Lecture 11
Plan: flow control

Review:
Hamming coding: 7-4 coding (7 bits for 4 bits of information)
how to code-decode hamming code

Flow control (flow control and its combination with error control**)**

Q:/ What is flow control?
A:/ techniques for assuring that the transmitting entity doesn't overwhelm a receiving entity with data (I.e. receiver's buffer won't overflow)

Q:/ Why can't we simply synchronize both parties?
A:/ the network, especially internet, is unpredictable. To ensure synchronization, the speed has to set to be very low compared with the available maximal bit rate.

What can we do?

Assuming that no frame is damaged and no frame is lost (a huge assumption)
1). stop-and-wait

- Each frame is acknowledged
- sender cannot send another frame until the previous frame is acknowledged

It is the easiest form of flow control (receiver's buffer will never overflow) and is used when the frame size is large

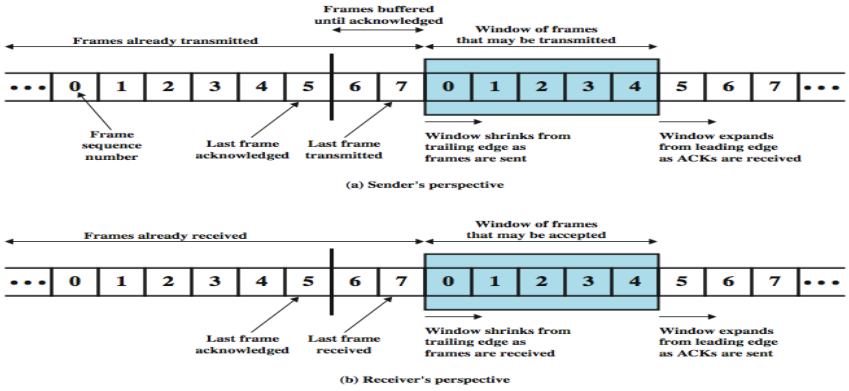
Q:/ Why is it not widely used?
A:/
usually frame size is not large (why?)
may severely waste the bandwidth, especially on networks with long prorogation time
(transmission time: time used to send all bits onto medium; prorogation time: time to send the bits to the receiver)
e.g. in a network with prorogation delay of 1s, the transmission time of a frame is 1ms.
Using stop and wait, the turn around time of the frame and ack is 2 s. I.e. the frame rate is .5 frame /s
In deed, the maximal frame rate can be 1000 frames/s

Q:/ How can we improve it? Or what is the fundamental problem with stop and wait?
A:/ in stop-and-wait, only one frame at a time is in transit. This is the problem. → sliding window

2). Sliding-window flow control
Idea: sender can send w frames without waiting for any acknowledgment
receiver can accept w frames (buffer size is w)

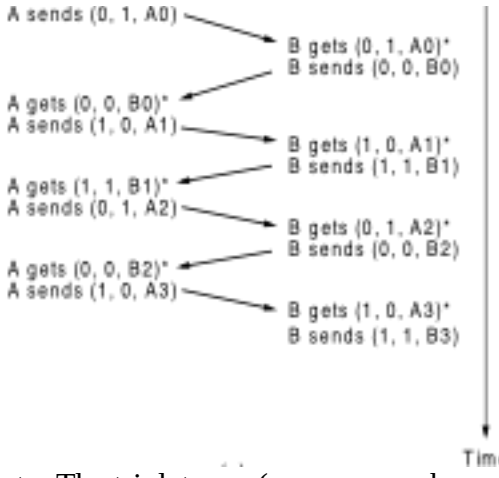
Details:
each frame has a sequence number usually from 0 to 2^k-1 where k is the # of bits used for frame number (mod around 2^k-1)
examples of sliding window.
If we have a window of size 7 and the frames are numbered from 0 to 7.
sender side: window shrinks from the trailing edge when more frames are sent.
Window expands from the leading edge as ACKs are received
receiver side: window shrinks from the trailing edge when more frames are received
window expands from the leading edge as ACKs are sent

Sliding window s like a pipeline (queue)



Explanation: RR3 means "all frames up to frame #2 are acknowledged. In fact, the receiver can get seven frames beginning with frame #3)
Note

- All unacknowledged frames are buffered by the receiver and the sender
- piggybacking can be used when the communication is full duplex (piggyback the ACK on the outgoing frames)



note: The triplets are (sequence, ack number, packet number). * means the packets are sent to the network layer.

- stop-and-wait is basically a sliding window with k=1 (sequence number is 0 1)

If the window is large, then the sender won't have to wait.
Q:/ how large?
A:/ w>=round-time/transmission_time (basically during the time for the first frame to be acknowledged, there are always frames available in the window to send)

Q:/ What if there are errors? – like damaged frames or lost frames?
A:/ combine error control and flow control together.
Note: error detection is used to test if a frame is damaged or not.
Error control is how to handle damaged or lost frames

1. stop-and-wait ARQ (automatic repeat request)
It follows the scheme of stop-and-wait.
But it accommodates lost/damaged frames or ACK (each frame or ack can be lost or damged)
lost frame/ lost ACK/damaged ACK: use timer on the sender's side.
damage frame: resend

normal:
sender sends frame 0
receiver receives frame 0, send ACK 1 (ready to receive frame 1)
sender got the ACK 1.
sender sends frame 1
receiver receives frame 1, sends ACK0 (ready to receive frame 0)
etc

What if
1. a frame is damaged
receiver may ignore the frame or sends back a NAK (negative acknowledgement)
sender will eventually time out and resend the frame (or resend the frame once NAK is received)
eventually the receiver got the correct frame and sends an ACK for it.

2. a frame is lost
sender will eventually timeout
sender resend and receiver acknowledge

3. ACK/NAK got lost or damaged
sender will eventually time-out and resend the frame
receiver got the frame that is out of sync. (expecting 1 but got 0)
receiver discards the duplicate and resend the previous ACK/NAK

As discussed before, stop-and-wait is not efficient (ie a sliding window with window length =1)

Q:/ What if we want to use sliding window idea?
sender sends 0, 1, ----, w-1 frames without acknowledgement.

What if frame 2 is lost?

A:/ The receiver got a frame that is out of order
if the newly arrived frame number is lower than expected → it's a resend
if larger → sth was missing between expected and the frame just received

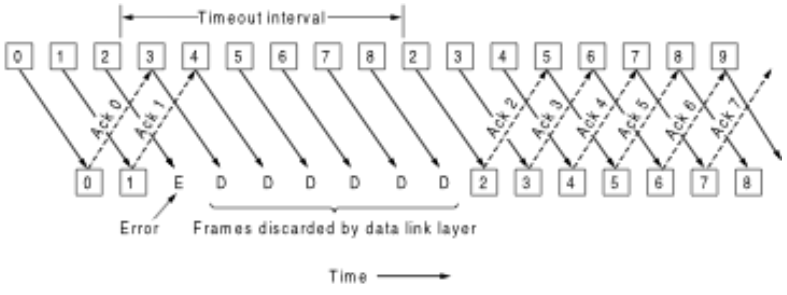
Two solutions
1. Go back N
sender can send w frames without acknowledgement (window size =w)
receiver only accepts the next frame in sequence (window size =1)
sender maintains a timer for each unacknowledged frame

Sender: behaves like sliding window
idea: resend the all frames from the last acknowledged forward
receiver B
sender A

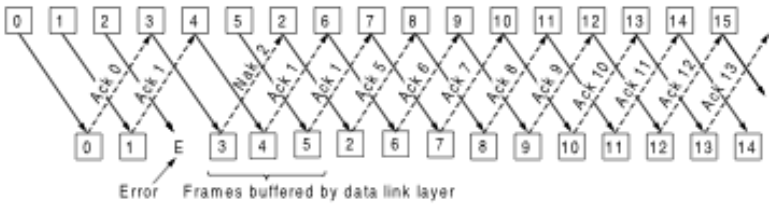
lost/damaged frame
if A sends other frames before the timer of the damaged/lost frame is timed out → B sends REJ to A (also tells A what it is expecting). Everything after the expected frame will be resent
if A times out first, then A sends a query to B asking what B is expecting. → resend similar as before
damaged/lost ACK
if another ACK is sent back before the timer of sender times out → good. no action is needed since ACK is cumulative
if sender times out first, it ask B again about what it is expecting → resend everything

Shortcoming: may keep resending good frames because of one bad frame in the middle

Solution
2. selective reject
Idea: the receiver only requests bad frames to be resent.
catch: needs a bigger buffer in receiver and needs the ability to reassembly out of order frames (not commonly used)


Timeout interval
0 1 2 3 4 5 6 7 8 2 3 4 5 6 7 8 9
Ack 0
Ack 1
Ack 2
Ack 3
Ack 4
Ack 5
Ack 6
Ack 7
0 1 E D D D D D D 2 3 4 5 6 7 8
Error
Frames discarded by data link layer
Time

(a)


0 1 2 3 4 5 2 6 7 8 9 10 11 12 13 14 15
Ack 0
Ack 1
Nak 2
Ack 1
Ack 1
Ack 5
Ack 6
Ack 7
Ack 8
Ack 9
Ack 10
Ack 11
Ack 12
Ack 13
0 1 E 3 4 5 2 6 7 8 9 10 11 12 13 14
Error
Frames buffered by data link layer

(b)