OmniPGW □作指南

OmniPGW - 数据包网关控制平面 **(PGW-C)**

□ Omnitouch □□□□□□

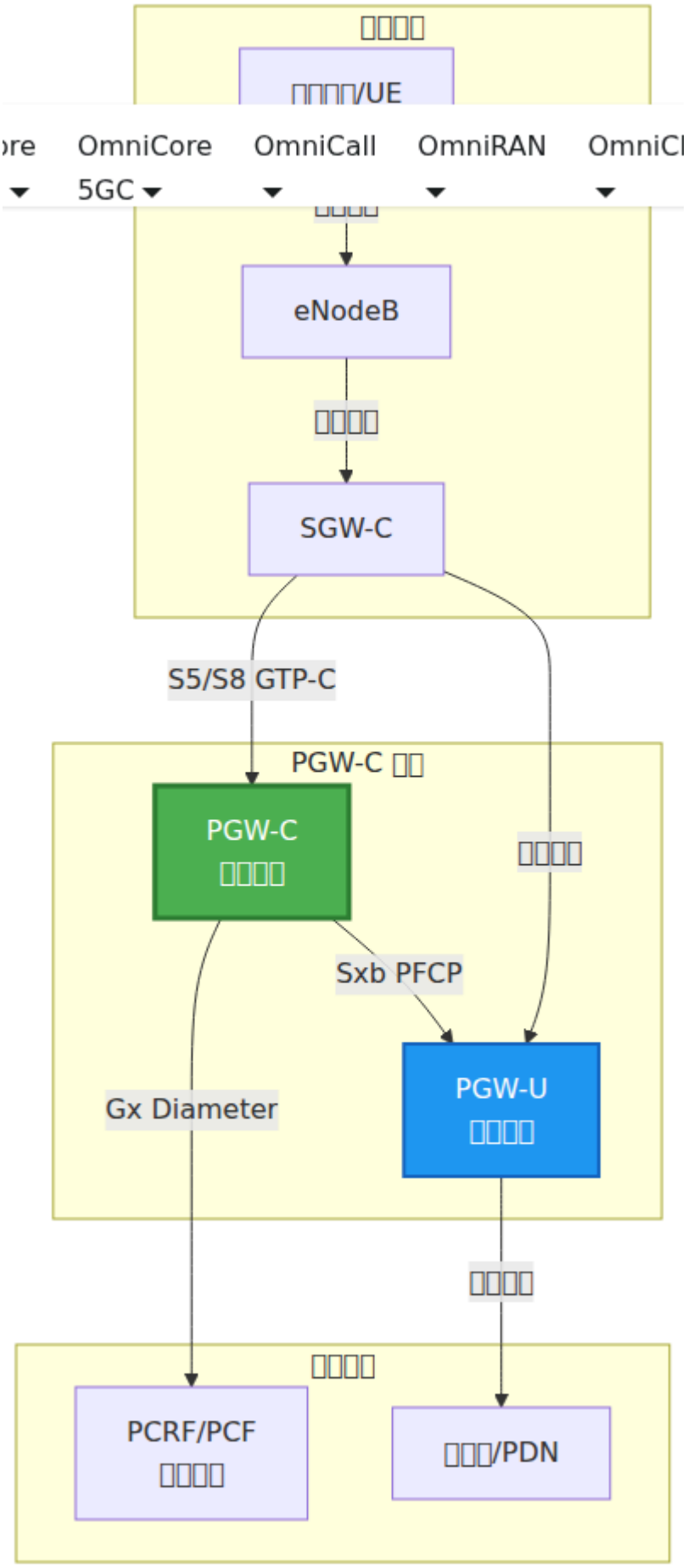
目录

1. □述
2. 架构
3. 网络接口
4. 关键□念
5. 快速入门
6. 配置
7. 实时操作监控 (OAM API)
8. 监控与指标
9. 详细文档
10. 附加资源

概述

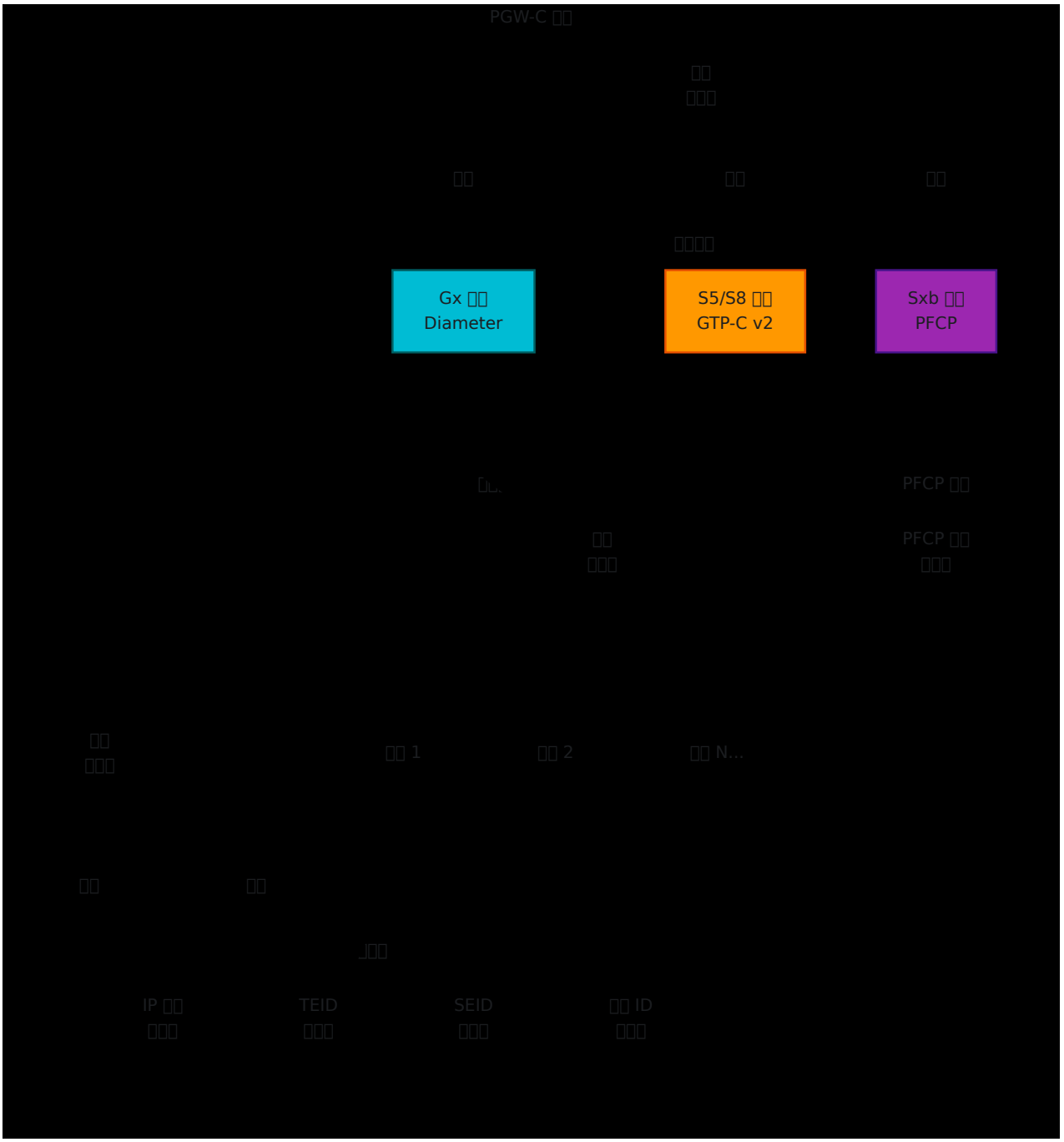
OmniPGW 是一个高性能的数据包网关控制平面 (PGW-C) 实现，用于 3GPP LTE 演进包核心 (EPC) 网络，由 Omnitouch 网络服务开发。它管理数据会话的控制平面功能，包括：

- 会话管理 - 创建、修改和终止 UE (用户设备) 数据会话
- **IP** 地址分配 - 从配置的池中为移动设备分配 IP 地址
- 策略与计费控制 - 与 PCRF 接口进行策略执行和计费
- 用户平面协调 - 控制 PGW-U (用户平面) 进行数据包转发


□□□□
□□□□/UE
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCl
□□□□
eNodeB
□□□□
SGW-C
S5/S8 GTP-C
PGW-C □□
PGW-C
□□□□
□□□□
Sxb PFCP
Gx Diameter
PGW-U
□□□□
□□□□
□□□□
PCRF/PCF
□□□□
□□□/PDN

PGW-C 的功能

接受来自 SGW-C 的会话请求 通过 S5/S8 接口 (GTP-C)
从配置的子网池中分配 UE IP 地址
通过 Gx 接口 (Diameter) 向 PCRF 请求策略决策
通过 Sxb 接口 (PFCP) 在 PGW-U 中编程转发规则
通过承载上下文和 QoS 规则管理 QoS 执行
跟踪计费信息 以供计费系统使用



架构

组件概述

过程架构

PGW-C 基于 Elixir/OTP 构建，并使用监督进程架构：

- 应用监督器 - 管理所有组件的顶层监督器

协议代理 - 处理进出协议消息
会话进程 - 每个活动 PDN 连接一个 GenServer
注册表 - 跟踪分配的资源 (IP、TEID、SEID 等)
PFCP 节点管理器 - 维护与 PGW-U 对等体的 PFCP 关联

每个组件都受到监督，并在故障时会自动重启，确保系统的可靠性。

网络接口

PGW-C 实现了三个主要的 3GPP 接口：

S5/S8 接口 (GTP-C v2)

目的： SGW-C 和 PGW-C 之间的控制平面信令

协议：基于 UDP 的 GTP-C 版本 2

关键消息：

创建会话请求/响应
删除会话请求/响应
创建承载请求/响应
删除承载请求/响应

配置：请参阅 S5/S8 配置

Sxb 接口 (PFCP)

目的： PGW-C 和 PGW-U 之间的控制平面信令

协议：基于 UDP 的 PFCP (数据包转发控制协议)

关键消息：

关联设置请求/响应
会话建立请求/响应

- 会话修改请求/响应
- 会话删除请求/响应
- 心跳请求/响应

配置：请参阅 PFCP/Sxb 接口文档

Gx ���□ (Diameter)

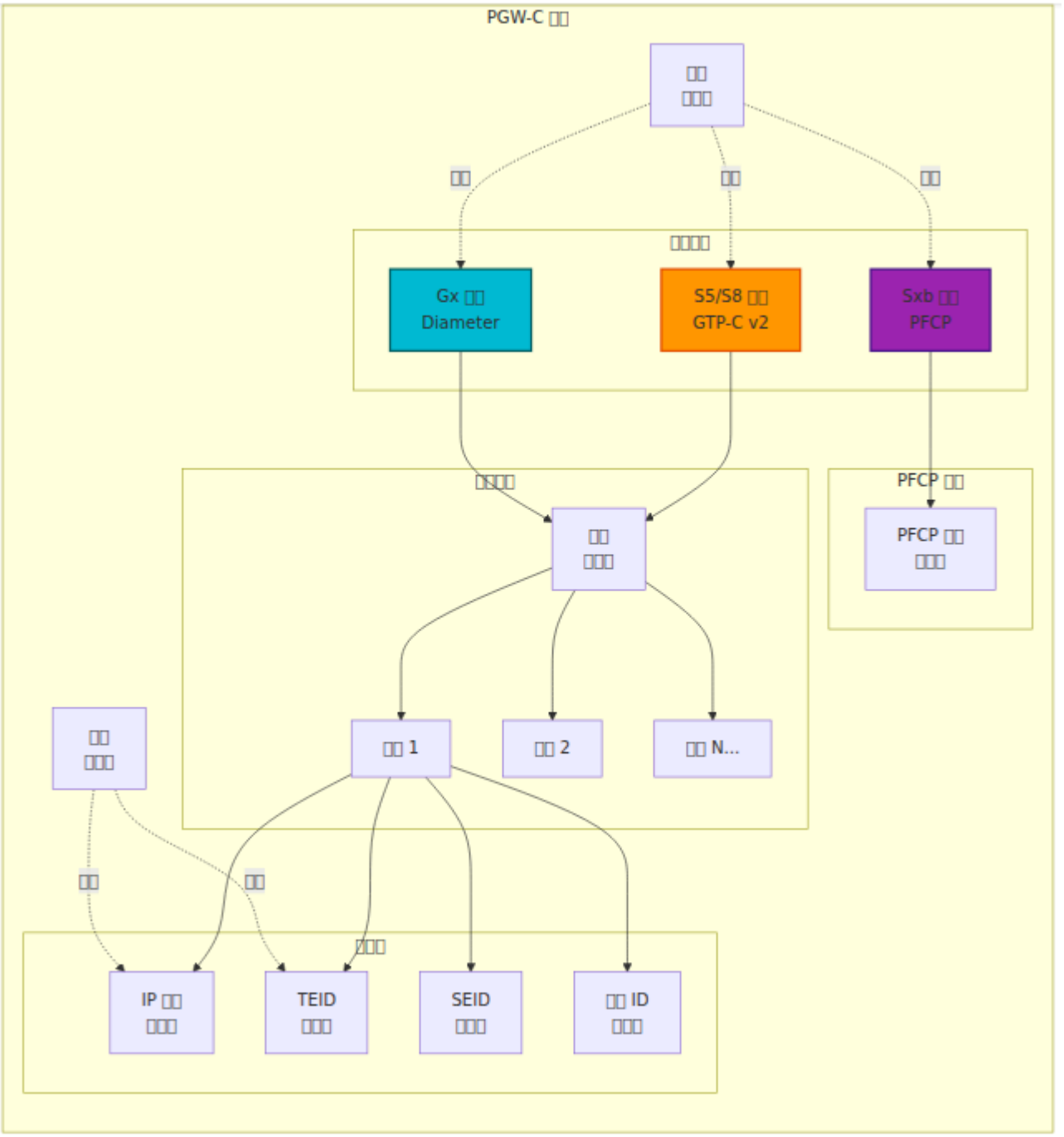
目的：策略与计费规则功能 (PCRF) 接口

协议： Diameter (IETF RFC 6733)

关键消息：

- 信用控制初始请求/应答 (CCR-I/CCA-I)
- 信用控制终止请求/应答 (CCR-T/CCA-T)

配置：请参阅 Diameter Gx 接口文档



关键概念
PDN 会话

PDN (数据包数据网络) 会话表示 UE 与外部网络（如互联网）的数据连接。每个会话具有： UE IP 地址 - 从配置的子网池中分配

APN (接入点名称) - 标识外部网络
承载上下文 - 包含 QoS 参数和隧道信息
计费 ID - 用于计费的唯一标识符
TEID (隧道端点 ID) - S5/S8 接口隧道标识符 SEID (会话端点 ID) - Sxb 接口会话标识符

承载上下文

承载表示具有特定 QoS 特性的流量流：

默认承载 - 每个 PDN 会话创建
专用承载 - 针对特定 QoS 需求的附加承载
EBI (EPS 承载 ID) - 每个承载的唯一标识符
QoS 参数 - QCI、ARP、比特率 (MBR、GBR)

PFCP 规则

PGW-C 向 PGW-U 编程数据包处理规则：

PDR (数据包检测规则) - 匹配数据包（上行/下行）
FAR (转发动作规则) - 指定转发行为
QER (QoS 执行规则) - 强制比特率限制
BAR (缓冲动作规则) - 控制数据包缓冲

请参阅 PFCP 接口文档 获取详细信息。

IP 地址分配

UE IP 地址从配置的子网池中分配：

基于 APN 的选择 - 不同的 APN 可以使用不同的子网
动态分配 - 从可用范围中随机选择 IP
静态分配 - 支持 UE 请求的 IP 地址
冲突检测 - 确保唯一的 IP 分配

请参阅 UE IP 池分配 以获取配置，以及 IPv6 / 双栈 以获取 IPv6 和 IPv4v6 PDN 操作。

快速入门

先决条件

- Elixir ~1.16
- Erlang/OTP 26+
- 与 SGW-C、PGW-U 和 PCRF 的网络连接
- 对 LTE EPC 架构的理解

启动 **OmniPGW**

1. 在 **`config/runtime.exs`** 中配置运行时设置
2. 编译应用程序：

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

3. 启动应用程序：

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

验证□作

检查日志以确认成功启动：

```
[info] □动 OmniPGW...
[info] 在 127.0.0.42:42069 □动指标导出器
[info] 在 127.0.0.10 □动 S5/S8 代理
[info] 在 127.0.0.20 □动 Sxb 代理
[info] □动 Gx 代理
[info] □动 PFCP 节点管理器
[info] OmniPGW 成功□动
```

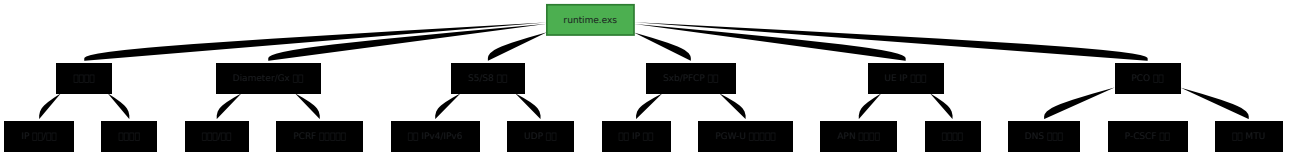
访问指标位于 `http://127.0.0.42:42069/metrics`（配置的地址）。

配置

所有运行时配置在 config/runtime.exs 中定义。此文件在应用程序启动时进行评估。以下是完整示例；每个部分在下面的参考表中描述，并在 配置参考 中详细说明。

```
# config/runtime.exs
import Config

# □□□□
config :logger, level: :info

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

  # Diameter/Gx (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-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 □□)
  s5s8: %{
    local_ipv4_address: "10.0.0.20"
  },

  # Sxb/PFCP (PGW-U □□)
  sxb: %{
```

```
    local_ip_address: "10.0.0.20"
  },

  # UPF 选择
  upf_selection: %{
    rules: [
      %{
        name: "默认互联网",
        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}  # 备用
        ]
      }
    ],
    fallback_pool: [
      %{remote_ip_address: "10.0.0.21", remote_port: 8805, weight:
100}
    ]
  },

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

  # 协议配置选项
```



```
  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
  }
```

配置分为几个部分：

配置概述

快速配置参考

部分	目的	文档
metrics	Prometheus 指标导出器	监控指南
diameter	Gx 接口到 PCRF	Diameter Gx 配置
s5s8	GTP-C 接口到 SGW-C	S5/S8 配置
sxb	PFCP 接口到 PGW-U	PFCP 配置
ue	UE IP 地址池	IP 池配置
pco	协议配置选项	PCO 配置
CDR	离线计费和使用报告	CDR 格式
charging	在线/离线计费模式，紧急覆盖	计费覆盖

有关详细信息，请参阅 完整配置指南。

实时□作监控 (OAM API)

OmniPGW 提供一个 **OAM REST API** 用于实时监控和操作，提供系统状态的即时可见性。根据需要查询这些读取端点（或在轮询间隔/从脚本中）以获取实时操作视图 - 无需对主机的命令行访问或原始指标查询。

访问 OAM API

API 通过 HTTPS/TLS 提供。所有路由都在 `/api` 前缀下提供。

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

交互式 API 文档 (Swagger UI) 可在以下地址访问:

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

示例健康检查（使用 `-k` 进行自签名实验室证书）：

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

可用端点:

目的	端点	替代（旧页面）
深入了解特定用户 (IMSI/MSISDN/IP)	`POST /api/ue_search`	UE 搜索
列出/搜索活动 PDN 会话	`GET /api/sessions[?search=<imsi|msisdn|ip>]`	PGW 会话
按 IMSI 获取一个会话	`GET /api/sessions/<imsi>`	PGW 会话（详细）
会话历史/审计事件	`GET /api/session_history[?type=&search=&page=&page_size=]`	会话历史
网络拓扑□览	`GET /api/topology`	网络拓扑
UE IP 地址池利用率	`GET /api/ip_pools`, `GET /api/ip_pools/<name>`	IP 池
UPF / PFCP 对等体状态	`GET /api/upf`, `GET /api/upf/<ip>`	PFCP 会话 / UPF 状态
UPF 选择规则、健康、PCO 配置	`GET /api/upf_selection`	UPF 选择
Diameter (Gx/Gy) 对等体连接性	`GET /api/diameter[?status=connected|disconnected]`, `GET /api/diameter/<origin-host>`	Diameter 对等体
每个会话的在线计费 (Gy) 状态	`GET /api/charging[?search=<imsi>]`, `GET /api/charging/<imsi>`	Gy 状态
紧急覆盖：禁用在线计费节点范围	`PUT /api/charging/override`, `GET /api/charging/override`, `DELETE /api/charging/override`	-
P-CSCF DNS 发现状态	`GET /api/pcscf_monitor`	P-CSCF 监控

目的	端点	替代（旧页面）
健康检查	`GET /api/status`	-

要获取实时仪表板样式的视图，请在间隔内轮询集合端点，例如：

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

> 注意：以前的 日志 页面（实时日志流）和 **Gy** 模拟器 页面没有 OAM API 等效项。从主机/您的日志聚合器流式传输日志，并使用 `GET /api/charging` 检查实时 Gy/在线计费状态。

关键特性

按需和轮询数据：

- 每个端点返回来自 OmniPGW 进程的当前实时状态
- 在间隔内使用 `watch`/cron/a 脚本进行轮询以获取自动刷新的视图
- 状态字段（例如对等体 `connected`/`associated`）在 JSON 中明确返回

搜索与过滤：

- 通过 `GET /api/sessions?search=...` 按 IMSI、IP、MSISDN 或 APN 搜索会话
- 通过 `?status=connected|disconnected` 过滤 Diameter 对等体
- 通过 `?search=<imsi>` 过滤计费状态

完整详细记录：

- 每个实体端点（`/api/sessions/<imsi>`、`/api/upf/<ip>`、`/api/diameter/<origin-host>`、`/api/charging/<imsi>`）返回完整状态作为 JSON
- 检查完整的会话、对等体配置和能力

访问控制：

- 通过 HTTPS/TLS 在 8443 端口提供
- 仅绑定到管理 IP，并使用您的访问控制进行前端处理

- 旨在用于 NOC/操作团队和自动化使用

作工作流程

会话故障排除（深入了解）：

```
1. 用户报告连接问题
2. POST /api/ue_search，JSON 主体包含 IMSI、MSISDN 或 IP
3. 查看响应中的综合会话详细信息：
   a) 活动会话 - 验证会话是否存在且参数正确
   b) 当前位置信息 - 检查 TAC、单元 ID、服务网络
   c) 承载信息 - 验证默认和专用承载
      - QCI、MBR/GBR、计费规则名称
      - APN-AMBR 限制
   d) 计费信息 - Gy 会话 ID、配额状态（也可 GET /api/charging/<imsi>）
   e) 策略信息 - Gx 会话、已安装的 PCC 规则
   f) 最近事件 - GET /api/session_history?search=<imsi>
4. 如果未找到会话 → GET /api/diameter?status=disconnected 检查 PCRF 连接性
```

快速会话查找：

```
1. 用户报告问题
2. GET /api/sessions?search=<imsi|msisdn> (或 GET
/api/sessions/<imsi>)
3. 验证会话是否存在且基本详细信息：
   - 分配的 UE IP 地址
   - QoS 参数
   - 建立的隧道端点
4. 要进行详细分析 → POST /api/ue_search
```

系统健康检查：

```
1. GET /api/upf                  → 验证所有 PGW-U 对等体 "已关联"
2. GET /api/diameter?status=connected → 验证所有 PCRF 对等体 "已连接"
3. GET /api/sessions             → 检查活动会话数量与容量
4. GET /api/status               → 整体健康检查 ({"result":"ok"})
```

容量监控：

计算 GET /api/sessions 返回的条目
与许可/预期容量进行比较
识别峰值使用时间
监控跨 APN 的分布；使用 GET /api/ip_pools 获取每个池的利用率

API 与指标

使用 OAM API 进行：

深入了解用户故障排除 (POST /api/ue_search)
单个会话详细信息和状态检查 (GET /api/sessions/<imsi>)
实时对等体状态 (GET /api/upf, GET /api/diameter)
跨所有接口的快速健康检查 (GET /api/status)
按 IMSI/MSISDN/IP 故障排除特定用户
承载 QoS 分析 (MBR、GBR、QCI)
策略和计费规则检查 (GET /api/charging)
会话历史和审计跟踪 (GET /api/session_history)
IP 池容量监控 (GET /api/ip_pools)
验证配置和规则 (GET /api/upf_selection)

使用 Prometheus 指标进行：

历史趋势
警报和通知
容量规划图
性能分析
长期监控

最佳实践：同时使用两者 - OAM API 用于即时操作，Prometheus 用于趋势和警报。

监控与指标

除了 OAM API，OmniPGW 还提供与 Prometheus 兼容的指标用于监控：

可用指标

会话指标

teid_registry_count - 活动 S5/S8 会话 seid_registry_count - 活动 PFCP 会话 session_id_registry_count - 活动 Gx 会话 address_registry_count - 分配的 UE IP 地址 charging_id_registry_count - 活动计费 ID

消息指标

s5s8_inbound_messages_total - 接收到的 GTP-C 消息 sxb_inbound_messages_total - 接收到的 PFCP 消息 gx_inbound_messages_total - 接收到的 Diameter 消息
消息处理持续时间分布

错误指标

s5s8_inbound_errors_total - S5/S8 协议错误
sxb_inbound_errors_total - PFCP 协议错误
gx_inbound_errors_total - Diameter 错误

会话生命周期指标

pgw_session_create_total{result,cause,pdn_type,rat_type} - PDN 会话创建尝试，按结果（success / failure）、返回的 GTP 原因、PDN 类型
（ipv4 / ipv6 / ipv4v6）和 RAT 类型进行细分 pgw_session_delete_total{pdn_type} - 按 PDN 类型的 PDN 会话删除 pgw_session_modify_total - 处理的修改承载请求

UE IP 池指标

ue_pool_addresses_allocated{pool,ip_version} - 每个池（APN 模式）和 IP 版
本的分配 UE 地址
ue_pool_addresses_total{pool,ip_version} - 池的总容量 ue_pool_addresses_utilization_ratio{pool,ip_version} - 分配的比例（0.0-1.0）；用于池耗尽警报。注意共享地址范围的池各自计算共享分配

  - `ue_ip_allocation_failures_total{ip_version,reason}` - □□□□

文档结构

```
OmniPGW 文档
├── OPERATIONS.md (本指南)
│
└── docs/
    ├── 配置与设置
    │   ├── configuration.md          ��整的 runtime.exs 参考
    │   ├── ue-ip-allocation.md       IP 池配置
    │   └── pco-configuration.md      DNS、P-CSCF、MTU 设置
    │
    ├── 网络接口
    │   ├── pfcp-interface.md         Sxb/PFCP (PGW-U 通信)
    │   ├── diameter-gx.md            Gx (PCRF 通信)
    │   ├── diameter-gy.md            Gy/Ro (OCS 通信)
    │   └── s5s8-interface.md         S5/S8 (SGW-C 通信)
    │
    └── 操作
        ├── session-management.md     PDN 会话生命周期
        ├── charging-override.md      破玻璃在线计费覆盖
        └── monitoring.md             Prometheus 指标与警报
```

按主题的文档

□ 入门

文档	描述	目的
OPERATIONS.md	主要操作指南（本文件）	□述和快速入门

⚙ 配置

文档	描述	行数
configuration.md	完整的 runtime.exs 配置参考	1,600+
ue-ip-allocation.md	UE IP 池管理和分配	943
ipv6-dual-stack.md	PGW-C 和 UPF 之间的 IPv6 / IPv4v6 PDN 操作	-
pco-configuration.md	协议□□置选项（DNS、P-CSCF、MTU）	344

□ 网络接口

文档	描述	行数
pfcp-interface.md	PFCP/Sxb 接口到 PGW-U	1,355
diameter-gx.md	Diameter Gx 接口到 PCRF（策略控制）	941
diameter-gy.md	Diameter Gy/Ro 接口到 OCS（在线计费）	1,100+
charging-override.md	破玻璃覆盖以在 OCS 事件期间禁用在线计费	-
s5s8-interface.md	GTP-C S5/S8 接口到 SGW-C	456

□ □作与监控

文档	描述	行数
session-management.md	PDN 会话生命周期和操作	435
monitoring.md	Prometheus 指标、Grafana 仪表板、警报	807
data-cdr-format.md	CDR 文件格式、URR 配置、离线计费	847
qos-bearers.md	QoS 和承载管理、策略控制	448
troubleshooting.md	故障排除程序和常见问题	687

 高级功能

文档	描述	行数
pcscf-monitoring.md	P-CSCF 发现和健康监控	894

文档特点

Mermaid 图表

所有文档都包含 **Mermaid** 图表 以便于理解:

- 架构图
- 时序图（消息流）
- 状态机
- 网络拓扑

 实用示例

每个文档包括:

- 真实的配置示例
- 可直接复制粘贴的配置
- 常见用例

 故障排除

每个接口文档包括：

- 常见问题和解决方案
- 调试命令
- 诊断指标

 交叉引用

文档之间进行了广泛的交叉链接，以便于导航。

阅读路径

对于网络运营商

1. OPERATIONS.md - 述（本文件）
2. configuration.md - 设置
3. monitoring.md - 监控
4. session-management.md - 日常操作

对于网络工程师

1. OPERATIONS.md - 架构述（本文件）
2. pfcp-interface.md - 用户平面控制
3. diameter-gx.md - 策略控制
4. diameter-gy.md - 在线计费
5. s5s8-interface.md - 会话管理
6. ue-ip-allocation.md - IP 管理

对于配置与部署

1. configuration.md - 完整参考
2. ue-ip-allocation.md - IP 池
3. pco-configuration.md - 网络参数
4. monitoring.md - 设置监控

文档统计

- 文档总数: 14
- 总行数: ~10,900+
- 总大小: ~265 KB
- **Mermaid** 图表: 75+
- 代码示例: 150+

涉及的关键概念

架构

- □ □□/用户平面分离
- □ OTP/Elixir 架构
- □ □□□□
- □ □□ GenServer 的会话

协议

- □ PFCP (数据包转发控制协议)
- □ GTP-C v2 (GPRS 隧道协议)
- □ Diameter (RFC 6733)

3GPP 接口

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

□作

- □ □□□□
- □ IP 分配策略
- □ QoS 执行
- □ □□□□
- □ □□□□□

其他资源

3GPP 规范

规范	标题
TS 29.274	GTP-C v2（S5/S8 接口）
TS 29.244	PFCP（Sxb 接口）
TS 29.212	Diameter Gx 接口（策略控制）
TS 32.299	Diameter 计费应用（Gy/Ro）
TS 32.251	分组交换域计费
TS 23.401	EPC 架构

相关文档

- 配置参考：configuration.md - 完整的 `config/runtime.exs` 参考

在线计费覆盖（紧急切换）
在线计费覆盖是一个节点级开关。它立即将在线计费（Gy/OCS）从路径中移除，而无需重启。在OCS事件发生时使用它，当OCS不健康且用户受到影响时。

重启并不是禁用在线计费的可接受方式。重启会中断节点上的每个实时PDN会话。覆盖会改变已经建立的会话的行为，以及在其活动期间附加的会话的行为。

离线计费不受影响。PGW-C在覆盖活动期间继续计量使用并写入CDR。请参见CDR格式。

目录

1. 何时使用它
2. 覆盖的作用
3. API参考
4. 操作程序
5. 审计跟踪
6. 配置
7. 指标
8. 故障排除

何时使用它

当OCS导致用户影响，而您无法在OCS上修复时，请使用覆盖。有三种常见的故障模式。

故障模式	���户看到的情况	覆盖的作用
OCS拒绝或未响应CCR-Initial	新的附加会话被限制，或进入受限环境	新的附加会话不发送CCR-Initial并开始流量转发
OCS响应，但其计费不正确	用户在余额仍然存在的情况下被切断	解除受限环境，解除被阻止的承载
OCS保留预留并耗尽捆绑	余额在未使用的情况下消失	不再发送CCR-Update，因此窗口未被报告且未计费

不要对单个用户使用覆盖。要将一个会话移入或移出受限环境，请使用`PUT /api/charging/<imsi>`。请参见Diameter Gy接口。

覆盖的范围是一个节点。如果您运行多个PGW-C，请对每个节点应用覆盖。部分应用是可承受的。一些用户会恢复，其余的保持原样。

覆盖的作用

新会话

在覆盖活动期间，PGW-C将在线计费视为禁用。这与配置值charging.online::off的行为相同。PGW-C不发送 CCR-Initial。附加完成，默认承载转发流量。

覆盖优先于本地配置和PCRF。设置Online为启用的Gx CCA在覆盖活动期间不会重新启用计费。

现有会话

单独的配置标志仅对其后附加的会话有帮助。已经被阻止的用户是事件造成伤害的对象。当您启动覆盖时，PGW-C会清扫每个实时会话并执行两项操作。

1. 释放受限环境。如果会话在受限环境中，PGW-C会发送PFCP会话修改，移除重定向。这与OCS恢复信用时的路径相同。
2. 解除被阻止的承载。如果承载的转发动作规则设置为应用动作DROP，PGW-C会发送PFCP会话修改，将其设置回FORW。这涵盖了由CCR-Initial DIAMETER_USER_UNKNOWN分支限制的会话。

清扫是尽力而为。未响应的会话被视为失败，并在日志中记录。清扫将继续处理其余会话。PGW-C在会话之间并行运行清扫，因此所需时间由最慢的UPF响应决定，而不是它们的总和。

有一种情况无法修复。如果OCS在附加时发送了终止的最终单位动作，PGW-C已清除这些规则上的转发参数。PGW-C无法从会话状态重建它们。日志中会列出每个无法恢复的规则。用户必须重新附加以恢复。

已经打开Gy会话的会话

在覆盖活动期间，PGW-C不会为这些会话发送CCR-Update。使用报告被丢弃而不是缓冲。这是故意的。停用不能追溯性地对覆盖活动期间的窗口进行计费。

在分离时，PGW-C默认不发送CCR-Terminate。CCR-Terminate会在OCS上干净地释放预留，但每次分离发送一次会对已经不健康的OCS造成压力。OCS使用其自己的会话计时器来收回预留。如果您希望仍然发送CCR-Terminate，请将 send_ccr_t设置为true。

停用

停用会立即将节点恢复到正常计费状态。无需重启。PGW-C不会报告覆盖活动期间的窗口。在该窗口中的使用量以离线方式计量并写入CDR，但不会到达OCS。

API参考

覆盖是计费集合中的单例资源。标识符`override`是保留的。没有IMSI可以是该字符串。

所有路由都在OAM API的`/api`前缀下提供。请参见实时操作监控。

激活

```
PUT /api/charging/override
```

请求体：

```
{
  "ttl_seconds": 3600,
  "reason": "OCS incident INC-1234",
  "actor": "noc@example.com",
  "send_ccr_t": false
}
```

参数	类型	必需	默认	描述
`ttl_seconds`	整数	是	-	覆盖自动过期的秒数。最小`1`，最大`86400`（24小时）。
`reason`	字符串	是	-	自由文本，说明覆盖的原因。它出现在日志、CDR文件和`GET`中。
`actor`	字符串	否	呼叫者IP	谁启动了覆盖。当您省略此项时，PGW-C记录呼叫者的IP地址。
`send_ccr_t`	布尔	否	`false`	设置为`true`以在分离时仍发送Gy CCR-Terminate。保持`false`以保持静默。

响应：

```
{
  "online_charging": "off",
  "override": {
    "active": true,
    "reason": "OCS incident INC-1234",
    "actor": "noc@example.com",
    "ttl_seconds": 3600,
    "remaining_seconds": 3599,
    "activated_at": "2026-09-15T04:10:22.481Z",
    "expires_at": "2026-09-15T05:10:22.481Z",
    "send_ccr_t": false,
    "max_ttl_seconds": 86400
  },
  "relief": {
    "sessions_swept": 1842,
    "walled_gardens_released": 37,
    "bearers_ungated": 5,
    "failed": 0,
    "changed": [
      { "imsi": "313380000003530", "apn": "internet", "actions":
["walled_garden_released"] }
    ]
  }
}
```

字段	描述
`online_charging`	覆盖活动期间为`off`，否则为`on`。
`override.active`	覆盖处于启动状态且TTL未过期时为`true`。
`override.remaining_seconds`	自动过期的秒数。没有活动覆盖时为`null`。
`relief.sessions_swept`	PGW-C请求放弃其计费限制的实时会话数量。
`relief.walled_gardens_released`	离开受限环境的会话数量。
`relief.bearers_ungated`	被恢复为转发的会话数量。
`relief.failed`	未响应的会话数量。它们的状态未改变。
`relief.changed`	实际改变的会话，以及对每个会话采取的操作。

当`ttl_seconds`缺失、超出范围或不是整数时，以及当`reason`缺失或为空时，PGW-C会以`400 Bad Request`拒绝请求。

读取状态

```
GET /api/charging/override
```

响应包含与激活响应相同的`online_charging`和`override`字段，`remaining_seconds`从请求时间开始计算。没有`relief`字段。

列表端点`GET /api/charging`也在每个会话的计费状态旁边携带覆盖状态，字段为`charging_override`。

停用

```
DELETE /api/charging/override
```

请求体（可选）：

```
{ "actor": "oncall@example.com" }
```

响应包含与`GET`相同的字段，`active`设置为`false`。

□作程���

在OCS事件期间，请遵循以下步骤。

1. 确认OCS是原因。检查`GET /api/diameter?status=disconnected`以获取Gy对等状态，以及`GET /api/charging`以获取受限环境中的会话。
2. 在第一个PGW-C上启动覆盖。给出一个命名事件工单的`reason`。将`ttl_seconds`设置为覆盖您预期修复时间的值。
3. 阅读响应中的`relief`块。确认`failed`为`0`。如果不是，请参见故障排除。
4. 在每个剩余的PGW-C节点上重复步骤2和3。
5. 确认每个节点上的`charging_override_active`指标读取为`1`。
6. 修复OCS。
7. 在每个节点上使用`DELETE /api/charging/override`停用覆盖。
8. 确认每个节点上的`charging_override_active`指标读取为`0`，并且新的附加会话再次发送CCR-Initial。

如果您不停用覆盖，它将在TTL过期时自动过期。PGW-C在过期时会在日志中写入警告，并在CDR文件中写入注释行。

审计跟踪

可以静默保留的覆盖是收入泄漏。PGW-C在四个地方记录每个状态更改。

日志

PGW-C在激活、停用、TTL过期和重启后重新启动时写入`warning`。每行以`[CHARGING OVERRIDE]`为前缀，并携带原因、执行者和截止日期。清扫会写入第二个`warning`，记录更改和失败的会话数量。

CDR文件

PGW-C在当时打开的CDR文件中写入注释行。写入是异步的，因此在慢速挂载上，CDR目录不能阻止或失败覆盖本身。注释行以#开头，与CDR文件头使用的标记相同，因此调解会跳过它们。该行保留在文件记录中，这在故意未计费的流量窗口时是必要的。

CHARGING OVERRIDE ACTIVATED at 2026-09-15T04:10:22.481Z: online charging (Gy/OCS) disabled node-wide; usage in this window is
metered offline but NOT reported to the OCS. reason=OCS incident INC-1234 actor=noc@example.com ttl_seconds=3600 expires_at=2026-09-15T05:10:22.481Z send_ccr_t=false
CHARGING OVERRIDE DEACTIVATED at 2026-09-15T04:52:09.113Z:
online charging (Gy/OCS) re-enabled node-wide. usage between 2026-09-15T04:10:22.481Z and 2026-09-15T04:52:09.113Z (2507s) was not reported to the OCS. original_reason=OCS incident INC-1234 activated_by=noc@example.com deactivated_by=oncall@example.com

嵌入的换行符在 reason中被压缩为空格。注释永远不会分割CDR记录。请参见CDR格式。

指标

请参见指标。在覆盖活动期间，警报charging_override_active以便操作员被告知。

Web面板

PGW会话页面在覆盖活动期间显示横幅。横幅携带原因、执行者和剩余时间。

配置

覆盖不需要任何配置即可工作。一个可选键控制武装状态写入的位置，以便在重启后生效。

config:pgw_c,
Where the armed override is persisted across a restart charging_override_file: "/var/lib/pgw_c/charging_override.json"

参数	类型	必需	默认
`charging_override_file`	字符串	否	`/var/lib/pgw_c/charging_override.jso`

重启行为

事件期间的崩溃不能静默地为每个用户重新武装切断。PGW-C将武装覆盖写入`charging_override_file`并在启动时读取它。

重启不会延长TTL。PGW-C保持原始的`expires_at`。

如果PGW-C无法写入文件，它会记录警告并继续。覆盖在节点上生效，但不会���重启后生效。检查目录的文件系统权限。

指标

指标：`charging_override_active`
类型：Gauge
描述：`1`表示节点级覆盖禁用在线计费，`0`表示其他情况。
标签：无

指标：`charging_override_remaining_seconds`
类型：Gauge
描述：覆盖自动过期前剩余的秒数。没有活动覆盖时为`0`。
标签：无

这两个仪表在指标轮询间隔时刷新。请参见监控与指标以获取轮询间隔和导出配置。

示例查询：

```
# Any node with online charging bypassed right now
max(charging_override_active)

# Nodes where the override expires in less than 10 minutes
charging_override_remaining_seconds > 0 and
charging_override_remaining_seconds < 600
```

示例警报：

```
- alert: OnlineChargingOverrideActive
  expr: charging_override_active == 1
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "Online charging is bypassed on {{ $labels.instance
}}"
    description: "Subscriber usage is metered for CDRs but is not
rated online."
```

故障排除

激活请求返回**400**

症状：响应体中提到`ttl_seconds`或`reason`。

可能原因：

- `ttl_seconds`缺失、`0`、负数或大于`86400`
- `reason`缺失或为空

解决方案：

1. 将`ttl_seconds`设置为`1`到`86400`之间的整数。
2. 将`reason`设置为说明事件的文本。

清扫报告失败

症状：激活响应中的`relief.failed`大于`0`。

可能原因：

- 会话进程在调用超时内未响应
- UPF未接受PFCP会话修改

□□□

gy.enabled为false
PCRF将Gx CCA Online AVP设置为禁用

解决方案:

1. 检查charging和gy块。请参见配置参考。
2. 使用GET /api/charging/<imsi>检查Gx CCA。请参见Diameter Gx接口。

相关文档

Diameter Gy接口 - 在线计费、配额和受限环境
CDR格式 - 离线计费和CDR文件
监控与指标 - 指标导出和仪表板
故障排除 - 一般故障排除程序
操作指南 - 入口点

OmniPGW 配置指南

runtime.exs 配置的完整参考

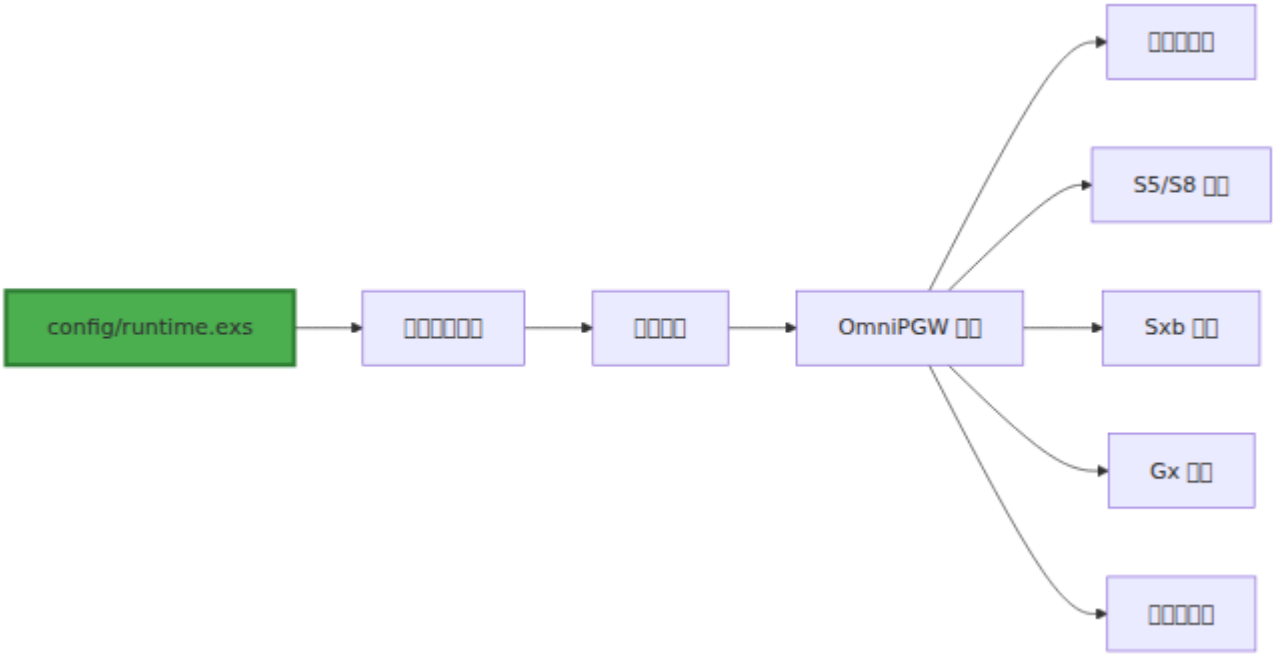
□ Omnitouch □□□□□□

目录

1. □述
2. 配置文件结构
3. 指标配置
4. Diameter/Gx 配置
5. S5/S8 配置
6. Gn/Gp 配置 (GGSN)
7. Sxb/PFCP 配置
    - UPF 选择策略
    - 使用 UPF 池进行负载均衡
    - 基于 DNS 的选择
    - 干运行模式
8. UE IP 池配置
9. PCO 配置
10. OAM API 配置
11. 完整示例
12. 配置验证

概述

OmniPGW 使用 运行时配置，该配置在 `config/runtime.exs` 中定义。此文件在 应用程序启动时 进行评估，并允许根据环境变量或外部源进行动态配置。



配置理念

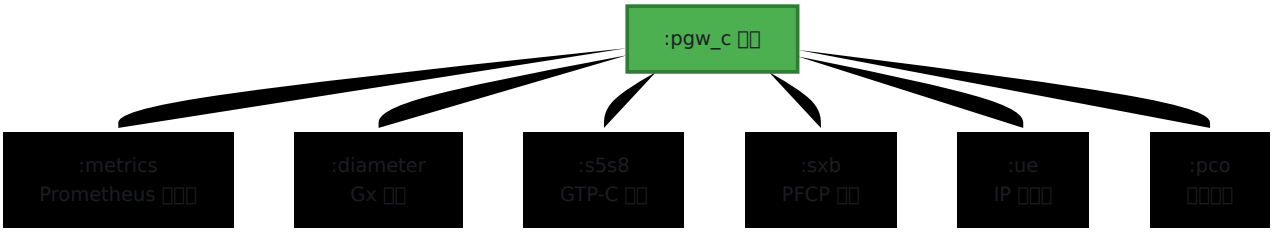
关键原则：

- 单一真实来源 - 所有配置在一个文件中
- 类型安全 - 启动时验证配置
- 环境灵活性 - 支持开发、测试、生产
- 清晰的默认值 - 合理的默认值与明确的覆盖

配置文件结构

文件位置

```
pgw_c/
├── config/
│   ├── config.exs          # 基础配置（导入 runtime.exs）
│   ├── dev.exs             # 开发特定配置
│   ├── prod.exs            # 生产特定配置
│   └── runtime.exs         # ← 主要配置文件
```



顶级结构

```
# config/runtime.exs
import Config

config :logger, level: :info

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

配置部分

指标配置

目的

配置 Prometheus 指标导出器以监控 OmniPGW。

配置块

```
config :pgw_c,
  metrics: %{
    # 用/禁用指标导出器
    enabled: true,

    # 绑定 HTTP 服务器的 IP 地址
    ip_address: "0.0.0.0",

    # 指标端点的端口
    port: 9090,

    # 多久轮询注册表（毫秒）
    registry_poll_period_ms: 10_000
  }
```

参数

参数	类型	默认值	描述
`enabled`	布尔值	`true`	启用指标导出器
`ip_address`	字符串（IP）	`"0.0.0.0"`	绑定地址（0.0.0.0 = 所有接口）
`port`	整数	`9090`	`/metrics` 端点的 HTTP 端口
`registry_poll_period_ms`	整数	`10_000`	注册表计数的轮询间隔

示例

生产 - 绑定到特定 IP：

```
metrics: %{
  enabled: true,
  ip_address: "10.0.0.20",  # 管理网络
  port: 9090,
  registry_poll_period_ms: 5_000  # 每 5 秒轮询一次
}
```

开发 - 仅限本地主机��

```
metrics: %{
  enabled: true,
  ip_address: "127.0.0.1",
  port: 42069,  # 非标准端口
  registry_poll_period_ms: 10_000
}
```

禁用指标：

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

访问指标

```
# 默认端点
curl http://<ip_address>:<port>/metrics

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

参见：监控与指标指南 以获取详细的指标文档。

Diameter/Gx 配置

目的

配置 Gx 接口的 Diameter 协议（PCRF 通信）。

配置块

```
config :pgw_c,
  diameter: %{
    # 监听 Diameter 连接的 IP 地址
    listen_ip: "0.0.0.0",

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

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

    # PCRF 对等体列表
    peer_list: [
      %{
        # PCRF Diameter 身份
        host: "pcrf.epc.mnc001.mcc001.3gppnetwork.org",

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

        # PCRF IP 地址
        ip: "10.0.0.30",

        # 主动与 PCRF 的连接
        initiate_connection: true
      }
    ]
  }
```

参数

参数	类型	必需	描述
`listen_ip`	字符串（IP）	是	Diameter 监听地址
`host`	字符串（FQDN）	是	OmniPGW 的 Origin-Host（必须是 FQDN）
`realm`	字符串（域）	是	OmniPGW 的 Origin-Realm
`peer_list`	列表	是	PCRF 对等体配置

对等体配置：

参数	类型	必需	描述
`host`	字符串（FQDN）	是	PCRF Diameter 身份
`realm`	字符串（域）	是	PCRF 域
`ip`	字符串（IP）	是	PCRF IP 地址
`initiate_connection`	布尔值	是	OmniPGW 是否连接到 PCRF

FQDN 格式

Diameter 身份必须是 **FQDN**：

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

# 错误
host: "omnipgw"                # 不是 FQDN
host: "10.0.0.20"              # 不允许使用 IP
```

3GPP 格式：

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

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

□□

□□ 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
    }
  ]
}
```

□□ 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-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 □□□□□□

```
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  # □□ PCRF □□
    }
  ]
}
```

□□□ Diameter Gx □□□□

S5/S8 配置

目的

配置与 SGW-C 通信的 GTP-C 接口。

配置块

```
config :pgw_c,
  s5s8: %{
    # S5/S8 接口的本地 IPv4 地址
    local_ipv4_address: "10.0.0.20",

    # 可选: 本地 IPv6 地址
    local_ipv6_address: nil,

    # 可选: 覆盖默认 GTP-C 端口 (2123)
    local_port: 2123,

    # GTP-C 请求超时 (毫秒) (默认: 500毫秒)
    # 等待 GTP-C 响应时每次尝试的超时
    request_timeout_ms: 500,

    # GTP-C 请求的重试次数 (默认: 3)
    # 总最大等待时间 = request_timeout_ms * request_attempts
    request_attempts: 3
  }
```

参数

参数	类型	默认值	描述
`local_ipv4_address`	字符串（IPv4）	必需	S5/S8 接口的 IPv4 地址
`local_ipv6_address`	字符串（IPv6）	`nil`	S5/S8 接口的 IPv6 地址（可选）
`local_port`	整数	`2123`	GTP-C 的 UDP 端口（标准端口 2123）
`request_timeout_ms`	整数	`500`	每次重试的超时（毫秒）
`request_attempts`	整数	`3`	放弃前的重试次数

协议细节

- 协议：GTP-C 版本 2
- 传输：UDP
- 标准端口：2123
- 方向：从 SGW-C 接收

示例

仅 **IPv4**（常见）：

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

IPv4 + IPv6 双栈：

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

自定义端口（非标准）：

```
s5s8: %{
  local_ipv4_address: "10.0.0.20",
  local_port: 2124  # 自定义端口
}
```

高延迟网络：

```
s5s8: %{
  local_ipv4_address: "10.0.0.20",
  request_timeout_ms: 1500,  # 每次尝试 1.5 秒
  request_attempts: 3         # 总计：最大 4.5 秒
}
```

超时配置

S5/S8 接口使用可配置的超时进行 GTP-C 请求/响应事务（创建承载请求、删除承载请求）。

总等待时间计算：

```
总最大等待 = request_timeout_ms × request_attempts
默认：500毫秒 × 3 = 1.5 秒
```

调优指南：

网络延迟	推荐超时	总等待时间
低延迟（<50ms）	200-300毫秒	600-900毫秒
正常（50-150ms）	500毫秒（默认）	1.5秒
高延迟（>150ms）	1000-2000毫秒	3-6秒
卫星/不稳定	2000-3000毫秒	6-9秒

何时调整：

- 增加超时 如果经常看到“创建承载请求超时”错误，但 Wireshark 显示响应已到达
- 减少超时 以便在低延迟环境中更快地检测失败
- 增加重试次数 对于存在丢包的不可靠网络

超时行为：

- 超时时，记录错误：`"创建承载请求超时"`
- 返回给 PCRF 的 Diameter 错误：结果代码 5012（无法遵从）
- 当 Charging-Rule-Remove 到达时，承载保持在早期存储中以进行清理

网络规划

IP 地址选择：

- 使用专用的管理/信令网络
- 确保从所有 SGW-C 节点的可达性
- 考虑冗余（VRRP/HSRP）以实现高可用性

防火墙规则：

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

Gn/Gp 配置（GGSN）

目的

配置与 SGSN 通信的 GTP-C v1 接口，使 GGSN 功能支持 2G/3G 网络。

配置块

```
config :pgw_c,
  gn: %{
    # Gn 接口的本地 IPv4 地址
    local_ipv4_address: "10.0.0.20",

    # 可选：本地 IPv6 地址（大多数 2G/3G 网络仅支持 IPv4）
    local_ipv6_address: nil,

    # GTP-C 端口（与 S5/S8 共享）
    local_port: 2123
  },

  # 在 PCO 中返回的 DNS 服务器（与 S5/S8 共享）
  dns: %{
    primary_ipv4: {8, 8, 8, 8},
    secondary_ipv4: {8, 8, 4, 4}
  }
```

参数

参数	类型	默认值	描述
`local_ipv4_address`	字符串（IPv4）	必需	Gn 接口的 IPv4 地址（GGSN 地址）
`local_ipv6_address`	字符串（IPv6）	`nil`	Gn 接口的 IPv6 地址（可选，少用）
`local_port`	整数	`2123`	GTP-C v1 的 UDP 端口（与 S5/S8 共享）

DNS 参数

参数	类型	默认值	描述
`primary_ipv4`	元组	`{8, 8, 8, 8}`	PCO 响应的主 DNS 服务器
`secondary_ipv4`	元组	`{8, 8, 4, 4}`	PCO 响应的备用 DNS 服务器

协议细节

- 协议：GTP-C 版本 1 (3GPP TS 29.060)
- 传输：UDP
- 标准端口：2123（与 GTP-C v2 共享）
- 方向：从 SGSN 接收

与 S5/S8 的共存

OmniPGW 可以同时作为 PGW（4G）和 GGSN（2G/3G）运行：

- 两个接口共享 UDP 端口 2123
- GTP 版本从消息头自动检测
- 同一 IP 地址可以服务于 4G 和 2G/3G 流量
- 共享基础设施：IP 池、UPF/PFCP、在线计费

参见： Gn/Gp 接口文档

Sxb/PFCP 配置

目的

配置与 PGW-U（用户平面）通信的 PFCP 接口。

配置块

```
config :pgw_c,
  sxb: %{
    # PFCP 通信的本地 IP 地址
    local_ip_address: "10.0.0.20",

    # 可选：覆盖默认 PFCP 端口（8805）
    local_port: 8805
  }
```

参数

参数	类型	默认值	描述
`local_ip_address`	字符串（IP）	必需	PFCP 监听地址
`local_port`	整数	`8805`	PFCP UDP 端口

> 重要：
>
> - 所有 **UPF** 对等体在启动时会从 **`upf_selection`** 配置（规则 + 后备池）自动注册
> - 自动注册的 UPF 使用合理的默认值：
>   - 自动生成的名称：`"UPF-<ip>:<port>"`
>   - 被动 PFCP 关联（等待 UPF 发起）
>   - 5 秒心跳间隔
> - UPF 选择规则和池在单独的 `upf_selection` 部分配置。请参见下面的 UPF 选择策略。
> - 支持动态 UPF 注册，适用于未在配置中的 DNS 发现的 UPF

示例

最小配置：

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

# 所有 upf_selection 中的 UPF 将自动注册为:
# - 自动生成的名称:“UPF-10.0.0.21:8805”
# - 被动 PFCP 关联(等待 UPF 连接)
# - 5 秒心跳间隔
```

自定义 **PFCP** 端口:

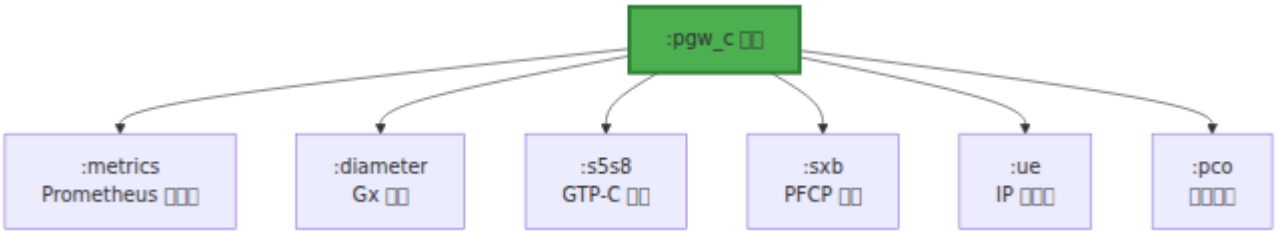
```
sxb: %{
  local_ip_address: "10.0.0.20",
  local_port: 8806  # 非标准 PFCP 端口
}
```

带有 **UPF** 选择的完整示例:

```
sxb: %{
  local_ip_address: "10.0.0.20"
},
upf_selection: %{
  rules: [
    %{
      name: "IMS 池",
      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}
  ]
}
# 所有 3 个 UPF（10.0.1.21、10.0.1.22、10.0.2.21）将自动注册
```

基于 **DNS** 的选择（动态注册）：

```
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 发现的 UPF 将在首次使用时动态注册
```



UPF 选择策略

重要：UPF 选择配置已简化。所有 UPF 对等体会从 `upf_selection` 配置中自动注册。

配置结构

UPF 选择在 **`upf_selection`** 部分配置，定义：

1. 静态规则 - 基于模式的路由与负载均衡池
2. **DNS** 设置 - 基于位置的动态 UPF 发现
3. 后备池 - 当没有规则匹配且 DNS 失败时的默认池

选择优先级顺序

1. 静态规则（最高优先级） - 基于模式的路由与负载均衡池
2. 基于 **DNS** 的选择（较低优先级） - 基于位置的动态 UPF 发现
3. 后备池（最低优先��） - 当没有规则匹配且 DNS 失败时的默认池

UPF 选择决策流程

可用匹配字段

静态规则可以匹配以下任一会话属性：

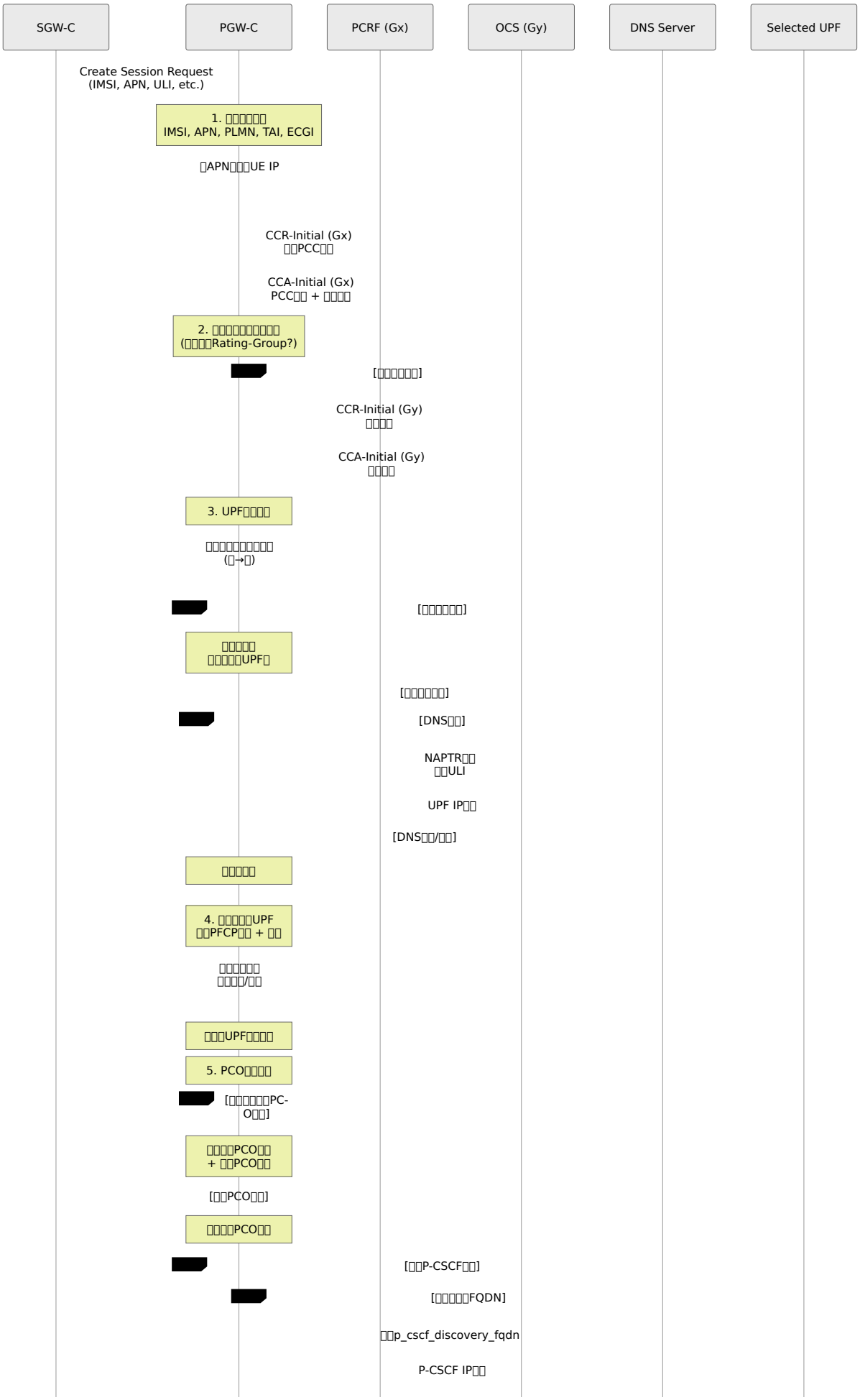
匹配字段	描述	示例模式
`:imsi`	国际移动用户身份	`^001001.*`（测试网络）
`:apn`	接入点名称 / DNN	`^internet\.` 或 `^ims\.`
`:serving_network_plmn_id`	服务网络标识符	`^001001$`
`:sgw_ip_address`	SGW IP 地址	`^10\.100\..*`
`:uli_tai_plmn_id`	跟踪区域 PLMN ID	`^313.*`
`:uli_ecgi_plmn_id`	E-UTRAN 小区 PLMN ID	`^313.*`

选择方法比较

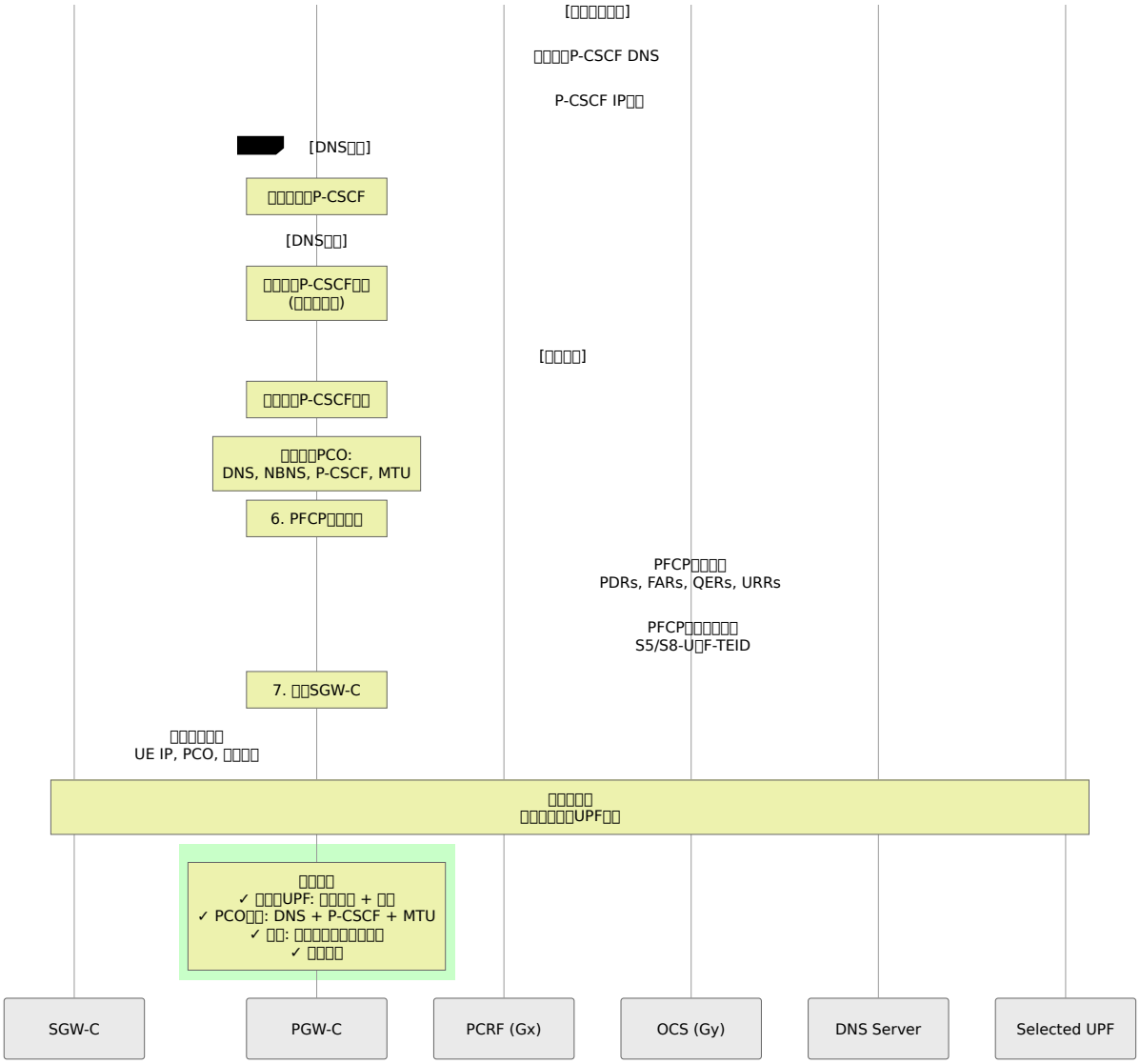
方法	何时使用	优点	缺点
UPF 池	生产部署	负载均衡、高可用性、灵活权重	需要多个 UPF
基于 **APN**	服务差异化	分别路由 IMS/互联网	静态配置
基于 **IMSI**	漫游场景	地理路由	正则表达式复杂性
基于 **DNS**	MEC/边缘计算	动态、基于位置	需要 DNS 基础设施
后备池	安全网	始终有 UPF	可能不是最佳选择
干运行模式	测试配置	安全测试	没有真实流量

完整会话建立流程

该图显示了包括UPF选择和PCO填充在内的完整端到端会话建立流程：



SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
DNS Server
Selected UPF
Create Session Request
(IMSI, APN, ULI, etc.)
1. 提取会话属性
IMSI, APN, PLMN, TAI, ECGI
从APN池分配UE IP
CCR-Initial (Gx)
请求PCC规则
CCA-Initial (Gx)
PCC规则 + 计费信息
2. 检查是否需要在线计费
(是否存在Rating-Group?)
[需要在线计费]
CCR-Initial (Gy)
请求配额
CCA-Initial (Gy)
授予配额
3. UPF选择过程
按优先级评估静态规则
(高→低)
[找到规则匹配]
规则匹配!
从规则获取UPF池
[没有规则匹配]
[DNS启用]
NAPTR查询
基于ULI
UPF IP地址
[DNS禁用/失败]
使用回退池
4. 过滤健康的UPF
检查PFCP关联 + 心跳
加权随机选择
处理活动/待命
选择的UPF已选定!
5. PCO填充过程
[规则匹配并有PCO覆盖]
使用规则PCO覆盖
+ 全局PCO回退
[没有PCO覆盖]
使用全局PCO配置
[启用P-CSCF发现]
[每规则发现FQDN]
查询p_cscf_discovery_fqdn
P-CSCF IP列表



□□□□□□

1. **UPF□□□□□□**
   - □□□□□□□□□□□□→ DNS□□ → □□□
   - □□□□□□□□□□□□
   - □□□□□□□□/□□□□
   - □□□ PFCP□□ □□□UPF□□□□□□
2. **PCO□□□□□□**
   - □□PCO□□ → P-CSCF DNS□□ → □□PCO□□
   - □□□□□□□□□□□□□□□□□□□□□□□□□□□□
   - □□□ PCO□□ □□□□□□□PCO□□
3. **P-CSCF□□□□□□**

每规则FQDN → 全局DNS发现 → 静态规则PCO → 全局静态PCO 参见：P-CSCF监控 以获取发现指标和健康跟踪

4. 计费集成：

PCRF确定是否需要在线计费（Rating-Group + Online=1）
OCS在会话建立之前授予配额
PGW-C跟踪配额并通过CCR-Update请求更多
参见：Diameter Gx接口 和 Diameter Gy接口 以获取计费详细信息

完整配置示例

这是一个完整的示例，显示了多池UPF选择和自动对等体注册：

```
config :pgw_c,
  # PFCP接口 - 所有UPF都从upf_selection自动注册
  sxb: %{
    local_ip_address: "127.0.0.20"
  },

  # UPF选择逻辑 - 此处定义的所有UPF都自动注册
  upf_selection: %{
    # 基于DNS的选择设置
    dns_enabled: false,
    dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],
    dns_suffix: "epc.3gppnetwork.org",
    dns_timeout_ms: 5000,

    # 静态选择规则（按优先级顺序评估）
    rules: [
      # 规则1: IMS流量 - 最高优先级
      %{
        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}
        ]
      },

      # 规则2: 企业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}
        ]
      },

      # 规则3: 互联网流量 - 负载均衡
```

```
      %{
        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}
        ]
      }
    ],

    # 回退池 - 当没有规则匹配且DNS失败时使用
    fallback_pool: [
      %{remote_ip_address: "127.0.0.21", remote_port: 8805,
weight: 100}
    ]
  }
```

关键特性

当前格式:

- □ □□□□□□□□□`upf_selection`的UPF在启动时自动注册
- □ □□□□□□□UPF选择和对等体配置在一个部分中
- □ □□□□□□□□□`upf_pool`格式（即使是单个UPF）
- □ □□□□□□□□□`fallback_pool`，具有加权分配
- □ **DNS**集成: DNS设置与选择规则并列
- □ □□□□□□□□□□□□□□□DNS发现的UPF
- □ □□□□□□□□□□UPF都以5秒的心跳进行监控

从先前格式迁移:

- 移除: `sxb.peer_list`字段（不再需要）
- 移除: 嵌入对等体配置中的`selection_list`

所有UPF定义现在放在upf_selection规则和回退池中

UPF池的工作原理：

1. 健康感知选择：只有健康的UPF接收流量

健康 = PFCP关联活动 + 连续错过心跳少于3次
不健康的UPF会被自动过滤
如果没有健康的UPF，则回退到所有UPF（快速失败）

2. 活动/待命支持：使用weight: 0表示待命UPF

活动UPF（weight > 0）：在健康时接收流量
待命UPF（weight == 0）：仅在所有活动UPF故障时接收流量激活时待命UPF被视为weight: 1

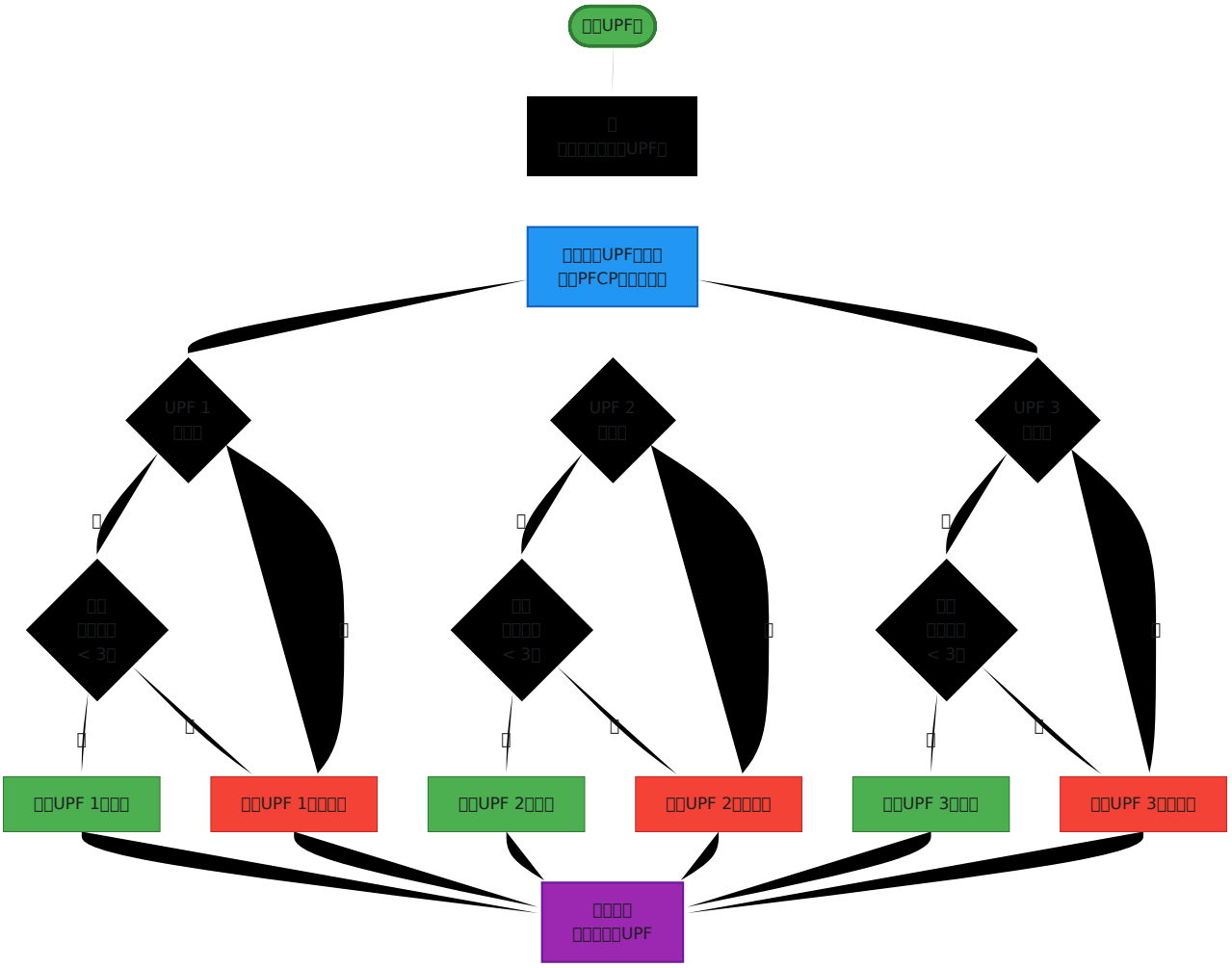
3. 加权随机选择：每个会话根据权重随机分配给健康的UPF

在上面的示例中：70%分配给.21，20%分配给.22，10%分配给.23
较高的权重 = 更多会话分配给该UPF
相等的权重 = 平均分配

4. 自动注册：池中的所有UPF在启动时自动注册

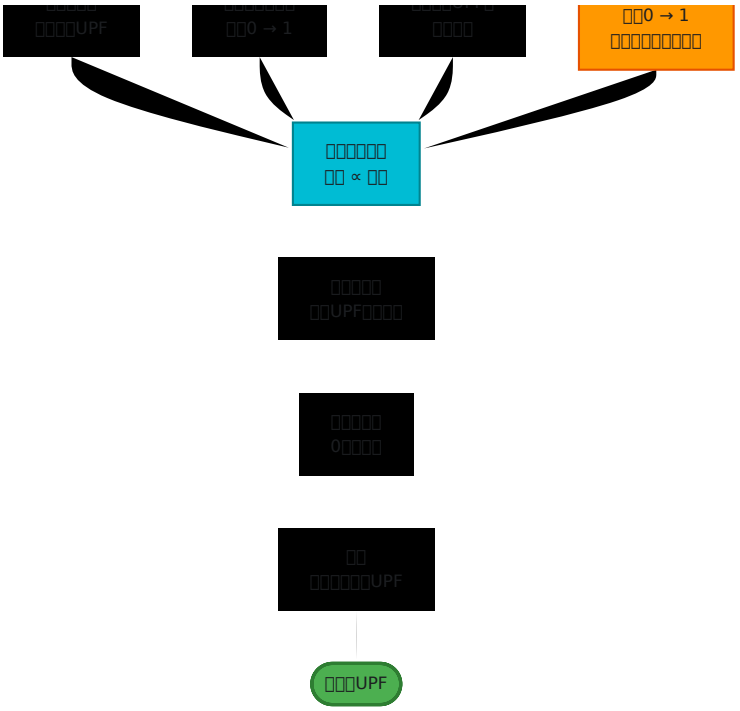
自动生成的名称："UPF-<ip>:<port>"
默认设置：被动PFCP关联，5秒心跳
对所有配置的UPF进行即时健康跟踪

健康感知选择与活动/待命



选择UPF池
检查每个UPF的健康
查询PFCP对等体状态
UPF 1
UPF 2
UPF 3
是
是
是
标记UPF 1为健康
标记UPF 1为不健康
标记UPF 2为健康
标记UPF 2为不健康
标记UPF 3为健康
标记UPF 3为不健康
过滤池以
仅保留健康UPF

是
日志：池中没有健康UPF
使用完整池作为回退
否，所有
是
否
是
否
激活待命UPF



□□□□□□□□□□

```
□: [
  UPF-A: □□ 50, □□ ✓
  UPF-B: □□ 30, □□ ✓
  UPF-C: □□ 20, □□ ✓
]

□□□: 50 + 30 + 20 = 100

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

□□□: 63 → □□UPF-B
□□□: 15 → □□UPF-A
□□□: 91 → □□UPF-C
```

□□/□□□□□□□□□□

```
初始池: [
  UPF-A: 权重 100, 健康 ✓    (活动)
  UPF-B: 权重 0, 健康 ✓      (待命)
]

场景1: UPF-A健康
→ 使用活动池: [UPF-A: 100]
→ 所有流量到UPF-A

场景2: UPF-A故障
→ 没有活动UPF健康
→ 激活待命: [UPF-B: 1]
→ 所有流量转移到UPF-B
→ 日志: "所有活动UPF故障，激活待命UPF"

场景3: 两者均不健康
→ 没有健康UPF
→ 使用完整池: [UPF-A: 100, UPF-B: 0]
→ 根据权重选择（尝试连接，可能失败）
→ 日志: "池中没有健康UPF，使用完整池作为回退"
```

常见权重模式:

```
# 平均分配（每个25%）
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}
]

# 主/备负载均衡（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}
]

# 活动/待命（100%主，0%待命，直到主故障）
upf_pool: [
  %{remote_ip_address: "10.0.1.21", remote_port: 8805, weight:
100},  # 活动
  %{remote_ip_address: "10.0.1.22", remote_port: 8805, weight: 0}
# 待命（仅在活动故障时）
]

# 活动与多个待命（激活时负载均衡）
upf_pool: [
  %{remote_ip_address: "10.0.1.1", remote_port: 8805, weight:
100},   # 活动
  %{remote_ip_address: "10.0.1.2", remote_port: 8805, weight: 0},
# 待命1
  %{remote_ip_address: "10.0.1.3", remote_port: 8805, weight: 0}
# 待命2
]
# 结果：活动获得100%。如果活动故障待命获得50/50%。

# A/B测试（50% / 50%）
upf_pool: [
  %{remote_ip_address: "10.0.1.100", remote_port: 8805, weight:
50},  # 旧版本
  %{remote_ip_address: "10.0.1.200", remote_port: 8805, weight:
50}   # 新版本
]
```

用例：

活动/待命故障转移：使用weight: 0表示热备UPF，仅在主UPF故障时激活健康感知HA：当UPF失去PFCP关联或错过心跳时自动故障转移水平扩展：在多个UPF之间分配负载以增加容量
高可用性：自动分配防止单个UPF过载
渐进式发布：使用权重进行金丝雀部署（例如，95%旧，5%新）
成本优化：将更多流量路由到更高容量的UPF
地理分布：在边缘UPF之间平衡会话

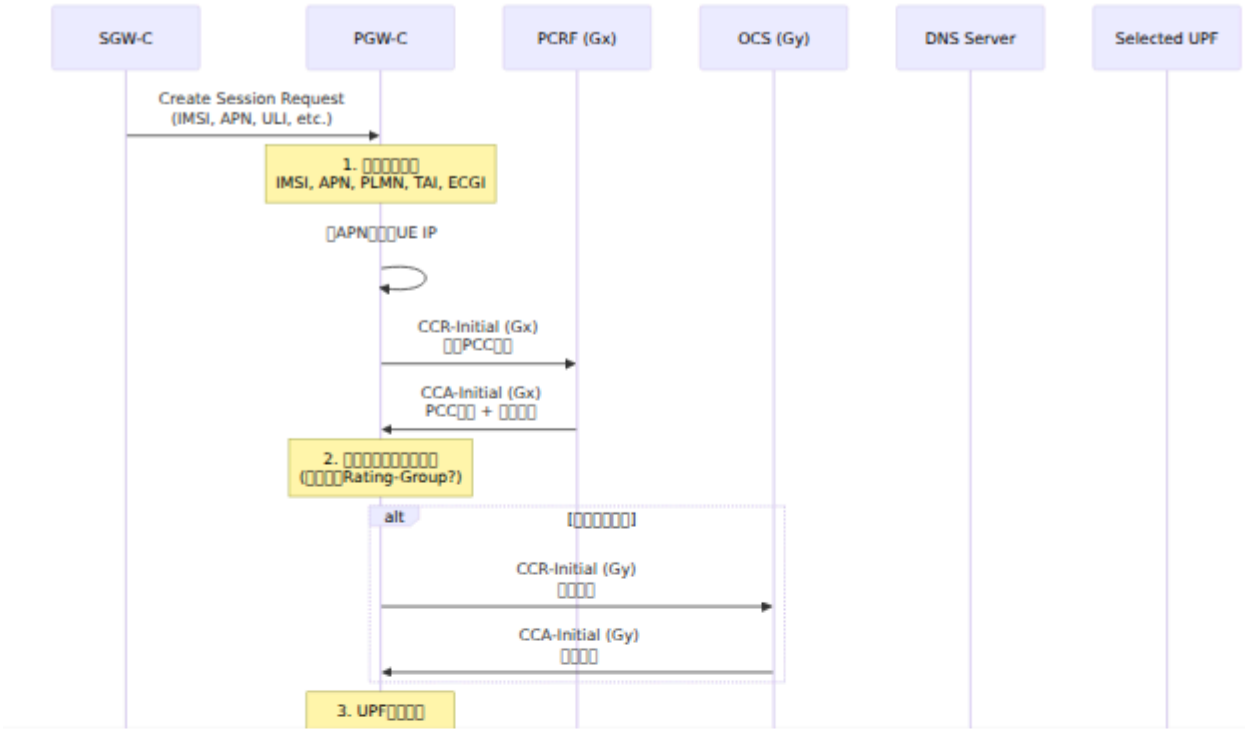
PCO（协议配置选项）覆盖：

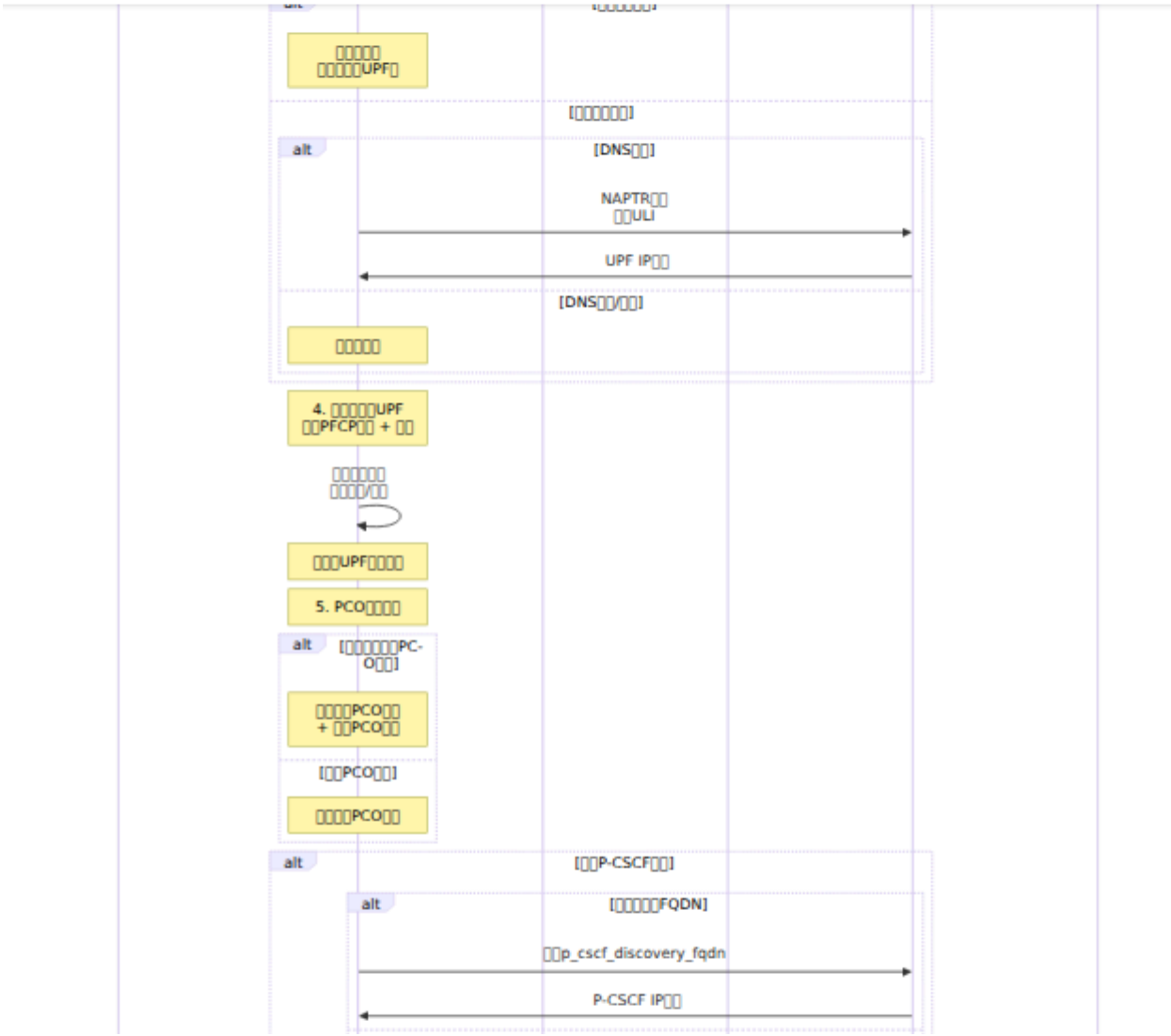
每个UPF选择规则可以选择性地指定自定义PCO值，这些值覆盖匹配会话的默认PCO配置。这允许不同的APN或流量类型接收不同的网络参数。

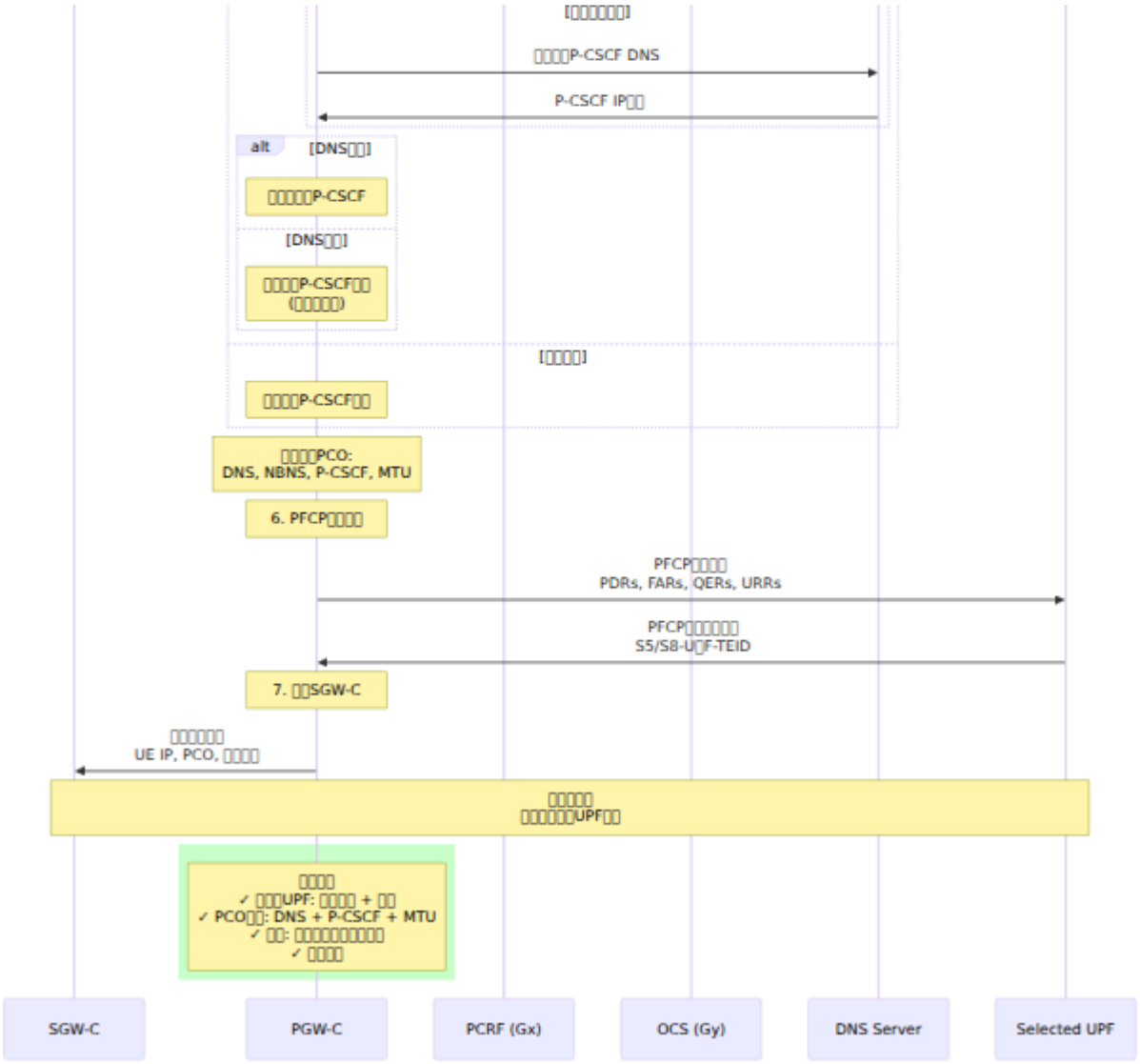
PCO覆盖的工作原理：

1. 部分覆盖：仅指定要覆盖的PCO字段
2. 默认回退：未指定的字段使用主pco配置中的值
3. 规则特定：每个规则可以有不同的PCO覆盖
4. 优先级合并：规则PCO优先于默认PCO

PCO□□□□


SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
DNS Server
Selected UPF
Create Session Request
(IMSI, APN, ULI, etc.)
1. □□□□□□
IMSI, APN, PLMN, TAI, ECGI
□APN□□□UE IP
CCR-Initial (Gx)
□□PCC□□
CCA-Initial (Gx)
PCC□□ + □□□□
2. □□□□□□□□□□
(□□□□Rating-Group?)
alt
[□□□□□□□]
CCR-Initial (Gy)
□□□□
CCA-Initial (Gy)
□□□□
3. UPF□□□□


□□□□□□
□□□□□UPF□
[□□□□□□□]
alt
[DNS□□]
NAPTR□□
□□ULI
UPF IP□□
[DNS□□/□□]
□□□□□
4. □□□□□UPF
□□PFCP□□ + □□
□□□□□□
□□□/□□
□□□UPF□□□□
5. PCO□□□□
alt
[□□□□□□PC-O□□]
□□□□PCO□□
+ □□PCO□□
[□□PCO□□]
□□□□PCO□□
alt
[□□P-CSCF□□]
alt
[□□□□□FQDN]
□□p_cscf_discovery_fqdn
P-CSCF IP□□



每个PCO字段的优先级顺序：

1. 规则PCO覆盖（最高优先级）
2. P-CSCF DNS发现（仅针对P-CSCF地址）
3. 全局PCO配置（最低优先级/回退）

示例：IMS规则覆盖DNS，企业规则覆盖所有内容

```
IMS会话（匹配“IMS流量”规则）：
├─ DNS服务器：来自全局（未在规则中覆盖）
├─ P-CSCF：来自DNS发现（规则中设置p_cscf_discovery_fqdn）
│  └─ 回退：如果DNS失败，则来自规则
└─ MTU：来自全局（未在规则中覆盖）

企业会话（匹配“企业流量”规则）：
├─ DNS服务器：来自规则（192.168.1.10, 192.168.1.11）
├─ P-CSCF：来自全局（未在规则中覆盖）
└─ MTU：来自规则（1500）

默认会话（未匹配规则）：
├─ DNS服务器：来自全局
├─ P-CSCF：来自全局或如果□用全局发现则来自DNS
└─ MTU：来自全局
```

可用的**PCO**覆盖字段：

- `primary_dns_server_address` - 主DNS服务器IP（IPv4）
- `secondary_dns_server_address` - 次DNS服务器IP（IPv4）
- `primary_dns_server_ipv6_address` - 主DNS服务器IP（IPv6，容器0x0003）
- `secondary_dns_server_ipv6_address` - 次DNS服务器IP（IPv6，容器0x0003）
- `primary_nbns_server_address` - 主WINS服务器IP
- `secondary_nbns_server_address` - 次WINS服务器IP
- `p_cscf_ipv4_address_list` - P-CSCF服务器IP的IPv4列表（用于IMS，容器0x000C） - 参见 PCO配置 和 P-CSCF监控 以获取动态P-CSCF发现
- `p_cscf_ipv6_address_list` - P-CSCF服务器IP的IPv6列表（用于IMS，容器0x0001）
- `ipv4_link_mtu_size` - MTU大小（以字节为单位）

每规则的**P-CSCF**发现：

除了PCO覆盖外，UPF选择规则可以指定动态P-CSCF发现：

p_cscf_discovery_fqdn - （字符串）用于基于DNS的P-CSCF发现的FQDN（例如，"pcscf.mnc001.mcc001.3gppnetwork.org"）

当设置此参数时：

1. PGW-C 在会话建立期间对指定的 FQDN 执行 DNS 查找
2. DNS 服务器返回 P-CSCF IP 地址列表
3. 发现的 P-CSCF 地址通过 PCO 发送给 UE
4. 如果 DNS 查找失败，则回退到 PCO 覆盖中的 p_cscf_ipv4_address_list（如果指定）或全局 PCO 配置
5. 有关监控发现成功/失败率，请参见 P-CSCF 监控

这对于以下情况特别有用：

IMS APNs - 不同的 IMS 网络具有不同的 P-CSCF 服务器
多租户部署 - 不同企业拥有专用的 P-CSCF 基础设施
地理路由 - DNS 根据 UE 位置返回最近的 P-CSCF
高可用性 - DNS 自动仅返回健康的 P-CSCF 服务器

示例：具有自定义 P-CSCF 的 IMS 流量：

```
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 发现: 动态查询 DNS 以获取 P-CSCF 地址
    # DNS 查找根据此 FQDN 返回当前 P-CSCF IP
    p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org",
    # IMS 会话获取自定义 P-CSCF 服务器（在 DNS 失败时用作回退）
    pco: %{
      p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"]
      # DNS、NBNS、MTU 将使用主 PCO 配置中的默认值
    }
  }
]
```

每条规则的 **UE IP** 分配：

每个 UPF 选择规则选择 **UE IP** 的分配方式。有三种相互一致的场景：

场景	规则配置	谁分配	使用的池
1. 默认	*(□□□)*	PGW-C（地址发送到 UPF）	APN 键控的 `ue.subnet_map`，回退到 `:default`
2. 命名池	`ue_ip_pool: "name"`	PGW-C（地址发送到 UPF）	命名的 `ue.subnet_map` 池 `"name"`
3. UPF 委派	`upf_assigns_ip: true`	UPF（PFCP CHV4）	UPF 上的一个池

- `upf_assigns_ip` - （布尔值，默认 `false`）当为 `true` 时，PGW-C 发送 UE-IP-Address IE，并带有 **CHV4**（“UP 分配”）标志，而不是显式地址，并从 UPF 的 Created-PDR 中读取分配的地址，位于会话建立响应中。**UPF** 从哪个池中提取完全是 **UPF** 方面的关注（如果 UPF 拥有多个池，它可能会根据 APN / PFCP 网络实例进行键控）—— PGW-C 不会发出任何关于池的信号。
- `ue_ip_pool` - （字符串，可选）当选择此规则时，**PGW-C** 从中分配的 `ue.subnet_map` 池的名称。这将 PGW-C 池与 UPF 通过选择规则关联起来（例如，使拥有/宣传 `100.64.20.0/24` 的 UPF 仅从该池中接收地址）。一个池可以由一个规则或多个规则引用。`nil` → 默认的 APN 键控 `ue.subnet_map` 行为（场景 1）。

`upf_assigns_ip` 和 `ue_ip_pool` 是互斥的 — 一个将分配委托给 UPF，另一个则保留在 PGW-C 中。如果规则同时设置了这两个，配置加载日志将记录警告，并且 `upf_assigns_ip` 优先（`ue_ip_pool` 被忽略）。

无论是 PGW-C 从与 UPF 绑定的池中分配（场景 2） □□ UPF 委派（场景 3），都允许每个 UPF 拥有一个子网并宣传一个单一的 OSPF 聚合（例如 `100.64.20.0/24`），而不是每个 UE 每次附加的 `/32` — 聚合在 UPF 启动时即存在，因此新附加的 UE 的下行流量立即可路由，消除了每个 UE 路由收敛的黑洞窗口。

> 重要 — 池/**UPF** 对齐：分配给 UE 的池必须落在承载它的 UPF 宣传的聚合内。对于场景 2，将每个 UPF 池绑定到其 `upf_pool` 为该单个 UPF 的规则。对于场景 3，每个 `upf_pool` UPF 必须配置其自己的不重叠 `ueip_pool`（omniupf `feature_ueip: true`）宣传该聚合（`route_advertise_pools`）。如果 UE 降落在一个 UPF 上，但其地址落在另一个 UPF 的聚合内，则下行会黑洞。

示例 — 场景 **2**（**PGW-C** 从规则绑定的每个 **UPF** 池中分配）：

```
# ue.subnet_map 定义命名池:
ue: %{
  subnet_map: %{
    "ims_upf03" => ["100.64.20.0/24"],
    "ims_upf04" => ["100.64.21.0/24"],
    default:     ["100.64.16.0/24"]
  }
},
# 每个规则固定一个 UPF 及其池:
upf_selection: %{
  rules: [
    %{name: "IMS→upf03", priority: 21, match_field: :apn,
match_regex: "^ims",
      upf_pool: [%{remote_ip_address: "10.179.1.42", remote_port:
8805, weight: 1}],
      ue_ip_pool: "ims_upf03"},
    %{name: "IMS→upf04", priority: 20, match_field: :apn,
match_regex: "^ims",
      upf_pool: [%{remote_ip_address: "10.179.1.45", remote_port:
8805, weight: 1}],
      ue_ip_pool: "ims_upf04"}
  ]
}
```

示例 — 场景 **3**（**UPF** 通过 **CHV4** 分配 **UE IP**）：

```
rules: [
  %{
    name: "IMS",
    priority: 20,
    match_field: :apn,
    match_regex: "^ims",
    upf_pool: [
      %{remote_ip_address: "10.179.1.42", remote_port: 8805,
weight: 50},
      %{remote_ip_address: "10.179.1.45", remote_port: 8805,
weight: 50}
    ],
    upf_assigns_ip: true   # UPF 分配 UE IP（CHV4）；池在 UPF 侧
  }
]
```

示例：具有自定义 **DNS** 的企业流量：

```
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}
    ],
    # 企业会话获取公司 DNS 和自定义 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 将使用主 PCO 配置中的默认值
    }
  }
]
```

示例：完全覆盖（所有 **PCO** 字段）：

```
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 会话获取完全自定义的 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: [],  # IoT 无 P-CSCF
      ipv4_link_mtu_size: 1280       # 对于受限设备的较小 MTU
    }
  }
]
```

用例：

- **IMS/VoLTE**：为语音服务提供运营商特定的 P-CSCF 服务器
- 企业 **APNs**：通过公司 DNS 服务器路由企业流量
- **IoT/M2M**：使用公共 DNS 和优化的 MTU 以适应低带宽设备
- 漫游：为访问用户提供本地 DNS 服务器
- 服务差异化：每种服务类型的不同网络参数

基于 **DNS** 的 **UPF** 选择：

启用基于用户位置信息（ULI）的动态 UPF 选择，使用 DNS NAPTR 查询。DNS 设置现在在 `upf_selection` 部分中配置。

注意：这提供基于地理或拓扑的 UPF 选择。有关与动态发现的 UPF 设置 PFCP 关联的更多信息，请参见 PFCP 接口 和 会话管理 以获取会话建立流程。

```
upf_selection: %{
  # 用基于 DNS 的选择
  dns_enabled: true,

  # 按优先顺序查询的位置类型
  dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],

  # 3GPP NAPTR 查询的 DNS 后缀
  dns_suffix: "epc.3gppnetwork.org",

  # DNS 查询超时（毫秒）
  dns_timeout_ms: 5000,

  # ... 规则和 fallback_pool ...
}
```

基于 DNS 的选择工作如下：

1. 优先级：仅当 没有静态规则匹配（优先级较低）时使用 DNS 选择
2. 查询生成：根据 UE 位置构建 DNS NAPTR 查询：
   - ECGI 查询：`eci-<hex>.ecgi.epc.mnc<MNC>.mcc<MCC>.epc.3gppnetwork.org`
   - TAI ���询：`tac-lb<hex>.tac-hb<hex>.tac.epc.mnc<MNC>.mcc<MCC>.epc.3gppnetwork.org`
   - RAI、SAI、CGI 查询遵循类似的 3GPP TS 23.003 格式
3. 回退层次：按优先顺序尝试每个位置类型，直到找到匹配
4. 对等匹配：DNS 结果会根据配置的对等列表进行过滤
5. 选择：选择匹配的对等（当前为第一个匹配，基于负载的选择即将到来）

示例 **DNS** 记录（在您的 DNS 服务器上配置）：

```
; NAPTR 记录用于 PLMN 001-001 中的 TAC 100
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.

; UPF 的 A 记录
upf-edge-1.example.com IN A 10.100.1.21
```

用例：

多接入边缘计算（MEC）：将会话路由到地理上最近的边缘 UPF
动态 UPF 发现：添加/删除 UPF 而无需重新配置 PGW-C
负载均衡：根据位置在 UPF 之间分配负载
网络切片：根据位置将不同的切片路由到不同的 UPF

UPF 健康监控

自动健康感知选择：PGW-C 持续监控所有 UPF 的健康状况，并自动将不健康的 UPF 排除在选择之外。

健康检查标准

当满足以下所有条件时，UPF 被视为 健康：

1. PFCP 关联活动：UPF 已建立 PFCP 关联
2. 心跳响应性：少于 3 次连续未响应的心跳
3. 进程存活：UPF 对等的 GenServer 进程正在运行

如果以下任何条件为真，则 UPF 被视为 不健康：

PFCP 关联未建立（associated: false）
3 次或更多连续心跳超时
UPF 对等进程崩溃或无响应

监控机制

对于配置的 UPF（在 upf_selection 中）：

健康跟踪在启动时立即开始
PFCP 关联持续监控
每 5 秒发送一次心跳
missed_heartbeats_consecutive 计数器跟踪连续失败
所有来自规则和回退池的 UPF 会自动注册

对于 DNS 发现的 UPF（动态注册）：

假定健康，直到第一次会话尝试

- 第一次使用时自动注册
- 注册后开始健康跟踪

选择行为

主动/备用模式（使用 `weight: 0` 时）：

1. 过滤仅健康的 UPF
2. 分为 主动（权重 > 0）和 备用（权重 == 0）
3. 如果有任何健康的 UPF，则使用主动 UPF
4. 如果所有主动 UPF 都不健康，则激活备用 UPF（视为权重 1）
5. 如果没有健康的 UPF，则回退到完整池

负载均衡模式（所有权重 > 0）：

1. 过滤仅健康的 UPF
2. 在健康的 UPF 中执行加权随机选择
3. 如果没有健康的 UPF，则回退到完整池

日志记录：

```
[debug] 使用主动 UPF 池（2/3 健康 UPF，1 备用）
[info] 所有主动 UPF 均已关闭，激活备用 UPF（1 备用 UPF，将权重 0 视为 1）
[warning] 池中没有健康的 UPF（共 3 个），使用完整池作为回退
```

检查 **UPF** 健康

以编程方式：

```
# 检查特定 UPF 是否健康
iex> PGW_C.PFCP_Node.is_peer_healthy?({10, 100, 1, 21})
true

# 获取详细的健康信息
iex> PGW_C.PFCP_Node.get_peer_health({10, 100, 1, 21})
%{
  associated: true,
  missed_heartbeats: 0,
  healthy: true,
  registered: true
}
```

通过 **OAM API**:

- 查询 `/api/upf_selection` 端点: `curl -k https://localhost:8443/api/upf_selection`
- 返回每个池中所有 UPF 的实时健康状态
- 状态值: Active-UP、Standby-Ready、Active-DOWN、Not Associated
- 角色值: ACTIVE（权重 > 0）、STANDBY（权重 == 0）、DYNAMIC（DNS 发现，不在配置中）
- 报告与关联 UPF 的心跳丢失计数器

健康监控最佳实践

1. 在 **upf_selection** 中配置 **UPF**: 所有规则和回退池中的 UPF 会自动监控

```
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}
  ]
}
# 所有 UPF 自动获得:
# - 5 秒心跳
# - □动时的健康监控
# - 自动生成的名称
```

2. 使用备用 **UPF**：配置热备用，权重为 0 以实现自动故障转移

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

3. 通过 **OAM API** 监控：定期轮询 `/api/upf_selection` 端点以检查 UPF 健康状态

4. 心跳监控：系统使用 3 次连续未响应心跳的固定阈值来确定对等健康。

动态 **UPF** 注册

功能：PGW-C 自动注册和监控通过 DNS 发现的 UPF，即使它们不在 `upf_selection` 配置中。

工作原理

当任何选择方法（静态规则、池或 DNS）返回一个尚未注册的 UPF 时，系统会自动：

1. 创建 **PFCP** 对等：为未知 UPF 生成默认对等配置
2. 发起 **PFCP** 关联：尝试与 UPF 建立 PFCP 关联
3. 注册到对等注册表：将 UPF 添加到内部对等跟踪系统
4. 开始心跳监控：开始定期心跳交换（10 秒间隔）
5. 跟踪存活状态：监控 UPF 的故障和恢复

动态 **UPF** 的默认配置

当 UPF 被动态注册时，它会收到以下默认配置：

```
%{
  name: "Dynamic-UPF-<IP>",                # 例如 "Dynamic-UPF-10-100-
1-21"
  remote_ip_address: <discovered_ip>,      # 从 DNS 或选择中获取的 IP
  remote_port: 8805,                       # 标准 PFCP 端口（可覆盖）
  initiate_pfcp_association_setup: true,   # PGW-C 发起关联
  heartbeat_period_ms: 10_000              # 10 秒心跳间隔
}
```

> 注意：动态 UPF 仅用于关联管理。它们在 `upf_selection` 规则中作为目标使用，而不是作为选择逻辑的来源。

示例：**DNS** 返回未知 **UPF**

```
# DNS 查询返回: upf-edge-2.example.com -> 10.200.5.99
# 此 UPF 不在您的 upf_selection 配置中

# 动态注册流程:
# 1. 系统检测到未知 UPF 10.200.5.99
# 2. 日志: “UPF {10, 200, 5, 99} 未预配置，尝试动态注册...”
# 3. 向 10.200.5.99:8805 发送 PFCP 关联设置请求
# 4. 如果 UPF 响应：关联建立，会话正常继续
# 5. 如果 UPF 不响应：会话优雅失败并显示清晰的错误消息
```

好处

□ □□□□□□□□□□□ DNS 的 UPF 选择现在无需预配置即可工作
□ □□□□□□□ UPF 添加到网络中而无需重新启动 PGW-C
□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□
□ □□□□□□□□□□ UPF 继续按原样工作
□ □□□□□□□□ UPF 获得与静态对等相同的心跳监控

故障处理

如果动态发现的 UPF 未能响应 PFCP 关联设置：

```
[error] PFCP 关联设置失败，动态 UPF {10, 200, 5, 99}: :timeout
[error] 动态注册 UPF {10, 200, 5, 99} 失败: :timeout。
        会话创建将失败。考虑将此 UPF 添加到 upf_selection 配置中。
```

会话创建将失败，但 PGW-C 保持稳定并继续处理其他会话。

何时预配置与动态注册

场景	建议
生产核心 **UPF**	在 `upf_selection` 中预配置（显式配置，从启动时监控）
DNS 发现的边缘 **UPF**	使用动态注册（与基础设施自动扩展）
测试/开发 **UPF**	任何方法均可（动态更方便）
关键任务 **UPF**	在 `upf_selection` 中预配置（确保从启动时监控）
短暂/自动扩展 **UPF**	使用动态注册（UPF 动态出现和消失）

监控动态 **UPF**

动态 UPF 在日志中以其自动生成的名称出现：

```
[info] 为动态 PFCP 对等配置创建动态-UPF-10-200-5-99 ({10, 200, 5, 99}:8805)
[info] 动态 UPF 对等动态-UPF-10-200-5-99 成功注册，PID #PID<0.1234.0>
```

您可以查询对等注册表以查看所有注册的对等（静态和动态）：

```
# 获取所有注册的对等
PGW_C.PFCP_Node.registered_peer_count()

# 检查特定 UPF 是否已注册
PGW_C.PFCP_Node.get_peer({10, 200, 5, 99})
# 返回: {:ok, #PID<0.1234.0>} 如果已注册，否则返回 :error
```

动态 **UPF** 的自定义端口

如果您的 UPF 使用非标准 PFCP 端口，您可以手动触发注册：

```
# 在自定义端口注册 UPF
PGW_C.PFCP_Node.register_dynamic_peer({10, 200, 5, 99}, 9999)
```

但是，基于 DNS 的选择和自动注册始终使用端口 8805（标准 PFCP 端口）。

UPF 选择干运行模式：

测试和验证您的 UPF 选择配置，而不影响真实会话：

```
config :pgw_c,
  # 用干运行模式进行测试（默认禁用）
  upf_selection_dry_run: true
```

当启用干运行模式时：

1. 不进行真实分配：会话不会实际分配给 UPF
2. 详细日志记录：选择决策将记录完整细节
3. 错误返回：`assign_sxb_peer/1` 返回 `{:error, :dry_run_mode}`
4. 会话预防：返回错误会阻止会话创建
5. 两种方法均可：适用于静态规则和基于 DNS 的选择

日志输出示例：

```
[warning] ⚠  UPF 选择干运行模式已□用 - 不会发生实际分配

[info] □ 干运行：静态规则匹配
  方法：静态规则
  匹配字段：:apn
  匹配正则：~r/^internet\./
  优先级：10
  选择的 UPF：10.0.1.21:8805

[warning] □ 干运行：将分配 UPF，但跳过实际分配
  会话 IMSI：001001000000670
  会话 APN：internet.apn
  选择方法：静态
  将链接到：10.0.1.21:8805
  ⚠  返回错误以防止会话创建
```

通过 **LiveView UI** 测试：

UPF 选择 LiveView 页面（`/api/upf_selection`）包含交互式测试界面：

1. 在 Web 面板中导航到 UPF 选择页面
2. 滚动到“测试 UPF 选择”部分
3. 输入测试会话属性：
   - IMSI（例如，`001001000000670`）
   - APN（例如，`internet.apn`）
   - 服务网络 PLMN ID（例如，`001001`）
4. 点击“测试选择”以模拟选择
5. 查看详细结果，显示：
   - 匹配的规则（或是否将使用 DNS）
   - 选择的 UPF 和对等名称
   - 匹配字段和模式详细信息
   - 规则优先级

LiveView 测试界面模拟选择逻辑而无需全局启用干运行模式，使其在生产环境中安全测试而不影响真实流量。

用例：

- 配置测试：在部署到生产之前验证路由规则

- 故障排除：了解特定会话为何路由到特定 UPF
- 培训：向运营团队演示 UPF 选择逻辑
- 开发：在开发期间测试新的选择规则

匹配字段：

- `:imsi` - 国际移动用户身份
- `:apn` - 接入点名称（APN/DNN）
- `:serving_network_plmn_id` - 服务网络 PLMN ID
- `:sgw_ip_address` - SGW IP 地址
- `:uli_tai_plmn_id` - 跟踪区域 PLMN ID
- `:uli_ecgi_plmn_id` - E-UTRAN 小区 PLMN ID

心跳调优：

```
# 激进（快速检测故障）
heartbeat_period_ms: 2_000  # 2 秒

# 标准（推荐）
heartbeat_period_ms: 5_000  # 5 秒

# 放松（高延迟网络）
heartbeat_period_ms: 10_000  # 10 秒
```

参见： PFCP 接口文档

UE IP 池配置

目的

配置用于 UE 分配的 IP 地址池，按 APN 组织。

配置块

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # APN "internet" 池
      "internet" => [
        "100.64.0.0/20"  # 4094 可用 IP
      ],

      # APN "ims" 池
      "ims" => [
        "100.64.16.0/22"  # 1022 可用 IP
      ],

      # 默认池用于未知 APN
      default: [
        "42.42.42.0/24"  # 254 可用 IP
      ]
    }
  }
```

参数

参数	类型	必需	描述
`subnet_map`	Map	是	将 APN 名称映射到子网列表
`default`	List (子网)	是	用于未知 APN 的后备池

子网格式

CIDR 表示法: `<network_address>/<prefix_length>`

可用 **IP** 计算:

CIDR	总 IP 数	可用 IP 数	示例范围
/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

示例

简单配置:

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

生产配置:

```
ue: %{
  subnet_map: %{
    # 通用互联网 - 大池
    "internet" => [
      "100.64.0.0/18"  # 16,382 IP
    ],

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

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

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

    # 默认后备
    default: [
      "42.42.42.0/24"  # 254 IP
    ]
  }
}
```

负载均衡（每个 **APN** 多个子网）：

```
ue: %{
  subnet_map: %{
    "internet" => [
      "100.64.0.0/22",   # 1022 IP
      "100.64.4.0/22",   # 1022 IP
      "100.64.8.0/22",   # 1022 IP
      "100.64.12.0/22"   # 1022 IP
    ],
    # 总计: 4088 IP, 随机分配
    default: ["42.42.42.0/24"]
  }
}
```

正则表达式模式匹配:

对于通配符 APN 匹配，使用以 `^` 开头的键来指示正则表达式模式:

```
ue: %{
  subnet_map: %{
    # 正则: 以 "ims" 开头的 APN (匹配 "ims", "ims.apn", "ims.something")
    "^ims" => [
      "100.64.10.0/24"
    ],

    # 正则: 以 "m2m." 开头的 APN (匹配 "m2m.test", "m2m.prod")
    "^m2m\." => [
      "100.64.20.0/24"
    ],

    # 仅精确匹配
    "enterprise.corp" => [
      "10.100.0.0/16"
    ],

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

模式规则:

- 以 `^` 开头的键是正则表达式模式

- 没有 `^` 的键是精确匹配（向后兼容）
- 正则表达式模式在配置解析时编译
- 第一个匹配的模式优先
- 反斜杠必须转义（`\` 表示 `\`）

常见模式：

- 以...开头：`"^ims"` 匹配 `ims`，`ims.apn`，`ims.foo.bar`
- 以...结尾：`"^.*\.corp$"` 匹配 `foo.corp`，`bar.corp`
- 包含：`"^.*test.*"` 匹配 `test`，`foo.test.bar`

IPv6 支持：

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

    # IPv6 池（前缀委派）
    "internet.ipv6" => [
      "2001:db8:1::/48"
    ],

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

推荐的 IP 范围

RFC 6598（运营商级 **NAT**）：

- 范围：`100.64.0.0/10`
- 大小：~400 万 IP
- 目的：为服务提供商 NAT 设计

私有 IP 范围（RFC 1918）：

10.0.0.0/8 - 1600 万 IP
172.16.0.0/12 - 100 万 IP 192.168.0.0/16 - 65,534 IP

参见：UE IP 池分配文档

PCO 配置

目的

配置发送给 UE 的协议配置选项（PCO）。

配置块

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

    # IPv6 DNS 服务器（可选，容器 0x0003）
    primary_dns_server_ipv6_address: nil,
    secondary_dns_server_ipv6_address: nil,

    # NBNS 服务器（可选，适用于 Windows 设备）
    primary_nbns_server_address: nil,
    secondary_nbns_server_address: nil,

    # IMS 的 P-CSCF 地址
    p_cscf_ipv4_address_list: ["10.0.0.50", "10.0.0.51"],   # 容器 0x000C
    p_cscf_ipv6_address_list: [],                           # 容器 0x0001

    # IPv4 MTU 大小
    ipv4_link_mtu_size: 1400
  }
```

参数

参数	类型	默认值	描述
`primary_dns_server_address`	字符串 (IPv4)	必需	主 DNS 服务器
`secondary_dns_server_address`	字符串 (IPv4)	可选	次 DNS 服务器
`primary_dns_server_ipv6_address`	字符串 (IPv6)	`nil`	主 IPv6 DNS 服务器 (容器 0x0003)
`secondary_dns_server_ipv6_address`	字符串 (IPv6)	`nil`	次 IPv6 DNS 服务器 (容器 0x0003)
`primary_nbns_server_address`	字符串 (IPv4)	`nil`	主 NBNS (NetBIOS) 服务器
`secondary_nbns_server_address`	字符串 (IPv4)	`nil`	次 NBNS 服务器
`p_cscf_ipv4_address_list`	列表 (IPv4)	`[]`	IMS 的 IPv4 P-CSCF 地址 (容器 0x000C)
`p_cscf_ipv6_address_list`	列表 (IPv6)	`[]`	IMS 的 IPv6 P-CSCF 地址 (容器 0x0001)
`ipv4_link_mtu_size`	整数	`1400`	最大传输单元大小

示例

公共 **DNS** (谷歌)：

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

□□ DNS□

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

IMS □□□

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

  # IMS/VoLTE □ P-CSCF
  p_cscf_ipv4_address_list: [
    "10.0.0.50",  # □ P-CSCF
    "10.0.0.51"   # □ P-CSCF
  ],

  ipv4_link_mtu_size: 1400
}
```

NBNS□Windows □□□□□

```
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 调优:

```
# 标准以太网
ipv4_link_mtu_size: 1500

# 为隧道开销减少
ipv4_link_mtu_size: 1400

# Jumbo 帧（如果支持）
ipv4_link_mtu_size: 9000
```

参见: PCO 配置文档

OAM API 配置

目的

配置用于管理和监控 OmniPGW 的 OAM REST API 端点。所有监控和管理都通过此 API 进行。

> 注意: API 配置仅在非测试环境中有效。当 `config_env()` 为 `:test`、`:test_mock` 或 `:test_impl` 时，配置将自动跳过。

REST API 配置

```
config :api_ex,
  api: %{
    # 网络设置
    port: 8443,
    listen_ip: "0.0.0.0",

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

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

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

API 参数

参数	类型	默认值	描述
`port`	整数	`8443`	REST API 的 HTTPS 端口
`listen_ip`	字符串（IP）	`"0.0.0.0"`	API 监听地址（0.0.0.0 = 所有接口）
`product_name`	字符串	`"PGW-C"`	产品名称（用于 API 元数据）
`title`	字符串	`"API - PGW-C"`	API 文档标题
`hostname`	字符串	`"localhost"`	API 服务器主机名
`enable_tls`	布尔值	`true`	启用 API 的 HTTPS
`tls_cert_path`	字符串（路径）	必需	SSL/TLS 证书文件路径
`tls_key_path`	字符串（路径）	必需	SSL/TLS 私钥文件路径
`routes`	列表（映射）	`[]`	API 路由定义

路由定义格式

`routes` 列表中的每个路由都是一个包含以下内容的映射：

字段	类型	描述	示例
`path`	字符串	路由的 URL 路径	`"/status"`
`module`	模块	处理请求的控制器模块	`ApiEx.Api.StatusController`
`actions`	列表（原子）	允许的控制器操作	`[:index, :show]`

示例

默认配置（开发）：

```
# API 在 localhost:8443
config :api_ex,
  api: %{
    port: 8443,
    listen_ip: "127.0.0.1",  # 仅限本地主机
    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]}
    ]
  }
```

生产配置:

```
# API 在管理接口
config :api_ex,
  api: %{
    port: 8443,
    listen_ip: "10.0.0.20",  # 管理网络
    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]}
    ]
  }
```

访问 API

查询端点：

```
# 状态端点（自签名证书，开发）
curl -k https://localhost:8443/api/status

# 使用受信任的证书（生产）
curl https://pgw-c-api.example.com:8443/api/status
```

所有路由都在 `/api` 前缀下提供。可用的端点包括 `/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` 和 `/api/upf_selection`。

交互式 **API** 文档（**Swagger UI**）：

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

TLS 证书设置

生成自签名证书（开发）：

```
# 生成私钥和证书
openssl req -x509 -newkey rsa:4096 -keyout priv/cert/omnitouch.pem
\
  -out priv/cert/omnitouch.crt -days 365 -nodes \
  -subj "/CN=localhost"
```

生产证书：

对于生产环境，使用受信任的证书颁发机构（CA）提供的证书：

- Let's Encrypt（免费，自动化）
- 商业 CA（DigiCert、GlobalSign 等）
- 企业部署的内部 CA

安全考虑

1. 在生产中始终使用 **TLS** - 设置 `enable_tls: true`
2. 限制 **listen_ip** - 在生产中使用特定的 IP 地址（而不是 0.0.0.0）
3. 使用有效的证书 - 避免在生产中使用自签名证书
4. 防火墙保护 - 限制对管理端口（8443）的访问
5. 强密钥文件 - 使用 4096 位 RSA 或等效
6. 定期更新 - 在证书到期前轮换证书

故障排除

问题：无法访问 **API**

```
# 检查端口是否在监听
netstat -tulpn | grep 8443

# 检查证书文件是否存在
ls -la priv/cert/

# 检查日志以获取□动错误
tail -f /var/log/pgw_c/application.log
```

问题：**SSL/TLS** 错误

- 验证证书和密钥路径是否正确
- 确保证书与主机名匹配
- 检查证书到期：`openssl x509 -in cert.crt -text -noout`
- 验证密钥文件权限（应可被 PGW-C 进程读取）

参见：监控指南 以获取 OAM API 使用详细信息

完整示例

生产就绪配置

```
# config/runtime.exs
import Config

# 日志配置
config :logger, level: :info

config :pgw_c,
  # 指标（Prometheus）
  metrics: %{
    enabled: true,
    ip_address: "10.0.0.20",  # 管理网络
    port: 9090,
    registry_poll_period_ms: 5_000
  },

  # Diameter/Gx（PCRF 接口）
  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 接口）
  s5s8: %{
```

```
    local_ipv4_address: "10.0.0.20"
  },

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

  # UPF □□
  upf_selection: %{
    rules: [
      %{
        name: "□□□□□",
        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}  # □□
        ]
      }
    ],
    fallback_pool: [
      %{remote_ip_address: "10.0.0.21", remote_port: 8805, weight:
100}
    ]
  },

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

```
      ]
    }
  },

  # 协议配置选项
  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
  }
```

配置验证

启动验证

OmniPGW 在启动时验证配置。检查日志：

```
[info] 从 runtime.exs 加载配置
[info] 验证配置...
[info] 配置有效
[info] □动 OmniPGW...
```

常见验证错误

无效的 **IP** 地址：

```
[error] s5s8.local_ipv4_address 中的无效 IP 地址："10.0.0"
```

缺少必需字段：

```
[error] 缺少必需配置：sxb.local_ip_address
```

无效的 **CIDR**：

```
[error] ue.subnet_map 中的无效子网: "100.64.1.0/33"
```

无效的 **Diameter** 身份:

```
[error] Diameter 主机必须是 FQDN, 而不是 IP: "10.0.0.20"
```

配置测试

测试配置而不启动:

```
# 验证语法
mix compile

# 检查配置加载
iex -S mix
iex> Application.get_env(:pgw_c, :metrics)
```

□□□□□□

□□

```
# 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
  }
```

□□□□□□

```
# 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
      }
    ]
  }
  # ... □□□□
```

□□□

```
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
```

相关文档

接口配置

- **PFCP** 接口 - Sxb/PFCP 配置，UPF 通信，会话建立
- **Diameter Gx** 接口 - PCRF 策略控制，PCC 规则，QoS 管理
- **Diameter Gy** 接口 - OCS 在线计费，配额管理，信用控制
- **S5/S8** 接口 - GTP-C v2 配置，SGW-C 通信（4G/LTE）
- **Gn/Gp** 接口 - GTP-C v1 配置，SGSN 通信（2G/3G GGSN）

网络配置

- **UE IP** 分配 - IP 池管理，基于 APN 的分配，DHCP
- **PCO** 配置 - 协议配置选项，DNS，P-CSCF，MTU
- **P-CSCF** 监控 - P-CSCF 发现监控，IMS 健康跟踪

□作指南

- 会话管理 - PDN 会话生命周期，承载管理
- 监控指南 - Prometheus 指标，警报，仪表板
- 数据 **CDR** 格式 - 离线计费记录，计费集成

返回□作指南

数据计费数据记录（CDR）格式

PGW-C 的离线计费

OmniPGW □ Omnitouch □□□□□□

目录

1. □述
2. CDR 文件格式
3. CDR 字段
4. CDR 事件
5. 文件结构
6. 配置
7. CDR 生成流程
8. 字段详细信息
9. 示例
10. 集成

概述

数据 **CDR**（计费数据记录）格式为数据包网关控制平面（PGW-C）提供离线计费功能。CDR 用于记录承载会话事件、数据使用情况和用户信息，以便进行计费和分析。

该通用格式与 SGW-C CDR 兼容，确保 EPC 基础设施中计费记录的一致性。

主要特点

- 基于 **CSV** 的格式 - 简单、易读的逗号分隔值
- 基于事件的记录 - 捕获承载开始、更新和结束事件
- 流量计量 - 记录上行和下行数据使用情况

- 自动轮换 - 基于时间间隔的可配置文件轮换
- 符合 **3GPP** 标准 - 遵循 3GPP TS 32.251（PS 域计费）和 TS 32.298（CDR 编码）

用例

用例	描述
离线计费	生成后付费计费的 CDR
分析	分析用户使用模式
审计跟踪	跟踪所有承载会话事件
容量规划	监控网络资源利用率
故障排除	调试会话和承载问题

CDR 文件格式

文件命名约定

```
<epoch_timestamp>
```

示例：

```
1726598022
```

文件名为文件创建时的 Unix 纪元时间戳（以秒为单位）。

文件位置

默认目录：

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

可通过 `cdr_directory` 参数在 config/runtime.exs 中配置。

文件头

每个 CDR 文件以包含元数据的多行头开始：

```
# 数据 CDR 文件：
# 文件开始时间：HH:MM:SS (unix_timestamp)
# 文件结束时间：HH:MM:SS (unix_timestamp)
# 网关名称：<gateway_name>
#
epoch,imsi,event,charging_id,msisdn,ue_imei,timezone_raw,plmn,tac,eci
```

头字段：

- 文件开始时间 - CDR 文件创建的时间（可读和 Unix 时间戳）
- 文件结束时间 - 文件轮换将发生的时间（可读和 Unix 时间戳）
- 网关名称 - PGW-C 实例的标识符（通过 `pgw_name` 参数配置）
- 列头 - 数据记录的 CSV 字段名称

CDR 字段

字段摘要

位置	字段名称	类型	描述
0	epoch	整数	事件时间戳（Unix 纪元秒）
1	imsi	字符串	国际移动用户身份
2	event	字符串	CDR 事件类型（例如，“default_bearer_start”）
3	charging_id	整数	承载的唯一计费标识符
4	msisdn	字符串	移动台 ISDN 号码（电话号码）
5	ue_imei	字符串	国际移动设备身份
6	timezone_raw	字符串	UE 时区（保留，目前为空）
7	plmn	整数	公共陆地移动网络标识符
8	tac	整数	跟踪区域代码
9	eci	整数	E-UTRAN 小区标识符
10	sgw_ip	字符串	SGW-C S5/S8 控制平面 IP 地址
11	ue_ip	字符串	UE IP 地址（IPv4|IPv6 格式）
12	pgw_ip	字符串	PGW-C S5/S8 控制平面 IP 地址
13	apn	字符串	接入点名称
14	qci	整数	QoS 类标识符

位置	字段名称	类型	描述
15	octets_in	整数	下行数据量（字节）
16	octets_out	整数	上行数据量（字节）

CDR 事件

事件类型

CDR 为三种类型的事件生成：

事件类型	格式	描述	生成时机
承载开始	`<type>_bearer_start`	承载建立	发送创建会话响应
承载更新	`<type>_bearer_update`	会话中的使用报告	来自用户平面的定期使用报告
承载结束	`<type>_bearer_end`	承载终止	删除会话请求/响应

承载类型：

- `default` - 默认承载（每个 PDN 连接一个）
- `dedicated` - 专用承载（每个 PDN 连接零个或多个）

事件示例

```
default_bearer_start       - 默认承载已建立
default_bearer_update      - 默认承载使用更新
default_bearer_end         - 默认承载已终止
dedicated_bearer_start     - 专用承载已建立
dedicated_bearer_update    - 专用承载使用更新
dedicated_bearer_end       - 专用承载已终止
```

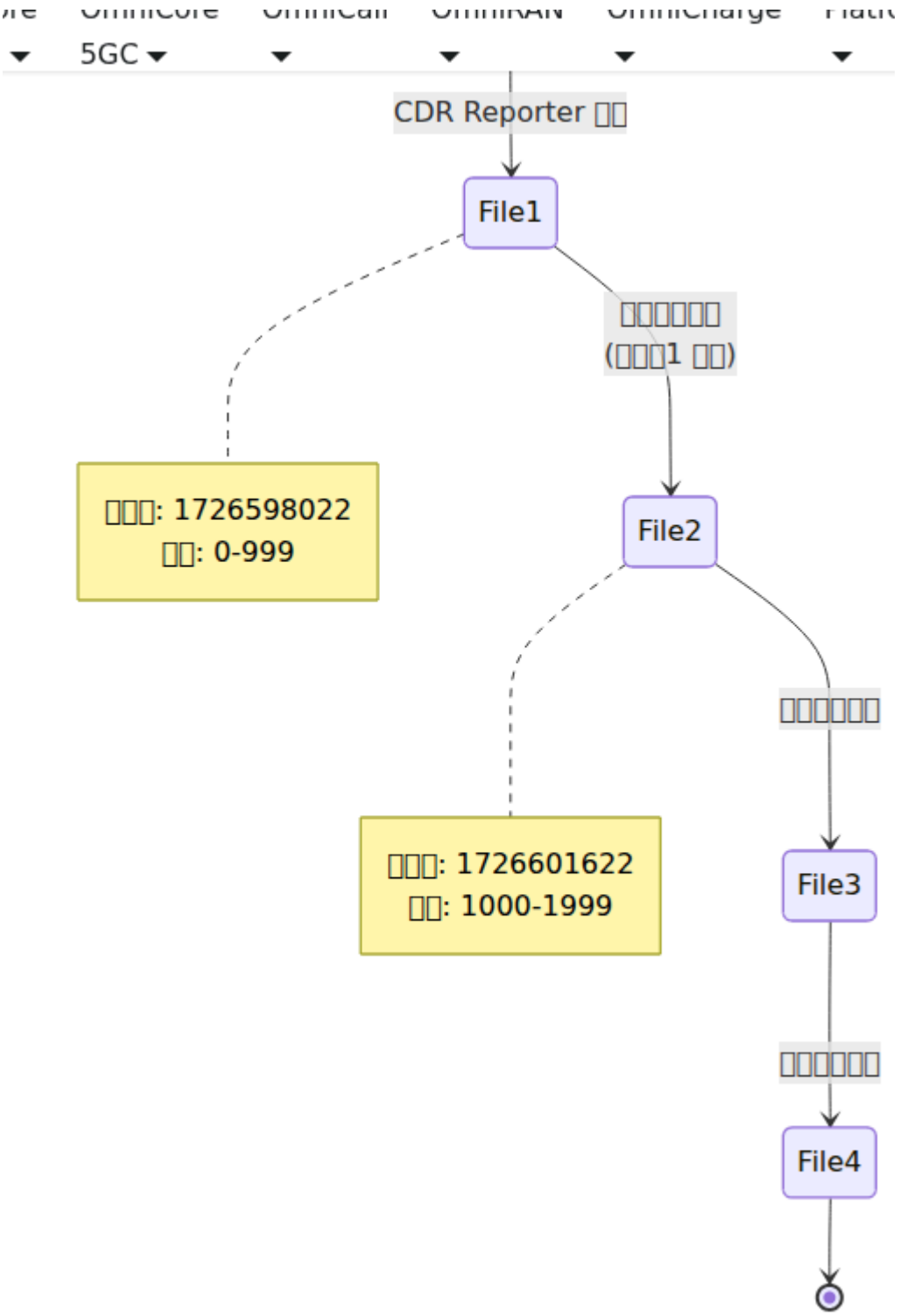
文件结构

示例 **CDR** 文件

```
# 数据 CDR 文件:
# 文件开始时间: 18:53:42 (1726598022)
# 文件结束时间: 19:53:42 (1726601622)
# 网关名称: 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
```

文件轮换

CDR 文件根据配置的持续时间自动轮换:



轮换过程：

1. 关闭当前 CDR 文件
2. 创建带有当前时间戳的新文件
3. 将头写入新文件
4. 继续将 CDR 记录到新文件中

□□

□□□□

PGW-C CDR □□□ `config/runtime.exs` □□□□

□□	□□	□□	□□□	□□□
`pgw_name`	□□□	PGW □□□□□ □□□□ CDR ��□□	"omni-pgw01"	□□□□□□□□ ID
`cdr_file_duration`	□□	□□□□□ □□□□□	3600000	3600000□1 □□□
`cdr_directory`	□□□	CDR □□□□□ □	"/tmp/pgw_c"	`/var/log/pgw_c/cdr`
`usage_report_interval`	□□	URR □ □□□□□ □□ - PGW-U □□□ □□□□□ □	60000	60000□1 □□□

□□□□

□□□□ (config/runtime.exs):

```
config :pgw_c,
  # CDR 文件配置
  pgw_name: "omni-pgw01",
  cdr_file_duration: 3_600_000,           # 1 小时
  cdr_directory: "/var/log/pgw_c/cdrs",

  # URR 配置（触发 PGW-U 的使用报告）
  usage_report_interval: 60_000           # 60 秒
```

生产环境：

```
config :pgw_c,
  pgw_name: "pgw-c-prod-01",
  cdr_file_duration: 3_600_000,           # 1 小时轮换
  cdr_directory: "/var/log/pgw_c/cdrs",
  usage_report_interval: 60_000           # 1 分钟更新
```

开发环境：

```
config :pgw_c,
  pgw_name: "pgw-c-dev",
  cdr_file_duration: 300_000,             # 测试用的 5 分钟轮换
  cdr_directory: "/tmp/pgw_c_cdrs",
  usage_report_interval: 30_000           # 更快测试的 30 秒更新
```

高流量：

```
config :pgw_c,
  pgw_name: "pgw-c-prod-heavy",
  cdr_file_duration: 1_800_000,           # 30 分钟轮换
  cdr_directory: "/mnt/fast-storage/cdrs",
  usage_report_interval: 300_000          # 5 分钟更新（减少开销）
```

URR（使用报告规则）

PGW-C 使用 **PFCP URR**（使用报告规则）来触发 PGW-U 的使用报告。当达到 URR 阈值或时间到期时，PGW-U 发送包含使用数据的会话报告请求，从而触发 CDR 生成。

URR 配置的工作原理**:**

1. `usage_report_interval`（以毫秒为单位）转换为 PFCP 时间阈值的秒数
2. PGW-C 在会话建立期间创建带有时间阈值的 URR
3. PGW-U 在配置的间隔内发送定期使用报告
4. 每个使用报告触发一个 `bearer_update` CDR 事件
5. 最终使用报告（在会话删除时）触发 `bearer_end` CDR 事件

示例**:** `usage_report_interval: 60_000` 意味着:

- PGW-U 每 60 秒报告一次使用情况
- 每 60 秒生成 CDR 更新事件
- 为计费提供细粒度的使用跟踪

URR 类型定义**:**

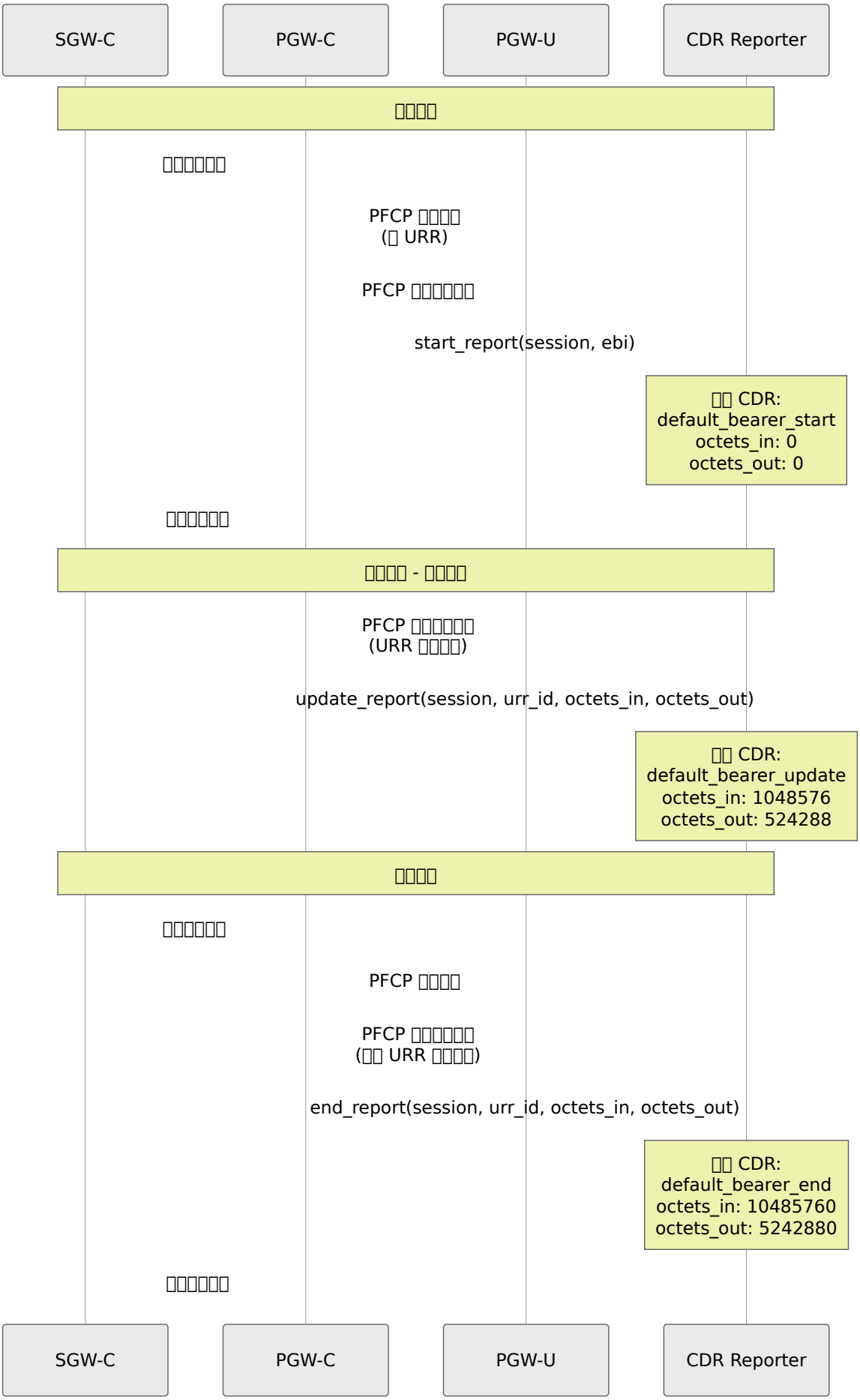
```
defmodule PGW_C.Session.Types.URR do
  typedstruct do
    field :urr_id, non_neg_integer()
    field :measurement_method, :duration | nil
    field :reporting_triggers, :time_threshold | nil
    field :time_threshold, non_neg_integer() | nil  # 秒
  end
end
```

请参阅 PFCP 接口文档 以获取 URR PFCP 详细信息，包括如何在会话建立期间创建 URR。

CDR 生成流程

承载生命周期 CDR 事件

PGW-C CDR 生成**:**



SGW-C
PGW-C
PGW-U
CDR Reporter
会话建立
创建会话请求
PFCP 会话建立
(带 URR)
PFCP 会话建立响应
start_report(session, ebi)
生成 CDR:
default_bearer_start
octets_in: 0
octets_out: 0
创建会话响应
会话活动 - 数据流动
PFCP 会话报告请求
(URR 使用报告)
update_report(session, urr_id, octets_in, octets_out)
生成 CDR:
default_bearer_update
octets_in: 1048576
octets_out: 524288
会话终止
删除会话请求
PFCP 会话删除
PFCP 会话删除响应
(最终 URR 使用报告)
end_report(session, urr_id, octets_in, octets_out)
生成 CDR:
default_bearer_end
octets_in: 10485760
octets_out: 5242880
删除会话响应
SGW-C
PGW-C
PGW-U
CDR Reporter

CDR 生成事件

1. 承载开始:

何时: 发送创建会话响应
目的: 记录承载建立时的零使用
octets_in: 0
octets_out: 0

2. 承载更新:

何时: 从 PGW-U 接收到 PFCP 会话报告请求（URR 使用报告）
目的: 记录增量数据使用情况
octets_in: 自承载开始以来的累计下行字节
octets_out: 自承载开始以来的累计上行字节
触发: URR 时间阈值到期（通过 usage_report_interval 配置）

3. 承载结束:

何时: 从 PGW-U 接收到 PFCP 会话删除响应（带最终使用报告）
目的: 记录会话终止前的最终数据使用情况
octets_in: 最终总下行字节
octets_out: 最终总上行字节

字段详细信息

1. epoch (时间戳)

类型: Unix 纪元时间戳（秒）

描述: CDR 事件发生的时间

示例:

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

---

2. imsi (用户身份)

类型：字符串（最多 15 位数字）

格式：MCCMNC + MSIN

描述：唯一标识用户的国际移动用户身份

示例：

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

来源：UE 上下文���在创建会话请求中接收

---

3. event (CDR 事件类型)

类型：字符串

格式：`<bearer_type>_bearer_<event>`

值：

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

确定：

- 如果 EBI（EPS 承载 ID）等于 LBI（链接承载 ID）：`default`
- 如果 EBI 不等于 LBI：`dedicated`

来源：承载上下文（EBI 与 LBI 比较）

---

4. charging_id（计费标识符）

类型：无符号 32 位整数

描述：用于跨网络元素的计费关联的唯一标识符

示例：

```
12345
```

来源：由 PGW-C 分配，在创建会话响应中接收

用途：

- 关联 SGW 和 PGW 之间的计费事件
- 用于 Diameter Gy/Gz 计费接口
- 每个承载唯一

---

5. msisdn（电话号码）

类型：字符串（E.164 格式）

描述：移动台 ISDN 号码（用户的电话号码）

格式：国家代码 + 国内号码

示例：

```
15551234567
 └┘└──────┘
  CC   国家
  (1) (5551234567)
```

来源：UE 上下文，通常来自 HSS 通过 MME

---

6. ue_imei (设备身份)

类型：字符串（15 位数字）

格式：TAC (8) + SNR (6) + Spare (1)

描述：国际移动设备身份（设备标识符）

示例：

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

来源：UE 上下文，从 MME 接收

---

7. timezone_raw (UE 时区)

类型：字符串（目前保留/为空）

描述：用于 UE 时区信息的保留字段

当前状态：未填充（CSV 中为空字段）

未来用途：可能包括时区偏移和夏令时标志

示例：

```
,  (空字段)
```

---

8. plmn (网络标识符)

类型：整数（遗留格式）

描述：以小端十六进制编码的公共陆地移动网络标识符

编码过程：

```
MCC: 505, MNC: 57
  ↓
"50557"
  ↓
交换对: "055570"
  ↓
十六进制转十进制: 0x055570 = 349552
```

示例：

```
349552  → MCC: 505, MNC: 57
```

来源：从 MME 获取的 UE 位置信息

注意：这是向后兼容的遗留编码格式

---

9. tac (跟踪区域代码)

类型：无符号 16 位整数

描述：跟踪区域代码标识 UE 所在的跟踪区域

范围：0 - 65535

示例：

```
1234
```

来源：UE 位置信息，从 MME 在创建会话请求中接收

用途：

- 标识移动管理区域
- 用于寻呼和位置更新
- TAI（跟踪区域标识）的一部分

---

10. eci (E-UTRAN 小区标识符)

类型：无符号 28 位整数

描述：E-UTRAN 小区标识符唯一标识为 UE 服务的小区

格式：eNodeB ID（20 位） + 小区 ID（8 位）

范围：0 - 268,435,455

示例：

```
5678
```

来源：从 MME 获取的 UE 位置信息

用途：

- 标识特定的小区塔和扇区
- 用于切换和移动管理
- 细粒度位置信息

---

11. sgw_ip (SGW 控制平面 IP)

类型：字符串（IPv4 或 IPv6 地址）

描述：SGW-C 的 S5/S8 控制平面 IP 地址（F-TEID）

格式：点分十进制（IPv4）或冒号十六进制（IPv6）

示例：

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

来源：本地配置，分配给 S5/S8 接口

---

12. ue_ip (UE IP 地址)

类型：字符串（IPv4|IPv6 格式）

描述：分配给 UE 的 PDN 连接的 IP 地址

格式：`<ipv4>|<ipv6>`

示例：

```
172.16.1.100|                          (仅 IPv4)
|2001:db8::1                           (仅 IPv6)
172.16.1.100|2001:db8::1              (双栈)
```

来源：PGW-C 的 PDN 地址分配（PAA）

注意：

- 空 IPv4：未分配 IPv4 地址
- 空 IPv6：未分配 IPv6 地址
- 两者均存在：双栈 PDN 连接

---

13. pgw_ip (PGW 控制平面 IP)

类型：字符串（IPv4 或 IPv6 地址）

描述：PGW-C 的 S5/S8 控制平面 IP 地址（远程 F-TEID）

格式：点分十进制（IPv4）或冒号十六进制（IPv6）

示例：

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

来源：从 PGW-C 的创建会话响应中接收

---

14. apn (接入点名称)

类型：字符串（最多 100 个字符）

描述：标识外部网络（PDN）的接入点名称

格式：类似 DNS 的标签格式

示例：

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

来源：从 MME 的创建会话请求中接收

用途：

- 确定连接到哪个外部网络
- 驱动策略和计费规则
- 可能确定 IP 地址池

---

15. qci (QoS 类标识符)

类型：无符号 8 位整数

描述：QoS 类标识符定义承载的服务质量

范围：1 - 9（标准化），128-254（运营商特定）

标准化 **QCI** 值：

QCI	资源类型	优先级	数据包延迟	数据包丢失	示例服务
1	GBR	2	100 ms	10^-2	对话语音
2	GBR	4	150 ms	10^-3	对话视频
3	GBR	3	50 ms	10^-3	实时游戏
4	GBR	5	300 ms	10^-6	非对话视频
5	Non-GBR	1	100 ms	10^-6	IMS 信令
6	Non-GBR	6	300 ms	10^-6	视频（缓冲）
7	Non-GBR	7	100 ms	10^-3	语音、视频、游戏
8	Non-GBR	8	300 ms	10^-6	视频（缓冲）
9	Non-GBR	9	300 ms	10^-6	默认承载

示例：

```
9  → 默认承载（尽力而为）
```

来源：PGW-C 的承载 QoS 参数

---

16. octets_in (下行流量)

类型: 无符号 64 位整数

描述: 在下行方向（网络 → UE）传输的字节数

单位: 字节

示例:

```
1048576  → 1 MB 下行
```

来源: PGW-U 的 PFCP 流量测量（通过 URR 使用报告）

注意:

- 对于 `update` 事件是累计的
- 对于 `end` 事件是最终总数
- 对于 `start` 事件始终为 0
- 报告由 URR 时间阈值触发（通过 `usage_report_interval` 配置）

---

17. octets_out (上行流量)

类型: 无符号 64 位整数

描述: 在上行方向（UE → 网络）传输的字节数

单位: 字节

示例:

```
524288  → 512 KB 上行
```

来源: PGW-U 的 PFCP 流量测量（通过 URR 使用报告）

注意:

- 对于 `update` 事件是累计的
- 对于 `end` 事件是最终总数
- 对于 `start` 事件始终为 0
- 报告由 URR 时间阈值触发（通过 `usage_report_interval` 配置）

示例

示例 1：基本会话与单次更新

时间线：

1. 承载建立
2. 5 分钟后：使用更新（10 MB 下行，5 MB 上行）
3. 会话终止

CDR 输出：

```
# 数据 CDR 文件：
# 文件开始时间：10:00:00 (1726570800)
# 文件结束时间：11:00:00 (1726574400)
# 网关名称：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
```

示例 2：双栈会话与多次更新

时间线：

1. 建立双栈承载（IPv4 + IPv6）
2. 多次使用更新
3. 会话终止

CDR 输出：

```
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
```

示例 3：专用承载的会话

时间线：

1. 建立默认承载（QCI 9）
2. 为视频创建专用承载（QCI 6）
3. 两个承载的使用更新
4. 删除专用承载
5. 终止默认承载

CDR 输出：

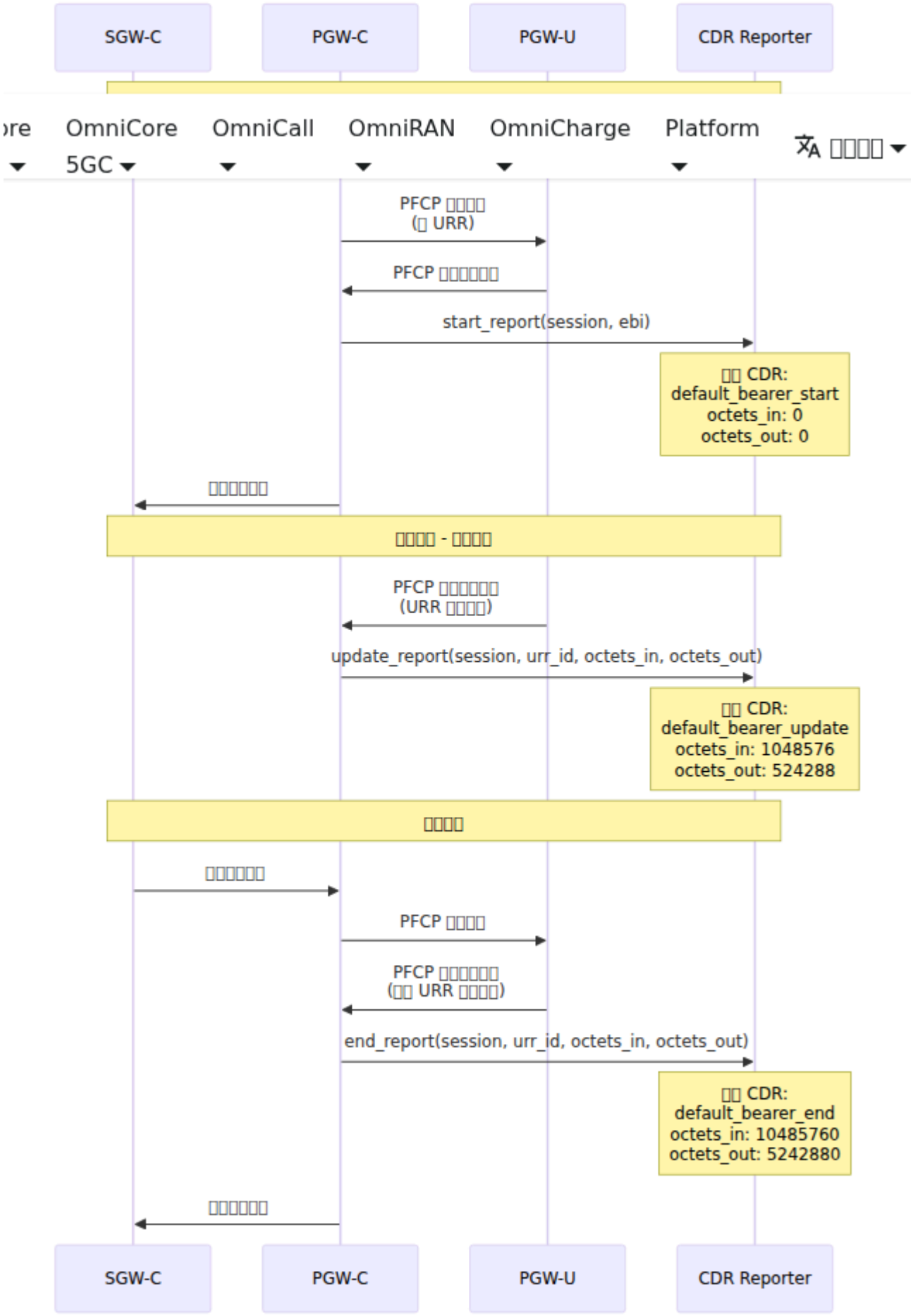
```
1726570800,310260333333333,default_bearer_start,10003,15553333333,333
1726571100,310260333333333,dedicated_bearer_start,10004,15553333333,3
1726571400,310260333333333,default_bearer_update,10003,15553333333,33
1726571400,310260333333333,dedicated_bearer_update,10004,15553333333,
1726571700,310260333333333,dedicated_bearer_end,10004,15553333333,333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

分析：

- 默认承载（10003）承载背景流量（10 MB 下行，4 MB 上行）
- 专用承载（10004）承载视频流量（200 MB 下行，2 MB 上行）
- 不同的 QCI 值（9 与 6）反映不同的 QoS 处理

□□

CDR □□□□


SGW-C
PGW-C
PGW-U
CDR Reporter
ore
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
PFCP □□□□
(□ URR)
PFCP □□□□□□
start_report(session, ebi)
□□ CDR:
default_bearer_start
octets_in: 0
octets_out: 0
□□□□□□
□□□□ - □□□□
PFCP □□□□□□
(URR □□□□)
update_report(session, urr_id, octets_in, octets_out)
□□ CDR:
default_bearer_update
octets_in: 1048576
octets_out: 524288
□□□□
□□□□□□
PFCP □□□□
PFCP □□□□□□
(□□ URR □□□□)
end_report(session, urr_id, octets_in, octets_out)
□□ CDR:
default_bearer_end
octets_in: 10485760
octets_out: 5242880
□□□□□□
SGW-C
PGW-C
PGW-U
CDR Reporter

CDR 收集方法

1. 基于文件的收集**:**

```
# 监视 CDR 目录 (PGW-C)
inotifywait -m /var/log/pgw_c/cdrs/ -e close_write | while read
path action file; do
    # 文件轮换完成，处理 CDR
    process_cdr "$path$file"
done
```

2. 实时流处理**:**

```
# 尾随并流式传输到处理管道
tail -F /var/log/pgw_c/cdrs/* | process_cdr_stream
```

相关文档

- 会话管理 - 会话生命周期和 CDR 触发
- PFCP 接口 - 通过 URR 从 PGW-U 进行使用报告
- 监控指南 - CDR 生成指标和警报
- 配置指南 - CDR 和 URR 配置参数
- Diameter Gx 接口 - CDR 中 QCI 值的策略控制
- Diameter Gy 接口 - 在线计费集成

3GPP 参考

- TS 32.251 - 包交换（PS）域计费
- TS 29.274 - 3GPP 演进分组系统（EPS）；GTP-C 协议
- TS 29.244 - CP 和 UP 节点之间的接口（PFCP） - **URR** 支持
- TS 32.298 - CDR 编码

CDR 格式 - *PGW-C □□□□□□□*

□ Omnitouch □□□□□□

文档版本**:** 1.0 最后更新**:** 2025-12-28

Diameter Gx 接口文档

策略和计费规则功能 **(PCRF)** 接口

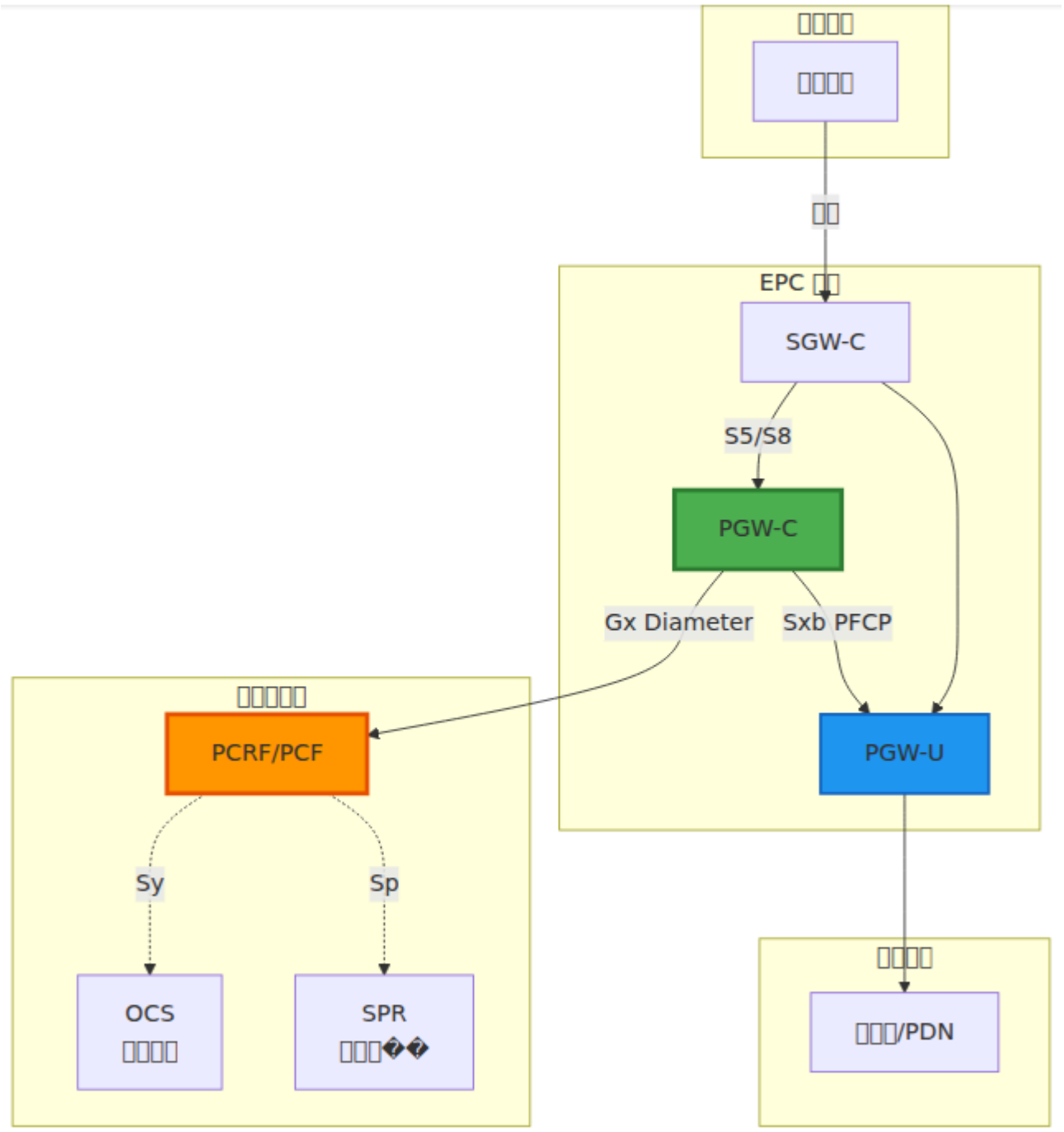
目录

1. □述
2. Gx 接口基础
3. Diameter 协议
4. 计费控制消息
5. 策略和计费规则
6. 位置变更报告
7. 配置
8. 消息流
9. 错误处理
10. 故障排除

概述

Gx 接口 将 PGW-C 连接到 **PCRF**（策略和计费规则功能）或 **PCF**（策略控制功能）在 5G 网络中。此接口使得：

- 动态策略控制 - 实时 QoS 和策略执行
- 计费控制 - 信用授权和使用跟踪
- 服务感知 - 应用级流量区分
- 用户资料管理 - 按用户应用策略



网络架构中的 **Gx**

□□□□
□□□□
□□
EPC □□
SGW-C
S5/S8
PGW-C
Gx Diameter
Sxb PFCP
PGW-U
□□□□□
PCRF/PCF
Sy
Sp
OCS
□□□□
SPR
□□□□
□□□□
□□□/PDN

关键功能

功能	描述
策略配置	PCRF 提供 PCC 规则，定义如何处理流量
QoS 控制	动态调整比特率和 QoS 参数
计费控制	预付费/后付费场景的信用授权
流量控制	根据策略启用/禁用流量
使用监控	跟踪每项服务的数据消耗

Gx 接口基础

3GPP 参考

- 规范: 3GPP TS 29.212
- **Diameter** 应用 **ID**: 16777238 (Gx)
- 协议: Diameter 基础协议 (RFC 6733)

会话概念

每个 UE PDN 连接都有一个对应的 **Gx** 会话，由 **Session-ID** 标识。此会话:

- 在 UE 附加时创建 (CCR-Initial)
- 在连接生命周期内更新 (CCR-Update) - 可选，例如策略变更或 位置变更报告
- 在 UE 脱离时终止 (CCR-Termination)

会话 ID 格式

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

组件:

- **Origin-Host**: PGW-C 的 Diameter 身份
- **high32**: 唯一标识符的高 32 位
- **low32**: 唯一标识符的低 32 位

Diameter 协议

消息结构

Diameter 消息是二进制编码的，具有以下结构:

```
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 (属性-值对)
├── AVP Header
│   ├── AVP Code
│   ├── Flags (V, M, P)
│   ├── AVP Length
│   └── Vendor ID (optional)
└── AVP Data
```

关键 **Diameter** 概念

AVP（属性-值对）：

- Diameter 中的基本数据单元
- 包含代码、标志和值
- 可以嵌套（分组 AVP）

命令：

- 请求/应答对
- CCR（计费控制请求）/ CCA（计费控制应答）

结果代码：

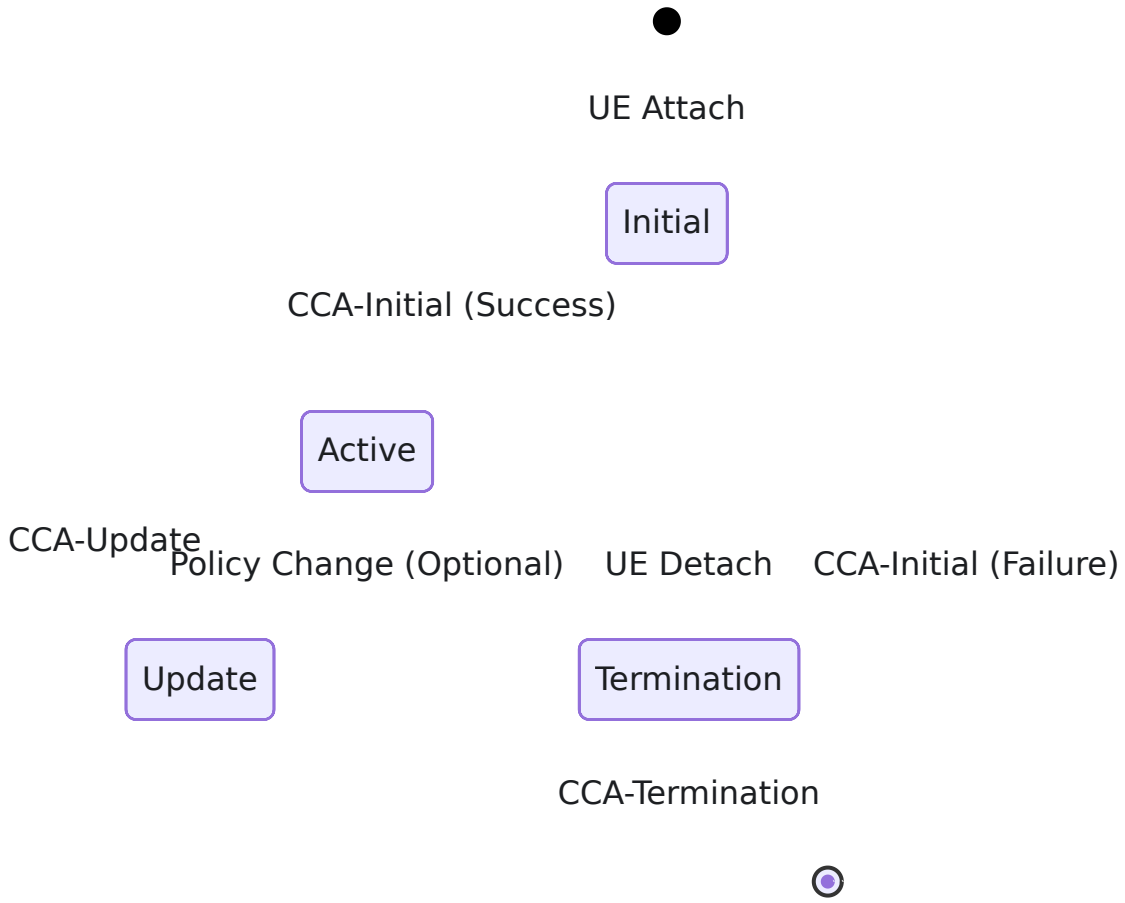
- `2001` - DIAMETER_SUCCESS
- `3xxx` - 协议错误
- `4xxx` - 瞬态故障

5xxx - 永久故障

计费控制消息

PGW-C 使用 Diameter 计费控制应用 (RFC 4006) 进行 Gx。

消息类型



CCR-Initial（计费控制请求 - 初始）

何时： UE 创建新的 PDN 连接
目的：
请求初始策略和计费规则
向 PCRF 提供 UE 和网络上下文

- 获取 QoS 参数和计费授权

PGW-C 发送的关键 **AVP**:

AVP 名称	AVP 代码	类型	描述
Session-Id	263	UTF8String	唯一 Gx 会话标识符
Auth-Application-Id	258	Unsigned32	16777238 (Gx)
Origin-Host	264	DiamIdent	PGW-C 的 Diameter 身份
Origin-Realm	296	DiamIdent	PGW-C 的 Diameter 领域
Destination-Realm	283	DiamIdent	PCRF 的领域
CC-Request-Type	416	Enumerated	1 = INITIAL_REQUEST
CC-Request-Number	415	Unsigned32	序列号（从 0 开始）
Subscription-Id	443	Grouped	UE 标识符 (IMSI/MSISDN)
Called-Station-Id	30	UTF8String	APN 名称
Framed-IP-Address	8	OctetString	分配的 UE IPv4 地址 (IPv4 和 IPv4v6 PDNs)
Framed-IPv6-Prefix	97	OctetString	分配的 UE IPv6 前缀 (仅 IPv6 PDNs)
3GPP-User-Location-Info	22	OctetString	初始用户 ULI (TS 29.061)
User-Equipment-Info	458	Grouped	UE 身份 (例如 IMEISV)，如果可用
IP-CAN-Type	1027	Enumerated	5 = 3GPP-EPS
RAT-Type	1032	Enumerated	从 GTP RAT 类型映射而来 (例如 1004 = EUTRAN)

AVP 名称	AVP 代码	类型	描述
QoS-Information	1016	Grouped	当前 QoS (APN-AMBR UL/DL)
Default-EPS-Bearer-QoS	1049	Grouped	默认承载 QCI/ARP (QCI 5)
Network-Request-Support	1024	Enumerated	网络发起的程序
Supported-Features	628	Grouped	Gx 特性列表 (Rel-8/9/10 Gx)

> 帧地址 AVP 由 UE 地址族选择：`Framed-IP-Address` 用于 IPv4（和 IPv4v6 PDN 的 IPv4），`Framed-IPv6-Prefix` 用于仅 IPv6 PDN。对于 IPv4v6，IPv6 前缀不会在 Gx 上传输。请参见 IPv6 / 双栈。

示例 **CCR-I** 结构：

```
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（计费控制应答 - 初始）

发送方：PCRF 响应 CCR-I

目的：

- 授权或拒绝会话
- 提供流量处理的 PCC 规则
- 指定 QoS 参数

PGW-C 接收的关键 **AVP**：

AVP 名称	**AVP** 代码	描述
Result-Code	268	成功 (2001) 或错误代码
Experimental-Result	297	厂商特定的结果代码
QoS-Information	1016	授权的 QoS (可能与请求不同)
Charging-Rule-Install	1001	要激活的 PCC 规则
Charging-Rule-Definition	1003	内联规则定义
Default-EPS-Bearer-QoS	1049	默认承载的 QoS

成功响应示例：

```
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（计费控制请求 **-** 终止）

何时： UE 脱离或 PDN 连接被删除

目的:

- 通知 PCRF 会话终止
- 最终计费/计费记录

与 **CCR-I** 的关键区别:

- `CC-Request-Type: TERMINATION_REQUEST (3)`
- 可能包括使用统计信息
- 简化的 AVP 集合

示例 **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

发送方: PCRF 响应 CCR-T

目的:

- 确认会话终止
- 不返回策略规则

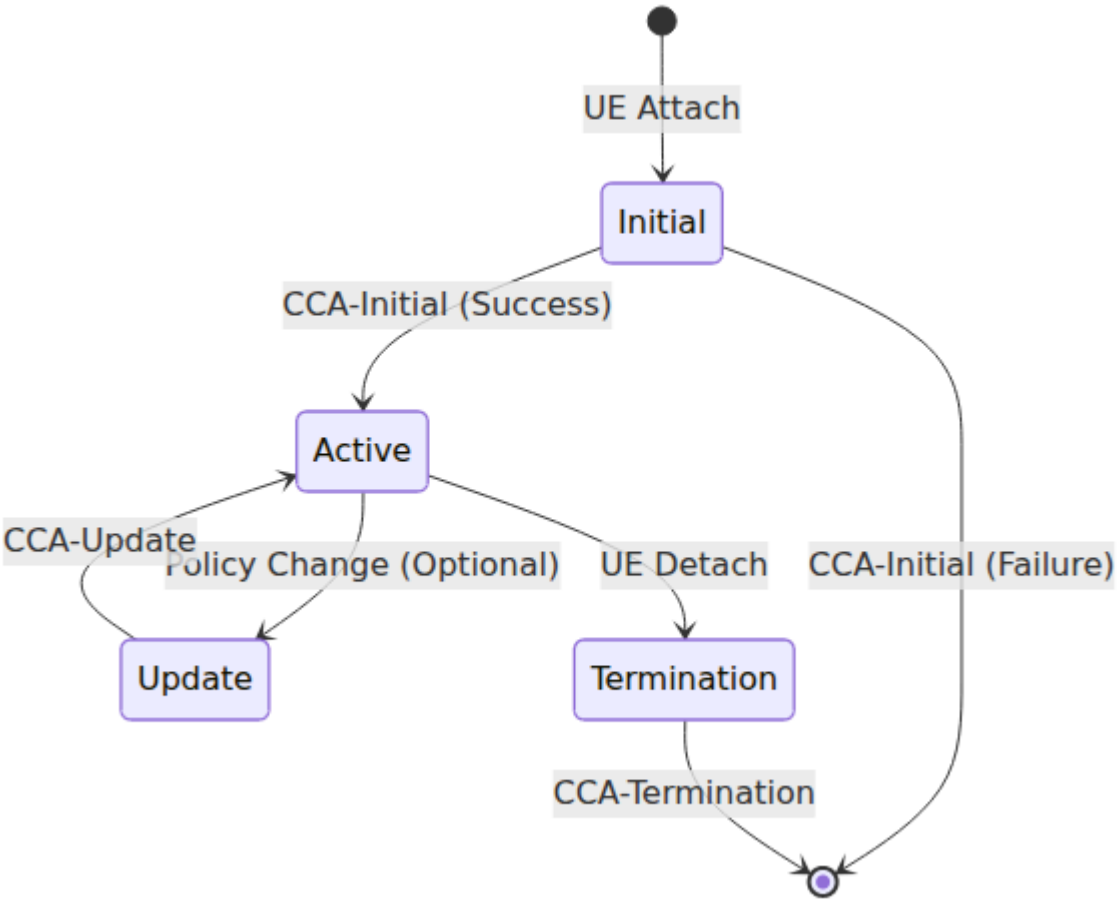
示例 **CCA-T**:

```
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
```

策略和计费规则

PCC 规则结构

PCC（策略和计费控制）规则 定义如何处理特定流量流：



规则组件

1. 规则名称：

规则的唯一标识符
示例："video_streaming_rule"

2. 优先级：

数字越小 = 优先级越高
范围：0-65535
当多个规则匹配时使用

3. 流过滤器（TFT - 流量流模板）：

定义哪些数据包匹配此规则
示例：
IP 5 元组：协议、源/目标 IP、源/目标端口

- `"permit out ip from any to 8.8.8.8 80"`

4. QoS 信息:

- **QCI**（**QoS** 类标识符）：1-9（标准化），128-254（运营商特定）
  - QCI 1: 对话语音
  - QCI 5: IMS 信令
  - QCI 9: 默认互联网
- **ARP**（分配和保留优先级）：抢占能力
- **MBR/GBR**：最大/保证比特率

5. 计费信息:

- 计费组：识别计费类别（由 OCS 使用 - 参见 Diameter Gy 接口）
- 计量方法：基于流���、时间或事��
- 在线/离线计费：OCS（通过 Diameter Gy 的预付费）与离线 CDR（后付费 - 参见 数据 CDR 格式）

6. 流量状态:

- 开放：允许流量
- 关闭：阻止流量

动态规则配置

PCRF 可以通过两种方式提供规则:

1. 预定义规则（按名称）：

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

2. 动态规则（内联定义）：

```
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 信息 AVP

APN-AMBR（聚合最大比特率）：

适用于此 APN 的所有非 GBR 承载:

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

PGW-C 响应:

- 更新内部 AMBR 状态
- 向 PGW-U 发送带有更新 QER 的会话修改请求

位置变更报告

PGW-C 可以在会话生命周期内通知 PCRF 当用户的 位置、无线接入技术 **(RAT)**、服务 **PLMN** 或 **UE** 时区 发生变化时。该变化在 **CCR-Update** 中作为 **3GPP-User-Location-Info** 和相关 AVP 传递。

报告是选择性的（**PCRF** 控制）

此报告是 基于 **PCRF** 的请求 — 正如 3GPP TS 29.212 §4.5.7 / §5.3.7 所要求的。PCRF □□ 它希望听到的事件，通过在 **CCA**（或 **RAR**）中包含一个或多个 **Event-Trigger** AVP。PGW-C 为每个 Gx 会话存储武装集，仅在以下

情况下发送 CCR-Update:

1. 武装的 Event-Trigger 的条件实际发生，并且
2. 存在该用户的 Gx 会话。

如果 PCRF 武装 没有 与位置相关的 Event-Triggers，PGW-C 将对位置变化保持沉默 — 它只是更新其内部状态并回答 GTP-C 修改承载请求。这使得对于不关心移动性的 PCRF 保持接口安静，对于关心的 PCRF 则符合要求。

武装集可以在会话期间的任何时间更改。 PGW-C 从 PCRF 发送的每个 Event-Trigger AVP 集更新它 — 在 CCA-Initial、CCA-Update 或 PCRF 发起的 **RAR** 中。当前的 Event-Trigger 集替换当前的武装集；不携带 Event-Trigger AVP 的消息将保持武装集不变。因此，PCRF 可以在没有位置报告的情况下开始会话，随后通过 RAR 打开（或关闭）它，反之亦然。

支持的事件触发器

Event-Trigger	值	触发时...	报告的 **AVP**
`RAT_CHANGE`	2	RAT-Type 变化（例如 EUTRAN → UTRAN）	`RAT-Type`
`PLMN_CHANGE`	4	服务网络（MCC/MNC）变化	`3GPP-SGSN-MCC-MNC`
`USER_LOCATION_CHANGE`	13	用户位置信息的任何变化	`3GPP-User-Location-Info`
`UE_TIME_ZONE_CHANGE`	25	UE 时区变化	`3GPP-MS-TimeZone`
`TAI_CHANGE`	26	跟踪区域标识变化	`3GPP-User-Location-Info`
`ECGI_CHANGE`	27	E-UTRAN 小区全局标识变化	`3GPP-User-Location-Info`

多个触发器可以同时触发（例如，移动到新小区的新跟踪区域触发 `USER_LOCATION_CHANGE`、`TAI_CHANGE` 和 `ECGI_CHANGE`）；所有触发并武装的触发器都列在 CCR-Update 的 `Event-Trigger` AVP 中。

位置来源

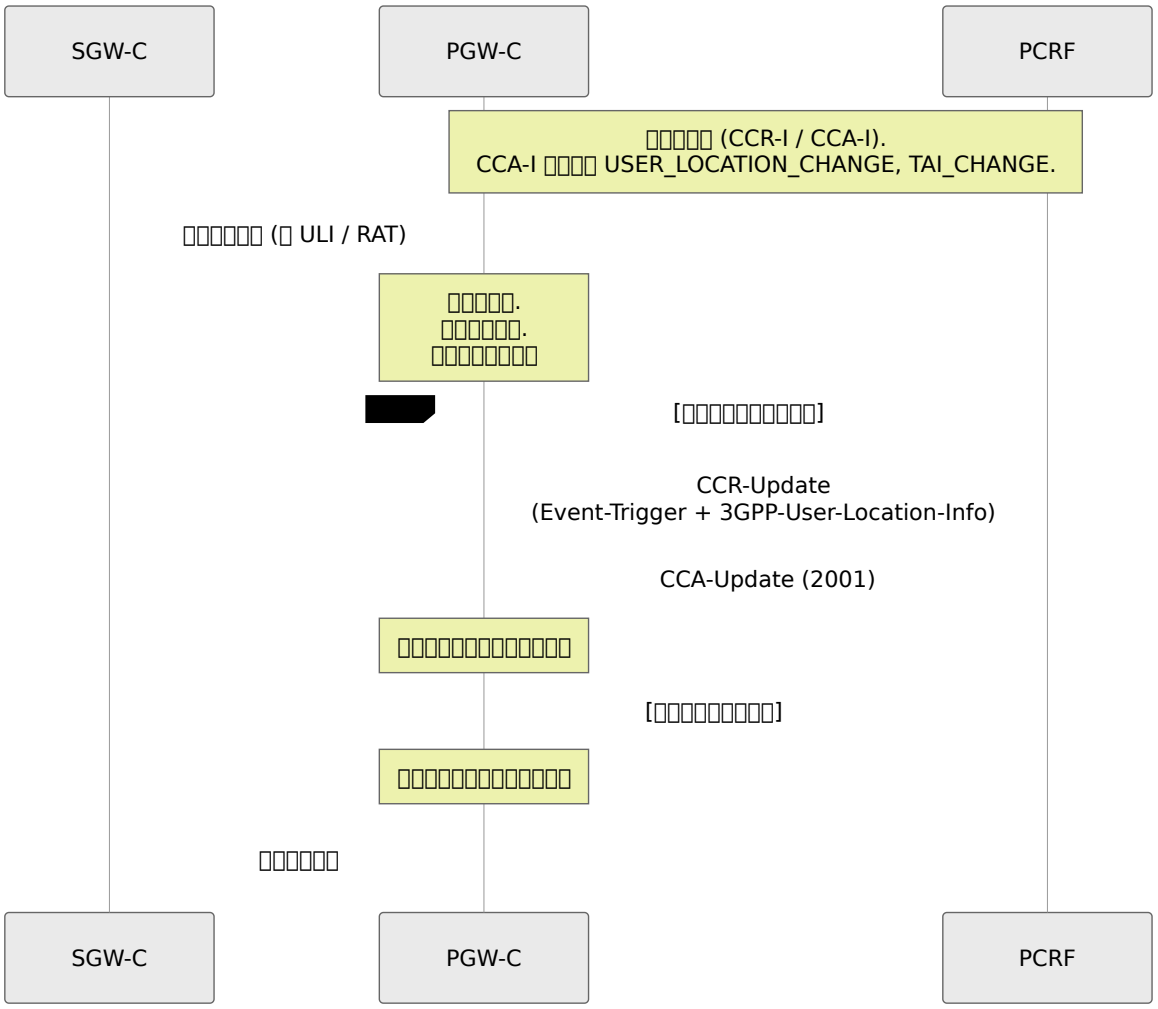
- **CCR-Initial** 现在包括用户的初始 `3GPP-User-Location-Info`，为 PCRF 提供基线。
- 在会话期间，**GTP-C** 修改承载请求（S5/S8）携带更新的用户位置信息、RAT-Type、服务网络和 UE 时区。PGW-C 解析这些信息，将其与存储的值进行比较，并报告任何武装的变化。
- 在修改承载请求中缺失的 IE 保持其先前值，因此部分修改承载请求永远不会看起来像是虚假的位置变化。

3GPP-User-Location-Info 编码

AVP 根据 **3GPP TS 29.061 §16.4.7.2** 编码：一个字节的 □□□□□□ 后跟位置字段，PLMN 身份根据 3GPP TS 24.008 §10.5.1.3 编码。支持的类型：

类型	值	内容
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 和 ECGI	130	PLMN + TAC + PLMN + ECI (LTE，优选)

□□□



> □□□□ □ PCRF □□□□□□□□□ GTP-C □□□□□□□□□□ CCR-Update □□□ PCRF □□□□□□PGW-C □□□□□□□□□ SGW-C — □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ API □□□□□□□□

□□ API (`GET /api/sessions` □ `GET /api/sessions/:imsi`) □□□□□□□□□JSON □□□□□□ □ `state` □□□□□□□□□□□□□□□□□□PCRF □□□□□□□□□□ `state.gx.requested_event_triggers` □□□□□□□□□□ Gx □□□□□□`session_id`□ `cc_request_number`□□□□□□/□□□□□□□□ `state.uli`□□□□□□□□□□□□□□□□□□□

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

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

- ﷺ是 Gx 事件触发器代码的列表；它替换武装集（其他 `gx` 字段在合并时保持不变）。
- 空列表（`[]`）会解除会话的所有位置报告。
- 响应回显更新后的、经过清理的会话 `state`。

此端点没有 保护措施：请求体是深度合并到实时会话中，因此 □□ 字段都可以更改 — 包括更改为破坏会话的值。具有下游影响的更改会传播到 UPF。更新 APN-AMBR 或默认承载 QoS 会重新计算默认承载 QER（QCI / MBR / GBR）并将其作为更新 QER 推送到 PFCP 会话修改中：

```
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 } }
```

当 `default_bearer_qos` 提供显式 MBR/GBR（GBR 承载）时，这些驱动 QER 速率限制；否则 APN-AMBR 被用作默认承载 MBR。

> 注意：覆盖武装触发器会更改 PGW-C 对 PCRF 武装的本地视图。PCRF 仍然可以通过 CCA 或 RAR 重新武装该集合，这将覆盖手动覆盖。

配置

基本 Gx 配置

编辑 `config/runtime.exs`：

```
config :pgw_c,
  diameter: %{
    # 监听 Diameter 连接的 IP 地址
    listen_ip: "0.0.0.0",

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

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

    # PCRF 对等体列表
    peer_list: [
      %{
        # PCRF Diameter 身份
        host: "pcrf.epc.mnc999.mcc999.3gppnetwork.org",

        # PCRF 领域（通常与 PGW-C 领域相同）
        realm: "epc.mnc999.mcc999.3gppnetwork.org",

        # PCRF IP 地址
        ip: "10.0.0.30",

        # PGW-C 是否发起与 PCRF 的连接
        # true = PGW-C 连接到 PCRF
        # false = 等待 PCRF 连接
        initiate_connection: true
      }
    ]
  }
```

多个 PCRF 对等体

用于冗余或地理分布：

```
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
      }
    ]
  }
```

负载均衡:

- Diameter 协议处理对等体选择
- 请求根据可用性分配
- 在对等体故障时自动故障转移

主机名解析

Diameter 身份必须是 **FQDN**（完全限定域名）：

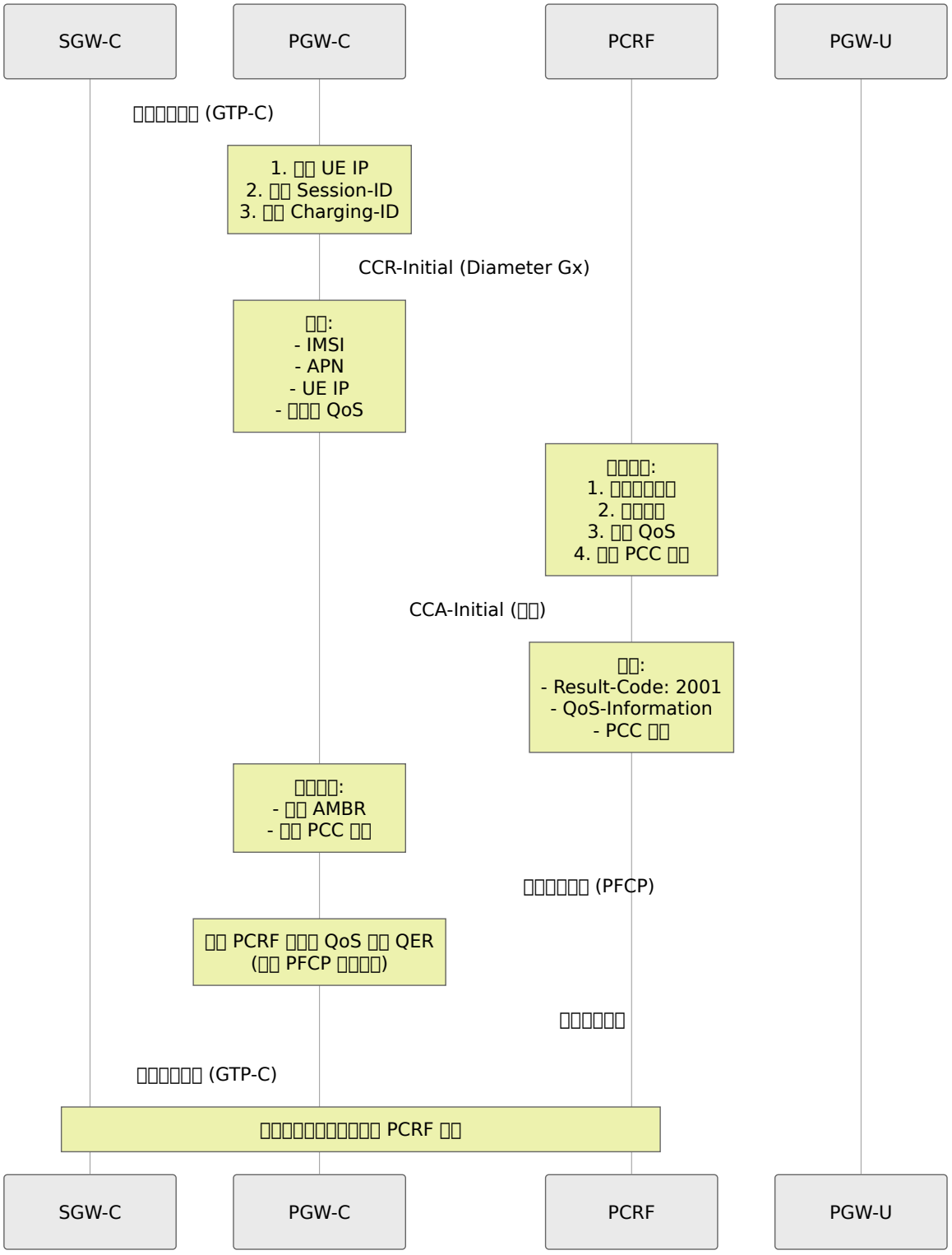
```
# 正确 - FQDN 格式
host: "pgw_c.epc.mnc999.mcc999.3gppnetwork.org"

# 不正确 - 不是有效的 Diameter 身份
host: "pgw_c"
host: "10.0.0.20"  # 不允许 IP 地址
```

领域格式:

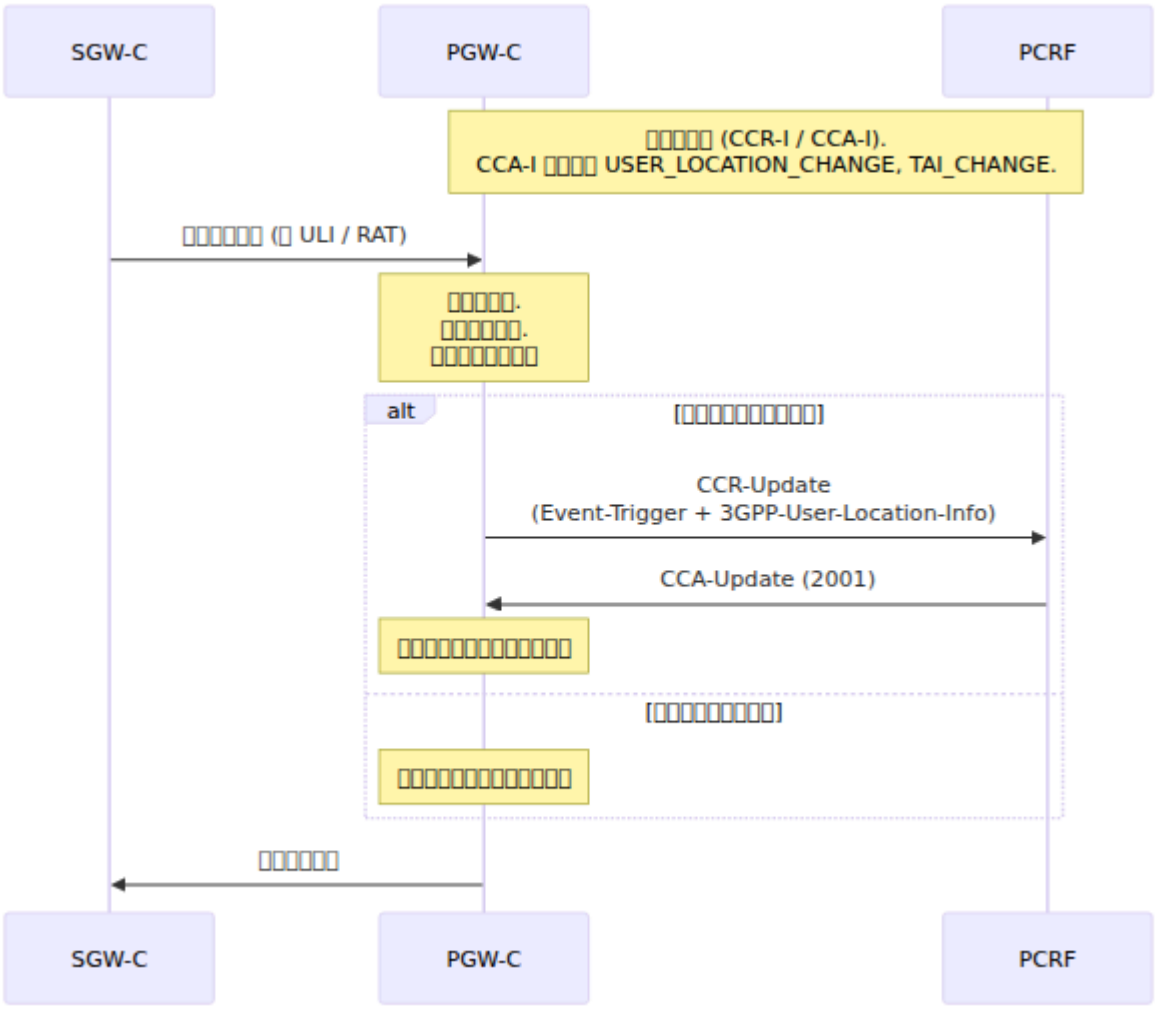
- 必须是有效的域名
- 通常与 3GPP PLMN 格式匹配：`epc.mncXXX.mccYYY.3gppnetwork.org`

□□□

□□□□□□□□



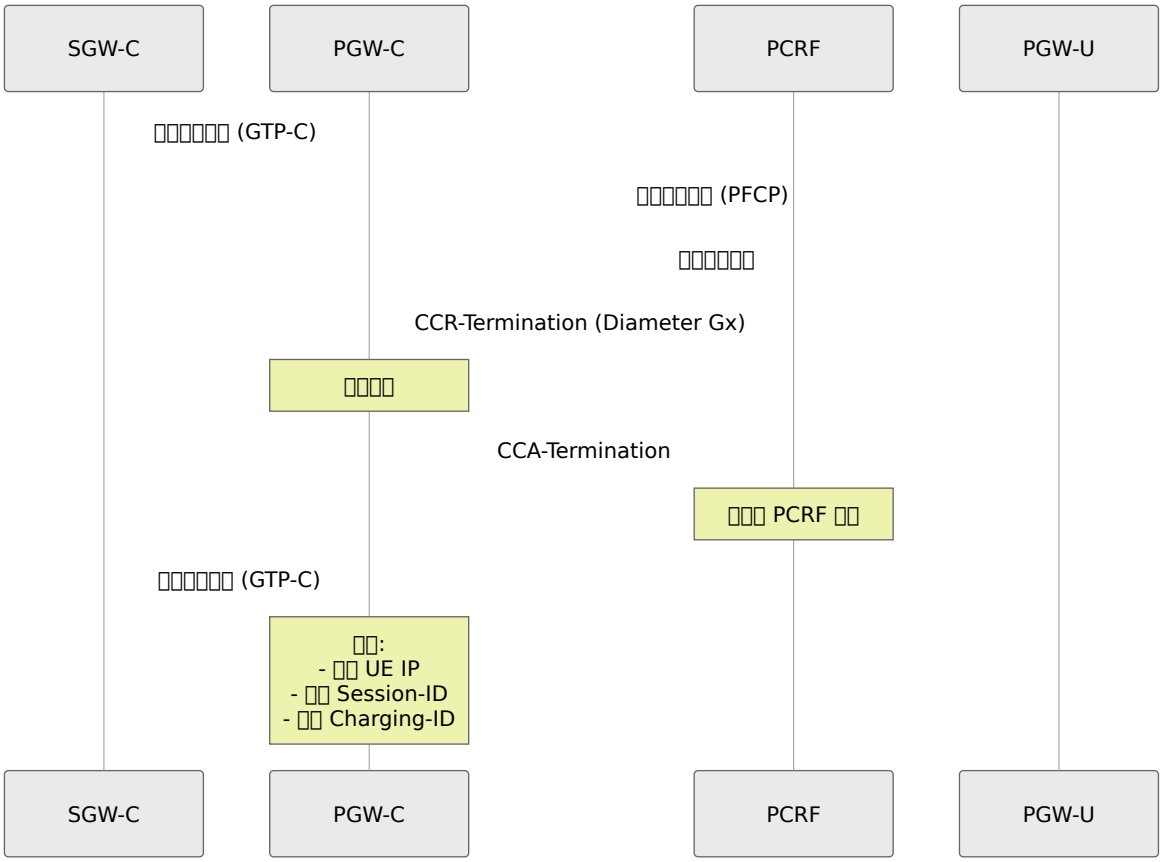
□□□ PCRF □ QoS □□□□□□□ QER□QoS □□□□□□□□□ PFCP □□□ PGW-U□□□□ PFCP □□ □□□ QER □□□□□



策略更新（网络发起）

SGW-C
PGW-C
PCRF
□□□□□□ (CCR-I / CCA-I).
CCA-I □□□□ USER_LOCATION_CHANGE, TAI_CHANGE.
□□□□□□□ (□ ULI / RAT)
□□□□□□.
□□□□□□□.
□□□□□□□□□
alt
[□□□□□□□□□□□□]
CCR-Update
(Event-Trigger + 3GPP-User-Location-Info)
CCA-Update (2001)
□□□□□□□□□□□□□□□
[□□□□□□□□□□]
□□□□□□□□□□□□□□□
□□□□□□□
SGW-C
PGW-C
PCRF

???□□□



□□□□

□□□□

PGW-C □□ CCA □□□□□□□ Diameter □□□□□

□□□□□□

□□	□□	□□
2001	DIAMETER_SUCCESS	□□□□□□□

□□□□ (5xxx)□

代码	名称	**PGW-C** 动作
5002	DIAMETER_UNKNOWN_SESSION_ID	记录错误，失败会话
5030	DIAMETER_USER_UNKNOWN	拒绝会话（用户未知）
5140	DIAMETER_ERROR_INITIAL_PARAMETERS	记录错误，重试或失败
5003	DIAMETER_AUTHORIZATION_REJECTED	拒绝会话（未授权）

瞬态故障 **(4xxx)**:

代码	名称	**PGW-C** 动作
4001	DIAMETER_AUTHENTICATION_REJECTED	重试或失败会话
4010	DIAMETER_TOO_BUSY	重试并退避
4012	DIAMETER_UNABLE_TO_COMPLY	记录错误，可能重试

实验性结果代码

厂商特定的错误代码:

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

常见的 **3GPP** 实验代码:

代码	名称	意义
5065	IP_CAN_SESSION_NOT_AVAILABLE	PCRF 无法建立会话
5143	INVALID_SERVICE_INFORMATION	服务数据无效

超时处理

CCR-I 超时:

如果 PCRF 在超时内未响应 CCR-Initial:

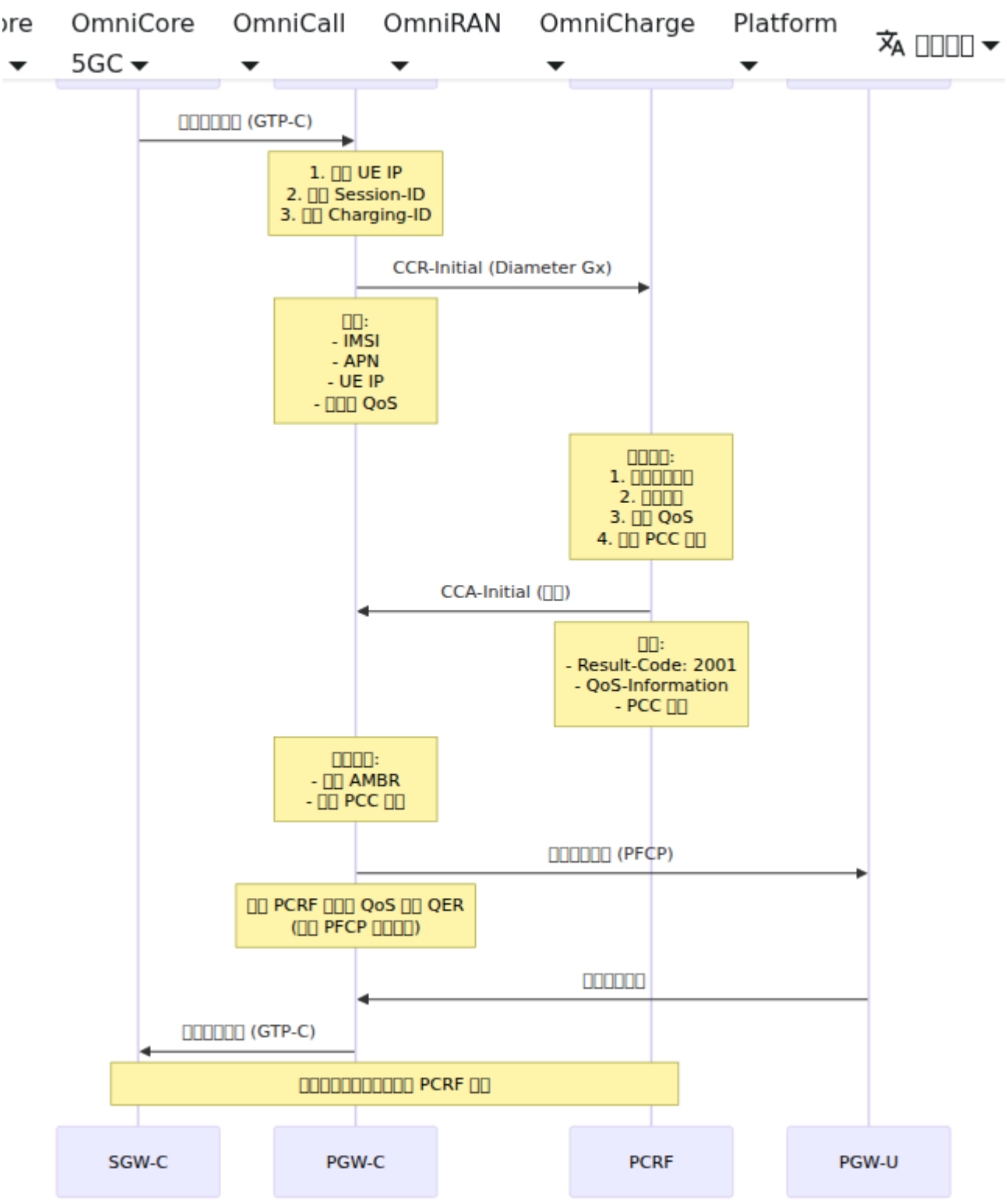
1. PGW-C 等待配置的超时（例如，5 秒）
2. 如果未收到 CCA:
   - 记录:“CCR-Initial 超时，Session-ID: ...”
   - 向 SGW-C 响应错误原因
   - 清理分配的资源
3. SGW-C 收到：创建会话响应（原因：远程对等体未响应）

对 **SGW-C** 的错误响应:

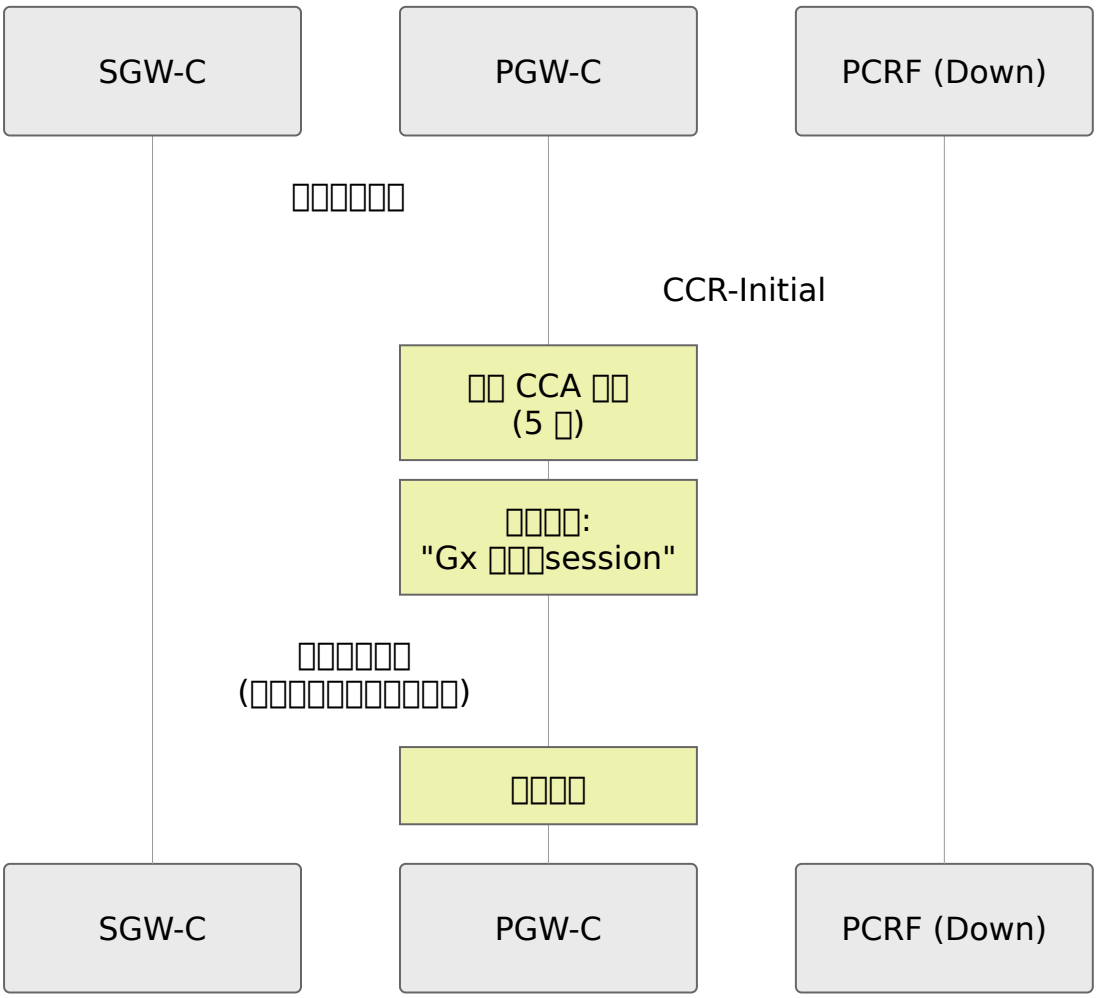
当 CCR-Initial 超时，PGW-C 向 SGW-C 发送创建会话响应，原因代码为 `:remote_peer_not_responding`。

故障场景

场景 **1**: **PCRF** 拒绝会话（用户未知）


re
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□□□□
□□□□□□□ (GTP-C)
1. □□ UE IP
2. □□ Session-ID
3. □□ Charging-ID
CCR-Initial (Diameter Gx)
□□:
- IMSI
- APN
- UE IP
- □□□ QoS
□□□□:
1. □□□□□□□
2. □□□□
3. □□ QoS
4. □□ PCC □□
CCA-Initial (□□)
□□:
- Result-Code: 2001
- QoS-Information
- PCC □□
□□□□:
- □□ AMBR
- □□ PCC □□
□□□□□□□ (PFCP)
□□ PCRF □□□ QoS □□ QER
(□□ PFCP □□□□)
□□□□□□□
□□□□□□□ (GTP-C)
□□□□□□□□□□□□□ PCRF □□
SGW-C
PGW-C
PCRF
PGW-U

□□ 2□PCRF □□□□□□



□□□□

□□□□

1. Diameter □□□□□□□□

□□□

- □□□“Diameter □□□□□□□”
- □□□ CCR-Initial

□□□□□□

- PCRF □□□
- □□□ PCRF IP □□□

- 防火墙阻止 Diameter 端口（3868）
- Diameter 身份（主机/领域）不正确

解决方案：

```
# 测试网络连接
ping <pcrf_ip>

# 测试 Diameter 端口 (TCP 3868)
telnet <pcrf_ip> 3868

# 检查 Diameter 身份配置
# 确保主机和领域是 FQDN，而不是 IP
```

验证配置：

```
config :pgw_c,
  diameter: %{
    # 必须是 FQDN，而不是 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 超时

症状：

- 创建会话请求失败
- 日志：“CCR-Initial 超时”

可能原因：

- PCRF 超载

- 网络延迟
- PCRF 未响应此 Session-ID

解决方案:

1. 检查 PCRF 日志中的错误
2. 验证 PCRF 是否正在处理请求
3. 检查网络延迟: `ping <pcrf_ip>`
4. 如果网络延迟较高，则增加超时

3. 会话被 PCRF 拒绝

症状:

- CCA-Initial 的 Result-Code != 2001
- 创建会话响应失败

常见结果代码:

结果代码	可能原因	解决方案
5030	IMSI 不在用户数据库中	在 HSS/SPR 中配置用户
5003	授权被拒绝	检查用户权限
4010	PCRF 太忙	重试或增加 PCRF 容量

检查日志:

```
# PGW-C 日志显示:
[error] Diameter Gx 错误: Result-Code 5030 (DIAMETER_USER_UNKNOWN)
[error] IMSI 310260999999999 被 PCRF 拒绝
```

4. QoS 未应用

症状:

- 会话已建立但 QoS 错误

- 比特率与预期值不匹配

调试步骤：

1. 检查 **CCA-Initial**：
    - 验证 `QoS-Information` AVP 是否存在
    - 检查 `APN-Aggregate-Max-Bitrate-UL/DL` 值
2. 检查 **PFCP** 会话建立：
    - 验证 QER 是否以正确的 MBR 值创建
    - 检查 PGW-U 日志以获取 QER 安装信息
3. 验证 **PCRF** 策略：
    - 检查 PCRF 配置
    - 验证用户资料是否包括正确的 QoS

5. Diameter 路由问题

症状：

- Diameter 消息未到达 PCRF
- 日志：“没有到目的领域的路由”

原因：

- 配置和消息之间的领域不匹配

解决方案：

确保一致性：

```
# 所有必须匹配
config :pgw_c,
  diameter: %{
    realm: "epc.mnc999.mcc999.3gppnetwork.org",  # PGW-C 的领域
    peer_list: [
      %{
        realm: "epc.mnc999.mcc999.3gppnetwork.org"  # PCRF 的领域（通常相同）
      }
    ]
  }
```

在 **CCR-Initial** 中：

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

监控 **Gx** 健康

关键指标：

```
# Gx 消息速率
rate(gx_inbound_messages_total{message_type="gx_CCA"}[5m])
rate(gx_outbound_messages_total{message_type="gx_CCR"}[5m])

# Gx 错误速率
rate(gx_inbound_errors_total[5m])

# Gx 响应成功率（新指标）
sum(rate(gx_outbound_responses_total{result_code_class="2xxx"}
[5m])) /
sum(rate(gx_outbound_responses_total[5m])) * 100

# 按 PCRF 主机的 Gx 响应失败
rate(gx_outbound_responses_total{result_code_class!="2xxx"}[5m])
by (diameter_host)

# Gx 会话计数
session_id_registry_count

# Gx 消息处理持续时间
histogram_quantile(0.95,
rate(gx_inbound_handling_duration_bucket[5m]))
```

按结果代码类别的响应指标：

`gx_outbound_responses_total` 指标提供了对发送到 PCRF 对等体的 Diameter 响应的详细可见性，按以下类别分类：

- **`message_type`**：响应消息类型（`gx_RAA`、`gx_CCA`）
- **`result_code_class`**：结果代码类别（`2xxx`、`3xxx`、`4xxx`、`5xxx`）
- **`diameter_host`**：接收响应的 PCRF 对等体

常见结果代码：

- **2001**（DIAMETER_SUCCESS） - 成功响应
- **3001**（DIAMETER_COMMAND_UNSUPPORTED） - 协议错误
- **5012**（DIAMETER_UNABLE_TO_COMPLY） - 无法执行请求
- **5030**（DIAMETER_USER_UNKNOWN） - 订阅者未找到

警报示例：

```
# □ Gx 错误率警报
- alert: GxErrorRateHigh
  expr: rate(gx_inbound_errors_total[5m]) > 0.1
  for: 5m
  annotations:
    summary: "检测到高 Gx 错误率"

# □ Gx 响应失败率警报
- 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: "高 Gx 响应失败率"
    description: "超过 10% 的 Gx 响应为失败"

# PCRF 特定失败的监控
- 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 }} 接收失败响应"
    description: "高失败率的 PCRF 主机"

# 会话拒绝警报
- alert: GxSessionRejection
  expr: rate(gx_inbound_errors_total{result_code="5030"}[5m]) >
0.01
  for: 5m
  annotations:
    summary: "PCRF 拒绝会话（USER_UNKNOWN）"
```

调试日志

启用详细的 Diameter 日志:

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

# 或在运行时
iex> Logger.configure(level: :debug)
```

查找：

- `[debug] 发送 CCR-Initial, Session-ID: ...`
- `[debug] 收到 CCA-Initial: Result-Code 2001`
- `[error] Diameter 错误: ...`

Diameter 对等体监控

Diameter Gx 对等���与 PCRF 的连接通过 **OAM REST API** 进行监控。API 通过 HTTPS/TLS 在 `https://<host>:8443` 提供，所有路由在 `/api` 前缀下。交互式 API 文档（Swagger UI）可在 `https://<host>:8443/api/docs` 获取。

Diameter 对等体端点

```
GET /api/diameter                          # 所有配置的对等体 + 实时状态
GET /api/diameter?status=disconnected      # 按连接状态过滤
GET /api/diameter/<origin-host>            # 通过 Origin-Host 获取一个对等体
```

示例：

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

`GET /api/diameter` 返回每个配置的 Diameter 对等体及其当前连接状态。添加 `?status=connected` 或 `?status=disconnected` 以按连接状态过滤，或附加对等体的 Origin-Host 到 `GET /api/diameter/<origin-host>` 以获取单个对等体。

对于每个对等体，返回以下字段：

字段	描述
`host`	对等体的 Diameter 身份 (Origin-Host)
`status`	连接状态（例如：已连接 / 未连接 / 正在连接）
`ip`	对等体 IP 地址
`port`	对等体端口
`transport`	TCP 或 SCTP
`connection_initiation`	OmniPGW 是否发起（出站）或接受（入站）连接
`realm`	对等体在 CEA 中报告的 Diameter 领域
`product_name`	对等体在 CEA 中报告的厂商产品字符串
`application_ids`	在 CEA 中协商的 Diameter 应用 ID（Gx = 16777238）

健康的部署显示所有配置的 PCRF 对等体为 已连接。所有 Gx 对等体的丧失会阻止策略安装并阻止新会话的创建。

作用例

监控 **PCRF** 连接性：

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

验证所有 PCRF 对等体显示为`已连接`，`connection_initiation` 与配置匹配，并且 `application_ids` 包含 Gx（16777238）。

故障排除会话创建失败（**Gx** 问题）：当用户会话因 "PCRF 超时" 错误而失败时，查询受影响的对等体：

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

- `status` 为`未连接` → 检查网络连接，验证 PCRF 是否正在运行，并确认防火墙规则允许 TCP 3868。

- `status` 为 已连接 但会话仍然失败 → 问题出在应用层（检查日志）；PCRF 可能拒绝订阅者。

验证 **Diameter** 配置：在配置新的 PCRF 对等体后，确认它出现并达到 已连接：

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

检查 `realm` 是否与配置匹配，`application_ids` 是否包含 Gx，并且 `product_name` 是否显示预期的 PCRF 标识符。

监控故障转移：当主 PCRF 故障时，`GET /api/diameter?status=disconnected` 将其列为 未连接，而备份 PCRF 保持 已连接。新会话会自动使用备份；当主服务器恢复时，其 `status` 返回为 已连接。

检测 **Diameter** 路由问题：

- 对等体显示为 已连接 但 `realm` 错误
- `application_ids` 不包括 Gx（16777238）
- `product_name` 与预期的 PCRF 不匹配

识别配置不匹配：如果 `connection_initiation` 报告对等体发起连接，而配置设置为 `initiate_connection: true`，则两端都在尝试发起，这可能导致连接竞争条件。

与指标集成

OAM API 提供实时对等体状态；将其与 Prometheus 结合使用：

- 历史 Gx 错误率
- CCR/CCA 消息计数
- 延迟趋势

`GET /api/diameter` 回答“现在是否正常工作？”；指标回答“它在一段时间内的工作情况如何？”

相关文档

配置和策略

- 配置指南 - Diameter 配置，PCRF 对等体设置
- **PFCP** 接口 - 通过 PCC 规则的 QER 进行 QoS 执行
- 会话管理 - 会话生命周期与策略集成
- **QoS** 和承载管理 - 详细的 QoS 配置和承载设置

计费集成

- **Diameter Gy** 接口 - 通过 PCC 规则触发的在线计费
- 数据 **CDR** 格式 - 带有策略信息的离线计费记录
- **PCO** 配置 - IMS 策略控制的 P-CSCF 交付

□作

- 监控指南 - Gx 指标，策略跟踪，PCRF 连接警报
- **S5/S8** 接口 - 与策略的承载管理集成

返回□作指南

直径在线计费（Gy/Ro 接口）

在线计费系统（OCS）接口

目录

1. □述
2. 3GPP 计费架构
3. Gy/Ro 接口基础
4. 信用控制消息
5. 在线计费流程
6. 承载计费控制
7. 多服务信用控制
8. 配置
9. 消息流程
10. 错误处理
11. 与 Gx 的集成
12. 故障排除

概述

Gy 接口（在 IMS 上下文中也称为 **Ro** 接口）将 PGW-C 连接到在线计费系统（OCS）以进行实时信用控制。这使得：

- 预付费计费 - 实时信用授权和扣除
- 实时信用控制 - 在服务交付之前授予配额
- 基于服务的计费 - 对语音、数据、短信等进行不同计费
- 即时账户更新 - 实时更新信用余额
- 服务拒绝 - 当信用耗尽时阻止服务

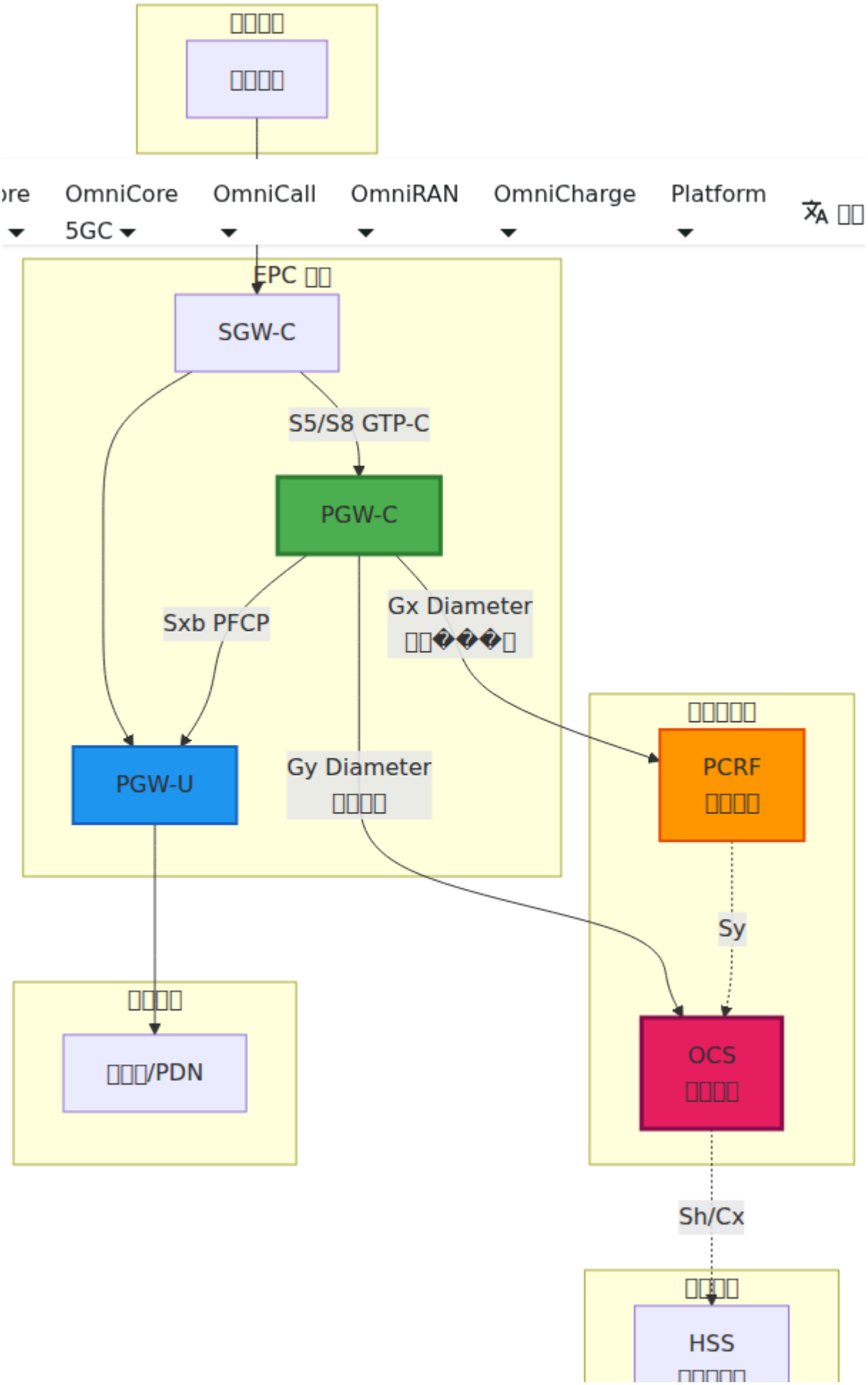
在线与离线计费

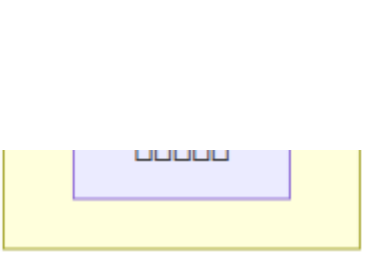
方面	在线计费 **(Gy/Ro)**	离线计费 **(Gz/Rf)**
时机	实时，在服务之前	服务交付后
用例	预付费用户	后付费用户
信用检查	是，在授予服务之前	否，账单稍后生成
系统	OCS（在线计费系统）	CGF/CDF（计费数据功能）
风险	无收入损失	有未付账单的风险
复杂性	高（实时要求）	较低（批处理）
用户影响	如果没有信用则拒绝服务	服务始终可用

另见：数据 CDR 格式 用于离线计费记录（后付费计费）

另见：会话管理 用于完整的 PDN 会话生命周期，包括计费集成

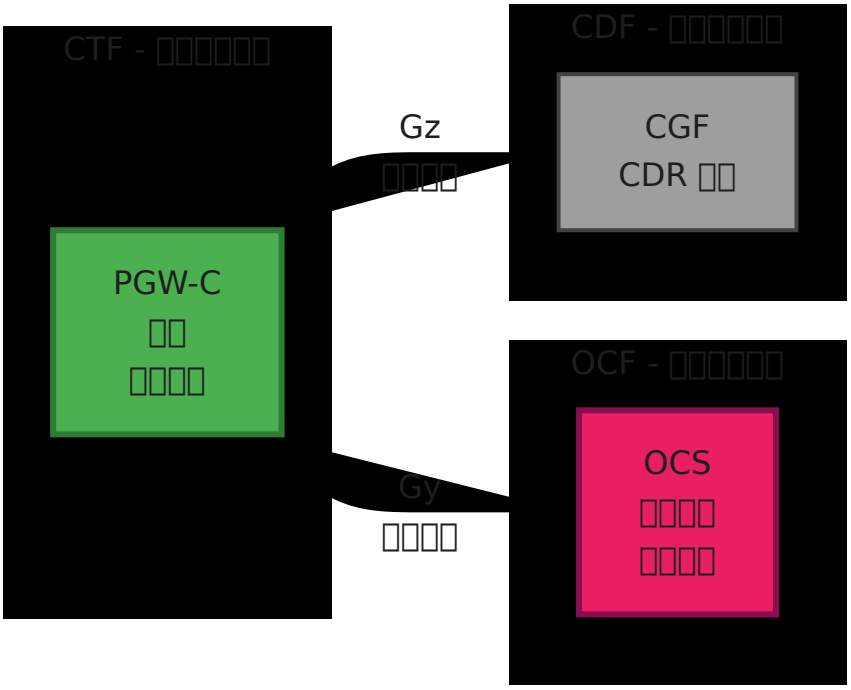
□□□□□□ Gy


□□□□
□□□□
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□
EPC □□
SGW-C
S5/S8 GTP-C
PGW-C
Sxb PFCP
Gx Diameter
□□□□□
PGW-U
Gy Diameter
□□□□
□□□□□
PCRF
□□□□
Sy
□□□□
□□□/PDN
OCS
□□□□
Sh/Cx
□□□□
HSS



关键功能

功能	描述
信用授权	在允许流量之前向 OCS 请求配额
配额管理	跟踪授予的单位（字节、时间、事件）
信用耗尽检测	监控剩余配额
重新授权	当达到阈值时请求额外配额
服务终止	当信用耗尽时停止服务
最终结算	会话结束时报告实际使用情况



3GPP 计费架构

计费参考点

计费触发功能（CTF）

PGW-C 作为 **CTF（计费触发功能）**，负责：

1. 检测可计费事件 - 会话开始、数据使用、会话结束
2. 请求信用授权 - 在允许服务之前
3. 跟踪配额消耗 - 监控授予的单位
4. 生成计费事件 - 触发信用请求
5. 执行信用控制 - 当配额耗尽时阻止流量

在线计费功能（OCF）

OCS 实现 **OCF（在线计费功能）**：

1. 账户余额管理 - 跟踪用户信用
2. 计费 - 确定每单位价格（每 MB、每秒等）
3. 信用预留 - 为授予的配额预留信用

4. 信用扣除 - 在使用报告时扣除
5. 策略决策 - 根据余额授予或拒绝

Gy/Ro 接口基础

3GPP 参考

- 规范：3GPP TS 32.299 (计费架构)
- 协议：3GPP TS 32.251 (PS 域计费)
- **Diameter 应用 ID**：4 (Gy/Ro - 信用控制应用)
- 基础协议：RFC 4006 (Diameter 信用控制应用)

会话概念

每个需要在线计费的 UE PDN 连接都有一个由 **Session-ID** 标识的 **Gy/Ro** 会话。此会话：

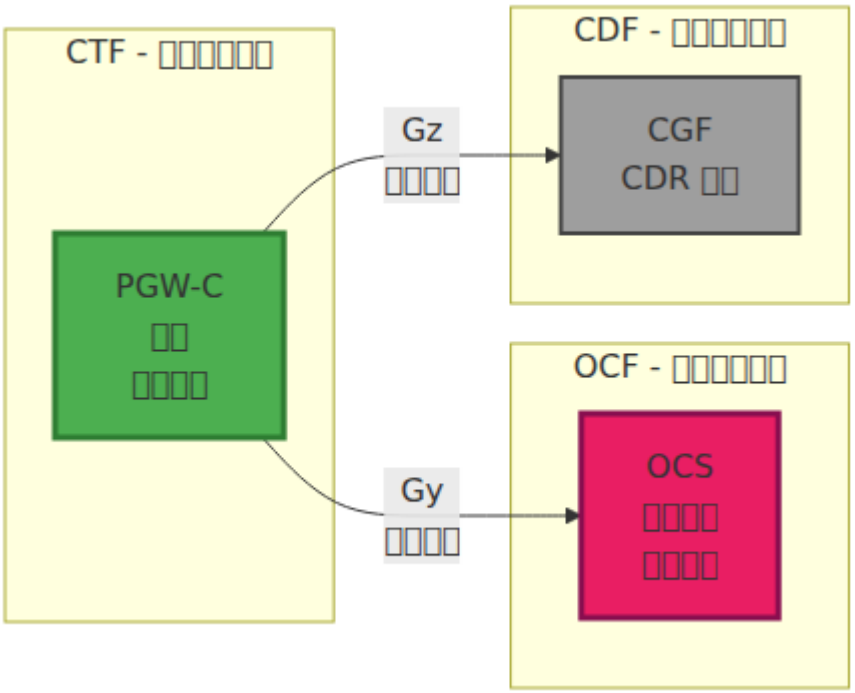
- 当承载需要在线计费时创建（CCR-Initial）
- 当配额被消耗时更新（CCR-Update）
- 当会话结束时终止（CCR-Termination）

会话 ID 格式

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

组件：

- **Origin-Host:** PGW-C 的 Diameter 身份
- **high32:** 唯一标识符的高 32 位
- **low32:** 唯一标识符的低 32 位
- **optional:** 附加标识符（例如，“gy”以区分 Gx）



信用控制消息

消息类型

CCR-Initial (信用控制请求 - 初始)

何时： UE 创建 PDN 连接并且承载需要在线计费

目的:

- 向 OCS 请求初始信用授权
- 为服务交付保留配额
- 建立 Gy/Ro 会话

PGW-C 发送的关键 **AVP**:

AVP 名称	**AVP** 代码	类型	描述
Session-Id	263	UTF8String	唯一的 Gy 会话标识符
Auth-Application-Id	258	Unsigned32	4 (信用控制)
Origin-Host	264	DiamIdent	PGW-C 的 Diameter 身份
Origin-Realm	296	DiamIdent	PGW-C 的 Diameter 域
Destination-Realm	283	DiamIdent	OCS 的域
CC-Request-Type	416	Enumerated	1 = INITIAL_REQUEST
CC-Request-Number	415	Unsigned32	序列号（从 0 开始）
Subscription-Id	443	Grouped	UE 标识符 (IMSI/MSISDN)
Service-Context-Id	461	UTF8String	计费上下文标识符
Multiple-Services-Credit-Control	456	Grouped	特定服务的信用请求
Requested-Service-Unit	437	Grouped	请求的配额（字节、时间等）
Used-Service-Unit	446	Grouped	已使用的配额（初始为 0）
Service-Identifier	439	Unsigned32	服务类型标识符
Rating-Group	432	Unsigned32	计费类别标识符

示例 **CCR-I** 结构:

```
CCR (命令代码: 272, 请求)
├── 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 (请求 10 MB)
└── Used-Service-Unit (Grouped)
    └── CC-Total-Octets: 0 (尚未使用)
```

CCA-Initial (信用控制应答 - 初始)

发送方: OCS 对 CCR-I 的响应

目的:

- 授予或拒绝信用授权
- 提供服务交付的配额
- 指定计费和收费参数

PGW-C 接收的关键 AVP:

AVP 名称	**AVP** 代码	描述
Result-Code	268	成功 (2001) 或错误代码
Multiple-Services-Credit-Control	456	特定服务的信用授予
Granted-Service-Unit	431	授予的配额（字节、时间等）
Validity-Time	448	配额有效期（秒）
Result-Code	268	每服务的结果代码
Final-Unit-Indication	430	当配额耗尽时的操作
Volume-Quota-Threshold	-	重新授权的阈值

成功响应示例：

```
CCA (命令代码: 272, 应答)
├── 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 (授予 10 MB)
    ├── Validity-Time: 3600 (配额有效期为 1 小时)
    └── Volume-Quota-Threshold: 8000000 (在使用 8 MB 时重新授权, 80%)
```

CCR-Update (信用控制请求 - 更新)

何时：

- 授予的配额阈值达到（例如，80% 被消耗）
- 有效时间到期
- 服务变更需要重新授权
- 资费时间变更

目的：

- 请求额外配额
- 报告之前授予的配额使用情况
- 更新计费参数

与 **CCR-I** 的关键区别：

- `CC-Request-Type: UPDATE_REQUEST (2)`
- `CC-Request-Number` 增加
- `Used-Service-Unit` 包含实际使用情况
- `Requested-Service-Unit` 用于更多配额

示例 **CCR-U** 结构：

```
CCR (命令代码: 272, 请求)
├── 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 (目前已使用 8 MB)
    └── Requested-Service-Unit (Grouped)
        └── CC-Total-Octets: 10000000 (请求另外 10 MB)
```

CCA-Update (信用控制应答 - 更新)

发送方: OCS 对 CCR-U 的响应

目的:

- 授予额外配额（如果有信用可用）
- 确认使用情况
- 更新计费参数

可能的结果:

1. 授予更多配额:

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

2. 最终配额（信用耗尽）:

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

3. 无信用可用:

```
CCA (更新)
├── 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 (信用控制请求 - 终止)

何时:

- UE 脱离
- PDN 连接被删除
- 会话因任何原因终止

目的:

- 最终使用报告
- 关闭 Gy/Ro 会话
- 最终结算

关键区别:

- `CC-Request-Type: TERMINATION_REQUEST (3)`
- `Used-Service-Unit` 包含最终使用情况
- 无 `Requested-Service-Unit`（不再需要配额）
- 包括 `Termination-Cause`

示例 **CCR-T** 结构:

```
CCR (命令代码: 272, 请求)
├── 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 (总使用 18.5 MB)
```

CCA-Termination (信用控制应答 - 终止)

发送方: OCS 对 CCR-T 的响应

目的:

- 确认会话终止
- 完成会计
- 释放保留的信用

示例 **CCA-T**:

```
CCA (命令代码: 272, 应答)
├── 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
```

在线计费流程

服务单元类型

OCS 可以以不同单位授予配额：

单元类型	**AVP**	描述	用例
时间	CC-Time	秒	语音通话、会话持续时间
体积	CC-Total-Octets	字节（总上行+下行）	数据服务
体积（分开）	CC-Input-Octets, CC-Output-Octets	字节（分开）	非对称计费
特定服务	CC-Service-Specific-Units	自定义单位	短信、彩信、API 调用
事件	-	计数事件	按使用付费服务

配额阈值管理

问题： PGW-C 如何知道何时请求更多配额？

解决方案： OCS 提供 **Volume-Quota-Threshold** 或 **Time-Quota-Threshold**。PGW-C 通过来自 PGW-U 的 PFCP 会话报告监控使用情况（见 PFCP 接口）。

示例流程：

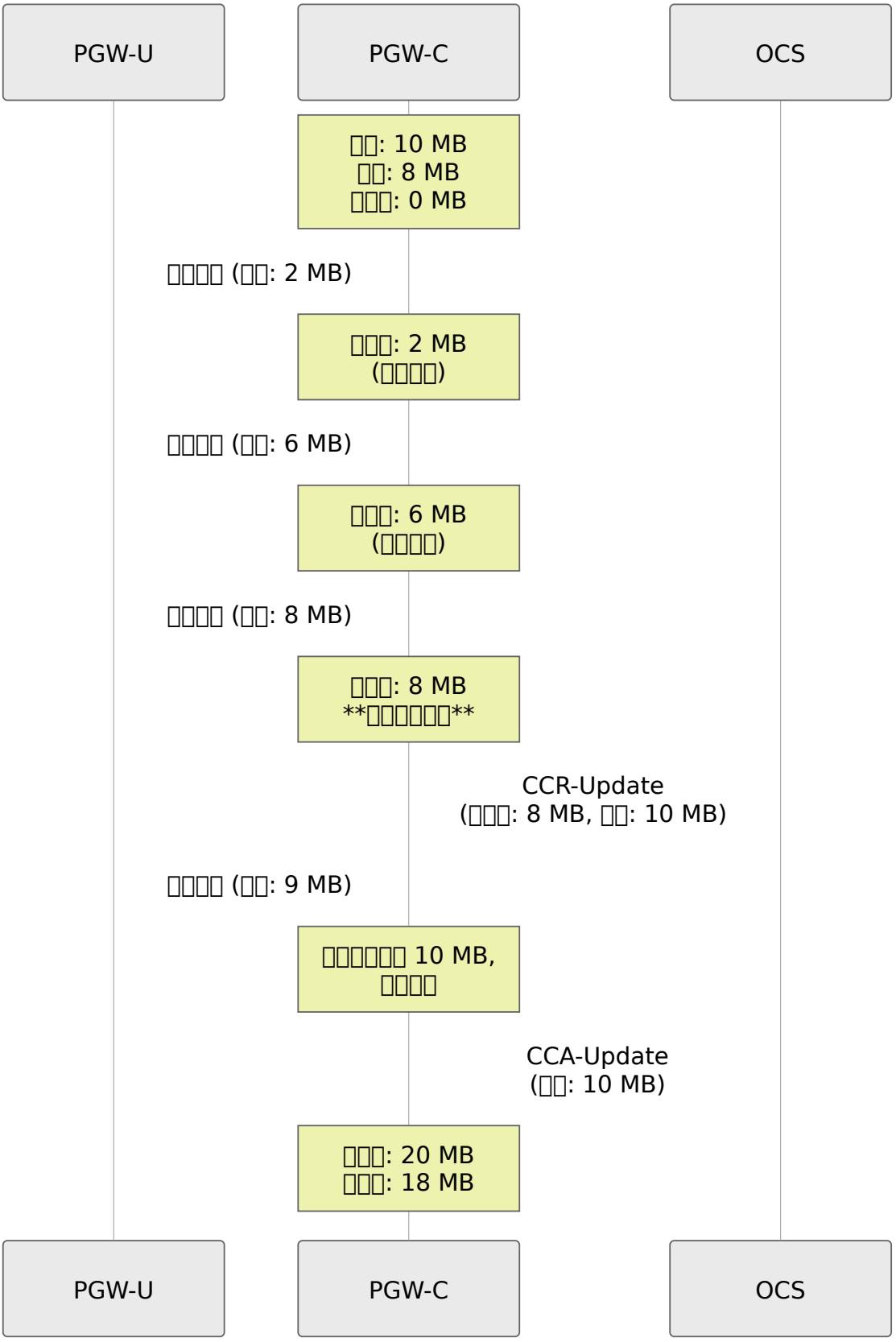
1. OCS 授予 10 MB 配额，阈值为 80%（8 MB）
2. PGW-C 通过 PGW-U 使用报告监控使用情况（PFCP 会话报告）
3. 当使用达到 8 MB：
   → PGW-C 发送 CCR-Update
   → 继续允许流量（不等待响应）
4. OCS 响应更多配额
5. 如果在发送 CCR-Update 之前配额耗尽：
   → PGW-C 必须阻止流量

阈值计算：

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

当 8 MB 被消耗 → 触发 CCR-Update
剩余缓冲区：2 MB（允许 OCS 响应的时间）
```

PGW-C 监控：

PGW-C 通过来自 PGW-U 的 **PFCP** 会话报告 监控使用情况：


PGW-U
PGW-C
OCS
□□: 10 MB
□□: 8 MB
□□□: 0 MB
□□□□ (□□: 2 MB)
□□□: 2 MB
(□□□□)
□□□□ (□□: 6 MB)
□□□: 6 MB
(□□□□)
□□□□ (□□: 8 MB)
□□□: 8 MB
□□□□□□
CCR-Update
(□□□: 8 MB, □□: 10 MB)
□□□□ (□□: 9 MB)
□□□□□□ 10 MB,
□□□□
CCA-Update
(□□: 10 MB)
□□□: 20 MB
□□□: 18 MB
PGW-U
PGW-C
OCS

□□□□□□

□□□□□□□□□□□□□□

OCS 在 CCA 中包含 **Final-Unit-Indication** AVP 以指定操作:

Final-Unit-Action	值	**PGW-C** 行为
TERMINATE	0	阻止所有流量，启动会话终止
REDIRECT	1	将流量重定向到门户（例如，充值页面）
RESTRICT_ACCESS	2	仅允许访问特定服务（例如，充值服务器）

示例：带重定向的最终单元

```
CCA (更新)
└── Multiple-Services-Credit-Control
    ├── Result-Code: DIAMETER_SUCCESS (2001)
    ├── Granted-Service-Unit
    │   └── CC-Total-Octets: 1000000 (最终 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 行动:

1. **TERMINATE:** 发送 CCR-T，删除承载
2. **REDIRECT:** 安装 PFCP 规则以将 HTTP 重定向到充值 URL
3. **RESTRICT_ACCESS:** 安装 PFCP 规则，仅允许白名单 IP 访问

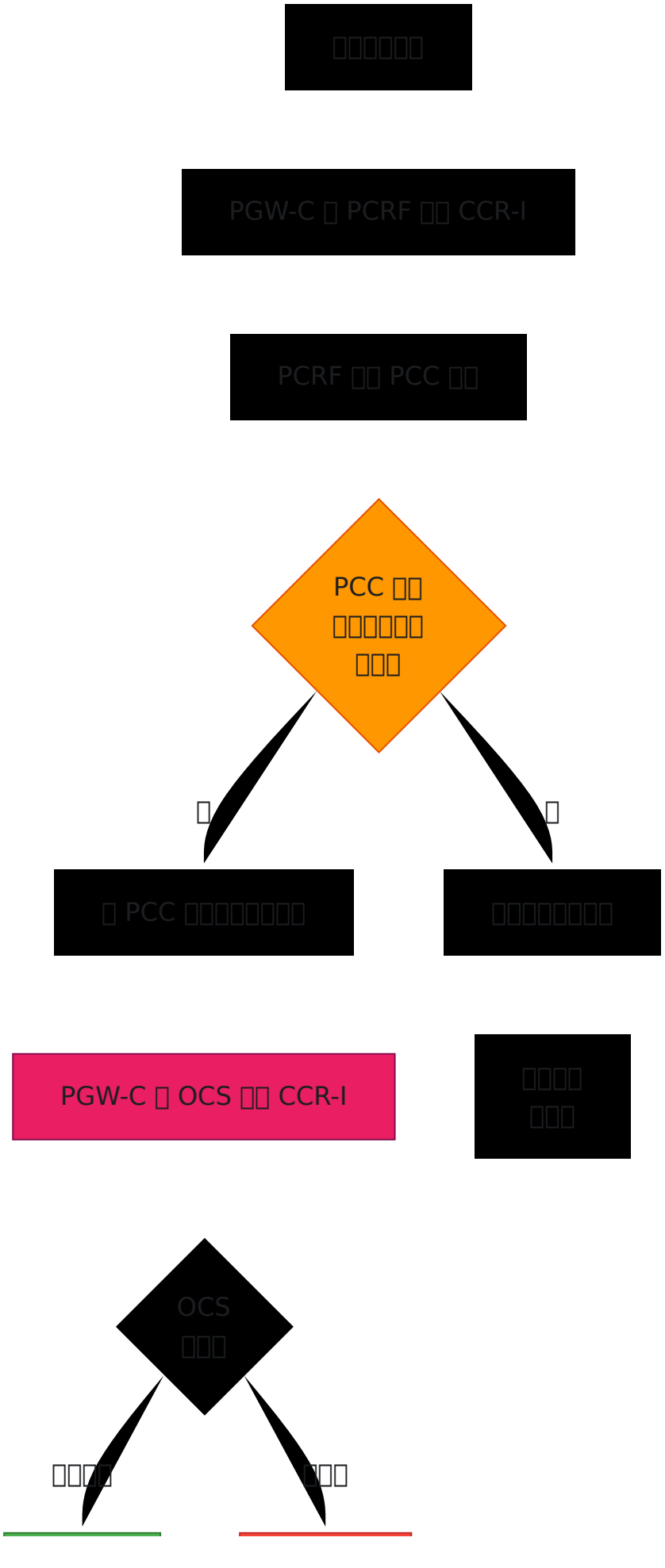
承载计费控制

什么控制承载是否收费?

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

承载计费由 PCRF 通过 Gx 接口提供的 PCC 规则 控制。请参见 Diameter Gx 接口 获取完整的 PCC 规则文档。

计费决策流程：


PGW-C 向 PCRF 发起 CCR-I
PCRF 向 PCC 决策
PCC 决策
向 PCC 发起会话修改
发起会话修改
PGW-C 向 OCS 发起 CCR-I
OCS
决策
允许通过
拒绝通过

□□□□□□□□ PCC □□

PCRF □□ (CCA-I □ Gx)□

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

PCC □□□□□□□□□ AVP□

AVP 名称	AVP 代码	值	描述
Rating-Group	432	Unsigned32	计费类别（映射到 OCS 中的资费）
Online	1009	0=禁用，1=启用	启用在线计费 (Gy)
Offline	1008	0=禁用，1=启用	启用离线计费 (Gz)
Metering-Method	1007	0=持续时间，1=体积，2=两者	计量方式
Reporting-Level	1011	0=服务，1=���费组	使用报告的粒度

承载计费决策矩阵

在线	离线	计费组	行为
1	0	存在	仅在线计费（预付费）
0	1	存在	仅离线计费（后付费）
1	1	存在	在线和离线均可（融合）
0	0	-	无计费（免费服务）

多个计费组

单个 PDN 连接可以有 多个具有不同计费组的承载：

示例场景：

```
默认承载（互联网）
├── Rating-Group: 100 (标准数据)
└── Online: 1

专用承载 1（视频流）
├── Rating-Group: 200 (视频服务)
└── Online: 1

专用承载 2（IMS 语音）
├── Rating-Group: 300 (语音)
└── Online: 1
```

PGW-C Gy 行为:

- 单个 **CCR-I** 具有多个 MSCC（多服务信用控制）部分:

```
CCR-Initial
├── Session-Id: "..."
└── Multiple-Services-Credit-Control
    ├── [Rating-Group: 100] → 标准数据
    ├── [Rating-Group: 200] → 视频服务
    └── [Rating-Group: 300] → 语音
```

OCS 响应:

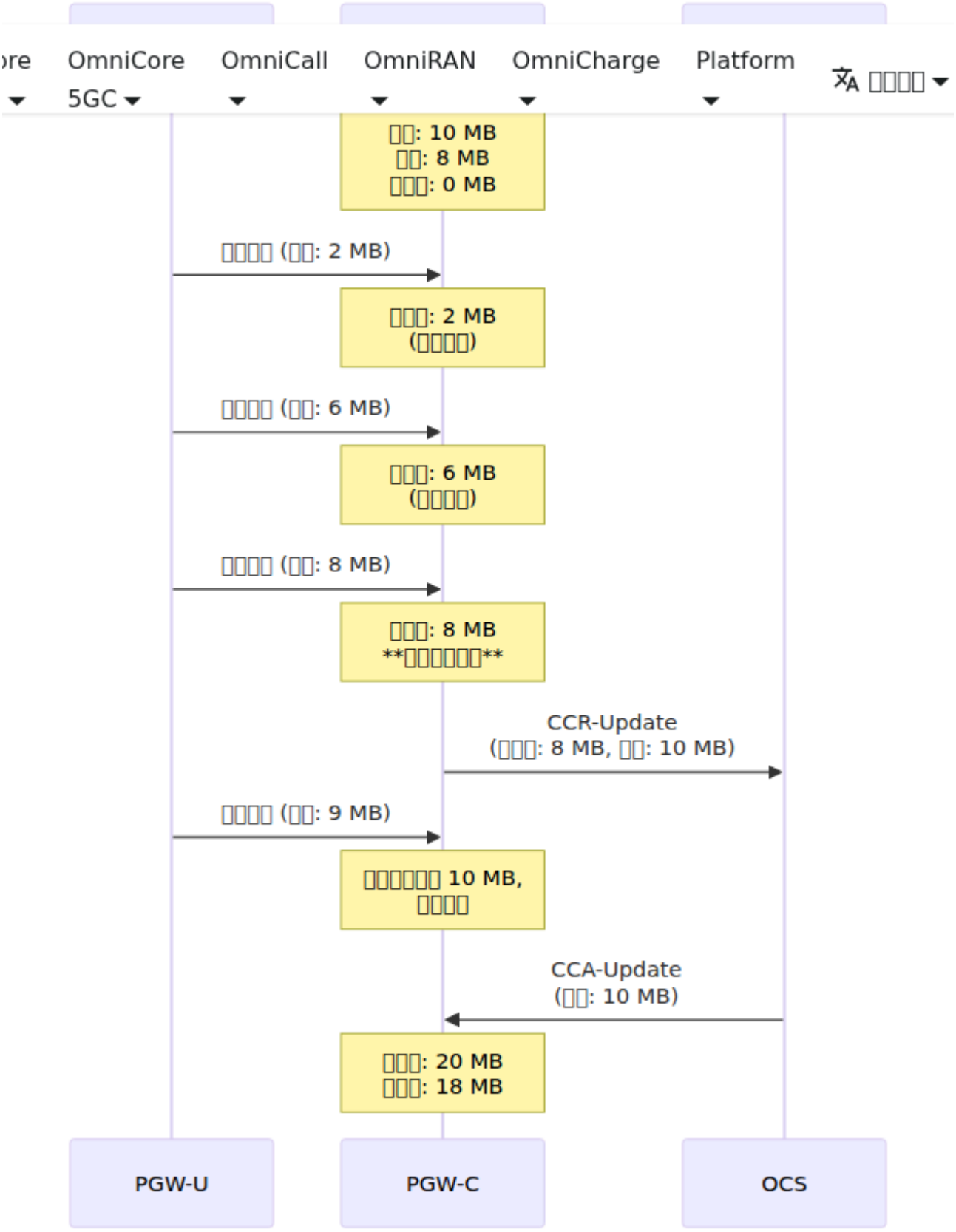
```
CCA-Initial
└── Multiple-Services-Credit-Control
    ├── [Rating-Group: 100] → 授予: 10 MB
    ├── [Rating-Group: 200] → 授予: 5 MB (视频更贵)
    └── [Rating-Group: 300] → 授予: 60 秒
```

按服务计费执行

PGW-C 跟踪每个计费组的配额:

```
# 伪代码
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}  # 毫秒
}
```

每个承载的使用监控：


OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□: 10 MB
□□: 8 MB
□□□: 0 MB
□□□□ (□□: 2 MB)
□□□: 2 MB
(□□□□)
□□□□ (□□: 6 MB)
□□□: 6 MB
(□□□□)
□□□□ (□□: 8 MB)
□□□: 8 MB
□□□□□□
CCR-Update
(□□□: 8 MB, □□: 10 MB)
□□□□ (□□: 9 MB)
□□□□□□ 10 MB,
□□□□
CCA-Update
(□□: 10 MB)
□□□: 20 MB
□□□: 18 MB
PGW-U
PGW-C
OCS

多服务信用控制

MSCC（多服务信用控制）AVP

目的：为特定服务/计费组分组计费信息

结构：

```
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)
│   └── [与 Requested-Service-Unit 相同的结构]
├── Granted-Service-Unit (Grouped, AVP 431)
│   └── [与 Requested-Service-Unit 相同的结构]
├── Validity-Time (Unsigned32, AVP 448)
├── Result-Code (Unsigned32, AVP 268)
└── Final-Unit-Indication (Grouped, AVP 430)
    └── Final-Unit-Action (Enumerated, AVP 449)
```

服务标识符与计费组

属性	服务标识符	计费组
目的	标识服务类型	标识计费类别
示例	1=数据, 2=语音, 3=短信	100=常规, 200=高级
粒度	广泛分类	特定资费
必需	可选	计费时必需
映射	可能映射到多个 RG	OCS 中的单一资费

示例:

```
Service-Identifier: 1 (数据服务)
├── Rating-Group: 100 (标准数据 - $0.01/MB)
└── Rating-Group: 200 (高级数据 - $0.05/MB)

Service-Identifier: 2 (语音)
└── Rating-Group: 300 (语音��话 - $0.10/分钟)
```

配置

基本 Gy 配置

编辑 `config/runtime.exs`:

```
config :pgw_c,
  gy: %{
    # 全局用或禁用在线计费
    enabled: true,

    # OCS 连接超时（毫秒）
    timeout_ms: 5000,

    # 默认配额请求（字节），如果 PCRF 未指定
    default_requested_quota: 10_000_000,  # 10 MB

    # 重新授权的阈值百分比
    # (0.8 = 在消耗 80% 配额时触发 CCR-Update)
    quota_threshold_percentage: 0.8,

    # OCS 超时发生时的操作
    # 选项: :block, :allow
    timeout_action: :block,

    # OCS 返回无信用时的操作
    # 选项: :terminate, :redirect
    no_credit_action: :terminate,

    # 充值的重定向 URL（在 no_credit_action: :redirect 时使用）
    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 对等配置
    peer_list: [
      # PCRF 用于策略控制 (Gx)
      %{
        host: "pcrf.epc.mnc999.mcc999.3gppnetwork.org",
        realm: "epc.mnc999.mcc999.3gppnetwork.org",
        ip: "10.0.0.30",
        initiate_connection: true
      },
      # OCS 用于在线计费 (Gy)
      %{
        host: "ocs.epc.mnc999.mcc999.3gppnetwork.org",
```

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

配置参数解释

`enabled`

- `true`: 在线计费处于活动状态，CCR 消息发送到 OCS
- `false`: 在线计费被禁用，不发送 Gy 消息

`timeout_ms`

- 等待 OCS 返回 CCA 响应的时间
- 推荐: 3000-5000 毫秒

`default_requested_quota`

- 如果 PCRF 未指定，则请求的默认配额
- 典型值: 1-100 MB

`quota_threshold_percentage`

- 当消耗此 % 的配额时触发 CCR-Update
- 推荐: 0.75-0.85（75%-85%）
- 更高 = 更少的消息，但有配额耗尽的风险
- 更低 = 更多的消息，但更安全

`timeout_action`

- **`:block`** - 如果 OCS 未响应则阻止流量（更安全，防止收入损失）
- **`:allow`** - 如果 OCS 未响应则允许流量（更好的用户体验，但有收入风险）

`no_credit_action`

- **`:terminate`** - 当信用耗尽时删除承载

- `:redirect` - 重定向到充值门户

环境特定配置

生产（预付费用户）：

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

测试/开发:

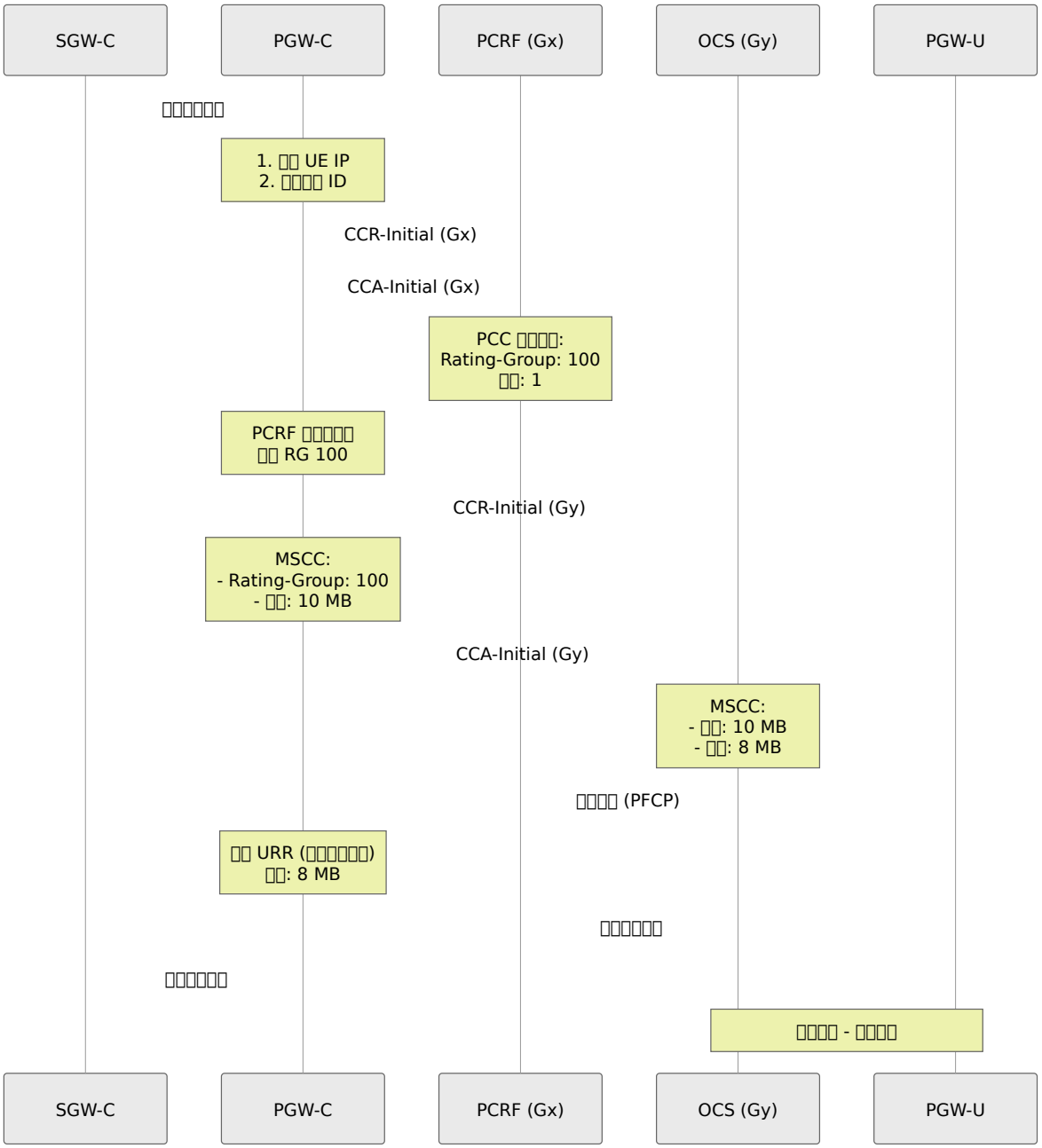
```
config :pgw_c,
  gy: %{
    enabled: false  # 测试时禁用
  }
```

混合（部分预付费，部分后付费）：

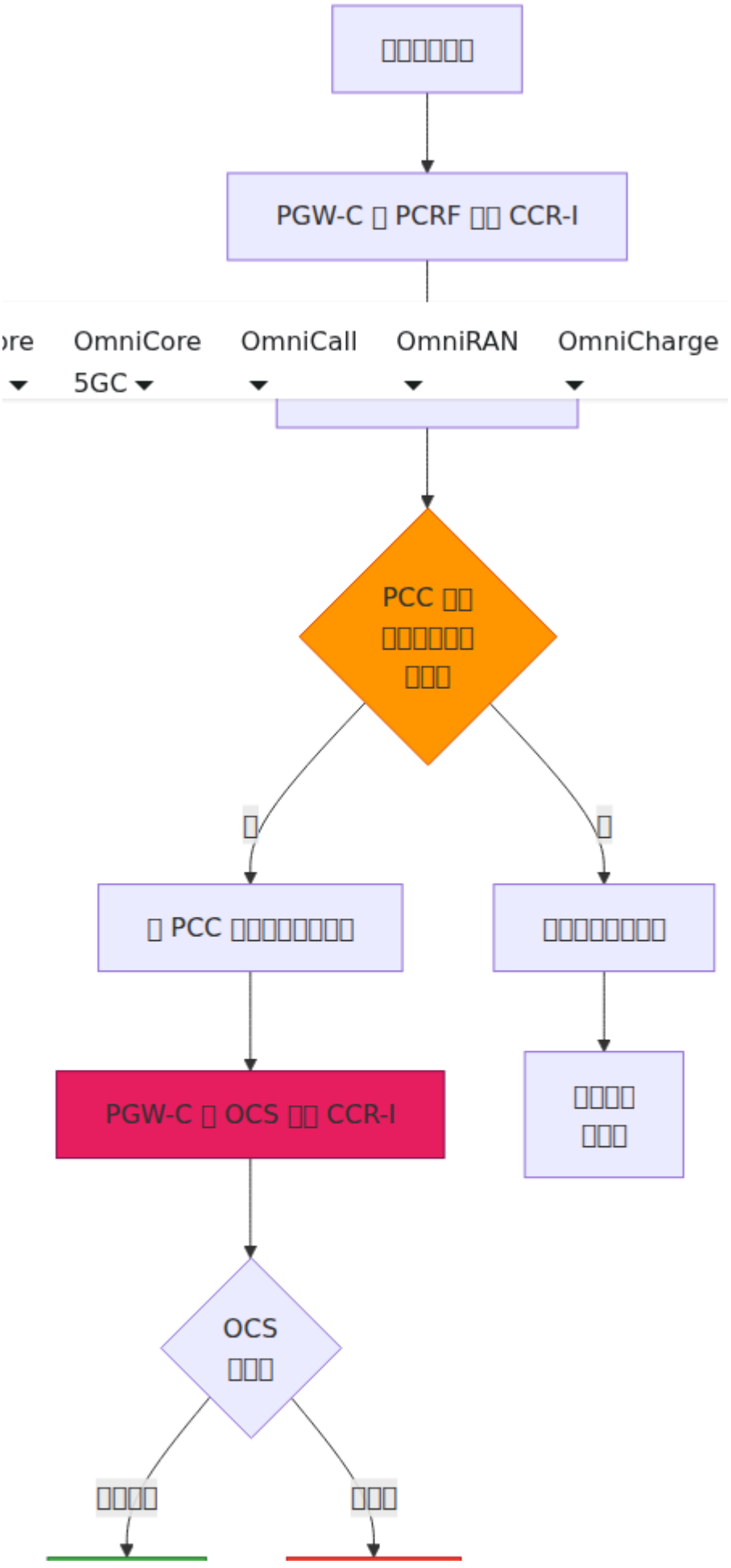
```
config :pgw_c,
  gy: %{
    enabled: true,  # 由 PCRF 按用户控制
    timeout_action: :allow,  # OCS 失败时不阻止后付费
    no_credit_action: :terminate
  }
```

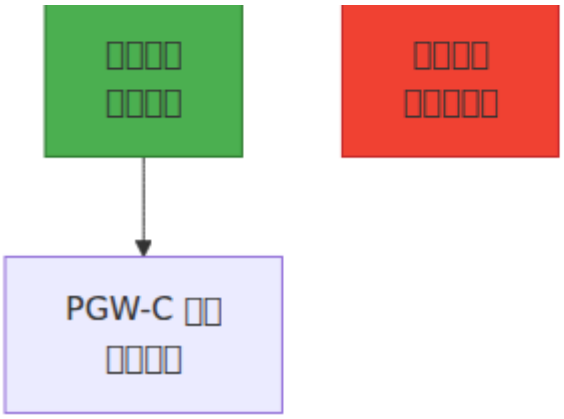
□□□□

□□□□□□□□□□

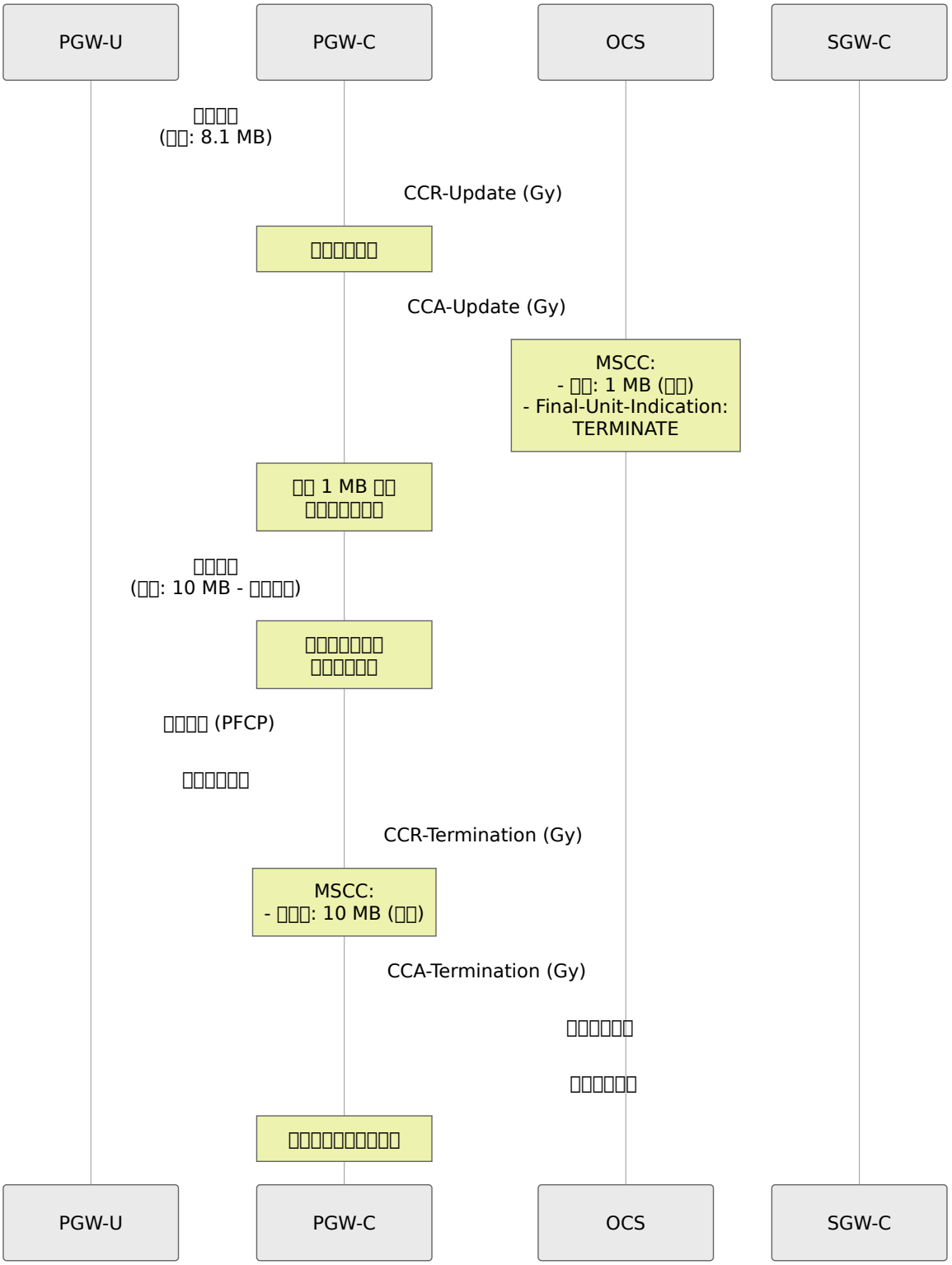

SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
PGW-U
□□□□□□□
1. □□ UE IP
2. □□□□ ID
CCR-Initial (Gx)
CCA-Initial (Gx)
PCC □□□□:
Rating-Group: 100
□□: 1
PCRF □□□□□
□□ RG 100
CCR-Initial (Gy)
MSCC:
- Rating-Group: 100
- □□: 10 MB
CCA-Initial (Gy)
MSCC:
- □□: 10 MB
- □□: 8 MB
□□□□ (PFCP)
□□ URR (□□□□□□□)
□□: 8 MB
□□□□□□□
□□□□□□□
□□□□ - □□□□
SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
PGW-U

□□□□□□ (CCR-Update)


□□□□□□□
PGW-C □ PCRF □□ CCR-I
ore
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
PCC □□ □□□□□□□ □□□
□
□
□ PCC □□□□□□□□□
□□□□□□□□□
PGW-C □ OCS □□ CCR-I
□□□□ □□□
OCS □□□
□□□□
□□□

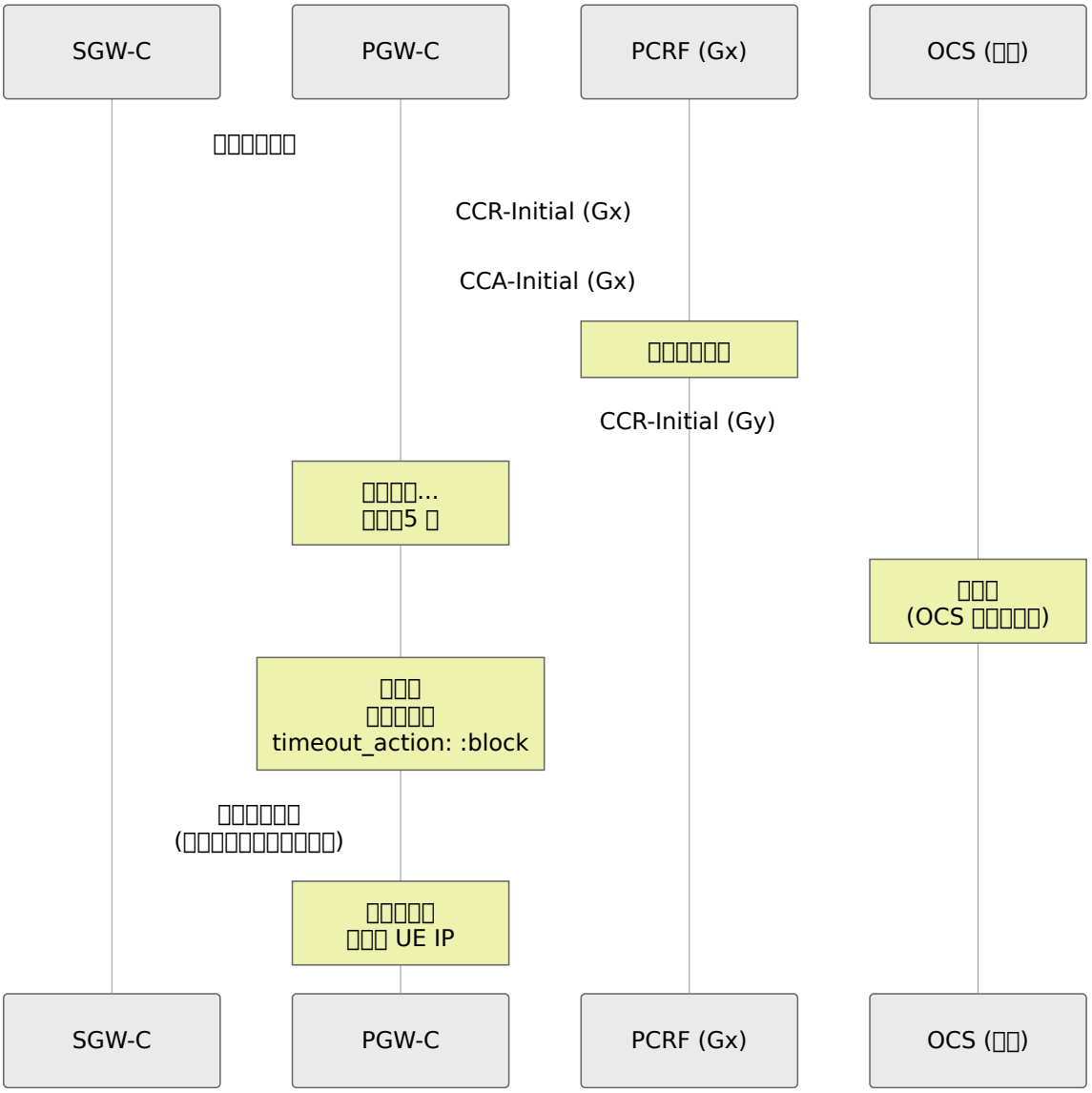

PGW-C

( )



PGW-U
PGW-C
OCS
SGW-C
会话报告
(使用: 8.1 MB)
CCR-Update (Gy)
请求更多配额
CCA-Update (Gy)
MSCC:
- 授予: 1 MB (最终)
- Final-Unit-Indication:
TERMINATE
仅剩 1 MB 配额
这是最后的授予
会话报告
(使用: 10 MB - 配额耗尽)
所有配额已消耗
必须终止会话
会话删除 (PFCP)
会话删除响应
CCR-Termination (Gy)
MSCC:
- 已使用: 10 MB (最终)
CCA-Termination (Gy)
删除承载请求
删除承载响应
会话因信用耗尽而终止
PGW-U
PGW-C
OCS
SGW-C

OCS □□□□


SGW-C
PGW-C
PCRF (Gx)
OCS (□□)
□□□□□□□
CCR-Initial (Gx)
CCA-Initial (Gx)
□□□□□□□
CCR-Initial (Gy)
□□□□...
□□□5 □
□□□
(OCS □□□□□)
□□□
□□□□□□
timeout_action: :block
□□□□□□□
(□□□□□□□□□□□□□)
□□□□□□
□□□ UE IP
SGW-C
PGW-C
PCRF (Gx)
OCS (□□)

□□□□

□□□□

□□□□□

代码	名称	□作
2001	DIAMETER_SUCCESS	继续使用授予的配额

瞬态故障 **(4xxx)**:

代码	名称	**PGW-C** □作
4010	DIAMETER_TOO_BUSY	重试并退避
4011	DIAMETER_UNABLE_TO_COMPLY	记录错误，可能重试
4012	DIAMETER_CREDIT_LIMIT_REACHED	终止或重定向

永久故障 **(5xxx)**:

代码	名称	**PGW-C** □作
5003	DIAMETER_AUTHORIZATION_REJECTED	拒绝会话
5031	DIAMETER_USER_UNKNOWN	拒绝会话（无效用户）

按服务结果代码

重要： Result-Code 可以出现在 两个级别:

1. 消息级别 - 整体结果
2. **MSCC** 级别 - 每服务结果

示例:

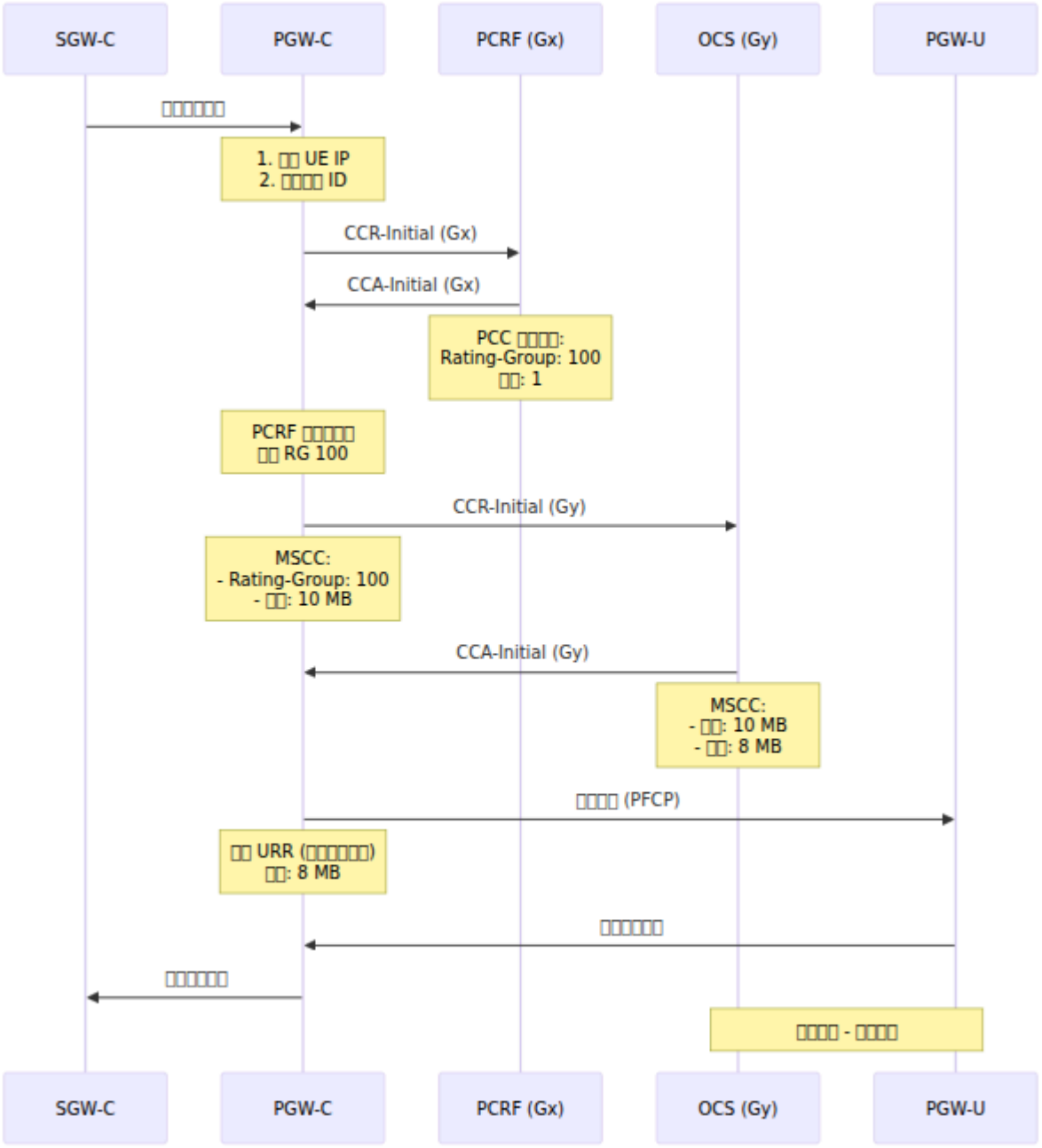
```
CCA-Initial
├── Result-Code: DIAMETER_SUCCESS (2001)  ← 消息级别: OK
└── Multiple-Services-Credit-Control
    ├── [Rating-Group: 100]
    │   └── Result-Code: DIAMETER_SUCCESS (2001)  ← RG 100: OK
    └── [Rating-Group: 200]
        └── Result-Code: DIAMETER_CREDIT_LIMIT_REACHED (4012)  ← RG 200: 无信用
```

PGW-C 行为:

- 允许 RG 100 的流量
- 阻止 RG 200 的流量

与 **Gx** 的集成

Gx 接口（PCRF 策略控制）确定是否需要在线计费，并提供驱动 Gy 计费的 Rating-Group。请参见 Diameter Gx 接 获取完整的策略控制文档



Gx 与 Gy 的关系

集成流程

1. 承载设置:

```
PGW-C 接收到创建会话请求
      ↓
发送 CCR-I 到 PCRF (Gx)
      ↓
接收 CCA-I 及 PCC 规则
      ↓
解析 PCC 规则:
      - 规则是否有计费组?
      - 在线 = 1 吗?
      ↓
如果是:
      发送 CCR-I 到 OCS (Gy) 及计费组
      ↓
      接收 CCA-I 及配额
      ↓
      如果授予配额: 继续
      如果无信用: 拒绝承载
如果否:
      不进行在线计费，继续
```

2. 动态策略更新 **(PCRF** 的 **RAR)**:

```
PCRF 通过 Gx 发送 RAR (重新授权请求)
      ↓
新的 PCC 规则添加，在线=1，计费组=200
      ↓
PGW-C 向 OCS (Gy) 发送 CCR-U
      - 为计费组 200 添加 MSCC
      ↓
OCS 授予新服务的配额
      ↓
安装带在线计费的专用承载
```

故障排除

常见问题

1. CCR-Initial 到 OCS 超时

症状：

- 会话失败，显示“OCS 超时”
- 日志：“CCR-Initial (Gy) 超时”

可能原因：

- OCS 无法访问
- 配置中的 OCS IP 不正确
- 防火墙阻止 Diameter 端口（3868）
- OCS 过载

解决方案：

```
# 测试网络连接
ping <ocs_ip>

# 测试 Diameter 端口 (TCP 3868)
telnet <ocs_ip> 3868

# 检查配置
# 确保在 peer_list 中配置了 OCS 对等方
```

2. 会话被 OCS 拒绝

症状：

- CCA-I 的 Result-Code != 2001
- 创建会话响应失败

常见结果代码：

结果代码	可能原因	解决方案
4012	信用限额达到	用户需要充值
5003	授权被拒绝	检查用户权限
5031	用户未知	在 OCS 中配置用户

调试步骤：

1. 检查 OCS 日志以获取拒绝原因
2. 验证 OCS 中的用户余额
3. 检查 CCR-I 中的 IMSI/MSISDN 是否与用户记录匹配

3. 未检测到配额耗尽

症状：

- 用户在余额耗尽后继续使用数据
- 未发送 CCR-Update

可能原因：

- PGW-U 中未安装 URR（使用报告规则）
- 阈值未正确配置
- 未收到 PFCP 会话报告

调试步骤：

1. 验证 PFCP 会话建立中的 URR：

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

2. 检查 PGW-U 日志以获取使用报告

3. 验证配置中的 quota_threshold_percentage

4. 不正确的计费组

症状：

OCS 拒绝，显示“未知计费组”
会话失败

原因：

CCR-I 中的计费组与 OCS 配置不匹配
PCRF 提供了无效的计费组

解决方案：

1. 验证 PCRF 中 PCC 规则的计费组
2. 检查 OCS 配置以获取有效的计费组
3. 确保 PCC 规则与 OCS 资费之间的映射

监控

关键指标

```
# Gy 消息速率
rate(gy_inbound_messages_total{message_type="cca"}[5m])
rate(gy_outbound_messages_total{message_type="ccr"}[5m])

# Gy 错误速率
rate(gy_inbound_errors_total[5m])

# 配额耗尽事件
rate(gy_quota_exhausted_total[5m])

# OCS 超时率
rate(gy_timeout_total[5m])

# Gy 消息处理持续时间
histogram_quantile(0.95,
rate(gy_inbound_handling_duration_bucket[5m]))
```

警报

```
# □ Gy 错误率警报
- alert: GyErrorRateHigh
  expr: rate(gy_inbound_errors_total[5m]) > 0.1
  for: 5m
  annotations:
    summary: "检测到高 Gy 错误率"

# OCS 超时警报
- alert: OcsTimeout
  expr: rate(gy_timeout_total[5m]) > 0.05
  for: 2m
  annotations:
    summary: "发生 OCS 超时"

# 信用耗尽激增警报
- alert: CreditExhaustionSpike
  expr: rate(gy_quota_exhausted_total[5m]) > 10
  for: 5m
  annotations:
    summary: "高信用耗尽率"
```

Gy 信用控制（OAM API）

在线计费通过 **OAM REST API** 进行检查和操作，通过 HTTPS/TLS 提供，地址为 `https://<host>:8443`。交互式 API 文档（Swagger UI）可在 `https://<host>:8443/api/docs` 获取。

在线计费状态

```
GET /api/charging                    # 所有在线计费会话 + 实时信用状态
GET /api/charging?search=<imsi>      # 按 IMSI 过滤会话
GET /api/charging/<imsi>             # 按 IMSI 获取单个会话的信用控制状态
```

示例：

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

`GET /api/charging` 返回在线计费状态（授予与使用的配额、有效性、OCS 最终单元操作，以及围墙花园/余额耗尽重定向状态）对于每个活动会话。添加 `?search=<imsi>` 以缩小列表，或将 IMSI 附加到 `GET /api/charging/<imsi>` 以获取单个用户的信用控制状态。

Gy 信用控制模拟器

```
POST /api/gy_simulator
```

发送合成的信用控制请求到配置的 OCS，并返回解析的应答——命令级结果代码、授予的配额、有效时间，以及任何最终单元指示——以便在线计费可以在 没有实时用户流量 的情况下进行操作。这是 OAM API 替代旧的基于浏览器的 Gy 模拟器。它要求 Gy 被启用（`gy.enabled: true`）；请求从 PGW-C 自身的 Gy 原始主机发送到配置的 OCS 域。

请求体（`application/json`）：

字段	类型	描述
`imsi`	string	必需。用户 IMSI。
`msisdn`	string	可选 MSISDN (E.164)。填充 `User-Name` 为 `<msisdn>@<apn>`。
`apn`	string	可选 APN。
`requested_units`	integer	可选请求的配额，以字节为单位 (Requested-Service-Unit → CC-Total-Octets)。
`rating_group`	integer	可选计费组/计费键（默认为 `1`）。
`service_id`	integer	可选服务标识符。
`cc_request_type`	string	`initial`（默认）、`update` 或 `terminate`（或数字 `1`/`2`/`3`）。

示例——请求 1 MB 的初始配额：

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

示例响应（CCA-Initial 授予 1 MB，有效期 1 小时）：

```
{
  "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"
}
```

在失败时，端点返回 `422`（`imsi` 缺失）或 `503`（Gy 被禁用，或 OCS 未响应——`reason` 字段携带底层错误，例如 `:timeout`）。

□作用例

- 检查用户的配额 — `GET /api/charging/<imsi>` 返回会话的当前授予配额和信用状态。
- 测试 **OCS** 集成而不产生流量 — `POST /api/gy_simulator` 确认 OCS 授予预期的配额/最终单元操作给定 IMSI、计费组或服务。
- 故障排除计费问题 — 使用 `GET /api/charging?search=<imsi>` 定位受影响的用户并查��其实时信用控制状态。

对于历史趋势（消息速率、配额耗尽、OCS 超时），使用上面 监控 中描述的 Prometheus 指标。

相关文档

计费与策略

- **Diameter Gx** 接口 - PCRF 策略控制，触发在线计费的 PCC 规则
- 数据 **CDR** 格式 - 后付费计费的离线计费记录
- 配置指南 - 完整的在线计费配置参数

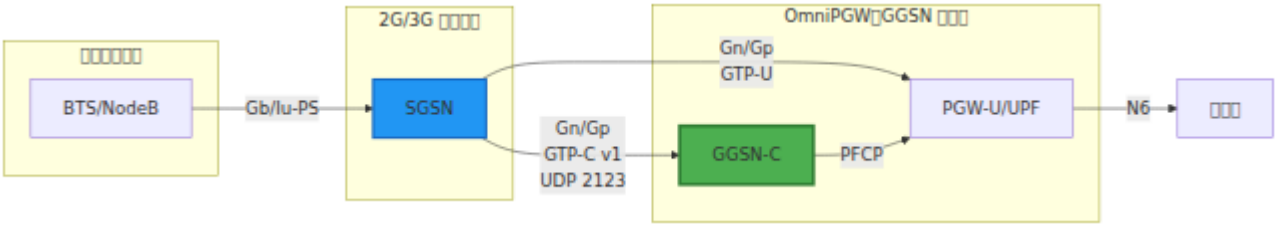
会话管理

- 会话管理 - PDN 会话生命周期，承载管理
- **PFCP** 接口 - 通过 URR 从 PGW-U 进行使用报告
- **S5/S8** 接口 - GTP-C 承载设置和拆除

□作

- 监控指南 - Gy 指标、配额跟踪、OCS 超时警报
- **UE IP** 分配 - 收费会话的 IP 池配置

返回□作指南



Gn/Gp 接口文档

与 **SGSN** 的 **GTP-C** 通信（**2G/3G** 网络）

OmniPGW □ Omnitouch □□□□□□

概述

Gn/Gp 接口使 OmniPGW 能够作为 2G/3G 网络的 **GGSN**（网关 GPRS 支持节点）使用 **GTP-C v1** 协议。该接口处理 GGSN 与 SGSN 之间的 PDP（分组数据协议）上下文管理。

- **Gn** 接口：连接同一 PLMN（本地网络）内的 SGSN
- **Gp** 接口：连接其他 PLMN（漫游场景）中的 SGSN

这两个接口使用相同的协议 - 区别仅在于拓扑结构。

与 **PGW (S5/S8)** 的主要区别

方面	**GGSN (Gn/Gp)**	**PGW (S5/S8)**
协议版本	GTP-C v1	GTP-C v2
3GPP 规范	TS 29.060	TS 29.274
会话概念	PDP 上下文	PDN 会话 + 承载
承载标识符	NSAPI (0-15)	EBI (5-15)
策略接口	无（静态策略）	Gx (PCRF)
QoS 模型	R99 QoS 配置文件	EPS 承载 QoS (QCI)
端口	2123（共享）	2123

协议细节

GTP-C 版本 **1**

- 协议: GTP-C v1 (3GPP TS 29.060)
- 传输: UDP
- 端口: 2123（与 S5/S8 GTP-C v2 共享）
- 接口类型: 控制平面

TEID（隧道端点标识符）

每个 PDP 上下文都有唯一的 TEID 用于控制平面和用户平面:

- **GGSN** 控制 **TEID**: 由 OmniPGW 为传入控制消息分配
- **GGSN** 数据 **TEID**: 由 UPF 为传入用户数据分配

- **SGSN** 控制 **TEID**: 由 SGSN 提供，用于传出控制消息
- **SGSN** 数据 **TEID**: 由 SGSN 提供，用于传出用户数据

消息流:
```
SGSN → GGSN: 目标 TEID = GGSN 的控制 TEID
GGSN → SGSN: 目标 TEID = SGSN 的控制 TEID
```

CUPS 支持（控制平面和用户平���分离）

OmniPGW 支持 2G/3G 的 CUPS 架构:

- 信令的 **GSN** 地址: OmniPGW 的控制平面 IP（来自 `gn.local_ipv4_address`）
- 用户流量的 **GSN** 地址: UPF 的 IP 地址（来自 PFCP 会话建立响应）

这允许 GGSN-C 和 PGW-U/UPF 在不同的节点上运行，具有不同的 IP 地址，这在 CUPS 部署中是标准做法。

NSAPI（网络服务接入点标识符）

NSAPI 用于识别订阅者会话中的单个 PDP 上下文:

- 范围: 5-15（0-4 保留）
- 使用: 每个订阅者可以有多个 PDP 上下文
- 等效: 类似于 LTE/EPC 中的 EBI

配置

基本配置

```
# config/runtime.exs
config :pgw_c,
  # 当存在 gn 配置时，Gn/Gp 接口自动□用
  gn: %{
    # Gn 接口的本地 IPv4 地址
    local_ipv4_address: "10.0.0.20",

    # 可选：本地 IPv6 地址
    local_ipv6_address: nil,

    # GTP-C v1 的端口（与 S5/S8 共享）
    local_port: 2123
  },

  # 在 PCO 中返回的 DNS 服务器（与 PGW 共享）
  dns: %{
    primary_ipv4: {8, 8, 8, 8},
    secondary_ipv4: {8, 8, 4, 4}
  }
```

Gn 接口参数

参数	类型	必需	默认	描述
`local_ipv4_address`	字符串	是	-	用于 Gn 接口的绑定 IPv4 地址。这是向 SGSN 广告的 GGSN 地址。
`local_ipv6_address`	字符串	否	`nil`	可选的 IPv6 地址。大多数 2G/3G 网络仅支持 IPv4。
`local_port`	整数	否	2123	GTP-C 的 UDP 端口。根据 3GPP TS 29.060 的标准端口。与 S5/S8 共享。

DNS □□

□□	□□	□□	□□	□□
`primary_ipv4`	□□	□	`{8, 8, 8, 8}`	□ PCO □□□□□ DNS □□□ IPv4 □□
`secondary_ipv4`	□□	□	`{8, 8, 4, 4}`	□ PCO □□□□□ DNS □□□ IPv4 □□
`primary_ipv6`	□□	□	`nil`	□ DNS □□□ IPv6 □□□
`secondary_ipv6`	□□	□	`nil`	□ DNS □□□ IPv6 □□□

□□□□

□□□□□□□

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

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

□□□□

PDP □□□□□

□□ **PDP** □□□□□

方向: SGSN → GGSN

目的: 建立新的 PDP 上下文以实现数据连接

关键信息元素（**IEs**）:

IE 名称	类型	描述
IMSI	身份	国际移动用户身份
NSAPI	整数	网络服务接入点标识符 (5-15)
TEID 数据 I	整数	SGSN 的用户平面隧道端点
TEID 控制	整数	SGSN 的控制平面隧道端点
终端用户地址	地址	请求的 PDN 类型和可选 IP
接入点名称	字符串	连接的 APN
GSN 地址（信令）	IP	SGSN 信令平面地址
GSN 地址（用户）	IP	SGSN 用户平面地址
QoS 配置文件	二进制	请求的 QoS 参数
MSISDN	身份	手机号码（可选）
选择模式	枚举	APN 选择模式
协议配置选项	选项	其他协议选项

示例:

```
创建 PDP 上下文请求
├── IMSI: 310260123456789
├── NSAPI: 5
├── TEID 数据 I: 0x12345678
├── TEID 控制: 0x87654321
├── 终端用户地址: IPv4（动态）
├── APN: internet
├── GSN 地址（信令）: 10.1.1.100
├── GSN 地址（用户）: 10.1.1.100
├── QoS 配置文件: [R99 QoS 参数]
└── PCO: [DNS 请求等]
```

创建 **PDP** 上下文响应

方向: GGSN → SGSN

目的: 确认 PDP 上下文创建并分配资源

关键信息元素:

IE 名称	类型	描述
原因	结果	成功或错误代码
需要重新排序	���尔	是否需要重新排序
恢复	计数器	GGSN 重启计数器
TEID 数据 I	整数	GGSN 的用户平面隧道端点
TEID 控制	整数	GGSN 的控制平面隧道端点
NSAPI	整数	确认的 NSAPI
计费 ID	二进制	计费关联 ID
终端用户地址	地址	分配的 UE IP 地址
GSN 地址（信令）	IP	GGSN 信令平面地址
GSN 地址（用户）	IP	GGSN 用户平面地址
QoS 配置文件	二进制	协商的 QoS 参数
协议配置选项	选项	DNS 服务器等。

成功响应：

```
创建 PDP 上下文响应
├── 原因: 请求已接受 (128)
├── TEID 数据 I: 0xAABBCCDD
├── TEID 控制: 0xDDCCBBAA
├── NSAPI: 5
├── 计费 ID: 0x11223344
├── 终端用户地址
│   └── IPv4: 100.64.1.42
├── GSN 地址（信令）: 10.0.0.20
├── GSN 地址（用户）: 10.0.0.20
├── QoS 配置文件: [协商的 QoS]
└── PCO
    ├── 主 DNS: 8.8.8.8
    └── 次 DNS: 8.8.4.4
```

更新 **PDP** 上下文请求

方向: SGSN → GGSN

目的: 修改现有的 PDP 上下文（例如，SGSN 迁移、QoS 更改）

常见场景:

- 跨 SGSN 切换（路由区域更新）
- QoS 重新协商
- SGSN 地址更改

关键信息元素:

IE 名称	描述
NSAPI	PDP 上下文标识符
TEID 数据 I	新的 SGSN 用户平面 TEID
GSN 地址	新的 SGSN 地址
QoS 配置文件	更新的 QoS（可选）

更新 **PDP** 上下文响应

方向: GGSN → SGSN

目的: 确认 PDP 上下文更新

关键信息元素:

IE 名称	描述
原因	成功或错误代码
TEID 数据 I	GGSN 用户平面 TEID
GSN 地址	GGSN 地址
QoS 配置文件	协商的 QoS

删除 **PDP** 上下文请求

方向: SGSN → GGSN

目的: 终止 PDP 上下文

关键信息元素:

IE 名称	描述
NSAPI	要删除的 PDP 上下文
拆除指示	删除此订阅者的所有上下文

删除 **PDP** 上下文响应

方向: GGSN → SGSN

目的: 确认 PDP 上下文删除

关键信息元素:

IE 名称	描述
原因	成功或错误代码

路径管理

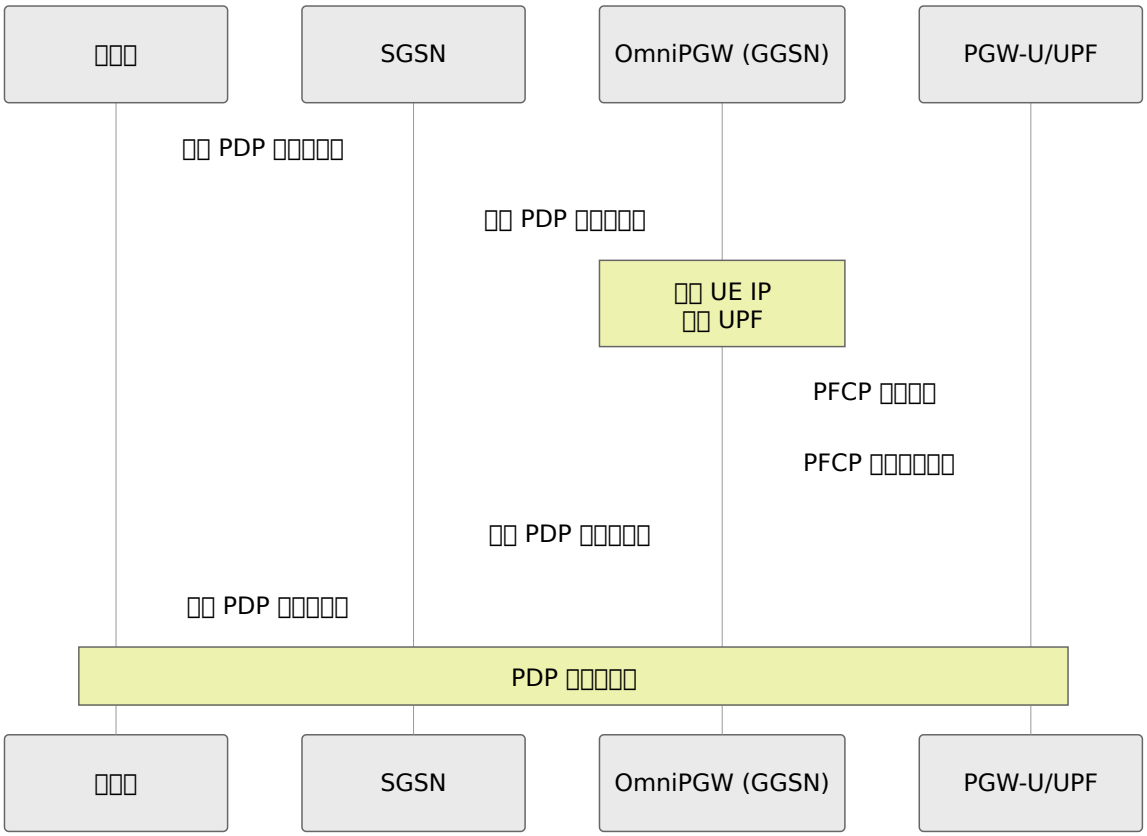
回声请求 / 响应

方向：双向

目的：路径保持活动和重启检测

关键信息元素：

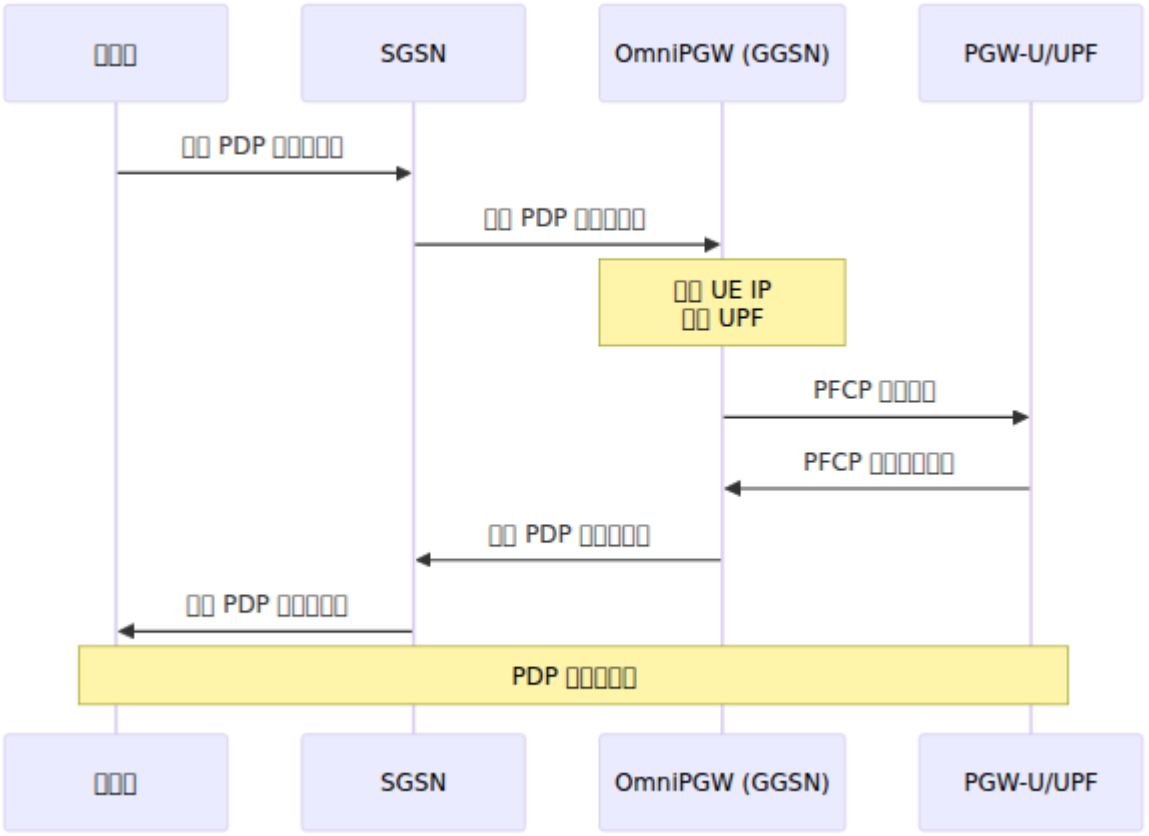
IE 名称	描述
恢复	检测节点重启的重启计数器



消息???

PDP 上下文激活

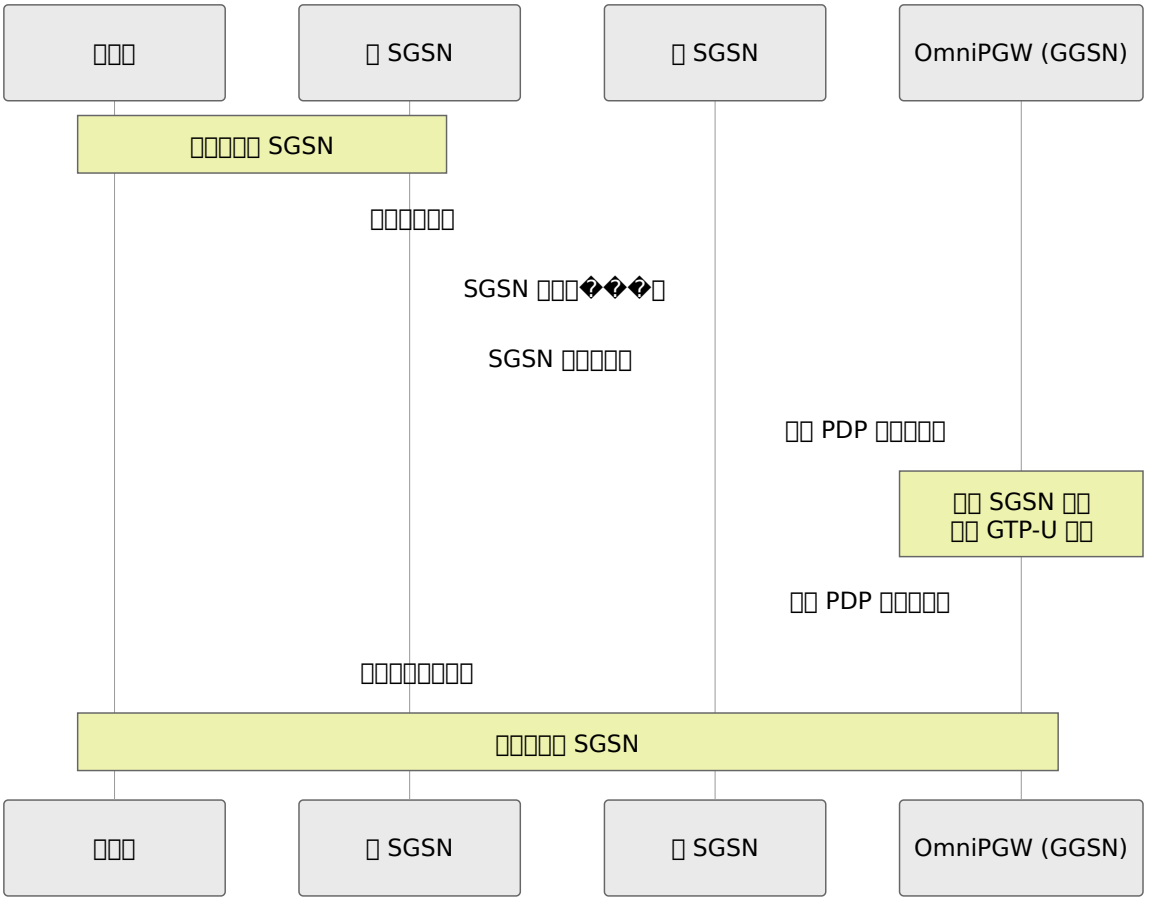
移动台
SGSN
OmniPGW (GGSN)
PGW-U/UPF
激活 PDP 上下文请求
创建 PDP 上下文请求
分配 UE IP
选择 UPF
PFCP 会话建立
PFCP 会话建立响应
创建 PDP 上下文响应
激活 PDP 上下文接受
PDP 上下文激活
移动台
SGSN
OmniPGW (GGSN)
PGW-U/UPF



PDP 上下文停用

□□□
SGSN
OmniPGW (GGSN)
PGW-U/UPF
□□ PDP □□□□□
□□ PDP □□□□□
□□ UE IP
□□ UPF
PFCP □□□□
PFCP □□□□□□
□□ PDP □□□□□
□□ PDP □□□□□
PDP □□□□□
□□□
SGSN
OmniPGW (GGSN)
PGW-U/UPF

□ SGSN □□



□□□□□□

OmniPGW □ GGSN □□□ PGW □□□□□□□□□□

UE IP □□□□

□□□ PGW □□□ AddressRegistry □□□□□□□□□□□□□□□□□□□□□□□□ UE IP □□□

UPF/PFCP □□□□

□□□ PGW □□□ PFCP □□□□□□

- □□ PDP □□□□□□ PDR/FAR/QER □□
- □□□□□□ upf_selection □□□□ UPF

- 有关详细信息，请参见 PFCP 接口。

在线计费（Gy）

可选的 Diameter Gy 集成可用：

- 预付费用户的信用控制
- 与 PGW 模式相同的 Gy 配置
- 有关配置，请参见 Diameter Gy 接口。

CDR 生成

使用与 PGW 相同的 CDR 生成。有关详细信息，请参见 数据 CDR 格式。

原因代码

成功

代码	名称	描述
128	请求已接受	操作成功

拒绝原因

代码	名称	使用时机
192	不存在	找不到 PDP 上下文
193	无效的消息格式	格式错误的 GTP 消息
194	IMSI 不知	找不到订阅者
195	MS 已 GPRS 脱离	MS 未附着
196	MS 未 GPRS 响应	MS 无法到达
197	MS 拒绝	MS 拒绝请求
198	版本不支持	GTP 版本不匹配
199	无可用资源	资源耗尽
200	服务不支持	不支持的服务
201	强制 IE 不正确	IE 验证失败
202	强制 IE 缺失	缺少必需的 IE
203	可选 IE 不正确	可选 IE 无效
204	系统故障	内部错误
205	漫游限制	不允许漫游
206	P-TMSI 签名不匹配	安全失败
207	GPRS 连接暂停	连接已暂停
208	身份验证失败	身份验证失败

代码	名称	使用时机
209	用户身份验证失败	用户身份验证失败

监控

Gn 接口指标

```
# 消息计数器
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"

# 事务延迟
gn_outbound_transaction_duration_bucket
```

有用的查询

PDP 上下文创建率:

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

成功率:

```
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]))
```

活动 **PDP** 上下文:

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

故障排除

问题：对创建 **PDP** 上下文没有响应

症状:

- SGSN 发送创建 PDP 上下文请求
- 没有收到响应
- SGSN 超时

可能原因:

1. 网络连接问题
2. OmniPGW 未在配置的 IP 上监听
3. 防火墙阻止 UDP 2123
4. UPF/PFCP 建立失败

调试:

```
# 检查 OmniPGW 是否在监听
netstat -ulnp | grep 2123

# 检查传入数据包
tcpdump -i any -n port 2123

# 验证配置
grep "local_ipv4_address" config/runtime.exs

# 检查 PFCP 连接
curl http://pgw:9090/metrics | grep pfcp
```

问题：PDP 上下文创建被拒绝

症状：

- 创建 PDP 上下文响应带有错误原因
- PDP 上下文未建立

常见原因：

```
原因 199（无可用资源）：
  → IP 池耗尽
  → 检查：curl http://pgw:9090/metrics | grep address_registry_count
  → 扩展 ue.subnet_map 中的 IP 池

原因 202（强制 IE 缺失）：
  → SGSN 未发送所需的 IEs
  → 检查 SGSN 配置
  → 审查 tcpdump 中的消息

原因 204（系统故障）：
  → UPF/PFCP 故障
  → 检查 PFCP 关联状态
  → 审查 OmniPGW 日志
```

问题：更新 PDP 上下文失败

症状：

- 更新 PDP 上下文响应带有错误原因
- 切换失败

可能原因:

- 找不到 PDP 上下文（TEID 不同）
- NSAPI 不匹配
- 会话已删除

调试:

```
# 检查活动会话
curl http://pgw:9090/metrics | grep session_count

# 审查日志以查找上下文查找
journalctl -u omnipgw | grep "PDP Context"
```

最佳实践

网络设计

1. 与 **S5/S8** 共享端口
    - Gn 和 S5/S8 共享 UDP 端口 2123
    - OmniPGW 自动检测 GTP 版本
    - 单个 IP 可以同时服务于 2G/3G 和 4G
2. **SGSN** 连接性
    - 确保所有 SGSN 可以访问 GGSN IP
    - 考虑为 Gn 流量设置专用 VLAN
    - 如果适用，计划漫游（Gp）
3. **IP** 地址规划
    - 相同的 IP 池同时服务于 PGW 和 GGSN 模式

考虑为 2G/3G 设置基于 APN 的独立池

性能

1. 会话容量

监控 session_count 指标
计划预期的 2G/3G 订阅者数量
考虑到 2G/3G 通常具有较少的并发会话

2. UDP 缓冲区大小

与 S5/S8 相同的建议
典型：每个套接字 4-8 MB

迁移考虑

从独立 GGSN 迁移到 OmniPGW 时：

1. APN 配置

确保 APN 与现有 GGSN 配置匹配
验证 PCO 中的 DNS 服务器设置

2. IP 池迁移

仔细规划 IP 池切换
考虑按 APN 渐进式迁移

3. QoS 映射

R99 QoS 配置文件按原样接受
2G/3G 不需要 QCI 转换

相关文档

核心功能

- 配置指南 - 完整的配置参考
- 会话管理 - 会话生命周期详细信息
- **UE IP** 分配 - IP 地址管理
- **PCO** 配置 - 协议配置选项

相关接口

- **S5/S8** 接口 - 4G/LTE 等效接口
- **PFCP** 接口 - 用户平面协调
- **Diameter Gy** 接口 - 在线计费

□作

- 监控指南 - 指标和警报
- 故障排除 - 常见问题
- 数据 **CDR** 格式 - CDR 生成

参考文献

- 3GPP TS 29.060 - GPRS 隧道协议（GTP）通过 Gn 和 Gp 接口
- 3GPP TS 23.060 - 通用分组无线服务（GPRS）；服务描述
- 3GPP TS 24.008 - 移动无线接口第 3 层规范（QoS 配置文件）

返回文档索引

OmniPGW Gn/Gp 接口 - *□ Omnitouch □□□□□□*

IPv6 / 双栈（IPv4v6）PDN 支持

在 OmniPGW（控制平面）和 OmniUPF（用户平面）上操作支持 IPv6 的 PDN 连接的端到端指南。此页面涵盖了 PDN 类型的协商和信令、两个节点上所需的配置，以及如何通过动态路由实现 UE 的 IPv6 可达性。

有关详细的 UE 地址池模型（APN 匹配、分配、冲突处理），请参见 UE IP Pool Allocation。有关 PFCP 和 Gx 接口的一般信息，请参见 PFCP/Sxb 和 Diameter Gx。

目录

- □述
- PDN 类型
- 端到端流程
- PGW-C 配置
- 用户平面（OmniUPF）
- 参考表
- 故障排除

概述

PDN 连接可以是 **IPv4**、**IPv6** 或 **IPv4v6**（双栈）。PDN 类型由 UE/SGW 在 GTPv2 创建会话请求中请求，并由 PGW-C 在响应中确认。实现端到端的 IPv6 PDN 需要在三个接口和两个节点之间协调行为：

节点 / 接口	**IPv6** 责任
OmniPGW — S5/S8	协商 PDN 类型；分配 UE 地址；返回 PDN 地址分配 (PAA)
OmniPGW — Gx	将 UE 地址传达给 PCRF：`Framed-IP-Address`（IPv4）用于 IPv4/IPv4v6 PDN，`Framed-IPv6-Prefix` 用于仅 IPv6 PDN
OmniPGW — Sxb	通过 PFCP 将 UE IPv6 地址和 PDN 类型信号传递给 UPF
OmniUPF — 数据平面	编程 IPv6 下行查找；转发/封装 IPv6 用户流量
OmniUPF — FRR	通过 OSPFv3 将每个 UE IPv6 `/128` 主机路由通告给边缘路由器

参考文献：3GPP TS 29.274 (GTPv2-C)，3GPP TS 29.212 (Gx)，3GPP TS 29.244 (PFCP)。

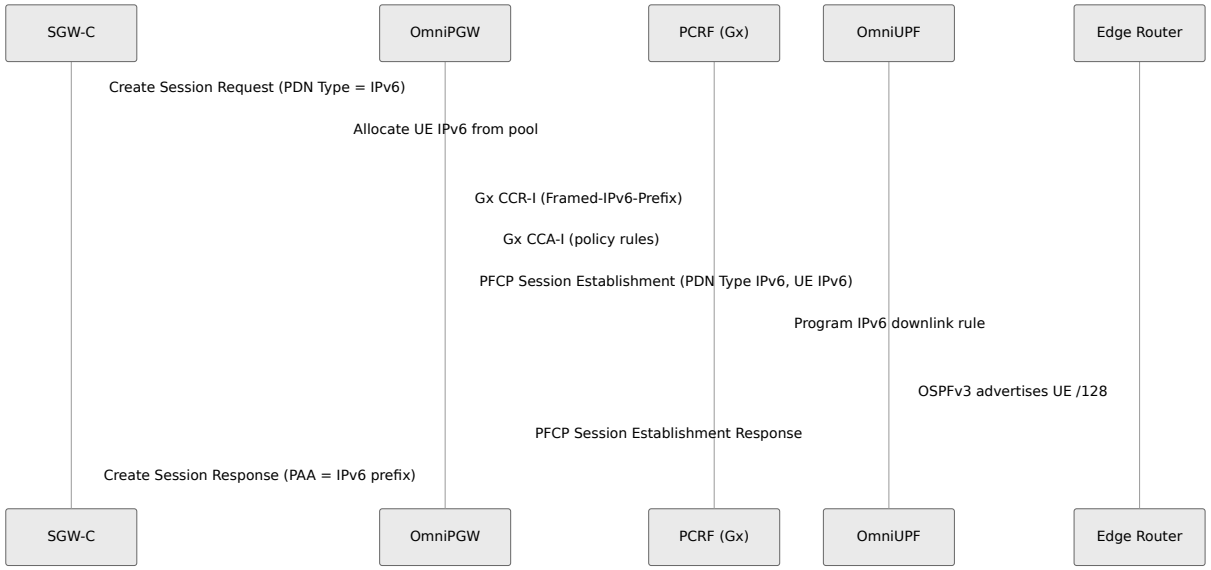
PDN 类型

值	**PDN** 类型	**UE** 接收	**PAA** 内容 **(TS 29.274 §8.14)**
1	IPv4	IPv4 地址	4 字节 IPv4 地址
2	IPv6	IPv6 前缀 (/64)	前缀长度字节 + 16 字节地址
3	IPv4v6	IPv4 地址 + IPv6 前缀	前缀长度字节 + 16 字节 IPv6 + 4 字节 IPv4

PGW-C 从 **PDN** 类型 **IE** 派生有效的 PDN 类型，如果 PDN 类型 IE 缺失，则回退到 **PAA IE** 的类型。相同的协商类型驱动返回给 SGW 的 PAA、发送给 UPF 的 PFCP PDN 类型 IE，以及分配的地址族。

> 没有 **IPv6** 池的 **IPv4v6**。如果请求 IPv4v6 PDN，但匹配的 APN 没有配置 IPv6 范围，PGW-C 不会崩溃；它会将 PDN 降级为仅 IPv4 或拒绝，具体取决于 APN 是否为紧急 (SOS) APN。请参见 APN 没有 IPv6 池时的 IPv4v6 请求。

□□□□□


SGW-C
OmniPGW
PCRF (Gx)
OmniUPF
Edge Router
Create Session Request (PDN Type = IPv6)
Allocate UE IPv6 from pool
Gx CCR-I (Framed-IPv6-Prefix)
Gx CCA-I (policy rules)
PFCP Session Establishment (PDN Type IPv6, UE IPv6)
Program IPv6 downlink rule
OSPFv3 advertises UE /128
PFCP Session Establishment Response
Create Session Response (PAA = IPv6 prefix)
SGW-C
OmniPGW
PCRF (Gx)
OmniUPF
Edge Router

□□□□

参数	类型	必需	默认	描述
`ue.subnet_map`	Map	是	-	将 APN 名称（或 `:default`）映射到 CIDR 字符串列表。`:default` 是当没有 APN 特定条目匹配时的后备。
（列表条目）	字符串（CIDR）	是	-	IPv4（例如 `100.64.16.0/24`）或 IPv6（例如 `2001:db8:6f85:70::/64`）前缀。地址族从字符串中检测。IPv6 UE 地址从配置的前缀中提取，并向 UE 信号发送 `/64`。

> 有关 APN 正则表达式匹配、每个 APN 的多个子网、静态地址和分配算法，请参见 UE IP Pool Allocation。

同时具有 IPv4 和 IPv6 CIDR 的 APN 支持 IPv4、IPv6 和 IPv4v6 PDN；PGW-C 从协商的 PDN ���型所需的任何族中进行分配（对于 IPv4v6，从每个族中分配一个地址）。

Gx (Framed-IPv6-Prefix)

Gx 不需要额外配置。PGW-C 根据地址族从 UE 地址设置 Gx 帧地址 AVP：

- 仅 **IPv6 PDN** — 使用 **`Framed-IPv6-Prefix`** AVP 传达 UE IPv6 前缀（RFC 3162 §2.3），而不是 `Framed-IP-Address`（仅承载 IPv4）。
- **IPv4** 和 **IPv4v6 PDN** — UE IPv4 地址在 `Framed-IP-Address` 中传达。对于 IPv4v6，IPv6 前缀不在 Gx 中承载（CCR 的帧地址 AVP 从 IPv4 地址派生）；IPv6 前缀仍通过 PFCP 信号传递给 UPF，并在 PAA 中传递给 UE。

对于仅 IPv6 的用户，PCRF 必须已配置并能够处理 `Framed-IPv6-Prefix`；否则信用控制请求将被丢弃，会话失败 — 请参见 故障排除。

用户平面 (OmniUPF)

为了让 IPv6 PDN 承载流量，OmniUPF 必须启用 IPv6，从 PGW-C 发送的 PFCP 会话中编程 IPv6 下行查找，并通过 OSPFv3 将每个 UE IPv6 `/128` 主机路由通告给边缘路由器，以便下行流量吸引到 UPF。

这些是配置在 OmniUPF 上的用户平面问题，而不是 PGW-C。有关内核设置、UPF UE 池和 OSPFv3 路由通告的信息，请参见 **OmniUPF IPv6 /** 双栈指南。从 PGW-C 的角度来看，唯一的要求是所选的 UPF 支持 IPv6 — PGW-C 通过 PFCP 信号传递 UE IPv6 地址和 PDN 类型（请参见 PFCP/Sxb）。

参考表

PDN 地址分配 (PAA) — TS 29.274 §8.14

PDN 类型	地址字段布局
IPv4	4 字节: IPv4 地址
IPv6	1 字节前缀长度 + 16 字节 IPv6 前缀/地址
IPv4v6	1 字节前缀长度 + 16 字节 IPv6 + 4 字节 IPv4

Gx 帧地址 AVP

AVP	代码	家族	格式	参考
`Framed-IP-Address`	8	IPv4	4 字节地址	RFC 4005
`Framed-IPv6-Prefix`	97	IPv6	保留字节 + 前缀长度 + 前缀	RFC 3162 §2.3

PFCP PDN 类型 IE — TS 29.244

值	意义
1	IPv4
2	IPv6
3	IPv4v6

故障排除

Gx CCR 超时用于 IPv6 会话

症状：某个用户的 IPv4 会话成功，但 IPv6/IPv4v6 会话返回原因 `remote_peer_not_responding`；Gx 信用控制请求超时。

可能原因：

- PCRF 无法处理 `Framed-IPv6-Prefix` AVP，并静默丢弃请求。
- 用户在 PCRF/HSS 上未配置 IPv6。

解决方案：

1. 确认用户在 PCRF/HSS 上已配置并启用。
2. 验证 PCRF 接受 Gx CCR-I 的 `Framed-IPv6-Prefix`。
3. 与同一用户的 IPv4 会话进行比较，以隔离 IPv6 特定行为。

IPv6 会话被接受但下行流量未到达 UE

症状：创建会话响应携带 IPv6 PAA，上行工作正常，但来自网络到 UE 的流量被丢弃。

这是用户平面（OmniUPF）的问题 — PGW-C 已经分配了地址并通过 PFCP 信号传递。通常原因是 UE `/128` 主机路由未到达边缘路由器（UPF 上未运行 OSPFv3/相邻，或收敛延迟）。有关路由通告配置和故障排除的信息，请参见 OmniUPF IPv6 / 双栈指南。

未分配 IPv6 地址

症状：IPv6 PDN 请求被拒绝，或未返回 IPv6 PAA。

可能原因：

- 匹配的 APN（或 `:default`）在 `ue.subnet_map` 中没有 IPv6 CIDR。
- 配置的 IPv6 池已耗尽。

解决方案：

1. 在 `ue.subnet_map` 中为 APN 的条目添加 IPv6 CIDR（请参见 PGW-C 配置）。

2. 确认池中有可用地址。

APN 没有 IPv6 池时的 IPv4v6 请求

症状：UE 针对在 `ue.subnet_map` 中仅配置了 IPv4 CIDR（没有 IPv6 范围）的 APN 请求 **IPv4v6** PDN。根据 APN 的不同，会话要么以仅 **IPv4** 的形式建立，要么被拒绝��

行为：当匹配的 APN 的 IPv6 列表为空时，IPv6 分配器报告没有 IPv6 池，而不是崩溃会话进程。会话过程然后根据 APN 类型选择（根据 TS 29.274 / TS 23.401 单地址承载 / PDN 类型降级处理）：

APN 类型	结果	**UE / PCRF / UPF** 看到的内容
紧急 **(SOS)** APN	降级为仅 **IPv4** — 分配 IPv4 地址，清除 IPv6 地址，将协商的 PDN 类型设置为 `IPv4`。	PAA、PFCP PDN 类型和 PFCP UE IP 为仅 IPv4；Gx 携带 `Framed-IP-Address`。**P-CSCF** 仍在 **PCO** 中返回，以便紧急 IMS 注册可以继续。记录 `warning`。
常规 APN	拒绝 — 会话不会静默降级；`{:error, :no_ipv6_pool}` 被显示，以便错误配置（或真正不支持的请求）可见。记录 `error`。	创建会话失败，原因非成功。

紧急 APN 通过其名称识别：APN 包含 `"sos"`（例如 `sos`、`sos.mnc410.mcc310.gprs`）。针对没有 IPv6 池的 APN 的仅 IPv6 请求同样被拒绝。

> 实际上，MME 通常会在请求到达 PGW-C 之前将常规用户的 IPv4v6 请求降级为其订阅的 IPv4，因此拒绝路径是一个后备，用于真正没有 IPv6 池的 IPv4v6 请求。SOS 降级路径确保紧急（例如 911/112）呼叫在仅 IPv4 部署中连接。

解决方案：

1. 对于应该提供双栈的常规 APN，在 APN 的条目中添加 IPv6 CIDR（请参见 PGW-C 配置）。
2. 对于仅 IPv4 部署的紧急 (SOS) APN，无需采取任何措施 — 仅 IPv4 的降级是故意的。仅在您希望 SOS PDN 为双栈时添加 IPv6 CIDR。

OmniPGW 监控与指标指南

Prometheus 集成与□作监控

□ Omnitouch □□□□□□

目录

1. □述
2. 指标端点
3. 可用指标
4. Prometheus 配置
5. Grafana 仪表板
6. 警报
7. 性能监控
8. 故障排除指标

概述

OmniPGW 提供两种互补的监控方法：

1. 实时 OAM REST API（在此简要介绍，详细信息请参见各自的接口文档）

- 实时会话查看器（`GET /api/sessions`）
- PFCP 对等状态（`GET /api/upf`）
- Diameter 对等连接性（`GET /api/diameter`）
- 单个会话检查（`GET /api/sessions/<imsi>`）

2. Prometheus 指标（本文件的主要重点）

- 历史趋势和分析
- 警报和通知

性能指标
容量规划

本文档重点关注 Prometheus 指标。有关 OAM REST API（通过 HTTPS 在端口 8443 提供，位于 /api 前缀下），请参见：

会话管理 - 会话 API
PFCP 接口 - 监控 API
Diameter Gx - 对等监控 API

Prometheus 指标概述

OmniPGW 暴露 兼容 Prometheus 的指标，以全面监控系统健康、性能和容量。这使得运营团队能够：

监控系统健康 - 跟踪活动会话、分配和错误
容量规划 - 理解资源利用趋势
性能分析 - 测量消息处理延迟
警报 - 主动通知问题
调试 - 确定问题的根本原因

监控架构

指标端点

配置

在 config/runtime.exs 中启用指标：

```
config :pgw_c,
  metrics: %{
    enabled: true,
    ip_address: "127.0.0.42",        # 默认绑定地址（使用 "0.0.0.0" 在所有接口上公开）
    port: 9090,                       # 默认 HTTP 端口
    registry_poll_period_ms: 10_000  # 轮询间隔（默认 10s）
  }
```

> 注意：随附的默认设置将指标端点绑定到 `127.0.0.42:9090`。要从远程 Prometheus 抓取它，请将 `ip_address` 设置为可访问的地址（例如 `"0.0.0.0"`）。

访问指标

HTTP 端点：

```
http://<omnipgw_ip>:<port>/metrics
```

示例：

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

输出格式

指标以 **Prometheus** 文本格式 公开：

```
# 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
```

可用指标

OmniPGW 暴露以下指标类别:

会话指标

活动会话计数:

指标名称	类型	描述
`teid_registry_count`	Gauge	活动 S5/S8 会话（TEID 计数）
`seid_registry_count`	Gauge	活动 PFCP 会话（SEID 计数）
`gx_session_id_registry_count`	Gauge	活动 Gx 会话（Diameter Session-ID 计数）
`gy_session_id_registry_count`	Gauge	活动 Gy 会话（Diameter Session-ID 计数）
`session_registry_count`	Gauge	活动会话（IMSI, EBI 对）
`address_registry_count`	Gauge	分配的 UE IP 地址
`charging_id_registry_count`	Gauge	活动计费 ID（请参见 数据 CDR 格式 以获取 CDR 计费记录）
`sxb_sequence_number_registry_count`	Gauge	待处理的 PFCP 响应（等待响应）
`s5s8_sequence_number_registry_count`	Gauge	待处理的 S5/S8 响应（等待响应）
`sxb_peer_registry_count`	Gauge	注册的 PFCP 对等进程数量

用法：

```
# 当前活动会话
teid_registry_count

# 会话创建速率（每秒）
rate(teid_registry_count[5m])

# 最近一小时的峰值会话
max_over_time(teid_registry_count[1h])
```

会话和承载生命周期：

计数器在 PDN 会话和承载被创建、修改和拆除时递增。

指标名称	类型	标签	描述
`pgw_session_create_total`	Counter	`result`, `cause`, `pdn_type`, `rat_type`	按结果（成功/失败）、GTP 原因、PDN 类型和 RAT 类型创建的 PDN 会话
`pgw_session_delete_total`	Counter	`pdn_type`	按 PDN 类型删除的 PDN 会话
`pgw_session_modify_total`	Counter	-	成功处理的修改承载请求
`pgw_bearer_create_total`	Counter	`type`, `qci`	按类型（默认/专用）和 QCI 创建的承载
`pgw_bearer_delete_total`	Counter	`type`	按类型（默认/专用）删除的承载

用法：

```
# 会话设置成功率（过去 5 分钟）
sum(rate(pgw_session_create_total{result="success"}[5m])) /
sum(rate(pgw_session_create_total[5m]))

# 按原因统计失败的会话设置
sum(rate(pgw_session_create_total{result="failure"}[5m])) by
(cause)
```

UE IP 地址池

池利用率计量周期性轮询；分配失败会在发生时计数。请参见 UE IP 分配。

指标名称	类型	标签	描述
`ue_pool_addresses_allocated`	Gauge	`pool`, `ip_version`	当前从池中分配的 UE IP 地址
`ue_pool_addresses_total`	Gauge	`pool`, `ip_version`	池中可用的总 UE IP 地址
`ue_pool_addresses_utilization_ratio`	Gauge	`pool`, `ip_version`	当前分配的池的比例（0.0–1.0）
`ue_ip_allocation_failures_total`	Counter	`ip_version`, `reason`	UE IP 分配失败（例如，池耗尽、静态冲突）

用法：

```
# 利用率超过 80% 的池
ue_pool_addresses_utilization_ratio > 0.8

# 按原因统计分配失败率
sum(rate(ue_ip_allocation_failures_total[5m])) by (reason)
```

消息计数器

S5/S8 (GTP-C) 消息：

□□□□	□□	□□	□□
`s5s8_inbound_messages_total`	Counter	`message_type`	□□□ S5/S8 □ □
`s5s8_outbound_messages_total`	Counter	`message_type`	□□□ S5/S8 □ □
`s5s8_inbound_errors_total`	Counter	`message_type`	S5/S8 □□□□□ □
`s5s8_outbound_errors_total`	Counter	`message_type`	S5/S8 □□□□□ □

□□□□□

- `create_session_request`
- `create_session_response`
- `delete_session_request`
- `delete_session_response`
- `create_bearer_request`
- `delete_bearer_request`

Sxb (PFCP) □□□

指标名称	类型	标签	描述
`sxb_inbound_messages_total`	Counter	`message_type`	总入站 PFCP 消息
`sxb_outbound_messages_total`	Counter	`message_type`	总出站 PFCP 消息
`sxb_inbound_errors_total`	Counter	`message_type`	PFCP 入站处理错误
`sxb_outbound_errors_total`	Counter	`message_type`	PFCP 出站处理错误
`sxb_outbound_responses_total`	Counter	`message_type`, `cause_code_class`, `peer_ip`	发送的 PFCP 响应，按 PFCP 原因代码类别和 UPF 对等分类

消息类型：

- `association_setup_request`
- `association_setup_response`
- `heartbeat_request`
- `heartbeat_response`
- `session_establishment_request`
- `session_establishment_response`
- `session_modification_request`

- `session_deletion_request`

Gx (Diameter) 消息:

指标名称	类型	标签	描述
`gx_inbound_messages_total`	Counter	`message_type`	总入站 Diameter 消息
`gx_outbound_messages_total`	Counter	`message_type`	总出站 Diameter 消息
`gx_inbound_errors_total`	Counter	`message_type`	Diameter 入站处理错误
`gx_outbound_errors_total`	Counter	`message_type`	Diameter 出站处理错误
`gx_outbound_responses_total`	Counter	`message_type`, `result_code_class`, `diameter_host`	发送的 Diameter 响应，按结果代码类别和对等主机分类

消息类型:

- `gx_CCA` (Credit-Control-Answer)
- `gx_CCR` (Credit-Control-Request)
- `gx_RAA` (Re-Auth-Answer)
- `gx_RAR` (Re-Auth-Request)

结果代码类别（用于 `gx_outbound_responses_total`）:

- `2xxx` - 成功响应（例如，2001 DIAMETER_SUCCESS）
- `3xxx` - 协议错误（例如，3001 DIAMETER_COMMAND_UNSUPPORTED）

- `4xxx` - 瞬态故障（例如，4001 DIAMETER_AUTHENTICATION_REJECTED）
- `5xxx` - 永久故障（例如，5012 DIAMETER_UNABLE_TO_COMPLY）

用法示例：

```
# 监控 Gx 响应成功率
sum(rate(gx_outbound_responses_total{result_code_class="2xxx"}[5m]))
sum(rate(gx_outbound_responses_total[5m])) * 100

# 按 PCRF 主机跟踪失败
rate(gx_outbound_responses_total{result_code_class!="2xxx"}[5m]) by (

# 统计成功的 Re-Auth-Answer 消息总数
gx_outbound_responses_total{message_type="gx_RAA",result_code_class="

# 针对特定 PCRF 的高失败率发出警报
rate(gx_outbound_responses_total{result_code_class=~"4xxx|5xxx",diame
[5m]) > 0.1
```

Gy (Diameter) 消息：

Gy 在线计费（OCS）消息计数器。请参见 Diameter Gy 以获取接口。

指标名称	类型	标签	描述
`gy_inbound_messages_total`	Counter	`message_type`	总入站 Gy 消息
`gy_outbound_messages_total`	Counter	`message_type`	总出站 Gy 消息
`gy_inbound_errors_total`	Counter	`message_type`	Gy 入站处理错误
`gy_outbound_errors_total`	Counter	`message_type`	Gy 出站处理错误

消息类型：

- `gy_CCA` (Credit-Control-Answer)
- `gy_CCR` (Credit-Control-Request)

Gn/Gp (GTPv1-C) 消息：

Gn/Gp GGSN 控制平面消息计数器（2G/3G SGSN 互通）。请参见 Gn 接口。

指标名称	类型	标签	描述
`gn_inbound_messages_total`	Counter	`message_type`	总入站 Gn/Gp 消息（来自 SGSN）
`gn_outbound_messages_total`	Counter	`message_type`	总出站 Gn/Gp 消息（到 SGSN）
`gn_inbound_errors_total`	Counter	`message_type`	Gn/Gp 入站处理错误
`gn_outbound_errors_total`	Counter	`message_type`	发送到 SGSN 的 Gn/Gp 错误响应

错误处理：

指标名称	类型	标签	描述
`rescues_total`	Counter	`module`, `function`	总救援块命中（异常处理）

延迟指标

入站消息处理持续时间：

指标名称	类型	标签	
`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`	
`gy_inbound_handling_duration`	Histogram	`request_message_type`	

出站事务持续时间：

指标名称	类型	标签
`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`

桶（秒）：

- 值: 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）

用法:

```
# 95th 百分位 S5/S8 延迟
histogram_quantile(0.95,
  rate(s5s8_inbound_handling_duration_bucket[5m])
)

# 平均 PFCP 延迟
rate(sxb_inbound_handling_duration_sum[5m]) /
rate(sxb_inbound_handling_duration_count[5m])
```

UPF 健康监控

UPF 对等指标:

指标名称	类型	标签	描述
`upf_peers_total`	Gauge	-	注册的 UPF 对等总数
`upf_peers_healthy`	Gauge	-	健康的 UPF 对等数量（已关联 + 心跳正常）
`upf_peers_unhealthy`	Gauge	-	不健康的 UPF 对等数量
`upf_peers_associated`	Gauge	-	具有活动 PFCP 关联的 UPF 对等数量
`upf_peers_unassociated`	Gauge	-	没有 PFCP 关联的 UPF 对等数量
`upf_peer_associated`	Gauge	`peer_ip`	特定 UPF 是否具有活动 PFCP 关联（1/0）
`upf_peer_healthy`	Gauge	`peer_ip`	特定 UPF 的健康状态（1=健康，0=不健康）
`upf_peer_missed_heartbeats`	Gauge	`peer_ip`	特定 UPF 连续错过的心跳次数
`upf_heartbeat_rtt`	Histogram	`peer_ip`	PFCP 心跳请求/响应往返时间（秒）

用法：

```
# 监控 UPF 池健康
upf_peers_healthy / upf_peers_total

# 针对不健康的 UPF 发出警报
upf_peers_unhealthy > 0

# 跟踪特定 UPF 健康
upf_peer_healthy{peer_ip="10.98.0.20"}

# 确定心跳问题的 UPF
upf_peer_missed_heartbeats > 2
```

警报示例:

```yaml
# 当 UPF 关闭时发出警报
- alert: UPF_Peer_Down
  expr: upf_peer_healthy == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "UPF {{ $labels.peer_ip }} 已关闭"
    description: "UPF 对等未响应 PFCP 心跳"

# 当多个 UPF 关闭时发出警报
- alert: UPF_Pool_Degraded
  expr: (upf_peers_healthy / upf_peers_total) < 0.5
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: "UPF 池降级"
    description: "只有 {{ $value | humanizePercentage }} 的 UPF 是健康的"

# 针对错过心跳的警告
- alert: UPF_Heartbeat_Issues
  expr: upf_peer_missed_heartbeats > 2
  for: 30s
  labels:
    severity: warning
  annotations:
    summary: "UPF {{ $labels.peer_ip }} 心跳问题"
    description: "{{ $value }} 连续错过心跳"
```

P-CSCF 健康监控

P-CSCF 服务器指标:

指标名称	类型	标签	描述
`pcscf_fqdns_total`	Gauge	-	正在监控的 P-CSCF FQDN 总数
`pcscf_fqdns_resolved`	Gauge	-	通过 DNS 成功解析的 P-CSCF FQDN
`pcscf_fqdns_failed`	Gauge	-	解析失败的 P-CSCF FQDN
`pcscf_servers_total`	Gauge	-	发现的 P-CSCF 服务器总数
`pcscf_servers_healthy`	Gauge	`fqdn`	每个 FQDN 健康的 P-CSCF 服务器
`pcscf_servers_unhealthy`	Gauge	`fqdn`	每个 FQDN 不健康的 P-CSCF 服务器

请参见： P-CSCF 监控指南 以获取详细的 IMS 健康跟踪。

Diameter 对等连接性

定期从 Diameter 堆栈轮询。

指标名称	类型	标签	描述
`diameter_peers_connected`	Gauge	`application`	当前连接的 Diameter 对等数量，按应用程序（`gx`/`gy`/`all`）

用法：

```
# 当没有 Gx (PCRF) 对等连接时发出警报
diameter_peers_connected{application="gx"} == 0
```

许可证指标

许可证状态：

指标名称	类型	描述
`license_status`	Gauge	当前许可证状态（1 = 有效，0 = 无效）

用法：

```
# 检查许可证是否有效
license_status == 1

# 针对无效许可证发出警报
license_status == 0
```

警报示例：

```
- alert: PGW_C_License_Invalid
  expr: license_status == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "PGW-C 许可证无效或过期"
    description: "许可证状态无效 - 创建会话请求被阻止"
```

无效许可证的影响：

当许可证无效或许可证服务器无法访问时，创建会话请求将被拒绝，并带有 GTP-C 原因代码 **“没有可用资源”**（**73**）。这在数据包捕获中可见，如下所示：

Wireshark □□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□”

注意:

- 注册到许可证服务器的产品名称: `omnipgwc`
- 许可证服务器 URL 在 `config/runtime.exs` 中的 `:license_client` 下配置
- 当许可证无效（`license_status == 0`）时，创建会话请求被阻止，GTP-C 原因代码为 73（没有可用资源）
- 无论许可证状态如何，UI 和监控仍然可访问
- Diameter、GTP-C 和 PFCP 对等继续保持连接
- 现有会话不受影响 - 仅阻止新会话的创建

系统指标

Erlang VM 指标:

指标名称	类型	描述
`vm_memory_total`	Gauge	总 VM 内存（字节）
`vm_memory_processes`	Gauge	进程使用的内存
`vm_memory_system`	Gauge	系统使用的内存
`vm_system_process_count`	Gauge	总 Erlang 进程
`vm_system_port_count`	Gauge	总开放端口

Prometheus 配置

抓取配置

将 OmniPGW 添加到 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'
```

多个 **OmniPGW** 实例

```
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'
```

服务发现

Kubernetes:

```
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'
```

验证

测试抓取:

```
# 检查 Prometheus 目标
curl http://prometheus:9090/api/v1/targets

# 查询一个指标
curl 'http://prometheus:9090/api/v1/query?
query=teid_registry_count'
```

Grafana 仪表板

仪表板设置

1. 添加 Prometheus 数据源:

```
配置 → 数据源 → 添加数据源 → Prometheus
URL: http://prometheus:9090
```

2. 导入仪表板:

创建新仪表板或从 JSON 导入。

关键面板

面板 1: 活动会话

```
# 查询
teid_registry_count

# 面板类型: Gauge
# 阈值:
#   绿色: < 5000
#   黄色: 5000-8000
#   红色: > 8000
```

面板 2: 会话速率

```
# 查询
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])

# 面板类型: 图形
# 单位: 请求/秒
```

面板 **3**: **IP** 池利用率

```
# 查询（对于 /24 子网，254 个 IP）
(address_registry_count / 254) * 100

# 面板类型：Gauge
# 单位：百分比（0-100）
# 阈值：
#   绿色：< 70%
#   黄色：70-85%
#   红色：> 85%
```

面板 **4**: 消息延迟（**95th** 百分位）

```
# 查询
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)

# 面板类型：图形
# 单位：毫秒
```

面板 **5**: 错误率

```
# 查询
rate(s5s8_inbound_errors_total[5m])

# 面板类型：图形
# 单位：错误/秒
# 警报阈值：> 0.1
```

面板 **6**: **Gx** 响应成功率

```
# 查询：计算成功 Gx 响应的百分比
sum(rate(gx_outbound_responses_total{result_code_class="2xxx"}
[5m])) /
sum(rate(gx_outbound_responses_total[5m])) * 100

# 面板类型：Gauge
# 单位：百分比（0-100）
# 阈值：
#   绿色：> 95%
#   黄色：90-95%
#   红色：< 90%
```

替代 - 按结果代码类别细分：

```
# 查询：按结果代码类别显示响应计数
sum(rate(gx_outbound_responses_total[5m])) by (result_code_class)

# 面板类型：饼图或条形图
# 图例：{{ result_code_class }}
```

替代 - 按 **PCRF** 响应状态：

```
# 查询：按 PCRF 主机显示响应
sum(rate(gx_outbound_responses_total[5m])) by (diameter_host,
result_code_class)

# 面板类型：堆叠条形图
# 图例：{{ diameter_host }} - {{ result_code_class }}
```

面板 **7**：**UPF** 健康状态

```
# 查询：整体池健康百分比
(upf_peers_healthy / upf_peers_total) * 100

# 面板类型：Gauge
# 单位：百分比（0-100）
# 阈值：
#   绿色：100%
#   黄色：50-99%
#   红色：< 50%
```

替代 - 每个 **UPF** 状态：

```
# 查询：单个 UPF 健康
upf_peer_healthy

# 面板类型：状态
# 映射：
#   1 = "UP"（绿色）
#   0 = "DOWN"（红色）
```

完整仪表板示例

```
{
  "dashboard": {
    "title": "OmniPGW - 操作仪表板",
    "panels": [
      {
        "title": "活动会话",
        "targets": [
          {
            "expr": "teid_registry_count",
            "legendFormat": "活动会话"
          }
        ],
        "type": "graph"
      },
      {
        "title": "会话创建速率",
        "targets": [
          {
            "expr":
"rate(s5s8_inbound_messages_total{message_type=\"create_session_reque
[5m])",
            "legendFormat": "会话/秒"
          }
        ],
        "type": "graph"
      },
      {
        "title": "IP 池利用率",
        "targets": [
          {
            "expr": "(address_registry_count / 254) * 100",
            "legendFormat": "池使用率 %"
          }
        ],
        "type": "gauge"
      },
      {
        "title": "消息延迟（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"
      }
    ]
  }
}
```

□□

□□□□

□□ `omnipgw_alerts.yml`□

```yaml
groups:
  - name: omnipgw
    interval: 30s
    rules:
      # 会话计数警报
      - alert: OmniPGW_HighSessionCount
        expr: teid_registry_count > 8000
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW 高会话计数"
          description: "{{ $value }} 活动会话（阈值：8000）"

      - alert: OmniPGW_SessionCountCritical
        expr: teid_registry_count > 9500
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "OmniPGW 会话计数危急"
          description: "{{ $value }} 活动会话接近容量"

      # IP 池警报
      - alert: OmniPGW_IPPoolUtilizationHigh
        expr: (address_registry_count / 254) * 100 > 80
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW IP 池利用率高"
          description: "IP 池 {{ $value }}% 已利用"

      - alert: OmniPGW_IPPoolExhausted
        expr: address_registry_count >= 254
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "OmniPGW IP 池耗尽"
          description: "没有可用的 IP 进行分配"

      # 错误率警报
```

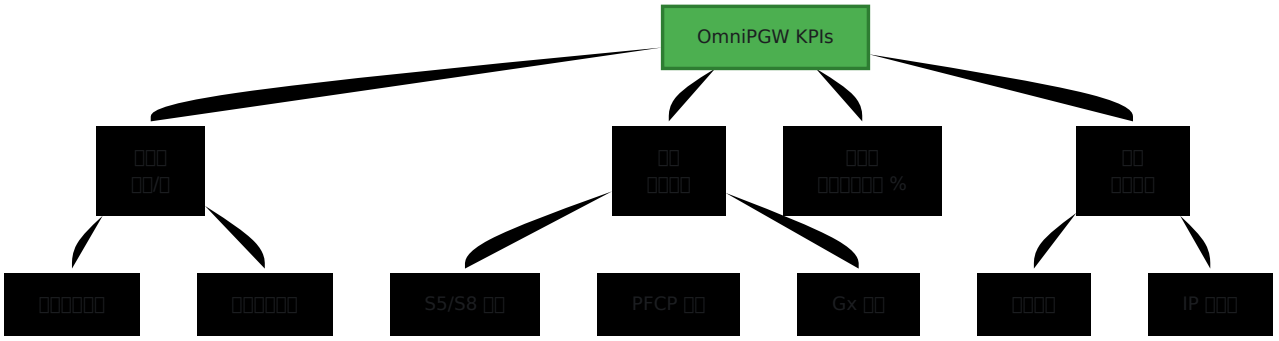
```
      - alert: OmniPGW_HighErrorRate
        expr: rate(s5s8_inbound_errors_total[5m]) > 0.1
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW 高错误率"
          description: "{{ $value }} 错误/秒在 S5/S8 接口上"

      - alert: OmniPGW_GxErrorRate
        expr: rate(gx_inbound_errors_total[5m]) > 0.05
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "OmniPGW Gx 错误"
          description: "{{ $value }} Diameter 错误/秒"

      # Gx 响应警报
      - 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 高 Gx 响应失败率"
          description: "{{ $value | humanizePercentage }} 的 Gx 响应是
失败（非 2xxx 结果代码）"

      - 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:
          summary: "PCRF {{ $labels.diameter_host }} 收到失败响应"
          description: "{{ $value }} 对 PCRF {{
$labels.diameter_host }} 的失败响应/秒"
```

```yaml
      # UPF 健康警报
      - alert: OmniPGW_UPF_PeerDown
        expr: upf_peer_healthy == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "UPF 对等 {{ $labels.peer_ip }} 已关闭"
          description: "UPF 未响应 PFCP 心跳"

      - alert: OmniPGW_UPF_PoolDegraded
        expr: (upf_peers_healthy / upf_peers_total) < 0.5
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "UPF 池降级"
          description: "{{ $value | humanizePercentage }} 的 UPF 是健康的（< 50%）"

      - alert: OmniPGW_UPF_HeartbeatFailures
        expr: upf_peer_missed_heartbeats > 2
        for: 30s
        labels:
          severity: warning
        annotations:
          summary: "UPF {{ $labels.peer_ip }} 心跳失败"
          description: "{{ $value }} 连续错过心跳"

      - alert: OmniPGW_UPF_AllDown
        expr: upf_peers_healthy == 0 and upf_peers_total > 0
        for: 30s
        labels:
          severity: critical
        annotations:
          summary: "所有 UPF 对等已关闭"
          description: "没有健康的 UPF 可用于会话创建"

      # 延迟警报
      - alert: OmniPGW_HighLatency
        expr: |
          histogram_quantile(0.95,
            rate(s5s8_inbound_handling_duration_bucket[5m])
```

```
      ) > 100000
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "OmniPGW 高消息延迟"
      description: "p95 延迟 {{ $value }}μs (> 100ms)"

  # 系统警报
  - alert: OmniPGW_HighMemoryUsage
    expr: vm_memory_total > 2000000000
    for: 10m
    labels:
      severity: warning
    annotations:
      summary: "OmniPGW 高内存使用"
      description: "VM 使用了 {{ $value | humanize }}B 内存"

  - alert: OmniPGW_HighProcessCount
    expr: vm_system_process_count > 100000
    for: 10m
    labels:
      severity: warning
    annotations:
      summary: "OmniPGW 高进程计数"
      description: "{{ $value }} Erlang 进程（潜在泄漏）"
```

AlertManager □□

```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 □□□{{ .GroupLabels.alertname }}'
        text: '{{ range .Alerts }}{{ .Annotations.description }}{{
end }}'

  - name: 'pagerduty'
    pagerduty_configs:
      - service_key: 'YOUR_PAGERDUTY_KEY'
```



性能监控

关键性能指标（KPIs）

吞吐量查询

会话设置速率：

```
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])
```

会话拆除速率：

```
rate(s5s8_inbound_messages_total{message_type="delete_session_request
[5m])
```

净会话增长：

```
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m]) -
rate(s5s8_inbound_messages_total{message_type="delete_session_request
[5m])
```

延迟分析

消息处理延迟（百分位数）：

```
# p50（中位数）
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])
)
```

按消息类型细分的延迟：

```
histogram_quantile(0.95,
  rate(s5s8_inbound_handling_duration_bucket[5m])
) by (request_message_type)
```

容量趋势

会话增长趋势（**24**小时）：

```
teid_registry_count -
teid_registry_count offset 24h
```

剩余容量：

```
# 对于最大容量 10,000 会话
10000 - teid_registry_count
```

容量耗尽的时间：

```
# 基于 1 小时增长率的天数
(10000 - teid_registry_count) /
(rate(teid_registry_count[1h]) * 86400)
```

故障排除指标

识别问题

问题：高会话拒绝率

查询：

```
rate(s5s8_inbound_errors_total[5m]) by (message_type)
```

行动：

- 检查错误日志
- 验证 PCRF 连接性（Gx 错误）
- 检查 IP 池耗尽

问题：会话设置缓慢

查询：

```
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)
```

行动：

- 检查 Gx 延迟（PCRF 响应时间）
- 检查 PFCP 延迟（PGW-U 响应时间）

- 查看系统资源使用情况

问题：**PCRF** 策略失败

查询：

```
# 整体 Gx 响应失败率
sum(rate(gx_outbound_responses_total{result_code_class!="2xxx"}
[5m])) /
sum(rate(gx_outbound_responses_total[5m])) * 100

# 按 PCRF 主机细分
sum(rate(gx_outbound_responses_total[5m])) by (diameter_host,
result_code_class)

# 特定结果代码类别
rate(gx_outbound_responses_total{result_code_class="5xxx"}[5m]) by
(diameter_host)
```

行动：

- 检查 PCRF 连接性和健康状况
- 查看 PCRF 中的订阅者配置文件（5xxx 错误通常表示策略问题）
- 验证 Diameter 对等配置
- 检查 PCRF 日志以获取相应错误
- 对于 5012（DIAMETER_UNABLE_TO_COMPLY），查看 Re-Auth-Request 处理

问题：怀疑内存泄漏

查询：

```
# 总内存趋势
rate(vm_memory_total[1h])

# 进程内存趋势
rate(vm_memory_processes[1h])

# 进程计数趋势
rate(vm_system_process_count[1h])
```

行动：

- 检查过期会话
- 查看注册计数
- 如果确认泄漏，则重启

调试查询

查找峰值会话时间：

```
max_over_time(teid_registry_count[24h])
```

比较当前与历史：

```
teid_registry_count /
avg_over_time(teid_registry_count[7d])
```

识别异常：

```
abs(
  teid_registry_count -
  avg_over_time(teid_registry_count[1h])
) > 100
```

最佳实践

指标收集

1. 抓取间隔： 15-30 秒（平衡粒度与负载）
2. 保留： 15 天以上以进行历史分析
3. 标签：使用一致的标签（实例、环境、站点）

仪表板设计

1. 概览仪表板 - NOC 的高层 KPI
2. 详细仪表板 - 每个接口的深入分析
3. 故障排除仪表板 - 错误指标和日志

警报设计

1. 避免警报疲劳 - 仅对可操作的问题发出警报
2. 升级 - 警告 → 危急，随着严重性升级
3. 上下文 - 在警报描述中包含运行手册链接

相关文档

配置和设置

- 配置指南 - Prometheus 指标配置，OAM REST API 设置
- 故障排除指南 - 使用指标进行调试

接口指标

- **PFCP** 接口 - PFCP 会话指标，UPF 健康监控
- **Diameter Gx** 接口 - Gx 策略指标，PCRF 交互跟踪
- **Diameter Gy** 接口 - Gy 计费指标，配额跟踪，OCS 超时
- **S5/S8** 接口 - GTP-C 消息指标，SGW-C 通信

专门监控

- **P-CSCF** 监控 - P-CSCF 发现指标，IMS 健康
- 会话管理 - 活动会话，会话生命周期指标
- **UE IP** 分配 - IP 池利用率指标

返回□作指南

OmniPGW 监控指南 - *□ Omnitouch □□□□□□*

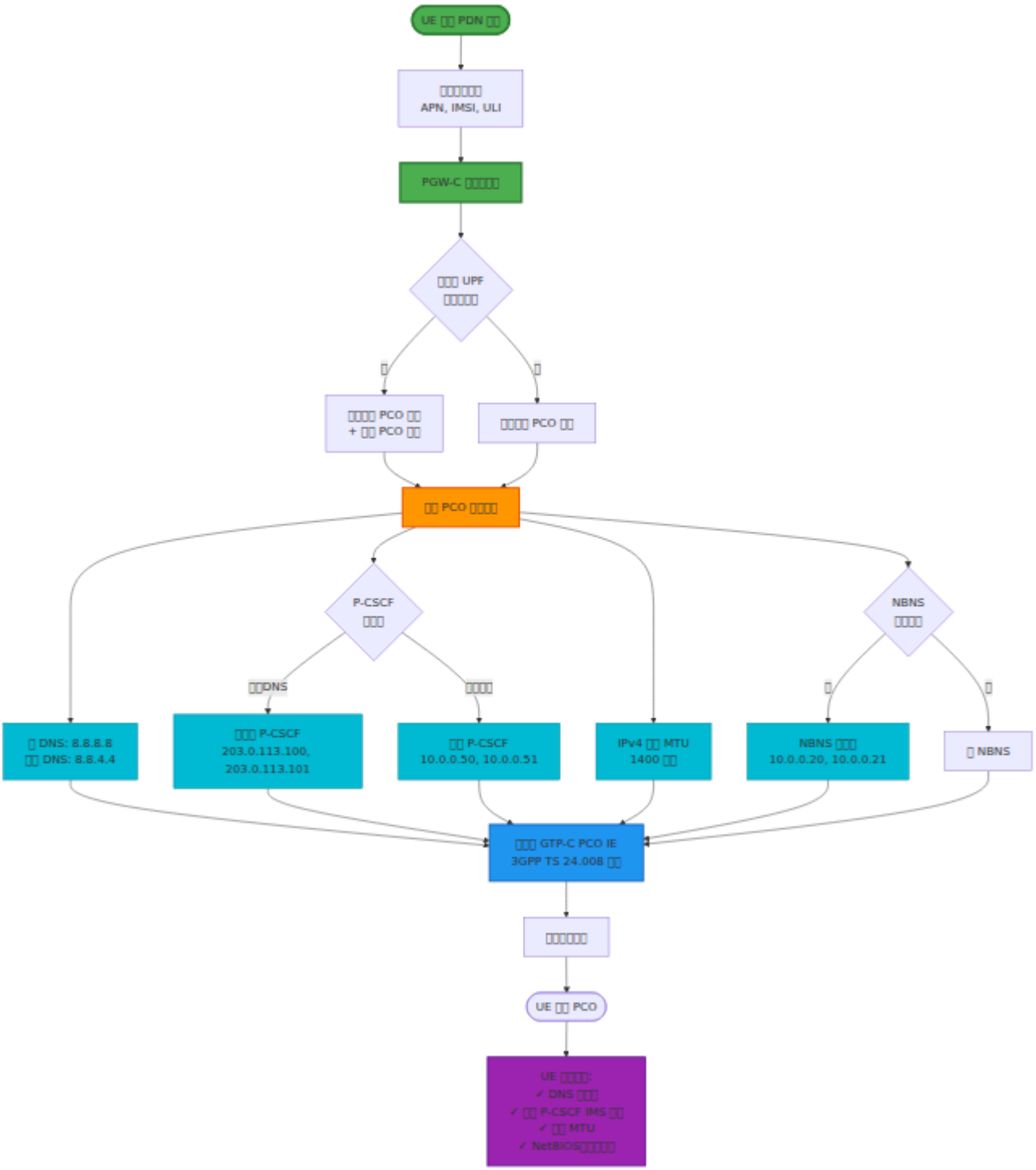
协议配置选项（PCO）

传递给 **UE** 的网络参数

OmniPGW □ Omnitouch □□□□□□

概述

PCO（协议配置选项**）**是在 PDN 连接建立期间发送给 UE（移动设备）的网络参数。这些参数使 UE 能够访问网络服务，如 DNS、IMS，并配置网络设置。



OmniPGW 发出的 **PCO** 容器 **(**网络 **→ MS)**:

IE 名称	容器 **ID**	描述	必需
DNS 服务器 IPv4 地址	0x000D	主 DNS	是
DNS 服务器 IPv4 地址	0x000D	备用 DNS	可选
DNS 服务器 IPv6 地址	0x0003	主/备用 DNS (IPv6)	可选
P-CSCF IPv4 地址	0x000C	IMS 的 P-CSCF (每个地址一个容器)	可选 (IMS)
P-CSCF IPv6 地址	0x0001	IMS 的 P-CSCF (每个地址一个容器)	可选 (IMS)
IPv4 链路 MTU	0x0010	最大传输单元	推荐

> 通过 **IPCP** 的 **DNS/NBNS**。当 UE 使用 **IPCP** (互联网协议控制协议，容器 `0x8021`) 配置请求而不是裸 DNS 容器请求 DNS (和 NBNS/WINS) 服务器时，OmniPGW 在 IPCP 配置拒绝 中回应，携带配置的地址作为 IPCP 选项 `ms_dns1`/`ms_dns2` 和 `ms_wins1`/`ms_wins2` (RFC 1877)。因此，NBNS/WINS 仅通过此 IPCP 路径传递——没有单独的 NBNS PCO 容器 (`0x0011`) 被发出。使用配置拒绝 (而不是确认) 是因为 PGW 提供的地址与 UE 的 `0.0.0.0` 占位符不同，并且回复回显请求的 IPCP 标识符 (RFC 1661 §5)。

> **IPv4** 和 **IPv6 P-CSCF / DNS**; 无 **E-CSCF**。OmniPGW 同时响应 IPv4 和 IPv6 P-CSCF 地址请求，将每个发现或配置的地址��由到匹配其族的容器: IPv4 地址放入 `0x000C`, IPv6 地址放入 `0x0001`。IPv6 P-CSCF 请求 (`p_cscf_ipv6_address_request`) 从 `p_cscf_ipv6_address_list` (或 DNS/AAAA 发现返回的 IPv6 地址) 提供，而 IPv6 DNS 请求 (`dns_server_ipv6_address_request`) 在容器 `0x0003` 中从 `primary_dns_server_ipv6_address` / `secondary_dns_server_ipv6_address` 提供。地址族在容器之间永远不会混合。没有 E-CSCF 容器处理。以上所有内容同样适用于 APCO (附加 PCO)。

配置

基本配置

```
# config/runtime.exs
config :pgw_c,
  pco: %{
    # DNS 服务器 (必需)
    primary_dns_server_address: "8.8.8.8",
    secondary_dns_server_address: "8.8.4.4",

    # IPv6 DNS 服务器 (可选, 容器 0x0003)
    primary_dns_server_ipv6_address: nil,
    secondary_dns_server_ipv6_address: nil,

    # NBNS 服务器 (可选, 适用于 Windows 设备)
    primary_nbns_server_address: nil,
    secondary_nbns_server_address: nil,

    # IMS/VoLTE 的 P-CSCF 地址 (可选)
    p_cscf_ipv4_address_list: [],   # 容器 0x000C
    p_cscf_ipv6_address_list: [],   # 容器 0x0001

    # P-CSCF 动态发现 (可选; 解析 A 和 AAAA 记录)
    p_cscf_discovery_enabled: false,
    p_cscf_discovery_dns_server: nil,
    p_cscf_discovery_timeout_ms: 5000,

    # IPv4 MTU 大小 (字节)
    ipv4_link_mtu_size: 1400
  }
```

PCO 参数

DNS 服务器地址

主 **DNS** 和备用 **DNS**:

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

常见 **DNS** 提供商:

提供商	主	备用
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

私有 **DNS**:

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

P-CSCF 地址 (IMS)

用于 **IMS/VoLTE** 服务:

```
pco: %{
  p_cscf_ipv4_address_list: [
    "10.0.0.50",  # 主 P-CSCF (IPv4, 容器 0x000C)
    "10.0.0.51"   # 备用 P-CSCF
  ],
  # IPv6 P-CSCF 地址 (容器 0x0001) 适用于 IPv6/IPv4v6 IMS PDN
  p_cscf_ipv6_address_list: [
    "2001:db8::50",
    "2001:db8::51"
  ]
}
```

每个地址仅在匹配其族的容器中发出，并且仅在 UE 请求该族时——IPv4 P-CSCF 请求接收 `0x000C` 容器，IPv6 P-CSCF 请求接收 `0x0001` 容器。双栈 IMS PDN 通常会请求两者。

P-CSCF (代理呼叫会话控制功能**)**:

- IMS 信令的入口点
- VoLTE、VoWiFi、RCS 所���
- UE 通过此服务器使用 SIP

P-CSCF 动态发现

基于 **DNS** 的 **P-CSCF** 发现:

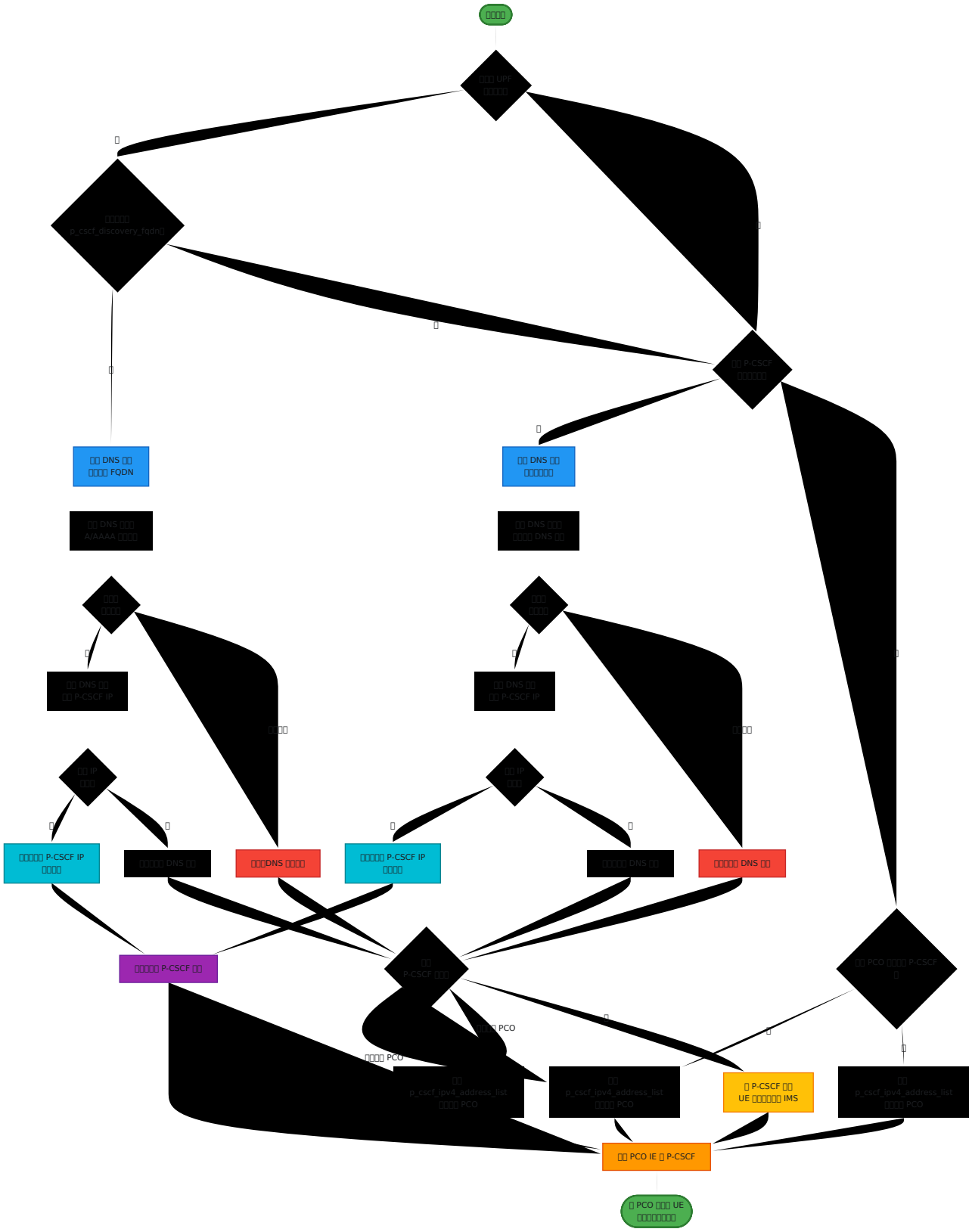
OmniPGW 支持通过 DNS 查询进行动态 P-CSCF 发现，如 3GPP TS 23.003 和 TS 24.229 中定义。当启用时，PGW-C 可以查询 DNS 以获取 P-CSCF 地址，而不是使用静态配置。

```
pco: %{
  # □用动态 P-CSCF 发现
  p_cscf_discovery_enabled: true,

  # P-CSCF 查询的 DNS 服务器 (作为元组)
  p_cscf_discovery_dns_server: {10, 179, 2, 177},

  # DNS 查询的超时 (毫秒)
  p_cscf_discovery_timeout_ms: 5000,

  # 静态 P-CSCF 列表 (如果 DNS 失败则使用)
  p_cscf_ipv4_address_list: ["10.0.0.50"]
}
```

工作原理:

1. 当 `p_cscf_discovery_enabled: true` 时，PGW-C 执行 DNS 查询以获取 P-CSCF 地址
2. DNS 查询发送到配置的 `p_cscf_discovery_dns_server`
3. 如果 DNS 查询成功，发现的 P-CSCF 地址通过 PCO 发送给 UE
4. 如果 DNS 查询失败或超时，则回退到静态 `p_cscf_ipv4_address_list`
5. 请参见 P-CSCF 监控 以获取详细的监控和指标

P-CSCF □□□□



□□□□□□□

1. □□□ **FQDN** □□ (□□□□□□) - `p_cscf_discovery_fqdn` □ UPF □□□□□
2. □□ **DNS** □□ - `p_cscf_discovery_enabled: true` □□□ PCO □□□

3. 规则 **PCO** 静态列表 - `p_cscf_ipv4_address_list` 在规则 PCO 覆盖中
4. 全局 **PCO** 静态列表（回退）- `p_cscf_ipv4_address_list` 在全局 PCO 配置中

监控：

所有 P-CSCF 发现尝试都被记录并跟踪指标：

- DNS 查询成功/失败率
- 发现延迟
- 回退使用统计
- 每规则和全局发现指标

请参见 P-CSCF 监控 以获取完整的监控详细信息。

配置选项：

参数	类型	默认值	描述
`p_cscf_discovery_enabled`	布尔值	`false`	启用基于 DNS 的动态 P-CSCF 发现
`p_cscf_discovery_dns_server`	元组 (IP)	`nil`	DNS 服务器 IP 地址作为 4 元组（例如，`{10, 179, 2, 177}`）
`p_cscf_discovery_timeout_ms`	整数	`5000`	DNS 查询的超时（毫秒）

用例：

- 动态 **IMS** 部署 - P-CSCF 地址根据 DNS 配置变化
- 地理负载均衡 - DNS 返回最近的 P-CSCF 服务器
- 高可用性 - DNS 自动返回可用的 P-CSCF 服务器
- 多租户环境 - 不同的订阅者获得不同的 P-CSCF 服务器

示例：使用 **DNS** 发现的生产 **IMS**

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

  # 启用动态 P-CSCF 发现
  p_cscf_discovery_enabled: true,
  p_cscf_discovery_dns_server: {10, 179, 2, 177},  # IMS DNS 服务器
  p_cscf_discovery_timeout_ms: 3000,

  # 回退 P-CSCF 地址（如果 DNS 失败）
  p_cscf_ipv4_address_list: [
    "10.0.0.50",  # 主回退
    "10.0.0.51"   # 备用回退
  ],

  ipv4_link_mtu_size: 1400
}
```

每规则 **P-CSCF** 发现：

P-CSCF 发现也可以根据 UPF 选择规则进行配置。这允许不同的 APN 使用不同的 DNS 服务器进行 P-CSCF 发现：

```
# 在 upf_selection 配置中
rules: [
  %{
    name: "IMS 流量",
    priority: 20,
    match_field: :apn,
    match_regex: "^ims",
    upf_pool: [...],

    # 每规则 P-CSCF 发现
    p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org"
  }
]
```

请参见 UPF 选择配置 以获取每规则 P-CSCF 发现的详细信息。

另见： P-CSCF 监控 以监控 P-CSCF 发现和健康状况

NBNS 服务器 (NetBIOS)

用于 **Windows** 设备兼容性:

```
pco: %{
  primary_nbns_server_address: "10.0.0.20",
  secondary_nbns_server_address: "10.0.0.21"
}
```

何时使用:

- 企业网络中有 Windows 设备
- 需要支持旧版应用程序
- 需要 NetBIOS 名称解析

链路 MTU 大小

最大传输单元:

```
pco: %{
  ipv4_link_mtu_size: 1400  # 字节
}
```

常见 **MTU** 值:

MTU	用例
1500	标准以太网 (无隧道)
1400	考虑 GTP 隧道开销
1420	减少开销
1280	IPv6 最小 MTU
1360	VPN/隧道环境

建议：对于 LTE 使用 **1400** 以考虑 GTP-U 开销。

配置示例

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
}
```

见：P-CSCF 监控 以监控 IMS 注册成功率和 P-CSCF 健康状况

企业 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 在 GTP-C 消息中的应用

创建会话响应

OmniPGW 在 创建会话响应 消息中包含 PCO:

```
创建会话响应
├── 原因: 请求已接受
├── UE IP 地址: 100.64.1.42
├── PCO (协议配置选项)
│   ├── DNS 服务器 IPv4 地址: 8.8.8.8
│   ├── DNS 服务器 IPv4 地址: 8.8.4.4
│   ├── P-CSCF IPv4 地址: 10.0.0.50
│   ├── P-CSCF IPv4 地址: 10.0.0.51
│   └── IPv4 链路 MTU: 1400
```

UE 处理

UE 接收 PCO 并:

1. 使用提供的服务器配置 DNS 解析器
2. 向 P-CSCF 注册 IMS 服务
3. 将接口 MTU 设置为指定值

故障排除

问题：UE 无法解析 DNS

症状：

- UE 有 IP 地址但无法访问互联网
- DNS 查找失败

可能原因：

1. PCO 配置中的 DNS 服务器地址不正确
2. 从 UE IP 池无法访问 DNS 服务器
3. 防火墙阻止 DNS 流量

解决方案：

```
# 测试 DNS 服务器可达性
ping 8.8.8.8

# 测试 UE 网络中的 DNS 解析
nslookup google.com 8.8.8.8

# 验证 PCO 配置
grep "primary_dns_server_address" config/runtime.exs
```

问题：IMS 注册失败

症状：

- VoLTE 呼叫失败
- UE 显示 "无 IMS 注册"

可能原因：

1. 缺少 P-CSCF 配置
2. P-CSCF IP 地址不正确
3. P-CSCF 无法访问

解决方案：

```
# 验证 P-CSCF 配置
pco: %{
  p_cscf_ipv4_address_list: ["10.0.0.50"]  # 确保不为空
}
```

问题：MTU 问题

症状：

- 一些网站加载，其他网站无法加载
- 大文件传输失败
- 分片问题

可能原因：

- MTU 对于隧道开销过大
- MTU 过小导致过度分片

解决方案：

```
# 推荐：对于 GTP 隧道使用 1400
pco: %{
  ipv4_link_mtu_size: 1400
}

# 如果仍然存在问题，尝试更低的值
pco: %{
  ipv4_link_mtu_size: 1360
}
```

最佳实践

DNS 配置

1. 使用可靠的 **DNS** 服务器
    - 公共: Google (8.8.8.8), Cloudflare (1.1.1.1)
    - 私有: 企业内部 DNS
2. 始终配置备用
    - 提供冗余
    - 提高可靠性
3. 考虑 **DNS** 安全性
    - 支持 DNSSEC 的解析器
    - 进行安全的 DNS 过滤

IMS 配置

1. 提供多个 **P-CSCF**
    - 至少 2 个以实现冗余
    - 如果可能，进行地理分布
2. 确保可达性
    - P-CSCF 必须可以从 UE IP 池访问
    - 测试 SIP 连接性

MTU 优化

1. 考虑开销
    - GTP-U: 36 字节 (IPv4)
    - IPsec: 可变 (50-100 字节)

2. 标准 **LTE MTU**
    - 推荐: **1400** 字节
    - 平衡吞吐量和兼容性
3. 进行端到端测试
    - 路径 MTU 发现
    - 使用大数据包进行测试

相关文档

配置指南

- 配置指南 - 完整的 runtime.exs 参考，UPF 选择与 PCO 覆盖
- **UE IP** 分配 - IP 池管理，基于 APN 的分配
- **P-CSCF** 监控 - P-CSCF 发现监控，健康跟踪，指标

会话和接口管理

- 会话管理 - PDN 会话生命周期，承载建立
- **S5/S8** 接口 - GTP-C 协议，PCO 编码和传递
- **PFCP** 接口 - 用户平面会话建立

IMS 和 VoLTE

- **Diameter Gx** 接口 - IMS 承载的策略控制
- 监控指南 - 与 PCO 相关的指标和仪表板

返回□作指南

OmniPGW PCO 配置 - *□ Omnitouch □□□□□□*

P-CSCF 发现与监控

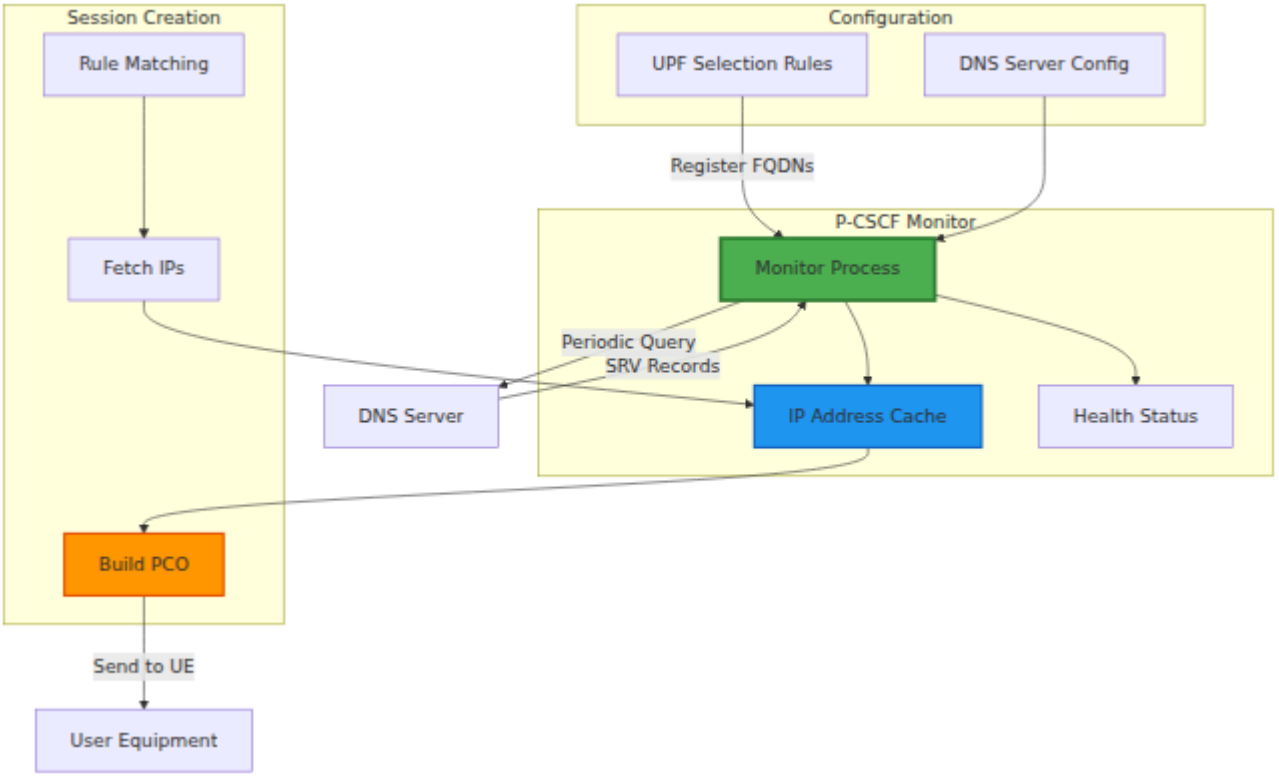
动态 **P-CSCF** 服务器发现与实时监控

OmniPGW □ Omnitouch □□□□□□

概述

P-CSCF（代理呼叫会话控制功能**）**发现与监控 通过 DNS SRV 查询和实时 SIP OPTIONS 健康检查提供 IMS P-CSCF 服务器的动态发现。此功能使得：

- 按规则 **P-CSCF** 发现：不同流量类型使用不同的 P-CSCF 服务器
- 自动监控：后台进程持续监控 DNS 解析（每 60 秒一次）
- **SIP OPTIONS** 健康检查：通过 SIP OPTIONS ping 验证 P-CSCF 服务器是否存活
  - 优先使用 **TCP**：优先通过 TCP 尝试 SIP OPTIONS（可靠性更高）
  - **UDP** 备用：如果 TCP 失败则回退到 UDP
  - 状态跟踪：根据响应将每个服务器标记为 :up 或 :down
- 实时健康跟踪：OAM API 暴露解析状态、发现的 IP 和健康状态
- 优雅回退：三层回退策略以确保最大可靠性
- **Prometheus** 指标：通过 Prometheus 指标实现全面可观察性



目录

1. 快速开始
2. 配置
3. 工作原理
4. OAM API 监控
5. 指标与可观察性
6. 回退策略
7. DNS 配置
8. 故障排除
9. 最佳实践

快速开始

基本配置

```
# config/runtime.exs

# 全局 PCO 配置（用于 P-CSCF 发现的 DNS 服务器）
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 流量 - 动态 P-CSCF 发现
      %{
        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 发现 FQDN（有关更多 UPF 选择规则，请参见配置指南）
        p_cscf_discovery_fqdn:
"pcscf.mnc001.mcc001.3gppnetwork.org",
        # 静态回退（请参见 PCO 配置指南）
        pco: %{
          p_cscf_ipv4_address_list: ["203.0.113.100",
"203.0.113.101"]
        }
      }
    ]
  }
```

请参见 配置指南 以获取完整的 UPF 选择规则配置，以及 PCO 配置 以获取静态 P-CSCF 回退选项。

访问监控

1. 启动 OmniPGW
2. 查询 OAM API: `curl -k https://localhost:8443/api/pcscf_monitor`
3. 检查返回的解析状态和发现的 IP（重新运行请求以获取当前状态）

配置

全局 **P-CSCF** 发现设置

在 PCO 部分配置用于 P-CSCF 发现的 DNS 服务器:

```
pco: %{
  # 用于 P-CSCF 发现的 DNS 服务器（与提供给 UE 的 DNS 分开）
  p_cscf_discovery_dns_server: "192.0.2.177",

  # 启用 P-CSCF DNS 发现功能
  p_cscf_discovery_enabled: true,

  # DNS SRV 查询超时（毫秒）
  p_cscf_discovery_timeout_ms: 5000,

  # 静态 P-CSCF 地址（全局回退）
  p_cscf_ipv4_address_list: ["203.0.113.146"]
}
```

按规则 **P-CSCF FQDN**

每个 UPF 选择规则可以指定其自己的 P-CSCF 发现 FQDN:

```
upf_selection: %{
  rules: [
    # IMS 流量 - IMS 特定 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"]  # 回退
      }
    },

    # 企业 - 企业特定 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"]  # 回退
      }
    },

    # 互联网 - 无 P-CSCF 发现（使用全局配置）
    %{
      name: "Internet Traffic",
      match_field: :apn,
      match_regex: "^internet",
      upf_pool: [...]
      # 无 p_cscf_discovery_fqdn - 使用全局 PCO 配置
    }
  ]
}
```

工作原理

启动过程

1. 应用程序启动

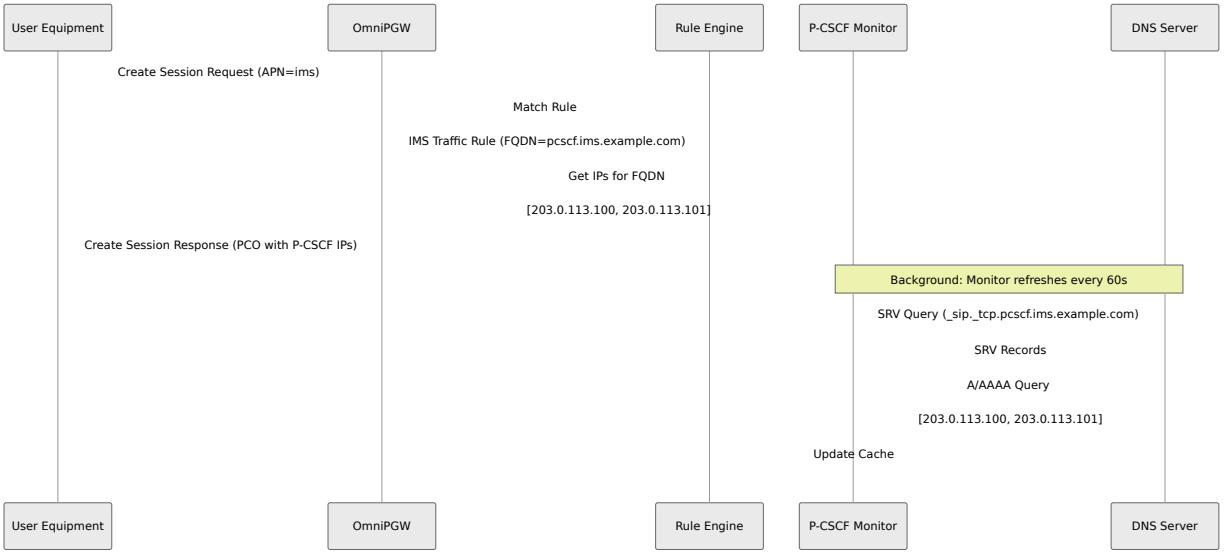
P-CSCF 监控 GenServer 初始化
配置解析器从 UPF 选择规则中提取所有唯一的 P-CSCF FQDN

2. FQDN 注册

每个唯一的 FQDN 都会注册到监控器
监控器对每个 FQDN 执行初始 DNS SRV 查询
SIP OPTIONS 健康检查（并行处理所有发现的服务器）：
首先尝试 TCP（SIP/2.0/TCP 在 5060 端口）
如果 TCP 失败，则回退到 UDP（SIP/2.0/UDP 在 5060 端口）将每个服务器标记为:up（响应）或:down（无响应/超时）
结果（IP、健康状态或错误）与时间戳一起缓存

3. 定期监控（每 60 秒）

监控器刷新所有 FQDN
DNS 查询在后台运行，不会阻塞
对于每个发现的服务器：
通过 TCP 发送 SIP OPTIONS（超时：5 秒）
如果 TCP 失败，则尝试 UDP（超时：5 秒）
根据响应更新健康状态
缓存更新为最新的 DNS 结果和健康状态



会话创建流程

DNS 查询过程

监控器使用 **DNS SRV** 记录进行直接 P-CSCF 发现：

1. **SRV** 查询：查询 `_sip._tcp.{fqdn}` 的 SRV 记录
2. 优先级排序：按优先级和权重排序
3. 目标提取：从 SRV 记录中提取目标主机名
4. 主机名解析：将目标主机名解析为 IP 地址（A/AAAA 记录）
5. 缓存：缓存解析的 IP 及其状态和时间戳

P-CSCF 地址选择优先级

当规则上同时配置了 **FQDN** 和静态 **PCO** 时，**FQDN** 优先：

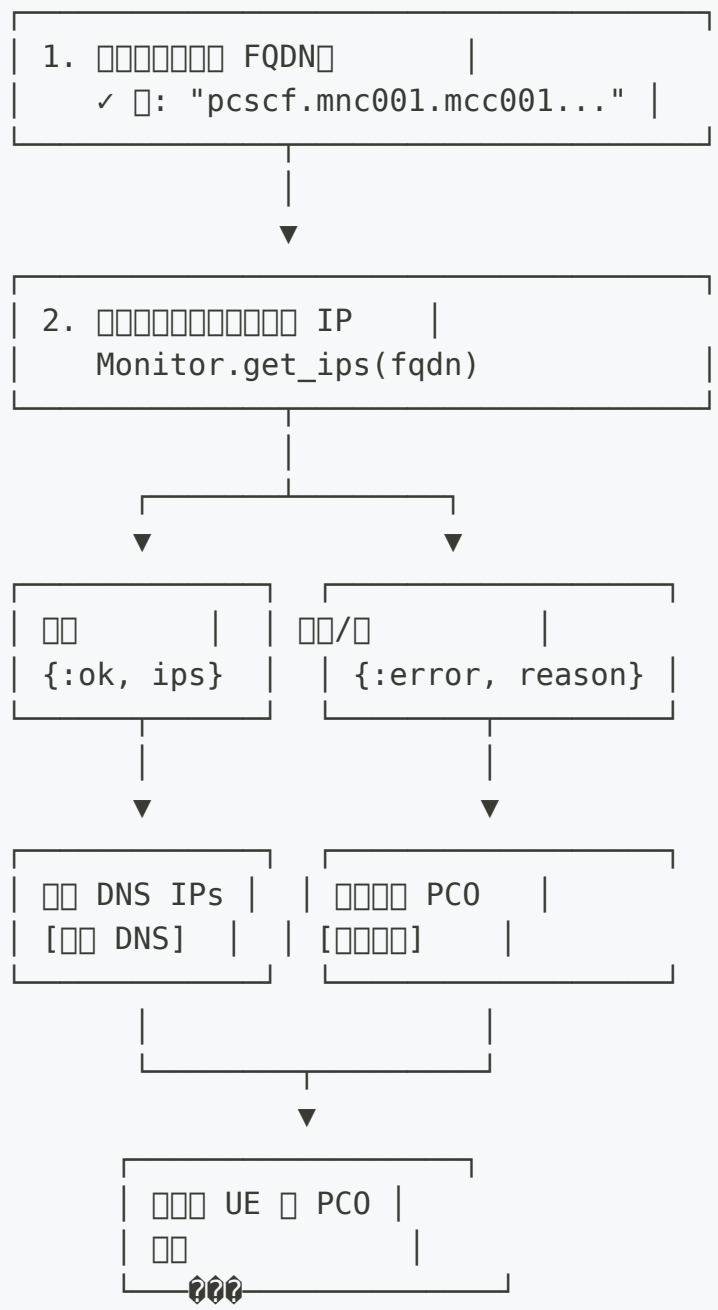
```
%{
  name: "IMS Traffic",
  p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org",  #
← 首先尝试
  pco: %{
    p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"]
# ← 回退
  }
}
```

选择逻辑：

条件	**P-CSCF** 来源	使用的 **IP**	日志消息
FQDN 成功解析	DNS 发现（监控）	从 DNS 发现的 IP	`"使用来自 FQDN pcscf.example.com 的 P-CSCF 地址"`
FQDN 解析失败	规则 PCO 覆盖	来自 `pco.p_cscf_ipv4_address_list` 的静态 IP	`"未能从 FQDN 获取 P-CSCF IP，回退到静态配置"`
FQDN 返回空列表	规则 PCO 覆盖	来自 `pco.p_cscf_ipv4_address_list` 的静态 IP	触发回退
监控不可用	规则 PCO 覆盖	来自 `pco.p_cscf_ipv4_address_list` 的静态 IP	错误触发回退
未配置 **FQDN**	规则 PCO 覆盖或全局	来自规则或全局配置的静态 IP	直接使用静态配置

示例流程：

IMS □□□□□□□□□□□


1. □□□□□□□ FQDN□
✓ □: "pcscf.mnc001.mcc001..."
2. □□□□□□□□□□□ IP
Monitor.get_ips(fqdn)
□□
{:ok, ips}
□□/□
{:error, reason}
□□ DNS IPs
[□□ DNS]
□□□□ PCO
[□□□□]
□□□ UE □ PCO
□□

□□□□□

□□ 1□DNS □□□□□□ □

```
配置:
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  pco.p_cscf_ipv4_address_list: ["203.0.113.100"]

DNS 结果: [203.0.113.150, 203.0.113.151]
UE 接收: [203.0.113.150, 203.0.113.151] ← 来自 DNS
注意: 当 DNS 成功时，静态 PCO 被忽略
```

场景 **2**: **DNS** 失败，优雅回退 ⚠

```
配置:
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  pco.p_cscf_ipv4_address_list: ["203.0.113.100"]

DNS 结果: 错误 :no_naptr_records
UE 接收: [203.0.113.100] ← 来自静态 PCO
注意: 尽管 DNS 失败，会话仍然成功
```

场景 **3**: 未配置 **FQDN**

```
配置:
  # 未配置 p_cscf_discovery_fqdn
  pco.p_cscf_ipv4_address_list: ["192.168.1.50"]

UE 接收: [192.168.1.50] ← 来自静态 PCO
注意: 未尝试 DNS 发现
```

为什么采用这种设计?

1. 优先动态: DNS 提供灵活性、负载均衡和位置感知路由
2. 确保可靠性: 静态回退确保会话不会因 DNS 问题而失败
3. 零手动干预: 自动故障转移，无需操作员参与
4. 生产安全: 兼具敏捷性和稳定性

建议: 在生产部署中始终配置 FQDN 和静态 PCO:

```
# ✓ 推荐：动态回退
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com",  # 优先
  pco: %{
    p_cscf_ipv4_address_list: ["203.0.113.100"]     # 安全网
  }
}

# ⚠ 风险：仅动态（回退到全局 PCO）
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  # 无规则特定回退！
}

# ✓ 有效：仅静态（无 DNS 开销）
%{
  pco: %{
    p_cscf_ipv4_address_list: ["192.168.1.50"]
  }
}
```

OAM API 监控

P-CSCF 监控端点

监控通过 OAM REST API（HTTPS/TLS 在端口 `8443` 上）公开。所有路由都在 `/api` 前缀下提供。查询 P-CSCF 发现监控器：

```
GET /api/pcscf_monitor
```

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

该端点不接受路径或查询参数，并在每次调用时返回当前发现状态。交互式 API 文档（Swagger UI）可在 `https://localhost:8443/api/docs` 访问。

响应内容：

- 概述统计
  - 监控的总 FQDN 数量
  - 成功解析的 FQDN 数量
  - 失败的解析数量
  - 发现的 P-CSCF IP 总数
- 每个 **FQDN** 的详细信息
  - 正在监控的 FQDN
  - 解析状态（已解析 / 失败 / 待定）
  - 发现的 IP 数量
  - 解析的 IP 地址列表（带每个服务器的详细信息）
  - 最后更新时间戳
  - 健康状态：每个发现的服务��报告：
    - IP 地址和端口
    - 主机名（来自 DNS SRV 目标）
    - 健康指示器（正常 / 异常）
- 监控指标
  - 总 **FQDN** 数量：注册用于监控的唯一 FQDN 数量
  - 成功解析：通过 DNS 成功解析的 FQDN
  - 失败的 **DNS** 解析：未能解析的 FQDN
  - 总 **P-CSCF** 服务器：通过所有 FQDN 发现的服务器总数
  - 健康（**SIP OPTIONS** 正常）：响应 SIP OPTIONS 健康检查的服务器
  - 不健康（**SIP OPTIONS** 异常）：未响应 SIP OPTIONS 的服务器
  - **DNS** 成功率：成功 DNS 解析的百分比
  - 健康检查间隔：SIP OPTIONS 健康检查的频率（60 秒，超时 5 秒）

响应提供了对 DNS 解析健康和通过 SIP OPTIONS 的 P-CSCF 服务器可用性的当前可见性。要刷新，只需重新运行 `GET /api/pcscf_monitor` — 每次调用返回最新状态。

UPF 选择状态

UPF 选择状态端点（`GET /api/upf_selection`）报告每个规则的 P-CSCF 发现状态：

```
 IMS 流量（优先级 20）
    匹配: APN 匹配 ^ims
    池: UPF-IMS-Primary (10.100.2.21:8805)

     P-CSCF 发现
        FQDN: pcscf.mnc001.mcc001.3gppnetwork.org
        状态: ✓ 已解析（2 个 IP）
        解析的 IP: 203.0.113.100, 203.0.113.101

    ⚙ PCO 覆盖
        主 DNS: 198.51.100.195
        P-CSCF（静态回退）: 203.0.113.100, 203.0.113.101
```

指标与可观察性

Prometheus 指标

P-CSCF 监控系统通过 Prometheus（默认端口 42069）公开指标:

Gauge 指标

```
# FQDN 级别指标
pcscf_fqdns_total                          # 监控的总 FQDN 数量
pcscf_fqdns_resolved                       # 成功解析的 FQDN（DNS 成功）
pcscf_fqdns_failed                         # 失败的 FQDN 解析（DNS 失败）

# 服务器级别指标（汇总）
pcscf_servers_total                        # 通过 DNS SRV 发现的总 P-CSCF 服务
器
pcscf_servers_healthy                      # 响应 SIP OPTIONS 的服务器（汇总）
pcscf_servers_unhealthy                    # 未响应 SIP OPTIONS 的服务器（汇总）

# 服务器级别指标（按 FQDN 带标签）
pcscf_servers_healthy{fqdn="..."}          # 特定 FQDN 的健康服务器
pcscf_servers_unhealthy{fqdn="..."}        # 特定 FQDN 的不健康服务器
```

健康检查详细信息:

- `healthy`: 服务器响应了 SIP OPTIONS ping（TCP 或 UDP）
- `unhealthy`: 服务器未能响应 SIP OPTIONS（每种传输 5 秒超时）

指标示例

DNS 解析指标:

```
# 查询成功解析的 FQDN
pcscf_fqdns_resolved

# 计算 DNS 成功率
(pcscf_fqdns_resolved / pcscf_fqdns_total) * 100

# 发现的服务器总数
pcscf_servers_total
```

SIP OPTIONS 健康指标:

```
# 所有 FQDN 的健康服务器总数
pcscf_servers_healthy

# 不健康服务器总数
pcscf_servers_unhealthy

# 计算健康检查成功率
(pcscf_servers_healthy / pcscf_servers_total) * 100

# 特定 FQDN 的健康服务器
pcscf_servers_healthy{fqdn="pcscf.mnc001.mcc001.3gppnetwork.org"}

# 所有服务器都宕机的警报
pcscf_servers_healthy == 0 AND pcscf_servers_total > 0
```

Prometheus 警报示例:

```yaml
# 当所有 P-CSCF 服务器都宕机时发出警报
- alert: AllPCSCFServersDown
  expr: pcscf_servers_healthy == 0 AND pcscf_servers_total > 0
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "所有 P-CSCF 服务器都不健康"
    description: "{{ $value }} 个健康服务器（0） - 所有 SIP OPTIONS 检查失败"

# 当超过 50% 的服务器宕机时发出警报
- alert: MajorityPCSCFServersDown
  expr: (pcscf_servers_healthy / pcscf_servers_total) < 0.5
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "大多数 P-CSCF 服务器不健康"
    description: "只有 {{ $value }}% 的服务器响应 SIP OPTIONS"

# 关于 DNS 解析失败的警报
- alert: PCSCFDNSResolutionFailed
  expr: pcscf_fqdns_failed > 0
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "P-CSCF DNS 解析失败"
    description: "{{ $value }} 个 FQDN 正在失败解析"
```

日志记录

监控器记录关键事件：

```
[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}]
```

回退策略

该系统使用 三层回退策略 以确保最大可靠性：

第 1 层：DNS 发现（优先）

```
p_cscf_discovery_fqdn: "pcscf.ims.example.com"
```

- 监控器查询 DNS 并缓存解析的 IP
- 会话使用可用的缓存 IP
- 优点：动态、负载均衡、位置感知

第 2 层：规则特定静态 PCO（回退）

```
pco: %{
  p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"]
}
```

- 如果 DNS 发现失败或返回无 IP，则使用
- 规则特定的静态配置
- 优点：规则特定回退，可预测

第 3 层：全局 PCO 配置（最后手段）

```
# 全局 pco 配置
pco: %{
  p_cscf_ipv4_address_list: ["203.0.113.146"]
}
```

- 如果没有规则特定配置且 DNS 失败，则使用
- 全局默认 P-CSCF 地址
- 优点：始终可用，防止会话失败

回退逻辑示例

```
会话匹配 "IMS 流量" 规则：

1. 尝试 DNS 发现 "pcscf.ims.example.com"
   ├─ 成功 → 使用 [203.0.113.100, 203.0.113.101] ✓
   └─ 失败 → 尝试下一层

2. 尝试规则的 PCO 覆盖
   ├─ 配置 → 使用 [203.0.113.100, 203.0.113.101] ✓
   └─ 未配置 → 尝试下一层

3. 使用全局 PCO 配置
   └─ 使用 [203.0.113.146] ✓（始终成功）
```

DNS 配置

DNS 服务器设置

配置 DNS 服务器以进行 P-CSCF 发现的 SRV 和 A/AAAA 记录：

```
; P-CSCF 的 SRV 记录（_sip._tcp 前缀会自动查询）
_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 记录
pcscf1.example.com. IN A 203.0.113.100
pcscf2.example.com. IN A 203.0.113.101
```

重要：OmniPGW 会自动在配置的 FQDN 前添加 `_sip._tcp.`。如果您配置 `p_cscf_discovery_fqdn: "pcscf.mnc001.mcc001.3gppnetwork.org"`，系统将查询 `_sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org`。

SRV 记录格式

SRV 记录遵循以下格式：

```
_service._proto.domain. IN SRV priority weight port target.
```

- 优先级：较低的值具有较高的优先级（10 在 20 之前）
- 权重：在相同优先级之间进行负载均衡（较高 = 更多流量）
- 端口：SIP 端口（通常 TCP 为 5060，UDP 为 5060）
- 目标：要解析为 IP 地址的主机名

测试 DNS 配置

```
# 查询 SRV 记录（注意 _sip._tcp 前缀）
dig SRV _sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org @192.0.2.177

# 预期输出：
# _sip._tcp.pcscf.mnc001.mcc001.3gppnetwork.org. 300 IN SRV 10 50
5060 pcscf1.example.com.

# 将 P-CSCF 主机名解析为 IP
dig A pcscf1.example.com @192.0.2.177

# 预期输出：
# pcscf1.example.com. 300 IN A 203.0.113.100
```

故障排除

问题：FQDN 显示“失败”状态

症状：

- `GET /api/pcscf_monitor` 报告失败状态
- 错误：`:nxdomain`、`:timeout` 或 `:no_naptr_records`

可能原因：

1. DNS 服务器不可达
2. FQDN 在 DNS 中不存在
3. 未配置 NAPTR 记录
4. DNS 服务器超时

解决方案：

```
# 1. 测试 DNS 服务器连接
ping 192.0.2.177

# 2. 手动测试 NAPTR 查询
dig NAPTR pcscf.mnc001.mcc001.3gppnetwork.org @192.0.2.177

# 3. 检查 OmniPGW 日志
grep "P-CSCF" /var/log/pgw_c.log

# 4. 验证配置
grep "p_cscf_discovery_dns_server" config/runtime.exs

# 5. 重新查询 OAM API 以获取当前状态
curl -k https://localhost:8443/api/pcscf_monitor
```

问题：未返回 IP

症状：

- `GET /api/pcscf_monitor` 报告“0 IP”
- 状态可能是已解析或失败

可能原因：

1. NAPTR 记录存在但替换的 FQDN 无法解析
2. 服务字段与 IMS/SIP 模式不匹配
3. 缺少 A/AAAA 记录

解决方案：

```
# 检查 NAPTR 记录服务字段
dig NAPTR pcscf.example.com @192.0.2.177

# 确保服务包含 "SIP" 或 "IMS":
# 正确: "SIP+D2U", "x-3gpp-ims:sip"
# 错误: "HTTP", "FTP"

# 检查 A/AAAA 记录是否存在
dig pcscf1.example.com A @192.0.2.177
```

问题：会话使用错误的 P-CSCF

症状：

- UE 收到意外的 P-CSCF 地址
- 使用静态回退而不是发现的 IP

可能原因：

1. DNS 发现失败但回退正常工作
2. 规则匹配不正确
3. FQDN 未注册

解决方案：

```
# 1. 检查 P-CSCF 监控页面
# 验证 FQDN 是否已注册并解析

# 2. 检查会话日志
grep "Using P-CSCF addresses from FQDN" /var/log/pgw_c.log

# 3. 检查 UPF 选择页面
# 验证规则是否显示正确的 FQDN 和��态

# 4. 测试规则匹配
# 使用特定 APN 创建会话并验证匹配的规则
```

问题：高 DNS 查询延迟

症状：

- 会话创建缓慢
- 指标显示高 `pcscf_discovery_query_duration_seconds`

可能原因：

1. DNS 服务器性能问题
2. 到 DNS 服务器的网络延迟
3. 超时设置过高

解决方案:

```
# 降低查询超时
pco: %{
  p_cscf_discovery_timeout_ms: 2000  # 从 5000ms 降低
}

# 考虑使用更靠近的 DNS 服务器
pco: %{
  p_cscf_discovery_dns_server: "10.0.0.10"  # 本地 DNS
}
```

最佳实践

1. DNS 服务器选择

使用专用 **DNS** 服务器

```
pco: %{
  # 专用于 P-CSCF 发现的专用 DNS（与 UE DNS 不同）
  p_cscf_discovery_dns_server: "192.0.2.177",

  # UE DNS 服务器（提供给移动设备）
  primary_dns_server_address: "8.8.8.8",
  secondary_dns_server_address: "8.8.4.4"
}
```

为什么?

- 分离关注点: UE DNS 与内部 IMS DNS
- 不同的访问策略和安全性
- 独立的扩展和可靠性

2. 始终配置静态回退

```
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com",  # 优先
  pco: %{
    p_cscf_ipv4_address_list: ["203.0.113.100"]  # 必需的回退
  }
}
```

为什么?

- 确保会话即使在 DNS 失败时也成功
- 优雅降级
- 满足 SLA 要求

3. 针对每种流量类型使用特定的 FQDN

```
rules: [
  # IMS
  %{
    name: "IMS",
    match_regex: "^ims",
    p_cscf_discovery_fqdn:
"pcscf.ims.mnc001.mcc001.3gppnetwork.org"
  },

  # 企业
  %{
    name: "Enterprise",
    match_regex: "^enterprise",
    p_cscf_discovery_fqdn: "pcscf.enterprise.example.com"
  }
]
```

为什么?

- 每个服务使用不同的 P-CSCF 池
- 更好的负载分配

- 服务特定路由

4. 监控 DNS 查询性能

```
# 当 P-CSCF 查询延迟高时发出警报
alert: HighPCSCFQueryLatency
expr: histogram_quantile(0.95,
pcscf_discovery_query_duration_seconds_bucket) > 2
for: 5m
labels:
  severity: warning
annotations:
  summary: "P-CSCF DNS 查询缓慢（p95 > 2s）"
```

5. 定期进行 DNS 健康检查

- **OAM API**: 每日轮询 `GET /api/pcscf_monitor`
- 指标: 监控 `pcscf_monitor_fqdns_failed` 指标
- 日志: 关注 DNS 错误
- 测试: 定期验证 DNS 记录是否存在

6. 配置适当的超时

```
# 生产: 平衡可靠性与延迟
pco: %{
  p_cscf_discovery_timeout_ms: 5000  # 5 秒
}

# 高性能: 优先考虑速度，依赖回退
pco: %{
  p_cscf_discovery_timeout_ms: 2000  # 2 秒
}
```

7. 使用 DNS 冗��

配置主 DNS 和备用 DNS:

```
# 主 P-CSCF DNS
pcscf.mnc001.mcc001.3gppnetwork.org. IN NAPTR 10 50 "s" "SIP+D2U"
"" _sip._udp.pcscf1.example.com.

# 备用 P-CSCF DNS
pcscf.mnc001.mcc001.3gppnetwork.org. IN NAPTR 20 50 "s" "SIP+D2U"
"" _sip._udp.pcscf2.example.com.
```

相关文档

- **PCO** 配置 - 协议配置选项，DNS 和 P-CSCF 设置
- 配置指南 - 完整的 OmniPGW 配置参考
- 监控 - 指标、日志记录和可观察性
- 会话管理 - 会话生命周期和 PCO 交付
- **PFCP** 接口 - 用户平面功能通信

返回主文档

PFCP/Sxb 接口文档

数据包转发控制协议 - PGW-C 与 PGW-U 的通信

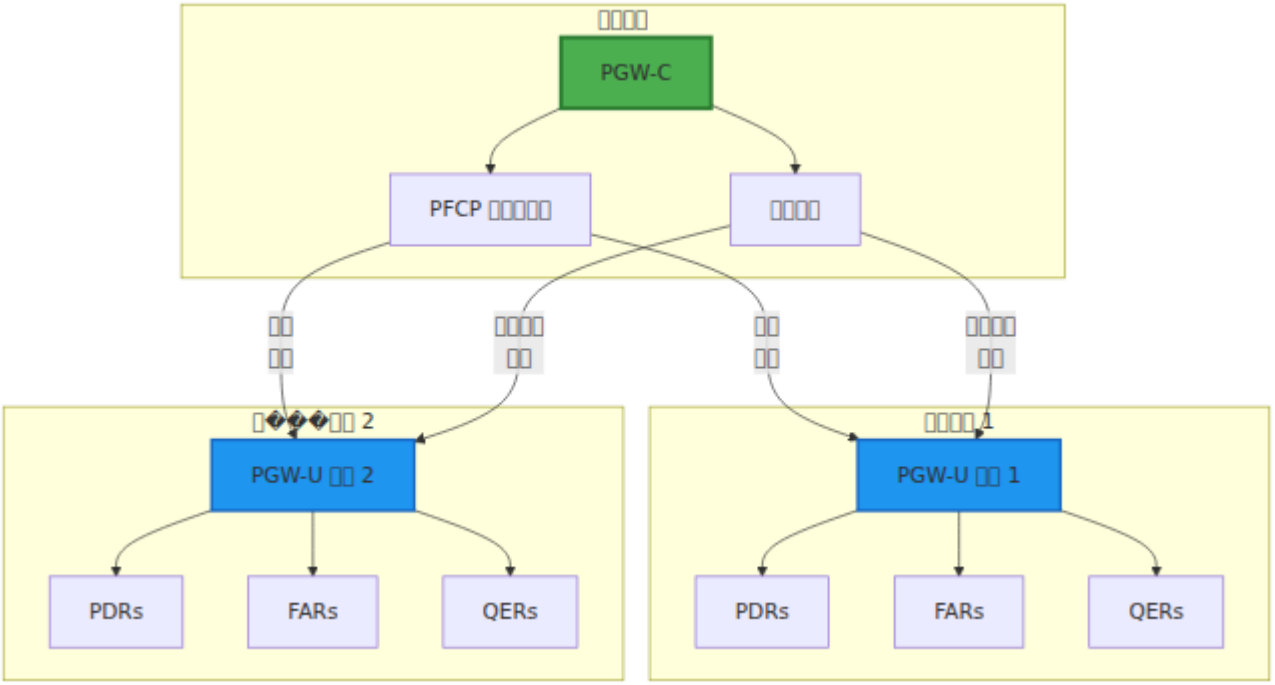
目录

1. □述
2. 协议基础
3. PFCP 关联管理
4. PFCP 会话管理
5. 数据包处理规则
6. 配置
7. 基于 DNS 的 UPF 选择
8. 消息流
9. 故障排除
10. PFCP 监控
11. 相关文档

概述

Sxb 接口 使用 **PFCP**（数据包转发控制协议）进行 PGW-C（控制平面）和 PGW-U（用户平面）之间的通信。这种分离允许：

- 控制平面（**PGW-C**） - 处理信令、会话管理、策略决策
- 用户平面（**PGW-U**） - 以高速处理实际的数据包转发



PFCP 架构

协议基础

PFCP 版本

PGW-C 实现 **PFCP** 版本 **1**（3GPP TS 29.244）。

传输

- 协议：UDP
- 默认端口：8805
- 消息格式：使用 PFCP 规范进行二进制编码

节点 ID 类型

PFCP 对等体通过节点 ID 进行识别，可以是：

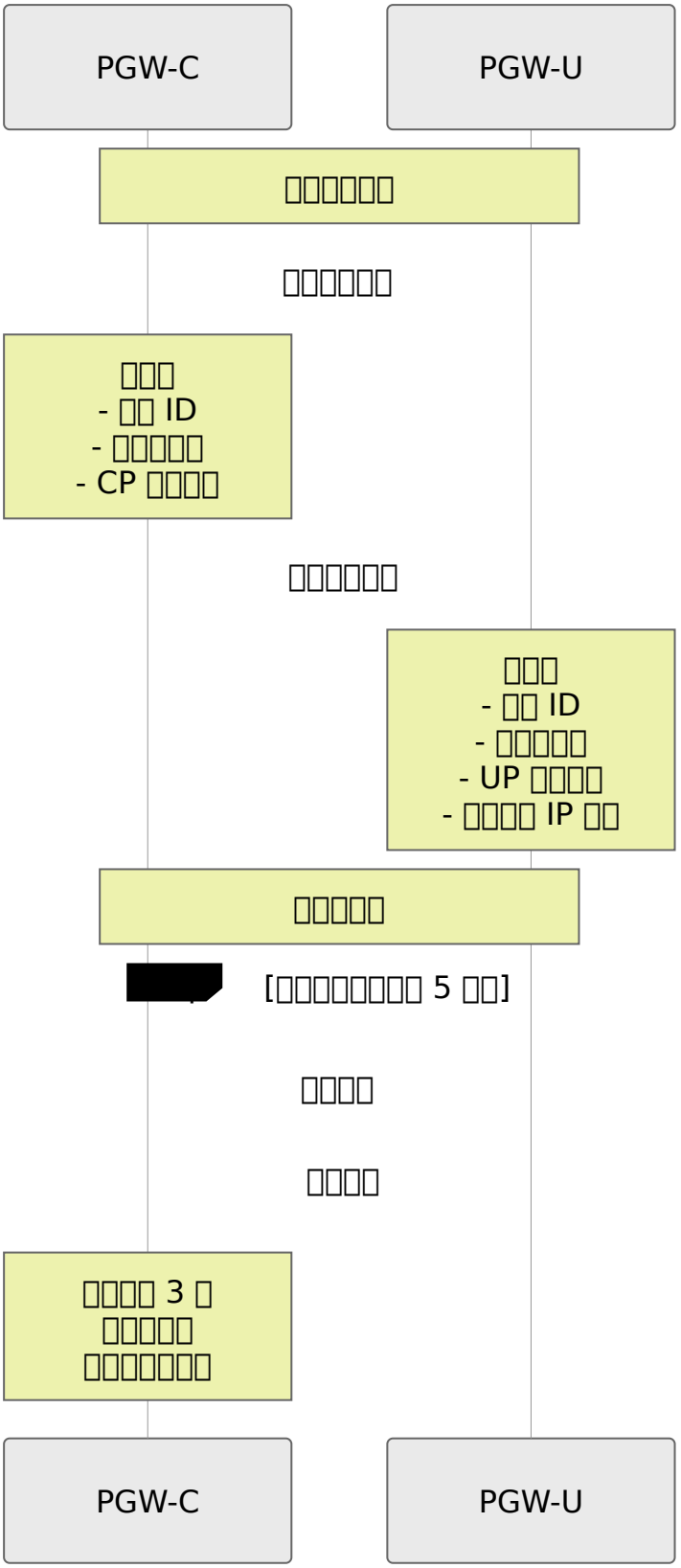
- **IPv4** 地址 - 最常见
- **IPv6** 地址

FQDN（完全合格域名）

PFCP 关联管理

在会话管理发生之前，必须在 PGW-C 和 PGW-U 之间建立 PFCP 关联。

[redacted]


PGW-C
PGW-U
[redacted]
[redacted]
[redacted]
- [redacted] ID
- [redacted]
- CP [redacted]
[redacted]
[redacted]
- [redacted] ID
- [redacted]
- UP [redacted]
- [redacted] IP [redacted]
[redacted]
[redacted]
[redacted] 5 [redacted]
[redacted]
[redacted]
[redacted] 3 [redacted]
[redacted]
[redacted]
PGW-C
PGW-U

对等状态管理

每个 PFCP 对等体维护状态：

字段	描述
`is_associated`	布尔值，指示关联状态
`remote_node_id`	对等体的节点 ID（IP 或 FQDN）
`remote_ip_address`	用于通信的 IP 地址
`remote_port`	UDP 端口（默认 8805）
`heartbeat_period_ms`	心跳间隔（毫秒）
`missed_heartbeats_consecutive`	丢失的心跳计数
`up_function_features`	支持的用户平面功能
`up_recovery_time_stamp`	对等体的恢复时间戳

心跳机制

目的：检测对等体故障并保持关联活跃

配置：

```
# 在 config/runtime.exs 中
sxb: %{
  local_ip_address: "10.0.0.20"
},
upf_selection: %{
  fallback_pool: [
    %{remote_ip_address: "10.0.0.21", remote_port: 8805, weight:
100}
  ]
}
# 所有 UPF 自动注册，心跳间隔为 5 秒
```

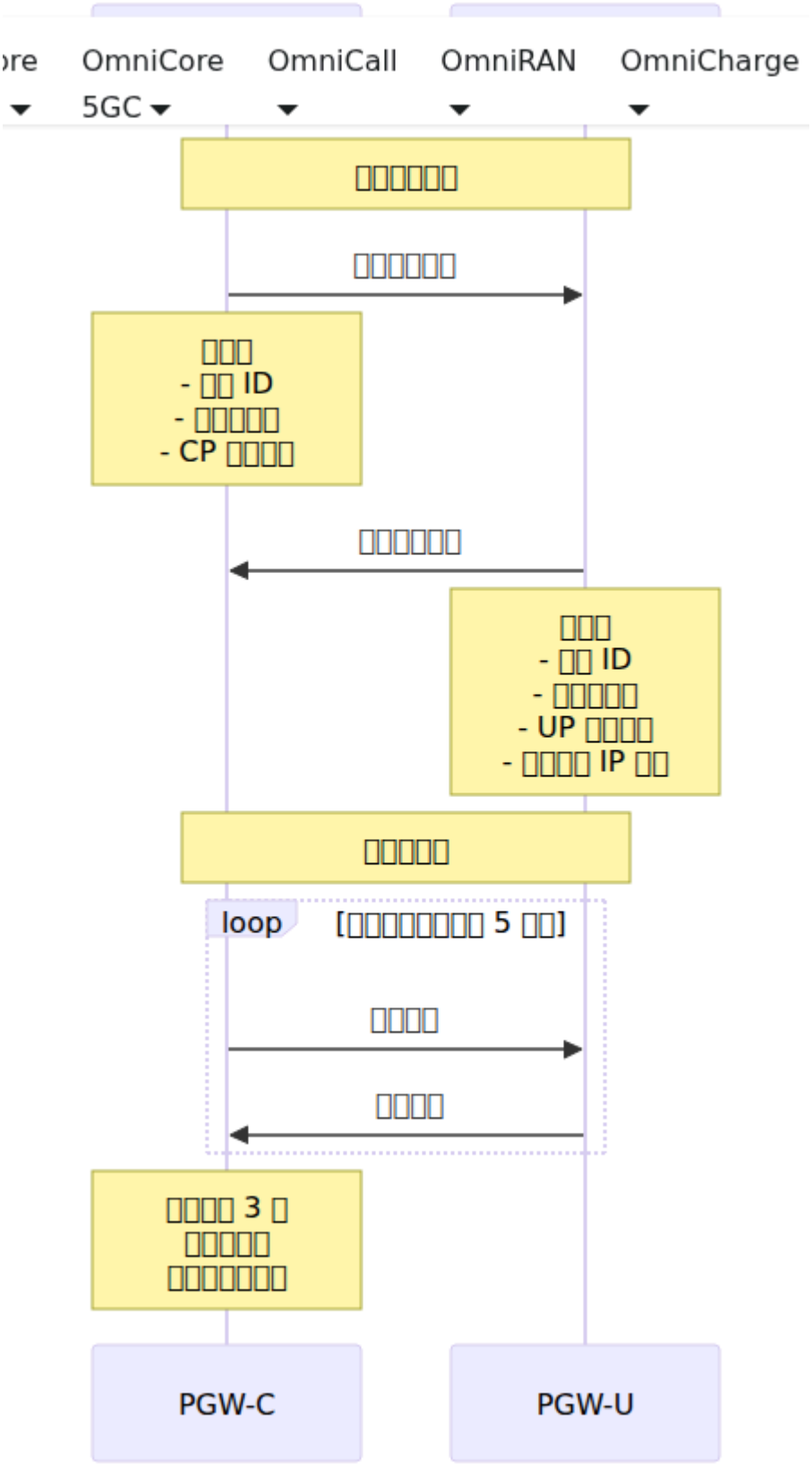
故障检测：

- 每个丢失的心跳会增加 `missed_heartbeats_consecutive`
- 通常配置为在连续 3 次丢失后失败
- 失败的关联会阻止与该对等体的新会话

PFCP 会话管理

PFCP 会话为每个 UE PDN 连接创建，以在用户平面中编程转发规则。

□□□□□□□


ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
□□□□□□□
□□□□□□□
□□□
- □□ ID
- □□□□□
- CP □□□□
□□□□□□□
□□□
- □□ ID
- □□□□□
- UP □□□□
- □□□□ IP □□
□□□□□
loop
[□□□□□□□□ 5 □□]
□□□□
□□□□
□□□□ 3 □
□□□□□
□□□□□□□
PGW-C
PGW-U

会话建立

何时： UE 附着并创建 PDN 连接

PGW-C 发送给 PGW-U：

会话建立请求 包含：

SEID（会话端点 ID） - 唯一会话标识符
节点 ID - PGW-C 的节点 ID
F-SEID - 完全合格的 SEID（包括 IP + SEID） PDRs - 数据包检测规则（通常 2 个：上行 + 下行） FARs - 转发动作规则（通常 2 个：上行 + 下行） QERs - QoS 强制规则（比特率限制）
BAR - 缓冲动作规则（用于下行缓冲）

PGW-U 响应：

会话建立响应 包含：

原因 - 成功或失败原因
F-SEID - PGW-U 的会话端点
创建的 PDRs - 对创建规则的确认
F-TEID - S5/S8 接口的完全合格 TEID

会话修改

何时： QoS 变化、策略更新或承载修改发生时

修改可以包括：

添加新的 PDRs、FARs、QERs
删除现有规则
更新规则参数

会话删除

何时： UE 脱离或 PDN 连接终止时

过程：

1. PGW-C 发送会话删除请求，包含 SEID
2. PGW-U 删除所有规则并释放资源
3. PGW-U 响应会话删除响应

F-TEID 分配

F-TEID（完全合格隧道端点标识符）识别 GTP-U 隧道端点，用于用户平面流量。在建立 PFCP 会话时，必须分配 F-TEID，以标识 UPF 应将上行流量发送到何处。可以采用两种方法：

理解 F-TEID 分配

分配的内容： F-TEID 由以下组成：

TEID（隧道端点标识符） - 32 位数字，标识隧道
IP 地址 - 发送 GTP-U 数据包的地址（UPF 的 IP 地址）

问题：谁分配 TEID 值？

选项 1：UPF 分配（推荐默认）

PGW-C 说“请为我分配一个 TEID”（CHOOSE 标志）
UPF 从其本地池中选择一个 TEID，并响应该值

选项 2：PGW-C 分配（兼容模式）

PGW-C 选择一个 TEID，并告诉 UPF“使用这个特定的 TEID” UPF 使用提供的 TEID，而无需分配

UPF 分配（默认 - 推荐）

配置：

```
sxb: %{
  allocate_uplink_f_teid: false  # 默认
}
```

工作原理：

1. PGW-C 构建带有 F-TEID CHOOSE 标志的 PFCP 会话建立请求
2. UPF 接收请求，从其内部池中分配 TEID
3. UPF 响应分配的 F-TEID（TEID + IP 地址）
4. PGW-C 在会话生命周期内存储分配的 F-TEID

为什么这通常更好：

□ □□□□□

- UPF 拥有用户平面 = UPF 管理用户平面标识符
- PGW-C 无需跟踪 UPF 可用的 TEID
- 每个组件管理自己的资源池

□ □ **PGW-C** 可扩展性

- 多个 PGW-C 实例可以与同一 UPF 通信，而无需协调
- 不同 PGW-C 实例之间没有 TEID 冲突的风险
- UPF 确保所有控制平面对等体之间的唯一性

□ □□ **3GPP** 行为

- CHOOSE 标志在 3GPP TS 29.244 中定义
- 现代 UPF 实现支持该标志
- 遵循“让所有者分配”的原则

□ □□□□□□□□

- 如果 PGW-C 重启，UPF 仍然拥有 TEID 命名空间
- 无需同步 TEID 分配状态
- UPF 可以继续使用现有的 TEID

何时使用：

- □ □□ UPF 的生产部署（默认）
- □ □□ UPF 池的多 PGW-C 部署
- □ □□态控制平面的云原生架构
- □ □□□□□□ 3GPP PFCP 行为

潜在问题：

- ⚠ 一些遗留或专有的 UPF 实现不支持 CHOOSE 标志
- ⚠ 如果会话建立失败并出现“缺少强制 IE”或类似错误，UPF 可能不支持 CHOOSE

PGW-C 分配（遗留兼容性）

配置：

```
sxb: %{
  allocate_uplink_f_teid: true
}
```

工作原理：

1. PGW-C 在会话创建期间从本地池中分配 TEID
2. PGW-C 构建带有显式 TEID 值的 PFCP 会话建立请求
3. UPF 接收请求，使用提供的 TEID，而无需分配
4. PGW-C 和 UPF 都跟踪相同的 TEID 值

您可能需要此的原因：

□ **UPF** 不支持 **CHOOSE**

- 一些 UPF 实现（尤其是遗留/专有）不支持动态分配
- UPF 期望在 PFCP 会话建立请求中提供显式 TEID
- 兼容性的唯一解决方法

□ □□ **TEID** 管理

- 如果您需要 PGW-C 充分了解所有分配的 TEID
- 有助于调试用户平面问题（PGW-C 知道确切的 TEID 值）
- 可以将数据包捕获中的 TEID 与会话状态相关联

□ □□□□□

- 如果您需要可预测的 TEID 分配模式
- 一些测试环境可能需要特定的 TEID 范围

权衡：

⚠ 多 **PGW-C** 需要协调

- 共享 UPF 的多个 PGW-C 实例必须避免 TEID 冲突
- 需要��
  - 每个 PGW-C 的分区 TEID 范围（复杂配置）
  - 共享 TEID 分配服务（额外基础设施）
  - 接受随机分配的冲突风险（□率低）

⚠ 状态同步

- PGW-C 必须跟踪分配的 TEID 以避免重用
- PGW-C 重启时 TEID 池状态丢失（必须从会话重建）
- 更复杂的故障转移场景

⚠ 非标准行为

- 不是预期的 PFCP 设计模式
- 可能与所有期望 CHOOSE 的 UPF 实现不兼容

何时使用：

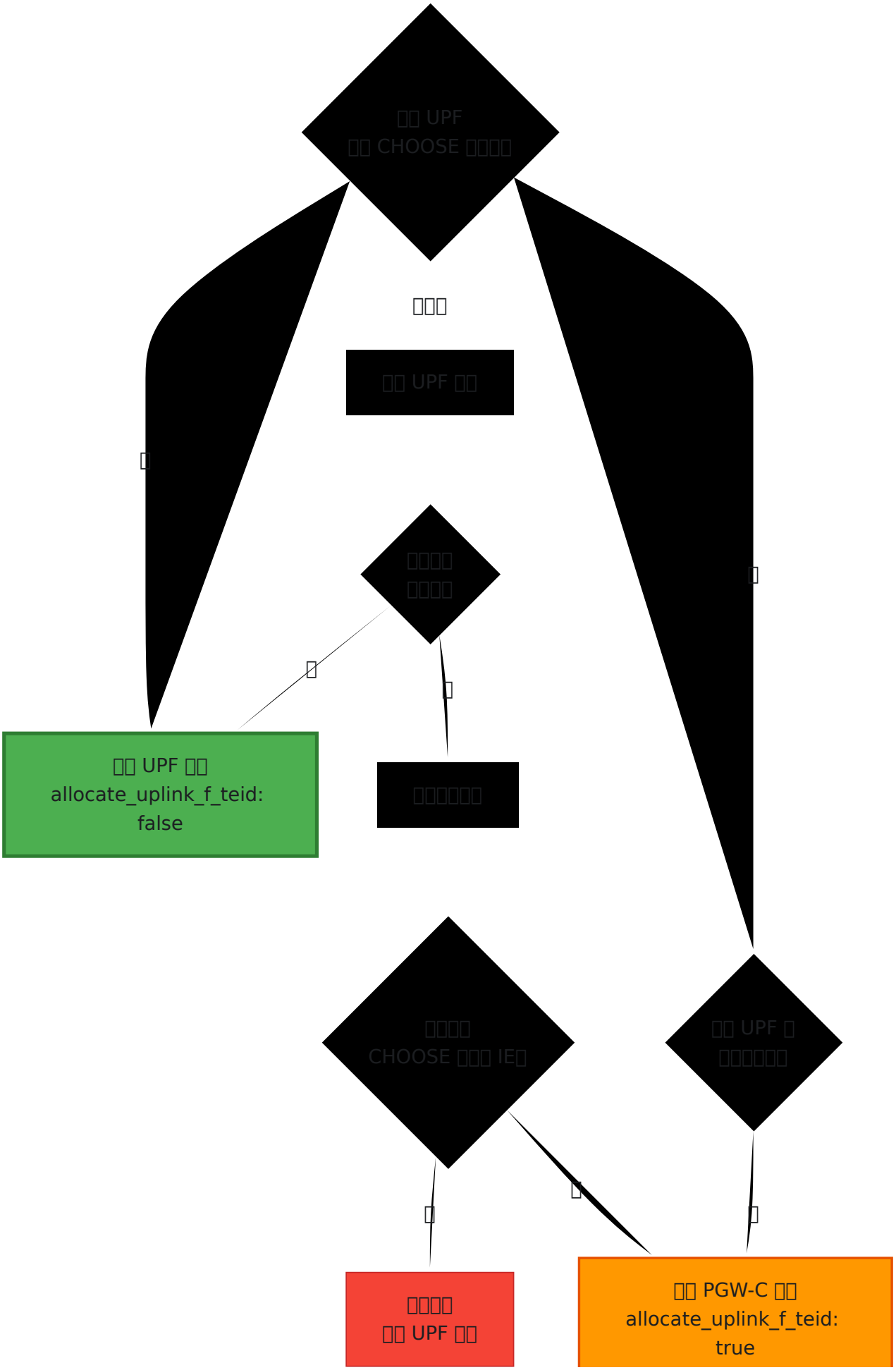
- ⚠ 仅在 **UPF** 不支持 **CHOOSE** 标志时
- ⚠ 遗留 UPF 实现（例如，一些专有硬件）
- ⚠ 特定的兼容性要求
- ⚠ 需要 PGW-C TEID 可见性的调试场景

TEID 冲突处理： PGW-C 使用随机分配和冲突检测：

- TEID 范围：1 到 0xFFFFFFFF（42 亿个值）
- 冲突□率：在 100 万个会话时约为 0.023%
- 冲突时自动重试（对调用者透明）

- 会话终止时自动释放 TEID

如何选择



□□ UPF
□□ CHOOSE □□□□
不知道
□□ UPF □□
□□□□
□□□□
是
□
□□□□□□□
使用 UPF 分配
allocate_uplink_f_teid:
false
□□□□
CHOOSE □□□ IE□
□□ UPF □
□□□□□□□
□
□
□
不同问题
检查 UPF 日志
使用 PGW-C 分配
allocate_uplink_f_teid:
true

故障排除

症状：会话建立立即失败

检查 PFCP 日志：

```
# 查找与 CHOOSE 相关的错误
grep -i "choose\|mandatory.*missing" /var/log/pgw_c.log

# 检查 PFCP 会话建立响应原因代码
grep "Session Establishment Response" /var/log/pgw_c.log
```

如果 **UPF** 拒绝 **CHOOSE** 标志：

- 错误可能会说“缺少强制 IE”或“无效 IE”
- UPF 期望显式 F-TEID，但收到 CHOOSE
- 解决方案：设置 `allocate_uplink_f_teid: true`

如果 **PGW-C** 分配导致问题：

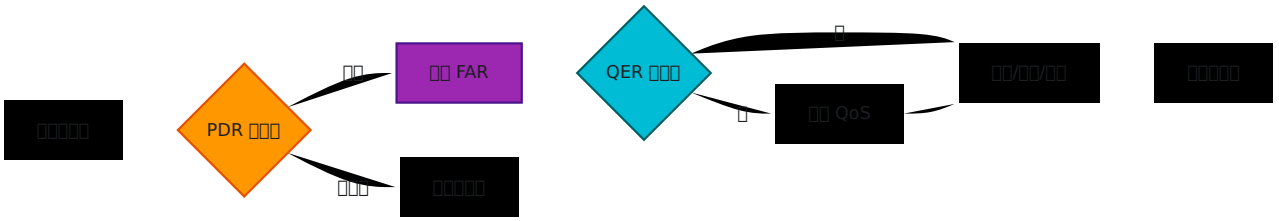
- 非常少见 - TEID 空间巨大（40 亿个值）
- 检查 TEID 耗尽（在数百万会话以下不太可能）：

```
# 检查注册计数
grep "registered_teid_count" /var/log/pgw_c.log
```

在模式之间切换：

```
# 编辑 config/runtime.exs
sxb: %{
  local_ip_address: "10.0.0.20",
  allocate_uplink_f_teid: false  # 如果 UPF 不支持 CHOOSE，则更改为 true
}
```

然后重启 PGW-C：



```
systemctl restart pgw_c
```

验证哪个模式处于活动状态：检查 PFCP 数据包捕获：

```
# 捕获 PFCP 流量
tcpdump -i any -n port 8805 -w pfcp.pcap

# 在 Wireshark 中打开并查看会话建立请求
# 如果 F-TEID 显示“CHOOSE”标志：UPF 分配模式
# 如果 F-TEID 显示显式 TEID 值：PGW-C 分配模式
```

数据包处理规则

PFCP 使用一组规则来定义用户平面如何处理数据包。

规则架构

PDR（数据包检测规则）

目的：确定该规则适用于哪些数据包

典型 **PGW-C** 配置：

PDR #1 - 下行：

```
PDR ID: 1
优先级: 100
PDI（数据检测信息）：
  - 源接口: CORE（互联网侧）
  - UE IP 地址: 100.64.1.42/32
FAR ID: 1（关联的转发规则）
```

PDR #2 - 上行:

```
PDR ID: 2
优先级: 100
PDI（数据包检测信息）：
  - 源接口: ACCESS（SGW 侧）
  - F-TEID: <S5/S8 隧道端点>
FAR ID: 2（关联的转发规则）
QER ID: 1（QoS 强制）
```

关键 **PDR** 字段:

- **PDR ID** - 唯一规则标识符（每个会话）
- 优先级 - 规则匹配优先级（越高越具体）
- **PDI** - 匹配标准（接口、IP、TEID 等）
- 外部头移除 - 在入口处剥离 GTP-U 头
- **FAR ID** - 关联的转发动作
- **QER ID** - 关联的 QoS 强制（可选）

FAR（转发动作规则）

目的：定义对匹配数据包的处理方式

FAR #1 - 下行（互联网 → **UE**）：

```
FAR ID: 1
应用动作: 转发
转发参数:
  - 目标接口: ACCESS (到 SGW)
  - 外部头创建: GTP-U/UDP/IPv4
  - 远程 F-TEID: <SGW S5/S8 隧道端点>
```

FAR #2 - 上行（**UE →** 互联网）：

```
FAR ID: 2
应用动作: 转发
转发参数:
  - 目标接口: CORE (到互联网)
  - (无外部头 - 纯 IP 转发)
```

关键 **FAR** 字段：

- **FAR ID -** 唯一规则标识符
- 应用动作 **-** 转发、丢弃、缓冲、通知
- 转发**��**数：
  - 目标接口（ACCESS/CORE）
  - 外部头创建（添加 GTP-U 隧道）
  - 网络实例（VRF/路由表）

QER（QoS 强制规则）

目的：强制比特率限制和 QoS 参数。QER 还可以跟踪使用情况，以进行在线计费配额管理（请参见 Diameter Gy 接口 以获取计费控制）。

示例 **QER**：

```
QER ID: 1
门状态: 开放
最大比特率:
  - 上行: 100 Mbps
  - 下行: 50 Mbps
保证比特率: (可选，仅适用于 GBR 承载)
  - 上行: 10 Mbps
  - 下行: 10 Mbps
```

关键 **QER** 字段:

- **QER ID** - 唯一规则标识符
- 门状态 - 开放（允许）或关闭（阻止）
- **MBR** - 最大比特率（上行/下行）
- **GBR** - 保证比特率（适用于专用承载）
- **QCI** - QoS 类标识符（影响调度）

BAR（缓冲动作规则）

目的：控制 UE 空闲时的下行数据包缓冲

示例 **BAR**:

```
BAR ID: 1
下行数据通知延迟: 100ms
建议缓冲数据包数量: 10
```

用于：空闲模式 DRX（不连续接收）优化

配置

基本 **Sxb** 配置

编辑 `config/runtime.exs`:

```
config :pgw_c,
  sxb: %{
    # PFCP 通信的本地 IP 地址
    local_ip_address: "10.0.0.20",

    # 可选：覆盖默认端口（8805）
    local_port: 8805,

    # PFCP 请求超时（毫秒）（默认：500ms）
    # 等待 UPF 响应的时间，直到重新传输
    # 应该大于或等于预期的 UPF 处理时间，以避免重复会话
    request_timeout_ms: 500,

    # PFCP 请求的重试尝试次数（默认：3）
    # 总最大等待时间 = request_timeout_ms * request_attempts
    request_attempts: 3,

    # 可选：控制用户平面的 F-TEID 分配
    # 当为 false（默认）：UPF 分配 F-TEID（CHOOSE 标志）
    # 当为 true：PGW-C 预分配 F-TEID 并提供显式值
    # 注意：某些 UPF 可能不支持 CHOOSE 标志并要求显式分配
    allocate_uplink_f_teid: false
  },

  # UPF 选择 - 此处定义的所有 UPF 自动注册
  upf_selection: %{
    fallback_pool: [
      %{
        # PGW-U IP 地址
        remote_ip_address: "10.0.0.21",

        # PFCP 端口（默认：8805）
        remote_port: 8805,

        # 负载均衡权重（100 = 正常，0 = 待机）
        weight: 100
      }
    ]
  }
```

请求超时配置

PFCP 接口使用可配置的超时来处理对 UPF 的会话建立、修改和删除请求。

参数:

参数	类型	默认值	描述
`request_timeout_ms`	整数	500	等待 UPF 响应的时间（毫秒），直到重新传输
`request_attempts`	整数	3	失败请求之前的最大传输尝试次数

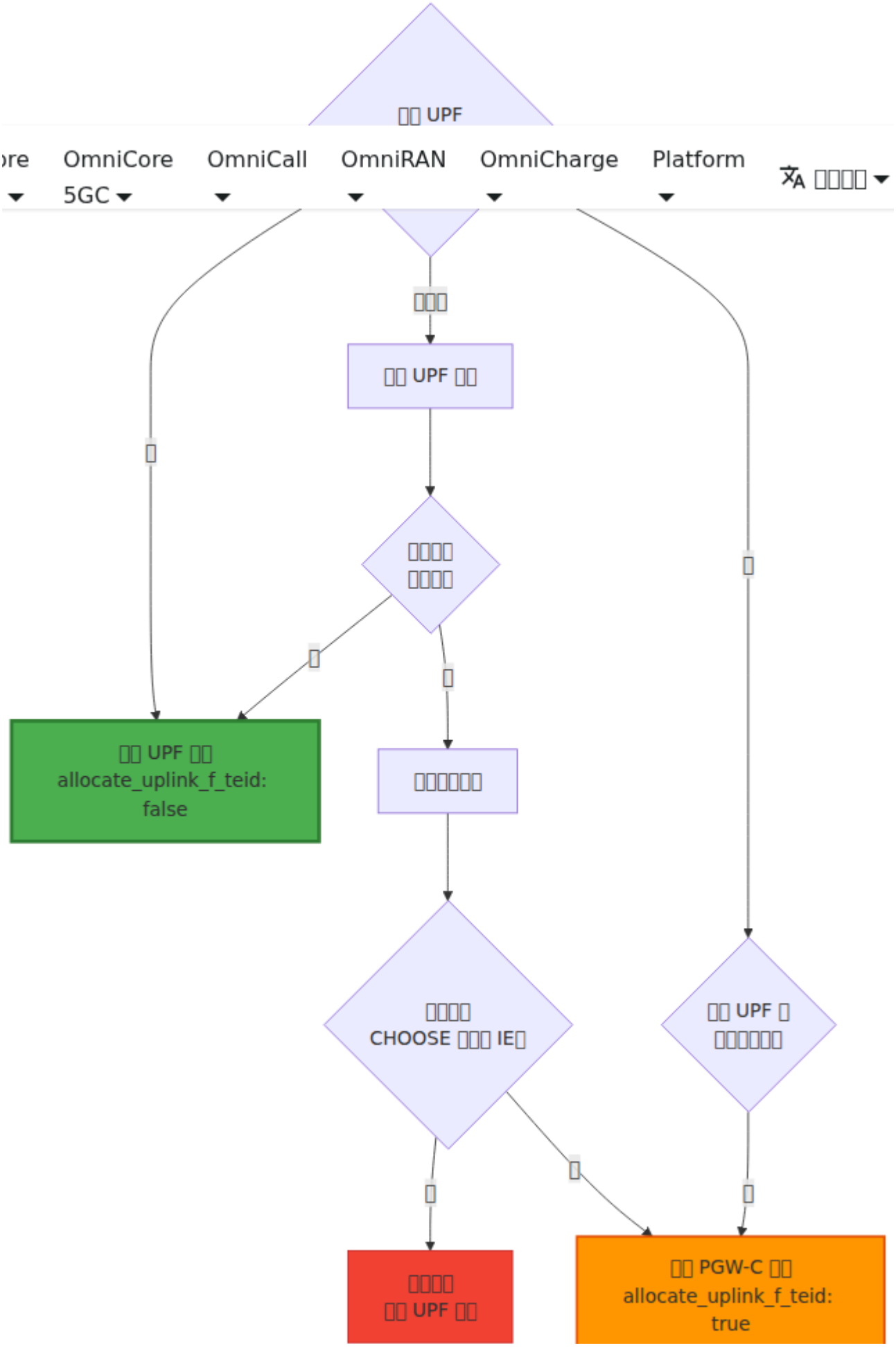
总等待时间: `request_timeout_ms × request_attempts`

默认行为: 500ms × 3 次尝试 = 总最大等待 **1.5** 秒

为什么超时配置很重要

如果 `request_timeout_ms` 相对于 UPF 处理时间设置得太低:

1. PGW-C 发送会话建立请求
2. 超时在 UPF 响应之前到期
3. PGW-C 以相同的序列号重新传输
4. UPF 处理两个请求并创建 重复 **PFCP** 会话
5. PGW-C 收到第一个响应并存储一个会话 ID
6. 第二个会话在 UPF 中变为 孤立的


□□ UPF
ore
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□□□
□□□
□□ UPF □□
□
□□□□ □□□□
□
□
□□ UPF □□ allocate_uplink_f_teid: false
□□□□□□□
□□□□ CHOOSE □□□ IE□
□□ UPF □ □□□□□□□
□
□
□
□□□□ □□ UPF □□
□□ PGW-C □□ allocate_uplink_f_teid: true

调优指南

UPF 响应时间	推荐的 `request_timeout_ms`	总等待时间
快速（<100ms）	200-300ms	600-900ms（3 次尝试）
正常（100-300ms）	500ms（默认）	1.5s（3 次尝试）
慢（300-500ms）	750-1000ms	2.25-3s（3 次尝试）
非常慢（>500ms）	1500-2000ms	4.5-6s（3 次尝试）

建议：将 `request_timeout_ms` 设置为至少 **2×** 预期的 **UPF** 响应时间，以避免因重新传输引起的孤立会话。

示例 - 慢 **UPF**

```
sxb: %{
  local_ip_address: "10.0.0.20",
  request_timeout_ms: 1000,   # 每次尝试 1 秒
  request_attempts: 3          # 总计：最大 3 秒
}
```

诊断超时问题

超时设置过低的症状：

- UPF 报告的 PFCP 会话数量超过预期
- 孤立会话在 UPF 中随时间累积
- 日志消息：“会话建立请求超时”，后跟成功的会话

如何诊断：

1. 通过 UPF API 或管理界面检查 UPF 会话计数
2. 与 PGW-C 活动会话计数进行比较
3. 如果 UPF 有更多会话，则存在因重新传输导致的孤立会话

解决方案：

1. 增加 `request_timeout_ms` 以超过 UPF 响应时间
2. 重启 PGW-C 以应用新配置
3. 从 UPF 中清除孤立会话（手动清理或 UPF 重启）

多个 **PGW-U** 对等体

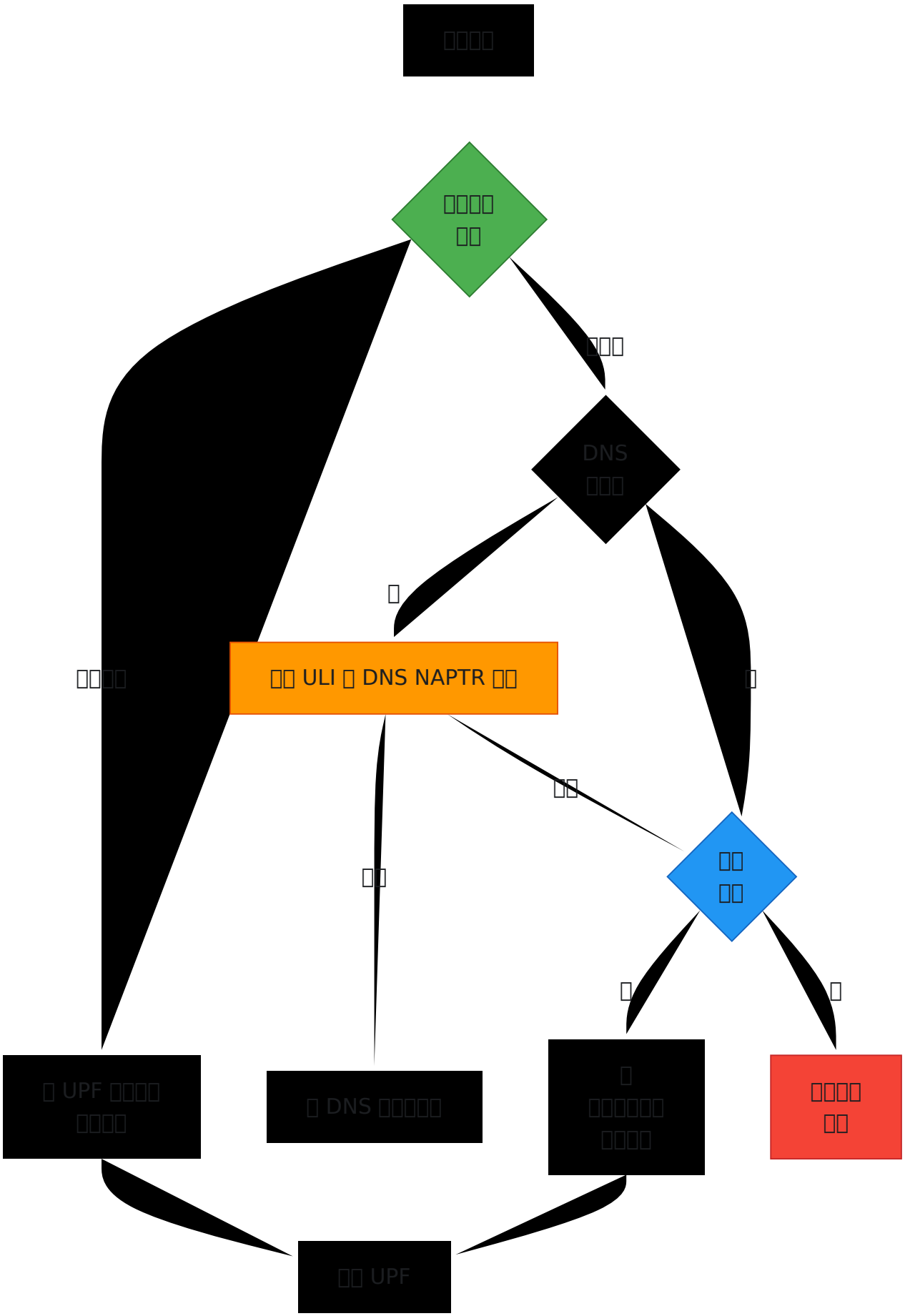
用于负载均衡或冗余：

```
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% 流量
      %{remote_ip_address: "10.0.2.21", remote_port: 8805, weight:
50}   # 50% 流量
    ]
  }
# 两个 UPF 自动注册，心跳间隔为 5 秒
```

UPF 选择配置

PGW-C 使用 三级 UPF 选择系统，具有基于优先级的规则：

1. 静态规则（最高优先级） - 基于会话属性进行匹配
2. 基于 DNS 的选择（中等优先级） - 通过 DNS NAPTR 查询进行位置感知路由
3. 回退池（最低优先级） - 当没有规则匹配时的默认 UPF 池


DNS
否
是
是
否
是
否

□□□□

□□□ **UPF** □□□□

```
config :pgw_c,
  # PFCP 接口
  sxb: %{
    local_ip_address: "10.0.0.20"
  },

  # UPF 选择: 此处定义的所有 UPF 自动注册
  upf_selection: %{
    #
==========================================================================
    # 基于 DNS 的选择（位置感知路由）
    #
==========================================================================
    # 使用用户位置信息（ULI）查询 DNS
    # 提供基于小区位置的动态 UPF 选择
    dns_enabled: false,
    dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],
    dns_suffix: "epc.3gppnetwork.org",
    dns_timeout_ms: 5000,

    #
==========================================================================
    # 静态选择规则（按优先级评估）
    #
==========================================================================
    # 规则按优先级从高到低检查
    # 第一个匹配的规则决定 UPF 池
    rules: [
      # 规则 1: IMS ���量 - 最高优先级
      %{
        name: "IMS 流量",
        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}
        ],
        # 可选: 此规则的 PCO 重写
        pco: %{
          p_cscf_ipv4_address_list: ["203.0.113.100",
```

```
"203.0.113.101"]
        }
      },

      # 规则 2: 企业 APN - 高优先级
      %{
        name: "企业流量",
        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
        }
      },

      # 规则 3: 漫游用户 - 中等优先级
      %{
        name: "漫游用户",
        priority: 10,
        match_field: :serving_network_plmn_id,
        match_regex: "^(310|311|312|313)",  # 美国网络
        upf_pool: [
          %{remote_ip_address: "10.100.4.21", remote_port: 8805,
weight: 100}
        ]
      },

      # 规则 4: 互联网流量 - 较低优先级
      %{
        name: "互联网流量",
        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}
        ]
      }
    ],

    #
================================================================================
    # 回退池（最后的手段）
    #
================================================================================
    # 当没有规则匹配且 DNS 选择失败或被禁用时使用
    fallback_pool: [
      %{remote_ip_address: "127.0.0.21", remote_port: 8805, weight:
100}
    ]
  }
```

支持的匹配字段

匹配字段	描述	示例值
`:imsi`	国际移动用户身份	`"310260123456789"`
`:apn`	接入点名称	`"internet"`，`"ims"`
`:serving_network_plmn_id`	服务网络 PLMN（MCC+MNC）	`"310260"`（测试网络）
`:sgw_ip_address`	SGW IP 地址（字符串格式）	`"10.0.1.50"`
`:uli_tai_plmn_id`	跟踪区 PLMN ID	`"310260"`
`:uli_ecgi_plmn_id`	E-UTRAN 小区 PLMN ID	`"310260"`

UPF 池和负载均衡

每个规则可以指定一个 **UPF** 池，并使用加权随机选择：

```
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}
]
```

加权选择的工作原理：

1. 计算总权重：50 + 30 + 20 = 100
2. 生成随机数：0.0 到 100.0
3. 根据累积权重范围选择 UPF：
   - 0-50：UPF-1（50% 概率）
   - 50-80：UPF-2（30% 概率）
   - 80-100：UPF-3（20% 概率）

用例：

- 均匀分配：所有权重相等（33, 33, 34）
- 主/备：高权重主（80），低权重备（20）
- 基于容量：权重与 UPF 容量成比例

PCO 重写

规则可以重写 PCO（协议配置选项）值：

```
%{
  name: "IMS 流量",
  match_field: :apn,
  match_regex: "^ims",
  upf_pool: [...],
  pco: %{
    # 仅重写特定字段
    p_cscf_ipv4_address_list: ["203.0.113.100", "203.0.113.101"],
    # 其他字段使用主 PCO 配置中的默认值
  }
}
```

可用的 **PCO** 重写字段:

- `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`

基于 **DNS** 的选择

启用时，PGW-C 根据用户位置信息执行 DNS NAPTR 查询:

```
upf_selection: %{
  dns_enabled: true,
  dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],
  dns_suffix: "epc.3gppnetwork.org",
  dns_timeout_ms: 5000
}
```

查询优先级:

1. **ECGI**（E-UTRAN 小区全局标识符） - 最具体
2. **TAI**（跟踪区标识符） - 小区区域
3. **RAI**（路由区域标识符） - 3G/2G 区域
4. **SAI**（服务区域标识符） - 3G 服务区域

5. **CGI**（小区全局标识符） - 2G 小区

示例 **DNS** 查询：

```
# 对于 ECGI 查询：
eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org

# 对于 TAI 查询：
tac-lb64.tac-hb00.tac.epc.mnc999.mcc999.epc.3gppnetwork.org
```

DNS 选择过程：

1. 按优先顺序尝试查询（首先 ECGI，然后 TAI 等）
2. 如果 DNS 返回候选项，则使用第一个结果（如有需要则动态注册）
3. 选择返回的 UPF
4. 如果没有 DNS 匹配或 DNS 被禁用，则回退到回退池

请参见 基于 DNS 的 UPF 选择 以获取详细信息。

基于 **DNS** 的 **UPF** 选择

概述

基于 DNS 的 UPF 选择提供 位置感知路由，通过使用 UE 当前小区的用户位置信息（ULI）执行 DNS NAPTR 查询。

3GPP 参考： TS 23.003 - UPF 发现的 DNS 程序

好处：

- 根据地理位置自动选择 UPF
- 无需为每个小区手动配置规则
- 动态适应网络拓扑变化
- 通过路由到最近的 UPF 减少回程流量

工作原理

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'

□试

配置

```
config :pgw_c,
  upf_selection: %{
    # □用基于 DNS 的选择
    dns_enabled: true,

    # 查询优先级：首先尝试 ECGI，然后 TAI，然后 RAI 等
    dns_query_priority: [:ecgi, :tai, :rai, :sai, :cgi],

    # 查询的 DNS 后缀
    dns_suffix: "epc.3gppnetwork.org",

    # DNS 查询超时
    dns_timeout_ms: 5000,

    # 静态规则仍然优先于 DNS
    rules: [...],

    # 如果 DNS 失败，则回退
    fallback_pool: [...]
  }
```

DNS 查询格式

DNS 查询是使用来自 GTP-C 消息的用户位置信息（ULI）构建的：

1. ECGI（E-UTRAN 小区全局标识符）

最具体 - LTE 小区级路由

格式:

```
eci-<HEX-ECI>.ecgi.epc.mnc<MNC>.mcc<MCC>.<dns_suffix>
```

示例:

```
# 小区 ID: 0x1A2B3C（1,715,004 十进制）
# PLMN: MCC=999, MNC=999
eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org
```

何时使用: LTE（4G）网络

2. TAI（跟踪区标识符）

小区区域 - 同一跟踪区中的多个小区

格式:

```
tac-lb<LB>.tac-hb<HB>.tac.epc.mnc<MNC>.mcc<MCC>.<dns_suffix>
```

示例:

```
# TAC: 0x0064（100 十进制）
# 低字节: 0x64, 高字节: 0x00
tac-lb64.tac-hb00.tac.epc.mnc999.mcc999.epc.3gppnetwork.org
```

何时使用: LTE（4G）跟踪区

3. RAI（路由区域标识符）

3G/2G 路由区域

格式:

```
rac<RAC>.lac-lb<LB>.lac-hb<HB>.lac.rai.mnc<MNC>.mcc<MCC>.
<dns_suffix>
```

示例：

```
# RAC: 0x0A（10 十进制）
# LAC: 0x1234（4660 十进制）
rac0a.lac-lb34.lac-hb12.lac.rai.mnc999.mcc999.epc.3gppnetwork.org
```

何时使用： 3G/2G UMTS/GPRS 网络

4. SAI（服务区域标识符）

3G 服务区域

格式：

```
sac<SAC>.lac-lb<LB>.lac-hb<HB>.lac.sai.mnc<MNC>.mcc<MCC>.
<dns_suffix>
```

示例：

```
# SAC: 0x0001
# LAC: 0x1234
sac0001.lac-lb34.lac-
hb12.lac.sai.mnc999.mcc999.epc.3gppnetwork.org
```

何时使用： 3G UMTS 服务区域

5. CGI（小区全局标识符）

2G 小区级

格式：

```
ci<CI>.lac-lb<LB>.lac-hb<HB>.lac.cgi.mnc<MNC>.mcc<MCC>.
<dns_suffix>
```

示例：

```
# CI: 0x5678
# LAC: 0x1234
ci5678.lac-lb34.lac-hb12.lac.cgi.mnc999.mcc999.epc.3gppnetwork.org
```

何时使用：2G GSM 小区

DNS 响应处理

NAPTR 记录格式：

DNS 返回 NAPTR 记录，指向 UPF IP 地址：

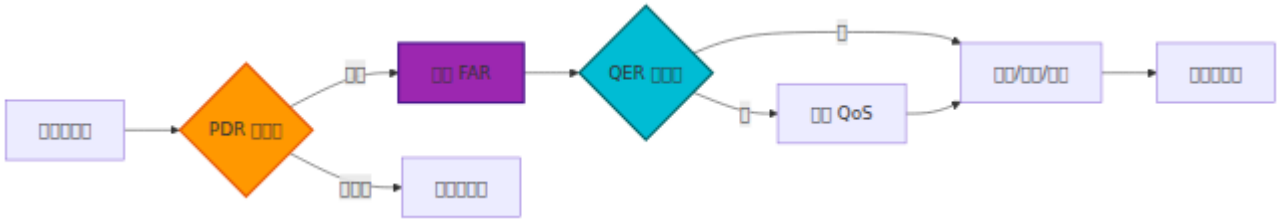
```
eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org.
  IN NAPTR 10 50 "a" "x-3gpp-upf:x-s5-gtp:x-s8-gtp" ""
upf1.epc.mnc999.mcc999.3gppnetwork.org.

upf1.epc.mnc999.mcc999.3gppnetwork.org.
  IN A 10.100.1.21
```

PGW-C 处理：

1. 解析 NAPTR 记录以提取 UPF IP 地址
2. 从 DNS 响应中选择第一个候选项
3. 如果尚未配置，则动态注册（或实现基于负载的选择）

示例：

```
DNS 返回: [10.100.1.21, 10.100.5.99]

选定:     10.100.1.21 (第一个候选项)
操作:       如果不在 upf_selection 中，则动态注册
```



选择优先级示例

用例

1. 地理负载均衡

场景：运营商在多个城市拥有 UPF

DNS 配置：

```
# 芝加哥小区
eci-aaa.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → UPF-芝加哥（10.1.1.21）

# 纽约小区
eci-bbb.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → UPF-纽约（10.2.1.21）

# 洛杉矶小区
eci-ccc.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → UPF-洛杉矶（10.3.1.21）
```

好处：用户自动路由到最近的 UPF，减少延迟和回程流量

2. 边缘计算

场景：在小区站点部署 MEC（多接入边缘计算）UPF

DNS 配置：

```
# 每个小区指向本地边缘 UPF
eci-*.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org → 本地边缘 UPF
```

好处：边缘应用的超低延迟

3. 动态网络拓扑

场景：由于升级或维护，UPF 地址发生变化

好处：更新 DNS 记录，无需更改 PGW-C 配置

故障排除 DNS 选择

DNS 查询失败

症状：

- 日志：“DNS UPF 选择失败：:nxdomain”
- 会话回退到回退池

可能原因：

1. DNS 服务器未正确配置
2. DNS 区域未填充小区 ID
3. GTP-C 消息中未包含 ULI

解决方案：

```
# 手动测试 DNS 查询
dig eci-1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org NAPTR

# 检查 PGW-C 日志中的 DNS 查询
grep "DNS UPF selection: querying" /var/log/pgw_c.log

# 验证会话中存在 ULI
# 检查会话状态中的 "uli" 字段
```

DNS 返回未知 UPF

行为：

- DNS 返回的候选 UPF 不在 `upf_selection` 中
- 系统自动尝试动态注册

- 如果 PFCP 关联成功，则使用该 UPF 进行会话
- 如果 PFCP 关联失败，则回退到回退池

示例：

```
DNS 返回: [10.99.1.50]
upf_selection: [10.100.1.21, 10.100.1.22]

操作：动态注册 10.99.1.50
   - 发送 PFCP 关联设置
   - 如果成功：用于会话
   - 如果超时：回退到回退池
```

解决方案选项：

1. 在 `upf_selection` 中预配置以进行即时监控：

```
upf_selection: %{
  fallback_pool: [
    %{remote_ip_address: "10.99.1.50", remote_port: 8805, weight:
100}
  ]
}
```

2. 更新 DNS 以返回预配置的 UPF IP

3. 允许动态注册（推荐用于 MEC/边缘场景）

查询超时

症状：

- 日志：“DNS UPF 选择：查询超时”
- 会话建立时间较长

解决方案：

```
upf_selection: %{
  dns_timeout_ms: 10000  # 将超时增加到 10 秒
}
```

监控 DNS 选择

指标:

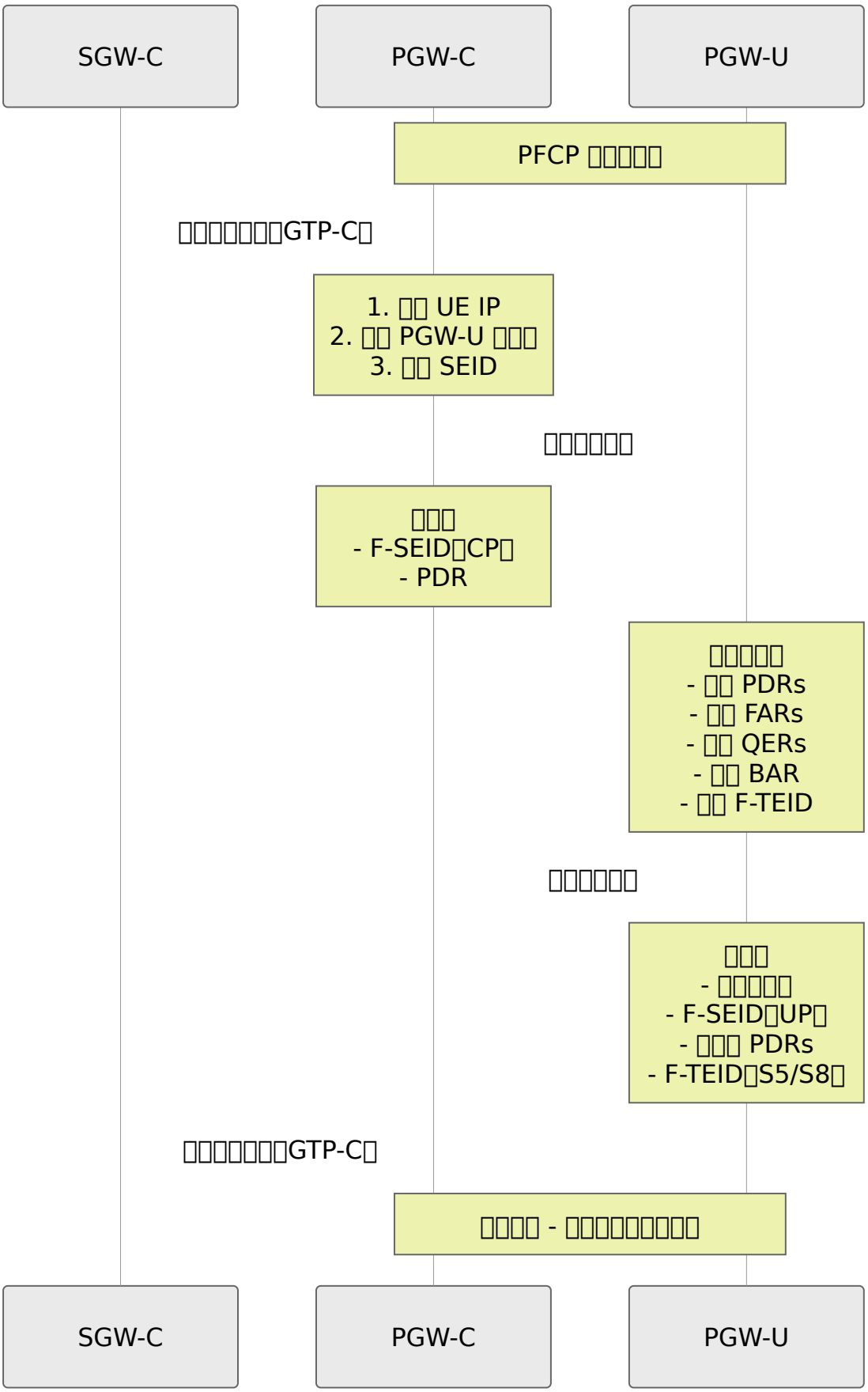
```
# DNS 查询成功率
rate(upf_selection_dns_success_total[5m]) /
rate(upf_selection_dns_attempts_total[5m])

# DNS 查询延迟
histogram_quantile(0.95,
rate(upf_selection_dns_duration_seconds_bucket[5m]))

# 回退使用情况（指示 DNS 问题）
rate(upf_selection_fallback_used_total[5m])
```

日志���

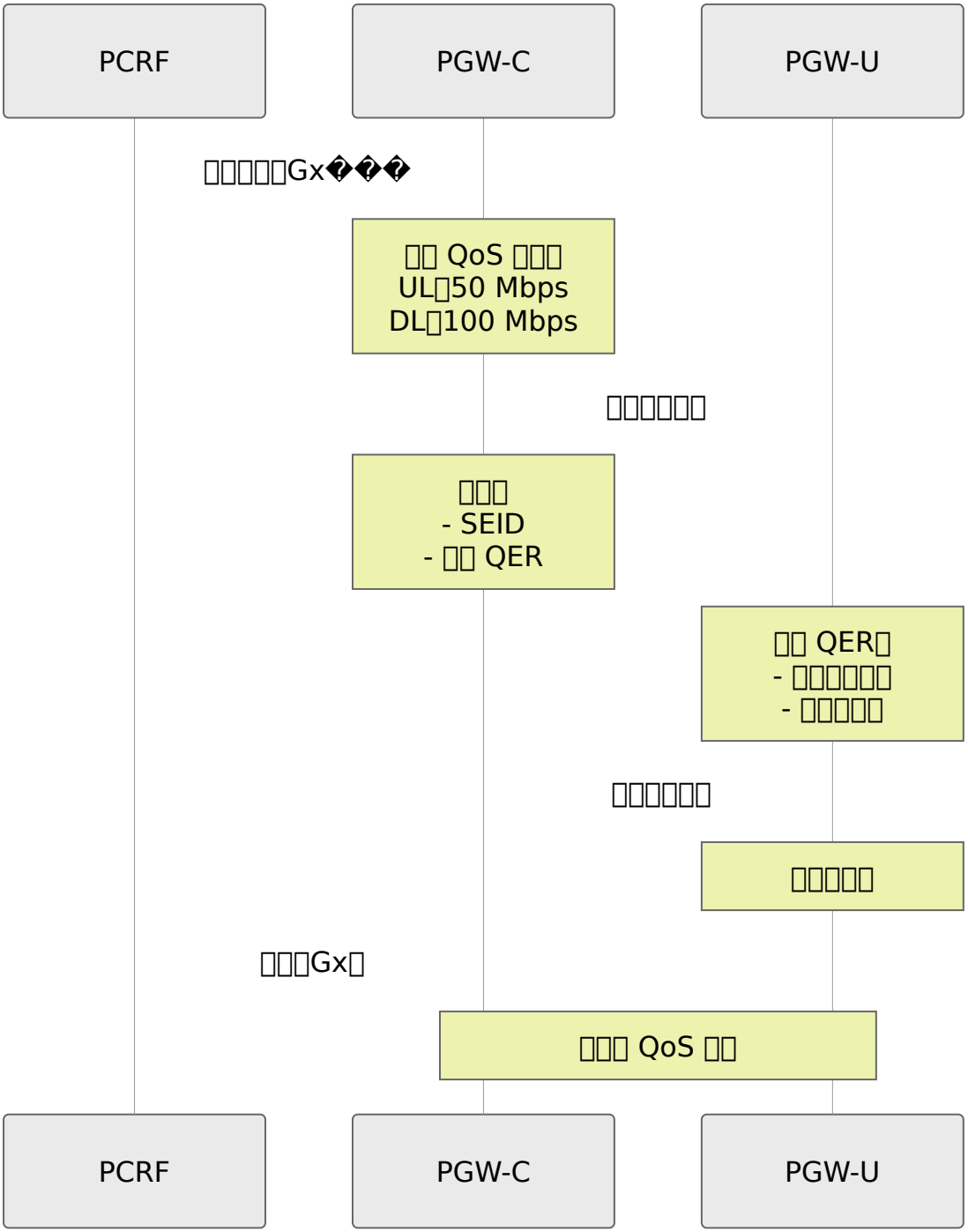
```
[debug] DNS UPF 选择: 查询 eci-
1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org
[debug] DNS UPF 选择: 从 DNS 获得 2 个候选项
[info]  DNS UPF 选择: 选择 10.100.1.21
```

消息流

完整的会话建立流



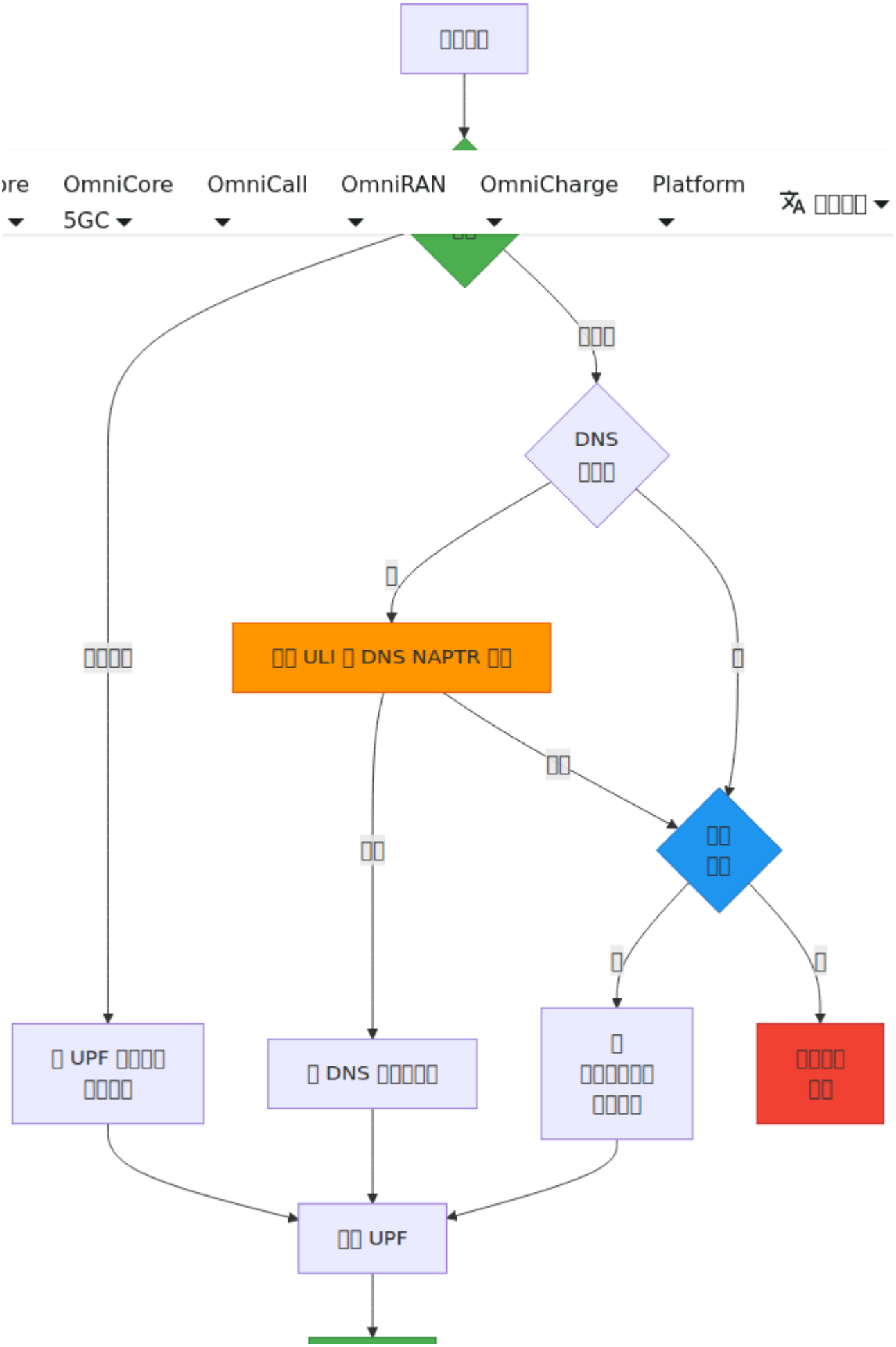
SGW-C
PGW-C
PGW-U
PFCP 关联已建立
创建会话请求（GTP-C）
1. 分配 UE IP
2. 选择 PGW-U 对等体
3. 生成 SEID
会话建立请求
包含:
- F-SEID（CP）
- PDR
安装规则:
- 创建 PDRs
- 创建 FARs
- 创建 QERs
- 创建 BAR
- 分配 F-TEID
会话建立响应
包含:
- 原因: 成功
- F-SEID（UP）
- 创建的 PDRs
- F-TEID（S5/S8）
创建会话响应（GTP-C）
会话活动 - 用户平面转发已启用
SGW-C
PGW-C
PGW-U



会话修改流

PCRF
PGW-C
PGW-U
策略更新（Gx���
新的 QoS 限制:
UL: 50 Mbps
DL: 100 Mbps
会话修改请求
包含:
- SEID
- 更新 QER
更新 QER:
- 应用新比特率
- 更新限速器
会话修改响应
原因：成功
确认（Gx）
更新的 QoS 活动
PCRF
PGW-C
PGW-U

□□□□□□


OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
DNS
ULI DNS NAPTR
UPF
DNS

故障排除

常见问题

1. 关联设置失败

症状：

- 日志消息：“PFCP 关联设置失败”
- 对关联设置请求没有响应

可能原因：

- PGW-U 不可达（网络问题）
- PGW-U 未运行
- 防火墙阻止 UDP 端口 8805
- 配置中的 `remote_ip_address` 不正确

解决方案：

```
# 测试连通性
ping <pgw_u_ip_address>

# 测试 UDP 端口
nc -u -v <pgw_u_ip_address> 8805

# 检查防火墙
iptables -L -n | grep 8805
```

2. 心跳失败

症状：

- 日志：“连续心跳失败：3”

关联标记为失效

可能原因：

网络延迟或数据包丢失
PGW-U 过载
心跳间隔过于激进

解决方案：

心跳周期固定为 5 秒，失败阈值为连续 3 次丢失的心跳。

3. 会话建立失败

症状：

创建会话响应带有错误原因
日志：“PFCP 会话建立失败”

可能原因：

没有可用的 PGW-U 对等体
PGW-U 资源耗尽
无效的规则配置

检查：

1. 验证至少一个对等体的 is_associated = true
2. 检查 PGW-U 日志中的错误
3. 验证 SEID 的唯一性

4. 重复 SEID 错误

症状：

会话建立响应：原因“会话上下文未找到”

原因：

SEID 冲突（非常罕见）

- PGW-U 重启而 PGW-C 未知

解决方案：

- 重新启动 PFCP 关联（触发新的恢复时间戳）
- PGW-C 将检测到 PGW-U 重启并清理旧会话

监控 **PFCP** 健康

监控的指标：

```
# PFCP 对等体关联状态
pfcp_peer_associated{peer="PGW-U Primary"} 1

# 活动 PFCP 会话
seid_registry_count 150

# PFCP 消息速率
rate(sxb_inbound_messages_total[5m])

# PFCP 错误
rate(sxb_inbound_errors_total[5m])

# 心跳失败
pfcp_consecutive_heartbeat_failures{peer="PGW-U Primary"} 0
```

警报示例：

```
# 关联失效时警报
- alert: PFCPAssociationDown
  expr: pfcp_peer_associated == 0
  for: 1m
  annotations:
    summary: "PFCP 对等体 {{ $labels.peer }} 已失效"

# 会话建立失败率□时警报
- alert: PFCPSessionEstablishmentFailureHigh
  expr:
rate(sxb_inbound_errors_total{message_type="session_establishment_res
[5m]) > 0.1
  for: 5m
  annotations:
    summary: "PFCP 会话建立失败率高"
```

PFCP 监控

PFCP/Sxb 关联和 UPF 对等体状态通过 OAM REST API 进行监控。该 API 通过 HTTPS/TLS 在端口 `8443` 上提供，所有路由���在 `/api` 前缀下。交互式文档（Swagger UI）可在 `https://<omnipgw-ip>:8443/api/docs` 获取。

UPF / PFCP 对等体状态

```
GET /api/upf              # 所有 UPF / PFCP 对等体 + 状态
GET /api/upf/<ip>         # 通过 IP 获取一个 UPF
```

目的：监控与所有配置的 PGW-U 对等体的 PFCP 关联状态。

示例：

```
curl -k https://<omnipgw-ip>:8443/api/upf
curl -k https://<omnipgw-ip>:8443/api/upf/10.0.0.1
```

每个配置的 **PGW-U** 对等体返回的 信息：

字段	描述
对等体名称	配置中的友好名称
IP 地址	远程 PGW-U IP
关联状态	已关联或未关联
节点 ID	PFCP 节点标识符
恢复时间戳	对等体的最后重启时间
心跳周期	配置的心跳间隔
连续丢失的心跳	当前失败计数
UP 功能特性	PGW-U 广告的能力

`GET /api/upf/<ip>` 返回单个对等体的完整状态，包括 UP 功能特性位图和关联时间戳。

PFCP 会话检查

活动 PFCP 会话通过每个用户的会话详细信息可见：

```
GET /api/sessions/<imsi>
```

会话详细信息包括关联的 PGW-U 对等体和编程的转发规则（PDRs、FARs、QERs、BARs）以及会话的 F-SEIDs。

作用例

监控 **PFCP** 关联健康：

1. GET /api/upf
2. 验证所有对等体显示“已关联”
3. 检查丢失的心跳计数 = 0
4. 如果某个对等体显示“未关联”:
   - 检查对等体 IP 可达性
   - 验证对等体是否在运行
   - 检查防火墙 (UDP 8805)

故障排除会话建立失败:

1. 用户会话未能建立
2. GET /api/sessions/<imsi> - 会话存在吗?
3. 检查会话的 PFCP 状态 - PFCP 会话已创建?
4. 如果没有 PFCP 会话:
   - GET /api/upf - 是否有任何对等体已关联?
   - 检查 PFCP 错误的日志
5. 如果 PFCP 会话存在:
   - 检查 PDRs/FARs 以验证规则是否已编程
   - 问题可能在下游 (PGW-U 或网络)

验证对等体负载分配:

1. 配置多个 PGW-U 对等体
2. GET /api/upf 以查看每个对等体的状态
3. 验证会话在对等体之间分配
4. 确定是否有一个对等体负载不成比例

检测对等体故障:

- `GET /api/upf` 报告每个对等体的“未关联”状态
- 丢失的心跳计数在完全故障之前显示退化
- 根据 OAM API 数据设置监控警报

优势:

- 可脚本化监控 - 从工具或警报中轮询 `GET /api/upf`
- 明确状态 - 每个对等体的已关联/未关联
- 对等体健康趋势 - 丢失的心跳计数显示早期警告

- 会话级[illegible]查 - 查看编程的确切 PDRs/FARs/QERs

相关文档

配置

- 配置指南 - UPF 选择、健康监控、PFCP 配置
- 会话管理 - PDN 会话生命周期、承载建立

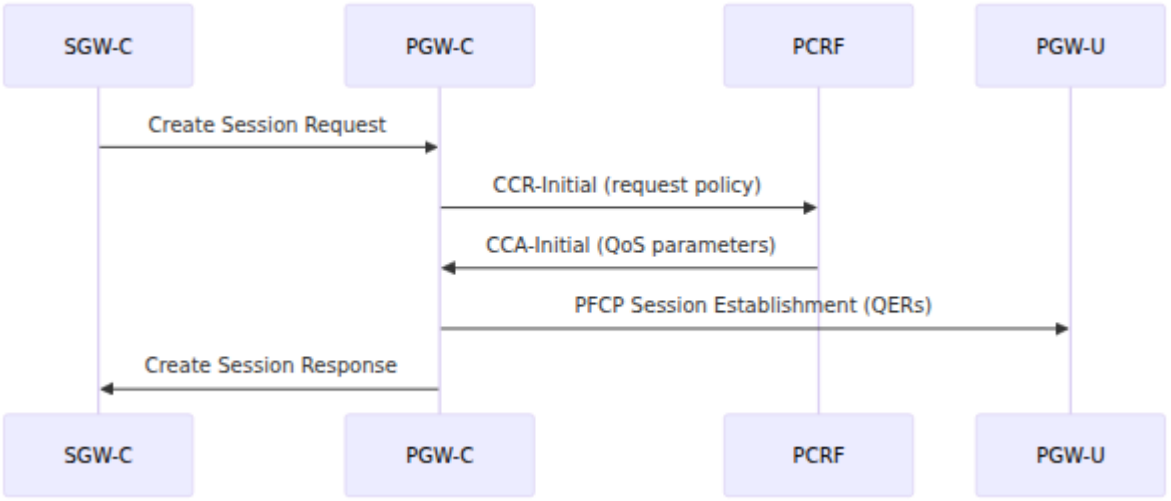
计费和监控

- **Diameter Gx** 接口 - 驱动 PFCP QoS 强制的 PCC 规则
- **Diameter Gy** 接口 - 通过 URR 进行在线计费配额管理
- 数据 **CDR** 格式 - 从 PFCP 使用报告生成 CDR
- 监控指南 - PFCP 指标、会话跟踪、UPF 健康警报

网络接口

- **S5/S8** 接口 - 控制平面承载管理
- **UE IP** 分配 - 通过 PFCP 进行 UE 地址分配

返回[illegible]作指南



QoS 和承载管理

概述

PGW-C 实现了一个基于策略的承载和 QoS 管理系统，协调三个关键接口：

- **Gx (Diameter)** - 从 PCRF 接收策略决策和 QoS 参数
- **S5/S8 (GTP-C)** - 与 SGW-C 管理承载上下文
- **Sxb (PFCP)** - 将 QoS 强制规则编程到 PGW-U

架构流程

关键概念

- 会话：包含 UE 信息、承载映射、PDR/FAR/QER/BAR 映射和 AMBR
- 承载上下文：将 EBI (EPS 承载 ID) 链接到特定的 PDR、FAR 和 QER
- **QER (QoS 强制规则)**：在用户平面中强制 MBR/GBR 限制和门状态
- 默认承载：始终在 PDN 会话中创建，提供基本连
- 专用承载：根据 PCRF 策略动态创建，提供特定的 QoS 保证

配置

重要提示：动态 **QoS** 策略

所有 QoS 参数都是通过 Diameter Gx 接口动态接收自 PCRF，并在 PCRF 中定义（有关更多信息，请参见 OmniHSS）。

运营商在 `config/runtime.exs` 中配置 **PCRF** 连接：

```
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 策略、计费规则和带宽限制在 **PCRF** 上配置，而不是在 **PGW-C** 配置文件中。

承载生命周期

默认承载创建

默认承载在 PDN 会话建立期间创建：

工作流程:

1. SGW-C ���送 Create Session Request
2. PGW-C 从配置的池中分配 UE IP 地址
3. PGW-C 发送 CCR-Initial 到 PCRF, 包含 IMSI、APN、IP 地址
4. PCRF 响应 CCA-Initial, 包含 QoS 参数:
    - Default-EPS-Bearer-QoS (QCI, ARP)
    - QoS-Information (AMBR 调整)

5. PGW-C 创建承载上下文，包含：
固定 ID：下行 PDR=1，上行 PDR=2，下行 FAR=1，上行 FAR=2，QER=1，BAR=1
QER 使用承载 QoS 中的 MBR 编程
6. PGW-C 向 PGW-U 发送 PFCP Session Establishment Request
7. PGW-C 向 SGW-C 发送 Create Session Response

默认承载特性：

在 PDN 会话的整个生命周期内始终存在
通常使用 QCI 5 或 QCI 9（非 GBR）
EBI 在会话状态中跟踪
不能独立删除（删除它会终止会话）

专用承载创建

专用承载根据 PCRF 策略动态创建：

触发器： PCRF 发来的重新认证请求（RAR），包含 Charging-Rule-Install

工作流程：

1. PCRF 发送 RAR，包含 Charging-Rule-Definition，内容包括： Charging-Rule-Name（策略规则标识符）
Flow-Information（数据包过滤器） QoS-Information（QCI、MBR、GBR、ARP）
Precedence（规则匹配优先级）
2. PGW-C 将动态规则转换为 PFCP 实体：
每个 Flow-Information 条目 → 新 PDR，带 SDF 过滤器
QoS-Information → 新 QER，带 MBR/GBR 强制
Flow-Description → IP 5 元组匹配规则
3. PGW-C 发送 PFCP Session Modification Request，以添加 PDRs/FARs/QERs
4. PGW-C 向 SGW-C 发起 Create Bearer Request
5. SGW-C 以 Create Bearer Response 响应，确认建立

示例 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)
```

承载修改

承载 QoS 可以通过以下方式修改:

- **Gx RAR**, 更新 Charging-Rule-Definition
- **PFCP Session Modification**, 更新现有 QERs（更改比特率）、FARs（更改转发）或 PDRs（更改数据包过滤器）

承载删除

触发器:

- **Delete Session Request**（SGW 发起） - 删除默认承载并终止会话
- **Re-Auth Request with Charging-Rule-Remove**（PCRF 发起） - 删除专用承载

工作流程:

1. 从会话状态中移除承载
2. 移除相关的 PDRs/FARs/QERs
3. 向 SGW-C 发送 Delete Bearer Request（如果是 PCRF 发起）
4. 发送 PFCP Session Modification（移除规则）或 Session Deletion（如果是默认承载）

QoS 参数

QCI (QoS 类标识符)

来源：PCRF 通过 Gx `QoS-Class-Identifier` AVP

标准值：

- **QCI 1**：对话语音（GBR，100ms 延迟预算）
- **QCI 2**：对话视频（GBR，150ms 延迟预算）
- **QCI 3**：实时游戏（GBR，50ms 延迟预算）
- **QCI 4**：非对话视频（GBR，300ms 延迟预算）
- **QCI 5**：IMS 信令（非 GBR，100ms 延迟预算） - 默认用于默认承载
- **QCI 6**：视频（基于 TCP）、直播流（非 GBR，300ms 延迟预算）
- **QCI 7**：语音、互动游戏（非 GBR，100ms 延迟预算）
- **QCI 8**：视频（基于 TCP），例如 YouTube（非 GBR，300ms 延迟预算）
- **QCI 9**：默认互联网（非 GBR，300ms 延迟预算）

运营商注意事项：

- QCI 从 PCRF 接收，并在承载级 QoS IE 中向 SGW-C 信号
- PGW-C 不直接强制 QCI 行为 - 实际强制通过 QERs 中的 MBR/GBR 实现
- 较低的 QCI 值通常表示更高的优先级
- QCI 决定数据包转发处理和调度优先级

ARP（分配和保留优先级）

来源：PCRF 通过 `Allocation-Retention-Priority` 分组 AVP

组成部分：

- 优先级级别：1（最高优先级）到 15（最低优先级）
- 抢占能力：此承载是否可以抢占低优先级承载？
  - 0 = 启用（可以抢占其他）
  - 1 = 禁用（不能抢占）
- 抢占脆弱性：此承载是否可以被高优先级承载抢占？

0 = 启用（可以被抢占）
1 = 禁用（不能被抢占）

默认值：

优先级级别：1
抢占能力：启用（0）
抢占脆弱性：禁用（1）

运营商注意事项：

ARP 向 SGW-C 信号，最终到 eNodeB
PGW-C 不强制执行 - 强制通常在 eNodeB 的无线接入控制期间进行
在网络拥塞期间用于确定允许或丢弃哪些承载
对于紧急服务（优先级级别 1）和高价值服务至关重要

MBR (最大比特率)

来源： PCRF 通过 Max-Requested-Bandwidth-UL 和 Max-Requested-Bandwidth-DL AVP

格式：每秒字节（内部转换为 kbps：bytes / 1000）

适用于：所有承载（默认和专用）

工作原理：

PGW-C 创建 QER，带 mbr: %Bitrate{ul: kbps_ul, dl: kbps_dl}
QER 通过 PFCP 发送到 PGW-U
PGW-U 强制速率限制（流量控制）
超过 MBR 的流量将被丢弃

示例：

```
Max-Requested-Bandwidth-UL: 5000000 (5 Mbps)
Max-Requested-Bandwidth-DL: 10000000 (10 Mbps)

→ QER 创建为 mbr: {ul: 5000, dl: 10000} kbps
→ PGW-U 丢弃超过 5 Mbps 的上行数据包
→ PGW-U 丢弃超过 10 Mbps 的下行数据包
```

GBR（保证比特率）

来源：PCRF 通过 `Guaranteed-Bitrate-UL` 和 `Guaranteed-Bitrate-DL` AVP

格式：每秒字节（转换为 kbps）

适用于：仅专用承载（GBR 承载）

工作原理：

- 如果在 Charging-Rule-Definition 中指定了 GBR，则承载为 **GBR** 类型
- PGW-U 通过 QER 强制最低比特率保证
- 需要在 eNodeB 进行适当调度以保留无线资源
- GBR 承载有接入控制 - 如果资源不可用，可以被拒绝

示例：

```
Guaranteed-Bitrate-UL: 1000000 (1 Mbps)
Guaranteed-Bitrate-DL: 2000000 (2 Mbps)

→ QER 创建为 gbr: {ul: 1000, dl: 2000} kbps
→ 网络保证至少 1 Mbps 的上行和 2 Mbps 的下行
→ 用于 VoIP、视频通话、直播流
```

运营商注意事项：

- GBR 需要足够的网络容量规划
- 过度订阅 GBR 资源会导致接入失败
- 通过会话计数和承载指标监控 GBR 使用情况

AMBR（聚合最大比特率）

来源：PCRF 通过 `APN-Aggregate-Max-Bitrate-UL` 和 `APN-Aggregate-Max-Bitrate-DL` AVP

范围：适用于 所有非 **GBR** 承载，针对 APN（而不是每个承载）

工作原理：

- AMBR 是会话中所有非 GBR 承载的聚合限制
- 在 Create Session Response 中发送到 SGW-C
- 强制通常在 eNodeB/SGW 进行
- PGW-C 在会话状态中存储 AMBR，并将其信号发送到 SGW-C

示例：

```
APN-Aggregate-Max-Bitrate-UL: 50000000 (50 Mbps)
APN-Aggregate-Max-Bitrate-DL: 100000000 (100 Mbps)

→ 所有非 GBR 承载的总和不能超过 50 Mbps 上行 / 100 Mbps 下行
→ 各个承载受其自身 MBR 限制
→ AMBR 为每个 UE/APN 提供额外的整体上限
```

运营商注意事项：

- 通过 HSS/PCRF 中的用户配置文件设置
- 用于强制订阅级别（例如，10 Mbps 计划与 100 Mbps 计划）
- 不影响 GBR 承载

流状态和门控

流状态 **(Gx)** 到门状态 **(PFCP)** 映射

PCRF 控制是否允许流量通过 Charging-Rule-Definition 中的 `Flow-Status` AVP：

Flow-Status (Gx)	Gate-Status (PFCP QER)	意义
0 = ENABLED-UPLINK	ul: OPEN, dl: CLOSED	仅允许上行流量
1 = ENABLED-DOWNLINK	ul: CLOSED, dl: OPEN	仅允许下行流量
2 = ENABLED	ul: OPEN, dl: OPEN	两个方向均允许
3 = DISABLED	ul: CLOSED, dl: CLOSED	不允许任何流量
4 = REMOVED	ul: CLOSED, dl: CLOSED	正在删除承载

用例：

- **DISABLED**：用于停用服务或信用耗尽（数据包被丢弃但承载保留）
- **ENABLED-UPLINK**：不常见，但可以用于仅上传服务
- **ENABLED-DOWNLINK**：仅下载服务或信用有限的场景
- **ENABLED**：正常操作

监控与可观察性

Prometheus 指标

会话级指标：

```
session_registry_count          # 活动承载（IMSI，EBI 对）
address_registry_count          # 分配的 UE IP
charging_id_registry_count      # 活动计费会话
```

Gx 接口指标：

```
gx_inbound_messages_total{message_type="gx_RAR"}      # 来自 PCRF 的策略更新
gx_outbound_messages_total{message_type="gx_CCR"}     # 向 PCRF 的策略请求
gx_outbound_transaction_duration_bucket               # 到 PCRF 的延迟
```

PFCP 接口指标：

```
sxb_outbound_messages_total{message_type="pfcp_session_establishment_
sxb_outbound_messages_total{message_type="pfcp_session_modification_r
sxb_outbound_transaction_duration_bucket
```

承载创建指标：

```
s5s8_inbound_messages_total{message_type="create_session_request"}
# 默认承载
s5s8_outbound_messages_total{message_type="create_bearer_request"}
# 专用承载
```

通过 **OAM API** 监控

承载和 QoS 状态通过 OAM REST API 进行检查（HTTPS/TLS 在端口 `8443`，路由在 `/api` 前缀下）。交互式文档（Swagger UI）可在 `https://<omnipgw-ip>:8443/api/docs` 获取。

承载和 QoS 信息通过每个用户的会话详细信息公开：

```
GET /api/sessions/<imsi>        # 会话详细信息，包括承载、QCI/5QI、计费规则
```

示例：

```
curl -k https://<omnipgw-ip>:8443/api/sessions/999000123456789
```

会话详细信息包括会话的活动承载、它们的 QoS 参数（QCI/5QI、MBR、GBR、AMBR）、已安装的计费规则和编程的 QER 映射（门状态和比特率）。没有专用承载端点；承载状态始终从其所属的会话中读取。

关键日志消息

```
[debug] Sending Credit Control Request: ...          # CCR 到 PCRF
[debug] Handling Credit Control Answer: ...          # CCA 来自 PCRF
(包含 QoS)
[debug] Handling Re-Auth Request                     # RAR 来自 PCRF
(策略变更)
[debug] Sending Session Establishment Request        # PFCP 到 PGW-
U (编程 QERs)
[debug] Sending Session Modification Request         # PFCP 到 PGW-
U (更新 QERs)
```

□作任务

验证应用于会话的 **QoS**

1. `GET /api/sessions/<imsi>` (例如, `curl -k https://<omnipgw-ip>:8443/api/sessions/999000123456789`)
2. 在响应中找到该用户的会话
3. 检查会话详细信息
4. 检查 **qer_map** 部分:

   ```
   qer_id: 1
   gate_status: {ul: OPEN, dl: OPEN}
   mbr: {ul: 50000, dl: 100000}  # kbps
   gbr: {ul: 10000, dl: 20000}   # kbps (或 nil 对于非 GBR)
   ```

5. 验证值与预期的 PCRF 策略匹配

排查缺失的 **QoS**

症状: 会话已创建但未应用 QoS

步骤:

1. 检查 **PCRF** 连接性:

- 验证 PCRF 对等状态 = "connected"（查看 Diameter/Gx 指标和对等状态）
- 如果断开，检查网络连接和 Diameter 配置

2. 验证 **CCR/CCA** 交换:
   - 在 OmniPGW 日志中搜索 "Credit Control Answer"
   - 验证 CCA 日志中存在 `QoS-Information` AVP
   - 检查 CCA 中的错误（Result-Code 应为 2001 = SUCCESS）
3. 验证 **PFCP** 编程:
   - 在日志中搜索 "PFCP Session Establishment Request"
   - 验证消息中包含 QER
   - 检查 PGW-U 日志中的 PFCP 处理错误
4. 检查 **PCRF** 策略配置:
   - 验证 PCRF 中的用户配置文件
   - 确认存在 APN 特定的策略规则
   - 检查 PCRF 日志中的策略评估错误

监控承载创建速率

Prometheus 查询:

```
# 默认承载创建速率（会话/秒）
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])

# 专用承载创建速率
rate(s5s8_outbound_messages_total{message_type="create_bearer_request
[5m])

# 来自 PCRF 的策略更新速率
rate(gx_inbound_messages_total{message_type="gx_RAR"}[5m])
```

容量规划

关键指标监控：

```
# UE IP 地址利用率（百分比）
(address_registry_count / <configured_pool_size>) * 100

# 活动承载计数
session_registry_count

# PCRF 查询延迟（P95）
histogram_quantile(0.95, gx_outbound_transaction_duration_bucket)
```

容量限制：

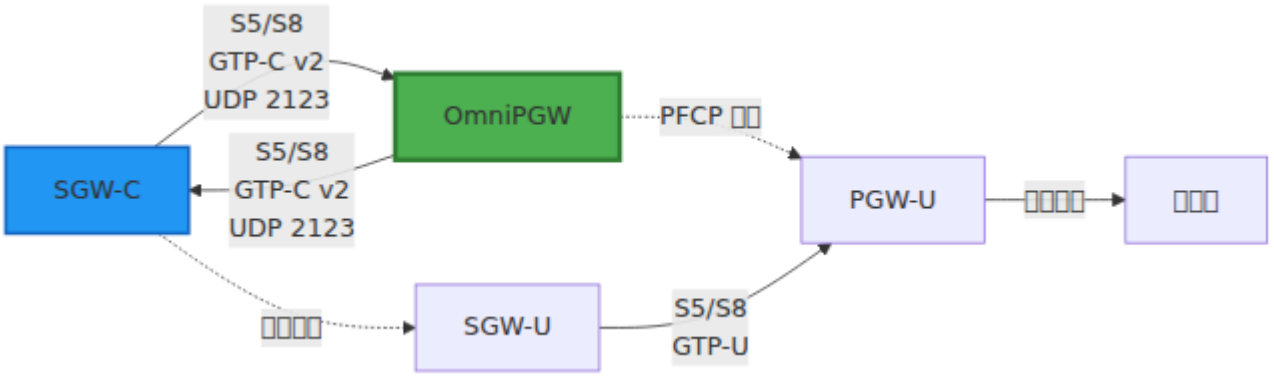
- 地址池大小：在 `config/runtime.exs` 中配置，位于 `ue.subnet_map` 下
- TEID 空间：32 位（40 亿个唯一标识符，自动管理）
- 并发会话：通常受地址池大小限制

规划指南：

- 监控 IP 地址利用率 - 在超过 80% 之前扩展池
- 监控 PCRF 延迟 - 高延迟会影响会话建立时间
- 监控专用承载创建速率 - 指示策略复杂性

相关文档

- 会话管理 - PDN 会话生命周期
- Diameter Gx 接口 - PCRF 策略协议详细信息
- PFCP 接口 - 用户平面编程
- 配置指南 - 系统配置
- 监控指南 - 指标和可观察性



S5/S8 接口文档

与 **SGW-C** 的 **GTP-C** 通信

OmniPGW □ Omnitouch □□□□□□

概述

S5/S8 接口使用 **GTP-C v2**（GPRS 隧道协议 - 控制平面）协议将 OmniPGW 连接到 SGW-C（服务网关控制平面）。该接口处理网关之间的会话管理信令。

协议详情

GTP-C 版本 2

- 协议：GTP-C v2 (3GPP TS 29.274)
- 传输：UDP
- 端口：2123（标准）
- 接口类型：控制平面

TEID（隧道端点标识符）

每个会话都有一个唯一的 **TEID** 用于路由消息：

- 本地 **TEID** - 由 OmniPGW ���传入消息分配
- 远程 **TEID** - 由 SGW-C 为传出消息分配

```
消息流:
  SGW-C → OmniPGW: 目标 TEID = OmniPGW 的本地 TEID
  OmniPGW → SGW-C: 目标 TEID = SGW-C 的远程 TEID
```

配置

基本配置

```
# config/runtime.exs
config :pgw_c,
  s5s8: %{
    # S5/S8 接口的本地 IPv4 地址
    local_ipv4_address: "10.0.0.20",

    # 可选: 本地 IPv6 地址
    local_ipv6_address: nil,

    # 可选: 覆盖默认端口
    local_port: 2123,

    # GTP-C 请求超时（毫秒）（默认: 500ms）
    # 等待 GTP-C 响应（创建承载、删除承载等）时每次尝试的超时
    request_timeout_ms: 500,

    # GTP-C 请求的重试次数（默认: 3）
    # 总最大等待时间 = request_timeout_ms * request_attempts
    # 示例: 500ms * 3 次尝试 = 1500ms（1.5 秒）总计
    request_attempts: 3
  }
```

超时配置

S5/S8 接口使用可配置的超时来处理 GTP-C 请求/响应事务。

参数：

- **`request_timeout_ms`** - 每次重试的超时（毫秒）（默认：500ms）
- **`request_attempts`** - 放弃之前的重试次数（默认：3）

总等待时间：`request_timeout_ms × request_attempts`

默认行为：500ms × 3 次尝试 = 最大总等待 **1.5** 秒

调优指南：

��络延迟	推荐 **`request_timeout_ms`**	总等待时间
低延迟 (<50ms)	200-300ms	600-900ms（3 次尝试）
正常（50-150ms）	500ms（默认）	1.5s（3 次尝试）
高延迟 (>150ms)	1000-2000ms	3-6s（3 次尝试）
不稳定/卫星	2000-3000ms	6-9s（3 次尝试）

示例 - 高延迟网络：

```
s5s8: %{
  local_ipv4_address: "10.0.0.20",
  request_timeout_ms: 1500,  # 每次尝试 1.5 秒
  request_attempts: 3          # 总计：最大 4.5 秒
}
```

当超时发生时：

- OmniPGW 记录错误：`"创建承载请求超时"`
- 返回错误给 PCRF（直径结果代码：5012 UNABLE_TO_COMPLY）
- 承载保持在早期存储中以通过 Charging-Rule-Remove 进行清理

网络要求

防火墙规则：

```
# 允许来自 SGW-C 网络的 GTP-C
iptables -A INPUT -p udp --dport 2123 -s <sgw_network>/24 -j
ACCEPT

# 允许向 SGW-C 的出站 GTP-C
iptables -A OUTPUT -p udp --dport 2123 -d <sgw_network>/24 -j
ACCEPT
```

路由:

```
# 确保到 SGW-C 网络的路由
ip route add <sgw_network>/24 via <gateway_ip> dev eth0
```

消息类型

S5/S8 接口处理 PDN 会话管理的 GTP-C 信令。有关详细的会话生命周期和状态管理，请参见 会话管理指南。

会话管理

创建会话请求

方向: SGW-C → OmniPGW

目的: 建立新的 PDN 连接

关键信息元素（**IEs**）:

IE 名称	类型	描述
IMSI	身份	国际移动用户身份
MSISDN	身份	手机号码
APN	字符串	接入点名称（例如，“internet”）
RAT 类型	枚举	无线接入技术（EUTRAN）
承载上下文	分组	默认承载信息
UE 时区	时间戳	UE 的时区
ULI	分组	用户位置信息（TAI，ECGI）
服务网络	PLMN	服务网络的 MCC/MNC

示例：

```
创建会话请求
├── IMSI: 310260123456789
├── MSISDN: 14155551234
├── APN: internet
├── RAT 类型: EUTRAN (6)
├── 承载上下文
│   ├── EBI: 5
│   ├── 承载 QoS (QCI 9, ARP, 比特率)
│   └── S5/S8 F-TEID (SGW-U 隧道端点)
└── ULI
    ├── TAI: MCC 310, MNC 260, TAC 12345
    └── ECGI: MCC 310, MNC 260, ECI 67890
```

创建会话响应

方向：OmniPGW → SGW-C

目的：确认会话创建

关键信息元素：

IE 名称	类型	描述
原因	结果	成功或错误代码
承载上下文	分组	承载信息
PDN 地址分配	IP	分配的 UE IP 地址（见 UE IP 分配）
APN 限制	枚举	APN 使用限制
PCO	选项	协议配置选项（见 PCO 配置）

成功响应：

```
创建会话响应
├── 原因： 请求已接受 (16)
├── PDN 地址分配
│    └── IPv4: 100.64.1.42
├── 承载上下文
│    ├── EBI: 5
│    ├── 原因： 请求已接受
│    └── S5/S8 F-TEID (PGW-U 隧道端点来自 PFCP)
├── APN 限制: Public-1 (1)
└── PCO
     ├── DNS 服务器: 8.8.8.8
     ├── DNS 服务器: 8.8.4.4
     └── 链接 MTU: 1400
```

> PDN 地址分配布局遵循协商的 PDN 类型（TS 29.274 §8.14）：一个 4 字节的 IPv4 地址；一个前缀长度字节加上 16 字节的 IPv6 前缀；或者对于 IPv4v6，一个前缀长度字节加上 16 字节的 IPv6 加上 4 字节的 IPv4。如果请求了一个没有 IPv6 池的紧急（SOS）APN 的 IPv4v6 PDN，则 PGW-C 降级为仅 IPv4 的 PAA（常规 APN 被拒绝）——见 当 APN 没有 IPv6 池时的 IPv4v6 请求。

删除会话请求

方向： SGW-C → OmniPGW

目的：终止 PDN 连接

关键信息元素：

IE 名称	描述
EBI	要删除的 EPS 承载 ID
关联 EBI	相关承载（可选）

删除会话响应

方向： OmniPGW → SGW-C

目的：确认会话删除

关键信息元素：

IE 名称	描述
原因	成功或错误代码

承载管理

创建承载请求

方向： OmniPGW → SGW-C

目的：创建专用承载（由 PCRF 策略发起）

触发条件：

- PCRF 发送新的 PCC 规则要求专用承载
- OmniPGW 请求 SGW-C 建立承载

删除承载请求

方向： OmniPGW → SGW-C 或 SGW-C → OmniPGW

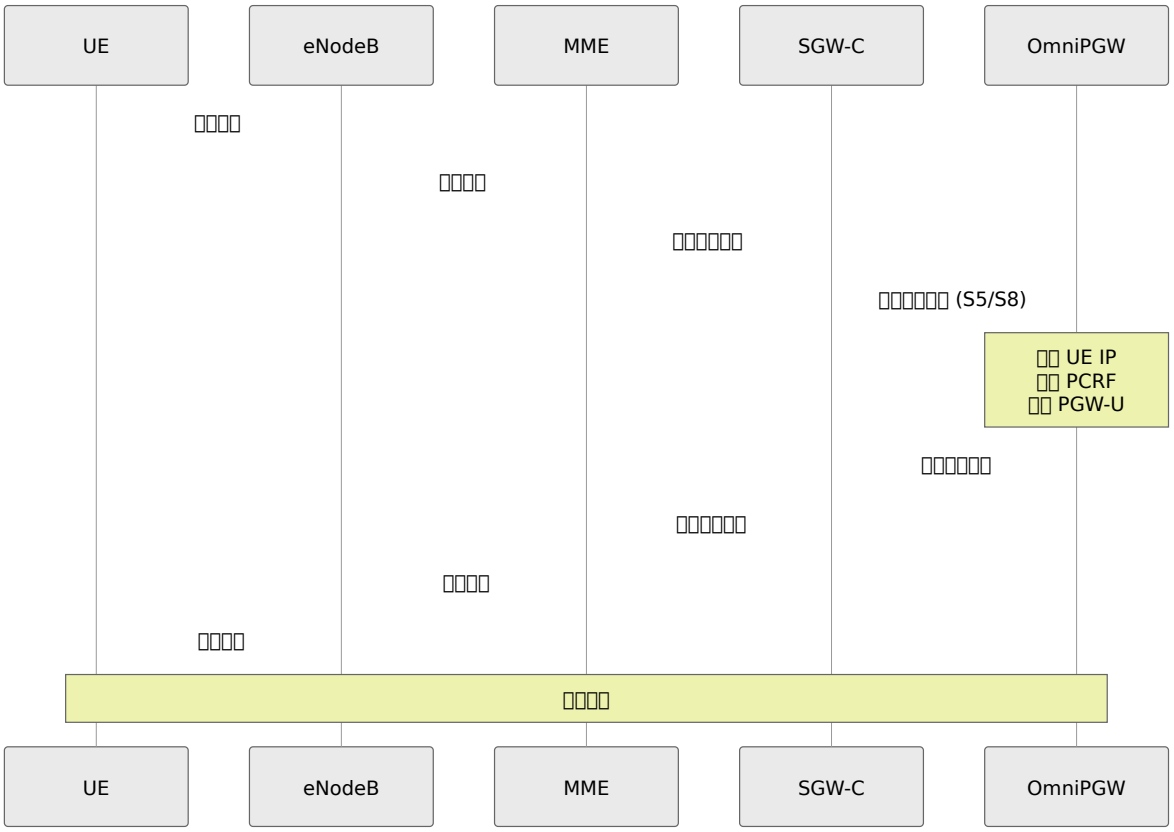
□□□ □□□□□□□

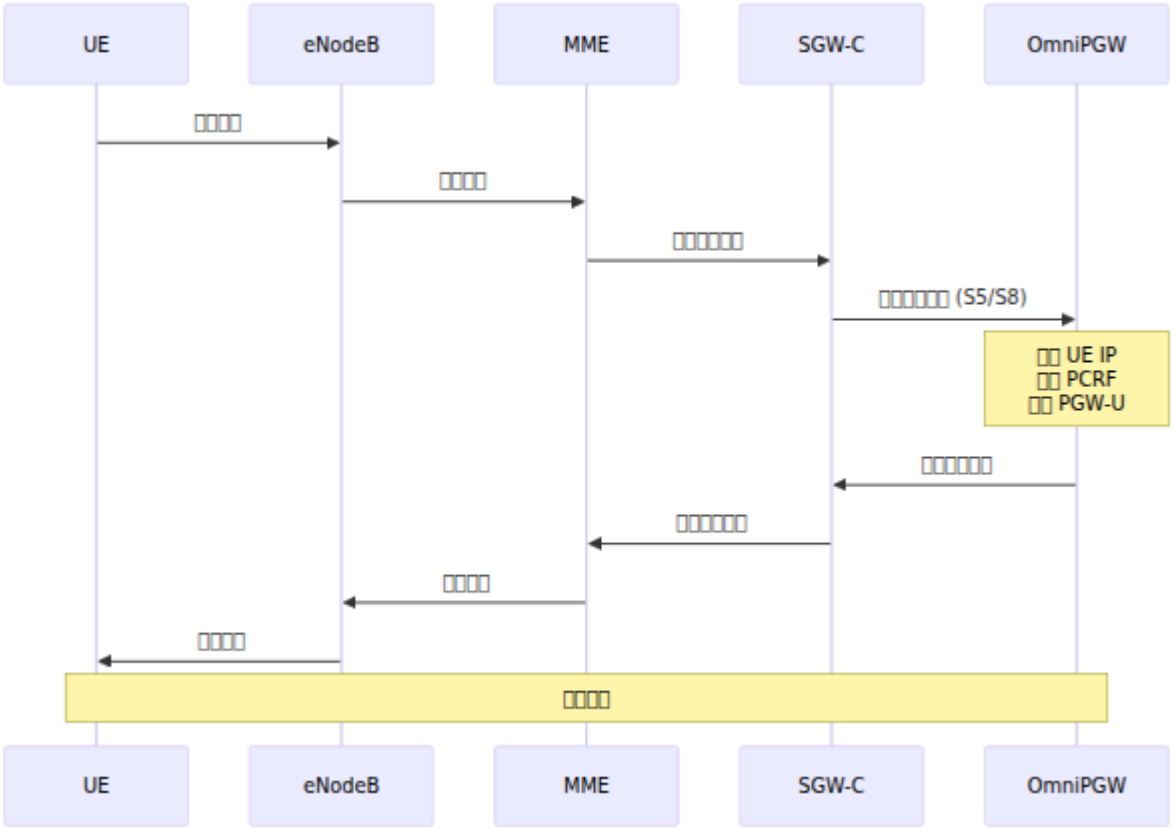
□□□

- **PGW** □□□ PCRF □□□□□□□□□□□□
- **SGW** □□□ □□□□□□□

□□□

□□□□


UE
eNodeB
MME
SGW-C
OmniPGW
□□□□
□□□□
□□□□□□□
□□□□□□□ (S5/S8)
□□ UE IP
□□ PCRF
□□ PGW-U
□□□□□□□
□□□□□□□
□□□□
□□□□
□□□□
UE
eNodeB
MME
SGW-C
OmniPGW



会话终止

原因代码

成功

代码	名称	描述
16	请求已接受	操作成功

错误（永久失败）

代码	名称	使用时机
65	用户未知	PCRF 拒绝（未找到 IMSI）
66	无可用资源	IP 池耗尽
93	服务不支持	无效的 APN
94	TFT 中的语义错误	无效的流量流模板

错误（瞬态失败）

代码	名称	使用时机
72	远程对等方未响应	PCRF/PGW-U 超时
73	与网络发起请求冲突	同时操作

监控

S5/S8 指标

```
# 消息计数器
s5s8_inbound_messages_total{message_type="create_session_request"}
s5s8_inbound_messages_total{message_type="delete_session_request"}

# 错误计数器
s5s8_inbound_errors_total

# 消息处理延迟
s5s8_inbound_handling_duration_bucket

# 活动 TEID
teid_registry_count
```

有用的查询

会话创建率：

```
rate(s5s8_inbound_messages_total{message_type="create_session_request
[5m])
```

错误率：

```
rate(s5s8_inbound_errors_total[5m])
```

延迟（**p95**）：

```
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)
```

故障排除

问题：没有来自 **OmniPGW** 的响应

症状：

- SGW-C 发送创建会话请求
- 没有收到响应
- SGW-C 超时

原因：

1. 网络连接问题
2. OmniPGW 未在配置的 IP 上监听
3. 防火墙阻止 UDP 2123
4. 请求中的 TEID 错误

调试：

```
# 检查 OmniPGW 是否在监听
netstat -ulnp | grep 2123

# 检查传入数据包
tcpdump -i any -n port 2123

# 验证配置
grep "local_ipv4_address" config/runtime.exs

# 检查防火墙
iptables -L -n | grep 2123
```

问题：会话创建失败

症状：

- 创建会话响应带有错误原因
- 会话未建立

常见原因:

```
原因 65（用户未知）：
  → PCRF 拒绝用户
  → 检查 HSS/SPR 中的 IMSI

原因 66（无资源）：
  → IP 池耗尽
  → 检查: curl http://pgw:9090/metrics | grep address_registry_count
  → 扩展 IP 池

原因 72（远程对等方未响应）：
  → PCRF 超时或 PGW-U 停机
  → 检查 Gx 连接
  → 检查 PFCP 关联
```

问题: TEID 冲突

症状:

- 消息路由到错误的会话
- 意外行为

原因:

- TEID 在清理之前被重用
- TEID 分配中的错误

解决方案:

- 确保 TEID 分配唯一
- 检查 TEID 注册表是否泄漏

最佳实践

网络设计

1. 专用网络接口

为 S5/S8 使用单独的 VLAN
与管理流量隔离

2. MTU 优化

确保 MTU 支持 GTP 头
最小 MTU：1500 字节（1464 有效载荷 + 36 GTP）

3. 冗余

多个 OmniPGW 实例
从 SGW-C 进行基于 DNS 的负载均衡

性能

1. UDP 缓冲区大小

为高负载增加套接字缓冲区
典型：每个套接字 4-8 MB

2. 连接限制

针对预期的会话数量进行规划
监控 TEID 注册表计数

安全

1. IP 过滤

仅允许来自已知 SGW-C IP 的 GTP-C
使用 iptables 或网络 ACL

2. 消息验证
    - OmniPGW 验证所有传入消息
    - 拒���格式错误的 GTP-C 数据包

相关文档

核心功能

- 配置指南 - S5/S8 接口配置，本地 IP 设置
- 会话管理 - PDN 会话生命周期，承载建立
- **UE IP** 分配 - 通过创建会话响应交付 IP 地址
- **PCO** 配置 - GTP-C 消息中的 PCO 参数

相关接口

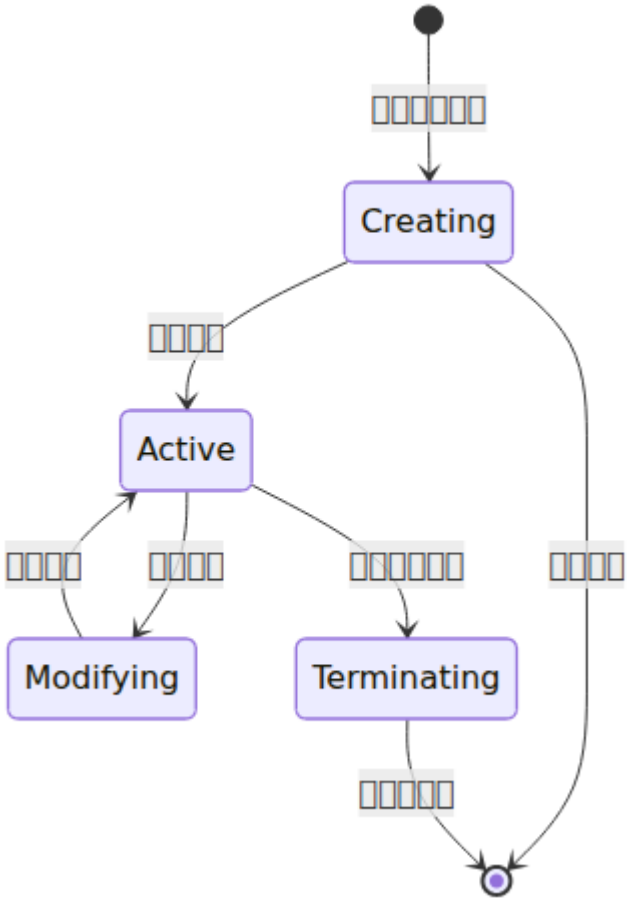
- **Gn/Gp** 接口 - 2G/3G GGSN 功能的 GTP-C v1
- **PFCP** 接口 - 与 S5/S8 控制平面的用户面协调
- 直径 **Gx** 接口 - 与承载设置的策略集成
- 直径 **Gy** 接口 - 与承载管理的计费集成

□作

- 监控指南 - S5/S8 GTP-C 指标，消息跟踪
- 数据 **CDR** 格式 - 从 GTP-C 会话生成 CDR

返回□作指南

OmniPGW S5/S8 接口 - *□ Omnitouch □□□□□□*



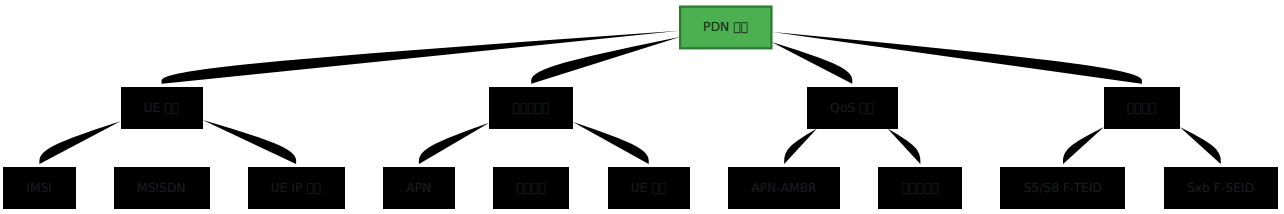
会话管理指南

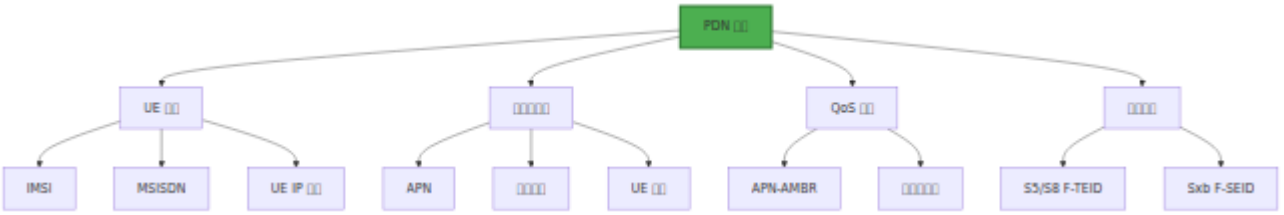
PDN 连接生命周期和□作

OmniPGW □ Omnitouch □□□□□□

概述

一个 **PDN（**分组数据网络**）** 会话 代表 UE 通过 OmniPGW 的数据连接。每个会话协调多个接口和资源以启用数据连接。





会话组件

会话标识符

每个会话有多个标识符用于不同的接口：

标识符	接口	目的
TEID	S5/S8 (GTP-C)	SGW-C 通信的隧道端点 ID
SEID	Sxb (PFCP)	PGW-U 通信的会话端点 ID
Session-ID	Gx (Diameter)	PCRF 通信的 Diameter 会话
Charging-ID	计费	唯一的计费 ID

会话数据

会话创建

调用流程

步骤

1. 接收创建会话请求 (S5/S8)

会话创建通过 S5/S8 接口上的 GTP-C 信令启动。有关完整的 GTP-C 协议细节和消息格式，请参见 S5/S8 接口。

输入：

- IMSI, MSISDN, IMEI
- APN (例如, "internet")
- RAT 类型 (EUTRAN)
- UE 位置 (TAI, ECGI)
- 承载上下文 (QoS, F-TEID)

2. 资源分配

```
- 从 APN 池中分配 UE IP
- 生成计费 ID
- 生成 Gx 会话-ID
- 分配 S5/S8 TEID
- 选择 PGW-U 对等体
```

3. 策略请求 (Gx)

向 PCRF 请求策略:

- 发送 CCR-初始
- 接收带有 QoS 和 PCC 规则的 CCA-初始

4. 用户平面设置 (PFCP)

使用转发规则配置 PGW-U:

- 发送会话建立请求
- 包括 PDRs, FARs, QERs, BAR
- 接收 S5/S8 隧道的 F-TEID

5. 响应 SGW-C

发送创建会话响应：

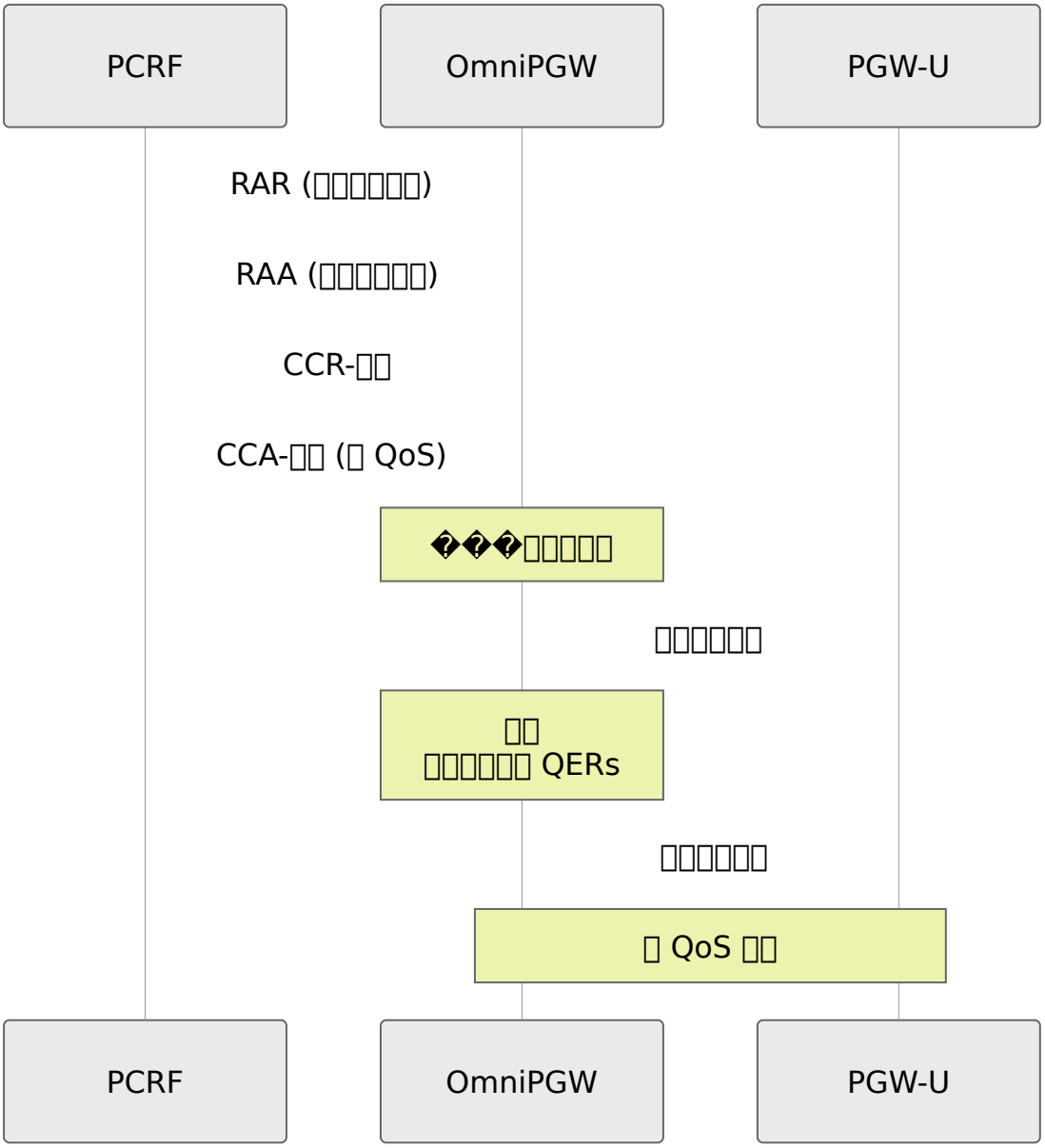
- UE IP 地址
- S5/S8 F-TEID (来自 PGW-U)
- PCO (DNS, P-CSCF, MTU)
- 承载上下文

会话修改

触发器

会话可以由于以下原因进行修改：

- **QoS** 变化 - PCRF 更新比特率
- 承载□作 - 添加/移除专用承载
- 切换 - SGW 更改
- 策略更新 - PCRF 发来的新 PCC 规则

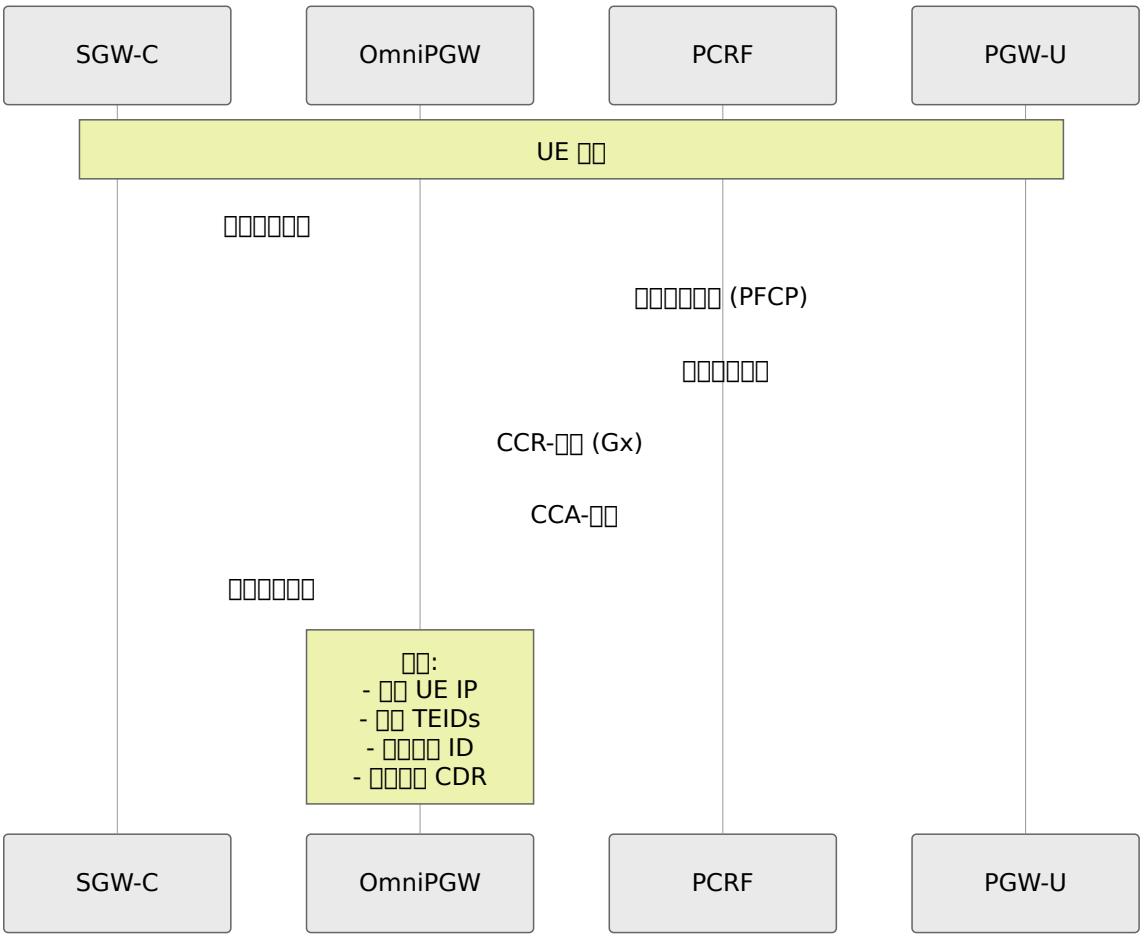


QoS 修改流程

PCRF
OmniPGW
PGW-U
RAR (重新认证请求)
RAA (重新认证响应)
CCR-更新
CCA-更新 (新 QoS)
新内部状态
会话修改请求
使用
新比特率更新 QERs
会话修改响应
新 QoS 活动
PCRF
OmniPGW
PGW-U

□□□□

□□□□

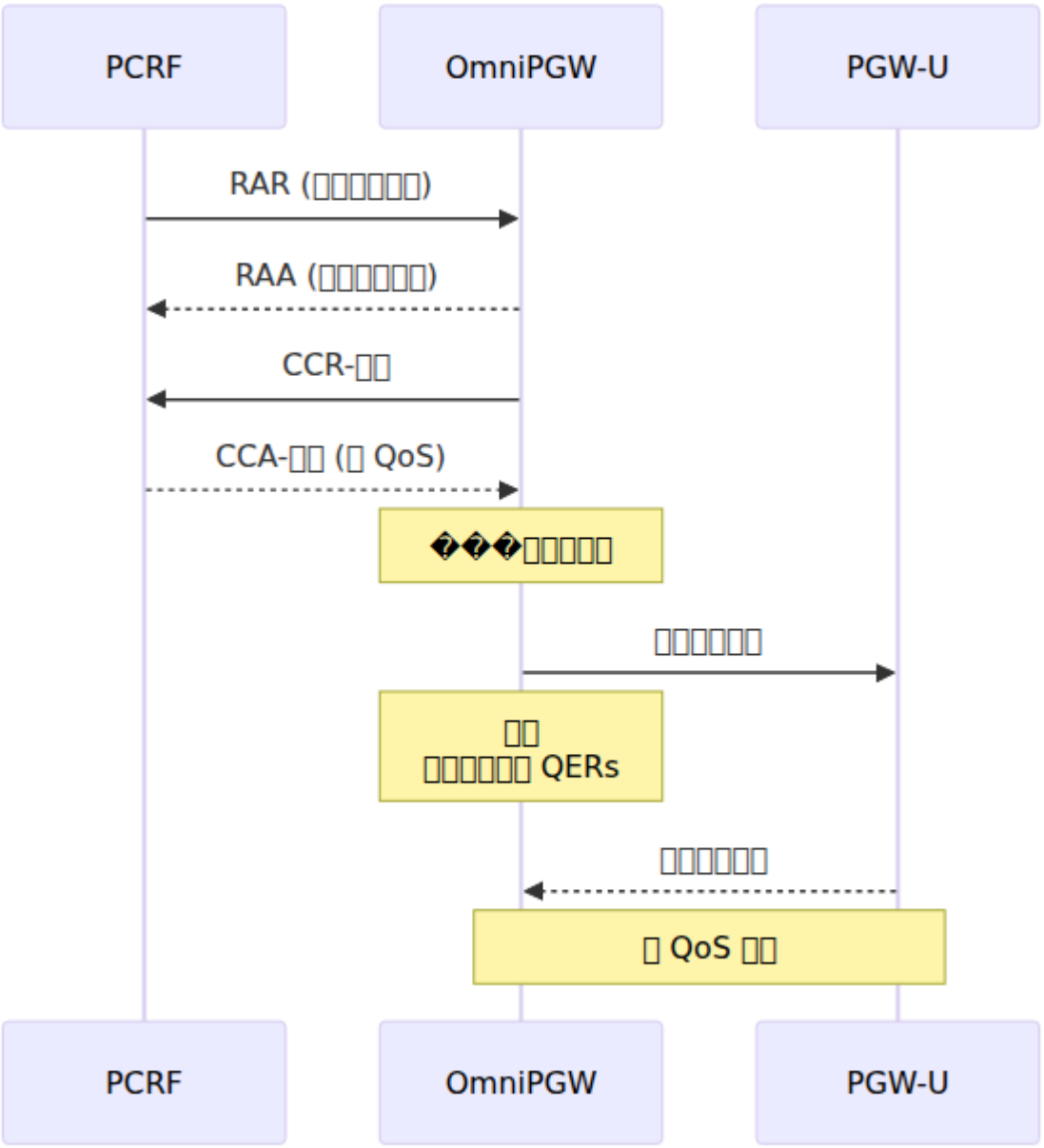


□□□□

□□□□□:

1. UE IP □□ → □□□□
2. TEID → □□□□□□□□
3. SEID → □□□□□□□□
4. □□-ID → □□□□□□□□
5. □□-ID → □□
6. □□□□□□□

□□□□□□□□:



- 为离线计费写入最终 CDR（计费数据记录）- 请参见 数据 CDR 格式

会话状态

状态机

会话跟踪

注册表查找：

```
□ TEID (S5/S8):
  TEID 0x12345678 → □□ PID

□ SEID (Sxb):
  SEID 0xABCDEF → □□ PID

□ □□-ID (Gx):
  "pgw.example.com;123;456" → □□ PID

□ UE IP:
  100.64.1.42 → □□ PID

□ IMSI + EBI:
  "310260123456789" + EBI 5 → □□ PID
```

□□□□

□□□□□□

```
# □□□□□□
teid_registry_count

# PFCP □□
seid_registry_count

# Gx □□
session_id_registry_count
```

会话指标

```
# 会话创建速率
rate(s5s8_inbound_messages_total{message_type="create_session_request

# 会话删除速率
rate(s5s8_inbound_messages_total{message_type="delete_session_request

# 会话创建延迟 (p95)
histogram_quantile(0.95,

rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m])
)
```

常见问题

会话创建失败

原因：

1. **IP** 池耗尽 - 没有可用的 IP
2. **PCRF** 无法访问 - Gx 超时
3. **PGW-U** 停机 - 没有可用的 PFCP 对等体
4. **PCRF** 拒绝 - 用户未知，未授权

调试：

```
# 检查 IP 池
curl http://pgw:9090/metrics | grep address_registry_count

# 检查 PCRF 连接
# 检查日志中的 Gx 错误

# 检查 PGW-U 关联
# 验证 PFCP 对等体状态
```

会话卡住/陈旧

症状：

- 会话未正确删除
- 资源未释放
- 注册表显示的计数高于预期

原因：

1. 未收到删除会话请求
2. 会话进程崩溃未清理
3. 注册表泄漏

解决方案：

```
# 重□ OmniPGW (释放所有会话)
# 实施会话超时机制
```

UE 无法建立会话

症状：

- UE 附加失败
- 创建会话响应带有错误原因

常见原因及响应：

原因值	含义	行动
用户未知	PCRF 拒绝 (IMSI 不在数据库中)	提供用户
无可用资源	IP 池耗尽	扩展 IP 池
远程对等体未响应	PCRF/PGW-U 超时	检查连接
服务不支持	无效 APN	配置 APN 池

最佳实践

会话限制

配置适当的容量：

```
预期并发用户: 10,000
每用户会话开销: ~10KB RAM
会话总 RAM: ~100MB

Erlang VM 设置:
  - 最大进程: 262,144 (默认)
  - 进程堆大小: 根据负载调整
```

会话清理

确保适当的清理：

1. 始终响应删除会话请求
2. 实施会话超时以处理陈旧会话
3. 监控注册表计数以防泄漏

高可用性

会话冗余：

- 使用无状态设计（会话与实例绑定）
- 实施会话数据库以实现 HA（未来）
- DNS/负载均衡器用于故障转移

会话数据元素

会话存储了哪些信息？

每个活动的 PDN 会话维护以下信息：

UE 识别:

IMSI: "310260123456789" (用户身份) MSISDN: "14155551234" (电话号码) MEI/IMEI: 设备标识符

PDN 连接详细信息:

APN: "internet" (网络名称)
UE IP 地址: 100.64.1.42 (分配的 IP)
PDN 类型: IPv4, IPv6, 或 IPv4v6

会话标识符:

计费 ID: 唯一的计费标识符
默认承载 EBI: EPS 承载标识符 (通常为 5)

QoS 参数:

APN-AMBR: 聚合最大比特率
上行: 100 Mbps
下行: 50 Mbps

转发规则:

PDRs (数据包检测规则): 匹配数据包
FARs (转发动作规则): 转发/丢弃动作
QERs (QoS 强制规则): 速率限制
BAR (缓冲动作规则): 下行缓冲

接口上下文:

- S5/S8 状态: 本地/远程 TEIDs, SGW-C 地址
- Sxb 状态: 本地/远程 SEIDs, PGW-U 地址
- Gx 状态: Diameter 会话-ID, 请求计数器

实时会话监控

OmniPGW 提供实时 **OAM REST API** 用于检查和管理活动会话，无需查询指标或日志。API 通过 HTTPS/TLS 在端口 `8443` 上提供，所有路由在 `/api` 前缀下:

```
GET    /api/sessions                    # 所有活动会话
GET    /api/sessions?search=<imsi>      # 按 IMSI / MSISDN / IP 过滤
GET    /api/sessions/<imsi>             # 单个会话的完整详细信息
PUT    /api/sessions/<imsi>             # 更新活动会话状态 (操作员操作)
DELETE /api/sessions/<imsi>             # 拆除会话 (操作员操作)
```

交互式 API 文档 (Swagger UI) 可在 `https://<omnipgw-ip>:8443/api/docs` 访问。

```
# 列出所有活动会话
curl -k https://<omnipgw-ip>:8443/api/sessions

# 按 IMSI, MSISDN 或 IP 地址过滤
curl -k "https://<omnipgw-ip>:8443/api/sessions?
search=310170123456789"
```

UE 搜索与深入分析

端点:

```
GET /api/sessions?search=<imsi|msisdn|ip>    # 查找用户的会话
GET /api/sessions/<imsi>                     # 单个会话的完整详细信息
```

`search` 参数接受以下任意内容:

- **IMSI** (例如, "310170123456789")

- **MSISDN** (电话号码)
- **IP** 地址 (例如, "100.64.1.42")

```
# 深入分析单个用户的会话
curl -k https://<omnipgw-ip>:8443/api/sessions/310170123456789
```

会话详细信息返回全面的信息:

a) 活动会话

- 此用户的所有活动会话
- IMSI, MSISDN, UE IP 地址
- APN, RAT 类型
- PGW TEID, SGW TEID

b) 当前位置信息 来自会话的实时位置信息:

- **TAC** (跟踪区域代码) - UE 所在的跟踪区域
- 小区 **ID (ECI)** - E-UTRAN 小区标识符
- **ECGI** - E-UTRAN 小区全球标识符 (PLMN + ECI)
- **MCC/MNC** - 移动国家代码 / 移动网络代码

c) 承载信息 详细的承载列表及 QoS 参数:

□□□□:

- EBI (EPS 承载标识符)
- QCI (QoS 类别标识符)
- 计费规则名称
- APN-AMBR (上行/下行)

□□□□ (如果活动):

- EBI, QCI, 计费规则名称
- MBR UL/DL (最大比特率)
- GBR UL/DL (保证比特率)

d) 计费信息（Gy 接口）

- Gy 会话 ID
- 授予配额，使用配额
- 计费特性

e) 策略信息（Gx 接口）

- Gx 会话 ID
- PCRF 源/目标主机
- CC 请求编号
- 安装的计费规则（来自承载的 PCC 规则）

f) 最近事件

- 此用户的事件历史（见 会话历史与审计日志）
- 会话创建/更新/删除事件

用例：

- 排查特定用户问题
- 验证会话建立
- 检查分配的 IP 地址
- 检查会话参数

所有活动会话

端点：

```
GET    /api/sessions                    # 列出所有活动 PDN 会话
GET    /api/sessions?search=<term>      # 过滤列表
PUT    /api/sessions/<imsi>             # 更新活动会话状态
DELETE /api/sessions/<imsi>             # 拆除会话
```

目的：列举所有活动 PDN 会话。根据您的首选间隔轮询 `GET /api/sessions` 以实时查看会话表。

```
# 所有活动会话的快照
curl -k https://<omnipgw-ip>:8443/api/sessions

# 按 IMSI 前缀, UE IP, MSISDN 或 APN 过滤
curl -k "https://<omnipgw-ip>:8443/api/sessions?search=100.64"

# 对特定会话的操作员操作
curl -k -X PUT    https://<omnipgw-
ip>:8443/api/sessions/310170123456789   # 更新状态
curl -k -X DELETE https://<omnipgw-
ip>:8443/api/sessions/310170123456789   # 拆除
```

快速会话信息（每个会话返回）：

- **IMSI** - 用户身份
- **UE IP** - 分配的 IP 地址
- **SGW TEID** - 来自 SGW 的 S5/S8 隧道 ID
- **PGW TEID** - 来自 OmniPGW 的 S5/S8 隧道 ID
- **APN** - 接入点名称

过滤（通过 `?search=`）：

- IMSI（例如，"310260"）
- UE IP 地址（例如，"100.64"）
- MSISDN / 电话号码
- APN 名称

完整详细信息（`GET /api/sessions/<imsi>`）：

- 完整的用户信息（IMSI, MSISDN, IMEI）
- 网络上下文（RAT 类型，服务网络 MCC/MNC）
- QoS 参数（AMBR 上行/下行）
- 隧道标识符（以十六进制格式的两个 TEIDs）
- 用于调试的进程 ID
- 完整的会话状态（原始数据结构）

交叉引用的健康端点

OAM API 还公开了会话视图所依赖的对等/拓扑健康：

```
GET /api/upf          # PFCP / UPF 对等体状态和关联
GET /api/diameter     # Diameter (Gx/Gy) 对等体连接
```

```
# 验证用户平面和 Diameter 对等体连接
curl -k https://<omnipgw-ip>:8443/api/upf
curl -k https://<omnipgw-ip>:8443/api/diameter
```

用例：

- 验证对等连接 (UPF 关联, PCRF/OCS 连接)
- 将会话建立失败与对等健康相关联
- 快速网络健康检查

会话历史与审计日志

端点：

```
GET /api/session_history[?type=&search=&page=&page_size=]
```

目的：跟踪历史会话事件和审计轨迹。

参数：

- `type` - 按事件类型过滤 (例如，会话创建，会话删除)
- `search` - 按 IMSI, MSISDN, IP 地址或 TEID 过滤
- `page`, `page_size` - 分页浏览结果

```
# 某用户最近的删除事件，第一页
curl -k "https://<omnipgw-ip>:8443/api/session_history?
type=session_deleted&search=310170123456789&page=1&page_size=50"
```

跟踪的事件类型：

- 会话创建事件
- 会话删除事件
- 修改事件
- 错误事件

用例：

- 合规性审计轨迹（分页浏览并归��结果）
- 历史会话分析
- 排查过去的问题
- 生成使用报告
- 跟踪会话模式随时间变化

作用例

会话验证：

1. 用户报告连接问题
2. GET /api/sessions?search=<imsi|msisdn> 查找用户
3. GET /api/sessions/<imsi> 验证会话存在且 UE 有 IP 地址
4. 检查 QoS 值是否与用户计划匹配
5. 验证隧道端点是否已建立

容量监控：

- 轮询 `GET /api/sessions`（或 `teid_registry_count` 指标）以获取活动会话计数
- 与许可容量进行比较
- 按 APN 识别使用模式（通过 `?search=<apn>`）

故障排除：

- 通过任何标识符查找特定会话，使用 `?search=`
- 检查完整的会话状态（`GET /api/sessions/<imsi>`）无需 SSH/IEx
- 验证 SGW 和 PGW TEIDs 在系统之间是否匹配
- 检查从 PCRF 应用的 AMBR 值

- 使用 `DELETE /api/sessions/<imsi>` 拆除卡住的会话

相较于指标的优势：

- 查看单个会话详细信息（指标显示汇总）
- 通过 `?search=` 进行搜索和过滤
- 实时状态检查
- 可脚本化/自动化（通过 HTTPS 的 JSON）以集成工具

相关文档

核心会话功能

- **PFCP** 接口 - 用户平面会话建立，PDRs，FARs，QERs，URRs
- **UE IP** 分配 - IP 地址分配，APN 池管理
- **PCO** 配置 - 传递给 UE 的 DNS，P-CSCF，MTU 参数
- 配置指南 - UPF 选择，会话建立流程

策略和计费

- **Diameter Gx** 接口 - PCRF 策略控制，PCC 规则，QoS 管理
- **Diameter Gy** 接口 - OCS 在线计费，配额跟踪
- 数据 **CDR** 格式 - 离线计费记录生成

网络接口

- **S5/S8** 接口 - GTP-C 协议，SGW-C 通信
- **QoS** 和承载管理 - 承载 QoS 强制

□作

- 监控指南 - 会话指标，活动会话跟踪，警报
- **P-CSCF** 监控 - IMS 会话监控

□□□□□□

OmniPGW □□□□ - *□ Omnitouch □□□□□□*

OmniPGW 故障排除指南

故障排除程序和常见问题

□ Omnitouch □□□□□□

目录

1. □述
2. 故障排除工具
3. 会话建立问题
4. PFCP / 用户平面问题
5. Diameter (Gx/Gy) 问题
6. IP 分配问题
7. 快速参考

概述

本指南提供了针对常见 OmniPGW 操作问题的逐步故障排除程序。每个问题包括：

- 症状：您将观察到的情况
- 可能原因：常见根本原因
- 诊断：如何确认原因
- 解决方案：逐步修复
- 预防：如何避免再次发生

相关文档

- 监控指南 - Prometheus 指标、警报、性能监控
- 配置指南 - 系统配置参考

故障排除工具

OAM API

基本 **URL:** `https://<omnipgw_ip>:8443` (HTTPS/TLS; 所有路由在 `/api` ��缀下)

交互式 **API** 文档 **(Swagger UI):** `https://<omnipgw_ip>:8443/api/docs`

关键端点:

- `GET /api/sessions[?search=<imsi|msisdn|ip>]`, `GET /api/sessions/<imsi>` - 活动会话查看器（按 IMSI、IP、MSISDN 搜索）
- `GET /api/diameter[?status=connected|disconnected]`, `GET /api/diameter/<origin-host>` - Diameter 对等状态 (Gx PCRF, Gy OCS)
- `GET /api/charging[?search=<imsi>]`, `GET /api/charging/<imsi>` - 在线计费 (Gy) 状态
- `GET /api/upf`, `GET /api/upf/<ip>` - UPF / PFCP 对等状态 (PGW-U 连接性)
- `GET /api/ip_pools`, `GET /api/ip_pools/<name>` - IP 池利用率
- `GET /api/session_history[?type=&search=&page=&page_size=]` - 会话历史
- `GET /api/status` - 健康检查 (`{"result":"ok"}`)

示例:

```
curl -k https://<omnipgw_ip>:8443/api/sessions
```

日志流不再由平台提供；请使用主机日志 (`journalctl`/日志文件) 和以下过滤器。

Prometheus 指标

访问**:** `http://<omnipgw_ip>:9090/metrics`

关键指标:

- `teid_registry_count` - 活动会话
- `address_registry_count` - 分配的 UE IP
- `sxb_inbound_errors_total` - PFCP 错误

- `gx_inbound_errors_total` - Diameter Gx 错误
- `gy_inbound_errors_total` - Diameter Gy 错误

请参见 监控指南 以获取完整的指标参考。

日志���析

日志：检查主机日志 (`journalctl -u omnipgw_c` 或配置的日志文件) 并 grep 以下字符串。要获取会话生命周期事件的结构化、可查询视图，请使用 `GET /api/session_history[?type=&search=&page=&page_size=]`。

常见日志过滤器：

- "create_session_request" - 会话建立
- "Credit Control" - Gx/Gy 交互
- "PFCP Session" - 用户平面编程
- "error" 或 "ERROR" - 错误消息
- "timeout" - 超时问题

会话建立问题

问题：创建会话请求被拒绝，原因是 "没有可用资源"

症状：

- SGW-C 收到创建会话响应，原因是 "没有可用资源" (73)
- 所有新的会话尝试失败
- 现有会话继续工作
- 日志：`[PGW-C] 创建会话请求被阻止 - 无效许可证`

Wireshark □□□□□□□□□□□□□□□ "□□□□□□□"

可能原因:

- 无效或过期的 OmniPGW 许可证
- 许可证服务器无法访问

诊断:

1. 检查许可证指标:

   ```
   license_status
   ```

   - 值为 0 表示许可证无效

2. 检查日志中的许可证警告:

   - 搜索 "license" 或 "License"
   - 查找 "无法联系许可证服务器" 消息

3. 验���许可证服务器���接性:

   - 检查 `config/runtime.exs` 中 `:license_client` 下的配置 URL
   - 默认: `https://localhost:10443/api`

解决方案:

1. 验证许可证服务器是否可达：

```
curl -k https://<license_server_ip>:10443/api/status
```

2. 检查 `config/runtime.exs` 中的许可证配置：

```
config :license_client,
  license_server_api_urls:
["https://<license_server_ip>:10443/api"],
  licensee: "您的公司名称"
```

3. 验证产品是否已获得许可：
   - 产品名称：`omnipgwc`
   - 联系 Omnitouch 验证许可证状态
4. 在配置更改后重启 **OmniPGW**

预防：

- 监控 `license_status` 指标并设置关键警报
- 确保许可证服务器的高可用性
- 在许可证到期前设置许可证到期警报

问题：创建会话请求被拒绝（其他原因）

症状：

- SGW-C 收到创建会话响应，带有错误原因
- 用户无法建立 PDN 连接
- 指标：`s5s8_inbound_errors_total` 增加

可能原因：

1. IP 池耗尽
2. PCRF (Gx) 无法访问或拒绝策略

3. PGW-U (PFCP) 不可用
4. 无效的 APN 配置

诊断：

1. 检查 IP 池利用率：

address_registry_count

如果等于配置的池大小，则池已耗尽

2. 检查 PCRF 连接性：

查询 GET /api/diameter (或 GET /api/diameter?status=disconnected)
查找 PCRF 对等状态 = "disconnected"
日志：搜索 "Credit Control Answer" 以查找错误

3. 检查 PFCP 对等状态：

查询 GET /api/upf (或GET /api/upf/<ip>针对特定 PGW-U)
查找 "Association: DOWN"
指标：pfcp_peer_associated = 0

4. 检查 APN 配置：

检查 config/runtime.exs 中的 ue.apn_map
验证请求的 APN 是否存在于配置中

解决方案：

对于 IP 池耗尽：

1. 确定过期会话：查询 GET /api/sessions (可选 GET /api/sessions?search= <imsi|msisdn|ip>)，查找旧会话
2. 在 config/runtime.exs 中扩展 IP 池：

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => "10.0.0.0/23"  # 从 /24 更改为 /23（容量翻倍）
    }
  }
```

3. 重启 OmniPGW
4. 验证：`curl http://<ip>:9090/metrics | grep address_registry_count`

对于 **PCRF** 连接性问题：

1. 检查网络连接性：`ping <pcrf_ip>`
2. 验证 PCRF Diameter 服务：`telnet <pcrf_ip> 3868`
3. 检查 `config/runtime.exs` 中的 Diameter 对等配置
4. 如果配置更改，重启 OmniPGW
5. 通过 `GET /api/diameter?status=connected` 验证（对等应显示 "connected"）

对于 **PFCP** 问题：

- 请参见 PFCP / 用户平面问题 部分

预防：

- 监控 IP 池利用率，并在 80% 时发出警报
- 监控 PCRF 连接性，并设置 Diameter 对等警报
- 实施闲置会话的清理

问题：会话卡在中间状态

症状：

- 会话出现在 `GET /api/sessions` 中，但不完整
- 指标显示会话数量增加，但没有用户流量
- 删除会话请求失败或超时

可能原因：

1. PFCP 会话建立失败，但 S5/S8 会话已创建
2. PCRF CCR-Initial 超时
3. 创建承载请求（专用承载）失败
4. 会话设置期间的网络中断

诊断：

1. 通过 **OAM API** 搜索会话：
   - `GET /api/sessions/<imsi>` (或 `GET /api/sessions?search=<imsi>`)
   - 检查 `pfcp_seid` 是否存在（如果缺失，则 PFCP 失败）
   - 检查 `gx_session_id` 是否存在（如果缺失，则 Gx 失败）
2. 检查日志中的 **IMSI**：
   - 按 IMSI 过滤日志
   - 查找 "会话建立请求"（PFCP）
   - 查找 "信用控制请求"（Gx）
   - 查找超时或错误消息
3. 检查指标：

```
# 有 TEID 但没有 PFCP 会话的会话
teid_registry_count - seid_registry_count

# 有 TEID 但没有 Gx 会话的会话
teid_registry_count - session_id_registry_count
```

解决方案：

1. 对于 **PFCP** 建立失败：
   - 检查 PGW-U 健康和日志
   - 验证 PFCP 关联：`GET /api/upf`
   - 从 SGW-C 发送删除会话请求以进行清理
2. 对于 **Gx** 超时问题：

检查 PCRF 延迟：histogram_quantile(0.95, rate(gx_outbound_transaction_duration_bucket[5m]))
如果需要，增加 config/runtime.exs 中的 Gx 超时
发送删除会话请求以进行清理

3. 手动清理（最后手段）：

当前需要重启 OmniPGW 以清除卡住的会话
在重启前后监控 teid_registry_count 以确认清理

预防：

监控 PFCP 和 Gx 延迟指标
对于不完整的会话实施会话超时/清理
对注册计数不匹配发出警报

PFCP / 用户平面问题

问题：PFCP 关联失败

症状：

GET /api/upf 显示 "Association: DOWN"
所有新的会话建立失败
指标：pfcp_peer_associated = 0
日志："PFCP 心跳超时" 或 "关联设置失败"

可能原因：

1. PGW-U 无法访问（网络问题）
2. PGW-U 崩溃或重启
3. PFCP 配置不匹配（IP、端口）
4. 防火墙阻止 UDP 8805

诊断：

1. 检查网络连接性：

```
ping <pgw_u_ip>
nc -u -v <pgw_u_ip> 8805
```

2. 检查 **PFCP** 配置：

   - 检查 `config/runtime.exs` 中的 `upf.peer_list`
   - 验证 IP 地址和节点 ID 是否与 PGW-U 配置匹配

3. 检查 **PGW-U** 状态：

   - 访问 PGW-U 日志
   - 验证 PGW-U 是否正在运行：`systemctl status omnipgw_u`（或等效命令）

4. 检查指标：

```
# 心跳失败
pfcp_consecutive_heartbeat_failures

# PFCP 错误率
rate(sxb_inbound_errors_total[5m])
```

解决方案：

1. 对于网络问题：

   - 验证路由：`traceroute <pgw_u_ip>`
   - 检查防火墙规则：确保允许 UDP 8805
   - 检查安全组（如果是云部署）

2. 对于 **PGW-U** 崩溃：

   - 重启 PGW-U 服务
   - 等待 30 秒以重新建立关联
   - 通过 `GET /api/upf` 验证（应显示 "Association: UP"）

3. 对于配置问题：

更正 config/runtime.exs 中的 PFCP 对等配置
重启 OmniPGW
验证关联是否建立

预防：

监控 pfcp_peer_associated 指标并设置关键警报
监控 pfcp_consecutive_heartbeat_failures（当 > 2 时发出警报）
实施冗余 PGW-U 实例
启用 PFCP 保持活动/心跳（应为默认设置）

问题：PFCP 会话修改失败

症状：

专用承载创建失败
QoS 策略更新（来自 PCRF RAR）失败
日志："会话修改请求失败"
指标：
sxb_inbound_errors_total{message_type="session_modification_respon se"} 增加

可能原因：

1. 无效的 PFCP 规则（PDR/FAR/QER 引用）
2. PGW-U 资源耗尽
3. 规则 ID 冲突
4. PGW-U 软件错误

诊断：

1. 检查日志：

过滤 "会话修改" 和 SEID
查找 PFCP 响应中的错误原因代码
常见原因："规则 ID 已存在"，"资源耗尽"

2. 检查 **PGW-U** 日志：
    - 查找 PFCP 处理错误
    - 检查资源利用率（CPU、内存）
3. 通过 **OAM API** 检查会话状态：
    - `GET /api/sessions/<imsi>`（或 `GET /api/sessions?search=<imsi>`）
    - 检查 `pdr_map`、`far_map`、`qer_map` 是否存在冲突
    - 查找重复的 ID

解决方案：

1. 对于规则冲突：
    - 删除并重新创建专用承载
    - 如果持续存在，删除会话并让 UE 重新连接
2. 对于 **PGW-U** 资源问题：
    - 检查 PGW-U 容量（会话、PDR、吞吐量）
    - 如有需要，扩展 PGW-U
    - 减少受影响 PGW-U 实例的会话负载
3. 对于软件错误：
    - 捕获完整会话状态（`GET /api/sessions/<imsi>`）
    - 捕获 PFCP 消息日志
    - 向供应商报告并提供重现步骤

预防：

- 监控 PGW-U 资源利用率
- 在暂存环境中测试���用承载创建
- 监控 `sxb_inbound_errors_total` 并设置警报

Diameter (Gx/Gy) 问题

问题：PCRF 对等断开（Gx）

症状：

- `GET /api/diameter`（或 `GET /api/diameter?status=disconnected`）显示 PCRF 对等 "disconnected"
- 创建的会话没有 QoS 策略（应用默认 QCI=5）
- 日志："Diameter 对等连接失败" 或 "CER/CEA 超时"

可能原因：

1. PCRF 无法访问（网络问题）
2. PCRF 服务宕机
3. Diameter 配置不匹配（Origin-Host、Realm）
4. 防火墙阻止 TCP 3868

诊断：

1. 检查网络连接性：

   ```
   ping <pcrf_ip>
   telnet <pcrf_ip> 3868
   ```

2. 检查 **Diameter** 配置：

   - 检查 `config/runtime.exs` 中的 `diameter.peer_list`
   - 验证 `host`、`realm`、`ip` 是否与 PCRF 配置匹配
   - 检查 `origin_host` 是否与 PCRF 预期匹配

3. 检查 **PCRF** 日志：

   - 查找来自 PGW-C 的 CER（能力交换请求）
   - 查找拒绝原因

4. 检查指标：

```
# Diameter 连接错误
diameter_peer_connected{peer="<pcrf_host>"}
```

解决方案:

1. 对于网络问题:
   - 验证到 PCRF 的路由
   - 检查防火墙规则: 确保允许 TCP 3868
   - 测试连接性: `nc -v <pcrf_ip> 3868`
2. 对于 **PCRF** 服务宕机:
   - 重启 PCRF 服务
   - 等待自动重新连接 (30 秒重试间隔)
   - 通过 `GET /api/diameter?status=connected` 验证
3. 对于配置不匹配:
   - 更正 `config/runtime.exs` Diameter 配置:

```
config :pgw_c,
  diameter: %{
    host: "pgw-c.epc.mnc999.mcc999.3gppnetwork.org",  # 必须与 PCRF 配置匹配
    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
      }
    ]
  }
```

   - 重启 OmniPGW
   - 验证连接是否建立

预防:

- 监控 Diameter 对等连接性并设置关键警报
- 实施冗余 PCRF 实例（如果支持）
- 在运行手册中记录 Diameter 配置

问题：**CCR/CCA** 超时（**Gx** 策略请求）

症状：

- 会话建立缓慢（> 5 秒）
- 日志："信用控制请求超时"
- 指标：`gx_outbound_transaction_duration` 非常高（> 5s）
- 创建的会话应用默认 QoS（回退行为）

可能原因：

1. PCRF 过载
2. PCRF 数据库缓慢
3. 网络延迟
4. PCRF 软件问题

诊断：

1. 检查 **Gx** 延迟：

   ```
   # P95 延迟
   histogram_quantile(0.95,
   rate(gx_outbound_transaction_duration_bucket[5m]))

   # P99 延迟（异常值）
   histogram_quantile(0.99,
   rate(gx_outbound_transaction_duration_bucket[5m]))
   ```

2. 检查 **PCRF** 健康：

   - 访问 PCRF 监控仪表板
   - 检查 CPU、内存、数据库连接
   - 查看 PCRF 日志以查找慢查询

3. 检查网络延迟：

```
ping -c 100 <pcrf_ip> | tail -1  # 检查平均延迟
```

4. 检查日志：

   - 计算 CCR/CCA 交换次数：过滤 "信用控制"
   - 测量 "发送 CCR" 和 "收到 CCA" 之间的时间

解决方案：

1. 对于 **PCRF** 过载：

   - 扩展 PCRF（添加实例）
   - 如果可能，减少 CCR 消息大小
   - 调整 PCRF 线程池/工作线程

2. 对于网络延迟：

   - 调查网络路径（路由器、交换机）
   - 考虑将 PGW-C 和 PCRF 共同放置

3. 临时解决方法（增加超时）：

   - 编辑 `config/runtime.exs`：

     ```
     config :pgw_c,
       diameter: %{
         transaction_timeout_ms: 10000  # 从默认 5000 增加
       }
     ```

   - 重启 OmniPGW
   - 注意：这只是掩盖问题；修复根本原因

预防：

- 监控 Gx 延迟并设置警报（警告 > 1s，关键 > 5s）
- 为预期的会话速率进行 PCRF 容量规划
- 在负载下测试 PCRF 性能

问题：OCS 对等断开（Gy）

症状：

GET /api/diameter (或 GET /api/diameter?status=disconnected) 显示 OCS
对等 "disconnected"；GET /api/charging 反映在线计费 (Gy) 状态
会话无法计费（在线计费失败）
日志："Gy 对等连接失败"

诊断和解决方案：

类似于 PCRF 对等断开，但针对 Gy 接口。

关键区别：

端口：通常为 TCP 3868（与 Gx 相同）
影响：计费失败，可能拒绝会话或在没有计费的情况下允许会话（取决于配置）
配置：检查 diameter.peer_list 中的 OCS 条目

参见：Diameter Gy 接口 以获取 Gy 特定的故障排除

IP 分配问题

问题：IP 池耗尽

症状：

创建会话请求被拒绝，原因是 "没有可用资源"
指标：address_registry_count 等于配置的池大小
GET /api/sessions 显示许多活动会话；GET /api/ip_pools 显示池已满日志："IP 分配失败：池耗尽"

可能原因：

1. 池对用户基数过小

2. 会话未释放 IP（删除会话失败）
3. 快速会话更替而未清理
4. IP 地址泄漏

诊断：

1. 检查池利用率：

```
# 对于 /24 子网（254 个 IP）
(address_registry_count / 254) * 100
```

2. 检查配置的池大小：
   - 检查 `config/runtime.exs` 中的 `ue.subnet_map`
   - 示例："10.0.0.0/24" = 254 可用 IP
3. 比较会话计数与 **IP** 计数：

```
# 应大致相等
teid_registry_count
address_registry_count
```

4. 查看活动会话：
   - `GET /api/sessions`（和 `GET /api/ip_pools/<name>` 以获取每池利用率）
   - 按会话开始时间排序
   - 查找非常旧的会话（潜在泄漏）

解决方案：

立即（扩展池）：

1. 编辑 `config/runtime.exs`（每个 APN 映射到一列表 的 CIDR 字符串）：

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => ["10.0.0.0/22"]  # 1022 个 IP（之前为 /24 = 254 个 IP）
    }
  }
```

2. 重启 OmniPGW
3. 验证：会话现在可以建立

长期（清理）：

1. 通过 `GET /api/sessions` 确定过期会话
2. 与 SGW-C 协调发送删除会话请求
3. 在 PCRF/SGW 上实施会话超时策略
4. 监控 `address_registry_count` 以验证清理后池是否释放

预防：

- 监控 IP 池利用率并设置警报：
  - 警告：> 70%
  - 关键：> 85%
- 趋势分析以预测耗尽
- 实施会话闲置超时
- 定期会话审计

问题：分配重复的 IP 地址

症状：

- UE 报告 IP 地址冲突
- 日志："IP 已分配" 警告
- 在 `GET /api/sessions` 中有两个会话具有相同的 IP 地址

可能原因：

1. 软件错误（罕见）
2. 崩溃后的数据库不一致
3. 手动干预错误

诊断：

1. 通过 OAM API 搜索 IP：

GET /api/sessions?search=<ip>
检查是否多个 IMSI 拥有相同的 IP

2. 检查日志：

搜索 IP 地址
查找 "IP 分配" 事件

解决方案：

1. 确定受影响的会话：

注意两个 IMSI 的重复 IP

2. 删除一个会话：

与 SGW-C 协调发送删除会话请求以删除一个 IMSI
优先删除较新的会话

3. UE 重新连接：

UE 应自动重新连接
将收到新的唯一 IP

4. 如果持续存在：

重启 OmniPGW 以重建 IP 注册表
所有会话将丢失（协调维护窗口）

预防：

监控重复分配（目前没有内置指标）
定期进行数据库完整性检查（如果适用）

问题：在没有 IPv6 池的 APN 上的 IPv4v6 会话

症状：

UE 请求 IPv4v6 PDN，但匹配的 APN 仅在 ue.subnet_map 中配置了 IPv4 CIDR
一个 紧急 (SOS) APN 以 IPv4-only 方式出现（PAA 中没有 IPv6）；日志显示警告 "紧急 APN 没有 IPv6 池...；授予 IPv4-only"
一个 常规 APN 会话 被拒绝；日志显示错误 "APN 没有 IPv6 池...；拒绝 IPv4v6 请求"

可能原因：

1. APN 故意没有 IPv6 范围（在 IPv4-only 部署中常见）
2. APN 的 ue.subnet_map 条目中遗漏了 IPv6 CIDR

行为：

IPv6 分配器报告没有 IPv6 池（它不会在空池上崩溃）。
紧急 (SOS) APN（名称包含 "sos"）降级为 IPv4-only — 保留 IPv4 地址，清除 IPv6，PDN 类型设置为 IPv4，并且 P-CSCF 仍在 PCO 中返回，因此紧急 IMS 注册工作。
常规 APN 被拒绝（没有静默降级），显示不支持的请求/配置错误。

解决方案：

1. 如果 APN 应提供双栈，请在其 ue.subnet_map 条目中添加 IPv6 CIDR 并重启 OmniPGW。
2. 如果 IPv4-only 行为是有意的（例如在 IPv4-only 网络上的 SOS APN），则无需采取任何措施。

请参见 APN 没有 IPv6 池时的 IPv4v6 请求 和 UE IP 分配 → 问题 4。

快速参考

常见 **Prometheus** 查询

```
# 活动会话
teid_registry_count

# 会话设置速率（每秒）
rate(s5s8_inbound_messages_total{message_type="create_session_request

# IP 池利用率（对于 /24 子网）
(address_registry_count / 254) * 100

# P95 会话设置延迟
histogram_quantile(0.95,
rate(s5s8_inbound_handling_duration_bucket{request_message_type="crea
[5m]))

# 错误率
rate(s5s8_inbound_errors_total[5m])

# PCRF 延迟
histogram_quantile(0.95, rate(gx_outbound_transaction_duration_bucket

# PFCP 关联状态
pfcp_peer_associated
```

常见日志过滤器

在主机日志中 grep 这些字符串（`journalctl -u omnipgw_c` 或配置的日志文件）。对于会话生命周期事件，`GET /api/session_history[?type=&search=]` 提供相同的信息以结构化形式。

过滤器	目的
IMSI	查找特定用户的所有日志
"create_session"	会话建立流程
"delete_session"	会话拆除流程
"Credit Control"	Gx PCRF 交互
"PFCP Session"	用户平面编程
"error"	所有错误消息
"timeout"	超时问题
"Association"	PFCP 关联事件

健康检查命令

```
# 检查服务状态
systemctl status omnipgw_c

# 检查 OAM API 健康 (HTTPS/TLS)
curl -k https://<omnipgw_ip>:8443/api/status

# 检查指标端点
curl http://<omnipgw_ip>:9090/metrics

# 检查活动会
curl http://<omnipgw_ip>:9090/metrics | grep teid_registry_count

# 检查 PFCP 关联
curl http://<omnipgw_ip>:9090/metrics | grep pfcp_peer_associated

# 检查 IP 池使用情况
curl http://<omnipgw_ip>:9090/metrics | grep
address_registry_count
```

相关文档

- 监控指南 - Prometheus 指标、Grafana 仪表板、警报
- 配置指南 - 系统配置参考
- 会话管理 - 会话生命周期详细信息
- **PFCP** 接口 - PFCP 故障排除详细信息
- **Diameter Gx** 接口 - Gx 策略故障排除
- **Diameter Gy** 接口 - Gy 计费故障排除
- **QoS** 和承载管理 - 与 QoS 相关的问题

返回□作指南

UE IP池分配文档

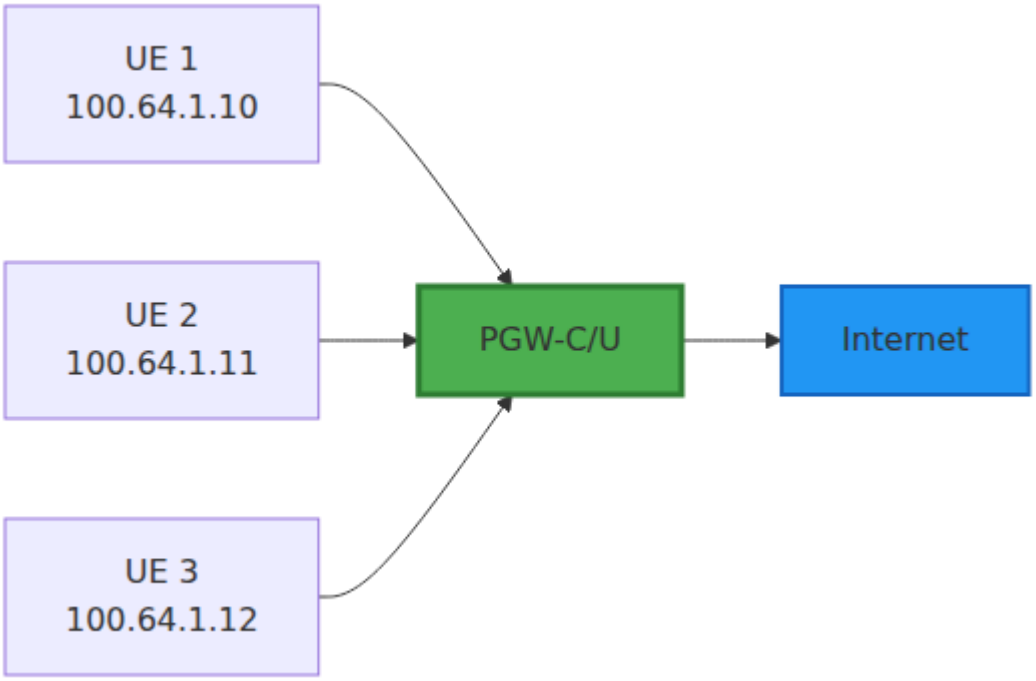
移动设备的IP地址管理

目录

1. □述
2. IP分配□念
3. 配置
4. 分配过程
5. 高级主题
6. 监控
7. 故障排除

概述

PGW-C在UE（用户设备）建立PDN（分组数据网络）连接时分配IP地址。这是一个关键功能，使移动设备能够与外部网络通信。



为什么IP分配很重要

每个UE从PGW-C接收一个唯一的IP地址，该地址：

- 在网络上标识设备
- 路由设备的流量
- 启用计费和策略执行
- 在PDN连接期间保持有效

支持的IP版本

IP版本	支持	描述
IPv4	是	标准IPv4地址
IPv6	是	IPv6地址和前缀
IPv4v6	是	双栈（IPv4和IPv6）

IP分配概念

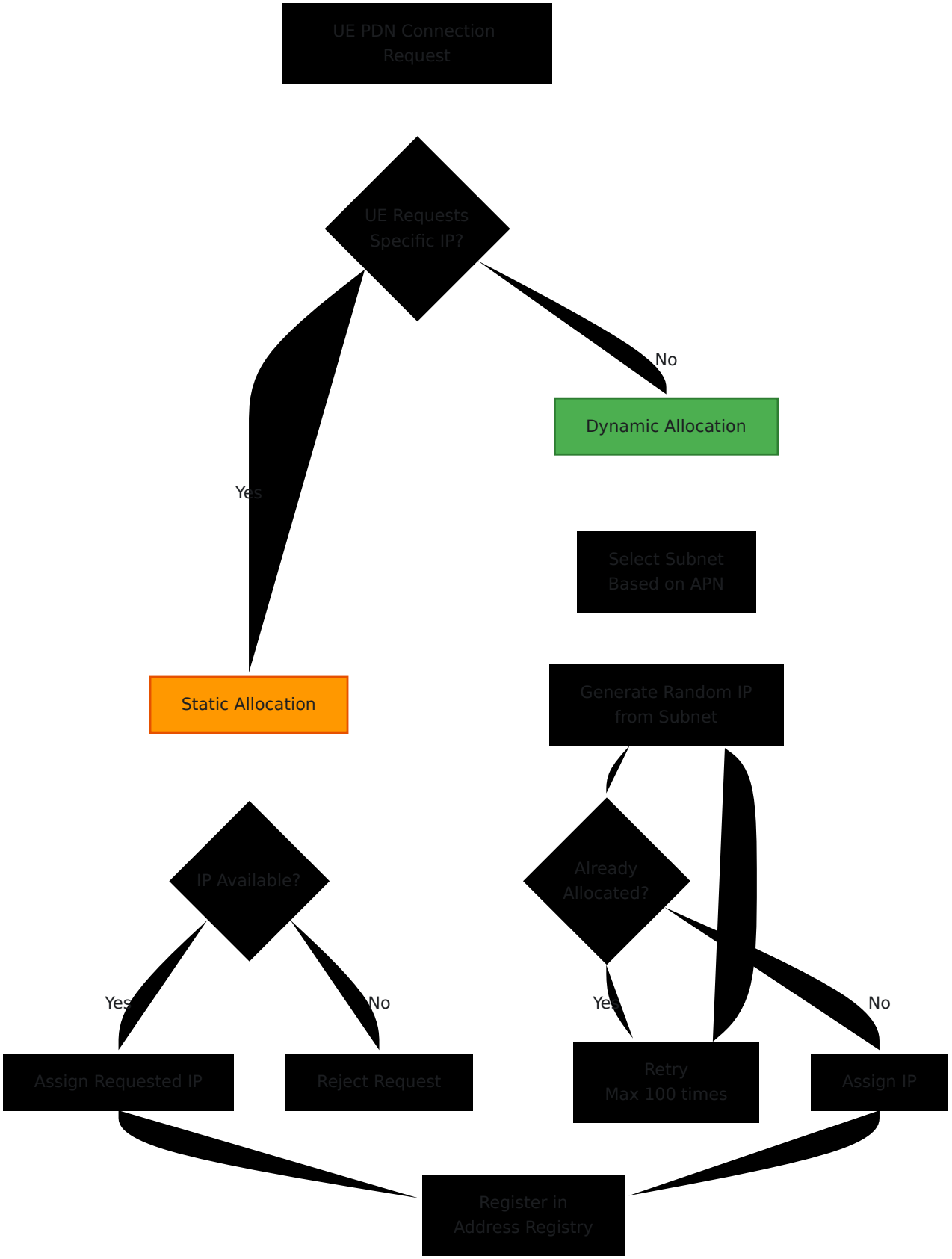
PDN类型

当UE请求PDN连接时，它指定一个**PDN**类型：

PDN类型	描述	分配的地址
IPv4	仅IPv4连接	单个IPv4地址
IPv6	仅IPv6连接	IPv6前缀（例如，/64）
IPv4v6	双栈连接	IPv4地址和IPv6前缀

分配方法

PGW-C支持两种IP分配方法：



1. 动态分配（最常见）：

- PGW-C从配置的池中选择IP
- 随机选择以避免可预测性

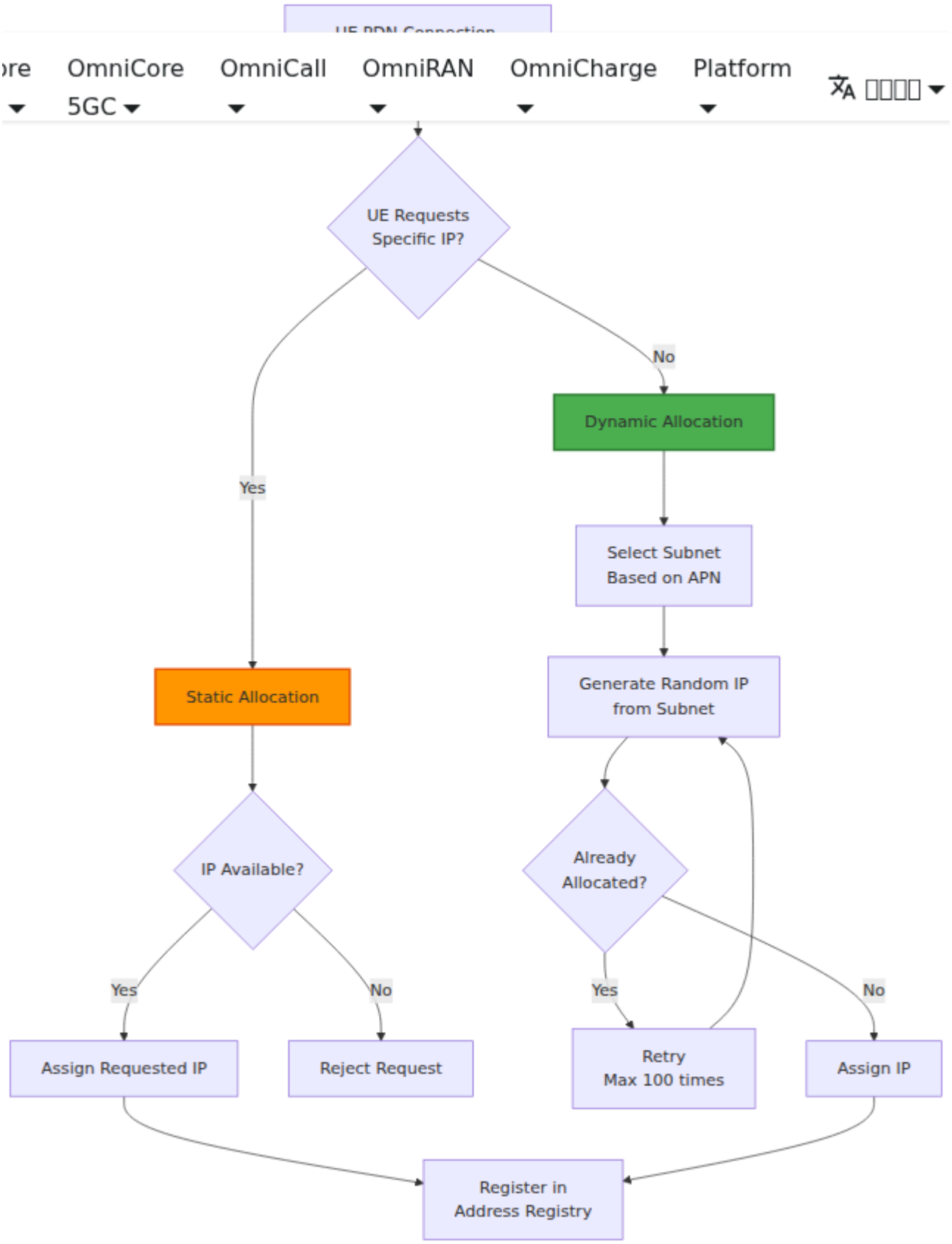
碰撞检测确保唯一性

2. 静态分配：

UE在GTP-C消息中请求特定IP
PGW-C验证可用性
对于具有固定IP的企业设备很有用

基于APN的子网选择

不同的**APN（接入点名称）**可以使用不同的IP池：



好处:

- 流量隔离 - 不同APN路由到不同网络
- 策略差异化 - 每个APN应用不同策略

- 容量规划 - 根据预期使用情况调整池的大小
- 计费 - 按服务类型跟踪使用情况

地址注册表

地址注册表跟踪已分配的IP:

功能	描述
注册	映射UE IP → 会话进程PID
查找	根据UE IP查找会话
注销	会话结束时释放IP
碰撞检测	防止重复分配

配置

基本配置

编辑`config/runtime.exs`:

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # APN "internet"使用两个子网
      "internet" => [
        "100.64.1.0/24",    # 254个可用IP
        "100.64.2.0/24"     # 254个可用IP
      ],

      # APN "ims"使用一个子网
      "ims" => [
        "100.64.10.0/24"
      ],

      # 未知APN的默认池
      default: [
        "42.42.42.0/24"
      ]
    }
  }
```

APN的正则表达式模式匹配

对于多个APN共享相同子网池的情况，可以使用正则表达式模式而不是精确的APN名称。这对于通配符APN匹配很有用。

模式规则：

- 以`^`开头的键被视为正则表达式模式
- 没有`^`的键被精确匹配（向后兼容）
- 按顺序评估模式 - 第一个匹配的胜出
- 如果没有模式匹配，则回退到`default`

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # 正则表达式：以"ims"开头的APN（例
如，"ims"，"ims.apn"，"ims.something.else"）
      "^ims" => [
        "100.64.10.0/24"
      ],

      # 正则表达式：以"m2m."开头的APN（例如，"m2m.test"，"m2m.prod"）
      "^m2m\." => [
        "100.64.20.0/24"
      ],

      # 仅精确匹配 - "enterprise.corp"精确匹配
      "enterprise.corp" => [
        "10.100.0.0/16"
      ],

      # 未匹配APN的默认池
      default: [
        "42.42.42.0/24"
      ]
    }
  }
```

重要说明：

- 正则表达式模式使用标准Elixir/Erlang正则表达式语法
- 在Elixir字符串中，反斜杠必须转义（使用`\\`表示`\`）
- 开头的`^`是必需的，以指示正则表达式模式
- 精确匹配和正则表达式模式可以在同一配置中混合使用
- 正则表达式模式的顺序很重要 - 更具体的模式应放在前面

常见模式示例：

模式类型	正则键	匹配
以...开头	`"^ims"`	`ims`，`ims.apn`，`ims.anything`
以...结尾	`"^.*\.corp$"`	`foo.corp`，`bar.corp`
包含	`"^.*test.*"`	`test`，`foo.test.bar`，`testing`
精确（带点）	`"^internet\.apn$"`	仅`internet.apn`

后缀匹配示例：

要匹配以特定后缀（如`.corp`）结尾的APN，请使用`^.*\.suffix$`：

```
subnet_map: %{
  # 匹配以".corp"结尾的APN
  "^.*\.corp$" => ["10.100.0.0/16"],

  # 匹配以".iot"结尾的APN
  "^.*\.iot$" => ["10.200.0.0/16"],

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

示例模式匹配：

APN请求	匹配的键	使用的池
`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

子网表示法

CIDR表示法: `<network>/<prefix_length>`

CIDR	可用**IP**	示例范围
/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

注意:

- 网络地址（例如，100.64.1.0）不被分配
- 广播地址（例如，100.64.1.255）不被分配
- PGW-C从`<network> + 1`到`<broadcast> - 1`进行分配

每个**APN**多个子网

跨子网负载均衡：

```
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"
      ]
    }
  }
```

选择方法：

- PGW-C从列表中随机选择一个子网
- 提供基本的负载均衡
- 每个会话独立选择一个子网

好处：

- 在多个子网之间分配负载
- 更容易扩展容量（添加新子网）
- 路由策略的灵活性

现实世界示例

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # 一般互联网访问
      "internet" => [
        "100.64.0.0/20"     # 4094个IP供一般使用
      ],

      # IMS（LTE语音）
      "ims" => [
        "100.64.16.0/22"    # 1022个IMS IP
      ],

      # 企业APN
      "enterprise.corp" => [
        "10.100.0.0/16"     # 65534个企业IP
      ],

      # IoT设备（低比特率）
      "iot.m2m" => [
        "100.64.20.0/22"    # 1022个IoT IP
      ],

      # 默认回退
      default: [
        "42.42.42.0/24"     # 254个未知APN的IP
      ]
    }
  }
```

IPv6配置

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => [
        # IPv4池
        "100.64.1.0/24"
      ],
      "internet.ipv6" => [
        # IPv6池（前缀委派）
        "2001:db8:1::/48"
      ],
      default: [
        "42.42.42.0/24"
      ]
    }
  }
```

IPv6前缀委派:

- UE通常接收一个/64前缀
- 允许UE分配多个IP（例如，用于共享网络）
- 示例: UE接收`2001:db8:1:a::/64`

双栈（**IPv4v6**）配置

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => [
        "100.64.1.0/24",        # IPv4池
        "2001:db8:1::/48"       # IPv6池（将用于IPv6分配）
      ]
    }
  }
```

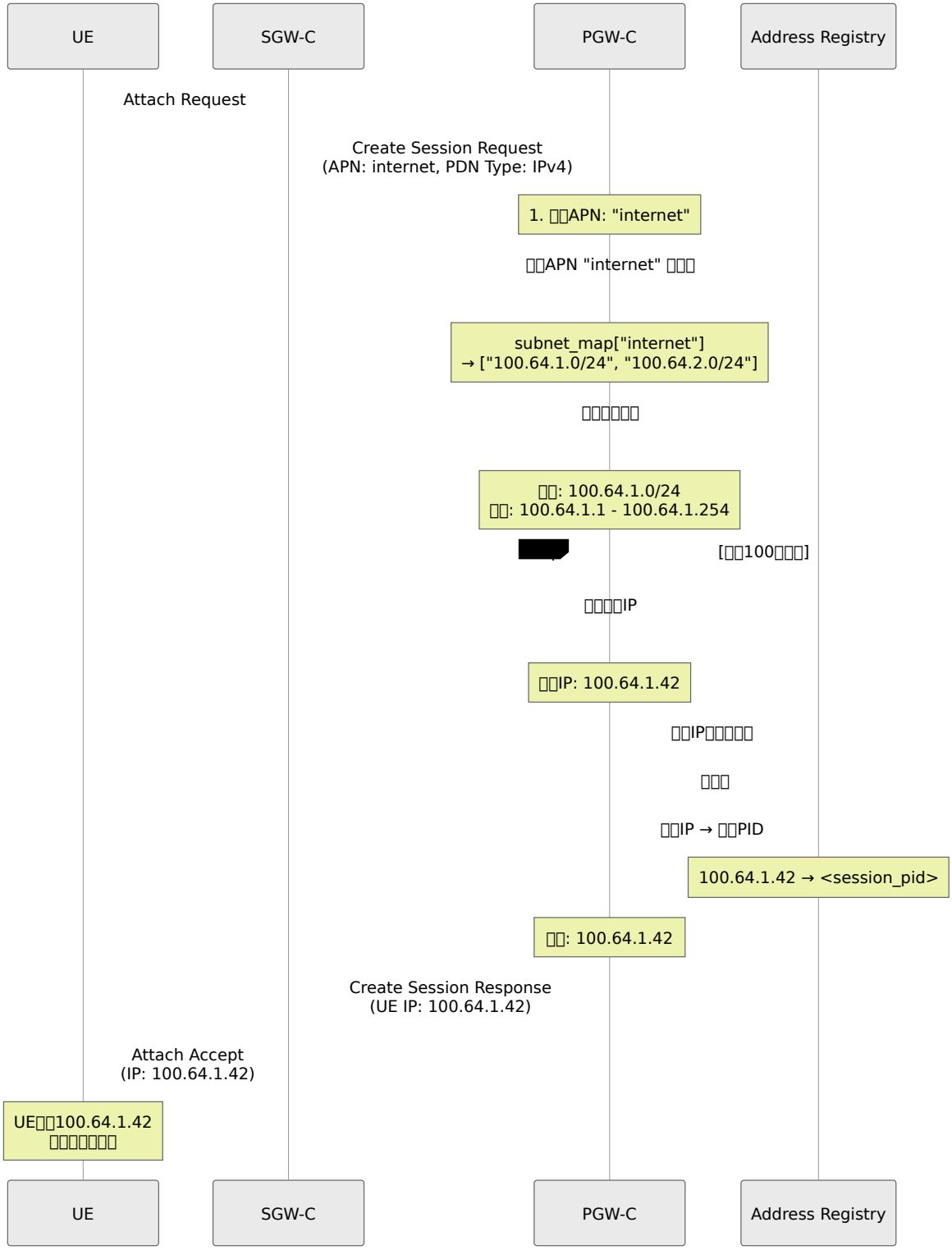
双栈分配:

UE请求PDN类型：IPv4v6
PGW-C同时分配IPv4地址和IPv6前缀
两个地址同时有效

分配过程

IP分配发生在会话创建期间，当PGW-C通过S5/S8接口接收到创建会话请求时。有关GTP-C消息的详细信息，请参见 S5/S8接口和会话管理以了解会话生命周期。

□□□□□IPv4□□


UE
SGW-C
PGW-C
Address Registry
Attach Request
Create Session Request
(APN: internet, PDN Type: IPv4)
1. □□APN: "internet"
□□APN "internet" □□□
subnet_map["internet"]
→ ["100.64.1.0/24", "100.64.2.0/24"]
□□□□□□□
□□: 100.64.1.0/24
□□: 100.64.1.1 - 100.64.1.254
[□□100□□□]
□□□□IP
□□IP: 100.64.1.42
□□IP□□□□□
□□□
□□IP → □□PID
100.64.1.42 → <session_pid>
□□: 100.64.1.42
Create Session Response
(UE IP: 100.64.1.42)
Attach Accept
(IP: 100.64.1.42)
UE□□100.64.1.42
□□□□□□□□
UE
SGW-C
PGW-C
Address Registry

□□□□

□□□□□□□□

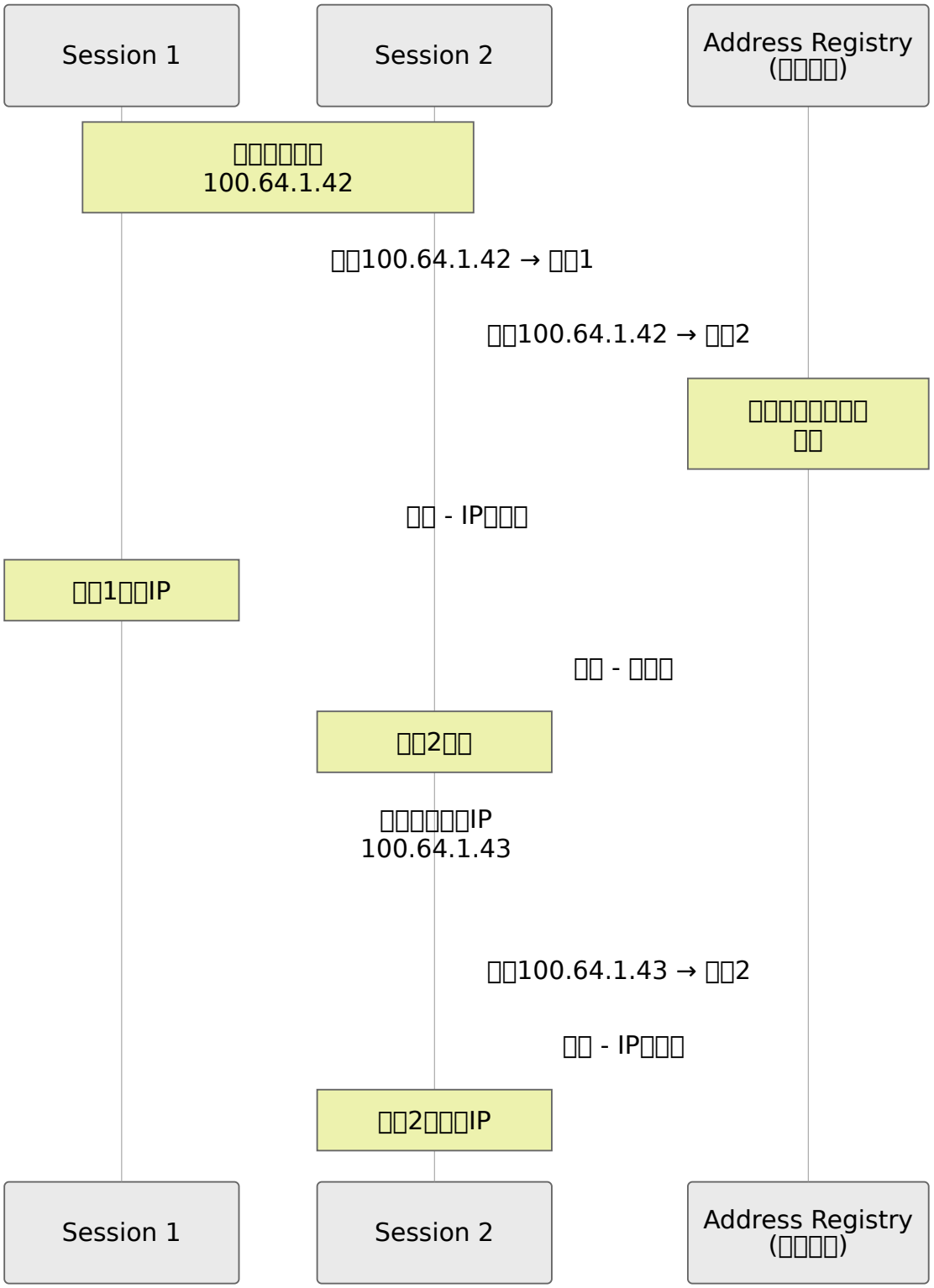
1. 子网查找：系统检索请求的APN的配置子网
2. 随机选择：从可用列表中随机选择一个子网
3. **IP**生成：在子网范围内生成一个随机IP
4. 唯一性检查：系统验证IP是否未被分配
5. 重试逻辑：如果检测到碰撞，最多重试100次，生成新的随机IP
6. 注册：一旦找到唯一IP，就将其注册到会话中

关键��计点：

- 最多**100**次尝试：防止在池几乎耗尽时出现无限循环
- 随机选择：避免可预测的IP分配模式以提高安全性
- 原子□作：基于进程的注册表确保没有重复分配
- 回退到默认：如果在配置中未找到APN，则使用默认池

碰撞处理

场景：两个会话同时尝试分配相同的IP


Session 1
Session 2
Address Registry
(□□□□□)
□□□□□□□□
100.64.1.42
□□100.64.1.42 → □□1
□□100.64.1.42 → □□2
□□□□□□□□□□□
□□
□□ - IP□□□
□□1□□IP
□□ - □□□
□□2□□
□□□□□□□IP
100.64.1.43
□□100.64.1.43 → □□2
□□ - IP□□□
□□2□□□IP
Session 1
Session 2
Address Registry
(□□□□□)

□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□
- □□□□□□□□□IP□□
- □□□□□□□□IP□□□

- 被拒绝的会话会自动重试新的随机IP

默认子网回退

场景：UE请求未���APN

示例配置：

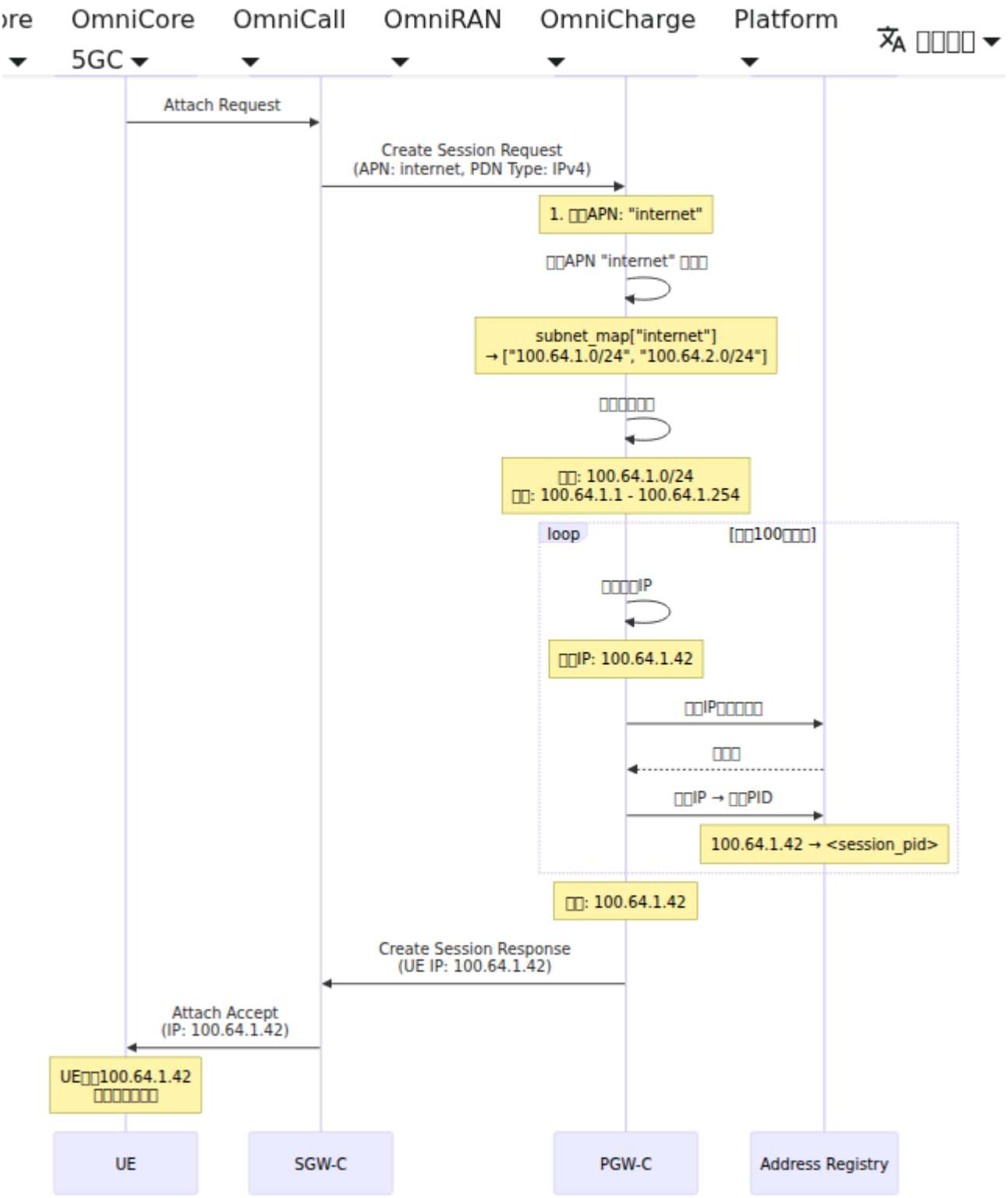
```
# 配置
subnet_map: %{
  "internet" => ["100.64.1.0/24"],
  default: ["42.42.42.0/24"]
}
```

行为：

- UE请求APN: "unknown.apn"
- 系统在subnet_map中查找"unknown.apn"
- 未找到，因此回退到默认池
- 从42.42.42.0/24分配IP

回退逻辑：

1. 首先尝试在配置中查找APN特定池
2. 如果未找到，则使用`default`池
3. 如果未配置默认池，则分配失败



会话终止时的去分配

自动清理：

- 当会话进程终止时，注册表会进行清理
- IP立即可用于新的分配
- 无需手动干预

高级主题

池耗尽

场景：池中的所有IP都已分配

```
池: 100.64.1.0/24 (254个可用IP)
已分配: 254个IP
新请求到达 → 耗尽
```

发生了什么：

1. PGW-C尝试100次随机分配
2. 所有尝试发现IP已分配
3. 返回：`{:error, :ue_ip_address_allocation_failed}`
4. 会话建立失败
5. SGW-C收到错误响应

预防措施：

```
# 监控池利用率
address_registry_count / total_pool_size > 0.8  # 在80%时警报

# 在耗尽之前扩展池
"internet" => [
  "100.64.1.0/24",
  "100.64.2.0/24",  # 添加额外子网
  "100.64.3.0/24"
]
```

静态IP分配

用例：企业设备需要固定IP

GTP-C消息格式：

```
创建会话请求
├── IMSI: 310260123456789
├── APN: enterprise.corp
├── PDN地址分配（IE）
│   └── PDN类型: IPv4
│   └── IPv4地址: 10.100.0.50  ← UE请求特定IP
```

OmniPGW处理:

1. 提取请求的**IP**: 从请求中解析PDN地址分配IE
2. 验证**IP**: 检查请求的IP是否在该APN的配置池中
3. 检查可用性: 验证IP是否未分配给其他会话
4. 分配或拒绝:
    - 如果可用: 将请求的IP分配给该会话
    - 如果不可用: 以适当的原因代码拒绝会话

可能的结果:

- 成功: UE获得其请求的确切IP地址
- 失败（**IP**已使用）: 会话被拒绝 - IP已分配
- 失败（**IP**不在池中）: 会话被拒绝 - IP不在��置范围内

IPv6前缀委派

UE请求**IPv6**:

```
创建会话请求
├── PDN类型: IPv6
```

PGW-C分配**/64**前缀:

```
分配的前缀: 2001:db8:1:a::/64

UE可以使用:
- 2001:db8:1:a::1
- 2001:db8:1:a::2
- ...（18个千万亿地址）
```

好处:

- UE可以分配多个IP（例如，共享网络）
- 支持SLAAC（无状态地址自动配置）
- 消除NAT需求

双栈分配

UE请求IPv4v6:

```
创建会话请求
├── PDN类型: IPv4v6
```

PGW-C同时分配:

```
IPv4: 100.64.1.42
IPv6: 2001:db8:1:a::/64
```

流量处理:

- IPv4流量使用IPv4地址
- IPv6流量使用IPv6前缀
- 两者同时有效
- 单独的GTP隧道（或双栈隧道）

当APN没有IPv6池时:

如果请求IPv4v6 PDN，但匹配的APN只有IPv4 CIDR（没有IPv6范围），IPv6分配器返回`{:error, :no_ipv6_pool}`而不是崩溃，并且会话过程根据APN类型决定:

APN类型	结果
紧急（SOS） APN（名称包含"sos"）	降级为仅IPv4 — 保留IPv4地址，清除IPv6地址，并将协商的PDN类型设置为IPv4。PAA、PFCP和Gx都以仅IPv4的形式出现，P-CSCF仍在PCO中返回，以便进行紧急IMS注册（TS 29.274 / TS 23.401单地址承载 / PDN类型降级）。
常规 APN	拒绝 — 请求不会被默默降级；错误配置（或不支持的请求）会被显现。

对没有IPv6池的APN进行仅IPv6请求也会被拒绝。请参见APN没有IPv6池时的IPv4v6请求以获取完整的行为表。

私有与公共IP地址

私有IP池（RFC 1918）：

```
# 不可在公共互联网路由
subnet_map: %{
  "internet" => [
    "10.0.0.0/8",
    "172.16.0.0/12",
    "192.168.0.0/16"
  ]
}
```

需要在PGW-U处进行NAT以访问互联网

公共IP池:

```
# 可路由的公共IP（仅为示例）
subnet_map: %{
  "internet" => [
    "203.0.113.0/24"   # 公共IP块
  ]
}
```

无需NAT - 直接互联网路由

建议:

- 使用私有**IP**（**RFC 6598**）：`100.64.0.0/10`（运营商级NAT）
- 仅为特殊服务保留公共IP

监控

IP池管理

IP池的分配和利用率通过OAM REST API进行监控（HTTPS/TLS在端口`8443`，路由在`/api`前缀下）。交互式文档（Swagger UI）可在`https://<omnipgw-ip>:8443/api/docs`访问。

```
GET /api/ip_pools              # 所有池 + 利用率
GET /api/ip_pools/<name>       # 按名称获取一个池
```

示例：

```
curl -k https://<omnipgw-ip>:8443/api/ip_pools
curl -k https://<omnipgw-ip>:8443/api/ip_pools/Internet
```

每个配置池返回的信息：

字段	描述
名称	池/APN标识符（例如，“default”，“ims.something.else”，“Internet”）
总计	池中的IP数量
已用	当前已分配的IP
可用	可供分配的剩余IP
利用率%	当前已分配的池百分比

用例：

- 在维护之前快速检查容量
- 识别接近耗尽的池
- 验证池配置
- 按APN监控分配模式

关键指标

地址注册表计数：

```
# 当前已分配的IP
address_registry_count

# 池利用率（需要计算）
address_registry_count / <total_pool_size> * 100
```

示例：

```
池: 100.64.1.0/24 (254个IP)
已分配: 150个IP
利用率: 150 / 254 = 59%
```

警报

```
# 在□池利用率时警报
- alert: UEIPPoolUtilizationHigh
  expr: address_registry_count > 200  # 对于/24池
  for: 10m
  annotations:
    summary: "UE IP池利用率超过80%"
    description: "当前: {{ $value }} / 254个IP已分配"

# 在池耗尽时警报
- alert: UEIPPoolExhausted
  expr: address_registry_count >= 254  # 对于/24池
  for: 1m
  annotations:
    summary: "UE IP池耗尽 - 没有可用IP"

# 在分配失败时警报
- alert: UEIPAllocationFailures
  expr: rate(ue_ip_allocation_failures_total[5m]) > 0
  for: 5m
  annotations:
    summary: "UE IP分配失败发生"
```

Grafana仪表板

面板1: IP池利用率

```
# 显示百分比的仪表
(address_registry_count / 254) * 100
```

面板2: 随时间变化的已分配IP

```
# 时间序列
address_registry_count
```

面板3: 分配率

```
# 新分配的速率
rate(address_registry_count[5m])
```

面板4：池耗尽风险

```
# 直到耗尽的天数（基于当前速率）
(254 - address_registry_count) / rate(address_registry_count[1h])
```

故障排除

问题1：会话建立失败（没有可用IP）

症状：

- 创建会话响应：原因“请求被拒绝”
- 日志：“UE IP地址分配失败”

可能的原因：

1. 池耗尽

   ```
   # 检查当前分配
   curl http://<pgw_c_ip>:42069/metrics | grep
   address_registry_count
   ```

2. 配置错误

```
# 验证子网配置
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => [
        "100.64.1.0/24"  # 确保有效的CIDR
      ]
    }
  }
```

3. **APN**错误配置

```
# 如果未找到APN，则回退到默认
# 确保存在默认池
subnet_map: %{
  default: ["42.42.42.0/24"]
}
```

解决方案：

- 扩展池：添加更多子网
- 清理过期会话：重启PGW-C以释放泄漏的IP
- 验证配置：检查`runtime.exs`中的拼写错误

问题**2**：**IP**地址碰撞

症状：

- 两个UE接收到相同的IP（非常罕见）
- 路由问题

原因：

- 地址注册表中的错误（不应发生）

调试：

```
# 检查日志中的重复IP
grep "already_registered" /var/log/pgw_c.log
```

解决方案：

- 应该自我修正（第二个会话重试）
- 如果持续存在，报告错误

问题3：使用错误的IP池

症状：

- UE从意外的子网接收IP
- APN "internet"从"ims"池获取IP

原因：

- 不正确的subnet_map配置

验证：

```
# 检查精确的APN字符串匹配
subnet_map: %{
  "internet" => [...],     # 区分大小写
  "Internet" => [...]      # 不同的APN!
}
```

解决方案：

- 确保APN名称完全匹配（区分大小写）
- 对于捕获所有的情况使用默认池

问题4：IPv6分配失败

症状：

- UE请求IPv6，收到错误

- UE请求**IPv4v6** PDN，但APN没有IPv6范围，结果是 获得仅**IPv4**会话（SOS APN）或会话被拒绝（常规APN）

可能的原[illegible]：

1. 未配置**IPv6**池

```
# 缺少IPv6子网
subnet_map: %{
  "internet" => [
    "100.64.1.0/24"  # 仅IPv4
  ]
}
```

2. 无效的**IPv6**前缀

```
# 前缀过小（应为/48或更大）
"internet" => [
  "2001:db8::/128"  # 错误 - 没有分配空间
]
```

没有**IPv6**池的**IPv4v6**请求（降级与拒绝）：

当**IPv4v6**请求命中没有IPv6范围的APN时，`register_new_ipv6/1` 返回`{:error, :no_ipv6_pool}`（它不会崩溃）。会话然后：

- 紧急（**SOS**）**APN**（名称包含`"sos"`）：降级为仅**IPv4** — 分配IPv4地址，清除IPv6地址，将协商的PDN类型设置为IPv4，并仍然返回P-CSCF在PCO中。记录为警告。
- 常规**APN**：拒绝，返回`{:error, :no_ipv6_pool}` — 不进行静默降级。记录为错误。

请参见APN没有IPv6池时的IPv4v6请求。

解决方案：

```
# 添加IPv6池（对于应提供双栈的常规APN）
subnet_map: %{
  "internet" => [
    "100.64.1.0/24",
    "2001:db8:1::/48"  # IPv6池
  ]
}
```

对于仅IPv4部署的紧急（SOS）APN，IPv4-only降级是故意的 — 仅在SOS PDN应为双栈时添加IPv6 CIDR。

问题5：高池利用率

症[illegible]：

- 接近池耗尽
- `address_registry_count`接近最大值

主动措施：

1. 添加子网：

```
"internet" => [
  "100.64.1.0/24",    # 现有
  "100.64.2.0/24",    # 新子网（增加254个IP）
  "100.64.3.0/24"     # 新子网（增加254个IP）
]
```

2. 使用更大的子网：

```
# 将/24替换为/22
"internet" => [
  "100.64.0.0/22"  # 1022个可用IP
]
```

3. 会话清理：
    - 监控过期会话
    - 确保正确处理删除会话请求

最佳实践

容量规划

计算所需的池大小：

```
预期并发用户：10,000
峰值并发：30%（3,000个同时会话）
增长缓冲：50%
所需IP：3,000 * 1.5 = 4,500个IP

子网：/20（4,094个可用IP） - 太小
子网：/19（8,190个可用IP） - 足够
```

子网选择

推荐：

- 使用`100.64.0.0/10`（RFC 6598 - 运营商级NAT）
- 提供400万个IP
- 保留给服务提供商NAT

避免：

- 公共IP（昂贵，有限）
- 与企业VPN冲突的常见私有范围

配置布局

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # 主要互联网APN - 大池
      "internet" => [
        "100.64.0.0/18"  # 16,382个IP
      ],

      # IMS - 较小的专用池
      "ims" => [
        "100.64.64.0/22"  # 1,022个IP
      ],

      # 企业 - 中等池
      "enterprise.corp" => [
        "100.64.68.0/22"  # 1,022个IP
      ],

      # IoT - 大池以支持许多设备
      "iot.m2m" => [
        "100.64.72.0/20"  # 4,094个IP
      ],

      # 默认 - 小回退
      default: [
        "100.64.127.0/24"  # 254个IP
      ]
    }
  }
```

相关文档

配置

- 配置指南 - UE IP池配置，APN子网映射
- **PCO**配置 - 与IP地址一起交付的DNS、P-CSCF、MTU

- 会话管理 - 会话生命周期，PDN设置期间的IP分配
- **PFCP**接口 - 通过PFCP到UPF的UE地址分配

网络规划

- **S5/S8**接口 - 通过GTP-C交付IP地址
- **Diameter Gx**接口 - IP分配的策略控制

□作

- 监控指南 - IP池利用率指标，分配跟踪
- 数据**CDR**格式 - 用于计费关联的CDR中的UE IP地址

返回□作指南