Chapter 3

The Data Link Layer
Data Link Layer

- Algorithms for achieving reliable, efficient communication between two adjacent machines.
- Adjacent means two machines are physically connected by a communication channel that acts like a wire (bits are delivered in exactly the same order in which they are sent)
Role of the Data Link Layer

To provide services to the network layer(3) by enhancing the services provided by the physical layer (1).

Several types of services:
- **reliable service on a link (almost no error, no loss, no reordering)**
  - error detection and recovery at the data link layer
  - flow control to protect buffers at receiver
- **unreliable service on a link (multiple Access)**
  - error detection done at the data link layer
  - recovery done at higher layers
Data Link Layer Design Issues

- Services Provided to the Network Layer
- Framing
- Error Control
- Flow Control
Functions of the Data Link Layer

- Provide service interface to the network layer
- Dealing with transmission errors
- Regulating data flow
  - Slow receivers not swamped by fast senders
Relationship between packets and frames.
Services Provided to Network Layer

(a) Virtual communication.
(b) Actual communication.
Services Provided to the Network Layer

1. Unacknowledged Connectionless Service
2. Acknowledged Connectionless Service
3. Acknowledged Connection-oriented Service
   (Three phases)
   · Connection establishment
   · data transfer
   · disconnection
Services to Network layer (1)

- **Unacknowledged connectionless service**
  - No connection required and **without acknowledgement** for data frames
  - Appropriate for **low error rate and real-time traffic**
  - Error recovery is up to **higher layer** (or no error recovery e.g. digitized voice)
Services to Network layer (2)

- Acknowledged connectionless service
  - No connection required but each frame is individually acknowledged
  - Useful for unreliable channel, such as wireless systems.
  - Transport layer may do *message recovery* but is more expensive than *frame recovery* at data link layer
Services to Network layer (3)

- Acknowledged connection-oriented service
  - Guarantee error-free and in sequence delivery of data frames
  - Consists of three phases
    - Connection set up (variables and buffers initialization)
    - Data frame transmission
    - Connection termination (free of variable and buffers)
Services Provided to Network Layer (2)

Placement of the data link protocol.
Framing

Break bit stream from physical layer into frames for error detection and recovery.

Four framing methods

1. Character count
2. Flag bytes with byte (character) stuffing
3. Starting and ending flags, with bit stuffing
4. Physical layer coding coding violations
Framing Approaches (1)

Character Count

- Indicate the frame boundary by frame length
- Once an error, frame boundary cannot be recognized and thus recovery is impossible.
A character stream.  (a) Without errors.  (b) With one error.
Framing Approaches (2)

Flag bytes with byte stuffing (character stuffing)

– Delimit the frame by flag bytes

– Prevent frame boundary from appearing at the data content by character stuffing.

– **Character Stuffing**: inserting **ESC** ahead of accidental flag byte within the data content.
Framing Approaches (2)

(a) A frame delimited by flag bytes.
(b) Four examples of byte sequences before and after stuffing.
Framing Approaches (3)

Starting and ending flags, with bit stuffing

– Begin and end with a flag byte “01111110”
– Prevent a flag from appearing in data by bit stuffing.
– Bit stuffing: inserting 0 after five continuous bit “1” data appear.
Framing Approaches (3)

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 | 1 1 1 1 1 0 0 1 0
   6 consecutive 1’s

(b) 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1 0
   5 consecutive “1”
   + 1 “0”
   Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Bit stuffing
(a) The original data.
(b) The data as they appear on the line.
(c) The data as they are stored in receiver’s memory after destuffing.
Framing Approaches (4)

Physical Layer Coding Violations.

- Used when physical layer encoding contains redundancy
- Example: 01 (H), 10 (L) then 00 or 11 can be used to delimit a frame.

Note: One or more combination of the approaches may be used to provide extra protection for framing.
Neutral versus bipolar bit streams. (a) Alternate 1’s and 0’s transmitted in a neutral mode.
(b) Equivalent in a bipolar mode.
(c) Framing by coding violation.
Error Control

- Need mechanisms such as
  - Error detection
  - Acknowledgement or NACK
  - Timer
  - Retransmission
  - Sequence numbering
Flow Control

- When receiver processes frames slower than the sender, congestion occurs.
- Needs some feedback to prevent sender from sending faster than the receiver can process.

(or a flow rate agreement in advance for Virtual Circuits)
Error Detection and Correction

- Error-Correcting Codes
- Error-Detecting Codes
Error Detection and Correction

Error-Correcting codes

An \( n \)-bit codeword \(( n = m + r )\) consists of \( m \) data bits and \( r \) redundant bits.

Hamming distance is the number of bits positions in which two codewords differ.

Example: The Hamming distance between the codewords 10001001 and 10110001 is 3.
Error Correcting Code

- Codeword = Data + Check-bit
- \( n \)-bit codeword = \( m \) data bit + \( r \) check-bit
- \( 2^m \) out of \( 2^n \) are legal
- Hamming distance = The minimum number of bit positions in which two codewords differ.
- \( H = d+1 \) --> detect \( d \) errors;
- \( H = 2d+1 \) --> correct \( d \) errors.
Error-Correcting Codes

- **Parity bit:**
  - detect single bit error and $H = 2$.

- **Example of $H = 5 \Rightarrow d = (5-1)/2 = 2$**
  - $0000000000$
  - $0000011111$
  - $1111100000$
  - $1111111111$
  - $00000000111$ ➞ two bit errors in $0000011111$
  - $00000000011$ ➞ three bit errors in $0000011111$

  Out of Correcting Capability
Error Correcting Codes

- Correcting single bit error of \( n \)-bit codeword requires \((m + r + 1) \leq 2^r\) (lower bound for \( r \))
- \( n + 1 \) bit patterns dedicated to one codeword \( \Rightarrow (n+1)2^m \leq 2^n \) and \( n = m + r \)

\[
\text{N possible bit patterns at a distance 1 from it}
\]
Hamming code

- Achieve lower bound of $r$
- The codeword is numbered consecutively starting from left end as 1.
- The bits of powers of 2 (1, 2, 4, 8, …) are check bits; the rest (3, 5, 7, …) are filled with $m$ data bits.
- A check bit forces the parity of some collection of bits, including itself, to be even (or odd).
Hamming Code (2)

- A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8)

- **Checking algorithm**
  - Initialize counter == 0
  - Examine all check bits
    - If check bit k is error, add k into the counter.
    - After all check bits are checked, the counter contains the number of the incorrect bit.
### ASCII

<table>
<thead>
<tr>
<th>Transmitting</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>2^3</td>
<td>2^2</td>
<td>2^1</td>
<td>2^0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If received 1 1 1 1 1 1 1 0 0 1 0 0 1
If error occurs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Error in second bit!
Use of a Hamming code to correct burst errors.
Error Detecting Code

- **Parity code**
  - detect single or odd # of bit errors
  - detect burst error of n-bit by matrix checksum on each column of n-bit wide and h-bit high data and put the checksum at the h+1 row.

- **Cyclic redundancy code**
  - Polynomial code
  - Using Exclusive OR in addition and subtraction.
3.2.2 Error-Detecting Codes

Error-Correcting V.S. Error-detecting
with Retransmission

Cyclic redundancy code (CRC) is commonly used.

\[ CRC-12 = x^{12} + x^{11} + x^3 + x^2 + x + 1 \]
\[ CRC-16 = x^{16} + x^{15} + x^2 + 1 \]
\[ CRC-CCITT = x^{16} + x^{12} + x^5 + 1 \]

When the CRC method is employed the sender and receiver must agree upon a generator polynomial \( G(x) \).

The checksum is computed and then the sender transmits the checksum frame.
The algorithm for computing the checksum and the method for forming the checksum frame are as follows:

1. Let $r$ be the degree of $G(x)$. Append $r$ zero bits to the low-order end of the frame, so it now contains $m + r$ bits and corresponds to the polynomial $x^r M(x)$.

2. Divide the bit string corresponding to $G(x)$ into the bit string corresponding to $x^r M(x)$ using modulo 2 division.

3. Subtract the remainder (which is always $r$ or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial $T(x)$. 
CRC Algorithm

- Shift $M(x)$ left by $r$ bits ($r$ is the degree of $G(x)$).
- Divide $x^rM(x)$ by $G(x)$
- Subtract $x^rM(x)$ by the remainder in last step to obtain the checksumed frame to be transmitted, $T(x)$.
- $T(x)$ is divisible by $G(x)$
CRC Error Check

- Receive T’(x)
- Divide T’(x)/G(x)
  - if no remainder, the frame is accepted
  - if yes, error is found.
  - T’(x) = T(x) + E(x)
  - T’(x)/G(x) = T(x)/G(x) + E(x)/G(x) = E(x)/G(x)
Error Detection of CRC

- **Single bit error:** $E(x) = x^i$
  - Not divisible by $G(x)$ if $G(x)$ has more than one term.

- **Double bit Error:** $E(x) = x^i + x^j = x^j(x^{i-j} + 1)$
  - Low degree polynomials the give protection to long frames are known.
  - E.g., $x^{15} + x^{14} + 1$ will not divide $x^k + 1$ for any $k < 32768$

- **Odd # of bits in Error:**
  - No polynomial with odd No. of terms contain a factor of $(x+1)$
  - if $G(x)$ contain $(x+1)$, indivisible for any odd No. of errors.
Error Detection of CRC (2)

- Detect all errors of length \(< r\), if the degree of G(x) is r.

- Undetectable error of r+1 bit (the first and the last of the r+1 bits must be 1) with prob. \(1/2^{(r-1)}(r+1-2\) intermediate bits)

- Undetected error of longer than r+1 bits with prob. \(1/2^r\)

- Example of G(x)
  - CRC-12 = \(x^{12}+x^{11}+x^3+x^2+x^1+1\)
Error-Detecting Codes

Calculation of the polynomial code checksum.

Frame : 1101011011
Generator: 10011
Message after 4 zero bits are appended: 11010110110000

Transmitted frame: 11010110111110

Message

Checksum

Remainder
General Implementation of the Polynomial Code Checksum by Shift Register and Adder

\[ G(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots \ldots a_1 x + a_0 \]
BCC Accumulation Using CRC-12, Transmit Sequence

NOTES

□ = BCC REGISTER STAGE
⊕ = EXCLUSIVE-OR
CRC-12 POLYNOMIAL = \(x^{12} + x^{11} + x^9 + x^2 + x + 1\)
LSB = LEAST SIGNIFICANT BIT OF REGISTER (SENT FIRST)
MSB = MOST SIGNIFICANT BIT OF REGISTER (SENT LAST)
BCC Accumulation Using CRC-CCITT, Transmit Sequence

**Notes:**

- **□** = BCC Register Stage
- **○** = Exclusive-Or

CRC-CCITT Polynomial = \( x^{16} - x^{12} + x^5 + 1 \)

**LSB** = Least Significant Bit of Register (Sent First)

**MSB** = Most Significant Bit of Register (Sent Last)
3.3 Elementary Data Link Protocols

3.3.1 An unrestricted Simplex Protocol  (No Ack, No Sequence #)
3.3.2 A Simplex Stop-and-Wait Protocol  (for flow control, half duplex channel)  (with Ack)
   (No sequence number)
3.3.3 A Simplex Protocol for a Noisy Channel
   (stop-and-wait)

![Diagram of data link protocols]

- A synchronized to network layer
If there is [no sequence number], consider the following scenario:

1. The network layer on \( A \) gives packet 1 to its data link layer. The packet is correctly received at \( B \) and passed to the network layer on \( B \). \( B \) sends an acknowledgement frame back to \( A \).

2. The acknowledgement frame gets lost completely. It just never arrives at all. Life would be a great deal simpler if the channel only mangled and lost data frames and not control frames, but sad to say, the channel is not very discriminating.

3. The data link layer on \( A \) eventually times out. Not having received an acknowledgement, it (incorrectly) assumes that its data frame was lost or damaged and sends the frame containing packet 1 again.
4. The **duplicate frame** also arrives at data link layer on \( B \) perfectly and is unwittingly passed to the network layer there. If \( A \) is sending a file to \( B \), part of the file will be **duplicated** (i.e., the copy of the file made by \( B \) will be incorrect and the error will not have been detected). In other words, the protocol will fail.
Data Link Protocols

- **ARQ: Automatic Repeat reQuest**
  - Timer
  - Acknowledgement
  - Sequence number
  - Retransmission

- **ARQ must handle**
  - Garbled frames
  - Lost frames
  - Lost acks
  - Duplicate frames
Complicate Issue in Full-Duplex

- Both sides (A and B) can send data simultaneously.
- Intermix ack and data frame in each direction by **piggybacking**.
- How long can an ACK hold before is sent?
Sliding Window Protocols

- Each outbound frame contains a sequence number, ranging from 0 to $2^n - 1$ for n-bit field.

- Sender keeps a sending window
  - represent the frames with the sequence numbers in the list that are sent but as yet not acknowledged.

- Receiver keeps a receiving window
  - Represent the acceptable frame sequence numbers.
Sliding Window Protocols (2)

- A One-Bit Sliding Window Protocol
- A Protocol Using Go Back N
- A Protocol Using Selective Repeat
Sliding Window Protocols (3)

- **Stop and Wait:** \( w = 1 \).
  - The sender sends one frame and then waits for an acknowledgement before proceeding.

- **Go back N:** \( w = \text{maximal sequence number} - 1 = \text{MAX}_\text{SEQ} \)
  - Corresponding receiving window of size 1.
  - Sender retransmit all packets starting with the damaged or lost one.

- **Selective Repeat:** \( w = \frac{\text{half of the maximal sequence number}}{2} \)
  - Receiver stores all correct frames
  - Sender retransmit the bad frames.
Sliding Window Protocols (3)

A sliding window of size 1, with a 3-bit sequence number.
(a) Initially.
(b) After the first frame has been sent.
(c) After the first frame has been received.
(d) After the first acknowledgement has been received.
A One-Bit Sliding Window Protocol (2)

Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.
3.4.2 A Protocol Using Go Back n

( maximal number of unacknowledged frame
  = maximal window size = maximal sequence number - 1 )

= MAX_SEQ
Consider the following scenario with sequence number of 8

1. The sender sends frames 0 through 7.
2. A piggybacked acknowledgement for frame 7 eventually comes back to the sender.
3. The sender sends another eight frames, again with sequence numbers 0 through 7.
4. Now another piggybacked acknowledgement for frame 7 comes in.

Did all eight frame belonging to the second batch arrive successful?

\[ \text{A} \]

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \uparrow \text{ack7} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\uparrow \text{ack7} \\
\end{array}
\]

\[ \text{A} \]

\[
\begin{array}{cccccccc}
0 & -0, k, A0 \\
\end{array}
\]

\[(s, 7, BX)\]

seq \quad \text{ack} \quad \text{info}\]
3.4.3 A Protocol Using **Selective Repeat**

(maximal window size = half of the maximal sequence number)

$$= (\text{MAX}_\text{SEQ} +1)/2$$

---

**Fig. 3-16.** Effect of an error when the receiver window size is large.
A Sliding Window Protocol Using Selective Repeat (5)

(a) Initial situation with a window size seven.
(b) After seven frames sent and received, but not acknowledged.
(c) Initial situation with a window size of four.
(d) After four frames sent and received, but not acknowledged.
If the window size is larger than half of the maximum sequence number, consider the following scenario (e.g., maximum sequence number = 8, window size = 7):

Can the receiver tell that these packets are not duplicate?
Fig. 3-18. Simulation of multiple timers in software.
Protocol Verification

- Finite State Machined Models
- Petri Net Models
Protocol Specification and Verification

- **Finite State Machine Models:**
  - Each protocol machine (i.e., sender or receiver) is always in a specific state at every instant of time.
  - States consists of all the values of its variables, including the program counter.
  - Example: stop-and-wait protocol, the receiver can be at waiting for frame 0 or waiting for frame 1 state.
Finite State Machine

- The state of the complete system is the combination of all the states of the two protocol machines and the channel.

- Example of channel state:
  - a zero frame or a one frame moving from sender to receiver, an ack frame going the other way, or an empty channel.
  - If sender and receiver each has two states, the system has 16 states.
FSM (2)

- Transition between states occurred when event happens.
- A state is designated as initial state.
- Reachability analysis is performed to see if all states are reachable from the initial state.
- FSM is regarded as \((S, M, I, T)\)
  - \(S\): the set of states
  - \(M\): the set of frames
  - \(I\): the set of initial states
  - \(T\): the set of transitions
Finite State Machined Models

(a) State diagram for protocol 3. (b) Transmissions.

Receiver expected number

Initial state

Ack lost

Ack lost

The frame sender trying to send

State of channel, 0, 1, A, or empty

<table>
<thead>
<tr>
<th>Transition</th>
<th>Who runs?</th>
<th>Frame accepted</th>
<th>Frame emitted</th>
<th>To network layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>_</td>
<td>(frame lost)</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>1</td>
<td>R</td>
<td>0</td>
<td>A</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>A</td>
<td>1</td>
<td>_</td>
</tr>
<tr>
<td>3</td>
<td>R</td>
<td>1</td>
<td>A</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td>A</td>
<td>0</td>
<td>_</td>
</tr>
<tr>
<td>5</td>
<td>R</td>
<td>0</td>
<td>A</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>R</td>
<td>1</td>
<td>A</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>S</td>
<td>(timeout)</td>
<td>0</td>
<td>_</td>
</tr>
<tr>
<td>8</td>
<td>S</td>
<td>(timeout)</td>
<td>1</td>
<td>_</td>
</tr>
</tbody>
</table>
Reachability Analysis Capability

- **Incompleteness**
  - Ex. Actions are not specified for an event

- **Deadlock**
  - There exists a set of states from which no exit and from which no progress can be made.

- **Extraneous transition**
  - Protocol specifies the handling of an event that can never happen.
Reachability Analysis

Formally, a finite state machine model of a protocol can be regarded as a quadruple \((S,M,I,T)\) where:

- \(S\) is the set of states the processes and channel can be in.
- \(M\) is the set of frames that can be exchanged over the channel.
- \(I\) is the set of initial states of the processes.
- \(T\) is the set of transitions between states.

At the beginning of time, all processes are in their initial states. Then events begin to happen, such as frames becoming available for transmission or timers going off. Each event may cause one of the processes or the channel to take an action and switch to a new state. By carefully enumerating each possible successor to each state, one can build the reachability graph and analyze the protocol.
Deadlock

- A deadlock is characterized by the existence of a subset of states that is reachable from the initial state and has two properties:
  - There is no transition out of the subset
  - There are no transitions in the subset cause forward progress.
A Petri net with two places and two transitions.
Petri Net Models (2)

A Petri net model for protocol 3.
Petri net can be represented in convenient algebraic form. Each transition contributes one rule to the grammar. Each rule specifies the input and output places of the transition.

1: BD → AC
2: A → AC
3: AD → BE
4: B → BE
5: C →
6: D → \{ \text{token loss} \}
7: E →
8: CF → DF
9: EG → DG
10: CG → DF
11: EF → DG
Example of DLC

- SDLC (Synchronous Data Link Control) -- IBM
- ADCCP -- ANSI
- HDLC (High-level Data Link Control) -- ISO
- LAP (Link Access Procedure) and LAPB -- CCITT
- They are all derived from SDLC
3.6 Example Data Link Protocols

3.6.1 HDLC (High-level Data Link Control)

This protocol is derived from IBM's SNA, called SDLC (Synchronous Data Link Control protocol). ANSI modified it to become ADCCP (Advanced Data Communication Control Procedure). ISO modified it to become HDLC. CCITT modified HDLC for its LAP (Link Access Procedures) as part of X.25 but later modified it again to LAPB.

Useful for multi-drop line

```
<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>&gt;0</th>
<th>16</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>011111110</td>
<td>Address</td>
<td>Control</td>
<td>Data</td>
<td>Checksum</td>
<td>0111111110</td>
</tr>
</tbody>
</table>
```

Fig. 3-24. Frame format for bit-oriented protocols.
High-Level Data Link Control (2)

Frame sequence number

<table>
<thead>
<tr>
<th>Bits</th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>0</td>
<td>Seq</td>
<td>P/F</td>
<td>Next</td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
<td>0</td>
<td>Type</td>
<td>P/F</td>
</tr>
<tr>
<td>(c)</td>
<td>1</td>
<td>1</td>
<td>Type</td>
<td>P/F</td>
</tr>
</tbody>
</table>

Piggy back ack

Control field of
(a) An information frame.
(b) A supervisory frame.
(c) An unnumbered frame.
HDLC Flavored DLC

- Bit-oriented and use bit-stuffing
- Frame structure (See Fig 3-24)
- Delimited by flag 01111110
- Consists of three kinds of frames
  - Information, supervisory, and unnumbered
- Sliding window with n=3 bits
- Full-duplex with piggy back
Supervisory Frames

- Defined by Type field
- Examples
  - Receive Ready (RR): ACK
  - Reject: NACK
  - Receive No Ready (RNR): ACK with flow control
  - Selective Reject: selective repeat.
Figure 7.13  HDLC Two-Way Alternate Communication Using NRM. (a) REJ error recovery. (b) SREJ recovery. (c) Checkpoint recovery using RR. (d) Checkpoint recovery using 1 Frame.
Figure 7.14 Use of HDLC on Multipoint Link, Including Polling and Error Recovery (adapted from [CARL80]). © 1980 IEEE. Reprinted by permission.
Figure 7.15  An Example of Asynchronous Balanced Mode HDLC.
This is a modified version of figure in [CARL80].
Shading indicates direction of transmission. © 1980 IEEE.
Reprinted by permission.
DLC in the Internet

- PC dials up an Internet service provider’s router and act like a full-blown Internet host. Two protocols: SLIP and PPP.

- SLIP:
  - Send raw IP packets with a special flag type (0xC0) at the end for framing.
  - Character stuffing with (0xDB, 0xDC)
  - Newer version adds header compression (consecutive TCP, IP tends to have the same header)
Problems with SLIP

- No error detection or correction
- Support only IP, (a problem for Novell LANs).
- Must know the other’s IP in advance. No dynamic IP assignment is possible.
- No authentication
- Not an Internet Standard.
The Data Link Layer in the Internet

A home personal computer acting as an internet host.
PPP – Point-to-Point Protocol

- A DLC protocol to solve all the problems of SLIP
- Two protocols are used for connection negotiation:
  - LCP (Link Control Protocol) for PPP parameters set up.
  - NCP (Network Control Protocol) for network layer (IP) parameter set up (e.g., IP assignment and release)
PPP

- HDLC like but character-oriented
- Address and Control fields are default constant and can be omit after negotiation.
- Protocol: what kinds of packet types in the payload, such as LCP, NCP, IP, IPX, …
  - Multiprotocol framing mechanism
  - Provide error detection, option negotiation, header compression, and optionally, reliable transmission
The PPP full frame format for unnumbered mode operation.
A simplified phase diagram for bring a line up and down.
### PPP – Point to Point Protocol (3)

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure-request</td>
<td>I → R</td>
<td>List of proposed options and values</td>
</tr>
<tr>
<td>Configure-ack</td>
<td>I ← R</td>
<td>All options are accepted</td>
</tr>
<tr>
<td>Configure-nak</td>
<td>I ← R</td>
<td>Some options are not accepted</td>
</tr>
<tr>
<td>Configure-reject</td>
<td>I ← R</td>
<td>Some options are not negotiable</td>
</tr>
<tr>
<td>Terminate-request</td>
<td>I → R</td>
<td>Request to shut the line down</td>
</tr>
<tr>
<td>Terminate-ack</td>
<td>I ← R</td>
<td>OK, line shut down</td>
</tr>
<tr>
<td>Code-reject</td>
<td>I ← R</td>
<td>Unknown request received</td>
</tr>
<tr>
<td>Protocol-reject</td>
<td>I ← R</td>
<td>Unknown protocol requested</td>
</tr>
<tr>
<td>Echo-request</td>
<td>I → R</td>
<td>Please send this frame back</td>
</tr>
<tr>
<td>Echo-reply</td>
<td>I ← R</td>
<td>Here is the frame back</td>
</tr>
<tr>
<td>Discard-request</td>
<td>I → R</td>
<td>Just discard this frame (for testing)</td>
</tr>
</tbody>
</table>

The LCP frame types.