UNIT II

- Switching Methods
- Transmission Media
- Framing
- Error Detection & Correction Methods
- Sliding Window Protocol
- Encoding Techniques
- HDLC
Circuit Switching

(a) Circuit switching.
(b) Packet switching.
Switching Schemes

(1) Circuit Switching
(2) Packet Switching (Store-and-Forward)
**Circuit Switching**

- **Circuit Switching** - When the user places a telephone call, the switching equipment within the telephone system seeks out a physical copper path all the way from our telephone to the receiver.
- When a call passes through a switching office, a physical connection is established between the sender and the receiver.
- Copper connection is the dedicated path between both ends that exists and continues to exist until the call is completed.
- Minimum time between start of ringing and end of dialing is 10 seconds (Exceed for long distance calls) during this time interval, phone systems will be hunting for a copper path.
- No Congestions.
Circuit Switching

1. Control message sets up a path from origin to destination
2. Return signal informs source that data transmission may proceed
3. Data transmission begins
4. Entire path remains allocated to the transmission (whether used or not)
5. When transmission is complete, source releases the circuit
Circuit Switching

- Call request signal
- Propagation Delay
- Call accept signal
- Data Transmission Time
- Data
- Transmission Delay

A B C D

Routers/Switches
Packet Switching

• Messages are split into smaller pieces called packets
• These packets are numbered and addressed and sent through the network one at a time
• Utilizes the path for a very small time
• Router forward the packet before the next packet arrives (Reduces the delay, Improving the throughput)
Packet Switching

Time

Header

Transmission Delay

A B C D

Pkt 1
Pkt 2
Pkt 3
Pkt 1
Pkt 2
Pkt 3
Pkt 1
Pkt 2
Pkt 3
Switched Networks

Circuit Switched

Packet Switched

store-and-forward
Datagram

- Each packet treated independently
- Packets can take any practical route
- Packets may arrive out of order
- Packets may go missing
- Up to receiver to re-order packets and recover from missing packets
## Comparisons

<table>
<thead>
<tr>
<th>Item</th>
<th>Circuit Switched</th>
<th>Packet Switched</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Dedicated Copper Path</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2. Bandwidth available</td>
<td>Fixed</td>
<td>Dynamic</td>
</tr>
<tr>
<td>3. Store and forward transmission</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>4. Each path follows the same route</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>5. Call setup</td>
<td>Required</td>
<td>Not needed</td>
</tr>
<tr>
<td>6. When can congestion occur</td>
<td>At set up time</td>
<td>On every packet</td>
</tr>
<tr>
<td>7. Charging</td>
<td>Per min</td>
<td>Per packet</td>
</tr>
</tbody>
</table>
Twisted Pair

- Physical media which is used for transmission is known as **transmission media**

Transmission Media types:
- Guided Media – wire (Twisted pair, Fiber optics, coaxial cable)
- Unguided – wireless
Twisted Pair

• Transmission time is measured in minutes or hours

• **Structure**: Consists of two insulated twisted copper wires typically about 1mm thick

• Twisted pairs can run several kilometers without amplification but for longer distances repeaters are needed

• Twisted pairs can be used for either analog or digital transmission

• Twisted pairs are widely used because of their
Twisted Pair

Varieties

• Category 1
• Category 2
• Category 3
• Category 4
• Category 5
Unshielded and Shielded TP

• Unshielded Twisted Pair (UTP)
  – Ordinary telephone wire
  – Cheapest
  – Easiest to install
  – Suffers from external EM interference

• Shielded Twisted Pair (STP)
  – Metal braid or sheathing that reduces interference
  – More expensive
  – Harder to handle (thick, heavy)
Twisted Pair

- **Category 3** twisted pairs consists of two insulated wires twisted together (10 mbps)
- Four such pairs are typically grouped together in a plastic cover for protection and to keep 8 wires together

**Category 5**

- More twisted pair per cm which results in less crosstalk and better quality signal over long distance
- Suitable for high speed computer
Twisted Pair

(a) Category 3 UTP.
(b) Category 5 UTP.
Coaxial cable

• Coaxial cable also known as coax has a single coated copper wire center and an outer metal mesh that acts as both grounding circuit and an electromagnetic shield to reduce interference

• Outer layer is the plastic cable jacket

• Coax cable connects to a hosts NIC and other devices with a barrel connector
Coaxial Cable
Coaxial Cable

(a) Three examples of a light ray from inside a silica fiber impinging on the air/silica boundary at different angles.

(b) Light trapped by total internal reflection.
Coaxial Cable

Attenuation of light through fiber in the infrared region.

- 0.85μ Band
- 1.30μ Band
- 1.55μ Band

Wavelength (microns)
(a) Side view of a single fiber.
(b) End view of a sheath with three fibers.
Fiber Media

- Fiber optic cable uses light pulses conducted through special glass conductors to carry data
- Cladding - Material that reflects escaping light into the core
- **Adv:** Greater bandwidth, Can run much faster than cable without needing a signal enhanced
- **Disadv:** Higher cost, special training for installing fiber
- Fiber optic cable can carry light in only one direction so fiber cable usually include a pair of fiber cores
Fiber Optic

- Light generated by either a laser or LED that converts the data to light pulses
- Receiving end photodiode interpret the light signal decode the bit pattern
- **Single Mode Fiber** - Long run,
- **Multi Mode Fiber** - Short run
Fiber Optic

Fiber Media Modes

Single-Mode
- Polymeric Coating
- Glass Cladding 125 microns dia
- Glass Core=8-10 microns
- Produces single straight path for light

Multimode
- Coating
- Glass Cladding 125 microns dia
- Glass Core=50/62.5 microns
- Allows multiple path for light

- Small Core
- Less Dispersion
- Suited for long distance applications (up to 100 km, 62.14 mi.)
- Uses lasers as the light source often within campus backbones for distance of several thousand meters

- Larger core than single-mode cable (50 microns or greater)
- Allows greater dispersion and therefore, loss of signal
- Used for long distance applications, but shorter than single-mode (up to ~2km, 6560 ft)
- Uses LEDs as the light source often within LANs or distances of couple hundred meters within a campus network
A fiber optic ring with active repeaters.
The DLL provides these services to the Network Layer above it:

- Data handed to a DLL by a Network Layer on one module, are handed to the Network Layer on another module by that DLL.

- The remote Network Layer peer should receive the identical message generated by the sender (e.g., if the data link layer adds control information, the header information must be removed before the message is passed to the Network Layer).

- The Network Layer may want to be sure that all messages it sends, will be delivered correctly (e.g., none lost, no corruption).
Frames are the unit of transmission. Consists of data plus control bits (header information).

Of special interest is:

```c
typedef struct frame {
  // definition of frame structure
} event_type;
```

```c
void wait_for_event( event_type *event );
```

```c
wait_for_event() suspends the process until an event occurs.
```
Flow control

- Stop and wait: Send one frame at a time
- Sliding window: Send several frames at a time
#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; /* what kind of a frame is it? */
    seq_nr seq; /* sequence number */
    seq_nr ack; /* acknowledgement number */
    packet info; /* the network layer packet */
} frame; /* frames are transported in this layer */
Elementary Data Link Protocols

/* 1. Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* 2. Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* 3. Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* 4. Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(packet *p);

/* 5. Pass the frame to the physical layer for transmission. */
void to_physical_layer(packet *p);

/* 6. Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* 7. Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* 8. Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* 9. Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* 10. Allow the network layer to cause a network_layer_event. */
void enable_network_layer(void);

/* 11. Forbid the network layer from causing a network_layer_event. */
void disable_network_layer(void);
Elementary Data Link Protocols

An Unrestricted Simplex Protocol
Assumptions:

Data transmission in one direction only (simplex).

No errors take place on the physical channel.

The sender/receiver can generate/consume an infinite amount of data.

Always ready for sending/receiving.
Elementary Data Link Protocols

/* 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 */
    }
}
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 */
    }
}
Elementary Data Link Protocols

Simplex Stop and Wait Protocol

Assumptions:
How to prevent the sender from flooding the receiver
Sender ships one frame and then waits for acknowledgment
(stop and wait.)
The contents of the acknowledgment frame are unimportant.
Stop and Wait

WT = Wait time

Sender

WT

Data

ACK

Data

ACK

Data

ACK

EOT

Receiver

Time

Time
Elementary Data Link Protocols

/* 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); /* send it on its way */
        wait_for_event(&event); /* do not proceed until given the go ahead */
    }
}

void receiver2(void)
{
    frame r, s; /* filled in by wait, but not used here */
    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 */
        to_physical_layer(&s); /* send a dummy frame to awaken sender */
    }
}
Elementary Data Link Protocols

SIMPLEX PROTOCOL FOR A NOISY CHANNEL:

Assumptions:

The channel is noisy and we can lose frames (they never arrive).

Simple approach, add a time-out to the sender so if no ACK after a certain period, it retransmits the frame.

Scenario of a bug that could happen if we’re not careful:

A transmits frame one
B receives A1
B generates ACK
ACK is lost
A times out, retransmits
B gets duplicate copy of A1 (and sends it on to network layer.)

Use a sequence number. How many bits? 1-bit is sufficient for this simple case because only concerned about two successive frames.

Positive Acknowledgment with Retransmission (PAR): Sender waits for positive acknowledgment before advancing to the next data item. (Numerous alternatives to this we will see later.)
/* 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; /* buffer for an outbound frame */
    packet buffer; /* buffer for an outbound packet */
    event_type event; /* frame_arrival is the only possibility */
    next_frame_to_send = 0;
    from_network_layer(&buffer); /* go get something to send */
    while (true) {
        s.info = buffer; /* copy it into s 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(&event); /* frame arrival or cksum err, or timeout */
        if (event == frame_arrival) {
            from_physical_layers(&s); /* Get the ACK */
            if (s.ack == next_frame_to_send) {
                from_network_layers(&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); /* only possibility is frame arrival */
        if (event == frame_arrival) { /* A valid frame has arrived */
            from_physical_layer(&r); /* go get the inbound frame */
            if (r.seq == frame_expected) { /* This is what we’ve been waiting
                for */
                to_network_layer(&r.info); /* pass the data to the network
                layer */
                inc(frame_expected); /* next time expect the other seq #
                */
            }
            to_physical_layer(&s); /* send a dummy frame to awaken sender */
        }
    }
}
A Problem unresolved by this protocol is this:

How long should the timer be?

What if too long? (inefficient)

What if too short? A problem because the ACK does not contain the sequence number of the frame which is being ACK'd. So, which frame is being ACK’d?

Scenario:

A sends frame A0
time out of A
resend frame A0
B receives A0, ACKS
B receives A0 again, ACKS again (does not accept)
A gets A0 ACK, sends frame A1
A1 gets lost
A gets second A0 ACK (assumes it’s ACK of A1), sends A2
B gets A2 (rejects, not correct seq. number)

Will lose two frames before getting back on track (with A3)
Assumptions:

Use more realistic Two-way communication.

We now have two kinds of **frames** (containing a "kind" field):

- **Data**
- **ACK** containing (sequence number of last correctly received frame).

**Piggybacking** - add acknowledgment to data frames going in reverse direction.

Piggybacking issue: For better use of bandwidth, how long should we wait for outgoing data frame before sending the ACK on its own.
Sliding Window Protocols

**Sliding window** :: sender has a window of frames and maintains a list of consecutive sequence numbers for frames that it is permitted to send without waiting for ACKs. receiver has a window that is a list of frame sequence numbers it is permitted to accept.
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.
/* Protocol 4 (sliding window) is bi-directional and is more robust than protocol 3 */

#define MAX-SEQ 1 /* must be 1 for protocol 4 */
typedef enum {frame-arrival, cksum-err, timeout} event-type;
#include "protocol.h"

void protocol4 (void) {
    seq-nr next-frame-to-send; /* 0 or 1 only */
    seq-nr frame-expected; /* 0 or 1 only */
    frame r, s; /* scratch variables */
    packet buffer; /* current packet being sent */
    event-type event;
    next-frame-to-send = 0; /* next frame on the outbound stream */
    frame-expected = 0; /* number of frame arriving frame expect */
    from-network-layer(&buffer); /* fetch a packet from the network layer */
    s.info = buffer; /* prepare to send the initial frame */
    s.seq = next-frame-to-send; /* insert sequence number into frame */
    s.ack = 1 -frame-expected; /* piggybacked ack */
    to-physical-layer(&s); /* transmit the frame */
    start-timer(s.seq); /* start the timer running */
while (true) {
    wait-for-event(&event); /* frame-arrival, cksum-err, or timeout */
    if (event == frame-arrival) {
        /* a frame has arrived undamaged. */
        from-physical-layer(&r);
        if (r.seq == frame-expected) {
            /* Handle inbound frame stream. */
            to-network-layer(&r.info);
            inc(frame-expected);
        }
    }
    if (r.ack == next-frame-to-send) { /* handle outbound frame stream. */
        /* pass packet to network layer */
        from-network-layer(&buffer);
        /* get new pkt from network layer */
        inc(next-frame-to-send);
    }
    s.info = buffer; /* construct outbound frame */
    s.seq = next-frame-to-send; /* insert sequence number into it */
    s.ack = 1 - frame-expected; /* seq number of last received frame */
    to-physical-layer(&s);
    start-timer(s.seq); /* transmit a frame */
}
Sender does not wait for each frame to be ACK’ed. Rather it sends many frames with the assumption that they will arrive. Must still get back ACKs for each frame.

Provides more efficient use of transmit bandwidth, but error handling is more complex.

What if 20 frames transmitted, and the second has an error. Frames 3-20 will be ignored at receiver side? Sender will have to retransmit. What are the possibilities?

Two strategies for receive Window size:
Example of a **sliding window** protocol. Contains a sequence number whose maximum value, $\text{MaxSeq}$, is $2n - 1$.

For stop-and-wait sliding window protocol, $n = 1$.

Essentially same as Simplex Protocol, except ACKs are numbered, which solves early time out problem. Two-way communication.

Protocol works, all frames delivered in correct order.

Requires little buffer space.

Poor line utilization due to stop-and-wait. (To be solved in next example.)
Go-Back-n: If one frame is lost then all the data frames should be sent from the last frame acknowledged.

Selective Repeat: Only the specific frame is retransmitted.
Sliding Window Protocols

GoBack-N
Sliding Window Protocols

- Timeout interval
- Error
- Frames discarded by data link layer

J. Godwin Ponsam
Go-Back-N

Figure 10.20: Go-back-n, lost data frame

Figure 10.21: Go-back-n, lost ACK
Selective Repeat Sliding Window Protocols
Sliding Window Protocols

Go Back n.

Timeout interval

Error
Frames discarded by data link layer

Time

J. Godwin Ponsam
Selective Repeat

Frames buffered by data link layer

Error
/* Protocol5 (pipelining) allows multiple outstanding frames. The sender may transmit up to MAX-SEQ frames without waiting for an ack. In addition, unlike the previous protocols, the network layer is not assumed to have a new packet all the time. Instead, the network layer causes a network-layer-ready event when there is a packet to send. */

#define MAX-SEQ 7 /* should be 2^n-1 */
typedef enum {frame-arrival, cksum-err, timeout, network-layer-ready} event-type;
#include "protocol.h" /* Return true if (a <= b < c circularly; false otherwise. */

static boolean between(seq-nr a, seq-nr b, seq-nr c) {
    if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
        return(true);
    else
        return(false);
}

static void send-data(seq-nr frame-nr, seq-nr frame-expected, packet buffer[]) {
    /* Construct and send a data frame. */
    frame s;
    s.info = buffer[frame-nr]; /* insert packet into frame */
    s.seq = frame-nr; /* insert sequence number into frame */
    s.ack = (frame-expected + MAX-SEQ) % (MAX-SEQ + 1); /* piggyback ack */
    to-physical-layer(&s);
    start-timer(frame-nr); /* transmit the frame */
    /* start the timer running */
}

J. Godwin Ponsam
void protocol5(void) {
    seq-nr    next-frame-to-send;    /* MAX-SEQ > 1; used for outbound stream */
    seq-nr    ack-expected;         /* oldest frame as yet unacknowledged */
    seq-nr    frame-expected;       /* next frame expected on inbound */
    stream */
    frame     r;                    /* scratch variable */
    packet    buffer[MAX-SEQ + 1 ]; /* buffers for the outbound stream */
    seq-nr    nbuffered;            /* # output buffers currently in use */
    seq-nr    i;                    /* used to index into the buffer array */
    event-type event;               /* allow network-layer-ready events */
    enable-network-layer();
    ack-expected = 0;                /* next ack expected inbound */
    next-frame-to-send = 0;          /* next frame going out */
    frame-expected = 0;              /* number of frame expected inbound */
    nbuffered = 0;                   /* initially no packets are buffered */
}
while (true) {
    wait-for-event(&event);  /* four possibilities: see event-type */
    switch(event) {
        case network_layer_ready: /* the network layer has a packet to send */
            from-network_layer(&buffer[next-frame-to-send]); /* fetch new packet */
            nbuffered = nbuffered + 1; /* expand the sender's window */
            send-data(next_frame-to-send, frame-expected, buffer); /* transmit the frame */
            inc(next_frame-to-send); /* advance sender's upper window edge */
            break;
        case frame-arrival: /* a data or control frame has arrived */
            from_physical_layer(&r); /* get incoming frame from physical layer */
            if (r.seq == frame-expected) { /* Frames are accepted only in order. */
                to_network-layer(&r.info); /* pass packet to network layer */
                inc(frame-expected); /* advance lower edge of receiver's window */
            }
        }
        while (between(ack-expected, r.ack, next_frame_to_send)) { /* Ack n implies n-1, n-2, etc. Check this. */
            nbuffered = nbuffered -1; /* one frame fewer buffered */
            stop-timer(ack-expected); /* frame arrived intact; stop timer */
            inc(ack-expected); /* contract sender's window */
        }
        break;
    }
}

Sliding Window Protocols

case cksum-err: break; /* just ignore bad frames */

case timeout: /* trouble; retransmit all outstanding frames*/
    next-frame-to-send = ack-expected; /* start retransmitting here */
    for (i = 1; i <= nbuffered; i++) {
        send-data(next-frame-to-send, frame-expected, buffer); /* resend 1 frame */
        inc(next-frame-to-send); /* prepare to send the next one */
    }
    break;
}

if (nbuffered < MAX-SEQ)
    enable_network_layer();
else
    disable_network_layer();
}
Fig. 3-17. Simulation of multiple timers in software.
Digital to Digital Encoding
Digital to Digital Encoding

- Digital/digital encoding
  - Unipolar
  - Polar
  - Bipolar
Unipolar Encoding

- **Problem:**
  1. DC component
  2. Synchronization
- In Unipolar encoding one voltage level stands for binary 0 and another one stands for binary 1
- Polarity defines whether its positive polarity or negative polarity
- Average amplitude is non-zero
- Uses only one polarity usually 1 other usually 0
- In expensive
Polar Encoding

- Polar
  - NRZ
    - NRZ-L
    - NRZ-I
  - RZ
  - Biphasic
    - Manchester
    - Differential Manchester
Polar Encoding

- Uses 2 voltage levels: one positive and one negative.
- DC component problem is alleviated in polar encoding.
- Non return to zero (NRZ)
- NRZ-L – Level of the signal depends on the type of bit:
  - Positive voltage - bit 0
  - Negative voltage - bit 1
- Problem: Long stream of 0s or 1s in the data.
Polar Encoding

**Figure 5.6** *NRZ-L and NRZ-I encoding*

- **NRZ-L** and **NRZ-I**

  - Transition because next bit is 1.
Polar Encoding

- NRZ-I: The signal is inverted if a 1 is encountered.
- NRZ-I is superior to NRZ-L due to the synchronization provided by the signal change each time a 1 bit is encountered.
- The existence of 1’s in the data stream allows the receiver to resynchronize its timer.
- Problem: String of 0s still cause a problem.
Polar Encoding

• **Return to zero (RZ)**

• To assure synchronization there must be a signal change for each bit

• In RZ, the signal change is not between bits but during each bit

• 1 bit is represented by positive to zero and 0 bit is represented by negative to zero

• Disadvantage is it requires 2 signal changes to encode 1 bit
Polar Encoding

Figure 5.7  RZ encoding

These transitions can be used for synchronization.
Polar Encoding

- Zero is \[ \_ \_ \_ \_ \]
- One is \[ \_ \_ \_ \_ \]

Amplitude

0 1 0 0 1 1 1 1 0

Manchester

Differential Manchester

Presence of transition at the beginning of bit time means zero.
Polar Encoding

- Biphase
- **Manchester encoding**
- Uses the inversion at the middle of each bit interval for both synchronization and bit representation
- Negative to positive transition is binary 1
- Positive to negative transition is binary 0
- Achieves same synchronization as RZ but with only two levels of amplitude
Polar Encoding

• **Differential Manchester Encoding**

  • Transition at the middle of the bit is used only for synchronization but the presence or absence of an additional transition is used to identify the bit

  • Transition means binary 0 and no transition means binary 1

  • Requires two signal changes to represent binary 0 and only one for binary 1
Types of Bipolar Encoding

- AMI – Alternate Mark Inversion
- B8zs
- HDB3
AMI

The 1s are positive and negative alternately.
Bipolar Encoding

- Uses three voltage levels positive, negative and zero
- BMI (Bipolar Alternate Inversion)
- Zero voltage represents binary 0
- Binary 1s are represented by alternating positive and negative voltages
- DC component is zero
- Long sequence of 1s synchronized but 0s are not synchronized
- Solved by Bipolar 8 zero substitution (B8zs), high density bipolar 3 (HDB3)
Bipolar Encoding

• Anytime eight or more 0s are encountered in the data stream. The soln provided by B8zs is to force artificial signal changes called violation

• Anytime eight 0s occur B8zs introduces the change in the pattern based on the polarity of the previous 1

• If the previous 1 bit was positive the eight 0s will be encodes as zero, zero, zero, positive, negative, zero,negative,positive

• If the previous 1 bit was negative the violations are same but with inverted polarities
Bipolar Encoding

Polarity of previous bit

0 0 0 0 0 0 0 0 0

Will change to

0 0 0 0 + - 0 - +

Violation

Violation

Polarity of previous bit

- 0 0 0 0 0 0 0 0 0

Will change to

- 0 0 0 0 - + 0 + -

Violation

Violation
Bipolar Encoding

1's Odd

1's Even
Bipolar Encoding

• **HDB3**

• If four 0s come one after another we change the pattern in one of four ways based on the polarity of the previous 1 and the number of 1s since the last substitution

• No of 1’s odd

• No of 1’s Even
HDLC Station Types

• Primary station
  – Controls operation of link
  – Frames issued are called commands

• Secondary station
  – Under control of primary station
  – Frames issued called responses

• Combined station
  – May issue commands and responses
HDLC Link Configurations

• Unbalanced
  – One primary and one or more secondary stations
  – Supports full duplex and half duplex

• Balanced
  – Two combined stations
  – Supports full duplex and half duplex
Three data transfer modes

- **NRM** - Used with an unbalanced configuration. Primary initiate data transfer to secondary. Secondary responses

- **ABM** - Balanced – Either combined station initiate transmission

- **ARM** - Unbalanced – Secondary initiate transmission
HDLC Configuration

Unbalanced

Primary

Command

Secondary

Response

Secondary

Response
HDLC Configuration

Balanced

Combined

Command/response

Combined

Command/response
## HDLC Modes

<table>
<thead>
<tr>
<th>Station type</th>
<th>NRM</th>
<th>ARM</th>
<th>ABM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary &amp; secondary</td>
<td>Primary &amp; secondary</td>
<td>Combined</td>
<td></td>
</tr>
<tr>
<td>Initiator</td>
<td>Primary</td>
<td>Either</td>
<td>Any</td>
</tr>
</tbody>
</table>
Figure 11-16

HDLC Frame Types

User data coming from upper layer.

I-frame

Flag Address Control Information FCS Flag
HDLC Frame Types

S-frame

Flag Address Control FCS Flag
Figure 11-16-continued

HDLC Frame Types

Management Data

Flag Address Control Information FCS Flag

U-frame
Frame Structure

• All transmissions in frames
• Single frame format for all data and control exchanges
Frame Structure Diagram

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>Information</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
</table>

8 bits, extendable
Figure 11-20

**HDLC Control Field**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>Information</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
</table>

- **I-Frame**
  - P/F: Poll/ final bit
  - N(S): Sequence number of frame sent
  - N(R): Sequence number of next frame expected

- **S-Frame**
  - Code: Code for supervisory or unnumbered frame

- **U-Frame**
  - Code: Code for supervisory or unnumbered frame
Flag Fields

• Delimit frame at both ends
• 01111110
• 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
Bit Stuffing Example

Original Pattern:

111111111111011111101111110

After bit-stuffing

11111101111110110111111010111111010
Address Field

- Identifies secondary station that sent or will receive frame
- Usually 8 bits long
- May be extended to multiples of 7 bits with prior agreement
  - leftmost bit of each octet indicates that it is the last octet (1) or not (0)
Frame Types

• Information - data to be transmitted to user
  – Acknowledgment is piggybacked on information frames

• Supervisory – ARQ messages (RR/RNR/REJ/SREJ) when piggyback not used

• Unnumbered - supplementary link control functions. For examples,
  – setting the modes
  – disconnect

• Control field is different for each frame type
<table>
<thead>
<tr>
<th>Name</th>
<th>Command/Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>C/R</td>
<td>Exchange user data</td>
</tr>
<tr>
<td>Supervisory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive ready</td>
<td>C/R</td>
<td>Positive ack; Ready to Receive I Frame</td>
</tr>
<tr>
<td>Receive not ready</td>
<td>C/R</td>
<td>Positive ack; NOT Ready to Receive I Frame</td>
</tr>
<tr>
<td>Reject</td>
<td>C/R</td>
<td>Negative ack; goback N</td>
</tr>
<tr>
<td>Selective Reject</td>
<td>C/R</td>
<td>Negative ack Selective Reject</td>
</tr>
<tr>
<td>Unnumbered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNRM/SNRME</td>
<td>C</td>
<td>Set Mode = 7 bit sequence numbers</td>
</tr>
<tr>
<td>SARM/SARME</td>
<td>C</td>
<td>Set Mode = 7bit sequence numbers</td>
</tr>
<tr>
<td>SABM/SABME</td>
<td>C</td>
<td>Set Mode = 7bit sequence numbers</td>
</tr>
<tr>
<td>SIM</td>
<td>C</td>
<td>Initialize link control functions</td>
</tr>
<tr>
<td>DISC</td>
<td>C</td>
<td>Terminate logical link connection</td>
</tr>
<tr>
<td>UA</td>
<td>R</td>
<td>Ack acceptance of one of the set mode commands</td>
</tr>
<tr>
<td>DM</td>
<td>R</td>
<td>Responder in disconnected mode</td>
</tr>
<tr>
<td>Request Disconnect(RD)</td>
<td>R</td>
<td>Request for DISC command</td>
</tr>
<tr>
<td>Request Initialization modeRIM)</td>
<td>R</td>
<td>Initialization needed</td>
</tr>
<tr>
<td>Unnumbered Information (UI)</td>
<td>C/R</td>
<td>Used to exchange control information</td>
</tr>
<tr>
<td>Unnumbered Poll</td>
<td>C</td>
<td>Used to solicit control information</td>
</tr>
<tr>
<td>Reset(RSET)</td>
<td>C</td>
<td>Used to recovery</td>
</tr>
<tr>
<td>Exchange Identification(XID)</td>
<td>C/R</td>
<td>Used to request/report status</td>
</tr>
<tr>
<td>Test(TEST)</td>
<td>C/R</td>
<td>Exchange identical information</td>
</tr>
<tr>
<td>Frame reject(FRMR)</td>
<td>R</td>
<td>Report receipt of unacceptable frame</td>
</tr>
</tbody>
</table>
Control Field Diagram

(c) 8-bit control field format

(d) 16-bit control field format
Poll/Final Bit

• Use depends on context. A typical use is below.

• Command frame
  – P bit set to 1 to solicit (poll) supervisory frame from peer

• Response frame
  – F bit set to 1 to indicate response to soliciting command
Use of P/F Field

Primary

P = 1, RNR

Select

Secondary
Use of P/F Field

Primary

Secondary

Positive response to select

F = 1, RR

Negative response to select

F = 1, RNR
Information Field

- Only in information and some unnumbered frames
- Must contain integral number of octets
- Variable length
Frame Check Sequence Field

- FCS
- Error detection
- 16 bit CRC
- Optional 32 bit CRC
HDLC Operation

• Exchange of information, supervisory and unnumbered frames

• Three phases
  – Initialization
  – Data transfer
  – Disconnect
Initialization

• Issue one of six set-mode commands
  – Signals other side that initialization is requested
  – Specifies mode (NRM, ABM, ARM)
  – Specifies 3- or 7-bit sequence numbers

• If request accepted HDLC module on other side transmits unnumbered acknowledged (UA) frame

• If request rejected, disconnected mode (DM) sent
Data Transfer

• Both sides may begin to send user data in I-frames
  – N(S): sequence number of outgoing I-frames
    • modulo 8 or 128, (3- or 7-bit)
  – N(R) acknowledgment for I-frames received
    • I-frame expected next

• S-frames also used for flow and error control
  – Receive ready (RR) frame acknowledges last I-frame received
    • Indicating next I-frame expected
    • Used when no reverse data
  – Receive not ready (RNR) acknowledges, but also asks peer to suspend transmission of I-frames
    • When ready, send RR to restart
  – REJ initiates go-back-N ARQ
    • Indicates last I-frame received has been rejected
    • Retransmission is requested beginning with N(R)
  – Selective reject (SREJ) requests retransmission of single frame
Disconnect

• Send disconnect (DISC) frame
• Remote entity must accept by replying with UA
  – Informs layer 3 user connection terminated
Examples of Operation (1)

(a) Link setup and disconnect
(b) Two-way data exchange
(c) Busy condition
Examples of Operation (2)

(d) Reject recovery

(e) Timeout recovery
Error Detection and Error Correction

- Detecting and correcting errors requires redundancy - sending additional information along with the data.

**Error Detecting Codes:** Include enough redundancy bits to detect errors and use ACKs and retransmissions to recover from the errors.

**Error Correcting Codes:** Include enough redundancy to detect and correct errors.
Error Detection

1. Parity Check
2. CRC (Polynomial Code)
3. Checksum

1. Parity Check
   - In this technique a redundant bit called a parity bit is appended to every data so that the total number of 1’s in the unit becomes even or odd
   - 1110111 1101111 1110010 1101100 1100100
   - 1110111
   - 0
   - 1101111
   - 0
   - 1110010
   - 0
   - 1101100
   - 0
   - 1100100
   - 1
   - Receiver knows that the data is corrupted, so discards it and asks for retransmission
   - If 2 single bit error occurs then the error cannot be detected
CRC

- A sequence of redundant bits called the CRC or the CRC remainder is appended to the end of a data unit.
- At the destination, the incoming data unit is divided by the same number.
- Sender:
  1. First, a string of 0s is appended to the data unit. The number n is one less than the predetermined divisor which is n+1 bits.
  2. The newly elongated data unit is divided by the divisor using a process called binary division.
  3. CRC of n bits derived in step 2 replaces the appended 0’s at the end of the data unit.

Receiver:
  1. The receiver treats the whole string as a unit and divides it by the same divisor that was used to find the CRC remainder.
  2. If the string arrives without error, the CRC checker yields a remainder of zero, and the data unit passes. If the string has been changed in transit, the division yields a non-zero remainder, and the data unit does not pass.

- CRC generator is most often represented not as a string of 1s and 0s but as an algebraic polynomial.
Figure 9-10

CRC

Sender

00...0 Data

n bits

Divisor n+1 bits

Remainder

CRC n bits

Receiver

CRC

Data

CRC

Data

Divisor

Remainder

Zero, accept
Non-zero, reject
Binary Division

Divisor: 1101

Quotient:

000

10010
1101
1010
1101
1110
1101
0110
0000

Remainder: 001
Polynomial

\[ x^7 + x^5 + x^2 + x + 1 \]
Figure 9-13

Polynomial and Divisor

Polynomial

\[ x^7 + x^5 + x^2 + x + 1 \]

Divisor

[Diagram showing polynomial division with coefficients 1, 0, 1, 0, 0, 1, 1, 1]
Hamming Code

Redundancy bits
Hamming Code

\( r_1 \) will take care of these bits

\[
\begin{array}{ccccccccc}
1011 & 1001 & 0111 & 0101 & 0011 & 0001 \\
11 & 9 & 7 & 5 & 3 & 1 \\
d & d & d & r_8 & d & d & d & r_4 & d & r_2 & r_1
\end{array}
\]

\( r_2 \) will take care of these bits

\[
\begin{array}{ccccccccc}
101110110 & 01110110 & 00110010 \\
11 & 10 & 7 & 6 & 3 & 2 \\
d & d & d & r_8 & d & d & d & r_4 & d & r_2 & r_1
\end{array}
\]
Hamming Code

$r_4$ will take care of these bits

011101100101 0100

7 6 5 4

d d d r₈ d d d d r₄ d r₂ r₁

$r₈$ will take care of these bits

101110101001 1000

11 10 9 8

d d d r₈ d d d d r₄ d r₂ r₁
Figure 9-20

Example of Hamming Code

Data: 1 0 0 1 1 0 1

Adding \( r_1 \):
1 0 0 1 1 0 1

Adding \( r_2 \):
1 0 0 1 1 0 1 0 1

Adding \( r_4 \):
1 0 0 1 1 0 0 1 0 1

Adding \( r_8 \):
1 0 0 1 1 1 0 0 1 0 1

Code: 1 0 0 1 1 1 0 0 1 0 1
Single-bit error

Figure 9-21

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

Error
Error Detection

The bit in position 7 is in error.
Checksum

• Sender

1. The checksum generator subdivides the data unit into equal segments of n bits
2. These bits are added together using ones complement
3. The total is then complemented and appended to the end of the original data unit as redundancy bits called the checksum field
Checksum

• Receiver:
  1. Subdivides the data unit and adds all segments together and complements the result
  2. If the checksum field is zero then it accepts, else the receiver rejects it
Disclaimer

The contents of the slides are solely for the purpose of teaching students at SRM University. All copyrights and Trademarks of organizations/persons apply even if not specified explicitly.
Review questions

• 1. What is the difference between Circuit Switching and Packet Switching?
• 2. List the methods of framing
• 3. Define Bit stuffing
• 4. Define Piggybacking
• 5. What is the advantage of selective repeat ARQ over Goback-N ARQ?
• 6. List the types of frames in HDLC
• 7. Define Unnumbered frames
• 8. List the problems in Unipolar Encoding
• 9. Define link configuration in HDLC
• 10. Define Stop and Wait ARQ
bibliography

• 4. Behrouz A. Forouzan, Data and Computer