Data Communication Networks
Data Link Layer

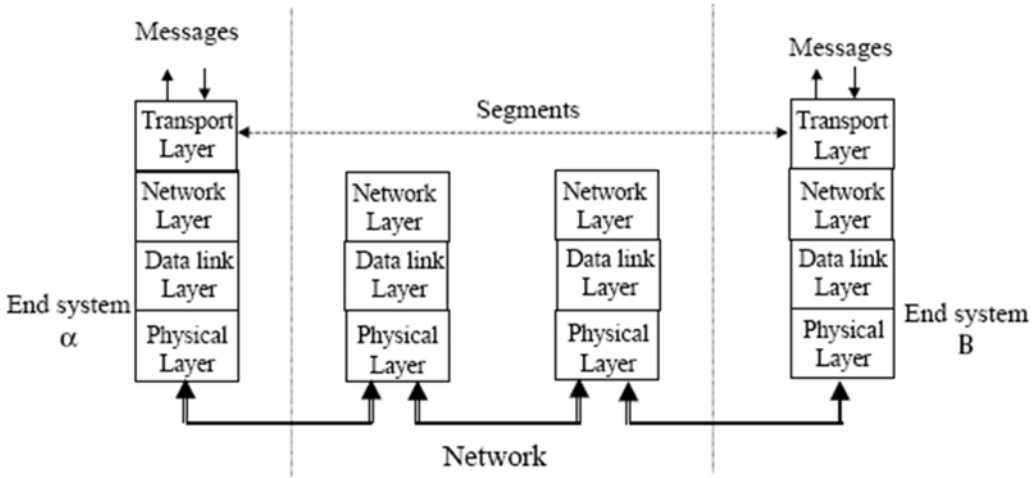
M. R. Pakravan
Department of Electrical Engineering
Sharif University of Technology

Data link layer

- Data Link Layer Purpose:
  - Ensure reliable, efficient communication between neighbor nodes
- Four specific functions:
  - Provide *services* to network layer
  - Framing
  - Error handling
  - Flow control
- Overview
  - Design issues
  - Error detection and correction
  - Elementary protocols
  - Sliding window protocols
  - Example protocols

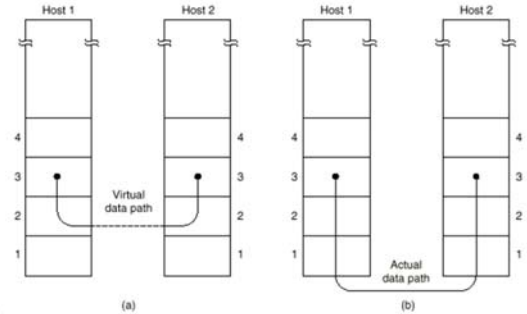
Data Link Layer

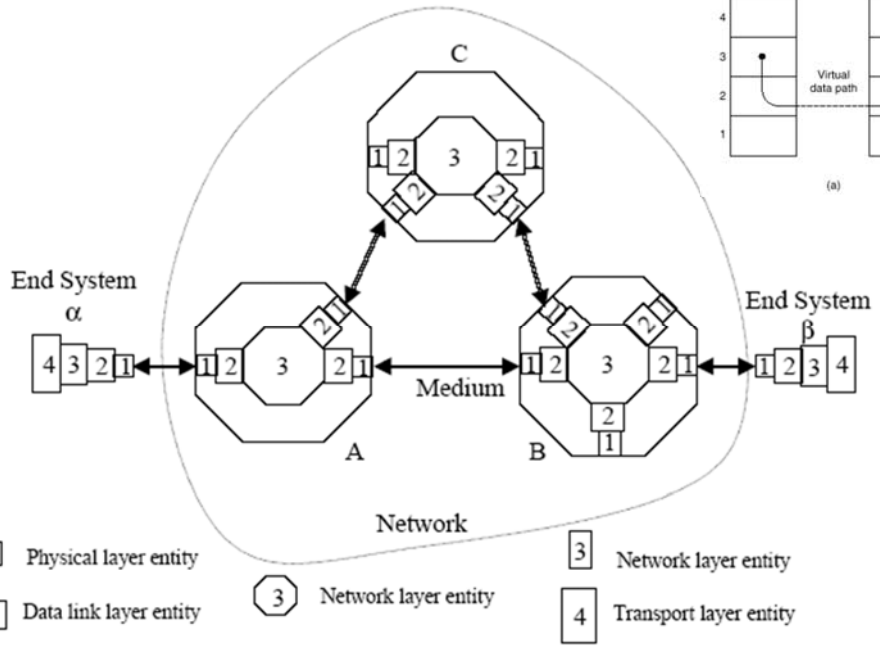
- Operation Perspective



Data Link Layer

- Data Link operation in a network





Services Provided to Network Layer

- Services provided to Network Layer:
  - Unacknowledged Connectionless
  - Acknowledged Connectionless
  - Acknowledged connection oriented

Services Provided to Network Layer

- Unacknowledged connectionless
  - Just send frames towards the destination
  - No connection is established, No connection released
  - No acknowledge of received frames
  - No attempt to recover lost frames
- Appropriate in case of:
  - Real-time traffic: speech, video: Short delay more important than 100% reliability
  - Low error channels: leave error correction to higher layers
  - Most LAN's are using this service class
- Example: Ethernet

Service to Network Layer

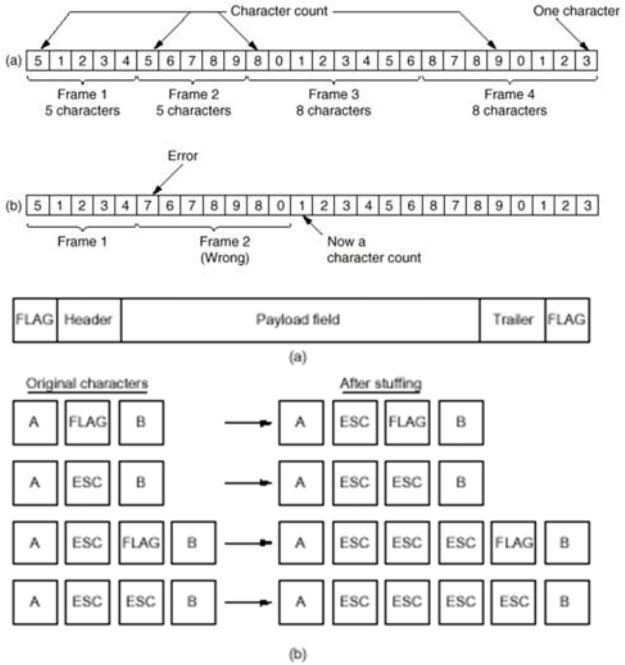
- Acknowledged connectionless
  - Each frame is acknowledged, but no connection is established
  - Acknowledgment is a service that can be performed by the transport layer as well
  - Data link layer provides this service to avoid long delays that could be caused if frames are not acknowledged
  - Specially important over un-reliable channels such as wireless
- Example: 802.11 (WiFi)

Service to Network Layer

- Acknowledged Connection Oriented
  - Establishes a connection before sending data
  - Each frame is numbered
  - Data link layer guarantees the delivery of a SINGLE copy of EVERY frame (With acknowledged connectionless, it is possible to receive multiple copies of a frame due to a lost ACK)
  - Releases the connection at the end of conversation (To release software and hardware resources tied up to the connection)
  - Provides a reliable bit stream to NL

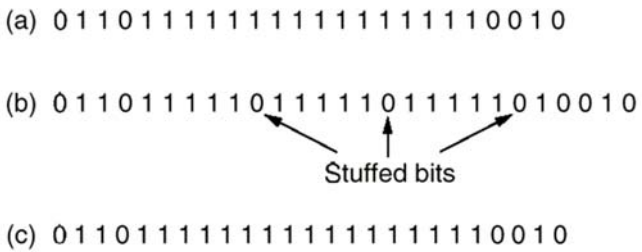
Framing

- Character Count
  - Use character count symbols to specify the end of frame
  - Sensitive to error. Difficult to recover if sync is lost
- Character stuffing
  - Insert ASCII known characters to specify boundaries (FLAG)
  - Insert ESC before accidental FLAGs or ESCs of binary files. To be removed by the receiver's data link layer
  - Choice of marking characters (ESC, FLAG) Depends on 8 bit ASCII character set



Framing

- Bit Stuffing
  - Use a flag bit pattern (01111110)
  - Stuff a zero after 5 1's if it happened in user data
  - Easy to recover the frame if sync is lost
- Coding Violation
  - Send physical layer signals that can not be associated with valid data sets to indicate frame boundaries
  - Example: coding violations in 4B/5B codes.

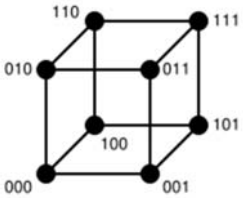


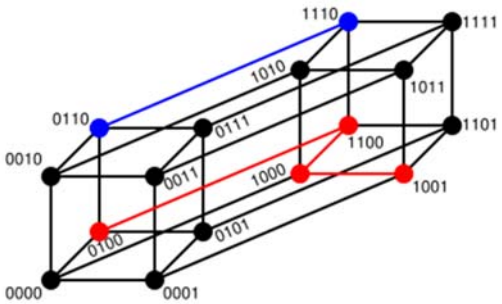
Error Detection and Correction

- Error handling
  - Errors usually occur in bursts
  - Advantage: fewer frames are corrupted
  - Disadvantage: burst errors harder to detect and correct
- Two approaches:
  - Detection
  - Correction
- Both require extra bits per data word or message
- Idea: add r extra bits to frame data
- Not all 2^{m+r} codes are legal (only 2^{m} codes are legal)
- n = m + r bits code word, m bits for data and r bits added for error control

Error Detection and Correction

- Hamming distance
  - The Hamming distance (HD) of 2 code words a and b is the number of bit positions in which a and b are different:
    HD (a,b)= Number of ones in
    (a XOR b)
  - Note that HD (a,b) bit errors are required to convert a into b (or vice versa)
  - If a code-set is designed to ensure a minimal hamming distance HD_{min} between any two valid code-words:
    - If $HD_{min} = d + 1$: d bit errors can be detected
    - If $HD_{min} = 2d + 1$: d bit errors can be corrected





Hamming distance

- Example 1: parity
  - Add (parity) bit to every data word such that total numbers of ones is even (or odd)
    - $HD_{min} = 2$
    - single bit errors are detectable
- Example 2:
  - 4 data words with $HD_{min} = 5$
  - double errors can be corrected
    - 0000.0000.00
    - 0000.0111.11
    - 1111.1000.00
    - 1111.1111.11
    - If 0000.0001.11 arrives, what is the original?

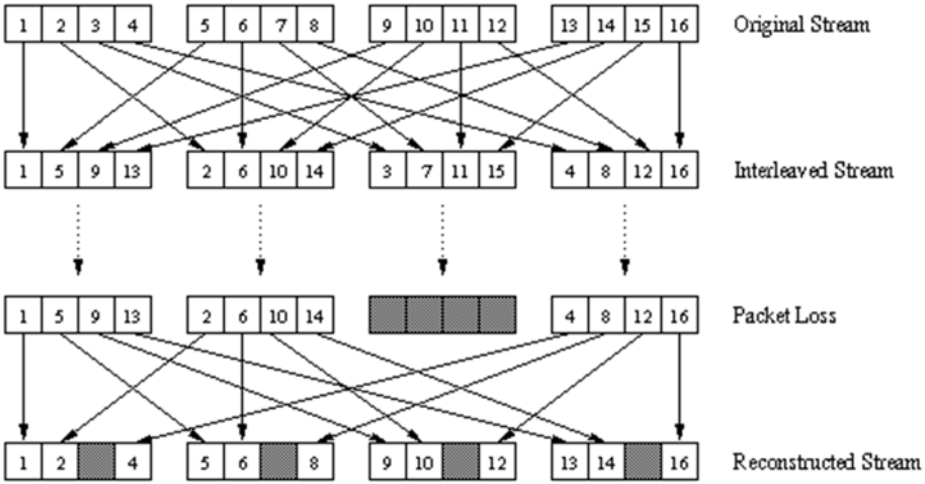
Hamming distance

- How many redundant (r) bits do we need for a single bit error correction?
  - Assume m bit message with r bits added for single error correction (n = m+r bits)
  - How many legal code-words? 2^m
  - How many illegal codes per code-words? $n*(2^m)$
  - $(n+1)*(2^m)<2^n$ => $(m+r+1)<2^r$
  - This is the minimum number of bits (r) we need to correct a single error in n bits with r parity bits
  - Example: M=8 => r>=4 ; M=16 => r>=5 ; M=240 => r>=8
- Error detection with retransmission often much more efficient than error correction
- *Example:*
  - Error rate 10^{-6} ---- Block size 1000 bits
  - Single bit error correcting requires 10 bits / block
  - Single bit error detection requires 1 bit / block
  - Retransmission of 1 block (= 1001 bits) per 1000 blocks is more efficient

Error Detection

- Burst errors
  - How to correct burst errors of length k ?
  - Arrange k words (to be send) as
    sequence of k rows (including check bits) and transmit this column wise
- No Interleaving
  - Error-free message: aaaabbbbccccddddeeeeffffgggg
  - Transmission with a burst error: aaaabbbbccczzzzdeeeeffffgggg
- Interleaving
  - Error-free transmission: aaaabbbbccccddddeeeeffffgggg
  - Interleaved: abcdefgabcdefgabcdefgabcdefg
  - Transmission with a burst error: abcdefgabcdzzzzbcdefgabcdefg
  - De-interleaved with a burst error: aazabbbbccccddddezeefzffgzgg
- Cost of interleaving?

Interleaving

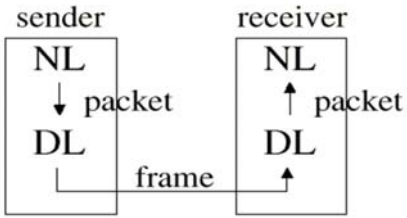


Cyclic Redundancy Check (CRC)

- View bit sequence as polynomial, e.g. (degree 5):110001=> x^5 + x^4 + x^0
- Idea: add r bits such that resulting polynomial is dividable by a certain polynomial; if not an error occurred
- Polynomial Arithmetic (modulo 2):
  - Add and Subtract are identical: use bit wise XOR (no carry/borrow needed)
  - Div: just like binary division, except of using the above subtract
- CRCs can easily be constructed and checked in hardware using a *linear feedback shift register* (LFSR)
- Standard polynomials:
  - CRC-12 = $x^{12} + x^{11} + x^3 + x^2 + x + 1$
  - CRC-16 = $x^{16} + x^{15} + x^2 + 1$
  - CRC-CCITT = $x^{16} + x^{12} + x^5 + 1$
- Example: CRC-16 and CRC-CCITT can catch:
  - All single and double errors, all odd errors all burst errors with length < 16
  - 99.997 % of all 17-bit burst errors and 99.998 % of all longer burst errors

Elementary DL Protocols

- Assumptions:
  - Simplex traffic: a long data stream from sender to receiver
  - Checksum errors are signaled by the physical layer
  - NL, DL, PL are independent processes
  - Connection-oriented, reliable service needed at NL => one and only one copy of each frame and in the exact sequence should be received by NL



Frame = packet + header

General Protocol Definitions

```
#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 defi
typedef enum {data, ack, nak} frame_kind;  /* frame_kind definition *

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

/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the char
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Unrestricted Simplex

- No error in communication channel
- No flow control
- Infinite buffer space at receiver side

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free,
   and the receiver is assumed to be able to process all the input infinitely fast.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                      /* buffer for an outbound frame */
  packet buffer;                /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;          /* copy it into s for transmission */
      to_physical_layer(&s);    /* send it on its way */
  }                             /* Tomorrow, and tomorrow, and tomorrow,
                                   Creeps in this petty pace from day to day
                                   To the last syllable of recorded time
                                       - Macbeth, V, v */
}

void receiver1(void)
{
  frame r;
  event_type event;             /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event); /* only possibility is frame_arrival */
      from_physical_layer(&r); /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
  }
}
```

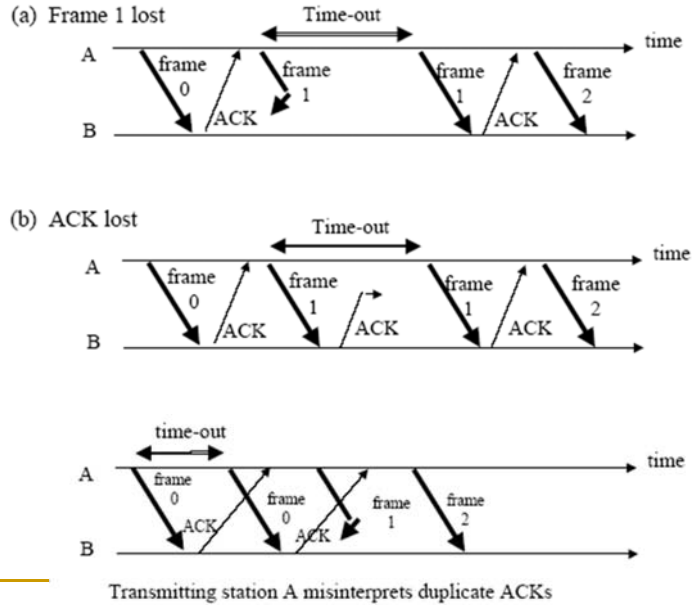
Stop and Wait Simplex

- One directional flow of data
- Error Free communication channel
- Receiver has finite processing power and finite buffer size
- Including flow-control to prevent flooding of receiver
- Flow Control: Prevent a fast sender from swamping a slow receiver

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                      /* buffer for an outbound frame */
  packet buffer;                /* buffer for an outbound packet */
  event_type event;             /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);                  /* go get something to send */
      s.info = buffer;          /* copy it into s for transmission */
      to_physical_layer(&s);    /* bye bye little frame */
      wait_for_event(&event);   /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                   /* buffers for frames */
  event_type event;             /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);   /* only possibility is frame_arrival */
      from_physical_layer(&r);  /* go get the inbound frame */
      to_network_layer(&r.info);/* pass the data to the network layer */
      to_physical_layer(&s);    /* send a dummy frame to awaken sender */
  }
}
```

Stop and Wait

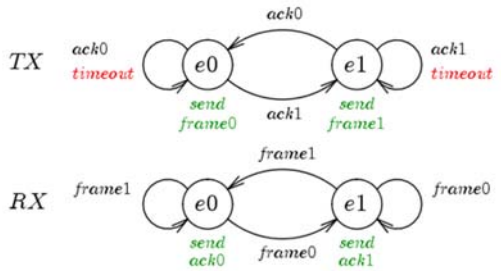
- Why an ACK is needed?
  - Sometimes, data link layer has to verify that the frames are received properly.
- What if a frame is totally lost?
  - Use timer to prevent freeze
- What if an ACK is lost?
  - Duplicated version of that frame would be accepted unless frames have sequence numbers
  - Some services want a single copy of every frame
  - So, frame numbering may need to be used.
  - A one bit number is sufficient in this protocol (Only a frame m and the next frame m+1 can be confused at the receiver)
- Should ACKs have numbers?



Transmitting station A misinterprets duplicate ACKs

Noisy Channel Simplex

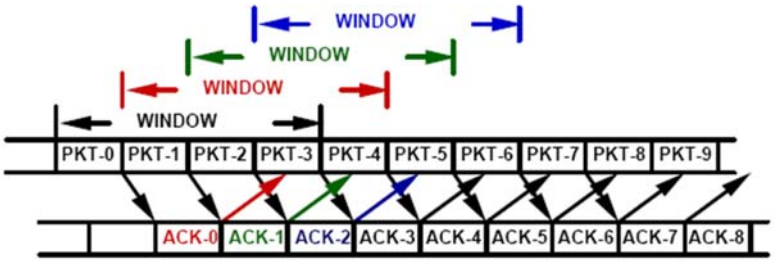
- Unreliable communication channel
- ACK is used to verify correct reception of a frame
- Sequence number needed (0 or 1) to distinguish retransmitted frames




```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1                       /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;            /* seq number of next outgoing frame */
  frame s;                              /* scratch variable */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;               /* initialize outbound sequence numbers */
  from_network_layer(&buffer);          /* fetch first packet */
  while (true) {
      s.info = buffer;                  /* construct a frame for transmission */
      s.seq = next_frame_to_send;       /* insert sequence number in frame */
      to_physical_layer(&s);            /* send it on its way */
      start_timer(s.seq);               /* if (answer takes too long, time out */
      wait_for_event(&event);           /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
          from_physical_layer(&s);      /* get the acknowledgement */
          if (s.ack == next_frame_to_send) {
              from_network_layer(&buffer); /* get the next one to send */
              inc(next_frame_to_send); /* invert next_frame_to_send */
          }
      }
  }
}

void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
      wait_for_event(&event);           /* possibilities: frame_arrival, cksum_err */
      if (event == frame_arrival) {     /* a valid frame has arrived. */
          from_physical_layer(&r);      /* go get the newly arrived frame */
          if (r.seq == frame_expected) { /* this is what we have been waiting for. */
              to_network_layer(&r.info); /* pass the data to the network layer */
              inc(frame_expected);      /* next time expect the other sequence nr */
          }
          s.ack = 1 - frame_expected;   /* tell which frame is being acked */
          to_physical_layer(&s);        /* none of the fields are used */
      }
  }
}
```

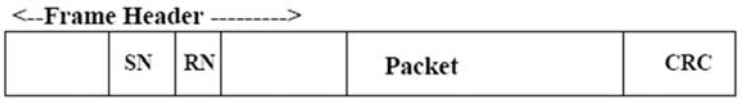
Go Back N

- Stop and Wait is inefficient when propagation delay is larger than the packet transmission time – Can only send one packet per round-trip time
- Go Back N allows transmission of new packets before earlier ones are acknowledged
- Go back N uses a window mechanism where the sender can send packets that are within a “window” (range) of packets – The window advances as acknowledgements for earlier packets are received



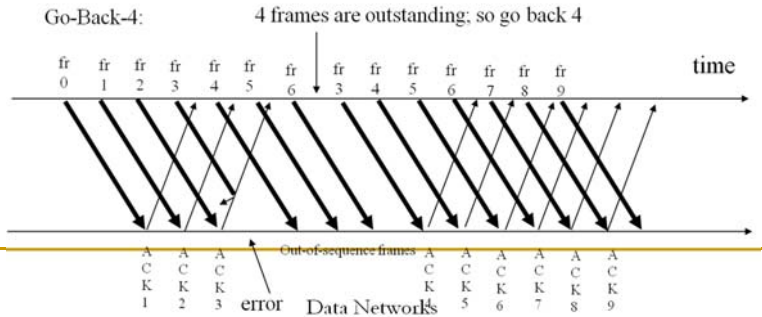
Go Back N

- Window size = N
- Sender cannot send packet i+N until it has received the ACK for packet i
- Receiver operates just like in Stop and Wait
  - Receive packets in order
  - Receiver cannot accept packet out of sequence
  - Send RN = i + 1 => ACK for all packets up to and including i
- Use of piggybacking
  - When traffic is bi-directional RN's are piggybacked on packets going in the other direction
  - Each packet contains a SN field indicating that packet's sequence number and a RN field acknowledging packets in the other direction



Go Back N

- The transmitter has a "window" of N packets that can be sent without acknowledgements
- This window ranges from the last value of RN obtained from the receiver (denoted SNmin) to SNmin+N-1
- When the transmitter reaches the end of its window, or times out, it goes back and retransmits packet SNmin
- Let SNmin be the smallest number packet not yet ACKed
- Let SNmax be the number of the next packet to be accepted from the higher layer (I.e., the next new packet to be transmitted)

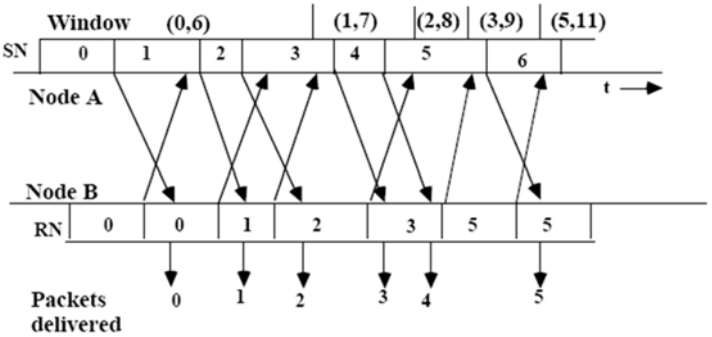


Go Back N – Sender Algorithm

- SNmin = 0; SNmax = 0
- Repeat
  - If SNmax < SNmin + N (entire window not yet sent)
    - Send packet SNmax ;
    - SNmax = SNmax + 1;
  - If packet arrives from receiver with RN > SNmin
    - SNmin = RN;
  - If SNmin < SNmax (there are still some unacknowledged packets) and sender cannot send any new packets
    - Choose some packet between SNmin and SNmax and re-send it or do not send anything
- The last rule says that when you cannot send any new packets you should re-send an old (not yet ACKed) packet
- There may be two reasons for not being able to send a new packet
  - Nothing new from higher layer
  - Window expired (SNmax = SNmin + N)
- No set rule on which packet to re-send
  - Least recently sent

Go Back N – Receiver Algorithm

- RN = 0;
- Repeat
  - When a good packet arrives,
    - if SN = RN
      - Accept packet
      - Increment RN = RN +1
- At regular intervals send an ACK packet with RN
  - Most implementations send an ACK whenever they receive a packet from the other direction
  - ACK can be delayed for piggybacking
- Receiver reject all packets with SN not equal RN
  - However, those packets may still contain useful RN numbers
- Note that packet RN-1 must be accepted at B before a frame containing request RN can start transmission at B



Notes of Go Back N

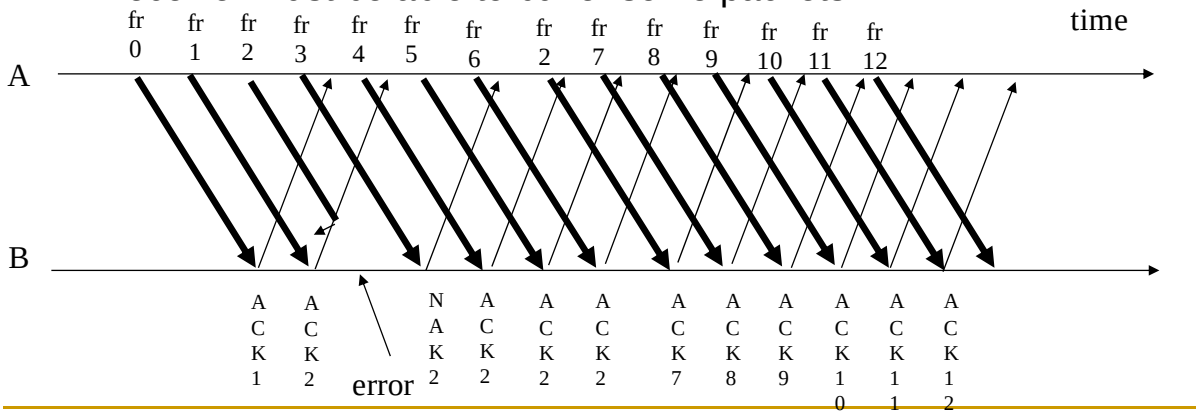
- Requires no buffering of packets at the receiver
- Sender must buffer up to N packets while waiting for their ACK
- Sender must re-send entire window in the event of an error
- Receiver can only accept packets in order
  - Receiver must deliver packets in order to higher layer
  - Cannot accept packet i+1 before packet i
  - This removes the need for buffering
  - This introduces the need to re-send the entire window upon error
- The major problem with Go Back N is this need to re-send the entire window when an error occurs.
- This is due to the fact that the receiver can only accept packets in order

Go Back-N Window Size

- Packets can be numbered modulo M where M > N
  - Because at most N packets can be sent simultaneously
- Example: 3-bit Sequence Number => frame numbers: 0 to 7 (8 numbers)
- Maximum Window size should be 7 or smaller (N=7)
  - Case I: N=8
    - TX: Transmitter sends frames 0 to 7 (8 frames)
    - TX: Receives ACK7
    - TX: Transmitter sends frames 0 to 7 (next series of 8 frames)
    - TX: Receives ACK7
    - Which ACK7 is this? (Which batch of 8 frames have arrived properly)
  - Case II: N=7
    - TX: Transmitter sends frames 0 to 6 (7 frames)
    - TX: Receives ACK6
    - TX: Transmitter sends frames 7,0,1,2,3,4,5 (next series of 7 frames)
    - TX: Receives ACK6
    - TX can know that the entire second set has been discarded because there were no frame number 6 in the second batch

Selective Repeat

- Selective Repeat attempts to retransmit only those packets that are actually lost (due to errors)
  - Receiver must be able to accept packets out of order
  - Since receiver must release packets to higher layer in order, the receiver must be able to buffer some packets



Selective Repeat

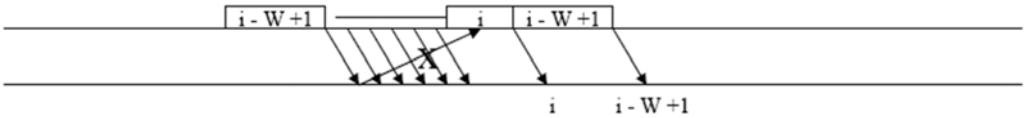
- Retransmission requests
  - Implicit
    - The receiver acknowledges every good packet, packets that are not ACKed before a time-out are assumed lost or in error
    - Notice that this approach must be used to be sure that every packet is eventually received
  - Explicit An explicit NAK (selective reject)
    - can request retransmission of just one packet
    - This approach can expedite the retransmission but is not strictly needed
  - One or both approaches are used in practice

Selective Repeat Algorithm

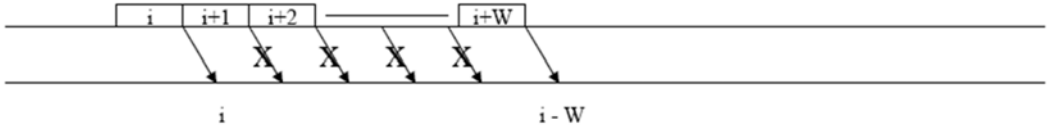
- Sender:
  - Can transmit new packets as long as their number is within W (Window Size) of all un-ACKed packets
  - Retransmits un-ACKed packets after a timeout (Or upon a NAK if NAK is employed)
  - Must buffer all packets until they are ACKed (Up to W un-ACKed packets are possible)
- Receiver:
  - ACKs all correct packets
  - Stores correct packets until they can be delivered in order to the higher layer
  - Must buffer packets until they can be delivered in order i.e., until all lower numbered packets have been received. This is needed for orderly delivery of packets to the higher layer
  - Up to W packets may have to be buffered (in case the first packet of a window is lost)
- Implication of buffer size = W
  - Number of un-ACKed packets at sender =< W (Buffer limit at sender)
  - Number of un-ACKed packets at sender cannot differ by more than W (Buffer limit at the receiver required to deliver packets in order)
  - Packets must be numbered modulo M >= 2W (using log2(M) bits)

Window Size Consideration in SRP

- Lets consider the range of packets that may follow packet i at the receiver

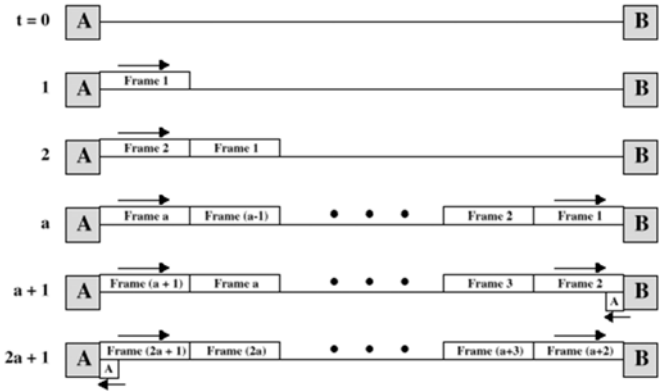


- Packet i may be followed by the first packet of the window (i -W+1) if it requires retransmission

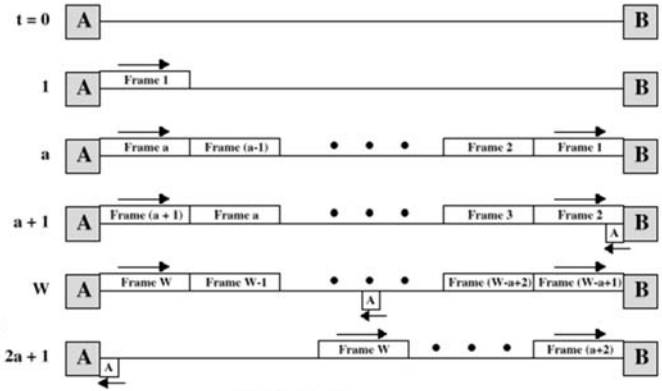


- Packet i may be followed by the last packet of the window (i+W) if all Of the frames between i and i +W are lost
- Receiver must differentiate between packets i -W+1 ... i +W
  - These 2W packets can be differentiated using Mod 2W numbering
  - Therefore, in SRP we need more bits for frame numbering

Performance Analysis of L2 Protocols



(a) W 2a + 1

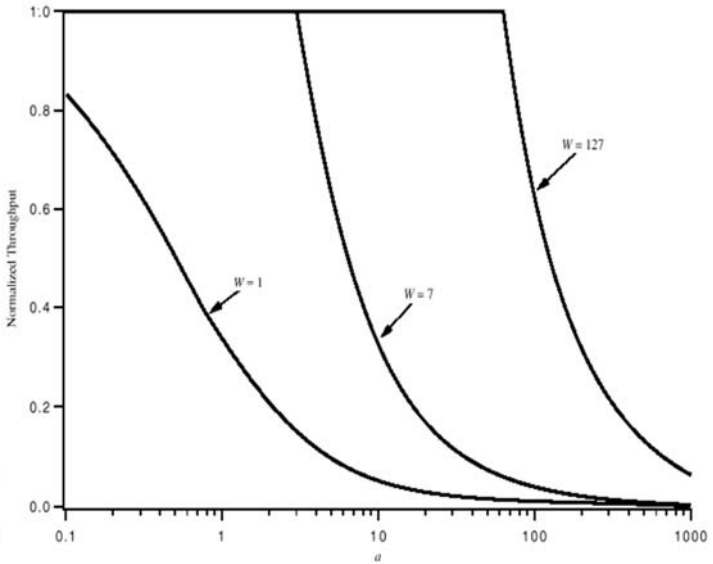


(b) W < 2a + 1

Link Utilization

- Error Free Operation
  - Stop and Wait: $U = \frac{1}{2a+1}$ $a = \frac{T_{prop}}{T_{frame}}$
  - Sliding Window (W=window Size=2^n-1):

$$U = \begin{cases} 1 & W > 2a+1 \\ \frac{W}{2a+1} & W < 2a+1 \end{cases}$$



Link Utilization

- Probability of frame error = *p*
- Stop and Wait Utilization
  - Probability of a frame requiring exactly k transmission = $p^{k-1}(1-p)$
  - Expected Number of Transmission for a frame (N_r):

$$E[N] = N_r = \sum_{k=1}^{\infty} k \times \text{Prob}(k \text{ Transmissi on}) = \sum_{k=1}^{\infty} kp^{k-1}(1-p) = \frac{1}{1-p}$$

  - Utilization should be divided by N_r : $U = \frac{1-p}{1+2a}$
- Selective Reject Utilization
  - Exact same idea as stop and wait

$$U = \begin{cases} 1-p & W > 2a+1 \\ \frac{W(1-p)}{1+2a} & W < 2a+1 \end{cases}$$

Link Utilization

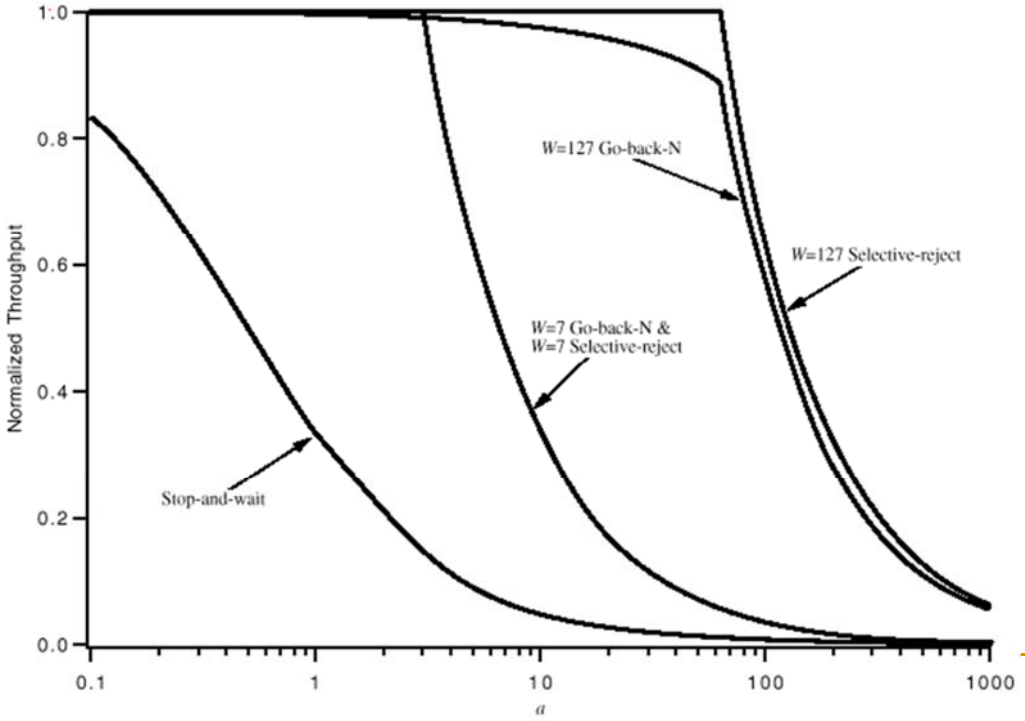
- Go back N Utilization
  - Each frame error requires re-transmission of *L* packets where $L \geq 1$
  - Total number of frames that should be transmitted if the original frame must be transmitted k times=$f(k)=1+(k-1)L$

$$E[N] = N_r = \sum_{k=1}^{\infty} f(k) \times \text{Prob}(k \text{ Transmissi on}) = \sum_{k=1}^{\infty} (1 - L + kL) p^{k-1} (1 - p)$$

$$= (1 - L) \sum_{k=1}^{\infty} p^{k-1} (1 - p) + L \sum_{k=1}^{\infty} k p^{k-1} (1 - p) = (1 - L) + \frac{L}{1 - p} = \frac{1 - p + LP}{1 - p}$$

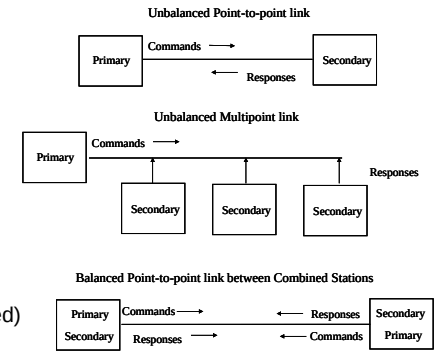
$$L \approx \begin{cases} 1 + 2a & W > 1 + 2a \\ W & W \leq 1 + 2a \end{cases} \qquad U = \begin{cases} \dfrac{1 - p}{1 + 2ap} & W > 2a + 1 \\ \dfrac{W(1 - p)}{(2a + 1)(1 - p + Wp)} & W < 2a + 1 \end{cases}$$

Link Utilization



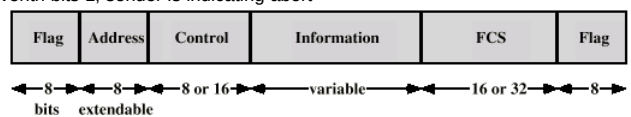
HDLC

- HDLC Link Configurations (Full duplex & Half duplex)
  - Unbalanced: One primary and one or more secondary stations
  - Balanced: Two combined stations
- Normal Response Mode (NRM)
  - Unbalanced configuration
  - Primary initiates transfer to secondary. Secondary may only transmit data in response to command from primary
  - Used on multi-drop lines
  - Host computer as primary.
  - Terminals as secondary
- Asynchronous Balanced Mode (ABM)
  - Balanced configuration (Most widely used)
  - Either station may initiate transmission without receiving permission. No polling overhead



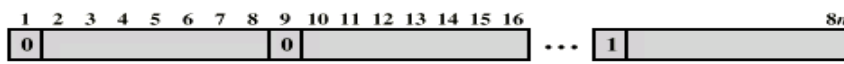
HDLC

- Frame Structure
  - All transmissions in frames
  - Single frame format for all data and control exchanges
  - On idle point to point links, flag sequences are transmitted continuously
- Flag Fields
  - Delimit frame at both ends with 01111110
  - May close one frame and open another
  - Receiver hunts for flag sequence to synchronize
  - Bit stuffing used to avoid confusion with data containing 01111110
    - 0 inserted after every sequence of five 1s
    - If receiver detects five 1s it checks next bit
    - If 0, it is deleted
    - If 1 and seventh bit is 0, accept as flag
    - If sixth and seventh bits 1, sender is indicating abort



HDLC Frame Fields

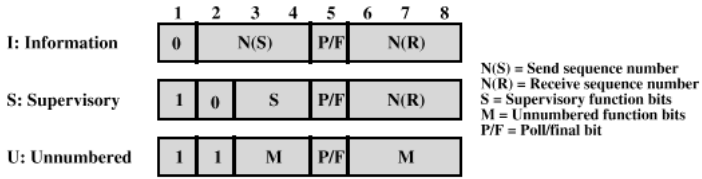
- Address Field
  - Identifies secondary station that sent or will receive frame
  - Usually 8 bits long
  - May be extended to multiples of 7 bits
    - LSB of each octet indicates that it is the last octet (1) or not (0)
  - All ones (11111111) is for broadcast



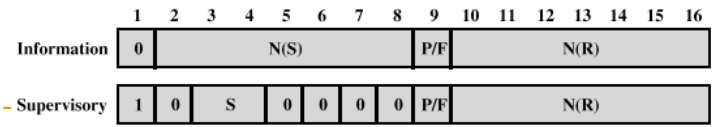
(b) Extended Address Field

HDLC Control Field

- Control frame varies for different frame types
  - Information - data to be transmitted to user (next layer up)
    - Flow and error control piggybacked on information frames
  - Supervisory - ARQ when piggyback not used
  - Unnumbered - supplementary link control
- First one or two bits of control field: frame type identification



(c) 8-bit control field format



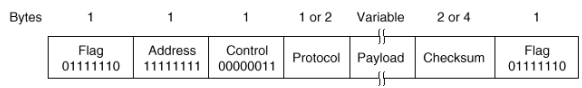
(d) 16-bit control field format

HDLC frame types

- Supervisory frames
  - Receive Ready Frame (SS=00)
    - Used when there are no information frames available to piggyback the ACK
  - Reject Frame (SS=01)
    - Receiver tells the sender to go back and start sending from the frame number N(R)
  - Receive Not Ready (SS=10)
    - All frames up to frame number N(R) have been received properly. But the receiver can not accept any frames at this time. (e.g. due to a full buffer)
  - Selective Reject (SS=11)
    - Receiver asks the sender to re-send frame number N(R)
- Unnumbered frames
- Used to start up the link or tear it down
- Some examples:
  - SABM: Set ABM Mode
  - SNRM: Set NRM Mode
  - SABME: Set ABM Extended Mode
  - DISC: Disconnect
  - UA: Unnumbered Acknowledgment
  - FRMR: Frame Reject

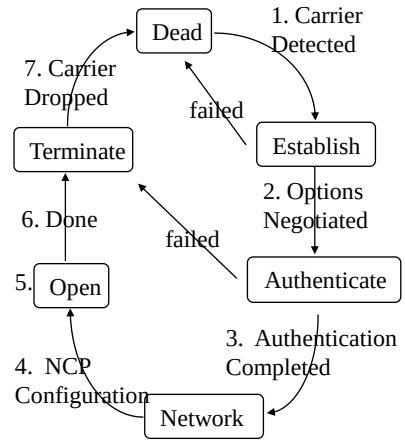
PPP - Point-to-Point Protocol

- Provides a method for encapsulation of network layer packets over point to point links. It provides:
  - An HDLC-like Framing with error detection (Byte oriented)
  - LCP (link control protocol) for bringing up lines, testing, negotiating options
  - NCP (network control protocol) for each supported network layer
- PPP frame format
  - Flag byte: 0111.1110 / character stuffing
  - Address field (default = 1111.1111)
  - Control field (default = 0000.0011)
  - Unnumbered frames with unreliable transmission; no sequence numbers
  - Protocol field: indicates packet type
    - 0x indicates IP, IPX or XNS
    - 1x indicates LCP, NCP or others
  - Payload: 1500 bytes default length
  - Checksum of 2 (default) or 4 bytes



Typical PPP Operation

- Example 1: Home PC to Internet Service Provider
  - PC calls router via modem.
  - PC and router exchange LCP packets to negotiate PPP parameters.
  - Check on identities.
  - NCP packets exchanged to configure the network layer, e.g., TCP/IP (requires IP address assignment).
  - Data transport, e.g. send/receive IP packets.
  - NCP used to tear down the network layer connection (free up IP address); LCP used to shut down data link layer connection.
  - Modem hangs up.
- Example 2: PPP use for transmission of packets over SONET SDH links



Data link layer in ATM

- There are no physical layer characteristics specific to ATM
  - ATM cells can be carried by various physical layer services such as SONET/SDH, ADSL or wireless links
- TC (transmission convergence) sub layer corresponds to DL layer
- Transmission: each cell of 53 byte includes 5 byte header
  - Add 4 bytes of identification and 1 byte of HEC (header checksum)
  - No checksum for payload. (BER<< 10^{-12} in fiber optic links)
  - Match speed to underlying physical transmission system by inserting empty or maintenance (OAM) cells
- Reception of a cell
  - Locate cell boundaries
  - Verify header
  - Discard cells with invalid headers
  - Process maintenance cells
  - Pass data to ATM layer

Data link layer in ATM

- How to detect cell boundaries (there are no marks !!)
  - Trick: constantly check HEC field after each incoming bit
  - Use 40-bits shift register to shift the incoming pattern bit-by-bit until the frame header is found
  - After the first header match, jump 53 bytes (length of an ATM frame) and re-check the HEC (is this the right HEC for the assumed cell?)
  - Repeat the header check d times in a row, If all correct, go to synch Mode
  - Change of accidental correct synchronization = 2^{-8d}
  - 'DeSynch' after a incorrect headers
  - TC sub layer uses header of ATM field for synchronization: no proper separation of layers. E.g. changing header format for ATM affects TC layer