Layer 2: Data Link Layer

- Material covered: 3.1-3.2, 3.4 and 3.6
- Goal: to transmit error free frames efficiently over the physical link
  - between two adjacent machines
  - adjacent means machines connected by a physical channel
- Potential problems due to
  - communication circuits make errors occasionally
  - the transmission has a finite data rate and a non-zero propagation delay
Layer 2: Data Link Layer

- **Design issues:**
  - how to group the bits of the physical layer into frames? How big?
  - how do we detect an error in a frame?
  - how to provide **error control** to retransmit damaged or lost frames?
  - what delimits the end of a frame?
  - how to do **flow control** to prevent a fast sender from drowning a slow receiver?
  - for a broadcasting network, how to control access to the shared channel? (to be covered in chapter 4)

- **Examples:**
  - Internet data link protocols: PPP
  - ATM data link protocols
Services provided to the network layer

- **Three possible data link services to the network layer software**
  - unacknowledged connectionless service: e.g., most LANs
  - acknowledged connectionless service, e.g., on wireless channels
    - consider the delay to retransmit a long message vs. only to retransmit a short frame
    - providing acknowledgement at the data link layer is not a requirement but an optimization, since the transport layer can always do acknowledgement at the message level
    - the data link layer handles damaged, lost and duplicate frames
  - acknowledged connection-oriented service
    - a connection is established before any data are transferred
    - the data link layer guarantees that each frame sent is received in the right order

Fig. 3-2. Placement of the data link protocol.
Framing

- **Slice a raw bit stream up into discrete frames**
  - why? To apply error detection to a manageable unit of transmission

- **Frame Boundary - How do we know when a frame starts and ends? Four methods below:**
  - Character count: header indicates number of bytes
    - problem: what if the header is corrupt, can’t tell end of frame
  - Special starting and ending characters
    - ASCII: DLE STX … DLE STE
    - need to use **character stuffing** to handle embedded DLEs in data
      - send two DLEs to indicate a DLE character in data
  - Special bit patterns (or **flags**)
    - no restriction on # of bits in a data frame and # of bits to represent a character (i.e., no longer tied to ASCII)
    - 01111110 - indicates end of frame
    - need to use **bit stuffing** to send 01111110 embedded in data
      - insert 0 after 5 1’s
  - physical layer invalid bit patterns
    - e.g., if the physical layer encodes 1 by a high-low pair and 0 by low-high, then low-low or high-high can be used as delimiters
In practice, a combination of a character count along with one of the other methods is used for extra safety.

When a frame arrives, the count field is used to locate the end of a frame.

The frame is Ok if the delimiter is present at the correct position and the checksum is correct.

Fig. 3-3. A character stream. (a) Without errors. (b) With one error.

If the receiver loses track of where it is, all it has to do is to scan the input bit sequence for the flag sequence while it discards all the proceeding bits.

Fig. 3-5. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver’s memory after destuffing.
Other Data Link Layer Functions

- **Error Control**
  - the receiver can send a positive or negative ACK to the sender to let the sender know whether a frame was received correctly (i.e., not damaged)
  - what happens if the frame vanishes completely due to a noise?
    - The sender can start a timer when it sends a frame, so if the packet is lost, the sender can retransmit the packet again
  - what happens if the ACK is lost and the timer goes off?
    - A frame can be transmitted multiple times
  - can use sequence numbers to eliminate duplicate frames
  - this introduces overhead, but useful if probability of failure is high

- **Flow Control**
  - to provide rate matching between the sender and receiver
  - rules must be defined to let the sender know when it can send the next frame of the next set of frames, normally initiated by the receiver, e.g., credit-based flow control
Error Correcting Codes

- **Idea:** add redundant information to permit error recovery
  - this is the dual of data compression (remove redundancy)

- **Hamming distance (n)**
  - number of bit positions that differ in two words
  - key idea: need n single bit errors to go from one word to the other
  - to detect d errors, need a hamming distance of d + 1 from **any other valid word.**
  - to recover d errors, need a hamming distance of 2d + 1
    - any error of d bits is still closer to a unique correct word

- **Parity bit**
  - ensure that every word has an odd (or even) # of 1’s
  - this makes the Hamming distance equal to 2
  - permits detection of one 1-bit error
Error Codes (cont.)

- **Error Recovery**
  - Given \( m \) bits of data and \( r \) bits of error code and \( m + r = n \)
  - Want to correct any 1-bit error
  - There are \( n \) incorrect words one bit from each valid \( m \)-bit data message
    - so need \( n+1 \) words (of which \( n \) words are incorrect and 1 is correct) for each \( m \)-bit valid message
    - thus \( (n + 1) 2^m \leq 2^n \)
    - but \( n = m + r \) so \( (m + r + 1) \leq 2^r \)

m data bits \quad r \text{ redundant error-correcting bits}
Error Codes (cont.)

- **Hamming Error Correcting Code**
  - used to recover from any 1-bit error
  - all bits are numbered from left (starting from bit 1 at the left)
    - power of two positions contain parity bits (e.g., at bit positions 1, 2, 4, 8, etc.)
    - other positions contain data bits (at bit positions 3, 5, 6, 7, 9, 10, etc.)
  - each parity bit checks the parity of a group of bits
    - bit 1 checks the group (1,3,5,7,9,11, ...)
    - bit 2 checks the group (2,3,6,7,10,11, ...)
    - bit 4 checks the group (4,5,6,7,12, ...)
    - bit 8 checks the group (8,9,10,11,12, ...)
  - each bit is checked by all parity bits in its sum of power expansion
    - if parity bits 1, 2, and 8 are in error => bit 11 is incorrect
    - if only parity bit 8 is in error => parity bit 8 itself is incorrect
Hamming Code Example

A trick to use hamming code to correct burst errors

- the sender sends data by column rather than by row
  - all k bits of one column are sent before the next column
- the receiver reconstructs the m x k matrix (with m rows and k columns) and then applies the hamming code to each row
  - a burst error may introduce k errors into a single column but this will be corrected since each row can correct a 1-bit error
  - in effect, we use kr bits to check a block km bits long

<table>
<thead>
<tr>
<th>Char</th>
<th>ASCII</th>
<th>Hamming</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1001000</td>
<td>0011001000</td>
</tr>
<tr>
<td>a</td>
<td>1100001</td>
<td>10111001001</td>
</tr>
<tr>
<td>m</td>
<td>1101101</td>
<td>11101010101</td>
</tr>
<tr>
<td>I</td>
<td>1101001</td>
<td>01101011001</td>
</tr>
</tbody>
</table>

Bit 11 in error

01101011000 ?
Error Detection

- **Less bits are required**
  - if errors are infrequent, this works better
  - assume that re-transmission is possible

- **Cyclic Redundancy Codes (CRC)**
  - also known as polynomial code
    - treating a k+1-bit frame as a degree k polynomial with k+1 terms, ranging from $x^k$ to $x^0$ with coefficients of 0 and 1 only
    - for example, 110001 is a degree 5 polynomial $x^5 + x^4 + x^0$ with coefficients 1, 1, 0, 0, 0, and 1.
    - Polynomial arithmetic is done modulo 2 (exclusive OR): no carries for addition and no borrows for subtraction

```
  10011011
+ 11001010
  01010001
```

```
  11110000
- 10100110
  01010110
```
CRC

- The basic idea is that a checksum is appended to a frame in such a way that the polynomial represented by the checksummed frame is divisible by $G(x)$
  - $G(x)$ is a generator function of degree $r$ agreed upon by the sender and receiver
    - $G(x) = x^r + \ldots$
  - Let $M(x)$ be the frame and let $M'(x)$ be the $M(x)$ with $r$ 0’s appended to the end of it
  - divide $M'(x)$ into $G(x)$ and compute the remainder (which is always $r$ bits including the possible 0’s in front) using polynomial arithmetic
  - use the $r$-bit remainder as the $r$-bit CRC code
  - Let $T(x)$ be $M(x)$ with the $r$-bit CRC appended to the end of it
  - send $T(x)$ to the receiver as the checksummed frame
  - when the receiver gets the checksummed frame $T(x)$, it tries to divide it by $G(x)$
    - if no remainder $\Rightarrow$ no error
CRC Example

Frame : 1101011011
Generator: 10011
Message after appending 4 zero bits: 11010110000

110100001010
1001111010110000
000100000000
00010000000
00010000000
00010000000
00010000000
00010000000
00010000000
00010000000
10110
10011
01010
00000
10100
10011
01110
00000
1110

Transmitted frame: 110101101111110

Fig. 3-7. Calculation of the polynomial code checksum.
What kinds of errors can be detected by CRC?

- The error is can be represented in the form of \(E(x)\): each 1 bit in \(E(x)\) corresponds to a bit that has been inverted (that is, in error)
- We can use this representation because we use modulo-2 arithmetic to do the check sum calculation
  - one single-bit error: 1 bit in \(E(x)\) is a 1 and all others in \(E(x)\) are 0
    - \(E(x) = x^i\) where \(i\) is the bit position in error counting from the right of the frame received
  - two isolated single-bit errors: 2 separate 1 bits in \(E(x)\)
    - this is also called a double-bit error
    - \(E(x) = x^i + x^j\)
  - \(k\) single-bit errors: \(k\) separate 1 bits in \(E(x)\)
  - a burst error: \(E(x)\) contains an initial 1, a mixture of 0s and 1s and a final 1bit, with all other bits in \(E(x)\) being 0
    - \(E(x) = x^i (x^{r-1} + \ldots + x^0)\) for an \(r\)-bit burst error
  - the receiver computes \((T(x)+E(x))/G(x) = E(x)/G(x)\) because we know \(T(x)/G(x)=0\) for the original checksummed frame \(T(x)\)
CRC (cont.)

- It can be shown that a CRC can detect the following errors:
  - all single bit errors, as long as \( G(x) \) contains two or more terms: 
    \[ E(x) = x^i \] so as long as \( G(x) \) contains two or more terms, \( E(x) \) cannot divide \( G(x) \)
  - all double-bit errors: \( E(x) = x^i + x^j = x^i (x^{i-j} + 1) \), so as long as \( G(x) \) does not divide \( x^i \) (e.g., by containing the \( x^0 \) term) and does not divide \( x^k + 1 \) for a sufficiently large \( k \) (up to the maximum frame length), we will be able catch double errors
    - e.g., \( G(x) = x^{15} + x^{14} + 1 \) can detect one double-bit error since it contains \( x^0 \) and does not divide \( x^k + 1 \) for any \( k \) below 32,768
  - any odd number of single bit errors: \( E(x) \) cannot divide \( x+1 \), so if we make \( G(x) \) contain \( x+1 \) as a factor, then \( E(x)/G(x) \) cannot be zero
    - e.g., \( E(x) = x^5 + x^4 + x \) cannot divide \( x+1 \), so if we select \( G(x) = x^{15} + x^{14} + x + 1 \) then it will catch this kind of error
  - all burst errors less than or equal to \( r \) bits:
    - an \( r \)-bit burst error can be represented by \( E(x) = x^i (x^{r-1} + \ldots + x^0) \) where \( i \) indicates how far the \( r \)-bit burst error is located from the right-hand end of the received frame
    - if we make \( G(x) \) contain the \( x^0 \) term then it will not have \( x^i \) as a factor; since the degree of the parenthesized expression is less than \( r \), so \( E(x)/G(x) \) can never be zero
Several G(x)’s are standardized
- CRC-12 = \( x^{12} + x^{11} + x^3 + x^2 + x + 1 \)
- CRC-16 = \( x^{16} + x^{15} + x^2 + 1 \)
- CRC-CCITT = \( x^{16} + x^{12} + x^5 + 1 \)
  - all contain more than two terms
  - all does not divide \( x^k + 1 \) for a sufficiently large k
  - all have \( x+1 \) as a factor
  - all contain the \( x^0 \) term

16 bit CRC will catch
- all single and double bit errors
- all errors with an odd number of bits
- all burst errors of length less than 16
Information Flow between Network and Data Link Layers

- Recall that each layer will add its own header or trailer for its layer protocol.
- When the data link layer accepts a packet, it encapsulates the packet in a frame by adding a header and trailer to it.
- Whatever in the packet is of no concern to the data link layer, every bit of which must be delivered to the destination network layer.

Fig. 1-11. Example information flow supporting virtual communication in layer 5.
Data Link Protocols

- **Stop And Wait -- One frame at a time**
  - send a frame and wait for an ACK
  - what happens if ACK is lost due to noisy channel?
    - need sequence number to tell re-transmission from next packet
    - sender puts sequence number in the header -- how many bits are needed for the sequence number?
    - receiver puts the received frame’s sequence number in ACK
    - receiver updates its sequence number after receiving a frame

- **Sliding Window**
  - Use a n-bit field to allow the sequence number to go from 0 to \(2^n - 1\)
  - sender maintains a **sending window** indicating sequence numbers for which frame sent but not yet ACKed
  - receiver maintains a **receiving window** indicating sequence numbers for which it may accept.
  - sending and receiving windows do not have to be same size
One Bit Sliding Window - Stop and Wait

- Window size = 1 for both the sender and receiver
- stop and wait, so only 1-bit sequence number is needed
  - (seq, ack, frame) notation is used in this example
  - e.g., (0,1,B0) means a frame B0 has a sequence number 0 and also it piggybacks an ACK for a earlier packet received with sequence number 1
  - duplicates can occur under bad timing scenarios

Fig. 3-14. Two scenarios for protocol 4. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.
Pipelining in Sliding Window Protocols

- **How large should the sender’s window be?**
  - Let frame size = $l$ bits; channel data rate = $b$ bits/sec; round-trip propagation delay = $R$ sec. Then the time interval between the first message is sent and the first ACK returns is $R + l/b$.
  - Use pipelining to keep the channel busy during this interval.
    - make sender’s window large enough, e.g., $l = 1000b$; $b = 50kbps$; $l/b = 20$ msec; $R = 500$ msec => $W(sender) = 26$

- **How about if a frame in the middle is damaged or lost?**
  - **Go Back N**: the receiver simply discards all subsequent frames.
    - this in effect means $W(receiver) = 1$
  - **Selective Repeat**: the receiver stores some subsequent frames.
    - $W(receiver) > 1$ (so a large receiver buffer is needed)
    - any frame within the receiver window can be accepted and buffered until all the preceding ones have arrived
    - when the missing ones arrive, the receiver sends back an ACK with the highest sequence number in a sequence of frames.
Effect of Go Back N and Selective Repeat

Fig. 3-15. (a) Effect of an error when the receiver window size is 1. (b) Effect of an error when the receiver window size is large.
Go Back N Implementation (Protocol 5)

- \( W(\text{receiver}) = 1 \) and \( W(\text{sender}) = 0 \) to \( \text{MAX\_SEQ} = 2^n - 1 \) dynamically where \( n \) is the number of bits for the seq. #
  - no more than \( \text{MAX\_SEQ} \) unACK’ed frames outstanding at any time
  - this means that there are \( \text{MAX\_SEQ} + 1 \) distinct sequence #’s, from 0, 1, … to \( \text{MAX\_SEQ} \), e.g., \( \text{MAX\_SEQ} = 7 \) for a 3-bit seq. #
    - this is to prevent confusion about the first batch of 0-7 frames and the second batch of 0-7 frames, with an ACK on 7, e.g., the sender does not know if the ACK is for the 1st or 2nd batch; also if the ACK is lost the receiver will accept
  - sender must buffer transmitted frames until they are ACKed
  - sender must keep a timer for each outstanding frame
  - receiver will simply discard duplicates

Fig. 3-17. Simulation of multiple timers in software.
Selective Repeat Implementation (Protocol 6)

- **Number of buffers**
  - same way: buffer size = window size for both the sender and receiver

- **Piggyback**
  - after an in-sequence frame arrives, can start an auxiliary timer; if no reverse traffic exists before the timer expires, a separate ACK is sent

- **Error handling**
  - when the receiver gets a frame out-of-sequence, it can send a negative ACK (called a NAK) to resynchronize the sender
    - useful when the ACK timer is loose

- **What should be the maximum window size for the sender and the receiver?**
  - the sequence number of an arriving frame is checked to see if it falls within the window: if yes and not a duplicate, then accept the frame
  - Can it be $W(\text{sender}) = 0$ to $\text{MAX\_SEQ}$ dynamically and $W(\text{receiver}) = \text{MAX\_SEQ}$ always? No - let’s see an example
Problems with Select Repeat Sliding Protocols

- **Example:** 0 - 6 sent; received and ACKed by the receiver, but the acknowledgements got lost
  - sender window valid sequence numbers are now 0,1,2,3,4,5,6
  - receiver window valid sequence numbers are now 7,0,1,2,3,4,5
  - the ACK is lost
  - sender times out and resends a frame with sequence number 0
    - is it a duplicate or a new frame?
    - receiver sees that 0 is within its window, so it accepts it as a new frame and sends a piggyback ACK for the previous 0-6 frames received
      - the sender now sends 7,0,1,2,3,4,5 to the receiver
        - new 0 will be discarded as a duplicate, and
        - new 7 and old 0 will be passed to the network layer

- **The problem here is that the receiver cannot distinguish duplicates from new frames**
Select Repeat Sliding Protocols (cont.)

- Solution: not allowing the new range of valid sequence numbers overlapping the old ones
- 4 bits for sequence numbers: only 8 unacknowledged frames should be outstanding at any time, e.g., #0-7 for the old range and #8-15 for the new range
  - the maximum window size should be at most half the range of the sequence number
  - in the previous example, when the receiver gets frame #0 it can immediately tell that the frame is a duplicate

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.
Exercises on Sliding Window Protocols

- **Q**: frames of 1000 bits are sent over a 1-Mbps satellite channel. Acknowledgements are always piggybacked onto data frames in the reverse traffic. The headers are very short. Three-bit sequence numbers are used. What is the maximum achievable channel utilization for (a) stop and wait (b) go back N (protocol 5) (c) selective repeat (protocol 6)?

- **Ans**: to send one frame, it takes $1000/1,000,000 = 1$ msec.
  - for stop and wait: the max. channel utilization is $1/(1+270+1+270)$
  - for go back N: $W$(sender)$=7$, so the max. channel utilization is $7/(1+270+1+270)$
  - for selective repeat: $W$(sender)$=4$, so the answer is $4/542=0.74%$
Q: An error-free 64kbps satellite channel is used to send 512-byte frames in one direction, with very short ACKs coming back the other way. What is the max. throughput for a window size 7?

Ans: the ACK is very short, so we can assume it takes zero time to transmit (see figure below). A 512-byte frame occupies the channel for 512*8/64,000 = 64 msec. The round trip propagation delay is 270*2=540 msec, so we need a window of (540+64)/64 = 10 frames to keep the channel busy

- for window size = 7, max. throughput = (7*512*8)/(540+64)
Example Data Link Protocols

- **HDLC - High Level Data Link Control**
  - used in X.25 (a connection-oriented virtual-circuit network)
  - on idle channel, the flag sequence (01111110) is transmitted continuously - **bit stuffing** is used for payload data
  - bit oriented (not character oriented)
  - a sliding window protocol with 8 sequence numbers being used
    - up to 7 unACKed frames may be outstanding at any time
  - an 8-bit control field:
    - with the 1st and 2nd bits indicating the frame type
    - contains flow control information such as seq #, ACK, NAK, etc.
    - can also allow the receiver to buffer out-of-sequence frames, i.e., if the receiver has a receiving window greater than 1

```
8 8  8 variable 16  8
flag address control payload checksum flag
01111110          01111110
```

For lines with multiple terminals using CRC-CCITT
Internet Point-to-Point Data Link Protocols

- Data link protocols for LANs (not point-to-point): chapter 4
- Data link protocols for Internet point-to-point lines: here
  - WAN subnet lines (trunks) that connect routers
  - phone lines that connect home PCs with Internet Provider routers
    - e.g., a dial-up home PC-router connection for Internet access via phone line is like a temporary leased PC-router line
- SLIP (Serial Line IP)
  - handle raw IP packets only
  - use 0x0C0 for framing
  - use character stuffing
  - does not do error control
  - can compress the header to improve the speed
  - both ends must each have a permanent IP address
    - a problem for Provider
Internet Point-to-Point Data Link Protocols

- **PPP - Point-to-Point Protocol** (an Internet Standard)
  - A multiprotocol framing mechanism based on character stuffing
    - encapsulates different protocol packets in the payload field, e.g.,
      - network layer data packets: IP, IPX, AppleTalk, etc.
      - control packets (between two ends): LCP and NCP
  - Link Control Protocol (LCP)
    - allows options to be negotiated such as the payload size for data frames, authentication, network protocol to use, compression, etc.
    - brings the physical line up, tests it and brings it down when done
  - Network Control Protocol (NCP)
    - negotiates network-layer options specific to the protocol selected
    - There is one NCP for each protocol supported, e.g., the main thing "the NCP for IP" does is IP dynamic address assignment

<table>
<thead>
<tr>
<th>flag</th>
<th>Address</th>
<th>control</th>
<th>protocol</th>
<th>payload</th>
<th>checksum</th>
<th>flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111110</td>
<td>11111111</td>
<td>00000011</td>
<td></td>
<td></td>
<td></td>
<td>01111110</td>
</tr>
</tbody>
</table>

All stations Default is no seq or ACK LCP, NCP, IP, IPX in payload?
PPP: An Internet Standard Data Link Protocol

- How does it work for a home PC via modem dial up?
  - once the router-PC modem physical layer connection is established, the home PC sends a series of LCP packets in the payload field of PPP frames to the router to select the PPP options to be used
  - then, a series of NCP packets are sent to configure the network layer

  - if the PC wants to run TCP/IP, then the Provider’s router PPP will execute **NCP for IP** which will dynamically assign an IP to the home PC and also configure the router ready for IP packets
  - if the PC wants to run Novell’s IPX instead then PPP will execute **NCP for IPX** instead

---

**Fig. 3-28.** A simplified phase diagram for bringing a line up and down.
ATM Data Link Protocol

- ATM physical layer covers OSI physical and data link layers
- The transmission convergence (TC) sublayer in the physical layer corresponds to the OSI data link layer
  - On output, TC takes a sequence of cells, computes a CRC to each cell header, converts the result to a bit stream and matches the bit stream’s data rate to the rate of the underlying physical system
  - If a synchronous medium is used, idle cells need to be invented
  - If a SONET STM-1 line is used, the true user data rate is only \( \frac{260}{270} \times 155.52 \text{ Mbps} = 149.76 \text{ Mbps} \), so TC at an ATM source needs to put an OAM (operation and maintenance) cell as every 27th cell; OAM cells are given to the ATM layer in the ATM switches for exchanging control information and are not sent out any more
  - Mapping from ATM onto the payload field is carrier specific, e.g., T1, T3, FDDI, etc.; standards exist for different transmission carriers
  - On input, TC takes an incoming bit stream, locates the cell boundaries, verifies the headers, processes the filler (OAM or idle) cells and passes the data cells up to the ATM layer
ATM Data Link Protocol (cont.)

- **Error control on the 5-byte header**
  - Use the rightmost 1-byte CRC (called **Header Error Control**) to checksum the remaining 4 bytes of VC and control information in the header
  - the polynomial $x^8 + x^2 + x + 1$ is used to generate the checksum
  - ATM is designed for use over fiber and fiber is highly reliable, so this decision to checksum only the header is made to reduce the probability of cells being delivered incorrectly due to a header error

- **No flow control -- it is up to the higher layer**

- **No frame flags used, so how to locate cell boundaries?**
  - Method 1 based on the **HEC Algorithm**: use a 40-bit shift register to check for a valid 8-bit HEC bit by bit
    - the shift register contains a valid cell header if the 8-bit HEC at the end of the 40-bit is valid for the 32 bits ahead of it
    - not always correct -- since the HEC is only 8 bits long, the probability of finding an incorrect HEC by accident is $1/2^8$ assuming random bits, which is a moderately large value.
How to locate cell boundaries in ATM?

- **Method 2: use a finite-state machine to improve accuracy**
  - HUNT -- bit by bit check by using the HEC algorithm (loss of synchronization state)
  - PRESYNCH -- cell by cell (resynchronization state)
  - SYNCH -- cell by cell (synchronization state)
  - Initial state is HUNT
    - HUNT -> PRESYNCH when one cell with checksum matches
    - PRESYNCH -> HUNT when an incorrect HEC is detected
    - PRESYNCH -> SYNCH after $\delta$ consecutive correct HECs are detected
      - the probability of getting into SYNCH by mistake is now $1/2^{8\delta}$
        - can be made arbitrarily small by choosing a large enough $\delta$
    - SYNCH -> HUNT after $\alpha$ consecutive incorrect HECs are detected

- **Detecting loss of synchronization**
  - one bad cell is probably in error: just discard the cell
  - many bad cells is likely a slip (loss of sync)
  - if $\alpha$ bad cells are seen in a row, switch from SYNCH to HUNT mode
How to locate cell boundaries in ATM?

![Diagram of cell delineation heuristic]

Fig. 3-30. The cell delineation heuristic.

- **Malicious users trying to garble the data?**
  - By inserting a data pattern into the **payload** field of consecutive cells that imitates the HEC algorithm

- **Solution:** the payload bits can be scrambled on transmission and descrambled on reception
Q: Suppose that $\alpha=5$ and a per-bit error rate is $10^{-5}$. Once the system is synchronized (in the SYNCH state), how long will it remain so despite occasional HEC errors? Assume the line is running OC-3 at 155 Mbps.

Ans: the probability that a cell is correct is $(1-10^{-5})^{40}$, so the probability of an incorrect HEC is $1 - (1-10^{-5})^{40}$ or about $4 \times 10^{-4}$. The probability of having $\alpha=5$ consecutive bad HECs is therefore $(4 \times 10^{-4})^5$ or about $10^{-17}$. In other words, to lose synchronization will take about $10^{17}$ cells or about $10^{17} \times 53 \times 8/155\text{Mbps} = 870$ years assuming the line is running OC-3 at 155Mbps.