

CSE 461: Introduction to Computer Communications Networks  
Winter 2009

Module 3  
Direct Link Networks – Part A

John Zahorjan  
zahorjan@cs.washington.edu  
534 Allen Center

This Module's Topics

Overview of Computer Networking

- Overview – Scope of today's discussion
- Encoding / Framing / Error Detection
- **Reliable Transmission**

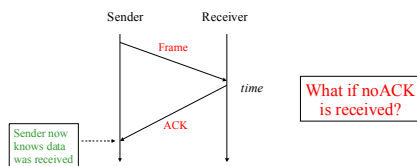
1/21/09

CSE 461 09wi

2

Reliable Transmission

- Because there may be uncorrectable errors (no matter what ECC scheme is used), how can the sender be sure that the receiver got the data?
  - Receiver must tell sender
  - Common scheme: positive acknowledgements (ACKs)



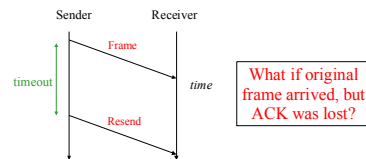
1/21/09

CSE 461 09wi

3

Timeouts / Automatic Repeat Request (ARQ)

- If no ACK comes back, the sender must re-send the data (ARQ)
  - When is the sender sure that no ACK is coming back?
    - Because as a practical matter delays are very difficult to bound, in some sense it can never be sure
    - Sender chooses some reasonable **timeout** – if the ACK isn't back in that much time, it assumes it will never see an ACK, and re-sends



1/21/09

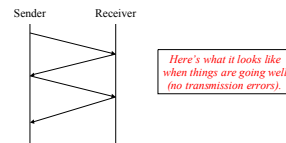
CSE 461 09wi

4

## Duplicate Detection: Sequence Numbers

- So that the receiver can detect (and discard) duplicates, distinct frames are given distinct sequence numbers
  - E.g., 0, 1, 2, 3, ...
- When a frame is re-sent, it is re-sent with the same sequence number as the original
- The receiver keeps some information about what sequence numbers it has seen, and discards arriving packets that are duplicates

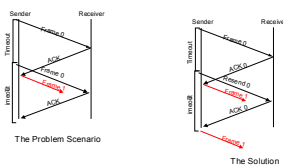
## Stop-and-Wait Protocol



- Sender doesn't send next packet until he's sure receiver has last packet
- The packet/ACK sequence enables reliability
- Sequence numbers help avoid problem of duplicate packets

## Stop & wait sequence numbers

- Sequence numbers enable the receiver to discard duplicates
- ACKs must carry sequence number info as well



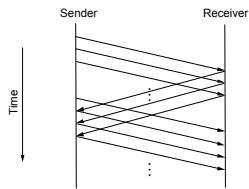
- Stop & wait allows one outstanding frame, requires two distinct sequence numbers

## Problem with Stop-And-Wait: Performance

- Problem: "keeping the pipe full"
  - If the bandwidth-delay product is much larger than a packet size, the sender will be unable to keep the link busy
- Example
  - 1.5Mbps link x 45ms RTT = 67.5Kb (8KB)
  - 1KB frames implies 1/8th link utilization
- Solution: allow multiple frames "in flight"

## Solution: Allow Multiple Frames in Flight

- This is a form of pipelining



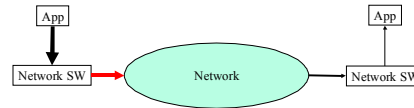
1/21/09

CSE 461 09wi

9

## Flow Control

Why can't we allow the sender to send as fast as it can, timing out and re-sending each frame as necessary?



1/21/09

CSE 461 09wi

10

## Flow Control

- Flow control:
  - Receiver needs to buffer data until it can be delivered to higher layers
    - if the sender is much faster than the receiver, it will overwhelm it, causing the receiver to run out of buffer space
  - Additionally, if a frame is lost, the receiver will receive frames "out of order". It wants to buffer those frames to avoid retransmission, but cannot deliver them to the client until the missing frame is re-sent and received
  - Finally, sender needs to buffer frames in case it has to resend them
- Flow control is the notion that the sender must limit the rate at which it transmits to something below the raw bandwidth of the link
- A common, important approach to flow control is the *sliding window protocol*

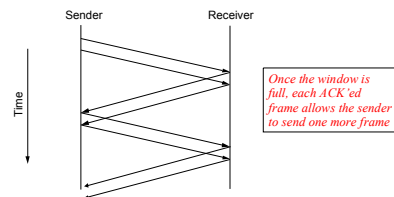
1/21/09

CSE 461 09wi

11

## Sliding Window Protocol

- There is some maximum number of un-ACK'ed frames the sender is allowed to have in flight
  - We call this "the window size"
  - Example: window size = 2



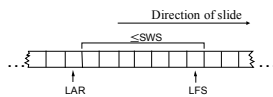
1/21/09

CSE 461 09wi

12

## Sliding Window: Sender

- Assign sequence number to each frame (**SeqNum**)
- Maintain three state variables:
  - send window size (**SWS**)
  - last acknowledgment received (**LAR**)
  - last frame sent (**LFS**)
- Maintain invariant:  $LFS - LAR \leq SWS$



- Advance **LAR** when ACK arrives
- Buffer up to **sws** frames

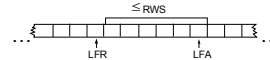
1/21/09

CSE 461 09wi

13

## Sliding Window: Receiver

- Maintain three state variables
  - receive window size (**RWS**)
  - largest frame sequence number acceptable (**LFA**)
  - last frame received (**LFR**)
- Maintain invariant:  $LFA - LFR \leq RWS$



- Frame **SeqNum** arrives:
  - if  $LFR < SeqNum \leq LFA \Rightarrow$  accept + send ACK
  - if  $SeqNum \leq LFR$  or  $SeqNum > LFA \Rightarrow$  discard

1/21/09

CSE 461 09wi

14

## ACKs

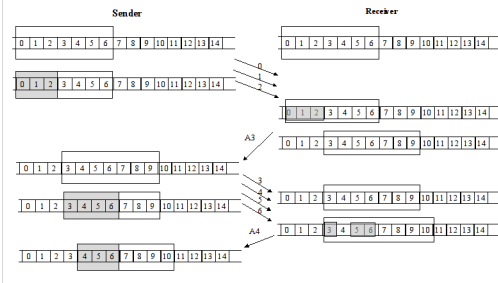
- Send **cumulative** ACKs
  - send ACK for largest frame such that all frames less than this have been received
  - Why?
- Send an ACK each time a packet with **SeqNum** in the window arrives
  - even if you've seen that packet already
  - Why?

1/21/09

CSE 461 09wi

15

## Sliding Window Example

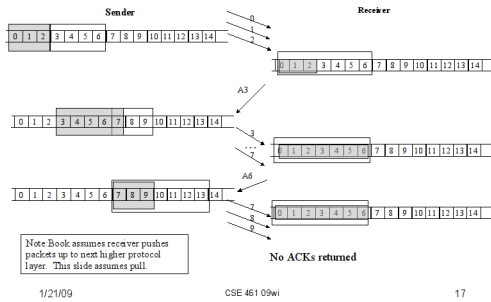


1/21/09

CSE 461 09wi

16

## What If Receiver Client Is Slow?



## Sequence Number Space

- **SeqNum** field is finite; sequence numbers wrap around
- Sequence number space must be larger than number of outstanding frames
- $SWS \leq \text{MaxSeqNum} - 1$  is not sufficient
  - suppose 3-bit **SeqNum** field (0..7)
  - $SWS = RWS = 7$
  - sender transmit frames 0..6
  - arrive successfully, but ACKs lost
  - sender retransmits 0..6
  - receiver expecting 7, 0..5, but receives the original incarnation of 0..5
- $SWS < (\text{MaxSeqNum} + 1) / 2$  is correct rule
- Intuitively, **SeqNum** "slides" between two halves of sequence number space

1/21/09

CSE 461 09wi

18

## Sliding Window Summary

- Sliding window is best known algorithm in networking
- First role is to enable reliable delivery of packets
  - Timeouts and acknowledgements
- Second role is to enable in order delivery of packets
  - Receiver doesn't pass data up to app until it has packets in order
- Third role is to enable flow control
  - Prevents server from overflowing receiver's buffer

1/21/09

CSE 461 09wi

19