changelog

6 Sep 2024: add slides giving example of ambiguous seperators when escaping is not sufficient

aligning bits

let's transmit these (binary) messages:

```
001
0110
0010
```

```
001|0110|0010
```

aligning bits

let's transmit these (binary) messages:

```
001
0110
0010
```

`001011000010`

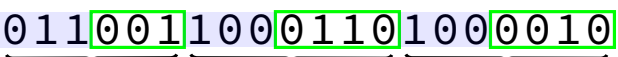
problem: can't tell where messages/start end

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

size 'header'

let's transmit these (binary) messages:

```
001
0110
0010
```

put 3-bit message size at beginning of messages

011001100011010010

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

assumption: no gaps between messages?

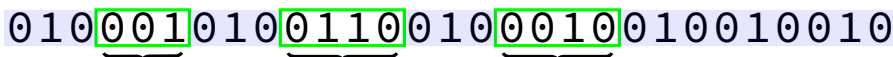
need to transmit *something* in between messages

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

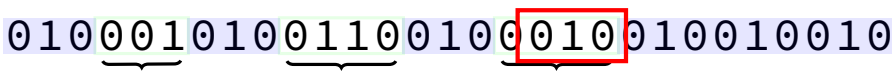


start/end symbol

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

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

`010001010011010010001010010010`

problem: messages can contain `010` or end with `01`

one solution: replace `01` in messages with `011`

(need to undo replacement when receiving)

`010001011010011110010001101010`

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

escaping?

can think of replacement similar to escaping strings in C

- start/end marker is "
- " → \"
- \ → \\

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

but needed tweaks to idea to work with bits instead of bytes

some physical layers allow transmitting bytes at a time

example: upcoming assignment

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

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

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)

4

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)

4

sending `10` and `01`

`10000001`0000

sending `1` and `001`

`10000001`0000

oops!

textbook example: start/end = `01111110`

types of transmission errors

desynchronization:

- missing bits/bytes
- adding bits/bytes

flipping bits

- from ‘noise’/‘interference’

types of transmission errors

desynchronization:

- missing bits/bytes
- adding bits/bytes

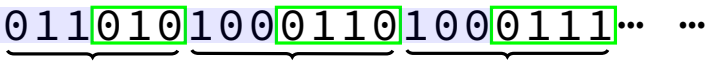
flipping bits

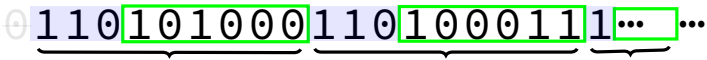
- from ‘noise’/‘interference’

desynchronization and framing (1)

with purely size-based framing

almost all future sizes messed up

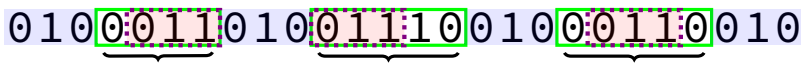




desynchronization and framing (2)

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

types of transmission errors

desynchronization:

- missing bits/bytes
- adding bits/bytes

flipping bits

- from ‘noise’/‘interference’

flipping bits?

flipping bits basically has same problems as synchornization

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

checksums

unsolved issue: will generate frames with bad data

could rely on other layers to deal with that, but...

- a lot better to detect this early

checksum idea

instead of sending “message”

say Hash(“message”) = 0xABCDEF12

then send “0xABCDEF12,message”

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

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

can send some extra bits can *correct* some errors pretty reliably

"error correcting code"

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

framing homework

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

`send_message(MESSAGE)`

`handle_bit_from_network(BIT)`

calls `got_message_function(MESSAGE)`

but:

need to indicate message boundaries somehow

need to handle bit flips and missing bits without losing everything

backup slides