Reliable Transmission

Chapter 2

- CRC is used to detect errors.
- Some error codes are strong enough to correct errors.
- The overhead is typically too high.
- Corrupt frames must be discarded.
- A link-level protocol that wants to deliver frames reliably must recover from these discarded frames.
- This is accomplished using a combination of two fundamental mechanisms
  - Acknowledgements and Timeouts

MK MORGAN KAUFMANN 1

Reliable Transmission

Chapter 2

- An *acknowledgement* (ACK for short) is a small control frame that a protocol sends back to its peer saying that it has received the earlier frame.
  - A control frame is a frame with header only (no data).
- The receipt of an *acknowledgement* indicates to the sender of the original frame that its frame was successfully delivered.

MK MORGAN KAUFMANN 2

Reliable Transmission

Chapter 2

- If the sender does not receive an *acknowledgment* after a reasonable amount of time, then it retransmits the original frame.
- The action of waiting a reasonable amount of time is called a *timeout*.
- The general strategy of using *acknowledgements* and *timeouts* to implement reliable delivery is sometimes called Automatic Repeat reQuest (ARQ).

MK
3

Stop and Wait Protocol

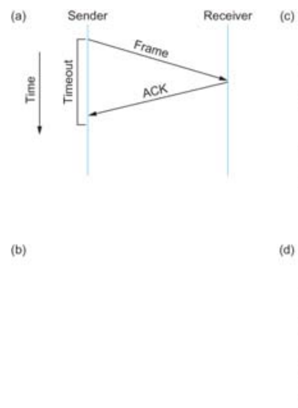
Chapter 2

- Idea of stop-and-wait protocol is straightforward
  - After transmitting one frame, the sender waits for an acknowledgement before transmitting the next frame.
  - If the acknowledgement does not arrive after a certain period of time, the sender times out and retransmits the original frame

MK
4

Stop and Wait Protocol

Chapter 2



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

5

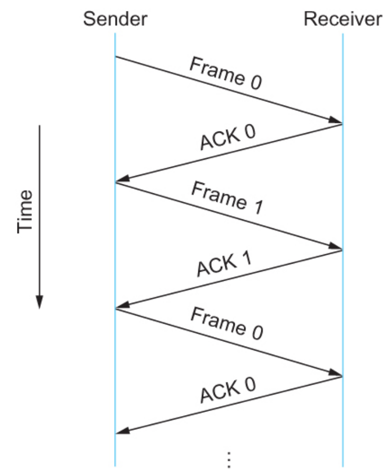
Stop and Wait Protocol

Chapter 2

- If the acknowledgment is lost or delayed in arriving
  - The sender times out and retransmits the original frame,
  - Receiver will think that it is the next frame since it has correctly received and acknowledged the first frame
  - As a result, duplicate copies of frames will be delivered
- How to solve lost ACK problem
  - Use 1 bit sequence number (0 or 1)
  - The sender retransmits frame 0,
  - The receiver can determine that it is seeing a second copy of frame 0 rather than the first copy of frame 1 and therefore can ignore it
  - the receiver still acknowledges it, in case the first ACK was lost

6

Stop and Wait Protocol



Timeline for stop-and-wait with 1-bit sequence number

Stop and Wait Protocol - issue

- With Stop and Wait, The sender has only one outstanding frame on the link at a time
  - This may be far below the link's capacity – poor utilization
  - Sending rate = (bits per frame)/(time per frame = 1 RTT)

Stop and Wait Protocol - issue

Chapter 2

- Consider sending a 1 KB (1024 Bytes) frame over a 1.5 Mbps link with a 45 ms RTT
  - The link has a delay × bandwidth product of 67.5 Kb or approximately 8 KB (1.5E6*0.045 = 67,500 bits = 8,437 Bytes)
  - Since the sender can send only one frame per RTT
    - Maximum Sending rate is

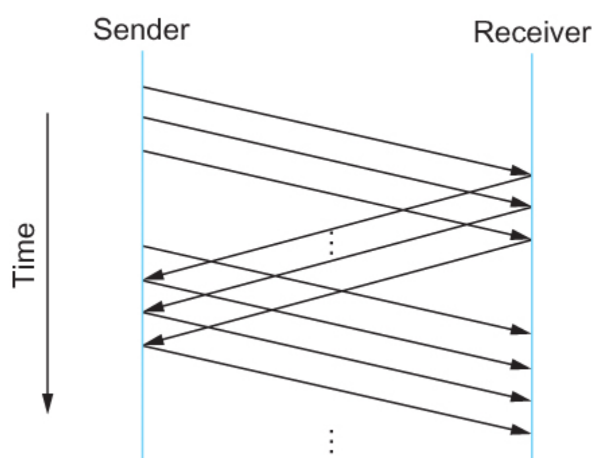
    Bits per frame ÷ Time per frame = 1024 × 8 ÷ 0.045 = 182 Kbps

    Or about one-eighth of the link's capacity
  - To use the link fully, then sender should transmit up to eight frames before having to wait for an acknowledgement

9

Sliding Window Protocol

Chapter 2



Timeline for Sliding Window Protocol

10

Sliding Window Protocol - Sender

Chapter 2

- Sender assigns a sequence number denoted as SeqNum to each frame.
  - Assume it can grow infinitely large
- Sender maintains three variables
  - Sending Window Size (SWS)
    - Upper bound on the number of outstanding (unacknowledged) frames that the sender can transmit
  - Last Acknowledgement Received (LAR)
    - Sequence number of the last acknowledgement received
  - Last Frame Sent (LFS)
    - Sequence number of the last frame sent

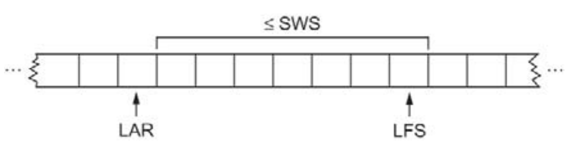
MK 11

Sliding Window Protocol - Sender

Chapter 2

- Sender also maintains the following invariant (property)

  LFS – LAR ≤ SWS



Sliding Window on Sender

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

MK 12

Sliding Window Protocol - Sender

Chapter 2

- When an acknowledgement arrives
  - the sender moves LAR to right
  - Allows the sender to transmit another frame
- The sender associates a timer with each frame it transmits
  - It retransmits the frame if the timer expires before the ACK is received
- Note that the sender has to be willing to buffer up to SWS frames
  - WHY?

13

Sliding Window Protocol - Rcvr

Chapter 2

- Receiver maintains three variables
  - Receiving Window Size (RWS)
    - Upper bound on the number of out-of-order frames that the receiver is willing to accept
  - Largest Acceptable Frame (LAF)
    - Sequence number of the largest acceptable frame
  - Last Frame Received (LFR)
    - Sequence number of the last frame received and acknowledged

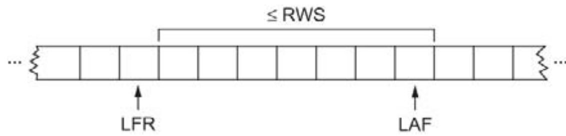
14

Sliding Window Protocol - Rcvr

Chapter 2

- Receiver also maintains the following invariant

  LAF – LFR ≤ RWS



Sliding Window on Receiver

MK 15

Sliding Window Protocol - Rcvr

Chapter 2

- When a frame with sequence number SeqNum arrives, what does the receiver do?
  - If SeqNum ≤ LFR or SeqNum > LAF
    - Discard it (the frame is outside the receiver window)
  - If LFR < SeqNum ≤ LAF
    - Accept it
    - Now the receiver needs to decide whether or not to send an ACK

MK 16

Sliding Window Protocol - Rcvr

- Let SeqNumToAck
  - Denote the largest sequence number *not yet acknowledged*,
  - All frames with sequence number less than SeqNumToAck have been received
  - When Frame with sequence number SeqNumToAck is received, an ACK is sent for that frame
- The receiver acknowledges the receipt of SeqNumToAck even if higher-numbered packets have been received
  - This acknowledgement is said to be cumulative.
- The receiver then sets
  - LFR = SeqNumToAck and adjusts
  - LAF = LFR + RWS

Sliding Window Protocol

For example, suppose LFR = 5 and RWS = 4

(i.e. the last ACK that the receiver sent was for seq. no. 5 and SeqNumToAck is set to 6)

➔ LAF = 9

If frames 7 and 8 arrive before frame 6, they will be buffered because they are within the receiver window

But no ACK will be sent since frame 6 is yet to arrive

Frames 7 and 8 are out of order

Frame 6 arrives (it is late because it was lost first time and had to be retransmitted)

Receiver Acknowledges Frame 8

Receiver bumps LFR to 8

Receiver moves LAF to 12 (LAF = LFR + RWS)

Receiver sets SeqNumToAck to 9

Issues with Sliding Window Protocol

- When timeout occurs, the amount of data in transit decreases
  - Sender 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

Issues with Sliding Window Protocol

- Negative Acknowledgement (NAK)
  - Receiver sends NAK for frame 6 when frame 7 arrive (in the previous example)
    - Unnecessary since sender's timeout mechanism will be sufficient to catch the situation
    - Adds additional complexity
- Additional Acknowledgement
  - Receiver sends additional ACK for frame 5 when frame 7 arrives
    - Sender uses duplicate ACK as a clue for frame loss
- Selective Acknowledgement
  - Receiver will acknowledge exactly those frames it has received, rather than the highest number frames
    - Receiver will acknowledge frames 7 and 8
    - Sender knows frame 6 is lost
    - Sender can keep the pipe full (additional complexity)

Issues with Sliding Window Protocol

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 setting
    - RWS = 1
      No buffer for frames that arrive out of order
    - RWS = SWS
      The receiver buffers frames that the sender transmits
  - It does not make any sense to keep RWS > SWS. WHY?

Cannot have more than SWS frames arrive out of order

Issues with Sliding Window Protocol

- Finite Sequence Number
  - Frame sequence number is specified in the header field
    - Finite size
      3 bits: eight possible sequence number: 0, 1, 2, 3, 4, 5, 6, 7
    - It is necessary to wrap around – reuse sequence numbers

Issues with Sliding Window Protocol

- How to distinguish between different incarnations of the same sequence number?
  - Number of possible sequence number must be larger than the number of outstanding frames allowed
    - Stop and Wait:
      - One outstanding frame ➔ 2 distinct sequence number (0 and 1)
    - Let MaxSeqNum be the number of available sequence numbers
    - SWS + 1 ≤ MaxSeqNum
      - Is this sufficient?

Issues with Sliding Window Protocol

- SWS + 1 ≤ MaxSeqNum
  - Is this sufficient?
  - Depends on RWS
  - If RWS = 1, then sufficient
  - If RWS = SWS, then not good enough
- For example, we have eight sequence numbers

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

    RWS = SWS = 7

  Sender sends 0, 1, …, 6

  Receiver receives 0, 1, … ,6

  Receiver acknowledges 0, 1, …, 6

  ACK (0, 1, …, 6) are lost

  Sender retransmits 0, 1, …, 6

  Receiver is expecting 7, 0, …., 5

Issues with Sliding Window Protocol

- To avoid confusion on which packet has been received,
  - If RWS = SWS (remember makes no sense for RWS > SWS)
    SWS < (MaxSeqNum + 1)/2 or MaxSeqNum > 2*SWS – 1

    MaxSeqNum is the number of sequence numbers required.

    For Stop and Wait ARQ, SWS = 1, so MaxSeqNum > 1

  - If RWS < SWS, then MaxSeqNum may be less than 2*SWS - 1

Roles of Sliding Window Protocol

- Serves three different roles
  - Reliable
  - Preserve the order
    - Each frame has a sequence number
    - The receiver makes sure that it does not pass a frame up to the next higher-level protocol until it has already passed up all frames with a smaller sequence number
  - Frame control (flow control)
    - Receiver is able to throttle the sender
      - Keeps the sender from transmitting more data than the receiver is able to process
      - Can send how many more frames it can accept
- Sample code on pages 111-117