ELECTRONIC & TELECOMMUNICATION ENGINEERING
UNIVERSITY OF MORATUWA

EN2150 - Communication Network Engineering

SDN Algorithm Development

Name	Index Number
S.M.S.M.B.Abeyrathna	210005H
M.M.H.H.B. Gallalla	210174X
A.A.W.L.R.Amarasinghe	210031H
R.N.Abeywardane	210015M

Team - Tronic Pros

Date - 01.09.2024

1 Algorithm for Traffic Engineering

The primary goal of Traffic Engineering is to optimise the allocation of network resources by intelligently distributing traffic across all available network links, rather than simply directing it through the shortest path. This approach aims to leverage multiple paths to enhance resource utilisation while preventing congestion on any single link. The significance of this method lies in its ability to avoid the overloading of specific links, which can lead to traffic congestion, and simultaneously prevent the underuse of other links, thus avoiding resource wastage. To address this issue, we introduce an algorithm that effectively utilises the existing network infrastructure, enabling the routing of packets along diverse paths instead of confining them to the shortest route. The algorithm consists of several stages, each justified by its contribution to achieving the desired network efficiency.

1. Network Initialization
2. Optimal Path Determination.
3. Traffic and Network Monitoring
4. Traffic Distribution Management
5. Routing Rules Implementation
6. Adaptive Routing Adjustments
7. Network Security and Traffic Segregation
8. Regular Performance Review

1.1 Network Initialization

Network initialization is a crucial first step in Traffic Engineering. During this phase, the network controller collects comprehensive information about the network, including:

- Network Architecture: Detailed layout and topology of the network.
- Network Capabilities: The operational capacity and features of the network components.
- Initial Conditions: Starting status and parameters of the network elements.

Additionally, this phase may involve assessing:

- Traffic Patterns: Expected traffic loads and usage trends.
- Performance Metrics: Baseline performance indicators for comparison.
- Resource Availability: Initial allocation of resources such as bandwidth and processing power.
- Configuration Settings: Default settings and parameters for network devices and protocols.

1.1.1 Building the Traffic Demand Matrix:

A traffic demand matrix is used to gain an initial understanding of traffic flows between different nodes in the network. It indicates the amount of traffic expected or currently flowing between each pair of nodes, which can be estimated from historical data and real time

monitoring. This matrix helps identify typical traffic patterns, allowing the controller to predict potential congestion points and allocate resources more effectively.

The traffic matrix can be adjusted based on observed traffic patterns. Networks with stable traffic patterns may not require frequent updates. However, networks with highly dynamic traffic patterns will need continual updates to the traffic matrix.

1.1.2 Processing the Collected Data

Once the network information is collected, it is essential to convert all the data into a uniform format that is suitable for algorithmic processing. Next, we assign weights to the links based on the gathered data, such as latency and bandwidth. For example, links with lower latency and higher bandwidth might receive lower weights. This approach allows the routing algorithm to effectively utilise the processed information.

1.1.3 Network Topology Discovery

The controller needs to identify all nodes and links within the network. This can be achieved using common protocols such as the Link Layer Discovery Protocol (LLDP) for discovering the network topology and SNMP for monitoring link status. These protocols help in mapping out the arrangement of nodes and links. For each node and link, the controller gathers and stores information about their capabilities, including link bandwidth, latency, processing power, memory and the number of interfaces on each node. In summary, for a small and straightforward network, the controller will know each link's bandwidth and latency, its current status (active or down), as well as the connections between the nodes.

1.1.4 Understanding Link Capacities and Constraints

To understand how much traffic each link can handle before congestion occurs, it's important to determine the link's capacity, the maximum amount of data that can be transmitted per unit of time along with any constraints such as policy restrictions, physical limitations, and security requirements. The controller will then set a utilisation threshold to avoid congestion. Historical usage patterns of the links can help identify peak usage periods. Real time network data will be used to adjust the controller’s decisions according to the current state of the network.

1.2 Traffic and Network Monitoring

This marks the beginning of the iterative process for the algorithm. The controller continuously monitors the links and selects the most suitable path based on the gathered information. For each link, the controller retrieves the following data:

- Current Link Utilisation
- Available Bandwidth
- Packet Loss Rates

In addition, the cost (C) calculated using traditional methods from the previous step is also available. Using this information, the overall cost (F) for each path is determined as follows:

$$F=w_1e^{B/L}+w_2e^{P}+w_3C$$

- w_1 ,w_2 and w_3 are weights assigned to each parameter.
- L is the current link utilisation.
- B is the available bandwidth.
- P is the packet loss rate.
- C is the cost calculated using traditional methods.

In this formula, the cost is influenced by the ratio of link utilisation to available bandwidth, packet loss rate and the traditional cost, each scaled by their respective weights.

1.3 Routing Rules Implementation

In this step, we implement the new traffic distribution by updating the forwarding rules on network devices. The controller manages the forwarding tables and makes forwarding decisions on a per-flow basis using the five tuples, which includes source IP address, destination IP address, source port, destination port, and protocol. This enables the SDN controller to set precise paths for different types of traffic, ensuring that various flows are handled according to their characteristics or desired Quality of Service. Once the paths are determined, the SDN controller must push these updated rules to the network devices. This process involves modifying the forwarding tables of the devices to reflect the new routing paths. The controller typically uses a protocol like OpenFlow to communicate with the devices and install the necessary flow entries. It is crucial to ensure that the network remains consistent and that performance standards are maintained during the implementation of these rules.

1.4 Optimal Path Determination

In this step, we establish the initial shortest routing paths for all possible traffic flows within the network using traditional shortest path algorithms. These algorithms identify the most efficient path between two nodes based on criteria such as distance or link cost. Depending on the network's specific requirements and conditions, we can use algorithms like Dijkstra's Algorithm or Bellman Ford Algorithm. Typically, Dijkstra's algorithm is preferred for its efficiency and scalability, especially in complex networks, due to its lower time complexity compared to Bellman Ford. However, if the network's cost structures allow for negative weights, Bellman Ford would be necessary despite its higher computational cost.

To compute these paths, we assign weights to each link in the network based on factors such as:

- Hop Count: The number of intermediate devices the data passes through. A link weight of 1 for each hop favours paths with fewer hops.

- Latency: The time required for data to traverse the link. Links with lower latency may be assigned lower weights to make them more favourable.
- Bandwidth: Links with higher available bandwidth might receive lower weights to ensure they are utilised more frequently.

Once the shortest paths from the source node to all other nodes are determined, these paths are used to update the routing tables of network devices. Each entry in the routing table includes:

- Destination Network: The ultimate destination for the traffic.
- Next Hop Address: The address of the next device in the path toward the destination.
- Interface: The specific interface the device should use to forward the data to the next hop.

The controller automatically updates these tables to reflect the shortest paths computed by the algorithm.

1.5 Load Balancing

Based on the cost calculation mentioned earlier, the path with the lowest overall cost (F) is selected for routing traffic. This calculation takes into account link utilisation, available bandwidth and packet loss rate, which facilitates load balancing. By considering all three factors, we ensure effective load distribution. For instance, if a path has high link utilisation, the term w_1e^L will be low, increasing the cost of that route and thus discouraging its use. The use of an exponential function penalises routes with any unfavourable factor more significantly compared to a linear cost approach, which would not penalise routes as strongly.

1.6 Dynamic Adjustment

The network must continuously monitor traffic conditions and adjust routing paths as needed. This dynamic adjustment is crucial, especially in the event of a link or router failure. When such a failure occurs, the controller promptly recalculates the cost of available paths and determines the most suitable alternative route. This real time adaptation helps maintain network performance and reliability by promptly responding to disruptions and ensuring that traffic continues to flow efficiently. Additionally, the controller may use updated metrics and historical data to make informed decisions about path adjustments, enhancing the network's overall resilience and efficiency.

1.7 Network Security and Traffic Segregation

Security and isolation are essential for ensuring that the network functions securely and that traffic is segregated according to policies and security requirements. In this step, we implement measures to safeguard sensitive data, comply with regulatory standards, and prevent different types of traffic from interfering with each other. To achieve this, we recommend using Virtual Routing and Forwarding (VRFs) for traffic segmentation,

Multiprotocol Label Switching (MPLS) for efficient path selection, and Internet Protocol Security (IPsec) for encryption. These measures collectively enhance network security and ensure proper traffic isolation.

1.7.1 Security

Certain types of traffic, such as data from financial applications or confidential communications, may need to be routed over more secure paths. These paths should be selected based on criteria such as minimal exposure to the internet, support for encryption, or physical security of the infrastructure.

MPLS links will be prioritised for routing sensitive traffic. MPLS enables the creation of dedicated paths with specific Quality of Service (QoS) and security features, reducing exposure to less secure or untrusted network segments. IPsec will be implemented on these MPLS links to provide encryption, ensuring that data remains protected even if intercepted. IPsec is designed to integrate seamlessly with MPLS, offering end to end encryption without compromising performance.

The routing algorithm will be configured to avoid routing sensitive traffic through public or shared internet connections, even if these paths are shorter or less congested. Instead, it will favour paths that use dedicated MPLS links with IPsec encryption. The algorithm will continuously evaluate the security level of each potential path segment, excluding any segments that do not meet the required security standards.

1.7.2 Isolation

To achieve traffic isolation, two common methods are VLANs (Virtual LANs) and VRFs (Virtual Routing and Forwarding).

VLANs enable network administrators to divide a physical network into multiple logical networks. Devices on different VLANs cannot directly communicate with each other, even if they are connected to the same physical switch. Traffic is tagged with an ID when it enters the network, and switches are configured to handle these tags, allowing only traffic from specific tags on designated ports. By assigning different traffic types to different VLANs, network isolation is maintained.

On the other hand, VRFs allow multiple instances of routing tables to coexist within a single router or network device, effectively creating separate virtual routers on the same physical hardware. Each VRF has its own routing table, isolated from the others, and network interfaces are assigned to specific VRFs. This ensures that traffic entering the router on a given interface is processed according to the correct routing table.

While VLANs provide isolation at Layer 2, enabling communication between VLANs requires a Layer 3 device, which can be complex and potentially introduce security risks. Managing VLANs can also become cumbersome in large networks with many segments. In contrast, VRFs operate at Layer 3, where routing occurs, offering more granular control over traffic. They are scalable and allow for the creation of multiple isolated virtual routers within a single physical router, each with its own routing policies, making them more secure. Therefore, we propose using VRFs for traffic isolation and security in our approach.

1.8 Regular Performance Review

Networks are dynamic and frequently change, making periodic reevaluation essential for maintaining optimal performance. This process ensures that the network adapts to these changes, preventing congestion, optimising resource usage, and meeting quality of service and security requirements.Our algorithm includes regular assessments of network performance to re-calibrate routing decisions and ensure optimal operation. We plan to implement re-evaluation every few minutes to manage network traffic promptly.

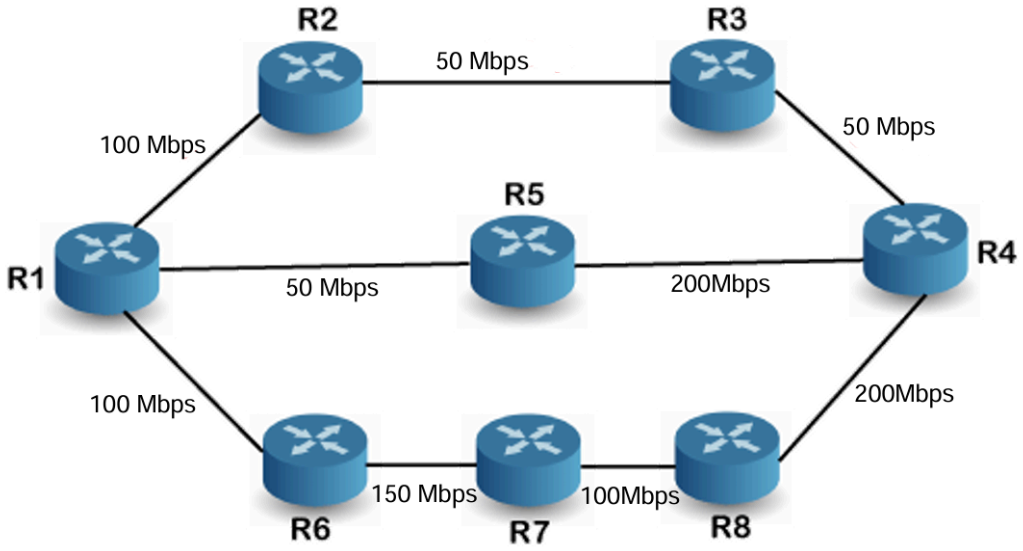
To achieve this, we continuously monitor key network parameters, including link utilisation, latency, packet loss, jitter, throughput and error rates. Utilisation and latency are tracked in real time, while throughput and error rates are collected less frequently, as they require data from specific time intervals to provide accurate measures.

The collected metrics are analysed to determine if routing paths need adjustment. This analysis helps identify trends and anomalies, such as sudden increases in link utilisation and unusual traffic patterns that may indicate policy violations.
Based on this analysis, we recalibrate routing decisions, adjusting paths, traffic policies, and network configurations to maintain optimal performance. We also assess whether load balancing is needed to distribute traffic across multiple paths, preventing overloading of any single link. Additionally, QoS adjustments may be made to prioritise critical applications, ensuring they receive the necessary bandwidth and low latency paths.

Bandwidth reservations and allocations are adjusted according to current usage trends. If certain links consistently approach their utilisation thresholds, we may lower those thresholds to accommodate future traffic. After recalibrating these decisions, the controller updates the forwarding rules, QoS policies, and security configurations on network devices. We also implement feedback loops to monitor the immediate effects of changes to ensure they achieve the desired results. Changes are deployed across the network, with a phased implementation for large, complex networks to minimise risks. A rollback procedure is established in case the changes lead to unintended negative consequences. Following deployment, key metrics are closely monitored to validate the impact of the changes. We compare these metrics with previous ones to determine if the issues have been resolved and ensure that no new issues or vulnerabilities are introduced. This ongoing process relies on real-time data to adjust paths and policies as needed.

1.9 Example Network Scenario

In this network, we have routers R1, R2, R3, R4, R5, R6, R7, and R8 interconnected with links of varying capacities, as shown in the diagram. The controller needs to manage traffic demands efficiently to prevent congestion and ensure optimal performance.



Traffic Demand

Suppose there is a traffic demand of 150 Mbps from router R1 to router R4. The shortest path based on the highest capacity links and the traditional method would be R1 → R6 → R7 → R8 → R4, as these links have the highest capacities (100 Mbps, 150 Mbps, 100 Mbps, and 200 Mbps, respectively).

Traffic Management

Initial Path Selection

The controller initially routes the 150 Mbps traffic via the path R1 → R6 → R7 → R8 → R4.

1. **Traffic Utilization Monitoring**: The controller monitors the traffic and realizes that routing all 150 Mbps through this path would lead to significant utilization, especially on the R6 → R7 link (100% utilization) and R7 → R8 link (100% utilization). This could lead to potential congestion or performance degradation.
2. **Alternative Path**: To avoid overutilization of these links, the controller decides to split the traffic. One option is to send part of the traffic through an alternative path, such as R1 → R2 → R5 → R4 or R1 → R6 → R5 → R4.

Traffic Split

- **Primary Path (R1 → R6 → R7 → R8 → R4)**: 100 Mbps of traffic is routed through this path.
- **Alternative Path (R1 → R2 → R5 → R4)**: The remaining 50 Mbps of traffic is routed through this path.

Load Balancing and Optimization

By splitting the traffic, the controller ensures that no single link becomes overutilized. The R1 → R6 → R7 → R8 → R4 path carries 100 Mbps, utilizing 67% of the capacity on the R6 → R7 link, while the R1 → R2 → R5 → R4 path utilizes 50 Mbps on each of the links involved, remaining within safe utilization limits.

This approach not only prevents congestion but also maximizes the overall network performance by utilizing available link capacities efficiently. The controller dynamically adjusts the traffic split based on real-time network conditions, ensuring that traffic demands are met without overloading any part of the network.

1.10 Algorithm Overview

Here's a Pseudocode for the Routing Algorithm for Traffic Engineering:

```
// Initialize network information
    Function NetworkInitialization()
        networkTopology = DiscoverNetworkTopology()
        networkCapabilities = GatherNetworkCapabilities()
        initialConditions = GatherInitialConditions()
        trafficMatrix = BuildTrafficDemandMatrix()
        processedData = ProcessCollectedData()
        linkCapacities = UnderstandLinkCapacitiesAndConstraints()
        utilisationThreshold = SetUtilisationThresholds(linkCapacities)
    EndFunction

    // Build traffic demand matrix
    Function BuildTrafficDemandMatrix()
        matrix = InitializeMatrix(nodes)
        for each (src, dst) in matrix
            matrix[src][dst] = EstimateTraffic(src, dst)
        EndFor
        Return matrix
    EndFunction

    // Process collected data
    Function ProcessCollectedData()
        data = CollectData()
        Return AssignWeights(data)
    EndFunction

    // Discover network topology
    Function DiscoverNetworkTopology()
        topology = InitializeTopology()
        for each link in network
            topology[link] = DiscoverLinkInfo(link)
        EndFor
        Return topology
    EndFunction

    // Understand link capacities and constraints
    Function UnderstandLinkCapacitiesAndConstraints()
        capacities = InitializeCapacities()
        for each link in network
            capacities[link] = (DetermineCapacity(link), IdentifyConstraints(link))
        EndFor
        Return capacities
    EndFunction

    // Monitor traffic and network conditions
    Function MonitorTrafficAndNetworkConditions()
        While NetworkIsActive
            for each link in network
                cost = w1 * (GetCurrentUtilisation(link) / GetAvailableBandwidth(link)) + w2 *
GetPacketLossRate(link) + w3 * CalculateTraditionalCost(link)
                pathCosts[link] = cost
            EndFor

            DetermineOptimalPaths()
            ApplyRoutingRules()
            DynamicAdjustment()
            CheckSecurityAndIsolation()
            PerformRegularPerformanceReview()
        EndWhile
    EndFunction
```

```
// Determine optimal paths
  Function DetermineOptimalPaths()
    for each src in nodes
      for each dst in nodes
        shortestPath = FindShortestPath(src, dst, pathCosts)
        UpdateRoutingTable(src, dst, shortestPath)
      EndFor
    EndFor
  EndFunction

  // Apply new routing rules
  Function ApplyRoutingRules()
    for each flow in flows
      path = DeterminePathForFlow(flow)
      UpdateForwardingRules(flow, path)
    EndFor
  EndFunction

  // Dynamic adjustment of paths
  Function DynamicAdjustment()
    if LinkOrRouterFails()
      RecalculatePathCosts()
      UpdateRoutingTableWithNewPaths()
    EndIf
  EndFunction

  // Check network security and traffic segregation
  Function CheckSecurityAndIsolation()
    ImplementTrafficSegregationPolicies() // using VLANs or VRFs
    EnsureDataEncryptionAndSecurity() // using IPsec
  EndFunction

  // Regular performance review
  Function PerformRegularPerformanceReview()
    metrics = CollectNetworkMetrics()
    if MetricsIndicateAdjustmentNeeded(metrics)
      RecalibrateRoutingDecisions()
      UpdateForwardingRulesAndPolicies()
      MonitorImpactOfChanges()
    EndIf
  EndFunction

  // Main execution
  NetworkInitialization()
  MonitorTrafficAndNetworkConditions()
EndAlgorithm
```

This routing algorithm developed aims to enhance network performance by focusing on both Traffic Engineering and Quality of Service. This algorithm facilitates dynamic traffic routing across multiple paths, ensuring optimal utilization of all network links. Unlike traditional protocols that rely solely on the shortest path, this method considers factors like link status, interface bandwidth, and current traffic conditions. By continuously monitoring these variables, the algorithm adjusts traffic distribution to prevent congestion, thereby maximizing the return on investment (ROI) for the network infrastructure.

2. Quality of Service (QoS) Routing Algorithm

Objective

To enhance the user experience by routing traffic according to the specific requirements of different network applications, thereby optimizing performance and efficiently managing network resources. Unlike conventional approaches that generally focus on maximizing network utilization without considering the individual needs of applications, this algorithm tailors routing decisions to meet the specific demands of each application type.

Different applications have unique network requirements to achieve high Quality of Service (QoS). For instance:

- **Streaming video content** requires substantial bandwidth, with latency being less critical due to the buffering features in these services.
- **Data transfers** benefit from high bandwidth and low packet loss to ensure quick and reliable file transmission.
- **Virtual meetings and video calls** are particularly sensitive to both low latency and minimal jitter to maintain smooth, real-time communication.
- **Online multiplayer games** require very low latency and minimal jitter to deliver a responsive and seamless gaming experience.
- **Voice calls over the internet** depend on low latency to avoid delays and maintain clear communication.
- **Email and instant messaging** demand moderate bandwidth and low packet loss to ensure messages are delivered promptly and reliably.

By refining the traditional traffic engineering algorithm to account for these specific needs, the network can provide optimized performance tailored to each application's requirements, ensuring a superior user experience across all types of services.

Algorithm Breakdown

Pseudocode for the QoS Routing Algorithm

```
// Step 1: Application Identification
function identifyApplication(packet):
    appType = DPI(packet)  // Use Deep Packet Inspection to identify application
    return appType

// Step 2: QoS Requirements Mapping
function mapQoSRequirements(appType):
    if appType == "VoIP":
        return {latency: <150ms, jitter: <30ms, bandwidth: "low", packetLoss: <1%}
    elif appType == "Video Streaming":
        return {latency: <300ms, jitter: <50ms, bandwidth: "high", packetLoss: <1%}
    else:
        return {latency: "moderate", jitter: "moderate", bandwidth: "medium", packetLoss: "moderate"}

// Step 3: Priority Assignment
function assignPriority(flowID, qosRequirements):
    if qosRequirements.latency == "low":
        return "high"
    elif qosRequirements.bandwidth == "high":
        return "medium"
    else:
        return "low"

// Step 4: Resource Reservation
function reserveResources(path, qosRequirements):
    RSVP(path, qosRequirements.bandwidth)  // Reserve required bandwidth and resources
```

```
// Step 5: Path Selection
function selectPath(source, destination, qosRequirements):
    possiblePaths = findPaths(source, destination)
    bestPath = null
    minCost = infinity

    for path in possiblePaths:
        cost = calculateCost(path, qosRequirements)  // Cost based on latency, jitter, etc.
        if cost < minCost:
            minCost = cost
            bestPath = path

    return bestPath
// Step 6: Adaptive QoS Management
function manageQoS(flowID, qosRequirements, bestPath):
    monitorNetworkConditions(flowID)
    if networkConditionsChange():
        newPath = selectPath(flowID.source, flowID.destination, qosRequirements)
        updateFlowPath(flowID, newPath)

// Step 7: Security and Isolation
function ensureSecurityAndIsolation(flowID):
    applyEncryption(flowID)
    applyTrafficIsolation(flowID)


// Main Function
function QoSRouting(packet):
    appType = identifyApplication(packet)
    qosRequirements = mapQoSRequirements(appType)
    priority = assignPriority(packet.flowID, qosRequirements)
    bestPath = selectPath(packet.source, packet.destination, qosRequirements)
    reserveResources(bestPath, qosRequirements)
    ensureSecurityAndIsolation(packet.flowID)
    manageQoS(packet.flowID, qosRequirements, bestPath)
```

1. Application Identification

Process:

- Implement Deep Packet Inspection (DPI) to analyse and identify the type of application (e.g., VoIP, video streaming, web browsing).
- Utilise Service Function Chaining (SFC) to classify and tag traffic based on the identified application.

Purpose:

- Different applications have unique QoS needs. Identifying the application allows the network to apply appropriate QoS policies, ensuring that each application receives the required network performance.

2. QoS Requirements Mapping

Process:

- Map each identified application to specific QoS parameters:
    - Latency: Acceptable delay before service degradation.
    - Jitter: Variability in packet arrival times.
    - Bandwidth: Required data throughput.
    - Packet Loss: Tolerable loss percentage.

Example:

- VoIP: Requires low latency (<150ms), minimal jitter (<30ms), and low packet loss (<1%).
- Video Streaming: Needs high bandwidth (e.g., 3-5 Mbps for HD), moderate latency, and minimal packet loss.

Purpose:

- Understanding these requirements allows for precise resource allocation and prioritisation, ensuring optimal performance for each type of traffic.

3. Priority Assignment

Process:

- Assign priorities using Class of Service (CoS) or Differentiated Services Code Point (DSCP) based on the application's QoS requirements.
- Example: VoIP might be given the highest priority, video streaming medium priority, and web browsing lower priority.

Purpose:

- Priority marking ensures that critical traffic (e.g., VoIP) is processed first and receives the necessary resources, maintaining high service quality.

4. Resource Reservation

Process:

- Use the Resource Reservation Protocol (RSVP) to reserve bandwidth and other necessary resources on the selected path for high-priority applications.
- Ensure that resources like buffer space and processing power are available to handle prioritized traffic.

Purpose:

- Reserving resources prevents congestion from impacting high-priority applications, maintaining consistent performance even under heavy network load.

5. Path Selection

Process:

- Calculate the best path using a Cost Function that prioritizes key QoS metrics (e.g., latency, bandwidth, packet loss).
- For example, a VoIP call may prioritize the path with the lowest latency, while video streaming may prioritize bandwidth.

Purpose:

- Selecting the optimal path based on application-specific QoS needs ensures that users experience the best possible performance for their application.

6. Adaptive QoS Management

Process:

- Continuously monitor network conditions and adjust QoS parameters dynamically.
- Implement Traffic Shaping and Queuing Mechanisms like Priority Queuing (PQ) and Weighted Fair Queuing (WFQ) to manage traffic effectively.

Purpose:

- Real-time adaptation ensures that QoS is maintained even as network conditions change, preventing degradation in user experience.

7. Security and Isolation

Process:

- Implement End-to-End Encryption to secure sensitive traffic.
- Use Virtual LANs (VLANs) or Virtual Routing and Forwarding (VRF) to isolate traffic, especially in multi-tenant environments.

Purpose:

- Ensures that QoS is provided without compromising security, protecting data integrity and privacy.

Example Scenarios

- VoIP Call: Identified as high-priority, routed through the lowest latency path with reserved bandwidth. Even under congestion, the call quality remains high due to prioritized resources.
- Video Streaming: Requires high bandwidth and is routed through a path with sufficient bandwidth. The path may not be the shortest but ensures smooth streaming without interruptions.

Conclusion

The QoS routing algorithm dynamically adapts to the needs of different applications, ensuring that network resources are used efficiently to deliver the best possible user experience. By continuously monitoring and adjusting to network conditions, it maintains optimal performance across diverse traffic types while safeguarding security and isolation.

3 References

1. Moorkattil, Xavier, Software Defined Networking (SDN) (September 27, 2022). Available at SSRN https://ssrn.com/abstract=4230813
2. Badotra, Sumit. (2017). A Review Paper on Software Defined Networking. International Journal of Advanced Computer Research. https://www.researchgate.net/publication/319550243_A_Review_Paper_on_Software_Defined_Networking
3. Maria Daniela Tache,Ovidiu Păscuțoiu, Eugen Borcoci (2024) Optimization Algorithms in SDN: Routing, Load Balancing, and Delay Optimization https://www.mdpi.com/2076-3417/14/14/5967
4. Faezi, S., Shirmarz, A. A Comprehensive Survey on Machine Learning using Software Defined Networks (SDN). Hum-Cent Intell Syst 3, 312–343 (2023). https://doi.org/10.1007/s44230-023-00025-3
5. Majda Omer, Elbasheer Abdulaziz, Aldegheishem Jaime, Lloret Nabil Alrajeh (2021) A QoS-Based routing algorithm over software defined networks https://www.sciencedirect.com/science/article/abs/pii/S1084804521002162
6. Blake, S., "An architecture for differentiated services." RFC 2475, 1998. https://datatracker.ietf.org/doc/html/rfc2475
7. Braden, R., et al. "Resource ReSerVation Protocol (RSVP)--Version 1 Functional Specification." RFC 2205, 1997. https://dl.acm.org/doi/10.17487/RFC2205
8. Chen, S., and K. Nahrstedt. "An overview of quality-of-service routing for next-generation high-speed networks: Problems and solutions." IEEE Network, 1998. https://ieeexplore.ieee.org/abstract/document/752646
9. Guerin, R., . "QoS routing mechanisms and OSPF extensions." GLOBECOM, 1997. https://ieeexplore.ieee.org/document/644603
10. Ferguson, P., and G. Huston. "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing." RFC 2827, 2000. https://datatracker.ietf.org/doc/html/rfc2827