IGCSE Computer Science CIE

YOUR NOTES
↓

2. Data transmission

CONTENTS

YOUR NOTES
⬇

2.1 Types and Methods of Data Transmission

Data Packets

Data Packets

What are packets?

- Information on the internet is broken down into **packets** and are created by **TCP** and transmitted over the internet
    - Packets are small chunks of information/data
    - TCP stands for **Transmission Control Protocol** and is used for organising data transmission over networks
- Small chunks of data are easier and quicker to **route** over the internet than big chunks of data
    - **Routing** involves finding the most optimal path over a network
- Data can include anything from text, images, audio, video, animations, etc, or any combination of these

What do packets contain?

- Packets are "chunks" of information. This information is called the "**payload**"
- Packets act like postage letters, each one has:
    - a delivery address (**destination IP address**)
    - a return address (**source IP address**),
    - and a message (**data payload**)
- Packets are split into two parts, the **header** and **trailer**
- The header contains:
    - **Source IP**
    - **Destination IP**
    - **Payload** (the information)
    - **Packet number**
    - **Error checker** e.g. a checksum or parity bit
- The trailer contains:
    - Additional error checks
    - **End of packet notification**

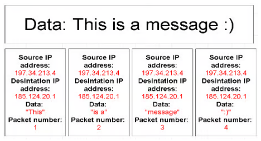
Data: This is a message :)
Source IP address: 197.34.213.4
Desintation IP address: 185.124.20.1
Data: "This"
Packet number: 1
Source IP address: 197.34.213.4
Desintation IP address: 185.124.20.1
Data: "is a"
Packet number: 2
Source IP address: 197.34.213.4
Desintation IP address: 185.124.20.1
Data: "message"
Packet number: 3
Source IP address: 197.34.213.4
Desintation IP address: 185.124.20.1
Data: ":)"
Packet number: 4

Figure 1: To transmit the message "This is a message :)" over the internet, the TCP might break the message down into 4 packets

- Each packet in Figure 1 contains a source IP address, destination IP address, payload (the data) and a packet number. Error checking and end of packet notifications have not been included in this example
- Individual packet structure depends on the **protocol** used to create them; variations exist

YOUR NOTES
⬇

- Error checks make sure that when a packet is received there is minimal or no **corruption** of the data
  - **Corruption** is where packet data is changed or lost in some way, or data is gained that originally was not in the packet
  - A parity bit checks that no bits have been flipped from 0 to 1 or vice versa
  - A checksum performs a calculation and compares the result to the checksum value. If the values are different then the data has been corrupted

How are packets sent across the internet?

- Sending packets over the internet is called **packet switching** and is more efficient than circuit switching
- Packet switching involves:
  - Breaking down a file into packets and sending these packets down different routes over the internet (via routers) from a source to a destination and reassembling them at the end
  - **Packet numbers** allow for the original message, which has been broken down into smaller parts, to be re-assembled in the correct order, or assembled like a jigsaw, once all of the packets have been received
  - Routers contain **routing tables** which keep track of nearby routers like a map or contacts list
    - Routers know which nearby router is closer to the destination device
    - Like normal car traffic, data traffic builds up on the internet. Routers can see this and decide to send a packet down a different route that avoids traffic.
    - Packets from the same message can take different routes from the sender to the receiver, and may arrive in different orders. The receiver's computer reassembles the message by reordering the packets using the packet numbers
  - If a packet does not reach its destination the receiver can send a **resend request** to the sender to resend the packet

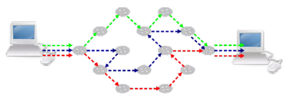

Figure 2: Packets take different routes across a network from the source address to reach the destination address as shown by the green, blue and red routes taken

- The advantages of packet switching are:
  - Interference and corruption are minimal as individual packets can be resent if they are lost or damaged
  - The whole file doesn't need to be resent if corruption occurs, only the individual packets that were corrupted need to be resent. This saves time and internet bandwidth
  - Packet switching is quicker than sending a large packet as each packet finds the quickest way around the network
  - It's harder to hack an individual's data as each packet contains minimal data, and travels through the network separately

YOUR NOTES
⬇

Worked Example

A local market shop wishes to arrange delivery of goods from a supplier. Anna, the shop owner, decides to send an email to request the delivery of the goods at a certain date and time.

Describe how packet switching is used to send this email and **how** it can be protected from corruption.

[8]

Answer:

- The business email is first broken down into packets which are given a source address (where its come from) and a destination address (where it's going to) [1 mark]
- Each packet receives a packet number so that the email can be reassembled when it reaches its destination [1 mark]
- Each packet also receives an error check such as a parity bit. A parity bit checks whether any bits have been flipped due to corruption [1 mark]
- Each packet is sent over the internet via routers. Routers contain routing tables that determine the next closest router to the destination [1 mark]
- Packets may take different routes depending on internet traffic and arrive at their destination in any order [1 mark]
- Packets are checked for errors using the error checks and missing packets can be requested to be resent [1 mark]
- Once all packets have been received then they can be put together in order using the packet numbers [1 mark]
- Once assembled the original email can be read by the other business [1 mark]

Exam Tip

For high marks make sure your answer is coherent, that is it follows logically from one point to the next. Some marks depend on previous points you have made. Explaining parity bits without mentioning error checking first may not gain you additional marks

YOUR NOTES
⬇

Data Transmission

Wired connections

- Wires can be:
  - Serial
    - **One bit** is sent at a time across a **single wire**

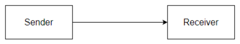

Figure 1: A sender sends a stream of bits in sequence, one after the one across a single wire

  - Parallel
    - **Multiple bits** are sent at a time across **several wires**.
    - Transmission is **asynchronous** as **some bits** may arrive **quicker** than others. This is known as **skewing** or **skewed** data.
      - Asynchronous transmission means data **does not always arrive at the same time**

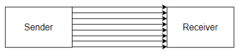

Figure 2: A sender sends a stream of bits in sequence, one after the other across multiple wires at the same time

  - Simplex
    - Simplex transmissions are **unidirectional** and travel in only **one direction**
  - Half-duplex
    - Half-duplex transmissions are **bidirectional** i.e. can travel in **both directions**, but **not simultaneously!**
  - Full-duplex
    - Full-duplex transmissions are **bidirectional** but can transmit signals in **both directions at the same time**
- Wires can be combinations of serial, parallel, simplex, half-duplex and full-duplex

	Simplex	Half-duplex	Full-duplex
Serial	Serial-Simplex	Serial-Half-duplex	Serial-Full-duplex
Parallel	Parallel-Simplex	Parallel-Half-duplex	Parallel-Full-duplex

Figure 3: Wire types can be combined between serial/parallel and simplex/half-duplex/full-duplex

  - Serial-Simplex
    - Data is transmitted **one bit at a time** in a **single direction** on **one wire**
  - Serial-Half-duplex

YOUR NOTES
⬇

        - Data can be transmitted in **both directions** on a **single wire** but only **one bit** at a time can be transmitted in **one direction** at a time
    - Serial-Full-duplex
        - Data can be transmitted in **both directions** at the **same time** on a **single wire one bit** at a time
    - Parallel-Simplex
        - **Multiple wires** transmit **one bit** at a time in **one direction**
    - Parallel-Half-duplex
        - **Multiple wires** send **multiple bits** of data in **both directions** but **only one direction at a time**
    - Parallel-Full-duplex
        - **Multiple wires** send **multiple bits** of data in **both directions** at the **same time**

Advantages and disadvantages of each method

	Advantages	Disadvantages
Serial	• Serial transmission is **cheap** over short and long distances as the cost of wire is fairly inexpensive	• Data transmission is **slow**, especially over long distances as only **small quantities of data** can be transmitted as a time • Serial transmission is expensive over very long distances as the cost of wire dramatically increases
Parallel	• Parallel transmission is **fast** as **large quantities of data** can be transmitted at any one time	• Parallel transmission is **expensive** over short distances as multiple wires need to be purchased. Transmission is very expensive over long distances as the cost of wires dramatically increases with the distance • **Delays** can be caused if data arrives asynchronously as the receiver has to wait for all of the bits before accepting new data. This is especially true over longer distances ◦ **Buffers** may be used to store data temporarily while waiting for all bits to arrive

YOUR NOTES
⬇

Simplex	• Simplex wires are **cheap** as only one wire is used	• Data transmission is **slow** as data still travels one bit at a time in only one direction at a time • Simplex transmission requires **two sets of wires** for **bidirectional transmission** meaning it can become expensive
Half-duplex	• Half-duplex transmission is **cheaper than simplex** for bidirectional transmission as it requires fewer wires	• Transmission is still **slow** as data travels one bit at a time in only one direction at a time
Full-duplex	• Full-duplex transmission is **faster** as data can travel in both directions simultaneously. The receiver does not have to wait for the sender to stop before they can start transmitting their data	• Full-duplex is **expensive** as the wire technology to transmit in both directions is more difficult to implement

Example scenarios of using each method

- Serial
  - Connecting an external hard drive to a computer
  - Transmitting data over a telephone line
- Parallel
  - Transmitting data from a computer to a printer using a multi-wire connector
- Simplex
  - Transmitting data from a computer to a printer. The printer doesn't need to send data back to the computer
    - Modern versions of devices such as printers may send acknowledgement signals to confirm they have received the data. This may require half-duplex rather than simplex connections
- Half-duplex
  - Phone conversations where only one person needs to speak at a time
  - A walkie-talkie is a two way radio with a push to speak button. The receiver is turned off while the transmitter is turned on. This prevents you from hearing the other person while you speak
- Full-duplex
  - Broadband connections to the internet. Data must be sent and received at the same time. Accessing information on the internet is known as downloading information. Putting information onto the internet for others to access is known as uploading
  - Phone conversations where both people can talk and be heard at the same time allowing them to interrupt each other

YOUR NOTES
↓

Worked Example

A company has a website that is stored on a web server

The company uses parallel half-duplex data transmission to transmit the data for the new videos to the web server.

Explain why parallel half-duplex data transmission is the most appropriate method.

[6]

Answer:

- Parallel would allow for the fastest transmission [1 mark]
- as large amounts of data [1 mark]
- can be uploaded and downloaded [1 mark]
- but this does not have to be at the same time [1 mark]
- Data is not required to travel a long distance [1 mark]
- Therefore skewing is not a problem [1 mark]

Exam Tip

1. Any four of these points qualifies as a full answer however make sure your answer is cohesive. Saying "Parallel would allow for the fastest transmission but this does not have to be at the same time" would qualify as one mark as only the first part makes sense and follows logically
2. When given context in the question, for example a web server that stores a website, your answer must make reference to this. Mark scheme answers such as "can be uploaded and downloaded" do not make sense without this context. You could lose marks by misreading the question and not providing a relevant answer

2.3 Encryption

YOUR NOTES
↓

Encryption

Encryption

- Many threats exist to system and network security. Examples include:
  - Malware
  - Viruses
  - Spyware
  - Hackers
  - Denial of service attacks
  - Social engineering
  - SQL injection
- **Hackers** are people who try to gain **unlawful or unauthorised** access to computers, networks and data by writing programs
- They look for **weaknesses** in the system and use them to gain access
- Hackers have various motives such as **financial gain, a challenge or protests** etc
- Hackers sometimes target data in order to **steal and use it**, or **block** people from using the data by creating programs called **ransomware**
- Hackers may also used packet sniffer to intercept and read data transmitted across the internet or a network
- Hackers will often want to use people's information and therefore it is beneficial to **encrypt** your data

What is encryption?

- Encryption involves **encoding data into a form that cannot be understood** using an algorithm
  - An example could be turning the phrase "Computer Science" into "YekLKEZizFuFjHNCjHj3Md7qyTiGxLNNwPVFZtJU74I="
- Once encrypted, data can be **decrypted** which turns the **encrypted data into data that can be understood again**
- **Encryption doesn't prevent hackers** from hacking but makes the data **hard if not impossible to read** unless they have matching decryption tools
- There are two types of encryption: **symmetric encryption** and **asymmetric encryption**

Symmetric and asymmetric encryption

YOUR NOTES
↓

- Encryption relies on the use of a **key**. A key is a **binary string** of a certain length that when applied to an encryption algorithm can encrypt **plaintext** information and decrypt **ciphertext**
  - **Plaintext** is the name for **data before it is encrypted**
  - **Ciphertext** is the name for **data after it is encrypted**
- Keys can **vary in size** and **act like passwords**, enabling people to protect information. A single incorrect digit in the key means the data cannot be decrypted correctly. Strong modern keys can be up to or over 1000 bits long!

Symmetric encryption

- In symmetric encryption both parties are given an **identical secret key** which can be used to encrypt or decrypt information
- **Key distribution problem:** If a hacker gains access to the key then they can decrypt intercepted information
- Methods exist to send the **secret key** to the receiver **without sending it electronically**:
  - Both parties could **verbally** share the key in person
  - Both parties may use **standard postage mail** to share the key (some businesses and banks may do this to ensure someone's identity and authenticity)
  - An algorithm may be used to calculate the key by **sharing secret non-key information**. An example is shown below.
  - **Symmetric Encryption Walkthrough**
    - Both parties A and B choose a number, for example **A = 3, B = 2**
    - Both parties enter their own respective numbers into the following equations: **7^A MOD 11 or 7^B MOD 11**. ^ is another way of writing "to the power of"
      - 7^3 MOD 11 = 2, 7^2 MOD 11 = 5
    - Both parties **swap their respective answers**. **A receives 5 and B receives 2**. These answers **replace the initial 7** number and the calculations are performed again
    - Both parties enter their **new number** into the following equations: 5^3 MOD 11 or 2^2 MOD 11
      - 5^3 MOD 11 = 4, 2^2 MOD 11 = 4
    - The answer should match for both parties and this becomes the encryption and decryption key value
- Once the **key is generated**, it can be **applied to the plaintext** in the algorithm that then **produces the ciphertext** which is sent to the receiver
- The receiver gets a **copy of the ciphertext and the key** and applies the encryption algorithm. The algorithm then produces the original plaintext for the receiver

Asymmetric encryption

- In asymmetric encryption also known as public key encryption, two keys are used:
  - **Public key: a key known to everyone**
  - **Private key: a key known only to the receiver**
- **Both keys are needed** to encrypt and decrypt information

YOUR NOTES
⬇

- Asymmetric encryption works as follows:
  - **Person A** uses a **symmetric key** to **encrypt their message** then **encrypts their symmetric key** using the **public key** known to both Person A and Person B
  - **Person A sends their message** over the network or internet
  - **Person B decrypts the symmetric key** using their **secret private key** then uses the **symmetric key to decrypt** the original message
- Asymmetric encryption works such that **only one private key can be used** to decrypt the message and it is **not sent over the internet** like a symmetric key
- Keys can be very large, for example over 1000 bits. To get the correct key a hacker would have to calculate almost every possible combination. To illustrate, a key with only 100 bits would generate 1,267,650,600,228,229,401,496,703,205,376 different combinations

How are encryption keys created?

- Encryption keys can be created **manually, randomly or via an algorithm**
- Strong encryption keys are created using a **hashing algorithm**
- A hashing algorithm is a **non-reversible mathematical algorithm** that converts a given input into an output. Once the output has been generated it is **unable to be converted back** to the original input
- Encryption keys are created by **supplying a message or key to the hashing algorithm** which turns it into a string of characters usually shown in **hexadecimal**
- **SHA-2** is an example of a hashing algorithm that creates hashed keys of 244, 256, 384 or 512 bit length
  - If the text string "Computer Science" is run through the SHA-2 algorithm, it would return a 512 bit key in hexadecimal as:
  - "B6e175f5fc647b1a9ce17019594ce55b58e8fd03e3c584ee384121c8b4c7753d"
- The hashed encryption key can then be **sent symmetrically or kept secret** as part of an **asymmetric private key**. **Both sender and receiver need a copy of the key** to decrypt information regardless of using symmetric or asymmetric encryption

Why use hashed encryption keys?

- In symmetric encryption, the key must be sent with the message to the receiver. If a **hacker intercepts the key they can read the message**
- In asymmetric encryption, the public key is available to everyone and would **not be useful to a hacke**r. The **hacker must guess the private key** in order to read the message
- Hashing algorithms are **many-to-one**. This means that **many input values, messages or keys can produce the same hash key output**
- A hashed encryption key means the hacker must first **unhash the key** before it is useful
- As hashing algorithms are **non-reversible** this is extremely difficult
- With SHA-2 for example, a hacker who wants to find the symmetric or asymmetric private key must calculate over 1.3x10^154 combinations; that is 13 with 153 0's after it. With the computing power available today, this is virtually if not actually impossible

YOUR NOTES
⬇

Worked Example

Complete the sentences about symmetric encryption. Use the terms from the list. Some of the terms in the list will not be used. You should only use a term once.

	algorithm	cipher	copied	delete	key plain
private	public	standard	stolen	understood unreadable	

The data before encryption is known as ______ text. To scramble the data, an encryption ______, which is a type of ______, is used. The data after encryption is known as ______ text. Encryption prevents the data from being ______ by a hacker.

[5]

Answer:

One mark for each correct term in the correct place.

- plain [1 mark]
- algorithm/key [1 mark]
- key/algorithm [1 mark]
- cipher [1 mark]
- Understood [1 mark]

2.1 Types and Methods of Data Transmission

USB

YOUR NOTES
↓

USB

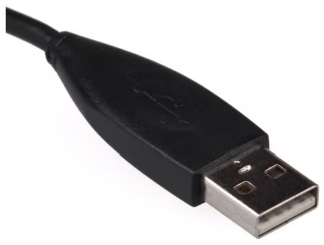
Figure 1: A USB-A connector

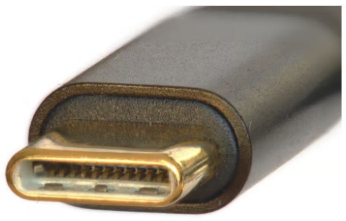
*Figure 2: A USB-C connector***

- The **Universal Serial Bus** (USB) is an **asynchronous** and serial method of transmitting data between devices and has become an industry standard
- Many devices use USB such as keyboards, mice, video cameras, printers, portable media players, mobile phones, disk drives, network adapters, etc
- Different USB connectors exist for different devices. Some examples are:
  - USB-A (flash drives, mice, keyboards, external HDD, etc)
  - USB-B (printers, scanners, optical drives, floppy drives, etc)
  - USB-C
- USB-C is becoming the new standard of USB due to its small size and speed
- When a device is connected to a USB port the computer is:
  - **Automatically detects** that the device has been **connected**
  - **Automatically recognised** and the **appropriate device driver is loaded** so that the **device can communicate with the computer**
    - If the device is new, the computer will look for a matching device driver. If one cannot be found then the user must download and install an appropriate driver manually

Advantages and disadvantages of USB

Advantages	Disadvantages
Devices are **automatically detected** and **drivers are automatically loaded** for communication. This simplifies the data transmission process for the user	The **maximum** cable length is **roughly 5 metres** meaning it cannot be used over long distances, **limiting its use**

Cable connectors fit in only one way. This prevents incorrect connections and **ensures compatible data transmission**	Older versions of USB have **limited transmission rates** for example **USB 2.0** has **480Mbps**
As USB usage is standardised, there is a lot of support available online and from retailers	**Very old USB standards may not be supported** in the near future (USB 1.1, USB 2.0, etc)
Several different **data transmission rates** are supported. The newest transmission rate as of 2022 is **USB4 2.0** with **80 Gbps** (81,920 Mbps, 170× faster than USB 2.0)	
Newer USB standards are **backwards compatible** with older USB standards	

Worked Example

Julia uses a USB connection to transfer data onto her USB flash memory drive.

(i) One benefit of using a USB connection is that it is a universal connection. State two other benefits of using a USB connection.

Benefit 1:

Benefit 2:

[2]

(ii) Identify the type of data transmission used in a USB connection.

[1]

Answer:

- (i) Any two from:
  - It cannot be inserted incorrectly [1 mark]
  - Supports different transmission speeds [1 mark]
  - High speed transmission [1 mark]
  - Automatically detected (not connected) // automatically downloads drivers [1 mark]
  - Powers the device (for data transfer) [1 mark]
- (ii)
  - Serial [1 mark]

2.2 Methods of Error Detection

Error Checking

YOUR NOTES
⬇

Error Checking

- When data has been received it could be subject to errors and Corruption

Why check for errors?

- Computers expect data in certain formats
  - A **format** is a way of **arranging the data** so that it can be easily understood by people and by computers
  - People agree to certain formats so that systems work more **efficiently** and there is little chance of misunderstanding each other
- An example of a format is **date and time**. Date and time can have multiple formats such as:
  - 13/04/14 (DD/MM/YY)
  - 12/31/2020 (MM/DD/YYYY)
  - Jul-04-16 (MMM/DD/YY)
- Computers usually perform processes and calculations on data. If the data is not as expected, things can go wrong. For example, if a receiver expected to receive a date in format DD/MM/YY as 03/04/17 but received 04/03/17, did the sender mean 3rd April 2017 or 4th March 2017?
- An **error or corruption** occurs when data received is **not as expected** and therefore is **difficult or impossible to process**

How is data represented?

- All data is represented in binary as 1's (**high voltage**) and 0's (**low voltage**)
  - For example, the number 67 is represented as 01000011
  - An error could cause one of the bits to **flip** from 1 to 0 or vice versa
  - If the sixth most significant bit is flipped, **01000011** -> **01000111**, 67 becomes 71
  - The number 67 in **ASCII** represents the **uppercase** letter **'C'** whereas the number 71 represents 'G'
  - By **flipping** a single bit, the meaning of the binary string has changed
  - If a book was transmitted over a network, assuming many bits were flipped, it would likely be difficult to read the original text

How can errors cause problems?

- Some errors are small and trivial such as a single swapped letter in a large text
- Other errors, in data such as postage addresses, aerospace coordinates or bank transfers, can be difficult to rectify or be disastrous
- Any job or task that relies on a computer to perform highly sensitive or secure processes must have methods of **error checking and correction**

How do errors occur?

- Errors can occur using wired or wireless technology due to **interference**
- Examples of interference include wire degradation or electrical fields changing the signal
- Results of interference include:

- **Data loss** - data is lost in transmission
- **Data gain** - additional data is received
- **Data change** - some bits have been changed or flipped

- Wireless technology uses **radio signals or other electromagnetic signals** to transmit data
  - These signals can be blocked by **physical barriers** such as buildings, walls, cars or other objects
  - Interference can be caused by **bad weather** such as rain or clouds, or by other **wireless signals** or **electromagnetic radiation**
- **Wired** technology carries more chance of causing an error as physical components can be **damaged, degrade or receive interference from outside signals**
  - Data loss can also occur from **interruptions to data transmission** such as a **blocked signal** or if the **transmission is intermittent**

Worked Example

Alex receives an email over a wireless connection from a work colleague containing an important document.

Identify what interference Alex could experience when sending this email and **identify** the outcomes of interference. Further **explain why** Alex should check to make sure the document contains no errors.

[4]

Answer:

- Weather conditions or physical barriers such as building can affect signals, for example bits could be flipped in the document making it hard to understand the original meaning [1 mark]
- Alex should be aware that interference can cause wirelessly received data to contain errors or corruption [1 mark]
- Data could be lost, additional data could be gained or data could be changed [1 mark]
- As Alex received an important work document they need to check for errors so that their work is unaffected and they do not receive incorrect information [1 mark]

YOUR NOTES
↓

Error Detection Methods

Parity check

- The parity checking **protocol** determines whether bits in a transmission have been **corrupted**
- Every **byte** transmitted has **one of its bits allocated as a parity bit**
- **The sender and receiver must agree before transmission whether they are using odd or even parity**
- If **odd parity** is used then there must be an **odd number of 1's** in the byte, **including the parity bit**
- If **even parity** is used then there must be an **even number of 1's** in the byte, **including the parity bit**
- The **value of the parity bit** is determined by counting the **number of 1's** in the byte, **including the parity bit**
- If the **number of 1's does not match** the **agreed parity** then an **error has occurred**
- Parity checks **only check** that an **error has occurred**, they **do not reveal where** the error(s) occurred

Even parity

- Below is an arbitrary binary string

EVEN Parity bit	Byte						
0	1	0	1	1	0	1	0

- If an **even parity bit** is used then **all bits** in the byte, including the parity bit, must **add up to an even number**
  - There are four 1's in the byte. This means the parity bit must be 0 otherwise the whole byte, including the parity bit, would add up to five which is an odd number

Odd parity

- Below is an arbitrary binary string

ODD Parity bit	Byte						
1	1	0	1	1	0	1	0

- If an **odd parity bit** is used then **all bits** in the byte, including the parity bit, must a**dd up to an odd number**
  - There are four 1's in the byte. This means the parity bit must be a 1 otherwise the whole byte, including the parity bit, would add up to four which is an even number
- The table below shows a number of examples of the agreed parity between a sender and receiver and the parity bit used for each byte

YOUR NOTES
⬇

Example #	Agreed parity	Parity bit	Main bit string								Total number of 1's
#1	ODD	0	1	1	0	1	0	1	1		5
#2	EVEN	1	0	0	0	1	0	0	0		2
#3	EVEN	1	0	1	0	1	1	1	1		6
#4	ODD	1	0	1	1	1	0	0	1		5
#5	ODD	1	1	0	1	0	1	0	1		5
#6	EVEN	0	1	0	0	1	1	1	0		4

- Example #1: The agreed parity is **odd**. All of the 1's in the main bit string are added **(5)**. As this number is **odd already** the **parity bit** is set to **0** so the whole byte **stays odd**
- Example #2: The agreed parity is **even**. All of the 1's in the main bit string are added **(1)**. As this number is **odd** the **parity bit** is set to **1** to make the **total number of 1's even (2)**
- Example #6: The agreed parity is **even**. All of the 1's in the main bit string are added **(4)**. As this number is **even already** the **parity bit** is set to **0** so the whole byte **stays even**

How do errors occur?

- When using parity bits, an **error** occurs when the number of **total bits does not match the agreed parity**
- Bits can be **flipped** or **changed** due to **interference** on a wire or wirelessly due to **weather or other signals**

Example #	Agreed parity	Parity bit	Main bit string							Total number of 1's	Error
#1	ODD	1	1	1	0	1	0	1	1	6	Error
#2	EVEN	1	0	0	0	1	0	0	0	2	No error
#3	EVEN	1	0	1	1	1	1	1	1	7	Error
#4	ODD	1	0	1	1	1	0	0	1	5	No error
#5	ODD	1	1	0	1	0	1	1	1	6	Error
#6	EVEN	0	1	0	0	0	1	1	0	3	Error

- Example #1: The agreed parity is **odd** but the total number of 1's is **even (6)**. An error has occurred somewhere
- Example #2: The agreed parity is **even** and the total number of 1's is **even (2)**. No error has occurred here

YOUR NOTES
⬇

- Example #3: The agreed parity is **even** but the total number of 1's is **odd (7)**. An error has occurred somewhere
- **Parity checks are quick and easy to implement but fail to detect bit swaps that cause the parity to remain the same**
- Below is an arbitrary binary string. The agreed parity is **odd** and the total number of 1's is **five (odd)**

Agreed parity	Parity bit								Total number of 1's	Error
ODD	1	0	1	0	1	0	1	1	5	No error

Agreed parity	Parity bit								Total number of 1's	Error
ODD	1	1	0	0	0	0	0	1	3	No error

Parity bytes and parity blocks

- **Parity checks do not pinpoint errors** in data, only that an **error has occurred**
- **Parity blocks** and **parity bytes** can be used to check an error has occurred and where the error is located
    - A **parity block** consists of a **block of data with the number of 1's totalled horizontally and vertically**
    - A **parity byte** is also sent with the data which contains the **parity bits from the vertical parity calculation**
- Below is a parity block with a parity byte at the bottom and a parity bit column in the second column

ODD	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	1	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	1	1	1	1

YOUR NOTES
⬇

- The above table uses **odd parity**
- Each **byte row** calculates the **horizontal parity** as a parity bit **as normal**
- Each **bit column** calculates the **vertical parity** for each row. This is the **parity byte**. It is **calculated before transmission** and **sent with the parity block**
- Each **parity bit tracks if a flip error** occurred in a byte while the **parity byte** calculates if an **error occurred in a bit column**
- By **cross referencing** both **horizontal and vertical** parity values the **error can be pinpointed**
- In the above example the **byte 3 / bit 5 cell** is the error and **should be a 0 instead**
- The error could be **fixed automatically or a retransmission request could be sent** to the sender

Exam Tip

Remember, parity bits only track if an error occurred, not where it is located. The parity bit itself might be the error. Parity bytes are calculated before transmission so act as a check on the parity bits themselves

Checksums

YOUR NOTES
⬇

- **Checksums determine** if data has been **corrupted** but **do not reveal where**
- Data is sent in **blocks** and an **additional checksum value is added at the end of the block**
- Checksums are **custom user-created algorithms** that **perform mathematical calculations** on data
- An example of a custom checksum algorithm in computer science is:
  - A **checksum byte** is defined as a value between **1 and 255** which is stored in 8 bits. **8 bits are collectively known as a byte**
  - If the **sum of all of the bytes** of a transmitted block of data is **<= 255** then the **checksum value** is the **sum of all of the bytes**
  - If the **sum of all of the bytes is > 255** then the checksum is calculated with an algorithm:
    - X = sum of all of the bytes
    - Y = X / 256
    - Round down Y to nearest whole number
    - Z = Y * 256
    - Checksum = X - Z

Custom Checksum Walkthrough

  - If X = 1496
    - Y = 1496 / 256 = 5.84
    - Rounded down Y = 5
    - Z = 5 * 256 = 1280
    - Checksum = 1496 - 1280 = 216
  - The **checksum value in this example would be 216**
- When a block of data is to be transmitted, the **checksum is first calculated** and **then transmitted** with the rest of the data
- When the **data is received** the **checksum value is calculated** based on the received data and **compared to the checksum value received**. If they are the **same** then the data **does not contain any errors**
- If an **error does occur** then a **resend request is sent** and the data is **retransmitted**

Exam Tip

It is worth noting that the checksum value itself may become corrupted or contain errors! Checksums may be sent multiple times and tallied. The most common checksum is then assumed to be the correct one

Echo check

- Echo checks involve **transmitting the received data back to the sender**. The **sender** then **checks the data** to see if any errors occurred during transmission
- This method **isn't reliable** as an **error** could have **occurred when the sender transmits the data or when the receiver transmits the data**. Neither will know when the error occurred.
- If an **error does occur** the sender will **retransmit the data**

YOUR NOTES
⬇

Worked Example

Four 7-bit binary values are transmitted from one computer to another. A parity bit is added to each binary value creating 8-bit binary values. All the binary values are transmitted and received correctly.

(a) Identify whether each 8-bit binary value has been sent using odd or even parity by writing odd or even in the type of parity column.

8-bit binary value	Type of parity
01100100	
10010001	
00000011	
10110010	

[4]

(b) An error may not be detected when using a parity check. Identify why an error may not be detected.

[1]

Answer:

- **8a**
  - Odd [1 mark]
  - Odd [1 mark]
  - Even [1 mark]
  - Even [1 mark]
- **8b**
  - Any one from: [1 mark]
    - there is a transposition of bits
    - it does not check the order of the bits (just the sum of 1s/0s)
    - even number of bits change
    - incorrect bits still add up to correct parity

Exam Tip

Do not add your own parity bit of 1 or 0. This has already been done for you

YOUR NOTES
⬇

Check Digits

- **Check Digits to determine** if data has been corrupted but **do not reveal where**
- Data is sent in **blocks** and an **additional check digit value is added at the end of the block**
- Check Digits are c**ustom user-created algorithms** that **perform mathematical calculations** on data
- An example of a check digit is the **ISBN** value on books:
  - Each book has a unique ISBN number that identifies the book
  - A standard ISBN number may be **ten digits**, for example, **965-448-765-9**
  - The **check digit value is the final digit (9 in this example)**. This number is **chosen specifically** so that when the **algorithm is completed** the **result is a whole number (an integer) with no remainder parts**
  - A **check digit algorithm** is performed on the **ISBN number**. If the result is a whole number then the ISBN is **valid**
- **ISBN Check Digit Walkthrough**
  - To calculate an ISBN check digit the following algorithm is performed on 965-448-765-9:
    - Multiply each ISBN digit by 1 to 10 and add them all up:
      - 9x1 + 6x2 + 5x3 + 4x4 + 4x5 + 8x6 + 7x7 + 6x8 + 5x9 + 9x10 = 352
    - Take the total number and divide it by 11: 352/11 = 32. 32 is a whole number with no remainder so the ISBN is valid

- Another example of a check digit is **barcodes** on purchasable items:
  - Barcodes consist of **black and white lines** which can be scanned using barcode scanners. Barcode scanners **shine a laser** on the black and white lines which **reflect light** into the scanner. The scanner reads the **distance between these lines as numbers** and can identify the item
  - Barcodes also use a **set of digits** to **uniquely identify each item**. The number of digits varies from code to code. An example would be **9780201379624**
  - The **final digit** on a barcode is usually the **check digit**. Like an ISBN this can be used to **validate and authenticate** an item
- **Barcode Check digit Walkthrough**
  - To calculate a barcode check digit we can perform the following example algorithm on **9780201379624** (the final digit is the check digit):
    - Multiply each digit by 1 or 3, alternating between these two values and add them all up
      - 9x1 + 7x3 + 8x1 + 0x3 + 2x1 + 0x3 + 1x1 + 3x3 + 7x1 + 9x3 + 6x1 + 2x3 = 96
    - The check digit (the final digit) should be the difference between the sum and the closest multiple of 10 that is larger or equal to the sum
      - 96 rounded to the nearest 10 = 100
      - 100-96 = 4
      - The check digit is therefore 4 which matches our original barcode number
- An example of a custom checksum algorithm in computer science is:
  - A **check digit byte** is defined as a value between **1 and 255** which is stored in 8 bits. **8 bits are collectively known as a byte**

YOUR NOTES
⬇

  - If the **sum of all of the bytes** of a transmitted block of data is **<= 255** then the **check digit value** is the **sum of all of the bytes**
  - If the **sum of all of the bytes is > 255** then the check digit is calculated with an algorithm:
    - X = sum of all of the bytes
    - Y = X / 256
    - Round down Y to a nearest whole number
    - Z = Y * 256
    - Checksum = X - Z
- **Custom Check digit Walkthrough**
  - If X = 1496
    - Y = 1496 / 256 = 5.84
    - Rounded down Y = 5
    - Z = 5 * 256 = 1280
    - Checksum = 1496 - 1280 = 216
  - The **check digit value in this example would be 216**
- When a block of data is to be transmitted, the **check digit is first calculated** and **then transmitted** with the rest of the data
- When the **data is received** the **check digit value is calculated** based on the received data and **compared to the check digit value received**. If they are the **same** then the data **does not contain any errors**
- If an **error does occur** then a **resend request is sent** and the data is **retransmitted**

YOUR NOTES
⬇

Worked Example

Check digit algorithms are used to determine whether an error has occurred in transmitted data.

a)
State the names of two examples of a checksum algorithm.

[2]

b)
Describe the process a check digit algorithm uses to determine if an error has occurred

[5]

Answer:

(a)

- ISBN [1 mark]
- Barcode [1 mark]

(b)

- Before data is transmitted a **checksum value** is **calculated** [1 mark]
- The checksum value is transmitted with the data [1 mark]
- The **receiver calculates** the **checksum value**using the received data [1 mark]
- The calculated checksum is compared to the transmitted checksum [1 mark]
- **If they are the same then there is no error otherwise an error has occurred [1 mark]**

ARQ

YOUR NOTES
↓

Automatic Repeat Request (ARQ)

- When the **receiver receives transmitted data** it must check for the **presence of any errors**. Errors can usually be **detected but not always pinpointed**
- An **Automatic Repeat Request** is a protocol that **notifies the sender** that an **error has occurred** and that the **data received is incorrect**. It works as follows:
  - If an **error is detected** the **receiver sends a negative acknowledgement** transmission to indicate the data is corrupted
  - If **no error is detected** the **receiver sends a positive acknowledgement** transmission meaning the data is correct
  - If the **receiver does not send any acknowledgement transmission** then the **sender waits** for a certain **time period known as a time-out** before **automatically resending** the data
  - This process is **repeated until all data has been received and acknowledged**