

Data transmission

IGCSE Computer Science 0478 · Topic 2 revision notes

Everything from the 6 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

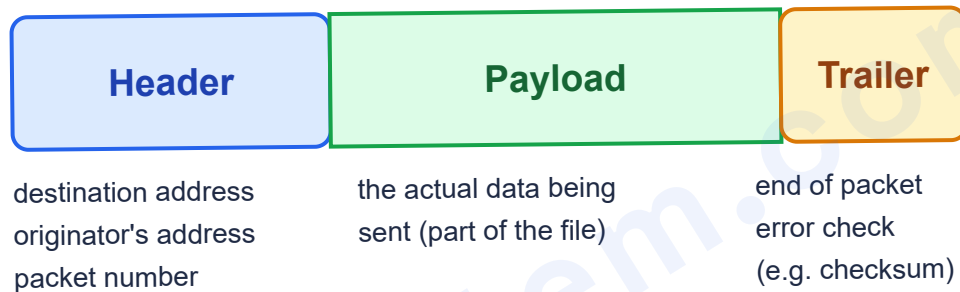
- | | |
|---|---|
| 1. <u>Packets and packet switching.</u> | 4. <u>Parity checks, checksums and echo checks</u> |
| 2. <u>Serial and parallel, simplex, half-duplex and full-duplex</u> | 5. <u>Check digits and automatic repeat query (ARQ)</u> |
| 3. <u>The universal serial bus (USB)</u> | 6. <u>Symmetric and asymmetric encryption</u> |

1 Packets and packet switching

Syllabus 2.1

Data sent over a network is broken down into **packets**. Each packet has a **header** (destination address, originator's address and packet number), a **payload** (the actual data) and a **trailer** (marks the end of the packet and holds an error check). In **packet switching** each packet may take a **different route**, chosen by **routers**, so packets can arrive **out of order**; once the last packet arrives they are **reordered** using their packet numbers.

the structure of one packet



a file is split into many packets like this one

Fig. 1: every packet has a header, a payload and a trailer

Worked example: Reordering

Packets of an email arrive in the order 3, 1, 5, 2, 4. Explain what the receiving computer does.

1. It waits until the last packet has arrived.
2. It reads the packet number in each header.
3. It reorders the packets as 1, 2, 3, 4, 5 and joins the payloads to rebuild the email.

Answer: Reorders the packets using their packet numbers once all have arrived

Header, payload, trailer.

- ✓ **Header:** destination address, originator's address, packet number.
- ✓ **Payload:** the actual data.
- ✓ **Trailer:** end of packet marker and an error check.
- ✓ **Packet switching:** split, number, different routes via routers, may arrive out of order, reorder when the last one arrives.

Watch out!

Packets do not make the data itself smaller. The total amount of data is the same (in fact slightly more, because every packet adds a header and trailer).

The packet number goes in the header, not the trailer. Learn the three header items exactly: destination address, originator's address, packet number.

It is the **router** that controls the route a packet takes. A packet does not "choose" its own path.

2 Serial and parallel, simplex, half-duplex and full-duplex

Syllabus 2.1

Every connection is described in two ways. **How many bits at once:** **serial** sends one bit at a time down a single wire; **parallel** sends several bits at the same time down several wires. **Which direction:** **simplex** is one direction only; **half-duplex** is both directions but only one at a time; **full-duplex** is both directions at the same time. Parallel is faster over **short** distances but suffers **skew** (bits arriving at different times) over long ones, so serial is used for long distances.

Simplex



Half-duplex



Full-duplex

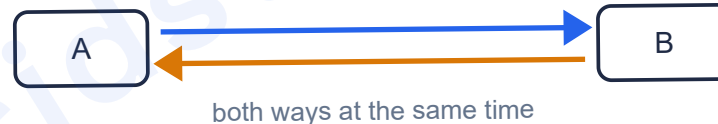


Fig. 1: the direction of data flow in simplex, half-duplex and full-duplex

Worked example: Video call

Ali makes a video call to a friend in another country. Identify the most suitable transmission method and justify it.

1. The friend is far away, so the data travels a long distance: **serial**, because there is no skew and the bits arrive in order.
2. Both people talk and send video at the same time: **full-duplex**, so data travels both ways at once.

Answer: Serial full-duplex

Two choices, two reasons.

- ✓ **Serial:** one bit at a time, one wire, long distances, no skew.
- ✓ **Parallel:** several bits at once, several wires, fast but short distances only (skew).
- ✓ **Direction:** simplex one way; half-duplex both ways in turn; full-duplex both ways at once.
- ✓ **Scenario questions:** name both parts and link each reason to the scenario.

Watch out!

"Parallel is faster" is only true over short distances. If a question involves a long cable, the answer is serial, and the reason is skew.

Do not just name the method. A justification such as "because data only needs to travel in one direction" is what earns the second mark.

thelucidstem.com

3 The universal serial bus (USB)

Syllabus 2.1

USB (universal serial bus) is an **industry standard interface** for connecting devices to a computer. It uses **serial** transmission: data is sent **one bit at a time**. When a device is plugged in, the computer **automatically detects** it and loads the correct **driver**. The cable can also **supply power** to the device. The main drawbacks are a **limited cable length** and slower speeds on older USB versions.

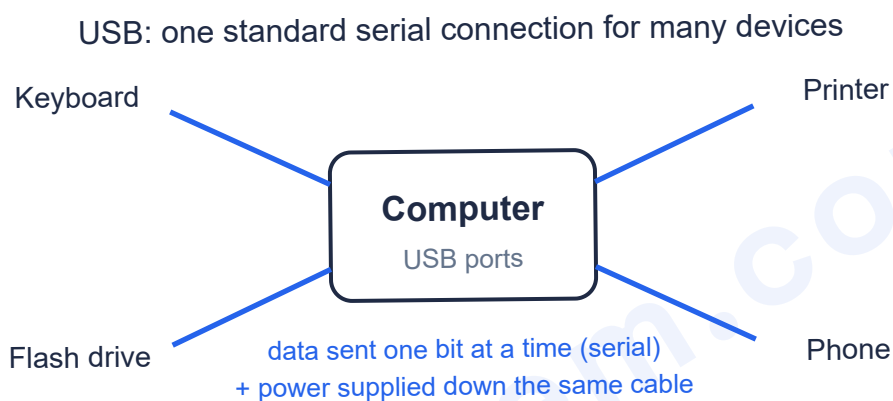


Fig. 1: many different devices connect through the same USB standard

Worked example: A new mouse

Describe what happens when a new USB mouse is plugged into a laptop.

1. The laptop detects that a device has been connected.
2. It recognises the device as a mouse and loads the correct driver automatically.
3. Movement data is sent to the laptop serially, one bit at a time, and the mouse is powered through the USB cable.

Answer: Detected, driver loaded, data sent serially, powered by the cable

Serial, standard, simple.

- ✓ **How:** serial, one bit at a time.
- ✓ **On connection:** detected automatically, correct driver loaded.
- ✓ **Benefits:** industry standard, auto-detection, supplies power, fits one way, backward compatible.
- ✓ **Drawbacks:** limited cable length, older versions slower, limited power.

Watch out!

USB is *serial*, not *parallel*. If a question asks how USB transmits data, "one bit at a time" is the key phrase.

"It is fast" is too vague to earn a benefit mark. Give a specific benefit such as automatic detection, power supply or being an industry standard.

thelucidstem.com

4 Parity checks, checksums and echo checks

Syllabus 2.2

Interference during transmission can cause **data loss**, **data gain** or **data change**, so data must be checked when it arrives. A **parity check** adds a **parity bit** so that each byte has an even (**even parity**) or odd (**odd parity**) number of 1s; a **parity block** adds a parity byte too, so the exact bit in error can be found. A **checksum** is a value calculated from the data and sent with it, then recalculated by the receiver. In an **echo check** the receiver sends the data back and the sender compares it with the original.

even parity: the total number of 1s must be even



parity bit

7 data bits 1101000 contain three 1s

3 is odd, so the parity bit is set to 1 → four 1s in total

if the receiver counts an odd number of 1s, an error has occurred

Fig. 1: setting the parity bit for even parity

Worked example: Even parity

Add an even parity bit (as the leftmost bit) to the 7-bit data 1011001.

1. Count the 1s: 1, 0, 1, 1, 0, 0, 1 gives four 1s.
2. Four is already even, so the parity bit is 0.

Answer: 01011001

Agree, send, recheck.

- ✓ **Parity:** count the 1s; the parity bit makes the total even (even parity) or odd (odd parity).
- ✓ **Parity block:** odd row + odd column = the bit in error; flip it to correct it.
- ✓ **Checksum:** calculated from the data, sent with it, recalculated by the receiver and compared.
- ✓ **Echo check:** data sent back to the sender, who compares it with the original.

Watch out!

A parity check cannot detect an error where **two** bits change (for example 01011001 becoming 01011111); the number of 1s stays even, so the error is missed.

A single parity bit only shows that a byte is wrong. Only a parity block (parity bits plus a parity byte) shows exactly which bit is wrong, so it can be corrected.

An echo check cannot tell where the error happened: the data may have been correct on arrival and changed on the way back. Either way, it has to be sent again.

5 Check digits and automatic repeat query (ARQ)

Syllabus 2.2

A **check digit** is an extra digit **calculated from the other digits** of a number and added to the end. When the number is entered, the check digit is **recalculated** and compared; if it does not match, a **data entry** error has been made. Check digits are used in **ISBNs** and **barcodes**. **Automatic repeat query (ARQ)** checks **transmitted** data: the receiver sends a **positive acknowledgement** if the data is correct or a **negative acknowledgement** if not, and the sender resends if it gets a negative acknowledgement or if a **timeout** passes with no acknowledgement.

ISBN-13 check digit for 978-1-86197-271-?

9	7	8	1	8	6	1	9	7	2	7	1	?
$\times 1$	$\times 3$	$\times 1$	$\times 3$	$\times 1$	$\times 3$	$\times 1$	$\times 3$	$\times 1$	$\times 3$	$\times 1$	$\times 3$	
9	21	8	3	8	18	1	27	7	6	7	3	

add the products: total = 118

$118 \text{ MOD } 10 = 8$, and $10 - 8 = 2$

check digit = 2 → 978-1-86197-271-2

(if the remainder is 0, the check digit is 0)

Fig. 1: calculating an ISBN-13 check digit with weights 1 and 3

Worked example: Calculate

Calculate the check digit for the ISBN that begins 978095748422.

1. Digits 9 7 8 0 9 5 7 4 8 4 2 2, weights 1 3 1 3 1 3 1 3 1 3 1 3.
2. Products: 9, 21, 8, 0, 9, 15, 7, 12, 8, 12, 2, 6.
3. Total = 109. $109 \text{ MOD } 10 = 9$.
4. Check digit = $10 - 9 = 1$.

Answer: 1, giving 9780957484221

Recalculate and compare.

- ✓ **Check digit:** calculated from the other digits; recalculated on entry; mismatch = entry error.
- ✓ **ISBN-13:** weights 1, 3, 1, 3 ...; add; MOD 10; 10 minus the remainder (0 stays 0).
- ✓ **ARQ:** positive acknowledgement if correct, negative acknowledgement if not.
- ✓ **Timeout:** no acknowledgement within the set time, so the sender resends.

Watch out!

A check digit detects errors in data entry. Parity, checksums and echo checks detect errors in transmission. Mixing these up loses marks.

Do not stop at the remainder. The remainder 3 is not the check digit: subtract it from 10 (unless the remainder is 0).

The timeout is on the sender. Write "the sender waits a set time for an acknowledgement; if none arrives, it resends the data".

6 Symmetric and asymmetric encryption

Syllabus 2.3

Encryption scrambles **plaintext** into **ciphertext** using an algorithm and a **key**, so that if data is **intercepted** it is **meaningless** to anyone without the key. Encryption does not stop interception. In **symmetric** encryption the **same key** encrypts and decrypts, so the key has to be shared, and it could be intercepted too. In **asymmetric** encryption there is a matching pair of keys: data encrypted with the receiver's **public key** can only be decrypted with the receiver's **private key**, which is never sent.

asymmetric encryption: Bob's key pair

an interceptor sees only the ciphertext

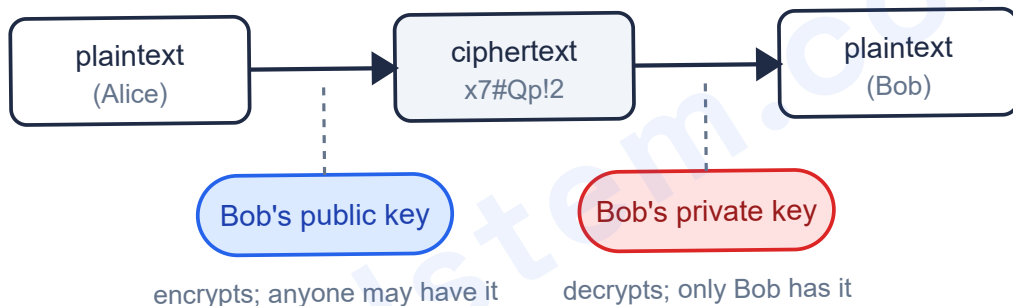


Fig. 1: anyone can encrypt with Bob's public key, but only Bob's private key decrypts

Worked example: Caesar shift

Encrypt HELLO using a Caesar cipher with a key of 3.

1. Shift each letter 3 places along the alphabet.
2. $H \rightarrow K$, $E \rightarrow H$, $L \rightarrow O$, $L \rightarrow O$, $O \rightarrow R$.

Answer: KHOOR

Who holds which key?

- ✓ **Purpose:** intercepted data is meaningless without the key; it does not stop interception.
- ✓ **Plaintext to ciphertext:** using an algorithm and a key.
- ✓ **Symmetric:** one key encrypts and decrypts; the key must be shared and could be intercepted.
- ✓ **Asymmetric:** encrypt with the receiver's public key; decrypt with the receiver's private key, which is never sent.

Watch out!

Encryption does *not* stop data being intercepted or stolen. It makes intercepted data meaningless (unreadable) without the key. Write exactly that.

Real symmetric encryption uses very long keys, not a shift of 3. The Caesar cipher only shows the idea: the same key locks and unlocks.

The sender uses the receiver's public key, not its own. The answer "the message is encrypted with a private key and decrypted with a public key" is the wrong way round for sending secret data.