Lecture 4:
Reliable Transmission

CSE 123: Computer Networks
Alex C. Snoeren

HW 1 out TODAY; due 2/1
Lecture 4 Overview

- Finishing Error Detection
  - Cyclic Remainder Check (CRC)

- Handling errors
  - Automatic Repeat Request (ARQ)
  - Acknowledgements (ACKs) and timeouts
  - Stop-and-Wait
Error Detection – CRC

- View data bits, D, as a binary number
- Choose r+1 bit pattern (generator), G
- Goal: choose r CRC bits, R, such that
  - \(<D,R>\) exactly divisible by G (modulo 2)
  - Receiver knows G, divides \(<D,R>\) by G. If non-zero remainder: error detected!
  - Can detect all burst errors less than r+1 bits
- Widely used in practice (Ethernet, FDDI, ATM)

\[
D \times 2^r \text{ XOR } R
\]

\text{bit pattern}\n
\text{mathematical formula}
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

$$1101 = (1 \cdot X^3) + (1 \cdot X^2) + (0 \cdot X^1) + (1 \cdot X^0)$$

- 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

\[ x^3 + x^2 + 1 = 1101 \]
\[ x^7 + x^4 + x^3 + x = 10011010 \]

Generator

Message

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

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

Result:

Transmit message followed by remainder:

\[ 10011010101 \]
CRC Example Decoding

$x^3 + x^2 + 1$
$x^{10} + x^7 + x^6 + x^4 + x^2 + 1 = 10011010101$

Generator
Received Message

Result:
CRC test is passed

Remainder
$D \mod g$

$k + 1$ bit check sequence $g$, equivalent to a degree-$k$ polynomial

1101

10011010101

1101

1101

1101

1101

1101

0
CRC Example Failure

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

= 1101

Generator

= 10010110101

Received Message

\[ \text{Received message} \]

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

\[ 0101 \]

\[ 1101 \]

Two bit errors

\[ 10010110101 \]

\[ 1000 \]

\[ 1101 \]

\[ 1011 \]

\[ 1101 \]

\[ 1101 \]

0101

Result:

CRC test failed

CSE 123 – Lecture 4: 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>
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
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 throw away corrupt frames!
  - 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

http

TCP

IP

Ethernet

interface

application layer

transport layer

network layer

link layer

host

router

host

CSE 123 – Lecture 4: Reliable Transmission
Reliable Transmission

- The data networking version of the same problem
  - How do we reliably send a message when packets can be lost/corrupted in the network?

- Two options
  - Detect a loss/corruption and retransmit
  - Send data redundantly to tolerate loss/corruption
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)
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
Stop-and-Wait Performance

- Lousy performance if xmit 1 pkt << prop. delay
  - How bad?

- Want to utilize all available bandwidth
  - Need to keep more data “in flight”
  - How much? Called the bandwidth-delay product

- Also limited by quality of timeout (how long?)
Pipelined Transmission

- Keep multiple packets “in flight”
  - Allows sender to make efficient use of the link
  - Sequence numbers ensure receiver can distinguish frames

- Sender buffers outstanding un-acked packets
  - Receiver ACKs the highest *consecutive* frame received
    » ACKs are cumulative (covers current frame and all previous)
For Next Time

- Homework due in two weeks: 2/1