EE 122: Error detection and reliable transmission

Ion Stoica
September 16, 2002

High Level View

- Goal: transmit correct information
- Problem: bits can get corrupted
  - Electrical interference, thermal noise
- Solution
  - Detect errors
  - Recover from errors
    - Correct errors
    - Retransmission

Overview

- Error detection
- Reliable transmission

Error Detection

- Problem: detect bit errors in packets (frames)
- Solution: add redundancy bits to each packet
- Goals:
  - Reduce overhead, i.e., reduce the number of redundancy bits
  - Increase the number and the type of bit error patterns that can be detected
- Examples:
  - Two-dimensional parity
  - Checksum
  - Cyclic Redundancy Check (CRC)
Two-dimensional Parity

- Add one extra bit to a 7-bit code such that the number of 1’s in the resulting 8 bits is even (for even parity, and odd for odd parity).
- Add a parity byte for the packet.
- Example: five 7-bit character packet, even parity

<table>
<thead>
<tr>
<th>Packet</th>
<th>Even Parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>010100</td>
<td>1</td>
</tr>
<tr>
<td>101101</td>
<td>2</td>
</tr>
<tr>
<td>100110</td>
<td>3</td>
</tr>
<tr>
<td>111010</td>
<td>4</td>
</tr>
<tr>
<td>1000111</td>
<td>5</td>
</tr>
</tbody>
</table>

How Many Errors Can you Detect?

- All 1-bit errors
- Example:

<table>
<thead>
<tr>
<th>Packet</th>
<th>Error Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>011010</td>
<td></td>
</tr>
<tr>
<td>101010</td>
<td></td>
</tr>
<tr>
<td>100011</td>
<td></td>
</tr>
<tr>
<td>111011</td>
<td></td>
</tr>
<tr>
<td>110111</td>
<td></td>
</tr>
</tbody>
</table>

- All 2-bit errors
- Example:

<table>
<thead>
<tr>
<th>Packet</th>
<th>Error Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110100</td>
<td>1</td>
</tr>
<tr>
<td>1011010</td>
<td>2</td>
</tr>
<tr>
<td>1000111</td>
<td>3</td>
</tr>
<tr>
<td>1110111</td>
<td>4</td>
</tr>
<tr>
<td>1001111</td>
<td>5</td>
</tr>
</tbody>
</table>

- All 3-bit errors
- Example:

<table>
<thead>
<tr>
<th>Packet</th>
<th>Error Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110100</td>
<td>1</td>
</tr>
<tr>
<td>1011010</td>
<td>2</td>
</tr>
<tr>
<td>1000111</td>
<td>3</td>
</tr>
<tr>
<td>1110111</td>
<td>4</td>
</tr>
<tr>
<td>1001111</td>
<td>5</td>
</tr>
</tbody>
</table>
How Many Errors Can you Detect?

- Most 4-bit errors
- Example of 4-bit error that is not detected:

<table>
<thead>
<tr>
<th>Error Bits</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110100</td>
<td>✗</td>
</tr>
<tr>
<td>1011010</td>
<td>✗</td>
</tr>
<tr>
<td>0001111</td>
<td></td>
</tr>
<tr>
<td>1101100</td>
<td></td>
</tr>
<tr>
<td>1001011</td>
<td></td>
</tr>
<tr>
<td>1000110</td>
<td></td>
</tr>
</tbody>
</table>

How many errors can you correct?

Checksum

- Sender: add all words of a packet and append the result (checksum) to the packet
- Receiver: add all words of a packet and compare the result with the checksum
- Can detect all 1-bit errors
- Example: Internet checksum
  - Use 1’s complement addition

1’s Complement Revisited

- Negative number $-x$ is $x$ with all bits inverted
- When two numbers are added, the carry-on is added to the result
- Example: $-15 + 16$; assume 8-bit representation

  $15 = 00001111 \rightarrow -15 = 11110001$
  $16 = 00010000$
  $15 + 16 = 1$

  $00000001$

Cyclic Redundancy Check (CRC)

- Represent a (n+1)-bit message by an n-degree polynomial $M(x)$
  - E.g., $10101101 \rightarrow M(x) = x^7 + x^4 + x^2 + 1$
- Choose a divisor k-degree polynomial $C(x)$
- Compute remainder $R(x)$ of $M(x) \times x^k / C(x)$, and then compute $T(x) = M(x) \times x^k - R(x)$
  - $T(x)$ is divisible by $C(x)$
  - First n coefficients of $T(x)$ represent $M(x)$
- Sender:
  - Compute and send $T(x)$, i.e., the coefficients of $T(x)$
- Receiver:
  - Let $T'(x)$ be the (n+k)-degree polynomial generated from the received message
  - If $C(x)$ divides $T'(x)$ → no errors; otherwise errors
**Some Polynomial Arithmetic Modulo 2 Properties**

- If $C(x)$ divides $B(x)$, then $\deg(B(x)) \geq \deg(C(x))$.
- Subtracting $C(x)$ from $B(x)$ reduces to perform an XOR on each pair of matching coefficients of $C(x)$ and $B(x)$.
  
  E.g.:
  
  $B(x) = x^7 + x^5 + x^3 + x^2 + x^0 \rightarrow 10101101$
  
  $C(x) = x^3 + x^1 + x^0 \rightarrow 00010111$
  
  $B(x) - C(x) = x^7 + x^5 + x^2 + x^1 \rightarrow 10100110$

**Computing $T(x)$**

- Compute the reminder $R(x)$ of $M(x)*x^k \div C(x)$.
- Example: send packet 110111, assume $C(x) = 101$
  - $k = 2$, $M(x)*x^2 \rightarrow 11011100$
  - Compute $R(x)$

```
+--------+--------+--------+--------+--------+--------+
| 101    | 1101   | 1101   | 1101   | 1101   |
+--------+--------+--------+--------+--------+
| 100    | 101    |        |        |        |
| 100    | 101    |        |        |        |
| 100    | 101    |        |        |        |
+--------+--------+--------+--------+--------+
| 101    |        |        |        |        |
| 101    |        |        |        |        |
| 101    |        |        |        |        |
+--------+--------+--------+--------+--------+
```

- $T(x) = M(x)*x^k - R(x) \rightarrow 11011100 \ xor \ 1 = 11011101$

**CRC Properties**

- Detect all single-bit errors if coefficients of $x^1$ and $x^0$ of $C(x)$ are one.
- Detect all double-bit errors, if $C(x)$ has a factor with at least three terms.
- Detect all number of odd errors, if $C(x)$ contains factor $(x+1)$.
- Detect all burst of errors smaller than $k$ bits.

**Overview**

- Error detection
  - Reliable transmission
### Reliable Transmission

- **Problem:** obtain correct information once errors are detected  
- **Solutions:**  
  - Use error correction codes (can you give an example of error detection code that can also correct errors?)  
  - Use retransmission (we'll do this in details)  
- **Algorithmic challenges:**  
  - Achieve high link utilization, and low overhead

### Latency, Bandwidth, Round-Trip Time

- **Latency** = propagation + transmit + queue  
  - Propagation: time it takes the signal to propagate along the link  
  - Transmit: time it takes to transmit the packet = (packet size) / (link bandwidth)  
  - Queue: time for which the packet waits into the adapter at the sender before being transmitted  
- **Note:** next we'll assume short packets, i.e., transmit term can be neglected!  
- **Round-Trip Time** (RTT) = time it takes a packet to travel from sender to destination and back  
  - RTT = one-way latency from sender to receiver + one-way latency from receiver to sender

### Automatic Repeat Request (ARQ) Algorithms

- **Use two basic techniques:**  
  - Acknowledgements (ACKs)  
  - Timeouts  
- **Two examples:**  
  - Stop-and-go  
  - Sliding window

### Stop-and-Go

- **Receiver:** send an acknowledge (ACK) back to the sender upon receiving a packet (frame)  
- **Sender:** excepting first packet, send a packet only upon receiving the ACK for the previous packet

---

![Stop-and-Go Diagram](image-url)
What Can Go Wrong?

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame lost</td>
<td>Resend it on Timeout</td>
</tr>
<tr>
<td>ACK lost</td>
<td>Resend packet</td>
</tr>
<tr>
<td>ACK delayed</td>
<td>Resend packet</td>
</tr>
</tbody>
</table>

Need a mechanism to detect duplicate packet.

Stop-and-Go Disadvantage

- May lead to inefficient link utilization
- Example: assume
  - One-way propagation = 15 ms
  - Bandwidth = 100 Mbps
  - Packet size = 1000 bytes → transmit = \( 8 \times 1000 \) / 100 = 0.08 ms
  - Neglect queue delay → Latency = approx. 15 ms; RTT = 30 ms

Example:
- One-way propagation = 15 ms
- Bandwidth = 100 Mbps
- Packet size = 1000 bytes
- Transmit time = \( 8 \times 1000 \) / 100 = 0.08 ms
- Ignore queue delay

Stop-and-Go Disadvantage (cont’d)

- Send a message every 30 ms → Throughput = \( 8 \times 1000 \) / 0.03 = 26.66 Mbps
- Thus, the protocol uses less than 0.3% of the link capacity!

Solution

- Don’t wait for the ACK of the previous packet before sending the next one!
Sliding Window Protocol: Sender

- Each packet has a sequence number
  - Assume infinite sequence numbers for simplicity
- Sender maintains a window of sequence numbers
  - SWS (sender window size) – maximum number of packets that can be sent without receiving an ACK
  - LAR (last ACK received)
  - LFS (last frame sent)

<table>
<thead>
<tr>
<th>Acknowledged packets</th>
<th>Packets not acknowledged yet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sliding Window Protocol: Receiver

- Receiver maintains a window of sequence numbers
  - RWS (receiver window size) – maximum number of out-of-sequence packets that can received
  - LFR (last frame received) – last frame received in sequence
  - LAF (last acceptable frame)
  - LAF – LFR <= RWS

Example

- Assume SWS = 3

![Diagram showing the sliding window protocol with sequence numbers and an example scenario.]

Note: usually ACK contains the sequence number of the first packet in sequence expected by receiver.

Sliding Window Protocol: Receiver

- Let seqNum be the sequence number of arriving packet
- If (seqNum <= LFR) or (seqNum >= LAF)
  - Discard packet
- Else
  - Accept packet
  - ACK largest sequence number seqNumToAck, such that all packets with sequence numbers <= seqNumToAck were received

<table>
<thead>
<tr>
<th>Packets in sequence</th>
<th>Packets out-of-sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Properties of ARQ Protocols

- Reliability
- Increase link utilization (only for sliding window protocols)
- Flow control: a sender cannot send at a rate greater than the rate at which the receiver can consume the packets
- Packet order
  - in the case the Sliding Window Protocol the size of receiver window (RWS) specifies how many out-of-order packets can be stored at the receiver

Summary

- There are two steps required to transmit frames (packets) reliably
  - Detect when packets experience errors or are lost (we’ll talk more about packet loss in the context of TCP)
  - Two-dimensional parity
  - Checksum
  - Cyclic Redundancy Check (CRC)
  - Use packet retransmission to recover from errors
  - Stop-and-go
  - Sliding window protocol