CS4220
Computer Networks

Lecture 3 Data Link Layer

Dr. Xiaobo Charles Zhou
Department of Computer Science

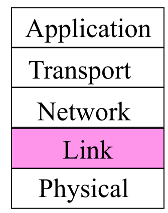
Data Link Layer Design Issues

- Services Provided to the Network Layer
  - Provide service interface to the network layer
- Data Link Layer Design Issues
  - Framing
- Error Detection and Correction
  - Dealing with transmission errors
- Elementary Data Link Protocols
- Flow Control – Sliding Window Protocols
  - Slow receivers not swamped by fast senders

Can error control and flow control be done at other layers, e.g., the transport layer?

The Data Link Layer

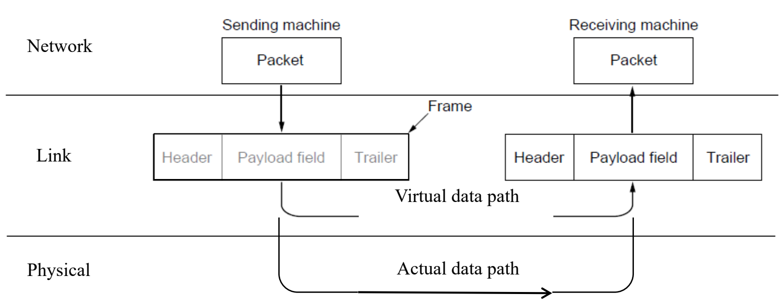
- **Responsible for delivering frames of information over a single "wirel-like" link**
  - **Handles transmission errors and regulates the flow of data**



What is the essential property of a single "wire-like" link?
bits are delivered in order

Frames

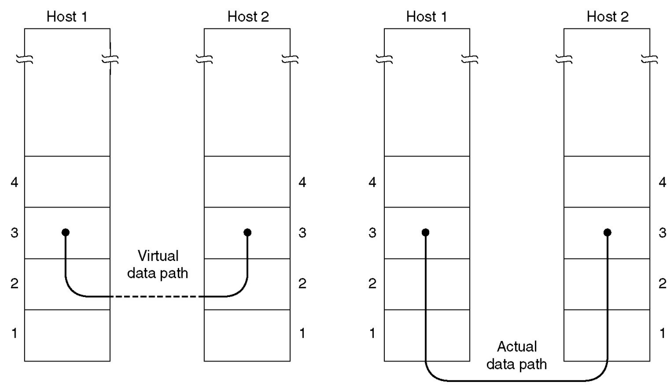
- **Link layer accepts <u>packets</u> from the network layer, and encapsulates them into <u>frames</u> that it sends using the physical layer; reception is the opposite process**



Relationship between packets and frames.

Services Provided to Network Layer

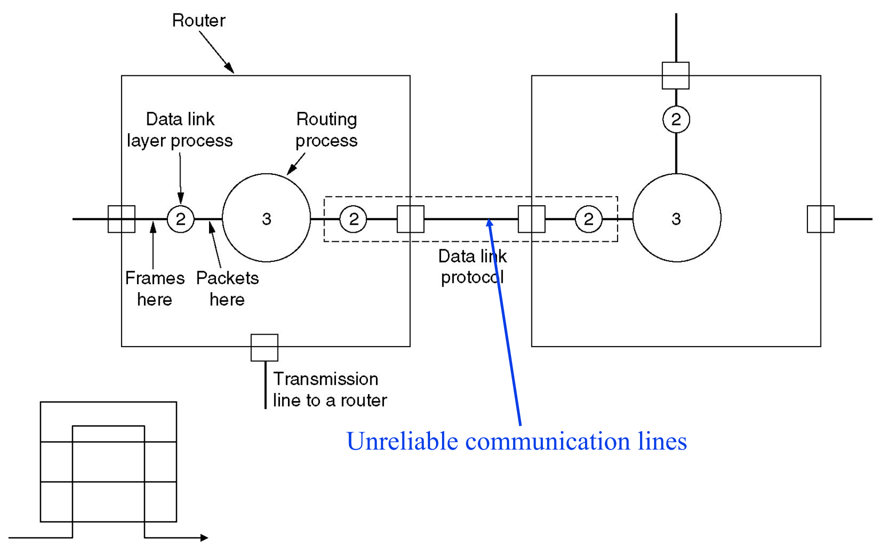
- Unacknowledged connectionless service
- Acknowledged connectionless service; say in wireless networks (optimization vs. requirement)
- Acknowledged connection-oriented service (no duplicate)



(a) Virtual communication. (b) Actual communication.

Services Provided to Network Layer (2)

Placement of the data link protocol.



Framing

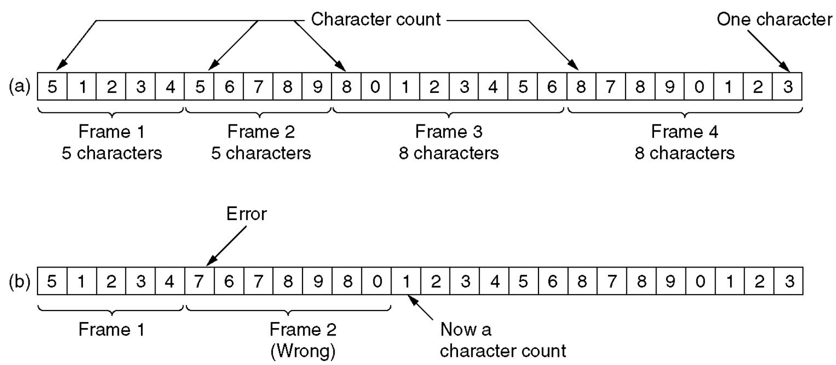
- Framing: to break the bit stream up into discrete frames and compute the checksum for each frame.

Framing Methods

- Byte count »
- Flag bytes with byte stuffing »
- Flag bits with bit stuffing »
- Physical layer coding violations
  - Use non-data symbol to indicate frame

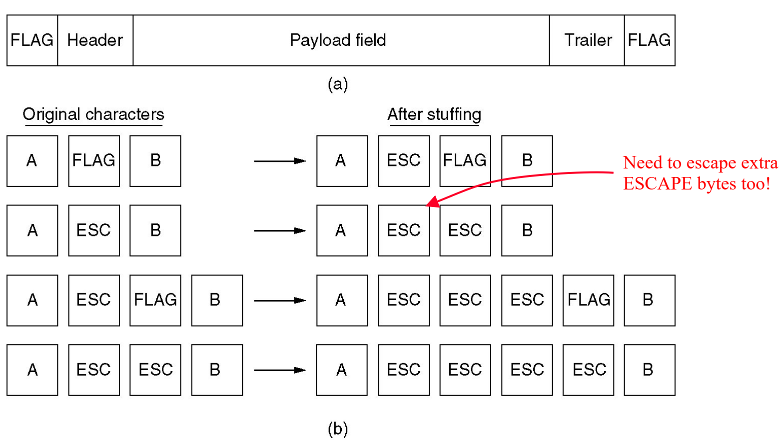
Framing – Character/Byte Count

- Character/Byte count: use a field in the header to specify the number of characters/bytes in the frame
- Problem: Simple but difficult to resynchronize after an error



Framing- Flag Byte with Byte Stuffing

- Flag byte starts and ends each frame (used in PPP)
  - Longer, but easy to resynchronize after error.



(a) A frame delimited by flag bytes.
(b) Four examples of byte sequences before and after stuffing.

Framing- Flag Byte with Bit Stuffing

- **Problem with flag byte and byte stuffing: not all character codes use 8-bit characters.**
- **To allow a data frame with arbitrary number of bits and allow character codes with arbitrary number of bits per character**
  - **Flag byte: 01111110 (0X7E), which has consecutive 1s**
  - **ESC: 0 (if 5 consecutive 1s)**

How to do de-stuffing?

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in receiver's memory after de-stuffing.

Example

Q: The following character encoding is used in a data link protocol:

X: 01101111, Y: 11110011, FLAG 01111110, ESC: 11100000

Show the bit sequence transmitted in binary for the four-character frame: X Y ESC FLAG (space omitted)

(a) Flag bytes with byte stuffing

(b) Starting and ending flag bytes, with bit stuffing

Error Control

- Error control repairs frames that are received in error
  - Requires errors to be detected at the receiver
  - Typically retransmit the unacknowledged frames
  - Timer protects against lost acknowledgements
- Detecting errors and retransmissions are next topics.

Error Control (2)

- What are errors?
  - Single errors vs. errors in burst
  - Advantages vs. disadvantages (if same BER)
    - Many 1-bit-error blocks vs. one 100-bit-error block
- Error-Correcting Codes
- Error-Detecting Codes
- What if a whole frame vanished or a whole packet is lost?
  - Flow control: Acknowledgement, retransmission, sequencing

Error Detection and Correction: Common Methods

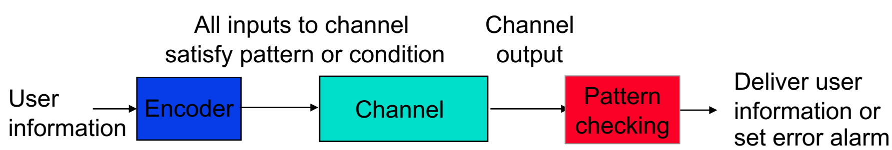
- Error codes add structured redundancy to data so errors can be either detected, or corrected.
- Error correction codes:
  - Hamming codes »
  - Binary convolutional codes »
  - Reed-Solomon and Low-Density Parity Check codes
    - Mathematically complex, widely used in real systems
- Error detection codes:
  - Parity »
  - Checksums »
  - Cyclic redundancy codes (CRC) »

Codeword and Hamming Distance

- A n-bit codeword: a frame of m-bit data plus r-bit redundant check bits (checksum); n = m + r
- The number of bit positions in which two codewords differ is called the Hamming distance. Or, it can be defined as the minimum bit flips to turn one valid codeword into any other valid one.
- Example: 10001001 and 10110001 using XOR
- Example with 4 codewords of 10 bits (n=2, k=8):
  - 0000000000, 0000011111, 1111100000, and 1111111111
  - Hamming distance is 5

Key Idea

- All transmitted data blocks ("codewords") satisfy a pattern
  - If received block doesn't satisfy pattern, it is in error
  - Redundancy(r)
- Blindspot: when channel transforms a codeword into another codeword



Error Detecting Codes – Parity bit

- **Parity bit: to make the number of 1 bits in a codeword even or odd (r = 1)**
  - **Example: 10110100**

Can a parity bit used to detect a single-bit error in a codeword?

Can a parity bit used to detect a double-bit error in a codeword? Triple…?

Can a parity bit used to correct a single-bit error in a codeword?

Parity bit used in ASCII code

How Good is the Single Parity Check Code?

- ***Redundancy***:
  - Single parity check code adds 1 redundant bit per m information bits: overhead = $1/(m + 1)$
- ***Coverage***: all error patterns with odd # of errors can be detected
  - An error pattern is a binary $(m + 1)$-tuple with 1s where errors occur and 0's elsewhere
  - Of 2^{m+1} binary $(m + 1)$-tuples, ½ are odd, so 50% of error patterns can be detected
- **Is it possible to detect more errors if we add more check bits?**
- **Yes, with the right codes**

What if Bit Errors Are Random?

- **Many transmission channels introduce bit errors at random, independently of each other, and with probability p**
- **Some error patterns are more probable than others:**

$$P[10000000] = p(1-p)^7 = (1-p)^8 \left(\frac{p}{1-p}\right) \text{ and}$$

$$P[11000000] = p^2(1-p)^6 = (1-p)^8 \left(\frac{p}{1-p}\right)^2$$

Error-Detecting Codes – CRC Base

- Cyclic Redundancy Check (CRC) use polynomial code, which is based on treating bit strings as representation of polynomials with coefficients of 0 and 1 only.
- A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^k-1 to x^0. Such a polynomial is said to be of degree k-1

  Example: 110001

  What is its degree?

  What are its polynomial and coefficients?
- Binary polynomial arithmetic is done by per-bit XOR

  Example: 10011011 + 11001010

  11110000 - 10100110

Binary Polynomial Arithmetic

- Binary vectors map to polynomials

$$(i_{k-1}, i_{k-2}, \ldots, i_2, i_1, i_0) \rightarrow i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_2x^2 + i_1x + i_0$$

Addition:

$$(x^7 + x^6 + 1) + (x^6 + x^5) = x^7 + x^6 + x^6 + x^5 + 1$$
$$= x^7 + (1+1)x^6 + x^5 + 1$$
$$= x^7 + x^5 + 1 \quad \text{since } 1+1 \bmod 2 = 0$$

Multiplication:

$$(x + 1)(x^2 + x + 1) = x(x^2 + x + 1) + 1(x^2 + x + 1)$$
$$= x^3 + x^2 + x + (x^2 + x + 1)$$
$$= x^3 + 1$$

Error-Detecting Codes – CRC Idea

- Both the sender and the receiver agree upon a **generator polynomial *G(x)*** as 1 xxx…x 1 in advance.

  Given a frame of m bits (a polynomial ***M(x)***), the idea of CRC is to append a **checksum** to the end of the frame in such a way that the **polynomial represented by the checksumed frame** is divisible by ***G(x)***. When the receiver gets the checksummed frame, it tries dividing it by ***G(x)***. If there is a remainder, there has been a transmission error.

What kind of errors can be detected?

How the checksum is calculated?

Error-Detecting Codes – CRC Algorithm

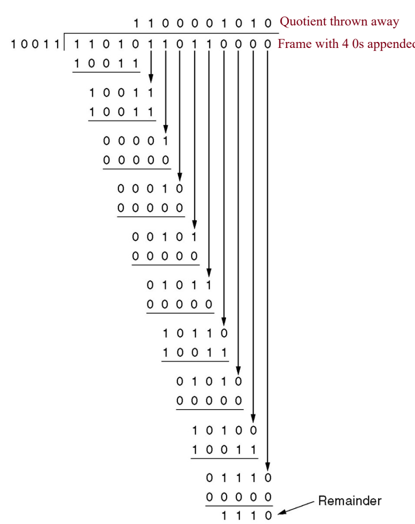
- Let r be the degree of G(x). Append r 0s to the low-order end of the frame, resulting ***x^r M(x)*** .
- Divide the bit string of ***G(x)*** into the bit string of ***x^r M(x)***, using modulo 2 division.
- Subtract the reminder from the bit string of ***x^r M(x)*** using modulo 2 subtraction. The result is the checksummed frame to be transmitted, called T(x).

T(x) is divisible by G(x)!

Frame : 1 1 0 1 0 1 1 0 1 1

Generator: 1 0 0 1 1 What is the generator polynomial?

Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



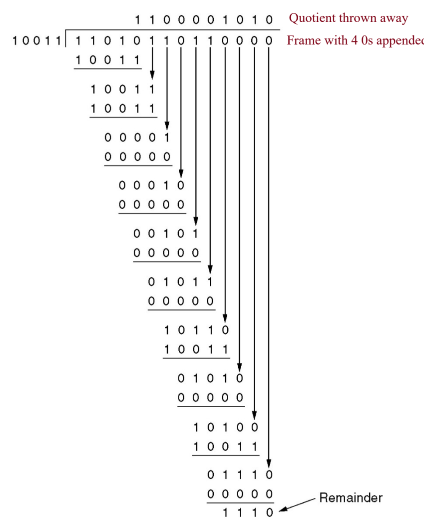
Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

Error-Detecting Codes – CRC Algorithm (cont.)

Q: What is the actual bit string to be transmitted?

If the third bit from the left is inverted during transmission, how this error is detected at the receiver's end?

Frame : 1 1 0 1 0 1 1 0 1 1
Generator: 1 0 0 1 1
Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



Why data link protocols almost always put the CRC in a trailer rather than in a header?

Computed with simple shift/XOR circuits

Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

Error-Detecting Codes – CRC Analysis

- **What kind of errors will be detected?**
- **Imagine that a transmission error occurs, so that instead of *T(x)* arriving, *T(x) + E(x)* arrives. Each 1 bit in *E(x)* corresponds to a bit that has been inverted**

  Example: 11001 (sent) ---- > 10101 (received)

  If *E(x)* is divisible by *G(x)*, the error will slip by! So, how we select *G(x)*?!

Designing Good Polynomial Codes

- **Select generator polynomial so that likely error patterns are not multiples of *G(x)***
- ***Detecting Single Errors***
  - **$E(x) = x^i$ for error in location $i + 1$**
  - **If *G(x)* has more than 1 term, it cannot divide x^i**
- ***Detecting Double Errors***
  - **$E(x) = x^i + x^j = x^i(x^{j-i}+1)$ where $j>i$**
  - **If *G(x)* has more than 1 term, it cannot divide x^i**
  - **If G(x) is a *primitive* polynomial, it cannot divide x^m+1 for all $m<2^{n-k}-1$ (Need to keep codeword length less than $2^{n-k}-1$)**
    - **x^15+x^14+1 won't divide x^k + 1 for k < 32, 768**
  - **Primitive polynomials can be found by consulting coding theory books**

Standard Generator Polynomials

- **CRC-8:**

  $= x^8 + x^2 + x + 1$ — ATM

- **CRC-16:**

  $= x^{16} + x^{15} + x^2 + 1$
  $= (x + 1)(x^{15} + x + 1)$ — Bisync

- **CCITT-16:**

  $= x^{16} + x^{12} + x^5 + 1$ — HDLC, XMODEM, V.41

- **CCITT-32:** — IEEE 802 (Ethernet), DoD, V.42

  $= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

All bursts of length 32 or less and all bursts affecting an odd number of bits

Example 2

Q: **How a bit loss be detected in bit stuffing? Can it always be detected?**

FLAG	Header	Payload field	Trailer	FLAG

Error Correcting Codes – An Error-correcting Code

- **Given a complete list of the valid codewords, the minimum hamming distance of any two codewords is defined as the hamming distance of the complete code**
- **Example: a complete code with four legal codewords of 0000000000, 0000011111, 1111100000, 1111111111**

  What is the hamming distance of the code?

  How many error-bits at most can it correct?

  How many error-bits at most can it detect?

  What is the hamming distance of a code with a parity bit?

  What is the relationship between the hamming distance and the number of error-bits to be detected and corrected?

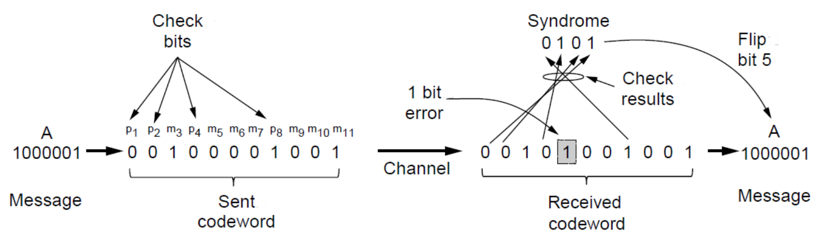
Error Correcting Codes – Low Limit on r

- A n-bit codeword: a frame of m-bit data plus r-bit redundant check bits (n = m + r)
- What is the lower limit on the number of bits needed to correct single-bit errors in a n-bit codeword?
  - (n+1) 2^m <= 2^n

Error Correction – Hamming code

Hamming code gives a simple way to add check bits and correct up to a single bit error:

- Check bits are parity over subsets of the codeword
- Recomputing the parity sums (syndrome) gives the position of the error to flip, or 0 if there is no error



(11, 7) Hamming code adds 4 check bits and can correct 1 error

Error Correction – Hamming code (Cont)

- A n-bit codeword: a frame of m-bit data plus r-bit redundant check bits (n = m + r)
- Use of a Hamming code to detect a& correct a single-bit error in a codeword
  - The bits that are powers of 2 are used as check bits.
  - The rest are filled up with the data bits
  - Each check bit forces the parity of some collection of bits, including itself, to be even (or odd)
  - To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2
  - A bit is checks by just those check bits occuring in its expansion (11 = 1 + 2 + 8)
- Example: a n-bit codeword containing a 7-bit data 1001000

  1001000 → 00110010000 (even-parity used)

  How to correct it if 00100010000 is received instead?
  How to correct it if 00110010001 is received instead?
  How many check bits needed to d&c a single error in a 10-bit message

Error-Correcting Codes – Burst Errors

- What to do if errors come in burst, instead of isolated single-bit errors?

Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

What is the maximum length of a burst that can be corrected in a sequence of k codewords?

Use of a Hamming code to correct burst errors.

Error Detecting Codes vs. Error Correcting Codes

- **Consider a channel on which errors are isolated and the error rate is 10^-6. Let the block size (m) be 1000 bits.**

  How many total bits required to provide single-bit error (per block) corrections for 1 Mbits (10^6 bits) data? (extra 10000)

  How many total bits required to provide the error detection + retransmission? (extra 2001)

  Why wireless networks prefer error correction while wired networks may go for error detection and retransmission?

  What kind of applications prefer error correction instead of detection?

Error Bounds – Hamming distance

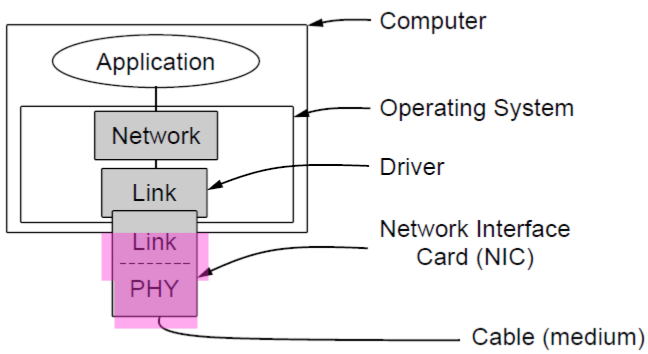
Hamming distance is the minimum bit flips to turn one valid codeword into any other valid one.

Bounds for a code with distance:

- **2d+1 – can correct d errors (e.g., 2 errors above)**
- **d+1 – can detect d errors (e.g., 4 errors above)**

Link layer environment (1)

Commonly implemented as NICs and OS drivers; network layer (IP) is often OS software



Link layer environment (2)

° **Link layer protocol implementations use library functions**

- **See code (`protocol.h`) for more details**

Group	Library Function	Description
Network layer	from_network_layer(&packet) to_network_layer(&packet) enable_network_layer() disable_network_layer()	Take a packet from network layer to send Deliver a received packet to network layer Let network cause “ready” events Prevent network “ready” events
Physical layer	from_physical_layer(&frame) to_physical_layer(&frame)	Get an incoming frame from physical layer Pass an outgoing frame to physical layer
Events & timers	wait_for_event(&event) start_timer(seq_nr) stop_timer(seq_nr) start_ack_timer() stop_ack_timer()	Wait for a packet / frame / timer event Start a countdown timer running Stop a countdown timer from running Start the ACK countdown timer Stop the ACK countdown timer

Service Models

- The *service model* specifies the information transfer service layer-n provides to layer-(n+1)
- The most important distinction is if the service is:
  - Connection-oriented
  - Connectionless
- Service model possible features:
  - Arbitrary message size or structure
  - Sequencing and Reliability
  - Timing and Flow control
  - Multiplexing
  - Privacy, integrity, and authentication

Reliability & Sequencing

- *Reliability*: what transmission is reliable?
  - *Sequencing*: Are messages or information stream delivered in order?
  - *duplication*
- How to provide reliable communication?
  - Examples: TCP and HDLC

Utopian Simplex Protocol

An optimistic protocol (p1) to get us started

- **Assumes no errors, and receiver as fast as sender**
- **Considers one-way data transfer**

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

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

 Sender loops blasting frames

  }
```

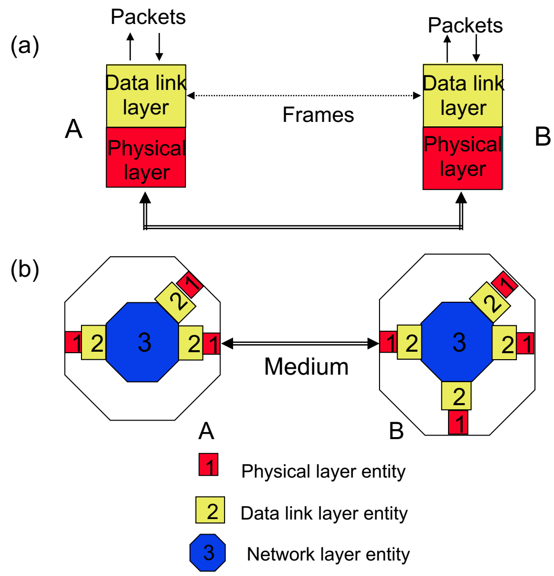
```
void receiver1(void)
{
  frame r;
  event_type event;

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

- **That's it, no error or flow control …**

Flow Control

- **Prevents a fast sender from out-pacing a slow receiver**
  - **If destination layer-(n+1) does not retrieve its information fast enough, destination layer-n buffers may overflow**
- ***Flow Control* provide backpressure mechanisms that control transfer according to availability of buffers at the destination**
- **Examples: HDLC (Link Layer) and TCP (Transport Layer)**
  - **Rare in the Link layer as NICs run at "wire speed"**
    Receiver can take data as fast as it can be sent

Error Control in Data Link Layer



- Data Link operates over *wire-like*, directly-connected systems
- Frames can be corrupted or lost, but arrive in order
- Data link performs error-checking & retransmission
- Ensures error-free packet transfer between two systems

Elementary Data Link Protocols

How to make sure all frames are eventually delivered to the network layer at the destination in the proper order?

- A Simplex Stop-and-Wait Protocol
- A Simplex Protocol for a Noisy Channel

Stop-and-Wait – Error-free channel

Protocol (p2) ensures sender can't outpace receiver:

- **Receiver returns a dummy frame (ack) when ready**
- **Only one frame out at a time – called stop-and-wait**
- **We added flow control!**

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

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

Sender waits to for ack after passing frame to physical layer

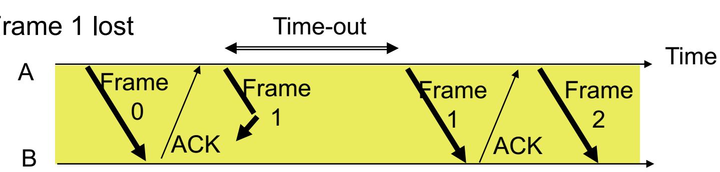
```
void receiver2(void)
{
  frame r, s;
  event_type event;
  while (true) {
       wait_for_event(&event);
       from_physical_layer(&r);
       to_network_layer(&r.info);
       to_physical_layer(&s);
  }
}
```

Receiver sends ack after passing frame to network layer

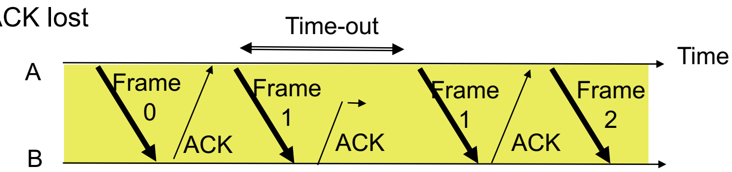
What if the Channel is Noisy?

Need for Sequence Numbers!

(a) Frame 1 lost



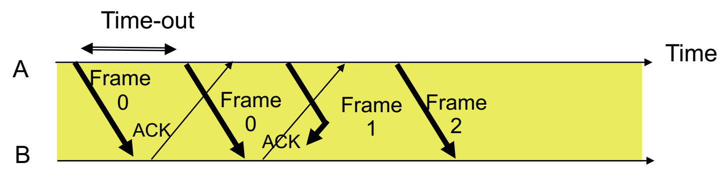
(b) ACK lost



- **In cases (a) & (b) the transmitting station A acts the same way**
- **But in case (b) the receiving station B accepts frame 1 twice**
- **Question: How is the receiver to know the second frame is also frame 1?**
- **Answer:** *Add frame sequence number in header*

What if the Channel is Noisy? (cont.)

(c) Premature Time-out



- **The transmitting station A misinterprets duplicate ACKs**
- **Incorrectly assumes second ACK acknowledges Frame 1**
- **Question: How is the receiver to know second ACK is for frame 0?**
- **Answer: *Add frame sequence number in ACK header***

Q: What is the minimum # of bits required for the Seq#?

Q2: What is so-called *Proactive ACK Scheme*

A Simplex Protocol for a Noisy Channel

- **PAR: a positive acknowledgement w/ retransmission protocol**

Q1: how to avoid packet lose?

Q2: how to avoid packet duplicate?

Q3: What is the minimum # of bits required for the Seq#?

Q4: what happen if ACK is garbled one?

```
/* 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) {
               stop_timer(s.ack);            /* turn the timer off */
               from_network_layer(&buffer);  /* get the next one to send */
               inc(next_frame_to_send);      /* invert next_frame_to_send */
          }
     }
  }
}
```

Continued →

A Simplex Protocol for a Noisy Channel (ctd.)

```
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);           /* send acknowledgement */
      }
  }
}
```


Sliding Window Protocols (ARQ)

How about two-way data frame transmission?

- **A One-Bit Sliding Window Protocol (Stop-and Wait)**
- **A Protocol Using Go Back N**
- **A Protocol Using Selective Repeat**

ARQ (Automatic Repeat rQquest) protocols **combine error detection, retransmission, and sequence numbering to provide reliability & sequencing**

Sliding Window Protocols - Piggybacking

- Since in the two-way transmission, data frames and ACK frames are interleaving, why not have a free ride of ACK upon a data delivering?
  - How a receiver can tell if the frame is data or ACK?
- For piggybacking, how long should the data link layer wait for a packet onto which to piggyback the ACK?

Metrics: Efficiency, Complexity, Buffer Requirements

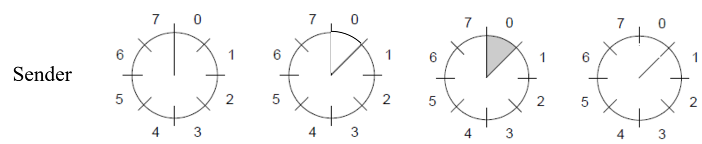
Sliding Window concept (1)

- Sender maintains window of frames it can send
  - Needs to buffer them for possible retransmission
  - Window advances with next acknowledgements
- Receiver maintains window of frames it can receive
  - Needs to keep buffer space for arrivals
  - Window advances with in-order arrivals

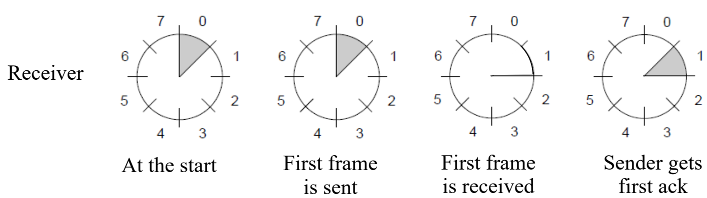
Sliding Window concept (2)

A sliding window advancing at the sender and receiver

- Ex: window size is 1, with a 3-bit sequence number.



What is the buffer size needed in the sender?



Sliding Window concept (3)

- ○ **Larger windows enable <u>pipelining</u> for efficient link use**
  - Stop-and-wait (w=1) is inefficient for long links
  - Best window (w) depends on bandwidth-delay (BD)
  - Want w ≥ 2BD+1 to ensure high link utilization

- ○ **Pipelining leads to different choices for errors/buffering**
  - We will consider <u>Go-Back-N</u> and <u>Selective Repeat</u>

A One-Bit Sliding Window Protocol (Stop and Wait)

- **Window size is 1: a stop-and-wait protocol, bi-directional**

```
/* Protocol 4 (sliding window) is bidirectional. */
#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;                      /* frame expected next */
  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 */
```

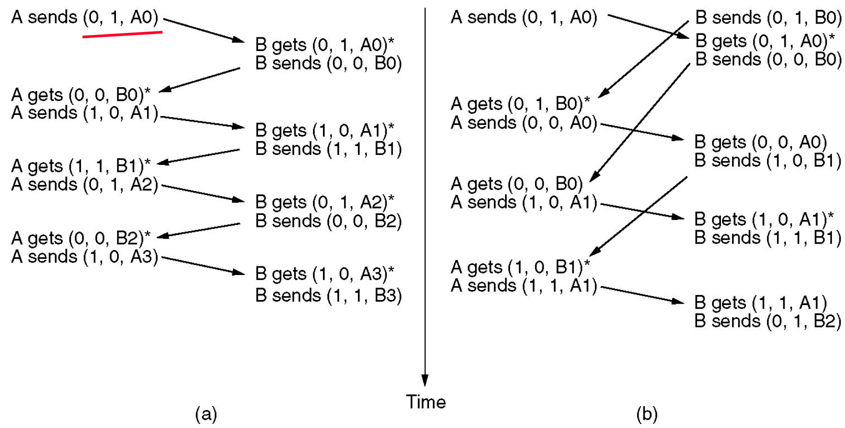
Continued →

A One-Bit Sliding Window Protocol (ctd.)

```
 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);           /* 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 seq number expected next */
        }
        if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
            stop_timer(r.ack);             /* turn the timer off */
            from_network_layer(&buffer);   /* fetch new pkt from network layer */
            inc(next_frame_to_send);       /* invert senderís 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 */
 }
}
```

Can the protocol accept out-of-order frames? Why?

A One-Bit Sliding Window Protocol (2)



Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case (simultaneous sending). The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

What is the problem with the case (b)?

Applications of Stop-and-Wait ARQ

- **IBM *Binary Synchronous Communications protocol* (Bisync): character-oriented data link control**
- ***Xmodem*: modem file transfer protocol**
- ***Trivial File Transfer Protocol* (RFC 1350): simple protocol for file transfer over UDP**

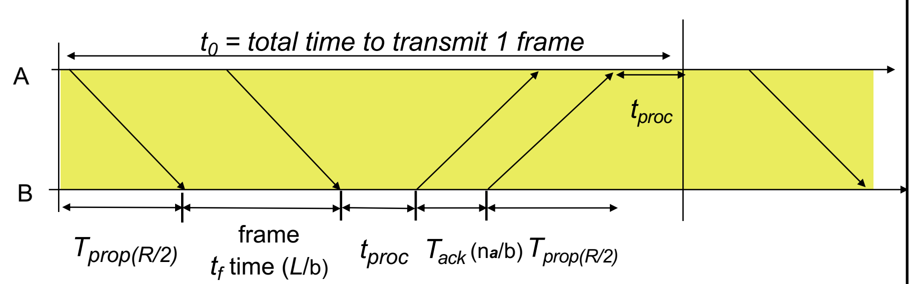
A One-Bit Sliding Window Protocol – Performance 1

- Q1: consider a 50-kbps satellite channel with a 500-msec round-trip delay. Frame size is 1000-bit. What is the best bandwidth utilization (efficiency) of the one-bit sliding window protocol (stop-and-wait)? (20/520)
- Q2: A channel has a bit rate of 10 kbps and a propagation delay of 40 msec. For what range of frame sizes does 1-bit sliding window give an bandwidth utilization (efficiency) of at least 50%?

A One-Bit Sliding Window Protocol – Performance 2

- General model: Given: the channel capacity *b* bits/sec, the frame size *L* bits, the round-trip propagation time *R* sec. What is the time required to transmit a single frame? And what is the line utilization? L/(L + bR)

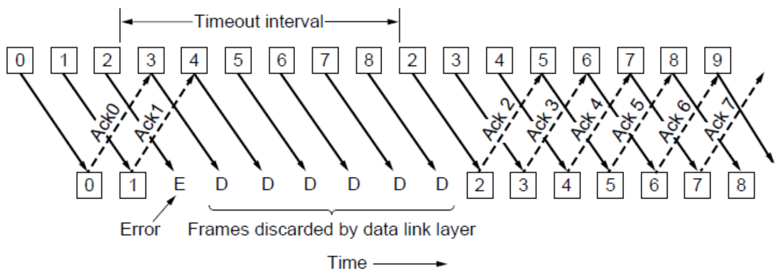
  The combination of a long transit time (high RTT), high bandwidth, and short frame length gives bad efficiency to the stop-and-wait protocol.



Pipelining: why not allow sending more frames before ACKs back?

ARQ: Go-Back-N (1)

- Receiver only accepts/acks frames that arrive in order:
  - Discards frames that follow a missing/errored frame
  - Sender times out and resends all outstanding frames



Go Back N (2)

- Given N-bit sequence numbers, what is the maximum number of frames that can be outstanding in "go back N"?

  MAX_SEQ = 2^N – 1 while there are 2^N sequence numbers. Should the maximum number be MAX_SEQ or MAX_SEQ + 1?

  Example: n = 3; sequence numbers: 0, 1, 2, …, 7

  1) The sender sends 8 frames 0 through 7

  2) A piggybacked ACK for frame 7 eventually back to sender

  3) The sender sends another 8 frames (0 through 7)

  4) Another piggybacked ACK for frame 7 eventually back to sender

  Can the sender tell if all 8 frames in second batch got received or all got lost?

Go-Back-N (3)

- Tradeoff made for Go-Back-N:
  - Simple strategy for receiver; needs only 1 frame
  - Wastes link bandwidth for errors with large windows; entire window is retransmitted
- Implemented as p5 (see code in book)

Sliding Window Protocol Using Go Back N (cont.)

```
/* Protocol 5 (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"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <=b < c circularly; false otherwise. */
  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;                           /* scratch variable */

  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);             /* transmit the frame */
  start_timer(frame_nr);             /* start the timer running */
}
```

Continued →

Sliding Window Protocol Using Go Back N (cont.)

```
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;

  enable_network_layer();           /* allow network_layer_ready events */
  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 */
```

Continued →

Sliding Window Protocol Using Go Back N (cont.)

```
while (true) {
  wait_for_event(&event);          /* four possibilities: see event_type above */

  switch(event) {
    case network_layer_ready:      /* the network layer has a packet to send */
         /* Accept, save, and transmit a new frame. */
         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 */
          }
```

Continued →

Sliding Window Protocol Using Go Back N (cont.)

```
            /* Ack n implies n – 1, n – 2, etc.  Check for this. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                  /* Handle piggybacked ack. */
                  nbuffered = nbuffered – 1; /* one frame fewer buffered */
                  stop_timer(ack_expected); /* frame arrived intact; stop timer */
                  inc(ack_expected);      /* contract sender's window */
            }
            break;

      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 */
            }
    }
    if (nbuffered < MAX_SEQ)
            enable_network_layer();
    else
            disable_network_layer();
  }
}
```

What happens if not much reverse traffic?

What is the maximum # of frames can be outstanding?

Applications of Go-Back-N ARQ

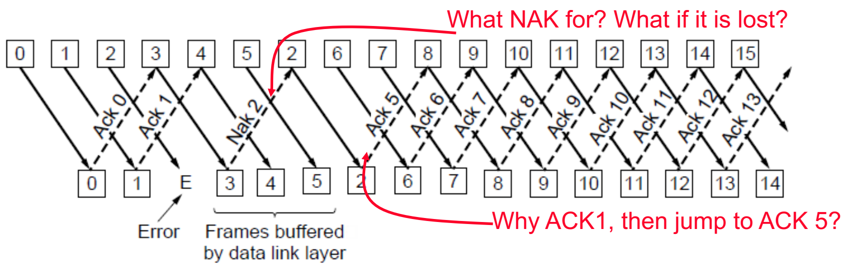
° ***HDLC*** **(High-Level Data Link Control): bit-oriented data link control**

° ***V.42 modem*****: error control over telephone modem links**

Pipelining - Selective Repeat (1)

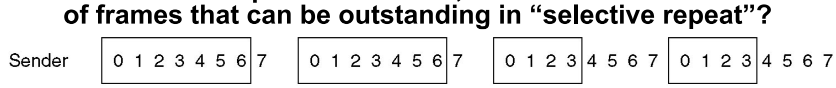
Receiver accepts frames anywhere in receive window

- **<u>Cumulative ack</u> indicates highest in-order frame**
- **NAK (negative ack) causes sender retransmission of a missing frame before a timeout resends window**



A Sliding Window Protocol Using Selective Repeat

- **Given N-bit sequence numbers, what is the maximum number of frames that can be outstanding in "selective repeat"?**



(a) Initial situation with a window size seven.

(b) After seven frames sent and received, but not acknowledged.

Example: n = 3; sequence numbers: 0, 1, 2, …, 7; window size 7

1) Sender sends 7 frames from N0.0 to No. 6

2) Receiver got all, acknowledge, and advances its window to 7, 0, …5

3) However, the acknowledges all got lost

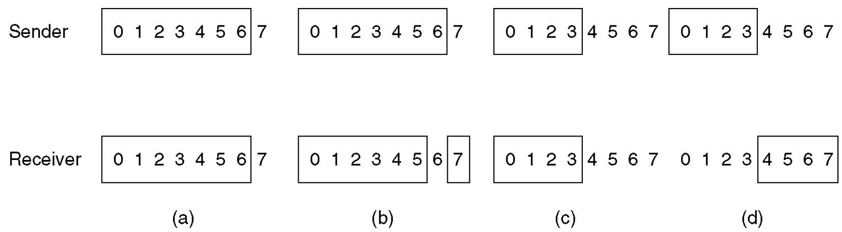
4) Sender times out, retransmit 0

5) Receive accepts 0 (as a new frame 0), as it is in the receiving window, and acknowledge frames 0 to 6 as they were received

6) Happy sender sends 7, 0, 1, ---, 5. Frame 7 is correct, but 0 will be seen as a "false duplicate" in receiver and will be discarded

A Sliding Window Protocol Using Selective Repeat

- **Given N-bit sequence numbers, what is the maximum number of frames that can be outstanding in "selective repeat"?**



(a) Initial situation with a window size seven.
(b) After seven frames sent and received, but not acknowledged.
(c) Initial situation with a window size of four.
(d) After four frames sent and received, but not acknowledged.

The essence of the problem is after the receiver advanced its window, the new range of valid sequence numbers overlapped with the old one. The receiver cannot distinguish a duplicate from a new frame. *Then, what to do*?

A Sliding Window Protocol Using Selective Repeat (1)

```
/* Protocol 6 (nonsequential receive) accepts frames out of order, but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                              /* should be 2^n – 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                         /* no nak has been sent yet */
seq_nr oldest_frame = MAX- SEQ + 1;            /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) II ((c < a) && (a <= b)) II ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                                     /* scratch variable */

  s.kind = fk;                                 /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                            /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;               /* one nak per frame, please */
  to_physical_layer(&s);                       /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                            /* no need for separate ack frame */
}
```

What is the buffer size?

What to do if no much reverse traffic?

What is this for?

Continued →

A Sliding Window Protocol Using Selective Repeat (2)

```
void protocol6(void)
{
  seq_nr ack_expected;                    /* lower edge of sender's window */
  seq_nr next_frame_to_send;              /* upper edge of sender's window + 1 */
  seq_nr frame_expected;                  /* lower edge of receiver's window */
  seq_nr too_far;                         /* upper edge of receiver's window + 1 */
  int i;                                  /* index into buffer pool */
  frame r;                                /* scratch variable */
  packet out_buf[NR_BUFS];                /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];                 /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];               /* inbound bit map */
  seq_nr nbuffered;                       /* how many output buffers currently used */
  event_type event;

  enable_network_layer();                 /* initialize */
  ack_expected = 0;                       /* next ack expected on the inbound stream */
  next_frame_to_send = 0;                 /* number of next outgoing frame */
  frame_expected = 0;
  too_far = NR_BUFS;
  nbuffered = 0;                          /* initially no packets are buffered */
  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

Continued →

A Sliding Window Protocol Using Selective Repeat (3)

```
while (true) {
  wait_for_event(&event);                 /* five possibilities: see event_type above */
  switch(event) {
    case network_layer_ready:             /* accept, save, and transmit a new frame */
        nbuffered = nbuffered + 1;        /* expand the window */
        from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
        send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
        inc(next_frame_to_send);          /* advance upper window edge */
        break;

    case frame_arrival:                   /* a data or control frame has arrived */
        from_physical_layer(&r);          /* fetch incoming frame from physical layer */
        if (r.kind == data) {
            /* An undamaged frame has arrived. */
            if ((r.seq != frame_expected) && no_nak)
               send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
            if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
                /* Frames may be accepted in any order. */
                arrived[r.seq % NR_BUFS] = true;     /* mark buffer as full */
                in_buf[r.seq % NR_BUFS] = r.info;    /* insert data into buffer */
                while (arrived[frame_expected % NR_BUFS]) {
                    /* Pass frames and advance window. */
                    to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                    no_nak = true;
                    arrived[frame_expected % NR_BUFS] = false;
                    inc(frame_expected);  /* advance lower edge of receiver's window */
                    inc(too_far);         /* advance upper edge of receiver's window */
                    start_ack_timer();    /* to see if a separate ack is needed */
                }
            }
        }
```

What is this for?

Continued →

A Sliding Window Protocol Using Selective Repeat (4)

```
            if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next  frame  to  send))
                  send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

            while (between(ack_expected, r.ack, next_frame_to_send)) {
                  nbuffered = nbuffered    1;             /* handle piggybacked ack */
                  stop_timer(ack_expected % NR_BUFS);      /* frame arrived intact */
                  inc(ack_expected);                      /* advance lower edge of sender's window */
            }
            break;
      case cksum_err:
            if (no_nak) send_frame(nak, 0, frame_expected, out_buf);/* damaged frame */
            break;
      case timeout:
            send_frame(data, oldest_frame, frame_expected, out_buf);/* we timed out */
            break;
      case ack_timeout:
            send_frame(ack,0,frame_expected, out_buf);      /* ack timer expired; send ack */
   }
   if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
 }
}
```

What is this for?

What tradeoff to be made for the timer and NAK/ACK?

ACK Strategies

- **To transfer a file between two computers, two acknowledgement strategies are possible. In the first one, the file is chopped up into packets, which are individually acknowledged by the receiver. In the second one, the packets are not acknowledged individually, but the entire file is acknowledged when it arrives.**

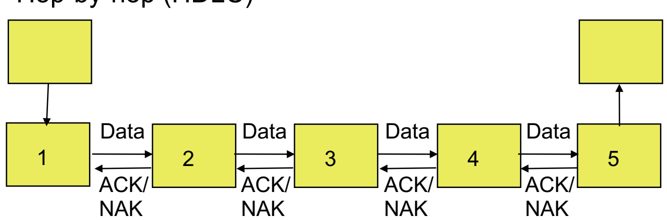
  What are advantages and disadvantage of these two strategies?

End-to-End vs. Hop-by-Hop

- **A service feature can be provided by implementing a protocol**
  - **end-to-end across the network**
  - **across every hop in the network**
- **Example:**
  - **Perform error control at every hop in the network or only between the source and destination?**
  - **Perform flow control between every hop in the network or only between source & destination?**
- **We next consider the tradeoffs between the two approaches**

Which Approach Preferred

Hop-by-hop (HDLC)

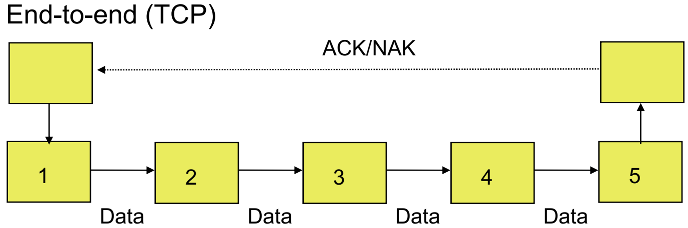


Hop-by-hop cannot ensure E2E correctness

Faster recovery

In situations with infrequent or frequent errors, which one?

End-to-end (TCP)



Simple inside the network

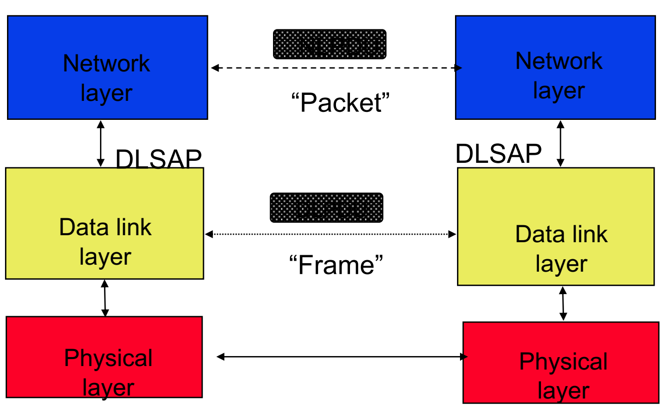
More scalable if complexity at the edge

Protocol Verification

- **Finite State Machined Models**
- **Petri Net Models**

Example 1: High-Level Data Link Control (HDLC)

° **Bit-oriented data link control**
  - **Derived from IBM Synchronous Data Link Control (SDLC)**



High-Level Data Link Control (HDLC)

Frame format for bit-oriented protocols.

Bits	8	8	8	≥ 0	16	8
	0 1 1 1 1 1 1 0	Address	Control	Data	Checksum	0 1 1 1 1 1 1 0

Address: multi-drop or p2p (commands)
Control field gives HDLC its functionality, w/ 3bits for Sequencing
Checksum: x^16 + x^12 + x^5 + 1

What is the appropriate length for "Data" field?

High-Level Data Link Control (2)

Bits	1	3		1	3
(a)	0	Seq		P/F	Next
(b)	1	0	Type	P/F	Next
(c)	1	1	Type	P/F	Modifier

Control field of

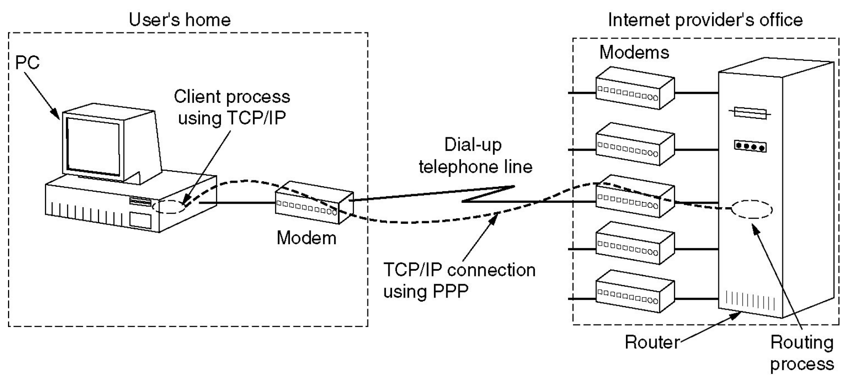
(a) An information frame.

(b) A supervisory frame.

(c) An unnumbered frame.

Example 2: The Data Link Layer in the Internet (PPP)

- Two situations that point-to-point communication is primarily used
  - Router-router leased line connection.
  - Dial-up host-router connections.



A home personal computer acting as an internet host.

PPP – Point to Point Protocol

- PPP provides three features
  - A method for framing, error detection, option negotiation, header compression, and optional reliable transmission
  - A link control protocol (LCP) for bringing lines up, testing, negotiating, and bring them down.
  - A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used; e.g., NCP (Network Control Protocol).

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

The PPP full frame format for unnumbered mode operation.

What is the key difference between PPP framing and HDLC framing?

Bit-oriented vs. byte-oriented (character-oriented)

Reliability option vs. no reliability option

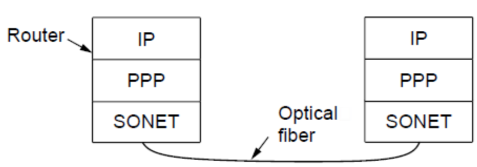
PPP Applications

PPP used in many point-to-point applications

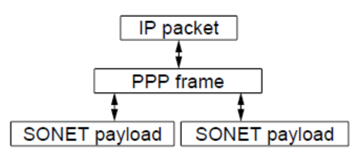
- Telephone Modem Links
- Packet over SONET
  - IP→PPP→SONET
- PPP is also used over shared links such as Ethernet to provide LCP, NCP, and authentication features
  - PPP over Ethernet (RFC 2516)
  - Used over DSL

Packet over SONET

- Packet over SONET is the method used to carry IP packets over SONET optical fiber links
  - Uses PPP (Point-to-Point Protocol) for framing



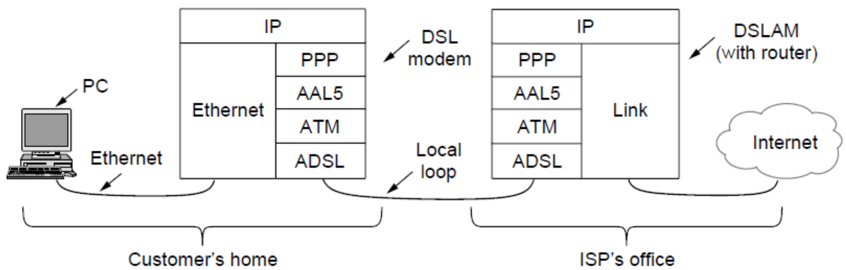
Protocol stacks



PPP frames may be split over SONET payloads

ADSL

- ADSL (Asymmetric Digital Subscriber Loop), widely used for broadband Internet over local loops
  - ADSL runs from modem (customer) to DSLAM (ISP)
  - IP packets are sent over PPP and AAL5/ATM (over)
  - AAL5: ATM adaptation layer 5



Reading

- Chapter 3 of the text