Hamming codes can also correct burst of errors by using **iterleaving**

To correct bursts of length \( k \), transmit groups of \( k \) codewords by columns → the burst will affect at most 1 bit in each codeword.

Sounds nice … and it really is! However, what is a **downside** of this method?
Error Correction – Convolutional codes

Operates on a stream of bits, keeping internal state

- Internal state = \{S1 S2 ... S6\}
- Output stream is a function of all preceding input bits
- Sums are XOR

Popular NASA binary convolutional code, now used in 802.11
Error Correction – Convolutional codes

Rate: \( r = \frac{1}{2} \)
- Each input bit generates two output bits
Constraint length: \( k = 7 \)
- If a given bit is at the input, it takes 7 CLK cycles until it is out (i.e. until it does not affect the output anymore)
QUIZ: Convolutional codes

The states S1 – S6 are all initialized with zeroes. Assume the following sequence of bits (right-to-left): 1101. Show the two output bit streams.
Convolutional codes – how about decoding?

**Viterbi** algorithm (not covered here)

- See Sec. 3.6.6 of Computer Architecture text (Null & Lobur)
- **Estimate** what sequence of inputs is *more likely* to have *generated* the outputs
- Allows soft-decision decoding, i.e. the individual bits are assigned *probabilities* of being 0 or 1, and the final decision is made only after examining an entire sequence
- By delaying the decision, we can exploit correlation info!
- Error correction is achieved naturally: the correct sequence is the most likely one!
Convolutional codes – conclusion

- A convolutional code is decoded by finding the sequence of input bits that is the most likely to have been transmitted!
- The decision is not done for each bit individually, but, through to the use of probability, we can take into account the context.
- This allows correction of bursts of errors!
Recap. of error-correcting codes

Hamming

Convolutional

Reed-Solomon

Low-Density Parity Check (LDPC)
3.2.2 Error-Detecting Codes

Old method: **parity bit(s)**

Assume one parity bit per codeword:

- What is the Hamming distance of such a code?
- Accordingly, how many errors can such a code \[\text{Detect}\,?\]
  - Correct?

What if there are multiple errors in the codeword, e.g. a **burst** of consecutive errors?

- It all depends on whether the length of the burst is odd or even!
- Detection probability is 0.5 (!)
Problem 8: To provide more reliability than a single parity bit, a code uses one parity bit for all even-numbered bits, and one for all odd-numbered bits.

What is the Hamming distance of this code?

How many errors can it reliably

- Detect?
- Correct?
Problem 8: To provide more reliability than a single parity bit, a code uses one parity bit for all even-numbered bits, and one for all odd-numbered bits.

What is the Hamming distance of this code?

2

How many errors can it reliably

• Detect?
• Correct?

To detect $d$ errors, a code with Hamming distance $d+1$ (or higher) is needed.

To correct $c$ errors, a code with Hamming distance $2\cdot c + 1$ (or higher) is needed.
Problem 8: To provide more reliability than a single parity bit, a code uses one parity bit for all even-numbered bits, and one for all odd-numbered bits.

What is the Hamming distance of this code? 2

How many errors can it reliably
• Detect? 1
• Correct? 0
Error-Detecting: parity bit(s)

How to deal with bursts of errors?
Use interleaving, as in Hamming codes!

Not all bits are flipped in a burst!
Error Detection: Checksums

They are a variation of interleaved parity bits

Checksum treats data as $N$-bit words and adds $N$ check bits at the end of the entire frame
- The bits are the modulo $2^N$ sum of the words

Example: Internet 16-bit 1’s complement checksum
- Use 4 bits for this quick example (from L to R):

1110 0110 0111 $\rightarrow$
Solution

Add two words at a time, any carry-out from MSB propagates as carry-in for the LSB.

```
1110 0110 0111
```

```
  1110
+ 0110
------
  0100
```

```
  0100
+ 0111
------
  1100  —— This is the checksum!
```
Error Detection: Checksums

Checksum has improved error detection over the simple parity bits

- Reliably detects bursts up to \( N \) errors
- Detects random errors with probability \( 1 - 2^N \)
- Vulnerable to systematic errors, e.g. added zeroes
Error-Detecting Codes

Newer method: polynomial codes (a.k.a. CRC)

A sequence of k bits is interpreted as binary polynomial of degree k-1

Polynomial arithmetic reduces to XOR. (No carries or borrows!)

Generator polynomial $G(x)$ of degree r, agreed upon by sender and receiver

- Both MSB and LSB of $G(x)$ must be 1

Algorithm for sender (see next slide):

- $M(x) \rightarrow 2^r M(x)$ by appending $r$ zeroes

- Checksum is remainder of division of $2^r M(x)$ by $G(x)$

- Add (subtract) checksum to $2^r M(x) \rightarrow$ checksummed message is divisible by $G(x)$

Algorithm for receiver: divide message by $G(x)$ and check if remainder is 0.
Unlike regular division, only the MSBs are checked to decide if quotient bit is 0 or 1. Here the quotient bit is 1, even though G is larger!
QUIZ: CRC

Find the CRC bits

Frame: 10011101
Generator: 1011

Frame with 3 zero bits: 10011101000

Long division (with XOR):

\[
\begin{array}{l}
1011 \\
\underline{10011101} \\overbrace{000} \\
\end{array}
\]
QUIZ: CRC

First step

\[
\begin{array}{c}
1011 \big| 100111010000 \\
\underline{1011} \\
0101
\end{array}
\]

Continue …
Frame: 10011101
Generator: 1011
Frame with 3 zero bits: 1001101000

Long division (with XOR):

```
  101 00001
/ 1011 [10011101 000]
  1011
  0101
  0000
  1011
  1011
  0000
  0000
  0001
  0000
  0010
  0000
  0100
  0000
  1000
  1011
```

Remainder: 011

Frame with CRC bits: 10011101011
CRC conclusion

Types of errors reliably detected by any CRC:

• Single-bit
• Two isolated single-bit
• Any odd number of single-bits
• All bursts of length less than \( r \)
• The probability of a burst of length \( r+1 \) to go undetected is \((1/2)^{r-1}\)

Can be easily implemented in the hardware, with only XOR gates and shift registers.

The IP protocol (L3) uses a CRC32:

\[ x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1 \]
Detection vs. Correction

- Detection (coupled w/retransmission) is better at low error rates
- Correction is better at high error rates
QUIZ: Error handling

List:

• The 2 error-\textit{correction} codes we studied:
  •
  •

• The 3 error-\textit{detection} codes we studied:
  •
  •
  •
3.3 Elementary Data Link Protocols

- Link layer environment
- Utopian Simplex Protocol
- Stop-and-Wait Protocol for Error-free channel
- Stop-and-Wait Protocol for Noisy channel
Link layer (L2) environment

Commonly implemented as NICs and OS drivers; network layer (IP) is often OS software
<table>
<thead>
<tr>
<th>Group</th>
<th>Library Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network layer</td>
<td>from_network_layer(&amp;packet) from_network_layer(&amp;packet) to_network_layer(&amp;packet)</td>
<td>Take a packet from network layer to send</td>
</tr>
<tr>
<td></td>
<td>enable_network_layer() disable_network_layer()</td>
<td>Deliver a received packet to network layer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Let network cause “ready” events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prevent network “ready” events</td>
</tr>
<tr>
<td>Physical layer</td>
<td>from_physical_layer(&amp;frame) from_physical_layer(&amp;frame) to_physical_layer(&amp;frame)</td>
<td>Get an incoming frame from physical layer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pass an outgoing frame to physical layer</td>
</tr>
<tr>
<td>Events &amp; timers</td>
<td>wait_for_event(&amp;event) start_timer(seq_nr) stop_timer(seq_nr) start_ack_timer() stop_ack_timer()</td>
<td>Wait for a packet / frame / timer event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start a countdown timer running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stop a countdown timer from running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start the ACK countdown timer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stop the ACK countdown timer</td>
</tr>
</tbody>
</table>

Link layer protocol implementations use library functions

See C code in `protocol.h` for more details (p.217)
P1: Utopian Simplex Protocol

A simple and optimistic protocol to get us started:

- Assumes: no errors, and receiver is as fast as sender
- Data transfer is one way, from Sender to Receiver (a.k.a. Simplex)
- The only possible event is the arrival of an undamaged frame

```c
void sender1(void)
{
  frame s;
  packet buffer;

  while (true) {
    from_network_layer(&buffer);
    s.info = buffer;
    to_physical_layer(&s);
  }
}

void receiver1(void)
{
  frame r;
  event_type event;

  while (true) {
    wait_for_event(&event);
    from_physical_layer(&r);
    to_network_layer(&r.info);
  }
}
```

Sender loops blasting frames  Receiver loops eating frames
P2: Simplex Stop-and-Wait for Error-free channel

This protocol ensures sender can’t outpace receiver:
- Receiver returns a dummy frame (ACK, s in the code) when ready
- Only one frame out at a time – called stop-and-wait
- We added flow control!

```c
void sender2(void)
{
    frame s;
    packet buffer;
    event_type event;

    while (true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
        wait_for_event(&event);
    }
}
```

Sender waits to for ack after passing frame to physical layer

```c
void receiver2(void)
{
    frame r, s;
    event_type event;
    while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```

Receiver sends ack after passing frame to network layer
P3: Simplex Stop-and-Wait Protocol for Noisy Channel

The frames now include an error-detection (but not correction!) mechanism.

Accordingly, there are 3 types of events possible:
• frame arrives OK
• checksum error
• timer timeout

First stab: Use a timer for each data frame sent. Receiver sends back ACK if frame was OK. If timer times out, sender retransmits that frame.

• How can this mechanism fail? (Think about the ACK…)
P3: Simplex Stop-and-Wait Protocol for Noisy Channel

If ACK is lost, sender will retransmit, but receiver has no way of knowing it’s a duplicate!

Solution: Include sequence numbers.

This mechanism is called positive acknowledgement w/retransmission (PAR), a.k.a. automatic repeat request (ARQ).

Both sender and receiver have memory in the wait state:

sender: \texttt{next\_frame\_to\_send}
receiver: \texttt{frame\_expected}

For stop-and-wait, 2 sequence numbers (1 bit) are sufficient: 0, 1, 0, 1, 0, …
P3: Simplex Stop-and-Wait Protocol for Noisy Channel

Sender loop:

Send frame (or retransmission)
Set timer for retransmission
Wait for ack or timeout

If a good sequence number,
then set up for the next frame,
else the old frame will be retransmitted

```c
void sender3(void) {
    seq nr next_frame_to_send;
    frame s;
    packet buffer;
    event_type event;

    next_frame_to_send = 0;
    from_network_layer(&buffer);
    while (true) {
        s.info = buffer;
        s.seq = next_frame_to_send;
        to_physical_layer(&s);
        start_timer(s.seq);
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&s);
            if (s.ack == next_frame_to_send) {
                stop_timer(s.ack);
                from_network_layer(&buffer);
            }
            inc(next_frame_to_send);
        }
    }
}
```

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
P3: Simplex Stop-and-Wait Protocol for Noisy Channel

Receiver loop:

Wait for a frame

If it’s new then take it and advance expected frame

Ack current frame

```c
void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;
    frame_expected = 0;
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            s.ack = 1 - frame_expected;
            to_physical_layer(&s);
        }
    }
}
```
FSM for Stop and Wait Protocol

Transmitter

- Send Pkt 0
  - Ack 1 Received
  - Timeout or Ack 1 Received
  - Pkt 0 transmitted

- Wait Ack 1
  - Timeout or Ack 0 Received
  - Pkt 1 transmitted

- Wait Ack 0
  - Ack 0 Received

- Send Pkt 1

Receiver

- Wait Pkt 0
  - Pkt 0 received
  - Send Ack 0

- Wait Pkt 1
  - Pkt 1 received
  - Send Ack 1

Source: http://www.eng.ucy.ac.cy/gellinas/Lecture5.pdf

Not in our text
What is the big problem of stop-and-wait protocols?

Hint: Think of very long lines …
QUIZ: Simplex

What is the meaning of a simplex communication protocol?
QUIZ: Simplex

Why did we call P2 and P3 *simplex*?
Doesn’t the receiver send back ACK frames?!?
3.4 Sliding Window Protocols
Connection is now **Duplex**

Idea: **piggybacking** → when a data frame arrives, instead of immediately sending a separate ACK frame, the receiver waits until L3 passes it the next pkt. → acknowledgement is attached to the outgoing data frame, using the ACK field in the header.

Advantage: better use of bandwidth
Complication: how long to wait for a pkt. to piggyback?

Worst-case: if it has no data to transmit, the receiver still has to generate a separate ACK frame (otherwise the sender will re-transmit, assuming the frame did not make it).
How do the sender and receiver know whether a given incoming frame is data or ACK?

Look at the protocol definitions in `protocol.h` (p.217):

```c
#define MAX_PKT 1024  /* determines packet size in bytes */
typedef enum {false, true} boolean; /* boolean type */
typedef unsigned int seq_nr;      /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;  /* packet definition */
typedef enum {data, ack, nak} frame_kind; /* frame_kind definition */

typedef struct {
  frame_kind kind;
  seq_nr seq;    /* frames are transported in this layer */
  seq_nr ack;    /* what kind of frame is it? */
  packet info;   /* sequence number */
  /* acknowledgement number */
  /* the network layer packet */
} frame;
```
P4: 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.

Notes:
- For window of size 1, frames can only be Rx-ed in order, but for larger sizes they can be out of order.
- Rx window is fixed-size (always 1 here), but Tx window is not (0 or 1 above).
Wait a second – isn’t this the old Stop-and-Wait??
P4: Sliding Window of size 1, with a 3-bit sequence number

Wait a second – isn’t this the old Stop-and-Wait??

Yes, a window of max. size one means Stop-and-Wait!

Frames Tx-ed, but not yet ACK-ed. Must store in case it will need re-Tx.

Frames it may accept. Anything else is silently discarded.
Sliding Window Protocols

QUIZ: Where are the sequence numbers defined in `protocol.h` (p.217)?

```c
#define MAX_PKT 1024

typedef enum {false, true} boolean;
typedef unsigned int seq_nr;
typedef struct {unsigned char data[MAX_PKT];} packet;    /* packet definition */
typedef enum {data, ack, nak} frame_kind;

/* frame_kind definition */

typedef struct {
    frame_kind kind;
    seq_nr seq;
    seq_nr ack;
    packet info;
} frame;

/* frames are transported in this layer */
/* what kind of frame is it? */
/* sequence number */
/* acknowledgement number */
/* the network layer packet */
```
Sliding Window Protocols

Stop-and-wait (w=1) is inefficient for long links.
Larger windows enable pipelining for efficient use of Bw.
To make use of the larger sequence numbers, the windows must be open wider → w > 1

Advantage: even better use of bandwidth, since one ACK (piggybacked or not) can confirm multiple frames

Disadvantages:
• To synchronize sequence # of sender and receiver, a connection has to be established.
• Since multiple transmitted frames can be unacknowledged, the sender needs to store them, in case they need retransmission.
Sliding Window Protocols

To use sequence numbers, the windows must be open wider $\rightarrow w > 1$

How large should $w$ be?

Best window size $w$ depends on the \textbf{bandwidth-delay product} (BD)

$BD = \text{(link bandwidth)} \times \text{(one-way propagation delay)}$

QUIZ: What units does BD have?
BD product

BD = (link bandwidth) x (one-way propagation delay)

Unit: bits (or Bytes, or frames)

Intuitive meaning of BD product: how much info is in the pipeline!

QUIZ: What is BD for a 50km OC-3 link?
Sliding Window Protocols

BD = (link bandwidth) x (one-way propagation delay)

If we measure BD in frames, \( w \geq \text{floor}(2 \cdot BD)+1 \) will ensure maximum link utilization, because the ACK frame arrives back before the sender reaches the end of the TX window.

Link utilization \( \rho \leq \frac{w}{(2BD+1)} \)

Note that this is an upper bound! (If \( w \) increases, we still cannot have utilization > 1 ...)
If we measure BD in frames, \( w \geq 2 \cdot BD + 1 \) will ensure maximum link utilization.

Link utilization \( \rho \leq w / (2BD+1) \)

A. What is the optimal \( w \)?

B. What is the maximum utilization for the 50km OC-3 link, for a frame size of 1000 Bytes and window size 8 frames?
Nota bene: Sequence numbers are not a panacea – even P4 can be “derailed” in certain scenarios!

Read the “slow start” and other P4 scenarios on p.231 of our text (Figure 3-17).
More advanced sliding window protocols.

Pipelining can be implemented with different choices for error control and buffering.

We consider:

1. Go-Back-N
2. Selective Repeat
P5: Go-Back-N

Receiver only accepts/acks frames that arrive in order:
- Discards frames that follow a missing/errored frame
- Sender times out and resends all outstanding frames
- The example below assumes receiver does not have any data frames to pyggyback ACKs
Tradeoff made for Go-Back-N:

- Simple strategy for receiver: it needs to deal with only 1 frame at a time
- Wastes link bandwidth for errors with large windows, since entire window is retransmitted

FYI: See C code in text, pp. 236-7
QUIZ: Go-Back-N

What is the sender’s window size $w$ in this scenario?

What is the receiver’s window size $w$?
QUIZ: Go-Back-N

What is the sender’s window size $w$ in this scenario? 7

What is the receiver’s window size $w$? 1
Draw the timeline of the following scenario: sender’s window is $w = 5$ (timeout interval is the time needed to Tx 5 frames), and frame 3 is in error. Sequence numbers are represented on 3 bits (0 through 7).
Go-Back-N

Can the number of sequence numbers $s$ be equal to the window size $\rightarrow w = s$ ?

No! Consider this scenario:

- $w = s = 8$, so sequence numbers are 0-7
- Sender has sent frames 0-7 and receives ACK for 7
- Sender sends new frames 0-7, but 0 is garbled
- Receiver discards 0 and all subsequent
- On the next piggybacked ACK, receiver still acknowledges the 7 from the first batch, but sender thinks it’s the 7 from the second batch!
Go-Back-N

Can the number of sequence numbers $s$ be equal to the window size $\rightarrow w = s$ ?

Conclusion: $w \leq s - 1 = \text{MAX_SEQ}$
QUIZ: BD product and Go-Back-N

- BD = (link bandwidth) x (one-way propagation delay)
- If we measure BD in frames, maximum link utilization will be achieved if \( w \geq 2BD + 1 \)
- To avoid errors in Go-Back-N, we need \( w \leq s - 1 = \text{MAX}_\text{SEQ} \)

We transmit 100-Byte packets over a T3 line that is 40 miles long. If we use a Go-Back-N protocol, what is the minimum length \( s \) of the sequence that will ensure maximum link utilization and no errors?
- How many bits are needed for the sequence numbers?
Solution

c₀ = speed of light in vacuum = 186,000 miles per second

c_{Cu} = speed of light in copper (Cat 5) = 0.77 \cdot c₀ = 143,220 \text{ mi/s}

BD = 44.736 \text{ Mbps} \cdot \frac{40}{143,220} \text{ mi/s} = 12,494.3 \text{ bits}

BD in frames = \frac{12,494.3}{100 \cdot 8} = 15.6 \text{ frames}

w_{min} = \text{floor}(2 \cdot BD) + 1 = 31 + 1 = 32

To avoid errors in Go-Back-N, we need \( w \leq s - 1 \)

s_{min} = w_{min} + 1 = 33

MAX_SEQ = s - 1 = 32

The range of sequence numbers is 0 \ldots 32 \ (33 \text{ numbers})

\log_2(33) = 5.04 \rightarrow \text{round up (why?)} \text{ to } 6 \text{ bits}
P6: Selective Repeat

Receiver accepts frames anywhere in the receive window

- **Cumulative Ack** indicates highest in-order frame.
- **NAK** (Negative AcK) causes sender retransmission of a missing frame before a timeout occurs.
QUIZ: BD

What does BD stand for?

What does it mean, intuitively?
QUIZ: BD

What does BD stand for?
Bandwidth-delay product

What does it mean, intuitively?
How much information (bits, Bytes, frames) is in transit on a fully-loaded communication link (one way).
Q: So what is a sliding window?
A: The set of (sequence) numbers that the sender/receiver is permitted to send/receive at a given moment in time.

Note: Sender and receiver do not need to have the same window size!
3.5 Example Data Link Protocols

- PPP (Point-to-Point Protocol)
- Packet over SONET
- ADSL (Asymmetric Digital Subscriber Loop)
- ATM (Asynchronous Transfer Mode)

Note: The examples given here are for WAN/MAN networks, which use point-to-point (ptp) connections.
The next chapter will present the LAN Data Link layer, which uses broadcast connections.
PPP = Point-to-Point Protocol

Two main uses for p-t-p connection in the Internet:

- A home PC connected to the Internet
- Routers connected in a WAN (next slide)
PPP – L2 in the Internet

Routers connected in the core of a WAN (a.k.a. subnet).
PPP – Point to Point Protocol
(RFC 1661, 1662, 1663 etc.)

The PPP full frame format for unnumbered (unreliable) mode of operation

This is the standard mode, specified in RFC 1661 (the framing is detailed in 1662), and distinguished by Protocol = 0x03.

RFC 1663 specifies the numbered mode, which yields reliable communication.

3 main functions:
• Framing (including error detection).
• Connection establishment (LCP = Link Ctrl. Protocol)
• Negotiation of L3 options (NCP = Network Ctrl. Protocol)
PPP is the successor to the older protocol HDLC.

PPP is a **byte-oriented** Protocol.

Framing uses a flag (0x7E) and **byte stuffing** with ESC = 0x7D.

Address is always 11111111 (RFC 1662: “Any frame with unrecognized address should be discarded”).

Protocol fields starting with 0 denote an L3 protocol (e.g. IP).

Protocol fields starting with 1 denote LCP or other control prot.

Payload is always a multiple of 8 bits, i.e. byte-oriented!
Header compression:

- Since *Address* and *Control* fields are constant, RFC 1661 provides a compression option (in compressed mode they are not Tx-ed).
- *Protocol* specifies what L3 packet is being encapsulated as *Payload*. An option permits to reduce it from 2 to 1 byte.

*Payload* has variable length, up to some negotiated maximum. If it was not negotiated → default = 1500B. May be padded.

*Checksum* is normally 4 byte, but 2 byte (HDLC-style) is an option.
3.2. Modification of the Basic Frame

- The Link Control Protocol can negotiate modifications to the standard HDLC-like frame structure. However, modified frames will always be clearly distinguishable from standard frames.

- Address-and-Control-Field-Compression

- When using the standard HDLC-like framing, the Address and Control fields contain the hexadecimal values 0xff and 0x03 respectively. When other Address or Control field values are in use, Address- and-Control-Field-Compression MUST NOT be negotiated.

- On transmission, compressed Address and Control fields are simply omitted.

- On reception, the Address and Control fields are decompressed by examining the first two octets. If they contain the values 0xff and 0x03, they are assumed to be the Address and Control fields. If not, it is assumed that the fields were compressed and were not transmitted.

- By definition, the first octet of a two octet Protocol field will never be 0xff (since it is not even). The Protocol field value 0x00ff is not allowed (reserved) to avoid ambiguity when Protocol-Field-Compression is enabled and the first Information field octet is 0x03.
PPP connection life-cycle

LCP negotiation
- Carrier detected
- Both sides agree on options
- Authentication successful

Data Tx takes place
- E.g. getting a temporary IP address

States:
- Establish
- Authenticate
- Open
- Network
- Terminate
- Dead

Transitions:
- Failed
- Done
- NCP configuration
Packet over SONET is the method used to carry IP packets over SONET optical fiber links

- Uses PPP (Point-to-Point Protocol) for framing

Protocol stacks

PPP frames may be split over SONET payloads
Packet-over-ADSL

Widely used for broadband Internet over local loops

- ADSL runs from modem (customer) to DSLAM (ISP)
- IP packets are sent over PPP and AAL5/ATM
What is ATM?

ATM was a major technology in the 1990s that was hyped to win in the convergence of the Internet and telecoms, but IP won instead.

- **Pro:** The short, fixed-size cells give flexibility: can mix voice and data without having the voice wait for a whole data packet
- **Con:** High overhead

ATM is now used only in niches such as ADSL and WAN links.

**Structure of an ATM cell** (source: http://www.gl.com/lightspeed1000-atm-analyzer.html)
Packet-over-ADSL

PPP data is sent in AAL5 frames over ATM cells:

- ATM is a link layer that uses short, fixed-size cells (53 bytes); each cell has a virtual circuit identifier.
- AAL5 (ATM Adaptation Layer 5) is a protocol that allows to break the upper layer’s payload into 48-byte chunks that can fit inside an ATM cell (segmentation and reassembly).
- PPP frame is converted to an AAL5 frame using PPPoA (RFC 2364).

![AAL5-diagram.png]

The AAL5 frame is then divided into 48-byte cells, each of which goes into one ATM cell with 5 header bytes (not shown).
PPPoA QUIZ

Why is the **Pad** field needed?
Why is it limited to 0-47 Bytes?

PPP data is sent in AAL5 frames over ATM cells:

![Diagram of AAL5 payload](image)

The AAL5 frame is then divided into 48 byte pieces, each of which goes into one ATM cell with 5 header bytes (not shown)
Why is the **Pad** field needed? Why is it limited to 0-47 Bytes?

A: The payload of each ATM cell is 48 Bytes, so the AAL5 frame length must be a multiple of 48.

Homework for Ch.3
Due Thu, April 12

End of chapter problems:
• 3, 4, 5, 6, 7, 10, 15, 18, 32, 37, 38

Hint for #37: Take the greatest PPP compression possible!
Review of Ch.3 formulas

To **detect** $d$ errors Hamming distance $\geq d+1$
To **correct** $c$ errors, Hamming distance $\geq 2 \cdot c + 1$

BD = (link bandwidth) x (one-way propagation delay)

$w \geq \text{floor}(2 \cdot \text{BD}) + 1$ for max. link utilization ($\rho$)

$\rho \leq \frac{w}{2 \cdot \text{BD} + 1}$

Go-Back-N: $w \leq s - 1$
Selective Repeat: $w \leq s/2$

---

**PPP frame:**

```
Flag 01111110 Address 11111111 Control 00000011 Protocol Payload Checksum Flag 01111110
```

**AAL5 frame:**

```
1 or 2 Variable 0 to 47 2 2 4
```

**ATM cell:**

```
Header Payload
5 bytes 48 bytes
AAL5 trailer
```