CPE348: Introduction to Computer Networks

Lecture #6: Chapter 2.3

Jianqing Liu
Assistant Professor of Electrical and Computer Engineering, University of Alabama in Huntsville

jianqing.liu@uah.edu
http://jianqingliu.net

Reliable Transmission - Review

- CRC, checksum are used to detect errors.
- If errors,
  - corrupted frames must be discarded;
  - correct frames must be recovered or retransmitted.

Reliable Transmission - Motivation

- Error correction code (ECC) like interleaving and Turbo code typically has high overhead (but still in use)!

- An alternative: re-transmission
  - A link-level mechanism: **Acknowledgement**

Reliable Transmission

- An *acknowledgement* (ACK) is a small control frame saying that it has received the earlier frame.
  - ACK frame only has header (no data)
  - ACK and non-ACK

- The receipt of an ACK indicates that the prior frame was successfully delivered.

Reliable Transmission

But, the ACK could get lost/corrupted too!

Reliable Transmission

- A link-level mechanism: **Timeout**
  - If ACK is not received after a while, then the sender retransmits the original frame.

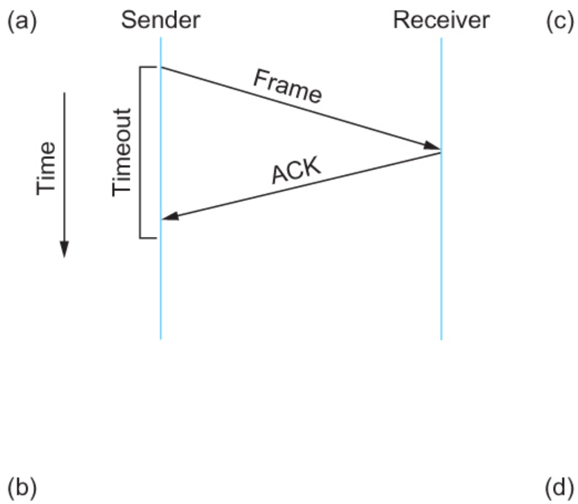
The combination of ***ACK*** and ***Timeout*** to implement reliable delivery is called

Automatic Repeat reQuest (ARQ).

ARQ - Stop and Wait Protocol

- One exemplary ARQ protocol - stop-and-wait
  - After transmitting one frame, the sender waits for an ACK before transmitting the next one.
  - If the ACK does not arrive after Timeout, the sender retransmits the prior frame.

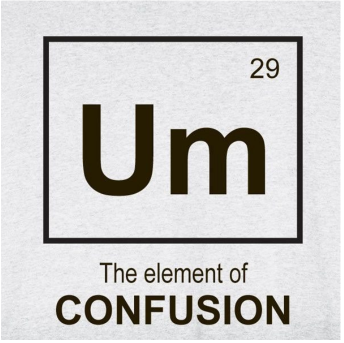
ARQ - Stop and Wait Protocol



Timeline showing four different scenarios for the stop-and-wait algorithm.
(a) The ACK is received before the timer expires; (b) the original frame is lost; (c) the ACK is lost; (d) the timeout fires too soon

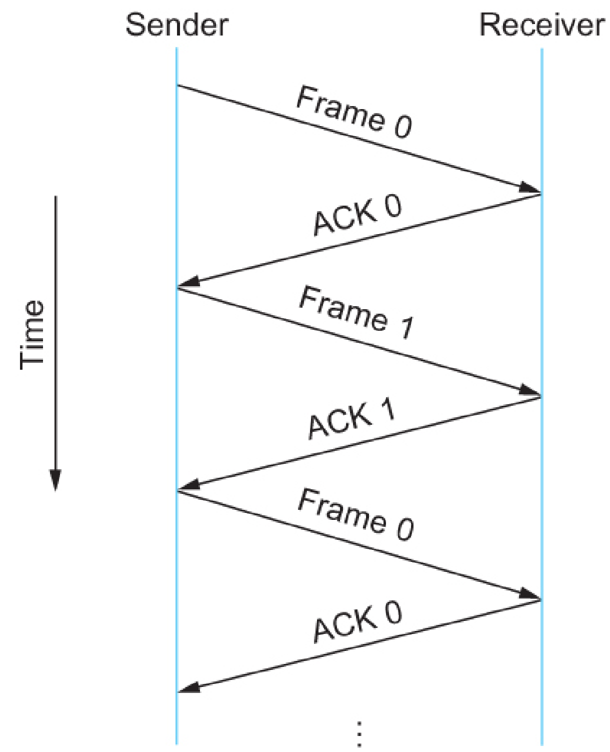
Stop and Wait Protocol – issue 1

- Misunderstanding: when the ACK is lost or delayed
  - The sender re-transmits;
  - The receiver gets a duplicate copy, which causes confusion



Stop and Wait Protocol – issue 1

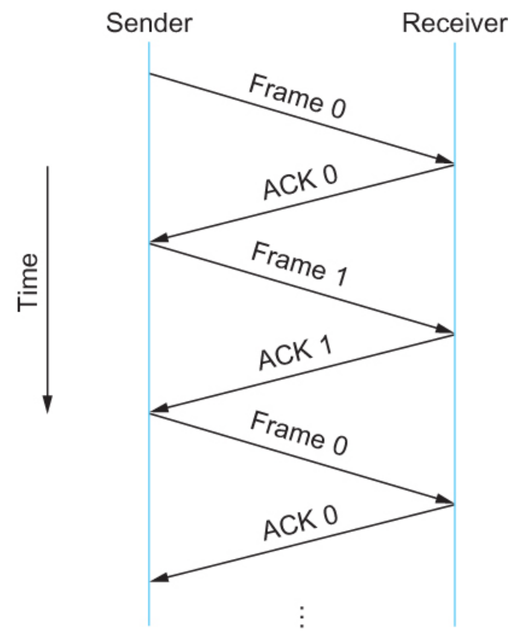
- Solution
  - Use 1 bit sequence number (0 or 1) to represent a frame



Stop and Wait Protocol – issue 2

- Efficiency: the sender has only one outstanding frame on the link at a time
  - poor utilization of channel capacity
  - Sending rate = (bits per frame)/(time per frame = 1 RTT)

- Question: what is the effective data rate in this diagram?



Stop and Wait Protocol – issue 2

- Question cont'

Consider sending a 1 KB frame over a 1.5 Mbps link with a 45 ms RTT

- Since the sender can send only one frame per RTT
    - Maximum Sending rate is

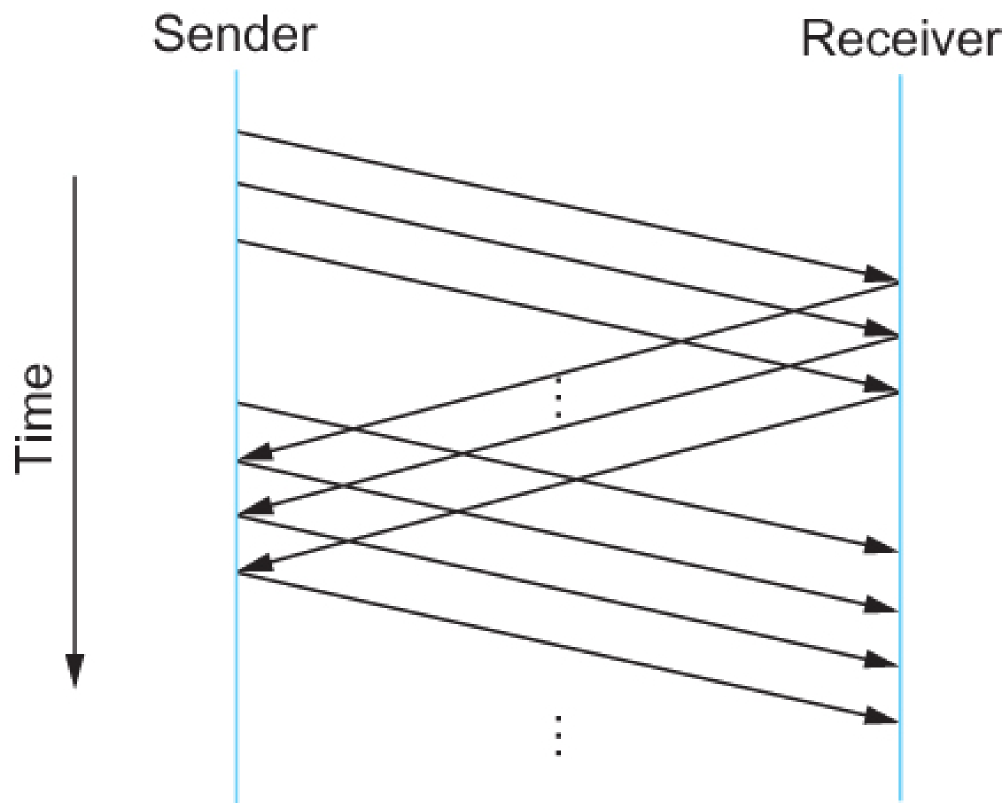
    Bits per frame ÷ Time per frame = 1024 × 8 ÷ 0.045 = 182 Kbps

    Or about one-eighth of the link's capacity

- An alternative solution
    - delay × bandwidth product:

    1.5E6*0.045 = 67,500 bits = 8,437 Bytes, or about eight-times of the frame size

Sliding Window Protocol



The solution to improving efficiency of stop-and-wait protocol

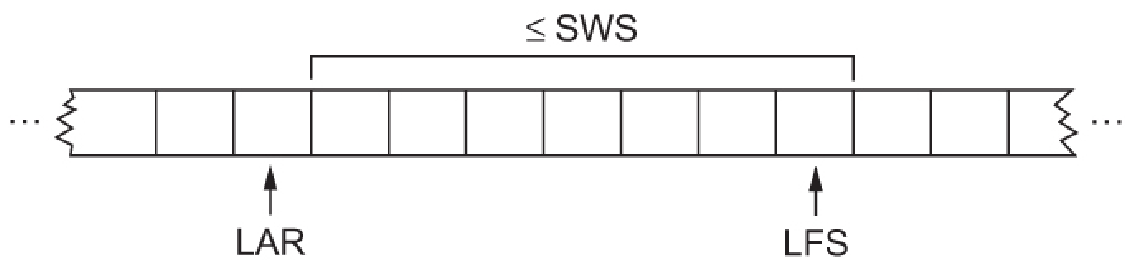
Sliding Window Protocol - Sender

- Sender assigns a sequence number to a frame
- Sender maintains three pointers
  - Sending Window Size (SWS)
  - Last Acknowledgement Received (LAR)
  - Last Frame Sent (LFS)

Sliding Window Protocol - Sender

- One property of these pointers:

$$LFS - LAR \le SWS$$



Sliding Window on Sender

- LAR moves right as ACKs received
- LFS moves right as frames are sent

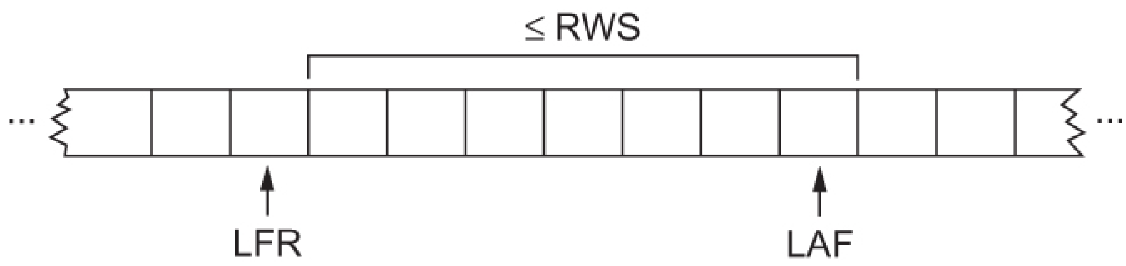
Sliding Window Protocol - Rcvr

- Receiver maintains three pointers
  - Receiving Window Size (RWS)
  - Largest Acceptable Frame (LAF)
  - Last Frame Received (LFR)

Sliding Window Protocol - Rcvr

- Also, one property of these pointers

$$LAF – LFR \leq RWS$$



Sliding Window on Receiver

Sliding Window Protocol - Rcvr

- When a frame arrives, what does the receiver do?

Check

- If SeqNum ≤ LFR or SeqNum > LAF
    - Discard it (the frame is outside the receiver window)
- If LFR < SeqNum ≤ LAF
    - Accept it
    - Send an ACK

Sliding Window Protocol - Rcvr

- Keep a pointer SeqNumToAck
  - Denote the largest SeqNum *not yet ACKed*,
  - All frames with SeqNum less than SeqNumToAck have been received
- Even if higher-numbered packets have been received, the receiver holds the ACK till the frame SeqNumToAck is received.
- The receiver then sets
  - LFR = SeqNumToAck and adjusts
  - LAF = LFR + RWS

Sliding Window Protocol – example

For example, suppose LFR = 1 and RWS = 4

(i.e. the last ACK that the receiver sent was for seq. no. 1 and SeqNumToAck is set to 2)

➔ LAF = 5

If frames 3 and 4 arrive before frame 2, they will be buffered because they are within the receiver window

- But no ACK will be sent since frame 2 is yet to arrive
- Frames 3 and 4 are out of order

Frame 2 arrives (it is late because it was lost first time and had to be retransmitted)

- Receiver Acknowledges Frame 4
- Receiver bumps LFR to 4
- Receiver moves LAF to 8 (LAF = LFR + RWS)
- Receiver sets SeqNumToAck to 5

Sliding Window Protocol – issue 1

- When timeout occurs,
  - Sender waits and is unable to advance its window
- When the packet loss occurs, this scheme is no longer keeping the pipe full
  - The longer it takes to notice that a packet loss has occurred, the more severe the problem becomes
- How to improve this
  - Negative Acknowledgement (NAK)
  - Additional Acknowledgement
  - Selective Acknowledgement

Efficiency!

Sliding Window Protocol – issue 1

- Negative Acknowledgement (NAK)
  - Receiver sends NAK for frame 2 when frame 3 arrive (solicitate)
- Additional Acknowledgement
  - Receiver sends additional ACK for frame 1 when frame 3 arrives (duplicate)
- Selective Acknowledgement
  - Receiver will acknowledge exactly those frames it has received, rather than the highest number frames (explicit)
    - Receiver will acknowledge frames 3 and 4
    - Sender knows frame 2 is lost

Sliding Window Protocol – issue 2

How to select the window size

- SWS is easy to compute
  - Use Delay × Bandwidth/(frame size) – keeps the pipe full
- RWS can be anything
  - Two common settings
    - RWS = 1, OK!
      No buffer for frames that arrive out of order
    - RWS = SWS, OK!
      The rcvr buffers frames that sender transmits
    - RWS > SWS. NO!
      Why?

Sliding Window Protocol – issue 2

Before we present the issue,
let's first look into the SeqNum!

- Frame sequence number
  - Specified in the header
  - Finite size

    3 bits: eight possible sequence number: 0, 1, 2, 3, 4, 5, 6, 7

    MaxSeqNum = 8
  - Wrap around – reuse sequence numbers

Sliding Window Protocol – issue 2

How does the MaxSeqNum affect SWS and RWS?

- Question: SWS + 1 ≤ MaxSeqNum, is this sufficient?
  - Depends on RWS
  - If RWS = 1, then sufficient
  - If RWS = SWS, then not good enough

- Example, we have 4 sequence numbers: 0, 1, 2, 3
  RWS = SWS = 3
  Sender sends 0, 1, 2
  Receiver receives 0, 1, 2 and ACKs 0, 1, 2
  ACK (0, 1, 2) are lost
  Sender re-transmits 0, 1, 2
  Receiver is expecting 3, 0, 1

Confusion!

Sliding Window Protocol – issue 2

- To avoid confusion,
  - If RWS = SWS (remember makes no sense for RWS > SWS)
    SWS < (MaxSeqNum + 1)/2 or MaxSeqNum > 2*SWS – 1
  - If RWS < SWS, then MaxSeqNum may be less than 2*SWS - 1

In general, SWS should be no more than a half of MaxSeqNum, but depends!