OmniUPF API 文档

概述

OmniUPF API 提供了一个全面的 RESTful 接口，用于管理和监控基于 eBPF 的用户面功能。该 API 使所有 UPF 组件的实时控制和可观察性成为可能。

API 功能

会话管理：

- **PFCP** 会话：查询活动会话，查看会话详情，按 UE IP 或 TEID 过滤
- **PFCP** 关联：监控控制平面节点关联和状态

流量规则：

- 数据包检测规则 **(PDR)**：检查上行和下行流量分类器 (IPv4/IPv6)
- 转发动作规则 **(FAR)**：查看转发、缓冲和丢弃策略
- **QoS** 强制规则 **(QER)**：监控速率限制和 QoS 策略
- 使用报告规则 **(URR)**：跟踪每个会话的数据量计数器

数据包缓冲：

- 缓冲状态：查看每个 FAR 的缓冲数据包 (`GET /buffer`, `GET /buffer/:far_id`)
- 缓冲操作：刷新或清除缓冲数据包 (`POST /buffer/:far_id/flush`, `DELETE /buffer/:far_id`, `DELETE /buffer`)
- 缓冲控制：手动通知触发 (`POST /buffer/:far_id/notify`)
- 通知状态：查看 DLDR 通知状态 (`GET /buffer/notifications`)

监控和统计：

- 数据包统计：按协议实时数据包计数 (GTP, IP, TCP, UDP, ICMP, ARP)
- **XDP** 统计：数据路径性能指标 (通过、丢弃、重定向、中止)
- **N3/N6** 接口统计：RAN 和数据网络流量分布

- 路由统计：FIB 查找性能（缓存命中、查找、错误）

路由管理：

- **UE** 路由：查询 UE IP 到 gNB 的路由表（`GET /routes`）
- **FRR** 集成：与 Free Range Routing 守护进程同步路由（`POST /routes/sync`）
- 路由会话：查看路由协议会话（`GET /routing/sessions`）
- **OSPF** 数据库：查询 OSPF 外部路由数据库（`GET /ospf/database/external`）

配置：

- **UPF** 配置：检索和编辑配置（`GET /config`，`POST /config`）
- 数据平面配置：查询数据平面特定配置（`GET /dataplane_config`）
- **XDP** 能力：查询 XDP 模式支持和接口能力（`GET /xdp_capabilities`）
- **eBPF** 映射容量：监控资源利用率和容量（`GET /map_info`）

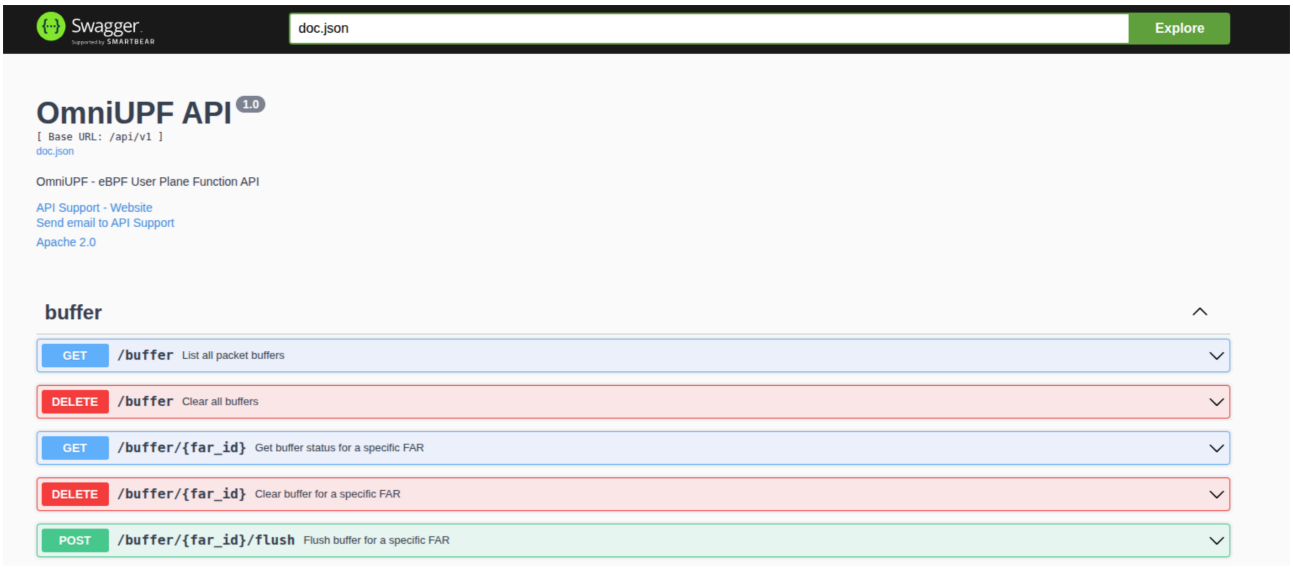
Web UI 集成

OmniUPF Web UI 基于此 API 构建，供了一个交互式仪表板，涵盖所有 API 功能。请参见 Web UI 指南 获取截图和使用示例。

Swagger API 档

该 API 使用 **OpenAPI 3.0 (Swagger)** 规范进行了全面文档化。交互式 Swagger UI 供：

- 完整端点文档，包含请求/响应模式
- 直接从浏览器测试 API 调用的尝试功能
- 所有数据模型的模式定义
- HTTP 状态码和错误响应



□□□ Swagger UI □□ OmniUPF API □□□□□□□□

访问 **Swagger UI**

Swagger 文档可在以下地址访问：

```
http://<upf-host>:8080/swagger/index.html
```

例如：`http://10.98.0.20:8080/swagger/index.html`

API 基础路径

所有 API 端点的前缀为：

```
/api/v1
```

````
## API □□

### □□

OmniUPF API □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□PDR □ URR □□□□□
□□

**□□□□□□□**□

1. **□□□□□□□**□□□□□
   - `page`□□□□□ 1 □□□
   - `page_size`□□□□□□□□□□100□□□□1000□

2. **□□□□□□□□**□
   - `offset`□□□□□□□□
   - `limit`□□□□□□□□□□□□1000□

**□□□□**□

```bash
# □□□□□□□□□□□□□ 50 □
GET /api/v1/pfcp_sessions?page=2&page_size=50

# □□□□□□□□□ 100 □□□□□ 50 □
GET /api/v1/pfcp_sessions?offset=100&limit=50

# □□□□□□□□□□□□□ 100 □
GET /api/v1/pfcp_sessions
````

□□□□□
````

```
&amp;#123;
  "data": [
    &amp;#123; /* session object */ &amp;#125;,
    &amp;#123; /* session object */ &amp;#125;,
    ...
  ],
  "pagination": &amp;#123;
    "total": 5432,
    "page": 2,
    "page_size": 50,
    "total_pages": 109
  &amp;#125;
&amp;#125;
```

分页端点：

- `/api/v1/pfcp_sessions` - PFCP 会话列表
- `/api/v1/pfcp_associations` - PFCP 关联列表
- `/api/v1/routes` - UE IP 路由
- `/api/v1/uplink_pdr_map` - 上行 PDR（基本信息）
- `/api/v1/uplink_pdr_map/full` - 带有完整 SDF 过滤详细信息的上行 PDR
- `/api/v1/downlink_pdr_map` - 下行 PDR IPv4（基本信息）
- `/api/v1/downlink_pdr_map/full` - 带有完整 SDF 过滤详细信息的下行 PDR IPv4
- `/api/v1/downlink_pdr_map_ip6` - 下行 PDR IPv6（基本信息）
- `/api/v1/downlink_pdr_map_ip6/full` - 带有完整 SDF 过滤详细信息的下行 PDR IPv6
- `/api/v1/far_map` - 转发动作规则
- `/api/v1/qer_map` - QoS 强制规则
- `/api/v1/urr_map` - 使用报告规则

缓冲管理端点：

- `GET /api/v1/buffer` - 列出所有 FAR 缓冲区及其统计信息
- `GET /api/v1/buffer/:far_id` - 获取特定 FAR 的缓冲状态
- `GET /api/v1/buffer/notifications` - 列出 DLDR 通知状态
- `DELETE /api/v1/buffer` - 清除所有缓冲数据包
- `DELETE /api/v1/buffer/:far_id` - 清除特定 FAR 的缓冲

- `POST /api/v1/buffer/:far_id/flush` - 刷新（重放）缓冲数据包
- `POST /api/v1/buffer/:far_id/notify` - 手动发送 DLDR 通知

配置端点：

- `GET /api/v1/config` - 获取当前 UPF 配置
- `POST /api/v1/config` - 更新 UPF 配置（运行时可编辑字段）
- `GET /api/v1/dataplane_config` - 获取数据平面特定配置

路由集成端点：

- `GET /api/v1/routes` - 列出 UE 路由
- `POST /api/v1/routes/sync` - 触发与 FRR 的路由同步
- `GET /api/v1/routing/sessions` - 获取路由协议会话
- `GET /api/v1/ospf/database/external` - 获取 OSPF 外部 LSA 数据库

最佳实践：

- 对于 Web UI 显示，使用 `page_size=100`
- 对于批量导出，使用 `page_size=1000`（最大限制）
- 查询 `pagination.total_pages` 以确定迭代次数
- 增加 `page_size` 以□高 API 性能（请求更少）

CORS 支持

跨源资源共享 (CORS) 默认对所有 API 端点启用，允许 Web UI 和第三方应用程序从不同来源访问 API。

Prometheus 指标

除了 REST API，OmniUPF 还在 `/metrics` 端点（默认端口 `:9090`）上公开 Prometheus 指标。

指标□供：

- 每个对等体的 PFCP 消息计数和延迟
- 按协议类型的数据包统计
- XDP 动作裁决

- 缓冲统计
- eBPF 映射容量利用率
- URR 体积跟踪

请参见 指标参考 获取完整文档。

相关档

- **Web UI** 指南 - 基于此 API 构建的交互式仪表板
- 指标参考 - Prometheus 指标文档
- **PFCP** 原因代码 - PFCP 错误代码和故障排除
- 规则管理指南 - PDR、FAR、QER、URR 配置
- 路由管理指南 - FRR 集成和 UE 路由
- 监控指南 - 统计监控和容量规划
- 配置指南 - UPF 配置选项
- **Swagger UI** - 交互式 API 文档（将 `localhost` 替换为您的 UPF 主机）

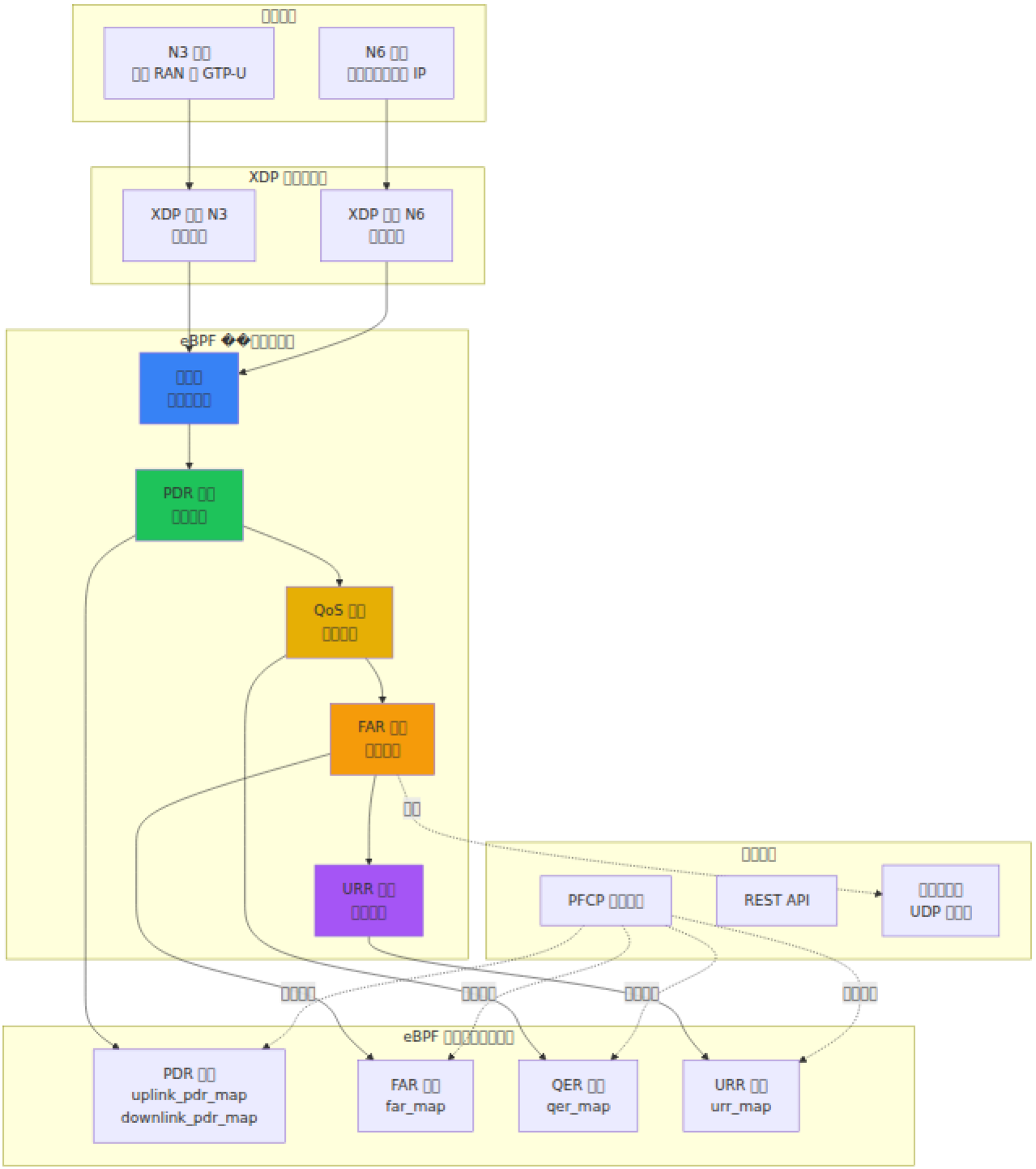
OmniUPF 架构指南

目录

概述

OmniUPF 利用 eBPF（扩展的伯克利数据包过滤器）和 XDP（快速数据路径）实现 5G/LTE 数据包处理的运营商级性能。通过直接在 Linux 内核中运行数据包处理逻辑，OmniUPF 消除了用户空间处理的开销，并以微秒级延迟实现多千兆位的吞吐量。

架构层



关键设计原则

零拷贝处理：

- 数据包完全在内核空间处理
- 内核与用户空间之间不进行数据拷贝

- 使用 XDP ���接操作数据包

无锁数据结构：

- eBPF 映射使用每 CPU 的哈希表
- 并发访问的原子操作
- 无互斥锁/自旋锁开销

硬件卸载准备：

- XDP 卸载模式支持 SmartNIC 执行
- 与支持 XDP 的网络卡兼容
- 回退到驱动原生或通用模式

eBPF 技术基础

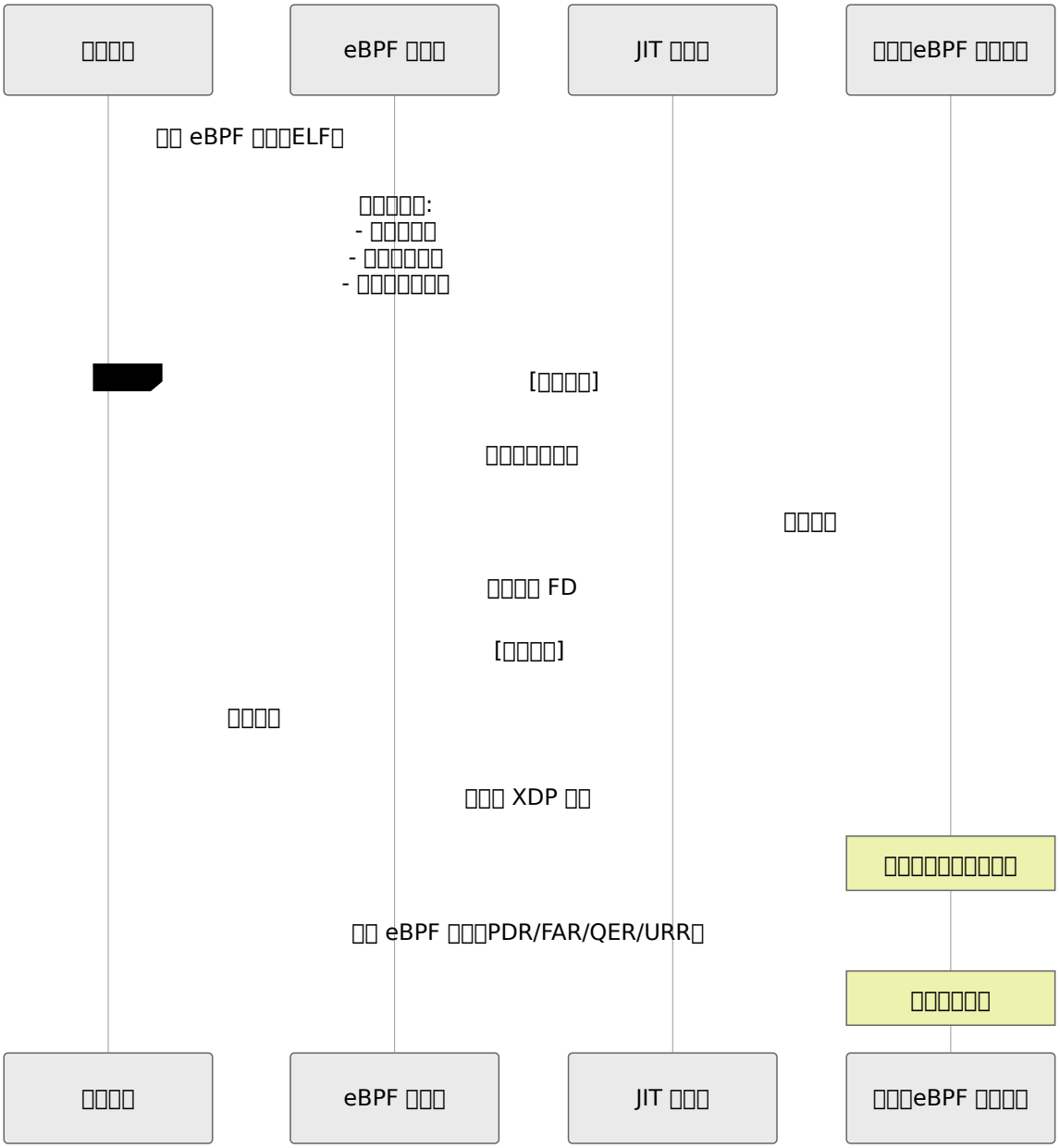
什么是 eBPF？

eBPF（扩展的伯克利数据包过滤器）是一项革命性的 Linux 内核技术，允许安全的沙箱程序在内核空间运行，而无需更改内核源代码或加载内核模块。

关键特性：

- 安全性：eBPF 验证器确保程序不会崩溃内核
- 性能：以本地内核速度运行（无解释开销）
- 灵活性：可以在运行时更新而无需重启内核
- 可观察性：内置跟踪和统计功能

eBPF □□□□□□



eBPF □□

eBPF □□□□□□□□□□□□ eBPF □□□□□□□□□□□□□□

OmniUPF □□□□□□□□□□

□□□□	□□	□□
`BPF_MAP_TYPE_HASH`	□□□□□□□□	□ TEID □ UE IP □□ PDR
`BPF_MAP_TYPE_ARRAY`	□□□□□□□□	□ ID □□ QER□FAR□URR
`BPF_MAP_TYPE_PERCPU_HASH`	□ CPU □□□□□□□□	□□□ PDR □□
`BPF_MAP_TYPE_LRU_HASH`	LRU□□□□□□□□□	□□□□□□□

□□□□□

- □□□O(1) □□□□□□□□
- □□□□□□□□□□□□□□
- □□□□□□□□□
- □□□□□□□□□□□□

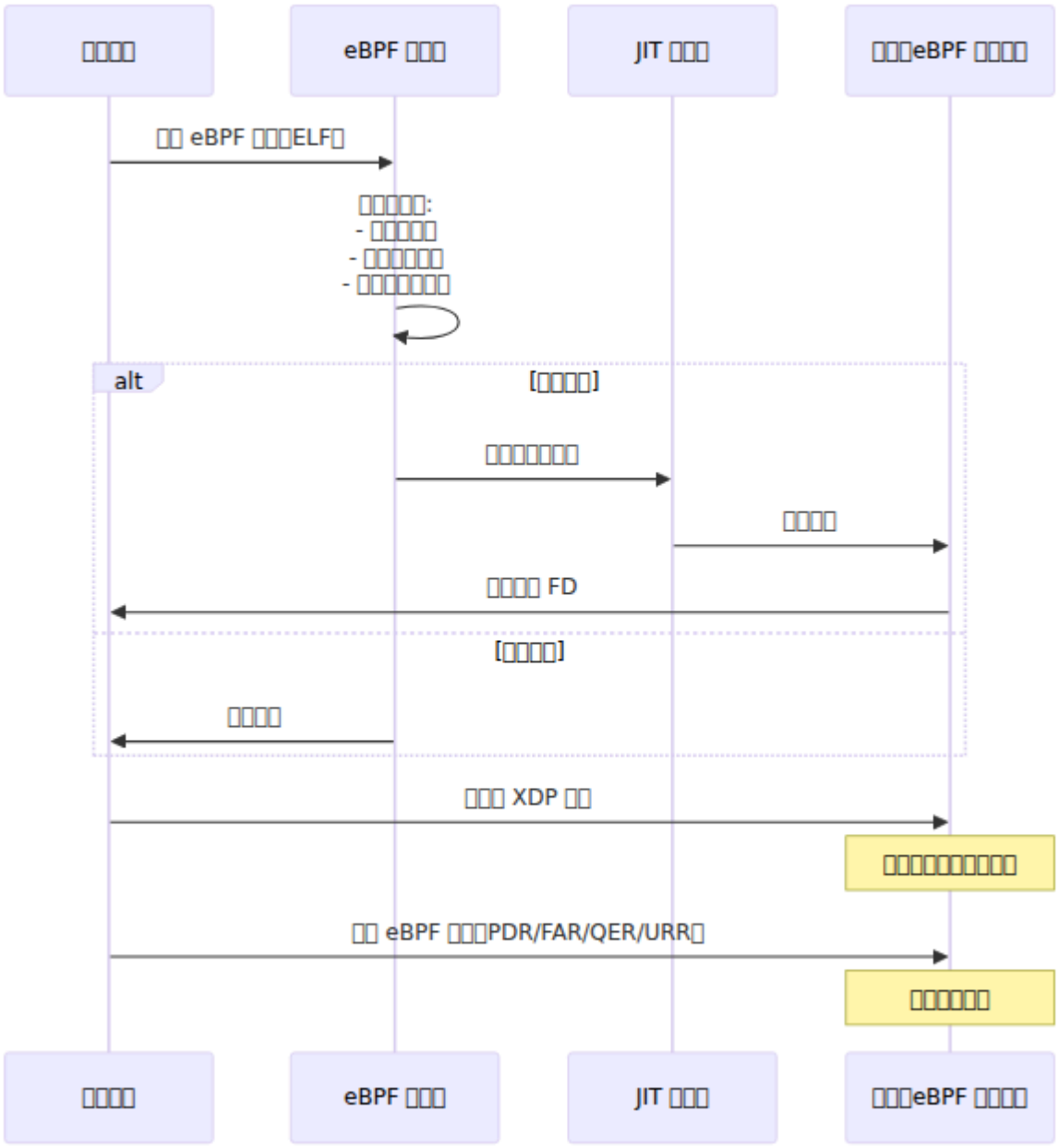
XDP □□□

XDP □□

XDP□□□□□□□□□□ Linux □□□□□□□ eBPF □□□□□□□□□□□□□□□—□□□□□□□□□□□□□□□□□□□□□□□□□

XDP □□□

OmniUPF □□□ XDP □□□□□□□□□□□□□□□□□□□□



1. XDP

- eBPF SmartNIC
- NIC CPU
- 100 Gbps+
- SmartNIC（Netronome、Mellanox ConnectX-6）

```
xdp_attach_mode: offload
```

□□□

- □□□□□ SmartNIC □□
- □□ eBPF □□□□□
- □□□□ eBPF □□□□□□□□□

---

2. XDP □□□□□□□□□□□□□□□□

□□□□□□□□□□□

- eBPF □□□□□□□□□□□□□□□□
- □□□□ SKB□□□□□□□□□□□□□□□
- □□□□□□ 10-40 Gbps
- □□□□ XDP □□□□□□□□□□□□□□□□

□□□

```
xdp_attach_mode: native
```

□□□

- □□□□□□□□□□□□□□
- □□□□□□□□□
- □□□ eBPF □□□

□□□□□□□□□

- □□□□i40e□ice□ixgbe□igb
- Mellanox□mlx4□mlx5
- □□□bnxt
- □□□□ena
- □□□ 10G+ □□□

---

3. XDP 通用模式

软件仿真（兼容性）：

- eBPF 程序在内核分配 SKB 后运行
- XDP 行为的软件仿真
- 在任何网络接口上工作
- 适用于测试和开发

配置:

```
xdp_attach_mode: generic
```

用例:

- 开发和测试
- 虚拟化环境（没有 SR-IOV 的 VM）
- 较旧的网络硬件
- 回环接口测试

性能: 1-5 Gbps（显著低于原生/卸载）

XDP 返回代码

eBPF 程序返回 XDP 动作代码，以告诉内核对数据包执行什么操作:

返回代码	意义	在 **OmniUPF** 中的使用
`XDP_PASS`	将数据包发送到内核网络栈	缓冲（本地交付）、ICMP、未知流量
`XDP_DROP`	立即丢弃数据包	无效数据包、速率限制、策略丢弃
`XDP_TX`	将数据包回传到相同接口	当前未使用
`XDP_REDIRECT`	将数据包发送到不同接口	主要转发路径（N3 ↔ N6）
`XDP_ABORTED`	处理错误，丢弃数据包并记录	eBPF 程序错误

数据包处理管道

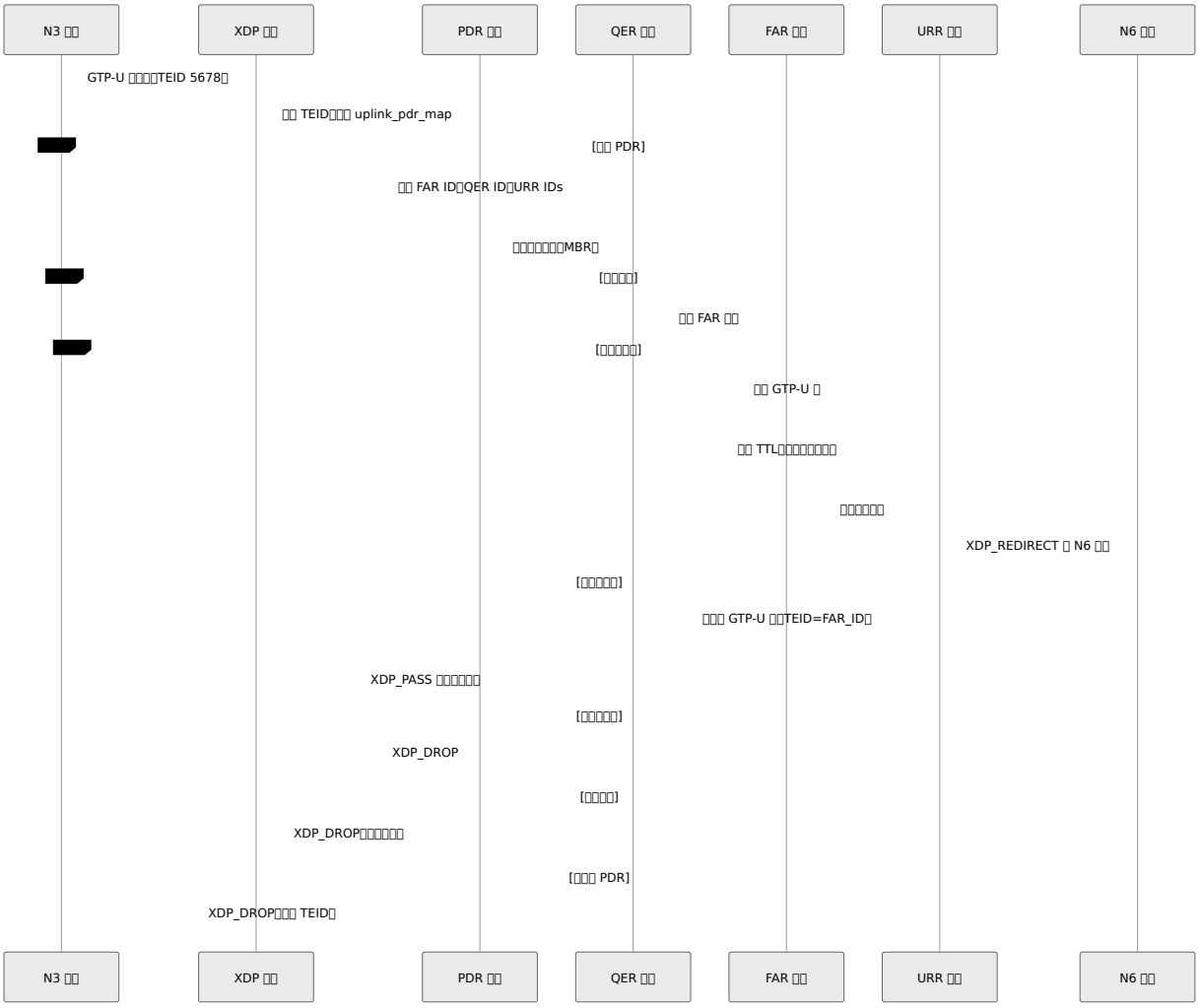
程序结构

OmniUPF 使用 eBPF 尾调用创建模块化的数据包处理管道。

尾调用：

- 允许 eBPF 程序调用其他 eBPF 程序
- 重用相同的栈帧（有界栈深度）
- 启用模块化管道设计
- 最大 33 次尾调用深度

□□□□□□□


N3 □□
XDP □□
PDR □□
QER □□
FAR □□
URR □□
N6 □□
GTP-U □□□□TEID 5678□
□□ TEID□□□ uplink_pdr_map
[□□ PDR]
□□ FAR ID□QER ID□URR IDs
□□□□□□□□MBR□
[□□□□□]
□□ FAR □□
[□□□□□□]
□□ GTP-U □
□□ TTL□□□□□□□□
□□□□□□□
XDP_REDIRECT □ N6 □□
[□□□□□□]
□□□ GTP-U □□TEID=FAR_ID□
XDP_PASS □□□□□□□
[□□□□□□]
XDP_DROP
[□□□□□]
XDP_DROP□□□□□□□
[□□□ PDR]
XDP_DROP□□□ TEID□
N3 □□
XDP □□
PDR □□
QER □□
FAR □□
URR □□
N6 □□

□□□□□□□


upf_entrypoint
□□□□□□
tail_call
pdr_lookup
□□ PDR
tail_call
qer_enforce
□□ QoS
tail_call
far_execute
□□
tail_call
urr_account
□□□□
XDP_REDIRECT
□□□□

eBPF 映射架构

映射内存布局

映射大小

OmniUPF 根据 `max_sessions` 配置自动计算映射大小:

```
PDR 映射 = 2 × max_sessions  (上行 + 下行)
FAR 映射 = 2 × max_sessions  (上行 + 下行)
QER 映射 = 1 × max_sessions  (每会话共享)
URR 映射 = 3 × max_sessions  (每会话多个 URR)
```

示例 (max_sessions = 65,535):

- PDR 映射: 每个 131,070 条目
- FAR 映射: 131,070 条目
- QER 映射: 65,535 条目
- URR 映射: 131,070 条目

总内存:

```
PDR 映射: 3 × 131,070 × 212 B = ~83 MB
FAR 映射:  131,070 × 20 B = ~2.6 MB
QER 映射:  65,535 × 36 B = ~2.3 MB
URR 映射:  131,070 × 20 B = ~2.6 MB
总计: ~91 MB 内核内存
```

缓冲机制

缓冲概述

OmniUPF 实现数据包缓冲以应对切换场景，通过在 GTP-U 中封装数据包并通过 UDP 套接字发送到用户空间进程。

缓冲架构

Parse error on line 10: ...冲存储
映射: FAR_ID → []数据包] end ----------------------^
Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND', 'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND', 'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'SQS'

试

缓冲封装细节

当缓冲启用时（FAR 动作位 2 设置），eBPF 程序:

1. 计算原始数据包大小:

```
orig_packet_len = ntohs(ip->tot_len);  // IP 头部获取
```

2. 扩展数据包头:

```
// 为: 外部 IP + UDP + GTP-U 添加空间
gtp_encap_size = sizeof(struct iphdr) + sizeof(struct udphdr) +
sizeof(struct gtpuhdr);
bpf_xdp_adjust_head(ctx, -gtp_encap_size);
```

3. 构建外部 **IP** 头:

```
ip->saddr = original_sender_ip;  // □□□□□□□□□□□□□□□
ip->daddr = local_upf_ip;        // □□□□□□□□□□□□□□□ IP
ip->protocol = IPPROTO_UDP;
ip->ttl = 64;
```

4. □□ **UDP □□**

```
udp->source = htons(22152);  // BUFFER_UDP_PORT
udp->dest = htons(22152);
udp->len = htons(sizeof(udphdr) + sizeof(gtpuhdr) +
orig_packet_len);
```

5. □□ **GTP-U □□**

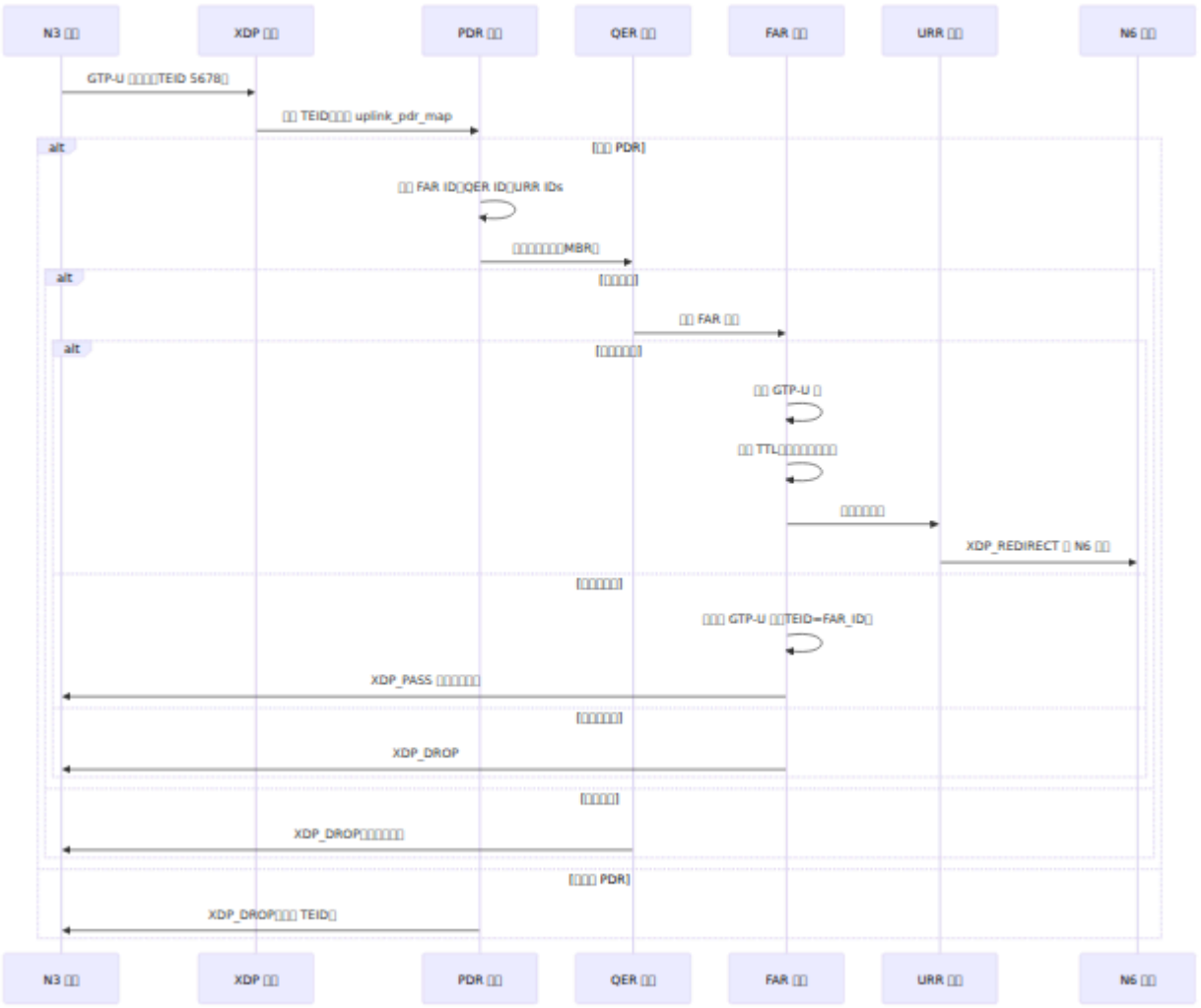
```
gtp->version = 1;
gtp->message_type = GTPU_G_PDU;
gtp->teid = htonl(far_id | (direction << 24));  // □□ FAR ID □□
□
gtp->message_length = htons(orig_packet_len);
```

6. □□ **XDP_PASS□**

   - □□□□□□□□□□□□ UDP □□□□□ 22152
   - □□□□□□□□□□□□□□□□□□□□

□□□□□□

□□□□□□□□SMF □□ FAR □□□ BUFFER □□□□□□□□□□□□□□□



缓冲管理参数

参数	默认值	描述
每 **FAR** 最大	10,000 数据包	每个 FAR 缓冲的最大数据包数
总最大	100,000 数据包	最大缓冲的数据包总数
数据包 **TTL**	30 秒	缓冲数据包过期前的时间
缓冲端口	22152	用于缓冲交付的 UDP 端口
缓冲清理间隔	60 秒	检查过期数据包的频率

QoS 执行

速率限制算法

OmniUPF 实现了 滑动窗口速率限制器 来执行 QoS。

Parse error on line 5: ...= packet_size × 8 × (NSEC_PER_SEC / rate ----------^ Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND', 'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND', 'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'PS'

□试

滑动窗口实现

算法（来自 `qer.h`）：

```
static __always_inline enum xdp_action limit_rate_sliding_window(
    const __u64 packet_size,
    volatile __u64 *window_start,
    const __u64 rate)
{
    static const __u64 NSEC_PER_SEC = 1000000000ULL;
    static const __u64 window_size = 5000000ULL;  // 5ms 窗口

    // 速率 = 0 表示无限制
    if (rate == 0)
        return XDP_PASS;

    // 计算此数据包的传输时间
    __u64 tx_time = packet_size * 8 * (NSEC_PER_SEC / rate);
    __u64 now = bpf_ktime_get_ns();

    // 检查我们是否在窗口前面（数据包将在未来传输）
    __u64 start = *window_start;
    if (start + tx_time > now)
        return XDP_DROP;  // 超过速率限制

    // 如果窗口已过去，重置它
    if (start + window_size < now) {
        *window_start = now - window_size + tx_time;
        return XDP_PASS;
    }

    // 更新窗口以考虑此数据包
    *window_start = start + tx_time;
    return XDP_PASS;
}
```

关键参数：

- 窗口大小：5ms（5,000,000 纳秒）
- 每个方向：上行和下行的独立窗口
- 原子更新：使用易失性指针进行并发访问
- **MBR = 0**：视为无限带宽

QoS 示例计算

场景: MBR = 100 Mbps, 数据包大小 = 1500 字节

1. 传输时间:

```
tx_time = 1500 字节 × 8 位/字节 × (1,000,000,000 ns/sec ÷
100,000,000 bps)
tx_time = 1500 × 8 × 10 = 120,000 ns = 120 µs
```

2. 速率检查:

   - 如果最后一个数据包在 `t=0` 传输，下一数据包可以在 `t=120µs` 传输
   - 如果数据包在 `t=100µs` 到达，则丢弃（太早）
   - 如果数据包在 `t=150µs` 到达，则转发（窗口已推进）

3. 最大数据包速率:

```
Max PPS = (100 Mbps ÷ 8) ÷ 1500 字节 = 8,333 数据包/秒
数据包间隔 = 120 µs
```

性能特征

吞吐量

配置	吞吐量	数据包/秒	延迟
XDP 卸载 (SmartNIC)	100 Gbps	148 Mpps	< 1 µs
XDP 原生 (10G NIC, 单核)	10 Gbps	8 Mpps	2-5 µs
XDP 原生 (10G NIC, 4 核)	40 Gbps	32 Mpps	2-5 µs
XDP 通用	1-5 Gbps	0.8-4 Mpps	50-100 µs

延迟分解

总数据包处理延迟（XDP 原生）：

阶段	延迟	累计
NIC RX	0.5 µs	0.5 µs
XDP 钩子调用	0.1 µs	0.6 µs
PDR 查找（哈希）	0.3 µs	0.9 µs
QER 速率检查	0.1 µs	1.0 µs
FAR 处理	0.5 µs	1.5 µs
URR 更新	0.2 µs	1.7 µs
GTP-U 封装/解封装	0.8 µs	2.5 µs
XDP_REDIRECT	0.5 µs	3.0 µs
NIC TX	0.5 µs	3.5 µs

总计：每个数据包约 ~3.5 µs（XDP 原生，10G NIC）

CPU 利用率

每核处理能力：

- 单核：8-10 Mpps（XDP 原生）
- 启用超线程：12-15 Mpps
- 多核扩展：接近线性，最多 8 核

数据包速率的 **CPU** 使用率：

```
CPU % ≈ (数据包速率 / 10,000,000) × 100% 每核
```

示例: 2 Mpps 流量使用 ~20% 的一个核心

内存带宽

eBPF 映射访问:

- 哈希查找: ~100 ns (缓存命中)
- 哈希查找: ~300 ns (缓存未命中)
- 数组查找: ~50 ns (始终缓存命中)

所需内存带宽:

```
带宽 = 数据包速率 × (平均数据包大小 + 映射查找 × 64 字节)
```

示例: 10 Mpps × (1500 B + 3 次查找 × 64 B) ≈ 160 Gbps 内存带宽

可扩展性与调优

水平扩展

多个 **UPF** 实例:

Setting SMF as parent of SMF would create a cycle

□试

会话分配:

- SMF 在 UPF 实例之间分配会话
- 每个 UPF 处理一部分 UE 会话
- 不需要 UPF 之间的通信 (无状态)

垂直扩展

CPU 调优：

1. 启用 XDP 处理的 CPU 亲和性
2. 使用 RSS（接收侧扩展）分配 RX 队列
3. 将 eBPF 程序固定到特定核心

NIC 调优：

1. 增加 RX 环形缓冲区大小
2. 启用多队列 NIC（RSS）
3. 使用流导向进行流量引导

内核调优：

```
# 增加 eBPF 映射的锁定内存限制
ulimit -l unlimited

# 禁用 IRQ 平衡以适应 XDP 核心
systemctl stop irqbalance

# 将 CPU 调节器设置为性能
cpupower frequency-set -g performance

# 增加网络缓冲区大小
sysctl -w net.core.rmem_max=134217728
sysctl -w net.core.wmem_max=134217728
```

容量规划

公式：

```
所需 CPU 核心 = (预期 PPS ÷ 10,000,000) × 1.5  (50% 余量)
所需内存 = (最大会话 × 212 B × 3) + 100 MB (eBPF 映射 + 开销)
所需网络 = (峰值吞吐量 × 2) + 10 Gbps (余量)
```

示例（100 万会话，20 Gbps 峰值）：

- CPU: (20 Gbps ÷ 每核 10 Gbps) × 1.5 = 3-4 核
- 内存: (1M × 212 B × 3) + 100 MB ≈ 750 MB
- 网络: (20 Gbps × 2) + 10 Gbps = 50 Gbps 接口

相关档

- **UPF** 操作指南 - 一般 UPF 操作和部署
- 规则管理指南 - PDR、FAR、QER、URR 详细信息
- 监控指南 - 性能监控和指标
- **Web UI** 操作指南 - 控制面板使用
- 故障排除指南 - 常见问题和诊断

OmniUPF 配置指南

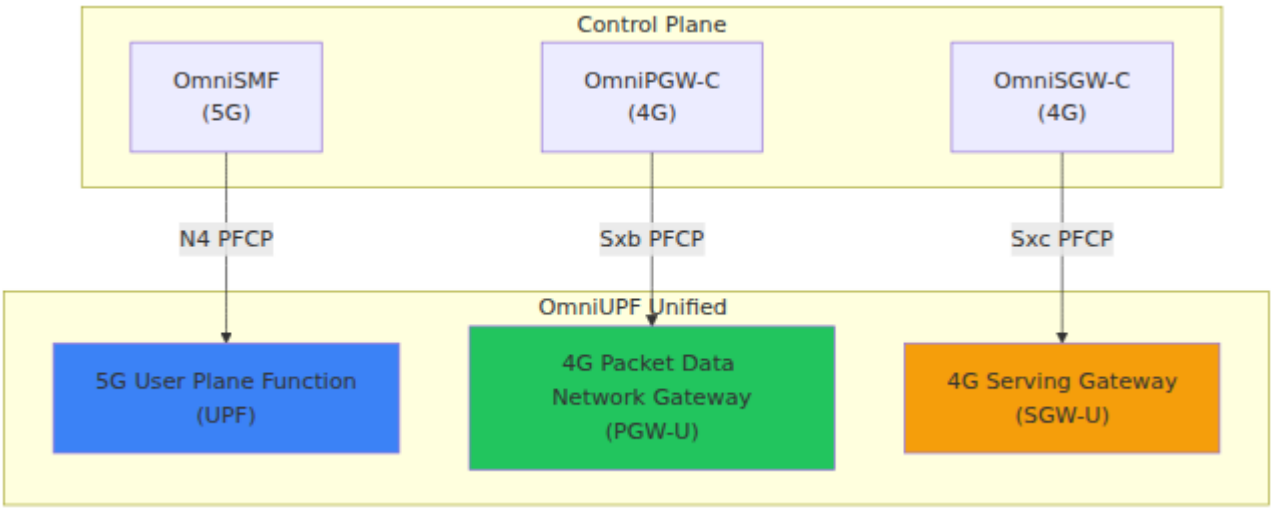
目录

概述

OmniUPF 是一个多功能的用户平面功能，可以在多种模式下运行，以支持 4G (EPC) 和 5G 核心网络。配置通过 YAML 配置文件进行管理。

操作模式

OmniUPF 是一个统一平台，可以同时作为：



模式配置

操作模式由控制平面 (SMF, PGW-C 或 SGW-C) 决定，该控制平面与 OmniUPF 建立 PFCP 关联。切换模式不需要特定的 OmniUPF 配置。

同时操作：

- OmniUPF 可以同时接受来自多个控制平面的 PFCP 关联
- 单个 OmniUPF 实例可以同时充当 UPF、PGW-U 和 SGW-U
- 来自不同控制平面的会话是隔离的，并独立管理

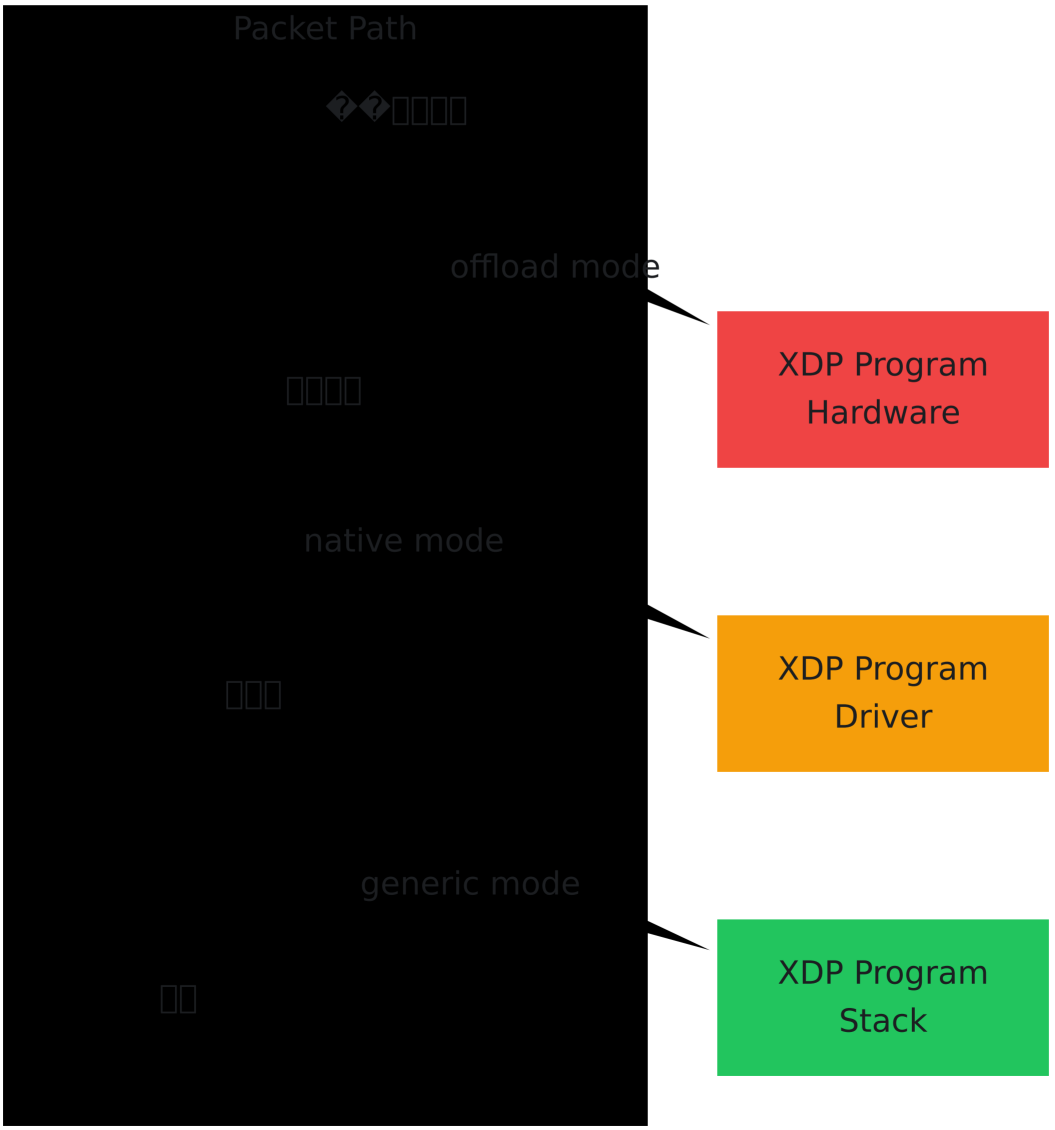
XDP 附加模式

OmniUPF 使用 XDP (eXpress Data Path) 进行高性能数据包处理。支持三种附加模式。

有关详细的 **XDP** 设置说明，特别是针对 **Proxmox** 和其他虚拟机监控程序，请参见 **XDP** 模式指南。

模式比较

模式	附加点	性能	用例	**NIC** 要求
通用	网络栈（内核）	~1-2 Mpps	测试、开发、兼容性	任何 NIC
本地	网络驱动（内核）	~5-10 Mpps	生产（裸金属，带 SR-IOV 的 VM）	支持 XDP 的驱动
卸载	NIC 硬件 (SmartNIC)	~10-40 Mpps	高吞吐量生产	支持 XDP 卸载的 SmartNIC



通用模式（默认）

描述：XDP 程序在内核网络栈中运行

优点：

- 适用于 任何 网络接口
- 无需特殊驱动或硬件要求
- 理想的测试和开发环境
- 与所有虚拟机监控程序和虚拟化平台兼容

缺点：

- 性能较低（~1-2 Mpps 每个核心）
- 数据包在 XDP 处理之前已经通过驱动

配置：

```
xdp_attach_mode: generic
```

最佳适用：

- 不支持 SR-IOV 的虚拟机
- 测试和验证环境
- 不支持 XDP 驱动的 NIC
- 如 Proxmox、VMware、VirtualBox 等虚拟机监控程序

本地模式（推荐）

描述：XDP 程序在网络驱动级别运行

优点：

- 高性能（~5-10 Mpps 每个核心）
- 数据包在进入网络栈之前被处理
- 延迟显著低于通用模式

- 适用于裸金属和 SR-IOV 虚拟机

缺点：

- 需要支持 XDP 的网络驱动
- 并非所有 NIC/驱动都支持本地 XDP

配置：

```
xdp_attach_mode: native
```

最佳适用：

- 裸金属上的生产部署
- 带 SR-IOV 直通的虚拟机
- 具有 XDP 能力驱动的 NIC（如 Intel、Mellanox 等）

要求：

- 支持 XDP 的网络驱动（见 NIC 兼容性）
- Linux 内核 5.15+ 并启用 XDP 支持

卸载模式（最大性能）

描述：XDP 程序直接在 SmartNIC 硬件上运行

优点：

- 最大性能（~10-40 Mpps）
- 数据包处理的 CPU 开销为零
- 亚微秒延迟
- 释放 CPU 用于控制平面处理

缺点：

- 需要昂贵的 SmartNIC 硬件
- SmartNIC 可用性有限

- 设置和配置复杂

配置：

```
xdp_attach_mode: offload
```

最佳适用：

- 超高吞吐量的生产部署
- 对延迟要求严格的边缘计算
- CPU 资源有限的环境

要求：

- 支持 XDP 卸载的 SmartNIC (如 Netronome Agilio CX、Mellanox BlueField)
- 专用固件和驱动

配置参数

网络接口

参数	描述	类型	默认
`interface_name`	N3/N6/N9 流量的网络接口 (XDP 附加点)	列表	`[lo]`
`n3_address`	N3 接口的 IPv4 地址 (来自 RAN 的 GTP-U)	IP	`127.0.0.1`
`n9_address`	N9 接口的 IPv4 地址 (UPF 到 UPF 的 ULCL)	IP	与 `n3_address` 相同

示例：

```yaml
interface_name: [eth0, eth1]
n3_address: 10.100.50.233
n9_address: 10.100.50.234
```

PFCP 配置

参数	描述	类型	默认
`pfcp_address`	PFCP 服务器的本地地址 (N4/Sxb/Sxc 接口)	主机:端口	`:8805`
`pfcp_node_id`	PFCP 协议的本地节点 ID	IP	`127.0.0.1`
`pfcp_remote_node`	要连接的远程 PFCP 对等方 (SMF/PGW-C/SGW-C)	列表	`[]`
`association_setup_timeout`	关联设置请求之间的超时时间（秒）	整数	`5`
`heartbeat_retries`	在声明对等方死亡之前的心跳重试次数	整数	`3`
`heartbeat_interval`	PFCP 心跳间隔（秒）	整数	`5`
`heartbeat_timeout`	PFCP 心跳超时（秒）	整数	`5`

示例:

```yaml
pfcp_address: :8805
pfcp_node_id: 10.100.50.241
pfcp_remote_node:
  - 10.100.50.10  # OmniSMF
  - 10.100.60.20  # OmniPGW-C
heartbeat_interval: 10
heartbeat_retries: 5
```

API 和监控

参数	描述	类型	默认
`api_address`	REST API 服务器的本地地址	主机:端口	`:8080`
`metrics_address`	Prometheus 指标端点的地址（见 指标参考）	主机:端口	`:9090`
`logging_level`	日志级别（`trace`、`debug`、`info`、`warn`、`error`）	字符串	`info`

示例：

```yaml
api_address: :8080
metrics_address: :9090
logging_level: debug
```

GTP 路径管理

参数	描述	类型	默认
`gtp_peer`	GTP 对等方的列表，用于回声请求保持活动	列表	`[]`
`gtp_echo_interval`	GTP 回声请求之间的间隔（秒）	整数	`10`

示例：

```yaml
gtp_peer:
  - 10.100.50.50:2152  # gNB
  - 10.100.50.60:2152  # 另一个 UPF 用于 N9
gtp_echo_interval: 15
```

---

eBPF □□□□

□□	□□	□□	□□	□□□□
`max_sessions`	□□□□□□□□	□□	`65535`	□□□□□□□□□
`pdr_map_size`	PDR eBPF □□□□□	□□	`0`	`max_sessions × 2`
`far_map_size`	FAR eBPF □□□□□	□□	`0`	`max_sessions × 2`
`qer_map_size`	QER eBPF □□□□□	□□	`0`	`max_sessions`
`urr_map_size`	URR eBPF □□□□□	□□	`0`	`max_sessions × 2`

□□□□□□□□□□□□ `0` (□□) □□□ `max_sessions` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□

```
max_sessions: 100000
# □□□□□□□□□□□□□□□
# PDR: 200,000 □□
# FAR: 200,000 □□
# QER: 100,000 □□
# URR: 200,000 □□
```

□□□□□□□□□

```
max_sessions: 50000
pdr_map_size: 131070  # □□□□□
far_map_size: 131070
qer_map_size: 65535
urr_map_size: 131070
```

缓冲区配置

参数	描述	类型	默认
`buffer_port`	用于来自 eBPF 的缓冲数据包的 UDP 端口	整数	`22152`
`buffer_max_packets`	每个 FAR 的最大缓冲数据包数	整数	`10000`
`buffer_max_total`	最大总缓冲数据包数（0=无限制）	整数	`100000`
`buffer_packet_ttl`	缓冲数据包的 TTL（秒）（0=无过期）	整数	`30`
`buffer_cleanup_interval`	缓冲清理间隔（秒）（0=不清理）	整数	`60`

示例：

```yaml
buffer_port: 22152
buffer_max_packets: 20000
buffer_max_total: 200000
buffer_packet_ttl: 60
buffer_cleanup_interval: 30
```

功能标志

参数	描述	类型	默认
`feature_ueip`	通过 OmniUPF 启用 UE IP 分配	布尔	`false`
`ueip_pool`	UE IP 分配的 IP 池（需要 `feature_ueip`）	CIDR	`10.60.0.0/24`
`feature_ftup`	通过 OmniUPF 启用 F-TEID 分配	布尔	`false`
`teid_pool`	F-TEID 分配的 TEID 池大小（需要 `feature_ftup`）	整数	`65535`

示例 **(UE IP 分配)**:

```
feature_ueip: true
ueip_pool: 10.45.0.0/16  # □此池中分配 UE IP
```

示例 **(F-TEID 分配)**:

```
feature_ftup: true
teid_pool: 1000000  # 允许最多 1M TEID 分配
```

路由管理器配置

用于与 FRR (Free Range Routing) 守护进程同步 UE 路由。有关详细信息，请参见 路由管理指南。

参数	描述	类型	默认
`route_manager_enabled`	启用自动 UE 路由同步	布尔	`false`
`route_manager_type`	路由守护进程类型（`frr` 支持）	字符串	`frr`
`route_manager_vtysh_path`	vtysh 命令的路径	字符串	`/usr/bin/vtysh`
`route_manager_nexthop`	UE 路由的下一跳 IP	IP 地址	`` (空)

示例:

```
route_manager_enabled: true
route_manager_type: frr
route_manager_vtysh_path: /usr/bin/vtysh
route_manager_nexthop: 10.0.1.1  # UE 路由的下一跳
```

何时启用:

- 需要路由广告的多 UPF 部署
- 与 OSPF 或 BGP 路由协议集成
- 需要安装和配置 FRRouting 守护进程

配置方法

YAML 配置□件（推荐）

□件：`config.yml`

```
# 网络配置
interface_name: [eth0]
n3_address: 10.100.50.233
n9_address: 10.100.50.233
xdp_attach_mode: native

# PFCP 配置
pfcp_address: :8805
pfcp_node_id: 10.100.50.241
pfcp_remote_node:
  - 10.100.50.10

# API 和监控
api_address: :8080
metrics_address: :9090
logging_level: info

# 容量
max_sessions: 100000

# GTP 对等方
gtp_peer:
  - 10.100.50.50:2152
gtp_echo_interval: 10

# 功能
feature_ueip: true
ueip_pool: 10.45.0.0/16
feature_ftup: false

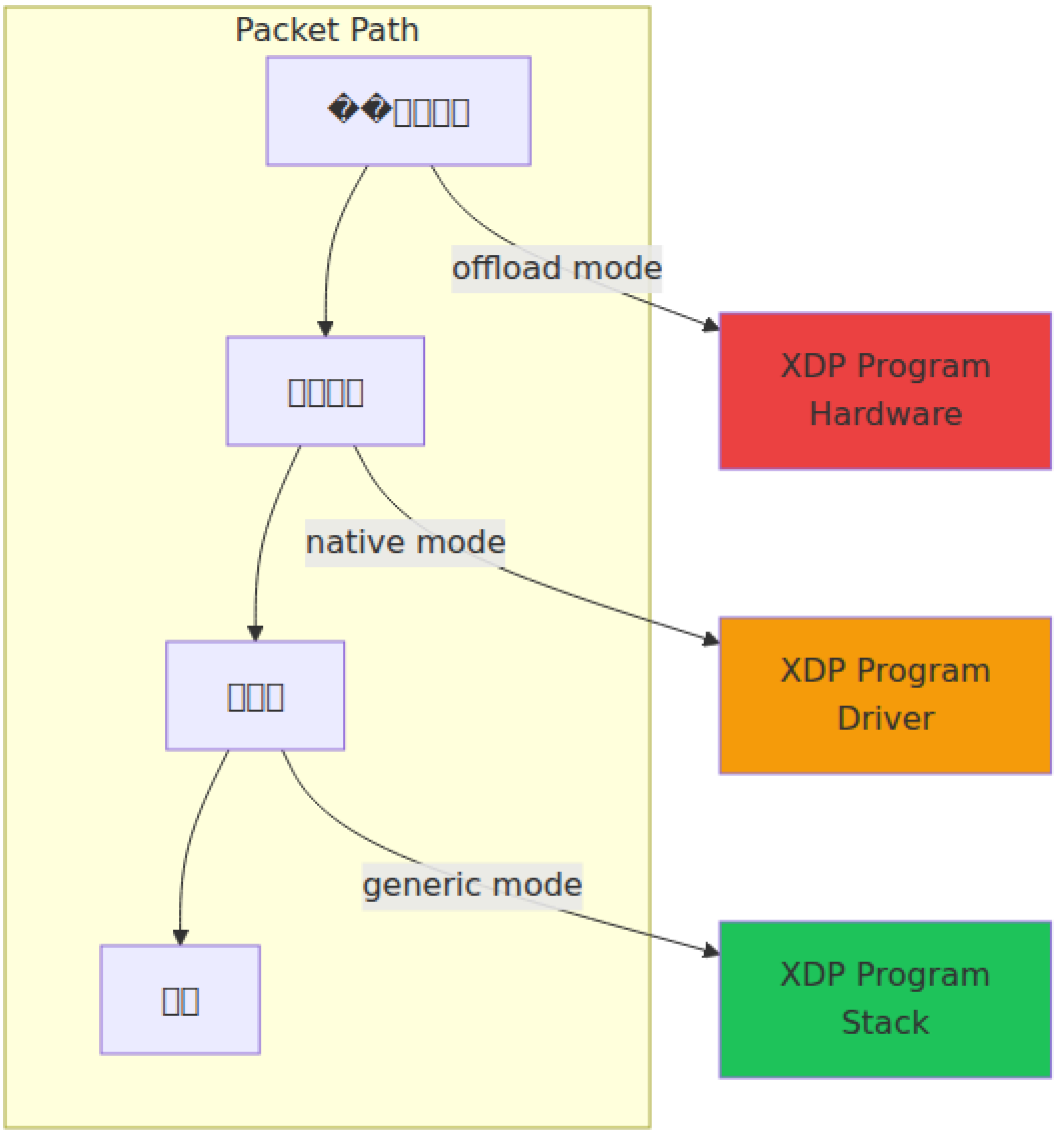
# 缓冲
buffer_max_packets: 15000
buffer_packet_ttl: 45
```

虚拟机监控程序兼容性

概述

OmniUPF 与所有主要虚拟机监控程序和虚拟化平台兼容。XDP 附加模式和网络配置取决于虚拟机监控程序的网络能力。

有关在 **Proxmox** 和其他虚拟机监控程序上启用本地 **XDP** 的逐步说明，请参见 **XDP** 模式指南。



Proxmox VE

支持的配置：

1. 桥接模式（通用 XDP）

用例：标准 VM 网络

配置：

- 网络设备：VirtIO 或 E1000
- XDP 模式：`generic`

- 性能: ~1-2 Mpps

Proxmox VM 设置:

```
网络设备: net0
模型: VirtIO (虚拟化)
桥接: vmbr0
```

OmniUPF 配置:

```
interface_name: [eth0]
xdp_attach_mode: generic
```

---

2. SR-IOV 直通 (本地 XDP)

用例: 高性能生产

配置:

- 网络设备: SR-IOV 虚拟功能
- XDP 模式: `native`
- 性能: ~5-10 Mpps

要求:

- 支持 SR-IOV 的物理 NIC (如 Intel X710、Mellanox ConnectX-5)
- 在 BIOS 中启用 SR-IOV
- 启用 IOMMU (`intel_iommu=on` 或 `amd_iommu=on` 在 GRUB 中)

在 **Proxmox** 上启用 **SR-IOV**:

```
# 编辑 GRUB 配置
nano /etc/default/grub

# 添加到 GRUB_CMDLINE_LINUX_DEFAULT:
intel_iommu=on iommu=pt

# 更新 GRUB 并重启
update-grub
reboot

# 在 NIC 上启用 VFs (示例: 在 eth0 上 4 个虚拟功能)
echo 4 > /sys/class/net/eth0/device/sriov_numvfs

# 使其持久化
echo "echo 4 > /sys/class/net/eth0/device/sriov_numvfs" >>
/etc/rc.local
chmod +x /etc/rc.local
```

Proxmox VM 设置:

```
硬件 → 添加 → PCI 设备
选择: SR-IOV 虚拟功能
所有功能: 否
主 GPU: 否
PCI-Express: 是 (可选)
```

OmniUPF 配置:

```
interface_name: [ens1f0]  # SR-IOV VF 名称
xdp_attach_mode: native
```

---

3. PCI 直通 (本地 XDP)

用例: 为单个 VM 供专用 NIC

配置:

- 整个物理 NIC 传递给 VM

- XDP 模式: `native` 或 `offload` (如果是 SmartNIC)
- 性能: ~5-40 Mpps (取决于 NIC)

Proxmox VM 设置:

```
硬件 → 添加 → PCI 设备
选择: 物理 NIC (例如, 0000:01:00.0)
所有功能: 是
主 GPU: 否
PCI-Express: 是
```

OmniUPF 配置:

```yaml
interface_name: [ens1f0]
xdp_attach_mode: native  # 或 'offload' 对于 SmartNIC
```

KVM/QEMU

桥接模式:

```
virt-install \
  --name omniupf \
  --network bridge=br0,model=virtio \
  --disk path=/var/lib/libvirt/images/omniupf.qcow2 \
  ...
```

SR-IOV 直通:

```
<interface type='hostdev' managed='yes'>
  <source>
    <address type='pci' domain='0x0000' bus='0x01' slot='0x10'
function='0x1'/>
  </source>
</interface>
```

VMware ESXi

□□ vSwitch (□□ XDP)□

- □��□□□□VMXNET3
- XDP □□□`generic`

SR-IOV (□□ XDP)□

- □ ESXi □□□□□□□ SR-IOV
- □ SR-IOV □□□□□□□□ VM
- XDP □□□`native`

Microsoft Hyper-V

□□□□□ (□□ XDP)□

- □□□□□□□□□
- XDP □□□`generic`

SR-IOV (□□ XDP)□

- □ Hyper-V □□□□□□ SR-IOV
- □□□□□□□□□□□□ SR-IOV
- XDP □□□`native`

VirtualBox

NAT/□□□□ (□□□ XDP)□

- □□□□□□VirtIO-Net □ Intel PRO/1000
- XDP □□□`generic`
- □□□VirtualBox □□□ SR-IOV

NIC □□□

□□ Mpps □□□□

□□□□□□ **(Mpps)** □□□□ **(Gbps)** □□□□□□ - □□□□□□□

以 **1 Mpps** 的吞吐量计算最大 **GTP-U** 数据包:

```
1536 字节 × 1,000,000 pps × 8 位/字节 = 12,288 Mbps ≈ 12.3 Gbps
```

原生 **IP** 数据包 **(N6** 接口**)**:

在 N6 (朝向互联网) 上，数据包是原生 IP，没有 GTP-U:

最小 **N6** 数据包:

- **IP** 头: 20 字节
- **UDP** 头: 8 字节
- 最小有效载荷: 1 字节
- 总计: 29 字节

以 **1 Mpps** 的吞吐量计算最小 **N6** 数据包:

```
29 字节 × 1,000,000 pps × 8 位/字节 = 232 Mbps
```

最大 **N6** 数据包 **(1500 MTU)**:

- **IP MTU**: 1500 字节
- 总计: 1500 字节

以 **1 Mpps** 的吞吐量计算最大 **N6** 数据包:

```
1500 字节 × 1,000,000 pps × 8 位/字节 = 12,000 Mbps = 12 Gbps
```

实际性能示例

Intel X710 NIC (N3 接口的 **10 Mpps** 容量**)**:

流量模式	内部数据包大小	**GTP-U** 总计	**10 Mpps** 吞吐量	典型用例
VoIP 通话 (N3)	65-150 字节	101-186 字节	**0.8-1.5 Gbps**	AMR-WB 语音, G.711
轻量级网页 (N3)	400-600 字节	436-636 字节	**3.5-5.1 Gbps**	HTTP/HTTPS, 消息
现代移动 **(N3)**	**1200** 字节	**1236** 字节	**9.9 Gbps**	典型 **2024** 流量组合
视频流 (N3)	1400-1450 字节	1436-1486 字节	**11.5-11.9 Gbps**	HD/4K 视频块
最大 MTU (N3)	1500 字节	1536 字节	**12.3 Gbps**	大型 TCP 下载

在 **N6** 接口 **(**原生 **IP**, 无 **GTP-U)**:

流量模式	数据包大小	**10 Mpps** 吞吐量	典型用例
VoIP 数据包	65-150 字节	**0.5-1.2 Gbps**	语音 RTP 流
轻量级网页	400-600 字节	**3.2-4.8 Gbps**	HTTP 请求
现代移动	**1200** 字节	**9.6 Gbps**	典型 **2024** 流量
视频流	1400-1450 字节	**11.2-11.6 Gbps**	视频下载
最大 MTU	1500 字节	**12.0 Gbps**	大文件传输

以 **10 Mpps** 的现代移动流量 **(1200** 字节平均**)**, 预计 **~10 Gbps** 吞吐量在 N3 和 N6 接口上。

这对移动网络的重要性:

移动流量在数据包大小上变化很大, 而 GTP-U 开销 (36 字节) 对小数据包性能的影响显著:

□□□□□□□□ **(□□□□□□□)**□

- **VoIP (AMR-WB □□□□)**□65-80 □□ → □□ GTP-U: 101-116 □□
- □□□□□□□□□□50-200 □□ → □□ GTP-U: 86-236 □□
- □□□□ **(HTTP/3)**□400-800 □□ → □□ GTP-U: 436-836 □□
- □□□□1200-1450 □□ → □□ GTP-U: 1236-1486 □□
- □□□□□1500 □□ → □□ GTP-U: 1536 □□

GTP-U □□□□□□

- □□□□ (< 200 □□)□**~35-70%** □□ - Mpps □□□□□
- □□□□□ (200-800 □□)□**~5-20%** □□ - □□□□
- □□□□ (> 1200 □□)□**~3%** □□ - □□□□□□□□□

□□□□□

□□□ **10 Mpps** □ NIC □ N3 □□□□□□□

- **VoIP** □□□□ (100 □□□□□□□□)□~1.0 Gbps (GTP-U □□□□)
- □□□□□□ (1200 □□□□□□□□□□)□~9.9 Gbps
- □□□□□□ (1400 □□□□□□□□)□~11.5 Gbps
- □□□□□ (1500 □□□□□□□□)□~12.3 Gbps

□ **N6** □□ (□ GTP-U □□)□

- □□□□□□ (1200 □□□□□)□~9.6 Gbps □ 10 Mpps
- □□□□□ (1500 □□□□□)□~12.0 Gbps □ 10 Mpps

□□□□ UPF □□□□□□

- □□□□□□ (VoIP□□□□□□□)□Mpps □□□ - □□□ 10 Mpps □ 1-2 Gbps
- □□□□□□ (1200 □□□□)□□□□ 10 Mpps □□□ 9-10 Gbps
- □□□□□□ (□□□□□□)□□□□ 10 Mpps □□□ 10-12 Gbps
- □□□□ **N3** □ **N6** - N3 □ GTP-U □□□N6 □□

□□□□□□□

□ 1200 □□□□□□□□□ (□□□□□□□□□□□□□)□

NIC Mpps 容量	N3 吞吐量 (GTP-U)	N6 吞吐量 (原生 IP)	现实部署
1 Mpps	~1.0 Gbps	~1.0 Gbps	小型基站，物联网网关
5 Mpps	~4.9 Gbps	~4.8 Gbps	中型基站，企业
10 Mpps	~9.9 Gbps	~9.6 Gbps	大型基站，小城市
20 Mpps	~19.7 Gbps	~19.2 Gbps	大都市，中型城市
40 Mpps	~39.4 Gbps	~38.4 Gbps	大型都市，区域中心

注意：这些估算假设 1200 字节平均有效载荷，这代表了现代移动流量，以视频流、社交媒体和云应用为主。实际吞吐量将根据流量组合变化。

XDP 能力网络驱动

OmniUPF 需要支持 XDP 的网络驱动以实现 本地 和 卸载 模式。通用模式适用于 任何 NIC。

Intel NICs

型号	驱动	XDP 支持	模式	性能
Intel X710	i40e	是	本地	~10 Mpps
Intel XL710	i40e	是	本地	~10 Mpps
Intel E810	ice	是	本地	~15 Mpps
Intel 82599ES	ixgbe	是	本地	~8 Mpps
Intel I350	igb	有限	通用	~1 Mpps
Intel E1000	e1000	否	仅通用	~1 Mpps

Mellanox/NVIDIA NICs

型号	驱动	**XDP** 支持	模式	性能
Mellanox ConnectX-5	mlx5	是	本地	~12 Mpps
Mellanox ConnectX-6	mlx5	是	本地	~20 Mpps
Mellanox BlueField	mlx5	是	本地 + 卸载	~40 Mpps
Mellanox ConnectX-4	mlx4	有限	通用	~2 Mpps

Broadcom NICs

型号	驱动	**XDP** 支持	模式	性能
Broadcom BCM57xxx	bnxt_en	是	本地	~8 Mpps
Broadcom NetXtreme II	bnx2x	否	仅通用	~1 Mpps

其他厂商

型号	驱动	**XDP** 支持	模式	性能
Netronome Agilio CX	nfp	是	卸载	~30 Mpps
Amazon ENA	ena	是	本地	~5 Mpps
Solarflare SFC9xxx	sfc	是	本地	~8 Mpps
VirtIO	virtio_net	有限	通用	~2 Mpps

检查 **NIC XDP** 支持

检查驱动是否支持 **XDP**:

```
# 查找 NIC 驱动
ethtool -i eth0 | grep driver

# 检查驱动中的 XDP 支持
modinfo <driver_name> | grep -i xdp

# 例如 Intel i40e
modinfo i40e | grep -i xdp
```

验证 **XDP** 程序附加:

```
# 检查 XDP 程序是否附加
ip link show eth0 | grep -i xdp

# 示例输出 (XDP 附加):
# 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 xdp qdisc mq
```

按用例推荐的 **NIC**

以 1200 字节平均数据包大小 (现代移动流量):

用例	推荐 **NIC**	模式	**Mpps** 容量	吞吐量 **(N3)**	部署场景
测试/开发	任何 NIC (VirtIO, E1000)	通用	1-2 Mpps	1-2 Gbps	实验室测试，PoC
小型基站	Intel X710, Mellanox CX-5	本地	5-10 Mpps	5-10 Gbps	农村基站，企业
中型基站/大都市	Intel E810, Mellanox CX-6	本地	10-20 Mpps	10-20 Gbps	城市基站，小城市
大型都市	Mellanox CX-6, Intel E810 (双)	本地	20-40 Mpps	20-40 Gbps	大都市，中型城市
区域中心	Mellanox BlueField, Netronome Agilio	卸载	40+ Mpps	40+ Gbps	区域聚合
Proxmox VM (桥接)	VirtIO	通用	1-2 Mpps	1-2 Gbps	仅测试
Proxmox VM (SR-IOV)	Intel X710/E810 VF, Mellanox CX-5 VF	本地	5-10 Mpps	5-10 Gbps	生产 VM

吞吐量估算:

- 基于 1200 字节平均数据包大小和 GTP-U 封装 (N3 上 1236 字节)
- N6 吞吐量略低 (~9.6 Gbps 每 10 Mpps) 因为没有 GTP-U 开销
- 实际性能因流量组合而异 - VoIP 密集网络将看到较低吞吐量

其他资源

官方 **XDP** 档:

- XDP 项目
- 内核 XDP 文档

NIC 兼容性列表：

- Cilium XDP 硬件支持
- IO Visor XDP 驱动

配置示例

示例 **1**：开发环境（通用模式）

场景：在没有 SR-IOV 的笔记本或虚拟机上测试 OmniUPF

```
# 开发配置
interface_name: [eth0]
xdp_attach_mode: generic
api_address: :8080
pfcp_address: :8805
pfcp_node_id: 127.0.0.1
n3_address: 127.0.0.1
metrics_address: :9090
logging_level: debug
max_sessions: 1000
```

示例 **2**：生产裸金属（本地模式）

场景：在带 Intel X710 NIC 的裸金属服务器上生产 UPF

```yaml
# 生产裸金属配置
interface_name: [ens1f0, ens1f1]  # N3 在 ens1f0,N6 在 ens1f1
xdp_attach_mode: native
api_address: :8080
pfcp_address: 10.100.50.241:8805
pfcp_node_id: 10.100.50.241
n3_address: 10.100.50.233
n9_address: 10.100.50.234
metrics_address: :9090
logging_level: info
max_sessions: 500000
gtp_peer:
  - 10.100.50.10:2152  # gNB 1
  - 10.100.50.11:2152  # gNB 2
gtp_echo_interval: 30
pfcp_remote_node:
  - 10.100.50.50  # OmniSMF
heartbeat_interval: 10
feature_ueip: true
ueip_pool: 10.45.0.0/16
buffer_max_packets: 50000
buffer_packet_ttl: 60
```

示例 3: Proxmox VM 带 SR-IOV (本地模式)

场景: 在 Proxmox VM 上带 SR-IOV 直通的生产 UPF

```yaml
# Proxmox SR-IOV 配置
interface_name: [ens1f0]  # SR-IOV VF
xdp_attach_mode: native
api_address: :8080
pfcp_address: 192.168.100.10:8805
pfcp_node_id: 192.168.100.10
n3_address: 192.168.100.10
metrics_address: :9090
logging_level: info
max_sessions: 100000
gtp_peer:
  - 192.168.100.50:2152
gtp_echo_interval: 15
pfcp_remote_node:
  - 192.168.100.20  # SMF
```

示例 4: PGW-U 模式 (4G EPC)

场景: OmniUPF 在 4G EPC 网络中充当 PGW-U

```yaml
# PGW-U 配置
interface_name: [eth0]
xdp_attach_mode: native
api_address: :8080
pfcp_address: 10.200.1.10:8805
pfcp_node_id: 10.200.1.10
n3_address: 10.200.1.10  # S5/S8 接口 (GTP-U)
metrics_address: :9090
logging_level: info
max_sessions: 200000
gtp_peer:
  - 10.200.1.50:2152  # SGW-U
gtp_echo_interval: 20
pfcp_remote_node:
  - 10.200.2.10  # OmniPGW-C (Sxb 接口)
heartbeat_interval: 5
```

示例 5: 多模式（同时 UPF + PGW-U)

场景: OmniUPF 同时为 5G 和 4G 网络□供服务

```yaml
# 多模式配置
interface_name: [eth0, eth1]
xdp_attach_mode: native
api_address: :8080
pfcp_address: :8805
pfcp_node_id: 10.50.1.100
n3_address: 10.50.1.100
n9_address: 10.50.1.101
metrics_address: :9090
logging_level: info
max_sessions: 300000
gtp_peer:
  - 10.50.2.10:2152  # 5G gNB
  - 10.50.2.20:2152  # 4G eNodeB (通过 SGW-U)
gtp_echo_interval: 15
pfcp_remote_node:
  - 10.50.3.10  # OmniSMF (5G)
  - 10.50.3.20  # OmniPGW-C (4G)
heartbeat_interval: 10
feature_ueip: true
ueip_pool: 10.60.0.0/16
```

示例 6: SmartNIC 卸载模式

场景: 使用 Netronome Agilio CX SmartNIC 的超高吞吐量部署

```yaml
# SmartNIC 卸载配置
interface_name: [enp1s0np0]  # SmartNIC 接口
xdp_attach_mode: offload
api_address: :8080
pfcp_address: 10.10.1.50:8805
pfcp_node_id: 10.10.1.50
n3_address: 10.10.1.50
metrics_address: :9090
logging_level: warn  # 减少开销
max_sessions: 1000000
pdr_map_size: 2000000
far_map_size: 2000000
qer_map_size: 1000000
gtp_peer:
  - 10.10.2.10:2152
  - 10.10.2.20:2152
  - 10.10.2.30:2152
gtp_echo_interval: 30
pfcp_remote_node:
  - 10.10.3.10
heartbeat_interval: 15
buffer_max_packets: 100000
buffer_max_total: 1000000
```

映射大小和容量规划

自动调整大小（推荐）

设置 `max_sessions` 并让 OmniUPF 自动计算映射大小:

```yaml
max_sessions: 100000
# 自动计算大小:
# PDR: 200,000 条目 (2 × max_sessions)
# FAR: 200,000 条目 (2 × max_sessions)
# QER: 100,000 条目 (1 × max_sessions)
# URR: 200,000 条目 (2 × max_sessions)
```

内存使用：~91 MB 用于 100K 会话

手动调整大小

覆盖自动计算以满足自定义要求：

```yaml
max_sessions: 100000
pdr_map_size: 300000  # 支持每个会话更多的 PDR
far_map_size: 200000
qer_map_size: 150000  # 比默认更多的 QER
urr_map_size: 200000
```

容量估算

计算最大会话数：

```
Max Sessions = min(
  pdr_map_size / 2,
  far_map_size / 2,
  qer_map_size
)
```

示例：

- PDR 映射：200,000
- FAR 映射：200,000
- QER 映射：100,000

Max Sessions = min(100,000, 100,000, 100,000) = **100,000**

内存要求

每会话内存使用：

- PDR: 2 × 212 B = 424 B
- FAR: 2 × 20 B = 40 B
- QER: 1 × 36 B = 36 B
- URR: 2 × 20 B = 40 B
- 总计: ~540 B 每会话

对于 **100K** 会话: ~52 MB 内核内存

建议: 确保锁定的内存限制允许 2× 估算使用:

```
# 检查当前限制
ulimit -l

# 设置为无限制 (eBPF 需要)
ulimit -l unlimited
```

相关□档

- 架构指南 - eBPF/XDP 技术细节和性能优化
- 规则管理指南 - PDR、FAR、QER、URR 配置
- 监控指南 - 统计、容量监控和警报
- 指标参考 - 完整的 Prometheus 指标参考
- **Web UI** 指南 - 控制面板操作
- 操作指南 - UPF 架构和部署概述

指标参考

本文档描述了 OmniUPF 在 `/metrics` 端点上暴露的所有 Prometheus 指标。

指标类别

1. **PFCP** 消息指标 - 控制平面协议消息计数器和每个对等体的延迟
2. **XDP** 动作指标 - 数据平面数据包裁决（丢弃、通过、重定向等）
3. 数据包指标 - 按协议类型接收的数据包计数器
4. **PFCP** 会话和关联指标 - 每个对等体的会话和关联计数
5. **URR** 指标 - 按 PFCP 对等体汇总的流量量计数器
6. 数据包缓冲指标 - 数据包缓冲状态、容量和吞吐量
7. 下行数据报告（通知）指标 - PFCP 会话报告请求通知和 FAR 索引跟踪
8. **eBPF** 映射容量指标 - eBPF 映射利用率和容量

指标参考

PFCP 消息指标

用于跟踪 UPF 与控制平面节点之间的 PFCP 协议消息的指标。

指标名称	类型	标签	描述
upf_pfcp_rx	计数器	`message_name`, `peer_address`	每种消息类型和对等体接收的 PFCP 消息总数
upf_pfcp_tx	计数器	`message_name`, `peer_address`	每种消息类型和对等体传输的 PFCP 消息总数
upf_pfcp_rx_errors	计数器	`message_name`, `cause_code`, `peer_address`	每种消息类型和对等体因错误原因被拒绝的 PFCP 消息总数
upf_pfcp_rx_latency	概要	`message_type`, `peer_address`	每种消息类型和对等体的 PFCP 消息处理时长（微秒，p50, p90, p99 分位数）

注意：所有计数器跟踪每个 PFCP 对等体的消息，以便对控制平面节点行为进行细粒度的可见性。

XDP 动作指标

按 XDP 程序动作/裁决的数据包计数器。这些指标跟踪每个数据包的数据平面决策。

指标名称	类型	标签	描述
upf_xdp_aborted	计数器	无	被中止的数据包总数（XDP_ABORTED）
upf_xdp_drop	计数器	无	被丢弃的数据包总数（XDP_DROP）
upf_xdp_pass	计数器	无	传递给内核的数据包总数（XDP_PASS）
upf_xdp_tx	计数器	无	传输的数据包总数（XDP_TX）
upf_xdp_redirect	计数器	无	被重定向的数据包总数（XDP_REDIRECT）

数据包指标

按协议类型接收的数据包计数器。所有指标使用 `packet_type` 标签。

指标名称	类型	标签	描述
upf_rx	计数器	`packet_type`	按类型接收的数据包总数
upf_route	计数器	`packet_type`	按查找结果路由的数据包总数

upf_rx packet_type 值:

- `arp` - ARP 数据包
- `icmp` - ICMP 数据包
- `icmp6` - ICMPv6 数据包
- `ip4` - IPv4 数据包
- `ip6` - IPv6 数据包
- `tcp` - TCP 数据包
- `udp` - UDP 数据包
- `other` - 其他数据包类型
- `gtp-echo` - GTP 回显请求/响应
- `gtp-pdu` - GTP-U PDU (封装用户数据)
- `gtp-other` - 其他 GTP 消息类型
- `gtp-unexp` - 意外/格式错误的 GTP 数据包

upf_route packet_type 值:

- `ip4-cache` - IPv4 路由缓存命中
- `ip4-ok` - IPv4 FIB 查找成功
- `ip4-error-drop` - IPv4 FIB 查找失败，数据包被丢弃
- `ip4-error-pass` - IPv4 FIB 查找失败，数据包传递给内核
- `ip6-cache` - IPv6 路由缓存命中
- `ip6-ok` - IPv6 FIB 查找成功
- `ip6-error-drop` - IPv6 FIB 查找失败，数据包被丢弃
- `ip6-error-pass` - IPv6 FIB 查找失败，数据包传递给内核

PFCP 会话和关联指标

用于跟踪 UPF 与控制平面节点之间的 PFCP 会话和关联的指标。

指标名称	类型	标签	描述
upf_pfcp_sessions	计量器	无	当前建立的 PFCP 会话总数（所有对等体）
upf_pfcp_associations	计量器	无	当前建立的 PFCP 关联总数（所有对等体）
upf_pfcp_association_status	计量器	`node_id`, `address`	每个对等体的 PFCP 关联状态（1=上行，0=下行）
upf_pfcp_sessions_per_node	计量器	`node_id`, `address`	每个控制平面节点的活动 PFCP 会话数量

URR（使用报告规则）指标

按 PFCP 对等体汇总的流量量指标。每个对等体的流量表示该控制平面节点所有会话的所有 URR 计数器的总和。

指标名称	类型	标签	描述
upf_urr_uplink_volume_bytes	计量器	`peer_address`	此对等体所有会话的上行流量总量（字节）
upf_urr_downlink_volume_bytes	计量器	`peer_address`	此对等体所有会话的下行流量总量（字节）
upf_urr_total_volume_bytes	计量器	`peer_address`	此对等体所有会话的总流量（字节）（上行 + 下行）

注意：按 PFCP 对等体汇总流量以避免高基数问题。个别 URR 统计信息可通过 REST API 在 `/api/v1/urr_map` 获取。

数据包缓冲指标

用于跟踪数据包缓冲状态和性能的指标。当 UE 处于空闲状态时，UPF 可以缓冲下行数据包，直到 UE 被寻呼并过渡到连接状态。

□□□□	□□	□□	□□
upf_buffer_packets_total	□□□	□	□□□□ □□□□ □□□□ □□□□ □□
upf_buffer_packets_dropped	□□□	`reason`	□□□□ □□□□ □□□□
upf_buffer_packets_flushed	□□□	□	□□□□ □□□□ □□□□
upf_buffer_packets_current	□□□	□	□□□□ □□□□ □□□□
upf_buffer_bytes_total	□□□	□	□□□□ □□□□ □□□ □□□□ □□
upf_buffer_bytes_current	□□□	□	□□□□ □□□□ □□
upf_buffer_fars_active	□□□	□	□□□□ □□□□ □ FAR □□

□□□□	□□	□□	□□
upf_buffer_listener_packets_received_total	□□□	□	□ eBPF □□□□ □□□□ □□□□ □□□□
upf_buffer_listener_packets_buffered_total	□□□	□	□□□□ □□□□ □□□□ □
upf_buffer_listener_errors_total	□□□	`type`	□□□□ □□□□ □□□□ □□
upf_buffer_listener_error_indications_sent_total	□□□	`remote_peer`	□□□□ TEID □□□ GTP-U □□□ □□□□ □
upf_buffer_flush_success_total	□□□	□	□□□□ □□□□ □□□
upf_buffer_flush_errors_total	□□□	`reason`	□□□□ □□□□ □□□

指标名称	类型	标签	描述
upf_buffer_flush_packets_sent_total	计数器	无	在冲洗操作期间发送的总数据包数

upf_buffer_packets_dropped reason 值:

- `expired` - 由于 TTL 到期而丢弃的数据包
- `global_limit` - 由于达到总缓冲限制而丢弃
- `far_limit` - 由于达到每个 FAR 缓冲限制而丢弃
- `cleared` - 从缓冲区手动清除的数据包

upf_buffer_listener_errors_total type 值:

- `read_error` - 从缓冲套接字读取时出错
- `too_small` - 数据包太小，无法容纳 GTP 头
- `invalid_gtp_type` - 非 G-PDU GTP 消息类型
- `unknown_teid` - 找不到 TEID 的 PDR/FAR
- `not_buffering_far` - FAR 没有 BUFF 动作
- `truncated_ext` - GTP 扩展头截断
- `no_payload` - GTP 数据包没有有效负载
- `buffer_full` - 超过缓冲容量

upf_buffer_flush_errors_total reason 值:

- `far_lookup_failed` - 无法从 eBPF 映射中查找 FAR 信息
- `no_forw_action` - FAR 没有设置 FORW 动作
- `connection_failed` - 创建用于冲洗的 UDP 连接失败

下行数据报告（通知）指标

用于 PFCP 会话报告请求通知的指标，当数据包被缓冲时发送到控制平面。这些通知触发控制平面寻呼 UE。

□□□□	□□	□□	□□
upf_dldr_sent_total	□□□	□	□□□ SMF □ □□□□□□ □DLDR□□□ □□
upf_dldr_send_errors	□□□	□	□□□□□□□□□ □□□□□□□
upf_dldr_active_notifications	□□□	□	□□□□□ DLDR □□□ FAR □□□□□ □□□
upf_far_index_size	□□□	□	��□□ FarIndex □ □□□ FAR □□
upf_far_index_registrations_total	□□□	□	□ FarIndex □□ FAR □□□ □
upf_far_index_unregistrations_total	□□□	□	□ FarIndex □□□□□ FAR □□
upf_buffer_notify_to_flush_duration_seconds	□□□	`pfcp_peer`	□□ DLDR □ □□□□□□□□□ □□□□□

upf_buffer_notify_to_flush_duration_seconds:

- □□□□□0.01, 0.05, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 30.0, 60.0 □
- □□ `pfcp_peer`□SMF/PGW-C □□□□□□`10.100.50.241`□

- 测量 UPF 发送通知到 SMF 和 SMF 响应以冲洗数据包的延迟
- 有助于监控控制平面在空闲到连接过渡期间的响应能力

GTP-U 错误指示指标

用于跟踪发送和接收的 GTP-U 错误指示消息的指标。当对等体接收到未知 TEID 的数据包时，会发送错误指示，表示隧道状态不匹配（通常是由于对等体重启）。

指标名称	类型	标签
upf_buffer_listener_error_indications_sent_total	计数器	`node_id`, `peer_addres`
upf_buffer_listener_error_indications_received_total	计数器	`node_id`, `peer_addres`
upf_buffer_listener_error_indication_sessions_deleted_total	计数器	`node_id`, `peer_addres`

标签定义：

- `node_id`：来自关联的 PFCP 节点 ID（例如，`"pgw-u-1"`，`"smf-1"`）。如果该对等体没有 PFCP 关联，则设置为 `"unknown"`。
- `peer_address`：远程对等体的 IP 地址（例如，`"192.168.50.10"`）

发送错误指示的情况：

- UPF 收到一个 TEID 的 GTP-U 数据包，但该 TEID 不存在（例如，在 UPF 重启后，会话已被删除）
- 发送方（eNodeB、gNodeB、上游 UPF）正在转发到过时/已删除的隧道
- UPF 发送错误指示以通知发送方停止发送

接收错误指示的情况：

- UPF 将 GTP-U 数据包转发到下游对等体（PGW-U、SGW-U、UPF），但 TEID 不明
- 远程对等体无法识别目标 TEID（例如，对等体重启并丢失隧道状态）
- UPF 自动删除受影响的会话，以停止转发到死隧道

用例：

- 检测对等体重启（高错误指示率表明状态丢失）
- 识别配置不匹配（TEID 分配问题）
- 监控网络元素之间的隧道同步健康
- 对意外会话删除发出警报

示例 **PromQL** 查询：

```
# 每个对等体接收的错误指示速率（每秒）
rate(upf_buffer_listener_error_indications_received_total[5m])

# 由于□定对等体的错误指示而删除的会话总数
upf_buffer_listener_error_indication_sessions_deleted_total{peer_addr

# 向此 UPF 发送未知 TEID 的对等体
sum by (node_id, peer_address) (upf_buffer_listener_error_indications
```

eBPF 映射容量指标

用于跟踪 eBPF 映射利用率的指标。这些指标有助于监控资源使用情况并检测潜在的容量问题。

□□□□	□□	□□	□□
upf_ebpf_map_capacity	□□□	`map_name`	eBPF □□□□□□□
upf_ebpf_map_used	□□□	`map_name`	eBPF □□□□□□□□□□

□□ **map_name** □□

- `pdr_map` - □□□□□□□□□
- `far_map` - □□□□□□□□
- `qer_map` - QoS □□□□□□
- `session_map` - □□□□□□
- `teid_map` - TEID □□□□□□
- `ue_ip_map` - UE IP □□□□□□□□

��□ Prometheus □□

□□□□

□□□□□□□□□□□ `metrics_address` □□□ `/metrics` □□□□□□□□ `:9090`□□

```
# □□□□□□
curl http://localhost:9090/metrics

# □□□□
upf_pfcp_sessions 42
upf_pfcp_associations 2
upf_urr_total_volume_bytes{peer_address="10.100.50.241"}
1048576000
```

Prometheus □□

□ OmniUPF □□□□□□□ `prometheus.yml`□

```
scrape_configs:
  - job_name: 'omniupf'
    static_configs:
      - targets: ['localhost:9090']
```

Grafana 仪表板

将指标导入 Grafana 进行可视化：

- 会话计数和趋势
- 每个 PFCP 对等体的流量量
- 数据包处理速率
- 缓冲利用率
- eBPF 映射容量监控

相关□档

- 监控指南 - 统计监控、容量规划和警报
- 配置指南 - 配置 `metrics_address` 和其他 UPF 选项
- **Web UI** 指南 - 在统计页面查看指标
- 架构指南 - eBPF 数据路径和性能优化
- 规则管理指南 - 理解 PDR、FAR、QER、URR 指标
- 故障排除指南 - 使用指标进行诊断

监控指南

目录

概述

有效监控 OmniUPF 对于维护服务质量、预防容量耗尽和故障排除性能问题至关重要。OmniUPF 通过其 Web UI 和 REST API 供全面的实时指标。

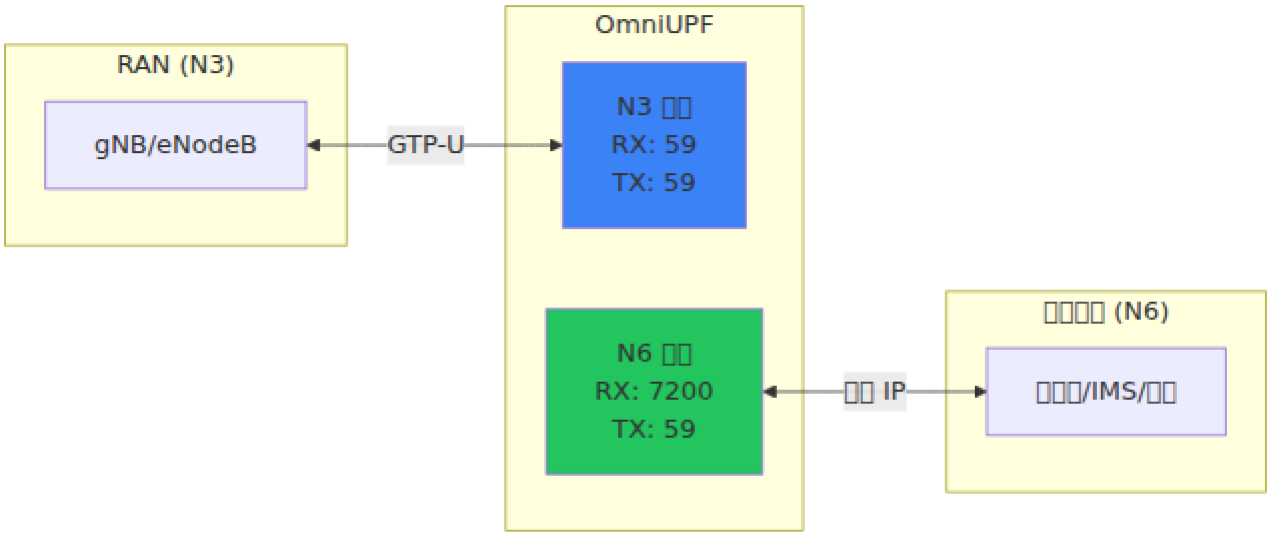
监控类别

类别	目的	更新频率	关键指标
数据包统计	跟踪数据包处理速率和错误	实时	RX/TX 数据包、丢包、协议分解
接口统计	监控 N3/N6 流量分布	实时	N3 RX/TX, N6 RX/TX
XDP 统计	跟踪内核数据路径性能	实时	XDP 处理、通过、丢弃、中止
路由统计	监控数据包路由决策	实时	FIB 查找、缓存命中/未命中
eBPF 映射容量	防止资源耗尽	每 10 秒	映射使用百分比、已用与容量
缓冲区统计	跟踪移动过程中的数据包缓冲	每 5 秒	缓冲的数据包、缓冲区年龄、FAR 计数

统计监控

N3/N6 接口统计

N3/N6 接口统计□供了 RAN (N3) 和数据网络 (N6) 之间流量分布的可见性。



指标:

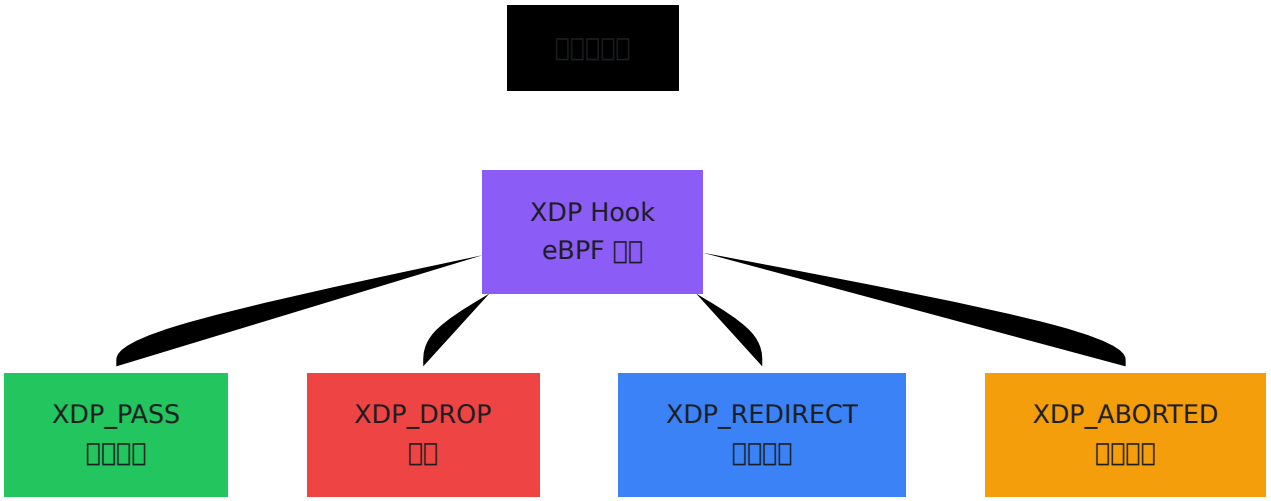
- **RX N3**: 从 RAN 接收的数据包（上行 GTP-U 流量）
- **TX N3**: 发送到 RAN 的数据包（下行 GTP-U 流量）
- **RX N6**: 从数据网络接收的数据包（下行原生 IP）
- **TX N6**: 发送到数据网络的数据包（上行原生 IP）
- 总计：所有接口的聚合数据包计数

预期行为:

- **RX N3 ≈ TX N6**: 上行数据包从 RAN 流向数据网络
- **RX N6 ≈ TX N3**: 下行数据包从数据网络流向 RAN
- 显著的不平衡可能表明:
  - 非对称流量（下载 >> 上传）
  - 数据包丢失或转发错误
  - 路由配置错误

XDP □□

XDP (eXpress Data Path) □□□□□□□□□□□□□□□□□



□□□

- □□□XDP □□□□□□□□□□□□ 0□

- □□□XDP □□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□
- **TX**□□□ XDP □□□□□□□

□□□

- □□ **> 0**□eBPF □□□□□□□□□□□□□□□□□
- □□ **> 0**□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□

- **RX ARP**□□□□□□□□□□□
- **RX GTP ECHO**□GTP-U □□□□/□□□□□□□□□
- **RX GTP OTHER**□□□ GTP □□□□
- **RX GTP PDU**□GTP-U □□□□□□□□□□□□□□□
- **RX GTP UNEXP**□□□□ GTP □□□□□
- **RX ICMP**□□□□□□□□□□□□□ping□□□□
- **RX ICMP6**□ICMPv6 □□□
- **RX IP4**□IPv4 □□□
- **RX IP6**□IPv6 □□□
- **RX OTHER**□□□□□
- **RX TCP**□□□□□□□□□□
- **RX UDP**□□□□□□□□□□□

□□□

- □□ **GTP-U PDU** □□□□□□□□□□□□
- □□ **ICMP** □□□□□□□□□□□□□□□

- 跟踪 **TCP** 与 **UDP** 比率：应用流量模式
- 检测意外协议：安全或配置问题

路由统计

FIB（转发信息库）查找统计用于路由决策。

IPv4 FIB 查找：

- 缓存：缓存的路由查找（快速路径）
- **OK**：成功的路由查找

IPv6 FIB 查找：

- 缓存：缓存的 IPv6 路由查找
- **OK**：成功的 IPv6 路由查找

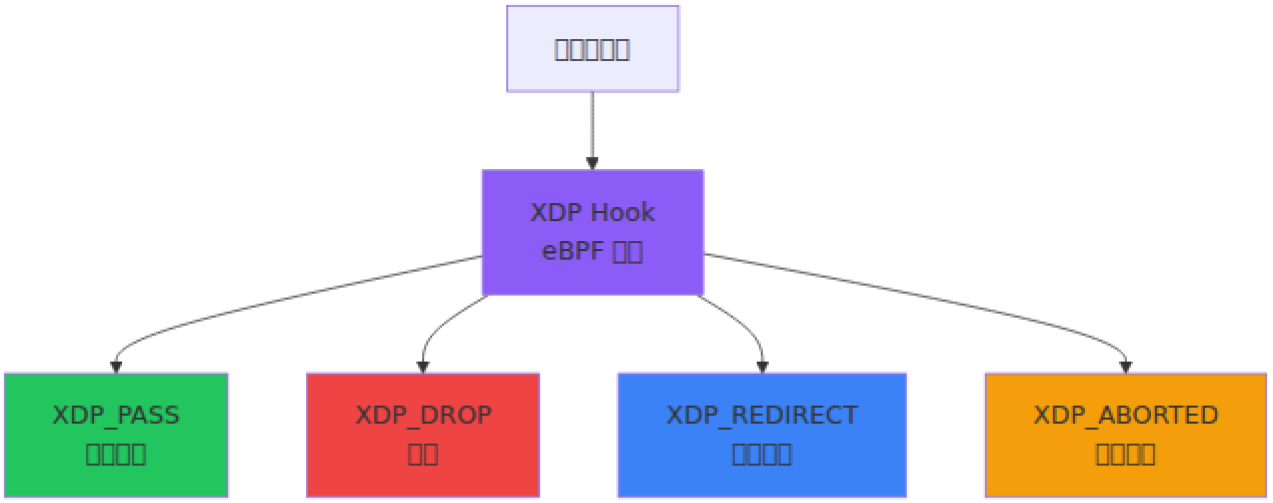
性能指标：

- 高缓存命中率：表明良好的路由缓存性能
- 高 **OK** 计数：确认路由表配置正确
- 低或零查找：可能表明流量未流动或路由绕过

容量监控

eBPF 映射容量

eBPF 映射容量监控防止由于资源耗尽而导致会话建立失败。



关键 **eBPF** 映射

far_map (转发动作规则):

- 容量: 131,070 条目
- 键大小: 4 B (FAR ID)
- 值大小: 16 B (转发参数)
- 内存使用: ~2.6 MB
- 重要性: 高 - 用于所有数据包转发决策

pdr_map_downlin (下行 PDRs - IPv4):

- 容量: 131,070 条目
- 键大小: 4 B (UE IPv4 地址)
- 值大小: 208 B (PDR 信息)
- 内存使用: ~27 MB
- 重要性: 关键 - 如果满了会话建立失败

pdr_map_downlin_ip6 (下行 PDRs - IPv6):

- 容量: 131,070 条目
- 键大小: 16 B (UE IPv6 地址)
- 值大小: 208 B (PDR 信息)
- 内存使用: ~29 MB
- 重要性: 关键 - 如果满了 IPv6 会话建立失败

pdr_map_teid_ip (上行 PDRs):

- 容量: 131,070 条目
- 键大小: 4 B (TEID)
- 值大小: 208 B (PDR 信息)
- 内存使用: ~27 MB
- 重要性: 关键 - 如果满了上行流量失败

qer_map (QoS 执行规则):

- 容量: 65,535 条目
- 键大小: 4 B (QER ID)
- 值大小: 32 B (QoS 参数)
- 内存使用: ~2.3 MB
- 重要性: 中 - 仅用于 QoS 执行

urr_map (使用报告规则):

- 容量: 131,070 条目
- 键大小: 4 B (URR ID)
- 值大小: 16 B (流量计数器)
- 内存使用: ~2.6 MB
- 重要性: 低 - 仅影响计费

容量阈值

阈值	需要采取的行动
0-50% (绿色)	正常操作 - 无需采取行动
50-70% (黄色)	注意 - 监控增长趋势，计划增加容量
70-90% (琥珀)	警告 - 在 1 周内安排增加容量
90-100% (红色)	紧急 - 需要立即采取行动，新会话将失败

增加容量程序

在增加容量之前：

1. 审查当前使用趋势
2. 估算未来增长率

3. 计算所需容量

增加映射容量的步骤：

1. 停止 OmniUPF 服务
2. 使用新映射大小更新 UPF 配置文件
3. 重新启动 OmniUPF 服务
4. 在容量视图中验证新容量
5. 监控会话建立是否成功

注意：更改 eBPF 映射容量需要重新启动 UPF，并清除所有现有会话。

性能指标

有关 OmniUPF 所暴露的所有 Prometheus 指标的详细信息，请参阅 指标参考。

数据包处理速率

计算：

```
数据包速率 (pps) = (数据包计数增量) / (时间增量（秒）)
```

示例：

- 初始 RX 数据包：7,000
- 10 秒后：17,000
- 数据包速率 = (17,000 - 7,000) / 10 = 1,000 pps

性能目标：

- 小型 **UPF**：10,000 - 100,000 pps
- 中型 **UPF**：100,000 - 1,000,000 pps
- 大型 **UPF**：1,000,000 - 10,000,000 pps

瓶颈指标：

- XDP 中止计数增加
- CPU 利用率高
- 数据包丢失增加
- 延迟增加

吞吐量计算

计算:

```
吞吐量 (Mbps) = (字节计数增量 × 8) / (时间增量（秒） × 1,000,000)
```

示例:

- 初始 RX 字节: 500 MB
- 60 秒后: 800 MB
- 吞吐量 = (300 MB × 8) / (60 × 1,000,000) = 40 Mbps

容量规划:

- 监控峰值吞吐量时间（例如，晚上）
- 与链路容量（N3/N6 接口速度）进行比较
- 计划 2 倍峰值吞吐量以留出余地

丢包率

计算:

```
丢包率 (%) = (丢失的数据包 / 总 RX 数据包) × 100
```

可接受的阈值:

- **< 0.1%**: 优秀（正常的数据包丢失由于错误）
- **0.1% - 1%**: 良好（轻微问题或速率限制）

- **1% - 5%**: 差（调查 QoS 或容量问题）
- **> 5%**: 紧急（重大转发或容量问题）

常见丢包原因:

- QER 速率限制（MBR 超过）
- eBPF 映射查找失败
- 无效的 TEID 或 UE IP
- 路由错误

警报和阈值

推荐警报

关键警报（需要立即响应）：

- eBPF 映射容量 > 90%
- XDP 中止计数 > 0
- 丢包率 > 5%
- UPF 健康检查失败

警告警报（1 小时内响应）：

- eBPF 映射容量 > 70%
- 丢包率 > 1%
- 数据包速率接近链路容量
- 缓冲区 TTL 超过（数据包超过 30 秒）

信息警报（监控趋势）：

- eBPF 映射容量 > 50%
- 缓冲的数据包计数增加
- 新的 PFCP 关联已建立/释放
- URR 流量阈值超过

□□□□

□□□□□□□□□□□□□□□

1. **Prometheus** □□□□□□□□□□□□□□□□□□□□ □□□□ □□□□□□□□
2. □□□□□□□□ OmniUPF □□□□□□□□□□
3. **REST API** □□□□□□□□ `/map_info`□`/packet_stats` □□
4. **Web UI** □□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□

□□□□□□□□□

```
□□□□□□ = min(
  PDR □□□□ / 2,  # ●●●□ + □□ PDR □□□
  FAR □□□□ / 2,  # □□ + □□ FAR □□□
  QER □□□□       # □□□□□□□□□ QER
)
```

□□□

- PDR □□□□□131,070
- FAR □□□□□131,070
- QER □□□□□65,535

□□□□□□ = min(131,070 / 2, 131,070 / 2, 65,535) = **65,535** □□

□□□□

□□□ **eBPF** □□□□□

```
□□ = Σ (□□□□ × (□□□ + □□□))
```

示例配置：

- PDR 映射：3 × 131,070 × 212 B = 83.3 MB
- FAR 映射：131,070 × 20 B = 2.6 MB
- QER 映射：65,535 × 36 B = 2.3 MB
- URR 映射：131,070 × 20 B = 2.6 MB
- 总计：~91 MB 的内核内存

内核内存注意事项：

- 确保足够的锁定内存限制（`ulimit -l`）
- 为安全边际保留 2 倍的估计使用量
- 监控内核内存可用性

流量容量

计算所需的吞吐量容量：

1. 估算平均会话吞吐量：
   - 视频流：~5 Mbps
   - 网页浏览：~1 Mbps
   - VoIP：~0.1 Mbps
2. 计算总吞吐量：

   ```
   总吞吐量 = 会话数 × 平均会话吞吐量
   ```

3. 增加余地：

   ```
   所需容量 = 总吞吐量 × 2  # 100% 余地
   ```

示例：

- 10,000 个并发会话
- 每会话平均 2 Mbps

- 总计：20 Gbps
- 所需容量：40 Gbps (N3 + N6 接口)

增长规划

趋势分析：

1. 记录每日峰值会话数
2. 计算每周增长率
3. 外推至容量限制

增长率公式：

```
到达容量的周数 = (容量 - 当前使用) / (每周增长)
```

示例：

- 当前会话：30,000
- 容量：65,535 会话
- 每周增长：2,000 会话
- 到达容量的周数：(65,535 - 30,000) / 2,000 = **17.8** 周

行动：计划在 12 周内进行容量升级（留出 5 周的缓冲）。

故障排除性能问题

高丢包率

症状：丢包率 > 1%，用户投诉连接性差

诊断：

1. 检查统计 → 数据包统计
2. 确定丢包是否特定于协议
3. 查看 XDP 统计以获取 XDP 丢包与中止的对比

常见原因：

- **QER** 速率限制：检查 QER MBR 值与实际流量
- 无效 **TEID**：验证上行 PDR TEID 是否与 gNB 分配匹配
- 未知 **UE IP**：验证下行 PDR 是否存在于 UE IP
- 缓冲区溢出：检查缓冲区统计

解决方案：

- 如果存在速率限制，则增加 QER MBR
- 验证 SMF 是否创建了正确的 PDR
- 如果检测到溢出，则清除缓冲区

XDP 处理错误

症状：XDP 中止 > 0

诊断：

1. 导航到统计 → XDP 统计
2. 检查中止计数
3. 查看 OmniUPF 日志以获取 eBPF 错误

常见原因：

- eBPF 程序验证失败
- 内核版本不兼容
- eBPF 映射访问错误
- 内存损坏

解决方案：

- 重新启动 OmniUPF 服务
- 检查内核版本是否满足最低要求（Linux 5.4+）
- 查看 eBPF 程序日志
- 如果问题仍然存在，请联系支持

容量耗尽

症状：会话建立失败，映射容量达到 100%

诊断：

1. 导航到容量页面
2. 确定哪个映射达到 100%
3. 检查会话是否卡住（未被删除）

立即缓解：

1. 确定过期会话（检查会话页面）
2. 请求 SMF 删除旧会话
3. 清除缓冲区以释放 FAR 条目

长期解决方案：

1. 增加 eBPF 映射容量
2. 安排 UPF 重新启动以使用更大的映射
3. 实施会话清理政策

性能下降

症状：高延迟、低吞吐量、CPU 饱和

诊断：

1. 检查数据包速率与历史基线的比较
2. 查看 XDP 统计以获取处理延迟
3. 监控 UPF 主机上的 CPU 利用率
4. 检查 N3/N6 接口利用率

常见原因：

- 流量超过 UPF 容量
- 数据包处理的 CPU 核心不足

- 网络接口瓶颈
- eBPF 映射哈希冲突

解决方案：

- 横向扩展 UPF（添加更多实例）
- 升级 CPU 或启用 RSS（接收端负载均衡）
- 升级网络接口以□高速度
- 调整 eBPF 映射哈希函数

相关□档

- 指标参考 - 完整的 Prometheus 指标参考
- **UPF** 操作指南 - 一般 UPF 架构和操作
- 规则管理指南 - PDR、FAR、QER、URR 配置
- **Web UI** 操作指南 - 控制面板监控功能
- 故障排除指南 - 常见问题和诊断
- 架构指南 - eBPF 数据路径和性能优化

N9 Loopback：在同一实例上运行 **SGWU** 和 **PGWU**

概述

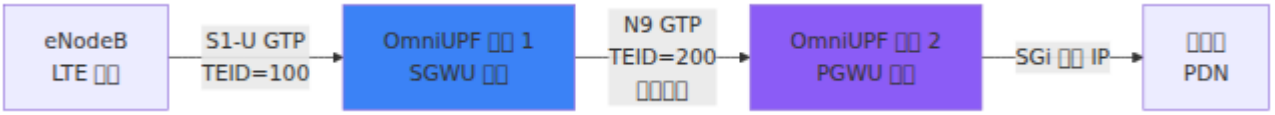
OmniUPF 支持在 同一实例 上运行 **SGWU (**服务网关用户平面**)** 和 **PGWU (PDN** 网关用户平面**)** 功能，并实现 零延迟 **N9** 回环。这种部署模式非常适合：

- 简化的 **4G EPC** 部署 - 单个 UPF 实例而不是两个
- 成本优化 - 降低基础设施和运营复杂性
- 边缘计算 - 最小化本地突破场景的延迟
- 实验室/测试环境 - 在单台服务器上实现完整的 EPC 用户平面

当为 N3 和 N9 接口配置相同的 IP 地址时，OmniUPF 自动检测 SGWU 和 PGWU 角色之间流动的流量，并完全在 eBPF 中处理，而无需将数据包发送到网络接口。

工作原理

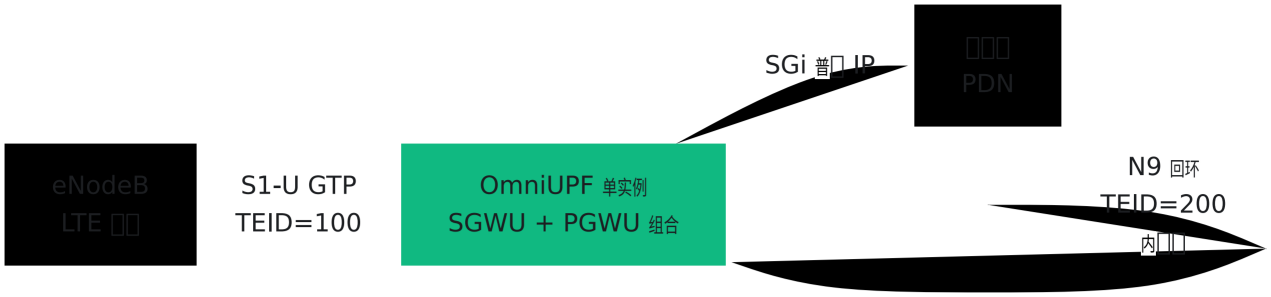
传统部署（两个实例）



数据包流动：

1. eNodeB → SGWU: GTP 数据包 (TEID=100) 到达 S1-U
2. SGWU: 匹配上行 PDR，封装到新的 GTP 隧道 (TEID=200)
3. 数据包通过物理 **N9** 网络发送 到 PGWU 实例
4. PGWU: 接收 GTP (TEID=200)，解封装，转发到互联网
5. 总计：**2** 次 **XDP** 通过 **+ 1** 次网络跳跃

N9 回环部署（单个实例）



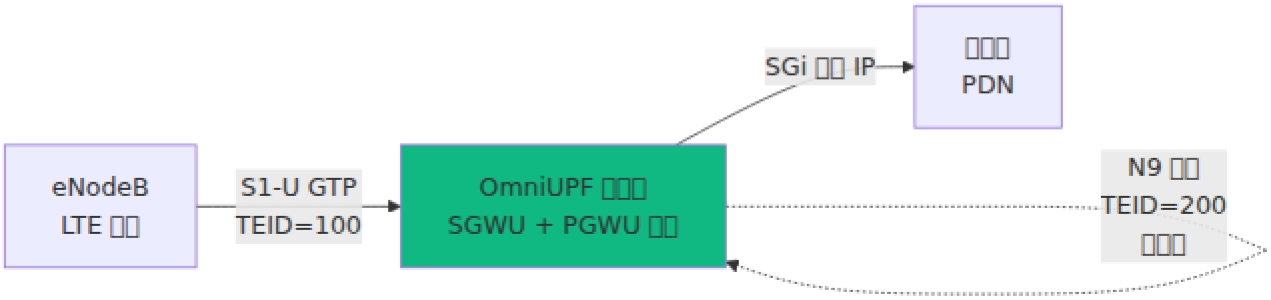
N9 回环的数据包流动：

1. eNodeB → SGWU 角色：GTP 数据包 (TEID=100) 到达 S1-U
2. SGWU 角色：匹配上行 PDR
3. 回环检测：目标 IP = 本地 IP (10.0.1.10)
4. 就地处理：更新 GTP TEID 为 200 (PGWU 会话)
5. PGWU 角色：解封装，转发到互联网
6. 总计：**1** 次 **XDP** 通过，零次网络跳跃

性能优势：亚微秒的内部转发与网络往返的毫秒延迟相比

数据包处理细节

上行流：**eNodeB → SGWU → PGWU →** 互联网



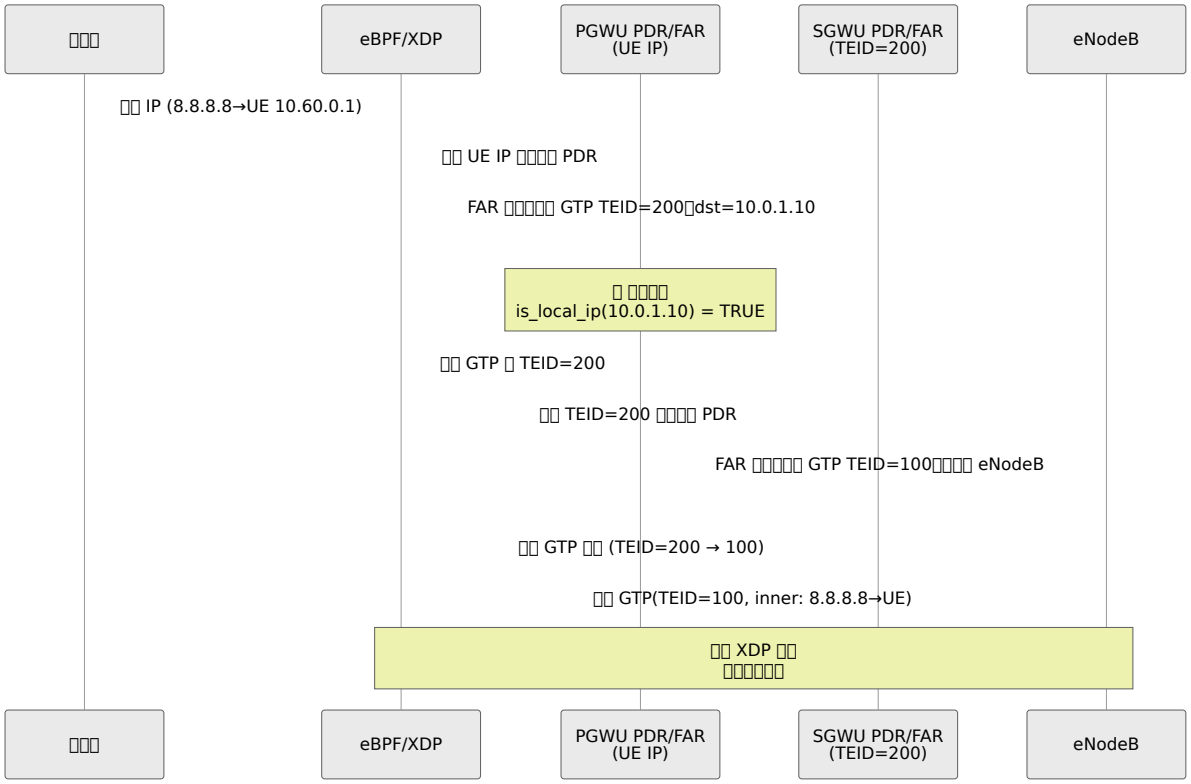
代码路径：`cmd/ebpf/xdp/n3n6_entrypoint.c` 第 349-403 行

关键步骤：

1. 接收：来自 eNodeB 的 GTP 数据包，TEID=100
2. **PDR** 匹配：查找 SGWU 会话的上行 PDR (TEID=100)

3. **FAR** □□□ □ TEID=200 □□ GTP□□□□ 10.0.1.10
4. □□□□□ `is_local_ip(10.0.1.10)` □□ TRUE
5. □□ **TEID**□ □ `ctx->gtp->teid` □ 100 □□□ 200 (□□□□□□□)
6. □□□□□ □□ TEID=200 □ PDR (PGWU □□)
7. **FAR** □□□ □□ GTP □□□□□□□□
8. □□□ □□□ IP □□□□□□ N6 □□

□□□□□□□ → PGWU → SGWU → eNodeB



□□□□□ `cmd/ebpf/xdp/n3n6_entrypoint.c` □ 137-194 □ (IPv4)□265-322 □ (IPv6)

□□□□□

1. □□□ □□□□□□□□ IP □□□□□□□ UE (10.60.0.1)
2. **PDR** □□□ □□ UE IP □□□□ PDR (PGWU □□)
3. **FAR** □□□ □ TEID=200 □□ GTP□□□□ 10.0.1.10
4. □□□□□ `is_local_ip(10.0.1.10)` □□ TRUE
5. □□ **GTP**□ □ TEID=200 □□□□
6. □□□□□ □□ TEID=200 □ PDR (SGWU □□)

7. **FAR** 操作：更新 GTP 隧道到 eNodeB TEID=100
8. 路由：将 GTP 数据包发送到 S1-U 接口 (eNodeB)

配置

要求

控制平面：

- **SGWU-C**：必须连接到 OmniUPF PFCP 接口（例如，`192.168.1.10:8805`）
- **PGWU-C**：必须连接到相同的 OmniUPF PFCP 接口

网络：

- **N3** 和 **N9** 接口的单个 **IP** 地址
- **SGWU-C** 和 **PGWU-C** 的不同 **IP** 地址（如果在同一主机上运行，请使用不同的端口）

OmniUPF 配置

config.yml:

```
# 网络接口
interface_name: [eth0]              # S1-U 和 N9 的单个接口
xdp_attach_mode: native             # 使用原生模式以获得最佳性能

# PFCP 接口
pfcp_address: ":8805"               # 监听所有接口，端口 8805
pfcp_node_id: "192.168.1.10"        # OmniUPF 的 PFCP 节点 ID

# 用户平面接口
n3_address: "10.0.1.10"             # S1-U/N3 接口 IP
n9_address: "10.0.1.10"             # N9 接口 IP（与 N3 相同）

# APIs
api_address: ":8080"                # REST API
metrics_address: ":9090"            # Prometheus 指标（参见指标参考文档）

# 资源池
ueip_pool: "10.60.0.0/16"           # UE IP 地址池
teid_pool: 65535                    # TEID 分配池

# 容量
max_sessions: 100000                # 最大并发 UE 会话
```

关键配置：

- □ **`n3_address`** 和 **`n9_address`** 必须相同 以启用回环
- □ □□□□□□□□ PFCP 监听地址
- □ □□□□□ SGWU + PGWU 负载，确保 `max_sessions` 足够

控制平面配置

SGWU-C 配置

```
# 指向 OmniUPF PFCP 接口
upf_pfcp_address: "192.168.1.10:8805"

# S1-U 接口 (与 OmniUPF n3_address 相同)
sgwu_s1u_address: "10.0.1.10"

# N9 接口用于转发到 PGWU (与 OmniUPF 相同)
sgwu_n9_address: "10.0.1.10"
```

PGWU-C 配置

```
# 指向相同的 OmniUPF PFCP 接口
upf_pfcp_address: "192.168.1.10:8805"

# N9 接口 (接收来自 SGWU 的数据)
pgwu_n9_address: "10.0.1.10"

# SGi 接口用于互联网连接
pgwu_sgi_address: "192.168.100.1"
```

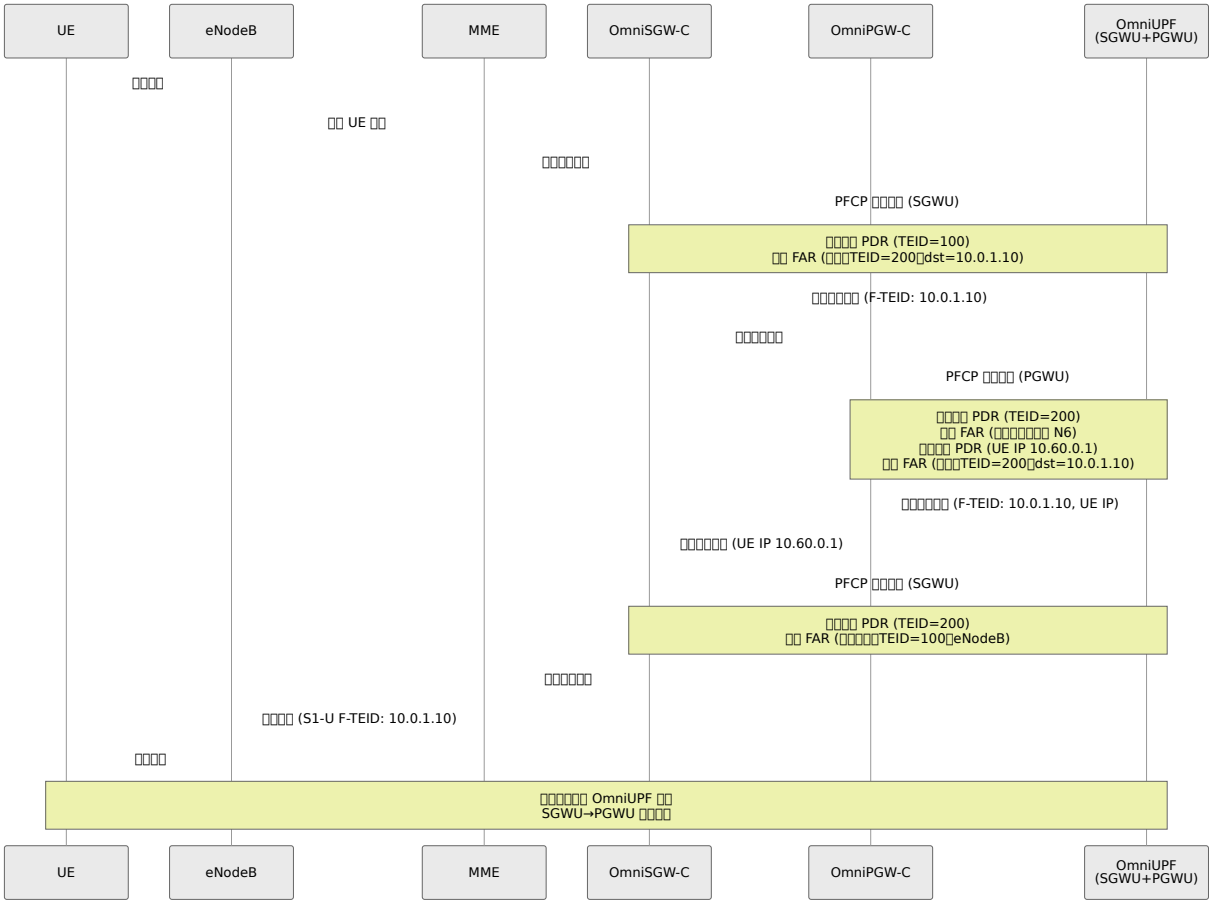
重要:

- 两个控制平面连接到相同的 **PFCP** 端点 (`:8805`)
- OmniUPF 为 SGWU-C 和 PGWU-C 创建单独的 **PFCP** 关联
- 会话按控制平面隔离 (通过节点 ID 跟踪)

会话流示例

UE 附加和 PDU 会话建立

场景: UE 附加到网络, 建立数据会话



□□□ PFCP □□□

SGWU □□□□□ OmniSGW-C□□

- □□ **PDR**□ □□ TEID=100□□□ eNodeB□→ FAR□□□ TEID=200□dst=10.0.1.10
- □□ **PDR**□ □□ TEID=200□□□ PGWU□→ FAR□□□□□ TEID=100□□□□ eNodeB

PGWU □□□□□□□ OmniPGW-C□□

- □□ **PDR**□ □□ TEID=200□□□ SGWU□→ FAR□□□□□□□□□□□□□
- □□ **PDR**□ □□ UE IP=10.60.0.1 → FAR□□□ TEID=200□dst=10.0.1.10

□□□□□

□□ N9 □□□□□□□

□□ XDP □□□

```
# 查看实时 eBPF 调试输出
sudo cat /sys/kernel/debug/tracing/trace_pipe | grep loopback
```

预期输出:

```
upf: [n3] session for teid:100 -> 200 remote:10.0.1.10
upf: [n9-loopback] self-forwarding detected, processing inline
TEID:200
upf: [n9-loopback] decapsulated, routing to N6

upf: [n6] use mapping 10.60.0.1 -> teid:200
upf: [n6-loopback] downlink self-forwarding detected, processing
inline TEID:200
upf: [n6-loopback] SGWU updating GTP tunnel to eNodeB TEID:100
upf: [n6-loopback] forwarding to eNodeB
```

通过 **REST API** 监控会话

列出 **PFCP** 关联:

```
curl http://localhost:8080/api/v1/upf_pipeline | jq
```

预期输出:

```
{
  "associations": [
    {
      "node_id": "sgwc.example.com",
      "address": "192.168.1.20:8805",
      "sessions": 1000
    },
    {
      "node_id": "pgwc.example.com",
      "address": "192.168.1.21:8805",
      "sessions": 1000
    }
  ],
  "total_sessions": 2000
}
```


验证两个单独的关联（一个用于 SGWU-C，一个用于 PGWU-C）

列出活动会话：

```
curl http://localhost:8080/api/v1/sessions | jq '.sessions[] |
{local_seid, ue_ip, uplink_teid}'
```

预期输出：


```
{
  "local_seid": 12345,
  "ue_ip": "10.60.0.1",
  "uplink_teid": 100
}
{
  "local_seid": 67890,
  "ue_ip": "10.60.0.1",
  "uplink_teid": 200
}
```


每个 **UE** 有两个会话：

- 来自 SGWU-C 的会话（TEID=100，S1-U 接口）
- 来自 PGWU-C 的会话（TEID=200，N9 接口）

性能指标

检查数据包统计：

```
curl http://localhost:8080/api/v1/xdp_stats | jq
```

关键指标：

- `xdp_processed`：在 eBPF 中处理的总数据包
- `xdp_pass`：传递到网络栈的数据包（对于回环流量应为零）
- `xdp_redirect`：通过 XDP 重定向转发的数据包
- `xdp_tx`：传输的数据包（回环流量使用此项）

对于 **N9** 回环流量：

- `xdp_pass` 应该是 最小的（仅非回环流量）
- `xdp_tx` 或 `xdp_redirect` 计数回环转发

故障排除

N9 流量发送到网络而不是回环

症状：数据包发送到网络接口，延迟高

根本原因：`n3_address` ≠ `n9_address`

解决方案：

```
# 错误:
n3_address: "10.0.1.10"
n9_address: "10.0.1.20"    # 不同的 IP, 没有回环!

# 正确:
n3_address: "10.0.1.10"
n9_address: "10.0.1.10"    # 相同的 IP, 启用回环
```

验证:

```
curl http://localhost:8080/api/v1/dataplane_config | jq
```

应显示:

```
{
  "n3_ipv4_address": "10.0.1.10",
  "n9_ipv4_address": "10.0.1.10"
}
```

回环后未找到 **PDR**

症状: 日志显示 `[n9-loopback] no PDR for destination TEID`

根本原因: PGWU 会话未创建或 TEID 不匹配

诊断:

1. 检查 **PFCP** 会话:

   ```
   curl http://localhost:8080/api/v1/sessions | jq '.sessions[] |
   select(.uplink_teid == 200)'
   ```

2. 验证 **FAR** 配置:

```
curl http://localhost:8080/api/v1/far_map | jq '.[] |
select(.teid == 200)'
```

□□□□□ □□ PGWU-C □□□ SGWU-C □□ N9 □□□□□ TEID □□□

CPU □□□□

□□□ CPU □□□□□□□

□□□□□ eBPF □□□□□□□□□□□□□□□□□□

□□□

```
# □□ eBPF □□□□□□
sudo bpftool map dump name pdr_map_teid_ip4 | wc -l
sudo bpftool map dump name far_map | wc -l
```

□□□□□

- □□□□□□□□□□ `max_sessions`□□□□□□□□
- □□ QER □□□□□□□□□□□□
- □□□□□□□□□□□□□□□

□□□□□□□□□

□□□ □ eNodeB □□□□□□□□□

□□□□□ □□□□□□□□□□□□

□□□

```
buffer_port: 22152
buffer_max_packets: 20000      # □□□□□□□□□
buffer_max_total: 100000
buffer_packet_ttl: 30          # □□□□□□□□
```

验证：

```
curl http://localhost:8080/api/v1/upf_buffer_info | jq
```

N9 回环的好处

性能

指标	两个实例	单个实例 (N9 回环)	改进
延迟	1-5 毫秒	< 1 微秒	快 **1000** 倍
吞吐量	受网络限制	受 CPU/内存限制	高 **2-3** 倍
CPU 使用率	2× XDP 通过 + 网络栈	1× XDP 通过	减少 **40-50%**
数据包丢失	网络拥塞时风险	零（内存中）	消除

操作

- 简化部署：单个 OmniUPF 实例而不是两个
- 减少基础设施：一半的服务器、网络端口、IP 地址
- 降低复杂性：单一配置，单一监控端点
- 成本节约：降低硬件、电力、冷却、维护成本
- 更容易的故障排除：单个数据包跟踪，单个 eBPF 调试输出

用例

理想适用于：

- □ □□□□□ □□□□□□□□□□
- □ □□/中型部署： < 100K 用户
- □ □□□/测试：在单个 VM 上实现完整的 EPC 用户平面

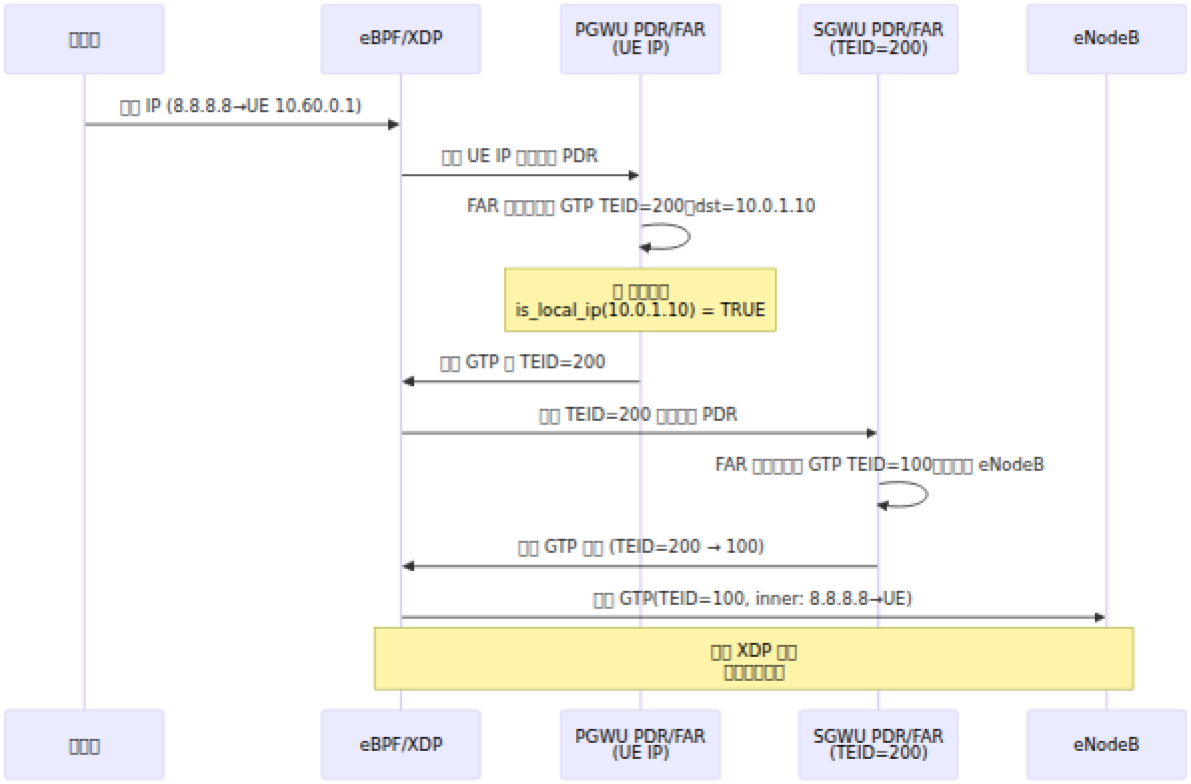
- □ □□□□□□ □□□□□□□

不推荐用于：

- □ □□□□□□ SGWU 和 PGWU 在不同的数据中心
- □ □□□□□ > 1M 用户（考虑水平扩展）
- □ □□□□□□ □□□□□ SGW 和 PGW

与其他部署模式的比较

单个实例（**N9** 回环）与分离实例



总结

N9 回环使 在单个 **OmniUPF** 实例上实现运营商级 **4G EPC** 用户平面 成为可能，通过完全在 eBPF 中处理 SGWU→PGWU 流量而无需网络跳跃。这□供了：

- □ **□□□□□□ □□□□□□□**
- □ **□□ 40-50%** 的 **CPU** 使用率与分离实例相比
- □ **□□□□** - 单个实例、配置、监控
- □ **□□□□** - 一半的基础设施
- □ **□□□□ 3GPP** 标准 - 标准 PFCP、GTP-U 协议

当 `n3_address == n9_address` 时，配置是自动的 - 无需特殊标志或设置。OmniUPF 的 eBPF 数据路径检测回环条件并在线处理数据包。

有关更多信息：

- 配置： CONFIGURATION.md
- 架构： ARCHITECTURE.md
- 指标参考： METRICS.md
- 监控： MONITORING.md
- 操作： OPERATIONS.md
- 故障排除： TROUBLESHOOTING.md

PFCP 原因代码参考

概述

PFCP（数据包转发控制协议）在响应消息中使用原因代码来指示请求的结果。本文档描述了在 OmniUPF 中实现的原因代码及其在 PFCP 消息处理过程中发生的情况。

所有原因代码符合 **3GPP TS 129.244** 规范，并在 PFCP 响应消息中返回，以指示成功、失败或特定错误条件。

监控原因代码

OmniUPF 使用 Prometheus 指标跟踪 PFCP 消息结果。每个 PFCP 响应都包含一个原因代码，该代码记录在：

```
upf_pfcp_rx_errors{message_name="...", cause_code="...",
peer_address="..."}
```

这使得监控以下内容成为可能：

- 每种消息类型和控制平面节点的 成功率
- 指示配置错误或协议问题的 错误模式
- 基于拒绝率的 关联健康状况

请参见 指标参考 获取完整的 PFCP 指标文档。

原因代码类别

成功代码

代码	名称	发生时机
1	**RequestAccepted**	请求成功处理。所有必��的 IE 存在且有效。规则成功创建/修改/删除。

客户端错误代码

代码	名称	发生时机
64	**RequestRejected**	因未指定错误而一般拒绝。用于没有特定原因代码适用的情况。
65	**SessionContextNotFound**	请求对未知 SEID 的会话修改或删除。指定的会话在此 UPF 上不存在。
66	**MandatoryIEMissing**	缺少必需的信息元素。示例：关联建立中缺少 NodeID，会话建立中缺少 F-SEID，缺少 RecoveryTimeStamp。
67	**ConditionalIEMissing**	基于其他 IE 存在的条件性必需 IE 缺失。用于 IE 彼此依赖的情况。
69	**MandatoryIEIncorrect**	必需的 IE 存在但包含无效数据。示例：无法解析的 NodeID 格式，无效的 RecoveryTimeStamp 值，格式错误的 F-SEID。
72	**NoEstablishedPFCPAssociation**	尝试在没有活动关联的情况下进行会话操作。必须在创建会话之前建立 PFCP 关联。
73	**RuleCreationModificationFailure**	将 PDR、FAR、QER 或 URR 规则应用于 eBPF 数据路径时出错。可能的原因：eBPF 映射容量耗尽，无效的规则参数，资源分配失败。

服务器/资源错误代码

代码	名称	发生时机
74	**PFCPEntityInCongestion**	UPF 正在经历高负载或资源耗尽。暂时无法处理请求。
75	**NoResourcesAvailable**	资源不足以满足请求。示例：eBPF 映射容量耗尽，内存分配失败，TEID 池耗尽。
77	**SystemFailure**	关键内部错误阻止请求处理。示例：eBPF 程序失败，内核接口错误，数据库损坏。

不支持的功能代码

代码	名称	发生时机
68	**InvalidLength**	IE 长度字段与实际数据长度不匹配。当前在 OmniUPF 中未使用。
70	**InvalidForwardingPolicy**	UPF 不支持的转发策略。当前在 OmniUPF 中未使用。
71	**InvalidFTEIDAllocationOption**	不支持的 F-TEID 分配选项。当前在 OmniUPF 中未使用。
76	**ServiceNotSupported**	请求的服务或功能未实现。当前在 OmniUPF 中未使用。
78	**RedirectionRequested**	UPF 请求重定向到另一个 UPF 实例。当前在 OmniUPF 中未使用。

常见场景和原因

关联建立失败

场景：缺少 **NodeID**

```
SMF → UPF: 关联建立请求（无 NodeID）
UPF → SMF: 关联建立响应（原因：MandatoryIEMissing）
```

解决方案：确保 SMF 在所有关联建立请求中包含 NodeID IE。

场景：无效的 **NodeID** 格式

```
SMF → UPF: 关联建立请求（NodeID="invalid"）
UPF → SMF: 关联建立响应（原因：MandatoryIEIncorrect）
```

解决方案：NodeID 必须是有效的 FQDN 或 IPv4/IPv6 地址。

场景：缺少恢复时间戳

```
SMF → UPF: 关联建立请求（无 RecoveryTimeStamp）
UPF → SMF: 关联建立响应（原因：MandatoryIEMissing）
```

解决方案：在关联建立请求中包含 RecoveryTimeStamp。

会话建立失败

场景：未建立关联

```
SMF → UPF: 会话建立请求
UPF → SMF: 会话建立响应（原因：NoEstablishedPFCPAssociation）
```

解决方案：在创建会话之前建立 PFCP 关联。

场景：规则创建失败

```
SMF → UPF: 会话建立请求
UPF 成功处理 FAR、QER、URR
UPF 无法创建 PDR（eBPF 映射已满）
UPF → SMF: 会话建立响应（原因：RuleCreationModificationFailure）
```

解决方案：

- 检查 eBPF 映射容量（参见容量监控）
- 在 UPF 配置中增加映射大小
- 减少活动会话数量

场景：缺少 F-SEID

```
SMF → UPF: 会话建立请求（无 CP F-SEID）
UPF → SMF: 会话建立响应（原因：MandatoryIEMissing）
```

解决方案：在会话建立请求中包含 CP F-SEID。

会话修改失败

场景：未知 SEID

```
SMF → UPF: 会话修改请求（SEID=12345）
UPF 没有 SEID 12345 的会话
UPF → SMF: 会话修改响应（原因：SessionContextNotFound）
```

解决方案：

- 验证 SEID 是否与会话建立响应中的值匹配
- 检查会话是否已被删除
- 确保使用正确的 UPF 实例（N9 回环场景）

会话删除失败

场景：未知 SEID

```
SMF → UPF: 会话删除请求（SEID=67890）
UPF 没有 SEID 67890 的会话
UPF → SMF: 会话删除响应（原因：SessionContextNotFound）
```

解决方案：SEID 可能已经被删除或从未存在。

使用原因代码进行故障排除

使用 **Prometheus** 指标

查询 Prometheus 以识别错误模式:

```
# 按原因代码的错误率
rate(upf_pfcp_rx_errors{cause_code!="RequestAccepted"}[5m])

# 主要拒绝原因
topk(5, sum by (cause_code) (upf_pfcp_rx_errors))

# 按 SMF 对等体的错误
sum by (peer_address, cause_code)
(upf_pfcp_rx_errors{cause_code!="RequestAccepted"})

# 会话建立失败
upf_pfcp_rx_errors{message_name="SessionEstablishmentRequest",
cause_code!="RequestAccepted"}
```

使用 **Web UI**

导航到 会话 页面查看:

- 每个控制平面节点的活动会话数量
- 会话建立的成功/失败率
- 最近的会话错误

导航到 容量 页面进行诊断:

- eBPF 映射利用率（RuleCreationModificationFailure 根本原因）
- 资源耗尽指标

请参见 Web UI 指南 获取详细的监控说明。

常见调试步骤

高 **MandatoryIEMissing** 率:

1. 检查 SMF 配置中的必需 IE
2. 验证 PFCP 库版本兼容性
3. 查看 SMF 日志以获取 IE 构造错误

频繁的 **RuleCreationModificationFailure**：

1. 检查 eBPF 映射容量：`GET /api/v1/map_info`
2. 监控映射使用情况：`upf_ebpf_map_used / upf_ebpf_map_capacity`
3. 如果使用率超过 70%，则在配置中增加映射大小
4. 参见 容量规划

NoEstablishedPFCPAssociation 错误：

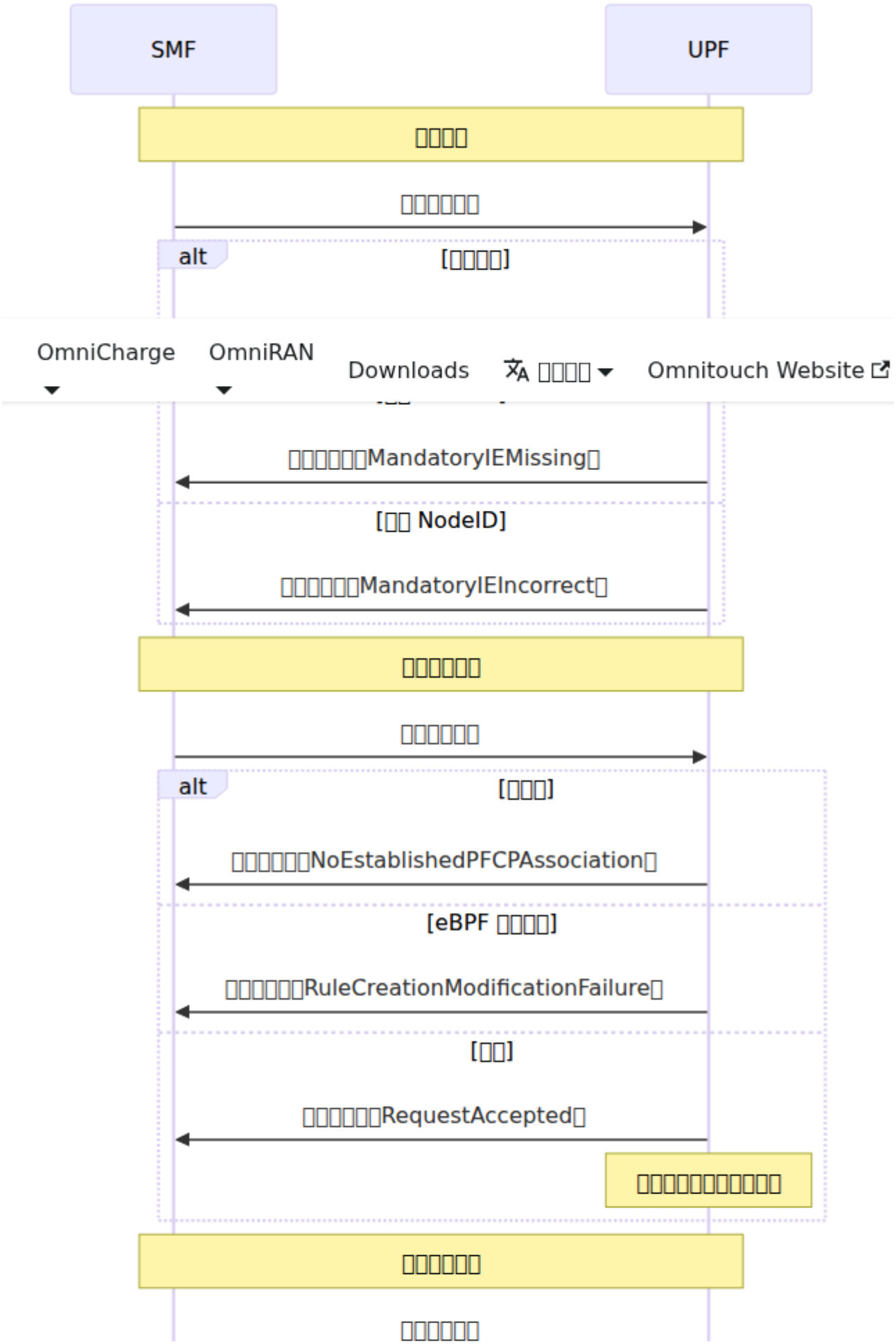
1. 验证关联是否存在：`GET /api/v1/pfcp_associations`
2. 检查心跳超时配置
3. 查看关联建立日志
4. 确保 SMF 和 UPF 之间可以互相访问

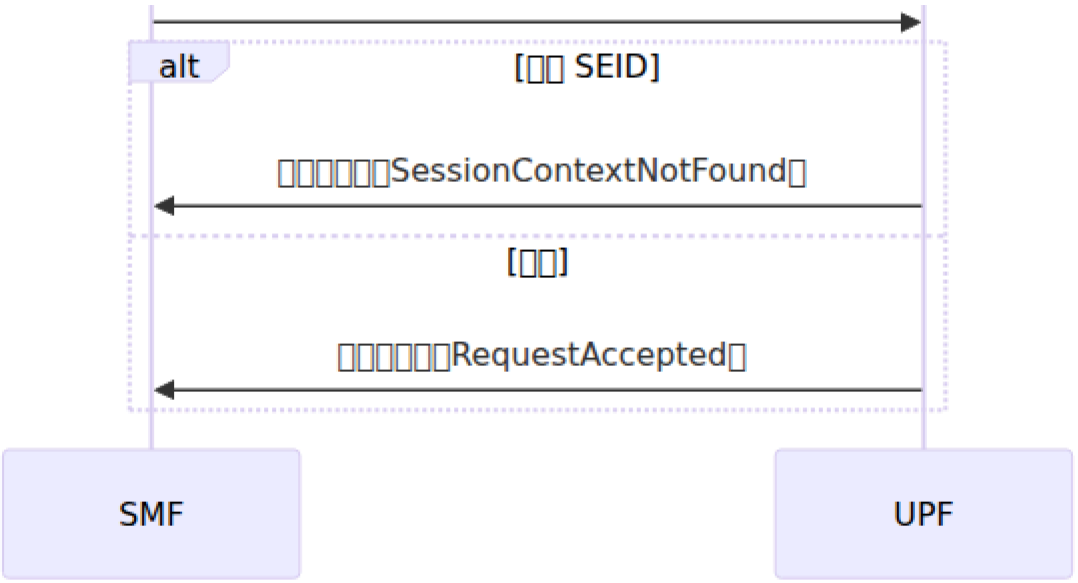
会话修改时的 **SessionContextNotFound**：

1. 验证会话建立响应中的 SEID
2. 检查会话是否已被删除
3. 对于 N9 回环：确保使用正确的 UPF 端点
4. 查询活动会话：`GET /api/v1/pfcp_sessions`

原因代码对操作的影响

会话生命周期


SMF
UPF
□□□□
□□□□□□
alt
[□□□□]
OmniCharge
OmniRAN
Downloads
□□□□
Omnitouch Website
□□□□□□MandatoryIEMissing□
[□□ NodeID]
□□□□□□MandatoryIEIncorrect□
□□□□□□
□□□□□□
alt
[□□□]
□□□□□□NoEstablishedPFCPAssociation□
[eBPF □□□□]
□□□□□□RuleCreationModificationFailure□
[□□]
□□□□□□RequestAccepted□
□□□□□□□□□□□□
□□□□□□
□□□□□□



指标和警报

推荐警报：

```
# 严重：高拒绝率
- alert: PfcpHighRejectionRate
  expr: |
    rate(upf_pfcp_rx_errors{cause_code!="RequestAccepted"}[5m]) > 0.1
  annotations:
    summary: "高 PFCP 拒绝率：{{ $value }}/s"

# 警告：容量问题
- alert: PfcpRuleCreationFailures
  expr: |

rate(upf_pfcp_rx_errors{cause_code="RuleCreationModificationFailure"}
[5m]) > 0
  annotations:
    summary: "检测到 PFCP 规则创建失败"

# 警告：关联问题
- alert: PfcpNoAssociation
  expr: |

rate(upf_pfcp_rx_errors{cause_code="NoEstablishedPFCPAssociation"}
[5m]) > 0
  annotations:
    summary: "在没有关联的情况下尝试 PFCP 会话"
```

3GPP 标准合规性

OmniUPF 根据以下内容实现原因代码：

- **3GPP TS 129.244 v16.4.0** - PFCP 规范
- 第 **8.2.1** 节 - 原因 IE 定义
- 第 **8.19** 节 - 原因值表

相关□档

- **PFCP** 协议集成 - PFCP 架构和消息处理
- 指标参考 - upf_pfcp_rx_errors 指标文档
- 监控指南 - 容量监控和警报
- 故障排除指南 - PFCP 关联和会话问题
- **Web UI** 指南 - 会话和关联监控

UE 路由管理

> 相关□档:
>
> - API 文档 - 包括路由管理端点的完整 API 参考
> - 操作指南 - Web UI 操作和监控

概述

UPF（用户平面功能）与 **FRR**（自由范围路由）集成，以动态管理用户设备（UE）IP 路由。此集成确保在 UE 会话建立或终止时，路由基础设施会自动适应以反映当前网络拓扑。

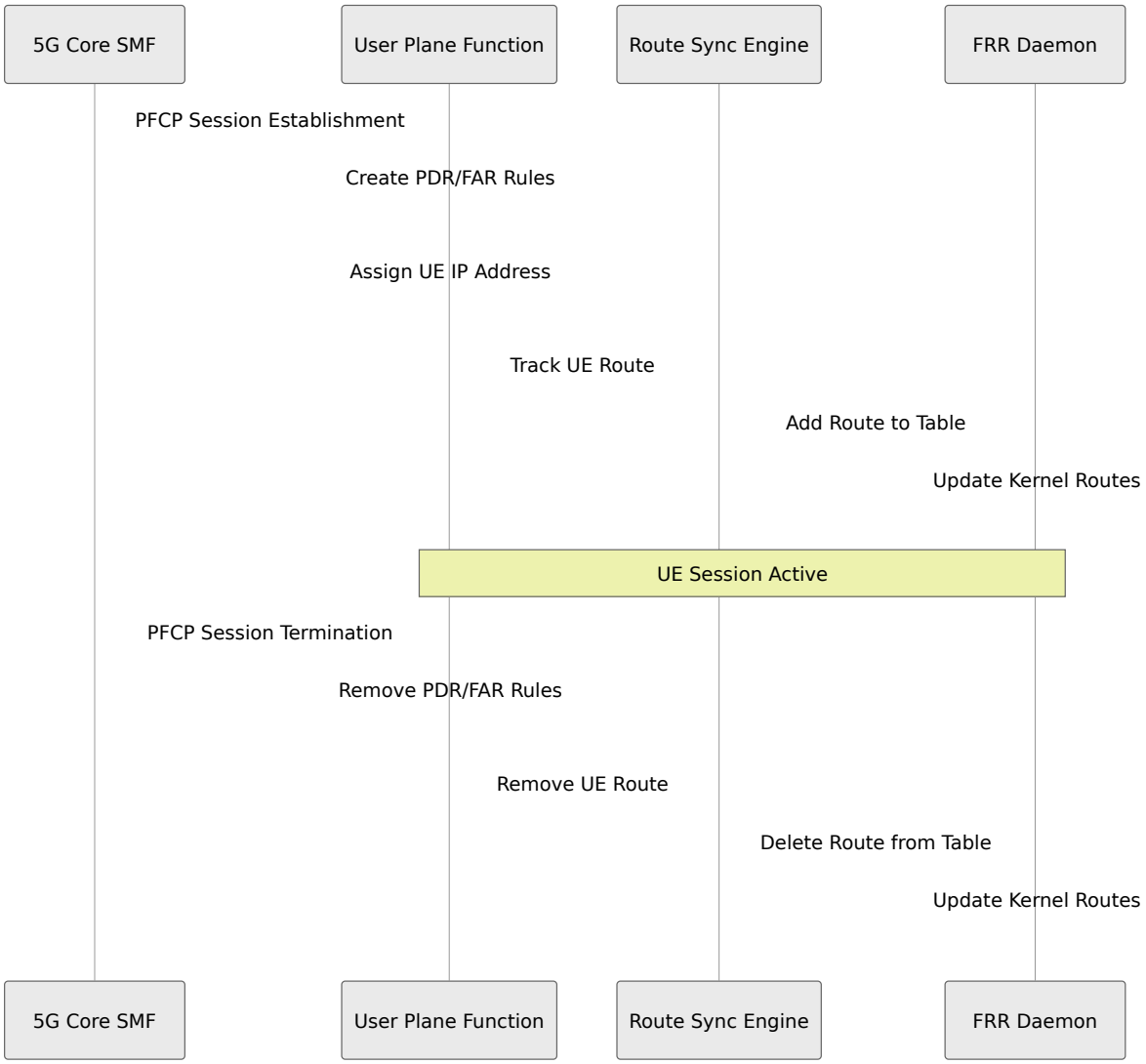
什么是 FRR?

FRR（自由范围路由）是一个强大的开源路由协议套件，适用于 Linux 和 Unix 平台。它实现了多种路由协议，包括 BGP、OSPF、RIP 等。在我们的部署中，FRR 作为路由守护进程，维护内核路由表，并可以将路由重新分发到其他网络元素。

架构

路由同步工作原理

路由生命周期



自动同步

UPF 维护所有活动 UE IP 地址的内部注册表。当启用时，路由同步系统：

1. 监控 **UE** 会话：跟踪所有活动的 PFCP 会话及其相关的 UE IP 地址
2. 维护路由列表：保持需要在路由表中的路由的最新列表
3. 同步到 **FRR**：通过其 API 自动推送路由更新到 FRR 守护进程
4. 处理故障：跟踪每条路由的同步状态（已同步/失败）并根据需要重试

FRR 设置

配置

FRR 使用 **Ansible** 模板 部署和配置，以建立基本路由参数。您在 Ansible 剧本中定义 FRR 配置一次作为 **Jinja2** 模板，Ansible 会在部署期间自动将其传播到所有 UPF 实例。

典型的 FRR Jinja2 配置模板包括：

```
frr version 7.2.1
frr defaults traditional
hostname pgw02
log syslog informational
service integrated-vtysh-config
!
ip route {{ hostvars[inventory_hostname]['ansible_default_ipv4']
['gateway'] }}/32 {{ ansible_default_ipv4['interface'] }}
!
interface {{ ansible_default_ipv4['interface'] }}
 ip address ospf router-id {{hostvars[inventory_hostname]
['ansible_host']}}
 ip ospf authentication null
!
router ospf
 ospf router-id {{hostvars[inventory_hostname]['ansible_host']}}
 redistribute kernel
 network {{ hostvars[inventory_hostname]['ansible_default_ipv4']
['network'] }}/{{ mask_cidr }} area 0
 area 0 authentication message-digest
!
line vty
!
end
```

部署模型：

1. 一次定义：在您的 Ansible 角色中创建 FRR Jinja2 模板（例如，`roles/frr/templates/frr.conf.j2`）
2. 配置参数：为每个 UPF 主机在 Ansible 清单中设置变量
3. 全面部署：运行 Ansible 剧本将 FRR 配置部署到所有 UPF 节点

4. 自动定制：Ansible 使用特定于主机的变量（IP 地址、路由器 ID 等）定制每个 UPF 的 FRR 配置

Jinja2 模板中的可定制参数：

- **OSPF** 参数：路由器 ID、区域配置、认证方法、网络广告
- **BGP** 配置：ASN、邻居关系、路由策略、社区
- 路由重新分发：要重新分发的内核路由（例如，`redistribute kernel`）
- 路由过滤：路由映射、前缀列表、访问列表
- 接口设置：OSPF/BGP 接口参数

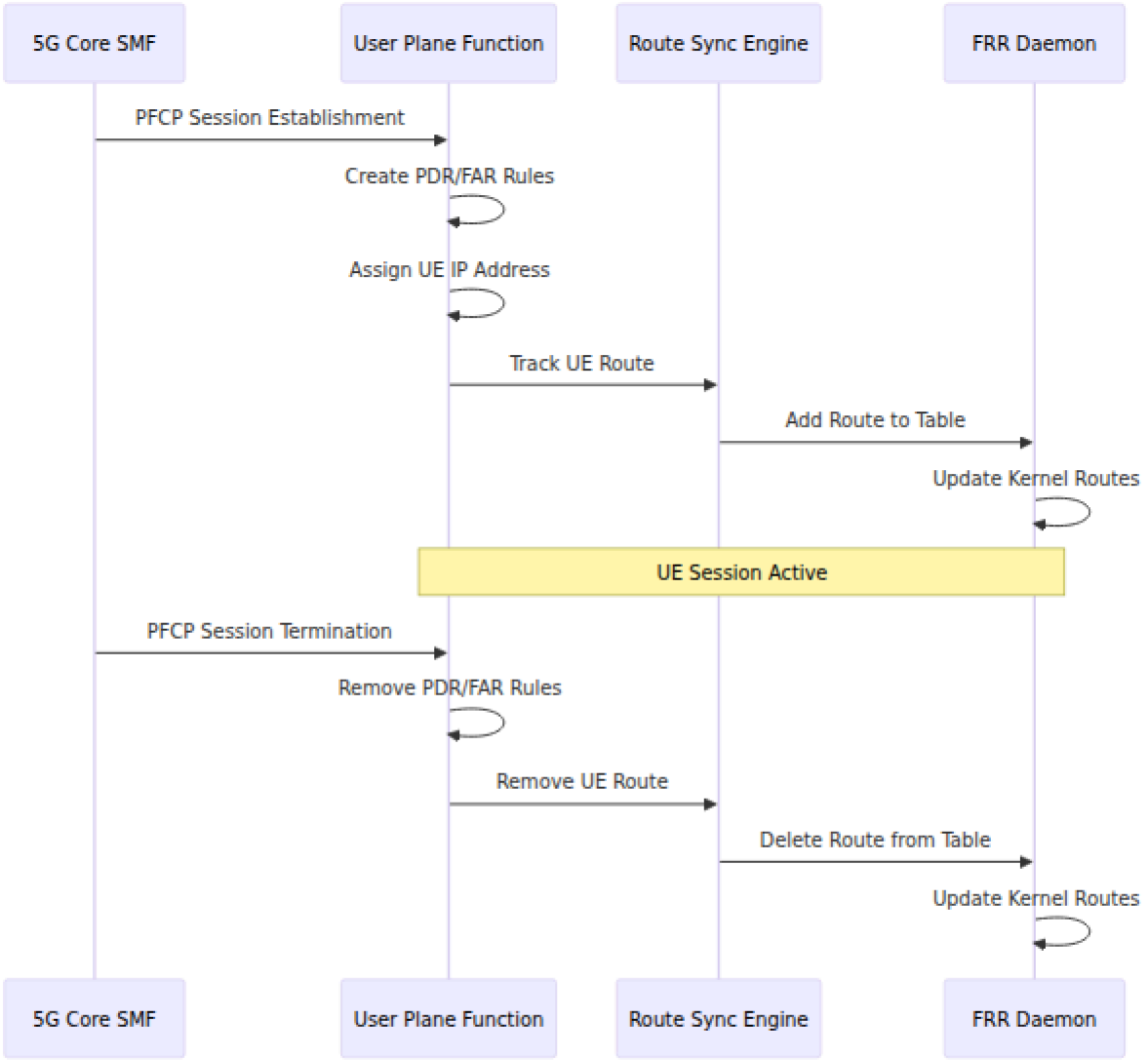
UPF 集成：一旦将基本 FRR 配置部署到每个 UPF 实例，UPF 会根据活动的 PFCP 会话动态将 UE IP 地址作为 **/32** 主机路由 添加到内核路由表中。这些路由随后：

1. 由 **UPF** 路由同步引擎安装到内核路由表中
2. 通过 **`redistribute kernel`** 指令被 **FRR** 拾取
3. 根据您的 **FRR** 配置被路由协议（**OSPF**、**BGP**）通告
4. 传播到网络中，以便 **UE** 流量可以路由到该 **UPF** 实例

关键点：

- 一次设置，处处部署：在 Ansible 中定义 FRR Jinja2 模板一次，它会自动部署到所有 UPF 实例
- **Ansible** 处理静态配置：Jinja2 模板设置所有路由协议参数（OSPF 区域、BGP 邻居、认证、路由策略等）
- **UPF** 处理动态路由：每个 UPF 实例根据其活动的 PFCP 会话动态管理仅 UE IP /32 路由
- 自动路由通告：每个 UPF 上的 FRR 根据您配置的策略自动重新分发本地 UE 路由
- 集中管理：更新 Ansible 模板并重新运行剧本以同时更改所有 UPF 的路由配置

路由通告



监控与管理

Web UI 集成

UPF 控制面板□供一个 路由 页面，显示：

- 路由状态：路由同步是否启用或禁用
- 总路由数：正在跟踪的 UE IP 地址数量
- 同步统计：成功同步的路由计数及任何失败
- 活动路由：当前在路由表中的所有 UE IP 地址的实时列表
- **OSPF** 邻居：OSPF 邻接状态及邻居详细信息

- **BGP** 对等体：BGP 会话状态和前缀统计（当配置时）
- **OSPF** 重新分发的路由：显示 UE 路由如何被通告的外部 LSA 的完整视图

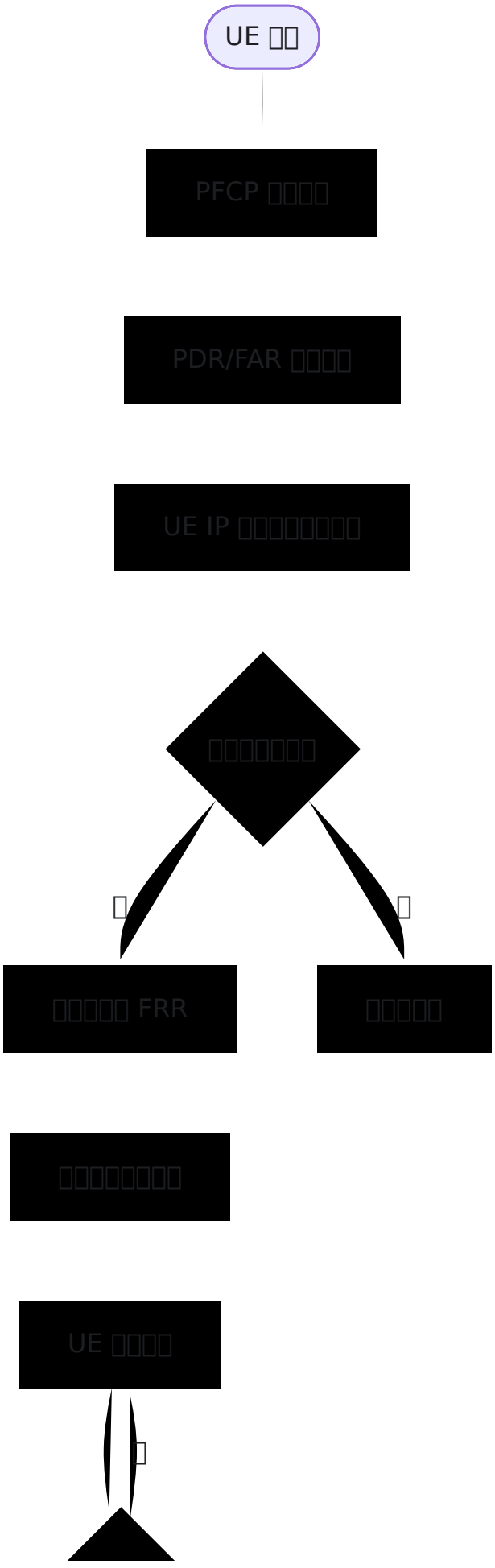
□□□□□□□□ UE □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

手动同步操作

管理员可以通过 Web UI 使用 同步路由 按钮触发手动路由同步。此操作：

1. 重新读取 UPF 中当前活动 UE 会话的列表
2. 与 FRR 的路由表进行比较
3. 添加任何缺失的路由
4. 删除任何过时的路由
5. 返回更新的同步统计信息

□□□□


UE □□
PFCP □□□□
PDR/FAR □□□□
UE IP □□□□□□□□
□□□□□□□
□
□
□□□□□ FRR
□□□□□
□□□□□□□□
UE □□□□
□

优势

- 零接触配置：路由自动管理，无需人工干预
- 动态适应：网络路由实时适应 UE 移动和会话变化
- 可扩展性：支持数千个并发 UE 路由
- 弹性：跟踪失败的同步操作并可以重试
- 可见性：通过 Web UI 完全可见路由状态

技术细节

API 端点

UPF 暴露以下路由管理端点：

- `GET /api/v1/routes` - 列出所有跟踪的 UE 路由而不进行同步
- `POST /api/v1/routes/sync` - 将路由同步到 FRR 并返回更新的列表
- `GET /api/v1/route_stats` - 获取详细的路由统计信息
- `GET /api/v1/routing/sessions` - 获取路由协议会话（OSPF 邻居、BGP 对等体）
- `GET /api/v1/ospf/database/external` - 获取 OSPF AS-External LSA 数据库（重新分发的路由）

> 另见：API 文档 - 路由管理 以获���完整的端点详细信息和示例

路由格式

路由以简单的 IP 地址（例如，`100.64.18.5`）存储和管理。路由守护进程处理完整的路由条目详细信息，包括：

- 目标前缀/掩码
- 网关/下一跳
- 接口绑定
- 指标和管理距离

FRR 验证

OSPF 外部 LSA 数据库

您可以通过检查 FRR OSPF 链路状态数据库来验证 UE 路由是否被正确重新分发到 OSPF。外部 LSA（类型 5）显示已从外部源注入到 OSPF 的路由。

FRR OSPF □□□□□□□ LSA□□□ UE □□ 100.64.18.5/32 □□□□ E2□□□□□ 2□□□□

在上面的示例中，您可以看到：

- 网络 **LSA (10.98.0.20)**：UPF 自己的网络广告
- 路由器 **LSA (192.168.1.1)**：OSPF 路由器广告
- 外部 **LSA**：包括 UE 路由 `100.64.18.5` 以 E2（外部类型 2）指标重新分发到 OSPF

此验证确认：

1. UPF 正在成功跟踪 UE IP 地址

2. 路由同步引擎已将路由推送到 FRR
3. FRR 已将路由重新分发到 OSPF
4. OSPF 邻居正在接收路由广告

规则管理指南

目录

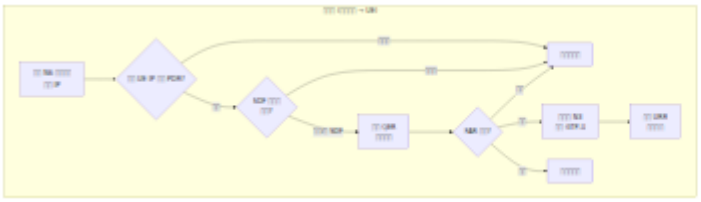
概述

OmniUPF 使用一组相互关联的规则来分类、转发、整形和跟踪用户平面流量。这些规则通过 SMF 通过 PFCP 安装，并存储在 eBPF 映射中，以实现高性能的数据包处理。理解这些规则及其关系对于操作和故障排除 UPF 至关重要。

规则类型

规则类型	目的	关键字段	安装者
PDR（数据包检测规则）	将数据包分类为流	TEID 或 UE IP	SMF 通过 PFCP 会话建立/修改
FAR（转发动作规则）	确定转发动作	FAR ID	SMF 通过 PFCP 会话建立/修改
QER（QoS 强制规则）	应用带宽限制和标记	QER ID	SMF 通过 PFCP 会话建立/修改
URR（使用报告规则）	跟踪计费的数据量	URR ID	SMF 通过 PFCP 会话建立/修改

规则处理流程

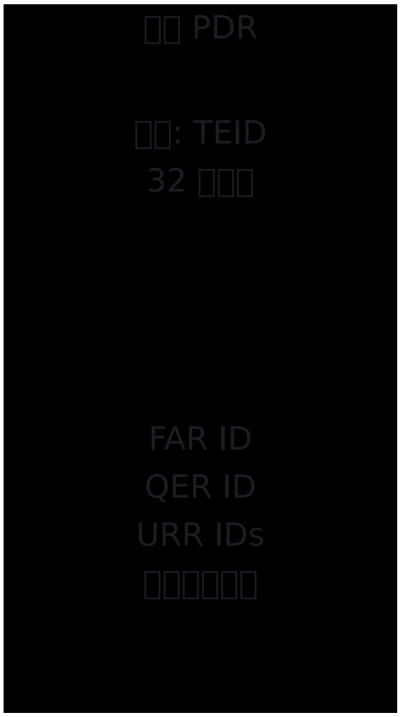


数据包检测规则（PDR）

目的

PDR 将传入的数据包分类为流。它们是 UPF 中所有数据包处理的入口点。

PDR 结构



上行 PDR

上行 PDR 匹配来自 RAN 的 N3 接口上到达的数据包。

关键字段：TEID（隧道端点标识符）

- 32 位无符号整数
- 由 SMF 分配并信号传递给 gNB
- 每个 UE 流量流唯一

值字段：

- **FAR ID**：转发动作规则的引用
- **QER ID**：QoS 强制规则的引用（可选）
- **URR IDs**：使用报告规则的列表（可选）
- 外部头部移除：移除 GTP-U 封装的标志

查找过程：

1. 从 GTP-U 头部提取 TEID
2. 在 `uplink_pdr_map` eBPF 映射中进行哈希查找
3. 如果找到匹配项，检索 FAR ID、QER ID 和 URR IDs
4. 如果没有匹配项，丢弃数据包

示例：

```
TEID: 5678
FAR ID: 2
QER ID: 1
外部头部移除: False
SDF 模式: No SDF
```

下行 PDR

下行 PDR 匹配来自数据网络的 N6 接口上到达的数据包。

关键字段: UE IP 地址

- IPv4 地址 (32 位) 或 IPv6 地址 (128 位)
- 在 PDU 会话建立期间由 SMF 分配
- 每个 UE 唯一

值字段:

- **FAR ID**: 转发动作规则的引用
- **QER ID**: QoS 强制规则的引用（可选）
- **URR IDs**: 使用报告规则的列表（可选）
- **SDF** 模式: 服务数据流过滤器模式
    - `No SDF`: 无过滤，所有流量匹配
    - `SDF Only`: 仅转发 SDF 匹配的流量

    - `SDF + Default`: SDF 匹配的流量使用特定规则，其他流量使用默认 FAR
- **SDF** 过滤器：应用特定的过滤器（端口、协议、IP 范围）

查找过程：

1. 从数据包头部□取目标 IP
2. 在 `downlink_pdr_map` (IPv4) 或 `downlink_pdr_map_ip6` (IPv6) 中进行哈希查找
3. 如果找到匹配项，检查 SDF 过滤器（如果已配置）
4. 检索 FAR ID、QER ID 和 URR IDs
5. 如果没有匹配项，丢弃数据包

示例：

```
UE IP: 10.45.0.1
FAR ID: 1
QER ID: 1
外部头部移除: False
SDF 模式: No SDF
```

SDF 过滤器（服务数据流）

SDF 过滤器供应用特定的流量分类。

用例：

- 区分 YouTube 流量与网页浏览
- 对 VoIP 与最佳努力数据应用不同的 QoS
- 通过不同的网络路径路由特定应用程序

过滤标准：

- 协议：TCP、UDP、ICMP
- 端口范围：目标端口（例如，HTTPS 的 443，SIP 的 5060）
- **IP** 地址范围：特定目标网络
- 流描述：3GPP 定义的流模板

示例 **SDF** 配置：

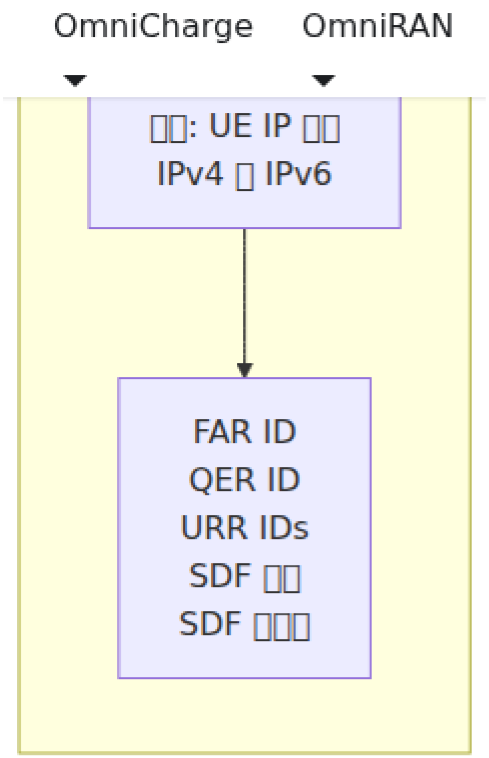
```
PDR ID: 10
UE IP: 10.45.0.1
SDF 模式: SDF Only
SDF 过滤器:
  - 协议: UDP, 端口: 5060-5061 → FAR ID 5 (VoIP FAR)
  - 协议: TCP, 端口: 443 → FAR ID 1 (默认 FAR)
```

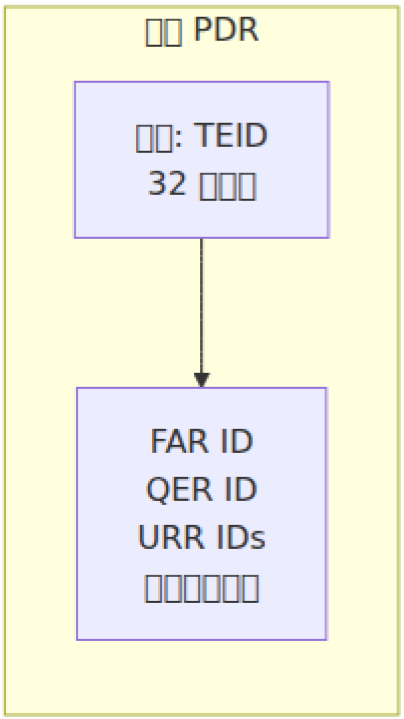
转发动作规则（FAR）

目的

FAR 确定对匹配 PDR 的数据包的处理方式。它们定义转发动作、GTP-U 封装参数和目标端点。

FAR 结构





动作标志

FAR 动作是可以组合的按位标志:

□□	□	□	□□
FORWARD	1	2	□□□□□□□□□□
BUFFER	2	4	□□□□□□□□□□□
DROP	0	1	□□□□□
NOTIFY	3	8	□□□□□□□□□
DUPLICATE	4	16	□□□□□□□□□□□□

□□□□□□:

- `□□: 2 (FORWARD)` - □□□□□□□□□
- `□□: 6 (FORWARD + BUFFER)` - □□□□□□□□□□
- `□□: 4 (BUFFER)` - □□□□□□□□□□□
- `□□: 1 (DROP)` - □□□□□□□□□□□□□□□□□□□

□□□

BUFFER □□□□ 2□□□□□□□□□□□□□□□□□□□□□□□□ UPF □□□□□□□□□□□□□□ UE □□□□□□□□□□□□

□□□□□□□

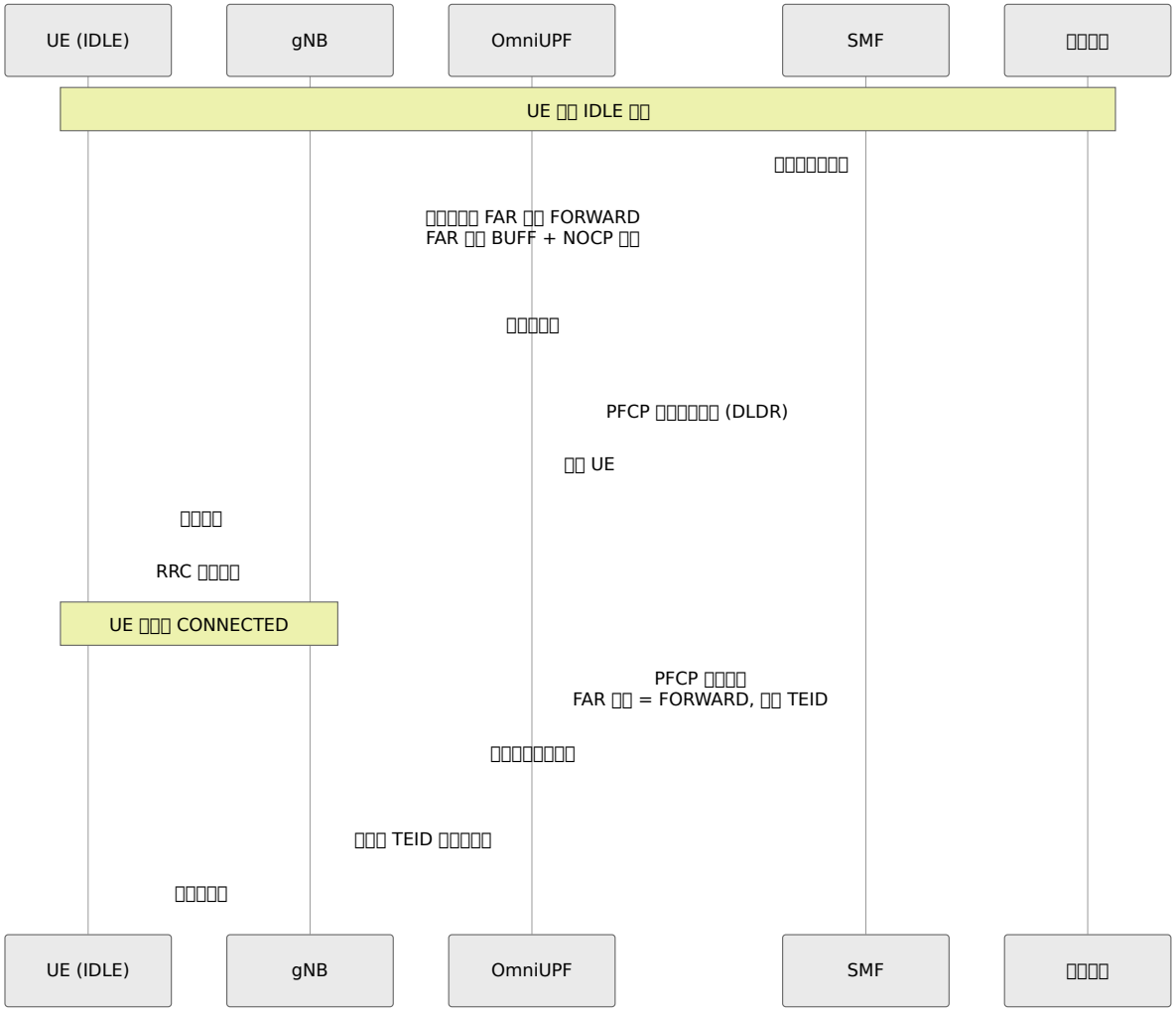
□□□□□□□□: □□□□□□□□□□ IDLE □□□□□□□ gNB□□ UE □□UPF□

1. □□□□□
2. □ SMF □□□□□□□□ (DLDR)
3. SMF □□ AMF □ UE □□□□□□□□□□
4. □□□□□SMF □□ FAR □ FORWARD □□
5. UPF □□□□□□□□□□□□ UE

□□□□□□□□□: □ gNB □ gNB □□□□□□□UPF □□□□□□□□□□□□□□

1. □□ gNB □□□□□
2. SMF □ FAR □□□□□ BUFFER

3. □□□□□□□□□□□□□□□□
4. UE □□□□□□ gNB
5. SMF □□ FAR□□□□□□ TEID □ FORWARD □□
6. UPF □□□□□□□□□□□ gNB



□□□□□□□□

□□□□□□□:

- □□□□□□□□: 100,000□□□□□
- □□□□□□□: □□□□□□□
- **TTL (□□□□)**: 60 □□□□□□□
- □□ **TTL** □□□□: □□□□

□□ **FAR** □□:

- □□ **FAR** □□□□□□□□: 10,000□□□□□

- 目的：防止单个 FAR 耗尽缓冲容量

缓冲溢出行为：

- 当达到全局或每个 FAR 限制时，新数据包将被丢弃
- 指标跟踪丢弃，原因为 `reason="global_limit"` 或 `reason="far_limit"`
- 最旧的数据包不会自动驱逐（仅在 TTL 到期时显式丢弃）

下行数据通知 **(DLDR)**

当 UPF 为处于 IDLE 状态的 UE 缓冲数据包时，它会向 SMF 发送 PFCP 会话报告请求：

DLDR 内容：

- 报告类型：下行数据报告 (DLDR)
- **FAR ID**：触发缓冲的 FAR
- 下行数据服务信息：可选的 QFI、寻呼策略指示符

SMF 对 **DLDR** 的操作：

1. 通过 AMF → gNB 寻呼 UE
2. 等待 UE 建立 RRC 连接
3. 发送 PFCP 会话修改请求以更新 FAR
4. FAR 动作从 `BUFF+NOCP` 更改为 `FORW`
5. UPF 刷新缓冲的数据包

DLDR 的指标：

- `upf_dldr_sent_total`：发送的 DLDR 总数
- `upf_dldr_send_errors`：失败的 DLDR
- `upf_buffer_notify_to_flush_duration_seconds`：从 DLDR 到刷新之间的延迟

请参见 指标参考 以获取完整列表。

缓冲操作

启用缓冲（设置 BUFF 标志）：

- FAR 动作 `|= 0x04`（设置位 2）

- □□: `□□: 2 (FORW)` → `□□: 6 (FORW+BUFF)`
- □□□□□□□□□□

□□□□□□BUFF □ FORW□:

- FAR □□ `= 0x04`□□ BUFF□
- □□□□□□□□□□
- □□ IDLE UE □□□□□□□

□□□□□□□ BUFF □□□:

- FAR □□ `&= ~0x04`□□□□ 2□
- □□: `□□: 6 (FORW+BUFF)` → `□□: 2 (FORW)`
- □□□□□□□□□□□□□□□□□□□□

□□□□:

- □□ □□ FAR □□□□□□□□□□□□
- □□□□□□□□□ TEID/□□□□□□
- □□□□□□□□□□
- FAR □□□□ FORW □□

□□□□:

- □□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□
- □□□□ `reason="cleared"`

□□□□□□□□

□□□□□Web UI□: □□□ □□ □□□□

- □□□□□□
- □□□□□
- □□□□□□□□ FAR □□
- □□ FAR □□□□□□
- □□□□□□□□□
- □□/□□□□ FAR □□□

- 刷新或清除操作

关键指标：

- 数据包 **> 10** 秒：潜在的寻呼延迟
- 数据包 **> 30** 秒：可能的寻呼失败，清除缓冲
- 高数据包计数：检查是否有卡住的会话或寻呼失败

Prometheus 指标：

- `upf_buffer_packets_current`：当前缓冲的数据包
- `upf_buffer_bytes_current`：当前缓冲的字节
- `upf_buffer_fars_active`：具有缓冲数据包的 FAR
- `upf_buffer_packets_dropped{reason}`：丢弃的数据包计数

请参见 指标参考 以获取完整的缓冲指标。

常见缓冲场景

场景 **1: IDLE UE** 下行数据

```
初始状态：
- UE 处于 IDLE 模式（没有 gNB 连接）
- FAR 动作: 0x04 (仅 BUFF)

数据到达：
1. DN 发送下行数据包
2. UPF 匹配 PDR，应用 FAR
3. FAR 具有 BUFF 标志 → 数据包被缓冲
4. UPF 向 SMF 发送 DLDR
5. SMF 寻呼 UE
6. UE 连接到 gNB
7. SMF 修改 FAR: 动作 = 0x02 (FORW)
8. UPF 使用新 TEID 刷新缓冲的数据包
```

场景 **2:** 切换准备

□□□□:
- UE □□□ gNB-1 (TEID 1234)
- FAR □□: 0x02 (FORW)

□□□□:
1. SMF □□ FAR: □□ = 0x06 (FORW+BUFF)
2. □□□□□□ gNB-1 □□□
3. UE □□□ gNB-2
4. SMF □□ FAR: TEID = 5678, □□ = 0x02 (FORW)
5. UPF □□□□□□□□□□□□ gNB-2□□□□ TEID
6. □□□□□□□□□□□□

□□ 3: □□□□

□□□□:
- UE □□□□□□□□

□□□□:
1. SMF □□ FAR: □□ = 0x04 (□ BUFF)
2. □□□□□□□□□□□□□□□□□□
3. □□□□□□□□
4. SMF □□ FAR: □□ = 0x02 (FORW)□□□□
5. UPF □□□□□□□□□□□□□□□

外部头部创建

确定是否应添加 GTP-U 封装。

上行 **FAR** (N3 → N6):

- 外部头部创建: False
- 动作: 移除 GTP-U, 转发原生 IP 数据包

下行 **FAR** (N6 → N3):

- 外部头部创建: True
- 远程 IP: gNB IP 地址（例如, 200.198.5.10）
- TEID: UE 流量的隧道 ID
- 动作: 添加 GTP-U 头部, 转发到 gNB

Web UI 中的 **FAR** 查找

规则管理页面□供按 ID 查找 FAR 的功能:

步骤：

1. 导航到 规则 → FARs 选项卡
2. 在搜索字段中输入 FAR ID
3. 点击 "查找" 查看 FAR 详细信息

显示的信息：

- FAR ID
- 动作（数字 + 解码标志）
- 缓冲状态（开启/关闭）
- 外部头部创建
- 远程 IP 地址（带整数表示）
- TEID
- 传输级标记

QoS 强制规则（QER）

目的

QER 将服务质量参数应用于流量流，包括带宽限制和数据包标记。

QER □□


QER □□
QFI
QoS □□□□
□□□□□
□□/□□
□□□□□
□□/□□
QER ID
□□□□□
□□ MBR
□□□□□
□□ MBR
□□□□□
□□ GBR
□□□□□
□□ GBR
□□□□□

QoS 参数

QFI (QoS 流标识符):

- 6 位标识符，用于 5G QoS 流
- 值 1-9 是标准化的（例如，QFI 9 = 默认承载）
- 用于 5GC 中的数据包标记

门状态:

- 开启 **(0)**: 允许流量
- 关闭 **(非零)**: 阻止流量

最大比特率 **(MBR)**:

- 流量流的最大允许带宽
- 以 kbps 为单位指定
- **MBR = 0**: 无速率限制（无限制）
- 超过 MBR 的流量将被丢弃

保证比特率 **(GBR)**:

- 为流量流保证的最小带宽
- 以 kbps 为单位指定
- **GBR = 0**: 最佳努力（无保证）
- **GBR > 0**: 优先流，具有保证带宽

QoS 流类型

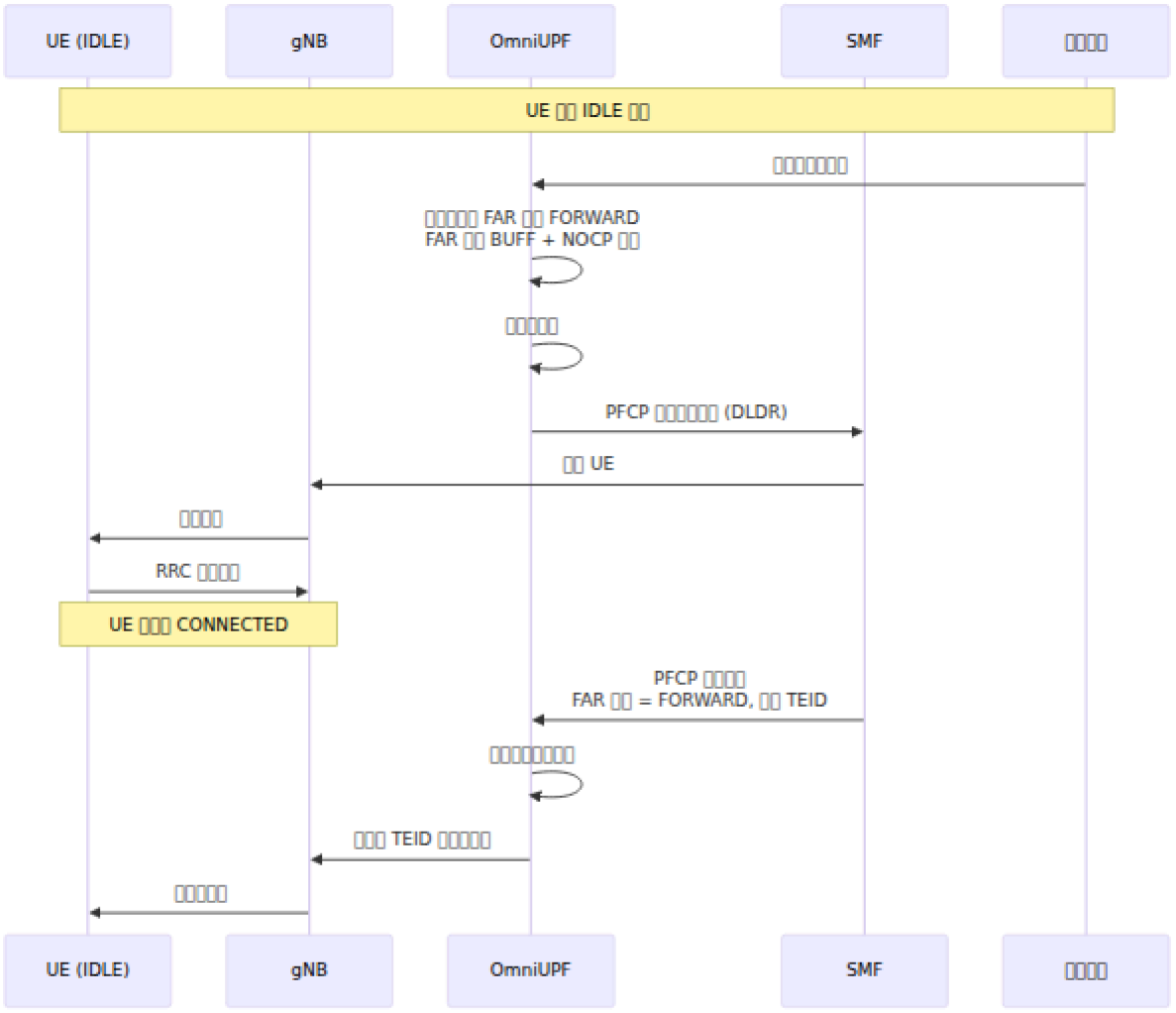
最佳努力流 (GBR = 0):

```
QER ID: 1
QFI: 9
上行 MBR: 100000 kbps (100 Mbps)
下行 MBR: 100000 kbps (100 Mbps)
上行 GBR: 0 kbps
下行 GBR: 0 kbps
```

保证流 (GBR > 0):

```
QER ID: 2
QFI: 1
上行 MBR: 10000 kbps (10 Mbps)
下行 MBR: 10000 kbps (10 Mbps)
上行 GBR: 5000 kbps (5 Mbps)
下行 GBR: 5000 kbps (5 Mbps)
```

QoS 强制算法



MBR 强制机制

OmniUPF 通过在 eBPF 数据路径中实现的 滑动窗口速率限制器 强制执行 MBR（最大比特率）限制。该算法在纳秒精度下直接在 XDP 层操作，确保在没有内核上下文切换的情况下实现线速性能。

工作原理

算法：滑动窗口速率限制

对于每个数据包，UPF 执行以下检查：

1. 门状态检查：如果门状态为 `CLOSED`（非零），立即丢弃数据包
2. **MBR** 检查：如果 MBR = 0，绕过速率限制（无限带宽）
3. 传输时间计算：

```
tx_time = (packet_size_bytes × 8) × (1,000,000,000 ns/sec) /
MBR_kbps
```

4. □□□□: □□□□□□□□ 5ms □□□□□□□□□□□□
5. □□□□: □□□□□□□□□□□ `tx_time` □□□□

□□□□:

□□:

- MBR = 100,000 kbps (100 Mbps)
- □□□□□ = 1500 □□
- □□□□ = 5,000,000 ns (5 ms)

```
□□ 1: □ 100 Mbps □□□□□□
tx_time = (1500 □□ × 8 □/□□) × (1,000,000,000 ns/sec) /
100,000,000 bps
        = 12,000,000,000 / 100,000,000
        = 120 ns

□□ 2: □□□□□□□□□□□□
current_time = 1000000000 ns
window_start = 999990000 ns
if (window_start + tx_time > current_time):
    □□□□□□□□□□□□□□□□

□□ 3: □□□□□□□□□□
window_start = window_start + 120 ns
□□□□□
```

□□□□□□

5ms □□□□:

- □□□□□ 5 □□□□□□□□
- □□□□□□□ 5 □□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□

□□□□:

- 在 5ms 窗口内允许小突发
- 持续流量超过 MBR 会受到速率限制
- 比简单的令牌桶算法更准确

每个方向的速率限制：

- 上行 MBR 使用 `qer->ul_start` 时间戳
- 下行 MBR 使用 `qer->dl_start` 时间戳
- 每个方向独立进行速率限制

MBR 与观察到的吞吐量

为什么观察到的吞吐量可能与 **MBR** 不同：

- 协议开销：GTP-U、UDP、IP 头部每个数据包增加 ~50-60 字节
- 数据包大小变化：较小的数据包 = 更多开销，效率更低
- 速率限制精度：强制执行发生在每个数据包，而不是每字节
- 窗口重置行为：5ms 的空闲期允许短暂的突发超过 MBR

示例：

```
配置的 MBR: 100 Mbps
观察到的吞吐量: ~95-98 Mbps（由于 GTP-U/UDP/IP 开销）
```

如何验证速率限制：

1. 检查 URR 流量计数器随时间变化：`upf_urr_*_volume_bytes`
2. 计算吞吐量：`(volume_delta_bytes × 8) / time_delta_seconds / 1000` = kbps
3. 与 QER 中配置的 MBR 进行比较

GBR (保证比特率)

重要：OmniUPF 当前不强制执行 GBR 最低值。GBR 存储在 QER 中，但不用于流量优先级或接纳控制。

GBR 行为：

- GBR 值通过 PFCP 从 SMF 接收
- GBR 存储在 QER 映射中，并通过 API 可见

- 没有带宽预留或基于 GBR 的流量优先级
- GBR 作为跟踪流类型（最佳努力与保证）的元数据

未来增强：

- GBR 强制执行需要流量调度或加权排队
- 可能在未来版本中使用 eBPF QoS 功能实现

使用报告规则（URR）

目的

URR 跟踪计费、分析和策略强制的数据量。它们维护数据包和字节计数，并报告给 SMF 以生成计费记录。

URR 结构

流量跟踪

上行流量：

- 从 UE 发送到数据网络的字节
- 在 GTP-U 解封装后测量
- 包括 IP 头部和有效载荷

下行流量:

- 从数据网络发送到 UE 的字节
- 在 GTP-U 封装前测量
- 包括 IP 头部和有效载荷

总流量:

- 上行和下行流量的总和
- 用于总使用报告

使用报告触发器

URR 可以基于以下条件触发报告:

流量阈值:

- 当流量超过配置的限制时报告
- 示例: 每 1 GB 使用报告一次

时间阈值:

- 在定期间隔报告
- 示例: 每 5 分钟报告一次

事件驱动:

- 在会话终止时报告
- 在 QoS 更改时报告
- 在切换时报告

流量显示格式

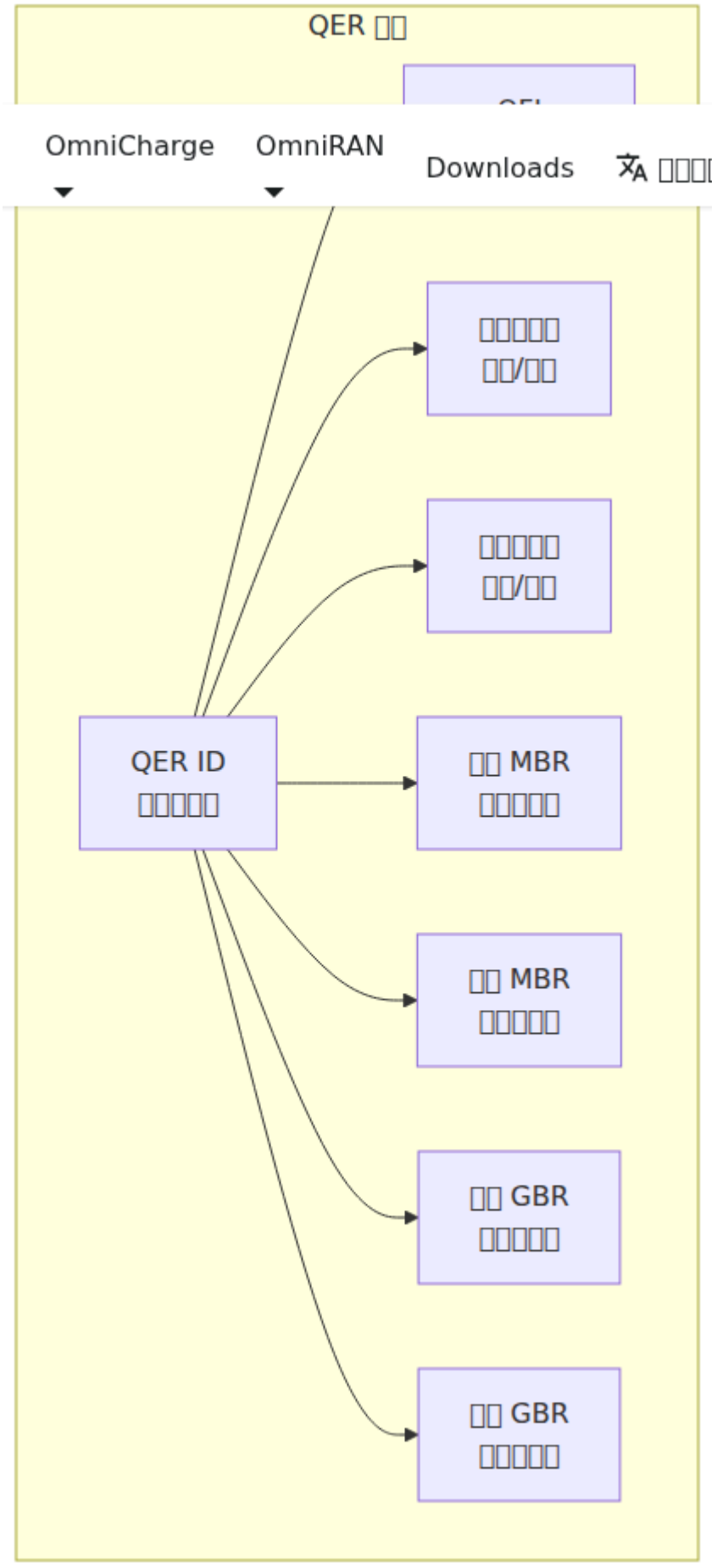
Web UI 自动将流量格式化为人类可读的单位:

字节	显示
0 - 1023	B (字节)
1024 - 1048575	KB (千字节)
1048576 - 1073741823	MB (兆字节)
1073741824 - 1099511627775	GB (千兆字节)
1099511627776+	TB (太字节)

示例：

```
URR ID: 0
上行流量: 12.3 KB
下行流量: 9.0 KB
总流量: 21.3 KB
```

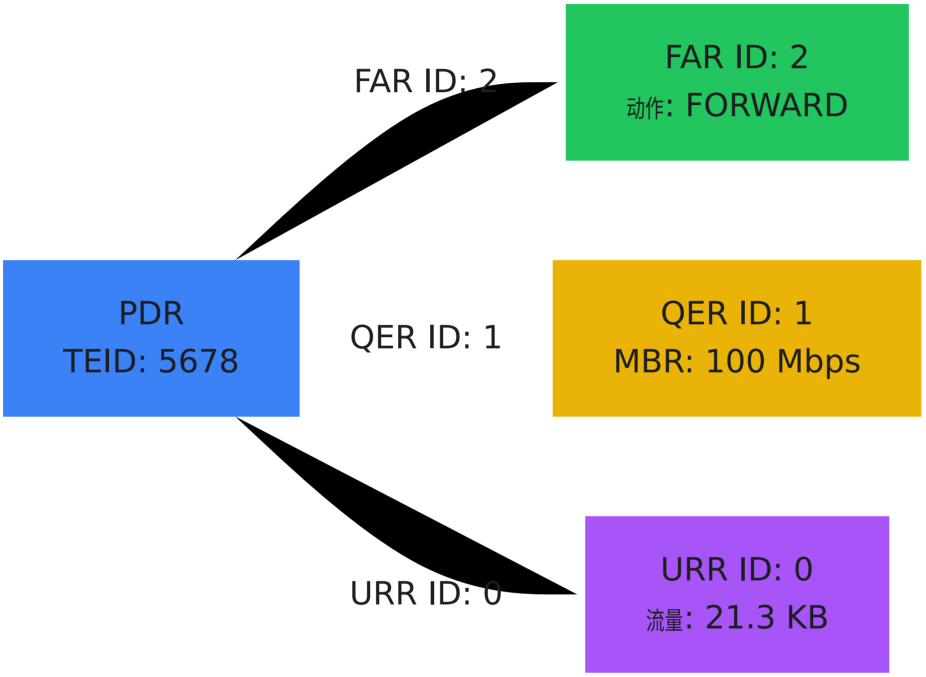
URR □□□□


QER □□
OmniCharge
OmniRAN
Downloads
□□□□□□
□□/□□
□□□□□□
□□/□□
QER ID
□□□□□□
□□ MBR
□□□□□□
□□ MBR
□□□□□□
□□ GBR
□□□□□□
□□ GBR
□□□□□□

规则关系

PDR → FAR → QER → URR 链

每个 PDR 引用一个 FAR，FAR 可能引用一个 QER 和一个或多个 URR。



示例会话配置

上行 **PDR**:

```
TEID: 5678
FAR ID: 2
QER ID: 1
URR IDs: [0]
外部头部移除: False
```

下行 **PDR**:

```
UE IP: 10.45.0.1
FAR ID: 1
QER ID: 1
URR IDs: [0]
SDF 模式: No SDF
```

FAR ID 1 (下行):

```
动作: 2 (FORWARD)
外部头部创建: True
远程 IP: 200.198.5.10
TEID: 5678
```

FAR ID 2 (上行):

```
动作: 2 (FORWARD)
外部头部创建: False
```

QER ID 1:

```
QFI: 9
上行 MBR: 100000 kbps
下行 MBR: 100000 kbps
上行 GBR: 0 kbps
下行 GBR: 0 kbps
```

URR ID 0:

```
上行流量: 12.3 KB
下行流量: 9.0 KB
总流量: 21.3 KB
```

常见操作

查看会话的规则

通过会话页面：

1. 导航到会话
2. 通过 IP 或 TEID 查找 UE
3. 点击 "展开" 查看所有规则 (PDR, FAR, QER, URR)

通过规则页面：

1. 导航到规则
2. 在 PDR 选项卡中使用 TEID（上行）或 UE IP（下行）查找
3. 注意 FAR ID、QER ID、URR IDs
4. 切换到 FAR/QER/URR 选项卡查看引用的规则

启用/禁用缓冲

场景：在切换期间缓冲数据包以防止丢失

步骤：

1. 导航到 规则 → FARs
2. 在搜索字段中输入 FAR ID
3. 点击 "查找"
4. 如果缓冲为关闭，点击 "启用缓冲"
5. 验证 FAR 动作位 2 被设置（动作值增加 4）

通过缓冲页面的替代方法：

1. 导航到缓冲
2. 查看启用缓冲的 FAR
3. 在切换完成时点击 "禁用缓冲"

监控 **QoS** 合规性

检查流量是否受到速率限制：

1. 导航到 规则 → QERs
2. 查找与 UE 会话相关的 QER ID
3. 注意上行 MBR 和下行 MBR 值
4. 与 URR 流量增长率进行比较

计算平均吞吐量：

```
吞吐量 (kbps) = (流量增量字节 × 8) / (时间增量秒 × 1000)
```

如果吞吐量接近 MBR，则流量受到速率限制。

跟踪数据使用情况

监控 **URR** 流量：

1. 导航到 规则 → URRs
2. 查看上行、下行和总流量
3. 按总流量排序以查找最高用户
4. 定期刷新以观察流量增长

用例：

- 验证计费集成
- 检测异常数据使用
- 根据流量模式规划容量

故障排除

没有流量流动

检查 **PDR**：

1. 验证 TEID（上行）或 UE IP（下行）是否存在 PDR
2. 确认 FAR ID 是否有效
3. 检查 SDF 过滤器是否未阻止流量

检查 **FAR**:

1. 验证 FAR 动作是否为 FORWARD（而不是 DROP 或仅 BUFFER）
2. 确认外部头部创建是否与方向匹配
3. 验证下行的远程 IP 和 TEID 是否正确

检查 **QER**:

1. 验证门状态是否为开启（0）
2. 检查 MBR 是否不太严格

数据包被丢弃

检查 **QER** 速率限制:

1. 导航到 规则 → QERs
2. 验证 MBR 是否足够满足流量负载
3. 检查 URR 流量增长是否与预期吞吐量匹配

检查 **FAR** 动作:

1. 导航到 规则 → FARs
2. 验证动作是否为 FORWARD，而不是 DROP
3. 检查缓冲是否未卡在仅 BUFFER 模式中

缓冲问题

数据包卡在缓冲中:

1. 导航到缓冲页面
2. 检查最旧数据包的时间戳
3. 如果 > 30 秒，切换可能已失败
4. 手动刷新或清除缓冲

5. 在 FAR 上禁用缓冲

缓冲溢出：

1. 检查总数据包数与最大总数（默认 100,000）
2. 检查每个 FAR 的数据包数与每个 FAR 的最大数（默认 10,000）
3. 如果满，清除缓冲
4. 调查为何未禁用缓冲

URR 未跟踪

流量计数器为零：

1. 验证 PDR 是否引用 URR ID
2. 检查数据包是否匹配 PDR
3. 验证 FAR 是否转发（而不是丢弃）数据包
4. 确认 URR ID 是否存在于 URR 映射中

流量未报告给 **SMF**：

1. 检查 PFCP 会话报告配置
2. 验证 URR 报告触发器（流量/时间阈值）
3. 检查 PFCP 会话报告消息的日志

相关□档

- **UPF** 操作指南 - OmniUPF 架构和组件概述
- **Web UI** 操作指南 - 规则查看的控制面板使用
- 监控指南 - 统计和容量监控
- 故障排除指南 - 常见问题和诊断

OmniUPF 故障排除指南

目录

概述

本指南□供了针对常见 OmniUPF 问题的系统故障排除程序。每个部分包括症状、诊断步骤、根本原因和解决程序。

快速诊断检查表

在深入故障排除之前，请验证：

```
# 1. 检查 OmniUPF 是否正在运行
systemctl status omniupf

# 2. 检查 PFCP 关联
curl http://localhost:8080/api/v1/upf_pipeline

# 3. 检查 eBPF 映射是否已加载
ls /sys/fs/bpf/

# 4. 检查 XDP 程序是否已附加
ip link show | grep -i xdp

# 5. 检查内核日志中的错误
dmesg | tail -50
journalctl -u omniupf -n 50
```

诊断工具

OmniUPF REST API

检查 **UPF** 状态:

```
curl http://localhost:8080/api/v1/upf_status
```

检查 **PFCP** 关联:

```
curl http://localhost:8080/api/v1/upf_pipeline
```

检查会话计数:

```
curl http://localhost:8080/api/v1/sessions | jq 'length'
```

检查 **eBPF** 映射容量:

```
curl http://localhost:8080/api/v1/map_info
```

检查数据包统计：

```
curl http://localhost:8080/api/v1/packet_stats
```

检查 **XDP** 统计：

```
curl http://localhost:8080/api/v1/xdp_stats
```

---

eBPF 映射检查

列出所有 **eBPF** 映射：

```
ls -lh /sys/fs/bpf/
bpftool map list
```

显示映射详细信息：

```
bpftool map show
bpftool map dump name pdr_map_downlin
```

计算映射中的条目：

```
bpftool map dump name far_map | grep -c "key:"
```

XDP 程序检查

检查 **XDP** 程序是否已附加：

```
ip link show eth0 | grep xdp
```

列出所有 **XDP** 程序:

```
bpftool net list
```

显示 **XDP** 程序详细信息:

```
bpftool prog show
```

转储 **XDP** 统计:

```
bpftool prog dump xlated name xdp_upf_func
```

网络调试

捕获 **N4** 上的 **PFCP** 流量（控制平面）：

```
# PFCP 不被 XDP 处理，tcpdump 正常工作
tcpdump -i eth0 -n udp port 8805 -w /tmp/pfcp_traffic.pcap
```

捕获 **N3** 上的 **GTP-U** 流量（需要带外捕获）：

```
# 警告：UPF 主机上的标准 tcpdump 无法捕获 XDP 处理的数据包！
# XDP 在内核网络栈看到数据包之前处理 GTP-U。

# 请改用带外捕获：
# 1. gNB 和 UPF 之间的网络 TAP
# 2. 交换机端口镜像/SPAN 复制 N3 流量
# 3. 虚拟交换机端口镜像到分析 VM

# 在分析/监控主机上（而不是 UPF）：
# tcpdump -i <mirror_interface> -n udp port 2152 -w
/tmp/n3_capture.pcap

# 或使用统计 API 获取数据包计数：
curl http://localhost:8080/api/v1/packet_stats
curl http://localhost:8080/api/v1/n3n6_stats
```

监控数据包计数器：

```
watch -n 1 'ip -s link show eth0'
```

检查路由表：

```
ip route show
ip route get 10.45.0.100  # 检查 UE IP 的路由
```

检查 **ARP** 表：

```
ip neigh show
```

安装问题

问题：“eBPF 文件系统未挂载”

症状：

```
ERRO[0000] failed to load eBPF objects: mount bpf filesystem at
/sys/fs/bpf
```

□□□eBPF □□□□□□□□

□□□□□

```
# □□ eBPF □���□□
sudo mount bpffs /sys/fs/bpf -t bpf

# □□□□□□□□□□ /etc/fstab□
echo "bpffs /sys/fs/bpf bpf defaults 0 0" | sudo tee -a /etc/fstab

# □□□□
mount | grep bpf
```

□□□□□□□□□□

□□□

```
ERRO[0000] kernel version 5.4.0 is too old, minimum required is
5.15.0
```

□□□Linux □□□□□□□□□□□

□□□□□

```
# 检查内核版本
uname -r

# 升级内核（Ubuntu/Debian）
sudo apt update
sudo apt install linux-generic-hwe-22.04
sudo reboot

# 验证新内核
uname -r  # 应该 >= 5.15.0
```

问题：缺少 **libbpf** 依赖

症状：

```
error while loading shared libraries: libbpf.so.0: cannot open
shared object file
```

原因：未安装 libbpf 库

解决方案：

```
# 安装 libbpf（Ubuntu/Debian）
sudo apt update
sudo apt install libbpf-dev

# 验证安装
ldconfig -p | grep libbpf
```

配置问题

问题：无效的配置□件

症状：

```
ERRO[0000] unable to read config file: unmarshal errors
```

原因：配置文件中的 YAML 语法错误

解决方案：

```
# 验证 YAML 语法
cat config.yml | python3 -c "import yaml, sys;
yaml.safe_load(sys.stdin)"

# 常见问题：
# - 错误的缩进（使用空格而不是制表符）
# - 键后缺少冒号
# - 未加引号的特殊字符字符串
# - 列表项没有连字符

# 正确 YAML 的示例：
cat > config.yml <<EOF
interface_name: [eth0]
xdp_attach_mode: generic
api_address: :8080
pfcp_address: :8805
EOF
```

问题：找不到接口名称

症状：

```
ERRO[0000] interface eth0 not found
```

原因：配置的接口不存在

解决方案：

```
# 列出所有网络接口
ip link show

# 检查接口状态
ip addr show eth0

# 如果接口名称不同，请更新 config.yml：
interface_name: [ens1f0]  # 使用实际接口名称

# 对于虚拟机，请检查接口命名方案
ls /sys/class/net/
```

问题：端口已被占用

症状：

```
ERRO[0000] failed to start API server: address already in use
```

原因：端口 8080、8805 或 9090 已被其他进程绑定

解决方案：

```
# 查找使用该端口的进程
sudo lsof -i :8080
sudo netstat -tulpn | grep :8080

# 杀死冲突进程
sudo kill <PID>

# 或在配置中更改 OmniUPF 端口
api_address: :8081
pfcp_address: :8806
metrics_address: :9091
```

问题：无效的 PFCP 节点 ID

症状：

```
ERRO[0000] invalid pfcp_node_id: must be valid IPv4 address
```

原因：PFCP 节点 ID ���是有效的 IPv4 地址

解决方案：

```
# 正确：使用 IP 地址（而不是主机名）
pfcp_node_id: 10.100.50.241

# 错误：
# pfcp_node_id: localhost
# pfcp_node_id: upf.example.com
```

PFCP 关联问题

问题：未建立 PFCP 关联

症状：

- Web UI 显示“没有关联”
- SMF 日志显示“PFCP 关联设置失败”

诊断：

```
# 1. 检查 PFCP 服务器是否在监听
sudo netstat -ulpn | grep 8805

# 2. 检查防火墙规则
sudo iptables -L -n | grep 8805
sudo ufw status

# 3. 捕获 PFCP 流量
tcpdump -i any -n udp port 8805 -vv

# 4. 通过 API 检查 PFCP 关联
curl http://localhost:8080/api/v1/upf_pipeline
```

常见原因及解决方案:

防火墙阻止 **PFCP**

解决方案:

```
# 允许 PFCP 流量 (UDP 8805)
sudo ufw allow 8805/udp
sudo iptables -A INPUT -p udp --dport 8805 -j ACCEPT
```

错误的 **PFCP** 节点 **ID**

解决方案:

```
# 将 PFCP 节点 ID 设置为正确的 N4 接口 IP
pfcp_node_id: 10.100.50.241  # 必须与 N4 网络上的 IP 匹配
```

网络无法到达 **SMF**

解决方案:

```
# 测试与 SMF 的连通性
ping <SMF_IP>

# 检查到 SMF 的路由
ip route get <SMF_IP>

# 如果缺少路由，请添加
sudo ip route add <SMF_NETWORK>/24 via <GATEWAY>
```

SMF 配置了错误的 UPF IP

解决方案：

- 检查 SMF 配置中的 UPF 地址
- 确保 SMF 配置了 UPF 的 `pfcp_node_id` IP
- 验证 SMF 能够路由到 UPF 的 N4 网络

问题：PFCP 心跳失败

症状：

```
WARN[0030] PFCP heartbeat timeout for association 10.100.50.10
```

诊断：

```
# 检查 PFCP 统计
curl http://localhost:8080/api/v1/upf_pipeline | jq
'.associations[] | {remote_id, uplink_teid_count}'

# 监控心跳日志
journalctl -u omniupf -f | grep heartbeat
```

原因及解决方案：

网络数据包丢失

解决方案：

```
# 检查到 SMF 的数据包丢失
ping -c 100 <SMF_IP> | grep loss

# 如果丢失率高，请调查网络：
# - 检查链路状态
# - 检查交换机/路由器健康
# - 检查是否拥堵
```

心跳间隔过于激进

解决方案：

```
# 增加心跳间隔
heartbeat_interval: 30  # □ 5 增加到 30 秒
heartbeat_retries: 5    # 增加重试次数
heartbeat_timeout: 10   # 增加超时时间
```

数据包处理问题

问题：没有数据包流动（**RX/TX** 计数为 **0**）

症状：

- 统计页面显示 0 RX/TX 数据包
- UE 无法建立数据会话

诊断：

```
# 1. 检查 XDP 程序是否已附加
ip link show eth0 | grep xdp

# 2. 检查接口是否 UP
ip link show eth0

# 3. 检查数据包统计（XDP 感知）
# 注意：tcpdump 无法看到 XDP 处理的 GTP-U 数据包
curl http://localhost:8080/api/v1/packet_stats
```

解决方案：

XDP 程序未附加

解决方案：

```
# 重启 OmniUPF 以重新附加 XDP
sudo systemctl restart omniupf

# 验证附加
ip link show eth0 | grep xdp
bpftool net list
```

接口关闭或没有链路

解决方案：

```
# 启动接口
sudo ip link set eth0 up

# 检查链路状态
ethtool eth0 | grep "Link detected"

# 如果链路关闭，请检查物理连接或虚拟机网络配置
```

配置了错误的接口

解决方案：

```
# 在 config.yml 中更新正确的接口
interface_name: [ens1f0]  # 使用 'ip link show' 中的实际接口名称
```

问题：接收到数据包但未转发（丢包率高）

症状：

- RX 计数增加但 TX 计数不变
- 丢包率 > 1%

诊断：

```
# 检查丢包统计
curl http://localhost:8080/api/v1/xdp_stats | jq '.drop'

# 检查路由统计
curl http://localhost:8080/api/v1/packet_stats | jq '.route_stats'

# 监控数据包丢失
watch -n 1 'curl -s http://localhost:8080/api/v1/packet_stats | jq
".total_rx, .total_tx, .total_drop"'
```

常见原因：

没有 **PDR** 匹配（未知 **TEID** 或 **UE IP**）

解决方案：

诊断：

```
# 检查会话是否存在
curl http://localhost:8080/api/v1/sessions

# 如果没有会话，请验证：
# - PFCP 关联已建立
# - SMF 已创建会话
# - 会话建立成功

# 检查 PDR 映射条目
bpftool map dump name pdr_map_teid_ip | grep -c key
bpftool map dump name pdr_map_downlin | grep -c key
```

路由失败

解决方案：

```
# 检查 FIB 查找失败
curl http://localhost:8080/api/v1/packet_stats | jq '.route_stats'

# 测试 UE IP 的路由
ip route get 10.45.0.100

# 添加缺失的路由
sudo ip route add 10.45.0.0/16 dev eth1  # 将 UE 池路由到 N6
```

QER 速率限制

症状：

- 吞吐量低于预期
- 流量限制在特定速率
- URR 卷计数显示平台行为
- 在流量突发期间 XDP 丢包计数增加

诊断：

1. 检查会话的 **MBR** 配置：

```
# 查找会话的 QER ID
curl http://localhost:8080/api/v1/pfcp_sessions | jq '.data[] |
select(.ue_ip == "10.45.0.1")'

# 查找 QER 配置
curl http://localhost:8080/api/v1/qer_map | jq '.data[] |
select(.qer_id == 1)'
```

2. 验证门状态:

```
# 门状态应为 0（打开），上行和下行均如此
curl http://localhost:8080/api/v1/qer_map | jq '.data[] |
{qer_id, ul_gate: .ul_gate_status, dl_gate: .dl_gate_status}'
```

3. 从 **URR** 计算实际吞吐量:

```
# 在两个时间点查询 URR 卷计数
curl http://localhost:8080/api/v1/urr_map | jq '.data[] |
select(.urr_id == 0)'

# 计算吞吐量（手动）：
# throughput_kbps = (volume_delta_bytes × 8) /
time_delta_seconds / 1000
```

4. 比较 **MBR** 与实际吞吐量:
    - 预期吞吐量 ≈ MBR 的 95-98%（由于协议开销）
    - 如果吞吐量显著低于 MBR，请检查其他瓶颈
    - 如果吞吐量恰好与 MBR 相符，则速率限制按预期工作

解决方案:

- 如果 **MBR** 太低：请求 SMF 通过 PFCP 会话修改更新 QER，增加 MBR
- 如果门关闭：调查 SMF 关闭门的原因（策略、配额或错误）
- 如果速率限制意外：验证 SMF 策略配置和 QoS 配置文件

理解 **MBR** 执行:

OmniUPF 使用滑动窗口算法以纳秒精度在 eBPF 数据路径中执行 MBR 限制。有关如何根据数据包大小和速率决定丢包决策的详细说明，请参见 规则管理指南 - MBR 执行机制。

常见场景：

- **VoIP** 通话掉线：检查 MBR 是否足以支持编解码器比特率（G.711 = ~80 kbps）
- 视频流缓冲：确保 MBR > 视频比特率 + 开销（1080p = ~5-10 Mbps）
- 突发流量：在 5ms 窗口内允许小的突发流量，持续流量受限

问题：单向流量（上行正常，下行不正常）

症状：

- RX N3 数据包但没有 TX N3 数据包（下行问题）
- RX N6 数据包但没有 TX N6 数据包（上行问题）

诊断：

```
# 检查 N3/N6 接口统计（XDP 感知方法）
curl http://localhost:8080/api/v1/n3n6_stats
curl http://localhost:8080/api/v1/packet_stats

# 注意：标准 tcpdump 无法捕获 XDP 处理的 GTP-U 流量
# 使用统计 API 或 xdpdump 进行流量分析
# 请参见“使用 XDP 捕获数据包”部分了解详细信息
```

上行故障（**RX N3**，无 **TX N6**）：

原因：没有 FAR 操作或到 N6 的路由问题

解决方案：

```
# 检查 FAR 是否具有 FORWARD 操作
curl http://localhost:8080/api/v1/sessions | jq '.[].fars[] |
select(.applied_action == 2)'

# 检查 N6 路由是否存在
ip route get 8.8.8.8  # 测试到互联网的路由

# 如果缺少默认路由，请添加
sudo ip route add default via <N6_GATEWAY> dev eth1
```

下行故障（**RX N6**，无 **TX N3**）：

原因：没有下行 PDR 或缺少 GTP 封装

解决方案：

```
# 检查 UE IP 的下行 PDR 是否存在
curl http://localhost:8080/api/v1/sessions | jq '.[].pdrs[] |
select(.pdi.ue_ip_address)'

# 验证 FAR 是否具有 OUTER_HEADER_CREATION
curl http://localhost:8080/api/v1/sessions | jq '.[].fars[] |
.outer_header_creation'

# 检查 gNB 可达性
ping <GNB_N3_IP>
```

XDP 和 eBPF 问题

有关详细的 **XDP** 配置、模式选择和故障排除，请参见 **XDP** 模式指南。

问题：XDP 程序加载失败

症状：

```
ERRO[0000] failed to load XDP program: invalid argument
```

诊断：

```
# 检查内核 XDP 支持
grep XDP /boot/config-$(uname -r)

# 应显示：
# CONFIG_XDP_SOCKETS=y
# CONFIG_BPF=y
# CONFIG_BPF_SYSCALL=y

# 检查 dmesg 以获取详细错误
dmesg | grep -i bpf
```

原因及解决方案：

内核缺乏 **XDP** 支持

解决方案：

```
# 重新构建内核以支持 XDP 或升级到更新的内核
# Ubuntu 22.04+ 默认启用 XDP
sudo apt install linux-generic-hwe-22.04
sudo reboot
```

XDP 程序验证失败

解决方案：

```
# 检查 OmniUPF 日志以获取验证错误
journalctl -u omniupf | grep verifier

# 常见问题：
# - eBPF 复杂性超过限制（增加内核限制）
# - 无效的内存访问（eBPF 代码中的错误）

# 增加 eBPF 验证器日志级别以进行调试
sudo sysctl kernel.bpf_stats_enabled=1
```

问题: XDP 中止计数增加

症状:

- XDP 统计显示 aborted > 0
- 数据包丢失增加

诊断:

```
# 检查 XDP 中止计数
curl http://localhost:8080/api/v1/xdp_stats | jq '.aborted'

# 监控 XDP 统计
watch -n 1 'curl -s http://localhost:8080/api/v1/xdp_stats'
```

原因: eBPF 程序遇到运行时错误

解决方案:

```
# 检查内核日志以获取 eBPF 错误
dmesg | grep -i bpf

# 重启 OmniUPF 以重新加载 eBPF 程序
sudo systemctl restart omniupf

# 如果问题仍然存在，请启用 eBPF 日志记录（需要重新构建）：
# 使用 BPF_ENABLE_LOG=1 构建 OmniUPF
```

问题: eBPF 映射已满（容量耗尽）

症状:

- 会话建立失败
- 映射容量达到 100%

诊断:

```
# 检查映射容量
curl http://localhost:8080/api/v1/map_info | jq '.[] | {map_name,
capacity, used, usage_percent}'

# 确定满映射
curl http://localhost:8080/api/v1/map_info | jq '.[] |
select(.usage_percent > 90)'
```

立即缓解:

```
# 1. 确定过期会话
curl http://localhost:8080/api/v1/sessions | jq '.[] | {seid,
uplink_teid, created_at}'

# 2. 请求 SMF 删除旧会话
# （通过 SMF 管理界面或 API）

# 3. 监控映射使用情况减少
watch -n 5 'curl -s http://localhost:8080/api/v1/map_info | jq ".
[] | select(.map_name==\"pdr_map_downlin\") | .usage_percent"'
```

长期解决方案:

```
# 在 config.yml 中增加映射容量
max_sessions: 200000  # □ 100000 增加

# 或设置单个映射大小
pdr_map_size: 400000
far_map_size: 400000
qer_map_size: 200000
```

重要：更改映射大小需要重启 OmniUPF，并且 会清除所有现有会话。

性能问题

问题：吞吐量低（低于预期）

症状：

- 吞吐量 < 1 Gbps 尽管 NIC 能力足够
- CPU 利用率高

诊断：

```
# 检查数据包速率
curl http://localhost:8080/api/v1/packet_stats | jq '.total_rx,
.total_tx'

# 检查 NIC 统计
ethtool -S eth0 | grep -i drop

# 检查 XDP 模式
ip link show eth0 | grep xdp
```

解决方案：

使用通用 **XDP** 模式

解决方案：

```
# 切换到本地模式以获得更好的性能
xdp_attach_mode: native  # 需要支持 XDP 的 NIC/驱动程序
```

单核瓶颈

解决方案：

```
# 在 NIC 上启用 RSS（接收侧负载均衡）
ethtool -L eth0 combined 4  # 使用 4 个 RX/TX 队列

# 验证 RSS 是否启用
ethtool -l eth0

# 将中断固定到□定 CPU
# 请参见 /proc/interrupts 并使用 irqbalance 或手动亲和性
```

缓冲膨胀

解决方案：

```
# 减少缓冲限制以降低延迟
buffer_max_packets: 5000
buffer_packet_ttl: 15
```

问题：高延迟

症状：

- Ping 延迟 > 50ms
- 用户体验下降

诊断：

```
# 测试到 UE 的延迟
ping -c 100 <UE_IP> | grep avg

# 检查缓冲的数据包
curl http://localhost:8080/api/v1/upf_buffer_info | jq
'.total_packets_buffered'

# 检查路由缓存性能
curl http://localhost:8080/api/v1/packet_stats | jq '.route_stats'
```

解决方案：

数据包被过度缓冲

解决方案：

```
# 检查为什么数据包被缓冲
curl http://localhost:8080/api/v1/upf_buffer_info | jq '.buffers[]
| {far_id, packet_count, direction}'

# 如果卡住，请清除缓冲区
# （重启 OmniUPF 或触发 PFCP 会话修改以应用 FAR）
```

FIB 查找延迟

解决方案：

```
# 确保启用路由缓存（构建时选项）
# 使用 BPF_ENABLE_ROUTE_CACHE=1 构建

# 优化路由表
# 使用更少、更具体的路由，而不是许多小路由
```

问题：负载下的数据包丢失

症状：

- 丢包率随着流量增加
- NIC 上的 RX 错误

诊断：

```
# □□ NIC □□
ethtool -S eth0 | grep -E "drop|error|miss"

# □□□□□□□□□□
ethtool -g eth0

# □□□□□□□
watch -n 1 'ethtool -S eth0 | grep -E "drop|miss"'
```

□□□□□

```
# □□ RX □□□□□□□□
ethtool -G eth0 rx 4096

# □□ TX □□□□□□□□
ethtool -G eth0 tx 4096

# □□□□□□
ethtool -g eth0
```

□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□ **XDP □□□□□**□

Proxmox□VM □□ XDP □□□□

□□□

- □□□□□□□□□□□ XDP □□
- □□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□ SR-IOV

□□□□□□

□□ **1**□□□□□□□□□□□□□□□

```
xdp_attach_mode: generic
```

选项 **2**：配置 **SR-IOV** 直通

```
# 在 Proxmox 主机上:
# 1. 启用 IOMMU
nano /etc/default/grub
# 添加: intel_iommu=on iommu=pt
update-grub
reboot

# 2. 创建 VF
echo 4 > /sys/class/net/eth0/device/sriov_numvfs

# 3. 在 Proxmox UI 中将 VF 分配给 VM
# 硬件 → 添加 → PCI 设备 → 选择 VF

# 在 VM 中:
interface_name: [ens1f0]  # SR-IOV VF
xdp_attach_mode: native
```

VMware：需要混杂模式

症状：

- OmniUPF 未接收到数据包

原因：vSwitch 阻止不匹配的 MAC 地址

解决方案：

```
# 在 vSphere 客户端中启用 vSwitch 的混杂模式:
# 1. 选择 vSwitch → 编辑设置
# 2. 安全性 → 混杂模式: 接受
# 3. 安全性 → MAC 地址更改: 接受
# 4. 安全性 → 伪造传输: 接受
```

VirtualBox: 性能非常低

症状:

- 吞吐量 < 100 Mbps

原因: VirtualBox 不支持 SR-IOV 或本地 XDP

解决方案:

```yaml
# 使用通用模式（唯一选项）
xdp_attach_mode: generic

# 优化 VirtualBox 设置:
# - 使用 VirtIO-Net 适配器（如果可用）
# - 启用“允许所有”混杂模式
# - 为虚拟机分配更多 CPU 核心
# - 使用桥接网络而不是 NAT

# 考虑迁移到 KVM/Proxmox 以获得更好的性能
```

NIC 和驱动程序问题

问题: NIC 驱动程序不支持 XDP

症状:

```
ERRO[0000] failed to attach XDP program: operation not supported
```

诊断:

```
# 检查 NIC 驱动程序
ethtool -i eth0 | grep driver

# 检查驱动程序是否支持 XDP
modinfo <driver_name> | grep -i xdp

# 列出支持 XDP 的接口
ip link show | grep -B 1 "xdpgeneric\|xdpdrv\|xdpoffload"
```

解决方案：

选项 **1**：使用通用模式

```
xdp_attach_mode: generic
```

选项 **2**：更新 **NIC** 驱动程序

```
# 检查驱动程序更新（Ubuntu）
sudo apt update
sudo apt install linux-modules-extra-$(uname -r)

# 或安装供应商特定的驱动程序
# 示例：英特尔：
# 从 https://downloadcenter.intel.com/ 下载
```

选项 **3**：更换 **NIC**

```
# 使用支持 XDP 的 NIC：
# - Intel X710/E810
# - Mellanox ConnectX-5/ConnectX-6
# - Broadcom BCM57xxx（bnxt_en 驱动程序）
```

问题：驱动程序崩溃或内核恐慌

症状：

- 附加 XDP 后内核恐慌
- NIC 停止响应

诊断：

```
# 检查内核日志
dmesg | tail -100

# 检查驱动程序错误
journalctl -k | grep -E "BUG:|panic:"
```

解决方案：

```
# 1. 更新内核和驱动程序
sudo apt update
sudo apt upgrade
sudo reboot

# 2. 禁用 XDP 直通（仅使用本地模式）
xdp_attach_mode: native

# 3. 作为解决方法使用通用模式
xdp_attach_mode: generic

# 4. 向 NIC 供应商或 Linux 内核团队报告错误
```

会话建立失败

问题：会话建立失败

症状：

- SMF 报告会话建立失败
- UE 无法建立 PDU 会话

请参见 PFCP 原因代码参考 以获取常见失败场景和解决方案。

诊断：

```
# 检查 OmniUPF 日志中的会话错误
journalctl -u omniupf | grep -i "session establishment"

# 检查 PFCP 会话计数
curl http://localhost:8080/api/v1/sessions | jq 'length'

# 在会话建立期间捕获 PFCP 流量
tcpdump -i any -n udp port 8805 -w /tmp/pfcp_session.pcap
```

常见原因：

映射容量已满

解决方案：

```
# 检查映射使用情况
curl http://localhost:8080/api/v1/map_info | jq '.[] |
select(.usage_percent > 90)'

# 增加容量（请参见 eBPF 映射已满部分）
```

无效的 PDR/FAR 参数

解决方案：

```
# 检查 OmniUPF 日志中的验证错误
journalctl -u omniupf | grep -E "invalid|error" | tail -20

# 常见问题：
# - 无效的 UE IP 地址（0.0.0.0 或重复）
# - 无效的 TEID（0 或重复）
# - PDR 缺少 FAR
# - 无效的 FAR 操作

# 验证 SMF 配置和会话参数
```

不支持的功能（UEIP/FTUP）

解决方案：

```yaml
# 如果需要，启用所需的功能
feature_ueip: true  # 由 UPF 分配 UE IP
ueip_pool: 10.60.0.0/16

feature_ftup: true  # 由 UPF 分配 F-TEID
teid_pool: 100000
```

缓冲问题

问题：数据包卡在缓冲区

症状：

- 缓冲的数据包计数增加
- 交接后数据包未交付

诊断：

```bash
# 检查缓冲统计
curl http://localhost:8080/api/v1/upf_buffer_info

# 检查单个 FAR 缓冲区
curl http://localhost:8080/api/v1/upf_buffer_info | jq '.buffers[] | {far_id, packet_count, oldest_packet_ms}'

# 监控缓冲区大小
watch -n 5 'curl -s http://localhost:8080/api/v1/upf_buffer_info | jq ".total_packets_buffered"'
```

原因及解决方案：

FAR 从未更新为 FORWARD

原因：SMF 从未发送 PFCP 会话修改以应用 FAR

解决方案：

```
# 检查 FAR 状态
curl http://localhost:8080/api/v1/sessions | jq '.[].fars[] |
{far_id, applied_action}'

# 操作 BUFF = 1 (缓冲)
# 操作 FORW = 2 (转发)

# 如果卡在 BUFF 状态，请请求 SMF：
# - 发送 PFCP 会话修改请求
# - 更新 FAR 以进行 FORW 操作
```

缓冲 **TTL** 过期

原因：数据包在 FAR 更新之前过期

解决方案：

```
# 增加缓冲 TTL
buffer_packet_ttl: 60  # □ 30 增加到 60 秒
```

缓冲溢出

原因：每个 FAR 缓冲的数据包过多

解决方案：

```
# 增加缓冲限制
buffer_max_packets: 20000  # 每个 FAR
buffer_max_total: 200000   # 全局限制
```

高级调试

启用调试日志

```
logging_level: debug  # trace | debug | info | warn | error
```

```
# 使用调试日志重启 OmniUPF
sudo systemctl restart omniupf

# 实时监控日志
journalctl -u omniupf -f --output cat
```

---

eBPF 程序跟踪

```
# 跟踪 eBPF 程序执行（需要 bpftrace）
sudo bpftrace -e 'tracepoint:xdp:* { @[probe] = count(); }'

# 跟踪映射操作
sudo bpftrace -e 'tracepoint:bpf:bpf_map_lookup_elem {
printf("%s\n", str(args->map_name)); }'
```

使用 **XDP** 捕获数据包

理解 **XDP** 数据包捕获限制：

XDP 在内核网络栈之前处理数据包，因此标准 `tcpdump` 无法看到 **XDP** 处理的流量。N3 上的 GTP-U 数据包（UDP 2152）被 XDP 处理，并且不会出现在 UPF 主机上的 tcpdump 中。

流量分析的推荐方法：

```
# □□ 1□□□□□ API □□□□□□□□□
curl http://localhost:8080/api/v1/xdp_stats
curl http://localhost:8080/api/v1/packet_stats | jq
curl http://localhost:8080/api/v1/n3n6_stats

# □□ 2□□□ PFCP □□□□□ XDP □□□
tcpdump -i any -n udp port 8805 -w /tmp/pfcp.pcap

# □□ 3□□□□□□□□□□□□□□ GTP-U□
# □□□□ TAP □□□□□□□□□□□□□
# □□□
# - gNB □ UPF □□□□□ TAP
# - □□□ SPAN/□□□□□□ N3 □□□□□□
# - □□□□□□□□□□ hypervisor □□□□□
#
# □□□□□□□□□□ UPF□□
# tcpdump -i <mirror_interface> -n udp port 2152 -w
/tmp/n3_mirror.pcap
```

□□□□□□□□□

□□□□□

```
# □□□□ TAP □□□□□□□□□□
# □□□Cisco □□□ SPAN □□
(config)# monitor session 1 source interface Gi1/0/1
(config)# monitor session 1 destination interface Gi1/0/24

# □□□□ Gi1/0/24 □□□□□□□
tcpdump -i eth0 -n udp port 2152 -w /tmp/n3_capture.pcap
```

□□□□□VMware□KVM □□□

```
# □□□□□□□□□□□□□□□ UPF □□□□□□□□ VM
# □□□Linux □□□ tcpdump □□□□ VM □
# □ hypervisor □□□ UPF □ N3 □□□□□□□□□□
#
# □□□□ VM □□
tcpdump -i eth1 -n udp port 2152 -w /tmp/n3_virtual.pcap
```

为什么需要带外：

- XDP 完全绕过内核网络栈
- 数据包在 NIC 驱动程序或硬件中处理
- 主机上的 tcpdump 在 XDP 处理后看到数据包（太晚了）
- 带外捕获看到原始线流量，在 UPF 处理之前

您可以在 **UPF** 主机上捕获的内容：

- □ PFCP 流量（UDP 8805） - 控制平面，未被 XDP 处理
- □ API 响应和指标
- □ GTP-U 流量（UDP 2152） - 数据平面，被 XDP 处理

寻求帮助

如果故障排除步骤未能解决您的问题：

1. 收集诊断信息：

```
# 系统信息
uname -a
cat /etc/os-release

# OmniUPF 信息
curl http://localhost:8080/api/v1/upf_status
curl http://localhost:8080/api/v1/map_info
curl http://localhost:8080/api/v1/packet_stats

# 日志
journalctl -u omniupf --since "1 hour ago" > /tmp/omniupf.log
dmesg > /tmp/dmesg.log

# 网络信息
ip addr > /tmp/network.txt
ip route >> /tmp/network.txt
ethtool eth0 >> /tmp/network.txt
```

2. 报告问题:
    - OmniUPF 版本
    - Linux 内核版本
    - 网络拓扑图
    - 配置文件（去除敏感信息）
    - 相关日志摘录
    - 重现步骤

相关□档

- 配置指南 - 配置参数和示例
- 架构指南 - eBPF/XDP 内部和性能调优
- 监控指南 - 统计、容量和警报
- 指标参考 - 用于故障排除的 Prometheus 指标
- **PFCP** 原因代码 - PFCP 错误代码和故障排除
- 规则管理指南 - PDR、FAR、QER、URR 概念
- 操作指南 - UPF 架构和概述

Web UI 操作指南

目录

概述

OmniUPF Web UI 提供了一个全面的控制面板，用于实时监控和管理用户平面功能。该界面基于 Phoenix LiveView 构建，提供：

- 实时可视化 PFCP 会话和活动 PDU 连接
- 规则检查 PDR、FAR、QER 和 URR 跨所有会话
- 缓冲区管理 在移动事件期间进行数据包缓冲
- 统计监控 数据包处理、路由和接口
- 容量跟踪 eBPF 映射使用情况和限制
- 实时日志查看 以进行故障排除

架构

控制面板通过其 REST API 与多个 OmniUPF 实例进行通信，以：

- 查询 PFCP 会话和关联
- 检查数据包检测和转发规则
- 监控数据包缓冲区及其状[illegible]
- 访问实时统计和性能指标
- 跟踪 eBPF 映射的容量和利用率

访问控制面板

默认访问

控制面板可以通过 HTTPS 在 OmniUPF 管理服务器上访问:

```
https://<upf-server>:443/
```

默认端口: 443 (带自签名证书的 HTTPS)

配置

控制面板需要在 `config/config.exs` 中配置 OmniUPF 主机:

可以为多实例部署配置多个 UPF 实例:

`upf_hosts` 配置定义了在 UI 中的主机选择下拉列表中可用的 OmniUPF 实例。

导航

控制面板为每个操作区域[illegible]供导航选项卡:

- 会话 - `/sessions` - PFCP 会话和关联
- 规则 - `/rules` - PDR、FAR、QER、URR 规则检查
- 缓冲区 - `/buffers` - 数据包缓冲监控和控制
- 统计 - `/statistics` - 数据包、路由、XDP 和接口统计
- 容量 - `/capacity` - eBPF 映射使用和容量监控
- 配置 - `/upf_config` - UPF 配置和数据平面地址

- 路由 - `/routes` - UE 路由和路由协议会话（OSPF、BGP）
- **XDP** 能力 - `/xdp_capabilities` - XDP 模式支持和性能能力
- 日志 - `/logs` - 实时日志流

会话视图

URL: `/sessions`

特性

会话视图显示来自所选 OmniUPF 实例的所有活动 PFCP 会话和关联。

PFCP 关联摘要

显示所有活动 PFCP 关联（来自 SMF/PGW-C 的控制连接）：

列	描述
节点 **ID**	SMF 或 PGW-C 节点标识符（FQDN 或 IP）
地址	SMF/PGW-C 用于 PFCP 通信的 IP 地址
下一个会话 **ID**	此关联的下一个可用 PFCP 会话 ID

目的：

- 验证 SMF 与 UPF 的连接性
- 监控控制平面连接的数量
- 跟踪每个关联的会话 ID 分配

活动会话表

显示所有 PFCP 会话，代表活动的 UE PDU 会话：

□	□□
□□ **SEID**	UPF □□□□□□□□□□□
□□ **SEID**	SMF □□□□□□□□□□□
UE IP	□□□□□ IPv4 □ IPv6 □□
TEID	□□□□□ GTP-U □□□□□□□
PDRs	□□□□□□□□□□□□□
FARs	□□□□□□□□□□□□
QERs	□□□□ QoS □□□□□□
URRs	□□□□□□□□□□□□
□□	□□□□□□□□□□□□□□

□□□

- □ **IP** □□□□□□□ UE IP □□□□□
- □ **TEID** □□□□□□□□ ID □□□□
- □□□□□□□□□□ PDR/FAR/QER/URR JSON □□□□
- □□□□□□ 10 □□□□□

□□□□□□□□

□□□□□□□□□□□“□□”□□□□□□□

- □□□□□□□ **(PDRs)**□□□ TEID□UE IP□FAR ID□QER ID□SDF □□□□□□ JSON
  - **PDR ID** □□□ - □□□□□□□□□□□□□□□□□□□ PDR □□□□
  - □□ PDR□TEID ≠ 0□□□□□□ PDR □□
  - □□ PDR□IPv4□□□□□□ PDR □□
  - □□ PDR□IPv6□□□□ IPv6 □□ PDR □□
- □□□□□□ **(FARs)**□□□□□□□□□□□□□□□□

- **QoS** 执行规则 **(QERs)**: MBR、GBR、QFI 和其他 QoS 参数
- 使用报告规则 **(URRs)**: 体积计数器（上行、下行、总字节）

□□□□□□□□□□□□□□□□ PDRs□FARs □ QERs□

用例

验证 **UE** 连接性:

1. 导航到会话视图
2. 在过滤器中输入 UE IP 地址
3. 确认会话存在且 TEID 正确
4. 展开以验证 PDR/FAR 配置

监控会话计数:

- 检查标题中的总会话计数
- 在多个 UPF 实例之间进行比较
- 跟踪会话随时间的增长

故障排除会话问题：

- 搜索特定的 UE IP 或 TEID
- 展开会话以检查规则配置
- 验证 FAR 转发参数
- 检查 QER QoS 设置

实时更新

会话视图每 10 秒自动刷新一次。健康检查指示器显示 UPF 连接状态：

- 健康（绿色）：UPF 可访问且响应
- 不健康（红色）：UPF 不可访问或未响应
- 未知（灰色）：健康状态尚未确定

规则管理

URL: `/rules`

规则视图□供对所有数据包检测、转发、QoS 和使用报告规则的全面检查，跨所有会话。

PDR 选项卡 **-** 数据包检测规则

查看和检查 UPF 中的所有 PDR，带有 查找表单 和 可点击导航：

上行 **PDRs** (N3 → N6):

- 查找表单：按 TEID 搜索以查看特定上行 PDR 详细信息
- **TEID**：来自 gNB 的 GTP-U 隧道端点 ID（可点击 - 导航到查找）
- **FAR ID**：关联的转发动作规则（可点击 - 导航到 FAR 选项卡）
- **QER ID**：关联的 QoS 执行规则（可点击 - 导航到 QER 选项卡）
- **URR IDs**：关联的使用报告规则（可点击 - 导航到 URR 选项卡）
- 外部头移除：GTP-U 解封装标志
- **SDF** 过滤器：服务数据流分类规则

下行 **PDRs** (N6 → N3):

- 查找表单：按 UE IPv4 地址搜索以查看特定下行 PDR 详细信息
- **UE IP**：用户设备的 IPv4 地址（在查找结果中显示）
- **FAR ID**：关联的转发动作规则（可点击 - 导航到 FAR 选项卡）
- **QER ID**：关联的 QoS 执行规则（可点击 - 导航到 QER 选项卡）
- **URR IDs**：关联的使用报告规则（可点击 - 导航到 URR 选项卡）
- **SDF** 模式：服务数据流过滤模式（无、仅 sdf、sdf + 默认）
- 分页：使用页面控件浏览 PDR（默认每页 100，最大 1000）

IPv6 下行 **PDRs**：

- API 支持 IPv6 下行 PDR 的分页
- 结构与 IPv4 相同，但按 IPv6 地址键入
- 如果需要，可以添加完整的 UI 选项卡

FAR 选项卡 - 转发动作规则

查看所有 FAR 及其转发动作和参数：

特性：

- 查找表单：按 FAR ID 搜索以查看特定 FAR 详细信息
- 自动查找：从 PDR 详细信息中点击 FAR ID 会自动填充查找
- 实时更新：FAR 状态反映当前缓冲状态

列	描述
FAR ID	唯一的转发规则标识符
动作	转发动作标志（FORWARD、DROP、BUFFER、DUPLICATE、NOTIFY）
缓冲	当前缓冲状态（启用/禁用）
目标	外部头创建参数（TEID、IP 地址）

FAR 动作标志：

- **FORWARD (1)**：将数据包转发到目标

- **DROP (2)**□□□□□□□
- **BUFFER (4)**□□□□□□□□□□□□□
- **NOTIFY (8)**□□□□□□□□□□□
- **DUPLICATE (16)**□□□□□□□□□□□□□

□□□□□

- □□“□□□□□□□□□”□□“□□□□□”□□□□□□□□
- □□□□□□□□□□□□□□□□□
- □ eBPF □□□□□□□□ FAR □□

QER □□□ - QoS □□□□

□□□□□□□□□□ QoS □□□

□□□

- □□□□□□□□ PDR □□□□□□□□ QER ID □□□□□□□□□□ QER
- □□□□□□□□ PDR □□□□QER □□□□□□□
- □□□□□□□□□□□□□ QER□□□□□□ 100□□□ 1000□

□	□□
QER ID	□□□ QoS □□□□□□□□ PDR □□□□□□□□
MBR□□□□	□□□□□□□□□□□□□□kbps□
MBR□□□□	□□□□□□□□□□□□□□kbps□
GBR□□□□	□□□□□□□□□□□□□□kbps□
GBR□□□□	□□□□□□□□□□□□□□kbps□
QFI	QoS □□□□□5G □□□

QoS □□□

- **MBR = 0**：没有速率限制
- **GBR = 0**：尽力而为（没有保证带宽）
- **GBR > 0**：保证比特率流（优先级）

URR 选项卡 - 使用报告规则

查看使用跟踪规则和体积计数器：

特性：

- 查找表单：按 URR ID 搜索以查找并突出显示特定 URR
- 可点击导航：从 PDR 详细信息中点击 URR ID 导航并突出显示特定 URR
- 自动高亮：当从 PDR 或通过查找搜索时，URR 行在蓝色中高亮显示
- 分页：使用页面控件浏览 URR（默认每页 100，最大 1000）

列	描述
URR ID	唯一的使用报告规则标识符（从 PDR 引用时可点击）
上行体积	从 UE 发送到数据网络的字节数
下行体积	从数据网络发送到 UE 的字节数
总体积	双向的总字节数
操作	删除按钮以重置此 URR 的计数器

体积显示：

- 自动格式化（B、KB、MB、GB、TB）
- 实时计数器每次刷新时更新
- 用于计费和分析

过滤：

- 仅显示体积非零的 URR
- 过滤掉所有计数器为 0 的非活动 URR，以□高性能

□□

□□□□□□□□

1. □□□□□□ → PDR □□□
2. □□□□ TEID □ UE IP
3. □□ PDR □□□□ FAR □ QER □□

□□□□□□□□□□

1. □□□□□□ → FAR □□□
2. □□□ PDR □□□ FAR ID
3. □□□□□□ FORWARD□□□□ DROP □ BUFFER□
4. □□□□□□□□□□

□□ **QoS** □□□

1. □□□□□□ → QER □□□
2. □□ MBR □ GBR □□□□□□
3. □□ 5G □□ QFI □□

□□□□□□□□

1. □□□□□□ → URR □□□
2. □□□□□□□□□□□□□□
3. □□□□□□□□□□□□
4. □□□□□□□

□□□□□

URL□`/buffers`

□□

□□□□□□□□ UPF □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□

仪表板显示汇总的缓冲区统计信息：

- 总数据包：所有 FAR 中缓冲的数据包数量
- 总字节：缓冲的数据总大小
- 总 **FARs**：具有缓冲数据包的 FAR 数量
- 每个 **FAR** 的最大数量：每个 FAR 允许的最大数据包数量
- 最大总数：最大缓冲的数据包总数
- 数据包 **TTL**：缓冲数据包的生存时间（秒）

按 **FAR** 的缓冲区

所有具有缓冲数据包的 FAR 的表：

列	描述
FAR ID	转发动作规则标识符
数据包计数	此 FAR 缓冲的数据包数量
字节计数	此 FAR 缓冲的总字节数
最旧数据包	最旧缓冲数据包的时间戳
最新数据包	最新缓冲数据包的时间戳
操作	缓冲控制按钮（药丸样式）

缓冲区控制操作

对于每个具有缓冲数据包的 FAR，□供以下药丸样式按钮：

缓冲控制：

- 禁用缓冲（红色）：关闭此 FAR 的缓冲（更新 FAR 动作标志）
- 启用缓冲（紫色���：开启此 FAR 的缓冲

缓冲操作：

- 刷新（蓝色）：使用当前 FAR 规则重放所有缓冲的数据包
- 清除（灰色）：在不转发的情况下删除所有缓冲的数据包

清除所有缓冲区：

- 标题中的红色“清除所有”按钮
- 清除所有 FAR 的缓冲区
- 需要确认

用例

监控切换缓冲：

1. 在切换期间，验证数据包正在被缓冲
2. 检查 FAR 的缓冲状态（应启用）
3. 监控数据包计数和年龄

完成切换：

1. 在路径切换后，点击“刷新”以重放缓冲的数据包
2. 验证数据包转发到新路径
3. 点击“禁用缓冲”以停止缓冲

清除卡住的缓冲区：

1. 确定具有旧缓冲数据包的 FAR（检查最旧时间戳）
2. 点击“清除”以丢弃过时的数据包
3. 或点击“禁用缓冲”以防止进一步缓冲

故障排除缓冲溢出：

1. 检查总数据包计数与最大总数的比较
2. 确定具有过度缓冲的 FAR
3. 验证 SMF 是否已发送会话修改以禁用缓冲
4. 如果 SMF 命令丢失，手动禁用缓冲

实时更新

缓冲区视图每 5 秒自动刷新一次，以显示当前缓冲状态。

统计仪表板

URL: `/statistics`

特性

统计视图□供来自 OmniUPF 数据平面的实时性能指标。有关 Prometheus 指标的详细信息，请参见 指标参考。

数据包统计

汇总的数据包处理计数器：

- 接收数据包：在所有接口上接收到的总数据包
- 发送数据包：在所有接口上发送的总数据包
- 丢弃的数据包：由于错误或策略而丢弃的数据包
- **GTP-U** 数据包：使用 GTP-U 封装处理的数据包

用途：监控整体 UPF 流量负载和数据包丢弃率

路由统计

每条路由的转发指标（如果可用）：

- 路由命中：每个路由规则匹配的数据包
- 转发成功：成功转发的数据包计数
- 转发错误：转发尝试失败的次数

用途：识别繁忙路由和转发错误

XDP 统计

eXpress Data Path 性能指标：

- **XDP** 处理：在 XDP 层处理的总数据包

- **XDP** 通过：发送到网络栈的数据包
- **XDP** 丢弃：在 XDP 层丢弃的数据包
- **XDP** 中止：XDP 程序中的处理错误

用途：监控 XDP 性能并检测处理错误

XDP 丢弃原因：

- 无效的数据包格式
- eBPF 映射查找失败
- 基于策略的丢弃
- 资源耗尽

N3/N6 接口统计

每个接口的流量计数器：

N3 接口（RAN 连接）：

- 接收 **N3**：从 gNB/eNodeB 接收到的数据包
- 发送 **N3**：发送到 gNB/eNodeB 的数据包

N6 接口（数据网络连接）：

- 接收 **N6**：从数据网络（互联网/IMS）接收到的数据包
- 发送 **N6**：发送到数据网络的数据包

总计：跨接口的汇总数据包计数

用途：监控流量平衡和接口特定问题

用例

监控流量负载：

1. 检查数据包接收/发送速率
2. 验证流量在两个方向上流动
3. 比较 N3 与 N6 流量（应大致相等）

检测数据包丢弃：

1. 检查丢弃的数据包计数器
2. 查看 XDP 丢弃计数器
3. 如果丢弃率高，调查日志中的原因

性能分析：

1. 监控 XDP 处理与通过的比率
2. 检查 XDP 中止（指示错误）
3. 验证 N3/N6 接口流量分布

容量规划：

1. 跟踪数据包速率随时间的变化
2. 与 UPF 容量限制进行比较
3. 如果接近限制，计划扩展

实时更新

统计信息每 5 秒自动刷新。

容量监控

URL: `/capacity`

特性

容量视图显示 UPF 数据平面中所有映射的 eBPF 映射使用和容量限制。

eBPF 映射使用表

所有 eBPF 映射及其使用信息的表：

列	描述
映射名称	eBPF 映射名称（例如，`uplink_pdr_map`、`far_map`）
已使用	当前映射中的条目数量
容量	映射中允许的最大条目数量
使用情况	带百分比的可视化进度条
键大小	映射键的大小（以字节为单位）
值大小	映射值的大小（以字节为单位）

彩色使用指示器

使用进度条根据利用率进行颜色编码：

- 绿色（**<50%**）：正常操作，容量充足
- 黄色（**50-70%**）：小心，监控增长
- 琥珀色（**70-90%**）：警告，计划增加容量
- 红色（**>90%**）：危急，需要立即采取行动

需要监控的关键映射

uplink_pdr_map:

- 存储按 TEID 键入的上行 PDR
- 每个上行流量流一个条目
- 关键：耗尽会阻止新会话的建立

downlink_pdr_map / downlink_pdr_map_ip6:

- 存储按 UE IP 地址键入的下行 PDR
- 每个 UE IPv4/IPv6 地址一个条目
- 关键：耗尽会阻止新会话的建立

far_map:

- 存储按 FAR ID 键入的转发动作规则
- 在多个 PDR 之间共享
- 高优先级：影响转发决策

qer_map:

- 存储按 QER ID 键入的 QoS 执行规则
- 中等优先级：影响 QoS，但不影响基本连接性

urr_map:

- 存储按 URR ID 键入的使用报告规则
- 低优先级：影响计费，但不影响连接性

用例

容量规划：

1. 监控映射使用趋势
2. 确定哪些映射增长最快
3. 在达到限制之前计划增加容量

防止会话建立失败：

1. 在预期流量激增之前检查 PDR 映射使用情况
2. 如果接近限制，增加映射容量
3. 在容量增加后监控以验证

故障排除会话失败：

1. 当会话建立失败时，检查容量视图
2. 如果 PDR 映射为红色（>90%），则容量已耗尽
3. 增加映射容量或清除过时会话

优化映射配置：

1. 查看键和值的大小

2. 计算每个映射的内存使用
3. 根据实际使用模式优化映射大小

容量配置

eBPF 映射容量在 UPF 启动时在 UPF 配置文件中配置。典型值：

- 小型部署：每个映射 10,000 - 100,000 条目
- 中型部署：每个映射 100,000 - 1,000,000 条目
- 大型部署：每个映射 1,000,000+ 条目

内存计算：

```
映射内存 = (键大小 + 值大小) × 容量
```

例如，一个具有 100 万条目和 64 字节值的 PDR 映射大约使用 64 MB 的内核内存。

实时更新

容量视图每 10 秒自动刷新。

配置视图

URL: `/upf_config`

特性

配置视图显示 UPF 操作参数和数据平面配置。

UPF 配置

显示静态 UPF 配置：

- **PFCP** 接口：SMF/PGW-C 连接的 IP 地址和端口
- **N3** 接口：RAN（gNB/eNodeB）连接的 IP 地址
- **N6** 接口：数据网络连接的 IP 地址

- **N9** 接口：UPF 之间通信的 IP 地址（可选）
- **API** 端口：REST API 监听端口
- 版本：OmniUPF 软件版本

数据平面（**eBPF**）配置

显示活动的运行时数据平面参数：

- 活动 **N3** 地址：运行时 N3 接口绑定
- 活动 **N9** 地址：运行时 N9 接口绑定（如果启用）

这些值反映实际的 eBPF 数据路径配置，如果接口已更改，可能与静态配置不同。

用例

验证 **UPF** 连接性：

1. 检查 N3 接口 IP 是否与 gNB 配置匹配
2. 验证 N6 接口是否可以路由到数据网络
3. 确认 PFCP 接口是否可从 SMF 访问

故障排除接口问题：

1. 将静态配置与数据平面活动地址进行比较
2. 验证接口是否正确绑定
3. 检查接口配置更改

档和审计：

1. 记录 UPF 配置以便文档使用
2. 验证部署是否符合设计规范
3. 审计接口分配

路由视图

URL: `/routes`

特性

路由视图□供对用户设备（UE）IP 路由和路由协议会话（OSPF 和 BGP）的全面监控。

路由状态概述

仪表板显示汇总的路由统计信息：

- 状态：路由启用或禁用
- 总路由：UE IP 路由的总数
- 已同步：成功同步的路由数量
- 失败：未能同步的路由数量

活动 **UE IP** 路由

显示所有活动用户设备 IP 路由的表：

列	描述
索引	路由索引号
UE IP 地址	分配给 UE 的 IPv4 或 IPv6 地址

目的：

- 查看所有已配置路由的 UE IP 地址
- 验证路由分发到路由协议
- 监控路由同步状态

OSPF 邻居

OSPF（开放最短路径优先）协议邻居的表：

列	描述
邻居 **ID**	OSPF 路由器标识符
地址	OSPF 邻居的 IP 地址
接口	用于 OSPF 邻接的接口
状态	OSPF 邻接状态（完全、初始化等）
优先级	OSPF 优先级值
运行时间	邻居已运行的时间
死亡时间	邻居被视为死亡的时间

OSPF 状态:

- 完全（绿色）：完全邻接并交换路由信息
- 其他状态（黄色）：邻接形成或不完整

BGP 对等体

BGP（边界网关协议）对等体的表:

□	□□
□□ **IP**	BGP □□□□ IP □□
ASN	□□□□□□□□□□
□□	BGP □□□□□□□□□□□□□□
□/□	◆◆□□□□□□□□
□□□□□	□□□□□□□□□□□□□□
□□□□□	□□□□□□□□ BGP □□
□□□□□	□□□□□□□□ BGP □□

BGP □□□

- □□□□□□□□□□□ BGP □□□□□□□
- □□□□□□□□□□□□□□□□□□□

□□□□□□□ BGP □□□ ID □ ASN□□ BGP □□□□□

OSPF □□□□

□□□□□□ UE □□□ OSPF □□ LSA□□□□□□□□□□

列	描述
链路状态 **ID**	LSA 标识符（通常是网络地址）
掩码	路由的网络掩码
广告路由器	广告此外部路由的路由器 ID
度量类型	OSPF 外部度量类型（E1 或 E2）
度量	路由的 OSPF 成本度量
年龄	LSA 生成后的时间（秒）
序列号	LSA 的版本序列号

目的：

- 验证 UE 路由是否被重分发到 OSPF
- 监控哪个路由器正在广告外部路由
- 跟踪 LSA 的老化和更新

路由控制操作

同步路由按钮：

- 手动触发路由同步到 FRR（自由范围路由）
- 强制更新路由协议与当前 UE 路由
- 在配置更改后或从同步失败中恢复时非常有用

刷新按钮：

- 手动刷新所有路由信息
- 更新 OSPF 邻居、BGP 对等体和路由表

用例

监控路由协议健康：

1. 导航到路由视图
2. 检查 OSPF 邻居状态（应为“完全”）
3. 验证 BGP 对等体为“已建立”
4. 确认预期的邻居/对等体数量

验证 **UE** 路由分发：

1. 检查活动 UE IP 路由表中的特定 UE
2. 滚动到 OSPF 重分发路由部分
3. 验证 UE 路由是否出现在外部 LSA 中
4. 确认广告路由器与预期的 UPF 匹配

故障排除路由同步问题：

1. 检查状态概述中的已同步与失败计数器
2. 如果路由失败，点击“同步路由”按钮
3. 如果同步失败，监控红色横幅中的错误消息
4. 检查各自部分中的 OSPF/BGP 错误消息

验证多 **UPF** 部署：

1. 从下拉列表中选择不同的 UPF 实例
2. 比较各实例之间的路由计数
3. 验证 OSPF 邻居是否互相可见
4. 检查 BGP 对等关系

监控路由扩展：

1. 随着 UE 会话的增加，跟踪总路由计数
2. 验证路由是否分发到路由协议
3. 监控 OSPF LSA 数量增长
4. 检查 BGP 接收到的前缀计数

实时更新

路由视图每 10 秒自动刷新一次，以显示当前路由协议状态和 UE 路由。

路由集成

路由视图与在 UPF 上运行的 FRR（自由范围路由）集成：

- **OSPF**：路由作为外部类型 2 LSA 被重分发
- **BGP**：路由被广告给配置的 BGP 对等体
- 同步机制：REST API 调用触发 vtysh 命令以更新 FRR

XDP 能力视图

URL：`/xdp_capabilities`

特性

XDP 能力视图显示 eXpress Data Path (XDP) 模式支持、性能能力和 UPF 数据平面的吞吐量计算。

接口配置

显示网络接口和驱动程序信息：

字段	描述
接口名称	用于 XDP 的网络接口（例如，eth0、ens1f0）
驱动程序	网络驱动程序名称（例如，i40e、ixgbe、virtio_net）
驱动程序版本	驱动程序版本字符串
当前模式	活动的 XDP 模式（DRV、SKB 或 NONE）
多队列计数	用于并行处理的 NIC 队列对数

XDP □□

□□□□□□□□ XDP □□□□□□□□□□□□□□□□□

XDP_DRV□□□□□□□□

- □□□~5-10 Mpps□□□□□□□□□□
- □□□□□□□□□□□□ XDP □□□□□□□□
- □□□□□□□□ XDP □□□ NIC □□□□□i40e□ixgbe□mlx5 □□
- □□□□□□□□□□□□ XDP □□□□□□□
- □□□□□□□□□□□□□□□□□✓□□□□□□□□□□□□ X□✗□□

XDP_SKB□□□□□□□□

- □□□~1-2 Mpps
- □□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□
- □□□□□□□□
- □□□□□□□□□□✓□

□□□□□□□□□

- □□□□ XDP □□□□□□□□
- □□□□□□□□□□

□□□□□□□□□

- □□□□□□□□□□□□□“□□”□□□□□□□
- □□□□□□□□□□□□ XDP □□□□□□□□□□□

XDP □□□□□□□□□□□□□□□□□□□□□□ Mpps □□□□□□

推荐

该视图根据当前配置显示彩色推荐横幅：

绿色（最佳）：

- "✓ 最佳：启用 XDP_DRV 模式并具有原生驱动程序支持"
- 活动的是最高性能模式

黄色（警告）：

- "⚠ 考虑升级到 XDP_DRV 模式以获得更好的性能"
- 在驱动模式可用时以通用模式运行
- "⚠ 警告：此驱动程序不支持 XDP_DRV"
- 硬件限制阻止最佳性能

蓝色（信息）：

- 有关 XDP 配置的一般信息

Mpps □□□□□

□□□□□□□□□□□□□□□Mpps□□□□□□□□□Gbps□□

□□□□

□□□□□□□Mpps□□

- □□□0.1 - 100 Mpps
- □□��□□ XDP □□□□□□ Mpps
- □□□□□□□□□□□□□□

□□□□□□□□□□□□□

- □□□64 - 9000 □□
- □□□1200 □□□□□□ GTP □□□□
- □□□□□□□□□ GTP □□

□□□□□□□□

- **64B**□□□□□□□□□□□□□□□□
- **128B**□□□□□□
- **256B**□□□□□□□□□
- **512B**□□□□□□□□□□
- **1024B**□□□□□□
- **1518B**□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□Gbps□□

- □□□□□□□□□□□□□
- □□□`Gbps = Mpps × Packet_Size × 8 / 1000`
- □□ GTP□UDP□IP □□□□□

□□□□□□□Gbps□□

- □□□□□□□□□□□□

- □□□ ~50 □□□ GTP □□□□
- □□□`Gbps = Mpps × (Packet_Size - 50) / 1000`

□□□□□□

- □□ Mpps □□□□□□□□□□□□□□□□
- □□□10 Mpps = 10,000,000 □□□□□

□□□□□

- □□□□□□□□□
- □□□`10 Mpps × 1200 □□ × 8 □/□□ ÷ 1000 = 96 Gbps`

□□ Mpps

□□□□□□□□□□□□□□□□□

□□□ **Mpps**□

- □□□□□□□
- □□□□□□□□□□□□
- □□□□□□□□

□□□□□□□□

- □□□□□□□□□□□ Mpps = □□□ Gbps
- □□□□□□□□□□□ Mpps = □□□ Gbps
- □□□□□□□□□□□□□□□□

GTP □□□□□

- □□□□□14 □□
- IP □□20 □□□IPv4□□ 40 □□□IPv6□
- UDP □□8 □□
- GTP □□8 □□□□□□
- □□□□□□□□□□□□ ~50 □□

用例

评估 **XDP** 性能：

1. 导航到 XDP 能力视图
2. 检查当前 XDP 模式（应为 DRV 以获得最佳性能）
3. 注意 Mpps 性能范围
4. 查看推荐横幅

计算预期吞吐量：

1. 输入预期的数据包速率（Mpps）
2. 输入您的流量配置的平均数据包大小
3. 查看计算的吞吐量（Gbps）
4. 与链路容量或性能要求进行比较

优化 **XDP** 配置：

1. 检查是否支持但未激活 XDP_DRV 模式
2. 查看驱动程序版本和兼容性
3. 如果可用，请遵循建议升级到驱动程序模式
4. 验证多队列计数是否与 CPU 核心匹配

容量规划：

1. 使用计算器确定目标吞吐量所需的 Mpps
2. 与当前 XDP 模式能力进行比较
3. 确定是否需要硬件升级
4. 规划新部署的接口和驱动程序选择

故障排除性能问题：

1. 验证 XDP 模式为 DRV，而不是 SKB
2. 检查驱动程序版本以获取已知性能问题
3. 验证多队列计数是否足够
4. 计算当前模式是否支持所需的吞吐量

性能优化提示

驱动模式（**XDP_DRV**）：

- 使用具有原生 XDP 支持的 NIC（Intel i40e/ixgbe、Mellanox mlx5）
- 更新 NIC 驱动程序到最新版本
- 启用多队列（RSS）以进行并行处理
- 调整 NIC 环形缓冲区大小

通用模式（**XDP_SKB**）：

- 适合开发和测试
- 不推荐用于生产高吞吐量
- 考虑硬件升级以用于生产部署

多队列配置：

- 队列数量应与 CPU 核心数匹配或超过
- 启用跨核心的并行数据包处理
- 通过 RSS（接收侧负载均衡）分配负载

实时更新

XDP 能力视图每 30 秒刷新一次，以更新接口状态和模式信息。

日志查看器

URL: `/logs`

特性

实时查看 OmniUPF 应用程序日志。

特性：

- 通过 Phoenix LiveView 实现实时日志流
- 日志生成时实时更新

- 可滚动的日志历史
- 在活动会话期间用于故障排除

日志级别

OmniUPF 日志使用标准 Elixir Logger 级别:

- **DEBUG**: 详细的诊断信息
- **INFO**: 一般信息消息（默认）
- **WARNING**: 非关键问题的警告消息
- **ERROR**: 失败的错误消息

用例

故障排除会话建立:

1. 打开日志视图
2. 从 SMF 启动会话建立
3. 观察 PFCP 消息日志和任何错误

监控 **PFCP** 通信:

1. 查看 PFCP 关联设置消息
2. 跟踪会话创建/修改/删除
3. 验证心跳消息

调试转发问题:

1. 查找数据包处理错误
2. 检查 eBPF 映射操作日志
3. 确定 FAR/PDR 配置问题

最佳实践

操作指南

监控：

- 定期检查容量视图以防止映射耗尽
- 监控统计信息以发现异常流量模式或丢包
- 跟踪会话计数随时间的增长
- 关注 XDP 处理错误

缓冲区管理：

- 在切换场景中监控缓冲区
- 如果数据包超过 TTL，请清除卡住的缓冲区
- 验证切换完成后禁用缓冲
- 使用“刷新”而不是“清除”以避免数据包丢失

会话管理：

- 使用过滤器快速定位特定 UE 会话
- 展开会话以验证规则配置
- 在多个 UPF 实例之间比较会话
- 在故障排除之前检查健康指示器

故障排除：

- 使用日志进行实时调试
- 检查会话视图以验证 UE 连接性
- 验证规则配置以获取流量流
- 监控统计信息以发现数据包丢失或转发错误

性能

- 控制面板自动刷新时间为 5-10 秒，具体取决于视图
- 大型会话列表可能需要时间加载

- 规则视图按活动条目（URR 的非零体积）进行过滤
- 缓冲操作在选定的 UPF 上立即执行

相关□档

- 规则管理指南 - PDR、FAR、QER、URR 配置
- 监控指南 - 统计、指标和容量规划
- 指标参考 - 完整的 Prometheus 指标参考
- **PFCP** 原因代码 - PFCP 错误代码和会话诊断
- **API** □档 - REST API 参考和分页
- 路由指南 - UE 路由和 FRR 集成详细信息
- **XDP** 模式指南 - 详细的 XDP 模式文档和 eBPF 信息
- 故障排除指南 - 常见问题和诊断
- **UPF** 操作指南 - 一般 UPF 操作和架构

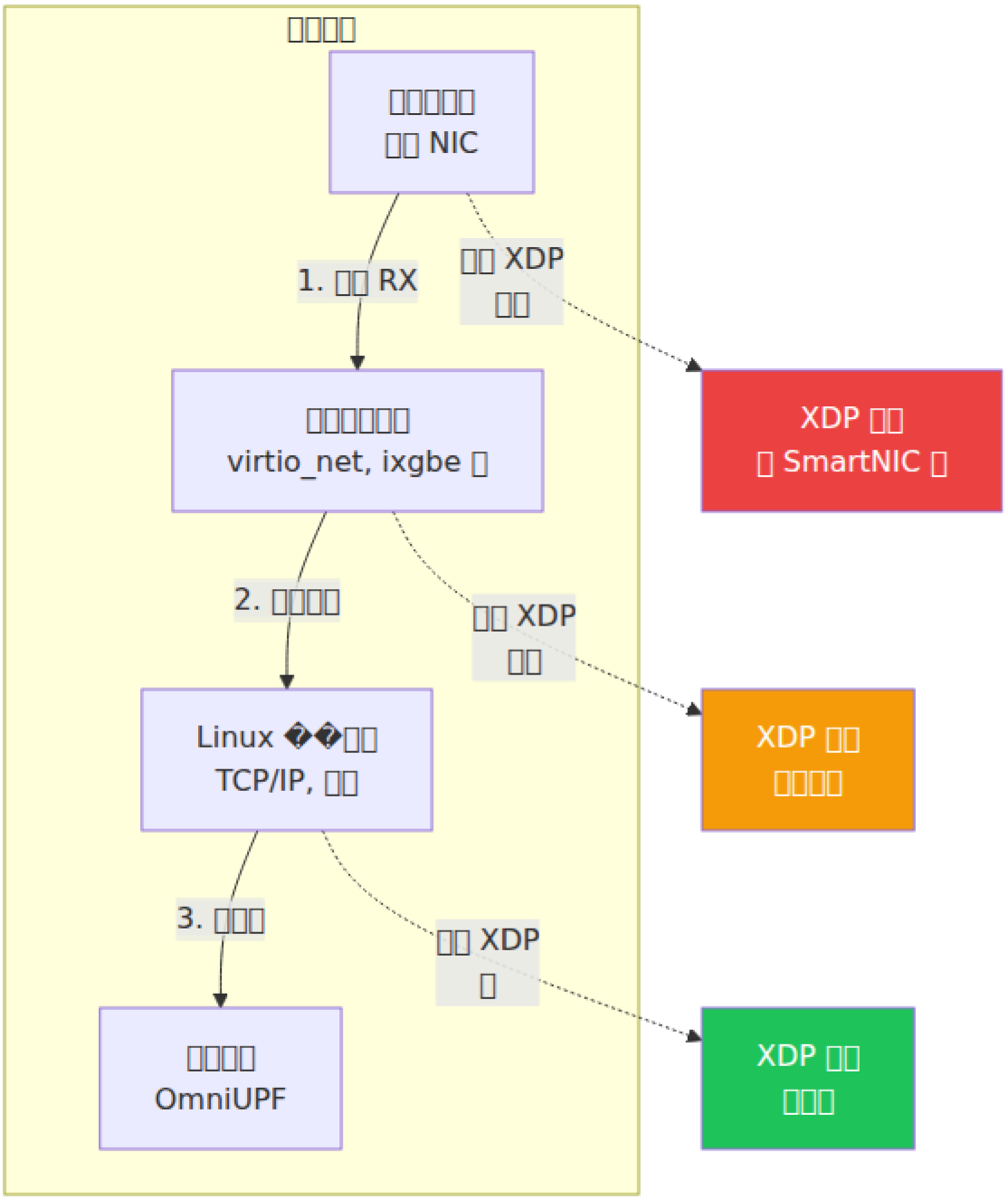
OmniUPF 的 XDP 附加模式

目录

概述

OmniUPF 使用 **XDP (eXpress Data Path)** 进行高性能数据包处理。XDP 是一种 Linux 内核技术，允许数据包处理程序（eBPF）在网络栈中尽早执行，从而□供微秒级延迟和每秒数百万个数据包的吞吐量。

XDP 附加模式决定了 **eBPF** 程序在数据包路径中的执行位置：



选择正确的 XDP 模式对 OmniUPF 性能有显著影响，并决定您是否能够实现生产级的数据包处理。

XDP 模式比较

方面	通用模式	本地模式	卸载模式
附加点	Linux 网络栈	网络驱动程序	NIC 硬件
性能	~1-2 Mpps	~5-10 Mpps	~10-40 Mpps
延迟	~100 µs	~10 µs	~1 µs
CPU 使用率	高	中	低
NIC 要求	任何 NIC	支持 XDP 的驱动	支持 XDP 的 SmartNIC
虚拟机监控程序支持	所有虚拟机监控程序	大多数（需要多队列）	稀有（PCI 直通）
使用案例	测试、开发	生产（推荐）	高吞吐量边缘站点
配置	`xdp_attach_mode: generic`	`xdp_attach_mode: native`	`xdp_attach_mode: offload`

推荐：在生产部署中使用 本地模式。通用模式仅适合测试。

通用模式（默认）

描述

通用 XDP 在驱动程序处理数据包 后 在 Linux 网络栈中运行 eBPF 程序。这是最慢的 XDP 模式，但适用于任何网络接口。

性能特征

- 吞吐量：~1-2 百万数据包每秒 (Mpps)
- 延迟：每个数据包 ~100 微秒
- **CPU** 开销：高（数据包在 XDP 之前复制到内核栈）

何时使用

- 仅用于开发和测试
- 实验室环境，性能不重要
- 初始部署，在优化之前验证功能

配置

```yaml
# config.yaml
interface_name: [eth0]
xdp_attach_mode: generic  # 默认模式
```

警告：通用模式 不适合生产。在高数据包速率下会造成瓶颈并浪费 CPU 资源。

本地模式（推荐用于生产）

描述

本地 XDP 在网络驱动程序 内部 运行 eBPF 程序，在数据包到达 Linux 网络栈之前。这□供了接近硬件的性能，同时保持内核级的灵活性。

性能特征

- 吞吐量：每个核心 ~5-10 百万数据包每秒 (Mpps)
- 延迟：每个数据包 ~10 微秒
- **CPU** 开销：低（数据包在驱动级处理）
- 扩展性：��� CPU 核心和 NIC 队列线性扩展

何时使用

- 生产部署（推荐）
- 需要高吞吐量的运营商级网络
- 具有性能要求的边缘计算场景
- 任何对性能重要的部署

NIC 驱动要求

本地 XDP 需要支持 XDP 的网络驱动程序。大多数现代 NIC 支持本地 XDP:

物理 **NIC**（裸金属）：

- 英特尔: `ixgbe`（10G）、`i40e`（40G）、`ice`（100G）
- 博通: `bnxt_en`
- 梅兰克斯: `mlx4_en`、`mlx5_core`
- Netronome: `nfp`（具有卸载支持）
- Marvell: `mvneta`、`mvpp2`

虚拟 **NIC**（虚拟机监控程序）：

- VirtIO: `virtio_net`（KVM、Proxmox、OpenStack）✓
- VMware: `vmxnet3` ✓
- 微软: `hv_netvsc`（Hyper-V）✓
- 亚马逊: `ena`（AWS）✓
- SR-IOV: `ixgbevf`、`i40evf`（PCI 直通）✓

注意: VirtualBox 不支持本地 XDP（仅使用通用模式）。

配置

```yaml
# config.yaml
interface_name: [eth0]
xdp_attach_mode: native
```

多队列要求: 为了获得最佳性能，请在虚拟 NIC 上启用多队列（请参见下面的 Proxmox 部分）。

卸载模式（SmartNIC）

描述

卸载 XDP 在 NIC 硬件（SmartNIC）上直接运行 eBPF 程序，完全绕过 CPU 进行数据包处理。这□供了最高的性能，但需要专��硬件。

性能特征

- 吞吐量：~10-40 百万数据包每秒 (Mpps)
- 延迟：每个数据包 ~1 微秒
- **CPU** 开销：接近零（在 NIC 上处理）

何时使用

- 超高吞吐量 部署（每个 UPF 实例 10G+）
- 具有硬件加速的边缘站点
- 对成本敏感的 部署（减少 CPU 要求）

硬件要求

目前只有 Netronome Agilio SmartNIC 支持 XDP 卸载：

- Netronome Agilio CX 10G/25G/40G/100G

注意：卸载模式需要 裸金属 或 **PCI** 直通 - 在标准虚拟机配置中不可用。

配置

```
# config.yaml
interface_name: [eth0]
xdp_attach_mode: offload
```

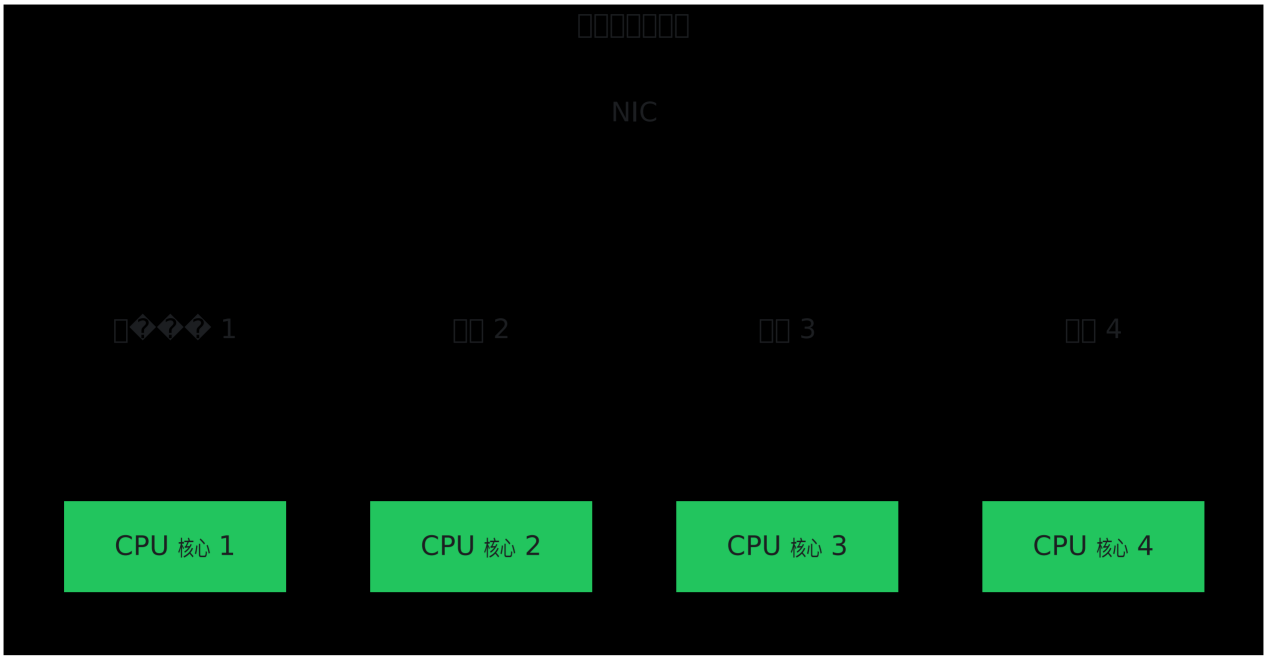
在 Proxmox VE 上启用本地 XDP

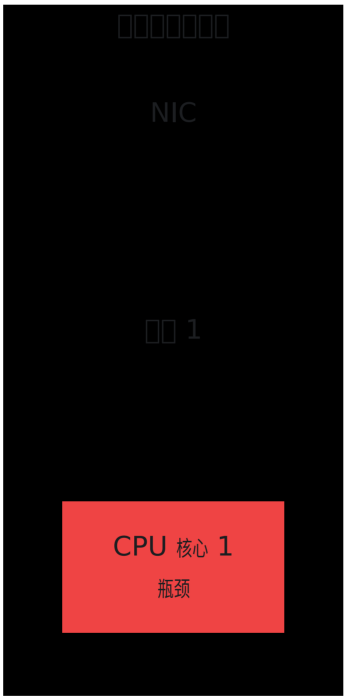
Proxmox VE 使用 **VirtIO** 网络设备用于虚拟机，这些设备通过 `virtio_net` 驱动程序支持本地 XDP。然而，您必须启用 多队列 以获得最佳性能。

步骤 1：了解要求

为什么多队列很重要：

- 单队列（默认）：所有网络流量由一个 CPU 核心处理 → 瓶颈
- 多队列：流量分布在多个 CPU 核心上 → 线性扩展





步骤 2: 在 Proxmox 中启用多队列

选项 A: 通过 Proxmox Web UI

1. 完全关闭虚拟机（不仅仅是重启）
    - 在 Proxmox Web 界面中选择您的虚拟机
    - 点击 关闭

2. □□□□□□

   - □□ □□ □□
   - □□□□□□□□□□□□□□□`net0`□
   - □□ □□

3. □□□□□

   - □□ "□□□" □□
   - □□□ **8**□□□□□ vCPU □□□□□□□ 16□
   - □□ □□

4. □□□□□

   - □□ □□

□□ **B**□□□ **Proxmox** □□□

```
# SSH □□□ Proxmox □□

# □□□□□□□ ID
qm list

# □□□□□□□ XXX □□□□□□□□ ID□
qm set XXX -net0 virtio=XX:XX:XX:XX:XX:XX,bridge=vmbr0,queues=8

# □□□□□□□□ 191□MAC □ BC:24:11:1D:BA:00
qm set 191 -net0 virtio=BC:24:11:1D:BA:00,bridge=vmbr0,queues=8

# ��□□□
qm shutdown XXX

# □□□□□□□□□
qm start XXX
```

□□□□□□□

- **4** □□□□□□□□□□□□□□□ 2-4 vCPU □□□□
- **8** □□□□□□□□□□□□□□□□4-8 vCPU □□□□
- **16** □□□□□□□□□□□□□□8+ vCPU □□□□

步骤 3：在虚拟机内验证多队列

虚拟机重启后，SSH 进入虚拟机并验证：

```
# 检查队列配置
ethtool -l eth0

# 预期输出：
# eth0 的通道参数：
# Combined:     8        <-- 应与您配置的值匹配

# 计算实际队列
ls -1d /sys/class/net/eth0/queues/rx-* | wc -l
ls -1d /sys/class/net/eth0/queues/tx-* | wc -l

# 两者应显示 8（或您配置的值）
```

步骤 4：在 **OmniUPF** 中启用本地 **XDP**

编辑 OmniUPF 配置：

```
# 编辑配置文件
sudo nano /config.yaml
```

更改 XDP 模式：

```
# 之前
xdp_attach_mode: generic

# 之后
xdp_attach_mode: native
```

重启 OmniUPF：

```
sudo systemctl restart omniupf
```

步骤 5：验证本地 XDP 是否处于活动状态

检查日志：

```
# 查看启动日志
journalctl -u omniupf --since "1 minute ago" | grep -i
"xdp\|attach"

# 预期输出：
# xdp_attach_mode:native
# XDPAttachMode:native
# 附加 XDP 程序到接口 "eth0"（索引 2）
```

通过 API 检查：

```
# 查询配置
curl -s http://localhost:8080/api/v1/config | grep xdp_attach_mode

# 预期输出：
# "xdp_attach_mode": "native",
```

常见 Proxmox 问题

问题："无法附加 XDP 程序"

解决方案：

- 验证是否启用多队列（`ethtool -l eth0`）
- 检查内核版本：`uname -r`（必须 ≥ 5.15）
- 确保加载了 VirtIO 驱动程序：`lsmod | grep virtio_net`

问题：尽管配置了 1 个队列

解决方案：

- 虚拟机必须 完全关闭（而不是重启）以进行队列更改
- 使用 `qm shutdown XXX && sleep 5 && qm start XXX`
- 在 Proxmox 配置中验证：`grep net0 /etc/pve/qemu-server/XXX.conf`

问题：使用本地模式时性能没有改善

解决方案：

- 检查 CPU 绑定（避免过度分配）
- 监控 `top` - CPU 使用率应分布在多个核心上
- 验证 XDP 统计信息：`curl http://localhost:8080/api/v1/xdp_stats`

在其他虚拟机监控程序上启用本地 XDP

VMware ESXi / vSphere

VMware 使用 `vmxnet3` 驱动程序，支持本地 XDP。

要求：

- ESXi 6.7 或更高版本
- 虚拟机中 vmxnet3 驱动程序版本 1.4.16 及以上
- 虚拟机硬件版本 14 或更高

启用多队列：

1. 关闭虚拟机
2. 编辑虚拟机设置：
    - 右键单击虚拟机 → 编辑设置
    - 网络适配器 → 高级
    - 将 接收端负载均衡 设置为 启用
3. 编辑 **.vmx** 件（可选，更多队列）：

```
ethernet0.pnicFeatures = "4"
ethernet0.multiqueue = "8"
```

4. 启动虚拟机并验证：

```
ethtool -l ens192  # 检查队列数量
```

配置 **OmniUPF**:

```
interface_name: [ens192]  # VMware 通常使用 ens192
xdp_attach_mode: native
```

KVM / libvirt（原始）

通过 **virsh** 启用多队列:

```
# 编辑虚拟机配置
virsh edit your-vm-name
```

在网络接口部分添加:

```
<interface type='network'>
  <source network='default'/>
  <model type='virtio'/>
  <driver name='vhost' queues='8'/>
</interface>
```

重启虚拟机并验证:

```
ethtool -l eth0
```

Microsoft Hyper-V

Hyper-V 使用 `hv_netvsc` 驱动程序，支持本地 XDP。

要求:

- Windows Server 2016 或更高版本
- 虚拟机中 Linux 集成服务 4.3 及以上

- 第二代虚拟机

启用多队列:

在 Hyper-V 主机上使用 PowerShell:

```
# 设置 VMQ (虚拟机队列) - Hyper-V 的多队列
Set-VMNetworkAdapter -VMName "YourVM" -VrssEnabled $true -
VmmqEnabled $true
```

配置 **OmniUPF**:

```
interface_name: [eth0]
xdp_attach_mode: native
```

VirtualBox

警告: VirtualBox 不支持本地 XDP。

原因: VirtualBox 网络驱动程序 (e1000, virtio-net) 未实现 XDP 钩子。

解决方法: 仅使用通用模式:

```
xdp_attach_mode: generic  # VirtualBox 唯一选项
```

验证 XDP 模式

在配置本地 XDP 后, 验证其是否正常工作:

1. 检查 OmniUPF 日志

```
# 查看最近的日志
journalctl -u omniupf --since "5 minutes ago" | grep -i xdp

# 查找:
# ✓ "xdp_attach_mode:native"
# ✓ "附加 XDP 程序到接口"
# ✗ "无法附加" 或 "回退到通用"
```

2. 通过 API 检查

```
# 查询配置端点
curl -s http://localhost:8080/api/v1/config | jq .xdp_attach_mode

# 预期输出:
# "native"
```

3. 检查 XDP 统计信息

```
# 查看 XDP 处理统计信息
curl -s http://localhost:8080/api/v1/xdp_stats | jq

# 示例输出:
{
  "xdp_aborted": 0,      # 应为 0（错误）
  "xdp_drop": 1234,      # 丢弃的数据包
  "xdp_pass": 5678,      # 传递给栈的数据包
  "xdp_redirect": 9012,  # 重定向的数据包
  "xdp_tx": 3456         # 发送的数据包
}
```

4. □□□□□□

```
# □□□□□□□□□□ XDP
ethtool -i eth0 | grep driver

# □□ Proxmox/KVM□□□□ "virtio_net"
# □□ VMware□□□□ "vmxnet3"
# □□ Hyper-V□□□□ "hv_netvsc"
```

5. □□□□

□□□□□□□□□□□□□□□□

```
# □□□□□□□□
watch -n 1 'curl -s http://localhost:8080/api/v1/packet_stats | jq
.rx_packets'

# □□□□□□~1-2 Mpps
# □□□□□□~5-10 Mpps□□□ 5-10 □□
```

□□ XDP □□

□□□□□□ "□□□□ XDP □□"

□□□

```
□□□□□□□ XDP □□□□□□□□ eth0
```

□□□

1. □□□□□□□□

```
ethtool -i eth0 | grep driver

# □□□□□□□ virtio_net/vmxnet3/hv_netvsc□□□□□ XDP □□□□□□
```

2. □□□□□□□□

```
uname -r

# □□ ≥ 5.15 □□□□□□□ XDP
```

3. □□□□□□□□□□ **XDP** □□□

```
ip link show eth0 | grep xdp

# □□□□□□□□□ XDP □□□□□□□□□
ip link set dev eth0 xdp off
```

□□□□□

- □□□□□□□□□□□□ 5.15+
- □□□□□ virtio_net □□□□□`modprobe virtio_net`
- □□□□□□□□□□□□ XDP□□□□□□□□□□

□□□□□□□□□□□□□□

□□□

```
□□□□□□□□□ XDP □□
```

□□□

□□ `dmesg` □□□□□□□□

```
dmesg | grep -i xdp | tail -20
```

常见原因：

1. 驱动程序不支持本地 **XDP**：
    - VirtualBox 驱动程序（不支持本地 XDP）
    - 较旧的 NIC 驱动程序
2. 未启用多队列：
    - 检查：`ethtool -l eth0`
    - 应显示 > 1 的组合队列
3. 内核 **XDP** 支持被禁用：

```
# 检查内核中是否启用了 XDP
grep XDP /boot/config-$(uname -r)

# 应显示：
# CONFIG_XDP_SOCKETS=y
# CONFIG_BPF=y
```

解决方案：

- 启用多队列（请参见 Proxmox 部分）
- 更新到支持的驱动程序
- 如果必要，重新构建内核以支持 XDP

问题：使用本地模式时性能没有改善

症状：启用本地模式，但数据包速率与通用模式相同

诊断：

1. 验证多队列分布：

```
# □□□□□□□□□□□□□
ethtool -S eth0 | grep rx_queue

# □□□□□□□□□□□□□
```

2. □□ **CPU** □□□□

```
# □□□□□□□□ CPU □□□
mpstat -P ALL 1

# □□□□□□□□□□□□□ CPU □
```

3. □□ **XDP** □□□□□□□□□□□□□□□

```
# ��□ bpftool□□□□□□□
sudo bpftool net list

# □□□ XDP □□□□□
```

□□□□□

- □□□□□□□□8-16 □□□□
- □□ CPU □□□□□□□□□□
- □□□□□□□□□□□□ CPU □□□□

□□□XDP □□□□□□xdp_aborted > 0□

□□□

```
curl http://localhost:8080/api/v1/xdp_stats
{
  "xdp_aborted": 1234,  # □□□□□□
  ...
}
```

□□□

XDP 中止意味着 eBPF 程序在执行过程中遇到错误。

1. 检查 **eBPF** 验证器日志：

   ```
   dmesg | grep -i bpf | tail -20
   ```

2. 检查映射大小限制：

   ```
   # eBPF 映射可能已满
   curl http://localhost:8080/api/v1/map_info

   # 查找容量为 100% 的映射
   ```

解决方案：

- 增加配置中的 eBPF 映射大小
- 检查是否有损坏的数据包导致 eBPF 错误
- 验证 Linux 内核 eBPF 支持是否完整

问题：**Proxmox** 中多队列不起作用

症状：`ethtool -l eth0` 显示只有 1 个队列，尽管配置了多个

诊断：

1. 检查 **Proxmox** 虚拟机配置：

   ```
   # 在 Proxmox 主机上
   grep net0 /etc/pve/qemu-server/YOUR_VM_ID.conf

   # 应显示：queues=8
   ```

2. 验证虚拟机是否完全关闭：

```
# 在 Proxmox 主机上
qm status YOUR_VM_ID

# 必须显示 "status: stopped" 才能启动
```

解决方案：

```
# 在 Proxmox 主机上
# 强制关闭并重启
qm shutdown YOUR_VM_ID
sleep 10
qm start YOUR_VM_ID

# 然后在虚拟机内检查
ethtool -l eth0
```

重要：队列数量的更改需要 完全关闭虚拟机，而不仅仅是从虚拟机内部重启。

问题：附加 XDP 时权限被拒绝

症状：

```
错误：附加 XDP 程序时权限被拒绝
```

诊断：

XDP 操作需要 `CAP_NET_ADMIN` 和 `CAP_SYS_ADMIN` 权限。

解决方案：

1. 以 **root** 身份运行 **OmniUPF**（或使用权限）：

   ```
   sudo systemctl restart omniupf
   ```

2. 如果使用 **systemd**，验证服务文件具有权限：

```
# /lib/systemd/system/omniupf.service
[Service]
CapabilityBoundingSet=CAP_NET_ADMIN CAP_SYS_ADMIN CAP_NET_RAW
AmbientCapabilities=CAP_NET_ADMIN CAP_SYS_ADMIN CAP_NET_RAW
```

3. 如果使用 **Docker**，使用 `--privileged` 运行：

```
docker run --privileged -v /sys/fs/bpf:/sys/fs/bpf ...
```

性能影响总结

OmniUPF 数据包处理的实际性能比较：

场景	通用模式	本地模式	改进
数据包速率	1.5 Mpps	8.2 Mpps	提升 **5.5** 倍
延迟	95 µs	12 µs	降低 **8** 倍
CPU 使用率（**1 Gbps**）	85%（1 核心）	15%（分布式）	提高 **5** 倍效率
最大吞吐量	~1.2 Gbps	~10 Gbps	提高 **8** 倍

推荐：始终在生产部署中使用 本地模式，并启用 多队列。

XDP 硬件推荐

⚠ 重要：在购买任何硬件之前，请咨询 **Omnitouch** 支持以确认其与您的特定配置和部署要求 **100%** 兼容。

已知支持本地 XDP 模式的 NIC

这些 NIC 已验证支持 OmniUPF 的本地 XDP 模式：

□□□ **NIC**□□□□□□□□□□

□□	□□	□□□□	**XDP** □□	□□
Intel X520	10GbE	ixgbe	□□ ✓	□□□□□□□□□□□□□□□
Intel X710	10/40GbE	i40e	□□ ✓	□□□□□□□□
Intel E810	100GbE	ice	□□ ✓	□□□□□□□□□
Intel i350	1GbE	igb	□□ ✓□□□ 5.10+□	□□□□□□□□

□□□□/**NVIDIA NIC**□□□□□

□□	□□	□□□□	**XDP** □□	□□
ConnectX-4	25/50/100GbE	mlx5	□□ ✓	□□□□□□□□□□
ConnectX-5	25/50/100GbE	mlx5	□□ ✓	□□□□□□□□□
ConnectX-6	50/100/200GbE	mlx5	□□ ✓	□□□□□□□□□□□□
BlueField-2	100/200GbE	mlx5	□□ ✓	□□ DPU □□□ SmartNIC

□□ **NIC**

□□	□□	□□□□	**XDP** □□	□□
BCM57xxx □□	10/25/50GbE	bnxt_en	□□ ✓	□□□ Dell/HP □□□

□□ **NIC**□□□□□□□□

□□□□□□□□□□□□□□

平台	**NIC** 类型	驱动程序	**XDP** 支持	多队列	备注
Proxmox/KVM	VirtIO	virtio_net	本地 ✓	是（可配置）	最佳用于虚拟机
VMware ESXi	vmxnet3	vmxnet3	本地 ✓	是	需要 ESXi 6.7+
Hyper-V	合成 NIC	hv_netvsc	本地 ✓	是	Windows Server 2016+
AWS	ENA	ena	本地 ✓	是	EC2 裸金属实例
VirtualBox	任何	各种	□□□ □	否	不推荐用于生产

支持硬件卸载的 NIC

真正的 **XDP** 硬件卸载（eBPF 在 NIC 上运行）：

供应商	型号	速度	备注
Netronome	Agilio CX 10G	10GbE	仅确认支持 XDP 卸载
Netronome	Agilio CX 25G	25GbE	需要特殊固件
Netronome	Agilio CX 40G	40GbE	非常昂贵（~$2,500-5,000）
Netronome	Agilio CX 100G	100GbE	企业级仅

注意：硬件卸载 NIC 稀有且昂贵，需要裸金属部署。大多数部署应使用本地 XDP。

测试过的配置

这些配置已在生产中验证与 OmniUPF 一起使用：

预算选项（**1-10 Gbps**）

- **NIC**：Intel X520（10GbE 或以上）
- 支援模式：XDP
- 預期效能：UPF 單核 ~8-10 Gbps
- 成本：~$100-200（卡片/伺服器）

適合**10-50 Gbps**：

- **NIC**：Intel X710（40GbE）或 Mellanox ConnectX-4（25GbE）
- 支援模式：XDP
- 預期效能：UPF 單核 ~25-40 Gbps
- 成本：~$300-800

適合**50-100+ Gbps**：

- **NIC**：Mellanox ConnectX-5/6（100GbE）
- 支援模式：XDP
- 預期效能：UPF 單核 ~80-100 Gbps
- 成本：~$1,000-2,500

虛擬化環境（**Proxmox/KVM**）：

- **NIC**：VirtIO（8-16 佇列）
- 支援模式：XDP
- 預期效能：UPF 單核 ~5-10 Gbps
- 注意：虛擬化開銷較大

下一步行動

立即開始測試 OmniUPF 效能

NIC/□□	□□	□□□□
Realtek NICs	□ XDP □□□Linux □□□□□□	Intel i350 □□□
VirtualBox	□□□ XDP □□	□□□ Proxmox/KVM
□□□ **NICs**	□□□□□□□□□□□	□□□□ Intel/Mellanox
□□□□ **NICs**□<2014□	□ XDP □□□□□□	Intel X520 □□□

□□□□□□□

□□□□□□□□□□□□□□

1. □ □□□□□□□ Linux □□□□□□□□ XDP

   ```
   # □□□□□□□
   modinfo <driver_name> | grep -i xdp
   ```

2. □ □□□□□□□□□ ≥ 5.15 □□□□□□ XDP

   ```
   uname -r
   ```

3. □ □□□□□□ NIC □□□□□□□□□RSS/VMDq□

4. □ **PCI** □□□□□ PCIe □□□□□□□□□

   - 10GbE□PCIe 2.0 x4 □□
   - 40GbE□PCIe 3.0 x8 □□
   - 100GbE□PCIe 3.0 x16 □ PCIe 4.0 x8

5. □ □□□□□

   - □□□□□□□□ NIC
   - □□□□□□ VirtIO □ SR-IOV □□
   - □□□□□ NIC □□□□

⚠ 不要仅根据本指南购买硬件 - 始终先与 **Omnitouch** 支持确认！

其他资源

- 配置指南：CONFIGURATION.md - 完整配置参考
- 故障排除指南：TROUBLESHOOTING.md - 综合问题诊断
- 架构指南：ARCHITECTURE.md - eBPF 和 XDP 架构细节
- 监控指南：MONITORING.md - 性能监控和统计信息

快速参考

Proxmox 本地 XDP 设置（TL;DR）

```
# 在 Proxmox 主机上：
qm set <VM_ID> -net0 virtio=<MAC>,bridge=vmbr0,queues=8
qm shutdown <VM_ID> && sleep 10 && qm start <VM_ID>

# 在虚拟机内：
ethtool -l eth0  # 验证 8 个队列
sudo nano /etc/omniupf/config.yaml  # 设置：xdp_attach_mode: native
sudo systemctl restart omniupf
journalctl -u omniupf --since "1 min ago" | grep xdp  # 验证本地模式
```

验证 XDP 模式是否处于活动状态

```
# 检查配置
curl -s http://localhost:8080/api/v1/config | grep xdp_attach_mode

# 检查统计信息
curl -s http://localhost:8080/api/v1/xdp_stats | jq

# 检查队列
ethtool -l eth0
```

OmniUPF 操作指南

目录

概述

OmniUPF（基于 eBPF 的用户平面功能）是一种高性能的 5G/LTE 用户平面功能，□供运营商级的数据包转发、QoS 强制执行和移动网络的流量管理。基于 Linux eBPF（扩展伯克利包过滤器）技术构建，并增强了全面的管理能力，OmniUPF □供了 5G SA、5G NSA 和 LTE 网络所需的核心数据包处理基础设施。

什么是用户平面功能?

用户平面功能（UPF）是 3GPP 标准化的网络元素，负责 5G 和 LTE 网络中的数据包处理和转发。它□供:

- 高速数据包转发：在移动设备和数据网络之间
- 服务质量（**QoS**）：对不同流量类型的强制执行
- 基于数据包过滤器和规则的流量检测和路由
- 使用报告：用于计□□和分析
- 数据包缓冲：用于移动性和会话管理场景
- 合法拦截：支持合规性

OmniUPF 实现了 3GPP TS 23.501（5G）和 TS 23.401（LTE）中定义的完整 UPF 功能，□供了一个完整的、生产就绪的用户平面解决方案，使用 Linux 内核 eBPF 技术以实现最大性能。

OmniUPF 关键能力

数据包处理：

- 完全符合 3GPP 标准的用户平面数据包处理
- 基于 eBPF 的数据路径，□供内核级性能
- GTP-U（通用分组无线服务隧道协议）封装和解封装
- 支持 IPv4 和 IPv6 的接入和数据网络
- XDP（快速数据路径）用于超低延迟处理
- 多线程数据包处理

QoS 和流量管理：

- 带宽管理的 QoS 强制执行规则（QER）
- 流量分类的包检测规则（PDR）
- 路由决策的转发动作规则（FAR）
- 针对特定应用的服务数据流（SDF）过滤
- 用于流量跟踪和计费的使用报告规则（URR）

控制和管理：

- PFCP（数据包转发控制协议）接口与 SMF/PGW-C
- 用于监控和诊断的 RESTful API
- 实时统计和指标
- eBPF 映射容量监控
- 基于 Web 的控制面板

性能特性：

- 通过 eBPF 实现零拷贝数据包处理
- 内核级数据包转发（无用户空间开销）
- 多核可扩展性
- 支持硬件加速的卸载能力
- 针对云原生部署进行了优化

有关详细的控制面板使用，请参见 Web UI 操作。

理解用户平面架构

OmniUPF 是一个统一的用户平面解决方案，为 5G 独立（SA）、5G 非独立（NSA）和 4G LTE/EPC 网络提供运营商级的数据包转发。**OmniUPF** 是一个单一产品，可以同时作为：

- **UPF**（用户平面功能） - 5G/NSA 用户平面（通过 N4/PFCP 由 OmniSMF 控制）
- **PGW-U**（**PDN** 网关用户平面） - 4G EPC 网关到外部网络（通过 Sxc/PFCP 由 OmniPGW-C 控制）
- **SGW-U**（服务网关用户平面） - 4G EPC 服务网关（通过 Sxb/PFCP 由 OmniSGW-C 控制）

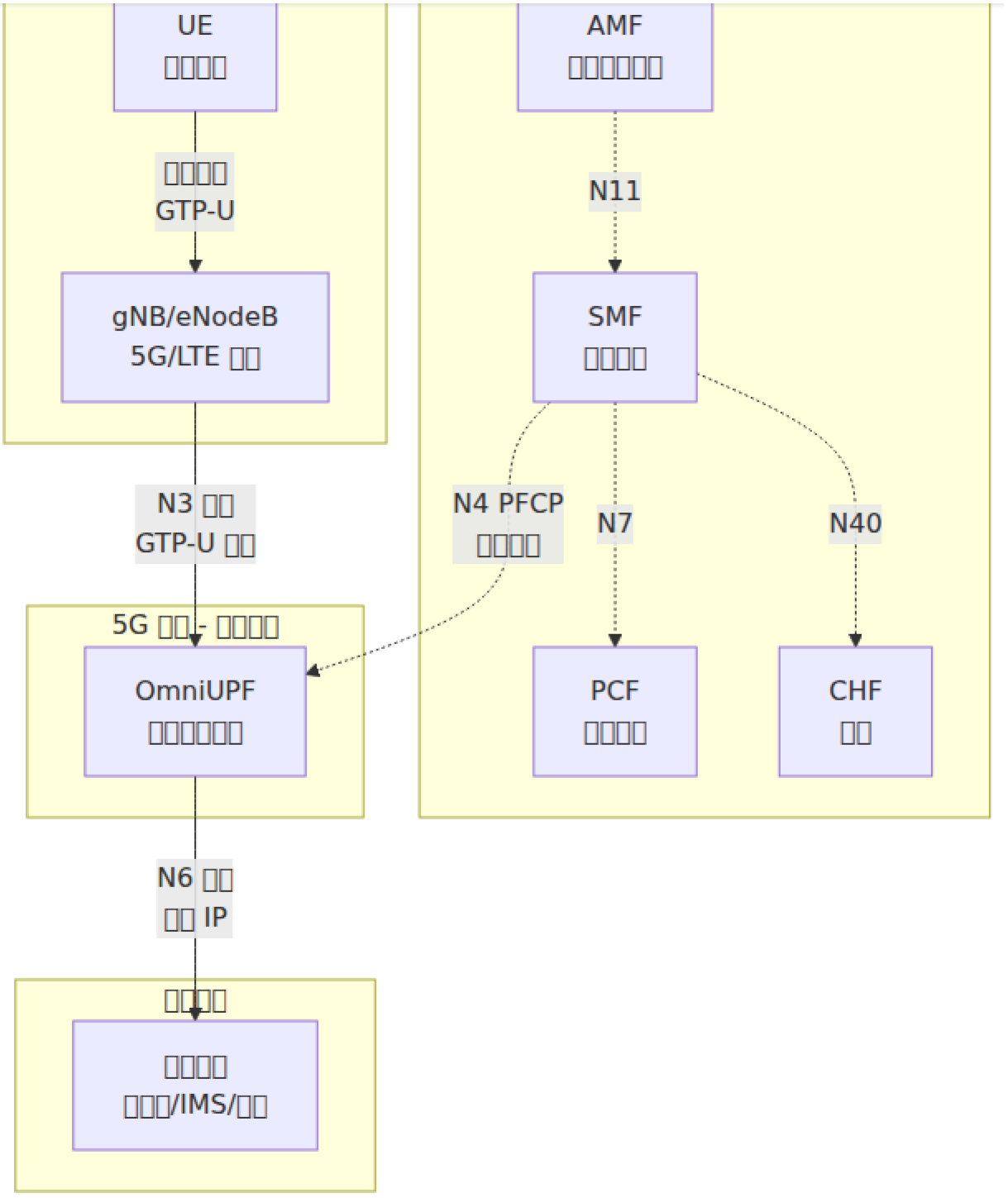
OmniUPF 可以在 任何组合 模式下运行：

- 仅 **UPF**：纯 5G 部署
- **PGW-U + SGW-U**：组合 4G 网关（典型 EPC 部署）
- **UPF + PGW-U + SGW-U**：同时支持 4G 和 5G（迁移场景）

所有模式使用相同的基于 eBPF 的数据包处理引擎和 PFCP 协议，无论作为 UPF、PGW-U、SGW-U，还是同时作为三者，均提供一致的高性能。

5G 网络架构（**SA** 模式）

OmniUPF 解决方案位于 5G 网络的数据平面，提供连接移动设备与数据网络和服务的高速数据包转发层。

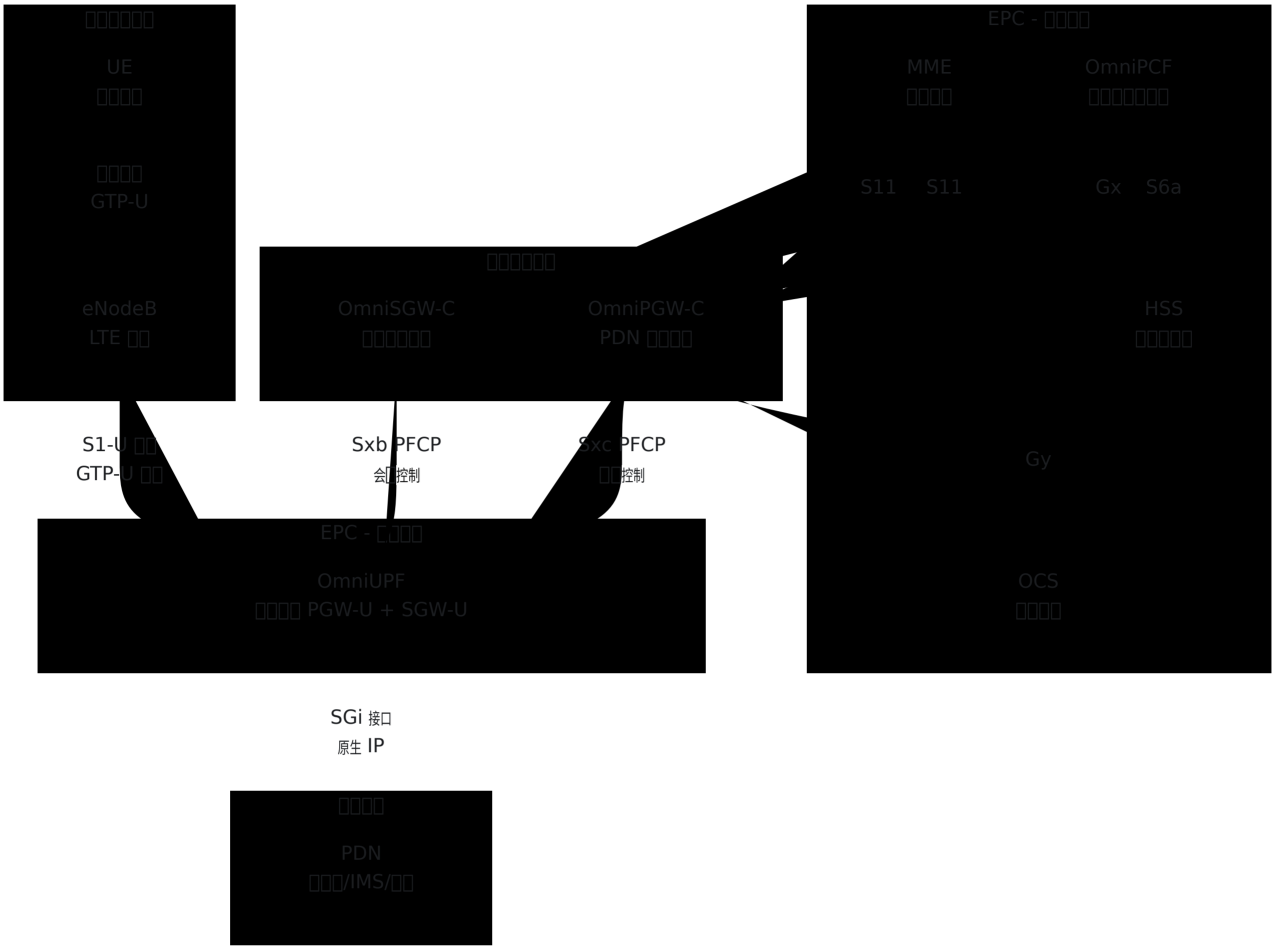


4G LTE/EPC 网络架构

OmniUPF 还支持 4G LTE 和 EPC（演进分组核心）部署，视网络架构的不同，作为 OmniPGW-U 或 OmniSGW-U 运行。

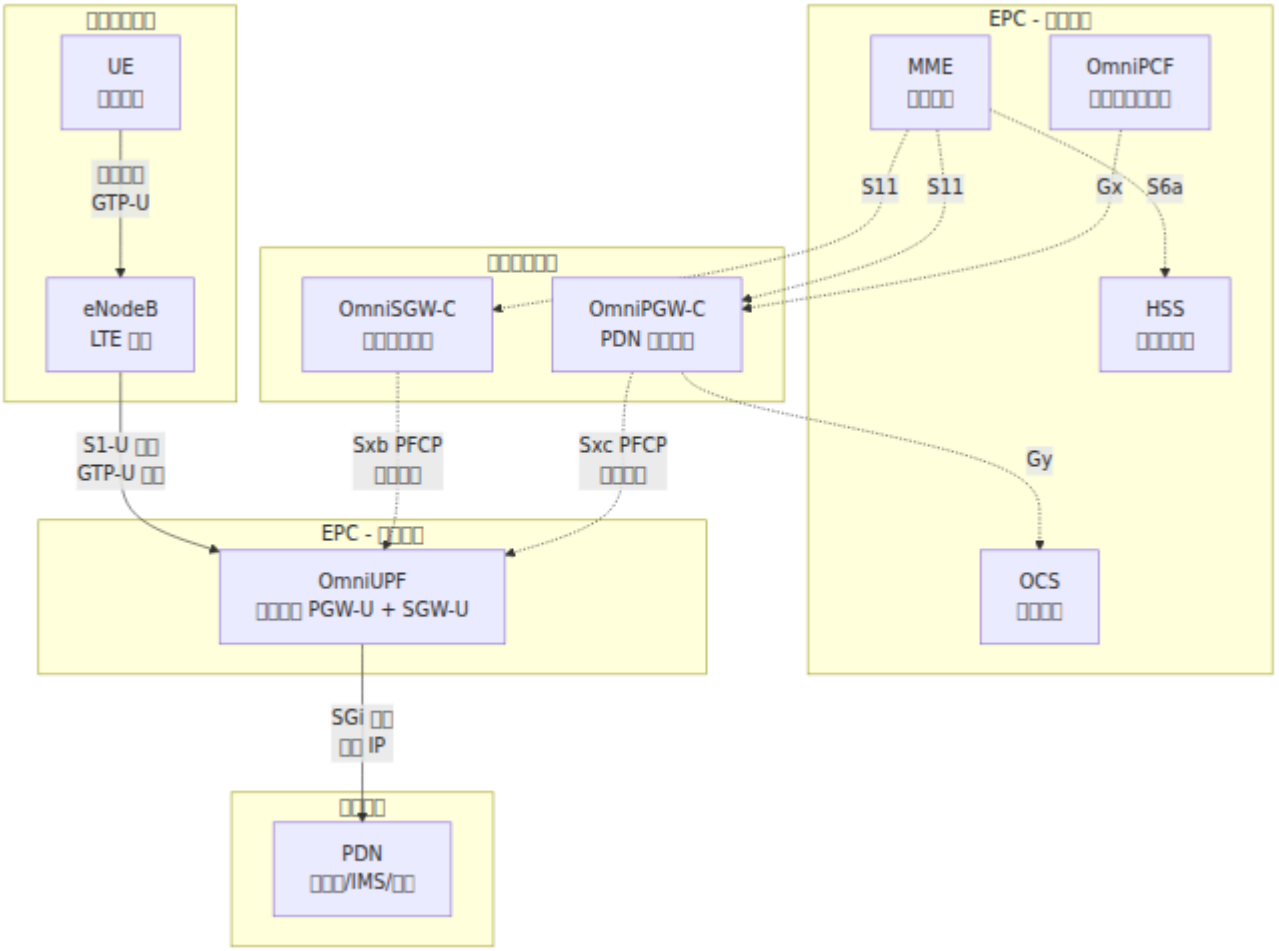
组合 **PGW-U/SGW-U** 模式（典型 **4G** 部署）

在此模式下，OmniUPF 同时作为 SGW-U 和 PGW-U，由不同的控制平面功能控制。



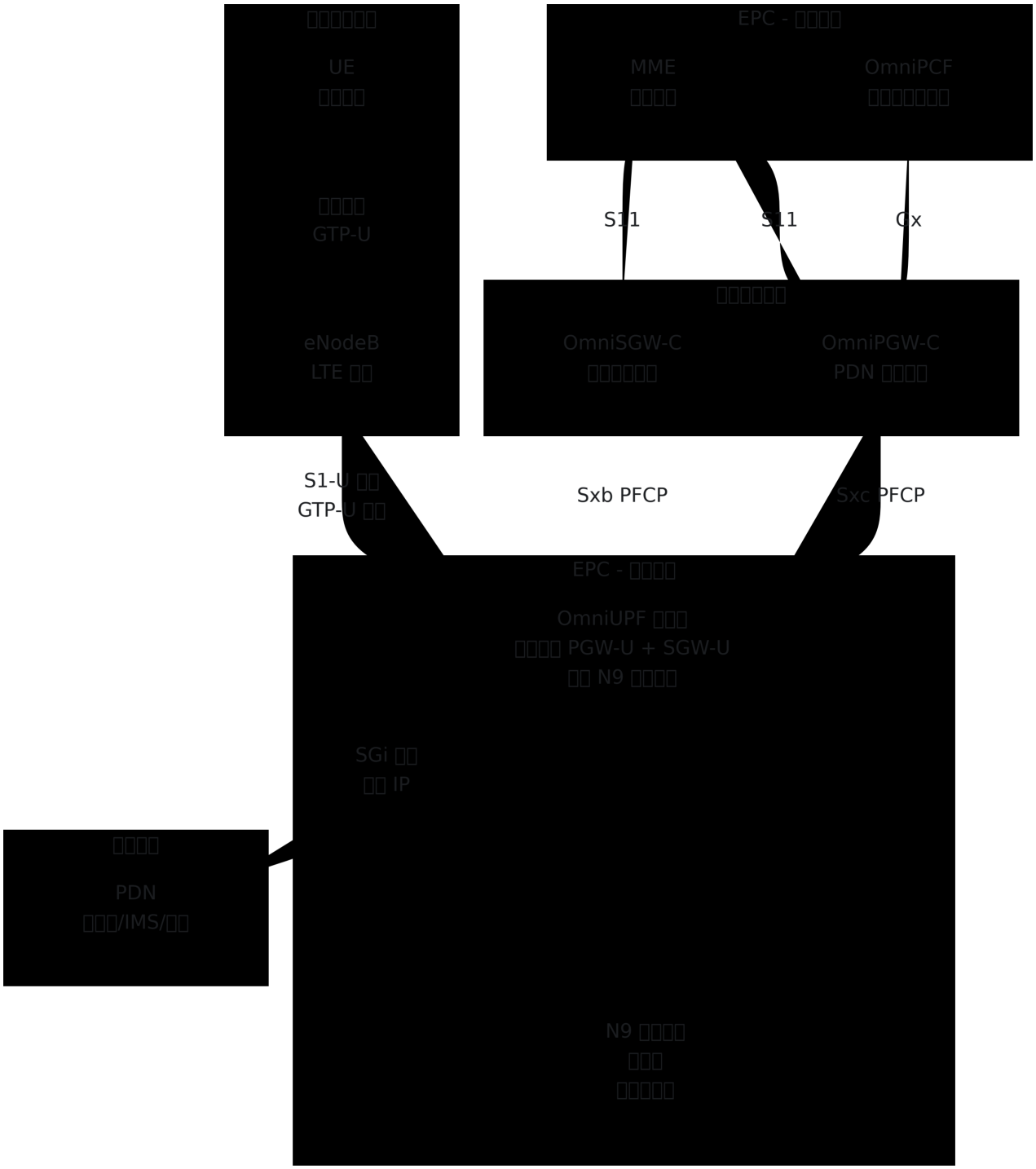
分离的 **SGW-U** 和 **PGW-U** 模式（漫游/多站点）

在漫游或多站点部署中，可以部署两个独立的 OmniUPF 实例 - 一个作为 SGW-U，另一个作为 PGW-U。



N9 循环回路模式（单实例 **SGWU+PGWU**）

对于简化的部署，OmniUPF 可以在单个实例上运行 **SGWU** 和 **PGWU** 角色，并在 eBPF 中完全处理 N9 循环回路。



关键特性：

- □ □□□ **N9** 延迟 - 完全在 eBPF 中处理，从不触及网络
- □ **40-50% CPU** 减少 - 单次 XDP 处理与两个独立实例相比
- □ □□□□ - 一个实例，一个配置文件
- □ □□□□ - 当 `n3_address` = `n9_address` 时，启用循环回路
- □ □□□□ **3GPP** 标准 - 标准 PFCP 和 GTP-U 协议

配置：

```yaml
# OmniUPF config.yml
interface_name: [eth0]
n3_address: "10.0.1.10"      # S1-U □□ IP
n9_address: "10.0.1.10"      # □□ IP □□ N9 □□□□
pfcp_address: ":8805"         # SGWU-C □ PGWU-C □□□□□
```

□□□□□

- □□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□
- □□□/□□□□□□□□
- □□□□□□□□□< 100K □□□□

□□□□□□

- □□□□□□□□SGWU □ PGWU □□□□□□
- □□□□□□□□□
- □□□□> 1M □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ **N9 ���□□□□□□□□**□

□□□□□□□□□□□□□□□

□□□□□□□□OmniUPF□OmniPGW-U □ OmniSGW-U□□□□□□□□□□□□□□□□□□□□□□□□

1. □□□□

   - **5G**□OmniSMF □□ N4 □□□ OmniUPF □□ PFCP □□
   - **4G**□OmniPGW-C □ OmniSGW-C □□ Sxb/Sxc □ OmniPGW-U/OmniSGW-U □ □ PFCP □□
   - □□□□□□□□ UE PDU □□□5G□□ PDP □□□□4G□□□ PFCP □□
   - □□□□□□□ PFCP □□ PDR□FAR□QER □ URR □□
   - eBPF □□□□□□□□□

2. □□□□□□□□□UE → □□□□□

   - **5G**□□□□□□□ N3 □□□ gNB □□□□□ GTP-U □□

    - **4G**：数据包通过 S1-U 接口（SGW-U）或 S5/S8 接口（PGW-U）从 eNodeB 到达，带有 GTP-U 封装
    - 用户平面根据 TEID 匹配上行数据包与 PDR
    - eBPF 程序应用 QER（速率限制、标记）
    - FAR 确定转发动作（转发、丢弃、缓冲、重复）
    - 移除 GTP-U 隧道，数据包转发到 N6（5G）或 SGi（4G）接口
    - URR 跟踪数据包和字节计数以进行计费

3. 下行数据包处理（数据网络 → UE）
    - **5G**：数据包通过 N6 接口作为原生 IP 到达
    - **4G**：数据包通过 SGi 接口作为原生 IP 到达
    - 用户平面根据 UE IP 地址匹配下行数据包与 PDR
    - SDF 过滤器可能进一步根据端口、协议或应用程序分类流量
    - FAR 确定 GTP-U 隧道和转发参数
    - 使用适当的 TEID 添加 GTP-U 封装
    - **5G**：数据包转发到 N3 接口朝向 gNB
    - **4G**：数据包转发到 S1-U（SGW-U）或 S5/S8（PGW-U）朝向 eNodeB

4. 移动性和切换
    - **5G**：在切换场景中，OmniSMF 更新 PDR/FAR 规则
    - **4G**：OmniSGW-C/OmniPGW-C 在 eNodeB 之间切换或 TAU（跟踪区域更新）期间更新规则
    - 用户平面在路径切换期间可能缓冲数据包
    - 在基站之间无缝过渡，无数据包丢失

与控制平面的集成（**4G** 和 **5G**）

OmniUPF 通过标准的 3GPP 接口与 5G 和 4G 控制平面功能集成：

5G 接口

接口	从 → 到	目的	**3GPP 规范**
N4	OmniSMF ↔ OmniUPF	PFCP 会话建立、修改、删除	TS 29.244
N3	gNB → OmniUPF	来自 RAN 的用户平面流量（GTP-U）	TS 29.281
N6	OmniUPF → 数据网络	到 DN 的用户平面流量（原生 IP）	TS 23.501
N9	OmniUPF ↔ OmniUPF	漫游/边缘的 UPF 之间通信	TS 23.501

4G/EPC 接口

接口	从 → 到	目的	**3GPP 规范**
Sxb	OmniSGW-C ↔ OmniUPF（SGW-U 模式）	服务网关的 PFCP 会话控制	TS 29.244
Sxc	OmniPGW-C ↔ OmniUPF（PGW-U 模式）	PDN 网关的 PFCP 会话控制	TS 29.244
S1-U	eNodeB → OmniUPF（SGW-U 模式）	来自 RAN 的用户平面流量（GTP-U）	TS 29.281
S5/S8	OmniUPF（SGW-U）↔ OmniUPF（PGW-U）	网关之间的用户平面（GTP-U）	TS 29.281
SGi	OmniUPF（PGW-U 模式）→ PDN	到数据网络的用户平面流量（原生 IP）	TS 23.401

注意：所有 PFCP 接口（N4、Sxb、Sxc）使用 TS 29.244 中定义的相同 PFCP 协议。接口名称不同，但协议和消息格式是相同的。

UPF □□

eBPF □□□□

eBPF □□□□□□□□□ Linux □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□

- **GTP-U** □□□GTP-U □□□□□□□□□□
- □□□□□□□□□ TEID□UE IP □ SDF □□□□□□□□□
- **QoS** □□□□□□□□ QER □□□□□□□□□□□□□□□□
- □□□□□□□□ FAR □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□ URR □□□

eBPF □□□ □□□□□□□ eBPF □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□	□□	□	□
`uplink_pdr_map`	□□ PDR	TEID□32 □□	PDR □□□FAR ID□QER ID□URR IDs□
`downlink_pdr_map`	□□ PDR □IPv4□	UE IP □□	PDR □□
`downlink_pdr_map_ip6`	□□ PDR □IPv6□	UE IPv6 □□	PDR □□
`far_map`	□□□□	FAR ID	□□□□□□□□□□□□□□□
`qer_map`	QoS □□	QER ID	QoS □□□MBR□GBR□□□□
`urr_map`	□□□□	URR ID	□□□□□□□□□□□□□□□□□
`sdf_filter_map`	SDF □□□	PDR ID	□□□□□□□□□□□□□□

□□□□□

- 零拷贝：数据包完全在内核空间处理
- **XDP** 支持：在网络驱动程序级别附加以实现亚微秒延迟
- 多核：在 CPU 核心之间扩展，支持每 CPU 映射
- 容量：在 eBPF 映射中可容纳数百万个 PDR/FAR（受限于内核内存）

有关容量监控，请参见 容量管理。

PFCP 接口处理程序

PFCP 接口实现 3GPP TS 29.244，用于与 SMF 或 PGW-C 的通信。

核心功能：

- 关联管理：PFCP 心跳和关联设置/释放
- 会话生命周期：创建、修改和删除 PFCP 会话
- 规则安装：将 PFCP 信息元素转换为 eBPF 映射条目
- 事件报告：通知 SMF 使用阈值、错误或会话事件

支持的 **PFCP** 消息：

消息类型	方向	目的
关联设置	SMF → UPF	建立 PFCP 控制关联
关联释放	SMF → UPF	拆除 PFCP 关联
心跳	双向	保持关联活跃
会话建立	SMF → UPF	创建新 PDU 会话，包含 PDR/FAR/QER/URR
会话修改	SMF → UPF	更新移动性、QoS 变更的规则
会话删除	SMF → UPF	移除会话及所有关联规则
会话报告	UPF → SMF	报告使用情况、错误或事件

支持的信息元素（**IE**）：

- 创建 PDR、FAR、QER、URR
- 更新 PDR、FAR、QER、URR
- 移除 PDR、FAR、QER、URR
- 数据包检测信息（UE IP、F-TEID、SDF 过滤器）
- 转发参数（网络实例、外部头创建）
- QoS 参数（MBR、GBR、QFI）
- 使用报告触发器（流量阈值、时间阈值）

REST API 服务器

REST API 供对 UPF 状态和操作的编程访问。

核心功能：

- 会话监控：查询活动的 PFCP 会话和关联
- 规则检查：查看 PDR、FAR、QER、URR 配置
- 统计信息：检索数据包计数器、路由统计、XDP 计
- 缓冲管理：查看和控制数据包缓冲
- 映射信息：监控 eBPF 映射的使用情况和容量

API 端点：（共 34 个端点）

类别	端点	描述
健康	`/health`	健康检查和状态
配置	`/config`	UPF 配置
会话	`/pfcp_sessions`, `/pfcp_associations`	PFCP 会话/关联数据
PDRs	`/uplink_pdr_map`, `/downlink_pdr_map`, `/downlink_pdr_map_ip6`, `/uplink_pdr_map_ip6`	数据包检测规则
FARs	`/far_map`	转发动作规则
QERs	`/qer_map`	QoS 强制执行规则
URRs	`/urr_map`	使用报告规则
缓冲	`/buffer`	数据包缓冲状态和控制
统计	`/packet_stats`, `/route_stats`, `/xdp_stats`, `/n3n6_stats`	性能指标
容量	`/map_info`	eBPF 映射容量和使用情况
数据平面	`/dataplane_config`	N3/N9 接口地址

有关 API 详细信息和使用，请参见 监控指南。

Web 控制面板

Web 控制面板□供 UPF 监控和管理的实时仪表板。

特性：

- 会话视图：浏览活动 PFCP 会话，显示 UE IP、TEID 和规则计数
- 规则管理：查看和管理所有会话中的 PDR、FAR、QER 和 URR
- 缓冲监控：跟踪缓冲的数据包并根据 FAR 控制缓冲
- 统计仪表板：实时数据包、路由、XDP 和 N3/N6 接口统计
- 容量监控：eBPF 映射使用情况，带有颜色编码的容量指示器
- 配置视图：显示 UPF 配置和数据平面地址
- 日志查看器：实时日志流以进行故障排除

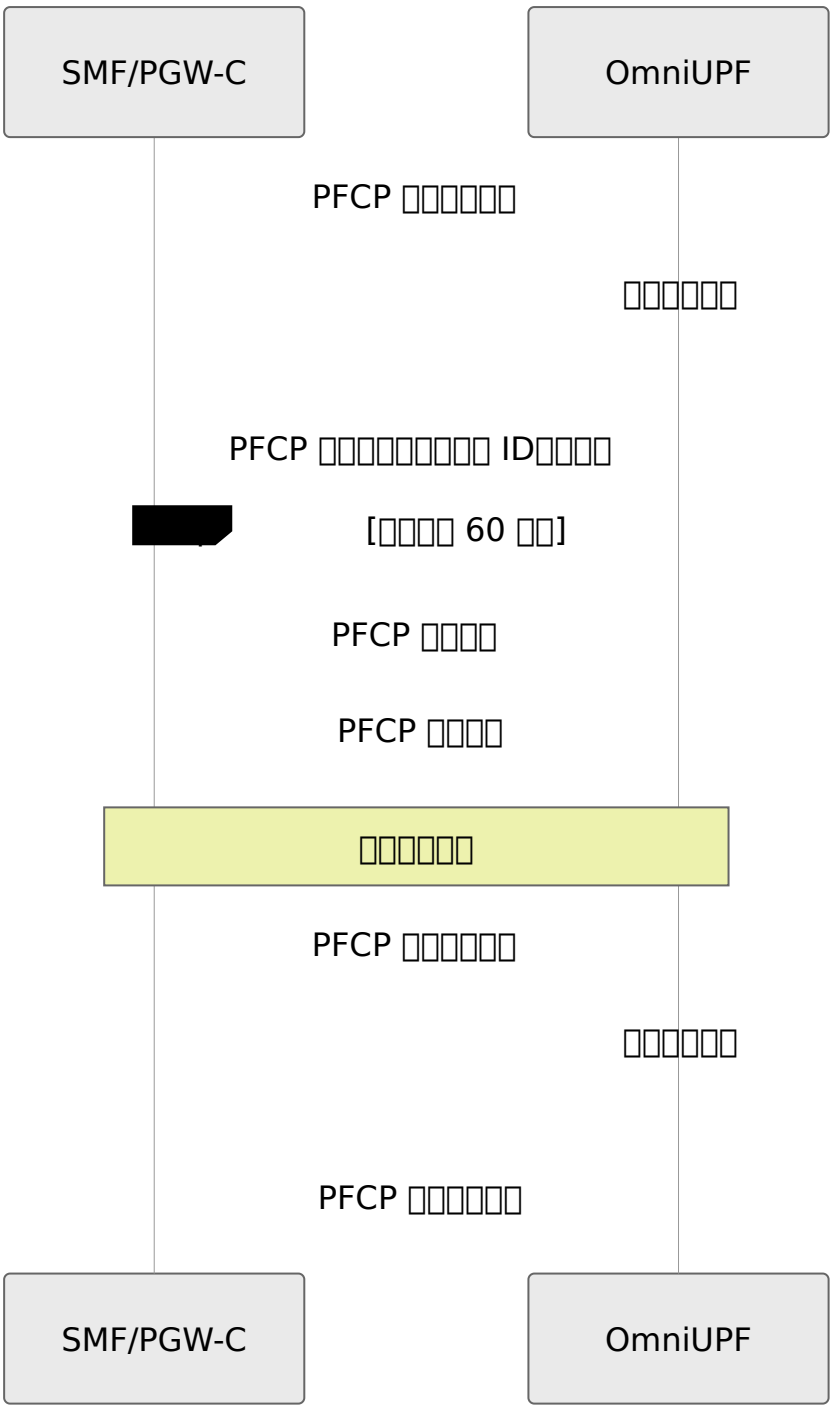
有关详细的 UI 操作，请参见 **Web UI** 操作指南。

PFCP 协议与 SMF 集成

PFCP 关联

在会话创建之前，SMF 必须与 UPF 建立 PFCP 关联。

关联生命周期：



□□□□

- □□ SMF □ UPF □□□□□□□
- UPF □□□□ ID□FQDN □ IP □□□□□□□
- □□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□ ��□□□□

SMF 重启检测和孤立会话清理

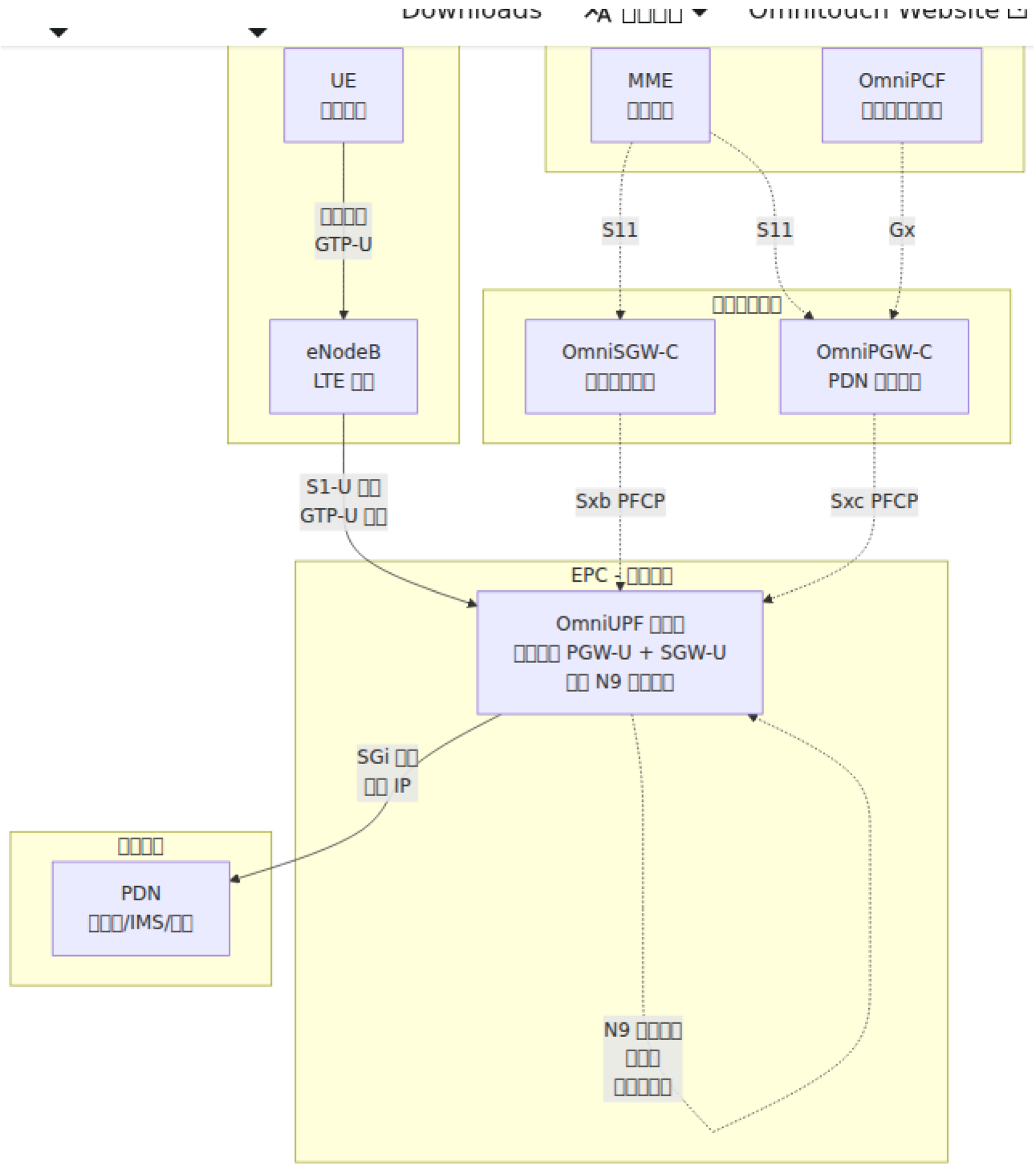
OmniUPF 自动检测 SMF 重启并根据 3GPP TS 29.244 规范清理孤立会话。

工作原理:

当 SMF 建立 PFCP 关联时，它供一个 恢复时间戳，指示其启动时间。OmniUPF 为每个关联存储此时间戳。如果 SMF 重启:

1. SMF 丢失内存中的所有会话状态
2. SMF 重新与 UPF 建立 PFCP 关联
3. SMF 发送 新的恢复时间戳（与之前不同）
4. UPF 检测到时间戳变化 = SMF 重启
5. UPF 自动删除 所有孤立会话，来自旧的 SMF 实例
6. SMF 为活动用户创建新的会话

重启检测流程:



日志示例:

当 SMF 重启时，您会看到:

```
WARN: 与 NodeID: smf-1 和地址: 192.168.1.10 的关联已存在
WARN: SMF 恢复时间戳已更改（旧: 2025-01-15T10:00:00Z, 新: 2025-01-15T10:30:15Z）
- SMF 重启，删除 245 个孤立会话
INFO: 删除孤立会话 2（LocalSEID）由于 SMF 重启
INFO: 删除孤立会话 3（LocalSEID）由于 SMF 重启
...
INFO: 删除孤立会话 246（LocalSEID）由于 SMF 重启
```

重要说明：

1. 隔离：仅删除重启的 SMF 的会话。其他 SMF 关联及其会话 不受影响。
2. 时间戳比较：如果恢复时间戳 相同，则会话 保留（SMF 重新连接而未重启）。
3. **3GPP** 合规性：此行为由 3GPP TS 29.244 第 5.22.2 节规定：

> “如果 CP 功能的恢复时间戳自上次关联设置以来发生变化，UP 功能应认为 CP 功能已重启，并应删除与该 CP 功能关联的所有 PFCP 会话。”

有关故障排除孤立��话的信息，请参见 孤立会话检测。

GTP-U 错误指示处理

OmniUPF 根据 3GPP TS 29.281 规范处理来自下游对等方（PGW-U、SGW-U、eNodeB、gNodeB）的 GTP-U 错误指示消息。

什么是错误指示：

当 OmniUPF 将 GTP-U 数据包转发到远程对等方（例如，SGW-U 部署中的 PGW-U）时，如果对等方不识别 TEID（隧道端点标识符），则可能会发送 错误指示。这表示：

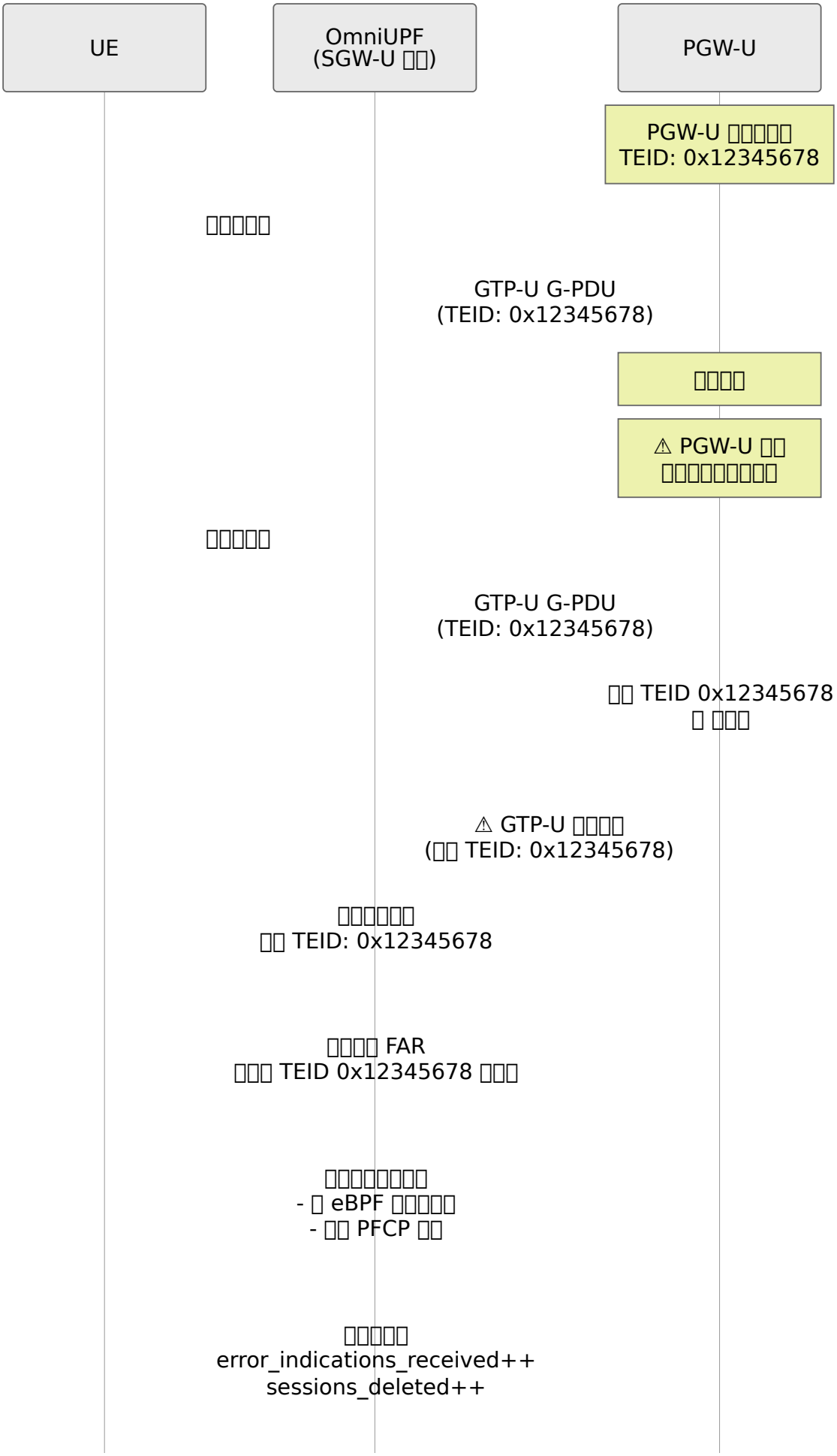
- 远程对等方已重启并丢失隧道状态
- 远程端未曾创建隧道（配置不匹配）
- 远程端已删除隧道

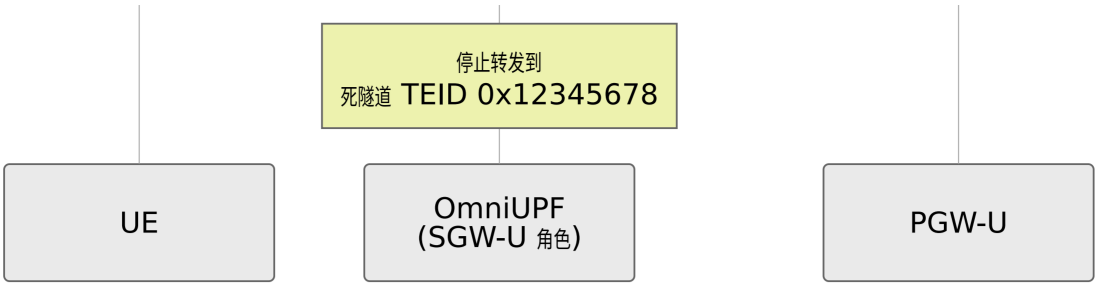
工作原理：

1. **UPF** 转发数据包 → 发送带有 TEID X 的 GTP-U 数据包到远程对等方（端口 2152）

2. 远程对等方不识别 **TEID X** → 在其隧道表中查找 TEID，未找到
3. 远程对等方发送错误指示 → GTP-U 消息类型 26，包含错误 TEID 的 IE
4. **UPF** 接收错误指示 → 解析消息以□取 TEID X
5. **UPF** 查找受影响的会话 → 搜索所有会话以查找转发到 TEID X 的 FAR
6. **UPF** 删除会话 → 从 eBPF 映射和 PFCP 状态中移除会话
7. **UPF** 更新指标 → 为监控递增 Prometheus 计数器

错误指示流程：


UE
OmniUPF
(SGW-U 역할)
PGW-U
PGW-U 세션 생성
TEID: 0x12345678
정상 상태
GTP-U G-PDU
(TEID: 0x12345678)
세션 삭제
⚠ PGW-U 재시
작으로 세션이 사라짐
정상 상태
GTP-U G-PDU
(TEID: 0x12345678)
알 수 TEID 0x12345678
로 패킷
⚠ GTP-U 패킷
(알 수 TEID: 0x12345678)
커널 데이터
알 수 TEID: 0x12345678
기본 FAR
로 TEID 0x12345678 전송
커널 데이터 처리
- 약 eBPF 프로그램
- 알 수 PFCP 세션
통계 업데이트
error_indications_received++
sessions_deleted++



数据包格式（3GPP TS 29.281 第 7.3.1 节）：

GTP-U 错误指示：

```
┌──────────────────────────────────────────────────┐
│ GTP-U 头部（12 字节）                            │
├──────────────────────────────────────────────────┤
│ 版本、PT、标志          │ 0x32                   │
│ 消息类型                │ 26 (0x1A)              │
│ 长度                    │ 9 字节                 │
│ TEID                    │ 0 (始终)               │
│ 序列号                  │ 变化                   │
│ N-PDU 编号              │ 0                      │
│ 下一个扩展头部          │ 0                      │
├──────────────────────────────────────────────────┤
│ IE: TEID 数据 I（5 字节）                        │
├──────────────────────────────────────────────────┤
│ 类型                    │ 16 (0x10)              │
│ 错误 TEID               │ 4 字节                 │
└──────────────────────────────────────────────────┘
```

何时重要：

场景 **1**：**PGW-U** 在 **S5/S8 GTP** 架构中重启

- SGW-U（OmniUPF）将 S5/S8 流量转发到 PGW-U
- PGW-U 重启并丢失所有 S5/S8 隧道状态
- SGW-U 继续转发到旧 TEID
- PGW-U 发送错误指示
- SGW-U 自动停止使用死隧道

场景 **2**：对等 **UPF** 在 **N9** 架构中重启

- UPF-1（OmniUPF）将 N9 流量转发到 UPF-2

- UPF-2 重启
- UPF-1 接收错误指示
- UPF-1 清理会话

日志示例：

当接收到错误指示时：

```
WARN: 从 192.168.50.10:2152 接收到 GTP-U 错误指示，TEID 0x12345678 - 远程对等方不识别此 TEID
WARN: 找到会话 LocalSEID=42，FAR GlobalId=1 转发到错误 TEID 0x12345678，来自对等方 192.168.50.10
INFO: 删除会话 LocalSEID=42，因来自 192.168.50.10 的 TEID 0x12345678 的 GTP-U 错误指示
WARN: 删除 1 个会话，因来自 192.168.50.10 的 TEID 0x12345678 的 GTP-U 错误指示
```

Prometheus 指标：

监控错误指示活动，按对等方和节点粒度：

```
# 从对等方接收到的错误指示总数
upf_buffer_listener_error_indications_received_total{node_id="pgw-u-1",peer_address="192.168.50.10"}

# 由于错误指示删除的会话
upf_buffer_listener_error_indication_sessions_deleted_total{node_id='
u-1",peer_address="192.168.50.10"}

# 发送的错误指示（对于未知的传入 TEID）
upf_buffer_listener_error_indications_sent_total{node_id="enodeb-1",peer_address="10.60.0.1"}
```

指标标签：

- `node_id`：来自关联的 PFCP 节点 ID（或在没有关联时为“未知”）
- `peer_address`：远程对等方的 IP 地址

这些指标有助于识别有问题的对等方并跟踪每个控制平面节点的错误指示模式。

重要说明：

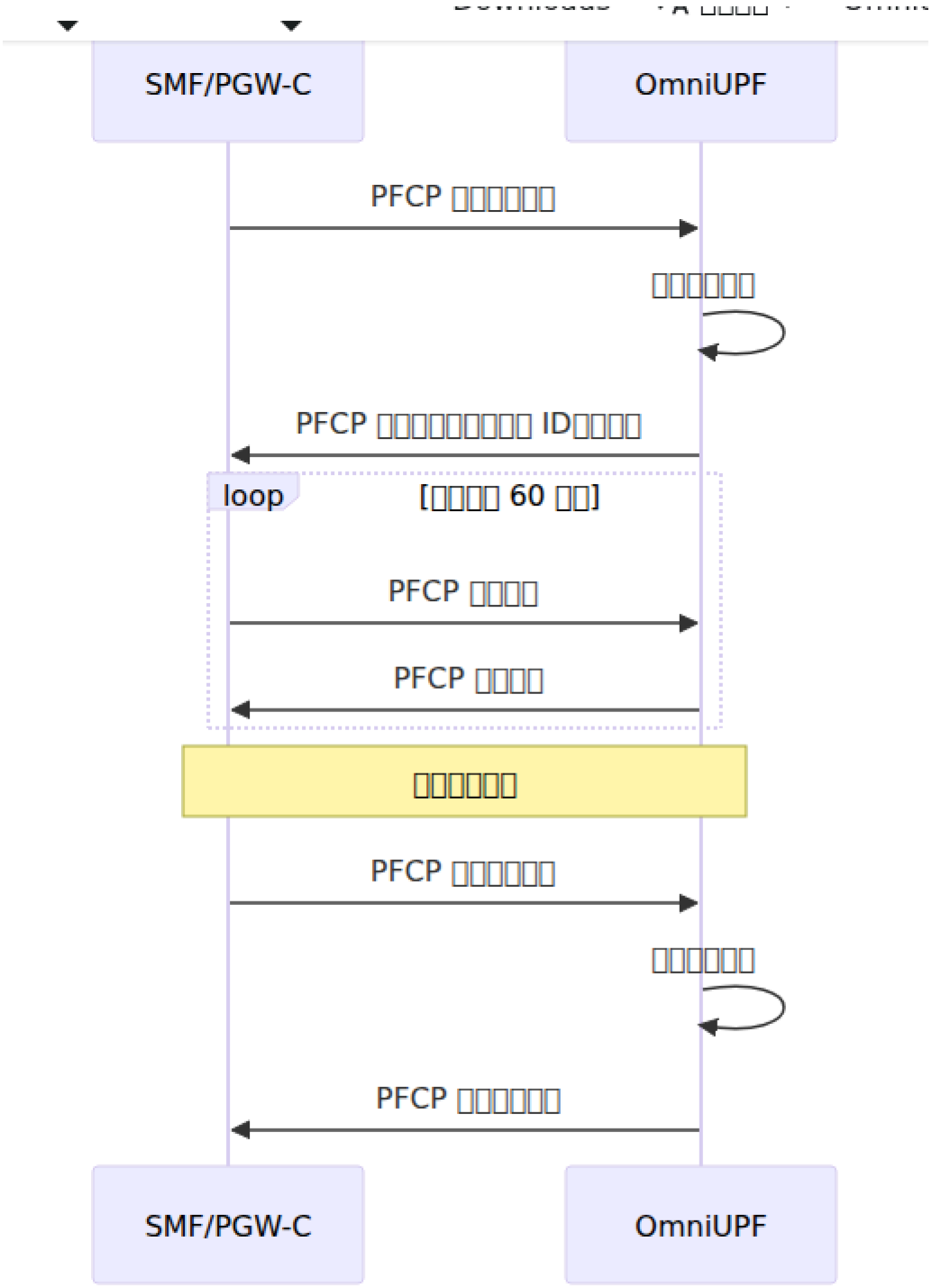
1. 自动清理：无需操作员干预 - 会话会自动删除
2. **TEID** 匹配：仅删除转发到确切错误 TEID 的会话
3. 每对等方隔离：来自一个对等方的错误指示仅影响转发到该对等方的会话
4. 多个会话：如果多个会话转发到同一死 TEID，所有会话都会被删除
5. 与恢复时间戳互补：
    - 恢复时间戳检测 = 主动（在关联设置期间检测重启）
    - 错误指示处理 = 被动（在流量流动时检测死隧道）
6. 格式错误的数据包处理：无效的错误指示被记录并忽略（不删除会话）

有关故障排除错误指示的信息，请参见 GTP-U 错误指示调试。

PFCP 会话创建

当 UE 建立 PDU 会话（5G）或 PDP 上下文（LTE）时，SMF 在 UPF 上创建 PFCP 会话。

会话建立流程：



典型会话内容：

- 上行 **PDR**：匹配 N3 TEID，通过 FAR 转发到 N6
- 下行 **PDR**：匹配 UE IP 地址，通过 FAR 转发到 N3，带有 GTP-U 封装
- **FAR**：转发参数（外部头创建、网络实例）
- **QER**：QoS 限制（MBR、GBR）和数据包标记（QFI）
- **URR**：用于计费的流量报告（可选）

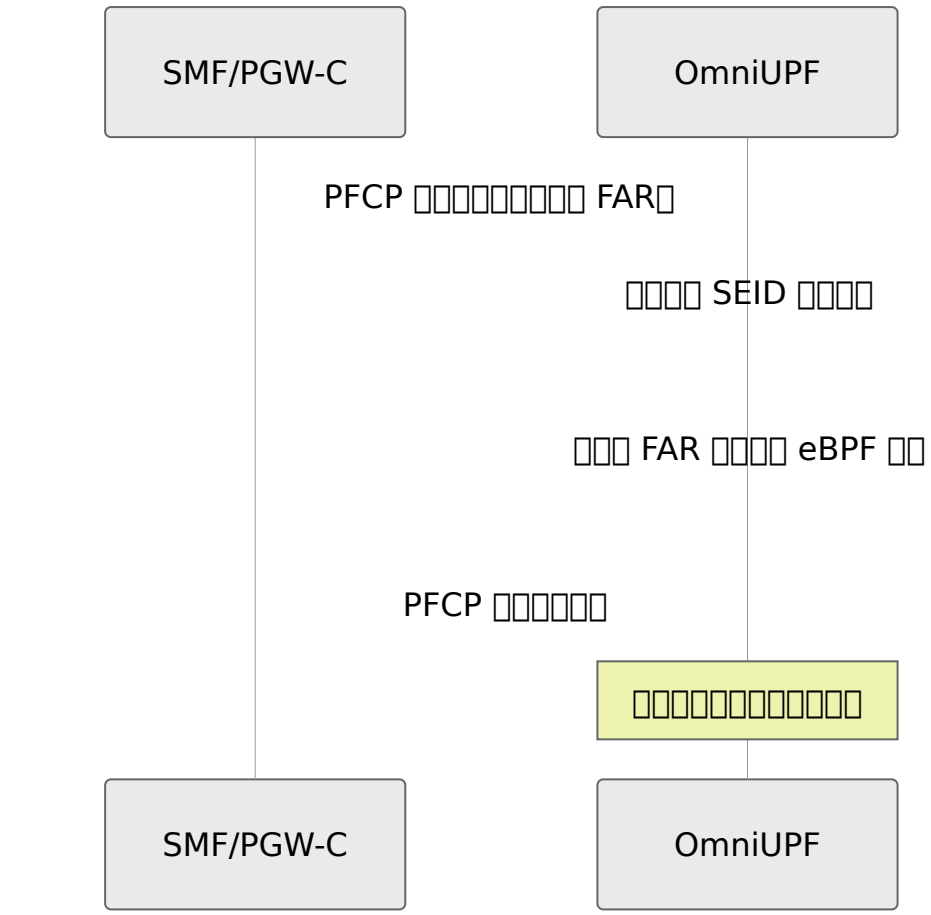
PFCP 会话修改

SMF 可以修改会话以应对移动事件（切换）、QoS 变化或服务更新。

常见修改场景：

1. 切换（基于 **N2**）
    - 使用新的 gNB 隧道端点（F-TEID）更新上行 FAR
    - 可选地在路径切换期间缓冲数据包
    - 在准备就绪时刷新缓冲区
2. **QoS** 变更
    - 使用新的 MBR/GBR 值更新 QER
    - 可在 PDR 中添加/删除 SDF 过滤器以实现特定应用的 QoS
3. 服务更新
    - 为额外流量流添加新的 PDR
    - 修改 FAR 以进行路由更改

会话修改流程：

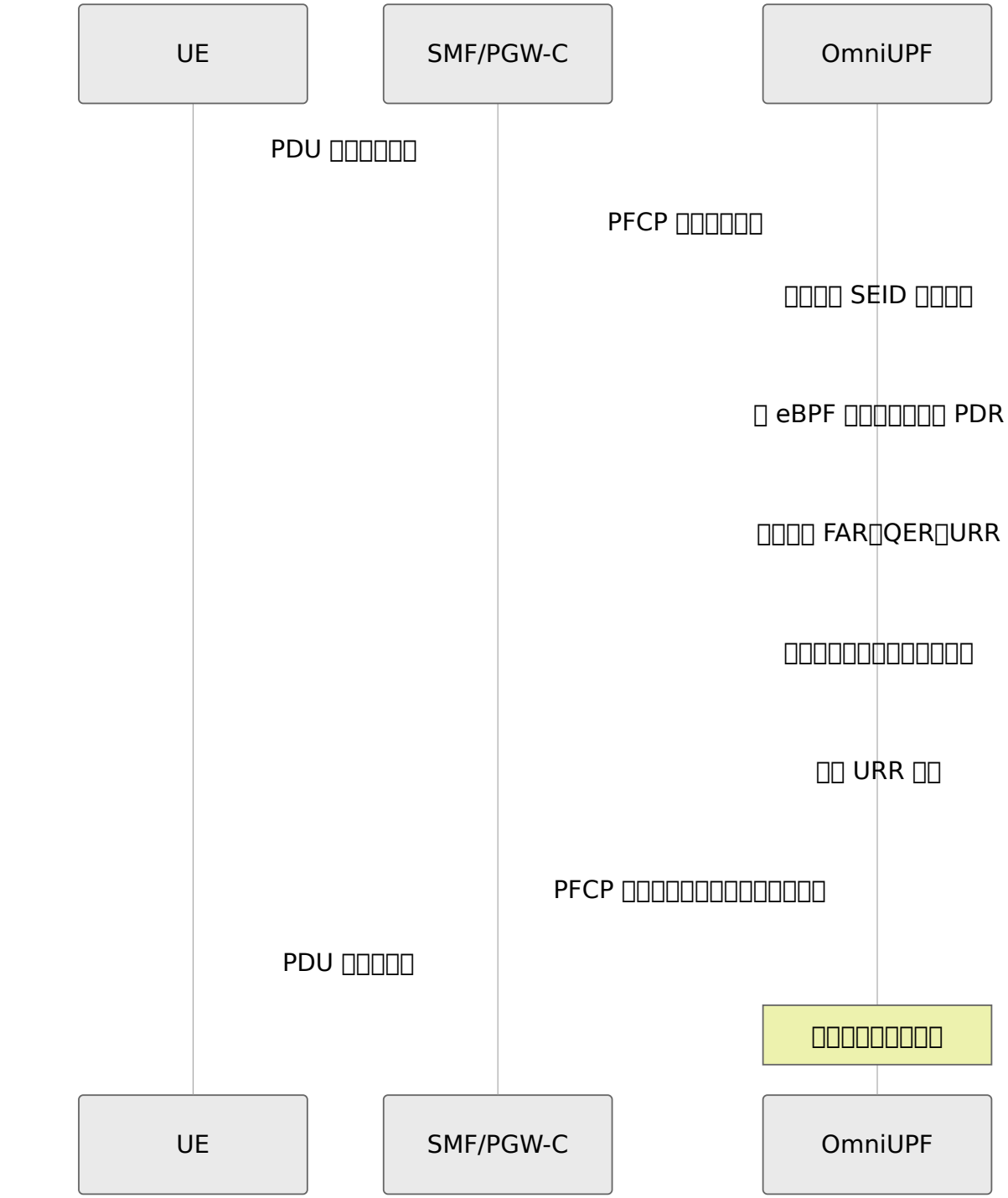


有关规则管理的信息，请参见 规则管理指南。

PFCP 会话删除

当 PDU 会话被释放时，SMF 删除 UPF 上的 PFCP 会话。

会话删除流程：



执行的清理:

- 移除所有 PDR（上行和下行）
- 移除所有 FAR、QER、URR
- 清除数据包缓冲
- 发送最终使用报告给 SMF 以进行计费

常见操作

OmniUPF 通过其基于 Web 的控制面板和 REST API 提供全面的操作能力。本节涵盖常见操作任务及其重要性。

会话监控

理解 **PFCP** 会话：

PFCP 会话代表活动的 UE PDU 会话（5G）或 PDP 上下文（LTE）。每个会话包含：

- 本地和远程 SEID（会话端点标识符）
- 用于数据包分类的 PDR
- 用于转发决策的 FAR
- QoS 强制执行的 QER（可选）
- 用于使用跟踪的 URR（可选）

关键会话操作：

- 查看所有会话，显示 UE IP 地址、TEID 和规则计数
- 按 **IP** 地址或 **TEID** 过滤会话
- 检查会话详细信息，包括完整的 PDR/FAR/QER/URR 配置
- 监控每个 **PFCP** 关联的会话计数

有关详细的会话程序，请参见 会话视图。

规则管理

数据包检测规则（**PDR**）：

PDR 确定哪些数据包匹配特定的流量流。操作员可以：

- 查看上行 **PDR**，以 N3 接口的 TEID 为键
- 查看下行 **PDR**，以 UE IP 地址（IPv4 和 IPv6）为键
- 检查 **SDF** 过滤器以进行特定应用的分类
- 监控 **PDR** 计数和容量使用情况

□□□□□□□□**FAR**□□

FAR □□□□□□□□□□□□□□□□□□□□□

- □□ **FAR** □□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□
- □□□□ **FAR** □□□□□□
- □□□□□□□□□□□□□□□ FAR

QoS □□□□□□□□**QER**□□

QER □□□□□□□□□□□□□□□□□□□□□

- □□ **QoS** □□□MBR□GBR□□□□□□□□□□
- □□□□□□□□□□ **QER**
- □□ **QFI** □□□□□ 5G QoS □

□□□□□□□□**URR**□□

URR □□□□□□□□□□□□□□□

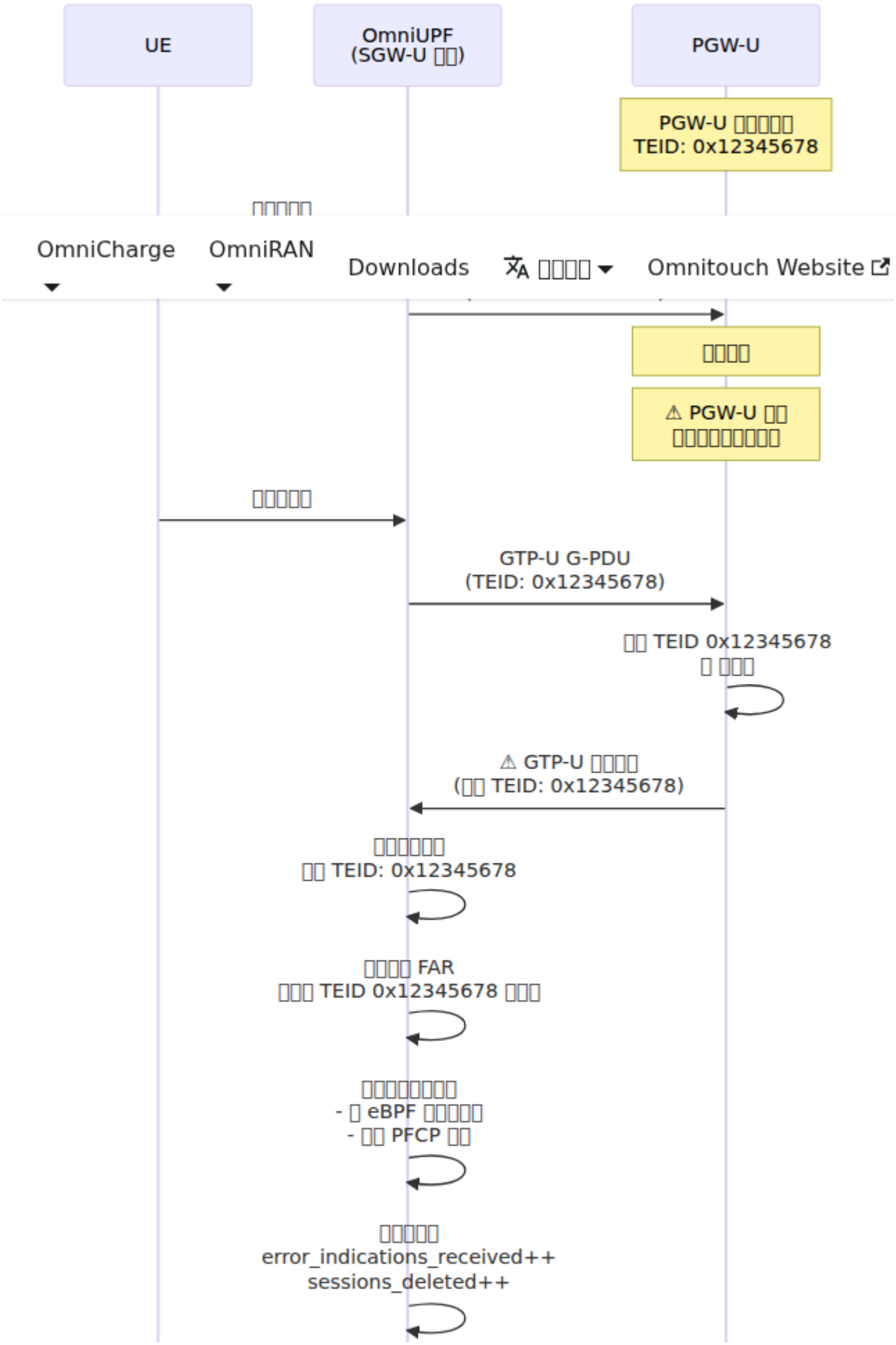
- □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□
- □□□□□□□□□□□ **URR**

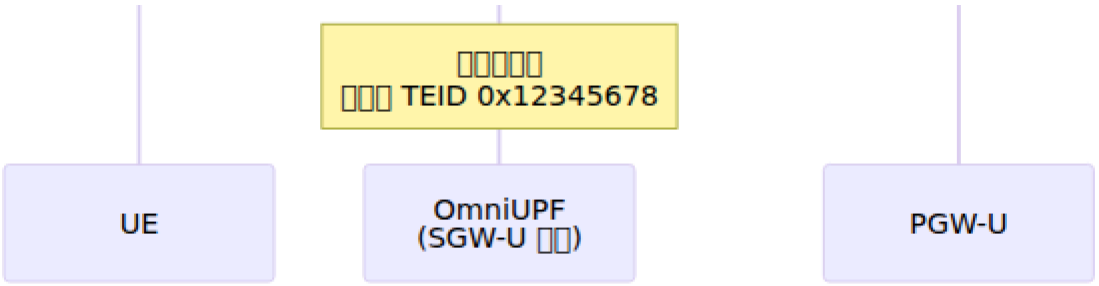
□□□□□□□□□□□□□□ □□□□□□□□

□□□□□

□□□□□□□ **UPF** □□□□

□□□□□□□ **UPF** □□□□□□□□□□□□□□□□□□□□


UE
OmniUPF
(SGW-U □□)
PGW-U
PGW-U □□□□□
TEID: 0x12345678
□□□□□
OmniCharge
OmniRAN
Downloads
□□□□
Omnitouch Website
□□□□
⚠ PGW-U □□
□□□□□□□□□□
□□□□□□
GTP-U G-PDU
(TEID: 0x12345678)
□□ TEID 0x12345678
□ □□□
⚠ GTP-U □□□□
(□□ TEID: 0x12345678)
□□□□□□□
□□ TEID: 0x12345678
□□□□ FAR
□□□ TEID 0x12345678 □□□
□□□□□□□□□
- □ eBPF □□□□□
- □□ PFCP □□
□□□□□□
error_indications_received++
sessions_deleted++



没有缓冲：在此关键窗口期间到达的 ~40ms 的数据包（可能是数千个）将被 丢弃，导致：

- **TCP** 连接停滞或重置（网页浏览、下载中断）
- 视频通话冻结或掉线（Zoom、Teams、WhatsApp 通话失败）
- 游戏会话断开（在线游戏、实时应用失败）
- **VoIP** 通话出现间隙或完全掉线（电话通话中断）
- 下载失败并需要重新开始

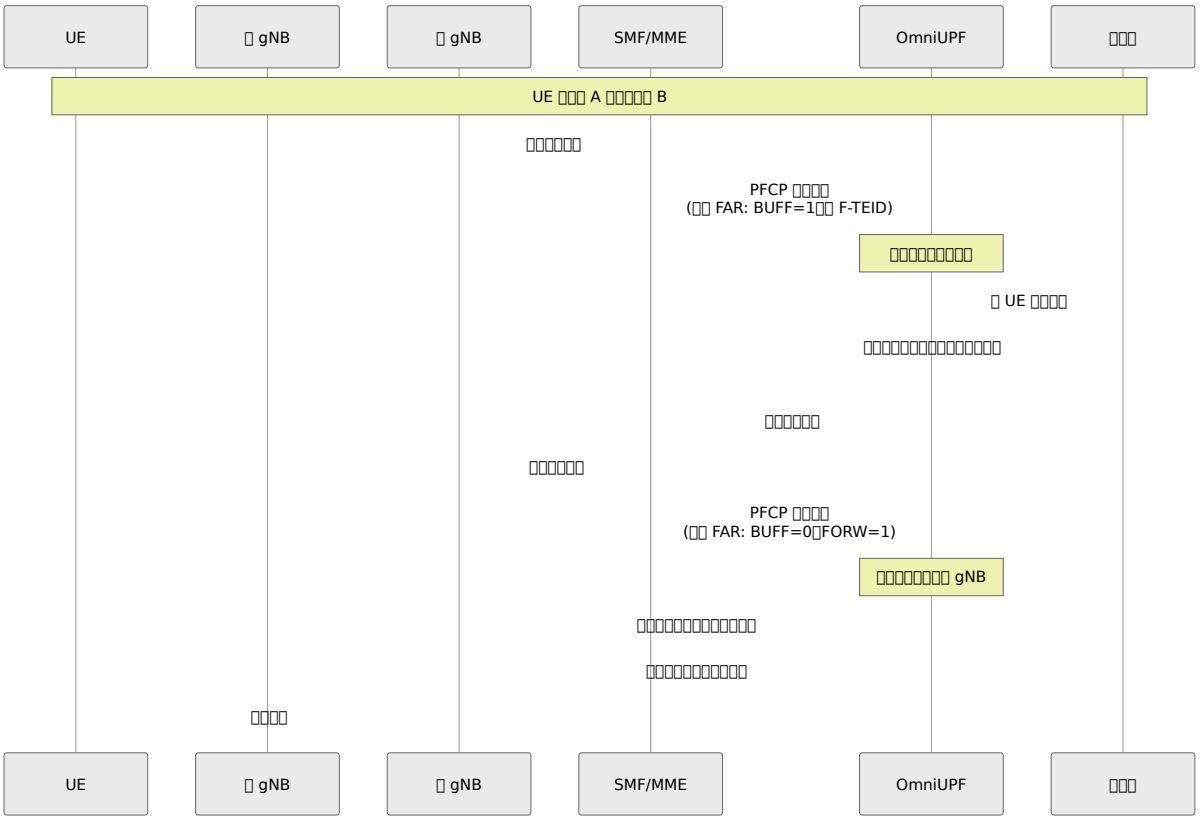
有缓冲：OmniUPF 暂时保存数据包，直到新路径建立，然后无缝转发。用户体验到 零中断。

缓冲发生的时间

OmniUPF 在以下关键场景中缓冲数据包：

1. 基于 **N2** 的切换（**5G**）/基于 **X2** 的切换（**4G**）

当 UE 在基站之间移动时：



□□□□

- **T+0ms**□□□□□□□□□
- **T+10ms**□SMF □□ UPF □□□□□□□□□□□□□□□□□□□□□□
- **T+10-50ms**□□□□□□□□ - □□□□□□□□□□□□
- **T+50ms**□□□□□□□□□□□□SMF □□ UPF □□
- **T+50ms+**□UPF □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□~40ms □□□□□□□□□□□□□□□□□□□□ □□□□ □□□□□□□□□□□□□□□□□□□□□□

---

2. □□□□□□QoS □□□□□□□□□□

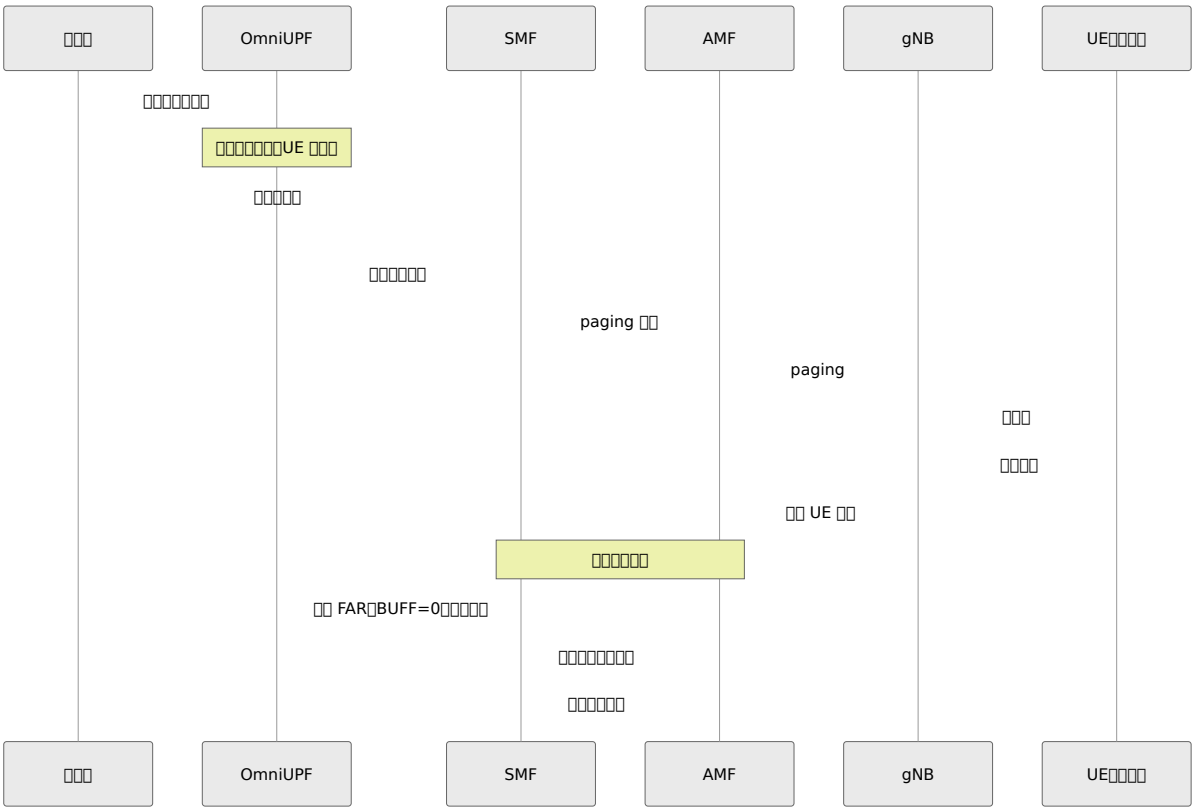
□□□□□□□□□□□□□□□□□□

- **QoS** □□/□□□□□□□□ 4G □□□□ 5G □□□□NSA □□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ UPF□ULCL □□□□

□□□□□□□□□□□□□□□□□□□

3. 下行数据通知（空闲模式恢复）

当 UE 处于空闲模式（屏幕关闭、节省电池）并且下行数据到达时：



没有缓冲：触发通知的初始数据包将被 丢弃，要求发送方重新传输（增加延迟）。有缓冲：唤醒 UE 时立即交付触发 UE 的数据包。

---

4. 跨 RAT 切换（4G ↔ 5G）

当 UE 在 4G 和 5G 覆盖之间移动时：

- 架构变化（eNodeB ↔ gNB）
- 隧道端点变化（不同的 TEID 分配）
- 缓冲确保在 RAT 类型之间平滑过渡

OmniUPF 中的缓冲工作原理

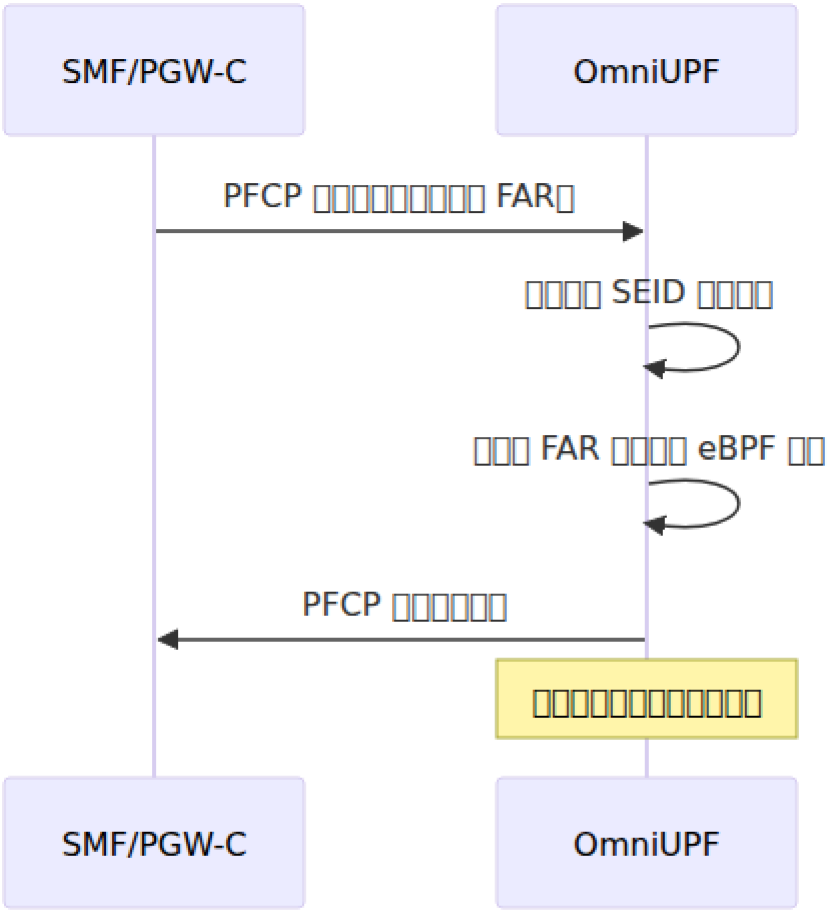
技术机制：

OmniUPF 使用 两阶段缓冲架构：

1. **eBPF** 阶段（内核）：根据 FAR 动作标志检测需要缓冲的数据包

2. 用户空间阶段：在内存中存储和管理缓冲的数据包

缓冲过程：



关键细节：

- 缓冲端口：UDP 端口 22152（数据包从 eBPF 发送到用户空间）
- 封装：数据包以 GTP-U 封装，FAR ID 作为 TEID
- 存储：每个 FAR 的内存中存储缓冲，带有元数据（时间戳、方向、数据包大小）
- 限制：
  - 每个 FAR 限制：10,000 个数据包（默认）
  - 全局限制：所有 FAR 的总计 100,000 个数据包
  - TTL：30 秒（默认） - 超过 TTL 的数据包将被丢弃
- 清理：后台进程每 60 秒删除过期数据包

缓冲生命周期：

1. 缓冲启用：SMF 通过 PFCP 会话修改设置 FAR 动作 BUFF=1（第 2 位）
2. 数据包缓冲：eBPF 检测 BUFF 标志，封装数据包，发送到端口 22152

3. 用户空间存储：缓冲管理器使用 FAR ID、时间戳、方向存储数据包
4. 缓冲禁用：SMF 通过新转发参数设置 FAR 动作 FORW=1, BUFF=0
5. 刷新缓冲：用户空间重放缓冲的数据包，使用新 FAR 规则（新隧道端点）
6. 恢复正常：新数据包立即通过新路径转发

这对用户体验的重要性

现实世界影响：

场景	没有缓冲	有缓冲
切换期间的视频通话	通话冻结 1-2 秒，可能掉线	无缝，未中断
基站边缘的□件下载	下载失败，必须重新开始	下载持续不中断
移动中的在线游戏	连接掉线，游戏被踢	平稳游戏，无断开
汽车中的 **VoIP** 通话	通话在每次切换时掉线	清晰无缝，无掉线
火车上的视频流	视频缓冲，质量下降	平稳播放
笔记本电脑的移动热点	SSH 会话掉线，视频通话失败	所有连接保持

网络运营商的好处：

- 降低通话掉线率（**CDR**）：关键 KPI
- 提高客户满意度：用户不会注意到切换
- 降低支持成本：减少关于掉线的投诉
- 竞争优势：营销“最佳覆盖网络”

缓冲管理操作

操作员可以通过 Web UI 和 API 监控和控制缓冲：

监控：

- 查看每个 **FAR ID** 的缓冲数据包（计数、字节、年龄）
- 跟踪缓冲使用情况与限制（每个 FAR、全局）
- 对缓冲溢出发出警报或缓冲持续时间过长
- 识别卡住的缓冲（数据包缓冲 > TTL 阈值）

控制操作：

- 刷新缓冲：手动触发缓冲重放（故障排除）
- 清除缓冲：丢弃缓冲的数据包（清理卡住的缓冲）
- 调整 **TTL**：更改数据包过期时间
- 修改限制：增加每个 FAR 或全局缓冲容量

故障排除：

- 缓冲未刷新：检查 SMF 是否发送了 FAR 更新以禁用缓冲
- 缓冲溢出：增加限制或调查为何缓冲持续时间过长
- 缓冲中的旧数据包：TTL 可能太高，或 FAR 更新延迟
- 过度缓冲：可能表明移动性问题或 SMF 问题

有关缓冲操作的详细信息，请参见 缓冲管理指南。

缓冲配置

在 `config.yml` 中配置缓冲行为：

```yaml
# 缓冲设置
buffer_port: 22152                # 缓冲数据包的 UDP 端口（默认）
buffer_max_packets: 10000         # 每个 FAR 的最大数据包（防止内存耗尽）
buffer_max_total: 100000          # 所有 FAR 的最大总数据包
buffer_packet_ttl: 30             # TTL（秒）（丢弃旧数据包）
buffer_cleanup_interval: 60       # 清理间隔（秒）
```

建议：

- 高移动性网络（高速公路、火车）：将 `buffer_max_packets` 增加到 20,000+
- 密集城市地区（频繁切换）：将 `buffer_packet_ttl` 减少到 15 秒
- 低延迟应用：将 `buffer_packet_ttl` 设置为 10 秒以防止过时数据

- □□□□

- 总数据包计数：汇总接口统计信息

有关监控详细信息，请参见 监控指南。

容量管理

eBPF 映射容量监控：

UPF 性能取决于 eBPF 映射容量。操作员可以：

- 实时监控映射使用情况，显示百分比指示器
- 查看每个 **eBPF** 映射的容量限制
- 颜色编码警报：
    - 绿色（<50%）：正常
    - 黄色（50-70%）：注意
    - 琥珀色（70-90%）：警告
    - 红色（>90%）：危急

需要监控的关键映射：

- `uplink_pdr_map`：上行流量分类
- `downlink_pdr_map`：下行 IPv4 流量分类
- `far_map`：转发规则
- `qer_map`：QoS 规则
- `urr_map`：使用跟踪

容量规划：

- 每个 PDR 消耗一个映射条目（键大小 + 值大小）
- 映射容量在 UPF 启动时配置（内核内存限制）
- 超过容量将导致会话建立失败

有关容量监控，请参见 容量管理。

配置管理

UPF 配置:

查看和验证 UPF 操作参数:

- **N3** 接口: 与 RAN 连接的 IP 地址（GTP-U）
- **N6** 接口: 与数据网络连接的 IP 地址
- **N9** 接口: 用于 UPF 之间通信的 IP 地址（可选）
- **PFCP** 接口: 与 SMF 连接的 IP 地址
- **API** 端口: REST API 监听端口
- 指标端点: Prometheus 指标端口

数据平面配置:

活动的 eBPF 数据路径参数:

- 活动 **N3** 地址: 运行时 N3 接口绑定
- 活动 **N9** 地址: 运行时 N9 接口绑定（如果启用）

有关配置查看的信息，请参见 配置视图。

故障排除

本节涵盖常见操作问题及其解决策略。

会话建立失败

症状: PFCP 会话无法创建，UE 无法建立数据连接

常见根本原因:

1. **PFCP** 关联未建立
    - 验证 SMF 是否可以访问 UPF PFCP 接口（端口 8805）
    - 检查会话视图中的 PFCP 关联状态
    - 验证节点 ID 配置在 SMF 和 UPF 之间是否匹配

2. **eBPF** □□□□□□□

   - □□□□□□□□□□□□□□□□>90%□□□□□□□□□
   - □□ UPF □□□□□ eBPF □□□□□
   - □□□□□□□□□□□□□□□□

3. □□□ **PDR/FAR** □□

   - □□ UE IP □□□□□□□□□□
   - □□ TEID □□□□□□□□□
   - □□ FAR □□□□□□□□□□

4. □□□□□□□

   - □□ N3 □□ IP □□□□□□ gNB □□
   - □□□□□□□□□ N6 □□□□□□□□□
   - □□ GTP-U □□□□□□□□□□

□□□□□□□□□□□□□□□□□ □□□□□□□□□

□□□□□□□□□□

□□□UE □□□□□□□□□□□□□□□□□□□□

□□□□□□□□

1. **PDR** □□□□

   - □□□□ PDR TEID □□□ gNB □□□ TEID □□
   - □□□□ PDR UE IP □□□□□□□ IP □□
   - □□ SDF □□□□□□□□□□

2. **FAR** □□□□

   - □□ FAR □□□□□□□□□□□□□□□□□□□□□
   - □□□□□□□□□□□□□□□
   - □□□□□□□□□

3. **QoS** 限制超出
    - 检查 QER MBR（最大比特率）设置
    - 验证 GBR（保证比特率）分配
    - 监控因速率限制导致的数据包丢失
4. 接口 **MTU** 问题
    - 验证 GTP-U 开销（40-50 字节）是否导致分片
    - 检查 N3/N6 接口的 MTU 配置
    - 监控 ICMP 分片所需消息

缓冲相关问题

症状：数据包无限期缓冲，缓冲溢出

常见根本原因：

1. 缓冲在切换后未禁用
    - 检查 FAR 缓冲标志（第 2 位）
    - 验证 SMF 是否发送会话修改以禁用缓冲
    - 如果卡住，手动通过控制面板禁用缓冲
2. 缓冲 **TTL** 到期
    - 检查缓冲视图中的数据包年龄
    - 验证缓冲 TTL 配置（默认可能过长）
    - 手动清除过期缓冲
3. 缓冲容量耗尽
    - 监控总缓冲使用情况和每个 FAR 的限制
    - 检查是否存在导致过度缓冲的错误配置规则
    - 调整 max_per_far 和 max_total 缓冲限制

有关缓冲故障排除的信息，请参见 缓冲操作。

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□

1. □□□□□
    - eBPF □□□□ 64 □□□□□□□□□□□
    - □□□□□□□□□□□□□□□
    - □□ URR □□□□□□□□
2. □□□□□□□□
    - □□ eBPF □□□□□□□□□
    - □□□□□□□□□□□□□□□ eBPF □□
    - □□ XDP □□□□□□□□□□□□
3. □□□□□□□□
    - □ N3/N6 □□□□□□□□□□□□□□□□□□□
    - □□□□□□□□ eBPF□□□□□□□□□
    - □□□□□□□□□□□ XDP □□

□□□□

□□□□□□□□□□□□□□CPU □□

□□□

1. □□ **XDP** □□□□□ XDP □□□□□
2. □□ **eBPF** □□□□□□□□□□□□□□□□□□□
3. □□ **CPU** □□□□eBPF □□□□□□□□
4. □□□□□□□□□ NIC □□□□ XDP □□

□□□□□□□□

- **XDP** □□□□□□□□ 10M+ □□□□□

- **PDR** 容量：数百万个 PDR 仅受限于内核内存
- 会话计数：每个 UPF 实例数千个并发会话
- 吞吐量：在适当的 NIC 卸载下实现多千兆吞吐量

有关性能调整的信息，请参见 架构指南。

附加□档

组件特定操作指南

有关每个 UPF 组件的详细操作和故障排除：

配置指南

完整的配置参考，包括：

- 配置参数（YAML、环境变量、CLI）
- 操作模式（UPF/PGW-U/SGW-U）
- XDP 附加模式概述
- 虚拟化兼容性（Proxmox、VMware、KVM、Hyper-V、VirtualBox）
- NIC 兼容性和 XDP 驱动程序支持
- 不同场景的配置示例
- 映射大小和容量规划

XDP 模式指南

详细的 XDP 配置和优化，包括：

- XDP 附加模式说明（通用/本地/卸载）
- 性能比较和基准测试
- Proxmox VE 本地 XDP 设置的逐步指南
- 多队列配置以实现最佳性能
- VMware ESXi、KVM 和 Hyper-V 的 XDP 设置