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>0110100</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1011010</td>
<td>0</td>
</tr>
<tr>
<td>0010110</td>
<td>1</td>
</tr>
<tr>
<td>1110101</td>
<td>1</td>
</tr>
<tr>
<td>1001011</td>
<td>0</td>
</tr>
<tr>
<td>1000110</td>
<td>1</td>
</tr>
</tbody>
</table>
How Many Errors Can you Detect?

- All 1-bit errors
- Example:

```
0110100
0000110
1110101
1001011
1000110
```

odd number of 1’s
How Many Errors Can you Detect?

- All 2-bit errors
- Example:

```
0110100  1
1011010  0
0001111  1
1110101  1
1001011  0
1000110  1
```

error bits

odd number of 1’s on column
How Many Errors Can you Detect?

- All 3-bit errors
- Example:

```
<table>
<thead>
<tr>
<th></th>
<th>error bits</th>
<th>odd number of 1's on column</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110100</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1011010</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0001111</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1101011</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1001011</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1001110</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```
How Many Errors Can you Detect?

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

```
0110100 1
1011010 0
0000111 1
1100100 1
1001011 0
1000110 1
```

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

\[
\begin{align*}
15 &= 00001111 \\
-15 &= 11110000 \\
16 &= 00001000 \\
\text{Carry-on} &= 1 \\
15 + 16 &= 00000001 \\
\end{align*}
\]
Cyclic Redundancy Check (CRC)

Represent a (n+1)-bit message by an n-degree polynomial M(x)
- E.g., 10101101 \( \rightarrow \) \( M(x) = x^7 + x^5 + x^3 + x^2 + x^0 \)

Choose a divisor k-degree polynomial \( C(x) \)

Compute reminder \( R(x) \) of \( M(x) \cdot x^k / C(x) \), and then compute \( T(x) = M(x) \cdot x^k - R(x) \)
- \( T(x) \) 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 \( T'(x) \) \( \rightarrow \) no errors; otherwise errors
Some Polynomial Arithmetic Modulo 2 Properties

- If \( C(x) \) divides \( B(x) \), then degree\((B(x))\) \( \geq \) degree\((C(x))\)
- 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.:

\[
\begin{align*}
B(x) &= x^7 + x^5 + x^3 + x^2 + x^0 \rightarrow 10101111 \\
C(x) &= x^3 + x^1 + x^0 \rightarrow 00001011
\end{align*}
\]

\[
B(x) - C(x) = x^7 + x^5 + x^2 + x^1 \rightarrow 10100110
\]
Computing T(x)

- Compute the reminder R(x) of $M(x)x^k / C(x)$
- $T(x) = M(x)x^k - R(x)$
- Example: send packet 110111, assume $C(x) = 101$
  - $k = 2$, $M(x)x^k \rightarrow 11011100$
  - Compute $R(x)$
- $T(x) = M(x)x^k - R(x) \rightarrow 11011100 \text{ xor } 1 = 11011101$
CRC Properties

- Detect all single-bit errors if coefficients of $x^k$ 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 to 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
What Can Go Wrong?

Frame lost → resent it on Timeout

ACK lost → resent packet
Need a mechanisms to detect duplicate packet

ACK delayed → resent packet
Need a bit to differentiate between ACK for current and previous 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 $\rightarrow$ transmit = $(8 \times 1000)/10^8 = 0.08$ ms
  - Neglect queue delay $\rightarrow$ Latency = approx. 15 ms; RTT = 30 ms
Stop-and-Go Disadvantage (cont’d)

- Send a message every 30 ms → Throughput = (8*1000)/0.03 = 0.2666 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)

Acknowledged packets  Packets not acknowledged yet

LAR

LFS seq. numbers
Example

- Assume SWS = 3

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

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

Packets in sequence
Packets out-of-sequence

LFR  LAF  seq. numbers
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