Lecture 6: Reliable Transmission

CSE 123: Computer Networks
Alex C. Snoeren

HW 1 due NOW
Lecture 6 Overview

- Cyclic Remainder Check (CRC)
- Automatic Repeat Request (ARQ)
  - Acknowledgements (ACKs) and timeouts
- Stop-and-Wait
Checkums are easy to compute, but very fragile
- In particular, burst errors are frequently undetected
- We’d rather have a scheme that “smears” parity

Need to remain easy to implement in hardware
- So far just shift registers and an XOR gate

We’ll stick to Modulo-2 arithmetic
- Multiplication and division are XOR-based as well
- Let’s do some examples…
Modulo-2 Arithmetic

- **Multiplication**

  
  \[
  \begin{array}{c}
  1101 \\
  \underline{110} \\
  0000 \\
  11010 \\
  110100 \\
  \underline{\text{101110}} \\
  \end{array}
  \]

- **Division**

  
  \[
  1101 \overline{110} \]

  
  \[
  \begin{array}{c}
  101110 \\
  \underline{110} \\
  111 \\
  \underline{110} \\
  011 \\
  \underline{\text{000}} \\
  \underline{\text{110}} \\
  \end{array}
  \]
Cyclic Remainder Check

- Idea is to divide the incoming data, $D$, rather than add
  - The divisor is called the generator, $g$
- We can make a CRC resilient to $k$-bit burst errors
  - Need a generator of $k+1$ bits
- Divide $2^kD$ by $g$ to get remainder, $r$
  - Remainder is called frame check sequence
- Send $2^kD - r$ (i.e., $2^kD$ XOR $r$)
  - Note $2^kD$ is just $D$ shifted left $k$ bits
  - Remainder must be at most $k$ bits
- Receiver checks that $(2^kD-r)/g = 0$
CRC: Rooted in Polynomials

- We’re *actually* doing polynomial arithmetic
  - Each bit is actually a coefficient of corresponding term in a $k^{th}$-degree polynomial

  $1101$ is $(1 \times X^3) + (1 \times X^2) + (0 \times X^1) + (1 \times X^0)$

- Why do we care?
  - Can use the properties of finite fields to analyze effectiveness
  - Says any generator with two terms catches single bit errors
CRC Example Encoding

\[
\begin{align*}
    x^3 + x^2 + 1 &= 1101 \\
    x^7 + x^4 + x^3 + x &= 10011010
\end{align*}
\]

Generator
Message

k + 1 bit check sequence \(g\), equivalent to a degree-k polynomial

\[
\begin{align*}
    1101 & \quad 10011010000 \\
    1101 & \quad 1001 \\
    1101 & \quad 1000 \\
    1101 & \quad 1011 \\
    1101 & \quad 1100 \\
    1101 & \quad 1000 \\
    101 & \quad 1101 \\
\end{align*}
\]

Message plus \(k\) zeros (*\(2^k\))

Result:
Transmit message followed by remainder:

\[
10011010101
\]
CRC in Hardware

- Key observation is only subtract when MSB is one
  - Recall that subtraction is XOR
  - No explicit check for leading one by using as input to XOR

- Hardware cost very similar to checksum
  - We’re only interested in remainder at the end
  - Only need $k$ registers as remainder is only $k$ bits
CRC Example Decoding

\[ x^3 + x^2 + 1 = 1101 \]
\[ x^{10} + x^7 + x^6 + x^4 + x^2 + 1 = 10011010101 \]

\[ k + 1 \text{ bit check sequence } g, \text{ equivalent to a degree}-k \text{ polynomial} \]

Received message, no errors

Result:

CRC test is passed
CRC Example Failure

\[ x^3 + x^2 + 1 = 1101 \]
\[ x^{10} + x^7 + x^5 + x^4 + x^2 + 1 = 10010110101 \]

\[ \text{Generator} \]
\[ \text{Received Message} \]

\[ k + 1 \text{ bit check sequence } g, \text{ equivalent to a degree-}k \text{ polynomial} \]
\[ 1101 \]
\[ 10010110101 \]
\[ \text{Received message} \]
\[ 1000 \]
\[ 1101 \]
\[ \text{Two bit errors} \]
\[ 1011 \]
\[ 1101 \]
\[ 1101 \]
\[ 1101 \]
\[ 0101 \]

Remainder
\[ D \mod g \]

Result:
**CRC test failed**

CSE 123 – Lecture 6: Reliable Transmission
# Common Generators

<table>
<thead>
<tr>
<th>Generator</th>
<th>Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC-8</td>
<td>$x^8 + x^2 + x^1 + 1$</td>
</tr>
<tr>
<td>CRC-10</td>
<td>$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$</td>
</tr>
<tr>
<td>CRC-12</td>
<td>$x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$</td>
</tr>
<tr>
<td>CRC-16</td>
<td>$x^{16} + x^{15} + x^2 + 1$</td>
</tr>
<tr>
<td>CRC-CCITT</td>
<td>$x^{16} + x^{12} + x^5 + 1$</td>
</tr>
<tr>
<td>CRC-32</td>
<td>$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$</td>
</tr>
</tbody>
</table>
Error Handling Summary

- Add redundant bits to detect if frame has errors
  - A few bits can detect errors
  - Need more to correct errors

- Strength of code depends on Hamming Distance
  - Number of bitflips between codewords

- Checksums and CRCs are typical methods
  - Both cheap and easy to implement in hardware
  - CRC much more robust against burst errors
Picking up the Pieces

- Link layer is lossy
  - We deliberately threw away corrupt frames last lecture
  - Infrequent bit errors still lead to occasional frame errors
    » 10,000+ bits in each frame

- Things get even harrier if we consider multiple links
  - In a few lectures, we’ll start sending frames on long trips
  - Each intermediate stop might lose, corrupt, reorder, etc.
  - Regardless of cause, we’ll call loss events drops

- We want to provide reliable, in-order delivery
  - Can—and will—do this at multiple layers
Moving up the Stack

CSE 123 – Lecture 6: Reliable Transmission
Simple Idea: ARQ

- Receiver sends **acknowledgments (ACKs)**
  - Sender “times out” and retransmits if it doesn’t receive them
- Basic approach is generically referred to as **Automatic Repeat Request (ARQ)**

CSE 123 – Lecture 6: Reliable Transmission
Not So Fast…

- Loss can occur on ACK channel as well
  - Sender cannot distinguish data loss from ACK loss
  - Sender will retransmit the data frame
- ACK loss—or early timeout—results in duplication
  - The receiver thinks the retransmission is new data
Sequence Numbers

- Sequence numbers solve this problem
  - Receiver can simply ignore duplicate data
  - But must still send an ACK! (Why?)

- Simplest ARQ: **Stop-and-wait**
  - Only one outstanding frame at a time
For Next Time

- Read 2.6 in P&D