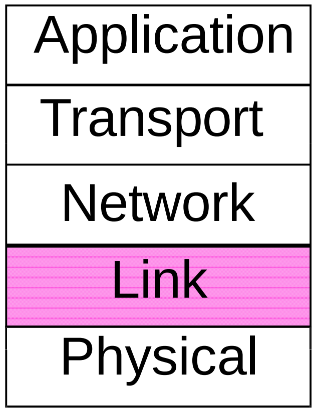
# The Datalink Layer
# Chapter 3

- How to transmit information among two machines between two adjacent machines in a reliable and efficient manner?
  - Conceptually two machines are connected like a wire (coaxial cable, telephone line, or wireless channel)
  - There is a non-zero propagation delay
  - Communication channel makes errors

## The Data Link Layer

Responsible for delivering **frames** of information over a single link

- Handles transmission errors and regulates the flow of data
- Provides well-defined service to network layer
- Dealing with transmission errors
- Regulates data between high-speed sender and slow speed receiver



## Outline

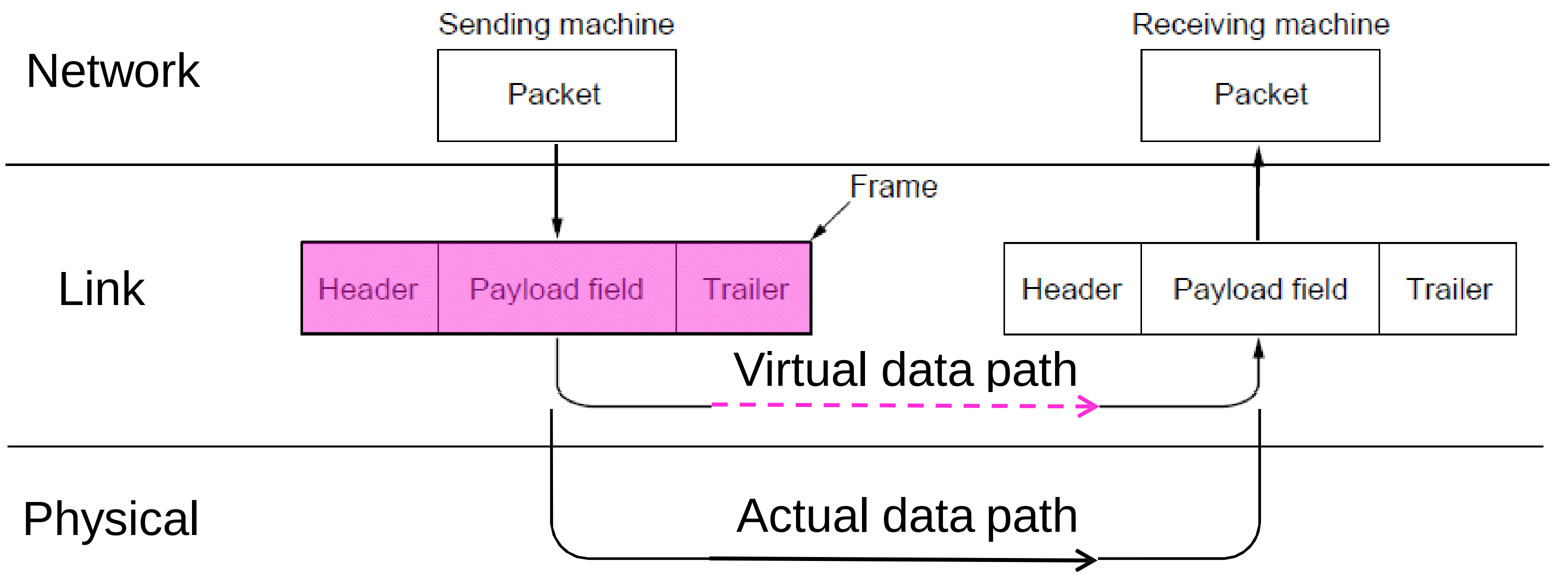
- Data Link Layer Design Issues
- Error Detection and Correction
- Elementary Data Link Protocols
- Sliding Window Protocols
- Example Data Link Protocols

## Data Link Layer Design Issues

- Frames »
- Possible services »
- Framing methods »
- Error control »
- Flow control »

## Frames

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



## Possible Services

Unacknowledged connectionless service

- Frame is sent with no connection / error recovery
- Ethernet is an example

Acknowledged connectionless service

- Frame is sent with retransmissions if needed
- Example is 802.11 (WiFi)

Acknowledged connection-oriented service

- First phase: Connection is set-up
- Second phase: Frames are transmitted
- Third phase: connection is released.

# Framing Methods

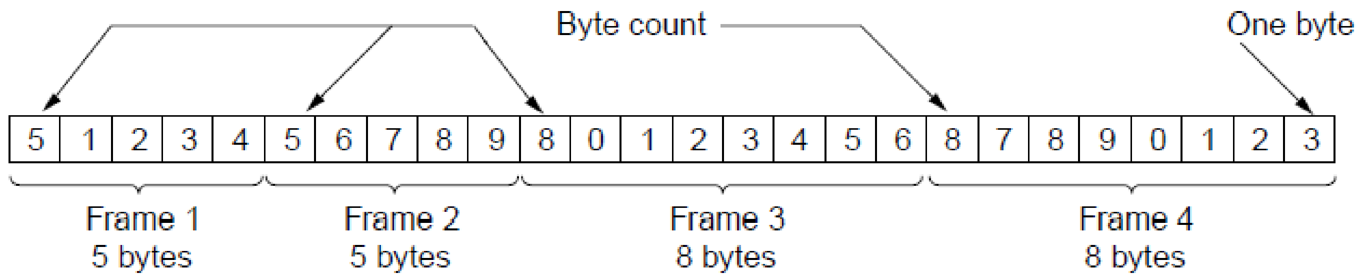
- Datalink layer must use services provided by physical layer
  - The physical layer may add some redundant signals, but the transmission may not be  not error free
  - It is up to datalink layer to detect, if necessary, correct errors.
- Approach
  - Break the bitstream into discrete frames, compute the checksum and include checksum with the frame. When the frame arrives, checksum is recomputed. If it is different, steps will be taken to deal with the issue.
- Four methods of framing
  - Byte count »
  - Flag bytes with byte stuffing »
  - Flag bits with bit stuffing »
  - Physical layer coding violations
    - Use non-data symbol to indicate frame

# Framing – Byte count

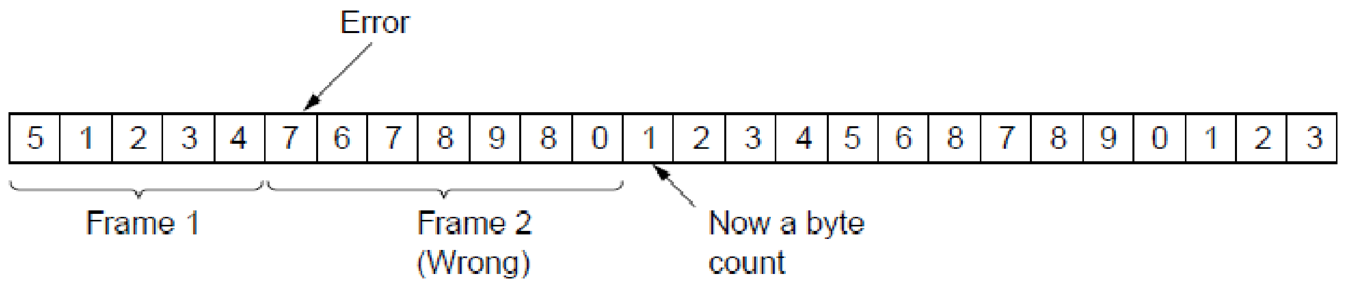
Frame begins with a count of the number of bytes in it

- Simple, but difficult to resynchronize after an error

Expected case



Error case

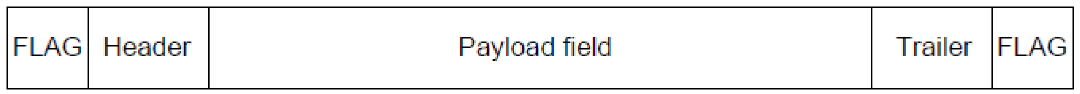


# Framing – Byte stuffing

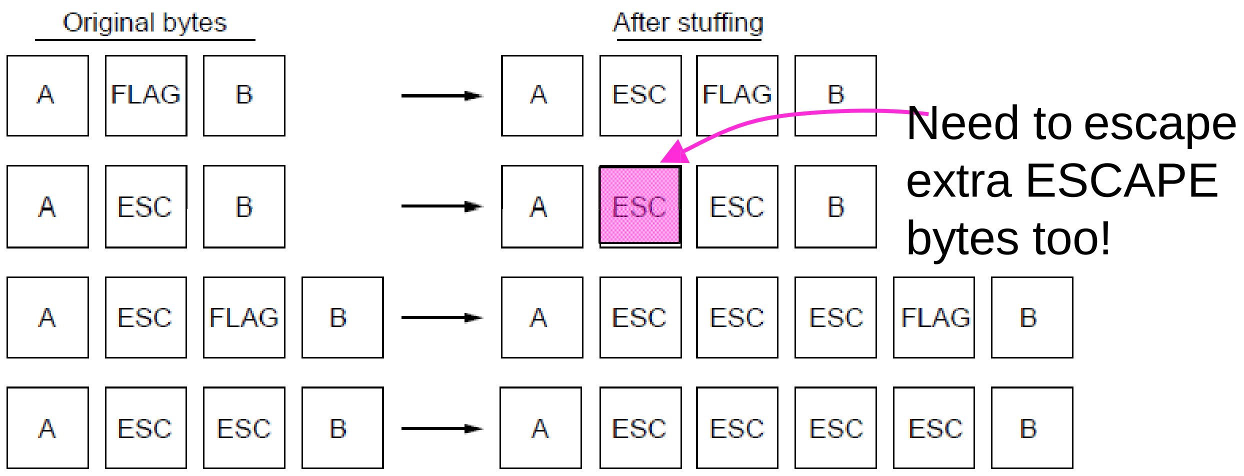
Special flag bytes delimit frames; occurrences of flags in the data must be stuffed (escaped), at the both end of frame

- Longer, but easy to resynchronize after error

Frame format



Stuffing examples



# Framing – Bit stuffing

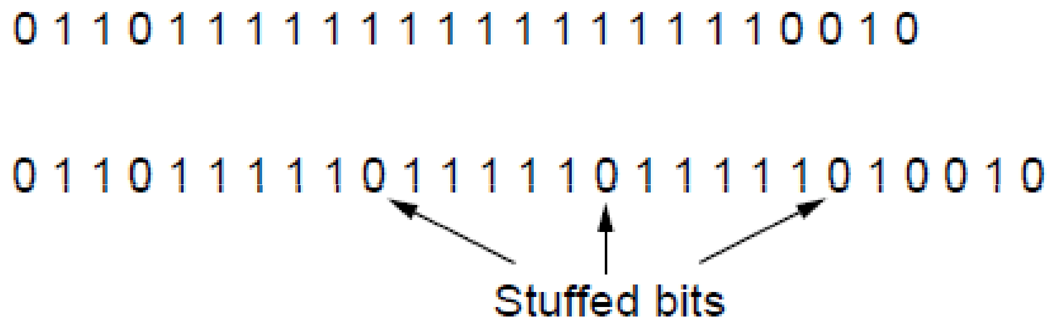
Stuffing done at the bit level:

- Frame flag has six consecutive 1s (not shown)
- On transmit, after five 1s in the data, a 0 is added
- On receive, a 0 after five 1s is deleted

Data bits

Transmitted bits

with stuffing



## About Error Control

- Issue: How to make sure all frames are eventually delivered to the network layer in order?
- For connectionless service, it is OK if the sender keeps sending the frames continuously
- However, for reliable, connection-oriented service, it would not be fine at all.
- Approach
  - Error control repairs frames that are received in error
  - Requires errors to be detected at the receiver
  - Feedback in terms of positive and negative acks.
  - Timer protects against lost acknowledgements
  - Typically retransmit the unacknowledged frames
  - When frames are transmitted multiple times, there is an issue
    - Assign sequence numbers

## Design Issue: Flow Control

How to prevent a fast sender from out-pacing a slow receiver?

- Feedback based flow control
  - Receiver sends back information to the sender giving it the permission to send the data
- Rate-based flow control
  - Built-in mechanism that limits the rate.
- Overall, a set of well-defined rules are formed
- Receiver gives feedback on the data it can accept
- Rare in the Link layer as NICs run at “wire speed”
  - Receiver can take data as fast as it can be sent

Flow control is a topic in the Link and Transport layers.

## Error Detection and Correction

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 »

## Error Bounds – Hamming distance

Code turns data of n bits into codewords of n+k bits

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

- Example with 4 codewords of 10 bits (n=2, k=8):
  - 0000000000, 0000011111, 1111100000, and 1111111111
  - Hamming distance is 5

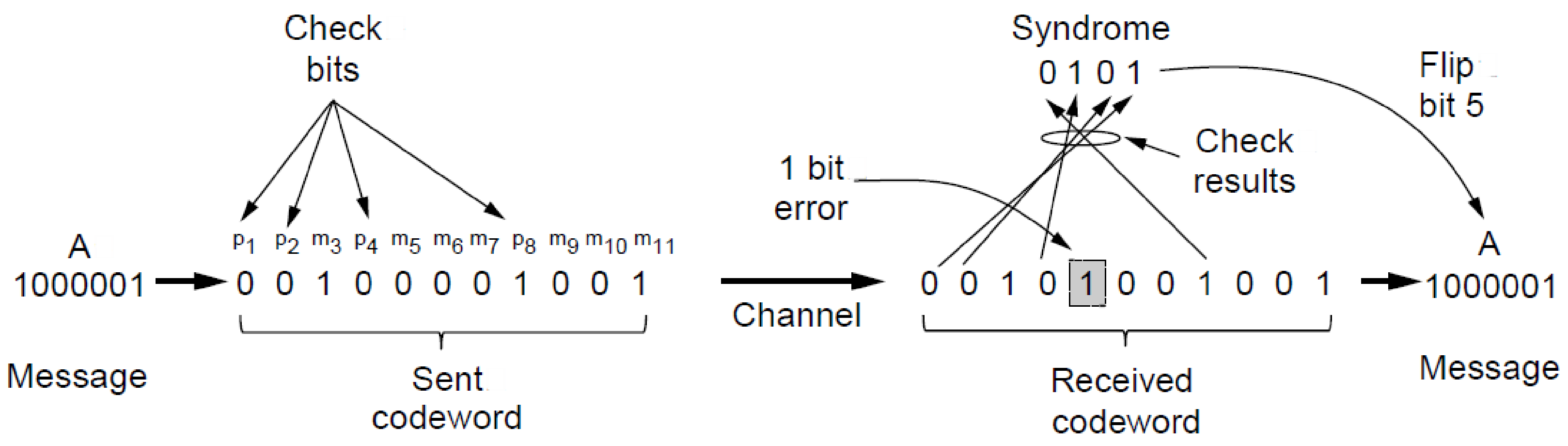
Bounds for a code with distance:

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

# 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

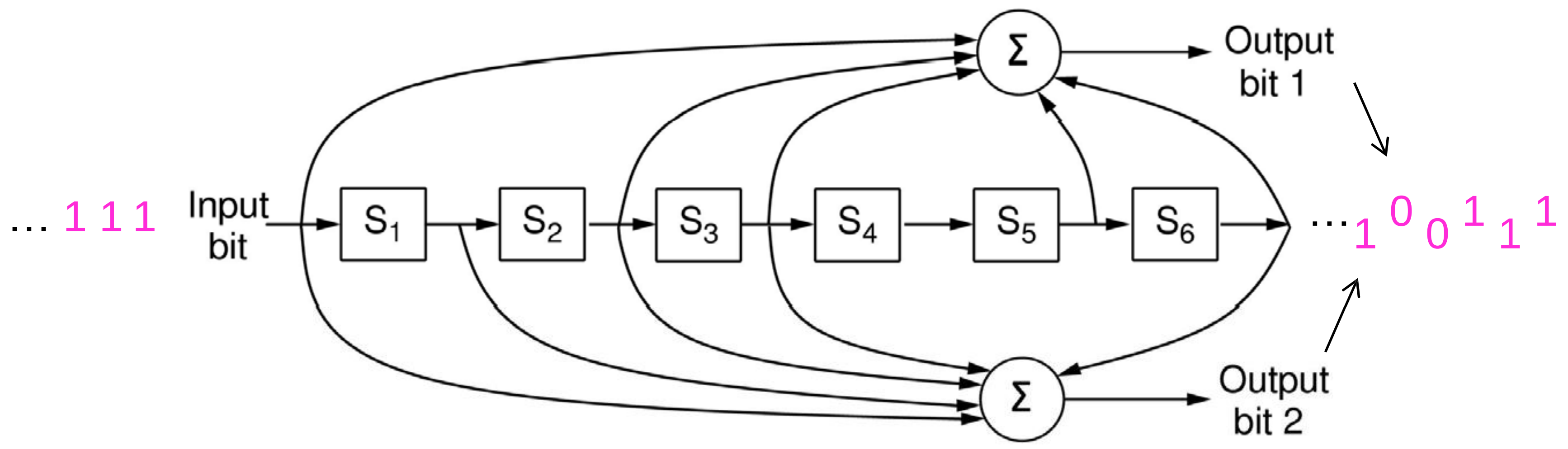


(11, 7) Hamming code adds 4 check bits and can correct 1 error

## Error Correction – Convolutional codes

Operates on a stream of bits, keeping internal state

- Output stream is a function of all preceding input bits
- Bits are decoded with the Viterbi algorithm



Popular NASA binary convolutional code (rate = ½) used in 802.11

## Error Detection – Parity (1)

Parity 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)

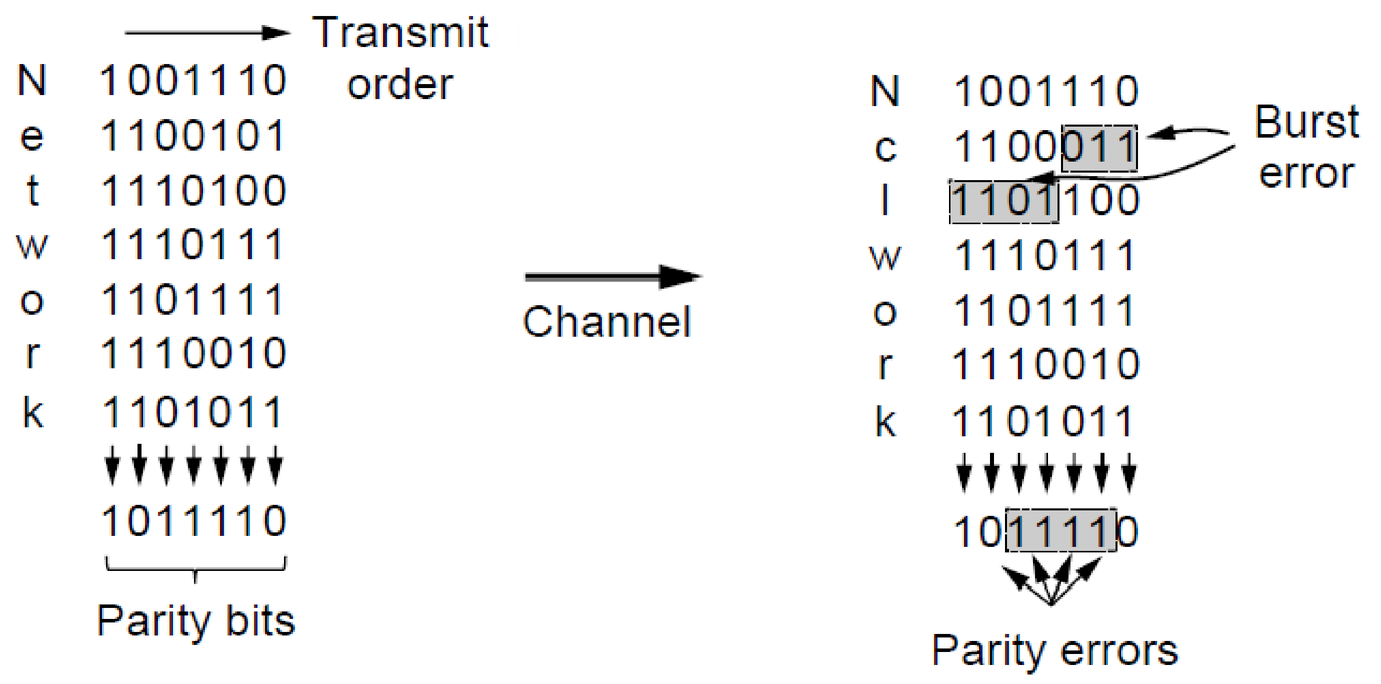
Simple way to detect an *odd* number of errors

- Ex: 1 error, 11100$\underline{1}$01; detected, sum is wrong
- Ex: 3 errors, 11$\underline{011}$001; detected sum is wrong
- Ex: 2 errors, 1110$\underline{11}$01; *not detected*, sum is right!
- Error can also be in the parity bit itself
- Random errors are detected with probability ½

# Error Detection – Parity (2)

Interleaving 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 treats data as N-bit words and adds N check bits that are the modulo $2^N$ sum of the words

- Ex: Internet 16-bit 1s complement checksum

Properties:

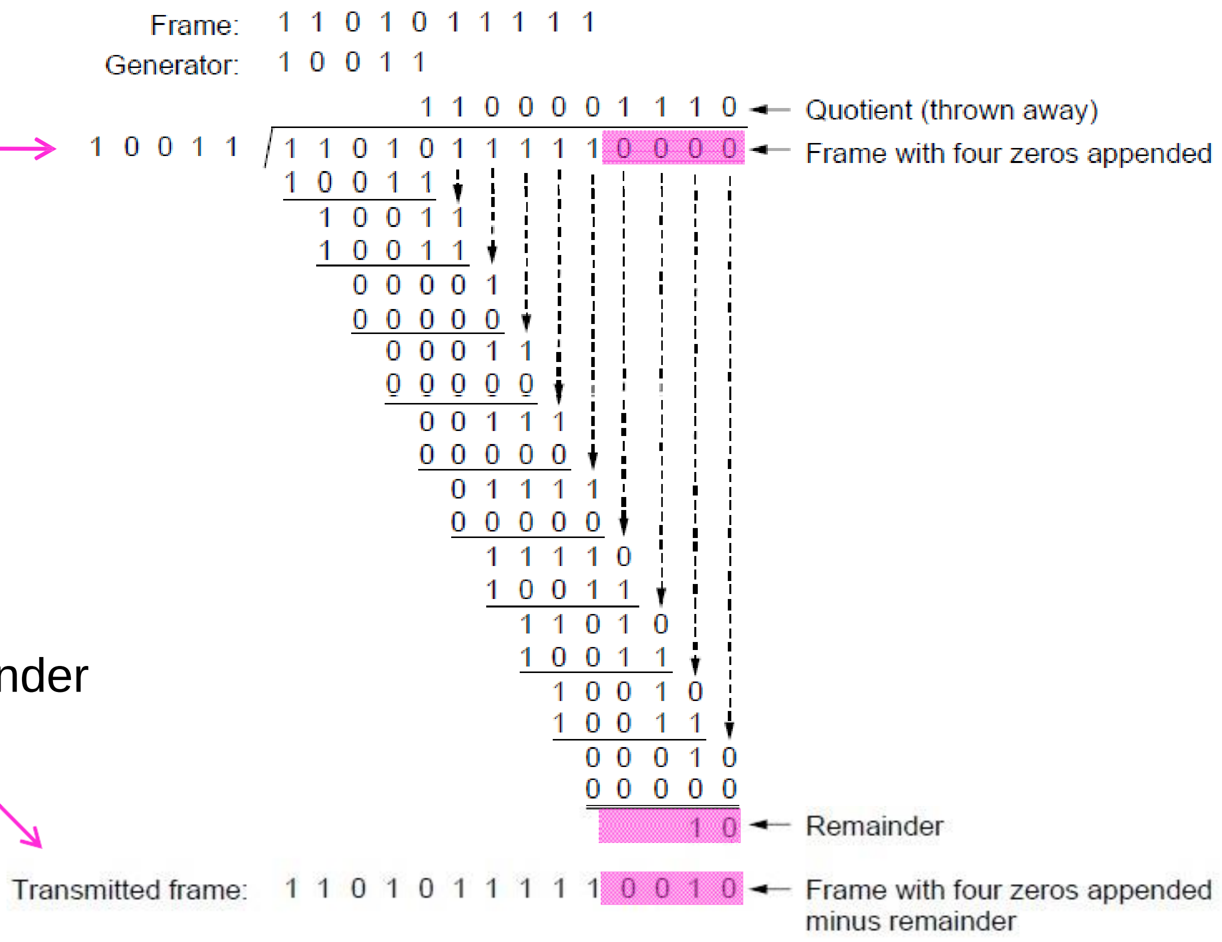
- Improved error detection over parity bits
- Detects bursts up to N errors
- Detects random errors with probability $1\text{-}2^N$
- Vulnerable to systematic errors, e.g., added zeros

## Error Detection – CRCs (1) Cyclic Redundancy Checks

Adds bits so that transmitted frame viewed as a polynomial is evenly divisible by a generator polynomial

Start by adding 0s to frame and try dividing

Offset by any reminder to make it evenly divisible



Frame: 1 1 0 1 0 1 1 1 1 1
Generator: 1 0 0 1 1
1 1 0 0 0 0 1 1 1 0 ← Quotient (thrown away)
1 0 0 1 1 | 1 1 0 1 0 1 1 1 1 1 0 0 0 0 ← Frame with four zeros appended
1 0 0 1 1
1 0 0 1 1
1 0 0 1 1
0 0 0 0 1
0 0 0 0 0
0 0 0 1 1
0 0 0 0 0
0 0 1 1 1
0 0 0 0 0
0 1 1 1 1
0 0 0 0 0
1 1 1 1 0
1 0 0 1 1
1 1 0 1 0
1 0 0 1 1
1 0 0 1 0
1 0 0 1 1
0 0 0 1 0
0 0 0 0 0
1 0 ← Remainder
Transmitted frame: 1 1 0 1 0 1 1 1 1 1 0 0 1 0 ← Frame with four zeros appended minus remainder

# Error Detection – CRCs (2)

Based on standard polynomials:

- Ex: Ethernet 32-bit CRC is defined by:

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

- Computed with simple shift/XOR circuits

Stronger detection than checksums:

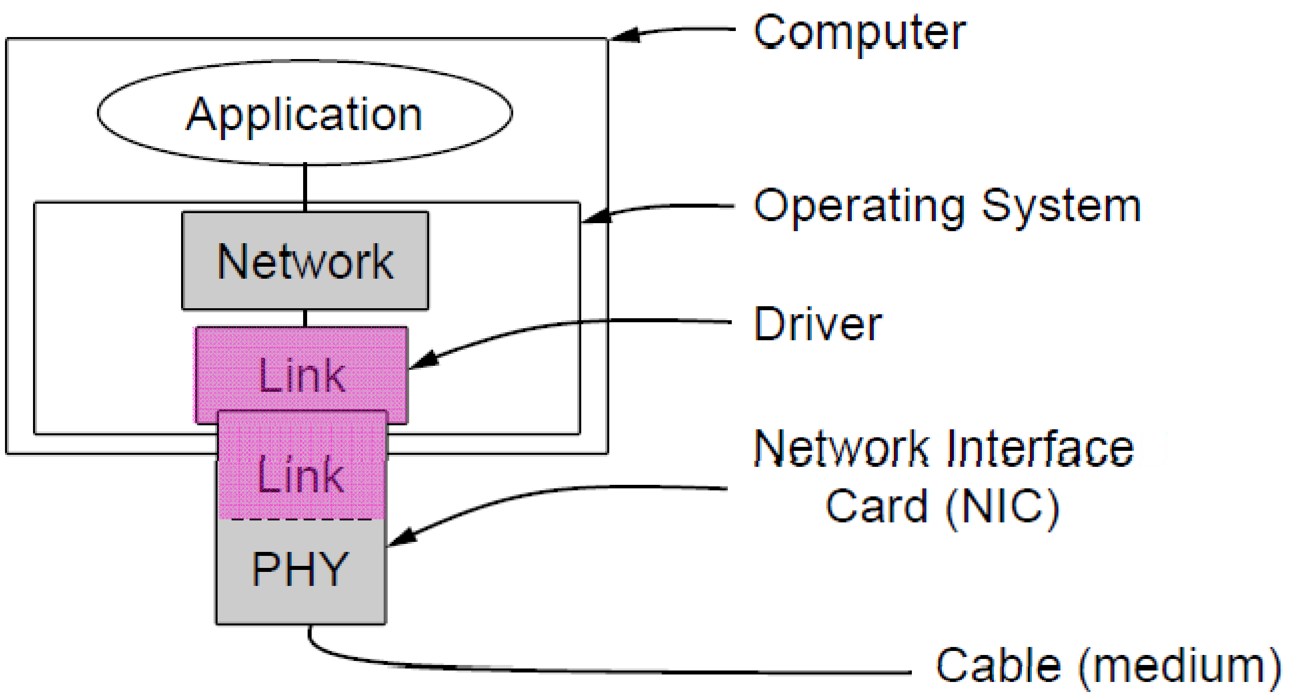
- E.g., can detect all double bit errors
- Not vulnerable to systematic errors

## Elementary Data Link Protocols

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

# Link layer environment (1)

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



# Link layer environment (2)

Link layer protocol implementations use library functions

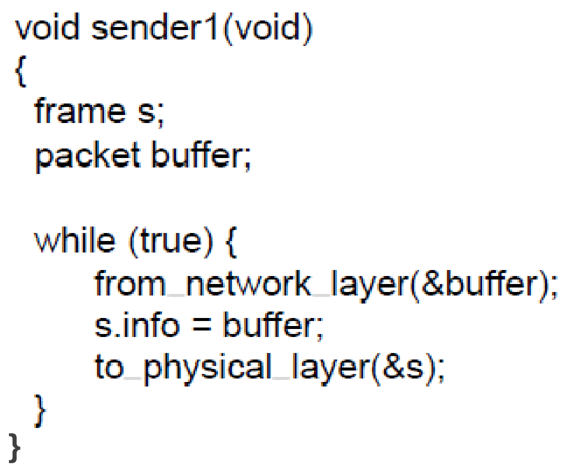
- See code (`protocol.h`) for more details

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

## Utopian Simplex Protocol

An optimistic protocol (p1) to get us started

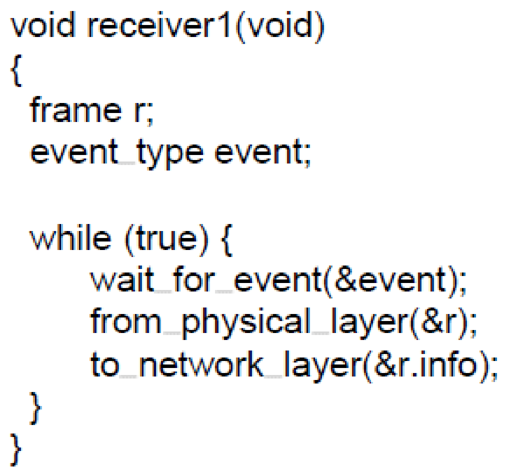
- Assumes no errors, and receiver as fast as sender
- Considers one-way data transfer



```
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 …

## 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 <u>stop-and-wait</u>
- 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

## Stop-and-Wait – Noisy channel (1)

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

- Else 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

# Stop-and-Wait – Noisy channel (2)

Sender loop (p3):

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 (3)

Receiver loop (p3):

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);
      }
  }
}
```

# Sliding Window Protocols

- Sliding Window concept »
- One-bit Sliding Window »
- Go-Back-N »
- Selective Repeat »

# Sliding Window Protocols

- In the previous protocols, data frames are transmitted in one direction only
- In most practical situations, data frames have to be transmitted in both directions.
- Each link has a forward channel and reverse channel. The capacity of reverse channel is wasted.
- Alternative
  - Use the same link in both directions.
  - Data frames of A and B are intermixed with ack frames of A and B.
- Piggybacking
  - Temporarily delaying ack so that they can be hooked into the next outgoing frame.
- However, piggybacking increases the delay.
  - Use hybrid
    - If possible use piggybacking otherwise send a separate ack frame.

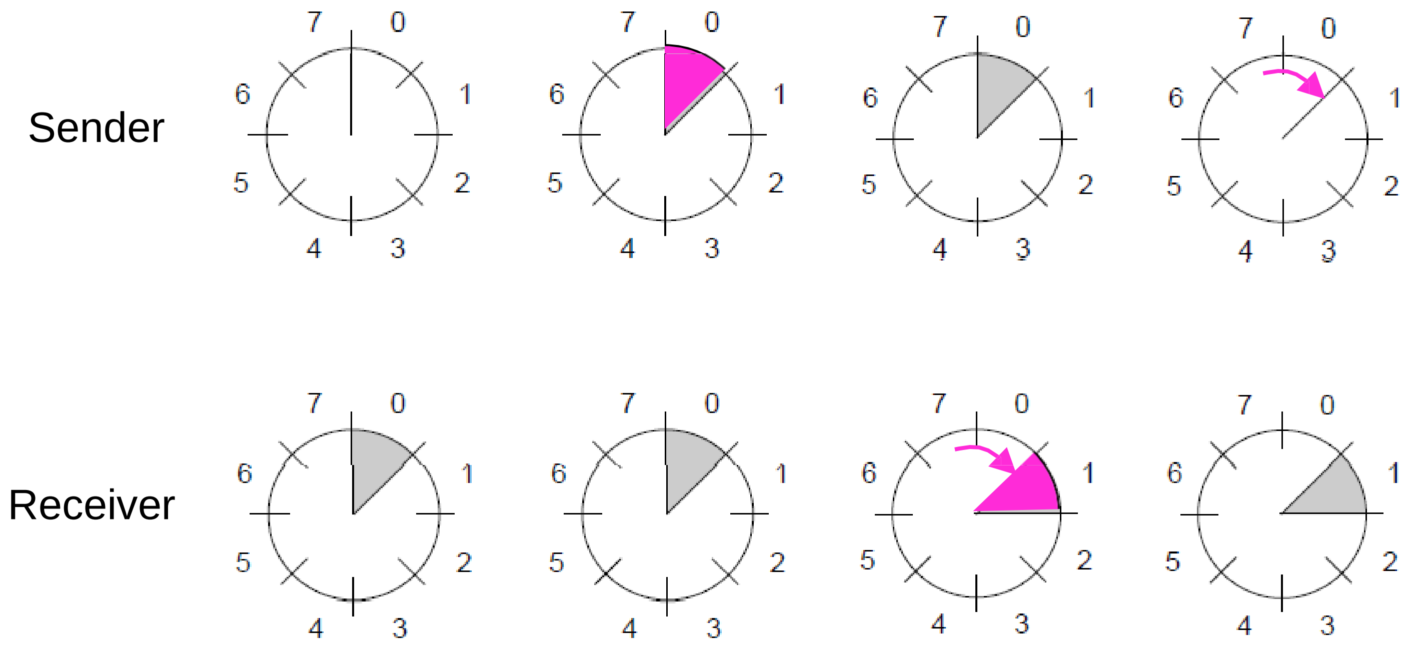
## Sliding Window Protocols

- Summary of sliding window protocols
  - Sender maintains a sequence numbers for frames (sending window)
  - Receiver also maintains a receiving window.
  - Whenever a packet is received from network layer, it gives a highest sequence number and the upper edge of the window is advanced by one.
  - When ack, arrives lower edge of the window is advanced by one.
  - Sender needs n buffers to hold unack frames

# Sliding Window concept (2)

A sliding window advancing at the sender and receiver

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



At the start

First frame is sent

First frame is received

Sender gets first ack

# Sliding Window concept (3)

Larger windows enable pipelining 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 Go-Back-N and Selective Repeat

## One-Bit Sliding Window (1)

Transfers data in both directions with stop-and-wait

- Piggybacks acks on reverse data frames for efficiency
- Handles transmission errors, flow control, early timers

Each node is sender and receiver (p4):

```
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 (2)

Wait for frame or timeout

If a frame with new data then deliver it

If an ack for last send then prepare for next data frame

(Otherwise it was a timeout)

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

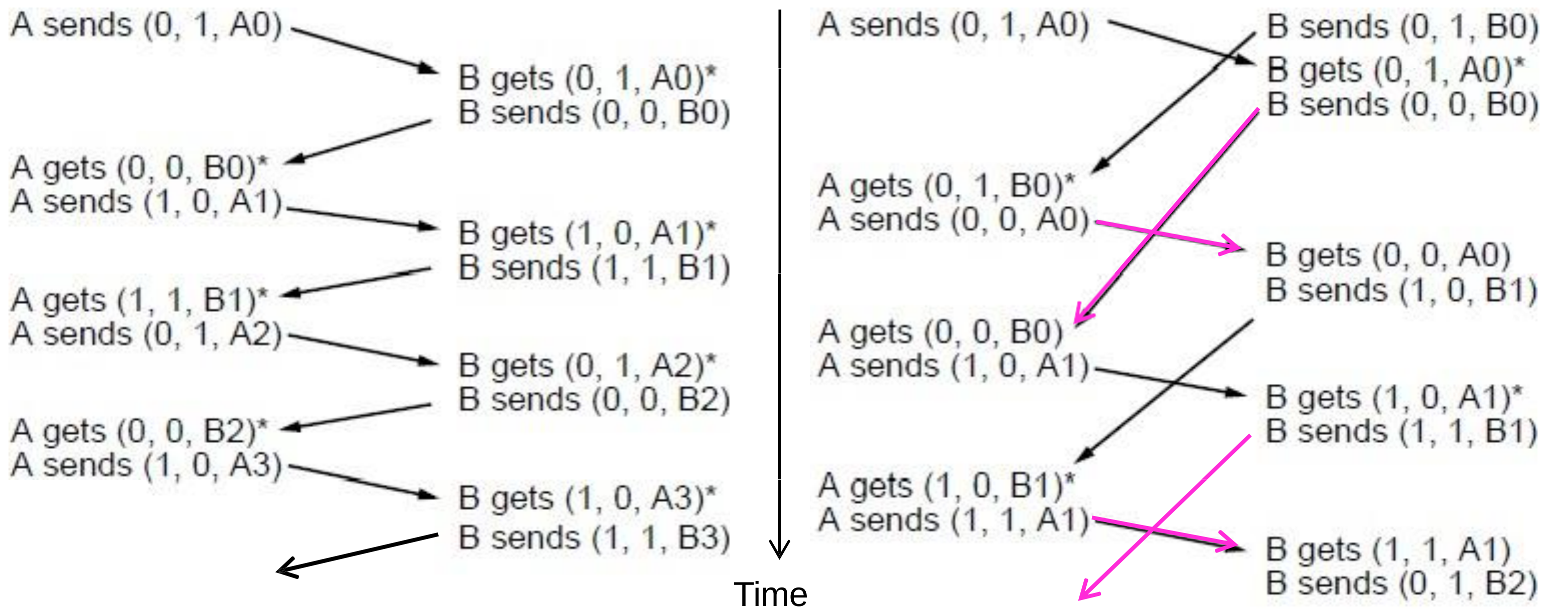
```
. . .

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);
  }
}
```

## One-Bit Sliding Window (3)

Two scenarios show subtle interactions exist in p4:

- Simultaneous start [right] causes correct but slow operation compared to normal [left] due to duplicate transmissions.



Notation is (seq, ack, frame number). Asterisk indicates frame accepted by network layer .

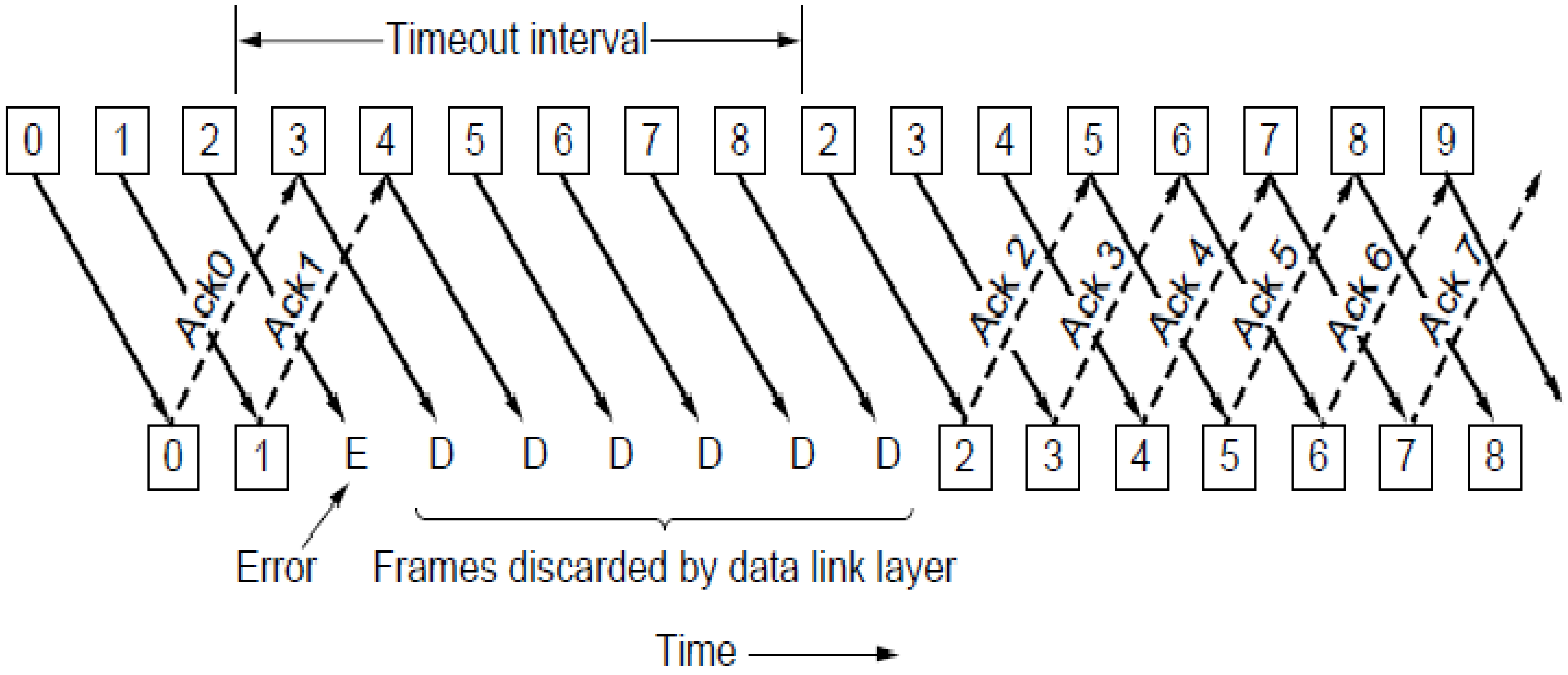
Normal case

Correct, but poor performance

# Go-Back-N Sliding window protocol

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 Protocol

Tradeoff made for Go-Back-N:

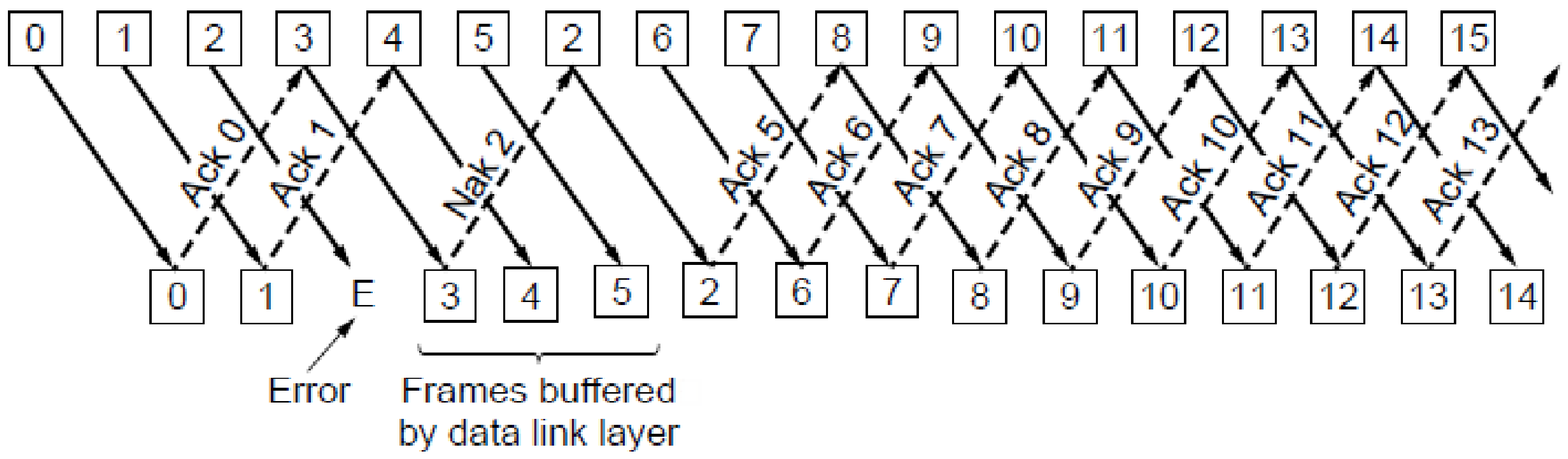
- Simple strategy for receiver; needs only 1 frame
- Wastes link bandwidth for errors with large windows; entire window is retransmitted

Implemented as p5 (see code in book)

# Selective Repeat Sliding Window Protocol

Receiver accepts frames anywhere in receive window

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



## About 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)

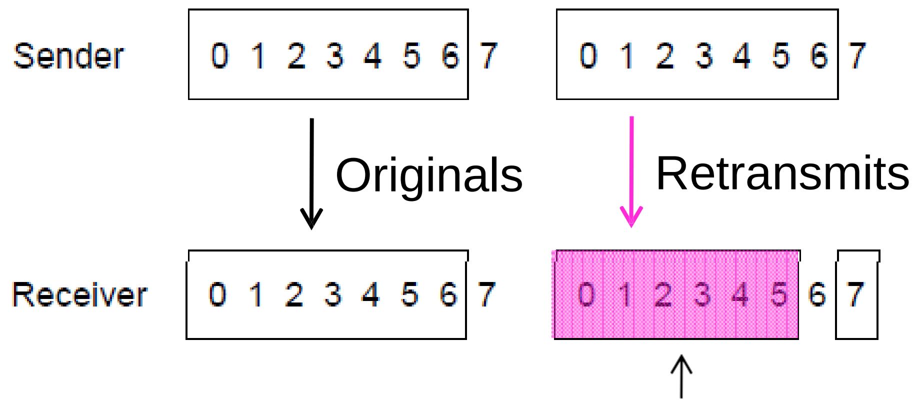
Implemented as p6 (see code in book)

# Selective Repeat

For correctness, we require:

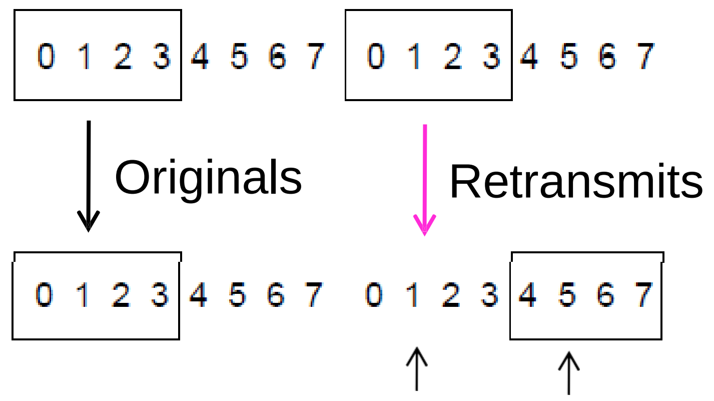
- Sequence numbers (s) at least twice the window (w)

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



New receive window overlaps old – retransmits ambiguous

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



New and old receive window don’t overlap – no ambiguity

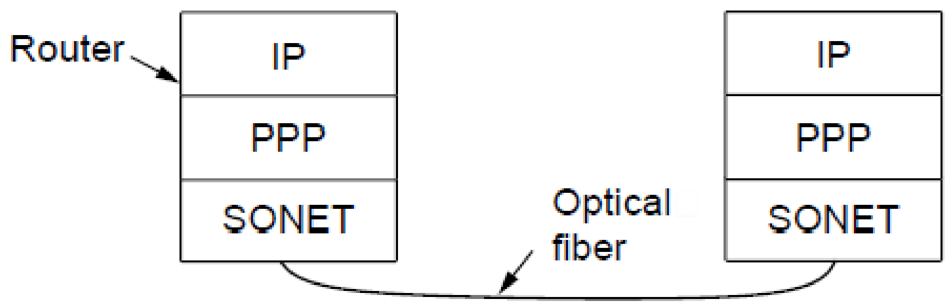
## Example Data Link Protocols

- Packet over SONET (Synchronous Optical Network (SONET)
- PPP (Point-to-Point Protocol) »
- ADSL (Asymmetric Digital Subscriber Loop) »

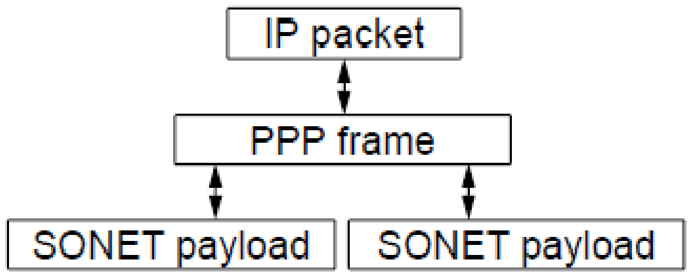
# Packet over SONET

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

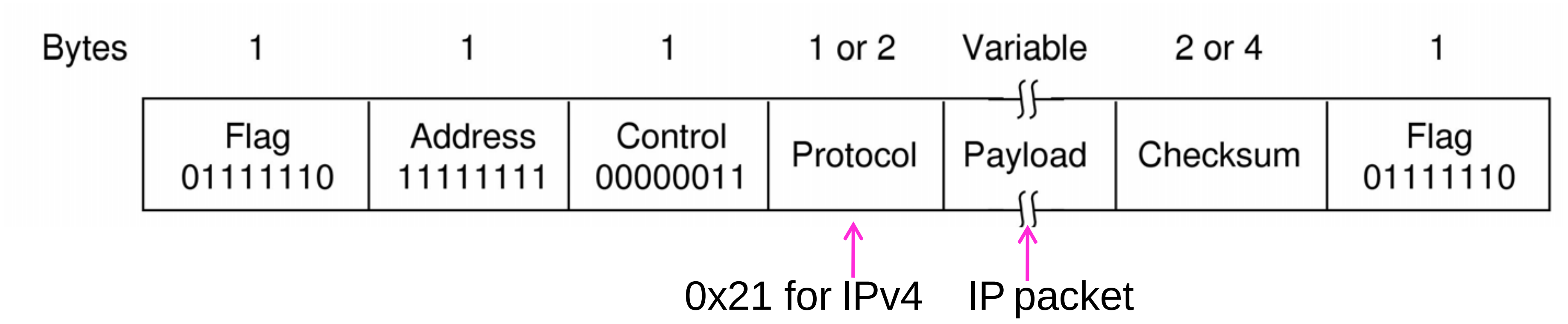


PPP frames may be split over SONET payloads

# PPP (1)

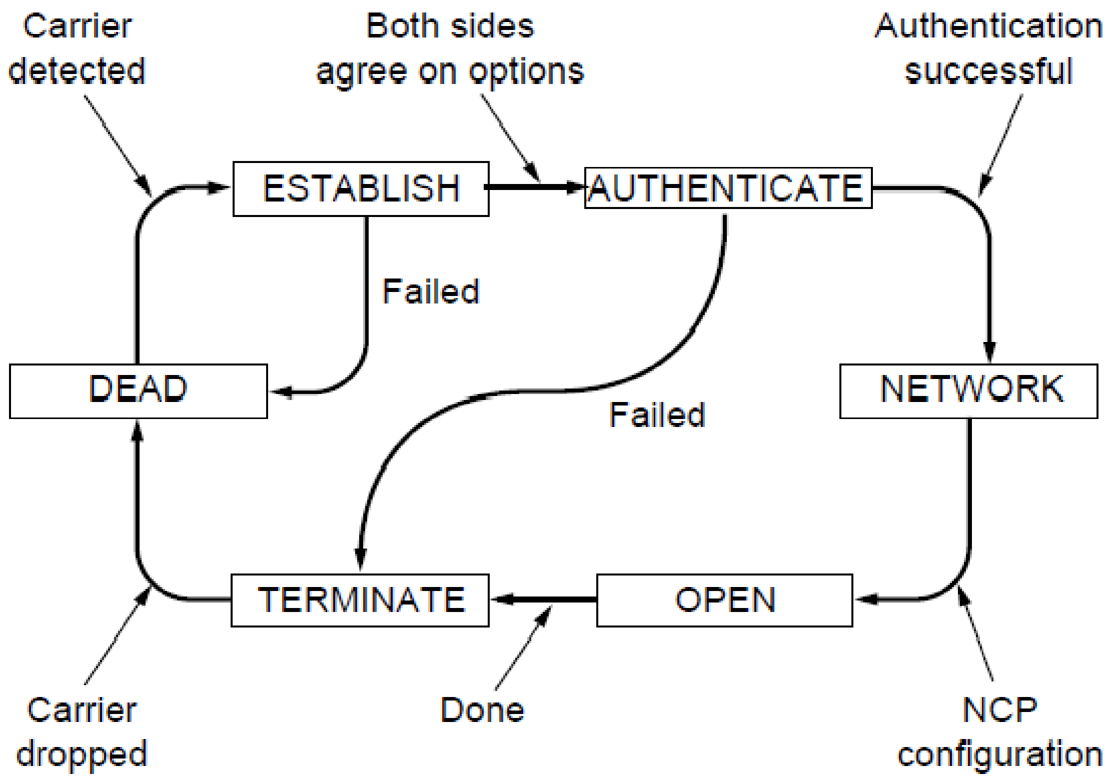
PPP (Point-to-Point Protocol) is a general method for delivering packets across links

- Framing uses a flag (0x7E) and byte stuffing
- "Unnumbered mode" (connectionless unacknow-ledged service) is used to carry IP packets
- Errors are detected with a checksum



# PPP (2)

A link control protocol brings the PPP link up/down

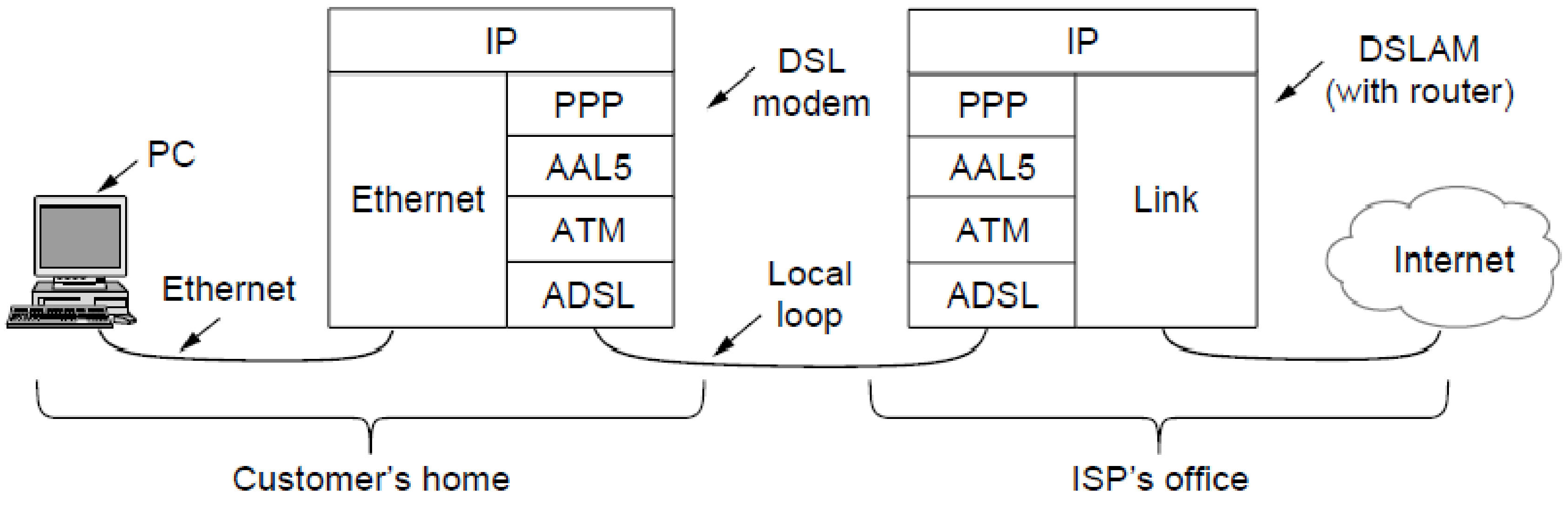


State machine for link control

# ADSL (1)

Widely used for broadband Internet over local loops

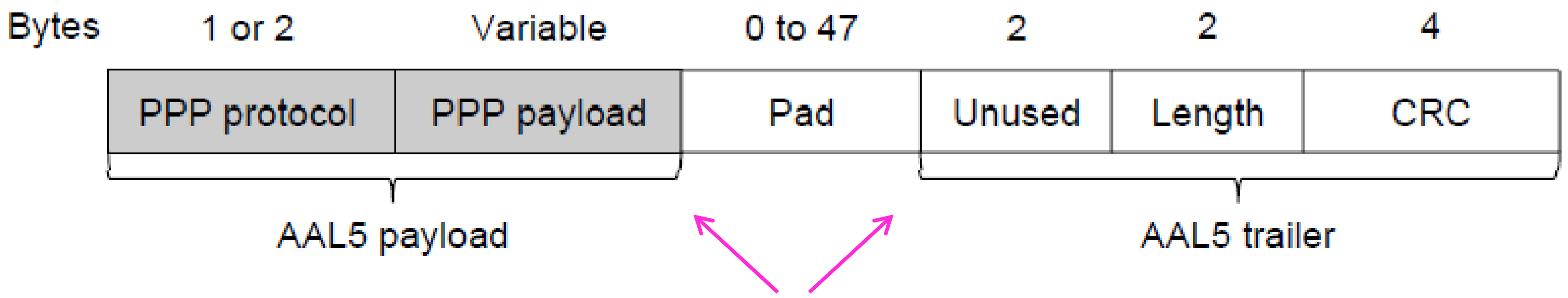
- ADSL runs from modem (customer) to DSLAM (ISP)
- IP packets are sent over PPP and AAL5/ATM (over)



## ADSL (2)

PPP data is sent in AAL5 frames over ATM cells:

- ATM is a link layer that uses short, fixed-size cells (53 bytes); each cell has a virtual circuit identifier
- AAL5 is a format to send packets over ATM
- PPP frame is converted to a AAL5 frame (PPPoA)



AAL5 frame is divided into 48 byte pieces, each of which goes into one ATM cell with 5 header bytes

End

Chapter 3