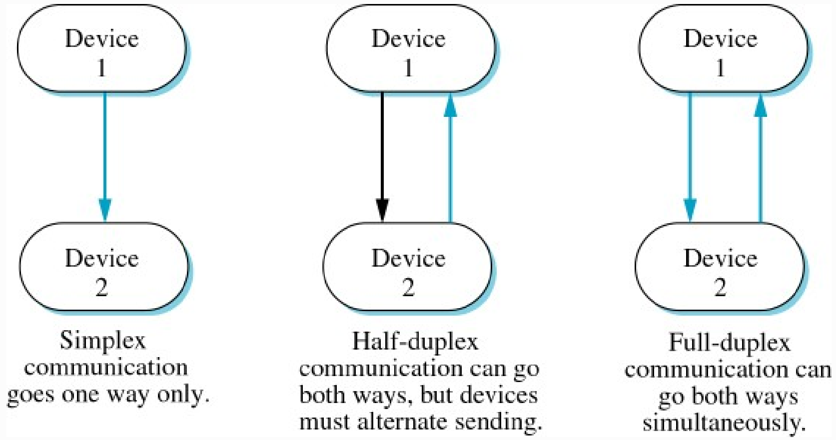
Lectures 12, 13, 14:
Connections, Protocols, Link Control, LANs

Nevil Brownlee

314 S2T 2007

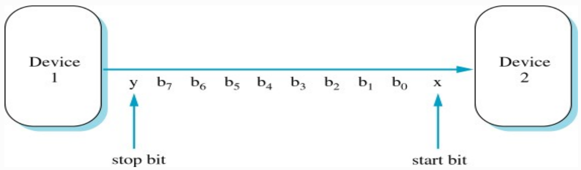
Transmission Modes 4.3

- Parallel (many wires) or Serial (one wire)
- Direction-related

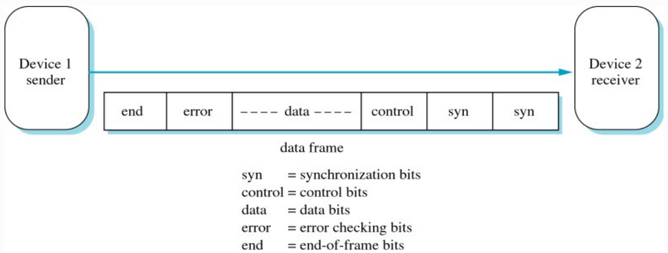


Transmission Modes 4.3

- Time-related
  - asynchronous: may start/stop at any time



  - synchronous: uses a continuous clock



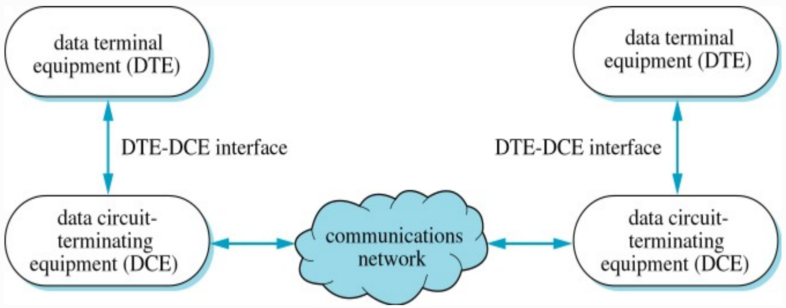
  - isochronous: imposes gaps to match transmission rates

Interface Standards 4.4

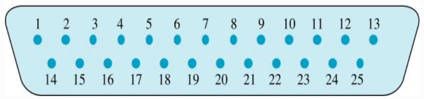
- There are lots of 'standard' interfaces for connecting devices together
- Shay has good descriptions of:
  - EIA-232 (RS-232) <= we only look at this one
  - USB
  - IEEE 1394 (Firewire)
  - X.21

RS-232 Serial Interface

- Connects DTE (computer) to DCE (modem)

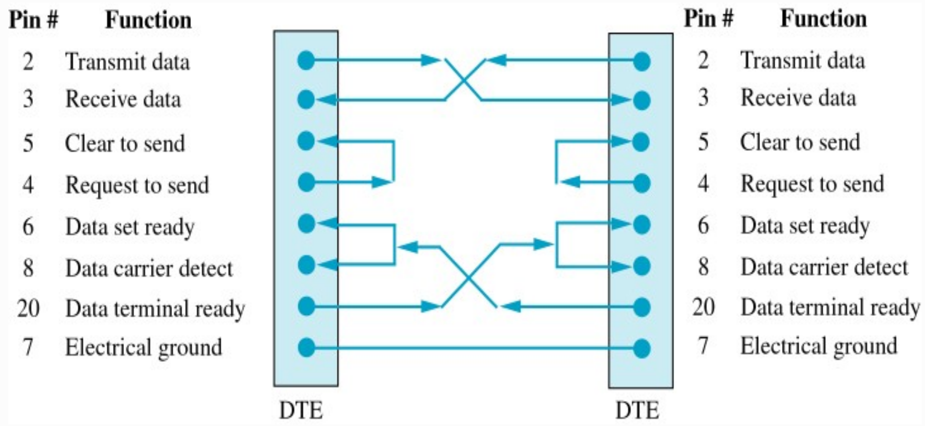


- 25-pin connector, we normally use only 9



RS-232 Serial Interface

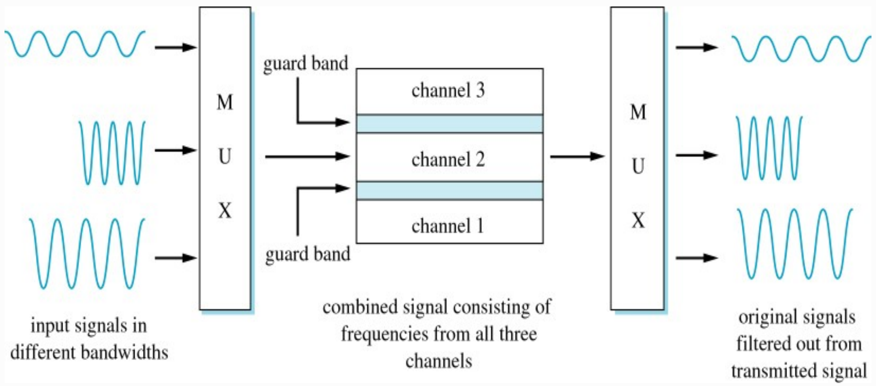
- *Null Modem* for connecting two DTEs



- *Not used here:* pin 22 = Ring Indicator, pin 1 = Protective Earth

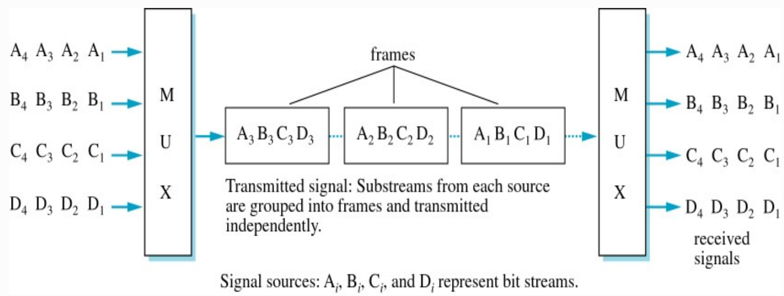
Multiplexing 4.5

- Ways of carrying several different connections over a common link
- Frequency-Division (FDM):



Multiplexing (2)

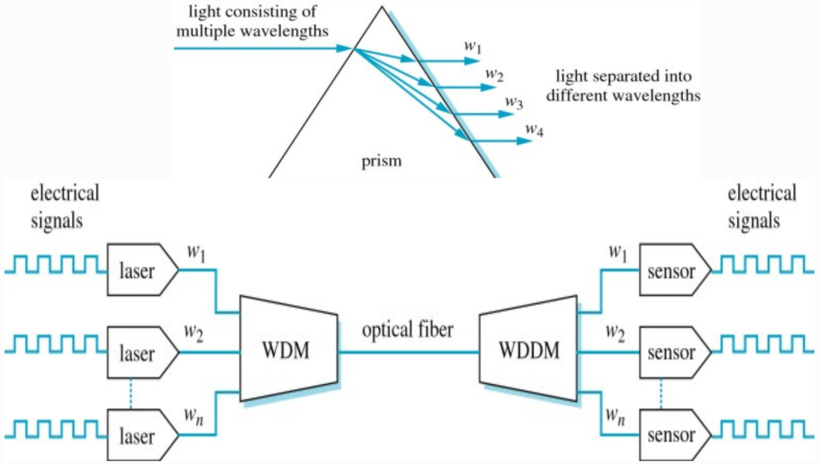
- Time-Division (TDM):



- Statistical Multiplexing
  - Much the same as TDM, but doesn't use fixed time allocations (slots)
  - Receiver must be able to identify incoming frames

Multiplexing (3)

- Wave-Division (WDM):



Flow Control 8.1

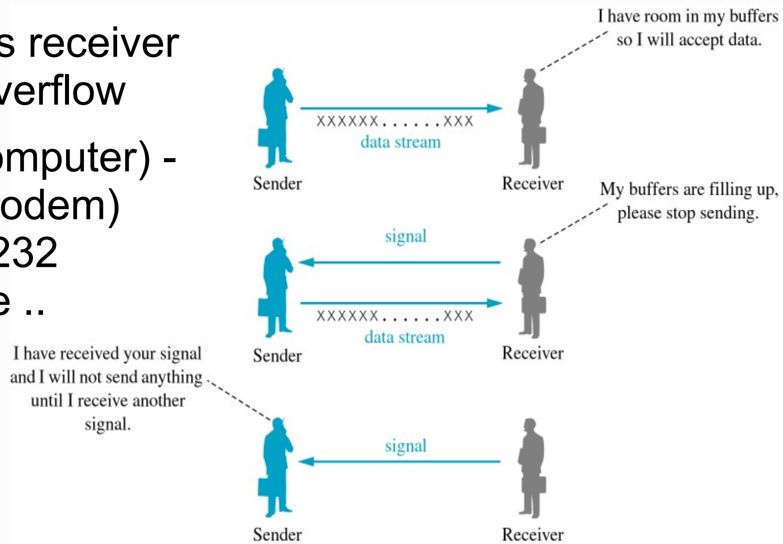
- Need for flow control
  - how can we send long messages, e.g. big files?
  - what happens when messages get lost, or are corrupted when they arrive?
  - what if the receiving *host* is busy, i.e. slow to accept incoming data?
  - how will a sender cope with lost (undelivered) messages?
  - will both hosts be able to send/receive at the same time?

What *is* Flow Control?

- Messages are broken into *frames*
- Flow Control defines
  - “the way frames are sent, tracked and controlled”
  - may be simple or complex
- Many examples of protocols around us, e.g. traffic rules (Road Code), 'phone conversations
- How can we be sure that a protocol is *correct?*
  - works properly
  - will never suddenly 'freeze'

Signaling 8.2

- *Receiver tells sender when it's ready to receive*
- Prevents receiver buffer overflow
- DTE (computer) - DCE (modem) via RS-232 interface ..



X-ON/X-OFF

- Over the DTE-DCE path ..
  - send ASCII X-OFF (0x13, ^S) to stop transmission
  - send X-ON (0x11, ^Q) to start it again
- This is *in-band* signalling, i.e. send signal on same path as data

- How quickly does the transmitter stop sending?
- How can we send 0x11 or 0x13 to the receiver?

Frame-oriented Control 8.3

- Idea is to break large sequences of chars into smaller *frames*
- Frames are sent from one *user* (higher protocol layer) to another

- *Unrestricted* protocol
  - simply assume it's always safe to send
  - not really a useable protocol!

Stop-and-Wait

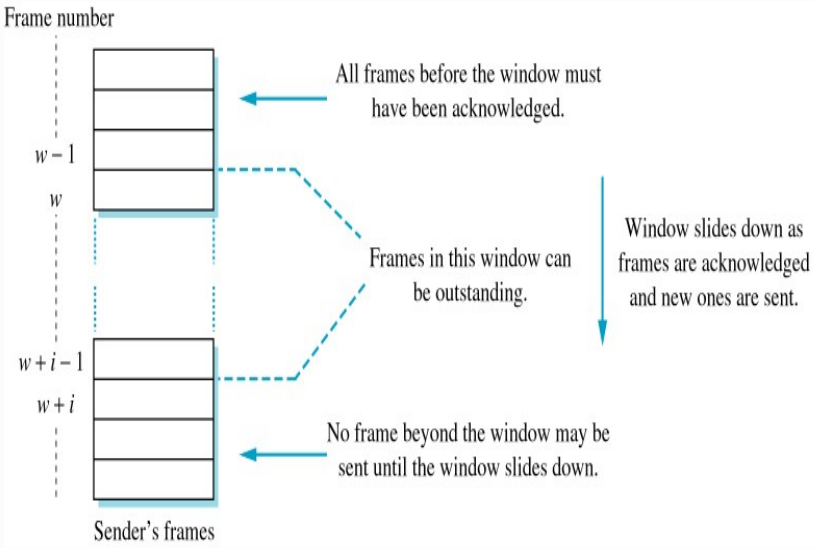
- Sender:
  - send frame, wait for ACK or NAK
  - if NAK, send frame again. Repeat unil get ACK
- Receiver:
  - receive frame, check for errors
  - if OK, send ACK. otherwise send NAK

- No way to handle lost (therefore not ACKed) frames

Protocol Efficiency: Effective data rate

- Shay derives formulae, we “just work it out”
- Remember, *velocity* = *distance / time*
  - in wire or fibre, *v* is about 2/3 the speed of light, i.e. 2×10^8 m/s
  - Auckland-Hamilton is about 120 km, so a byte takes $(120 \times 10^3)/(2 \times 10^8)$ = 0.6 ms to get there
  - If we send a 1500-Byte frame at 10 Mb/s, it will take $(1500 \times 8) /(10 \times 10^6)$ = 1.2 ms to transmit
  - Assume that ACK is a 64-Byte frame, 0.0512 ms
  - Therefore, to send frame and receive ACK takes roughly 1.25 + 2 x 0.6 = 2.45 ms
  - Effective bit rate is $(1500 \times 8)/(2.45 \times 10^{-3})$ = 4.9 Mb/s

Sliding Window 8.4



Sliding Window / *Go-back-n*

- Idea here is to have a maximum of *i* frames on the wire at any time. *i* is the *window size*
- Each frame has a sequence number, sender must hold each frame until it is ACKed
- Sender keeps track of *w*, sequence number of first (of *i frames*) in window. When frame *w* is ACKed, sender can forget it
- Window does not move until earliest frame has been ACKed

Go-back-n

- Shay develops a frame format for two-way communication

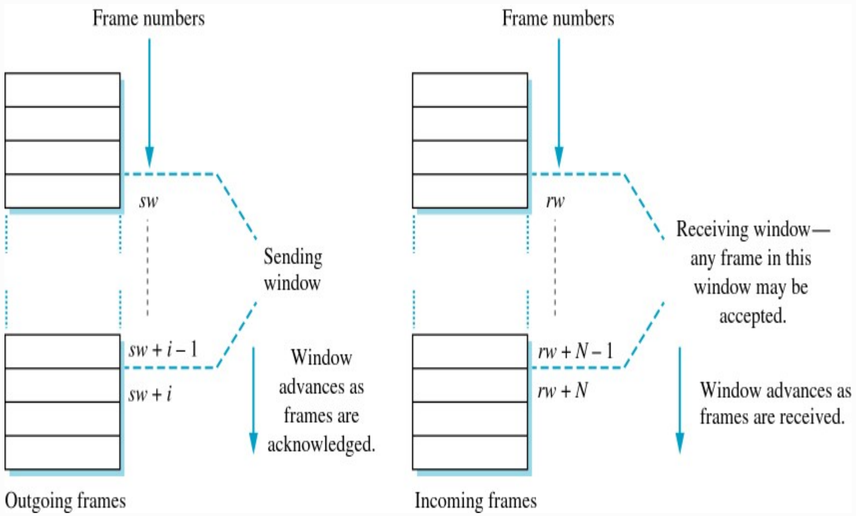
Source	Destination	Number	ACK	Type	...Data...	CRC

- Data frame in one direction can carry an ACK for the other direction, i.e. a *piggy-backed ACK*
- To handle lost frames, he has an *ACK timer* at the receiver ..
- and a *frame timer* at the transmitter

Sequence Numbers

- Sequence Numbers fit in a K-bit field; there can be at most 2^K frames in the window
- K should be big enough to handle the maximum window size we expect to use
- They are *unsigned* numbers, and can *wrap,* i.e. count through 2^K-2, 2^K-1, 0, 1, 2, ...
  You can think of the sequence numbers as being arranged in a circle
- What happens if a host crashes and restarts?
- Some protocols used *lollipop sequence numbering* to handle restarts! (see Wikipedia)

Selective Repeat 8.5



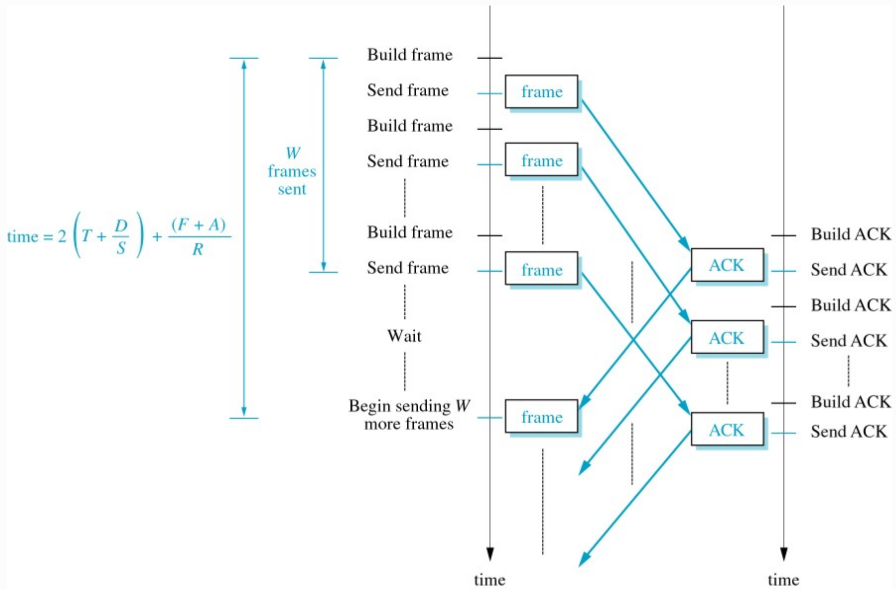
Selective Repeat (2)

- Any frame can be ACKed, specifying it's sequence number
- Frames arriving out of sequence are *buffered* until earlier frames have been ACKed
- When a NAK is received, only the NAKed frame is resent (Go-Back-n resent the whole window!)
- If a frame timer expires (no ACK or NAK), only the timed-out frame is resent
- Piggy-backed ACK acknowledges the *last frame delivered to the user,* so the sender knows that all frames up to that one have been safely received

Efficiency of Sliding Window Protocols 8.6

- For a particular window size, message size, transmission speed and link distance, we can "just work it out," as we did for stop-and-wait
- *We assume no lost or damaged packets !*
- Two cases
  - we get our first message ACKed before we've sent a whole window. That allows us to keep sending at full link speed
  - we have to wait for an ACK after sending a window, then we can send another window. Shay has a diagram illustrating this ..

Sending whole window and waiting



Numerical examples

- Sending 100x 1500B frames in 20-frame windows, Auckland-Hamilton on a 10 Mb/s link
  - as for Stop-and-Wait: 1.2ms to send frame, 1.2ms round-trip time.
    Any window > 2 frames can run at full speed, 10 Mb/s
- As above, but with 64B frames
  - send time is $(64 \times 8)/(10 \times 10^6)$ = 0.0512 ms
  - time to send 20 frames = 20 x 0.0512 = 1.024 ms
  - first ACK returns after 1.2+2*0.0512 = 1.3024 ms
  - effective bit rate is (20 * 64 * 8)/1.3024 = 7.862 Mb/s
  - note the effect of using a *small frame size !*

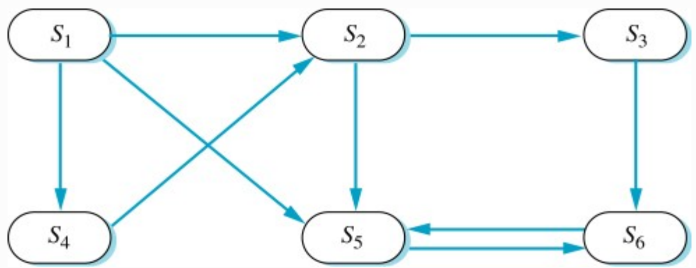
Bandwith-Delay Product (BDP)

- BDP for a link = data rate x link delay
- Auckland-Hamilton at 10 Mb/s:
  BDP = 10 Mb/s x 0.6 ms = 16.67 kb
  = 2083 B
- This is the maximum number of bits we can have 'on the wire'
- Need to have buffers at least this big so that transport protocol can keep the link busy
- Bigger frames sizes help to keep the link busy – less *protocol overhead*

Protocol Correctness 8.7

- Shay discusses two ways to describe systems:
  - Finite State Machines
  - Petri nets
- Finite State Machine models a system as being in one of a finite set of *states*
- State Transition Diagrams (STDs) are graphs, each vertex represents a state, and each edge a transition between states
- Petri nets are more detailed, we won't discuss them further

State Transition Diagrams



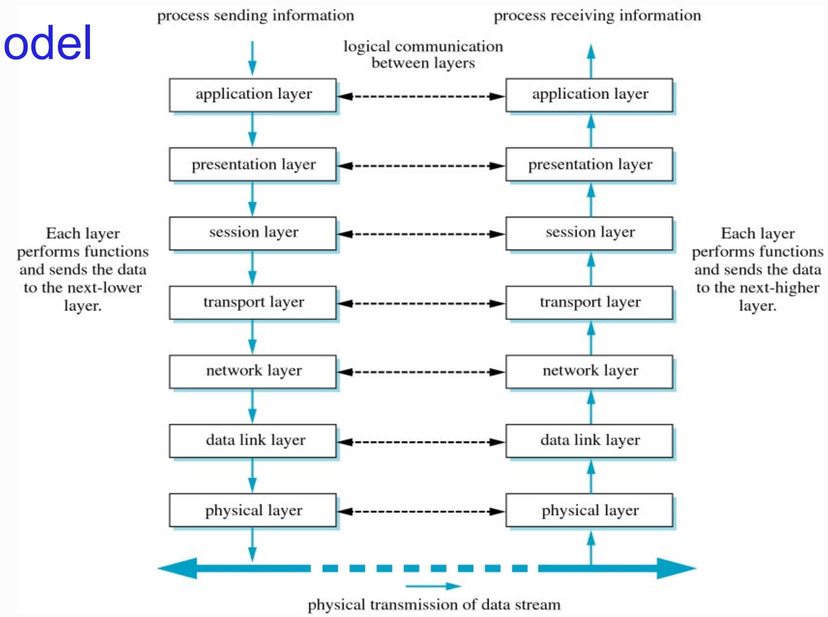
- Look for problems on graph
  - No edges pointing to S_1
  - $S_5 - S_6$ is an infinite loop
- This kind of analysis helps find flaws
  - *it doesn't prove correctness!*

Protocol Layers, the OSI Model 1.4

- Layers are an abstraction, they provide a simple view of what happens in a communication system
- Layer *n*
  - provides services to layer *n+1*
  - uses services from layer *n-1*
- Generally we implement systems this way, but sometimes we may find it useful to peek between layers, or 'break layer purity'

OSI Model

7
6
5
4
3
2
1



- OSI has 7 layers, TCP/IP collapses 5-7 into 5

Introduction to LANs 9.1

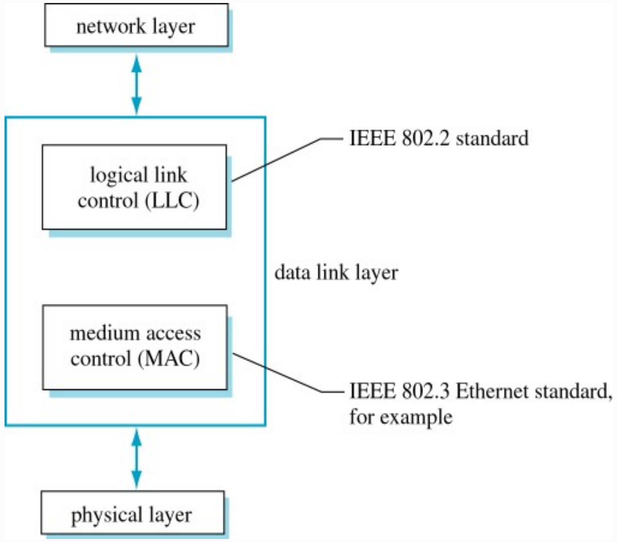
- LANs connect many hosts (devices) together
- Link medium may be copper (coax or UTP), fibre or wireless
- Topology may be
  - *bus:* hosts share the medium by taking turns
  - *ring:* access is controlled by pasing a token
- Ethernet – today's most common LAN physical layer – uses a bus topology
- Point-to-point link is a LAN with only two hosts

LAN Layers

- Layer 1 is the Physical layer. On this layer, you've already looked at signaling and modulation methods
- Layer 2, the Link layer, is where hosts talk to each other. Protocols here send frames (packets) to other hosts, and receive frames in response
- Layer 3, the Network layer, is used to pass packets between LANs. For example, we often use IP to pass frames between Ethernet-connected hosts

Data Link Control 9.2

- Link layer is divided in two – LLC and MAC
- Shay presents HDLC, a fore-runner of IEEE 802.2
- These are bit-oriented protocols



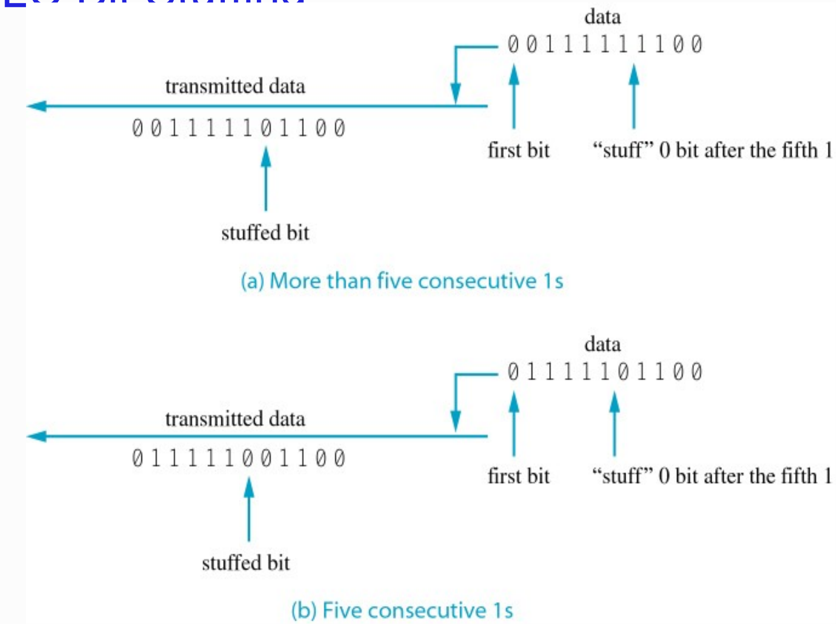
HDLC Frame Format

- *Flag* pattern, 01111110(six 1s) marks start and end of frame. Receiver watches medium for flags

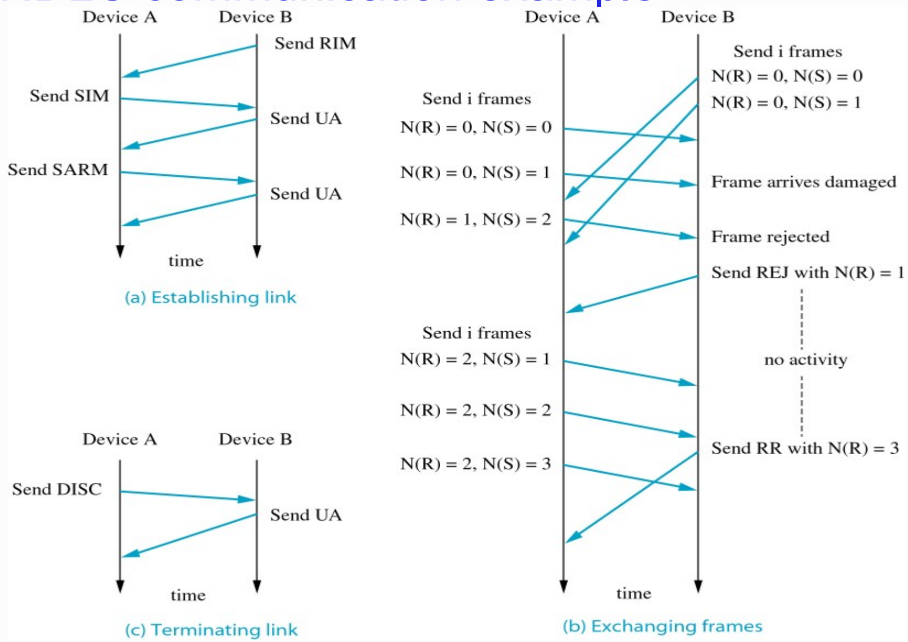
number of bits:	8	8 or 16	8 or 16	variable	16 or 32	8
	Flag	Address	Control	------- Data -------	FCS	Flag

- How do we send the flag pattern within the data part of the frame?

HDLC Bit Stuffing



HDLC communication example



802.2 Header Formats

DSAP address 8 bits	SSAP address 8 bits	Control field 8 or 16 bits	Information field N*8 bits

- DSAP, SSAP are Service Access Point addresses
  - 04 = IBM SNA, 06 = IP,
    AA = SNAP (Subnetwork Attachment Point)

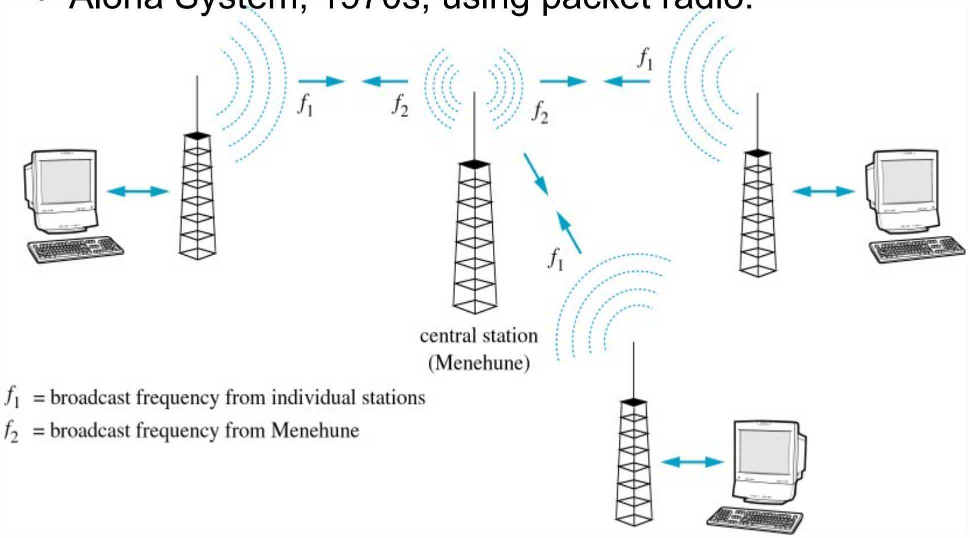
AA AA 03 LLC	00 00 00 3 octet OUI	08 00

2-octet type field

- OUI = Organisation Unique Identifier
- Type field values are Ethernet type (Ethertype) values
  - 0800 = IP, 0806 = ARP, 6003 = DECnet phase IV, ...

Contention Protocols 4.7

- Basic idea: Hosts must *share* the medium
- Aloha System, 1970s, using packet radio:



Aloha Protocol

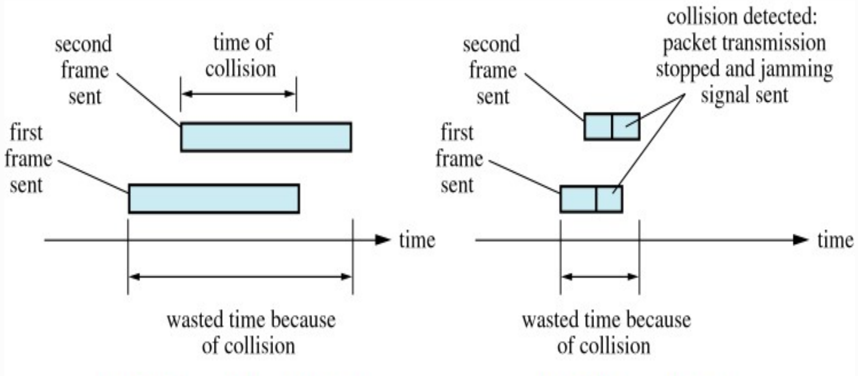
- Any host can broadcast a message to Menehune at any time
- If the message is received correctly, Menehune ACKs it (on a different frequency)
- If two host transmissions overlap (and interfere) the message is lost
- If a message is not ACKed the host assumes it was lost, waits a random time, then resends
- Worked and was simple, but not a very efficient use of the medium

Carrier Sense Multiple Access (CSMA)

- Like Aloha, *listen to medium* for any activity
- If no activity, transmit; otherwise wait
- Can still get collisions, various ways to reduce them:
  - use 'slot time,' hosts can only transmit at start of a slot
  - random choice, probability *p*, to decide whether to transmit or wait for next slot
  - Fig. 4.44 compare various schemes

Collision Detection

- Start transmitting any time, but watch medium for a collision
- When collision detected, stop tranmitting, send jam signal
- This is CSMA/CD



(a) Collision without detection (b) Collision detection