Lectures 13 -15

Connections, Protocols, Link and Flow Control, LANs

Brian Carpenter

COMPSCI 314 S2C 2011

Assignment 2

- Due September 23 - start lab work *soon*
  - worth 5% of the final grade
- Understand the basic concepts of protocol layers and encapsulation by practical observation in the CS lab.
- Read and understand the tutorial document (16 pages) linked as "support material" at http://www.cs.auckland.ac.nz/courses/compsci314s2c/assignments/
- Learn to use the packet capture tools provided.
- Capture data for each part of the assignment, and analyse the results to answer the questions.
  - don't forget to include captured data in your PDF file

Background for Assignment 2

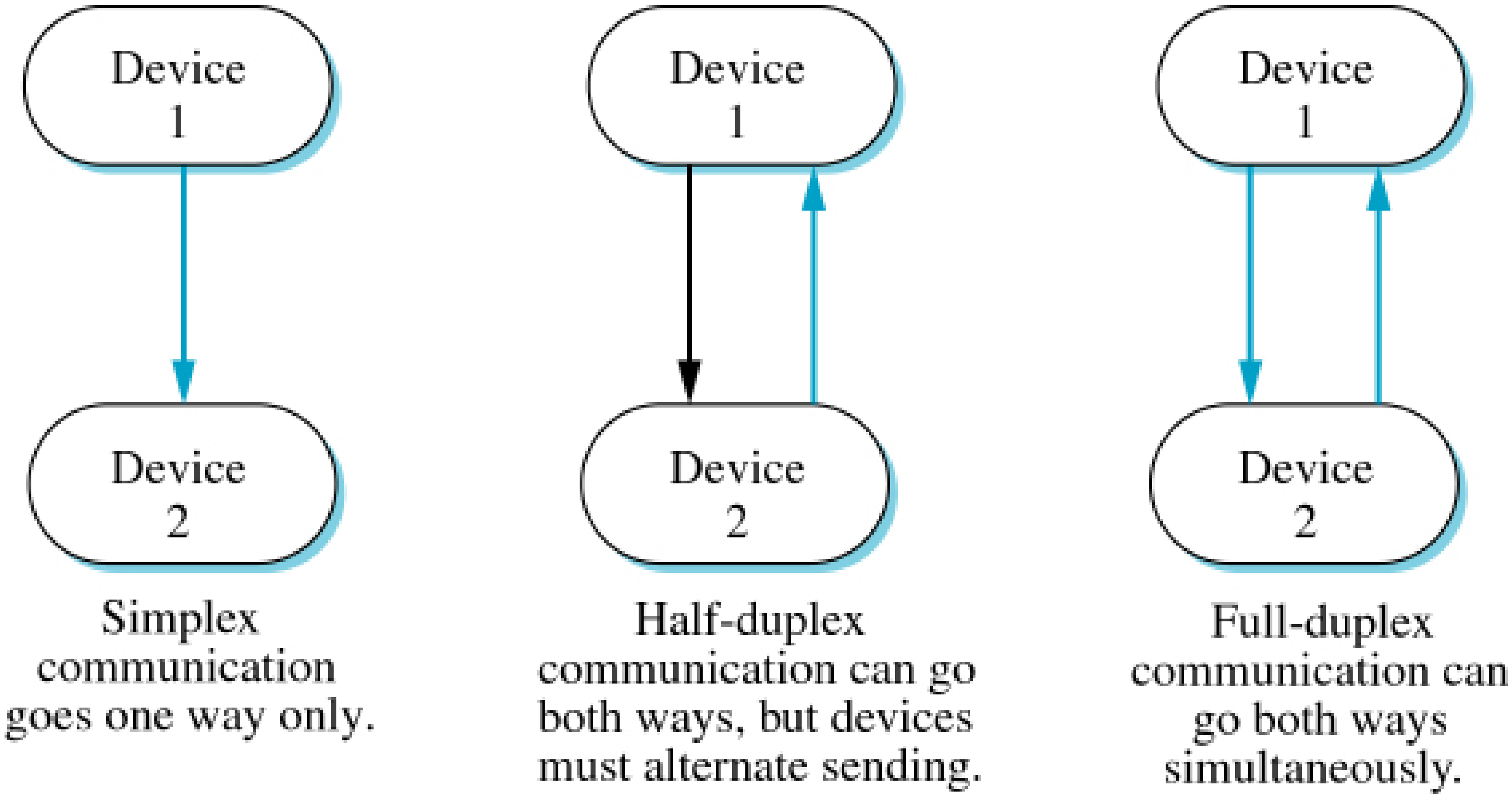
- Revise the material about TCP/IP from CompSci 215, or look ahead in Shay Chapters 9, 10 and 11.
- Packets include a sequence of *headers* corresponding to protocol layers.
- If we can inspect the packets, we can see and understand the headers.
- The CS lab machines have some tools for 314 students to allow capture and inspection of your packets.
  - capture packets to and from your IP address only, for privacy reasons

Tools installed on CS lab Windows

- wincap - a simple library supporting the windump command line utility.
- wireshark - a packet capture tool with a GUI.
- Using the tutorial, try out some packet captures like the examples given.
- Then you should be ready to start the assignment.

Transmission Modes - getting bits down a wire (Shay 4.3)

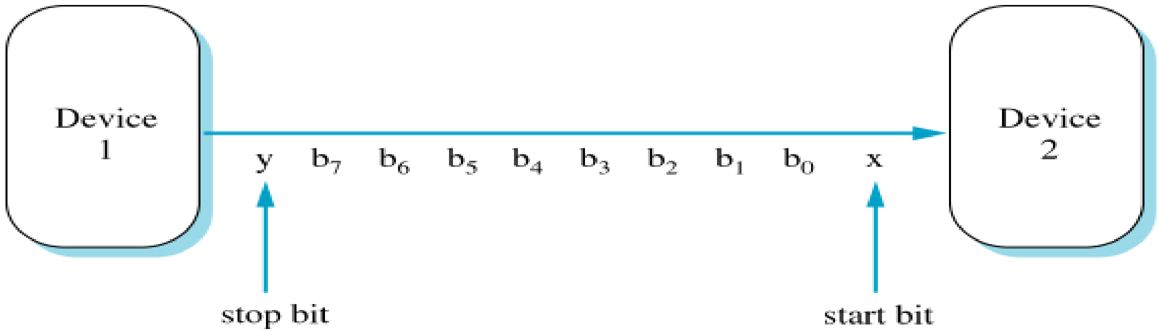
- Parallel (many wires) or Serial (one wire)
- Direction-related



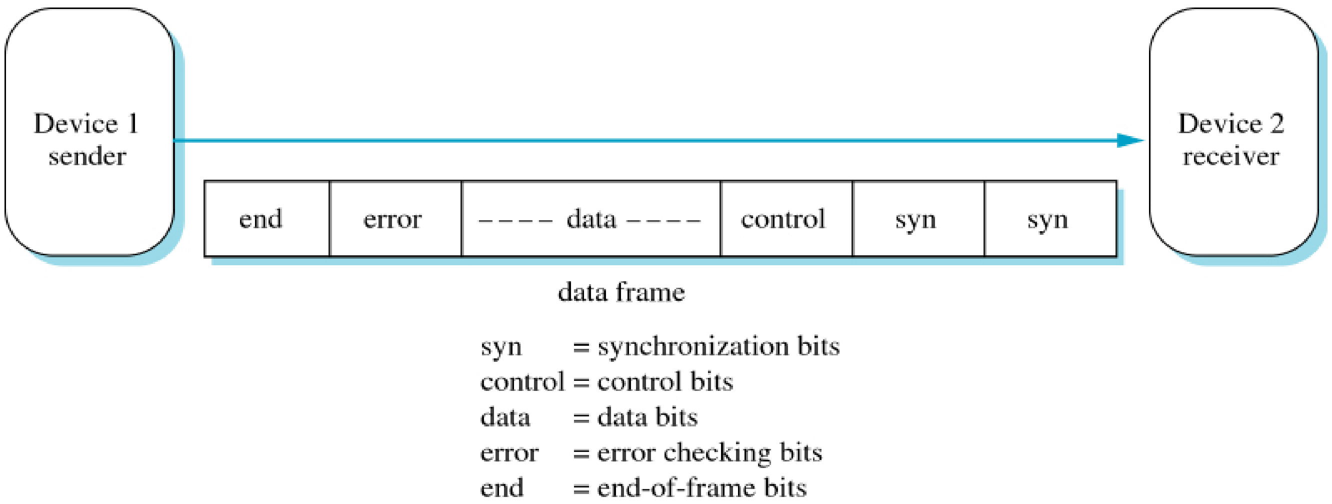
Diagrams from Shay are used throughout the lecture slides, without further acknowledgement.

Transmission Modes

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



  - synchronous: uses a continuous clock



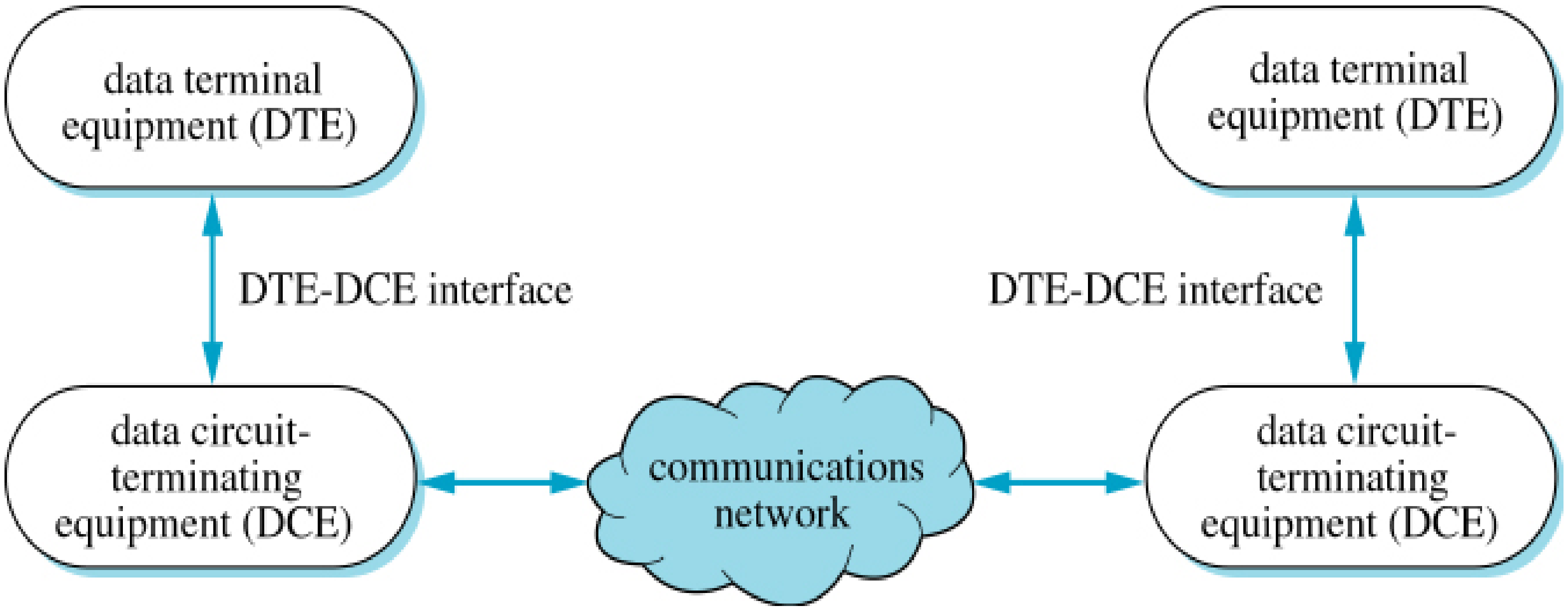
  - isochronous: inserts gaps to match transmission rates

Interface Standards (Shay 4.4)

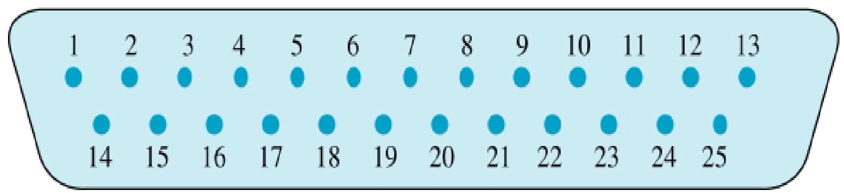
- There are lots of 'standard' interfaces for connecting devices together
- Shay has good descriptions of:
  - EIA-232 (RS-232) ← *we only look at this one, as a simple example, even though it's old.*
  - USB
  - IEEE 1394 (Firewire)
  - X.21

RS-232 Serial Interface

- Connects DTE (computer) to DCE (modem)

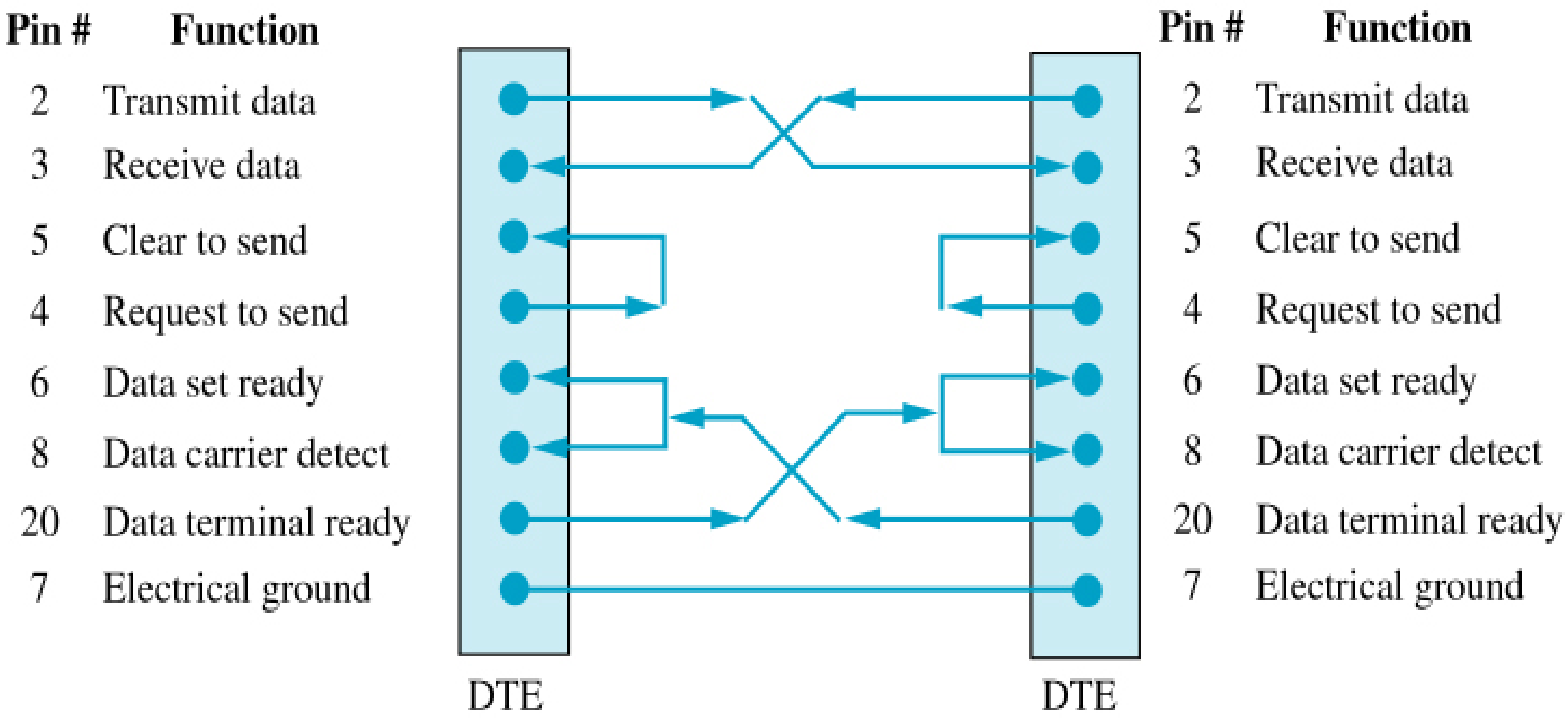


- 25-pin connector, we normally use only 9



RS-232 Serial Interface

- *Null Modem* for connecting two DTEs



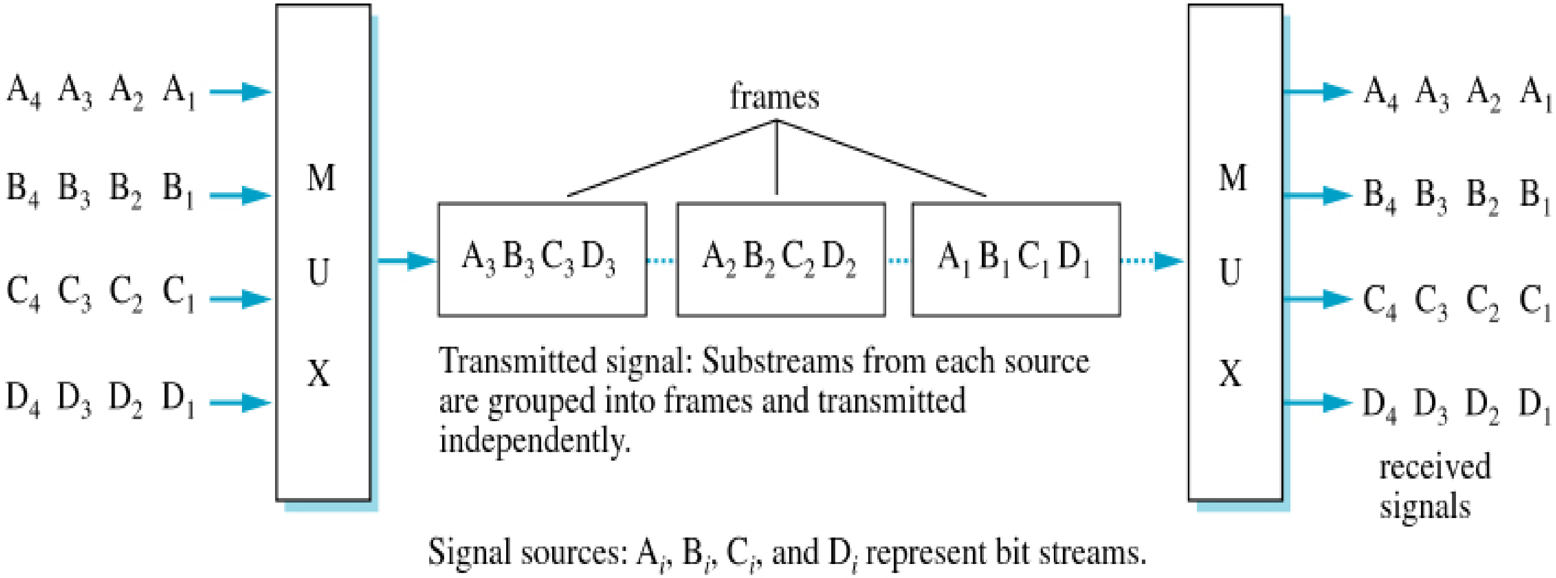
- *Not shown here:* pin 22 = Ring Indicator, pin 1 = Protective Earth

Multiplexing (Shay 4.5)

- Carrying several different connections over a common link.
  - Multiplexing makes one cable do the work of several separate cables.
- Useful because long distance cables are expensive and need to be shared.
  - Even within a building, you don't want a cable for every user back to a single central point.
- There are several methods of multiplexing.

Multiplexing (2): several bit streams, one channel

- Time-Division Multiplexing(TDM)



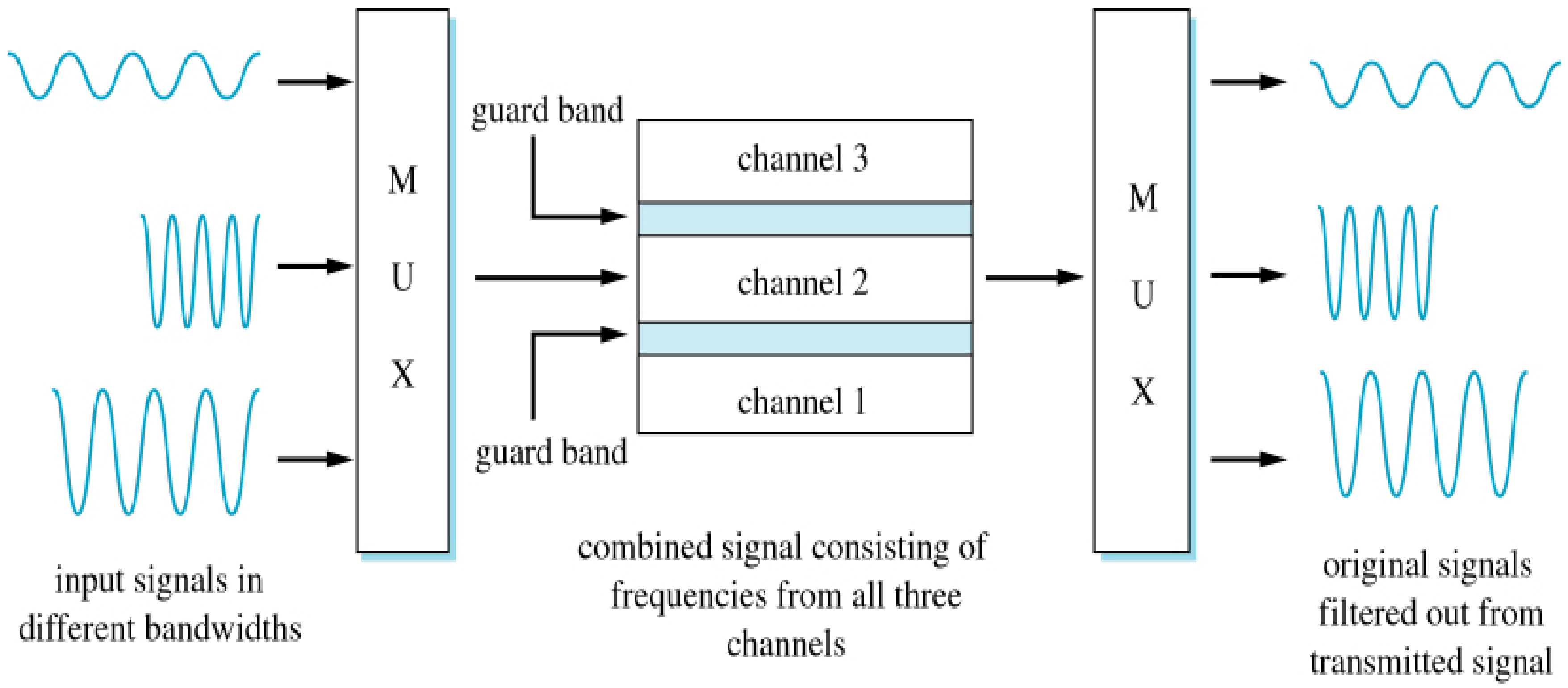
- A, B, C, D take regular turns
  - If a stream has nothing to send, its time slot is wasted.

Multiplexing (3): several bit streams, one channel

- Statistical Multiplexing
  - Similar to TDM, but doesn't use fixed time slots
  - If A has more to send than B, it will use more of the available time, and the timing will not be regular.
  - If a stream has nothing to send, another stream can use the channel.
  - Receiver must be able to identify incoming frames since there is no fixed time sequence.
  - Design must make sure that all bit streams get a fair share when the load is heavy.

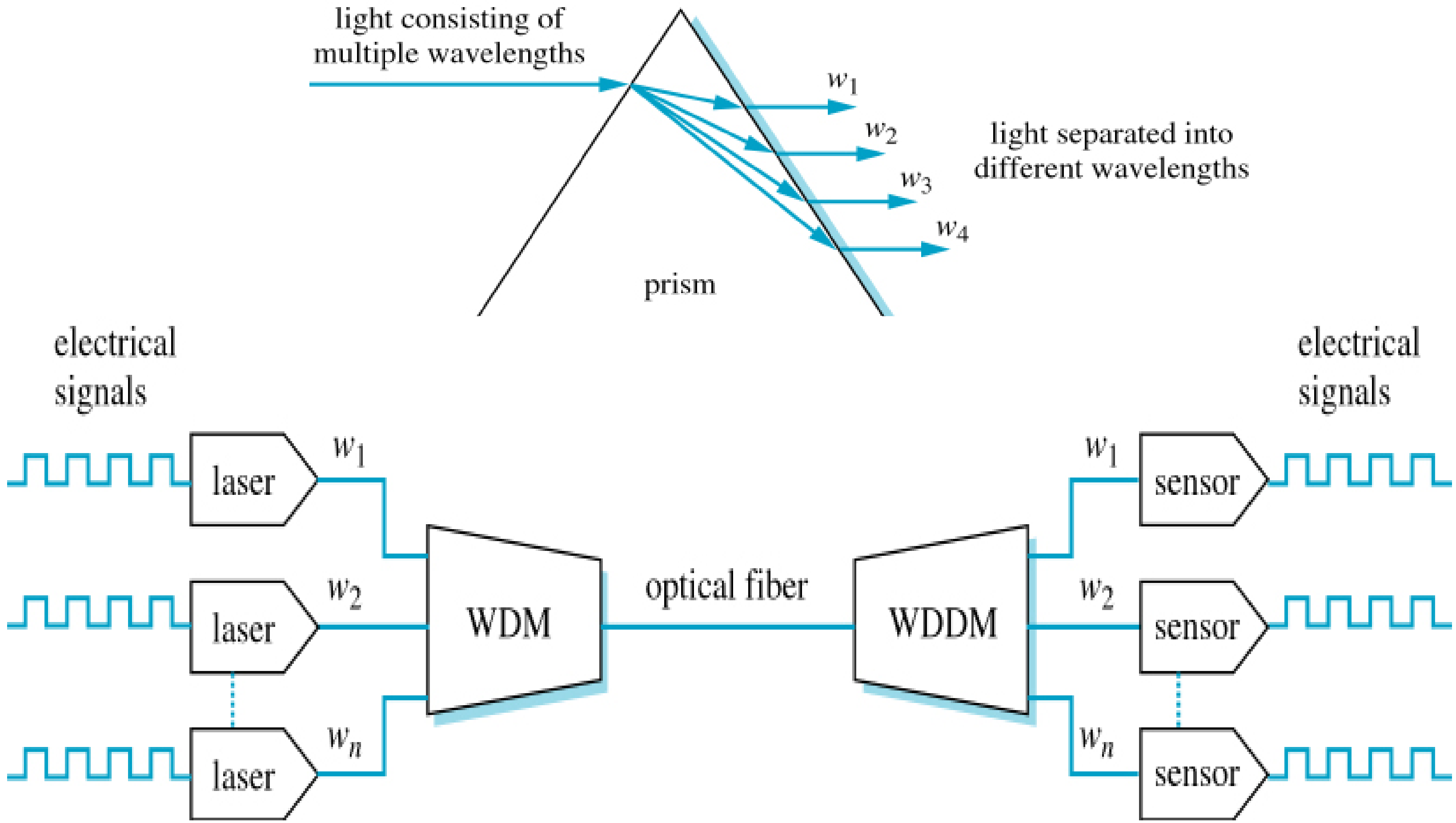
Multiplexing (4): radio-frequency signals

- Remember that each RF signal can carry a data stream as modulation



Multiplexing (5): optical (lightwave) signals

- Wave-Division (WDM):



Protocols

- A network protocol is a set of rules that defines how devices talk to each other.
  - It's an algorithm with two* separate participants.
  - They'd better obey the rules; otherwise communication will fail.
- 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'

*_or possibly more than two_

Flow Control (Shay 8.1)

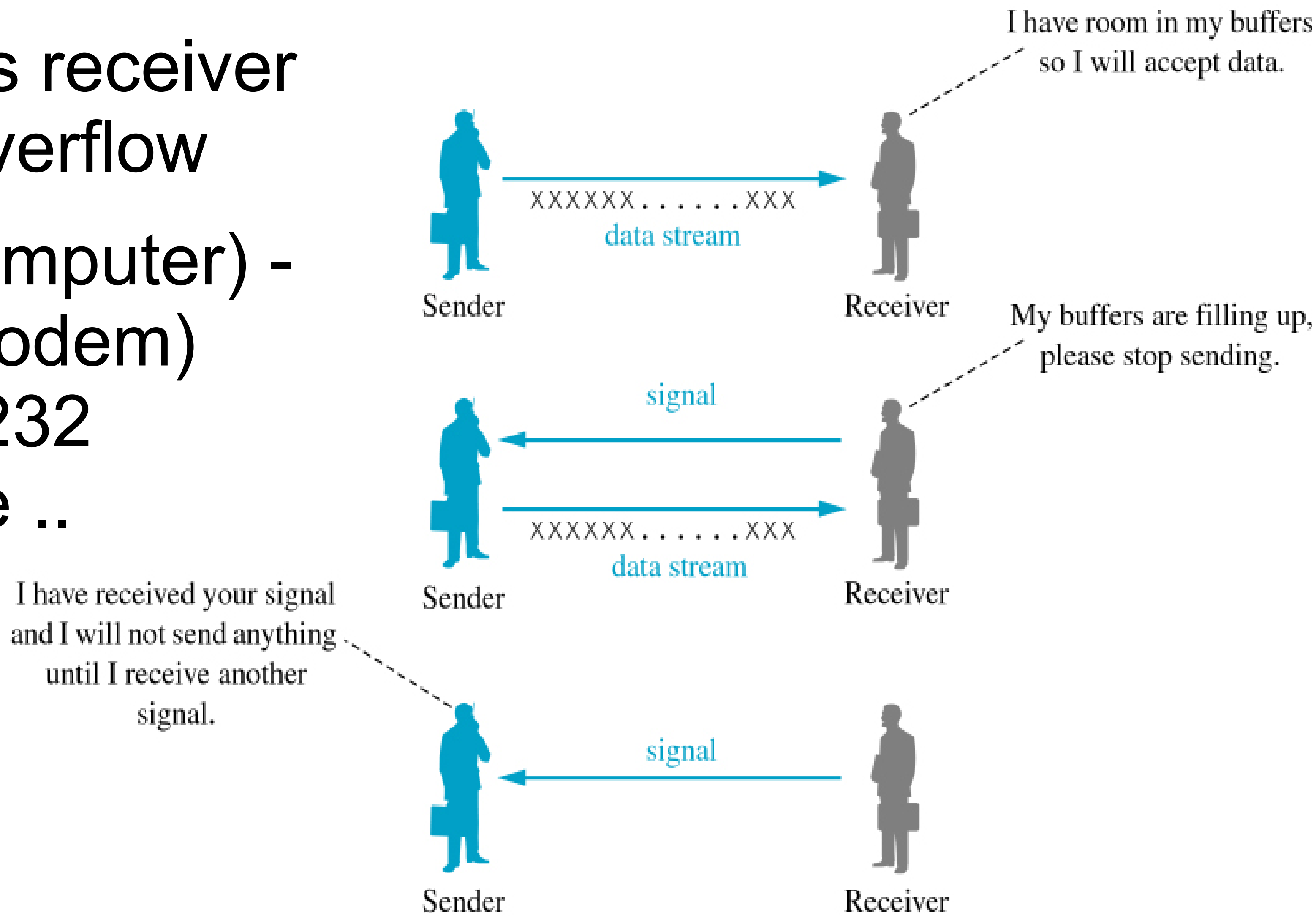
- Flow Control manages the flow of data so that the sender doesn't send too fast for the receiver
  - how can we send long messages, e.g. big files?
  - what happens when messages get lost, or are corrupted when they arrive?
  - what if the receiving *host* is busy, i.e. slow to accept incoming data?
  - 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* (or packets)
- Flow Control defines
  - “the way frames are sent, tracked and controlled”
  - may be simple or complex
  - Flow Control is a very basic kind of protocol

Signalling (Shay 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 (Shay 8.3)

- Idea is to break large sequences of characters into smaller *frames*
- Frames are sent from one *user* (higher protocol layer) to another
- Simplest approach: “Unrestricted” protocol
  - just 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 until get ACK
- Receiver:
  - receive frame, check for errors
  - if OK, send ACK; otherwise send NAK

- No way to handle lost frames (therefore no ACK and no NAK)

"ACK" = acknowledge; "NAK" = negative ACK

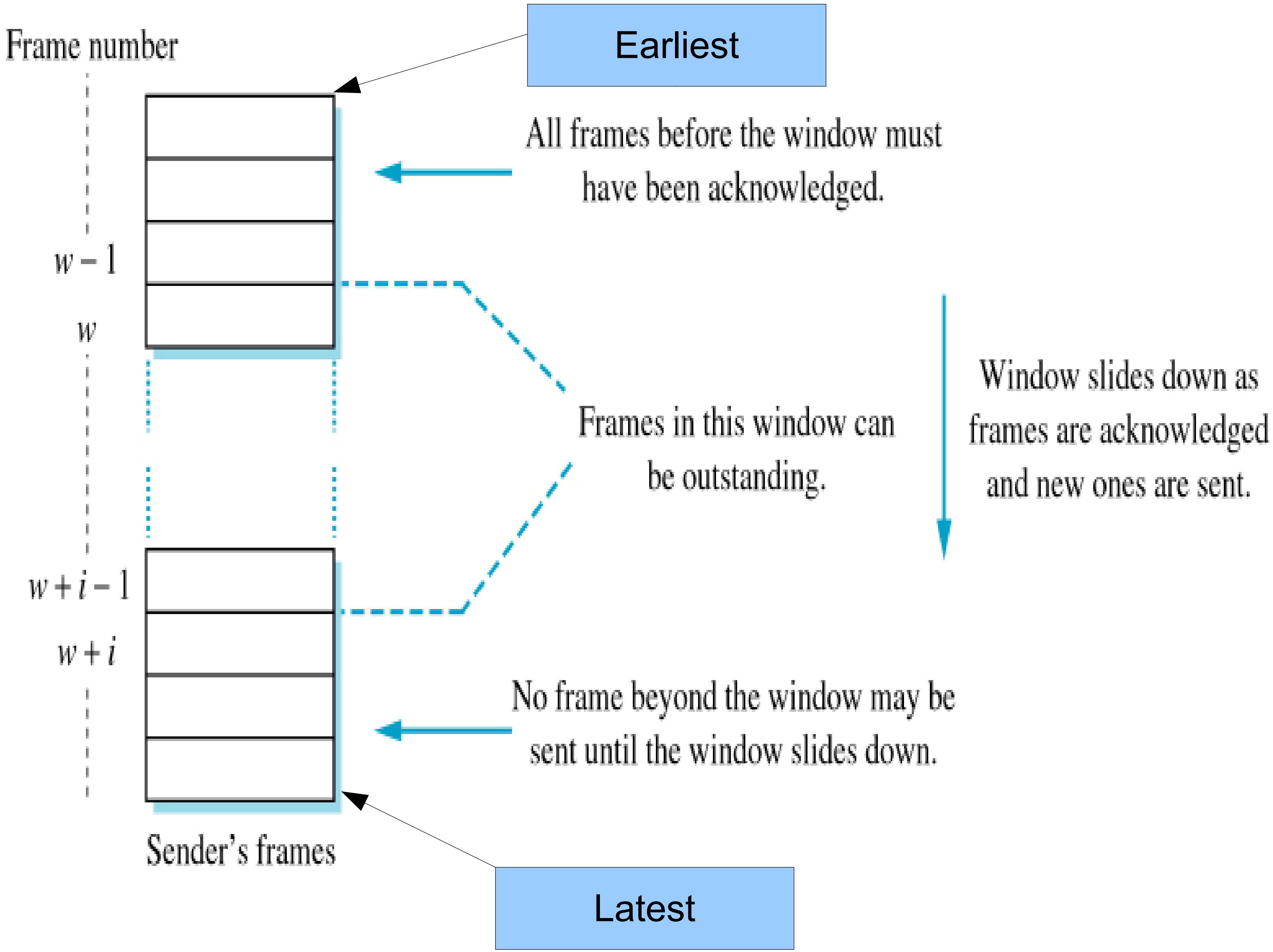
Protocol Efficiency: Effective data rate

- *Signal velocity* in wire or fibre is ~2/3 speed of light, i.e. 2×10^8 m/s
- Assume frame size is 1500 Bytes = 1500 x 8 bits = 12000 bits
- Assume transmission rate is 10 Mb/s = 10^7 b/s
- Then a frame takes 12000 /10^7 s = 1.2 ms to output on the wire
- Assume ACK is a 64 Byte frame → 0.0512 ms to output
- Auckland-Hamilton is about 120 km, so a signal takes *distance / velocity* = $(120 \times 10^3)/(2 \times 10^8)$ s = 0.6 ms to get there or back.
- Therefore, to send frame and receive ACK takes roughly 1.2 + 0.6 + 0.05 + 0.6 = 2.45 ms
- Effective bit rate is $12000/(2.45 \times 10^{-3})$ = 4.9 Mb/s
  - *49% efficient: half the time is wasted waiting for ACKs.*

Side note: a catch in the notation

- Convention:
  - Mb/s for megabits per second
  - MB/s for megabytes per second
- Often leads to confusion, especially with marketing people, journalists, and politicians.
- If there is any chance of confusion, write "megabits" or "megabytes" in full.
- In data communications, we normally discuss megabits. But when considering application throughput, megabytes are more useful.

Sliding Window (Shay 8.4)



Sliding Window

- 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 remember 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. Then it can slide down one place.

Go-back-n

- Shay develops a frame format for two-way communication

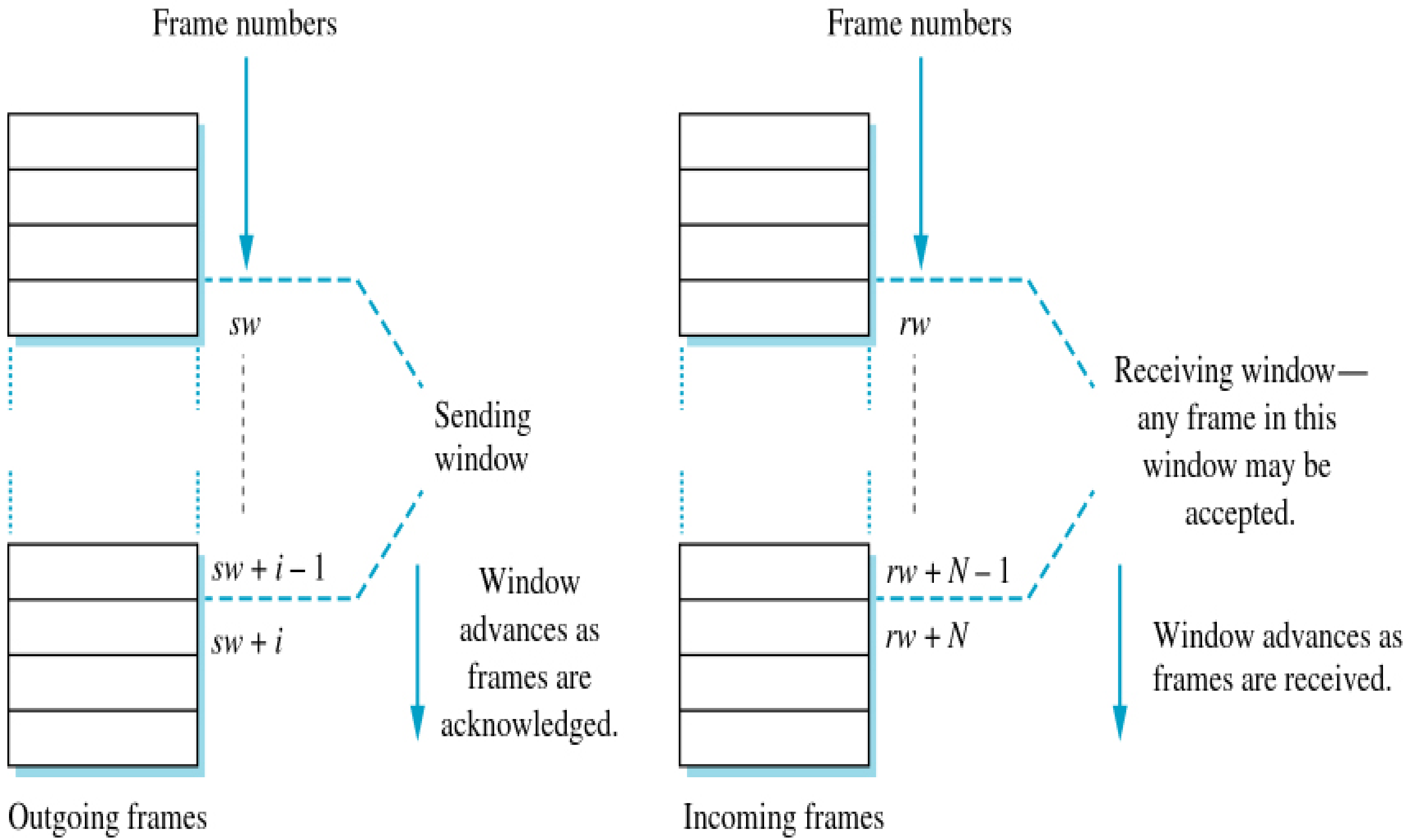
Source	Destination	Number	ACK	Type	...Data...	CRC

- 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
- When the receiver detects a missing ACK, it tells the transmitter to go back N packets and try again

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! (search Wikipedia)

Selective Repeat (Shay 8.5)



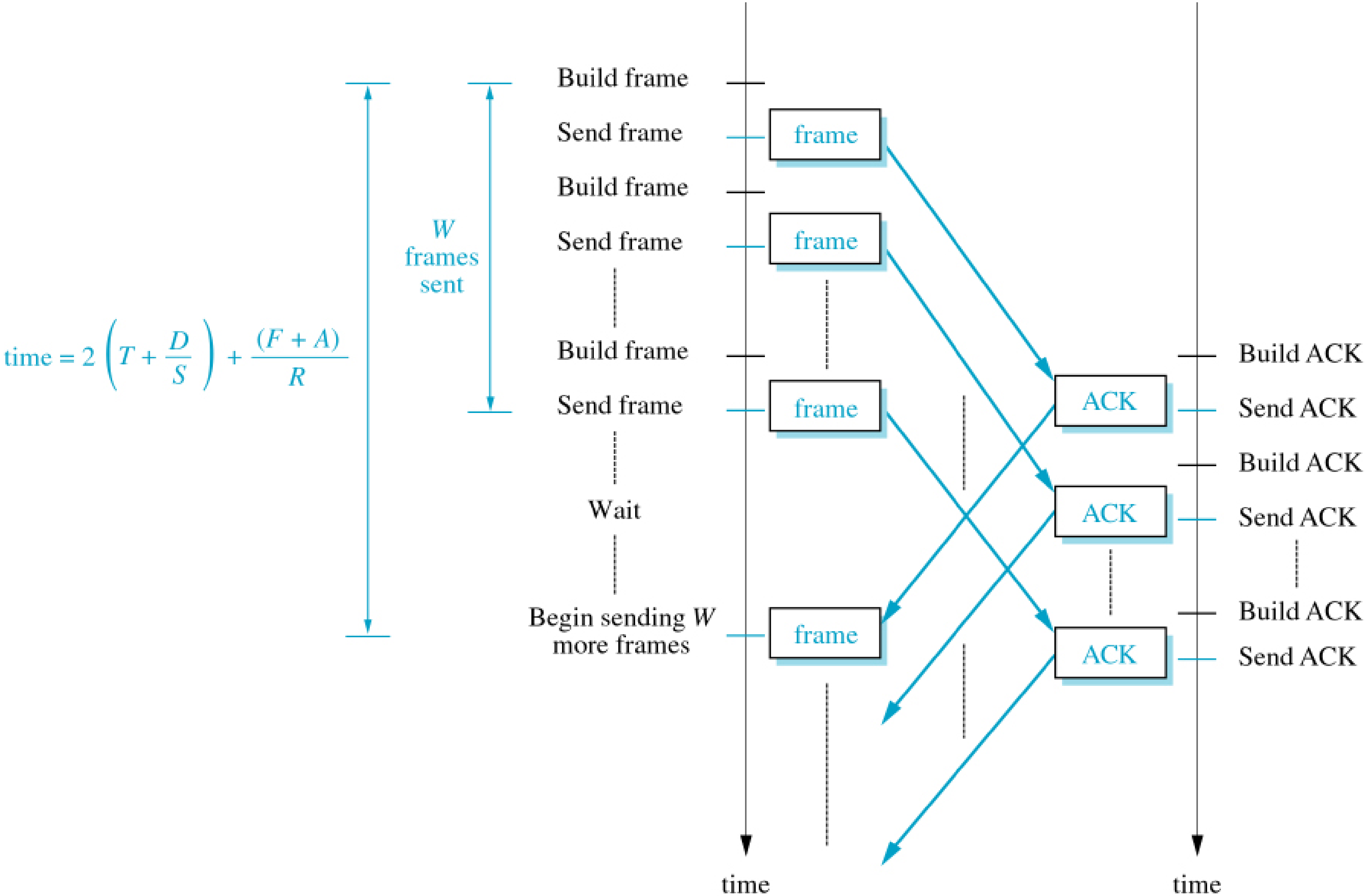
Selective Repeat (2)

- Any frame can be ACKed, specifying its 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
  - Note that Shay gives formulae if you prefer that
- *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 example (1)

- Sending many 1500B frames in 20-frame windows, Auckland-Hamilton on a 10 Mb/s link
  - Just like Stop-and-Wait: 1.2 ms to output frame, 0.6+0.6=1.2 ms round-trip time.
  - Any window >2 frames can run at full speed, 10 Mb/s, because ACK will be back after 1.2 + 1.2 = 2.4 ms*, just in time to release the 3rd packet in the queue.
  - *99% efficient* (you'd be brave to claim 100%)

*We have ignored the 0.05 ms to output the ACK frame, because it is small compared to the other numbers.

Numerical example (2)

- Same thing, but with 64B data frames and a window size of 20 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 0.0512+1.2+0.0512 = 1.3024 ms*
  - effective bit rate is (20 * 64 * 8)/1.3024 = 7.862 Mb/s
  - → *79% efficient*
  - note the effect of using a *small frame size!* We would need a larger window to get near to 100% efficiency.

*In this case we could not ignore the 0.05 ms to output the ACK frame, since it is the same as a data frame.

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 = 6000 bits
  = 750 B
- This is the maximum number of bits we can have 'on the wire'
- Need to have buffers at least double this so that transport protocol can keep the link busy
  – fill the wire once, and then again before first ACK returns
- Bigger frames sizes help to keep the link busy – less *protocol overhead*

**one-way delay, not round-trip time*

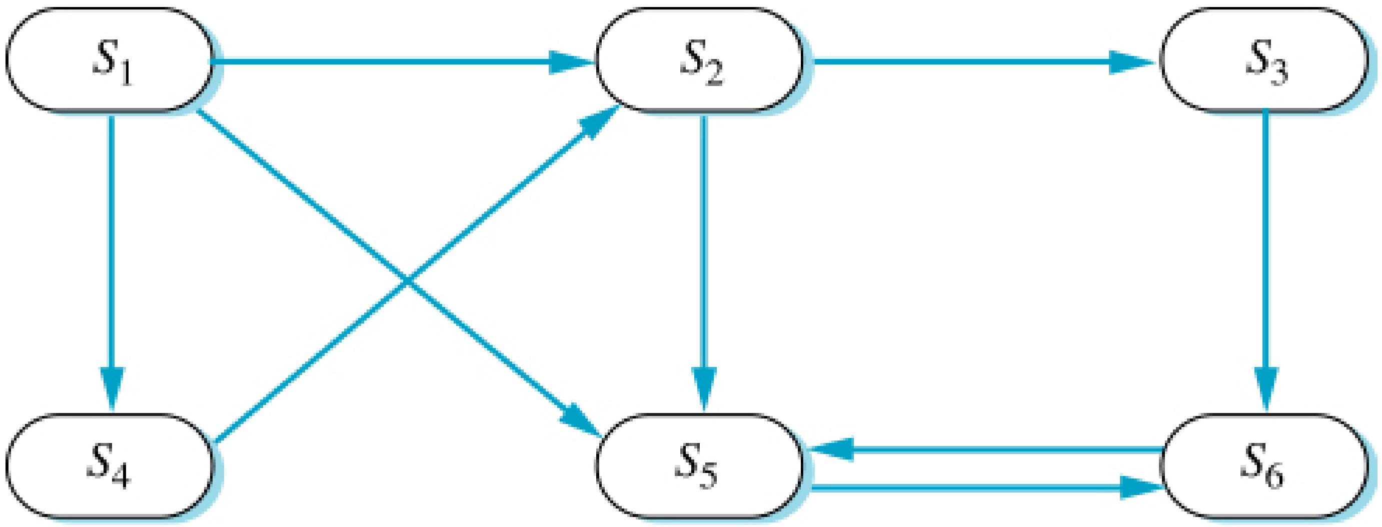
Take a deep breath... flow control methods

- X-ON/X-OFF
  - tell the other end to shut up
- Stop and wait
  - send one frame and wait for acknowledgment
- Sliding window
  - send multiple frames, and keep in step with acks
- Sliding window with go-back-N
  - simple error handling
- Sliding window with selective repeat
  - smarter error handling
- Later, we'll discuss TCP, the Internet's main flow control protocol.

Protocol Correctness (Shay 8.7)

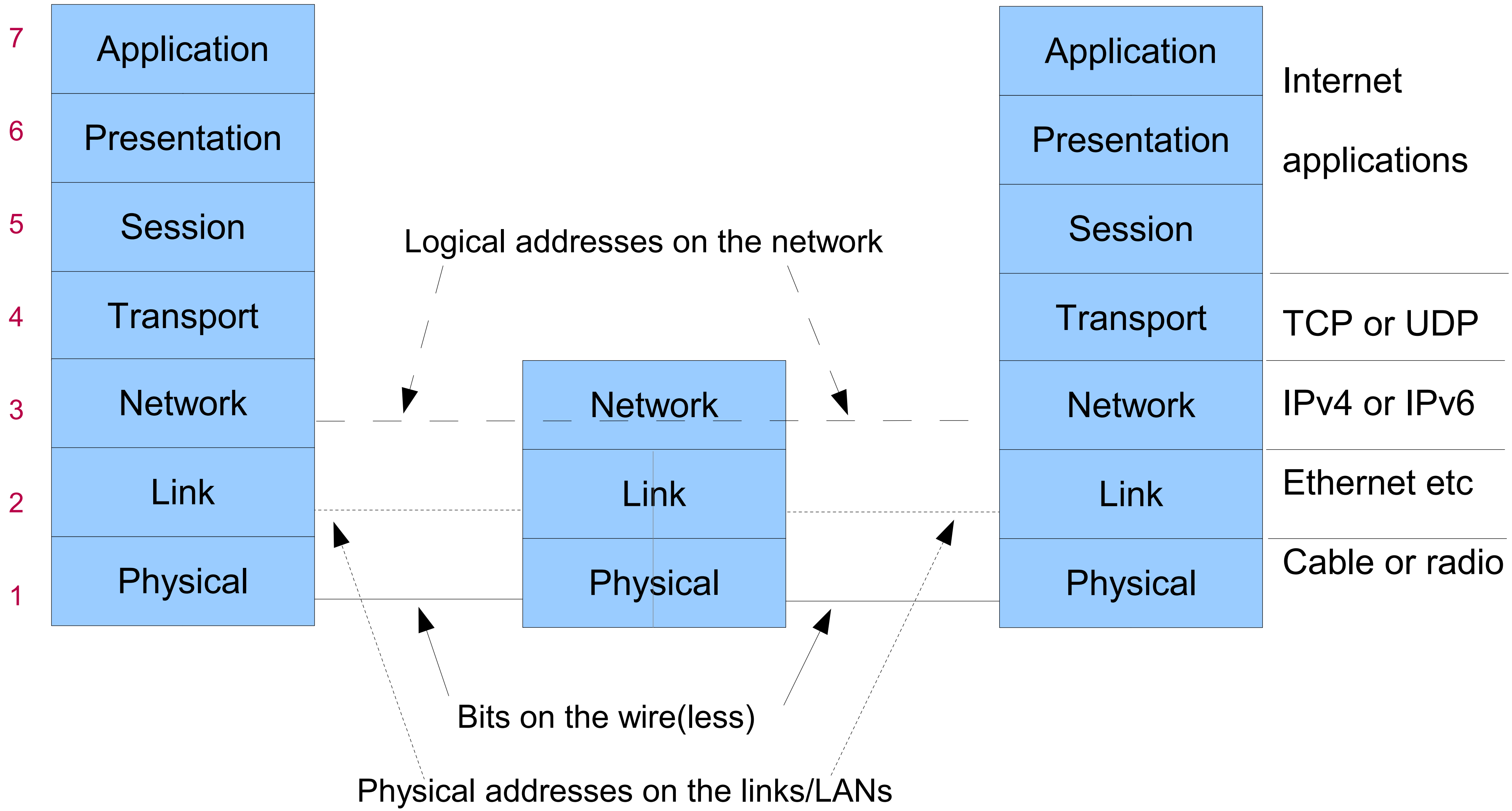
- Shay discusses two ways to describe systems:
  - Finite State Machines
  - Petri nets
- Finite State Machine models a system as being in one of a finite set of *states*
- State Transition Diagrams (STDs) are graphs, each vertex represents a state, and each edge a transition between states
- Petri nets 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!*

Reminder: the formal model



Protocol Layers

- 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*
- The lectures about coding mainly concerned layers 1 and 2.
- Flow control can be used at various layers, e.g. 2 or 4.
- For the next few lectures we will talk about Local Area Networks (LANs), which provide layers 1 and 2.

Introduction to LANs (Shay 9.1)

- LANs connect many hosts (devices) together
- Link may be copper (coax or UTP), fibre or wireless
- Topology may be
  - *bus:* hosts share the link by taking turns
  - *ring:* access is controlled by passing a token round
  - *star:* each host is wired back to a hub
- Ethernet
  - today's most common LAN
  - started with a bus topology
  - morphed into a star over the years.

LAN Layers

- Layer 1 is the Physical layer.
  - On this layer, we'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 packets between Ethernet hosts. We will see this later.

The story of the link layer (Shay 9.2)

- To properly understand modern link layer methods such as switched Gigabit Ethernet and WiFi, we need to understand the history of the link layer.
- To allow hardware products from different companies to work together, link layers have been standardised for many years.
  - International standards (mainly from the ITU)
  - US standards that have become dominant in the market (mainly from the IEEE 802 committee)
- We'll talk about HDLC, 802.2, Aloha, CSMA, Ethernet (802.3) and Wi-Fi (802.11)

High-Level Data Link Control (HDLC)

- Mainly intended for point-to-point links
- Designed before Local Area Networks existed in their modern form
  - a fore-runner of the IEEE 802.2 standard
- Still used when a simple serial connection is needed
- Frame-oriented link layer protocol
  - That is, a bit stream protocol with a frame structure

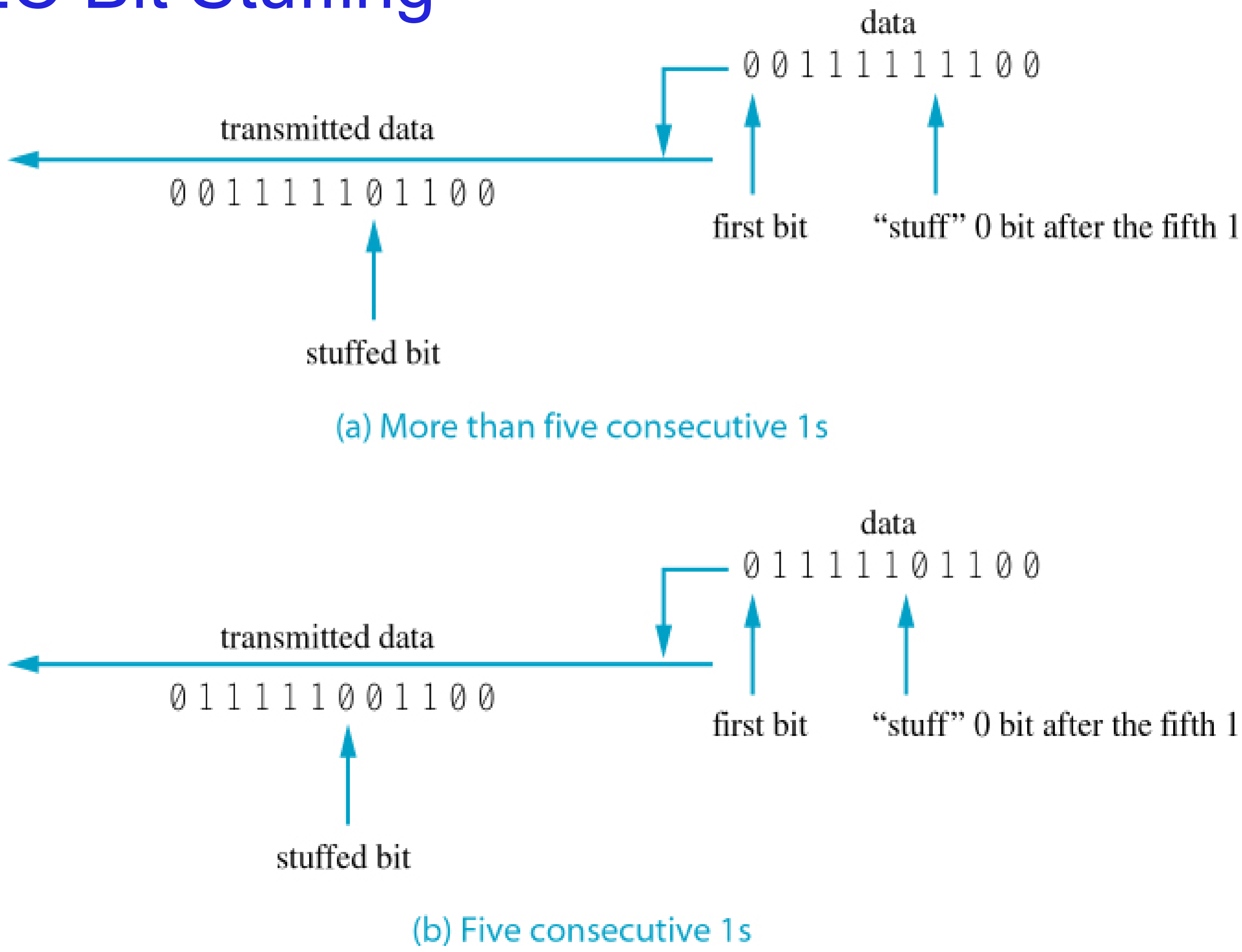
HDLC Frame Format

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

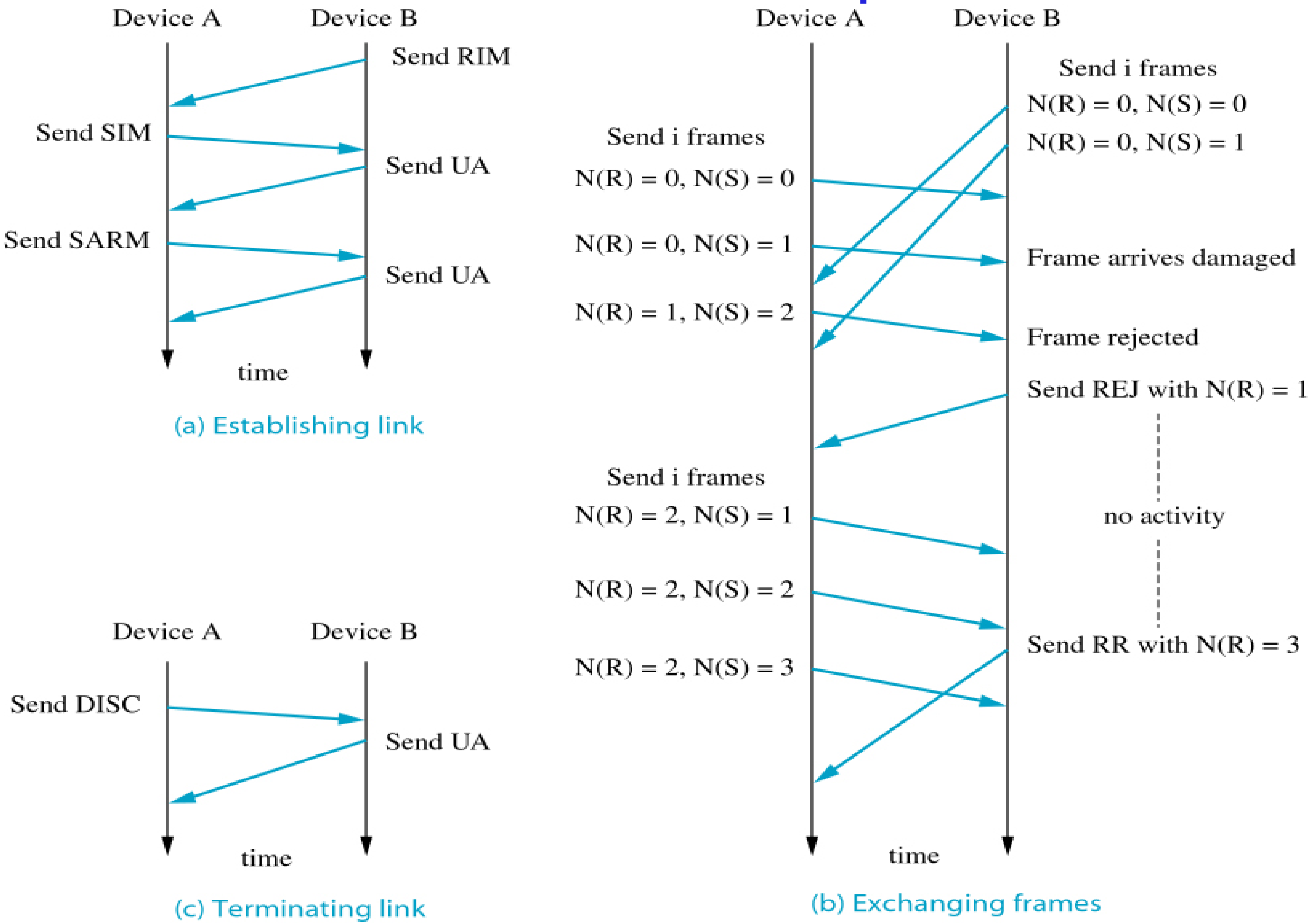
number of bits:	8	8 or 16	8 or 16	variable	16 or 32	8
	Flag	Address	Control	------- Data -------	FCS	Flag

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

HDLC Bit Stuffing

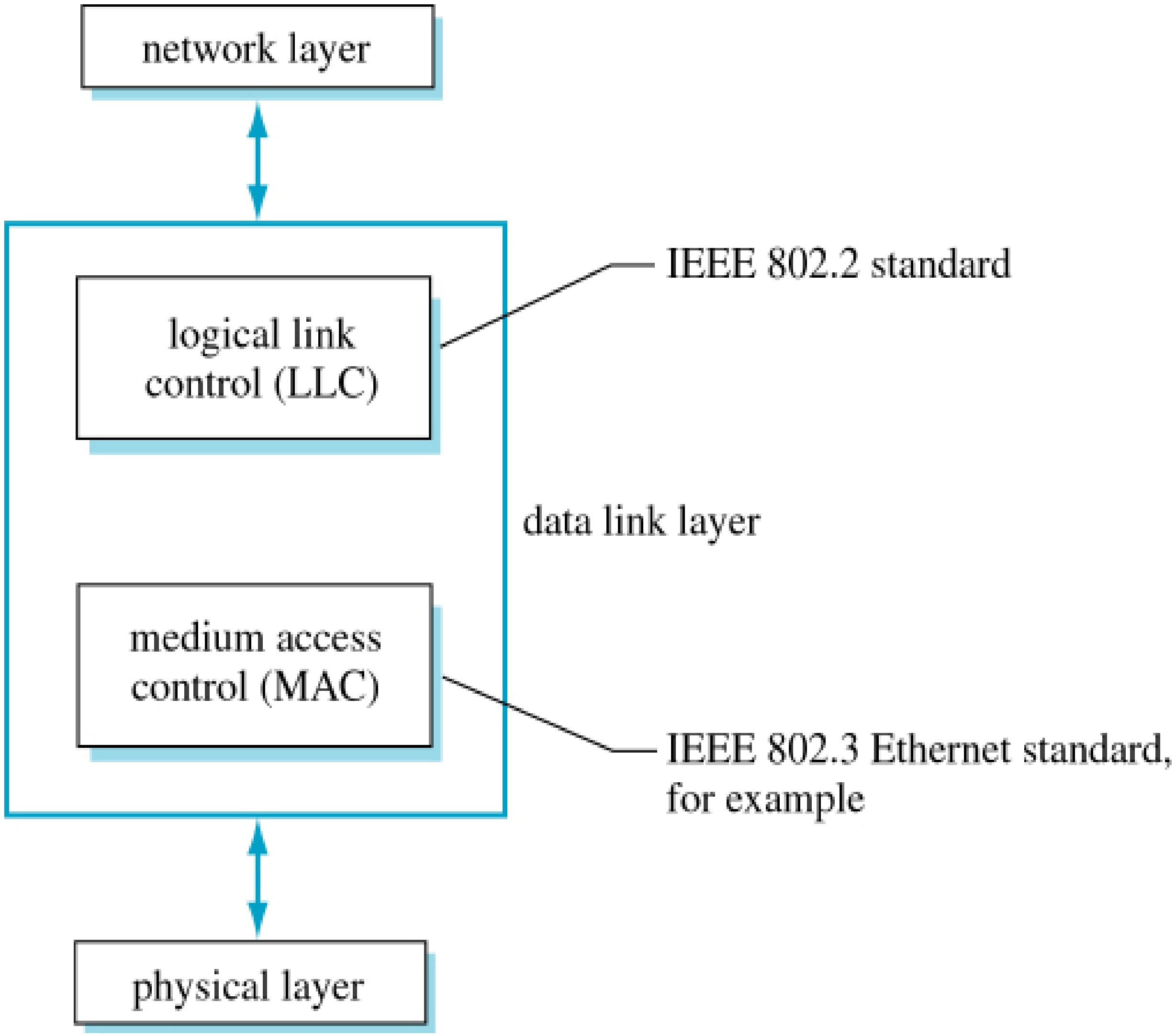


HDLC communication example



IEEE 802 Data Link Control (Shay 9.2)

- Link layer is divided in two: LLC and MAC sublayers.



802.2 Logical Link Control (LLC)

- Mainly intended for Local Area Networks (but designed in the abstract)
- Frame-oriented link layer protocol
- As we'll see, a bit complicated

802.2 LLC Header Formats (if used)

General form of LLC header

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_{16} = IBM SNA
  - 06_{16} = IP
  - AA_{16} = SNAP (Subnetwork Attachment Point)
  - etc. (many values defined)

802.2 LLC Header Formats (if used)

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

SNAP header (8 bytes)

AA AA 03 LLC	00 00 00 3 octet OUI	08 00 2 octet Ethertype	payload data

- OUI = Organisation Unique Identifier (zero for Ethernet types)
- Type field values are Ethernet type (Ethertype) values
  - 0800_{16} = IP, 0806_{16} = ARP, 6003_{16} = DECnet phase IV, ...

Isn't that a bit complicated?

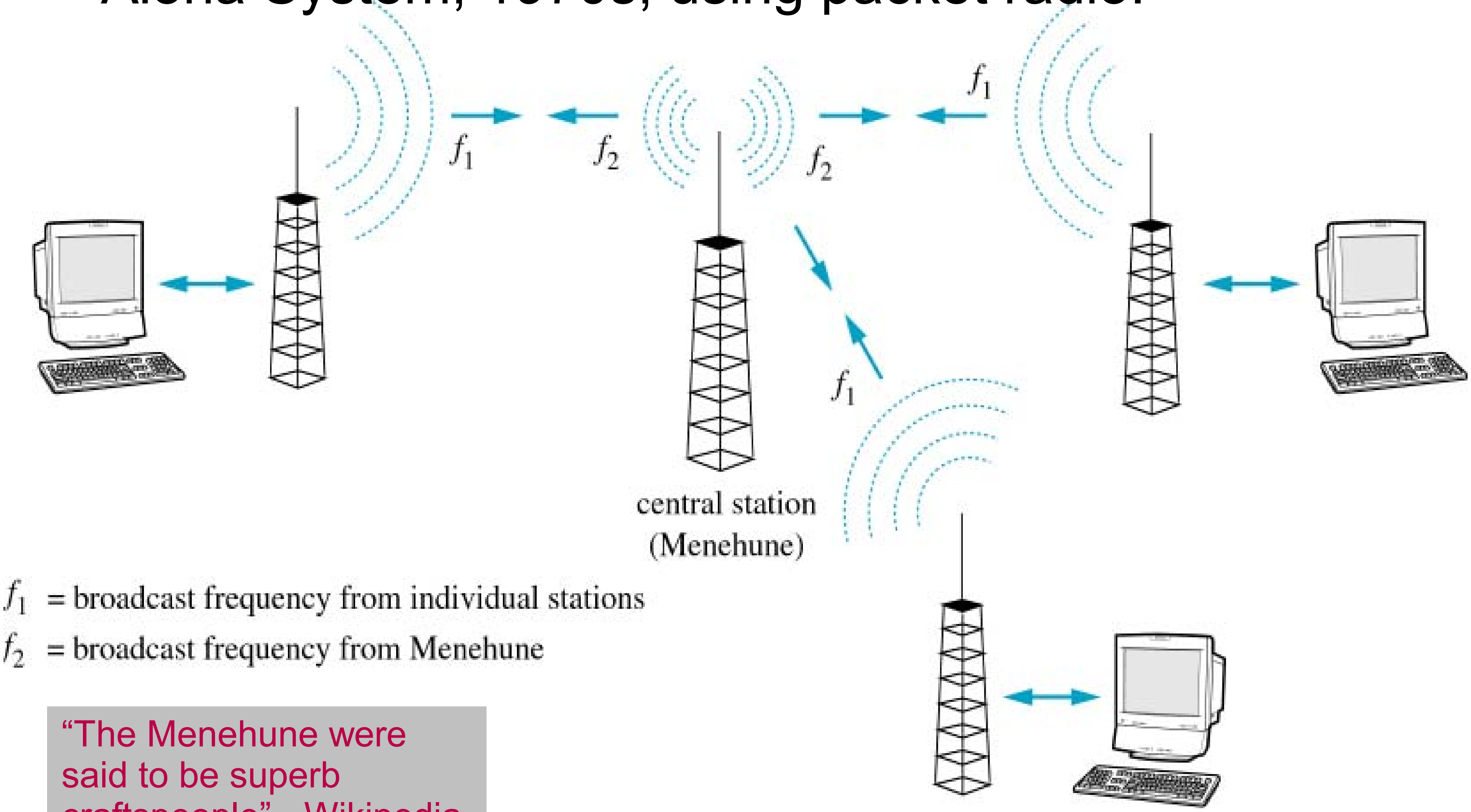
- Yes. LLC is quite complicated because it was designed by a committee, and it offers a lot of flexibility, enough to satisfy every member of the committee.
  - Flexibility in a protocol design usually makes it more complicated.
  - Did you notice two different ways to encapsulate IP?
  - Simple designs are usually more reliable, because there's less to go wrong.
  - Therefore, flexibility is not always a good thing.
- Personal opinion: LLC is a bad design.

Medium Access Control (MAC)

- We saw that this is part of Layer 2
- Why is it different from Flow Control?
  - Flow Control manages the flow of frames (or packets) so that the sender doesn't send too fast for the receiver
  - MAC manages physical access to the medium (cable, fibre, or wireless link) so that two senders don't talk at once
- Why is it different from multiplexing?
  - Multiplexing shares Layer 1 channel between multiple pairs of senders and receivers
  - MAC allows each sender full access to the channel, one at a time.
- Now we will discuss several MAC methods

Contention Protocols (Shay 4.7)

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



"The Menehune were said to be superb craftspeople" - Wikipedia

Aloha Protocol

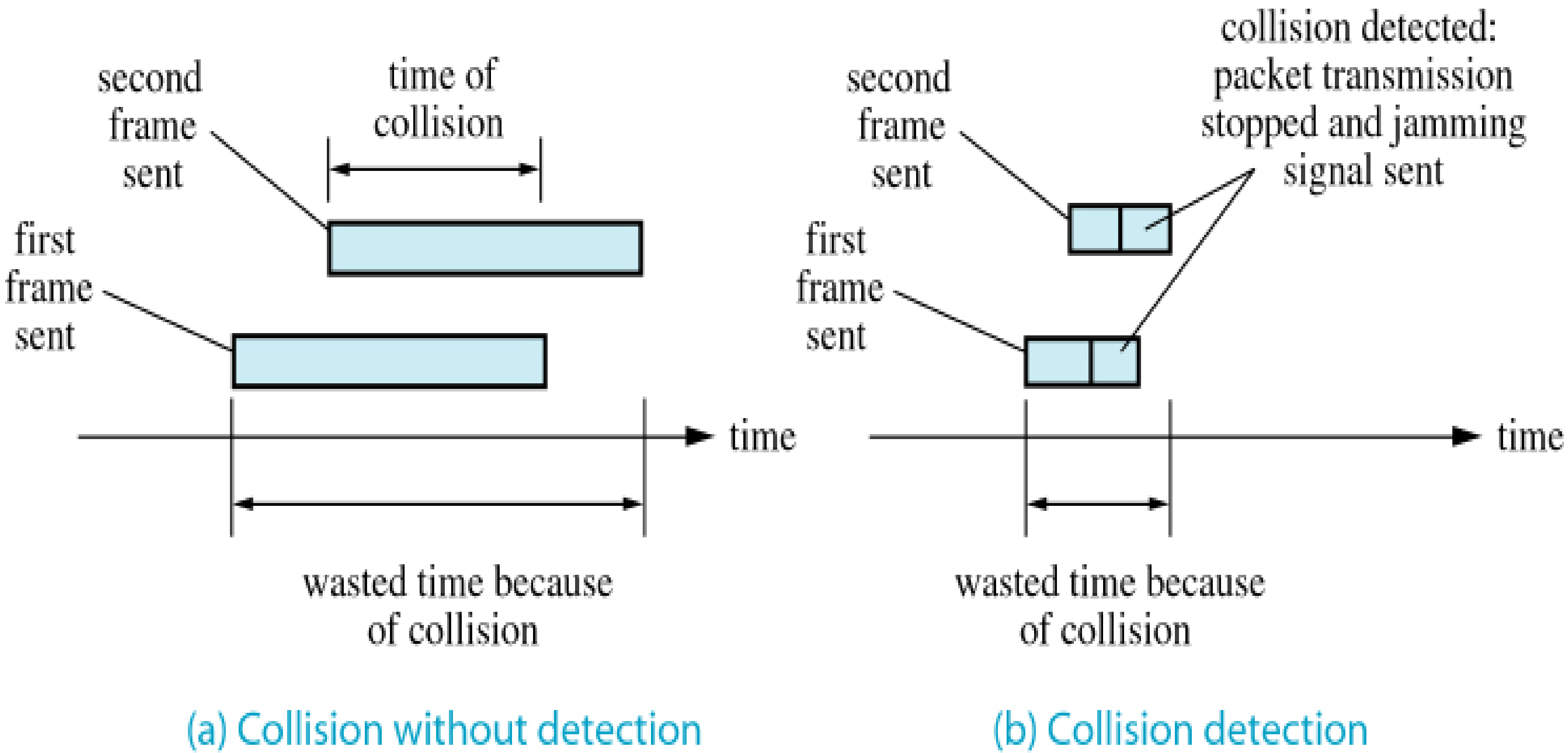
- Any host can broadcast a message to Menehune at any time
- If the message is received correctly, Menehune ACKs it (on a different frequency)
- If two host transmissions overlap (and interfere) the message is lost
- If a message is not ACKed the host assumes it was lost, waits a random time, then resends
- 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
  - Shay Fig. 4.44 compares various schemes

Collision Detection

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



How to exit a stop sign using CSMA/CD

Oops!

Oops *!#&*

OK! 3rd time lucky