Open5GS API Documentation

Release v1.0.3

Ashwin Sathishkumar

Dec 20, 2024

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Prerequisites

Before installing the Open5GS API, ensure your system meets the following requirements:

- Python 3.9 or higher
- Docker 27.2.0 or higher
- sudo privileges
- Linux-based operating system (Ubuntu 22.04 LTS or higher recommended)

1.2 System Dependencies

First, unzip the file `hso-open5gs-ueransim-ueransim-new.zip` and open it in the integrated terminal. Then, install the required system packages:

```
sudo apt update
sudo apt install python3-venv docker.io
```

1.3 Building Required Images

1. Build the Open5GS base image:

   ```
   cd open5gs
   ./build-image.sh
   ```

2. Build the UERANSIM image:

   ```
   cd ueransim
   ./build_image.sh
   ```

1.4 Virtual Environment Setup

1. Install Python virtual environment:

   ```
   sudo apt install python3-venv
   ```

2. Create and activate a virtual environment:

```
python3 -m venv myvenv
source myvenv/bin/activate
```

1.5 Required Packages

Install the required Python packages within the virtual environment:

```
pip install flask ruamel.yaml psutil
```

1.6 Docker Configuration

1. Add your user to the docker group:

   ```
   sudo usermod -aG docker $USER
   ```

2. Restart the Docker service:

   ```
   sudo systemctl restart docker
   ```

> **Note**
>
> You may need to log out and log back in for the group changes to take effect.

1.7 Open5GS Package Installation

The open5gsapi package can be installed using:

```
pip install open5gsapi
```

Verify the installation by running the CLI command:

```
open5gsapi --version
```

1.8 System Requirements

Recommended specfications for running the project are:

- CPU: 4 cores
- RAM: 8 GB

> **Warning**
>
> Supported on Ubuntu 22.04 or higher. Windows and macOS are not officially supported.

CHAPTER

TWO

QUICK START

2.1 Setting File Paths

After package installation, set the path to your pcf.yaml and .env files relative to the directory where your application files are created.

```
from open5gsapi import open5gs

# Sets the PCF configuration file path
open5gs.set_config_path('/path/to/pcf.yaml')
```

```
# Sets the UERANSIM configuration file path
open5gs.set_env_path('/path/to/.env')
```

If the configuration files are edited manually after loading:

```
# Explicitly reload the configurations
open5gs.reload_config()  # Reload PCF configuration
open5gs.reload_env()     # Reload environment configuration
```

2.2 Configuration Files Overview

2.2.1 PCF Policy Configuration

The default `pcf.yaml` configuration file in Open5GS looks like this:

```
policy:
  - plmn_id:
      mcc: 999
      mnc: 70
    slice:
      - sst: 1  # 1,2,3,4
        default_indicator: true
        session:
          - name: internet
            type: 3  # 1:IPv4, 2:IPv6, 3:IPv4v6
            ambr:
              downlink:
                value: 1
                unit: 3  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
```

(continues on next page)

(continued from previous page)

```
            uplink:
              value: 1
              unit: 3
          qos:
            index: 9  # 1, 2, 3, 4, 65, 66, 67, 75, 71, 72, 73, 74, 76, 5, 6, 7, 8, 9,␣
↪69, 70, 79, 80, 82, 83, 84, 85, 86
            arp:
              priority_level: 8  # 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
              pre_emption_vulnerability: 1  # 1: Disabled, 2:Enabled
              pre_emption_capability: 1  # 1: Disabled, 2:Enabled
        - name: ims
          type: 3  # 1:IPv4, 2:IPv6, 3:IPv4v6
          ambr:
            downlink:
              value: 1
              unit: 3  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
            uplink:
              value: 1
              unit: 3  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
          qos:
            index: 5  # 1, 2, 3, 4, 65, 66, 67, 75, 71, 72, 73, 74, 76, 5, 6, 7, 8, 9,␣
↪69, 70, 79, 80, 82, 83, 84, 85, 86
            arp:
              priority_level: 1  # 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
              pre_emption_vulnerability: 1  # 1: Disabled, 2:Enabled
              pre_emption_capability: 1  # 1: Disabled, 2:Enabled
          pcc_rule:
            - qos:
                index: 1  # 1, 2, 3, 4, 65, 66, 67, 75, 71, 72, 73, 74, 76, 5, 6, 7, 8,
↪ 9, 69, 70, 79, 80, 82, 83, 84, 85, 86
                arp:
                  priority_level: 1  # 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,␣
↪15
                  pre_emption_vulnerability: 1   # 1: Disabled, 2:Enabled
                  pre_emption_capability: 1   # 1: Disabled, 2:Enabled
                mbr:
                  downlink:
                    value: 82
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
                  uplink:
                    value: 82
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
                gbr:
                  downlink:
                    value: 82
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
                  uplink:
                    value: 82
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
              flow:
                - direction: 2
                  description: "permit out icmp from any to assigned"
```

(continues on next page)

(continued from previous page)

```
                - direction: 1
                  description: "permit out icmp from any to assigned"
                - direction: 2
                  description: "permit out udp from 10.200.136.98/32 23455 to assigned␣
↪1-65535"
                - direction: 1
                  description: "permit out udp from 10.200.136.98/32 1-65535 to␣
↪assigned 50021"
            - qos:
                index: 2  # 1, 2, 3, 4, 65, 66, 67, 75, 71, 72, 73, 74, 76, 5, 6, 7, 8,
↪ 9, 69, 70, 79, 80, 82, 83, 84, 85, 86
                arp:
                  priority_level: 4  # 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,␣
↪15
                  pre_emption_vulnerability: 2   # 1: Disabled, 2:Enabled
                  pre_emption_capability: 2   # 1: Disabled, 2:Enabled
                mbr:
                  downlink:
                    value: 802
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
                  uplink:
                    value: 802
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
                gbr:
                  downlink:
                    value: 802
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
                  uplink:
                    value: 802
                    unit: 1  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
  - plmn_id:
      mcc: 001
      mnc: 01
    slice:
      - sst: 1  # 1,2,3,4
        sd: 000001
        default_indicator: true
        session:
          - name: internet
            type: 3  # 1:IPv4, 2:IPv6, 3:IPv4v6
            ambr:
              downlink:
                value: 1
                unit: 3  # 0:bps, 1:Kbps, 2:Mbps, 3:Gbps, 4:Tbps
              uplink:
                value: 1
                unit: 3
            qos:
              index: 9  # 1, 2, 3, 4, 65, 66, 67, 75, 71, 72, 73, 74, 76, 5, 6, 7, 8, 9,␣
↪69, 70, 79, 80, 82, 83, 84, 85, 86
              arp:
                priority_level: 8  # 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
```

(continues on next page)

(continued from previous page)

```
                pre_emption_vulnerability: 1  # 1: Disabled, 2:Enabled
                pre_emption_capability: 1  # 1: Disabled, 2:Enabled
```

Of these configurations, our API allows modification of:

1. Session Level Parameters:
   - `name`: Session identifier
   - `type`: Session type (1:IPv4, 2:IPv6, 3:IPv4v6)
   - `ambr`: Aggregate Maximum Bit Rate
     - downlink/uplink value and unit (0:bps - 4:Tbps)
   - `qos`: Quality of Service parameters
     - index (1-9, 65-86)
     - `arp`: Allocation Retention Priority
       * priority_level (1-15)
       * pre_emption_vulnerability (1:Disabled, 2:Enabled)
       * pre_emption_capability (1:Disabled, 2:Enabled)
2. PCC Rule Parameters:
   - `qos`: Rule-specific QoS parameters (same as session QoS)
   - `mbr/gbr`: Maximum/Guaranteed Bit Rate
     - downlink/uplink value and unit (0:bps - 4:Tbps)
   - `flow`: Traffic flow parameters
     - direction (1:Uplink, 2:Downlink)
     - description (flow matching rules)

2.2.2 Environment Configuration

The default `.env` file looks like this:

```
BASE=..
OPEN5GS=${BASE}/open5gs/open5gs
UERANSIM=${BASE}/ueransim/UERANSIM
CONFIG=.
NUM_UES=1
```

Of these configurations, our API allows modification of:

- `NUM_UES`: Number of UE instances (must be positive integer)

CHAPTER

THREE

CORE FEATURES

3.1 UE Configuration

3.1.1 Setting Number of UEs

```
# Get current number of UEs
current_ues = open5gs.get_num_ues()
print(f"Current number of UEs: {current_ues}")

# Set new number of UEs (must be done before update_config())
open5gs.set_num_ues(3)
```

3.2 Network Traffic Analysis

3.2.1 Launching Wireshark

```
# Launch Wireshark with default GTP filtering
open5gs.launch_gtp_wireshark()

# Eg., Launch with custom filters
open5gs.launch_wireshark(
    ip_address="10.10.0.1",
    display_filter="ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16",
    capture_filter="port 53"
)
```

3.3 UE and UPF Operations

3.3.1 Getting API URLs

```
# Get UE API URL
UE_API_URL = open5gs.ue("send")
# Result: "http://10.10.0.132:8080/send"

# Get UPF API URL
UPF_API_URL = open5gs.upf("receive/sensor")
# Result: "http://10.10.0.112:8081/receive/sensor"
```

3.3.2 Sending and Receiving Data

The API supports sending both JSON data and binary data (like video frames):

```
# Basic data sending (e.g., JSON data)
data = {
    "sensor_id": 1,
    "temperature": 25.5,
    "humidity": 60
}
response = open5gs.send_data(UE_API_URL, data)

# For mMTC with multiple UEs
base_port = 8080
for ue_index in range(num_ues):
    port_offset = ue_index  # Each UE uses a different port (8080 + offset)
    sensor_data = {
        "sensor_id": ue_index,
        "temperature": 25.5
    }
    response = open5gs.send_data(UE_API_URL, sensor_data, port_offset=ue_index)

# For video streaming (binary data)
ret, buffer = cv2.imencode('.jpg', frame, encode_params)
frame_data = buffer.tobytes()  # Binary data automatically detected
response = open5gs.send_data(UE_API_URL, frame_data)

# Receiving data from UPF
# For JSON data (e.g., sensor data)
received_data = open5gs.receive_data(UPF_API_URL)
if received_data:
    print(f"Received sensor data: {received_data}")

# For video frames
frame_data = open5gs.receive_data(UPF_STREAM_URL)
if frame_data:  # Binary data automatically detected by content-type
    return Response(frame_data, mimetype='image/jpeg')
```

The `send_data` and `receive_data` methods automatically:

- Detect data type (JSON vs binary)
- Handle appropriate content types
- Collect network performance metrics
- Support port offsets for multiple UEs
- Handle errors with appropriate exceptions

3.4 Network Performance Metrics

The API automatically collects and provides network performance metrics based on the type of data transfer:

3.4.1 Video Streaming Metrics

For binary data like video frames, use `get_metrics()`:

```
# Get video streaming metrics
metrics = open5gs.get_metrics()  # For binary data/video frames
print(f"Throughput: {metrics['throughput']['total_mbps']} Mbps")
print(f"Average latency: {metrics['latency']['avg_ms']} ms")
```

Video metrics structure:

```
{
    "throughput": {
        "total_mbps": 2.5,        # Total throughput in Mbps
    },
    "latency": {
        "min_ms": 10.5,           # Minimum latency
        "max_ms": 50.2,           # Maximum latency
        "avg_ms": 25.7,           # Average latency
        "jitter_ms": 5.2          # Latency variation
    },
    "packets": {
        "total": 1000,            # Total packets sent
        "avg_size_bytes": 1024    # Average packet size
    }
}
```

Additional frame metrics for video streaming:

```
frame_metrics = open5gs.get_frame_metrics()
if frame_metrics:
    print(f"Current FPS: {frame_metrics['frame_rate']['current_fps']}")
    print(f"Average frame size: {frame_metrics['frame_size']['avg_bytes']/1024:.1f} KB")
```

Frame metrics structure:

```
{
    "frame_metrics": {
        "total_frames": 300,
        "frame_rate": {
            "current_fps": 30.0,
            "frame_time_ms": 33.3,
            "frame_time_variation_ms": 1.2
        },
        "frame_size": {
            "avg_bytes": 51200,
            "max_bytes": 65536,
            "min_bytes": 32768
        },
        "latency": {
            "min_ms": 15.0,
            "max_ms": 45.0,
            "avg_ms": 25.0,
            "jitter_ms": 5.0
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

```
# Reset video metrics if needed
open5gs.reset_metrics()
```

3.4.2 Sensor Data Metrics

For JSON data like sensor readings, use `get_sensor_metrics()`:

```
# Get sensor metrics
sensor_metrics = open5gs.get_sensor_metrics()  # For JSON data/sensor readings
print(f"Reading rate: {sensor_metrics['sensor_metrics']['reading_rate']['current_rps']}␣
↪readings/sec")
print(f"Lost readings: {sensor_metrics['sensor_metrics']['reading_rate']['readings_lost
↪']}")
```

Sensor metrics structure:

```
{
    "throughput": {
        "total_mbps": 1.5        # Total throughput in Mbps
    },
    "latency": {
        "min_ms": 5.0,           # Minimum latency
        "max_ms": 25.0,          # Maximum latency
        "avg_ms": 15.0,          # Average latency
        "jitter_ms": 2.5         # Latency variation
    },
    "sensor_metrics": {
        "reading_rate": {
            "current_rps": 10.0,           # Readings per second
            "reading_interval_ms": 100.0,   # Average interval between readings
            "total_readings": 1000,        # Total readings received
            "readings_lost": 5             # Number of lost readings
        },
        "by_sensor": {
            "1": {                         # Per-sensor statistics
                "total_readings": 500,
                "readings_lost": 2,
                "latest_value": {"temperature": 25.5},
                "min_value": {"temperature": 20.0},
                "max_value": {"temperature": 30.0},
                "avg_value": 24.5
            }
        }
    }
}
```

```
# Reset sensor metrics if needed
open5gs.reset_sensor_metrics()
```

The metrics collection works automatically when using `send_data()` and `receive_data()`. The API detects the data type and uses the appropriate metrics collector:

- Binary data → Video metrics
- JSON data with 'sensor_id' → Sensor metrics

Common utility methods for both types:

```
throughput = open5gs.get_throughput()           # Current throughput in Mbps
latency_stats = open5gs.get_latency_stats()     # Latency statistics
```

3.5 PCF Configuration Management

3.5.1 Listing and Viewing Sessions

```
# List all sessions
sessions = open5gs.list_sessions()
print("Current sessions:", sessions)

# Get details of a specific session
session_name = "video-streaming"
session_details = open5gs.get_session_details(session_name)
print(f"Details of session '{session_name}':", session_details)
```

3.5.2 Modifying Session Parameters

```
# Modify session parameters
session = open5gs.policy.session('video-streaming')
session.ambr.downlink(value=10000000, unit=1)
session.ambr.uplink(value=20000000, unit=1)
session.qos(index=5)
session.arp(priority_level=7, pre_emption_vulnerability=2, pre_emption_capability=1)

# Modify PCC rule parameters
session.pcc_rule[0].qos(index=3)
session.pcc_rule[0].mbr.downlink(value=2000000, unit=1)
session.pcc_rule[0].gbr.uplink(value=1000000, unit=1)
session.pcc_rule[0].add_flow(direction=2, description="permit out ip from any to assigned
↪")
```

3.5.3 Managing Sessions

```
# Add a new session
new_session = open5gs.policy.add_session('new-session-name')
new_session.ambr.downlink(value=5000000, unit=1)
new_session.ambr.uplink(value=1000000, unit=1)

# Remove a session
open5gs.policy.remove_session('session-to-remove')

# Rename a session
open5gs.rename_session('old-session-name', 'new-session-name')
```

3.5.4 Updating Configuration

After making changes to the configuration, you need to update the system:

```
# Update PCF YAML file
open5gs.update_pcf()

# Tear down and redeploy containers with new configuration
open5gs.update_config()

# Run initialization scripts and start background processes
open5gs.run_background_nodes()
```

You can check the status of these operations:

```
# Check completion status
if open5gs.is_update_pcf_complete():
    print("PCF update complete")

if open5gs.is_update_config_complete():
    print("Configuration update complete")

# Check background nodes status
if open5gs.is_run_background_nodes_complete():
    print("Background nodes are running")
else:
    status = open5gs.get_background_process_status()
    print(f"Background process status: {status}")
```

CHAPTER

FOUR

API REFERENCE

4.1 Program File Management

1. `open5gs.set_config_path(path: str)`: Set the PCF configuration file path
2. `open5gs.set_env_path(path: str)`: Set the environment file path
3. `open5gs.reload_config()`: Reload the PCF configuration
4. `open5gs.reload_env()`: Reload the environment configuration
5. `open5gs.get_num_ues() -> int`: Get current number of UEs
6. `open5gs.set_num_ues(num_ues: int)`: Set number of UEs

See also

Internal file management functions are detailed in *Dev - File Management*

4.2 UE and UPF Operations

1. `open5gs.ue(endpoint: str) -> str`: Get the UE API URL
2. `open5gs.upf(endpoint: str) -> str`: Get the UPF API URL
3. `open5gs.send_data(endpoint: str, data: Any, port_offset: int = 0) -> Dict[str, Any]`: Send data to specified endpoint

- `endpoint`: API endpoint URL
- `data`: Data to send (JSON or binary)
- `port_offset`: Port offset for multiple UEs (default=0)

4. `open5gs.receive_data(endpoint: str) -> Any`: Receive data from specified endpoint

See also

Internal API endpoint management functions are detailed in *Dev - API Endpoint Management*

4.3 Network Performance Monitoring

1. `open5gs.get_metrics() -> Dict[str, Any]`: Get comprehensive network performance metrics
2. `open5gs.get_frame_metrics() -> Optional[Dict[str, Any]]`: Get frame-specific metrics for video streaming
3. `open5gs.get_throughput() -> float`: Get current throughput in Mbps
4. `open5gs.get_latency_stats() -> Dict[str, float]`: Get latency statistics
5. `open5gs.reset_metrics()`: Reset all network metrics

See also

Internal network and sensor metrics collection functions are detailed in *Dev - Metrics Management*

4.4 Network Analysis

1. `open5gs.launch_wireshark(ip_address: str, display_filter: str, capture_filter: str) -> bool`: Launch Wireshark with custom filters
2. `open5gs.launch_gtp_wireshark() -> bool`: Launch Wireshark with GTP filtering

See also

Internal Wireshark management functions are detailed in *Dev - Wireshark Interface Management*

4.5 PCF Configuration Management

1. `open5gs.list_sessions() -> List[str]`: Get a list of all session names
2. `open5gs.get_session_details(name: str) -> Dict[str, Any]`: Get details of a specific session
3. `open5gs.rename_session(old_name: str, new_name: str)`: Rename a session
4. `open5gs.policy.session(name: str) -> Session`: Get or create a session
5. `open5gs.policy.add_session(name: str) -> Session`: Add a new session
6. `open5gs.policy.remove_session(name: str)`: Remove a session

4.5.1 Session Methods

1. `session.ambr.downlink(value: int, unit: int)`: Set downlink AMBR
2. `session.ambr.uplink(value: int, unit: int)`: Set uplink AMBR
3. `session.qos(index: int)`: Set QoS index
4. `session.arp(priority_level: int, pre_emption_vulnerability: int, pre_emption_capability: int)`: Set ARP parameters

4.5.2 PCC Rule Methods

1. `session.pcc_rule[index].qos(index: int)`: Set QoS index for a PCC rule
2. `session.pcc_rule[index].mbr.downlink(value: int, unit: int)`: Set downlink MBR for a PCC rule
3. `session.pcc_rule[index].mbr.uplink(value: int, unit: int)`: Set uplink MBR for a PCC rule
4. `session.pcc_rule[index].gbr.downlink(value: int, unit: int)`: Set downlink GBR for a PCC rule
5. `session.pcc_rule[index].gbr.uplink(value: int, unit: int)`: Set uplink GBR for a PCC rule
6. `session.pcc_rule[index].add_flow(direction: int, description: str)`: Add a flow to a PCC rule

4.6 Configuration Update

1. `open5gs.update_pcf()`: Update the PCF YAML file
2. `open5gs.update_config()`: Tear down and redeploy containers with new configuration
3. `open5gs.run_background_nodes()`: Run initialization scripts and start background processes in UE and UPF containers

4.7 Background Process Management

1. `open5gs.is_run_background_nodes_complete() -> bool`: Check if background nodes are running
2. `open5gs.get_background_process_status() -> Dict[str, Any]`: Get detailed status of background processes

See also

Internal process monitoring and verification functions are detailed in *Dev - Process Management*

CHAPTER

FIVE

ERROR HANDLING

5.1 ConfigurationError

Raised when there are issues related to the overall configuration of the Open5GS system. It may occur in the following scenarios:

- The configuration file (pcf.yaml) cannot be found or read.
- The environment file (.env) cannot be found or read.
- There are structural problems with the configuration files.
- Unable to initialize or access necessary components of the Open5GS system.
- Attempting to access or modify a non-existent session.
- Failing to restart the PCF service.
- Attempting to modify NUM_UES after update_config() has been called.
- Wireshark is not installed when attempting to launch packet capture.

Example usage:

```
try:
    open5gs.set_config_path('/path/to/pcf.yaml')
    open5gs.set_env_path('/path/to/.env')
except ConfigurationError as e:
    print(f"Configuration error: {e}")
    # Handle the error (e.g., exit the program or use a default configuration)
```

5.2 ValidationError

Raised when the input values provided for specific configuration parameters are invalid or out of the allowed range. It typically occurs when:

- Setting an invalid QoS index.
- Providing an out-of-range value for AMBR.
- Using an incorrect value for ARP parameters.
- Setting an invalid session type.
- Adding an invalid flow direction in PCC rules.
- Setting an invalid number of UEs (must be positive integer).
- Providing invalid Wireshark filter expressions.

Example usage:

```
try:
    session = open5gs.policy.session('internet')
    session.qos(index=100)  # 100 is not a valid QoS index
    open5gs.set_num_ues(0)  # 0 is not a valid number of UEs
except ValidationError as e:
    print(f"Validation error: {e}")
    # Handle the error (e.g., use a default value or prompt the user for valid input)
```

5.3 CommunicationError

Raised when there are issues communicating with the UE or UPF components. It may occur when:

- Sending data to the UE API fails.
- Receiving data from the UPF API fails.
- Unable to establish connection with Wireshark for packet capture.
- Background nodes (UE/UPF) fail to start or become unresponsive.
- TUN interfaces are not properly set up for UE communication.

Example usage 1:

```
try:
    response = open5gs.send_data(UE_API_URL, data)
    if not open5gs.launch_gtp_wireshark():
        raise CommunicationError("Failed to launch Wireshark")
except CommunicationError as e:
    print(f"Communication error: {e}")
    # Handle the error (e.g., retry the operation or log the failure)
```

Example usage 2:

```
# Example of sending data with port offset
try:
    for ue_index in range(num_ues):
        data = {"sensor_id": ue_index, "value": 25.5}
        response = open5gs.send_data(UE_API_URL, data, port_offset=ue_index)
except CommunicationError as e:
    print(f"Communication error: {e}")
    # Handle error (e.g., check if UE interface is ready)
```

You can also check the detailed status of background processes when encountering errors:

```
try:
    open5gs.run_background_nodes()
    if not open5gs.is_run_background_nodes_complete():
        status = open5gs.get_background_process_status()
        if status['error_message']:
            raise CommunicationError(f"Background process error: {status['error_message
↪']}")
except CommunicationError as e:
```

(continues on next page)

(continued from previous page)

```
    print(f"Communication error: {e}")
    # Handle the error (e.g., check system requirements or restart services)
```

5.4 Monitoring Errors

Example of monitoring network performance:

```
try:
    metrics = open5gs.get_metrics()
    if metrics['frame_metrics']:  # Frame data detected
        print(f"Streaming performance:")
        print(f"FPS: {metrics['frame_metrics']['frame_rate']['current_fps']}")
        print(f"Frame latency: {metrics['frame_metrics']['latency']['avg_ms']} ms")
    else:  # Regular data
        print(f"Network performance:")
        print(f"Throughput: {metrics['throughput']['total_mbps']} Mbps")
        print(f"Latency: {metrics['latency']['avg_ms']} ms")
except Exception as e:
    print(f"Error getting metrics: {e}")
    # Handle error (e.g., log error, use default values)
```

Metrics are collected with microsecond precision using monotonic time to ensure accurate measurements.

CHAPTER

SIX

USER TROUBLESHOOTING

This chapter covers common issues users may encounter and their solutions.

6.1 Docker Related Issues

6.1.1 Docker Group Permissions

If you encounter permission issues with Docker:

```
# Check if user is in docker group
groups  # Verify 'docker' is listed
newgrp docker  # Apply group changes without logout
```

6.1.2 MongoDB AVX Support

If MongoDB fails to start with CPU capability errors:

1. This occurs because MongoDB versions >4.4 require AVX instruction set support
2. Solution: Modify docker-compose.yaml to use an older MongoDB version:

```
mongodb:
  container_name: mongodb
  image: mongo:4.4  # Use version 4.4 or lower
  expose:
    - 27017
  networks:
    - corenet
  volumes:
    - mongodbdata:/data/db
```

6.2 Virtual Environment Issues

6.2.1 Command Not Found Errors

If you encounter "Command not found" errors:

```
# Ensure virtual environment is activated
source myvenv/bin/activate
```

6.2.2 Package Installation Issues

If pip installations fail:

1. Verify virtual environment is activated
2. Update pip:

```
pip install --upgrade pip
```

3. If issues persist, try clearing pip cache:

```
pip cache purge
```

6.3 Network Configuration Failures

6.3.1 Understanding the Network Setup Process

The API's network configuration involves three main phases:

1. Configuration file updates (pcf.yaml and .env)
2. Docker container deployment
3. Gateway code initialization (auto-ue-api.py and upf-api.py)

See also

Implementation details of the gatway codes avaiable in *Gateway Implementation*

6.3.2 TUN Interface Verification

The critical step is verifying TUN interface creation after container deployment:

1. Check UE container logs for TUN interface status:

```
# From deployment directory
docker logs -f ue
```

2. Look for successful setup message:

   "Connection setup for PDU session[1] is successful, TUN interface[uesimtunX, 10.45.0.Y] is up"

Note

- UE TUN interfaces start from 10.45.0.2
- UPF uses 10.45.0.1
- Current hardware limitations support up to ~50 UEs

6.3.3 Manual Container Management

For debugging network issues, these Docker commands are available (from `deployment` directory):

1. Container Access:

```
# Access UE container
docker exec -it ue bash

# Access UPF container
docker exec -it upf bash
```

2. Log Inspection:

```
# View UE logs
docker logs -f ue

# View UPF logs
docker logs -f upf
```

3. Deployment Management:

```
# Tear down deployment
docker compose down -t 1 -v

# Deploy containers
docker compose up -d
```

6.3.4 Gateway Code Location

When accessing containers, gateway codes are located at:

- UPF container: `cd src/upf` for upf-api.py
- UE container: Root directory contains auto-ue-api.py

Important

Gateway codes expose endpoints at:

- UE: 10.10.0.132:8080 (for sending data)
- UPF: 10.10.0.112:8081 (for receiving data)

These endpoints must be accessible before application interactions can begin.

6.3.5 Network Verification Steps

Checking Interface IPs

To verify UE interface IPs:

```
# Access UE container
docker exec -it ue bash

# List all interfaces
ifconfig

# Look for ``uesimtunX`` interfaces (each shows assigned IP in 10.45.0.x range)
```

To verify UPF interface IP:

```
# Access UPF container
docker exec -it upf bash

# List all interfaces
ifconfig

# Look for ``ogstun`` interface IP
```

Verifying 5G Tunnel Traffic

To monitor data transmission through the 5G tunnel:

1. Identify interface with 10.10.0.1:

   ```
   # From host machine (outside containers)
   ifconfig
   ```

2. Launch Wireshark:

   ```
   sudo wireshark
   ```

3. Monitor Traffic:
   - Select identified interface
   - Apply filter: "gtp"
   - Verify successful transmission by observing:
     - Source IP (uesimtunX IP)
     - Destination IP (ogstun IP)
     - GTP-encapsulated UDP packets
     - Original message content in packet details

> **Tip**
>
> Successful transmission shows your message encapsulated in GTP<UDP> from uesimtunX IP to ogstun IP, confirming proper 5G tunnel operation.

6.4 Pip Command Errors

If you encounter issues with `pip` commands in your virtual environment, such as the error *ModuleNotFoundError: No module named 'pip'*, follow these steps to resolve the problem:

Firstly, ensure the virtual environment is active

```
source myenv/bin/activate
# If the virtual environment is active, the terminal prompt should include the name of␣
↪the environment, e.g., `(myenv)`.
```

If `pip` is missing, you can manually reinstall it using the following steps:

1. Download the *get-pip.py* script:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

2. Run the script with the Python interpreter from your virtual environment:

```
myenv/bin/python3 get-pip.py
```

If the above steps fail, your system Python installation may be incomplete. Reinstall Python to ensure all components are present:

```
sudo apt update # Update your package manager
sudo apt install --reinstall python3 python3-venv python3-pip # Reinstall␣
↪Python
```

If the issue persists, delete and recreate the virtual environment:

1. Remove the problematic virtual environment:

```
rm -rf myenv
```

2. Create a new virtual environment:

```
python3 -m venv myenv
```

3. Activate the new virtual environment:

```
source myenv/bin/activate
```

4. Upgrade *pip*:

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip

# Ensure that it is properly installed
pip --version
```

CHAPTER

SEVEN

DEVELOPER REFERENCE

7.1 Dev - Wireshark Interface Management

Internal methods supporting *Network Analysis*:

```
_check_wireshark_installed() -> bool
"""Checks if Wireshark executable exists in /usr/bin/wireshark or /usr/local/bin/
↪wireshark."""

_get_interface_name(ip_address: str) -> Optional[str]
"""Uses 'ip addr show' command to find network interface name matching given IP address."
↪""

_get_display_environment() -> dict
"""Sets up X11 display environment variables required for Wireshark GUI, including␣
↪DISPLAY, XAUTHORITY, and display permissions."""
```

7.2 Dev - File Management

Internal methods supporting *Program File Management*:

```
_ensure_config_loaded(force_reload: bool = False)
"""Verifies PCF YAML exists, checks if file was modified since last load, reloads if␣
↪needed or forced."""

_ensure_env_loaded(force_reload: bool = False)
"""Verifies .env file exists, checks modification time, reloads NUM_UES and other␣
↪environment variables if modified."""

_read_config() -> Dict[str, Any]
"""Reads PCF YAML file and parses configuration using ruamel.yaml preserving quotes and␣
↪formatting."""

_read_env_file() -> Dict[str, str]
"""Parses .env file line by line, handling key=value pairs and ignoring comments."""

_write_env_file()
"""Updates .env file with current configuration including NUM_UES value."""
```

7.3 Dev - API Endpoint Management

Internal methods supporting *UE and UPF Operations*:

```
_check_api_endpoint(url: str, timeout: int = 1) -> bool
"""Attempts HTTP GET request to endpoint with 1-second timeout, verifies 200/204␣
↪response."""

_check_ue_interfaces() -> Tuple[bool, Optional[str]]
"""Verifies uesimtun interfaces exist in UE container and APIs respond on ports 8080+."""

_check_upf_api() -> Tuple[bool, Optional[str]]
"""Verifies UPF API responds on port 8081 at configured base URL."""
```

7.4 Dev - Process Management

Internal methods supporting *Background Process Management*:

```
_monitor_background_processes(timeout: int = 100)
"""Monitors UE and UPF API readiness for 100 seconds, updates status dictionary with␣
↪endpoints availability."""

_verify_tun_interfaces(timeout: int = 30) -> bool
"""Monitors Docker logs for 30 seconds to verify successful creation of uesimtun␣
↪interfaces and PDU sessions."""

_check_tmux_server() -> bool
"""Verifies if tmux server is running by attempting to list sessions."""

_restart_pcf_service()
"""Tears down existing Docker containers and redeploys using docker-compose up."""
```

7.5 Dev - Metrics Management

7.5.1 NetworkMetricsCalculator Internal Methods

Internal methods supporting *Video Streaming Metrics*:

```
_update_interfaces()
"""Monitors uesimtun/ogstun interfaces, collects IP addresses and IO statistics using␣
↪psutil."""

_get_interface_address(iface: str) -> Optional[str]
"""Retrieves first IPv4 address for given interface using psutil.net_if_addrs()."""

_calculate_jitter(timestamps: deque) -> float
"""Calculates RFC 3550 packet jitter using 1/16 smoothing factor and minimum 2␣
↪timestamps."""

_measure_interface_latency(interface: str) -> float
"""Executes single ping with 1s timeout to 10.45.0.1 through specified interface."""
```

(continues on next page)

(continued from previous page)

```
_monitor_network()
"""Background thread running _update_interfaces() every second with 5s retry on failure."
↪""

calculate_metrics() -> Dict[str, Any]
"""Returns dictionary containing throughput, latency, frame rates, timing and size␣
↪statistics."""

record_data_sent(data: Any, timestamp: float) -> None
"""Records packet/frame metrics including size, intervals and estimated frame loss."""

record_data_received(data: Any, timestamp: float, content_type: str) -> None
"""Processes received data latency, frame sequence tracking and timing metrics."""

reset()
"""Clears all metric collections and counters, initializes new monitoring period."""
```

7.5.2 SensorMetricsCalculator Internal Methods

Methods supporting sensor data metrics for *Sensor Data Metrics*:

```
record_data_sent(data: Any, timestamp: float)
"""Records JSON-encoded sensor data size and updates packet metrics."""

record_data_received(data: Any, timestamp: float, content_type: str)
"""Processes sensor data latency (0-1000ms), throughput and RFC 3550 jitter."""

calculate_metrics() -> Dict[str, Any]
"""Returns dictionary of latency, throughput, jitter and packet size statistics."""

reset()
"""Clears all sensor metrics collections and initializes new monitoring period."""
```

7.6 Developer Functions Reference

7.6.1 Configuration Loading

The configuration loading process used by `set_config_path()` and `set_env_path()`:

1. `_ensure_config_loaded()` tracks file modification times to avoid unnecessary reloads
2. `_read_config()` preserves YAML formatting and quotes when loading PCF configuration
3. `_ensure_env_loaded()` manages NUM_UES and path variables in .env file

7.6.2 Process Monitoring

Background process monitoring supporting `run_background_nodes()`:

1. `_monitor_background_processes()` tracks UE and UPF API availability with timeout
2. `_verify_tun_interfaces()` monitors Docker logs for successful PDU session and TUN interface setup
3. Status updates track UE API, UPF API readiness and any error conditions

7.6.3 Wireshark Integration

The Wireshark launch process supporting `launch_wireshark()` and `launch_gtp_wireshark()`:

1. `_check_wireshark_installed()` verifies Wireshark binary location
2. `_get_interface_name()` maps IP addresses to network interface names for packet capture
3. `_get_display_environment()` configures X11 display access for Wireshark GUI window

7.6.4 Network Metrics Collection

The metrics collection process supporting `get_metrics()` and video streaming-related KPIs:

1. `_update_interfaces()` tracks interface statistics using psutil
2. `_calculate_jitter()` implements RFC 3550 jitter calculations
3. `_measure_interface_latency()` monitors interface-specific ping times
4. `calculate_metrics()` computes throughput, latency and frame statistics
5. Thread-safe data recording via `record_data_sent()` and `record_data_received()`

7.6.5 Sensor Metrics Collection

The sensor metrics process supporting `get_sensor_metrics()`:

1. `record_data_sent()` tracks JSON-encoded sensor packet sizes
2. `record_data_received()` calculates latency, throughput and jitter
3. `calculate_metrics()` computes JSON data-specific performance statistics

7.7 Gateway Implementation

The open5gsapi uses two gateway implementations to handle data transmission between UE and UPF containers. These gateways expose Flask endpoints (10.10.0.132:8080 for UE, 10.10.0.112:8081 for UPF) that persist independently of container redeployments, enabling continuous application interaction during policy configuration or UERANSIM updates.

7.7.1 UE API Gateway

> **Note**
>
> Location: `/auto-ue-api.py` in UE container

Key Components:

1. Interface Management:
   - Detects and manages uesimtun interfaces
   - Assigns ports starting from 8080 (one per UE)
2. Endpoints:
   - `/send` (POST): Handles data transmission to UPF
   - `/get_ue_ip` (GET): Returns UE interface IP (for verification of endpoint availability)
3. Data Handling:

- JSON data sent directly
- Binary data (e.g., frames) split into chunks (65000 bytes)
- UDP socket communication to UPF (10.45.0.1:5005)

7.7.2 UPF API Gateway

> **Note**
>
> Location: `/src/upf/upf-api.py` in UPF container

Key Components:

1. Message Queue:
   - Centralized queue for all received data
   - Thread-safe implementation
2. Frame Assembly:
   - Reassembles chunked binary data
   - Handles out-of-order packet reception
3. Endpoints:
   - `/receive/sensor` (GET): JSON data with metadata
   - `/receive/streamer` (GET): Binary data as image/jpeg

7.7.3 Communication Flow

1. Data Sending:
   - Application → UE API (8080+N ports)
   - UE API → UPF (UDP 5005)
   - UPF → Message Queue
2. Data Retrieval:
   - Application ← UPF API (8081)
   - UPF API ← Message Queue

CHAPTER

EIGHT

CONTRIBUTIONS

This API documentation is hosted and maintained on GitHub: https://github.com/ashwinsathish/Open5GS-API.

Also available on PyPI: https://pypi.org/project/open5gsapi/.