OmniPGW Operations Guide

OmniPGW - Packet Gateway Control Plane (PGW-C)

by Omnitouch Network Services

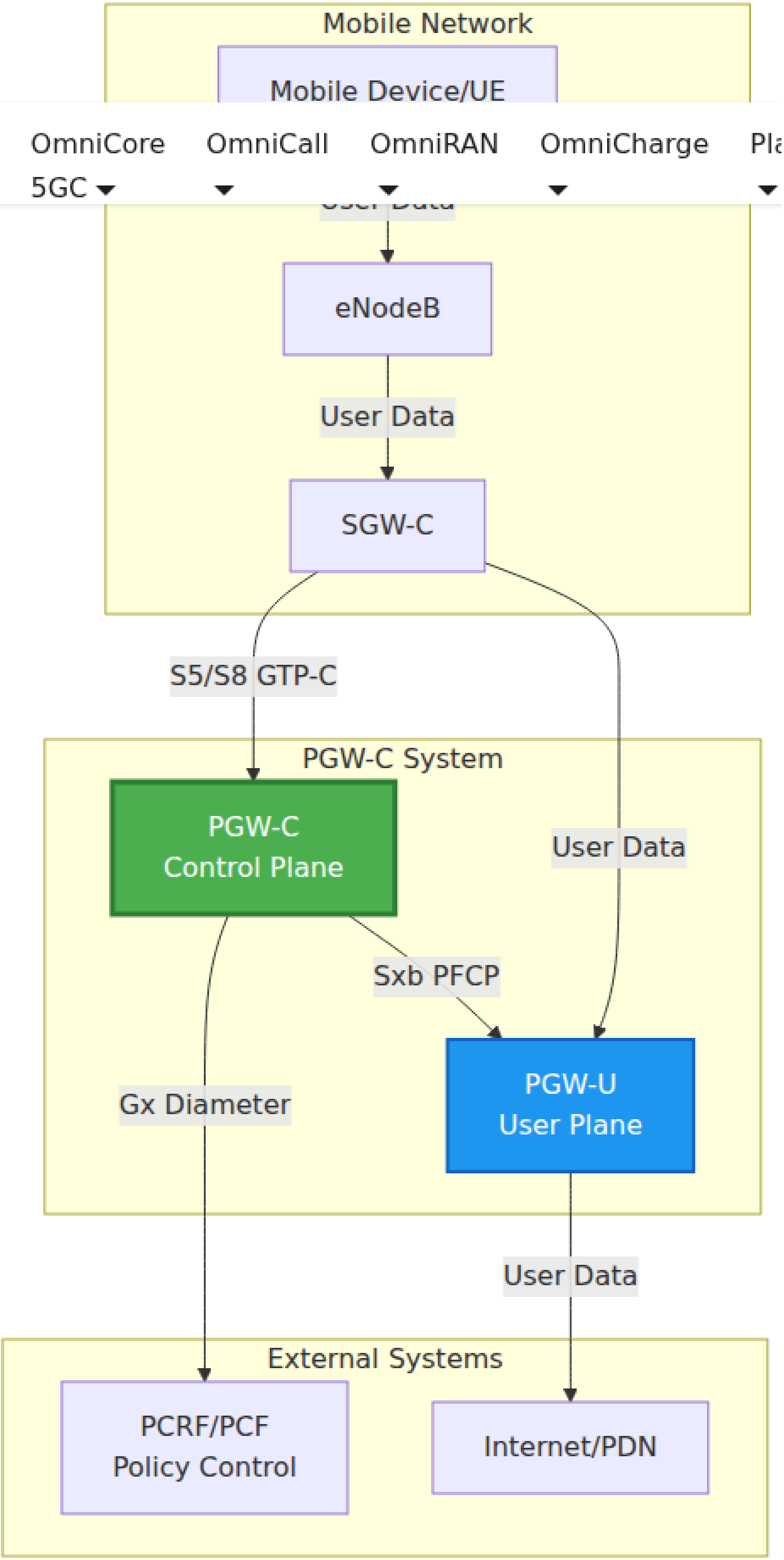
Table of Contents

Overview

OmniPGW is a high-performance Packet Gateway Control Plane (PGW-C) implementation for 3GPP LTE Evolved Packet Core (EPC) networks, developed by Omnitouch Network Services. It manages the control plane functions for data sessions, including:

- **Session Management** - Creating, modifying, and terminating UE (User Equipment) data sessions

- **IP Address Allocation** - Assigning IP addresses to mobile devices from configured pools
- **Policy & Charging Control** - Interfacing with PCRF for policy enforcement and charging
- **User Plane Coordination** - Controlling the PGW-U (User Plane) for packet forwarding

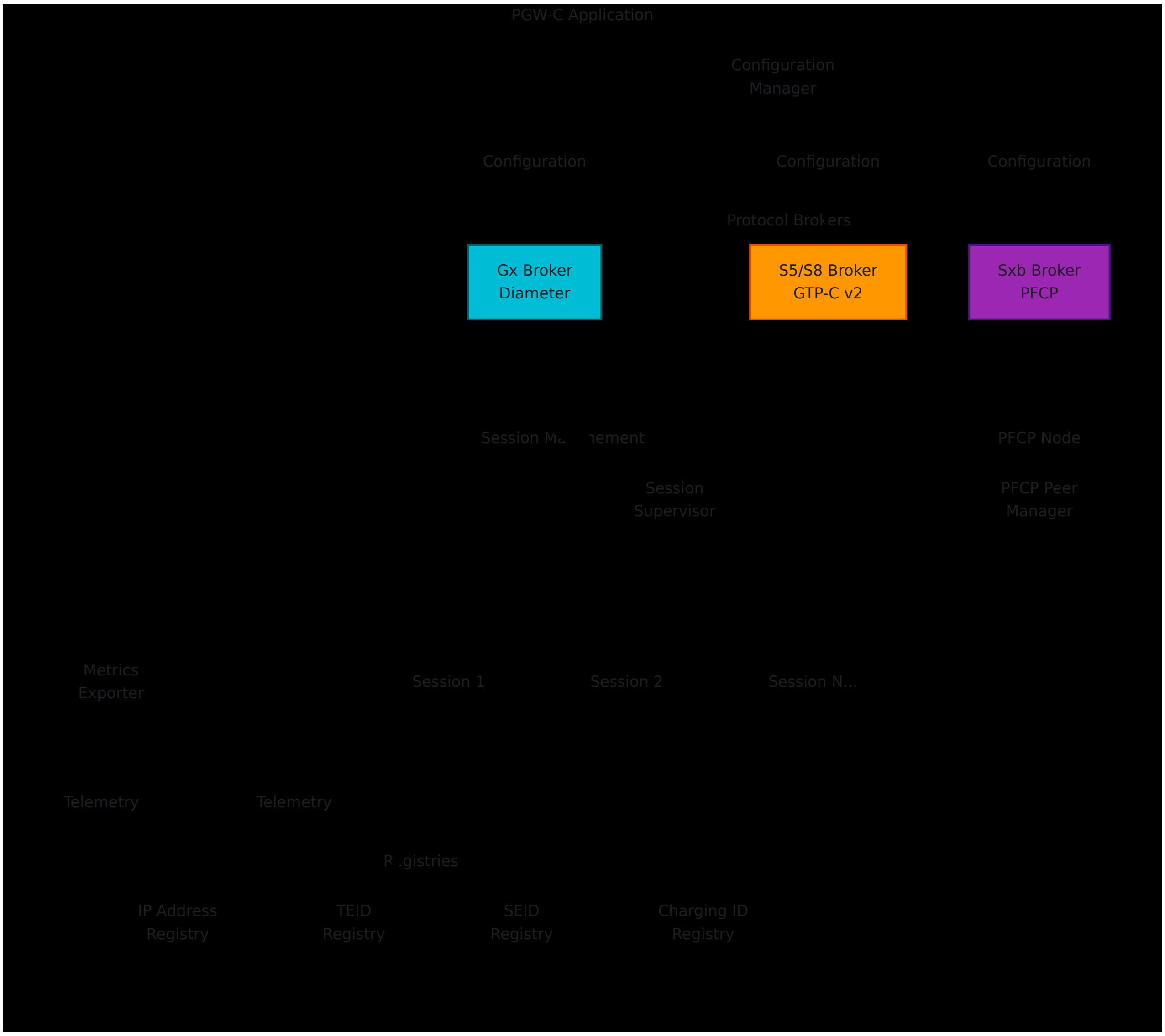

Mobile Network
Mobile Device/UE
eNodeB
User Data
SGW-C
S5/S8 GTP-C
User Data
PGW-C System
PGW-C
Control Plane
Sxb PFCP
Gx Diameter
PGW-U
User Plane
User Data
External Systems
PCRF/PCF
Policy Control
Internet/PDN

What PGW-C Does

- **Accepts session requests** from SGW-C via S5/S8 interface (GTP-C)
- **Allocates UE IP addresses** from configured subnet pools
- **Requests policy decisions** from PCRF via Gx interface (Diameter)
- **Programs forwarding rules** in PGW-U via Sxb interface (PFCP)
- **Manages QoS enforcement** through bearer contexts and QoS rules
- **Tracks charging information** for billing systems

Architecture

Component Overview



Process Architecture

PGW-C is built on Elixir/OTP and uses a supervised process architecture:

- **Application Supervisor** - Top-level supervisor managing all components
- **Protocol Brokers** - Handle incoming/outgoing protocol messages
- **Session Processes** - One GenServer per active PDN connection
- **Registries** - Track allocated resources (IPs, TEIDs, SEIDs, etc.)
- **PFCP Node Manager** - Maintains PFCP associations with PGW-U peers

Each component is supervised and will automatically restart on failure, ensuring system reliability.

Network Interfaces

PGW-C implements three primary 3GPP interfaces:

S5/S8 Interface (GTP-C v2)

Purpose: Control plane signaling between SGW-C and PGW-C

Protocol: GTP-C Version 2 over UDP

Key Messages:

- Create Session Request/Response
- Delete Session Request/Response
- Create Bearer Request/Response
- Delete Bearer Request/Response

Configuration: See S5/S8 Configuration

Sxb Interface (PFCP)

Purpose: Control plane signaling between PGW-C and PGW-U

Protocol: PFCP (Packet Forwarding Control Protocol) over UDP

Key Messages:

- Association Setup Request/Response
- Session Establishment Request/Response
- Session Modification Request/Response
- Session Deletion Request/Response
- Heartbeat Request/Response

Configuration: See PFCP/Sxb Interface Documentation

Gx Interface (Diameter)

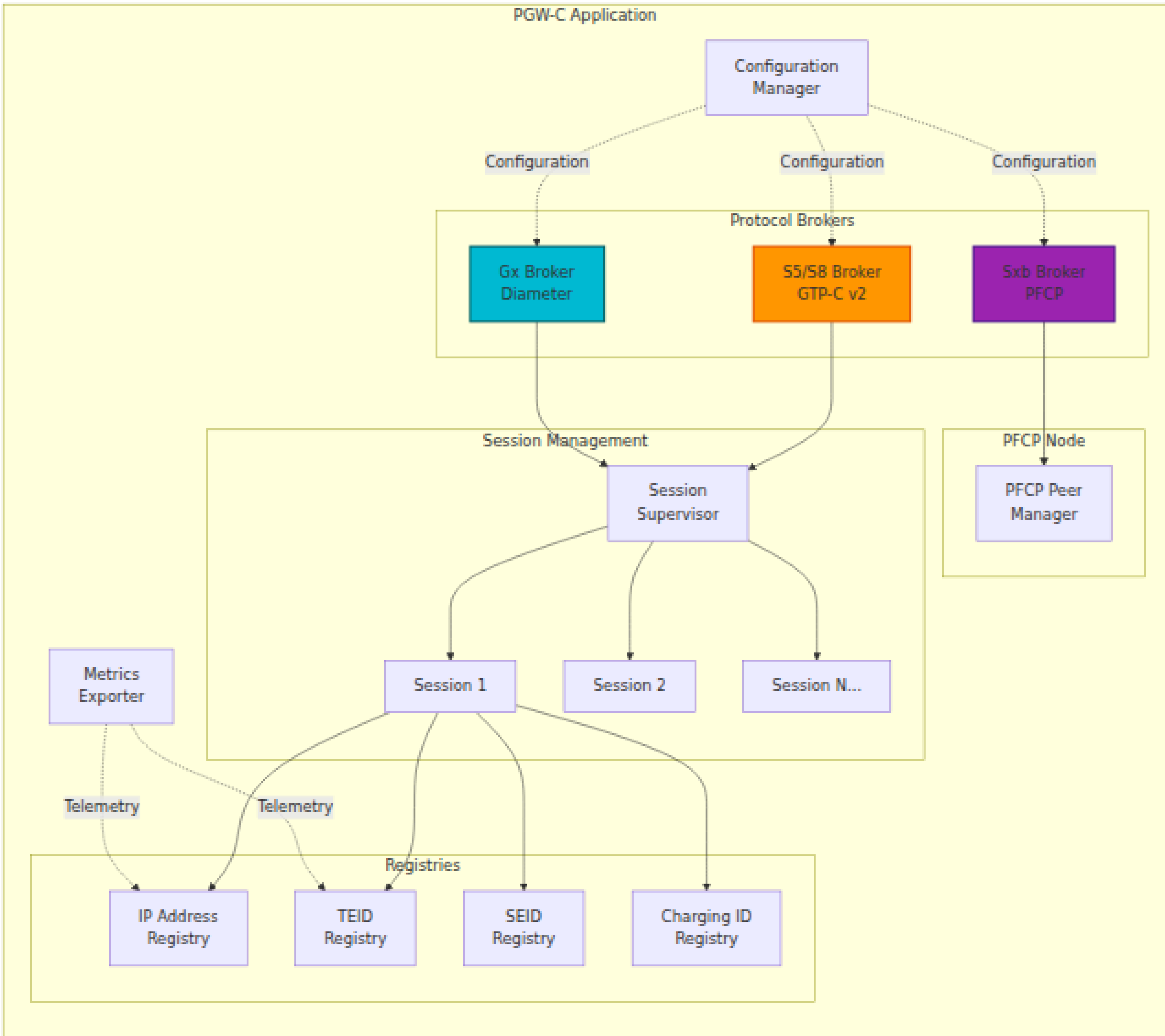
Purpose: Policy and Charging Rules Function (PCRF) interface

Protocol: Diameter (IETF RFC 6733)

Key Messages:

- Credit Control Initial Request/Answer (CCR-I/CCA-I)
- Credit Control Termination Request/Answer (CCR-T/CCA-T)

Configuration: See Diameter Gx Interface Documentation



Key Concepts

PDN Session

A PDN (Packet Data Network) Session represents a UE's data connection to an external network (like the Internet). Each session has:

- **UE IP Address** - Allocated from a configured subnet pool
- **APN** (Access Point Name) - Identifies the external network
- **Bearer Context** - Contains QoS parameters and tunnel information
- **Charging ID** - Unique identifier for billing
- **TEID** (Tunnel Endpoint ID) - S5/S8 interface tunnel identifier
- **SEID** (Session Endpoint ID) - Sxb interface session identifier

Bearer Context

A bearer represents a traffic flow with specific QoS characteristics:

- **Default Bearer** - Created with every PDN session
- **Dedicated Bearers** - Additional bearers for specific QoS needs
- **EBI** (EPS Bearer ID) - Unique identifier for each bearer
- **QoS Parameters** - QCI, ARP, bitrates (MBR, GBR)

PFCP Rules

The PGW-C programs the PGW-U with packet processing rules:

- **PDR** (Packet Detection Rule) - Matches packets (uplink/downlink)
- **FAR** (Forwarding Action Rule) - Specifies forwarding behavior
- **QER** (QoS Enforcement Rule) - Enforces bitrate limits
- **BAR** (Buffering Action Rule) - Controls packet buffering

See PFCP Interface Documentation for details.

IP Address Allocation

UE IP addresses are allocated from configured subnet pools:

- **APN-based selection** - Different APNs can use different subnets
- **Dynamic allocation** - Random IP selection from available range
- **Static allocation** - Support for UE-requested IP addresses
- **Collision detection** - Ensures unique IP assignment

See UE IP Pool Allocation for configuration, and IPv6 / Dual-Stack for IPv6 and IPv4v6 PDN operation.

Getting Started

Prerequisites

- Elixir ~1.16
- Erlang/OTP 26+
- Network connectivity to SGW-C, PGW-U, and PCRF
- Understanding of LTE EPC architecture

Starting OmniPGW

1. **Configure runtime settings** in `config/runtime.exs`
2. **Compile the application:**

   ```
   mix deps.get
   mix compile
   ```

3. **Start the application:**

   ```
   mix run --no-halt
   ```

Verifying Operation

Check the logs for successful startup:

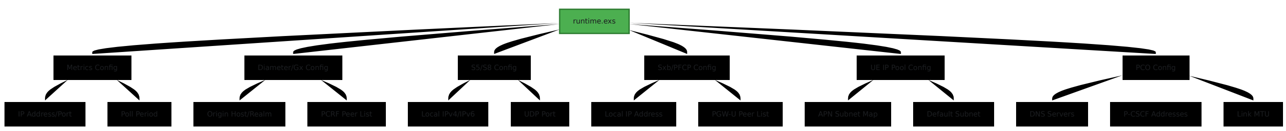
```
[info] Starting OmniPGW...
[info] Starting Metrics Exporter on 127.0.0.42:42069
[info] Starting S5/S8 Broker on 127.0.0.10
[info] Starting Sxb Broker on 127.0.0.20
[info] Starting Gx Broker
[info] Starting PFCP Node Manager
[info] OmniPGW successfully started
```

Access metrics at `http://127.0.0.42:42069/metrics` (configured address).

Configuration

All runtime configuration is defined in `config/runtime.exs`. The configuration is structured into several sections:

Configuration Overview



Quick Configuration Reference

Section	Purpose	Documentation
metrics	Prometheus metrics exporter	Monitoring Guide
diameter	Gx interface to PCRF	Diameter Gx Config
s5s8	GTP-C interface to SGW-C	S5/S8 Config
sxb	PFCP interface to PGW-U	PFCP Config
ue	UE IP address pools	IP Pool Config
pco	Protocol Configuration Options	PCO Config
CDR	Offline charging & usage reporting	CDR Format

See the Complete Configuration Guide for detailed information.

Real-Time Operations Monitoring (OAM API)

OmniPGW exposes an **OAM REST API** for real-time monitoring and operations, providing instant visibility into system status. Query these read endpoints on demand (or on a polling interval / from a script) for a live operational view - no command-line access to the host or raw metrics queries required.

Accessing the OAM API

The API is served over HTTPS/TLS. All routes are served under the `/api` prefix.

```
https://<omnipgw-ip>:8443/
```

Interactive API documentation (Swagger UI) is available at:

```
https://<omnipgw-ip>:8443/api/docs
```

Example health check (use `-k` for a self-signed lab certificate):

```
curl -k https://localhost:8443/api/status
{"result":"ok"}
```

Available Endpoints:

Purpose	Endpoint	Replaces (old page)
Deep dive into a specific subscriber (IMSI/MSISDN/IP)	`POST /api/ue_search`	UE Search
List / search active PDN sessions	`GET /api/sessions[?search=<imsi|msisdn|ip>]`	PGW Sessions
One session by IMSI	`GET /api/sessions/<imsi>`	PGW Sessions (detail)
Session history / audit events	`GET /api/session_history[?type=&search=&page=&page_size=]`	Session History
Network topology overview	`GET /api/topology`	Network Topology
UE IP address pool utilization	`GET /api/ip_pools`, `GET /api/ip_pools/<name>`	IP Pools
UPF / PFCP peer status	`GET /api/upf`, `GET /api/upf/<ip>`	PFCP Sessions / UPF Status
UPF selection rules, health, PCO config	`GET /api/upf_selection`	UPF Selection
Diameter (Gx/Gy) peer connectivity	`GET /api/diameter[?status=connected|disconnected]`, `GET /api/diameter/<origin-host>`	Diameter Peers

Purpose	Endpoint	Replaces (old page)
Online-charging (Gy) status per session	`GET /api/charging[?search=<imsi>]`, `GET /api/charging/<imsi>`	Gy status
P-CSCF DNS discovery status	`GET /api/pcscf_monitor`	P-CSCF Monitor
Health check	`GET /api/status`	-

For a live dashboard-style view, poll the collection endpoints on an interval, for example:

```
watch -n 2 'curl -sk https://localhost:8443/api/sessions'
watch -n 1 'curl -sk https://localhost:8443/api/diameter?
status=disconnected'
```

> Note: the former **Logs** page (live log streaming) and the **Gy Simulator** page have no OAM API equivalent. Stream logs from the host / your log aggregator, and use `GET /api/charging` to inspect live Gy/online-charging state.

Key Features

On-Demand and Polled Data:

- Every endpoint returns current live state from OmniPGW processes
- Poll on an interval with `watch`/cron/a script for an auto-refreshing view
- Status fields (e.g. peer `connected`/`associated`) are returned explicitly in the JSON

Search & Filter:

- Search sessions by IMSI, IP, MSISDN, or APN via `GET /api/sessions?search=...`
- Filter Diameter peers by `?status=connected|disconnected`
- Filter charging state by `?search=<imsi>`

Full Detail Records:

- Per-entity endpoints (`/api/sessions/<imsi>`, `/api/upf/<ip>`, `/api/diameter/<origin-host>`, `/api/charging/<imsi>`) return complete state as JSON
- Inspect full session, peer configuration, and capabilities

Access Control:

- Served over HTTPS/TLS on port 8443
- Bind to the management IP only and front with your access controls
- Intended for NOC/operations team and automation use

Operational Workflows

Session Troubleshooting (Deep Dive):

```
1. User reports connection issue
2. POST /api/ue_search with a JSON body containing the IMSI,
MSISDN, or IP
3. Review the comprehensive session details in the response:
   a) Active Sessions - Verify session exists with correct
parameters
   b) Current Location - Check TAC, Cell ID, serving network
   c) Bearer Information - Verify default and dedicated bearers
      - QCI, MBR/GBR, Charging Rule Names
      - APN-AMBR limits
   d) Charging Information - Gy session ID, quota status (also GET
/api/charging/<imsi>)
   e) Policy Information - Gx session, installed PCC rules
   f) Recent Events - GET /api/session_history?search=<imsi>
4. If session not found → GET /api/diameter?status=disconnected
for PCRF connectivity
```

Quick Session Lookup:

```
1. User reports issue
2. GET /api/sessions?search=<imsi|msisdn> (or GET
/api/sessions/<imsi>)
3. Verify session exists with basic details:
   - UE IP address allocated
   - QoS parameters
   - Tunnel endpoints established
4. For detailed analysis → POST /api/ue_search
```

System Health Check:

```
1. GET /api/upf              → Verify all PGW-U peers "Associated"
2. GET /api/diameter?status=connected → Verify all PCRF peers
"Connected"
3. GET /api/sessions         → Check active session count vs.
capacity
4. GET /api/status           → Overall health check
({"result":"ok"})
```

Capacity Monitoring:

- Count entries returned by `GET /api/sessions`
- Compare to licensed/expected capacity
- Identify peak usage times
- Monitor distribution across APNs; use `GET /api/ip_pools` for per-pool utilization

API vs. Metrics

Use the OAM API for:

- Deep-dive subscriber troubleshooting (`POST /api/ue_search`)
- Individual session details and state inspection (`GET /api/sessions/<imsi>`)
- Real-time peer status (`GET /api/upf`, `GET /api/diameter`)

- Quick health checks across all interfaces (`GET /api/status`)
- Troubleshooting specific users by IMSI/MSISDN/IP
- Bearer QoS analysis (MBR, GBR, QCI)
- Policy and charging rule inspection (`GET /api/charging`)
- Session history and audit trails (`GET /api/session_history`)
- IP pool capacity monitoring (`GET /api/ip_pools`)
- Verifying configuration and rules (`GET /api/upf_selection`)

Use Prometheus Metrics for:

- Historical trends
- Alerting and notifications
- Capacity planning graphs
- Performance analysis
- Long-term monitoring

Best Practice: Use both together - the OAM API for immediate operations, Prometheus for trends and alerts.

Monitoring & Metrics

In addition to the OAM API, OmniPGW exposes Prometheus-compatible metrics for monitoring:

Available Metrics

- **Session Metrics**
  - `teid_registry_count` - Active S5/S8 sessions
  - `seid_registry_count` - Active PFCP sessions
  - `session_id_registry_count` - Active Gx sessions
  - `address_registry_count` - Allocated UE IP addresses
  - `charging_id_registry_count` - Active charging IDs
- **Message Metrics**

  - `s5s8_inbound_messages_total` - GTP-C messages received
  - `sxb_inbound_messages_total` - PFCP messages received
  - `gx_inbound_messages_total` - Diameter messages received
  - Message handling duration distributions

- **Error Metrics**

  - `s5s8_inbound_errors_total` - S5/S8 protocol errors
  - `sxb_inbound_errors_total` - PFCP protocol errors
  - `gx_inbound_errors_total` - Diameter errors

- **Session Lifecycle Metrics**

  - `pgw_session_create_total{result,cause,pdn_type,rat_type}` - PDN session creation attempts, broken down by result (`success`/`failure`), the GTP cause returned, PDN type (`ipv4`/`ipv6`/`ipv4v6`) and RAT type
  - `pgw_session_delete_total{pdn_type}` - PDN session deletions by PDN type
  - `pgw_session_modify_total` - Modify Bearer Requests handled

- **UE IP Pool Metrics**

  - `ue_pool_addresses_allocated{pool,ip_version}` - Allocated UE addresses per pool (APN pattern) and IP version
  - `ue_pool_addresses_total{pool,ip_version}` - Total pool capacity
  - `ue_pool_addresses_utilization_ratio{pool,ip_version}` - Fraction allocated (0.0-1.0); useful for pool-exhaustion alerting. Note pools that share an address range each count the shared allocations
  - `ue_ip_allocation_failures_total{ip_version,reason}` - Allocation failures (`pool_exhausted`, `already_registered`)

- **Per-Peer UPF / PFCP Metrics**

  - `upf_peer_associated{peer_ip}` - 1/0 PFCP association state per peer
  - `upf_peer_healthy{peer_ip}` - 1/0 health per peer
  - `upf_peer_missed_heartbeats{peer_ip}` - Consecutive missed heartbeats per peer

- `upf_heartbeat_rtt` - PFCP heartbeat round-trip time distribution, by `peer_ip`

- **Bearer & QoS Metrics**

  - `pgw_bearer_create_total{type,qci}` - Bearers created by type (`default`/`dedicated`) and QCI
  - `pgw_bearer_delete_total{type}` - Bearers deleted by type

- **Diameter Peer Metrics**

  - `diameter_peers_connected{application}` - Connected Diameter peers by application (`gx`/`gy`/`all`)

Accessing Metrics

Metrics are exposed via HTTP at the configured endpoint:

```
curl http://127.0.0.42:42069/metrics
```

See Monitoring & Metrics Guide for dashboard setup and alerting.

Detailed Documentation

This section provides a comprehensive overview of all OmniPGW documentation. Documents are organized by topic and use case.

Documentation Structure

```
OmniPGW Documentation
├── OPERATIONS.md (This Guide)
│
└── docs/
    ├── Configuration & Setup
    │   ├── configuration.md         Complete runtime.exs
reference
    │   ├── ue-ip-allocation.md      IP pool configuration
    │   └── pco-configuration.md     DNS, P-CSCF, MTU settings
    │
    ├── Network Interfaces
    │   ├── pfcp-interface.md        Sxb/PFCP (PGW-U
communication)
    │   ├── diameter-gx.md           Gx (PCRF communication)
    │   ├── diameter-gy.md           Gy/Ro (OCS communication)
    │   └── s5s8-interface.md        S5/S8 (SGW-C communication)
    │
    └── Operations
        ├── session-management.md    PDN session lifecycle
        └── monitoring.md            Prometheus metrics &
alerting
```

Documentation by Topic

🚀 Getting Started

Document	Description	Purpose
OPERATIONS.md	Main operations guide (this document)	Overview and quick start

⚙ Configuration

Document	Description	Lines
configuration.md	Complete runtime.exs configuration reference	1,600+
ue-ip-allocation.md	UE IP pool management and allocation	943
ipv6-dual-stack.md	IPv6 / IPv4v6 PDN operation across PGW-C and UPF	-
pco-configuration.md	Protocol Configuration Options (DNS, P-CSCF, MTU)	344

🔌 Network Interfaces

Document	Description	Lines
pfcp-interface.md	PFCP/Sxb interface to PGW-U	1,355
diameter-gx.md	Diameter Gx interface to PCRF (Policy Control)	941
diameter-gy.md	Diameter Gy/Ro interface to OCS (Online Charging)	1,100+
s5s8-interface.md	GTP-C S5/S8 interface to SGW-C	456

📊 Operations & Monitoring

Document	Description	Lines
session-management.md	PDN session lifecycle and operations	435
monitoring.md	Prometheus metrics, Grafana dashboards, alerting	807
data-cdr-format.md	CDR file format, URR configuration, offline charging	847
qos-bearers.md	QoS & bearer management, policy control	448
troubleshooting.md	Troubleshooting procedures and common issues	687

🔧 Advanced Features

Document	Description	Lines
pcscf-monitoring.md	P-CSCF discovery and health monitoring	894

Documentation Features

📊 Mermaid Diagrams

All documents include **Mermaid charts** for visual understanding:

- Architecture diagrams
- Sequence diagrams (message flows)
- State machines
- Network topology

💡 Practical Examples

Every document includes:

- Real-world configuration examples
- Copy-paste ready configs
- Common use cases

□ Troubleshooting

Each interface document includes:

- Common issues and solutions
- Debug commands
- Metrics for diagnosis

□ Cross-References

Documents are extensively cross-linked for easy navigation.

Reading Paths

For Network Operators

1. OPERATIONS.md - Overview (this document)
2. configuration.md - Setup
3. monitoring.md - Monitoring
4. session-management.md - Day-to-day operations

For Network Engineers

1. OPERATIONS.md - Architecture overview (this document)
2. pfcp-interface.md - User plane control
3. diameter-gx.md - Policy control
4. diameter-gy.md - Online charging
5. s5s8-interface.md - Session management
6. ue-ip-allocation.md - IP management

For Configuration & Deployment

1. configuration.md - Complete reference
2. ue-ip-allocation.md - IP pools
3. pco-configuration.md - Network parameters
4. monitoring.md - Set up monitoring

Document Statistics

- **Total Documents:** 14
- **Total Lines:** ~10,900+
- **Total Size:** ~265 KB
- **Mermaid Diagrams:** 75+
- **Code Examples:** 150+

Key Concepts Covered

Architecture

- □ Control/User plane separation
- □ OTP/Elixir architecture
- □ Process supervision
- □ GenServer-based sessions

Protocols

- □ PFCP (Packet Forwarding Control Protocol)
- □ GTP-C v2 (GPRS Tunnelling Protocol)
- □ Diameter (RFC 6733)

3GPP Interfaces

- □ Sxb (PGW-C ↔ PGW-U)
- □ Gx (PGW-C ↔ PCRF)
- □ Gy/Ro (PGW-C ↔ OCS)
- □ S5/S8 (SGW-C ↔ PGW-C)

Operations

- □ Session management
- □ IP allocation strategies
- □ QoS enforcement
- □ Charging integration
- □ Monitoring & alerting

Additional Resources

3GPP Specifications

Spec	Title
TS 29.274	GTP-C v2 (S5/S8 interface)
TS 29.244	PFCP (Sxb interface)
TS 29.212	Diameter Gx interface (Policy Control)
TS 32.299	Diameter Charging Applications (Gy/Ro)
TS 32.251	Packet Switched domain charging
TS 23.401	EPC architecture

Related Documentation

- Configuration file: config/runtime.exs

OmniPGW Configuration Guide

Complete Reference for runtime.exs Configuration

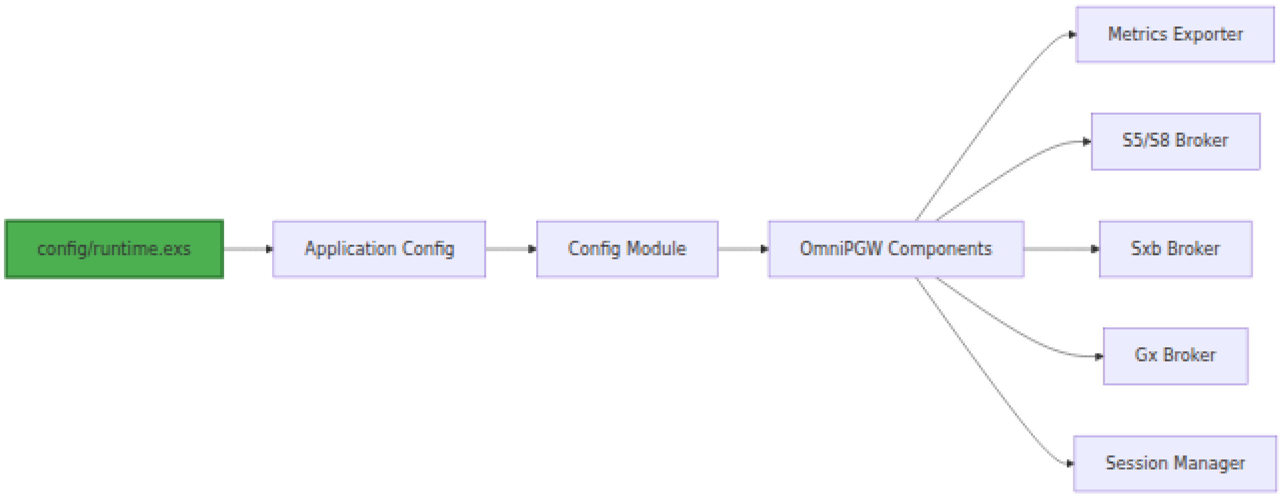
by Omnitouch Network Services

Table of Contents

Overview

OmniPGW uses **runtime configuration** defined in `config/runtime.exs`. This file is evaluated at **application startup** and allows for dynamic configuration based on environment variables or external sources.

Configuration Philosophy



Key Principles:

- **Single Source of Truth** - All configuration in one file
- **Type Safety** - Configuration validated at startup
- **Environment Flexibility** - Support for dev, test, production
- **Clear Defaults** - Sensible defaults with explicit overrides

---

Configuration File Structure

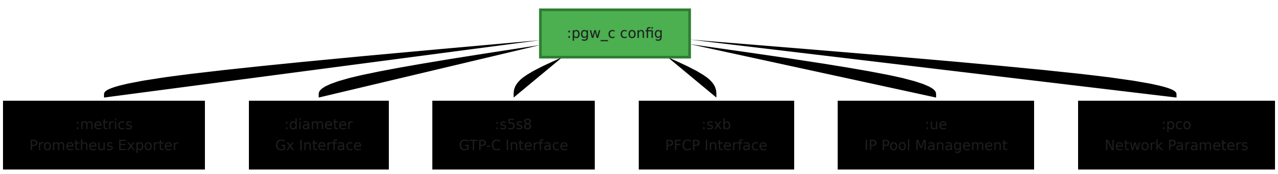
File Location

```
pgw_c/
├── config/
│   ├── config.exs         # Base configuration (imports
runtime.exs)
│   ├── dev.exs            # Development-specific config
│   ├── prod.exs           # Production-specific config
│   └── runtime.exs        # ← Main configuration file
```

Top-Level Structure

```
# config/runtime.exs
import Config

config :logger, level: :info

config :pgw_c,
  metrics: %{...},
  diameter: %{...},
  s5s8: %{...},
  sxb: %{...},
  ue: %{...},
  pco: %{...}
```

Configuration Sections



Metrics Configuration

Purpose

Configure the Prometheus metrics exporter for monitoring OmniPGW.

Configuration Block

```
config :pgw_c,
  metrics: %{
    # Enable/disable metrics exporter
    enabled: true,

    # IP address to bind HTTP server
    ip_address: "0.0.0.0",

    # Port for metrics endpoint
    port: 9090,

    # How often to poll registries (milliseconds)
    registry_poll_period_ms: 10_000
  }
```

Parameters

Parameter	Type	Default	Description
`enabled`	Boolean	`true`	Enable metrics exporter
`ip_address`	String (IP)	`"0.0.0.0"`	Bind address (0.0.0.0 = all interfaces)
`port`	Integer	`9090`	HTTP port for `/metrics` endpoint
`registry_poll_period_ms`	Integer	`10_000`	Polling interval for registry counts

Examples

Production - Bind to specific IP:

```
metrics: %{
  enabled: true,
  ip_address: "10.0.0.20",  # Management network
  port: 9090,
  registry_poll_period_ms: 5_000  # Poll every 5 seconds
}
```

Development - Localhost only:

```
metrics: %{
  enabled: true,
  ip_address: "127.0.0.1",
  port: 42069,  # Non-standard port
  registry_poll_period_ms: 10_000
}
```

Disable metrics:

```
metrics: %{
  enabled: false
}
```

Accessing Metrics

```
# Default endpoint
curl http://<ip_address>:<port>/metrics

# Example
curl http://10.0.0.20:9090/metrics
```

See: Monitoring & Metrics Guide for detailed metrics documentation.

Diameter/Gx Configuration

Purpose

Configure the Diameter protocol for Gx interface (PCRF communication).

Configuration Block

```
config :pgw_c,
  diameter: %{
    # IP address to listen for Diameter connections
    listen_ip: "0.0.0.0",

    # OmniPGW's Diameter identity (Origin-Host)
    host: "omnipgw.epc.mnc001.mcc001.3gppnetwork.org",

    # OmniPGW's Diameter realm (Origin-Realm)
    realm: "epc.mnc001.mcc001.3gppnetwork.org",

    # List of PCRF peers
    peer_list: [
      %{
        # PCRF Diameter identity
        host: "pcrf.epc.mnc001.mcc001.3gppnetwork.org",

        # PCRF realm
        realm: "epc.mnc001.mcc001.3gppnetwork.org",

        # PCRF IP address
        ip: "10.0.0.30",

        # Initiate connection to PCRF
        initiate_connection: true
      }
    ]
  }
```

Parameters

Parameter	Type	Required	Description
`listen_ip`	String (IP)	Yes	Diameter listen address
`host`	String (FQDN)	Yes	OmniPGW's Origin-Host (must be FQDN)
`realm`	String (Domain)	Yes	OmniPGW's Origin-Realm
`peer_list`	List	Yes	PCRF peer configurations

Peer Configuration:

Parameter	Type	Required	Description
`host`	String (FQDN)	Yes	PCRF Diameter identity
`realm`	String (Domain)	Yes	PCRF realm
`ip`	String (IP)	Yes	PCRF IP address
`initiate_connection`	Boolean	Yes	Whether OmniPGW connects to PCRF

FQDN Format

Diameter identities MUST be FQDNs:

```
# CORRECT
host: "omnipgw.epc.mnc001.mcc001.3gppnetwork.org"

# INCORRECT
host: "omnipgw"              # Not a FQDN
host: "10.0.0.20"            # IP not allowed
```

3GPP Format:

```
<hostname>.epc.mnc<MNC>.mcc<MCC>.3gppnetwork.org

Examples:
- omnipgw.epc.mnc001.mcc001.3gppnetwork.org  (MCC=001, MNC=001)
- pgw-c.epc.mnc260.mcc310.3gppnetwork.org   (MCC=310, MNC=260 - US
T-Mobile)
```

Examples

Single PCRF:

```
diameter: %{
  listen_ip: "0.0.0.0",
  host: "omnipgw.epc.mnc001.mcc001.3gppnetwork.org",
  realm: "epc.mnc001.mcc001.3gppnetwork.org",
  peer_list: [
    %{
      host: "pcrf.epc.mnc001.mcc001.3gppnetwork.org",
      realm: "epc.mnc001.mcc001.3gppnetwork.org",
      ip: "10.0.0.30",
      initiate_connection: true
    }
  ]
}
```

Multiple PCRFs (Redundancy):

```
diameter: %{
  listen_ip: "0.0.0.0",
  host: "omnipgw.epc.mnc001.mcc001.3gppnetwork.org",
  realm: "epc.mnc001.mcc001.3gppnetwork.org",
  peer_list: [
    %{
      host: "pcrf-primary.epc.mnc001.mcc001.3gppnetwork.org",
      realm: "epc.mnc001.mcc001.3gppnetwork.org",
      ip: "10.0.1.30",
      initiate_connection: true
    },
    %{
      host: "pcrf-backup.epc.mnc001.mcc001.3gppnetwork.org",
      realm: "epc.mnc001.mcc001.3gppnetwork.org",
      ip: "10.0.2.30",
      initiate_connection: true
    }
  ]
}
```

PCRF-Initiated Connection:

```
diameter: %{
  listen_ip: "0.0.0.0",
  host: "omnipgw.epc.mnc001.mcc001.3gppnetwork.org",
  realm: "epc.mnc001.mcc001.3gppnetwork.org",
  peer_list: [
    %{
      host: "pcrf.epc.mnc001.mcc001.3gppnetwork.org",
      realm: "epc.mnc001.mcc001.3gppnetwork.org",
      ip: "10.0.0.30",
      initiate_connection: false  # Wait for PCRF to connect
    }
  ]
}
```

See: Diameter Gx Interface Documentation

S5/S8 Configuration

Purpose

Configure the GTP-C interface for communication with SGW-C.

Configuration Block

```
config :pgw_c,
  s5s8: %{
    # Local IPv4 address for S5/S8 interface
    local_ipv4_address: "10.0.0.20",

    # Optional: Local IPv6 address
    local_ipv6_address: nil,

    # Optional: Override default GTP-C port (2123)
    local_port: 2123,

    # GTP-C request timeout in milliseconds (default: 500ms)
    # Timeout per attempt when waiting for GTP-C responses
    request_timeout_ms: 500,

    # Number of retry attempts for GTP-C requests (default: 3)
    # Total maximum wait time = request_timeout_ms *
request_attempts
    request_attempts: 3
  }
```

Parameters

Parameter	Type	Default	Description
`local_ipv4_address`	String (IPv4)	Required	S5/S8 interface IPv4 address
`local_ipv6_address`	String (IPv6)	`nil`	S5/S8 interface IPv6 address (optional)
`local_port`	Integer	`2123`	UDP port for GTP-C (standard port 2123)
`request_timeout_ms`	Integer	`500`	Timeout per retry attempt in milliseconds
`request_attempts`	Integer	`3`	Number of retry attempts before giving up

Protocol Details

- **Protocol:** GTP-C Version 2
- **Transport:** UDP
- **Standard Port:** 2123
- **Direction:** Receives from SGW-C

Examples

IPv4 Only (Common):

```
s5s8: %{
  local_ipv4_address: "10.0.0.20"
}
```

IPv4 + IPv6 Dual-Stack:

```
s5s8: %{
  local_ipv4_address: "10.0.0.20",
  local_ipv6_address: "2001:db8::20"
}
```

Custom Port (Non-Standard):

```
s5s8: %{
  local_ipv4_address: "10.0.0.20",
  local_port: 2124  # Custom port
}
```

High Latency Network:

```
s5s8: %{
  local_ipv4_address: "10.0.0.20",
  request_timeout_ms: 1500,  # 1.5 seconds per attempt
  request_attempts: 3         # Total: 4.5 seconds max
}
```

Timeout Configuration

The S5/S8 interface uses configurable timeouts for GTP-C request/response transactions (Create Bearer Request, Delete Bearer Request).

Total Wait Time Calculation:

```
Total Maximum Wait = request_timeout_ms × request_attempts
Default: 500ms × 3 = 1.5 seconds
```

Tuning Guidelines:

Network Latency	Recommended Timeout	Total Wait Time
Low latency (<50ms)	200-300ms	600-900ms
Normal (50-150ms)	500ms (default)	1.5s
High latency (>150ms)	1000-2000ms	3-6s
Satellite/unstable	2000-3000ms	6-9s

When to Adjust:

- **Increase timeout** if seeing frequent "Create Bearer Request timed out" errors but Wireshark shows responses arriving
- **Decrease timeout** for faster failure detection in low-latency environments
- **Increase retry attempts** for unreliable networks with packet loss

Timeout Behavior:

- On timeout, error is logged: `"Create Bearer Request timed out"`
- Diameter error returned to PCRF: Result-Code 5012 (UNABLE_TO_COMPLY)
- Bearer remains in early storage for cleanup when Charging-Rule-Remove arrives

Network Planning

IP Address Selection:

- Use dedicated management/signaling network
- Ensure reachability from all SGW-C nodes
- Consider redundancy (VRRP/HSRP) for HA

Firewall Rules:

```
# Allow GTP-C from SGW-C
iptables -A INPUT -p udp --dport 2123 -s <sgw_c_network> -j ACCEPT
```

Gn/Gp Configuration (GGSN)

Purpose

Configure the GTP-C v1 interface for communication with SGSNs, enabling GGSN functionality for 2G/3G networks.

Configuration Block

```
config :pgw_c,
  gn: %{
    # Local IPv4 address for Gn interface
    local_ipv4_address: "10.0.0.20",

    # Optional: Local IPv6 address (most 2G/3G networks are IPv4-only)
    local_ipv6_address: nil,

    # GTP-C port (shared with S5/S8)
    local_port: 2123
  },

  # DNS servers returned in PCO (shared with S5/S8)
  dns: %{
    primary_ipv4: {8, 8, 8, 8},
    secondary_ipv4: {8, 8, 4, 4}
  }
```

Parameters

Parameter	Type	Default	Description
`local_ipv4_address`	String (IPv4)	Required	Gn interface IPv4 address (GGSN address)
`local_ipv6_address`	String (IPv6)	`nil`	Gn interface IPv6 address (optional, rarely used)
`local_port`	Integer	`2123`	UDP port for GTP-C v1 (shared with S5/S8)

DNS Parameters

Parameter	Type	Default	Description
`primary_ipv4`	Tuple	`{8, 8, 8, 8}`	Primary DNS server for PCO response
`secondary_ipv4`	Tuple	`{8, 8, 4, 4}`	Secondary DNS server for PCO response

Protocol Details

- **Protocol:** GTP-C Version 1 (3GPP TS 29.060)
- **Transport:** UDP
- **Standard Port:** 2123 (shared with GTP-C v2)
- **Direction:** Receives from SGSNs

Coexistence with S5/S8

OmniPGW can operate as both PGW (4G) and GGSN (2G/3G) simultaneously:

- Both interfaces share UDP port 2123

- GTP version is auto-detected from message headers
- Same IP address can serve both 4G and 2G/3G traffic
- Shared infrastructure: IP pools, UPF/PFCP, online charging

See: Gn/Gp Interface Documentation

Sxb/PFCP Configuration

Purpose

Configure the PFCP interface for communication with PGW-U (User Plane).

Configuration Block

```
config :pgw_c,
  sxb: %{
    # Local IP address for PFCP communication
    local_ip_address: "10.0.0.20",

    # Optional: Override default PFCP port (8805)
    local_port: 8805
  }
```

Parameters

Parameter	Type	Default	Description
`local_ip_address`	String (IP)	Required	PFCP listen address
`local_port`	Integer	`8805`	PFCP UDP port

> **Important:**

> - **All UPF peers are automatically registered** from the `upf_selection` configuration (rules + fallback pool) at startup
> - Auto-registered UPFs use sensible defaults:
>   - Auto-generated name: `"UPF-<ip>:<port>"`
>   - Passive PFCP association (wait for UPF to initiate)
>   - 5-second heartbeat interval
> - UPF selection rules and pools are configured in the separate `upf_selection` section. See UPF Selection Strategies below.
> - Dynamic UPF registration is supported for DNS-discovered UPFs that aren't in the configuration

Examples

Minimal Configuration:

```
sxb: %{
  local_ip_address: "10.0.0.20"
}

# All UPFs in upf_selection will be automatically registered with:
# - Auto-generated name: "UPF-10.0.0.21:8805"
# - Passive PFCP association (wait for UPF to connect)
# - 5-second heartbeat interval
```

Custom PFCP Port:

```
sxb: %{
  local_ip_address: "10.0.0.20",
  local_port: 8806  # Non-standard PFCP port
}
```

Complete Example with UPF Selection:

```
sxb: %{
  local_ip_address: "10.0.0.20"
},
upf_selection: %{
  rules: [
    %{
      name: "IMS Pool",
      priority: 10,
      match_field: :apn,
      match_regex: ~r/^ims$/,
      upf_pool: [
        %{remote_ip_address: "10.0.1.21", remote_port: 8805,
weight: 100},
        %{remote_ip_address: "10.0.1.22", remote_port: 8805,
weight: 100}
      ]
    }
  ],
  fallback_pool: [
    %{remote_ip_address: "10.0.2.21", remote_port: 8805, weight:
100}
  ]
}
# All 3 UPFs (10.0.1.21, 10.0.1.22, 10.0.2.21) are automatically
registered
```

DNS-Based Selection (Dynamic Registration):

```
sxb: %{
  local_ip_address: "10.0.0.20"
},
upf_selection: %{
  dns_enabled: true,
  dns_query_priority: [:ecgi, :tai],
  dns_suffix: "epc.3gppnetwork.org",
  fallback_pool: [
    %{remote_ip_address: "10.0.2.21", remote_port: 8805, weight:
100}
  ]
}
# DNS-discovered UPFs will be dynamically registered on first use
```

UPF Selection Strategies

Important: UPF selection configuration has been simplified. All UPF peers are automatically registered from the `upf_selection` configuration.

Configuration Structure

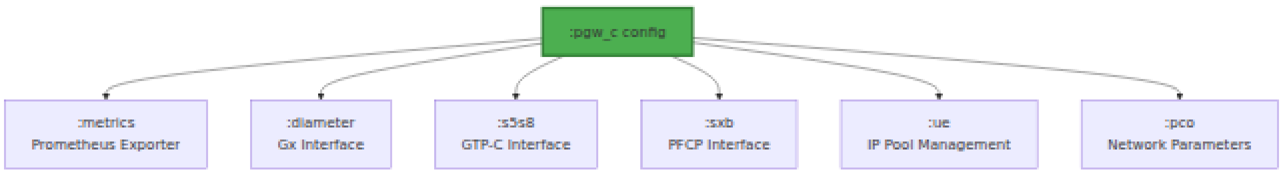
UPF selection is configured in the **`upf_selection`** section which defines:

1. **Static Rules** - Pattern-based routing with load balancing pools
2. **DNS Settings** - Location-based dynamic UPF discovery
3. **Fallback Pool** - Default pool when no rules match and DNS fails

Selection Priority Order

1. **Static Rules** (Highest Priority) - Pattern-based routing with load balancing pools
2. **DNS-Based Selection** (Lower Priority) - Location-based dynamic UPF discovery
3. **Fallback Pool** (Lowest Priority) - Default pool when no rules match and DNS fails

UPF Selection Decision Flow



Available Match Fields

Static rules can match on any of these session attributes:

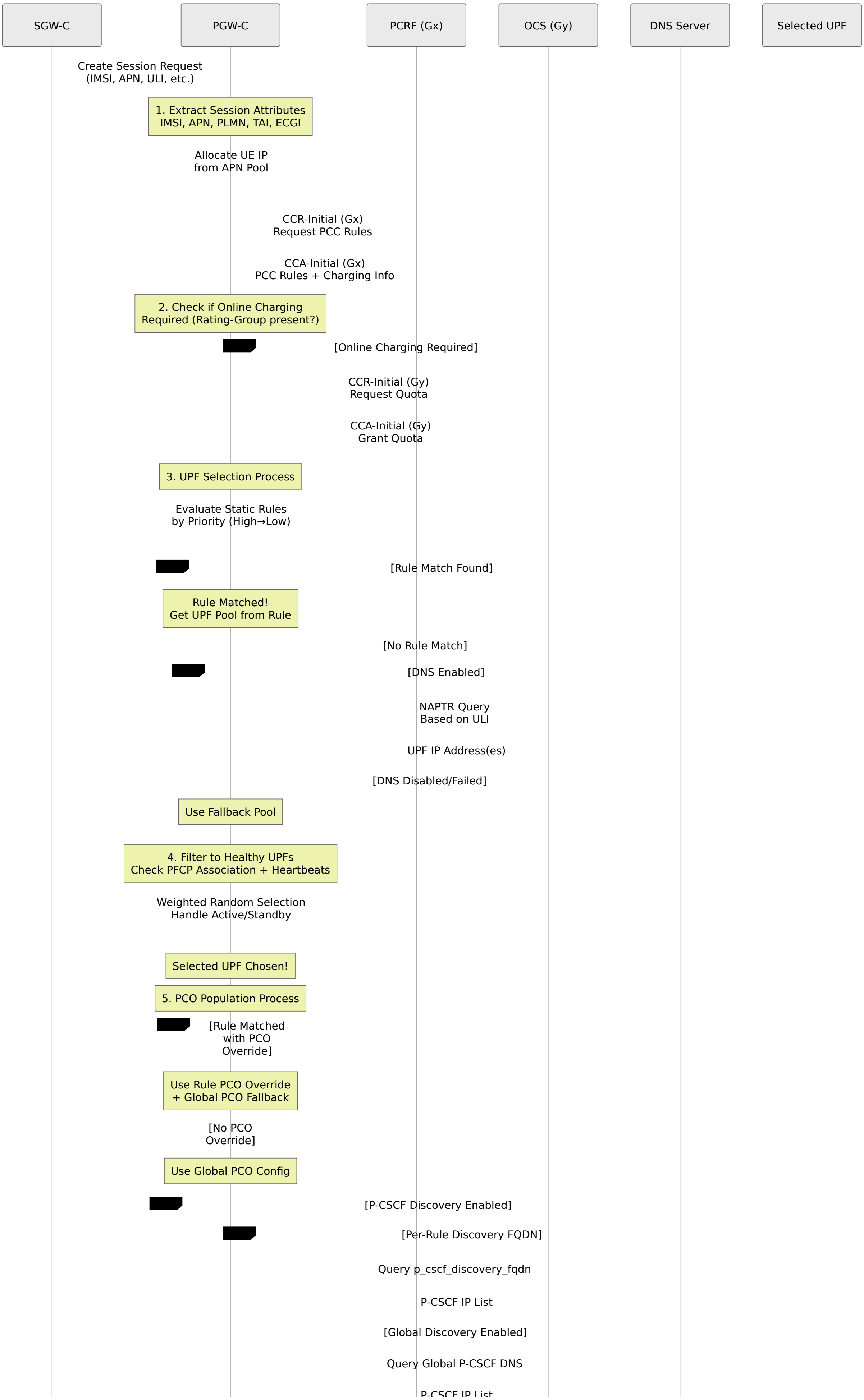
Match Field	Description	Example Pattern
`:imsi`	International Mobile Subscriber Identity	`^001001.*` (test network)
`:apn`	Access Point Name / DNN	`^internet\.` or `^ims\.`
`:serving_network_plmn_id`	Serving network identifier	`^001001$`
`:sgw_ip_address`	SGW IP address	`^10\.100\..*`
`:uli_tai_plmn_id`	Tracking Area PLMN ID	`^313.*`
`:uli_ecgi_plmn_id`	E-UTRAN Cell PLMN ID	`^313.*`

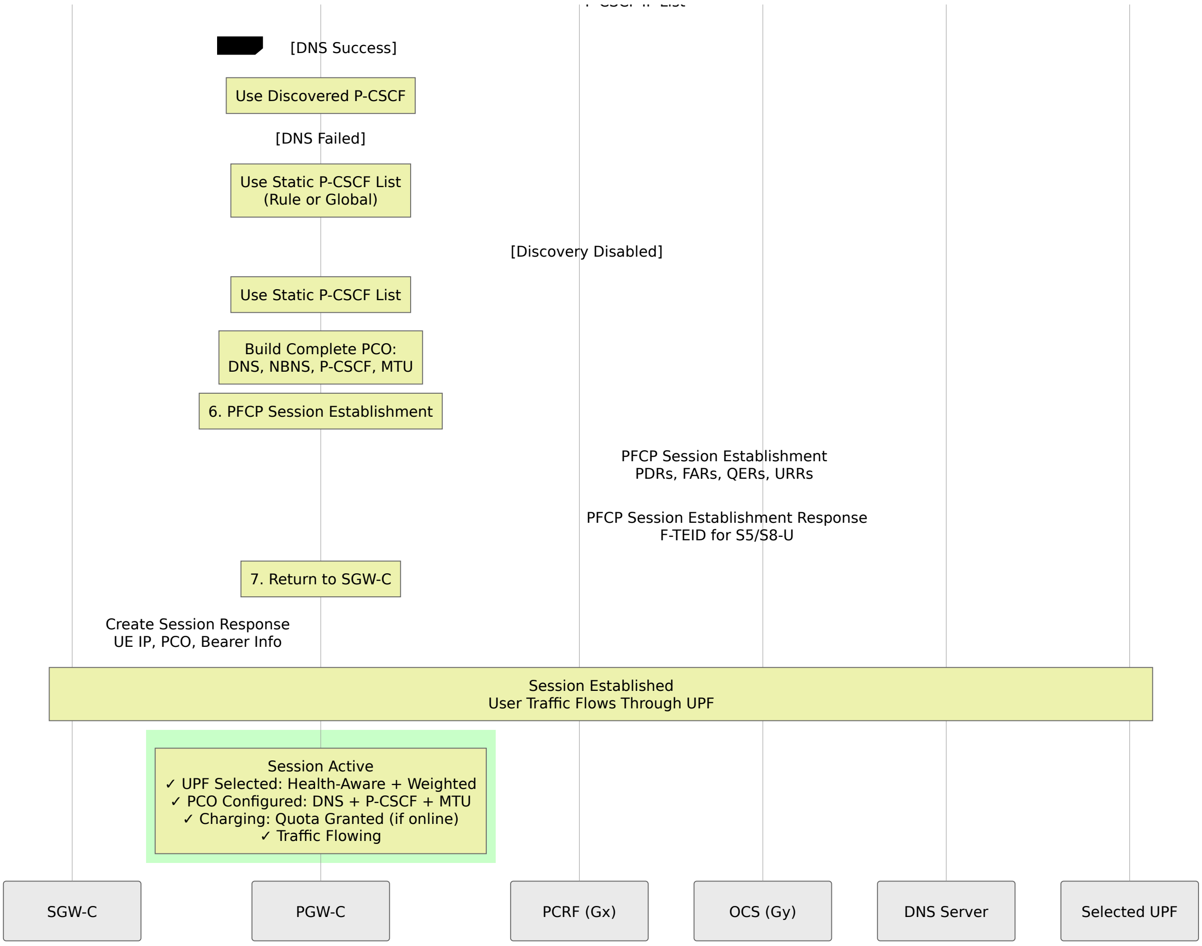
Selection Methods Comparison

Method	When to Use	Pros	Cons
UPF Pools	Production deployments	Load balancing, HA, flexible weights	Requires multiple UPFs
APN-Based	Service differentiation	Route IMS/Internet separately	Static configuration
IMSI-Based	Roaming scenarios	Geographic routing	Regex complexity
DNS-Based	MEC/Edge computing	Dynamic, location-aware	Requires DNS infrastructure
Fallback Pool	Safety net	Always have a UPF	May not be optimal
Dry-Run Mode	Testing configs	Safe testing	No real traffic

Complete Session Establishment Flow

This diagram shows the complete end-to-end flow of session establishment including UPF selection and PCO population:


SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
DNS Server
Selected UPF
Create Session Request
(IMSI, APN, ULI, etc.)
1. Extract Session Attributes
IMSI, APN, PLMN, TAI, ECGI
Allocate UE IP
from APN Pool
CCR-Initial (Gx)
Request PCC Rules
CCA-Initial (Gx)
PCC Rules + Charging Info
2. Check if Online Charging
Required (Rating-Group present?)
[Online Charging Required]
CCR-Initial (Gy)
Request Quota
CCA-Initial (Gy)
Grant Quota
3. UPF Selection Process
Evaluate Static Rules
by Priority (High→Low)
[Rule Match Found]
Rule Matched!
Get UPF Pool from Rule
[No Rule Match]
[DNS Enabled]
NAPTR Query
Based on ULI
UPF IP Address(es)
[DNS Disabled/Failed]
Use Fallback Pool
4. Filter to Healthy UPFs
Check PFCP Association + Heartbeats
Weighted Random Selection
Handle Active/Standby
Selected UPF Chosen!
5. PCO Population Process
[Rule Matched
with PCO
Override]
Use Rule PCO Override
+ Global PCO Fallback
[No PCO
Override]
Use Global PCO Config
[P-CSCF Discovery Enabled]
[Per-Rule Discovery FQDN]
Query p_cscf_discovery_fqdn
P-CSCF IP List
[Global Discovery Enabled]
Query Global P-CSCF DNS
P-CSCF IP List



Key Decision Points:

1. **UPF Selection Priority:**
   - Static Rules (Pattern Match) → DNS Discovery → Fallback Pool
   - Health filtering applied at all stages
   - Active/Standby logic for high availability
   - **See:** PFCP Interface for UPF communication details
2. **PCO Population Priority:**
   - Rule PCO Override → P-CSCF DNS Discovery → Global PCO Config
   - Per-field merging (rule overrides specific fields, global provides defaults)
   - **See:** PCO Configuration for detailed PCO parameters
3. **P-CSCF Discovery Priority:**
   - Per-Rule FQDN → Global DNS Discovery → Static Rule PCO → Global Static PCO

- **See:** P-CSCF Monitoring for discovery metrics and health tracking

4. **Charging Integration:**

   - PCRF determines if online charging required (Rating-Group + Online=1)
   - OCS grants quota before session establishment
   - PGW-C tracks quota and requests more via CCR-Update
   - **See:** Diameter Gx Interface and Diameter Gy Interface for charging details

Complete Configuration Example

Here's a complete example showing multi-pool UPF selection with automatic peer registration:

```
config :pgw_c,
  # PFCP Interface - All UPFs are auto-registered from
upf_selection
  sxb: %{
    local_ip_address: "127.0.0.20"
  },

  # UPF Selection Logic - All UPFs defined here are automatically
registered
  upf_selection: %{
    # DNS-based selection settings
    dns_enabled: false,
    dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],
    dns_suffix: "epc.3gppnetwork.org",
    dns_timeout_ms: 5000,

    # Static selection rules (evaluated in priority order)
    rules: [
      # Rule 1: IMS Traffic - Highest Priority
      %{
        name: "IMS Traffic",
        priority: 20,
        match_field: :apn,
        match_regex: "^ims",
        upf_pool: [
          %{remote_ip_address: "10.100.2.21", remote_port: 8805,
weight: 80},
          %{remote_ip_address: "10.100.2.22", remote_port: 8805,
weight: 20}
        ]
      },

      # Rule 2: Enterprise APN
      %{
        name: "Enterprise Traffic",
        priority: 15,
        match_field: :apn,
        match_regex: "^(enterprise|corporate)\.apn",
        upf_pool: [
          %{remote_ip_address: "10.100.3.21", remote_port: 8805,
weight: 100}
        ]
      },
```

```
      # Rule 3: Internet Traffic - Load Balanced
      %{
        name: "Internet Traffic",
        priority: 5,
        match_field: :apn,
        match_regex: "^internet",
        upf_pool: [
          %{remote_ip_address: "10.100.1.21", remote_port: 8805, weight: 33},
          %{remote_ip_address: "10.100.1.22", remote_port: 8805, weight: 33},
          %{remote_ip_address: "10.100.1.23", remote_port: 8805, weight: 34}
        ]
      }
    ],

    # Fallback pool - Used when no rules match and DNS fails
    fallback_pool: [
      %{remote_ip_address: "127.0.0.21", remote_port: 8805, weight: 100}
    ]
  }
```

Key Features

Current Format:

- □ **Automatic Registration**: All UPFs from `upf_selection` are automatically registered at startup
- □ **Centralized Configuration**: All UPF selection and peer configuration in one section
- □ **Required Pools**: All rules use `upf_pool` format (even for single UPF)
- □ **Structured Fallback**: Dedicated `fallback_pool` with weighted distribution
- □ **DNS Integration**: DNS settings alongside selection rules
- □ **Dynamic Registration**: DNS-discovered UPFs are automatically registered on first use

- □ **Health Monitoring**: All configured UPFs are monitored with 5-second heartbeats

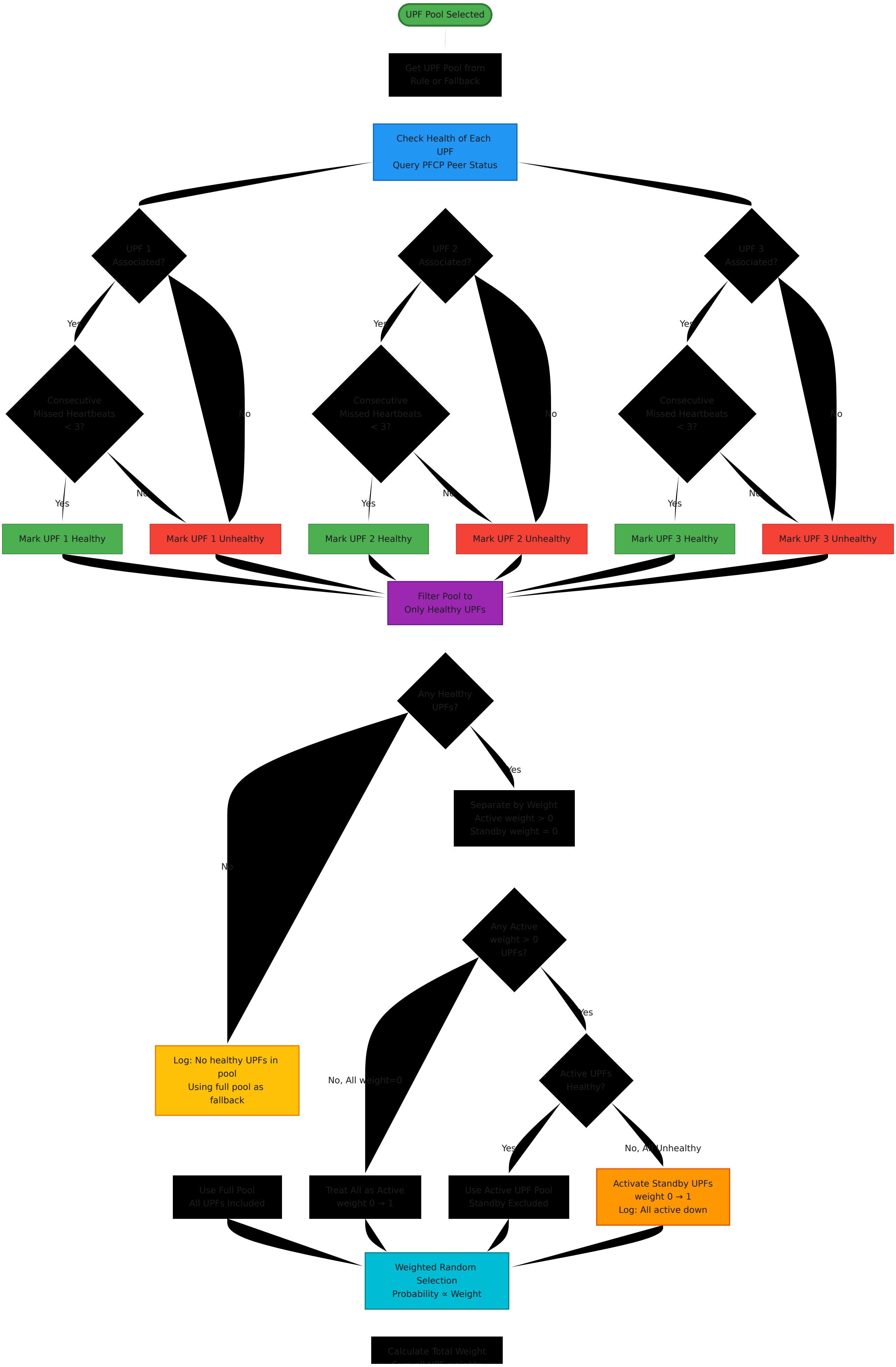
Migration from Previous Format:

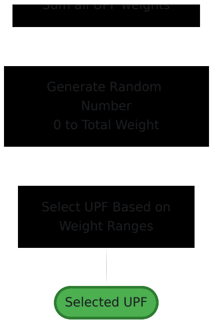
- Removed: `sxb.peer_list` field (no longer needed)
- Removed: `selection_list` embedded in peer configurations
- All UPF definitions now go in `upf_selection` rules and fallback pool

How UPF Pools Work:

1. **Health-Aware Selection**: Only healthy UPFs receive traffic
    - Healthy = PFCP association active + less than 3 consecutive missed heartbeats
    - Unhealthy UPFs are automatically filtered out
    - Falls back to all UPFs if none are healthy (fail-fast)
2. **Active/Standby Support**: Use `weight: 0` for standby UPFs
    - **Active UPFs** (weight > 0): Receive traffic when healthy
    - **Standby UPFs** (weight == 0): Only receive traffic when all active UPFs are down
    - Standby UPFs are treated as `weight: 1` when activated
3. **Weighted Random Selection**: Each session is randomly assigned to a healthy UPF based on weights
    - In the example above: 70% go to .21, 20% to .22, 10% to .23
    - Higher weight = more sessions assigned to that UPF
    - Equal weights = equal distribution
4. **Automatic Registration**: All UPFs in pools are automatically registered at startup
    - Auto-generated names: `"UPF-<ip>:<port>"`
    - Default settings: passive PFCP association, 5-second heartbeats
    - Immediate health tracking for all configured UPFs

Health-Aware Selection with Active/Standby


UPF Pool Selected
Check Health of Each UPF Query PFCP Peer Status
Yes
No
Mark UPF 1 Healthy
Mark UPF 1 Unhealthy
Mark UPF 2 Healthy
Mark UPF 2 Unhealthy
Mark UPF 3 Healthy
Mark UPF 3 Unhealthy
Filter Pool to Only Healthy UPFs
Log: No healthy UPFs in pool Using full pool as fallback
No, All weight=0
No, All Unhealthy
Activate Standby UPFs weight 0 → 1 Log: All active down
Weighted Random Selection Probability ∝ Weight



Weighted Random Selection Example:

```
Pool: [
  UPF-A: weight 50, healthy ✓
  UPF-B: weight 30, healthy ✓
  UPF-C: weight 20, healthy ✓
]

Total Weight: 50 + 30 + 20 = 100

Weight Ranges:
  UPF-A: 0-49   (50%)
  UPF-B: 50-79  (30%)
  UPF-C: 80-99  (20%)

Random number: 63 → Selects UPF-B
Random number: 15 → Selects UPF-A
Random number: 91 → Selects UPF-C
```

Active/Standby Failover Example:

```
Initial Pool: [
  UPF-A: weight 100, healthy ✓   (Active)
  UPF-B: weight 0, healthy ✓     (Standby)
]

Scenario 1: UPF-A Healthy
→ Use Active Pool: [UPF-A: 100]
→ All traffic to UPF-A

Scenario 2: UPF-A Fails
→ No active UPFs healthy
→ Activate Standby: [UPF-B: 1]
→ All traffic fails over to UPF-B
→ Log: "All active UPFs down, activating standby UPFs"

Scenario 3: Both Unhealthy
→ No healthy UPFs
→ Use full pool: [UPF-A: 100, UPF-B: 0]
→ Select with weights (attempt connection, may fail)
→ Log: "No healthy UPFs in pool, using full pool as fallback"
```

Common Weight Patterns:

```
# Equal distribution (25% each)
upf_pool: [
  %{remote_ip_address: "10.0.1.1", remote_port: 8805, weight: 1},
  %{remote_ip_address: "10.0.1.2", remote_port: 8805, weight: 1},
  %{remote_ip_address: "10.0.1.3", remote_port: 8805, weight: 1},
  %{remote_ip_address: "10.0.1.4", remote_port: 8805, weight: 1}
]

# Primary/backup load balanced (90% / 10%)
upf_pool: [
  %{remote_ip_address: "10.0.1.21", remote_port: 8805, weight:
90},
  %{remote_ip_address: "10.0.1.22", remote_port: 8805, weight: 10}
]

# Active/Standby (100% primary, 0% standby until primary fails)
upf_pool: [
  %{remote_ip_address: "10.0.1.21", remote_port: 8805, weight:
100},  # Active
  %{remote_ip_address: "10.0.1.22", remote_port: 8805, weight: 0}
# Standby (only when active down)
]

# Active with multiple standbys (load balanced when activated)
upf_pool: [
  %{remote_ip_address: "10.0.1.1", remote_port: 8805, weight:
100},   # Active
  %{remote_ip_address: "10.0.1.2", remote_port: 8805, weight: 0},
# Standby 1
  %{remote_ip_address: "10.0.1.3", remote_port: 8805, weight: 0}
# Standby 2
]
# Result: Active gets 100%. If active fails, standbys get 50/50%.

# A/B testing (50% / 50%)
upf_pool: [
  %{remote_ip_address: "10.0.1.100", remote_port: 8805, weight:
50},  # Old version
  %{remote_ip_address: "10.0.1.200", remote_port: 8805, weight:
50}   # New version
]
```

Use Cases:

- **Active/Standby Failover**: Use `weight: 0` for hot standby UPFs that only activate when primaries fail
- **Health-Aware HA**: Automatic failover when UPFs lose PFCP association or miss heartbeats
- **Horizontal Scaling**: Distribute load across multiple UPFs to increase capacity
- **High Availability**: Automatic distribution prevents single UPF overload
- **Gradual Rollouts**: Use weights for canary deployments (e.g., 95% old, 5% new)
- **Cost Optimization**: Route more traffic to higher-capacity UPFs
- **Geographic Distribution**: Balance sessions across edge UPFs

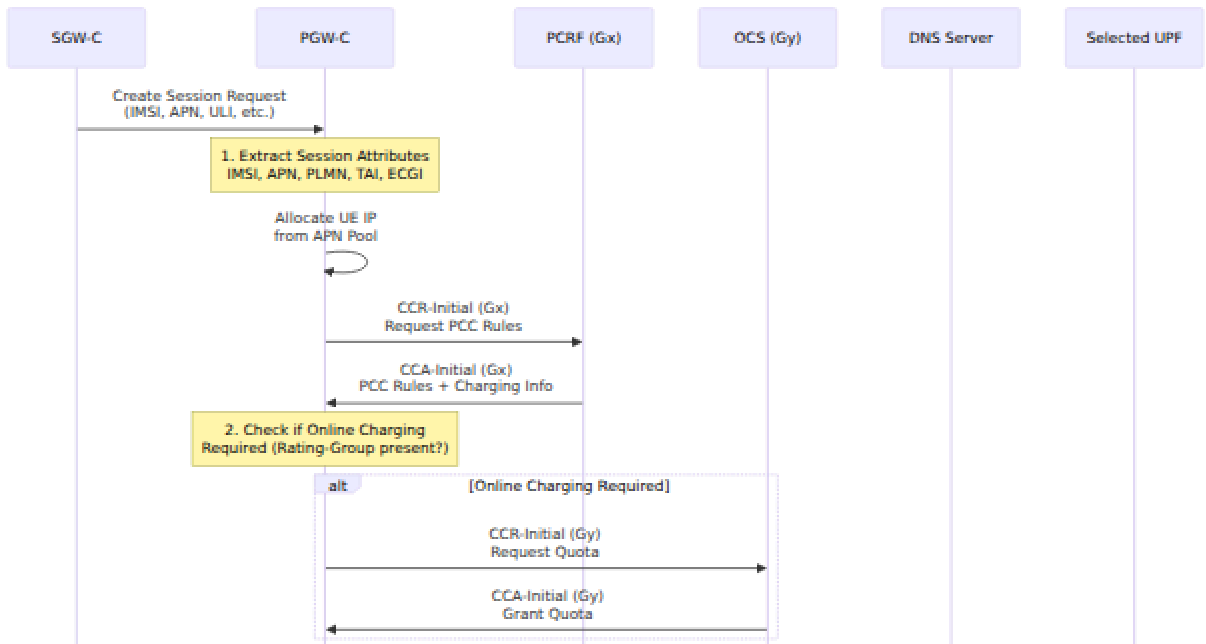
PCO (Protocol Configuration Options) Overrides:

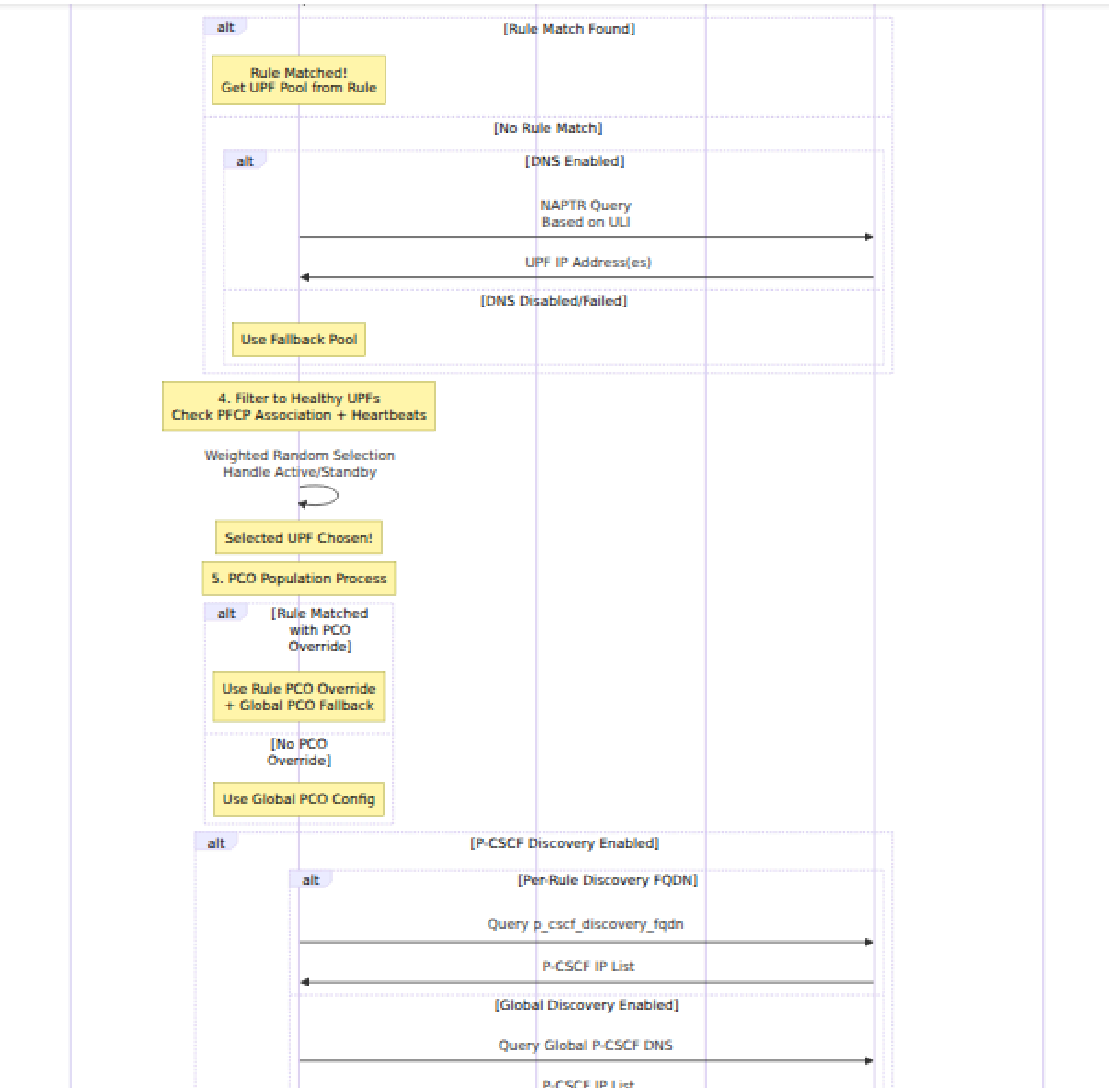
Each UPF selection rule can optionally specify custom PCO values that override the default PCO configuration for matching sessions. This allows different APNs or traffic types to receive different network parameters.

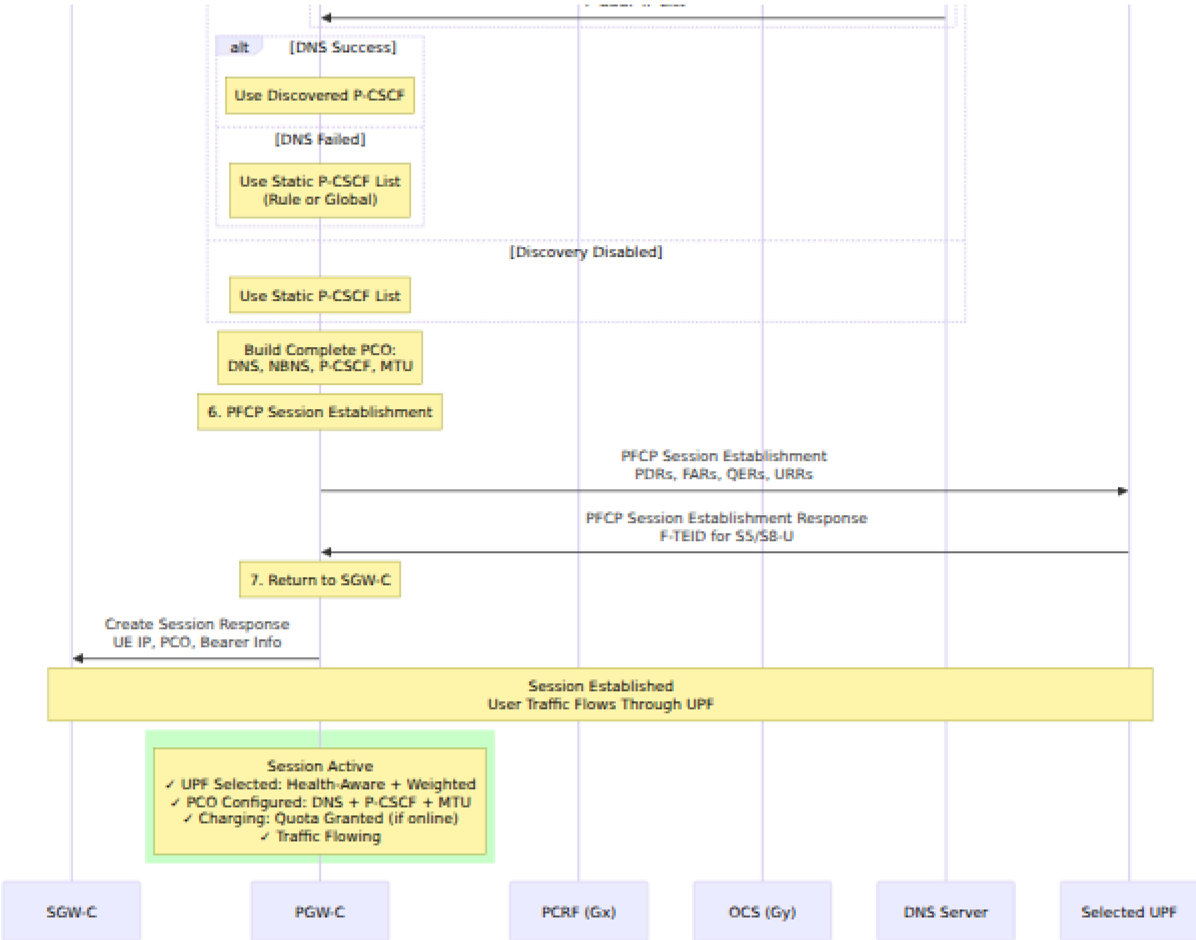
How PCO Overrides Work:

1. **Partial Overrides**: Only specify the PCO fields you want to override
2. **Default Fallback**: Unspecified fields use values from the main `pco` config
3. **Rule-Specific**: Each rule can have different PCO overrides
4. **Priority Merging**: Rule PCO takes priority over default PCO

PCO Population Hierarchy


SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
DNS Server
Selected UPF
Create Session Request
(IMSI, APN, ULI, etc.)
1. Extract Session Attributes
IMSI, APN, PLMN, TAI, ECGI
Allocate UE IP
from APN Pool
CCR-Initial (Gx)
Request PCC Rules
CCA-Initial (Gx)
PCC Rules + Charging Info
2. Check if Online Charging
Required (Rating-Group present?)
alt
[Online Charging Required]
CCR-Initial (Gy)
Request Quota
CCA-Initial (Gy)
Grant Quota


alt
[Rule Match Found]
Rule Matched!
Get UPF Pool from Rule
[No Rule Match]
alt
[DNS Enabled]
NAPTR Query
Based on ULI
UPF IP Address(es)
[DNS Disabled/Failed]
Use Fallback Pool
4. Filter to Healthy UPFs
Check PFCP Association + Heartbeats
Weighted Random Selection
Handle Active/Standby
Selected UPF Chosen!
5. PCO Population Process
alt
[Rule Matched
with PCO
Override]
Use Rule PCO Override
+ Global PCO Fallback
[No PCO
Override]
Use Global PCO Config
alt
[P-CSCF Discovery Enabled]
alt
[Per-Rule Discovery FQDN]
Query p_cscf_discovery_fqdn
P-CSCF IP List
[Global Discovery Enabled]
Query Global P-CSCF DNS



Priority Order for Each PCO Field:

1. **Rule PCO Override** (Highest Priority)
2. **P-CSCF DNS Discovery** (for P-CSCF addresses only)
3. **Global PCO Configuration** (Lowest Priority / Fallback)

Example: IMS Rule Overrides DNS, Enterprise Rule Overrides Everything

```
IMS Session (matched "IMS Traffic" rule):
├─ DNS Servers: FROM GLOBAL (not overridden in rule)
├─ P-CSCF: FROM DNS DISCOVERY (p_cscf_discovery_fqdn set in rule)
│  └─ Fallback: FROM RULE if DNS fails
└─ MTU: FROM GLOBAL (not overridden in rule)

Enterprise Session (matched "Enterprise Traffic" rule):
├─ DNS Servers: FROM RULE (192.168.1.10, 192.168.1.11)
├─ P-CSCF: FROM GLOBAL (not overridden in rule)
└─ MTU: FROM RULE (1500)

Default Session (no rule matched):
├─ DNS Servers: FROM GLOBAL
├─ P-CSCF: FROM GLOBAL or DNS if global discovery enabled
└─ MTU: FROM GLOBAL
```

Available PCO Override Fields:

- `primary_dns_server_address` - Primary DNS server IP (IPv4)
- `secondary_dns_server_address` - Secondary DNS server IP (IPv4)
- `primary_dns_server_ipv6_address` - Primary DNS server IP (IPv6, container 0x0003)
- `secondary_dns_server_ipv6_address` - Secondary DNS server IP (IPv6, container 0x0003)
- `primary_nbns_server_address` - Primary WINS server IP
- `secondary_nbns_server_address` - Secondary WINS server IP
- `p_cscf_ipv4_address_list` - List of IPv4 P-CSCF server IPs (for IMS, container 0x000C) - See PCO Configuration and P-CSCF Monitoring for dynamic P-CSCF discovery
- `p_cscf_ipv6_address_list` - List of IPv6 P-CSCF server IPs (for IMS, container 0x0001)
- `ipv4_link_mtu_size` - MTU size in bytes

P-CSCF Discovery Per Rule:

In addition to PCO overrides, UPF selection rules can specify dynamic P-CSCF discovery:

- `p_cscf_discovery_fqdn` - (String) FQDN for DNS-based P-CSCF discovery (e.g., `"pcscf.mnc001.mcc001.3gppnetwork.org"`)

When this parameter is set:

1. PGW-C performs DNS lookup for the specified FQDN during session establishment
2. DNS server returns list of P-CSCF IP addresses
3. Discovered P-CSCF addresses are sent to UE via PCO
4. If DNS lookup fails, falls back to `p_cscf_ipv4_address_list` from PCO override (if specified) or global PCO config
5. See P-CSCF Monitoring for monitoring discovery success/failure rates

This is particularly useful for:

- **IMS APNs** - Different IMS networks with different P-CSCF servers
- **Multi-tenant deployments** - Different enterprises with dedicated P-CSCF infrastructure
- **Geographic routing** - DNS returns closest P-CSCF based on UE location
- **High availability** - DNS automatically returns only healthy P-CSCF servers

Example: IMS Traffic with Custom P-CSCF:

```
rules: [
  %{
    name: "IMS Traffic",
    priority: 20,
    match_field: :apn,
    match_regex: "^ims",
    upf_pool: [
      %{remote_ip_address: "10.100.2.21", remote_port: 8805,
weight: 80},
      %{remote_ip_address: "10.100.2.22", remote_port: 8805,
weight: 20}
    ],
    # P-CSCF Discovery: Dynamically query DNS for P-CSCF addresses
    # DNS lookup returns current P-CSCF IPs based on this FQDN
    p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org",
    # IMS sessions get custom P-CSCF servers (used as fallback if
DNS fails)
    pco: %{
      p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"]
      # DNS, NBNS, MTU will use defaults from main pco config
    }
  }
]
```

Example: Enterprise Traffic with Custom DNS:

```
rules: [
  %{
    name: "Enterprise Traffic",
    priority: 15,
    match_field: :apn,
    match_regex: "^(enterprise|corporate)\.apn",
    upf_pool: [
      %{remote_ip_address: "10.100.3.21", remote_port: 8805,
weight: 100}
    ],
    # Enterprise sessions get corporate DNS and custom MTU
    pco: %{
      primary_dns_server_address: "192.168.1.10",
      secondary_dns_server_address: "192.168.1.11",
      ipv4_link_mtu_size: 1500
      # P-CSCF, NBNS will use defaults from main pco config
    }
  }
]
```

Example: Complete Override (All PCO Fields):

```
rules: [
  %{
    name: "IoT APN - Fully Custom",
    priority: 10,
    match_field: :apn,
    match_regex: "^iot\.m2m",
    upf_pool: [
      %{remote_ip_address: "10.100.5.21", remote_port: 8805,
weight: 100}
    ],
    # IoT sessions get completely custom PCO
    pco: %{
      primary_dns_server_address: "8.8.8.8",
      secondary_dns_server_address: "8.8.4.4",
      primary_nbns_server_address: "10.0.0.100",
      secondary_nbns_server_address: "10.0.0.101",
      p_cscf_ipv4_address_list: [],  # No P-CSCF for IoT
      ipv4_link_mtu_size: 1280       # Smaller MTU for constrained
devices
    }
  }
]
```

Use Cases:

- **IMS/VoLTE**: Provide carrier-specific P-CSCF servers for voice services
- **Enterprise APNs**: Route corporate traffic through company DNS servers
- **IoT/M2M**: Use public DNS and optimized MTU for low-bandwidth devices
- **Roaming**: Provide local DNS servers for visiting subscribers
- **Service Differentiation**: Different network parameters per service type

DNS-Based UPF Selection:

Enable dynamic UPF selection based on User Location Information (ULI) using DNS NAPTR queries. DNS settings are now configured within the `upf_selection` section.

Note: This provides geographic or topology-based UPF selection. See PFCP Interface for PFCP association setup with dynamically discovered UPFs and Session Management for session establishment flows.

```
upf_selection: %{
  # Enable DNS-based selection
  dns_enabled: true,

  # Location types to query in priority order
  dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],

  # DNS suffix for 3GPP NAPTR queries
  dns_suffix: "epc.3gppnetwork.org",

  # DNS query timeout in milliseconds
  dns_timeout_ms: 5000,

  # ... rules and fallback_pool ...
}
```

DNS-based selection works as follows:

1. **Priority**: DNS selection is used only when **NO static rules match** (lower priority)
2. **Query Generation**: Builds DNS NAPTR queries based on UE location:
   - ECGI query: `eci-<hex>.ecgi.epc.mnc<MNC>.mcc<MCC>.epc.3gppnetwork.org`
   - TAI query: `tac-lb<hex>.tac-hb<hex>.tac.epc.mnc<MNC>.mcc<MCC>.epc.3gppnetwork.org`
   - RAI, SAI, CGI queries follow similar 3GPP TS 23.003 format
3. **Fallback Hierarchy**: Tries each location type in priority order until a match is found
4. **Peer Matching**: DNS results are filtered against configured peer list
5. **Selection**: Chooses matching peer (currently first match, load-based selection coming soon)

Example DNS Records (configure on your DNS server):

```
; NAPTR record for TAC 100 in PLMN 001-001
tac-lb64.tac-hb00.tac.epc.mnc001.mcc001.epc.3gppnetwork.org IN
NAPTR 10 50 "a" "x-3gpp-upf:x-sxb" "" upf-edge-1.example.com.

; A record for the UPF
upf-edge-1.example.com IN A 10.100.1.21
```

Use Cases:

- **Multi-access Edge Computing (MEC)**: Route sessions to geographically closest edge UPFs
- **Dynamic UPF Discovery**: Add/remove UPFs without reconfiguring PGW-C
- **Load Balancing**: Distribute load across UPFs based on location
- **Network Slicing**: Route different slices to different UPFs per location

UPF Health Monitoring

Automatic Health-Aware Selection: The PGW-C continuously monitors the health of all UPFs and automatically excludes unhealthy UPFs from selection.

Health Check Criteria

A UPF is considered **healthy** when ALL of the following conditions are met:

1. **PFCP Association Active**: The UPF has an established PFCP association
2. **Heartbeat Responsiveness**: Less than 3 consecutive missed heartbeats
3. **Process Alive**: The UPF peer GenServer process is running

A UPF is considered **unhealthy** if ANY of the following are true:

- PFCP association is not established (`associated: false`)
- 3 or more consecutive heartbeat timeouts
- UPF peer process has crashed or is unresponsive

Monitoring Mechanism

For Configured UPFs (in `upf_selection`):

- Health tracking starts immediately at boot
- PFCP association is monitored continuously
- Heartbeats are sent every 5 seconds
- `missed_heartbeats_consecutive` counter tracks consecutive failures
- All UPFs from rules and fallback pool are automatically registered

For DNS-Discovered UPFs (dynamic registration):

- Assumed healthy until first session attempt
- Registered automatically on first use
- Health tracking begins after registration

Selection Behavior

Active/Standby Mode (when using `weight: 0`):

1. Filter to only healthy UPFs
2. Separate into **active** (weight > 0) and **standby** (weight == 0)
3. Use active UPFs if any are healthy
4. Activate standby UPFs (treat as weight 1) if all active are unhealthy
5. Fall back to full pool if no healthy UPFs exist

Load-Balanced Mode (all weight > 0):

1. Filter to only healthy UPFs
2. Perform weighted random selection among healthy UPFs
3. Fall back to full pool if no healthy UPFs exist

Logging:

```
[debug] Using active UPF pool (2/3 healthy UPFs, 1 standby)
[info] All active UPFs down, activating standby UPFs (1 standby UPFs, treating weight 0 as 1)
[warning] No healthy UPFs in pool (3 total), using full pool as fallback
```

Checking UPF Health

Programmatically:

```
# Check if a specific UPF is healthy
iex> PGW_C.PFCP_Node.is_peer_healthy?({10, 100, 1, 21})
true

# Get detailed health information
iex> PGW_C.PFCP_Node.get_peer_health({10, 100, 1, 21})
%{
  associated: true,
  missed_heartbeats: 0,
  healthy: true,
  registered: true
}
```

Via OAM API:

- Query the `/api/upf_selection` endpoint: `curl -k https://localhost:8443/api/upf_selection`
- Returns real-time health status for all UPFs in each pool
- Status values: Active-UP, Standby-Ready, Active-DOWN, Not Associated
- Role values: ACTIVE (weight > 0), STANDBY (weight == 0), DYNAMIC (DNS-discovered, not in config)
- Heartbeat miss counter reported for associated UPFs

Health Monitoring Best Practices

1. **Configure UPFs in upf_selection**: All UPFs in rules and fallback pools are automatically monitored

```
upf_selection: %{
  rules: [
    %{
      name: "Internet Traffic",
      priority: 10,
      match_field: :apn,
      match_regex: "^internet",
      upf_pool: [
        %{remote_ip_address: "10.100.1.21", remote_port: 8805,
weight: 100}
      ]
    }
  ],
  fallback_pool: [
    %{remote_ip_address: "10.100.2.21", remote_port: 8805,
weight: 100}
  ]
}
# All UPFs automatically get:
# - 5-second heartbeats
# - Health monitoring from startup
# - Auto-generated names
```

2. **Use standby UPFs**: Configure hot standbys with `weight: 0` for automatic failover

```
upf_pool: [
  %{remote_ip_address: "10.1.1.1", remote_port: 8805, weight:
100},  # Active
  %{remote_ip_address: "10.1.1.2", remote_port: 8805, weight:
0}     # Standby
]
```

3. **Monitor via OAM API**: Regularly poll the `/api/upf_selection` endpoint to check UPF health status

4. **Heartbeat monitoring**: The system uses a fixed threshold of 3 consecutive missed heartbeats to determine peer hea.

Dynamic UPF Registration

Feature: The PGW-C automatically registers and monitors UPFs discovered through DNS, even if they aren't in the `upf_selection` configuration.

How It Works

When any selection method (static rules, pools, or DNS) returns a UPF that's not already registered, the system automatically:

1. **Creates a PFCP Peer**: Generates a default peer configuration for the unknown UPF
2. **Initiates PFCP Association**: Attempts to establish a PFCP association with the UPF
3. **Registers in Peer Registry**: Adds the UPF to the internal peer tracking system
4. **Starts Heartbeat Monitoring**: Begins periodic heartbeat exchanges (10-second intervals)
5. **Tracks Liveness**: Monitors the UPF for failures and recovery

Default Configuration for Dynamic UPFs

When a UPF is dynamically registered, it receives the following default configuration:

```
%{
  name: "Dynamic-UPF-<IP>",              # e.g., "Dynamic-UPF-10-100-1-21"
  remote_ip_address: <discovered_ip>,     # IP from DNS or selection
  remote_port: 8805,                      # Standard PFCP port (overridable)
  initiate_pfcp_association_setup: true,  # PGW-C initiates association
  heartbeat_period_ms: 10_000             # 10-second heartbeat interval
}
```

> **Note**: Dynamic UPFs are registered purely for association management. They are used as targets in `upf_selection` rules, not as sources of selection logic.

Example: DNS Returns Unknown UPF

```
# DNS query returns: upf-edge-2.example.com -> 10.200.5.99
# This UPF is NOT in your upf_selection configuration

# Dynamic registration flow:
# 1. System detects unknown UPF 10.200.5.99
# 2. Logs: "UPF {10, 200, 5, 99} not pre-configured, attempting
dynamic registration..."
# 3. Sends PFCP Association Setup Request to 10.200.5.99:8805
# 4. If UPF responds: Association established, session continues
normally
# 5. If UPF doesn't respond: Session fails gracefully with clear
error message
```

Benefits

□ **True Dynamic Discovery**: DNS-based UPF selection now works without pre-configuration □ **Automatic Scaling**: Add UPFs to your network without restarting PGW-C □ **Graceful Degradation**: If association fails, sessions fail cleanly (no crashes) □ **Backwards Compatible**: Pre-configured UPFs continue to work exactly as before □ **Full Monitoring**: Dynamic UPFs get the same heartbeat monitoring as static peers

Failure Handling

If a dynamically discovered UPF fails to respond to PFCP Association Setup:

```
[error] PFCP Association Setup failed for dynamic UPF {10, 200, 5,
99}: :timeout
[error] Failed to dynamically register UPF {10, 200, 5, 99}:
:timeout.
        Session creation will fail. Consider adding this UPF to
the upf_selection configuration.
```

The session creation will fail, but the PGW-C remains stable and continues processing other sessions.

When to Pre-Configure vs. Dynamic Registration

Scenario	Recommendation
Production Core UPFs	Pre-configure in `upf_selection` (explicit configuration, monitored from startup)
DNS-Discovered Edge UPFs	Use dynamic registration (scales automatically with infrastructure)
Test/Development UPFs	Either approach works (dynamic is more convenient)
Mission-Critical UPFs	Pre-configure in `upf_selection` (ensures monitoring from startup)
Ephemeral/Auto-Scaled UPFs	Use dynamic registration (UPFs come and go dynamically)

Monitoring Dynamic UPFs

Dynamic UPFs appear in logs with their auto-generated names:

```
[info] Creating dynamic PFCP peer configuration for Dynamic-UPF-10-200-5-99 ({10, 200, 5, 99}:8805)
[info] Dynamic UPF peer Dynamic-UPF-10-200-5-99 registered successfully with PID #PID<0.1234.0>
```

You can query the peer registry to see all registered peers (both static and dynamic):

```
# Get all registered peers
PGW_C.PFCP_Node.registered_peer_count()

# Check if a specific UPF is registered
PGW_C.PFCP_Node.get_peer({10, 200, 5, 99})
# Returns: {:ok, #PID<0.1234.0>} if registered, :error otherwise
```

Custom Port for Dynamic UPFs

If your UPFs use a non-standard PFCP port, you can manually trigger registration:

```
# Register UPF at custom port
PGW_C.PFCP_Node.register_dynamic_peer({10, 200, 5, 99}, 9999)
```

However, DNS-based selection and automatic registration always use port 8805 (standard PFCP port).

UPF Selection Dry-Run Mode:

Test and validate your UPF selection configuration without affecting real sessions:

```
config :pgw_c,
  # Enable dry-run mode for testing (disabled by default)
  upf_selection_dry_run: true
```

When dry-run mode is enabled:

1. **No Real Assignment**: Sessions are not actually assigned to UPFs
2. **Detailed Logging**: Selection decisions are logged with full details
3. **Error Return**: `assign_sxb_peer/1` returns `{:error, :dry_run_mode}`
4. **Session Prevention**: Returning an error prevents session creation
5. **Both Methods**: Works with both static rules and DNS-based selection

Log Output Example:

```
[warning] ⚠️  UPF SELECTION DRY-RUN MODE ENABLED - No actual
assignment will occur

[info] 🔍 DRY-RUN: Static rule matched
  Method: Static Rule
  Match Field: :apn
  Match Regex: ~r/^internet\./
  Priority: 10
  Selected UPF: 10.0.1.21:8805

[warning] 🚫 DRY-RUN: Would assign UPF but skipping actual
assignment
  Session IMSI: 001001000000670
  Session APN: internet.apn
  Selection Method: static
  Would Link To: 10.0.1.21:8805
  ⚠️  Returning error to prevent session creation
```

Testing via LiveView UI:

The UPF Selection LiveView page (`/api/upf_selection`) includes an interactive testing interface:

1. Navigate to the UPF Selection page in the web panel
2. Scroll to the "Test UPF Selection" section
3. Enter test session attributes:
    - IMSI (e.g., `001001000000670`)
    - APN (e.g., `internet.apn`)
    - Serving Network PLMN ID (e.g., `001001`)
4. Click "Test Selection" to simulate the selection
5. View detailed results showing:
    - Which rule matched (or if DNS would be used)
    - Selected UPF and peer name
    - Match field and pattern details
    - Rule priority

The LiveView testing interface simulates the selection logic **without requiring dry-run mode** to be enabled globally, making it safe to test in production environments without affecting real traffic.

Use Cases:

- **Configuration Testing**: Validate routing rules before deploying to production
- **Troubleshooting**: Understand why specific sessions route to specific UPFs
- **Training**: Demonstrate UPF selection logic to operations teams
- **Development**: Test new selection rules during development

Match Fields:

- `:imsi` - International Mobile Subscriber Identity
- `:apn` - Access Point Name (APN/DNN)
- `:serving_network_plmn_id` - Serving network PLMN ID
- `:sgw_ip_address` - SGW IP address
- `:uli_tai_plmn_id` - Tracking Area PLMN ID
- `:uli_ecgi_plmn_id` - E-UTRAN Cell PLMN ID

Heartbeat Tuning:

```
# Aggressive (detect failures quickly)
heartbeat_period_ms: 2_000  # 2 seconds

# Standard (recommended)
heartbeat_period_ms: 5_000  # 5 seconds

# Relaxed (high-latency networks)
heartbeat_period_ms: 10_000  # 10 seconds
```

See: PFCP Interface Documentation

UE IP Pool Configuration

Purpose

Configure IP address pools for UE allocation, organized by APN.

Configuration Block

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # APN "internet" pools
      "internet" => [
        "100.64.0.0/20"  # 4094 usable IPs
      ],

      # APN "ims" pools
      "ims" => [
        "100.64.16.0/22"  # 1022 usable IPs
      ],

      # Default pool for unknown APNs
      default: [
        "42.42.42.0/24"  # 254 usable IPs
      ]
    }
  }
```

Parameters

Parameter	Type	Required	Description
`subnet_map`	Map	Yes	Maps APN names to subnet lists
`default`	List (Subnets)	Yes	Fallback pool for unknown APNs

Subnet Format

CIDR Notation: `<network_address>/<prefix_length>`

Usable IP Calculation:

CIDR	Total IPs	Usable IPs	Example Range
/24	256	254	100.64.1.1 - 100.64.1.254
/23	512	510	100.64.0.1 - 100.64.1.254
/22	1024	1022	100.64.0.1 - 100.64.3.254
/21	2048	2046	100.64.0.1 - 100.64.7.254
/20	4096	4094	100.64.0.1 - 100.64.15.254
/16	65536	65534	100.64.0.1 - 100.64.255.254

Examples

Simple Configuration:

```
ue: %{
  subnet_map: %{
    "internet" => ["100.64.1.0/24"],
    default: ["42.42.42.0/24"]
  }
}
```

Production Configuration:

```
ue: %{
  subnet_map: %{
    # General internet - large pool
    "internet" => [
      "100.64.0.0/18"  # 16,382 IPs
    ],

    # IMS (VoLTE)
    "ims" => [
      "100.64.64.0/22"  # 1,022 IPs
    ],

    # Enterprise APN
    "enterprise.corp" => [
      "10.100.0.0/16"  # 65,534 IPs
    ],

    # IoT devices
    "iot.m2m" => [
      "100.64.72.0/20"  # 4,094 IPs
    ],

    # Default fallback
    default: [
      "42.42.42.0/24"  # 254 IPs
    ]
  }
}
```

Load Balancing (Multiple Subnets per APN):

```
ue: %{
  subnet_map: %{
    "internet" => [
      "100.64.0.0/22",   # 1022 IPs
      "100.64.4.0/22",   # 1022 IPs
      "100.64.8.0/22",   # 1022 IPs
      "100.64.12.0/22"   # 1022 IPs
    ],
    # Total: 4088 IPs, randomly distributed
    default: ["42.42.42.0/24"]
  }
}
```

Regex Pattern Matching:

For wildcard APN matching, use keys starting with `^` to indicate regex patterns:

```
ue: %{
  subnet_map: %{
    # Regex: APNs starting with "ims" (matches "ims", "ims.apn",
"ims.something")
    "^ims" => [
      "100.64.10.0/24"
    ],

    # Regex: APNs starting with "m2m." (matches "m2m.test",
"m2m.prod")
    "^m2m\." => [
      "100.64.20.0/24"
    ],

    # Exact match only
    "enterprise.corp" => [
      "10.100.0.0/16"
    ],

    default: ["42.42.42.0/24"]
  }
}
```

Pattern Rules:

- Keys starting with `^` are regex patterns
- Keys without `^` are exact matches (backwards compatible)
- Regex patterns are compiled at config parse time
- First matching pattern wins
- Backslashes must be escaped (`\` for `\`)

Common Patterns:

- **Starts with:** `"^ims"` matches `ims`, `ims.apn`, `ims.foo.bar`
- **Ends with:** `"^.*\.corp$"` matches `foo.corp`, `bar.corp`
- **Contains:** `"^.*test.*"` matches `test`, `foo.test.bar`

IPv6 Support:

```
ue: %{
  subnet_map: %{
    # IPv4 pools
    "internet" => [
      "100.64.0.0/20"
    ],

    # IPv6 pools (prefix delegation)
    "internet.ipv6" => [
      "2001:db8:1::/48"
    ],

    default: [
      "42.42.42.0/24"
    ]
  }
}
```

Recommended IP Ranges

RFC 6598 (Carrier-Grade NAT):

- **Range:** `100.64.0.0/10`
- **Size:** ~4 million IPs

- **Purpose:** Designed for service provider NAT

Private IP Ranges (RFC 1918):

- `10.0.0.0/8` - 16 million IPs
- `172.16.0.0/12` - 1 million IPs
- `192.168.0.0/16` - 65,534 IPs

See: UE IP Pool Allocation Documentation

PCO Configuration

Purpose

Configure Protocol Configuration Options (PCO) sent to UE.

Configuration Block

```
config :pgw_c,
  pco: %{
    # DNS servers
    primary_dns_server_address: "8.8.8.8",
    secondary_dns_server_address: "8.8.4.4",

    # IPv6 DNS servers (optional, container 0x0003)
    primary_dns_server_ipv6_address: nil,
    secondary_dns_server_ipv6_address: nil,

    # NBNS servers (optional, for Windows devices)
    primary_nbns_server_address: nil,
    secondary_nbns_server_address: nil,

    # P-CSCF addresses for IMS
    p_cscf_ipv4_address_list: ["10.0.0.50", "10.0.0.51"],   #
container 0x000C
    p_cscf_ipv6_address_list: [],                           #
container 0x0001

    # IPv4 MTU size
    ipv4_link_mtu_size: 1400
  }
```

Parameters

Parameter	Type	Default	Description
`primary_dns_server_address`	String (IPv4)	Required	Primary DNS server
`secondary_dns_server_address`	String (IPv4)	Optional	Secondary DNS server
`primary_dns_server_ipv6_address`	String (IPv6)	`nil`	Primary IPv6 DNS server (container 0x0003)
`secondary_dns_server_ipv6_address`	String (IPv6)	`nil`	Secondary IPv6 DNS server (container 0x0003)
`primary_nbns_server_address`	String (IPv4)	`nil`	Primary NBNS (NetBIOS) server
`secondary_nbns_server_address`	String (IPv4)	`nil`	Secondary NBNS server
`p_cscf_ipv4_address_list`	List (IPv4)	`[]`	IPv4 P-CSCF addresses for IMS (container 0x000C)
`p_cscf_ipv6_address_list`	List (IPv6)	`[]`	IPv6 P-CSCF addresses

Parameter	Type	Default	Description
			for IMS (container 0x0001)
`ipv4_link_mtu_size`	Integer	`1400`	Maximum Transmission Unit size

Examples

Public DNS (Google):

```
pco: %{
  primary_dns_server_address: "8.8.8.8",
  secondary_dns_server_address: "8.8.4.4",
  ipv4_link_mtu_size: 1400
}
```

Private DNS:

```
pco: %{
  primary_dns_server_address: "10.0.0.10",
  secondary_dns_server_address: "10.0.0.11",
  ipv4_link_mtu_size: 1400
}
```

IMS Configuration:

```
pco: %{
  primary_dns_server_address: "10.0.0.10",
  secondary_dns_server_address: "10.0.0.11",

  # P-CSCF for IMS/VoLTE
  p_cscf_ipv4_address_list: [
    "10.0.0.50",  # Primary P-CSCF
    "10.0.0.51"   # Secondary P-CSCF
  ],

  ipv4_link_mtu_size: 1400
}
```

NBNS (Windows Compatibility):

```
pco: %{
  primary_dns_server_address: "10.0.0.10",
  secondary_dns_server_address: "10.0.0.11",
  primary_nbns_server_address: "10.0.0.20",
  secondary_nbns_server_address: "10.0.0.21",
  ipv4_link_mtu_size: 1400
}
```

MTU Tuning:

```
# Standard Ethernet
ipv4_link_mtu_size: 1500

# Reduced for tunneling overhead
ipv4_link_mtu_size: 1400

# Jumbo frames (if supported)
ipv4_link_mtu_size: 9000
```

See: PCO Configuration Documentation

OAM API Configuration

Purpose

Configure the OAM REST API endpoints for managing and monitoring OmniPGW. All monitoring and management is performed through this API.

> **Note:** API configuration is only active in non-test environments. The configuration is automatically skipped when `config_env()` is `:test`, `:test_mock`, or `:test_impl`.

REST API Configuration

```
config :api_ex,
  api: %{
    # Network settings
    port: 8443,
    listen_ip: "0.0.0.0",

    # API metadata
    product_name: "PGW-C",
    title: "API - PGW-C",
    hostname: "localhost",

    # TLS settings
    enable_tls: true,
    tls_cert_path: "priv/cert/omnitouch.crt",
    tls_key_path: "priv/cert/omnitouch.pem",

    # Route definitions
    routes: [
      %{
        path: "/status",
        module: ApiEx.Api.StatusController,
        actions: [:index]
      }
    ]
  }
```

API Parameters

Parameter	Type	Default	Description
`port`	Integer	`8443`	HTTPS port for REST API
`listen_ip`	String (IP)	`"0.0.0.0"`	API listen address (0.0.0.0 = all interfaces)
`product_name`	String	`"PGW-C"`	Product name (for API metadata)
`title`	String	`"API - PGW-C"`	API documentation title
`hostname`	String	`"localhost"`	API server hostname
`enable_tls`	Boolean	`true`	Enable HTTPS for API
`tls_cert_path`	String (Path)	Required	Path to SSL/TLS certificate file
`tls_key_path`	String (Path)	Required	Path to SSL/TLS private key file
`routes`	List (Maps)	`[]`	API route definitions

Route Definition Format

Each route in the `routes` list is a map with:

Field	Type	Description	Example
`path`	String	URL path for the route	`"/status"`
`module`	Module	Controller module handling requests	`ApiEx.Api.StatusController`
`actions`	List (Atoms)	Allowed controller actions	`[:index, :show]`

Examples

Default Configuration (Development):

```
# API on localhost:8443
config :api_ex,
  api: %{
    port: 8443,
    listen_ip: "127.0.0.1",  # localhost only
    hostname: "localhost",
    enable_tls: true,
    tls_cert_path: "priv/cert/dev-cert.crt",
    tls_key_path: "priv/cert/dev-key.pem",
    routes: [
      %{path: "/status", module: ApiEx.Api.StatusController,
actions: [:index]}
    ]
  }
```

Production Configuration:

```
# API on management interface
config :api_ex,
  api: %{
    port: 8443,
    listen_ip: "10.0.0.20",  # Management network
    product_name: "OmniPGW-C",
    hostname: "pgw-c-api.example.com",
    enable_tls: true,
    tls_cert_path: "/etc/ssl/certs/pgw-c-api.crt",
    tls_key_path: "/etc/ssl/private/pgw-c-api.key",
    routes: [
      %{path: "/status", module: ApiEx.Api.StatusController,
actions: [:index]},
      %{path: "/sessions", module: ApiEx.Api.SessionController,
actions: [:index, :show]}
    ]
  }
```

Accessing the API

Query an endpoint:

```
# Status endpoint (self-signed cert, development)
curl -k https://localhost:8443/api/status

# With a trusted certificate (production)
curl https://pgw-c-api.example.com:8443/api/status
```

All routes are served under the `/api` prefix. Available endpoints include `/api/status`, `/api/sessions`, `/api/upf`, `/api/diameter`, `/api/charging`, `/api/ip_pools`, `/api/session_history`, `/api/topology`, `/api/ue_search` (POST), `/api/pcscf_monitor`, and `/api/upf_selection`.

Interactive API docs (Swagger UI):

```
https://localhost:8443/api/docs
```

TLS Certificate Setup

Generate Self-Signed Certificate (Development):

```
# Generate private key and certificate
openssl req -x509 -newkey rsa:4096 -keyout priv/cert/omnitouch.pem
\
  -out priv/cert/omnitouch.crt -days 365 -nodes \
  -subj "/CN=localhost"
```

Production Certificate:

For production, use certificates from a trusted Certificate Authority (CA):

- Let's Encrypt (free, automated)
- Commercial CA (DigiCert, GlobalSign, etc.)
- Internal CA for enterprise deployments

Security Considerations

1. **Always use TLS in production** - Set `enable_tls: true`
2. **Restrict listen_ip** - Use specific IP addresses in production (not 0.0.0.0)
3. **Use valid certificates** - Avoid self-signed certs in production
4. **Firewall protection** - Restrict access to the management port (8443)
5. **Strong key files** - Use 4096-bit RSA or equivalent
6. **Regular updates** - Rotate certificates before expiration

Troubleshooting

Issue: Cannot access the API

```
# Check if port is listening
netstat -tulpn | grep 8443

# Check certificate files exist
ls -la priv/cert/

# Check logs for startup errors
tail -f /var/log/pgw_c/application.log
```

Issue: SSL/TLS errors

- Verify certificate and key paths are correct
- Ensure certificate matches the hostname
- Check certificate expiration: `openssl x509 -in cert.crt -text -noout`
- Verify key file permissions (should be readable by PGW-C process)

See: Monitoring Guide for OAM API usage details

Complete Example

Production-Ready Configuration

```
# config/runtime.exs
import Config

# Logger configuration
config :logger, level: :info

config :pgw_c,
  # Metrics (Prometheus)
  metrics: %{
    enabled: true,
    ip_address: "10.0.0.20",  # Management network
    port: 9090,
    registry_poll_period_ms: 5_000
  },

  # Diameter/Gx (PCRF interface)
  diameter: %{
    listen_ip: "0.0.0.0",
    host: "omnipgw.epc.mnc001.mcc001.3gppnetwork.org",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    peer_list: [
      %{
        host: "pcrf-primary.epc.mnc001.mcc001.3gppnetwork.org",
        realm: "epc.mnc001.mcc001.3gppnetwork.org",
        ip: "10.0.1.30",
        initiate_connection: true
      },
      %{
        host: "pcrf-backup.epc.mnc001.mcc001.3gppnetwork.org",
        realm: "epc.mnc001.mcc001.3gppnetwork.org",
        ip: "10.0.2.30",
        initiate_connection: true
      }
    ]
  },

  # S5/S8 (SGW-C interface)
  s5s8: %{
```

```
    local_ipv4_address: "10.0.0.20"
  },

  # Sxb/PFCP (PGW-U interface)
  sxb: %{
    local_ip_address: "10.0.0.20"
  },

  # UPF Selection
  upf_selection: %{
    rules: [
      %{
        name: "Default Internet",
        priority: 1,
        match_field: :apn,
        match_regex: ~r/^internet/,
        upf_pool: [
          %{remote_ip_address: "10.0.0.21", remote_port: 8805, weight: 100},
          %{remote_ip_address: "10.0.0.22", remote_port: 8805, weight: 0}  # Standby
        ]
      }
    ],
    fallback_pool: [
      %{remote_ip_address: "10.0.0.21", remote_port: 8805, weight: 100}
    ]
  },

  # UE IP Pools
  ue: %{
    subnet_map: %{
      "internet" => [
        "100.64.0.0/18"  # 16,382 IPs
      ],
      "ims" => [
        "100.64.64.0/22"  # 1,022 IPs
      ],
      "enterprise.corp" => [
        "10.100.0.0/16"  # 65,534 IPs
      ],
      default: [
        "100.64.127.0/24"  # 254 IPs
```

```
      ]
    }
  },

  # Protocol Configuration Options
  pco: %{
    primary_dns_server_address: "8.8.8.8",
    secondary_dns_server_address: "8.8.4.4",
    p_cscf_ipv4_address_list: ["10.0.0.50", "10.0.0.51"],
    ipv4_link_mtu_size: 1400
  }
```

Configuration Validation

Startup Validation

OmniPGW validates configuration at startup. Check logs:

```
[info] Loading configuration from runtime.exs
[info] Validating configuration...
[info] Configuration valid
[info] Starting OmniPGW...
```

Common Validation Errors

Invalid IP Address:

```
[error] Invalid IP address in s5s8.local_ipv4_address: "10.0.0"
```

Missing Required Field:

```
[error] Missing required configuration: sxb.local_ip_address
```

Invalid CIDR:

```
[error] Invalid subnet in ue.subnet_map: "100.64.1.0/33"
```

Invalid Diameter Identity:

```
[error] Diameter host must be FQDN, not IP: "10.0.0.20"
```

Configuration Testing

Test configuration without starting:

```
# Validate syntax
mix compile

# Check configuration loading
iex -S mix
iex> Application.get_env(:pgw_c, :metrics)
```

Environment-Specific Configuration

Development

```
# config/dev.exs
import Config

config :logger, level: :debug

config :pgw_c,
  metrics: %{
    enabled: true,
    ip_address: "127.0.0.1",
    port: 42069,
    registry_poll_period_ms: 10_000
  },
  diameter: %{
    listen_ip: "0.0.0.0",
    host: "omnipgw-dev.local",
    realm: "local",
    peer_list: [
      %{
        host: "pcrf-dev.local",
        realm: "local",
        ip: "127.0.0.1",
        initiate_connection: true
      }
    ]
  },
  s5s8: %{
    local_ipv4_address: "127.0.0.10"
  },
  sxb: %{
    local_ip_address: "127.0.0.20"
  },
  upf_selection: %{
    fallback_pool: [
      %{remote_ip_address: "127.0.0.21", remote_port: 8805,
weight: 100}
    ]
  },
  ue: %{
```

```
    subnet_map: %{
      "internet" => ["100.64.1.0/24"],
      default: ["42.42.42.0/24"]
    }
  },
  pco: %{
    primary_dns_server_address: "8.8.8.8",
    secondary_dns_server_address: "8.8.4.4",
    ipv4_link_mtu_size: 1400
  }
```

Using Environment Variables

```
# config/runtime.exs
import Config

config :pgw_c,
  metrics: %{
    enabled: System.get_env("METRICS_ENABLED", "true") == "true",
    ip_address: System.get_env("METRICS_IP", "0.0.0.0"),
    port: String.to_integer(System.get_env("METRICS_PORT",
"9090")),
    registry_poll_period_ms: 10_000
  },
  diameter: %{
    listen_ip: System.get_env("DIAMETER_LISTEN_IP", "0.0.0.0"),
    host: System.get_env("DIAMETER_HOST") || raise("DIAMETER_HOST
required"),
    realm: System.get_env("DIAMETER_REALM") ||
raise("DIAMETER_REALM required"),
    peer_list: [
      %{
        host: System.get_env("PCRF_HOST") || raise("PCRF_HOST
required"),
        realm: System.get_env("PCRF_REALM") ||
System.get_env("DIAMETER_REALM"),
        ip: System.get_env("PCRF_IP") || raise("PCRF_IP
required"),
        initiate_connection: true
      }
    ]
  }
  # ... rest of config
```

Usage:

```
export DIAMETER_HOST="omnipgw.epc.mnc001.mcc001.3gppnetwork.org"
export DIAMETER_REALM="epc.mnc001.mcc001.3gppnetwork.org"
export PCRF_HOST="pcrf.epc.mnc001.mcc001.3gppnetwork.org"
export PCRF_IP="10.0.0.30"

mix run --no-halt
```

Related Documentation

Interface Configuration

- **PFCP Interface** - Sxb/PFCP configuration, UPF communication, session establishment
- **Diameter Gx Interface** - PCRF policy control, PCC rules, QoS management
- **Diameter Gy Interface** - OCS online charging, quota management, credit control
- **S5/S8 Interface** - GTP-C v2 configuration, SGW-C communication (4G/LTE)
- **Gn/Gp Interface** - GTP-C v1 configuration, SGSN communication (2G/3G GGSN)

Network Configuration

- **UE IP Allocation** - IP pool management, APN-based allocation, DHCP
- **PCO Configuration** - Protocol Configuration Options, DNS, P-CSCF, MTU
- **P-CSCF Monitoring** - P-CSCF discovery monitoring, IMS health tracking

Operational Guides

- **Session Management** - PDN session lifecycle, bearer management
- **Monitoring Guide** - Prometheus metrics, alerts, dashboards
- **Data CDR Format** - Offline charging records, billing integration

---

Back to Operations Guide

Data Charging Data Record (CDR) Format

Offline Charging for PGW-C

OmniPGW by Omnitouch Network Services

Table of Contents

Overview

The **Data CDR (Charging Data Record)** format provides offline charging capabilities for the Packet Gateway Control Plane (PGW-C). CDRs are generated to record bearer session events, data usage, and subscriber information for billing and analytics purposes.

This common format is compatible with SGW-C CDRs, ensuring consistency in charging records across the EPC infrastructure.

Key Features

- **CSV-based format** - Simple, human-readable comma-separated values
- **Event-based recording** - Captures bearer start, update, and end events
- **Volume metering** - Records uplink and downlink data usage
- **Automatic rotation** - Configurable file rotation based on time intervals
- **3GPP compliant** - Follows 3GPP TS 32.251 (PS domain charging) and TS 32.298 (CDR encoding)

Use Cases

Use Case	Description
Offline Charging	Generate CDRs for postpaid billing
Analytics	Analyze subscriber usage patterns
Audit Trail	Track all bearer session events
Capacity Planning	Monitor network resource utilization
Troubleshooting	Debug session and bearer issues

CDR File Format

File Naming Convention

```
<epoch_timestamp>
```

Example:

```
1726598022
```

The filename is the Unix epoch timestamp (in seconds) of when the file was created.

File Location

Default directory:

- PGW-C: `/var/log/pgw_c/cdrs/`

Configurable via `cdr_directory` parameter in config/runtime.exs.

File Header

Each CDR file begins with a multi-line header containing metadata:

```
# Data CDR File:
# File Start Time: HH:MM:SS (unix_timestamp)
# File End Time: HH:MM:SS (unix_timestamp)
# Gateway Name: <gateway_name>
#
epoch,imsi,event,charging_id,msisdn,ue_imei,timezone_raw,plmn,tac,eci
```

Header Fields:

- **File Start Time** - When the CDR file was created (human-readable and Unix timestamp)
- **File End Time** - When the file rotation will occur (human-readable and Unix timestamp)
- **Gateway Name** - Identifier for the PGW-C instance (configured via `pgw_name` parameter)
- **Column Headers** - CSV field names for the data records

CDR Fields

Field Summary

Position	Field Name	Type	Description
0	epoch	integer	Event timestamp (Unix epoch seconds)
1	imsi	string	International Mobile Subscriber Identity
2	event	string	CDR event type (e.g., "default_bearer_start")
3	charging_id	integer	Unique charging identifier for the bearer
4	msisdn	string	Mobile Station ISDN Number (phone number)
5	ue_imei	string	International Mobile Equipment Identity
6	timezone_raw	string	UE timezone (reserved, currently empty)
7	plmn	integer	Public Land Mobile Network identifier
8	tac	integer	Tracking Area Code
9	eci	integer	E-UTRAN Cell Identifier
10	sgw_ip	string	SGW-C S5/S8 control plane IP address

Position	Field Name	Type	Description
11	ue_ip	string	UE IP address (IPv4|IPv6 format)
12	pgw_ip	string	PGW-C S5/S8 control plane IP address
13	apn	string	Access Point Name
14	qci	integer	QoS Class Identifier
15	octets_in	integer	Downlink data volume (bytes)
16	octets_out	integer	Uplink data volume (bytes)

CDR Events

Event Types

CDRs are generated for three types of events:

Event Type	Format	Description	When Generated
Bearer Start	`<type>_bearer_start`	Bearer establishment	Create Session Response sent
Bearer Update	`<type>_bearer_update`	Usage reporting during session	Periodic usage reports from user plane
Bearer End	`<type>_bearer_end`	Bearer termination	Delete Session Request/Response

Bearer Types:

- `default` - Default bearer (one per PDN connection)
- `dedicated` - Dedicated bearer (zero or more per PDN connection)

Event Examples

```
default_bearer_start      - Default bearer established
default_bearer_update     - Default bearer usage update
default_bearer_end        - Default bearer terminated
dedicated_bearer_start    - Dedicated bearer established
dedicated_bearer_update   - Dedicated bearer usage update
dedicated_bearer_end      - Dedicated bearer terminated
```

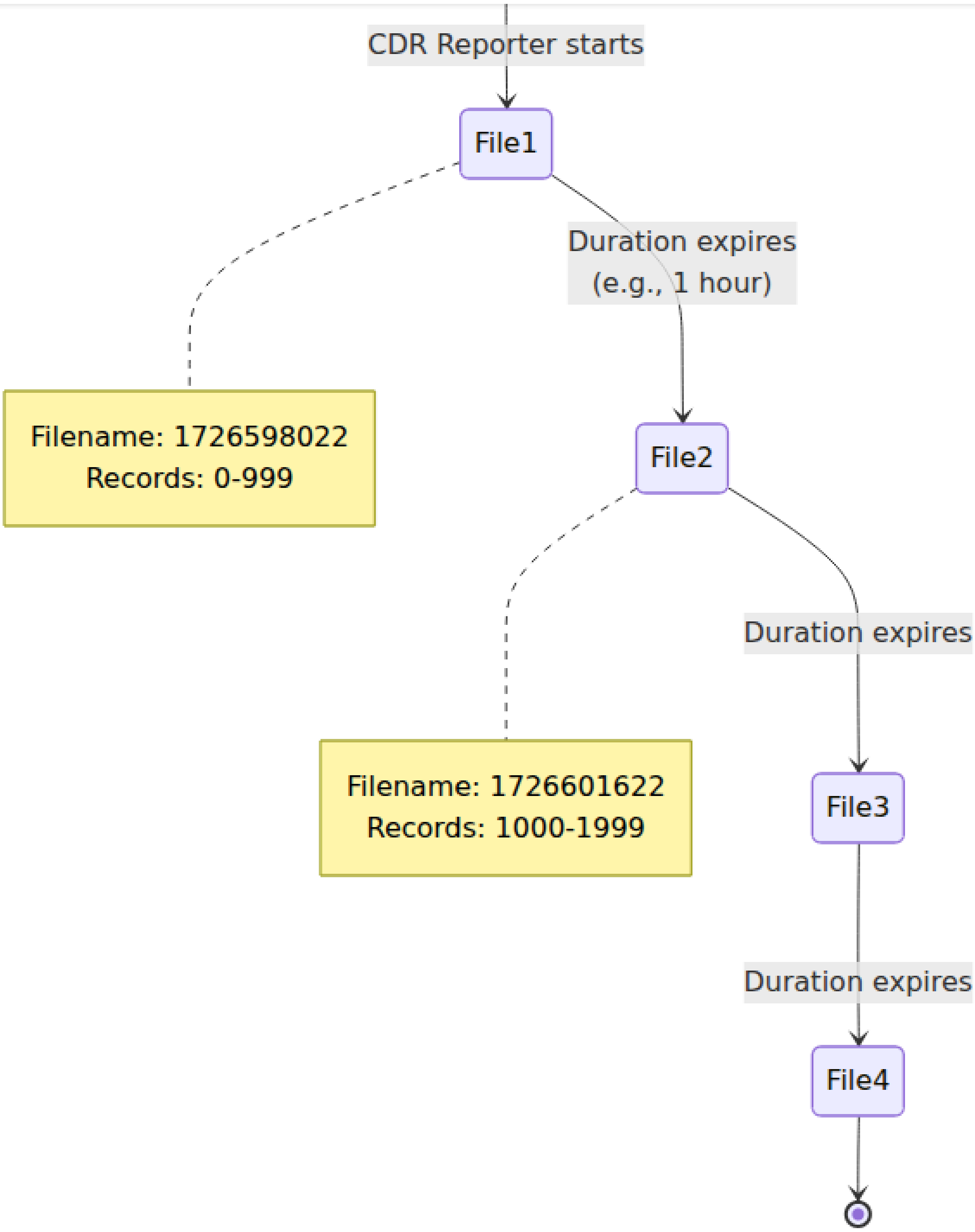
File Structure

Example CDR File

```
# Data CDR File:
# File Start Time: 18:53:42 (1726598022)
# File End Time: 19:53:42 (1726601622)
# Gateway Name: sgw-c-prod-01
# epoch,imsi,event,charging_id,msisdn,ue_imei,timezone_raw,plmn,tac,e
1726598022,310260123456789,default_bearer_start,12345,15551234567,123
1726598322,310260123456789,default_bearer_update,12345,15551234567,12
1726598622,310260123456789,default_bearer_update,12345,15551234567,12
1726598922,310260123456789,default_bearer_end,12345,15551234567,12345
```

File Rotation

CDR files are automatically rotated based on the configured duration:



Rotation Process:

1. Close current CDR file
2. Create new file with current timestamp
3. Write header to new file
4. Continue recording CDRs to new file

Configuration

Configuration Parameters

PGW-C CDR generation is configured in `config/runtime.exs`:

Parameter	Type	Description	Default	Reco
`pgw_name`	string	PGW instance identifier (appears in CDR headers)	"omni-pgw01"	Use hos instance
`cdr_file_duration`	integer	File rotation interval (ms)	3600000	360000
`cdr_directory`	string	CDR output directory path	"/tmp/pgw_c"	`/var/lo`
`usage_report_interval`	integer	URR reporting interval (ms) - how often PGW-U sends usage reports	60000	60000 (

Configuration Examples

Minimal Configuration (config/runtime.exs):

```
config :pgw_c,
  # CDR file configuration
  pgw_name: "omni-pgw01",
  cdr_file_duration: 3_600_000,           # 1 hour
  cdr_directory: "/var/log/pgw_c/cdrs",

  # URR configuration (triggers usage reports from PGW-U)
  usage_report_interval: 60_000           # 60 seconds
```

Production:

```
config :pgw_c,
  pgw_name: "pgw-c-prod-01",
  cdr_file_duration: 3_600_000,           # 1 hour rotation
  cdr_directory: "/var/log/pgw_c/cdrs",
  usage_report_interval: 60_000           # 1 minute updates
```

Development:

```
config :pgw_c,
  pgw_name: "pgw-c-dev",
  cdr_file_duration: 300_000,             # 5 minute rotation for testing
  cdr_directory: "/tmp/pgw_c_cdrs",
  usage_report_interval: 30_000           # 30 second updates for faster testing
```

High-Volume:

```
config :pgw_c,
  pgw_name: "pgw-c-prod-heavy",
  cdr_file_duration: 1_800_000,           # 30 minute rotation
  cdr_directory: "/mnt/fast-storage/cdrs",
  usage_report_interval: 300_000          # 5 minute updates (reduce overhead)
```

URR (Usage Reporting Rules)

PGW-C uses **PFCP URRs (Usage Reporting Rules)** to trigger usage reports from PGW-U. When a URR threshold is reached or time expires, PGW-U sends a Session Report Request containing usage data, which triggers CDR generation.

How URR Configuration Works:

1. `usage_report_interval` (in ms) is converted to seconds for PFCP time threshold
2. PGW-C creates URR with time threshold during session establishment
3. PGW-U sends periodic usage reports at configured interval
4. Each usage report triggers a `bearer_update` CDR event
5. Final usage report (on session deletion) triggers `bearer_end` CDR event

Example: `usage_report_interval: 60_000` means:

- PGW-U reports usage every 60 seconds
- CDR update events generated every 60 seconds
- Granular usage tracking for billing

URR Type Definition:

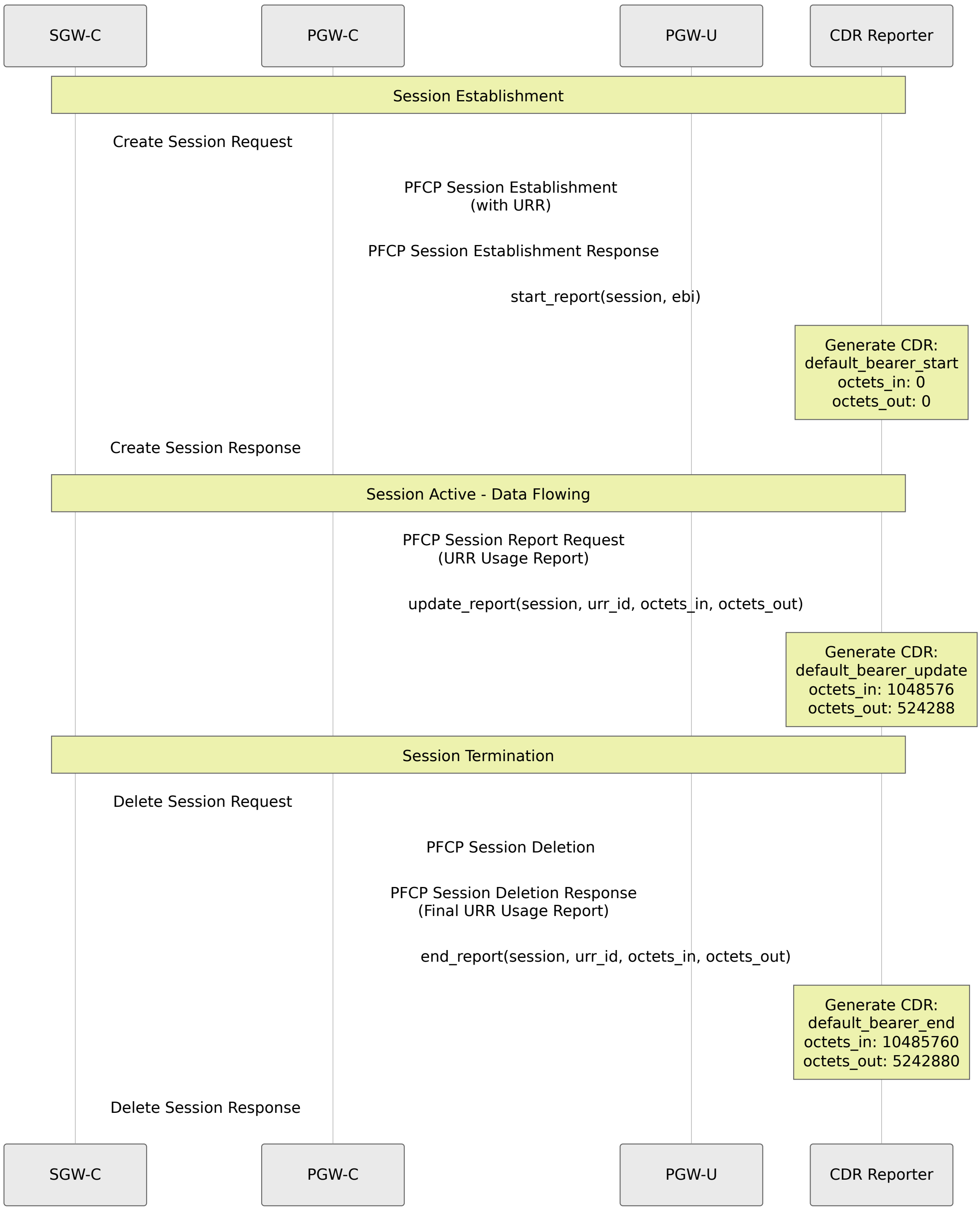
```
# lib/core/session/types.ex
defmodule PGW_C.Session.Types.URR do
  typedstruct do
    field :urr_id, non_neg_integer()
    field :measurement_method, :duration | nil
    field :reporting_triggers, :time_threshold | nil
    field :time_threshold, non_neg_integer() | nil  # seconds
  end
end
```

See PFCP Interface Documentation for URR PFCP details and lib/core/session/impl/procedures.ex:468 for URR creation during session establishment.

CDR Generation Flow

Bearer Lifecycle CDR Events

PGW-C CDR Generation:



CDR Generation Events

1. Bearer Start:

- **When:** Create Session Response is sent
- **Purpose:** Records bearer establishment with zero usage
- **octets_in:** 0
- **octets_out:** 0

2. Bearer Update:

- **When:** PFCP Session Report Request received from PGW-U (URR usage report)
- **Purpose:** Records incremental data usage
- **octets_in:** Cumulative downlink bytes since bearer start
- **octets_out:** Cumulative uplink bytes since bearer start
- **Trigger:** URR time threshold expires (configured via `usage_report_interval`)

3. Bearer End:

- **When:** PFCP Session Deletion Response received from PGW-U (with final usage report)
- **Purpose:** Records final data usage before session termination
- **octets_in:** Final total downlink bytes
- **octets_out:** Final total uplink bytes

Field Details

1. epoch (Timestamp)

Type: Unix epoch timestamp (seconds)

Description: The time when the CDR event occurred

Example:

```
1726598022  → 2025-09-17 18:53:42 UTC
```

---

2. imsi (Subscriber Identity)

Type: String (up to 15 digits)

Format: MCCMNC + MSIN

Description: International Mobile Subscriber Identity uniquely identifying the subscriber

Example:

```
310260123456789
  └──┬─┘ └──┬─┘ └──────┬──────┘
     MCC  MNC      MSIN
    (310)(260) (123456789)
```

Source: UE context, received in Create Session Request

---

3. event (CDR Event Type)

Type: String

Format: `<bearer_type>_bearer_<event>`

Values:

- `default_bearer_start`
- `default_bearer_update`
- `default_bearer_end`
- `dedicated_bearer_start`
- `dedicated_bearer_update`

- `dedicated_bearer_end`

Determination:

- If EBI (EPS Bearer ID) equals LBI (Linked Bearer ID): `default`
- If EBI does not equal LBI: `dedicated`

Source: Bearer context (EBI vs LBI comparison)

---

4. charging_id (Charging Identifier)

Type: Unsigned 32-bit integer

Description: Unique identifier for charging correlation across network elements

Example:

```
12345
```

Source: Assigned by PGW-C, received in Create Session Response

Usage:

- Correlates charging events across SGW and PGW
- Used in Diameter Gy/Gz charging interfaces
- Unique per bearer

---

5. msisdn (Phone Number)

Type: String (E.164 format)

Description: Mobile Station ISDN Number (subscriber's phone number)

Format: Country code + national number

Example:

```
15551234567
 └┬┘ └────┬────┘
  CC  National
  (1) (5551234567)
```

Source: UE context, typically from HSS via MME

---

6. ue_imei (Equipment Identity)

Type: String (15 digits)

Format: TAC (8) + SNR (6) + Spare (1)

Description: International Mobile Equipment Identity (device identifier)

Example:

```
123456789012345
└───────┬──────┘ └───┬───┘ └┘
   TAC   SNR S
```

Source: UE context, received from MME

---

7. timezone_raw (UE Timezone)

Type: String (currently reserved/empty)

Description: Reserved field for UE timezone information

Current Status: Not populated (empty field in CSV)

Future Use: May include timezone offset and daylight saving time flag

Example:

```
,  (empty field)
```

---

8. plmn (Network Identifier)

Type: Integer (legacy format)

Description: Public Land Mobile Network identifier encoded as little-endian hex

Encoding Process:

```
MCC: 505, MNC: 57
  ↓
"50557"
  ↓
Swap pairs: "055570"
  ↓
Hex to decimal: 0x055570 = 349552
```

Example:

```
349552  → MCC: 505, MNC: 57
```

Source: UE location information from MME

Note: This is a legacy encoding format for backward compatibility

---

9. tac (Tracking Area Code)

Type: Unsigned 16-bit integer

Description: Tracking Area Code identifies the tracking area where the UE is located

Range: 0 - 65535

Example:

```
1234
```

Source: UE location information, received from MME in Create Session Request

Usage:

- Identifies mobility management area
- Used for paging and location updates
- Part of TAI (Tracking Area Identity)

---

10. eci (E-UTRAN Cell Identifier)

Type: Unsigned 28-bit integer

Description: E-UTRAN Cell Identifier uniquely identifies the cell serving the UE

Format: eNodeB ID (20 bits) + Cell ID (8 bits)

Range: 0 - 268,435,455

Example:

```
5678
```

Source: UE location information from MME

Usage:

- Identifies specific cell tower and sector
- Used for handover and mobility management
- Granular location information

---

11. sgw_ip (SGW Control Plane IP)

Type: String (IPv4 or IPv6 address)

Description: SGW-C's S5/S8 control plane IP address (F-TEID)

Format: Dotted decimal (IPv4) or colon-hex (IPv6)

Example:

```
10.0.0.15       (IPv4)
2001:db8::15    (IPv6)
```

Source: Local configuration, assigned to S5/S8 interface

---

12. ue_ip (UE IP Address)

Type: String (IPv4|IPv6 format)

Description: IP address assigned to the UE for the PDN connection

Format: `<ipv4>|<ipv6>`

Examples:

```
172.16.1.100|                      (IPv4 only)
|2001:db8::1                       (IPv6 only)
172.16.1.100|2001:db8::1          (Dual-stack)
```

Source: PDN Address Allocation (PAA) from PGW-C

Notes:

- Empty IPv4: No IPv4 address allocated
- Empty IPv6: No IPv6 address allocated
- Both present: Dual-stack PDN connection

---

13. pgw_ip (PGW Control Plane IP)

Type: String (IPv4 or IPv6 address)

Description: PGW-C's S5/S8 control plane IP address (remote F-TEID)

Format: Dotted decimal (IPv4) or colon-hex (IPv6)

Example:

```
10.0.0.20       (IPv4)
2001:db8::20    (IPv6)
```

Source: Received in Create Session Response from PGW-C

---

14. apn (Access Point Name)

Type: String (up to 100 characters)

Description: Access Point Name identifying the external network (PDN)

Format: DNS-like label format

Examples:

```
internet
ims
mms
enterprise.corporate
```

Source: Received in Create Session Request from MME

Usage:

- Determines which external network to connect to
- Drives policy and charging rules
- May determine IP address pool

15. qci (QoS Class Identifier)

Type: Unsigned 8-bit integer

Description: QoS Class Identifier defines the bearer's quality of service

Range: 1 - 9 (standardized), 128-254 (operator-specific)

Standardized QCI Values:

QCI	Resource Type	Priority	Packet Delay	Packet Loss	Example Service
1	GBR	2	100 ms	10^-2	Conversational Voice
2	GBR	4	150 ms	10^-3	Conversational Video
3	GBR	3	50 ms	10^-3	Real-time Gaming
4	GBR	5	300 ms	10^-6	Non-conversational Video
5	Non-GBR	1	100 ms	10^-6	IMS Signaling
6	Non-GBR	6	300 ms	10^-6	Video (buffered)
7	Non-GBR	7	100 ms	10^-3	Voice, Video, Gaming
8	Non-GBR	8	300 ms	10^-6	Video (buffered)
9	Non-GBR	9	300 ms	10^-6	Default Bearer

Example:

```
9  → Default bearer (best effort)
```

Source: Bearer QoS parameters from PGW-C

---

16. octets_in (Downlink Volume)

Type: Unsigned 64-bit integer

Description: Number of bytes transmitted in the downlink direction (network → UE)

Units: Bytes

Example:

```
1048576  → 1 MB downlink
```

Source: PFCP Volume Measurement from PGW-U (via URR usage reports)

Notes:

- Cumulative for `update` events
- Final total for `end` events
- Always 0 for `start` events
- Reports triggered by URR time threshold (configured via `usage_report_interval`)

---

17. octets_out (Uplink Volume)

Type: Unsigned 64-bit integer

Description: Number of bytes transmitted in the uplink direction (UE → network)

Units: Bytes

Example:

```
524288  → 512 KB uplink
```

Source: PFCP Volume Measurement from PGW-U (via URR usage reports)

Notes:

- Cumulative for `update` events
- Final total for `end` events
- Always 0 for `start` events
- Reports triggered by URR time threshold (configured via `usage_report_interval`)

Examples

Example 1: Basic Session with Single Update

Timeline:

1. Bearer established
2. 5 minutes later: Usage update (10 MB down, 5 MB up)
3. Session terminated

CDR Output:

```
# Data CDR File:
# File Start Time: 10:00:00 (1726570800)
# File End Time: 11:00:00 (1726574400)
# Gateway Name: pgw-c-01
# epoch,imsi,event,charging_id,msisdn,ue_imei,timezone_raw,plmn,tac,e
1726570800,310260111111111,default_bearer_start,10001,15551111111,111
1726571100,310260111111111,default_bearer_update,10001,15551111111,11
1726571400,310260111111111,default_bearer_end,10001,15551111111,11111
```

Example 2: Dual-Stack Session with Multiple Updates

Timeline:

1. Dual-stack bearer established (IPv4 + IPv6)
2. Multiple usage updates
3. Session terminated

CDR Output:

```
1726570800,310260222222222,default_bearer_start,10002,15552222222,222
1726571100,310260222222222,default_bearer_update,10002,15552222222,22
1726571400,310260222222222,default_bearer_update,10002,15552222222,22
1726571700,310260222222222,default_bearer_update,10002,15552222222,22
1726572000,310260222222222,default_bearer_end,10002,15552222222,22222
```

Example 3: Session with Dedicated Bearer

Timeline:

1. Default bearer established (QCI 9)
2. Dedicated bearer created for video (QCI 6)
3. Usage updates for both bearers
4. Dedicated bearer deleted
5. Default bearer terminated

CDR Output:

```
1726570800,310260333333333,default_bearer_start,10003,15553333333,333
1726571100,310260333333333,dedicated_bearer_start,10004,15553333333,3
1726571400,310260333333333,default_bearer_update,10003,15553333333,33
1726571400,310260333333333,dedicated_bearer_update,10004,15553333333,
1726571700,310260333333333,dedicated_bearer_end,10004,15553333333,333
1726572000,310260333333333,default_bearer_end,10003,15553333333,33333
```

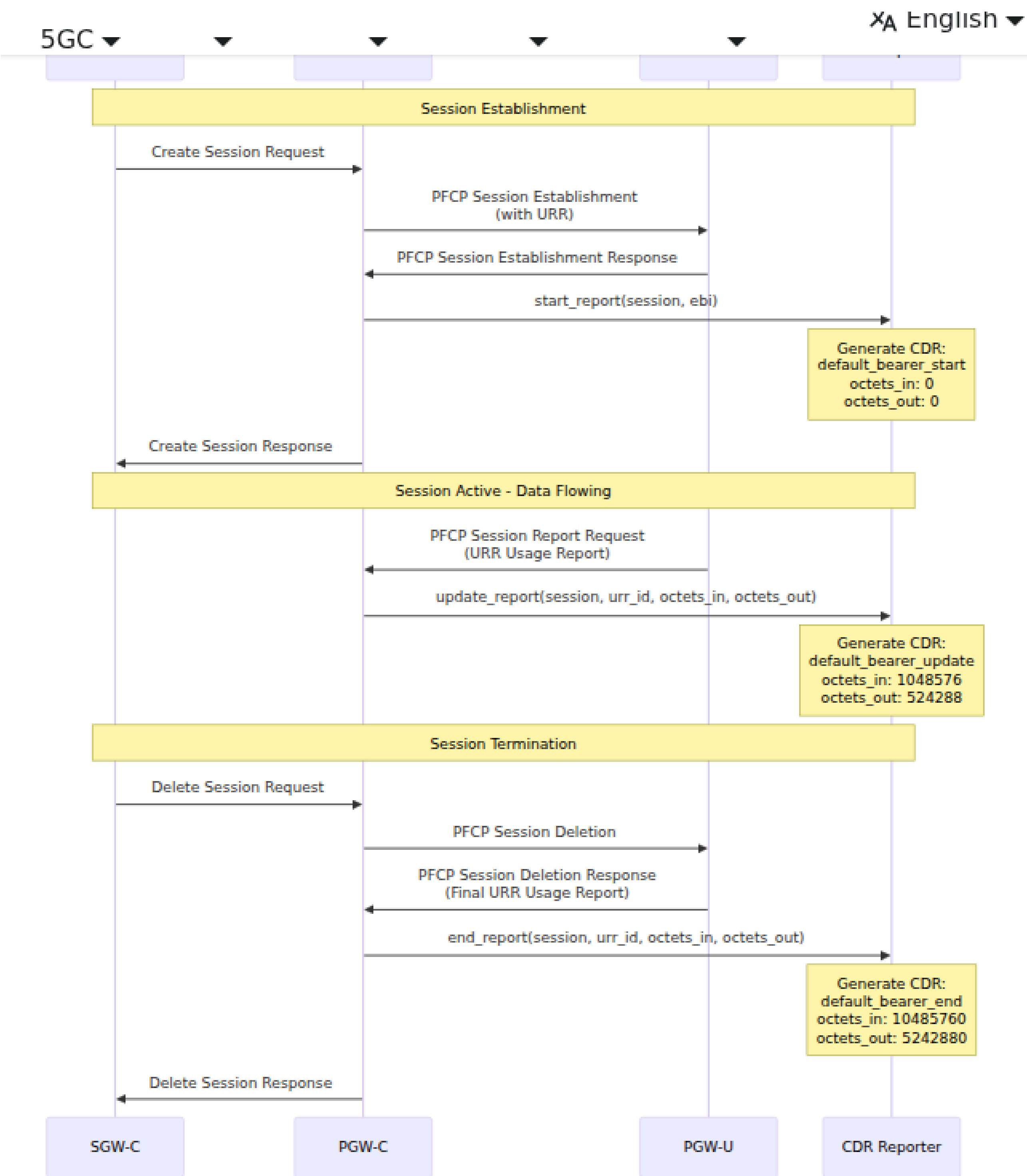
Analysis:

- Default bearer (10003) carries background traffic (10 MB down, 4 MB up)
- Dedicated bearer (10004) carries video traffic (200 MB down, 2 MB up)
- Different QCI values (9 vs 6) reflect different QoS treatment

---

Integration

CDR Processing Pipeline



CDR Collection Methods

1. File-based Collection:

```
# Monitor CDR directory (PGW-C)
inotifywait -m /var/log/pgw_c/cdrs/ -e close_write | while read
path action file; do
    # File rotation completed, process CDR
    process_cdr "$path$file"
done
```

2. Real-time Streaming:

```
# Tail and stream to processing pipeline
tail -F /var/log/pgw_c/cdrs/* | process_cdr_stream
```

Related Documentation

- Session Management - Session lifecycle and CDR triggers
- PFCP Interface - Usage reporting from PGW-U via URRs
- Monitoring Guide - CDR generation metrics and alerting
- Configuration Guide - CDR and URR configuration parameters
- Diameter Gx Interface - Policy control for QCI values in CDRs
- Diameter Gy Interface - Online charging integration

3GPP References

- TS 32.251 - Packet Switched (PS) domain charging
- TS 29.274 - 3GPP Evolved Packet System (EPS); GTP-C protocol
- TS 29.244 - Interface between CP and UP nodes (PFCP) - **URR support**
- TS 32.298 - CDR encoding

CDR Format - *Offline Charging Records for PGW-C*

Developed by Omnitouch Network Services

Documentation Version: 1.0 **Last Updated:** 2025-12-28

Diameter Gx Interface Documentation

Policy and Charging Rules Function (PCRF) Interface

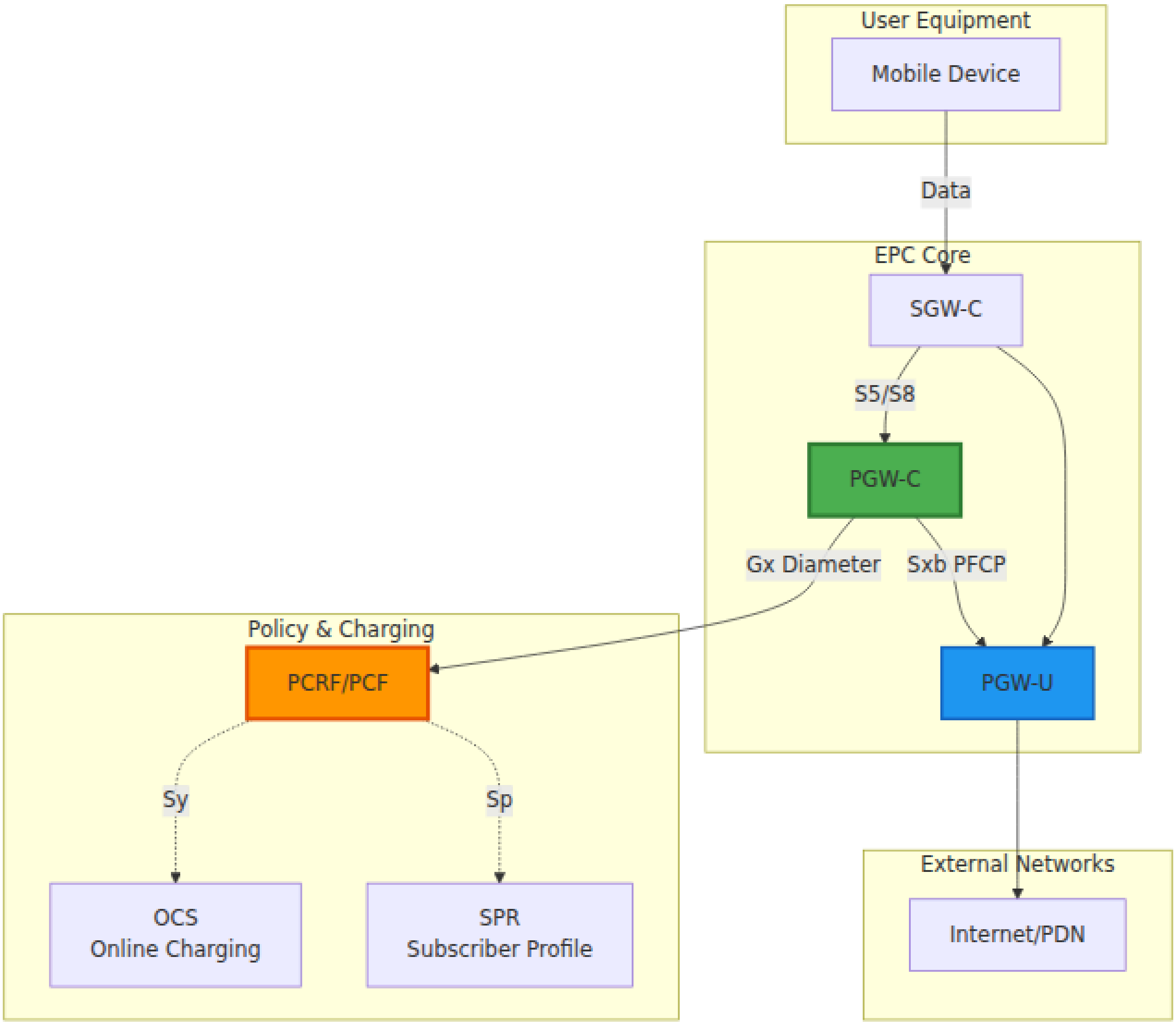
Table of Contents

Overview

The **Gx interface** connects PGW-C to the **PCRF (Policy and Charging Rules Function)** or **PCF (Policy Control Function)** in 5G networks. This interface enables:

- **Dynamic Policy Control** - Real-time QoS and policy enforcement
- **Charging Control** - Credit authorization and usage tracking
- **Service Awareness** - Application-level traffic differentiation
- **Subscriber Profile Management** - Per-user policy application

Gx in the Network Architecture



Key Functions

Function	Description
Policy Provisioning	PCRF provides PCC rules defining how to handle traffic
QoS Control	Dynamic adjustment of bitrates and QoS parameters
Charging Control	Credit authorization for prepaid/postpaid scenarios
Gating Control	Enable/disable traffic flows based on policy
Usage Monitoring	Track data consumption per service

Gx Interface Basics

3GPP Reference

- **Specification:** 3GPP TS 29.212
- **Diameter Application ID:** 16777238 (Gx)
- **Protocol:** Diameter Base Protocol (RFC 6733)

Session Concept

Each UE PDN connection has a corresponding **Gx session** identified by a **Session-ID**. This session:

- Created when UE attaches (CCR-Initial)
- Updated during the connection lifetime (CCR-Update) - optional, e.g. policy changes or location change reporting
- Terminated when UE detaches (CCR-Termination)

Session ID Format

```
Session-ID: <Origin-Host>;<high32>;<low32>[;<optional>]
Example: omni-
pgw_c.epc.mnc999.mcc999.3gppnetwork.org;1234567890;98765
```

Components:

- **Origin-Host:** PGW-C's Diameter identity
- **high32:** High 32 bits of unique identifier
- **low32:** Low 32 bits of unique identifier

Diameter Protocol

Message Structure

Diameter messages are binary-encoded with the following structure:

```
Diameter Header (20 bytes)
├── Version (1 byte) = 1
├── Message Length (3 bytes)
├── Flags (1 byte)
│   ├── R: Request (1) / Answer (0)
│   ├── P: Proxiable
│   ├── E: Error
│   └── T: Potentially retransmitted
├── Command Code (3 bytes)
├── Application ID (4 bytes) = 16777238 (Gx)
├── Hop-by-Hop ID (4 bytes)
└── End-to-End ID (4 bytes)

AVPs (Attribute-Value Pairs)
├── AVP Header
│   ├── AVP Code
│   ├── Flags (V, M, P)
│   ├── AVP Length
│   └── Vendor ID (optional)
└── AVP Data
```

Key Diameter Concepts

AVP (Attribute-Value Pair):

- Basic data unit in Diameter
- Contains a code, flags, and value
- Can be nested (Grouped AVP)

Command:

- Request/Answer pair
- CCR (Credit-Control-Request) / CCA (Credit-Control-Answer)

Result Codes:

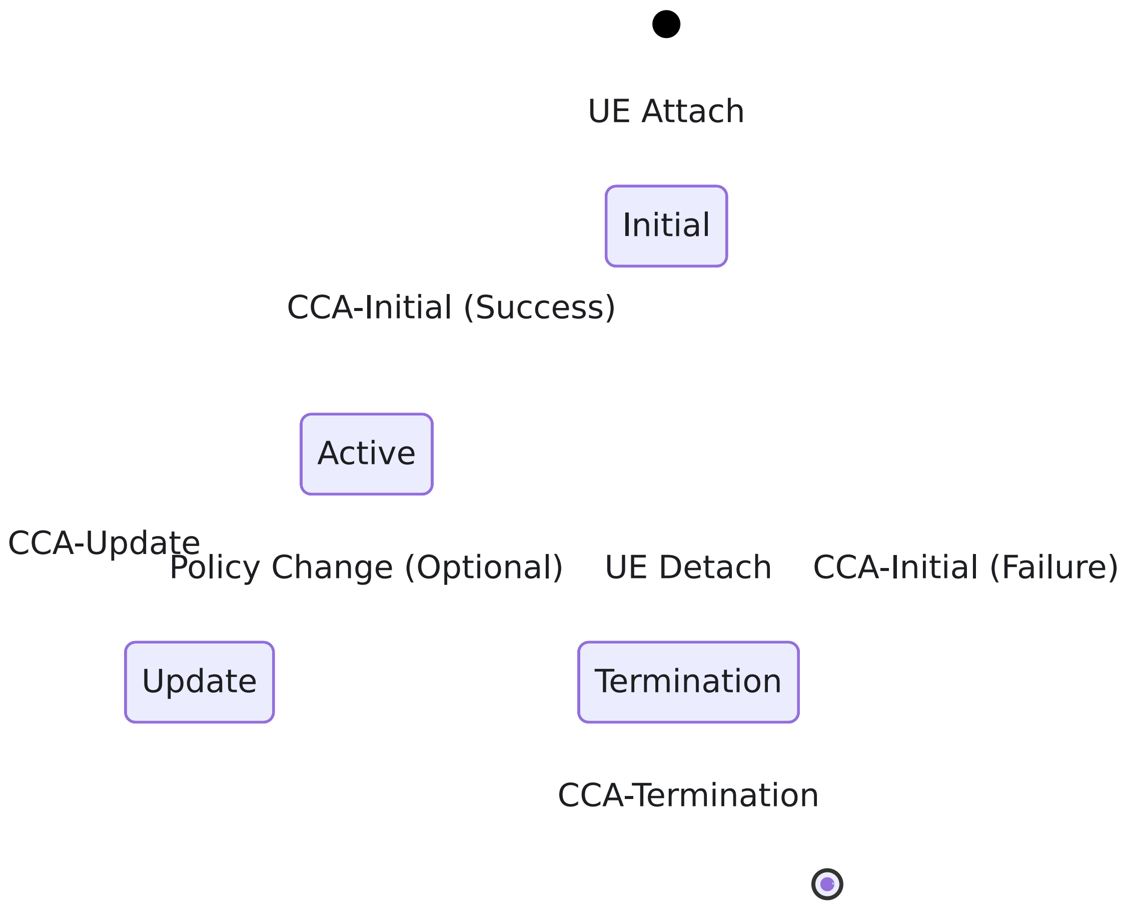
- `2001` - DIAMETER_SUCCESS
- `3xxx` - Protocol errors
- `4xxx` - Transient failures

- `5xxx` - Permanent failures

Credit Control Messages

PGW-C uses the **Diameter Credit Control Application** (RFC 4006) for Gx.

Message Types



CCR-Initial (Credit Control Request - Initial)

When: UE creates a new PDN connection

Purpose:

- Request initial policy and charging rules
- Provide UE and network context to PCRF

- Obtain QoS parameters and charging authorization

Key AVPs Sent by PGW-C:

AVP Name	AVP Code	Type	Description
Session-Id	263	UTF8String	Unique Gx session identifier
Auth-Application-Id	258	Unsigned32	16777238 (Gx)
Origin-Host	264	DiamIdent	PGW-C's Diameter identity
Origin-Realm	296	DiamIdent	PGW-C's Diameter realm
Destination-Realm	283	DiamIdent	PCRF's realm
CC-Request-Type	416	Enumerated	1 = INITIAL_REQUEST
CC-Request-Number	415	Unsigned32	Sequence number (starts at 0)
Subscription-Id	443	Grouped	UE identifier (IMSI/MSISDN)
Called-Station-Id	30	UTF8String	APN name
Framed-IP-Address	8	OctetString	Allocated UE IPv4 address (IPv4 and IPv4v6 PDNs)
Framed-IPv6-Prefix	97	OctetString	Allocated UE IPv6 prefix (IPv6-only PDNs)
3GPP-User-Location-Info	22	OctetString	Initial subscriber ULI (TS 29.061)
User-Equipment-Info	458	Grouped	UE identity (e.g. IMEISV), when available
IP-CAN-Type	1027	Enumerated	5 = 3GPP-EPS

AVP Name	AVP Code	Type	Description
RAT-Type	1032	Enumerated	Mapped from the GTP RAT type (e.g. 1004 = EUTRAN)
QoS-Information	1016	Grouped	Current QoS (APN-AMBR UL/DL)
Default-EPS-Bearer-QoS	1049	Grouped	Default bearer QCI/ARP (QCI 5)
Network-Request-Support	1024	Enumerated	Network-initiated procedures
Supported-Features	628	Grouped	Gx feature list (Rel-8/9/10 Gx)

> The framed-address AVP is selected by the UE address family: `Framed-IP-Address` for IPv4 (and the IPv4 of an IPv4v6 PDN), `Framed-IPv6-Prefix` for an IPv6-only PDN. For IPv4v6 the IPv6 prefix is not carried on Gx. See IPv6 / Dual-Stack.

Example CCR-I Structure:

```
CCR (Command Code: 272, Request)
├── Session-Id: "pgw_c.example.com;123;456"
├── Auth-Application-Id: 16777238
├── Origin-Host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org"
├── Origin-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── Destination-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── CC-Request-Type: INITIAL_REQUEST (1)
├── CC-Request-Number: 0
├── Subscription-Id (Grouped)
│   ├── Subscription-Id-Type: END_USER_IMSI (1)
│   └── Subscription-Id-Data: "310260123456789"
├── Called-Station-Id: "internet"
├── Framed-IP-Address: 100.64.1.42
├── IP-CAN-Type: 3GPP-EPS (5)
├── RAT-Type: EUTRAN (1004)
├── QoS-Information (Grouped)
│   ├── APN-Aggregate-Max-Bitrate-UL: 100000000 (100 Mbps)
│   └── APN-Aggregate-Max-Bitrate-DL: 50000000 (50 Mbps)
├── Network-Request-Support: 1
└── Supported-Features: [...]
```

CCA-Initial (Credit Control Answer - Initial)

Sent by: PCRF in response to CCR-I

Purpose:

- Authorize or reject the session
- Provide PCC rules for traffic handling
- Specify QoS parameters

Key AVPs Received by PGW-C:

AVP Name	AVP Code	Description
Result-Code	268	Success (2001) or error code
Experimental-Result	297	Vendor-specific result codes
QoS-Information	1016	Authorized QoS (may differ from request)
Charging-Rule-Install	1001	PCC rules to activate
Charging-Rule-Definition	1003	Inline rule definitions
Default-EPS-Bearer-QoS	1049	QoS for default bearer

Success Response Example:

```
CCA (Command Code: 272, Answer)
├── Session-Id: "pgw_c.example.com;123;456"
├── Result-Code: DIAMETER_SUCCESS (2001)
├── Origin-Host: "pcrf.example.com"
├── Origin-Realm: "example.com"
├── Auth-Application-Id: 16777238
├── CC-Request-Type: INITIAL_REQUEST (1)
├── CC-Request-Number: 0
├── QoS-Information (Grouped)
│   ├── APN-Aggregate-Max-Bitrate-UL: 50000000 (50 Mbps - reduced)
│   └── APN-Aggregate-Max-Bitrate-DL: 100000000 (100 Mbps -
increased)
├── Charging-Rule-Install (Grouped)
│   ├── Charging-Rule-Name: "default_internet_rule"
│   └── Charging-Rule-Name: "video_streaming_rule"
└── Charging-Rule-Definition (Grouped)
    ├── Charging-Rule-Name: "default_internet_rule"
    ├── QoS-Information: {...}
    └── Precedence: 1000
```

CCR-Termination (Credit Control Request - Termination)

When: UE detaches or PDN connection is deleted

Purpose:

- Notify PCRF of session termination
- Final accounting/charging record

Key Differences from CCR-I:

- `CC-Request-Type: TERMINATION_REQUEST (3)`
- May include usage statistics
- Simplified AVP set

Example CCR-T:

```
CCR (Command Code: 272, Request)
├── Session-Id: "pgw_c.example.com;123;456"
├── Auth-Application-Id: 16777238
├── Origin-Host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org"
├── Origin-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── Destination-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── CC-Request-Type: TERMINATION_REQUEST (3)
├── CC-Request-Number: 1
└── Termination-Cause: DIAMETER_LOGOUT (1)
```

CCA-Termination

Sent by: PCRF in response to CCR-T

Purpose:

- Acknowledge session termination
- No policy rules returned

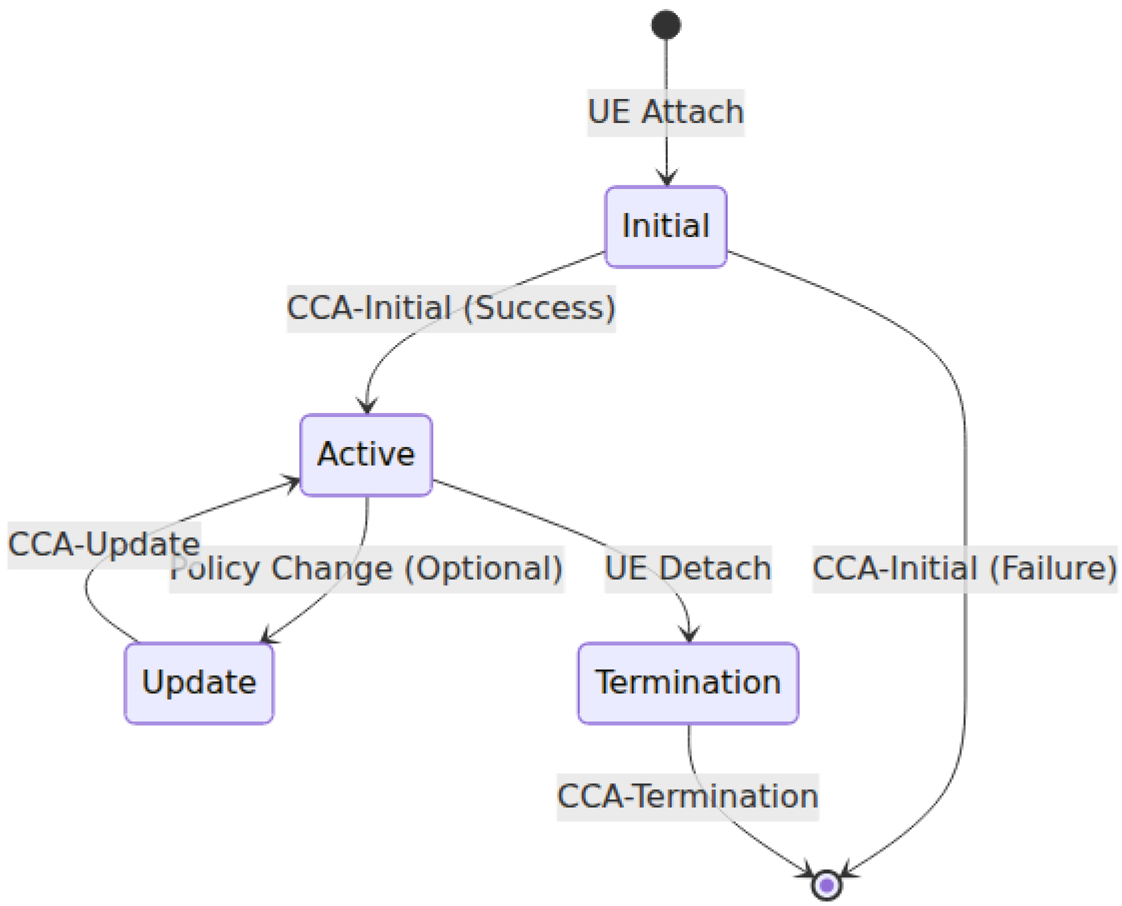
Example CCA-T:

```
CCA (Command Code: 272, Answer)
├── Session-Id: "pgw_c.example.com;123;456"
├── Result-Code: DIAMETER_SUCCESS (2001)
├── Origin-Host: "pcrf.example.com"
├── Origin-Realm: "example.com"
├── Auth-Application-Id: 16777238
├── CC-Request-Type: TERMINATION_REQUEST (3)
└── CC-Request-Number: 1
```

Policy and Charging Rules

PCC Rule Structure

A **PCC (Policy and Charging Control) Rule** defines how to handle specific traffic flows:



Rule Components

1. Rule Name:

- Unique identifier for the rule
- Example: `"video_streaming_rule"`

2. Precedence:

- Lower number = higher priority
- Range: 0-65535
- Used when multiple rules match

3. Flow Filters (TFT - Traffic Flow Template):

- Defines which packets match this rule
- Examples:
  - IP 5-tuple: Protocol, Src/Dst IP, Src/Dst Port
  - `"permit out ip from any to 8.8.8.8 80"`

4. QoS Information:

- **QCI (QoS Class Identifier):** 1-9 (standardized), 128-254 (operator-specific)
  - QCI 1: Conversational Voice
  - QCI 5: IMS Signaling
  - QCI 9: Default Internet
- **ARP (Allocation and Retention Priority):** Pre-emption capability
- **MBR/GBR:** Maximum/Guaranteed Bit Rates

5. Charging Information:

- **Rating Group:** Identifies charging category (used by OCS - see Diameter Gy Interface)
- **Metering Method:** Volume, time, or event-based
- **Online/Offline Charging:** OCS (prepaid via Diameter Gy) vs. offline CDRs (postpaid - see Data CDR Format)

6. Gating Status:

- **OPEN:** Allow traffic
- **CLOSED:** Block traffic

Dynamic Rule Provisioning

PCRF can provide rules in two ways:

1. Predefined Rules (by name):

```
Charging-Rule-Install (Grouped)
├── Charging-Rule-Name: "gold_subscriber_internet"
└── Charging-Rule-Name: "video_qos_boost"
```

2. Dynamic Rules (inline definition):

```
Charging-Rule-Definition (Grouped)
├── Charging-Rule-Name: "dynamic_rule_123"
├── Precedence: 100
├── Flow-Information (Grouped)
│   ├── Flow-Description: "permit out ip from any to 192.0.2.0/24"
│   └── Flow-Direction: DOWNLINK
├── QoS-Information (Grouped)
│   ├── QoS-Class-Identifier: 5
│   ├── Max-Requested-Bandwidth-UL: 10000000
│   └── Max-Requested-Bandwidth-DL: 50000000
└── Rating-Group: 1000
```

QoS Information AVP

APN-AMBR (Aggregate Maximum Bit Rate):

Applies to all non-GBR bearers for this APN:

```
QoS-Information (Grouped)
├── APN-Aggregate-Max-Bitrate-UL: 100000000  # 100 Mbps
└── APN-Aggregate-Max-Bitrate-DL: 200000000  # 200 Mbps
```

PGW-C Response:

- Updates internal AMBR state
- Sends Session Modification Request to PGW-U with updated QER

Location Change Reporting

PGW-C can notify the PCRF when a subscriber's **location**, **radio access technology (RAT)**, **serving PLMN** or **UE time zone** changes during the life of a session. The change is carried in a **CCR-Update** as the **3GPP-User-Location-Info** and related AVPs.

Reporting is opt-in (PCRF-controlled)

This reporting is **conditional on the PCRF requesting it** — exactly as required by 3GPP TS 29.212 §4.5.7 / §5.3.7. The PCRF *arms* the events it wants to hear about by including one or more **Event-Trigger** AVPs in a **CCA** (or in an **RAR**). PGW-C stores the armed set per Gx session and only sends a CCR-Update when:

1. an armed Event-Trigger's condition actually occurs, **and**
2. a Gx session exists for the subscriber.

If the PCRF arms **no** location-related Event-Triggers, PGW-C stays silent on location changes — it simply updates its internal state and answers the GTP-C Modify Bearer Request. This keeps the interface quiet for PCRFs that don't care about mobility, and compliant for those that do.

The armed set can change at any time during the session. PGW-C updates it from every Event-Trigger AVP set the PCRF sends — in the CCA-Initial, in a CCA-Update, or in a PCRF-initiated **RAR**. A present Event-Trigger set replaces the currently armed set; a message that carries no Event-Trigger AVP leaves the armed set unchanged. So a PCRF can start a session with no location reporting and later turn it on (or off) with an RAR, and vice-versa.

Supported Event-Triggers

Event-Trigger	Value	Fires when…	Reported AVP
`RAT_CHANGE`	2	RAT-Type changes (e.g. EUTRAN → UTRAN)	`RAT-Type`
`PLMN_CHANGE`	4	Serving network (MCC/MNC) changes	`3GPP-SGSN-MCC-MNC`
`USER_LOCATION_CHANGE`	13	Any change in User Location Info	`3GPP-User-Location-Info`
`UE_TIME_ZONE_CHANGE`	25	UE time zone changes	`3GPP-MS-TimeZone`
`TAI_CHANGE`	26	Tracking Area Identity changes	`3GPP-User-Location-Info`
`ECGI_CHANGE`	27	E-UTRAN Cell Global Identity changes	`3GPP-User-Location-Info`

Multiple triggers can fire at once (e.g. moving to a new cell in a new tracking area fires `USER_LOCATION_CHANGE`, `TAI_CHANGE` **and** `ECGI_CHANGE`); all fired-and-armed triggers are listed in the CCR-Update's `Event-Trigger` AVPs.

Where location comes from

- **CCR-Initial** now includes the subscriber's initial `3GPP-User-Location-Info`, giving the PCRF a baseline.
- During the session, a **GTP-C Modify Bearer Request** (S5/S8) carries the updated User Location Info, RAT-Type, Serving Network and UE Time Zone.

PGW-C parses these, compares them against the stored values, and reports any armed change.

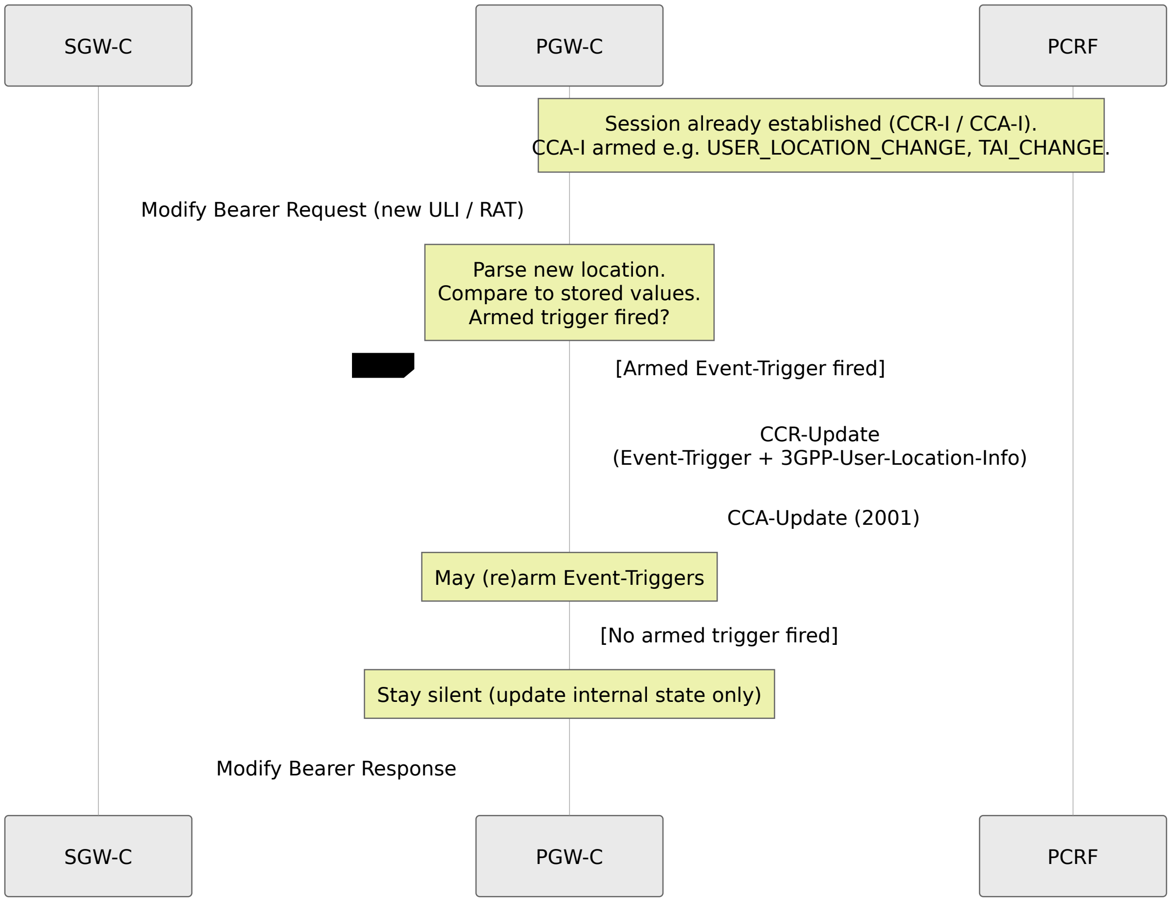
- IEs **absent** from a Modify Bearer Request keep their previous value, so a partial Modify Bearer Request never looks like a spurious location change.

3GPP-User-Location-Info encoding

The AVP is encoded per **3GPP TS 29.061 §16.4.7.2**: a one-octet *Geographic Location Type* followed by the location field(s), with PLMN identities encoded per 3GPP TS 24.008 §10.5.1.3. Supported types:

Type	Value	Contents
CGI	0	PLMN + LAC + CI (2G)
SAI	1	PLMN + LAC + SAC (3G)
RAI	2	PLMN + LAC + RAC
TAI	128	PLMN + TAC (LTE)
ECGI	129	PLMN + ECI (LTE)
TAI and ECGI	130	PLMN + TAC + PLMN + ECI (LTE, preferred)

Message Flow



> **Robustness:** reporting the change to the PCRF never blocks the GTP-C Modify Bearer Response. If the CCR-Update times out or the PCRF returns an error, PGW-C logs it and still answers the SGW-C — the subscriber's data path is not held up by a policy-reporting failure.

Inspecting the armed triggers via the API

The Sessions API (`GET /api/sessions` and `GET /api/sessions/:imsi`) returns a complete, JSON-safe dump of the session under the `state` key, in addition to the curated top-level fields. The PCRF-armed Event-Triggers are visible at `state.gx.requested_event_triggers`, alongside the rest of the Gx session state (`session_id`, `cc_request_number`, destination host/realm) and the current `state.uli`. This makes it possible to confirm from operations tooling whether a subscriber is actually subscribed to location reporting.

Overriding the armed triggers via the API

The armed set can also be changed directly on a live session with **`PATCH /api/sessions/:imsi`** (useful for testing the feature without a PCRF, or to force reporting on/off). The body is **deep-merged into the session state**, so the path mirrors the structure returned by `GET` — the armed triggers live under `gx`:

```
PATCH /api/sessions/310260000000001
Content-Type: application/json

{ "gx": { "requested_event_triggers": [13, 26, 27] } }
```

- The value is a list of Gx Event-Trigger codes; it replaces the armed set (other `gx` fields are left untouched by the merge).
- An empty list (`[]`) disarms all location reporting for the session.
- The response echoes the updated, sanitized session `state`.

This endpoint has **no guard rails**: the body is deep-merged into the live session, so *any* field can be changed — including to a value that breaks the session. Changes that have downstream effects are propagated to the UPF. Updating the APN-AMBR or the default-bearer QoS recomputes the default-bearer QER (QCI / MBR / GBR) and pushes it as an Update QER in a PFCP Session Modification:

```
PATCH /api/sessions/310260000000001
Content-Type: application/json

{ "ambr": { "uplink": 200000, "downlink": 200000 } }
```

```
PATCH /api/sessions/310260000000001
Content-Type: application/json

{ "default_bearer_qos": { "qci": 5, "mbr_ul": 300000, "mbr_dl": 250000 } }
```

When `default_bearer_qos` provides explicit MBR/GBR (GBR bearers) those drive the QER rate limits; otherwise the APN-AMBR is used as the default-bearer

MBR.

> Note: overriding the armed triggers changes PGW-C's local view of what the PCRF armed. The PCRF can still re-arm the set later via a CCA or RAR, which will overwrite a manual override.

Configuration

Basic Gx Configuration

Edit `config/runtime.exs`:

```
config :pgw_c,
  diameter: %{
    # IP address to listen for Diameter connections
    listen_ip: "0.0.0.0",

    # PGW-C's Diameter identity (Origin-Host)
    host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org",

    # PGW-C's Diameter realm (Origin-Realm)
    realm: "epc.mnc999.mcc999.3gppnetwork.org",

    # List of PCRF peers
    peer_list: [
      %{
        # PCRF Diameter identity
        host: "pcrf.epc.mnc999.mcc999.3gppnetwork.org",

        # PCRF realm (usually same as PGW-C realm)
        realm: "epc.mnc999.mcc999.3gppnetwork.org",

        # PCRF IP address
        ip: "10.0.0.30",

        # Whether PGW-C initiates connection to PCRF
        # true = PGW-C connects to PCRF
        # false = Wait for PCRF to connect
        initiate_connection: true
      }
    ]
  }
```

Multiple PCRF Peers

For redundancy or geographic distribution:

```
config :pgw_c,
  diameter: %{
    listen_ip: "0.0.0.0",
    host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org",
    realm: "epc.mnc999.mcc999.3gppnetwork.org",
    peer_list: [
      %{
        host: "pcrf-primary.example.com",
        realm: "epc.mnc999.mcc999.3gppnetwork.org",
        ip: "10.0.1.30",
        initiate_connection: true
      },
      %{
        host: "pcrf-backup.example.com",
        realm: "epc.mnc999.mcc999.3gppnetwork.org",
        ip: "10.0.2.30",
        initiate_connection: true
      }
    ]
  }
```

Load Balancing:

- Diameter protocol handles peer selection
- Requests distributed based on availability
- Automatic failover on peer failure

Hostname Resolution

Diameter Identities must be FQDNs (Fully Qualified Domain Names):

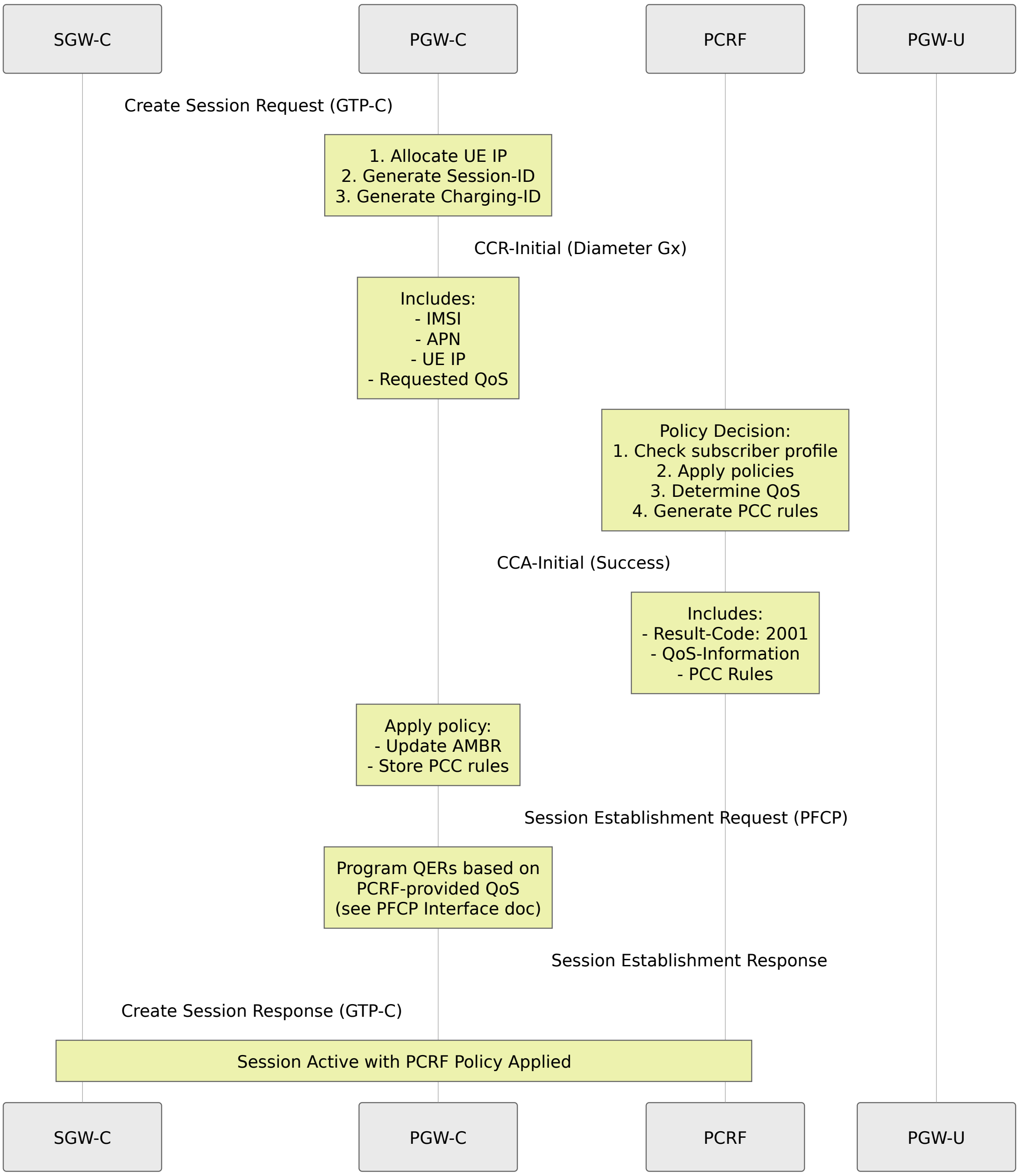
```
# CORRECT - FQDN format
host: "pgw_c.epc.mnc999.mcc999.3gppnetwork.org"

# INCORRECT - Not a valid Diameter Identity
host: "pgw_c"
host: "10.0.0.20"  # IP addresses not allowed
```

Realm Format:

- Must be valid domain name
- Typically matches 3GPP PLMN format: `epc.mncXXX.mccYYY.3gppnetwork.org`

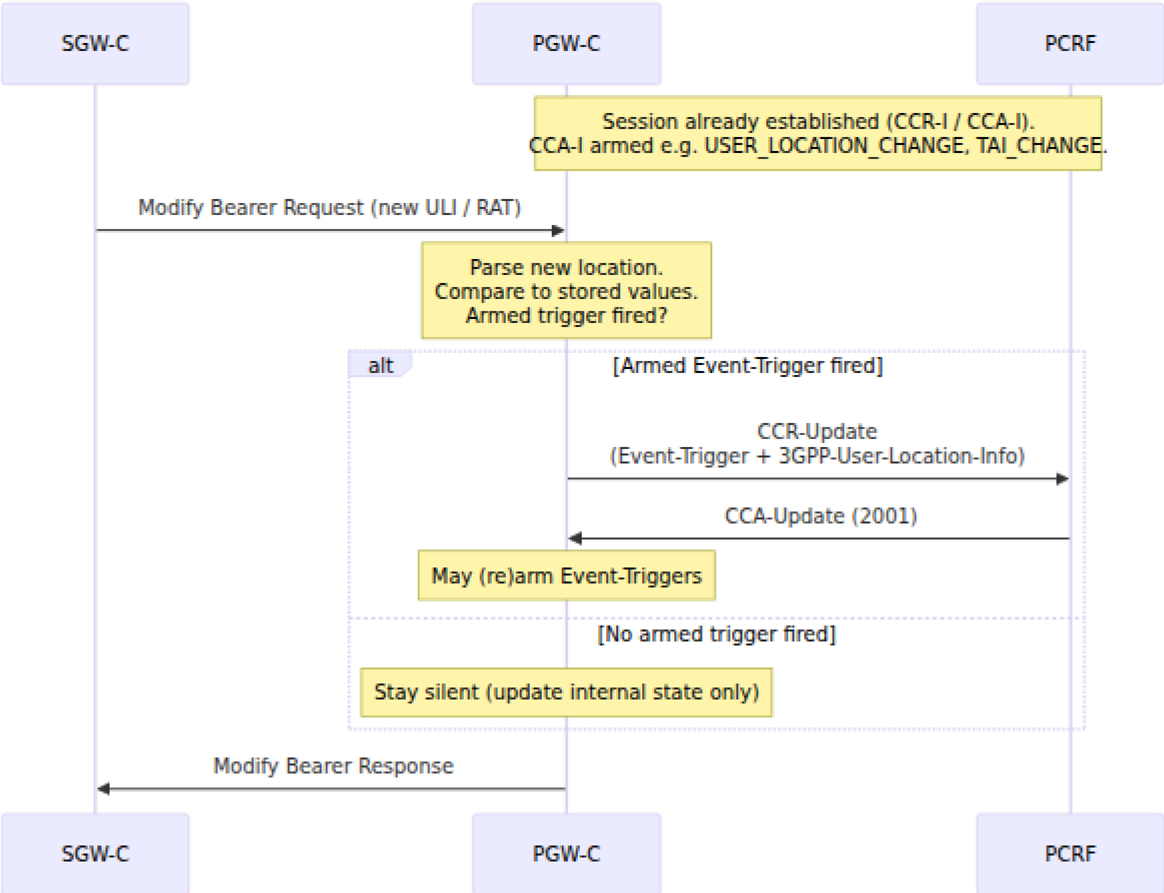
Message Flows

Successful Session Establishment

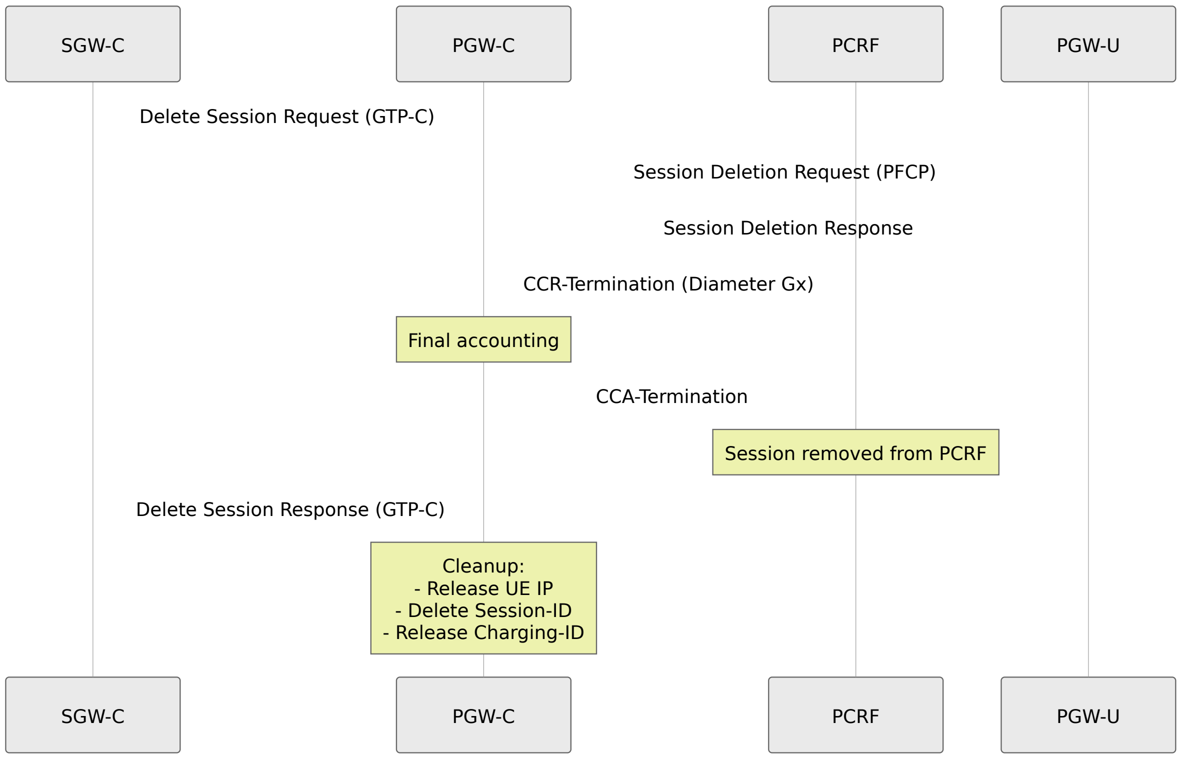


Note: QoS parameters from PCRF are translated into QERs (QoS Enforcement Rules) and programmed into PGW-U via PFCP. See PFCP Interface for QER details.

Policy Update (Network-Initiated)



Session Termination



Error Handling

Result Codes

PGW-C handles various Diameter result codes in CCA messages:

Success Codes:

Code	Name	Action
2001	DIAMETER_SUCCESS	Continue session establishment

Permanent Failures (5xxx):

Code	Name	PGW-C Action
5002	DIAMETER_UNKNOWN_SESSION_ID	Log error, fail session
5030	DIAMETER_USER_UNKNOWN	Reject session (User Unknown)
5140	DIAMETER_ERROR_INITIAL_PARAMETERS	Log error, retry or fail
5003	DIAMETER_AUTHORIZATION_REJECTED	Reject session (Not Authorized)

Transient Failures (4xxx):

Code	Name	PGW-C Action
4001	DIAMETER_AUTHENTICATION_REJECTED	Retry or fail session
4010	DIAMETER_TOO_BUSY	Retry with backoff
4012	DIAMETER_UNABLE_TO_COMPLY	Log error, may retry

Experimental Result Codes

Vendor-specific error codes:

```
Experimental-Result (Grouped)
├── Vendor-Id: 10415 (3GPP)
└── Experimental-Result-Code: <vendor-specific code>
```

Common 3GPP Experimental Codes:

Code	Name	Meaning
5065	IP_CAN_SESSION_NOT_AVAILABLE	PCRF cannot establish session
5143	INVALID_SERVICE_INFORMATION	Service data invalid

Timeout Handling

CCR-I Timeout:

If PCRF doesn't respond to CCR-Initial within timeout:

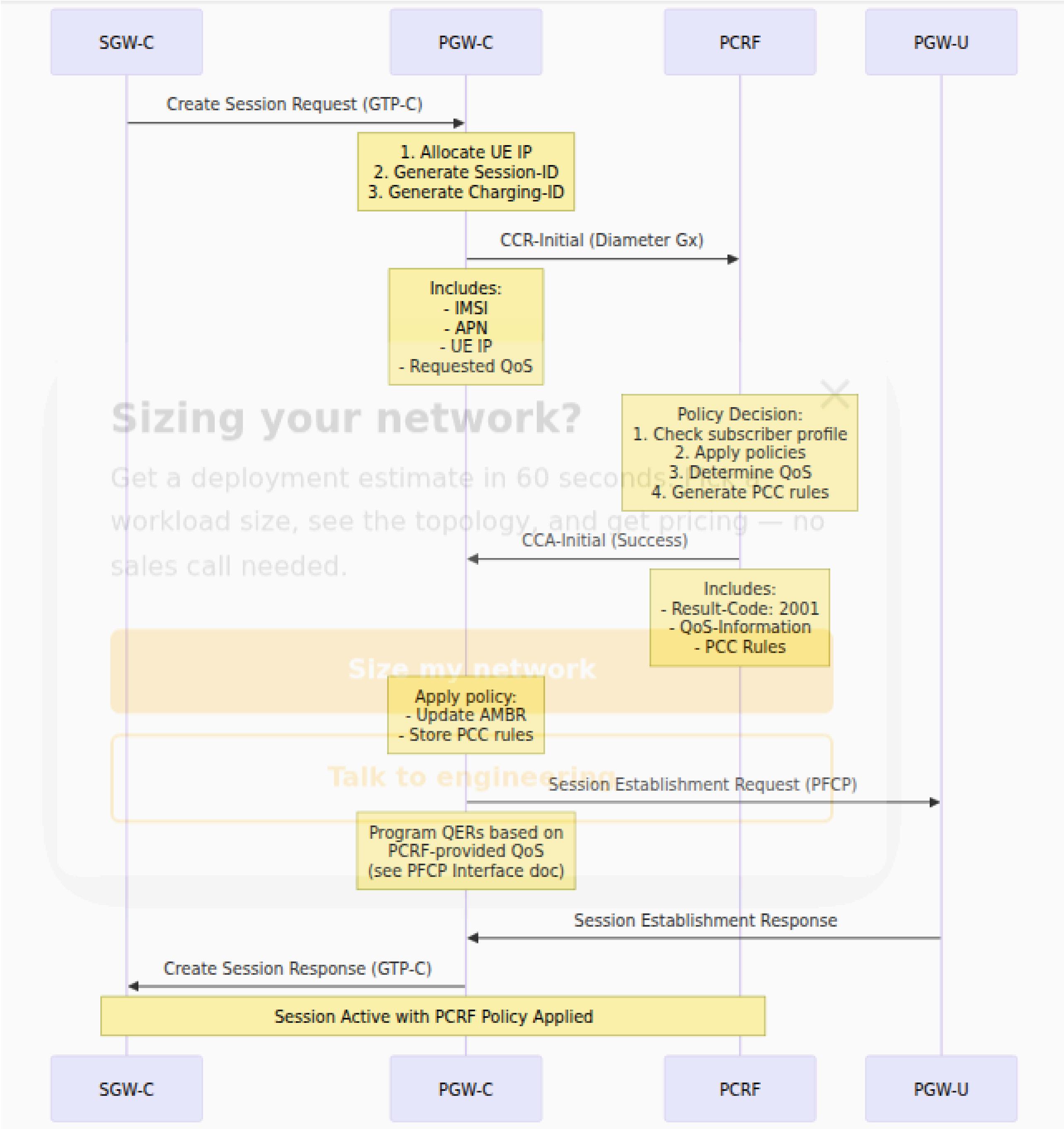
```
1. PGW-C waits for configured timeout (e.g., 5 seconds)
2. If no CCA received:
   - Log: "CCR-Initial timeout for Session-ID: ..."
   - Respond to SGW-C with error cause
   - Clean up allocated resources
3. SGW-C receives: Create Session Response (Cause: Remote Peer Not Responding)
```

Error Response to SGW-C:

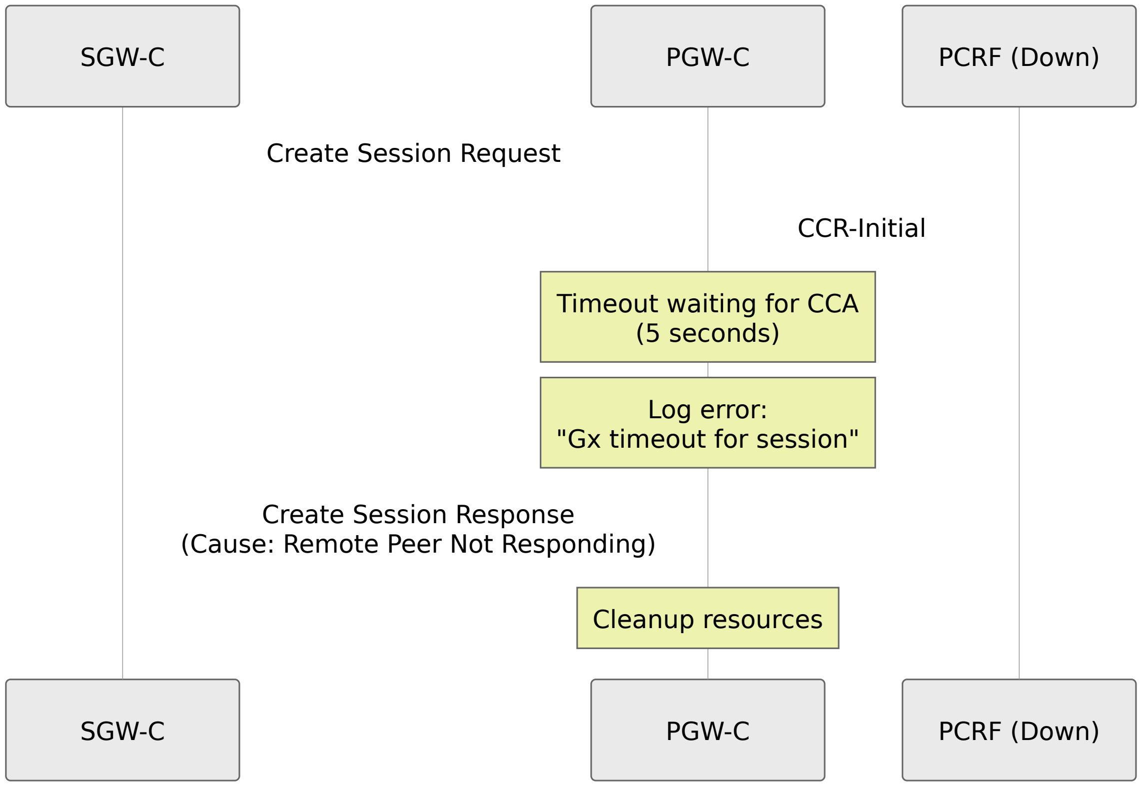
When CCR-Initial times out, the PGW-C sends a Create Session Response with cause code `:remote_peer_not_responding` to the SGW-C.

Failure Scenarios

Scenario 1: PCRF Rejects Session (User Unknown)



Scenario 2: PCRF Temporarily Unavailable



Troubleshooting

Common Issues

1. Diameter Peer Connection Fails

Symptoms:

- Log: "Diameter peer not connected"
- No CCR-Initial sent

Possible Causes:

- PCRF not reachable
- Incorrect PCRF IP in configuration
- Firewall blocking Diameter port (3868)
- Incorrect Diameter identities (host/realm)

Resolution:

```
# Test network connectivity
ping <pcrf_ip>

# Test Diameter port (TCP 3868)
telnet <pcrf_ip> 3868

# Check Diameter identity configuration
# Ensure host and realm are FQDNs, not IPs
```

Verify Configuration:

```
config :pgw_c,
  diameter: %{
    # Must be FQDN, not IP
    host: "pgw_c.epc.mnc999.mcc999.3gppnetwork.org",
    realm: "epc.mnc999.mcc999.3gppnetwork.org",
    peer_list: [
      %{
        host: "pcrf.epc.mnc999.mcc999.3gppnetwork.org",
        ip: "10.0.0.30"
      }
    ]
  }
```

2. CCR-Initial Timeouts

Symptoms:

- Create Session Request fails
- Log: "CCR-Initial timeout"

Possible Causes:

- PCRF overloaded
- Network latency
- PCRF not responding to this Session-ID

Resolution:

1. Check PCRF logs for errors
2. Verify PCRF is processing requests
3. Check network latency: `ping <pcrf_ip>`
4. Increase timeout if network latency is high

3. Sessions Rejected by PCRF

Symptoms:

- CCA-Initial with Result-Code != 2001
- Create Session Response fails

Common Result Codes:

Result Code	Likely Cause	Resolution
5030	IMSI not in subscriber database	Provision subscriber in HSS/SPR
5003	Authorization rejected	Check subscriber permissions
4010	PCRF too busy	Retry or add PCRF capacity

Check Logs:

```
# PGW-C logs show:
[error] Diameter Gx error: Result-Code 5030
(DIAMETER_USER_UNKNOWN)
[error] IMSI 310260999999999 rejected by PCRF
```

4. QoS Not Applied

Symptoms:

- Session established but wrong QoS

- Bitrates don't match expected values

Debugging Steps:

1. **Check CCA-Initial:**
   - Verify `QoS-Information` AVP present
   - Check `APN-Aggregate-Max-Bitrate-UL/DL` values
2. **Check PFCP Session Establishment:**
   - Verify QER created with correct MBR values
   - Check PGW-U logs for QER installation
3. **Verify PCRF Policy:**
   - Check PCRF configuration
   - Verify subscriber profile includes correct QoS

5. Diameter Routing Issues

Symptoms:

- Diameter messages not reaching PCRF
- Log: "No route to Destination-Realm"

Cause:

- Realm mismatch between configuration and messages

Resolution:

Ensure consistency:

```
# All must match
config :pgw_c,
  diameter: %{
    realm: "epc.mnc999.mcc999.3gppnetwork.org",  # PGW-C's realm
    peer_list: [
      %{
        realm: "epc.mnc999.mcc999.3gppnetwork.org"  # PCRF's realm
(usually same)
      }
    ]
  }
```

In CCR-Initial:

```
Origin-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
Destination-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
```

Monitoring Gx Health

Key Metrics:

```
# Gx message rates
rate(gx_inbound_messages_total{message_type="gx_CCA"}[5m])
rate(gx_outbound_messages_total{message_type="gx_CCR"}[5m])

# Gx error rates
rate(gx_inbound_errors_total[5m])

# Gx response success rate (new metric)
sum(rate(gx_outbound_responses_total{result_code_class="2xxx"}
[5m])) /
sum(rate(gx_outbound_responses_total[5m])) * 100

# Gx response failures by PCRF host
rate(gx_outbound_responses_total{result_code_class!="2xxx"}[5m])
by (diameter_host)

# Gx session count
session_id_registry_count

# Gx message handling duration
histogram_quantile(0.95,
rate(gx_inbound_handling_duration_bucket[5m]))
```

Response Metrics by Result Code Class:

The `gx_outbound_responses_total` metric provides detailed visibility into Diameter responses sent to PCRF peers, categorized by:

- **`message_type`**: Response message type (`gx_RAA`, `gx_CCA`)
- **`result_code_class`**: Result code category (`2xxx`, `3xxx`, `4xxx`, `5xxx`)
- **`diameter_host`**: PCRF peer receiving the response

Common Result Codes:

- **2001** (DIAMETER_SUCCESS) - Successful response
- **3001** (DIAMETER_COMMAND_UNSUPPORTED) - Protocol error
- **5012** (DIAMETER_UNABLE_TO_COMPLY) - Cannot execute request
- **5030** (DIAMETER_USER_UNKNOWN) - Subscriber not found

Alert Examples:

```
# Alert on high Gx error rate
- alert: GxErrorRateHigh
  expr: rate(gx_inbound_errors_total[5m]) > 0.1
  for: 5m
  annotations:
    summary: "High Gx error rate detected"

# Alert on high Gx response failure rate
- alert: GxResponseFailureRate
  expr: |

sum(rate(gx_outbound_responses_total{result_code_class!="2xxx"}
[5m])) /
    sum(rate(gx_outbound_responses_total[5m])) > 0.1
  for: 5m
  annotations:
    summary: "High Gx response failure rate"
    description: "More than 10% of Gx responses are failures"

# Alert on PCRF-specific failures
- alert: GxPCRFFailures
  expr:
rate(gx_outbound_responses_total{result_code_class=~"4xxx|5xxx"}
[5m]) by (diameter_host) > 0.05
  for: 3m
  annotations:
    summary: "PCRF {{ $labels.diameter_host }} receiving failure
responses"
    description: "High failure rate for PCRF host"

# Alert on session rejection
- alert: GxSessionRejection
  expr: rate(gx_inbound_errors_total{result_code="5030"}[5m]) >
0.01
  for: 5m
  annotations:
    summary: "PCRF rejecting sessions (USER_UNKNOWN)"
```

Debug Logging

Enable verbose Diameter logging:

```
# config/runtime.exs
config :logger, level: :debug

# Or at runtime
iex> Logger.configure(level: :debug)
```

Look for:

- `[debug] Sending CCR-Initial for Session-ID: ...`
- `[debug] Received CCA-Initial: Result-Code 2001`
- `[error] Diameter error: ...`

Diameter Peer Monitoring

Diameter Gx peer connectivity to the PCRF is monitored through the **OAM REST API**. The API is served over HTTPS/TLS at `https://<host>:8443` with all routes under the `/api` prefix. Interactive API documentation (Swagger UI) is available at `https://<host>:8443/api/docs`.

Diameter Peer Endpoints

```
GET /api/diameter                        # all configured peers +
live state
GET /api/diameter?status=disconnected # filter by connection state
GET /api/diameter/<origin-host>       # one peer by Origin-Host
```

Example:

```
curl -k https://localhost:8443/api/diameter
```

`GET /api/diameter` returns every configured Diameter peer and its current connection state. Add `?status=connected` or `?status=disconnected` to filter by connection state, or append the peer's Origin-Host to `GET /api/diameter/<origin-host>` to fetch a single peer.

For each peer the following fields are returned:

Field	Description
`host`	Diameter Identity (Origin-Host) of the peer
`status`	Connection state (e.g. Connected / Disconnected / Connecting)
`ip`	Peer IP address
`port`	Peer port
`transport`	TCP or SCTP
`connection_initiation`	Whether OmniPGW initiates (outbound) or accepts (inbound) the connection
`realm`	Diameter Realm reported by the peer in CEA
`product_name`	Vendor product string reported by the peer in CEA
`application_ids`	Diameter application IDs negotiated in CEA (Gx = 16777238)

A healthy deployment shows all configured PCRF peers as **Connected**. Loss of all Gx peers prevents policy install and blocks new session creation.

Operational Use Cases

Monitor PCRF Connectivity:

```
curl -k https://localhost:8443/api/diameter
```

Verify all PCRF peers show `Connected`, that `connection_initiation` matches configuration, and that `application_ids` include Gx (16777238).

Troubleshoot Session Creation Failures (Gx Issues): When user sessions fail with "PCRF timeout" errors, query the affected peer:

```
curl -k https://localhost:8443/api/diameter/<origin-host>
```

- `status` is `Disconnected` → check network connectivity, verify the PCRF is running, and confirm firewall rules allow TCP 3868.
- `status` is `Connected` but sessions still fail → the issue is at the application level (check logs); the PCRF may be rejecting subscribers.

Verify Diameter Configuration: After configuring a new PCRF peer, confirm it appears and reaches `Connected`:

```
curl -k https://localhost:8443/api/diameter/<origin-host>
```

Check that `realm` matches configuration, `application_ids` include Gx, and `product_name` shows the expected PCRF identifier.

Monitor Failover: When a primary PCRF fails, `GET /api/diameter?status=disconnected` lists it as `Disconnected` while the backup PCRF remains `Connected`. New sessions automatically use the backup; when the primary recovers its `status` returns to `Connected`.

Detect Diameter Routing Issues:

- Peer shows `Connected` but `realm` is wrong
- `application_ids` does not include Gx (16777238)
- `product_name` does not match the expected PCRF

Identify Configuration Mismatches: If `connection_initiation` reports the peer initiating while the configuration sets `initiate_connection: true`, both ends are attempting to initiate, which may cause connection race conditions.

Integration with Metrics

The OAM API provides real-time peer state; combine it with Prometheus for:

- Historical Gx error rates
- CCR/CCA message counts
- Latency trends

`GET /api/diameter` answers "Is it working right now?"; metrics answer "How has it been working over time?"

Related Documentation

Configuration and Policy

- **Configuration Guide** - Diameter configuration, PCRF peer setup
- **PFCP Interface** - QoS enforcement via QERs from PCC rules
- **Session Management** - Session lifecycle with policy integration
- **QoS & Bearer Management** - Detailed QoS configuration and bearer setup

Charging Integration

- **Diameter Gy Interface** - Online charging triggered by PCC rules
- **Data CDR Format** - Offline charging records with policy info
- **PCO Configuration** - P-CSCF delivery for IMS policy control

Operations

- **Monitoring Guide** - Gx metrics, policy tracking, PCRF connectivity alerts
- **S5/S8 Interface** - Bearer management integration with policy

---

Back to Operations Guide

Diameter Online Charging (Gy/Ro Interface)

Online Charging System (OCS) Interface

Table of Contents

Overview

The **Gy interface** (also called **Ro interface** in IMS contexts) connects PGW-C to the **Online Charging System (OCS)** for real-time credit control. This enables:

- **Prepaid Charging** - Real-time credit authorization and deduction

- **Real-time Credit Control** - Grant quota before service delivery
- **Service-Based Charging** - Different charging for voice, data, SMS, etc.
- **Immediate Account Updates** - Credit balance updates in real-time
- **Service Denial** - Block service when credit exhausted

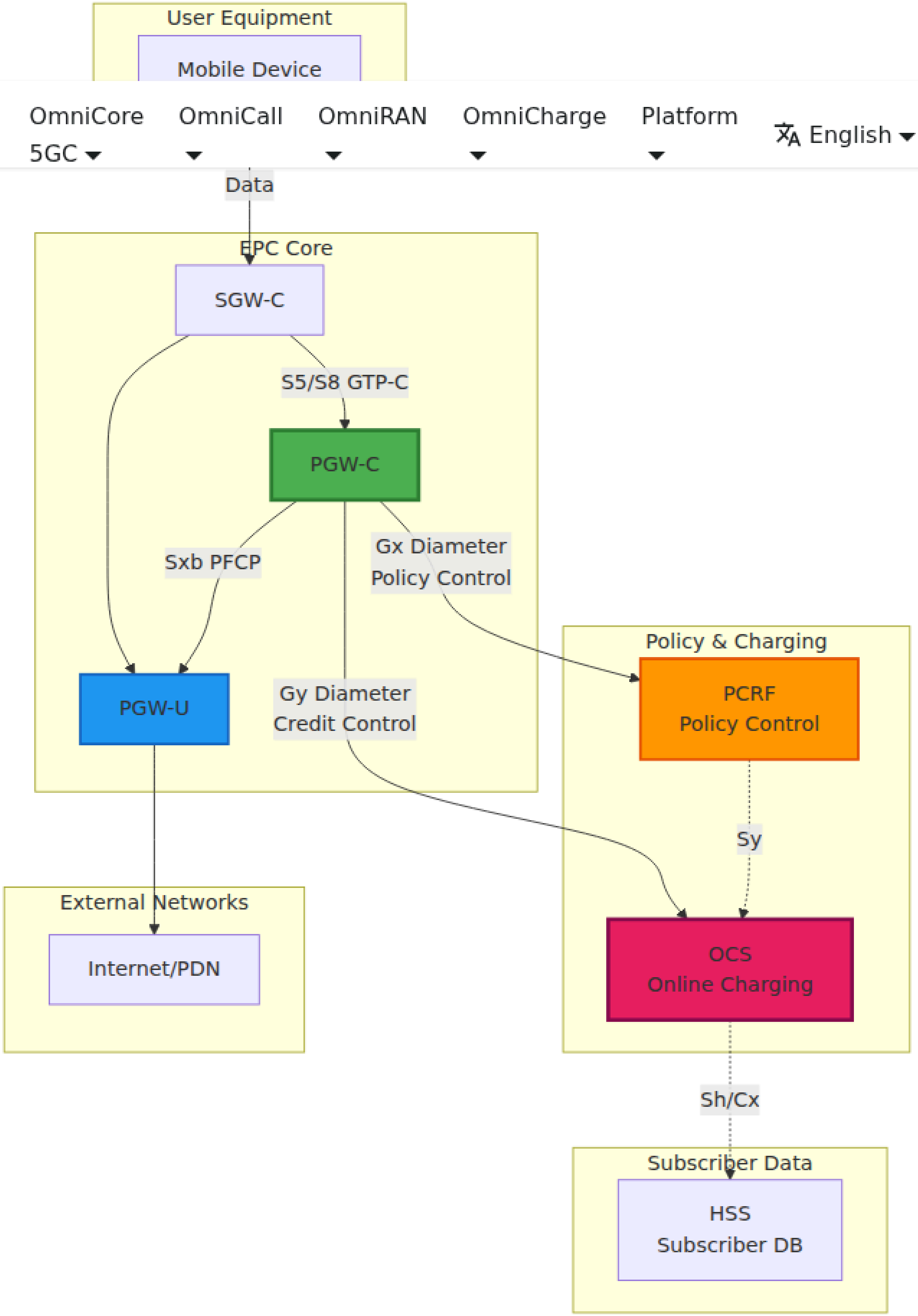
Online vs. Offline Charging

Aspect	Online Charging (Gy/Ro)	Offline Charging (Gz/Rf)
Timing	Real-time, before service	After service delivery
Use Case	Prepaid subscribers	Postpaid subscribers
Credit Check	Yes, before granting service	No, bill generated later
System	OCS (Online Charging System)	CGF/CDF (Charging Data Function)
Risk	No revenue loss	Risk of unpaid bills
Complexity	High (real-time requirements)	Lower (batch processing)
User Impact	Service denied if no credit	Service always available

See also: Data CDR Format for offline charging records (postpaid billing)

See also: Session Management for complete PDN session lifecycle including charging integration

Gy in the Network Architecture

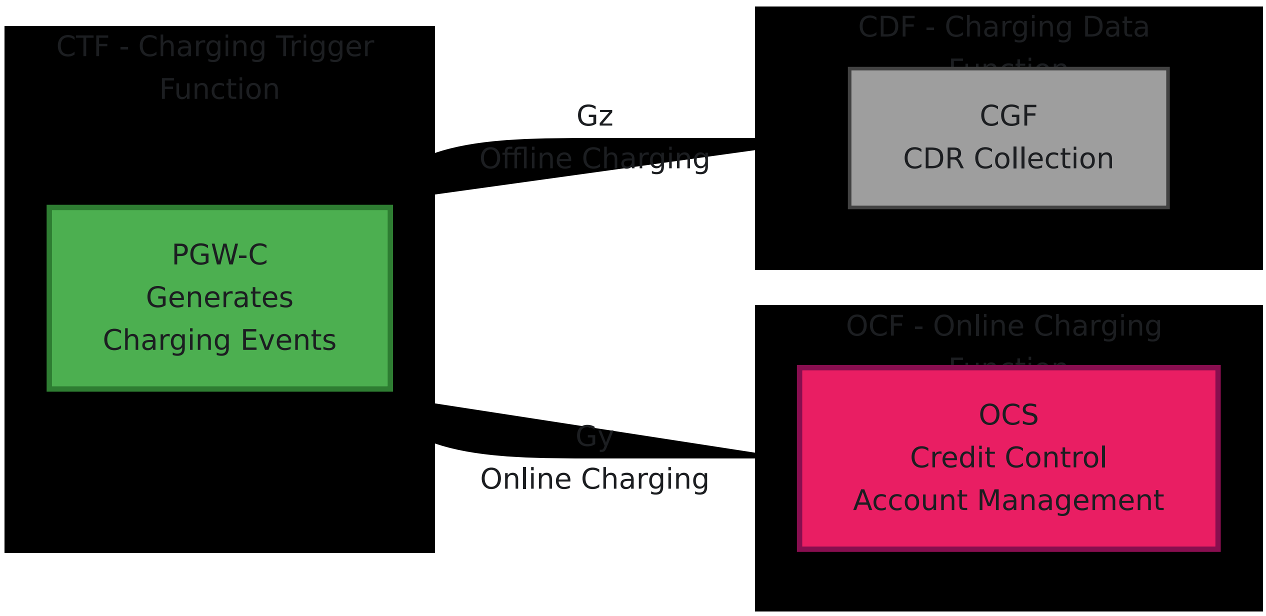

User Equipment
Mobile Device
Data
EPC Core
SGW-C
S5/S8 GTP-C
PGW-C
Sxb PFCP
Gx Diameter
Policy Control
PGW-U
Gy Diameter
Credit Control
Policy & Charging
PCRF
Policy Control
Sy
OCS
Online Charging
External Networks
Internet/PDN
Sh/Cx
Subscriber Data
HSS
Subscriber DB

Key Functions

Function	Description
Credit Authorization	Request quota from OCS before allowing traffic
Quota Management	Track granted units (bytes, time, events)
Credit Depletion Detection	Monitor remaining quota
Re-authorization	Request additional quota when threshold reached
Service Termination	Stop service when credit exhausted
Final Settlement	Report actual usage upon session end

3GPP Charging Architecture

Charging Reference Points



Charging Trigger Function (CTF)

PGW-C acts as a **CTF (Charging Trigger Function)**, responsible for:

1. **Detecting chargeable events** - Session start, data usage, session end
2. **Requesting credit authorization** - Before allowing service
3. **Tracking quota consumption** - Monitor granted units
4. **Generating charging events** - Trigger credit requests
5. **Enforcing credit control** - Block traffic when quota exhausted

Online Charging Function (OCF)

The OCS implements the **OCF (Online Charging Function)**:

1. **Account balance management** - Track subscriber credit
2. **Rating** - Determine price per unit (per MB, per second, etc.)
3. **Credit reservation** - Reserve credit for granted quota
4. **Credit deduction** - Deduct upon usage report

5. **Policy decisions** - Grant or deny based on balance

Gy/Ro Interface Basics

3GPP Reference

- **Specification:** 3GPP TS 32.299 (Charging architecture)
- **Protocol:** 3GPP TS 32.251 (PS domain charging)
- **Diameter Application ID:** 4 (Gy/Ro - Credit Control Application)
- **Base Protocol:** RFC 4006 (Diameter Credit Control Application)

Session Concept

Each UE PDN connection requiring online charging has a **Gy/Ro session** identified by a **Session-ID**. This session:

- Created when bearer requires online charging (CCR-Initial)
- Updated when quota is consumed (CCR-Update)
- Terminated when session ends (CCR-Termination)

Session ID Format

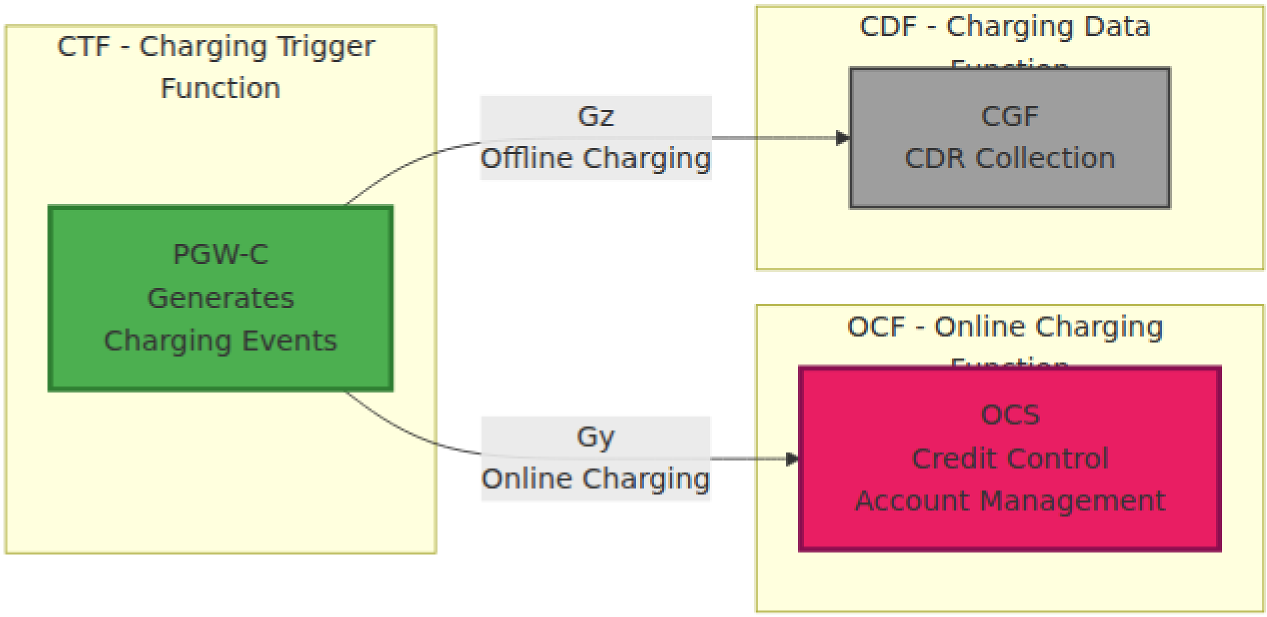
```
Session-ID: <Origin-Host>;<high32>;<low32>[;<optional>]
Example: omni-
pgw_c.epc.mnc999.mcc999.3gppnetwork.org;9876543210;12345;gy
```

Components:

- **Origin-Host:** PGW-C's Diameter identity
- **high32:** High 32 bits of unique identifier
- **low32:** Low 32 bits of unique identifier
- **optional:** Additional identifier (e.g., "gy" to distinguish from Gx)

Credit Control Messages

Message Types



CCR-Initial (Credit Control Request - Initial)

When: UE creates a PDN connection and bearer requires online charging

Purpose:

- Request initial credit authorization from OCS
- Reserve quota for service delivery
- Establish Gy/Ro session

Key AVPs Sent by PGW-C:

AVP Name	AVP Code	Type	Description
Session-Id	263	UTF8String	Unique Gy session identifier
Auth-Application-Id	258	Unsigned32	4 (Credit Control)
Origin-Host	264	DiamIdent	PGW-C's Diameter identity
Origin-Realm	296	DiamIdent	PGW-C's Diameter realm
Destination-Realm	283	DiamIdent	OCS's realm
CC-Request-Type	416	Enumerated	1 = INITIAL_REQUEST
CC-Request-Number	415	Unsigned32	Sequence number (starts at 0)
Subscription-Id	443	Grouped	UE identifier (IMSI/MSISDN)
Service-Context-Id	461	UTF8String	Charging context identifier
Multiple-Services-Credit-Control	456	Grouped	Service-specific credit requests
Requested-Service-Unit	437	Grouped	Requested quota (bytes, time, etc.)
Used-Service-Unit	446	Grouped	Used quota (0 for initial)
Service-Identifier	439	Unsigned32	Service type identifier

AVP Name	AVP Code	Type	Description
Rating-Group	432	Unsigned32	Charging category identifier

Example CCR-I Structure:

```
CCR (Command Code: 272, Request)
├── Session-Id: "pgw_c.example.com;123;456;gy"
├── Auth-Application-Id: 4
├── Origin-Host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org"
├── Origin-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── Destination-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── CC-Request-Type: INITIAL_REQUEST (1)
├── CC-Request-Number: 0
├── Subscription-Id (Grouped)
│   ├── Subscription-Id-Type: END_USER_IMSI (1)
│   └── Subscription-Id-Data: "310260123456789"
├── Subscription-Id (Grouped)
│   ├── Subscription-Id-Type: END_USER_E164 (0)
│   └── Subscription-Id-Data: "15551234567"
├── Service-Context-Id: "32251@3gpp.org"
├── Multiple-Services-Credit-Control (Grouped)
│   ├── Service-Identifier: 1
│   ├── Rating-Group: 100
│   └── Requested-Service-Unit (Grouped)
│       └── CC-Total-Octets: 10000000 (request 10 MB)
└── Used-Service-Unit (Grouped)
    └── CC-Total-Octets: 0 (no usage yet)
```

CCA-Initial (Credit Control Answer - Initial)

Sent by: OCS in response to CCR-I

Purpose:

- Grant or deny credit authorization
- Provide quota for service delivery

- Specify rating and charging parameters

Key AVPs Received by PGW-C:

AVP Name	AVP Code	Description
Result-Code	268	Success (2001) or error code
Multiple-Services-Credit-Control	456	Service-specific credit grants
Granted-Service-Unit	431	Granted quota (bytes, time, etc.)
Validity-Time	448	Quota validity period (seconds)
Result-Code	268	Per-service result code
Final-Unit-Indication	430	Action when quota exhausted
Volume-Quota-Threshold	-	Threshold for re-authorization

Success Response Example:

```
CCA (Command Code: 272, Answer)
├── Session-Id: "pgw_c.example.com;123;456;gy"
├── Result-Code: DIAMETER_SUCCESS (2001)
├── Origin-Host: "ocs.example.com"
├── Origin-Realm: "example.com"
├── Auth-Application-Id: 4
├── CC-Request-Type: INITIAL_REQUEST (1)
├── CC-Request-Number: 0
└── Multiple-Services-Credit-Control (Grouped)
    ├── Result-Code: DIAMETER_SUCCESS (2001)
    ├── Service-Identifier: 1
    ├── Rating-Group: 100
    ├── Granted-Service-Unit (Grouped)
    │   └── CC-Total-Octets: 10000000 (granted 10 MB)
    ├── Validity-Time: 3600 (quota valid for 1 hour)
    └── Volume-Quota-Threshold: 8000000 (re-auth at 8 MB used,
80%)
```

CCR-Update (Credit Control Request - Update)

When:

- Granted quota threshold reached (e.g., 80% consumed)
- Validity time expires
- Service change requires re-authorization
- Tariff time change

Purpose:

- Request additional quota
- Report usage of previously granted quota
- Update charging parameters

Key Differences from CCR-I:

- `CC-Request-Type: UPDATE_REQUEST (2)`
- `CC-Request-Number` incremented
- `Used-Service-Unit` contains actual usage

- `Requested-Service-Unit` for more quota

Example CCR-U Structure:

```
CCR (Command Code: 272, Request)
├── Session-Id: "pgw_c.example.com;123;456;gy"
├── Auth-Application-Id: 4
├── Origin-Host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org"
├── Origin-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── Destination-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── CC-Request-Type: UPDATE_REQUEST (2)
├── CC-Request-Number: 1
└── Multiple-Services-Credit-Control (Grouped)
    ├── Service-Identifier: 1
    ├── Rating-Group: 100
    ├── Used-Service-Unit (Grouped)
    │   └── CC-Total-Octets: 8000000 (8 MB used so far)
    └── Requested-Service-Unit (Grouped)
        └── CC-Total-Octets: 10000000 (request another 10 MB)
```

CCA-Update (Credit Control Answer - Update)

Sent by: OCS in response to CCR-U

Purpose:

- Grant additional quota (if credit available)
- Acknowledge usage
- Update charging parameters

Possible Outcomes:

1. More Quota Granted:

```
CCA (Update)
└── Multiple-Services-Credit-Control
    ├── Result-Code: DIAMETER_SUCCESS (2001)
    ├── Granted-Service-Unit
    │   └── CC-Total-Octets: 10000000 (another 10 MB)
    └── Validity-Time: 3600
```

2. Final Quota (Credit Exhausted):

```
CCA (Update)
└── Multiple-Services-Credit-Control
    ├── Result-Code: DIAMETER_SUCCESS (2001)
    ├── Granted-Service-Unit
    │   └── CC-Total-Octets: 1000000 (only 1 MB left)
    └── Final-Unit-Indication
        └── Final-Unit-Action: TERMINATE (0)
```

3. No Credit Available:

```
CCA (Update)
├── Result-Code: DIAMETER_CREDIT_LIMIT_REACHED (4012)
└── Multiple-Services-Credit-Control
    ├── Result-Code: DIAMETER_CREDIT_LIMIT_REACHED (4012)
    └── Final-Unit-Indication
        └── Final-Unit-Action: TERMINATE (0)
```

CCR-Termination (Credit Control Request - Termination)

When:

- UE detaches
- PDN connection deleted
- Session terminated for any reason

Purpose:

- Final usage report
- Close Gy/Ro session
- Final settlement

Key Differences:

- `CC-Request-Type: TERMINATION_REQUEST (3)`
- `Used-Service-Unit` contains final usage
- No `Requested-Service-Unit` (no more quota needed)
- Includes `Termination-Cause`

Example CCR-T Structure:

```
CCR (Command Code: 272, Request)
├── Session-Id: "pgw_c.example.com;123;456;gy"
├── Auth-Application-Id: 4
├── Origin-Host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org"
├── Origin-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── Destination-Realm: "epc.mnc999.mcc999.3gppnetwork.org"
├── CC-Request-Type: TERMINATION_REQUEST (3)
├── CC-Request-Number: 5
├── Termination-Cause: DIAMETER_LOGOUT (1)
└── Multiple-Services-Credit-Control (Grouped)
    ├── Service-Identifier: 1
    ├── Rating-Group: 100
    └── Used-Service-Unit (Grouped)
        └── CC-Total-Octets: 18500000 (18.5 MB total usage)
```

CCA-Termination (Credit Control Answer - Termination)

Sent by: OCS in response to CCR-T

Purpose:

- Acknowledge session termination
- Complete accounting
- Release reserved credit

Example CCA-T:

```
CCA (Command Code: 272, Answer)
├── Session-Id: "pgw_c.example.com;123;456;gy"
├── Result-Code: DIAMETER_SUCCESS (2001)
├── Origin-Host: "ocs.example.com"
├── Origin-Realm: "example.com"
├── Auth-Application-Id: 4
├── CC-Request-Type: TERMINATION_REQUEST (3)
└── CC-Request-Number: 5
```

Online Charging Flows

Service Unit Types

The OCS can grant quota in different units:

Unit Type	AVP	Description	Use Case
Time	CC-Time	Seconds	Voice calls, session duration
Volume	CC-Total-Octets	Bytes (total up+down)	Data services
Volume (separate)	CC-Input-Octets, CC-Output-Octets	Bytes (separate)	Asymmetric charging
Service-Specific	CC-Service-Specific-Units	Custom units	SMS, MMS, API calls
Events	-	Counted events	Pay-per-use services

Quota Threshold Management

Problem: How does PGW-C know when to request more quota?

Solution: OCS provides a **Volume-Quota-Threshold** or **Time-Quota-Threshold**. PGW-C monitors usage via PFCP Session Reports from PGW-U (see PFCP Interface).

Example Flow:

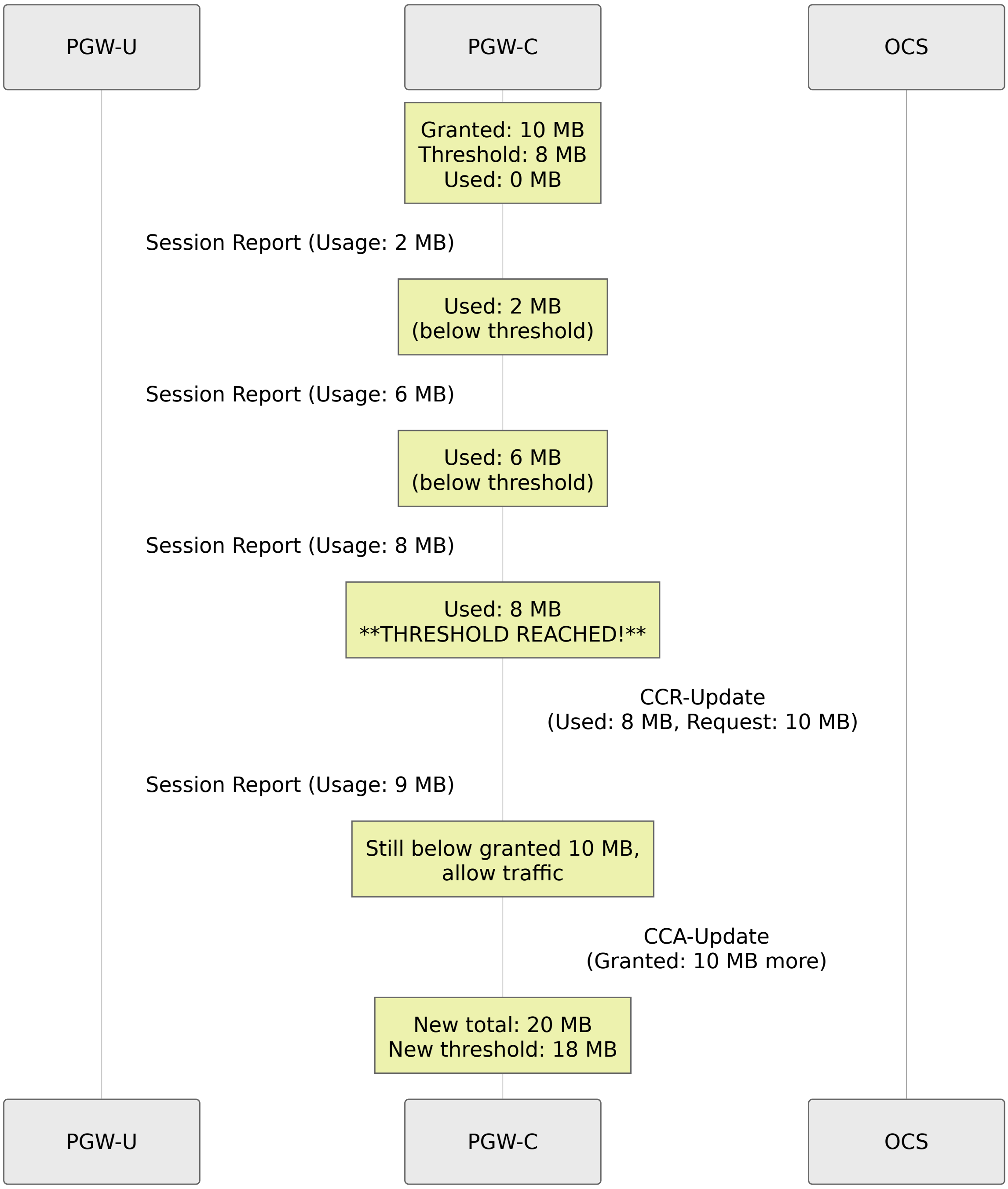
```
1. OCS grants 10 MB quota with 80% threshold (8 MB)
2. PGW-C monitors usage via PGW-U usage reports (PFCP Session
Reports)
3. When usage reaches 8 MB:
   → PGW-C sends CCR-Update
   → Continue allowing traffic (don't wait for response)
4. OCS responds with more quota
5. If quota exhausted before CCR-Update sent:
   → PGW-C must block traffic
```

Threshold Calculation:

```
Granted-Service-Unit: 10000000 bytes (10 MB)
Volume-Quota-Threshold: 8000000 bytes (8 MB)

When 8 MB consumed → Trigger CCR-Update
Remaining buffer: 2 MB (allows time for OCS response)
```

PGW-C Monitoring:

PGW-C monitors usage via **PFCP Session Reports** from PGW-U:



Final Unit Indication

What happens when credit is exhausted?

OCS includes **Final-Unit-Indication** AVP in CCA to specify action:

Final-Unit-Action	Value	PGW-C Behavior
TERMINATE	0	Block all traffic, initiate session termination
REDIRECT	1	Redirect traffic to portal (e.g., top-up page)
RESTRICT_ACCESS	2	Allow access only to specific services (e.g., top-up server)

Example: Final Unit with Redirect

```
CCA (Update)
└── Multiple-Services-Credit-Control
    ├── Result-Code: DIAMETER_SUCCESS (2001)
    ├── Granted-Service-Unit
    │   └── CC-Total-Octets: 1000000 (final 1 MB)
    └── Final-Unit-Indication
        ├── Final-Unit-Action: REDIRECT (1)
        └── Redirect-Server (Grouped)
            ├── Redirect-Address-Type: URL (2)
            └── Redirect-Server-Address:
"http://topup.example.com"
```

PGW-C Actions:

1. **TERMINATE:** Send CCR-T, delete bearer
2. **REDIRECT:** Install PFCP rule to redirect HTTP to top-up URL
3. **RESTRICT_ACCESS:** Install PFCP rules allowing only whitelisted IPs

---

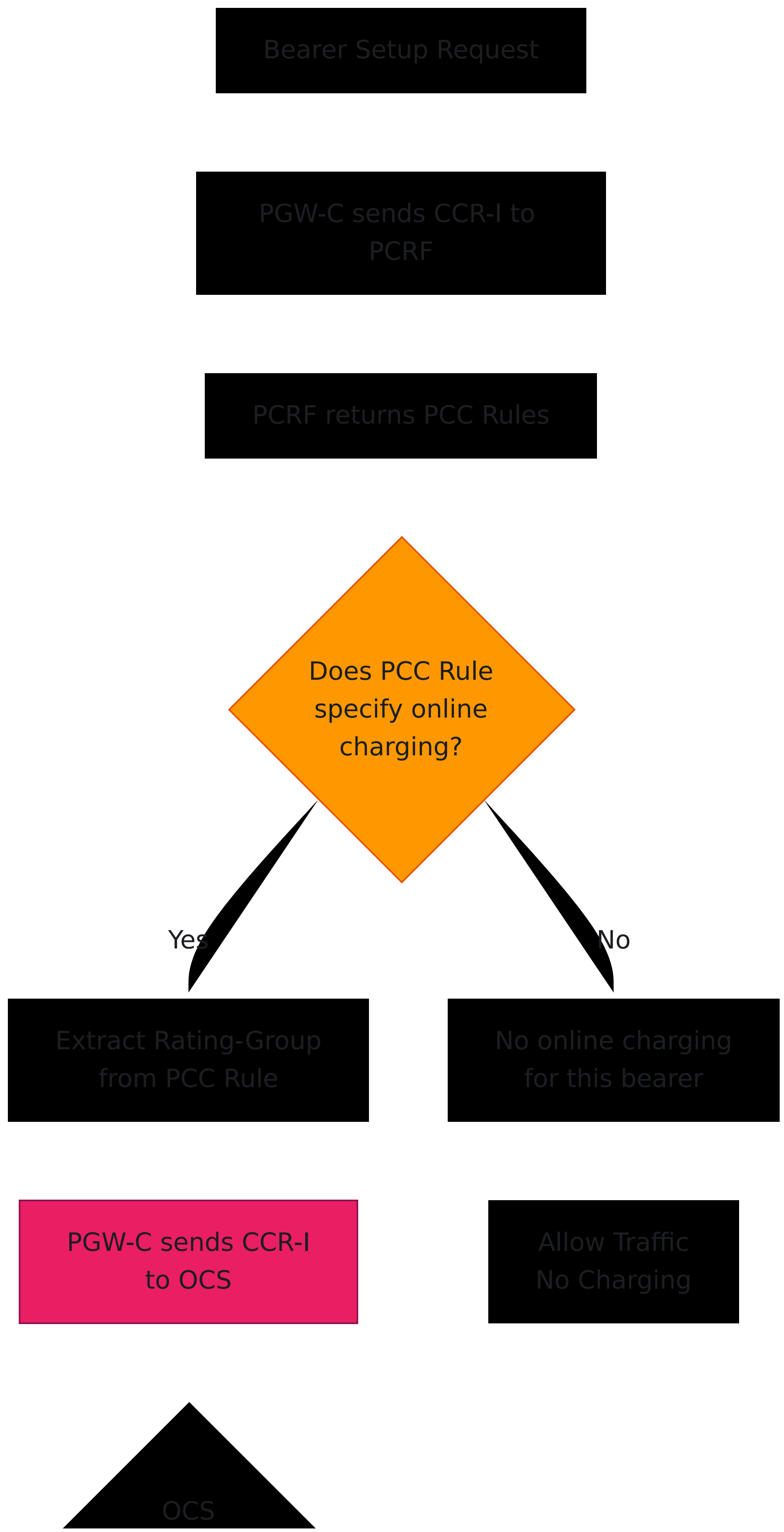
Bearer Charging Control

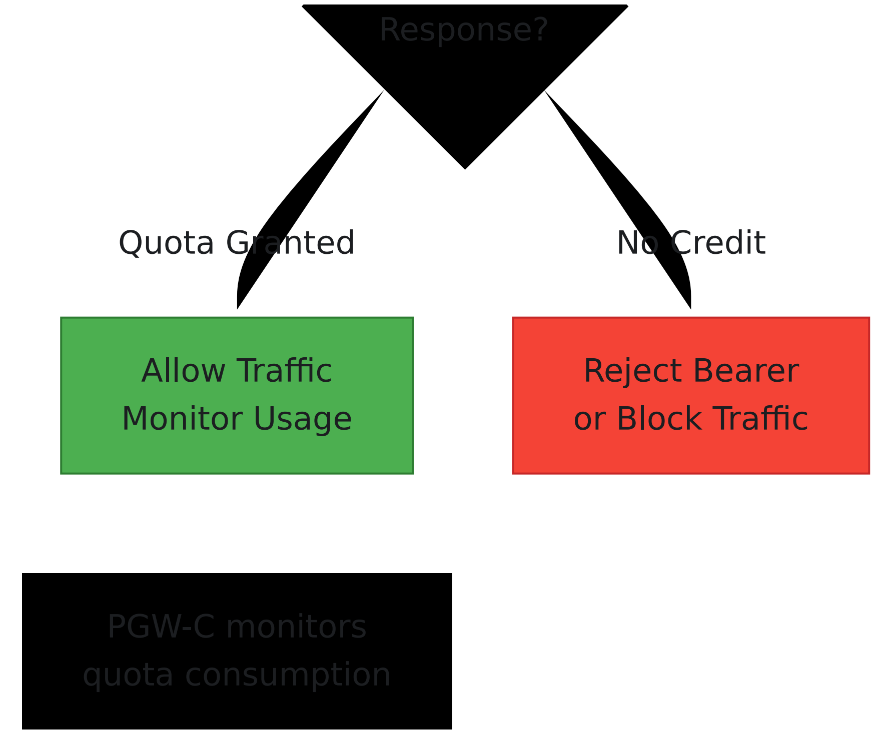
What Controls if a Bearer is Charged?

3GPP Specification: TS 23.203, TS 29.212, TS 32.251

Bearer charging is controlled by **PCC Rules** provisioned by the PCRF via the Gx interface. See Diameter Gx Interface for complete PCC rule documentation.

Charging Decision Flow:


Bearer Setup Request
PGW-C sends CCR-I to PCRF
PCRF returns PCC Rules
Does PCC Rule specify online charging?
Yes
No
Extract Rating-Group from PCC Rule
No online charging for this bearer
PGW-C sends CCR-I to OCS
Allow Traffic No Charging
OCS



PCC Rule with Charging Information

PCRF Response (CCA-I on Gx):

```
CCA (Gx Interface)
└── Charging-Rule-Definition (Grouped)
    ├── Charging-Rule-Name: "prepaid_data_rule"
    ├── Rating-Group: 100
    ├── Online: 1 (enable online charging)
    ├── Offline: 0 (disable offline charging)
    ├── Metering-Method: VOLUME (1)
    ├── Precedence: 100
    ├── Flow-Information: [...]
    └── QoS-Information: [...]
```

Key Charging AVPs in PCC Rules:

AVP Name	AVP Code	Values	Description
Rating-Group	432	Unsigned32	Charging category (maps to tariff in OCS)
Online	1009	0=Disable, 1=Enable	Enable online charging (Gy)
Offline	1008	0=Disable, 1=Enable	Enable offline charging (Gz)
Metering-Method	1007	0=Duration, 1=Volume, 2=Both	What to meter
Reporting-Level	1011	0=Service, 1=Rating Group	Granularity of usage reports

Bearer Charging Decision Matrix

Online	Offline	Rating-Group	Behavior
1	0	Present	Online charging only (prepaid)
0	1	Present	Offline charging only (postpaid)
1	1	Present	Both online and offline (convergent)
0	0	-	No charging (free service)

Multiple Rating Groups

A single PDN connection can have **multiple bearers with different rating groups**:

Example Scenario:

```
Default Bearer (Internet)
├── Rating-Group: 100 (Standard Data)
└── Online: 1

Dedicated Bearer 1 (Video Streaming)
├── Rating-Group: 200 (Video Service)
└── Online: 1

Dedicated Bearer 2 (IMS Voice)
├── Rating-Group: 300 (Voice)
└── Online: 1
```

PGW-C Gy Behavior:

- **Single CCR-I** with multiple MSCC (Multiple-Services-Credit-Control) sections:

```
CCR-Initial
├── Session-Id: "..."
└── Multiple-Services-Credit-Control
    ├── [Rating-Group: 100] → Standard Data
    ├── [Rating-Group: 200] → Video Service
    └── [Rating-Group: 300] → Voice
```

OCS Response:

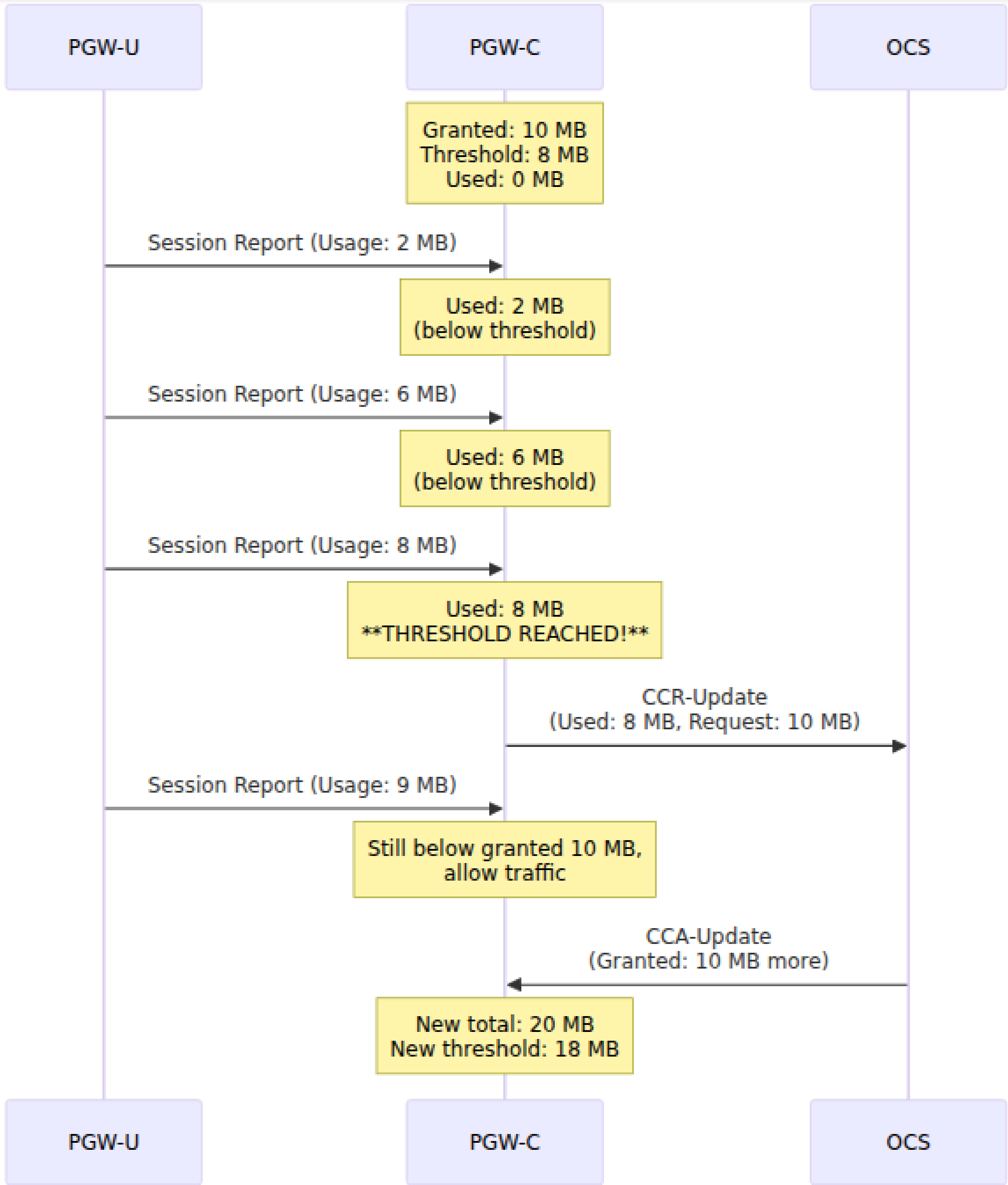
```
CCA-Initial
└── Multiple-Services-Credit-Control
    ├── [Rating-Group: 100] → Granted: 10 MB
    ├── [Rating-Group: 200] → Granted: 5 MB (video more expensive)
    └── [Rating-Group: 300] → Granted: 60 seconds
```

Per-Service Charging Enforcement

PGW-C tracks quota per Rating-Group:

```
# Pseudocode
state.charging_quotas = %{
  100 => %{granted: 10_000_000, used: 0, threshold: 8_000_000},
  200 => %{granted: 5_000_000, used: 0, threshold: 4_000_000},
  300 => %{granted: 60_000, used: 0, threshold: 48_000}  #
milliseconds
}
```

Usage Monitoring per Bearer:



Multiple Services Credit Control

MSCC (Multiple-Services-Credit-Control) AVP

Purpose: Group charging information for a specific service/rating group

Structure:

```
Multiple-Services-Credit-Control (Grouped, AVP 456)
├── Service-Identifier (Unsigned32, AVP 439)
├── Rating-Group (Unsigned32, AVP 432)
├── Requested-Service-Unit (Grouped, AVP 437)
│   ├── CC-Time (Unsigned32, AVP 420)
│   ├── CC-Total-Octets (Unsigned64, AVP 421)
│   ├── CC-Input-Octets (Unsigned64, AVP 412)
│   └── CC-Output-Octets (Unsigned64, AVP 414)
├── Used-Service-Unit (Grouped, AVP 446)
│   └── [Same structure as Requested-Service-Unit]
├── Granted-Service-Unit (Grouped, AVP 431)
│   └── [Same structure as Requested-Service-Unit]
├── Validity-Time (Unsigned32, AVP 448)
├── Result-Code (Unsigned32, AVP 268)
└── Final-Unit-Indication (Grouped, AVP 430)
    └── Final-Unit-Action (Enumerated, AVP 449)
```

Service-Identifier vs. Rating-Group

Attribute	Service-Identifier	Rating-Group
Purpose	Identifies service type	Identifies charging category
Example	1=Data, 2=Voice, 3=SMS	100=Regular, 200=Premium
Granularity	Broad classification	Specific tariff
Required	Optional	Required for charging
Mapping	May map to multiple RGs	Single tariff in OCS

Example:

```
Service-Identifier: 1 (Data Service)
├── Rating-Group: 100 (Standard Data - $0.01/MB)
└── Rating-Group: 200 (Premium Data - $0.05/MB)

Service-Identifier: 2 (Voice)
└── Rating-Group: 300 (Voice Calls - $0.10/min)
```

Configuration

Basic Gy Configuration

Edit `config/runtime.exs`:

```
config :pgw_c,
  gy: %{
    # Enable or disable online charging globally
    enabled: true,

    # OCS connection timeout (milliseconds)
    timeout_ms: 5000,

    # Default quota request (bytes) if not specified by PCRF
    default_requested_quota: 10_000_000,  # 10 MB

    # Threshold percentage for re-authorization
    # (0.8 = trigger CCR-Update at 80% quota consumed)
    quota_threshold_percentage: 0.8,

    # Action when OCS timeout occurs
    # Options: :block, :allow
    timeout_action: :block,

    # Action when OCS returns no credit
    # Options: :terminate, :redirect
    no_credit_action: :terminate,

    # Redirect URL for top-up (used if no_credit_action:
:redirect)
    topup_redirect_url: "http://topup.example.com"
  },
  diameter: %{
    listen_ip: "0.0.0.0",
    host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org",
    realm: "epc.mnc999.mcc999.3gppnetwork.org",

    # OCS peer configuration
    peer_list: [
      # PCRF for policy control (Gx)
      %{
        host: "pcrf.epc.mnc999.mcc999.3gppnetwork.org",
        realm: "epc.mnc999.mcc999.3gppnetwork.org",
        ip: "10.0.0.30",
        initiate_connection: true
      },
      # OCS for online charging (Gy)
      %{
```

```
      host: "ocs.epc.mnc999.mcc999.3gppnetwork.org",
      realm: "epc.mnc999.mcc999.3gppnetwork.org",
      ip: "10.0.0.40",
      initiate_connection: true
    }
  ]
}
```

Configuration Parameters Explained

`enabled`

- `true`: Online charging active, CCR messages sent to OCS
- `false`: Online charging disabled, no Gy messages

`timeout_ms`

- Time to wait for CCA response from OCS
- Recommended: 3000-5000 ms

`default_requested_quota`

- Default quota to request if PCRF doesn't specify
- Typical values: 1-100 MB

`quota_threshold_percentage`

- Trigger CCR-Update when this % of quota consumed
- Recommended: 0.75-0.85 (75%-85%)
- Higher = fewer messages, but risk of quota exhaustion
- Lower = more messages, but safer

`timeout_action`

- **`:block`** - Block traffic if OCS doesn't respond (safer, prevents revenue loss)
- **`:allow`** - Allow traffic if OCS doesn't respond (better UX, revenue risk)

`no_credit_action`

- `:terminate` - Delete bearer when credit exhausted
- `:redirect` - Redirect to top-up portal

Environment-Specific Configuration

Production (prepaid subscribers):

```
config :pgw_c,
  gy: %{
    enabled: true,
    timeout_action: :block,
    no_credit_action: :terminate,
    quota_threshold_percentage: 0.8
  }
```

Test/Development:

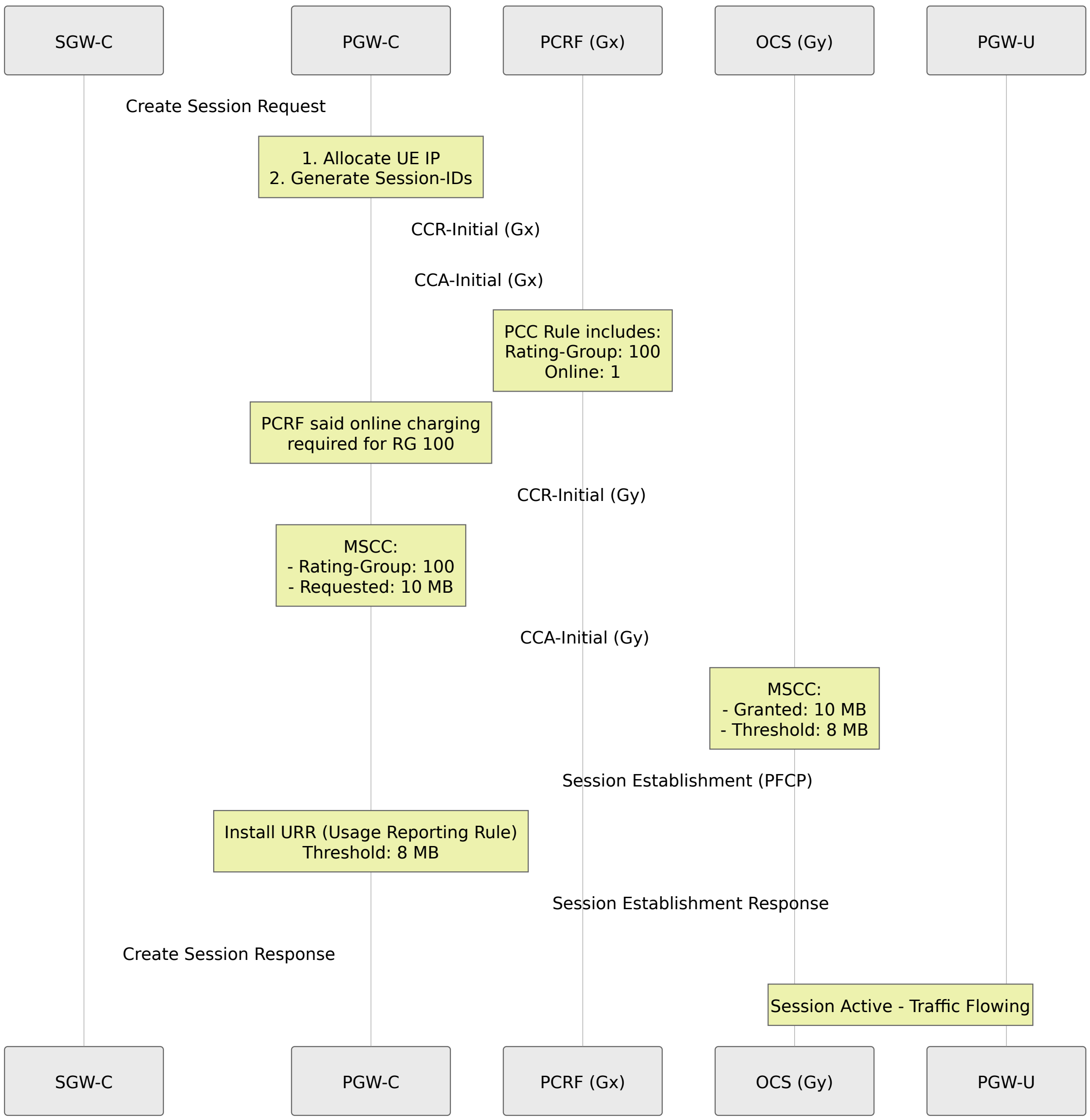
```
config :pgw_c,
  gy: %{
    enabled: false  # Disable for testing
  }
```

Hybrid (some prepaid, some postpaid):

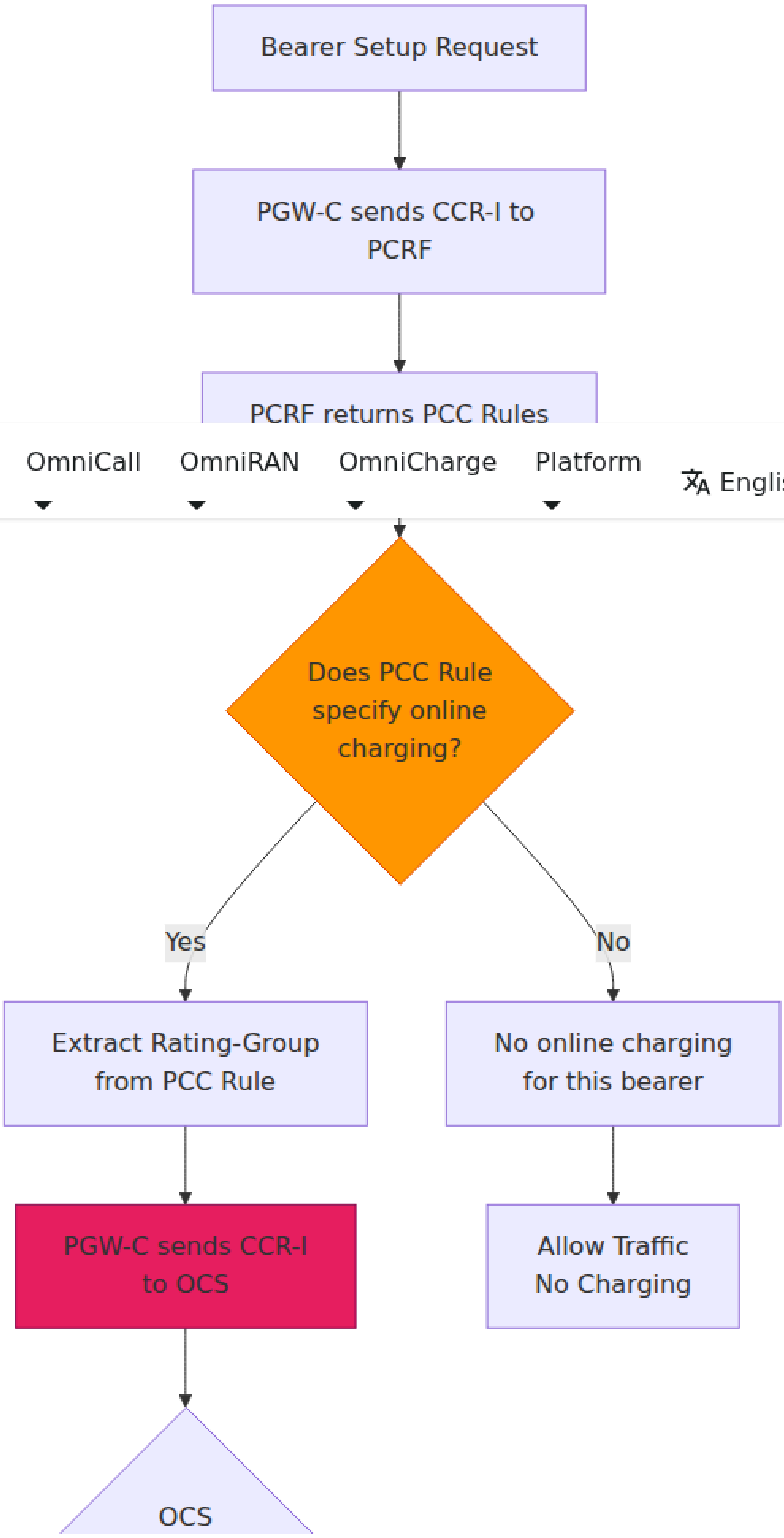
```
config :pgw_c,
  gy: %{
    enabled: true,  # Controlled per-subscriber by PCRF
    timeout_action: :allow,  # Don't block postpaid on OCS failure
    no_credit_action: :terminate
  }
```

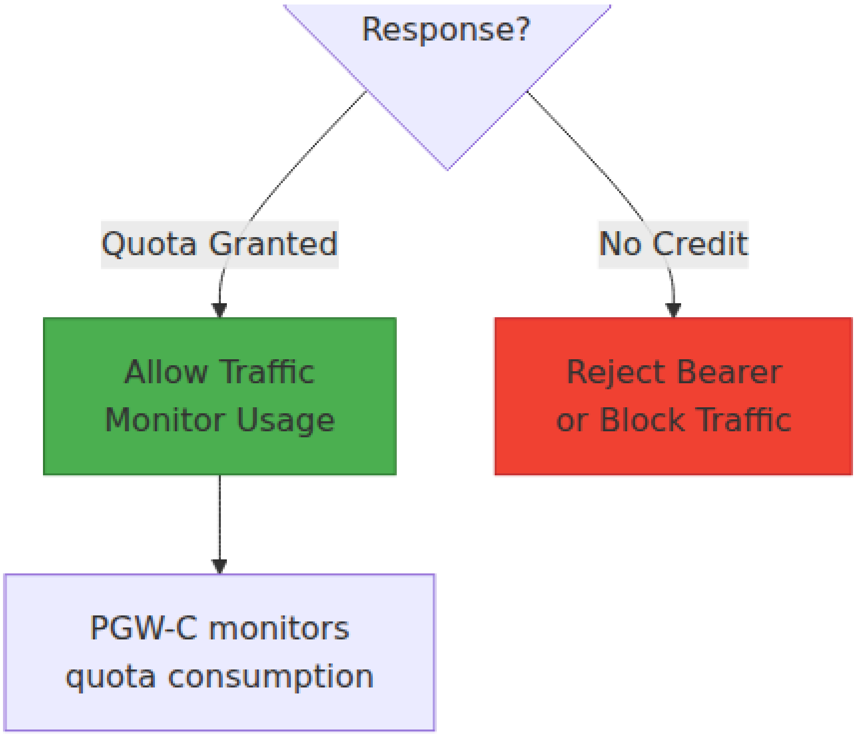
Message Flows

Successful Session with Online Charging

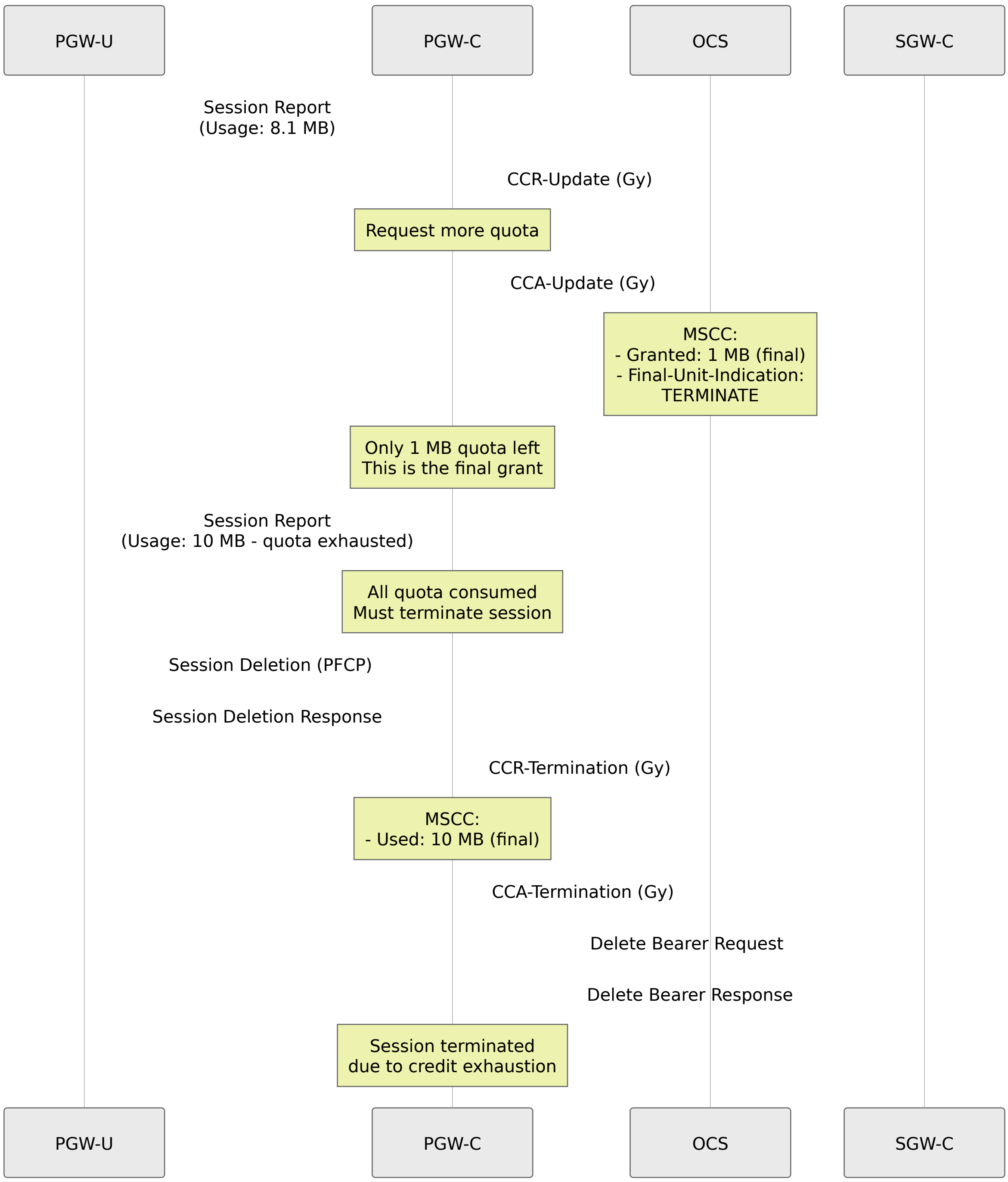


Quota Re-authorization (CCR-Update)

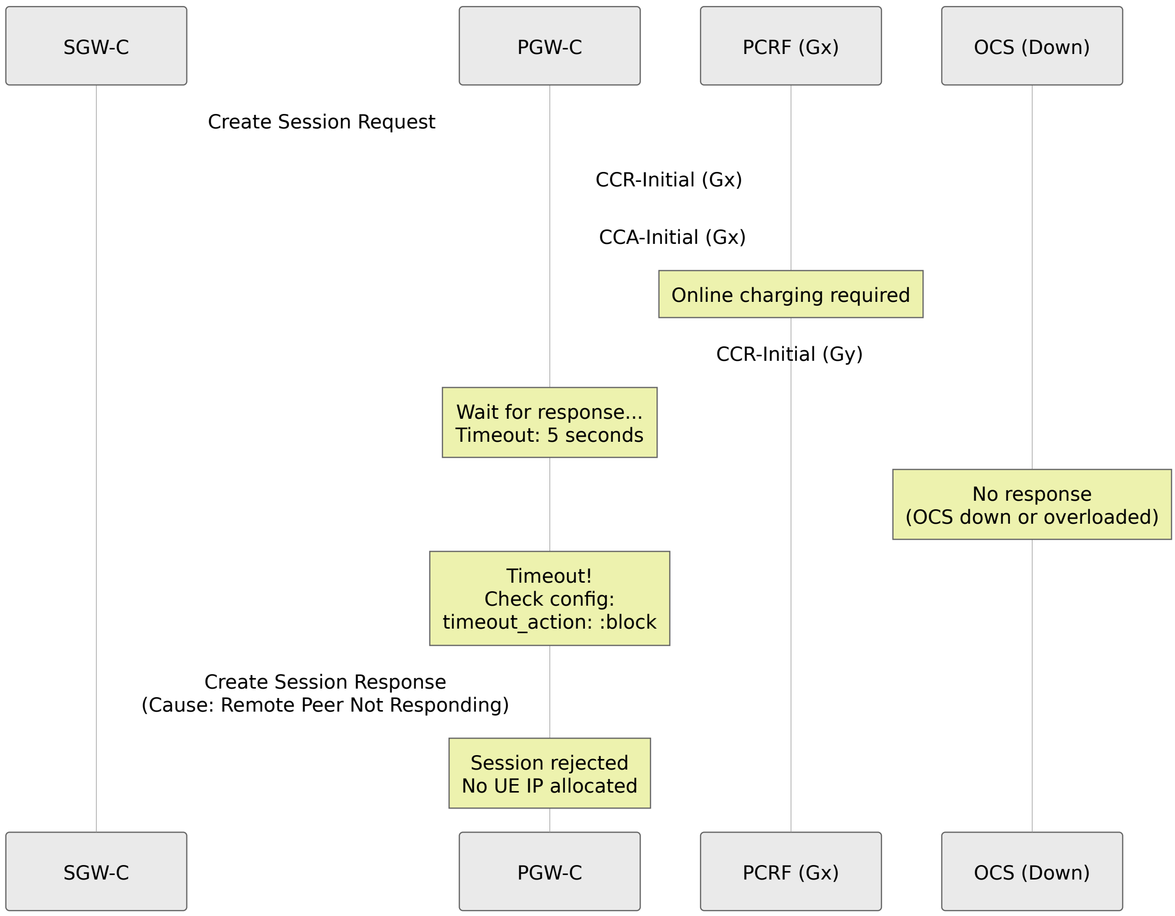

Bearer Setup Request
PGW-C sends CCR-I to PCRF
PCRF returns PCC Rules
Does PCC Rule specify online charging?
Yes
No
Extract Rating-Group from PCC Rule
No online charging for this bearer
PGW-C sends CCR-I to OCS
Allow Traffic No Charging
OCS



Credit Exhaustion (Final Unit)


PGW-U
PGW-C
OCS
SGW-C
Session Report
(Usage: 8.1 MB)
CCR-Update (Gy)
Request more quota
CCA-Update (Gy)
MSCC:
- Granted: 1 MB (final)
- Final-Unit-Indication:
TERMINATE
Only 1 MB quota left
This is the final grant
Session Report
(Usage: 10 MB - quota exhausted)
All quota consumed
Must terminate session
Session Deletion (PFCP)
Session Deletion Response
CCR-Termination (Gy)
MSCC:
- Used: 10 MB (final)
CCA-Termination (Gy)
Delete Bearer Request
Delete Bearer Response
Session terminated
due to credit exhaustion
PGW-U
PGW-C
OCS
SGW-C

OCS Timeout Handling



Error Handling

Result Codes

Success Codes:

Code	Name	Action
2001	DIAMETER_SUCCESS	Continue with granted quota

Transient Failures (4xxx):

Code	Name	PGW-C Action
4010	DIAMETER_TOO_BUSY	Retry with backoff
4011	DIAMETER_UNABLE_TO_COMPLY	Log error, may retry
4012	DIAMETER_CREDIT_LIMIT_REACHED	Terminate or redirect

Permanent Failures (5xxx):

Code	Name	PGW-C Action
5003	DIAMETER_AUTHORIZATION_REJECTED	Reject session
5031	DIAMETER_USER_UNKNOWN	Reject session (invalid subscriber)

Per-Service Result Codes

Important: Result-Code can appear at **two levels**:

1. **Message level** - Overall result
2. **MSCC level** - Per-service result

Example:

```
CCA-Initial
├── Result-Code: DIAMETER_SUCCESS (2001)  ← Message level: OK
└── Multiple-Services-Credit-Control
    ├── [Rating-Group: 100]
    │   └── Result-Code: DIAMETER_SUCCESS (2001)  ← RG 100: OK
    └── [Rating-Group: 200]
        └── Result-Code: DIAMETER_CREDIT_LIMIT_REACHED (4012)  ←
RG 200: No credit
```

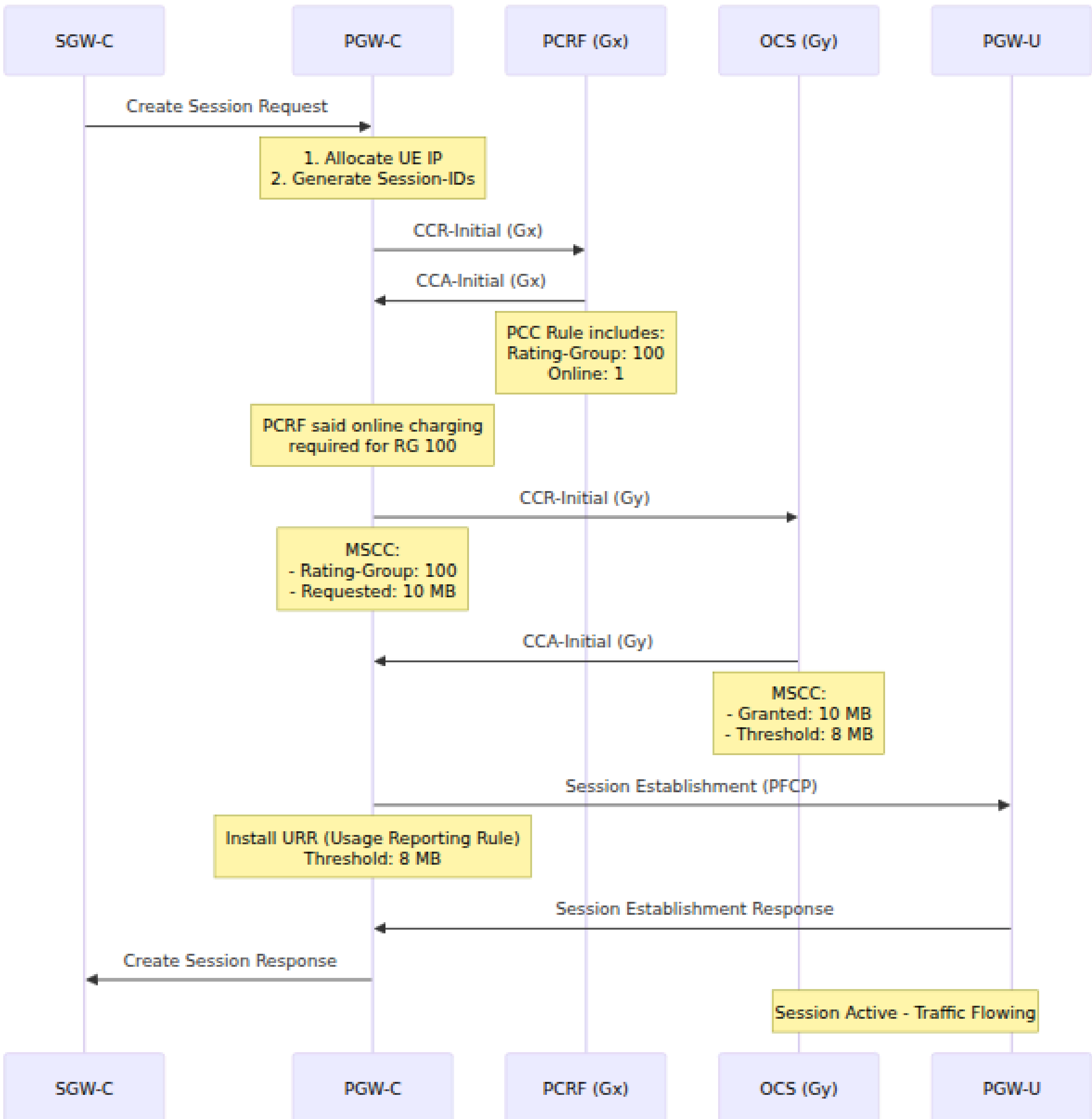
PGW-C Behavior:

- Allow traffic for Rating-Group 100
- Block traffic for Rating-Group 200

Integration with Gx

The Gx interface (PCRF policy control) determines whether online charging is required and provides the Rating-Group that drives Gy charging. See Diameter Gx Interface for complete policy control documentation.

Gx and Gy Relationship



Integration Flow

1. Bearer Setup:

```
PGW-C receives Create Session Request
    ↓
Send CCR-I to PCRF (Gx)
    ↓
Receive CCA-I with PCC Rules
    ↓
Parse PCC Rules:
    - Does rule have Rating-Group?
    - Is Online = 1?
    ↓
If YES:
    Send CCR-I to OCS (Gy) with Rating-Group
    ↓
    Receive CCA-I with quota
    ↓
    If quota granted: Proceed
    If no credit: Reject bearer
If NO:
    Proceed without online charging
```

2. Dynamic Policy Update (RAR from PCRF):

```
PCRF sends RAR (Re-Auth-Request) on Gx
    ↓
New PCC Rule added with Online=1, Rating-Group=200
    ↓
PGW-C sends CCR-U to OCS (Gy)
    - Add MSCC for Rating-Group 200
    ↓
OCS grants quota for new service
    ↓
Install dedicated bearer with online charging
```

Troubleshooting

Common Issues

1. CCR-Initial to OCS Timeouts

Symptoms:

- Sessions fail with "OCS timeout"
- Log: "CCR-Initial (Gy) timeout"

Possible Causes:

- OCS not reachable
- Incorrect OCS IP in configuration
- Firewall blocking Diameter port (3868)
- OCS overloaded

Resolution:

```
# Test network connectivity
ping <ocs_ip>

# Test Diameter port (TCP 3868)
telnet <ocs_ip> 3868

# Check configuration
# Ensure OCS peer is configured in peer_list
```

2. Sessions Rejected by OCS

Symptoms:

- CCA-I with Result-Code != 2001
- Create Session Response fails

Common Result Codes:

Result Code	Likely Cause	Resolution
4012	Credit limit reached	Subscriber needs to top-up
5003	Authorization rejected	Check subscriber permissions
5031	User unknown	Provision subscriber in OCS

Debug Steps:

1. Check OCS logs for rejection reason
2. Verify subscriber balance in OCS
3. Check IMSI/MSISDN in CCR-I matches subscriber record

3. Quota Exhaustion Not Detected

Symptoms:

- User continues using data after balance exhausted
- No CCR-Update sent

Possible Causes:

- URR (Usage Reporting Rule) not installed in PGW-U
- Threshold not configured correctly
- PFCP Session Reports not received

Debug Steps:

1. Verify URR in PFCP Session Establishment:

```
Create URR
├── URR-ID: 1
├── Measurement-Method: VOLUME
├── Volume-Threshold: 8000000 (8 MB)
└── Reporting-Triggers: VOLUME_THRESHOLD
```

2. Check PGW-U logs for usage reports

3. Verify `quota_threshold_percentage` in config

4. Incorrect Rating-Group

Symptoms:

- OCS rejects with "Unknown Rating-Group"
- Sessions fail

Cause:

- Rating-Group in CCR-I doesn't match OCS configuration
- PCRF provisioned invalid Rating-Group

Resolution:

1. Verify Rating-Group in PCC Rule from PCRF
2. Check OCS configuration for valid Rating-Groups
3. Ensure mapping between PCC Rules and OCS tariffs

Monitoring

Key Metrics

```
# Gy message rates
rate(gy_inbound_messages_total{message_type="cca"}[5m])
rate(gy_outbound_messages_total{message_type="ccr"}[5m])

# Gy error rates
rate(gy_inbound_errors_total[5m])

# Quota exhaustion events
rate(gy_quota_exhausted_total[5m])

# OCS timeout rate
rate(gy_timeout_total[5m])

# Gy message handling duration
histogram_quantile(0.95,
rate(gy_inbound_handling_duration_bucket[5m]))
```

Alerts

```yaml
# Alert on high Gy error rate
- alert: GyErrorRateHigh
  expr: rate(gy_inbound_errors_total[5m]) > 0.1
  for: 5m
  annotations:
    summary: "High Gy error rate detected"

# Alert on OCS timeout
- alert: OcsTimeout
  expr: rate(gy_timeout_total[5m]) > 0.05
  for: 2m
  annotations:
    summary: "OCS timeouts occurring"

# Alert on credit exhaustion spike
- alert: CreditExhaustionSpike
  expr: rate(gy_quota_exhausted_total[5m]) > 10
  for: 5m
  annotations:
    summary: "High rate of credit exhaustion"
```

Gy Credit Control (OAM API)

Online charging is inspected and exercised through the **OAM REST API**, served over HTTPS/TLS at `https://<host>:8443`. Interactive API documentation (Swagger UI) is available at `https://<host>:8443/api/docs`.

Online Charging Status

```
GET /api/charging                 # all online-charging sessions +
live credit state
GET /api/charging?search=<imsi>   # filter sessions by IMSI
GET /api/charging/<imsi>          # one session's credit-control
state by IMSI
```

Example:

```
curl -k https://localhost:8443/api/charging/310170123456789
```

`GET /api/charging` returns the online-charging state (granted vs. used quota, validity, the OCS Final-Unit-Action, and walled-garden / out-of-credit redirect state) for each active session. Add `?search=<imsi>` to narrow the list, or append the IMSI to `GET /api/charging/<imsi>` to fetch a single subscriber's credit-control state.

Gy Credit Control Simulator

```
POST /api/gy_simulator
```

Sends a synthetic Credit-Control-Request to the configured OCS and returns the parsed answer — command-level Result-Code, granted quota, validity time, and any Final-Unit-Indication — so online charging can be exercised **without live subscriber traffic**. This is the OAM API replacement for the old browser-based Gy simulator. It requires Gy to be enabled (`gy.enabled: true`); the request is sent from the PGW-C's own Gy origin host to the configured OCS realm.

Request body (`application/json`):

Field	Type	Description
`imsi`	string	**Required.** Subscriber IMSI.
`msisdn`	string	Optional MSISDN (E.164). Populates `User-Name` as `<msisdn>@<apn>`.
`apn`	string	Optional APN.
`requested_units`	integer	Optional requested quota in octets (Requested-Service-Unit → CC-Total-Octets).
`rating_group`	integer	Optional Rating-Group / charging key (default `1`).
`service_id`	integer	Optional Service-Identifier.
`cc_request_type`	string	`initial` (default), `update`, or `terminate` (or the numeric `1`/`2`/`3`).

Example — request 1 MB of initial quota for a subscriber:

```
curl -k -X POST https://localhost:8443/api/gy_simulator \
  -H 'content-type: application/json' \
  -d
'{"imsi":"310170123456789","msisdn":"14155551234","apn":"internet","r
```

Example response (CCA-Initial granting 1 MB, valid 1 hour):

```json
{
  "session_id": "omni-pgw_c.epc...;734825199;app_gy_sim",
  "cc_request_type": 1,
  "cc_request_type_name": "initial",
  "result_code": "diameter_success",
  "accepted": true,
  "ocs_host": "ocs.epc.example.org",
  "ocs_realm": "epc.example.org",
  "granted": { "total_octets": 1000000, "input_octets": null, "output_octets": null, "time": null },
  "validity_time": 3600,
  "final_unit_action": null,
  "final_unit_redirect": null,
  "mscc_result_code": "diameter_success"
}
```

On failure the endpoint returns `422` (`imsi` missing) or `503` (Gy disabled, or the OCS did not answer — the `reason` field carries the underlying error, e.g. `:timeout`).

Operational Use Cases

- **Inspect a subscriber's quota** — `GET /api/charging/<imsi>` returns the session's current granted quota and credit status.
- **Test OCS integration without traffic** — `POST /api/gy_simulator` to confirm the OCS grants the expected quota / Final-Unit-Action for a given IMSI, rating group, or service.
- **Troubleshoot charging issues** — locate an affected subscriber with `GET /api/charging?search=<imsi>` and review its live credit-control state.

For historical trends (message rates, quota exhaustion, OCS timeouts) use the Prometheus metrics described in Monitoring above.

Related Documentation

Charging and Policy

- **Diameter Gx Interface** - PCRF policy control, PCC rules that trigger online charging
- **Data CDR Format** - Offline charging records for postpaid billing
- **Configuration Guide** - Complete online charging configuration parameters

Session Management

- **Session Management** - PDN session lifecycle, bearer management
- **PFCP Interface** - Usage reporting from PGW-U via URRs
- **S5/S8 Interface** - GTP-C bearer setup and teardown

Operations

- **Monitoring Guide** - Gy metrics, quota tracking, OCS timeout alerts
- **UE IP Allocation** - IP pool configuration for charged sessions

---

Back to Operations Guide

Gn/Gp Interface Documentation

GTP-C Communication with SGSN (2G/3G Networks)

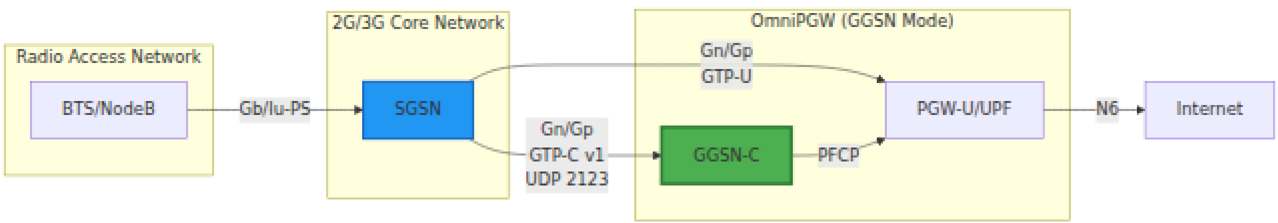
OmniPGW by Omnitouch Network Services

Overview

The **Gn/Gp interface** enables OmniPGW to function as a **GGSN** (Gateway GPRS Support Node) for 2G/3G networks using the **GTP-C v1** protocol. This interface handles PDP (Packet Data Protocol) context management between the GGSN and SGSNs.

- **Gn interface**: Connects to SGSNs within the same PLMN (home network)
- **Gp interface**: Connects to SGSNs in other PLMNs (roaming scenarios)

Both interfaces use identical protocols - the distinction is purely topological.



Key Differences from PGW (S5/S8)

Aspect	GGSN (Gn/Gp)	PGW (S5/S8)
Protocol Version	GTP-C v1	GTP-C v2
3GPP Specification	TS 29.060	TS 29.274
Session Concept	PDP Context	PDN Session + Bearers
Bearer Identifier	NSAPI (0-15)	EBI (5-15)
Policy Interface	None (static policy)	Gx (PCRF)
QoS Model	R99 QoS Profile	EPS Bearer QoS (QCI)
Port	2123 (shared)	2123

Protocol Details

GTP-C Version 1

- **Protocol**: GTP-C v1 (3GPP TS 29.060)
- **Transport**: UDP
- **Port**: 2123 (shared with S5/S8 GTP-C v2)
- **Interface Type**: Control Plane

TEID (Tunnel Endpoint Identifier)

Each PDP context has unique TEIDs for control and user plane:

- **GGSN Control TEID**: Allocated by OmniPGW for incoming control messages
- **GGSN Data TEID**: Allocated by UPF for incoming user data

- **SGSN Control TEID**: Provided by SGSN for outgoing control messages
- **SGSN Data TEID**: Provided by SGSN for outgoing user data

```
Message Flow:
  SGSN → GGSN: Destination TEID = GGSN's Control TEID
  GGSN → SGSN: Destination TEID = SGSN's Control TEID
```

CUPS Support (Control and User Plane Separation)

OmniPGW supports CUPS architecture for 2G/3G:

- **GSN Address for Signalling**: OmniPGW's control plane IP (from `gn.local_ipv4_address`)
- **GSN Address for User Traffic**: UPF's IP address (from PFCP Session Establishment Response)

This allows the GGSN-C and PGW-U/UPF to run on separate nodes with different IP addresses, which is standard for CUPS deployments.

NSAPI (Network Service Access Point Identifier)

The NSAPI identifies individual PDP contexts within a subscriber session:

- **Range**: 5-15 (0-4 reserved)
- **Usage**: Multiple PDP contexts per subscriber possible
- **Equivalent**: Similar to EBI in LTE/EPC

Configuration

Basic Configuration

```
# config/runtime.exs
config :pgw_c,
  # Gn/Gp interface is enabled automatically when gn config is
present
  gn: %{
    # Local IPv4 address for Gn interface
    local_ipv4_address: "10.0.0.20",

    # Optional: Local IPv6 address
    local_ipv6_address: nil,

    # Port for GTP-C v1 (shared with S5/S8)
    local_port: 2123
  },

  # DNS servers returned in PCO (shared with PGW)
  dns: %{
    primary_ipv4: {8, 8, 8, 8},
    secondary_ipv4: {8, 8, 4, 4}
  }
```

Gn Interface Parameters

Parameter	Type	Required	Default	Description
`local_ipv4_address`	String	Yes	-	IPv4 address to bind for Gn interface. This is the GGSN address advertised to SGSNs.
`local_ipv6_address`	String	No	`nil`	Optional IPv6 address. Most 2G/3G networks are IPv4-only.
`local_port`	Integer	No	2123	UDP port for GTP-C. Standard port per 3GPP TS 29.060. Shared with S5/S8.

DNS Parameters

Parameter	Type	Required	Default	Description
`primary_ipv4`	Tuple	No	`{8, 8, 8, 8}`	Primary DNS server IPv4 address returned in PCO.
`secondary_ipv4`	Tuple	No	`{8, 8, 4, 4}`	Secondary DNS server IPv4 address returned in PCO.
`primary_ipv6`	Tuple	No	`nil`	Primary DNS server IPv6 address.
`secondary_ipv6`	Tuple	No	`nil`	Secondary DNS server IPv6 address.

Network Requirements

Firewall Rules:

```
# Allow GTP-C from SGSN network
iptables -A INPUT -p udp --dport 2123 -s <sgsn_network>/24 -j
ACCEPT

# Allow outbound GTP-C to SGSN
iptables -A OUTPUT -p udp --dport 2123 -d <sgsn_network>/24 -j
ACCEPT
```

Message Types

PDP Context Management

Create PDP Context Request

Direction: SGSN → GGSN

Purpose: Establish a new PDP context for data connectivity

Key IEs (Information Elements):

IE Name	Type	Description
IMSI	Identity	International Mobile Subscriber Identity
NSAPI	Integer	Network Service Access Point Identifier (5-15)
TEID Data I	Integer	SGSN's user plane tunnel endpoint
TEID Control	Integer	SGSN's control plane tunnel endpoint
End User Address	Address	Requested PDN type and optional IP
Access Point Name	String	APN for the connection
GSN Address (Signalling)	IP	SGSN signalling plane address
GSN Address (User)	IP	SGSN user plane address
QoS Profile	Binary	Requested QoS parameters
MSISDN	Identity	Mobile phone number (optional)
Selection Mode	Enum	APN selection mode
Protocol Configuration Options	Options	Additional protocol options

Example:

```
Create PDP Context Request
├── IMSI: 310260123456789
├── NSAPI: 5
├── TEID Data I: 0x12345678
├── TEID Control: 0x87654321
├── End User Address: IPv4 (dynamic)
├── APN: internet
├── GSN Address (Signalling): 10.1.1.100
├── GSN Address (User): 10.1.1.100
├── QoS Profile: [R99 QoS parameters]
└── PCO: [DNS request, etc.]
```

Create PDP Context Response

Direction: GGSN → SGSN

Purpose: Acknowledge PDP context creation with allocated resources

Key IEs:

IE Name	Type	Description
Cause	Result	Success or error code
Reordering Required	Boolean	Whether reordering is needed
Recovery	Counter	GGSN restart counter
TEID Data I	Integer	GGSN's user plane tunnel endpoint
TEID Control	Integer	GGSN's control plane tunnel endpoint
NSAPI	Integer	Confirmed NSAPI
Charging ID	Binary	Charging correlation ID
End User Address	Address	Allocated UE IP address
GSN Address (Signalling)	IP	GGSN signalling plane address
GSN Address (User)	IP	GGSN user plane address
QoS Profile	Binary	Negotiated QoS parameters
Protocol Configuration Options	Options	DNS servers, etc.

Success Response:

```
Create PDP Context Response
├── Cause: Request Accepted (128)
├── TEID Data I: 0xAABBCCDD
├── TEID Control: 0xDDCCBBAA
├── NSAPI: 5
├── Charging ID: 0x11223344
├── End User Address
│   └── IPv4: 100.64.1.42
├── GSN Address (Signalling): 10.0.0.20
├── GSN Address (User): 10.0.0.20
├── QoS Profile: [Negotiated QoS]
└── PCO
    ├── Primary DNS: 8.8.8.8
    └── Secondary DNS: 8.8.4.4
```

Update PDP Context Request

Direction: SGSN → GGSN

Purpose: Modify an existing PDP context (e.g., SGSN relocation, QoS change)

Common scenarios:

- Inter-SGSN handover (routing area update)
- QoS renegotiation
- SGSN address change

Key IEs:

IE Name	Description
NSAPI	PDP context identifier
TEID Data I	New SGSN user plane TEID
GSN Address	New SGSN addresses
QoS Profile	Updated QoS (optional)

Update PDP Context Response

Direction: GGSN → SGSN

Purpose: Acknowledge PDP context update

Key IEs:

IE Name	Description
Cause	Success or error code
TEID Data I	GGSN user plane TEID
GSN Address	GGSN addresses
QoS Profile	Negotiated QoS

Delete PDP Context Request

Direction: SGSN → GGSN

Purpose: Terminate a PDP context

Key IEs:

IE Name	Description
NSAPI	PDP context to delete
Teardown Ind	Delete all contexts for this subscriber

Delete PDP Context Response

Direction: GGSN → SGSN

Purpose: Acknowledge PDP context deletion

Key IEs:

IE Name	Description
Cause	Success or error code

Path Management

Echo Request / Response

Direction: Bidirectional

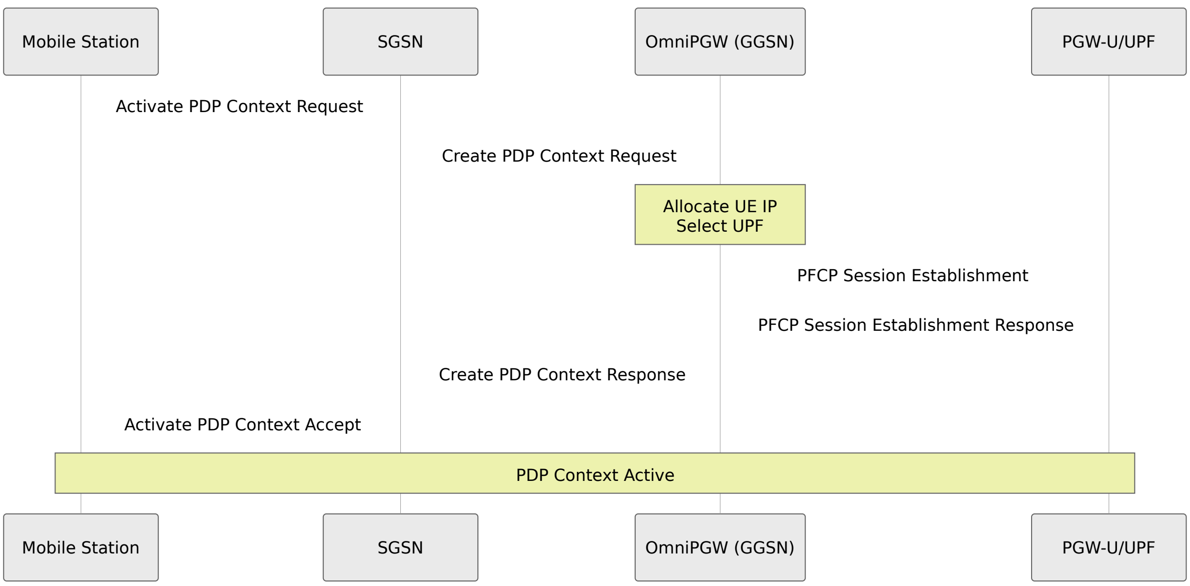
Purpose: Path keepalive and restart detection

Key IEs:

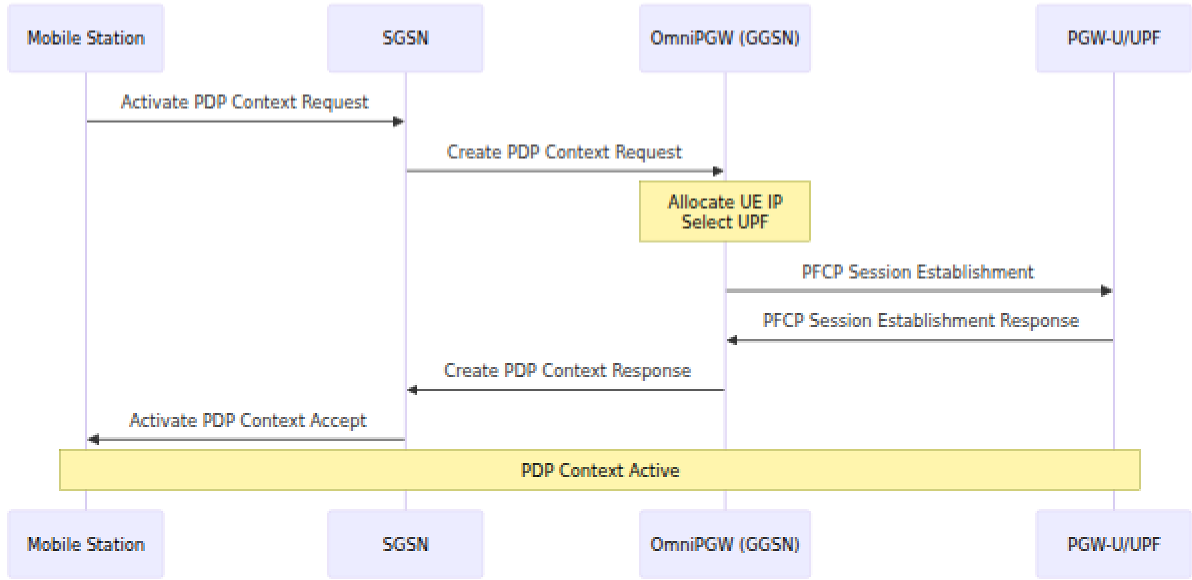
IE Name	Description
Recovery	Restart counter for detecting node restarts

Message Flows

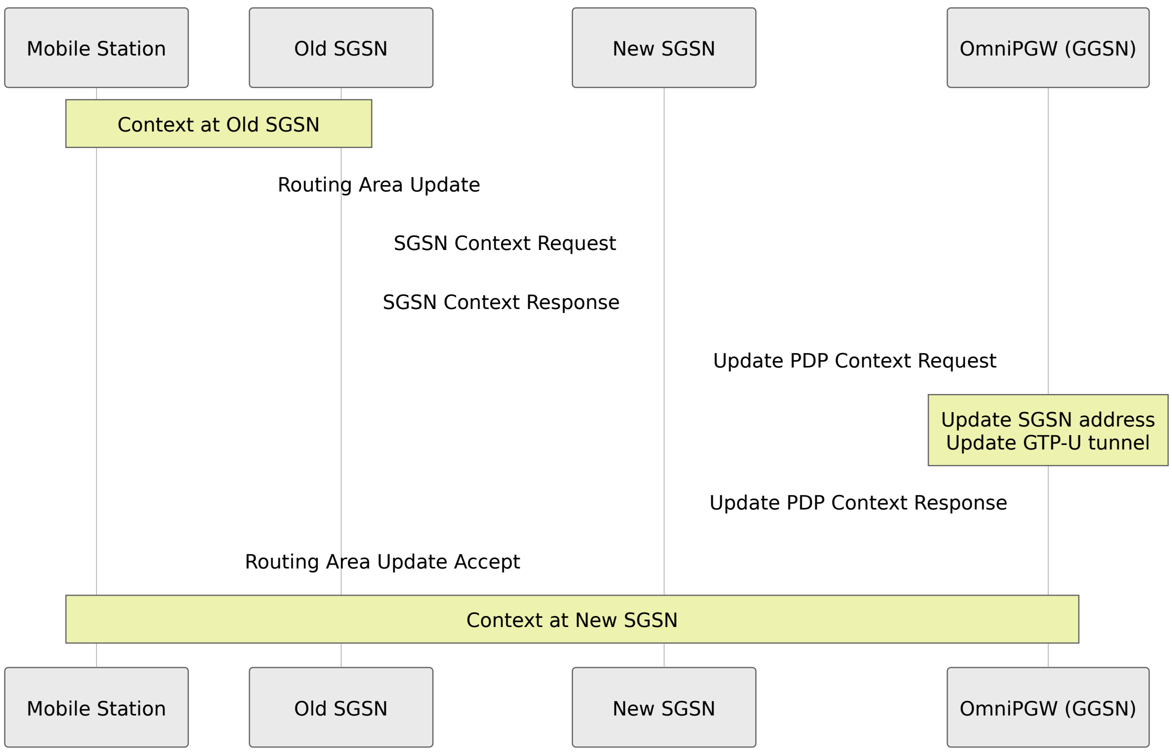
PDP Context Activation



PDP Context Deactivation



Inter-SGSN Handover



Shared Infrastructure

OmniPGW's GGSN functionality shares infrastructure with PGW mode:

UE IP Address Allocation

Uses the same AddressRegistry and subnet configuration as PGW. See UE IP Allocation for configuration details.

UPF/PFCP Session Management

Uses the same PFCP session establishment as PGW:

- PDR/FAR/QER rules created per PDP context
- UPF selection via the same upf_selection configuration
- See PFCP Interface for details

Online Charging (Gy)

Optional Diameter Gy integration available:

- Credit control for prepaid subscribers
- Same Gy configuration as PGW mode
- See Diameter Gy Interface for configuration

CDR Generation

Uses the same CDR generation as PGW. See Data CDR Format for details.

Cause Codes

Success

Code	Name	Description
128	Request Accepted	Successful operation

Rejection Causes

Code	Name	When Used
192	Non-existent	PDP context not found
193	Invalid message format	Malformed GTP message
194	IMSI not known	Subscriber not found
195	MS is GPRS Detached	MS not attached
196	MS is not GPRS Responding	MS unreachable
197	MS Refuses	MS rejected request
198	Version not supported	GTP version mismatch
199	No resources available	Resource exhaustion
200	Service not supported	Unsupported service
201	Mandatory IE incorrect	IE validation failed
202	Mandatory IE missing	Required IE absent
203	Optional IE incorrect	Optional IE invalid
204	System failure	Internal error
205	Roaming restriction	Roaming not allowed
206	P-TMSI Signature mismatch	Security failure
207	GPRS connection suspended	Connection suspended
208	Authentication failure	Auth failed

Code	Name	When Used
209	User authentication failed	User auth failed

Monitoring

Gn Interface Metrics

```
# Message counters
gn_inbound_messages_total{message_type="create_pdp_context_request"}
gn_inbound_messages_total{message_type="update_pdp_context_request"}
gn_inbound_messages_total{message_type="delete_pdp_context_request"}

gn_outbound_messages_total{message_type="create_pdp_context_response'
gn_outbound_messages_total{message_type="update_pdp_context_response'
gn_outbound_messages_total{message_type="delete_pdp_context_response'

# Transaction latency
gn_outbound_transaction_duration_bucket
```

Useful Queries

PDP Context Creation Rate:

```
rate(gn_inbound_messages_total{message_type="create_pdp_context_reque
[5m])
```

Success Rate:

```
sum(rate(gn_outbound_messages_total{message_type="create_pdp_context_
[5m]))
  /
sum(rate(gn_inbound_messages_total{message_type="create_pdp_context_r
[5m]))
```

Active PDP Contexts:

```
gn_inbound_messages_total{message_type="create_pdp_context_request"}
  -
gn_inbound_messages_total{message_type="delete_pdp_context_request"}
```

Troubleshooting

Issue: No Response to Create PDP Context

Symptoms:

- SGSN sends Create PDP Context Request
- No response received
- Timeout at SGSN

Possible causes:

1. Network connectivity issue
2. OmniPGW not listening on configured IP
3. Firewall blocking UDP 2123
4. UPF/PFCP establishment failure

Debug:

```
# Check OmniPGW is listening
netstat -ulnp | grep 2123

# Check for incoming packets
tcpdump -i any -n port 2123

# Verify configuration
grep "local_ipv4_address" config/runtime.exs

# Check PFCP connectivity
curl http://pgw:9090/metrics | grep pfcp
```

Issue: PDP Context Creation Rejected

Symptoms:

- Create PDP Context Response with error cause
- PDP context not established

Common causes:

```
Cause 199 (No resources available):
  → IP pool exhausted
  → Check: curl http://pgw:9090/metrics | grep
address_registry_count
  → Expand IP pool in ue.subnet_map

Cause 202 (Mandatory IE missing):
  → SGSN not sending required IEs
  → Check SGSN configuration
  → Review message in tcpdump

Cause 204 (System failure):
  → UPF/PFCP failure
  → Check PFCP association status
  → Review OmniPGW logs
```

Issue: Update PDP Context Fails

Symptoms:

- Update PDP Context Response with error cause
- Handover fails

Possible causes:

- PDP context not found (different TEID)
- NSAPI mismatch
- Session already deleted

Debug:

```
# Check active sessions
curl http://pgw:9090/metrics | grep session_count

# Review logs for context lookup
journalctl -u omnipgw | grep "PDP Context"
```

Best Practices

Network Design

1. **Shared Port with S5/S8**

   - Gn and S5/S8 share UDP port 2123
   - OmniPGW auto-detects GTP version from message
   - Single IP can serve both 2G/3G and 4G

2. **SGSN Connectivity**

   - Ensure all SGSNs can reach GGSN IP
   - Consider dedicated VLAN for Gn traffic
   - Plan for roaming (Gp) if applicable

3. **IP Address Planning**

    - Same IP pools serve both PGW and GGSN modes
    - Consider separate APN-based pools for 2G/3G

Performance

1. **Session Capacity**

    - Monitor `session_count` metric
    - Plan for expected 2G/3G subscriber count
    - Consider that 2G/3G typically has fewer concurrent sessions

2. **UDP Buffer Sizes**

    - Same recommendations as S5/S8
    - Typical: 4-8 MB per socket

Migration Considerations

When migrating from standalone GGSN to OmniPGW:

1. **APN Configuration**

    - Ensure APNs match existing GGSN configuration
    - Verify DNS server settings in PCO

2. **IP Pool Migration**

    - Plan IP pool cutover carefully
    - Consider gradual migration per APN

3. **QoS Mapping**

    - R99 QoS profiles accepted as-is
    - No QCI translation needed for 2G/3G

Related Documentation

Core Functions

- **Configuration Guide** - Complete configuration reference
- **Session Management** - Session lifecycle details
- **UE IP Allocation** - IP address management
- **PCO Configuration** - Protocol Configuration Options

Related Interfaces

- **S5/S8 Interface** - 4G/LTE equivalent interface
- **PFCP Interface** - User plane coordination
- **Diameter Gy Interface** - Online charging

Operations

- **Monitoring Guide** - Metrics and alerting
- **Troubleshooting** - Common issues
- **Data CDR Format** - CDR generation

References

- 3GPP TS 29.060 - GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface
- 3GPP TS 23.060 - General Packet Radio Service (GPRS); Service description
- 3GPP TS 24.008 - Mobile radio interface Layer 3 specification (QoS profiles)

Back to Documentation Index

IPv6 / Dual-Stack (IPv4v6) PDN Support

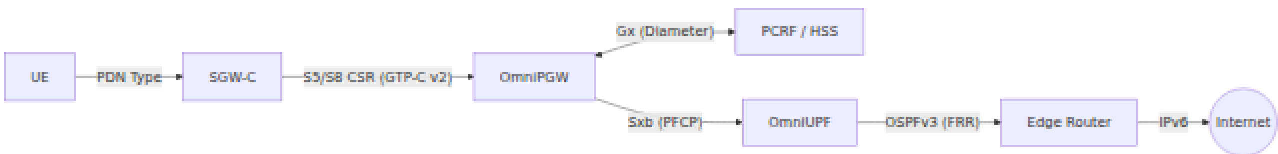
End-to-end guide for operating IPv6-capable PDN connections across the OmniPGW (control plane) and OmniUPF (user plane). This page covers how the PDN type is negotiated and signalled, the configuration required on both nodes, and how UE IPv6 reachability is achieved with dynamic routing.

For the detailed UE address-pool model (APN matching, allocation, collision handling), see UE IP Pool Allocation. For the PFCP and Gx interfaces in general, see PFCP/Sxb and Diameter Gx.

Table of Contents

Overview

A PDN connection can be **IPv4**, **IPv6**, or **IPv4v6** (dual-stack). The PDN type is requested by the UE/SGW in the GTPv2 Create Session Request and confirmed by the PGW-C in the response. Delivering an IPv6 PDN end-to-end requires coordinated behaviour across three interfaces and both nodes:



Node / Interface	Responsibility for IPv6
OmniPGW — S5/S8	Negotiate PDN type; allocate UE address(es); return the PDN Address Allocation (PAA)
OmniPGW — Gx	Convey the UE address to the PCRF: `Framed-IP-Address` (IPv4) for IPv4/IPv4v6 PDNs, `Framed-IPv6-Prefix` for IPv6-only PDNs
OmniPGW — Sxb	Signal the UE IPv6 address and PDN type to the UPF over PFCP
OmniUPF — data plane	Program the IPv6 downlink lookup; forward/encapsulate IPv6 user traffic
OmniUPF — FRR	Advertise each UE IPv6 `/128` host route to the edge router via OSPFv3

References: 3GPP TS 29.274 (GTPv2-C), 3GPP TS 29.212 (Gx), 3GPP TS 29.244 (PFCP).

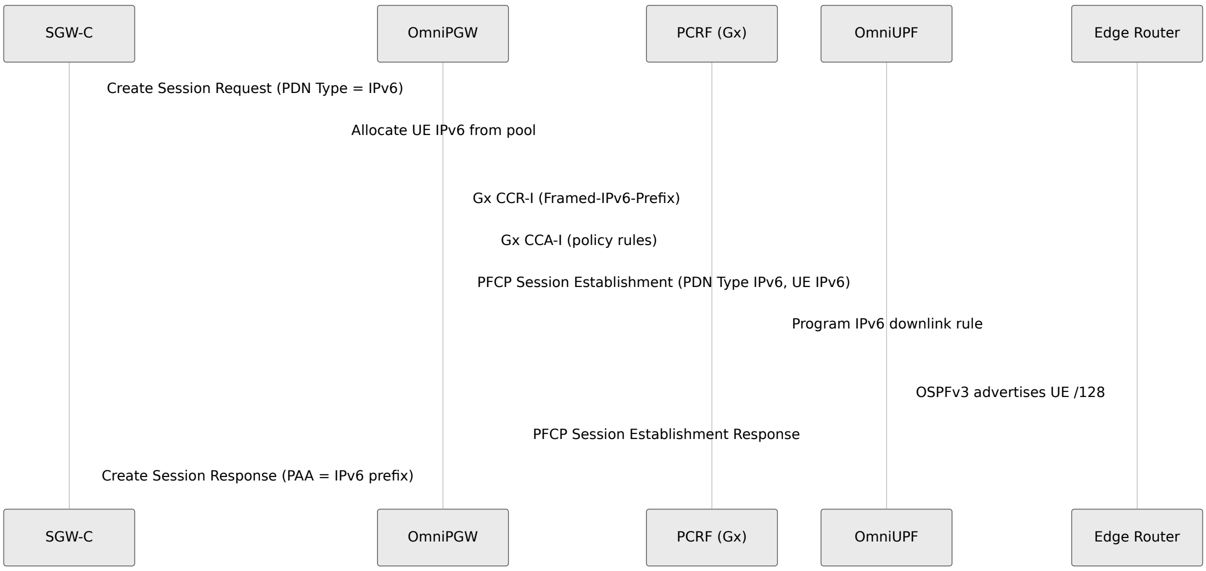
PDN Types

Value	PDN Type	UE receives	PAA contents (TS 29.274 §8.14)
1	IPv4	IPv4 address	4-byte IPv4 address
2	IPv6	IPv6 prefix (/64)	prefix-length octet + 16-byte address
3	IPv4v6	IPv4 address + IPv6 prefix	prefix-length octet + 16-byte IPv6 + 4-byte IPv4

The PGW-C derives the effective PDN type from the **PDN Type IE**, falling back to the type of the **PAA IE** if the PDN Type IE is absent. The same negotiated type drives the PAA returned to the SGW, the PFCP PDN Type IE sent to the UPF, and which address family/families are allocated.

> **IPv4v6 with no IPv6 pool.** If an IPv4v6 PDN is requested but the matched APN has no IPv6 range configured, the PGW-C does **not** crash; it either degrades the PDN to IPv4-only or rejects it, depending on whether the APN is an emergency (SOS) APN. See IPv4v6 request when the APN has no IPv6 pool.

End-to-End Flow



A retransmitted Create Session Request (same sequence number) is handled idempotently and produces the same response.

PGW-C Configuration

UE IPv6 Address Pool

UE addresses are drawn from per-APN pools under the `ue.subnet_map` key in `runtime.exs`. Add an IPv6 CIDR alongside the IPv4 CIDR for any APN that should offer IPv6 or dual-stack. The pool is parsed by address family automatically.

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # APN name => list of CIDR strings (IPv4 and/or IPv6)
      default: [
        "100.64.16.0/24",
        "2001:db8:6f85:70::/64"
      ]
    }
  }
```

Parameter	Type	Required	Default	Description
`ue.subnet_map`	Map	Yes	-	Maps an APN name (or `:default`) to a list of CIDR strings. `:default` is the fallback when no APN-specific entry matches.
(list entries)	String (CIDR)	Yes	-	An IPv4 (e.g. `100.64.16.0/24`) or IPv6 (e.g. `2001:db8:6f85:70::/64`) prefix. Address family is detected from the string. IPv6 UE addresses are drawn from within the configured prefix and a `/64` is signalled to the UE.

> For APN regex matching, multiple subnets per APN, static addresses, and the allocation algorithm, see UE IP Pool Allocation.

An APN with **both** an IPv4 and an IPv6 CIDR supports IPv4, IPv6, and IPv4v6 PDNs; the PGW-C allocates from whichever family/families the negotiated PDN type requires (for IPv4v6, one address from each).

Gx (Framed-IPv6-Prefix)

No additional configuration is required for Gx. The PGW-C sets the Gx framed address AVP from the UE address by family:

- **IPv6-only PDN** — the UE IPv6 prefix is conveyed using the **`Framed-IPv6-Prefix`** AVP (RFC 3162 §2.3), not `Framed-IP-Address` (which carries IPv4 only).
- **IPv4 and IPv4v6 PDNs** — the UE IPv4 address is conveyed in `Framed-IP-Address`. For IPv4v6 the IPv6 prefix is **not** carried on Gx (the CCR's framed-address AVPs are derived from the IPv4 address); the IPv6 prefix is still signalled to the UPF over PFCP and to the UE in the PAA.

For an IPv6-only subscriber the PCRF must be provisioned and able to process `Framed-IPv6-Prefix`; otherwise the Credit-Control-Request is dropped and the session fails — see Troubleshooting.

User Plane (OmniUPF)

For an IPv6 PDN to carry traffic, the OmniUPF must be IPv6-enabled, program the IPv6 downlink lookup from the PFCP session the PGW-C sends, and advertise each UE IPv6 `/128` host route to the edge router (via OSPFv3) so downlink traffic is attracted to the UPF.

Those are user-plane concerns configured on the OmniUPF, not the PGW-C. See the **OmniUPF IPv6 / Dual-Stack guide** for kernel settings, the UPF UE pool, and OSPFv3 route advertisement. From the PGW-C's perspective, the only requirement is that the selected UPF supports IPv6 — the PGW-C signals the UE IPv6 address and PDN type over PFCP (see PFCP/Sxb).

Reference Tables

PDN Address Allocation (PAA) — TS 29.274 §8.14

PDN Type	Address field layout
IPv4	4 octets: IPv4 address
IPv6	1 octet prefix length + 16 octets IPv6 prefix/address
IPv4v6	1 octet prefix length + 16 octets IPv6 + 4 octets IPv4

Gx Framed Address AVPs

AVP	Code	Family	Format	Reference
`Framed-IP-Address`	8	IPv4	4-byte address	RFC 4005
`Framed-IPv6-Prefix`	97	IPv6	reserved octet + prefix length + prefix	RFC 3162 §2.3

PFCP PDN Type IE — TS 29.244

Value	Meaning
1	IPv4
2	IPv6
3	IPv4v6

Troubleshooting

Gx CCR times out for IPv6 sessions

Symptoms: IPv4 sessions for a subscriber succeed, but IPv6/IPv4v6 sessions return cause `remote_peer_not_responding`; the Gx Credit-Control-Request times out.

Possible causes:

- The PCRF cannot process the `Framed-IPv6-Prefix` AVP and silently discards the request.
- The subscriber is not provisioned for IPv6 on the PCRF/HSS.

Resolution:

1. Confirm the subscriber is provisioned and enabled on the PCRF/HSS.
2. Verify the PCRF accepts `Framed-IPv6-Prefix` for Gx CCR-I.
3. Compare against an IPv4 session for the same subscriber to isolate IPv6-specific behaviour.

IPv6 session is accepted but downlink traffic does not reach the UE

Symptoms: The Create Session Response carries an IPv6 PAA, uplink works, but traffic from the network to the UE is dropped.

This is a user-plane (OmniUPF) concern — the PGW-C has already allocated the address and signalled it over PFCP. The usual cause is that the UE `/128` host route is not reaching the edge router (OSPFv3 not running/adjacent on the UPF, or convergence delay). See the OmniUPF IPv6 / Dual-Stack guide for route-advertisement configuration and troubleshooting.

No IPv6 address is allocated

Symptoms: An IPv6 PDN request is rejected, or no IPv6 PAA is returned.

Possible causes:

- The matched APN (or `:default`) has no IPv6 CIDR in `ue.subnet_map`.
- The configured IPv6 pool is exhausted.

Resolution:

1. Add an IPv6 CIDR to the APN's entry in `ue.subnet_map` (see PGW-C Configuration).
2. Confirm the pool has free addresses.

IPv4v6 request when the APN has no IPv6 pool

Symptoms: A UE requests an **IPv4v6** PDN against an APN that has only an IPv4 CIDR configured in `ue.subnet_map` (no IPv6 range). Depending on the APN, the session either comes up **IPv4-only** or is **rejected**.

Behaviour: The IPv6 allocator (`AddressRegistry.register_new_ipv6/1`) returns `{:error, :no_ipv6_pool}` when the matched APN has an empty IPv6 list, rather than crashing the session process. The session procedure then chooses by APN type (per TS 29.274 / TS 23.401 single-address-bearer / PDN-type-downgrade handling):

APN type	Result	What the UE / PCRF / UPF see
Emergency (SOS) APN	**Degrade to IPv4-only** — assign the IPv4 address, clear the IPv6 address, set the negotiated PDN type to `IPv4`.	PAA, PFCP PDN Type and PFCP UE IP are IPv4-only; Gx carries `Framed-IP-Address`. The **P-CSCF is still returned in the PCO** so the emergency IMS registration can proceed. A `warning` is logged.
Regular APN	**Reject** — the session is not silently downgraded; `{:error, :no_ipv6_pool}` is surfaced so the misconfiguration (or genuinely unsupported request) is visible. An `error` is logged.	Create Session fails with a non-success cause.

An emergency APN is identified by its name: the APN contains `"sos"` (e.g. `sos`, `sos.mnc410.mcc310.gprs`). IPv6-only requests against an APN with no IPv6 pool are likewise rejected.

> In practice the MME normally downgrades a regular subscriber's IPv4v6 request to its subscribed IPv4 before the request ever reaches the PGW-C, so the reject path is a backstop for an IPv4v6 request that genuinely arrives without an IPv6 pool. The SOS degrade path keeps emergency (e.g. 911/112) calls connecting on an IPv4-only deployment.

Resolution:

1. For a regular APN that should offer dual-stack, add an IPv6 CIDR to the APN's entry in `ue.subnet_map` (see PGW-C Configuration).
2. For an emergency (SOS) APN on an IPv4-only deployment, no action is required — the IPv4-only degrade is intentional. Add an IPv6 CIDR only if

you intend SOS PDNs to be dual-stack.

OmniPGW Monitoring & Metrics Guide

Prometheus Integration and Operational Monitoring

by Omnitouch Network Services

Table of Contents

Overview

OmniPGW provides two complementary monitoring approaches:

1. Real-Time OAM REST API (covered briefly here, detailed in respective interface docs)

- Live session viewer (`GET /api/sessions`)
- PFCP peer status (`GET /api/upf`)
- Diameter peer connectivity (`GET /api/diameter`)
- Individual session inspection (`GET /api/sessions/<imsi>`)

2. Prometheus Metrics (main focus of this document)

- Historical trends and analysis
- Alerting and notifications
- Performance metrics
- Capacity planning

This document focuses on **Prometheus metrics**. For the OAM REST API (served over HTTPS on port `8443`, under the `/api` prefix), see:

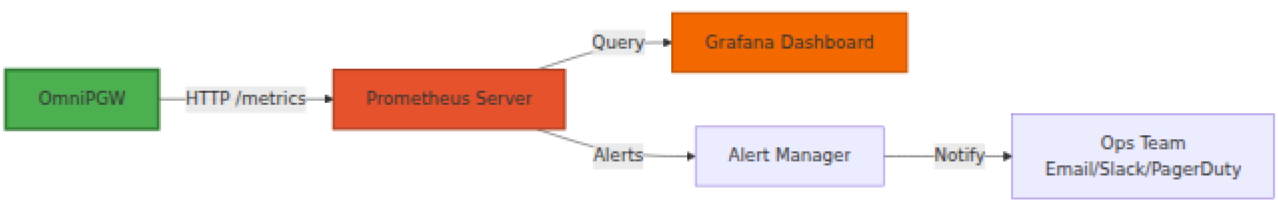
- Session Management - Sessions API
- PFCP Interface - Monitoring API
- Diameter Gx - Peer Monitoring API

Prometheus Metrics Overview

OmniPGW exposes **Prometheus-compatible metrics** for comprehensive monitoring of system health, performance, and capacity. This enables operations teams to:

- **Monitor System Health** - Track active sessions, allocations, and errors
- **Capacity Planning** - Understand resource utilization trends
- **Performance Analysis** - Measure message handling latency
- **Alerting** - Proactive notification of issues
- **Debugging** - Identify root causes of problems

Monitoring Architecture



Metrics Endpoint

Configuration

Enable metrics in `config/runtime.exs`:

```
config :pgw_c,
  metrics: %{
    enabled: true,
    ip_address: "127.0.0.42",       # Default bind address (use
"0.0.0.0" to expose on all interfaces)
    port: 9090,                      # Default HTTP port
    registry_poll_period_ms: 10_000  # Poll interval for polled
gauges (default 10s)
  }
```

> **Note:** The shipped defaults bind the metrics endpoint to `127.0.0.42:9090`. To scrape it from a remote Prometheus, set `ip_address` to a reachable address (e.g. `"0.0.0.0"`).

Accessing Metrics

HTTP Endpoint:

```
http://<omnipgw_ip>:<port>/metrics
```

Example:

```
curl http://10.0.0.20:9090/metrics
```

Output Format

Metrics are exposed in **Prometheus text format**:

```
# HELP teid_registry_count The number of TEID registered to
sessions
# TYPE teid_registry_count gauge
teid_registry_count 150

# HELP address_registry_count The number of addresses registered
to sessions
# TYPE address_registry_count gauge
address_registry_count 150

# HELP s5s8_inbound_messages_total The total number of messages
received from S5/S8 peers
# TYPE s5s8_inbound_messages_total counter
s5s8_inbound_messages_total{message_type="create_session_request"}
1523
s5s8_inbound_messages_total{message_type="delete_session_request"}
1487
```

Available Metrics

OmniPGW exposes the following metric categories:

Session Metrics

Active Session Counts:

Metric Name	Type	Description
`teid_registry_count`	Gauge	Active S5/S8 sessions (TEID count)
`seid_registry_count`	Gauge	Active PFCP sessions (SEID count)
`gx_session_id_registry_count`	Gauge	Active Gx sessions (Diameter Session-ID count)
`gy_session_id_registry_count`	Gauge	Active Gy sessions (Diameter Session-ID count)
`session_registry_count`	Gauge	Active sessions (IMSI, EBI pairs)
`address_registry_count`	Gauge	Allocated UE IP addresses
`charging_id_registry_count`	Gauge	Active charging IDs (see Data CDR Format for CDR billing records)
`sxb_sequence_number_registry_count`	Gauge	Pending PFCP responses (awaiting response)
`s5s8_sequence_number_registry_count`	Gauge	Pending S5/S8 responses (awaiting response)
`sxb_peer_registry_count`	Gauge	Number of registered PFCP peer processes

Usage:

```
# Current active sessions
teid_registry_count

# Session creation rate (per second)
rate(teid_registry_count[5m])

# Peak sessions in last hour
max_over_time(teid_registry_count[1h])
```

Session & Bearer Lifecycle:

Counters incremented as PDN sessions and bearers are created, modified and torn down.

Metric Name	Type	Labels	Description
`pgw_session_create_total`	Counter	`result`, `cause`, `pdn_type`, `rat_type`	PDN session creations, by result (success/failure), GTP cause, PDN type and RAT type
`pgw_session_delete_total`	Counter	`pdn_type`	PDN session deletions, by PDN type
`pgw_session_modify_total`	Counter	-	Modify Bearer Requests successfully handled
`pgw_bearer_create_total`	Counter	`type`, `qci`	Bearers created, by type (default/dedicated) and QCI
`pgw_bearer_delete_total`	Counter	`type`	Bearers deleted, by type (default/dedicated)

Usage:

```
# Session setup success ratio (last 5m)
sum(rate(pgw_session_create_total{result="success"}[5m])) /
sum(rate(pgw_session_create_total[5m]))

# Failed session setups by cause
sum(rate(pgw_session_create_total{result="failure"}[5m])) by
(cause)
```

UE IP Address Pool

Pool utilisation gauges are polled periodically; allocation failures are counted as they occur. See UE IP Allocation.

Metric Name	Type	Labels	Descrip
`ue_pool_addresses_allocated`	Gauge	`pool`, `ip_version`	UE IP address currentl allocate from a p
`ue_pool_addresses_total`	Gauge	`pool`, `ip_version`	Total UE address availabl a pool
`ue_pool_addresses_utilization_ratio`	Gauge	`pool`, `ip_version`	Fraction (0.0–1.0 a pool currentl allocate
`ue_ip_allocation_failures_total`	Counter	`ip_version`, `reason`	UE IP allocatio failures pool exhaust static collision

Usage:

```
# Pools above 80% utilisation
ue_pool_addresses_utilization_ratio > 0.8

# Allocation failure rate by reason
sum(rate(ue_ip_allocation_failures_total[5m])) by (reason)
```

Message Counters

S5/S8 (GTP-C) Messages:

Metric Name	Type	Labels	Description
`s5s8_inbound_messages_total`	Counter	`message_type`	Total inbound S5/S8 messages
`s5s8_outbound_messages_total`	Counter	`message_type`	Total outbound S5/S8 messages
`s5s8_inbound_errors_total`	Counter	`message_type`	S5/S8 inbound processing errors
`s5s8_outbound_errors_total`	Counter	`message_type`	S5/S8 outbound processing errors

Message Types:

- `create_session_request`
- `create_session_response`

- `delete_session_request`
- `delete_session_response`
- `create_bearer_request`
- `delete_bearer_request`

Sxb (PFCP) Messages:

Metric Name	Type	Labels	Descript
`sxb_inbound_messages_total`	Counter	`message_type`	Total inbound PFCP message:
`sxb_outbound_messages_total`	Counter	`message_type`	Total outbounc PFCP message:
`sxb_inbound_errors_total`	Counter	`message_type`	PFCP inbound processin errors
`sxb_outbound_errors_total`	Counter	`message_type`	PFCP outbounc processin errors
`sxb_outbound_responses_total`	Counter	`message_type`, `cause_code_class`, `peer_ip`	PFCP response sent, categoriz by PFCP cause coi class and UPF peer

Message Types:

- `association_setup_request`
- `association_setup_response`
- `heartbeat_request`
- `heartbeat_response`
- `session_establishment_request`
- `session_establishment_response`
- `session_modification_request`
- `session_deletion_request`

Gx (Diameter) Messages:

Metric Name	Type	Labels	Descript
`gx_inbound_messages_total`	Counter	`message_type`	Total inbound Diameter message
`gx_outbound_messages_total`	Counter	`message_type`	Total outbounc Diameter message
`gx_inbound_errors_total`	Counter	`message_type`	Diameter inbound processin errors
`gx_outbound_errors_total`	Counter	`message_type`	Diameter outbounc processin errors
`gx_outbound_responses_total`	Counter	`message_type`, `result_code_class`, `diameter_host`	Diameter response sent, categoriz by result code clas and peer host

Message Types:

- `gx_CCA` (Credit-Control-Answer)
- `gx_CCR` (Credit-Control-Request)
- `gx_RAA` (Re-Auth-Answer)
- `gx_RAR` (Re-Auth-Request)

Result Code Classes (for `gx_outbound_responses_total`):

- `2xxx` - Success responses (e.g., 2001 DIAMETER_SUCCESS)
- `3xxx` - Protocol errors (e.g., 3001 DIAMETER_COMMAND_UNSUPPORTED)
- `4xxx` - Transient failures (e.g., 4001 DIAMETER_AUTHENTICATION_REJECTED)
- `5xxx` - Permanent failures (e.g., 5012 DIAMETER_UNABLE_TO_COMPLY)

Usage Examples:

```
# Monitor Gx response success rate
sum(rate(gx_outbound_responses_total{result_code_class="2xxx"}[5m]))
sum(rate(gx_outbound_responses_total[5m])) * 100

# Track failures by PCRF host
rate(gx_outbound_responses_total{result_code_class!="2xxx"}[5m]) by (

# Count total successful Re-Auth-Answer messages
gx_outbound_responses_total{message_type="gx_RAA",result_code_class='

# Alert on high failure rate to specific PCRF
rate(gx_outbound_responses_total{result_code_class=~"4xxx|5xxx",diame
[5m]) > 0.1
```

Gy (Diameter) Messages:

Gy online-charging (OCS) message counters. See Diameter Gy for the interface.

Metric Name	Type	Labels	Description
`gy_inbound_messages_total`	Counter	`message_type`	Total inbound Gy messages
`gy_outbound_messages_total`	Counter	`message_type`	Total outbound Gy messages
`gy_inbound_errors_total`	Counter	`message_type`	Gy inbound processing errors
`gy_outbound_errors_total`	Counter	`message_type`	Gy outbound processing errors

Message Types:

- `gy_CCA` (Credit-Control-Answer)
- `gy_CCR` (Credit-Control-Request)

Gn/Gp (GTPv1-C) Messages:

Gn/Gp GGSN control-plane message counters (2G/3G SGSN interworking). See Gn Interface.

Metric Name	Type	Labels	Description
`gn_inbound_messages_total`	Counter	`message_type`	Total inbound Gn/Gp messages (from SGSN)
`gn_outbound_messages_total`	Counter	`message_type`	Total outbound Gn/Gp messages (to SGSN)
`gn_inbound_errors_total`	Counter	`message_type`	Gn/Gp inbound processing errors
`gn_outbound_errors_total`	Counter	`message_type`	Gn/Gp error responses sent to the SGSN

Error Handling:

Metric Name	Type	Labels	Description
`rescues_total`	Counter	`module`, `function`	Total rescue blocks hit (exception handling)

Latency Metrics

Inbound Message Processing Duration:

Metric Name	Type	Labels
`s5s8_inbound_handling_duration`	Histogram	`request_message_type`
`gn_inbound_handling_duration`	Histogram	`request_message_type`
`sxb_inbound_handling_duration`	Histogram	`request_message_type`
`gx_inbound_handling_duration`	Histogram	`request_message_type`

Metric Name	Type	Labels	I
			t (c e s r
`gy_inbound_handling_duration`	Histogram	`request_message_type`	C (r h t (c e s r

Outbound Transaction Duration:

Metric Name	Type	Labels
`s5s8_outbound_transaction_duration`	Histogram	`request_message_type`
`gn_outbound_transaction_duration`	Histogram	`request_message_type`
`sxb_outbound_transaction_duration`	Histogram	`request_message_type`
`gx_outbound_transaction_duration`	Histogram	`request_message_type`
`gy_outbound_transaction_duration`	Histogram	`request_message_type`

Buckets (seconds):

- Values: 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0
- (100μs, 500μs, 1ms, 5ms, 10ms, 50ms, 100ms, 500ms, 1s, 5s)

Usage:

```
# 95th percentile S5/S8 latency
histogram_quantile(0.95,
  rate(s5s8_inbound_handling_duration_bucket[5m])
)

# Average PFCP latency
rate(sxb_inbound_handling_duration_sum[5m]) /
rate(sxb_inbound_handling_duration_count[5m])
```

UPF Health Monitoring

UPF Peer Metrics:

Metric Name	Type	Labels	Description
`upf_peers_total`	Gauge	-	Total number of registered UPF peers
`upf_peers_healthy`	Gauge	-	Number of healthy UPF peers (associated + heartbeats OK)
`upf_peers_unhealthy`	Gauge	-	Number of unhealthy UPF peers
`upf_peers_associated`	Gauge	-	Number of UPF peers with active PFCP association
`upf_peers_unassociated`	Gauge	-	Number of UPF peers without PFCP association
`upf_peer_associated`	Gauge	`peer_ip`	Whether a specific UPF has an active PFCP association (1/0)
`upf_peer_healthy`	Gauge	`peer_ip`	Health status of specific UPF (1=healthy, 0=unhealthy)
`upf_peer_missed_heartbeats`	Gauge	`peer_ip`	Consecutive missed heartbeats for specific UPF

Metric Name	Type	Labels	Description
`upf_heartbeat_rtt`	Histogram	`peer_ip`	PFCP heartbeat request/response round-trip time (seconds)

Usage:

```
# Monitor UPF pool health
upf_peers_healthy / upf_peers_total

# Alert on unhealthy UPFs
upf_peers_unhealthy > 0

# Track specific UPF health
upf_peer_healthy{peer_ip="10.98.0.20"}

# Identify UPFs with heartbeat issues
upf_peer_missed_heartbeats > 2
```

Alerting Examples:

```yaml
# Alert when UPF goes down
- alert: UPF_Peer_Down
  expr: upf_peer_healthy == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "UPF {{ $labels.peer_ip }} is down"
    description: "UPF peer not responding to PFCP heartbeats"

# Alert when multiple UPFs are down
- alert: UPF_Pool_Degraded
  expr: (upf_peers_healthy / upf_peers_total) < 0.5
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: "UPF pool degraded"
    description: "Only {{ $value | humanizePercentage }} of UPFs
are healthy"

# Warning on missed heartbeats
- alert: UPF_Heartbeat_Issues
  expr: upf_peer_missed_heartbeats > 2
  for: 30s
  labels:
    severity: warning
  annotations:
    summary: "UPF {{ $labels.peer_ip }} heartbeat issues"
    description: "{{ $value }} consecutive missed heartbeats"
```

P-CSCF Health Monitoring

P-CSCF Server Metrics:

Metric Name	Type	Labels	Description
`pcscf_fqdns_total`	Gauge	-	Total P-CSCF FQDNs being monitored
`pcscf_fqdns_resolved`	Gauge	-	P-CSCF FQDNs successfully resolved via DNS
`pcscf_fqdns_failed`	Gauge	-	P-CSCF FQDNs that failed DNS resolution
`pcscf_servers_total`	Gauge	-	Total P-CSCF servers discovered
`pcscf_servers_healthy`	Gauge	`fqdn`	Healthy P-CSCF servers per FQDN
`pcscf_servers_unhealthy`	Gauge	`fqdn`	Unhealthy P-CSCF servers per FQDN

See: P-CSCF Monitoring Guide for detailed IMS health tracking.

Diameter Peer Connectivity

Polled periodically from the Diameter stack.

Metric Name	Type	Labels	Description
`diameter_peers_connected`	Gauge	`application`	Number of currently connected Diameter peers, by application (`gx`/`gy`/`all`)

Usage:

```
# Alert when no Gx (PCRF) peers are connected
diameter_peers_connected{application="gx"} == 0
```

License Metrics

License Status:

Metric Name	Type	Description
`license_status`	Gauge	Current license status (1 = valid, 0 = invalid)

Usage:

```
# Check if license is valid
license_status == 1

# Alert on invalid license
license_status == 0
```

Alerting Example:

```yaml
- alert: PGW_C_License_Invalid
  expr: license_status == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "PGW-C license invalid or expired"
    description: "License status is invalid - create session requests are being blocked"
```

Impact of Invalid License:

When the license is invalid or the license server is unreachable, **Create Session Requests will be rejected** with GTP-C cause code **"No resources**

available" (73). This is visible in packet captures as shown below:

Wireshark capture showing Create Session Response with "No resources available" cause when license is invalid

Notes:

- Product name registered with license server: `omnipgwc`
- License server URL is configured in `config/runtime.exs` under `:license_client`
- When license is invalid (`license_status == 0`), create session requests are blocked with GTP-C cause code 73 (No resources available)
- UI and monitoring remain accessible regardless of license status
- Diameter, GTP-C, and PFCP peers continue to maintain connections
- Existing sessions are not affected - only new session creation is blocked

System Metrics

Erlang VM Metrics:

Metric Name	Type	Description
`vm_memory_total`	Gauge	Total VM memory (bytes)
`vm_memory_processes`	Gauge	Memory used by processes
`vm_memory_system`	Gauge	Memory used by system
`vm_system_process_count`	Gauge	Total Erlang processes
`vm_system_port_count`	Gauge	Total open ports

Prometheus Configuration

Scrape Configuration

Add OmniPGW to Prometheus `prometheus.yml`:

```yaml
# prometheus.yml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'omnipgw'
    static_configs:
      - targets: ['10.0.0.20:9090']
        labels:
          instance: 'omnipgw-01'
          environment: 'production'
          site: 'datacenter-1'
```

Multiple OmniPGW Instances

```yaml
scrape_configs:
  - job_name: 'omnipgw'
    static_configs:
      - targets:
          - '10.0.0.20:9090'
          - '10.0.0.21:9090'
          - '10.0.0.22:9090'
        labels:
          environment: 'production'
```

Service Discovery

Kubernetes:

```yaml
scrape_configs:
  - job_name: 'omnipgw'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        action: keep
        regex: omnipgw
      - source_labels: [__meta_kubernetes_pod_ip]
        target_label: __address__
        replacement: '${1}:9090'
```

Verification

Test scrape:

```
# Check Prometheus targets
curl http://prometheus:9090/api/v1/targets

# Query a metric
curl 'http://prometheus:9090/api/v1/query?
query=teid_registry_count'
```

Grafana Dashboards

Dashboard Setup

1. Add Prometheus Data Source:

```
Configuration → Data Sources → Add data source → Prometheus
URL: http://prometheus:9090
```

2. Import Dashboard:

Create a new dashboard or import from JSON.

Key Panels

Panel 1: Active Sessions

```
# Query
teid_registry_count

# Panel Type: Gauge
# Thresholds:
#   Green: < 5000
#   Yellow: 5000-8000
#   Red: > 8000
```

Panel 2: Session Rate

```
# Query
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])

# Panel Type: Graph
# Unit: requests/sec
```

Panel 3: IP Pool Utilization

```
# Query (for /24 subnet with 254 IPs)
(address_registry_count / 254) * 100

# Panel Type: Gauge
# Unit: percent (0-100)
# Thresholds:
#   Green: < 70%
#   Yellow: 70-85%
#   Red: > 85%
```

Panel 4: Message Latency (95th Percentile)

```
# Query
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)

# Panel Type: Graph
# Unit: milliseconds
```

Panel 5: Error Rate

```
# Query
rate(s5s8_inbound_errors_total[5m])

# Panel Type: Graph
# Unit: errors/sec
# Alert Threshold: > 0.1
```

Panel 6: Gx Response Success Rate

```
# Query: Calculate percentage of successful Gx responses
sum(rate(gx_outbound_responses_total{result_code_class="2xxx"}
[5m])) /
sum(rate(gx_outbound_responses_total[5m])) * 100

# Panel Type: Gauge
# Unit: percent (0-100)
# Thresholds:
#   Green: > 95%
#   Yellow: 90-95%
#   Red: < 90%
```

Alternative - Breakdown by Result Code Class:

```
# Query: Show response counts by result code class
sum(rate(gx_outbound_responses_total[5m])) by (result_code_class)

# Panel Type: Pie Chart or Bar Chart
# Legend: {{ result_code_class }}
```

Alternative - Per-PCRF Response Status:

```
# Query: Show responses by PCRF host
sum(rate(gx_outbound_responses_total[5m])) by (diameter_host,
result_code_class)

# Panel Type: Stacked Bar Chart
# Legend: {{ diameter_host }} - {{ result_code_class }}
```

Panel 7: UPF Health Status

```
# Query: Overall pool health percentage
(upf_peers_healthy / upf_peers_total) * 100

# Panel Type: Gauge
# Unit: percent (0-100)
# Thresholds:
#   Green: 100%
#   Yellow: 50-99%
#   Red: < 50%
```

Alternative - Per-UPF Status:

```
# Query: Individual UPF health
upf_peer_healthy

# Panel Type: Stat
# Mappings:
#   1 = "UP" (Green)
#   0 = "DOWN" (Red)
```

Complete Dashboard Example

```
{
  "dashboard": {
    "title": "OmniPGW - Operations Dashboard",
    "panels": [
      {
        "title": "Active Sessions",
        "targets": [
          {
            "expr": "teid_registry_count",
            "legendFormat": "Active Sessions"
          }
        ],
        "type": "graph"
      },
      {
        "title": "Session Creation Rate",
        "targets": [
          {
            "expr":
"rate(s5s8_inbound_messages_total{message_type=\"create_session_reque
[5m])",
            "legendFormat": "Sessions/sec"
          }
        ],
        "type": "graph"
      },
      {
        "title": "IP Pool Utilization",
        "targets": [
          {
            "expr": "(address_registry_count / 254) * 100",
            "legendFormat": "Pool Usage %"
          }
        ],
        "type": "gauge"
      },
      {
        "title": "Message Latency (p95)",
        "targets": [
          {
            "expr": "histogram_quantile(0.95,
```

```
rate(s5s8_inbound_handling_duration_bucket[5m]))",
            "legendFormat": "S5/S8 p95"
          },
          {
            "expr": "histogram_quantile(0.95,
rate(sxb_inbound_handling_duration_bucket[5m]))",
            "legendFormat": "PFCP p95"
          }
        ],
        "type": "graph"
      }
    ]
  }
}
```

Alerting

Alert Rules

Create `omnipgw_alerts.yml`:

```yaml
groups:
  - name: omnipgw
    interval: 30s
    rules:
      # Session Count Alerts
      - alert: OmniPGW_HighSessionCount
        expr: teid_registry_count > 8000
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW high session count"
          description: "{{ $value }} active sessions (threshold:
8000)"

      - alert: OmniPGW_SessionCountCritical
        expr: teid_registry_count > 9500
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "OmniPGW session count critical"
          description: "{{ $value }} active sessions approaching
capacity"

      # IP Pool Alerts
      - alert: OmniPGW_IPPoolUtilizationHigh
        expr: (address_registry_count / 254) * 100 > 80
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW IP pool utilization high"
          description: "IP pool {{ $value }}% utilized"

      - alert: OmniPGW_IPPoolExhausted
        expr: address_registry_count >= 254
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "OmniPGW IP pool exhausted"
          description: "No IPs available for allocation"
```

```yaml
      # Error Rate Alerts
      - alert: OmniPGW_HighErrorRate
        expr: rate(s5s8_inbound_errors_total[5m]) > 0.1
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW high error rate"
          description: "{{ $value }} errors/sec on S5/S8
interface"

      - alert: OmniPGW_GxErrorRate
        expr: rate(gx_inbound_errors_total[5m]) > 0.05
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW Gx errors"
          description: "{{ $value }} Diameter errors/sec"

      # Gx Response Alerts
      - alert: OmniPGW_GxResponseFailureRate
        expr: |

sum(rate(gx_outbound_responses_total{result_code_class!="2xxx"}
[5m])) /
          sum(rate(gx_outbound_responses_total[5m])) > 0.1
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW high Gx response failure rate"
          description: "{{ $value | humanizePercentage }} of Gx
responses are failures (non-2xxx result codes)"

      - alert: OmniPGW_GxPCRFFailures
        expr:
rate(gx_outbound_responses_total{result_code_class=~"4xxx|5xxx"}
[5m]) by (diameter_host) > 0.05
        for: 3m
        labels:
          severity: warning
        annotations:
```

```yaml
          summary: "PCRF {{ $labels.diameter_host }} receiving
failure responses"
          description: "{{ $value }} failure responses/sec to PCRF
{{ $labels.diameter_host }}"

      # UPF Health Alerts
      - alert: OmniPGW_UPF_PeerDown
        expr: upf_peer_healthy == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "UPF peer {{ $labels.peer_ip }} down"
          description: "UPF not responding to PFCP heartbeats"

      - alert: OmniPGW_UPF_PoolDegraded
        expr: (upf_peers_healthy / upf_peers_total) < 0.5
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "UPF pool degraded"
          description: "{{ $value | humanizePercentage }} of UPFs
are healthy (< 50%)"

      - alert: OmniPGW_UPF_HeartbeatFailures
        expr: upf_peer_missed_heartbeats > 2
        for: 30s
        labels:
          severity: warning
        annotations:
          summary: "UPF {{ $labels.peer_ip }} heartbeat failures"
          description: "{{ $value }} consecutive missed
heartbeats"

      - alert: OmniPGW_UPF_AllDown
        expr: upf_peers_healthy == 0 and upf_peers_total > 0
        for: 30s
        labels:
          severity: critical
        annotations:
          summary: "All UPF peers down"
          description: "No healthy UPFs available for session
creation"
```

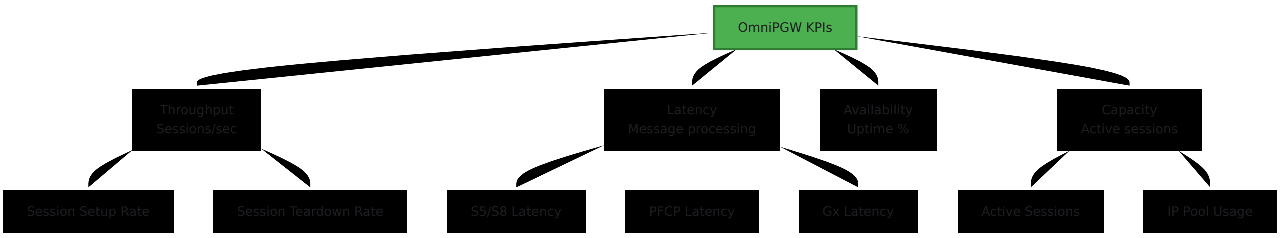
```yaml
      # Latency Alerts
      - alert: OmniPGW_HighLatency
        expr: |
          histogram_quantile(0.95,
            rate(s5s8_inbound_handling_duration_bucket[5m])
          ) > 100000
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW high message latency"
          description: "p95 latency {{ $value }}µs (> 100ms)"

      # System Alerts
      - alert: OmniPGW_HighMemoryUsage
        expr: vm_memory_total > 2000000000
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW high memory usage"
          description: "VM using {{ $value | humanize }}B memory"

      - alert: OmniPGW_HighProcessCount
        expr: vm_system_process_count > 100000
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW high process count"
          description: "{{ $value }} Erlang processes (potential
leak)"
```

AlertManager Configuration

```yaml
# alertmanager.yml
global:
  resolve_timeout: 5m

route:
  receiver: 'ops-team'
  group_by: ['alertname', 'instance']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 12h

  routes:
    - match:
        severity: critical
      receiver: 'pagerduty'

    - match:
        severity: warning
      receiver: 'slack'

receivers:
  - name: 'ops-team'
    email_configs:
      - to: 'ops@example.com'

  - name: 'slack'
    slack_configs:
      - api_url:
'https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK'
        channel: '#omnipgw-alerts'
        title: 'OmniPGW Alert: {{ .GroupLabels.alertname }}'
        text: '{{ range .Alerts }}{{ .Annotations.description }}{{
end }}'

  - name: 'pagerduty'
    pagerduty_configs:
      - service_key: 'YOUR_PAGERDUTY_KEY'
```

Performance Monitoring

Key Performance Indicators (KPIs)



Throughput Queries

Session Setup Rate:

```
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])
```

Session Teardown Rate:

```
rate(s5s8_inbound_messages_total{message_type="delete_session_request
[5m])
```

Net Session Growth:

```
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m]) -
rate(s5s8_inbound_messages_total{message_type="delete_session_request
[5m])
```

Latency Analysis

Message Processing Latency (Percentiles):

```
# p50 (Median)
histogram_quantile(0.50,
  rate(s5s8_inbound_handling_duration_bucket[5m])
)

# p95
histogram_quantile(0.95,
  rate(s5s8_inbound_handling_duration_bucket[5m])
)

# p99
histogram_quantile(0.99,
  rate(s5s8_inbound_handling_duration_bucket[5m])
)
```

Latency Breakdown by Message Type:

```
histogram_quantile(0.95,
  rate(s5s8_inbound_handling_duration_bucket[5m])
) by (request_message_type)
```

Capacity Trending

Session Growth Trend (24h):

```
teid_registry_count -
teid_registry_count offset 24h
```

Capacity Remaining:

```
# For max capacity of 10,000 sessions
10000 - teid_registry_count
```

Time to Capacity Exhaustion:

```
# Days until capacity exhausted (based on 1h growth rate)
(10000 - teid_registry_count) /
(rate(teid_registry_count[1h]) * 86400)
```

Troubleshooting Metrics

Identifying Issues

Issue: High Session Rejection Rate

Query:

```
rate(s5s8_inbound_errors_total[5m]) by (message_type)
```

Action:

- Check error logs
- Verify PCRF connectivity (Gx errors)
- Check IP pool exhaustion

Issue: Slow Session Setup

Query:

```
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)
```

Action:

- Check Gx latency (PCRF response time)
- Check PFCP latency (PGW-U response time)

- Review system resource usage

Issue: PCRF Policy Failures

Queries:

```
# Overall Gx response failure rate
sum(rate(gx_outbound_responses_total{result_code_class!="2xxx"}
[5m])) /
sum(rate(gx_outbound_responses_total[5m])) * 100

# Breakdown by PCRF host
sum(rate(gx_outbound_responses_total[5m])) by (diameter_host,
result_code_class)

# Specific result code classes
rate(gx_outbound_responses_total{result_code_class="5xxx"}[5m]) by
(diameter_host)
```

Action:

- Check PCRF connectivity and health
- Review subscriber profiles in PCRF (5xxx errors often indicate policy issues)
- Verify Diameter peer configuration
- Check PCRF logs for corresponding errors
- For 5012 (DIAMETER_UNABLE_TO_COMPLY), review Re-Auth-Request handling

Issue: Memory Leak Suspected

Queries:

```
# Total memory trend
rate(vm_memory_total[1h])

# Process memory trend
rate(vm_memory_processes[1h])

# Process count trend
rate(vm_system_process_count[1h])
```

Action:

- Check for stale sessions
- Review registry counts
- Restart if leak confirmed

Debugging Queries

Find Peak Session Time:

```
max_over_time(teid_registry_count[24h])
```

Compare Current vs. Historical:

```
teid_registry_count /
avg_over_time(teid_registry_count[7d])
```

Identify Anomalies:

```
abs(
  teid_registry_count -
  avg_over_time(teid_registry_count[1h])
) > 100
```

Best Practices

Metric Collection

1. **Scrape Interval:** 15-30 seconds (balance granularity vs. load)
2. **Retention:** 15+ days for historical analysis
3. **Labels:** Use consistent labeling (instance, environment, site)

Dashboard Design

1. **Overview Dashboard** - High-level KPIs for NOC
2. **Detailed Dashboards** - Per-interface deep dive
3. **Troubleshooting Dashboard** - Error metrics and logs

Alert Design

1. **Avoid Alert Fatigue** - Only alert on actionable issues
2. **Escalation** - Warning → Critical with escalating severity
3. **Context** - Include runbook links in alert descriptions

Related Documentation

Configuration and Setup

- **Configuration Guide** - Prometheus metrics configuration, OAM REST API setup
- **Troubleshooting Guide** - Using metrics for debugging

Interface Metrics

- **PFCP Interface** - PFCP session metrics, UPF health monitoring
- **Diameter Gx Interface** - Gx policy metrics, PCRF interaction tracking

- **Diameter Gy Interface** - Gy charging metrics, quota tracking, OCS timeouts
- **S5/S8 Interface** - GTP-C message metrics, SGW-C communication

Specialized Monitoring

- **P-CSCF Monitoring** - P-CSCF discovery metrics, IMS health
- **Session Management** - Active sessions, session lifecycle metrics
- **UE IP Allocation** - IP pool utilization metrics

Back to Operations Guide

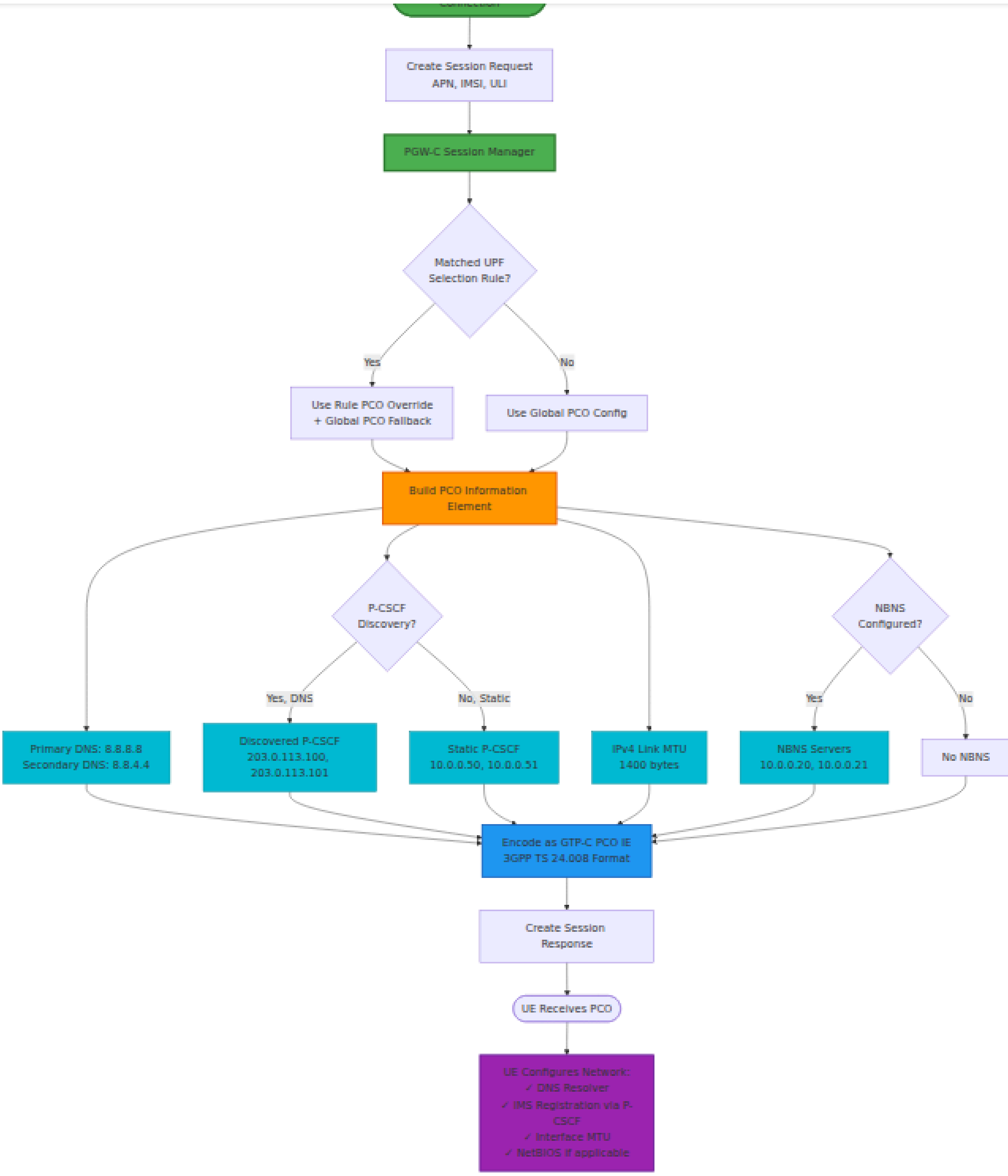
Protocol Configuration Options (PCO)

Network Parameters Delivered to UE

OmniPGW by Omnitouch Network Services

Overview

PCO (Protocol Configuration Options) are network parameters sent to the UE (mobile device) during PDN connection establishment. These parameters enable the UE to access network services like DNS, IMS, and configure network settings.



PCO containers OmniPGW emits (network → MS):

IE Name	Container ID	Description	Required
DNS Server IPv4 Address	0x000D	Primary DNS	Yes
DNS Server IPv4 Address	0x000D	Secondary DNS	Optional
DNS Server IPv6 Address	0x0003	Primary/Secondary DNS (IPv6)	Optional
P-CSCF IPv4 Address	0x000C	P-CSCF for IMS (one container per address)	Optional (IMS)
P-CSCF IPv6 Address	0x0001	P-CSCF for IMS (one container per address)	Optional (IMS)
IPv4 Link MTU	0x0010	Maximum transmission unit	Recommended

> **DNS/NBNS via IPCP.** When the UE asks for DNS (and NBNS/WINS) servers using an **IPCP** (Internet Protocol Control Protocol, container `0x8021`) Configure-Request rather than a bare DNS container, OmniPGW answers inside an IPCP **Configure-Nak** carrying the configured addresses as the IPCP options `ms_dns1`/`ms_dns2` and `ms_wins1`/`ms_wins2` (RFC 1877). NBNS/WINS is therefore delivered **only** through this IPCP path — there is no standalone NBNS PCO container (`0x0011`) emitted. A Configure-Nak (not an Ack) is used because the PGW supplies addresses that differ from the UE's `0.0.0.0` placeholders, and the reply echoes the request's IPCP Identifier (RFC 1661 §5).

> **IPv4 and IPv6 P-CSCF / DNS; no E-CSCF.** OmniPGW answers both IPv4 and IPv6 P-CSCF address requests, routing each discovered or configured address to the container matching its family: IPv4 addresses go in `0x000C`, IPv6 addresses go in `0x0001`. An IPv6 P-CSCF request (`p_cscf_ipv6_address_request`) is served from

> `p_cscf_ipv6_address_list` (or the IPv6 addresses returned by DNS/AAAA discovery), and an IPv6 DNS request (`dns_server_ipv6_address_request`) is served in container `0x0003` from `primary_dns_server_ipv6_address` / `secondary_dns_server_ipv6_address`. Address families are never mixed across containers. There is **no** E-CSCF container handling. All of the above applies equally to APCO (Additional PCO).

Configuration

Basic Configuration

```
# config/runtime.exs
config :pgw_c,
  pco: %{
    # DNS servers (required)
    primary_dns_server_address: "8.8.8.8",
    secondary_dns_server_address: "8.8.4.4",

    # IPv6 DNS servers (optional, container 0x0003)
    primary_dns_server_ipv6_address: nil,
    secondary_dns_server_ipv6_address: nil,

    # NBNS servers (optional, for Windows devices)
    primary_nbns_server_address: nil,
    secondary_nbns_server_address: nil,

    # P-CSCF addresses for IMS/VoLTE (optional)
    p_cscf_ipv4_address_list: [],   # container 0x000C
    p_cscf_ipv6_address_list: [],   # container 0x0001

    # P-CSCF Dynamic Discovery (optional; resolves A and AAAA
records)
    p_cscf_discovery_enabled: false,
    p_cscf_discovery_dns_server: nil,
    p_cscf_discovery_timeout_ms: 5000,

    # IPv4 MTU size (bytes)
    ipv4_link_mtu_size: 1400
  }
```

PCO Parameters

DNS Server Addresses

Primary and Secondary DNS:

```
pco: %{
  primary_dns_server_address: "8.8.8.8",
  secondary_dns_server_address: "8.8.4.4"
}
```

Common DNS Providers:

Provider	Primary	Secondary
Google	8.8.8.8	8.8.4.4
Cloudflare	1.1.1.1	1.0.0.1
Quad9	9.9.9.9	149.112.112.112
OpenDNS	208.67.222.222	208.67.220.220

Private DNS:

```
pco: %{
  primary_dns_server_address: "10.0.0.10",
  secondary_dns_server_address: "10.0.0.11"
}
```

P-CSCF Addresses (IMS)

For IMS/VoLTE Services:

```
pco: %{
  p_cscf_ipv4_address_list: [
    "10.0.0.50",  # Primary P-CSCF (IPv4, container 0x000C)
    "10.0.0.51"   # Secondary P-CSCF
  ],
  # IPv6 P-CSCF addresses (container 0x0001) for IPv6/IPv4v6 IMS
PDNs
  p_cscf_ipv6_address_list: [
    "2001:db8::50",
    "2001:db8::51"
  ]
}
```

Each address is emitted only in the container matching its family, and only when the UE requests that family — an IPv4 P-CSCF request receives the `0x000C` containers, an IPv6 P-CSCF request receives the `0x0001` containers. Dual-stack IMS PDNs typically request both.

P-CSCF (Proxy Call Session Control Function):

- Entry point for IMS signaling
- Required for VoLTE, VoWiFi, RCS
- UE uses SIP over this server

P-CSCF Dynamic Discovery

DNS-Based P-CSCF Discovery:

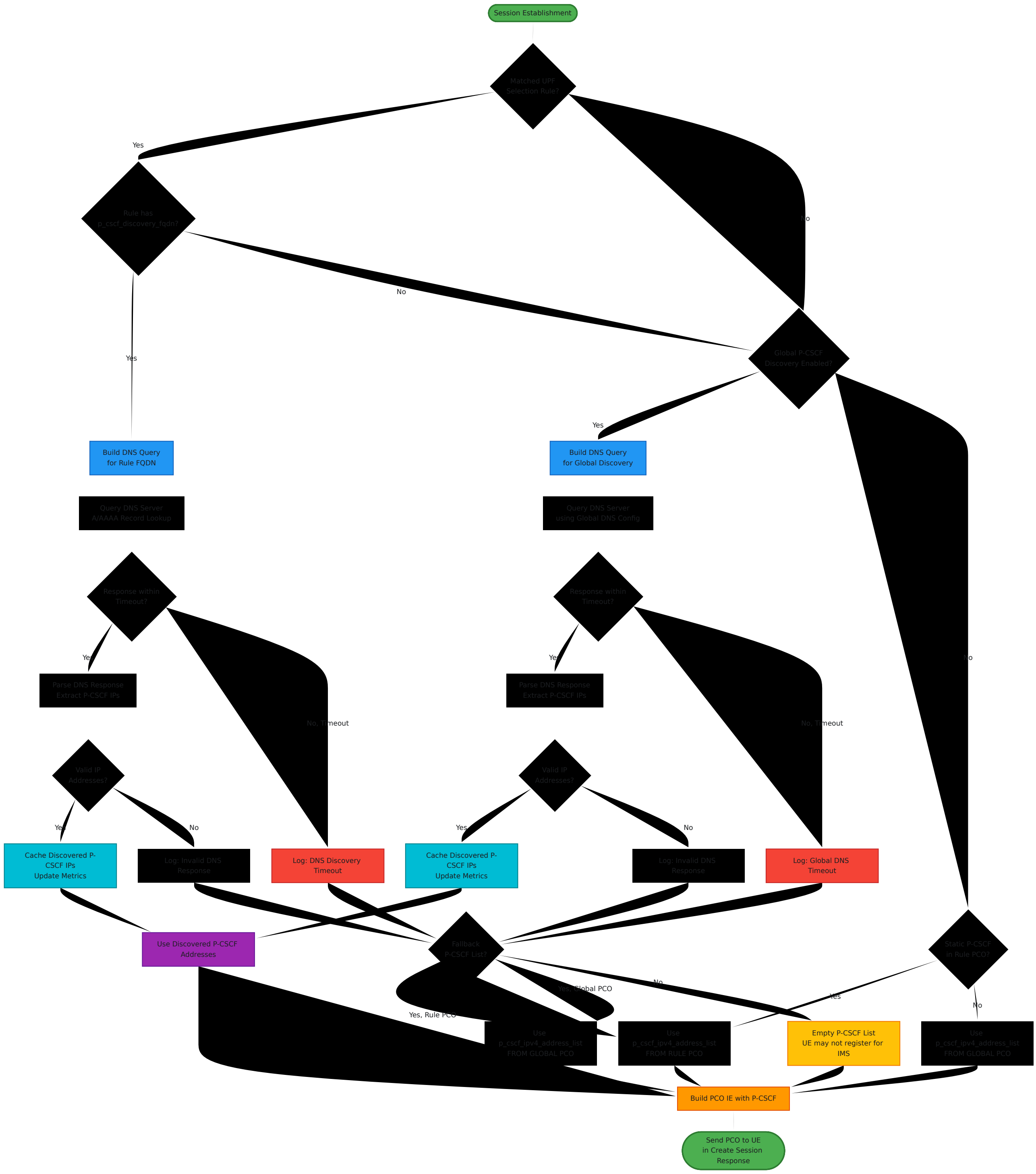
OmniPGW supports dynamic P-CSCF discovery via DNS queries as defined in 3GPP TS 23.003 and TS 24.229. When enabled, PGW-C can query DNS for P-CSCF addresses instead of using static configuration.

```
pco: %{
  # Enable dynamic P-CSCF discovery
  p_cscf_discovery_enabled: true,

  # DNS server for P-CSCF queries (as tuple)
  p_cscf_discovery_dns_server: {10, 179, 2, 177},

  # Timeout for DNS queries (milliseconds)
  p_cscf_discovery_timeout_ms: 5000,

  # Static P-CSCF list (used as fallback if DNS fails)
  p_cscf_ipv4_address_list: ["10.0.0.50"]
}
```

How It Works:

1. When `p_cscf_discovery_enabled: true`, PGW-C performs DNS queries for P-CSCF addresses
2. DNS query is sent to the configured `p_cscf_discovery_dns_server`
3. If DNS query succeeds, discovered P-CSCF addresses are sent to UE via PCO
4. If DNS query fails or times out, falls back to static `p_cscf_ipv4_address_list`
5. See P-CSCF Monitoring for detailed monitoring and metrics

P-CSCF Discovery Flow



Discovery Priority:

1. **Per-Rule FQDN Discovery** (Highest Priority) - `p_cscf_discovery_fqdn` in UPF selection rule
2. **Global DNS Discovery** - `p_cscf_discovery_enabled: true` in global PCO config
3. **Rule PCO Static List** - `p_cscf_ipv4_address_list` in rule PCO override

4. **Global PCO Static List** (Fallback) - `p_cscf_ipv4_address_list` in global PCO config

Monitoring:

All P-CSCF discovery attempts are logged and tracked with metrics:

- DNS query success/failure rates
- Discovery latency
- Fallback usage statistics
- Per-rule and global discovery metrics

See P-CSCF Monitoring for complete monitoring details.

Configuration Options:

Parameter	Type	Default	Description
`p_cscf_discovery_enabled`	Boolean	`false`	Enable dynamic DNS-based P-CSCF discovery
`p_cscf_discovery_dns_server`	Tuple (IP)	`nil`	DNS server IP address as 4-tuple (e.g., `{10, 179, 2, 177}`)
`p_cscf_discovery_timeout_ms`	Integer	`5000`	Timeout for DNS queries in milliseconds

Use Cases:

- **Dynamic IMS deployments** - P-CSCF addresses change based on DNS configuration
- **Geographic load balancing** - DNS returns closest P-CSCF servers
- **High availability** - DNS automatically returns available P-CSCF servers

- **Multi-tenant environments** - Different subscribers get different P-CSCF servers

Example: Production IMS with DNS Discovery

```
pco: %{
  primary_dns_server_address: "10.0.0.10",
  secondary_dns_server_address: "10.0.0.11",

  # Enable dynamic P-CSCF discovery
  p_cscf_discovery_enabled: true,
  p_cscf_discovery_dns_server: {10, 179, 2, 177},  # IMS DNS
server
  p_cscf_discovery_timeout_ms: 3000,

  # Fallback P-CSCF addresses (if DNS fails)
  p_cscf_ipv4_address_list: [
    "10.0.0.50",  # Primary fallback
    "10.0.0.51"   # Secondary fallback
  ],

  ipv4_link_mtu_size: 1400
}
```

Per-Rule P-CSCF Discovery:

P-CSCF discovery can also be configured per UPF selection rule. This allows different APNs to use different DNS servers for P-CSCF discovery:

```
# In upf_selection configuration
rules: [
  %{
    name: "IMS Traffic",
    priority: 20,
    match_field: :apn,
    match_regex: "^ims",
    upf_pool: [...],

    # Per-rule P-CSCF discovery
    p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org"
  }
]
```

See UPF Selection Configuration for details on per-rule P-CSCF discovery.

See also: P-CSCF Monitoring for monitoring P-CSCF discovery and health

NBNS Servers (NetBIOS)

For Windows Device Compatibility:

```
pco: %{
  primary_nbns_server_address: "10.0.0.20",
  secondary_nbns_server_address: "10.0.0.21"
}
```

When to Use:

- Enterprise networks with Windows devices
- Legacy application support
- NetBIOS name resolution required

Link MTU Size

Maximum Transmission Unit:

```
pco: %{
  ipv4_link_mtu_size: 1400  # bytes
}
```

Common MTU Values:

MTU	Use Case
1500	Standard Ethernet (no tunneling)
1400	GTP tunneling overhead accounted
1420	Reduced overhead
1280	IPv6 minimum MTU
1360	VPN/tunnel environments

Recommendation: Use **1400** for LTE to account for GTP-U overhead.

Configuration Examples

Internet APN

```
pco: %{
  primary_dns_server_address: "8.8.8.8",
  secondary_dns_server_address: "8.8.4.4",
  ipv4_link_mtu_size: 1400
}
```

IMS APN

```
pco: %{
  primary_dns_server_address: "10.0.0.10",
  secondary_dns_server_address: "10.0.0.11",
  p_cscf_ipv4_address_list: [
    "10.0.0.50",
    "10.0.0.51"
  ],
  ipv4_link_mtu_size: 1400
}
```

See: P-CSCF Monitoring for monitoring IMS registration success rates and P-CSCF health

Enterprise APN

```
pco: %{
  primary_dns_server_address: "10.100.0.10",
  secondary_dns_server_address: "10.100.0.11",
  primary_nbns_server_address: "10.100.0.20",
  secondary_nbns_server_address: "10.100.0.21",
  ipv4_link_mtu_size: 1400
}
```

PCO in GTP-C Messages

Create Session Response

OmniPGW includes PCO in the **Create Session Response** message:

```
Create Session Response
├── Cause: Request accepted
├── UE IP Address: 100.64.1.42
├── PCO (Protocol Configuration Options)
│   ├── DNS Server IPv4 Address: 8.8.8.8
│   ├── DNS Server IPv4 Address: 8.8.4.4
│   ├── P-CSCF IPv4 Address: 10.0.0.50
│   ├── P-CSCF IPv4 Address: 10.0.0.51
│   └── IPv4 Link MTU: 1400
```

UE Processing

The UE receives PCO and:

1. Configures DNS resolver with provided servers
2. Registers with P-CSCF for IMS services
3. Sets interface MTU to specified value

Troubleshooting

Issue: UE Cannot Resolve DNS

Symptoms:

- UE has IP address but cannot access internet
- DNS lookups fail

Possible Causes:

1. Incorrect DNS server addresses in PCO config
2. DNS servers not reachable from UE IP pool
3. Firewall blocking DNS traffic

Resolution:

```
# Test DNS server reachability
ping 8.8.8.8

# Test DNS resolution from UE network
nslookup google.com 8.8.8.8

# Verify PCO configuration
grep "primary_dns_server_address" config/runtime.exs
```

Issue: IMS Registration Fails

Symptoms:

- VoLTE calls fail
- UE shows "No IMS registration"

Possible Causes:

1. Missing P-CSCF configuration
2. Incorrect P-CSCF IP addresses
3. P-CSCF not reachable

Resolution:

```
# Verify P-CSCF configuration
pco: %{
  p_cscf_ipv4_address_list: ["10.0.0.50"]  # Ensure not empty
}
```

Issue: MTU Problems

Symptoms:

- Some websites load, others don't
- Large file transfers fail
- Fragmentation issues

Possible Causes:

- MTU too large for tunneling overhead
- MTU too small causing excessive fragmentation

Resolution:

```
# Recommended: 1400 for GTP tunneling
pco: %{
  ipv4_link_mtu_size: 1400
}

# If still having issues, try lower
pco: %{
  ipv4_link_mtu_size: 1360
}
```

Best Practices

DNS Configuration

1. **Use Reliable DNS Servers**

    - Public: Google (8.8.8.8), Cloudflare (1.1.1.1)
    - Private: Internal DNS for enterprise

2. **Always Configure Secondary**

    - Provides redundancy
    - Improves reliability

3. **Consider DNS Security**

    - DNSSEC-capable resolvers
    - DNS filtering for security

IMS Configuration

1. **Provide Multiple P-CSCF**
   - At least 2 for redundancy
   - Geographic distribution if possible
2. **Ensure Reachability**
   - P-CSCF must be reachable from UE IP pool
   - Test SIP connectivity

MTU Optimization

1. **Account for Overhead**
   - GTP-U: 36 bytes (IPv4)
   - IPsec: Variable (50-100 bytes)
2. **Standard LTE MTU**
   - Recommended: **1400 bytes**
   - Balances throughput and compatibility
3. **Test End-to-End**
   - Path MTU discovery
   - Test with large packets

Related Documentation

Configuration Guides

- **Configuration Guide** - Complete runtime.exs reference, UPF selection with PCO overrides
- **UE IP Allocation** - IP pool management, APN-based allocation

- **P-CSCF Monitoring** - P-CSCF discovery monitoring, health tracking, metrics

Session and Interface Management

- **Session Management** - PDN session lifecycle, bearer establishment
- **S5/S8 Interface** - GTP-C protocol, PCO encoding and delivery
- **PFCP Interface** - User plane session establishment

IMS and VoLTE

- **Diameter Gx Interface** - Policy control for IMS bearers
- **Monitoring Guide** - PCO-related metrics and dashboards

Back to Operations Guide

P-CSCF Discovery and Monitoring

Dynamic P-CSCF Server Discovery with Real-Time Monitoring

OmniPGW by Omnitouch Network Services

Overview

P-CSCF (Proxy Call Session Control Function) Discovery and Monitoring provides dynamic discovery of IMS P-CSCF servers using DNS SRV queries with real-time SIP OPTIONS health checking. This feature enables:

- **Per-Rule P-CSCF Discovery**: Different P-CSCF servers for different traffic types
- **Automatic Monitoring**: Background process continuously monitors DNS resolution (every 60 seconds)
- **SIP OPTIONS Health Checks**: Verifies P-CSCF servers are alive via SIP OPTIONS pings
  - **TCP First**: Attempts SIP OPTIONS via TCP (preferred for reliability)
  - **UDP Fallback**: Falls back to UDP if TCP fails
  - **Status Tracking**: Marks each server as :up or :down based on response
- **Real-Time Health Tracking**: OAM API exposes resolution status, discovered IPs, and health status
- **Graceful Fallback**: Three-tier fallback strategy for maximum reliability
- **Prometheus Metrics**: Full observability via Prometheus metrics

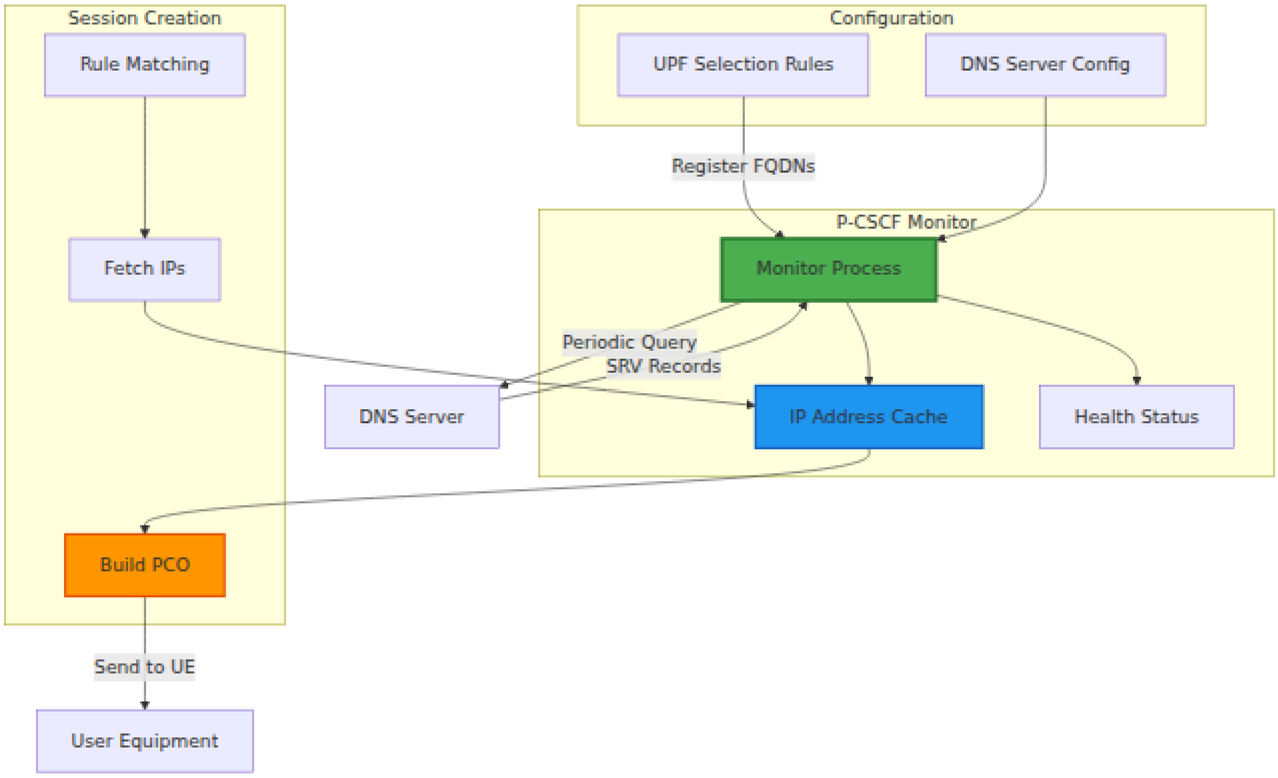


Table of Contents

Quick Start

Basic Configuration

```
# config/runtime.exs

# Global PCO configuration (DNS server for P-CSCF discovery)
config :pgw_c,
  pco: %{
    p_cscf_discovery_dns_server: "192.0.2.177",
    p_cscf_discovery_enabled: true,
    p_cscf_discovery_timeout_ms: 5000
  },

  upf_selection: %{
    rules: [
      # IMS Traffic - Dynamic P-CSCF discovery
      %{
        name: "IMS Traffic",
        priority: 20,
        match_field: :apn,
        match_regex: "^ims",
        upf_pool: [
          %{remote_ip_address: "10.100.2.21", remote_port: 8805,
weight: 80}
        ],
        # P-CSCF Discovery FQDN (see Configuration Guide for more
UPF selection rules)
        p_cscf_discovery_fqdn:
"pcscf.mnc001.mcc001.3gppnetwork.org",
        # Static fallback (see PCO Configuration Guide)
        pco: %{
          p_cscf_ipv4_address_list: ["203.0.113.100",
"203.0.113.101"]
        }
      }
    ]
  }
```

See Configuration Guide for complete UPF selection rule configuration and PCO Configuration for static P-CSCF fallback options.

Access Monitoring

1. Start OmniPGW
2. Query the OAM API: `curl -k https://localhost:8443/api/pcscf_monitor`
3. Inspect the returned resolution status and discovered IPs (re-run the request for current state)

Configuration

Global P-CSCF Discovery Settings

Configure the DNS server used for P-CSCF discovery in the PCO section:

```
pco: %{
  # DNS server for P-CSCF discovery (separate from DNS given to
UE)
  p_cscf_discovery_dns_server: "192.0.2.177",

  # Enable P-CSCF DNS discovery feature
  p_cscf_discovery_enabled: true,

  # Timeout for DNS SRV queries (milliseconds)
  p_cscf_discovery_timeout_ms: 5000,

  # Static P-CSCF addresses (global fallback)
  p_cscf_ipv4_address_list: ["203.0.113.146"]
}
```

Per-Rule P-CSCF FQDNs

Each UPF selection rule can specify its own P-CSCF discovery FQDN:

```
upf_selection: %{
  rules: [
    # IMS Traffic - IMS-specific P-CSCF
    %{
      name: "IMS Traffic",
      match_field: :apn,
      match_regex: "^ims",
      upf_pool: [...],
      p_cscf_discovery_fqdn:
"pcscf.ims.mnc001.mcc001.3gppnetwork.org",
      pco: %{
        p_cscf_ipv4_address_list: ["203.0.113.100"]  # Fallback
      }
    },

    # Enterprise - Enterprise-specific P-CSCF
    %{
      name: "Enterprise Traffic",
      match_field: :apn,
      match_regex: "^enterprise",
      upf_pool: [...],
      p_cscf_discovery_fqdn: "pcscf.enterprise.example.com",
      pco: %{
        p_cscf_ipv4_address_list: ["192.168.1.50"]  # Fallback
      }
    },

    # Internet - No P-CSCF discovery (uses global config)
    %{
      name: "Internet Traffic",
      match_field: :apn,
      match_regex: "^internet",
      upf_pool: [...]
      # No p_cscf_discovery_fqdn - uses global PCO config
    }
  ]
}
```

How It Works

Startup Process

1. **Application Starts**

   - P-CSCF Monitor GenServer initializes
   - Config parser extracts all unique P-CSCF FQDNs from UPF selection rules

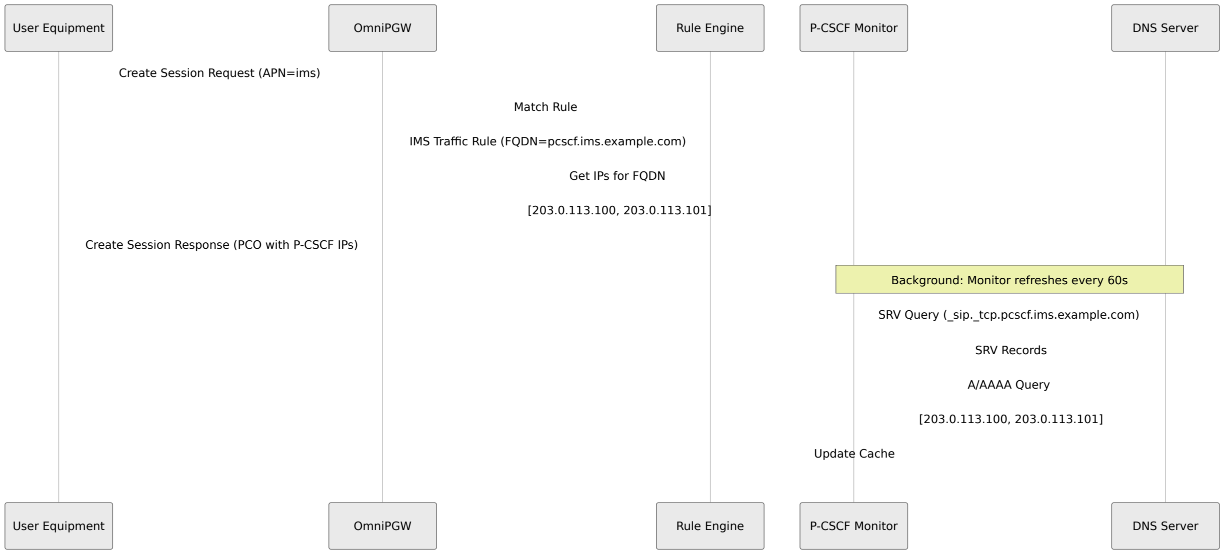
2. **FQDN Registration**

   - Each unique FQDN is registered with the monitor
   - Monitor performs initial DNS SRV query for each FQDN
   - **SIP OPTIONS Health Check** (in parallel for all discovered servers):
     - Try TCP first (`SIP/2.0/TCP` on port 5060)
     - If TCP fails, fall back to UDP (`SIP/2.0/UDP` on port 5060)
     - Mark each server as `:up` (responds) or `:down` (no response/timeout)
   - Results (IPs, health status, or errors) are cached with timestamps

3. **Periodic Monitoring** (Every 60 seconds)

   - Monitor refreshes all FQDNs
   - DNS queries run in background without blocking
   - For each discovered server:
     - Send SIP OPTIONS via TCP (timeout: 5 seconds)
     - If TCP fails, try UDP (timeout: 5 seconds)
     - Update health status based on response
   - Cache is updated with latest DNS results and health status

Session Creation Flow



DNS Query Process

The monitor uses **DNS SRV records** for direct P-CSCF discovery:

1. **SRV Query**: Query SRV records at `_sip._tcp.{fqdn}`
2. **Priority Sorting**: Sort by priority and weight
3. **Target Extraction**: Extract target hostnames from SRV records
4. **Hostname Resolution**: Resolve target hostnames to IP addresses (A/AAAA records)
5. **Caching**: Cache resolved IPs with status and timestamp

P-CSCF Address Selection Precedence

When both FQDN and static PCO are configured on a rule, FQDN takes precedence:

```
%{
  name: "IMS Traffic",
  p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org",  #
← Tried FIRST
  pco: %{
    p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"]
# ← Fallback
  }
}
```

Selection Logic:

Condition	P-CSCF Source	IPs Used	Log Mess
FQDN resolves successfully	DNS Discovery (Monitor)	Discovered IPs from DNS	`"Using P-CS` `addresses fr` `FQDN` `pcscf.exampl`
FQDN fails to resolve	Rule PCO Override	Static IPs from `pco.p_cscf_ipv4_address_list`	`"Failed to` `CSCF IPs fro` `FQDN..., fal` `back to stat` `config"`
FQDN returns empty list	Rule PCO Override	Static IPs from `pco.p_cscf_ipv4_address_list`	Fallback trigg
Monitor unavailable	Rule PCO Override	Static IPs from `pco.p_cscf_ipv4_address_list`	Error triggers fallback
No FQDN configured	Rule PCO Override or Global	Static IPs from rule or global config	Uses static co directly

Example Flow:

```
Session Creation for IMS Traffic Rule:
┌─────────────────────────────────────┐
│ 1. Check if FQDN configured?        │
│    ✓ Yes: "pcscf.mnc001.mcc001..." │
└─────────────────────────────────────┘
              │
              ▼
┌─────────────────────────────────────┐
│ 2. Query Monitor for cached IPs     │
│    Monitor.get_ips(fqdn)             │
└─────────────────────────────────────┘
              │
      ┌───────┴────────┐
      ▼                ▼
┌─────────────┐  ┌──────────────────┐
│ SUCCESS     │  │ FAILED/EMPTY     │
│ {:ok, ips}  │  │ {:error, reason} │
└─────────────┘  └──────────────────┘
      │                  │
      ▼                  ▼
┌─────────────┐  ┌──────────────────┐
│ Use DNS IPs │  │ Use Static PCO   │
│ [from DNS]  │  │ [from config]    │
└─────────────┘  └──────────────────┘
      │                  │
      └────────┬─────────┘
               ▼
      ┌──────────────────┐
      │ Send to UE in    │
      │ PCO message      │
      └──────────────────┘
```

Real-World Scenarios:

Scenario 1: DNS Discovery Works

```
Config:
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  pco.p_cscf_ipv4_address_list: ["203.0.113.100"]

DNS Result: [203.0.113.150, 203.0.113.151]
UE Receives: [203.0.113.150, 203.0.113.151] ← From DNS
Note: Static PCO is ignored when DNS succeeds
```

Scenario 2: DNS Fails, Graceful Fallback ⚠

```
Config:
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  pco.p_cscf_ipv4_address_list: ["203.0.113.100"]

DNS Result: ERROR :no_naptr_records
UE Receives: [203.0.113.100] ← From static PCO
Note: Session succeeds despite DNS failure
```

Scenario 3: No FQDN Configured

```
Config:
  # No p_cscf_discovery_fqdn
  pco.p_cscf_ipv4_address_list: ["192.168.1.50"]

UE Receives: [192.168.1.50] ← From static PCO
Note: DNS discovery not attempted
```

Why This Design?

1. **Prefer Dynamic**: DNS provides flexibility, load balancing, and location-aware routing
2. **Ensure Reliability**: Static fallback ensures sessions never fail due to DNS issues
3. **Zero Manual Intervention**: Automatic failover without operator involvement
4. **Production Safe**: Best of both worlds - agility with stability

Recommendation: Always configure both FQDN and static PCO for production deployments:

```
# ✓ RECOMMENDED: Dynamic with fallback
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com",  # Preferred
  pco: %{
    p_cscf_ipv4_address_list: ["203.0.113.100"]     # Safety net
  }
}

# ⚠ RISKY: Dynamic only (falls back to global PCO)
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  # No rule-specific fallback!
}

# ✓ VALID: Static only (no DNS overhead)
%{
  pco: %{
    p_cscf_ipv4_address_list: ["192.168.1.50"]
  }
}
```

OAM API Monitoring

P-CSCF Monitor Endpoint

Monitoring is exposed through the OAM REST API (HTTPS/TLS on port `8443`). All routes are served under the `/api` prefix. Query the P-CSCF discovery monitor with:

```
GET /api/pcscf_monitor
```

```
curl -k https://localhost:8443/api/pcscf_monitor
```

The endpoint takes no path or query parameters and returns the current discovery state on each call. Interactive API documentation (Swagger UI) is available at `https://localhost:8443/api/docs`.

Response contents:

- **Overview Statistics**
    - Total FQDNs monitored
    - Successfully resolved FQDNs
    - Failed resolutions
    - Total discovered P-CSCF IPs
- **Per-FQDN Details**
    - FQDN being monitored
    - Resolution status (Resolved / Failed / Pending)
    - Number of discovered IPs
    - List of resolved IP addresses (with per-server details)
    - Last update timestamp
    - **Health Status**: Each discovered server reports:
        - IP address and port
        - Hostname (from DNS SRV target)
        - Health indicator (Up / Down)
- **Monitoring Metrics**
    - **Total FQDNs**: Number of unique FQDNs registered for monitoring
    - **Successfully Resolved**: FQDNs that successfully resolved via DNS
    - **Failed DNS Resolutions**: FQDNs that failed to resolve
    - **Total P-CSCF Servers**: Total number of servers discovered across all FQDNs
    - **Healthy (SIP OPTIONS UP)**: Servers responding to SIP OPTIONS health checks
    - **Unhealthy (SIP OPTIONS DOWN)**: Servers not responding to SIP OPTIONS
    - **DNS Success Rate**: Percentage of successful DNS resolutions

    - **Health Check Interval**: Frequency of SIP OPTIONS health checks (60s, 5s timeout)

The response provides current visibility into both DNS resolution health and P-CSCF server availability via SIP OPTIONS. To refresh, simply re-run `GET /api/pcscf_monitor` — each call returns the latest state.

UPF Selection Status

The UPF selection status endpoint (`GET /api/upf_selection`) reports P-CSCF discovery status for each rule:

```
 IMS Traffic (Priority 20)
   Match: APN matching ^ims
   Pool: UPF-IMS-Primary (10.100.2.21:8805)

    P-CSCF Discovery
     FQDN: pcscf.mnc001.mcc001.3gppnetwork.org
     Status: ✓ Resolved (2 IPs)
     Resolved IPs: 203.0.113.100, 203.0.113.101

   ⚙ PCO Overrides
     Primary DNS: 198.51.100.195
     P-CSCF (static fallback): 203.0.113.100, 203.0.113.101
```

Metrics and Observability

Prometheus Metrics

The P-CSCF monitoring system exposes metrics via Prometheus (port 42069 by default):

Gauge Metrics

```
# FQDN-level metrics
pcscf_fqdns_total                       # Total number of monitored
FQDNs
pcscf_fqdns_resolved                    # Successfully resolved
FQDNs (DNS succeeded)
pcscf_fqdns_failed                      # Failed FQDN resolutions
(DNS failed)

# Server-level metrics (aggregate)
pcscf_servers_total                     # Total P-CSCF servers
discovered via DNS SRV
pcscf_servers_healthy                   # Servers responding to SIP
OPTIONS (aggregate)
pcscf_servers_unhealthy                 # Servers not responding to
SIP OPTIONS (aggregate)

# Server-level metrics (per-FQDN with label)
pcscf_servers_healthy{fqdn="..."}      # Healthy servers for
specific FQDN
pcscf_servers_unhealthy{fqdn="..."}    # Unhealthy servers for
specific FQDN
```

Health Check Details:

- `healthy`: Server responded to SIP OPTIONS ping (TCP or UDP)
- `unhealthy`: Server failed to respond to SIP OPTIONS (5s timeout per transport)

Metric Examples

DNS Resolution Metrics:

```
# Query successfully resolved FQDNs
pcscf_fqdns_resolved

# Calculate DNS success rate
(pcscf_fqdns_resolved / pcscf_fqdns_total) * 100

# Total discovered servers
pcscf_servers_total
```

SIP OPTIONS Health Metrics:

```
# Total healthy servers across all FQDNs
pcscf_servers_healthy

# Total unhealthy servers
pcscf_servers_unhealthy

# Calculate health check success rate
(pcscf_servers_healthy / pcscf_servers_total) * 100

# Healthy servers for a specific FQDN
pcscf_servers_healthy{fqdn="pcscf.mnc001.mcc001.3gppnetwork.org"}

# Alert on all servers down
pcscf_servers_healthy == 0 AND pcscf_servers_total > 0
```

Example Prometheus Alerts:

```yaml
# Alert when all P-CSCF servers are down
- alert: AllPCSCFServersDown
  expr: pcscf_servers_healthy == 0 AND pcscf_servers_total > 0
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "All P-CSCF servers are unhealthy"
    description: "{{ $value }} healthy servers (0) - all failed
SIP OPTIONS checks"

# Alert when more than 50% servers are down
- alert: MajorityPCSCFServersDown
  expr: (pcscf_servers_healthy / pcscf_servers_total) < 0.5
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "Majority of P-CSCF servers are unhealthy"
    description: "Only {{ $value }}% of servers are responding to
SIP OPTIONS"

# Alert on DNS resolution failures
- alert: PCSCFDNSResolutionFailed
  expr: pcscf_fqdns_failed > 0
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "P-CSCF DNS resolution failures"
    description: "{{ $value }} FQDN(s) failing to resolve"
```

Logging

The monitor logs key events:

```
[info] P-CSCF Monitor started
[info] Registering 2 unique P-CSCF FQDNs for monitoring:
["pcscf.ims.example.com", "pcscf.enterprise.example.com"]
[info] P-CSCF Monitor: Registering FQDN pcscf.ims.example.com
[debug] P-CSCF Monitor: Successfully resolved
pcscf.ims.example.com to 2 IPs
[warning] P-CSCF Monitor: Failed to resolve
pcscf.enterprise.example.com: :nxdomain
[debug] Using P-CSCF addresses from FQDN pcscf.ims.example.com:
[{10, 101, 2, 100}, {10, 101, 2, 101}]
```

Fallback Strategy

The system uses a **three-tier fallback strategy** for maximum reliability:

Tier 1: DNS Discovery (Preferred)

```
p_cscf_discovery_fqdn: "pcscf.ims.example.com"
```

- Monitor queries DNS and caches resolved IPs
- Session uses cached IPs if available
- **Advantage**: Dynamic, load-balanced, location-aware

Tier 2: Rule-Specific Static PCO (Fallback)

```
pco: %{
  p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"]
}
```

- Used if DNS discovery fails or returns no IPs
- Rule-specific static configuration
- **Advantage**: Rule-specific fallback, predictable

Tier 3: Global PCO Configuration (Last Resort)

```
# Global pco config
pco: %{
  p_cscf_ipv4_address_list: ["203.0.113.146"]
}
```

- Used if no rule-specific config and DNS fails
- Global default P-CSCF addresses
- **Advantage**: Always available, prevents session failure

Fallback Logic Example

```
Session matches "IMS Traffic" rule:

1. Try DNS discovery for "pcscf.ims.example.com"
   ├─ Success → Use [203.0.113.100, 203.0.113.101] ✓
   └─ Failed → Try next tier

2. Try rule's PCO override
   ├─ Configured → Use [203.0.113.100, 203.0.113.101] ✓
   └─ Not configured → Try next tier

3. Use global PCO config
   └─ Use [203.0.113.146] ✓ (Always succeeds)
```

DNS Configuration

DNS Server Setup

Configure DNS server with SRV and A/AAAA records for P-CSCF discovery:

```
; SRV records for P-CSCF (_sip._tcp prefix is queried
automatically)
_sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org. IN SRV 10 50 5060
pcscf1.example.com.
_sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org. IN SRV 20 50 5060
pcscf2.example.com.

; A records
pcscf1.example.com. IN A 203.0.113.100
pcscf2.example.com. IN A 203.0.113.101
```

Important: OmniPGW automatically prepends `_sip._tcp.` to the configured FQDN. If you configure `p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org"`, the system will query `_sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org`.

SRV Record Format

SRV records follow this format:

```
_service._proto.domain. IN SRV priority weight port target.
```

- **Priority**: Lower values have higher priority (10 before 20)
- **Weight**: For load balancing among same priority (higher = more traffic)
- **Port**: SIP port (typically 5060 for TCP, 5060 for UDP)
- **Target**: Hostname to resolve to IP address

Testing DNS Configuration

```
# Query SRV records (note the _sip._tcp prefix)
dig SRV _sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org @192.0.2.177

# Expected output:
# _sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org. 300 IN SRV 10 50
5060 pcscf1.example.com.

# Resolve P-CSCF hostname to IP
dig A pcscf1.example.com @192.0.2.177

# Expected output:
# pcscf1.example.com. 300 IN A 203.0.113.100
```

Troubleshooting

Issue: FQDN Shows "Failed" Status

Symptoms:

- `GET /api/pcscf_monitor` reports Failed status
- Error: `:nxdomain`, `:timeout`, or `:no_naptr_records`

Possible Causes:

1. DNS server not reachable
2. FQDN does not exist in DNS
3. No NAPTR records configured
4. DNS server timeout

Resolution:

```
# 1. Test DNS server connectivity
ping 192.0.2.177

# 2. Test NAPTR query manually
dig NAPTR pcscf.mnc001.mcc001.3gppnetwork.org @192.0.2.177

# 3. Check OmniPGW logs
grep "P-CSCF" /var/log/pgw_c.log

# 4. Verify configuration
grep "p_cscf_discovery_dns_server" config/runtime.exs

# 5. Re-query the OAM API for current state
curl -k https://localhost:8443/api/pcscf_monitor
```

Issue: No IPs Returned

Symptoms:

- `GET /api/pcscf_monitor` reports "0 IPs"
- Status may be Resolved or Failed

Possible Causes:

1. NAPTR records exist but replacement FQDNs don't resolve
2. Service field doesn't match IMS/SIP pattern
3. A/AAAA records missing

Resolution:

```
# Check NAPTR record service field
dig NAPTR pcscf.example.com @192.0.2.177

# Ensure service contains "SIP" or "IMS":
# CORRECT: "SIP+D2U", "x-3gpp-ims:sip"
# WRONG: "HTTP", "FTP"

# Check A/AAAA records exist
dig pcscf1.example.com A @192.0.2.177
```

Issue: Sessions Use Wrong P-CSCF

Symptoms:

- UE receives unexpected P-CSCF addresses
- Static fallback used instead of discovered IPs

Possible Causes:

1. DNS discovery failed but fallback is working
2. Rule matching incorrect
3. FQDN not registered

Resolution:

```
# 1. Check P-CSCF Monitor page
# Verify FQDN is registered and resolved

# 2. Check session logs
grep "Using P-CSCF addresses from FQDN" /var/log/pgw_c.log

# 3. Check UPF Selection page
# Verify rule shows correct FQDN and status

# 4. Test rule matching
# Create session with specific APN and verify which rule matches
```

Issue: High DNS Query Latency

Symptoms:

- Slow session creation
- Metrics show high `pcscf_discovery_query_duration_seconds`

Possible Causes:

1. DNS server performance issues
2. Network latency to DNS server
3. Timeout too high

Resolution:

```
# Reduce query timeout
pco: %{
  p_cscf_discovery_timeout_ms: 2000  # Reduce from 5000ms
}

# Consider using closer DNS server
pco: %{
  p_cscf_discovery_dns_server: "10.0.0.10"  # Local DNS
}
```

Best Practices

1. DNS Server Selection

Use Dedicated DNS Server

```
pco: %{
  # Dedicated DNS for P-CSCF discovery (not the same as UE DNS)
  p_cscf_discovery_dns_server: "192.0.2.177",

  # UE DNS servers (given to mobile devices)
  primary_dns_server_address: "8.8.8.8",
  secondary_dns_server_address: "8.8.4.4"
}
```

Why?

- Separate concerns: UE DNS vs. internal IMS DNS
- Different access policies and security
- Independent scaling and reliability

2. Always Configure Static Fallback

```
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com",  # Preferred
  pco: %{
    p_cscf_ipv4_address_list: ["203.0.113.100"]  # Required
fallback
  }
}
```

Why?

- Ensures sessions succeed even if DNS fails
- Graceful degradation
- Meets SLA requirements

3. Use Specific FQDNs per Traffic Type

```
rules: [
  # IMS
  %{
    name: "IMS",
    match_regex: "^ims",
    p_cscf_discovery_fqdn:
"pcscf.ims.mnc001.mcc001.3gppnetwork.org"
  },

  # Enterprise
  %{
    name: "Enterprise",
    match_regex: "^enterprise",
    p_cscf_discovery_fqdn: "pcscf.enterprise.example.com"
  }
]
```

Why?

- Different P-CSCF pools per service
- Better load distribution

- Service-specific routing

4. Monitor DNS Query Performance

```
# Alert on high P-CSCF query latency
alert: HighPCSCFQueryLatency
expr: histogram_quantile(0.95,
pcscf_discovery_query_duration_seconds_bucket) > 2
for: 5m
labels:
  severity: warning
annotations:
  summary: "P-CSCF DNS queries are slow (p95 > 2s)"
```

5. Regular DNS Health Checks

- **OAM API**: Poll `GET /api/pcscf_monitor` daily
- **Metrics**: Monitor `pcscf_monitor_fqdns_failed` metric
- **Logs**: Watch for DNS errors
- **Testing**: Periodically verify DNS records exist

6. Configure Appropriate Timeout

```
# Production: Balance reliability vs. latency
pco: %{
  p_cscf_discovery_timeout_ms: 5000  # 5 seconds
}

# High-performance: Favor speed, rely on fallback
pco: %{
  p_cscf_discovery_timeout_ms: 2000  # 2 seconds
}
```

7. Use DNS Redundancy

Configure primary and secondary DNS:

```
# Primary P-CSCF DNS
pcscf.mnc001.mcc001.3gppnetwork.org. IN NAPTR 10 50 "s" "SIP+D2U"
"" _sip._udp.pcscf1.example.com.

# Secondary P-CSCF DNS
pcscf.mnc001.mcc001.3gppnetwork.org. IN NAPTR 20 50 "s" "SIP+D2U"
"" _sip._udp.pcscf2.example.com.
```

Related Documentation

- **PCO Configuration** - Protocol Configuration Options, DNS and P-CSCF settings
- **Configuration Guide** - Complete OmniPGW configuration reference
- **Monitoring** - Metrics, logging, and observability
- **Session Management** - Session lifecycle and PCO delivery
- **PFCP Interface** - User Plane Function communication

Back to Main Documentation

PFCP/Sxb Interface Documentation

Packet Forwarding Control Protocol - PGW-C to PGW-U Communication

---

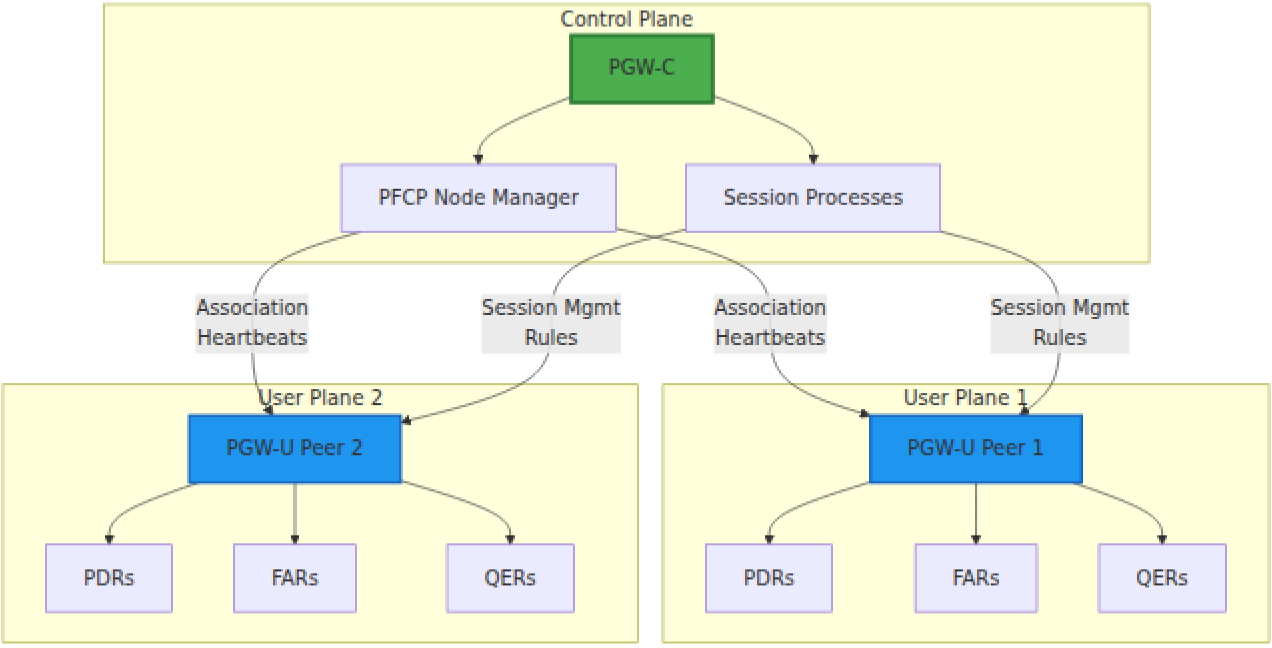
Table of Contents

Overview

The **Sxb interface** uses the **PFCP (Packet Forwarding Control Protocol)** for communication between the PGW-C (control plane) and PGW-U (user plane). This separation allows:

- **Control Plane (PGW-C)** - Handles signaling, session management, policy decisions
- **User Plane (PGW-U)** - Handles actual packet forwarding at high speed

PFCP Architecture



Protocol Basics

PFCP Version

PGW-C implements **PFCP Version 1** (3GPP TS 29.244).

Transport

- **Protocol:** UDP
- **Default Port:** 8805
- **Message Format:** Binary encoded using PFCP specification

Node ID Types

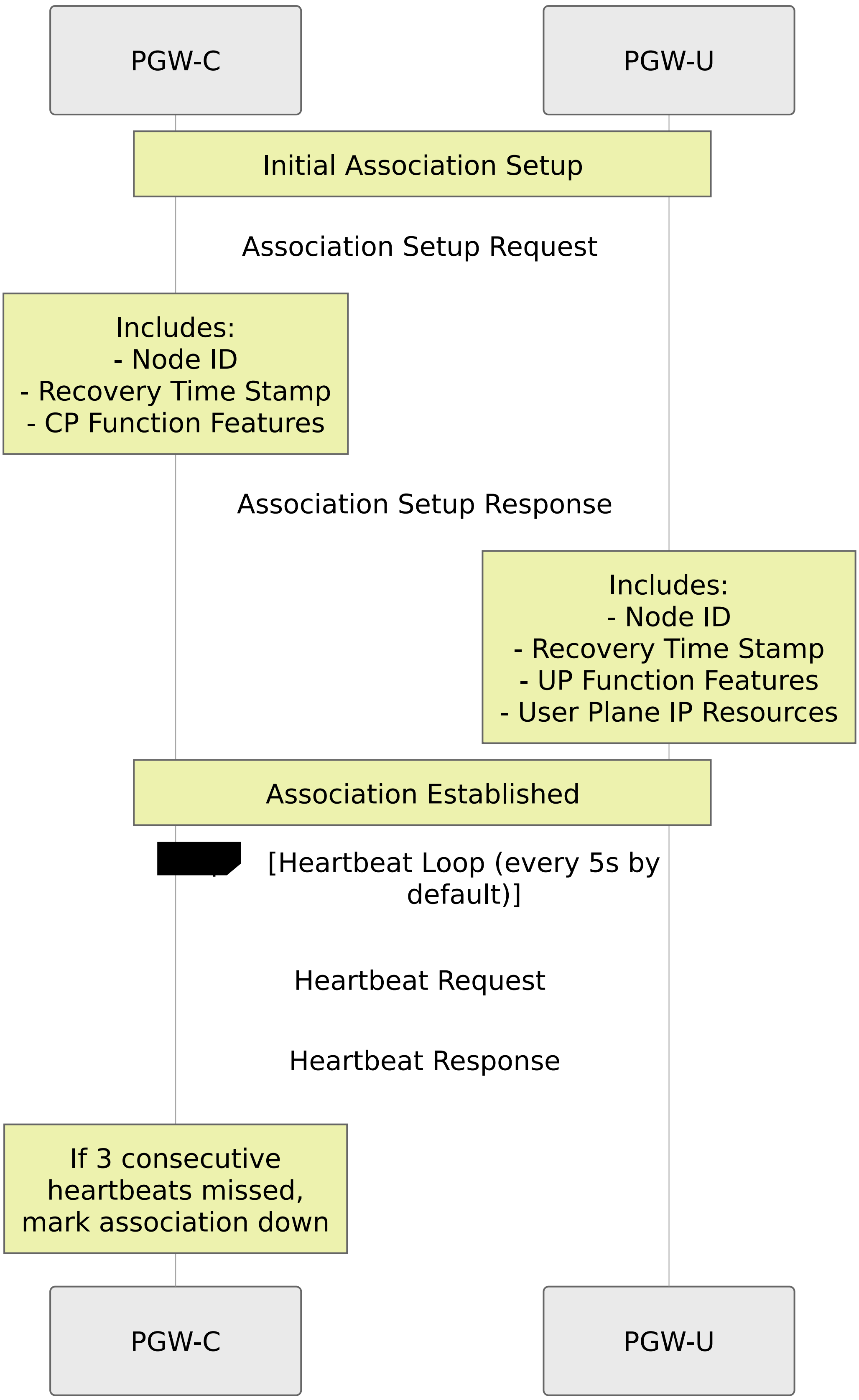
PFCP peers are identified by Node ID, which can be:

- **IPv4 Address** - Most common
- **IPv6 Address**
- **FQDN** (Fully Qualified Domain Name)

PFCP Association Management

Before session management can occur, a PFCP **association** must be established between PGW-C and PGW-U.

Association Setup Flow



Peer State Management

Each PFCP peer maintains state:

Field	Description
`is_associated`	Boolean indicating association status
`remote_node_id`	Peer's Node ID (IP or FQDN)
`remote_ip_address`	IP address for communication
`remote_port`	UDP port (default 8805)
`heartbeat_period_ms`	Heartbeat interval in milliseconds
`missed_heartbeats_consecutive`	Count of missed heartbeats
`up_function_features`	Supported user plane features
`up_recovery_time_stamp`	Peer's recovery timestamp

Heartbeat Mechanism

Purpose: Detect peer failures and maintain association liveness

Configuration:

```
# In config/runtime.exs
sxb: %{
  local_ip_address: "10.0.0.20"
},
upf_selection: %{
  fallback_pool: [
    %{remote_ip_address: "10.0.0.21", remote_port: 8805, weight: 100}
  ]
}
# All UPFs are automatically registered with 5-second heartbeats
```

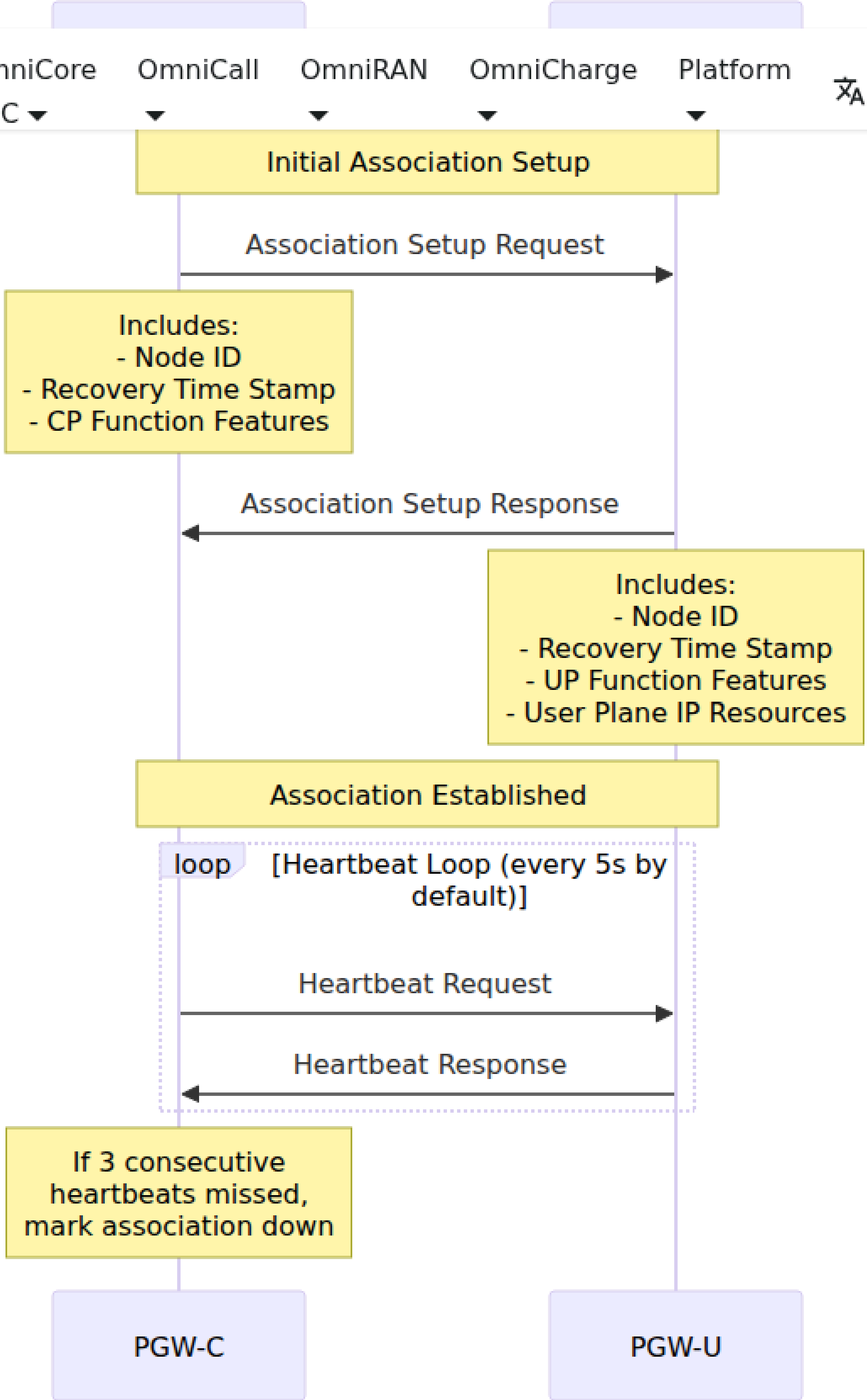
Failure Detection:

- Each missed heartbeat increments `missed_heartbeats_consecutive`
- Typically configured to fail after 3 consecutive misses
- Failed association prevents new sessions to that peer

PFCP Session Management

PFCP sessions are created for each UE PDN connection to program forwarding rules in the user plane.

Session Lifecycle



Session Establishment

When: UE attaches and creates a PDN connection

PGW-C sends to PGW-U:

Session Establishment Request containing:

- **SEID** (Session Endpoint ID) - Unique session identifier
- **Node ID** - PGW-C's Node ID
- **F-SEID** - Fully Qualified SEID (includes IP + SEID)
- **PDRs** - Packet Detection Rules (typically 2: uplink + downlink)
- **FARs** - Forwarding Action Rules (typically 2: uplink + downlink)
- **QERs** - QoS Enforcement Rules (bitrate limits)
- **BAR** - Buffering Action Rule (for downlink buffering)

PGW-U responds:

Session Establishment Response containing:

- **Cause** - Success or failure reason
- **F-SEID** - PGW-U's session endpoint
- **Created PDRs** - Acknowledgment of created rules
- **F-TEID** - Fully Qualified TEID for S5/S8 interface

Session Modification

When: QoS changes, policy updates, or bearer modifications occur

Modification can include:

- Adding new PDRs, FARs, QERs
- Removing existing rules
- Updating rule parameters

Session Deletion

When: UE detaches or PDN connection is terminated

Process:

1. PGW-C sends Session Deletion Request with SEID
2. PGW-U removes all rules and releases resources
3. PGW-U responds with Session Deletion Response

F-TEID Allocation

F-TEID (Fully Qualified Tunnel Endpoint Identifier) identifies GTP-U tunnel endpoints for user plane traffic. When establishing a PFCP session, someone must allocate the F-TEID that identifies where the UPF should send uplink traffic. There are two approaches:

Understanding F-TEID Allocation

What's Being Allocated: The F-TEID consists of:

- **TEID** (Tunnel Endpoint Identifier) - 32-bit number identifying the tunnel
- **IP Address** - Where to send GTP-U packets (the UPF's IP address)

The Question: Who allocates the TEID value?

Option 1: UPF Allocates (Recommended Default)

- PGW-C says "please allocate a TEID for me" (CHOOSE flag)
- UPF picks a TEID from its local pool and responds with the value

Option 2: PGW-C Allocates (Compatibility Mode)

- PGW-C picks a TEID and tells UPF "use this specific TEID"
- UPF uses the provided TEID without allocation

UPF Allocation (Default - Recommended)

Configuration:

```
sxb: %{
  allocate_uplink_f_teid: false  # Default
}
```

How It Works:

1. PGW-C builds PFCP Session Establishment Request with F-TEID CHOOSE flag
2. UPF receives request, allocates TEID from its internal pool
3. UPF responds with allocated F-TEID (TEID + IP address)
4. PGW-C stores allocated F-TEID for session lifetime

Why This is Better (Usually):

Separation of Concerns

- UPF owns user plane = UPF manages user plane identifiers
- No need for PGW-C to track what TEIDs UPF has available
- Each component manages its own resource pool

Multi-PGW-C Scalability

- Multiple PGW-C instances can talk to same UPF without coordination
- No risk of TEID collisions between different PGW-C instances
- UPF ensures uniqueness across all control plane peers

Standard 3GPP Behavior

- CHOOSE flag is defined in 3GPP TS 29.244 for this purpose
- Modern UPF implementations support it
- Follows "let the owner allocate" principle

Simpler Failover

- If PGW-C restarts, UPF still owns TEID namespace
- No need to synchronize TEID allocation state
- UPF can continue using existing TEIDs

When to Use:

- ✅ Production deployments with modern UPFs (default)
- ✅ Multi-PGW-C deployments sharing UPF pools
- ✅ Cloud-native architectures with stateless control planes
- ✅ You want standard 3GPP PFCP behavior

Potential Issues:

- ⚠ Some legacy or proprietary UPF implementations don't support CHOOSE flag
- ⚠ If session establishment fails with "mandatory IE missing" or similar, UPF may not support CHOOSE

PGW-C Allocation (Legacy Compatibility)

Configuration:

```
sxb: %{
  allocate_uplink_f_teid: true
}
```

How It Works:

1. PGW-C allocates TEID from local pool during session creation
2. PGW-C builds PFCP Session Establishment Request with explicit TEID value
3. UPF receives request, uses provided TEID without allocation
4. Both PGW-C and UPF track the same TEID value

Why You Might Need This:

❌ UPF Doesn't Support CHOOSE

- Some UPF implementations (especially legacy/proprietary) don't support dynamic allocation
- UPF expects explicit TEID in PFCP Session Establishment Request
- Only workaround for compatibility

❌ Centralized TEID Management

- If you need PGW-C to have full visibility into all allocated TEIDs
- Useful for debugging user plane issues (PGW-C knows exact TEID values)
- Can correlate TEID in packet captures with session state

□ **Deterministic Allocation**

- If you need predictable TEID allocation patterns
- Some test environments may require specific TEID ranges

Trade-offs:

⚠ **Coordination Required for Multi-PGW-C**

- Multiple PGW-C instances sharing a UPF must avoid TEID collisions
- Requires either:
    - Partitioned TEID ranges per PGW-C (complex configuration)
    - Shared TEID allocation service (additional infrastructure)
    - Accept collision risk with random allocation (low probability)

⚠ **State Synchronization**

- PGW-C must track allocated TEIDs to avoid reuse
- TEID pool state lost on PGW-C restart (must rebuild from sessions)
- More complex failover scenarios

⚠ **Non-Standard Behavior**

- Not the intended PFCP design pattern
- May not work with all UPF implementations expecting CHOOSE

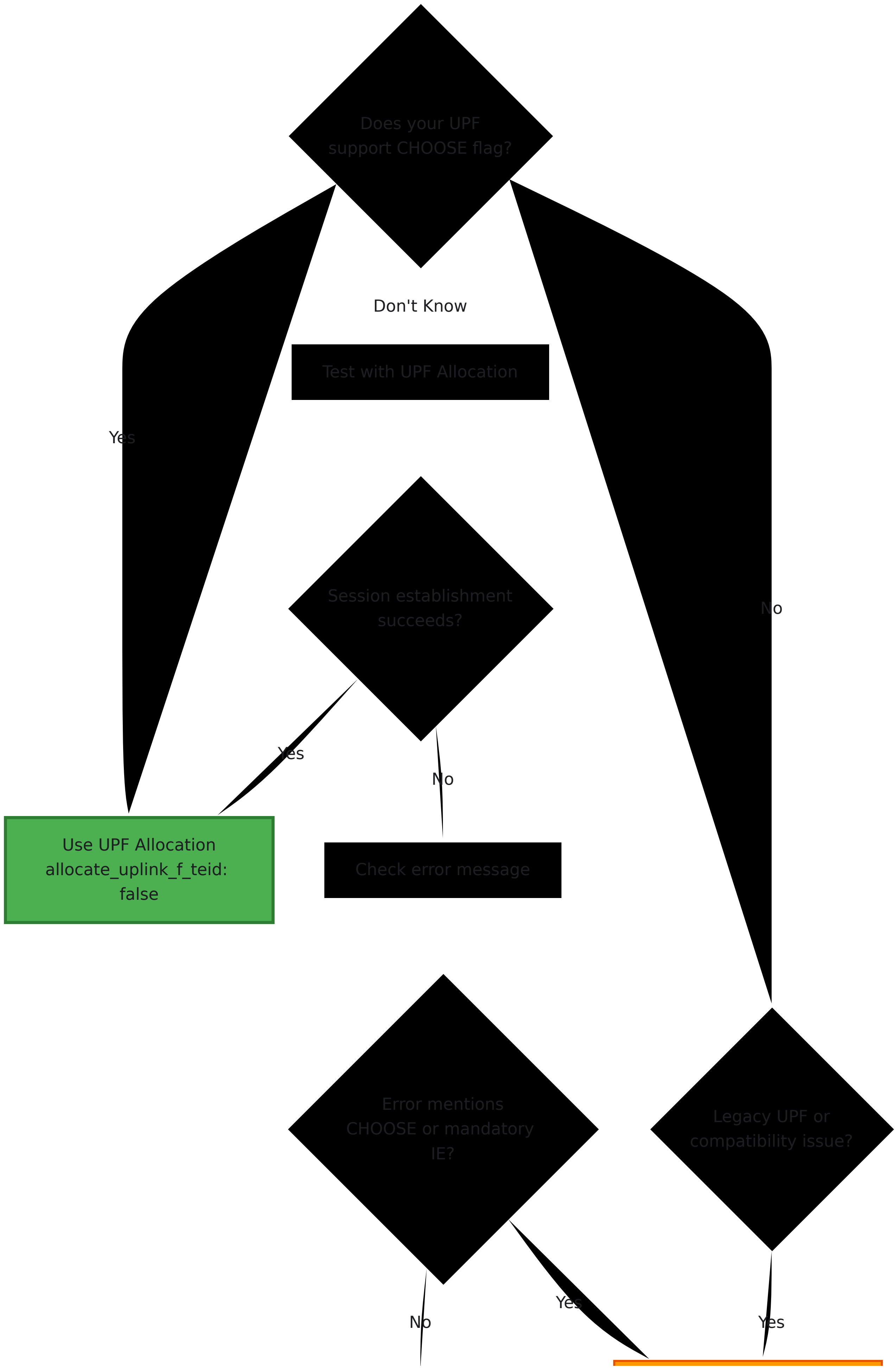
When to Use:

- ⚠ **Only when UPF doesn't support CHOOSE flag**
- ⚠ Legacy UPF implementations (e.g., some proprietary hardware)
- ⚠ Specific compatibility requirements
- ⚠ Debugging scenarios requiring PGW-C TEID visibility

TEID Collision Handling: PGW-C uses random allocation with collision detection:

- TEID range: 1 to 0xFFFFFFFF (4.2 billion values)
- Collision probability: ~0.023% at 1 million sessions
- Automatic retry on collision (transparent to caller)
- TEIDs automatically released when session terminates

How to Choose



Does your UPF support CHOOSE flag?
Don't Know
Test with UPF Allocation
Yes
Session establishment succeeds?
No
Yes
No
Use UPF Allocation allocate_uplink_f_teid: false
Check error message
Error mentions CHOOSE or mandatory IE?
Legacy UPF or compatibility issue?
No
Yes
Yes

Troubleshooting

Symptom: Session establishment fails immediately

Check PFCP logs:

```
# Look for CHOOSE-related errors
grep -i "choose\|mandatory.*missing" /var/log/pgw_c.log

# Check PFCP Session Establishment Response cause codes
grep "Session Establishment Response" /var/log/pgw_c.log
```

If UPF rejects CHOOSE flag:

- Error may say "Mandatory IE missing" or "Invalid IE"
- UPF expects explicit F-TEID but received CHOOSE
- **Solution:** Set `allocate_uplink_f_teid: true`

If PGW-C allocation causes issues:

- Very rare - TEID space is huge (4 billion values)
- Check for TEID exhaustion (unlikely below millions of sessions):

```
# Check registry count
grep "registered_teid_count" /var/log/pgw_c.log
```

Switching Between Modes:

```
# Edit config/runtime.exs
sxb: %{
  local_ip_address: "10.0.0.20",
  allocate_uplink_f_teid: false  # Change to true if UPF doesn't
support CHOOSE
}
```

Then restart PGW-C:

```
systemctl restart pgw_c
```

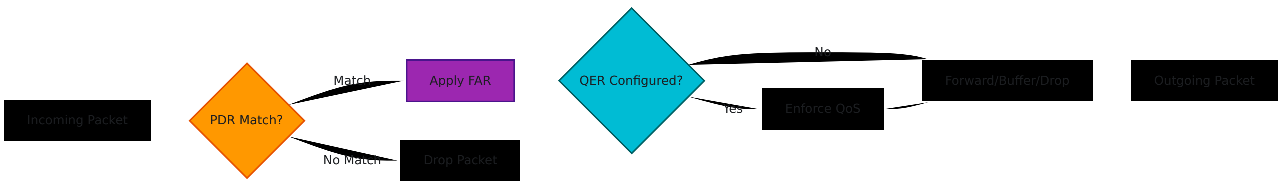
Verifying Which Mode is Active: Check PFCP packet captures:

```
# Capture PFCP traffic
tcpdump -i any -n port 8805 -w pfcp.pcap

# Open in Wireshark and look at Session Establishment Request
# If F-TEID shows "CHOOSE" flags: UPF allocation mode
# If F-TEID shows explicit TEID value: PGW-C allocation mode
```

Packet Processing Rules

PFCP uses a set of rules to define how the user plane processes packets.

Rule Architecture



PDR (Packet Detection Rule)

Purpose: Identify which packets this rule applies to

Typical PGW-C Configuration:

PDR #1 - Downlink:

```
PDR ID: 1
Precedence: 100
PDI (Packet Detection Information):
  - Source Interface: CORE (Internet side)
  - UE IP Address: 100.64.1.42/32
FAR ID: 1 (associated forwarding rule)
```

PDR #2 - Uplink:

```
PDR ID: 2
Precedence: 100
PDI (Packet Detection Information):
  - Source Interface: ACCESS (SGW side)
  - F-TEID: <S5/S8 tunnel endpoint>
FAR ID: 2 (associated forwarding rule)
QER ID: 1 (QoS enforcement)
```

Key PDR Fields:

- **PDR ID** - Unique rule identifier (per session)
- **Precedence** - Rule matching priority (higher = more specific)
- **PDI** - Matching criteria (interface, IP, TEID, etc.)
- **Outer Header Removal** - Strip GTP-U header on ingress
- **FAR ID** - Associated forwarding action
- **QER ID** - Associated QoS enforcement (optional)

FAR (Forwarding Action Rule)

Purpose: Define what to do with matched packets

FAR #1 - Downlink (Internet → UE):

```
FAR ID: 1
Apply Action: FORWARD
Forwarding Parameters:
  - Destination Interface: ACCESS (to SGW)
  - Outer Header Creation: GTP-U/UDP/IPv4
  - Remote F-TEID: <SGW S5/S8 tunnel endpoint>
```

FAR #2 - Uplink (UE → Internet):

```
FAR ID: 2
Apply Action: FORWARD
Forwarding Parameters:
  - Destination Interface: CORE (to Internet)
  - (No outer header - plain IP forwarding)
```

Key FAR Fields:

- **FAR ID** - Unique rule identifier
- **Apply Action** - FORWARD, DROP, BUFFER, NOTIFY
- **Forwarding Parameters:**
  - Destination interface (ACCESS/CORE)
  - Outer Header Creation (add GTP-U tunnel)
  - Network Instance (VRF/routing table)

QER (QoS Enforcement Rule)

Purpose: Enforce bitrate limits and QoS parameters. QERs can also track usage for online charging quota management (see Diameter Gy Interface for credit control).

Example QER:

```
QER ID: 1
Gate Status: OPEN
Maximum Bitrate:
  - Uplink: 100 Mbps
  - Downlink: 50 Mbps
Guaranteed Bitrate: (optional, for GBR bearers)
  - Uplink: 10 Mbps
  - Downlink: 10 Mbps
```

Key QER Fields:

- **QER ID** - Unique rule identifier
- **Gate Status** - OPEN (allow) or CLOSED (block)
- **MBR** - Maximum Bitrate (uplink/downlink)
- **GBR** - Guaranteed Bitrate (for dedicated bearers)
- **QCI** - QoS Class Identifier (affects scheduling)

BAR (Buffering Action Rule)

Purpose: Control downlink packet buffering when UE is idle

Example BAR:

```
BAR ID: 1
Downlink Data Notification Delay: 100ms
Suggested Buffering Packets Count: 10
```

Used for: Idle mode DRX (Discontinuous Reception) optimization

---

Configuration

Basic Sxb Configuration

Edit `config/runtime.exs`:

```
config :pgw_c,
  sxb: %{
    # Local IP address for PFCP communication
    local_ip_address: "10.0.0.20",

    # Optional: Override default port (8805)
    local_port: 8805,

    # PFCP request timeout in milliseconds (default: 500ms)
    # Time to wait for UPF response before retransmitting
    # Should be >= expected UPF processing time to avoid duplicate
sessions
    request_timeout_ms: 500,

    # Number of retry attempts for PFCP requests (default: 3)
    # Total maximum wait time = request_timeout_ms *
request_attempts
    request_attempts: 3,

    # Optional: Control F-TEID allocation for user plane
    # When false (default): UPF allocates F-TEID (CHOOSE flag)
    # When true: PGW-C pre-allocates F-TEID and provides explicit
value
    # Note: Some UPFs may not support CHOOSE flag and require
explicit allocation
    allocate_uplink_f_teid: false
  },

  # UPF Selection - All UPFs defined here are automatically
registered
  upf_selection: %{
    fallback_pool: [
      %{
        # PGW-U IP address
        remote_ip_address: "10.0.0.21",

        # PFCP port (default: 8805)
        remote_port: 8805,

        # Weight for load balancing (100 = normal, 0 = standby)
        weight: 100
      }
```

```
    ]
  }
```

Request Timeout Configuration

The PFCP interface uses configurable timeouts for Session Establishment, Modification, and Deletion requests to the UPF.

Parameters:

Parameter	Type	Default	Description
`request_timeout_ms`	Integer	500	Time in milliseconds to wait for UPF response before retransmitting
`request_attempts`	Integer	3	Maximum number of transmission attempts before failing the request

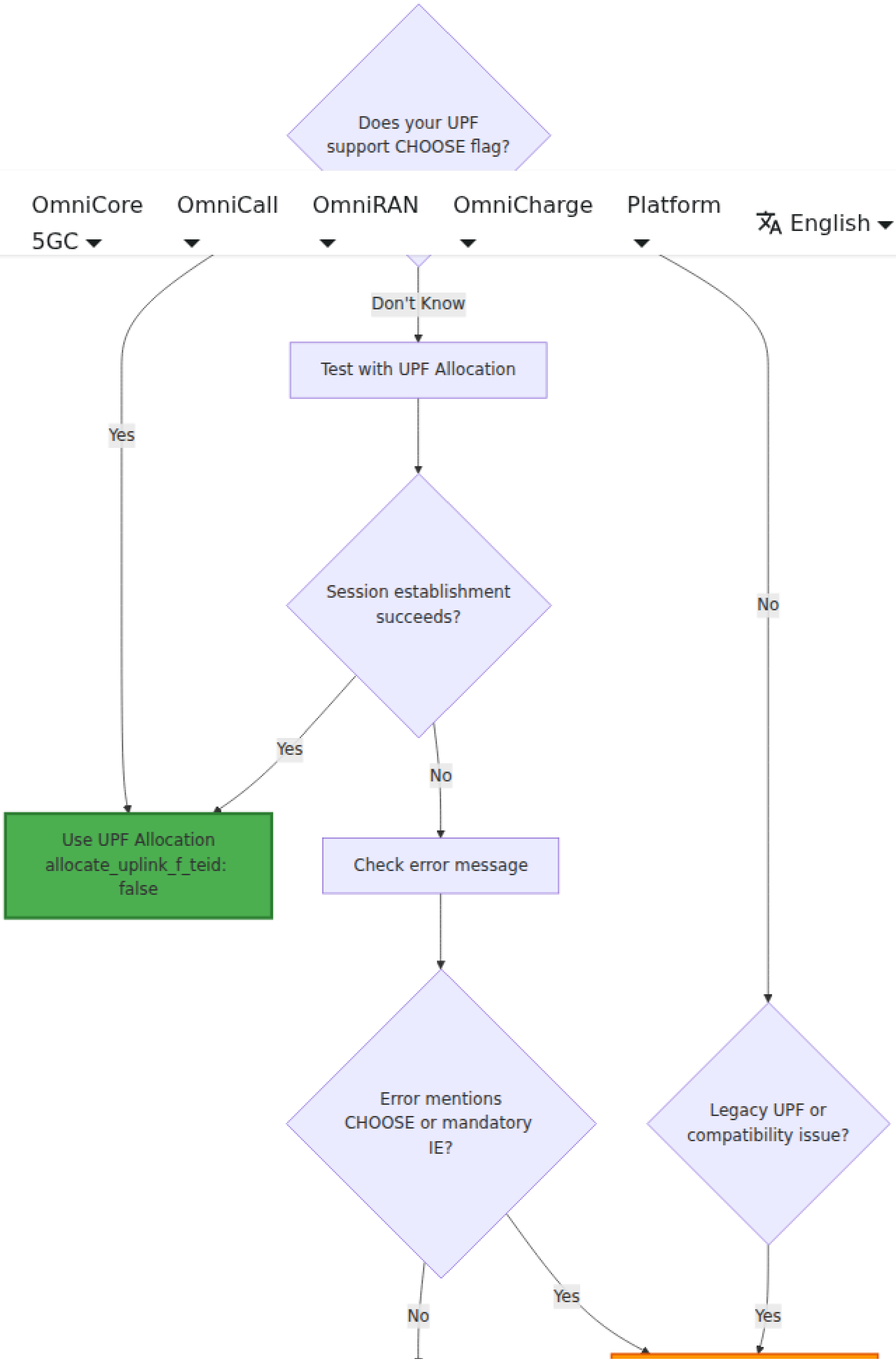
Total Wait Time: `request_timeout_ms × request_attempts`

Default behavior: 500ms × 3 attempts = **1.5 seconds total maximum wait**

Why Timeout Configuration Matters

If `request_timeout_ms` is set too low relative to UPF processing time:

1. PGW-C sends Session Establishment Request
2. Timeout expires before UPF responds
3. PGW-C retransmits with the same sequence number
4. UPF processes both requests and creates **duplicate PFCP sessions**
5. PGW-C receives first response and stores one session ID
6. Second session becomes **orphaned** in the UPF


Does your UPF support CHOOSE flag?
Don't Know
Test with UPF Allocation
Yes
Session establishment succeeds?
No
Yes
No
Use UPF Allocation allocate_uplink_f_teid: false
Check error message
Error mentions CHOOSE or mandatory IE?
Legacy UPF or compatibility issue?
No
Yes
Yes

Tuning Guidelines

UPF Response Time	Recommended `request_timeout_ms`	Total Wait Time
Fast (<100ms)	200-300ms	600-900ms (3 attempts)
Normal (100-300ms)	500ms (default)	1.5s (3 attempts)
Slow (300-500ms)	750-1000ms	2.25-3s (3 attempts)
Very slow (>500ms)	1500-2000ms	4.5-6s (3 attempts)

Recommendation: Set `request_timeout_ms` to at least **2× the expected UPF response time** to avoid retransmission-induced orphan sessions.

Example - Slow UPF

```
sxb: %{
  local_ip_address: "10.0.0.20",
  request_timeout_ms: 1000,  # 1 second per attempt
  request_attempts: 3         # Total: 3 seconds max
}
```

Diagnosing Timeout Issues

Symptoms of timeout too low:

- UPF reports more PFCP sessions than expected
- Orphan sessions accumulate in UPF over time

- Log messages: `"Session Establishment Request timed out"` followed by successful session

How to diagnose:

1. Check UPF session count via UPF API or management interface
2. Compare with PGW-C active session count
3. If UPF has more sessions, orphans exist from retransmissions

Resolution:

1. Increase `request_timeout_ms` to exceed UPF response time
2. Restart PGW-C to apply new configuration
3. Clear orphan sessions from UPF (manual cleanup or UPF restart)

Multiple PGW-U Peers

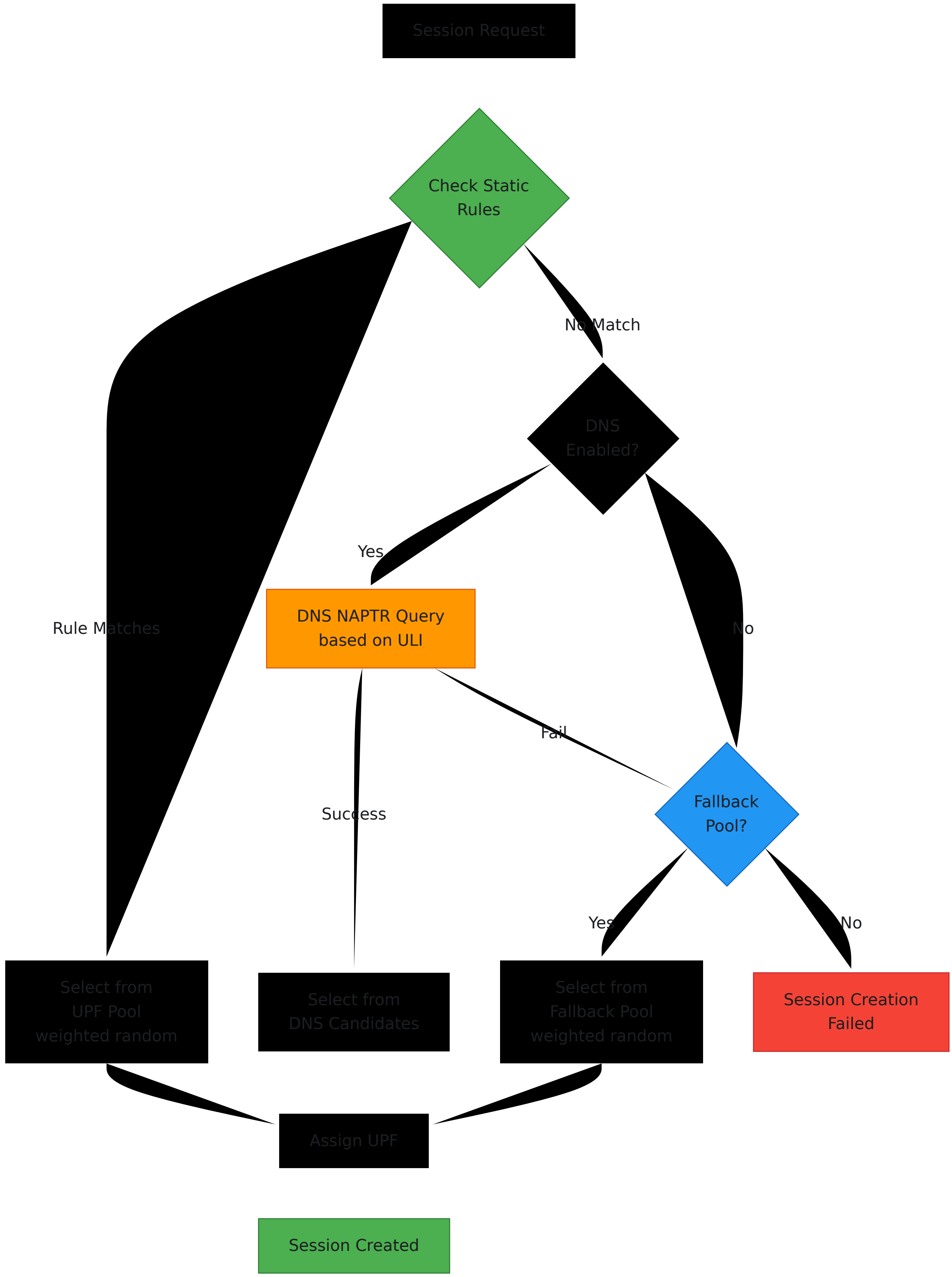
For load balancing or redundancy:

```
config :pgw_c,
  sxb: %{
    local_ip_address: "10.0.0.20"
  },
  upf_selection: %{
    fallback_pool: [
      %{remote_ip_address: "10.0.1.21", remote_port: 8805, weight: 50},  # 50% traffic
      %{remote_ip_address: "10.0.2.21", remote_port: 8805, weight: 50}   # 50% traffic
    ]
  }
# Both UPFs automatically registered with 5-second heartbeats
```

UPF Selection Configuration

PGW-C uses a **three-tier UPF selection system** with priority-based rules:

1. **Static Rules** (Highest Priority) - Match based on session attributes
2. **DNS-Based Selection** (Medium Priority) - Location-aware routing via DNS NAPTR queries
3. **Fallback Pool** (Lowest Priority) - Default UPF pool when no rules match



Complete UPF Selection Example

```
config :pgw_c,
  # PFCP Interface
  sxb: %{
    local_ip_address: "10.0.0.20"
  },

  # UPF Selection: All UPFs defined here are automatically registerec
  upf_selection: %{
    #
====================================================================
    # DNS-Based Selection (Location-Aware Routing)
    #
====================================================================
    # Queries DNS using User Location Information (ULI)
    # Provides dynamic UPF selection based on cell location
    dns_enabled: false,
    dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],
    dns_suffix: "epc.3gppnetwork.org",
    dns_timeout_ms: 5000,

    #
====================================================================
    # Static Selection Rules (Evaluated by Priority)
    #
====================================================================
    # Rules are checked from highest to lowest priority
    # First matching rule determines the UPF pool
    rules: [
      # Rule 1: IMS Traffic - Highest Priority
      %{
        name: "IMS Traffic",
        priority: 20,
        match_field: :apn,
        match_regex: "^ims",
        upf_pool: [
          %{remote_ip_address: "10.100.2.21", remote_port: 8805,
weight: 80},
          %{remote_ip_address: "10.100.2.22", remote_port: 8805,
weight: 20}
        ],
        # Optional: PCO overrides for this rule
        pco: %{
          p_cscf_ipv4_address_list: ["203.0.113.100",
```

```
"203.0.113.101"]
        }
      },

      # Rule 2: Enterprise APN - High Priority
      %{
        name: "Enterprise Traffic",
        priority: 15,
        match_field: :apn,
        match_regex: "^(enterprise|corporate)\.apn",
        upf_pool: [
          %{remote_ip_address: "10.100.3.21", remote_port: 8805,
weight: 100}
        ],
        pco: %{
          primary_dns_server_address: "192.168.1.10",
          secondary_dns_server_address: "192.168.1.11",
          ipv4_link_mtu_size: 1500
        }
      },

      # Rule 3: Roaming Subscribers - Medium Priority
      %{
        name: "Roaming Subscribers",
        priority: 10,
        match_field: :serving_network_plmn_id,
        match_regex: "^(310|311|312|313)",  # US networks
        upf_pool: [
          %{remote_ip_address: "10.100.4.21", remote_port: 8805,
weight: 100}
        ]
      },

      # Rule 4: Internet Traffic - Lower Priority
      %{
        name: "Internet Traffic",
        priority: 5,
        match_field: :apn,
        match_regex: "^internet",
        upf_pool: [
          %{remote_ip_address: "10.100.1.21", remote_port: 8805,
weight: 33},
          %{remote_ip_address: "10.100.1.22", remote_port: 8805,
weight: 33},
```

```
        %{remote_ip_address: "10.100.1.23", remote_port: 8805,
weight: 34}
      ]
    }
  ],

  #
==========================================================================
  # Fallback Pool (Last Resort)
  #
==========================================================================
  # Used when no rules match and DNS selection fails or is disabled
  fallback_pool: [
    %{remote_ip_address: "127.0.0.21", remote_port: 8805, weight:
100}
  ]
}
```

Supported Match Fields

Match Field	Description	Example Value
`:imsi`	International Mobile Subscriber Identity	`"310260123456789"`
`:apn`	Access Point Name	`"internet"`, `"ims"`
`:serving_network_plmn_id`	Serving network PLMN (MCC+MNC)	`"310260"` (test network)
`:sgw_ip_address`	SGW IP address (string format)	`"10.0.1.50"`
`:uli_tai_plmn_id`	Tracking Area PLMN ID	`"310260"`
`:uli_ecgi_plmn_id`	E-UTRAN Cell PLMN ID	`"310260"`

UPF Pool and Load Balancing

Each rule can specify a **UPF pool** with weighted random selection:

```
upf_pool: [
  %{remote_ip_address: "10.100.1.21", remote_port: 8805, weight: 50},
  %{remote_ip_address: "10.100.1.22", remote_port: 8805, weight: 30},
  %{remote_ip_address: "10.100.1.23", remote_port: 8805, weight: 20}
]
```

How Weighted Selection Works:

1. Calculate total weight: 50 + 30 + 20 = 100
2. Generate random number: 0.0 to 100.0
3. Select UPF based on cumulative weight ranges:
    - 0-50: UPF-1 (50% chance)
    - 50-80: UPF-2 (30% chance)
    - 80-100: UPF-3 (20% chance)

Use Cases:

- **Equal distribution:** All weights equal (33, 33, 34)
- **Primary/backup:** High weight primary (80), low weight backup (20)
- **Capacity-based:** Weight proportional to UPF capacity

PCO Overrides

Rules can override PCO (Protocol Configuration Options) values:

```
%{
  name: "IMS Traffic",
  match_field: :apn,
  match_regex: "^ims",
  upf_pool: [...],
  pco: %{
    # Override only specific fields
    p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"],
    # Other fields use defaults from main pco config
  }
}
```

Available PCO Override Fields:

- `primary_dns_server_address`
- `secondary_dns_server_address`
- `primary_nbns_server_address`
- `secondary_nbns_server_address`
- `p_cscf_ipv4_address_list`
- `ipv4_link_mtu_size`

DNS-Based Selection

When enabled, PGW-C performs DNS NAPTR queries based on User Location Information:

```
upf_selection: %{
  dns_enabled: true,
  dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],
  dns_suffix: "epc.3gppnetwork.org",
  dns_timeout_ms: 5000
}
```

Query Priority:

1. **ECGI** (E-UTRAN Cell Global Identifier) - Most specific
2. **TAI** (Tracking Area Identity) - Cell area
3. **RAI** (Routing Area Identity) - 3G/2G area

4. **SAI** (Service Area Identity) - 3G service area
5. **CGI** (Cell Global Identity) - 2G cell

Example DNS Query:

```
# For ECGI query:
eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org

# For TAI query:
tac-lb64.tac-hb00.tac.epc.mnc999.mcc999.epc.3gppnetwork.org
```

DNS Selection Process:

1. Try queries in priority order (ECGI first, then TAI, etc.)
2. If DNS returns candidates, use first result (dynamically registered if needed)
3. Select returned UPF
4. If no DNS match or DNS disabled, fall through to fallback pool

See DNS-based UPF Selection for detailed information.

DNS-based UPF Selection

Overview

DNS-based UPF selection provides **location-aware routing** by performing DNS NAPTR queries using User Location Information (ULI) from the UE's current cell.

3GPP Reference: TS 23.003 - DNS procedures for UPF discovery

Benefits:

- Automatic UPF selection based on geographic location
- No manual rule configuration per cell
- Dynamic adaptation to network topology changes

- Reduces backhaul by routing to nearest UPF

How It Works

Parse error on line 25: ... style PGWC fill:#4CAF50,stroke:#2E7 ----------------------^ Expecting 'SOLID_OPEN_ARROW', 'DOTTED_OPEN_ARROW', 'SOLID_ARROW', 'BIDIRECTIONAL_SOLID_ARROW', 'DOTTED_ARROW', 'BIDIRECTIONAL_DOTTED_ARROW', 'SOLID_CROSS', 'DOTTED_CROSS', 'SOLID_POINT', 'DOTTED_POINT', got 'TXT'

Try again

Configuration

```
config :pgw_c,
  upf_selection: %{
    # Enable DNS-based selection
    dns_enabled: true,

    # Query priority: try ECGI first, then TAI, then RAI, etc.
    dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],

    # DNS suffix for queries
    dns_suffix: "epc.3gppnetwork.org",

    # DNS query timeout
    dns_timeout_ms: 5000,

    # Static rules still take precedence over DNS
    rules: [...],

    # Fallback if DNS fails
    fallback_pool: [...]
  }
```

DNS Query Formats

DNS queries are built using User Location Information (ULI) from the GTP-C message:

1. ECGI (E-UTRAN Cell Global Identifier)

Most specific - LTE cell-level routing

Format:

```
eci-<HEX-ECI>.ecgi.epc.mnc<MNC>.mcc<MCC>.<dns_suffix>
```

Example:

```
# Cell ID: 0x1A2B3C (1,715,004 decimal)
# PLMN: MCC=999, MNC=999
eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org
```

When Used: LTE (4G) networks

2. TAI (Tracking Area Identity)

Cell area - Multiple cells in same tracking area

Format:

```
tac-lb<LB>.tac-hb<HB>.tac.epc.mnc<MNC>.mcc<MCC>.<dns_suffix>
```

Example:

```
# TAC: 0x0064 (100 decimal)
# Low byte: 0x64, High byte: 0x00
tac-lb64.tac-hb00.tac.epc.mnc999.mcc999.epc.3gppnetwork.org
```

When Used: LTE (4G) tracking areas

3. RAI (Routing Area Identity)

3G/2G routing area

Format:

```
rac<RAC>.lac-lb<LB>.lac-hb<HB>.lac.rai.mnc<MNC>.mcc<MCC>.
<dns_suffix>
```

Example:

```
# RAC: 0x0A (10 decimal)
# LAC: 0x1234 (4660 decimal)
rac0a.lac-lb34.lac-hb12.lac.rai.mnc999.mcc999.epc.3gppnetwork.org
```

When Used: 3G/2G UMTS/GPRS networks

4. SAI (Service Area Identity)

3G service area

Format:

```
sac<SAC>.lac-lb<LB>.lac-hb<HB>.lac.sai.mnc<MNC>.mcc<MCC>.
<dns_suffix>
```

Example:

```
# SAC: 0x0001
# LAC: 0x1234
sac0001.lac-lb34.lac-
hb12.lac.sai.mnc999.mcc999.epc.3gppnetwork.org
```

When Used: 3G UMTS service areas

5. CGI (Cell Global Identity)

2G cell-level

Format:

```
ci<CI>.lac-lb<LB>.lac-hb<HB>.lac.cgi.mnc<MNC>.mcc<MCC>.
<dns_suffix>
```

Example:

```
# CI: 0x5678
# LAC: 0x1234
ci5678.lac-lb34.lac-hb12.lac.cgi.mnc999.mcc999.epc.3gppnetwork.org
```

When Used: 2G GSM cells

DNS Response Processing

NAPTR Record Format:

DNS returns NAPTR records pointing to UPF IP addresses:

```
eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org.
  IN NAPTR 10 50 "a" "x-3gpp-upf:x-s5-gtp:x-s8-gtp" ""
upf1.epc.mnc999.mcc999.3gppnetwork.org.

upf1.epc.mnc999.mcc999.3gppnetwork.org.
  IN A 10.100.1.21
```

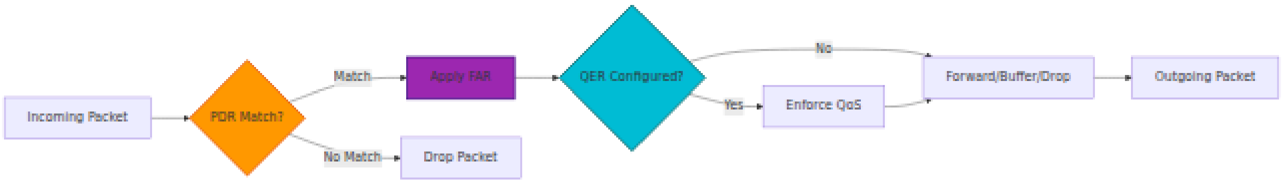
PGW-C Processing:

1. Parse NAPTR records to extract UPF IP addresses
2. Select first candidate from DNS response
3. Dynamically register if not already configured (or implement load-based selection)

Example:

```
DNS returns: [10.100.1.21, 10.100.5.99, 10.200.3.50]

Selected:    10.100.1.21 (first candidate)
Action:      Register dynamically if not in upf_selection
```

Selection Priority Example



Use Cases

1. Geographic Load Balancing

Scenario: Operator has UPFs in multiple cities

DNS Configuration:

```
# Chicago cell
eci-aaa.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → UPF-Chicago
(10.1.1.21)

# New York cell
eci-bbb.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → UPF-NewYork
(10.2.1.21)

# Los Angeles cell
eci-ccc.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → UPF-
LosAngeles (10.3.1.21)
```

Benefit: Users automatically routed to nearest UPF, reducing latency and backhaul

2. Edge Computing

Scenario: MEC (Multi-access Edge Computing) UPFs deployed at cell sites

DNS Configuration:

```
# Each cell points to local edge UPF
eci-*.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → Local Edge UPF
```

Benefit: Ultra-low latency for edge applications

3. Dynamic Network Topology

Scenario: UPF addresses change due to upgrades or maintenance

Benefit: Update DNS records without changing PGW-C configuration

Troubleshooting DNS Selection

DNS Query Failures

Symptoms:

- Log: "DNS UPF selection failed: :nxdomain"
- Sessions fall back to fallback pool

Possible Causes:

1. DNS server not configured correctly
2. DNS zone not populated for cell IDs
3. ULI not present in GTP-C message

Resolution:

```
# Test DNS query manually
dig eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org NAPTR

# Check PGW-C logs for DNS queries
grep "DNS UPF selection: querying" /var/log/pgw_c.log

# Verify ULI present in session
# Check "uli" field in session state
```

DNS Returns Unknown UPF

Behavior:

- DNS returns a candidate UPF not in `upf_selection`
- System automatically attempts dynamic registration
- If PFCP association succeeds, UPF is used for the session
- If PFCP association fails, falls back to fallback pool

Example:

```
DNS returns: [10.99.1.50]
upf_selection: [10.100.1.21, 10.100.1.22]

Action: Dynamically register 10.99.1.50
  - Send PFCP Association Setup
  - If success: Use for session
  - If timeout: Fall back to fallback pool
```

Resolution Options:

1. Pre-configure in `upf_selection` for immediate monitoring:

```
upf_selection: %{
  fallback_pool: [
    %{remote_ip_address: "10.99.1.50", remote_port: 8805, weight: 100}
  ]
}
```

2. Update DNS to return pre-configured UPF IPs

3. Allow dynamic registration (recommended for MEC/edge scenarios)

Query Timeout

Symptoms:

- Log: "DNS UPF selection: query timeout"

- Sessions take longer to establish

Resolution:

```
upf_selection: %{
  dns_timeout_ms: 10000  # Increase timeout to 10 seconds
}
```

Monitoring DNS Selection

Metrics:

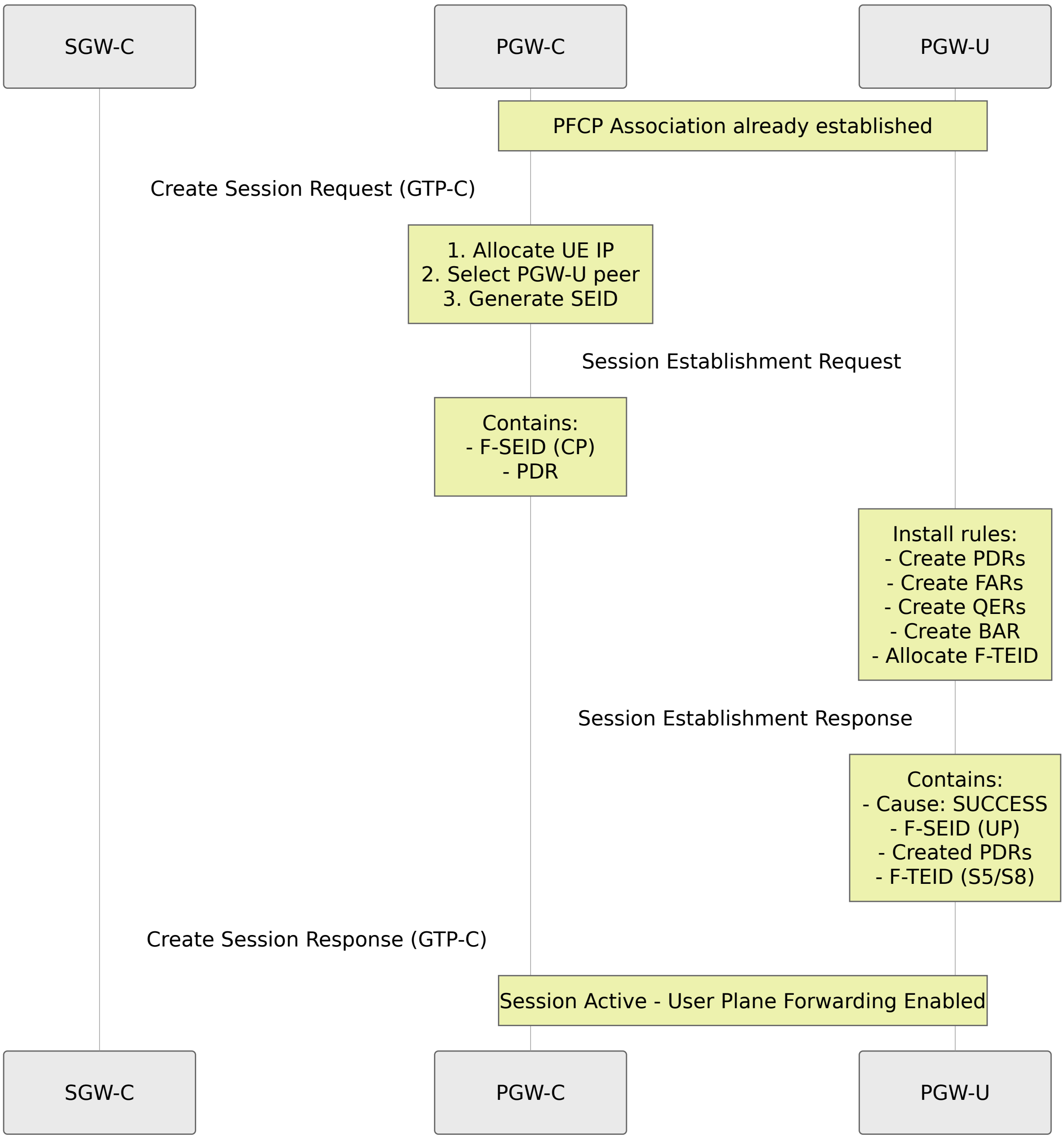
```
# DNS query success rate
rate(upf_selection_dns_success_total[5m]) /
rate(upf_selection_dns_attempts_total[5m])

# DNS query latency
histogram_quantile(0.95,
rate(upf_selection_dns_duration_seconds_bucket[5m]))

# Fallback usage (indicates DNS issues)
rate(upf_selection_fallback_used_total[5m])
```

Logs:

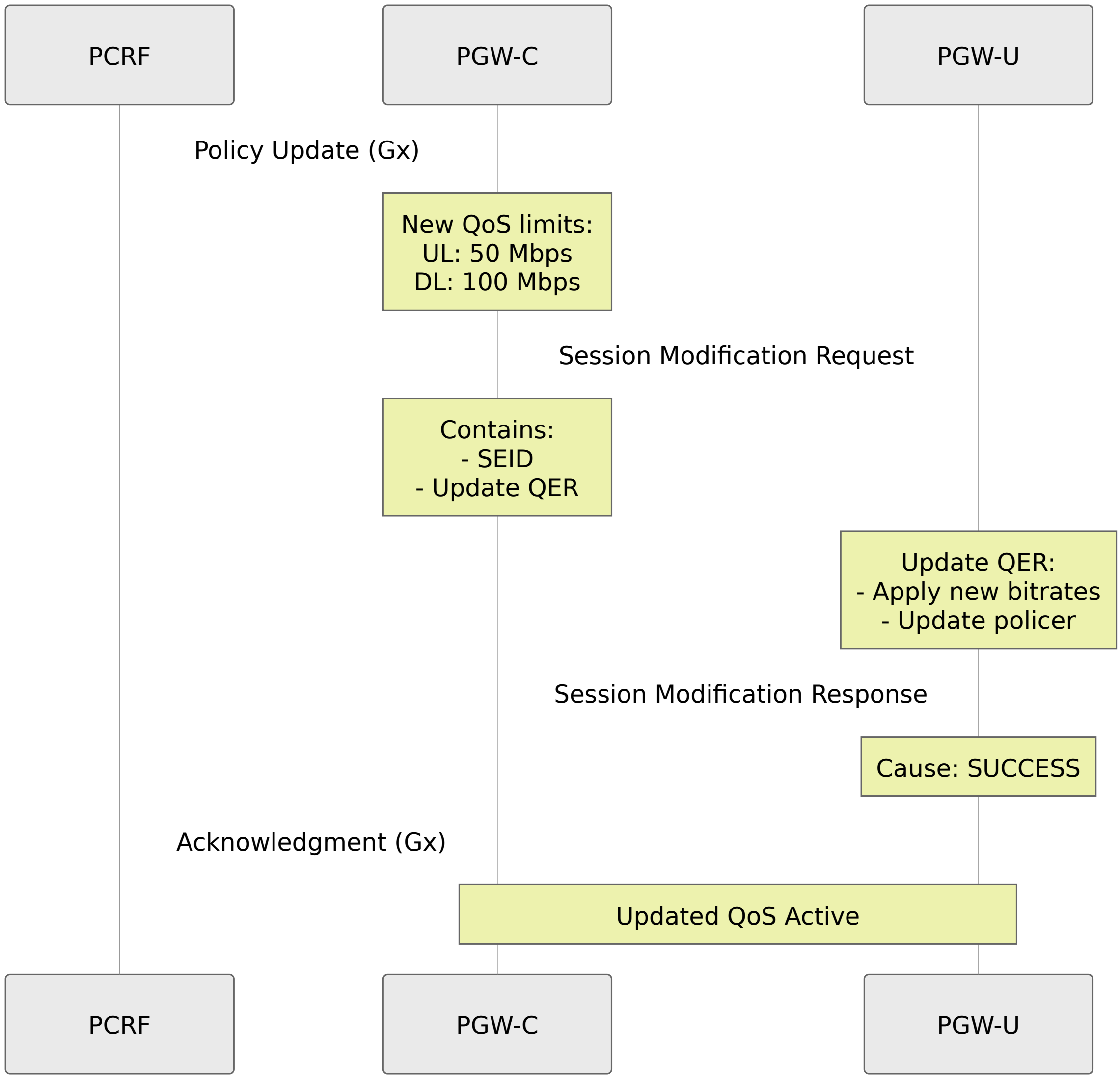
```
[debug] DNS UPF selection: querying eci-
1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org
[debug] DNS UPF selection: got 2 candidates from DNS
[info]  DNS UPF selection: selected 10.100.1.21
```

Message Flows

Complete Session Establishment Flow



Session Modification Flow



Heartbeat Failure Recovery

Troubleshooting

Common Issues

1. Association Setup Fails

Symptoms:

- Log message: "PFCP Association Setup failed"
- No response to Association Setup Request

Possible Causes:

- PGW-U not reachable (network issue)
- PGW-U not running
- Firewall blocking UDP port 8805
- Incorrect `remote_ip_address` in configuration

Resolution:

```
# Test connectivity
ping <pgw_u_ip_address>

# Test UDP port
nc -u -v <pgw_u_ip_address> 8805

# Check firewall
iptables -L -n | grep 8805
```

2. Heartbeats Failing

Symptoms:

- Log: "Consecutive heartbeat failures: 3"
- Association marked as down

Possible Causes:

- Network latency or packet loss
- PGW-U overloaded
- Heartbeat interval too aggressive

Resolution:

The heartbeat period is fixed at 5 seconds with a failure threshold of 3 consecutive missed heartbeats.

3. Session Establishment Fails

Symptoms:

- Create Session Response with error cause
- Log: "PFCP Session Establishment failed"

Possible Causes:

- No PGW-U peers available
- PGW-U resource exhaustion
- Invalid rule configuration

Check:

1. Verify at least one peer has `is_associated = true`
2. Check PGW-U logs for errors
3. Verify SEID uniqueness

4. Duplicate SEID Errors

Symptoms:

- Session Establishment Response: Cause "Session context not found"

Cause:

- SEID collision (very rare)
- PGW-U restart without PGW-C awareness

Resolution:

- Restart PFCP association (triggers new recovery timestamp)
- PGW-C will detect PGW-U restart and clean up old sessions

Monitoring PFCP Health

Metrics to Monitor:

```
# PFCP peer association status
pfcp_peer_associated{peer="PGW-U Primary"} 1

# Active PFCP sessions
seid_registry_count 150

# PFCP message rates
rate(sxb_inbound_messages_total[5m])

# PFCP errors
rate(sxb_inbound_errors_total[5m])

# Heartbeat failures
pfcp_consecutive_heartbeat_failures{peer="PGW-U Primary"} 0
```

Alert Examples:

```yaml
# Alert on association down
- alert: PFCPAssociationDown
  expr: pfcp_peer_associated == 0
  for: 1m
  annotations:
    summary: "PFCP peer {{ $labels.peer }} is down"

# Alert on high session establishment failures
- alert: PFCPSessionEstablishmentFailureHigh
  expr:
rate(sxb_inbound_errors_total{message_type="session_establishment_res
[5m]) > 0.1
  for: 5m
  annotations:
    summary: "High PFCP session establishment failure rate"
```

PFCP Monitoring

PFCP/Sxb association and UPF peer status are monitored through the OAM REST API. The API is served over HTTPS/TLS on port `8443` with all routes under the `/api` prefix. Interactive documentation (Swagger UI) is available at `https://<omnipgw-ip>:8443/api/docs`.

UPF / PFCP Peer Status

```
GET /api/upf             # all UPF / PFCP peers + status
GET /api/upf/<ip>        # one UPF by IP
```

Purpose: Monitor PFCP association status with all configured PGW-U peers.

Example:

```
curl -k https://<omnipgw-ip>:8443/api/upf
curl -k https://<omnipgw-ip>:8443/api/upf/10.0.0.1
```

Per-Peer Information returned for each configured PGW-U peer:

Field	Description
Peer Name	Friendly name from configuration
IP Address	Remote PGW-U IP
Association Status	Associated or Not Associated
Node ID	PFCP Node identifier
Recovery Timestamp	Last restart time of peer
Heartbeat Period	Configured heartbeat interval
Consecutive Missed Heartbeats	Current failure count
UP Function Features	Capabilities advertised by PGW-U

`GET /api/upf/<ip>` returns the full state for a single peer, including the UP function features bitmap and association timestamps.

PFCP Session Inspection

Active PFCP sessions are visible via the per-subscriber session detail:

```
GET /api/sessions/<imsi>
```

The session detail includes the associated PGW-U peer and the programmed forwarding rules (PDRs, FARs, QERs, BARs) and F-SEIDs for the session.

Operational Use Cases

Monitor PFCP Association Health:

```
1. GET /api/upf
2. Verify all peers show "Associated"
3. Check missed heartbeat count = 0
4. If a peer shows "Not Associated":
   - Check peer IP reachability
   - Verify peer is running
   - Check firewall (UDP 8805)
```

Troubleshoot Session Establishment Failures:

```
1. User session fails to establish
2. GET /api/sessions/<imsi> - session exists?
3. Inspect the session's PFCP state - PFCP session created?
4. If no PFCP session:
   - GET /api/upf - is any peer associated?
   - Check logs for PFCP errors
5. If PFCP session exists:
   - Inspect PDRs/FARs to verify rules programmed
   - Issue is likely downstream (PGW-U or network)
```

Verify Peer Load Distribution:

```
1. With multiple PGW-U peers configured
2. GET /api/upf to review each peer's state
3. Verify sessions distributed across peers
4. Identify if one peer has disproportionate load
```

Detect Peer Failures:

- `GET /api/upf` reports the "Not Associated" state per peer
- Missed heartbeat counter shows degradation before total failure
- Set up monitoring alerts based on the OAM API data

Advantages:

- **Scriptable monitoring** - Poll `GET /api/upf` from tooling or alerting
- **Explicit status** - Associated / not associated per peer
- **Peer health trends** - Missed heartbeat count shows early warning

- **Session-level inspection** - See exact PDRs/FARs/QERs programmed

---

Related Documentation

Configuration

- **Configuration Guide** - UPF selection, health monitoring, PFCP configuration
- **Session Management** - PDN session lifecycle, bearer establishment

Charging and Monitoring

- **Diameter Gx Interface** - PCC rules that drive PFCP QoS enforcement
- **Diameter Gy Interface** - Online charging quota management via URRs
- **Data CDR Format** - CDR generation from PFCP usage reports
- **Monitoring Guide** - PFCP metrics, session tracking, UPF health alerts

Network Interfaces

- **S5/S8 Interface** - Control plane bearer management
- **UE IP Allocation** - UE address assignment via PFCP

Back to Operations Guide

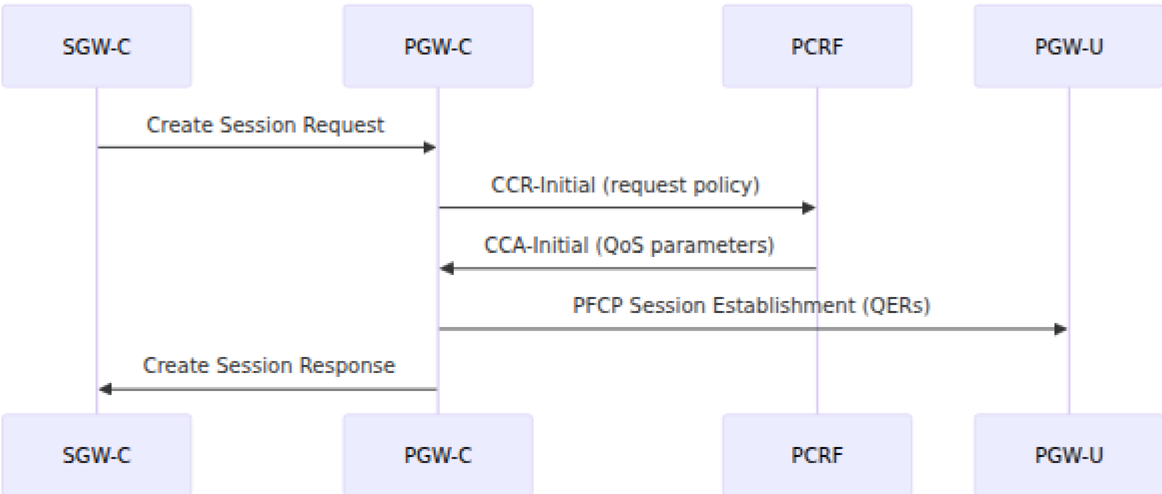
QoS & Bearer Management

Overview

The PGW-C implements a policy-driven bearer and QoS management system that coordinates three key interfaces:

- **Gx (Diameter)** - Receives policy decisions and QoS parameters from PCRF
- **S5/S8 (GTP-C)** - Manages bearer contexts with SGW-C
- **Sxb (PFCP)** - Programs QoS enforcement rules into PGW-U

Architecture Flow



Key Concepts

- **Session**: Contains UE information, bearer map, PDR/FAR/QER/BAR maps, and AMBR
- **Bearer Context**: Links EBI (EPS Bearer ID) to specific PDRs, FARs, and QERs
- **QER (QoS Enforcement Rule)**: Enforces MBR/GBR limits and gate status in the user plane

- **Default Bearer**: Always created with PDN session, provides basic connectivity
- **Dedicated Bearer**: Created dynamically based on PCRF policy, provides specific QoS guarantees

Configuration

Important: Dynamic QoS Policy

All QoS parameters are dynamically received from the PCRF via Diameter Gx interface and defined in the PCRF (See OmniHSS for more info).

Operators configure the **PCRF connection** in `config/runtime.exs`:

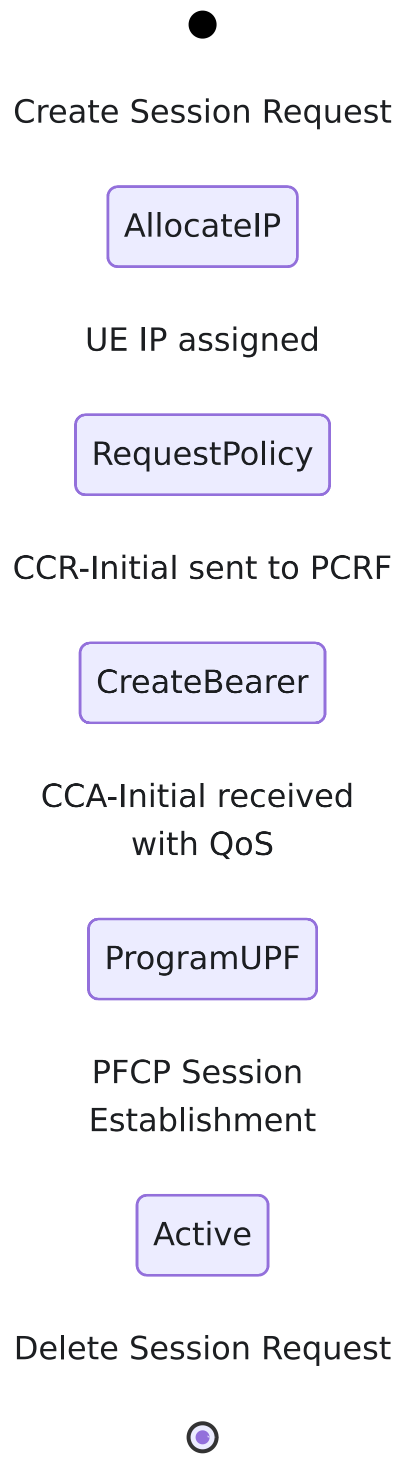
```
config :pgw_c,
  diameter: %{
    listen_ip: "0.0.0.0",
    host: "omni-pgw_c.epc.mnc999.mcc999.3gppnetwork.org",
    realm: "epc.mnc999.mcc999.3gppnetwork.org",
    peer_list: [
      %{
        host: "pcrf.epc.mnc999.mcc999.3gppnetwork.org",
        realm: "epc.mnc999.mcc999.3gppnetwork.org",
        ip: "192.168.1.100",
        initiate_connection: true
      }
    ]
  }
```

QoS policies, charging rules, and bandwidth limits are configured on the PCRF, not in PGW-C configuration files.

Bearer Lifecycle

Default Bearer Creation

The default bearer is created during PDN session establishment:



Workflow:

1. SGW-C sends Create Session Request
2. PGW-C allocates UE IP address from configured pool
3. PGW-C sends CCR-Initial to PCRF with IMSI, APN, IP address
4. PCRF responds with CCA-Initial containing QoS parameters:
   - Default-EPS-Bearer-QoS (QCI, ARP)
   - QoS-Information (AMBR adjustments)
5. PGW-C creates bearer context with:
   - Fixed IDs: Downlink PDR=1, Uplink PDR=2, Downlink FAR=1, Uplink FAR=2, QER=1, BAR=1
   - QER programmed with MBR from bearer QoS
6. PGW-C sends PFCP Session Establishment Request to PGW-U
7. PGW-C sends Create Session Response to SGW-C

Default bearer characteristics:

- Always exists for the lifetime of the PDN session
- Typically uses QCI 5 or QCI 9 (non-GBR)
- EBI tracked in session state
- Cannot be deleted independently (deleting it terminates the session)

Dedicated Bearer Creation

Dedicated bearers are created dynamically based on PCRF policy:

Trigger: Re-Auth Request (RAR) from PCRF with Charging-Rule-Install

Workflow:

1. PCRF sends RAR with Charging-Rule-Definition containing:
   - Charging-Rule-Name (policy rule identifier)
   - Flow-Information (packet filters)
   - QoS-Information (QCI, MBR, GBR, ARP)
   - Precedence (rule matching priority)
2. PGW-C translates dynamic rule to PFCP entities:
   - Each Flow-Information entry → new PDR with SDF Filter
   - QoS-Information → new QER with MBR/GBR enforcement

    - Flow-Description → IP 5-tuple matching rules
3. PGW-C sends PFCP Session Modification Request to add PDRs/FARs/QERs
4. PGW-C initiates Create Bearer Request to SGW-C
5. SGW-C responds with Create Bearer Response confirming establishment

Example Charging-Rule-Definition:

```
Charging-Rule-Name: "video_streaming"
Flow-Information:
  - Flow-Description: "permit in ip from any to 10.0.0.1 5000-6000"
    Flow-Direction: 1 (downlink)
QoS-Information:
  QoS-Class-Identifier: 7
  Max-Requested-Bandwidth-UL: 5000000 (5 Mbps)
  Max-Requested-Bandwidth-DL: 10000000 (10 Mbps)
  Guaranteed-Bitrate-UL: 1000000 (1 Mbps)
  Guaranteed-Bitrate-DL: 2000000 (2 Mbps)
Precedence: 100
Flow-Status: 2 (ENABLED)
```

Bearer Modification

Bearer QoS can be modified via:

- **Gx RAR** with updated Charging-Rule-Definition
- **PFCP Session Modification** to update existing QERs (change bitrates), FARs (change forwarding), or PDRs (change packet filters)

Bearer Deletion

Triggers:

- **Delete Session Request** (SGW-initiated) - Deletes default bearer and terminates session
- **Re-Auth Request with Charging-Rule-Remove** (PCRF-initiated) - Deletes dedicated bearer

Workflow:

1. Remove bearer from session state
2. Remove associated PDRs/FARs/QERs
3. Send Delete Bearer Request to SGW-C (if PCRF-initiated)
4. Send PFCP Session Modification (remove rules) or Session Deletion (if default bearer)

QoS Parameters

QCI (QoS Class Identifier)

Source: PCRF via Gx `QoS-Class-Identifier` AVP

Standard Values:

- **QCI 1**: Conversational Voice (GBR, 100ms delay budget)
- **QCI 2**: Conversational Video (GBR, 150ms delay budget)
- **QCI 3**: Real Time Gaming (GBR, 50ms delay budget)
- **QCI 4**: Non-Conversational Video (GBR, 300ms delay budget)
- **QCI 5**: IMS Signaling (non-GBR, 100ms delay budget) - **Default for default bearer**
- **QCI 6**: Video (TCP-based), Live Streaming (non-GBR, 300ms delay budget)
- **QCI 7**: Voice, Interactive Gaming (non-GBR, 100ms delay budget)
- **QCI 8**: Video (TCP-based), e.g., YouTube (non-GBR, 300ms delay budget)
- **QCI 9**: Default Internet (non-GBR, 300ms delay budget)

Operator Note:

- QCI is received from PCRF and signaled to SGW-C in Bearer-Level-QoS IE
- PGW-C does not directly enforce QCI behavior - actual enforcement is via MBR/GBR in QERs
- Lower QCI values typically indicate higher priority
- QCI determines packet forwarding treatment and scheduling priority

ARP (Allocation and Retention Priority)

Source: PCRF via `Allocation-Retention-Priority` grouped AVP

Components:

- **Priority-Level**: 1 (highest priority) to 15 (lowest priority)
- **Pre-emption-Capability**: Can this bearer pre-empt lower-priority bearers?
  - 0 = ENABLED (can pre-empt others)
  - 1 = DISABLED (cannot pre-empt)
- **Pre-emption-Vulnerability**: Can this bearer be pre-empted by higher-priority bearers?
  - 0 = ENABLED (can be pre-empted)
  - 1 = DISABLED (cannot be pre-empted)

Default Values:

- Priority-Level: 1
- Pre-emption-Capability: ENABLED (0)
- Pre-emption-Vulnerability: DISABLED (1)

Operator Note:

- ARP is signaled to SGW-C and ultimately to eNodeB
- **Not enforced by PGW-C** - enforcement is typically at eNodeB during radio admission control
- Used during network congestion to determine which bearers to admit or drop
- Critical for emergency services (priority-level 1) and high-value services

MBR (Maximum Bit Rate)

Source: PCRF via `Max-Requested-Bandwidth-UL` and `Max-Requested-Bandwidth-DL` AVPs

Format: Bytes per second (converted to kbps internally: `bytes / 1000`)

Applied to: All bearers (default and dedicated)

How it works:

- PGW-C creates QER with `mbr: %Bitrate{ul: kbps_ul, dl: kbps_dl}`
- QER sent to PGW-U via PFCP
- **PGW-U enforces rate limiting** (traffic policing)
- Excess traffic above MBR is dropped

Example:

```
Max-Requested-Bandwidth-UL: 5000000 (5 Mbps)
Max-Requested-Bandwidth-DL: 10000000 (10 Mbps)

→ QER created with mbr: {ul: 5000, dl: 10000} kbps
→ PGW-U drops uplink packets exceeding 5 Mbps
→ PGW-U drops downlink packets exceeding 10 Mbps
```

GBR (Guaranteed Bit Rate)

Source: PCRF via `Guaranteed-Bitrate-UL` and `Guaranteed-Bitrate-DL` AVPs

Format: Bytes per second (converted to kbps)

Applied to: Dedicated bearers only (GBR bearers)

How it works:

- If GBR is specified in Charging-Rule-Definition, bearer is **GBR type**
- PGW-U enforces minimum bitrate guarantee via QER
- Requires proper scheduling at eNodeB to reserve radio resources
- GBR bearers have admission control - can be rejected if resources unavailable

Example:

```
Guaranteed-Bitrate-UL: 1000000 (1 Mbps)
Guaranteed-Bitrate-DL: 2000000 (2 Mbps)

→ QER created with gbr: {ul: 1000, dl: 2000} kbps
→ Network guarantees at least 1 Mbps uplink and 2 Mbps downlink
→ Used for VoIP, video calls, live streaming
```

Operator Note:

- GBR requires sufficient network capacity planning
- Oversubscribing GBR resources leads to admission failures
- Monitor GBR usage via session counts and bearer metrics

AMBR (Aggregate Maximum Bit Rate)

Source: PCRF via `APN-Aggregate-Max-Bitrate-UL` and `APN-Aggregate-Max-Bitrate-DL` AVPs

Scope: Applies to **all non-GBR bearers** for the APN (not per-bearer)

How it works:

- AMBR is an aggregate limit across all non-GBR bearers in a session
- Sent to SGW-C in Create Session Response
- Enforcement typically at eNodeB/SGW
- PGW-C stores AMBR in session state and signals it to SGW-C

Example:

```
APN-Aggregate-Max-Bitrate-UL: 50000000 (50 Mbps)
APN-Aggregate-Max-Bitrate-DL: 100000000 (100 Mbps)

→ All non-GBR bearers combined cannot exceed 50 Mbps uplink / 100 Mbps downlink
→ Individual bearers limited by their own MBR
→ AMBR provides additional overall cap per UE/APN
```

Operator Note:

- Set via subscriber profile in HSS/PCRF
- Used to enforce subscription tiers (e.g., 10 Mbps plan vs 100 Mbps plan)
- Does not affect GBR bearers

Flow Status and Gating

Flow Status (Gx) to Gate Status (PFCP) Mapping

The PCRF controls whether traffic is allowed via the `Flow-Status` AVP in Charging-Rule-Definition:

Flow-Status (Gx)	Gate-Status (PFCP QER)	Meaning
0 = ENABLED-UPLINK	ul: OPEN, dl: CLOSED	Only uplink traffic allowed
1 = ENABLED-DOWNLINK	ul: CLOSED, dl: OPEN	Only downlink traffic allowed
2 = ENABLED	ul: OPEN, dl: OPEN	Both directions allowed
3 = DISABLED	ul: CLOSED, dl: CLOSED	No traffic allowed
4 = REMOVED	ul: CLOSED, dl: CLOSED	Bearer being deleted

Use cases:

- **DISABLED**: Used for parked services or credit exhaustion (packets dropped but bearer retained)
- **ENABLED-UPLINK**: Unusual, but could be used for upload-only services
- **ENABLED-DOWNLINK**: Download-only services or credit-limited scenarios

- **ENABLED**: Normal operation

Monitoring & Observability

Prometheus Metrics

Session-level metrics:

```
session_registry_count       # Active bearers (IMSI, EBI pairs)
address_registry_count       # Allocated UE IPs
charging_id_registry_count    # Active charging sessions
```

Gx interface metrics:

```
gx_inbound_messages_total{message_type="gx_RAR"}     # Policy
updates from PCRF
gx_outbound_messages_total{message_type="gx_CCR"}    # Policy
requests to PCRF
gx_outbound_transaction_duration_bucket              # Latency to
PCRF
```

PFCP interface metrics:

```
sxb_outbound_messages_total{message_type="pfcp_session_establishment_
sxb_outbound_messages_total{message_type="pfcp_session_modification_r
sxb_outbound_transaction_duration_bucket
```

Bearer creation metrics:

```
s5s8_inbound_messages_total{message_type="create_session_request"}
# Default bearers
s5s8_outbound_messages_total{message_type="create_bearer_request"}
# Dedicated bearers
```

Monitoring via the OAM API

Bearer and QoS state is inspected through the OAM REST API (HTTPS/TLS on port `8443`, routes under the `/api` prefix). Interactive documentation (Swagger UI) is available at `https://<omnipgw-ip>:8443/api/docs`.

Bearer and QoS information is exposed through the per-subscriber session detail:

```
GET /api/sessions/<imsi>       # session detail incl. bearers,
QCI/5QI, charging rules
```

Example:

```
curl -k https://<omnipgw-ip>:8443/api/sessions/999000123456789
```

The session detail includes the active bearers for the session, their QoS parameters (QCI/5QI, MBR, GBR, AMBR), the installed charging rules, and the programmed QER map (gate status and bit rates). There is no dedicated bearers endpoint; bearer state is always read from the session it belongs to.

Key Log Messages

```
[debug] Sending Credit Control Request: ...          # CCR to PCRF
[debug] Handling Credit Control Answer: ...          # CCA from
PCRF (contains QoS)
[debug] Handling Re-Auth Request                     # RAR from
PCRF (policy change)
[debug] Sending Session Establishment Request        # PFCP to
PGW-U (program QERs)
[debug] Sending Session Modification Request         # PFCP to
PGW-U (update QERs)
```

Operational Tasks

Verify QoS Applied to Session

1. `GET /api/sessions/<imsi>` (e.g., `curl -k https://<omnipgw-ip>:8443/api/sessions/999000123456789`)
2. Locate the session for the subscriber in the response
3. Inspect the session detail
4. Check the **qer_map** section:

   ```
   qer_id: 1
   gate_status: {ul: OPEN, dl: OPEN}
   mbr: {ul: 50000, dl: 100000}  # kbps
   gbr: {ul: 10000, dl: 20000}   # kbps (or nil for non-GBR)
   ```

5. Verify values match expected PCRF policy

Troubleshoot Missing QoS

Symptom: Session created but QoS not applied

Steps:

1. **Check PCRF connectivity:**

   - Verify PCRF peer status = "connected" (see the Diameter/Gx metrics and peer state)
   - If disconnected, check network connectivity and Diameter configuration

2. **Verify CCR/CCA exchange:**

   - Search the OmniPGW logs for "Credit Control Answer"
   - Verify `QoS-Information` AVP present in CCA log
   - Check for errors in CCA (Result-Code should be 2001 = SUCCESS)

3. **Verify PFCP programming:**

   - Search logs for "PFCP Session Establishment Request"

- Verify QER included in message
- Check PGW-U logs for PFCP processing errors

4. **Check PCRF policy configuration:**

   - Verify subscriber profile in PCRF
   - Confirm APN-specific policy rules exist
   - Check PCRF logs for policy evaluation errors

Monitor Bearer Creation Rate

Prometheus queries:

```
# Default bearer creation rate (sessions/second)
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])

# Dedicated bearer creation rate
rate(s5s8_outbound_messages_total{message_type="create_bearer_request
[5m])

# Policy update rate from PCRF
rate(gx_inbound_messages_total{message_type="gx_RAR"}[5m])
```

Capacity Planning

Key metrics to monitor:

```
# UE IP address utilization (percentage)
(address_registry_count / <configured_pool_size>) * 100

# Active bearer count
session_registry_count

# PCRF query latency (P95)
histogram_quantile(0.95, gx_outbound_transaction_duration_bucket)
```

Capacity limits:

- Address pool size: configured in `config/runtime.exs` under `ue.subnet_map`
- TEID space: 32-bit (4 billion unique identifiers, auto-managed)
- Concurrent sessions: typically limited by address pool size

Planning guidelines:

- Monitor IP address utilization - scale pool before exceeding 80%
- Monitor PCRF latency - high latency impacts session setup time
- Monitor dedicated bearer creation rate - indicates policy complexity

Related Documentation

- Session Management - PDN session lifecycle
- Diameter Gx Interface - PCRF policy protocol details
- PFCP Interface - User plane programming
- Configuration Guide - System configuration
- Monitoring Guide - Metrics and observability

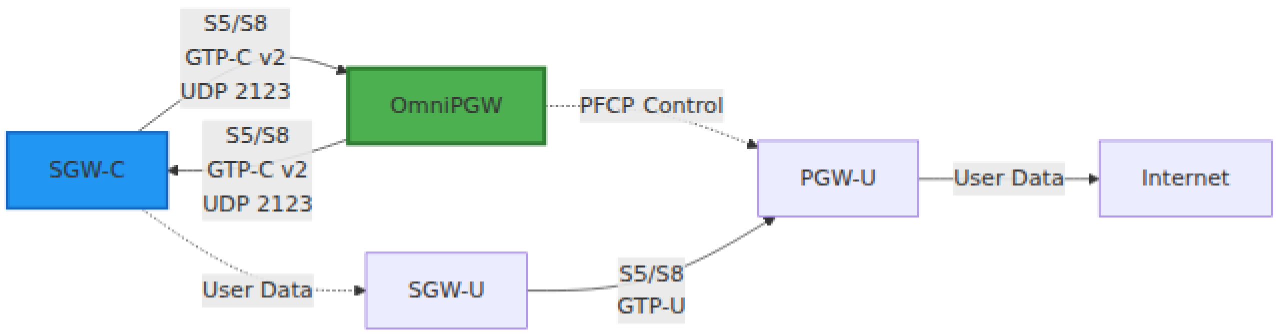
S5/S8 Interface Documentation

GTP-C Communication with SGW-C

OmniPGW by Omnitouch Network Services

Overview

The **S5/S8 interface** connects OmniPGW to the SGW-C (Serving Gateway Control plane) using the **GTP-C v2** (GPRS Tunnelling Protocol - Control plane) protocol. This interface handles session management signaling between the gateways.



Protocol Details

GTP-C Version 2

- **Protocol:** GTP-C v2 (3GPP TS 29.274)
- **Transport:** UDP
- **Port:** 2123 (standard)
- **Interface Type:** Control Plane

TEID (Tunnel Endpoint Identifier)

Each session has a unique **TEID** for routing messages:

- **Local TEID** - Allocated by OmniPGW for incoming messages
- **Remote TEID** - Allocated by SGW-C for outgoing messages

```
Message Flow:
  SGW-C → OmniPGW: Destination TEID = OmniPGW's Local TEID
  OmniPGW → SGW-C: Destination TEID = SGW-C's Remote TEID
```

Configuration

Basic Configuration

```
# config/runtime.exs
config :pgw_c,
  s5s8: %{
    # Local IPv4 address for S5/S8 interface
    local_ipv4_address: "10.0.0.20",

    # Optional: Local IPv6 address
    local_ipv6_address: nil,

    # Optional: Override default port
    local_port: 2123,

    # GTP-C request timeout in milliseconds (default: 500ms)
    # Timeout per attempt when waiting for GTP-C responses (Create
Bearer, Delete Bearer, etc.)
    request_timeout_ms: 500,

    # Number of retry attempts for GTP-C requests (default: 3)
    # Total maximum wait time = request_timeout_ms *
request_attempts
    # Example: 500ms * 3 attempts = 1500ms (1.5 seconds) total
    request_attempts: 3
  }
```

Timeout Configuration

The S5/S8 interface uses configurable timeouts for GTP-C request/response transactions.

Parameters:

- **`request_timeout_ms`** - Timeout in milliseconds per retry attempt (default: 500ms)
- **`request_attempts`** - Number of retry attempts before giving up (default: 3)

Total Wait Time: `request_timeout_ms × request_attempts`

Default behavior: 500ms × 3 attempts = **1.5 seconds total maximum wait**

Tuning Guidelines:

Network Latency	Recommended `request_timeout_ms`	Total Wait Time
Low latency (<50ms)	200-300ms	600-900ms (3 attempts)
Normal (50-150ms)	500ms (default)	1.5s (3 attempts)
High latency (>150ms)	1000-2000ms	3-6s (3 attempts)
Unstable/satellite	2000-3000ms	6-9s (3 attempts)

Example - High Latency Network:

```
s5s8: %{
  local_ipv4_address: "10.0.0.20",
  request_timeout_ms: 1500,  # 1.5 seconds per attempt
  request_attempts: 3         # Total: 4.5 seconds max
}
```

When timeout occurs:

- OmniPGW logs error: `"Create Bearer Request timed out"`
- Returns error to PCRF (Diameter Result-Code: 5012 UNABLE_TO_COMPLY)
- Bearer remains in early storage for cleanup via Charging-Rule-Remove

Network Requirements

Firewall Rules:

```
# Allow GTP-C from SGW-C network
iptables -A INPUT -p udp --dport 2123 -s <sgw_network>/24 -j
ACCEPT

# Allow outbound GTP-C to SGW-C
iptables -A OUTPUT -p udp --dport 2123 -d <sgw_network>/24 -j
ACCEPT
```

Routing:

```
# Ensure route to SGW-C network
ip route add <sgw_network>/24 via <gateway_ip> dev eth0
```

Message Types

The S5/S8 interface handles GTP-C signaling for PDN session management. For detailed session lifecycle and state management, see Session Management Guide.

Session Management

Create Session Request

Direction: SGW-C → OmniPGW

Purpose: Establish a new PDN connection

Key IEs (Information Elements):

IE Name	Type	Description
IMSI	Identity	International Mobile Subscriber Identity
MSISDN	Identity	Mobile phone number
APN	String	Access Point Name (e.g., "internet")
RAT Type	Enum	Radio Access Technology (EUTRAN)
Bearer Context	Grouped	Default bearer information
UE Time Zone	Timestamp	UE's timezone
ULI	Grouped	User Location Information (TAI, ECGI)
Serving Network	PLMN	MCC/MNC of serving network

Example:

```
Create Session Request
├── IMSI: 310260123456789
├── MSISDN: 14155551234
├── APN: internet
├── RAT Type: EUTRAN (6)
├── Bearer Context
│   ├── EBI: 5
│   ├── Bearer QoS (QCI 9, ARP, bitrates)
│   └── S5/S8 F-TEID (SGW-U tunnel endpoint)
└── ULI
    ├── TAI: MCC 310, MNC 260, TAC 12345
    └── ECGI: MCC 310, MNC 260, ECI 67890
```

Create Session Response

Direction: OmniPGW → SGW-C

Purpose: Acknowledge session creation

Key IEs:

IE Name	Type	Description
Cause	Result	Success or error code
Bearer Context	Grouped	Bearer information
PDN Address Allocation	IP	Allocated UE IP address (see UE IP Allocation)
APN Restriction	Enum	APN usage restrictions
PCO	Options	Protocol Configuration Options (see PCO Configuration)

Success Response:

```
Create Session Response
├── Cause: Request accepted (16)
├── PDN Address Allocation
│   └── IPv4: 100.64.1.42
├── Bearer Context
│   ├── EBI: 5
│   ├── Cause: Request accepted
│   └── S5/S8 F-TEID (PGW-U tunnel endpoint from PFCP)
├── APN Restriction: Public-1 (1)
└── PCO
    ├── DNS Server: 8.8.8.8
    ├── DNS Server: 8.8.4.4
    └── Link MTU: 1400
```

> The PDN Address Allocation layout follows the negotiated PDN type (TS 29.274 §8.14): a 4-byte IPv4 address; a prefix-length octet plus 16-byte IPv6 prefix; or for IPv4v6 a prefix-length octet plus 16-byte IPv6 plus 4-byte IPv4. If an IPv4v6 PDN is requested for an emergency (SOS) APN that has no IPv6 pool, the PGW-C degrades to an IPv4-only PAA (regular APNs are rejected) — see IPv4v6 request when the APN has no IPv6 pool.

Delete Session Request

Direction: SGW-C → OmniPGW

Purpose: Terminate PDN connection

Key IEs:

IE Name	Description
EBI	EPS Bearer ID to delete
Linked EBI	Related bearer (optional)

Delete Session Response

Direction: OmniPGW → SGW-C

Purpose: Acknowledge session deletion

Key IEs:

IE Name	Description
Cause	Success or error code

Bearer Management

Create Bearer Request

Direction: OmniPGW → SGW-C

Purpose: Create dedicated bearer (initiated by PCRF policy)

Triggered by:

- PCRF sends new PCC rule requiring dedicated bearer
- OmniPGW requests SGW-C to establish bearer

Delete Bearer Request

Direction: OmniPGW → SGW-C or SGW-C → OmniPGW

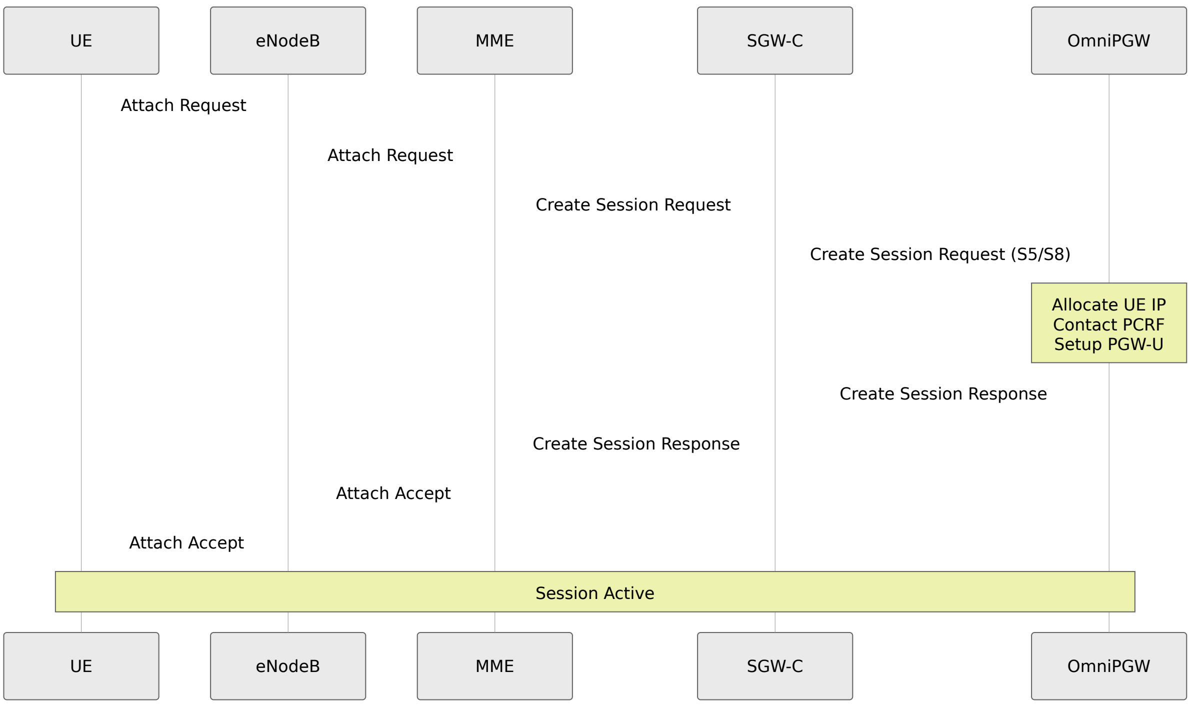
Purpose: Delete dedicated bearer

Scenarios:

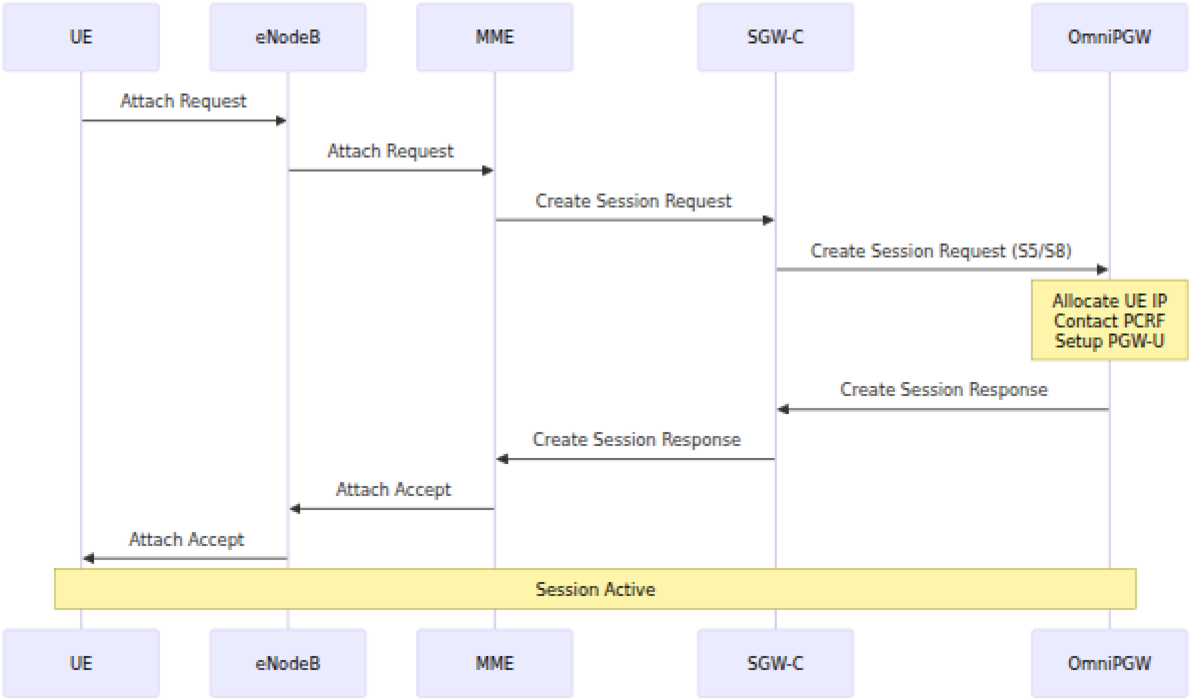
- **PGW-initiated:** PCRF policy change removes dedicated bearer
- **SGW-initiated:** Radio resource release

Message Flows

Session Establishment



Session Termination



Cause Codes

Success

Code	Name	Description
16	Request accepted	Successful operation

Errors (Permanent Failures)

Code	Name	When Used
65	User Unknown	PCRF rejected (IMSI not found)
66	No resources available	IP pool exhausted
93	Service not supported	Invalid APN
94	Semantic error in TFT	Invalid traffic flow template

Errors (Transient Failures)

Code	Name	When Used
72	Remote peer not responding	PCRF/PGW-U timeout
73	Collision with network initiated request	Simultaneous operations

---

Monitoring

S5/S8 Metrics

```
# Message counters
s5s8_inbound_messages_total{message_type="create_session_request"}
s5s8_inbound_messages_total{message_type="delete_session_request"}

# Error counters
s5s8_inbound_errors_total

# Message handling latency
s5s8_inbound_handling_duration_bucket

# Active TEIDs
teid_registry_count
```

Useful Queries

Session Creation Rate:

```
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])
```

Error Rate:

```
rate(s5s8_inbound_errors_total[5m])
```

Latency (p95):

```
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)
```

Troubleshooting

Issue: No Response from OmniPGW

Symptoms:

- SGW-C sends Create Session Request
- No response received
- Timeout at SGW-C

Causes:

1. Network connectivity issue
2. OmniPGW not listening on configured IP
3. Firewall blocking UDP 2123
4. Wrong TEID in request

Debug:

```
# Check OmniPGW is listening
netstat -ulnp | grep 2123

# Check for incoming packets
tcpdump -i any -n port 2123

# Verify configuration
grep "local_ipv4_address" config/runtime.exs

# Check firewall
iptables -L -n | grep 2123
```

Issue: Session Creation Fails

Symptoms:

- Create Session Response with error cause
- Session not established

Common Causes:

```
Cause 65 (User Unknown):
  → PCRF rejected subscriber
  → Check IMSI in HSS/SPR

Cause 66 (No resources):
  → IP pool exhausted
  → Check: curl http://pgw:9090/metrics | grep
address_registry_count
  → Expand IP pool

Cause 72 (Remote peer not responding):
  → PCRF timeout or PGW-U down
  → Check Gx connectivity
  → Check PFCP association
```

Issue: TEID Collision

Symptoms:

- Message routed to wrong session
- Unexpected behavior

Cause:

- TEID reused before cleanup
- Bug in TEID allocation

Resolution:

- Ensure unique TEID allocation
- Check TEID registry for leaks

Best Practices

Network Design

1. **Dedicated Network Interface**
    - Use separate VLAN for S5/S8
    - Isolate from management traffic
2. **MTU Optimization**
    - Ensure MTU supports GTP headers
    - Minimum MTU: 1500 bytes (1464 payload + 36 GTP)
3. **Redundancy**
    - Multiple OmniPGW instances
    - DNS-based load balancing from SGW-C

Performance

1. **UDP Buffer Sizes**
    - Increase socket buffers for high load
    - Typical: 4-8 MB per socket
2. **Connection Limits**
    - Plan for expected session count
    - Monitor TEID registry count

Security

1. **IP Filtering**
    - Only allow GTP-C from known SGW-C IPs
    - Use iptables or network ACLs

2. **Message Validation**
   - OmniPGW validates all incoming messages
   - Rejects malformed GTP-C packets

Related Documentation

Core Functions

- **Configuration Guide** - S5/S8 interface configuration, local IP setup
- **Session Management** - PDN session lifecycle, bearer establishment
- **UE IP Allocation** - IP address delivery via Create Session Response
- **PCO Configuration** - PCO parameters in GTP-C messages

Related Interfaces

- **Gn/Gp Interface** - GTP-C v1 for 2G/3G GGSN functionality
- **PFCP Interface** - User plane coordination with S5/S8 control plane
- **Diameter Gx Interface** - Policy integration with bearer setup
- **Diameter Gy Interface** - Charging integration with bearer management

Operations

- **Monitoring Guide** - S5/S8 GTP-C metrics, message tracking
- **Data CDR Format** - CDR generation from GTP-C sessions

Back to Operations Guide

OmniPGW S5/S8 Interface - *by Omnitouch Network Services*

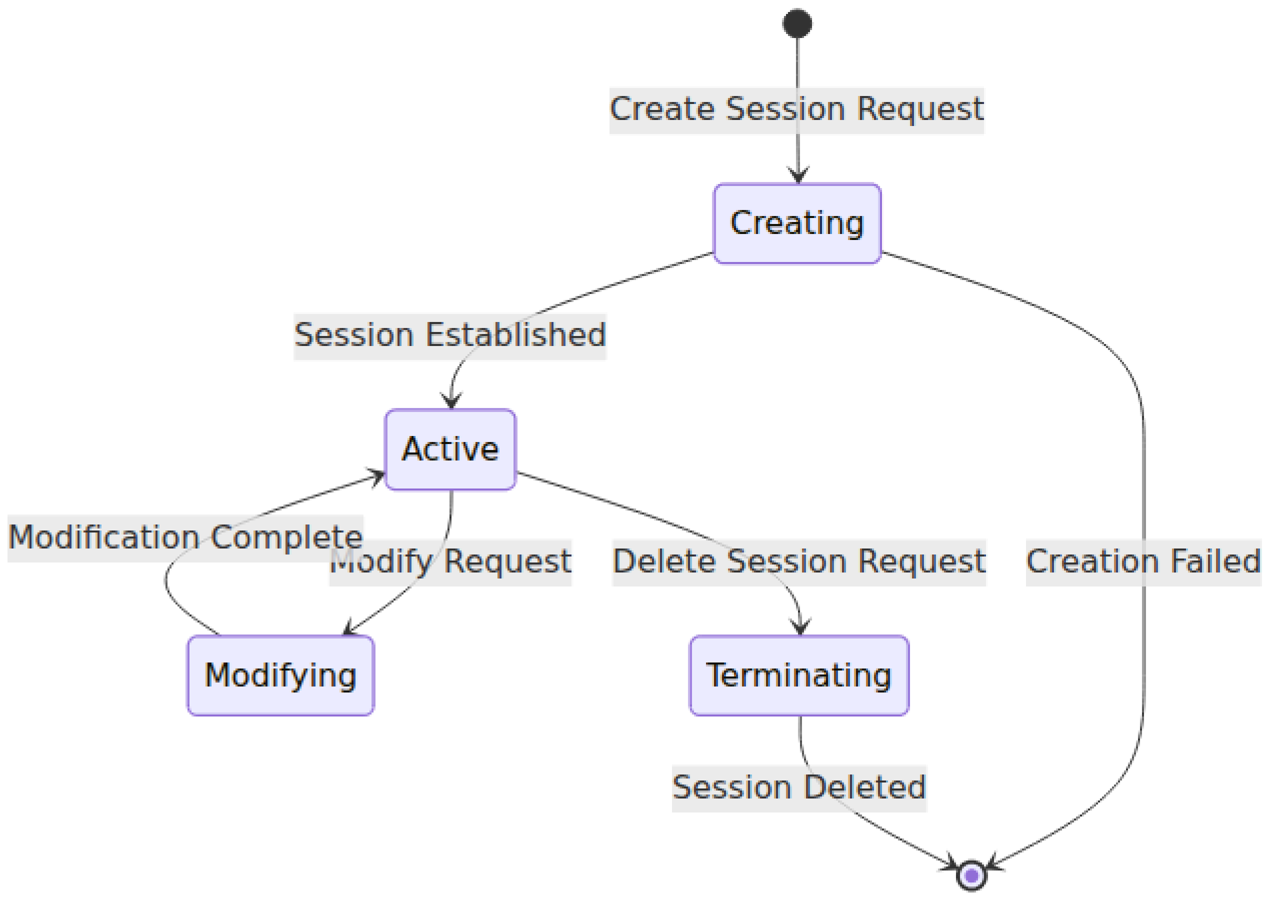
Session Management Guide

PDN Connection Lifecycle and Operations

OmniPGW by Omnitouch Network Services

Overview

A **PDN (Packet Data Network) Session** represents a UE's data connection through OmniPGW. Each session coordinates multiple interfaces and resources to enable data connectivity.



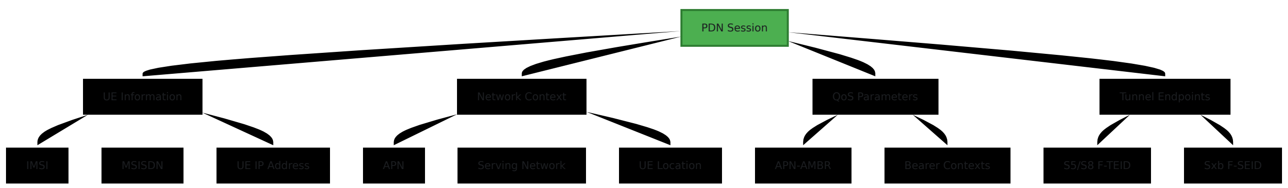
Session Components

Session Identifiers

Each session has multiple identifiers for different interfaces:

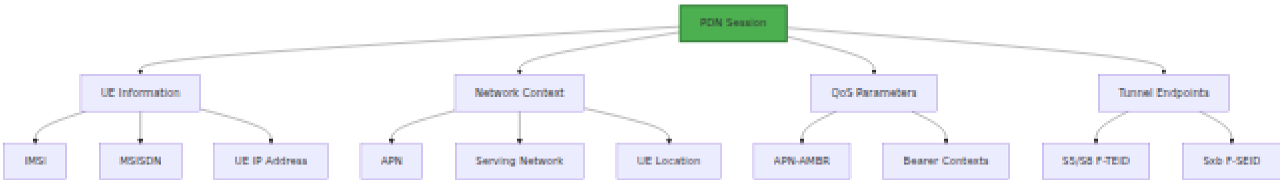
Identifier	Interface	Purpose
TEID	S5/S8 (GTP-C)	Tunnel Endpoint ID for SGW-C communication
SEID	Sxb (PFCP)	Session Endpoint ID for PGW-U communication
Session-ID	Gx (Diameter)	Diameter session for PCRF communication
Charging-ID	Accounting	Unique ID for billing/charging

Session Data



Session Creation

Call Flow



Steps

1. Receive Create Session Request (S5/S8)

Session creation is initiated via GTP-C signaling on the S5/S8 interface. See S5/S8 Interface for complete GTP-C protocol details and message formats.

Input:

- IMSI, MSISDN, IMEI
- APN (e.g., "internet")
- RAT Type (EUTRAN)
- UE Location (TAI, ECGI)
- Bearer Context (QoS, F-TEID)

2. Resource Allocation

```
- Allocate UE IP from APN pool
- Generate Charging ID
- Generate Gx Session-ID
- Allocate S5/S8 TEID
- Select PGW-U peer
```

3. Policy Request (Gx)

Request policy from PCRF:

- Send CCR-Initial

- Receive CCA-Initial with QoS and PCC rules

4. User Plane Setup (PFCP)

Program PGW-U with forwarding rules:

- Send Session Establishment Request
- Include PDRs, FARs, QERs, BAR
- Receive F-TEID for S5/S8 tunnel

5. Response to SGW-C

Send Create Session Response:

- UE IP Address
- S5/S8 F-TEID (from PGW-U)
- PCO (DNS, P-CSCF, MTU)
- Bearer Context

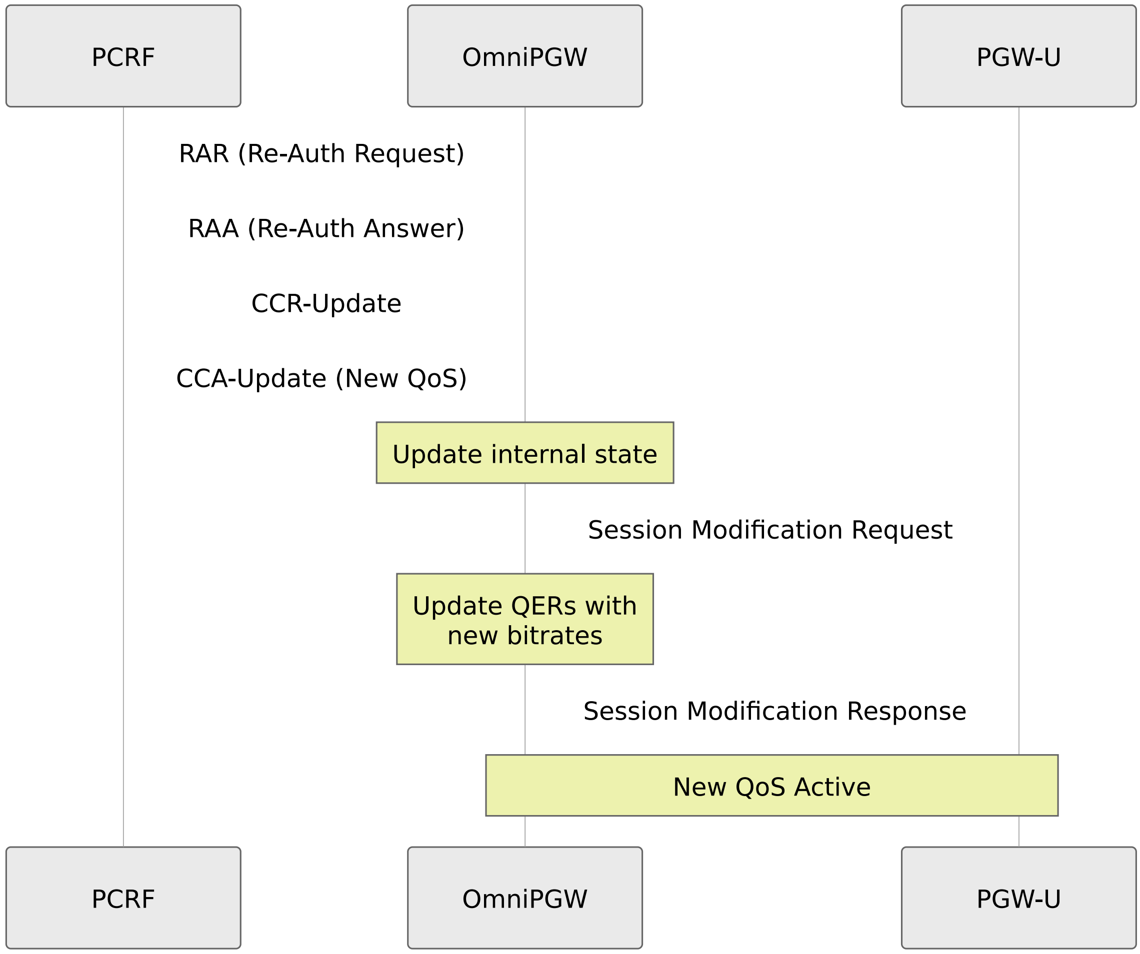
Session Modification

Triggers

Sessions can be modified due to:

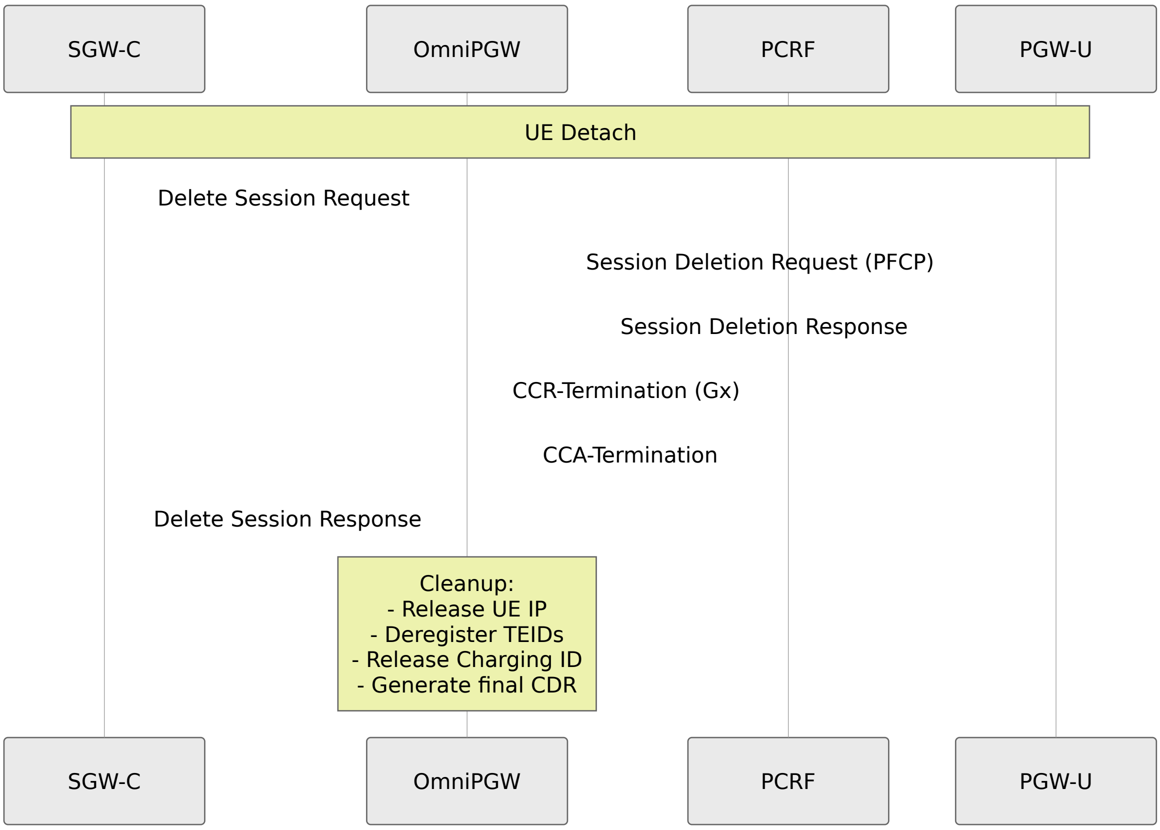
- **QoS Changes** - PCRF updates bitrates
- **Bearer Operations** - Add/remove dedicated bearers
- **Handover** - SGW change
- **Policy Updates** - New PCC rules from PCRF

QoS Modification Flow



Session Deletion

Call Flow



Cleanup Process

Resources Released:

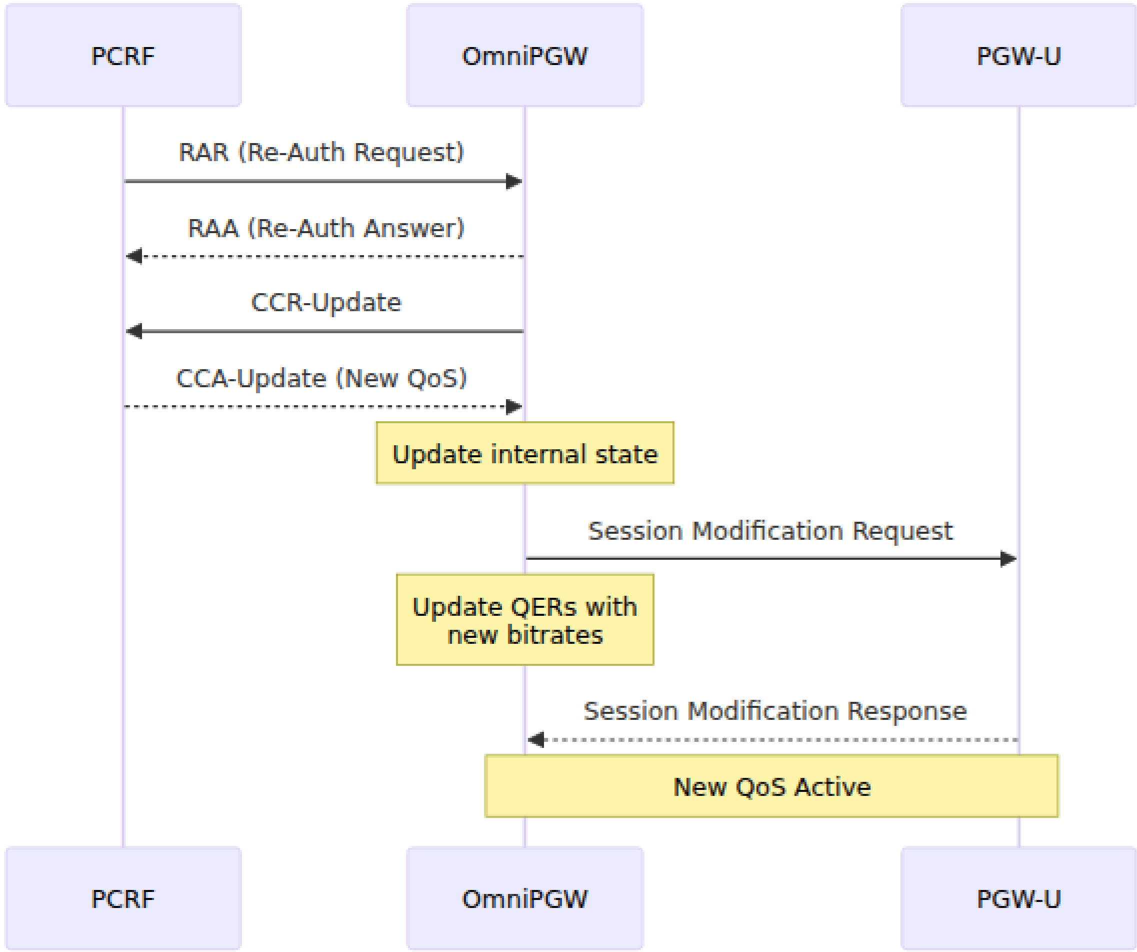
1. UE IP address → back to pool
2. TEID → removed from registry
3. SEID → removed from registry
4. Session-ID → removed from registry
5. Charging-ID → released
6. Session process terminated

Billing Records Generated:

- Final CDR (Charging Data Record) written for offline billing - See Data CDR Format

Session State

State Machine



Session Tracking

Registry Lookups:

```
By TEID (S5/S8):
  TEID 0x12345678 → Session PID

By SEID (Sxb):
  SEID 0xABCDEF → Session PID

By Session-ID (Gx):
  "pgw.example.com;123;456" → Session PID

By UE IP:
  100.64.1.42 → Session PID

By IMSI + EBI:
  "310260123456789" + EBI 5 → Session PID
```

Monitoring Sessions

Active Session Count

```
# Total active sessions
teid_registry_count

# PFCP sessions
seid_registry_count

# Gx sessions
session_id_registry_count
```

Session Metrics

```
# Session creation rate
rate(s5s8_inbound_messages_total{message_type="create_session_request

# Session deletion rate
rate(s5s8_inbound_messages_total{message_type="delete_session_request

# Session creation latency (p95)
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)
```

Common Issues

Session Creation Fails

Causes:

1. **IP Pool Exhausted** - No IPs available
2. **PCRF Unreachable** - Gx timeout
3. **PGW-U Down** - No PFCP peer available
4. **PCRF Rejection** - User unknown, not authorized

Debug:

```
# Check IP pool
curl http://pgw:9090/metrics | grep address_registry_count

# Check PCRF connectivity
# Check for Gx errors in logs

# Check PGW-U association
# Verify PFCP peer status
```

Session Stuck/Stale

Symptoms:

- Session not deleted properly
- Resources not released
- Registries show higher count than expected

Causes:

1. Delete Session Request not received
2. Session process crash without cleanup
3. Registry leak

Resolution:

```
# Restart OmniPGW (releases all sessions)
# Implement session timeout mechanism
```

UE Cannot Establish Session

Symptoms:

- UE attach fails
- Create Session Response with error cause

Common Causes & Responses:

Cause Value	Meaning	Action
User Unknown	PCRF rejected (IMSI not in database)	Provision subscriber
No Resources Available	IP pool exhausted	Expand IP pool
Remote Peer Not Responding	PCRF/PGW-U timeout	Check connectivity
Service Not Supported	Invalid APN	Configure APN pool

Best Practices

Session Limits

Configure appropriate capacity:

```
Expected concurrent users: 10,000
Session overhead per user: ~10KB RAM
Total RAM for sessions: ~100MB

Erlang VM settings:
  - Max processes: 262,144 (default)
  - Process heap size: Adjust based on load
```

Session Cleanup

Ensure proper cleanup:

1. Always respond to Delete Session Requests
2. Implement session timeout for stale sessions
3. Monitor registry counts for leaks

High Availability

Session Redundancy:

- Use stateless design (sessions tied to instance)
- Implement session database for HA (future)
- DNS/load balancer for failover

Session Data Elements

What Information Does a Session Store?

Each active PDN session maintains the following information:

UE Identification:

- IMSI: "310260123456789" (subscriber identity)
- MSISDN: "14155551234" (phone number)
- MEI/IMEI: Device identifier

PDN Connection Details:

- APN: "internet" (network name)
- UE IP Address: 100.64.1.42 (allocated IP)
- PDN Type: IPv4, IPv6, or IPv4v6

Session Identifiers:

- Charging ID: Unique billing identifier
- Default Bearer EBI: EPS Bearer Identifier (typically 5)

QoS Parameters:

- APN-AMBR: Aggregate Maximum Bit Rate
  - Uplink: 100 Mbps
  - Downlink: 50 Mbps

Forwarding Rules:

- PDRs (Packet Detection Rules): Match packets
- FARs (Forwarding Action Rules): Forward/drop actions
- QERs (QoS Enforcement Rules): Rate limiting
- BAR (Buffering Action Rule): Downlink buffering

Interface Context:

- S5/S8 State: Local/remote TEIDs, SGW-C address
- Sxb State: Local/remote SEIDs, PGW-U address
- Gx State: Diameter Session-ID, request counter

Live Session Monitoring

OmniPGW exposes a real-time **OAM REST API** for inspecting and managing active sessions without needing to query metrics or logs. The API is served over HTTPS/TLS on port `8443` with all routes under the `/api` prefix:

```
GET    /api/sessions                    # all active sessions
GET    /api/sessions?search=<imsi>   # filter by IMSI / MSISDN /
IP
GET    /api/sessions/<imsi>          # one session's full detail
PUT    /api/sessions/<imsi>          # update live session state
(operator action)
DELETE /api/sessions/<imsi>          # tear down a session
(operator action)
```

Interactive API docs (Swagger UI) are available at `https://<omnipgw-ip>:8443/api/docs`.

```
# List all active sessions
curl -k https://<omnipgw-ip>:8443/api/sessions

# Filter by IMSI, MSISDN, or IP address
curl -k "https://<omnipgw-ip>:8443/api/sessions?
search=310170123456789"
```

UE Search & Deep Dive

Endpoints:

```
GET /api/sessions?search=<imsi|msisdn|ip>   # find a subscriber's
sessions
GET /api/sessions/<imsi>                    # full detail for one
session
```

The `search` parameter accepts any of:

- **IMSI** (e.g., "310170123456789")
- **MSISDN** (phone number)
- **IP Address** (e.g., "100.64.1.42")

```
# Deep-dive a single subscriber's session
curl -k https://<omnipgw-ip>:8443/api/sessions/310170123456789
```

Session detail returns comprehensive information:

a) Active Sessions

- All active sessions for this subscriber
- IMSI, MSISDN, UE IP Address
- APN, RAT Type
- PGW TEID, SGW TEID

b) Current Location Real-time location data from the session:

- **TAC** (Tracking Area Code) - Tracking area where UE is located

- **Cell ID (ECI)** - E-UTRAN Cell Identifier
- **ECGI** - E-UTRAN Cell Global Identifier (PLMN + ECI)
- **MCC/MNC** - Mobile Country Code / Mobile Network Code

c) Bearer Information Detailed bearer listing with QoS parameters:

Default Bearer:

- EBI (EPS Bearer Identifier)
- QCI (QoS Class Identifier)
- Charging Rule Name
- APN-AMBR (uplink/downlink)

Dedicated Bearers (if active):

- EBI, QCI, Charging Rule Name
- MBR UL/DL (Maximum Bit Rate)
- GBR UL/DL (Guaranteed Bit Rate)

d) Charging Information (Gy Interface)

- Gy Session ID
- Granted Quota, Used Quota
- Charging Characteristics

e) Policy Information (Gx Interface)

- Gx Session ID
- PCRF Origin/Destination Host
- CC Request Number
- Installed Charging Rules (PCC rules from bearers)

f) Recent Events

- Event history for this subscriber (see Session History & Audit Log)
- Session create/update/delete events

Use Cases:

- Troubleshoot specific subscriber issues
- Verify session establishment
- Check assigned IP address
- Inspect session parameters

All Active Sessions

Endpoints:

```
GET    /api/sessions                  # list all active PDN
sessions
GET    /api/sessions?search=<term>   # filter the list
PUT    /api/sessions/<imsi>          # update a live session's
state
DELETE /api/sessions/<imsi>          # tear down a session
```

Purpose: Enumerate all active PDN sessions. Poll `GET /api/sessions` on your preferred interval for a live view of the session table.

```
# Snapshot of all active sessions
curl -k https://<omnipgw-ip>:8443/api/sessions

# Filter by IMSI prefix, UE IP, MSISDN, or APN
curl -k "https://<omnipgw-ip>:8443/api/sessions?search=100.64"

# Operator actions on a specific session
curl -k -X PUT    https://<omnipgw-
ip>:8443/api/sessions/310170123456789   # update state
curl -k -X DELETE https://<omnipgw-
ip>:8443/api/sessions/310170123456789   # tear down
```

Quick Session Information (returned per session):

- **IMSI** - Subscriber identity
- **UE IP** - Allocated IP address
- **SGW TEID** - S5/S8 tunnel ID from SGW
- **PGW TEID** - S5/S8 tunnel ID from OmniPGW

- **APN** - Access Point Name

Filtering (via `?search=`):

- IMSI (e.g., "310260")
- UE IP address (e.g., "100.64")
- MSISDN / phone number
- APN name

Full Details (`GET /api/sessions/<imsi>`):

- Full subscriber information (IMSI, MSISDN, IMEI)
- Network context (RAT type, serving network MCC/MNC)
- QoS parameters (AMBR uplink/downlink)
- Tunnel identifiers (both TEIDs in hex format)
- Process ID for debugging
- Complete session state (raw data structure)

Cross-Referenced Health Endpoints

The OAM API also exposes peer/topology health that the session view depends on:

```
GET /api/upf         # PFCP / UPF peer status and associations
GET /api/diameter    # Diameter (Gx/Gy) peer connectivity
```

```
# Verify user-plane and Diameter peer connectivity
curl -k https://<omnipgw-ip>:8443/api/upf
curl -k https://<omnipgw-ip>:8443/api/diameter
```

Use Cases:

- Verify peer connections (UPF associations, PCRF/OCS connectivity)
- Correlate session establishment failures with peer health
- Quick network health check

Session History & Audit Log

Endpoint:

```
GET /api/session_history[?type=&search=&page=&page_size=]
```

Purpose: Track historical session events and audit trail.

Parameters:

- `type` - filter by event type (e.g., session created, session deleted)
- `search` - filter by IMSI, MSISDN, IP address, or TEID
- `page`, `page_size` - paginate through results

```
# Recent deletion events for a subscriber, first page
curl -k "https://<omnipgw-ip>:8443/api/session_history?
type=session_deleted&search=310170123456789&page=1&page_size=50"
```

Event Types Tracked:

- Session creation events
- Session deletion events
- Modification events
- Error events

Use Cases:

- Audit trail for compliance (page through and archive results)
- Historical session analysis
- Troubleshoot past issues
- Generate usage reports
- Track session patterns over time

Operational Use Cases

Session Verification:

```
1. User reports connectivity issue
2. GET /api/sessions?search=<imsi|msisdn> to locate the subscriber
3. GET /api/sessions/<imsi> to verify the session exists and the
UE has an IP address
4. Check QoS values match subscriber plan
5. Verify tunnel endpoints are established
```

Capacity Monitoring:

- Poll `GET /api/sessions` (or the `teid_registry_count` metric) for the active session count
- Compare against licensed capacity
- Identify usage patterns by APN (via `?search=<apn>`)

Troubleshooting:

- Find a specific session by any identifier via `?search=`
- Inspect full session state (`GET /api/sessions/<imsi>`) without SSH/IEx
- Verify SGW and PGW TEIDs match between systems
- Check AMBR values applied from PCRF
- Tear down a stuck session with `DELETE /api/sessions/<imsi>`

Advantages Over Metrics:

- See individual session details (metrics show aggregates)
- Search and filter capabilities via `?search=`
- Real-time state inspection
- Scriptable/automatable (JSON over HTTPS) for integration with tooling

---

Related Documentation

Core Session Functions

- **PFCP Interface** - User plane session establishment, PDRs, FARs, QERs, URRs

- **UE IP Allocation** - IP address assignment, APN pool management
- **PCO Configuration** - DNS, P-CSCF, MTU parameters delivered to UE
- **Configuration Guide** - UPF selection, session establishment flows

Policy and Charging

- **Diameter Gx Interface** - PCRF policy control, PCC rules, QoS management
- **Diameter Gy Interface** - OCS online charging, quota tracking
- **Data CDR Format** - Offline charging records generation

Network Interfaces

- **S5/S8 Interface** - GTP-C protocol, SGW-C communication
- **QoS & Bearer Management** - Bearer QoS enforcement

Operations

- **Monitoring Guide** - Session metrics, active session tracking, alerts
- **P-CSCF Monitoring** - IMS session monitoring

Back to Operations Guide

OmniPGW Session Management - *by Omnitouch Network Services*

OmniPGW Troubleshooting Guide

Troubleshooting Procedures and Common Issues

by Omnitouch Network Services

Table of Contents

Overview

This guide provides step-by-step troubleshooting procedures for common OmniPGW operational issues. Each issue includes:

- **Symptom**: What you'll observe
- **Likely Causes**: Common root causes
- **Diagnosis**: How to confirm the cause
- **Resolution**: Step-by-step fix
- **Prevention**: How to avoid recurrence

Related Documentation

- **Monitoring Guide** - Prometheus metrics, alerting, performance monitoring
- **Configuration Guide** - System configuration reference

Troubleshooting Tools

OAM API

Base URL: `https://<omnipgw_ip>:8443` (HTTPS/TLS; all routes under the `/api` prefix)

Interactive API docs (Swagger UI): `https://<omnipgw_ip>:8443/api/docs`

Key Endpoints:

- `GET /api/sessions[?search=<imsi|msisdn|ip>]`, `GET /api/sessions/<imsi>` - Active session viewer (search by IMSI, IP, MSISDN)
- `GET /api/diameter[?status=connected|disconnected]`, `GET /api/diameter/<origin-host>` - Diameter peer status (Gx PCRF, Gy OCS)
- `GET /api/charging[?search=<imsi>]`, `GET /api/charging/<imsi>` - Online-charging (Gy) status
- `GET /api/upf`, `GET /api/upf/<ip>` - UPF / PFCP peer status (PGW-U connectivity)
- `GET /api/ip_pools`, `GET /api/ip_pools/<name>` - IP pool utilization
- `GET /api/session_history[?type=&search=&page=&page_size=]` - Session history
- `GET /api/status` - Health check (`{"result":"ok"}`)

Example:

```
curl -k https://<omnipgw_ip>:8443/api/sessions
```

Log streaming is no longer served by the platform; use host logs (`journalctl`/log files) with the filters below.

Prometheus Metrics

Access: `http://<omnipgw_ip>:9090/metrics`

Key Metrics:

- `teid_registry_count` - Active sessions
- `address_registry_count` - Allocated UE IPs
- `sxb_inbound_errors_total` - PFCP errors
- `gx_inbound_errors_total` - Diameter Gx errors
- `gy_inbound_errors_total` - Diameter Gy errors

See Monitoring Guide for complete metrics reference.

Log Analysis

Logs: Inspect host logs (`journalctl -u omnipgw_c` or the configured log file) and grep for the strings below. For a structured, queryable view of session lifecycle events use `GET /api/session_history[?type=&search=&page=&page_size=]`.

Common Log Filters:

- "create_session_request" - Session establishment
- "Credit Control" - Gx/Gy interactions
- "PFCP Session" - User plane programming
- "error" or "ERROR" - Error messages
- "timeout" - Timeout issues

Session Establishment Issues

Issue: Create Session Request Rejected with "No Resources Available"

Symptom:

- SGW-C receives Create Session Response with cause "No resources available" (73)
- All new session attempts fail
- Existing sessions continue working
- Logs: `[PGW-C] Create Session Request blocked - invalid license`

Wireshark capture showing Create Session Response with "No resources available" cause

Likely Cause:

- Invalid or expired OmniPGW license
- License server unreachable

Diagnosis:

1. **Check license metric:**

```
license_status
```

  - Value of 0 indicates invalid license

2. **Check logs for license warnings:**

   - Search for "license" or "License"
   - Look for "Unable to contact license server" messages

3. **Verify license server connectivity:**

   - Check configured URL in `config/runtime.exs` under `:license_client`
   - Default: `https://localhost:10443/api`

Resolution:

1. **Verify license server is reachable:**

   ```
   curl -k https://<license_server_ip>:10443/api/status
   ```

2. **Check license configuration in `config/runtime.exs`:**

   ```
   config :license_client,
     license_server_api_urls:
   ["https://<license_server_ip>:10443/api"],
     licensee: "Your Company Name"
   ```

3. **Verify product is licensed:**

   - Product name: `omnipgwc`
   - Contact Omnitouch to verify license status

4. **Restart OmniPGW after configuration changes**

Prevention:

- Monitor `license_status` metric with critical alerts
- Ensure license server high availability

- Set up license expiry alerts before expiration

Issue: Create Session Request Rejected (Other Causes)

Symptom:

- SGW-C receives Create Session Response with error cause
- Users cannot establish PDN connections
- Metric: `s5s8_inbound_errors_total` increasing

Likely Causes:

1. IP pool exhausted
2. PCRF (Gx) unreachable or rejecting policy
3. PGW-U (PFCP) unavailable
4. Invalid APN configuration

Diagnosis:

1. **Check IP pool utilization:**

   ```
   address_registry_count
   ```

   - If equals configured pool size, pool is exhausted

2. **Check PCRF connectivity:**

   - Query `GET /api/diameter` (or `GET /api/diameter?status=disconnected`)
   - Look for PCRF peer status = "disconnected"
   - Logs: Search "Credit Control Answer" for errors

3. **Check PFCP peer status:**

   - Query `GET /api/upf` (or `GET /api/upf/<ip>` for a specific PGW-U)
   - Look for "Association: DOWN"

- Metric: `pfcp_peer_associated` = 0

4. **Check APN configuration:**

   - Review `config/runtime.exs` under `ue.apn_map`
   - Verify requested APN exists in configuration

Resolution:

For IP Pool Exhaustion:

1. Identify stale sessions: query `GET /api/sessions` (optionally `GET /api/sessions?search=<imsi|msisdn|ip>`), look for old sessions
2. Expand IP pool in `config/runtime.exs`:

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => "10.0.0.0/23"  # Changed from /24 to /23
(doubles capacity)
    }
  }
```

3. Restart OmniPGW
4. Verify: `curl http://<ip>:9090/metrics | grep address_registry_count`

For PCRF Connectivity Issues:

1. Check network connectivity: `ping <pcrf_ip>`
2. Verify PCRF Diameter service: `telnet <pcrf_ip> 3868`
3. Check `config/runtime.exs` Diameter peer configuration
4. Restart OmniPGW if config changed
5. Verify via `GET /api/diameter?status=connected` (peer should show "connected")

For PFCP Issues:

- See PFCP / User Plane Issues section

Prevention:

- Monitor IP pool utilization with alerts at 80%
- Monitor PCRF connectivity with Diameter peer alerts
- Implement session cleanup for idle sessions

Issue: Sessions Stuck in Intermediate State

Symptom:

- Session appears in `GET /api/sessions` but incomplete
- Metrics show growing session count but no user traffic
- Delete Session Request fails or times out

Likely Causes:

1. PFCP Session Establishment failed but S5/S8 session created
2. PCRF CCR-Initial timed out
3. Create Bearer Request (dedicated bearer) failed
4. Network disruption during session setup

Diagnosis:

1. **Search for the session via the OAM API:**
    - `GET /api/sessions/<imsi>` (or `GET /api/sessions?search=<imsi>`)
    - Check if `pfcp_seid` is present (if missing, PFCP failed)
    - Check if `gx_session_id` is present (if missing, Gx failed)
2. **Check logs for the IMSI:**
    - Filter logs by IMSI
    - Look for "Session Establishment Request" (PFCP)
    - Look for "Credit Control Request" (Gx)
    - Look for timeout or error messages
3. **Check metrics:**

```
# Sessions with TEID but no PFCP session
teid_registry_count - seid_registry_count

# Sessions with TEID but no Gx session
teid_registry_count - session_id_registry_count
```

Resolution:

1. **For PFCP establishment failures:**
    - Check PGW-U health and logs
    - Verify PFCP association: `GET /api/upf`
    - Send Delete Session Request from SGW-C to cleanup
2. **For Gx timeout issues:**
    - Check PCRF latency: `histogram_quantile(0.95, rate(gx_outbound_transaction_duration_bucket[5m]))`
    - Increase Gx timeout in `config/runtime.exs` if needed
    - Send Delete Session Request to cleanup
3. **Manual cleanup (last resort):**
    - Currently requires OmniPGW restart to clear stuck sessions
    - Monitor `teid_registry_count` before/after restart to confirm cleanup

Prevention:

- Monitor PFCP and Gx latency metrics
- Implement session timeout/cleanup for incomplete sessions
- Alert on registry count mismatches

PFCP / User Plane Issues

Issue: PFCP Association Down

Symptom:

- `GET /api/upf` shows "Association: DOWN"
- All new session establishments fail
- Metric: `pfcp_peer_associated` = 0
- Logs: "PFCP heartbeat timeout" or "Association Setup failed"

Likely Causes:

1. PGW-U unreachable (network issue)
2. PGW-U crashed or restarted
3. PFCP configuration mismatch (IP, port)
4. Firewall blocking UDP 8805

Diagnosis:

1. **Check network connectivity:**

   ```
   ping <pgw_u_ip>
   nc -u -v <pgw_u_ip> 8805
   ```

2. **Check PFCP configuration:**

   - Review `config/runtime.exs` under `upf.peer_list`
   - Verify IP address and node ID match PGW-U configuration

3. **Check PGW-U status:**

   - Access PGW-U logs
   - Verify PGW-U is running: `systemctl status omnipgw_u` (or equivalent)

4. **Check metrics:**

```
# Heartbeat failures
pfcp_consecutive_heartbeat_failures

# PFCP error rate
rate(sxb_inbound_errors_total[5m])
```

Resolution:

1. **For network issues:**
    - Verify routing: `traceroute <pgw_u_ip>`
    - Check firewall rules: Ensure UDP 8805 allowed
    - Check security groups (if cloud deployment)
2. **For PGW-U crashes:**
    - Restart PGW-U service
    - Wait 30 seconds for association re-establishment
    - Verify via `GET /api/upf` (should show "Association: UP")
3. **For configuration issues:**
    - Correct `config/runtime.exs` PFCP peer configuration
    - Restart OmniPGW
    - Verify association established

Prevention:

- Monitor `pfcp_peer_associated` metric with critical alerts
- Monitor `pfcp_consecutive_heartbeat_failures` (alert at > 2)
- Implement redundant PGW-U instances
- Enable PFCP keepalive/heartbeat (should be default)

Issue: PFCP Session Modification Failures

Symptom:

- Dedicated bearer creation fails
- QoS policy updates (from PCRF RAR) fail
- Logs: "Session Modification Request failed"
- Metric: `sxb_inbound_errors_total{message_type="session_modification_response"}` increasing

Likely Causes:

1. Invalid PFCP rules (PDR/FAR/QER references)
2. PGW-U resource exhaustion
3. Rule ID conflicts
4. PGW-U software bug

Diagnosis:

1. **Check logs:**
   - Filter for "Session Modification" and SEID
   - Look for error cause codes in PFCP response
   - Common causes: "Rule ID already exists", "Out of resources"
2. **Check PGW-U logs:**
   - Look for PFCP processing errors
   - Check resource utilization (CPU, memory)
3. **Check session state via the OAM API:**
   - `GET /api/sessions/<imsi>` (or `GET /api/sessions?search=<imsi>`)
   - Review `pdr_map`, `far_map`, `qer_map` for conflicts
   - Look for duplicate IDs

Resolution:

1. **For rule conflicts:**
   - Delete and recreate dedicated bearer
   - If persistent, Delete Session and have UE reconnect

2. **For PGW-U resource issues:**

    - Check PGW-U capacity (sessions, PDRs, throughput)
    - Scale PGW-U if needed
    - Reduce session load on affected PGW-U instance

3. **For software bugs:**

    - Capture full session state (`GET /api/sessions/<imsi>`)
    - Capture PFCP message logs
    - Report to vendor with reproduction steps

Prevention:

- Monitor PGW-U resource utilization
- Test dedicated bearer creation in staging
- Monitor `sxb_inbound_errors_total` with alerts

Diameter (Gx/Gy) Issues

Issue: PCRF Peer Disconnected (Gx)

Symptom:

- `GET /api/diameter` (or `GET /api/diameter?status=disconnected`) shows PCRF peer "disconnected"
- Sessions created without QoS policies (default QCI=5 applied)
- Logs: "Diameter peer connection failed" or "CER/CEA timeout"

Likely Causes:

1. PCRF unreachable (network issue)
2. PCRF service down
3. Diameter configuration mismatch (Origin-Host, Realm)
4. Firewall blocking TCP 3868

Diagnosis:

1. **Check network connectivity:**

   ```
   ping <pcrf_ip>
   telnet <pcrf_ip> 3868
   ```

2. **Check Diameter configuration:**

   - Review `config/runtime.exs` under `diameter.peer_list`
   - Verify `host`, `realm`, `ip` match PCRF configuration
   - Check `origin_host` matches what PCRF expects

3. **Check PCRF logs:**

   - Look for CER (Capabilities-Exchange-Request) from PGW-C
   - Look for rejection reasons

4. **Check metrics:**

   ```
   # Diameter connection errors
   diameter_peer_connected{peer="<pcrf_host>"}
   ```

Resolution:

1. **For network issues:**

   - Verify routing to PCRF
   - Check firewall rules: Ensure TCP 3868 allowed
   - Test connectivity: `nc -v <pcrf_ip> 3868`

2. **For PCRF service down:**

   - Restart PCRF service
   - Wait for automatic reconnection (30s retry interval)
   - Verify via `GET /api/diameter?status=connected`

3. **For configuration mismatch:**

- Correct `config/runtime.exs` Diameter configuration:

  ```
  config :pgw_c,
    diameter: %{
      host: "pgw-c.epc.mnc999.mcc999.3gppnetwork.org",  #
  Must match PCRF config
      realm: "epc.mnc999.mcc999.3gppnetwork.org",
      peer_list: [
        %{
          host: "pcrf.epc.mnc999.mcc999.3gppnetwork.org",
          realm: "epc.mnc999.mcc999.3gppnetwork.org",
          ip: "192.168.1.100",
          initiate_connection: true
        }
      ]
    }
  ```

- Restart OmniPGW
- Verify connection established

Prevention:

- Monitor Diameter peer connectivity with critical alerts
- Implement redundant PCRF instances (if supported)
- Document Diameter configuration in runbook

Issue: CCR/CCA Timeouts (Gx Policy Requests)

Symptom:

- Session establishment slow (> 5 seconds)
- Logs: "Credit Control Request timeout"
- Metric: `gx_outbound_transaction_duration` very high (> 5s)
- Sessions created with default QoS (fallback behavior)

Likely Causes:

1. PCRF overloaded
2. PCRF database slow

3. Network latency
4. PCRF software issue

Diagnosis:

1. **Check Gx latency:**

   ```
   # P95 latency
   histogram_quantile(0.95,
   rate(gx_outbound_transaction_duration_bucket[5m]))

   # P99 latency (outliers)
   histogram_quantile(0.99,
   rate(gx_outbound_transaction_duration_bucket[5m]))
   ```

2. **Check PCRF health:**
   - Access PCRF monitoring dashboards
   - Check CPU, memory, database connections
   - Review PCRF logs for slow queries

3. **Check network latency:**

   ```
   ping -c 100 <pcrf_ip> | tail -1  # Check avg latency
   ```

4. **Check logs:**
   - Count CCR/CCA exchanges: Filter "Credit Control"
   - Measure time between "Sending CCR" and "Received CCA"

Resolution:

1. **For PCRF overload:**
   - Scale PCRF (add instances)
   - Reduce CCR message size if possible
   - Tune PCRF thread pools/workers

2. **For network latency:**

- Investigate network path (routers, switches)
- Consider co-locating PGW-C and PCRF

3. **Temporary workaround (increase timeout):**

    - Edit `config/runtime.exs`:

    ```
    config :pgw_c,
      diameter: %{
        transaction_timeout_ms: 10000  # Increase from default 5000
      }
    ```

    - Restart OmniPGW
    - **Note:** This only masks the issue; fix root cause

Prevention:

- Monitor Gx latency with alerts (warning > 1s, critical > 5s)
- Capacity plan PCRF for expected session rate
- Test PCRF performance under load

Issue: OCS Peer Disconnected (Gy)

Symptom:

- `GET /api/diameter` (or `GET /api/diameter?status=disconnected`) shows OCS peer "disconnected"; `GET /api/charging` reflects online-charging (Gy) status
- Sessions cannot be charged (online charging fails)
- Logs: "Gy peer connection failed"

Diagnosis and Resolution:

Similar to PCRF Peer Disconnected, but for Gy interface.

Key differences:

- Port: Typically TCP 3868 (same as Gx)
- Impact: Charging fails, sessions may be rejected or allowed without charging (depends on config)
- Configuration: Check `diameter.peer_list` for OCS entry

See: Diameter Gy Interface for Gy-specific troubleshooting

IP Allocation Issues

Issue: IP Pool Exhausted

Symptom:

- Create Session Request rejected with cause "No resources available"
- Metric: `address_registry_count` equals configured pool size
- `GET /api/sessions` shows many active sessions; `GET /api/ip_pools` shows the pool at capacity
- Logs: "IP allocation failed: pool exhausted"

Likely Causes:

1. Pool too small for subscriber base
2. Sessions not releasing IPs (Delete Session failures)
3. Rapid session churn without cleanup
4. IP address leak

Diagnosis:

1. **Check pool utilization:**

   ```
   # For /24 subnet (254 IPs)
   (address_registry_count / 254) * 100
   ```

2. **Check configured pool size:**

- Review `config/runtime.exs` under `ue.subnet_map`
- Example: "10.0.0.0/24" = 254 usable IPs

3. **Compare session count to IP count:**

```
# Should be approximately equal
teid_registry_count
address_registry_count
```

4. **Review active sessions:**

- `GET /api/sessions` (and `GET /api/ip_pools/<name>` for per-pool utilization)
- Sort by session start time
- Look for very old sessions (potential leaks)

Resolution:

Immediate (expand pool):

1. Edit `config/runtime.exs` (each APN maps to a **list** of CIDR strings):

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => ["10.0.0.0/22"]  # 1022 IPs (was /24 = 254 IPs)
    }
  }
```

2. Restart OmniPGW
3. Verify: Sessions can now establish

Long-term (cleanup):

1. Identify stale sessions via `GET /api/sessions`
2. Coordinate with SGW-C to send Delete Session Requests
3. Implement session timeout policy on PCRF/SGW
4. Monitor `address_registry_count` to verify pool freed up after cleanup

Prevention:

- Monitor IP pool utilization with alerts:
  - Warning: > 70%
  - Critical: > 85%
- Trend analysis to predict exhaustion
- Implement session idle timeout
- Regular session audits

Issue: Duplicate IP Address Assigned

Symptom:

- UE reports IP address conflict
- Logs: "IP already allocated" warning
- Two sessions in `GET /api/sessions` with same IP address

Likely Causes:

1. Software bug (rare)
2. Database inconsistency after crash
3. Manual intervention error

Diagnosis:

1. **Search for the IP via the OAM API:**

   - `GET /api/sessions?search=<ip>`
   - Check if multiple IMSIs have same IP

2. **Check logs:**

   - Search for IP address
   - Look for "IP allocation" events

Resolution:

1. **Identify affected sessions:**

    - Note both IMSIs with duplicate IP

2. **Delete one session:**

    - Coordinate with SGW-C to send Delete Session Request for one IMSI
    - Prefer deleting the newer session

3. **UE reconnects:**

    - UE should automatically reconnect
    - Will receive new unique IP

4. **If persistent:**

    - Restart OmniPGW to rebuild IP registry
    - All sessions will be lost (coordinate maintenance window)

Prevention:

- Monitor for duplicate allocations (no built-in metric currently)
- Regular database integrity checks (if applicable)

Issue: IPv4v6 Session on an APN With No IPv6 Pool

Symptom:

- A UE requests an **IPv4v6** PDN but the matched APN has only an IPv4 CIDR configured in `ue.subnet_map`
- An **emergency (SOS)** APN comes up **IPv4-only** (no IPv6 in the PAA); logs show a `warning` like "No IPv6 pool for emergency APN ...; granting IPv4-only"
- A **regular** APN has the session **rejected**; logs show an `error` like "No IPv6 pool for APN ...; rejecting IPv4v6 request"

Likely Causes:

1. The APN intentionally has no IPv6 range (common on IPv4-only deployments)
2. An IPv6 CIDR was omitted from the APN's `ue.subnet_map` entry by mistake

Behaviour:

- `AddressRegistry.register_new_ipv6/1` returns `{:error, :no_ipv6_pool}` (it does not crash on an empty pool).
- Emergency (SOS) APNs (name contains `"sos"`) degrade to IPv4-only — IPv4 address kept, IPv6 cleared, PDN type set to IPv4, and the P-CSCF is still returned in the PCO so emergency IMS registration works.
- Regular APNs are rejected (no silent downgrade), surfacing the unsupported request / misconfiguration.

Resolution:

1. If the APN should offer dual-stack, add an IPv6 CIDR to its `ue.subnet_map` entry and restart OmniPGW.
2. If the IPv4-only behaviour is intended (e.g. an SOS APN on an IPv4-only network), no action is required.

See IPv4v6 request when the APN has no IPv6 pool and UE IP Allocation → Issue 4.

Quick Reference

Common Prometheus Queries

```
# Active sessions
teid_registry_count

# Session setup rate (per second)
rate(s5s8_inbound_messages_total{message_type="create_session_request

# IP pool utilization (for /24 subnet)
(address_registry_count / 254) * 100

# P95 session setup latency
histogram_quantile(0.95,
rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m]))

# Error rate
rate(s5s8_inbound_errors_total[5m])

# PCRF latency
histogram_quantile(0.95, rate(gx_outbound_transaction_duration_bucket

# PFCP association status
pfcp_peer_associated
```

Common Log Filters

Grep these strings in host logs (`journalctl -u omnipgw_c` or the configured log file). For session lifecycle events, `GET /api/session_history[?type=&search=]` provides the same information in structured form.

Filter	Purpose
IMSI	Find all logs for specific subscriber
"create_session"	Session establishment flow
"delete_session"	Session teardown flow
"Credit Control"	Gx PCRF interactions
"PFCP Session"	User plane programming
"error"	All error messages
"timeout"	Timeout issues
"Association"	PFCP association events

Health Check Commands

```
# Check service status
systemctl status omnipgw_c

# Check OAM API health (HTTPS/TLS)
curl -k https://<omnipgw_ip>:8443/api/status

# Check metrics endpoint
curl http://<omnipgw_ip>:9090/metrics

# Check active sessions
curl http://<omnipgw_ip>:9090/metrics | grep teid_registry_count

# Check PFCP association
curl http://<omnipgw_ip>:9090/metrics | grep pfcp_peer_associated

# Check IP pool usage
curl http://<omnipgw_ip>:9090/metrics | grep
address_registry_count
```

Related Documentation

- **Monitoring Guide** - Prometheus metrics, Grafana dashboards, alerting
- **Configuration Guide** - System configuration reference
- **Session Management** - Session lifecycle details
- **PFCP Interface** - PFCP troubleshooting details
- **Diameter Gx Interface** - Gx policy troubleshooting
- **Diameter Gy Interface** - Gy charging troubleshooting
- **QoS & Bearer Management** - QoS-related issues

Back to Operations Guide

OmniPGW Troubleshooting Guide - *by Omnitouch Network Services*

UE IP Pool Allocation Documentation

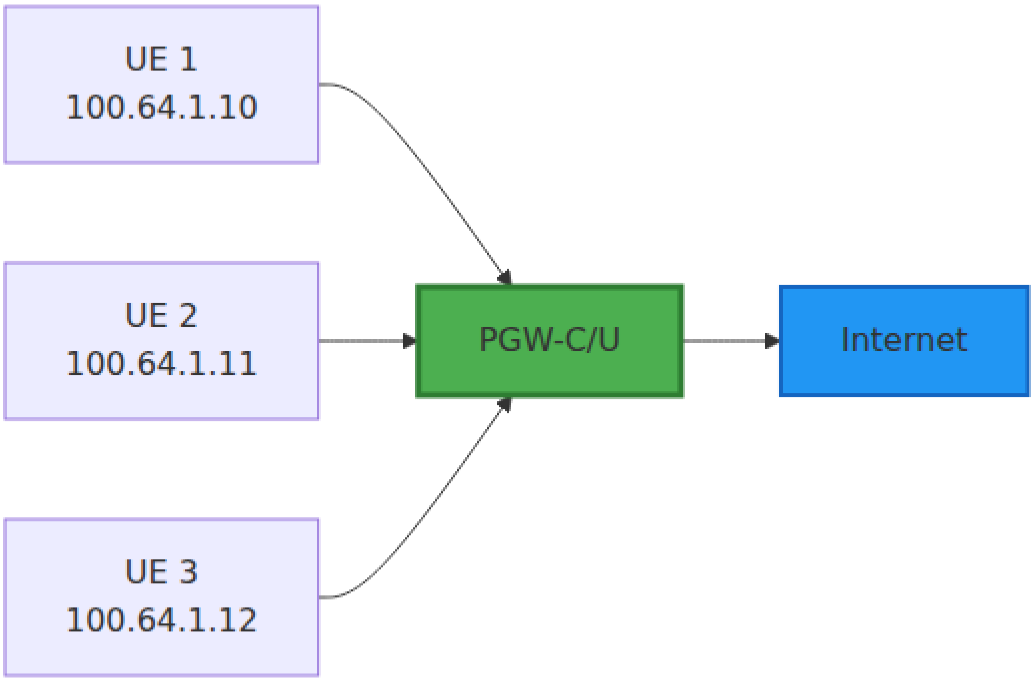
IP Address Management for Mobile Devices

Table of Contents

Overview

The PGW-C allocates IP addresses to UE (User Equipment) devices when they establish PDN (Packet Data Network) connections. This is a critical function that enables mobile devices to communicate with external networks.

Why IP Allocation Matters



Each UE receives a **unique IP address** from the PGW-C that:

- Identifies the device on the network
- Routes traffic to/from the device
- Enables charging and policy enforcement
- Persists for the duration of the PDN connection

Supported IP Versions

IP Version	Support	Description
IPv4	□ Full	Standard IPv4 addresses
IPv6	□ Full	IPv6 addresses and prefixes
IPv4v6	□ Full	Dual-stack (both IPv4 and IPv6)

IP Allocation Concepts

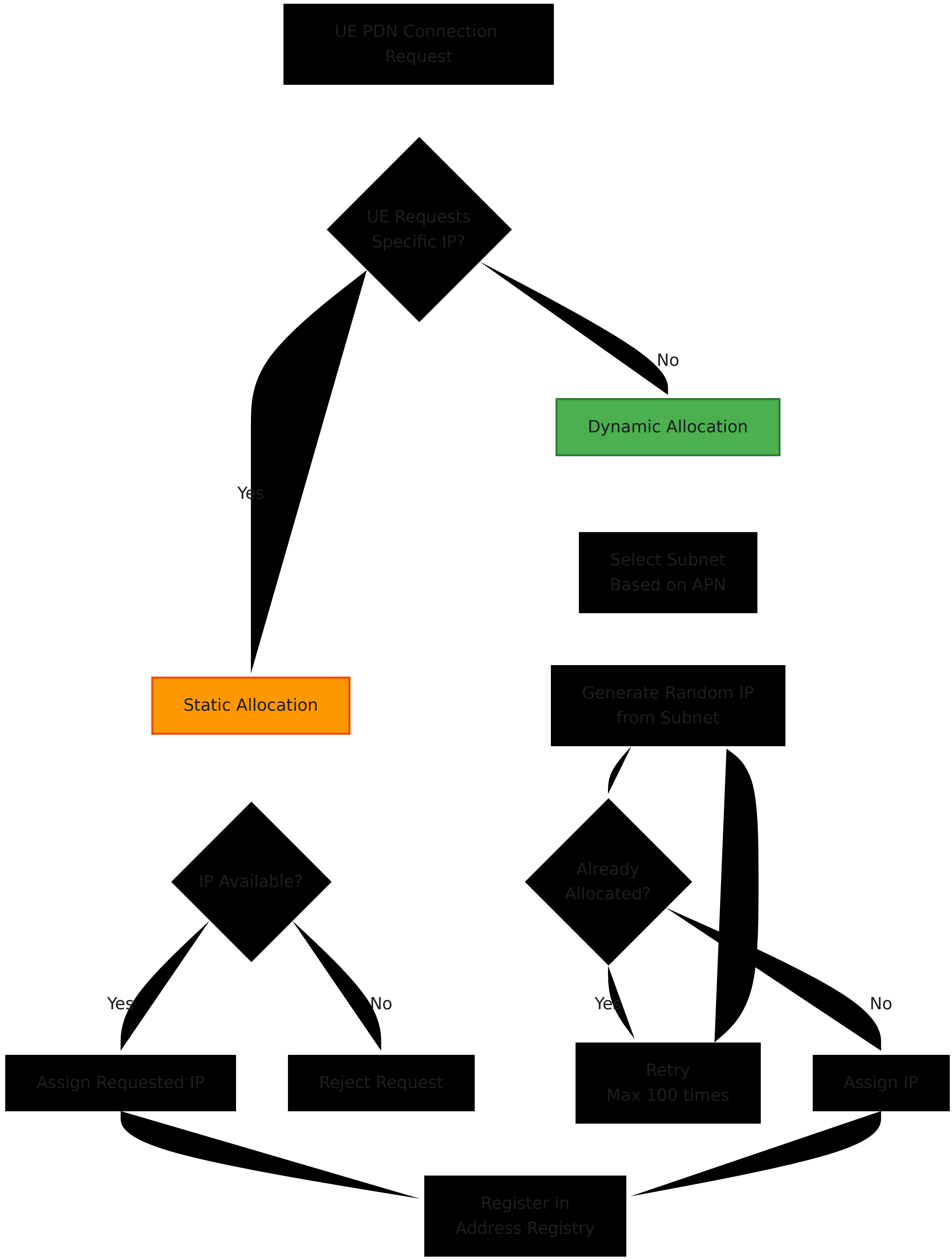
PDN Type

When a UE requests a PDN connection, it specifies a **PDN Type**:

PDN Type	Description	Allocated Addresses
IPv4	IPv4-only connection	Single IPv4 address
IPv6	IPv6-only connection	IPv6 prefix (e.g., /64)
IPv4v6	Dual-stack connection	Both IPv4 address and IPv6 prefix

Allocation Methods

PGW-C supports two IP allocation methods:



1. Dynamic Allocation (Most Common):

- PGW-C selects IP from configured pool
- Random selection to avoid predictability

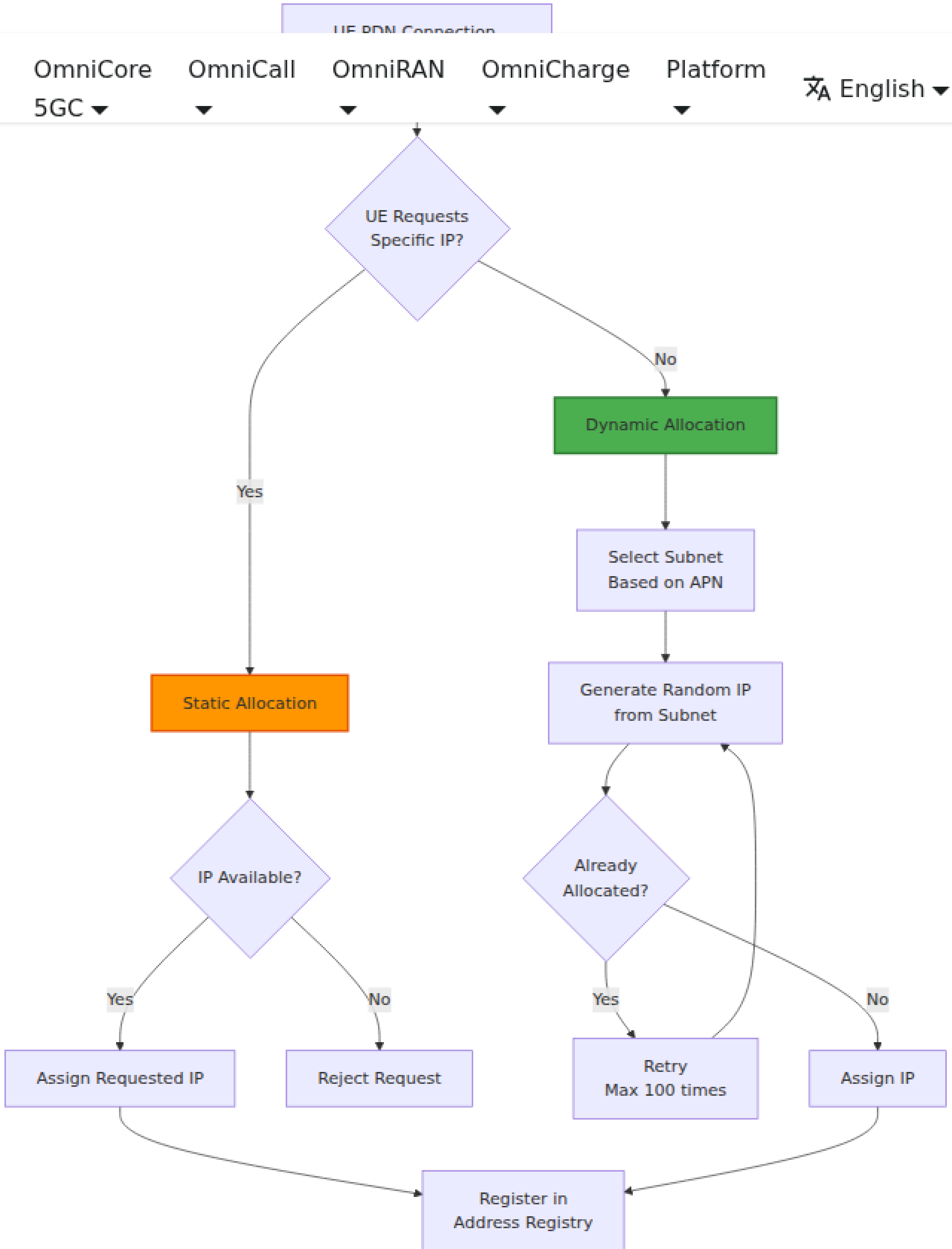
- Collision detection ensures uniqueness

2. Static Allocation:

- UE requests specific IP in GTP-C message
- PGW-C validates availability
- Useful for enterprise devices with fixed IPs

APN-Based Subnet Selection

Different **APNs (Access Point Names)** can use different IP pools:



Benefits:

- **Traffic Segregation** - Different APNs route to different networks
- **Policy Differentiation** - Apply different policies per APN

- **Capacity Planning** - Size pools based on expected usage
- **Billing** - Track usage by service type

Address Registry

The **Address Registry** tracks allocated IPs:

Function	Description
Registration	Maps UE IP → Session Process PID
Lookup	Find session by UE IP
Deregistration	Release IP when session ends
Collision Detection	Prevent duplicate allocations

Configuration

Basic Configuration

Edit `config/runtime.exs`:

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # APN "internet" uses two subnets
      "internet" => [
        "100.64.1.0/24",    # 254 usable IPs
        "100.64.2.0/24"     # 254 usable IPs
      ],

      # APN "ims" uses one subnet
      "ims" => [
        "100.64.10.0/24"
      ],

      # Default pool for unknown APNs
      default: [
        "42.42.42.0/24"
      ]
    }
  }
```

Regex Pattern Matching for APNs

For scenarios where multiple APNs share the same subnet pool, you can use **regex patterns** instead of exact APN names. This is useful for wildcard APN matching.

Pattern Rules:

- Keys starting with `^` are treated as regex patterns
- Keys without `^` are matched exactly (backwards compatible)
- Patterns are evaluated in order - first match wins
- Falls back to `default` if no pattern matches

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # Regex: APNs starting with "ims" (e.g., "ims", "ims.apn",
"ims.something.else")
      "^ims" => [
        "100.64.10.0/24"
      ],

      # Regex: APNs starting with "m2m." (e.g., "m2m.test",
"m2m.prod")
      "^m2m\." => [
        "100.64.20.0/24"
      ],

      # Exact match only - "enterprise.corp" exactly
      "enterprise.corp" => [
        "10.100.0.0/16"
      ],

      # Default pool for unmatched APNs
      default: [
        "42.42.42.0/24"
      ]
    }
  }
```

Important Notes:

- Regex patterns use standard Elixir/Erlang regex syntax
- Backslashes must be escaped in Elixir strings (use `\` for `\`)
- The `^` at the start is required to indicate a regex pattern
- Exact matches and regex patterns can be mixed in the same config
- Order matters for regex patterns - place more specific patterns first

Common Pattern Examples:

Pattern Type	Regex Key	Matches
Starts with	`"^ims"`	`ims`, `ims.apn`, `ims.anything`
Ends with	`"^.*\.corp$"`	`foo.corp`, `bar.corp`
Contains	`"^.*test.*"`	`test`, `foo.test.bar`, `testing`
Exact (with dots)	`"^internet\.apn$"`	`internet.apn` only

Suffix Matching Example:

To match APNs ending with a specific suffix (like `.corp`), use `^.*\.suffix$`:

```
subnet_map: %{
  # Match APNs ending with ".corp"
  "^.*\.corp$" => ["10.100.0.0/16"],

  # Match APNs ending with ".iot"
  "^.*\.iot$" => ["10.200.0.0/16"],

  default: ["42.42.42.0/24"]
}
```

Example Pattern Matching:

APN Request	Matched Key	Pool Used
`ims`	`^ims`	100.64.10.0/24
`ims.apn`	`^ims`	100.64.10.0/24
`ims.something.else`	`^ims`	100.64.10.0/24
`m2m.test`	`^m2m\.`	100.64.20.0/24
`m2m`	*default*	42.42.42.0/24
`enterprise.corp`	`enterprise.corp`	10.100.0.0/16
`foo.corp`	`^.*\.corp$`	10.100.0.0/16
`unknown.apn`	*default*	42.42.42.0/24

Subnet Notation

CIDR Notation: `<network>/<prefix_length>`

CIDR	Usable IPs	Example Range
/24	254	100.64.1.1 - 100.64.1.254
/23	510	100.64.0.1 - 100.64.1.254
/22	1022	100.64.0.1 - 100.64.3.254
/20	4094	100.64.0.1 - 100.64.15.254
/16	65534	100.64.0.1 - 100.64.255.254

Notes:

- Network address (e.g., 100.64.1.0) is not allocated
- Broadcast address (e.g., 100.64.1.255) is not allocated
- PGW-C allocates from `<network> + 1` to `<broadcast> - 1`

Multiple Subnets per APN

Load Balancing Across Subnets:

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => [
        "100.64.1.0/24",
        "100.64.2.0/24",
        "100.64.3.0/24",
        "100.64.4.0/24"
      ]
    }
  }
```

Selection Method:

- PGW-C randomly selects one subnet from the list
- Provides basic load balancing
- Each session independently selects a subnet

Benefits:

- Distribute load across multiple subnets
- Easier capacity expansion (add new subnets)
- Flexibility for routing policies

Real-World Example

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # General internet access
      "internet" => [
        "100.64.0.0/20"     # 4094 IPs for general use
      ],

      # IMS (Voice over LTE)
      "ims" => [
        "100.64.16.0/22"    # 1022 IPs for IMS
      ],

      # Enterprise APN
      "enterprise.corp" => [
        "10.100.0.0/16"     # 65534 IPs for enterprise
      ],

      # IoT devices (low bitrate)
      "iot.m2m" => [
        "100.64.20.0/22"    # 1022 IPs for IoT
      ],

      # Default fallback
      default: [
        "42.42.42.0/24"     # 254 IPs for unknown APNs
      ]
    }
  }
```

IPv6 Configuration

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => [
        # IPv4 pools
        "100.64.1.0/24"
      ],
      "internet.ipv6" => [
        # IPv6 pools (prefix delegation)
        "2001:db8:1::/48"
      ],
      default: [
        "42.42.42.0/24"
      ]
    }
  }
```

IPv6 Prefix Delegation:

- UE typically receives a /64 prefix
- Allows UE to assign multiple IPs (e.g., for tethering)
- Example: UE receives `2001:db8:1:a::/64`

Dual-Stack (IPv4v6) Configuration

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => [
        "100.64.1.0/24",        # IPv4 pool
        "2001:db8:1::/48"       # IPv6 pool (will be used for IPv6
allocation)
      ]
    }
  }
```

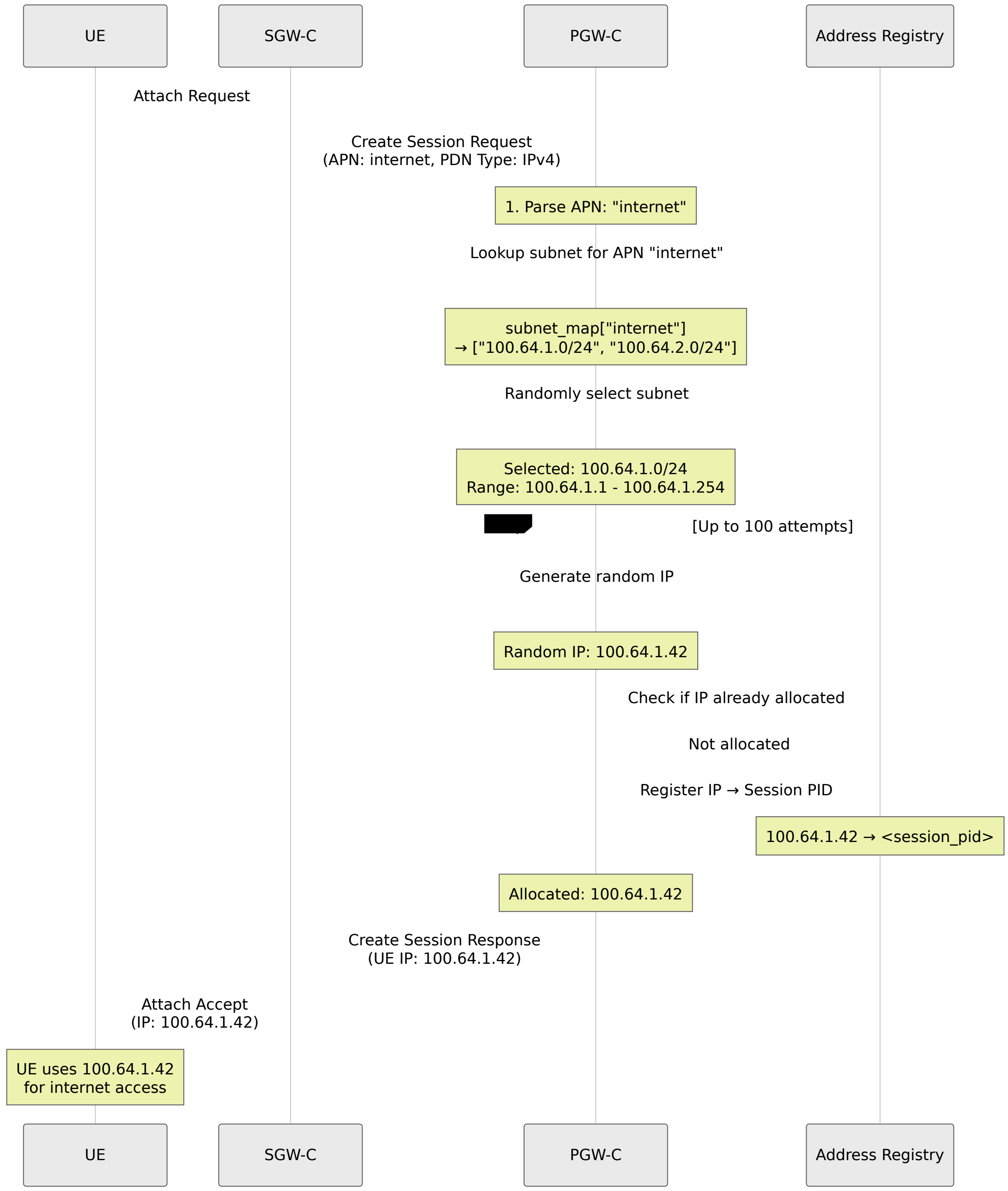
Dual-Stack Allocation:

- UE requests PDN Type: IPv4v6
- PGW-C allocates both IPv4 address and IPv6 prefix
- Both addresses active simultaneously

Allocation Process

IP allocation occurs during session creation when PGW-C receives a Create Session Request via the S5/S8 interface. See S5/S8 Interface for GTP-C message details and Session Management for session lifecycle.

Step-by-Step: Dynamic IPv4 Allocation



How It Works

Dynamic Allocation Process:

1. **Subnet Lookup:** System retrieves configured subnets for the requested APN

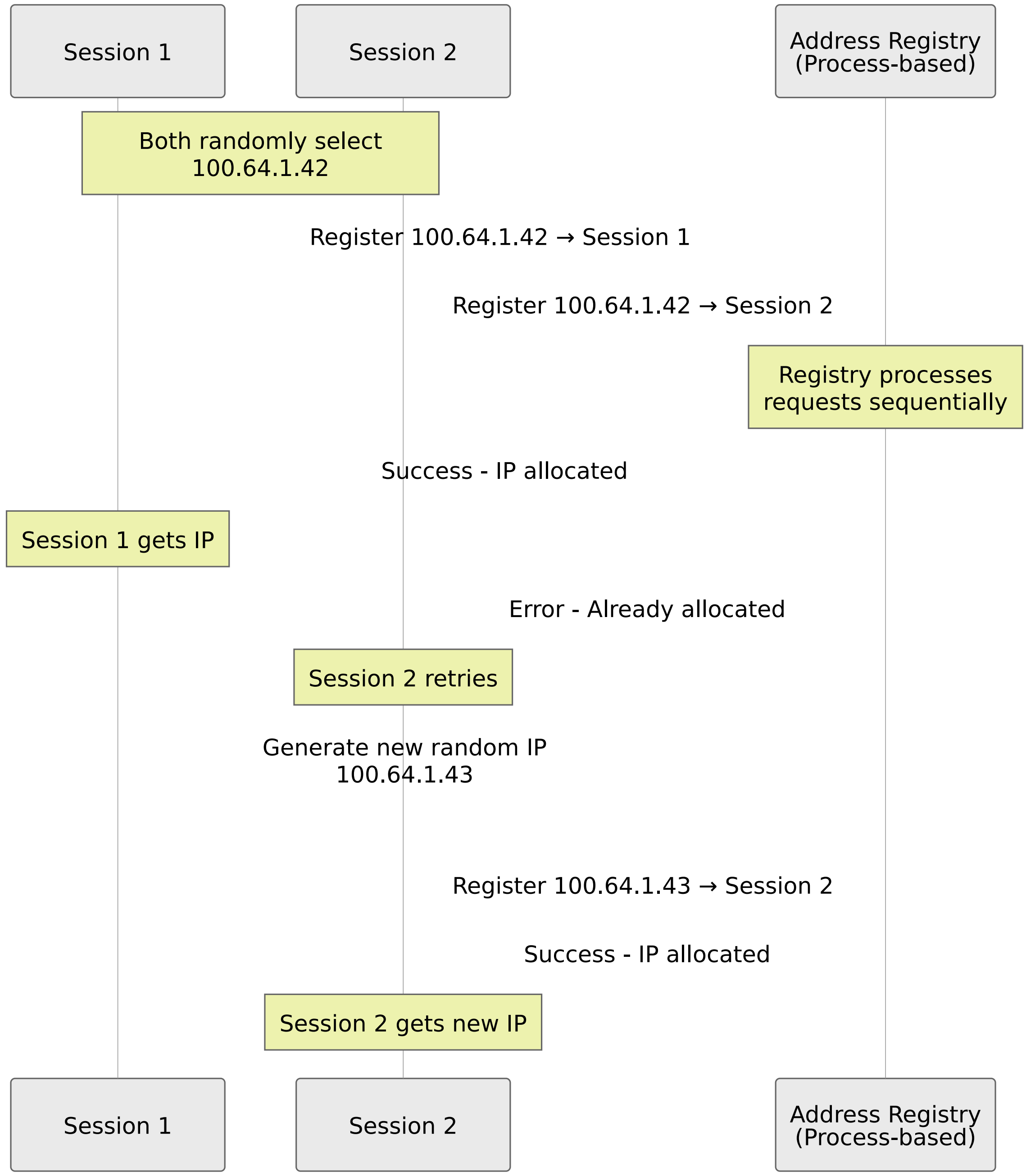
2. **Random Selection:** One subnet is randomly selected from the available list
3. **IP Generation:** A random IP is generated within the subnet range
4. **Uniqueness Check:** System verifies the IP hasn't been allocated
5. **Retry Logic:** If collision detected, retry up to 100 times with new random IP
6. **Registration:** Once unique IP found, it's registered to the session

Key Design Points:

- **Max 100 attempts:** Prevents infinite loops when pool is nearly exhausted
- **Random selection:** Avoids predictable IP assignment patterns for security
- **Atomic operations:** Process-based registry ensures no duplicate allocations
- **Fallback to default:** If APN not found in config, uses default pool

Collision Handling

Scenario: Two sessions try to allocate same IP simultaneously



How Collision Prevention Works:

- Registry processes requests one at a time (serialized)
- No race conditions possible
- First request to register an IP succeeds
- Subsequent requests for same IP are rejected
- Rejected sessions automatically retry with new random IP

Default Subnet Fallback

Scenario: UE requests unknown APN

Example Configuration:

```
# Config
subnet_map: %{
  "internet" => ["100.64.1.0/24"],
  default: ["42.42.42.0/24"]
}
```

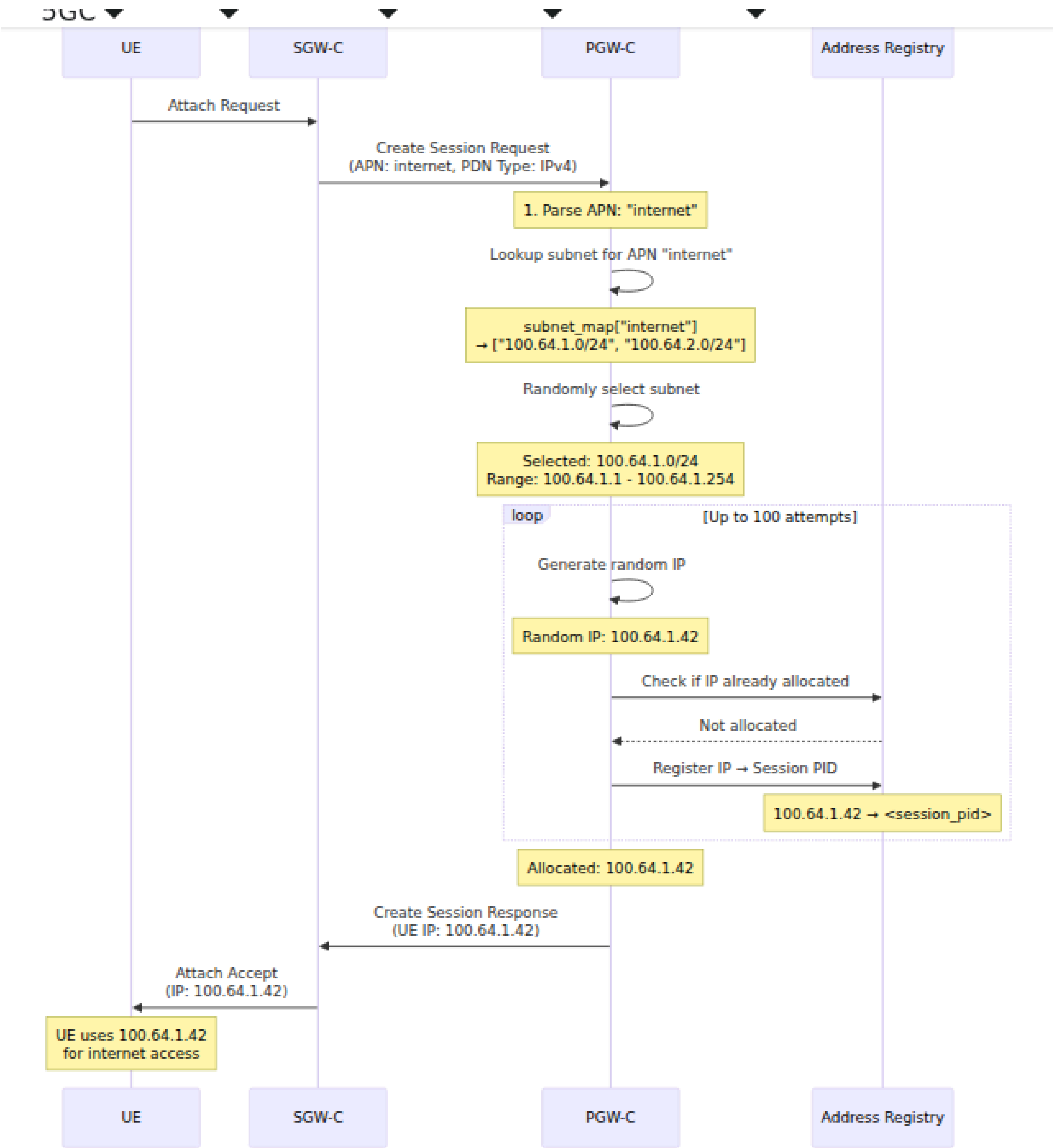
Behavior:

- UE requests APN: "unknown.apn"
- System looks for "unknown.apn" in subnet_map
- Not found, so falls back to default pool
- Allocates IP from 42.42.42.0/24

Fallback Logic:

1. First, try to find APN-specific pool in configuration
2. If not found, use the `default` pool
3. If no default configured, allocation fails

Deallocation on Session Termination



Automatic Cleanup:

- When session process terminates, registry cleans up
- IP immediately available for new allocations
- No manual intervention required

Advanced Topics

Pool Exhaustion

Scenario: All IPs in pool are allocated

```
Pool: 100.64.1.0/24 (254 usable IPs)
Allocated: 254 IPs
New request arrives → Exhaustion
```

What Happens:

1. PGW-C attempts 100 random allocations
2. All attempts find IP already allocated
3. Returns: `{:error, :ue_ip_address_allocation_failed}`
4. Session establishment fails
5. SGW-C receives error response

Prevention:

```
# Monitor pool utilization
address_registry_count / total_pool_size > 0.8  # Alert at 80%

# Expand pool before exhaustion
"internet" => [
  "100.64.1.0/24",
  "100.64.2.0/24",  # Add additional subnet
  "100.64.3.0/24"
]
```

Static IP Allocation

Use Case: Enterprise device needs fixed IP

GTP-C Message Format:

```
Create Session Request
├── IMSI: 310260123456789
├── APN: enterprise.corp
├── PDN Address Allocation (IE)
│   └── PDN Type: IPv4
│   └── IPv4 Address: 10.100.0.50  ← UE requests specific IP
```

OmniPGW Processing:

1. **Extract Requested IP:** Parse PDN Address Allocation IE from request
2. **Validate IP:** Check if requested IP is in configured pool for this APN
3. **Check Availability:** Verify IP is not already allocated to another session
4. **Allocate or Reject:**
    - If available: Allocate requested IP to this session
    - If unavailable: Reject session with appropriate cause code

Possible Results:

- **Success:** UE receives exactly the IP address it requested
- **Failure (IP in use):** Session rejected - IP already allocated
- **Failure (IP not in pool):** Session rejected - IP not in configured range

IPv6 Prefix Delegation

UE requests IPv6:

```
Create Session Request
├── PDN Type: IPv6
```

PGW-C allocates /64 prefix:

```
Allocated Prefix: 2001:db8:1:a::/64

UE can use:
- 2001:db8:1:a::1
- 2001:db8:1:a::2
- ... (18 quintillion addresses)
```

Benefits:

- UE can assign multiple IPs (e.g., tethering)
- Supports SLAAC (Stateless Address Autoconfiguration)
- Eliminates NAT requirement

Dual-Stack Allocation

UE requests IPv4v6:

```
Create Session Request
├── PDN Type: IPv4v6
```

PGW-C allocates both:

```
IPv4: 100.64.1.42
IPv6: 2001:db8:1:a::/64
```

Traffic Handling:

- IPv4 traffic uses IPv4 address
- IPv6 traffic uses IPv6 prefix
- Both active simultaneously
- Separate GTP tunnels (or dual-stack tunnel)

When the APN has no IPv6 pool:

If an IPv4v6 PDN is requested but the matched APN has only an IPv4 CIDR (no IPv6 range), the IPv6 allocator returns `{:error, :no_ipv6_pool}` instead of crashing, and the session procedure decides by APN type:

APN type	Result
Emergency (SOS) APN (name contains `"sos"`)	**Degrade to IPv4-only** — keep the IPv4 address, clear the IPv6 address, and set the negotiated PDN type to IPv4. The PAA, PFCP and Gx all come out IPv4-only, and the P-CSCF is still returned in the PCO so emergency IMS registration can proceed (TS 29.274 / TS 23.401 single-address-bearer / PDN-type downgrade).
Regular APN	**Reject** — the request is not silently downgraded; the misconfiguration (or unsupported request) is surfaced.

IPv6-only requests against an APN with no IPv6 pool are likewise rejected. See IPv4v6 request when the APN has no IPv6 pool in the IPv6 / Dual-Stack guide for the full behaviour table.

Private vs. Public IP Addresses

Private IP Pools (RFC 1918):

```
# Not routable on public internet
subnet_map: %{
  "internet" => [
    "10.0.0.0/8",
    "172.16.0.0/12",
    "192.168.0.0/16"
  ]
}
```

Requires NAT at PGW-U to access internet

Public IP Pools:

```
# Routable public IPs (example only)
subnet_map: %{
  "internet" => [
    "203.0.113.0/24"  # Public IP block
  ]
}
```

No NAT required - direct internet routing

Recommendation:

- Use **private IPs (RFC 6598)**: `100.64.0.0/10` (Carrier-Grade NAT)
- Reserve public IPs for special services only

Monitoring

IP Pool Management

IP pool allocation and utilization are monitored through the OAM REST API (HTTPS/TLS on port `8443`, routes under the `/api` prefix). Interactive documentation (Swagger UI) is available at `https://<omnipgw-ip>:8443/api/docs`.

```
GET /api/ip_pools            # all pools + utilization
GET /api/ip_pools/<name>     # one pool by name
```

Example:

```
curl -k https://<omnipgw-ip>:8443/api/ip_pools
curl -k https://<omnipgw-ip>:8443/api/ip_pools/Internet
```

Per-Pool Information returned for each configured pool:

Field	Description
Name	Pool / APN identifier (e.g., "default", "ims.something.else", "Internet")
Total	Number of IPs in the pool
Used	Currently allocated IPs
Free	Remaining IPs available for allocation
Utilization %	Percentage of the pool currently allocated

Use Cases:

- Quick capacity check before maintenance
- Identify pools approaching exhaustion
- Verify pool configuration
- Monitor allocation patterns by APN

Key Metrics

Address Registry Count:

```
# Current allocated IPs
address_registry_count

# Pool utilization (requires calculation)
address_registry_count / <total_pool_size> * 100
```

Example:

```
Pool: 100.64.1.0/24 (254 IPs)
Allocated: 150 IPs
Utilization: 150 / 254 = 59%
```

Alerts

```yaml
# Alert on high pool utilization
- alert: UEIPPoolUtilizationHigh
  expr: address_registry_count > 200  # For /24 pool
  for: 10m
  annotations:
    summary: "UE IP pool utilization above 80%"
    description: "Current: {{ $value }} / 254 IPs allocated"

# Alert on pool exhaustion
- alert: UEIPPoolExhausted
  expr: address_registry_count >= 254  # For /24 pool
  for: 1m
  annotations:
    summary: "UE IP pool exhausted - no IPs available"

# Alert on allocation failures
- alert: UEIPAllocationFailures
  expr: rate(ue_ip_allocation_failures_total[5m]) > 0
  for: 5m
  annotations:
    summary: "UE IP allocation failures occurring"
```

Grafana Dashboard

Panel 1: IP Pool Utilization

```
# Gauge showing percentage
(address_registry_count / 254) * 100
```

Panel 2: Allocated IPs Over Time

```
# Time series
address_registry_count
```

Panel 3: Allocation Rate

```
# Rate of new allocations
rate(address_registry_count[5m])
```

Panel 4: Pool Exhaustion Risk

```
# Days until exhaustion (based on current rate)
(254 - address_registry_count) / rate(address_registry_count[1h])
```

Troubleshooting

Issue 1: Session Establishment Fails (No IP Available)

Symptoms:

- Create Session Response: Cause "Request rejected"
- Log: "UE IP address allocation failed"

Possible Causes:

1. **Pool Exhausted**

   ```
   # Check current allocation
   curl http://<pgw_c_ip>:42069/metrics | grep
   address_registry_count
   ```

2. **Configuration Error**

```
# Verify subnet configuration
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => [
        "100.64.1.0/24"  # Ensure valid CIDR
      ]
    }
  }
```

3. **APN Misconfiguration**

```
# If APN not found, falls back to default
# Ensure default pool exists
subnet_map: %{
  default: ["42.42.42.0/24"]
}
```

Resolution:

- **Expand pool:** Add more subnets
- **Cleanup stale sessions:** Restart PGW-C to release leaked IPs
- **Verify config:** Check `runtime.exs` for typos

Issue 2: IP Address Collision

Symptoms:

- Two UEs receive same IP (very rare)
- Routing issues

Cause:

- Bug in Address Registry (should not happen)

Debug:

```
# Check for duplicate IPs in logs
grep "already_registered" /var/log/pgw_c.log
```

Resolution:

- Should self-correct (second session retries)
- If persistent, report bug

Issue 3: Wrong IP Pool Used

Symptoms:

- UE receives IP from unexpected subnet
- APN "internet" gets IP from "ims" pool

Cause:

- Incorrect subnet_map configuration

Verify:

```
# Check exact APN string matching
subnet_map: %{
  "internet" => [...],    # Case-sensitive
  "Internet" => [...]     # Different APN!
}
```

Resolution:

- Ensure APN names match exactly (case-sensitive)
- Use default pool for catch-all

Issue 4: IPv6 Allocation Fails

Symptoms:

- UE requests IPv6, receives error

- A UE requests an **IPv4v6** PDN against an APN with no IPv6 range and either gets an **IPv4-only** session (SOS APN) or has the session **rejected** (regular APN)

Possible Causes:

1. **No IPv6 pool configured**

   ```
   # Missing IPv6 subnets
   subnet_map: %{
     "internet" => [
       "100.64.1.0/24"  # Only IPv4
     ]
   }
   ```

2. **Invalid IPv6 prefix**

   ```
   # Too small prefix (should be /48 or larger)
   "internet" => [
     "2001:db8::/128"  # Wrong - no room for allocation
   ]
   ```

IPv4v6 with no IPv6 pool (degrade vs. reject):

When an **IPv4v6** request hits an APN with no IPv6 range, `register_new_ipv6/1` returns `{:error, :no_ipv6_pool}` (it does **not** crash). The session then:

- **Emergency (SOS) APN** (name contains `"sos"`): degrades to **IPv4-only** — assigns the IPv4 address, clears the IPv6 address, sets the negotiated PDN type to IPv4, and still returns the P-CSCF in the PCO. Logged as a `warning`.
- **Regular APN**: **rejects** with `{:error, :no_ipv6_pool}` — no silent downgrade. Logged as an `error`.

See IPv4v6 request when the APN has no IPv6 pool.

Resolution:

```
# Add IPv6 pool (for a regular APN that should offer dual-stack)
subnet_map: %{
  "internet" => [
    "100.64.1.0/24",
    "2001:db8:1::/48"  # IPv6 pool
  ]
}
```

For an emergency (SOS) APN on an IPv4-only deployment, the IPv4-only degrade is intentional — add an IPv6 CIDR only if SOS PDNs should be dual-stack.

Issue 5: High Pool Utilization

Symptoms:

- Nearing pool exhaustion
- `address_registry_count` approaching max

Proactive Measures:

1. **Add Subnets:**

```
"internet" => [
  "100.64.1.0/24",   # Existing
  "100.64.2.0/24",   # New subnet (adds 254 IPs)
  "100.64.3.0/24"    # New subnet (adds 254 IPs)
]
```

2. **Use Larger Subnets:**

```
# Replace /24 with /22
"internet" => [
  "100.64.0.0/22"  # 1022 usable IPs
]
```

3. **Session Cleanup:**

    - Monitor stale sessions
    - Ensure proper Delete Session Request handling

Best Practices

Capacity Planning

Calculate required pool size:

```
Expected concurrent users: 10,000
Peak concurrency: 30% (3,000 simultaneous sessions)
Growth buffer: 50%
Required IPs: 3,000 * 1.5 = 4,500 IPs

Subnet: /20 (4,094 usable IPs) - Too small
Subnet: /19 (8,190 usable IPs) - Sufficient
```

Subnet Selection

Recommended:

- Use `100.64.0.0/10` (RFC 6598 - Carrier-Grade NAT)
- Provides 4 million IPs
- Reserved for service provider NAT

Avoid:

- Public IPs (expensive, limited)
- Common private ranges that conflict with enterprise VPNs

Configuration Layout

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # Primary internet APN - large pool
      "internet" => [
        "100.64.0.0/18"  # 16,382 IPs
      ],

      # IMS - smaller dedicated pool
      "ims" => [
        "100.64.64.0/22"  # 1,022 IPs
      ],

      # Enterprise - medium pool
      "enterprise.corp" => [
        "100.64.68.0/22"  # 1,022 IPs
      ],

      # IoT - large pool for many devices
      "iot.m2m" => [
        "100.64.72.0/20"  # 4,094 IPs
      ],

      # Default - small fallback
      default: [
        "100.64.127.0/24"  # 254 IPs
      ]
    }
  }
```

Related Documentation

Configuration

- **Configuration Guide** - UE IP pool configuration, APN subnet mapping
- **PCO Configuration** - DNS, P-CSCF, MTU delivered with IP address

- **Session Management** - Session lifecycle, IP allocation during PDN setup
- **PFCP Interface** - UE address assignment via PFCP to UPF

Network Planning

- **S5/S8 Interface** - IP address delivery via GTP-C
- **Diameter Gx Interface** - Policy control for IP allocation

Operations

- **Monitoring Guide** - IP pool utilization metrics, allocation tracking
- **Data CDR Format** - UE IP addresses in CDRs for billing correlation

---

Back to Operations Guide