暨南大學
JINAN UNIVERSITY

International School
Jinan University

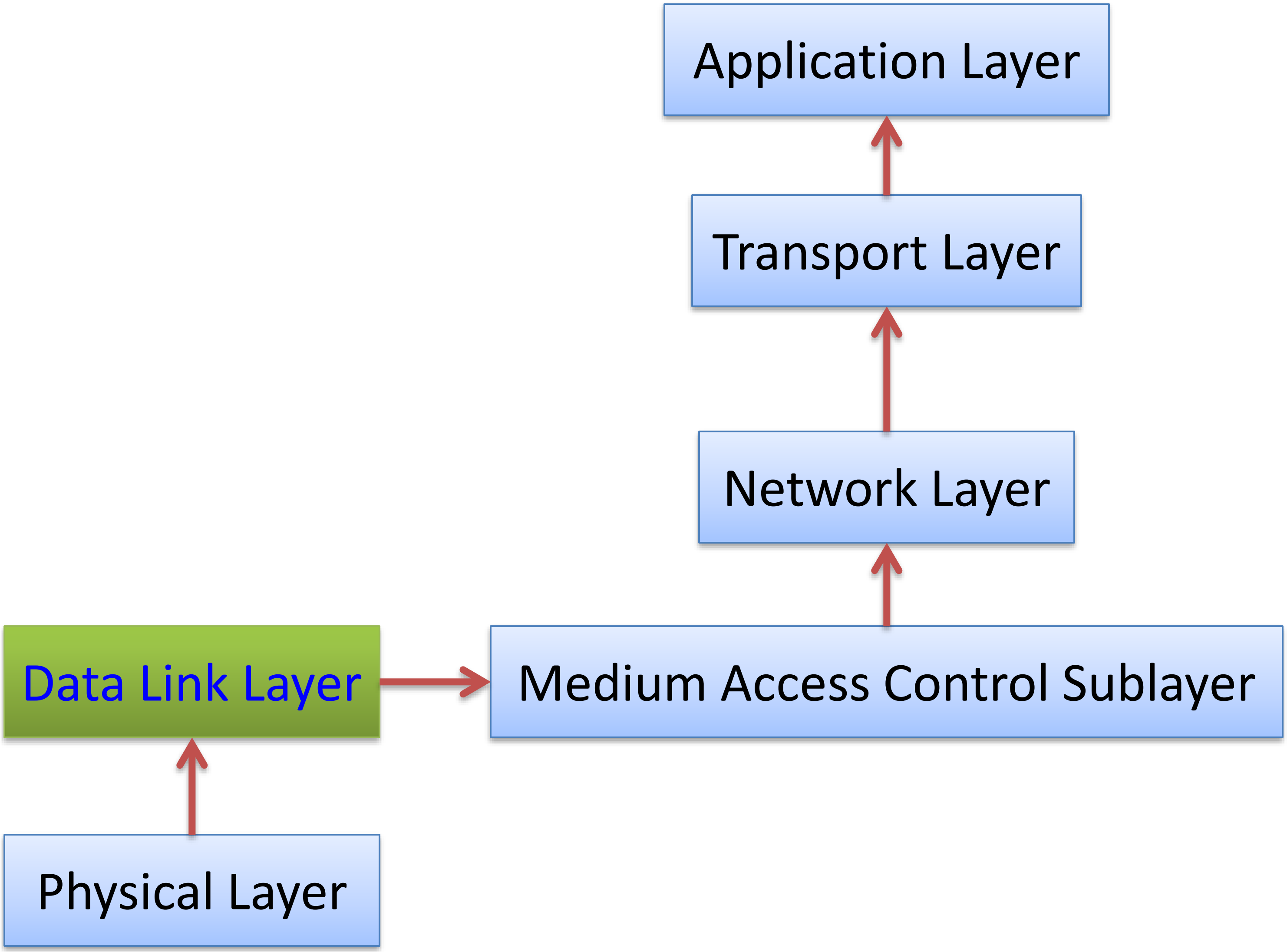
Computer Networks

L3 - Data Link Layer

Lecturer: CUI Lin

Department of Computer Science
Jinan University

Roadmap of this course

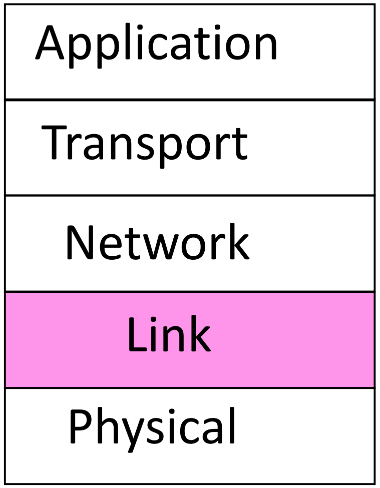


The Data Link Layer

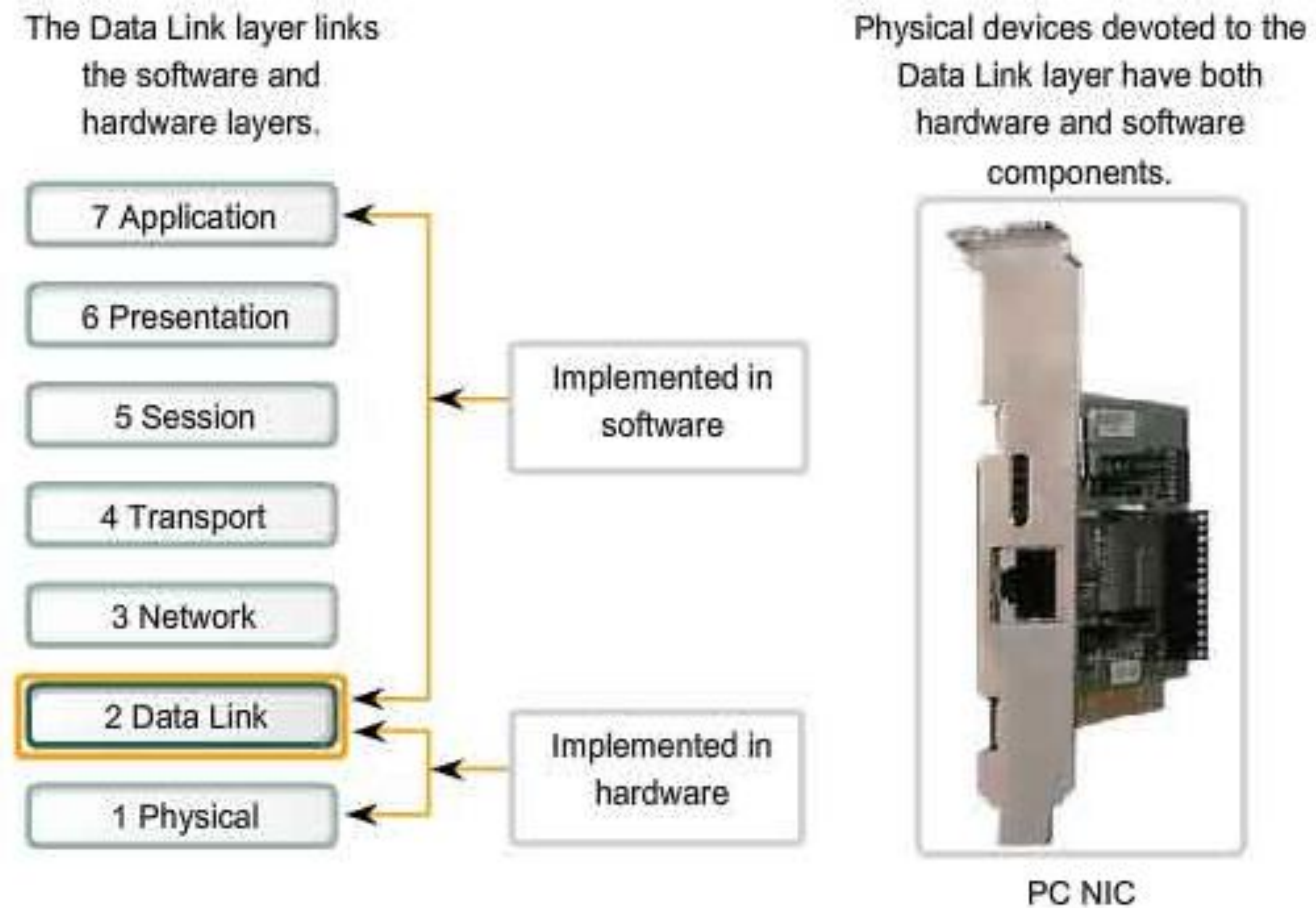
Chapter 2

The Data Link Layer

- Responsible for delivering frames of information over a single link
  - Handles transmission errors
  - Regulates the flow of data



The Data Link Layer



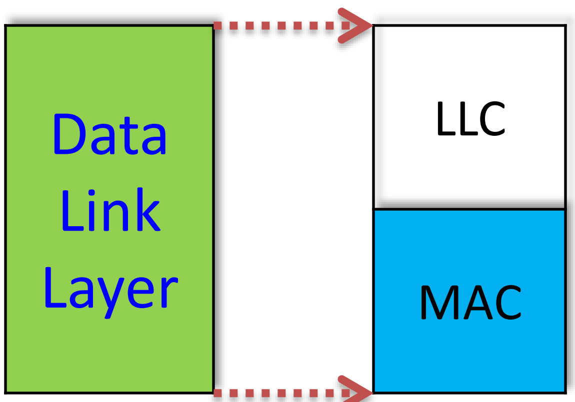
What is data link

- Links are communication channels that connect adjacent (相邻的) machines along the communication path
- Data link layer deals with algorithms for achieving reliable, efficient communication of frames (帧) between two adjacent machines

Two Sublayers of Data Link Layer

Data link layer has two sublayers:

- Logical Link Control (LLC)
  - Provides multiplexing mechanisms for network protocols
  - Optionally provides flow control, acknowledgment, and error notification
- Media Access Control (MAC)
  - Control access to the network medium, specially for shared medium

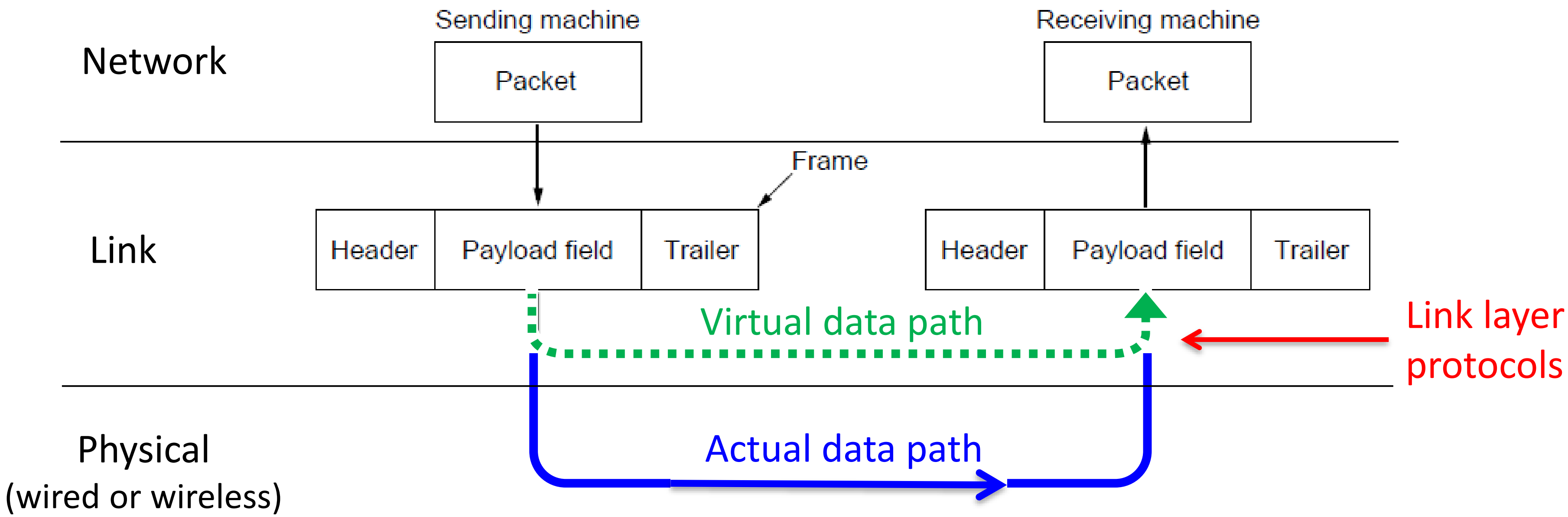


Outline

- Data Link Layer Design Issues
- Framing
- Error Control
  - Error Detection and Correction
- Flow Control
  - Sliding Window Protocols

Frames

- Link layer accepts packets from the network layer, and encapsulates (封装) them into frames that it sends using the physical layer
- Reception is the opposite process



Possible services in link layer

- Unacknowledged connectionless service
  - Frame is sent with no connection & error recovery
  - For low error rate link: Ethernet (以太网)
- Acknowledged connectionless service
  - Frame is sent with retransmissions if needed
  - For unreliable links: IEEE 802.11 (WiFi)
- Acknowledged connection-oriented service
  - Connection is set up
  - For long, unreliable links
  - Suitable for long unreliable link: Satellite channel

Properties and Limitations for a link

- Essential Property:
  - The bits are delivered in exactly the same order in which they are sent
- Limitations
  - Communication circuits make errors occasionally, e.g., noisy channel
  - They have only a finite data rate
  - There is a nonzero propagation delay between the time bit sent and the time it is received

Design issues

- Issues should be considered for link layer protocols:
  - Framing
  - Error control
  - Flow control

Let receivers identify each frame

Make sure each frame is delivered to destination correctly

Regulate flow of frames when speed of sender and reviver are different

Framing (成帧)

- The message unit of transmission in link layer is a frame(帧), which is just a fixed number of bits
- We need methods for breaking up a bit stream into frames
  - marking the start and end of each frame

Error Control

- Must ensure that all frames are eventually delivered to the network layer at the destination and in the proper order
  - Requires errors to be detected at the receiver
  - Typically retransmit unacknowledged frames
  - Timers protect against lost acknowledgements

Error Control

- Three common techniques to do this:
  - Acknowledgments
    - Receiver returns an acknowledgment (ACK) frame to the sender indicating the correct receipt of a frame.
    - Receiver sometimes can also return a negative ACK (NACK) for incorrectly-received frames.
  - Timers: for lost ACK/NACK
  - Sequence Numbers: to suppress duplicated frames

Error Control

- Three common techniques to do this:
  - Acknowledgments
  - Timers: for lost ACK/NACK
    - Retransmission timers are used to resend frames that don't produce an ACK, e.g., ACK lost.
    - When sending a frame, schedule a timer to expire at some time after the ACK should have been returned.
    - If the timer goes off, retransmit the frame.
  - Sequence Numbers: to suppress duplicated frames

Error Control

- Three common techniques to do this:
  - Acknowledgments
  - Timers: for lost ACK/NACK
  - Sequence Numbers: to suppress duplicated frames
    - Retransmissions may cause duplicated frames.
    - To suppress duplicates, add sequence numbers to each frame, so that the receiver can distinguish between new frames and repeats of old frames.
      - Bits used for sequence numbers depend on the number of frames that can be outstanding at any one time.

Flow Control

- Prevents a fast sender from out-pacing a slow receiver
  - Feedback-based flow control: receiver gives feedback on the data it can accept
    - Discuss in the Link layer, e.g., CSMA/CA
  - Rate-based flow control: the protocol has a built-in mechanism that limits the rate
    - Discuss in the Transport layer, e.g., TCP congestion control

Functions of the Data Link Layer

1. Providing a well-defined service interface to the network layer
2. Dealing with transmission errors

   Error Control
3. Regulating the flow of data so that slow receivers are not swamped by fast senders

   Flow Control

Outline

- Data Link Layer Design Issues
- Framing
- Error Control
  - Error Detection and Correction
- Flow Control
  - Sliding Window Protocols

Framing (成帧)

- Physical layer doesn't do much
  - It just pumps bits from one end to the other.
  - Things may go wrong ☹
- Physical layer is not guaranteed to be error free
- Link layer should: detect errors, correct if needed
- The unit of transmission is a frame, which is a fixed number of bits
  - Error detection/correction, as well as flow control, are all based on frames

Framing (成帧)

- Breaking up a bit stream into frames
  - Should be easy for receiver to locate start and end of each frame while using little of the channel bandwidth
- Four methods:
  - Byte count (字节计数法)
  - Flag bytes with byte stuffing (字节填充的标志字节法)
  - Starting and ending flags with bit stuffing (比特填充的标志字节法)
  - Physical layer coding violations (物理层编码违禁法)

Byte Count (字节计数法)

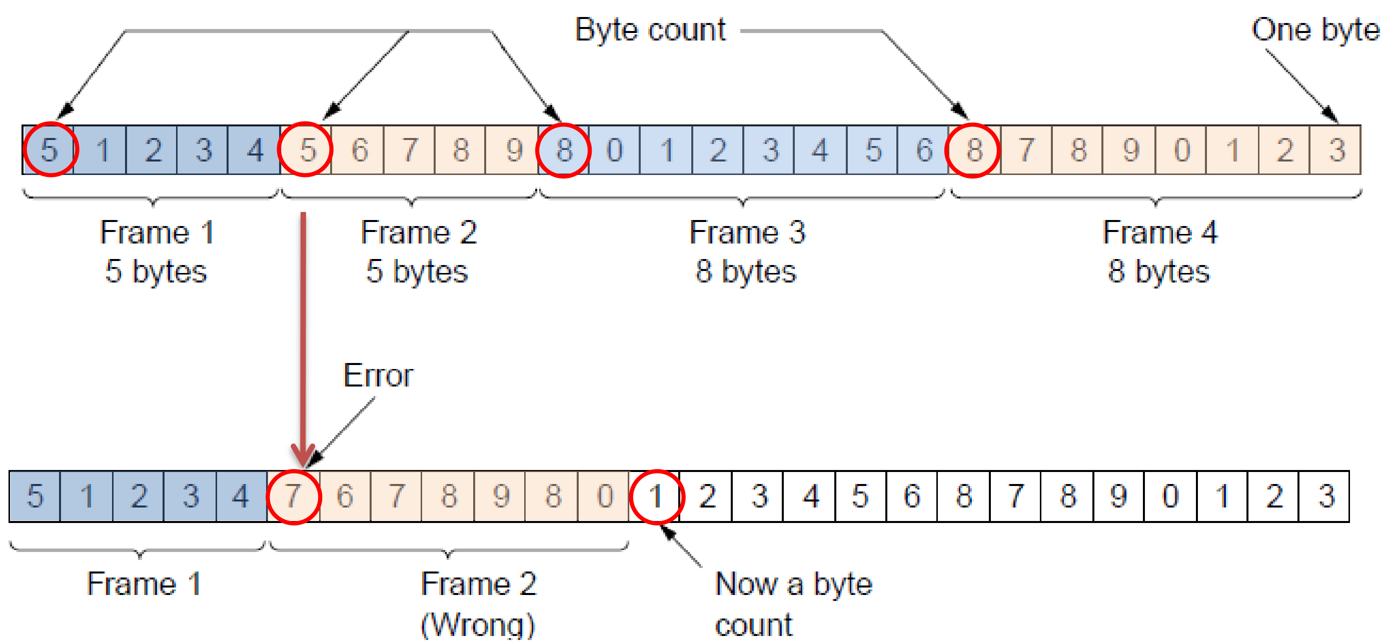
- This method uses a field in the frame header to specify the number of bytes in the frame
- When the data link layer at the destination sees the *byte count*, it knows how many bytes follow and hence where the end of the frame is
- Disadvantage: the count can be garbled (篡改) by a transmission error

Byte Count

- Frame begins with a count of the number of bytes in it
  - Simple, but difficult to resynchronize after an error

Expected case

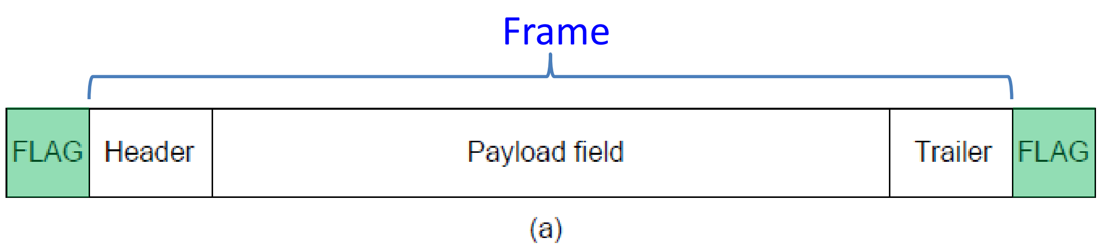
Error case



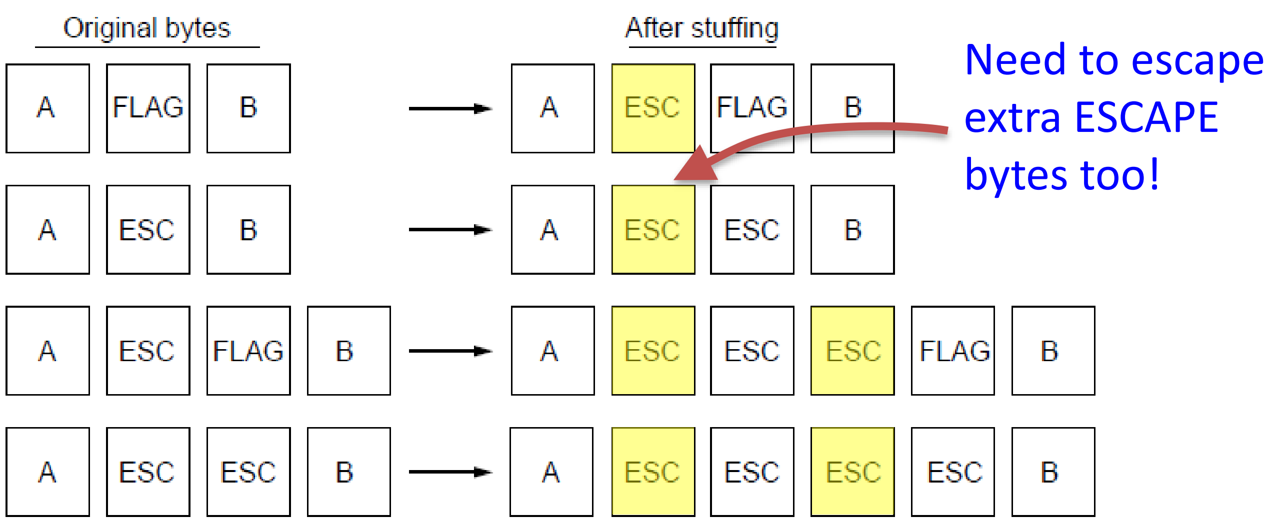
Byte Stuffing (字节填充)

- Special flag bytes delimit frames; occurrences of flags in the data must be stuffed (escaped)
  - Longer, but easy to resynchronize after error

Frame format with flag bytes



Stuffing examples



Bit Stuffing (位填充)

- Stuffing done at the bit level:
  - Frame flag has six consecutive "1"s: 01111110
  - When transmitting data bits, after five "1"s in the data, a "0" is added
  - Upon receiving, a "0" after five "1"s is deleted

Data bits 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Transmitted bits with stuffing 01111110 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 01111110

flag flag

Stuffed bits

After destuffing 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Physical layer coding violations

- Send a signal that doesn’t conform (符合) to any legal bits representation
- Example:
  - In the 4B/5B code, 4 data bits are mapped to 5 signal bits to ensure sufficient bit transitions
  - This means that 16 out of 32 signal possibilities are not used
  - We can use some reserved signals to indicate the start and end of frames

Physical layer coding violations

- 4B/5B Example: use “11000” represents the start of a frame:

Data	Code	Data	Code	Data	Code	Data	Code
0000	11110	0100	01010	1000	10010	1100	11010
0001	01001	0101	01011	1001	10011	1101	11011
0010	10100	0110	01110	1010	10110	1110	11100
0011	10101	0111	01111	1011	10111	1111	11101

- Advantage: easy to find the start and end of frames and there is no need to stuff the data

Combination of framing methods

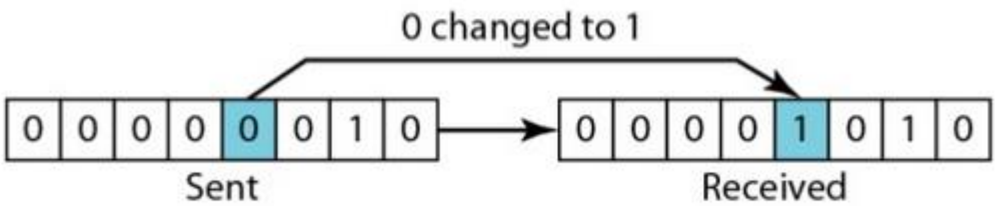
- Many data link protocols use a combination of these methods for safety
- Example: Ethernet and 802.11 (WiFi)
  - A frame begin with a well-defined pattern called a preamble (前导码)
  - The preamble is then followed by a length (i.e. count) field in the header that is used to locate the end of the frame

Outline

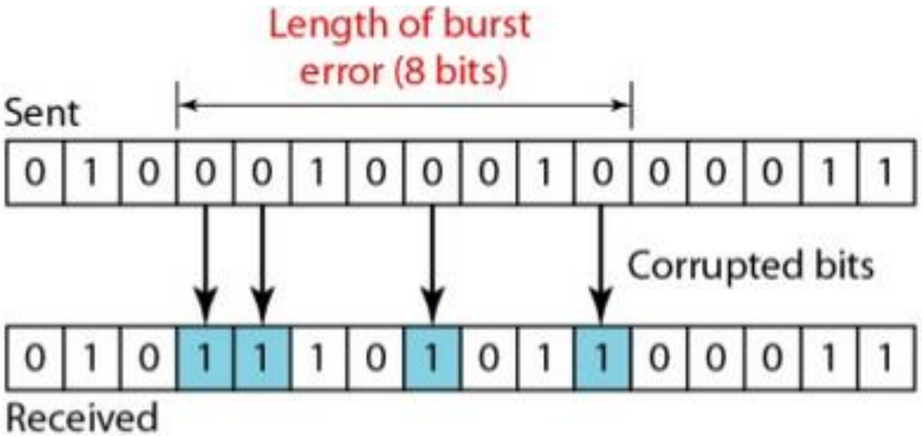
- Data Link Layer Design Issues
- Framing
- Error Control
  - Error Detection and Correction
- Flow Control
  - Sliding Window Protocols

Model of Errors

- Isolated single-bit error: caused by extreme values of thermal noise that overwhelm the signal briefly and occasionally



- Bursts errors: come in burst rather than singly



Error Detection and Correction

- Codes add structured redundancy (冗余) to data, so errors can be either detected, or corrected
- Error correction codes (FEC, Forward Error Correction) (纠错码, 或称前向纠错码)
  - Hamming codes
  - Binary convolutional codes
  - Reed-Solomon and Low-Density Parity Check codes
    - Mathematically complex, widely used in real systems
- Error detection codes (检错码)
  - Parity
  - Checksums
  - Cyclic redundancy codes (CRC)

Error Detection and Correction

- Code turns data of m bits into codewords of *m* data bits and *r* redundant bits (i.e., check bits)
- Codeword
  - Let the total length of a block be n ($n=m+r$)
  - Referred as (n, m) code
- Code rate
  - The proportion of the data-stream that is useful (non-redundant), namely m/n
  - Totally $2^m/2^n = 1/2^r$ possible legal messages

Error Bounds - Hamming distance

- Hamming distance (海明距离) is the minimum bit flips to turn one valid codeword into any other valid one
  - Example with 4 codewords of 10 bits:
    - 0000000000, 0000011111, 1111100000, and 1111111111
    - Hamming distance is 5
- If two codewords are *d* bits apart, *d* errors are required to convert one to the other

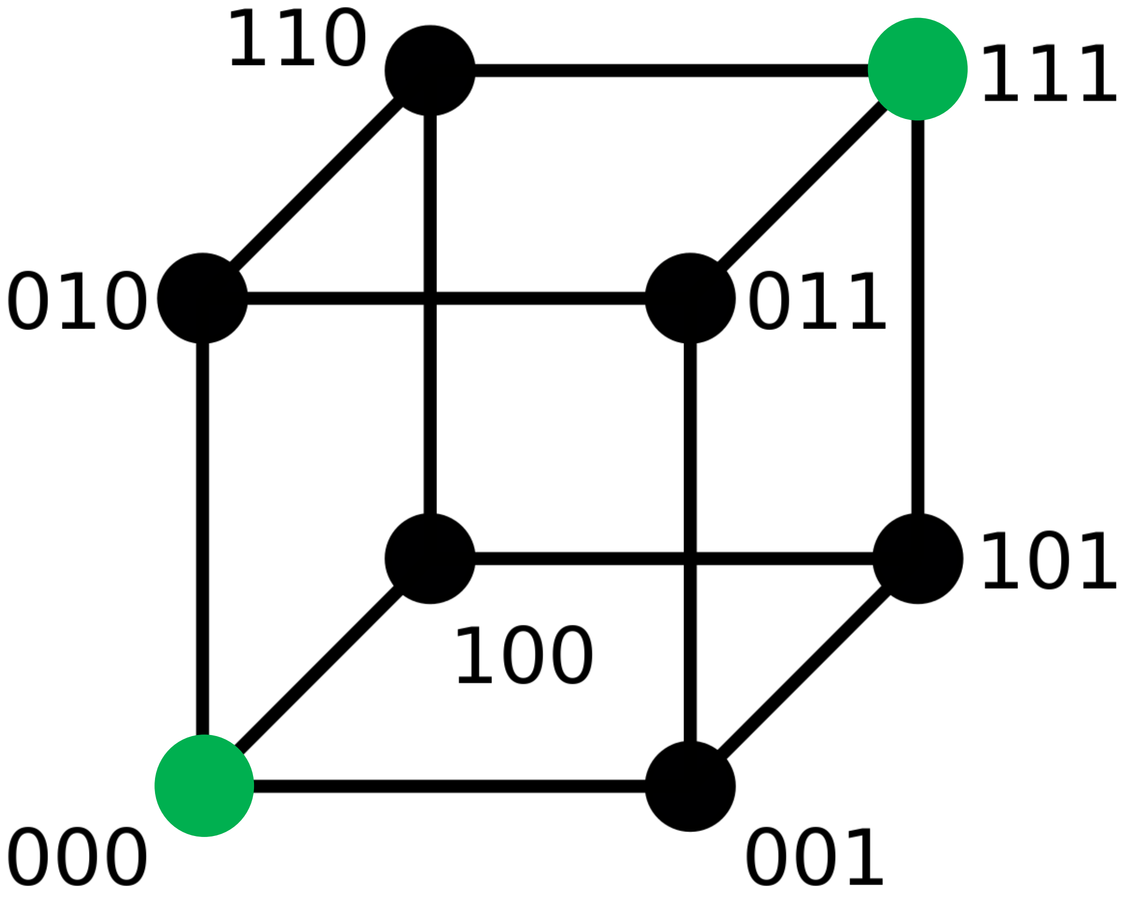
Hamming distance

- There are 2^m possible data words are legal
- By choosing check bits carefully, the resulting codeword will have a large hamming distance
- The larger the hamming distance, the better the codes are able to detect errors

Hamming distance

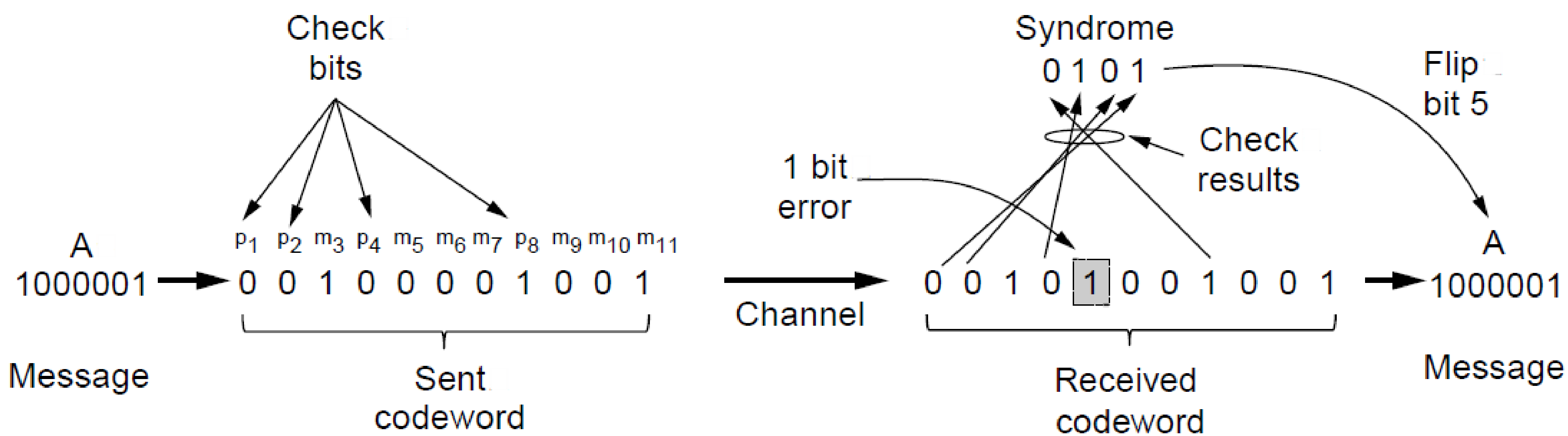
- To detect *d* bit errors requires having a Hamming distance of at least *$d + 1$* bits (e.g., 4 errors in previous example)
- To correct *d* errors requires distance of *$2d + 1$* bits
  - Intuitively, after *d* errors, the garbled messages is still closer to the original message than any other legal codeword (e.g., 2 errors in previous example)

Hamming Distance Example



Error Correction – Hamming code

- Hamming code (海明码) 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 gives the position of



(11, 7) Hamming code adds 4 check bits and can correct 1 error, distance 3

Ref: [Wiki]

Error-Detecting Codes

Three common error-detecting codes

- Parity
- Checksums
- Cyclic Redundancy Checks (CRCs)

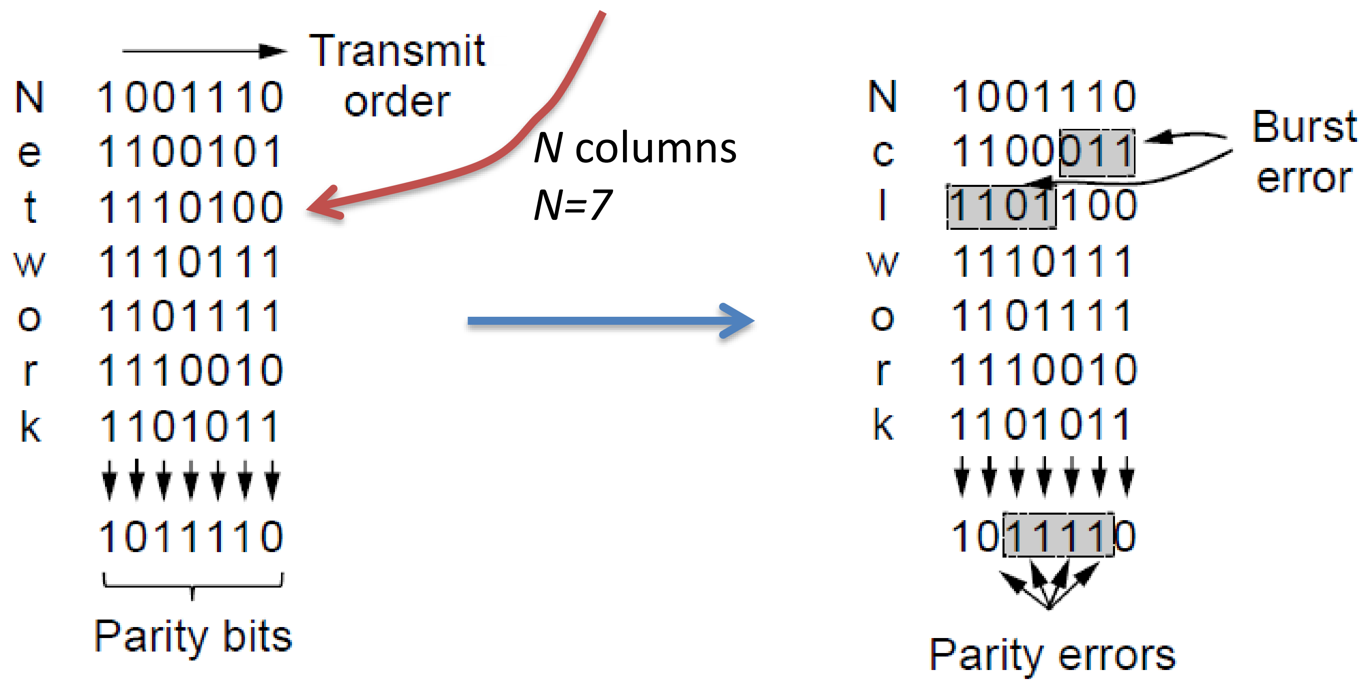
Error Detection – Parity

- Parity (奇偶) bit is added as the modulo 2 sum of data bits
  - Equivalent to XOR; this is even parity
  - Ex: 1110000 → 11100001
  - Detection checks if the sum is wrong (an error)

- Simple way to detect an *odd* number of errors
  - Ex: 1 error, 11100101; detected, sum is wrong
  - Ex: 3 errors, 11011001; detected sum is wrong
  - Ex: 2 errors, 11101101; *not detected*, sum is right!
- Error can also be in the parity bit itself
- Random errors are detected with probability 50%

Error Detection – Parity

- Interleaving (交错校验) of *N* parity bits detects burst errors up to *N*
  - Each parity sum is made over non-adjacent bits
  - An even burst of up to *N* errors will not cause it to fail



Error Detection – Checksums

- Checksum (校验和) means a group of check bits associated with a message, regardless of how they are calculated
- The checksum is usually placed at the end of the message, as the complement of the sum function.
- Errors may be detected by summing the entire received codeword, both data and checksum
  - If the result comes out to be zero, no error has been detected
- Example: 16bits Internet checksum

Codes for TCP/IP checksum

Cyclic Redundancy Check

- CRC (循环冗余校验码): the most popular error detection code at the link layer is based on polynomial code
- Allows us to acknowledge correctly received frames and to discard incorrect ones

Cyclic Redundancy Check

- A *k*-bits frame is regarded as the coefficient list for a polynomial with *k* terms, ranging from x^{k-1} to x^0 (called degree k-1)
- Based on standard polynomials:
  - Ex: 10111 is degree 4:

$$x^4+x^2+x^1+1$$

  - Computed with simple shift/XOR circuits

Cyclic Redundancy Check

- Idea: append CRC to the frame, so that transmitted frame viewed as a polynomial is evenly divisible by a generator polynomial $G(x)$
  - Both the high- and low-order bits of the generator must be 1
- Sender and receiver agree on the generator polynomial in advance

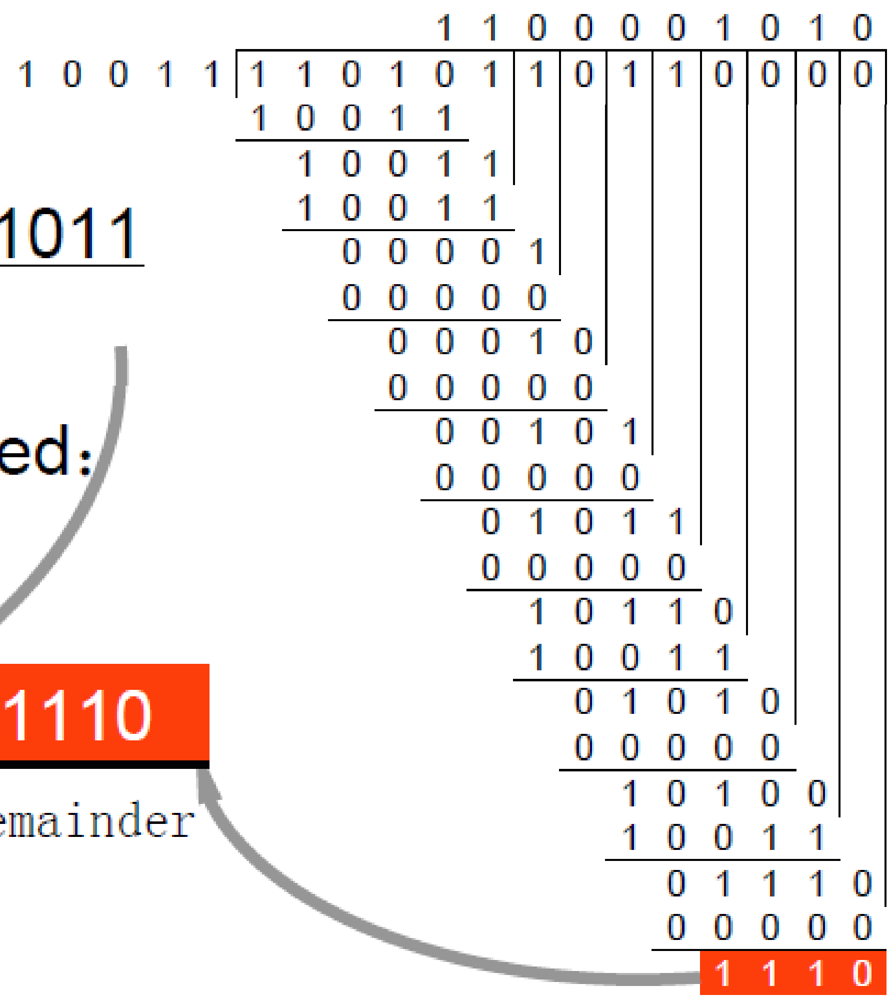
Algorithm for computing the CRC

- Suppose: $G(x)$ with degree r; A frame $M(x)$ with m bits
- Append r zero bits to the low-order end of the frame: $x^r M(x)$
- $R(x)$: remainder of $x^r M(x)/G(x)$
- $T(x)$: the resulted frame to be transmitted is $x^r M(x) - R(x)$
- Receiver: $T(x)/G(x) = 0$

Use modulo 2 division and substraction

Cyclic Redundancy Check

- Frame: <u>1101011011</u>
- $G(x) = x^4 + x + 1$
- Frame Transmitted:



Cyclic Redundancy Check

- Stronger detection
  - All single bit errors, if *G(X)* has >1 terms
  - All double-bit errors, if *G(x)* has a factor with at least 3 terms
  - Any odd number of errors, if *G(x)* has factor *(x+1)*
  - Any burst error of length < length of check bits

Standards Generator Polynomials

- Three polynomials are in common use they are:

CRC-16 = $x^{16}+x^{15}+x^{2}+1$ (used in HDLC)

CRC-CCITT = $x^{16}+x^{12}+x^{5}+1$

CRC-32 = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$ (used in Ethernet)

Outline

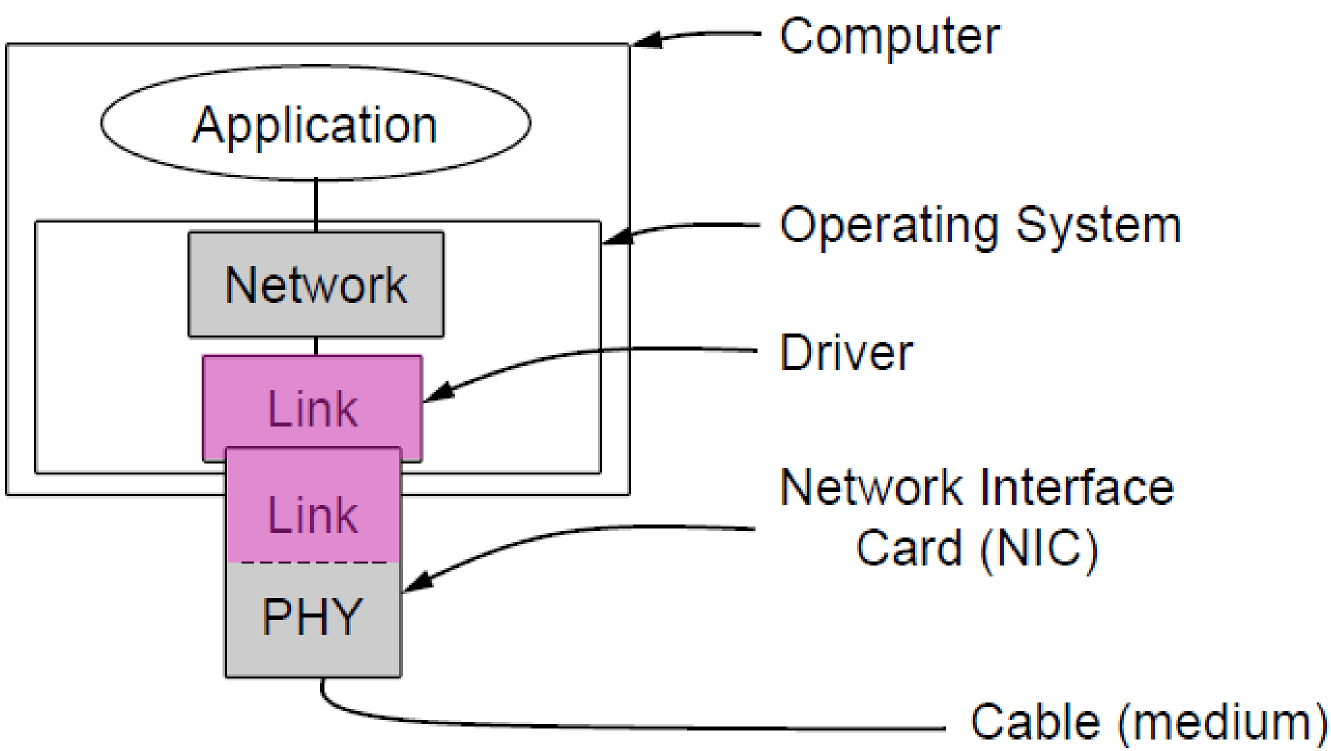
- Data Link Layer Design Issues
- Framing
- Error Control
  - Error Detection and Correction
- Flow Control
  - Sliding Window Protocols

Elementary Data Link Protocols 基本数据链路层协议

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

Link Layer Environment

- Link layer is commonly implemented as NICs and OS drivers
- Network layer (IP) is often OS software



Key Assumptions

- Physical Layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth
- Machine A wants to send a long stream of data to machine B, using a reliable, connection-oriented service.
- A is assumed to have an infinite supply of data ready to send and never has to wait for data to be produce
- Machines do not crash

Link Layer Environment

- Link layer protocol implementations use library functions
  - See code (*protocol.h*) for more details

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

Link Layer Environment

- 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

Utopian Simplex Protocol

- An optimistic protocol to get us started
  - Assumes no errors, and receiver is as fast as sender,
  - i.e., No Flow Control or Error Correction
  - Considers one-way data transfer (simplex)
  - The essence of this protocol
    - Sender loops sending frames, and receiver loops receiving frames

Utopian Simplex Protocol

```
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, so it is unrealistic

Next: No longer assume receiver can process incoming data infinitely fast, i.e., adding Flow Control.

Stop-and-Wait – Error-free channel

- Stop-and-Wait Protocol (停止等待协议) ensures sender can’t outpace receiver
  - Receiver returns a dummy frame (ACK) when ready
  - Only one frame out at a time
  - The essence of this protocol
    - Sender waits for ACK after passing frame to physical layer, and receiver sends ACK after passing frame to network layer

Stop-and-Wait – Error-free channel

```
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);
  }
}
```

```
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);
  }
}
```

ACK packet

Sender waits to for ACK after passing frame to physical layer

Receiver sends ACK after passing frame to network layer

We have **added flow control**!
But there is still **no error control**.

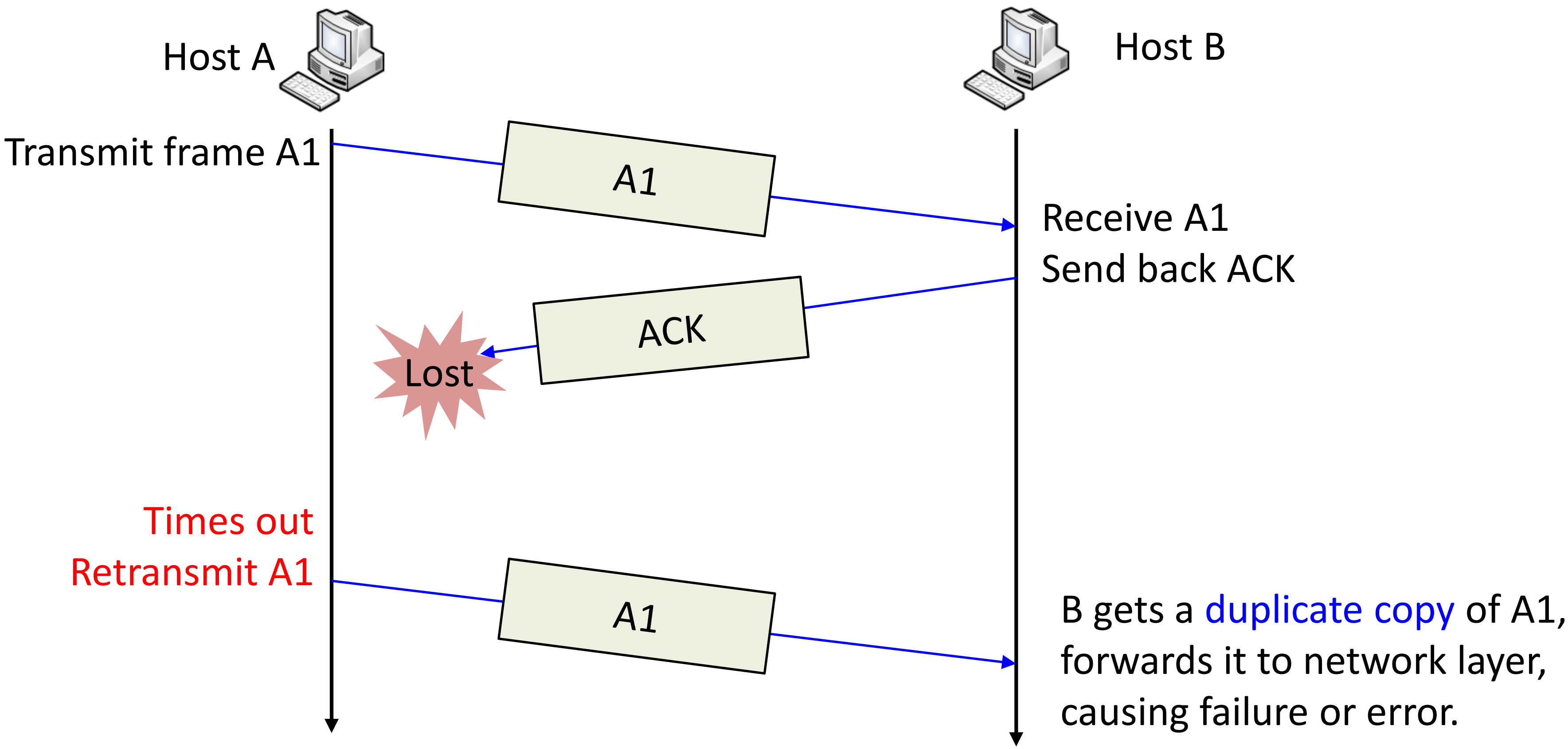
The channel maybe noisy and we may lose frames (they never arrive).

Simple approach:

add a timer to the sender so if no ACK after a certain period, it retransmits the frame.

Particular Scenario

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



Stop-and-Wait – Noisy channel

- ARQ (Automatic Repeat reQuest) * adds error control
  - Receiver ACKs frames that are correctly delivered
  - Sender sets timer and resends frame if no ACK
- For correctness, frames and ACKs must be numbered (called sequence number for each frame)
  - Otherwise, receiver can't tell retransmission (due to lost ACK or early timer) from new frame
  - For stop-and-wait, 2 numbers (1 bit) are sufficient

* Another name is PAR (Positive Acknowledgement with Retransmission).

Stop-and-Wait – Noisy channel

- Sender loop of ARQ

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

If a good ACK then set up for the next frame to send (else the old frame will be retransmitted)

```
void sender3(void) {
  seq_nr next_frame_to_send;
  frame s;
  packet buffer;
  event_type event;

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

Stop-and-Wait – Noisy channel

- Receiver loop of ARQ

Wait for a frame

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

Ack current frame

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

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);
       if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                 to_network_layer(&r.info);
                 inc(frame_expected);
            }
            s.ack = 1 – frame_expected;
            to_physical_layer(&s);
       }
  }
}
```

See Animation (PolyU) →

Stop-and-wait protocols only send out one unACKed frame on one direction each time.

Inefficient!!!

Allow multiple frames on both direction to improve efficiency:

- Bidirectional Transmission
- Multiple Frames in Flight

Solution → Sliding Window Protocols (滑动窗口协议)

Sliding Window Protocols

Assumptions:

- Provide Two-way communication
- If we use a separate link for data in both directions
  - For each link, the reverse channel (for ack) has the same capacity as the forward channel, and is almost entirely wasted
- Interleave two kinds of frames on the same link
  - Data frame
  - Control frame:
    - ACK: containing sequence number to ack the last correctly received frame

Typically, the sequence number of next frame to be sent

Piggybacking (捎带确认)

- Piggybacking: The technique of temporarily delaying outgoing ACKs so that they can be hooked onto the next outgoing data frame
- The principal advantage :
  - a better use of the available channel bandwidth
- Piggybacking issue:
  - For better use of bandwidth, how long should we wait for outgoing data frame before sending the ACK on its own. (set a timer)

Sliding Window Concept

Sending window

- 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

Receiving window

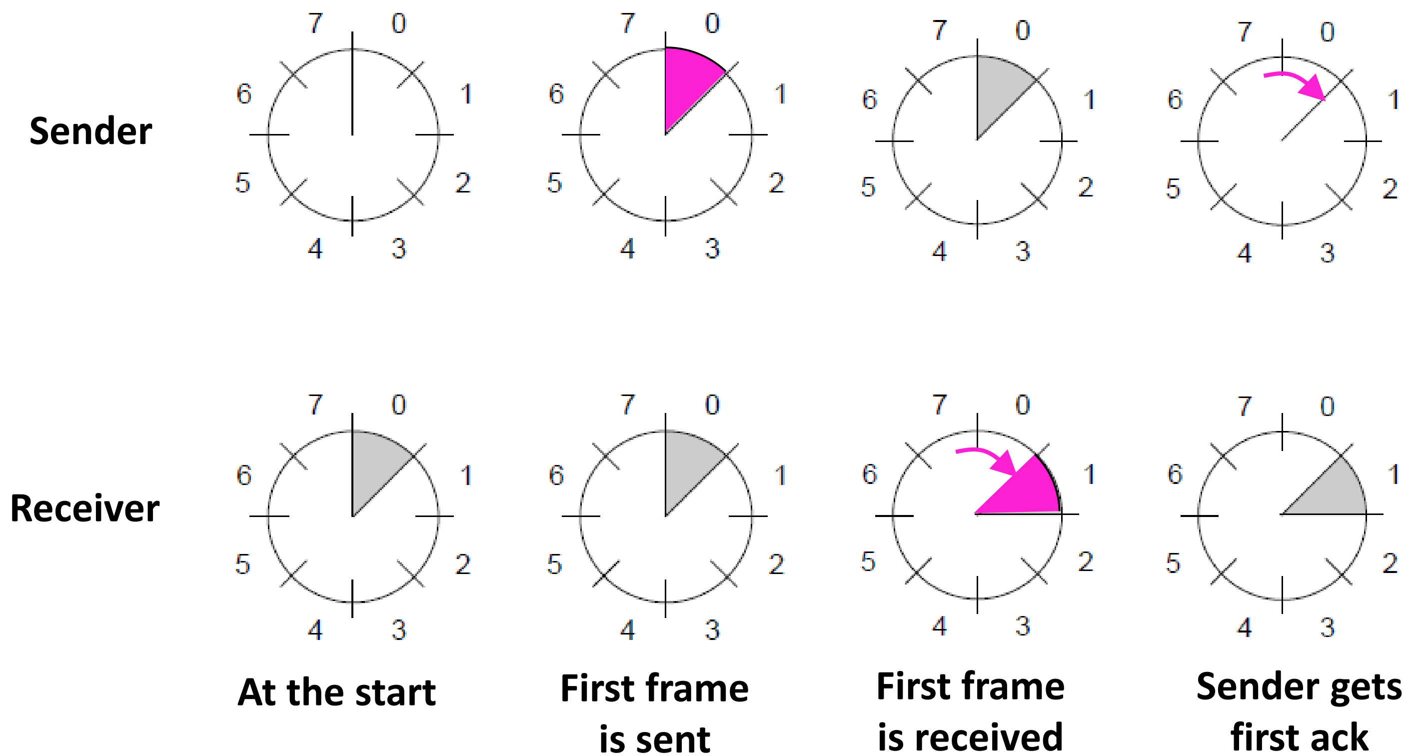
  - Needs to keep buffer space for arrivals
  - Window advances with in-order arrivals

Sending/Receiving Window

- The two windows need not have the same lower and upper limits or even have the same size
- And need not have the fixed size, they can grow or shrink over the time as frames are sent and received

Sliding Window Concept

- A sliding window advancing at the sender and receiver
  - Ex: window size is 1, with a 3-bit sequence number



Sliding Window Protocols

- 3 bidirectional protocols (full-duplex)
  - One-Bit Sliding Window Protocol
  - Protocol Using Go-Back-N
  - Protocol Using Selective Repeat
- Differ in terms of
  - Efficiency, complexity, and buffer requirements

One-Bit Sliding Window

- Transfers data in both directions with Stop-and-Wait
  - Window size: 1, i.e., one frame each time
  - Piggybacks ACKs on reverse data frames for efficiency
  - Handles transmission errors, flow control, early timers
- Each node is sender and receiver

```
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);
```

Prepare first frame

Launch it, and set timer

. . .

One-Bit Sliding Window

. . .

Wait for frame or timeout

If a frame with new data then deliver it

If it is an ACK for last sent data, prepare for next data frame

(Otherwise it was a timeout)

Send next data frame or retransmit old one; ACK the last data we received

```
while (true) {
    wait_for_event(&event);
    if (event == frame_arrival) {
        from_physical_layer(&r);
        if (r.seq == frame_expected) {
            to_network_layer(&r.info);
            inc(frame_expected);
        }
        if (r.ack == next_frame_to_send) {
            stop_timer(r.ack);
            from_network_layer(&buffer);
            inc(next_frame_to_send);
        }
    }
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 – frame_expected;
    to_physical_layer(&s);
    start_timer(s.seq);
  }
}
```

ack== seq. of last received frame

Allow multiple frames on flight to increase utilization?

Pipelining Strategies

- Stop-and-wait (w=1) is inefficient for long links
- Larger windows enable pipelining (管道化) for efficient link use:
  - Pipelining: keeping multiple frames in flight
  - 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.

Link utilization

- Need to determine channel capacity, or how many frames can fit inside the channel.
- **Bandwidth-delay product**: bandwidth (bits/sec) multiplied by one way transit time

BDP = Bandwidth * Delay => bit = bit/sec * sec

- Best window (w) depends on bandwidth-delay (BD), i.e., number of frames

$$\text{link utilization} \leq \frac{w}{1 + 2BD}$$

2BDBDBDack

BDP / frame_size = BD

BD is number of frames

BD is the number of frames transmitted in unit time

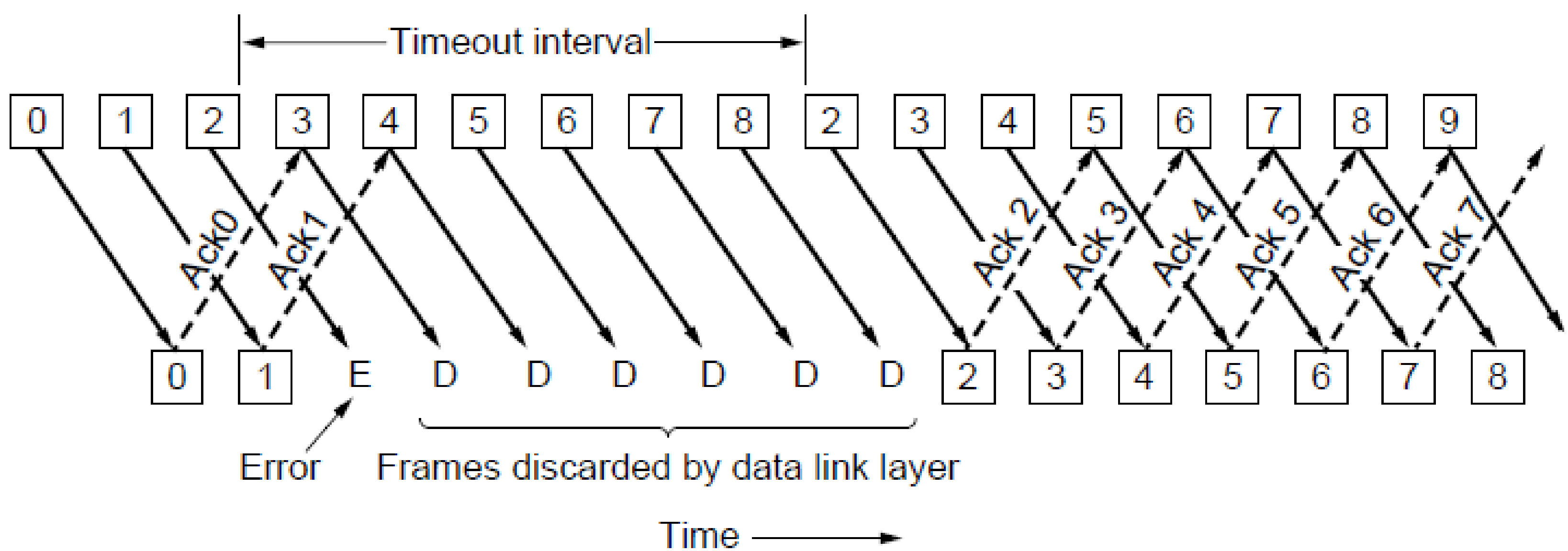
w = 1 for Stop-and-Wait protocol

Problem and Solution

- What if 20 frames are transmitted, and the 2nd has an error?
  - Frames 3-20 will be ignored at receiver side? Sender will have to retransmit.
- Pipelining leads to different choices for errors/buffering
  - Go-Back-N
  - Selective Repeat

Go-Back-N (回退N协议)

- 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

- Tradeoff made for Go-Back-N
  - Simple strategy for receiver: needs only 1 frame, i.e., receiving window size is 1
  - Wastes link bandwidth for errors with large windows: entire window may be retransmitted
- Implementation (see full code in book)

Sliding Window Protocol Using Go-Back-N

```
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);
}
```

Sequence No. **ACK No.**

```
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

```
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 */
          }
```

accept one frame each time

Continued →

Sliding Window Protocol Using Go Back N

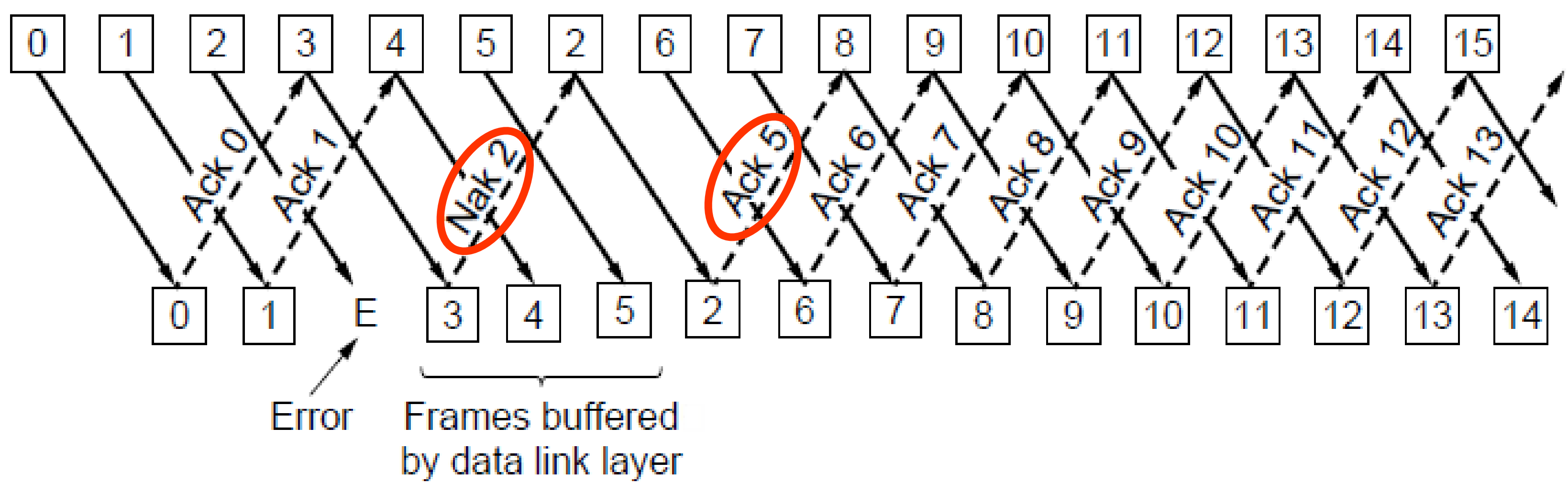
Cumulative ACK

```
            /* 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();
   }
}
```

[See Animation]() **(Prof. Pappu) →**

Selective Repeat (选择重传协议)

- Receiver accepts frames anywhere in receive window
  - **Cumulative ACK (累计确认)** indicates highest in-order frame
  - **NAK** (negative ACK) causes sender retransmission of a missing frame before a timeout resends



Selective Repeat

- Tradeoff made for Selective Repeat
  - More complex than Go-Back-N due to buffering at receiver and multiple timers at sender
  - More efficient use of link bandwidth as only lost frames are resent (with low error rates)
- Implementation (see full code in book)

A Sliding Window Protocol Using Selective Repeat

```
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)) || ((c < a) && (a <= b)) || ((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 */
}
```

Continued →

A Sliding Window Protocol Using Selective Repeat

```
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 */
                    }
               }
          }
```

NAK

Continued →

A Sliding Window Protocol Using Selective Repeat

Cumulative ACK

```
            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();
  }
}
```

[See Animation](#) (PolyU)→

ACK timer

- ACK timer:
  - Slightly larger than the normal time interval expected between sending a frame and receiving its ACK

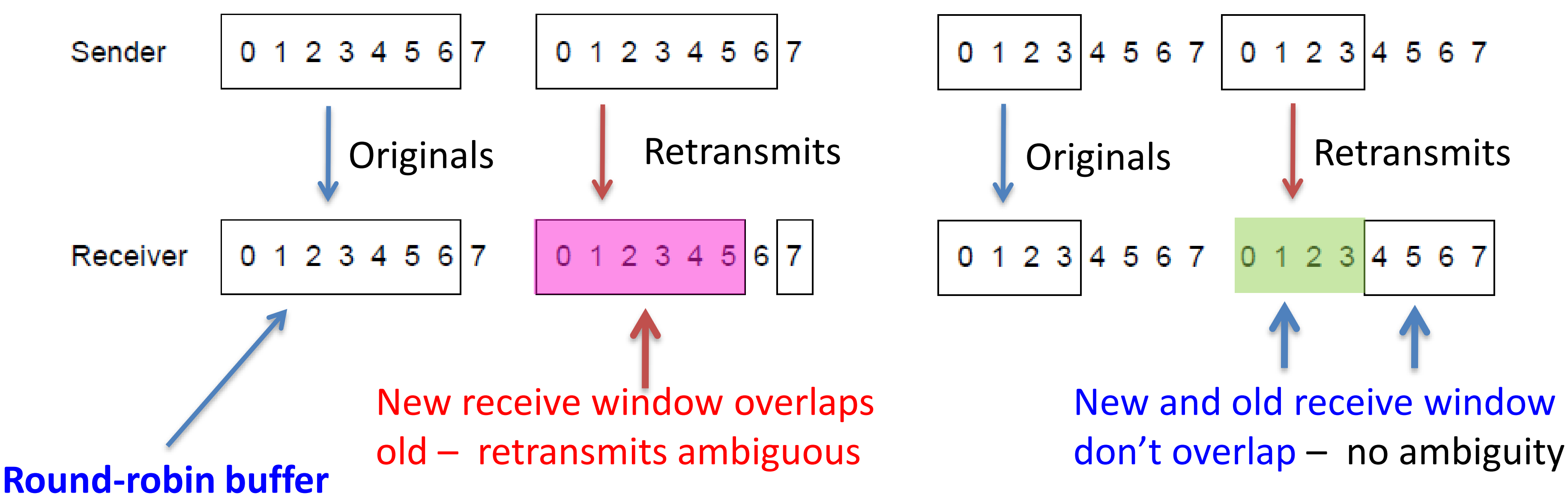
Nonsequential receiving problems(非顺序接收)

- Example:
  - $m = 3$ bits sequence no. (0~7), $w_S = w_R = 7$
- For correctness, we require
  - Sequence numbers (s) at least the total of both windows

$w_S+w_R<=2^m$

Error case (s=8, w=7)
too few sequence numbers

Correct (s=8, w=4)
enough sequence numbers



Windows size

Protocol	W_S: sending window	W_R: receiving window
Stop-and-Wait	1	1
Go-Back-N	>1($<= 2^m-1$)	1
Selective Repeat	>1	>1($<= 2^{m-1}$)

Sequence No. is *m bits*

$w_S >= w_R$, and $w_S + w_R <= 2^m$

Review

- Converting raw bit stream into frame stream:
  - byte count, byte stuffing, bit stuffing, etc.
- Error detection and correction
  - Hamming distance, parity, checksum, CRC
- Stop-and-Wait ARQ
- Sliding window
  - Go-Back-N, Selective Repeat

Thank You!

Q & A