**Unit 3**

**Network Layer**

**Network Layer Overview**

The **Network Layer** is responsible for **host-to-host communication** across different networks. It ensures data packets (datagrams) are routed from the source to the destination, even if they traverse multiple intermediate networks (e.g., the internet). This layer addresses **logical addressing**, **routing**, and **packet forwarding**, acting as the "postal service" of the network.

**Key Design Issues of the Network Layer**

**a) Addressing**

* Every device on the network needs a **unique logical address** (e.g., IPv4/IPv6 addresses).
* Addresses must be structured hierarchically (like postal codes) to simplify routing (e.g., subnetting in IP networks).

**b) Routing**

* **Routing algorithms** determine the optimal path for packets to reach the destination.
* **Static routing** uses preconfigured paths (simple but inflexible).
* **Dynamic routing** adapts to network changes (e.g., OSPF, BGP).

**c) Packet Forwarding**

* Routers use **forwarding tables** to decide where to send incoming packets based on their destination addresses.
* Forwarding can be **connection-oriented** (virtual circuits) or **connectionless** (datagram-based).

**d) Fragmentation & Reassembly**

* Networks have different **Maximum Transmission Unit (MTU)** sizes.
* The network layer **fragments** large packets into smaller chunks for transmission and **reassembles** them at the destination.

**e) Congestion Control**

* Prevents network overload by managing traffic flow (e.g., dropping packets during congestion, as in **RED – Random Early Detection**).

**f) Heterogeneity Handling**

* Interconnects diverse network types (Ethernet, Wi-Fi, satellite links) by standardizing protocols like **IP (Internet Protocol)**.

**Services Provided to the Transport Layer**

The Network Layer offers two primary types of services to the Transport Layer:

**a) Connection-Oriented Service (Virtual Circuit)**

* **How it works**: A dedicated path (virtual circuit) is established before data transfer, similar to a phone call.
  + **Setup**: Initial handshake to reserve resources (e.g., bandwidth).
  + **Data Transfer**: All packets follow the same path, ensuring in-order delivery.
  + **Teardown**: Connection is terminated after transfer.
* **Features**:
  + Guaranteed reliability and sequencing (e.g., used in **ATM networks**).
  + Higher overhead due to setup/teardown phases.

**b) Connectionless Service (Datagram)**

* **How it works**: Each packet is routed independently, like mailing letters. No prior setup is needed.
  + Packets may take different paths and arrive out of order.
  + The Transport Layer (e.g., **TCP**) handles reordering and retransmission.
* **Features**:
  + Low overhead, flexible, and scalable (e.g., **IP** in the internet).
  + No guarantees of delivery or order (best-effort service).

**Routing Algorithms**

**Adaptive vs. Non-Adaptive Routing Algorithms**

**a) Adaptive (Dynamic) Algorithms**

* **Definition**: Routing decisions change dynamically based on current network conditions (e.g., traffic load, link failures).
* **How it Works**:
  + Routers exchange real-time information (e.g., latency, congestion) with neighbors using protocols like **OSPF** or **RIP**.
  + Routes are recalculated periodically or triggered by topology changes.
* **Advantages**:
  + Responds to congestion or failures (e.g., rerouting traffic around a broken link).
  + Optimizes performance under varying loads.
* **Disadvantages**:
  + Higher overhead due to constant updates.
  + Complexity in implementation.
* **Examples**:
  + **Distance-Vector (RIP)**: Routers share entire routing tables with neighbors.
  + **Link-State (OSPF)**: Routers broadcast the state of their links to all nodes, enabling global topology maps.

**b) Non-Adaptive (Static) Algorithms**

* **Definition**: Routes are fixed and determined in advance, regardless of network conditions.
* **How it Works**:
  + Routes are manually configured (e.g., **static routing tables**).
  + No real-time updates or communication between routers.
* **Advantages**:
  + Simple to implement with minimal overhead.
  + Predictable performance.
* **Disadvantages**:
  + Cannot adapt to failures or congestion.
  + Inflexible for large or dynamic networks.
* **Examples**:
  + **Flooding**: Sends packets to all paths (used in emergency networks).
  + **Shortest Path (predefined)**: Uses fixed tables without updates.

**Optimality Principle**

* **Definition**: A foundational rule stating that *if router* ***B*** *lies on the optimal path from router* ***A*** *to router* ***C****, then the optimal path from* ***B*** *to* ***C*** *must follow the same route*.
* **Implications**:
  + Enables **shortest-path algorithms** (e.g., Dijkstra’s) to build routing tables by aggregating optimal subpaths.
  + Ensures consistency in routing decisions across the network.
* **Example**: If the shortest path from *New York* to *Los Angeles* passes through *Chicago*, the shortest path from *Chicago* to *Los Angeles* must be the same segment.

**Dijkstra’s Shortest Path Algorithm**

* **Purpose**: Finds the shortest path from a single source node to all other nodes in a weighted graph (e.g., network topology).
* **Steps**:
  1. Assign a tentative distance to all nodes (0 for the source, ∞ for others).
  2. Select the node with the smallest tentative distance (initially the source).
  3. Update distances to neighboring nodes using the edge weights.
  4. Mark the node as visited and repeat until all nodes are visited.
* **Key Features**:
  1. Uses a **greedy approach** (always picks the nearest unvisited node).
  2. Requires prior knowledge of the entire network topology.
  3. Time complexity: **O(n²)**, where *n* = number of nodes.
* **Example**: Used in **OSPF** to compute shortest paths in autonomous systems.

**Flow-Based Routing**

* **Definition**: Routes traffic based on **current network flow** (i.g., traffic load and link capacity) rather than static metrics like hop count.
* **How it Works**:
  + Uses mathematical models (e.g., **flow optimization**) to balance traffic across paths.
  + Requires knowledge of:
    - Traffic demand between source-destination pairs.
    - Link capacities.
  + Goal: Minimize congestion or maximize throughput.
* **Advantages**:
  + Optimizes resource utilization.
  + Reduces congestion in heavily loaded networks.
* **Disadvantages**:
  + Computationally intensive (requires solving linear equations).
  + Impractical for large, dynamic networks (used in offline planning).
* **Example**: Designing backbone networks for telecom operators.

**Hierarchical Routing**

* **Purpose**: Reduces routing table size and overhead in large networks by organizing routers into **hierarchical groups** (regions, clusters, or autonomous systems).
* **How it Works**:
  + **Levels of Hierarchy**:
    1. **Routers within a region** know detailed routes inside their region.
    2. **Boundary routers** summarize routes for other regions (e.g., advertising a single prefix for an entire region).
  + **Example**: The internet uses **Autonomous Systems (ASes)**. Each AS (e.g., Google’s network) uses internal routing protocols (OSPF) and external protocols (BGP) to communicate with other ASes.
* **Advantages**:
  + Scalability: Reduces routing table entries (e.g., a router in Europe doesn’t need routes for Asian networks).
  + Lowers update traffic.
* **Disadvantages**:
  + Suboptimal paths may occur due to summarization.
  + Increased complexity in managing hierarchies.

**Congestion Control Overview**

Congestion occurs when network traffic exceeds its capacity, leading to packet loss, delays, and degraded performance. **Congestion control algorithms** manage data flow to prevent or mitigate congestion. Two classic traffic-shaping algorithms are the **Leaky Bucket** and **Token Bucket**, which regulate the rate of data transmission.

**1. Leaky Bucket Algorithm**

**Concept**

Imagine a bucket with a small hole at the bottom. No matter how fast water (data) is poured into the bucket, it leaks out at a **constant rate**. The algorithm enforces a fixed output rate, smoothing out bursty traffic.

**How It Works**

1. **Bucket Capacity**: The bucket holds a finite number of packets.
2. **Input**: Incoming packets are added to the bucket.
3. **Output**: Packets leak out at a **fixed rate** (e.g., 1 Mbps), regardless of input rate.
4. **Overflow**: If the bucket is full, excess packets are **discarded** (or marked for retransmission).

**Advantages**

* **Traffic Smoothing**: Converts bursty traffic into a steady stream.
* **Prevents Congestion**: Limits the outgoing rate to match the network’s capacity.
* **Simple Implementation**: Easy to deploy in routers.

**Disadvantages**

* **No Burst Allowance**: Even if the network has capacity, bursts are flattened.
* **Packet Loss**: Drops excess packets during high traffic.

**Example**

Used in **ATM networks** to enforce constant cell rates.

**2. Token Bucket Algorithm**

**Concept**

Imagine a bucket that fills with **tokens** at a fixed rate. To send a packet, a token must be removed from the bucket. Tokens accumulate up to the bucket’s capacity, allowing controlled bursts.

**How It Works**

1. **Token Generation**: Tokens are added to the bucket at a **fixed rate** (e.g., 1 token/sec).
2. **Bucket Capacity**: Maximum tokens the bucket can hold (e.g., 100 tokens).
3. **Packet Transmission**: Each packet consumes one token. If tokens are available, packets are sent immediately.
4. **Burst Handling**: If tokens are available, multiple packets can be sent in a burst (up to the bucket size).
5. **No Tokens**: Packets are either **buffered** (if space) or **discarded** if the bucket is empty.

**Advantages**

* **Burst Tolerance**: Allows short-term bursts up to the bucket size.
* **Flexibility**: Balances steady rates with occasional bursts.
* **Efficient**: Uses network capacity more dynamically.

**Disadvantages**

* **Complexity**: Requires tracking tokens and managing buffers.
* **Overhead**: Token generation and management consume resources.

**Example**

Used in **Quality of Service (QoS)** for video streaming to allow bursts during high-action scenes.