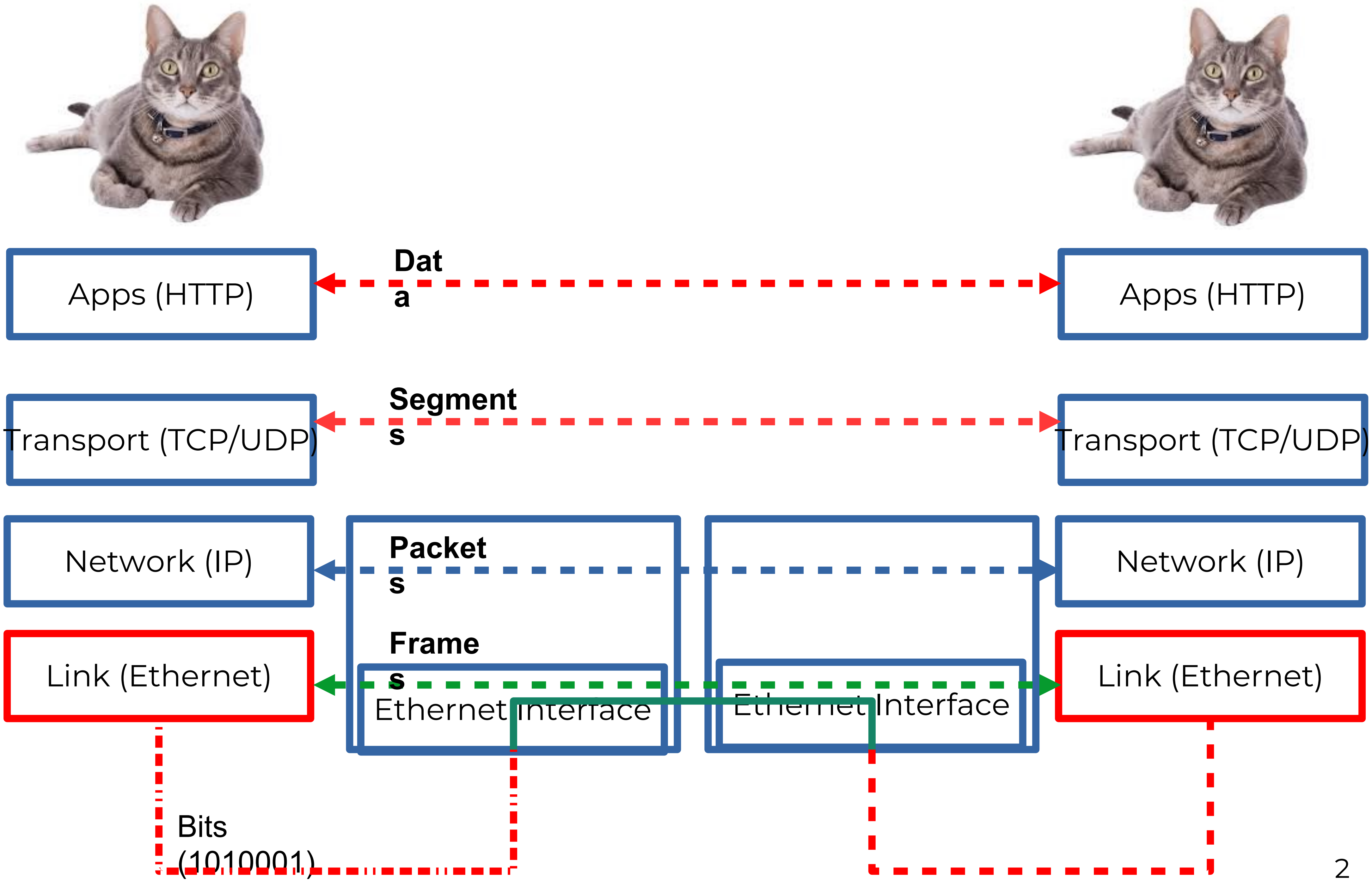
CSC4200/5200 – COMPUTER NETWORKING

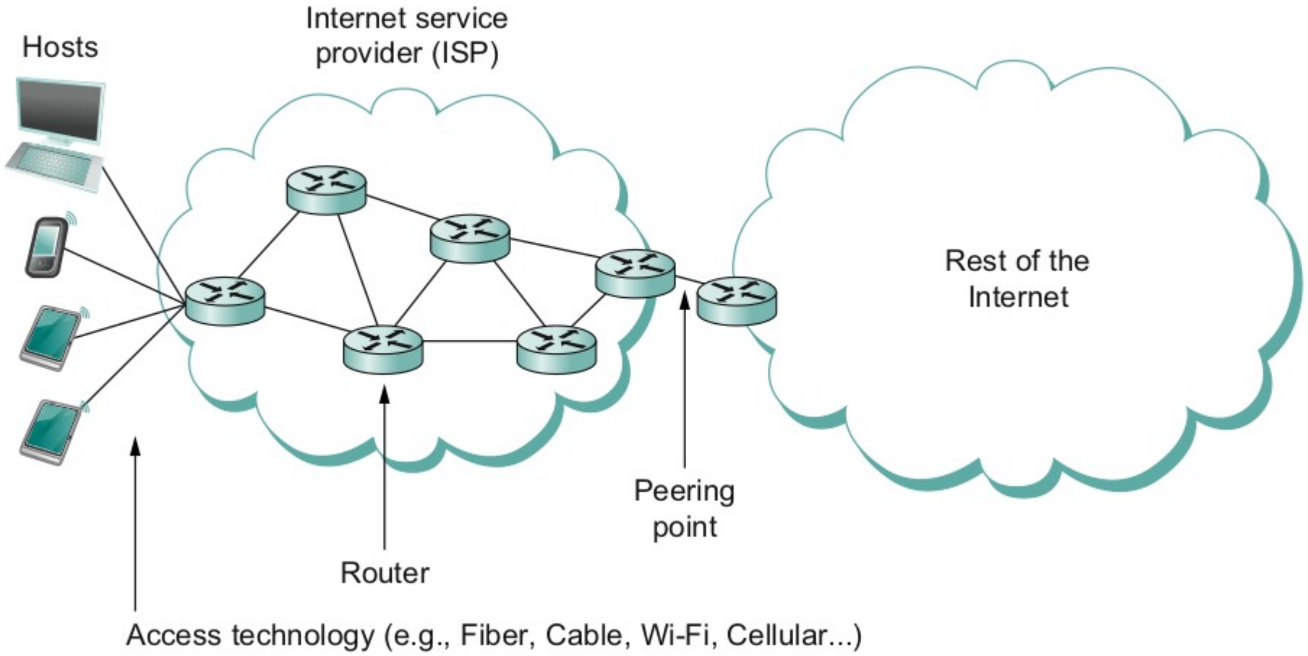
CONNECTING MACHINES TO A NETWORK

Instructor: Susmit Shannigrahi
sshannigrahi@tntech.edu

Tennessee TECH


Apps (HTTP)
Data
Apps (HTTP)
Transport (TCP/UDP)
Segments
Transport (TCP/UDP)
Network (IP)
Packets
Network (IP)
Link (Ethernet)
Frames
Link (Ethernet)
Ethernet Interface
Ethernet Interface
Bits
(1010001)

What does it take to create a link?



- Common abstractions
  - Why?

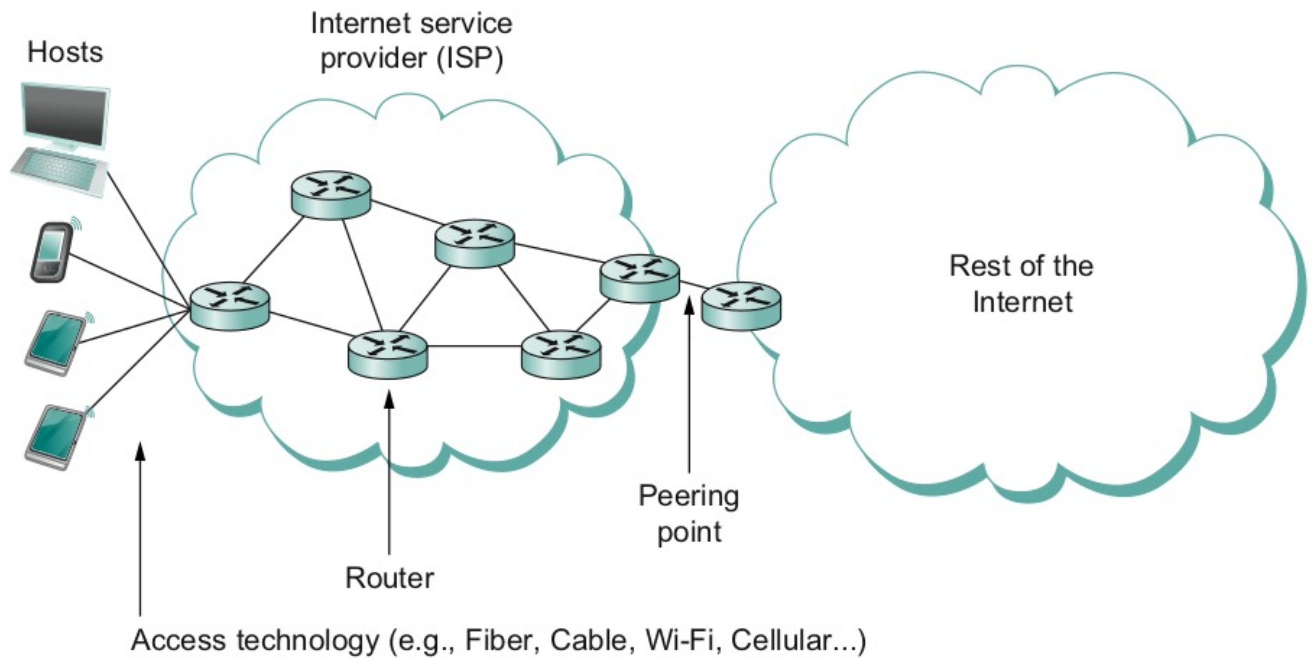
Two Steps to a Link

- Create a physical medium between nodes **(wire, fiber, air!)**

- Make it carry bits

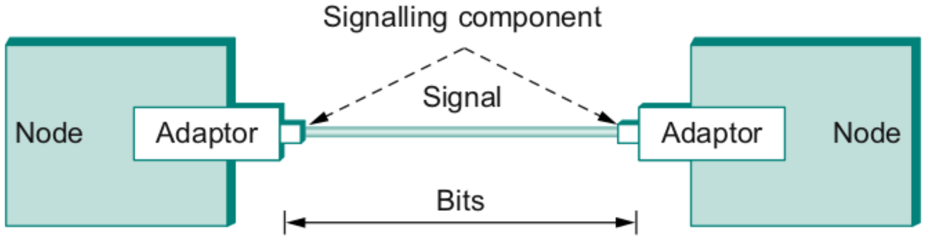
  - **Encoding** bits so that the other end understands **(encoding)**
  - Create bag of bits to create messages (**framing**)
  - Detect errors in frames (**error detection**)
  - Deal with lost frames **(reliable delivery)**
  - Create shared access to link, e.g, WiFi **(media access)**

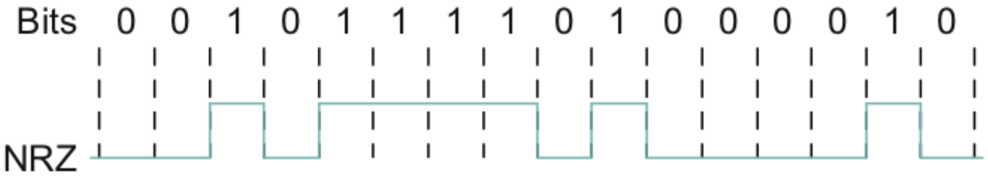
Step 1 - Create the Physical Link



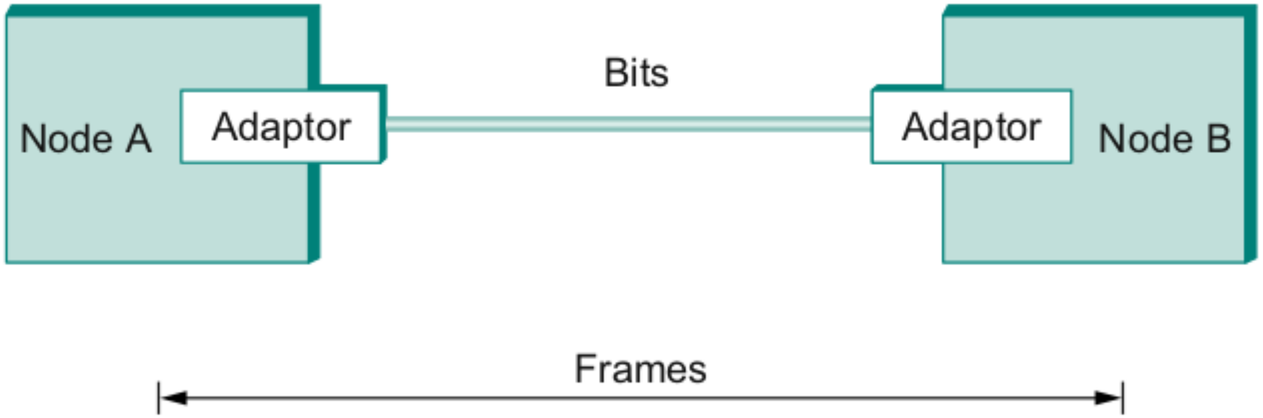
Step 2.1 - Encoding

- Bit pattern - 0101001
  - Must encode it into electrical signals and then decode it on the other end!





Step 2.2 – Create Frames – bag of bits



- Bits - between adaptors
- Frames – between hosts (two computers want to exchange messages)
  - The job of an adaptor is to find frames in a bit sequence

- Frames are link layer protocols

Step 2.2 - Framing

- Point-to-point
  - Special start of text character denoted as Flag
    - 01111110
  - Address, control : default numbers
  - Protocol for demux : IP / IPX
  - Payload : negotiated (1500 bytes)
  - Checksum : for error detection

Step 2.3 - Error Detection

- Bit errors are introduced into frames
  - Because of electrical interference and thermal noises
- Detecting Error
- Correction Error
- Two approaches when the recipient detects an error
  - Notify the sender that the message was corrupted, so the sender can send again.
    - If the error is rare, then the retransmitted message will be error-free
  - Using some error correct detection and correction algorithm, the receiver reconstructs the message

Error Detection

- Common technique for detecting transmission error
  - CRC (Cyclic Redundancy Check)
    - Used in HDLC, DDCMP, CSMA/CD, Token Ring
  - Other approaches
    - Two Dimensional Parity (BISYNC)
    - Checksum (IP)

Error Detection

- Basic Idea of Error Detection
  - To add redundant information to a frame that can be used to determine if errors have been introduced

• 0	• 1	• 0	• 1	• 0	• **0**

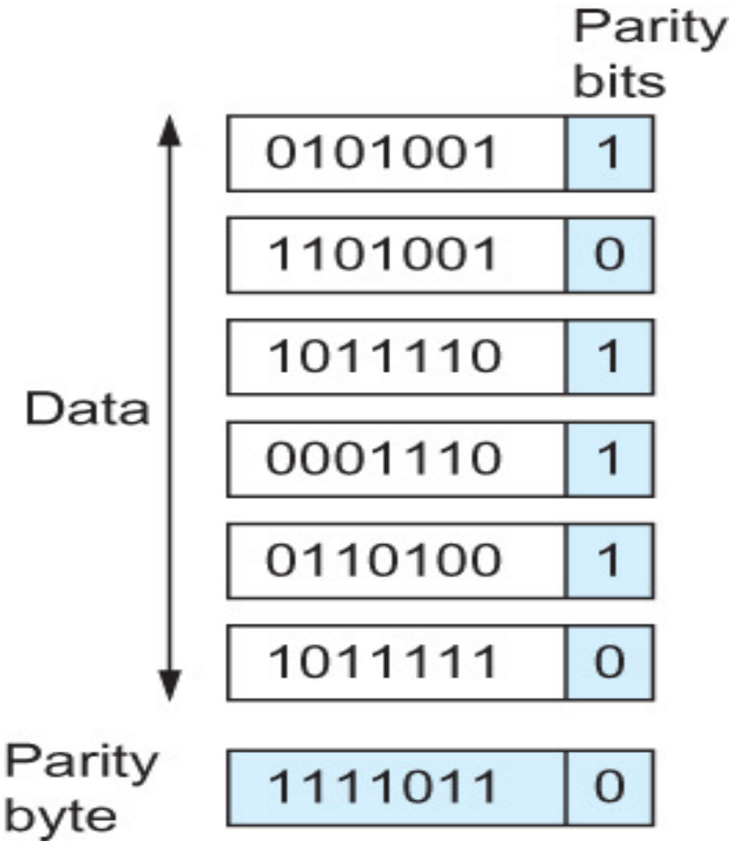
• 0	• 1	• 0	• 1	• 1	• **1**

Number of 1s
- Odd 1s = Parity bit 1
- Even 1s = Parity bit 0

Two-dimensional parity

- Two-dimensional parity does a similar calculation
- Extra parity byte for the entire frame, in addition to a parity bit for each byte
- Two-dimensional parity catches all 1-, 2-, and 3-bit errors and most 4-bit errors

Two-dimensional parity



Two Dimensional Parity

Number of 1s

- Odd 1s = Parity bit 1
- Even 1s = Parity bit 0

Do it both horizontally and vertically

Internet Checksum Algorithm

- Not used at the link level
- Add up all the words that are transmitted and then transmit the result of that sum
  - The result is called the checksum
- The receiver performs the same calculation on the received data and compares the result with the received checksum

Internet Checksum Algorithm

Server Side

1. It treats segment contents as sequence of 16-bit integers.
2. All segments are added. Let's call it sum.
3. Checksum : 1's complement of sum.(In 1's complement all 0s are converted into 1s and all 1s are converted into 0s).
4. Sender puts this checksum value in UDP checksum field.

Client Side:

1. Calculate checksum
2. All segments are added and then sum is added with sender's checksum.
3. Check that any 0 bit is presented in checksum. If receiver side checksum contains any 0 then error is detected. So the packet is discarded by receiver.

Internet Checksum Algorithm (RFC 1071)

- Calculate the checksum for **0110011001100110**0101010101010101**0000111100001111**

Break it into 16 bit integers.

- A = 0110011001100110
- B = 0101010101010101

A+B = 1011101110111011

- C = 0000111100001111

1100101011001010 (sum of all segments)

0011010100110101 (1's complement, 1→0, 0→ 1) <= this is the checksum

At receiver:

Add sum of all segments and checksum
1100101011001010

+0011010100110101

Others - Cyclic Redundancy Check (CRC)

- Reduce the number of extra bits and maximize protection
- N+1 bit message is N degree polynomial

  10011010 →

  $Msg(x)=(1\times x^7)+(0\times x^6)+(0\times x^5)+(1\times x^4)+(1\times x^3)+(0\times x^2)+(1\times x^1)+(0\times x^0)$
- $Msg(x)=x^7+x^4+x^3+x^1$

Others - Cyclic Redundancy Check (CRC)

- $Msg(x)=x^7+x^4+x^3+x^1$

- Pick a divisor polynomial (from a table)

  $C(x) = x^{3+}x^2+1$

- Divide M(X) by C(x) → subtract the reminder from M(x)
  - Gives you M’(X)
  - You can do this by performing a logical XOR
- Send M’(x)and C(x) to the recipient
  - If the result is 0, you received a good copy

Others - Cyclic Redundancy Check (CRC)

- $Msg(x)=x^7+x^4+x^3+x^1$

- Pick a divisor polynomial (from a table)

  $C(x) = x^{3+}x^2+1$

- Divide M(X) by C(x) → subtract the reminder from M(x)
  - Gives you M’(X)
  - You can do this by performing a logical XOR
- Send M’(x)and C(x) to the recipient
  - If the result is 0, you received a good copy

Others - Cyclic Redundancy Check (CRC)

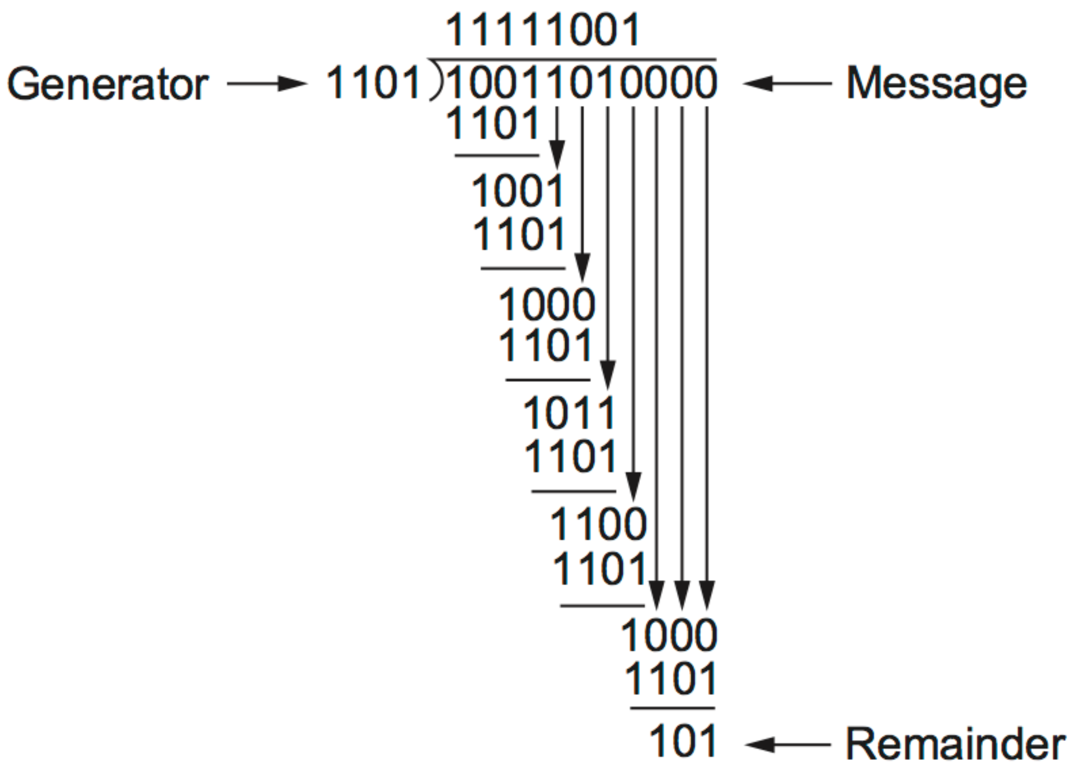
1. Msg(x)=10011010 = $x^7+x^4+x^3+x^1$

2. add k zeros at the end of the message, 3 in this case.
10011010000 ← T(x)

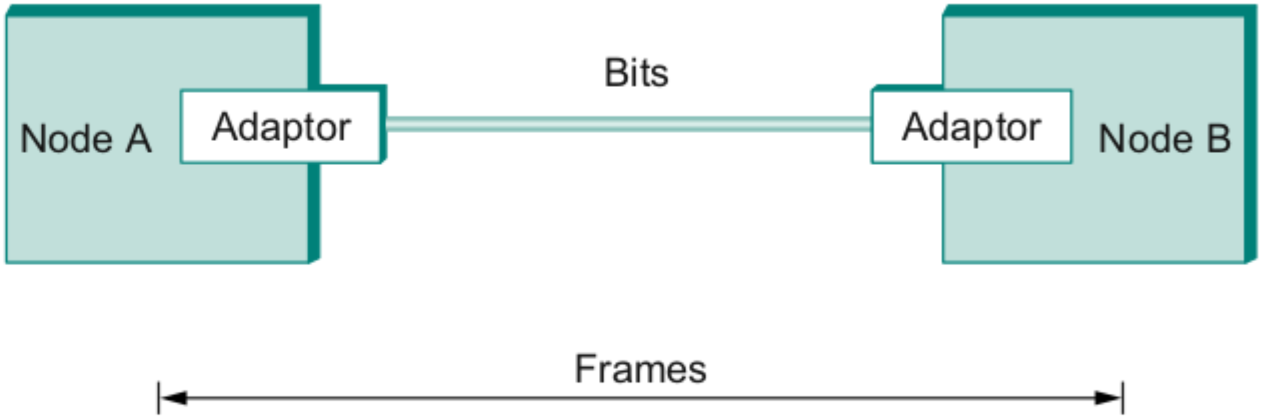
3. Pick a c(x) → x^3+x^2+1.

4. T(x)/c(x) → Reminder 101.

5. Subtract from message and send



Frames



- We are still sending frames between hosts!
- Shortcomings of error correction/detection?

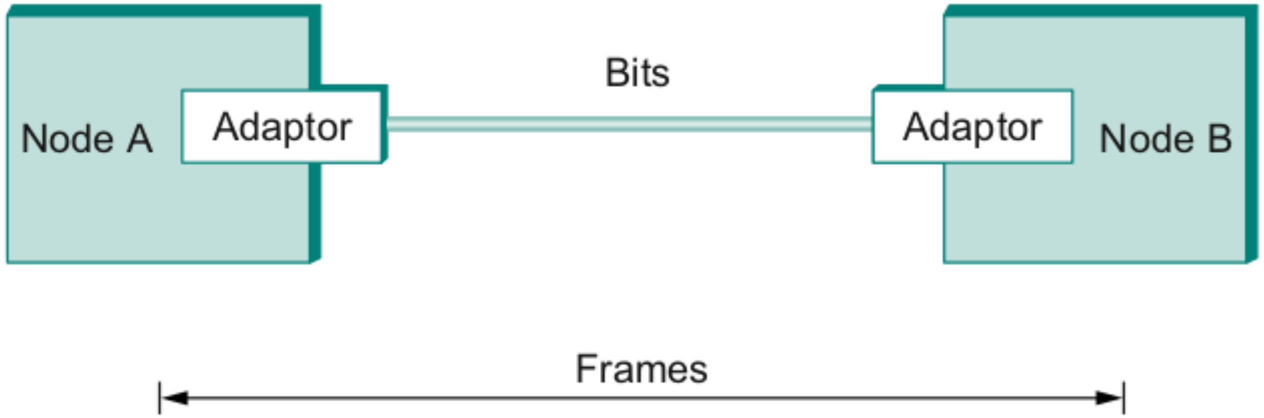
Step 2.4 - Reliable Delivery

- Frames might get lost
  - Too many bits lost
  - Clock did not sync properly
  - Error detected but the report got lost

- Can we build links that does not have errors?
  - Not possible

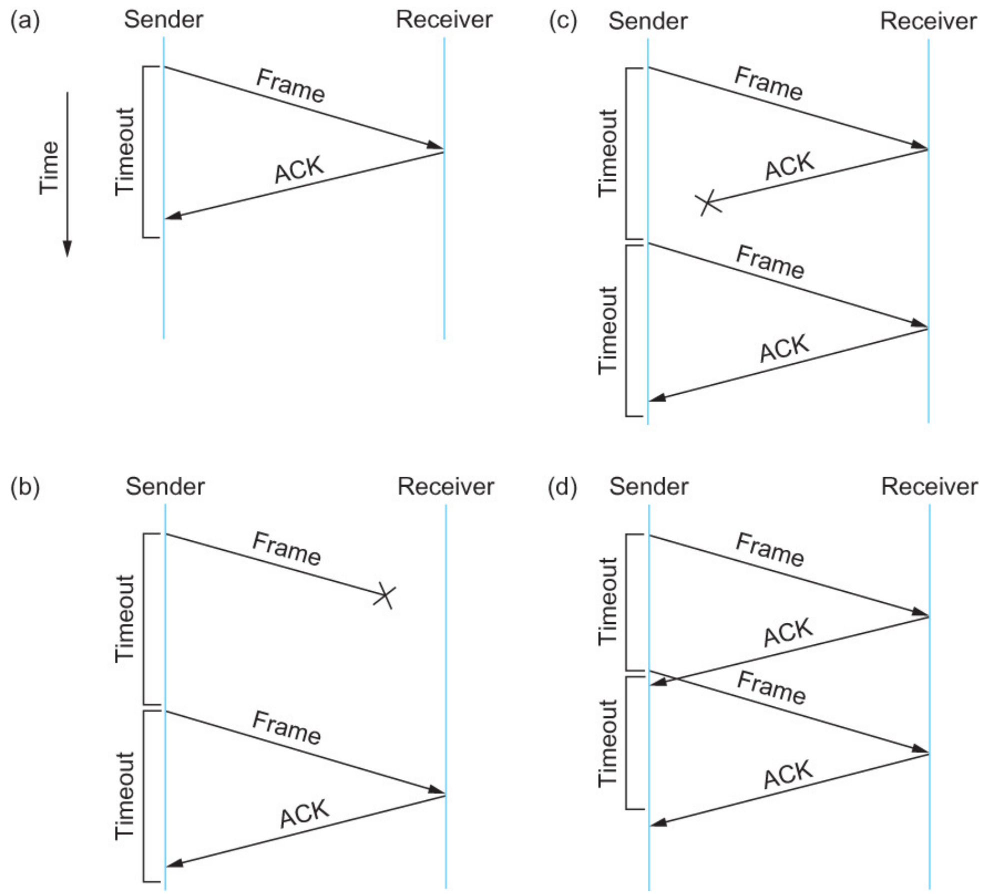
- How about all those error correction stuff we learned?
  - Can we add them to frames?
  - We could, but think of the overhead
  - What happens when the entire frame is lost?

Frames – bag of bits



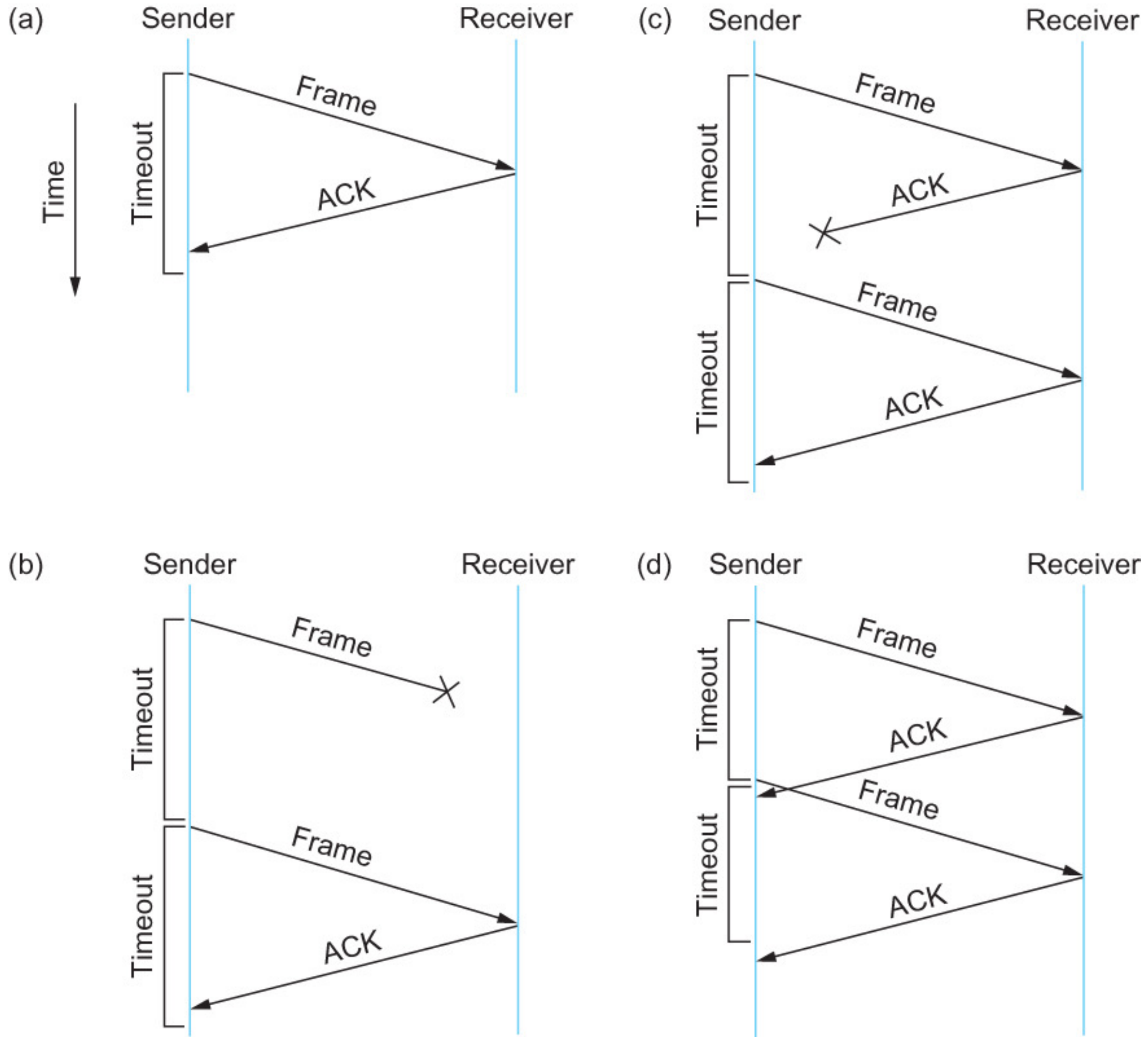
- Sending side – encapsulation, add error check bits, flow control
- Receiving side – extract frames, check for error, flow control

Stop and Wait



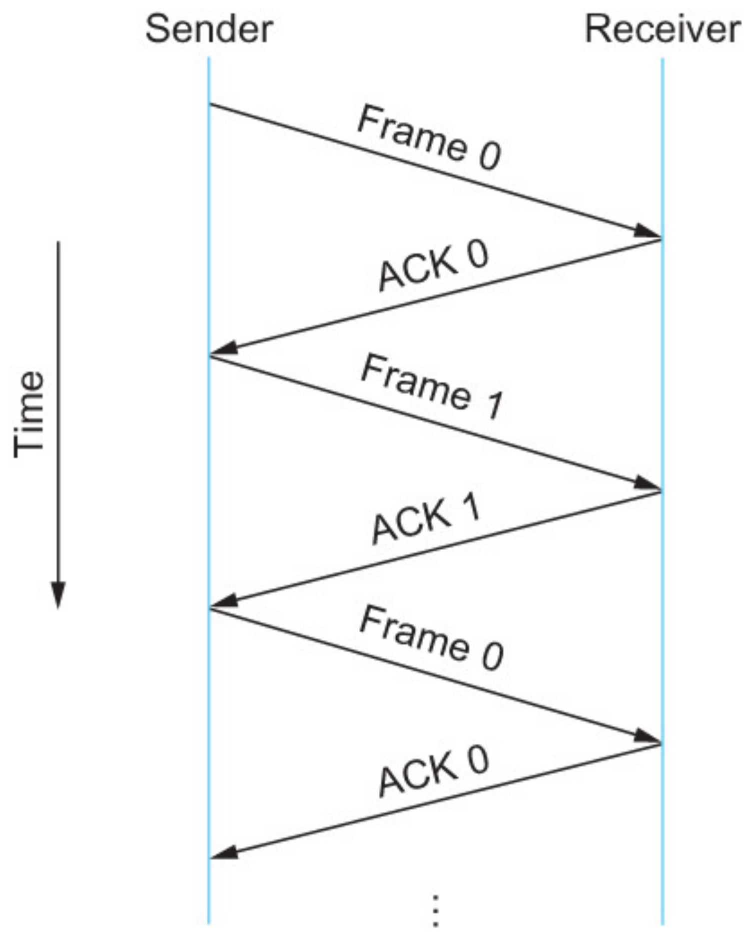
- Sender sends a frame, sets a timeout (e.g., 1 sec)
- Receiver receives the frame,
- sends an ACK
- Sender
  - sends the next frame on ACK
  - retransmits the same frame if timeout happens
- **Spot the bugs in the protocol**

Stop and Wait – Bugs (C and D)

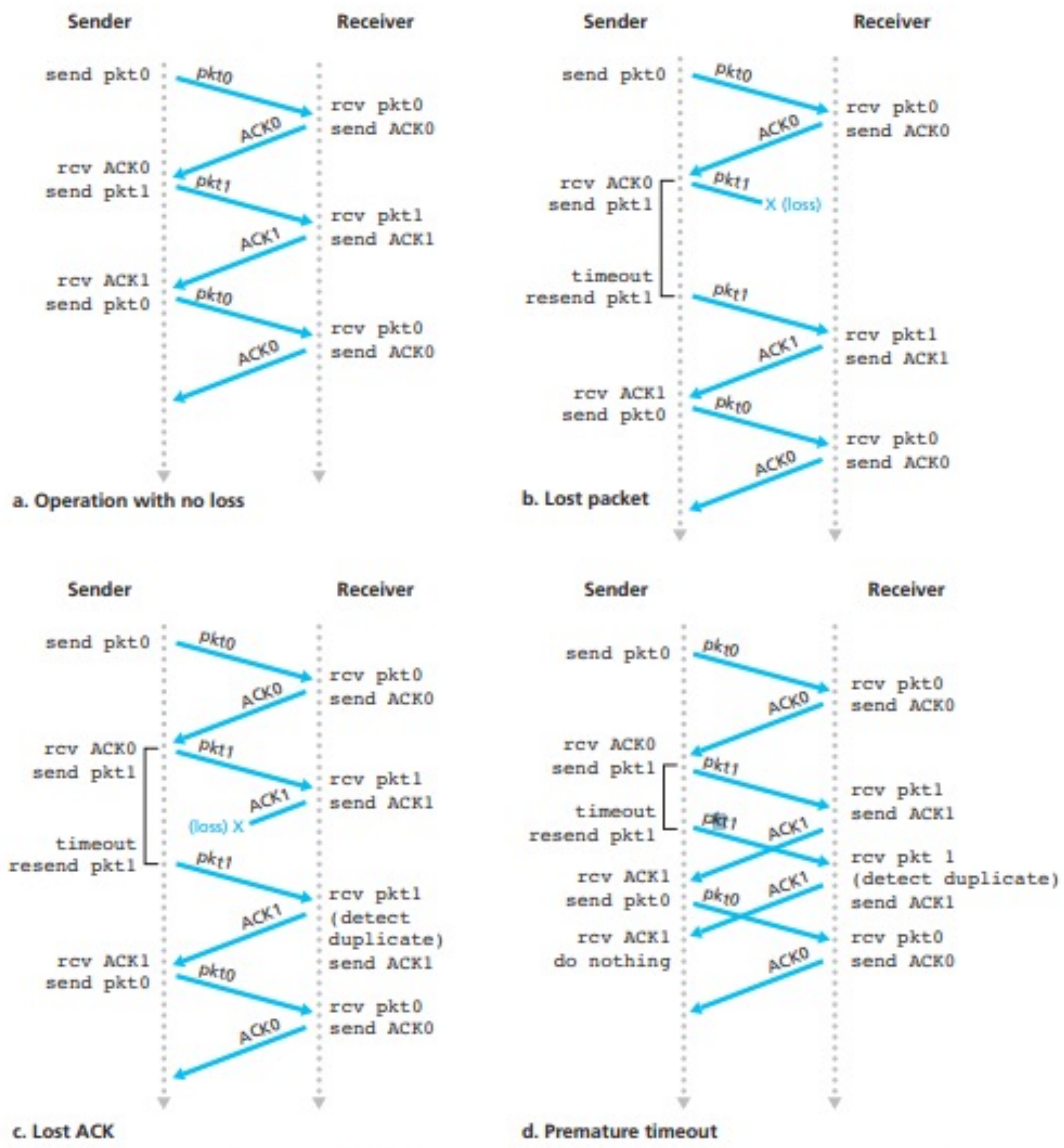


Stop and Wait – How to fix the bug?

Hint: Uniquely identify each packet



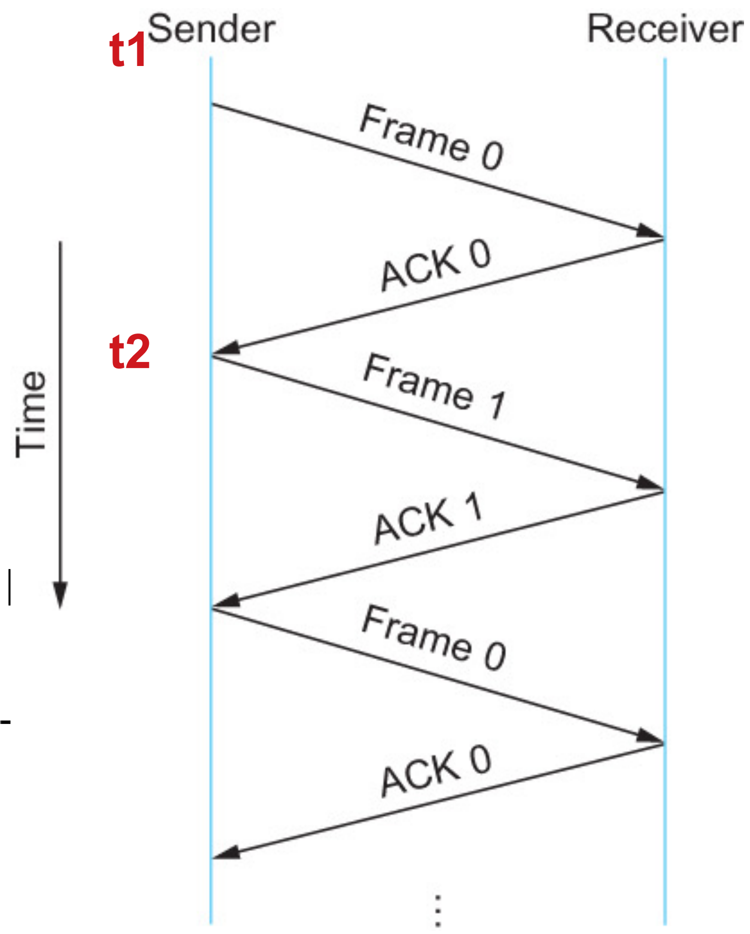
Stop and Wait v2



Stop and Wait - V2 Problems

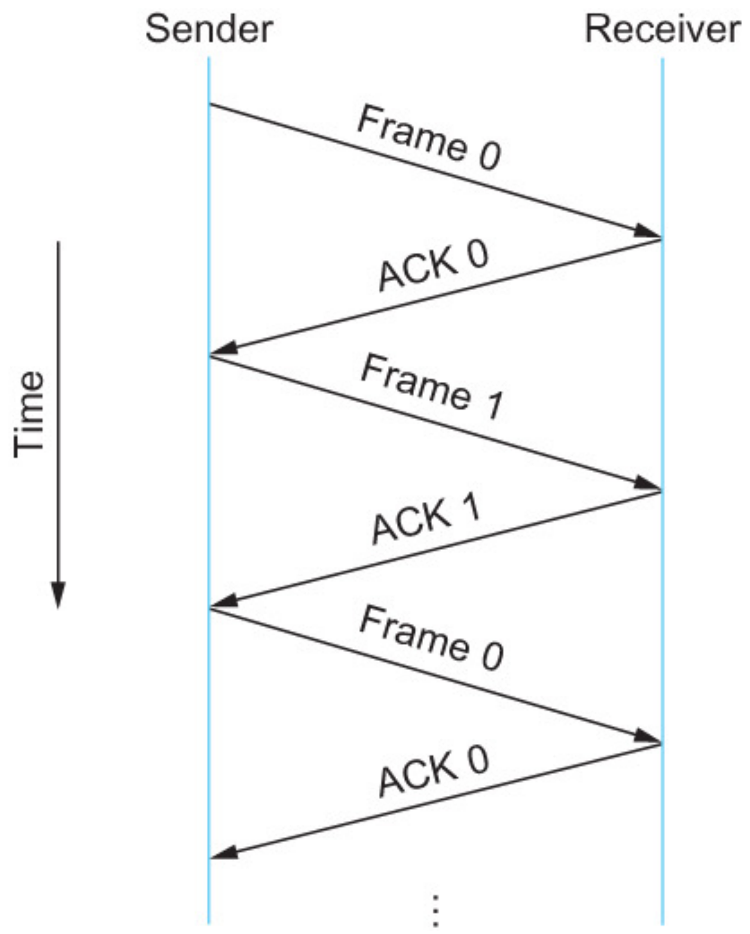
- Sender sets a timeout to wait for an ACK
  - Too small – retransmissions
  - Too large – long wait if frames are lost

- Solution:
  - Keep a running average of Round Trip Ti
  - EstimatedRTT = $(1 - \alpha) \cdot$ EstimatedRTT + $\alpha \cdot$ SampleRT
  - Timeout = 2*EstimatedRTT
  - Value of $\alpha = 0.125$
  - Where does α come from? RFC 6928 (for now)



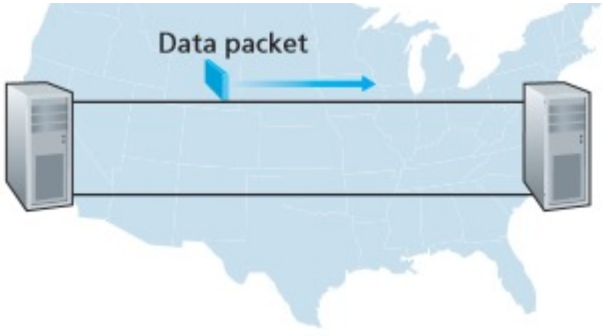
Stop and Wait – How to fix the bug?

Hint: Uniquely identify each packet



Stop and Wait – How does it perform?

- Bandwidth (R)= 1Gbps
- Packet size (L) = 1000 bytes
- RTT = 30ms
- T_{trans} = L/R = 8000bits/10^9bits/sec = 8microsecond
- T_{prop} = 15ms
- Total Delay = 15.008 ms

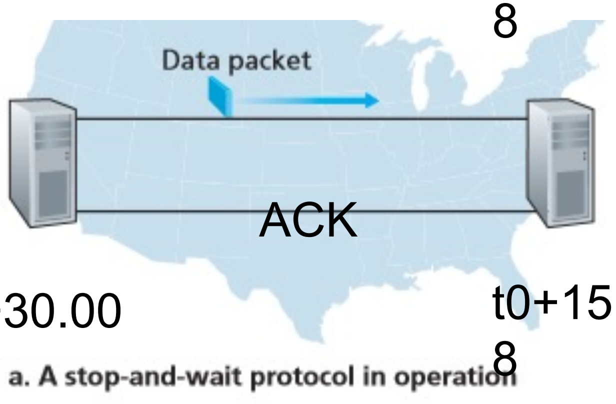


a. A stop-and-wait protocol in operation

Kurose/Ross

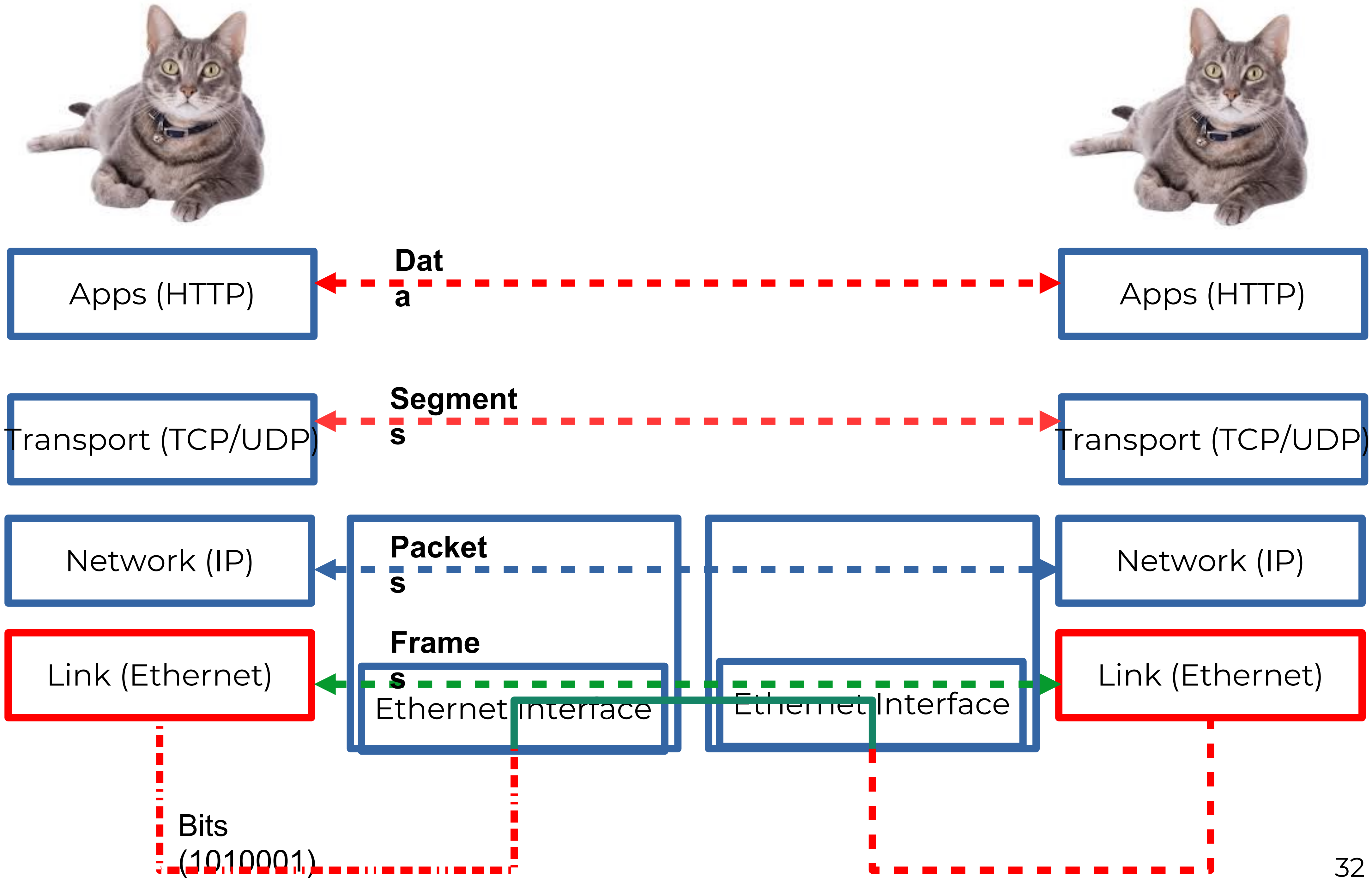
Stop and Wait – How does it perform?

- Sender transmits for only 0.008 ms in 30.008ms
- Utilization = 0.008/30.008 = 0.00027
- One bit at a time
- Worse when loss happens!



a. A stop-and-wait protocol in operation

Kurose/Ross



Apps (HTTP)
Dat
a
Apps (HTTP)
Transport (TCP/UDP)
Segment
s
Transport (TCP/UDP)
Network (IP)
Packet
s
Network (IP)
Link (Ethernet)
Frame
s
Ethernet Interface
Ethernet Interface
Link (Ethernet)
Bits
(1010001)

Reading Assignment

- Chapter 2.4 –Error detection and CRC-
  - Until Sliding window
  - Pay special attention to CRC