

Link Layer

Previously, on PHY

Claude Shannon (1916-2001)

- Father of information theory
 - “A Mathematical Theory of Communication”, 1948
- Fundamental contributions to digital computers, security, and communications

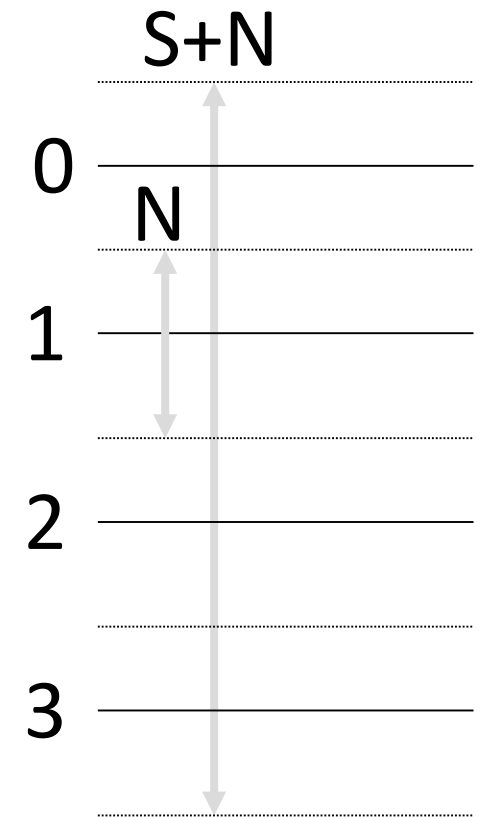
Electromechanical mouse
that “solves” mazes! →



Credit: Courtesy MIT Museum

Shannon Capacity

- How many levels we can distinguish depends on S/N
 - Or SNR, the Signal-to-Noise Ratio
 - Note noise is random, hence some errors
- SNR given on a log-scale in decibels:
 - $\text{SNR}_{\text{dB}} = 10\log_{10}(S/N)$



Shannon Capacity (2)

- Shannon limit is for capacity (C), the maximum information carrying rate of the channel:

$$C = B \log_2(1 + S/N) \text{ bits/sec}$$

Wired/Wireless Perspective (2)

- Wires, and Fiber
 - Engineer link to have requisite SNR and B
 - Can fix data rate

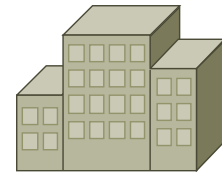
Engineer SNR for data rate

- Wireless
 - Given B, but SNR varies greatly, e.g., up to 60 dB!
 - Can't design for worst case, must adapt data rate

Adapt data rate to SNR

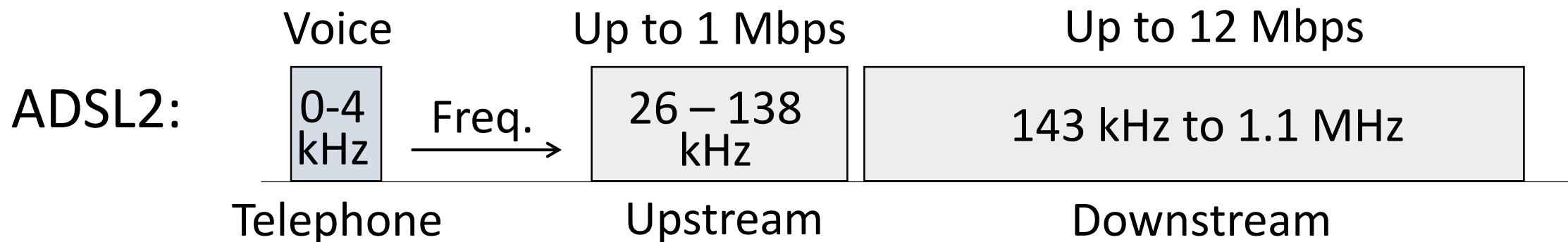
Putting it all together – DSL

- DSL (Digital Subscriber Line, see §2.6.3) is widely used for broadband; many variants offer 10s of Mbps
 - Reuses twisted pair telephone line to the home; it has up to ~ 2 MHz of bandwidth but uses only the lowest ~ 4 kHz



DSL (2)

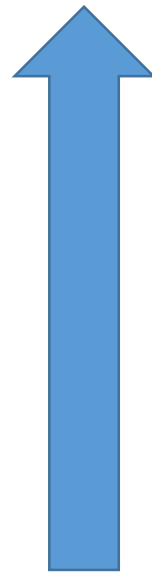
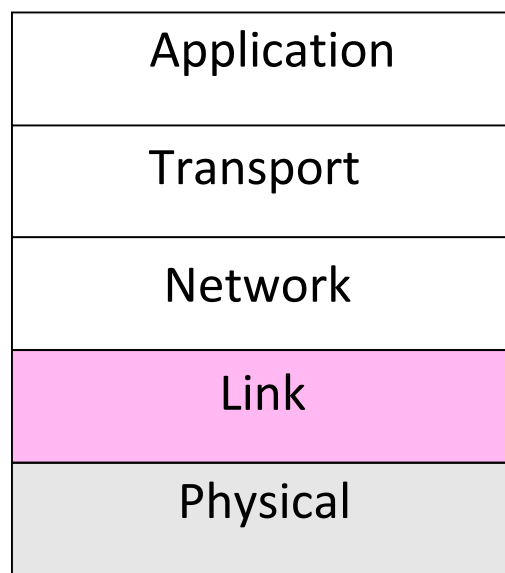
- DSL uses passband modulation (called OFDM)
 - Separate bands for upstream and downstream (larger)
 - Modulation varies both amplitude and phase (QAM)
 - High SNR, up to 15 bits/symbol, low SNR only 1 bit/symbol



Link Layer

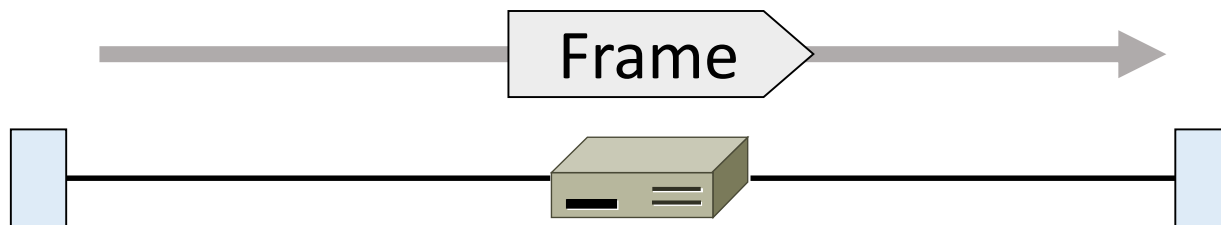
Where we are in the Course

- Moving on up to the Link Layer!

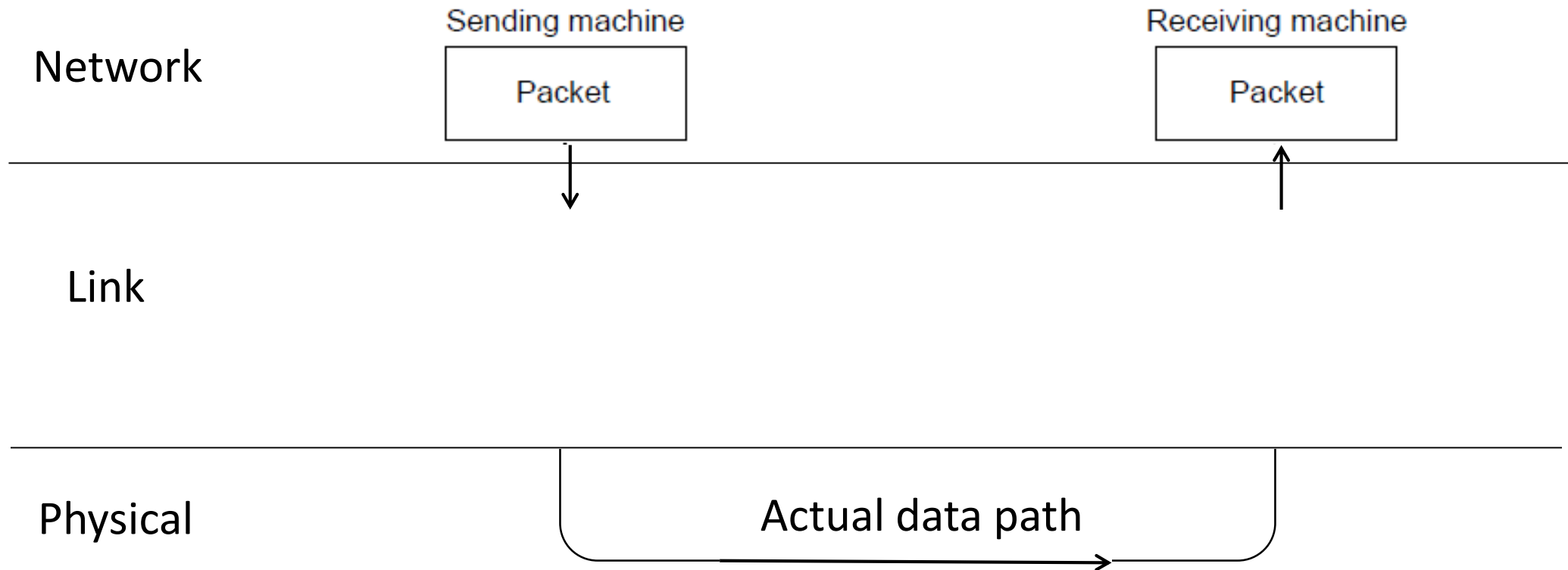


Scope of the Link Layer

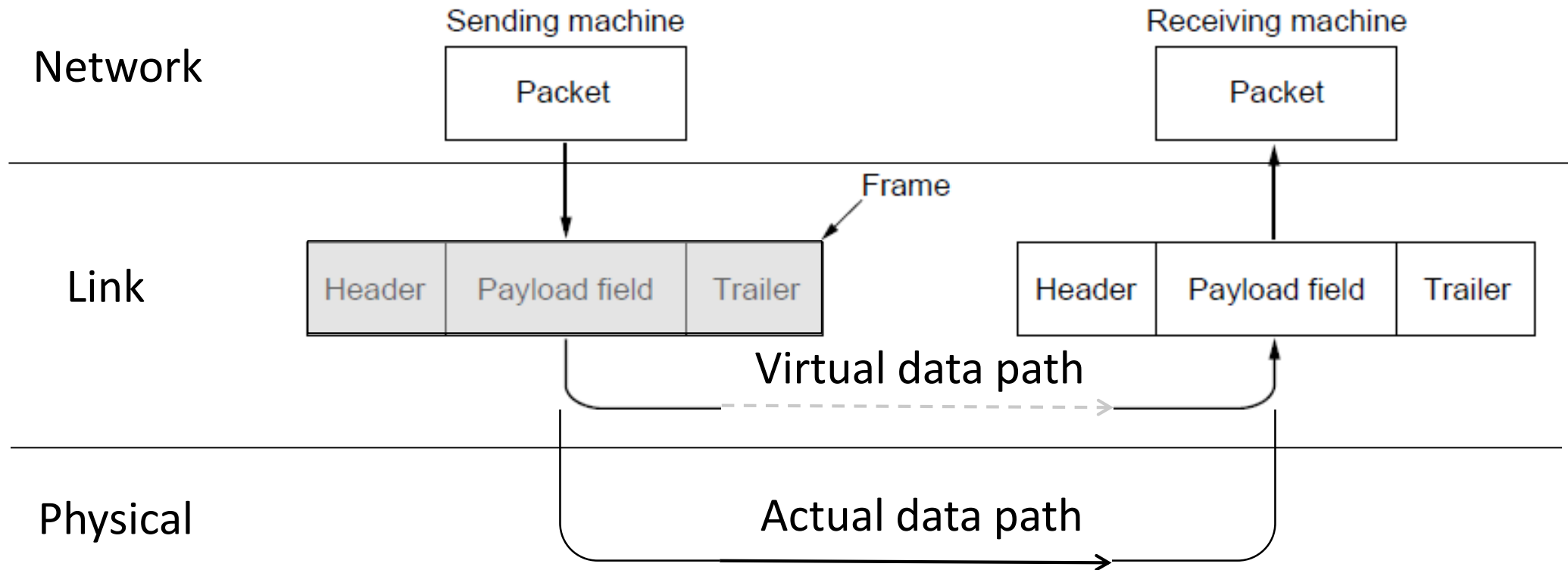
- Concerns how to transfer messages over one or more connected links
 - Messages are frames, of limited size
 - Builds on the physical layer



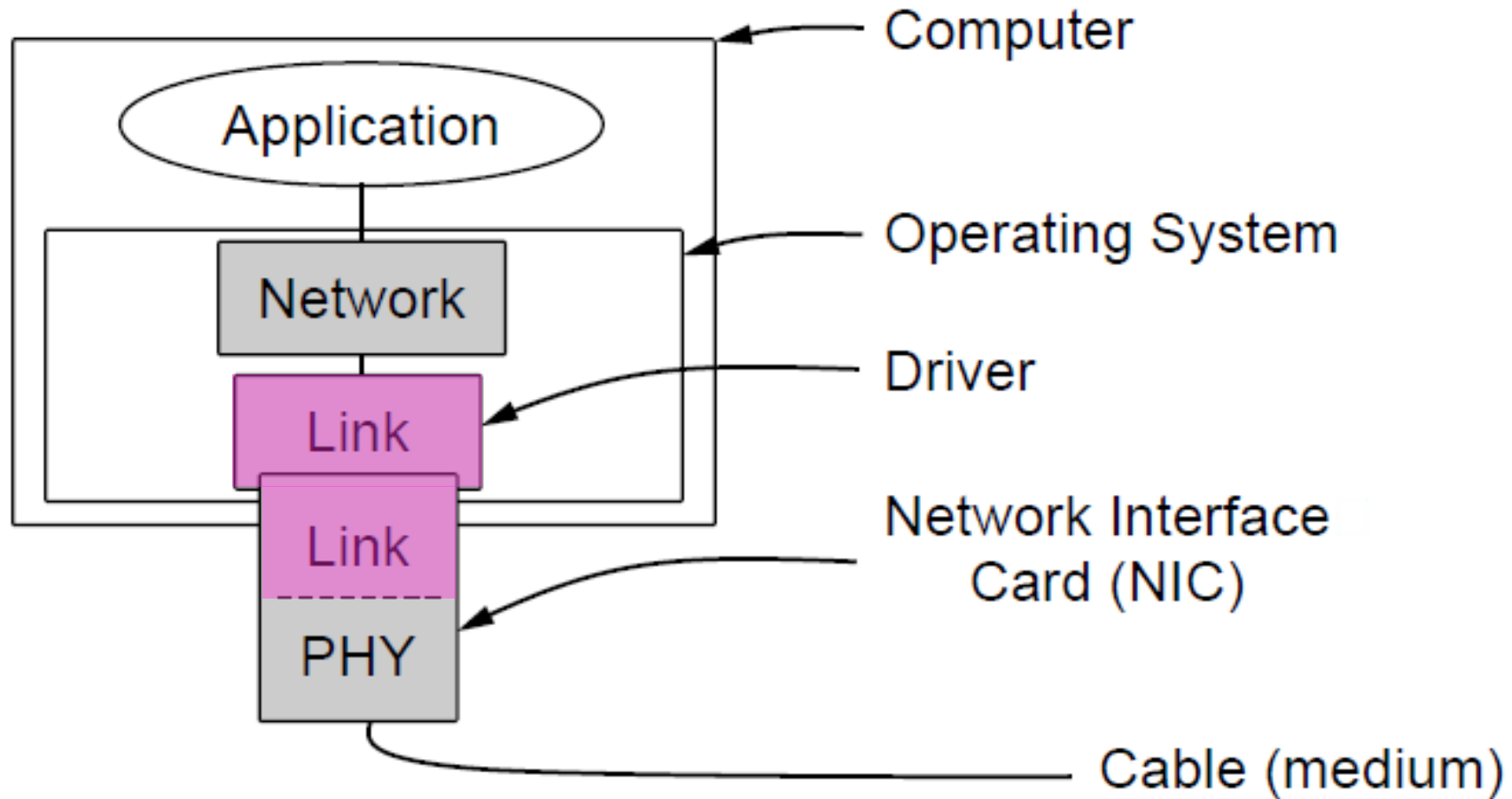
In terms of layers ...



In terms of layers (2)



Typical Implementation of Layers (2)



Topics

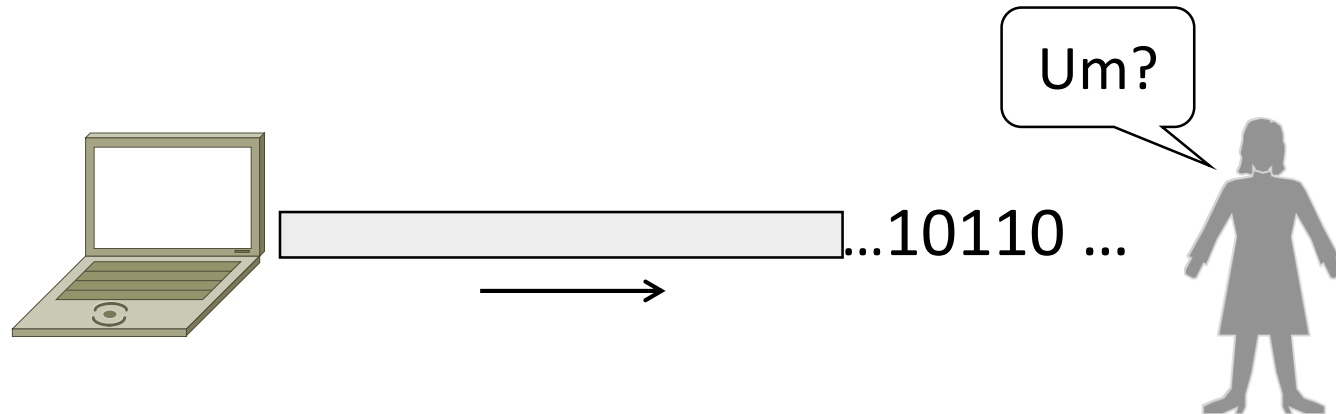
1. Framing
 - Delimiting start/end of frames
2. Error detection and correction
 - Handling errors
3. Retransmissions
 - Handling loss
4. Multiple Access
 - 802.11, classic Ethernet
5. Switching
 - Modern Ethernet

Framing

Delimiting start/end of frames

Topic

- The Physical layer gives us a stream of bits. How do we interpret it as a sequence of frames?



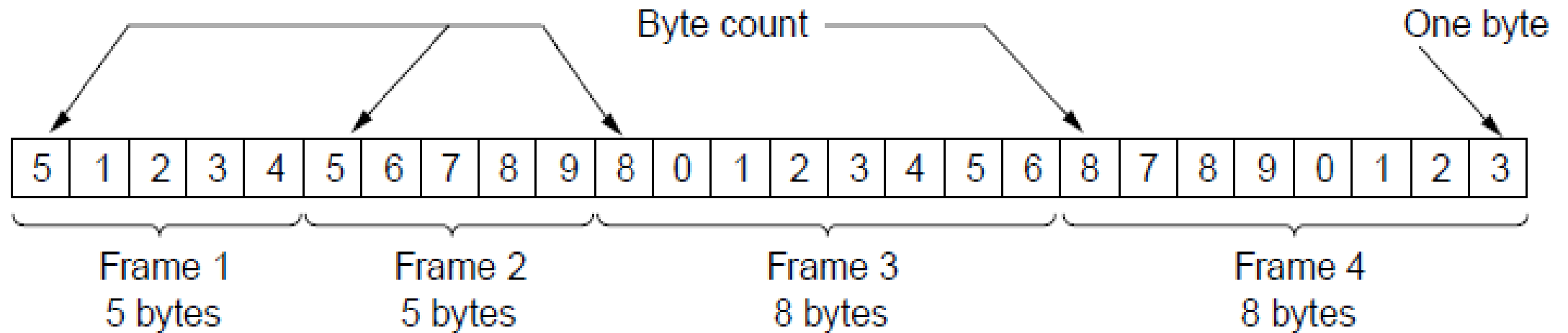
Framing Methods

- We'll look at:
 - Byte count (motivation)
 - Byte stuffing
 - Bit stuffing
- In practice, the physical layer often helps to identify frame boundaries
 - E.g., Ethernet, 802.11

Byte Count

- First try:
 - Let's start each frame with a length field!
 - It's simple, and hopefully good enough ...

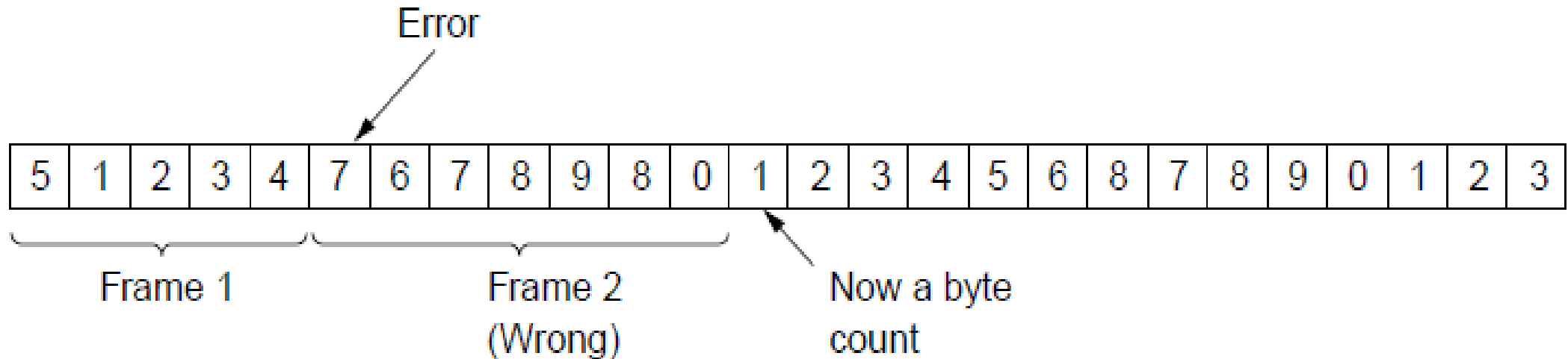
Byte Count (2)



- How well do you think it works?

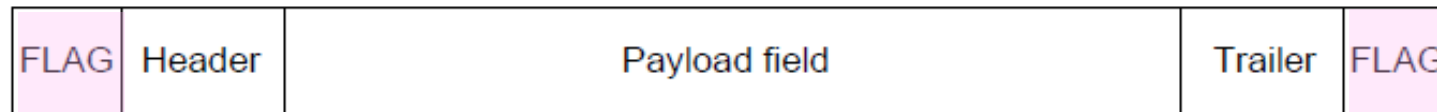
Byte Count (3)

- Difficult to re-synchronize after framing error
 - Want a way to scan for a start of frame



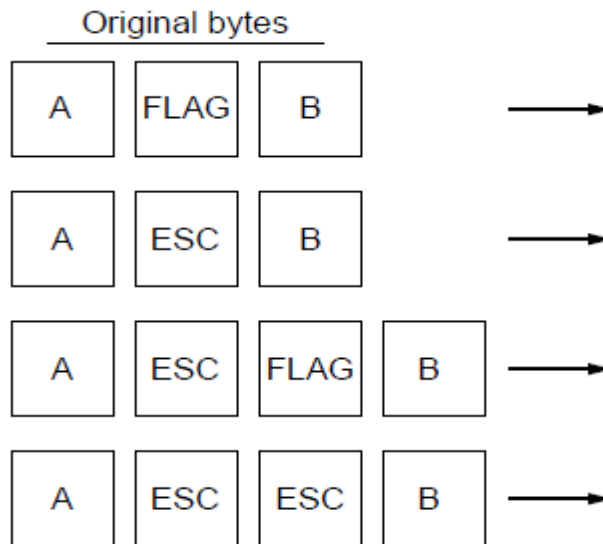
Byte Stuffing

- Better idea:
 - Have a special flag byte value for start/end of frame
 - Replace (“stuff”) the flag with an escape code
 - Complication: have to escape the escape code too!



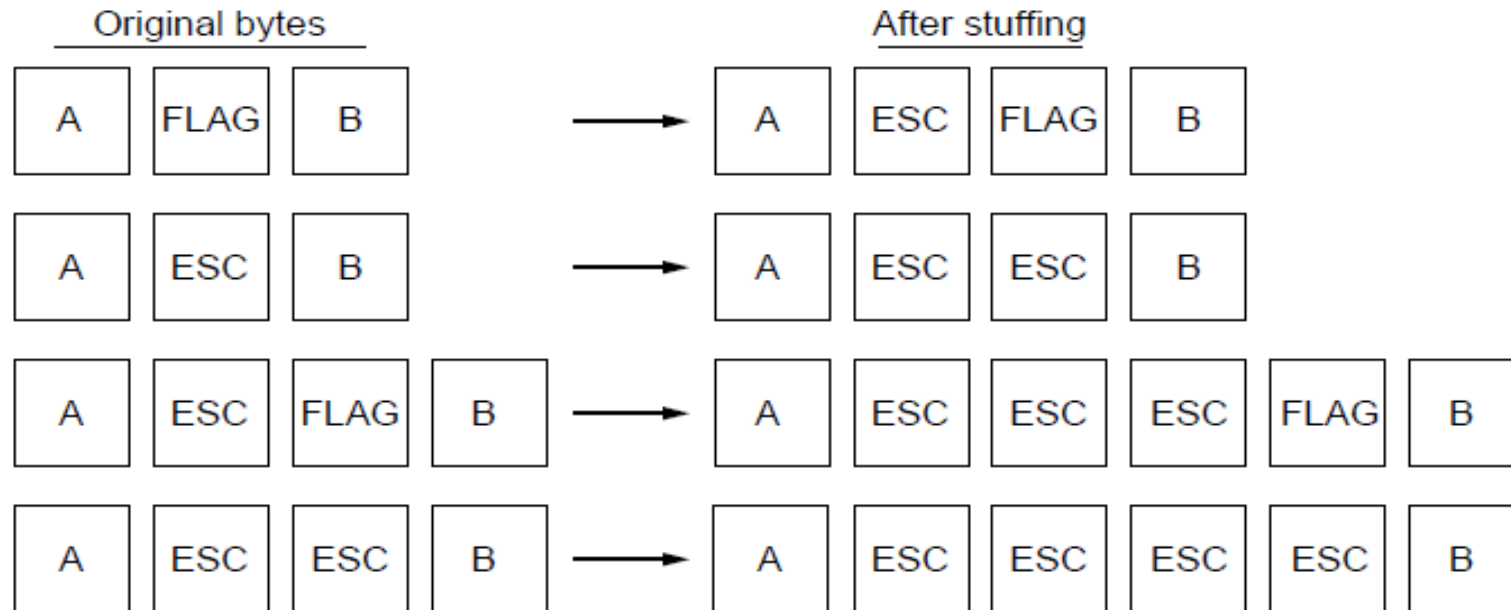
Byte Stuffing (2)

- Rules:
 - Replace each FLAG in data with ESC FLAG
 - Replace each ESC in data with ESC ESC



Byte Stuffing (3)

- Now any unescaped FLAG is the start/end of a frame

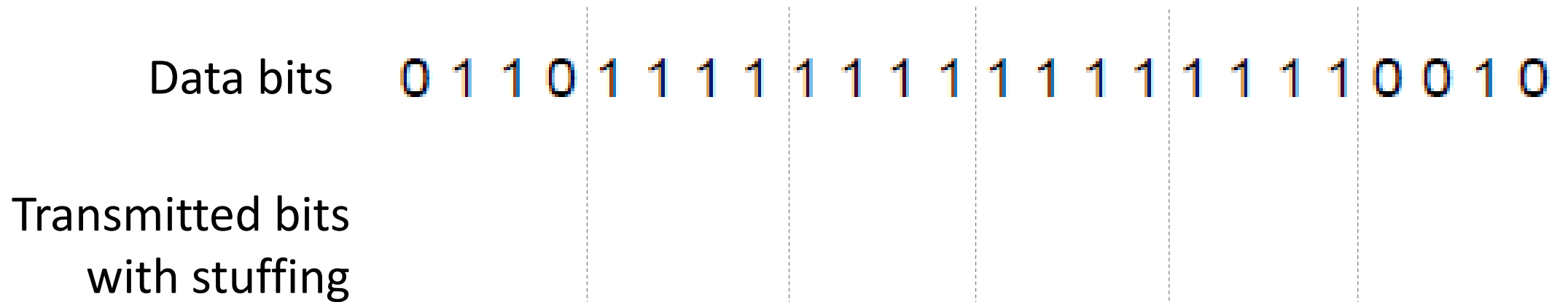


Bit Stuffing

- Can stuff at the bit level too
 - Call a flag six consecutive 1s
 - On transmit, after five 1s in the data, insert a 0
 - On receive, a 0 after five 1s is deleted

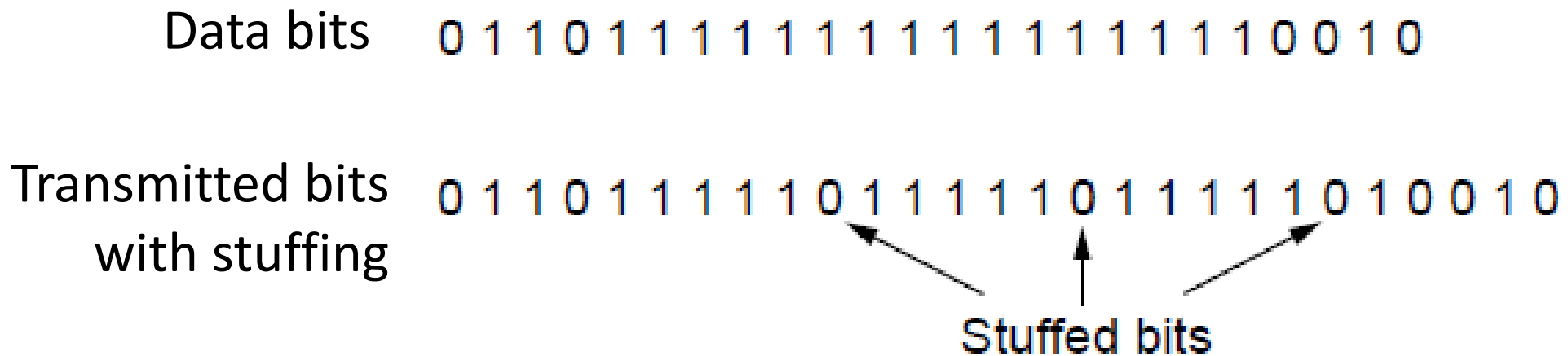
Bit Stuffing (2)

- Example:



Bit Stuffing (3)

- So how does it compare with byte stuffing?

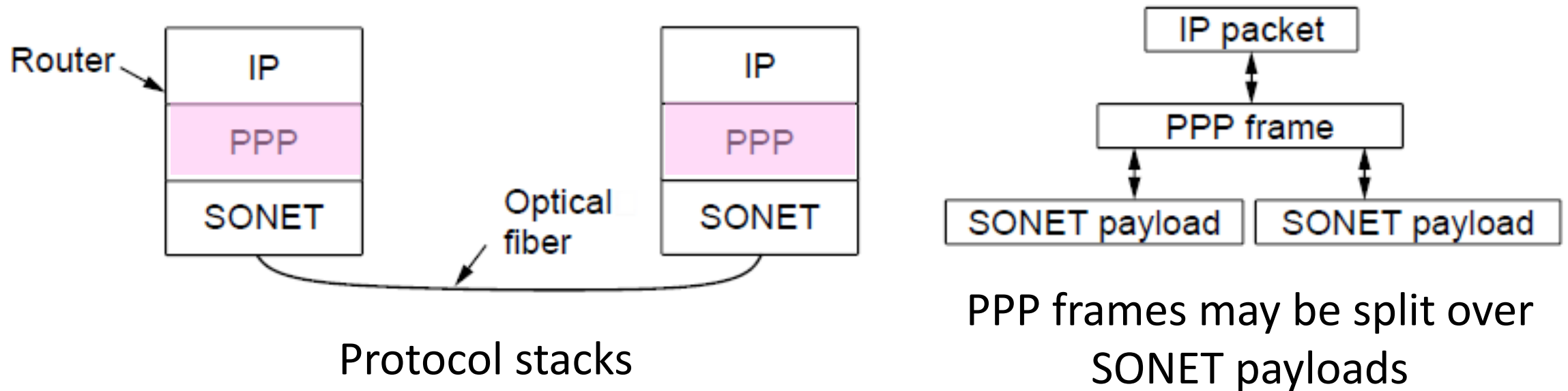


Link Example: PPP over SONET

- PPP is Point-to-Point Protocol
- Widely used for link framing
 - E.g., it is used to frame IP packets that are sent over SONET optical links

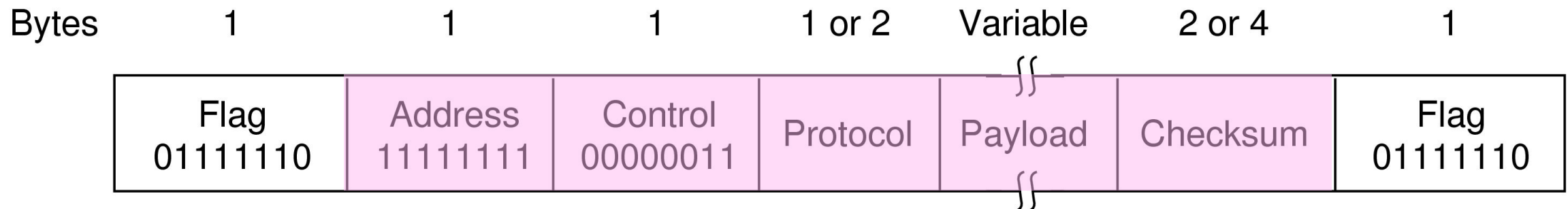
Link Example: PPP over SONET (2)

- Think of SONET as a bit stream, and PPP as the framing that carries an IP packet over the link



Link Example: PPP over SONET (3)

- Framing uses byte stuffing
 - **FLAG** is 0x7E and **ESC** is 0x7D



Link Example: PPP over SONET (4)

- Byte stuffing method:
 - To stuff (unstuff) a byte
 - add (remove) ESC (0x7D)
 - and XOR byte with 0x20
 - Removes **FLAG** from the contents of the frame

Error detection and correction

Handling errors

Topic

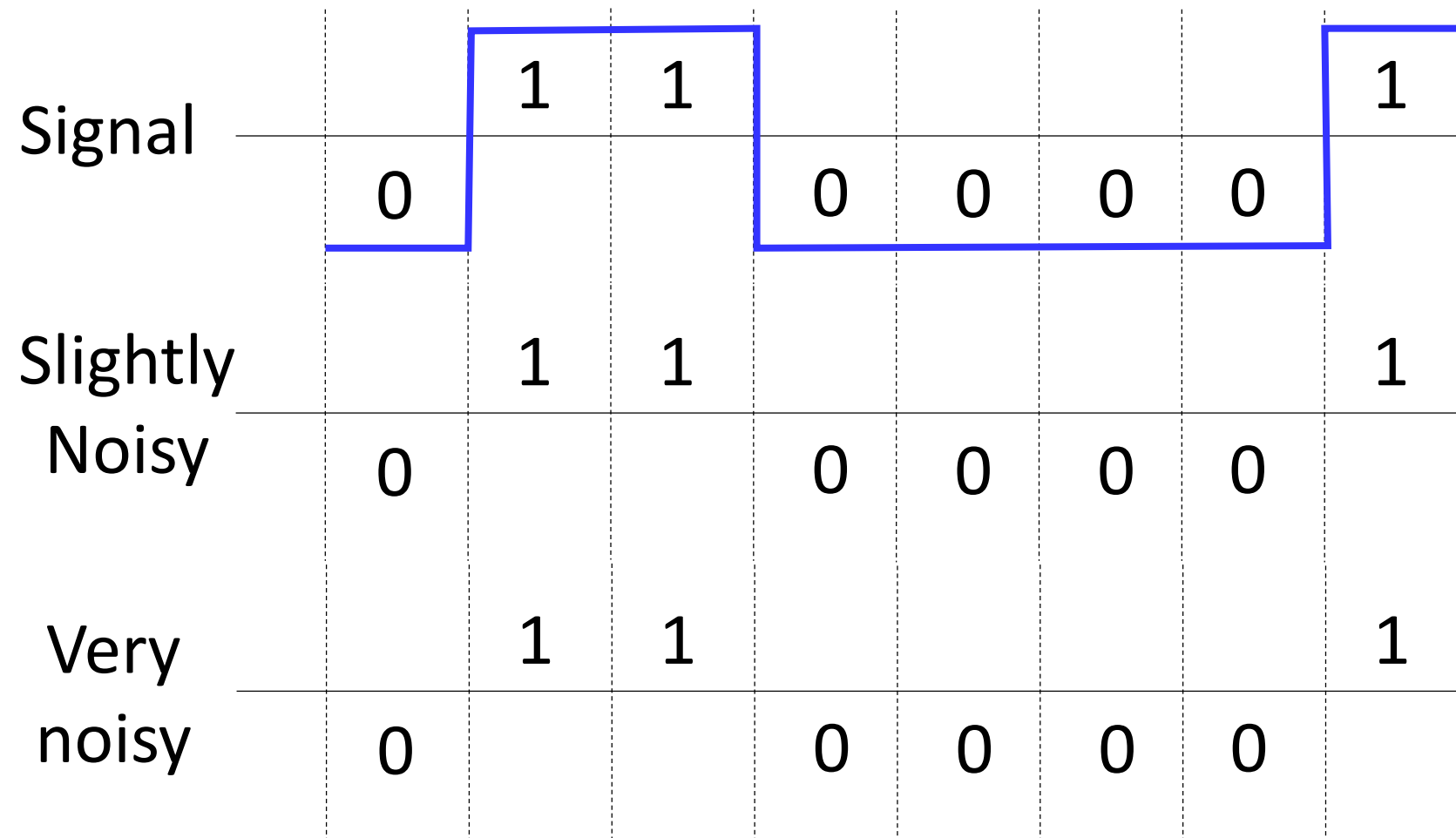
- Some bits will be received in error due to noise.

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

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

- Ideas?

Motivating Example

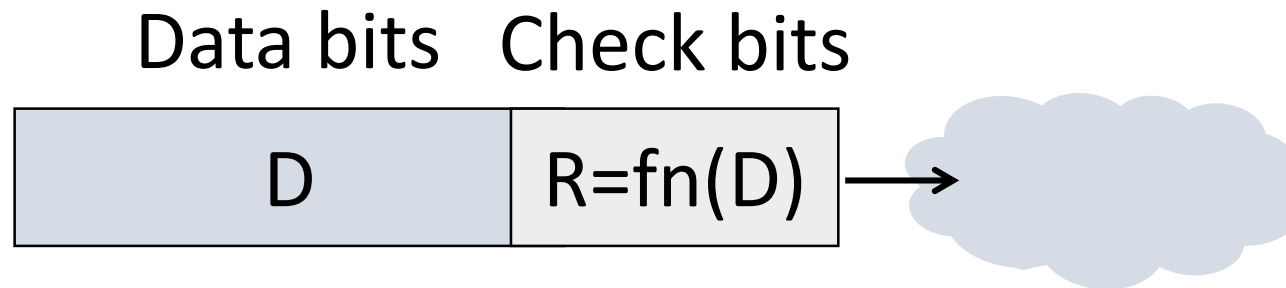
- A simple code to handle errors:
 - Send two copies! Error if different.
- How good is this code?
 - How many errors can it detect/correct?
 - How many errors will make it fail?

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)

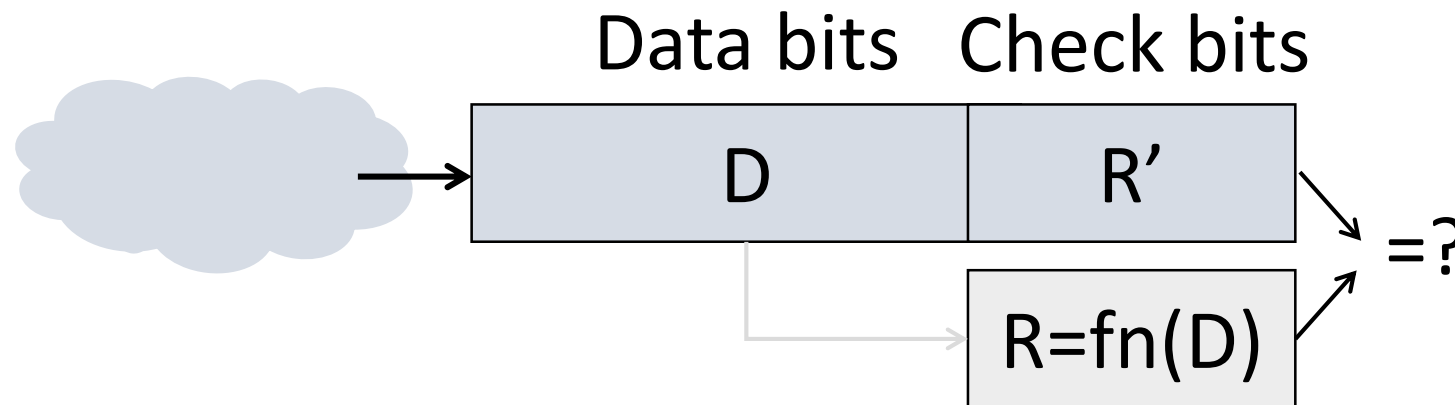


- Sender:
 - Compute R check bits based on the D data bits; send the codeword of $D+R$ bits

Using Error Codes (2)

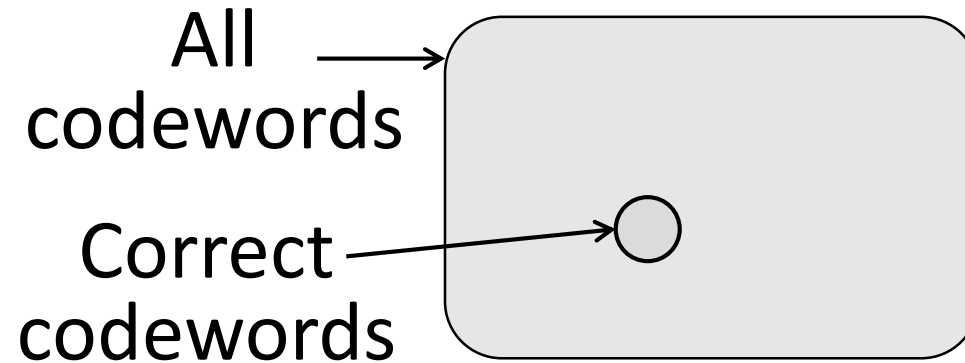
- Receiver:

- 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

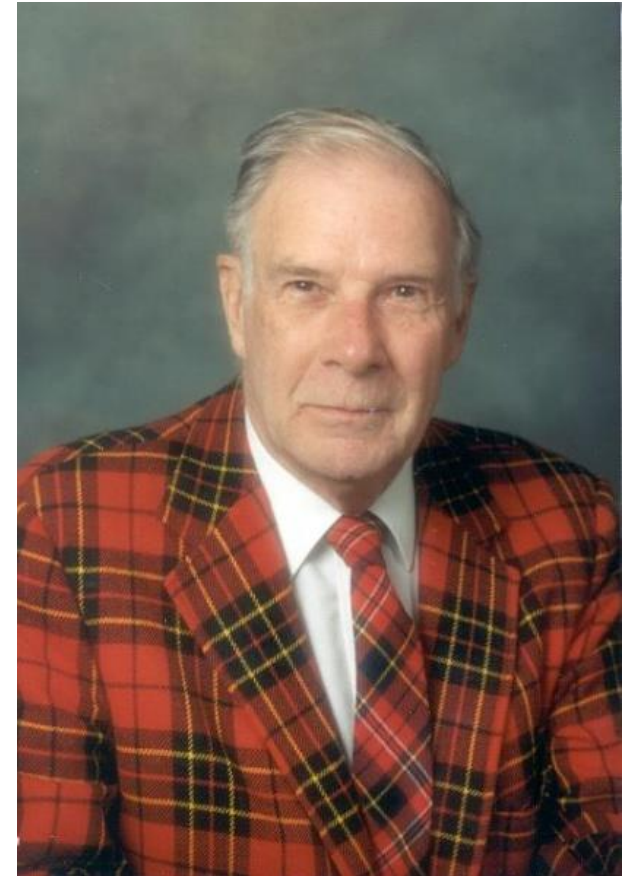
- For D data bits, R check bits:



- 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 coding is the minimum distance between any pair of codewords (bit-strings) that cannot be detected

Hamming Distance (2)

- Error detection:
 - For a coding of distance $d+1$, up to d errors will always be detected
- Error correction:
 - For a coding of distance $2d+1$, up to d errors can always be corrected by mapping to the closest valid codeword

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

1500 bytes	16 bits
------------	---------

- 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
f204
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
f204
f4f5
f6f7
+ (0000)
-----
2ddf1
  ↓
ddf1
+   2
-----
ddf3
  ↓
220c
```

Internet Checksum (4)

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
f204
f4f5
f6f7
+ 220c
-----
```

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

$$\begin{array}{r} 0001 \\ \text{f}204 \\ \text{f}4\text{f}5 \\ \text{f}6\text{f}7 \\ + 220\text{c} \\ \hline 2\text{f}\text{f}\text{f}\text{d} \\ \downarrow \\ \text{f}\text{f}\text{f}\text{d} \\ + \quad 2 \\ \hline \text{f}\text{f}\text{f}\text{f} \\ \downarrow \\ \text{0000} \end{array}$$

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:
 - $n = 302$, $k = \text{one digit}$, $C = 3$

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

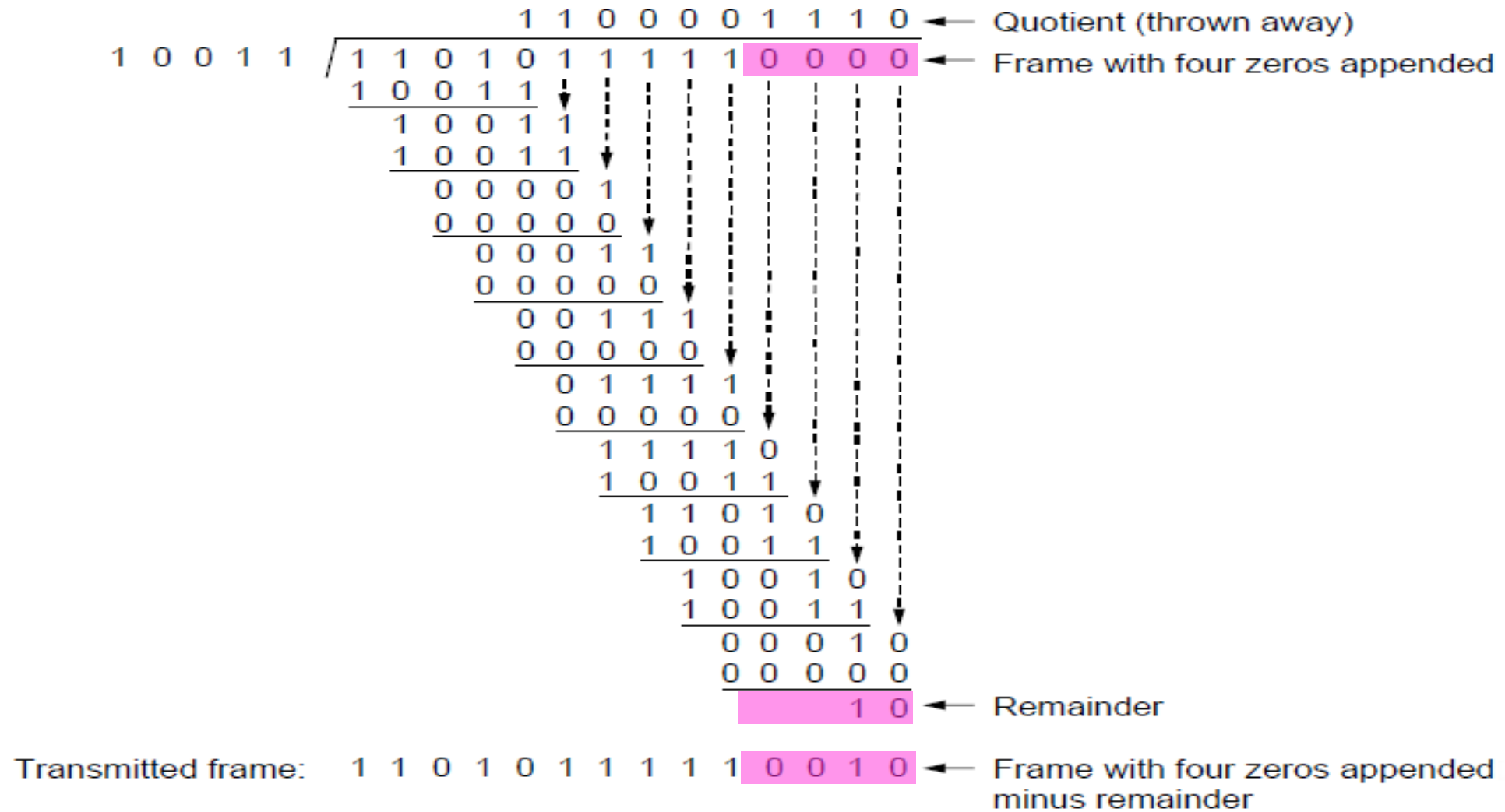
CRCs (4)

Data bits:
1101011111

1 0 0 1 1 | 1 1 0 1 0 1 1 1 1 1

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

CRCs (5)



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