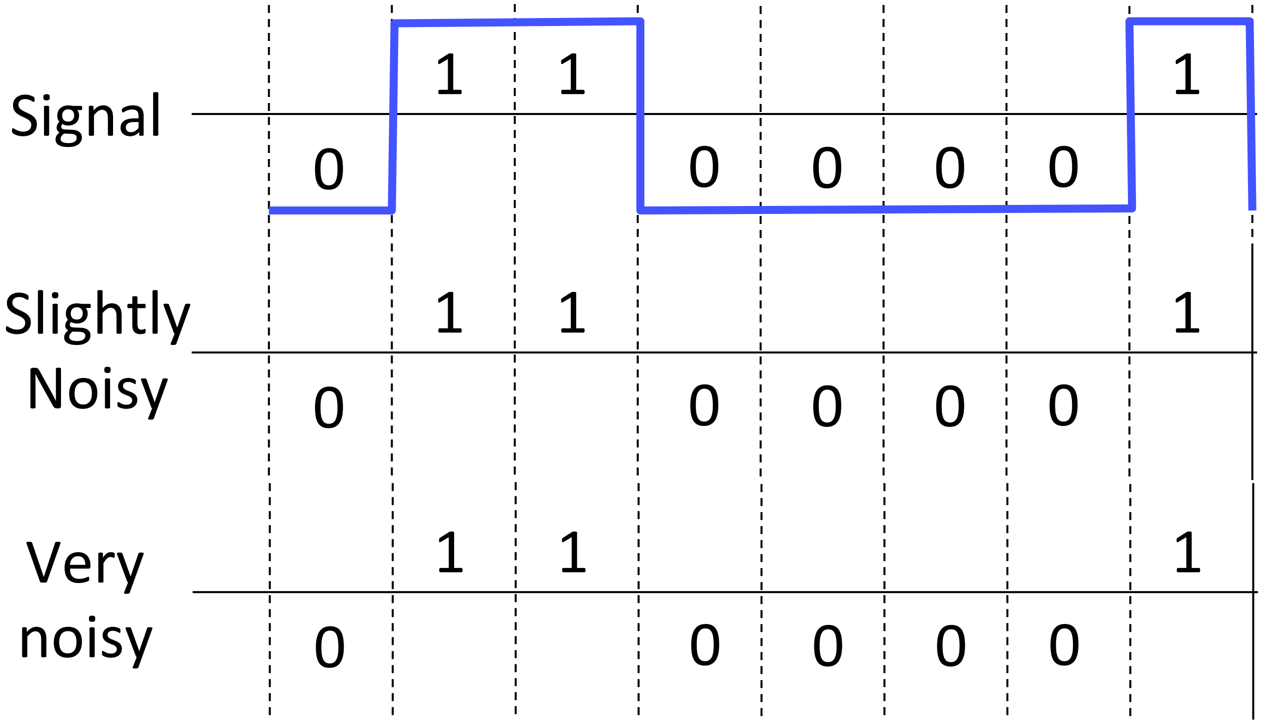
Computer Networks

Shyam Gollakota

Topic

- Some bits will be received in error due to noise. What can we do?
  - Detect errors with codes »
  - Correct errors with codes »
  - Retransmit lost frames ← Later
- Reliability is a concern that cuts across the layers – we'll see it again

Problem – Noise may flip received bits



Approach – Add Redundancy

- Error detection codes
  - Add check bits to the message bits to let some errors be detected
- Error correction codes
  - Add more check bits to let some errors be corrected

- Key issue is now to structure the code to detect many errors with few check bits and modest computation

Motivating Example

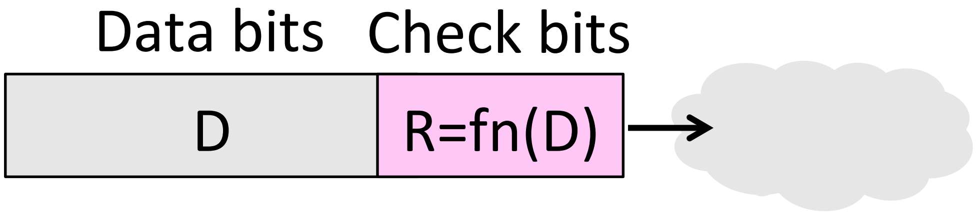
- A simple code to handle errors:
  - Send two copies! Error if different.
- How good is this code?
  - How many errors can it detect/correct?
  - How many errors will make it fail?

Motivating Example (2)

- We want to handle more errors with less overhead
  - Will look at better codes; they are applied mathematics
  - But, they can't handle all errors
  - And they focus on accidental errors (will look at secure hashes later)

Using Error Codes

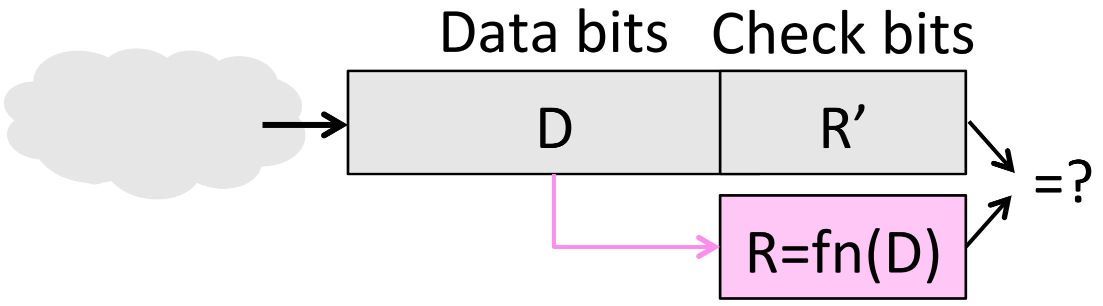
- Codeword consists of D data plus R check bits (=systematic block code)



- Sender:
  - Compute R check bits based on the D data bits; send the codeword of D+R bits

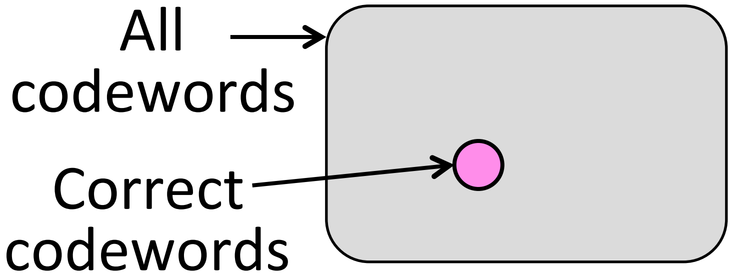
Using Error Codes (2)

- Receiver:
  - Receive D+R bits with unknown errors
  - Recompute R check bits based on the D data bits; error if R doesn't match R'



Intuition for Error Codes

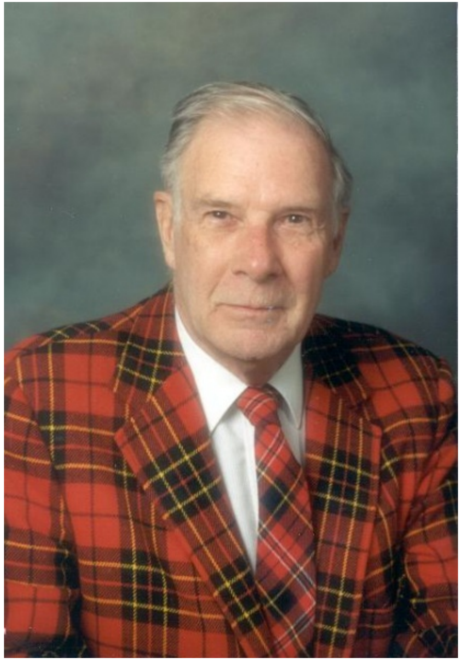
- For D data bits, R check bits:



- Randomly chosen codeword is unlikely to be correct; overhead is low

R.W. Hamming (1915-1998)

- Much early work on codes:
  - "Error Detecting and Error Correcting Codes", BSTJ, 1950
- See also:
  - "You and Your Research", 1986



Source: IEEE GHN, © 2009 IEEE

Hamming Distance

- Distance is the number of bit flips needed to change D_1 to D_2
- Hamming distance of a code is the minimum distance between any pair of codewords

Hamming Distance (2)

- Error detection:
  - For a code of distance d+1, up to d errors will always be detected

Hamming Distance (3)

- Error correction:
  - For a code of distance 2d+1, up to d errors can always be corrected by mapping to the closest codeword

Topic

- Some bits may be received in error due to noise. How do we detect this?
  - Parity »
  - Checksums »
  - CRCs »
- Detection will let us fix the error, for example, by retransmission (later).

Simple Error Detection – Parity Bit

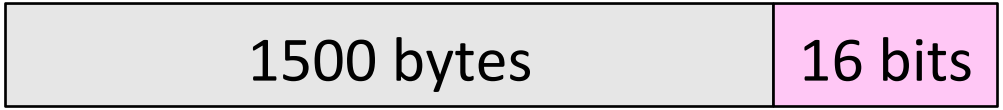
- Take D data bits, add 1 check bit that is the sum of the D bits
  - Sum is modulo 2 or XOR

Parity Bit (2)

- How well does parity work?
  - What is the distance of the code?
  - How many errors will it detect/ correct?
- What about larger errors?

Checksums

- Idea: sum up data in N-bit words
  - Widely used in, e.g., TCP/IP/UDP



- Stronger protection than parity

Internet Checksum

- Sum is defined in 1s complement arithmetic (must add back carries)
  - And it’s the negative sum
- *“The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words …”* – RFC 791

Internet Checksum (2)

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add

3. Add any carryover back to get 16 bits

4. Negate (complement) to get sum

```
0001
f203
f4f5
f6f7
```

Internet Checksum (3)

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

```
    0001
    f203
    f4f5
    f6f7
 +(0000)
 ------
   2ddf0
    ↓
    ddf0
 +     2
 ------
    ddf2
    ↓
    220d
```

Internet Checksum (4)

Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add

```
   0001
   f203
   f4f5
   f6f7
+  220d
 ------
```

3. Add any carryover back to get 16 bits

4. Negate the result and check it is 0

Internet Checksum (5)

Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
   0001
   f203
   f4f5
   f6f7
+  220d
------
  2fffd
    ↓
   fffd
+     2
------
   ffff
    ↓
   0000
```

Internet Checksum (6)

- How well does the checksum work?
  - What is the distance of the code?
  - How many errors will it detect/ correct?
- What about larger errors?

Cyclic Redundancy Check (CRC)

- Even stronger protection
  - Given n data bits, generate k check bits such that the n+k bits are evenly divisible by a generator C
- Example with numbers:
  - n = 302, k = one digit, C = 3

CRCs (2)

- The catch:
  - It's based on mathematics of finite fields, in which "numbers" represent polynomials
  - e.g, 10011010 is $x^7 + x^4 + x^3 + x^1$

- What this means:
  - We work with binary values and operate using modulo 2 arithmetic

CRCs (3)

- Send Procedure:

1. Extend the n data bits with k zeros
2. Divide by the generator value C
3. Keep remainder, ignore quotient
4. Adjust k check bits by remainder

- Receive Procedure:

1. Divide and check for zero remainder

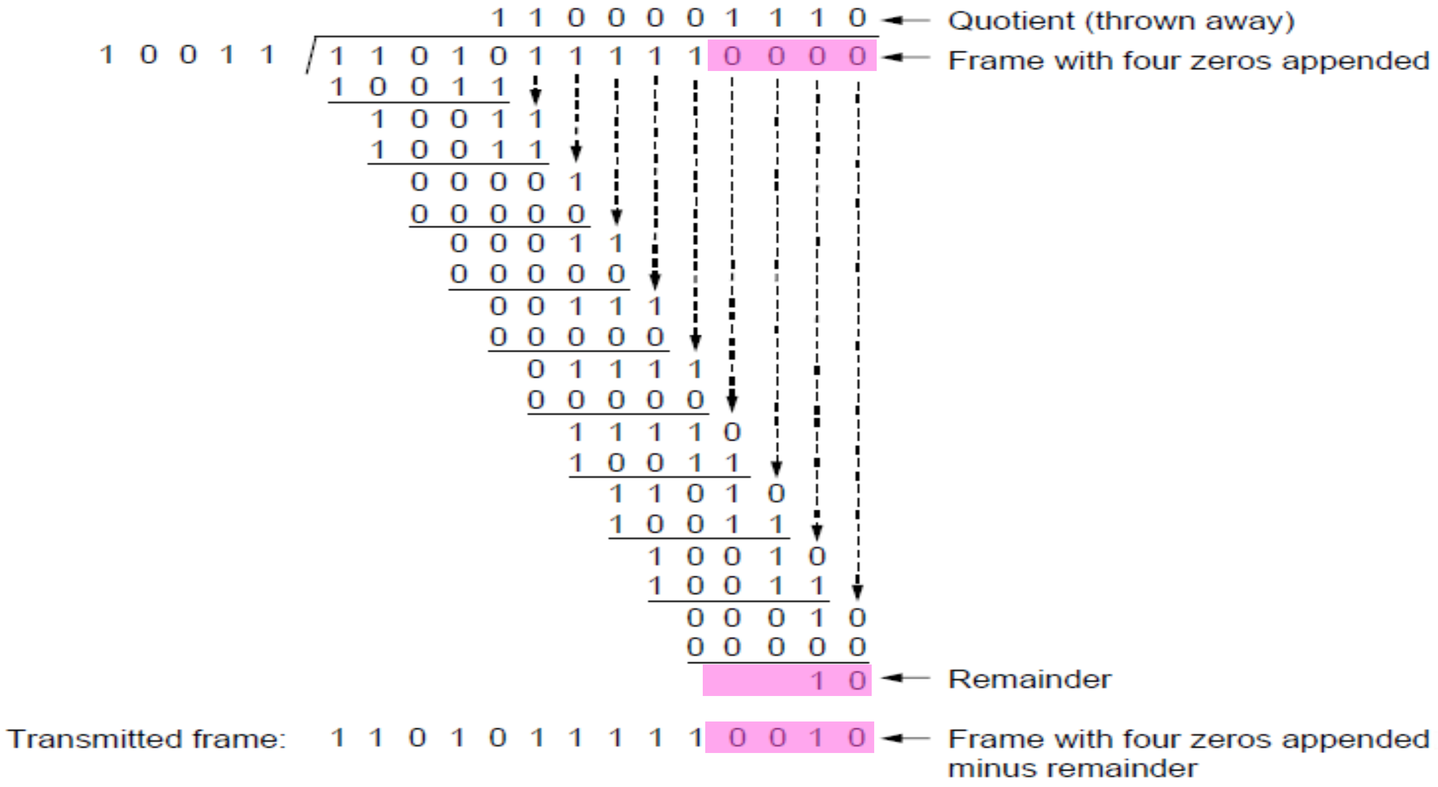
CRCs (4)

Data bits:
1101011111

Check bits:
$C(x)=x^4+x^1+1$
C = 10011
k = 4

$$10011 \overline{)\,1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1}$$

CRCs (5)



1 1 0 0 0 0 1 1 1 0
Quotient (thrown away)
1 0 0 1 1
1 1 0 1 0 1 1 1 1 1 0 0 0 0
Frame with four zeros appended
1 0 0 1 1
1 0 0 1 1
1 0 0 1 1
0 0 0 0 1
0 0 0 0 0
0 0 0 1 1
0 0 0 0 0
0 0 1 1 1
0 0 0 0 0
0 1 1 1 1
0 0 0 0 0
1 1 1 1 0
1 0 0 1 1
1 1 0 1 0
1 0 0 1 1
1 0 0 1 0
1 0 0 1 1
0 0 0 1 0
0 0 0 0 0
1 0
Remainder
Transmitted frame: 1 1 0 1 0 1 1 1 1 1 0 0 1 0
Frame with four zeros appended minus remainder

CRCs (6)

- Protection depend on generator
  - Standard CRC-32 is 10000010 01100000 10001110 110110111
- Properties:
  - HD=4, detects up to triple bit errors
  - Also odd number of errors
  - And bursts of up to k bits in error
  - Not vulnerable to systematic errors like checksums

Error Detection in Practice

- CRCs are widely used on links
  - Ethernet, 802.11, ADSL, Cable …
- Checksum used in Internet
  - IP, TCP, UDP … but it is weak
- Parity
  - Is little used

Topic

- Two strategies to handle errors:

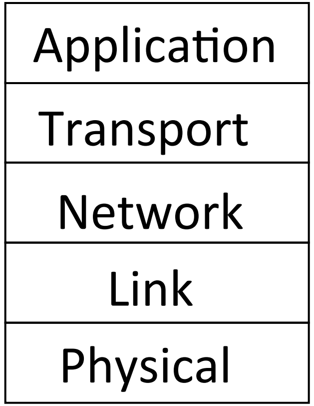
1. Detect errors and retransmit frame (Automatic Repeat reQuest, ARQ)

2. Correct errors with an error correcting code

Done this

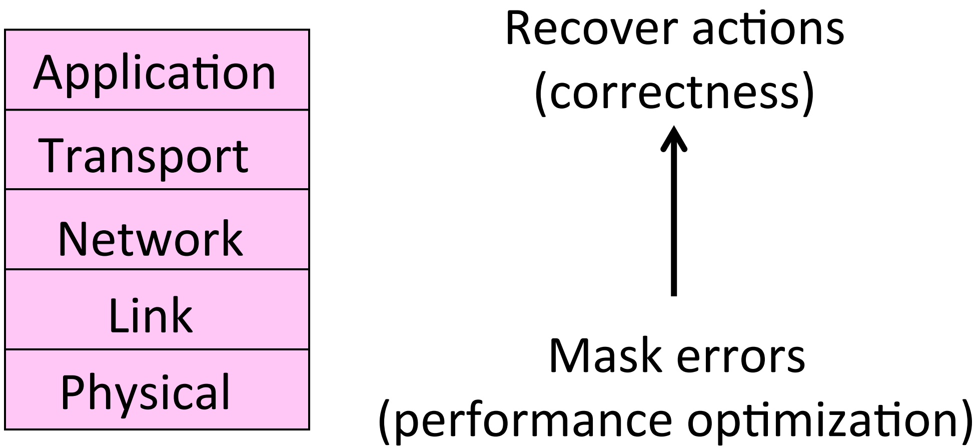
Context on Reliability

- Where in the stack should we place reliability functions?



Context on Reliability (2)

- Everywhere! It is a key issue
  - Different layers contribute differently

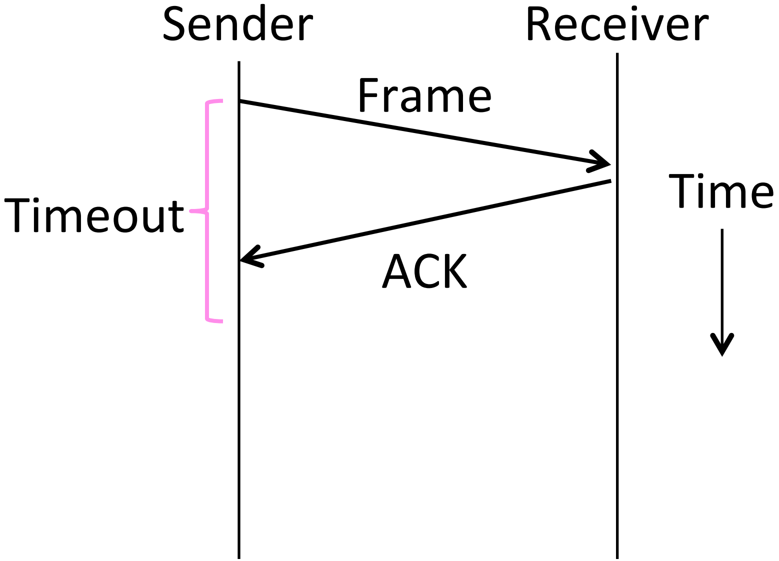


ARQ

- ARQ often used when errors are common or must be corrected
  - E.g., WiFi, and TCP (later)
- Rules at sender and receiver:
  - Receiver automatically acknowledges correct frames with an ACK
  - Sender automatically resends after a timeout, until an ACK is received

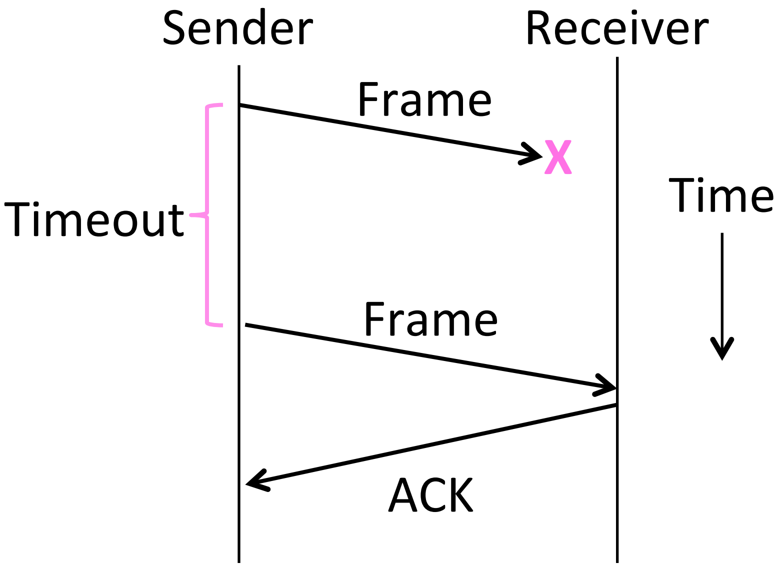
ARQ (2)

- Normal operation (no loss)



ARQ (3)

- Loss and retransmission



So What's Tricky About ARQ?

- Two non-trivial issues:
  - How long to set the timeout? »
  - How to avoid accepting duplicate frames as new frames »
- Want performance in the common case and correctness always

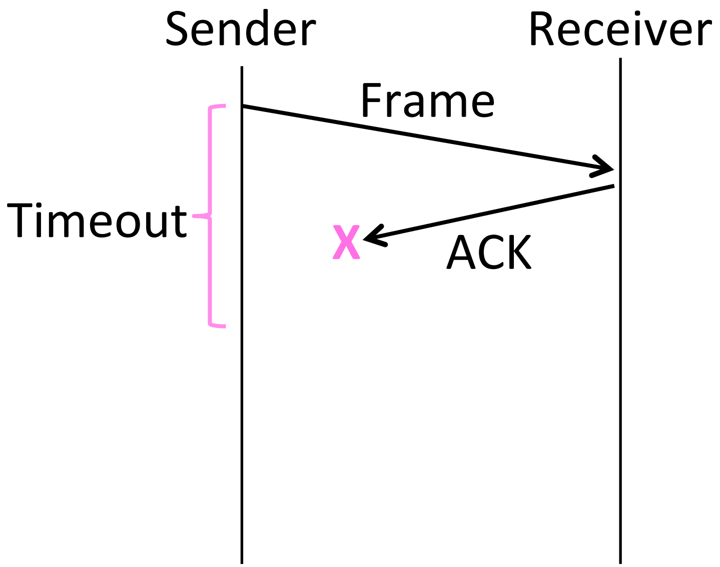
Timeouts

- Timeout should be:
  - Not too big (link goes idle)
  - Not too small (spurious resend)

- Fairly easy on a LAN
  - Clear worst case, little variation
- Fairly difficult over the Internet
  - Much variation, no obvious bound
  - We'll revisit this with TCP (later)

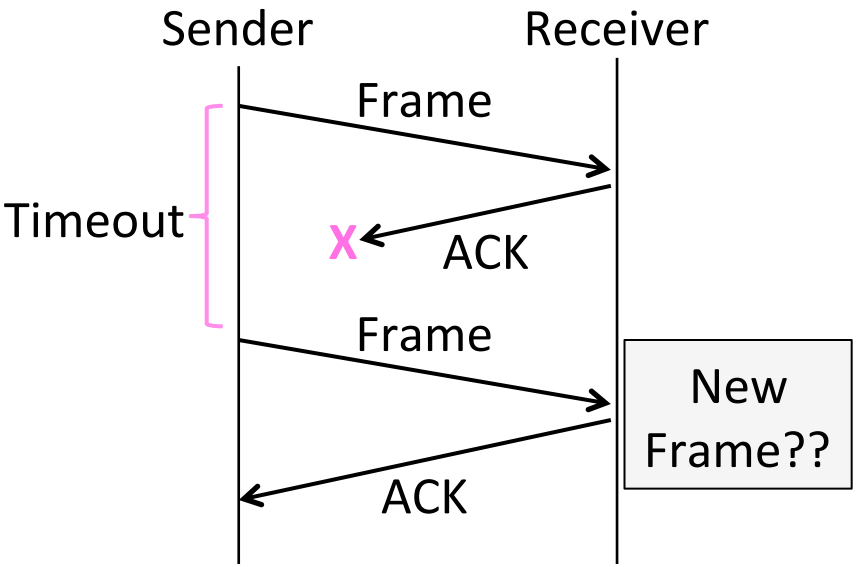
Duplicates

- What happens if an ACK is lost?



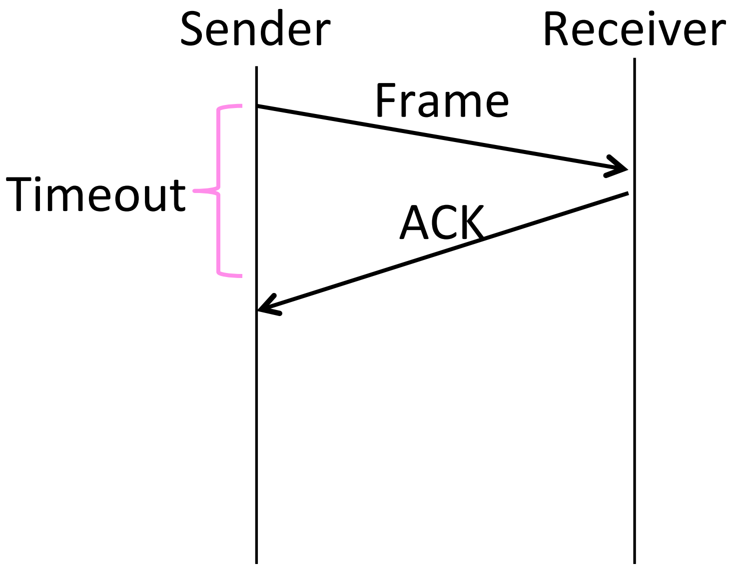
Duplicates (2)

- What happens if an ACK is lost?



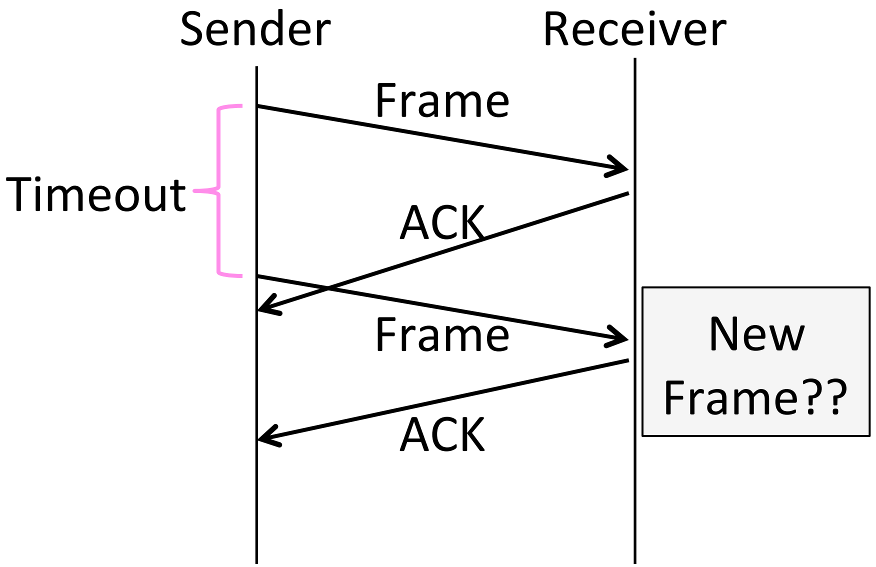
Duplicates (3)

- Or the timeout is early?



Duplicates (4)

- Or the timeout is early?



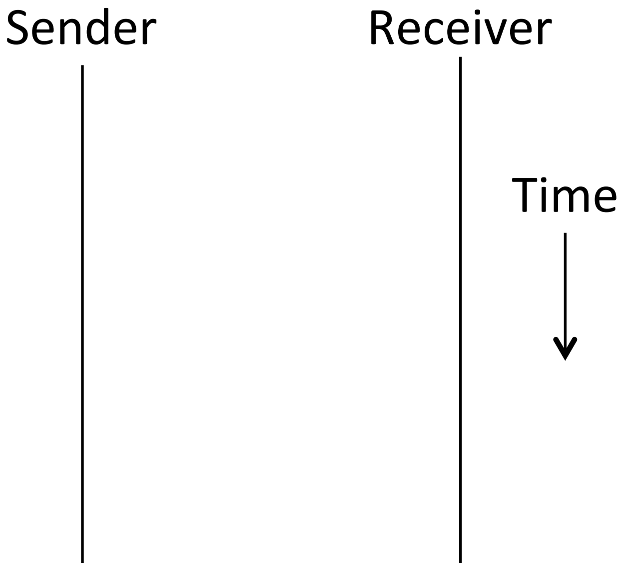
Sequence Numbers

- Frames and ACKs must both carry sequence numbers for correctness

- To distinguish the current frame from the next one, a single bit (two numbers) is sufficient
  - Called Stop-and-Wait

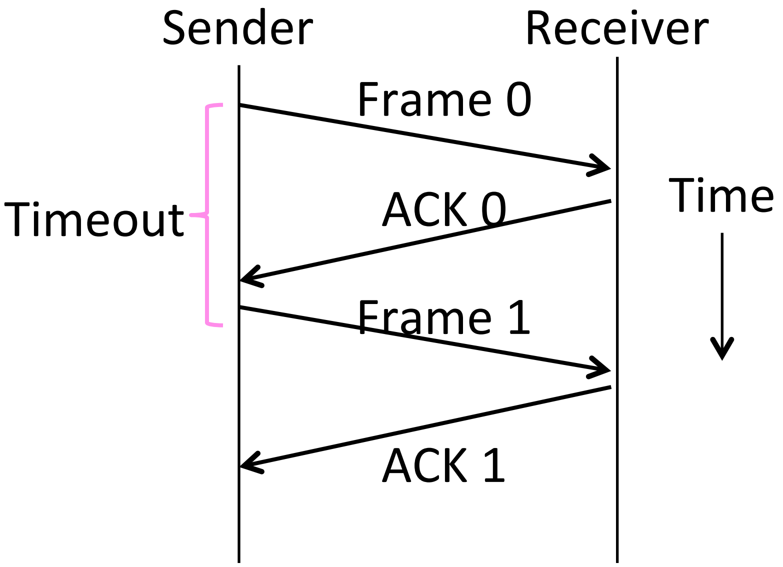
Stop-and-Wait

- In the normal case:



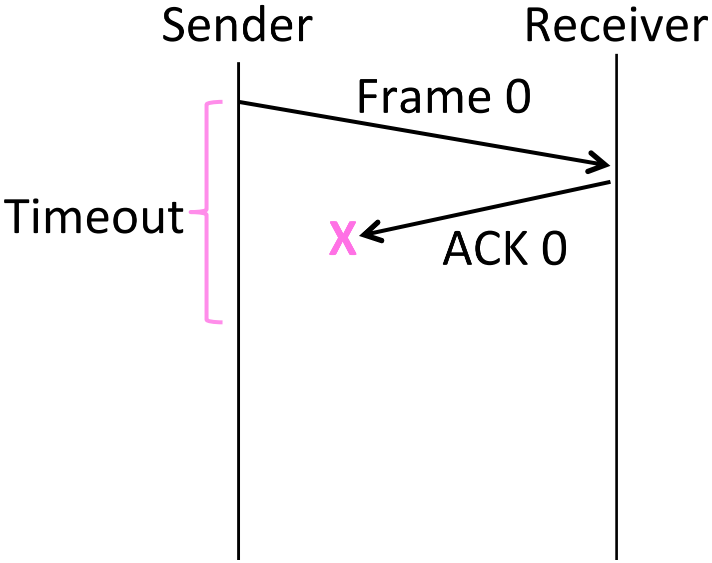
Stop-and-Wait (2)

- In the normal case:



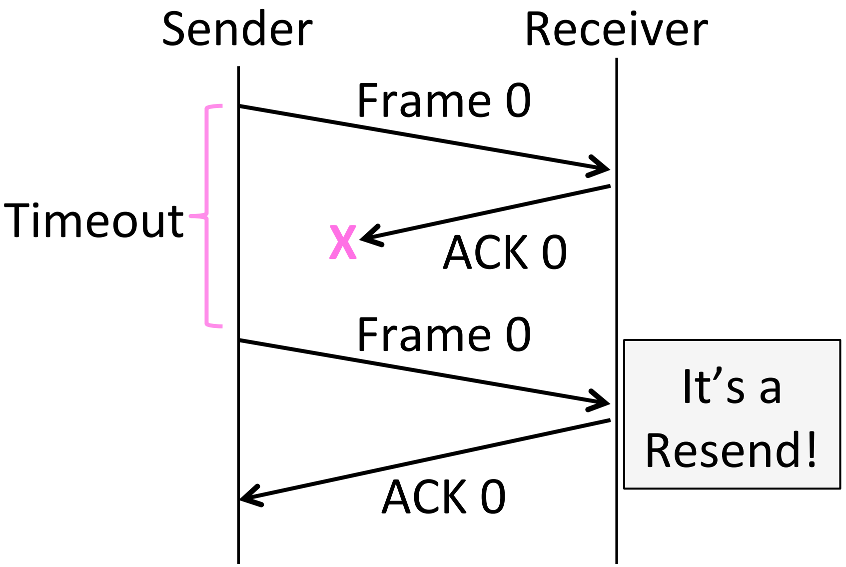
Stop-and-Wait (3)

- With ACK loss:



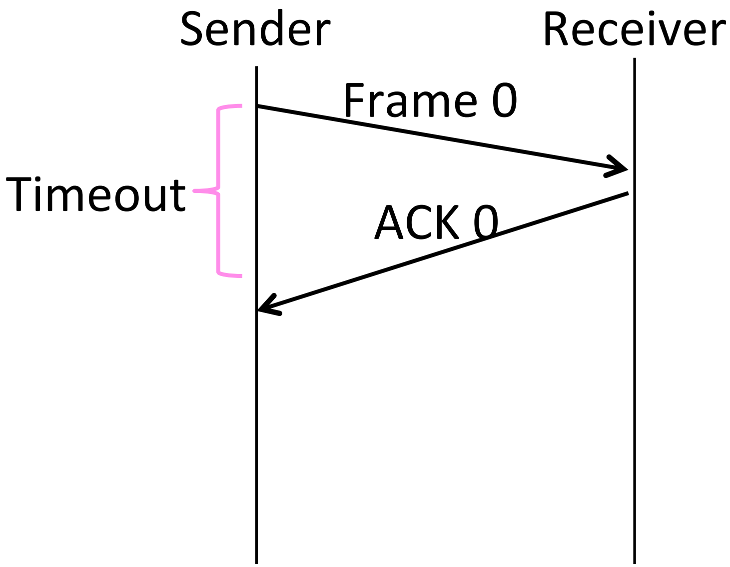
Stop-and-Wait (4)

- With ACK loss:



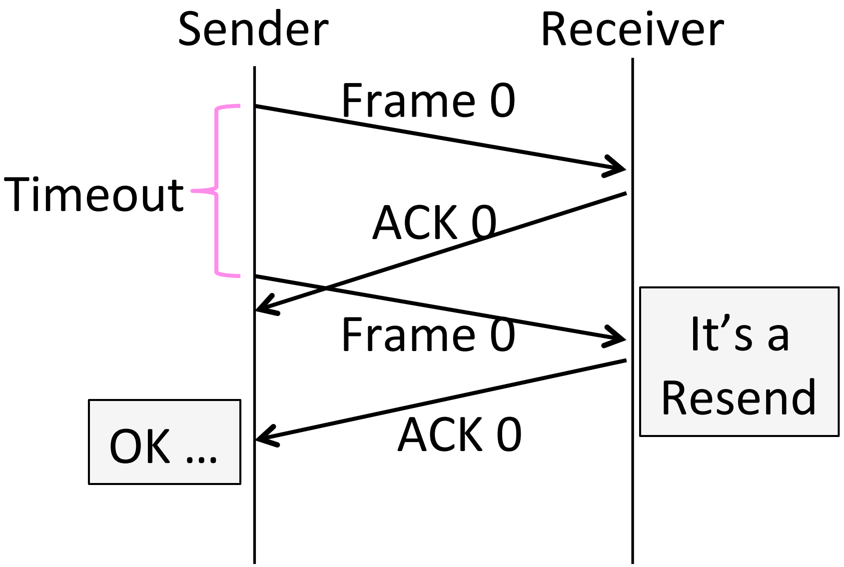
Stop-and-Wait (5)

- With early timeout:



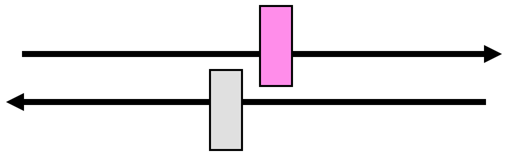
Stop-and-Wait (6)

- With early timeout:



Limitation of Stop-and-Wait

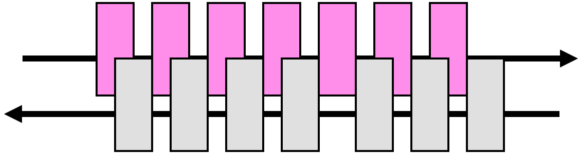
- It allows only a single frame to be outstanding from the sender:
  - Good for LAN, not efficient for high BD



- Ex: R=1 Mbps, D = 50 ms
  - How many frames/sec? If R=10 Mbps?

Sliding Window

- Generalization of stop-and-wait
  - Allows W frames to be outstanding
  - Can send W frames per RTT (=2D)



  - Various options for numbering frames/ACKs and handling loss
    - Will look at along with TCP (later)