Lecture 5 - Coding and Error Control
Peter Steenkiste
Eric Anderson

Fall 2013
www.cs.cmu.edu/~prs/15-441-F13
From Signals to Packets

Analog Signal

“Digital” Signal

Bit Stream

Packets

Packet Transmission

Sender -> Receiver

© CMU 2013
Link Layer: Implementation

- Implemented in “adapter”
  - E.g., PCMCIA card, Ethernet card
  - Typically includes: RAM, DSP chips, host bus interface, and link interface
Datalink Functions

- Framing: encapsulating a network layer datagram into a bit stream.
  - Add header, mark and detect frame boundaries
- Media access: controlling which frame should be sent over the link next.
- Error control: error detection and correction to deal with bit errors.
  - May also include other reliability support, e.g. retransmission
- Flow control: avoid that the sender outruns the receiver
- Hubbing, bridging: extend the size of the network
Outline

- Encoding and decoding
  - Translate between bits and digital signal
- Framing
  - Bit stream to packets
- Packet loss & corruption
  - Error detection
  - Flow control
  - Loss recovery
How Encode?

- Seems obvious, why take time with this?
Why Do We Need Encoding?

• Keep receiver synchronized with sender.
• Create control symbols, in addition to regular data symbols.
  • E.g. start or end of frame, escape, ...
• Error detection or error corrections.
  • Some codes are illegal so receiver can detect certain classes of errors
  • Minor errors can be corrected by having multiple adjacent signals mapped to the same data symbol
• Encoding can be done one bit at a time or in multi-bit blocks, e.g., 4 or 8 bits.
• Encoding can be very complex, e.g. wireless.
Non-Return to Zero (NRZ)

- $1 \rightarrow$ high signal; $0 \rightarrow$ low signal
- Used by Synchronous Optical Network (SONET)
- Long sequences of 1’s or 0’s can cause problems:
  - Sensitive to clock skew, i.e. hard to recover clock
  - DC bias hard to detect – low and high detected by difference from average voltage
Clock Recovery

- When to sample voltage?
- Synchronized sender and receiver clocks
- Need easily detectible event at both ends
  - Signal transitions help resync sender and receiver
  - Need frequent transitions to prevent clock skew
  - SONET XOR’s bit sequence to ensure frequent transitions
Non-Return to Zero Inverted (NRZI)

- 1 → make transition; 0 → signal stays the same
- Solves the problem for long sequences of 1’s, but not for 0’s.
Manchester Encoding

- Used by Ethernet
- 0=low to high transition, 1=high to low transition.
- Transition for every bit simplifies clock recovery
- DC balance has good electrical properties
- But you pay a price …
  - Doubles the number of transitions – more spectrum!
  - Circuitry must run twice as fast
4B/5B Encoding

• Data coded as symbols of 5 line bits $\rightarrow$ 4 data bits, so 100 Mbps uses 125 MHz.
  • Uses less frequency space than Manchester encoding
• Encoding ensures no more than 3 consecutive 0’s
• Uses NRZI to encode resulting sequence
• 16 data symbols, 8 control symbols
  • Data symbols: 4 data bits
  • Control symbols: idle, begin frame, etc.
• Example: FDDI.
## 4B/5B Encoding

<table>
<thead>
<tr>
<th>Data</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>11110</td>
</tr>
<tr>
<td>0001</td>
<td>01001</td>
</tr>
<tr>
<td>0010</td>
<td>10100</td>
</tr>
<tr>
<td>0011</td>
<td>10101</td>
</tr>
<tr>
<td>0100</td>
<td>01010</td>
</tr>
<tr>
<td>0101</td>
<td>01011</td>
</tr>
<tr>
<td>0110</td>
<td>01110</td>
</tr>
<tr>
<td>0111</td>
<td>01111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10010</td>
</tr>
<tr>
<td>1001</td>
<td>10011</td>
</tr>
<tr>
<td>1010</td>
<td>10110</td>
</tr>
<tr>
<td>1011</td>
<td>10111</td>
</tr>
<tr>
<td>1100</td>
<td>11010</td>
</tr>
<tr>
<td>1101</td>
<td>11011</td>
</tr>
<tr>
<td>1110</td>
<td>11100</td>
</tr>
<tr>
<td>1111</td>
<td>11101</td>
</tr>
</tbody>
</table>
Other Encodings

- 8B/10B: Fiber Channel and Gigabit Ethernet
- 64B/66B: 10 Gbit Ethernet (& 40 and 100 Gb/S)
- B8ZS: T1 signaling (bit stuffing)

Things to Remember

- Encoding necessary for clocking
- Lots of approaches
- Rule of thumb:
  - Little bandwidth $\rightarrow$ complex encoding
  - Lots of bandwidth $\rightarrow$ simple encoding
From Signals to Packets

Analog Signal

“Digital” Signal

Bit Stream

0 0 1 0 1 1 1 1 0 0 0 1

Packets

0100010101011001010101110110000011110101011101010101101011010111001

Header/Body

Packet Transmission

Sender → Receiver
Outline

• Encoding
  • Digital signal to bits

• Framing
  • Bit stream to packets

• Packet loss & corruption
  • Error detection
  • Flow control
  • Loss recovery
Framing

- How do we break up a stream of bits into frames?

0100010101011100101010101100000011101010110101010110101101010111001
Framing

- A link layer function, defining which bits have which function.
- Minimal functionality: mark the beginning and end of packets (or frames).
- Some techniques:
  - out of band delimiters (e.g. 4B/5B control symbols)
  - frame delimiter characters with character stuffing
  - frame delimiter codes with bit stuffing
  - synchronous transmission (e.g. SONET)
Out-of-band: E.g., 802.5

- 802.5/token ring uses 4b/5b
- Start delim & end delim are “illegal” data codes
Delimiter Based

- SYN: sync character
- SOH: start of header
- STX: start of text
- ETX: end of text

- What happens when ETX is in Body?
Character and Bit Stuffing

- Mark frames with special character.
  - What happens when the user sends this character?
  - Use escape character when controls appear in data:
    - *abc*def → *abc\*def
  - Very common on serial lines, in editors, etc.

- Mark frames with special bit sequence
  - must ensure data containing this sequence can be transmitted
  - example: suppose 11111111 is a special sequence.
  - transmitter inserts a 0 when this appears in the data:
    - 11111111 → 111111101
  - must stuff a zero any time seven 1s appear:
    - 11111110 → 111111100
  - receiver unstuffs.
Ethernet Framing

- Preamble is 7 bytes of 10101010 (5 MHz square wave) followed by one byte of 10101011
- Allows receivers to recognize start of transmission after idle channel

```
| preamble | datagram | length | more stuff |
```
Clock-Based Framing

- Used by SONET
- Fixed size frames (810 bytes)
- Look for start of frame marker that appears every 810 bytes
- Will eventually sync up
How avoid clock skew?

- Special bit sequences sent in first two chars of frame
  - But no bit stuffing. Hmmm?
- Lots of transitions by xoring with special pattern (and hope for the best)
Outline

• Encoding
  • Digital signal to bits

• Framing
  • Bit stream to packets

• Packet loss & corruption
  • Error detection
  • Flow control
  • Loss recovery
Error Coding

- Transmission process may introduce errors into a message.
  - Single bit errors versus burst errors
- Detection:
  - Requires a convention that some messages are invalid
  - Hence requires extra bits
  - An (n,k) code has codewords of n bits with k data bits and r = (n-k) redundant check bits
- Correction
  - Forward error correction: many related code words map to the same data word
  - Detect errors and retry transmission
Error Detection

- EDC - Error Detection and Correction bits (redundancy)
- D - Data protected by error checking, may include header fields
- Error detection not 100% reliable!
  - Protocol may miss some errors, but rarely
  - Larger EDC field yields better detection and correction
Parity Checking

Single Bit Parity:
Detect single bit errors

\[
\begin{array}{c}
\text{data bits} \\
0111000110101011 \\
\text{parity bit}
\end{array}
\]
### Internet Checksum

**Goal:** detect “errors” (e.g., flipped bits) in transmitted segment

<table>
<thead>
<tr>
<th><strong>Sender</strong></th>
<th><strong>Receiver</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Treat segment contents as sequence of 16-bit integers</td>
<td></td>
</tr>
<tr>
<td>Checksum: addition (1’s complement sum) of segment contents</td>
<td></td>
</tr>
<tr>
<td>Sender puts checksum value into checksum field in header</td>
<td></td>
</tr>
<tr>
<td>Compute checksum of received segment</td>
<td></td>
</tr>
<tr>
<td>Check if computed checksum equals checksum field value:</td>
<td></td>
</tr>
<tr>
<td>NO - error detected</td>
<td></td>
</tr>
<tr>
<td>YES - no error detected. But maybe errors nonetheless?</td>
<td></td>
</tr>
</tbody>
</table>
Basic Concept: Hamming Distance

- Hamming distance of two bit strings = number of bit positions in which they differ.
- If the valid words of a code have minimum Hamming distance \( D \), then \( D-1 \) bit errors can be detected.
- If the valid words of a code have minimum Hamming distance \( D \), then \( \left(\frac{D-1}{2}\right) \) bit errors can be corrected.
Cyclic Redundancy Codes (CRC)

- Commonly used codes that have good error detection properties.
  - Can catch many error combinations with a small number of redundant bits
- Based on division of polynomials.
  - Errors can be viewed as adding terms to the polynomial
  - Should be unlikely that the division will still work
- Can be implemented very efficiently in hardware.
- Examples:
  - CRC-32: Ethernet
  - CRC-8, CRC-10, CRC-32: ATM
CRC: Basic idea

- Treat bit strings as polynomials:
  \[ 1 \ 0 \ 1 \ 1 \ 1 \]
  \[ X^4 + X^2 + X^1 + X^0 \]
- Sender and Receiver agree on a *divisor* polynomial of degree \( k \)
- Message of \( M \) bits \( \rightarrow \) send \( M + k \) bits
- No errors if \( M + k \) is divisible by divisor polynomial
- If you pick the right divisor you can:
  - Detect all 1 & 2-bit errors
  - Any odd number of errors
  - All Burst errors of less than \( k \) bits
  - Some burst errors \( \geq k \) bits
Outline

- Encoding
  - Digital signal to bits
- Framing
  - Bit stream to packets
- Packet loss & corruption
  - Error detection
  - Flow control
  - Loss recovery
Link Flow Control and Error Recovery

- Dealing with receiver overflow: flow control.
- Dealing with packet loss and corruption: error control.
- Meta-comment: these issues are relevant at many layers.
  - Link layer: sender and receiver attached to the same “wire”
  - End-to-end: transmission control protocol (TCP) - sender and receiver are the end points of a connection
- How can we implement flow control?
  - “You may send” (windows, stop-and-wait, etc.)
  - “Please shut up” (source quench, 802.3x pause frames, etc.)
  - Where are each of these appropriate?
A Naïve Protocol

• Sender simply sends to the receiver whenever it has packets.
• Potential problem: sender can outrun the receiver.
  • Receiver too slow, buffer overflow, ..
• Not always a problem: receiver might be fast enough.
Adding Flow Control

- Stop and wait flow control: sender waits to send the next packet until the previous packet has been acknowledged by the receiver.
  - Receiver can pace the receiver
Drawback: Performance

Max Throughput = \( \frac{1 \text{ pkt}}{\text{Roundtrip Time}} \)
Window Flow Control

- Stop and wait flow control results in poor throughput for long-delay paths: packet size/roundtrip-time.
- Solution: receiver provides sender with a window that it can fill with packets.
  - The window is backed up by buffer space on receiver
  - Receiver acknowledges a packet every time a packet is consumed and a buffer is freed
Bandwidth-Delay Product

Max Throughput = \frac{\text{Window Size}}{\text{Roundtrip Time}}
Error Recovery

• Two forms of error recovery
  • Error Correcting Codes (ECC)
  • Automatic Repeat Request (ARQ)

• ECC
  • Send extra redundant data to help repair losses

• ARQ
  • Receiver sends acknowledgement (ACK) when it receives packet
  • Sender uses ACKs to identify and resend data that was lost

• Which should we use? Why? When?
Stop and Wait

- Simplest ARQ protocol
- Send a packet, stop and wait until acknowledgement arrives
- Will examine ARQ issues later in semester
Recovering from Error

- Packet lost
- ACK lost
- Early timeout
How to Recognize Retransmissions?

- Use sequence numbers
  - both packets and acks
- Sequence # in packet is finite → How big should it be?
  - For stop and wait?
- One bit – won’t send seq #1 until received ACK for seq #0
Implementation Issues with Window-based Protocol

- Receiver window size: # of out-of-sequence packets that the receiver can receive
- Sender window size: # of total outstanding packets that sender can send without acknowledged
- How to deal with sequence number wrap around?
What is Used in Practice?

- No flow or error control.
  - E.g. regular Ethernet, just uses CRC for error detection
- Flow control only.
  - E.g. Gigabit Ethernet
- Flow and error control.
  - E.g. X.25 (older connection-based service at 64 Kbs that guarantees reliable in order delivery of data)