

# Lecture 2:

## Layers and Framing

---

CSE 123: Computer Networks  
Aaron Schulman





# Our Problem

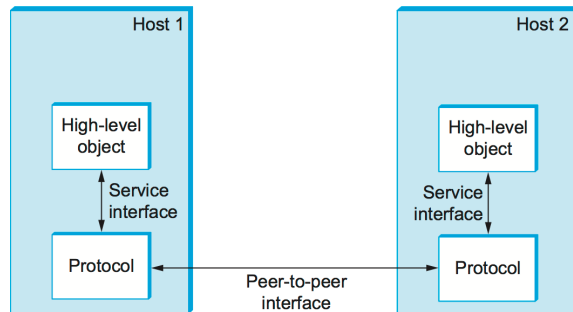
---

- 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 **if and only if** 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: reliability
  - ◆ ARPAnet (low layer) 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**

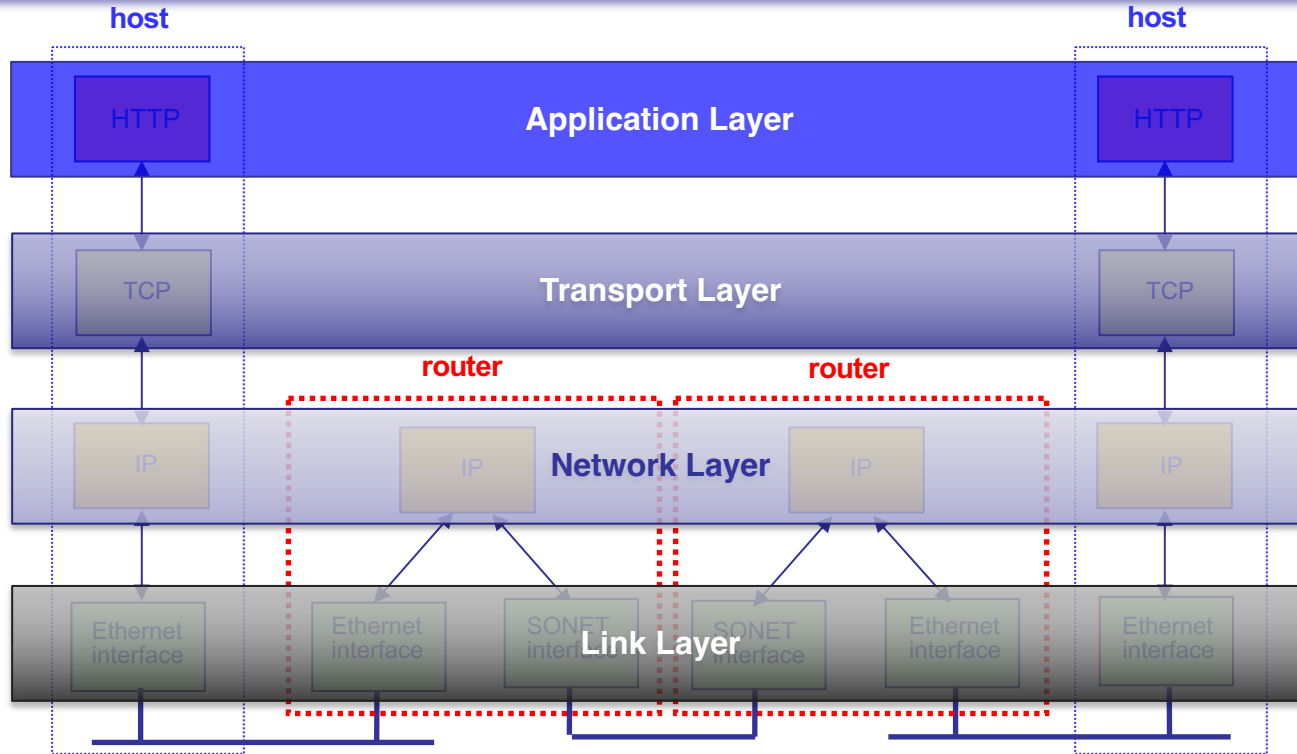


# Protocol Standardization

---

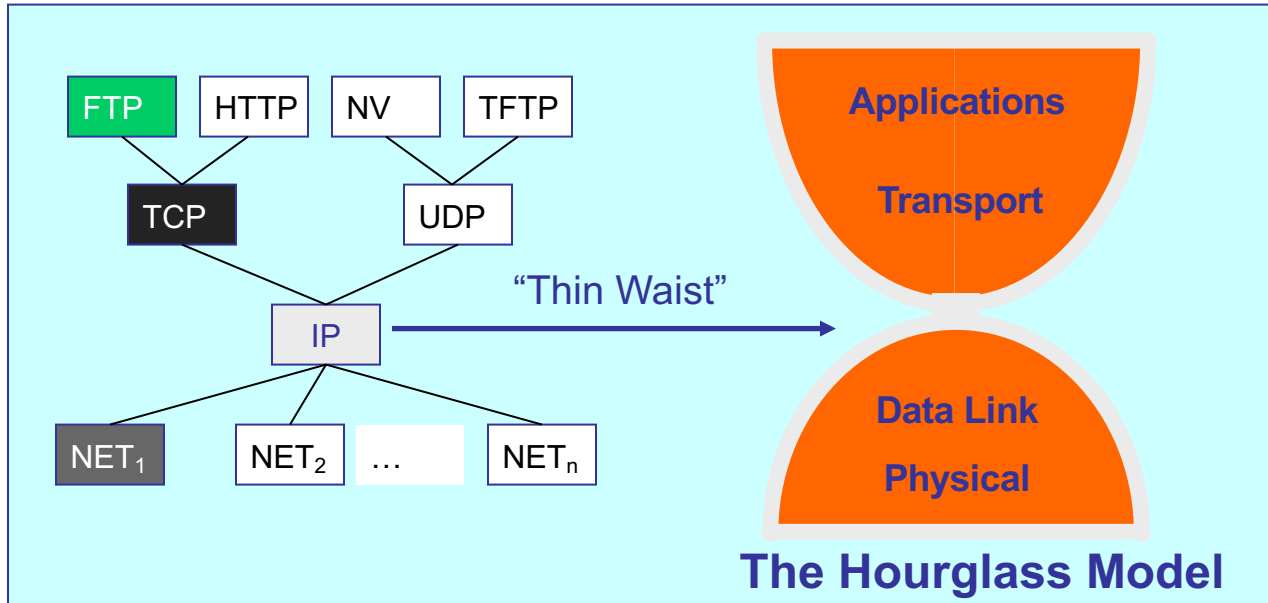
- Communicating hosts speaking the same protocol
  - ◆ Standardization to enable multiple implementations
  - ◆ Or, the same folks have to write all the software
- Internet Engineering Task Force (Meet + Party + Argue)
  - ◆ Based on working groups that focus on specific issues
  - ◆ Produces “Request 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





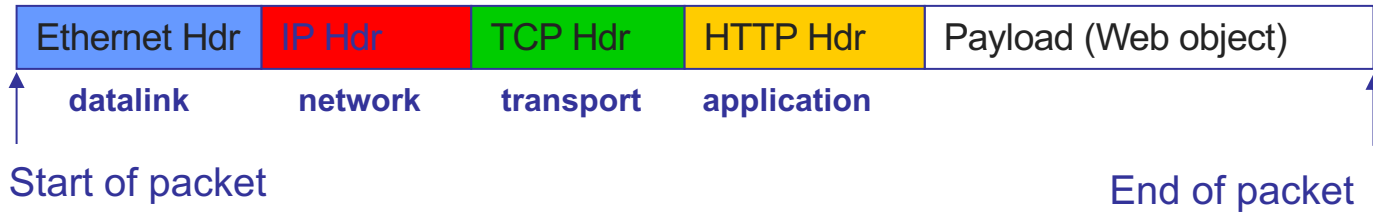
# Internet Protocol Suite





# Encapsulation via Headers

- Typical Web packet



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



# 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**
  - ◆ Identify the beginning and the end of the bitstream
- 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



# A sample bitstream

```
0100100100100000011101000110100001101001011011100110101100100000011101000110100001100001011101000010000001001001001000000
1110011011010000110000101101100011011000010000001101110011001010111011001100101011100100010000001110011011001010110010100
010100100000100100000011001110111001001100001011100000110100001000000110110101101111011100100110010100100000011011000110
1111011101100110010101101100011110010010000001110100011010000110000101101110001000000110000100100000011101000111001001100
1010110010100101110000101001000001001000000111010001110010011001010110010100100000011101110110100001101111011100110110010
1001000000110001101110010011101010110001101101001011000010110110000100000011100000111001001101111011100000110010101110010
011101000111100100010100100100101110011001000000110110001101111101101111011100000010110101100110011100100110010101100101
0010000001100011011011110110111001101110011001010110001101110100011010010111011001101001011101000111100100101110000101001
0000010010000001110100011100100110010101100101001000000111010001101000011000010111010000100000011011010111010101110011011
1010000100000011000100110010100100000011100110111010101110010011001010010000001110100011011110010000001110011011100000110
0001011011100001010010100110110111001000000111000001100001011000110110101110100011100110010000001100011011000
010110111000100000011100100110010101100001011000110110100001000000110010101110110011001010111001001000000100110
0010000010100111000101110000101001000110011010010111001001110011011101000010110000100000011101000110100001100101001000000
111001001101111011011101110100001000000110110101110101011100110111010000100000011000100110010100100000011100110110010101
10110001100101011000110111010001100101011001000010111000010100100001001111001001000000100100100010000101100001000000110
100101110100001000000110100101110011001000000110010101100110010101100011011101000110010101100100001011100001010010011
000110010101100001011100110111010000101101011000110110111011100110111010000100000011100000110000101110100011010000111001
1001000000110011001110010011011101101101001000000111001001101111011011110111010000100000011000010111001001100101001000
0001110100011100100110000101100011011001010110010000101110000101001001001011011100010000001110100011010000110010100100000
0111010001110010011001010110010100101100001000000111010001101000011001010111001101100101001000000111000001100001011101000
1101000011100110010000001100001011100100110010100100000011100000110110001100001011000101110000101100001010010
000010010000001101101011001010111001101101000010000001101001011100110010000001101101011000010110010001100101001000000110
0010011110010010000001100110011011110110110001101011011100110010000001101100011010010110101100101001000000110110101100
1010010110000010100101010001101000011001010110111000100000011000100111001001101001011001000110011101100101011100110010000
```



# A sample bitstream

```
0100100100100000011101000110100001101001011011100110101100100000011101000110100001100001011101000010000001001001001000000
1110011011010000110000101101100011011000010000001101110011001010111011001100101011100100010000001110011011001010110010100
010100100000100100000011001110111001001100001011100000110100001000000110110101101111011100100110010100100000011011000110
1111011101100110010101101100011110010010000001110100011010000110000101101110001000000110000100100000011101000111001001100
1010110010100101110000101001000001001000000111010001110010011001010110010100100000011101110110100001101111011100110110010
1001000000110001101110010011101010110001101101001011000010110110000100000011100000111001001101111011100000110010101110010
011101000111100100010100100100101110011001000000110110001101111101101111011100000010110101100110011100100110010101100101
001000000110001101101111011011100110111001100101011000110111010001101001011101100111100100101110000101001
000001001000000111010001110010011001010110010100100000011101000110100001100001011101000100000011011010111010101110011011
10100001000000110001001100101001000000111001101110101011100100110010010000001110100011011110010000001110011011100000110
000101101110000101001000110110111100100000011100000110000101100110101101100101110100011100110010000001100011011000
01011011100010000001110010011001010110000101100001011000001000000110010101110110011001010111001001000000100110
001000001010011100010111000010100100011001101001011100100111001101101000010110000100000011101000110100001100101001000000
111001001101110110111101110100001000000110110101110101011100110111010000100000011000100110010100100000011100110110010101
10110001100101011000110111010001100101011001000010111000010100100001001111001001000000100100100010000101100001000000110
1001011101000010000001101001011100110010000001100101011000110010101100011011101000110010101100100001011100001010010011
000110010101100001011100110111010000101101011000110110111011100110111010000100000011100000110000101110100011010000111001
1001000000110011001110010011011101101101001000000111001001101111011011110111010000100000011000010111001001100101001000
0001110100011100100110000101100011011001010110010000101110000101001001001011011100010000001110100011010000110010100100000
0111010001110010011001010110010100101100001000000111010001101000011001010111001101100101001000000111000001100001011101000
110100001110011001000000110000101110010011001010010000001110000011011000110010100100000011100001011100001010010
000010010000001101101011001010111001101101000010000001101001011100110010000001101101011000010110010001100101001000000110
0010011110010010000001100110011011110110110001101011011100110010000001101100011010010110101100101001000000110110101100
1010010110000010100101010001101000011001010110111000100000011000100111001001101001011001000110011101100101011100110010000
```



# 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



# 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)
  - ◆ Why 53?  $48 + 5$ . Why 48?



# 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

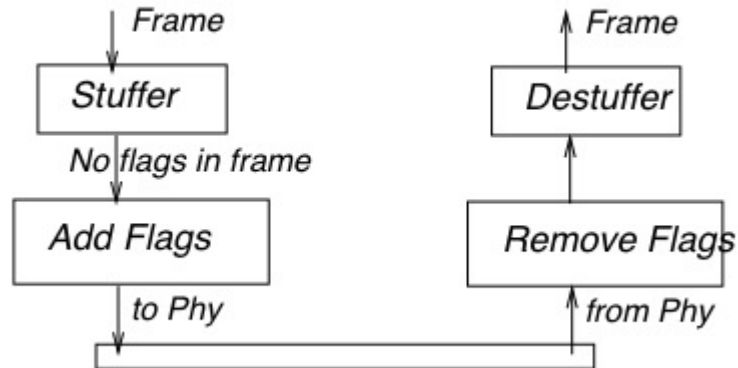
---

- 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 “unstuffs” 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

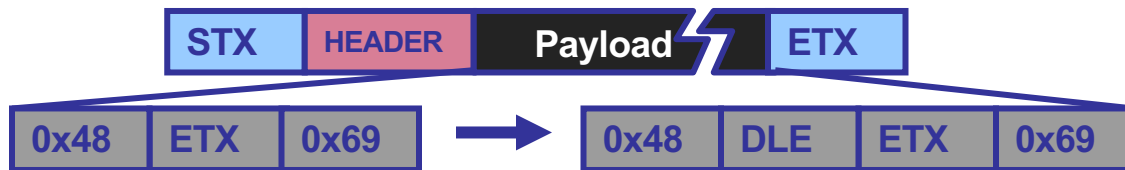
011111100001110111011111011111001  
011111**0**100001110111011111**0**011111**0**001

- 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)



# Byte Stuffing aka “Escaping”

- Same as bit stuffing, except at byte (character) level
  - ◆ Generally have two different flags, **STX** and **ETX**
  - ◆ Found in PPP, DDCMP, BISYNC, etc.
- Need to stuff if either appears in the payload
  - ◆ Prefix with another special character, **DLE** (data-link escape)
  - ◆ New problem: what if DLE appears in payload?
- Stuff DLE with DLE!
  - ◆ Could be as bad as 50% efficient to send all DLEs



# For Next Class

---



- Read 2.4
- Project out Friday; go to discussion today to review C!