framing

aligning bits

let’s transmit these (binary) messages:

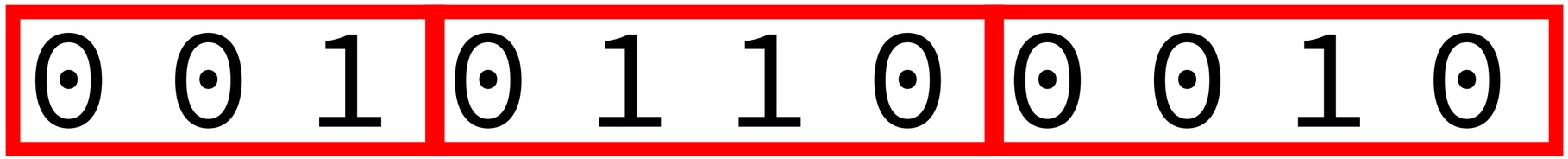
```
001
0110
0010
```

problem: can’t tell where messages/start end

aligning bits

let’s transmit these (binary) messages:

```
001
0110
0010
```



problem: can’t tell where messages/start end

aligning bits

let’s transmit these (binary) messages:

```
001
0110
0010
```

```
0 0 1 0 1 1 0 0 0 1 0
```

problem: can’t tell where messages/start end

framing problem, briefly

framing problem

dividing sequence of symbols into messages called *framing*

individual messages called *frames*

various solutions:

- have extra symbols for message start/end/etc.
  - probably most common with “fast” physical layers
- detect electrically if transmission is happening
- choose sequence of bits/bytes to represent “start message”
  - needs way of “ecsaping” those bits/bytes if they occur within message data

framing problem

dividing sequence of symbols into messages called *framing*

individual messages called *frames*

various solutions:

- have extra symbols for message start/end/etc.
  - *probably most common with "fast" physical layers*
- detect electrically if transmission is happening
- choose sequence of bits/bytes to represent "start message"
  - needs way of "ecsaping" those bits/bytes if they occur within message data

framing problem

dividing sequence of symbols into messages called *framing*

individual messages called *frames*

various solutions:

- have extra symbols for message start/end/etc.
  - probably most common with “fast” physical layers
- detect electrically if transmission is happening
- *choose sequence of bits/bytes to represent “start message”*
  - needs way of “ecsaping” those bits/bytes if they occur within message data

spare symbols

Gigabit ethernet uses scheme with 625 symbols

- 5 amplitude levels over 4 wires

gives room for spare symbols for message start/end/etc.

“bit stuffing”

scheme in textbook called “bit stuffing”

use `01111110` to represent “start/end of message”

make sure `01111110` doesn’t occur within messages accidentally:

- on transmission: replace `11111` in message with `111110`
- on receiving messages, replace each `111110` with `11111`

“bit stuffing” example

messages: 01111100 and 00000111

01111110**01111110**100011111110000001110111110

transmission errors

problem: transmitting symbols has some errors

bad cable, radio interference, mismatched clocks, …

will get frames with incorrect data

will get frames with missing/extra data from misdetecting start/end markers

could pass “bogus” frames to next layer, but…

better to detect this quickly

checksum idea

instead of sending “message” .5cm

say Hash(“message”) = 0xABCDEF12

then send “0xABCDEF12,message” .5cm

when receiving, recompute hash

discard message if checksum doesn’t match

checksum functions

hashes used to check messages called checksums

used at data link layer and upper layers

- lots of places networks want to check messages aren't corrupted .5cm

provides high probability we discard corrupted messages

- larger checksum $\rightarrow$ higher probability

example common checksums

IPv4, TCP —

- based on one's complement sum of data+metadata treated as 16-bit numbers
- one's complement addition = add normally with wraparound + add carry bit at end
- efficient to implement on processor with addition
- easy to compute incrementally

Ethernet

- 32-bit "cyclic redundancy code"
- easy to compute fast in hardware
- always detects up to 3 bits flipped (for sizes used in Ethernet)

beyond checksums

checksums detect errors pretty reliably…

by sending extra bits

sending extra bits can also correct some errors pretty reliably

“error correcting code”

- efficient ways to do this? covered in ECE/CS 4434

very common for wireless

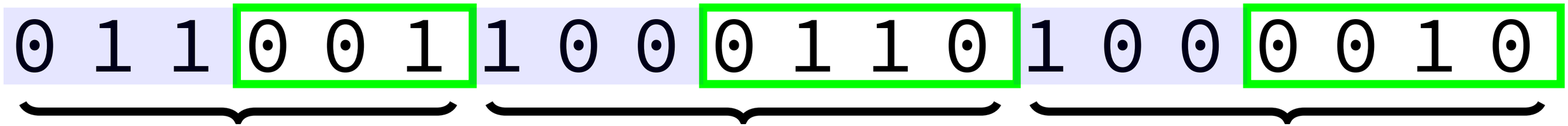
backup slides

size 'header'

let's transmit these (binary) messages:

```
001
0110
0010
```

put 3-bit message size at beginning of messages



read header, then determine number of bits to read before next header

assumption: no gaps between messages?

need to transmit something in between messages

start/end symbol

alternate idea: use bit sequence to mark beginning/end

example choice: send 010 between each frame

send extra 010s when no frames to send

problem: messages can contain 010 or end with 01

one solution: replace 01 in messages with 011

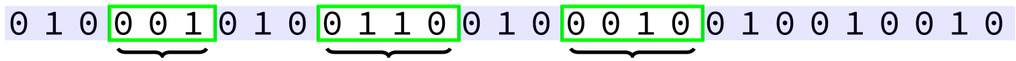
(need to undo replacement when receiving)

start/end symbol

alternate idea: use bit sequence to mark beginning/end

example choice: send 010 between each frame

send extra 010s when no frames to send



problem: messages can contain 010 or end with 01

one solution: replace 01 in messages with 011

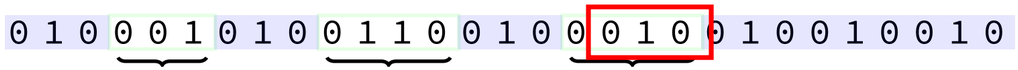
(need to undo replacement when receiving)

start/end symbol

alternate idea: use bit sequence to mark beginning/end

example choice: send 010 between each frame

send extra 010s when no frames to send



problem: messages can contain 010 or end with 01

one solution: replace 01 in messages with 011

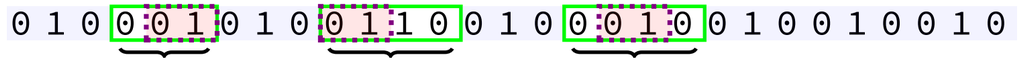
(need to undo replacement when receiving)

start/end symbol

alternate idea: use bit sequence to mark beginning/end

example choice: send 010 between each frame

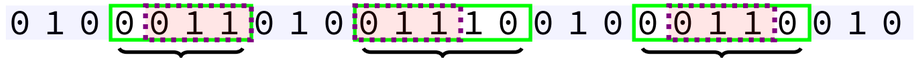
send extra 010s when no frames to send



problem: messages can contain 010 or end with 01

one solution: replace 01 in messages with 011

(need to undo replacement when receiving)



exercise: overhead

let’s say start/end symbol is 011

and we replace 01 with 010 .5cm

exercise (1): minimum number of bits used to send 1000 bit message?

exercise (2): maximum number of bits used to send 1000 bit message?

escaping?

can think of replacement similar to escaping strings in C

start/end marker is "

" $\rightarrow$ \"

\ $\rightarrow$ \\

represent `foo \R"3"13\` using `"foo \\R\"3\"13\\"`

but needed tweaks to idea to work with bits instead of bytes .5cm

some physical layers allow transmitting bytes at a time

framing protocols for those (example: PPP) use -like idea

help from physical layer?

suppose instead of transmitting 0 or 1

physical layer transmits 0 or 1 or 2 or 3 or 4

probably going to ‘waste’ one of these values

- example: transmit every two bits as 0 or 1 or 2 or 3 .5cm

idea: take advantage of leftover ‘symbol’ 4

use it to send start/end

- similar idea used in many versions of Ethernet

bad choice of start/end (1)

let’s say we choose delimiter 0000

what do we need to escape?

A. any 0000

A. any 000

B. any 00

C. any 0

bad choice of start/end (1)

sending 10 and 01

100000010000

sending 1 and 001

100000010000 .5cm

oops!

textbook example: start/end = 01111110

(and escaping 11111 $\rightarrow$ 111110)

types of transmission errors

desynchronization:

- missing bits/bytes
- adding bits/bytes

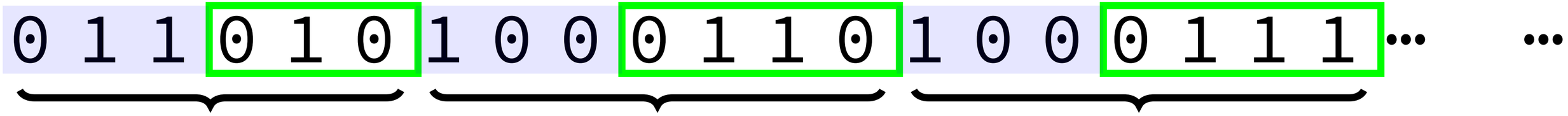
changing symbols

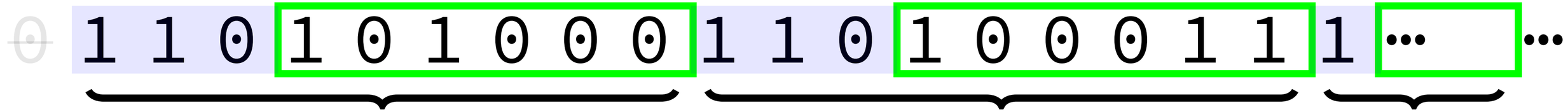
- from ‘noise’/‘interference’

desynchronization and framing (1)

with purely size-based framing

- almost all future sizes messed up

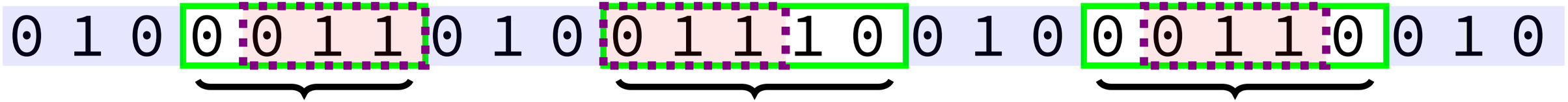


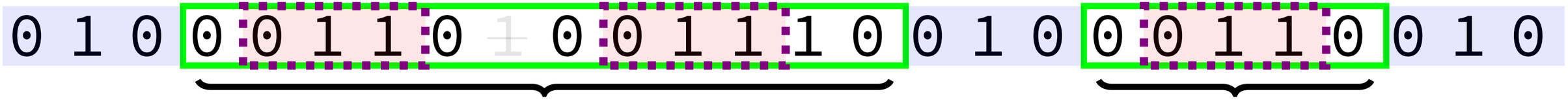


desynchronization and framing (2)

with start/end marker idea

- can have start/end-marker corrupted or added by corruption
- may mess up multiple frames, but will eventually be resync'd





fixed-sized frames

suppose all packets are same size

“clock-based framing”

example: SONET looks for start symbol every 810 bytes

- if starts being missing, try to resync

flipping bits?

flipping bits basically has same problems as synchornization

- can corrupt sizes and start/end markers
- can add extra start/end markers

transmission errors

problem: transmitting symbols has some errors

bad cable, radio interference, mismatched clocks, …

will get frames with incorrect data

will get frames with missing/extra data from misdetecting start/end markers

could pass “bogus” frames to next layer, but…

better to detect this quickly

checksum idea

instead of sending “message” .5cm

say Hash(“message”) = 0xABCDEF12

then send “0xABCDEF12,message” .5cm

when receiving, recompute hash

discard message if checksum doesn’t match

checksum functions

hashes used to check messages called checksums

used at data link layer and upper layers

- lots of places networks want to check messages aren't corrupted .5cm

provides high probability we discard corrupted messages

- larger checksum $\rightarrow$ higher probability

example common checksums

IPv4, TCP —

- based on one's complement sum of data+metadata treated as 16-bit numbers
- one's complement addition = add normally with wraparound + add carry bit at end
- efficient to implement on processor with addition
- easy to compute incrementally

Ethernet

- 32-bit "cyclic redundancy code"
- easy to compute fast in hardware
- always detects up to 3 bits flipped (for sizes used in Ethernet)

beyond checksums

checksums detect errors pretty reliably…

by sending extra bits

sending extra bits can also correct some errors pretty reliably

“error correcting code”

- efficient ways to do this? covered in ECE/CS 4434

very common for wireless

framing homework

implement send+receive messages (strings of bytes) using bits

send_message(MESSAGE)

handle_bit_from_network(BIT)

calls got_message_function(MESSAGE) .5cm

but:

need to indicate message boundaries somehow

need to handle bit flips and missing bits without losing everything

bit bytearrays

```
# In the given code for the assignment:
def bytes_to_bits(the_bytes):
    result = bytearray()
    for a_byte in the_bytes:
        for i in range(8):
            result.append(0 if (a_byte & (0x1 << i)) == 0 else 1)
    return result
```

```
bytes_to_bytes(b'\x93') ==
bytearray([1,0,0,1, 0,0,1,1])
```

using bytearray of 0s and 1s

(because Python doesn’t have convenient bitarray type)