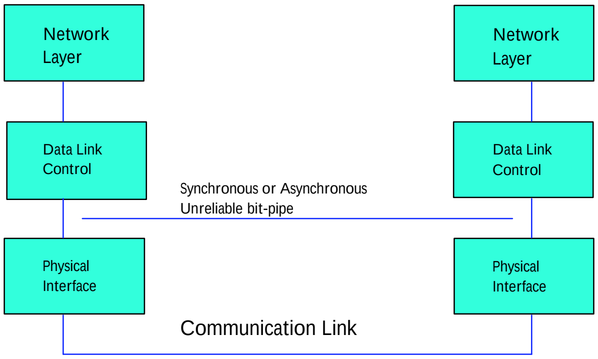
DLC Layer

Dr. G. A. Marin

The Virtual bit-pipe



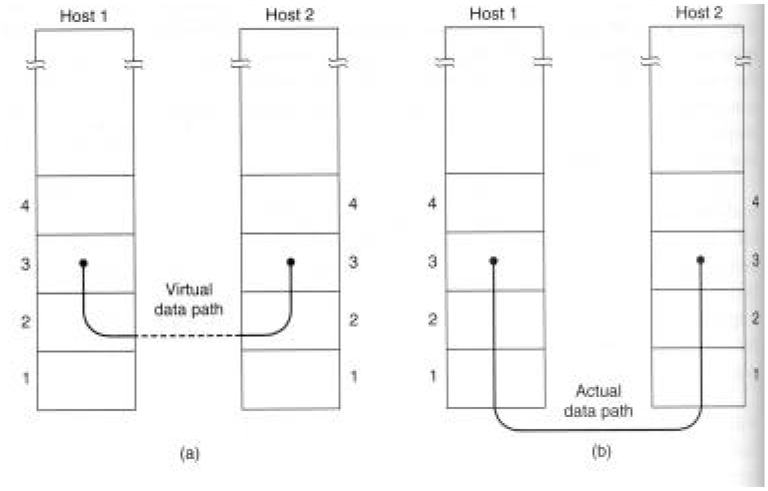
Common DLC Services: deliver data frames with

- Unacknowledged connectionless service
  - No Ack from destination
  - No connection established or released
  - No error recovery at DLC layer
  - Common in LANs and voice
- Acknowledged connectionless service
  - No connection established or released
  - Each frame is ACKed
  - Common in wireless
- Acknowledged connection-oriented service
  - Source & destination establish connection before data sent
  - DLC guarantees frame received once and in order (numbered)
  - Phases: Connection setup, transmission, connection release

DLC Service "Hides" Variety of Physical Media and Subnet Architectures

- Local Area Network (Ethernet, Token Ring)
- Metropolitan Area Networks (SMDS)
- Point-to-point connection
- Frame Relay
- ATM
- Wireless

DLC Service and Virtual Data Path



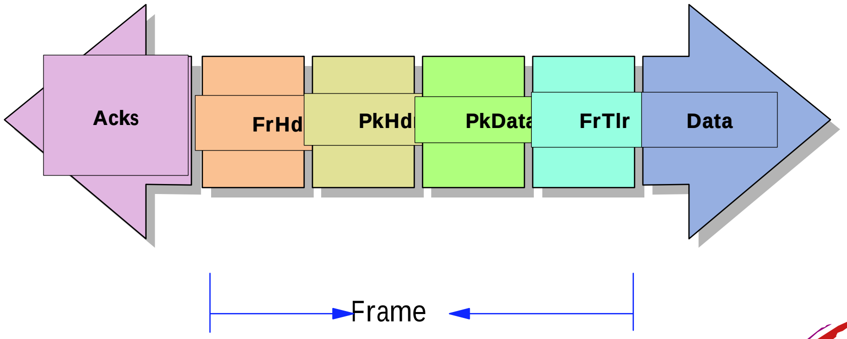
To do its job a DLC must:

- Create/Detect Frames
  - Add Frame Header and Trailer
- Insure Frame is Error-Free (with agreed probability)
- Cooperate with his peer through a protocol that supports the DLC service

Building/Finding Frames

- Phy layer delivers raw bits
    - received bits may be less, more, equal to sent bents
    - received bits may be errored
- Data link layer must find frames, correct errors
- Generally cannot rely on gaps between frames
- Use header and trailer to carry start/end flags and "checksums"
    - Ex. STXDLE starts frame
    - ETXDLE ends frame
    - or specific bit pattern: 01111110

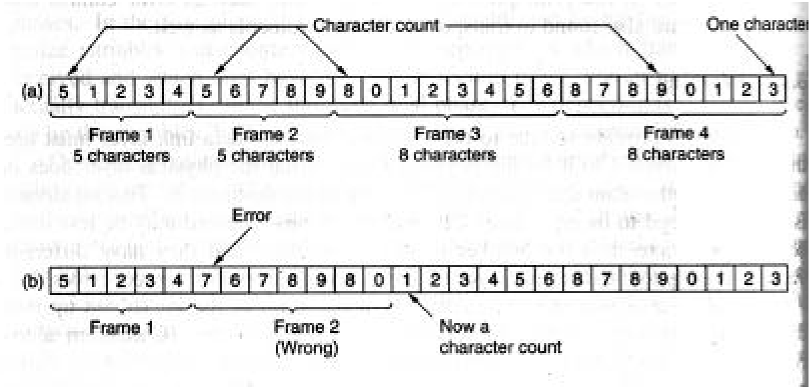
DLC Frame Carries NetLayer Pkt



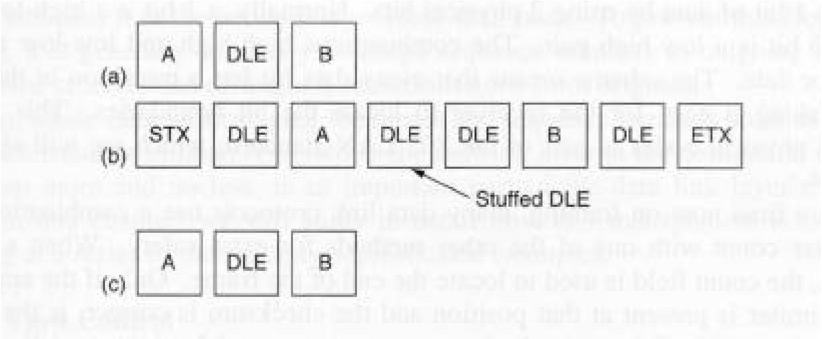
Key Methods for Recognizing/Separating Frames

- Character counting
- Starting/Ending characters with character stuffing
- Starting/Ending flags with bit stuffing

Character Counting



Characters sent, stuffed, received...



(a) A DLE B

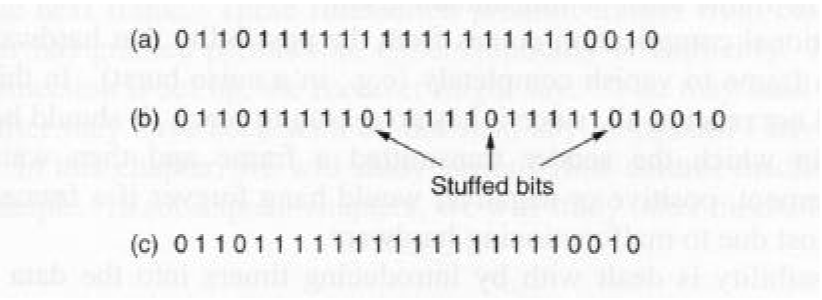
(b) STX DLE A DLE DLE B DLE ETX

Stuffed DLE

(c) A DLE B

Bits sent, stuffed, received...

Flag: 01111110



(a) 011011111111111111110010

(b) 011011111011111011111010010

Stuffed bits

(c) 011011111111111111110010

Error Detection (and Correction)

- Two basic philosophies
  - Error correction: Include enough redundant bits to enable error correction
  - Error detection: Include enough redundant bits to enable error detection only then request retransmission of frame
- Think of frame as m bits, redundant bits as r, with total m+r=n, the codeword.
  - r bits carried in frame trailer

Hamming Distance

- Hamming distance between two codewords is number of bits in which they differ:
  - 10001001 and 10110001 have Hamming distance of 3 (bits 3, 4, 5)
- Note 2^m messages possible and many more, 2^n, code words.
  - Hamming distance of "closest" two codewords is the Hamming distance of the code.
  - To DETECT d errors need a "d+1 code"
  - To CORRECT d errors need a "2d+1 code"
- Error detection and retransmission much more common

Example: 2 m-bits and 2 r-bits

- 00, 01, 10, 11 are allowable messages
- Suppose 0000, 0101, 1010, and 1111 are the code words.
  - Hamming distance is 2
  - Can detect 1 bit errors
    - If 1000 is received we know it is an error
  - Cannot correct 1 bit errors (need HD of 3)
    - If 1000 received we don't know if 1010 or 0000 was sent with 1-bit error.

Example: 2 m-bits & 3 r-bits

- 00, 01, 10, 11 are allowable messages (data)
- Codewords: 00000,00111,11010,11101
- Hamming Distance is 3.
- Can correct 1-bit errors
- Ex. If 11110 is received, the only codeword that is 1bit away is 11010.
- HW: Show you cannot correct all 2-bit errors

Common Error Detection Mechanisms

- Parity Bit
  - Add 1 bit per frame so that frame is always even or always odd
  - Will catch all single-bit errors
  - Probability of catching a burst error is 0.5
- Matrixed Parity Bits
  - Send blocks of frames as matrix n bits wide and k bits high.
  - Compute separate parity bit for each column and add as k+1st row.
  - Transmit row 1, row 2, ... row k+1 and accept only if all parity bits are correct.
  - Will detect single burst of length n and longer bursts with probability 1- 2^{-n}.
  - Cyclic Redundancy Code

Cyclic Redundancy Check

- m-bit message frame: i_{m-1} i_{m-2} ... i_1 i_0
- represented as: $i_{m-1}x^{m-1}+i_{m-2}x^{m-2}+...+i_1x+i_0$
  - Note: coefficients i_0, i_1, ..., i_{k-1} are 0 or 1.
- Example: 110001
  - represented: x^5+x^4+1
- Polynomial arithmetic is done mod 2 with addition and subtraction identical to "exclusive or"
  - 0+0=0
  - 0+1=1
  - 1+0=1
  - 1+1=0
  - no carries for addition or borrows for subtraction

CRC Fundamentals

- Transmit codewords of length m+r at the DLC layer with m message bits and r checksum bits.
- Message bits map to polynomial M(x) of order m-1.
- Checksum bits map to polynomial C(x) of order r-1.
- T(x) is polynomial that corresponds to the entire transmitted codeword of length m+r (degree m+r-1)
- Checksum is computed before transmission such that the polynomial T(x) is divisible by a generator polynomial, G.
  - $G(x) = x^r+g_{r-1}x^{r-1}+\ldots+g_1x+1$ and $r<m$
- The receiver also knows G and determines that an error has occurred if T(x)/G(x) is anything other than zero.

How to compute checksum (given G)

- Append r zero bits to low-order end of frame so that it now contains m+r bits corresponding to $x^rM(x)$.
- Divide: $x^rM(x)/G(x)$ using mod 2 arithmetic.
- Subtract the remainder from $x^rM(x)$.
  - G has degree r so remainder has degree less than r; thus, no bits of the original m are changed by this subtraction.
- The result of the subtraction is the checksummed frame to be transmitted (and corresponds to the polynomial T).

CRC Example

Frame: 1101011011 Generator: 10011

Message and appended 4 zeros: 11010110110000

```
              1100001010
10011 | 11010110110000
         10011
           10011
           10011
                010110
                  10011
                      10100
                      10011        Remainder
                          1110
```

Error Detecting Power of CRC

- Assume received polynomial is T(x)+E(x)
  - Each 1-bit coefficient in E(x) corresponds to errored bit of T.
- Receiver thus divides T(x)+E(x)/G(x)=E(x)/G(x) by design.
  - If G(x) is a factor of the error polynomial, the error will be undetected.
- Single bit error implies E(x)=x^i for some bit position i=1,2,...,n
  - As long as G(x) contains two or more terms, it cannot divide x^i so all single bit errors are detected.
- Two single-bit errors (double bit error) imply E(x)=x^i+x^j, i>j (say)
  - Write E(x)=$x^j(x^{i-j}+1)$. If we make sure G cannot divide x^k or x^k+1 for all k=1,2,...,n then G will catch all double bit errors.
  - Example, $x^{15}+x^{14}+1$ will not divide x^k+1 for k<32,768
- Odd number of erors means E(x) has odd number of terms.
  - It is known that no polynomial with oddd number of terms is divisible by x+1 (mod 2).
  - Thus, make sure G(x) has x+1 as factor and catch all errors involving odd number of bits.
- Most important CRC with r check bits will detect all burst errors of length less than or equal to r (read discussion in text).

International Standards for CRC Generator Polynomials

- CRC-12: $x^{12}+x^{11}+x^{3}+x^{2}+x+1$
- CRC-16: $x^{16}+x^{15}+x^{2}+1$
- CRC-CCITT: $x^{16}+x^{12}+x^{5}+1$

Retransmission Strategies

- General concept of Automatic Repeat Request (ARQ): detect frames with errors at receiving DLC and request transmitting DLC to repeat erroneous frames.
- Error detection usually done via CRC. Retransmissions handled via retransmission protocol.
- Correctness Issue: Does the protocol succeed in releasing each packet once (and only once), without errors, from the receiving DLC?
- Efficiency Issue: How much of the capacity of the channel is wasted by unnecessary waiting or by sending unnecessary transmissions?

Example: Stop-and-Wait ARQ with ACK/NAK

- A transmits a frame to B and waits.
- If frame ok at B, B sends ACK.
- If frame in error at B, B sends NAK.
- ACK and/or NAK also protected with CRC.
- If A receives ACK, sends next frame.
- If A receives NAK, sends last frame again.
- Because the frame or the ACK/NAK can be lost, A maintains a timer and retransmits if "pops."

Potential Problems

- The delay in the channel is arbitrary so A may time out and retransmit the old data in a second frame. (B will not know and will pass duplicate data to NL.)
- Potential solution: put a sequence number on the frames.
- Remaining problem: A sends frame k, B receives and sends ack. Ack is delayed. A sends frame k again. B receives (tosses) and sends ack. A receives first ack and sends k+1. A receives second ack and sends k+2 although k+1 has not really been acked by B.
- Solution: Instead of sending ACK/NAK, B sends a frame (still called ACK) that contains the number of the next frame awaited.

Algorithm at A (for A to B Tx)

- 1. Set the integer variable SN to 0.
- 2. Wait for packet from NL; when available assign number SN to new packet.
- 3. Transmit SNth packet in frame containing SN in the seq number field.
- 4. If an error-free frame is received from B containing a request number RN greater than SN, increase SN to RN and go to step 2. If no such frame is received within a given (timer) delay, go to step 3.

Algorithm at B (for A to B Tx)

- 1. Set the integer variable RN to 0 and repeat steps 2 and 3 forever.
- 2. When an error-free frame is received from A containing a sequence number SN equal to RN, release the received packet to the higher layer and increment RN.
- 3. At arbitrary times, but within bounded delay after receiving any error-free data frame from A, transmit a frame to A containing RN in the request number field.

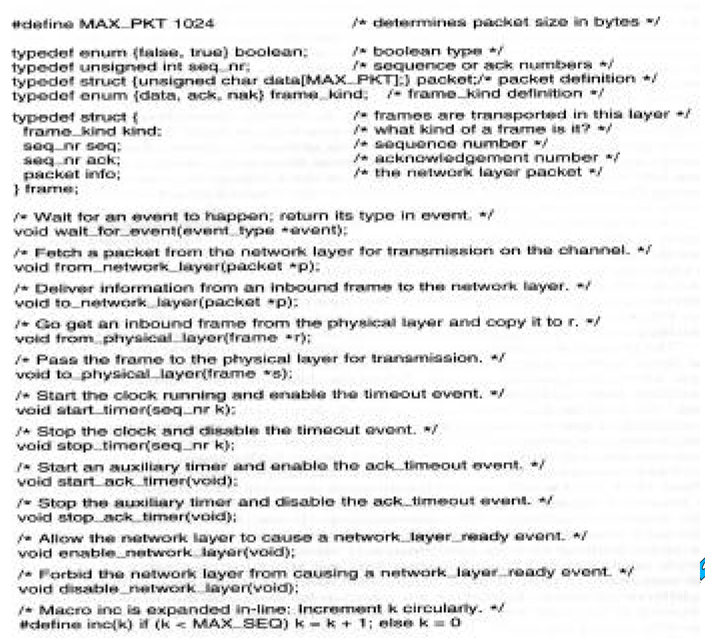
Proof of Correctness

- Can be proven that algorithm is "correct":
  - That is, a never-ending stream of packets can be accepted from the higher layer at A and delivered to the higher layer at B ir order and without repetitions or deletions.
- Correctness proofs generally involve two parts: safety and liveness.
  - safety: algorithm never produces an incorrect result (releases bad packet to NL).
  - liveness: algorithm can continue forever to produce results (never enters a deadlock).

Data Link Protocol Assumptions

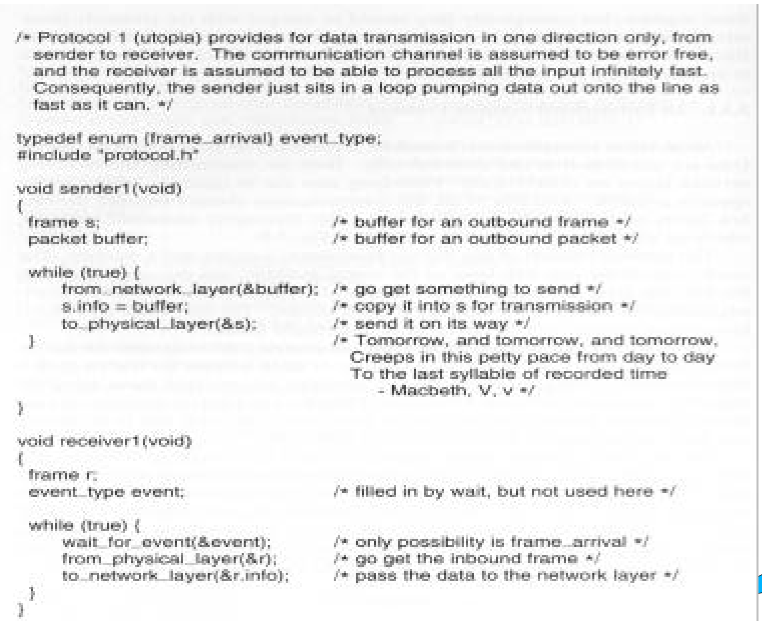
- Independence of layers
- A sends to B using reliable, connection-oriented service
  - Infinite supply of data ready to send
- DLC accepts packet from Net Layer adds FH and FT and submits to to_physical_layer. Receiver receives from from_physical_layer and submits pkt to Net Layer.
- CRC performed on both ends using hardware.
  - event = cksum_err or event = fr_arrival
- Usually channel assumed unreliable
  - timeout and retransmission may result

Definitions in Header File



```
#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 a frame is it? */
  seq_nr seq;                                 /* sequence number */
  seq_nr ack;                                 /* acknowledgement number */
  packet info;                                /* the network layer packet */
} frame;

/* 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);

/* 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
```

An Unrestricted Simplex Protocol



```
/* 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"

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 */
}

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 */
 }
}
```

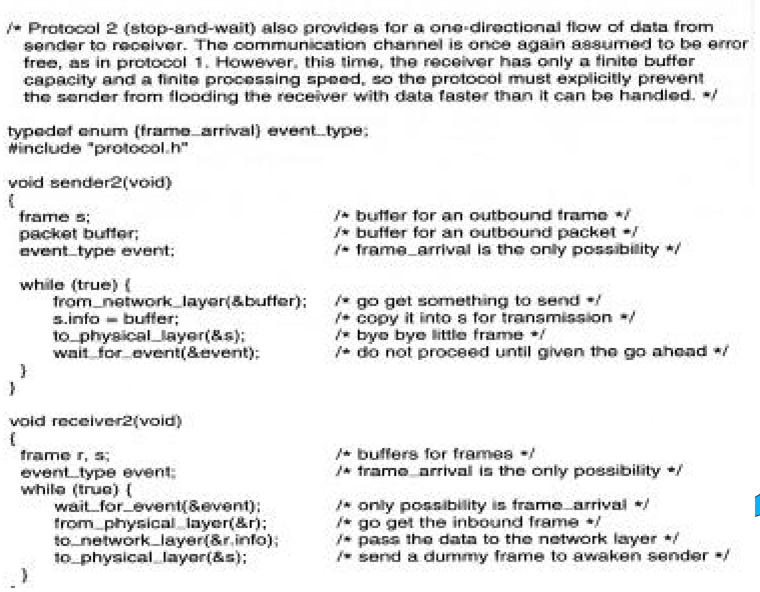
Comments

- Both Tx and Rx net layers are always ready.
- Processing time ignored.
- Infinite buffer space available.
- No damaged or lost frames.
  - No seq numbers or acks.
  - Only event = frame_arrival is possible.
  - Only info field of frame used.

Stop-and-Wait Comments

- Main problem is to prevent flooding the receiver faster than it can process.
- Rx process time may vary because of multiple lines, etc.
  - Inefficient just to assume worst case
- Solution is to wait for Rx to send ACK giving permission for Tx to send next frame.
- When ACK each frame: "Stop-and-Wait"
- Channel must support 2-way communication
  - Half-duplex suffient

Simplex Stop-and-wait protocol
Drop assumption that receiving NL is infinitely fast.

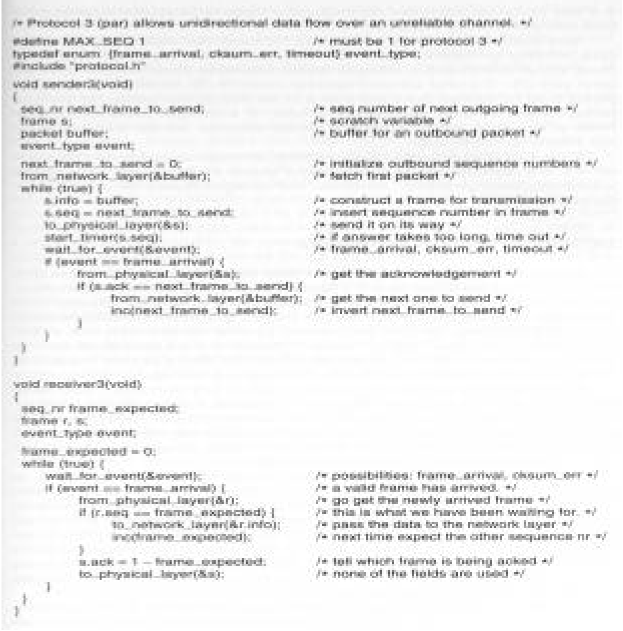


```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so 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 */
  }
}

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 */
  }
```

Add a Noisy Channel

- Channel creates errors so frames damaged or lost.
- Damaged frames are caught by checksum and discarded.
- Why not retransmit after timeout?
  - Might lose ACK! (would appear to be timeout)
- Solution is add sequence number to data frame and seq number to the ACK.
  - Sender remembers seq no of next frame to send
  - Receiver remembers seq no of next frame expected.
- Such protocols (wait for ACK before sending next seq number) most commonly called ARQ.

One-direction data flow over unreliable channel



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

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);             /* none of the fields are used */
     }
 }
}
```

ARQ Comments (ACK Each Frame)

- After Xmit frame sender starts timer at full value.
  - greater than transmission time + worst case processing time + ack transmission time
- Retransmit frame if timer "pops"
- After sender transmits frame it waits for:
  - ack frame to arrive undamaged: fetch next pkt from NL and overwrite buffer then advance seqno
  - If damaged ack or timer pop then retransmit from unchanged buffer with unchaged seqno.
- After Rx receives a frame: if valid pass to NL and return ACK with seqno of next frame. If dup or damaged, then toss it.

Piggybacking

- Return ack with data going back to source
- What if no data soon enough? (set timer)
- Much better use of channel and fewer frames to process

Sliding Window Protocols

- Each outbound frame contains a seq no between 0 and 2^n - 1 for some n.
- Sender maintains a set of seq nos it is permitted to send in "sending window."
  - Nos represent frames sent but not acked.
  - Advance top of window for new sent frame.
  - Advance bottom of window for ACK.
- Receiver maintains a "receiving window.
- Nos represent frames it may accept.
  - Discards any frame outside window & no ack.
  - When frame received at lower edge, passed to NL and acked.
  - When frame received in window but not a lower edge, it will be saved until at lower edge, then passed to NL.

Sliding Window Operation

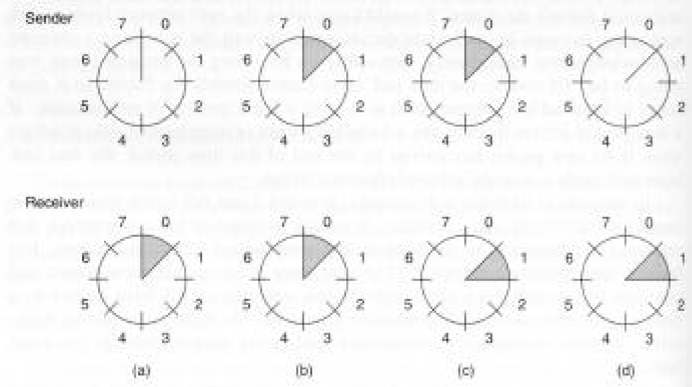


Fig. 3-12. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.