ICS 153
Introduction to Computer Networks

Inst: Chris Davison
cbdaviso@uci.edu
ICS 153
Data Link Layer Layer Protocols
Contents
• Simplex and Duplex Communication
• Frame Creation
• Flow Control
• Error Control
• Performance of Data Link Layer Protocols
• Character Orientated and Bit Orientated Protocols
ICS 153
Chapter 3
Homework

• # 3 #5 #6 #7 #8
• #14 #16 #26 #36 #37
Data Link Layer Functionality

- Recall:
  - Frame Creation
  - Error detection and/or correction
  - Flow Control

- Creates the illusion of a reliable link
Simplex and Duplex Communication

• Modes of Operation:
  – Simplex
  – Half Duplex
  – Full Duplex
Simplex Transmission

• Data: One way Only
  – Data is on a one-way street
Half Duplex Transmission

• Data: One direction at a time
  – Two way street but can only be going in one direction at a time
Full Duplex Transmission

• Data: Both ways simultaneously
  – Two way street with traffic on both lanes at one
Framing

• The data unit at the data link layer is the “frame”
• A frame is a group of bits, typically in sequence
• Issues:
  – Frame Creation
  – Frame Delineation
Frame Creation

- The Data Link Layer will accept a data unit from the Network Layer
  - Adds frame boundaries to the beginning and the end.
  - Additionally it will add control information to the frame.
Frame Delineation

• How to tell when a new frame starts
  – Character Count
  – Frame tags with character stuffing
  – Frame tags with bit stuffing
  – Physical layer coding violations
Delineation by Character Count

- Character count lists the number of characters in the data field of the frame
- Problem: corrupted control fields
  - Count can be garbled by a transmission error
Frame tagging with character stuffing

• Use starting and ending characters (tags) to mark the boundaries of the frame

• Problem: What if a tag character occurs in the data portion of the frame?
Character stuffing

• Insert extra escape characters when a tag appears in the data field

DLE  STX  Start Tag
DLE  ETX  End Tag
DLE  DLE  Character Stuffed DLE Code
Frame tagging with bit stuffing

• Bit strings may be used instead of character sequences to delineate frames

• More efficient
Bit Stuffing

• Each Frame begins with a start and end bit sequence, e.g., 01111110

• When a sender’s data link layer sees five 1’s in a row, it stuffs in a zero bit

• The receiver “unstuff” a zero after five consecutive 1’s.
Error Control

• No physical link is perfect
• Bits will be corrupted
• We can either:
  – detect errors and request retransmission
  – correct errors without retransmission
Error Control Definitions

• Codeword
  – unit of data containing data bits and check bits

• Hamming Distance
  – number of bit positions in which two codewords differ
  – XOR on two codewords and count the resulting 1’s
Hamming Distance
Example

10001001

XOR

1011001

00111000

• The Hamming distance between 10001001 and 1011001 is 3
Error Control

• Parity Bits
• Polynomial codes or checksums
Parity Bits

• Append a single parity bit to a sequence of bits
• If using “odd” parity, the parity bit is chosen to make the total number of 1’s in the bit sequence odd.
• If “even” parity, the parity bit makes the total number of 1’s in the bit sequence even
## Parity Bit Examples

<table>
<thead>
<tr>
<th>Transmitted Sequence</th>
<th>Parity</th>
<th>Parity Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010101</td>
<td>even</td>
<td>000101011</td>
</tr>
<tr>
<td>01111</td>
<td>even</td>
<td>011110</td>
</tr>
<tr>
<td>00010101</td>
<td>odd</td>
<td>000101010</td>
</tr>
<tr>
<td>01111</td>
<td>odd</td>
<td>011111</td>
</tr>
</tbody>
</table>
Parity Bits

• Only detects when there are an odd number of bit errors
• Does not detect an even number of bit errors
• It also has high overhead; it requires 1 extra bit for every several bits transmitted
Polynomial Codes

• Can detect errors on large chunks of data
• Has low overhead
• More robust than parity bit
• Requires the use of a “code polynomial”
  – e.g. $X^2 + 1$
Cyclic Redundancy Check

- CRC: Example of a polynomial code
- First: agree on a code polynomial
- Procedure:
  - 1. Let \( r \) be the degree of the code polynomial. Append \( r \) zero bits to the end of the transmitted bit string. Call the entire bit string \( S(x) \).
  - 2. Divide \( S(x) \) by the code polynomial using modulo 2 division.
  - 3. Subtract the remainder from \( S(x) \) using modulo 2 subtraction.
- Result is the checksummed message
Generating a CRC

Message: 1011
\[ = 1 \times x^3 + 0 \times x^2 + 1 \times x^1 + 1 \times x^0 \]
\[ = x^3 + x + 1 \]

Code Polynomial: \( x^2 + 1 \) (101)

Step 1: Compute \( S(x) \)
\[ r = 2 \quad \text{(degree of code polynomial)} \]
\[ S(x) = 101100 \]
Generating a CRC

Step 2: Modulo 2 divide:

```
   1001
  101 | 101100
     101
     001
     000
     000
     100
     101
     01
```
Generating a CRC

Step 3: Modulo 2 (XOR) subtract the remainder from S(X)

```
  101100
- 01
  101101  <-- Checksummed Message
```
Decoding a CRC

• Procedure
  – 1. Right-most n bits are checksum bits, where n is the degree of the code polynomial.
  – 2. Divide the checksummed message by the code polynomial using modulo 2 division. If the remainder is zero, there is no error detected.
Decoding a CRC

Agreed upon code polynomial:
\[ X^2 + 1 \ (101) \]

101101 \quad \text{ <-- Checksummed Msg.} \\
1011 \quad \text{ <-- Original Msg.}
Decoding a CRC

- Error Check the message:

\[
\begin{array}{c}
1001 \\
101 | 101101 \\
101 \\
001 \\
000 \\
000 \\
010 \\
000 \\
101 \\
101 \\
00 <= Remainder = 0
\end{array}
\]

No Error detected
Decoding a CRC with an Error

• When a bit error occurs, there is a large probability that it will produce a polynomial that is not an even multiple of the code polynomial, and thus errors can usually be detected.
Decoding a CRC with an Error

Original Message: 1011
CORRECT checksum: 101101
ERROR checksum: 101001

\[
\begin{array}{c}
1000 \\
\hline \\
101 & 101001 \\
\hline \\
101 \\
000 \\
\hline \\
000 \\
000 \\
\hline \\
000 \\
000 \\
\hline \\
001 \\
000 \\
\hline \\
01 <- Remainder = 1
\end{array}
\]
Choosing a CRC Polynomial

• The longer the polynomial, the smaller the probability of an undetected error

• Common standard polynomials

  CRC-12:
  \( x^{12} + x^{11} + x^3 + x^2 + x^1 + 1 \)

  CRC-16:
  \( x^{16} + x^{15} + x^2 + 1 \)

  CRC-CCITT:
  \( x^{16} + x^{12} + x^5 + 1 \)
Error Correction

• Parity Bits and polynomial codes detect errors, but can we correct them without retransmitting information?
  – Yes: by using Hamming Codes
Hamming Codes

• Hamming codes, like polynomial codes, are appended to the transmitted message

• Hamming codes, unlike polynomial codes, contain the information necessary to locate a single bit error
Calculating a Hamming Code

• Procedure:
  – Decide on even or odd parity
  – Place message bits in their non-power-of-two Hamming positions
  – Build a table listing the binary representation of each of the message bit positions
  – calculate the check bits
Hamming Code Example

• Parity: Odd
• Message to be sent: 1011
Calculating a Hamming Code

Check bits go into “power-of-two” positions:

\[2^0, 2^1, 2^2, \text{ etc...}\]

\[= 1, 2, 4, \text{ etc...}\]

Message bits go into “non-power-of-two” positions

\[= 3, 5, 6, 7, \text{ etc...}\]
Calculating a Hamming Code

Table

<table>
<thead>
<tr>
<th>ASCII</th>
<th>BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>

check bit 1: 1,3,5,7
check bit 2: 2,3,6,7
check bit 4: 4,5,6,7
Calculating a Hamming Code

Original Message: 1011
Check bit position 1 checks positions: 1, 3, 5, 7
and we are using odd parity
\[
\begin{array}{cccccc}
& & 1 & & 0 & 1 & 1 \\
\end{array}
\]
so check bit 1 = 1
\[
\begin{array}{cccccc}
1 & & 1 & & 0 & 1 & 1 \\
\end{array}
\]
Calculating a Hamming Code

Original Message: 1011
Check Bit position 2 checks positions: 2, 3, 6, 7
and we are using odd parity

__ __ 1 __ 0 1 1

so check bit 2 = 0

1 0 1 __ 0 1 1
Calculating a Hamming Code

Original Message: 1011
Check Bit position 4 checks positions: 4, 5, 6, 7
and we are using odd parity

\[ \underline{\_ \_ \_ 1 \_ \_ 0 1 1} \]

so check bit 4 = 1

\[ \underline{1 0 1 1 0 1 0 1} \]

\[ \underline{1} \]
Calculating a Hamming Code

• Original Message: 1011
• Sent Message: 1011011
• How do we check/correct for single bit errors in the sent message?
Correcting a bit error

Original Message: 1011
Transmitted Message: 1011011
Received Message: 101100\_1

Procedure:
- Initialize a counter (n) to zero.
- Examine each check bit for correct parity
  - If parity is incorrect add the check bit to the counter.
  - If $n = 0$ after all check bits are examined then the codeword is valid else $n = $ the
Correcting a bit error

Original Message: 1011
Transmitted Message: 1011011
Received Message: 1011001

check bit 1: 1,3,5,7
parity = odd
• 1011001
No Error in Check bit 1
N = 0
Correcting a bit error

Original Message: 1011
Transmitted Message: 1011011
Received Message: 1011001

check bit 2 : 3,6,7
parity = odd
• 1011001
Error in Check bit 2
N = N + 2
N = 2
Correcting a bit error

Original Message: 1011
Transmitted Message: 1011011
Received Message: 1011001

check bit 4 : 5,6,7
parity = odd
• 1011001
Error in Check bit 4
N = N + 4
N = 6
Correcting a bit error

• $N = 6$ (change bit 6)
Original Message: 1011
Transmitted Message: 1011011
Received Message: 1011001
Corrected Message: 1011011
Final Word on Hamming Codes

• Hamming codes can be used to locate and correct a single-bit error

• If more than one bit is in error, then a Hamming code cannot correct it

• Hamming codes, like parity bits, are only useful on short messages
Flow Control

• What happens if the sender tries to transmit faster than the receiver can accept?

• Data will be lost unless flow control is implemented
Some Flow Control Algorithms

• Simplex protocols
  – Flow control for the ideal network
  – Stop and Wait for noiseless channels
  – Stop and wait for noisy channels

• Full duplex protocols
  – Sliding window with Go Back N
  – Sliding window with Selective Repeat
Simplex Flow Control

• Data Only flows in one direction

• Acknowledgement stream may flow in the other direction
Flow Control in the Ideal Network

• Assumptions:
  – Error Free transmission line
  – Infinite buffer at the receiver

• No acknowledge of frames necessary
  – Since the transmission line is error-free and the receiver can buffer as many frames as it likes, no packet will ever be lost
Flow Control in the ideal Network

• Problem: The ideal network does not exist!
Stop and Wait with Noiseless channels

• Assumptions:
  – Error free transmission line
  – Finite buffer at the receiver

• Buffer Overflow
  – Buffer overflow may happen at the receiver when the sender sends frames at a rate faster than the receiver
Stop and Wait with Noiseless channels

• Solution to buffer overflow: Stop and Wait
  – The receiver sends an acknowledgement telling the sender to transmit the next frame.

  – The sender waits for the ACK and if the ACK comes, it transmits the next data frame.
Stop and Wait with Noiseless channels

Sender          Data
Receiver

Sender         ACK
Receiver

Sender         ACK
Receiver
Stop and Wait with Noiseless channels

• We assume an error-free transmission line and hence, ACK frames will not be lost.

• In the protocol there are two types of frames:
  – Data frames
  – ACK frames
  • The ACK frame does not carry any information since the arrival at the sender is the only important issue
Stop and Wait with Noiseless channels

• Problem:
  – Transmission lines are noisy and the do get errors
  – ACK frames can be lost!
Stop and Wait for Noisy Channels

Problems:

• Problem 1: Loss of an ACK frame
  – Since the transmission line is not error-free, an ACK frame may be lost, causing the data sender to wait indefinitely
Stop and Wait for Noisy Channels

Problems:

• Can we solve problem 1 by introducing a timeout period for the sender? Yes, but...

• Problem 2: Duplicate Frames
  – If the ACK frame for a certain data frame is lost, the sender will retransmit the same frame after a time-out period. Then the receiver will have two copies of the same frame.
Stop and Wait for Noisy Channels

Problems:

• Duplicate frames

Solution

– The sender uses a timer to retransmit data frames when an ACK has not arrived

– The sender includes a frame sequence number with each frame to distinguish one from another. Thus, the receiver knows when it has received duplicate frames.
Full Duplex Flow Control Protocols

• Data Frames are transmitted in both directions
  – Sliding window flow control protocols
Sliding Window Protocols Definitions

• **Sequence Number**
  – Each frame is assigned a sequence number that is incremented as each frame is transmitted

• **Sender’s Window**
  – Keeps sequence numbers of frames that have been sent but not yet acknowledged

• **Sender’s Window Size**
  – The number of frames the sender may transmit before receiving ACKs
Sliding Window Protocols

Definitions

• Receiver’s Window
  – Keeps sequence numbers of frames that the receiver is allowed to accept

• Receiver’s Window Size
  – The maximum number of frames the receiver may receive out of order
Sliding Window Protocols

General Remarks

• The sending and receiving windows do not have to be the same size

• Any frame which falls outside the receiving window is discarded at the receiver

• Unlike the sender’s window, the receiver’s window always remains at its initial size
Since we have full duplex transmission, we can “piggyback” an ACK onto the header of an outgoing data frame to make better use of the channel.

When a data frame arrives at the receiver, instead of immediately sending a separate ACK, the receiver waits until it is passed the next data frame to send. The acknowledgement is attached to the outgoing data frame.
Simple Sliding Window with Window Size of 1

- The protocol behaves identically to stop and wait for a noisy channel.
Sliding Window with Window Size $W$

- With a window size of 1, the sender waits for an ACK before sending another frame.
- With a window size of $W$, the sender can transmit up to $W$ frames before “being blocked”.
- We call using larger window sizes Pipelining.
Sender-Side Window with Window Size W=2

(a) Initial window state
(b) Send frame 0
(c) Send frame 1
(d) ACK for frame 0 arrives
(e) Send frame 2
(f) ACK for frame 1 arrives
(g) ACK for frame 2 arrives, send frame 3
(h) ACK for frame 3 arrives
Receiver-Side Window with Window Size $W=2$

- (a) Initial window state
- (b) Nothing happens
- (c) Frame 0 arrives, ACK frame 0
- (d) Nothing happens
- (e) Frame 1 arrives, ACK frame 1
- (f) Frame 2 arrives, ACK frame 2
- (g) Nothing happens
- (h) Frame 3 arrives, ACK frame 3
Why do Pipelining?

• In other words, why have a window size greater than 1?

• By allowing several frames into the network before receiving acknowledgement, pipelining keeps the transmission line from being idle.
  – Increases throughput
What about Errors?

• What if data or acknowledgement frames are lost when using a sliding window protocol?
Solution

- Two solutions:
  - Go Back N
  - Selective Repeat
Sliding Window with Go Back N

• When a receiver notices a missing or errored frame, it simply discards all frames with greater sequence numbers and sends no ACK
• The sender will eventually time out and retransmit all the frames in its sending window
• The Receiver’s window size is 1.
Sliding Window with Go Back N

- Ws - Sender Window size
- Wr - Receiver Window size
- N - Maximum Number of frames outstanding

- Wr=1
- N= largest sequence number
  - MAX_SEQ
  - numbered from 0…N
- Ws= N
Go Back N

• Go Back N can recover from errored or missing packets but...

• It is wasteful. If there are a lot of errors the sender will spend most of its time retransmitting useless information
  – This will waste a lot of bandwidth.
Sliding Window with Selective Repeat

- The sender retransmits only the frame with errors.
  - The receiver stores all the correct frames that arrive following the bad one.
    - This requires a significant amount of buffer space at the receiver.
  - When the sender notices that something is wrong, it just retransmits the one bad frame, not all of its successors.
Sliding Window with Selective Repeat

- **Ws** - Sender Window Size
- **Wr** - Receiver Window Size
- **N** - Maximum Number of frames outstanding

- **N** = MAX_SEQ
- **Wr and Ws** = \(1/2\) of (largest sequence number + 1)
  - numbered from 0…N
  - \((\text{MAX_SEQ} + 1)/2\)
Fig. 3-19. (a) Initial situation with a window of size seven. (b) After seven frames have been sent and received but not acknowledged. (c) Initial situation with a window size of four. (d) After four frames have been sent and received but not acknowledged.
Comparison of Go Back N vs. Selective Repeat

• Go Back N is a bandwidth intensive approach in dealing with pipelining errors. However, the receiver needs no buffer space because it has a window size of 1.

• Selective Repeat uses less bandwidth for retransmission but the receiver must have buffer space.
Protocol Specification Verification:

Finite State Machines

- **Finite State Machine**
  - Mathematical technique used to specify and verify protocols.
  - A sender or receiver (protocol machine) is always in a specific state at every instance of time.

- **Initial State**
  - Corresponds to the state of the system when it starts running

- **Transition**
  - Change from one state to another

- **Reachability Analysis**
  - Determining which states are
Finite State Machines

• Deadlock
  – Situation in which the protocol can make no forward progress.
Protocol Specification Verification: Petri Net Model

• Perti Net
  – Another technique for formally specifying protocols

• Place
  – A state which the system may be in

• Transition
  – Change from one place to another
  – A transition is **enabled** if it contains an input token
    • An enabled transition is said to “fire” when it moves a token from the input place to the output place
Pteri Net Model

• Arcs
  – Input arcs enter the place
  – Output arcs leave the place

• Petri Nets can be represented in Algebraic form.
Fig. 3-22. A Petri net with two places and two transitions.
Bit-Oriented and Character-Oriented Protocols

• Character Orientated Protocol
  – Basic Unit: Character
  – Frame length is a multiple of character size
  – Example: Internet PPP

• Bit Oriented Protocol
  – Basic unit: Bit
  – Frame length is not a multiple of character size
  – Example: HDLC
    • High Level Data Link Control
Example Character Orientated Protocol

- PPP = Point-to-Point Protocol
- Used Widely in the Internet
- PPP uses frame tags with character stuffing
Character Orientated Protocols

• With character orientated protocols, a frame is composed of characters in some character code (e.g., ASCII, EBCDIC, UNICODE).

• A computer with 9-bit characters cannot send arbitrary messages in ASCII code. They must be chopped and repacked into units of 8 bits.

• Bit orientated protocols do not require such chopping up and repackaging of
Example Bit Orientated Protocol

- HDLC = High-level Data Link Control
- Used in X.25 networks
- HDLC uses frame tags with bit stuffing