I. INTRODUCTION

(a) Network Topologies

(i) point-to-point communication

→ each station (i.e., computer, telephone, etc.) directly connected to all other stations

(ii) switched networks

(1) *circuit switched* eg. telephone network

→ fixed path established for duration of call and path dedicated to that call only, regardless of whether data is being transferred

one switch network:

- in practice, many switches in telephone network

two switch network:

"trunk" = shared link

→ many channels or calls on one "wire", typically using Time Division Multiplexing (TDM)

TDM

→ each channel uses trunk for slot of time in round robin fashion

(2) *packet switched* eg. ATM networks

→ data divided into "packets" and packets sent individually

- i.e., paths used on a per packet basis and, hence, one link can carry packets from many different paths associated with different sessions (link is shared in time)

- in general, switched communication requires less hardware → many links replaced by few switches at the expense of increased complexity

(iii) broadcast networks eg. local area networks (LANs)

→ when one station transmits, all stations receive

LAN architecture:

- small hardware complexity (no switches, few links) and control complexity distributed to stations rather than network

(iv) hybrid combinations of (i), (ii), (iii)

eg. the Internet (actually packet switching on top of different types of networks)

(b) Packet Switching

general view of switch or node:

- output queues required because if a link is busy sending a packet, other packets destined for that link must wait in a queue

- input queues required because multiple packets entering a switch at once may require same switch resources (eg. links) and this may not be possible

- not all switches have both input and output queues but all packets will be saved into a queue (or buffer of memory) at least once in a switch

(c) Packet Format

- many different packets for different protocols

- typical format includes following fields:

(d) Flow Control

- required to ensure that a receiving node or server in a network does not have queues overflow because it is not processing packets as quickly as they arrive

(i) Stop-and-Wait Flow Control

- transmitter sends packets and waits for short acknowledgement packet (call it an "ACK") before sending next packet

- if receiver queues are filled, receiver can withhold ACK

- problem: packets get delayed in network due to queuing at switches, propagation delays of electrical signals along links, and processing delay at receiver
  $\rightarrow$ transmitted might have to wait long time for ACK before sending next packet
  $\therefore$ inefficient

Example:

(ii) Sliding Window Flow Control

- allows more than one packet sent at a time

- uses "sequence number" of k bits, SN $\in \{0, 1, \ldots , 2^k-1\}$, and increases SN by one modulo 2^k for every packet sent

- up to W, where W = window size, packets can be outstanding (i.e., sent but ACK not yet received)

- window slides along as packets acknowledged and new frames sent

See figures.

Example:

(e) Error Control

- packets travelling through networks can be

    (1) damaged:

        - packet arrives at destination but checksum fails
        eg. bit error in information field

    (2) lost:

        eg. packet dropped in network due to queue overflow or bit error in start flag causes packet to be undetected at destination

- packets can be corrected by having receiver request retransmission of detected damaged packets or transmitter times out waiting for ACK

    → "automatic repeat request" (ARQ)

(i) Stop-and-Wait ARQ

- if received packet damaged, error detection techniques may allow the receive to detect error, in which case the receiver sends negative acknowledgement (NACK) to request retransmission and transmitter resends when NACK received

- if receiver times out waiting for ACK due to lost packet or lost ACK, then packet resent

- problem: What if ACK lost resulting in receiver getting duplicate frame?
- solution: label ACKs → ACK0/ACK1 indicates receiver ready for packet 0/1
  → "alternating bit protocol"

See figure.

(ii) Go-Back-N ARQ

- based on sliding window flow control
- basic concept:
  → if packet *i* NACKed or timeout before ACK received, packet *i* and all subsequent packets retransmitted
  → can result in retransmission of properly received packets → inefficiency

See figure → must have $W \leq 2^k - 1 \Rightarrow$ usually, $W = 2^k - 1$ for efficiency.

(iii) Selective Reject ARQ

- similar to Go-Back-N except only damaged and lost packets retransmitted

- more efficient than Go-Back-N but more complex at receiver because packets can be received out of order

See figure $\rightarrow$ must have $W \leq 2^{k-1} \Rightarrow W = 2^{k-1}$ for efficiency

(iv) "Piggybacking" ACKs

- often ACKs/ NACKs are included as part of data packet
  $\rightarrow$ even if ACK has already been sent, when packet sent, ACK will be resent

Why is window size W picked to be $W \leq 2^k - 1$ for go-back-N and not $W = 2^k$, where k = number of bits in sequence number field?

Consider $W = 8$, $k = 3$:

$\rightarrow$ say transmitter sends packet 0 and gets ACK 1 in packet
$\rightarrow$ then transmitter sends 8 frames: 1, 2, 3, ... , 7, 0 and receives ACK 1 in packet
$\rightarrow$ Have all frames been acknowledged or have all frames been lost?
$\Rightarrow$ use $W = 7$ to avoid

Why should $W \leq 2^{k-1}$ for selective reject? Can you answer?

(f) Error Detection

How does the receiver detect damaged packets?

(i) Parity Check

- add bit to data block to ensure either:

  → even number of 1s (even parity)
  → odd number of 1s (odd parity)

- consider adding a parity bit to an *n*-bit block of data

  → parity easily implemented as XOR of bits

$$d_{n-1} \oplus d_{n-2} \oplus \ldots \oplus d_1 \oplus d_0 \oplus p = 0 \quad \text{(even parity)}$$
$$\text{or} \qquad = 1 \quad \text{(odd parity)}$$

Example:

- data is often stored/transmitted as 7-bit ASCII code + parity bit = 8-bit character

-parity bit can be used to detect an odd number of errors but even number of errors will be undetected

(ii) Cyclic Redundancy Check (CRC)

- adds r-bit frame check sequence (FCS) to n-bit data sequence with $r < n$

- let $T = (n+r)$-bit frame
  $M = n$-bit message (i.e., first n bits of T)
  $F = r$-bit FCS (i.e., last r bits of T)
  P = special $r+1$ bit pattern used to generate FCS

- consider arithmetic based on bit-wise modulo-2 addition

  → uses bit-wise XOR (i.e., no carry or borrow) for addition and subtraction

- FCS chosen so that T/P has no remainder

- let $T = 2^r M \oplus F$

  M shifted left by r bits

How to choose F so that T/P has no remainder?

Let $2^r M/P = Q \oplus R/P$
where R = remainder and must be r bits

and set $F = R$

Is T now divisible by P?

$T/P = (2^r M \oplus R) \ / \ P$
$= (Q \oplus R/\text{P}) \oplus R/P = Q \ \ \therefore$ yes, divisible by P

- let Γ represent the received vector such that $\Gamma = T \oplus E$
where E = error vector with ones representing positions of bit errors

eg.

- error detection process:

(1) at transmitter, divide $2^r M$ by P and use remainder as FCS
(2) at receiver, divide Γ by P and
$\rightarrow$ if remainder of $\Gamma/P = 0$ then assume no error
$\rightarrow$ if remainder of $\Gamma/P \neq 0$ then assume error

Example:

- often binary strings represented as polynomials

  eg. 1 1 0 0 1 $\rightarrow x^4 + x^3 + 1$

  and polynomials with special properties define a CRC scheme

- standard FCS exist such as "CRC-32" and "CRC-16"

  CRC-16: $P(x) = x^{16} + x^{15} + x^2 + 1$

- in general, 1st and last terms of $P(x)$ must be non-zero and for a properly selected $P(x)$ it can be shown that the following errors are detectable:

  (1) all single bit error patterns
  (2) all double bit error patterns
  (3) any odd number of errors, as long as $P(x)$ has a factor of $x + 1$
  (4) all burst errors with length less than FCS length
  (5) most larger burst errors

- transmitter and receiver can use shift register to perform division by $P(x)$

- let input = 10110010011011

step	c_5	c_4	c_3	c_2	c_1	c_0	c_5+c_4	c_5+c_3	c_5+c_2	$c_5+input$	input
							c_5'	c_4'	c_3'	c_0'	
0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	1	1
3	0	0	0	1	0	1	0	0	1	1	1
4	0	0	1	0	1	1	0	1	0	0	0
5	0	1	0	1	1	0	1	0	1	0	0
6	1	0	1	1	0	0	1	0	0	0	1
7	1	0	0	0	0	0	1	1	1	1	0
8	1	1	1	0	0	1	0	0	1	1	0
9	0	0	1	0	1	1	0	1	0	1	1
10	0	1	0	1	1	1	1	0	1	1	1
11	1	0	1	1	1	1	1	0	0	1	0
12	1	0	0	1	1	1	1	1	0	0	1
13	1	1	0	1	1	0	0	1	0	0	1
14	0	1	0	1	0	0	1	0	1	0	0
15	1	0	1	0	0	0	1	0	1	1	0
16	1	0	1	0	0	1	1	0	1	1	0
17	1	0	1	0	1	1	1	0	1	1	0
18	1	0	1	1	1	1	1	0	0	1	0
19	1	0	0	1	1	1	1	1	0	1	0
20	1	1	0	1	1	1	← FCS				

(g) Error Detection Calculations

- typical channel model:

→ bit errors occur randomly and independently and occur with probability of P_e

→ P_e is called "bit error rate"

(i) Parity Check

Consider an *n*-bit character which has a parity bit added to it.

Some things can be calculated:

- probability of no errors in a character:

- probability of one bit error in a character:

- probability of k bit errors in a character:

- probability that error is detected:

- probability that a character has undetected errors:

Note that $P(1) > P(2) > \ldots$ so parity makes sense, since single bit errors most likely and these will be detected.

Now consider a "packet" constructed of m n-bit characters where each character has a parity bit added to it.

- probability of no errors in packet:

- probability packet has errors but that no errors are detected:

- probability that packet has a detected error:

(ii) CRC

Consider packet format:

and assume CRC capable of detecting 1, 2, or 3 bit errors
(in practice, CRCs detect more error patterns that this!)

- probability of no errors:

- probability of missed packet:

- probability that received packet must be retransmitted:

- probability that errors are undetected in packet:

- expected number of retransmissions
  assuming ACKs, NACKs not corrupted and assume $P_U << P_R$ and $P_M << P_R$

(h) Communication Architectures

- typical communication protocols are layered to

    (1) logically partition functionality
    (2) encourage development of standards

- example 3 layer model for file transfer:

Application layer: - file transfer commands, file data

Transport layer: - ensures both ends are active and ready to communicate
- ensures reliable delivery of data

header: - destination application address
- sequence number
- CRC

Network layer: - interfaces to network for connection setup, etc.

header: - destination computer address
- network facilities request (eg. quality of service, priority)

Open Systems Interface (OSI) Model:
$\rightarrow$ 7 layer framework for communication protocols

See figures.