5G Quality-on-Demand

Infosys Limited, Bangalore, India

December 2025

Document No		Version.Rev	
Authorized by		Date	

Table of Contents

1. Revision History

Revision	Modifications	Date	Author
A	First version	02-12-2025	Infosys Private 5G Deployment team

2. Introduction

Private 5G networks enable enterprises and industries to securely and efficiently onboard use case applications by providing high-speed, low-latency connectivity tailored to specific organizational needs. This dedicated infrastructure ensures enhanced reliability, scalability, and control over data traffic, allowing seamless integration of IoT devices, automation systems, and other mission-critical applications. In today's digital era, as organizations move towards agile and continuous delivery pipelines, the demand for real-time quality assurance grows exponentially. As diverse applications demand tailored network performance, Quality on Demand (QoD) emerges as a critical capability. QoD in 5G allows dynamic, context-aware allocation of network resources to meet specific service-level requirements in real time. In essence, QoD transforms 5G into an intelligent platform capable of meeting the evolving demands of digital ecosystems.

3. Problem Statement

Traditional network management systems lack the ability to measure and adjust quality metrics in real time. While they can monitor indicators such as latency and throughput, they do not provide a mechanism to dynamically request or enforce Quality of Service (QoS) based on changing traffic conditions. This limitation leads to inconsistent service delivery for the applications that require assured performance, particularly under dynamic traffic conditions. Below are the key challenges faced by the traditional systems. Our traditional systems lack the real-time ability to measure and adjust quality metrics effectively during. Even though we can monitor basic network health indicators (like latency or throughput), we have no way of dynamically requesting Quality of Service from network dependent on current traffic scenario.

- Networks cannot adapt to sudden spikes in demand or prioritize mission-critical services.
- Operators are unable to offer guaranteed performance tiers for premium applications.
- Applications like autonomous vehicles, industrial automation, and immersive AR/VR require ultra-reliable low-latency communication, which current systems cannot consistently deliver.
- Manual interventions and static configurations lead to reduced resource utilization and degraded user experience.

4. Solution Description

To overcome the limitations of static QoS management, Quality on Demand (QoD) for 5G introduces a dynamic, intelligent approach to service assurance. Quality on Demand offers the application developer the capability to request stable latency (reduced jitter) or minimum throughput for specified application data flows between application clients (within a user device) and Application Servers (backend services) End Users or applications can request specific network quality levels depending on their current activity, enabling optimal performance for diverse scenario. The key components of the solution are

- Free5gc 5G Core serves as the control and user plane.
- Infosys Edge Application Management (IEAM) for QoD Provisioning

- Nephio for Orchestration

Use Case Scenarios:

- Remote control of machines and vehicles (e.g. Automated Guided Vehicles, drones, robotic arms, factory production lines) require stable data throughput and low latency to ensure secure and efficient operations.
- Media and entertainment (e.g. online gamers and viewers of real-time streaming) require a network with a high level of performance to ensure good user experience.
- Computer vision and remote video processing applications (e.g. sending a continuous video stream to their backends for processing) require stable data throughput and low latency to generate time-sensitive outputs, such as alarms and events for computer vision, or a produced video stream for their audience.

This use case can be developed by leveraging and enhancing the functionalities of existing Linux foundation projects such as CAMARA, Nephio, Free5GC.

- Nephio should be enhanced further to be integrated with latest Free5GC version of v4.0.1 as the current version integrated with Nephio does not support Multiple QoS feature for Data plane.
- Nephio should be enhanced to support subscriber QoS provisioning capability.
- CAMARA QoD API integration with Nephio using Infosys Edge Application Management (IEAM).

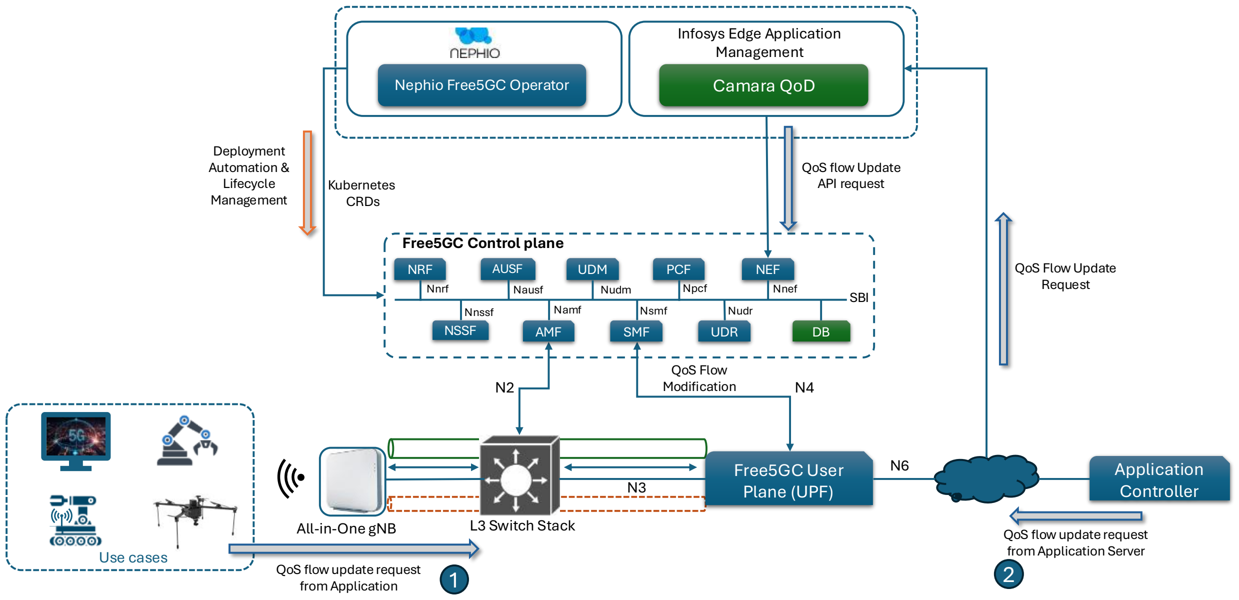


Figure 1. Solution Architecture

The above architecture shows how QoS (Quality of Service) flow updates are managed in free5GC 5G core using Infosys Edge Application Management (IEAM).

Applications/various machines used in use-cases (like drones or robots) send QoS update requests to an application controller. When the controller detects a degradation in the network, it sends a QoS flow update request to IEAM where it manages all applications in edge network. IEAM leverages Camara QoD APIs in the backend to forward the request to free5GC Control Plane (via NEF) which in turn communicates with PCF to apply policy changes. Camara QoD APIs are open standardized APIs which helps in exposing network capabilities. Updated QoS parameters are pushed to the UPF (User Plane Function), which handles traffic for the gNB and connected devices. Nephio Free5GC Operator automates deployment and lifecycles using Kubernetes CRDs.

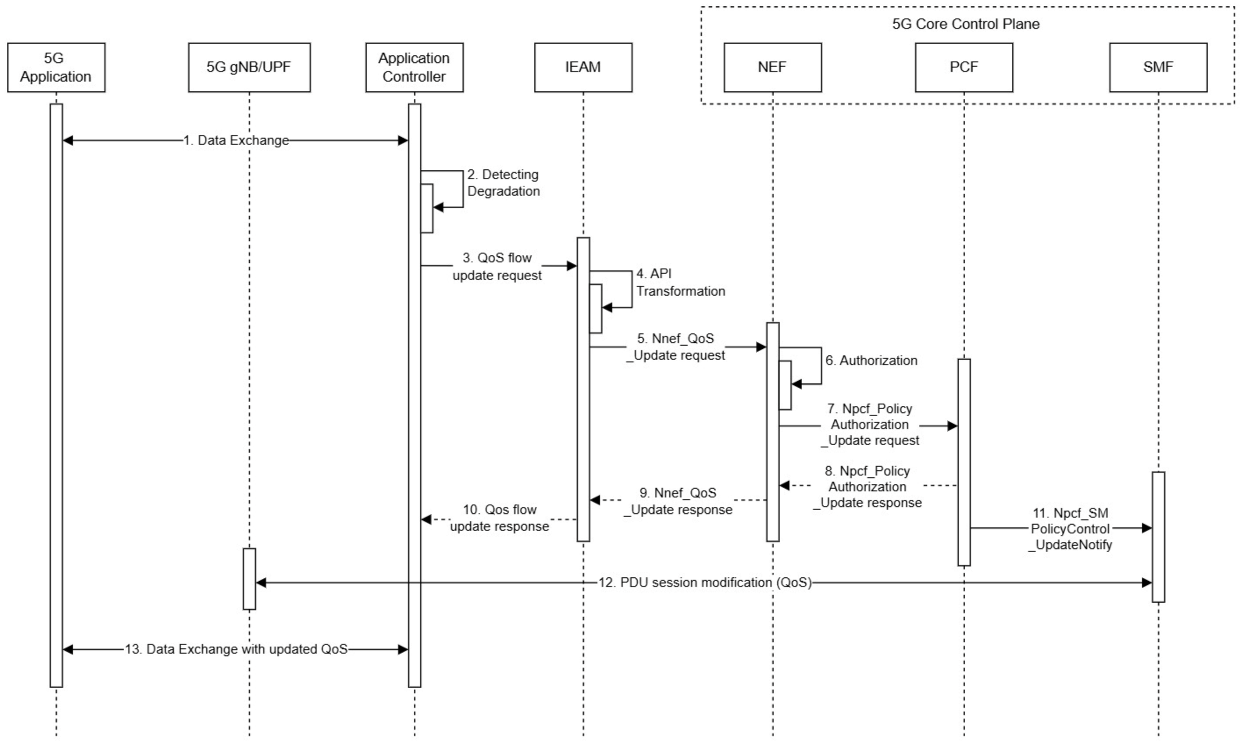

5G Core Control Plane
5G Application
5G gNB/UPF
Application Controller
IEAM
NEF
PCF
SMF
1. Data Exchange
2. Detecting Degradation
3. QoS flow update request
4. API Transformation
5. Nnef_QoS _Update request
6. Authorization
7. Npcf_Policy Authorization _Update request
8. Npcf_Policy Authorization _Update response
9. Nnef_QoS _Update response
10. Qos flow update response
11. Npcf_SM PolicyControl _UpdateNotify
12. PDU session modification (QoS)
13. Data Exchange with updated QoS

Figure 2. Quality Of Service (QOS) Flow

1. User Equipment (UE) establishes a connection, and data traffic is exchanged between the 5G application and the Application Controller.
2. The Application Controller monitors the data flow and identifies a degradation in service quality during traffic exchange
3. Upon detecting the degradation, the Application Controller initiates a Quality of Service (QoS) update request and transmits it to the Infosys Edge Application Management (IEAM) platform over CAMARA QoD API request.
4. Within IEAM, the CAMARA QoD API is transformed to NEF AF Session with QoS API to ensure interoperability, and the formatted request is forwarded to the Network Exposure Function (NEF).
5. The NEF receives the QoS flow update through the Nnef_AFSessionWithQoS_Update request.
6. Upon receiving the request, the NEF performs the necessary authorization procedures to validate and permit further processing of the request.
7. Once authorization is successfully completed, the NEF forwards the QoS update request to the Policy Control Function (PCF) by invoking the Npcf_PolicyAuthorization_Update request.

8. PCF receives the request and does the policy authorization and continues with subsequent process of PCC rule mapping and will send Npcf_PolicyAuthorization_Update response to NEF.
9. The NEF, after processing the response received from the PCF, returns a Nnef_AFSessionWithRequiredQoS_Update response to the IEAM.
10. IEAM, after processing the response, will send the QoS flow update response to the Application controller from which the QoS flow update request got triggered.
11. Meanwhile, once PCF updates its PCC rule, it will send Npcf_SMPolicy_Control_Update Notify to SMF.
12. Upon receiving the updated policy parameters from PCF, SMF will update the user plane by sending the information to UPF using PFCP modification and to gNB using N1N2 message transfer PDU modification.
13. Following the successful PDU modification, subsequent data traffic is transmitted using the updated QoS flow parameters, thereby improving the quality of service experienced by the UE.

5. Prerequisites

Free5GC Core

- Version: v4.0.1

CAMARA QOD API

- Version: v1.1.0

Integration with UERANSIM

Ensure the following environment setup when integrating Free5GC with UERANSIM:

- Operating System: Ubuntu 22.04.5 LTS
- Docker Version: 27.5.0
- GTP5G Kernel Module Version: 0.9.11

Integration with Actual Radio (gNB)

For integration with actual radio hardware (e.g., BTI gNB), use the following versions:

- Operating System: Ubuntu 22.04.5 LTS
- Docker Version: 27.5.0
- GTP5G Kernel Module Version: 0.9.14

Compute and Storage Requirements

The solution is tested with below compute and storage resources:

1. CPU: 16vCPU
2. Memory: 16GB

3. Storage: 250GB

6. CAMARA QOD API and QOS Support

6.1 Introduction:

The Camara Quality-On-Demand API **(v1.1.0)** provides an interface for the developers to request stable latency or prioritized throughput managed by the networks. The developer has a pre-defined set of Quality of Service (QoS) profiles which they could choose from depending on their latency or throughput requirements.

6.2 API functionality:

The usage of the QoD API is based on QoS profile classes and parameters which define App-Flows. Based on the API, QoS session resources can be created, queried, and deleted. Once an offered QoS profile class is requested, application users get a prioritized service with stable latency or throughput even in the case of congestion. The QoD API has the following characteristics:

- A specified App-Flow is prioritized to ensure stable latency or throughput for that flow.
- The prioritized App-Flow is described by providing information such as device IP address (or other device identifier) & application server IP addresses and port/port-ranges.
- The developer specifies the duration for which they need the prioritized App-flow.
- Stable latency or throughput is requested by selecting from the list of QoS profiles made available by the service provider to map latency and throughput requirements.
- The API consumer can optionally also specify callback URL (sink param) on which notifications for the session can be sent.

6.3 Camara API Request Flow:

- The CAMARA API flow consists of a CAMARA API consumer (Application Server) that sends CAMARA QoD API requests to the IEAM (Infosys Edge Application Management).
- IEAM is an Infosys platform which is used to receive and transform the camara API request into NEF-Compatible AF Session with QoS API formats by retrieving the required QoS profile details by QoS Profile name before being sent to the NEF endpoints in Free5gc.

Camara QoD API Request Flow:

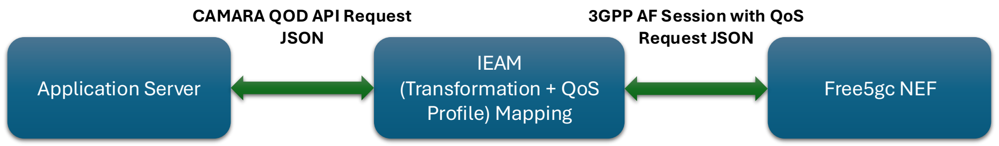

CAMARA QOD API Request JSON
3GPP AF Session with QoS Request JSON
Application Server
IEAM (Transformation + QoS Profile) Mapping
Free5gc NEF

Figure 3. Camara API Request Flow

6.4 Camara Quality-On-Demand API (v1.1.0):

6.4.1 CAMARA QOD API (POST and PATCH):

The CAMARA QoD API implementation for both POST and PATCH requests follows a common structure. The only difference between the two is the QoS profile value, which will be updated in the PATCH request payload. All other fields remain consistent with the original POST request format.

POST/PATCH ***{{apiRoot}}/quality-on-demand/v1/sessions***

```
{
  "device": {
    "ipv4Address": {
      "publicAddress": "10.61.0.1"  → UE IP Address
  }
  },
  "applicationServer": {
    "ipv4Address": "10.100.200.1"  → Application Server IP Address
  },
  "qosProfile": "voice_standard",  → QoS Profile name

  “flowDirection”: “uplink”,
  "sink": "https://endpoint.example.com/sink",
  "sinkCredential": {
    "credentialType": "PLAIN"
  },
  "duration": 3600
}
```

On receiving the above CAMARA request to IEAM:

- IEAM checks the session database for any existing session with the same ueIpv4Addr.

  If not found (POST Flow):

  - A new afId/scsAsID and sessionId will be generated.
  - Appropriate QoS profile components will be selected from IEAM, and flow description will be created using the AF IP and flow direction.
  - POST request is sent to NEF:

    POST ***https://<nef-host>:8000/3gpp-as-session-with-qos/v1/<afId or scsAsId>/subscriptions***

  - After sending request to NEF, IEAM will extract the subscriptionId from the NEF response. Store session metadata and will transform the response to match CAMARA QoD format. Following which, the response will be transformed to the Application Server (AF).

If found (PATCH Flow):

- PATCH request will extract the existing afId/scsAsId and subscriptionId.
- Generate a PATCH payload for sending it to NEF with the updated QoS profile an d flow description.
- Sent PATCH request to NEF:

PATCH ***https://<nef-host>:8000/3gpp-as-session-with-qos/v1/<afId or scsAsI d>/subscriptions/subscriptionId***

- After sending PATCH it updated the session metadata and responded to the AF with the session ID.

6.4.2 CAMARA QOD API (DELETE):

When session expires, IEAM generates a DELETE request to the NEF using the associated afI d (or scsAsId) and subscriptionId in the resource URI.

- Upon receiving a successful response from NEF, it removes the corresponding session meta data from the IEAM database.

DELETE ***https://<nef_host>:8000/3gpp-as-session-with-qos/v1/scs123/subscriptions/1***

- If the session needs to be deleted before duration expiry, the AF(AS) can trigger DELETE with the below URI. The sessionId (UUID format) will be sent in the below URI to be transformed i nto subscriptionId which is mapped against it for forwarding to NEF (*shown in above URI*).

DELETE ***{{apiRoot}}/quality-on-demand/v1/sessions/: sessionId***

7. NEF Development Code for managing QOS in Free5GC

7.1 Key Objective:

The below mentioned process is developed in the Free5gc NEF source code in which Quality on Demand use case needs to be processed.

- For 5G Quality on Demand use case, AF(IEAM) will send AF Session with QoS API to NEF with required QoS parameters and identifiers.
- For handling AF session with QoS API, developed the code in NEF for create, update and DELETE QoS subscription.
- Interacting with PCF via Npcf_policyauthorisation_api through Consumer.pcf_service for creating, updating and deleting AppSession in PCF.
- Developed and achieved Authentication and Authorization in NEF for AsSessionwithQoSApi.

7.2 Existing NEF code for managing QOS in free5gc

Existing NEF does not have the feature for handling the AsSessionWithQos API. So, we have developed code for handling the AsSessionWithQoS API as defined in 3GPP TS 29.122 [v17] in NEF.

7.3 Development for AS Session with Required QoS subscriptions

AsSessionWithQoS API is an interface (RESTful API) defined by 3GPP. It lets an application server (AF) or Service Capability Server (SCS) create a session with the Network Exposure Function (NEF). When creating this session, the API allows the application to request specific Quality of Service (QoS) settings based on what the app or service needs. All resource URIs of this API should have the following root:

{apiRoot}/3gpp-as-session-with-qos/v1

"apiRoot" is set as described in clause 3GPP TS 29.122 [v17] “apiName" shall be set to "3gpp-as-session-with-qos" and "apiVersion" shall be set to "v1" for the version defined in the present document.

Below figure depicts the code flow of NEF:

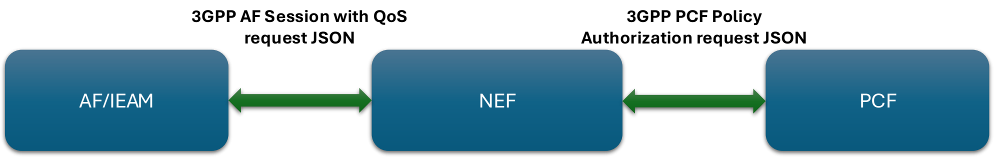

3GPP AF Session with QoS request JSON
3GPP PCF Policy Authorization request JSON
AF/IEAM
NEF
PCF

Figure 4. NEF Code Flow

7.3.1 AS Session with Required QoS Subscriptions Methods

Implemented below API endpoints in the SBI package for managing QoS subscriptions. These include POST, PATCH, and DELETE methods for handling subscription lifecycle.

- POST /: scsAsId/subscriptions - Creates a new QoS subscription.
- PATCH /: scsAsId/subscriptions/: subscriptionId - Updates an existing QoS subscription.
- DELETE /: scsAsId/subscriptions/: subscriptionId - Deletes a QoS subscription.

Each endpoint includes logic for content type validation, data deserialization, and response handling.

7.3.1.1 AsSessionWithQoSSubscription POST -Create

- The POST method creates a new subscription resource for a given AF in NEF. The NEF construct the URI of the created resource using that URI as per 3GPP TS 29.122 (v17).
- Resource URI: *POST* {***apiRoot}/3gpp-as-session-withqos/v1/{scsAsId}/subscriptions***
- Validates the request and deserializes into *AsSessionWithQoSSubscription (contains fields like Dnn, Gpsi, UeIpv4Addr, NotificationDestination, and MultiModDatFlows) which is present in qos_models.go.
- The POST request for creating a subscription reaches “*PostAsSessionWithQosSubscription”* in the Processor layer (asSessionwithQos.go).

- Checks for required identifiers (e.g., Gpsi or UeIpv4Addr). Ensures/locks AF context; creates a new subscription.
- Converts the QoS subscription to AppSessionContext and requests the downstream App Session creation via PCF "*PostAppSessions*".
- "*PostAppSession*" calling the Npcf_Policyauthorisation_api responsible for sending the POST request in PCF and creating AppSessionContext.
- Retrieves AppSessionID and store it with new subcriptionId.
- After completed successfully, sets Self to the created resource URI and returns 201 Created with Location.

7.3.1.2 AsSessionWithQoSSubscription PATCH -Modify

- As per 3GPP TS 29.122 (v17) the PATCH method allows to change the service information of an active subscription using below URI.
- ResourceURI: *PATCH {**apiRoot}/3gpp-as-session-withqos/v1/{scsAsId}/subscriptions/{subscriptionId}***
- Validates Content-Type JSON and Deserializes into *AsSessionWithQoSSubscriptionPatch Structure present in the qos_models.go. Logs the PATCH payload.
- The PATCH request for updating a subscription reaches "*PatchIndividualAsSessionWithQosSubscription ()*" present in Processer (asSessionwithQos.go) handling the PATCH request and update subscription.
- Finds the AF and the subscription for given subscriptionId; errors with 404 if not found.
- Ensures there's an active App Session (AppSessID) to PATCH.
- Convert the PATCH to AppSessionContextUpdateData using and calls "*PatchAppSession*".
- PCF "*PatchAppSession ()*" calling the Npcf_Policyauthorisation_api responsible for sending the PATCH request in PCF and updating AppSessionContext for given AppSessionID.Updates the local subscription state.
- Returns 200 OK with the updated subscription (or could return 204 No Content per spec; you currently return 200).

7.3.1.3 AsSessionwithQosSubscription DELETE — Terminate

- The DELETE method deletes the AsSessionWithQoSSubscription resource and terminates the related subscription as per 3GPP TS 29.122 [v17].
- Resource URI: *DELETE {**apiroot}/3gpp-as-session -with-qos/v1/{scsAsId}/subscriptions/subscriptionID)***
- The "*apiDeleteIndividualAsSessionWithQosSubscription*" function calling "*DeleteIndividualAsSessionWithQosSubscription* ()" present in Processer (asSessionwithQos.go) handling the DELETE request and removes subscription.
- Validates AF and subscription existence for given SubdcriptionId; returns 404 if missing. If an App Session exists for AppSessionId, calls PCF DeleteAppSession.
- *"DeleteAppSession ()"* calling the Npcf_Policyauthorisation_api responsible for sending the DELETE request in PCF and deleting AppSessionContext for AppSessionID.

- Removes subscription from “af.subs” completely for given subscriptionID. Returns 204 No Content.

7.3.2 Data Models and Structures

This section data model defined in the context package to support QoS subscription as per 3GPP TS 29.122 [v18] and authorization logic.

7.3.2.1 AsSessionMediaComponent

Represents a media component with flow information and bandwidth parameters as per 3GPP TS 29.122[v18]. AsSessionMediaComponent includes fields such as:

- FlowInfos
- MedCompN
- MedType
- MirBwUl
- MirBwDl
- MarBwUl
- MarBwDl

7.3.2.2 AsSessionWithQoSSubscription

Represents the structure for QoS subscription for POST requests as per 3GPP TS 29.122 [v18]. AsSessionWithQoSSubscription includes fields such as:

- SelfURI
- SupportedFeatures
- Dnn
- Snssai
- NotificationDestination
- UeIpv4Addr
- Gpsi
- MultiModDatFlows.

7.3.2.3 AsSessionWithQoSSubscriptionPatch

Represents the structure for PATCH operations to update QoS subscriptions as per 3GPP TS 29.122 [v18]. This includes fields such as:

- NotificationDestination
- MultiModDatFlows

7.3.2.4 AuthorizationJSON

Represents the structure used for OAuth token requests to get the token from NEF. This includes fields such as:

- Client_id
- Client_secret
- Grant_type

7.3.3 Mutual TLS Authentication

Mutual TLS (mTLS) authentication is a security mechanism that ensures both the client (AF) and the server (NEF) authenticate each other using TLS certificates before establishing a secure connection.

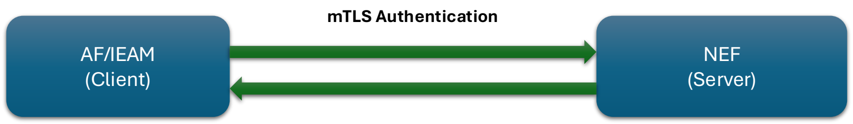


Figure 5. M-TLS Authentication between AF and NEF

As mentioned in clause 5.5 of 3GPP TS 33.187 [v17] we have done the mutual TLS authentication between AF and NEF have created the certificate and key for Application function (client) and NEF (server) signed by a Certificate Authority (CA).

Follow the steps below for m-TLS Authentication in NEF:

1.Create CA certificate.
2.Create Client (AF) key and certificate signed by CA.
3. Create Server (NEF) key and certificate signed by CA.
4. Add CA certificate and NEF key and certificate in free5gc-compose/cert.
5. Now go to "*free5gc-compose/config/nefcfg.yaml*" and make scheme as https, add CA certificate path.
6.After above step when will send the request towards NEF add CA certificate, Client key and certificate in the request.

7.3.4 OAuth Token Endpoint

1) NEF checks AF(IEAM) authorization using a token. It acts as the Authorization Server and generates tokens as given in 3GPP TS 29.122 [v17].
2) A new/token endpoint in the SBI package supports OAuth-based authorization. It accepts POST requests with application/x-www-form-urlencoded content, validates content type, parses form data, and checks required parameters: client_id, client_secret, grant_type.
3) Curl request to get token from NEF - *curl -v --cacert mTLS-certs/ca.pem --cert mTLS-certs/af-client.pem --key mTLS-certs/af-client.key POST https://<nef host IP>:8000/oauth2/token -H "Content-Type: application/x-www-form-urlencoded" -d "grant_type=client_credentials&client_id=scs123&client_secret=secret123"*
4) After validation, the request is wrapped in AuthorizationJSON and passed to the processor for token issuance.
5) The "*apiIssueOAuthToken ()*" calls "*IssueOAuthToken ()*" to generate the token.
   Processor steps:
   - Validate grant_type = client_credentials.
   - Verify client_id and client_secret.
   - Generate JWT using HMAC SHA-256 with claims (issuer, subject, audience, scope, iat).
   - Return JSON with access_token, token_type=Bearer, expires_in=0, and scope.

7.3.5 Authorization and Routing Setup

Endpoints are registered using the "*getAsSessionWithQoSEndpoints*" function and grouped under a common URI prefix using Gin's router. Group method "*util. check ()*" retrieve the token from authorization header.

As per 3GPP TS 29.122 [v17] It is the NEF responsibility to check whether the AF is authorized to use an API based on the "token". Once the NEF verifies the "token", it shall check whether the NEF identifier in the "token" matches its own published identifier, and whether the API name in the "token" matches its own published API name. If those checks are passed, the AF has full authority to access any resource or operation for the invoked API.

Authorization is enforced via middleware that checks the 'Authorization' header using the "*NefContext.AuthorizationCheck()*" method. Unauthorized requests are aborted with a401 response.

7.4 Final Outcome:

- After development, the new AsSessionwithQoSAPI for POST, PATCH, and DELETE subscription is getting handled successfully by the NEF.
- Create (POST): NEF validates and deserializes the QoS subscription request, converts it to an AppSessionContext, and sends it to PCF. On success, it stores the AppSessionID and returns a 201 Created with the resource URI.
- Modify (PATCH): NEF locates the subscription, converts the PATCH to AppSessionContextUpdateData, and updates the App Session in PCF. It updates its local state and returns 200 OK.
- Delete (DELETE): NEF deletes the subscription and AppSession in PCF, clears local state, and returns 204 No Content.
- Downstream Integration: All operations use the Npcf_Policyauthorisation API to manage App Sessions in PCF. PCF changes ensure correct QoS (MBR/GBR) propagation to SMF/UPF.
- Security & Validation: Endpoints enforce JSON content type, use Mutual tls Authentication and OAuth-based authorization, and log normalized payloads for traceability.

8. PCF Development Code for managing QOS in Free5GC

8.1. Key Objective:

The below mentioned process is currently available in the Free5gc code in which Quality on Demand use case need to be processed.

- For 5G Quality on Demand use case, AF will send AF Session with QoS API to NEF with required QoS parameters and identifiers.
- After that, NEF will send the App session context with required data to PCF via PCF policy authorization API as defined 3GPP TS 29.514 [v17].
- Once received, PCF will perform updating its PCC rule and will send the modified data to SMF via SM Policy control update notification.

- Once applied, SMF will check the state of User Plane Function and will send the Activate UPF session via PFCP modification process.
- Develop code in "*updateQosMedSubComp ()*" for updating the correct MBR, GBR values in PFCP modification packets in UPF.
- To develop code in PCF policy authorization code for PATCH and DELETE method to work properly it should remove specific subscription completely.

8.2. Existing PCF policy authorization code file for updating MBR, GBR and PATCH & DELETE request to work:

8.2.1 Existing Code for update MBR, GBR value in PFCP Modification:

1) In PCF policy authorization code, the POST or PATCH requests are passing from the PCF to the SMF, and then to the UPF via the PFCP Modification Procedure. QoS parameters for both POST and PATCH were not correctly reflected in the PFCP modification request. Specifically, the MBR and GBR values were set to 0.0, despite being properly defined in the request.

8.2.2 Existing Code for PATCH and DELETE request:

1) In PCF policy authorization code POST request was working fine and when a PATCH was applied, the new values were appended to the existing ones. This meant PATCH operations worked, but DELETE caused inconsistencies.
2) When DELETE was applied, the latest PCC rule associated with the App Session was deleted, causing the App Session itself to be removed. Upon recreation, it reverted to the previous PCC rule instead of the default values.

8.3. Development done PCF policy authorization code file for updating MBR, GBR and PATCH & DELETE request to work:

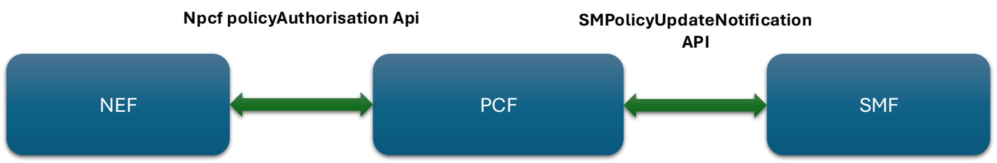


Figure 5. PCF Code Flow

8.3.1 Code changes for update MBR, GBR value in PFCP Modification:

1) After the development done to address this issue (8.2.1), a fix was implemented in the "*updateQosMedSubComp ()*" method within the policyauthorisation. go file. Source code is modified to ensure that the MBR and GBR values received in the request are correctly updated in the session context. As a result, the updated QoS parameters are now accurately reflected in the PFCP messages, and the corresponding UPF packets show the correct GBR and MBR values during packet inspection.

8.3.2 Code Changes for PATCH and DELETE request:

1) After the modification of above existing code (8.2.2), whenever a new PATCH is applied, the previous PCC rule assigned to that App Session is first tracked and deleted before sending updates to SMF.
2) This ensures that PATCH updates apply only the latest values, and DELETE operations remove the latest flow rules and AppSession cleanly, reverting to default values.
3) This code takes a snapshot of the current policy state including PCC rules, Traffic Control data, and QoS decisions. By storing these before any updates, the PCF can later detect what changed or was deleted when a new PATCH request arrives.
4) Update App Session and Track Deletions: Media Components Update: Converts request components to internal structure and updates them. AF Information Update: Updates AfAppId; AfRoutReq logic is commented out. Deleted Rules Tracking: Compares current vs previous PCC, TC, and QoS rules to identify removed entries. Store their IDs for SMF update.
5) Section merges the current and deleted rules into unified structures that will be sent in a Policy Update Notification. By setting deleted entries to `nil`, it ensures SMF is correctly informed about which rules should be removed.
6) Here, the merged policy decision (including both active and deleted rules) is included in the payload to be sent to SMF. This ensures that the SMF has the complete and correct state of all flow rules.
7) Finally, after the notification is sent, the deleted rules are removed from the PCF's internal state to maintain consistency and avoid stale data.

8.4 Final Outcome:

- After development, the QER in PFCP Session Modification sent by SMF to UPF now carries the correct Maximum Bit Rate (MBR) and Guaranteed Bit Rate (GBR) for both DL/UL—matching the AF request via NEF. Previously these values were falling back to 0.0 due to conversion logic; now they are assigned from MediaComponent as intended.
- In the developed Free5gc PCF Code, PATCH now replaces old rules, avoiding accumulation/appending. DELETE request removes the latest rules and reverts the App Session to default behavior (no stale PCC/TC/QoS remnants).
- SMF receives explicit null-valued entries for items to DELETE, so UPF removes outdated QER/FAR/URR cleanly.

9. SMF Developmental code for AMF forwarding in Free5GC

9.1. Key Objective:

The below mentioned process is currently available in the Free5gc code in which Quality on Demand use case need to be processed.

- AF sends an AF Session with QoS API request to PCF with required QoS parameters. NEF then forwards the App Session context to PCF via the Policy Authorization API (3GPP TS 29.514).

- PCF updates PCC rules and sends the modified policy to SMF through an SM Policy Control Update Notification (3GPP TS 29.512).
- SMF applies the session and PCC rules to the associated PDU Session context, triggers PFCP modification to update the UPF, and then initiates a PDU Session Modification towards gNB/UE.
- After PFCP modification, SMF sends an N1N2 Message Transfer to AMF, including "*PDUSessionModificationCommand*" and "*PDUSessionResourceModifyRequestTransfer*".
- AMF splits these into N1 and N2 messages and forwards them to UE and gNB using NAS and NGAP signaling.

9.2. Existing code in SM context update in SMF:

In Free5GC, the N1N2 Message Transfer procedure (3GPP TS 29.502 v17) is triggered towards the AMF during UE-initiated SM context create, update, or release requests. This applies to PDU Session establishment, modification, and rejects scenarios.

For UE- initiated SM context update (pdu_session. go):

1) "*HTTPUpdateSmContext ()*" handles the HTTP POST request for SM context update. "*HandlePDUSessionSMContextUpdate ()*" locates the existing SM context for modification.
2) The request includes BinaryDataN1SmMessage (NAS message), which is decoded and inspected for message type. "*HandlePDUSessionModificationRequest* ()" processes the modification and returns a response.
3) On success, the response is encoded, and:
    a. sendGSMPDUSessionModificationCommand prepares the N1 message.
    b. BuildPDUSessionResourceModifyRequestTransfer prepares the N2 message.
4) Both N1 and N2 messages are combined, and "*sendGSMPDUSessionModificationCommand ()*" triggers the "*N1N2MessageTransfer ()*" request to AMF.
5) AMF receives the request via "*HTTPN1N2MessageTransfer*" and processes it in "*HandleN1N2MessageTransferRequest*", splitting N1 and N2 containers for NAS and NGAP signaling.

For Network- initiated SM context update (pdu_session. go):

1) In the existing code of SMF for network initiated PDU modification where SM policy control update notification is handled, the process is getting ended at "*ActivateUPFSession*" function which handles the PFCP modification process towards UPF. So, have done development for N1N2transfer request triggered and reaches AMF/GNB

9.3. Development done in SM policy control update code for including N1N2 Message transfer:

In the development code, once the PFCP modification response is received, we have triggered the functions for invoking N1N2 message transfer toward AMF which is network-initiated. Below are the development procedures.

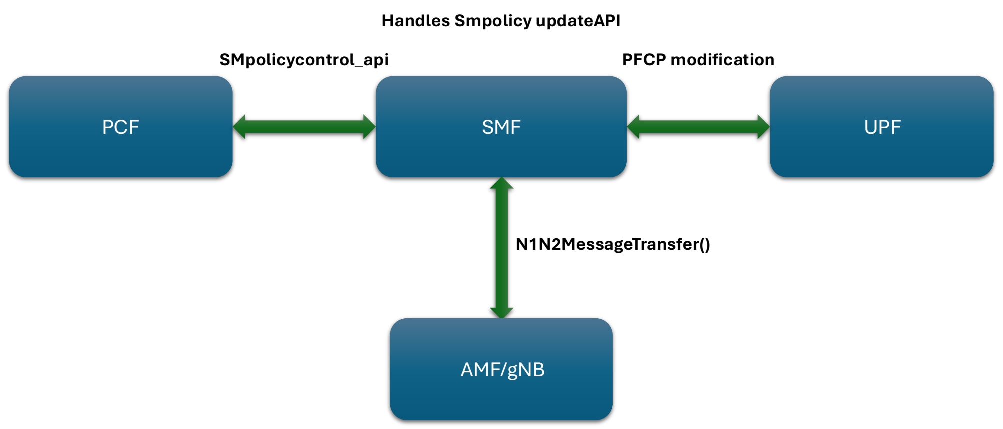


Figure 6. SMF Code Flow

1) The primary requirement is the construction of binary N1 and N2 message containers which need to be passed as a part of N1N2 Message transfer request. In our case, since it is a network initiated PDU session modification command, there is no choice of getting the Binary messages from the UE request. So, we have built both.
2) In the existing code, function named "BuildGSMPDUSessionModificationCommand" is re-used for building PDU session modification command with "smContext" as only input returns the output as NAS N1 message in Binary format.
3) After that, "BuildPDUSessionResourceModifyRequestTransfer" which constructs and returns a binary N2 message which need to be the part of "sendGSMPDUSessionModificationCommand" that triggers N1N2 Message transfer.
4) In the above function, another function is getting called which is "BuildNgapQosFlowAddOrModifyRequestItem" which is responsible for QoS flow modify in the NGAP n2 message. This includes the QoS characteristics into the N2 NGAP message container which will be then passed with N1N2 message transfer.
5) The development of invoking N1N2 message transfer as per 3GPP TS 29.502 v17 we have made in the "notifier. go" file after activating UPF Session for PFCP modification.
6) The development has made for *"BuildGSMPDUSessionModificationCommand ()"*, "BuildPDUSessionResourceModifyRequestTransfer ()" and then invoking the function named "sendGSMPDUSessionModificationCommand" which initiates the N1N2 message transfer.
7) N1N2 Message transfer function which calls the API communication client for communicating with AMF/gNB.

9.4. Other developments in SMF for enhancing NGAP message:

9.4.1 Optimization of QoS Rule Generation During PDU Session Establishment

1) In the *"BuildGSMPDUSessionEstablishmentAccept"* function within the "*gsm_build. go"* file, a block of code responsible for generating multiple QoS rules during PDU session establishment was commented out. This logic previously created QoS rules for each PCC rule present in the session context, resulting in the inclusion of multiple QoS rules in the NAS message.

2) However, for BTI radio (gNB), only a single default QoS rule is required during the initial PDU session establishment. The presence of multiple QoS rules led to inconsistencies and potential rejection of the NAS message by the gNB.
3) By commenting on this code block, the NAS message now includes only the default QoS rule, aligning with expected behavior for BTI gNB. As a result, PDU session establishment proceeds successfully, and the session setup is completed without errors.

9.4.2 Enhancement in NGAP Message Construction for QoS Flow Deletion

1) In the "*BuildPDUSessionResourceModifyRequestTransfer ()*" function within the ngap_build. go file, a conditional check was introduced:

   if len (qosFlowAddOrModifyRequestList.List) > 0 {}

1) This change ensures that the QosFlowAddOrModifyRequestList Information Element (IE) is included in the NGAP "*PDUSessionResourceModifyRequestTransfer"* message **only when** there are actual QoS flows to be added or modified. Previously, during DELETE operations, this IE was being constructed with an empty list, which led to errors in the SMF and prevented the N1N2Transfer request from reaching the AMF. As a result, the corresponding PDU Session Modification Command was not observed at the AMF or gNB.
2) By applying this conditional logic, the NGAP message now omits the QosFlowAddOrModifyRequestList IE when it is empty, thereby conforming to NGAP encoding rules and avoiding message rejection. Consequently, DELETE requests are successfully propagated to the AMF and gNB, and the PDU Session Modification Command packets are correctly captured—even in the absence of QoS flow additions or modifications.

9.4.3 Correction of PTI Assignment for Network-Initiated PDU Session Modification

1) In the "*BuildGSMPDUSessionModificationCommand ()"* function, the Procedure Transaction Identity (PTI) is now explicitly set to 0x00 to conform with 3GPP 24.501 [v17] specifications for network-initiated NAS Session Management procedures. Previously, the PTI was set up from *smContext.Pti*, which is appropriate only for UE-initiated procedures.

2) This change ensures that the NAS PDU Session Modification Command message is correctly interpreted by the UE, avoiding transaction mismatches and improving interoperability with compliant UE implementations.

   Note: Prior to this correction, using *smContext.Pti* for network-initiated procedures resulted in a PTI mismatch error during packet captures at the BTI gNB and AMF. This issue was resolved by explicitly setting PTI to 0x00, as required for network-originated NAS messages.

9.5 Final Outcome:

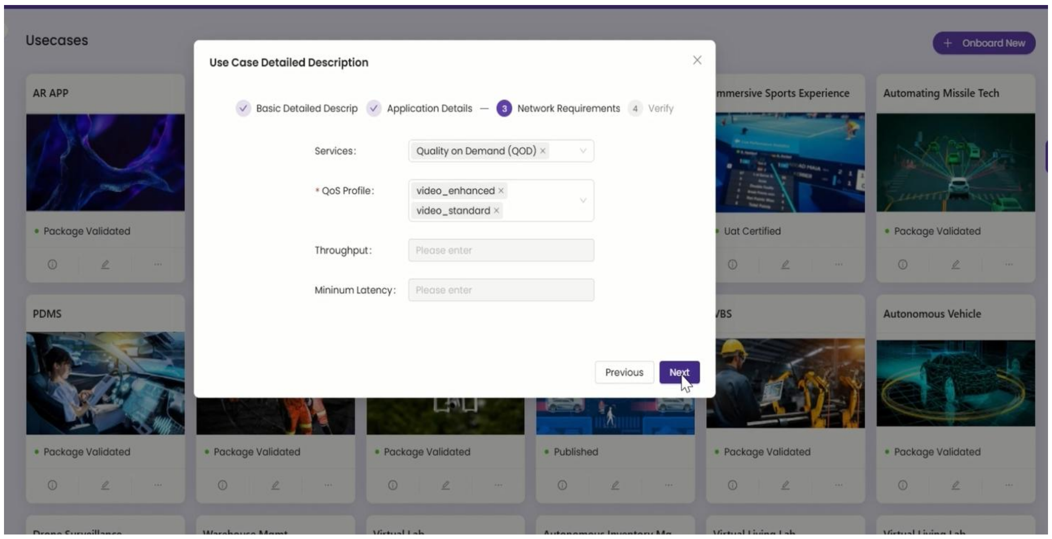
- After development, network initiated PDU session modification request is handled, SMF will build the "GSMPDUSessionModificationCommand" and sent it via "N1N2MessageTransfer", "PDUSessionResourceModifyRequestTransfer" fucntion invoked "N1N2MessageTransfer" where both N1 and N2 messages construct and sent to AMF and gNb.

- For DELETE request, PDUSessionResourceModify operations reach AMF and gNB successfully. The NGAP PDU includes the modification command without empty QoS flow IE, allowing gNB handling without IE validation failures.
- PDU establishment completes cleanly without the gNB rejecting or ignoring the NAS message due to unexpected multiple QoS Rules. Now only the default QoS rule (for the default QoS Flow with the session's default QFI) is present, as expected by the BTI gNB.

10. Results and demonstrations

Now let's see how QoD works in action. The validation of the entire workflow explained above can be performed through traffic testing by establishing a default QoS flow of 100 Mbps in the Free5GC GUI. The steps taken for testing workflow of 5g QOD service are as follows:

1. Onboard new use case of quality-on-demand in the IEAM.



Use Case Detailed Description

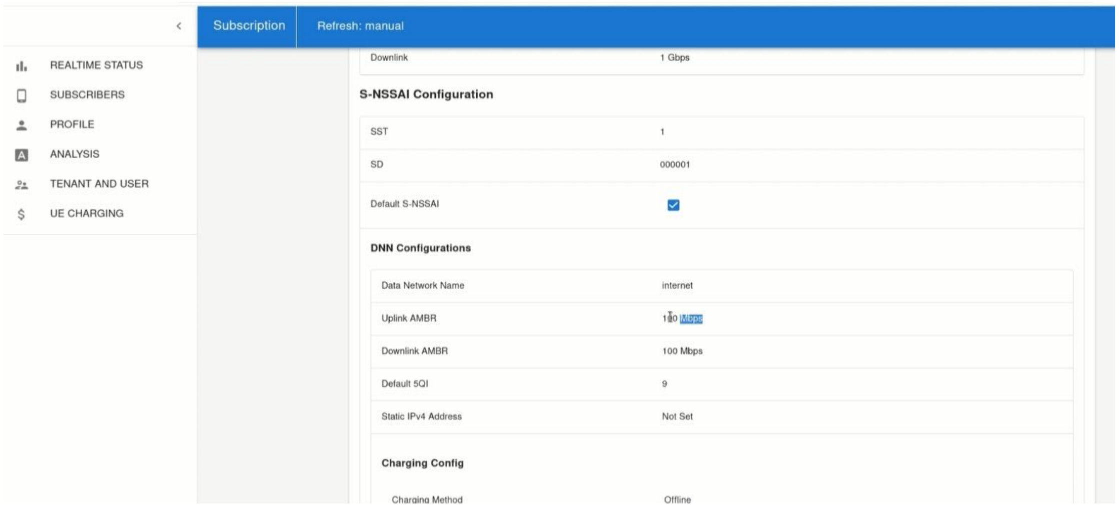
Services: Quality on Demand (QOD)

* QoS Profile: video_enhanced, video_standard

Throughput: Please enter

Mininum Latency: Please enter

2. Login to the free5gc Web UI, initially there is no subscriber adding the subscriber and set the uplink AMBR and downlink AMBR of 100 Mbps as default QoS.



Downlink	1 Gbps

S-NSSAI Configuration

SST	1
SD	000001
Default S-NSSAI	✓

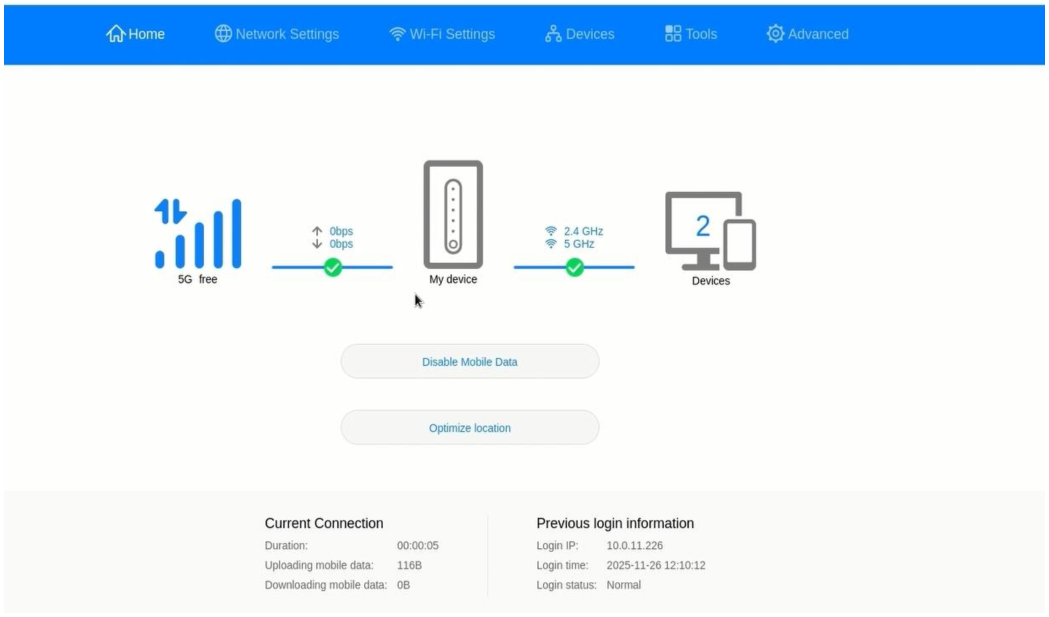
DNN Configurations

Data Network Name	internet
Uplink AMBR	100 Mbps
Downlink AMBR	100 Mbps
Default 5QI	9
Static IPv4 Address	Not Set

Charging Config

Charging Method	Offline

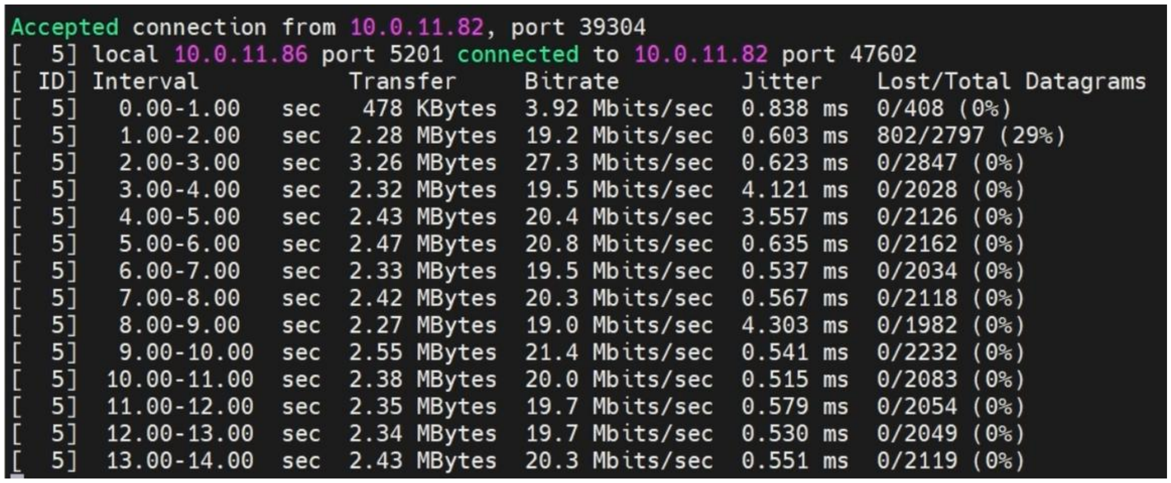
3. Connect the UE(CPE) to the Network and get Ue Ip address. Go back to the free5gc Web UI and check in real time status that UE is attached to free5gc.



Current Connection
Duration: 00:00:05
Uploading mobile data: 116B
Downloading mobile data: 0B

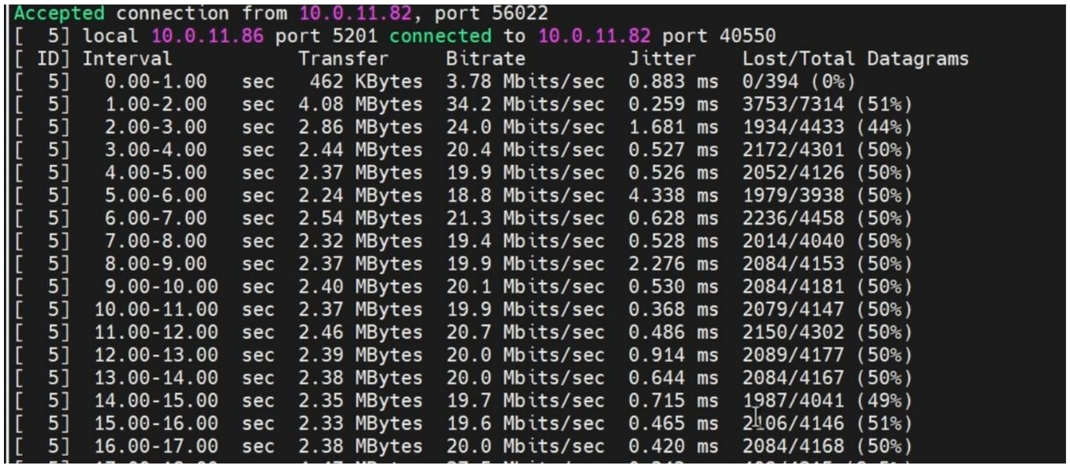
Previous login information
Login IP: 10.0.11.226
Login time: 2025-11-26 12:10:12
Login status: Normal

4. Once the PDU session established, a CAMARA POST request was sent via postman with a QoS profile "video_standard" specifying 20 Mbps. POST API:***{{apiRoot}}/quality-on-demand/v1/sessions.***
5. After that, Iperf command runs at client and the server side, passes the iperf throughput of 20 Mbps to verify the QoS of POST request, where no packet loss is observed, as shown below (20 Mbps with 0% loss).
6. Iperf client command (UE): "***iperf3 -c <UE IP address> -i1 -b40m -u -t1000 -l1200 -p 5201"***
   Iperf Server command (AF): "***iperf3 -s -i 1 -p 5201"***



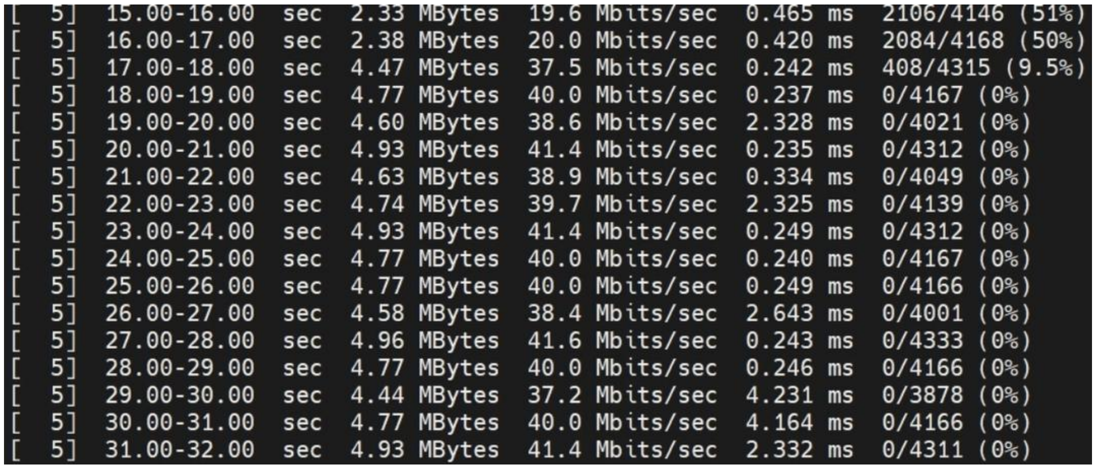
```
Accepted connection from 10.0.11.82, port 39304
[  5] local 10.0.11.86 port 5201 connected to 10.0.11.82 port 47602
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec   478 KBytes  3.92 Mbits/sec  0.838 ms  0/408 (0%)
[  5]   1.00-2.00   sec  2.28 MBytes  19.2 Mbits/sec  0.603 ms  802/2797 (29%)
[  5]   2.00-3.00   sec  3.26 MBytes  27.3 Mbits/sec  0.623 ms  0/2847 (0%)
[  5]   3.00-4.00   sec  2.32 MBytes  19.5 Mbits/sec  4.121 ms  0/2028 (0%)
[  5]   4.00-5.00   sec  2.43 MBytes  20.4 Mbits/sec  3.557 ms  0/2126 (0%)
[  5]   5.00-6.00   sec  2.47 MBytes  20.8 Mbits/sec  0.635 ms  0/2162 (0%)
[  5]   6.00-7.00   sec  2.33 MBytes  19.5 Mbits/sec  0.537 ms  0/2034 (0%)
[  5]   7.00-8.00   sec  2.42 MBytes  20.3 Mbits/sec  0.567 ms  0/2118 (0%)
[  5]   8.00-9.00   sec  2.27 MBytes  19.0 Mbits/sec  4.303 ms  0/1982 (0%)
[  5]   9.00-10.00  sec  2.55 MBytes  21.4 Mbits/sec  0.541 ms  0/2232 (0%)
[  5]  10.00-11.00  sec  2.38 MBytes  20.0 Mbits/sec  0.515 ms  0/2083 (0%)
[  5]  11.00-12.00  sec  2.35 MBytes  19.7 Mbits/sec  0.579 ms  0/2054 (0%)
[  5]  12.00-13.00  sec  2.34 MBytes  19.7 Mbits/sec  0.530 ms  0/2049 (0%)
[  5]  13.00-14.00  sec  2.43 MBytes  20.3 Mbits/sec  0.551 ms  0/2119 (0%)
```

7. Now, higher traffic requirement was tested using iPerf with a target throughput of 40 Mbps. However, only 20 Mbps was achieved, and a packet loss of approximately 50% was observed.



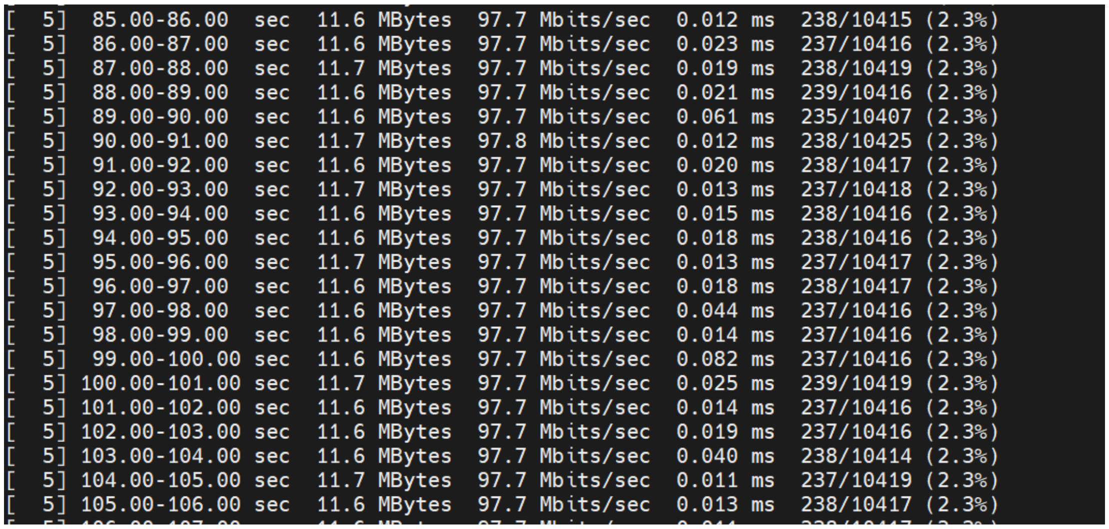
```
Accepted connection from 10.0.11.82, port 56022
[  5] local 10.0.11.86 port 5201 connected to 10.0.11.82 port 40550
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec   462 KBytes  3.78 Mbits/sec  0.883 ms  0/394 (0%)
[  5]   1.00-2.00   sec  4.08 MBytes  34.2 Mbits/sec  0.259 ms  3753/7314 (51%)
[  5]   2.00-3.00   sec  2.86 MBytes  24.0 Mbits/sec  1.681 ms  1934/4433 (44%)
[  5]   3.00-4.00   sec  2.44 MBytes  20.4 Mbits/sec  0.527 ms  2172/4301 (50%)
[  5]   4.00-5.00   sec  2.37 MBytes  19.9 Mbits/sec  0.526 ms  2052/4126 (50%)
[  5]   5.00-6.00   sec  2.24 MBytes  18.8 Mbits/sec  4.338 ms  1979/3938 (50%)
[  5]   6.00-7.00   sec  2.54 MBytes  21.3 Mbits/sec  0.628 ms  2236/4458 (50%)
[  5]   7.00-8.00   sec  2.32 MBytes  19.4 Mbits/sec  0.528 ms  2014/4040 (50%)
[  5]   8.00-9.00   sec  2.37 MBytes  19.9 Mbits/sec  2.276 ms  2084/4153 (50%)
[  5]   9.00-10.00  sec  2.40 MBytes  20.1 Mbits/sec  0.530 ms  2084/4181 (50%)
[  5]  10.00-11.00  sec  2.37 MBytes  19.9 Mbits/sec  0.368 ms  2079/4147 (50%)
[  5]  11.00-12.00  sec  2.46 MBytes  20.7 Mbits/sec  0.486 ms  2150/4302 (50%)
[  5]  12.00-13.00  sec  2.39 MBytes  20.0 Mbits/sec  0.914 ms  2089/4177 (50%)
[  5]  13.00-14.00  sec  2.38 MBytes  20.0 Mbits/sec  0.644 ms  2084/4167 (50%)
[  5]  14.00-15.00  sec  2.35 MBytes  19.7 Mbits/sec  0.715 ms  1987/4041 (49%)
[  5]  15.00-16.00  sec  2.33 MBytes  19.6 Mbits/sec  0.465 ms  2106/4146 (51%)
[  5]  16.00-17.00  sec  2.38 MBytes  20.0 Mbits/sec  0.420 ms  2084/4168 (50%)
```

8. Upgrading the app Session to “video_enhanced” of 40 Mbps by sending the CAMARA PATCH via postman. POST API***: {{ apiRoot}} /quality-on-demand/v1/sessions.*** Enhance QoS Profile to get better priority on Network and no packet loss is observed.



```
[  5]  15.00-16.00  sec  2.33 MBytes  19.6 Mbits/sec  0.465 ms  2106/4146 (51%)
[  5]  16.00-17.00  sec  2.38 MBytes  20.0 Mbits/sec  0.420 ms  2084/4168 (50%)
[  5]  17.00-18.00  sec  4.47 MBytes  37.5 Mbits/sec  0.242 ms  408/4315 (9.5%)
[  5]  18.00-19.00  sec  4.77 MBytes  40.0 Mbits/sec  0.237 ms  0/4167 (0%)
[  5]  19.00-20.00  sec  4.60 MBytes  38.6 Mbits/sec  2.328 ms  0/4021 (0%)
[  5]  20.00-21.00  sec  4.93 MBytes  41.4 Mbits/sec  0.235 ms  0/4312 (0%)
[  5]  21.00-22.00  sec  4.63 MBytes  38.9 Mbits/sec  0.334 ms  0/4049 (0%)
[  5]  22.00-23.00  sec  4.74 MBytes  39.7 Mbits/sec  2.325 ms  0/4139 (0%)
[  5]  23.00-24.00  sec  4.93 MBytes  41.4 Mbits/sec  0.249 ms  0/4312 (0%)
[  5]  24.00-25.00  sec  4.77 MBytes  40.0 Mbits/sec  0.240 ms  0/4167 (0%)
[  5]  25.00-26.00  sec  4.77 MBytes  40.0 Mbits/sec  0.249 ms  0/4166 (0%)
[  5]  26.00-27.00  sec  4.58 MBytes  38.4 Mbits/sec  2.643 ms  0/4001 (0%)
[  5]  27.00-28.00  sec  4.96 MBytes  41.6 Mbits/sec  0.243 ms  0/4333 (0%)
[  5]  28.00-29.00  sec  4.77 MBytes  40.0 Mbits/sec  0.246 ms  0/4166 (0%)
[  5]  29.00-30.00  sec  4.44 MBytes  37.2 Mbits/sec  4.231 ms  0/3878 (0%)
[  5]  30.00-31.00  sec  4.77 MBytes  40.0 Mbits/sec  4.164 ms  0/4166 (0%)
[  5]  31.00-32.00  sec  4.93 MBytes  41.4 Mbits/sec  2.332 ms  0/4311 (0%)
```

9. Finally, a CAMARA DELETE request was sent to delete the app session, which reverted the traffic to the default QoS of 100 Mbps (97.7 Mbps) as configured in the Web UI.



```
[  5]  85.00-86.00  sec  11.6 MBytes  97.7 Mbits/sec  0.012 ms  238/10415 (2.3%)
[  5]  86.00-87.00  sec  11.6 MBytes  97.7 Mbits/sec  0.023 ms  237/10416 (2.3%)
[  5]  87.00-88.00  sec  11.7 MBytes  97.7 Mbits/sec  0.019 ms  238/10419 (2.3%)
[  5]  88.00-89.00  sec  11.6 MBytes  97.7 Mbits/sec  0.021 ms  239/10416 (2.3%)
[  5]  89.00-90.00  sec  11.6 MBytes  97.7 Mbits/sec  0.061 ms  235/10407 (2.3%)
[  5]  90.00-91.00  sec  11.7 MBytes  97.8 Mbits/sec  0.012 ms  238/10425 (2.3%)
[  5]  91.00-92.00  sec  11.6 MBytes  97.7 Mbits/sec  0.020 ms  238/10417 (2.3%)
[  5]  92.00-93.00  sec  11.7 MBytes  97.7 Mbits/sec  0.013 ms  237/10418 (2.3%)
[  5]  93.00-94.00  sec  11.6 MBytes  97.7 Mbits/sec  0.015 ms  238/10416 (2.3%)
[  5]  94.00-95.00  sec  11.6 MBytes  97.7 Mbits/sec  0.018 ms  238/10416 (2.3%)
[  5]  95.00-96.00  sec  11.7 MBytes  97.7 Mbits/sec  0.013 ms  237/10417 (2.3%)
[  5]  96.00-97.00  sec  11.6 MBytes  97.7 Mbits/sec  0.018 ms  238/10417 (2.3%)
[  5]  97.00-98.00  sec  11.6 MBytes  97.7 Mbits/sec  0.044 ms  237/10416 (2.3%)
[  5]  98.00-99.00  sec  11.6 MBytes  97.7 Mbits/sec  0.014 ms  237/10416 (2.3%)
[  5]  99.00-100.00 sec  11.6 MBytes  97.7 Mbits/sec  0.082 ms  237/10416 (2.3%)
[  5] 100.00-101.00 sec  11.7 MBytes  97.7 Mbits/sec  0.025 ms  239/10419 (2.3%)
[  5] 101.00-102.00 sec  11.6 MBytes  97.7 Mbits/sec  0.014 ms  237/10416 (2.3%)
[  5] 102.00-103.00 sec  11.6 MBytes  97.7 Mbits/sec  0.019 ms  237/10416 (2.3%)
[  5] 103.00-104.00 sec  11.6 MBytes  97.7 Mbits/sec  0.040 ms  238/10414 (2.3%)
[  5] 104.00-105.00 sec  11.7 MBytes  97.7 Mbits/sec  0.011 ms  237/10419 (2.3%)
[  5] 105.00-106.00 sec  11.6 MBytes  97.7 Mbits/sec  0.013 ms  238/10417 (2.3%)
```