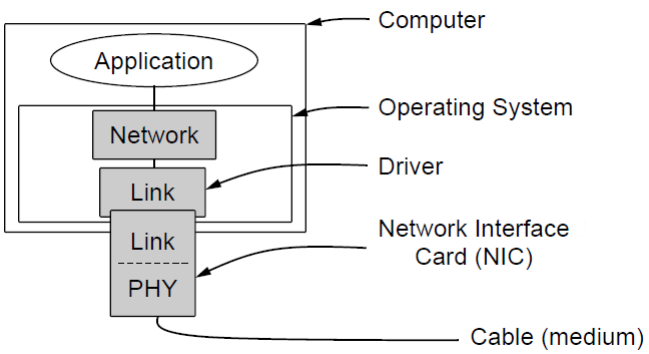
Elementary Data Link Protocols (1)

- Utopian Simplex Protocol
- Simplex Stop-and-Wait Protocol
  - Error-Free Channel
- Simplex Stop-and-Wait Protocol
  - Noisy Channel

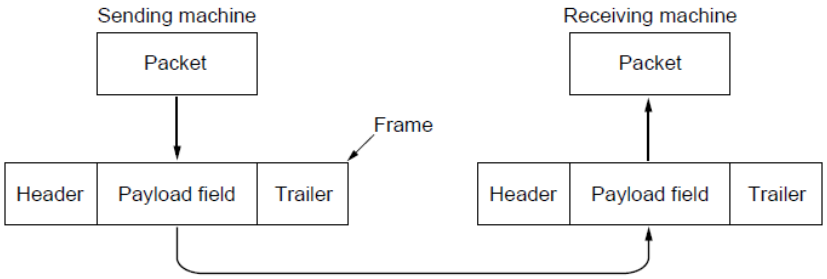
Elementary Data Link Protocols (2)



Implementation of the physical, data link, and network layers.

Elementary Data Link Protocols

- Data Link layer job is to:
  - FRAMING: encapsulate the data packets in a frame by adding the header and trailer.



80

Elementary Data Link Protocols

- Error Correction and Detection:
  - Control information is added to header, and
  - Checksum is added to trailer.
- Frame is then transmitted to the other machine.

81

Elementary Data Link Protocols

- Library procedures:
  - to_physical_layer
  - from_physical_layer
  - wait_for_event(&event)
    - event object will contain the information what has happened.
  - In the receiving end:
    - Data is being received and the checksum is being computed.
    - If the error has occurred the data link layer is being informed: event = chksum_err
    - If the data has arrived undamaged: event = frame_arrival.

82

Elementary Data Link Protocols (3)

```
#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;
```

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h*. . .

83

Elementary Data Link Protocols (4)

```
/* 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);

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

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

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

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

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h.*

84

Elementary Data Link Protocols (5)

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

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

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

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

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

Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h.*

85

Utopian Simplex Protocol

- As an initial example we will consider thoroughly unrealistic protocol, which we will nickname ‘‘Utopia,’’ is simply to show the basic structure on which we will build.
- Utopia protocol is as simple as it can be because it does not worry about the possibility of anything going wrong.
- Data are transmitted in one direction only.
- Both the transmitting and receiving network layers are always ready.
- Processing time can be ignored. Infinite buffer space is available.
- And best of all, the communication channel between the data link layers never damages or loses frames.

86

Utopian Simplex Protocol

- Assumptions:
  1. A wants to send a long stream of data to machine B.
  2. It uses reliable connection-oriented service.
  3. It assumes to have infinite supply of data ready to send,
  4. It does not have to wait for data to be produced.
  5. Machines A and B do not crash
  6. Data link layer treats the data as packets of pure data whose every bit is to be delivered to the destination's network layer.

87

Utopian Simplex Protocol (1)

```
/* 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 quickly.
   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"

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.

88

Utopian Simplex Protocol (2)

```
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.

89

Utopian Simplex Protocol

- Unrealistic
  - It does not handle flow control
  - It does not handle error correction
  - Resembles connectionless service that relies on higher layer to solve those problems.

90

A Simplex Stop-and-Wait Protocol for an Error-Free Channel

- Now we will tackle the problem of preventing the sender from flooding the receiver with frames faster than the latter is able to process them.
- A general solution to this problem is to have the receiver provide feedback to the sender.
- After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame.
- After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgement) frame arrives.

91

A Simplex Stop-and-Wait Protocol for an Error-Free Channel

- Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait.
- Although data traffic in this example is simplex, going only from the sender to the receiver, frames do travel in both directions.
- A halfduplex physical channel would suffice here.

92

Simplex Stop-and-Wait Protocol for a Noise-less Channel (1)

```
/* 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 processing 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"

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.

93

Simplex Stop-and-Wait Protocol for a Noise-less Channel (2)

```
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.

A Simplex Stop-and-Wait Protocol for an Error-Free Channel

- Solves the problem of flow control.
  - Blocking acknowledgment of the reception of the frame partially achieves this.
  - If acknowledgment frame gets lost the protocol will fail.
  - Error correction is not implemented.
  - One direction only.

Simplex Stop-and-Wait Protocol for a Noisy Channel

- Now let us consider the normal situation of a communication channel that makes errors.
- Frames may be either damaged or lost completely.
- However, we assume that if a frame is damaged in transit, the receiver hardware will detect this when it computes the checksum.
- If the frame is damaged in such a way that the checksum is nevertheless correct—an unlikely occurrence—this protocol (and all other protocols) can fail (i.e., deliver an incorrect packet to the network layer)

96

Simplex Stop-and-Wait Protocol for a Noisy Channel

- At first glance it might seem that a variation of previous protocol would work: adding a timer.
  - The sender could send a frame, but the receiver would only send an acknowledgement frame if the data were correctly received.
  - If a damaged frame arrived at the receiver, it would be discarded.
  - After a while the sender would time out and send the frame again.
  - This process would be repeated until the frame finally arrived intact.
- This scheme has a fatal flaw in it though.

97

Simplex Stop-and-Wait Protocol for a Noisy Channel

- The goal of the data link layer is to provide error-free, transparent communication between network layer processes.
- The network layer on machine A gives a series of packets to its data link layer, which must ensure that an identical series of packets is delivered to the network layer on machine B by its data link layer.
- In particular, the network layer on B has no way of knowing that a packet has been lost or duplicated.

98

Simplex Stop-and-Wait Protocol for a Noisy Channel

- Consider the following scenario:
  - The network layer on A gives packet 1 to its data link layer. The packet is correctly received at B and passed to the network layer on B. B sends an acknowledgement frame back to A.
  - The acknowledgement frame gets lost completely. It just never arrives at all.
  - The data link layer on A eventually times out. Not having received an acknowledgement, it (incorrectly) assumes that its data frame was lost or damaged and sends the frame containing packet 1 again.
  - The duplicate frame also arrives intact at the data link layer on B and is unwittingly passed to the network layer there. If A is sending a file to B, part of the file will be duplicated . In other words, the protocol will fail.

99

Simplex Stop-and-Wait Protocol for a Noisy Channel

- Clearly, what is needed is some way for the receiver to be able to distinguish a frame that it is seeing for the first time from a retransmission.
- The obvious way to achieve this is to have the sender put a sequence number in the header of each frame it sends.
- Then the receiver can check the sequence number of each arriving frame to see if it is a new frame or a duplicate to be discarded.
- The header might provide 1 bit, a few bits, 1 byte, or multiple bytes for a sequence number depending on the protocol.

100

Simplex Stop-and-Wait Protocol for a Noisy Channel

- A 1-bit sequence number (0 or 1) is therefore sufficient.
- At each instant of time, the receiver expects a particular sequence number next. When a frame containing the correct sequence number arrives, it is accepted and passed to the network layer, then acknowledged.
- Then the expected sequence number is incremented modulo 2 (i.e., 0 becomes 1 and 1 becomes 0).
- Any arriving frame containing the wrong sequence number is rejected as a duplicate.
- However, the last valid acknowledgement is repeated so that the sender can eventually discover that the frame has been received.
- Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called **ARQ** (Automatic Repeat reQuest) or **PAR** (Positive Acknowledgement with Retransmission).

101

Simplex Stop-and-Wait Protocol for a Noisy Channel

- After transmitting a frame and starting the timer, the sender waits for something exciting to happen.
- Only three possibilities exist: an acknowledgement frame arrives undamaged, a damaged acknowledgement frame staggers in, or the timer expires.
  - If a valid acknowledgement comes in, the sender fetches the next packet from its network layer and puts it in the buffer, overwriting the previous packet. It also advances the sequence number.
  - If a damaged frame arrives or the timer expires, neither the buffer nor the sequence number is changed so that a duplicate can be sent.
  - In all cases, the contents of the buffer (either the next packet or a duplicate) are then sent.

102

Simplex Stop-and-Wait Protocol for a Noisy Channel

- When a valid frame arrives at the receiver, its sequence number is checked to see if it is a duplicate.
- If not, it is accepted, passed to the network layer, and an acknowledgement is generated.
- Duplicates and damaged frames are not passed to the network layer, but they do cause the last correctly received frame to be acknowledged to signal the sender to advance to the next frame or retransmit a damaged frame.

103

Simplex Stop-and-Wait Protocol for a Noisy Channel (1)

```
/* Protocol 3 (PAR) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                       /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;            /* seq number of next outgoing frame */
  frame s;                              /* scratch variable */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;
```

A positive acknowledgement with retransmission protocol.

104

Simplex Stop-and-Wait Protocol for a Noisy Channel (2)

```
  next_frame_to_send = 0;               /* initialize outbound sequence numbers */
  from_network_layer(&buffer);          /* fetch first packet */
  while (true) {
      s.info = buffer;                  /* construct a frame for transmission */
      s.seq = next_frame_to_send;       /* insert sequence number in frame */
      to_physical_layer(&s);            /* send it on its way */
      start_timer(s.seq);               /* if answer takes too long, time out */
      wait_for_event(&event);           /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
          from_physical_layer(&s);      /* get the acknowledgement */
          if (s.ack == next_frame_to_send) {
              stop_timer(s.ack);        /* turn the timer off */
              from_network_layer(&buffer); /* get the next one to send */
              inc(next_frame_to_send);  /* invert next_frame_to_send */
          }
      }
  }
}
```

A positive acknowledgement with retransmission protocol.

105

Simplex Stop-and-Wait Protocol for a Noisy Channel(3)

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

 frame_expected = 0;
 while (true) {
     wait_for_event(&event);                 /* possibilities: frame_arrival, cksum_err */
     if (event == frame_arrival) {           /* a valid frame has arrived */
          from_physical_layer(&r);           /* go get the newly arrived frame */
          if (r.seq == frame_expected) {     /* this is what we have been waiting for */
               to_network_layer(&r.info);    /* pass the data to the network layer */
               inc(frame_expected);          /* next time expect the other sequence nr */
          }
          s.ack = 1 – frame_expected;        /* tell which frame is being acked */
          to_physical_layer(&s);             /* send acknowledgement */
     }
 }
}
```

A positive acknowledgement with retransmission protocol.

106

Sliding Window Protocols

- In the previous protocols, data frames were transmitted in one direction only.
- In most practical situations, there is a need to transmit data in both directions. One way of achieving this is to:
  - Running two instances of one of the previous protocols,
  - Each uses separate data link for simplex data traffic in different directions (“forward” channel and “reverse” channel).
- Better idea is to use the same channel in both directions.
  - A and B data frames are intermixed with acknowledgment frames from A and B.
  - Kind field in the header of the incoming frame will be used to tell the difference.

107

Sliding Window Protocols

- In addition to being able to use duplex communication, the further protocol improvement would be to send the acknowledgment frame together with the new data frame.
- This provides a "free ride" to the acknowledgement on the next outgoing data frame.
- The technique of delaying the acknowledgment for a bit to be added to the next frame is know as **piggybacking**.
- The principal advantage of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth.
- The ack field in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum.

108

Sliding Window Protocols

- This technique add another complication.
- ***How long should the data link layer wait for a packet onto which to piggyback the acknowledgement?***
- If the data link layer waits longer than the sender's timeout period, the frame will be retransmitted, defeating the whole purpose of having acknowledgements.
  - If the data frame is available within a short time period (less than msec) it will use piggybacking.
  - Otherwise, it will send separate acknowledgment.

109

Sliding Window

- At any instant of time the sender maintains a set of sequences numbers that correspond to frames that are permitted to be send.
- Those frames are said to fall within a **sending window**.
- Receiver also maintains a **receiving window** that corresponds to the set of frames that it is permitted to accept.

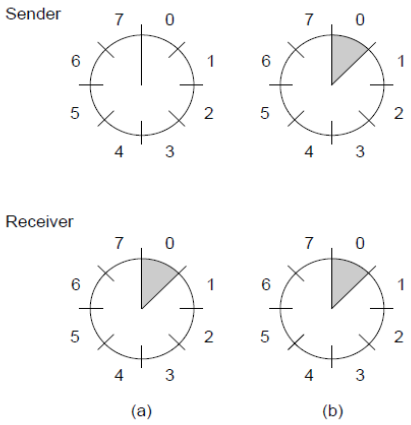
110

Sliding Window

- The sequence numbers within the sender's window represent the frames that have been sent or can be sent but are not yet been acknowledged.
- Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and
- The upper edge of the window is advanced by one.
- When acknowledgement arrives the lower edge is advance by one.
- This is how the protocol maintains the list of unacknowledged frames.

111

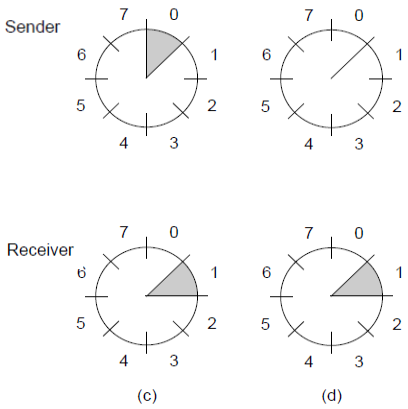
Sliding Window Protocols



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

112

Sliding Window Protocols



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

113

One-Bit Sliding Window Protocol

- Window size = 1
- Stop-and-wait
  Sender transmits a frame and waits for its acknowledgment before it sends the next one.
- Figure of next slide depicts such a protocol.

One-Bit Sliding Window Protocol (1)

```
/* Protocol 4 (Sliding window) is bidirectional. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 0 or 1 only */
  seq_nr frame_expected;                            /* 0 or 1 only */
  frame r, s;                                       /* scratch variables */
  packet buffer;                                    /* current packet being sent */
  event_type event;

  next_frame_to_send = 0;                           /* next frame on the outbound stream */
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network layer */
  s.info = buffer;                                  /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                       /* insert sequence number into frame */
  s.ack = 1 – frame_expected;                       /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
```

A 1-bit sliding window protocol.

One-Bit Sliding Window Protocol (2)

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

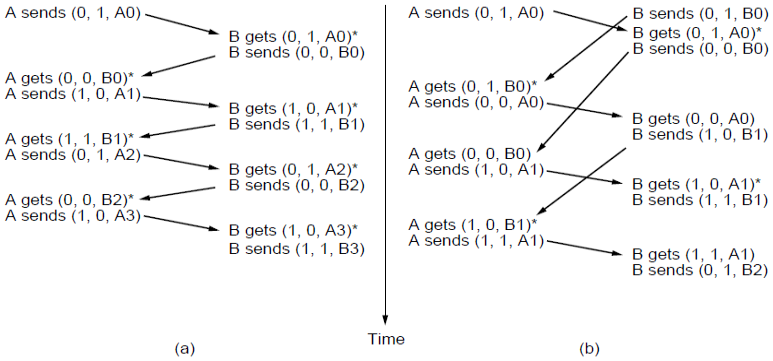
A 1-bit sliding window protocol.

One-Bit Sliding Window Protocol (3)

```
        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.

One-Bit Sliding Window Protocol (4)

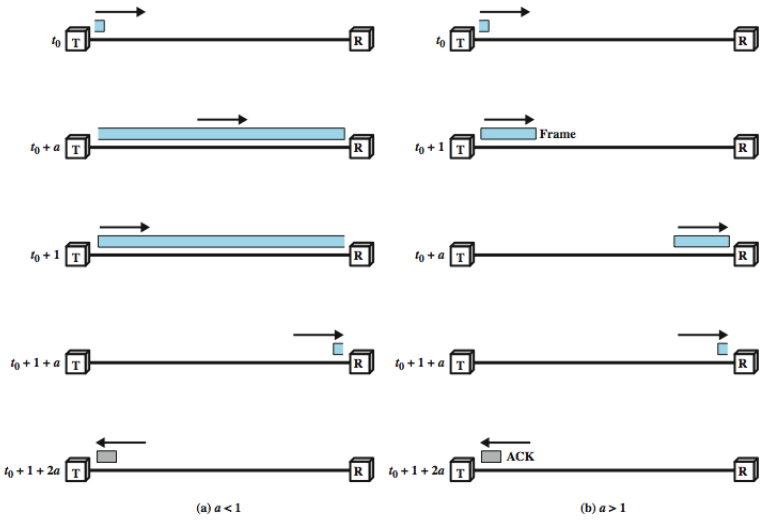


Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case.

The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet

118

One-Bit Sliding Window Protocol Link Utilization



119

Sliding Window Protocols

- Up-to-know the protocols depicted so far are based on the assumption that the time required for transmission and reception of the acknowledgment packets is negligible.
- If this assumptions is not true then round-trip time can have important implications in the efficiency of the communication and the bandwidth utilization.

120

Sliding Window Protocols

- Example:
  - 50 kbps satellite channel
  - 500 msec round-trip propagation delay.
  - Lets use Protocol 4 (previous slides) to send a 1000 bit frames via the satellite.
  - t = 0: the sender starts the first frame
  - t = 20 msec: the frame has been completely sent.
  - t = 270 msec: the frame has fully arrived at the satellite.
  - t = 520 msec: the acknowledgment has arrived at the sender.
  - Sender was blocked 500/520 or 96% of the time. In order to send the packed the sender utilized 4% of the available bandwidth.
- Clearly, the combination of a long propagation time, high bandwidth, and short frame length is disastrous in terms of efficiency.

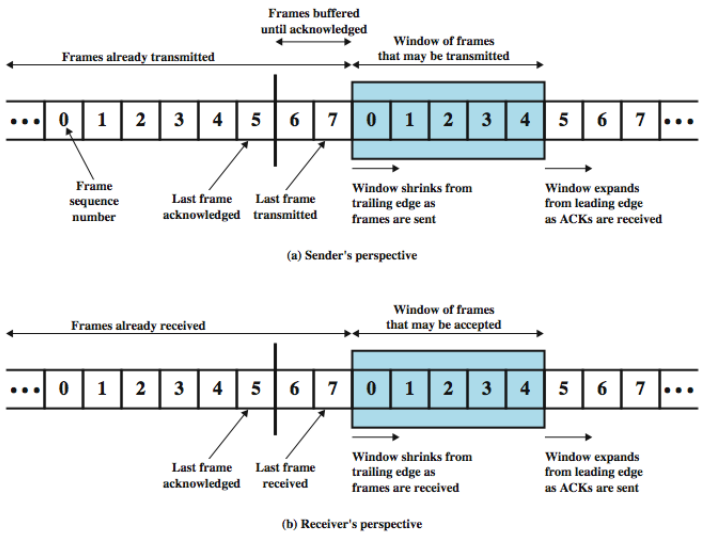
121

Sliding Window Protocols

- The problem?
  - Consequence of the rule that requires a sender to wait for an acknowledgment before sending another frame.
  - Relaxing this condition will enable achieving significantly better throughput.
- Solution!
  - Allowing the sender to transmit up to ***w*** frames before blocking, instead of just 1.
  - The sender will be able to continuously transmit frames since the acknowledgment will arrive for previous frames before the window becomes full.

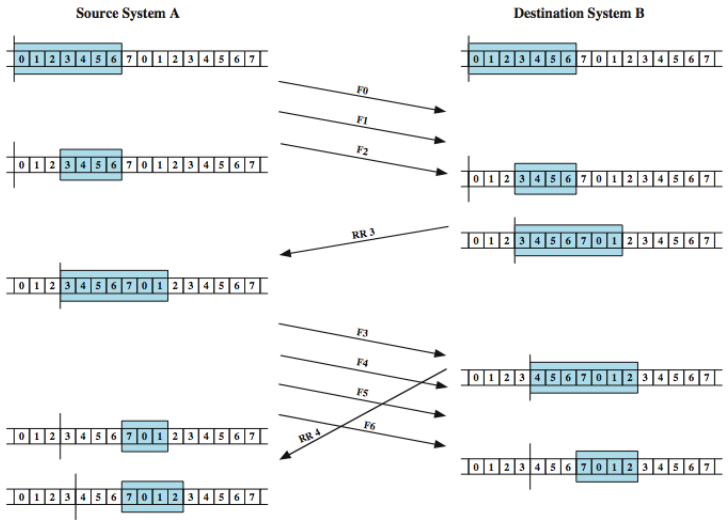
122

Sliding Window Diagram



123

Sliding Window Example



124

Sliding Window Protocols

- We must correctly establish, what is the appropriate size of w ?
  - Number of frames to fit inside the channel.
  - Capacity of the channel is determined by
    - Bandwidth in bits/sec multiplied with One-way transit (propagation) time, or
    - The **bandwidth-delay product** of the link: **BD**
    - divide this quantity by the number of bits in a frame to express it as a number of frames.
  - The size of window should be set to:
    - $w = 2BD+1$

125

Sliding Window Protocols

- Example:
  - Link with the bandwidth of 50 kbps
  - One way transit time of 250 msec
  - Bandwidth delay product is
    BD = 50 kbps x 250 msec = 12.5 kbits or 12.5 frames of 1000 bits.
  - 2BD+1 = 26 frames.
  - t = 0: sends a first frame and subsequent frames after 20 msec.
  - t = 520 msec: acknowledgment for the first frame is received, while 26 frames were transmitted.
  - Thereafter, acknowledgments will arrive every 20 msec.

126

Sliding Window Protocols

- So, the sender will always get permission to continue just when it needs it.
- 25 or 26 unacknowledged frames will always be outstanding.
- Hence, the senders maximum window size is 26 frames.

- For smaller window size, the utilization of the link will be less than 100%.

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

- This value is an Upper bound of the link utilization because:
  - It does not allow for any frame processing time
  - Treats acknowledgment frame as having zero length.

127

Sliding Window Protocols

- From the equation:
  - Large bandwidth-delay product requires a large window.
  - If the delay is high, the sender will rapidly exhaust its window even for a moderate bandwidth.
  - If the bandwidth is high, even for a moderate delay the sender will exhaust its window quickly unless it has a large window (e.g., a 1-Gbps link with 1-msec delay holds 1 megabit).
  - With stop-and-wait for which w = 1, of there is even one frame's worth of propagation delay, the efficiency will be less than 50%.
- This technique of keeping multiple frames in flight is an example of pipelining.
- Pipelining frames over an unreliable communication channel raises some serious issues.

128

Sliding Window Protocols

- What happens if a frame in the middle of a long stream is damaged or lost?
- Large numbers of succeeding frames will arrive at the receiver before the sender even finds out that anything is wrong.
- When a damaged frames arrives at the receiver, it obviously should be discarded, but what should the receiver do with all the corrected frames following it?
- The receiver data link layer is obligated to hand packets to the network layer in order.
- Two basic approaches are available for dealing with errors in the presence of pipelining
  - Go-Back-N
  - Selective Repeat

129