University of Glasgow

VIA VERITAS VITA

The Data Link Layer

Networked Systems 3
Lecture 5

Purpose of Data Link Layer

- Arbitrate access to the physical layer
  - Identify devices – addressing
  - Structure and frame the raw bitstream; detect and correct bit errors
  - Control access to the channel (media access control)
- Turn the raw bit stream into a structured communications channel

Addressing

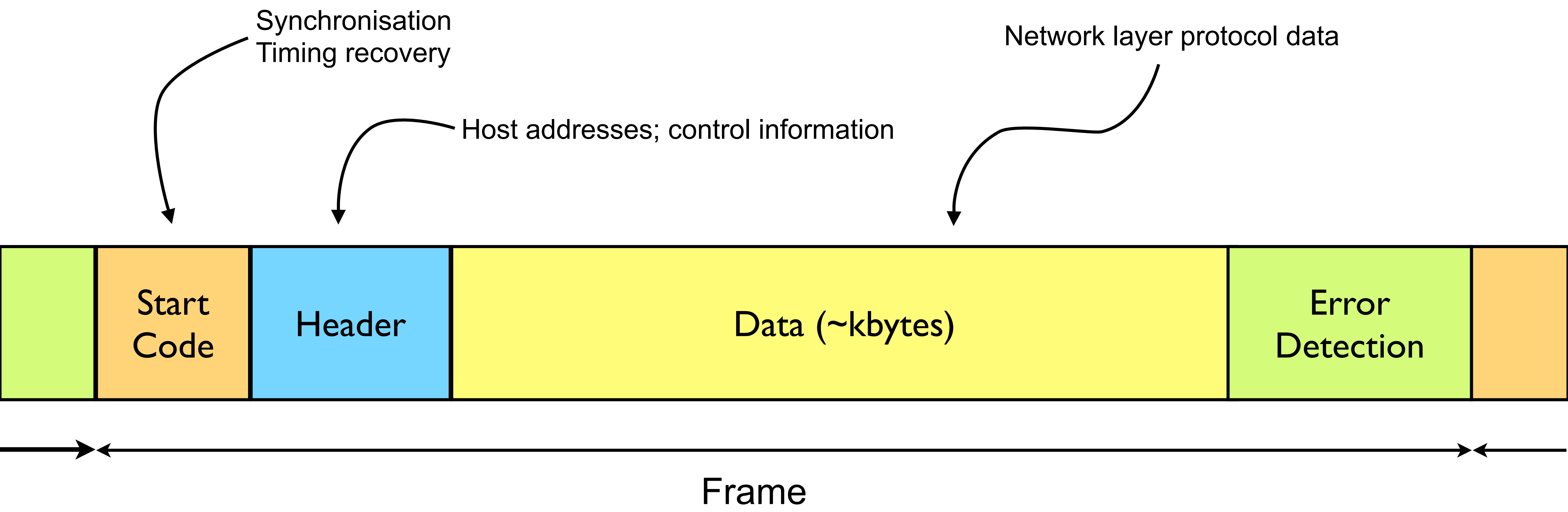
- Physical links can be *point-to-point* or *multi-access*
  - Wireless links are common example of multi-access, but several hosts can also be connected to a single cable to form multi-access wired link
  - Multi-access links require host addresses, to identify senders and receivers

- Host addresses may be *link-local* or *global* scope
  - Sufficient to be link-local (i.e., unique amongst hosts connected to a link)
  - Many data link layer protocols use global scope addresses
    - Examples: Ethernet and IEEE 802.11 Wi-Fi
    - Simpler to implement if devices can move, since don't need to change address when connected to a different link
    - Some privacy concerns

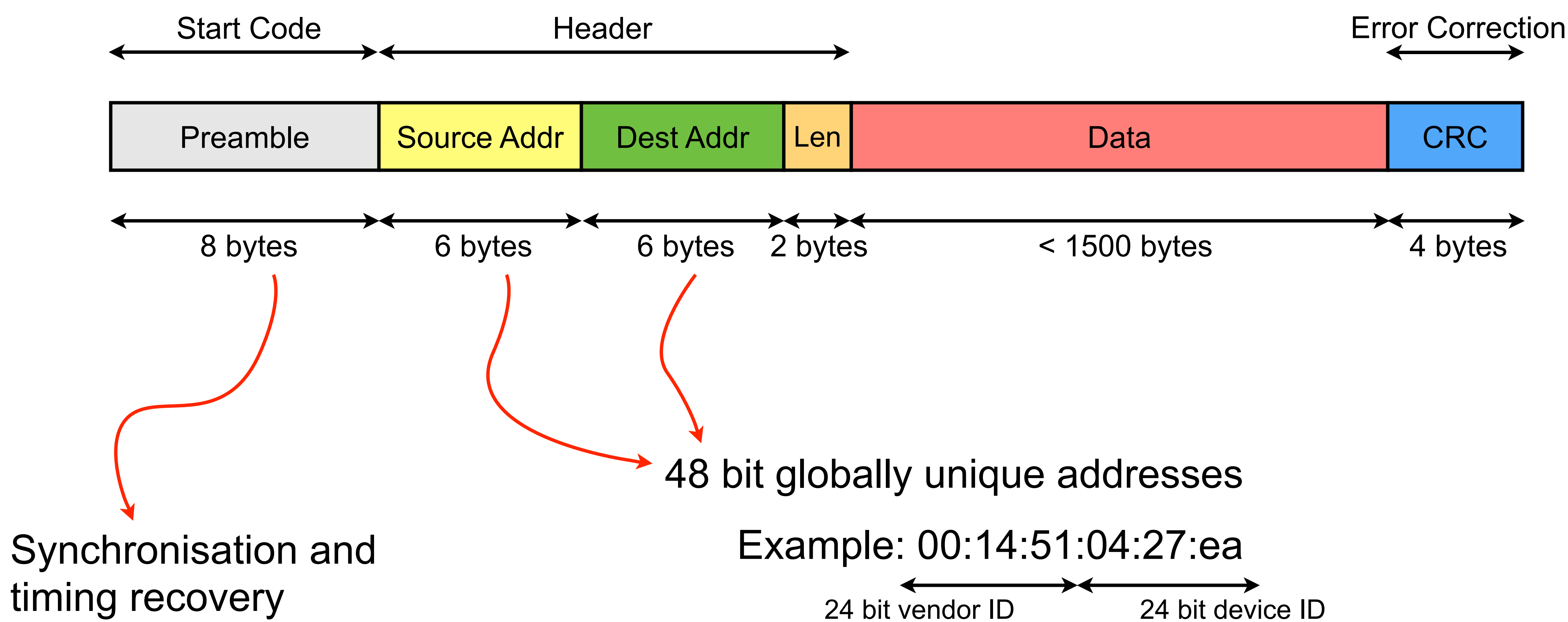
Framing and Synchronisation

- Physical layer provides unreliable raw bit stream
  - Bits might be corrupted
  - Timing can be disrupted
- Data link layer must correct these problems
  - Break the raw bit stream into *frames*
  - Transmit and repair individual frames
  - Limit scope of any transmission errors

Frame Structure



Example: Ethernet



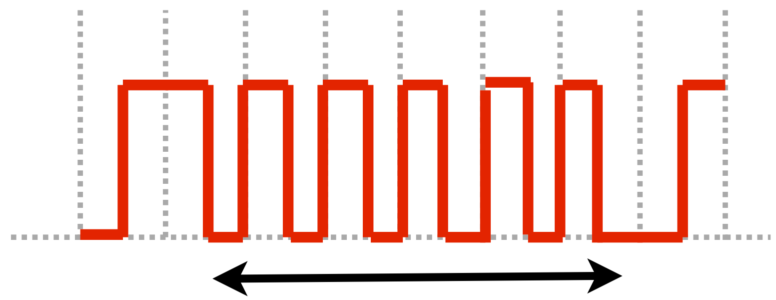
Synchronisation (1)

- How to detect the start of a message?
  - Leave gaps between frames
    - Problem – physical layer typically doesn't guarantee timing (clock skew, etc.)
  - Precede each frame with a length field
    - What if that length is corrupted? How to find next frame?
  - Add a special *start code* to beginning of frame
    - A unique bit pattern that *only* occurs at the start of each frame
    - Enables synchronisation after error – wait for next start code, begin reading frame headers

Synchronisation (2)

- What makes a good start code?
  - Must not appear in the frame headers, data, or error detecting code
  - Must allow timing recovery

0 1 1 1 1 1 1 0 →



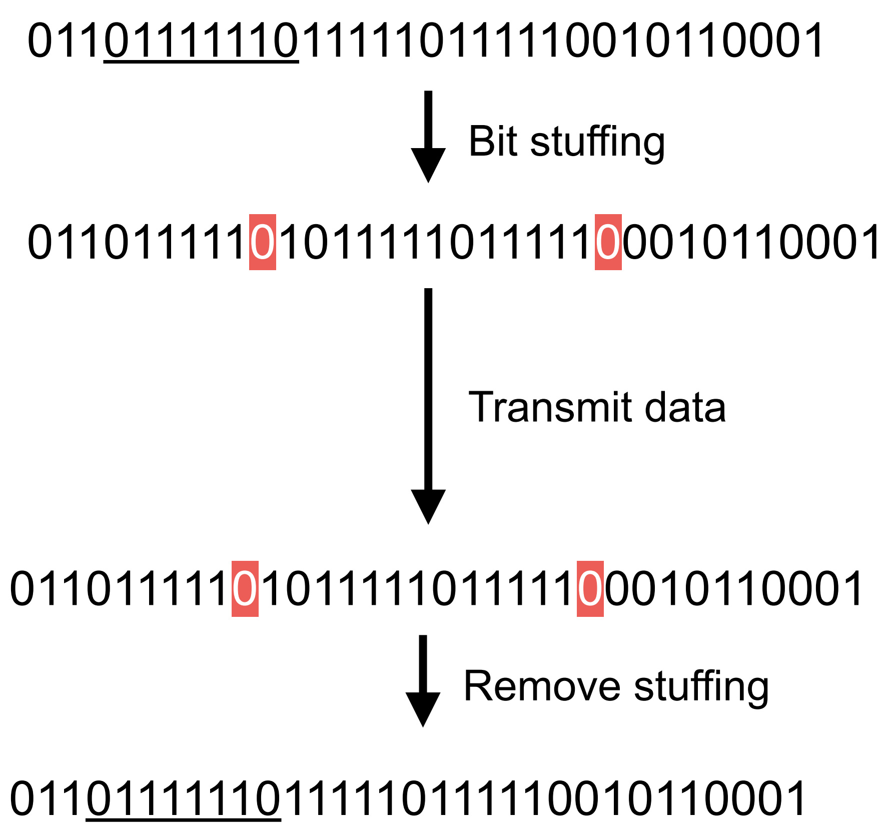
Manchester Encoding

Receiver measures timing

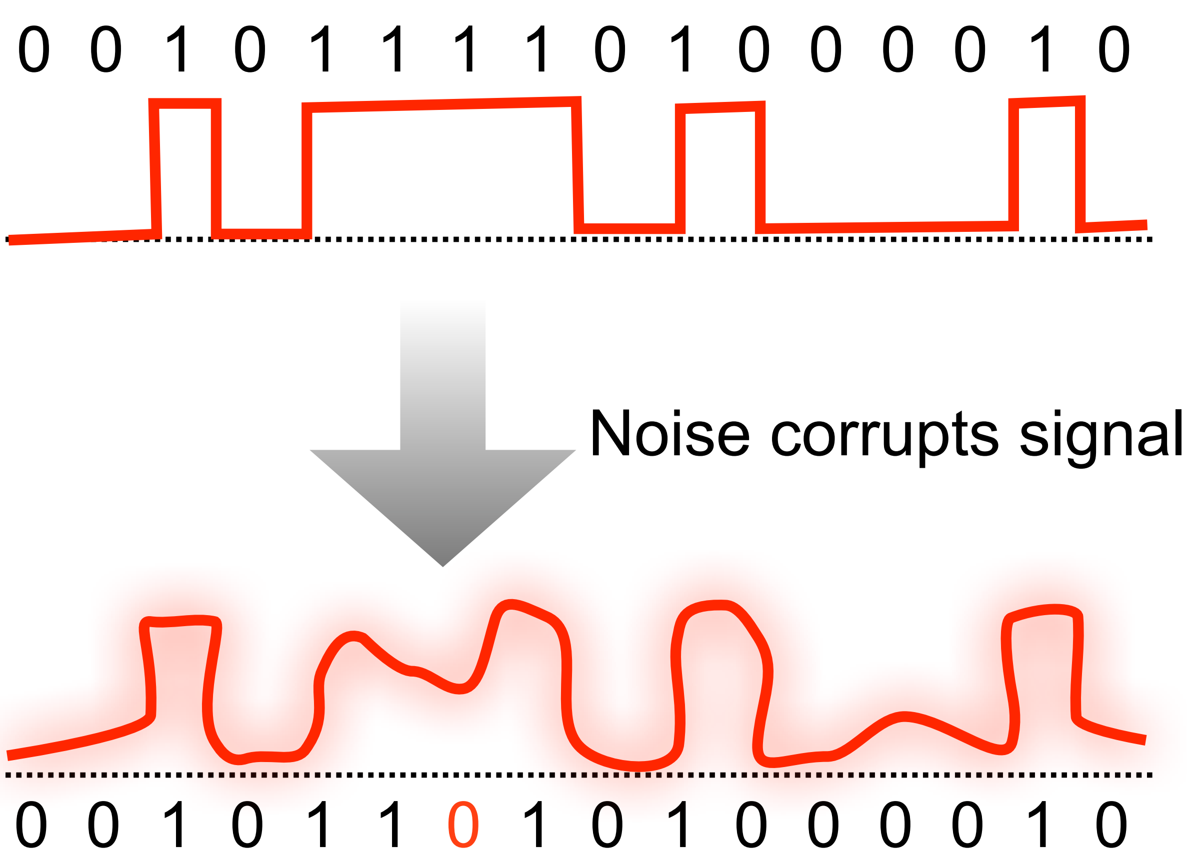
Start code should generate a regular pattern after physical layer coding

Synchronisation (3)

- What if start code appears in data? Use *bit stuffing* to give a transparent channel
- Sender inserts a 0 bit after sending any five consecutive 1 bits – unless sending start code
- If receiver sees five consecutive 1 bits, look at sixth bit:
    - If 0, has been stuffed, so remove
    - If 1, look at seventh bit:
        - If 0, start code
        - If 1, corrupt frame



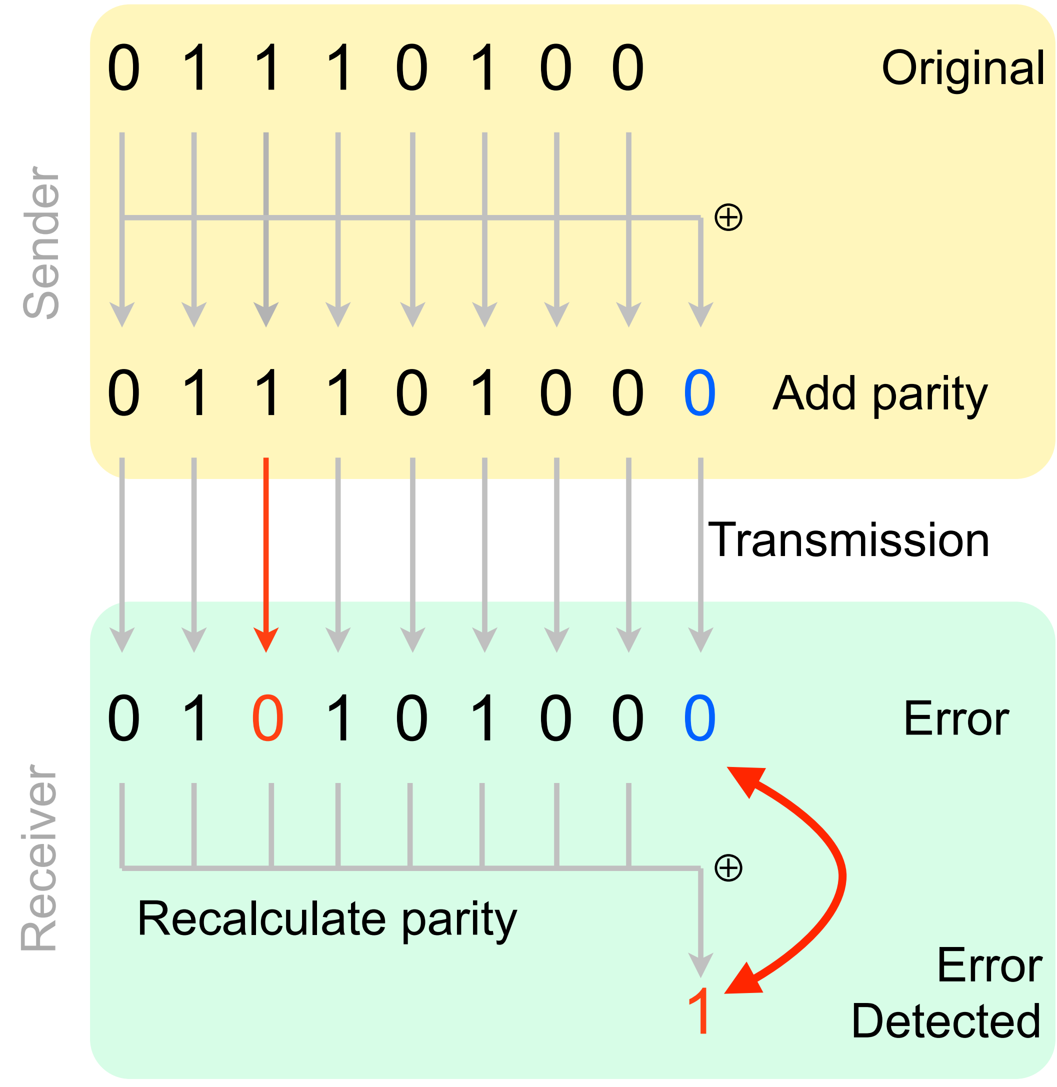
Error Detection



- Noise and interference at the physical layer can cause bit errors
  - Rare in wired links, common in wireless systems
- Add *error detecting code* to each packet

Parity Codes

- Simplest error detecting code
- Calculate *parity* of the data
  - How many 1 bits are in the data?
  - An odd number → parity 1
  - An even number → parity 0
  - Parity bit is the XOR (“⊕”) of data bits
- Transmit parity with the data, check at receiver
  - Detects all single bit errors



The Internet Checksum

```
#include <stdint.h>

// Internet checksum algorithm. Assumes
// data is padded to a 16-bit boundary.
uint16_t
internet_cksum(uint16_t *buf, int buflen)
{
    uint32_t sum = 0;

    while (buflen--) {
        sum += *(buf++);
        if (sum & 0xffff0000) {
            // Carry occurred, wrap around
            sum &= 0x0000ffff;
            sum++;
        }
    }
    return ~(sum & 0x0000ffff);
}
```

- Sum data values, send as a *checksum* in each frame
    - Internet protocol uses a 16 bit ones complement checksum
- Receiver recalculates, mismatch → bit error
- Better error detection than parity code
    - Detects many multiple bit errors

Other Error Detecting Codes

- Parity codes and checksums relatively weak
  - Simple to implement
  - Undetected errors reasonably likely
- More powerful error detecting codes exist
  - *Cyclic redundancy code* (CRC)
  - More complex → fewer undetected errors
  - (see recommended reading for details)

Error Correction

- Extend error detecting codes to *correct* errors
  - Transmit error correcting code as additional data within each frame
  - Allows receiver to correct (some) errors without contacting sender

Error Correcting Codes: Hamming Code

- Simple error correcting code:
  - Send *n* data bits and *k* check bits each word
  - Check bits are sent as bits 1, 2, 4, 8, 16, …
  - Each check bit codes parity for some data bits:
    - $b_1 = b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} \ldots$
    - $b_2 = b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} \oplus b_{14} \oplus b_{15} \ldots$
    - $b_4 = b_5 \oplus b_6 \oplus b_7 \oplus b_{12} \oplus b_{13} \oplus b_{14} \oplus b_{15} \ldots$
    - i.e., starting at check bit *i*, check *i* bits, skip *i* bits, repeat

Character	ASCII	Hamming Code
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	11111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	00101011111
d	1100100	11111001100
e	1100101	00111000101

Richard Hamming

Error Correcting Codes: Hamming Code

- On reception:
  - set *counter* = 0
    recalculate check bits, *k* = 1, 2, 4, 8, … in turn {
      if check bit *k* is incorrect {
        counter += *k*
      }
    }
    if (*counter* == 0) {
      no errors
    } else {
      bit *counter* is incorrect
    }
  - Corrects all single bit errors

Character	ASCII	Hamming Code
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	11111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	00101011111
d	1100100	11111001100
e	1100101	00111000101

Error Correcting Codes

- Other error correcting codes exist
- Tradeoff: complexity, amount of data added, ability to correct multi-bit errors

- Can also detect error, and request retransmission – error correcting codes not the only means of repair

Summary

- Data link layer
- Addressing
- Framing, synchronisation, start codes
- Error detecting and correcting codes