OmniPGW □置指南

runtime.exs □置的完整参考

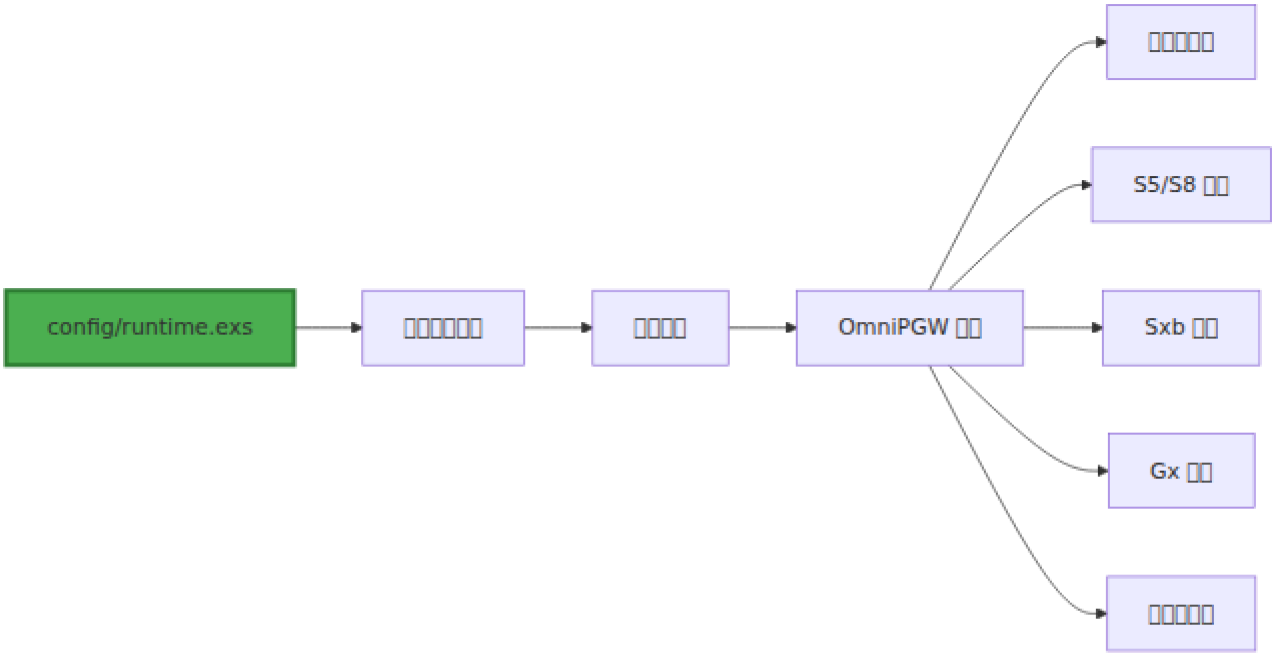
□ Omnitouch □□□□□□

目录

概述

OmniPGW 使用在 `config/runtime.exs` 中定义的 运行时□置。该文件在 应用程序启动时 进行评估，并允许基于环境变量或外部来源的动态配置。

□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 - 合理的默认值和明确的覆盖

□置文件结构

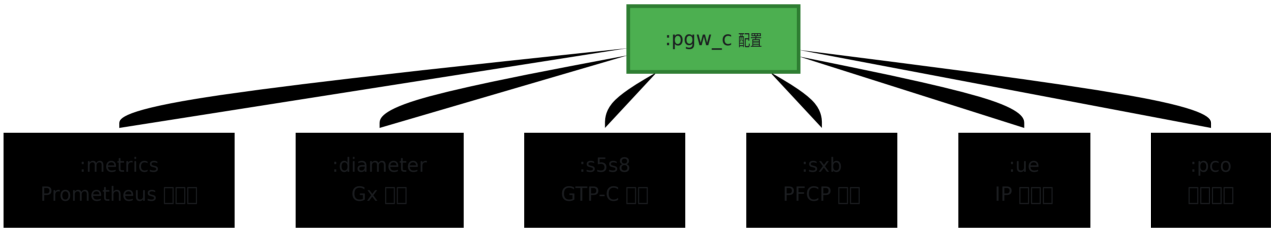
文件位置

```
pgw_c/
├── config/
│   ├── config.exs          # □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
```

顶层结构

```
# config/runtime.exs
import Config

config :logger, level: :info

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

□置部分



指标□置

目的

配置 Prometheus 指标导出器以监控 OmniPGW。

置块

```
config :pgw_c,
  metrics: %{
    # 启用/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
`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,  # ���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 □置

目的

配置 Diameter 协议以用于 Gx 接口（PCRF 通信）。

□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",

        # 启动与 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��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□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 □置

目的

配置 GTP-C 接口以与 SGW-C 通信。

□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 请求的重试尝试次数 (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`	每次重试尝试的超时（毫秒）
`request_attempts`	整数	`3`	放弃之前的重试尝试次数

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□置

S5/S8 接口使用可配置的超时来处理 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：

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

调整指南：

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 如果频繁出现“创建承载请求超时”错误，但 Wireshark 显示响应到达
- 减少超时 以便在低延迟环境中更快地检测失败
- 增加重试尝试 对于存在数据包丢失的不可靠网络

超时行为：

- 超时时，记录��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：

- 使用专用管理/信令网络
- 确保从所有 SGW-C 节点可达
- 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
```

Sxb/PFCP □置

目的

配置 PFCP 接口以与 PGW-U（用户平面）通信。

□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** 对等体会在启动时从 **`upf_selection`** □置（规则 + 回退池）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 启动）
>   - 5 秒心跳间隔
> - UPF 选择规则和池在单独的 `upf_selection` 部分中配置。请参见下面的 UPF 选择策略。
> - 支持动态 UPF 注册，适用于未在配置中发现的 DNS 发现的 UPF

示例

最小□置：

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

# 所有在 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
}
```

与 **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。

□置结构

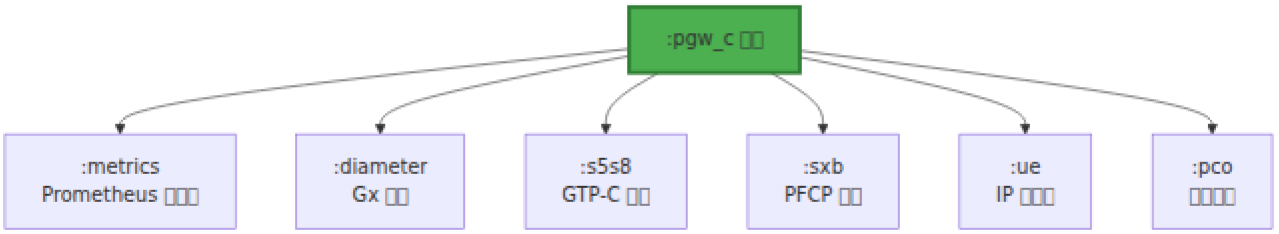
UPF 选择在 **`upf_selection`** 部分中配置，该部分定义：

1. 静态规则 - 基于模式的路由和负载均衡池
2. **DNS** 设置 - 基于位置的动态 UPF 发现
3. 回退池 - 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） - 基于模式的路由和负载均衡池
2. 基于 **DNS** 的选择（较低优先级） - 基于位置的动态 UPF 发现
3. 回退池（最低优先级） - 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□字段

静态规则可以匹配以下任何会话属性：

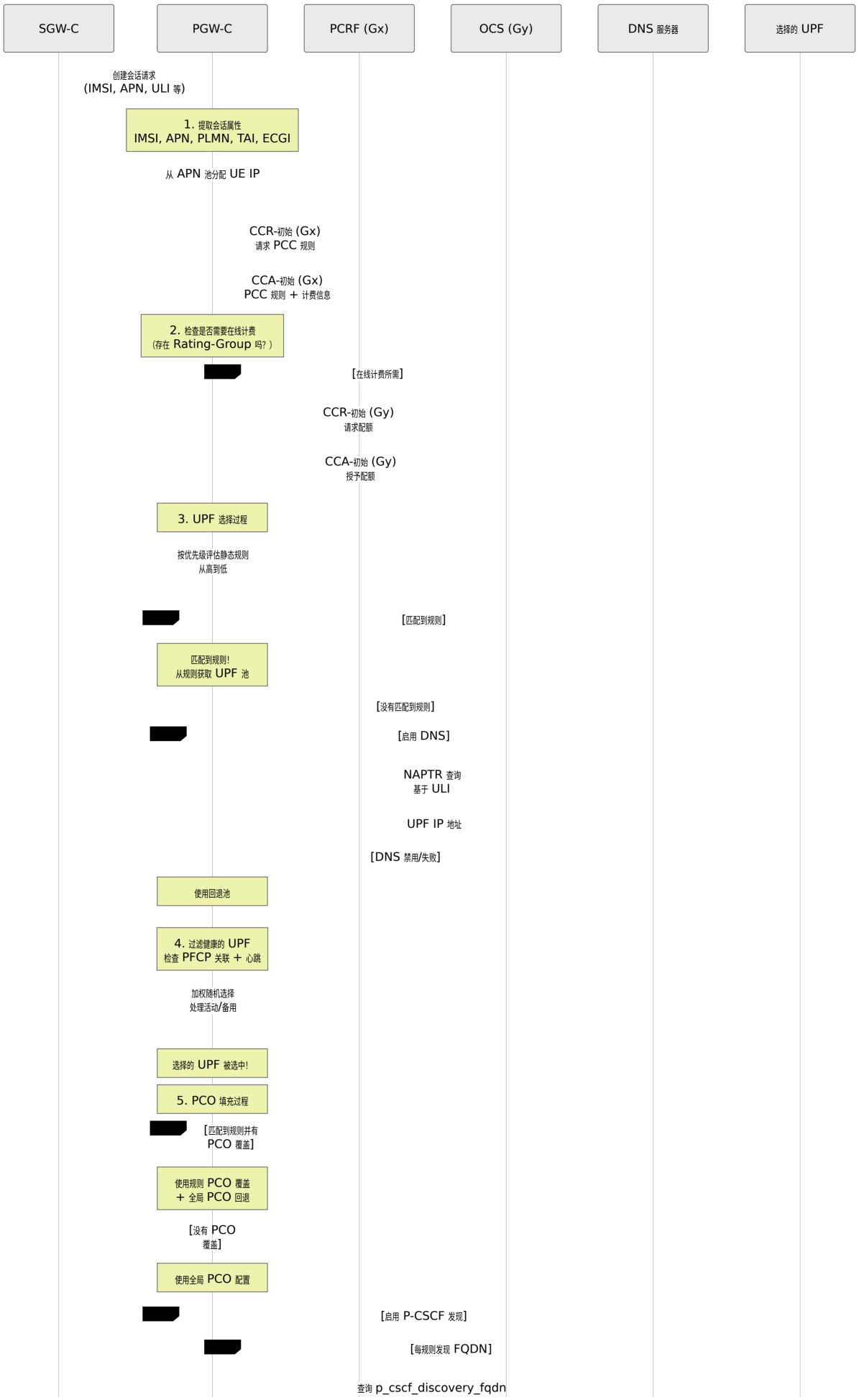
匹□字段	描述	示例模式
`:imsi`	国际移动用户身份	`^313380.*`（美国运营商）
`:apn`	接入点名称 / DNN	`^internet\.` 或 `^ims\.`
`:serving_network_plmn_id`	服务网络标识符	`^313380$`
`:sgw_ip_address`	SGW IP 地址	`^10\.100\..*`
`:uli_tai_plmn_id`	跟踪区 PLMN ID	`^313.*`
`:uli_ecgi_plmn_id`	E-UTRAN 小区 PLMN ID	`^313.*`

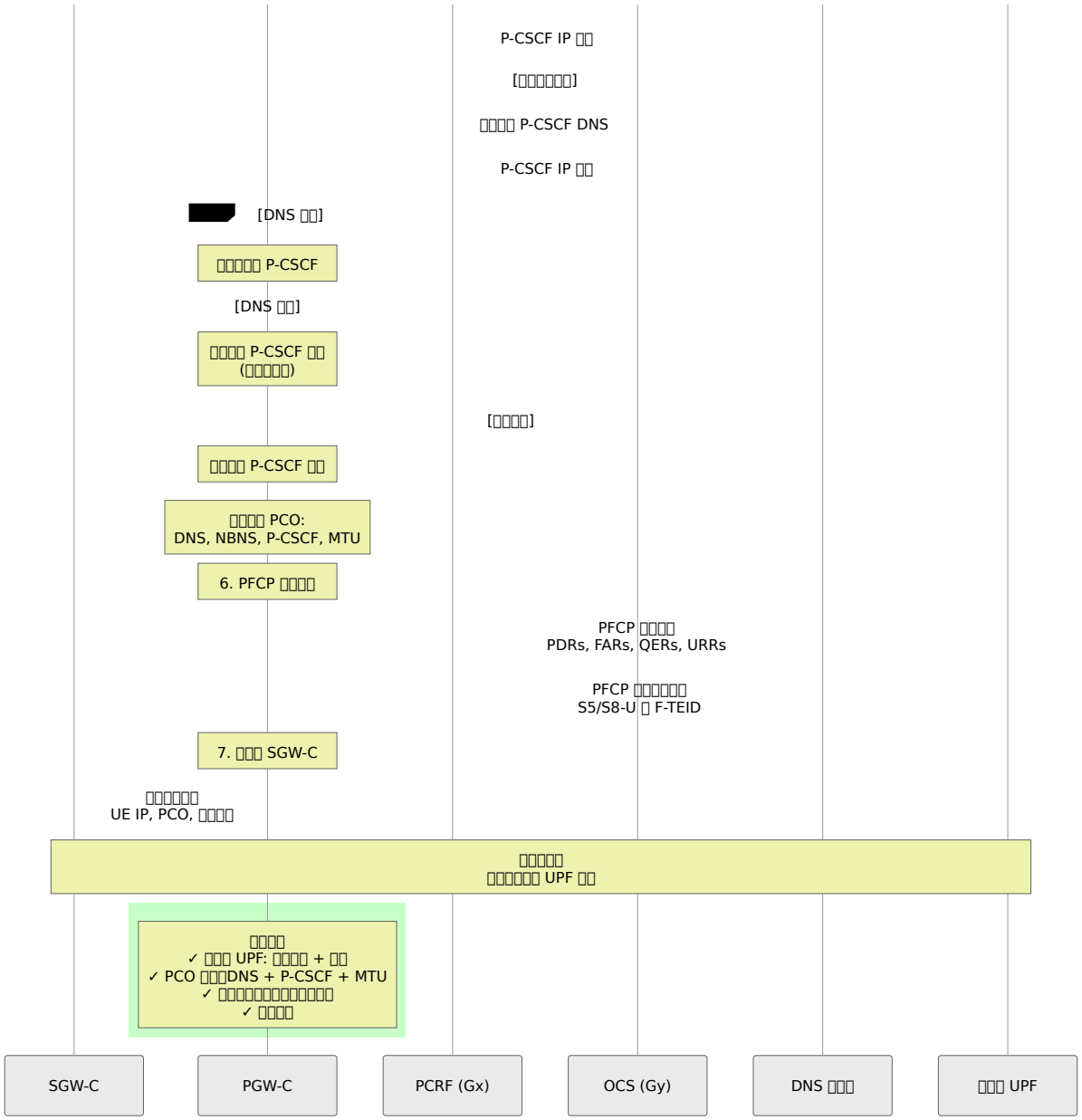
选择方法比较

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**	服务区分	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
回退池	安全网	始终有一个 UPF	可能不是最佳选择
干运行模式	测试配置	安全测试	无真实流量

完整会话建立流程

该图显示了会话建立的完整端到端流程，包括 UPF 选择和 PCO 填充：


SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
DNS 服务器
选择的 UPF
创建会话请求
(IMSI, APN, ULI 等)
1. 提取会话属性
IMSI, APN, PLMN, TAI, ECGI
从 APN 池分配 UE IP
CCR-初始 (Gx)
请求 PCC 规则
CCA-初始 (Gx)
PCC 规则 + 计费信息
2. 检查是否需要在线计费
(存在 Rating-Group 吗?)
[在线计费所需]
CCR-初始 (Gy)
请求配额
CCA-初始 (Gy)
授予配额
3. UPF 选择过程
按优先级评估静态规则
从高到低
[匹配到规则]
匹配到规则!
从规则获取 UPF 池
[没有匹配到规则]
[启用 DNS]
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/备用
选择的 UPF 被选中!
5. PCO 填充过程
[匹配到规则并有
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



□□□□□□

1. **UPF** □□□□□□

   - □□□□□□□□□□ → DNS □□ → □□□
   - □□□□□□□□□□□□□□
   - □□/□□□□□□□□□□□□
   - □□□ PFCP □□ □□□ UPF □□□□□□□

2. **PCO** □□□□□□

   - □□ PCO □□ → P-CSCF DNS □□ → □□ PCO □□
   - □□□□□□□□□□□□□□□□□□□□□□□□□□□□

- 参见： PCO 配置 以获取详细的 PCO 参数

3. **P-CSCF** 发现优先级：

   - 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□置示例

以下是一个完整示例，显示多池 UPF 选择与自动对等体注册：

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

  # UPF 选择逻辑 - 所有在此定义的 UPF 自动注册
  upf_selection: %{
    # 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 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}
        ]
      },

      # 规则 2: 企业 APN
      %{
        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}
        ]
      },

      # 规则 3: 互联网流量 - 负载均衡
```

```
      %{
        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 - 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
- □ □□□置：所有 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` 具有加权分布
- □ **DNS** 集成：DNS 设置与选择规则并排配置
- □ □□□□□DNS 发现的 UPF 在首次使用时自动注册
- □ □□□□□□□□□□ UPF 都通过 5 秒心跳进行监控

从先前格式迁移：

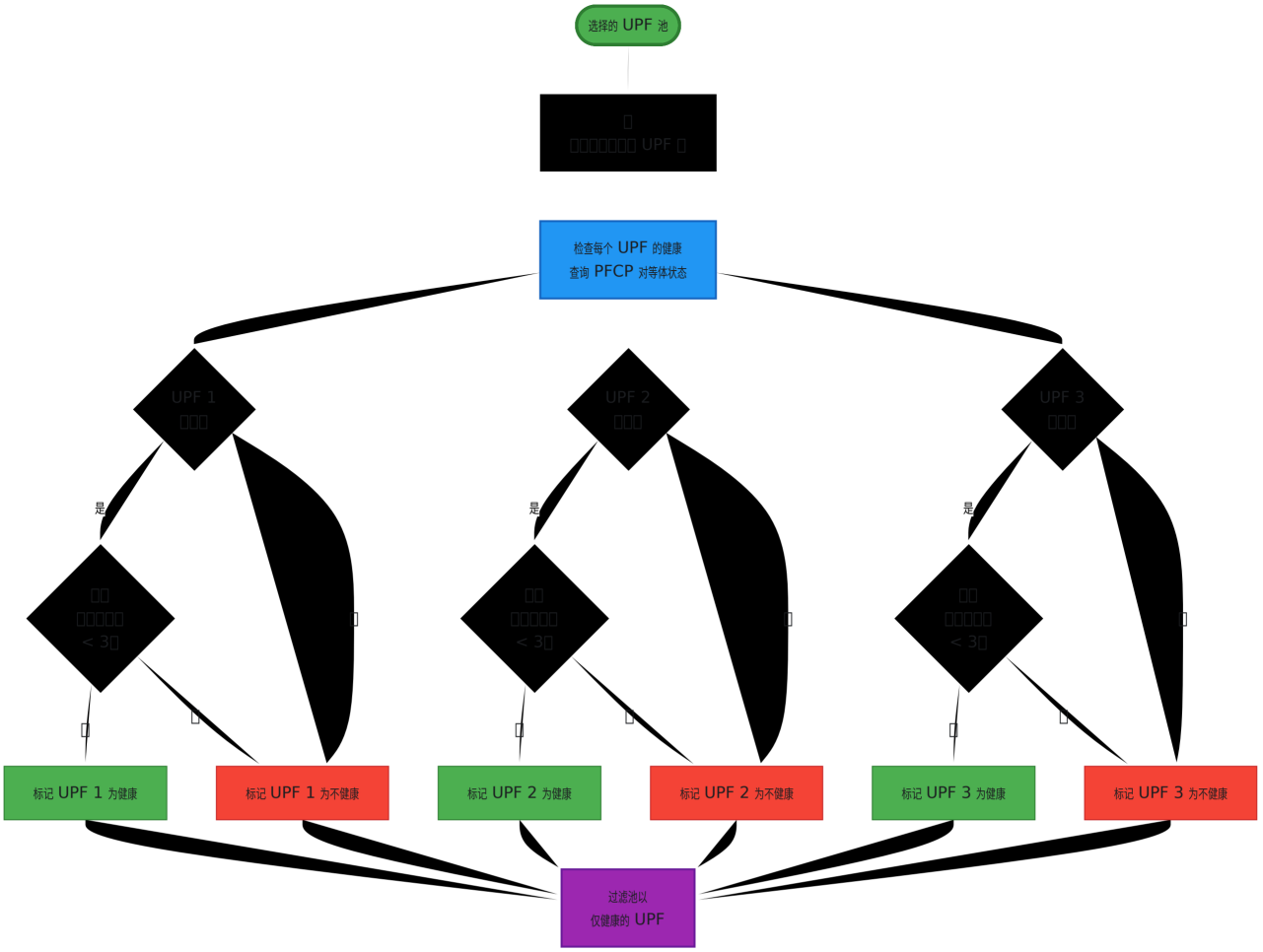
- 移除：`sxb.peer_list` 字段（不再需要）
- 移除：嵌入在对等体配置中的 `selection_list`

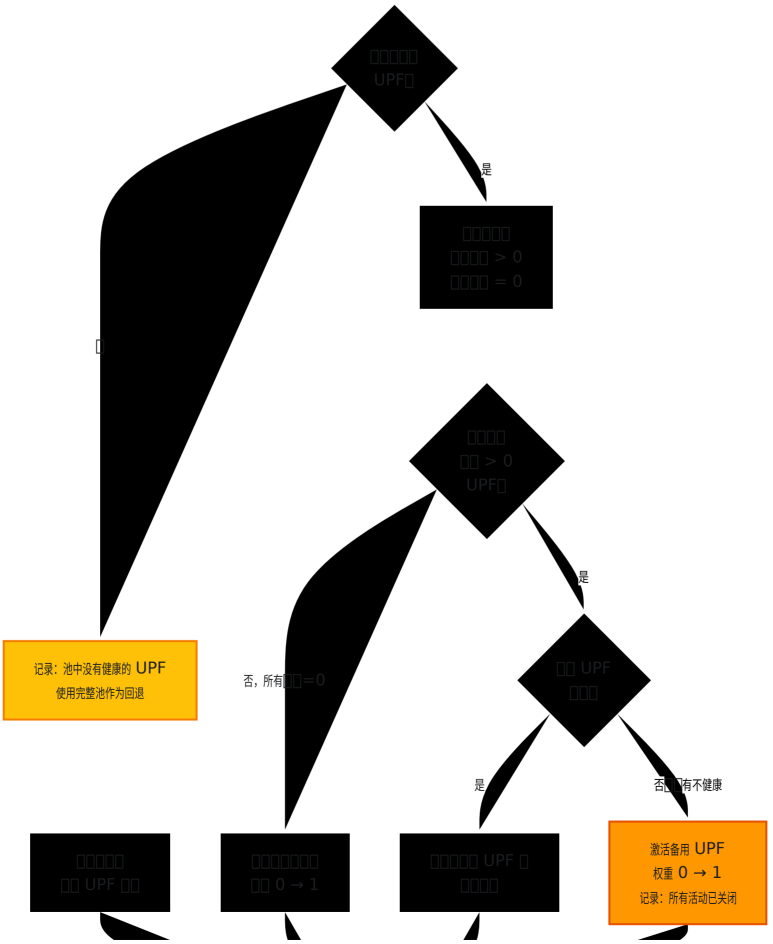
- 所有 UPF 定义现在放在 `upf_selection` 规则和回退池中

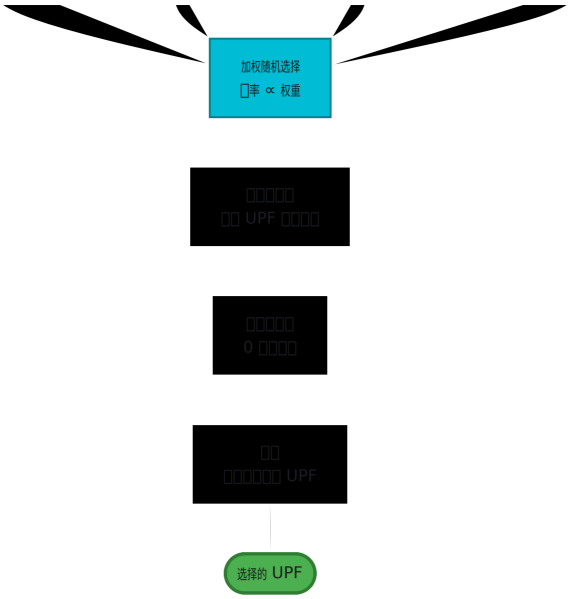
UPF 池的工作原理：

1. 健康监测选择：仅健康的 UPF 接收流量
    - 健康 = PFCP 关联活动 + 连续未响应少于 3 次心跳
    - 不健康的 UPF 会自动过滤掉
    - 如果没有健康的 UPF，则回退到所有 UPF
2. 活动/备用支持：使用 `weight: 0` 作为备用 UPF
    - 活动 **UPF**（权重 > 0）：在健康时接收流量
    - 备用 **UPF**（权重 == 0）：仅在所有活动 UPF 失败时接收流量
    - 当激活时，备用 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
    - 在上述示例中：70% 分配给 .21，20% 分配给 .22，10% 分配给 .23
    - 权重越高，分配给该 UPF 的会话越多
    - 权重相等 = 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
    - 所有配置的 UPF 立即进行健康监测

健康监测选择与活动/备用


选择的 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
仅健康的 UPF


是
记录：池中没有健康的 UPF
使用完整池作为回退
否，所有
是
是
否，有不健康
激活备用 UPF
权重 0 → 1
记录：所有活动已关闭



加权随机选择示例：

```
池: [
  UPF-A: 权重 50, 健康 ✓
  UPF-B: 权重 30, 健康 ✓
  UPF-C: 权重 20, 健康 ✓
]

总权重: 50 + 30 + 20 = 100

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

随机数: 63 → 选择 UPF-B
随机数: 15 → 选择 UPF-A
随机数: 91 → 选择 UPF-C
```

活动/备用故障转移示例：

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

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 失败
→ 没有健康的活动 UPF
→ 激活备用: [UPF-B: 1]
→ 所有流量故障转移到 UPF-B
→ 记录: 所有活动已关闭，激活备用 UPF

场景 3: 两者都不健康
→ 没有健康的 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）
→ 记录: 池中没有健康的 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/备用（100% 主，0% 备用，直到主失败）
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}
# 备用（仅在活动关闭时）
]

# 活动与多个备用（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},
# 备用 1
  %{remote_ip_address: "10.0.1.3", remote_port: 8805, weight: 0}
# 备用 2
]
# 结果：活动获得 100%。如果活动失败，备用获得 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/备用故障转移：使用 `weight: 0` 为热备用 UPF，仅在主 UPF 失败时激活
- 健康监测高可用性：当 UPF 失去 PFCP 关联或错过心���时自动故障转移
- 水平扩展：通过多个 UPF 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

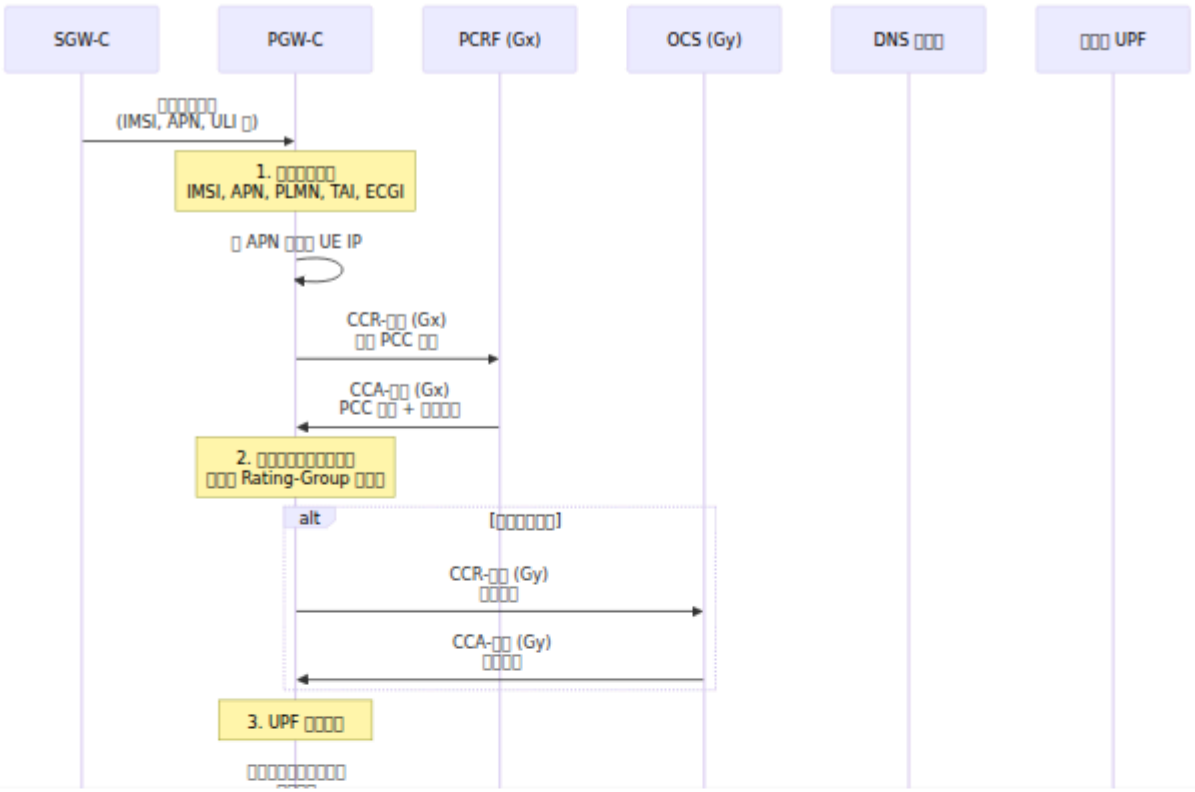
PCO（协议□置选项）覆盖：

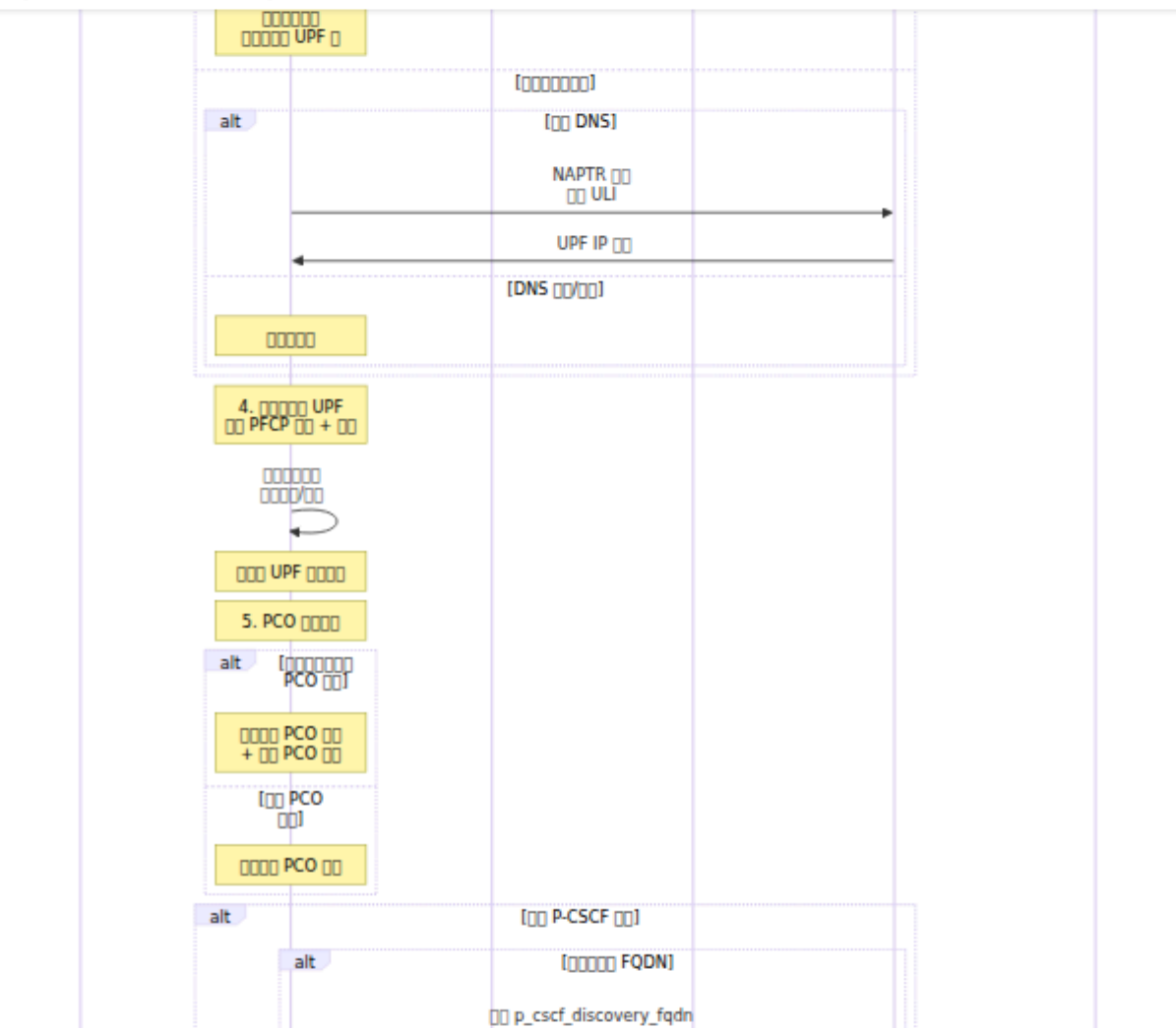
每个 UPF 选择规则可以选择性地指定自定义 PCO 值，这些值覆盖匹配会话的默认 PCO 配置。这允许不同的 APN 或流量类型接收不同的网络参数。

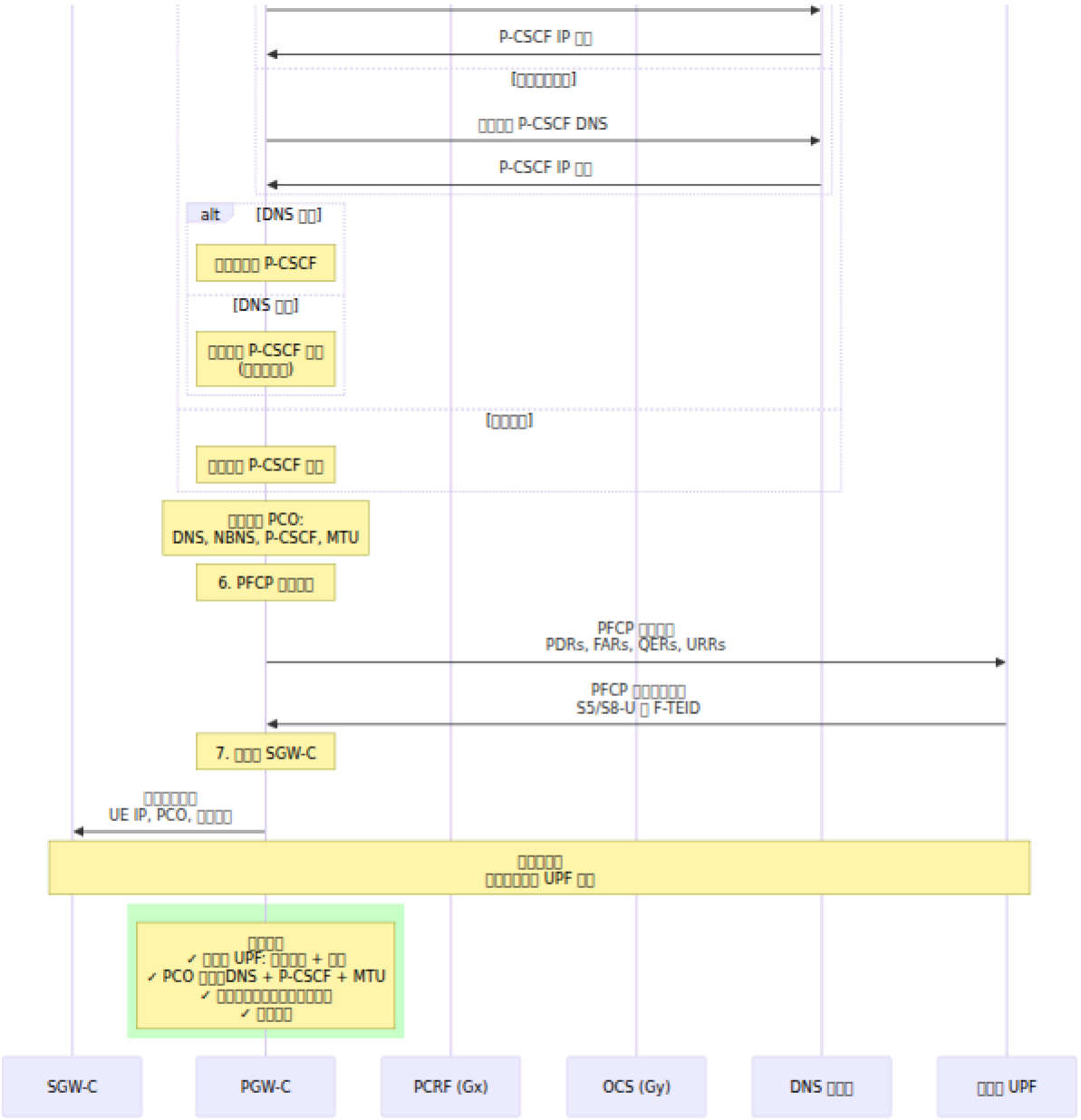
PCO 覆盖的工作原理：

1. 部分覆盖：仅指定要覆盖的 PCO 字段
2. 默认回退：未指定的字段使用来自主 `pco` 配置的值
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 □□□
□□□ UPF
□□□□□□□
(IMSI, APN, ULI □)
1. □□□□□□□
IMSI, APN, PLMN, TAI, ECGI
□ APN □□□ UE IP
CCR-□□ (Gx)
□□ PCC □□
CCA-□□ (Gx)
PCC □□ + □□□□
2. □□□□□□□□□□□
□□□ Rating-Group □□□
alt
[□□□□□□□]
CCR-□□ (Gy)
□□□□
CCA-□□ (Gy)
□□□□
3. UPF □□□□
□□□□□□□□□□□


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



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** □置（最低优先级 / 回退）

示例：**IMS** 规则覆盖 **DNS**，企业规则覆盖所有

```
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（规则中设置了 p_cscf_discovery_fqdn）
│  └─ 回退：来自规则（如果 DNS 失败）
└─ 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 DNS（如果启用全局发现）
└─ MTU：来自全局
```

可用的 **PCO** 覆盖字段：

- `primary_dns_server_address` - 主 DNS 服务器 IP
- `secondary_dns_server_address` - 次 DNS 服务器 IP
- `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 列表（用于 IMS） - 请参见 PCO 配置和 P-CSCF 监控以获取动态 P-CSCF 发现
- `ipv4_link_mtu_size` - MTU 大小（字节）

每规则的 **P-CSCF** 发现：

除了 PCO 覆盖，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.mnc380.mcc313.3gppnetwork.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. 请参见 P-CSCF 监控 以获取监控发现成功/失败率

这对于以下情况特别有用：

- **IMS APN** - 不同 IMS 网络具有不同的 P-CSCF 服务器
- 多租户部署 - 不同企业具有专用的 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 自动返回仅健康的 P-CSCF 服务器

示例：**IMS** 流量与自定义 **P-CSCF**：

```
rules: [
  %{
    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}
    ],
    # P-CSCF 发现：动态查询 DNS 以获取 P-CSCF 地址
    # DNS 查找返回□于此 FQDN 的当前 P-CSCF IP
    p_cscf_discovery_fqdn: "pcscf.mnc380.mcc313.3gppnetwork.org",
    # IMS 会话获得自定义 P-CSCF 服务器（如果 DNS 失败则使用回退）
    pco: %{
      p_cscf_ipv4_address_list: ["10.101.2.100", "10.101.2.101"]
      # DNS、NBNS、MTU 将使用主 pco 配置中的默认值
    }
  }
]
```

示例：企业流量与自定义 **DNS**：

```
rules: [
  %{
    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}
    ],
    # 企业会话获得公司 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：完整覆盖（所有 **PCO** 字段）：

```
rules: [
  %{
    name: "IoT APN - 完全自定义",
    priority: 10,
    match_field: :apn,
    match_regex: "^iot\.m2m",
    upf_pool: [
      %{remote_ip_address: "10.100.5.21", remote_port: 8805,
weight: 100}
    ],
    # IoT 会话获得完全自定义的 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 不需要 P-CSCF
      ipv4_link_mtu_size: 1280       # 约束设备的较小 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 **APN**: 通过公司 DNS 路由企业流量
- **IoT/M2M**: 使用公共 DNS 和优化的 MTU 适应低带宽设备
- 漫游: 为访问用户提供本地 DNS 服务器
- 服务区分: 每种服务类型提供不同的网络参数

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: 这提供了基于地理或拓扑的 UPF 选择。请参见 PFCP 接口 以获取与动态发现的 UPF 的 PFCP 关联设置和 会话管理 以获取会话建立流程。

```
upf_selection: %{
  # 启用□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,

  # ... 规则和回退池 ...
}
```

基于 DNS 的选择工作原理如下：

1. 优先级：仅在 没有静态规则匹□ 时使用 DNS 选择（较低优先级）
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 查询：`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：按优先顺序尝试每种位置类型，直到找到匹配
4. 对等体匹□：DNS 结果与配置的对等体列表进行过滤
5. 选择：选择匹配的对等体（当前为第一个匹配，负载基础选择即将到来）

示例 **DNS** 记录（在您的 DNS 服务器上配置）：

```
; NAPTR 记录用于 PLMN 313-380 中的 TAC 100
tac-lb64.tac-hb00.tac.epc.mnc380.mcc313.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：在不重新配置 PGW-C 的情况下添加/删除 UPF
- 负载均衡：根据位置在 UPF 之间分配负载
- 网络切片：将不同切片路由到不同 UPF 每个位置

UPF 健康监控

自动健康监测选择：PGW-C 持续监控所有 UPF 的健康状况，并自动排除不健康的 UPF。

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：连续未响应的心跳少于 3 次
3. 进程存活：UPF 对等体 GenServer 进程正在运行

如果任何以下条件为真，则 UPF 被视为 不健康：

- PFCP 关联未建立（`associated: false`）
- 连续 3 次或更多心跳超时
- UPF 对等体进程崩溃或无响应

监控机制

对于□置的 **UPF**（在 `upf_selection` 中）：

- 健康跟踪在启动时立即开始
- 持续监控 PFCP 关联
- 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，直到首次会话尝试

- 在首次使用时自动注册
- 注册后开始健康跟踪

选择行为

活动/备用模式（使用 `weight: 0` 时）：

1. 过滤仅健康的 UPF
2. 分开为 活动（权重 > 0）和 备用（权重 == 0）
3. 如果有任何健康的活动 UPF，则使用活动 UPF
4. 如果所有活动 UPF 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：

```
[debug] 使用活动 UPF 池（2/3 健康 UPF，1 备用）
[info] 所有活动 UPF 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（总共 3 个），使用完整池作为回退
```

检查 **UPF** 健康

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 **Web UI**:

- 导航到控制面板中的 `/upf_selection`
- 查看所有 UPF 在每个池中的实时健康状态
- □□□□□□ □□-UP□□ □□-□□□□ □□-关闭,□未关联
- 角色徽章: 活动（权重 > 0），备用（权重 == 0），动态（DNS 发现，不在配置中）
- 显示与关联 UPF 的心跳未命中计数器

健康监控最佳实践

1. 在 **upf_selection** 中□置 **UPF**: 所有 UPF 在规则和回退池中自动监控

```
upf_selection: %{
  rules: [
    %{
      name: "互联网流量",
      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
# - 启动时的健康监控
# - 自动生成的名称
```

2. 使用备用 **UPF**: 使用 `weight: 0` 配置热备用 UPF 以实现自动故障转移

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

3. 通过 **Web UI** 监控: 定期检查控制面板中的 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体的健康。

动态 **UPF** 注册

功能: PGW-C 自动注册并监控通过 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** 对等体：为未知 UPF 生成默认对等体配置
2. 启动 **PFCP** 关联：尝试与 UPF 建立 PFCP 关联
3. 在对等体注册表中注册：将 UPF 添加到内部对等体跟踪系统
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□置

当动态注册 UPF 时，它会接收以下默认配置：

```
%{
  name: "Dynamic-UPF-<IP>",                # 例如，"Dynamic-UPF-10-100-
1-21"
  remote_ip_address: <discovered_ip>,      # 来自 DNS 或选择的 IP
  remote_port: 8805,                       # 标准 PFCP 端口（可覆盖）
  initiate_pfcp_association_setup: true,   # PGW-C 启动关联
  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` 规则中用作目标，而不是选择逻辑的来源。

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��� 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. 记录: “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，显示清晰的错误消息
```

好处

□ □□□□□□□□□□□ DNS 的 UPF 选择现在可以在没有预配置的情况下工作
□ □□□□□□□□□□□□ PGW-C 的情况下添加 UPF
□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□
□ □□□□□□□□□□ UPF 继续按原样工作
□ □□□□□□□□ UPF 与静态对等体相同的心跳监控

故障处理

如果动态发现的 UPF 未能响应 PFCP 关联设置:

```
[error] 动态 UPF {10, 200, 5, 99} 的 PFCP 关联设置失败: :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□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，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**	两种方法均可（动态更方便）
关键任务 **UPF**	在 `upf_selection` 中预配置（确保启动时监控）
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 在日志中显示其自动生成的名称:

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

□□□

数据充电数据记录（CDR）格式

PGW-C 的离线充电

OmniPGW □ Omnitouch □□□□□□

目录

概述

数据 **CDR**（充电数据记录）格式为数据包网关控制平面（PGW-C）提供离线充电能力。CDR 被生成以记录承载会话事件、数据使用情况和用户信息，以便进行计费和分析。

此通用格式与 SGW-C CDR 兼容，确保 EPC 基础设施中充电记录的一致性。

主要特性

- 基于 **CSV** 的格式 - 简单、可读的人类可读的逗号分隔值
- 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 ��准** - 遵循 3GPP TS 32.251（PS 域充电）和 TS 32.298（CDR 编码）

用例

用例	描述
离线充电	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
```

文件名是文件创建时的 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（人类可读和 Unix 时间戳）
- 文件结束时间 - 文件轮换将发生的时间（人类可读和 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	承载的唯一充电标识符
4	msisdn	字符串	移动站 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`	会话期间的使用报告	用户平面发送的定期使用报告
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
```

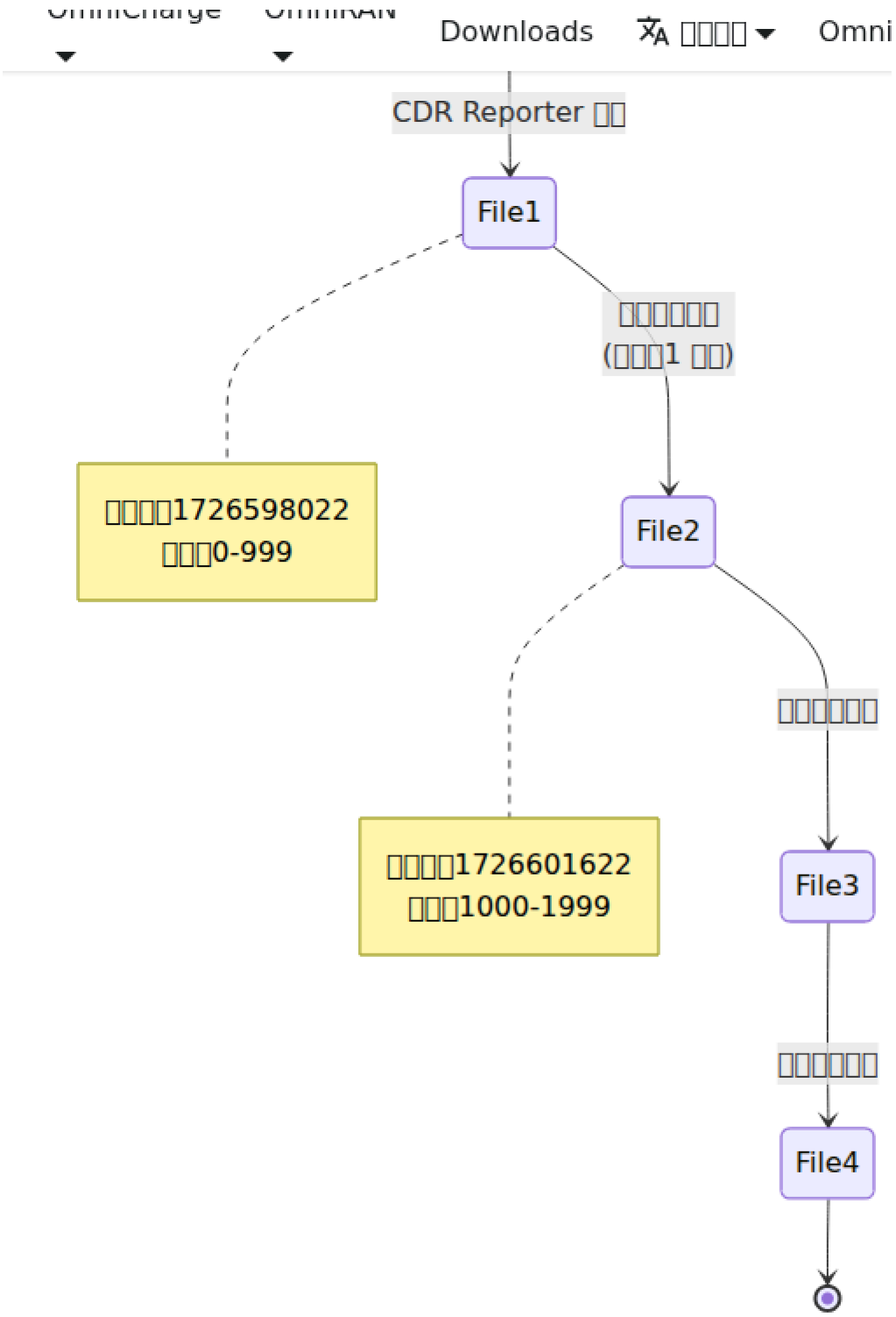
文件结构

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

□□

□□□□

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/cdrs`
`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 URRs**（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:

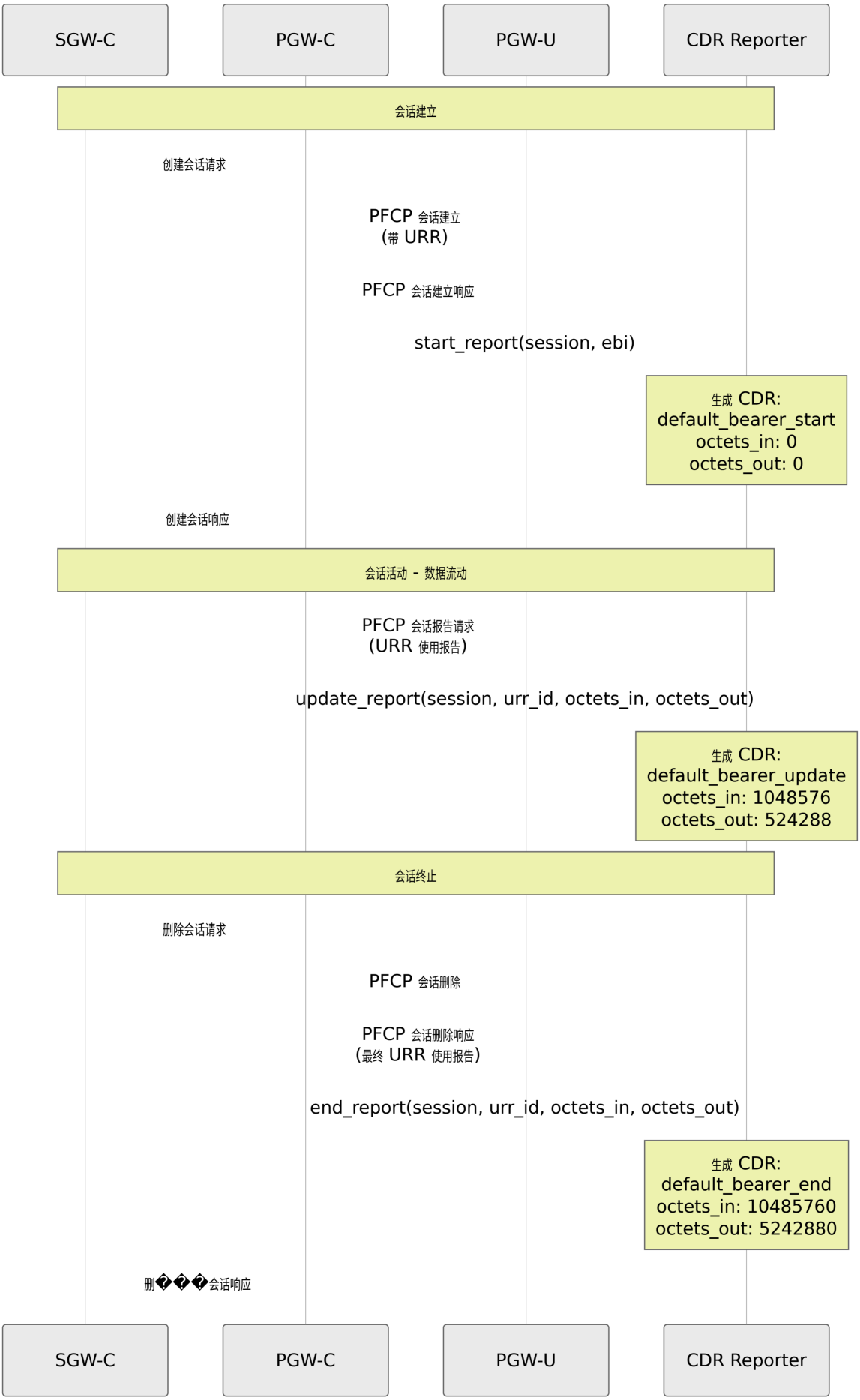
```
# lib/core/session/types.ex
defmodule PGW_C.Session.Types.URR do
  typedstruct do
    field :urr_id, non_neg_integer()
    field :measurement_method, :duration | nil
    field :reporting_triggers, :time_threshold | nil
    field :time_threshold, non_neg_integer() | nil  # 秒
  end
end
```

请参阅 PFCP 接口文档 以获取 URR PFCP 详细信息，以及 lib/core/session/impl/procedures.ex:468 中的会话建立期间的 URR 创建。

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���会话响应
SGW-C
PGW-C
PGW-U
CDR Reporter

CDR 生成事件

1. 承载开始：

- 何时：发送创建会话响应时
- 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 收到 PFCP 会话报告请求（URR 使用报告）
- 目的：记录增量数据使用
- **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 收到 PFCP 会话删除响应（带有最终使用报告）
- 目的：记录会话终止前的最终数据使用
- **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（充电标识符）

类型：无符号 32 位整数

描述：用于网络元素间充电关联的唯一标识符

示例：

```
12345
```

来源：由 PGW-C 分配，在创建会话响应中接收

用法：

- 关联 SGW 和 PGW 之间的充电事件
- 用于直径 Gy/Gz 充电接口
- 每个承载唯一

---

5. msisdn（电话号码）

类型：字符串（E.164 格式）

描述：移动站 ISDN 号码（用户的电话号码）

格式：国家代码 + 国家号码

示例：

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

来源：UE 上下文，通常通过 MME 从 HSS 获取

---

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（当前保留/为空）

描述：保留字段，用于 UE 时区信息

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：公共陆地移动网络标识符，编码为小端十六进制

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 位置数据

注意：这是为了向后兼容而使用的遗留编码格式

---

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 位置数据，从 MME 在创建会话请求中接收

用法：

- 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服务 UE 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 位置数据

用法：

- 标识特定的小区塔和扇区
- 用于切换和移动管理
- 细粒度位置数据

---

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 IP 地址，用于 PDN 连接

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��从 PGW-C 获取的 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 在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 在创建会话请求中接收

用法：

- 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：从 PGW-C 获取的承载 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：从 PGW-U 获取的 PFCP 流量测量（通过 URR 使用报告）

注意：

- 对于 `update` 事件为累计值
- 对于 `end` 事件为最终总值
- 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：从 PGW-U 获取的 PFCP 流量测量（通过 URR 使用报告）

注意：

- 对于 `update` 事件为累计值
- 对于 `end` 事件为最终总值
- 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**：单次更新的基本会话

时间线：

1. 承载建立
2. 5 分钟后：使用更新（下行 10 MB，上行 5 MB）
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**：具有多个更新的双栈会话

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: 具有专用承载的会话

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:

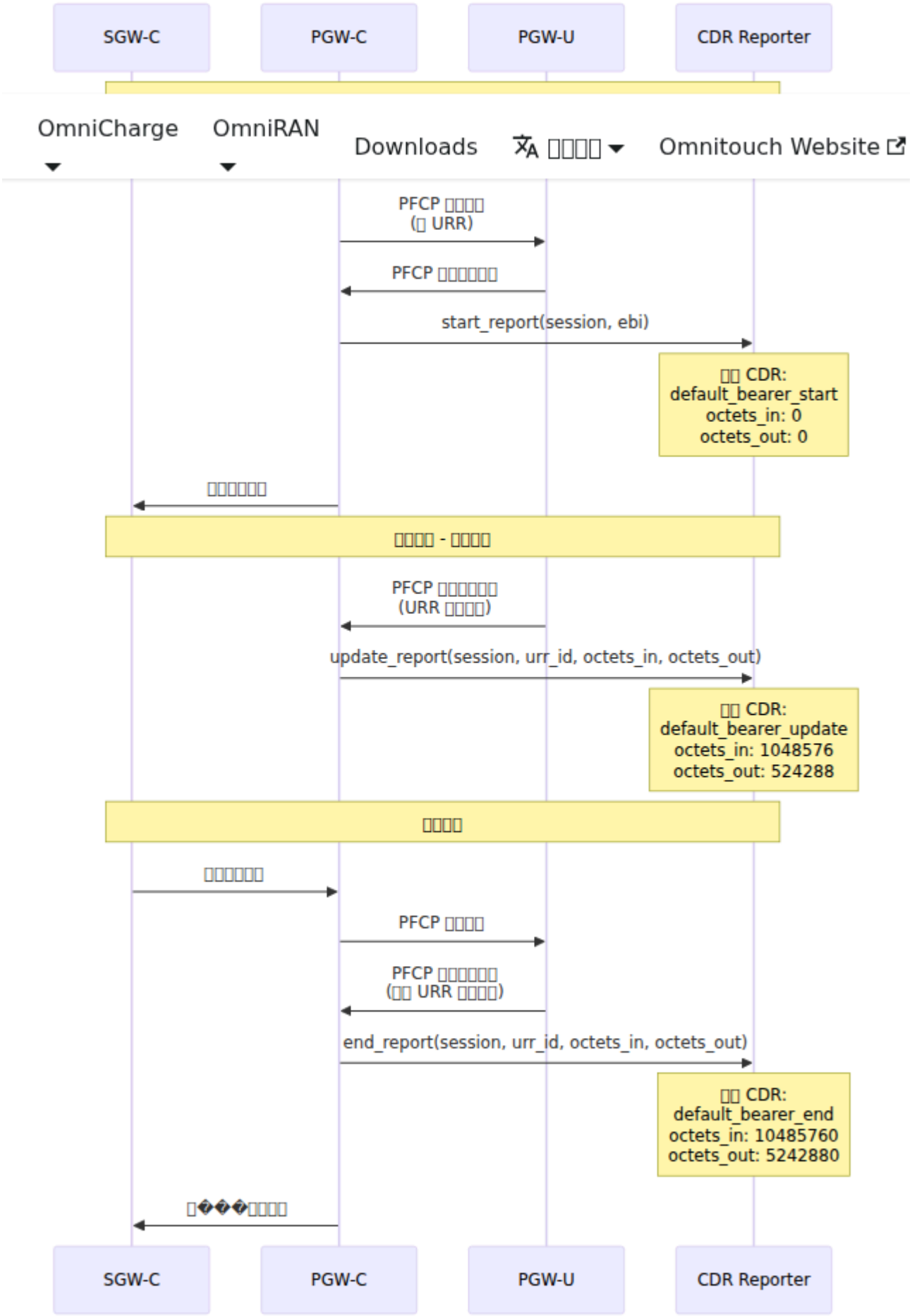
```
1726570800,310260333333333,default_bearer_start,10003,15553333333,333
1726571100,310260333333333,dedicated_bearer_start,10004,15553333333,3
1726571400,310260333333333,default_bearer_update,10003,15553333333,33
1726571400,310260333333333,dedicated_bearer_update,10004,15553333333,
1726571700,310260333333333,dedicated_bearer_end,10004,15553333333,333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

分析:

- 默认承载 (10003) 承载背景流量 (下行 10 MB, 上行 4 MB)
- 专用承载 (10004) 承载视频流量 (下行 200 MB, 上行 2 MB)
- 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
OmniCharge
OmniRAN
Downloads
Omnitouch Website
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:

```
# 监控 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:

```
# 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 URRs 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
- 直径 Gx 接口 - CDR 中 QCI 值的策略控制
- 直径 Gy 接口 - 在线充电集成

3GPP 参考

- TS 32.251 - 分组交换（PS）域充电
- 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

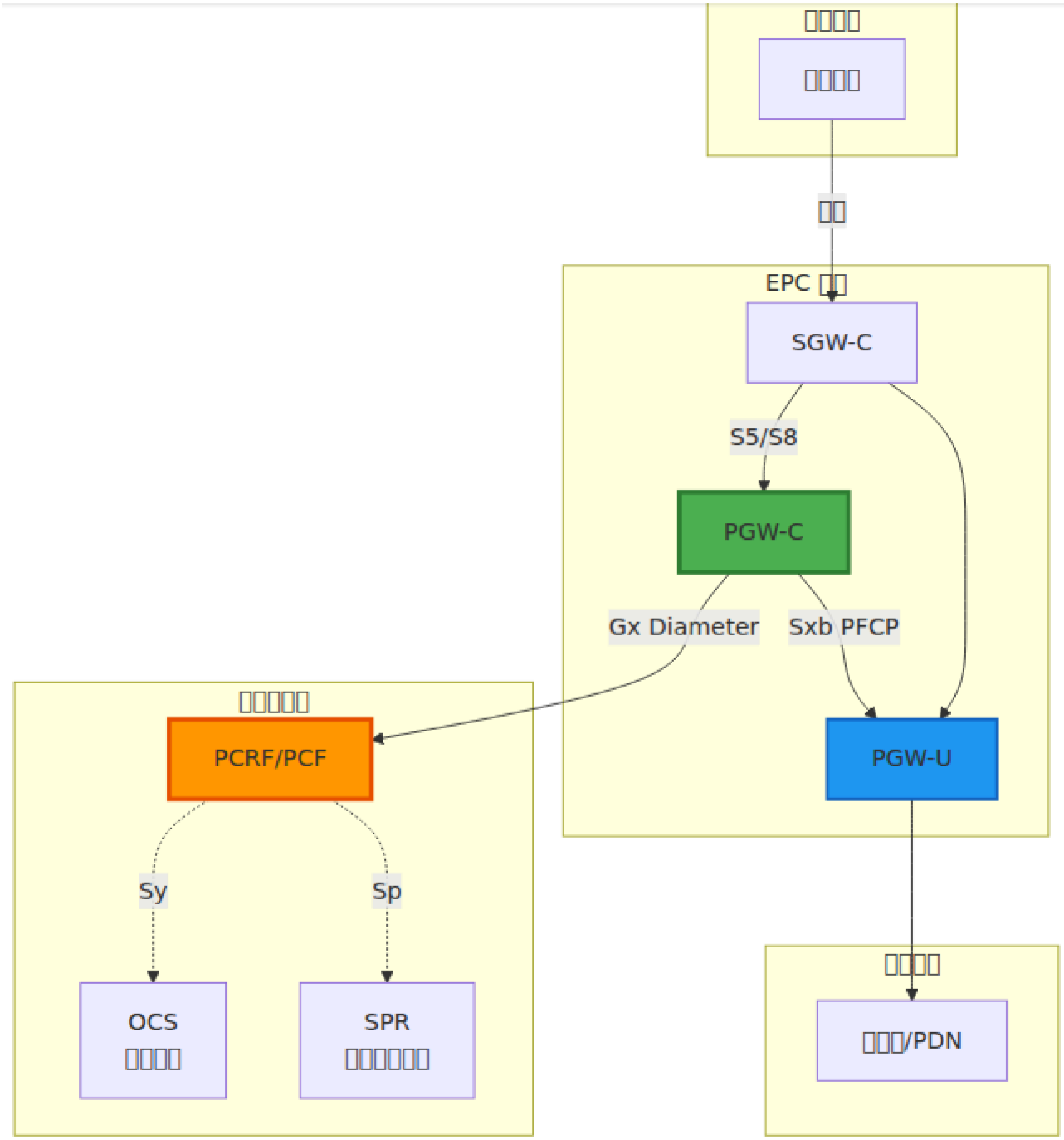
目录

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□置文件管理 - 每用户策略应用

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□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
- 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/响应对
- CCR（计费控制请求）/ CCA（计费控制响应）

结果代码：

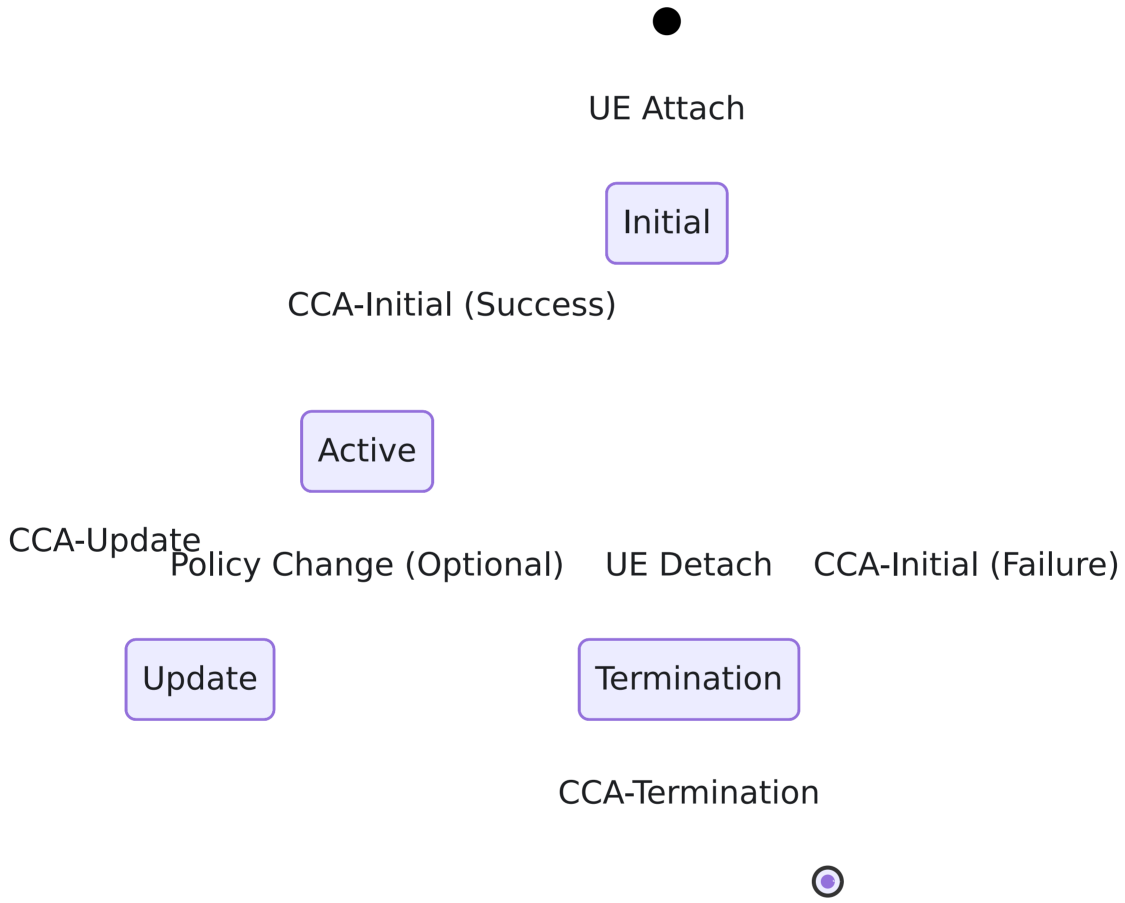
- `2001` - DIAMETER_SUCCESS
- `3xxx` - 协议错误
- `4xxx` - 瞬��故障

- `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
IP-CAN-Type	1027	Enumerated	5 = 3GPP-EPS
RAT-Type	1032	Enumerated	1004 = EUTRAN
QoS-Information	1016	Grouped	当前 QoS (AMBR)
Network-Request-Support	1024	Enumerated	网络发起的程序
Supported-Features	628	Grouped	Gx 功能列表

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（计费控制响应 - 初始）

发送方： PCRF 对 CCR-I 的响应

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	特定于供应商的结果代码
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/收费记录

与 **CCR-I** 的关键区别:

- `CC-Request-Type: TERMINATION_REQUEST (3)`
- 可能包括使用统计
- 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 对 CCR-T 的响应

目的:

- 确认会话终止
- 不返回策略规则

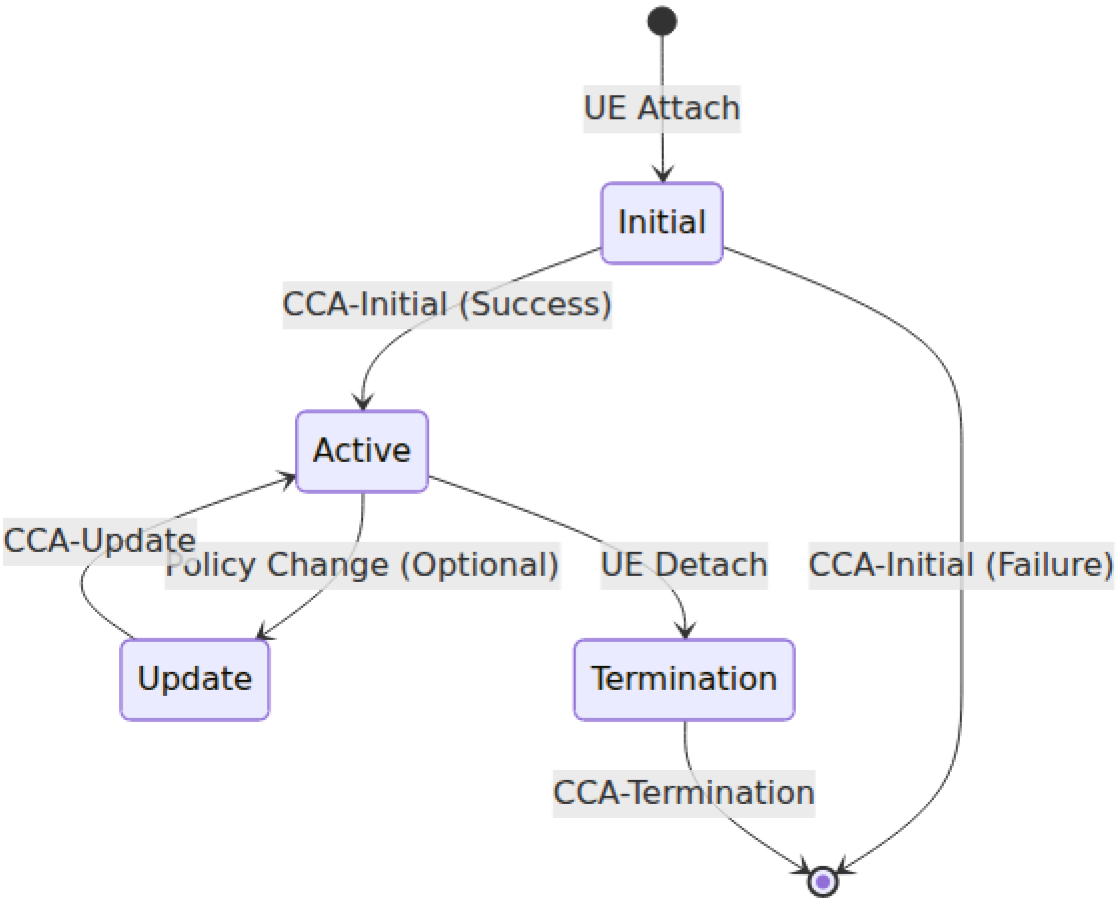
示例 **CCA-T**:

```
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□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：标识计费类别（由 OCS 使用 - 见 Diameter Gy 接口）
- 计量方法：基于流量、时间或事件
- 在线/离线计费： OCS（通过 Diameter Gy 的预付费）与离线 CDR（后付费 - 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□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 发送带有更新的 QER 的会话修改请求

□置

基本 Gx □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
- 请求根据可用性分发
- 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**（完全合格域名）：

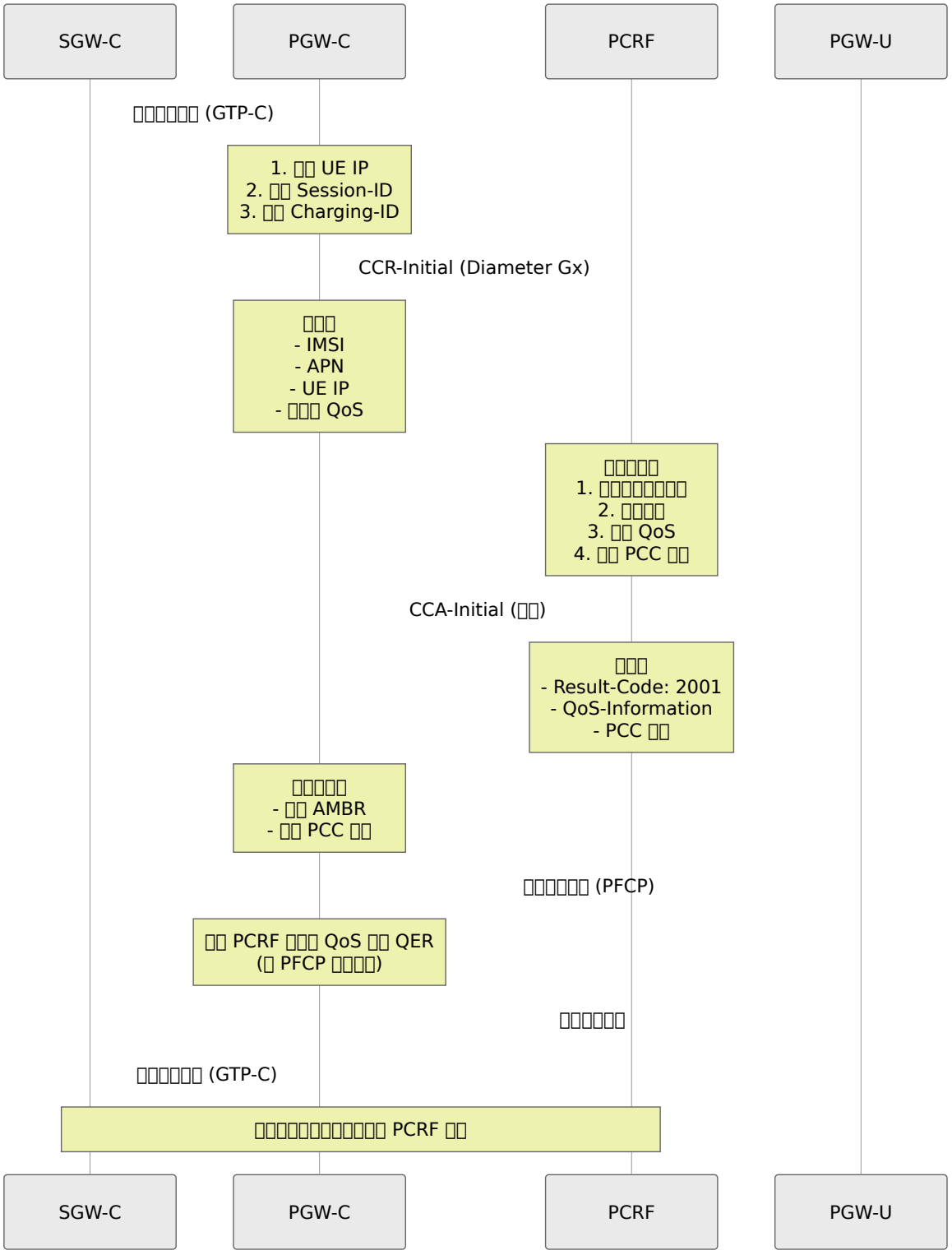
```
# 正确 - FQDN 格式
host: "pgw_c.epc.mnc999.mcc999.3gppnetwork.org"

# 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`

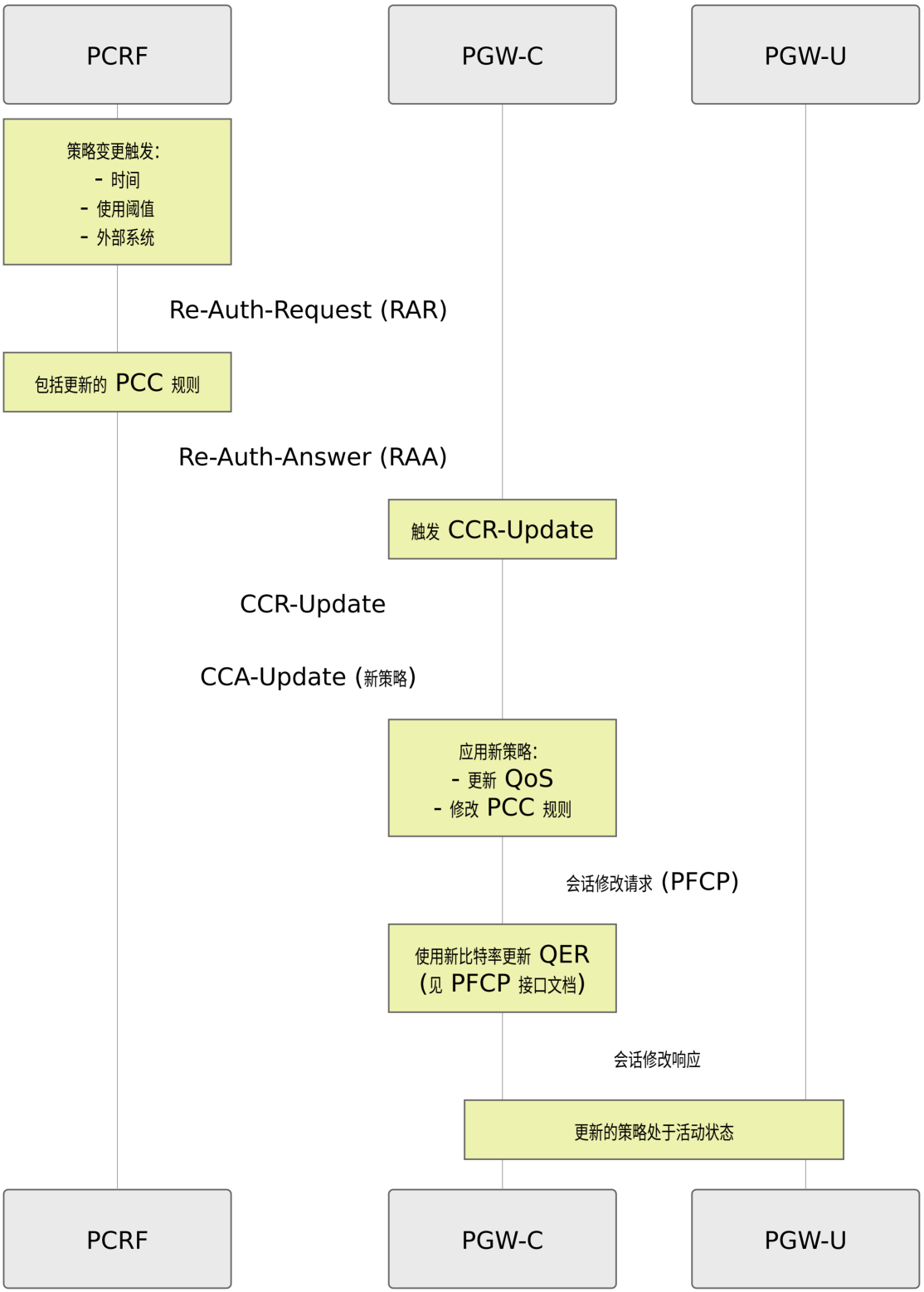
□□□

□□□□□□□□

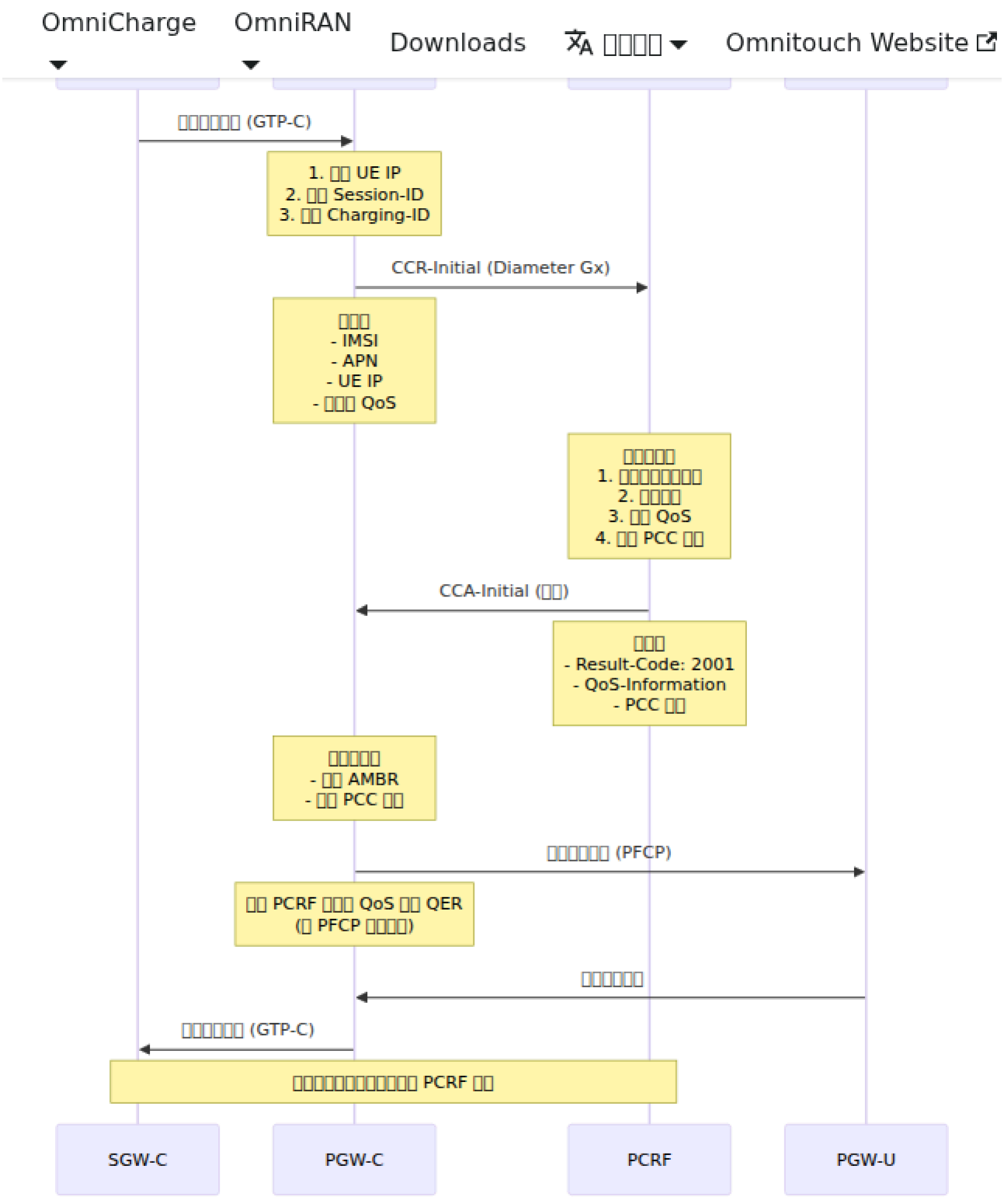


□□□ PCRF □ QoS □□□□□□□ QER□QoS □□□□□□□□□□ PFCP □□□ PGW-U□□□ QER □□□□□□□□□ □ PFCP □□□

策略更新（网络发起）


PCRF
PGW-C
PGW-U
策略变更触发:
- 时间
- 使用阈值
- 外部系统
Re-Auth-Request (RAR)
包括更新的 PCC 规则
Re-Auth-Answer (RAA)
触发 CCR-Update
CCR-Update
CCA-Update (新策略)
应用新策略:
- 更新 QoS
- 修改 PCC 规则
会话修改请求 (PFCP)
使用新比特率更新 QER
(见 PFCP 接口文档)
会话修改响应
更新的策略处于活动状态
PCRF
PGW-C
PGW-U

会话终止


OmniCharge
OmniRAN
Downloads
Omnitouch Website
(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

错误处理

结果代码

PGW-C 在 CCA 消息中处理各种 Diameter 结果代码:

成功代码:

代码	名称	操作
2001	DIAMETER_SUCCESS	继续会话建立

永久故障 **(5xxx)**:

代码	名称	**PGW-C** 操作
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** 操作
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

实验结果代码

特定于供应商的错误代码：

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

常见 **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：

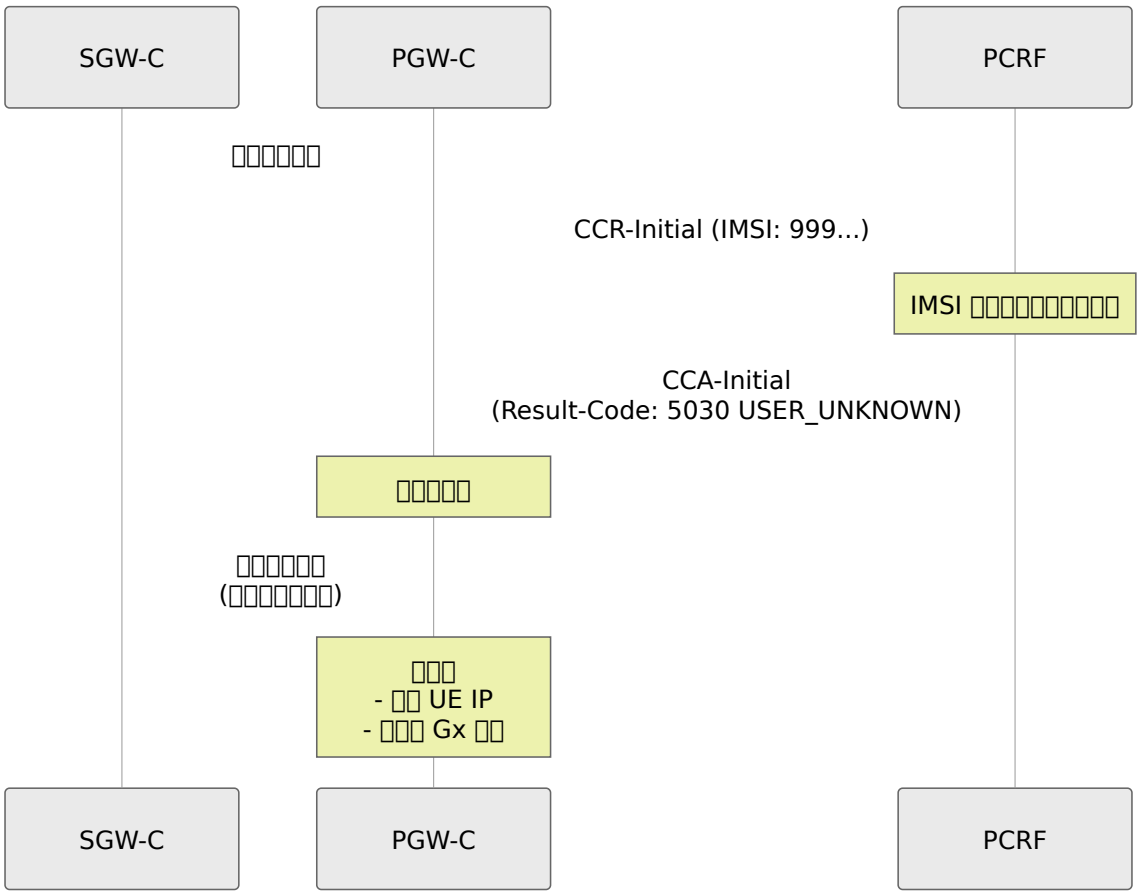
```
1. PGW-C 等待配置的超时（例如，5 秒）
2. 如果未收到 CCA：
   - 记录："Session-ID 的 CCR-Initial 超时: ..."
   - 向 SGW-C 响应错误原因
   - 清理分配的资源
3. SGW-C 接收：创建会话响应（原因：远程对等体未响应）
```

对 **SGW-C** 的错误响应：

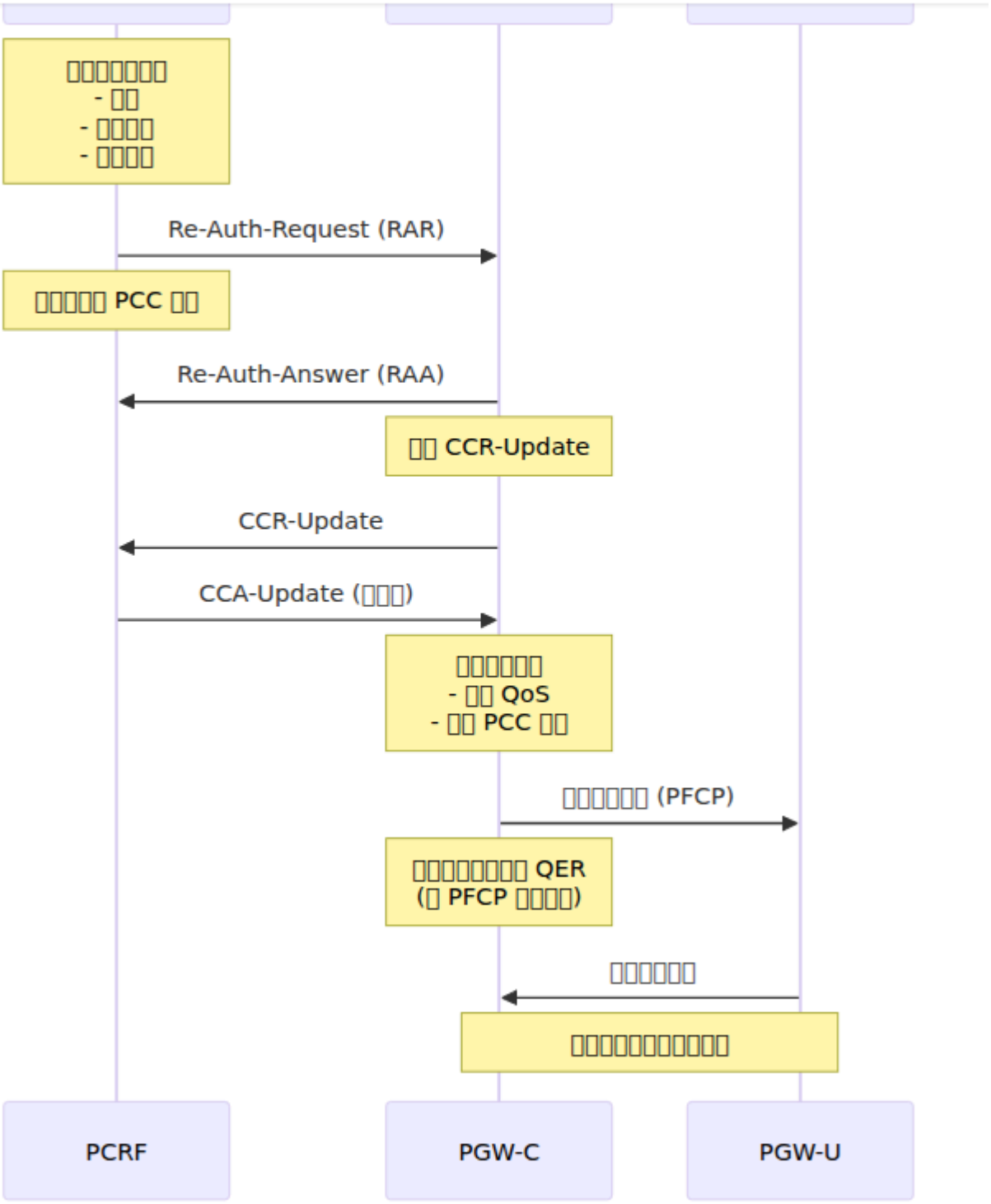
当 CCR-Initial 超时后，PGW-C 向 SGW-C 发送创建会话响应，原因代码为 `:remote_peer_not_responding`。

故障场景

场景 **1**：**PCRF** 拒绝会话（用户未知）


SGW-C
PGW-C
PCRF
□□□□□□□
CCR-Initial (IMSI: 999...)
IMSI □□□□□□□□□□□
CCA-Initial
(Result-Code: 5030 USER_UNKNOWN)
□□□□□
□□□□□□□
(□□□□□□□□)
□□□
- □□ UE IP
- □□□ Gx □□
SGW-C
PGW-C
PCRF

□□ 2□PCRF □□□□□


□□□□□□□□
- □□
- □□□□
- □□□□
Re-Auth-Request (RAR)
□□□□□ PCC □□
Re-Auth-Answer (RAA)
□□ CCR-Update
CCR-Update
CCA-Update (□□□)
□□□□□□□
- □□ QoS
- □□ PCC □□
□□□□□□□ (PFCP)
□□□□□□□□ QER
(□ PFCP □□□□)
□□□□□□□
□□□□□□□□□□□□
PCRF
PGW-C
PGW-U

故障排除

常见问题

1. Diameter 对等体连接失败

症状:

- 日志: “Diameter 对等体未连接”
- 未发送 CCR-Initial

可能原因:

- PCRF 无法访问
- 配置中 PCRF IP 不正确
- 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□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 过载
- 网络延迟
- PCRF 未响应此 Session-ID

解决方案：

1. 检查 PCRF 日志以获取错误
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 过于繁忙	重试或增加 PCRF 容量

检查日志：

```
# PGW-C 日志显示：
[error] Diameter Gx 错误：结果代码 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 是否使用正确的 MBR 值创建
    - 检查 PGW-U 日志以获取 QER 安装
3. 验证 **PCRF** 策略：
    - 检查 PCRF 配置

- 验证用户配置文件是否包含正确的 QoS

5. Diameter 路由问题

症状:

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

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） - 找不到用户

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"
    description: "PCRF 主机的失败率高"

# 会话拒绝的警报
- 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)
```

查找内容:

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

Web UI - Diameter 对等体监控

OmniPGW 包含一个实时 Web UI，用于监控 Diameter 对等体连接和状态。

Diameter 对等体页面

访问: `http://<omnipgw-ip>:<web-port>/diameter`

目的：实时监控 Diameter Gx 对等体与 PCRF 的连接

功能：

1. 对等体连接概述

- 连接计数 - 活动连接的 PCRF 对等体数量
- 未连接计数 - 配置但未连接的对等体数量
- 每 1 秒自动刷新（所有页面中最快的刷新）

2. 每个对等体状态信息 对于每个配置的 PCRF 对等体：

- 主机 - Diameter 身份 (Origin-Host)
- **IP** 地址 - PCRF IP
- 端口 - Diameter 端口（默认 3868）
- 状态 - 连接（绿色）/未连接（红色）
- 传输 - TCP 或 SCTP
- 连接发起 - 谁发起（PGW 或 PCRF）
- 领域 - Diameter 领域
- 产品名称 - PCRF 产品标识符（如果已广告）
- 应用 **ID** - 支持的 Diameter 应用（例如，Gx = 16777238）

3. 可展开的详细信息 单击任何对等体行以查看：

- 完整的对等体配置
- 功能交换（CER/CEA）详细信息
- 支持的功能
- 完整的连接状态

操作使用案例

监控 **PCRF** 连接性：

1. 在浏览器中打开 Diameter 页面
2. 验证所有 PCRF 对等体显示“已连接”
3. 检查连接发起是否与配置匹配
4. 验证应用 ID 包含 Gx（16777238）

故障排除会话创建失败（**Gx** 问题）：

1. 用户会话失败，出现“PCRF 超时”错误
2. 打开 Diameter 页面
3. 检查对等体状态：
   - 未连接？
     → 检查网络连接
     → 验证 PCRF 是否正在运行
     → 检查 TCP 3868 的防火墙规则
   - 已连接但会话失败？
     → 问题出在应用层（检查日志）
     → PCRF 可能拒绝用户

验证 **Diameter** 置：

1. 配置新的 PCRF 对等体后
2. 打开 Diameter 页面
3. 验证对等体出现在列表中
4. 检查状态是否更改为“连接”
5. 展开对等体以验证：
   - 领域与配置匹配
   - 应用 ID 包含 Gx
   - 产品名称显示 PCRF 标识符

监控故障转移：

场景：主 PCRF 故障
1. Diameter 页面显示主“未连接”
2. 验证备份 PCRF 仍“已连接”
3. 新会话自动使用备份
4. 当主恢复时，状态返回“已连接”

检测 **Diameter** 路由问题：

- 对等体显示“已连接”但领域错误
- 应用 ID 不包括 Gx（16777238）
- 产品名称与预期 PCRF 不匹配

识别□置不匹□：

```
Web UI 显示：
  连接发起：“对等体发起”
但配置说：
  initiate_connection: true

这表明：
  - OmniPGW 尝试连接
  - 但 PCRF 也在发起
  - 可能导致连接竞争条件
```

优点：

- 最快的刷新率 - 1 秒更新
- 可视连接状态 - 立即红/绿指示
- 无需 **Diameter** 工具 - 无需 Diameter CLI 工具
- 对等体□置可见 - 无需检查配置文件即可验证设置
- 应用级详细信息 - 查看支持的 Diameter 应用
- 领域验证 - 确认 Diameter 路由配置

与指标集成

虽然 Web UI 提供实时状态，但结合 Prometheus 进行：

- 历史 Gx 错误率
- CCR/CCA 消息计数
- 延迟趋势

Web UI = “现在是否正常工作？”指标 = “它在一段时间内的工作情况如何？”

相关文档

□置和策略

- □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 - 与策略集成的会话生命周期
- **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** □置 - IMS 策略控制的 P-CSCF 交付

操作

- 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

返回操作指南

Diameter 在线计费（Gy/Ro 接口）

在线计费系统（OCS）接口

目录

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）以实现实时信用控制。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

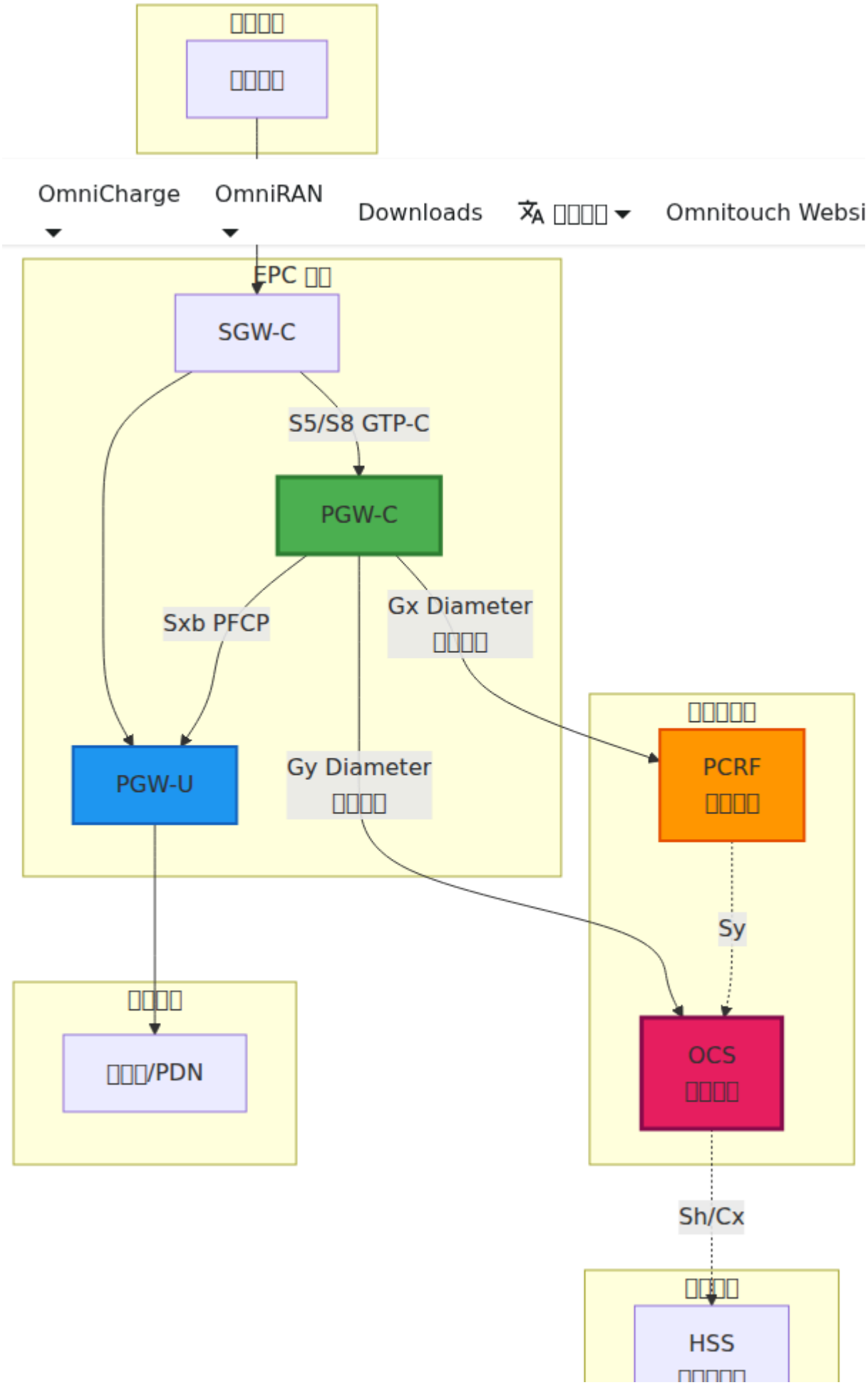
在线与离线计费

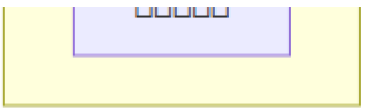
方面	在线计费 **(Gy/Ro)**	离线计费 **(Gz/Rf)**
时机	实时，在服务之前	服务交付后
用例	预付费用户	后付费用户
信用检查	是，在授予服务之前	否，账单稍后生成
系统	OCS（在线计费系统）	CGF/CDF（计费数据功能）
风险	无收入损失	有未付款账单的风险
复杂性	高（实时要求）	较低（批处理）
用户影响	如果没有信用则拒绝服务	服务始终可用

另见：数据 CDR 格式 用于离线计费记录（后付费账单）

另见：会话管理 包括计费集成的完整 PDN 会话生命周期

□□□□□□ Gy


□□□□
□□□□
OmniCharge
OmniRAN
Downloads
□□□□
Omnitouch Websi
EPC □□
SGW-C
S5/S8 GTP-C
PGW-C
Sxb PFCP
Gx Diameter
□□□□
PGW-U
Gy Diameter
□□□□
□□□□□
PCRF
□□□□
Sy
□□□□
□□□/PDN
OCS
□□□□
Sh/Cx
□□□□
HSS

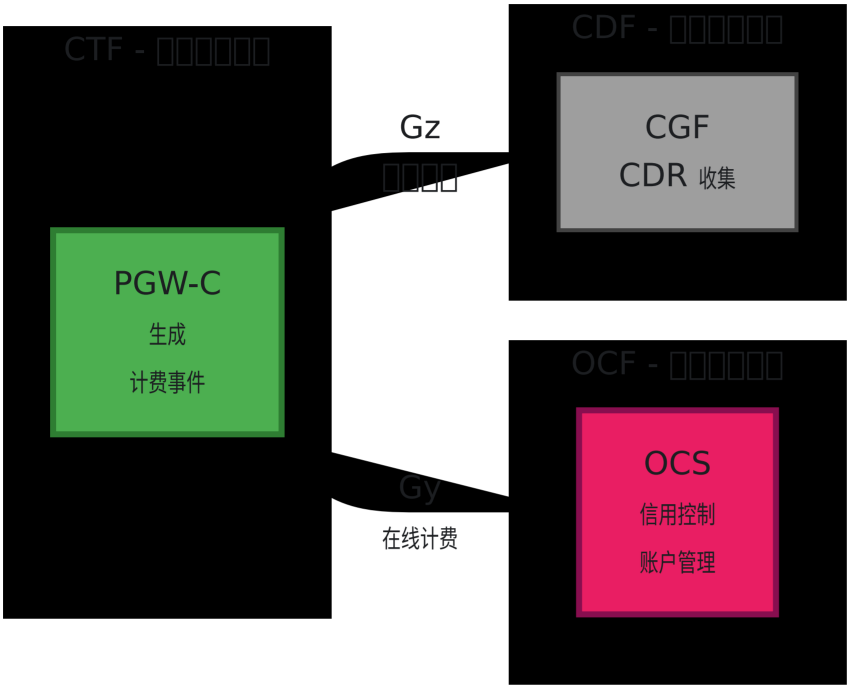


关键功能

功能	描述
信用授权	在允许流量之前向 OCS 请求配额
□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□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 - 为授予的配额保留信用

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。该会话:

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

会话 ID 格式

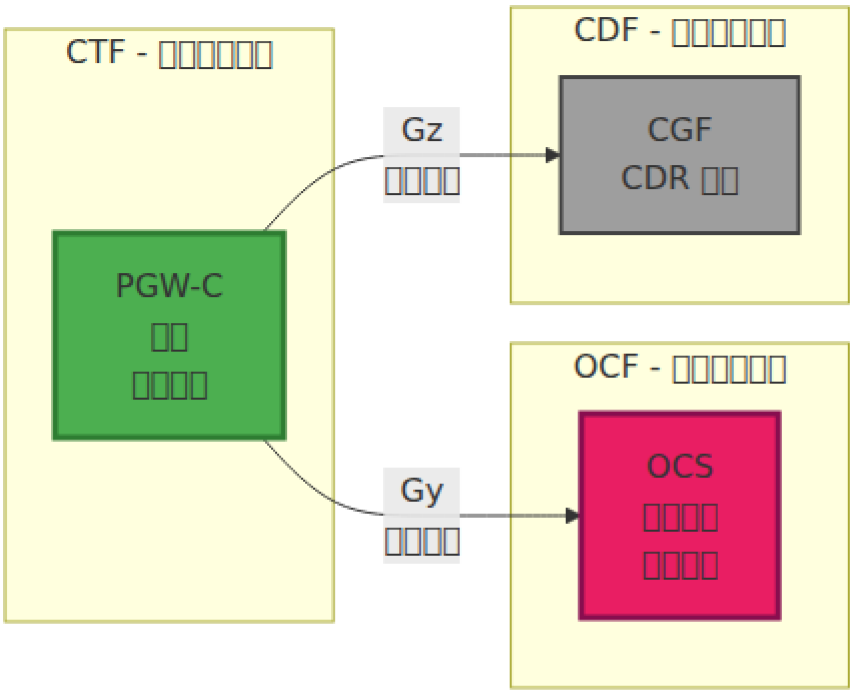
```
Session-ID: <Origin-Host>;<high32>;<low32>[;<optional>]
示例: omni-
pgw_c.epc.mnc999.mcc999.3gppnetwork.org;9876543210;12345;gy
```

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:** 额外标识符（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 领域
Destination-Realm	283	DiamIdent	OCS 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)
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	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 (分组)
│   ├── Subscription-Id-Type: END_USER_IMSI (1)
│   └── Subscription-Id-Data: "310260123456789"
├── Subscription-Id (分组)
│   ├── Subscription-Id-Type: END_USER_E164 (0)
│   └── Subscription-Id-Data: "15551234567"
├── Service-Context-Id: "32251@3gpp.org"
├── Multiple-Services-Credit-Control (分组)
│   ├── Service-Identifier: 1
│   ├── Rating-Group: 100
│   └── Requested-Service-Unit (分组)
│       └── CC-Total-Octets: 10000000 (请求 10 MB)
└── Used-Service-Unit (分组)
    └── 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	每服务结果代码
Final-Unit-Indication	430	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 (分组)
    ├── Result-Code: DIAMETER_SUCCESS (2001)
    ├── Service-Identifier: 1
    ├── Rating-Group: 100
    ├── Granted-Service-Unit (分组)
    │   └── 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，使用 80%）
- 有效期到期
- 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的使用情况
- 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 (分组)
    ├── Service-Identifier: 1
    ├── Rating-Group: 100
    ├── Used-Service-Unit (分组)
    │   └── CC-Total-Octets: 8000000 (到目前为止使用了 8 MB)
    └── Requested-Service-Unit (分组)
        └── 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. 授予更多额:

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

2. 最终额（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）
- 包含 `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 (分组)
    ├── Service-Identifier: 1
    ├── Rating-Group: 100
    └── Used-Service-Unit (分组)
        └── 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
- 完成计费
- 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 可以以不同的单位授予配额：

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
服务特定	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

□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 PGW-U 的 PFCP 会话报告监控使用情况（见 PFCP 接口）。

示例流程：

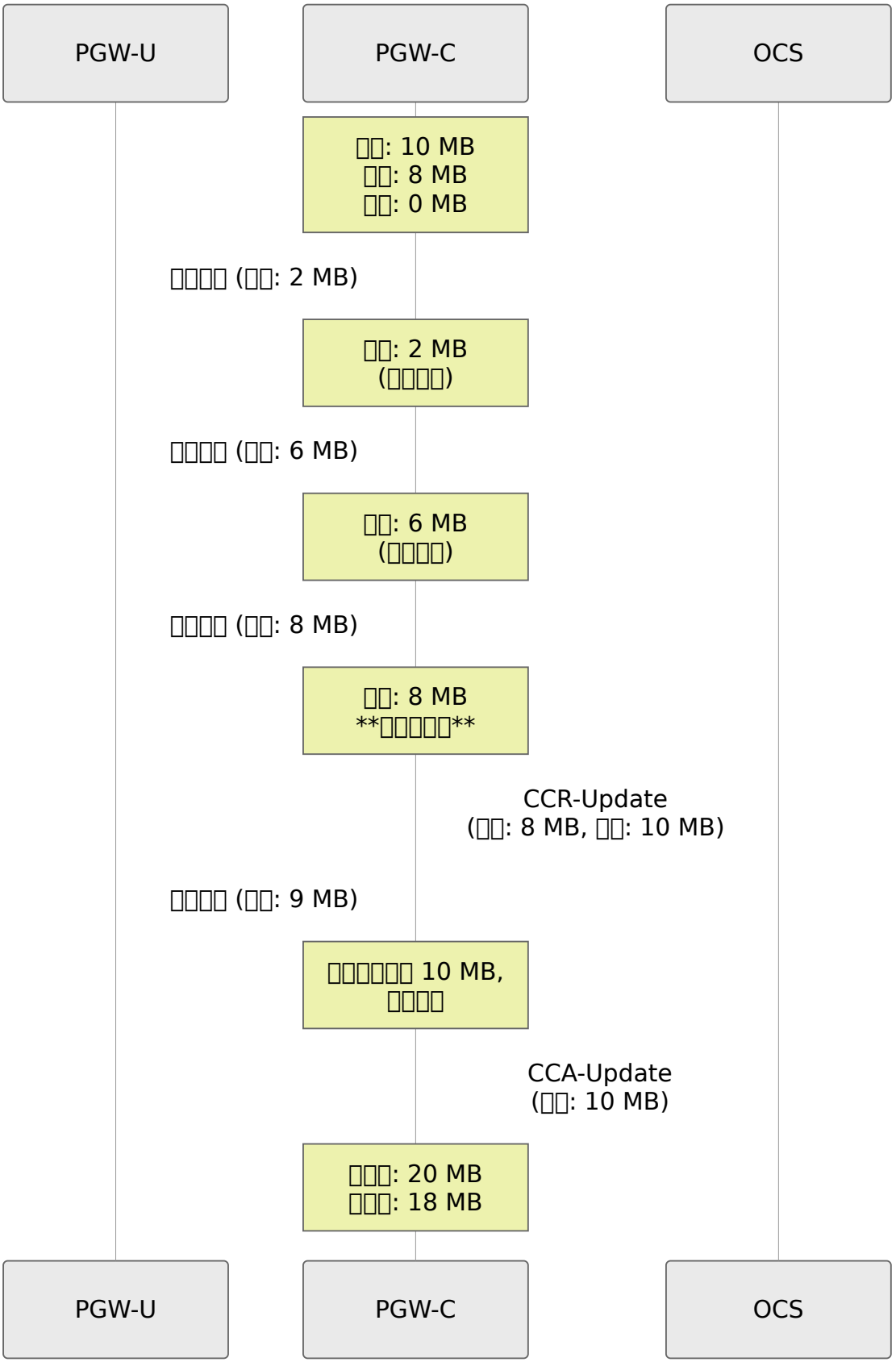
1. OCS 授予 10 MB 配额，阈值为 80%（8 MB）
2. PGW-C 通过 PGW-U 使用报告监控使用情况（PFCP 会话报告）
3. 当使用达到 8 MB：
   → PGW-C 发送 CCR-Update
   → 继续允许流量（不等待响应）
4. OCS 响应更多配额
5. 如果在发送 CCR-Update 之前配额耗尽：
   → PGW-C 必须阻止流量

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：2 MB（允许 OCS 响应的时间）
```

PGW-C 监控：

PGW-C 通过 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 中包括 **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 (分组)
            ├── 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 规则以重定向 HTTP 到充值 URL
3. **RESTRICT_ACCESS:** 安装 PFCP 规则，仅允许白名单 IP

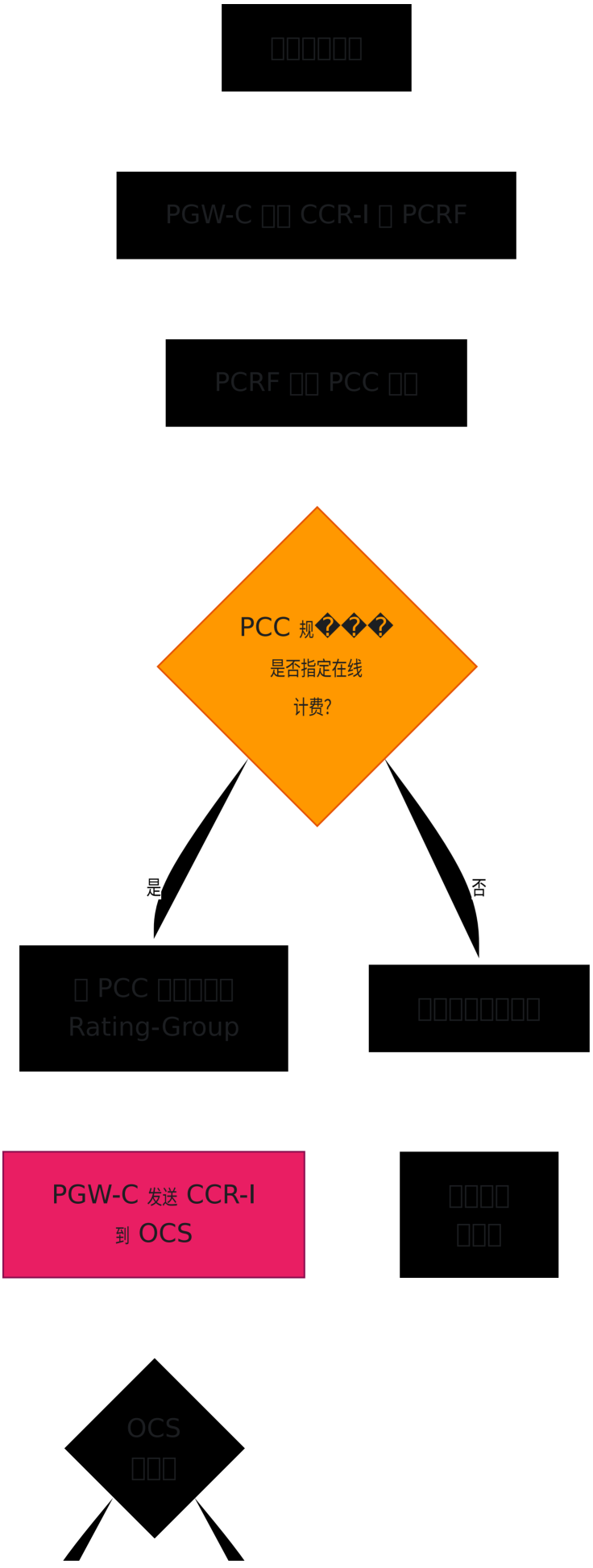
承载计费控制

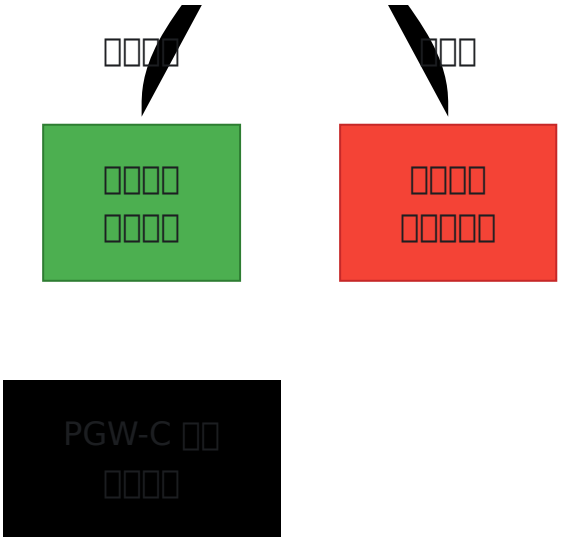
什么控制承载是否收费?

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 以获取完整的 PCC 规则文档。

计费决策流程:


PGW-C CCR-I PCRF
PCRF PCC
PCC 规
是否指定在线
计费?
是
否
PCC
Rating-Group
PGW-C 发送 CCR-I
到 OCS
OCS



□□□□□□□□ PCC □□

PCRF □□ (Gx □□ CCA-I)□

```
CCA (Gx □□)
└── Charging-Rule-Definition (□□)
    ├── 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���类别（映射到 OCS 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=计费组	使用报告的粒度

承载计费决策矩阵

在线	离线	**Rating-Group**	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（融合）
0	0	-	无计费（免费服务）

多个计费组

单个 PDN 连接可以有 多个承载和不同的计费组：

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**，包含多个 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:

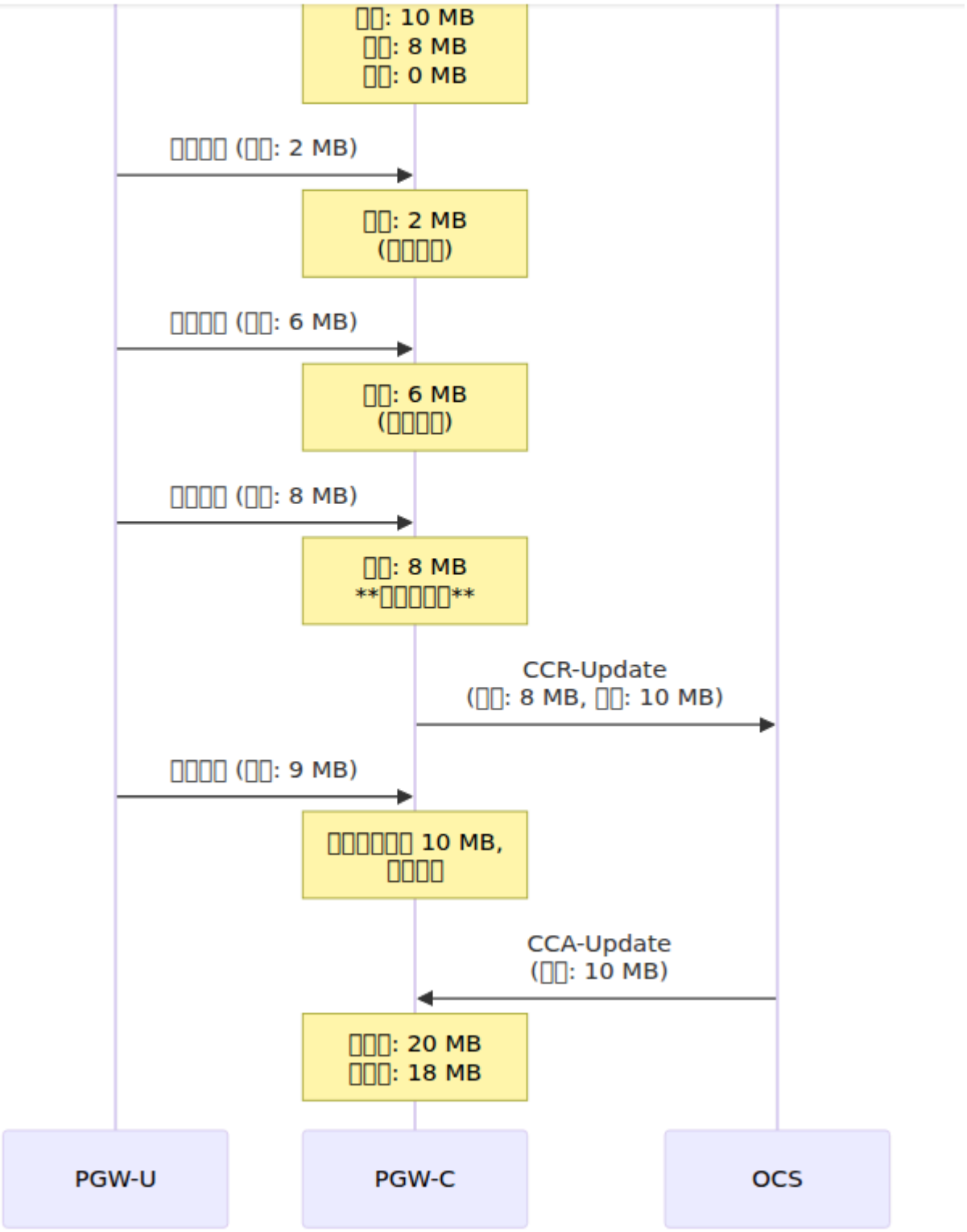
```
CCA-Initial
└── Multiple-Services-Credit-Control
    ├── [Rating-Group: 100] → 授予: 10 MB
    ├── [Rating-Group: 200] → 授予: 5 MB (视频更贵)
    └── [Rating-Group: 300] → 授予: 60 秒
```

按服务计费执行

PGW-C 跟踪每个 **Rating-Group** 的□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
}
```

按承载监控使用情况：


□□: 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 (分组, AVP 456)
├── Service-Identifier (Unsigned32, AVP 439)
├── Rating-Group (Unsigned32, AVP 432)
├── Requested-Service-Unit (分组, 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 (分组, AVP 446)
│   └── [与 Requested-Service-Unit 结构相同]
├── Granted-Service-Unit (分组, AVP 431)
│   └── [与 Requested-Service-Unit 结构相同]
├── Validity-Time (Unsigned32, AVP 448)
├── Result-Code (Unsigned32, AVP 268)
└── Final-Unit-Indication (分组, 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	必需用于计费
映射	可能映��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 (语音通话 - $0.10/分钟)
```

□置

基本 **Gy** □置

编辑 `config/runtime.exs`:

```
config :pgw_c,
  online_charging: %{
    # 全局启用或禁用在线计费
    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
      }
    ]
  }
```

置参数说明

`enabled`

- `true`: 在线计费激活，CCR 消息发送到 OCS
- `false`: 在线计费禁用，不发送 Gy 消息

`timeout_ms`

- 等待 OCS 的 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%)
- 较高 = 消息较少，但有配额耗尽的风险
- 较低 = 消息较多，但更安全

`timeout_action`

- `:block` - 如果 OCS 不响应则阻止流量（更安全，防止收入损失）
- `:allow` - 如果 OCS 不响应则允许流量（更好的用户体验，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□置

生产环境（预付费用户）：

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

测试/开发:

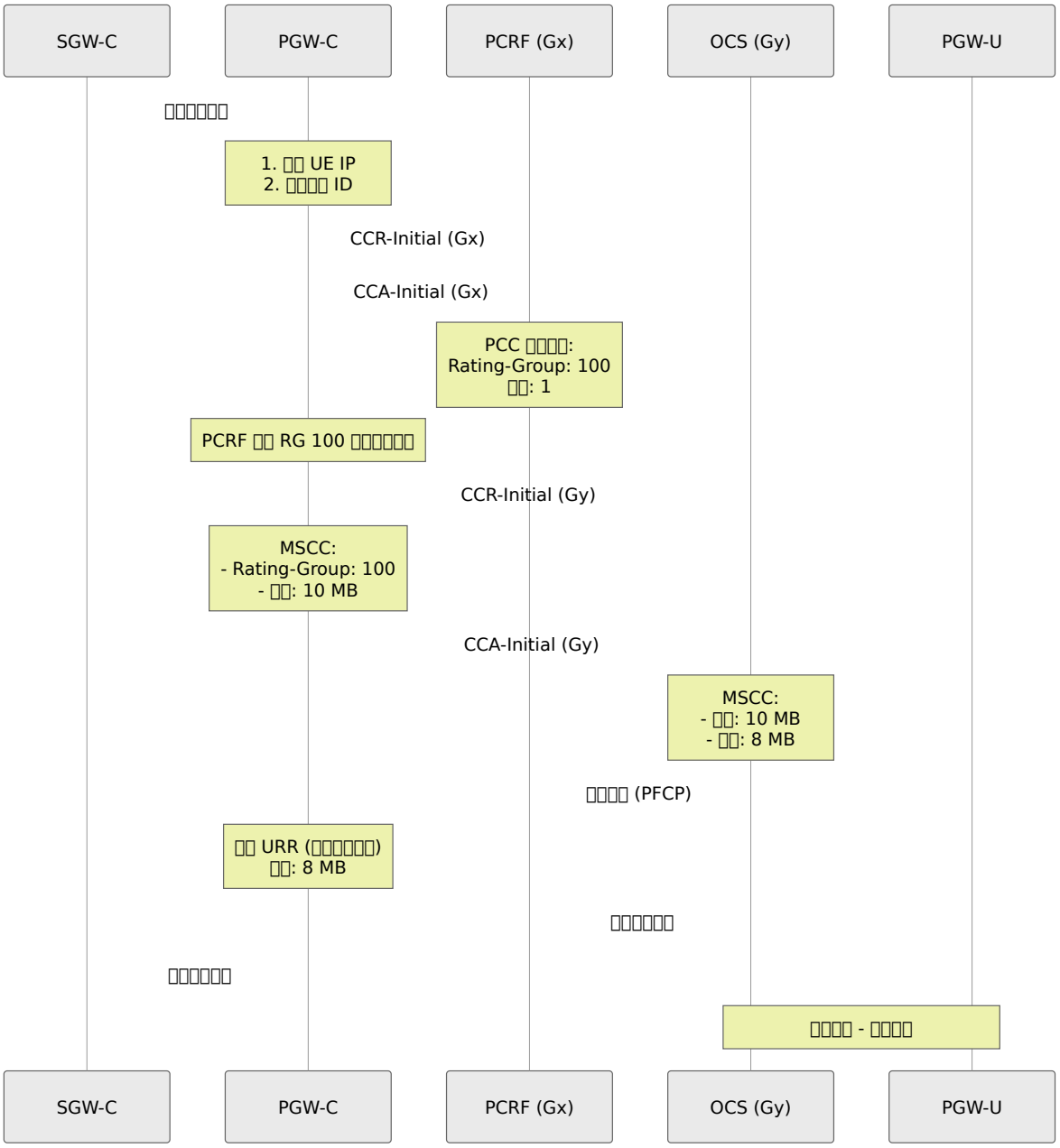
```
config :pgw_c,
  online_charging: %{
    enabled: false  # 测试时禁用
  }
```

混合（部分预付费，部分后付费）：

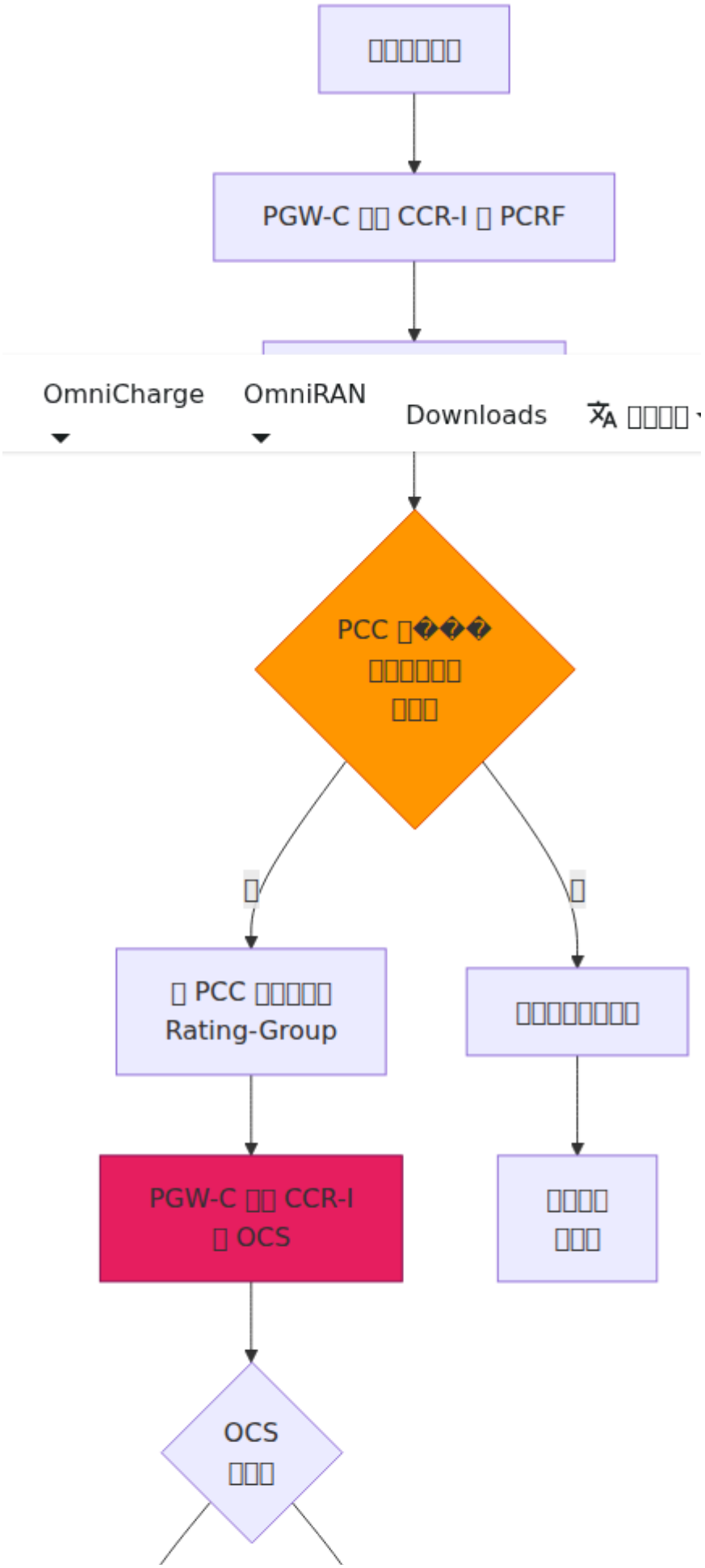
```
config :pgw_c,
  online_charging: %{
    enabled: true,  # 由 PCRF 控制每个用户
    timeout_action: :allow,  # OCS 失败时不阻止后付费
    no_credit_action: :terminate
  }
```

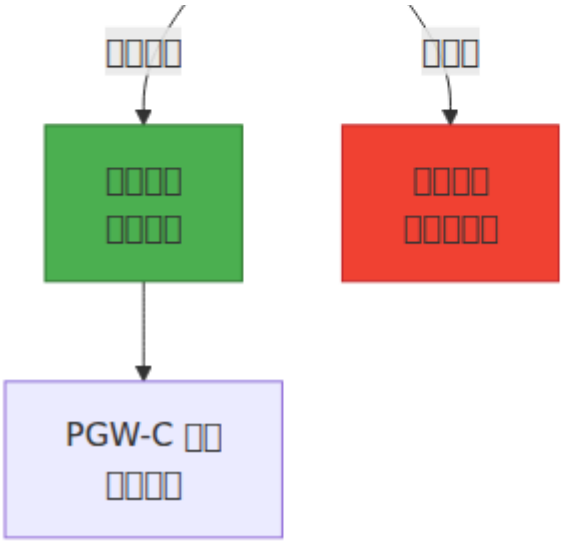
□□□□

□□□□□□□□□□

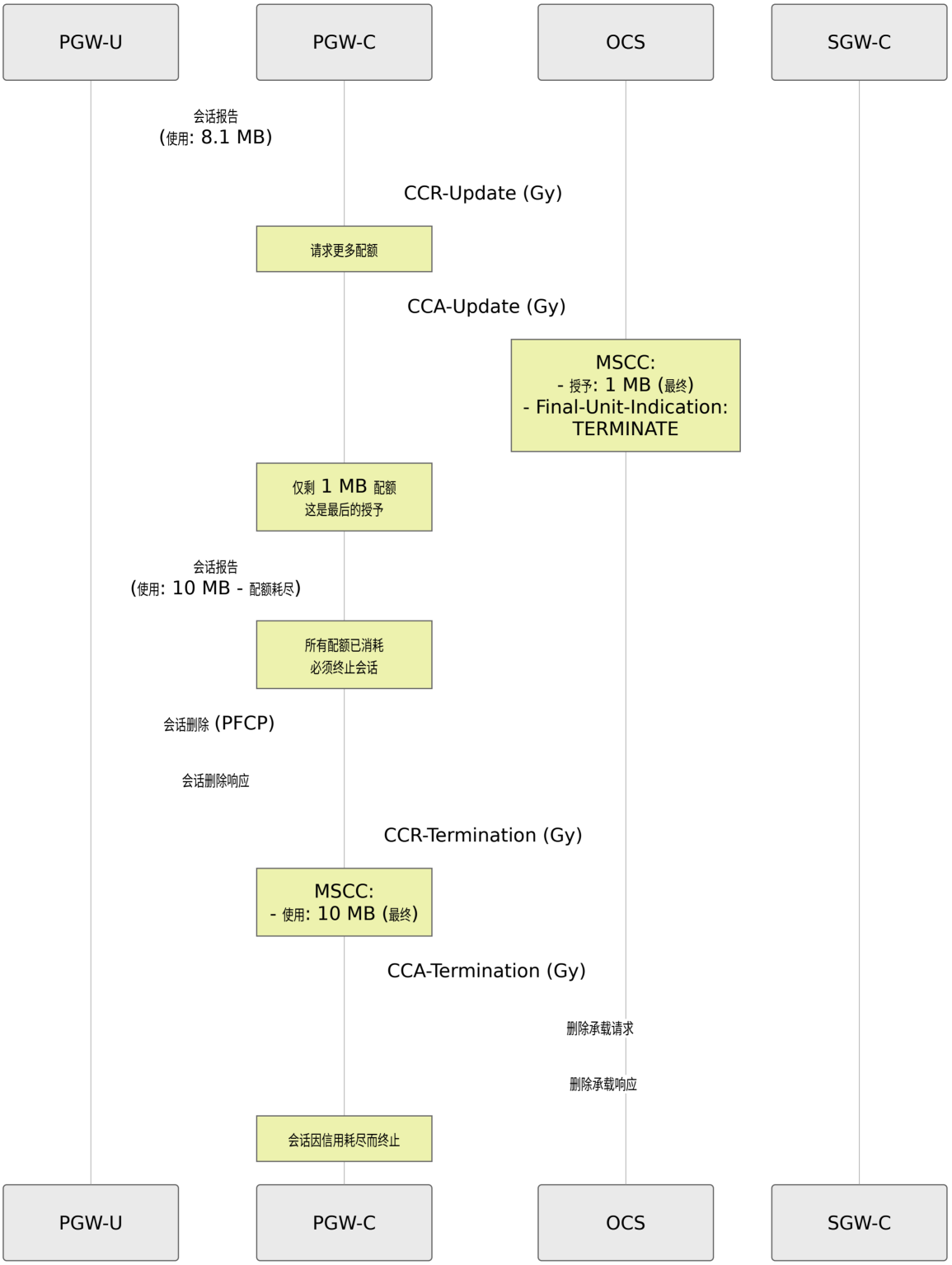

SGW-C
PGW-C
PCRF (Gx)
OCS (Gy)
PGW-U
□□□□□□□
1. □□ UE IP
2. □□□□ ID
CCR-Initial (Gx)
CCA-Initial (Gx)
PCC □□□□:
Rating-Group: 100
□□: 1
PCRF □□ RG 100 □□□□□□
CCR-Initial (Gy)
MSCC:
- Rating-Group: 100
- □□: 10 MB
CCA-Initial (Gy)
MSCC:
- □□: 10 MB
- □□: 8 MB
□□□□ (PFCP)
□□ URR (□□□□□□□)
□□: 8 MB
□□□□□□□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

□□□□□□ (CCR-Update)

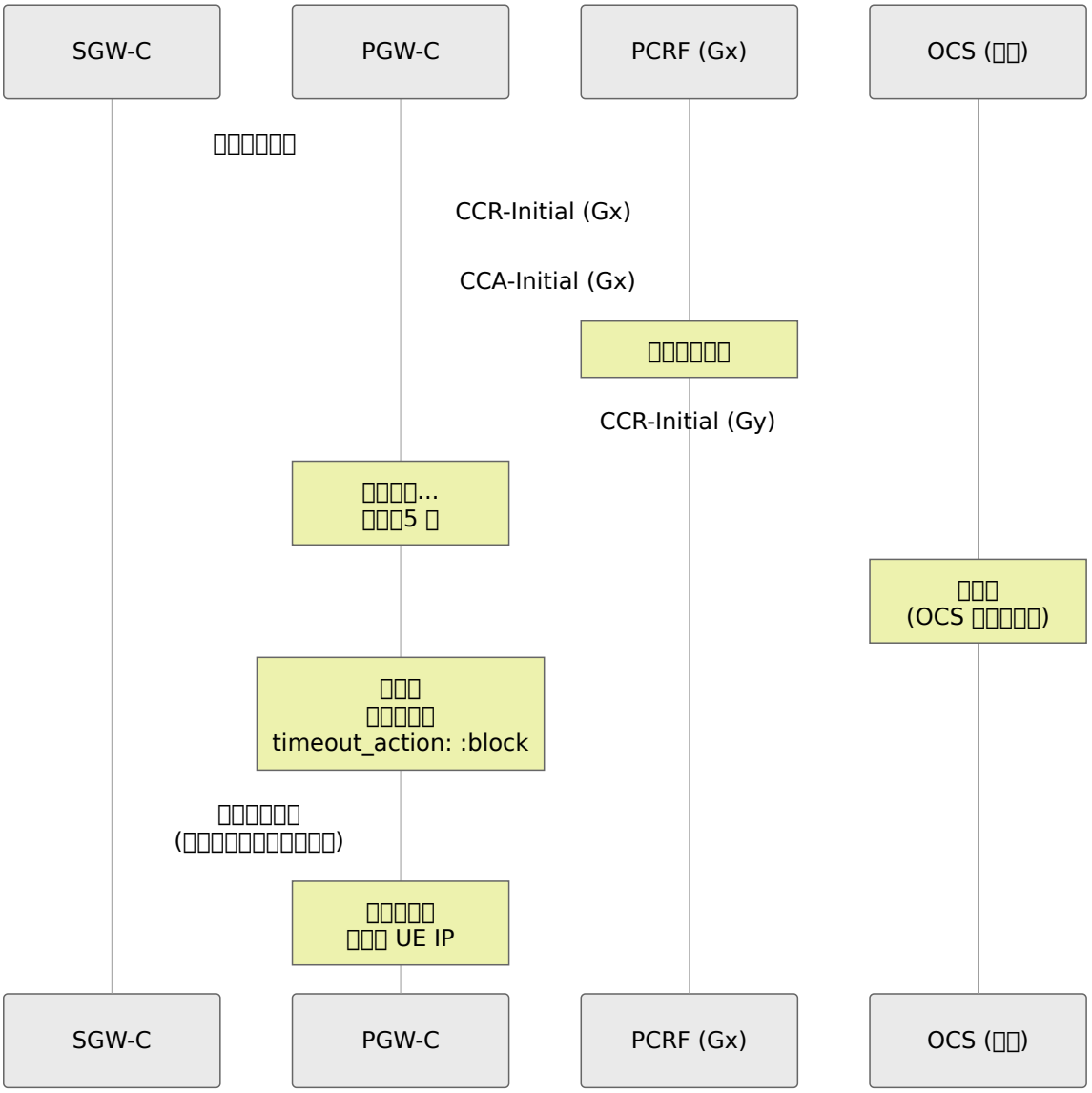

□□□□□□□
PGW-C □□ CCR-I □ PCRF
OmniCharge
OmniRAN
Downloads
□□□□
PCC □□□
□□□□□□□
□□□
□
□
□ PCC □□□□□
Rating-Group
□□□□□□□□□□
PGW-C □□ CCR-I
□ OCS
□□□□
□□□
OCS
□□□


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:
- 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	操作
2001	DIAMETER_SUCCESS	继续授予的配额

临时故障 **(4xxx)**:

代码	名称	**PGW-C** 操作
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** 操作
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
```

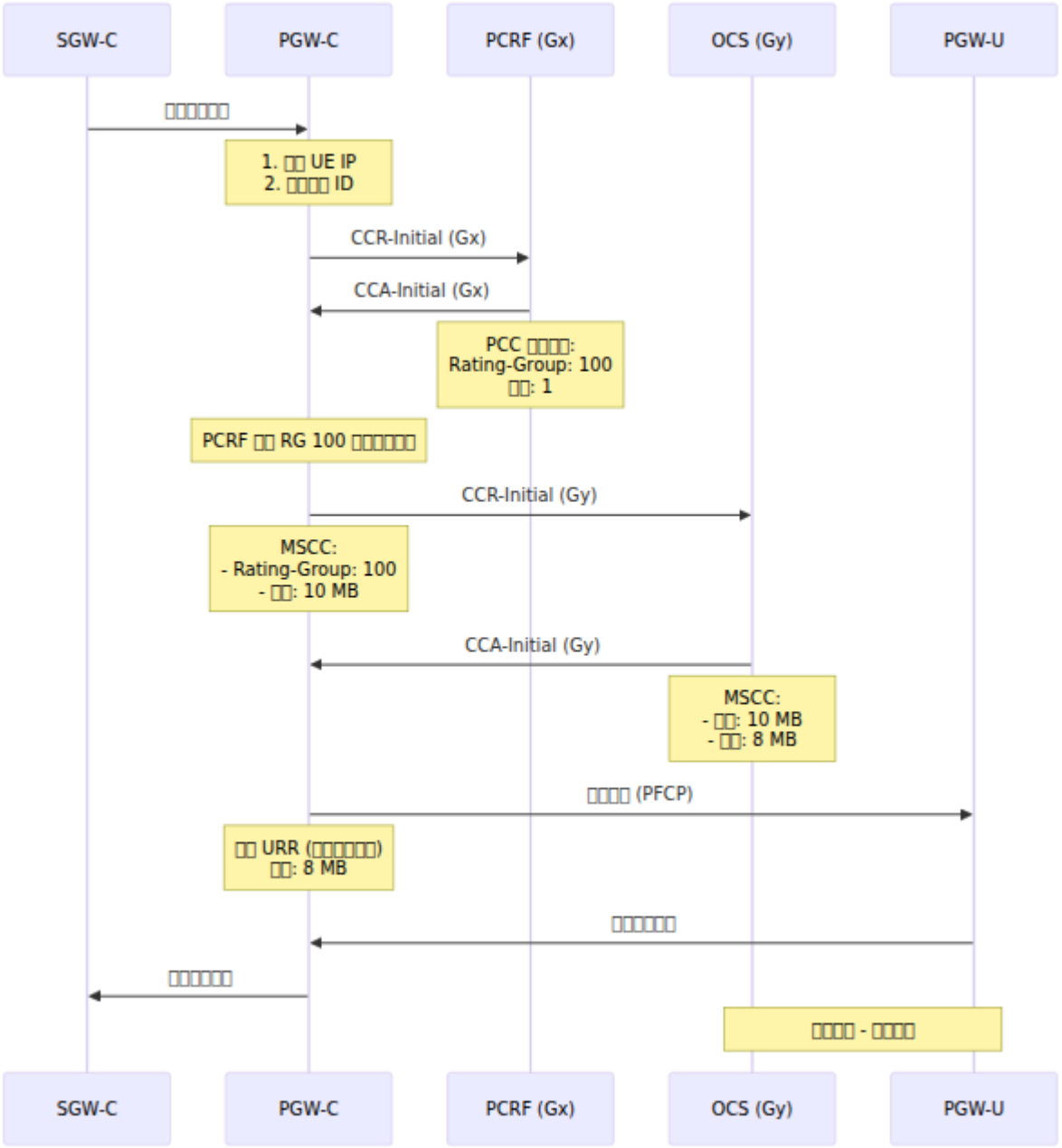
PGW-C 行为:

- 允许 Rating-Group 100 的流
- 阻止 Rating-Group 200 的流量

与 **Gx** 的集成

Gx 接口（PCRF 策略控制）决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 以获取完整的策略控制文档。

Gx □ Gy □□□


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

□□□□

1. □□□□□

```
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 Rating-Group?
     - 在线 = 1 吗?
     ↓
如果是:
     发送 CCR-I 到 OCS (Gy) 及 Rating-Group
     ↓
     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 在 Gx 上发送 RAR (重新授权请求)
     ↓
新的 PCC 规则添加，在线=1, Rating-Group=200
     ↓
PGW-C 发送 CCR-U 到 OCS (Gy)
     - 为 Rating-Group 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 不可达
- 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□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 Rating-Group

症状:

- OCS 拒绝，显示“未知的 Rating-Group”
- 会话失败

原因:

- CCR-I 中的 Rating-Group 与 OCS 配置不匹配
- PCRF 提供了无效的 Rating-Group

解决方案:

1. 验证 PCRF 中 PCC 规则的 Rating-Group
2. 检查 OCS 配置以获取有效的 Rating-Group
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"
```

Web UI - Gy 信用控制模拟器

OmniPGW 包含一个内置的 Gy/Ro 模拟器，用于测试在线计费功能，而无需外部 OCS。

访问: `http://<omnipgw-ip>:<web-port>/gy_simulator`

目的：测试和模拟预付费用户的在线计费场景

功能：

1. 请求参数

- **IMSI** - 用户身份（例如，“310170123456789”）
- **MSISDN** - 电话号码（例如，“14155551234”）
- 请求单位 - 要求的配额量（以字节为单位）
- 服务 **ID** - 服务类型标识符
- 计费组 - 计费类别

2. CCR-I 模拟

- 发送 CCR-Initial（信用控制请求初始）
- 模拟会话建立期间的初始配额请求
- 测试 OCS 集成而无需实时流量

3. 用例

- 开发测试 - 在开发期间测试 Gy 接口
- **OCS** 集成 - 验证 OCS 连接和��应
- 额测试 - 测试不同的配额场景

- 故障排除 - 调试计费问题
- 演示 - 向利益相关者演示在线计费

使用方法：

1. 输入用户详细信息（IMSI、MSISDN）
2. 设置请求单位（例如，1000000 表示 1 MB）
3. 配置服务 ID 和计费组
4. 点击“发送 CCR-I”
5. 查看 OCS 响应和授予的配额

好处：

- 测试期间无需外部 OCS
- 快速验证计费逻辑
- 安全的测试环境
- 适用于培训和演示

相关文档

计费和策略

- **Diameter Gx** 接口 - PCRF 策略控制，触发在线计费的 PCC 规则
- 数据 **CDR** 格式 - 离线计费记录用于后付费账单
- □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

操作

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

返回操作指南

OmniPGW 监控与指标指南

Prometheus 集成与操作监控

□ Omnitouch □□□□□□

目录

概述

OmniPGW 提供两种互补的监控方法：

1. 实时 **Web UI**（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
- PFCP 对等状态
- Diameter 对等连接
- 单个会话检查

2. Prometheus 指标（本文档的主要重点）

- 历史趋势和分析
- 告警和通知

- 性能指标
- 容量规划

本文档重点关注 **Prometheus** 指标。有关 Web UI 的详细信息，请参见：

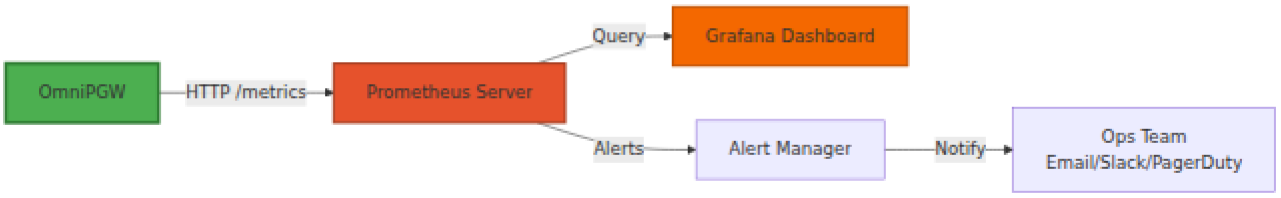
- 会话管理 - Web UI
- PFCP 接口 - Web UI
- Diameter Gx - Web UI

Prometheus 指标概述

OmniPGW 暴露 **Prometheus** 兼容指标，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
- 告警 - 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

□置

在 `config/runtime.exs` 中启用指标：

```
config :pgw_c,
  metrics: %{
    enabled: true,
    ip_address: "0.0.0.0",  # 绑定到所有接口
    port: 9090,               # HTTP 端口
    registry_poll_period_ms: 5_000  # 轮询间隔
  }
```

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 暴露:

```
# 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）
`session_id_registry_count`	Gauge	活动 Gx 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 PFCP 响应（等待响应）
`s5s8_sequence_number_registry_count`	Gauge	待处理 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])
```

消息计数器

S5/S8 (GTP-C) 消息：

指标名称	类型	标签	描述
`s5s8_inbound_messages_total`	Counter	`message_type`	总入站 S5/S8 消息
`s5s8_outbound_messages_total`	Counter	`message_type`	总出站 S5/S8 消息
`s5s8_inbound_errors_total`	Counter	`message_type`	S5/S8 处理错误

消息类型：

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

Sxb (PFCP) 消息：

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

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`	按结果代码类别和对等主机分类的 Diameter 响应

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 (

# 计算成功的 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
```

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`	
`sxb_inbound_handling_duration`	Histogram	`request_message_type`	
`gx_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`
`sxb_outbound_transaction_duration`	Histogram	`request_message_type`
`gx_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（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_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 的连续丢失心跳次数

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
```

告警示例：

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

# 当多个 UPF 下线时发出警报
- 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"

# 关于丢失心跳的警告
- 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 }} 次连续丢失心跳"
```

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	-	成功通过 DNS 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 的健康 P-CSCF 服务器
`pcscf_servers_unhealthy`	Gauge	`fqdn`	每个 FQDN 的不健康 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。

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
```

告警示例：

```yaml
- 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
- 当许可证无效时（`license_status == 0`），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 和监控仍然可以访问
- 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	总打开端口

Prometheus □置

抓取□置

将 OmniPGW 添加到 Prometheus `prometheus.yml`:

```
# 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:

创建一个新仪表板或从 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 - 按结果代码类别分解：

```
# 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：Stat
# 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:
      # 会话计数告警
      - 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 会话计数临界"
          description: "{{ $value }} 活动会话接近容量"

      # IP 池告警
      - 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: "没有可分配的 IP"

      # 错误率告警
```

```
      - 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"

      - 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 响应告警
      - 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"
```

```
      # UPF 健康告警
      - alert: OmniPGW_UPF_PeerDown
        expr: upf_peer_healthy == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "UPF 对等 {{ $labels.peer_ip }} 已下线"
          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 }} 次连续丢失心跳"

      - alert: OmniPGW_UPF_AllDown
        expr: upf_peers_healthy == 0 and upf_peers_total > 0
        for: 30s
        labels:
          severity: critical
        annotations:
          summary: "所有 UPF 对等已下线"
          description: "没有健康的 UPF 可用于会话创建"

      # 延迟告警
      - alert: OmniPGW_HighLatency
        expr: |
          histogram_quantile(0.95,
            rate(s5s8_inbound_handling_duration_bucket[5m])
```

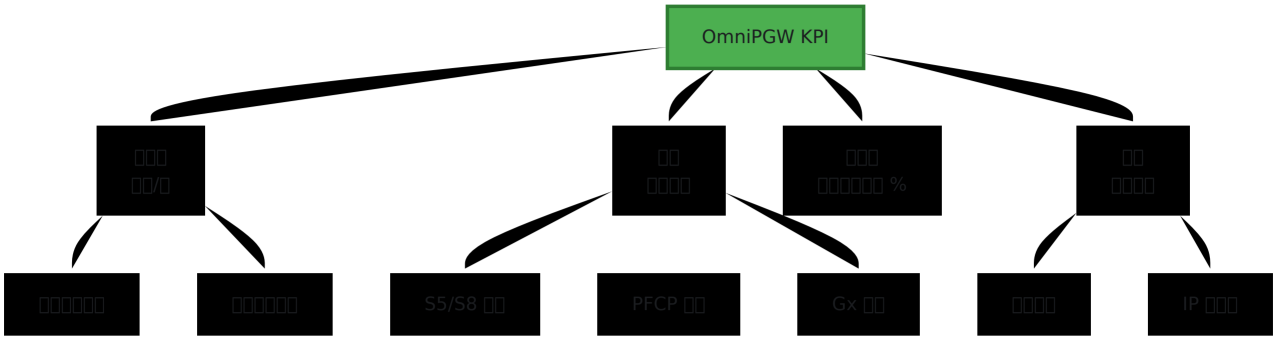
```
      ) > 100000
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "OmniPGW 高消息延迟"
      description: "p95 延迟 {{ $value }}µs (> 100ms)"

  # 系统告警
  - 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 {{ $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

关键绩效指标（KPI）



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])
)
```

按消息类型分解的延迟：

```
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
```

到达容量耗尽的时间：

```
# □于 1 小时增长速率的容量耗尽天数
(10000 - teid_registry_count) /
(rate(teid_registry_count[1h]) * 86400)
```

故障排��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（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）

- 审查系统资源使用情况

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 主机分解
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 连接和健康状况
- 审查 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），审查 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
- 审查注册计数
- 如果确认泄漏，请重启

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

告警设计

1. 避免告警疲劳 - 仅对可操作的问题发出告警
2. 升级 - 警告 → 关键，逐步升级严重性
3. 上下文 - 在告警描述中包含运行手册链接

相关文档

□置和设置

- □置指南 - Prometheus 指标□□□置，Web UI 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

专业监控

- **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□ - IP 池利用率指标

返回操作指南

OmniPGW 监控指南 - *□ Omnitouch □□□□□□*

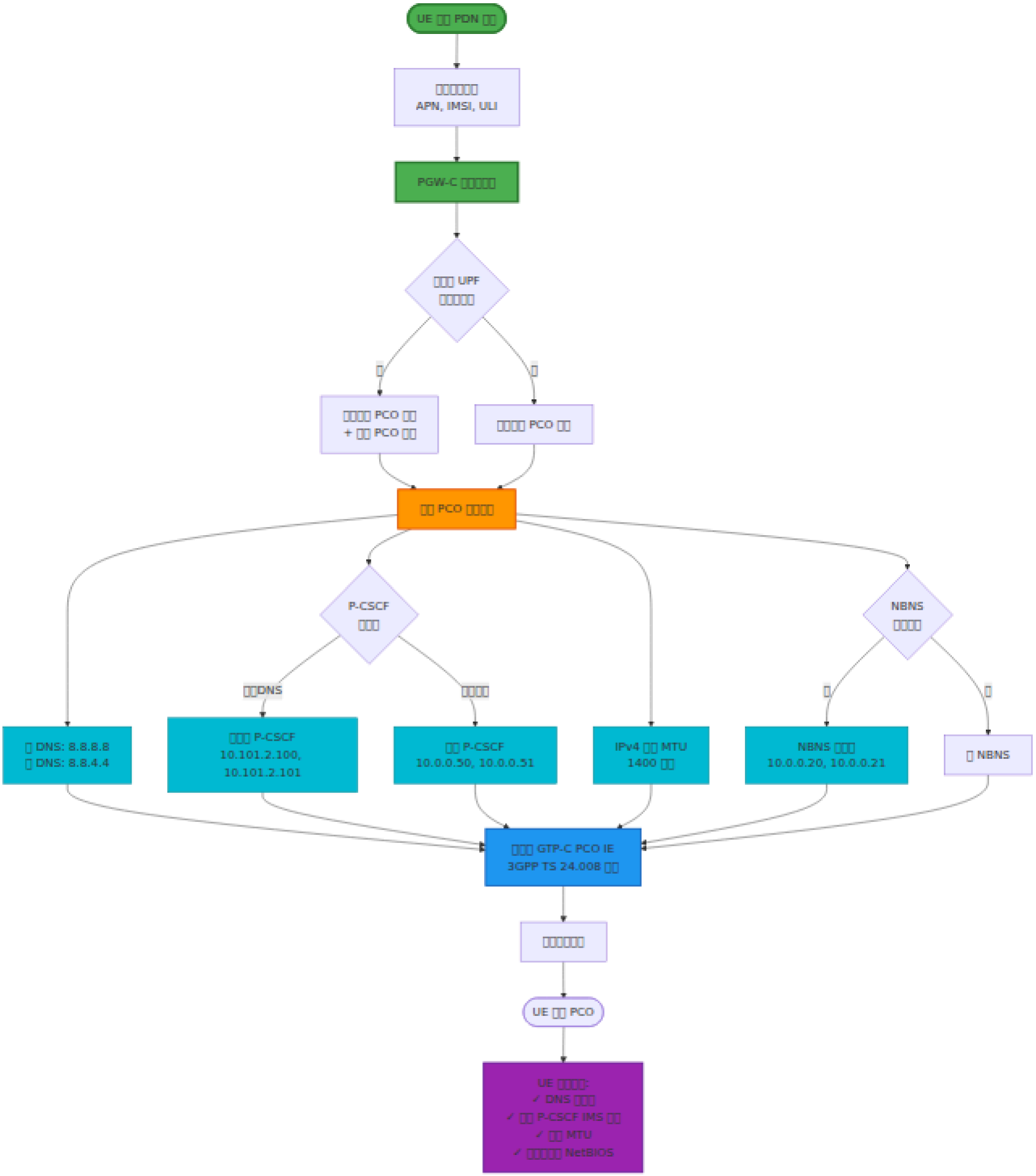
协议□置选项（PCO）

传递给 **UE** 的网络参数

OmniPGW □ Omnitouch □□□□□□

概述

PCO（协议□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。


UE □□ PDN □□
□□□□□□□
APN, IMSI, ULI
PGW-C □□□□□□
□□□ UPF
□□□□□□
□
□
□□□□ PCO □□
+ □□ PCO □□
□□□□ PCO □□
□□ PCO □□□□
P-CSCF
□□□
NBNS
□□□□
□□DNS
□□□□
□
□
□ DNS: 8.8.8.8
□ DNS: 8.8.4.4
□□□ P-CSCF
10.101.2.100,
10.101.2.101
□□ P-CSCF
10.0.0.50, 10.0.0.51
IPv4 □□ MTU
1400 □□
NBNS □□□
10.0.0.20, 10.0.0.21
□ NBNS
□□□ GTP-C PCO IE
3GPP TS 24.008 □□
□□□□□□□
UE □□ PCO
UE □□□□:
✓ DNS □□□
✓ □□ P-CSCF IMS □□
✓ □□ MTU
✓ □□□□□ NetBIOS

PCO 信息元素:

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	次 DNS	可选
P-CSCF IPv4 地址	0x000C	IMS 的 P-CSCF	可选 (IMS)
IPv4 链路 MTU	0x0010	最大传输单元	推荐
NBNS 服务器 IPv4 地址	0x0011	NetBIOS 名称服务器	可选

□置

基本□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",

    # 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: [],

    # P-CSCF 动态发现 (可选)
    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** 和次 **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 **DNS**	次 **DNS**
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
    "10.0.0.51"   # 次 P-CSCF
  ]
}
```

P-CSCF (代理呼叫会话控制功能**)**:

- IMS 信令的入口点
- VoLTE、VoWiFi、RCS 所必需
- UE 通过此服务器使用 SIP

P-CSCF 动态发现

基于 **DNS** 的 **P-CSCF** 发现：

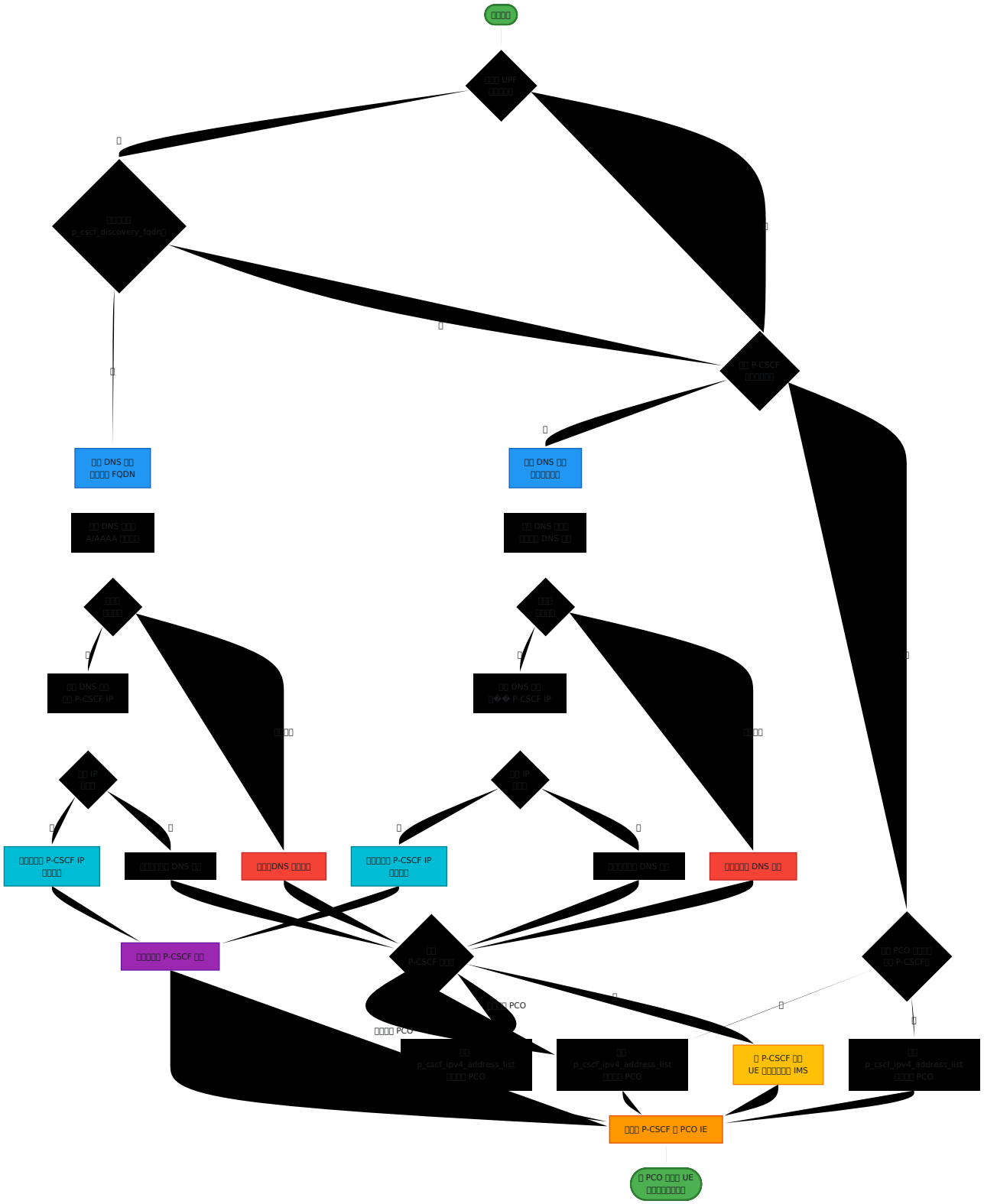
OmniPGW 支持通过 DNS 查询进行动态 P-CSCF 发现，具体定义见 3GPP TS 23.003 和 TS 24.229。当启用时，PGW-C 可以查询 DNS 获取 P-CSCF 地址，而不是使用静态配置。

```
pco: %{
  # 启用动态 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 会执行 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 地址将通过 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. 有关详细监控和指标，请参见 P-CSCF 监控

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 发现尝试都会记录并跟踪指标：

- DNS 查询成功/失败率
- 发现延迟
- 回退使用统计
- 按规则和全局发现指标

有关完整监控详细信息，请参见 P-CSCF 监控。

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 - 不同的订阅者获取不同的 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"   # 次回退
  ],

  ipv4_link_mtu_size: 1400
}
```

按规则 **P-CSCF** 发现:

P-CSCF 发现也可以按 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: [...],

    # 按规则 P-CSCF 发现
    p_cscf_discovery_fqdn: "pcscf.mnc380.mcc313.3gppnetwork.org"
  }
]
```

有关按规则 P-CSCF 发现的详细信息，请参见 UPF 选择配置。

另见: P-CSCF 监控 以监控 P-CSCF 发现和健康状态

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
- 旧版应用程序支持
- 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。

□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 查询失败

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 通话失败
- 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，其他网站不加载
- 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：GTP 隧道使用 1400
pco: %{
  ipv4_link_mtu_size: 1400
}

# 如果仍然有问题，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. 始终配置次级 **DNS**
    - 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 DNS 过滤以增强安全性

IMS 配置

1. 提供多个 **P-CSCF**
    - 至少 2 个以确保冗余
    - 如果可能，进行地理分布
2. 确保可达性
    - P-CSCF 必须可从 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

□置指南

- □置指南 - 完整的 runtime.exs 参考，UPF 选择与 PCO 覆盖
- **UE IP** 分□ - 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��生命周期，承载建立
- **S5/S8** 接口 - GTP-C 协议，PCO 编码和交付
- **PFCP** 接口 - 用户面会话建立

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

返回操作指南

P-CSCF 发现与监控

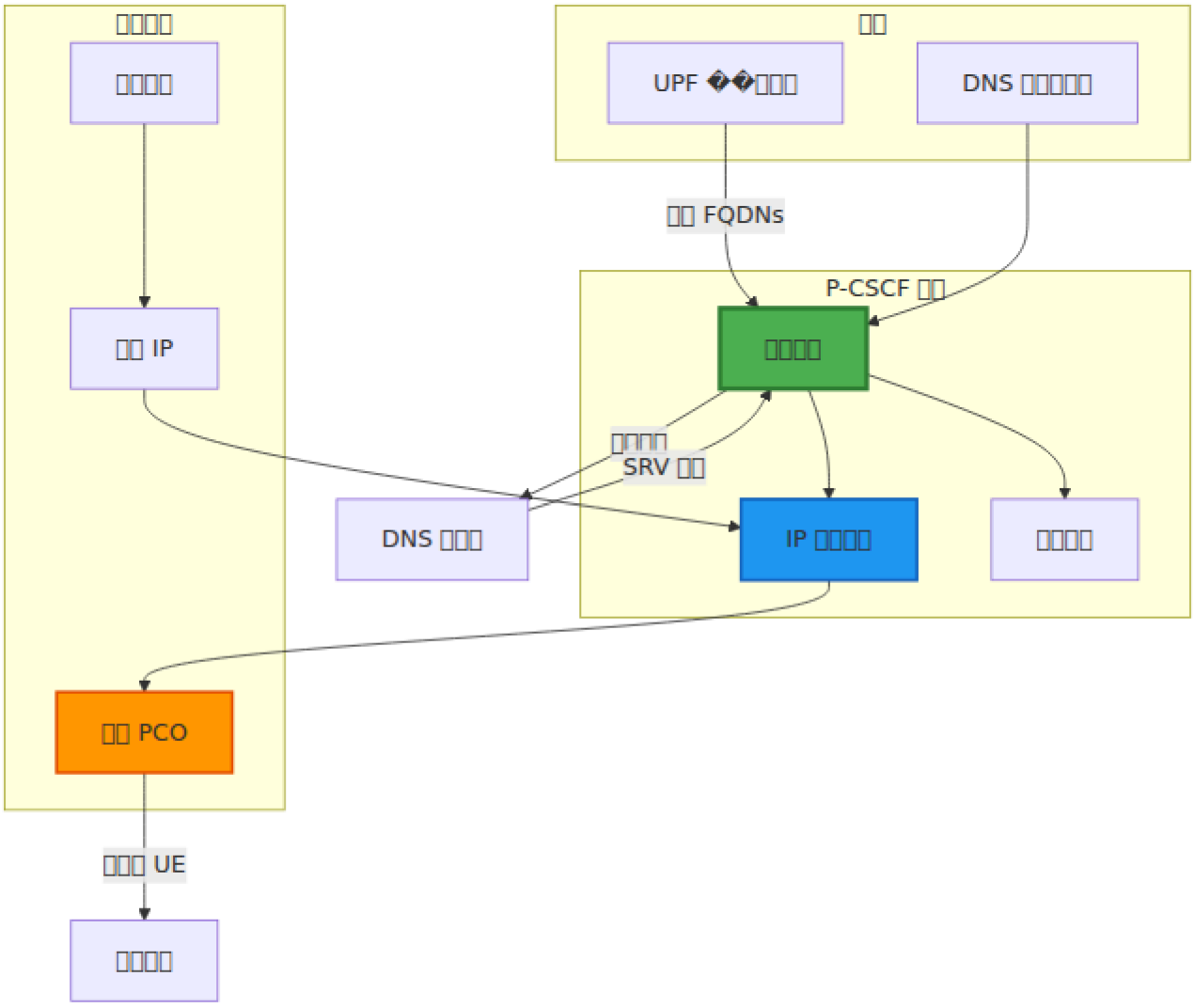
动态 **P-CSCF** 服务器发现与实时监控

OmniPGW □ Omnitouch □□□□□□

概述

P-CSCF（代理呼叫会话控制功能）发现与监控 提供使用 DNS SRV 查询的 IMS P-CSCF 服务器的动态发现，并进行实时 SIP OPTIONS 健康检查。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 **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：Web UI 显示解析状态、发现的 IP 和健康状态
- 优雅回退：三层回退策略以实现最大可靠性
- **Prometheus** 指标：通过 Prometheus 指标实现完全可观察性



目录

快速开始

基本□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: "10.179.2.177",
    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 流量",
        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.mnc380.mcc313.3gppnetwork.org",
        # 静态回退（请参见 PCO 配置指南）
        pco: %{
          p_cscf_ipv4_address_list: ["10.101.2.100",
"10.101.2.101"]
        }
      }
    ]
  }
```

请参阅 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. 导航至 **Web UI → P-CSCF** 监控 (`https://localhost:8086/pcscf_monitor`)
3. 查看实时解析状态和发现的 IP

□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: "10.179.2.177",

  # 启用 P-CSCF DNS 发现功能
  p_cscf_discovery_enabled: true,

  # DNS SRV 查询的超时(毫秒)
  p_cscf_discovery_timeout_ms: 5000,

  # 静态 P-CSCF 地址(全局回退)
  p_cscf_ipv4_address_list: ["10.101.2.146"]
}
```

按规则 **P-CSCF FQDNs**

每个 UPF 选择规则可以指定其自己的 P-CSCF 发现 FQDN:

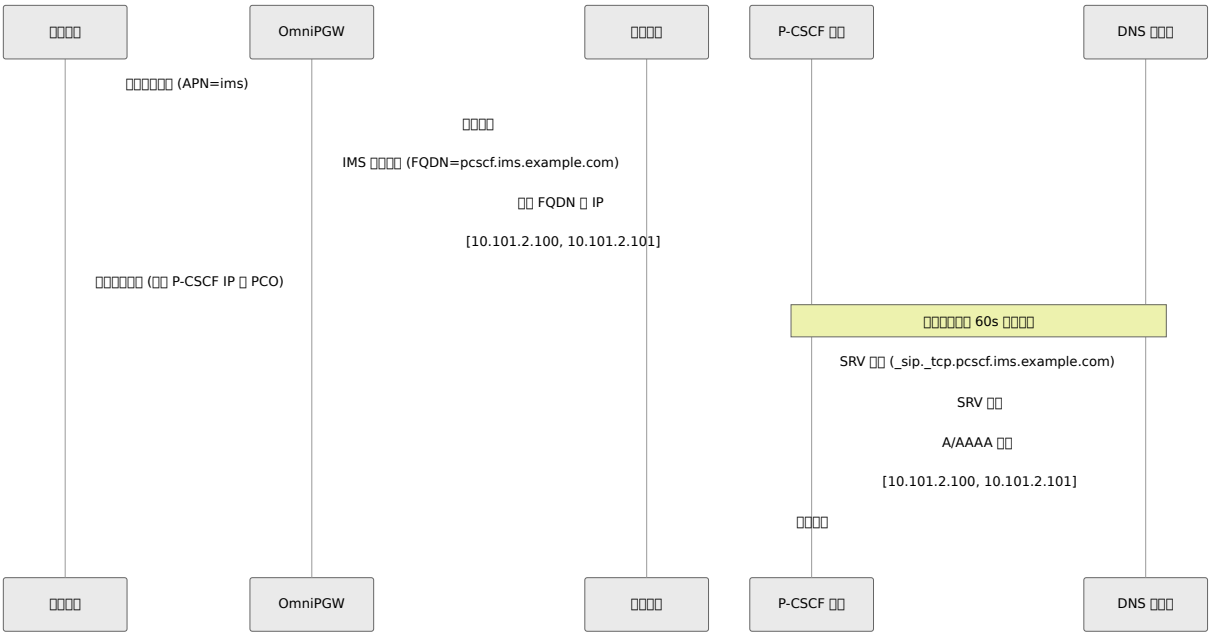
```
upf_selection: %{
  rules: [
    # IMS 流量 - IMS 特定的 P-CSCF
    %{
      name: "IMS 流量",
      match_field: :apn,
      match_regex: "^ims",
      upf_pool: [...],
      p_cscf_discovery_fqdn:
"pcscf.ims.mnc380.mcc313.3gppnetwork.org",
      pco: %{
        p_cscf_ipv4_address_list: ["10.101.2.100"]  # 回退
      }
    },

    # 企业 - 企业特定的 P-CSCF
    %{
      name: "企业流量",
      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: "互联网流量",
      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 FQDNs
2. **FQDN** 注册
    - 每个唯一的 FQDN 都会在监控器中注册
    - 监控器对每个 FQDN 执行初始 DNS SRV 查询
    - **SIP OPTIONS** 健康检查（对所有发现的服务器并行进行）：
        - 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`（有响应）或 `:down`（无响应/超时）
    - 结果（IP、健康状态或错误）将与时间戳一起缓存
3. 定期监控（每 60 秒）
    - 监控器刷新所有 FQDNs
    - 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

□□□□□□



DNS □□□□

□□□□□□ **DNS SRV** □□ □□□□□ P-CSCF □□□

1. **SRV** □□□□□□ `_sip._tcp.{fqdn}` □ SRV □□
2. □□□□□□□□□□□□□□□□□□
3. □□□□□□□ SRV □□□□□□□□□□□
4. □□□□□□□□□□□□□□□□□□□ IP □□□A/AAAA □□□
5. □□□□□□□□□ IP □□□□□□□□□

P-CSCF □□□□□□□

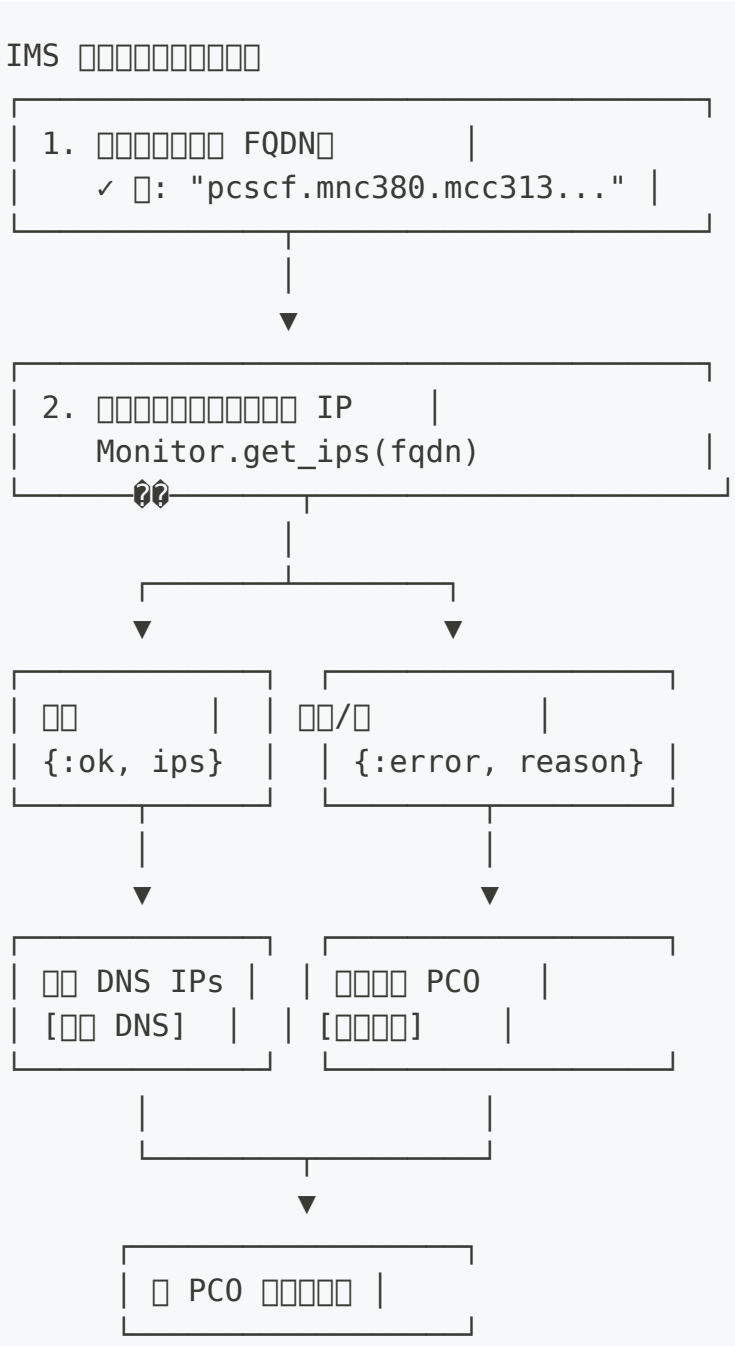
□□□□□□□□□□ **FQDN** □□□ **PCO** □□**FQDN** □□□

```
%{
  name: "IMS 流量",
  p_cscf_discovery_fqdn: "pcscf.mnc380.mcc313.3gppnetwork.org",  # ← 首先尝试
  pco: %{
    p_cscf_ipv4_address_list: ["10.101.2.100", "10.101.2.101"]   # ← 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 IPs..., 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	回退触发
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□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.mnc380.mcc313..."
2. □□□□□□□□□□□□□ IP
Monitor.get_ips(fqdn)
□□
{:ok, ips}
□□/□
{:error, reason}
□□ DNS IPs
[□□ DNS]
□□□□ PCO
[□□□□]
□ PCO □□□□□

□□□□□

□□ 1□DNS □□□□□□ □

```
配置：
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  pco.p_cscf_ipv4_address_list: ["10.101.2.100"]

DNS 结果：[10.101.2.150, 10.101.2.151]
UE 接收：[10.101.2.150, 10.101.2.151] ← 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: ["10.101.2.100"]

DNS 结果：ERROR :no_naptr_records
UE 接收：[10.101.2.100] ← 来自静态 PCO
注意：尽管 DNS 失败，会话仍然成功
```

场景 **3**：未□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，无需操作员干预
4. 生产安全：兼具敏捷性与稳定性

建议：在生产部署中始终配置 FQDN 和静态 PCO：

```
# ✓ 推荐：动态与回退
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com",  # 优先
  pco: %{
    p_cscf_ipv4_address_list: ["10.101.2.100"]     # 安全网
  }
}

# ⚠ 风险：仅动态（回退到全局 PCO）
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com"
  # 无规则特定的回退！
}

# ✓ 有效：仅静态（无 DNS 开销）
%{
  pco: %{
    p_cscf_ipv4_address_list: ["192.168.1.50"]
  }
}
```

Web UI 监控

P-CSCF 监控页面

访问监控界面: `https://localhost:8086/pcscf_monitor`

功能：

- 概述统计
  - 监控的总 FQDNs
  - 成功解析的 FQDNs
  - 解析失败的数量
  - 发现的 P-CSCF IP 总数
- **FQDN** 表
  - 正在监控的 FQDN
  - 解析状态（✓ 已解析 / ✗ 失败 / □ □□□□
  - 发现的 IP 数量
  - 解析的 IP 地址列表（带可扩展的服务器详细信息）
  - 最后更新时间戳
  - 每个 FQDN 的手动刷新按钮
  - 健康状态：每个发现的服务器显示：
    - IP 地址和端口
    - 主机名（来自 DNS SRV 目标）
    - 实时健康指示器（✓ 正常 / ✗ 不正常）

- 刷新控制
  - 刷新全部 按钮：立即重新查询所有 FQDNs
  - 每 **FQDN** 刷新：按需刷新单个 FQDN
  - 自动刷新：页面每 5 秒更新一次
- 监控指标仪表板
  - 总 **FQDNs**：注册用于监控的唯一 FQDN 数量
  - 成功解析：通过 DNS 成功解析的 FQDNs
  - **DNS** 解析失败：未能解析的 FQDNs
  - **P-CSCF** 服务器：所有 FQDNs 中发现的服务器总数
  - ✓ 健康（**SIP OPTIONS** 正常）：响应 SIP OPTIONS 健康检查的服务器
  - ✗ 不健康（**SIP OPTIONS** 不正常）：未响应 SIP OPTIONS 的服务器
  - **DNS** 成功率：成功 DNS 解析的百
  - 健康检查间隔：SIP OPTIONS 健康检查的频率（60s，5s 超时）

指标仪表板提供对 DNS 解析健康和 P-CSCF 服务器可用性的实时可见性，使用 SIP OPTIONS。

UPF 选择页面集成

UPF 选择页面（`/upf_selection`）显示每个规则的 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.mnc380.mcc313.3gppnetwork.org
       状态: ✓ 已解析 (2 个 IP)
       解析的 IPs: 10.101.2.100, 10.101.2.101

    ⚙ PCO 覆盖
       主 DNS: 10.103.2.195
       P-CSCF (静态回退): 10.101.2.100, 10.101.2.101
```

指标与可观察性

Prometheus 指标

P-CSCF 监控系统通过 Prometheus 暴露指标（默认端口 42069）：

Gauge 指标

```
# FQDN 级别指标
pcscf_fqdns_total                          # 监控的 FQDN 总数
pcscf_fqdns_resolved                       # 成功解析的 FQDNs（DNS 成功）
pcscf_fqdns_failed                         # 解析失败的 FQDNs（DNS 失败）

# 服务器级别指标（聚合）
pcscf_servers_total                        # 通过 DNS SRV 发现的 P-CSCF 服务器
总数
pcscf_servers_healthy                      # 响应 SIP OPTIONS 的服务器（聚合）
pcscf_servers_unhealthy                    # 未响应 SIP OPTIONS 的服务器（聚合）

# 服务器级别指标（每 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
```

健康检查详情：

- `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 5s 超时）

指标示例

DNS 解析指标:

```
# 查询成功解析的 FQDNs
pcscf_fqdns_resolved

# 计算 DNS 成功率
(pcscf_fqdns_resolved / pcscf_fqdns_total) * 100

# 总发现的服务器
pcscf_servers_total
```

SIP OPTIONS 健康指标:

```
# 所有 FQDNs 中健康服务器的总数
pcscf_servers_healthy

# 不健康服务器的总数
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.mnc380.mcc313.3gppnetwork.org"}

# 当所有服务器都不健康时发出警报
pcscf_servers_healthy == 0 AND pcscf_servers_total > 0
```

Prometheus 警报示例:

```yaml
# 当所有 P-CSCF 服务器都不健康时发出警报
- 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% 的服务器不健康时发出警报
- 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"

# 对 DNS 解析失败发出警报
- 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 监控启动
[info] 注册 2 个唯一的 P-CSCF FQDNs 进行监控: ["pcscf.ims.example.com",
"pcscf.enterprise.example.com"]
[info] P-CSCF 监控: 注册 FQDN pcscf.ims.example.com
[debug] P-CSCF 监控: 成功将 pcscf.ims.example.com 解析为 2 个 IP
[warning] P-CSCF 监控: 未能解析 pcscf.enterprise.example.com: :nxdomain
[debug] 使用来自 FQDN pcscf.ims.example.com 的 P-CSCF 地址: [{10, 101,
2, 100}, {10, 101, 2, 101}]
```

回退策略

该系统使用 三层回退策略 以实现最大可靠性：

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
- 优势：动态、负载均衡、位置感知

第 2 层：规则特定的静态 PCO（回退）

```
pco: %{
  p_cscf_ipv4_address_list: ["10.101.2.100", "10.101.2.101"]
}
```

- 如果 DNS 发现失败或返回无 IP，则使用
- 规则特定的静态配置
- 优势：规则特定的回退，预测性强

第 3 层：全局 PCO 配置（最后手段）

```
# 全局 PCO 配置
pco: %{
  p_cscf_ipv4_address_list: ["10.101.2.146"]
}
```

- 如果没有规则特定的配置且 DNS 失败，则使用
- 全局默认的 P-CSCF 地址
- 优势：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 [10.101.2.100, 10.101.2.101] ✓
   └─ 失败 → 尝试下一层

2. 尝试规则的 PCO 覆盖
   ├─ 配置 → 使用 [10.101.2.100, 10.101.2.101] ✓
   └─ 未配置 → 尝试下一层

3. 使用全局 PCO 配置
   └─ 使用 [10.101.2.146] ✓ (始终成功)
```

DNS 配置

DNS 服务器设置

配置 DNS 服务器，使用 SRV 和 A/AAAA 记录进行 P-CSCF 发现：

```
; P-CSCF 的 SRV 记录（_sip._tcp 前缀会自动查询）
_sip._tcp.pcscf.mnc380.mcc313.3gppnetwork.org. IN SRV 10 50 5060
pcscf1.example.com.
_sip._tcp.pcscf.mnc380.mcc313.3gppnetwork.org. IN SRV 20 50 5060
pcscf2.example.com.

; A 记录
pcscf1.example.com. IN A 10.101.2.100
pcscf2.example.com. IN A 10.101.2.101
```

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.mnc380.mcc313.3gppnetwork.org"`，系统将查询 `_sip._tcp.pcscf.mnc380.mcc313.3gppnetwork.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.mnc380.mcc313.3gppnetwork.org
@10.179.2.177

# 预期输出：
# _sip._tcp.pcscf.mnc380.mcc313.3gppnetwork.org. 300 IN SRV 10 50
5060 pcscf1.example.com.

# 解析 P-CSCF 主机名到 IP
dig A pcscf1.example.com @10.179.2.177

# 预期输出：
# pcscf1.example.com. 300 IN A 10.101.2.100
```

故障排除

问题：FQDN 显示“失败”状态

症状：

- Web UI 显示 ✗ 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 服务器连接性
ping 10.179.2.177

# 2. 手动测试 NAPTR 查询
dig NAPTR pcscf.mnc380.mcc313.3gppnetwork.org @10.179.2.177

# 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. 在 Web UI 中手动刷新
# 点击失败的 FQDN 旁边的“刷新”按钮
```

问题：未返回 IP

症状：

- Web UI 显示“0 个 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. 服务字段不匹配 IMS/SIP 模式
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 @10.179.2.177

# 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 @10.179.2.177
```

问题：会话使用错误的 P-CSCF

症状：

- UE 接收到意外的 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 "使用来自 FQDN 的 P-CSCF 地址" /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 和状态

# 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:

```
# 减少查询超时
pco: %{
  p_cscf_discovery_timeout_ms: 2000  # 从 5000ms 减少
}

# 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  器

```
pco: %{
  # 专用于 P-CSCF 发现的 DNS（与 UE DNS 不同）
  p_cscf_discovery_dns_server: "10.179.2.177",

  # 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
- 独立扩展和可靠性

2. 始终□置静态回退

```
%{
  p_cscf_discovery_fqdn: "pcscf.ims.example.com",  # 优先
  pco: %{
    p_cscf_ipv4_address_list: ["10.101.2.100"]  # 必需的回退
  }
}
```

为什么?

- 确保会话即使在 DNS 失败时也能成功
- 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.mnc380.mcc313.3gppnetwork.org"
  },

  # 企业
  %{
    name: "企业",
    match_regex: "^enterprise",
    p_cscf_discovery_fqdn: "pcscf.enterprise.example.com"
  }
]
```

为什么?

- 每项服务使用不同的 P-CSCF 池
- 更好的负载分配

- 服务特定路由

4. 监控 DNS 查询性能

```
# 对高 P-CSCF 查询延迟发出警报
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

- **Web UI**: 每天检查 P-CSCF 监控页面
- 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. 置适当的超时

```
# 生产环境: 平衡可靠性与延迟
pco: %{
  p_cscf_discovery_timeout_ms: 5000  # 5 秒
}

# 高性能: 优先速度，依赖回退
pco: %{
  p_cscf_discovery_timeout_ms: 2000  # 2 秒
}
```

7. 使用 DNS 冗余

配置主 DNS 和备用 DNS:

```
# 主 P-CSCF DNS
pcscf.mnc380.mcc313.3gppnetwork.org. IN NAPTR 10 50 "s" "SIP+D2U"
"" _sip._udp.pcscf1.example.com.

# 备用 P-CSCF DNS
pcscf.mnc380.mcc313.3gppnetwork.org. IN NAPTR 20 50 "s" "SIP+D2U"
"" _sip._udp.pcscf2.example.com.
```

相关文档

- **PCO □置** - 协议配置选项、DNS 和 P-CSCF 设置
- □置指南 - 完整的 OmniPGW 配置参考
- 监控 - 指标、日志和可观察性
- 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 通信

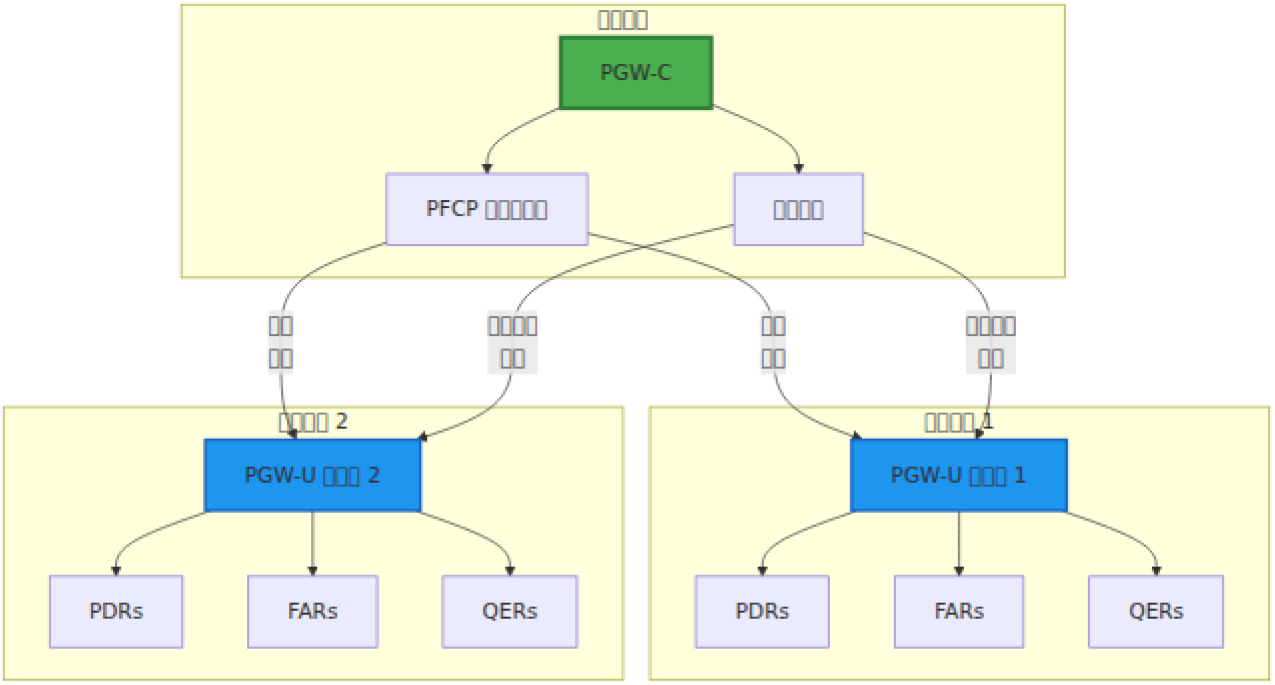
目录

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 进行标识，可以是：

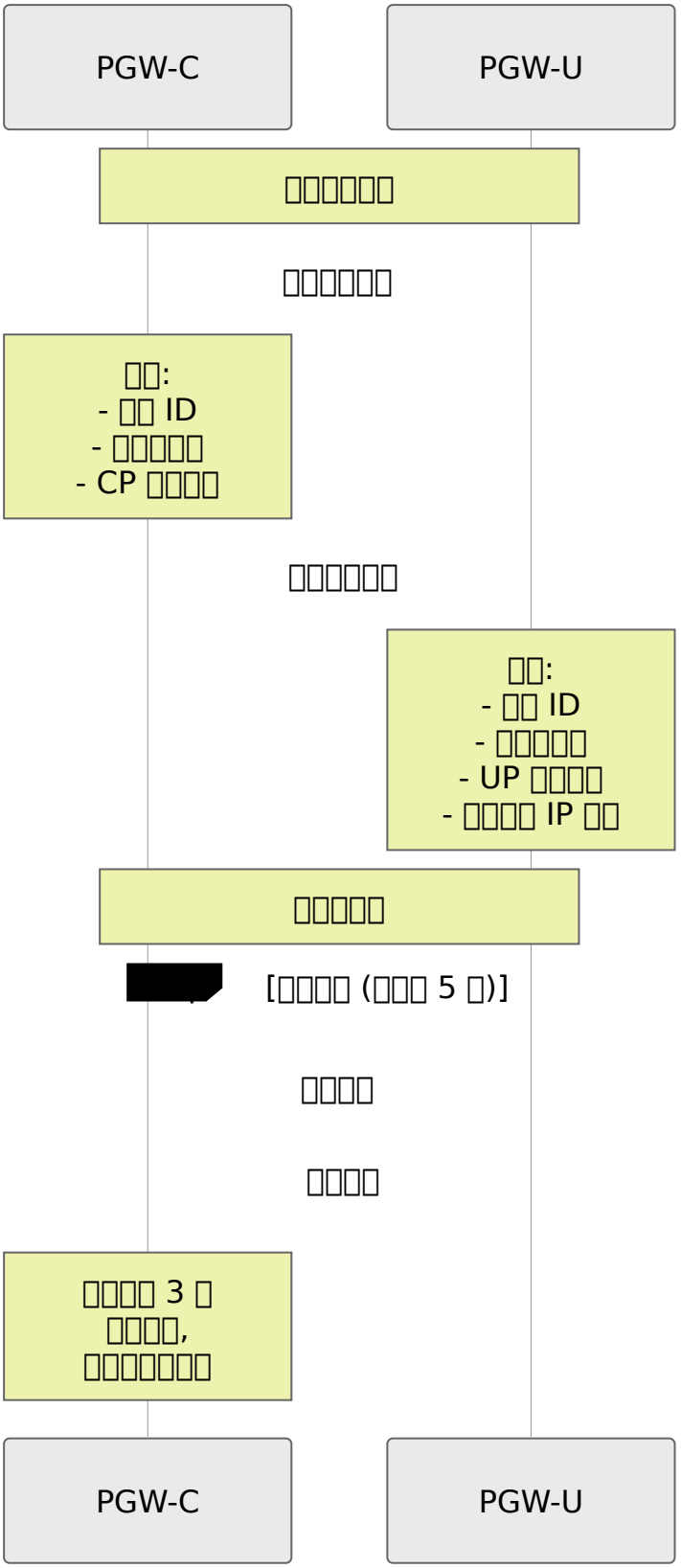
- **IPv4** 地址 - 最常见
- **IPv6** 地址

- **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] ([redacted] 5 [redacted])]
[redacted]
[redacted]
[redacted] 3 [redacted]
[redacted],
[redacted]
PGW-C
PGW-U

对等体状态管理

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`	丢失心跳的计数
`up_function_features`	支持的用户平面特性
`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

□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
```

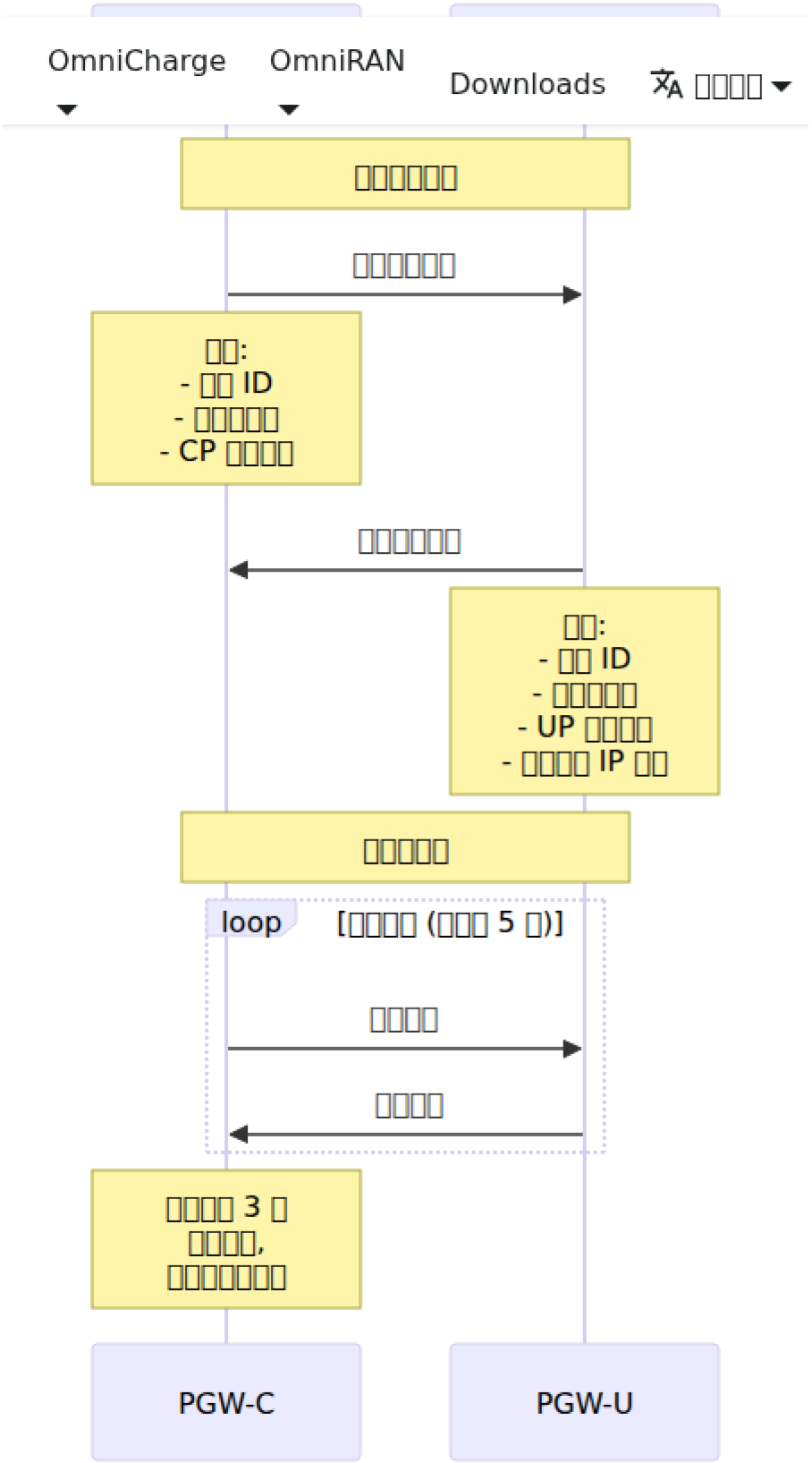
故障检测：

- 每个丢失的心跳增加 `missed_heartbeats_consecutive`
- 通常配置为连续丢失 3 次后失败
- 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，以编程用户平面的转发规则。

会话生命周期


OmniCharge
OmniRAN
Downloads
□□□□□
□□□□□□□
□□□□□□□
□□:
- □□ 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 发送到 **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 执行规则 (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** - 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 更改、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□

F-TEID（完全合格隧道端点标识符**）**用于标识用户平面流量的 GTP-U 隧道端点。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的位置。可以采用两种方法：

理解 **F-TEID** 分□

分□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 数据包的位置（UPF 的 IP 地址）

问题：谁分配 TEID 值?

选项 **1**: **UPF** 分□ **(**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□ **(**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，而不进行分配

UPF 分□ **(**默认 **-** 推荐**)**

□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 实现支持此功能
- 遵循“让所有者分配”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 TEID

何时使用：

- □ □□ UPF 的生产部署（默认）
- □ □□ UPF 池的多 PGW-C 部署
- □ □□□□□□□□□□□□□□
- □ □□□□□ 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□ (遗留兼容性)

□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，而不进行分配
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 PFCP 会话建立请求中有显式 TEID
- 兼容性的唯一解决方法

□ □□□ TEID 管理

- 如果您需要 PGW-C 完全了解所有分配的 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（���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：

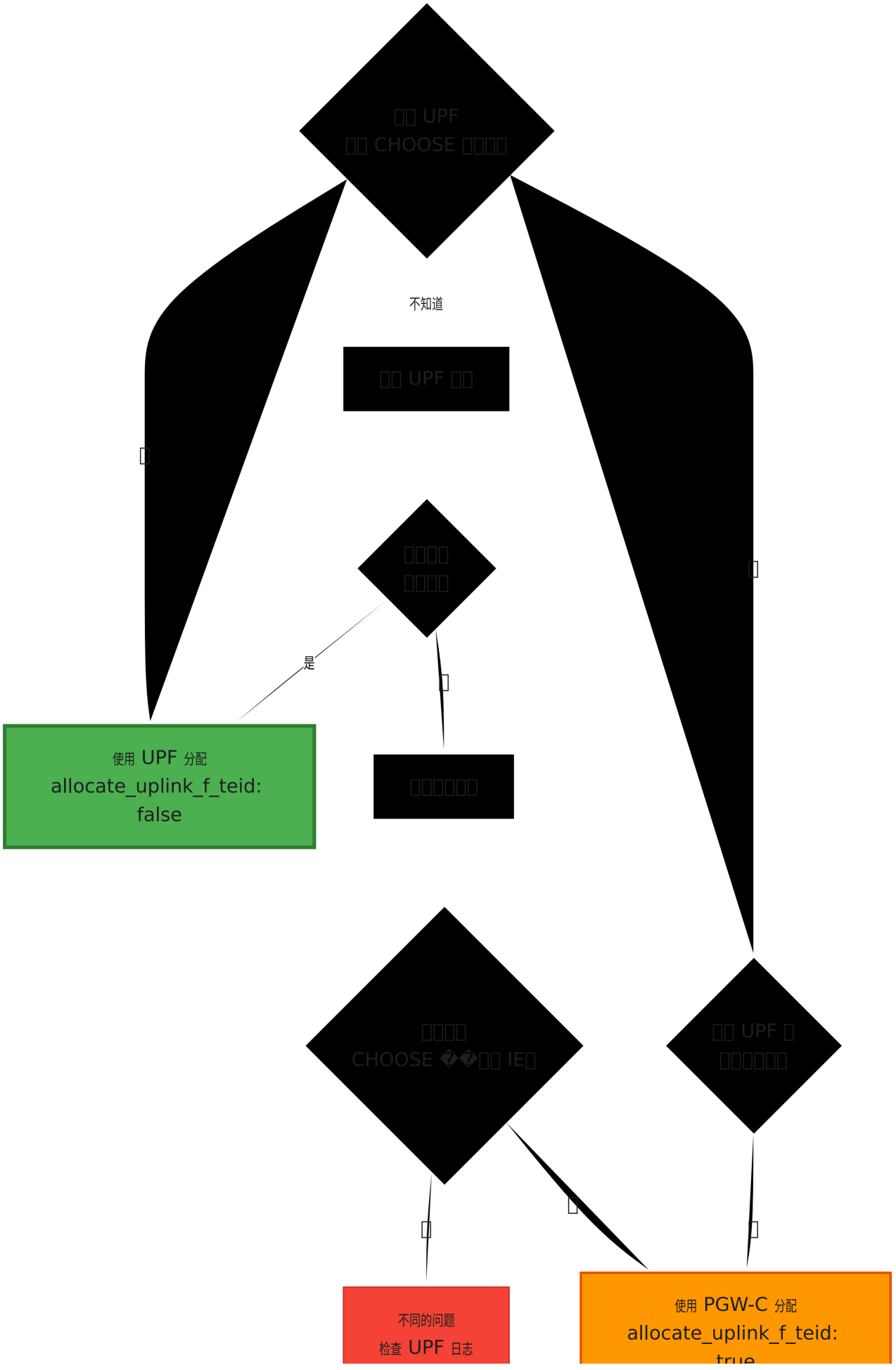
- ⚠ 仅当 **UPF** 不支持 **CHOOSE** 标志时
- ⚠ 遗留 UPF 实现（例如，一些专有硬件）
- ⚠ 特定的兼容性要求
- ⚠ 需要 PGW-C TEID 可见性的调试场景

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 □
□□□□□□□
不同的问题
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，但收到了 CHOOSE
- 解决方案：设置 `allocate_uplink_f_teid: true`

如果 **PGW-C** 分□导致问题：

- 非常罕见 - TEID 空间巨大（40 亿个值）
- 检查 TEID 耗尽（在数百万个会话以下不太可能）：

```
# 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
```

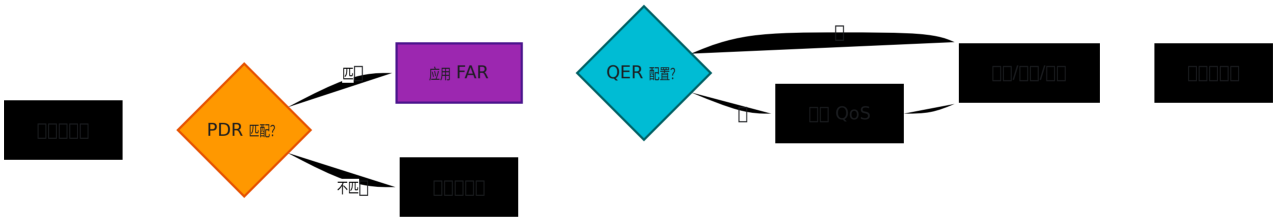
验证哪个模式处于活动状态：检查 PFCP 数据包捕获：

```
# 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 (数据包检测信息):
  - 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 (关联转发规则)
QER ID: 1 (QoS 执行)
```

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）
- 外部头部移除 - 在入口处剥离 GTP-U 头部
- **FAR ID** - 关联的转发动作
- **QER ID** - 关联的 QoS 执行（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）
   - 外部头部创建： 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）
   - （无外部头部 - 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
- 转发参数:
  - 目标接口 (ACCESS/CORE)
  - 外部头部创建 (添加 GTP-U 隧道)
  - 网络实例 (VRF/路由表)

QER (QoS 执行规则)

目的：执行比特率限制和 QoS ��数。QER 还可以跟踪在线计费配额管理（有关信用控制，请参见 Diameter Gy 接口）。

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（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
建议缓冲数据包计数: 10
```

用于：空闲模式 DRX (不连续接收) 优化

□置

基本 **Sxb** □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,

    # 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 标志并需要显式分配
    allocate_uplink_f_teid: false
  },

  # UPF 选择 - 此处定义的所有 UPF 会自动注册
  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
      }
    ]
  }
```

多个 **PGW-U** 对等体

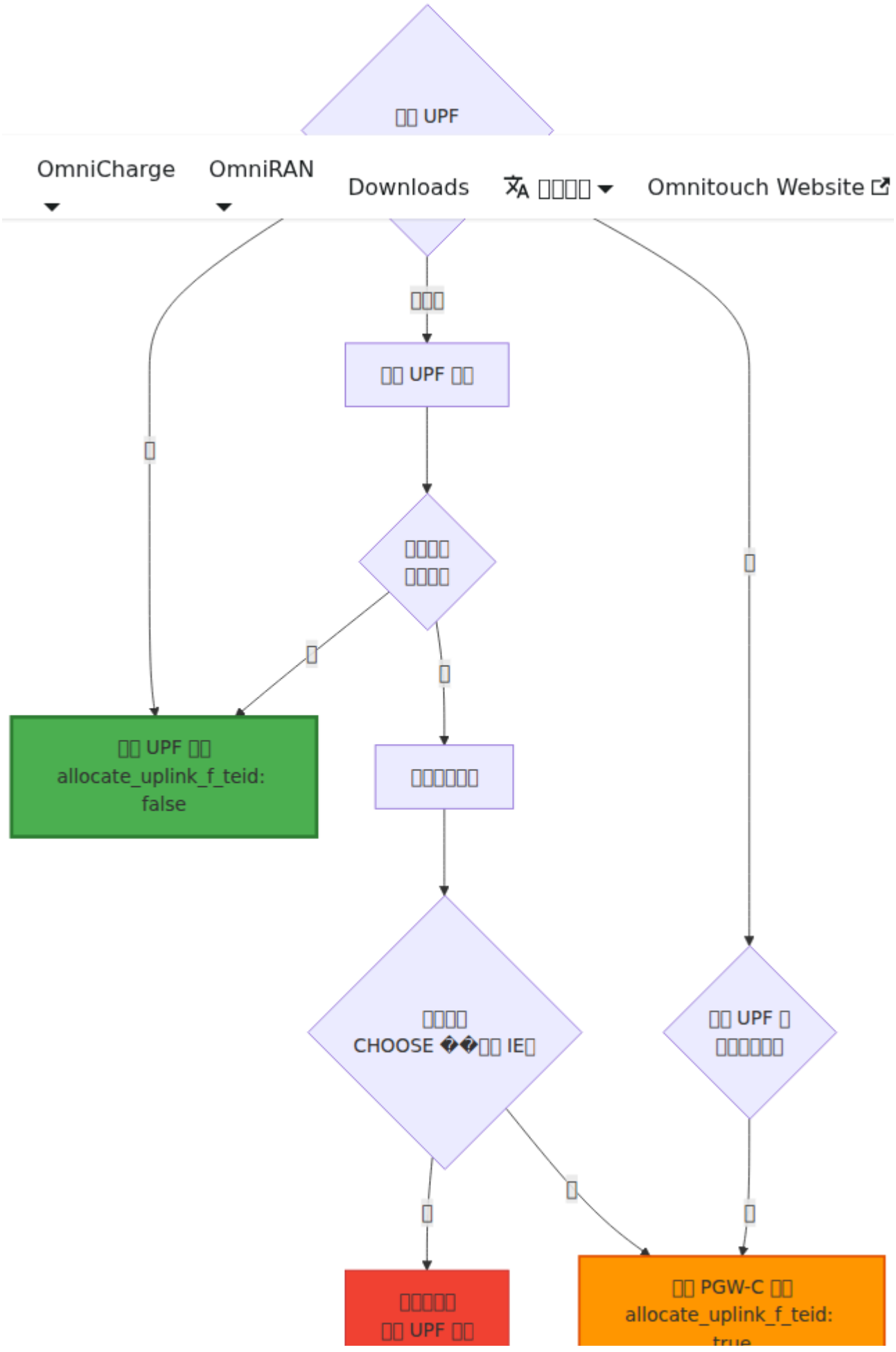
用于负载均衡或冗余:

```
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□置

PGW-C 使用 三级 **UPF** 选择系统，基于优先级的规则：

1. 静态规则（最高优先级）- 基于会话属性匹配
2. 基于 **DNS** 的选择（中等优先级）- 通过 DNS NAPTR 查询实现位置感知路由
3. 后备池（最低优先级）- 当没有规则匹配时的默认 UPF 池


□□ UPF
OmniCharge
OmniRAN
Downloads
□□□□
Omnitouch Website
□□□
□□ UPF □□
□□□□
□□□□
□
□
□
□□ UPF □□
allocate_uplink_f_teid:
false
□□□□□□□
□□□□
CHOOSE ××□□ IE□
□□ UPF □
□□□□□□□
□
□
□
□
□□□□□
□□ UPF □□
□□ PGW-C □□
allocate_uplink_f_teid:
true

□□□ 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 会自动注册
  upf_selection: %{
    #
==========================================================================
    # 于 DNS 的选择 (位置感知路由)
    #
==========================================================================
    # 使用用户位置信息 (ULI) 查询 DNS
    # 根据小区位置提供动态 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
    # 第一个匹配规则确定 UPF 池
    rules: [
      # 规则 1: IMS 流量 - 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: ["10.101.2.100", "10.101.2.101"]
```

```
        }
      },

      # 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}
        ]
      }
    ],

    #
==========================================================================
    # 后备池 （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"` (美国运营商)
`: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% □率)
   - 50-80: UPF-2 (30% □率)
   - 80-100: UPF-3 (20% □率)

用例:

- 均匀分布：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�� 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: ["10.101.2.100", "10.101.2.101"],
    # 其他字段使用主 PCO 配置的默认值
  }
}
```

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. 按优先级顺序尝试查询 (ECGI 首先，然后是 TAI 等)
2. 如果 DNS 返回候选项，使用第一个结果 (如有必要，动态注册)
3. 选择返回的 UPF
4. 如果没有 DNS 匹配或 DNS 被禁用，则回退到后备池

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

□置

```
config :pgw_c,
  upf_selection: %{
    # 启用□于 DNS 的选择
    dns_enabled: true,

    # 查询优先级：首先尝试 ECGI，然后是 TAI，然后是 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，则后备
    fallback_pool: [...]
  }
```

DNS 查询格式

DNS 查询使用来自 GTP-C 消息的用户位置信息 (ULI) 构建：

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
```

使用时机: 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
```

使用时机: 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
```

使用时机： 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
```

使用时机： 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
```

使用时机：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
```

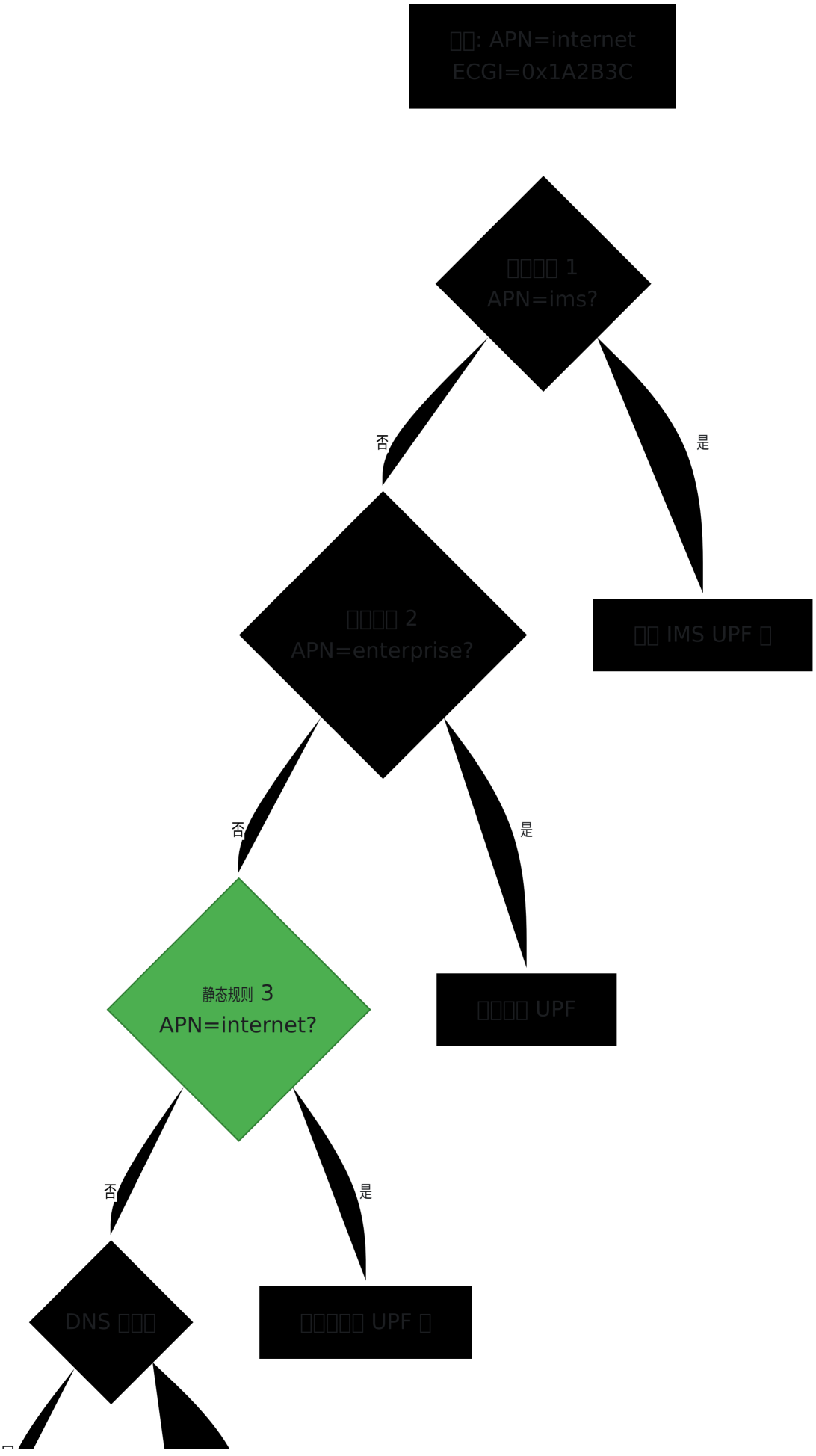
PGW-C 处理：

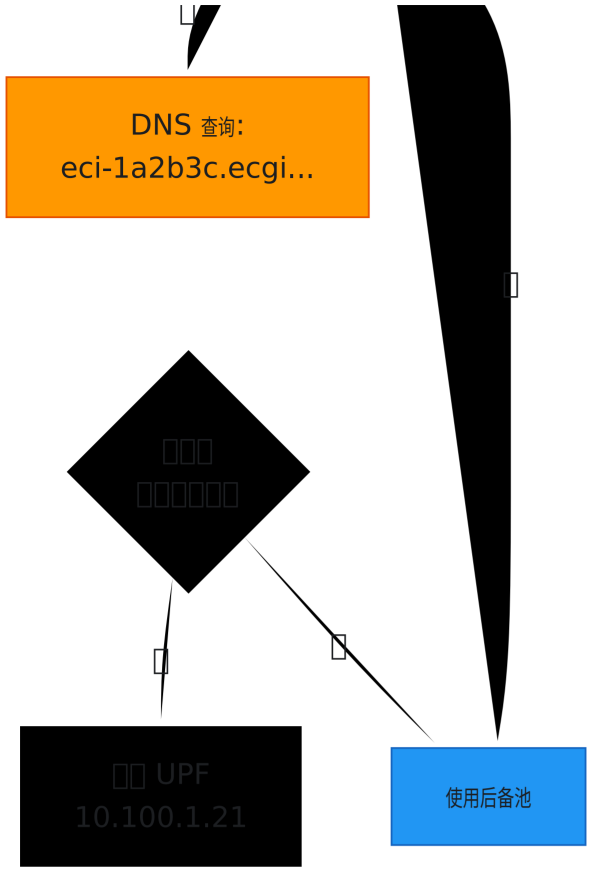
1. 解析 NAPTR 记录以提取 UPF IP 地址
2. 从 DNS 响应中选择第一个候选项
3. 如果尚未配置，则动态注册（或实现基于负载的选择）

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:      如未在 upf_selection 中注册，则动态注册
```

□□□□□□□


□□: APN=internet
ECGI=0x1A2B3C
□□□□ 1
APN=ims?
否
是
□□ IMS UPF □
□□□□ 2
APN=enterprise?
否
是
□□□□ UPF
静态规则 3
APN=internet?
否
是
□□□□□ UPF □
DNS □□□



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 □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：边缘应用程序的超低延迟

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 记录而无需更改 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”
- ��话回退到后备池

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 返回一个不在 `upf_selection` 中的候选 UPF
- 系统自动尝试动态注册
- 如果 PFCP 关联成功，则将其用于会话
- 如果 PFCP 关联失败，则回退到后备池

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: 回退到后备池
```

解决方案选项：

1. 在 `upf_selection` 中预先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��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：

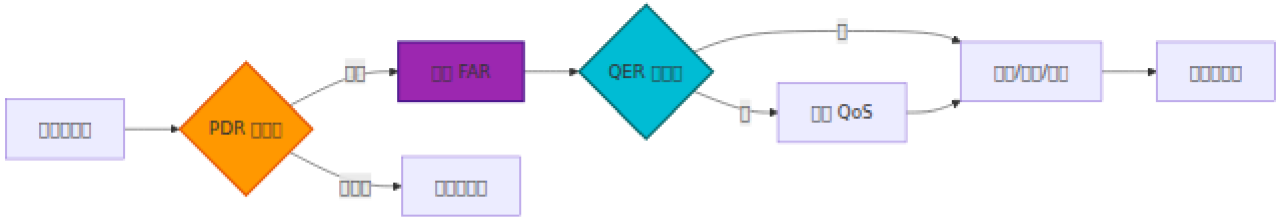
```
# DNS 查询成功率
rate(upf_selection_dns_success_total[5m]) /
rate(upf_selection_dns_attempts_total[5m])

# DNS 查询延迟
histogram_quantile(0.95,
rate(upf_selection_dns_duration_seconds_bucket[5m]))

# 后备使用情况（指示 DNS 问题）
rate(upf_selection_fallback_used_total[5m])
```

日志：

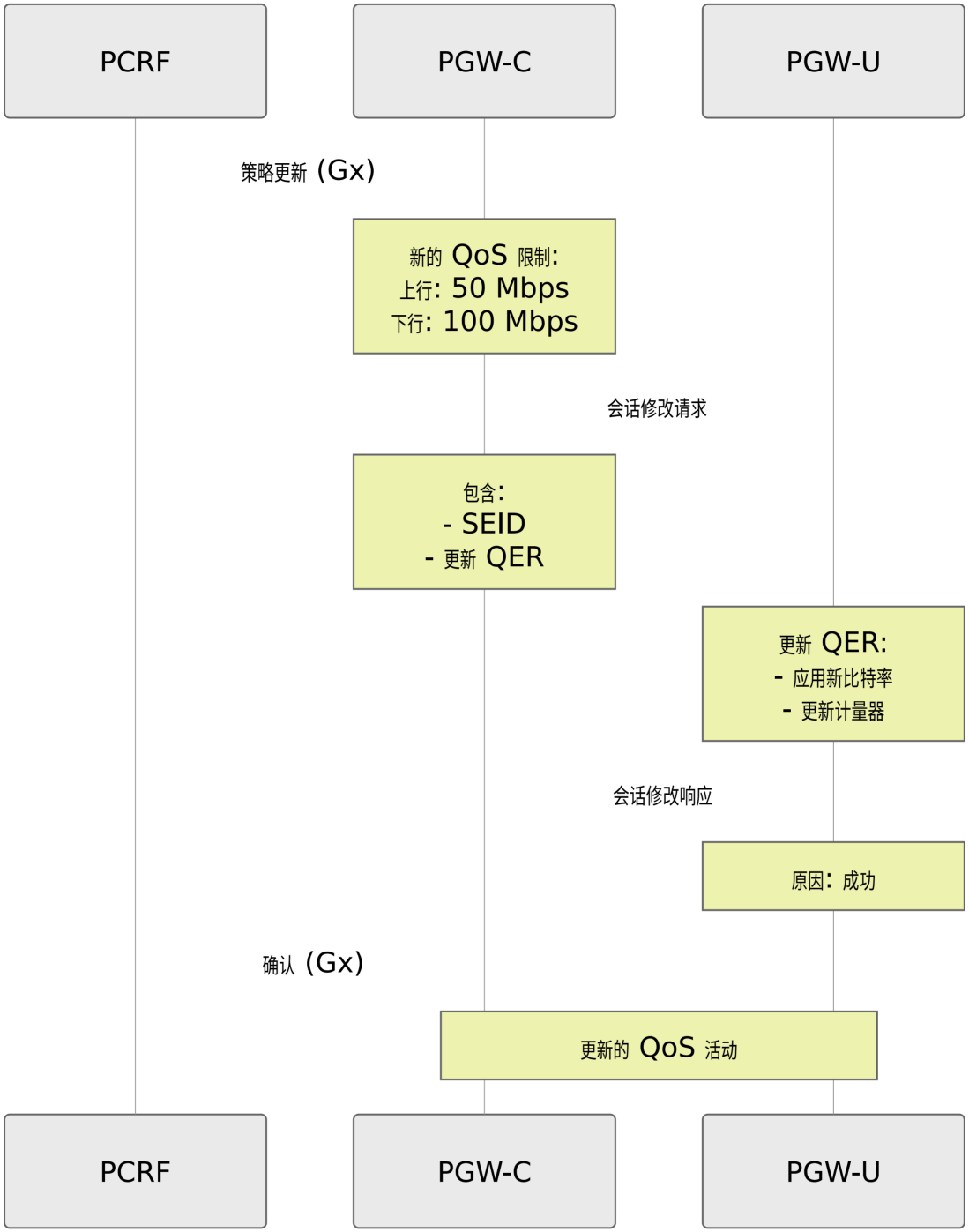
```
[debug] DNS UPF selection: querying eci-
1a2b3c.ecgi.epc.mnc999.mcc999.epc.3gppnetwork.org
[debug] DNS UPF selection: got 2 candidates from DNS
[info]  DNS UPF selection: selected 10.100.1.21
```

消息流

完整的会话建立流



会话修改流


PCRF
PGW-C
PGW-U
策略更新 (Gx)
新的 QoS 限制:
上行: 50 Mbps
下行: 100 Mbps
会话修改请求
包含:
- SEID
- 更新 QER
更新 QER:
- 应用新比特率
- 更新计量器
会话修改响应
原因: 成功
确认 (Gx)
更新的 QoS 活动
PCRF
PGW-C
PGW-U

□□□□□□


PGW-C
PGW-U (失败)
[心跳尝试]
心跳请求
超时（无响应）
missed_heartbeats = 1
心跳请求
超时（无响应）
missed_heartbeats = 2
心跳请求
超时（无响应）
missed_heartbeats = 3
将对等体标记为下线
is_associated = false
新会话将:
- 选择不同的对等体
- 或在没有对等体可用时失败
PGW-U 恢复
关联设置请求
关联设置响应
恢复关联
is_associated = true
missed_heartbeats = 0

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 无法访问（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：

```
# 测试连接
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”
- 关联标记为下线

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

���决方案:

心跳周期固定为 5 秒，失败阈值为连续丢失 3 次心跳。

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

要监控的指标：

```
# 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：

```
# 关联下线时警报
- alert: PFCPAssociationDown
  expr: pfcp_peer_associated == 0
  for: 1m
  annotations:
    summary: "PFCP 对□□□体 {{ $labels.peer }} 已下线"

# 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"
```

Web UI - PFCP 监控

OmniPGW 提供两个 Web UI 页面，用于实时监控 PFCP/Sxb 操作。

UPF/PFCP 对等体状态页面

访问: `http://<omnipgw-ip>:<web-port>/upf_status`

目的：监控与所有配置的 PGW-U 对等体的 PFCP 关联状态

功能：

1. 对等体状态概述

- 关联计数 - 活动 PFCP 关联的对等体数量
- 未关联计数 - 下线或未连接的对等体数量
- 每 2 秒自动刷新

2. 每个对等体的信息 对于每个配置的 PGW-U 对等体：

- 对等体名称 - 来自配置的友好名称
- **IP** 地址 - 远程 PGW-U IP
- 关联状态 - 关联（绿色）或未关联（红色）
- 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

3. 可展开的详细信息 单击任何对等体以查看：

- ���整的对等体配置
- UP 功能特性位图
- 关联时间戳
- 完整的对等体状态

PFCP 会话页面

访问：`http://<omnipgw-ip>:<web-port>/pfcp_sessions`

目的：查看 OmniPGW 与 PGW-U 之间的活动 PFCP 会话

功能：

1. 活动会话计数

- 活动 PFCP 会话的总数
- 实时更新

2. 会话信息 对于每个 PFCP 会话:

- 会话密钥 - 内部注册密钥
- 进程 **ID** - 会话进程标识符
- **IMSI** - 关联的用户（如果可用）
- 状态 - 会话状态

3. 完整的会话状态 可展开视图显示:

- 完整的 PFCP 会话上下文
- PDRs、FARs、QERs、BARs（转发规则）
- F-SEIDs（会话端点标识符）
- PGW-U 对等体关联

操作用例

监控 **PFCP** 关联健康:

```
1. 打开 UPF 状态页面
2. 验证所有对等体显示“关联”
3. 检查丢失的心跳计数 = 0
4. 如果对等体显示“未关联”:
   - 检查对等体 IP 可达性
   - 验证对等体正在运行
   - 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. 检查 PGW 会话页面 - 会话存在吗?
3. 检查 PFCP 会话页面 - PFCP 会话已创建?
4. 如果没有 PFCP 会话:
   - 检查 UPF 状态 - 是否有任何对等体关联?
   - 检查日志中的 PFCP 错误
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□:

1. 配置多个 PGW-U 对等体
2. 检查 PFCP 会话页面
3. 验证会话是否分布在对等体之间
4. 确定是否有一个对等体负载不成比例

检测对等体故障:

- 快速查看 UPF 状态页面
- 红色“未关联”徽章立即可见
- 丢失的心跳计数显示早期警告
- 根据 Web UI 数据设置监控警报

优点:

- 实时监控 - 无需查询指标或 SSH
- 可视状态 - 颜色编码的关联/未关联
- 对等体健康趋势 - 丢失的心跳计数显示早期警告
- 会话级检查 - 查看确切的 PDRs/FARs/QERs 编程
- 无需工具 - 只需一个网页浏览器

相关文档

□置

- □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 执行的 PCC 规则
- **Diameter Gy** 接口 - 通过 URRs 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 - 通过 PFCP 进行 UE 地址分配

返回操作指南

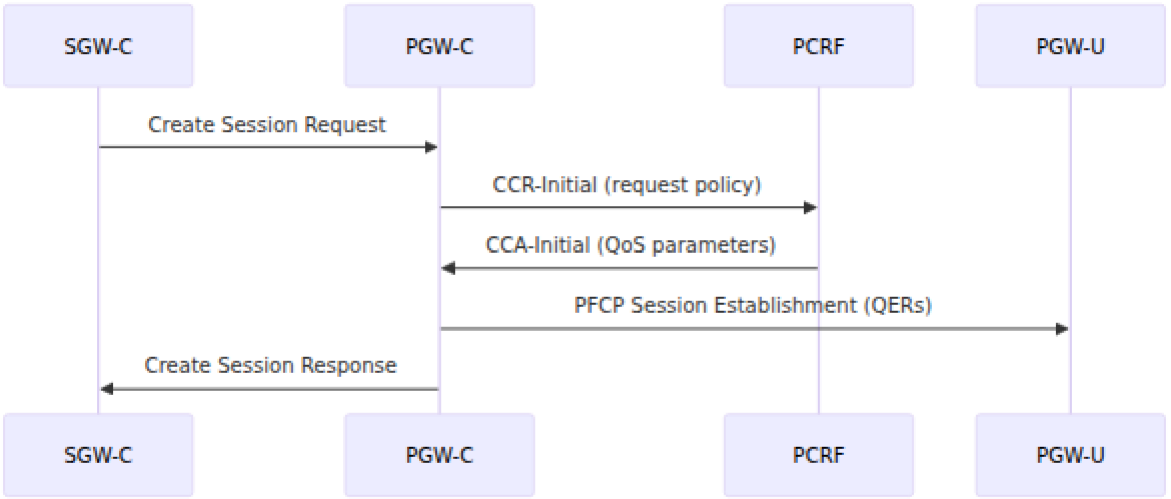
QoS 和承载管理

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)**：在用户平面中强制执行 MBR/GBR 限制和门状态
- 默认承载：始终与 PDN 会话一起创建，提供基本连接
- 专用承载：根据 PCRF 策略动态创建���提供特定的 QoS 保证

□置

重要提示：动态 QoS 策略

所有 QoS 参数通过 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□置，而不是在 **PGW-C** □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 发送 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 编程为承载 QoS 中的 MBR
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：来自 PCRF 的 Re-Auth Request (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 → 新的 PDR，带 SDF 过滤器
    - QoS-Information → 新的 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 响应 Create Bearer Response，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**，带更新的 Charging-Rule-Definition
- **PFCP Session Modification**，以更新现有的 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：

- **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：

- QCI 从 PCRF 接收，并在承载级 QoS IE 中向 SGW-C 信号
- PGW-C 不直接强制执行 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□和保留优先级)

来源：PCRF 通过 `Allocation-Retention-Priority` 分组 AVP

组件：

- 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）
- ���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：

- ARP 向 SGW-C 信号，最终传递到 eNodeB
- **PGW-C** 不强制执行 - 强制通常在 eNodeB 的无线接入控制中进行
- 在网络拥塞期间用于确定允许或删除哪些承载
- 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` AVPs

格式：每秒字节（内部转换为 kbps: `bytes / 1000`）

适用范围：所有承载（默认和专用）

工作原理：

- PGW-C 创建 QER，带有 `mbr: %Bitrate{ul: kbps_ul, dl: kbps_dl}`
- QER 通过 PFCP 发送到 PGW-U
- **PGW-U** 强制执行速率限制（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` AVPs

格式：每秒字节（转换为 kbps）

适用范围：仅适用于专用承载（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 强制执行最小比特率保证
- 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：

- 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` AVPs

范围：适用于 所有非 **GBR** 承载 的 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，并将其信号传递给 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：

- 通过 HSS/PCRF 中的订阅者配置文件设置
- 用于强制执行订阅层级（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 控制通过 `Flow-Status` AVP 在 Charging-Rule-Definition 中是否允许流量：

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	两个方向都允许
3 = DISABLED	ul: CLOSED, dl: CLOSED	不允许流量
4 = REMOVED	ul: CLOSED, dl: CLOSED	正在删除承载

用例：

- **DISABLED**：用于停放服务或信用耗尽（数据包丢弃但承载保留）
- **ENABLED-UPLINK**：不常见，但可以用于仅上传服务
- **ENABLED-DOWNLINK**：下载仅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

���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
```

Web UI 监控

PGW 会话页面 (`/pgw_sessions`):

- 按 IMSI、IP 地址、MSISDN 或 APN 搜索
- 查看每个会话的活动承载
- 检查承载 QoS 参数 (QCI, MBR, GBR, AMBR)
- 实时自动刷新 (2 秒)

Diameter 页面 (`/diameter`):

- PCRF 对等连接状态
- Gx 会话计数
- 对等状态（已连接/未连接）

日志页面 (`/logs`):

- 实时日志流

- 通过 "Credit Control" 过滤 CCR/CCA ���换
- 通过 "Re-Auth" 过滤 RAR 事件（策略变化）
- 通过 "PFCP" 过滤用户平面编程事件

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
（策略变化）
[debug] Sending Session Establishment Request        # PFCP 到 PGW-
U（编程 QERs）
[debug] Sending Session Modification Request         # PFCP 到 PGW-
U（更新 QERs）
```

操作任务

验证应用于会话的 QoS

1. 访问 Web UI → **PGW** 会话 页面
2. 搜索 IMSI（例如，`999000123456789`）
3. 展开会话详细信息
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 表示非 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：
   - 访问 Web UI → **Diameter** 页面
   - 验证 PCRF 对等状态 = "connected"
   - 如果未连接，请检查网络连接和 Diameter 配置
2. 验证 **CCR/CCA** 交换：
   - 访问 Web UI → **Logs** 页面
   - 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：
   - 搜索日志中的 "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 中的订阅者配置文件
   - 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

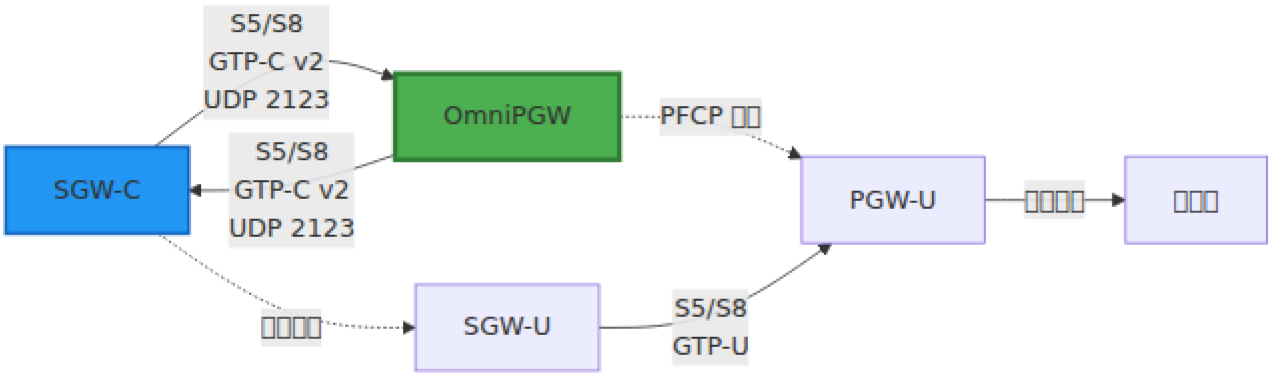
S5/S8 接口文档

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 分配用于传入消息
- 远端 **TEID** - 由 SGW-C 分配用于传出消息

```
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 请求超时时间（毫秒）（默认: 500ms）
    # 等待 GTP-C 响应时每次尝试的超时时间（创建承载、删除承载等）
    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`** - 每次重试尝试的超时时间（毫秒）（默认：500ms）
- **`request_attempts`** - 放弃前的重试次数（默认：3）

总等待时间：`request_timeout_ms × request_attempts`

默认行为：500ms × 3 次尝试 = 总最大等待 **1.5** 秒

调优指南：

网络延迟	推荐的 **`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便通过 Charging-Rule-Remove 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 **IEs**:

IE 名称	类型	描述
原因	结果	成功或错误代码
承载上��文	分组	承载信息
PDN 地址分配	IP	分配的 UE IP 地址（请参见 UE IP 分配）
APN 限制	枚举	APN 使用限制
PCO	选项	协议配置选项（请参见 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 (来自 PFCP 的 PGW-U 隧道端点)
├── 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
```

删除会话请求

方向: SGW-C → OmniPGW

目的: 终止 PDN 连接

关键 **IEs**:

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 **IEs**：

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

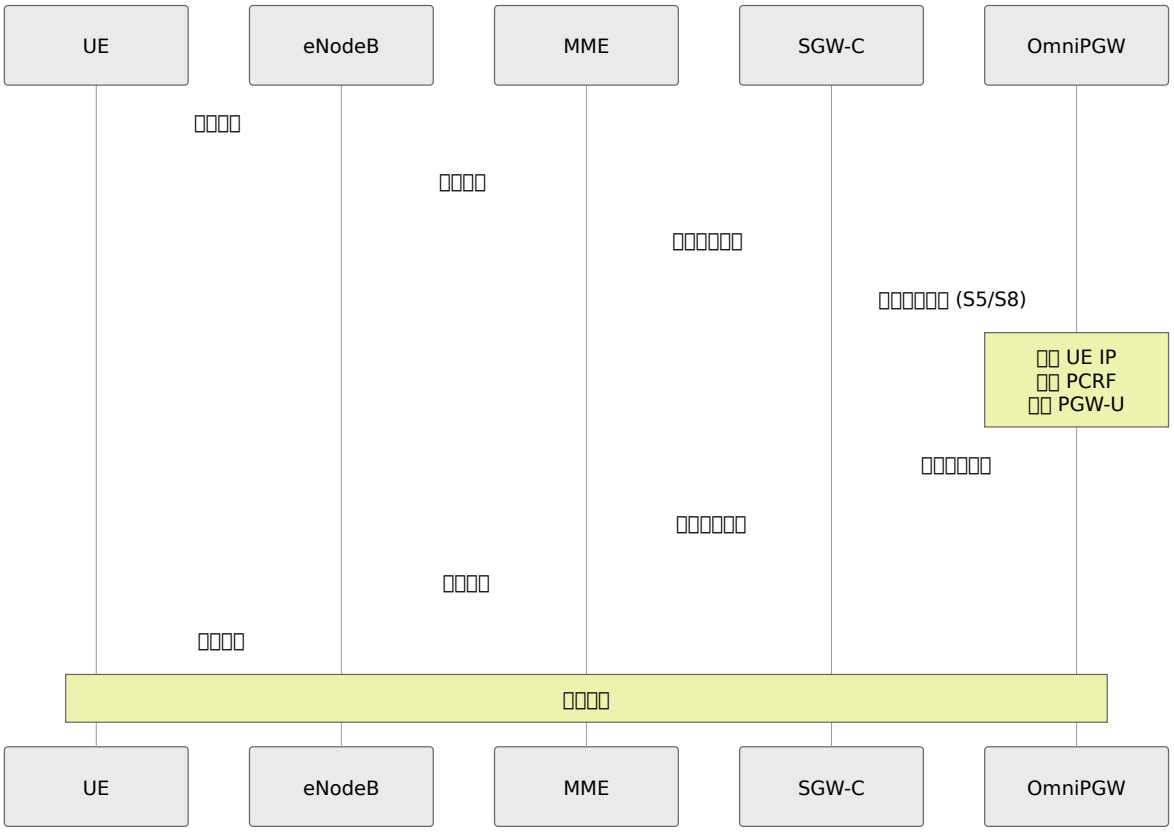
目的：删除专用承载

场景：

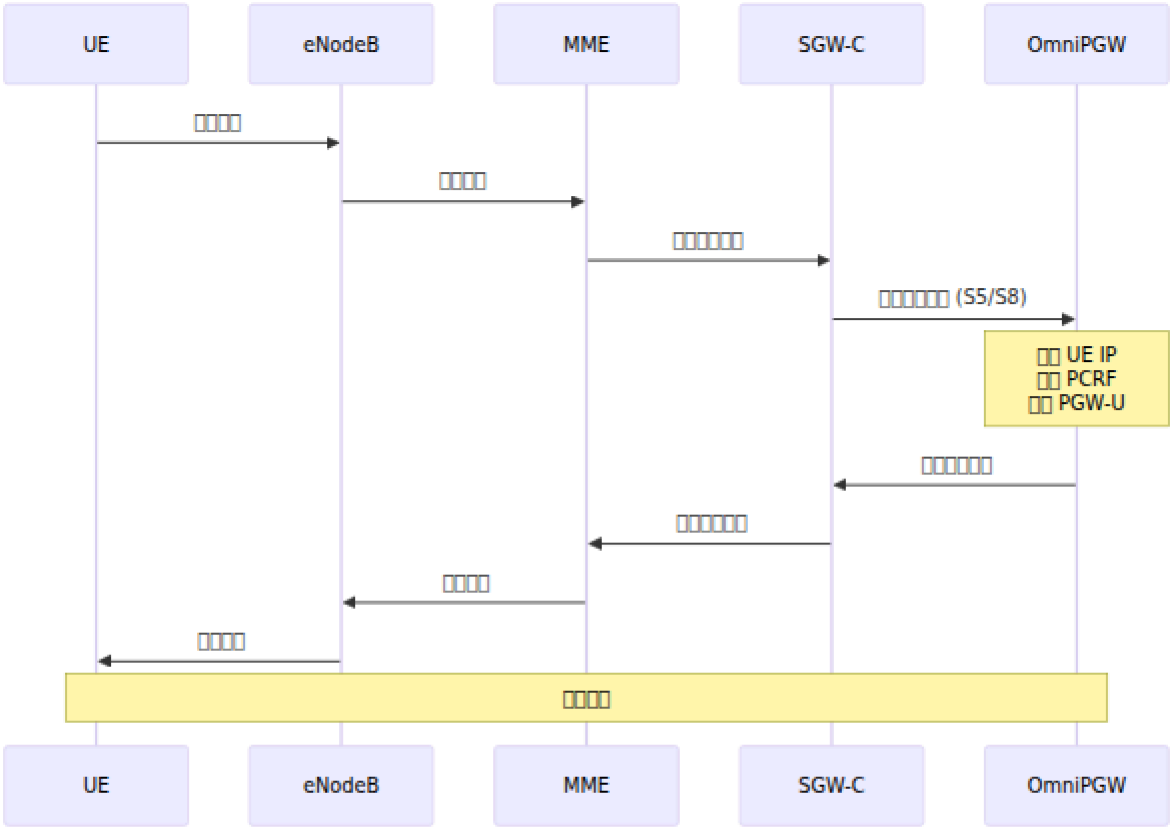
- **PGW** □□□ PCRF □□□□□□□□□□□□
- **SGW** □□□ □□□□□□□

□□□

□□□□


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（永久故障）

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	不支持的服务	无效的 APN
94	TFT 中的语义错误	无效的流量流模板

错误（瞬态故障）

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 停止工作
  → 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:

- 确保唯一的 TEID 分���
- 检查 TEID 注册表以查找泄漏

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 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
   - 一般: 每个套接字 4-8 MB
2. 连接限制
   - 计划预期的会话数量
   - 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
    - 拒绝格式错误的 GTP-C 数据包

相关文档

核心功能

- □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□ - 通过创建会话响应交付 IP 地址
- **PCO** □置 - GTP-C 消息中的 PCO 参数

相关接口

- **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���的策略集成
- 直径 **Gy** 接口 - 与承载管理的计费集成

操作

- 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

返回操作指南

OmniPGW S5/S8 接口 - *□ Omnitouch □□□□□□*

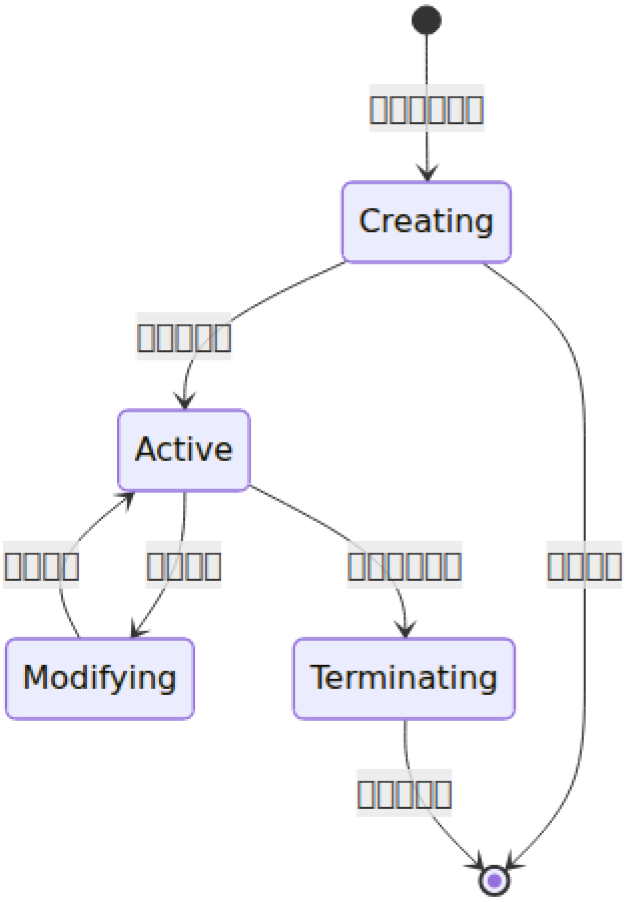
会话管理指南

PDN 连接生命周期和操作

OmniPGW □ Omnitouch □□□□□□

概述

一个 **PDN（**分组数据网络**）** 会话 代表了通过 OmniPGW 的 UE 数据连接。每个会话协调多个接口和资源以实现数据连接。



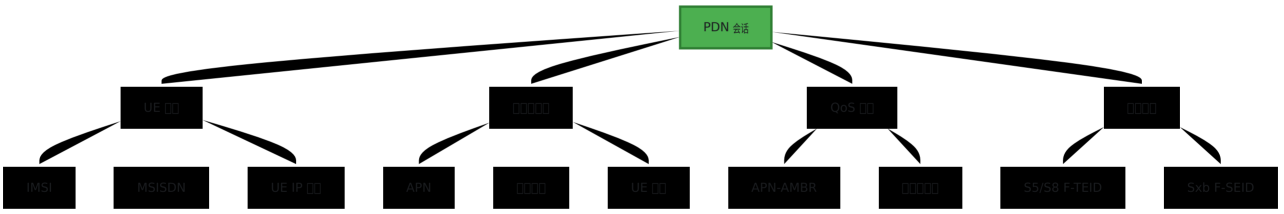
会话组件

会话标识符

每个会话都有多个用于不同接口的标识符：

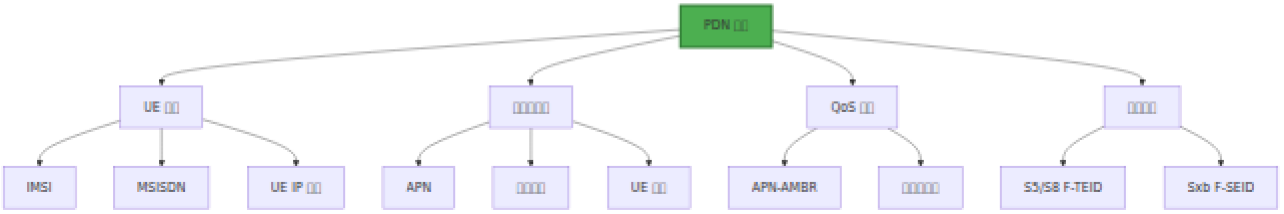
标识符	接口	目的
TEID	S5/S8 (GTP-C)	SGW-C 通信的隧道端点 ID
SEID	Sxb (PFCP)	PGW-U 通信的会话端点 ID
Session-ID	Gx (Diameter)	PCRF 通信的 Diameter 会话
Charging-ID	计费	唯一的计费/收费 ID

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

```
- 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

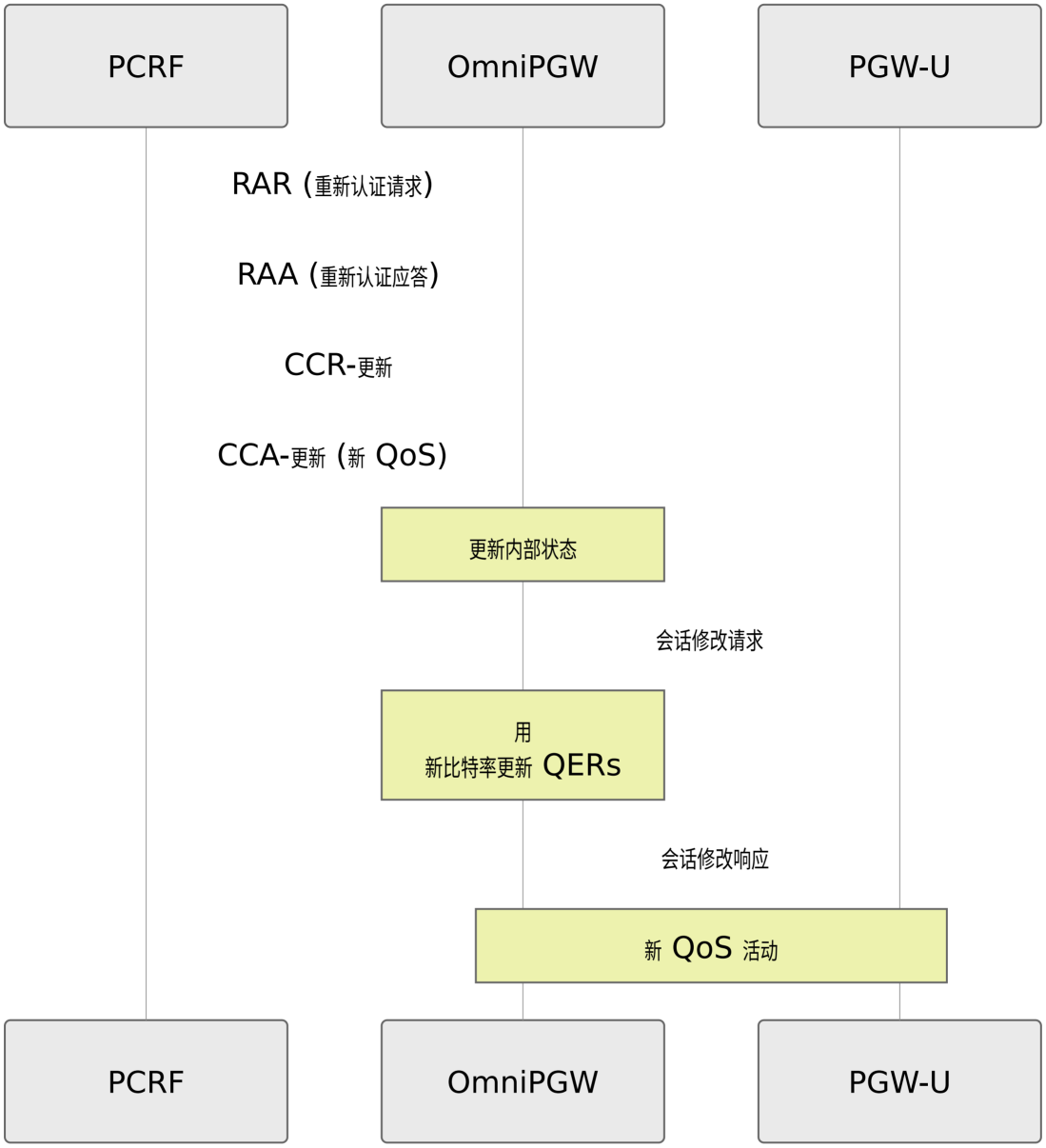
会话修改

触发器

会话可以由于以下原因被修改：

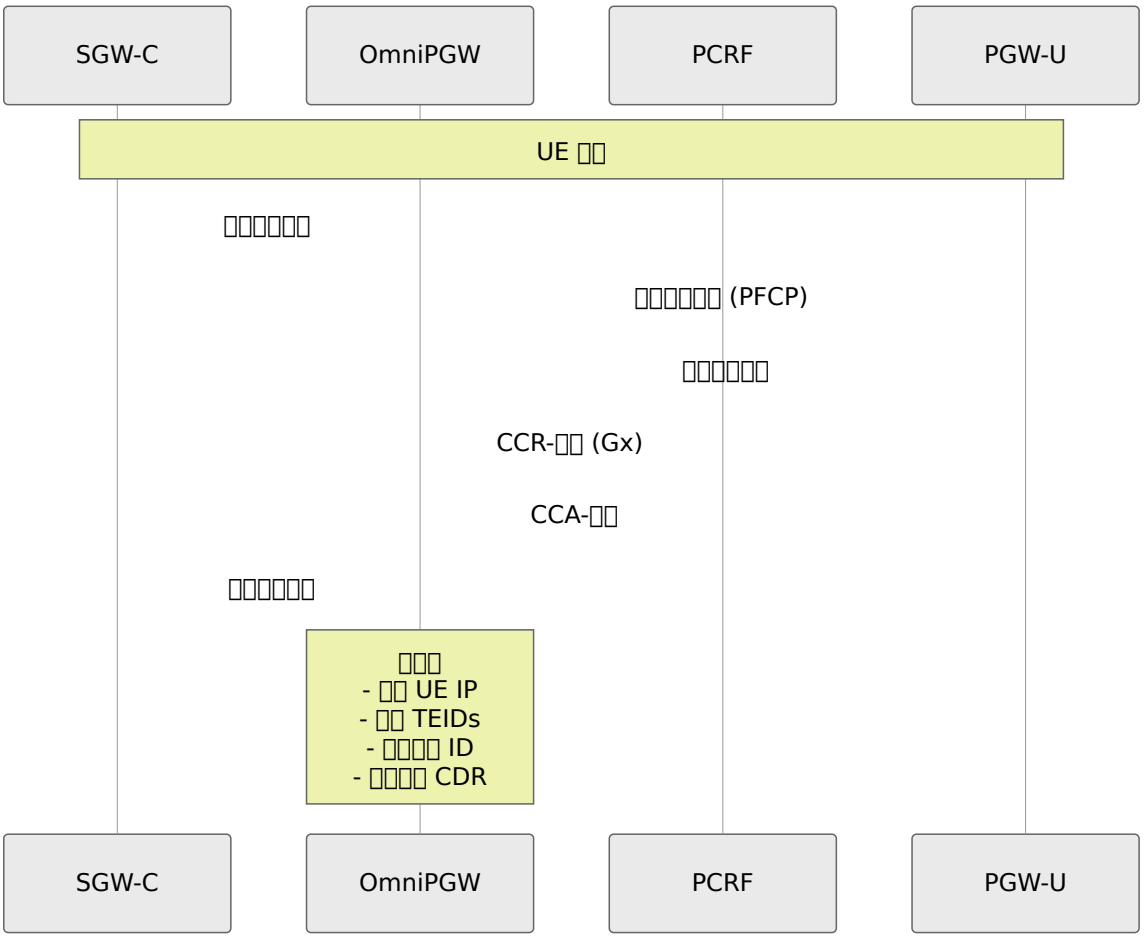
- **QoS** 变化 - PCRF 更新比特率
- 承载操作 - 添加/删除专用承载
- 切换 - SGW 更改
- 策略更新 - 来自 PCRF 的新 PCC 规则

QoS 修改流程


PCRF
OmniPGW
PGW-U
RAR (重新认证请求)
RAA (重新认证应答)
CCR-更新
CCA-更新 (新 QoS)
更新内部状态
会话修改请求
用
新比特率更新 QERs
会话修改响应
新 QoS 活动
PCRF
OmniPGW
PGW-U

□□□□

□□□□



□□□□

□□□□□□

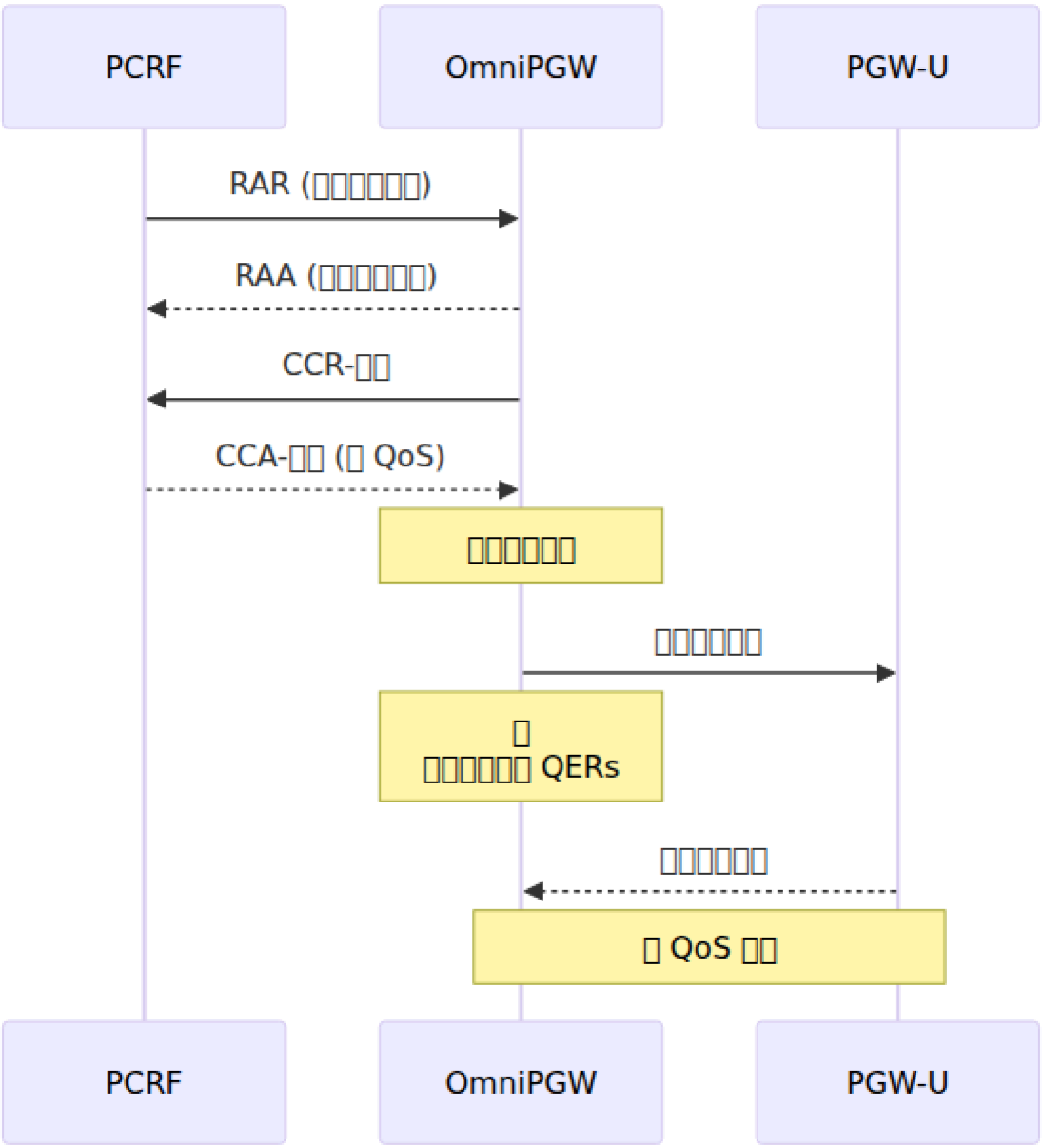
1. UE IP □□ → □□□□
2. TEID → □□□□□□□
3. SEID → □□□□□□□
4. □□-ID → □□□□□□□
5. □□-ID → □□
6. □□□□□□

□□□□□□□□

- 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���问 - Gx 超时
3. **PGW-U** 关闭 - 没有 PFCP 对等体可用
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 连接性
# 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/过时

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. 会话进程崩溃未进行清理
3. 注册表泄漏

解决方案:

```
# 重启 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 附着失败
- 创建会话响应带有错误原因

常见原因及响应:

原因值	意义	行动
用户未知	PCRF 拒绝 (IMSI 不在数据库中)	提供用户信息
没有可用资源	IP 池耗尽	扩大 IP 池
远程对等体未响应	PCRF/PGW-U 超时	检查连接性
服务不支持	无效 APN	配置 APN 池

最佳实践

会话限制

置适当的容量：

```
预期并发用户：10,000
每用户的会话开销：~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过时会话
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高可用性（未来）
- DNS/负载均衡器用于故障转移

会话数据元素

会话存储了什么信息?

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 连接详情:

- 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 执行规则): 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��: Diameter 会话-ID, 请求计数器

Web UI - 实时会话监控

OmniPGW 包含一个实时 **Web UI** 用于监控活动会话，无需查询指标或日志。

UE 搜索与深入分析

访问: `http://<omnipgw-ip>:<web-port>/ue_search`

目的: 搜索特定 UE 会话并查看详细信息

功能:

1. 搜索功能 按以下条件搜索会话:

- **IMSI** (例如, “310170123456789”)
- **MSISDN** (电话号码)

- **IP** 地址（例如，“100.64.1.42”）

2. 搜索选项

- 下拉选择器选择搜索类型
- 实时搜索，结果即时显示
- 清晰的界面和搜索提示

3. 深入分析结果 找到后，显示全面的会话信息：

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

基站数据库集成：如果配置了 OpenCellID 数据库，界面将显示：

- 基站的地理坐标（纬度/经度）
- 嵌入的 Google 地图显示确切的基站位置
- UE 最近已知小区站点的可视化地图

请参见下面的 基站数据库设置 以获取配置说明。

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，已用配额
- 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
- 已安装的计费规则 (来自承载的 PCC 规则)

f) 最近事件

- 此用户的事件历史
- 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

PGW 会话页面

访问：`http://<omnipgw-ip>:<web-port>/pgw_sessions`

目的**:** 实时查看所有活动 PDN 会话

功能:

1. 会话概述

- 实时会话计数 (每 2 秒更新)
- 所有活动会话的网格视图
- 无需刷新 - 自动更新

2. 快速会话信息 每个���话可见:

- **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

3. 搜索功能 按以下条件搜索会话:

- IMSI (例如, “310260”)
- UE IP 地址 (例如, “100.64”)
- MSISDN / 电话号码

- APN 名称

4. 可扩展详细信息 点击任何会话行以查看完整详细信息：

- 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/下行以人类可读格式显示)
- 隧道标识符 (两个 TEIDs 的十六进制格式)
- 用于调试的进程 ID
- 完整的会话状态 (原始数据结构)

网络拓扑视图

访问：`http://<omnipgw-ip>:<web-port>/topology`

目的：网络连接和活动会话的可视化表示

功能：

1. 拓扑可视化

- 网络元素的可视化图
- 显示 PGW-C（控制平面）节点
- 连接的 HSS（家庭用户服务器）对等体
- 活动会话计数显示

2. 互动元素

- 缩放控件（+/-）
- 中心视图按钮
- 点击节点查看详细信息
- 显示连接状态（绿色 = 活动，红色 = 关闭）

3. 会话计数

- 实时活动会话计数器
- 自动更新
- 负载的可视化指示

用例:

- 一目了然地了解网络架构
- 验证对等连接
- 监控拓扑变化
- 快速网络健康检查

会话历史与审计日志

访问：`http://<omnipgw-ip>:<web-port>/session_history`

目的：跟踪历史会话事件和审计轨迹

功能:

1. 事件过滤

- 按事件类型过滤（所有事件，会话创建，会话删除，等）
- 日期范围选择（开始日期 / 结束日期）
- 按 IMSI, MSISDN, IP 地址或 TEID 搜索

2. 导出功能

- 导出为 CSV 以便分析
- 包括所有过滤结果
- 适用于合规和报告

3. 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:

- 合规审计轨迹
- 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

操作用例

会话验证:

1. 用户报告连接问题
2. 按 IMSI 或电话号码在 Web UI 中搜索
3. 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:

- 一目了然地查看活动会话计数
- 与许可容量进行比较
- 按 APN 确定使用模式

故障排除:

- 按任何标识符查找特定会话
- 检查完整的会话状态，无需 SSH/IEx
- 验证 SGW 和 PGW TEIDs 在系统之间是否匹配
- 检查 PCRF 应用的 AMBR 值

相对于指标的优势：

- 查看单个会话详细信息（指标显示汇总）
- 搜索和过滤功能
- 人类可读格式（带宽以 Mbps 显示，而不是 bps）
- 实时状态检查
- 无需命令行访问

基站数据库设置

OmniPGW 可以与 OpenCellID 数据库集成，以在 UE 搜索界面中显示基站位置。此功能使得根据其服务小区站点可视化用户的地理位置。

概述

配置后，UE 搜索界面将：

- 显示基站坐标（纬度/经度）
- 显示基站位置的嵌入 Google 地图视图
- 提供用户位置的可视化确认
- 帮助排查基于位置的路由问题

设置

访问基站页面 `http://<omnipgw-ip>:<web-port>/cell_towers` 并点击“重新下载数据库”按钮。这将触发自动的后台下载和导入过程。

功能：

- 从 OpenCellID.org 下载最新数据

- 自动提取并导入到 SQLite
- 在后台运行（需要 10-15 分钟）
- 通过 Web 界面显示进度通知
- 安全：仅在确认新下载成功后删除旧数据库

首次设置：当您首次访问基站页面时，它将显示设置说明和“重新下载数据库”按钮。只需点击它以初始化数据库。

数据库信息

数据库位置：

- SQLite DB: `priv/cell_towers.db`
- CSV 下载（临时）: `priv/data/cell_towers.csv.gz`
- 索引：在 MCC, MNC, LAC, CellID 上自动创建以加快查找速度

数据库大小：

- 从 OpenCellID.org 下载的压缩文件约为 107 MB
- 导入时间：根据硬件不同，约需 10-15 分钟

查找性能：

- 基站查找已被索引且非常快速（<1ms）
- 对会话建立没有性能影响
- 仅在查看 UE 搜索结果时进行查找

启用的功能

设置后，以下功能将可用：

UE 搜索页面：

- 当前位置信息部分显示基站坐标
- 嵌入的 Google 地图显示基站位置
- 用户最后已知小区站点的可视化表示

基站 **Web UI**：

- 查看数据库统计信息（总记录数，数据库大小，创建日期）
- 重新下载数据库按钮 - 一键更新到最新的 OpenCellID 数据
- 浏览基站数据库
- 按 MCC, MNC, LAC, Cell ID 搜索
- 查看基站的地理分布
- 如果数据库尚未配置，查看设置说明

操作优势：

- 快速识别用户的地理位置
- 验证漫游场景
- 排查基于位置的问题
- 支持紧急服务位置要求

更新数据库

OpenCellID 数据库由社区维护并定期更新。

要刷新本地数据库：

1. 导航到 `http://<omnipgw-ip>:<web-port>/cell_towers`
2. 点击“重新下载数据库”按钮
3. 在弹出对话框中确认操作
4. 等待 10-15 分钟以完成后台下载/导入
5. 刷新页面以查看更新的统计信息

推荐更新频率：每月或每季度

注意： OpenCellID 可能会限制下载速率。如果您最近下载过，请等待几个小时再尝试。

故障排除

重新下载失败：

- 检查与 OpenCellID.org 的互联网连接
- 验证防火墙允许 HTTPS 下载
- 检查磁盘空间（需要 ~200 MB 的可用空间）

- 检查应用程序日志以获取具体错误消息
- OpenCellID 可能正在限制速率 - 等待几个小时后再试
- 检查 Web UI 是否显示后台任务的错误消息

数据库写入错误：

- 检查 `priv/` 目录中的数据库写入权限
- 确保有足够的磁盘空间 (~150 MB 用于数据库)
- 验证应用程序是否有权限在 `priv/` 中创建/删除文件

未找到基站：

- 数据库可能未覆盖所有基站
- OpenCellID 是社区贡献的，可能存在空白
- 基站数据可能对新部署的站点过时

地图未显示：

- 检查浏览器 JavaScript 控制台中的错误
- 验证 Google 地图嵌入权限
- 检查基站坐标是否有效

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 执行

操作

- 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

返回操作指南

OmniPGW 故障排除指南

故障排除程序和常见问题

□ Omnitouch □□□□□□

目录

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

- □置指南 - 系统配置参考

故障排除工具

Web UI

访问: `http://<omnipgw_ip>:4000`

关键页面:

- **/pgw_sessions** - 实时会话查看器（按 IMSI、IP、MSISDN、APN 搜索）
- **/diameter** - Diameter 对等状态（Gx PCRF, Gy OCS）
- **/pfcp_peers** - PFCP 对等状态（PGW-U 连接性）
- **/logs** - 实时日志流，带过滤功能

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 获取完整的指标参考。

日志分析

Web UI: 访问 **/logs** 页面并使用搜索过滤器

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. 验证许可证服务器连接性:

   - 检查 `config/runtime.exs` 中 `:license_client` 下配置的 URL
   - 默认: `https://localhost:10443/api`

解决方案:

1. 验证许可证服务器是否可访问:

   ```
   curl -k https://<license_server_ip>:10443/api/status
   ```

2. 检查 **`config/runtime.exs`** 中的许可证□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证:

   - 产品名称: `omnipgwc`
   - 联系 Omnitouch 验证许可证状态

4. 在□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：

   - Web UI → **/diameter** 页面
   - 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:
    - Web UI → **/pfcp_peers** 页面
    - 查找 "Association: DOWN"
    - 指标: `pfcp_peer_associated` = 0
4. 检查 **APN** 配置:
    - 查看 `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. 识别过期会话: Web UI → **/pgw_sessions**, 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** 连接问题:

1. 检查网络连接: `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, 则重启 OmniPGW
5. 通过 Web UI → **/diameter** 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，设置 80% 的警报
- 监控 PCRF 连接性，设置 Diameter 对等警报
- 实施闲置会话的清理

问题：会话停留在中间状态

症状：

- 会话在 Web UI 中显示但不完整
- 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. 在 **Web UI** 中搜索会话：
    - **/pgw_sessions** → 按 IMSI 搜索
    - 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 "Session Establishment Request"（PFCP）
    - 查找 "Credit Control Request"（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 健康状况和日志
   - 验证 PFCP 关联: Web UI → **/pfcp_peers**
   - 从 SGW-C 发送删除会话请求以清理
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，在 `config/runtime.exs` 中增加 Gx 超时
   - 发送删除会话请求以清理
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
- 实施超时/清理机制以处理不完整的会话
- 对注册计数不匹配发出警报

PFCP / 用户平面问题

问题：PFCP 关联中断

症状：

- Web UI → **/pfcp_peers** 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" ��� "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：

   ```
   ping <pgw_u_ip>
   nc -u -v <pgw_u_ip> 8805
   ```

2. 检查 **PFCP** □置：

   - 查看 `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 UDP 8805 被允许
   - 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 Web UI → **/pfcp_peers** 验证（应显示 "Association: UP"）
3. 对于□置问题：
   - 修正 `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���键警报
- 监控 `pfcp_consecutive_heartbeat_failures`（警报阈值 > 2）
- 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_response"}` 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 软件缺陷

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. 在 **Web UI** 中检查会话状态：
    - **/pgw_sessions** → 按 IMSI 查找会话
    - 检查 `pdr_map`、`far_map`、`qer_map` 是否存在冲突
    - 查找重复 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）
    - 如有必要，扩展 PGW-U
    - 减少受影响 PGW-U 实例上的会话负载
3. 对于软件缺陷：
    - 捕获完整的会话状态（Web UI 会话详细信息）
    - 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
- 在预发布环境中测试专用承载创建
- 监控 `sxb_inbound_errors_total` 并设置警报

Diameter (Gx/Gy) 问题

问题：**PCRF** 对等断开连接（**Gx**）

症状：

- Web UI → **/diameter** 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 服务关闭
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:

```
ping <pcrf_ip>
telnet <pcrf_ip> 3868
```

2. 检查 **Diameter** □置:

   - 查看 `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 期望的匹配

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 TCP 3868 被允许
   - 测试连接性：`nc -v <pcrf_ip> 3868`

2. 对于 **PCRF** 服务关闭:

   - 重启 PCRF 服务
   - 等待自动重新连接（30 秒重试间隔）
   - 通过 Web UI → **/diameter** 验证

3. 对于□置不匹□:

   - 修正 `config/runtime.exs` 中的 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** 健康状况：

   - 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/工作者

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 放置在同一位置

3. 临时解决方法（增加超时）：
   - 编辑 `config/runtime.exs`：

```
config :pgw_c,
  diameter: %{
    transaction_timeout_ms: 10000  # 从默认的 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** 对等断开连接（**Gy**）

症状：

- Web UI → **/diameter** 显示 OCS 对等 "disconnected"
- 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 对等断开连接，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，会话可能被拒绝或在未计费的情况下允许（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□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
- Web UI → **/pgw_sessions** 显示许多活动会话
- 日志："IP 分配失败：池耗尽"

可能原因：

1. 池对于用户基数过小
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□置的池大小：

   - 查看 `config/runtime.exs` 中的 `ue.subnet_map`
   - 示例："10.0.0.0/24" = 254 个可用 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：
   - Web UI → **/pgw_sessions**
   - 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`：

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      "internet" => "10.0.0.0/22"  # 1022 个 IP（之前是 /24 = 254 个 IP）
    }
  }
```

2. 重启 OmniPGW
3. 验证：会话现在可以建立

长期（清理）：

1. 在 Web UI 中识别过期会话
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���话闲置超时
- 定期会话审计

问题：分□重复的 IP 地址

症状：

- UE 报告 IP 地址冲突
- 日志：警告 "IP 已分配"
- Web UI 中有两个会话具有相同的 IP 地址

可能原因：

1. 软件缺陷（罕见）
2. 崩溃后的数据库不一致
3. 手动干预错误

诊断：

1. 在 **Web UI** 中搜索 **IP**：
    - **/pgw_sessions** → 按 IP 地址搜索
    - 检查是否有多个 IMSI 具有相同的 IP
2. 检查日志：
    - 搜索 IP 地址
    - 查找 "IP 分配" 事件

解决方案：

1. 识别受影响的会话：
    - 注意具有重复 IP 的两个 IMSI
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
   - 所有会话将丢失（协调维护窗口）

预防：

- 监控重复分配（当前没有内置指标）
- 定期数据库完整性检查（如适用）

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（**Web UI**）

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 Web UI
curl http://<omnipgw_ip>:4000

# 检查指标端点
curl http://<omnipgw_ip>:9090/metrics

# 检查活动会话
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
- □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 - QoS 相关问题

返回操作指南

UE IP 池分□文档

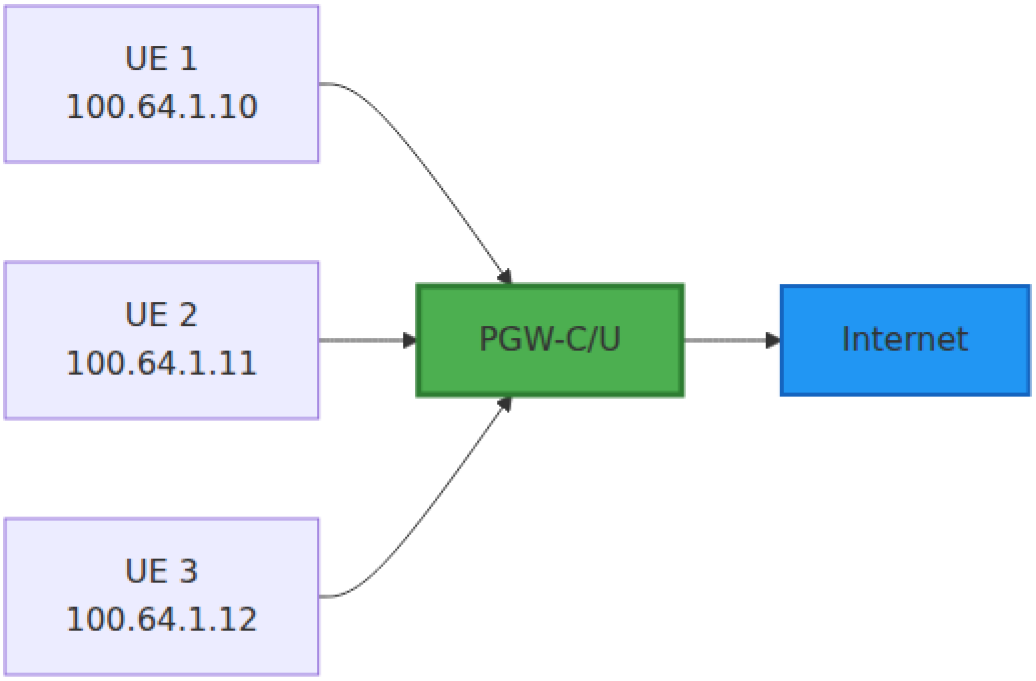
移动设备的 **IP** 地址管理

目录

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进行通信。

为什么 IP 分□很重要



每个 UE 从 PGW-C 收到一个唯一的 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	���持	描述
IPv4	□ □□	标准 IPv4 地址
IPv6	□ □□	IPv6 地址和前缀
IPv4v6	□ □□	双栈（IPv4 和 IPv6）

IP 分□概念

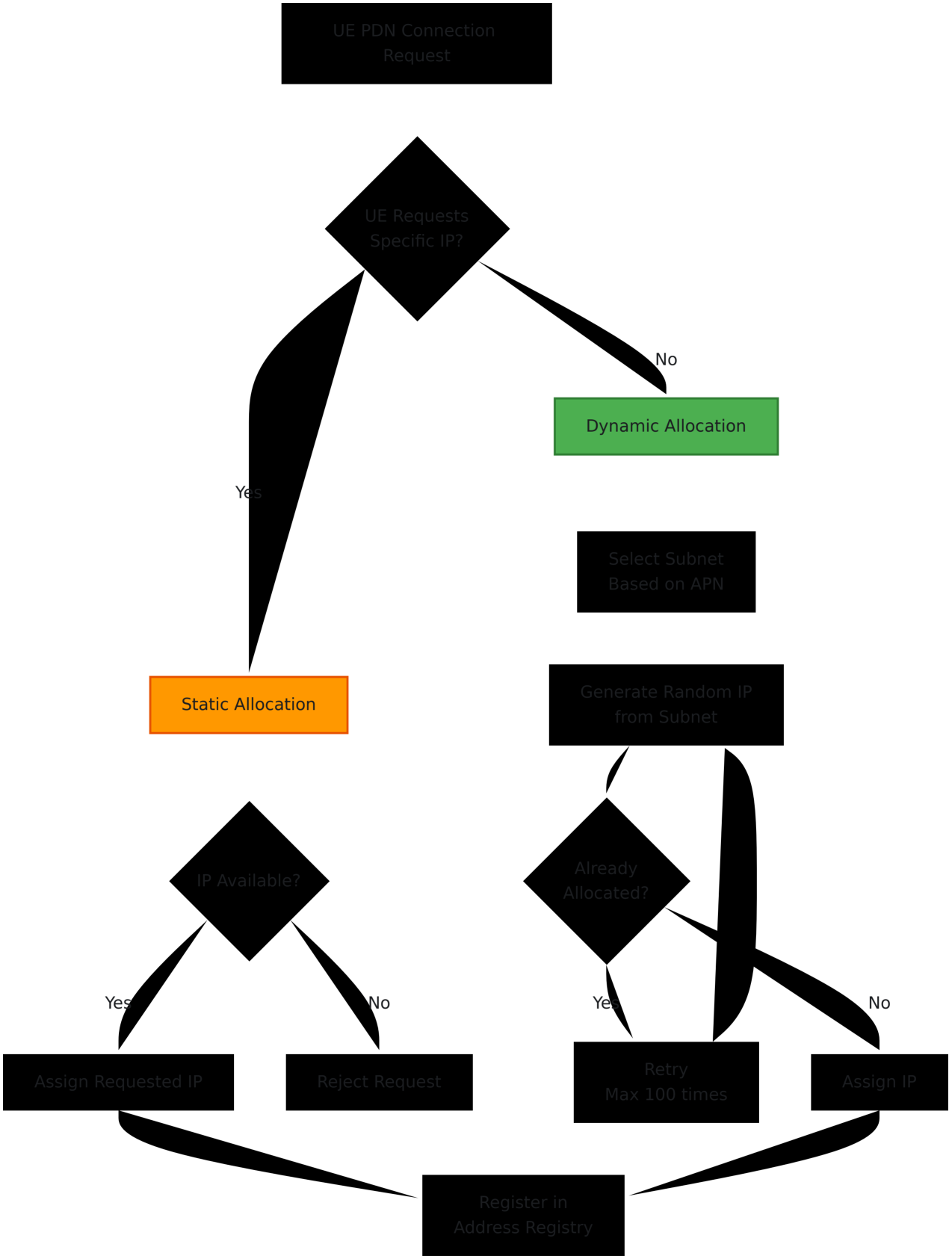
PDN 类型

当 UE 请求 PDN 连接时，它指定一个 **PDN** 类型：

PDN 类型	描述	分□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□方法

PGW-C 支持两种 IP 分配方法：



1. 动态分（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

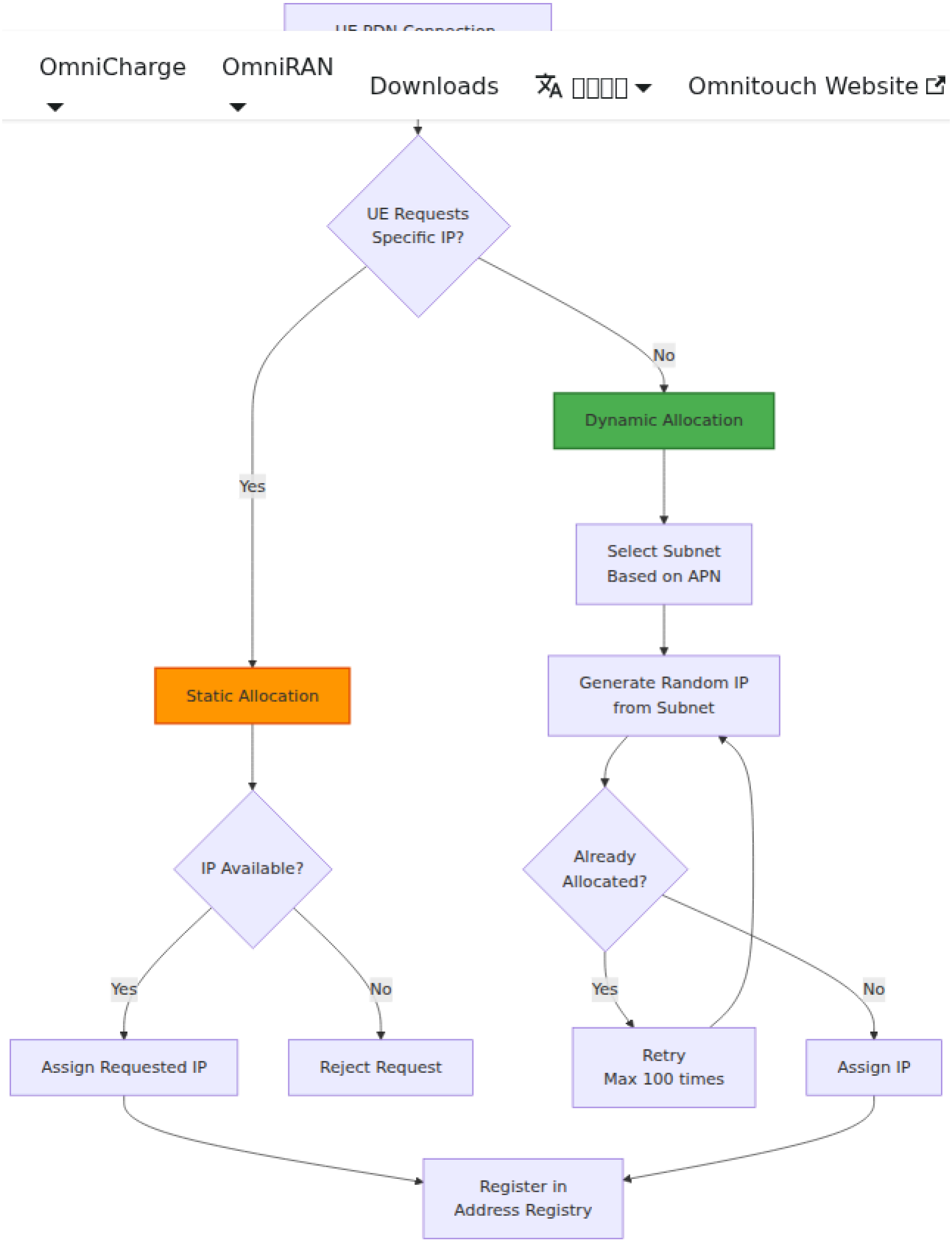
- 碰撞检测确保唯一性

2. 静态分□:

- 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 的企业设备非常有用

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

□置

基本□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"
      ],

      # 默认池用于未知 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）未分配
- 广播地址（例如，100.64.1.255）未分配
- 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）
- 路由策略灵活性

真实世界示例

```
config :pgw_c,
  ue: %{
    subnet_map: %{
      # 一般互联网访问
      "internet" => [
        "100.64.0.0/20"     # 4094 个用于一般用途的 IP
      ],

      # IMS（于 LTE 的语音）
      "ims" => [
        "100.64.16.0/22"    # 1022 个用于 IMS 的 IP
      ],

      # 企业 APN
      "enterprise.corp" => [
        "10.100.0.0/16"     # 65534 个用于企业的 IP
      ],

      # IoT 设备（低比特率）
      "iot.m2m" => [
        "100.64.20.0/22"    # 1022 个用于 IoT 的 IP
      ],

      # 默认回退
      default: [
        "42.42.42.0/24"     # 254 个用于未知 APN 的 IP
      ]
    }
  }
```

IPv6 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**）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）
      ]
    }
  }
```

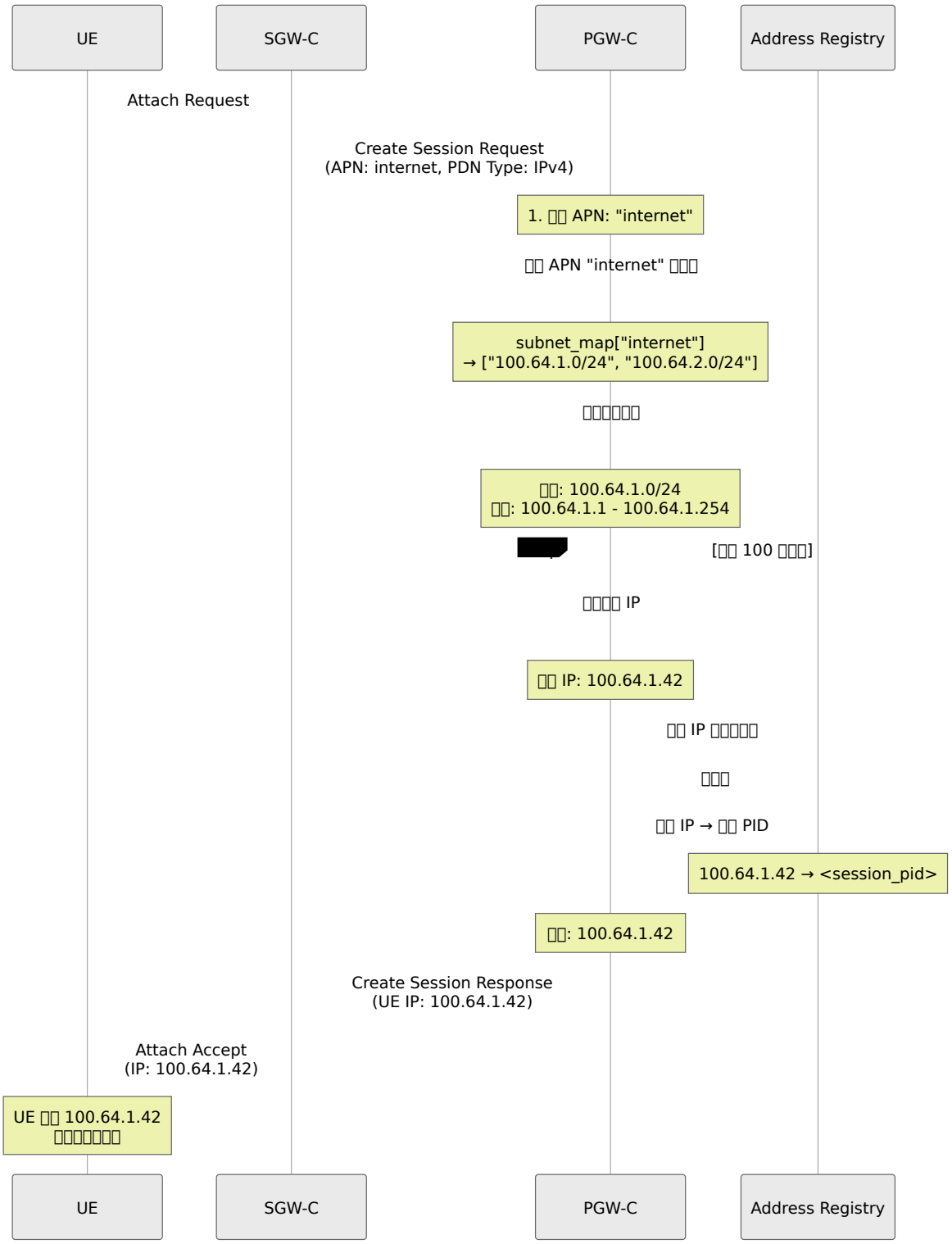
双栈分：

- UE 请求 PDN 类型: IPv4v6
- PGW-C 分配 IPv4 地址和 IPv6 前缀
- 两个地址同时处于活动状态

分□过程

IP 分配发生在会话创建期间，当 PGW-C 通过 S5/S8 接口接收创建会话请求时。有关 GTP-C 消息详细信息，请参见 S5/S8 接□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

□□□□

□□□□□□□□

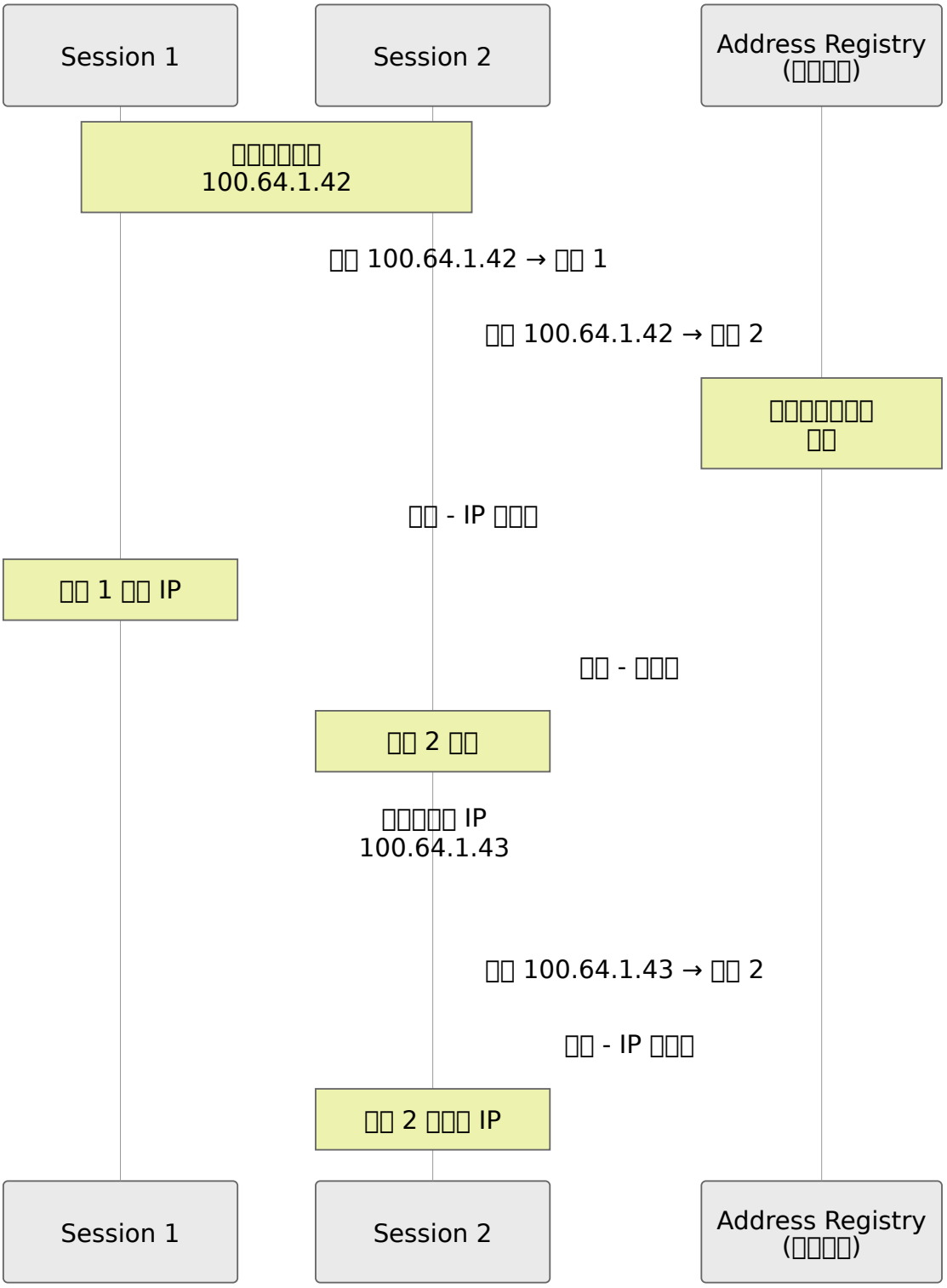
1. 子网查找：系统检索请求的 APN 的配置子网
2. 随机选择：从可用列表中随机选择一个子网
3. **IP** 生成：在子网范围内生成一��随机 IP
4. 唯一性检查：系统验证该 IP 是否未被分配
5. 重试逻辑：如果检测到碰撞，最多重试 100 次，使用新的随机 IP
6. 注册：一旦找到唯一 IP，就将其注册到会话

关键设计点：

- 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 分配模式以增强安全性
- 原子操作：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：两个会话尝试同时分配相同的 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
(□□□□□)

□□□□□□□□□□□□□□

- □□□□□□□□□□□□□□□

默认子网回退

场景：UE 请求未知 APN

示例配置：

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

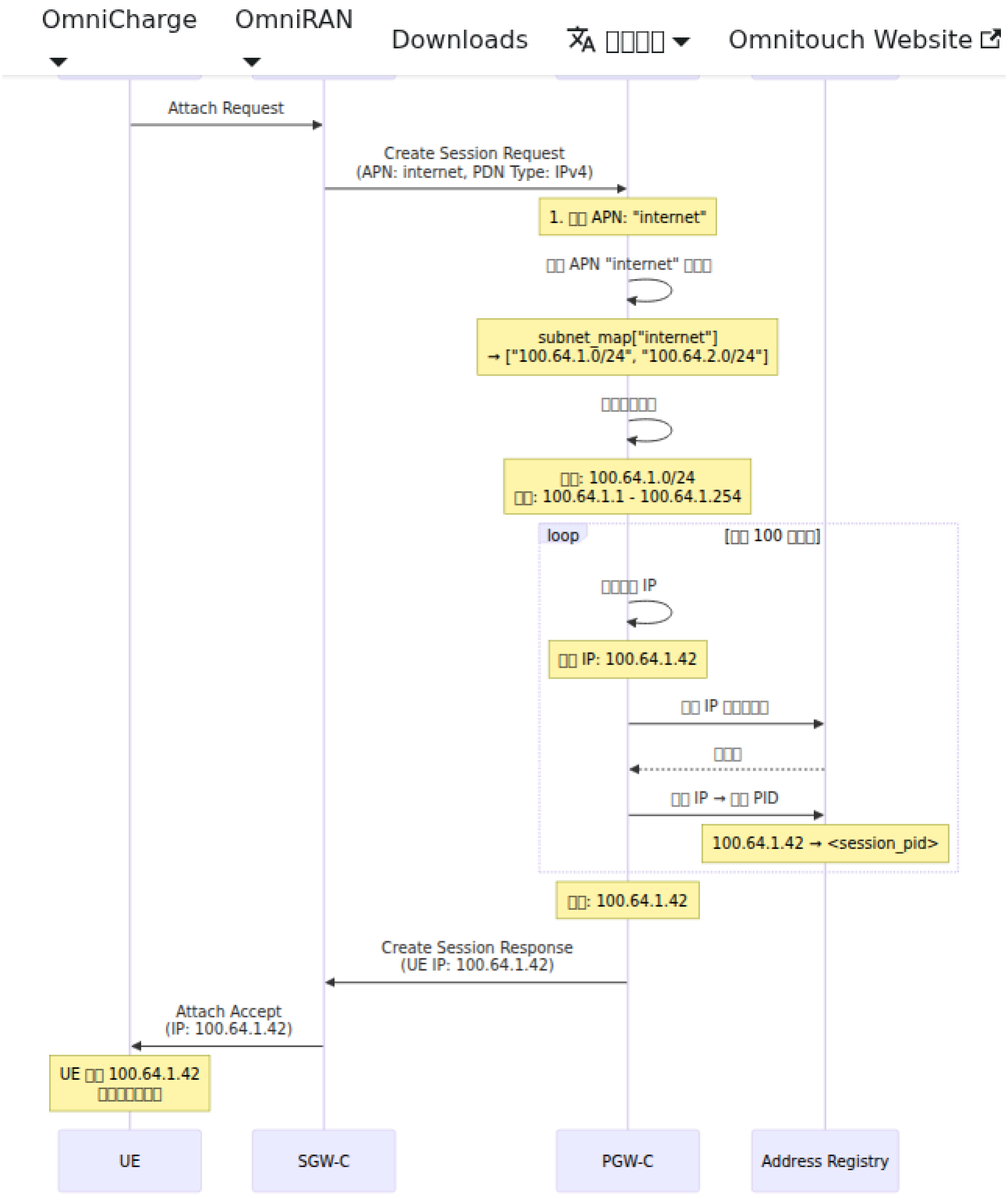
行为：

- 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 中分配 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□



自动清理：

- 当会话进程终止时，注册表进行清理
- IP 立即可用于新分配
- 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% 时发出警报

# 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□

用例：企业设备需要固定 IP

GTP-C 消息格式：

```
Create Session Request
├── 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□或拒绝:
    - ��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 收到它请求的确切 IP 地址
- 失败（**IP** 被使用）： 会话被拒绝 - IP 已分配
- 失败（**IP** 不在池中）： 会话被拒绝 - IP 不在配置范围内

IPv6 前缀委派

UE 请求 **IPv6**:

```
Create Session Request
├── PDN 类型: IPv6
```

PGW-C 分□ **/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 quintillion 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 要求

双栈分□

UE 请求 **IPv4v6**:

```
Create Session Request
├── PDN 类型: IPv4v6
```

PGW-C 同时分□两者:

```
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
- 两者同时处于活动状态
- 单独的 GTP 隧道（或双栈隧道）

私有与公共 **IP** 地址

私有 **IP** 池（**RFC 1918**）：

```
# 不可在公共互联网中路由
subnet_map: %{
  "internet" => [
    "10.0.0.0/8",
    "172.16.0.0/12",
    "192.168.0.0/16"
  ]
}
```

需要在 **PGW-U** 进行 **NAT** 以访问互联网

公共 **IP** 池:

```
# 可路由的公共 IP（仅示例）
subnet_map: %{
  "internet" => [
    "203.0.113.0/24"  # 公共 IP 块
  ]
}
```

不需要 **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

Web UI - IP 池管理

OmniPGW 提供实时 Web 界面，用于监控 IP 池的分配和利用情况。

访问: `http://<omnipgw-ip>:<web-port>/ip_pools`

功能：

1. 池概览

- 所有池的总 IP 数
- 当前已分配的地址
- 剩余可用 IP
- 实时利用率百分比

2. 每 **APN** 池状态 每个配置池显示：

- 池名称 - APN 标识符（例如，“default”，“ims.something.else”，“Internet”）
- **APN** 标签 - 配置的 APN 名称徽章
- **IP** 范围 - 显示子网范围的 CIDR 表示法
- 利用率 - 显示使用百分比的视觉指示器
- 分□统计：
  - 总计：池中的 IP 数量
  - 已分配：当前分配的 IP
  - 可用：剩余可分配的 IP

3. 实时更新

- 每 2 秒自动刷新
- 无需页面重新加载
- 实时利用率跟踪

用例：

- 维护前快速检查容量
- 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

```
# □池利用率警报
- 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"

# 池耗尽警报
- alert: UEIPPoolExhausted
  expr: address_registry_count >= 254  # 对于 /24 池
  for: 1m
  annotations:
    summary: "UE IP 池耗尽 - 无可用 IP"

# 分配失败警报
- 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□ **IP**

```
# 时间序列
address_registry_count
```

面板 **3:** 分□率

```
# 新分配的速率
rate(address_registry_count[5m])
```

面板 **4:** 池耗尽风险

```
# 距离耗尽的天数（□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（无 **IP** 可用）

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. □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 □置错误**

```
# 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□置：检查 `runtime.exs` 中的拼写错误

问题 2: IP 地址碰撞

症状：

- 两个 UE 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：

- 应自我修复（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 从意外子网接收 IP
- APN "internet" 从 "ims" 池获取 IP

原因：

- 子网映射配置不正确

验证：

```
# 检查确切的 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）
- 对于通用情况使用默认池

问题 4: IPv6 分□失败

症状：

- UE 请求 IPv6，收到错误

可能的原因：

1. 未□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
]
```

解决方案：

```
# 添加 IPv6 池
subnet_map: %{
  "internet" => [
    "100.64.1.0/24",
    "2001:db8:1::/48"  # IPv6 池
  ]
}
```

问题 **5:** 高池利用率

症状：

- 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

□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

□置

- □置指南 - UE IP 池配置，APN 子网映射
- **PCO** □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 分配 UE 地址到 UPF

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

操作

- 监控指南 - IP 池利用率指标，分配跟踪
- 数据 **CDR** 格式 - CDR 中的 UE IP 地址用于计费关联

返回操作指南

OmniPGW 操作指南

OmniPGW - 数据包网关控制平面 **(PGW-C)**

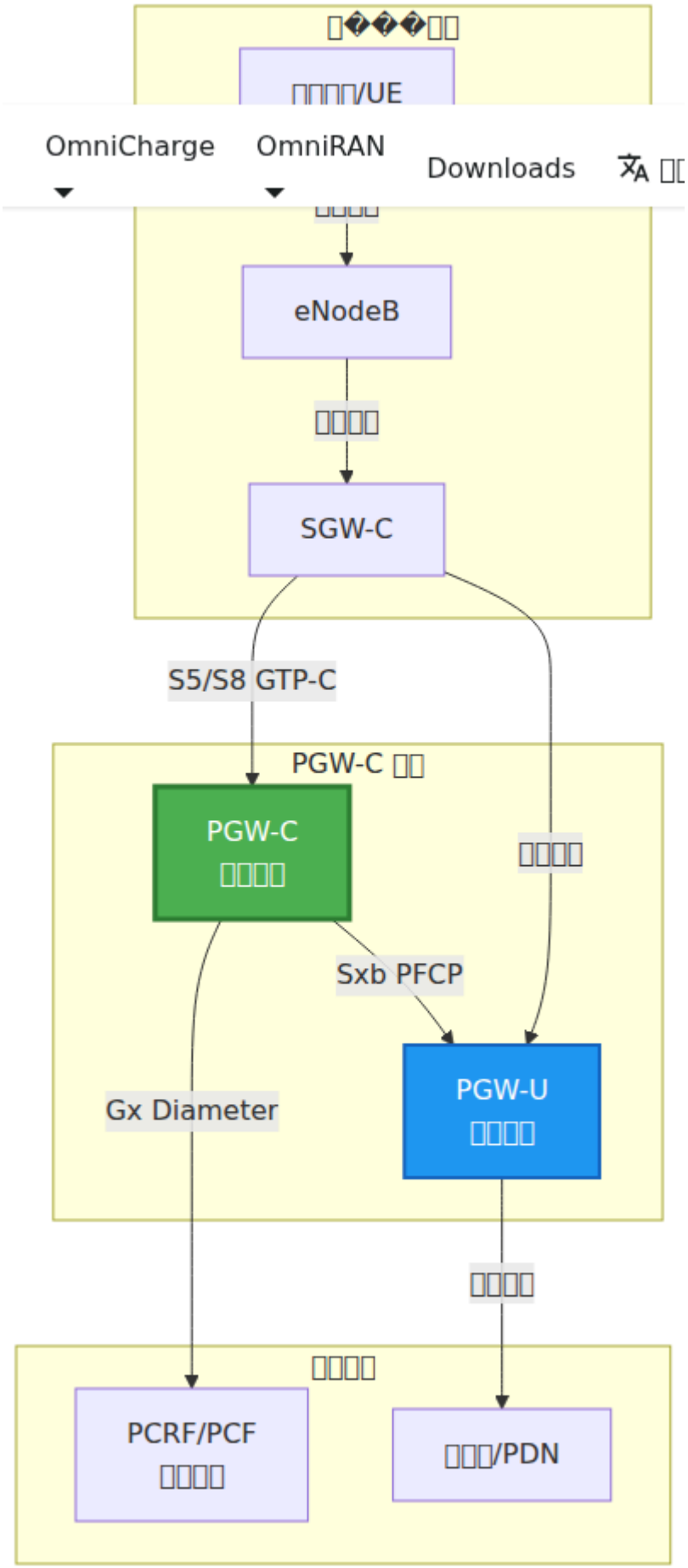
□ Omnitouch □□□□□□

目录

概述

OmniPGW 是一种高性能的数据包网关控制平面 (PGW-C) 实现，适用于 3GPP LTE 演进数据核心 (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、修改和终止用户设备 (UE) 数据会话
- **IP** 地址分□ - 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

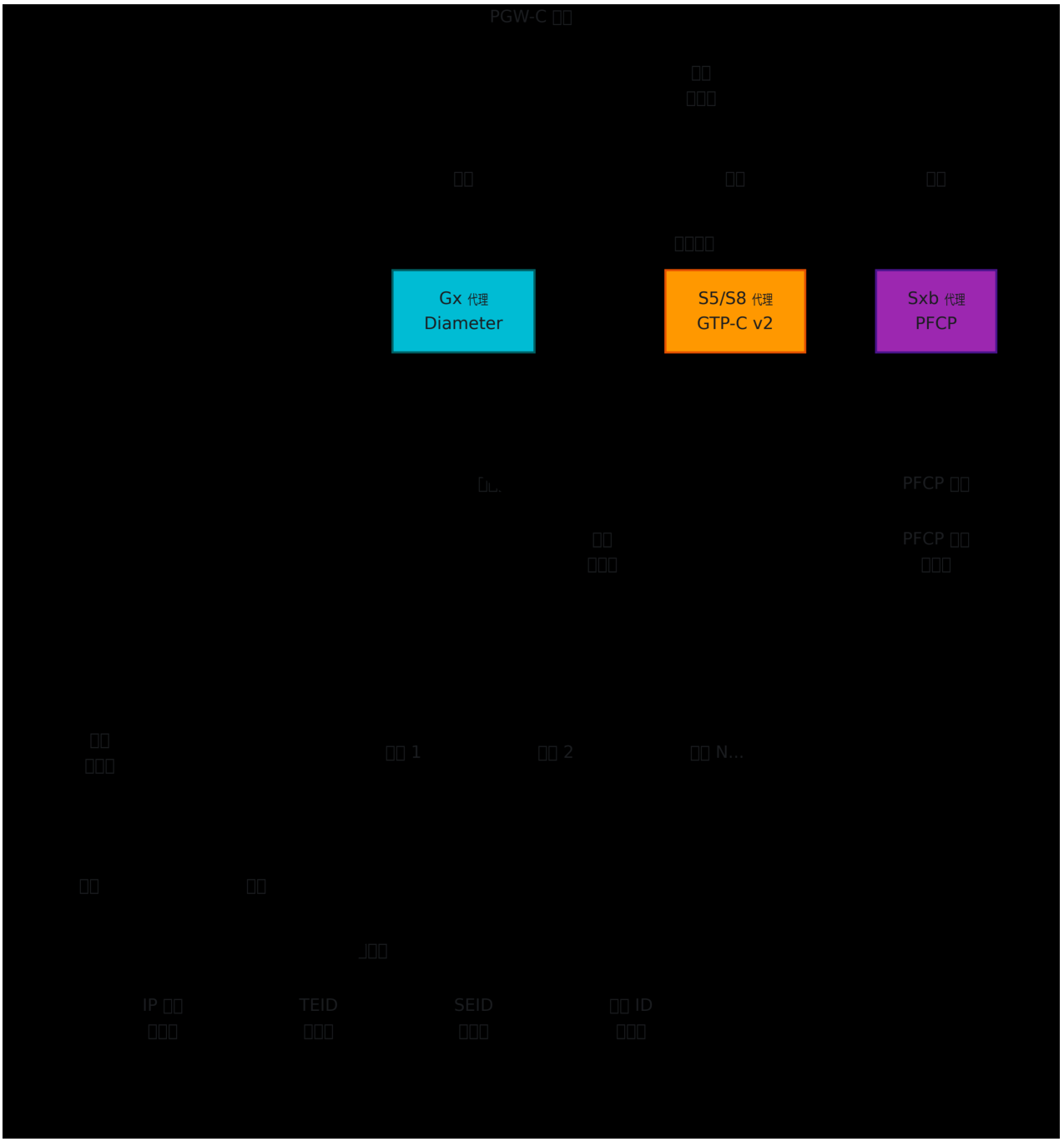

□□□/UE
OmniCharge
OmniRAN
Downloads
eNodeB
SGW-C
S5/S8 GTP-C
PGW-C □□
PGW-C
Sxb PFCP
Gx Diameter
PGW-U
□□□□
□□□□
PCRF/PCF
□□□□
□□□/PDN

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□置的子网池中分□ **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
- 跟踪计费系统的计费信息

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，并将在故障时自动重启，以确保系统的可靠性。

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 与 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：请参见 S5/S8 配置

Sxb 接口 (PFCP)

目的： PGW-C 与 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

□置：请参见 PFCP/Sxb 接口文档

Gx 接口 (Diameter)

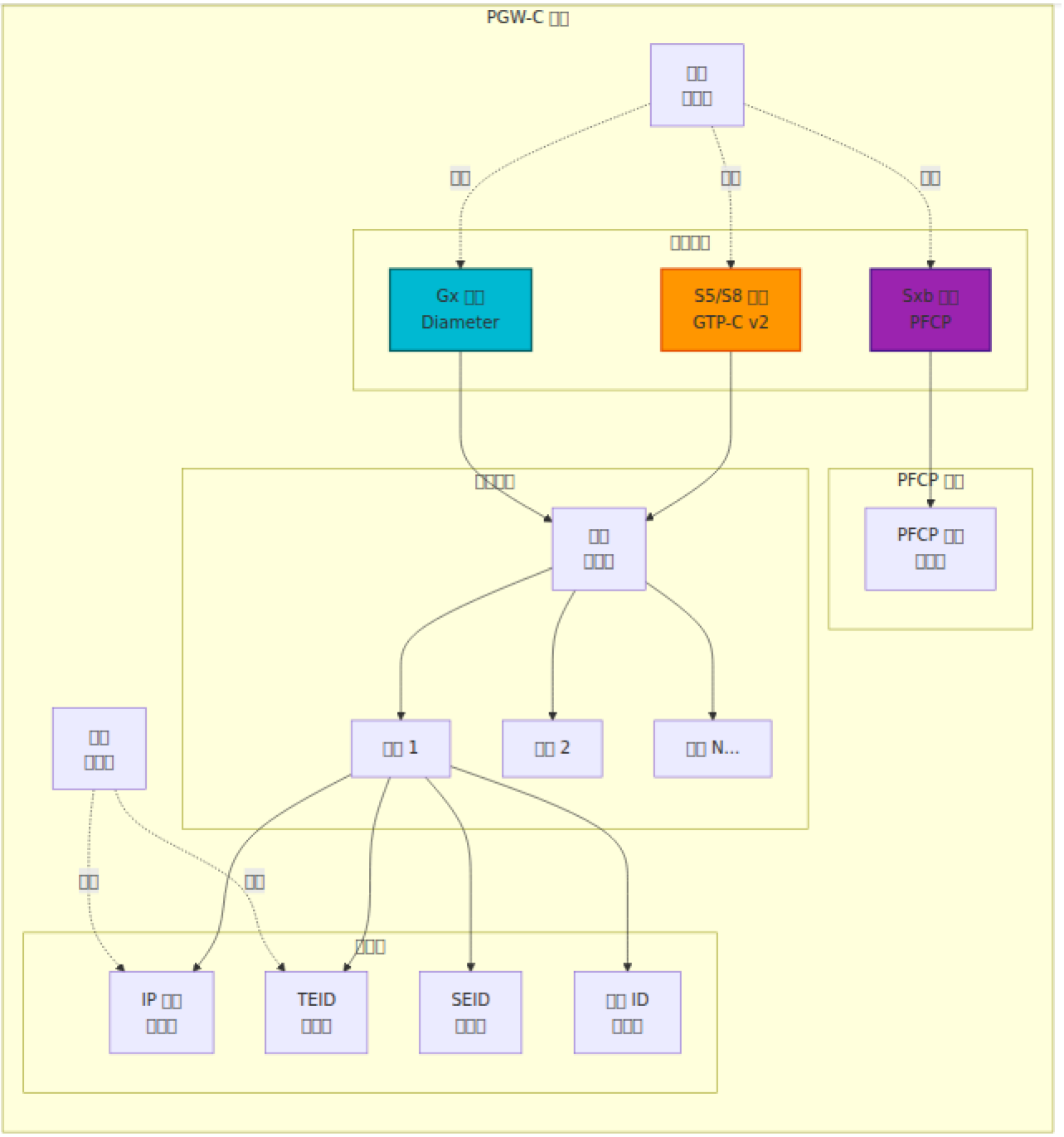
目的：策略与计费规则功能 (PCRF) 接口

协议： Diameter (IETF RFC 6733)

关键消息：

- 计费控制初始请求/应答 (CCR-I/CCA-I)
- 计费控制终止请求/□□答 (CCR-T/CCA-T)

□置：请参见 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 需求的额外承载
- **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 为 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) - 执行比特率限制
- **BAR** (缓冲动作规则) - 控制数据包缓冲

有关详细信息，请参见 PFCP 接口文档。

IP 地址分

UE IP 地址从配置的子网池中分配：

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

有关配置，请参见 UE IP 池分配。

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□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
   ```

验证操作

检查日志以确认成功启动:

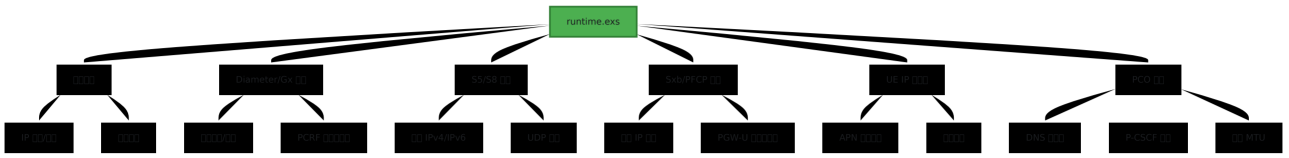
```
[info] 启动 OmniPGW...
[info] 在 127.0.0.42:42069 启动指标导出器
[info] 在 127.0.0.10 启动 S5/S8 代理
[info] 在 127.0.0.20 启动 Sxb 代理
[info] 启动 Gx 代理
[info] 启动 PFCP 节点管理器
[info] OmniPGW 成功启动
```

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

□置

所有运行时配置在 `config/runtime.exs` 中定义。配置结构分为几个部分：

□置概述



快速□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

有关详细信息，请参见 完整□置指南。

Web UI - 实时操作仪表板

OmniPGW 包含一个内置的 **Web UI**，用于实时监控和操作，提供对系统状态的即时可见性，无需命令行工具或指标查询。

访问 Web UI

```
http://<omnipgw-ip>:<web-port>/
```

可用页面：

页面	URL	目的	刷新率
UE 搜索	`/ue_search`	深入特定订阅者会话	按需
PGW 会话	`/pgw_sessions`	查看所有活动 PDN 会话	2 秒
会话历史	`/session_history`	会话事件的审计日志	5 秒
网络拓扑	`/topology`	可视化网络拓扑视图	5 秒
IP 池	`/ip_pools`	UE IP 地址池利用率	2 秒
PFCP 会话	`/pfcp_sessions`	查看与 PGW-U 的 PFCP 会话	2 秒
UPF 状态	`/upf_status`	监控 PFCP 对等体关联	2 秒
UPF 选择	`/upf_selection`	查看 UPF 选择规则和 P-CSCF 状态	静态
Diameter 对等体	`/diameter`	监控 PCRF 连接	1 秒
P-CSCF 监控	`/pcscf_monitor`	P-CSCF DNS 发现状态	5 秒
Gy 模拟器	`/gy_simulator`	测试 Gy/Ro 在线计费	按需
基站	`/cell_towers`	浏览 OpenCellID 数据库	静态
日志	`/logs`	实时日志流	实时

关键特性

实时更新：

- 所有页面自动刷新（无需手动重新加载）
- 来自 OmniPGW 进程的实时数据流
- 颜色编码状态指示器（绿色/红色）

搜索与过滤：

- 按 IMSI、IP、MSISDN 或 APN 搜索会话
- 无需页面重新加载的即时过滤

可扩展详细信息：

- 点击任何行以查看完整详细信息
- 检查完整会话状态
- 查看对等体配置和能力

无需身份验证（内部使用）：

- 从管理网络直接访问
- 设计用于 NOC/操作团队使用
- 仅绑定到管理 IP 以确保安全

操作工作流程

会话故障排除（深入分析）：

1. 用户报告连接问题
2. 打开 UE 搜索页面 (/ue_search)
3. 按 IMSI、MSISDN 或 IP 地址搜索
4. 查看全面的会话详细信息：
   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、地理位置
   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
   e) 策略信息 - Gx 会话、已安装的 PCC 规则
   f) 最近事件 - 会话历史和状态变化
5. 如果未找到会话 → 检查 Diameter 页面以获取 PCRF 连接
6. 如果存在位置问题 → 验证当前位置信息部分的□站数据

快速会话查找：

1. 用户报告问题
2. 打开 PGW 会话页面 (/pgw_sessions)
3. 按 IMSI 或电话号码搜索
4. 验证会话是否存在且□本详细信息正确：
   - 分配的 UE IP 地址
   - QoS 参数
   - 建立的隧道端点
5. 进行详细分析 → 点击会话以展开或使用 UE 搜索

系统健康检查：

1. 打开 UPF 状态页面 → 验证所有 PGW-U 对等体为 "已关联"
2. 打开 Diameter 页面 → 验证所有 PCRF 对等体为 "已连接"
3. 打开 PGW 会话 → 检查活动会话计数与容量

容量监控：

- 快速查看 PGW 会话计数
- 与许可/预期容量进行比较
- 确定高峰使用时间
- 监控 APN 之间的分布

Web UI 与指标

使用 **Web UI**:

- 深入分析订阅者故障排除 (UE 搜索)
- 单个会话详细信息和状态检查
- 实时对等体状态 (PFCP、Diameter)
- 跨所有接口的快速健康检查
- 按 IMSI/MSISDN/IP 故障排除特定用户
- 地理位置验证 (基站集成)
- 承载 QoS 分析 (MBR、GBR、QCI)
- 策略和计费规则检查
- 会话历史和审计跟踪
- IP 池容量监控
- 验证配置和规则

使用 **Prometheus** 指标:

- 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 - Web UI 用于即时操作，Prometheus 用于趋势和警报。

监控与指标

除了 Web UI, OmniPGW 还公开了与 Prometheus 兼容的指标以供监控:

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

访问指标

指标通过 HTTP 在配置的端点公开：

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

请参见 监控与指标指南 以获取仪表板设置和警报。

详细文档

本节提供了 OmniPGW 文档的全面□述。文档按主题和用例组织。

文档结构

```
OmniPGW 文档
├── OPERATIONS.md (本指南)
│
└── docs/
    ├── 配置与设置
    │   ├── configuration.md          完整的 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
        └── monitoring.md             Prometheus 指标与警报
```

按主题分类的文档

□ 入门

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

⚙ □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
pco-configuration.md	协议配置选项 (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+
s5s8-interface.md	GTP-C S5/S8 接口到 SGW-C	456

操作与监控

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

高级特性

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

文档特性

Mermaid 图表

所有文档都包含 **Mermaid** 图表 以便于可视化理解：

- 架构图
- 序列图（消息流）
- 状态机
- 网络拓扑

实用示例

每个文档都包括：

- 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

每个接口文档都包括：

- 常见问题和解决方案
- 调试命令
- 诊断指标

交叉引用

文档之间有广泛的交叉链接，以便于导航。

阅读路径

对于网络操作员

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□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

- □ □□/□□□□□□
- □ OTP/Elixir □□
- □ □□□□
- □ □□ GenServer □□□

□□

- □ PFCP (□□□□□□□□□□)
- □ GTP-C v2 (GPRS □□□□)
- □ Diameter (RFC 6733)

3GPP □□

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

□□

- □ □□□□
- □ IP □□□□
- □ QoS □□
- □ □□□□
- □ □□□□□

附加资源

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

- 配置文件: config/runtime.exs
