Error Detection

- Two types
  - Error Detection Codes (e.g. CRC, Parity, Checksums)
  - Error Correction Codes (e.g. Hamming, Reed Solomon)

- Basic Idea
  - Add redundant information to determine if errors have been introduced
    - Why redundant?
Error Detection Codes

- Naïve scheme
  - Send a duplicate copy of the message

- Problems
  - Takes up too much space
  - Poor performance.
    - Can’t even detect 2 bit errors

- Cyclic Redundancy Codes are common
Two Dimensional Parity

- Each byte is protected by a parity bit
- The entire frame is protected by a parity byte
Internet Checksum Algorithm

- Used by IP and TCP
- Algorithm treats data as 16 bit unsigned quantities
  - Add the data in the frame using 1’s complement arithmetic
  - Take the one’s complement of the result
  - IP and TCP store the sum as a 32 bit unsigned integer
    - Note: The example in the book uses a 16 bit checksum
Cyclic Redundancy Check

- Treat the (n+1) bit message as a polynomial of degree n. The bits are the coefficients of the polynomial.
  - $1101 = 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$

- Calculating CRC
  - Sender and transmitter choose a divisor polynomial of degree k. e.g $x^3 + x^2 + 1$
  - Add k bits to the (n+1) bit message such that the n+1+k bit message is exactly divisible by the divisor

- Choice of divisor is very important.
  - It determines the kind of errors that the CRC can guard against.
CRC Computation

- Given:
  - Message: $M(x)$
  - Divisor: $C(x)$

- Multiply $M(x)$ by $x^k$, i.e. add $k$ zeroes to the end of the message. Call this $T(x)$
- Divide $T(x)$ by $C(x)$.
- Subtract the remainder from $T(x)$

- The result is the message including the CRC
CRC Computation

- \( C(x) = x^3 + x^2 + 1 \)
- \( M(x) = x^7 + x^4 + x^3 + x \)
- Subtraction: logical XOR operation
Reliable Transmission

- Why?
  - Frame corruption can be severe
  - CRCs are not enough. Recall CRCs don’t correct errors

- Two fundamental mechanisms
  - Acknowledgment
  - Timeout

- General idea is called ARQ (Automatic Repeat Request)
Stop and Wait Protocol

- Simple operation
  - Transmit a packet
  - Wait for an acknowledgement (ACK)
  - If no ACK arrives within a preset time interval, assume that the packet is lost and retransmit

- Repeat the procedure until all packets have been successfully transmitted
Stop and Wait: Possible Scenarios

(a) Sender Receiver
   Frame
   ACK
   Timeout

(b) Sender Receiver
   Frame
   Timeout

(c) Sender Receiver
   Frame
   ACK
   Frame
   ACK

(d) Sender Receiver
   Frame
   ACK
   Frame
   ACK
Performance problems?

- No more than one packet in flight.
  - That’s usually bad, here’s why

- Take a 10Mbps network with a 50ms round trip time

- Delay bandwidth = $10^7 \times 0.050 = 500$ Kbits

- In Stop and Wait, only frame can be in flight. The max frame size is 1500 bytes
  - Hence sending rate =
    - $1500 \times 8 \div 0.050 = 240$ Kbps
  - This is much less than the link capacity of 10 Mbps
Performance Problems

- Using the actual 10Mbps Ethernet RTT of 50us (roughly)

- Delay bandwidth = $10^7 \times 50\text{us} = 500 \text{ bits}$

- In Stop and Wait, only frame can be in flight. The max Ethernet frame size is 1500 bytes
  - Hence sending rate =
    - $1500 \times 8 \div 50\text{us} = 240 \text{ Mbps}$
  - This is much greater than the link capacity of 10 Mbps
    - What happened??
Performance Analysis

\[ \text{Efficiency} = \frac{\text{TRANS}}{\text{TRANS} + \text{ACK} + 2\text{PROP}} \]
Performance Analysis

\[
\text{Efficiency} = \frac{\text{TRANS}}{\text{TRANS} + \text{ACK} + 2\text{PROP}}
\]

- Putting in numbers for 10 Mbps ethernet
  - Packet size: 1518 bytes
  - ACK size: 64 bytes
  - PROP: 51.2us

- Efficiency = 92.22%
  - More believable!

- Moral: If frame size exceeds delay bandwidth product, efficiency computation should be used.
Significance of Delay Bandwidth

- Delay bandwidth represents the amount of data that has left the transmitter and is still on the cable.

- Think of the cable as a pipe. This keeps the pipe full.

- Delay bandwidth also represents the upper bound on stability.

- More sophisticated ARQ algorithms try to match their sending rate to the dynamic delay bandwidth product.
  - Why is delay bandwidth dynamic?
Sliding Window Protocols

- Keep the pipe full
- Send N packets before expecting the first ACK
Sliding Window Protocol

- Send out N frames, each with a linearly increasing sequence number
- Sender uses 3 variables
  - Send window size: Upper bound on unACKed frames
  - LAR: Last ACK received
  - LFS: Last frame sent
- Invariant: LFS – LAR ≤ SWS

\[ \text{(Diagram showing LAR and LFS)} \]
Sliding Window Protocol

- Receiver maintains 3 variables
  - Receive window size: Upper bound on out of order frames received
  - LAF: Largest acceptable frame
  - LFR: Last frame (in sequence) received
  - Invariant: LAF – LFR ≤ RWS

\[
\text{LFR} \quad \text{LAF} \quad \leq \text{RWS}
\]
Operation

- Sender sends “send window size” (number of) frames
- Receiver ACKs last in sequence frame received.
- Error conditions:
  - Timeout
  - Receiver receives out of receive window frame
    - ACK it. Throw away the frame
  - Sender receives out of “send window” ack
    - Old delayed ACK. Throw it away
Performance Improvements

- Negative ACK
  - Receiver NACK’s frames that were not received
    - Additional complexity.
      - Loss of NACKs, receiver timeout mechanism needed

- Selective ACK:
  - Receiver ACKs specific frames. ACKs are not inclusive
  - Sender can use this to detect out of order arrival and retransmit
- Increases linearly with N until it reaches 100%
Efficiency

- E.g.:
  - Packet size = 2000 bits, ACK = 80 bits
  - Bandwidth = 155 Mbps ATM over fibre
  - Cable length: 30 km

\[
Efficiency = \min \left( \frac{N \times 2000}{2000 + 80 + 31000}, 1 \right)
\]

N = 1; Eff = 5.95%; N = 16; Eff = 96.73%
Window Size Settings

- Common modes
  - RWS = 1, SWS = N
    - Receiver does not buffer any out of order frames
  - RWS = SWS = N
    - Receiver can buffer as many frames as the sender sends.
  - RWS > SWS?
Finite Sequence Numbers

- Sequence numbers have a finite length. They increment by modulo arithmetic.

- Cases:
  - RWS = 1
  - MaxSeqNum >= SWS + 1
  - Why + 1
    - Receiver receives SWS, sends ACK
    - ACK is lost, receiver expects wrong frame
Finite Sequence Numbers

- **RWS = SWS**
  - MaxSeq Number $\geq 2 \times SWS$
  - Why?
    - Hint: receiver ACK is lost

- **RWS < SWS**
  - MaxSeq Number $\geq \max(SWS + 1, 2 \times RWS)$
  - How?