DLL Protocols

Rakesh Matam

Indian Institute of Information Technology Guwahati

rakesh@iiitg.ac.in

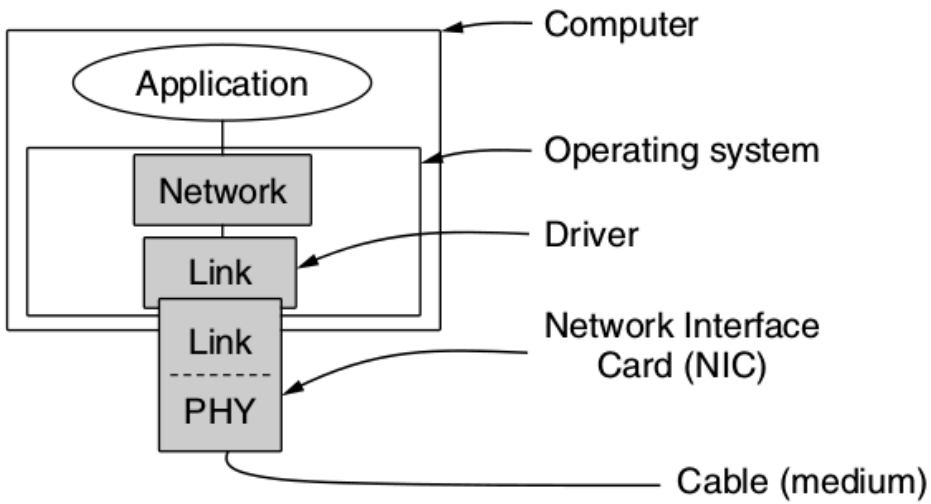
August 25, 2021

Elementary Data Link Layer Protocols

Assumptions:

- We assume that the physical layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth.
- The physical layer process and some of the data link layer process run on dedicate hardware called a NIC (Network Interface Card).
- The rest of the link layer process and the network layer process run on the main CPU as part of the operating system, with the software for the link layer process often taking the form of a **device driver**.

Elementary Data Link Layer Protocols



Elementary Data Link Layer Protocols

Assumptions:

- Machine **A** wants to send a long stream of data to machine **B**, using a reliable, connection-oriented service.
- Later, we will consider the case where **B** also wants to send data to **A** simultaneously.
- We also assume that machines do not crash. That is, these protocols deal with communication errors, but not the problems caused by computers crashing and rebooting.

Preliminaries:

```
#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;
```

Preliminaries:

- A *seq_nr* is a small integer used to number the frames so that we can tell them apart.
- These sequence numbers run from 0 up to and including *MAX_SEQ*, which is defined in each protocol needing it.
- A packet is the unit of information exchanged between the network layer and the data link layer on the same machine, or between network layer peers.
- In our model it always contains *MAX_PKT* bytes, but more realistically it would be of variable length.
- A frame is composed of four fields: kind, seq, ack, and info.
- The first three of which contain control information and the last of which may contain actual data to be transferred. These control fields are collectively called the frame header.

Preliminaries:

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

P1: A 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 ...

P2: A simplex stop-and-wait protocol for an error-free channel

To prevent the sender from flooding the receiver:

- One solution is to build the receiver to be powerful enough to process a continuous stream of back-to-back frames.
- A more general solution 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 wait until the little dummy (i.e., acknowledgement) frame arrives. This delay is a simple example of a **flow control protocol.**

P2: A simplex stop-and-wait protocol for an error-free channel

```
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 */
  }
}

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 */
  }
}
```

P3: Stop-and-Wait protocol for noisy channel

- Frames may be either damaged or lost completely.
- If a frame is damaged in transit, the receiver hardware will detect this while it computes checksum.
- What can we do?
- 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.

P3: Stop-and-Wait protocol for 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, so the data link layer must guarantee that no combination of transmission errors, however unlikely, can cause a duplicate packet to be delivered to a network layer.

P3: Stop-and-Wait protocol for noisy channel

Consider the following scenario:

1. 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*.
2. The acknowledgement frame gets lost completely. It just never arrives at all. Life would be a great deal simpler if the channel mangled and lost only data frames and not control frames, but sad to say, the channel is not very discriminating.
3. 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.
4. 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 (i.e., the copy of the file made by *B* will be incorrect and the error will not have been detected). In other words, the protocol will fail.

P3: Stop-and-Wait protocol for 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.

  What should be the *MAX_SEQ* of the sequence number?

P3: Stop-and-Wait protocol for noisy channel

- The ambiguity in this protocol is only between a frame, **m**, and its direct successor, **m + 1**.
- If frame **m** is lost or damaged, the receiver will not acknowledge it, so the sender will keep trying to send it.
- Once it has been correctly received, the receiver will send an acknowledgement to the sender.
- Depending upon whether the acknowledgement frame gets back to the sender correctly or not, the sender may try to send **m** or **m + 1**.
- A 1-bit sequence number (0 or 1) is therefore sufficient.

P3: Stop-and-Wait protocol for noisy channel

- 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 Acknowledgment with Retransmission).

P3: Stop-and-Wait protocol for noisy channel

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

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

P3: Stop-and-Wait protocol for noisy channel

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

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

Sliding Window Protocols

- Use the same link for data in both directions.
- A **kind** field in the header of an incoming frame is used by the receiver to tell whether the frame is data or an acknowledgement.
- ACK frames can be **piggybacked** on data frames.

Sliding Window Protocols

- Each outbound frame contains a sequence number, ranging from 0 to 2^n-1. This is so that the sequence number fits exactly in an n-bit field.
- The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send.
- These frames are said to fall within the **sending window.**

Sliding Window Protocols

- The receiver also maintains a **receiving window** corresponding to the set of frames it is permitted to accept.
- The sender's window and the receiver's window **need not** have the **same lower and upper limits or even have the same size.**

Sliding Window Protocols: Sender's side

- The sequence numbers within the sender's window represent **frames that have been sent or can be sent but are as yet not 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 an acknowledgement comes in, the lower edge is advanced by one. In this way the window continuously maintains a list of unacknowledged frames.

Sliding Window Protocols

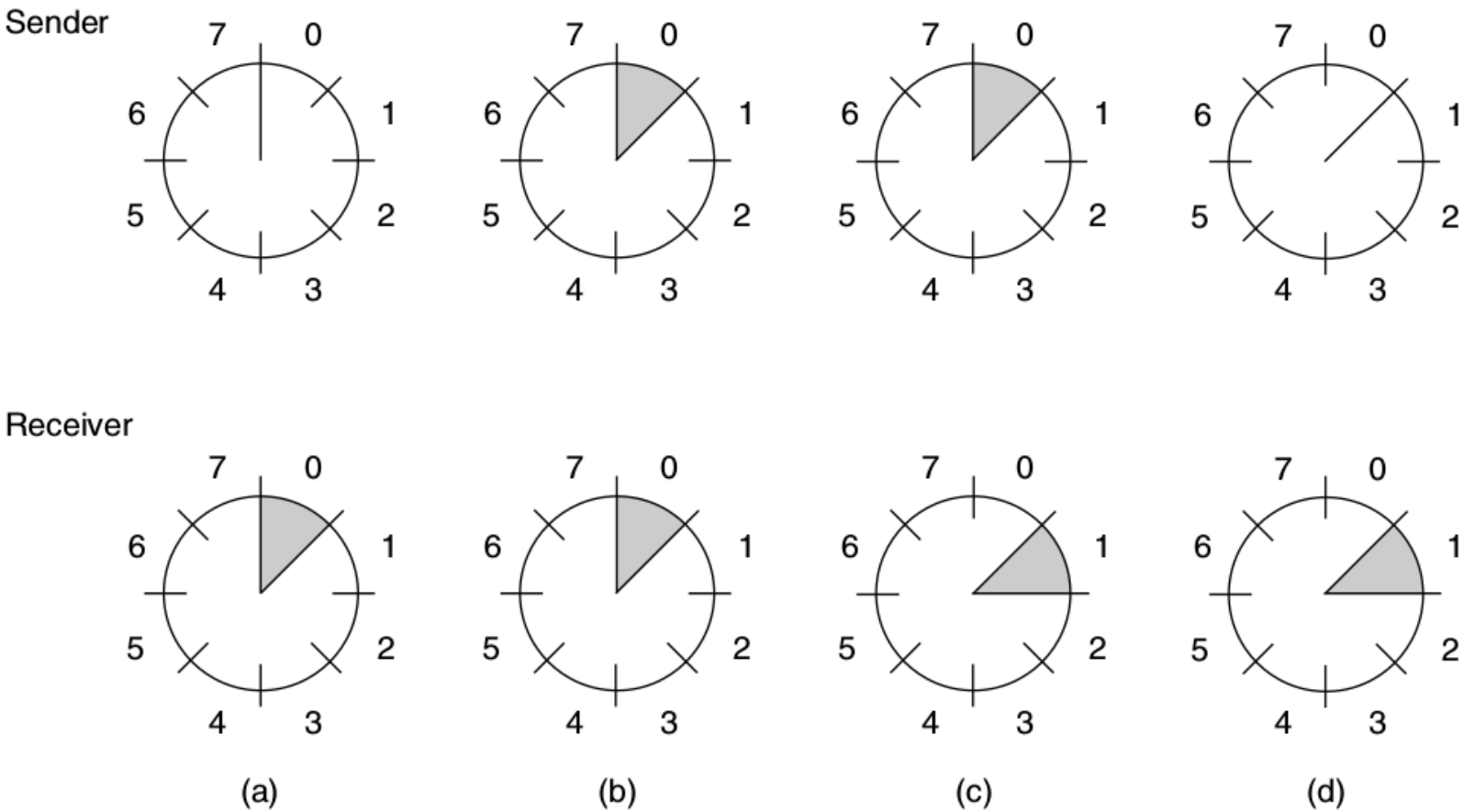


Figure 3-15. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

Sliding Window Protocols: Sender's side

- Since frames currently within the sender's window **may ultimately be lost or damaged** in transit, the sender must keep all of these frames in its memory for possible retransmission.
- Thus, if the maximum window size is **n**, the sender needs **n** buffers to hold the unacknowledged frames.
- If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free.

Sliding Window Protocols: Receiver's side

- The receiving data link layer's window corresponds to the **frames it may accept**. Any frame falling within the window is put in the receiver's buffer.
- When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer and the window is rotated by one.
- Any frame falling outside the window is discarded.
- In all of these cases, a subsequent acknowledgement is generated so that the sender may work out how to proceed.

One bit Sliding Window Protocol

- The sender transmits a frame and waits for its acknowledgement before sending the next one.
- *Next_frame_to_send* tells which frame the sender is trying to send and *frame_expected* tells which frame the receiver is expecting. In both cases 0 and 1 are the only possibilities.
- Under normal circumstances, one of the two data link layers goes first and transmits the first frame.
- When this (or any) frame arrives, the receiving data link layer checks to see if it is a duplicate.

One bit Sliding Window Protocol

- If the frame is the one expected, it is passed to the network layer and the receiver's window is slid up.
- The **acknowledgement field** contains the number of the last frame received without error.

One bit Sliding Window Protocol:Resiliency

- Assume that computer **A** is trying to send its frame 0 to computer **B** and that **B** is trying to send its frame 0 to **A**.

One bit Sliding Window Protocol

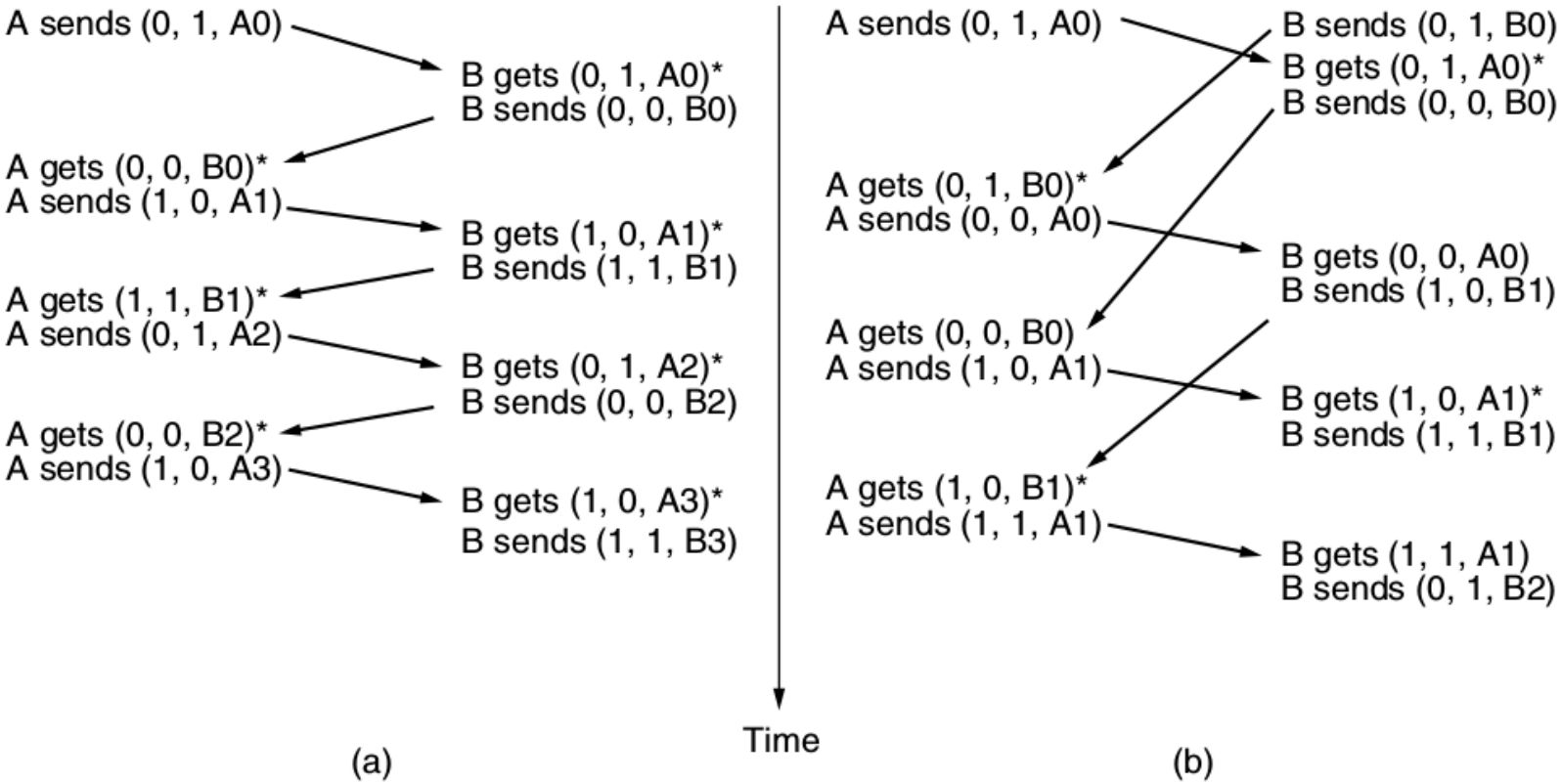


Figure 3-17. Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

One bit Sliding Window Protocol

```
/* 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 */
 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 */
         }
     }
     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 */
 }
}
```

Go-back-n SWP

- Till now the assumption was that the **transmission time** required for a frame to arrive at the receiver plus the transmission time for the acknowledgement to come back is **negligible**. Which is false.
- In these situations the **long round-trip time** can have **important implications** for the efficiency of the bandwidth utilization.
- Using Protocol 4: Evaluate the efficiency of a 50-kbps satellite channel with 500 msec round-trip propagation delay? Consider a 1000bit frame.

Go-back-n SWP

- The problem is because of the rule requiring a **sender to wait for an acknowledgement** before sending another frame.
- The solution lies in allowing the sender to transmit up to **w** frames before blocking, instead of just 1.
- With a large enough choice of **w** the sender will be able to continuously transmit frames since the acknowledgments will arrive for previous frames before the window becomes full, preventing the sender from blocking.

Go-back-n SWP

- To find an appropriate value for **w** we need to know how many frames can fit inside the channel as they propagate from sender to receiver.
- This capacity is determined by the **bandwidth** in bits/sec **multiplied by** the **one-way transit time**, or the **bandwidth-delay product** of the link.
- We can divide this quantity by the number of bits in a frame to express it as a number of frames. Call this quantity BD.

Go-back-n SWP

- For smaller window sizes, the utilization of the link will be less than 100% since the sender will be blocked sometimes.
- The utilization can be represented as the fraction of time that the sender is not blocked:

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

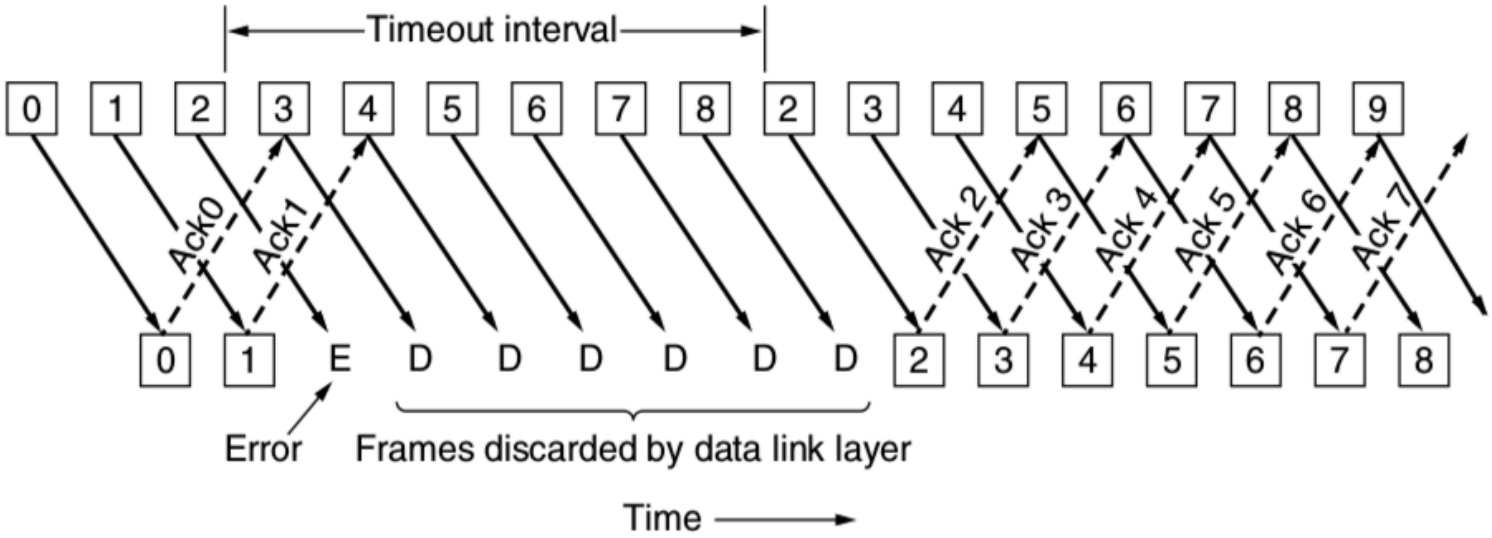
This value is an upper bound because it does not allow for any frame processing time and treats the acknowledgement frame as having zero length, since it is usually short.

Go-back-n SWP

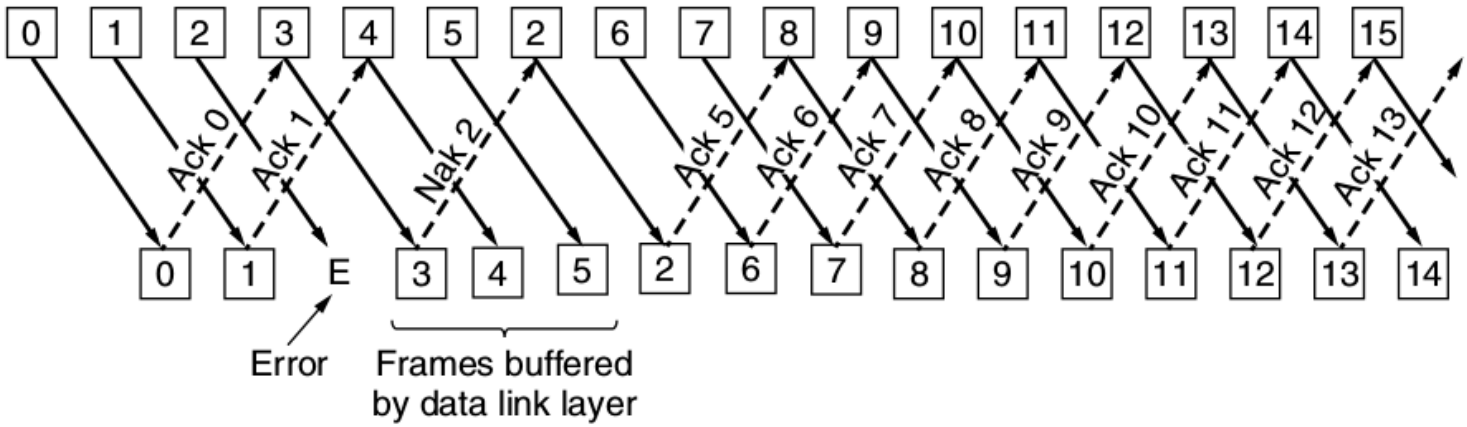
- The technique of keeping multiple frames in flight is called pipelining.
- Pipelining frames over an unreliable communication channel raises some serious issues.
- What happens if a frame in the middle of a long stream is damaged or lost?

Go-back-n

Two basic approaches are available for dealing with errors in the presence of pipelining.



Go-back-n: Selective Repeat

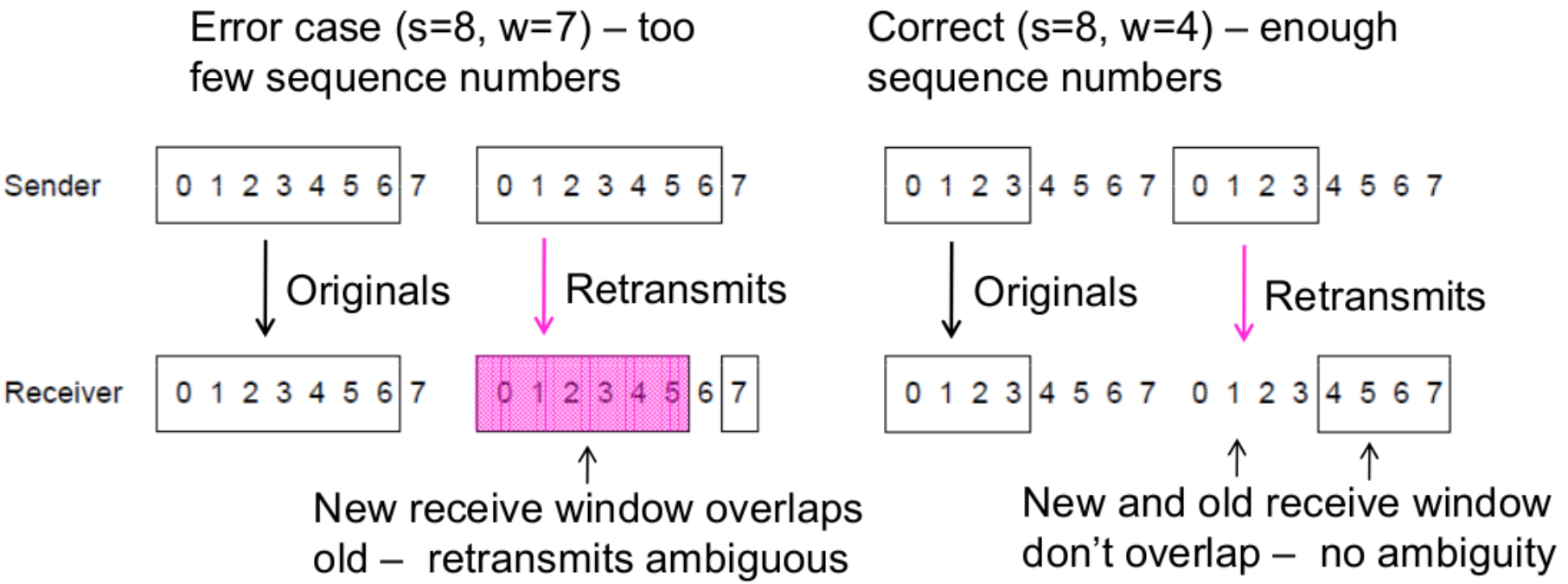


- Receiver accepts frames anywhere in receive window
- Cumulative ack indicates highest in-order frame
- NAK (negative ack) causes sender retransmission of a missing frame before a timeout resends window.

Go-back-n: Selective Repeat

For correctness, we require:

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



Example Data Link Layer Protocols

- Within a single building, LANs are widely used for interconnection, but most **wide-area network infrastructure** is built up from **point-to-point** lines.
- **Our focus:** Data link protocols found on point-to-point lines in the Internet in two common situations.
  - 1. Packets sent over **SONET** optical fiber links in wide-area networks.
  - These links are widely used, for example, to connect routers in the different locations of an ISP's network.
  - 2. **ADSL** links running on the local loop of the telephone network at the edge of the Internet.
  - These links connect millions of individuals and businesses to the Internet.

Example Data Link Layer Protocols

- The Internet needs **point-to-point** links for SONET and ADSL use cases.
- A standard protocol called PPP **(Point-to-Point Protocol)** is used to send packets over these links.
- SONET and ADSL links both apply PPP, but in different ways.

Packet over SONET (1)

- SONET,is the physical layer protocol that is most commonly used over the wide-area optical fiber links that make up the backbone of communications networks.
- It provides a bitstream that runs at a well-defined rate, for example 2.4 Gbps for an OC-48 link.
- This bitstream is organized as **fixed-size byte payloads** that recur every **125 μsec**, whether or not there is user data to send.
- To carry packets across these links, some framing mechanism is needed **to distinguish occasional packets** from the **continuous bitstream** in which they are transported.
- **PPP runs on IP** routers to provide this mechanism.

Packet over SONET(2)

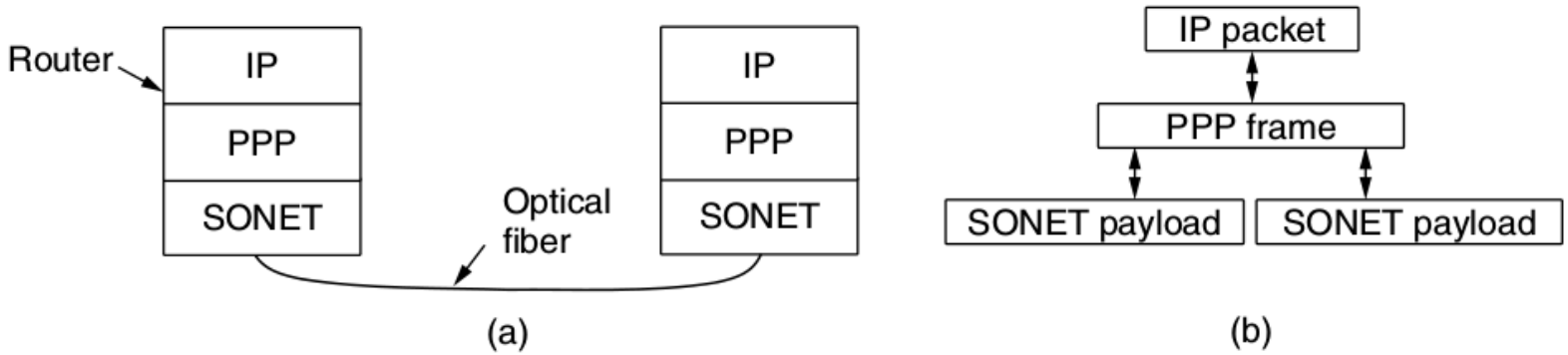


Figure 3-23. Packet over SONET. (a) A protocol stack. (b) Frame relationships.

Packet over SONET (3)

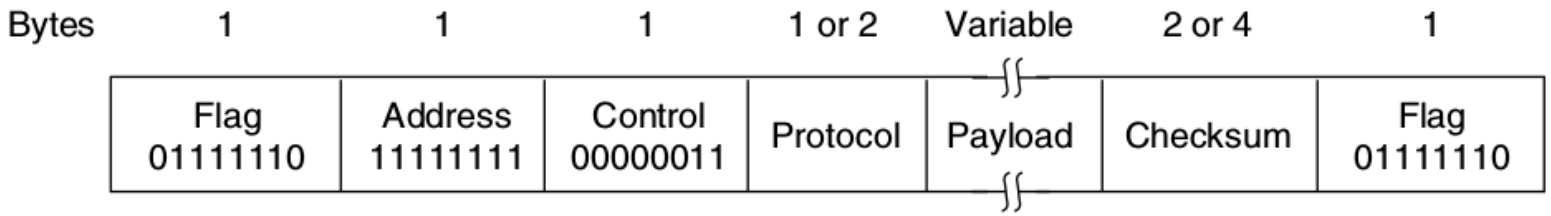
PPP improves on an earlier, simpler protocol called SLIP (Serial Line Internet Protocol). PPP provides three main features:

1. A framing method that unambiguously delineates the end of one frame and the start of the next one. **The frame format also handles error detection.**
2. A link control protocol for bringing lines up, testing them, negotiating options, and bringing them down again gracefully when they are no longer needed. This protocol is called LCP (Link Control Protocol).
3. A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used. The method chosen is to have a different **NCP (Network Control Protocol)** for each network layer supported.

PPP (1)

1. PPP is byte oriented.
2. PPP uses **byte stuffing** and all frames are an integral number of bytes.
3. PPP provides connectionless unacknowledged service. It is done using "unnumbered mode" which is nearly always used in the Internet.

Packet Format:PPP (2)



1. All PPP frames begin with the standard flag byte of 0x7E (01111110).
2. The flag byte is stuffed if it occurs within the **Payload** field using the escape byte 0x7D.
3. After the start-of-frame flag byte comes the **Address field**. This field is always set to the binary value 11111111 to indicate that all stations are to accept the frame.

Packet Format:PPP (2)

1. The *Address field* is followed by the **Control field**, the default value of which is 00000011. This value indicates an **unnumbered frame**.
2. Since the **Address** and **Control** fields are always constant in the default configuration, LCP provides the necessary mechanism for the two parties to negotiate an option to omit them altogether and save 2 bytes per frame.
3. The fourth PPP field is the **Protocol** field. Its job is to tell what kind of packet is in the **Payload** field.

Packet Format:PPP (3)

1. The **Payload** field is variable length, up to some negotiated maximum. If the length is not negotiated using LCP during line setup, a default length of 1500 bytes is used. Padding may follow the payload if it is needed.
2. After the Payload field comes the **Checksum** field, which is normally 2 bytes, but a 4-byte checksum can be negotiated.
3. PPP is a framing mechanism that can carry the packets of multiple protocols over many types of physical layers.

LCP:PPP (4)

1. Before PPP frames can be carried over SONET lines, the PPP link must be established and configured.

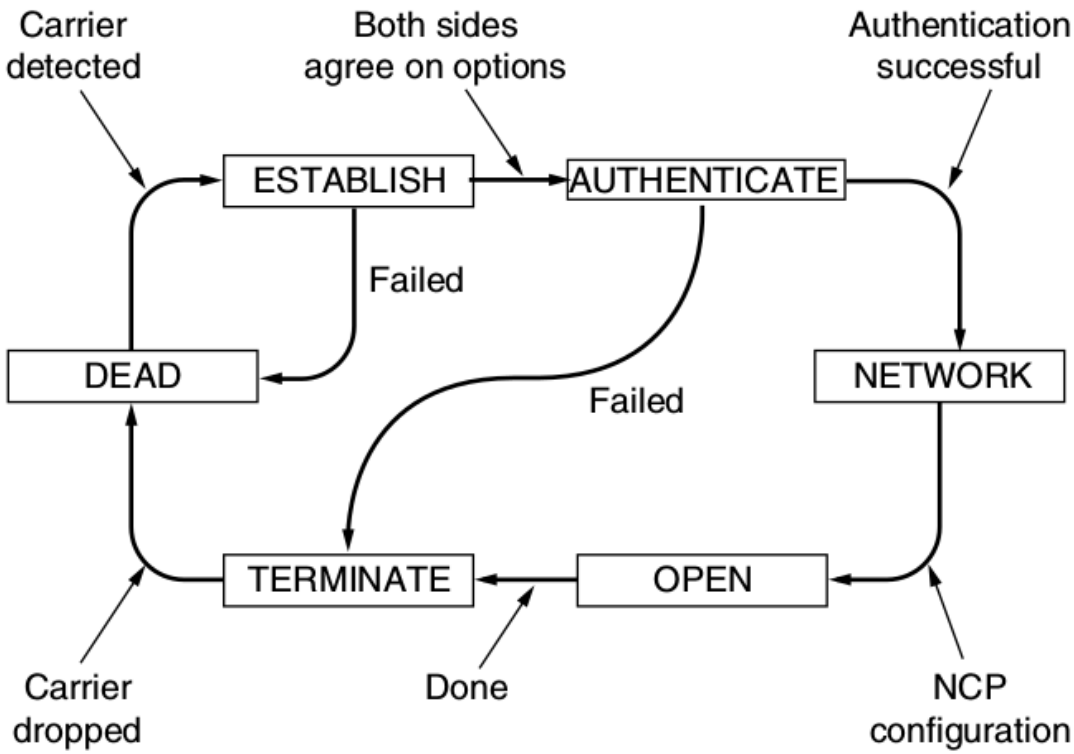
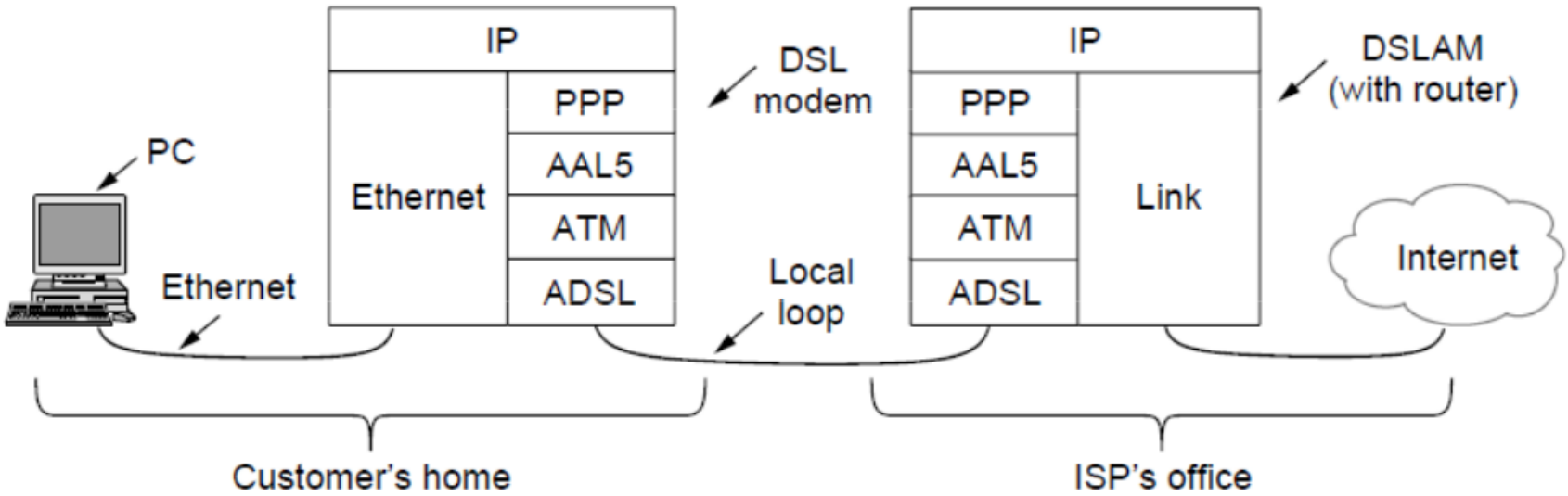


Figure: The phases that the link goes through when it is brought up, used, and taken down again

ADSL (1)

Widely used for broadband Internet over local loops

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



The End