decision

- How do you divide functionality across the layers?

- **End-to-end argument [Saltzer84]**
  - Functionality should be implemented at a lower layer iff it can be **correctly** and **completely** implemented there
  - Incomplete versions of a function can be used as a performance enhancement, but not for correctness

- Early, and still relevant, example
  - ARPAnet provided reliable link transfers between switches
  - Was this enough for reliable communication?
  - No, packets could still get corrupted on host-switch link, or inside of the switches
  - Hence, still need reliability at higher layers
Protocol Standardization

- Communicating hosts speaking the same protocol
  - Standardization to enable multiple implementations
  - Or, the same folks have to write all the software

- Internet Engineering Task Force
  - Based on working groups that focus on specific issues
  - Produces “Request For Comments” (RFCs)
    » Rough consensus and running code
    » After enough time passes, promoted to Internet Standards

- Other standards bodies exist
  - ISO, ITU, IEEE, etc.
TCP/IP Protocol Stack

CSE 123 – Lecture 2: Layers & Framing
Encapsulation via Headers

- Typical Web packet

Which header gets added first?
A. Ethernet
B. IP
C. TCP
D. HTTP

- Notice that layers add overhead
  - Space (headers), effective bandwidth
  - Time (processing headers, “peeling the onion”), latency
Internet Protocol Suite

The Hourglass Model

Applications
Transport
Data Link
Physical

“Thin Waist”

FTP
HTTP
NV
TFTP

TCP
UDP

IP

NET₁
NET₂
…
NETₙ

CSE 123 – Lecture 2: Layers & Framing
Later: Phy/(MAC)Link layer

- **Signal encoding**
  - Encode binary data from source node into signals that physical links carry
  - Signal is decoded back into binary data at receiving node
  - Work performed by network adapter at sender and receiver

- **Media access**
  - Arbitrate which nodes can send frames at any point in time
  - Not always necessary; e.g. point-to-point duplex links
For now: (Data) Link Layer

- **Framing**
  - Break stream of bits up into discrete chunks

- **Error handling**
  - Detect and/or correct errors in received frames

- **Multiplexing**
  - Determine appropriate destination for a given frame
  - Also not always required; again, point-to-point
Today’s Focus: Framing

● Break down a stream of bits into smaller, digestible chunks called frames

● Allows the link to be shared
  ✦ Multiple senders and/or receivers can time multiplex the link
  ✦ Each frame can be separately addressed

● Provides manageable unit for error handling
  ✦ Easy to determine whether something went wrong
  ✦ And perhaps even to fix it if desired
What’s a Frame?

- Wraps payload up with some additional information
  - Header usually contains addressing information
  - Maybe includes a trailer (w/checksum—next lecture)
- Basic unit of reception
  - Link either delivers entire frame payload, or none of it
  - Typically some maximum transmission unit (MTU)
- Some link layers require absence of frames as well
  - I.e., minimum gaps between frames

How do hosts find frames on the link?

A. Special delimiters
B. Fixed-length chunks
C. Arrive on a schedule
D. All of the above
Identifying Frames

- First task is to delineate frames
  - Receiver needs to know when a frame starts and ends
  - Otherwise, errors from misinterpretation of data stream

- Several different alternatives
  - Fixed length (bits) frames
  - Explicitly delimited frames
    » Length-based framing
    » Sentinel-based framing
  - Fixed duration (seconds) frames
Fixed-Length Frames

- Easy to manage for receiver
  - Well understood buffering requirements

- Introduces inefficiencies for variable length payloads
  - May waste space (padding) for small payloads
  - Larger payloads need to be fragmented across many frames
  - Very common inside switches

- Requires explicit design tradeoff
  - ATM uses 53-byte frames (cells)
Length-Based Framing

- To avoid overhead, we’d like variable length frames
  - Each frame declares how long it is
  - E.g. DECNet DDCMP

- What’s the issue with explicit length field?
  - Must correctly read the length field (bad if corrupted)
    - Need to decode *while* receiving
  - Still need to identify the beginning…
Sentinel-based Framing

- Allow for variable length frames
- Idea: mark start/end of frame with special “marker”
  - Byte pattern, bit pattern, signal pattern
- But… must make sure marker doesn’t appear in data

- Two solutions
  - Special non-data physical-layer symbol
    › Impact on efficiency (can’t use symbol for data) of code
  - Stuffing
    › Dynamically remove marker bit patterns from data stream
    › Receiver “unstuff”s data stream to reconstruct original data
Stuffing

- Insert bytes/bits into data stream to make sure that sentinel (flag) does not appear in payload
Bit-level Stuffing

- Avoid sentinel bit pattern in payload data
  - Commonly, sentinel is bit pattern 01111110 (0x7E)
  - Invented for SDLC/HDLC, now standard pattern
- Sender: any time five ones appear in outgoing data, insert a zero, resulting in 0111110

  Stuffed bits
  011111100001110111011111011111001
  011111001000011101110111111011111001

- Receiver: any time five ones appear, removes next zero
  - If there is no zero, there will either be six ones (sentinel) or
  - It declares an error condition!
  - Note bit pattern that cannot appear is 01111111 (0x7F)
What’s the worst case bit stream in terms of efficiency?

A. All zeros
B. All ones
C. Repeated 011111
D. Repeated 01111110
For Next Class

- Read 2.4
- Take a look at Project 1
- Discussion CANCELLED

What’s the worst case bit stream in terms of efficiency?

A. All zeros
B. All ones
C. Repeated 011111
D. Repeated 01111110