M&CPS (2 Part)

Oral Exam Paganelli:

1. ✅ Hidden Terminal

Wireless networks consist of **hosts which are** end devices running applications, often battery-powered. In general Wireless Networks may be based on the interaction hosts → base station or access point — or hosts → hosts.

There are two main modes:

- **Infrastructure mode:** hosts communicate via a base station or access point.
- **Ad hoc mode:** hosts communicate directly with each other (peer-to-peer).

CSMA/CD (Carrier Sense Multiple Access with Collision Detection) works poorly in wireless because:

- It detects collisions **while transmitting**, which works for wired but not wireless networks.
- The real problem is **interference at the receiver**, not at the sender.

This causes two classic issues:

- ***Hidden terminal problem:*** A node cannot detect another transmitter that interferes at the receiver.
- ***Exposed terminal problem:*** A node unnecessarily defers transmission due to sensing another nearby transmitter that wouldn't cause interference.

MACA stands for Multiple Access with Collision Avoidance

1. Stimulate the receiver into transmitting a short frame first
2. Then transmit a (long) data frame
3. Stations hearing the short frame refrain from transmitting during the transmission of the subsequent data frame

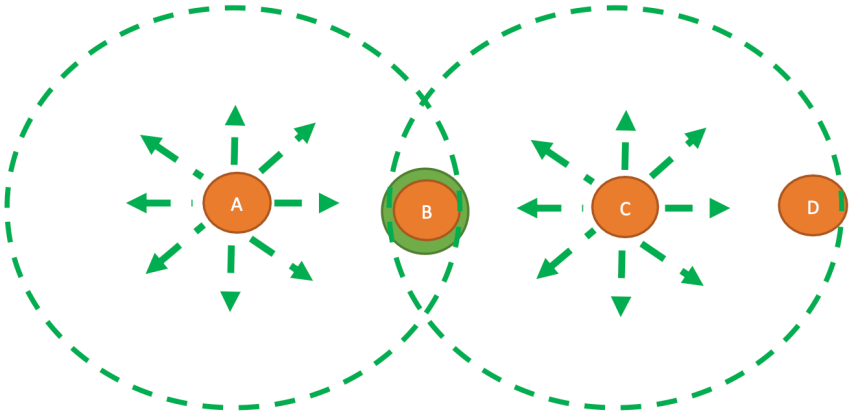
Basically, a transmitting node sends a Request to Send RTS and a receiving node answers with Clear to Send CTS.

Other nodes which hear CTS must stay silent until the transmission is over. Hearing RTS instead does not imply to stay silent, i.e. hearing a RTS does not forbid to transmit stuff to other nodes.

MACAW implements some improvements to MACA:

- ACK frame to acknowledge a successful data frame
- added Carrier Sensing to keep a station from transmitting RTS when a nearby station is also transmitting an RTS to the same destination
- Mechanisms to exchange information among stations and recognize temporary congestion problems. CSMA/CA used in IEEE 802.11 is based on MACAW.

Question:



Answer:

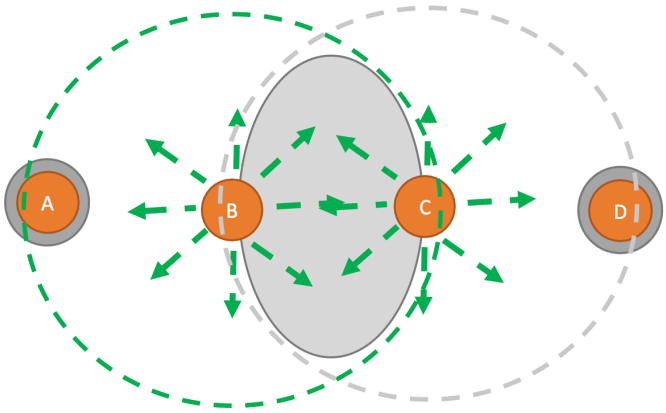
One of the challenges of terminals in wireless networks is the limited knowledge of terminals that are not in their range.

The **hidden terminal** problem is a problem that occurs when two or more stations that are respectively out of range of each other want to transmit simultaneously to the same device, thus generating collisions.

In the image it is possible to observe how node A turns out to be a hidden terminal for node C and vice versa during their respective communications with B. This happens because A, not being in the radio range of C, would not be able to detect a possible communication from C to B. Therefore, if A started to communicate with B while C is already communicating with B, this would lead to a collision in B.

2. ✅ Exposed Terminal

Question:



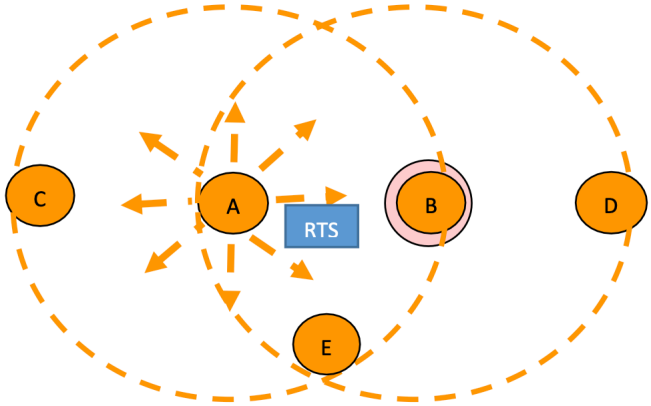
Answer:

The **exposed terminal problem** occurs in wireless networks when a node **unnecessarily defers** its transmission due to sensing nearby communication **that wouldn't actually interfere** with its intended transmission.

- B is transmitting to A.
- C wants to send to D, but C senses B's transmission.
- Since C hears B, it assumes the channel is busy and does not transmit.
- However, C → D communication would not interfere with B → A because:
  - A and D are out of each other's range.
  - The transmissions are on separate paths and do not collide.

3. ✅ RTS/CTS

Question:



Answer:

The **RTS (Request To Send) and CTS (Clear To Send)** mechanism is used within the MACA (Multiple Access with Collision Avoidance) protocol. In this protocol, in order to try to avoid (not solve) collisions and the hidden/exposed terminal problem, a short frame is sent before sending the data frame, so as to refrain other devices from sending messages.

The image shows the communication from node A to node B:

- Node A sends an RTS (Request-to-send) to node B, which is also received by nodes C and E. The RTS also contains the length of the data frame.
- Subsequently, B, ready to accept the message, responds to A with a CTS (Clear-to-send), which is received by nodes A, D and E.
- At this point, A can send the data frame to B.

However, this protocol does not solve the hidden/exposed terminal problem. In fact, C is an exposed terminal, since it receives the RTS of A but not the CTS of B: therefore C is free to transmit, but everything it receives will collide with the data sent by A; D is a hidden terminal because it receives the CTS from B but not the RTS from A, so D cannot transmit again until the data frame transmission is complete. Transmission that it will not hear. MACAW adds a final ACK —when data reception is complete— sent by B to overcome the problem, (right?).

What happens if B and C simultaneously send the RTS to node A? There is a collision between the RTSs, so a CTS is not generated. The solution to the problem is for B and C to use Binary Exponential Backoff to try to resend the RTS.

What happens if a node outside the range of A and B tries to communicate with E? E will not respond to the external node's RTS, since E has heard B's CTS. The external node will try (always using Binary Exponential Backoff) to resend the RTS until E responds

MACAW (MACA for Wireless networks) is an improvement of the MACA protocol: it adds ACK frame sent by the receiving node to acknowledge the received data, and other mechanisms for information exchange: the Data Sending frame, sent by the sender when it starts sending information; the RRTS (Request to RTS), sent by a node that has received an RTS, but was unable to respond due to another active communication.

4. ✅ Mobile Networks - I (Indirect Routing)

- **Mobile End Devices**
  - Phones or IoT devices that connect to mobile networks.
- **Base Station**
  - Connects the mobile device to the network.
  - Creates a private IP tunnel from the device to the network gateway.
  - Located at the edge of the carrier's network.
- **Home Subscriber Server (HSS)**
  - Stores user and device information (e.g., phone number, plan, permissions).
  - Helps with device authentication using the user’s home network.
- **Mobility Management Entity (MME)**
  - Authenticates the device with help from HSS.
  - Tracks the device’s location and manages handovers (when a device moves between cells).
  - Sets up a data path from the device to the PGW (gateway to the internet).
- **Serving Gateway (SGW)**

    - Forwards data between the radio network and the internet gateway.
    - Manages handovers between base stations and networks (e.g., 4G ↔ 3G).
- **Packet Data Network Gateway (PGW)**
    - Connects the mobile network to the internet.
    - Works like a router and provides NAT (Network Address Translation).

Control Plane – Decides how data should move through the network (e.g., routing, authentication).

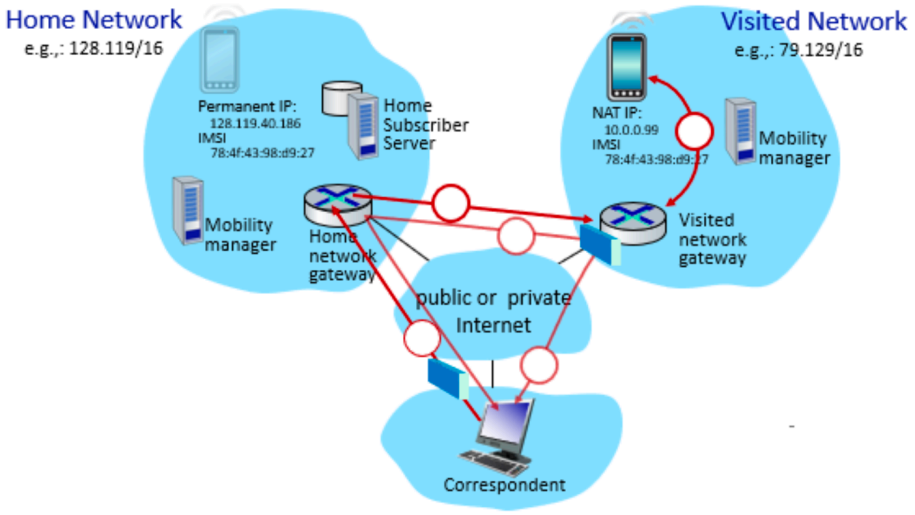
Data Plane – Actually moves the data between devices.

"Think of the control plane as being like the stoplights that operate at the intersections of a city. Meanwhile, the data plane (or the forwarding plane) is more like the cars that drive on the roads, stop at the intersections, and obey the stoplights"

Direct Routing – The sender sends data straight to the mobile device using its current IP.

Indirect Routing – Data first goes to the mobile device's home network, then is forwarded to the current location.

Question:



Answer:

The image shows how **indirect routing** for mobility works between a home network and a visited network. The home network represents the “native” network of a device with a subscription to a mobile provider (e.g. Verizon), where information about sub scribed devices is saved in the HHS (Home Subscriber Server). The visited network is any other network managed by a provider other than the one the device is subscribed to.

Three Required Protocols for Indirect Routing:

1. **Mobile-device-to-visited-network association protocol**
    - Allows the device to connect to a visited network and disconnect when it leaves.
2. **Visited-network-to-home-network-HSS registration protocol**
    - allows the visited network to register the location of the mobile device within the HSS of its home network.
3. **Datagram tunneling protocol**
    - Used to forward data from the home network’s gateway to the visited network’s gateway.
    - Includes encapsulation, NAT translation, and decapsulation.

🧭 How Indirect Routing Works – Step-by-Step:

1. The sender (Correspondent) uses the home address of the device it wants to communicate with as the destination address of the datagram.
2. The home network gateway receives the datagram, and forwards it (using the previously described tunneling protocol) to the visited network gateway, after consulting the HSS to discover the location of the mobile device.
3. The visited network gateway forwards the datagram to the mobile device (typically after using NAT)
4. The visited gateway router forwards the response to the sender (Correspondent) directly (b) or using the home network (a).

⚠️Indirect Routing Issue:

Indirect routing is inefficient in the case where a device B is a mobile device that has moved from its home network to the same network as device A (sender, which in this case has a different home network). Although both devices are now in the same network, the routing of messages from Device A to Device B does not occur directly. Instead, it follows the 'triangle' path of indirect routing, leading to unnecessary overhead.

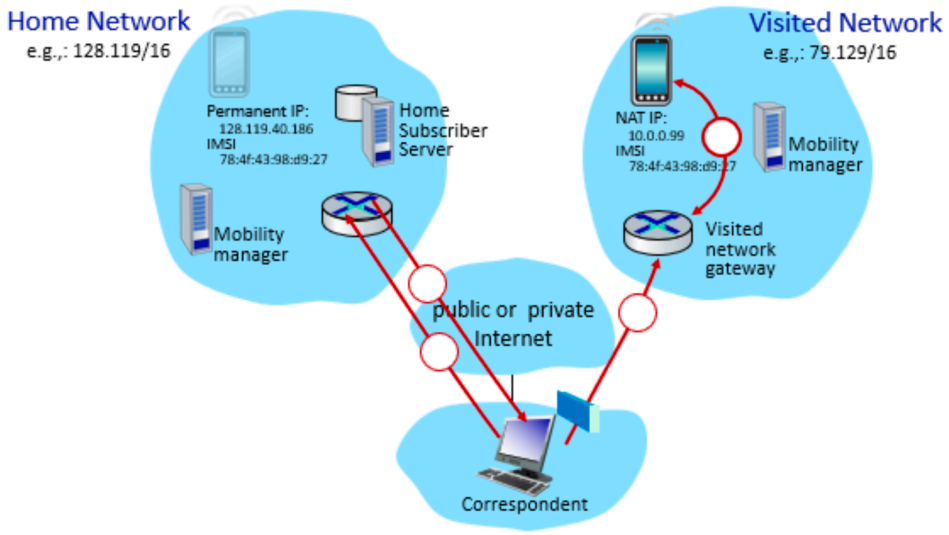
🆔 IMSI and Permanent IP

- **IMSI (International Mobile Subscriber Identifier)**: a unique number used to identify the user/device (like a MAC address).
- **Permanent IP**: the home network's IP address assigned to the device.

  Used for communication.
- **IMSI**: used for billing, authentication, and user identification.

5. ✅ Mobile Networks - II (Direct Routing)

Question:



Answer:

The image shows how **direct routing** for mobility between a home network and a visited network works. The home network represents the "native" network of a device with a subscription to a mobile provider (e.g. Verizon), where information about subscribed devices is stored in the HHS (Home Subscriber Server). The visited network is any other network managed by a provider other than the one to which the device is subscribed.

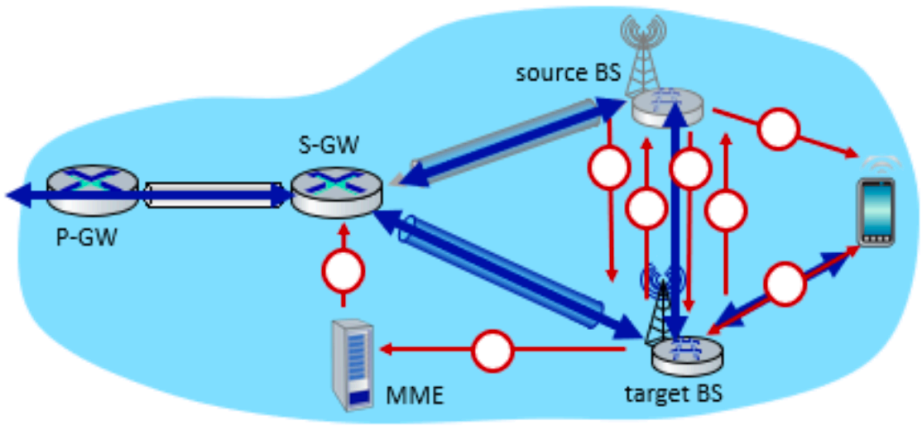
Unlike indirect routing, in which communications between Correspondent and mobile device pass through the home network, using direct routing the sender (Correspondent) directly sends the datagram packet to the mobile device in the visited network:

1. The sender (Correspondent) contacts the home network of the mobile device to which he wants to send a message through its permanent IP address.
2. The home network responds to the sender (after asking the HSS) with the IP assigned to the mobile device in the visited network.
3. The sender (Correspondent) sends the message to the mobile device using the IP just obtained.
4. The visited network gateway forwards the datagram to the mobile device (typically after using NAT)

Direct routing solves the problem of 'triangle routing', but at the cost of not being transparent to the Correspondent, since he must obtain the address assigned to the mobile device by the visited network. In indirect routing the Correspondent does not have to manage the mobility of the mobile device (which can also occur multiple times in a short period), making indirect routing the most used protocol.

6. ✅ Mobile Networks - III

Question:



Answer:

The image shows how a **phone connection (call or data)** is transferred between **two base stations** in the **same mobile network** in a **4G system**.

This process is called **handover**: it means **moving an active connection** from one base station to another **without losing the communication**.

Handover is needed when a **device moves** from one **cell** to another, and the signal from the current base station becomes **too weak** to keep the connection going.

🔁 The 7 Steps of Handover

The image shows **7 steps** that happen during the handover:

---

1. 📡 The current base station (BS) starts the handover process

- This happens if:

    - The **signal gets weak**, or
    - The base station is **overloaded**.
- The current BS chooses a **target BS** (the one that will take over).
- It sends a **Handover Request** to the target BS.

---

2. ⏱️ The target BS prepares for the handover

- It **pre-allocates** some **radio time slots** for the device.
- It replies with a **Handover Request ACK**, which contains info for the mobile device.

---

3. 📲 The current BS informs the mobile device

- It tells the mobile phone that it can now **connect to the new BS**.
- From the **mobile phone's point of view**, the handover is now complete.

---

4. 🔄 Switching the data path

- The old (source) BS **stops sending data directly to the device**.
- Instead, it **forwards the data** to the new (target) BS, which **then sends it to the device**.

---

5. 🧭 Update in the core network

- The **target BS** informs the **MME** (Mobility Management Entity) that it's now the BS for the device.
- The **MME** tells the **S-GW** (Serving Gateway) to update the **data tunnel**, so data is now routed to the **new BS**.

---

6. ✅ Confirmation of handover completion

- The **target BS** sends a confirmation (**ACK**) to the **source BS** to confirm the handover is complete.

---

7. 📡 Data from the device now goes through the new BS

- All **data sent by the mobile device** is now managed by the **new BS**, and goes through the updated **tunnel to the S-GW**.

7. ✅ SDN - Generalized Forwarding

Software-Defined Networking (SDN) is a modern way to design and manage computer networks. It separates the **"control plane"** (where decisions are made) from the **"data plane"** (where data is actually moved).

Traditionally, each router or switch made its own decisions about where to send data. With SDN, there is a **central controller** — like the "brain" of the network — that makes all the decisions, and the devices (routers, switches) just follow the instructions.

Motivation and Traffic Patterns

Modern distributed applications often use **many databases and servers** that need to **talk to each other**. This creates a lot of **"horizontal" traffic** (server-to-server), not just the usual **"vertical" traffic** (client-to-server). At first, this server-to-server traffic was **not very important**, but now it has become a **big part** of total network traffic.

Unified Communications (UC) are used more and more inside companies. Many tools like **chat**, **status updates (presence)**, **voice**, **mobile features**, **audio**, **web**, and **video** must now work together as **one single service** on the same platform.

With the growth of **cloud services**, a lot of company traffic is now sent to **remote cloud servers**. This puts **unexpected load** on company routers, which now handle more traffic than before.

Server virtualization is now very common. It increases the number of **virtual machines (VMs)** on each physical server. This also increases communication between VMs, especially the **"east-west" traffic** (VM-to-VM inside the datacenter), for things like **VM migration** or **remote storage access**.

Summary, Now, up to **70% of datacenter traffic** is **East-West** (server-to-server), and the **old three-layer network architecture** (access → aggregation → core) can no longer handle the **high bandwidth** needed for this kind of traffic.

Layering, as used in the **TCP/IP stack**, means breaking down data delivery into basic parts. This allows each part (or layer) to improve independently while still working together. This idea has worked very well. However, many problems still exist in the **Network Control and Management Plane**.

In **computer science**, we often use clear and simple ideas called **abstractions**. But **network control** is different — it depends a lot on **hardware** and complex, wordy **protocols**. There are no clear rules or models to guide how it works, which makes it hard to manage **traffic flows**.

SDN (Software-Defined Networking) helps by giving an **abstraction** — it offers a **centralized view** of the network to make control easier.

Network Layer

- **Forwarding:** Moves packets from a router's input to the correct output.

  → This happens in the **Data Plane**.

- **Routing:** Decides the path that packets take from the source to the destination.

  → This happens in the **Control Plane**.

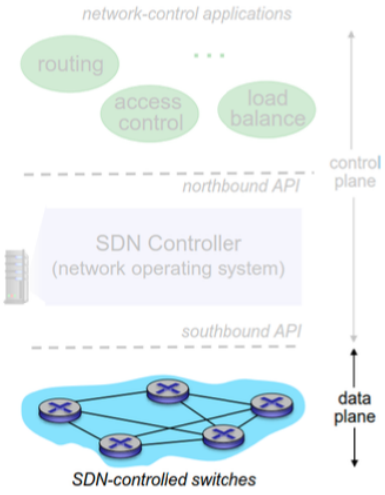


Figure 18.2: SDN Layer Architectures

SDN Architecture:

(Data Plane) Switches are fast, simple, and low-cost devices. They use hardware to handle general data forwarding and usually offer an API to control forwarding tables — for example, using **OpenFlow**.

The **SDN controller** (also called the network operating system) keeps track of the network's state and talks to:

- **Network control applications** (above it) using a **northbound API**
- **Switches** (below it) using a **southbound API**, like **OpenFlow**

SDN controllers are built as **distributed systems** to improve **performance**, **scalability**, **fault-tolerance**, and **reliability**.

(Control Plane) Network control applications are the "brains" of the system. They carry out control tasks by using services provided by the SDN controller. These apps are **unbundled**, meaning they can come from a **third party**, not just from the same company that made the SDN controller or routing hardware.

Data Plane

This is the **resource or infrastructure layer** that gives a **Forwarding Abstraction**. It includes forwarding devices that **do not have their own software to make decisions**. Instead, they move and process data based on decisions made by the **SDN control plane**.

Inside the routers, there is an **OpenFlow flow table**. This uses a pattern called **"match + action"**, which defines how packets are handled. For each packet, certain bits are checked against the rules in the flow table. When a rule **matches**, the related **action** is taken.

Some basic forwarding rule components are:

- **Match**: Header values in the packet that the router looks for
- **Actions**: What to do with a matching packet — for example, **drop**, **forward**, or **modify** it
- **Priority**: Helps choose the correct rule when multiple rules could match
- **Counters**: Keep track of the **number of packets or bytes** matching a rule

OpenFlow to control the Network Device

Data Plane Network devices, in addition to rule-based forwarding, also **communicate with the SDN controller** to manage forwarding rules using

the **OpenFlow switch protocol**.

With OpenFlow, the controller can **add**, **update**, or **delete** flow entries in the flow tables. This can be done:

- **Reactively** – in response to incoming packets
- **Proactively** – in advance, based on expected traffic

A **switch** may have **one or more flow tables** connected in a **pipeline**, allowing for more flexible processing.

Each flow table can include **instructions** that tell the switch what to do next. For example:

- Use a **"Goto" instruction** to move the packet to the next flow table
- Finally, **send the packet to an output port**

If a packet doesn't match any rule, it can also be **sent to the controller**, which can then:

- Create a **new flow rule** for that kind of packet
- Or decide to **drop** the packet

Control Plane:

The **OpenFlow protocol** works between the **SDN controller** and **switches**, using **TCP** (optionally with encryption).

There are **three types (classes)** of OpenFlow messages:

---

1. Controller-to-Switch Messages

Messages sent **from the controller to the switch**:

- **Features**: The controller asks the switch about its capabilities; the switch replies.
- **Configure**: The controller gets or sets the switch's configuration settings.
- **Modify-State (FlowMod)**: The controller **adds**, **deletes**, or **updates** flow entries in the switch's OpenFlow tables.

- **Packet-Out**: The controller tells the switch to send a specific packet out of a chosen port.

---

2. Asynchronous Messages

Messages sent **from the switch to the controller** (without being asked):

- **Packet-In**: The switch sends a packet to the controller (when no rule matched it).
- **Flow-Removed**: Notifies the controller that a flow rule was removed from the switch's table.
- **Port Status**: Informs the controller when there is a change (e.g. up/down) on a switch port.

---

3. Symmetric Messages

These are **miscellaneous** messages, sent in either direction without being triggered by an event.

Routing

In traditional networks, **routing** is done in a **distributed way** — each router makes its own decisions. But in **SDN-controlled networks**, it's better to **centralize the routing function** in the **SDN controller**. This allows the controller to have a complete and consistent view of the network, so it can calculate **shortest paths** and apply **application-aware routing policies**.

Because of this, **data plane switches don't need to handle routing themselves**. This reduces their processing and memory load, which leads to **better performance**.

The **centralized routing application** (part of the SDN controller) does two main things:

- **Link/Topology Discovery**
  - The routing function needs to be aware of links between data plane switches
  - In OpenFlow networks, this discovery is **not yet standardized**.

- **Topology Manager**
  - Keeps an up-to-date map of the network's structure (topology).
  - Calculates routes in the network, like the **shortest path** between two switches or between a switch and a host.

Topology Discovery

Each **OpenFlow (OF) switch** is pre-configured with the **IP address and TCP port** of the **SDN controller**, so it can connect to the controller as soon as it turns on. The switch also has a **preinstalled flow rule** that tells it to **send all LLDP (Link Layer Discovery Protocol) messages** to the controller using a **Packet-In message**.

LLDP is a protocol used to **discover directly connected (neighbor) devices**. It works by **advertising a switch's identity and capabilities** to its neighboring switches. Switches send **LLDP frames** regularly, based on a timer, to help the **controller discover the network topology**.

LLDP Message Contents

- **Chassis ID**: ID of the switch sending the LLDP packet
- **Port ID**: ID of the port that sends the LLDP packet
- **TTL (Time To Live)**: Time (in seconds) the info in the packet remains valid
- **End of LLDPDU**: Marks the end of the data in the LLDP packet

---

Discovery Process

1. The **controller sends a Packet-Out** message for **each active port** on every discovered switch. Each message contains an **LLDP packet**.
2. The switch **forwards the LLDP packet** out through the correct **Port ID**, sending it to any connected neighbor switches.
3. When a neighboring switch **receives the LLDP packet on a port** (that is **not the controller port**), it wraps it in a **Packet-In** message and **sends it back to the controller**. This message includes **metadata** such as:
   - The **Switch ID**

- The **Port ID** where the LLDP packet was received

4. When the **controller gets the packet back**, it reads the info and **learns that a link exists** between the two switches.

Google B4 WAN

Google has **two main backbone networks**:

- B2: Handles traffic that goes to and from the **public Internet**
- B4: Handles **inter-datacenter traffic** (traffic between Google's own data centers) — and this network has **grown a lot in recent years**

Why B4 was needed:

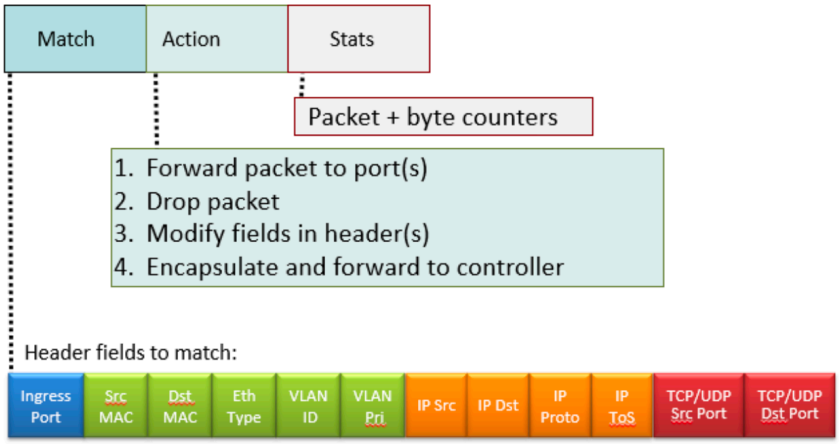
- It is **not connected to the public Internet**, so it's more secure and controlled
- It provides a **cost-effective way** to handle large volumes of traffic
- It is designed for **bursty** or **bulk traffic** — traffic that comes in large spikes or big batches

---

Types of traffic (flows) on the backend networks:

- **User data copies** — copying user information between data centers
- **Remote storage access** — accessing storage located in another data center
- **Large-scale data push** — syncing data or state across multiple data centers

Question:



Answer:

Generalized Forwarding (used in OpenFlow)

Generalized forwarding is one of the services provided by the **SDN data plane**.

It gives each router a **forwarding table** (also called a **flow table**) that uses the **"match + action" abstraction**.

This approach checks **different fields in the packet header** to decide what action to take.

Possible actions on a packet or flow are:

- **drop** (discard the packet)
- **forward** (send it to the next hop)
- **modify** (change something in the header)
- **send to controller** (for further processing)

A diagram (mentioned in the original) shows the different header fields that can be used to define **matching rules** in a flow table:

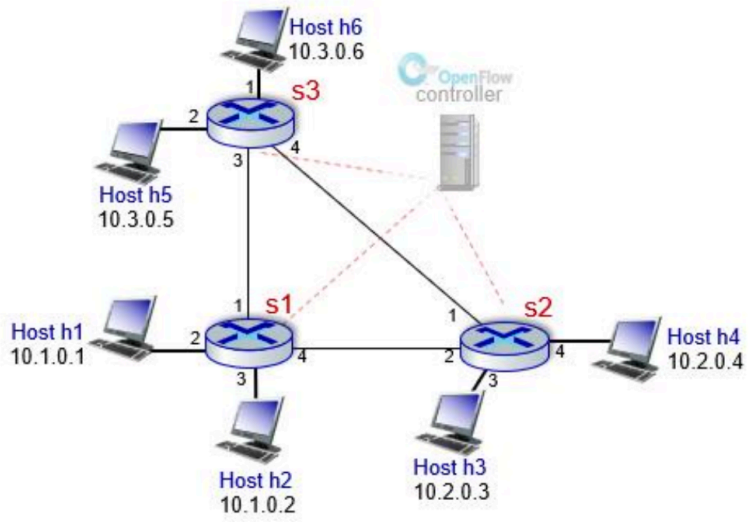
- **Link layer fields** (in green)
- **Network layer fields** (in orange)
- **Transport layer fields** (in red)

The "match + action" abstraction allows SDN to unify different types of devices, depending on how they behave:

- **Router**:
  - **Match**: longest matching IP prefix
  - **Action**: forward the packet to a specific link
- **Switch**:
  - **Match**: destination **MAC address**
  - **Action**: forward or flood (send to all ports) the packet
- **Firewall**:
  - **Match**: **IP address** and **TCP/UDP port number**
  - **Action**: allow or block the packet

8. ✅ SDN - Example

Question:



Answer:

Example of small SDN network with **3 switches** and **6 hosts** in total. The **SDN controller** (specifically an OpenFlow controller) will program the switches by installing a **flow table** in each switch. This table contains all the forwarding rules needed to manage the network traffic.

Key Difference from Traditional Networks

The important difference compared to traditional networks is the **separation of the control plane and data plane**.

- In traditional switches, the control and data planes are inside each device.
- In SDN, they are separated, allowing **centralized control** by the SDN controller, instead of programming each switch individually.

Example: Forwarding Rules for Traffic from Hosts h5 and h6 to Hosts h3 and h4 via switch s1

- **Switch s3:**
  - **Match:** Source IP = 10.3.., Destination IP = 10.2..
  - **Action:** Forward out port 3
- **Switch s1:**
  - **Match:** Incoming port = 1, Source IP = 10.3.., Destination IP = 10.2..
  - **Action:** Forward out port 4
- **Switch s2:**
  - **Match:** Incoming port = 2, Destination IP = 10.2.0.3
  - **Action:** Forward out port 3
  - **Match:** Incoming port = 2, Destination IP = 10.2.0.4
  - **Action:** Forward out port 4

Important OpenFlow Commands:

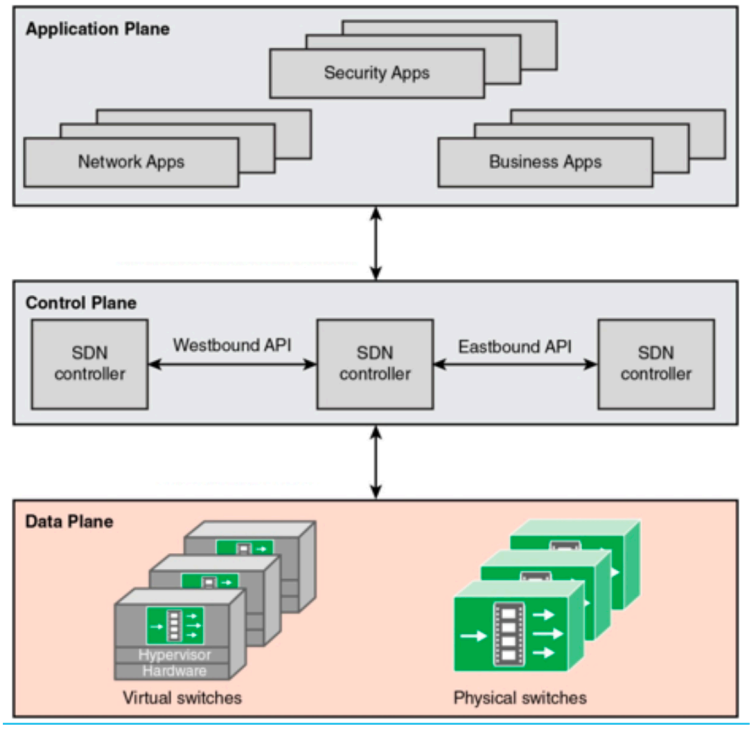
- **features**: ask switch for its capabilities
- **flowmod**: add, delete, or change entries in the OpenFlow tables
- **packet-in**: send a packet from a switch to the controller for handling
- **packet-out**: tell a switch to send a packet out of a specified port (can be a packet previously received with packet-in)

How LLDP Uses These Commands:

1. The controller **creates an LLDP packet** for every active port on every switch and sends it with **packet-out**.
2. A switch receiving this LLDP packet from the controller **forwards it to neighboring switches**.
3. A switch that **receives an LLDP packet from another switch** wraps it in a **packet-in** message and sends it to the controller, adding metadata like Switch ID and Port ID.
4. The controller uses this information to **build the map of links** between switches.

9. ✅ SDN - Architecture

Question:



Answer:

SDN Architecture Overview

The image shows the **SDN architecture**, starting from the ***data plane***, where the actual **forwarding of frames** happens

Next is the ***control plane***, where decisions are made about **where to send the traffic**.

Finally, there is the ***application plane***, which sits above the control plane and provides a higher-level **abstraction** that makes network management easier and more intuitive — usually using a **declarative approach**.

Data Plane Architecture (highlighted in the image)

The data plane is mainly made up of **physical switches** and **virtual switches**.

Each switch must implement a **forwarding model (or abstraction)** that follows instructions given by the **SDN controllers**.

Communication between Controller and Switches

The controller communicates with data plane switches using a **standard protocol** called **OpenFlow**.

This protocol lets the controller send commands on **Layer 2** to program the switches to perform specific actions.

One common pattern used is the **"match + action"** rule, but OpenFlow can do more than that.

Key Features of the Data Plane

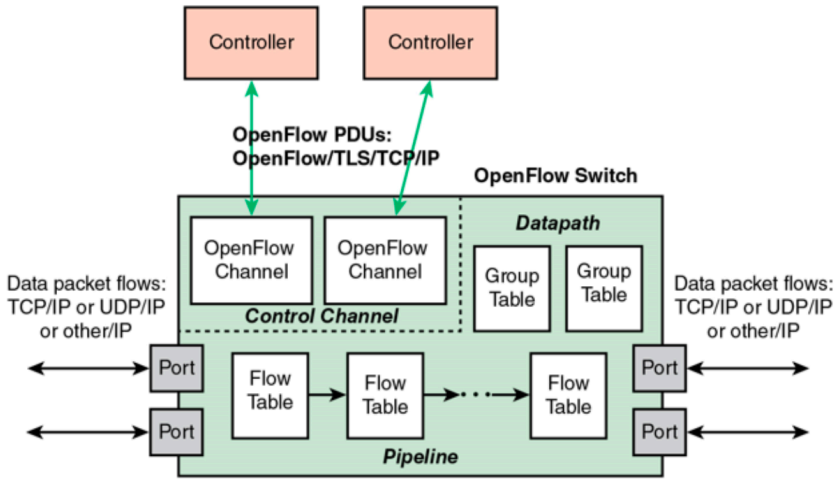
- **Packet forwarding**: Switches forward packets based on rules from the controller.
- **Traffic shaping**: Switches can prioritize certain types of traffic (e.g., voice or video) so important traffic is sent without delay.
- **Security**: Switches can apply security policies to block unauthorized access, like filtering packets from known dangerous addresses.
- **QoS (Quality of Service)**: Switches ensure certain traffic types get a guaranteed level of service quality.
- **Load balancing**: Switches distribute traffic efficiently to avoid congestion and improve network performance.

SDN uses different APIs to connect its layers:

- From **control plane to data plane**, we have **southbound APIs** (like OpenFlow). These let the control plane get information about the traffic and tell switches how to forward packets.
- Controllers provide **northbound APIs** that allow developers and network managers to create and apply custom network applications.

10. ✅ OpenFlow Switch

Question:



Answer:

OpenFlow switches are **network devices** (physical or virtual) that can be **programmed by an OpenFlow controller** using OpenFlow commands.

They use **flow tables** to manage network traffic and are made of the following components:

- A set of **flow tables**, organized in a pipeline.
- A set of group tables inside the datapath, which are used to define more complex behaviors. These allow rules based on **groups of flows** (using flow attributes).

Communication with Controller

- **OpenFlow Channels** allow the switch to **communicate with the controller** (in the control plane).
- Usually, a switch connects to **multiple controllers** to support **distributed control**.

  The **simplest setup** is **master-slave**, where only one controller is active.

  More complex setups may use multiple controllers in **active-passive redundancy** mode.

Switches have different types of ports:

multiple I/O ports for packet flows into and out of the device.

- **Physical ports**
- **Logical ports** (virtual versions of physical ports)
- **Reserved ports**

Flow Table Pipeline

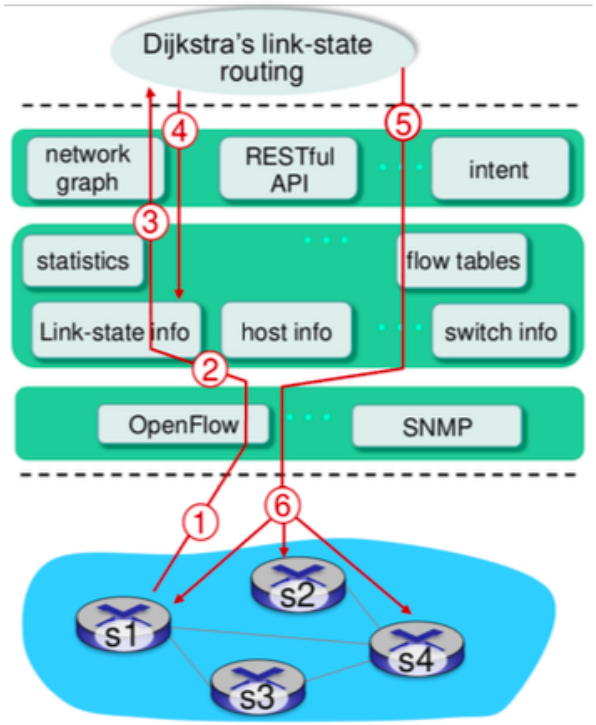
- The **flow tables are connected in a pipeline** inside the switch.
- When a packet arrives, it is checked against the **first table**.
- It continues through the pipeline **until a match is found**.
- If **multiple matches** exist in a table, the one with the **highest priority** is used.

If the only match is the **table-miss entry** (i.e., no specific rule applies), then one of three things can happen:

1. The packet is **sent to the next flow table**, if it exists.
2. The packet is **sent to the controller**, which will decide — based on its policies — to create a new flow rule or drop the packet.
3. The packet is **dropped**.

11. ✅ SDN - control/data plane interaction example

Question:



Answer:

The controller is made of three main parts:

- **Abstraction APIs**: Interfaces that allow **network control applications** to interact with the controller.
- **Distributed Database**: Stores **network state information**, such as **link status**, **switch state**, and **available services**.
- **Communication Components**: Responsible for **communicating with the switches**, usually through **southbound protocols** like OpenFlow.

The image shows an example of interaction between the control plane and the data plane in the case where a switch detects an error in one of its links (the link from s1 to s2). This interaction is composed of 6 total steps:

1. S1 detects link failure and uses OpenFlow port status message to notify controller.

2. SDN controller receives OpenFlow message, and updates link status info.

3. Dijkstra's routing algorithm gets called in this case

4. The algorithm accesses network graph info, link state info in controller, and computes new routes.

5. Link state routing app interacts with flow-table-computation component in SDN controller, which computes the new flow tables for switches.

6. The controller sends OpenFlow messages to install new tables in switches that need updating.

12. ✅ NFV – Forwarding graph

In typical enterprise networks, there are not only servers and switches, but also many **network functions**, like **firewalls, load balancers, intrusion detection systems**, and others.

These functions are usually run on **special hardware devices**, which are **expensive** and **difficult to manage**. Also, these components must follow a **strict order** (called **chaining**) that has to match the way the network is set up.

NFV (Network Function Virtualization) is about **moving these functions from hardware to software**, so they can **run on normal, cheaper hardware** (like standard servers or virtual machines).

NFV (Network Function Virtualization) - virtualization of network functions by implementing these functions in software and running them on virtualized infrastructure

A **network service** can be broken down into smaller parts called **Virtual Network Functions (VNFs)**. These VNFs can be **moved** and **started** in different places in the network **without needing to buy or install new hardware**.

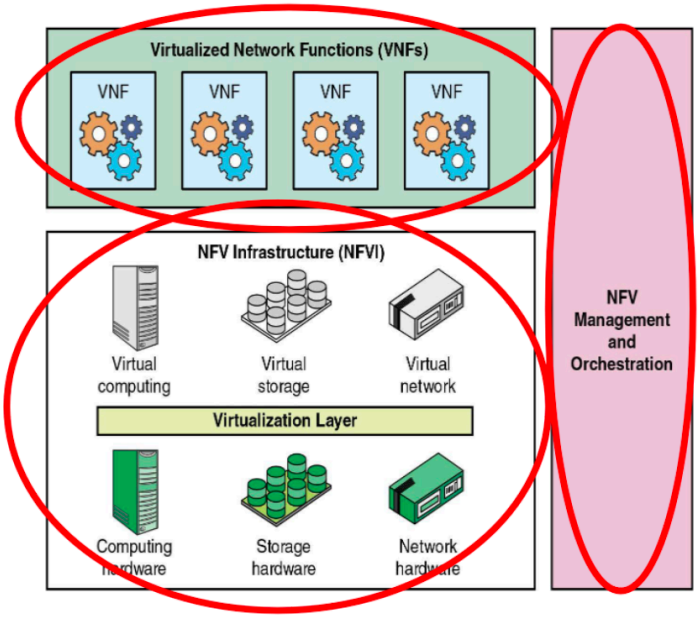
One of the most important goals of NFV is to **reduce costs**. There are two main types of costs:

- **CAPEX (Capital Expenditure):** the money spent on **buying hardware**. This cost is lower because we use **general-purpose (standard) hardware** instead of expensive, special devices.
- **OPEX (Operational Expenditure)**: the money spent on **managing and running the hardware**. This is reduced because we have **fewer devices to manage**.

Performance is also something to consider, because **software** might be **slower** than dedicated hardware. But standard hardware is getting **faster**, and the software can be **optimized** to work well on it.

Network Service: An **end-to-end network service** can be seen as a **forwarding graph**, which connects **network functions** and **endpoints**(like terminals or user devices). So, in simple terms, it is a system where different **Network Functions (NFs)** are connected to each other through the **underlying network infrastructure**.

The **nodes** that run **Virtual Network Functions (VNFs)** are called **Points of Presence (PoPs)**. These PoPs are connected using **logical links** (virtual connections), which are supported by the real, **physical links** of the network.



NFV Architecture:

- **NFV Infrastructure (NFVI)**

This includes the **hardware and software** needed to create the environment where **VNFs** run. NFVI virtualizes physical computing, storage, and networking and places them into resource pools

- **VNF (Virtual Network Function)**

A group of network functions that are **implemented in software** and run on **virtual resources** like computing, storage, and networking.

- **NFV Management and Orchestration (NFV-MANO)**

A **framework** used to **manage and control** all the resources inside the NFV environment.

The **NFV infrastructure** is made up of **standard hardware** and **virtual machines** managed by a **hypervisor**. The three main parts (or domains):

- **Compute Domain -** This includes **standard, high-volume servers** and **storage**, like the ones used in data centers.
- **Hypervisor Domain -** This part manages the compute resources and gives them to **virtual machines (VMs)** or **containers**, making the hardware easier to use and manage for software.
- **Infrastructure Network Domain -** This includes all the **regular, high-volume switches** connected together. They can be **configured** to provide the needed **network services**.

Virtual Switches connect **Virtual Interfaces (VIFs)** to **Physical Interfaces (PIFs)**, and also manage the traffic between VIFs on the same physical machine.

A **virtual switch** is a **software program** that works like a real switch (a **Layer 2 device** in networking).

VNF Placement Problem: Placing a **chain of VNFs** (Virtual Network Functions) is difficult. The goal is to **maximize a utility function**, while also following **infrastructure limits** and **quality of service (QoS) rules**.

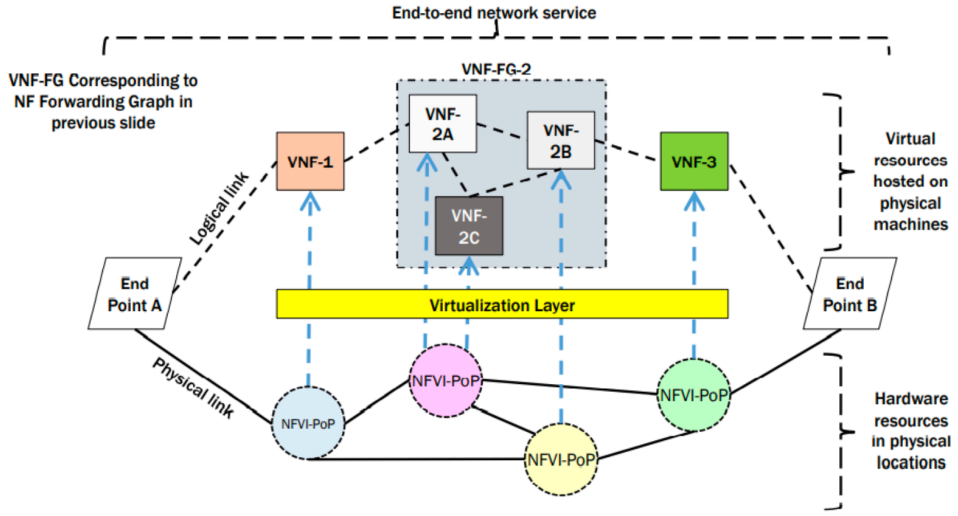
This utility function usually includes a mix of:

◇ Reducing total **network delay**;

◇ Reducing **deployment costs**;

◇ Increasing the amount of **available bandwidth**, and so on.

Question:



Answer:

Network Function Virtualization (NFV)

The increasing use of internet services that require **short and fast connections** has pushed operators to buy **specialized hardware** for each network function. This caused a **high-density deployment** of network infrastructure. To address this, operators are moving from **hardware-only** to **software-based solutions**.

What is NFV?

NFV uses **virtualization technologies** to **separate hardware from software**.

This allows **network functions** to be **moved or installed wherever needed**.

Key Concepts:

- A **Network Service** can be **split into multiple VNFs (Virtual Network Functions)** like:
  - Firewalls
  - Intrusion Detection Systems (IDS)
  - Load Balancers
  - etc.

- VNFs are connected in a **forwarding graph** (called **VNF-FG**), representing **how data flows** between them and to the end users.
- These logical links must be:
  - **Controlled by SDN controllers**, and
  - **Mapped to real physical paths** in the network infrastructure.
- **Deployment** can be:
  - By a single operator
  - Or across **multiple cooperating operators**

How VNFs are deployed:

- VNFs can run inside:
  - **Virtual Machines (VMs)** or
  - **Containers** → In that case they are called **CNFs (Containerized Network Functions)**
- These VMs or containers run on physical servers called **PoP (Point of Presence)**.

The VNF Placement Problem:

Deciding **where** to place VNFs is **not simple**. It must consider:

- **QoS**: latency, minimum bandwidth
- **Performance** and **resource limits** of each PoP
- **Cost**, **traffic efficiency**, and **resource usage**

There is **no single optimal solution**.

Possible solutions include:

- **Genetic algorithms**
- **Machine Learning**
- **Linear programming**, etc.

Example:

- A **cache service** (VNF-1) should be placed **near the user** (at the **network edge**) for better performance.
- Placing a cache **far from the endpoint** may cause **multiple hops** and increase delay.
- Some VNFs can be made of **other VNFs** (see VNF-FG-2 in the figure).

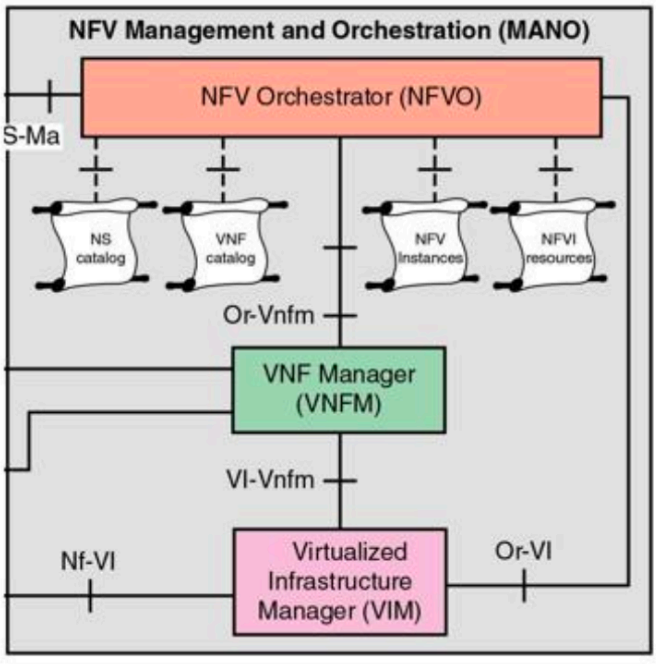
Dynamic Deployment:

Thanks to virtualization, we can:

- **Move** and **instantiate** VNFs **dynamically**
- **Allocate resources** as needed
- **Automate the process** using an **orchestrator**

13. ✅ NFV – Management and Orchestration

Question:



Answer:

NFV MANO Architecture (Management and Orchestration) is FRAMEWORK that:

- Takes care of setting up the **VNFs (Virtual Network Functions)** and all related operations, including:
  - **Configuring the VNFs**, and
  - **Configuring the infrastructure** (servers, storage, network) where the VNFs run
- Includes the **orchestration** and full **lifecycle management** of both physical and software resources that support virtualization, as well as managing the full life of VNFs
- Uses **databases** to store information, and **data models** that describe how functions, services, and resources should be deployed and managed
- Defines the **interfaces (APIs)** for communication between MANO (Management and Orchestration) components, and for integration with traditional network systems like OSS/BSS (Operations/Business Support Systems)

---

1. NFVO – NFV Orchestrator

The **NFVO** is responsible for:

- **Installing and setting up** new **Network Services (NS)** and VNF packages
- Managing the **full lifecycle** of services (setup, update, delete)
- Interacting with **traditional network hardware**

It also acts as the **service orchestrator**, meaning it **coordinates everything** needed to build and manage a complete **end-to-end network service** using VNFs.

The NFVO works using several internal **repositories**:

Repository	Description
NS Catalog	List of available Network Services and deployment templates
VNF Catalog	List of VNFs and their resource descriptors (VNFDs)
NFVI Resources	List of available physical and virtual resources

NFV Instances	Records of currently deployed NS and VNF instances

It receives service requests via a **Service Manifest** containing the configuration and network needs.

2. VNFM – VNF Manager

The **VNFM** is responsible for:

- Managing lifecycle of each individual **VNF instance**
- Handling: **Creation (instantiation), scaling (up/down), updates**, **Status checks (queries),** and **termination**
- Collecting **performance metrics**
- Detecting and reporting **errors/faults**

3. VIM – Virtualized Infrastructure Manager

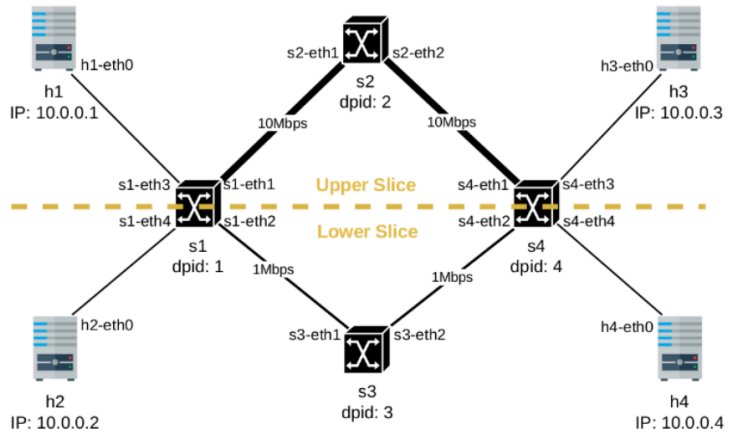
The **VIM** controls (manages) the **physical and virtual resources** used by VNFs:

- Computes
- Storage
- Network resources

The VIM ensures that the **infrastructure** can meet the **resource needs** of a service. A single VIM can manage an entire **infrastructure domain**. However, to cover a large or distributed infrastructure, **multiple VIMs** might be used in one NFV MANO.

14. Network slicing

Question:




```
def __init__(self, *args, **kwargs):
    super(TrafficSlicing, self).__init__(*args, **kwargs)

    # out_port = slice_to_port[dpid][in_port]
    self.slice_to_port = {
        1: {1: 3, 3: 1, 2: 4, 4: 2},
        4: {1: 3, 3: 1, 2: 4, 4: 2},
        2: {1: 2, 2: 1},
        3: {1: 2, 2: 1},
    }

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    in_port = msg.match["in_port"]
    dpid = datapath.id
    out_port = self.slice_to_port[dpid][in_port]

    actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
    match = datapath.ofproto_parser.OFPMatch(in_port=in_port)
    # add_flow(self, datapath, priority, match, actions)
    self.add_flow(datapath, 1, match, actions)
    self._send_package(msg, datapath, in_port, actions)
```

21

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # install the table-miss flow entry.
    match = parser.OFPMatch()
    actions = [
        parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
        ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # construct flow_mod message and send it.
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
        actions)]
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
          match=match, instructions=inst)
    datapath.send_msg(mod)
```

Answer:

Example of Network Slicing

Network slicing is a technique used in **SDN (Software Defined Networking)** to **separate and isolate network resources** in order to meet different needs on the **same physical infrastructure**.

This image shows how network slicing is implemented in an SDN to allow the **separation of network resources**. The goal of this example is to show that different needs can be met **on shared infrastructure** using slicing.

The architecture uses a **multi-hop topology**: The network includes **four hosts** (h1, h2, h3, h4) and **four switches** (s1, s2, s3, s4).

The network must be **split into two slices** for two services:

- ***Slice for video traffic (upper slice):***

  Video traffic (UDP port 9999) can use up to **10 Mbps** of bandwidth. This traffic has the **right to use this bandwidth independently** of other traffic types.

- ***Slice for other traffic (lower slice)*:**

  All other traffic types (not UDP port 9999) can use **any path in the network**, but they **must not interfere** with the video traffic.

Code Explanation – SDN Controller Behavior

📌 First snippet:

There is an **entry in a dictionary** for each switch.

Each entry maps the switch's **input port** to its **output port** – this defines how traffic must be forwarded. These mappings are used to **create flow rules**, which will later be **installed into the switches**.

📌 Topology:

The network structure is implemented in the file `network.py`. It defines the **hosts, switches**, and **physical links** between them.

📌 Event handling in Ryu (the SDN controller app):

Ryu applications can **listen for specific events**, and run functions when those events happen.

1. Packet in event (`EventOfPacketIn`)

When a switch **doesn't know what to do with a packet**, it sends the packet to the controller.

The controller receives:

- The **switch ID**
- The **input port** of the packet

Then the controller:

- Decides **where the packet should go** (based on the network topology)
- Calls the function `add_flow` to install a rule in the switch

After that, the switch will handle similar packets by itself (without asking the controller again).

The function `_send_package` sends the current packet back to the switch (via a packet out message), telling it what to do.

---

2. Switch features reply event (`EventOfSwitchFeatures`)

This happens when the controller receives a **feature reply message** from the switch, which was sent in response to a **periodic request** by the controller. The code installs a **default rule** into the switch:

- If a packet **doesn't match any rule** in the switch's table,
- The switch will **send it to the controller**.

So:

- `match = no matching entry (table-miss)`
- `action = send to controller`

📌 `add_flow` Function:

This function builds a **flow_mod message** using the parameters (match, action, etc.) and sends it to the right **switch** using a `datapath` object, which contains the **ID and address** of the target switch.

15. ✅ Fourier Series

Signals are variations of physical quantities that carry information.

Examples include: Radio, Biological, Acoustic, Electrical Signals.

A **signal** has these main characteristics:

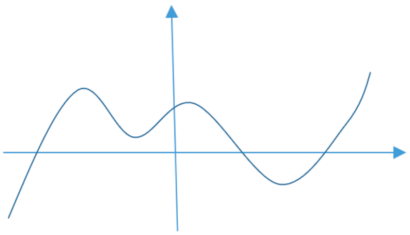
- It is a **function of one or more independent variables** (like time or space)
- It shows how a **measurable physical quantity changes over time or space**
- It **carries information**

A **signal** is a real function of time

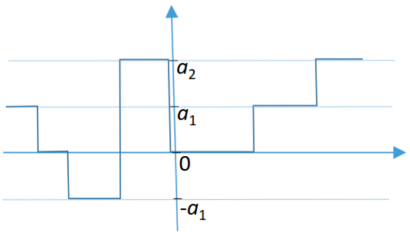
- Time-continuous and amplitude-continuous
- Time-continuous and quantized
- Time-discrete and amplitude-continuous
- Time-discrete and quantized

Signal types:

Continuous-time signal. A signal is called **continuous-time** (also known as **analog signal**) if it is defined for **all real numbers**.



Continuous time and amplitude



Continuous time and discrete amplitude

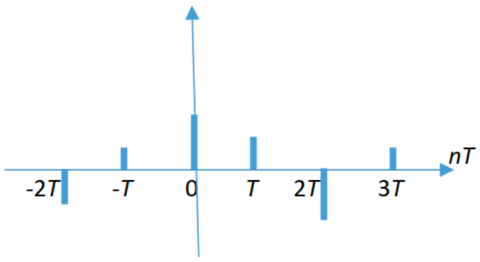
This means the **domain** (D-X axis) is the set of all real numbers (**R**).

- If the **range** (codomain C - Y axis) is also the set of real numbers (**R**), then it's a **continuous amplitude signal**. **Time-continuous and amplitude-continuous**
- If the **range** (codomain C - Y axis) is a **discrete set** (like the integers **Z, means fixed set of numbers (a1, a2, 0, -a1**), then it is a **discrete amplitude signal**, also called a **quantized signal**. **Time-continuous and quantized (amplitude discrete)**

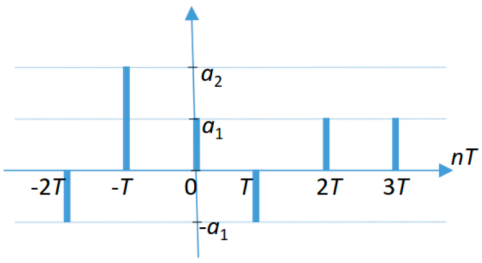
Discrete-time signal

A signal is called **discrete-time** if it is defined **only at specific (separated) time points**. The (**domain** D-X axis) is the set of time values like **ZT**, **means fixed set of time values (-2T, -T, 0, T, 2T)**. These are also called **discrete signals**.

For example, if Sampling Period T = 2, then Z_2 = ..., −4, −2, 0, 2, 4, ...



Discrete time and continuous amplitude



Discrete time and amplitude

11

- If the **range** (codomain C - Y axis) is the set of real numbers (**R**), then it's **Time-discrete and amplitude-continuous**
- If the **range** (codomain C - Y axis) is a **discrete set** (like the integers **Z, means fixed set of numbers (a1, a2, 0, -a1**), then it is a **Time-discrete and quantized (amplitude discrete)**

A **discrete signal** is called **digital** when the **range** (codomain C-Y axis) comes from a **finite set of symbols**. A **digital signal** is also called a **symbolic sequence**. An example of a symbolic (digital) signal is a **text**.

Transmission: Signals can be of different types depending on the **channel** they travel through — for example, **electromagnetic waves**, **sound**, or **light**.

At the **beginning** (source), a **transducer** changes a **message** into a **signal** that can travel through a medium. At the **end** (destination), another **transducer** changes the **signal** back into a **message**. For example, an **antenna** is a **transducer** that works with **electromagnetic signals**.

Transducer: any device that converts one form of energy into another.

Distortion means any change in a signal that affects its original shape or how its different frequency parts relate to each other. Examples include:

- **Attenuation:** the signal slowly loses strength as it travels through something (like a wire or the air).
    - If a signal has different frequency parts and goes through one channel, some parts may get weaker more than others depending on the frequency.
- **Multipath propagation:** the radio signal bounces off objects or the ground and reaches the receiver at slightly different times, causing changes in the signal.

Noise is an unwanted, random signal that we didn't plan for. It interferes with communication or signal measurement.

- **External noise:** comes from outside sources like nearby channels or broken equipment.
- **Internal noise:** comes from inside the system, like the thermal movement of electrons in wires.

Signal-to-noise ratio (SNR) is a measure of how much the message is affected during transmission. It shows how strong the received signal is compared to the background noise. *SNR in dB = 20 * log(signal/noise)*

Sampling is the process of turning a continuous signal into a discrete signal. The **sampling rate** is how many samples are taken each second. The ***sampling theorem*** says that if you take samples at a rate higher than **twice** the highest frequency in the signal, you can perfectly rebuild the original signal.

A **digital source** can come from **sampling and quantizing** an analog signal (like speech).

- At the **transmitter (Source)**: the analog signal is sampled at specific time intervals (these are called "discrete intervals").

  An ***analog-to-digital converter*** **(ADC)** changes the analog signal into a sequence of digital symbols.
- At the **receiver (Destination)**: the digital symbols are changed back into an analog signal using a **digital-to-analog converter (DAC)**.

However, **sampling and quantization** add a small distortion to the signal, called **quantization noise**, because of:

- The **sampling frequency** (how often you sample the signal)
- The **resolution** of the digital symbols (how many different values can be used to represent the analog signal)

Shannon's Theory: It explains the **limit** on how much information can be sent **reliably** over a channel that has **noise**.

- Let's define:
  - **B** = maximum **bandwidth** of the channel (in Hz)
  - **S** = maximum **power** of the signal sent
  - **N** = average **noise power** at the receiver
  - **C** = **channel capacity**, the highest possible data rate (in bits per second)

The formula is:

$$C = B \log_2\left(1 + \frac{S}{N}\right)$$

This means: The more bandwidth and signal power you have (compared to noise), the more information you can send through the channel.

Fourier series is a way to represent a **periodic signal** as a (possibly infinite) sum of **sine** and **cosine** functions. Each *sin(e)* and *cosin(e)* has its own **frequency**, **amplitude**, and **phase**.

A **continuous signal** $s(t):R \to R$ is **periodic** with period T if: $s(t)=s(t+T)$ for all $t \in R$

Example: Functions like sin(nt) and cos(nt) are periodic, and their period is $T=2\pi/n$ for any integer n.

If a signal is **not periodic**, we call it **aperiodic**. Since periodic signals repeat, we only need to study them in one period, for example in the interval [0,T], because their behavior is the same everywhere.

Also, for periodic signals it holds that:

$s(t+T)=s(t+2T)=s(t+nT)$ for all $t \in R$, $n \in Z$

Periodic Extension

If we have an **aperiodic** signal s(t), which exists only in the time interval [a,b) (that is, s(t)=0 outside this interval),

we can make it **periodic** by repeating it every T=b−a seconds.

This is called the **periodic extension** $s*(t)$

$$s^*(t) = \sum_{n=-\infty}^{\infty} s(t - nT)$$

In simple words: You copy the original signal and shift it left and right by all multiples of T, and then add them all together. This gives you a **repeating version** of the original signal.

Combining and decomposing signals:

It is possible to combine signals, e.g. by adding them together, to create new signals.

The Fourier series lets us **break down a periodic signal** into a **sum of sine and cosine functions**. This is like reversing the process of combining many waves into one.

More formally, the Fourier series breaks a function (or signal) into the **sum of an infinite number of continuous functions**, each one **oscillating at a different frequency**. This group of continuous functions forms the **basis** for the decomposition. The Fourier series uses a set of functions: $\varphi n(t), n \in Z$

This set of functions must be **orthogonal** — just like when we break a vector into parts in a vector space (for example, in x, y, z directions).

Definition 21.8 (Fourier series) *Given a continuous signal $s(t) : \mathbb{R} \to \mathbb{R}$, periodic in the interval $[-\pi, \pi]$ its Fourier series is defined as:*

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} (a_n \cos nt + b_n \sin nt)$$

Constant component

Amplitude of the harmonics

Harmonics

where

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)\,dt;\ a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)\cos nt\,dt;\ b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)\sin nt\,dt$$

It is rather complicate to assess the conditions under which an arbitrary $s(t)$ can be developed in a Fourier series, in particular necessary conditions are *not* known; however, there are sufficient conditions:

It's quite **difficult to define exactly when any function s(t)** can be written as a **Fourier series**. In fact, we **don't know all the necessary conditions**. However, we **do know some sufficient conditions** — if these are met, the Fourier series works.

Dirichlet conditions:

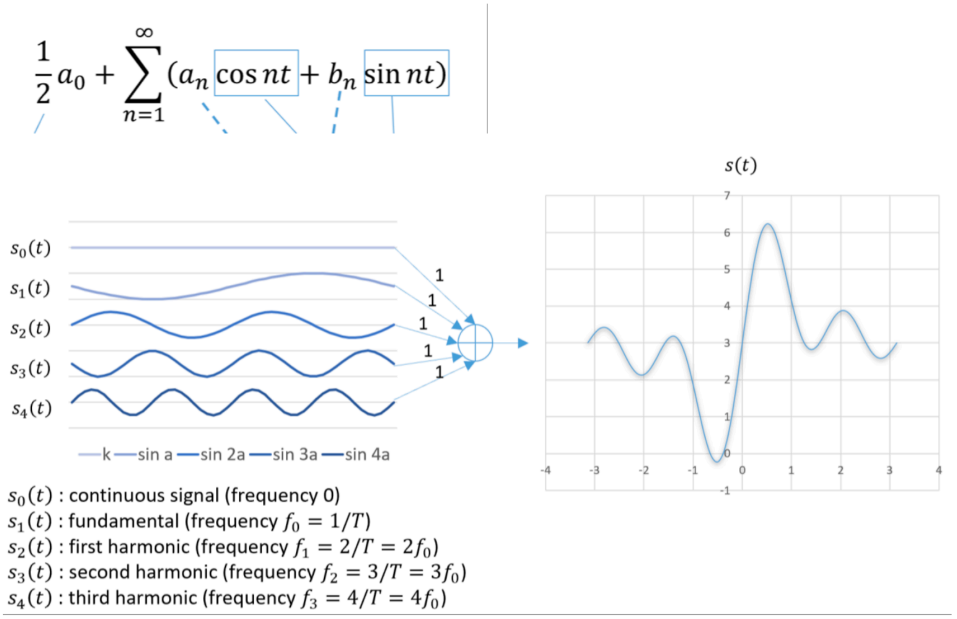
1. s(t) is **periodic**
2. s(t) is **piecewise continuous**

→ then the **Fourier series of s(t) exists and converges** for all real numbers.

A **piecewise continuous function** is made of a **finite number of continuous parts** over each small section (like in the interval [0, T]).

Also, at any point where the function jumps or has a break (discontinuity), the **limit still exists and is finite**.

Question: FOURIER SERIES



Answer:

The image shows the definition of the **Fourier series**, which is used to **break down a continuous and periodic signal**(defined in the interval [−π, π]) into its **harmonics** (which are usually sine and cosine functions that oscillate at different frequencies).

We can see that the signal s(t) on the right is made by **adding together four basic harmonics** and a **constant component** $s_0(t)$.

Above, we see the equation that defines the Fourier series for s(t), which is a function from $\mathbb{R}$ to $\mathbb{R}$ in the interval [−π, π].

The equation is used to **decompose the signal into an infinite sum of continuous functions**, which are **periodic and oscillate at different frequencies**. Here:

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)dt$$
$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)cos(nt)dt$$
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)sin(nt)dt$$

- a_0 is the constant component,

- **sin(nt)** and **cos(nt)** are the harmonics,
- $\mathbf{a_n}$ and $\mathbf{b_n}$ are the amplitudes of each harmonic.

We don't know all the necessary conditions for when s(t) can be written as a Fourier series, but there are two **sufficient conditions** (called **Dirichlet conditions**):

- s(t) must be **periodic**
- s(t) must be **piecewise continuous**

→ If these conditions are met, then the **Fourier series of s(t) exists and converges** for all real values.

The **Fourier series** can also be written using **exponential functions**, thanks to **Euler's exponential formula** (with period T = 1/F).

If a signal is **not periodic**, then we **can't use the Fourier series**, because the period would go to infinity (T → ∞), and the fundamental frequency would go to zero (F = 1/T → 0).

This means that the harmonics (multiples of F) would become **infinitely close to each other**.

In this case, we use the **Continuous Fourier Transform**, which gives us the **spectrum S(f)** of the signal — showing **how the signal is spread over different frequencies**.

16. ✅ DFT

Question:

$$S_f = \sum_{n=0}^{N-1} s_n e^{-j\frac{2\pi f}{N}n} \; with \; f = 0, 1, \ldots, N-1$$

Answer:

Fourier Transform, which gives us the **spectrum S(f)** of the signal — showing **how the signal is spread over different frequencies**. The **absolute value** of |Sf| shows the **amplitude** of the harmonic with frequency **f = 1/N**, where **f** is the **fundamental frequency**.

Each **harmonic** is made by adding **sine and cosine** functions, each multiplied by a **coefficient** that shows its amplitude (informally, the "weight" or strength of the harmonic).

The function in the figure shows the definition of a Discrete Fourier Transform (DFT), which is used for a **discrete, not continuous** signal. This signal comes from **sampling** a continuous signal, giving a series of observations taken at regular time intervals **T**. Since these observations appear as **points** on the graph, we assume each sample has an **"impulse"** (i.e., a **rectangle** on the plane).

So we break it into a **summation** up to **N**, which is the total number of observations (samples).

Since the samples are taken at time intervals **T = 1/N**, they are **periodic**.

So, instead of using frequencies in **R** (the set of all real numbers), we take **frequencies** from the set of **harmonics** of the fundamental frequency **1/N**. That is:

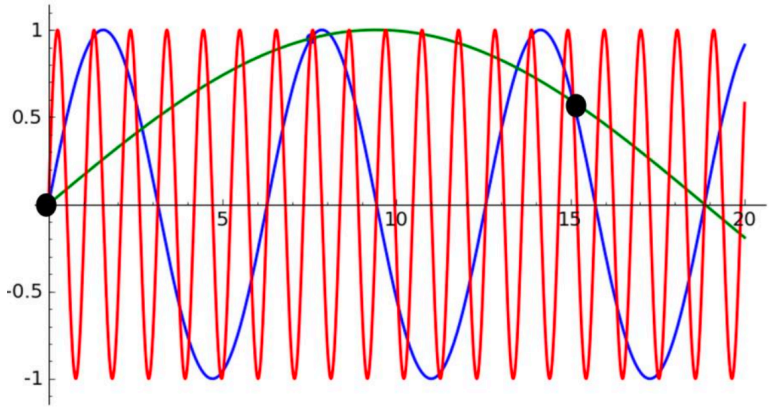
f = 0, 1/N, 2/N, ..., (N−1)/N

To make the formula easier to write, we use **f = 0, 1, ..., N−1** and divide the exponent by **N**.

So, based on all these assumptions, the formula in the figure shows how it is possible to calculate the **Discrete Fourier Transform (DFT)** from a **finite number of samples** of a signal.

17. ✅ Sampling and Aliasing

Question:



Answer:

Discrete Fourier Transform: Main Errors

There are two main types of **errors** that make the **Discrete Fourier Transform (DFT)** different from the **true spectrum** of the original signal we sampled:

📌 Aliasing:

If we **don't use a high enough sampling rate** for a signal that changes very fast, we might **not be able to reconstruct** it properly or **analyze its real frequencies**.

For example, if we only look at the sampled values and apply the Fourier Transform, we might think the signal has just **one frequency**, even though it could actually be the **sum of many harmonics** (or maybe a **multiple of a base frequency**).

So, signals like **s(t)** and **3s(t)** might **look the same** after sampling, even if they are very different in reality.

In the image (mentioned), three sine waves are shown, and they differ by **1/6 of the frequency** (f′ = f/6). They all match the **same black sample points**, but they are **three different signals**!

📌 Leakage:

When sampling a **periodic signal**, if the **first and last sample** in the sampling window **have different values**, it creates a **discontinuity** (a sudden jump in values).

This usually happens when the sampling **does not stop exactly at the end of a period**. This discontinuity can cause **errors** in the spectrum and false results during reconstruction.

To reduce this problem, we can try to **make the first and last samples closer to zero**.

Sampling a Continuous Signal

A discrete signal **g(nT)** can be obtained by **uniformly sampling** a continuous signal **s(t)**:

> g(nT) = s(nT)

Where:

- **T** = sampling period
- **F = 1/T** = sampling frequency

If we know that **fM** is the maximum frequency present in **s(t)** (i.e., above it the signal spectrum is zero), then the **Sampling Theorem** tells us that **s(t)** can be reconstructed from its samples if:

1. The samples are taken at **regular intervals**:

   > tn = nTc (with n ∈ ℤ)

2. The sampling period **Tc** is **less than or equal to** 1/(2fM), or in other words:

   > Minimum sampling frequency must be:
   >
   > fc_min = 1/Tc_max = **2fM** → also called **Nyquist Frequency**

So, the **Nyquist frequency** is **twice the highest frequency** in the signal **s(t)**.

If these conditions are satisfied, the original signal can be **reconstructed using interpolation**.

⚠️ Why Nyquist Sets a Frequency Limit

Nyquist asks for a **maximum frequency limit** in the signal.

Why? Because this helps **avoid confusion with higher frequencies** that could match the same sample points (causing aliasing).

But in the **real world**, it's very hard to have signals that are **perfectly limited in frequency**. Even if the signal is limited in frequency, we still **can't perfectly reconstruct it** unless we use **an infinite number of samples**.

So the theorem gives us the **minimum frequency** we must sample at, but **does not say how many samples** are needed to reconstruct the signal.

This means that using **a limited number of samples** will always give us **an approximation**, not perfection.

Why Sampling Frequency Matters

If we choose the **wrong maximum frequency** (or too low a sampling frequency), then **reconstruction may not work well**.

This is because the distance between **copies (replicas)** of the signal in frequency domain is equal to the **sampling frequency**.

The **higher the sampling frequency**, the more **separated the replicas**, and the easier it is to separate the true spectrum from the overlapping ones.

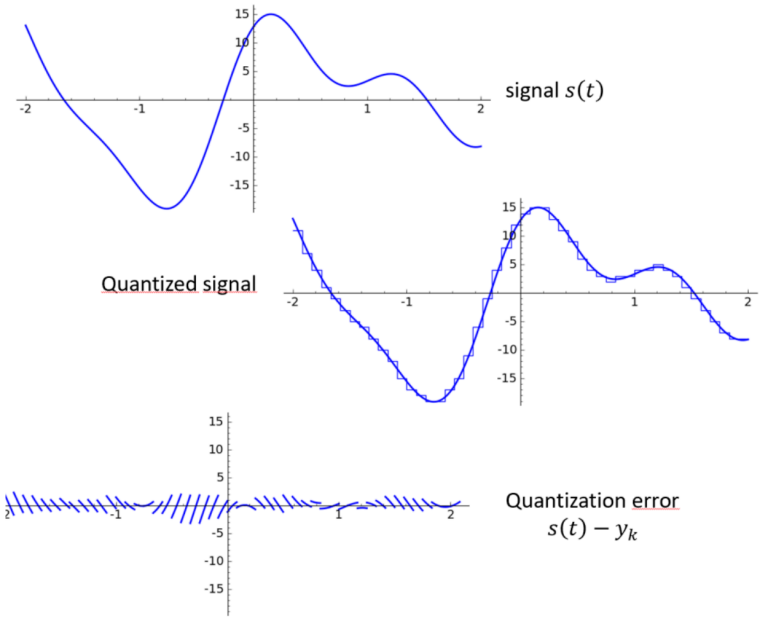
Practical Use of the Theorem

Even though **perfect reconstruction** is not always possible, the theorem gives a **good base (sampling period)** for reconstruction that is **accurate enough for most applications**, like **speech** (which is nearly band-limited).

Also, we can use **filters** to remove high frequencies (to better satisfy the sampling conditions). But **no filter is perfect**, so there will always be **some error**.

18. ✅ Quantization

Question:



Answer:

Quantizing with a **"scalar quantizer with R bits"** means approximating the value of a sample x_k with an integer y_k, which must be in the range **[0, $2^R - 1$]** (for example, Arduino uses 10 bits).

This is necessary because storing samples as floating-point numbers (4 or 8 bytes) is often **not possible on devices with limited hardware**, like in IoT systems. The more bits available, the **better the resolution** of the quantization. If we use **uniform quantization**, the resolution for a signal with values in the range **[0, M]** is **$M / 2^R$**.

Of course, this approximation causes **an error** in how the sample is represented — and in how the signal can be reconstructed later. In extreme cases, this can lead to:

- **Overload**, when the signal **s(t)** is greater than **$2^R - 1$** (i.e., outside the range that can be represented).
- Or **granular noise not being detected**, meaning the signal changes so little within an interval I_k that it is always represented by the same y_k.