Data Link Layer: Flow Control
Stop-and-Wait Data Link Protocols

- Such elementary protocols are also called PAR (Positive Acknowledgment with Retransmission) or ARQ (Automatic Repeat reQuest).
- Data frames are transmitted in one direction (simplex protocols) where each frame is individually acknowledged by the receiver by a separate acknowledgment frame.
- The sender transmits one frame, starts a timer and waits for an acknowledgment frame from the receiver before sending further frames.
- A time-out period is used where frames not acknowledged by the receiver are retransmitted automatically by the sender.
- Frames received damaged by the receiver are not acknowledged and are retransmitted by the sender when the expected acknowledgment is not received and timed out.
- A one bit sequence number (0 or 1) is used to distinguish between original data frames and duplicate retransmitted frames to be discarded.
- Such protocols result in a substantial percentage of wasted bandwidth and may fail under early time-out situations.
Definitions for Data Link Protocols  (protocol.h) 1/2

#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;

/* 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 channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);
Definitions for Data Link Protocols (protocol.h) 2/2

/* 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
Protocol 1: An Unrestricted Simplex Protocol

- Transmission in one direction
- The receiver is always ready to receive the next frame (has infinite buffer storage).
- Error-free communication channel.
- No acknowledgments or retransmissions used.
- If frame has \( d \) data bits and \( h \) overhead bits, channel bandwidth \( b \) bits/second:
  - maximum channel utilization  =  data size/frame size  =  \( d/(d + h) \)
  - maximum data throughput  =  \( d/(d + h) \) * channel bandwidth  =  \( d/(d + h) \) * \( b \)

Frame transmission time  =  \((d+h)/b\)
\( b = \) channel bandwidth

Data Frame 1
Data Frame 2
Data Frame 3
Data Frame 4
Data Frame 5
Data Frame 6

One way
Channel delay
or latency \( l \)

Sender

Receiver
An Unrestricted Simplex Protocol

```c
#include "protocol.h"

typedef enum {frame arrival, even type event} frame event;

text = frame發布(文本);  // send text to the sender

while (文本)
{
    while (true)
    {
        if (文本)
        {
            packet buffer;
            if (文本)
            {
                void send_text(void)
                {
                    // send text
                }
            }
        }
    }
}

void send_text(void)
{
    // send text
}
```

An unrestricted simplex protocol is a communication protocol that allows data to be transmitted in one direction only, from a sender to a receiver. The communication channel is assumed to be error-free, and the receiver is assumed to be able to process all the input that it receives. This ensures that the data transmitted is accurately received by the receiver. The protocol is used for data transmission in one direction, ensuring efficient transmission and minimizing overhead.
Protocol #2: A Simplex Stop-and-Wait Protocol

- **Simplex**: Data transmission in one direction
- The receiver may not be always ready to receive the next frame (finite buffer storage).
- Receiver sends a positive acknowledgment frame to sender to transmit the next data frame.
- Error-free communication channel assumed. No retransmissions used.
- Maximum channel utilization $\approx \frac{\text{time to transmit frame}}{\text{round trip time}} \times \frac{d}{d + h}$
  $\approx \frac{d}{b \times R}$
- Maximum data throughput $\approx \text{channel utilization} \times \text{channel bandwidth}$
  $\approx \frac{d}{b \times R} \times b = \frac{d}{R}$

![Diagram of Data Frame Transmission and Acknowledgments]
A Simplex Stop-and-Wait Protocol

Data Link Protocol #2
Protocol 3: A Simplex Positive Acknowledgment with Retransmission (PAR) Protocol

- The receiver may not be always ready to receive the next frame (finite buffer storage).
- Noisy communication channel; frames may be damaged or lost.
  - Frame not received correctly with probability $p$
- Receiver sends a positive acknowledgment frame to sender to transmit the next data frame. Any frame has a sequence number, either 0 or 1
- Maximum utilization and throughput similar to protocol 2 when the effect of errors is ignored.

![Diagram of Protocol 3](image-url)
Protocol 3: A Simplex PAR Protocol (continued)

Effect of Errors

- The sender starts a timer when transmitting a data frame.
- If data frame is lost or damaged (probability = $p$):
  - Receiver does not send an acknowledgment
  - Sender times out and retransmits the data frame
Data Link Protocol #3 1/2 (sender process)
A Simplex positive Acknowledgment with Retransmission Protocol
```c
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, checksum_err */
        if (event == frame_arrival) {
            from_physical_layer(&r);
            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 */
        }
    }
}
```

Data Link Protocol #3 2/2 (receiver process)
A Simplex positive Acknowledgment with Retransmission Protocol
Data Link Layer: Flow Control
Sliding Window Protocols

• These protocols allow both link nodes (A, B) to send and receive data and acknowledgments simultaneously.
• Acknowledgments are piggybacked into an acknowledgment field in the data frame header not as separate frames.
• If no new data frames are ready for transmission in a specified time, a separate acknowledgment frame is generated to avoid time-out.
• Each outbound frame contains a sequence number ranging from 0 to $2^{n-1}$ (n-bit field). $N = 1$ for stop-and-wait sliding window protocols.
• Sending window: A set of sequence numbers maintained by the sender and correspond to frame sequence numbers of frames sent out but not acknowledged.
• The maximum allowed size of the sending window $w$ correspond to the maximum number of frames the sender can transmit before receiving any acknowledgment without blocking (pipelining).
• All frames in the sending window may be lost or damaged and thus must be kept in memory or buffers until they are acknowledged.
Sliding Window Data Link Protocols

- **Receiving window:** A set of sequence numbers maintained by the receiver and indicate the frames sequence numbers it is allowed to receive and acknowledge.

- The size of the receiving window is fixed at a specified initial size.

- Any frame received with a sequence number outside the receiving window is discarded.

- The sending window and receiving window may not have the same upper or lower limits or have the same size.

- When pipelining is used, an error in a frame is dealt with in one of two ways:
  
  - **Go back n:**
    - The receiver discards all subsequent frames and sends no acknowledgments.
    - The sender times out and resends all the discarded frames starting with faulty frame.
  
  - **Selective repeat:**
    - The receiving data link stores all good frames received after a bad frame.
    - Only the bad frame is retransmitted upon time-out by the sender.
A Sliding Window Protocol of Size 1 with a 3-bit Sequence Number

(a) Initial state
(b) After the first frame has been sent
(c) After the first frame has been received
(d) After the first acknowledgment frame has been received
### Difference Between PAR and Sliding Window Protocols

**Positive Acknowledgment with Retransmission**

| Stop-and-Wait |  | Sliding Window |
|---------------|  | sequence # from 0 to 3 |
| **Data Frame 1, seq # 0** |  | **DF 1, seq # 0** |
| Ack Frame, seq # 0 |  | **DF 2, seq # 1** |
| **Data Frame 2, seq # 1** |  | **DF 3, seq # 2** |
| Ack Frame, seq # 1 |  | **DF 4, seq # 3** |
| **Data Frame 3, seq # 0** |  |  |
| Ack Frame, seq # 0 |  |  |
| **Data Frame 4, seq # 1** |  |  |
| Ack Frame, seq # 1 |  |  |
|  |  |  |
| Receiver |  | Sender | Receiver | Sender |
A 4-Frame Sending Window

(a) Initial window

12 11 10 9 9 7 6 5 4 3 2 1

still unsent

(b) After two frames have been acknowledged

12 11 10 9 9 7 6 5 4 3 2 1

already acknowledged

Unacknowledged or Pending Frames

After two frames have been acknowledged

window

12 11 10 9

After nine frames have been acknowledged
Data Link Protocol #4  1/2

A 1-bit Bi-directional Sliding Window Protocol

/* Protocol 4 (sliding window) is bidirectional 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;
    seq_nr frame_expected;
    frame_r, s;
    packet buffer;
    event_type event;
    next_frame_to_send = 0;
    frame_expected = 0;
    from_network_layer(&buffer);
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 - frame_expected;
    to_physical_layer(&s);
    start_timer(s.seq);
    /* next frame on the outbound stream */
    /* number of frame arriving frame expected */
    /* fetch a packet from the network layer */
    /* prepare to send the initial frame */
    /* insert sequence number into frame */
    /* piggybacked ack */
    /* transmit the frame */
    /* start the timer running */
Data Link Protocol #4  2/2
A 1-bit Bi-directional Sliding Window Protocol

```c
wait_for_event(&event); // frame_arrival, cksum_err, or timeout */
if (event == frame_arrival) { /* a frame has arrived undamaged. */
  from_physical_layer(&r); // go get it */
  if (r.seq == frame_expected) {
    /* Handle inbound frame stream. */
    to_network_layer(&r.info); // pass packet to network layer */
    inc(frame_expected); // invert sequence number expected next */
  }
  if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
    from_network_layer(&buffer); // fetch new pkt from network layer */
    inc(next_frame_to_send); // invert sender's sequence number */
  }
  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); // transmit a frame */
  start_timer(s.seq); // start the timer running */
}
```
Channel Utilization & Data Throughput

For Sliding Window Protocols

\[ b = \text{Channel bandwidth or transmission rate bits/sec} \]
\[ FS = \text{Frame size} = \text{# of data bits} + \text{# overhead bits} = d + h \]
\[ R = \text{Channel round trip time} \]
\[ N = \text{Send/receive window size} \]
\[ p = \text{Probability frame a data frame is lost or damaged} \]

- Ignoring errors, condition to maximize Utilization/Throughput:

\[
\text{Time to transmit } N \text{ frames } \geq \text{Round trip time} \\
\frac{FS}{b} \times N = \frac{(d + h)}{b} \times N \geq R
\]

Under this condition:

Maximum channel utilization \( \approx \frac{\text{data size/frame size}}{= \frac{d}{d + h}} \)

Maximum data throughput \( \approx \frac{d}{FS} = \frac{d}{d + h} \times b \)

- Including the effect of errors only on data frame; assuming selective repeat:

On the average \( p \) data frames have to be retransmitted

Under these condition: Total Data Frame overhead = \( h + p \times FS \)

Maximum channel utilization \( \approx \frac{d}{[(1 + p)\times FS]} \)

Maximum data throughput \( \approx \frac{d}{[(1 + p)\times FS]} \times b \)
Two Operation Sequences For Sliding Window Protocol (#4)

(a) Normal Protocol Operation:
No duplicate packets

(b) A special situation:
Half the frames contain duplicates

* Network layer accepts a packet.
(a) Effect of an error when the receiver size is 1

(b) Effect of an error when the receiver size is large
Finite State Machine Protocol Models

• A protocol may be represented by a finite state machine (protocol machine).
• States are chosen when the protocol machine is waiting for the next event (i.e. sending or receiving a protocol data unit PDU).
• The state of the complete protocol is the combination of the state of the two protocol machines and the channel.
• The state of the channel depends on its content.
• Each state may have one or more transitions to other states when protocol events occur.
• Incomplete state machine specification.
• Deadlock states.
State Transition Diagram For Protocol 3

- The protocol state machine states are represented by XYZ
  - $X = 0$ or 1 depending on the sequence number of the frame the sender is attempting to send.
  - $Y = 0$ or 1 depending on the sequence number of the frame the receiver expects.
  - $Z = 0, 1, A$ or empty (-) corresponding to the state (content) of the channel.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Who runs?</th>
<th>Frame accepted</th>
<th>Frame emitted</th>
<th>To network layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>(frame lost)</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>R</td>
<td>0</td>
<td>A</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>A</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>R</td>
<td>1</td>
<td>A</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td>A</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>R</td>
<td>0</td>
<td>A</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>R</td>
<td>1</td>
<td>A</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>S</td>
<td>(timeout)</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>S</td>
<td>(timeout)</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>
Data Link Protocol Example: HDLC - High-Level Data Link Control

- Bit-oriented protocol derived from IBM’s SNA data link protocol SDLC (Synchronous Data Link Control).
- Frame Types: Information, Supervisory, Unnumbered.
- Uses sliding window with 3-bit sequence numbers.
- Uses CRC-CCITT for error control.
- Protocol commands include:
  - DISC (DISConnect) used to disconnect a machine from the line.
  - SNRM (Set Normal Response Mode) brings a machine online and sets one machine as channel master and the other as slave (was used for dumb terminals when connected to mainframes).
  - SABM (Set Asynchronous Balanced Mode).
  - FRMR (FRaMe Reject) rejects a frame with correct checksum with impossible structure.
# HDLC Bit-Oriented Frame Format

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>≥ 0</th>
<th>16</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>011111110</td>
<td>Address</td>
<td>Control</td>
<td>Data</td>
<td>Checksum</td>
<td>011111110</td>
</tr>
</tbody>
</table>

### Information Frame

<table>
<thead>
<tr>
<th>Bits</th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>Seq</td>
<td>P/F</td>
<td>Next</td>
</tr>
</tbody>
</table>

### Supervisory Frame

<table>
<thead>
<tr>
<th>Bits</th>
<th>1</th>
<th>0</th>
<th>Type</th>
<th>P/F</th>
<th>Next</th>
</tr>
</thead>
</table>

### Unnumbered Frame

<table>
<thead>
<tr>
<th>Bits</th>
<th>1</th>
<th>1</th>
<th>Type</th>
<th>P/F</th>
<th>Modifier</th>
</tr>
</thead>
</table>
Data Link For Temporary Internet Host Connection

- Serial Line IP (SLIP).
- Point-to-Point Protocol (PPP).
Internet Data Link Protocols:
Serial Line IP (SLIP)  RFC 1055

- Send raw IP packets with a flag byte (0xC0) at the end for framing with character stuffing (data 0xC0 replaced with 0xDB 0xDC).
- Recent versions use header compression by omitting header fields in consecutive packets and frames.
- Does not include any form of error detection or correction.
- Supports only one network protocol: IP (Internet Protocol).
- Dynamic IP address assignment not supported.
- Lacks any form of authentication.
Internet Data Link Protocols:
Point-to-Point Protocol (PPP)

- Uses standard HDLC framing byte (01111110) with error detection.
- Uses Link Control Protocol (LCP) for bringing lines up, option negotiation, and to bring lines down.
- Network layer options and configurations are negotiated independent of the network layer used by utilizing different NCPs (Network Control Protocol) packets for each supported network layer.
- Support for several packet types by using a protocol field:
  - Network protocols (protocol field starts with 0): IP, IPX, AppleTalk etc.
  - Negotiating protocols (protocol field starts with 1): LCP, NCP.
- PPP is used for both dial-up network access and for router-to-router communication in subnets.
PPP Frame Format & Transition Diagram

PPP Frame Format for unnumbered mode operation.

Simplified PPP phase diagram for bringing a line up or down.
# PPP Line Control Packet Types

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