Topic 02: Data Transmission

IGCSE Computer Science 0478 | SenpaiCorner | Mr. Wei

Learning Objectives

2.1 Types and methods of data transmission: packets, packet header, payload, trailer, destination address, packet number, originator's address, packet switching, router, serial, parallel, simplex, half-duplex, full-duplex, universal serial bus (USB).

2.2 Methods of error detection: parity check (odd and even), parity byte, parity block, checksum, echo check, check digit, ISBNs, bar codes, automatic repeat query (ARQ), positive/negative acknowledgements, timeout.

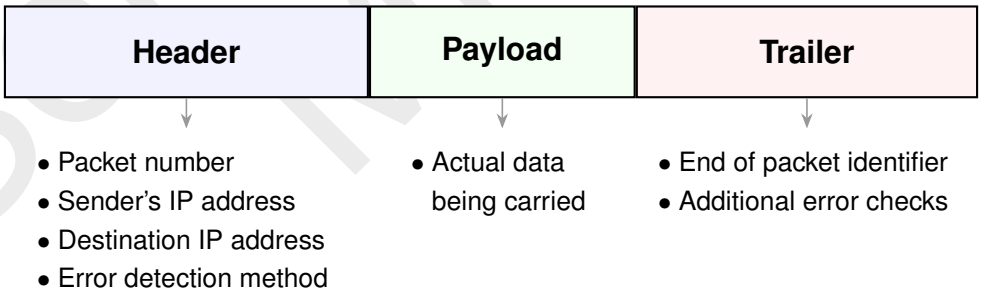
2.3 Encryption: plaintext, ciphertext, symmetric encryption, asymmetric encryption, public key, private key.

2.1 Types and Methods of Data Transmission

Data Packets

Data is broken down into **packets** before being transmitted across a network. Once all packets arrive at the destination, they are reassembled to form the original data.

Each packet has three sections:



Exam Tip — Packet Structure

The header contains *routing information* (IP addresses, packet number). The payload is the *actual data*. The trailer holds the *end-of-packet marker and extra error checking data*. Cambridge frequently asks which part stores the destination IP address — the answer is always the **header**. The payload is **not** in the header.

Corruption Defined Precisely

Corruption means packet data is *changed* or *lost* in some way — or data is *gained* that was not originally in the packet. Do not write "the data is damaged"; use change, loss, or gain of data.

Packet Switching

- Data is broken/split/divided into packets.
- Each packet **can take a different route** to the destination.
- A **router** controls the route a packet takes, selecting the shortest/fastest available path.
- Packets may arrive **out of order**.
- Once the last packet has arrived, packets are **reordered**.
- If a packet is missing or corrupted, it is **requested to be resent**.

Benefits of packet switching:

- Packets are small — easier to control and retransmit.
- Each packet can take a different route — important when one route is busy or out of action.
- Quicker overall transmission.
- Only individual lost or corrupted packets are resent — minimises wasted bandwidth and time.
- Harder to hack: each packet contains minimal data and travels separately.

Drawback of packet switching:

- Data must be reassembled at the destination.

Annotating a Packet-Switching Diagram

When annotating a diagram showing packet switching from Device A to Device B, credit-worthy points are:

1. Packets sent through several routers.
2. Packets take *different* routes from Device A to Device B.
3. Packets arrive out of order at Device B.
4. Packets are reordered once all have arrived at Device B.

Modes of Data Transmission

Mode	Description	Example Applications	Key Notes
Simplex	Data transmitted in one direction only (unidirectional).	Mic to computer; sensor to computer; computer to speaker; computer to monitor; webcam to computer.	Cheap (one wire); no feedback channel.
Half-duplex	Data transmitted in both directions, but *not* simultaneously.	Walkie-talkie	Fewer wires than full-duplex; slower.
Full-duplex	Data transmitted in both directions simultaneously.	Broadband internet; video conferencing; instant messaging; multiplayer games; computer to modem.	Fastest; most expensive.

Simplex Is Not Enough for a Printer

A printer connection must be at least **half-duplex**, not simplex. With simplex, data only flows in one direction, so the printer cannot send an error message back to the computer — and the computer cannot receive confirmation that printing succeeded.

Serial vs Parallel Data Transmission

Type	Description	Applications	Benefits	Drawbacks
Serial	One bit at a time; single wire.	USB; telephone line; Wi-Fi; long-distance links.	Cheaper to buy/install; less interference/crosstalk; bits arrive in order (synchronised); more reliable over long distances.	Slower; additional data may need to be sent to organise bits.
Parallel	Multiple bits simultaneously; multiple wires.	Integrated circuits; CPU buses; RAM (short distance only).	Faster; bits do not need organising before sending.	Expensive to set up; limited distance; more prone to interference; bits may arrive *skewed* (out of sync).

Choosing Serial or Parallel — Exam Decision Guide

- **Long distances** ⇒ serial: less interference, lower cost, bits arrive in order, reduce rate of errors.
- **Internal circuits** (CPU to RAM): parallel — distance is very short, high-speed transmission is essential.
- **Between rooms**: either can be justified.
  - Serial: data arrives in order, less chance of error, can transmit to another room without skewing.
  - Parallel: faster transmission speed, next room is within distance for parallel, unlikely to skew.
- **Large data over short distance** (e.g. uploading video to a server): parallel half-duplex — fastest transmission of large data; does not need simultaneous upload and download.

Combined Transmission Methods

The mode (simplex / half-duplex / full-duplex) and method (serial / parallel) combine to give six specific types:

Combined Type	Description
Serial simplex	One bit at a time; single wire; one direction only.
Serial half-duplex	One bit at a time; single wire; both directions — but only one direction at a time.
Serial full-duplex	One bit at a time; single wire; both directions simultaneously.
Parallel simplex	Multiple bits at a time; multiple wires; one direction only.
Parallel half-duplex	Multiple bits at a time; multiple wires; both directions — but only one direction at a time.
Parallel full-duplex	Multiple bits at a time; multiple wires; both directions simultaneously.

Choosing the Right Combined Method

A theatre has a microphone connected by cable to speakers around the hall. The most appropriate method is **serial simplex**: data only needs to travel from microphone to speakers (one direction); the long cable distance favours serial to avoid skewing and errors; transmission speed is adequate.

Universal Serial Bus (USB)

USB Aspect	Key Details and Characteristics
Data Transmission	• Uses **serial** transmission — data is sent one bit at a time down a single wire. • Plug-and-play architecture: the computer automatically detects the plugged-in device and installs the matching device driver.
Benefits	• **Universal connection** — highly compatible across different computer systems. • **Backward compatible** — seamlessly supports earlier USB generations. • **Serial advantages** — data is less likely to be skewed or corrupted; cheaper to produce. • **Convenience** — standard cables plug in securely one way; automatically handles driver setups. • **Power Delivery** — powers and charges connected peripherals directly without a separate power source. • **Independence** — does not rely on a wireless network connection to transfer data.
Drawbacks	• Strict maximum cable length limitations — cannot be deployed over long distances. • Data transfer rates are not as fast as some specialized wired alternatives; legacy versions suffer from throttled transmission rates. • Outdated USB legacy hardware standards may fail to interface with modern computers.
USB-C Improvements	• Reversible connector profile — can be inserted either way round. • Sleeker, smaller, and significantly thinner physical footprint. • Dramatically increased data transmission rates. • Supports vastly superior wattage for faster device power delivery.

2.2 Methods of Error Detection

Why Data Must Be Checked for Errors

- Computers require data in specific formats — processes and calculations go wrong if data is not in the correct format.
- Computers process information in binary (1s and 0s) — results are incorrect if the order of bits is changed.

How errors occur during transmission:

- **Data loss** — data is lost in transmission.
- **Data gain** — additional data is received that was not sent.
- **Data change** — some bits are changed or flipped.

Causes of interference:

- *Wireless:* signals blocked by physical barriers (buildings); bad weather (rain/clouds); interference from other wireless signals.
- *Wired:* physical components damaged or degraded; interference from external signals.

Error detection methods on the syllabus: parity check; checksum; echo check; check digit; automatic repeat query (ARQ).

Parity Check

A **parity bit** is a bit (0 or 1) added to a byte of data (in the most significant — left-most — bit position) to ensure the byte follows an agreed parity protocol (odd or even).

How odd parity check detects errors:

1. Parity is set to odd or even — sender and receiver must agree.
2. Data is split into bytes (blocks of 7 bits).
3. The sender counts the number of 1s in each byte.
4. A parity bit is added to each byte to make the total number of 1s match the agreed parity.
5. After transmission, the receiving device recounts the 1s in each byte.
6. If the count is *odd* — no error detected. If the count is *even* — an error is detected.

Parity Check Limitations

- A parity check detects **that** an error has occurred, but **not where** the error is.
- It **cannot detect** errors when an *even number* of bits change — incorrect bits still add up to the correct parity.
- It **cannot detect** a *transposition* of bits — the order changes but the count of 1s remains the same.

Parity Byte and Parity Block

A **parity block** extends the parity check by totalling the number of 1s both **horizontally** (per byte row) and **vertically** (per bit column). A **parity byte** is sent with the data; it contains the vertical parity bits from the column calculation.

Example parity block (odd parity):

	Parity bit	**Bit 2**	**Bit 3**	**Bit 4**	**Bit 5**	**Bit 6**	**Bit 7**	**Bit 8**
Byte 1	0	1	1	0	1	0	1	1
Byte 2	0	0	0	0	1	0	0	0
Byte 3	0	0	1	0	1	1	1	1
Byte 4	1	0	1	1	1	0	0	1
Byte 5	1	1	0	1	0	1	0	1
Byte 6	1	1	0	0	1	1	1	0
Byte 7	0	0	1	1	1	1	1	0
Byte 8	0	1	0	1	1	0	0	0
Parity byte	0	1	1	1	0	1	1	1

- Each byte row calculates horizontal parity as a parity bit — as in a standard parity check.
- Each bit column calculates vertical parity — resulting in the parity byte.
- The parity byte is calculated before transmission and sent with the block.
- By cross-referencing the failing row (byte) and failing column (bit position), the exact corrupted bit can be pinpointed.

Parity Block Detects What Parity Byte Cannot

A simple parity byte check will miss a *transposition* of bits within the same byte — the row parity stays correct. However, the parity block's vertical column parity will detect the change, and the intersection of the failing row and failing column locates the corrupted bit precisely.

Checksum

1. A checksum value is calculated from the data using an algorithm.
2. The value is transmitted together with the data.
3. The receiver recalculates the value using the same algorithm.
4. If the recalculated value **matches** the transmitted value — no error detected.
5. If the values **differ** — an error is detected and a request is sent to retransmit the data.

Echo Check

1. A copy of the received data is transmitted back to the sender.
2. The sender compares the returned data to the original sent data.
3. If they **match** — no error detected.
4. If they **do not match** — an error is detected and the sender retransmits the data.

Echo Check — Key Drawback

If the two sets of data differ, it is **unknown whether** the error occurred during the original transmission *or* during the return transmission of the copy. Note also that echo check does **not** use a calculated value — Cambridge has used this as a MCQ discriminator.

Check Digit

A check digit is a **validation method** used for data *entry* — not data transmission. It detects mis-typing, mis-scanning, and similar input errors.

1. A check digit is calculated from the inputted data using a mathematical algorithm.
2. The check digit is appended to the data.
3. After data entry, the digit is recalculated.
4. If the calculated digit **matches** the stored value — data entered is correct.
5. If it **does not match** — the data entered is incorrect.

Examples: International Standard Book Numbers (ISBN); bar codes.

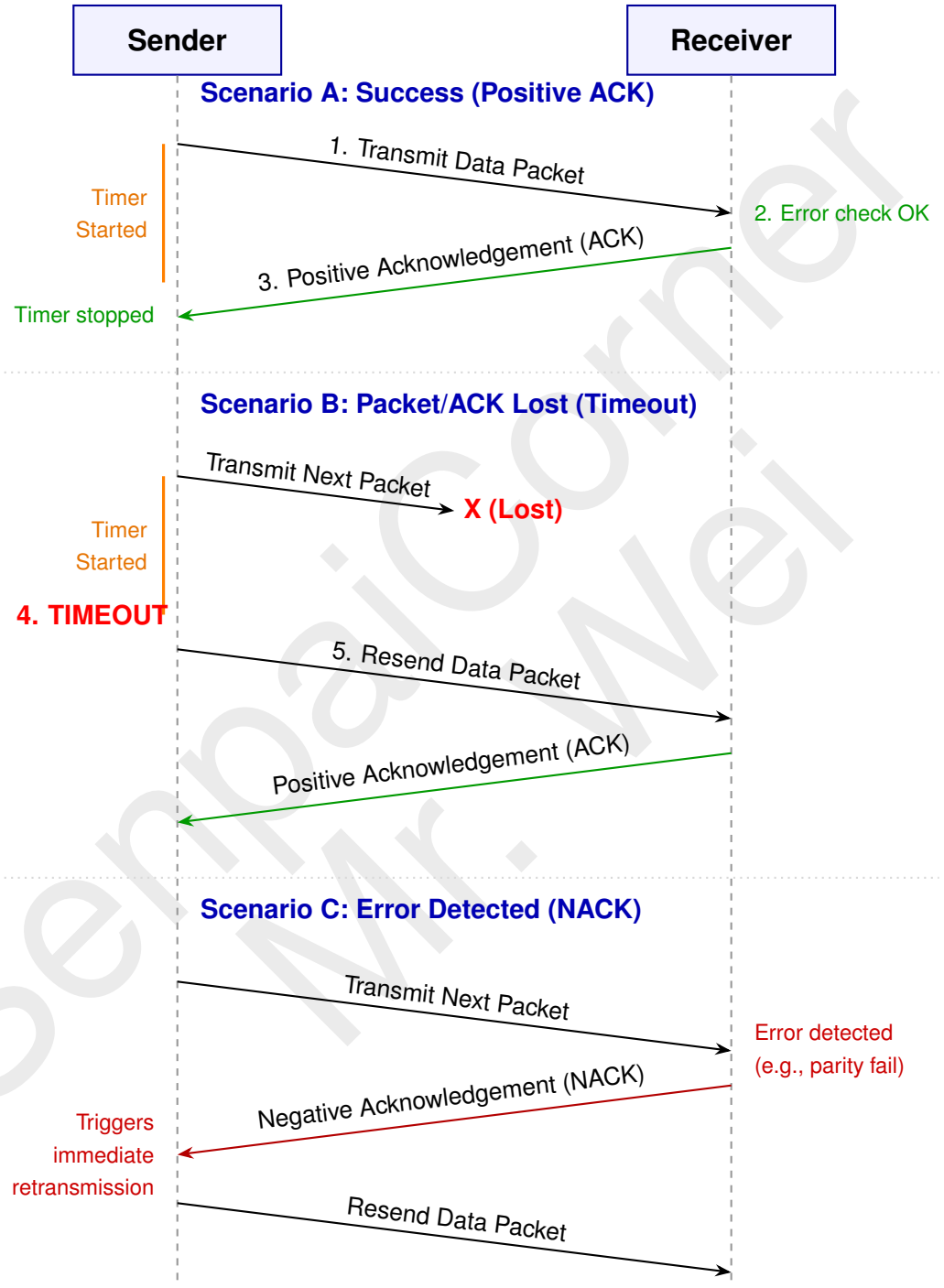
Check Digit vs Checksum

The check digit validates **data entry** (e.g. someone typing a barcode number). The checksum detects errors in **data transmission**. They share structural similarity — both calculate a value and compare it — but they address different failure modes.

Automatic Repeat Query (ARQ)

ARQ uses **positive and negative acknowledgements** combined with a **timeout** to ensure data is received correctly.

How ARQ operates using positive acknowledgement:



1. A timer is started when the sending device transmits a data packet.
2. The receiving device checks the packet for errors (e.g. using a parity check or checksum).
3. If the packet is error-free, a **positive acknowledgement** is sent back to the sender and the next packet is transmitted.
4. If the sending device does not receive an acknowledgement before the timer ends, a **timeout** occurs and the data packet is resent.
5. This continues until an acknowledgement is received or the maximum number of retransmission attempts is reached.

If an error is detected by the receiver, a **negative acknowledgement** is returned to the sender, triggering immediate retransmission.

ARQ combined with checksum:

- Checksum detects errors using a calculated value.
- ARQ handles retransmission using acknowledgement and timeout.
- Together: checksum identifies a corrupted packet; ARQ requests the packet be resent.

ARQ combined with parity check:

- Odd or even parity is agreed; a parity bit is added to each byte before transmission.
- After transmission, data is checked against the agreed parity.
- If no error is found, a positive acknowledgement is sent; if an error is found, a negative acknowledgement triggers retransmission.
- A timer and timeout operate in parallel, resending the packet if no acknowledgement arrives in time.

How Parity and ARQ Work Together — Mark Scheme Reasoning

A 6-mark "explain how parity checks and ARQ work together" answer needs *both* sides:

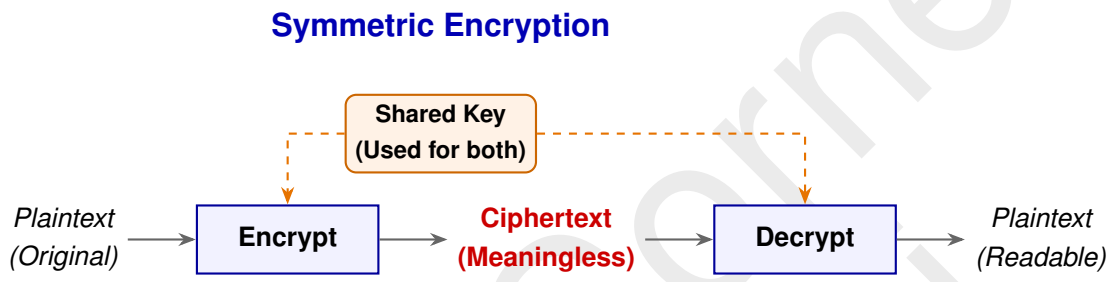
1. Parity side: agree odd/even; add parity bit to each byte; recount at receiver; detect mismatch.
2. ARQ side: positive acknowledgment if error-free; negative acknowledgment or timeout if error; retransmit until acknowledgment received or limit reached.

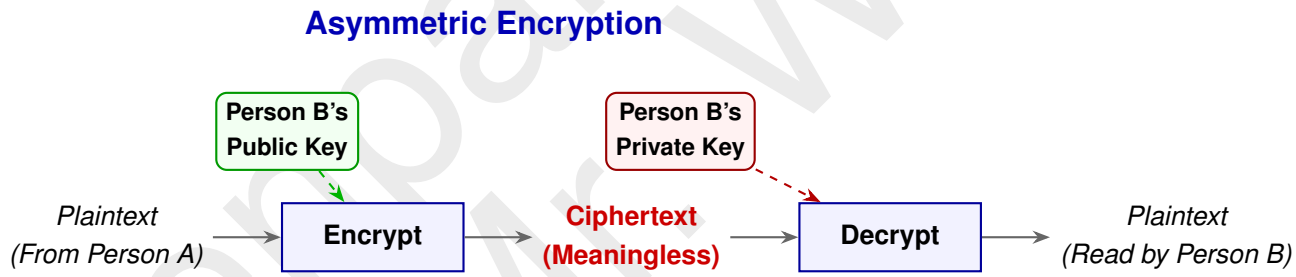
Award markers typically require at least two parity points and three ARQ points to reach full marks.

2.3 Encryption

Encryption scrambles/encodes data, making it meaningless to anyone who intercepts it.

- Uses an **encryption algorithm** and a **key**.
- **Plaintext**: the original, readable data before encryption.
- **Ciphertext**: the data after it has been encrypted.
- **Decryption**: the reverse process — converting ciphertext back into readable plaintext using a key.
- **Purpose**: makes data meaningless if intercepted during transmission.





Symmetric Encryption

- Uses the **same key** to both encrypt and decrypt the data.
- The sender encrypts data using the shared key.
- The receiver decrypts the ciphertext using the same key.
- **Drawback**: the key must be transmitted to the receiver, creating a security risk during key distribution.

Asymmetric Encryption

Uses a **public key** and a **private key**:

- **Public key** — known to all users; used to *encrypt* data.
- **Private key** — known only to the receiver; used to *decrypt* data.

How asymmetric encryption works:

1. Person A uses Person B's **public key** to encrypt the data.
2. Person A sends the ciphertext over the network.
3. Person B uses their **private key** to decrypt the data.

Advantage: only the private key can decrypt the message, and it is *never* sent over the network.

HTTPS and Asymmetric Encryption

When a browser connects to a secure website using HTTPS, asymmetric encryption protects the session. The web server provides its **public key**. The browser encrypts data using this public key. Only the server's **private key** can decrypt the data — so intercepted ciphertext cannot be read by a third party.

Improving Encryption Strength

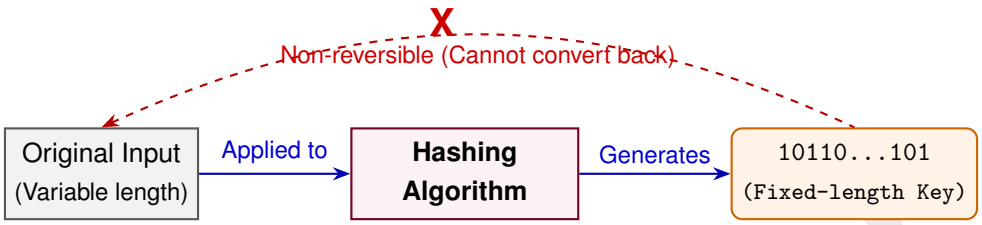
- **Increase key length** (e.g. use more than 128 bits) — generates far more possible key combinations; reduces the chance of decryption by brute force.
- **Use a more complex encryption algorithm**.

Symmetric vs Asymmetric — Quick Comparison

	Symmetric	Asymmetric
Keys used	Same key for encrypt and decrypt.	Public key to encrypt; private key to decrypt.
Key sharing risk	Key must be sent to receiver (security risk).	Private key never transmitted (more secure).
Exam keyword	"shared key"	"key pair" / "public and private key"

How Encryption Keys Are Created

A **key** is a binary string of a certain length which, when applied to an encryption algorithm, can encrypt plaintext and decrypt ciphertext.



- Keys can be created manually, randomly, or via an algorithm.
- Strong encryption keys are created using a **hashing algorithm** — a non-reversible mathematical algorithm that converts a given input into a fixed-length output. Once the output is generated, it *cannot* be converted back to the original input.
- A hashed key can be used symmetrically (shared between parties) or kept secret as an asymmetric private key.
- Both sender and receiver need a copy of the relevant key to decrypt information, regardless of whether symmetric or asymmetric encryption is used.