CS 640 Introduction to Computer Networks

Lecture 4

CS 640

Today's lecture

- Error detection
- Reliability through retransmission

CS 640

Error detection

- Typical errors important to protect against
  - Random bits flipped
  - Bursts of corrupted bits
- Aim of error detection schemes
  - Catch most common errors
  - No solution can catch all errors
  - Strengths depends on algorithm and size increase
- Example: parity bit

CS 640

Two dimensional parity

- Stronger than parity bit
  - Catches 1,2,3 bit errors
  - Catches most 4 bit errors
- Easy to compute

	Data	Parity bits
Data	0101001	1
	1101001	0
	1011110	1
	0001110	1
	0110100	1
	1011111	0
Parity byte	1111011	0

Internet checksum

- Sum of 16 bit words in message
- When result exceeds 2^{16} drop 17^{th} bit, add 1
- Uses 1's complement arithmetic
- Easy to compute in software (even in the '70s)
- Weaker error detection than CRC

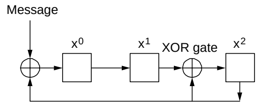
Cyclic redundancy check

```
                    11111001
Generator --> 1101 )10011010000 <-- Message
                    1101
                     1001
                     1101
                      1000
                      1101
                       1011
                       1101
                        1100
                        1101
                           1000
                           1101
                            101 <-- Remainder
```

- Message is a polynomial
- Coefficients modulo 2
  - Values can be 0 or 1
  - +,- same as XOR
- Checksum is remainder of division msg. / generator

CRC – contd.

- Size of remainder depends on size of generator
- Error detection properties depend on generator
- Standards specify generator
- Easy to implement in hardware and software



Today's lecture

- Error detection
- Reliability through retransmission

Reliable transmission

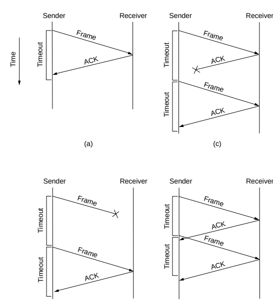
- Frames/packets can be lost, corrupted
- Retransmit to ensure reliability (error correction)
  - Most common at transport layer (layer 4)
  - Done in some data link layers too (layer 2)
- How does sender know when to retransmit?
  - Use *acknowledgements* and *timeouts*

Acknowledgements & Timeouts

- An *acknowledgement* (ACK) is a packet sent by one host in response to a packet it has received
  - Making a packet an ACK is simply a matter of changing a field in the transport header
  - Data can be *piggybacked* in ACKs
- A *timeout* is a signal that an ACK to a packet that was sent has not yet been received within a specified time
  - A timeout triggers a *retransmission* of the original packet from the sender
  - How are timers set?

CS 640

Acknowledgements & Timeouts



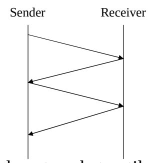
CS 640

Finding the right length for timeout

- Propagation delay: delay between transmission and receipt of packets between hosts
- Propagation delay can be used to estimate timeout period
- How can propagation delay be measured?
- What else must be considered in the measurement?
  - Harder for transport layer than for data link layer

CS 640

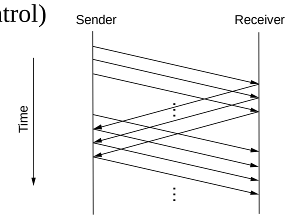
Stop-and-Wait Process



- Sender won't send next packet until sure receiver has last one
- The packet/Ack sequence enables reliability and flow control
- Sequence numbers help avoid problem of duplicate packets
- Problem: keeping the pipe full
- Example
  – 1.5Mbps link x 45ms RTT = 67.5Kb (8KB)
  – 1KB frames implies 1/8th link utilization

Solution: Pipeline via Sliding Window

- Allow multiple outstanding (un-ACKed) frames
- Upper bound on un-ACKed frames, called *window* (flow control)

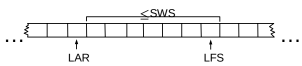


Buffering on Sender and Receiver

- Sender buffers data so that if data lost, it can resend
- Receiver buffers data so that if data is received out of order, it can be held until all packets are received
- How can we prevent the sender overflowing receiver's buffer (flow control)?
  – Receiver tells sender its buffer size during connection setup
- How can we ensure reliability?
  – Go-Back-N
    - Send all N un-ACKed packets when a loss is signaled (inefficient)
  – Selective retransmit
    - Only send un-ACKed packets (a bit trickier to implement)

Sliding Window: Sender

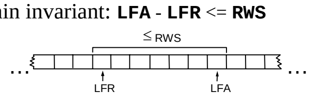
- Assign sequence number to each frame (**SeqNum**)
- Maintain three state variables:
  - send window size (**SWS**)
  - last acknowledgment received (**LAR**)
  - last frame sent (**LFS**)
- Maintain invariant: **LFS** - **LAR** <= **SWS**



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

Sliding Window: Receiver

- Maintain three state variables
  - receive window size (**RWS**)
  - largest frame acceptable (**LFA**)
  - last frame received (**LFR**)
- Maintain invariant: **LFA** - **LFR** <= **RWS**

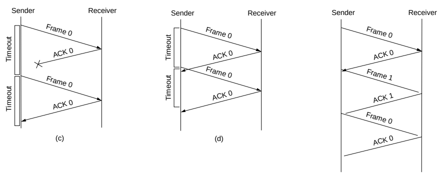


- Frame **SeqNum** arrives:
  - if **LFR** < **SeqNum** < = **LFA** then accept
  - if **SeqNum** < = **LFR** or **SeqNum** > **LFA** then discard
- Send *cumulative* ACKs – send ACK for largest frame such that all frames less than this have been received

Sequence Number Space

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

Stop & wait sequence numbers



- Simple sequence numbers enable *the client* to discard duplicate copies of the same frame
- Stop & wait allows one outstanding frame, requires two distinct sequence numbers

Another Pipelining Possibility: Concurrent Logical Channels

- Multiplex 8 logical channels over a single link
- Run stop-and-wait on each logical channel
- Maintain three state bits per channel
  - channel busy
  - current sequence number out
  - next sequence number in
- Header: 3-bit channel num, 1-bit sequence num
  - 4-bits total, same as sliding window protocol
- Separates *reliability* from *order*

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 next layer until it has packets in order
- Third role is to enable flow control
  - Prevents server from overflowing receiver’s buffer

Sliding Window Example