1

Computer Networks

B.Tech, Computer/Software Engineering

Delhi Technological University

Module 2_2: Data Link Layer Flow Control

Instructor: Divyashikha Sethia

Assistant Professor, Department of Software Engineering

2

Assumptions

- Physical, data link & Network layers are independent processes on a system
- M/C A sends long stream of data to M/C B using reliable connection oriented service
- M/Cs do not crash on power out

3

NA-N/W LAYER AT A NB-N/W LAYER AT B

DA-DATA LIN K LAYER AT A DB-DATA LINK LAYER AT B

- NA gives packets to DA, which puts them in frame. Calls to physical layer to send frame. Hardware computes checksum, appends & sends.
- DB waits_ for_ event, procedure generates interrupt to indicate frame arrival. DB does not set a loop waiting for frame arrival.
- DB H/W computes checksum. If OK uses from_physical_layer to acquire frame, strip down packet & passes to NB.N/W layer independent of framing details.
- Sequence no. from 0 to MAX_SEQ & back to 0.
- Packet has fixed length = MAX_PACKET.
- Header = (frame kind, seq. no., Ack. no)
- Frame kind as some frames contains only control info.
- Error recovery –ACKs & timeouts.
- Start_timer resets on-going timer
- No reply-timeout.

4

UNRESTRICTED SIMPLEX PROTOCOL

- Simplex-one way.
- NA & NB always ready.
- Common channel assumed to be error free & receiver able to process all data input infinitely quickly.
- Sender sits in a loop pumping data on the line out as fast as possible.

SIMPLEX STOP AND WAIT PROTOCOL

- NB can't process incoming data infinitely quickly
- Simplex
- NO errors
- Flow control stop fast sender A swamping slow receiver B.
- B provides feedback as dummy frame to A
- A waits for this dummy ACK before sending next packet.
- Half duplex in nature-allocation in A & B sending info over the channel.

SIMPLEX WITH ERROR RECOVERY

➢ PROBLEM:
- If error detected via checksum

➢ SOLUTION:
- Add timers. Receiver sends ACK if no error.
- SENDER: If no acknowledge received re-send

SIMPLEX WITH ERROR RECOVERY

➢ PROBLEM:
- ACK is lost, sender resends, Duplicates frame

➢ SOLUTION:
- Seq.number
- Ambiguity b/w frame m and m+1.
- Sender sends frame m
- If ACK then send m+1
- Else send m again
- One bit seq.no. is sufficient.
- Flip between 0 and 1 to express frame no. expected.
- This ACK is known as:
  - PAR (positive acknowledgement with retransmission) or
  - ARQ (Automatic repeat request)

AT RECEIVER:

If seq = = expected = 0 {
    Expected =1
    ACK= 0
}
Else {
    Expect = = 0 (unchanged)
    ACK=1 (RE-ACK PREVOIUS ONE)
}

AT SENDER:

If (event = = timeout) {
Send current frame again
}
else If (ACK= = next_frame_to_send) {
Send next frame
} Else (wrong or damaged ACK) {
Loop around send current frame again
}
Receiver ignores duplicates and damaged frames.

SLIDING WINDOW PROTOCOLS

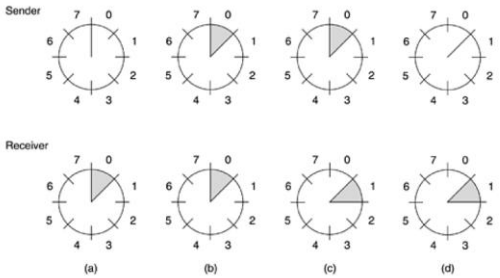
- Previous protocols single direction only.
- Using 2 channel one for frame and other for ACK is a waste of bandwidth
- Use same channel for 2 way communication. Data and control packet like ACK are differentiated by the kind of field in the header of frame

ANOTHER IMPROVEMENT

- When a data frame arrives instead of immediately sending an ACK, wait until N/W layer passes next packet. The ACK is attached to the on-going data frame. This technique of temporarily delaying outgoing ACK so that it can be hooked at outgoing data is called *PIGGYBACKING*.
- ADVANTAGES: Better use of resources (bandwidth).
- DISADVANTAGES: Added overhead of determining duration of expecting ACK.
- SOLUTION: Receiver must wait for a short duration for data from N/W layer else transmit frame with ACK.
- Three protocols are bidirectional protocol called sliding window protocol

Sliding Window Protocol

- Each frame has a seq.no.0 to power (2, n-1) (n bit field).
- Sender maintains a set of seq.no. corresponding to frames it is permitted to send. These falls within ***sending window.***
- These frames are sent but not acknowledged and frames yet to be sent.
- When packet comes from N/W layer, it is given highest no. and upper edge of window advanced by 1.
- When ACK comes lower edge is advanced by 1.
- Receiver has ***receiving windows*** - frames permitted to accept.



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 Protocol

- Complex protocol has freedom about orders in which frames are sent and received.
- Window size n=sender may have n unacknowledged frames.
- Need n buffers to hold then for possible retransmit.
- If window grows to maximum size, can't send further packet=>DA shuts NA.

Sender Window

- Sender window might grow
- As it receives frames to send and has ones unacknowledged.
- Starts with nothing, then NA gives it frames.
- Later window may shrink as frames are acknowledged and NA sends no more.

Receiver window constant size

- Size 1 means only accept in order.
- Size n will accept out of order (e.g. receives later frame, after earlier frames lost).
- Must buffer them and reorder them before sending to NB.
- Example 1: DB has buffers to receive frames 0-7, receives 1-7 frames out of order.
- Waiting for 0 = > can't send to NB yet. 0 gets lost then resent.
- When 0 received frames 0-7 sent to NB.

1-bit sliding window

- Window size 1.stop and wait.
- Must get ACK before sending next frame
- Both A & B are sending & receiving combined in a single loop.


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


- If both start simultaneously then there is problem.
- If one starts first only one in code (PROTOCOL 4) outside while loop.
- ACKs responding to inbound stream are piggybacked in an outbound stream.

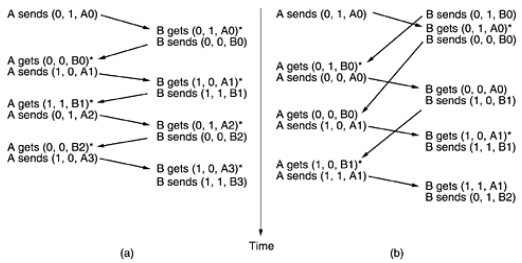


Figure 3 Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

- ➔Time is wasted🡐
- Provided frame can get through, no infinite loop & no duplicate packet at N/W layer process will complete.
- ➔Half the frames contain duplicates, protocol completes & retransmits.

MULTIPLE UN-ACKed FRAMES

•

Sometimes transmission time +ACK time is long => can't have stop n wait,
Must have multiple un- ACKed frames at any time.

As an example,
- 50-kbps satellite channel with a 500-msec round-trip propagation delay.
- Use protocol 4 to send 1000-bit frames via the satellite.
- At t = 0 the sender starts sending the first frame.
- At t = 20 msec the frame has been completely sent.
- Time frame fully arrives at receiver t =270 msec
- Time acknowledgement arrives back at the sender t = 520 under the best of circumstances (no waiting in the receiver and a short acknowledgement frame).
- This means that the sender was blocked during 500/520 or 96 percent of the time.
- In other words, only 4 percent of the available bandwidth was used.
- Clearly, the combination of a long transit time, high bandwidth, and short frame length is disastrous in terms of efficiency.

MULTIPLE UN-ACKed FRAMES...

➔Solution🡐
-Sender can transmit up to w frames without ACKs

- Chooses to fill the time up until first ACK arrives e.g.In this example w=26

- By the time it finishes sending 26 frames=20*26=520 msec first ACK comes back.

- Can send one more frame.

- Second ACK comes and so on…

-Concept based on ***pipelining*** : fill the channel with data to maximise line utilisation.

- Cost: more buffers for un-ack-ed frames

MULTIPLE UN-ACKed FRAMES...

NOTATION:
Need large window if:
i) Bandwidth b bits/sec is large (takes little time to send one frame or small window of frames waits rest of time).
OR
ii) Round trip delay R sec is large (need to fill time)
If b is large, exhaust small window even if R is moderate.
=>need large window if bR is large
Frame size=L bits.
Time to transmit 1 frame=L/b sec
Stop and wait:
Busy time =L/b sec. Idle time=R.
If L/b < R then busy less than ½ the time i.e. L<bR
=>need pipelining if bR is large
Fraction of busy time

$$\frac{L/b}{L/b+R} = \frac{L}{L+bR}$$

25

Handling Errors

"Go back n" and "Selective repeat"

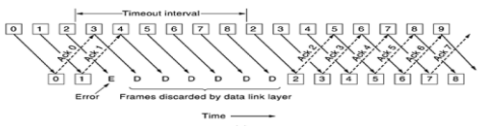
26

•If frame in middle seq. is lost, large no. of succeeding frames arrive at receiver before there is an error with lost frame.

•DB hands out packets to NB in sequence If out of sequence, must buffer them or discard them, while it waits before next in sequence.

27

"GO back N" (GBN)



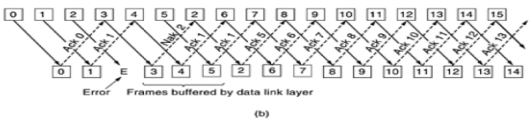
e.g. 1
Receiver B window size=1.
Discard all frames after error.
No ACK is sent. Sender eventually time outs for erred frames and retransmits.
Lot of resends are possible.
=>wastes bandwidth
=>DA needs buffer to hold frames waiting for ACK (might need retransmissions).

In this flow control enforce no more than MAX_SEQ un-ACKed frames are there.
DA disables and enables NA accordingly ACK s may get lost or damaged.

ACK n implies as ACK everything up to frame n.
Each frame has its own timers.
Multiple timers in software.

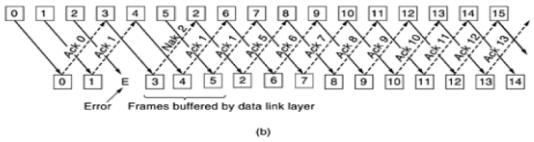
28

SELECTIVE REPEAT



Receiver B window size=n.
Good frames after error are buffered. (Cannot send them to NB and cannot discard).
Frame 2 times out and resent.
At timeout, A resents oldest un-ACKed frames, not a seq
OR
B sends NAK (negative ACK) to provoke a re-send of frame 2 instead of timeout. (=>save time).
While waiting B does not ACK 3-5, simply buffers them.
Sends ACK 1=> indicates that things are fine up to frame 1.

SELECTIVE REPEAT..



(b)

When frame 2 is received at B, it can send bunch 2-5 to NB hence can ACK everything up to 5 so sends ACK 5.
If NAK is lost, timeout will resend frame.
DB needs duffers in memory to hold frames.
Sender window starts at 0 and grows depending on frames it get from NA
Receiver window=constant max_size .In each slot, bit indicates if slot is full or empty.

COMPARISON OF GBN AND SR

	GBN	SR
LOW ERROR	✓	
HIGH ERROR		✓
CHEAP BANDWIDTH	✓	
COSTLY BANDWIDTH		✓
CHEAP MEMORY		✓
COSTLY MEMORY	✓	

NUMBERING FRAMES:

N bits for frame numbers.
2^n distinct frames 0…..2^n-1
After 2^n-1 loop round reuse 0…..2^n-1

LOST FRAMES – Go Back n

Make sure that no more than 2^n -1 frames are un-ACK ed at any point not 2^n frames.
Consider 3 bit frame no. while frames are 0-7

a. Sender sends frame no.0-7.
b. Piggybacked ACK for frame 7 comes eventually back to sender.
c. Sender sends another 8 frames with seq. 0-7.
d. Now again piggybacked ACK comes.

QUESTION:
For second batch not clear if 8 frames arrive sequentially or all 8 get lost.
In both cases receiver would be sending frame 7 as ACK.
Sender has no way of telling i.e. reason MAX no. of frames is kept 2^n-1.

•8 unACKed frames are bad.

- Transmit 0-7. If OK, will get ACK 7.
- Transmit next 0-7.
- If OK, will get ACK=7. (This is piggybacked indicates that what frames has reached).
- If not OK also will get ACK=7 as of previous batch.

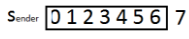
•7 unACKed frames OK.

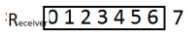
- Transmit 7 frames 0-6.
- If OK receive ACK=6.
- If error receive ACK=7. (previous lot ACK)
- If OK then send frames 7, 0-5.
- If error receive ACK=6.
-If OK then receive ACK=5.

0 1 2 3 4 5 6 7 ←
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 ←

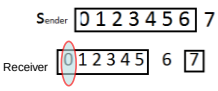
LOST ACKs with SR

Consider receiver buffers incoming out of seq. frames and there is lost ACK.
Consider frames 0-6.



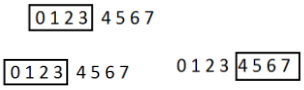


Receiver ACK s these, sends 0-6 to NB, clears buffers and moves window to 7, 0-5.



•Receiver ACK for 6 is lost
•Sender times out and retransmits frame 0.
•Receiver regards 0 of new batch. If hasn't seen 7 yet so ACKs upto 6.
•Sender receives ACK=6. Assumes entire old batch got through.
•Sender advances window and transmits 7, 0-5. 7 is accepted by receiver.
•Receiver passes new 7 and old (duplicate) 0 to NB, N/W layer NB gets incorrect packet and protocol fails.

➔SOLUTION←
SENDER WINDOW SHOULD NOT OVERLAP AT ALL WITH OLD WINDOW.
MAX SIZE=1/2 THE RANGE.



Receiver advances window to 4-7 after receives 0-3.
If it continues getting 0-3, knows these are resends and ignores them, keeps ACK ing though.
Once it gets 4-7, knows ACK s have get through & 0-3 are safe no. to use again.

WHAT TIMEOUT

TOO SHORT=>UNNECESSARY TIMEOUTS.
TOO LONG=>UNNECESSARY DELAY AFTER ERROR.

THANKS

References
Tanenbaum Chapter 3