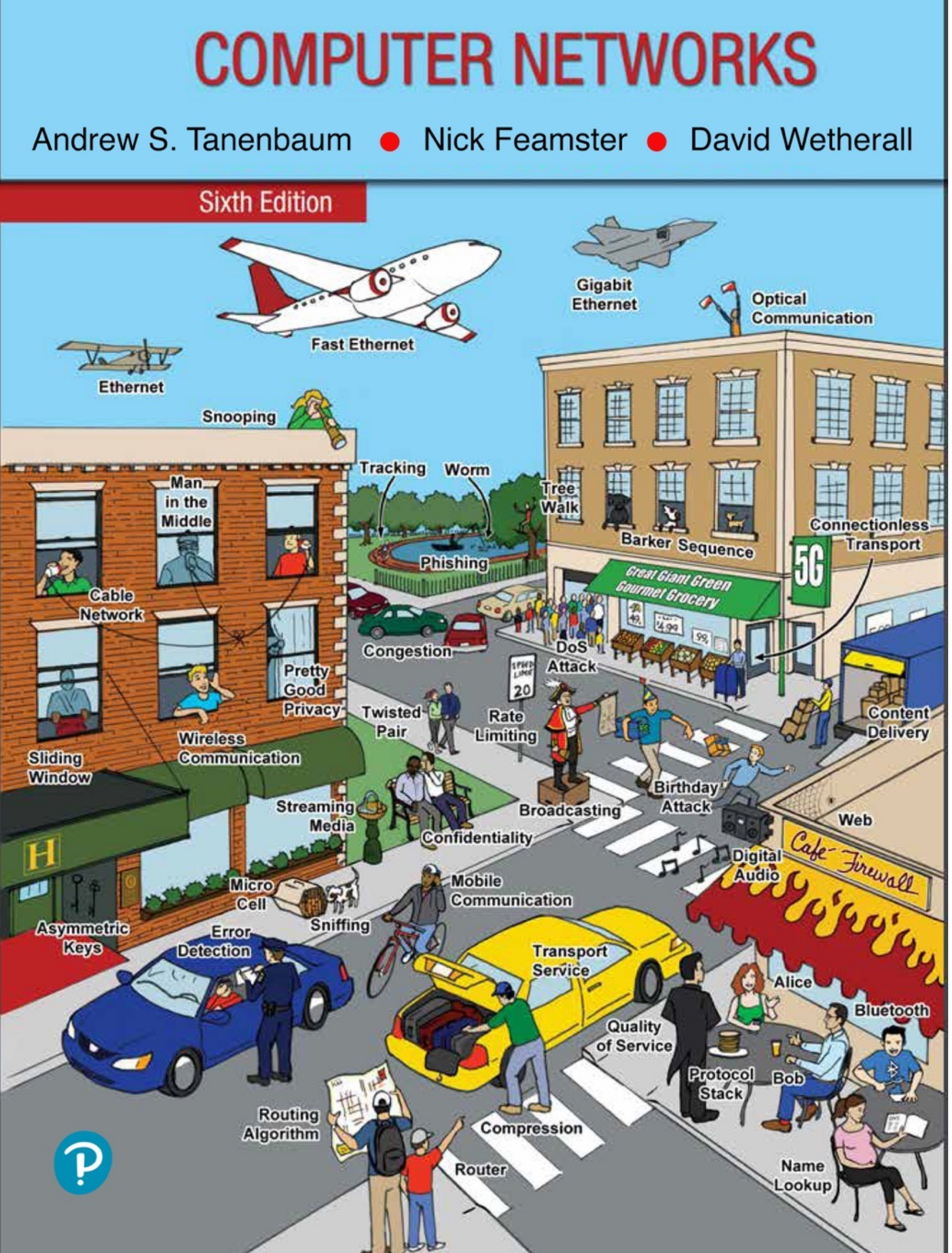


Chapter 3

The Data Link Layer

Questions for you

Why is Cellular Network called cellular?

Major differences among 1G, 2G, 3G, 4G, 5G

How satellite transmits data?

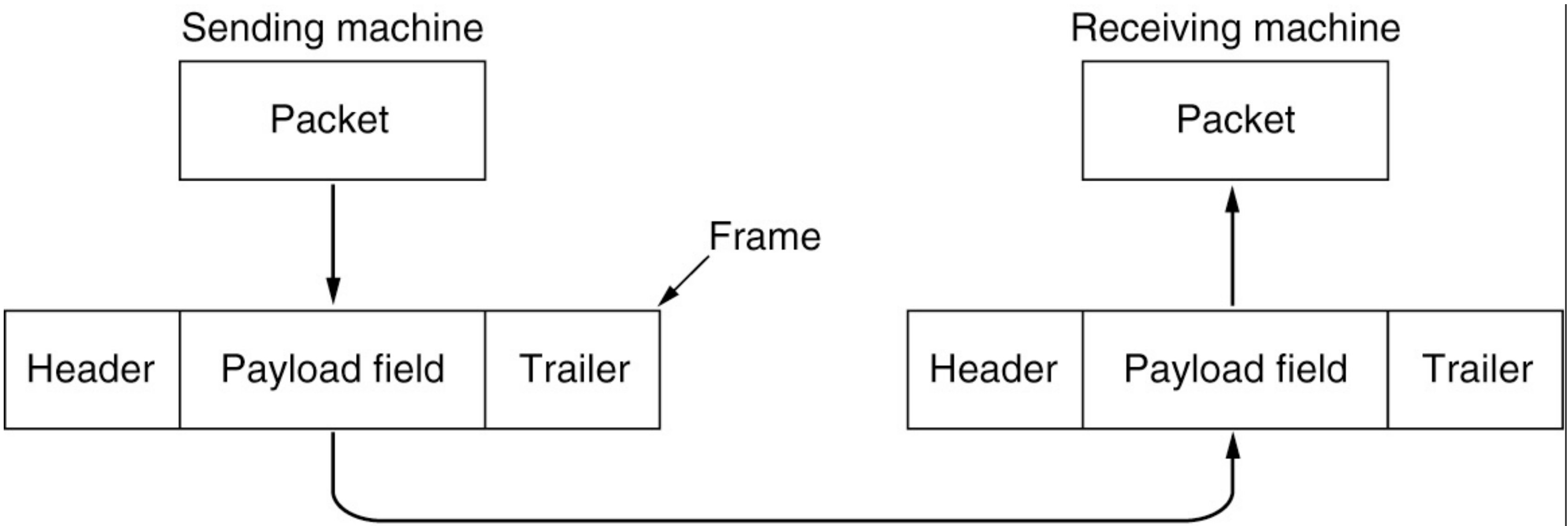
Data Link Layer Design Issues (1 of 2)

- Network layer services
- Framing
- Error control
- Flow control

Data Link Later Functions

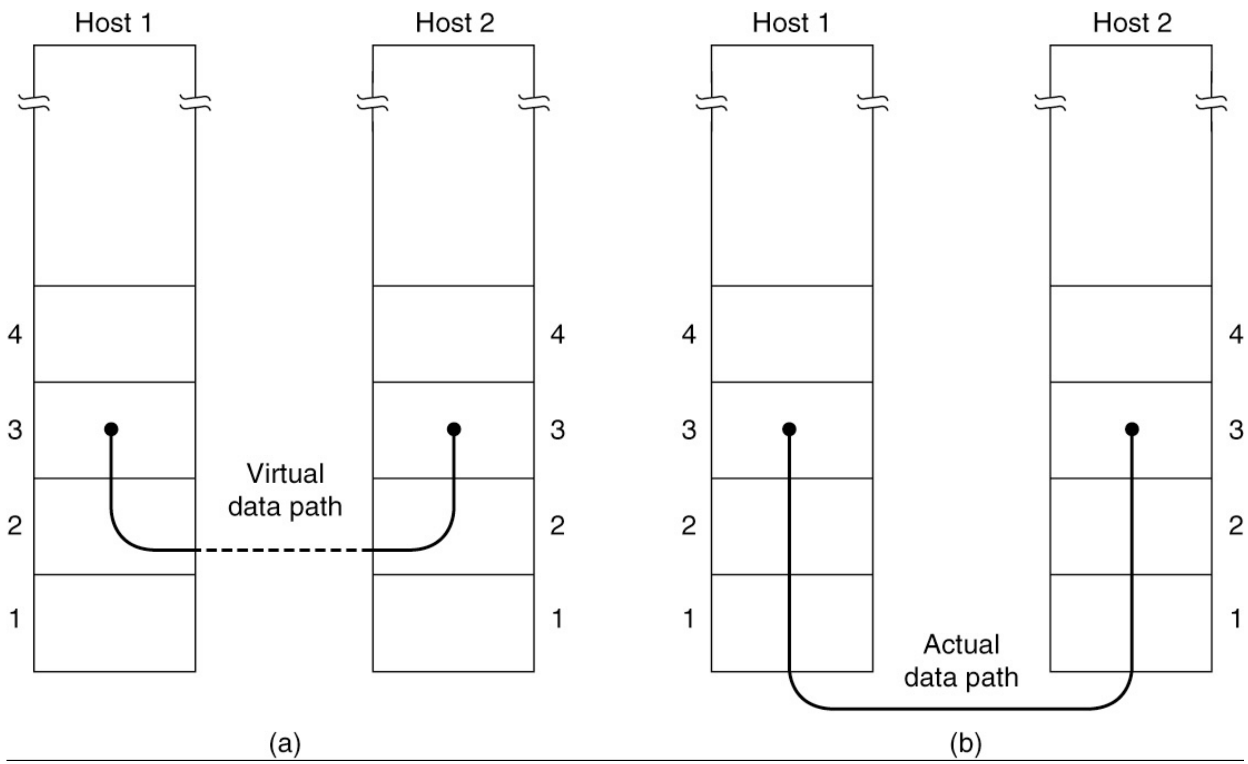
- Providing a well-defined service interface to the network layer.
- Framing sequences of bytes as self-contained segments.
- Detecting and correcting transmission errors.
- Regulating the flow of data so that slow receivers are not swamped by fast senders.

Data Link Layer Design Issues (2 of 2)



Relationship between packets and frames

Services Provided to The Network Layer (1 of 2)



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

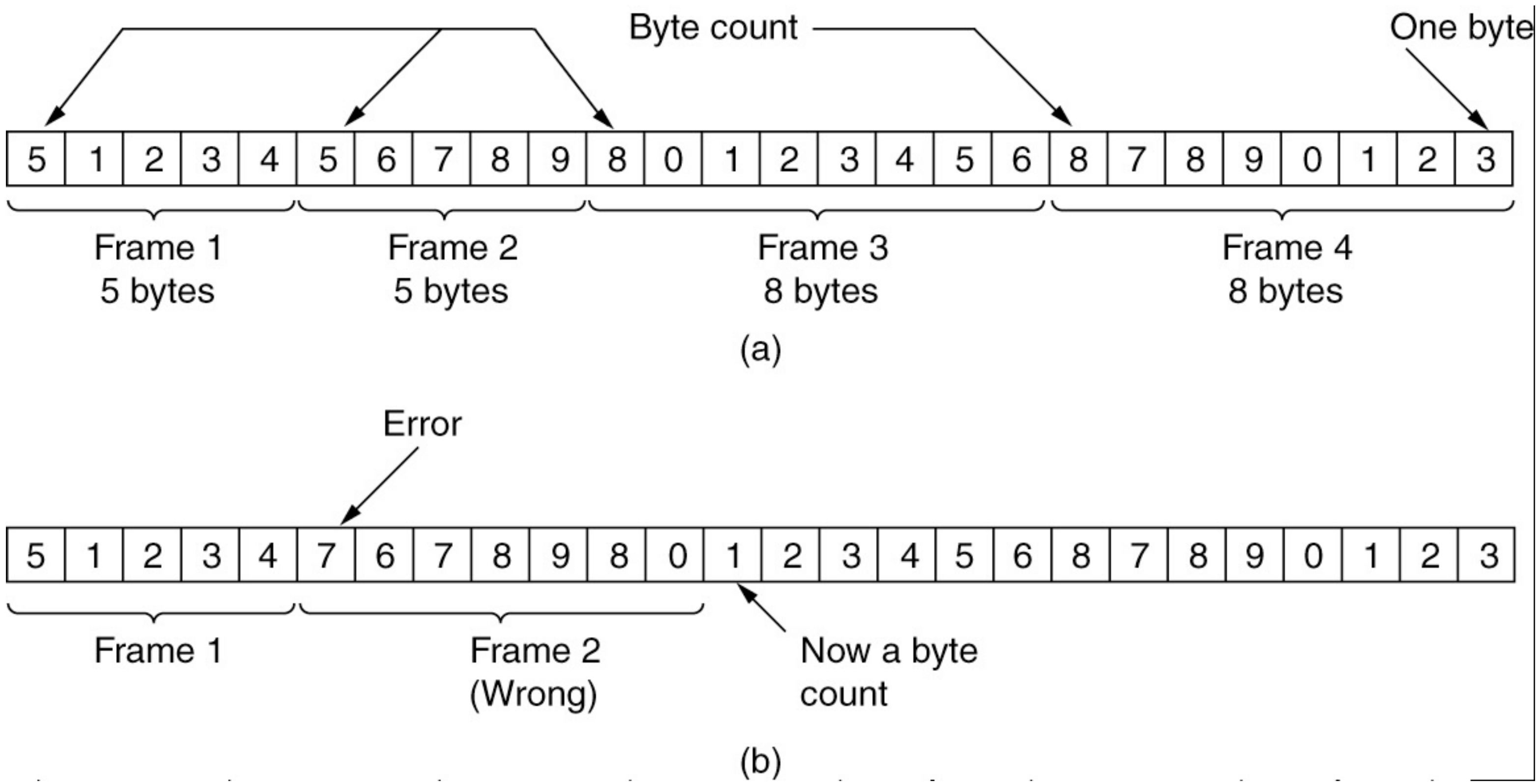
Services Provided to The Network Layer (2 of 2)

- Actual services vary from protocol to protocol
- Three possible services
  - Unacknowledged connectionless service
  - Acknowledged connectionless service
  - Acknowledged connection-oriented service

Framing (1 of 4)

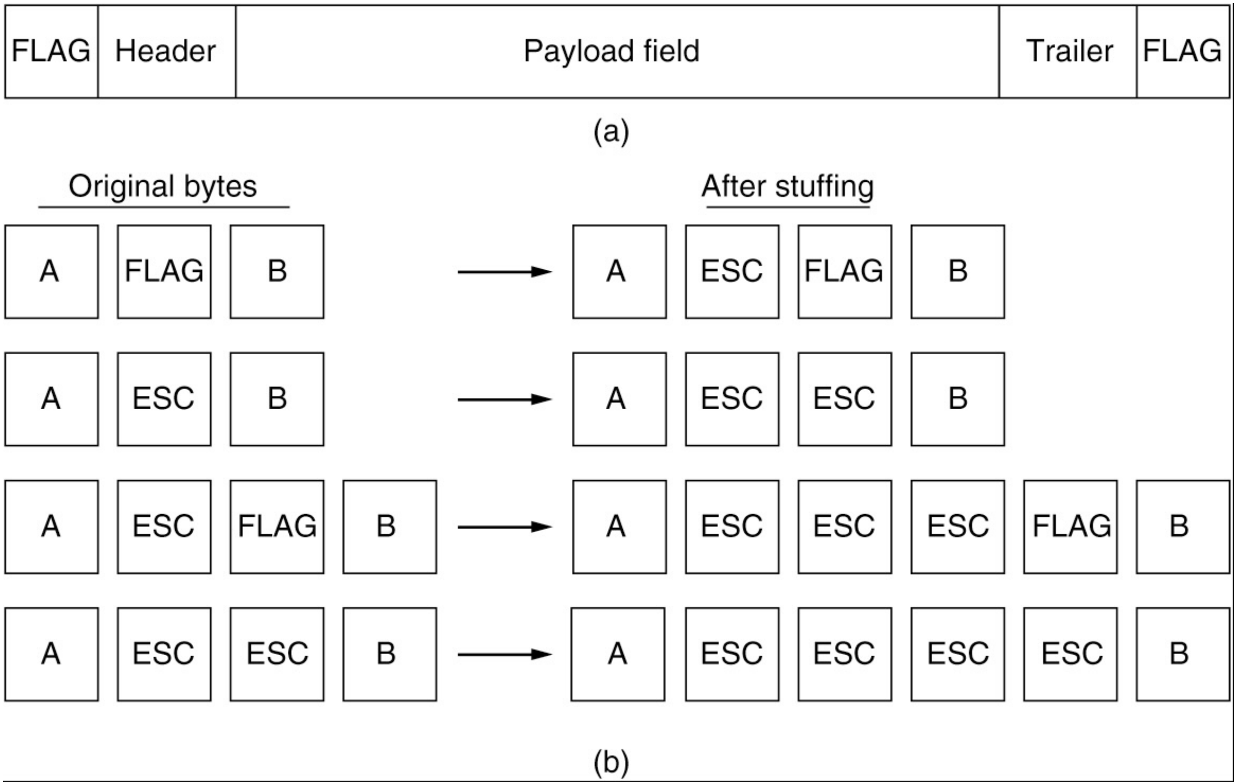
- Framing methods
  - Byte count
  - Flag bytes with byte stuffing
  - Flag bits with bit stuffing
  - Physical layer coding violations

Framing (2 of 4)



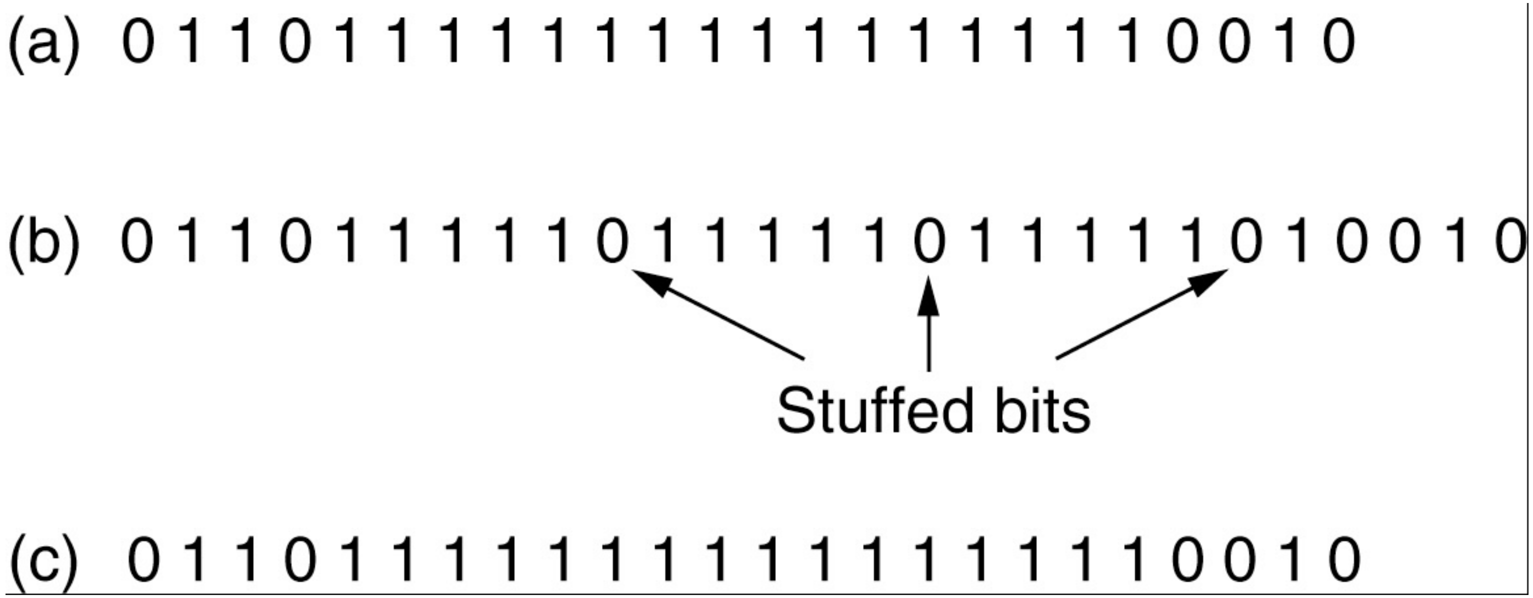
A byte stream. (a) Without errors. (b) With one error.

Framing (3 of 4)



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

Framing (4 of 4)



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

Error Control

- Ensuring all frames are eventually delivered:
  - To the network layer at the destination
  - In the proper order
- Ensures reliable, connection-oriented service
- Requires acknowledgement frames and timers

Flow Control

- Controlling the sending of transmission frames at a faster pace than they can be accepted
- Feedback-based flow control
  - Receiver sends back information to the sender giving it permission to send more data
  - Or receiver tells the sender how the receiver is doing
- Rate-based flow control
  - Protocol has a built-in mechanism
  - Mechanism limits the rate at which senders may transmit data
  - No feedback from the receiver is necessary

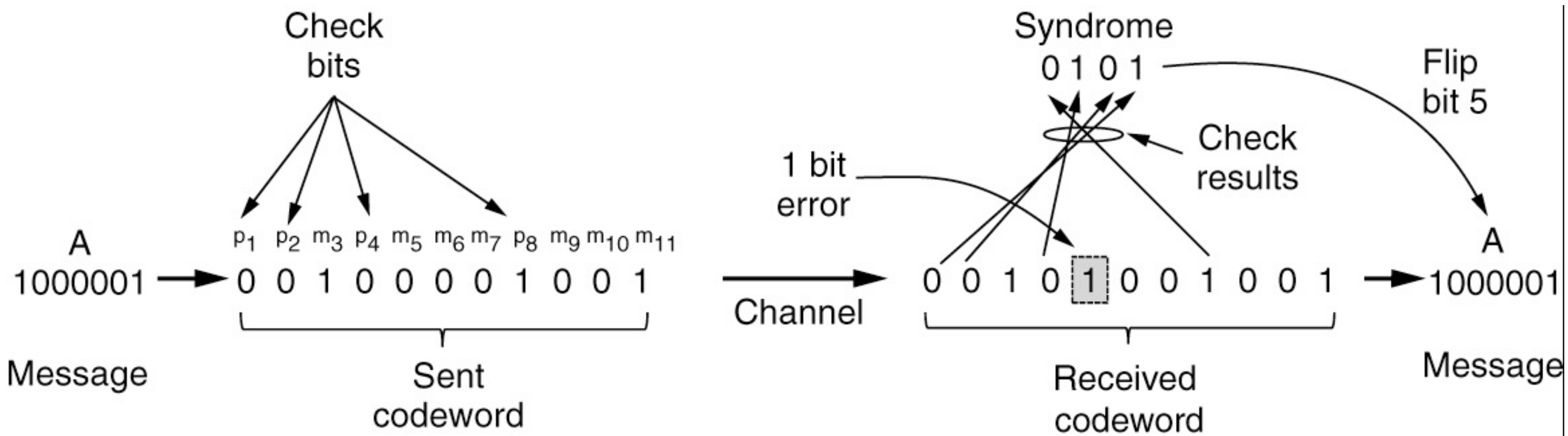
Error Detection and Correction

- Error-correcting codes
  - Referred to as FEC (Forward Error Correction)
  - Include enough redundant information to enable the receiver to deduce what the transmitted data must have been
- Error-detecting codes
  - Include only enough redundancy to allow the receiver to deduce that an error has occurred (but not which error) and have it request a retransmission
- Key consideration is the type of errors likely to occur

Error-Correcting Codes (1 of 4)

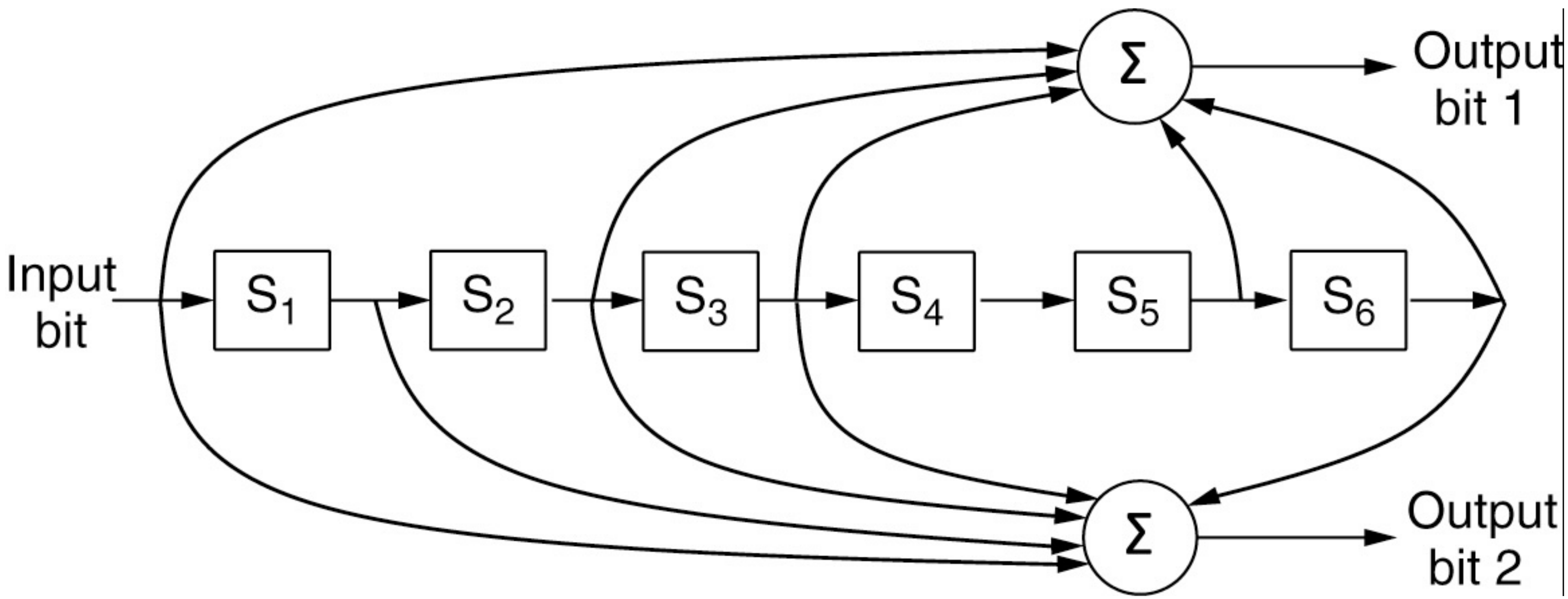
- Hamming codes
- Binary convolutional codes
- Reed-Solomon codes
- Low-Density Parity Check codes

Error-Correcting Codes (2 of 4)



Example of an (11, 7) Hamming code correcting a single-bit error

Error-Correcting Codes (3 of 4)



The NASA binary convolutional code used in 802.11
The encoder has memory!

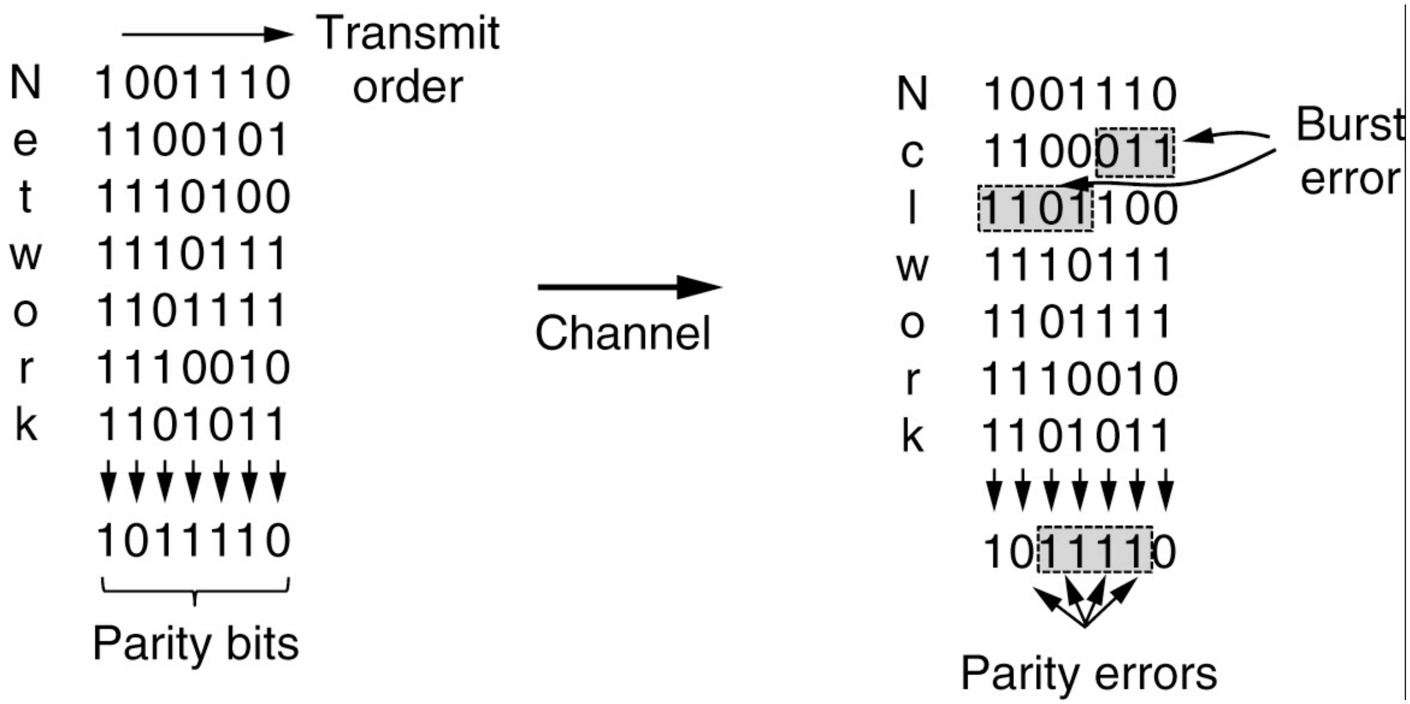
Error-Correcting Codes (4 of 4)

- Reed-Solomon codes
  - Linear block codes
  - Often systematic
  - Codes are based on the fact that every n degree polynomial is uniquely determined by n + 1 points
- LDPC (Low-Density Parity Check) codes
  - Linear block codes
  - Each output bit is formed from only a fraction of the input bits
  - Practical for large block sizes
  - Have excellent error-correction abilities that outperform many other codes

Error-Detecting Codes (1 of 3)

- Linear, systematic block codes
  - Parity
  - Checksums
  - Cyclic Redundancy Checks (CRCs)

Error-Detecting Codes (2 of 3)



Interleaving of parity bits to detect a burst error

Error-Detecting Codes (3 of 3)

- The word "checksum" is often used to mean a group of check bits associated with a message, regardless of how the bits are calculated.
- A group of parity bits is one example of a checksum.
- Stronger checksums based on a running sum of the data bits of the message.
- The checksum is usually placed at the end of the message, as the complement of the sum function.

Error-Detecting Codes (3 of 3)

- A third and stronger kind of error-detecting code is in widespread use at the link layer: the CRC (Cyclic Redundancy Check), also known as a polynomial code.

- Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only.

Example calculation of the CRC

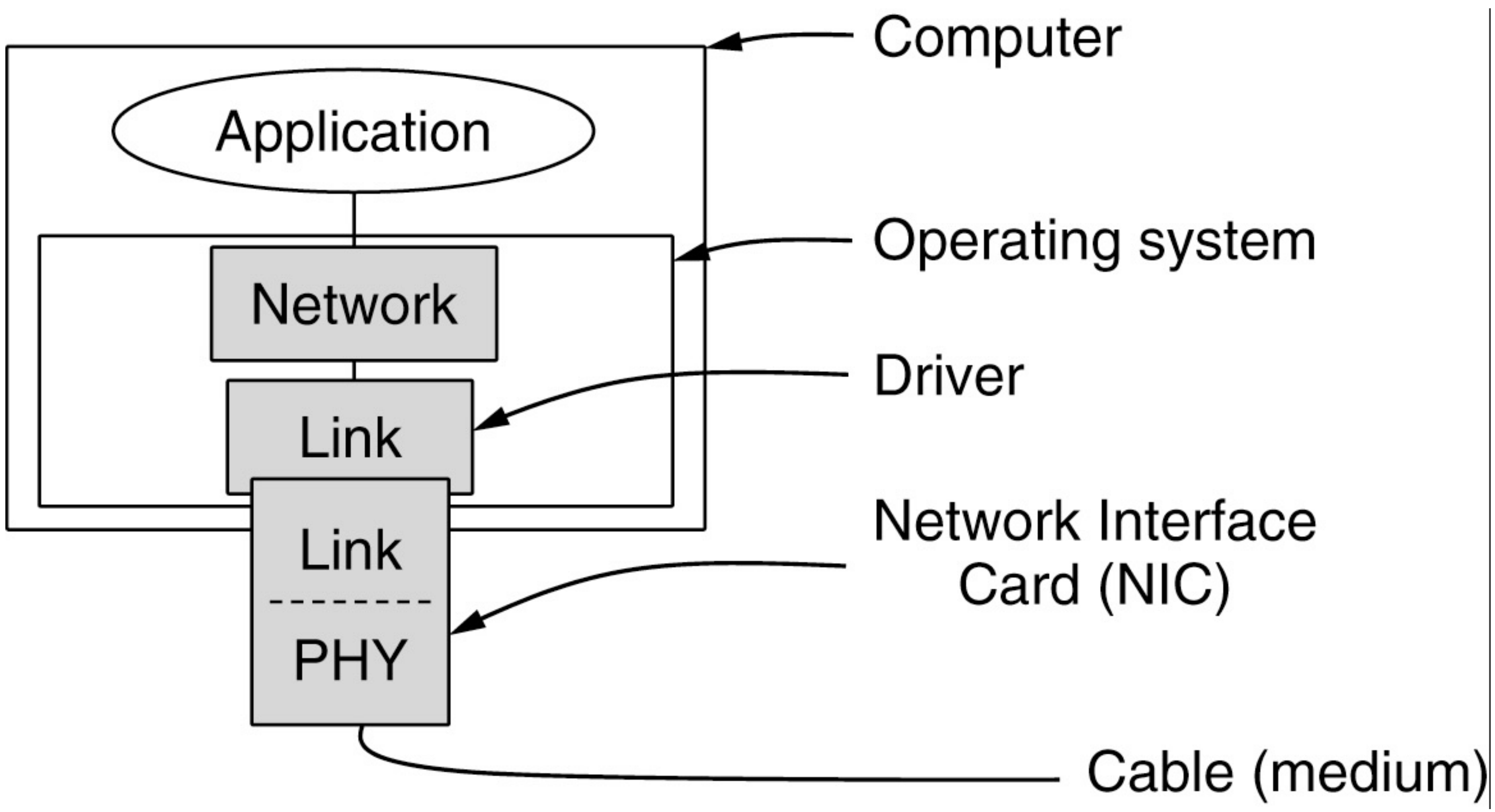
Elementary Data Link Protocols

- Assumptions underly the communication model
- Three simplex link-layer protocols
  - Utopia: No Flow Control or Error Correction
  - Adding Flow Control: Stop-and-Wait
  - Adding Error Correction: Sequence Numbers and ARQ

Initial Simplifying Assumptions (1 of 2)

- Independent processes
  - Physical, data link, and network layers are independent
  - Communicate by passing messages back and forth
- Unidirectional communication
  - Machines use a reliable, connection-oriented service
- Reliable machines and processes
  - Machines do not crash

Initial Simplifying Assumptions (2 of 2)



Implementation of the physical, data link, and network layers

Basic Transmission And Receipt (1 of 6)

```
#define MAX_PKT 4        /* 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 */
```

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h* … continued on the next slide

Basic Transmission And Receipt (2 of 6)

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

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h* … continued on the next slide

Basic Transmission And Receipt (3 of 6)

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

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

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

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h* … continued on the next slide

Basic Transmission And Receipt (4 of 6)

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

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

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

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h* … continued on the next slide

Basic Transmission And Receipt (5 of 6)

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

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

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

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h* … continued on the next slide

Basic Transmission And Receipt (6 of 6)

```
/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h* (end of code)

Simplex Link-Layer Protocols (1 of 10)

- Utopia: No Flow Control or Error Correction
- Adding Flow Control: Stop-and-Wait
- Adding Error Correction: Sequence Numbers and ARQ

Simplex Link-Layer Protocols (2 of 10)

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free,
   and the receiver is assumed to be able to process all the input infinitely
fast.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"
```

A utopian simplex protocol … continued on the next slide

Simplex Link-Layer Protocols (3 of 10)

```
void sender1(void)
{
  frame s;      /* buffer for an outbound frame */
  packet buffer;        /* buffer for an outbound packet */

  while (true) {
        from_network_layer(&buffer);    /* go get something to send */
        s.info = buffer;        /* copy it into s for transmission */
        to_physical_layer(&s);  /* send it on its way */
  }     /* tomorrow, and tomorrow, and tomorrow,
           Creeps in this petty pace from day to day
           To the last syllable of recorded time;
                - Macbeth, V, v */
}
```

A utopian simplex protocol … continued on the next slide

Simplex Link-Layer Protocols (4 of 10)

```
void receiver1(void)
{
  frame r;
  event_type event;     /* filled in by wait, but not used here */

  while (true) {
        wait_for_event(&event); /* only possibility is frame_arrival */
        from_physical_layer(&r);        /* go get the inbound frame */
        to_network_layer(&r.info);      /* pass the data to the network layer */
  }
}
```

A utopian simplex protocol (end of code)

Simplex Link-Layer Protocols (5 of 10)

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite procesing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"
```

A simplex stop-and-wait protocol … continued on the next slide

Simplex Link-Layer Protocols (6 of 10)

```
void sender2(void)
{
  frame s;      /* buffer for an outbound frame */
  packet buffer;        /* buffer for an outbound packet */
  event_type event;     /* frame_arrival is the only possibility */

  while (true) {
        from_network_layer(&buffer);    /* go get something to send */
        s.info = buffer;        /* copy it into s for transmission */
        to_physical_layer(&s);  /* bye bye little frame */
        wait_for_event(&event); /* do not proceed until given the go ahead */
  }
}
```

A simplex stop-and-wait protocol … continued on the next slide

Simplex Link-Layer Protocols (7 of 10)

```
void receiver2(void)
{
  frame r, s;   /* buffers for frames */
  event_type event;     /* frame_arrival is the only possibility */
  while (true) {
        wait_for_event(&event); /* only possibility is frame_arrival */
        from_physical_layer(&r);        /* go get the inbound frame */
        to_network_layer(&r.info);      /* pass the data to the network layer */
        to_physical_layer(&s);  /* send a dummy frame to awaken sender */
  }
}
```

A simplex stop-and-wait protocol (end of code)

Simplex Link-Layer Protocols (8 of 10)

```
/* 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"
```

A positive acknowledgement with retransmission protocol … continued on the next slide

Simplex Link-Layer Protocols (9 of 10)

```
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) {
                        from_network_layer(&buffer);    /* get the next one to send */
                        inc(next_frame_to_send);        /* invert next_frame_to_send */
                }
        }
  }
}
```

A positive acknowledgement with retransmission protocol … continued on the next slide

Simplex Link-Layer Protocols (10 of 10)

```
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);  /* only the ack field is use */
        }
  }
}
```

A positive acknowledgement with retransmission protocol (end of code)

Improving Efficiency

- Need bidirectional data transmission
- Link layer efficiency improvement
  - Send multiple frames simultaneously before receiving an acknowledgement

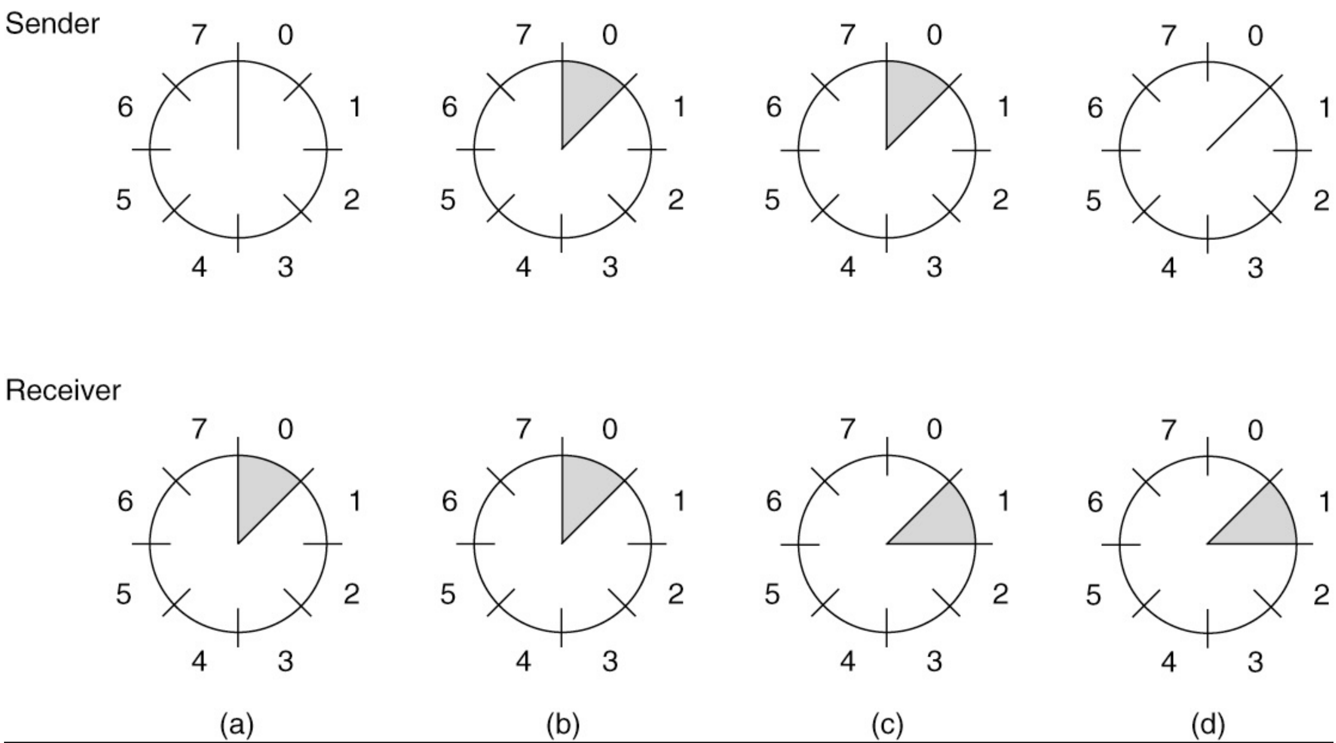
Bidirectional Transmission, Multiple Frames in Flight (1 of 3)

- Bidirectional transmission: piggybacking
  - Use the same link for data in both directions
  - Interleave data and control frames on the same link
  - Temporarily delay outgoing acknowledgements so they can be hooked onto the next outgoing data frame
- Piggybacking advantages
  - A better use of the available channel bandwidth
  - Lighter processing load at the receiver
- Piggybacking issue
  - Determining time data link layer waits for a packet to piggyback the acknowledgement

Bidirectional Transmission, Multiple Frames in Flight (2 of 3)

- Three bidirectional sliding window protocols
  - One-bit sliding window, go-back-n, selective repeat
- Consider any instant of time
  - Sender maintains a set of sequence numbers corresponding to frames it is permitted to send
  - Frames are said to fall within the sending window
  - Receiver maintains a receiving window corresponding to the set of frames it is permitted to accept
- Differ among themselves in terms of efficiency, complexity, and buffer requirements

Bidirectional Transmission, Multiple Frames in Flight (3 of 3)



A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

Examples (1 of 20)

```
/* Protocol 4 (sliding window) is bidirectional and is more robust than protocol 3. */

#define MAX_SEQ 1       /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
```

A 1-bit sliding window protocol … continued on the next slide

Examples (2 of 20)

```
void protocol4 (void)
{
  seq_nr next_frame_to_send;    /* 0 or 1 only */
  seq_nr frame_expected;        /* 0 or 1 only */
  frame r, s;   /* scratch variables */
  packet buffer;        /* current packet being sent */
  event_type event;

  next_frame_to_send = 0;       /* next frame on the outbound stream */
  frame_expected = 0;   /* number of frame arriving frame expected */
  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 */
```

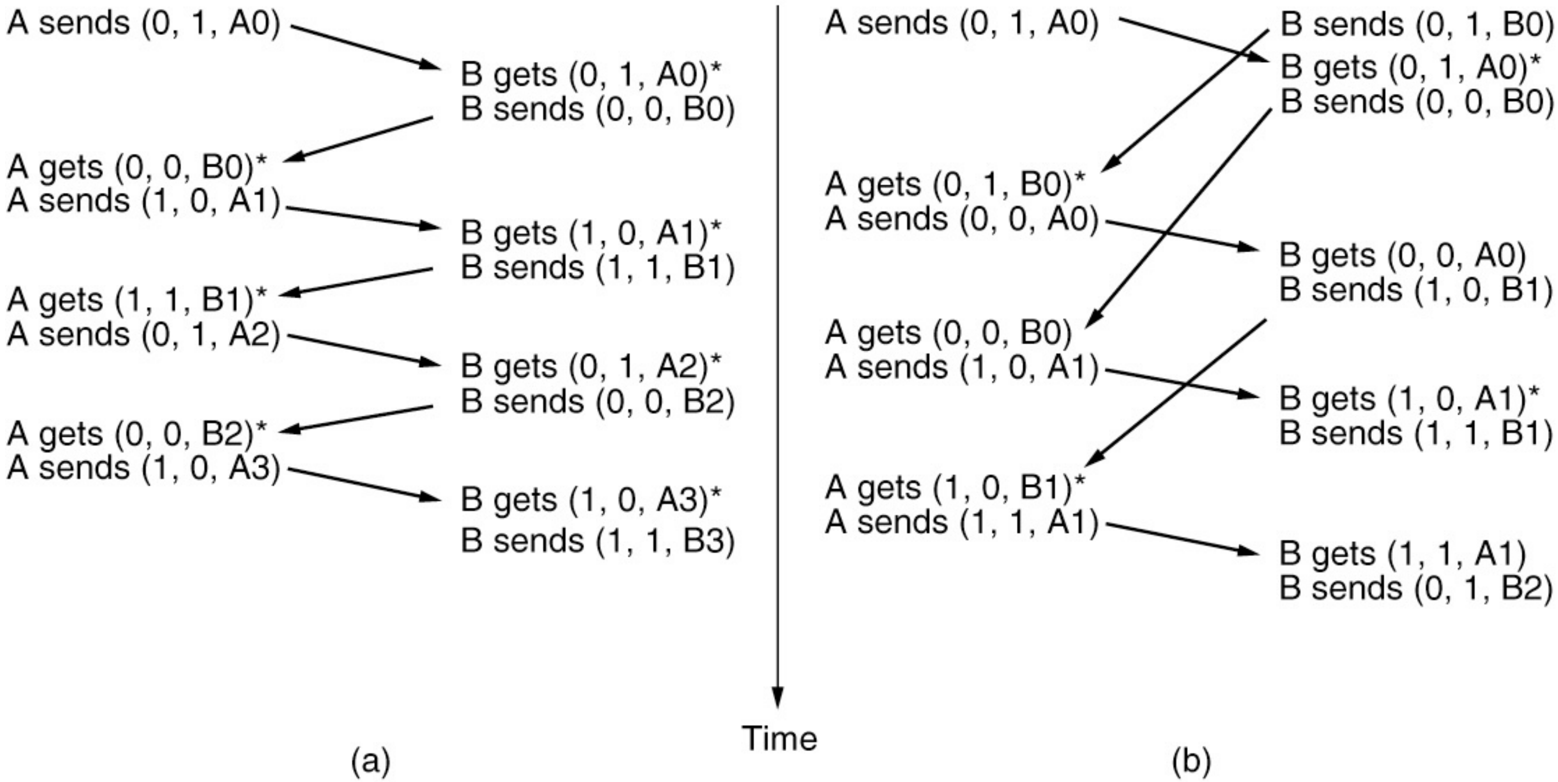
A 1-bit sliding window protocol … continued on the next slide

Examples (3 of 20)

```
while (true) {
      wait_for_event(&event); /* could be: frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) { /* a frame has arrived undamaged. */
              from_physical_layer(&r);        /* go get it */

              if (r.seq == frame_expected) {
                      /* Handle inbound frame stream. */
                      to_network_layer(&r.info);      /* pass packet to network layer */
                      inc(frame_expected);    /* invert sequence number expected next */
              }

              if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
                      from_network_layer(&buffer);    /* fetch new packet from network layer */
                      inc(next_frame_to_send);        /* invert sender's sequence number */
              }
      }

      s.info = buffer;        /* construct outbound frame */
      s.seq = next_frame_to_send;     /* insert sequence number into it */
      s.ack = 1 - frame_expected;     /* seq number of last received frame */
      to_physical_layer(&s);  /* transmit a frame */
      start_timer(s.seq);     /* start the timer running */
}
}
```

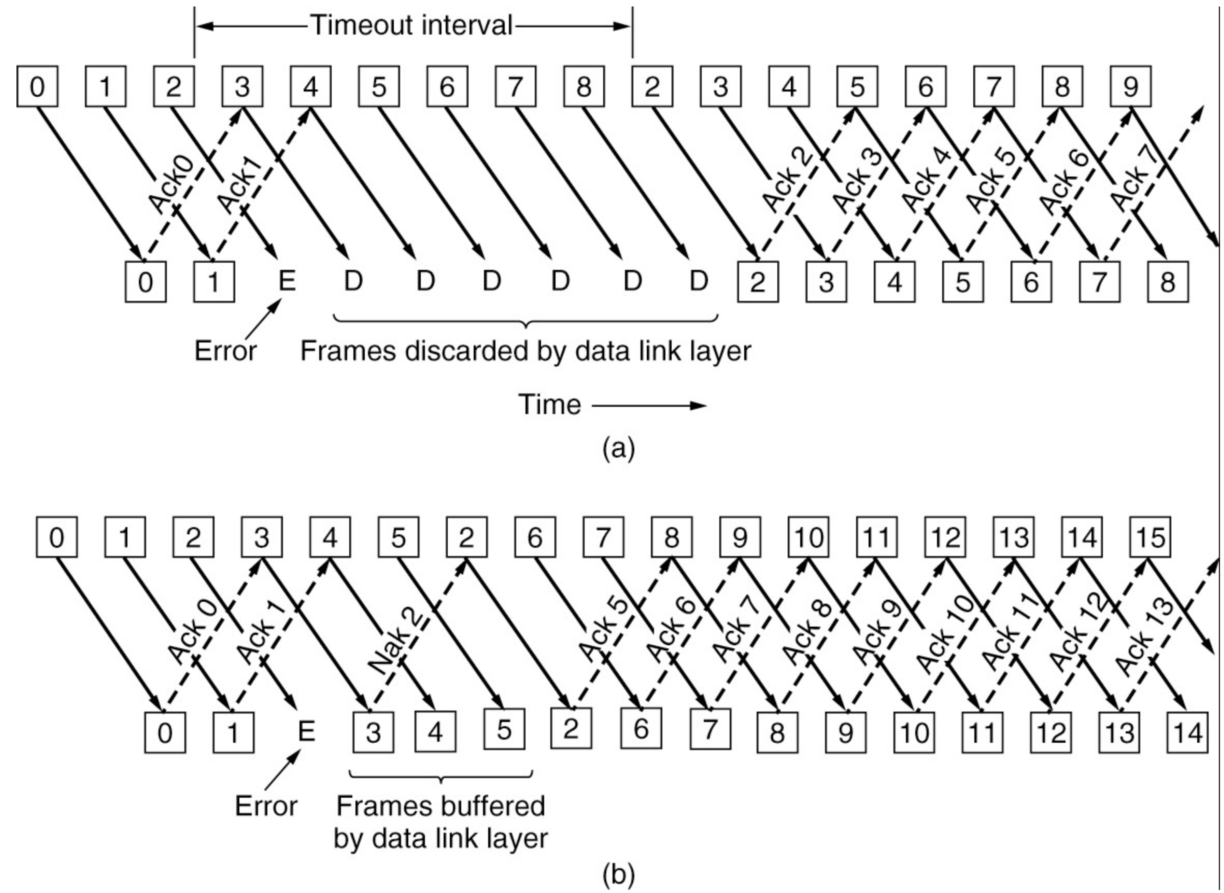
A 1-bit sliding window protocol (end of code)

Examples (4 of 20)



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

Examples (5 of 20)



Pipelining and error recovery. Effect of an error when (a) receiver's window size is 1 and (b) receiver's window size is large.

Examples (6 of 20)

```
/* Protocol 5 (Go-back-n) 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"
```

A sliding window protocol using go-back-n … continued on the next slide

Pearson

Examples (7 of 20)

```
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);
}
```

A sliding window protocol using go-back-n … continued on the next slide

Examples (8 of 20)

```
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 */
}
```

A sliding window protocol using go-back-n … continued on the next slide

Examples (9 of 20)

```
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 */
```

A sliding window protocol using go-back-n … continued on the next slide

Examples (10 of 20)

```
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 */
               }
```

A sliding window protocol using go-back-n … continued on the next slide

Examples (11 of 20)

```
 /* 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: ;       /* just ignore bad frames */
        break;
```

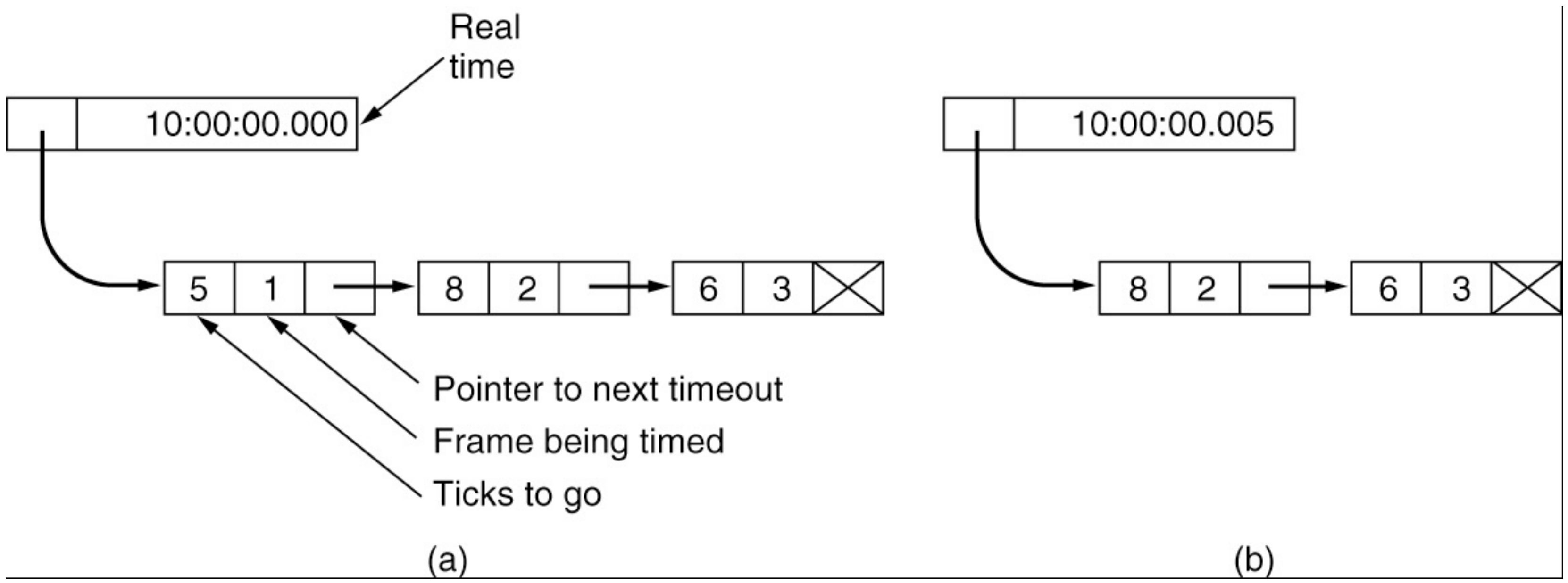
A sliding window protocol using go-back-n … continued on the next slide

Examples (12 of 20)

```
        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();
  }
}
```

A sliding window protocol using go-back-n (end of code)

Examples (13 of 20)



Simulation of multiple timers in software. (a) The queued timeouts. (b) The situation after the first timeout has expired.

Examples (14 of 20)

```
/* 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
   goes off, 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;        /* init value is for the simulator */
```

A sliding window protocol using selective repeat … continued on the next slide

Examples (15 of 20)

```
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 */
}
```

A sliding window protocol using selective repeat … continued on the next slide

Examples (16 of 20)

```
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;   /* frame number expected */
  too_far = NR_BUFS;    /* receiver's upper window + 1 */
  nbuffered = 0;        /* initially no packets are buffered */
```

A sliding window protocol using selective repeat … continued on the next slide

Examples (17 of 20)

```
for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
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;
```

A sliding window protocol using selective repeat … continued on the next slide

Examples (18 of 20)

```
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 */
                      }
              }
      }
```

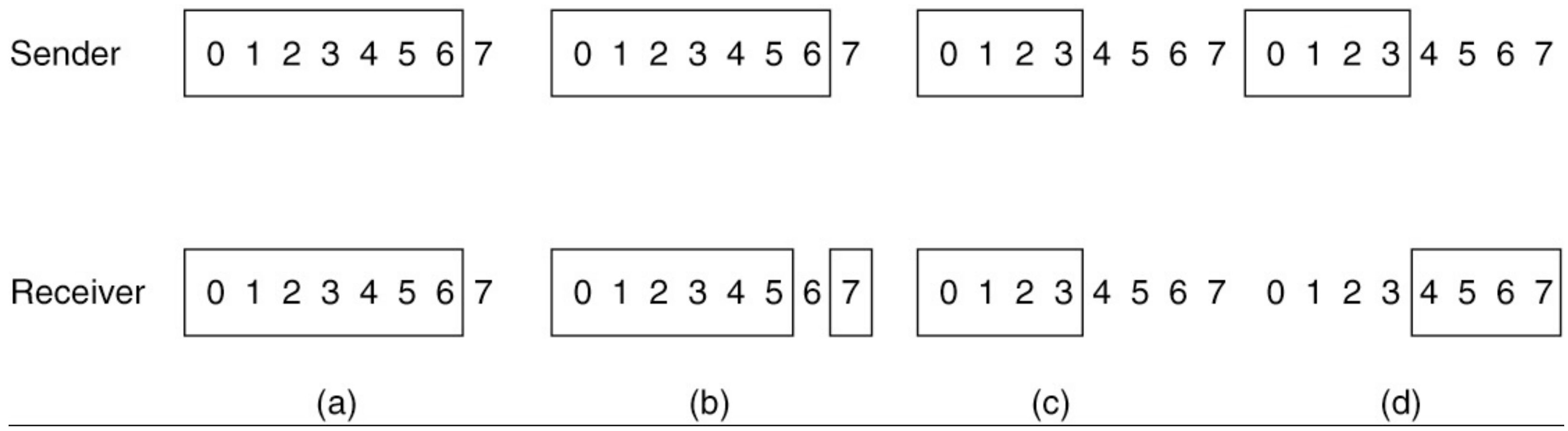
A sliding window protocol using selective repeat … continued on the next slide

Examples (19 of 20)

```
            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); break; /* damaged frame */
    case timeout: send_frame(data, oldest_frame, frame_expected, out_buf); break;   /* we timed out */
    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();
 }
}
```

A sliding window protocol using selective repeat (end of code)

Examples (20 of 20)

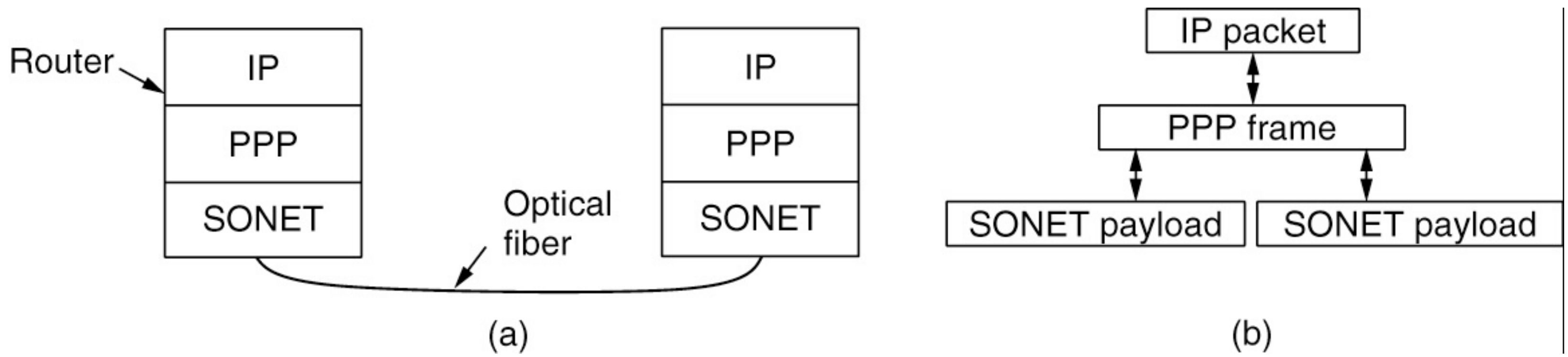


(a) Initial situation with a window of size 7. (b) After 7 frames have been sent and received but not acknowledged. (c) Initial situation with a window size of 4. (d) After 4 frames have been sent and received but not acknowledged.

Data Link Protocols in Practice

- Packet over SONET
- ADSL (Asymmetric Digital Subscriber Loop)

Packet over SONET (1 of 4)



Packet over SONET. (a) A protocol stack. (b) Frame relationships.

Packet over SONET (2 of 4)

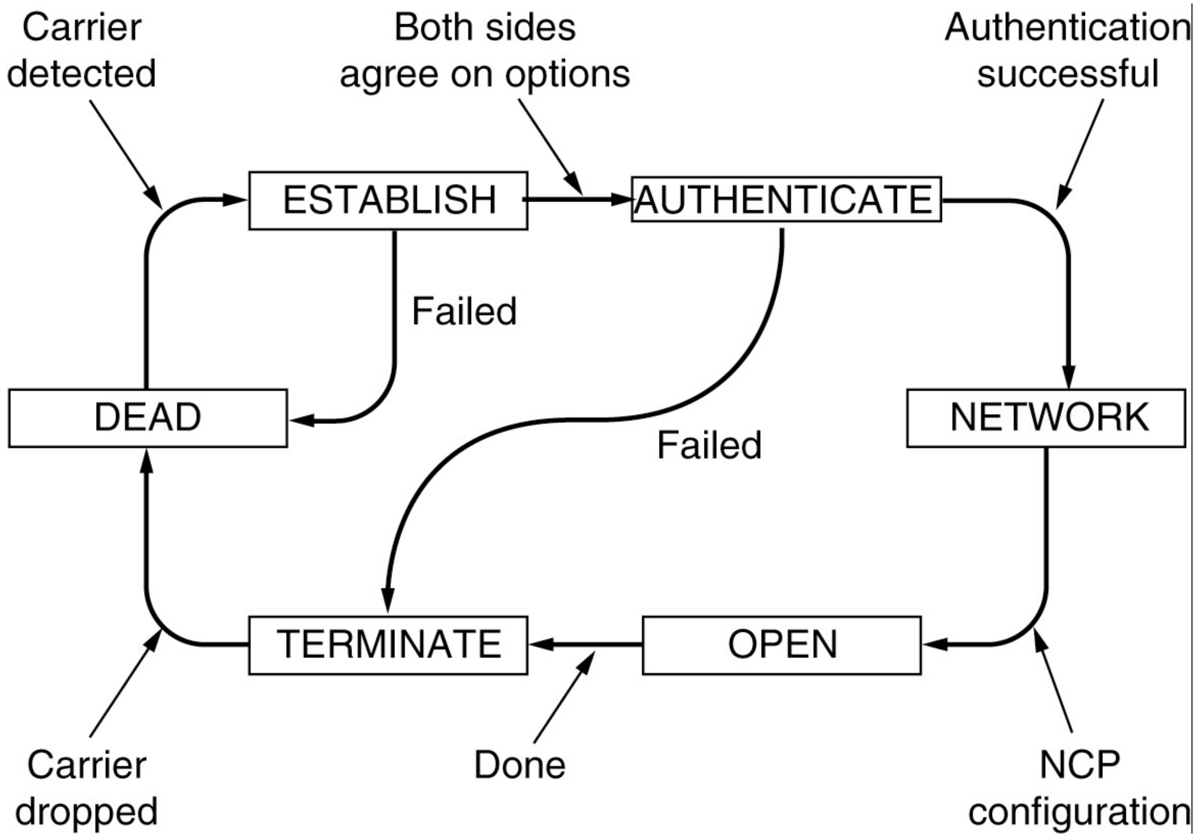
- PPP (Point-to-Point Protocol) features
  - Separate packets, error detection
  - Link Control Protocol (LCP)
  - Network Control Protocol (NCP)

Packet over SONET (3 of 4)

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

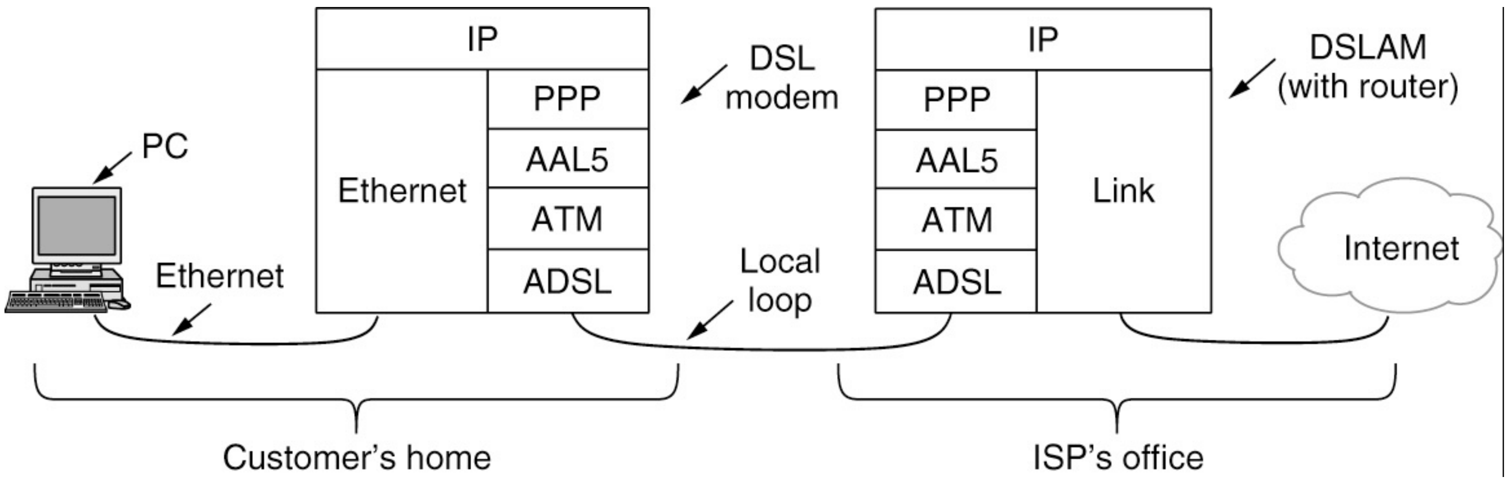
The PPP full frame format for unnumbered mode operation

Packet over SONET (4 of 4)



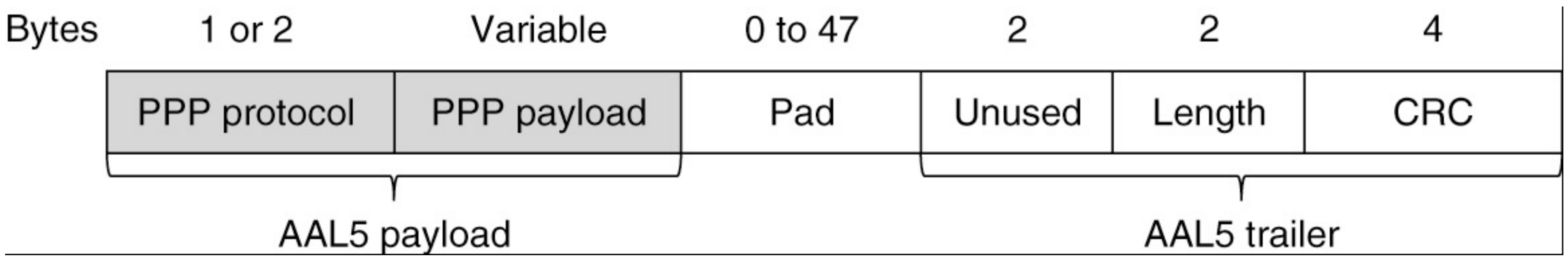
State diagram for bringing a PPP link up and down

ADSL (1 of 2)



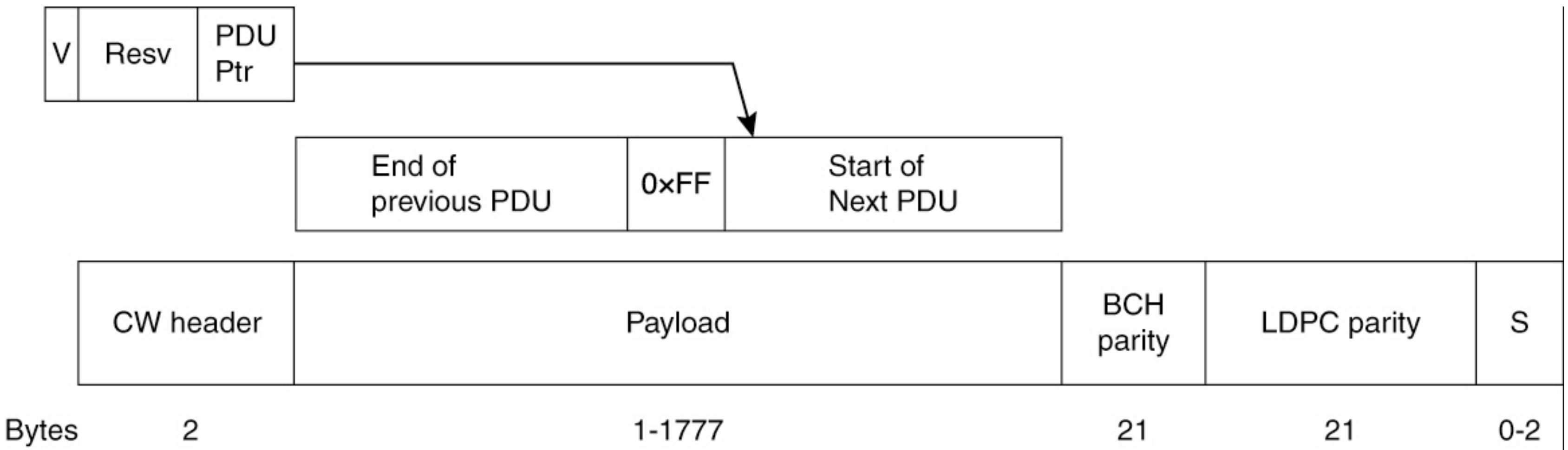
ADSL protocol stacks

ADSL (2 of 2)



AAL5 frame carrying PPP data

Data Over Cable Service Interface Specification (DOCSIS)



DOCSIS Frame to codeword mapping

Copyright