6.033 Computer System Engineering
Spring 2009

6.033 Lecture 10: Layering, Link Layer

Let's design a data network.
  How do we organize our design?

Reminder: mesh networks.
  [draw host/switch/mesh picture]

Idea: protocols.
  Two entities (peers) are talking.
  Protocol formally defines structure of conversation.
  Typically sequence of messages -- packets.
  Format:
    Specific set of message types.
    What does each bit of the message mean?
    [example: ethernet paper, dst, src, data, checksum]
  Rules for what happens next, state machines.
  Semantics: what does it mean?
  Often you can learn all you need from the formats...

But there are many levels of communication.
  [draw the on *mesh* net, nested...]
  Analog waveforms on a wire.
  Messages from one switch to the next.
  Similarly for host / switch.
  Application to application.
  There are many interacting protocols here.
  How do we organize them?

Idea: layers.
  Intuition: protocols nest.
  Inner protocols building blocks for outer ones.
  Let's formalize that way of organizing multiple protos.
  Choose and define a useful protocol. [box <--> box]
  Define s/w interface so other layers can use it. [stack up]
  It may in turn use more primitive layers. [stack down]
  Can build up functionality this way.
  But use modules, abstraction to control complexity.
  Hard part: choosing useful layer boundaries.

6.033 layer model:
  [draw stacks for host, switch, host]
  Physical: analog waveforms -> bits.
  Link: bits -> packets, single wire.
  Network: packet on wire -> packet to destination.
  End-to-end: packets -> connections or streams.
  Application.
  physical almost always tightly bound to Link.
  And application isn't a generally useful tool.
  Note: layer may have many clients
    multiple apps using e2e, multiple e2e using net
    need a way to multiplex them
  Note: net layer may use multiple links
  So the real picture in one host
    app1 app2 app3
      TCP UDP

```
        IP
     Eth WiFi
  Layers == outline of data networking topic.

Stack of layers:
  Repeated scheme for layer interaction
  Each layer adds/strips off its own header.
    Encapsulates higher layer's data as "payload".
    [add to pkt diagram; interior header &c]
  Each layer may split up higher layer's data.
    [stream split into payloads of packets]
  Each layer multiplexes multiple higher layers.
    [put protocol # field into packet]
  Each layer is (mostly) transparent to higher layers.
    data delivered up on far side == data in

Physical Layer
  Built out of wires.
  We will only talk about one direction.
  Just build two for bi-directional communication.
  [picture: Sender, wire, Receiver]
  can send "signals": voltages, optical power levels.
  What we want is a stream of bits.
  problems:
    which value, 0 or 1? vs noise
    when to sample at receiver

Straw man one:
  3 wires: data bit, ready, ack
  [picture]
  speed of light limits data speed.
    coast-to-coast: 30 bits/sec
    one mile: 200 kbits/sec

Straw man two:
  Assume sender and receiver have accurate oscillators.
    I.e. same frequency.
  [draw the two oscillators]
  Sender sends one bit per clock period.
  [picture of waveform. mark sampling spots.]
  Easy problem: phase changed by wire delay.
  Hard problem:
    You can probably build one accurate clock.
    You probably can't build two that agree.
    You can never assume synchronized clocks in dist systems.

Straw man three:
  Send the clock on a separate wire?
  Expensive, skew between wires if fast or long

The right answer:
  Idea: clock recovery from the signals themselves.
  Receiver keeps a local clock.
  Adjusts it if signal transitions aren't on clock transitions.
  balanced signals: you need transitions! no strings of zeroes.
  So can't directly encode 0/1 as high/low signals.
```

```
  Ethernet/Manchester has four wave-forms == "symbols"
    LH: 0 bit
    HL: 1 bit
    LL: idle
    HH: unused
    so only 2 of 4 codes used. [picture]
  Could be more efficient: 8 symbol values per 2 bits. [picture]
    LLH: 00
    LHL: 01
    HLL: 10
    HLH: 11
    others unused, e.g. LLL, to aid clock recovery
  Why not 128 signals per 127 bits?
    single signal error wrecks 127 bits...

Could we declare this to be a layer?
  Is it a generally useful abstraction?
  No: no way to share. Just bits.

Link Layer: Bits to Packets/Frames
  Why packets? Why not continuous data?
    Need to inter-mix packets from different conversations.
  How do we delimit packets?

Packet framing
  Link has two states: idle, sending data.
  Need to signal the state transitions.
  Idle -> Sending (start of packet).
  Sending -> Idle (end of packet).
  Let's signal idle with all low signals.
    LL not used by Manchester to send bits
  Start of packet easy: just send a one bit.
    Might want some longer alternating sequence for clock recovery.
    e.g. 8x LH followed by HL, in case rcvr misses first few LH
    Called a preamble.
    [packet picture: preamble, data, end-mark]
  mark end of packet?
    special code that cannot ordinarily appear in the data stream.
    transform data stream to eliminate magic bit sequence.
    OR two zero signals in manchester scheme.
  what if end-of-packet signal corrupted? wreck next packet?
  why don't we use a length field at start instead of a special code?

Can we put an interface here, and make a layer?
  Not yet: how do we know whose packet is whose?
  Really indicating which layer above us.
  All we need is a number in the packet header.

We could turn this into a layer.
  By convention, link layer detects errors.

What should we do about errors?
  E.g. noise corrupts some bits.
  We need to define the link layer abstraction.
  What can it reasonably *guarantee*?
  Could just ignore the problem, let higher layer deal.
```

    Careful higher layers probably check anyway.
  Can we guarantee to deliver all packets perfectly?
    Hard for link layer to make this guarantee.
    Ask previous switch to re-send? Complex, buffering.
    What if switch failes? => link cannot guarantee!
    Maybe app doesn't care anyway, e.g. real-time voice.
  Realistic answer:
    Guarantee that all packets delivered are correct.
    We're allowed (expected) to discard corrupted packets.
    Example of "best effort" contract.
    This means link layer just *detects* bit errors.

Error detection
  best plan depends on error patterns.
    single bit due to short noise.
    microwave oven wipes out many packets.
    coding wipes out one block of e.g. 2 bits.
  example: xor byte at the end.
    [picture: vertical column of bytes, xor at bottom]
    detects any one bit error, some multi-bit errors.
    doesn't detect e.g. byte exchange.
    much more sophisticated schemes available!
    usually called "checksums".

Error correction
  A single bit error causes link layer to discard whole packet.
  Seems wasteful.
  We can often squeeze out a little more performance.
  Here's a plan for correcting one-bit errors.
  [add a 9th bit to each byte.]
  Still have to discard if there are two incorrect bits.

What does our link-layer protocol look like?
  [refer back to Protocol board]
  Format: mostly framing, also proto #.
  What's the abstraction?
    exposes packets. (not e.g. files...)
    allows sharing among higher protocols.
    best-effort: may not deliver a packet.
    will rarely give you a corrupted packet.
  Ethernet MAC operates at about this level.

Now we can move a packet along a single wire.
  Next lecture: move packet along a path of switches.