OmniUPF 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，提供运营商级的数据包转发、QoS 执行和移动网络的流量管理。基于 Linux eBPF（扩展伯克利数据包过滤器）技术构建，并增强了全面的管理能力，OmniUPF 提供了 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。它提供：

- 高速数据包转发 在移动设备和数据网络之间
- 服务质量（**QoS**）执行不同流量类型
- 基于数据包过滤器和规则的流量检测和路由
- 用量报告 用于计费和分析
- 数据包缓冲 用于移动性和会话管理场景
- 合法拦截 支持以符合监管要求

OmniUPF 实现了 3GPP TS 23.501（5G）和 TS 23.401（LTE）中定义的完整 UPF 功能，提供了一个完整的、可投入生产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 □□□□

□□□□□□□

- □□□□ 3GPP □□□□□□□□□□□□□□
- □□ eBPF □□□□□□□□□□□□□□□
- GTP-U□GPRS □□□□□□□□□□□□
- □□□□□□□□□□ IPv4 □ IPv6 □□
- XDP□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□

QoS □□□□□□□

- □□□□□ QoS □□□□□QER□
- □□□□□□□□□□□□PDR□
- □□□□□□□□□□□□□FAR□
- □□□□□□□□□□□□□□SDF□□□
- □□□□□□□□□□□□□□□□URR□

□□□□□□□□□

- PFCP□□□□□□□□□□□□□□□ SMF/PGW-C
- □□□□□□□□ RESTful API
- □□□□□□□□
- eBPF □□□□□□
- □□ Web □□□□□

□□□□□

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

□□□□□□□□□□□□□□□□ Web UI □□□

□□□□□□□□

OmniUPF □□□□□□□□□□□□□□□□□ 5G □□□SA□□5G NSA □ 4G LTE/EPC □□□□□□□□□□□□□□□□□□□□**OmniUPF** □□□□□□□□□□□□□□□□□

- **UPF**□□□□□□□□□ - 5G/NSA □□□□□□□□ N4/PFCP □ OmniSMF □□□
- **PGW-U□PDN** □□□□□□□□ - 4G EPC □□□□□□□□□□□ Sxc/PFCP □ OmniPGW-C □□□
- **SGW-U**□□□□□□□□□□□ - 4G EPC □□□□□□□□ Sxb/PFCP □ OmniSGW-C □□□

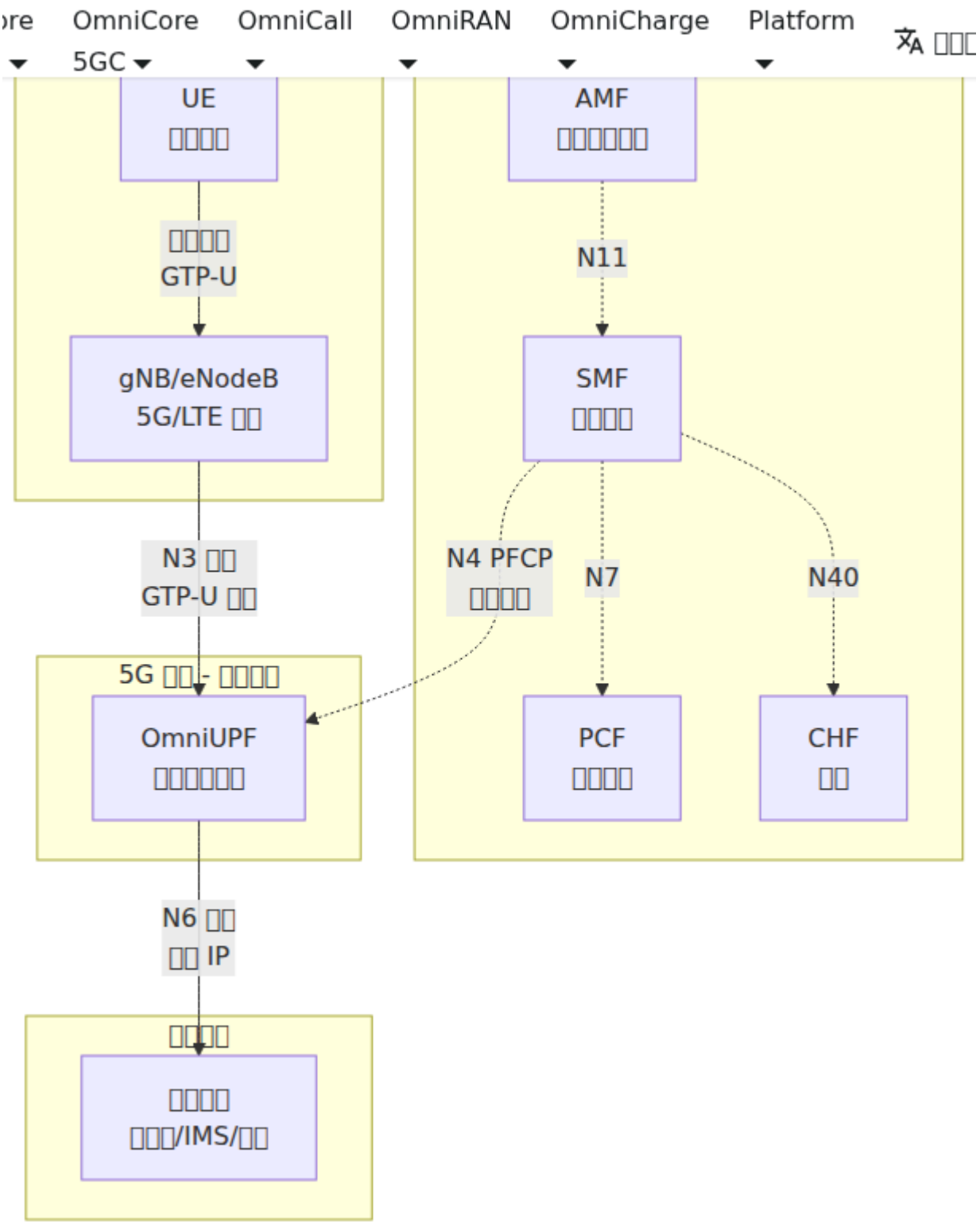
OmniUPF □□□ □□□□ □□□□□□□□□□

- □ **UPF**□□ 5G □□
- **PGW-U + SGW-U**□□□ 4G □□□□□□ EPC □□□
- **UPF + PGW-U + SGW-U**□□□□□ 4G □ 5G□□□□□□

□□□□□□□□□□□□ eBPF □□□□□□□□□ PFCP □□□□□□□□ UPF□PGW-U□SGW-U □□□□□□□□□□□□□□□□

5G □□□□□SA □□□

OmniUPF □□□□□□ 5G □□□□□□□□□□□□□□□□□□□□

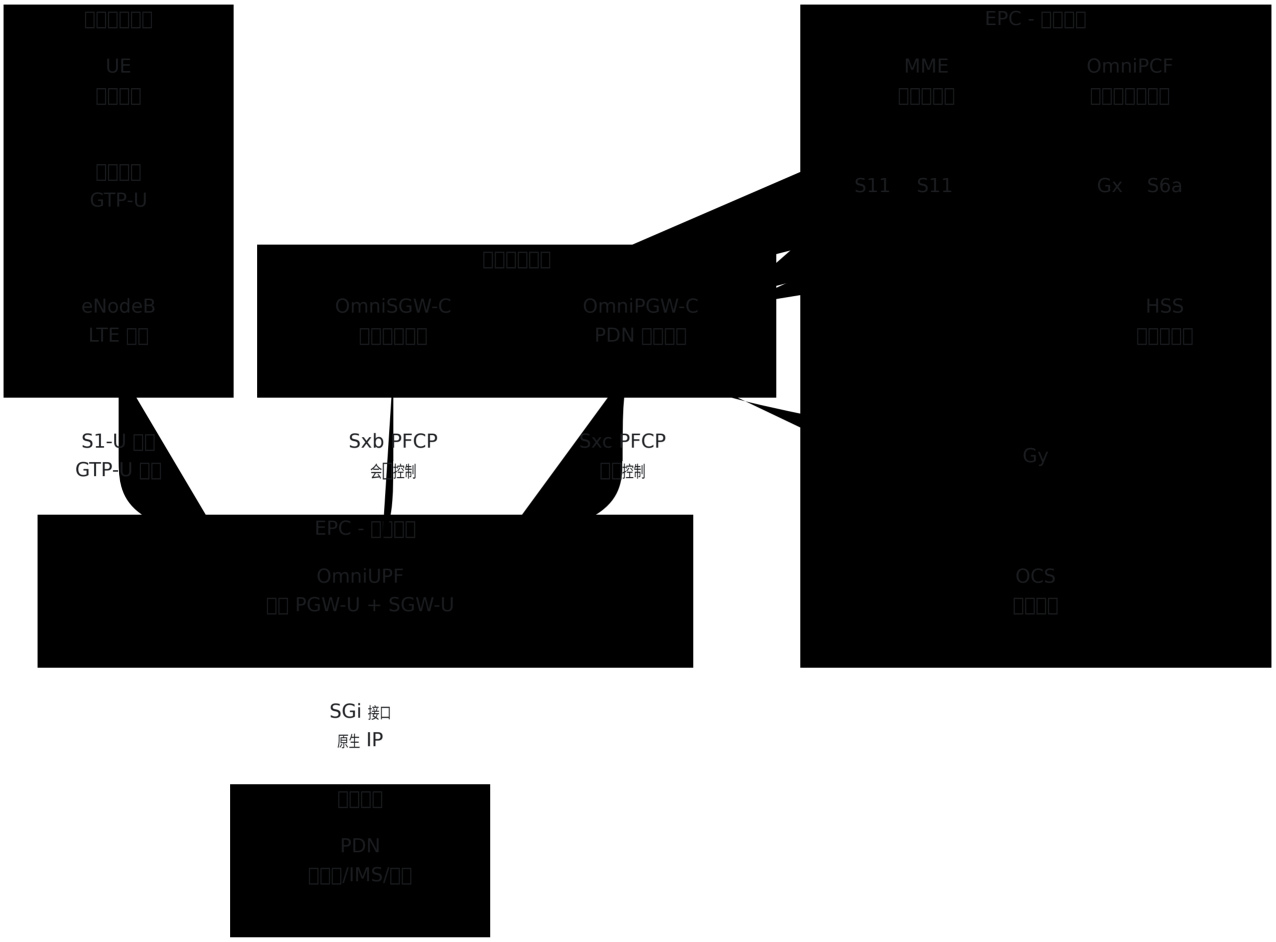


4G LTE/EPC

OmniUPF 4G LTE EPC OmniPGW-U OmniSGW-U

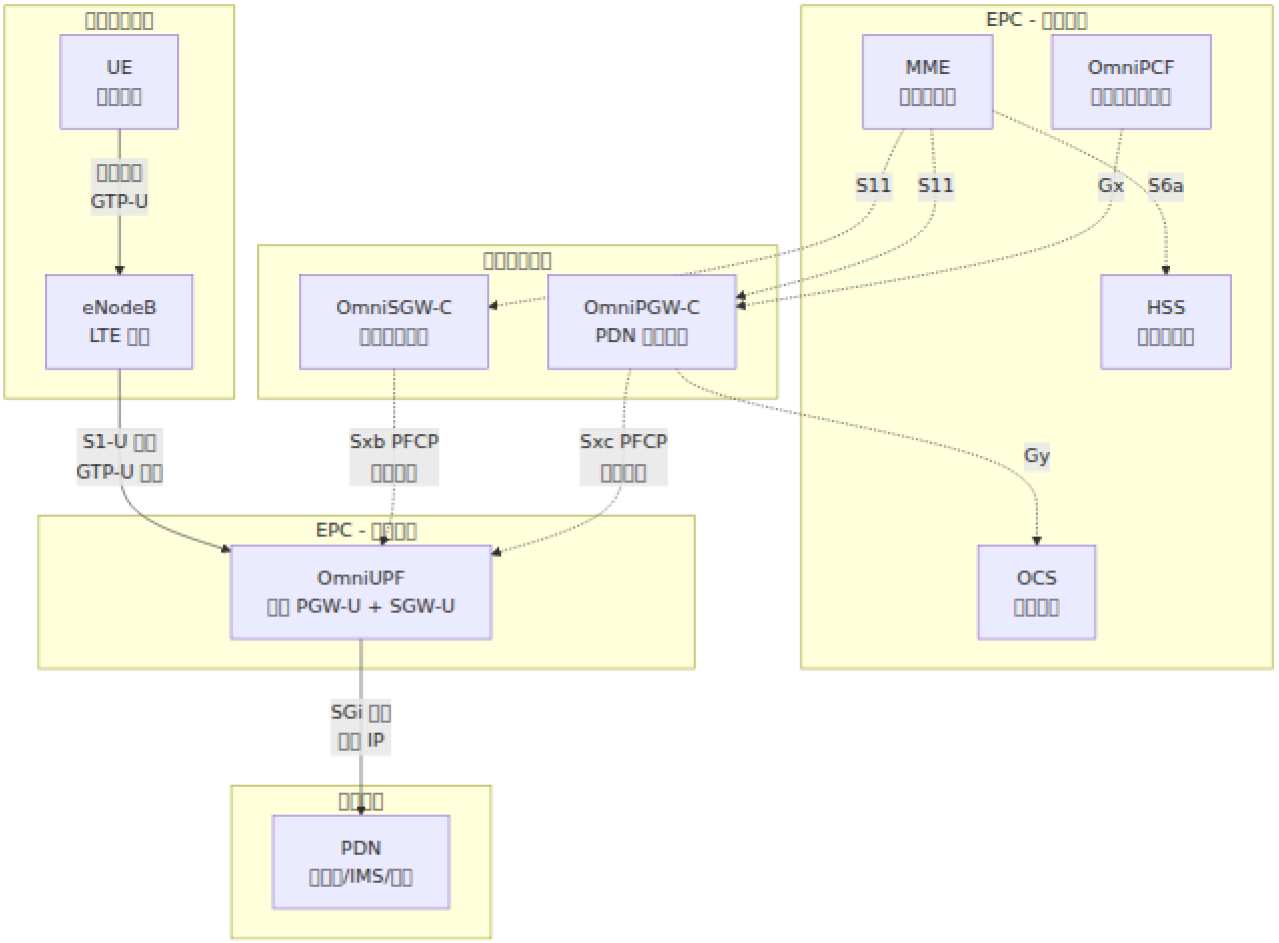
组合 **PGW-U/SGW-U** 模式（典型的 **4G** 部署）

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，一个作为 PGW-U。



用户平面功能在网络中的工作原理

用户平面功能（OmniUPF、OmniPGW-U 或 OmniSGW-U）作为转发平面，由相应的控制平面控制：

1. 会话建立
   - **5G**: OmniSMF 通过 N4 接口与 OmniUPF 建立 PFCP 关联
   - **4G**: OmniPGW-C 或 OmniSGW-C 通过 Sxb/Sxc 与 OmniPGW-U/OmniSGW-U 建立 PFCP 关联
   - 控制平面为每个 UE PDU 会话（5G）或 PDP 上下文（4G）创建 PFCP 会话
   - 用户平面通过 PFCP 接收 PDR、FAR、QER 和 URR 规则
   - eBPF 映射填充转发规则
2. 上行数据包处理（UE → 数据网络）
   - **5G**: 数据包通过 N3 接口从 gNB 到达，带有 GTP-U 封装
   - **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

- □□□□□□□ TEID □□□□□□□□ PDR □□
- eBPF □□□□ QER□□□□□□□□□□
- FAR □□□□□□□□□□□□□□□□□□□□□□
- □□ GTP-U □□□□□□□□□□ N6□5G□□ SGi□4G□□□
- URR □□□□□□□□□□□□□□□□□□

3. □□□□□□□□□□□□□ → UE□

- **5G**□□□□□□□ N6 □□□□□□□ IP □□
- **4G**□□□□□□□ SGi □□□□□□□ IP □□
- □□□□□□□ UE IP □□□□□□□□□□ PDR □□
- SDF □□□□□□□□□□□□□□□□□□□□□□□□
- FAR □□ GTP-U □□□□□□□□
- □□□□ TEID □ GTP-U □□
- **5G**□□□□□□□□ N3 □□□□ gNB
- **4G**□□□□□□□□ S1-U□SGW-U□□ S5/S8□PGW-U□□□ eNodeB

4. □□□□□□

- **5G**□OmniSMF □□□□□□□□□ PDR/FAR □□
- **4G**□OmniSGW-C/OmniPGW-C □ eNodeB □□□□□ TAU□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□

□□□□□□□□□□4G □ 5G□

OmniUPF □□□□□ 3GPP □□□ 5G □ 4G □□□□□□□□□□

5G □□

□□	□ → □	□□	**3GPP □□**
N4	OmniSMF ↔ OmniUPF	PFCP □□□□□□□□□□	TS 29.244
N3	gNB → OmniUPF	□□ RAN □□□□□□□□GTP-U□	TS 29.281
N6	OmniUPF → □□□□	□ DN □□□□□□□□□□ IP□	TS 23.501
N9	OmniUPF ↔ OmniUPF	□□/□□□□ UPF □□	TS 23.501

4G/EPC □□

□□	□ → □	□□	**3GPP □□**
Sxb	OmniSGW-C ↔ OmniUPF□SGW-U □□□	□□□□□ PFCP □□□□	TS 29.244
Sxc	OmniPGW-C ↔ OmniUPF□PGW-U □□□	PDN □□□ PFCP □□□□	TS 29.244
S1-U	eNodeB → OmniUPF□SGW-U □□	□□ RAN □□□□□□□□□GTP-U□	TS 29.281
S5/S8	OmniUPF□SGW-U□↔ OmniUPF□PGW-U□	□□□□□□□□□GTP-U□	TS 29.281
SGi	OmniUPF□PGW-U □□□→ PDN	□□□□□□□□□□□□□□□□□□□□□IP□	TS 23.401

□□□□□ PFCP □□N4□Sxb□Sxc□□□ TS 29.244 □□□□□□□ PFCP □□□□□□□□□□□□□□□□□□□□

UPF 组件

eBPF 数据路径

eBPF 数据路径是运行在 Linux 内核中的核心数据包处理引擎，以实现最大性能。

核心功能：

- **GTP-U** 处理：GTP-U 隧道的封装和解封装
- 数据包分类：使用 TEID、UE IP 或 SDF 过滤器将数据包与 PDR 规则匹配
- **QoS** 执行：根据 QER 规则应用速率限制和数据包标记
- 转发决策：执行 FAR 动作（转发、丢弃、缓冲、重复、通知）
- 用量跟踪：为基于流量的计费递增 URR 计数器

eBPF 映射：数据路径使用 eBPF 映射（内核内存中的哈希表）进行规则存储：

映射名称	目的	键	值
`uplink_pdr_map`	上行 PDR	TEID（32 位）	PDR 信息（FAR ID、QER ID、URR IDs）
`downlink_pdr_map`	下行 PDR（IPv4）	UE IP 地址	PDR 信息
`downlink_pdr_map_ip6`	下行 PDR（IPv6）	UE IPv6 地址	PDR 信息
`far_map`	转发规则	FAR ID	转发参数（动作、隧道信息）
`qer_map`	QoS 规则	QER ID	QoS 参数（MBR、GBR、标记）
`urr_map`	用量跟踪	URR ID	流量计数器（上行、下行、总计）
`sdf_filter_map`	SDF 过滤器	PDR ID	应用过滤器（端口、协议）

性能特征：

- □□□□□□□□□□□□□□□□□□□□
- **XDP** □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□ CPU □□□□□□□□□ CPU □□
- □□□eBPF □□□□□□□□□□□□ PDR/FAR□□□□□□□□□□

□□□□□□□□□□□ □□□□□□

PFCP □□□□□□

PFCP □□□□□ 3GPP TS 29.244□□□□□ SMF □ PGW-C □□□

□□□□□□

- □□□□□□PFCP □□□□□□□□□/□□
- □□□□□□□□□□□□□□□□□□□ PFCP □□
- □□□□□□□ PFCP □□□□□□□□ eBPF □□□□□
- □□□□□□□□ SMF □□□□□□□□□□□□□□□

PFCP □□□□□□

□□□□□	□□	□□
□□□□□	SMF → UPF	□□ PFCP □□□□□
□□□□□	SMF → UPF	□□ PFCP □□
□□	□□	□□□□□□□
□□□□□	SMF → UPF	□□□□□ PDU □□□□□□ PDR/FAR/QER/URR
□□□□□	SMF → UPF	□□□□□□□QoS □□□□□□
□□□□□	SMF → UPF	□□□□□□□□□□□□□□□□
□□□□□	UPF → SMF	□□□□□□□□□□□□

支持的信息元素（**IE**）：

- 创建 PDR、FAR、QER、URR
- 更新 PDR、FAR、QER、URR
- 删除 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、外部头部创建）
- QoS 参数（MBR、GBR、QFI）
- 用量报告触发器（流量阈值、时间阈值）

REST API 服务器

REST API 提供对 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 统计
- 缓冲管理：查看和控制数据包缓冲
- 映射信息：监控 eBPF 映射的使用和容量

API 端点：（共 34 个端点）

□□	□□	□□
□□	`/health`	□□□□□□□□
□□	`/config`	UPF □□
□□	`/pfcp_sessions`, `/pfcp_associations`	PFCP □□/□□□
PDRs	`/uplink_pdr_map`, `/downlink_pdr_map`, `/downlink_pdr_map_ip6`, `/uplink_pdr_map_ip6`	□□□□□□□
FARs	`/far_map`	□□□□□□
QERs	`/qer_map`	QoS □□□ ◆◆
URRs	`/urr_map`	□□□□□□
□□	`/buffer`	□□□□□□□□□□
□□	`/packet_stats`, `/route_stats`, `/xdp_stats`, `/n3n6_stats`	□□□□
□□	`/map_info`	eBPF □□□ □□□□
□□□□	`/dataplane_config`	N3/N9 □□ □□

□□ API □□□□□□□□□□□□ □□□□□

Web □□□□

Web □□□□□□□ UPF □□□□□□□□□□□□□

功能：

- 会话视图：浏览活动的 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

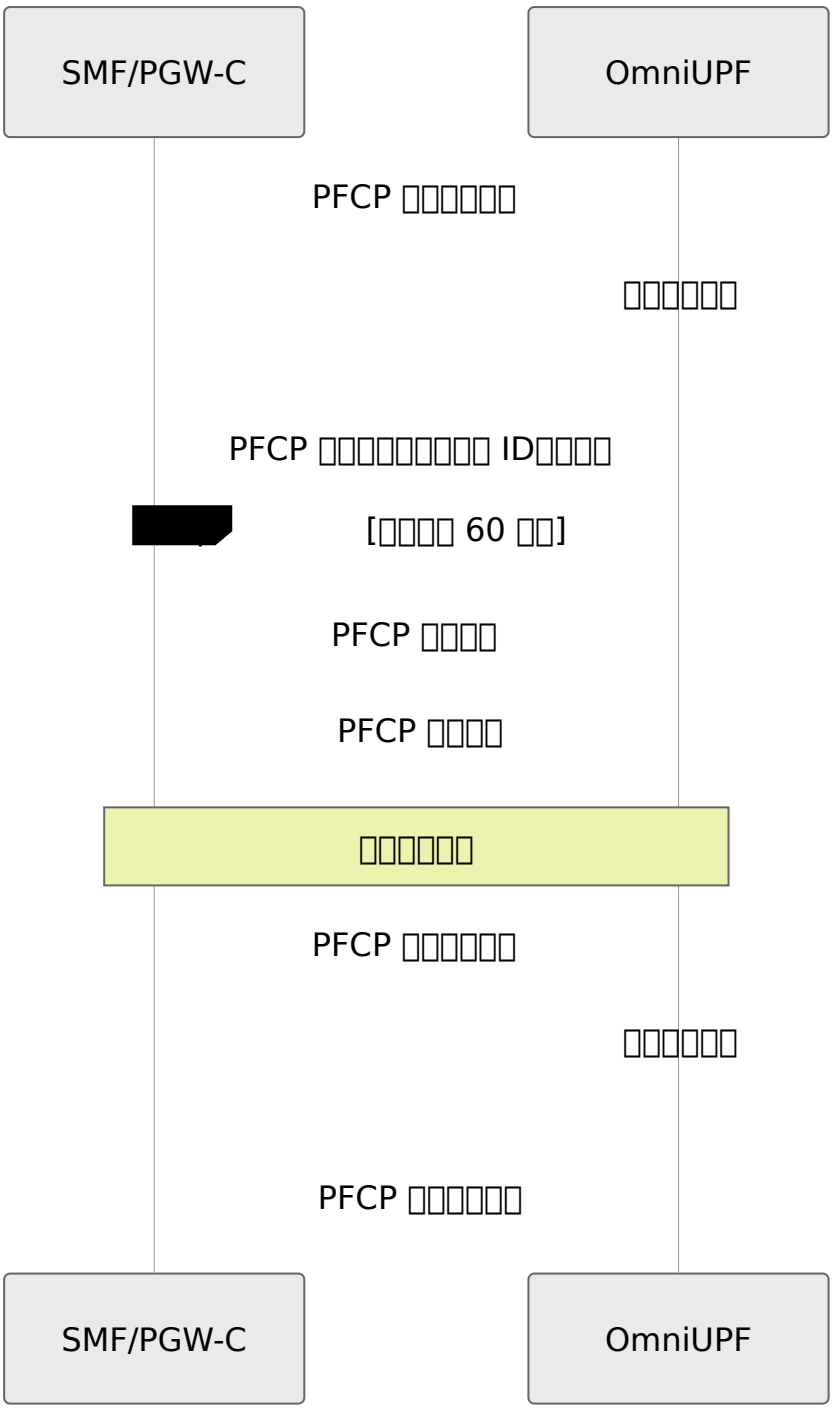
有关详细的 UI 操作，请参见 **Web UI** 作指南。

PFCP 协议和 SMF 集成

PFCP 关联

在创建会话之前，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

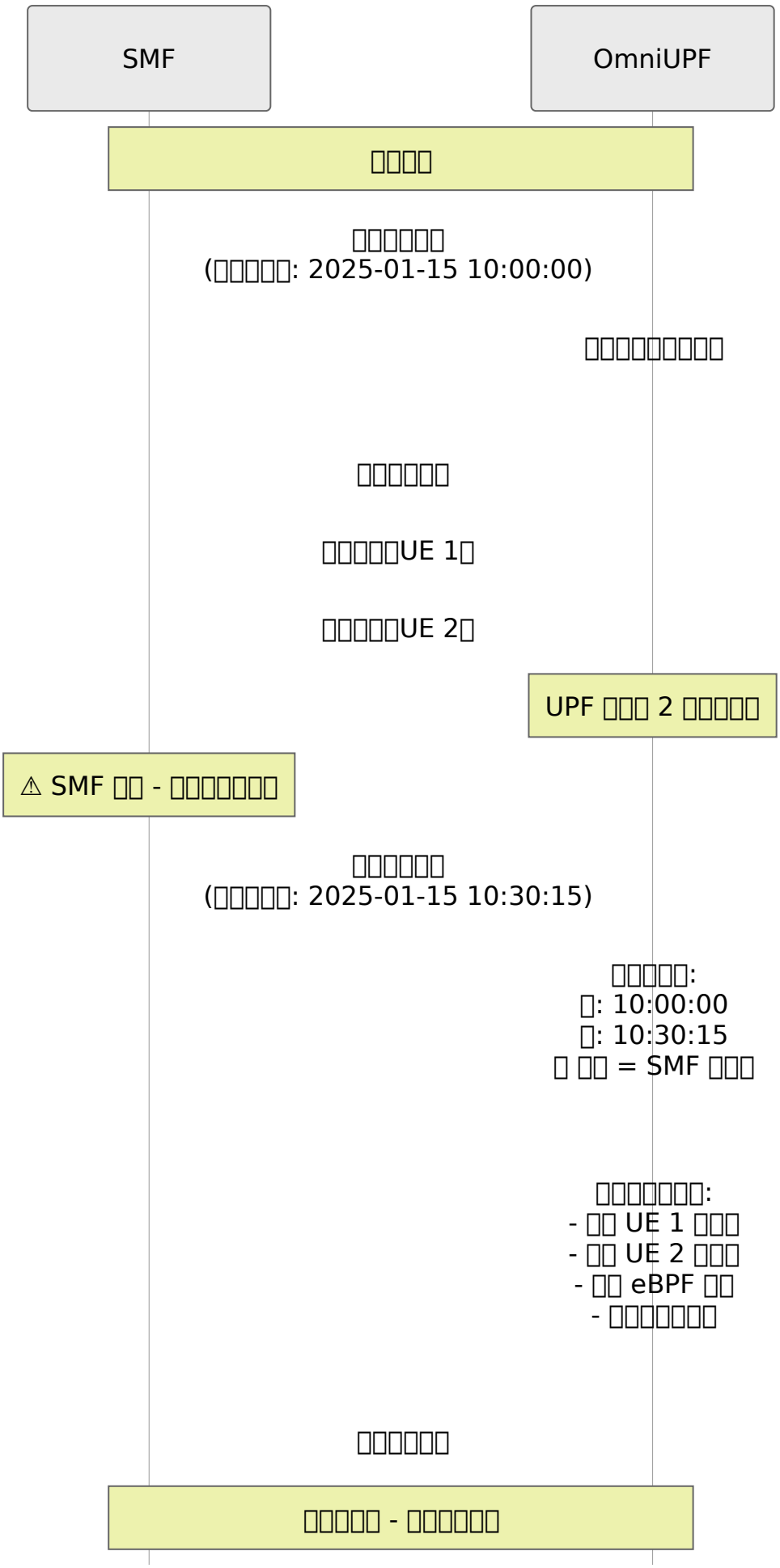
OmniUPF 自动检测 SMF 重启，并根据 3GPP TS 29.244 规范清理孤立会话。

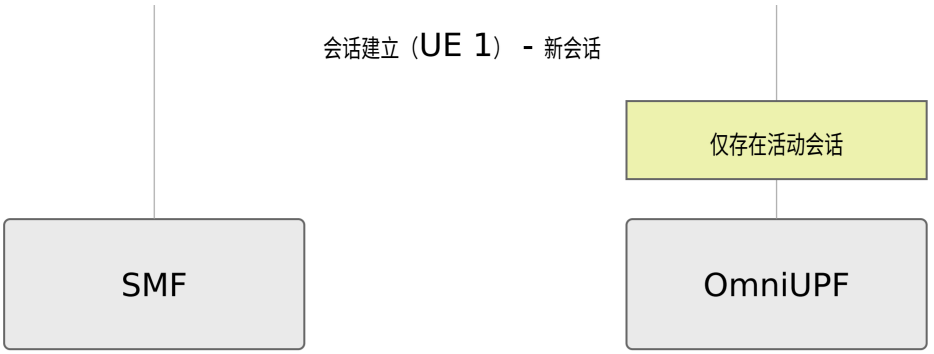
工作原理：

当 SMF 建立 PFCP 关联时，它提供一个 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 SMF 实例
6. SMF 为活动用户创建新的会话

重启检测流程：


SMF
OmniUPF
初始关联
关联设置请求
(恢复时间戳: 2025-01-15 10:00:00)
存储此关联的时间戳
关联设置响应
会话建立（UE 1）
会话建立（UE 2）
UPF 现在有 2 个活动会话
⚠ SMF 重启 - 丢失所有状态!
关联设置请求
(恢复时间戳: 2025-01-15 10:30:15)
比较时间戳:
旧: 10:00:00
新: 10:30:15
□ □□ = SMF 重启!
删除所有旧会话:
- 删除 UE 1 的会话
- 删除 UE 2 的会话
- 清除 eBPF 映射
- 重置会话计数器
关联设置响应
干净的状态 - 没有孤立会话



日志示例：

当 SMF 重启时，您会看到：

```
WARN: □ 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: 由于 SMF 重启，删除孤立会话 2（LocalSEID）
INFO: 由于 SMF 重启，删除孤立会话 3（LocalSEID）
...
INFO: 由于 SMF 重启，删除孤立会话 246（LocalSEID）
```

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：如果恢复时间戳是相同的，会话将被保留（SMF重新连接而不重启）。
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了变化，则UP功能应认为CP功能已重新启动，并应删除与该CP功能相关的所有PFCP会话。”

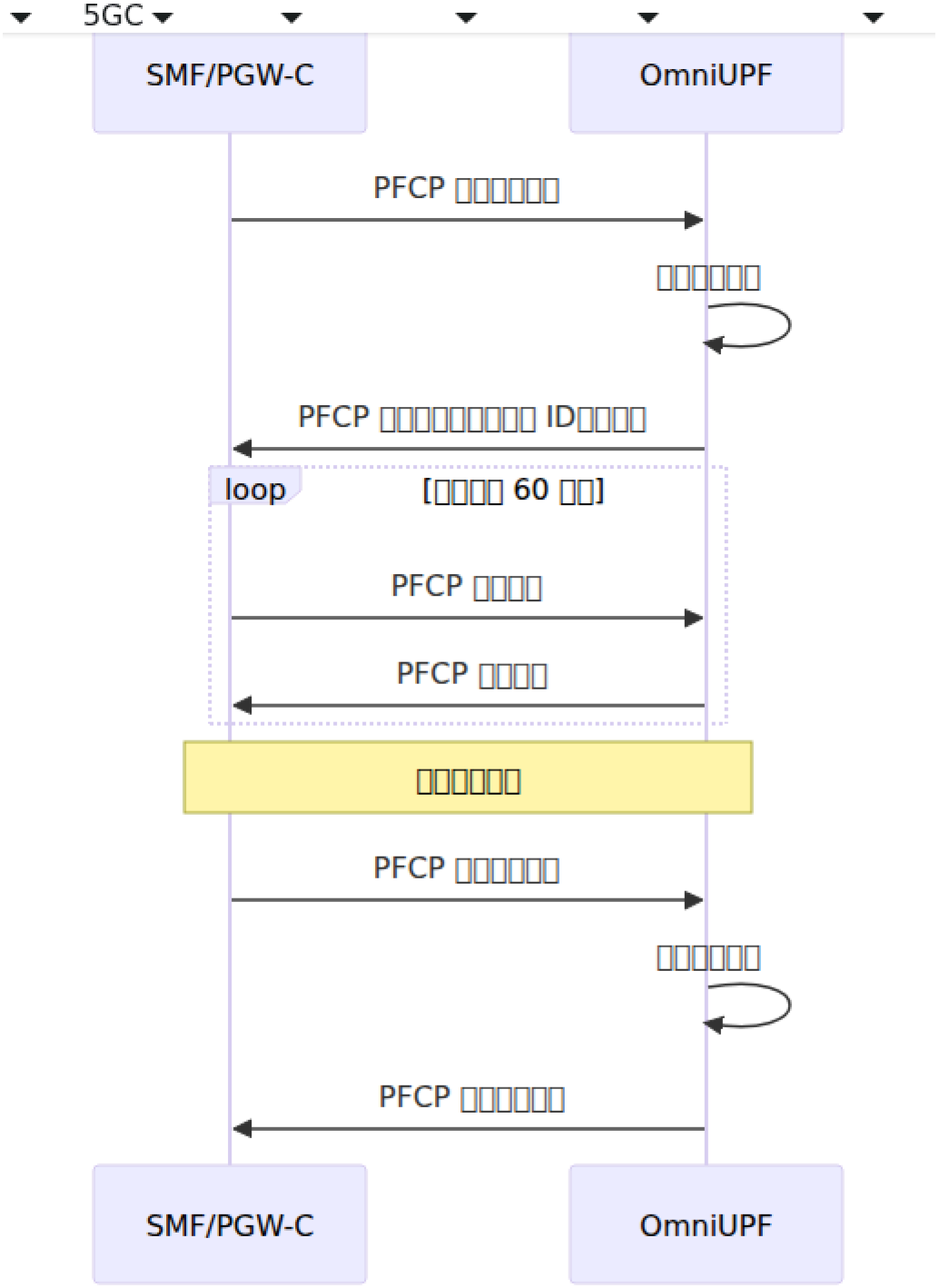
有关孤立会话的故障排除，请参见故障排除指南。

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：

□OmniUPF□GTP-U□□□□□□□□□□□□□□□□□□□□SGW-U□□□□PGW-U□□□□□□□□□□□□TEID□□□□□□□□□□□□□□□□□□□



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**：**S5/S8 GTP**架构中的**PGW-U**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**：**N9**架构中的对等**UPF**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：

接收错误指示时：

```
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: 因来自192.168.50.10的TEID 0x12345678的GTP-U错误指示删除了1个会话
```

Prometheus指标:

监控每个对等方和每个节点的错误指示活动:

```
# 从对等方接收到的总错误指示
upf_buffer_listener_error_indications_received_total{node_id="pgw-u-
1",peer_address="192.168.50.10"}

# 因错误指示而删除的会话
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（对于未知的入站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（如果没有关联，则为“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，全部会被删除

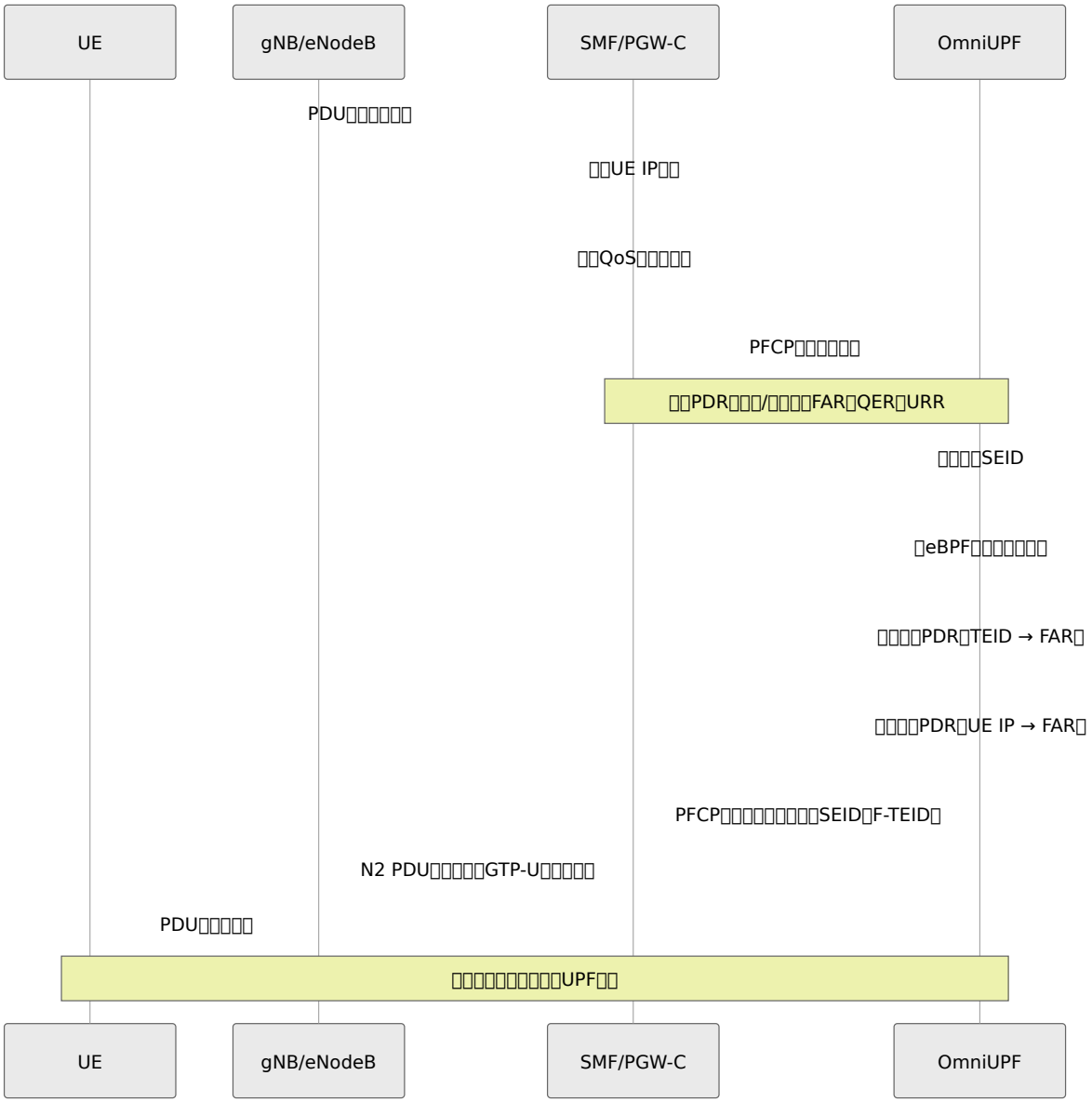
5. 与恢复时间戳互补：
    - 恢复时间戳检测 = 主动（在关联设置期间检测到重启）
    - 错误指示处理 = 被动（在流量流动时检测到死隧道）
6. 无效数据包处理：无效的错误指示会被记录并忽略（不会删除会话）

有关错误指示的故障排除，请参见故障排除指南。

PFCP会话创建

当UE建立PDU会话（5G）或PDP上下文（LTE）时，SMF会在UPF处创建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GTP-U封装的FAR转发到N3
- **FAR**：转发参数（外部头部创建、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）

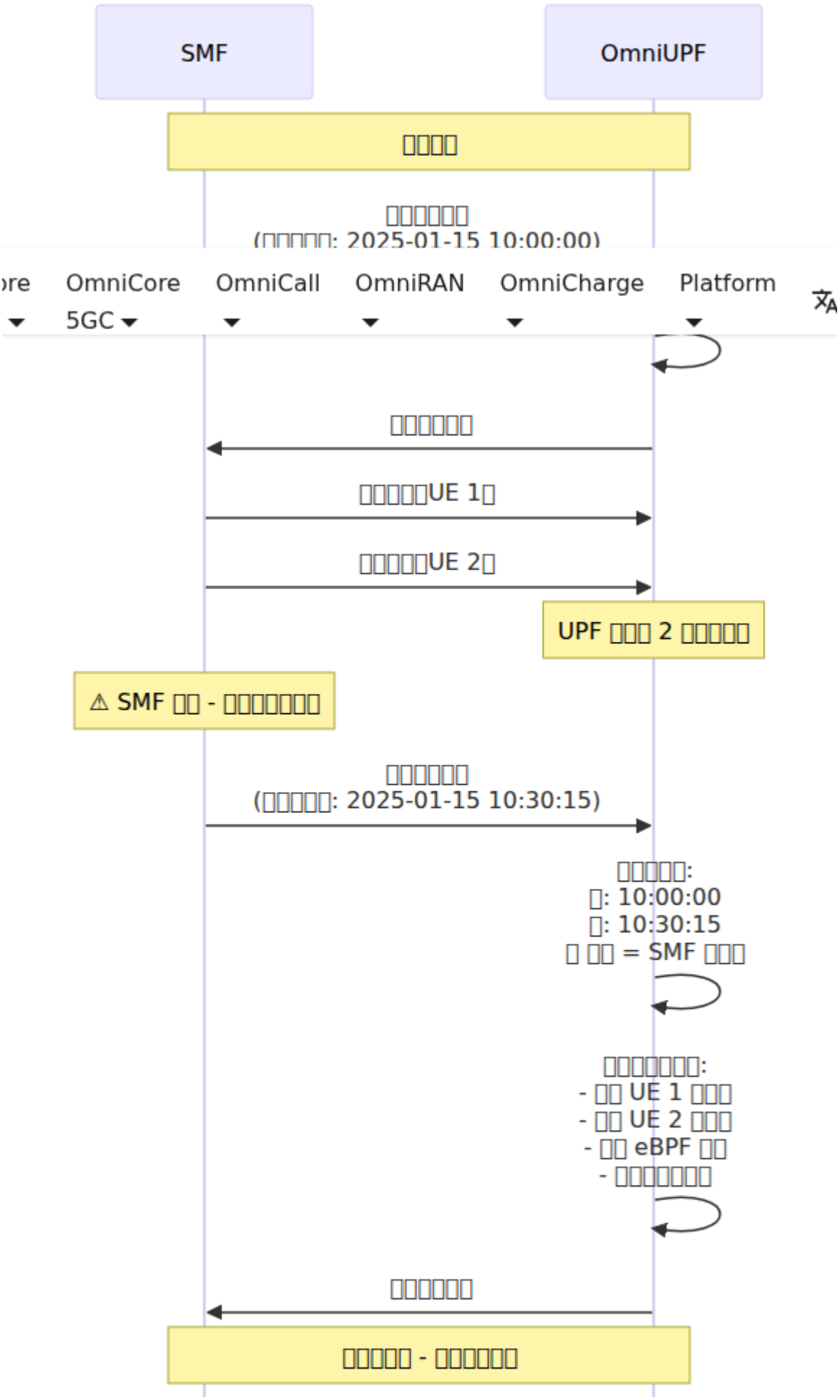
PFCP会话修改

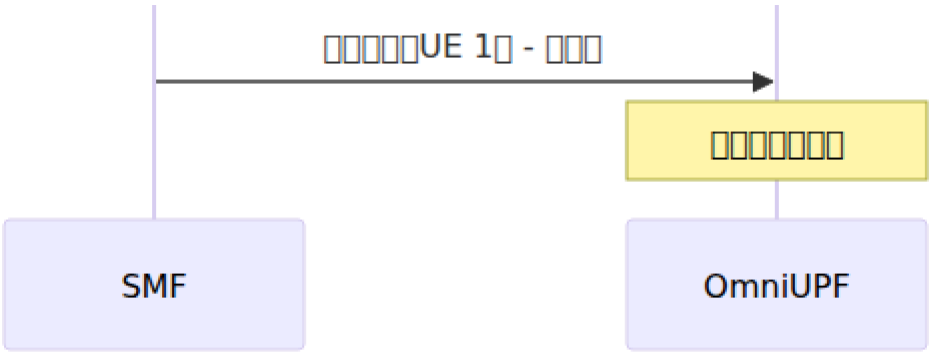
SMF可以修改会话以应对移动事件（切换）、QoS变化或服务更新。

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
    - 当准备好时将缓冲区刷新到新路径
2. **QoS**变化
    - 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
    - 为附加流量流添加新的PDR
    - 修改FAR以进行路由更改

会话修改流程：


SMF
OmniUPF
□□□□□
□□□□□□□
(□□□□□□: 2025-01-15 10:00:00)
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□□□□□□
□□□□□□UE 1□
□□□□□□UE 2□
UPF □□□ 2 □□□□□□
⚠ SMF □□ - □□□□□□□□
□□□□□□□
(□□□□□□: 2025-01-15 10:30:15)
□□□□□□:
□: 10:00:00
□: 10:30:15
□ □□ = SMF □□□
□□□□□□□□:
- □□ UE 1 □□□
- □□ UE 2 □□□
- □□ eBPF □□
- □□□□□□□□
□□□□□□□
□□□□□□ - □□□□□□□

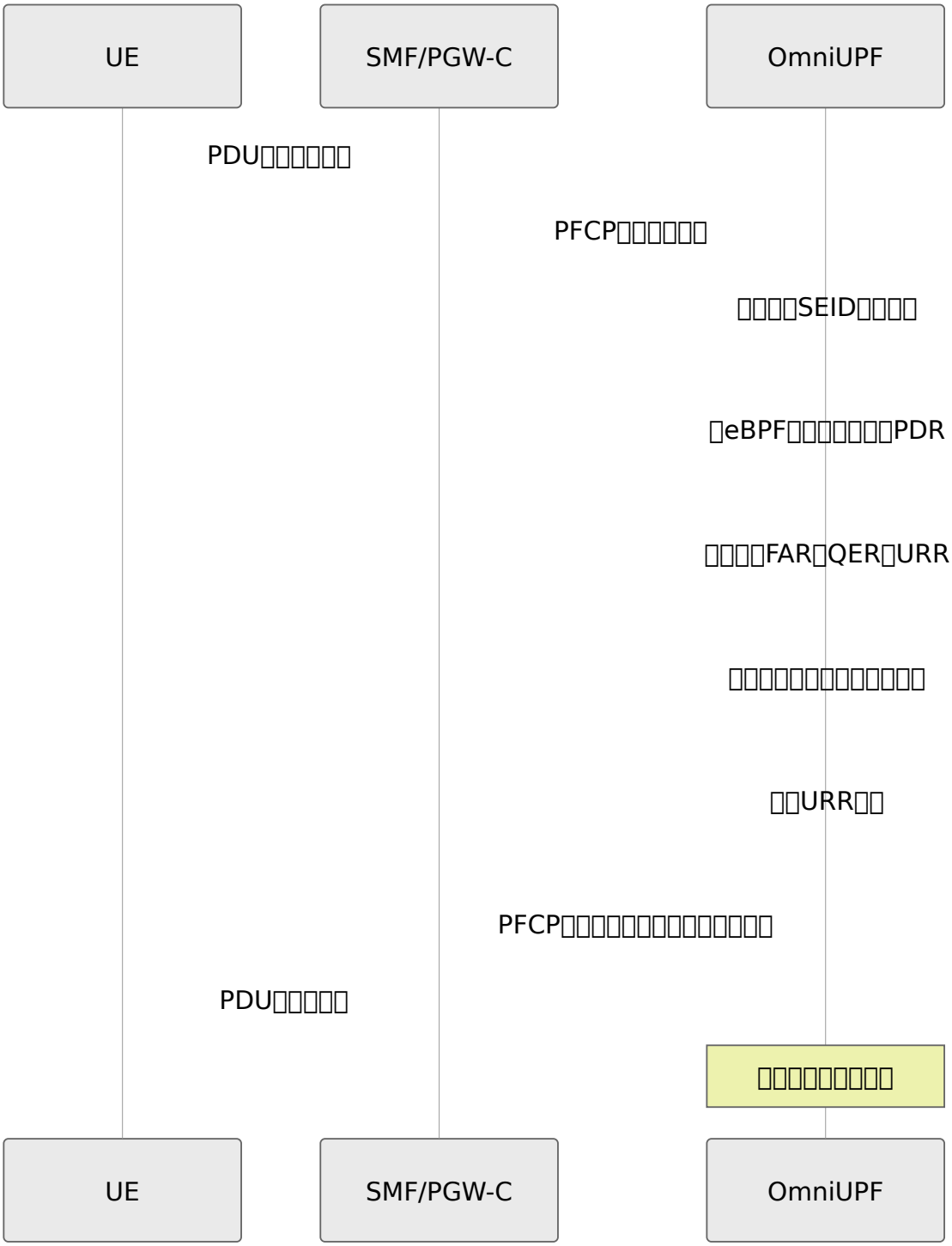


有关规则管理，请参见规则管理指南。

PFCP会话删除

当PDU会话被释放时，SMF会在UPF处删除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
- 清除数据包缓冲区
- 向SMF发送最终使用报告以进行计费

常见□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
- 用于QoS强制的QER（可选）
- 用于使用跟踪的URR（可选）

关键会话□作:

- 查看所有会话，包括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**，按N3���口的TEID键入
- 查看下行**PDR**，按UE IP地址（IPv4和IPv6）键入
- 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

转发□作规则（**FAR**）：

FAR定义对匹配数据包的处理。操作员可以：

- 查看**FAR**□作（转发、丢弃、缓冲、复制、通知）
- 检查转发参数（外部头部创建、目标）
- 监控每个**FAR**的缓冲状态
- 在故障排除期间切换特定**FAR**的缓冲

QoS强制规则（**QER**）：

QER应用带宽限制和数据包标记。操作员可以：

- 查看**QoS**参数（MBR、GBR、数据包延迟预算）
- 监控每个会话的活动**QER**
- 检查**5G QoS**流的**QFI**标记

使用报告规则（**URR**）：

URR跟踪计费的数据量。操作员可以：

- 查看流量计数器（上行、下行、总字节）
- 监控使用阈值和报告触发器
- 检查所有会话的活动**URR**

有关规则操作，请参见规则管理指南。

数据包缓冲

为什么缓冲对**UPF**至关重要

数据包缓冲是**UPF**最重要的功能之一，因为它防止在移动事件和会话重新配置期间的数据包丢失。如果没有缓冲，移动用户在每次从一个基站移动到另一个基站或网络条件变化时都���经历掉线、下载中断和实时通信失败。

问题：移动期间的数据包丢失

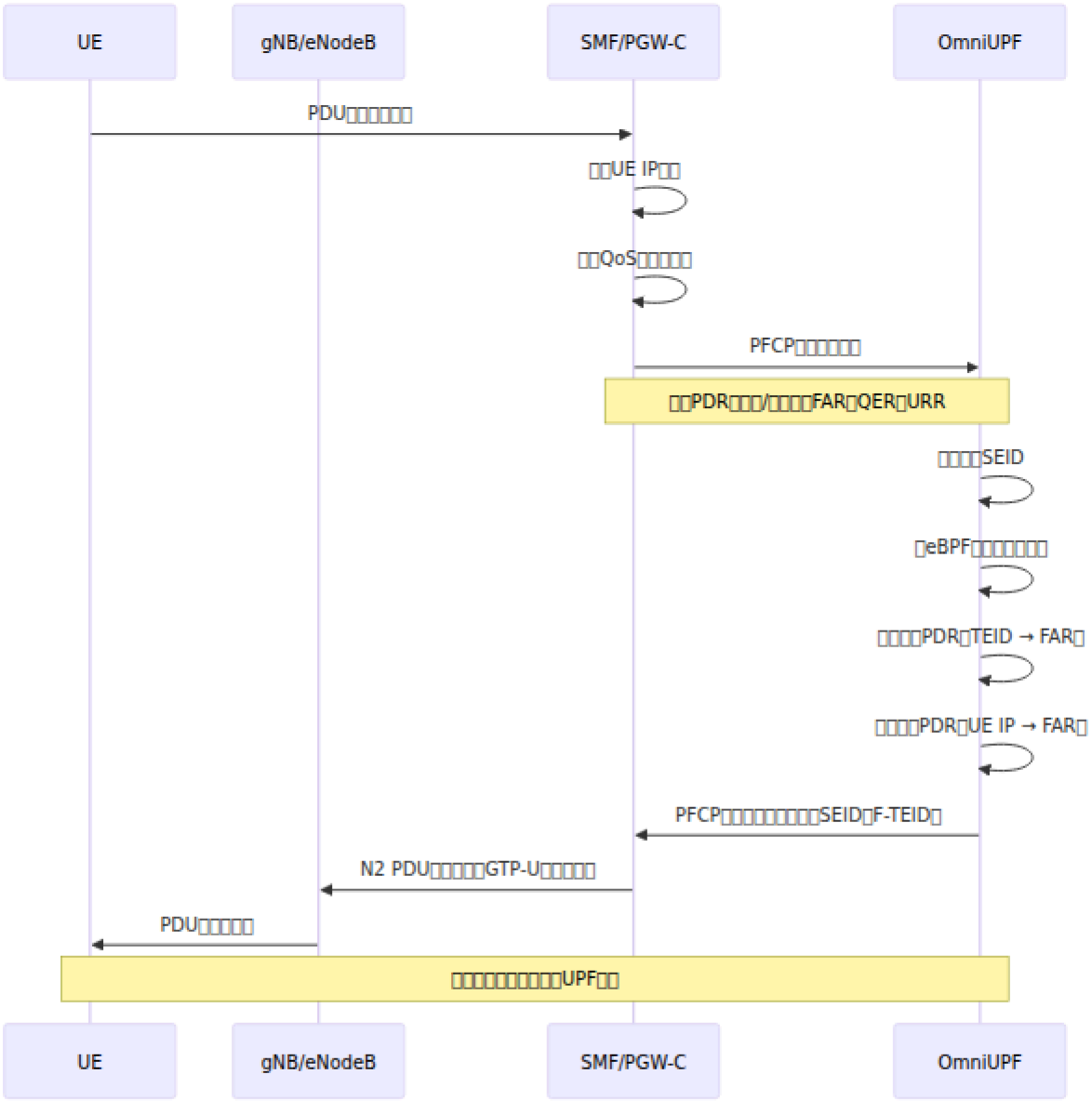
在移动网络中，用户不断移动。当设备从一个基站移动到另一个基站（切换）时，或者当网络需要重新配置数据路径时，会有一个关键窗口，在此期间数据包正在传输，但新路径尚未准备好：


OmniUPF
GTP-U
TEID: 67890
gNB
B


下行数据包
到达UE
OmniUPF
发送到哪里?
旧路径关闭
新路径未准备好


gNB
A
GTP-U
TEID: 12345
OmniUPF

□□□



时间线：

- **T+0ms**：旧路径仍然有效
- **T+10ms**：SMF告诉UPF缓冲（旧路���关闭，新路径未准备好）
- **T+10-50ms**：关键缓冲窗口 - 数据包到达但无法转发
- **T+50ms**：新路径准备就绪，SMF告诉UPF转发
- **T+50ms+**：UPF通过新路径刷新缓冲的数据包，然后立即转发新数据包

没有缓冲：~40ms的数据包（可能是数千个）将被丢失。有缓冲：零数据包丢失，无缝切换。

---

2. 会话修改（QoS变化、路径更新）

当网络需要更改会话参数时：

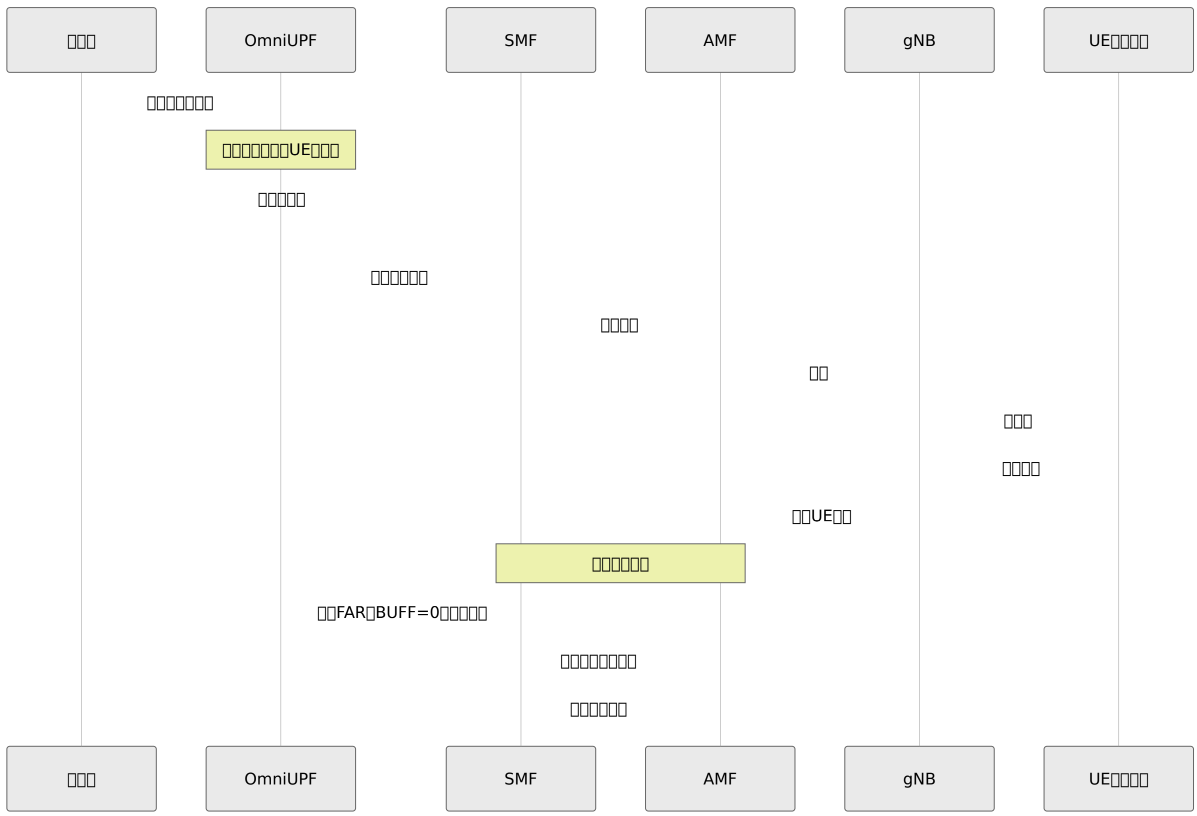
- **QoS**升级/降级：用户从4G移动到5G覆盖（NSA模式）
- 策略变化：企业用户进入公司校园（流量引导变化）
- 网络优化：核心网络将流量重新路由到更靠近的UPF（ULCL更新）

在修改过程中，控制平面可能需要原子地更新多个规则。缓冲确保数据包不会在部分/不一致的规则集下转发。

---

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，省电）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失，需要发送方重新传输（增加延迟）。有缓冲：唤醒UE时立即交付触发通知的数据包。

---

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类型之间平稳过渡

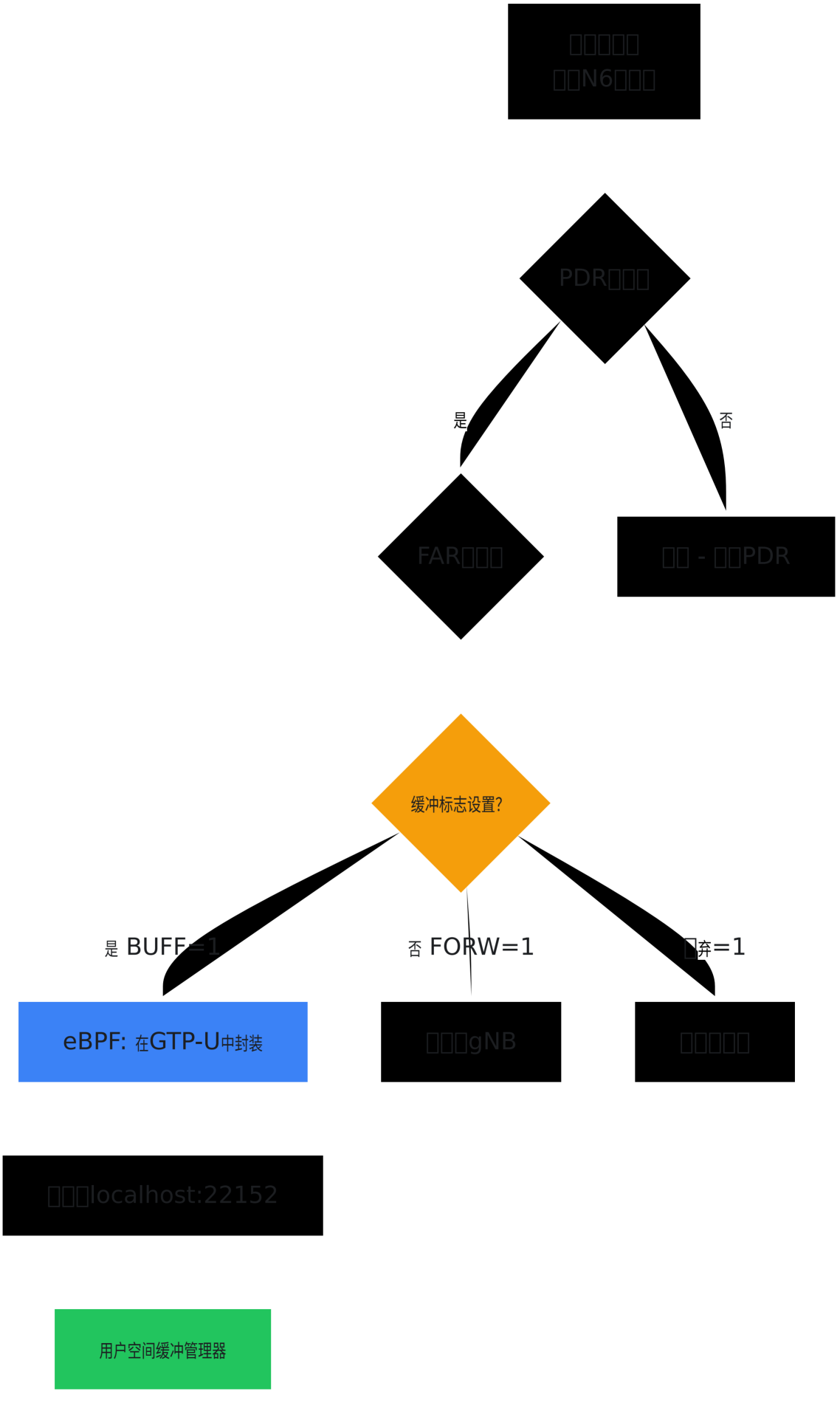
OmniUPF中的缓冲工作原理

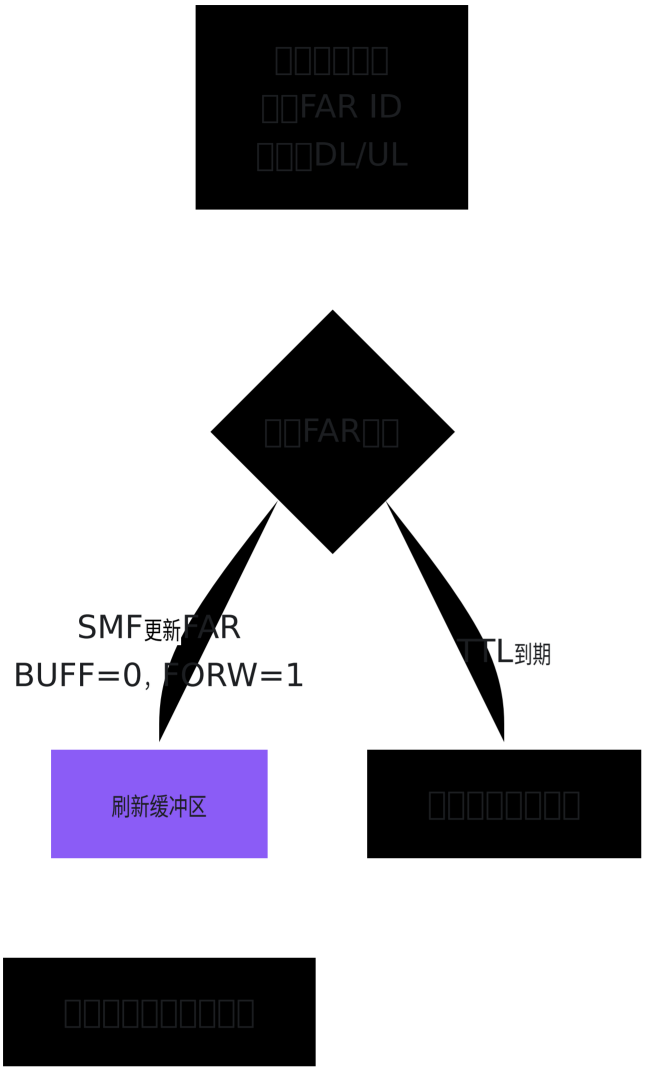
技术机制:

OmniUPF使用两阶段缓冲架构:

1. **eBPF**阶段（内核）：根据FAR操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:


□□□□□□
□□N6□□□
PDR□□□
是
否
FAR□□□
□□ - □□PDR
缓冲标志设置?
是 BUFF=1
否 FORW=1
□弃=1
eBPF: 在GTP-U中封装
□□□gNB
□□□□□
□□□localhost:22152
用户空间缓冲管理器



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：数据包用GTP-U封装，FAR ID作为TEID
- 存储：按FAR ID、时间戳、方向存储的内存缓冲区
- 限制：
  - 每FAR限制：10,000个数据包（默认）
  - 全局限制：所有FAR总共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秒删除过期的数据包

缓冲生命周期：

1. 启用缓冲：SMF通过PFCP会话修改设置FAR操作BUFF=1（第2位）
2. 缓冲数据包：eBPF检测到BUFF标志，封装数据包，发送到端口22152

3. 用户空间存储：缓冲管理器按FAR ID、时间戳、方向存储数据包
4. 禁用缓冲：SMF使用新转发参数设置FAR操作FORW=1，BUFF=0
5. 刷新缓冲区：用户空间通过新FAR规则（新隧道端点）重放缓冲的数据包
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，无中断
基站边缘的文件下载	下载失败，必须重新启动	下载继续，无中断
移动中的在线游戏	连接掉线，踢出游戏	平稳游戏，无断开
车载**VoIP**通话	通话在每次切换时掉线	清晰无掉线
火车上的视频流	视频缓冲，质量下降	流畅播放
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：

- 降低呼叫掉线率 **(CDR)**：网络质量的关键绩效指标
- 更高的客户满意度：用户不会注意到切换
- 降低支持成本：关于掉线连接的投诉更少
- 竞争优势：“覆盖最佳网络”的营销

缓冲管理□作

运营商可以通过 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）
- 缓冲溢出或过度缓冲持续时间的警报
- 识别卡住的缓冲区（缓冲的数据包 > TTL 阈值）

控制□作:

- 刷新缓冲区：手动触发缓冲重放（故障排除）
- 清除缓冲区：丢弃缓冲的数据包（清理卡住的缓冲区）
- 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���间过长
- 缓冲中的旧数据包：TTL 可能过高，或 FAR 更新延迟
- 过度缓冲：可能表明移动性问题或 SMF 问题

有关详细的缓冲操作，请参见 缓冲管理指南。

缓冲配置

在 `/etc/omniupf/runtime.exs` 中配置缓冲行为:

```
# 缓冲设置
buffer_port = 22152                    # 缓冲数据包的 UDP 端口（默认）
```

建议:

- 高移动性网络（高速公路、火车）：将 `buffer_max_packets` 增加到 20,000+

- □□□□□□□□□□□□□□□ `buffer_packet_ttl` □□□ 15 □
- □□□□□□□□ `buffer_packet_ttl` □□□ 10 □□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□ □□□□□□

□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□

- **RX** □□□□□□□□□□□□□□□□□
- **TX** □□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□
- **GTP-U** □□□□□□□□□□□□□

□□□□□□

□□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□
- □□  □□□□□□□□□□□□/□□
- □□□□□□□□□□□ TEID□□□□□ UE IP

XDP □□□

eXpress □□□□□□□□□□

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

N3/N6 □□□□□□

□□□□□□□□□□□□

- **N3 RX/TX**□□ RAN□gNB/eNodeB□□□□
- **N6 RX/TX**□□□□□□□□□□
- □□□□□□□□□□□□□□□

□□□□□□□□□□□□ □□□□□

□□□□

eBPF □□□□□□□□

UPF □□□□□□ eBPF □□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□ **eBPF** □□□□□□□□
- □□□□□□□□□
  - □□□<50%□□□□
  - □□□50-70%□□□□
  - □□□□70-90%□□□□
  - □□□>90%□□□□

□□□□□□□□□□□

- `uplink_pdr_map`□□□□□□□□
- `downlink_pdr_map`□□□ IPv4 □□□□
- `far_map`□□□□□
- `qer_map`□QoS □□
- `urr_map`□□□□□

□□□□□

- □□ PDR □□□□□□□□□□□□□ + □□□□
- □□□□□ UPF □□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□

□□□□□□□□□□ □□□□□

配置管理

UPF 配置：

查看和验证 UPF 操作参数：

- **N3** 接口：用于 RAN 连接的 IP 地址（GTP-U）
- **N6** 接口：用于数据网络连接的 IP 地址
- **N9** 接口：用于 UPF 之间通信的 IP 地址（可选）
- **PFCP** 接口：用于 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，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 SMF 和 UPF 之间的节点 ID 配置是否匹配

2. **eBPF** 映射容量耗尽

   - 检查容量视图中红色（>90%）的映射使用情况
   - 在 UPF 配置中增加 eBPF 映射大小
   - 如果映射已满，删除过期会话

3. 无效的 **PDR/FAR** 配置

   - 验证 UE IP 地址是否唯一且有效
   - 检查 TEID 分配是否冲突
   - 确保 FAR 引用有效的网络实例

4. 接口配置问题

   - 验证 N3 接口 IP 是否可以从 gNB 访问
   - 检查路由表以确保 N6 与数据网络的连接
   - 确认 GTP-U 流量未被防火墙阻止

有关详细故障排除，请参见 故障排除指南。

数据包丢失或转发问题

症状: UE 有连接但经历数据包丢失或没有流量

常见根本原因:

1. **PDR** 配置错误

   - 验证上行 PDR TEID 是否与 gNB 分配的 TEID 匹配
   - 检查下行 PDR UE IP 是否与分配的 IP 匹配
   - 检查 SDF 过滤器是否过于严格

2. **FAR** □作问题

   - 验证 FAR 操作是否为 FORWARD（而不是 DROP 或 BUFFER）
   - 检查 GTP-U 的外部头创建参数
   - 确保目标端点正确

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
    - 监控因速率限制而导致的数据包丢失
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 分片需要消息

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. 切换后未禁用缓冲
    - 检查 FAR 缓冲标志（位 2）
    - 验证 SMF 是否发送了会话修改以禁用缓冲
    - 如果卡住，通过控制面板手动禁用缓冲
2. 缓冲 **TTL** 过期
    - 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）
    - 手动清除过期的缓冲
3. 缓冲容量耗尽
    - 监控总缓冲使用情况和每个 FAR 限制
    - 检查是否存在配置错误的规则导致过度缓冲
    - 调整 max_per_far 和 max_total 缓冲限制

有关缓冲故障排除，请参见 缓冲操作。

统计异常

症状：意外的数据包计数，缺失的统计信息

常见根本原因：

1. 计数器溢出
    - eBPF 映射使用 64 位计数器（不应溢出）
    - 检查日志中的计数器重置事件
    - 验证 URR 报告是否正常工作
2. 路由统计未更新
    - 验证 eBPF 程序是否附加到接口
    - 检查内核版本是否支持所需的 eBPF 特性
    - 查看 XDP 统计以查找处理错误
3. 接口统计不匹配
    - 将 N3/N6 统计与内核接口计数进行比较
    - ���查是否有流量绕��� eBPF（例如，本地路由）
    - 验证所有流量是否通过 XDP 钩子流动

性能下降

症状：高延迟，低吞吐量，CPU 饱和

诊断：

1. 监控 **XDP** 统计：检查 XDP 丢弃或中止
2. 检查 **eBPF** 映射访问时间：哈希查找应在微秒级
3. 查看 **CPU** 利用率：eBPF 应在核心之间分配
4. 分析网络接口：验证 NIC 是否支持 XDP 卸载

可扩展性考虑：

- **XDP** 性能：每个核心每秒 10M+ 数据包

- **PDR** 容量：数百万 PDR，仅受内核内存限制
- 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

有关性能调优，请参见 架构指南。

附加文档

组件特定□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□述
- 虚拟化兼容性（Proxmox、VMware、KVM、Hyper-V、VirtualBox）
- NIC 兼容性和 XDP 驱动支持
- 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 附加模式解释（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 XDP 设置
- XDP 验证和故障排除
- XDP 性能的硬件选择

架构指南

深入的技术探讨，包括：

- eBPF 技术基础和程序生命周期
- 带尾调用的 XDP 数据包处理管道
- PFCP 协议实现
- 缓冲架构（GTP-U 封装到端口 22152）
- QoS 滑动窗口速率限制（5ms 窗口）
- 性能特征（3.5µs 延迟，10 Mpps/core）

规则管理指南

PFCP 规则参考，包括：

- 数据包检测规则 (PDR) - 流量分类
- 转发操作规则 (FAR) - 带操作标志的路由决策
- QoS 执行规则 (QER) - 带宽管理 (MBR/GBR)
- 使用报告规则 (URR) - 流量跟踪和报告
- 上行和下行数据包流图
- 规则处理逻辑和优先级

监控指南

统计和容量管理，包括：

- N3/N6 接口统计和流量分布
- XDP 处理统计（通过/丢弃/重定向/中止）
- 带颜色编码区域的 eBPF 映射容量监控
- 性能指标（数据包速率、吞吐量、丢包率）
- 容量规划公式和会话估算
- 警报阈值和最佳实践

Web UI 作指南

控制面板使用，包括：

- 仪表板述和导航

- 会话监控（健康/不健康状态）
- 规则检查（PDR、FAR、QER、URR 详细信息）
- 缓冲监控和数据包缓冲状态
- 实时统计仪表板
- eBPF 映射容量可视化
- 配置查看

API 文档

完整的 REST API 参考，包括：

- OpenAPI/Swagger 交互式文档
- API 分页（基于页面和基于偏移）
- PFCP 会话和关联端点
- 数据包检测规则 (PDR) - IPv4 和 IPv6
- 转发操作规则 (FAR)
- QoS 执行规则 (QER)
- 使用报告规则 (URR)
- 数据包缓冲管理
- 统计和监控端点
- 路由管理和 FRR 集成
- eBPF 映射信息
- 配置管理
- 认证和安全指南
- 常见 API 工作流和示例

指标参考

Prometheus 指标文档，包括：

- PFCP 消息指标（每个对等体的计数、延迟、错误）
- XDP 操作指标（数据平面裁决）
- 数据包指标（带数据包类型标签的协议级计数）
- PFCP 会话和关联指标（每个控制平面节点）
- URR 指标（每个 PFCP 对等体的流量）

- 数据包缓冲指标（缓冲状态、容量、吞吐量）
- 下行数据报告通知指标（DLDR 跟踪）
- eBPF 映射容量指标（资源利用）
- Prometheus 配置示例
- Grafana 仪表板建议

PFCP 原因代码参考

PFCP 错误代码文档，包括：

- 原因代码定义和 3GPP 合规性（TS 129.244）
- 每个原因代码发生的情况（成功、客户端错误、服务器错误）
- 常见故障场景及解决方案
- 使用 Prometheus 指标进行故障排除
- 关联设置和会话生命周期失败
- 高拒绝率的调试步骤
- 原因代码的警报建议

UE 路由管理指南

FRR 路由集成，包括：

- FRR（自由范围路由）□述和架构
- UE 路由同步生命周期
- 自动路由同步到路由守护进程
- 通过 OSPF 和 BGP 的路由广告
- OSPF 邻居监控
- OSPF 外部 LSA 数据库验证
- BGP 对等会话管理
- Web UI 路由监控界面
- 手动路由同步操作
- 路由流和架构的 Mermaid 图

IPv6 / 双栈指南

用户平面 IPv6 操作用于 IPv6 和 IPv4v6 PDN 会话：

- 内核前提条件（启用 IPv6 和转发）
- UE IPv6 地址池配置
- UE `/128` 主机路由的 OSPFv3 广告
- IPv6 下行/上行数据平面行为
- 故障排除 IPv6 转发和路由广告

故障排除指南

全面的问题诊断，包括：

- 快速诊断检查表和工具
- 安装和配置问题
- PFCP 关联失败
- 数据包处理问题
- XDP 和 eBPF 错误
- 性能下降
- 特定于虚拟化的故障（Proxmox、VMware、VirtualBox）
- NIC 和驱动程序问题
- 逐步解决程序

按用例的文档

安装和配置 **OmniUPF**

1. 从本指南开始以获取概述
2. 配置指南 获取设置参数
3. Web UI 指南 访问控制面板

在 **Proxmox** 上部署

1. XDP 模式指南 - Proxmox 本地 XDP 设置 - 从这里开始以提高性能
2. 配置指南 - 虚拟化兼容性
3. 配置指南 - Proxmox SR-IOV 设置
4. 故障排除 - Proxmox 问题

优化性能

1. XDP 模式指南 - 启用本地 **XDP** 以获得 **5-10** 倍的性能提升
2. 架构指南 - 性能优化
3. 配置指南 - XDP 模式
4. 监控指南 - 性能指标
5. 故障排除 - 性能问题

理解数据包处理

1. 架构指南 - 数据包处理管道
2. 规则管理指南
3. 监控指南 - 统计信息

规划容量

1. 配置指南 - 映射大小
2. 监控指南 - 容量规划
3. 监控指南 - 会话容量估算

管理 **UE** 路由和 **FRR** 集成

1. UE 路由管理指南 - 完整的路由集成指南
2. API 文档 - 路由管理 - 路由 API 端点
3. Web UI 指南 - 路由页面操作
4. UE 路由管理 - FRR 验证 - OSPF LSA 验证

使用 **REST API**

1. API 文档 - 完整的 API 参考
2. API 文档 - Swagger UI - 交互式 API 探索器
3. API 文档 - API 特性 - API 使用示例
4. Web UI 指南 - Web 界面作为 API 客户端示例

故障排除问题

1. 故障排除指南 - 从这里开始
2. 监控指南 - 检查统计信息和容量

3. Web UI 指南 - 使用控制面板诊断

快速参考

常见 API 端点

OmniUPF 提供用于监控和管理的 REST API:

```
# 状态和健康
GET http://localhost:8080/api/v1/upf_status

# PFCP 关联
GET http://localhost:8080/api/v1/upf_pipeline

# 会话
GET http://localhost:8080/api/v1/sessions

# 统计
GET http://localhost:8080/api/v1/packet_stats
GET http://localhost:8080/api/v1/xdp_stats

# 容量监控
GET http://localhost:8080/api/v1/map_info

# 缓冲统计
GET http://localhost:8080/api/v1/upf_buffer_info
```

有关完整的 API 文档，请访问 Swagger UI，地址为 `http://<upf-ip>:8080/swagger/index.html`

重要配置参数

```
# /etc/omniupf/runtime.exs
xdp_interfaces = "eth0"              # N3/N6/N9 流量的接口
xdp_attach_mode = "native"           # "generic" | "native" | "offload"
n3_address = "10.100.50.233"         # N3 接口 IP
pfcp_address = "10.100.50.241"       # PFCP 监听地址
pfcp_port = 8805                     # PFCP 端口
node_id = "10.100.50.241"            # PFCP 节点 ID

# 容量
max_sessions = 100_000               # 最大并发会话数

# API
api_port = 8080                      # REST API 端口
```

重要监控阈值

- **eBPF 映射容量 < 70%**: 正常操作
- **eBPF 映射容量 70-90%**: 计划在 1 周内增加容量
- **eBPF 映射容量 > 90%**: 危急 - 需要立即采取行动
- **数据包丢失率 < 0.1%**: 优秀
- **数据包丢失率 0.1-1%**: 良好 - 小问题
- **数据包丢失率 > 5%**: 危急 - 立即调查
- **XDP 中止 > 0**: eBPF 程序存在严重问题

3GPP 标准参考

OmniUPF 实现以下 3GPP 规范:

规范	标题	相关性
TS 23.501	5G 系统 (5GS) 的系统架构	5G UPF 架构和接口
TS 23.401	E-UTRAN 接入的通用分组无线服务 (GPRS) 增强	LTE UPF (PGW-U) 架构
TS 29.244	控制平面与用户平面节点之间的接口 (PFCP)	N4 PFCP 协议
TS 29.281	通用分组无线系统 (GPRS) 隧道协议用户平面 (GTPv1-U)	GTP-U 封装
TS 23.503	5G 系统 (5GS) 的策略和计费控制框架	QoS 和计费
TS 29.212	策略和计费控制 (PCC)	QoS 执行

术语表

5G 架构术语

- **3GPP**: 第三代合作伙伴计划 - 移动通信标准机构
- **AMF**: 接入和移动管理功能 - 5G 核心网络元素，用于接入控制
- **CHF**: 计费功能 - 5G 计费系统
- **DN**: 数据网络 - 外部网络（互联网、IMS、企业）
- **eNodeB**: 演进节点 B - LTE 基站
- **F-TEID**: 完全合格的隧道端点标识符 - 带 IP 地址的 GTP-U 隧道 ID
- **gNB**: 下一代节点 B - 5G 基站
- **GTP-U**: 通用分组无线服务隧道协议用户平面 - 用户数据的隧道协议
- **MBR**: 最大比特率 - QoS 参数，最大允许带宽
- **GBR**: 保证比特率 - QoS 参数，保证的最小带宽
- **N3**: RAN 和 UPF 之间的接口（用户平面流量）
- **N4**: SMF 和 UPF 之间的接口（PFCP 控制）
- **N6**: UPF 和数据网络之间的接口（用户平面流量）
- **N9**: 两个 UPF 之间的接口（UPF 之间的用户平面流量）
- **PCF**: 策略控制功能 - 5G 策略服务器

- **PDU**: 协议数据单元 - 5G 中的数据会话
- **PGW-C**: PDN 网关控制平面 - LTE 控制平面，相当于 SMF
- **PGW-U**: PDN 网关用户平面 - LTE 用户平面（UPF 相当于）
- **QFI**: QoS 流标识符 - 5G QoS 流标记
- **QoS**: 服务质量 - 流量优先级和带宽管理
- **RAN**: 无线接入网络 - 基站网络（gNB/eNodeB）
- **SEID**: 会话端点标识符 - PFCP 会话 ID
- **SMF**: 会话管理功能 - 5G 核心网络元素，用于会话控制
- **TEID**: 隧道端点标识符 - GTP-U 隧道 ID
- **UE**: 用户设备 - 移动设备
- **UPF**: 用户平面功能 - 5G 数据包转发网络元素

PFCP 协议术语

- **Association**: SMF 和 UPF 之间的控制关系
- **FAR**: 转发动作规则 - 决定数据包转发行为
- **IE**: 信息元素 - PFCP 消息组件
- **Node ID**: UPF 或 SMF 标识符 (FQDN 或 IP 地址)
- **PDR**: 数据包检测规则 - 将数据包分类到流中
- **PFCP**: 数据包转发控制协议 - N4 控制协议
- **QER**: QoS 执行规则 - 应用带宽限制和标记
- **SDF**: 服务数据流 - 应用特定的流量过滤器
- **Session**: 表示 UE PDU 会话或 PDP 上下文的 PFCP 会话
- **URR**: 使用报告规则 - 跟踪计费的数据量

eBPF 和 Linux 内核术语

- **BPF**: 伯克利数据包过滤器 - 内核数据包过滤技术
- **eBPF**: 扩展 BPF - 可编程内核数据路径
- **Hash Map**: eBPF 键值存储，用于快速查找
- **XDP**: eXpress 数据路径 - 驱动级别的内核数据包处理
- **Verifier**: 验证 eBPF 程序安全性的内核组件
- **Map**: 内核和用户空间之间共享的 eBPF 数据结构

- **Zero-copy**: 无需复制到用户空间的数据包处理

OmniUPF 产品术语

- **OmniUPF**: 基于 eBPF 的用户面功能（此产品）
- **Datapath**: 数据包处理引擎（eBPF 程序）
- **Control Plane**: PFCP 处理程序和会话管理
- **REST API**: 用于监控和管理的 HTTP API
- **Web UI**: 基于浏览器的控制面板

OmniUPF 架构指南

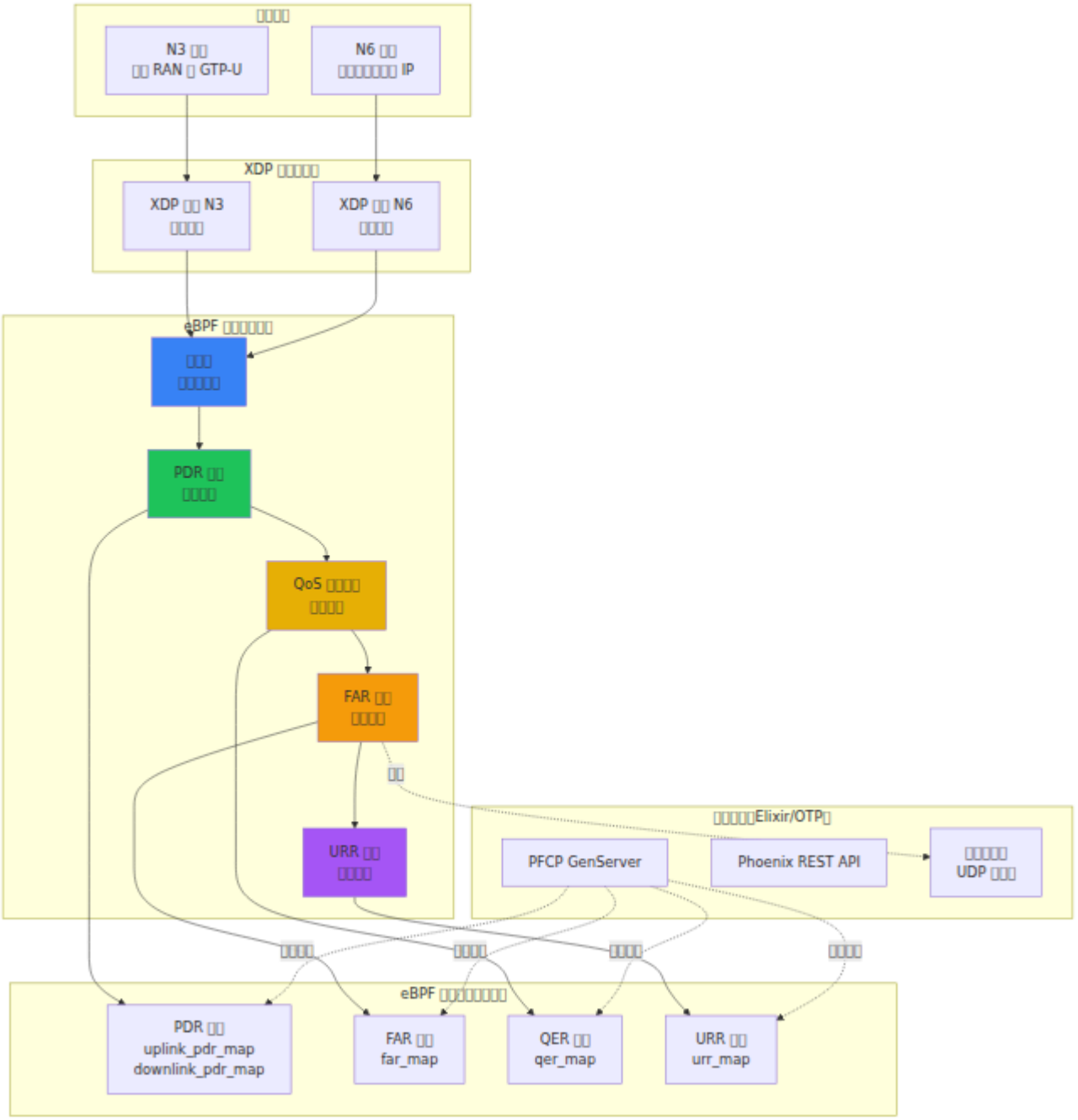
目录

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。控制平面使用 Elixir/OTP 实现，基于 GenServer 的 PFCP 会话管理，而数据平面直接在 Linux 内核中运行 eBPF 程序。这种分离消除了用户空间数据包处理的开销，并实现了微秒级延迟的多千兆吞吐量。

eBPF 对象文件（`ipentrypoint_bpf.o`）是从 C 源代码（在 `ebpf/xdp/` 目录中）编译而成，并通过 NIF 绑定到 libbpf 在运行时由 Elixir 控制平面加载。

⬚⬚⬚


N3 ⬚⬚
⬚⬚ RAN ⬚ GTP-U
N6 ⬚⬚
⬚⬚⬚⬚⬚⬚⬚ IP
XDP ⬚⬚⬚⬚⬚
XDP ⬚⬚ N3
⬚⬚⬚⬚
XDP ⬚⬚ N6
⬚⬚⬚⬚
eBPF ⬚⬚⬚⬚⬚⬚
⬚⬚⬚
⬚⬚⬚⬚⬚⬚
PDR ⬚⬚
⬚⬚⬚⬚
QoS ⬚⬚⬚⬚
⬚⬚⬚⬚
FAR ⬚⬚
⬚⬚⬚⬚
⬚⬚
URR ⬚⬚
⬚⬚⬚⬚
⬚⬚⬚⬚⬚Elixir/OTP⬚
PFCP GenServer
Phoenix REST API
⬚⬚⬚⬚⬚⬚
UDP ⬚⬚⬚
⬚⬚⬚⬚
⬚⬚⬚⬚
⬚⬚⬚⬚
⬚⬚⬚⬚
eBPF ⬚⬚⬚⬚⬚⬚⬚⬚
PDR ⬚⬚
uplink_pdr_map
downlink_pdr_map
FAR ⬚⬚
far_map
QER ⬚⬚
qer_map
URR ⬚⬚
urr_map

⬚⬚⬚⬚⬚⬚

⬚⬚⬚⬚⬚⬚

- ⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚
- ⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚
- ⬚⬚ XDP ⬚⬚⬚⬚⬚⬚⬚

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
- 没有互斥锁/自旋锁开销

硬件卸载准备：

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

eBPF 技术基础

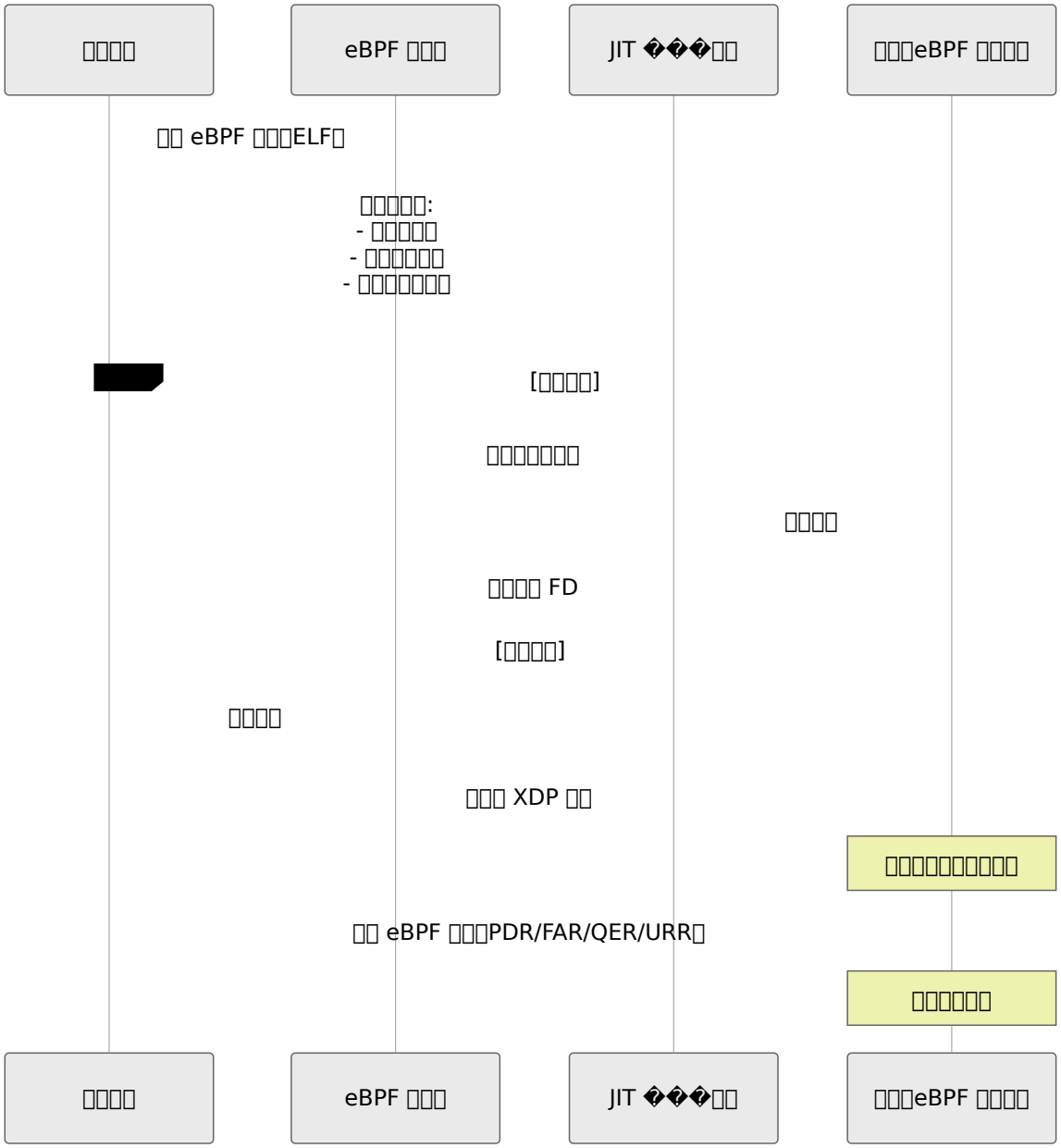
什么是 eBPF？

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、沙箱化的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：以原生内核速度运行（没有解释开销）
- 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 □□□□□□□□□□

映射类型	描述	用例
`BPF_MAP_TYPE_HASH`	带键值对的哈希表	按 TEID 或 UE IP 查找 PDR
`BPF_MAP_TYPE_ARRAY`	按整数索引的数组	按 ID 查找 QER、FAR、URR
`BPF_MAP_TYPE_PERCPU_HASH`	每 CPU 的哈希表（无锁）	高性能 PDR 查找
`BPF_MAP_TYPE_LRU_HASH`	LRU（最近最少使用）哈希	自动驱逐旧条目

映射□作:

- 查找: O(1) 哈希查找（亚微秒级）
- 更新: 来自用户空间的原子更新
- 删除: 立即移除条目
- 迭代: 批量操作用于映射转储

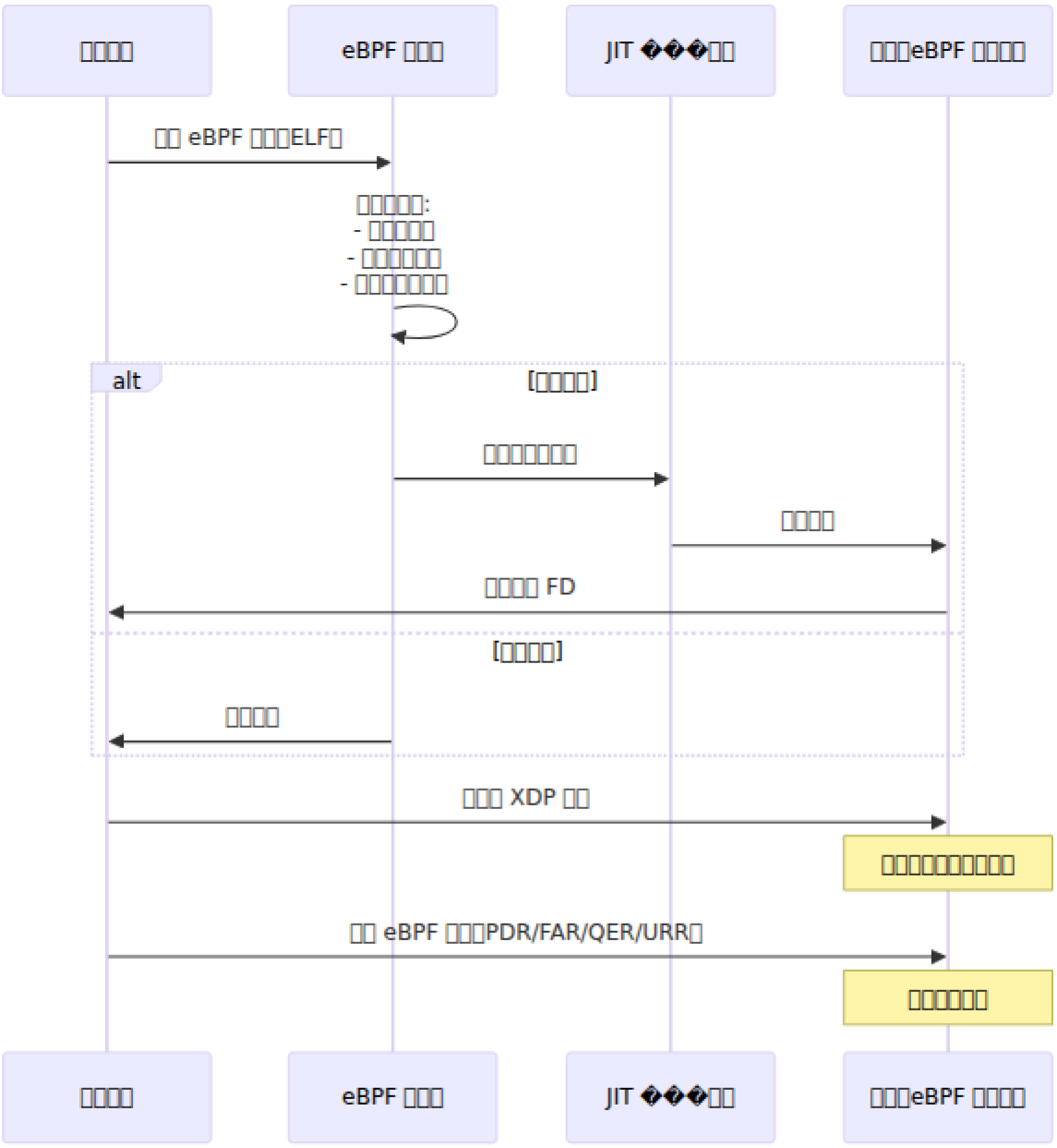
XDP 数据路径

XDP 概述

XDP（快速数据路径）是一个 Linux 内核钩子，允许 eBPF 程序在网络驱动接收数据包后、内核网络栈之前的最早时刻处理数据包。

XDP 附加模式

OmniUPF 支持三种 XDP 附加模式，每种模式具有不同的性能和兼容性特征。



1. XDP 卸载模式

硬件执行（最佳性能）：

- eBPF 程序直接在 SmartNIC 硬件上运行
- 数据包处理在 NIC 中进行，无需触及 CPU
- 实◆◆ 100 Gbps+ 吞吐量
- 需要兼容的 SmartNIC（Netronome 、Mellanox ConnectX-6）

配置（在 `runtime.exs` 中）：

```
xdp_attach_mode = "offload"
```

限制：

- 需要昂贵的 SmartNIC 硬件
- 限制 eBPF 程序复杂性
- 并非所有 eBPF 特性在硬件中受支持

---

2. XDP 原生模式（生产环境的默认模式）

驱动级执行（高性能）：

- eBPF 程序在网络驱动上下文中运行
- 数据包在 SKB（套接字缓冲区）分配之前处理
- 每个核心实现 10-40 Gbps
- 需要支持 XDP 的驱动（大多数现代驱动）

配置（在 `runtime.exs` 中）：

```
xdp_attach_mode = "native"
```

优点：

- 性能非常高（每秒数百万数据包）
- 广泛的硬件兼容性
- 完整的 eBPF 特性集

支持的驱动：

- 英特尔：i40e、ice、ixgbe、igb
- Mellanox：mlx4、mlx5
- 博通：bnxt
- 亚马逊：ena
- 大多数 10G+ 网络卡

---

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（在 `runtime.exs` 中）：

```
xdp_attach_mode = "generic"
```

用例:

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

性能: 1-5 Gbps（显著慢于原生/卸载）

XDP 返回代码

eBPF 程序返回 XDP 动作代码，以告知内核对数据包的处理方式:

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`	将数据包通过相同接口发送回去	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

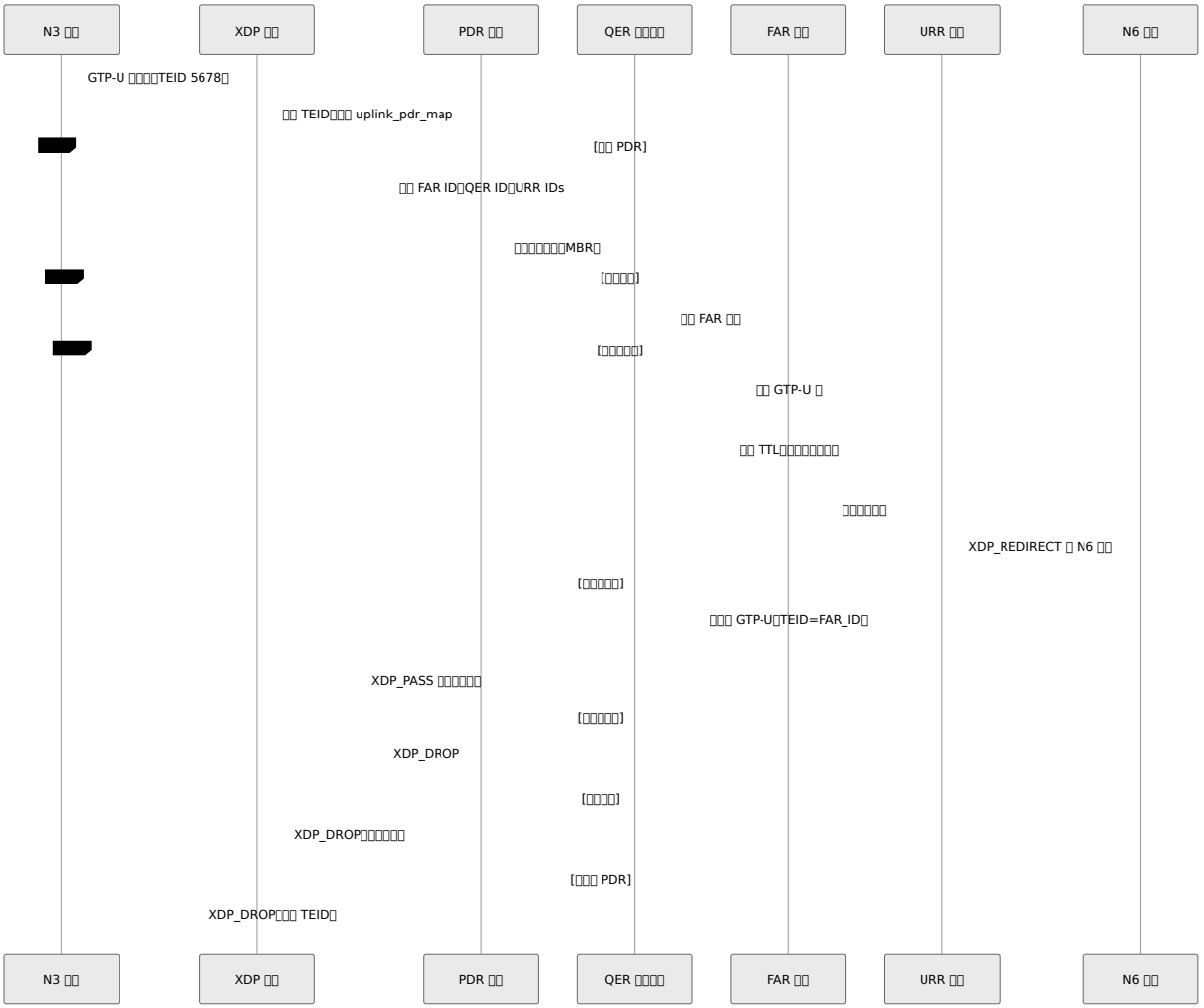
程序结构

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（有限的栈深度）
- 使模块化管道设计成为可能
- 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 实现了用于切换场景的数据包缓冲，通过将数据包封装在 GTP-U 中并通过 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

当启用缓冲时（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);  // 自 IP 头
```

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;  // 保留源地址以避免火星过滤
ip->daddr = local_upf_ip;        // 用户空间监听器绑定的本地 IP
ip->protocol = IPPROTO_UDP;
ip->ttl = 64;
```

4. 构建 **UDP** 头:

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

5. 构建 **GTP-U** 头:

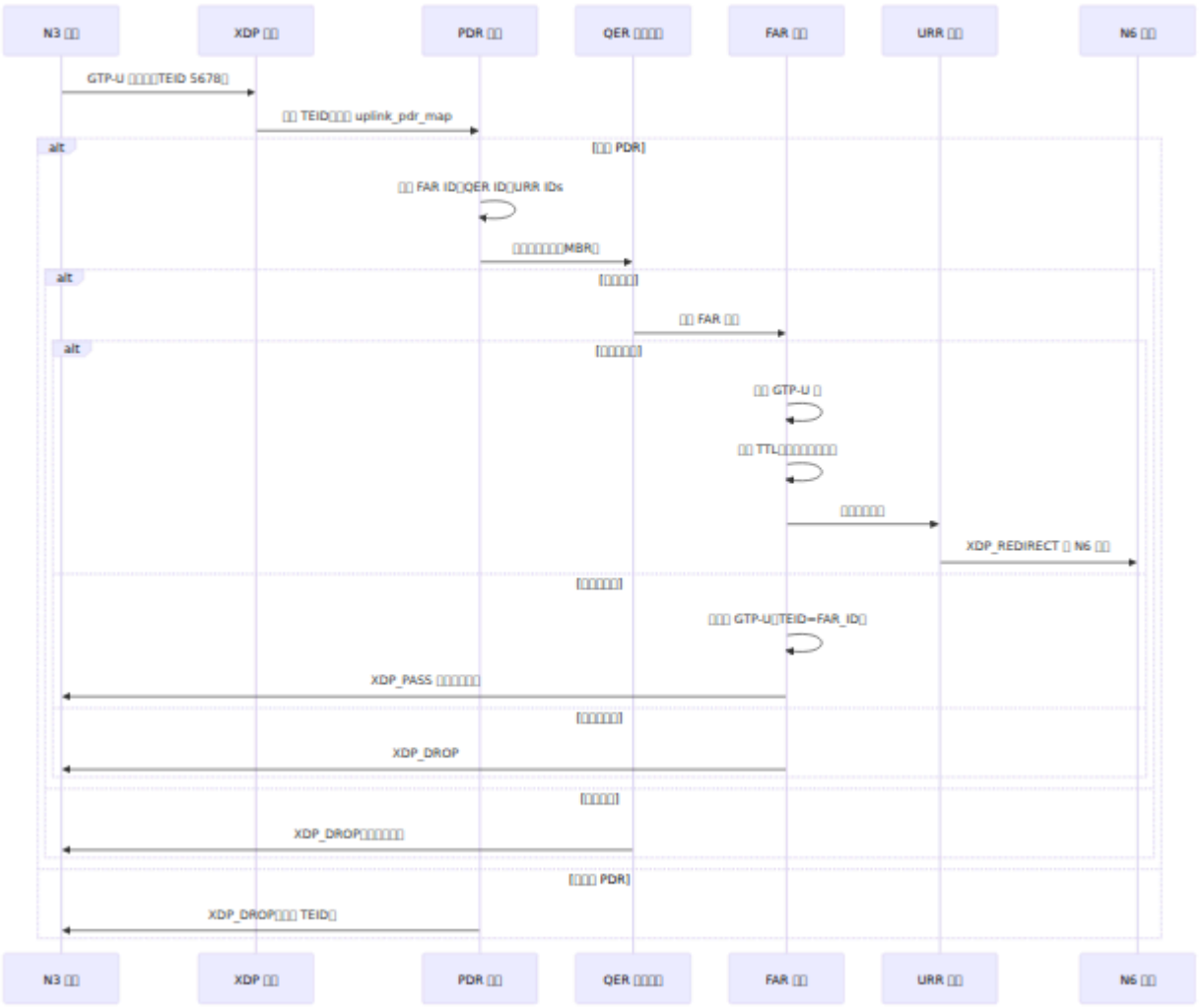
```
gtp->version = 1;
gtp->message_type = GTPU_G_PDU;
gtp->teid = htonl(far_id | (direction << 24));  // 编码 FAR ID 和方向
gtp->message_length = htons(orig_packet_len);
```

6. 返回 **XDP_PASS**:

   - 内核将数据包交付给本地 UDP 套接字，端口为 22152
   - 用户空间缓冲管理器接收并存储数据包

缓冲刷新□作

当切换完成时，SMF 更新 FAR 以清除 BUFFER 标志。缓冲的数据包被重放:



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	最大总缓冲数据包数
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 强制执行

速率限制算法

OmniUPF 实现了 滑动窗口速率限制器 以进行 QoS 强制执行。

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`）：

```c
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 意味着无限制
    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（数据包将在未□传输）
    __u64 start = *window_start;
    if (start + tx_time > now)
        return XDP_DROP;  // 超出速率限制

    // 如果窗口已过，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：使用易变指针进行并发访问
- **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:

   - 如果上一个数据包在 `t=0` 传输，则下一个数据包可以在 `t=120µs` 传输
   - 如果数据包在 `t=100µs` 到达，则被丢弃（太早）
   - 如果数据包在 `t=150µs` 到达，则被转发（窗口已前进）

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, 单核心)	10 Gbps	8 Mpps	2-5 µs
XDP 原生 (10G NIC, 4 核心)	40 Gbps	32 Mpps	2-5 µs
XDP 通用	1-5 Gbps	0.8-4 Mpps	50-100 µs

延迟细分

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 ~3.5 μs（XDP 原生，10G NIC）

CPU 利用率

每核心处理能力：

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

数据包速率的 **CPU** 使用情况：

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

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

可扩展性和调优

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. 为 XDP 处理启用 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 平衡以优化 XDP 核心
systemctl stop irqbalance

# 将 CPU 调频器设置为性能模式
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，峰值 20 Gbps）：

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

相关文档

- **UPF □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 □作指南** - 控制面板使用
- 故障排除指南 - 常见问题和诊断
- 围墙花园指南 - 使用缓冲机制进行用户空间数据包拦截的超出信用重定向

OmniUPF 配置指南

目录

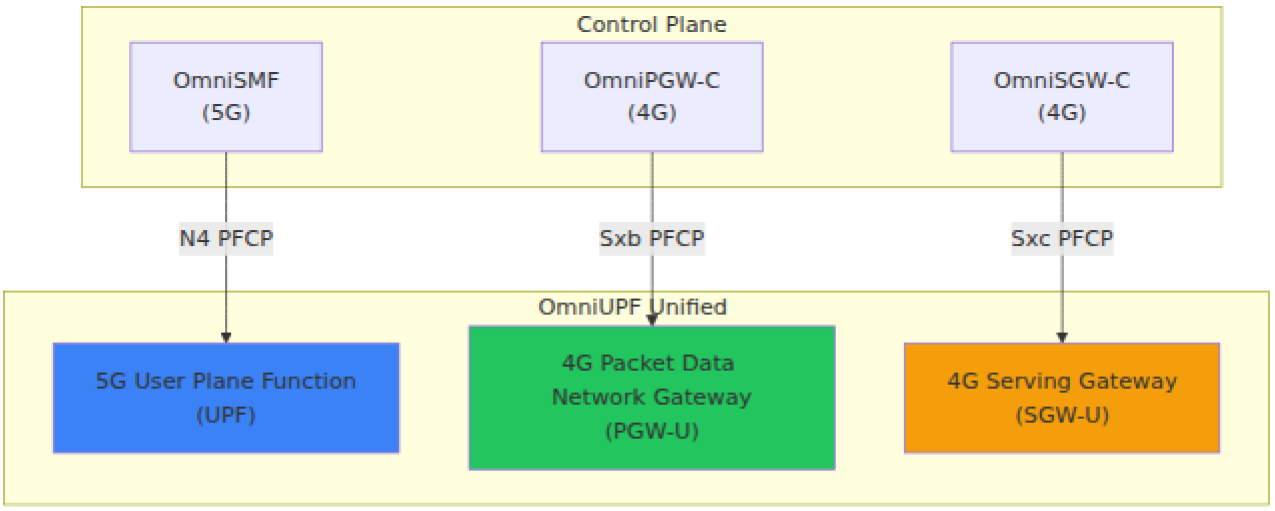
概述

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。控制平面在 Elixir/OTP 中实现，配置通过 Elixir 配置文件 (`runtime.exs`) 管理。

在安装包时，配置文件放置在 `/etc/omniupf/runtime.exs` 并在升级时保留。更改配置后，必须重新启动 UPF。

□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□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

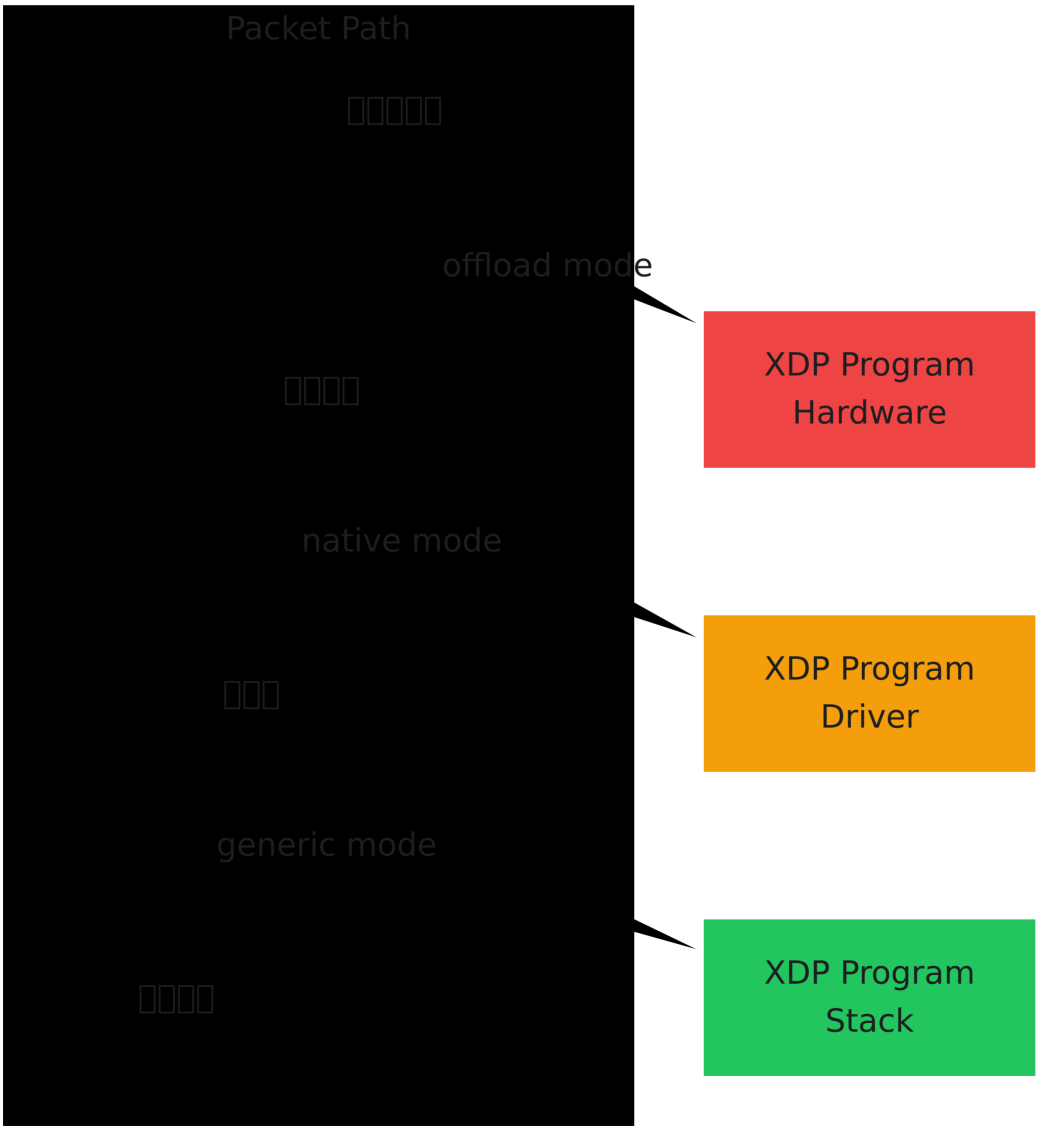
XDP 附加模式

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 **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
原生	网络驱动（内核）	~5-10 Mpps	生产（裸金属、带 SR-IOV 的虚拟机）	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



□□□□□□□□□

□□□XDP □□□□□□□□□□□□□

□□□

- □□□ □□ □□□□
- □□□□□□□□□□□□
- □□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□

□□□

- □□□□□□□□□□ ~1-2 Mpps□
- □□□□ XDP □□□□□□□□□

□□□□ `runtime.exs` □□□

```
xdp_attach_mode = "generic"
```

□□□□□

- □□□ SR-IOV □□□□
- □□□□□□□□
- □□□ XDP □□□ NIC
- Proxmox□VMware□VirtualBox □□□□□□□□□

□□□□□□□□□

□□□XDP □□□□□□□□□□□□□

□□□

- □□□□□□□□□ ~5-10 Mpps□
- □□□□□□□□□□□□□□□
- □□□□□□□□□□□

- 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/驱动都支持原生 XDP

配置（在 `runtime.exs` 中）：

```
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
- 支持 XDP 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+ 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（在 `runtime.exs` 中）：

```
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
`xdp_interfaces`	N3/N6/N9 流量的网络接口（以逗号分隔，或 `"auto"` 表示所有非回环）	字符串	`"auto"`
`gtpu_interface`	承载所有 GTP-U（N3/N9/S5-S8）的 NIC。与 `sgi_interface` 一起设置，以将 GTP-U 仅限 XDP 程序附加到此 NIC；未设置则为组合程序	字符串	`nil`
`sgi_interface`	承载纯 IP SGi/N6 的 NIC。与 `gtpu_interface` 一起设置，以将下行仅限 XDP 程序附加到此 NIC	字符串	`nil`
`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（在 `runtime.exs` 中）：

```
xdp_interfaces = "eth0,eth1"
n3_address = "10.100.50.233"
n9_address = "10.100.50.234"
```

每 **NIC** 数据路径拆分。当 GTP-U 和 SGi 通过不同的 NIC 传输时，命名每个 NIC，以便加载器为每个 NIC 附加单一用途的 XDP 程序（在 `gtpu_interface` 上仅限 GTP-U，在 `sgi_interface` 上仅限下行），而不是组合的“on-a-stick”程序。两个都必须出现在 `xdp_interfaces` 中。将两个都留空（默认）以将组合程序附加到每个接口。

```
xdp_interfaces = "eth1,eth2"
gtpu_interface = "eth1"   # 所有 GTP-U（N3/N9/S5-S8）
sgi_interface  = "eth2"   # 纯 IP SGi/N6
```

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）	IP 字符串	`"127.0.0.1"`
`pfcp_port`	PFCP 端口	整数	`8805`
`node_id`	PFCP 协议的本地节点 ID	IP 字符串	`"127.0.0.1"`
`heartbeat_interval_ms`	PFCP 心跳间隔（毫秒）	整数	`5000`
`heartbeat_timeout_ms`	PFCP 心跳超时（毫秒）	整数	`5000`
`heartbeat_retries`	声明对等方死亡之前的心跳重试次数	整数	`3`
`association_setup_timeout`	当 UPF 发起关联时，等待 PFCP 关联设置响应的超时（毫秒）	整数	`5000`
`pfcp_remote_nodes`	UPF 在启动时主动建立关联的控制平面 IP 地址字符串列表（UPF 发起）。为空表示 UPF 仅接受由控制平面发起的关联。	字符串列表	`[]`

示例（在 `runtime.exs` 中）：

```
pfcp_address = "10.100.50.241"
pfcp_port = 8805
node_id = "10.100.50.241"
heartbeat_interval_ms = 10_000
heartbeat_retries = 5
association_setup_timeout = 5_000
pfcp_remote_nodes = []                # 或例如 ["10.100.50.20",
"10.100.50.22"]
```

API 和监控

参数	描述	类型	默认
`api_port`	REST API 服务器（Phoenix）的本地端口	整数	`8080`
`log_level`	日志级别（`:debug`、`:info`、`:warning`、`:error`）	原子	`:info`

示例（在 `runtime.exs` 中）：

```
api_port = 8080
log_level = :debug
```

GTP 路径管理

参数	描述	类型	默认
`gtpu_port`	GTP-U 端口	整数	`2152`
`gtp_echo_interval`	默认回声间隔（秒）	整数	`10`
`gtp_echo_retries`	路径失败之前最大丢失的回声次数	整数	`3`
`gtp_peers`	GTP 对等体的回声请求保持活动列表	列表	`[]`

示例（在 `runtime.exs` 中）：

```
gtp_echo_interval = 15
gtp_echo_retries = 3
gtp_peers = [
  %{address: parse_ip.("10.100.50.50"), echo: true, echo_interval:
10},
  %{address: parse_ip.("10.100.50.60"), echo: false},
]
```

---

eBPF 映射容量

参数	描述	类型	默认
`max_sessions`	最大并发会话数	整数	`1_000_000`
`far_map_size`	FAR eBPF 映射的大小	整数	`131_070`
`qer_map_size`	QER eBPF 映射的大小	整数	`65_535`
`urr_map_size`	URR eBPF 映射的大小	整数	`65_535`

示例（在 `runtime.exs` 中）：

```
max_sessions = 100_000
far_map_size = 131_070
qer_map_size = 65_535
urr_map_size = 131_070
```

缓冲区配置

下行数据包在 UE 空闲时被缓冲，然后在 UE 被寻呼时刷新。这些参数限制了缓冲的数量和时间。

参数	描述	类型	默认
`buffer_port`	来自 eBPF 的缓冲数据包的 UDP 端口	整数	`22152`
`buffer_max_total`	在所有 FAR 中缓冲的最大数据包数（全局限制）	整数	`10_000`
`buffer_max_per_far`	每个 FAR 中缓冲的最大数据包数	整数	`100`
`buffer_ttl_ms`	缓冲数据包在被丢弃之前保持的时间（毫秒）	整数	`30_000`
`buffer_cleanup_interval_ms`	过期缓冲数据包被清理的间隔（毫秒）	整数	`5_000`

示例（在 `runtime.exs` 中）：

```
buffer_port = 22152
buffer_max_total = 10_000
buffer_max_per_far = 100
buffer_ttl_ms = 30_000
buffer_cleanup_interval_ms = 5_000
```

由于这些限制导致的丢弃由 `upf_buffer_packets_dropped_total` 计数（标签 `global_limit`、`far_limit`、`expired`）。请参见 指标参考。

功能标志

参数	描述	类型	默认
`feature_ueip`	启用 OmniUPF 的 UE IP 分配	布尔	`false`
`ueip_pool`	UE IP 分配的 IP 池（需要 `feature_ueip`）	CIDR 字符串	`"10.60.0.0/24"`
`feature_ftup`	启用 OmniUPF 的 F-TEID 分配	布尔	`true`
`teid_pool_start`	TEID 分配范围起始	整数	`1`
`teid_pool_end`	TEID 分配范围结束	整数	`10_000_000`

示例（**UE IP** 分配）（在 `runtime.exs` 中）：

```
feature_ueip = true
ueip_pool = "10.45.0.0/16"
```

示例（**F-TEID** 分配）（在 `runtime.exs` 中）：

```
feature_ftup = true
teid_pool_start = 1
teid_pool_end = 1_000_000
```

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	`true`
`route_manager_type`	路由守护进程类型（`"frr"` 或 `"static"`）	字符串	`"frr"`
`route_manager_nexthop`	注入 UE 路由的下一跳地址（空 = 直接连接）	字符串	`""`
`route_manager_vtysh_path`	FRR `vtysh` 二进制文件的路径（在 `route_manager_type` 为 `"frr"` 时使用）	字符串	`"/usr/bin/v`
`route_manager_sync_interval_ms`	跟踪路由与路由守护进程之间的完整协调间隔（毫秒）	整数	`600_000`
`route_batch_interval_ms`	批量增量路由添加/删除操作的去抖动窗口（毫秒）	整数	`1_000`

示例（在 `runtime.exs` 中）：

```
route_manager_enabled = true
route_manager_type = "frr"
route_manager_nexthop = ""
route_manager_vtysh_path = "/usr/bin/vtysh"
route_manager_sync_interval_ms = 600_000
route_batch_interval_ms = 1_000
```

何时启用：

- 需要路由广告的多 UPF 部署

- □ OSPF □ BGP □□□□□□□□
- □□□□□□□□ FRRouting □□□□

---

eBPF / XDP □□

□□	□□	□□	□□
`xdp_interfaces`	□□ XDP □□□□□□□□□□□□□□□□ `"auto"` □□□□□□□□	□□□	`"auto"`
`xdp_attach_mode`	XDP □□ `"generic"` □ `"native"` □ `"offload"`	□□□	`"generic"`
`ebpf_pin_path`	BPF □□□□□□□□□□	□□□	`"/sys/fs/bpf/upf_pipeli`
`xdp_obj_path`	□□□ eBPF □□□□□□	□□□	`"/etc/omniupf/ipentrypo`
`ebpf_backend`	□□□□□□□□□□ `:nif` □□□ NIF□□□□□□ `:port` □□□□□□□ □□□□ `:mock` □□□□□□□□□□ eBPF □□□□	□□	`:nif`

□□□□ `runtime.exs` □□□

```
xdp_interfaces = "auto"              # 或 "eth0,eth1" 用于特定接口
xdp_attach_mode = "native"
ebpf_pin_path = "/sys/fs/bpf/upf_pipeline"
xdp_obj_path = "/etc/omniupf/ipentrypoint_bpf.o"
ebpf_backend = :nif
```

配置文件

Elixir 配置文件 (runtime.exs)

文件：`/etc/omniupf/runtime.exs`

配置文件使用 Elixir 语法，采用简单的变量赋值以提高可读性。变量通过 `config :upf_ex, ...` 在文件底部应用于应用程序配置。

```
import Config

parse_ip = fn str ->
  {:ok, addr} = :inet.parse_address(String.to_charlist(str))
  addr
end

#
============================================================================
# PFCP (N4/Sx) 配置
#
============================================================================
pfcp_address = "10.100.50.241"        # 监听 PFCP 消息的 IP 地址
pfcp_port = 8805                       # PFCP 端口（标准：8805）
node_id = "10.100.50.241"             # 在关联设置中通告的节点 ID

#
============================================================================
# GTP-U 数据平面配置
#
============================================================================
n3_address = "10.100.50.233"          # N3 接口 IP（GTP-U 指向 RAN）
n9_address = n3_address                # N9 接口 IP（默认为 N3）
gtpu_port = 2152                       # GTP-U 端口（标准：2152）
buffer_port = 22152                    # 缓冲监听端口

#
============================================================================
# eBPF / XDP 配置
#
============================================================================
xdp_interfaces = "auto"               # "auto" 表示所有非回环，或以逗号分隔的列表
xdp_attach_mode = "native"            # XDP 模式："generic"、"native" 或 "d
ebpf_pin_path = "/sys/fs/bpf/upf_pipeline"
xdp_obj_path = "/etc/omniupf/ipentrypoint_bpf.o"

#
============================================================================
# 会话□资源池配置
#
============================================================================
max_sessions = 100_000
teid_pool_start = 1
```

```
teid_pool_end = 10_000_000
far_map_size = 131_070
qer_map_size = 65_535
urr_map_size = 65_535

#
============================================================================
# □□□□
#
============================================================================
feature_ueip = true
feature_ftup = true
ueip_pool = "10.45.0.0/16"

#
============================================================================
# □□□□
#
============================================================================
route_manager_enabled = true
route_manager_type = "frr"

#
============================================================================
# □□□□
#
============================================================================
heartbeat_interval_ms = 10_000
heartbeat_timeout_ms = 5_000
heartbeat_retries = 5

#
============================================================================
# GTP-U □□□□□□□
#
============================================================================
gtp_echo_interval = 10
gtp_echo_retries = 3
gtp_peers = [
  %{address: parse_ip.("10.100.50.50"), echo: true, echo_interval: 10
]

#
============================================================================
```

```
# HTTP API □□□□□
#
================================================================================
api_port = 8080
log_level = :info

#
================================================================================
# □□□□□□□□□□□□□□□□□□
#
================================================================================
config :upf_ex,
  pfcp_address: parse_ip.(pfcp_address),
  pfcp_port: pfcp_port,
  n3_address: parse_ip.(n3_address),
  n9_address: parse_ip.(n9_address),
  node_id: parse_ip.(node_id),
  api_port: api_port,
  ebpf_pin_path: ebpf_pin_path,
  xdp_obj_path: xdp_obj_path,
  interface_names: String.split(xdp_interfaces, ","),
  xdp_attach_mode: xdp_attach_mode,
  feature_ueip: feature_ueip,
  feature_ftup: feature_ftup,
  ueip_pool: ueip_pool,
  teid_pool_start: teid_pool_start,
  teid_pool_end: teid_pool_end,
  far_map_size: far_map_size,
  qer_map_size: qer_map_size,
  urr_map_size: urr_map_size,
  max_sessions: max_sessions,
  route_manager_enabled: route_manager_enabled,
  route_manager_type: route_manager_type,
  buffer_port: buffer_port,
  gtpu_port: gtpu_port,
  heartbeat_interval_ms: heartbeat_interval_ms,
  heartbeat_timeout_ms: heartbeat_timeout_ms,
  heartbeat_retries: heartbeat_retries,
  gtp_echo_interval: gtp_echo_interval,
  gtp_echo_retries: gtp_echo_retries,
  gtp_peers: gtp_peers

config :logger, level: log_level
```

管理服务

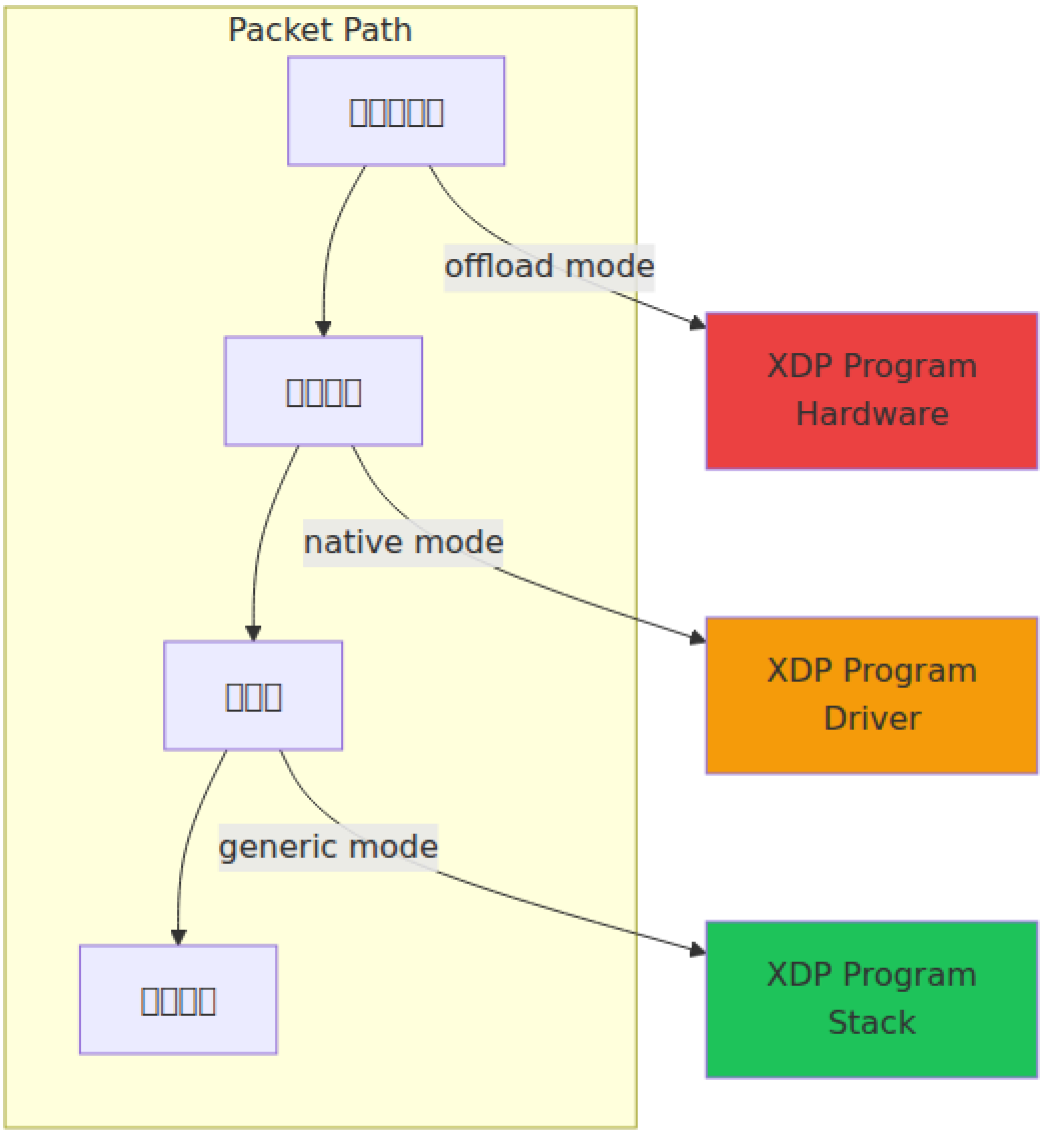
```
# 启动 UPF
sudo systemctl start omniupf

# 停止 UPF
sudo systemctl stop omniupf

# 在配置更改后重启
sudo systemctl restart omniupf

# 检查状态
sudo systemctl status omniupf

# 查看日志
sudo journalctl -u omniupf -f

# 或直接使用发布二进制文件
sudo /opt/omniupf/bin/upf_ex start
sudo /opt/omniupf/bin/upf_ex stop
sudo /opt/omniupf/bin/upf_ex daemon   # 作为后台守护进程启动
```

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原生 **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：标准虚拟机网络

配置：

- 网络设备：VirtIO 或 E1000
- XDP 模式：`generic`
- 性能：~1-2 Mpps

Proxmox 虚拟机设置：

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

OmniUPF 配置（在 `runtime.exs` 中）：

```
xdp_interfaces = "eth0"
xdp_attach_mode = "generic"
```

---

2. SR-IOV 直通（原生 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（在 GRUB 中设置 `intel_iommu=on` 或 `amd_iommu=on`）

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 上启用 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 虚拟机设置：

```
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（在 `runtime.exs` 中）：

```
xdp_interfaces = "ens1f0"    # SR-IOV VF 名称
xdp_attach_mode = "native"
```

---

3. PCI 直通（原生 XDP）

用例：将整个物理 NIC 传递给单个虚拟机

配置：

- 整个物理 NIC 传递给虚拟机

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

Proxmox 虚拟机设置:

```
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 (在 `runtime.exs` 中):

```
xdp_interfaces = "ens1f0"
xdp_attach_mode = "native"    # 或 "offload" 用于 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

标准 **vSwitch**（通用 **XDP**）：

- 网络适配器：VMXNET3
- XDP 模式：`generic`

SR-IOV（原生 **XDP**）：

- 在 ESXi 主机设置中启用 SR-IOV
- 将 SR-IOV 网络适配器添加到虚拟机
- XDP 模式：`native`

Microsoft Hyper-V

虚拟交换机（通用 **XDP**）：

- 网络适配器：合成
- XDP 模式：`generic`

SR-IOV（原生 **XDP**）：

- 在 Hyper-V 管理器中启用 SR-IOV
- 在虚拟网络适配器上配置 SR-IOV
- XDP 模式：`native`

VirtualBox

NAT/桥接模式（仅通用 **XDP**）：

- 网络适配器：VirtIO-Net 或 Intel PRO/1000
- XDP 模式：`generic`
- 注意：VirtualBox 不支持 SR-IOV

NIC 兼容性

理解 **Mpps** 与吞吐量

每秒数据包数（**Mpps**）和吞吐量（**Gbps**）并不直接等同 - 这种关系完全取决于数据包大小。移动网络流量在数据包大小上变化很大，从微小的 VoIP 数据包到大型视频流帧。

数据包大小对吞吐量的影响

在移动网络中，UPF 在 N3 接口上处理 GTP-U 封装的数据包，在 N6 接口上处理原生 IP 数据包。

GTP-U 封装开销（**N3** 接口）：

- 外部 **IPv4** 头：20 字节
- 外部 **UDP** 头：8 字节
- **GTP-U** 头：8 字节
- 总 **GTP-U** 开销：36 字节

最小 **GTP-U** 数据包（**N3**）：

- 内部 **IP** 头：20 字节（IPv4）
- 内部 **UDP** 头：8 字节
- 最小有效载荷：1 字节
- 内部数据包总计：29 字节
- 加上 **GTP-U** 开销：36 字节
- 总数据包大小：65 字节

以 **1 Mpps** 的吞吐量计算最小 **GTP-U** 数据包：

```
65 字节 x 1,000,000 pps x 8 位/字节 = 520 Mbps
```

最大 **GTP-U** 数据包（**N3**，**1500 MTU**）：

- 内部 **IP MTU**：1500 字节（完整内部 IP 数据包）
- 加上 **GTP-U** 开销：36 字节
- 总数据包大小：1536 字节

以 **1 Mpps** 的吞吐量计算最大 **GTP-U** 数据包:

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

原生 **IP** 数据包（**N6** 接口）:

在 N6（指向互联网）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 x 1,000,000 pps x 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 x 1,000,000 pps x 8 位/字节 = 12,000 Mbps = 12 Gbps
```

真实世界性能示例

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

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

在 10 Mpps 的现代移动流量（1200 字节平均）下，预计在 N3 和 N6 接口上的吞吐量约为 **~10 Gbps。

实际容量规划：

以 1200 字节的平均数据包大小（现代移动网络中视频流的典型）：

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

XDP 支持的网络驱动

OmniUPF 需要支持 XDP 的网络驱动以实现 原生 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	原生	~10 Mpps
Intel XL710	i40e	是	原生	~10 Mpps
Intel E810	ice	是	原生	~15 Mpps
Intel 82599ES	ixgbe	是	原生	~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	原生	~12 Mpps
Mellanox ConnectX-6	mlx5	是	原生	~20 Mpps
Mellanox BlueField	mlx5	是	原生 + 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	原生	~8 Mpps
Broadcom NetXtreme II	bnx2x	否	仅通用	~1 Mpps

其他供应商

型号	驱动	XDP 支持	模式	性能
Netronome Agilio CX	nfp	是	卸载	~30 Mpps
Amazon ENA	ena	是	原生	~5 Mpps
Solarflare SFC9xxx	sfc	是	原生	~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 字节的平均数据包大小（现代移动流量）：

□□	□□ NIC	□□	Mpps □□	□□□□□（N3）	□□□□
□□/□□	□□ NIC（VirtIO、E1000）	□□	1-2 Mpps	1-2 Gbps	□□□□□□□□□
□□□□	Intel X710、Mellanox CX-5	□□	5-10 Mpps	5-10 Gbps	□□□□□□□□
□□□□/□□□	Intel E810、Mellanox CX-6	□□	10-20 Mpps	10-20 Gbps	□□□□□□□□□
□□□□	Mellanox CX-6、Intel E810□□□	□□	20-40 Mpps	20-40 Gbps	□□□□□□□□□
□□□□	Mellanox BlueField、Netronome Agilio	□□	40+ Mpps	40+ Gbps	□□□□
Proxmox VM □□□	VirtIO	□□	1-2 Mpps	1-2 Gbps	□□□□□
Proxmox VM □SR-IOV□	Intel X710/E810 VF、Mellanox CX-5 VF	□□	5-10 Mpps	5-10 Gbps	□□□□□

□□□□

□□ XDP □□□

- XDP □□
- □□ XDP □□

NIC □□□□□□

- Cilium XDP □□□□
- IO Visor XDP □□

配置示例

示例 **1**：开发环境（通用模式）

场景：在笔记本电脑或虚拟机上测试 OmniUPF，无需 SR-IOV

```
# 开发配置 (/etc/omniupf/runtime.exs)
xdp_interfaces = "eth0"
xdp_attach_mode = "generic"
api_port = 8080
pfcp_address = "127.0.0.1"
pfcp_port = 8805
node_id = "127.0.0.1"
n3_address = "127.0.0.1"
log_level = :debug
max_sessions = 1_000
```

示例 **2**：生产裸金属（原生模式）

场景：在带 Intel X710 NIC 的裸金属服务器上生产 UPF

```
# 生产裸金属配置 (/etc/omniupf/runtime.exs)
xdp_interfaces = "ens1f0,ens1f1"    # N3 在 ens1f0 上, N6 在 ens1f1 上
xdp_attach_mode = "native"
api_port = 8080
pfcp_address = "10.100.50.241"
pfcp_port = 8805
node_id = "10.100.50.241"
n3_address = "10.100.50.233"
n9_address = "10.100.50.234"
log_level = :info
max_sessions = 500_000
gtp_echo_interval = 30
gtp_peers = [
  %{address: parse_ip.("10.100.50.10"), echo: true, echo_interval:
30},
  %{address: parse_ip.("10.100.50.11"), echo: true, echo_interval:
30},
]
heartbeat_interval_ms = 10_000
feature_ueip = true
ueip_pool = "10.45.0.0/16"
```

示例 3: Proxmox VM 与 SR-IOV (原生模式)

场景: 在 Proxmox VM 上使用 SR-IOV 直通的生产 UPF

```
# Proxmox SR-IOV 配置 (/etc/omniupf/runtime.exs)
xdp_interfaces = "ens1f0"           # SR-IOV VF
xdp_attach_mode = "native"
api_port = 8080
pfcp_address = "192.168.100.10"
pfcp_port = 8805
node_id = "192.168.100.10"
n3_address = "192.168.100.10"
log_level = :info
max_sessions = 100_000
gtp_echo_interval = 15
gtp_peers = [
  %{address: parse_ip.("192.168.100.50"), echo: true},
]
```

示例 4: PGW-U 模式 (4G EPC)

场景: OmniUPF 在 4G EPC 网络中充当 PGW-U

```
# PGW-U 配置 (/etc/omniupf/runtime.exs)
xdp_interfaces = "eth0"
xdp_attach_mode = "native"
api_port = 8080
pfcp_address = "10.200.1.10"
pfcp_port = 8805
node_id = "10.200.1.10"
n3_address = "10.200.1.10"          # S5/S8 接口 (GTP-U)
log_level = :info
max_sessions = 200_000
gtp_echo_interval = 20
gtp_peers = [
  %{address: parse_ip.("10.200.1.50"), echo: true, echo_interval:
20},
]
heartbeat_interval_ms = 5_000
```

示例 5: 多模式（UPF + PGW-U 同时）

场景: OmniUPF 同时为 5G 和 4G 网络提供服务

```
# 多模式配置 (/etc/omniupf/runtime.exs)
xdp_interfaces = "eth0,eth1"
xdp_attach_mode = "native"
api_port = 8080
pfcp_address = "10.50.1.100"
pfcp_port = 8805
node_id = "10.50.1.100"
n3_address = "10.50.1.100"
n9_address = "10.50.1.101"
log_level = :info
max_sessions = 300_000
gtp_echo_interval = 15
gtp_peers = [
  %{address: parse_ip.("10.50.2.10"), echo: true, echo_interval:
15},
  %{address: parse_ip.("10.50.2.20"), echo: true, echo_interval:
15},
]
heartbeat_interval_ms = 10_000
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

```
# SmartNIC 卸载配置 (/etc/omniupf/runtime.exs)
xdp_interfaces = "enp1s0np0"        # SmartNIC 接口
xdp_attach_mode = "offload"
api_port = 8080
pfcp_address = "10.10.1.50"
pfcp_port = 8805
node_id = "10.10.1.50"
n3_address = "10.10.1.50"
log_level = :warning
max_sessions = 1_000_000
far_map_size = 2_000_000
qer_map_size = 1_000_000
gtp_echo_interval = 30
gtp_peers = [
  %{address: parse_ip.("10.10.2.10"), echo: true},
  %{address: parse_ip.("10.10.2.20"), echo: true},
  %{address: parse_ip.("10.10.2.30"), echo: true},
]
heartbeat_interval_ms = 15_000
```

映射大小和容量规划

设置射大小

设置 `max_sessions` 并配置映射大小以适应您的部署：

```
max_sessions = 100_000
far_map_size = 131_070
qer_map_size = 65_535
urr_map_size = 131_070
```

内存使用：~91 MB 用于 100K 会话

容量估算

计算最大会话数:

```
最大会话数 = 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

最大会话数 = min(100,000, 100,000, 100,000) = **100,000**

内存要求

每会话内存使用:

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

对于 **100K** 会话: ~52 MB 内核内存

建议: 确保锁定内存限制允许 2 倍的预估使用:

```
# 检查当前限制
ulimit -l

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

相关文档

- 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
- □作指南 - UPF 架构和部署□述
- 围墙花园指南 - 超出信用重定向、捕获门户和白名单配置

用户平面上的 IPv6 / 双栈

OmniUPF 如何处理 IPv6 用户流量以支持 IPv6 和 IPv4v6 PDN 会话：内核前提条件、UE 地址池、从 PFCP 编程 IPv6 下行流量，以及通过 OSPFv3 向网络广告 UE 主机路由。

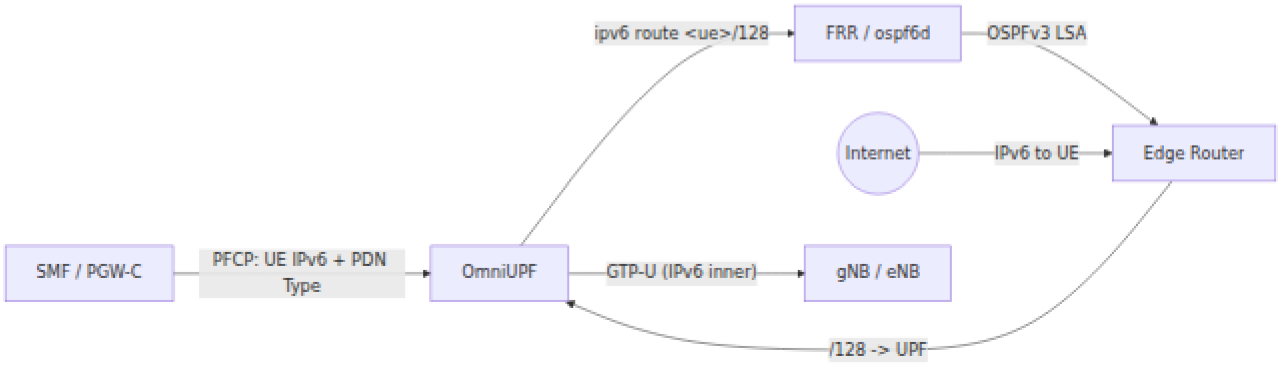
本页涵盖 用户平面 的职责。控制平面（PDN 类型协商、PAA、Gx，以及通过 PFCP 向 UPF 信令 UE IPv6 地址）由 SMF/PGW-C 处理。另请参见 路由管理 和 配置指南。

目录

概述

当建立 IPv6（或 IPv4v6）PDN 会话时，SMF/PGW-C 通过 PFCP 将 UE 的 IPv6 地址和 PDN 类型发送给 UPF。然后，UPF：

1. 编程 **IPv6** 下行查找，以便将发往 UE 的数据包进行匹配和隧道化。
2. 为 UE 地址添加 `/128` 主机路由，并通过 **OSPFv3** 将其广告给边缘路由器，以便下行流量被引导到该 UPF。
3. 在两个方向上转发用户流量（上行解封装，下行 GTP-U 封装）以处理 IPv6 内部数据包。



内核前提条件

必须在主机级别启用 IPv6 并开启转发。通过 sysctl drop-in 应用（例如，在 `/etc/sysctl.d/` 下）。

设置	必需值	描述
`net.ipv6.conf.all.disable_ipv6`	`0`	必须启用 IPv6。UPF 数据平面接口需要一个全局 IPv6 地址用于传输和路由。
`net.ipv6.conf.default.disable_ipv6`	`0`	确保新出现的接口上 IPv6 保持启用。
`net.ipv6.conf.all.forwarding`	`1`	启用 IPv6 转发，以便 UE 上行流量可以出口，下行流量被路由到数据平面。

数据平面接口还需要在传输段（朝向边缘路由器）上拥有一个全局 IPv6 地址，以便 OSPFv3 形成邻接并交换路由。

UE IPv6 地址池

当 UPF 分配 UE 地址（而不是从 SMF 接收时），在 `runtime.exs` 中设置一个 IPv6 池。当 SMF/PGW-C 通过 PFCP 提供 UE 地址时，该地址将直接使用，而此池不会被查询。

```
# /etc/omniupf/runtime.exs
config :upf_ex,
  ueip_pool: "2001:db8:6f85:70::/64"
```

参数	类型	必需	默认	描述
`ueip_pool`	字符串（CIDR）	否	-	当 UPF 执行地址分配时使用的 UE IP 池。接受 IPv4 或 IPv6 CIDR；地址族从字符串中检测。

OSPFv3 路由广告

每个活动的 UE IPv6 地址作为 `/128` 主机路由安装并广告给边缘路由器，镜像 IPv4 `/32` 的行为。路由由路由管理器管理，并通过 FRR 重新分发到 OSPF。

```
# /etc/omniupf/runtime.exs
config :upf_ex,
  route_manager_enabled: true,
  route_manager_type: "frr"
```

参数	类型	必需	默认	描述
`route_manager_enabled`	布尔值	否	`false`	启用 UE 主机路由管理。必须为 `true` 才能广告 UE 路由。
`route_manager_type`	字符串	否	-	路由后端。`"frr"` 通过 FRR/`vtysh` 注入路由；静态后端直接使用内核 `ip`/`ipv6` 路由命令。使用 `"frr"` 进行 OSPFv3 广告。

FRR 实例必须在传输接口上运行 **OSPFv3**（`ospf6d`）并重新分发静态路由，以便 UE `/128` 主机路由被洪泛到边缘路由器。IPv6 主机路由通过 `ipv6 route <addr>/128` 命令系列添加 - IPv4 `ip route` 命令不接受 IPv6 前缀。有关路由管理器生命周期的更多信息，请参见 路由管理。

数据平面行为

方向	行为
下行（网络 → UE）	内部 IPv6 目标与从 PFCP 会话编程的 IPv6 下行表进行匹配；匹配的数据包被 GTP-U 封装到 gNB/eNB（如果规则的动作是 BUFF，则缓冲）。
上行（UE → 网络）	GTP-U 被解封装，内部 IPv6 数据包通过内核 IPv6 FIB 转发到 N6 接口（因此需要转发 sysctl）。
缓冲（BUFF）	缓冲的 IPv6 下行数据包被交付给用户空间缓冲监听器，并在转发时刷新，行为与 IPv4 相同。

当前构建不需要 `accept_local` / 反向路径过滤放宽；缓冲路径使用非本地外部源，因此数据包不会被丢弃为 martians。

故障排除

IPv6 下行流量未到达 UE

症状：会话已建立且上行正常，但来自网络到 UE 的流量被丢弃。

可能原因：

- UE `/128` 主机路由未被广告（路由管理器禁用，或 UPF 上未运行 OSPFv3）。
- UPF 与边缘路由器之间的 OSPFv3 邻接未建立。
- 边缘路由器尚未安装 `/128`（收敛延迟）。

解决方案：

1. 确认 `route_manager_enabled: true` 和 `route_manager_type: "frr"`。
2. 验证 UPF FRR 实例在传输接口上运行 OSPFv3 并重新分发静态路由。
3. 确认 OSPFv3 邻接为完全状态，边缘路由器显示通过 OSPF 学习的 UE `/128`。

IPv6 用户流量完全未转发

症状：无论是上行还是下行 IPv6 都无法工作，即使存在路由。

可能原因：

- 主机上禁用了 IPv6，或转发关闭。
- 数据平面接口没有全局 IPv6 地址。

解决方案：

1. 验证 内核前提条件（`disable_ipv6=0`，`forwarding=1`）。
2. 确认数据平面接口在传输段上具有全局 IPv6 地址。

指标参考

本文档描述了 OmniUPF 在 `/api/metrics` 端点上暴露的所有 Prometheus 指标（HTTP API 运行在 `api_port` 上，默认值为 `8080`）。请注意 `/api` 前缀 - 每个 API 路由都在其下提供服务。

下面的指标名称、类型和标签反映了 OmniUPF 暴露的指标。

指标类别

1. **PFCP** 指标 - 控制平面消息、关联、会话和错误计数器
2. 会话生命周期指标 - 会话创建/删除/修改计数器和建立延迟
3. 用户空间慢路径和缓冲指标 - 下行缓冲、围墙花园和未知 TEID 错误指示的丢包计数器
4. 下行数据报告 **(DLDR)** 指标 - PFCP 会话报告请求通知和 FAR 索引跟踪
5. **GTP-U** 指标 - 回声响应和错误指示计数器
6. **URR** 指标 - 使用报告计数器和活动 URR 定时器
7. **eBPF** 映射指标 - 数据平面映射条目计量
8. 数据平面吞吐量 - 每个方向 (N3/N6) `_total` 计数器，由 `expose_xdp_metrics` 控制 (#57)；也可以通过 `/api/v1/xdp_stats` 按需获取
9. 资源/路由指标 - TEID/FAR-ID 池和 UE 路由计量
10. 围墙花园指标 - 捕获门户拦截、转发和 DNS 欺骗计数器

PFCP 指标

UPF 与其 PFCP 对等方 (SMF / PGW-C / SGW-C) 之间的控制平面协议计数器。

指标名称	类型	标签	描
`upf_pfcp_associations`	Gauge	none	当前活动的 PFCP 关联[
`upf_pfcp_sessions`	Gauge	none	当前活动的 PFCP 会话[
`upf_pfcp_peer_restarts_total`	Counter	`peer`	对等方重启检测（恢复时间[
`upf_pfcp_duplicate_requests_total`	Counter	none	检测到并抑制的重复 PFC
`upf_orphaned_sessions_deleted_total`	Counter	none	由于对等方重启而删除的会[
`upf_pfcp_messages_rx_total`	Counter	`type`, `peer`	按消息类型和对等方接收的
`upf_pfcp_messages_tx_total`	Counter	`type`, `peer`	按消息类型和对等方发送的
`upf_pfcp_errors_total`	Counter	`type`	PFCP 请求拒绝，按拒[`session_conte` `mandatory_ie_`

会话生命周期指标

指标名称	类型	标签
`upf_sessions_created_total`	Counter	none
`upf_sessions_deleted_total`	Counter	none
`upf_sessions_modified_total`	Counter	none
`upf_session_establishment_duration_microseconds`	Histogram	none

用户空间慢路径和缓冲指标

eBPF 数据平面将数据包转发到用户空间慢路径（UDP 端口 `22152`，默认）以执行三项不���的工作：下��缓冲 当 UE 空闲时（在 UE 被寻呼之前保持），围墙花园重定向，以及回答 未知 **TEID** 上行 G-PDU 的 GTP-U 错误指示。下面的

`buffered`/`flushed`/`current` □□□□□□□□□□□□□□□□□□□□□□□□ `upf_walled_garden_*` □□□□□□ TEID □□□ `upf_gtpu_error_indications_sent_total`□

□□□□□	□□	□□	□□
`upf_userspace_punt_received_total`	Counter	none	□□□□□□□□□□□□□□□□□□□□□ + □□□□ + □□ TEID□
`upf_buffer_packets_buffered_total`	Counter	none	□□□□□□□□□□□□□□
`upf_buffer_packets_flushed_total`	Counter	none	□□□□□□□□□□□□□□
`upf_buffer_packets_dropped_total`	Counter	`reason`	□□□□□□□□/□□□□□□□□
`upf_buffer_packets_current`	Gauge	none	□□□□□□□□□□□□□□
`upf_buffer_flush_operations_total`	Counter	none	□□□□□□□□□□□□□□ FAR □□□
`upf_buffer_flush_packets_sent_total`	Counter	none	□□□□□□□□□□□□□□
`upf_buffer_flush_errors_total`	Counter	`reason`	□□□□□□□□□□□□□□□

`upf_buffer_packets_dropped_total` **`reason`** □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- `global_limit` - □□□□□□□□□ (`buffer_max_total`)
- `far_limit` - □□□□ FAR □□□□□□□ (`buffer_max_per_far`)
- `expired` - □□□□□ TTL (`buffer_ttl_ms`) □□

- `read_error` - □□□□□□□□□□□□□□
- `too_small` - □□□□□□□□□□□ GTP □
- `invalid_gtp_type` - □ G-PDU GTP □□□□
- `not_buffering_far` - FAR □□ BUFF □□
- `error_indication_failed` - □□□□□□ TEID □□□ GTP-U □□□□

> □□ TEID □□ □□ □□□□□□□□□ GTP-U □□□□□□□□□□□
> `upf_gtpu_error_indications_sent_total` □□□□□□□□□□□□□□□□□□□□□□□□□□

`upf_buffer_flush_errors_total` **`reason`** □□

- `far_lookup_failed` - □□□ eBPF □□□□□ FAR □□
- `no_forw_action` - FAR □□□□ FORW □□
- `connection_failed` - □□□□  □□□ UDP □□□

□□□□□□ (DLDR) □□

□□□□□□□□□□□□□□□□□□□□ PFCP □□□□□□□□□□□□□□□□□□□ SMF/PGW-C □□ UE□

□□□□	□□	□□	□□
`upf_dldr_sent_total`	Counter	`peer`	□□□□□□□□□□□ SMF/PGW-C □□ □□
`upf_dldr_errors_total`	Counter	none	DLDR □□/□□□
`upf_dldr_skipped_total`	Counter	none	□□ FAR □□□□□□□□□ DLDR
`upf_far_index_size`	Gauge	none	□ DLDR □□□□□□ FAR □□

GTP-U □□

GTP-U □□□□□□□□□□□□□□□□□□□□□□□ TEID □□□□□□□□□□□□□□□□□□□□□□□

指标名称	类型	标签	描述
`upf_gtpu_echo_responses_tx_total`	Counter	none	发送的 GTP-U 回声响应
`upf_gtpu_error_indications_sent_total`	Counter	none	发送的 GTP-U 错误指示（针对未知的入站 TEID）
`upf_gtpu_error_indications_rx_total`	Counter	none	从对等方接收到的 GTP-U 错误指示
`upf_error_indication_sessions_deleted_total`	Counter	none	由于接收到错误指示而删除的会话

发送错误指示的情况： UPF 接收到一个 TEID 不存在的 GTP-U 数据包（例如，在重启后，或者会话已经被删除），并通知发送方停止。

接收错误指示的情况：下游对等方无法识别 UPF 转发的 TEID; UPF 删除受影响的会话以停止转发到一个死隧道。

URR (□□□□□□□) □□

□□□□□	□□	□□	□□
`upf_urr_reports_sent_total`	Counter	`trigger`	□□□ URR □□□□□□□□□ □□□□
`upf_urr_report_errors_total`	Counter	none	URR □□□□/□□□□
`upf_urr_active_timers`	Gauge	none	□□□ URR □□□□□□□□/□ □□□/□□□□□□

`trigger` □□□□ URR □□□□□□ - □□ `periodic` (PERIO), `volume_threshold` (VOLTH), `time_threshold` (TIMTH), `quota_validity` (QUVTI)□

□□ URR □□□□□□□□□□ Prometheus □□□□□□□□□□□□□□□ REST API □ `/api/v1/urr_map` □□□□ URR □□□□□

---

eBPF □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□	□□	□□	□□
`upf_ebpf_far_entries`	Gauge	none	eBPF □□□□ FAR □□
`upf_ebpf_pdr_uplink_entries`	Gauge	none	eBPF □□□□□□ PDR □□ □□□ TEID □ PDR□
`upf_ebpf_pdr_downlink_entries`	Gauge	none	eBPF □□□□□□ PDR □□ □□ UE IP □□□ PDR□

`upf_ebpf_far_entries` □□□□□□ FAR-id □ (`upf_far_ids_allocated`)□□□ far_map □□□□□ FAR id □□□□□□□PDR □□□□□□□□□□□□□□□□□□□□

数据平面吞吐量（N3/N6）

用户平面流量的每个方向数据包和字节计数器（**N3** = GTP-U 侧朝向 gNB/eNB；**N6** = SGi 侧朝向数据网络）来自 eBPF `upf_ext_stat` 每个 CPU 映射。

指标名称	类型	标签	描述
`upf_n3_rx_packets_total` / `upf_n3_tx_packets_total`	Counter	none	N3 (GTP-U) 数据包进出 XDP 数据路径
`upf_n6_rx_packets_total` / `upf_n6_tx_packets_total`	Counter	none	N6 (SGi) 数据包进出 XDP 数据路径
`upf_n3_rx_bytes_total` / `upf_n3_tx_bytes_total`	Counter	none	N3 (GTP-U) 字节进出
`upf_n6_rx_bytes_total` / `upf_n6_tx_bytes_total`	Counter	none	N6 (SGi) 字节进出

由 **`expose_xdp_metrics`** 控制（问题 **#57**）。在请求路径上读取 `upf_ext_stat`（每次 Prometheus 抓取）被证明会触发 `virtio_net` 通用 XDP **TX-ring wedge**。有两个缓解措施：读取在请求路径之外，由后台 `Sampler` 以固定间隔执行，从不在抓取时执行，并且它由 `config :upf_ex, expose_xdp_metrics: true` 控制。当禁用时，这些计数器不会被声明或导出。

这些是累积的 `_total` 计数器。底层的 `upf_ext_stat` 映射是固定的，因此它在 omniupf 重启时存活，但在 eBPF `.o` 重新部署时被清零；收集器将其绝对值转换为单调计数器增量，因此 `rate()` 在两者之间保持正确。不要将这些视为计量。使用 `rate()`。

按需 **JSON**：`GET /api/v1/xdp_stats` 返回相同的字段，并且仅在您明确调用时触及 eBPF。字段：`rx_n3` / `tx_n3` / `rx_n6` / `tx_n6`（数据包）和 `rx_n3_bytes` / `tx_n3_bytes` / `rx_n6_bytes` / `tx_n6_bytes`。

资源/路由指标

指标名称	类型	标签	描述
`upf_routes_total`	Gauge	none	当前由路由管理器跟踪的 UE IP 路由
`upf_teid_allocated`	Gauge	none	当前从池中分配的 TEID
`upf_far_ids_allocated`	Gauge	none	当前从池中分配的 FAR ID

围墙花园指标

用于超出信用/捕获门户围墙花园的计数器。有关行为，请参见 围墙花园指南。

指标名称	类型	标签	描述
`upf_walled_garden_active_redirects`	Gauge	none	活动的围墙花园重定向会话
`upf_walled_garden_packets_intercepted_total`	Counter	none	被围墙花园拦截的数据包
`upf_walled_garden_packets_dropped_total`	Counter	none	被围墙花园丢弃的数据包
`upf_walled_garden_packets_forwarded_total`	Counter	`dst_ip`	按目标 IP 转发的白名单数据包
`upf_walled_garden_bytes_forwarded_total`	Counter	`dst_ip`	按目标 IP 转发的白名单字节
`upf_walled_garden_dns_spoofed_total`	Counter	`domain`	按域名伪造到门户的 DNS 查询
`upf_walled_garden_dns_forwarded_total`	Counter	`domain`	按域名转发的

□□□□	□□	□□	□□
			DNS □□ □□□ □□ □□

> □□□□□ `dst_ip` □ `domain` □□□□□

Grafana □□□

□□□□□□□□□□□□□□□□

- □□□□□□□□ (`upf_pfcp_sessions`, `upf_pfcp_associations`)
- □□□□□□ N3/N6 □□□□□□□ `/api/v1/xdp_stats` □□□□□□□□□□□□□□□ #57□
- □□□□□ (`upf_buffer_packets_current`, □ `reason` □□□□)
- DLDR / □□□□ (`rate(upf_dldr_sent_total[5m])`)
- □□□□□□□□□□□□□□□□□□□□

□□□□

- □□□□ - □□□□□□□□□□□□□□
- □□□□ - □□ `api_port`□□□□□□□□ UPF □□
- **Web UI □□** - □□□□□□□□□□
- □□□□ - eBPF □□□□□□□□□□
- □□□□□□□ - □□ PDR□FAR□QER□URR □□
- □□□□□□□ - □□□□□□□□□□□□□□□
- □□□□□□□ - □□□□□□□□□

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 提供全面的实时指标。

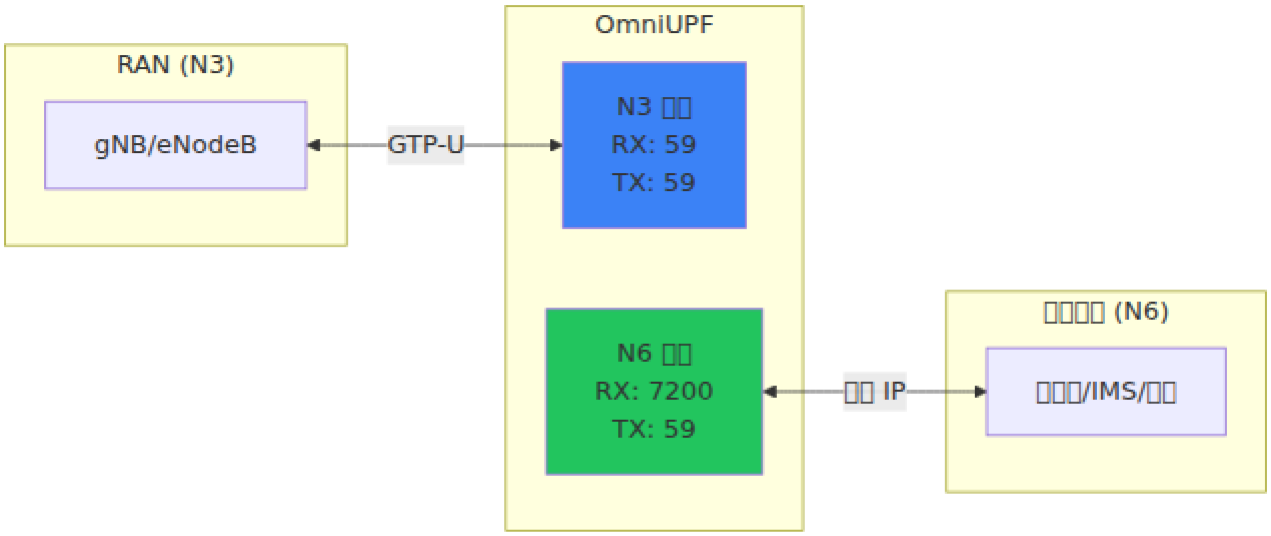
监控类别

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、终止
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	跟踪移动期间的数据包缓冲	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 接口统计提供 RAN (N3) 和数据网络 (N6) 之间流量分布的可见性。



指标：

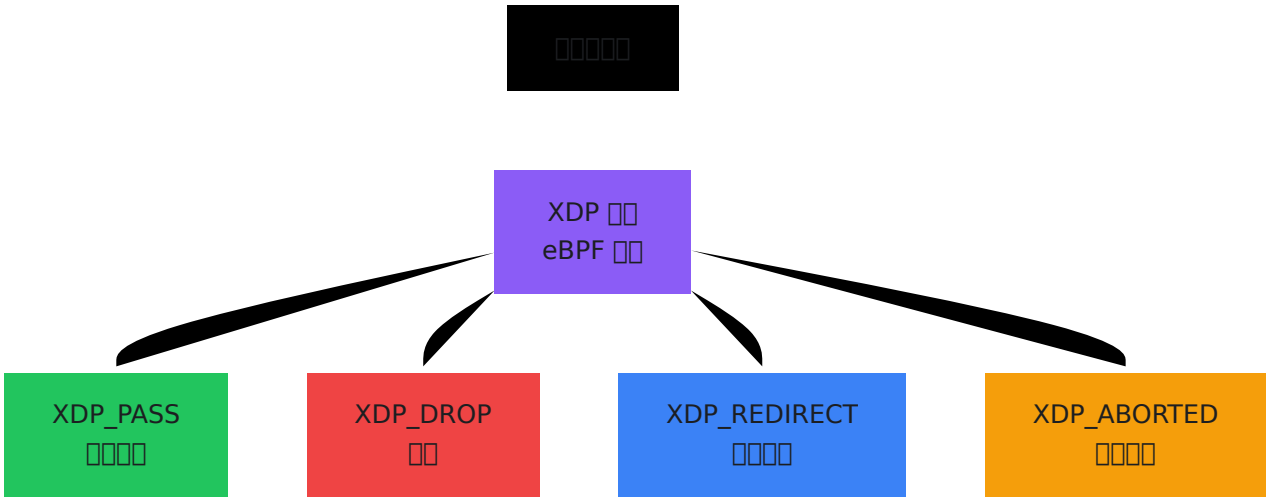
- **RX N3**：来自 RAN 的接收数据包（上行 GTP-U 流量）
- **TX N3**：发送到 RAN 的数据包（下行 GTP-U 流量）
- **RX N6**：来自数据网络的接收数据包（下行原生 IP）
- **TX N6**：发送到数据网络的数据包（上行原生 IP）
- 总计：所有接口的聚合数据包计数

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��错误

XDP □□

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



□□□

- □□□XDP □□□□□□□□□□□□ 0□

- 丢弃：XDP 程序故意丢弃的数据包
- 通过：传递给网络栈以进行进一步处理的数据包
- 重定向：直接重定向到输出接口的数据包
- **TX**：通过 XDP 发送的数据包

解释：

- 终止 **> 0**：eBPF 程序或内核兼容性存在严重问题
- 丢弃 **> 0**：基于策略的丢弃或无效数据包
- 通过高：大多数数据包在网络栈中处理（正常）
- 重定向高：数据包直接转发（最佳性能）

数据包统计

详细的数据包协议分解和处理计数器。

协议计数器：

- **RX ARP**：地址解析协议数据包
- **RX GTP ECHO**：GTP-U 回声请求/响应（保持连接）
- **RX GTP OTHER**：其他 GTP 控制消息
- **RX GTP PDU**：GTP-U 封装的用户数据（主要流量）
- **RX GTP UNEXP**：意外的 GTP 数据包类型
- **RX ICMP**：互联网控制消息协议（ping，错误）
- **RX ICMP6**：ICMPv6 数据包
- **RX IP4**：IPv4 数据包
- **RX IP6**：IPv6 数据包
- **RX OTHER**：其他协议
- **RX TCP**：传输控制协议数据包
- **RX UDP**：用户数据报协议数据包

用例：

- 监控 **GTP-U PDU** 计数：主要用户流量指标
- 检查 **ICMP** 以进行连接性：网络可达性测试

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

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）
- 成功：成功的路由查找

IPv6 FIB 查找：

- 缓存：缓存的 IPv6 路由查找
- 成功：成功的 IPv6 路由查找

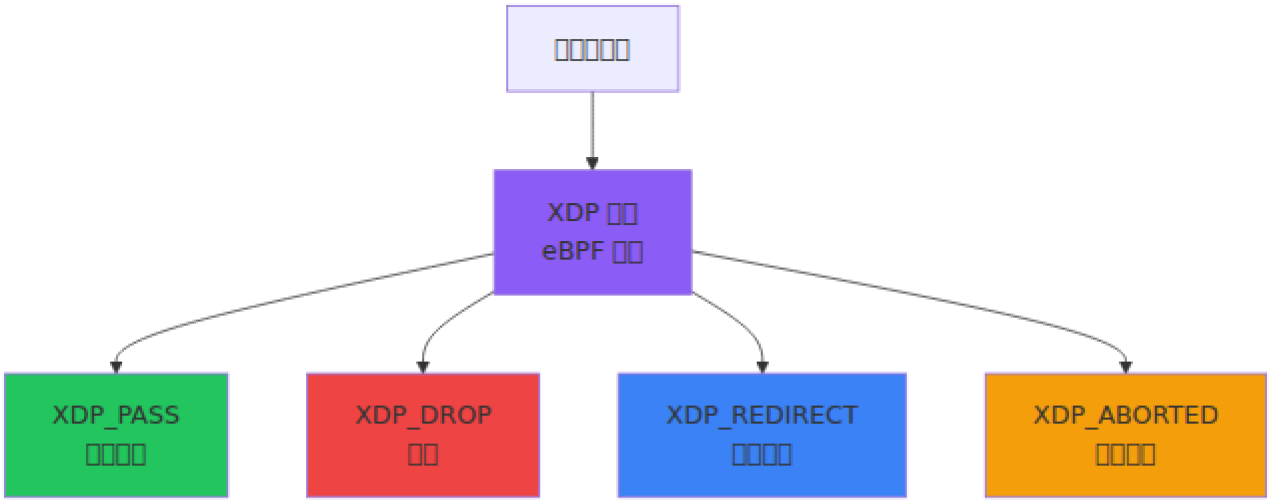
性能指标：

- 高缓存命中率：表明良好的路由缓存性能
- 高成功计数：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□□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% (琥珀色)	警告 - 在 1 周内安排增加容量
90-100% (红色)	危急 - 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，请参见 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 终止计数增加
- 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

每个方向的吞吐量在启用 `expose_xdp_metrics` 时作为 `_total` 计数器导出（`upf_n3_rx_packets_total`、`upf_n3_rx_bytes_total`，… N6 等效项）。N3 是 GTP-U 侧（gNB），N6 是 SGi 侧（数据网络）。eBPF `upf_ext_stat` 的读取是通过后台 `Sampler` 的抓取路径完成的（每次抓取读取是经过验证的 `virtio_net` TX-ring-wedge 触发器，问题 #57），并受到标志的限制。相同字段也可以通过 JSON 在 `/api/v1/xdp_stats` 上按需获取。有关详细信息，请参见 指标参考。

容量规划：

- 监控峰值吞吐量时间（例如，晚上时段）
- 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 (%) = (丢弃的数据包 / 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%**：危急（重大转发或容量问题）

常见丢包原因：

- QER 速率限制（MBR 超出）
- 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 终止计数 > 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%

- 缓冲数据包计数增加
- 新的 PFCP 关联建立/释放
- URR 流量阈值超出

警报配置

警报可以通过以下方式配置：

1. **Prometheus** 指标：导出指标以供外部监控（有关完整列表，请参见 指标参考）
2. 日志监控：解析 OmniUPF 日志以查找错误模式
3. **REST API** 轮询：定期查询 `/map_info`、`/packet_stats` 端点
4. **Web UI** 监控：通过统计和容量页面进行手动监控

容量规划

会话容量估算

计算最大会话数：

```
最大会话数 = min(
  PDR 映射容量 / 2,  # 下行 + 上行 PDR 每会话
  FAR 映射容量 / 2,  # 下行 + 上行 FAR 每会话
  QER 映射容量       # 可选，每会话一个 QER
)
```

示例：

- PDR 映射容量：131,070
- FAR 映射容量：131,070
- QER 映射容量：65,535

最大会话数 = min(131,070 / 2, 131,070 / 2, 65,535) = **65,535** 会话

内存容量

计算总 **eBPF** 映射内存：

```
内存 = Σ (映射容量 × (键大小 + 值大小))
```

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 倍估算使用量
- 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. 增加余量：

```
所需容量 = 总吞吐量 × 2  # 100% 余量
```

示例：

- 10,000 个并发会话
- 每个会话平均 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：

```
达到容量的周数 = (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
- 达到容量的周数：(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某种协议
3. 查看 XDP 统计以获取 XDP 丢包与终止的对比

常见原因：

- **QER** 速率限制：检查 QER MBR 值与实际流量
- 无效的 **TEID**：验证上行 PDR TEID 是否与 gNB 分配匹配
- 未知的 **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 终止 > 0

诊断：

1. 导航到统计 → XDP 统计
2. 检查终止计数
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
- 检查内核版本是否符合最低要求（Linux 5.4+）
- 查看 eBPF 程序日志
- 如果问题持续，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）

立即缓解措施:

1. 确定过期会话（检查会话页���）
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. 实施会话清理策略

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
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：

- 水平扩展 UPF（添加更多实例）
- 升级 CPU 或启用 RSS（接收侧平衡）
- 升级网络接口以提高速度
- 调整 eBPF 映射哈希函数

相关文档

- 指标参考 - 完整的 Prometheus 指标参考
- **UPF** 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** 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

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 中实现的原因代码以及它们在 PFCP 消息处理过程中发生的时机。

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：

- 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。所有强制信息元素存在且有效。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**	一般拒绝，原因不明。当没有适用的特定原因代码时使用。
65	**SessionContextNotFound**	请求修改或删除未知 SEID 的会话。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：关联设置中缺少 NodeID，会话建立中缺少 F-SEID，缺少 RecoveryTimeStamp。
67	**ConditionalIEMissing**	根据其他信息元素的存在，缺少条件必需的信息元素。当信息元素相互依赖时使用。
69	**MandatoryIEIncorrect**	必需的信息元素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。

□□□/□□□□□□

□□	□□	□□□□
74	**PFCPEntityInCongestion**	UPF □□□□□□□□□□□□□□□□□□□□□□□□□□
75	**NoResourcesAvailable**	□□□□□□□□□□□□□□□□eBPF □□□□□□□□□□□□□□□□ TEID □□□□
77	**SystemFailure**	□□□□□□□□□□□□□□□□□□□□eBPF □□□□□□□□□□□□□□□□ □□□□□

□□□□□□□□

□□	□□	□□□□
68	**InvalidLength**	□□□□□□□□□□□□□□□□□□□□□□□□□□□□ OmniUPF □□□□□
70	**InvalidForwardingPolicy**	UPF □□□□□□□□□□□□□□□ OmniUPF □□□□□
71	**InvalidFTEIDAllocationOption**	□□□□ F-TEID □□□□□□□□□ OmniUPF □□ □□□
76	**ServiceNotSupported**	□□□□□□□□□□□□□□□□□ OmniUPF □□□□□
78	**RedirectionRequested**	UPF □□□□□□□□□□ UPF □□□□□□□ OmniUPF □□□□□

□□□□□□□

□□□□□□

□□□□□ NodeID

```
SMF → UPF: 关联设置请求（无 NodeID）
UPF → SMF: 关联设置响应（原因：MandatoryIEMissing）
```

解决方案：确保 SMF 在所有关联设置请求中包含 NodeID 信息元素。

场景：无效的 **NodeID** 格式

```
SMF → UPF: 关联设置请求（NodeID="invalid"）
UPF → SMF: 关联设置响应（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: 关联设置请求（无 RecoveryTimeStamp）
UPF → SMF: 关联设置响应（原因：MandatoryIEMissing）
```

解决方案：在关联设置请求中包含 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（请参见 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 循环回路场景）

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])

# 顶部拒绝原因
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** 界面

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 配置中所需的信息元素
2. 验证 PFCP 库版本兼容性
3. 查看 SMF 日志以查找信息元素构造错误

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 > 70%，则在配置中增加映射大小
4. 请参见 容量规划

NoEstablishedPFCPAssociation 错误：

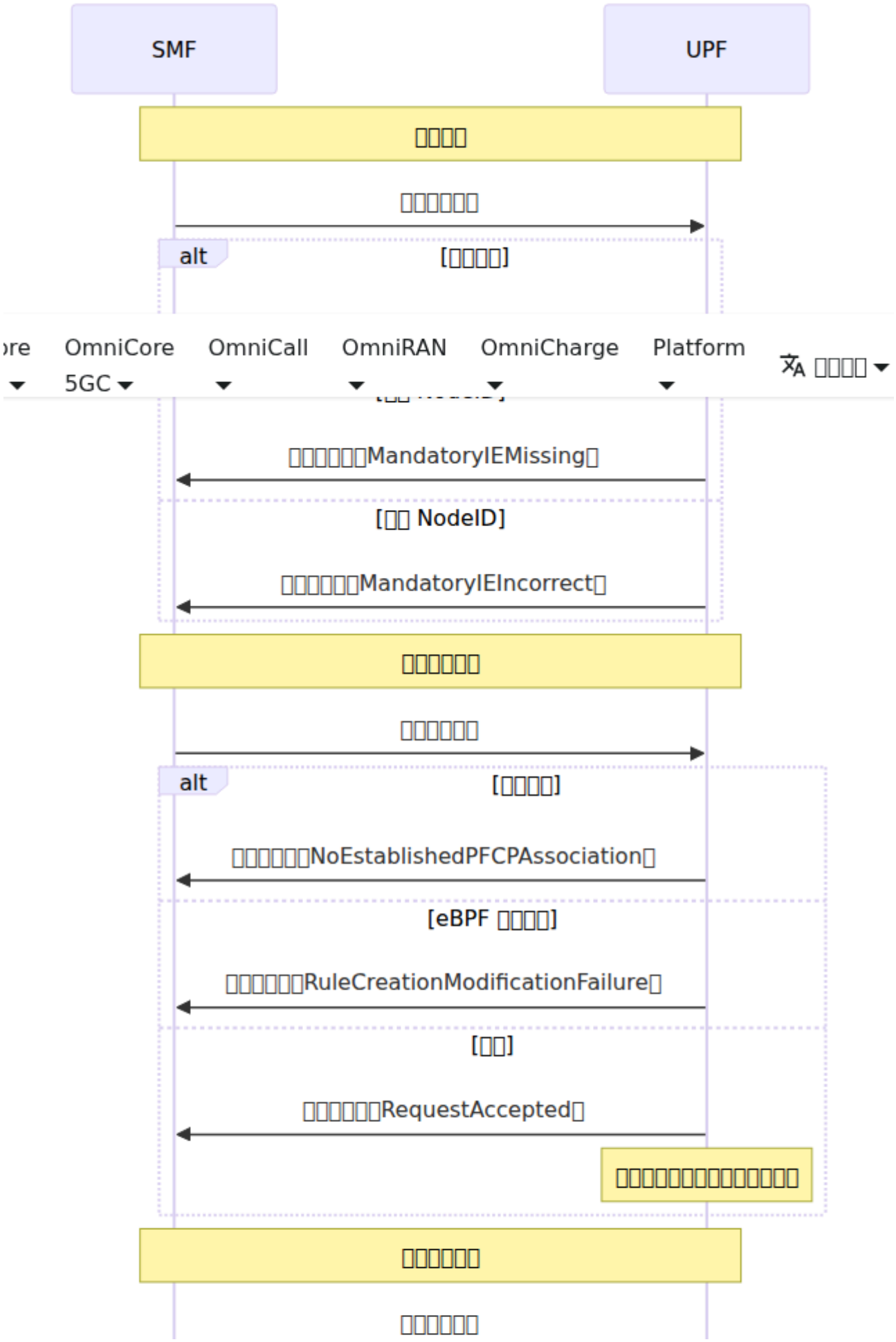
1. 验证关联是否存在： `GET /api/v1/pfcp_associations`
2. 检查心跳超时配置
3. 查看关联设置日志
4. 确保 SMF 和 UPF 可以相互访问

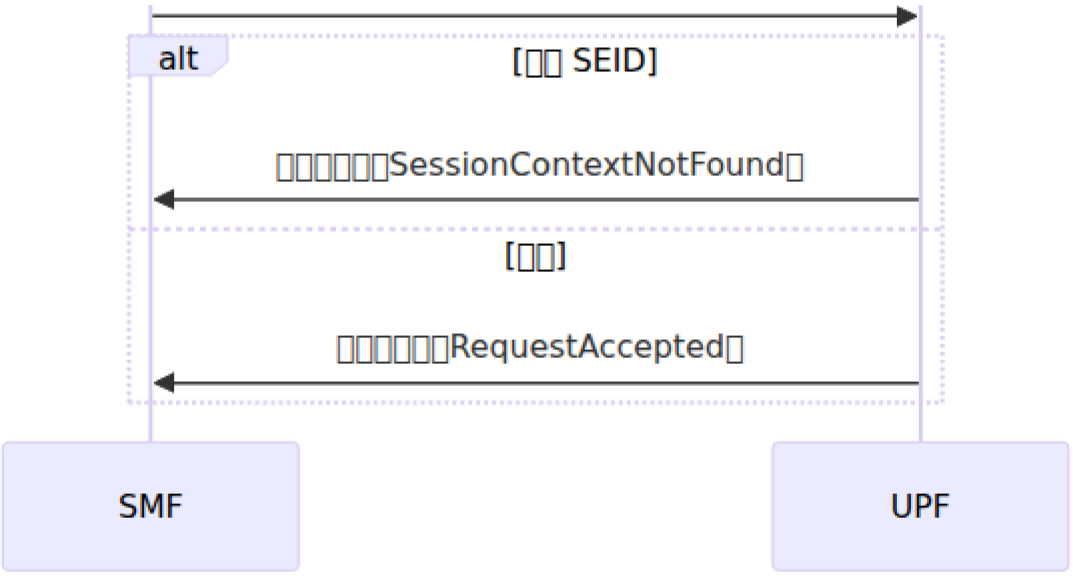
会话修改时的 **SessionContextNotFound**：

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 循环回路：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□作的影响

会话生命周期


SMF
UPF
□□□□
□□□□□□□
alt
[□□□□]
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□□□
□□□□□□□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 - 原因信息元素定义
- 第 **8.19** 节 - 原因值表

相关文档

- **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

规则管理指南

目录

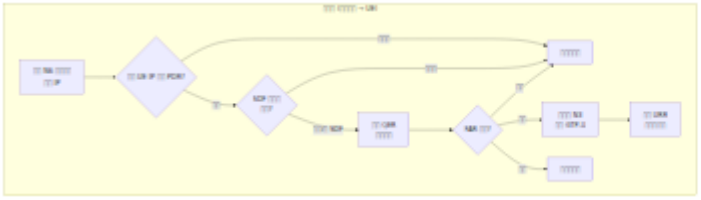
概述

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

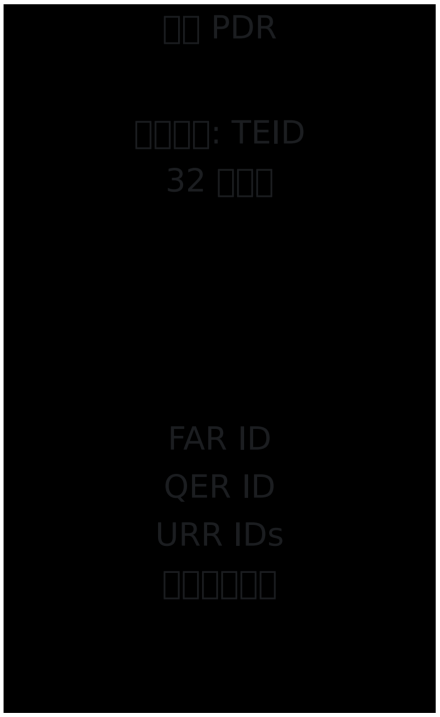


数据包检测规则（PDR）

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. 从数据包头部提取目标 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 过滤器提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

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：

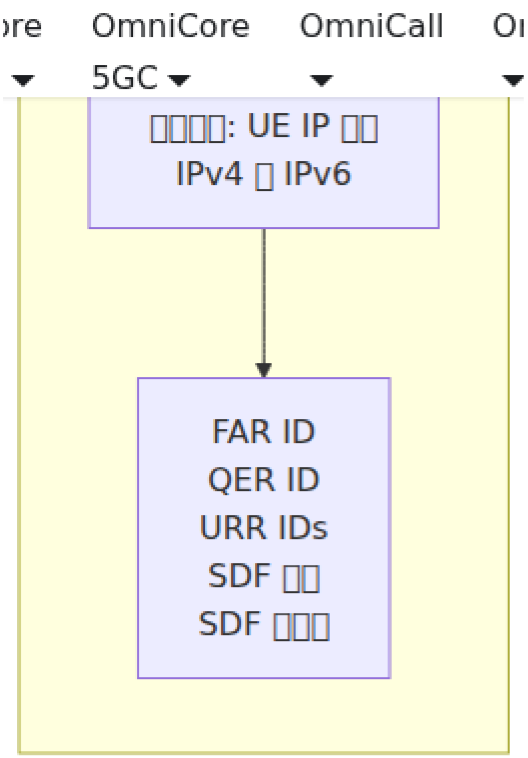
```
PDR ID: 10
UE IP: 10.45.0.1
SDF 模式: SDF Only
SDF 过滤器:
  - 协议: UDP, 端口: 5060-5061 → FAR ID 5 (VoIP FAR)
  - 协议: TCP, 端口: 443 → FAR ID 1 (默认 FAR)
```

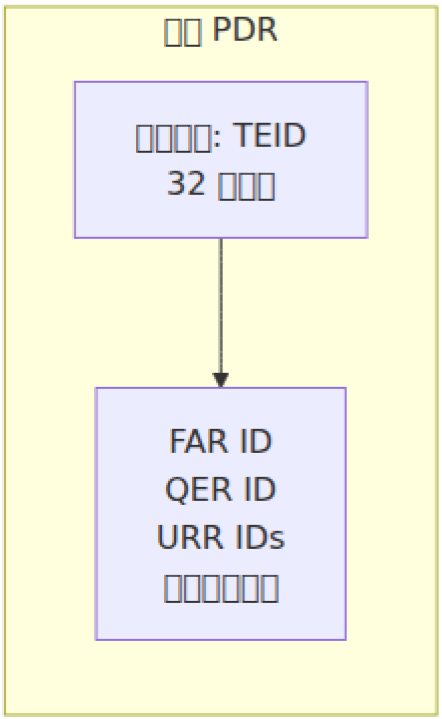
转发动作规则（**FAR**）

目的

FAR 确定对匹配 PDR 的数据包执行什么操作。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	□□□□□□□□□□□

□□□□□□:

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

□□□

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

□□□□□□□

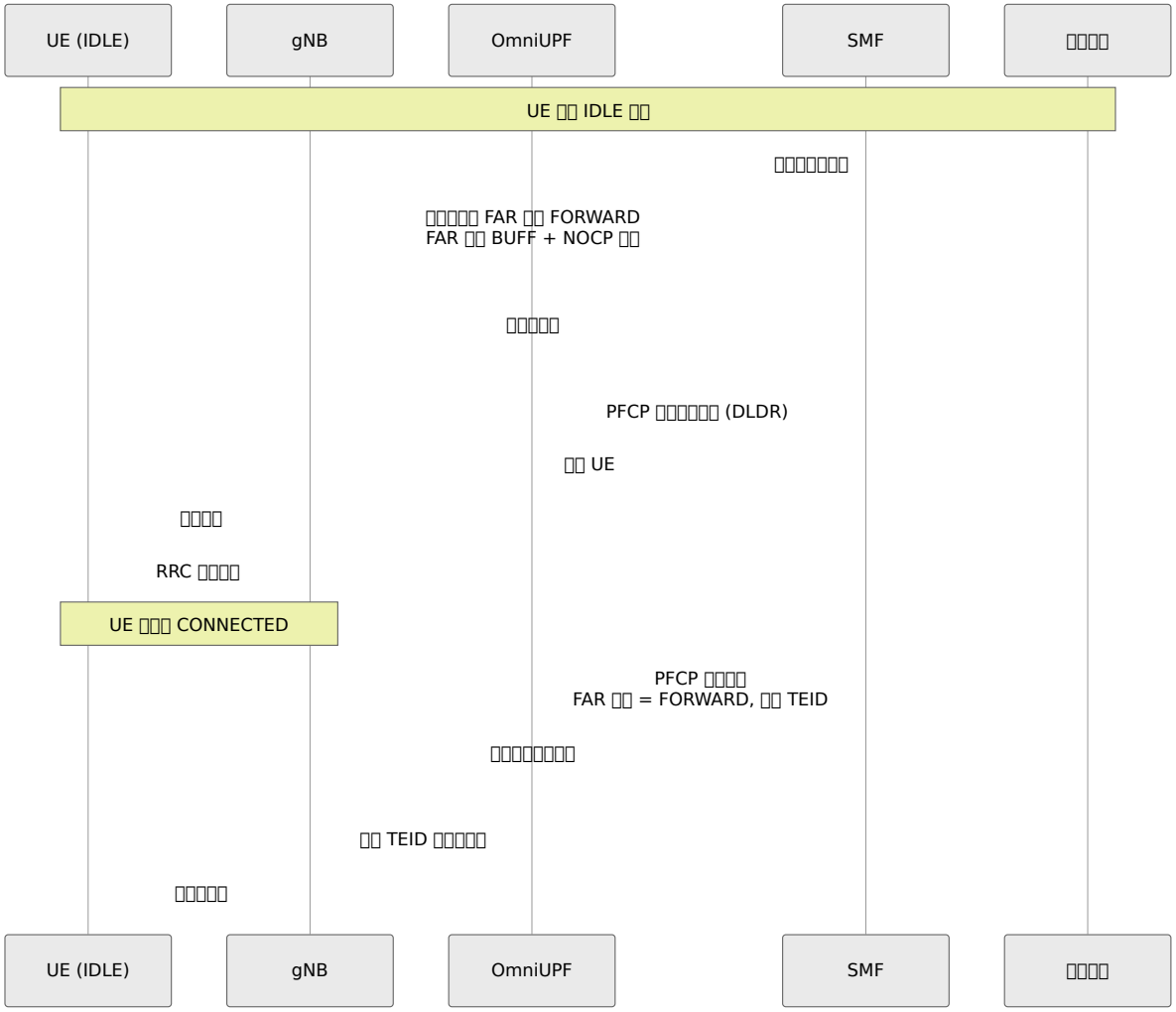
□□□□□□□□: □□□□□□□□□□ IDLE □□□□□□□ gNB□□ UE □□UPF□

1. □□□□□
2. □ SMF □□□□□□□□ (DLDR)
3. SMF □ 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
- 指标跟踪丢弃情况，`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 IDLE UE 缓冲数据包时，它向 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 QFI，寻呼策略指示器

SMF 对 **DLDR** 的□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`：发送的总 DLDR
- `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 获取完整列表。

缓冲□作

启用缓冲（设置 BUFF 标志）：

- FAR 动作 `|= 0x04`（设置位 2）

- 示例：`Action: 2 (FORW)` → `Action: 6 (FORW+BUFF)`
- 用于切换准备期间

仅缓冲模式（BUFF 无 FORW）：

- FAR 动作 `= 0x04`（仅 BUFF）
- 数据包被缓冲但不转发
- 用于 IDLE UE 状态（待寻呼）

禁用缓冲（清除 BUFF 标志）：

- FAR 动作 `&= ~0x04`（清除位 2）
- 示例：`Action: 6 (FORW+BUFF)` → `Action: 2 (FORW)`
- 缓冲的数据包在刷新或清除之前保持不变

刷新缓冲：

- 使用当前 FAR 规则重放所有缓冲的数据包
- 数据包将使用更新的 TEID/目标进行转发
- 成功刷新后缓冲被清空
- FAR 必须设置 FORW 动作

清除缓冲：

- 丢弃所有缓冲的数据包而不转发
- 在切换失败或会话被删除时使用
- 指标跟踪 `reason="cleared"`

监控缓冲的数据包

缓冲页面（Web UI）：导航到 缓冲 查看：

- 总缓冲数据包
- 总缓冲字节
- 具有缓冲数据包的 FAR 数量
- 每个 FAR 的数据包计数
- 最旧数据包的时间戳
- 启用/禁用每个 FAR 的缓冲

- □□□□□□□□

□□□□:

- □□□ **> 10** □: □□□□□□□□
- □□□ **> 30** □: □□□□□□□□□□□□□□
- □□□□□□□: □□□□□□□□□□□□□□□□□□

Prometheus □□:

- `upf_buffer_packets_current`: □□□□□□□□□
- `upf_buffer_bytes_current`: □□□□□□□□
- `upf_buffer_fars_active`: □□□□□□□□□ FAR
- `upf_buffer_packets_dropped{reason}`: □□□□□□□□□

□□□ □□□□ □□□□□□□□□□□□

□□□□□□□

□□ 1: IDLE UE □□□□

```
□□□□□:
- UE □□ IDLE □□□□□□ gNB □□□
- FAR □□: 0x04 (□ BUFF)

□□□□□:
1. DN □□□□□□□□
2. UPF □□ PDR□□□ FAR
3. FAR □□ BUFF □□ → □□□□□□□
4. UPF □ SMF □□ DLDR
5. SMF □□ UE
6. UE □□□ gNB
7. SMF □□ FAR: Action = 0x02 (FORW)
8. UPF □□□ TEID □□□□□□□□□
```

□□ 2: □□□□

```
初始状态:
- UE 连接到 gNB-1 (TEID 1234)
- FAR 动作: 0x02 (FORW)

切换过程:
1. SMF 修改 FAR: Action = 0x06 (FORW+BUFF)
2. 数据包转发到 gNB-1 并缓冲
3. UE 切换到 gNB-2
4. SMF 修改 FAR: TEID = 5678, Action = 0x02 (FORW)
5. UPF 将缓冲的数据包刷新到 gNB-2，使用新 TEID
6. 切换期间没有数据包丢失
```

场景 3: 路径切换

```
初始状态:
- UE 连接，活跃数据流

路径切换:
1. SMF 修改 FAR: Action = 0x04 (仅 BUFF)
2. 所有传入的数据包被缓冲（不转发）
3. 网络重新配置路径
4. SMF 修改 FAR: Action = 0x02 (FORW)，新目标
5. UPF 将所有缓冲的数据包刷新到新路径
```

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, 198.51.100.10）
- 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

规则管理页面提供按 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):

- 5G QoS 流的 6 位标识符
- 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

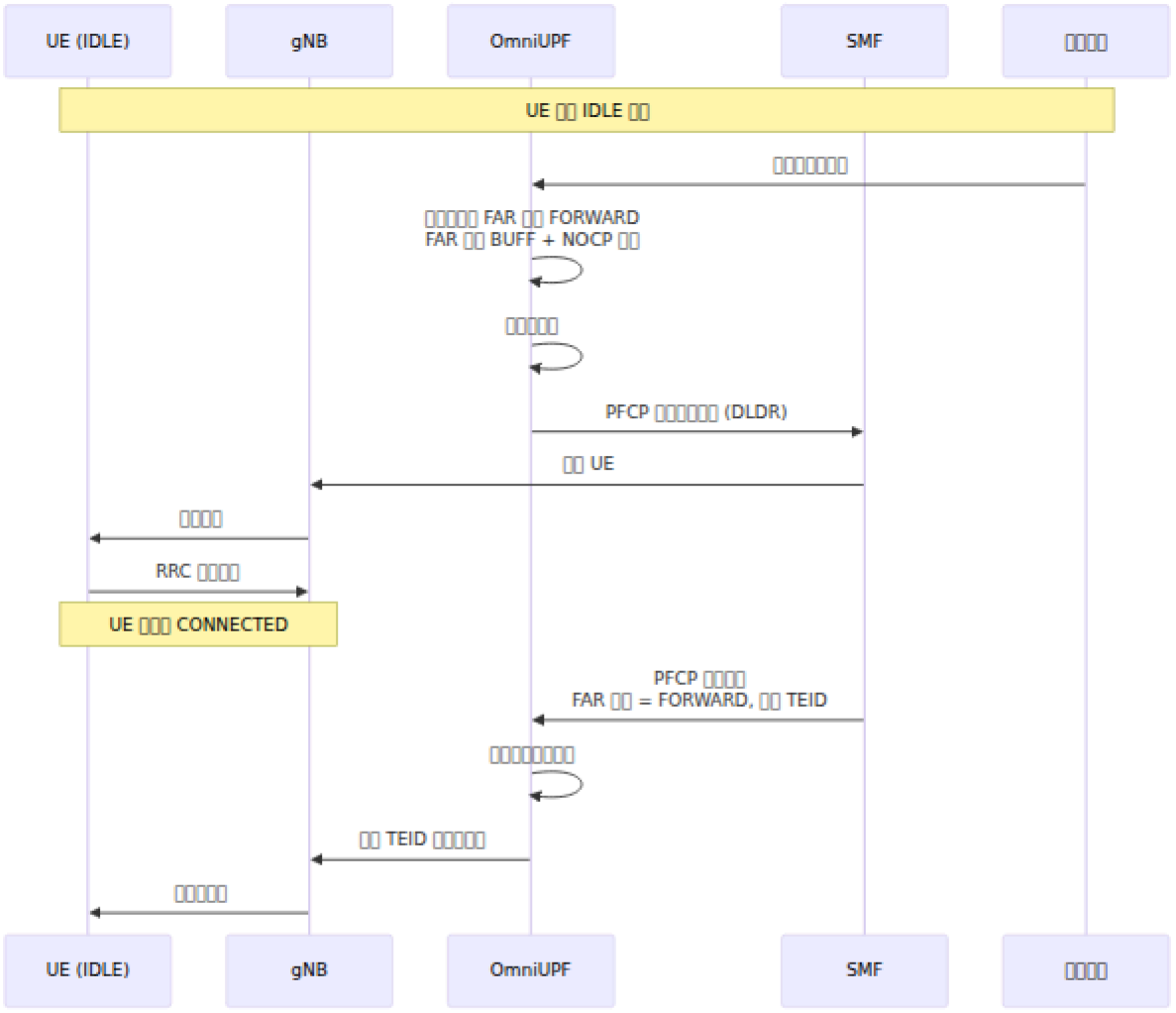
最佳努力流 (GBR = 0):

```
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 使用 滑动窗口速率限制器 来强制执行 MBR（最大比特率）限制，该机制在 eBPF 数据路径中实现。该算法以纳秒精度直接在 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. 窗口检查：如果当前时间在 5ms 的滑动窗口内，丢弃数据包
5. 窗口推进：如果数据包被允许，按 `tx_time` 推进窗口

示例计算：

假设：

- MBR = 100,000 kbps (100 Mbps)
- 数据包大小 = 1500 字节
- 窗口大小 = 5,000,000 ns (5 ms)

```
步骤 1: 以 100 Mbps 计算传输时间
tx_time = (1500 字节 × 8 位/字节) × (1,000,000,000 ns/sec) /
100,000,000 bps
        = 12,000,000,000 / 100,000,000
        = 120 ns

步骤 2: 检查数据包是否适合窗口
current_time = 1000000000 ns
window_start = 999990000 ns
if (window_start + tx_time > current_time):
    丢弃数据包（将超过速率限制）

步骤 3: 如果被允许，推进窗口
window_start = window_start + 120 ns
通过数据包
```

滑动窗口行为

5ms 窗口大小：

- 该算法使用 5 毫秒的滑动窗口
- 如果闲置超过 5ms，窗口会自动重置
- 防止在强制执行平均速率时出现突发饥饿

突发处理：

- 小的突发在 5ms 窗口内是允许的
- 持续流量超过 MBR 将受到速率限制
- 比简单的令牌桶算法更精确

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

速率限制强制点

上行 **(N3 → N6)**:

1. 数据包到达 N3 接口（来自 gNB）
2. 通过 TEID 查找 PDR
3. 通过 QER ID 查找 QER
4. 检查 `ul_gate_status` → 如果关闭则丢弃
5. 应用 `limit_rate_sliding_window()`，使用 `ul_maximum_bitrate`
6. 如果通过，转发到 N6 并更新 URR 计数器

下行 **(N6 → N3)**:

1. 数据包到达 N6 接口（来自数据网络）
2. 通过 UE IP 地址查找 PDR
3. 通过 QER ID 查找 QER
4. 检查 `dl_gate_status` → 如果关闭则丢弃
5. 应用 `limit_rate_sliding_window()`，使用 `dl_maximum_bitrate`
6. 如果通过，添加 GTP-U 头部并转发到 N3

N9 回环 (SGWU ↔ PGWU):

- 在 N9 回环场景中，可能会同时应用上行和下行 QER
- 每个 QER 在 SGWU 和 PGWU 边界独立检查

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:

- □□□□: GTP-U□UDP□IP □□□□□□□□□ ~50-60 □□
- □□□□□□□□: □□□□□□□ = □□□□□□□□□□
- □□□□□□□: □□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□: 5ms □□□□□□□□□□□□□□□ MBR

□□:

```
□□□ MBR: 100 Mbps
□□□□□□□□: ~95-98 Mbps□□□ GTP-U/UDP/IP □□□
```

□□□□□□□□□:

1. □□ URR □□□□□□□□□□□: `upf_urr_*_volume_bytes`
2. □□□□□: `(volume_delta_bytes × 8) / time_delta_seconds / 1000` = kbps
3. □ QER □□□□ MBR □□□□

GBR (□□□□□)

□□: OmniUPF □□□□□□□ GBR □□□□GBR □□□ QER □□□□□□□□□□□□□□□□□□

GBR □□:

- GBR □□□ PFCP □ SMF □□
- GBR □□□ QER □□□□□□ API □□
- □□□□□□□□□□ GBR □□□□□□
- GBR □□□□□□□□□□□□□□□□□□□□□□□

□□□□:

- GBR □□□□□□□□□□□□□□□□
- □□□□□□□□□□ eBPF QoS □□□□

使用报告规则 (URR)

目的

URR 跟踪数据量以进行计费、分析和策略强制。它们维护数据包和字节计数器，这些计数器将报告给 SMF 以生成计费记录。

URR 结构

流量跟踪

上行流量:

- 从 UE 传输到数据网络的字节
- 在 GTP-U 解封装后测量
- 包括 IP 头和有效载荷

下行流量:

- 从数据网络传输到 UE 的字节
- 在 GTP-U 封装前测量
- 包括 IP 头和有效载荷

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 可以根据以下条件触发报告:

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

基于事件:

- 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

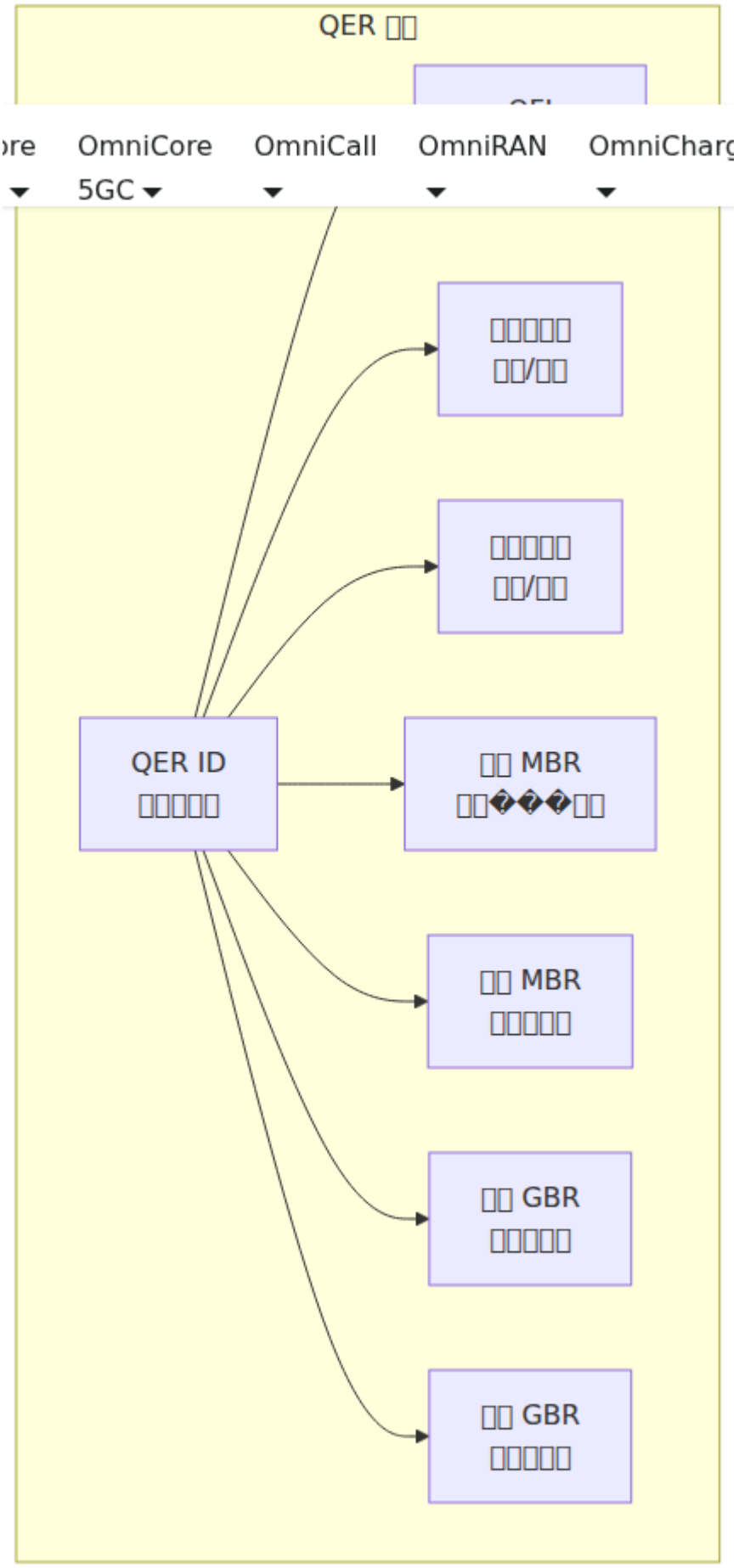
Web UI 自动将流量格式化为人类可读的单位:

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
```

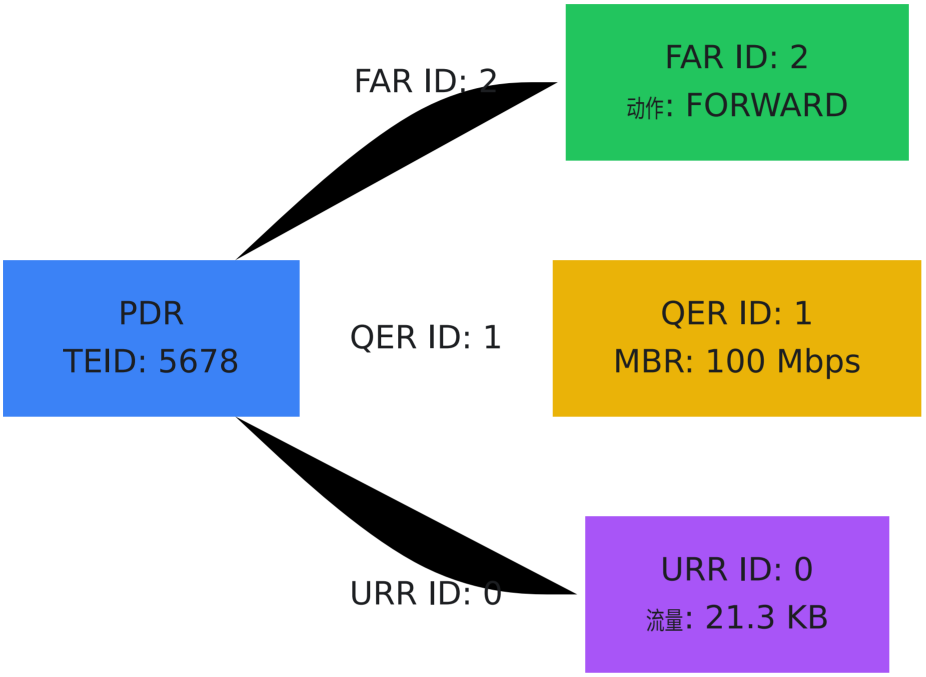
URR □□□□


QER □□
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharg
□□□□□
□□/□□
□□□□□
□□/□□
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: 198.51.100.10
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□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 是否设置（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. 切换完成后点击 "禁用缓冲"

□□ QoS □□□

□□□□□□□□□□□□□□:

1. □□□□□□ → QERs
2. □□□ UE □□□□□□ QER ID
3. □□□□ MBR □□□ MBR □
4. □ URR □□□□□□□□□□

□□□□□□□:

```
□□□ (kbps) = (□□□□□□□ × 8) / (□□□□□ × 1000)
```

□□□□□□□□ MBR□□□□□□□□□□□□

□□□□□□□□

□□ **URR** □□:

1. □□□□□□ → URRs
2. □□□□□□□□□□□□
3. □□□□□□□□□□□□□□□□
4. □□□□□□□□□□□□

□□:

- □□□□□□
- □□□□□□□□
- □□□□□□□□□□

□□□□

□□□□□□

□□ **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

相关文档

- **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

本指南提供了针对常见 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：无效的配置文件

症状：

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

□□□□□□□□□ YAML □□□□

□□□□□

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

# □□□□□
# - □□□□□□□□□□□□□□□□□□□□□
# - □□□□□□□
# - □□□□□□□□□□□□□□□
# - □□□□□□□□□□
```

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

□□□□□□□□□□

□□□

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

□□□□□□□□□□□□

□□□□□

```
# 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:

```yaml
# 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 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  # 从 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：

```
# 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  # 从 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

# 将中断固定到特定 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：

```
# 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:

```
# 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:

```
# 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  # 从 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：

```
# 方法 1: 使用统计 API 进行监控（推荐）
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

# 方法 2: 捕获 PFCP 流量（不受 XDP 影响）
tcpdump -i any -n udp port 8805 -w /tmp/pfcp.pcap

# 方法 3: 带外数据包捕获（推荐用于 GTP-U）
# 使用网络 TAP 或交换机端口镜像捕获流量
# 示例:
# - gNB 和 UPF 之间的物理 TAP
# - 交换机 SPAN/镜像端口复制 N3 流量到分析器
# - 虚拟交换机端口镜像到 hypervisor 中的分析器
#
# 在捕获主机上（而不是 UPF）:
# tcpdump -i <mirror_interface> -n udp port 2152 -w
/tmp/n3_mirror.pcap
```

带外捕获设置示例:

物理网络:

```
# 使用网络 TAP 或配置交换机端口镜像
# 示例: Cisco 交换机 SPAN 配置
(config)# monitor session 1 source interface Gi1/0/1
(config)# monitor session 1 destination interface Gi1/0/24

# 在连接到 Gi1/0/24 的监控主机上:
tcpdump -i eth0 -n udp port 2152 -w /tmp/n3_capture.pcap
```

虚拟环境（**VMware、KVM** 等）:

```
# 配置虚拟交换机端口镜像，将 UPF 流量发送到分析器 VM
# 示例: Linux 桥接 + tcpdump 在不同的 VM 上
# 在 hypervisor 上，将 UPF 的 N3 接口镜像到分析器接口

# 在分析器 VM 上:
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

相关文档

- 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

OmniUPF 围墙花园 / 超出信用重定向

目录

概述

围墙花园功能提供 原生超出信用强制执行，直接在 UPF 中实现，消除了对外部强制执行系统（MikroTik 地址列表、mangle 规则、DNAT）的需求。

当用户用完信用时，SMF 发送带有 `redirect_information` 的 PFCP 会话修改。OmniUPF 在用户空间中拦截该会话的所有流量，并强制执行强制门户体验：

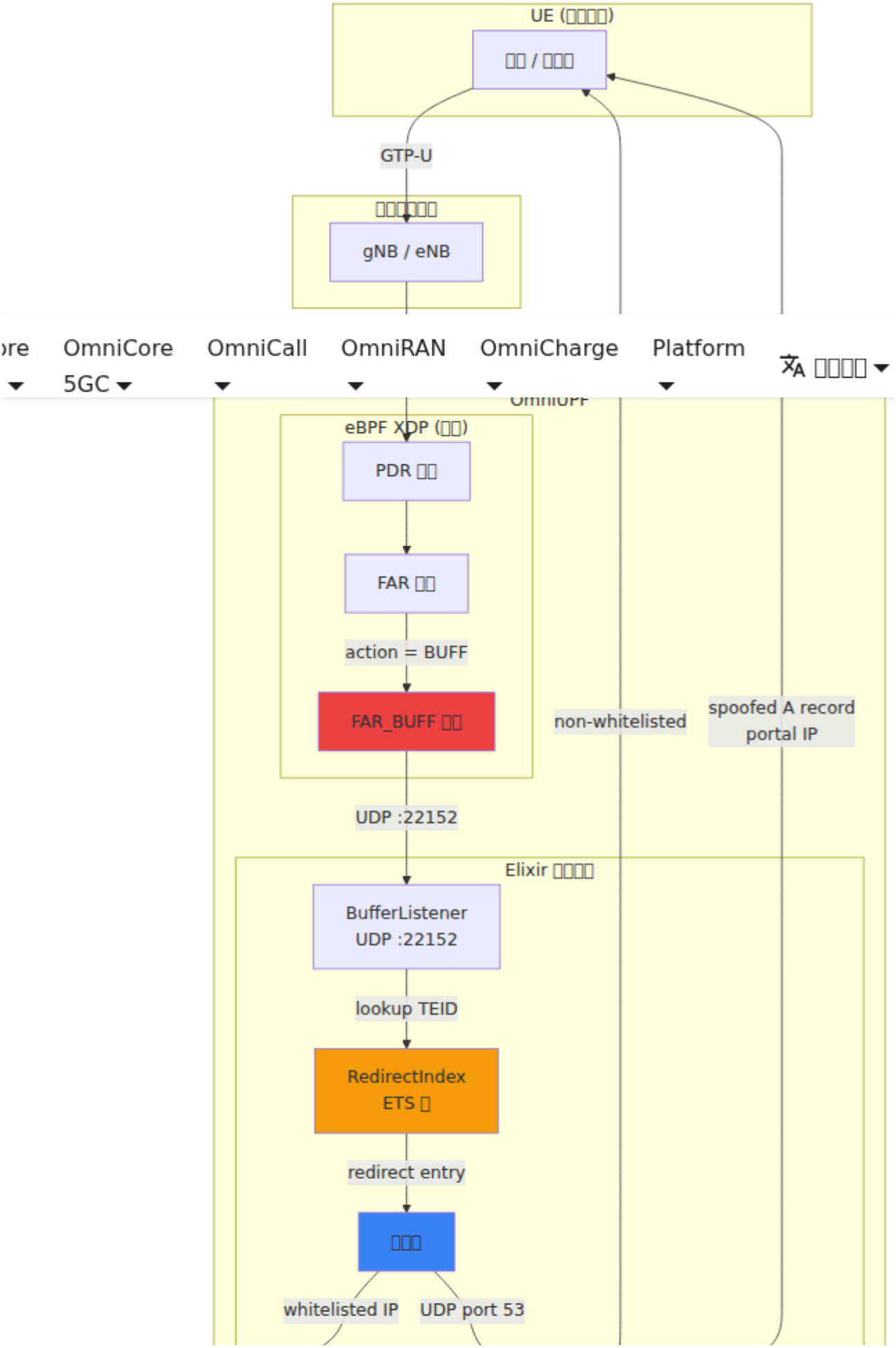
- **DNS** 查询 被伪造以返回门户服务器 IP，触发所有主要设备（Apple、Android、Windows）的强制门户检测
- 流量到门户和白名单 **IP**（支付处理器、验证码服务）正常转发，以便用户可以充值
- 所有其他流量 被静默丢弃

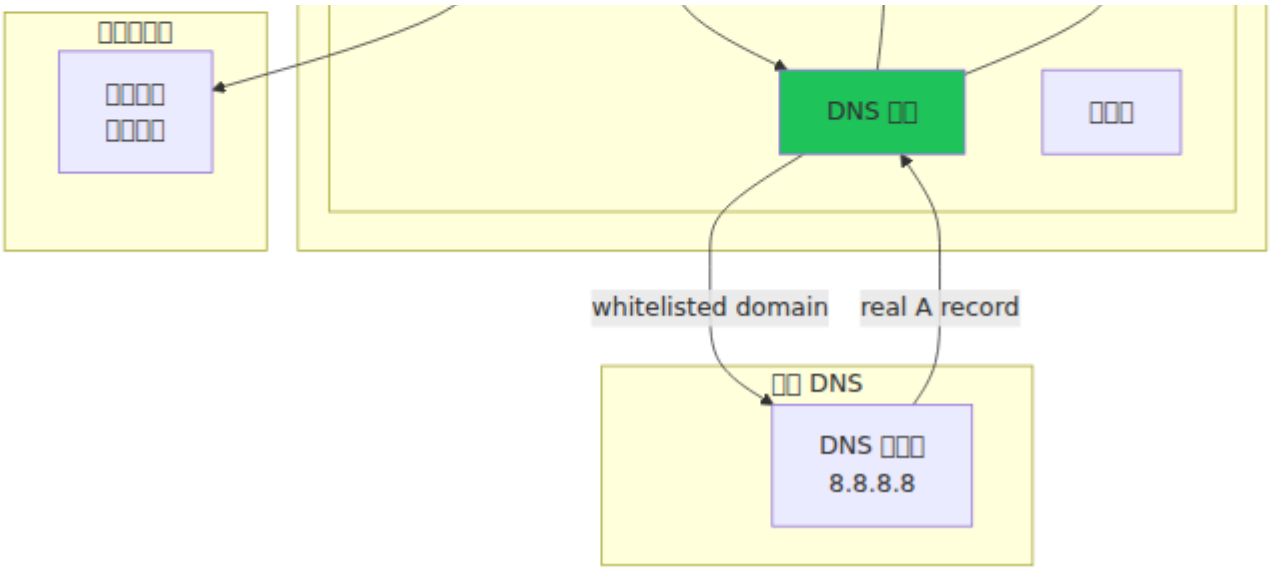
用户看到强制门户提示，并被引导到充值/支付页面。一旦信用恢复，SMF 将 FAR 更新回 FORW，正常转发立即恢复。

关键设计点

- 无需 **eBPF** 更改 -- 重用现有的 `FAR_BUFF` 动作将数据包重定向到用户空间

- 每会话重定向 -- 每个会话可以有不同的门户 IP 和重定向 URL，由 SMF 的 `redirect_information` IE 决定
- 白名单位于 **UPF** -- SMF 只说“重定向此会话”；UPF 决定允许通过哪些流量
- 仅上行拦截 -- 下行 FAR 保持 FORW，以便门户响应通过正常的 GTP 封装路径到达 UE


UE (□□□□)
□□ / □□□
GTP-U
□□□□□□□
gNB / eNB
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□□□
OmniUPF
eBPF XDP (□□)
PDR □□
FAR □□
action = BUFF
FAR_BUFF □□
non-whitelisted
spoofed A record
portal IP
UDP :22152
Elixir □□□□
BufferListener
UDP :22152
lookup TEID
RedirectIndex
ETS □
redirect entry
□□□
whitelisted IP
UDP port 53



处理流程

1. **UE** 发送 **GTP-U** 封装的用户数据包（DNS 查询、HTTP 请求等）到 UPF
2. **eBPF PDR** 匹配 查找到 PDR，应用 FAR
3. **FAR** 指示 **BUFF**（缓存）数据包，发送 FORW 控制消息-- eBPF 程序通过 UDP 端口 22152 向 BufferListener
4. **BufferListener** 收到 TEID，查询 RedirectIndex ETS 表
5. 收到 TEID 的 RedirectIndex 后**Dispatcher** 处理缓存 IP 数据包
6. 处理：
   - 如果是缓存的 DNS 响应，则提取 A 记录中的 IP
   - 如果是缓存的 DNS 查询，则将其转发到配置的上游 IP
   - 如果是缓存 IP 数据包 IP，则将其丢弃
   - 如果是其他数据包，则丢弃
7. **DNS/GTP-U** 层 封装后 GTP-U 头部中 DL TEID 设置为 gNB分配的 UE

N9 接口支持SGW + PGW 部署

在 4G EPC 部署中OmniUPF 可以作为 **SGW-U** 或 **PGW-U**（SGW-C 或 PGW-C 通过标准的 PFCP 接口SGW 会创建 N9 FAR，将下行数据转发到 PGW 分配的 TEID，通过 N3 接口将数据发送PGW 进行最终处理和计费）。

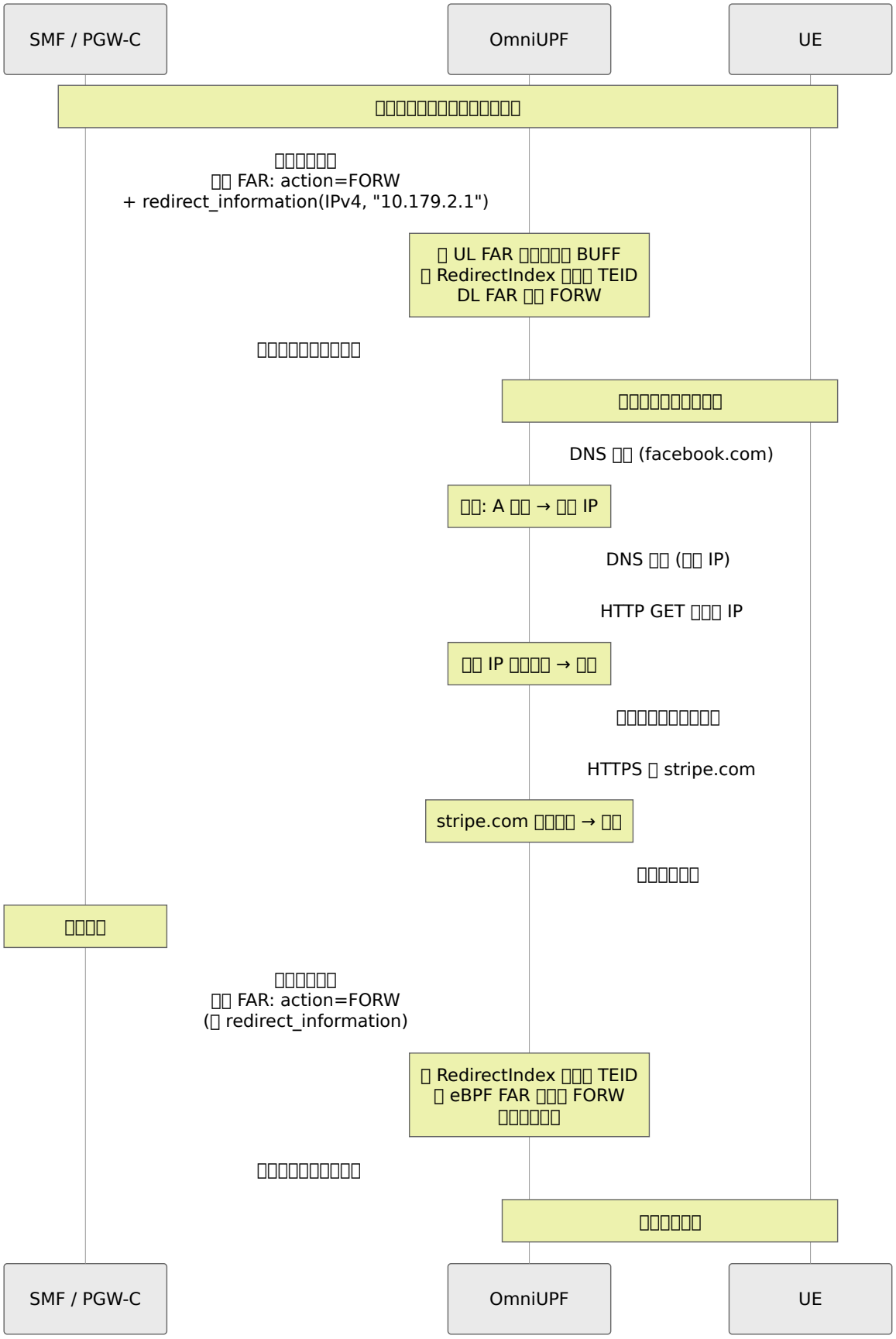
通过 `POST /v1/walled_garden` 添加的域名会被解析为一组动态规则，其处理流程如下：

1. **PGW** 通过 **TEID** 将带 SEID 的 PDR 发送

2. 代码扫描所有 PFCP 会话，以查找一个会话（SGW 会话），该会话具有一个 FAR，其 `teid` 与 PGW UL PDR TEID 之一匹配，并且设置了 `outer_header_creation`。该 FAR 是指向 PGW 的 N9 转发 FAR。
3. 两个 PGW UL FAR 和 SGW 面向 gNB 的 UL FAR 在 eBPF 映射中被覆盖为 `BUFF`。这是必要的，因为 eBPF 程序看到的数据包是从 eNB 到达时带有 SGW 的 TEID，而不是 PGW 的 TEID。
4. 只有 **SGW** 面向 **gNB** 的 **UL PDR** 被注册到 RedirectIndex（而不是 PGW UL PDR）。SGW UL PDR TEID 是 eBPF 在来自 eNB 的传入数据包中看到的内容。
5. 对于 **DL** 路径（将响应发送回 UE），调度器使用 **SGW** 会话的 **DL FAR**（指向 eNB 的 FAR，`remoteip != n3_address`）。DL TEID 在响应时从会话状态实时查找，而不是在激活时缓存 - 这处理了在重定向激活后通过后续会话修改分配 eNB TEID 的情况。

在直接 **UPF** 拓扑中（没有 SGW，SMF 直接与 UPF 通信），只有会话自己的 UL FAR 和 TEID 参与 - SGW 检测代码未找到任何内容，主会话的 UL PDR 直接注册。

PFCP 信令流


SMF / PGW-C
OmniUPF
UE
会话正常建立（默认承载激活）
会话修改请求
更新 FAR: action=FORW
+ redirect_information(IPv4, "10.179.2.1")
将 UL FAR 动作覆盖为 BUFF
在 RedirectIndex 中注册 TEID
DL FAR 保持 FORW
会话修改响应（成功）
用户现在在围墙花园中
DNS 查询 (facebook.com)
伪造: A 记录 → 门户 IP
DNS 响应 (门户 IP)
HTTP GET 到门户 IP
门户 IP 是白名单 → 转发
门户页面（充值表单）
HTTPS 到 stripe.com
stripe.com 是白名单 → 转发
支付页面加载
信用恢复
会话修改请求
更新 FAR: action=FORW
(无 redirect_information)
从 RedirectIndex 中注销 TEID
将 eBPF FAR 更新为 FORW
正常转发恢复
会话修改响应（成功）
正常服务恢复
SMF / PGW-C
OmniUPF
UE

⬚⬚ PFCP IEs

SMF ⬚⬚⬚ FAR ⬚ `forwarding_parameters` ⬚⬚⬚ `redirect_information` ⬚⬚⬚⬚⬚⬚⬚

IE	⬚⬚
`apply_action`	⬚ SMF ⬚⬚⬚ FORW⬚UPF ⬚⬚⬚⬚⬚⬚ BUFF⬚
`redirect_information`	⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚
`forwarding_parameters`	⬚⬚ `redirect_information` ⬚ `outer_header_creation`

⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚ 3GPP TS 29.244 ⬚ 8.2.20-1⬚⬚

⬚⬚	⬚	**UPF** ⬚⬚
IPv4	0	⬚⬚⬚⬚⬚ IPv4 ⬚⬚⬚⬚⬚⬚⬚⬚⬚ IP
IPv6	1	⬚⬚⬚⬚⬚ IPv6 ⬚⬚⬚⬚⬚⬚⬚⬚⬚ IP
URL	2	⬚ URL ⬚⬚⬚⬚⬚⬚ IP⬚⬚⬚⬚ IP ⬚⬚⬚⬚ IP⬚UPF ⬚⬚⬚ URL ⬚⬚ -- ⬚⬚⬚⬚⬚ API ⬚ ⬚⬚⬚⬚⬚⬚⬚⬚ Web ⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚
SIP URI	3	⬚⬚⬚⬚⬚

⬚⬚⬚⬚⬚⬚

⬚⬚⬚⬚⬚⬚⬚⬚ DNS ⬚⬚⬚⬚ ⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚

平台检测域

平台	检测域	预期响应
Apple (iOS/macOS)	`captive.apple.com`	HTTP 200，内容为 `<HTML><HEAD><TITLE>`成功`</TITLE></HEAD><BODY>`成功`</BODY></HTML>`
Android	`connectivitycheck.gstatic.com`	HTTP 204
Windows	`www.msftconnecttest.com`	HTTP 200，内容为 `Microsoft Connect Test`
Samsung	`connectivitycheck.samsung.com`	HTTP 200

当这些域解析到门户 IP 而不是它们的真实地址时，设备会检测到强制门户，并将门户页面呈现给用户，方式为：

- 通知/弹出窗口（iOS、Android）
- 自动浏览器重定向（Windows）

配置的门户 IP 上的门户服务器应为这些检测 URL 提供适当的响应，然后重定向到充值/支付页面。

门户服务器要求（专用 **IP /** 默认虚拟主机）

门户必须将强制门户页面作为其 **IP** 地址的 *□□* 响应提供，而不考虑 **HTTP `Host`** 头。实际上，这意味着为门户提供 自己的专用 **IP**，或使其成为共享主机上的 默认虚拟主机。

此要求直接源于围墙花园的工作原理：

1. 每个非白名单的 DNS 查询都被 伪造为门户 **IP** -- 包括平台强制门户探测 (`connectivitycheck.gstatic.com`、`captive.apple.com`、`www.msftconnecttest.com` 等)。
2. UE 然后打开 HTTP 到门户 IP，但请求携带 探测的 `Host` 头（例如 `Host: connectivitycheck.gstatic.com`），而不是门户的主机名。

3. UPF 将该请求原封不动地转发到门户服务器而不重写 `Host` 头（参见 原始套接字转发）。

如果门户 IP 是 基于名称的虚拟主机 -- 即门户页面仅为特定 `Host:` 值提供服务，并且 IP 与其他虚拟主机共享 -- 服务器将找不到与探测的 `Host` 头匹配的虚拟主机，并将提供其 默认虚拟主机，即错误页面。用户将落在该 IP 上托管的其他内容（另一个应用程序、CRM、空白默认站点）上，而不是充值页面。

请执行以下□作:

- 为强制门户提供 自己的 **IP** 地址（该 IP 返回门户页面的任何 `Host`），或
- 将门户设置为共享主机上的 默认虚拟主机，以便任何不匹配的 `Host:` 都会转发到门户页面。

避免：将 `walled_garden_portal_ip`（或一个 URL 类型的 `redirect_information` 其主机名解析）指向一个 默认 HTTP 响应 □□ 门户页面的 IP。UPF 不会代表门户执行基于 `Host` 的虚拟主机选择或 `Host` 重写。

配置

围墙花园配置位于运行时配置文件中。在生产安装（`.deb` 包）中，配置文件位于 `/etc/omniupf/runtime.exs`。发布启动脚本（`rel/env.sh`）检查此文件，如果存在，则设置 `RELEASE_CONFIG_DIR=/etc/omniupf`，以便 Erlang 发布使用它而不是捆绑的 `config/runtime.exs`。配置更改后必须重启 UPF。

```
#
===================================================================
# 围墙花园 / 超出信用重定向
#
===================================================================

# 启用围墙花园重定向强制执行
walled_garden_enabled = true

# 门户服务器 IP（强制门户 / 充值页面）
walled_garden_portal_ip = "10.179.2.1"

# 可选的 IPv6 门户服务器地址（IPv6 用户的后备）
walled_garden_portal_ip6 = "fd00:179:2::1"

# 用于白名单域查找的上游 DNS 解析器
walled_garden_dns_resolver = "8.8.8.8"

# 白名单域（用户在重定向时可以访问这些域）
# 支持通配符："*.stripe.com" 匹配 "api.stripe.com"
walled_garden_whitelist = [
  "stripe.com",
  "*.stripe.com",
  "js.stripe.com",
  "hcaptcha.com",
  "*.hcaptcha.com",
  "newassets.hcaptcha.com",
]
```

参数

参数	类型	必需	默认	描述
`walled_garden_enabled`	布尔	否	`false`	围墙花园功能的主开关。当 FAR 中的 `redirect_info` 略。
`walled_garden_portal_ip`	字符串（IPv4）	是（如果启用）	`10.179.2.1`	强制门户 / 充值服务器的 的 `redirect_in` URL 主机名无法解析时[SIP URI 时，这是备[IPv4/IPv6 类型，IE 对于 URL 类型，主机名 析。
`walled_garden_portal_ip6`	字符串（IPv6）	否	`nil`	可选的 IPv6 门户地址[备用门户，使用方式与 `walled_garder` 用于 IPv4 相同。当未[向后备不可用。
`walled_garden_dns_resolver`	字符串（IPv4）	否	`8.8.8.8`	用于转发白名单域查询的上[只有白名单域查询发送到此[在本地伪造。
`walled_garden_whitelist`	字符串列表	否	`[]`	用户在围墙花园中可以访问[法，请参见 白名单管理。

白名单管理

白名单控制用户在重定向时可以访问哪些域（及其解析的 IP）。这在 UPF 上配置，而不是 SMF -- SMF 只触发重定向。

模式语法

模式	匹配	不匹配
`stripe.com`	`stripe.com`、`api.stripe.com`、`js.stripe.com`	`evilstripe.com`、`notstripe.com`
`*.stripe.com`	`api.stripe.com`、`js.stripe.com`、`dashboard.stripe.com`	`stripe.com`（精确），`evilstripe.com`
`hcaptcha.com`	`hcaptcha.com`、`newassets.hcaptcha.com`	`evihcaptcha.com`

模式使用 子域锚定：`stripe.com` 匹配域本身及任何子域（`foo.stripe.com`），但不匹配仅包含该字符串的域（`evilstripe.com`）。匹配不区分大小写。

IP 缓存

当 DNS 代理转发白名单域的查询时，解析的 IP 地址会 自动缓存 在白名单中。这意味着：

1. 用户查询 `api.stripe.com`
2. DNS 代理转发到真实解析器，返回 `203.0.113.7`
3. `203.0.113.7` 被添加到白名单 IP 缓存中
4. 随后的 HTTP/HTTPS 流量到 `203.0.113.7` 被转发（而不是丢弃）

门户 IP 始终被白名单，无论配置如何。

推荐的白名单域

对于典型的充值门户，使用 Stripe 支付处理和 hCaptcha：

```
walled_garden_whitelist = [
  # 支付处理器
  "stripe.com",
  "*.stripe.com",

  # 验证码服务
  "hcaptcha.com",
  "*.hcaptcha.com",

  # Google 字体（如果门户使用它们）
  "fonts.googleapis.com",
  "fonts.gstatic.com",

  # 门户资产的 CDN（如果托管在外部）
  "cdn.example.com",
]
```

每会话重定向 URLs

每个 PFCP 会话可以有一个不同的重定向目标。SMF 通过 FAR 中的 `redirect_information` IE 控制这一点：

- **IPv4** 类型：提供的 IP 字符串被解析并用作该会话的门户 IP
- **IPv6** 类型：提供的 IPv6 字符串被解析并用作该会话的门户 IP
- **URL** 类型：主机名在 FAR 创建时提取并通过 DNS 解析。解析的 IP 用作门户 IP。UPF 不使用 URL 路径 -- 它仅存储以供 API 可见性。

这使得不同用户等级、MVNO 或服务计划重定向到不同门户的场景成为可能：

会话	SMF redirect_information	使用的门户 IP	重[
会话 A	`IPv4: 10.179.2.1`	`10.179.2.1`	`10.179.2.1`
会话 B	`IPv6: 2001:db8::1`	`2001:db8::1`	`2001:db8::1`
会话 C	`URL: https://topup.mvno.com/recharge`	解析的 `topup.mvno.com` 的 IP	`https://topup`

UPF ���储每会话的门户 IP 和重定向 URL，这些信息可以通过 API 查看。

API

所有围墙花园端点的基本路径是 `/v1/walled_garden`（没有 `/api` 前缀）。这些端点由配置的 `api_port`（默认：8080）上的 Phoenix HTTP 服务器提供。

GET /v1/walled_garden

返回所有活动的围墙花园重定向会话、白名单 IP、缓存的 CIDR 范围和会话详细信息。

响应：

```
{
  "redirect_count": 2,
  "redirects": [
    {
      "teid": "0x4000",
      "session_seid": 1,
      "portal_ip": "10.179.2.1",
      "redirect_url": null,
      "ue_ip": "10.60.0.1",
      "gnb_ip": "10.179.1.21",
      "dl_teid": "0x5000",
      "far_global_id": 42
    },
    {
      "teid": "0x4001",
      "session_seid": 2,
      "portal_ip": "10.179.2.2",
      "redirect_url": "https://topup.mvno.com",
      "ue_ip": "10.60.0.2",
      "gnb_ip": "10.179.1.21",
      "dl_teid": "0x5001",
      "far_global_id": 43
    }
  ],
  "whitelisted_ips": [
    {"ip": "10.179.2.1", "type": "portal"},
    {"ip": "203.0.113.7", "type": "resolved"},
    {"ip": "203.0.113.6", "type": "resolved"}
  ],
  "whitelisted_cidrs": ["192.168.0.0/24"]
}
```


□□□□□

字段	描述
`redirect_count`	当前活动的围墙花园会话数量
`redirects[].teid`	被拦截的上行 TEID（十六进制）
`redirects[].session_seid`	PFCP 会话 SEID
`redirects[].portal_ip`	此特定会话的门户 IP
`redirects[].redirect_url`	SMF 的重定向 URL（如果是 URL 类型），或 null
`redirects[].ue_ip`	UE IP 地址
`redirects[].gnb_ip`	GTP-U 响应的 gNB IP
`redirects[].dl_teid`	GTP-U 封装的下行 TEID
`redirects[].far_global_id`	内部 FAR 全局 ID
`whitelisted_ips`	当前白名单中的所有 IP（门户 + 缓存的 DNS 解析）
`whitelisted_cidrs`	通过白名单 API 添加的 CIDR 范围

POST /v1/walled_garden

通过 SEID 在现有 PFCP 会话上激活围墙花园重定向。这是运营商/API 触发的路径 - 正常路径是通过 PFCP 会话修改，FAR 中包含 `redirect_information`。API 路径对于测试和需要手动重定向会话而不涉及 SMF 的运营商非常有用。

请求体：

```
{
  "seid": 1,
  "url": "http://10.179.2.1/"
}
```

字段	类型	必需	描述
`seid`	整数	是	要重定向的 PFCP 会话的本地 SEID
`url`	字符串	是	重定向目标 - 一个 IPv4/IPv6 地址或 URL。如果给定 URL，则解析主机名并使用结果 IP 作为门户 IP。完整的 URL 字符串存储以供 API 可见性。

响应（**200 OK**）：

```
{
  "status": "redirect activated",
  "info": {
    "seid": 1,
    "sgw_seid": 7,
    "portal_ip": "10.179.2.1",
    "ue_ip": "10.60.0.1",
    "gnb_ip": "10.179.1.21",
    "ul_teids": ["0x4000", "0x4006"]
  }
}
```

`sgw_seid` 字段在检测到 N9 回环（SGW+PGW 配对会话）并且其上行 TEID 也被 BUFF 时为非空。`ul_teids` 列出在重定向索引中注册的所有上行 TEID。

错误响应：

状态	含义
404	会话未找到（SEID 不匹配任何活动 PFCP 会话）
400	缺少必需参数

DELETE /v1/walled_garden/:seid

停用会话的围墙花园重定向，恢复正常转发。该会话的所有上行 FAR（包括 N9 回环拓扑中的配对 SGW 会话 FAR）在 eBPF 映射中都被设置回 `action=FORW`，并且 TEID 从重定向索引中注销。

路径参数：`:seid` - 要停用的会话的本地 SEID。

响应（**200 OK**）：

```
{"status": "redirect removed", "info": {"seid": 1}}
```

错误响应：

状态	含义
404	会话未找到

GET /v1/walled_garden/whitelist

返回当前白名单：单个缓存的 IP（门户 IP 和 DNS 解析的 IP）以及通过 API 添加的任何 CIDR 范围。

响应：

```
{
  "ips": [
    {"ip": "10.179.2.1", "type": "portal"},
    {"ip": "203.0.113.7", "type": "resolved"}
  ],
  "cidrs": ["192.168.100.0/24"]
}
```

POST /v1/walled_garden/whitelist

在运行时将 IP 地址或 CIDR 范围添加到白名单，而无需重启 UPF。更改仅在内存中有效，并且在重启时不会持久化 - 将永久条目添加到 `runtime.exs` 中的 `walled_garden_whitelist`。

请求体（添加 **IP**）：

```
{"ip": "203.0.113.10"}
```

请求体（添加 **CIDR** 范围）：

```
{"cidr": "192.168.100.0/24"}
```

必须存在 `ip` 或 `cidr` 中的一个。添加 CIDR 范围意味着对该范围内的任何 IP 的流量都由调度器转发，而无需每个 IP 的 DNS 缓存条目。

响应（**200 OK - IP** 添加）：

```
{"status": "added", "ip": "203.0.113.10"}
```

响应（**200 OK - CIDR** 添加）：

```
{"status": "added", "cidr": "192.168.100.0/24"}
```

错误响应：

状态	含义
400	无效的 IP、无效的 CIDR 或缺少请求体字段

Prometheus 指标

计量器

指标：`upf_walled_garden_active_redirects` 类型：计量器 描述：当前在围墙花园重定向状态中的会话数量

示例查询：

```
# 当前重定向计数
upf_walled_garden_active_redirects
```

计数器

指标：`upf_walled_garden_packets_intercepted_total` 类型：计数器 描述：被围墙花园拦截的总数据包（来自重定向会话的所有上行流量）

指标：`upf_walled_garden_packets_dropped_total` 类型：计数器 描述：被围墙花园丢弃的总数据包（非白名单、非 DNS 流量）

指标：`upf_walled_garden_packets_forwarded_total` 类型：计数器 标签：

- `dst_ip` - 转发数据包的目标 IP 描述：通过围墙花园转发到白名单目标的数据包。按目标 IP 标记以便于每个目标的可见性。

指标：`upf_walled_garden_bytes_forwarded_total` 类型：计数器 标签：

- `dst_ip` - 转发流量的目标 IP 描述：每个目标 IP 通过围墙花园转发的字节数。使用此指标识别用户在重定向时访问的白名单服务。

指标：`upf_walled_garden_dns_spoofed_total` 类型：计数器 标签：

- `domain` - 被伪造的域 描述：被围墙花园伪造的 DNS 查询。按查询的域标记。

指标：`upf_walled_garden_dns_forwarded_total` 类型：计数器 标签：

- `domain` - 被转发的白名单域 描述：转发到真实解析器的 DNS 查询（白名单域）。按域标记。

示例查询

```
# 活动的围墙花园会话
upf_walled_garden_active_redirects

# 拦截率（数据包/秒）
rate(upf_walled_garden_packets_intercepted_total[5m])

# 丢弃率（应为拦截的多数）
rate(upf_walled_garden_packets_dropped_total[5m])

# 按目标转发流量（字节/秒）
sum by (dst_ip)
(rate(upf_walled_garden_bytes_forwarded_total[5m]))

# 按流量体积排名前 5 的白名单目标
topk(5, sum by (dst_ip)
(rate(upf_walled_garden_bytes_forwarded_total[5m])))

# 查询最多的伪造域
topk(10, sum by (domain)
(rate(upf_walled_garden_dns_spoofed_total[5m])))

# 白名单域查找率
sum by (domain) (rate(upf_walled_garden_dns_forwarded_total[5m]))

# 丢弃□转发的比率
sum(rate(upf_walled_garden_packets_dropped_total[5m]))
  / sum(rate(upf_walled_garden_packets_intercepted_total[5m]))
```

故障排除

设备上未出现强制门户

症状：用户被重定向（在 API 中可见），但设备未显示强制门户提示。

可能原因：

- □□□□□□□□□□□□□□ IP □□□
- □□□□□□□□□□□□□□□□□ URL
- DNS ���□□□□ UE□□□ GTP-U □□□

□□□□:

1. □□□□□□□□□□□□□□`curl http://<portal_ip>/`
2. □□□□□□□□□ URL□□□□Apple □ `GET /hotspot-detect.html`□
3. □□ `GET /v1/walled_garden` □□□□□□□□□
4. □□ Prometheus □□□`upf_walled_garden_dns_spoofed_total` □□□□□
5. □□□□ GTP-U □□□□□□□□□DL FAR □□□ FORW□

□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□□Stripe □□□

□□□□:

- □□□□□□□□□□□□□
- □□□□□□□□ CDN IP □□□□
- □□□□□□□□□□□□□

□□□□:

1. □□ `GET /v1/walled_garden` - □□ `whitelisted_ips` □□□□□□□ IP
2. □□ Prometheus: `upf_walled_garden_dns_forwarded_total{domain="stripe.com"}` □□□□□□
3. □□□□□□□□□□□□□□□`js.stripe.com`□`m.stripe.network`□
4. □□ `upf_walled_garden_bytes_forwarded_total` □ `dst_ip` □□□□□□

□□□□□□

□□: SMF □□□□ `redirect_information` □□□□□□□□□□□□□□□□□

□□□□:

- `walled_garden_enabled` □ `false`

- `redirect_information` IE 在错误的 FAR 中（必须在上行 FAR 中）
- FAR 动作未被覆盖为 BUFF

解决方案：

1. 验证配置：`walled_garden_enabled = true`
2. 检查 `GET /v1/pfcp_sessions` - 查看会话的 FAR 动作。上行 FAR 应显示动作 `0x04`（BUFF）
3. 检查 `GET /v1/walled_garden` - 会话的 TEID 应出现在重定向列表中
4. 检查 UPF 日志以获取会话修改期间的 `redirect_info` 消息

重定向在充值后未清除

症状：用户已充值，但流量仍被拦截。

可能原因：

- SMF 尚未发送会话修改以移除重定向
- BUFF->FORW 转换未被检测到

解决方案：

1. 检查 `GET /v1/walled_garden` - 会话是否仍在列表中？
2. 检查 `GET /v1/pfcp_sessions` - 验证 FAR 动作已更新回 `0x02`（FORW）
3. 检查 SMF 日志以确认其发送了会话修改
4. 检查 UPF 日志以获取“重定向已移除，注销围墙花园”消息

错误的下行 **TEID**（会话修改后过时）

症状：DNS 伪造响应或转发流量未到达 UE；GTP-U 数据包发送到正确的 gNB IP，但 eNB 将其丢弃为未知 TEID。

原因：在 4G/SGW+PGW 拓扑中，SGW DL FAR 中的 eNB TEID 可能在激活围墙花园重定向时为零或占位符 TEID（例如，`初始上下文设置`交换在会话建立后发生）。当 eNB 分配其 TEID 并通过会话修改将其发送回来时，SGW 会话中的 DL FAR 会被更新 - 但如果调度器缓存了旧值，它将使用错误的 TEID。

解决方案：调度器在响应时*实时*解析 DL TEID，通过查找 SGW 会话的当前 DL FAR（设置了 `outer_header_creation` 且 `remoteip != n3_address` 的 FAR）。只有在会话查找失败时，它才会回退到激活时存储的值。因此，过时的 TEID 应在下一个 DNS 或转发响应发送时自我修正。如果您仍然看到错误的 TEID：

1. □□ `GET /v1/walled_garden` - □□□□□□□□ `dl_teid` □□□□□□□□□□
2. □□ `GET /v1/pfcp_sessions` - □□ SGW □□□ FAR □□□□□ gNB □ FAR □□□□□□ `teid` □ `remoteip`□
3. □□ SGW □□□□□□□□□□□□□□□□□□□□□□□□□□□□ RedirectIndex □□□□□□□□□□□□□□□□□□□□ □□ `DELETE /v1/walled_garden/:seid` □□□□□□□□□ SMF □□□□□

□□□□ 304 □□□

□□: □□

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ HTTP 304□□□□□□□□□□□□□□□□□□□□□□□□ Web □□ □□□□□□□□□□□□□□`/hotspot-detect.html`□`/generate_204` □□□□□□□□□□ `If-Modified-Since` □ `If-None-Match` □□□□□ 304□□□□□□□□□□□□□□□□□□□□□□□□ 304□

□□□□:

1. UPF □□□□□□□□□□□□□□□ HTTP □□ - □□□□□□□□□□□□□□□□□□ Web □□□□□□□□
2. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□200□□□□ 304□□□□□□□□□□□□□ □□□□□□□□
3. □□□□□□□□□□□□□□□□□ `Cache-Control: no-store`□□□□ `ETag`/`Last-Modified` □□ □□□□□□□□□□□□□
4. □□□`curl -v -H "If-None-Match: foo" http://<portal_ip>/hotspot-detect.html` - □□□□ 200□□□□ 304□

□□□□□□□□□□□□□□□□□□□□

□□: □□□□□□□□□ `GET /v1/walled_garden` □□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□ -- □□□□ □□□□CRM □□□□□□□□ -- □□□□□□/□□□□□`curl` □□□□□□□□□□□ `Host:` □□□□□

```
curl -s -H 'Host: portal.example'
http://<portal_ip>/   # -> □□/□□□□   □□□□
curl -s -H 'Host: connectivitycheck.gstatic.com'
http://<portal_ip>/   # -> □□□□□□□□□□□□□
curl -s -H 'Host: <portal_ip>'
http://<portal_ip>/   # -> □□□□□□□□□□□
```

□□: □□ IP □ □□□□

症状	可能原因	修复
转发静默失败	`EPERM` - 原始套接字需要 root 或 `CAP_NET_RAW`	确保 OmniUPF 以 root 身份运行或具有 `CAP_NET_RAW` 权限
数据包已转发但没有响应到达 UE	N6 路由到门户缺失	在 UPF 主机上添加到门户子网的路由
日志中出现 `Walled garden: raw socket open failed`	缺少能力或内核限制	检查 `systemd` 服务 `AmbientCapabilities=CAP_NET_RAW`
转发正常，但 UE 得到错误的 IP 作为源	N6 进行 NAT 重写源	确保门户对 UE IP 的回复通过 UPF 路由回去

检查 UPF 日志以获取 `Walled garden forward failed` 和 `raw socket open failed` 消息。使用 `upf_walled_garden_packets_forwarded_total` 和 `upf_walled_garden_bytes_forwarded_total` Prometheus 指标确认流量是否正常流动。

Prometheus 的高基数警告

注意：围墙花园指标上的 `dst_ip` 和 `domain` 标签如果查询许多唯一目标或域，可能会产生高基数。在大型部署中，考虑使用记录规则来聚合这些指标：

```
# 记录规则以按 /24 子网聚合，而不是单个 IP
sum by (dst_subnet) (
  label_replace(
    rate(upf_walled_garden_bytes_forwarded_total[5m]),
    "dst_subnet", "$1.0/24", "dst_ip", "(\d+\.\d+\.\d+)\.\d+"
  )
)
```

Web UI 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：

- □□ PFCP □□□□□
- □□□□□□□□□□□□
- □□□□□□□□□□□□□□
- □□□□□□□□□□□□
- □□ eBPF □□□□□□□□□

□□□□□

□□□

□□□□□□□□ HTTPS □ OmniUPF □□□□□□□□□

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

□□□□□443□□□□□□□□ HTTPS□

□□

□□□□□□□ `config/config.exs` □□□ OmniUPF □□□

□□□□□□□□□□□□□ UPF □□□

`upf_hosts` □□□□□□ UI □□□□□□□□□□□□□□□ OmniUPF □□□

□□

□□□□□□□□□□□□□□□□□□□□□□

- □□ - `/sessions` - PFCP □□□□□
- □□ - `/rules` - PDR□FAR□QER□URR □□□□
- □□□ - `/buffers` - □□□□□□□□□□
- □□ - `/statistics` - □□□□□□□XDP □□□□□
- □□ - `/capacity` - eBPF □□□□□□□□□
- □□ - `/upf_config` - UPF □□□□□□□□□

- 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：

列	描述
本地 **SEID**	UPF 分配的会话端点标识符
远程 **SEID**	SMF 分配的会话端点标识符
UE IP	用户设备的 IPv4 或 IPv6 地址
TEID	上行流量的 GTP-U 隧道端点标识符
PDRs	会话中的数据包检测规则数量
FARs	会话中的转发动作规则数量
QERs	会话中的 QoS 执行规则数量
URRs	会话中的使用报告规则数量
□作	展开按钮以查看详细的规则信息

特性：

- 按 **IP** 过滤：查找特定 UE IP 地址的会话
- 按 **TEID** 过滤：按隧道端点 ID 查找会话
- 展开会话：查看完整的 PDR/FAR/QER/URR JSON 详细信息
- 自动刷新：每 10 秒更新一次

展开的会话视图：

当您点���会话上的“展开”时，视图显示：

- 数据包检测规则 **(PDRs)**：包含 TEID、UE IP、FAR ID、QER ID、SDF 过滤器的完整 JSON
  - **PDR ID** 可点击 - 点击以导航到规则选项卡并查看完整的 PDR 详细信息
  - 上行 PDR（TEID ≠ 0）链接到上行 PDR 查找
  - 下行 PDR（IPv4）链接到下行 PDR 查找
  - 下行 PDR（IPv6）链接到 IPv6 下行 PDR 查找
- 转发动作规则 **(FARs)**：动作标志、外部头创建、目标端点

- **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`

规则视图提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
□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，以提高性能

□□

□□□□□□□□

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
□作	缓冲控制按钮（药丸样式）

缓冲区控制□作

对于每个具有缓冲数据包的 FAR，提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□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

统计视图提供来自 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

文档和审计：

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

路由视图提供对用户设备（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□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□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

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

相关文档

- 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** 文档 - 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

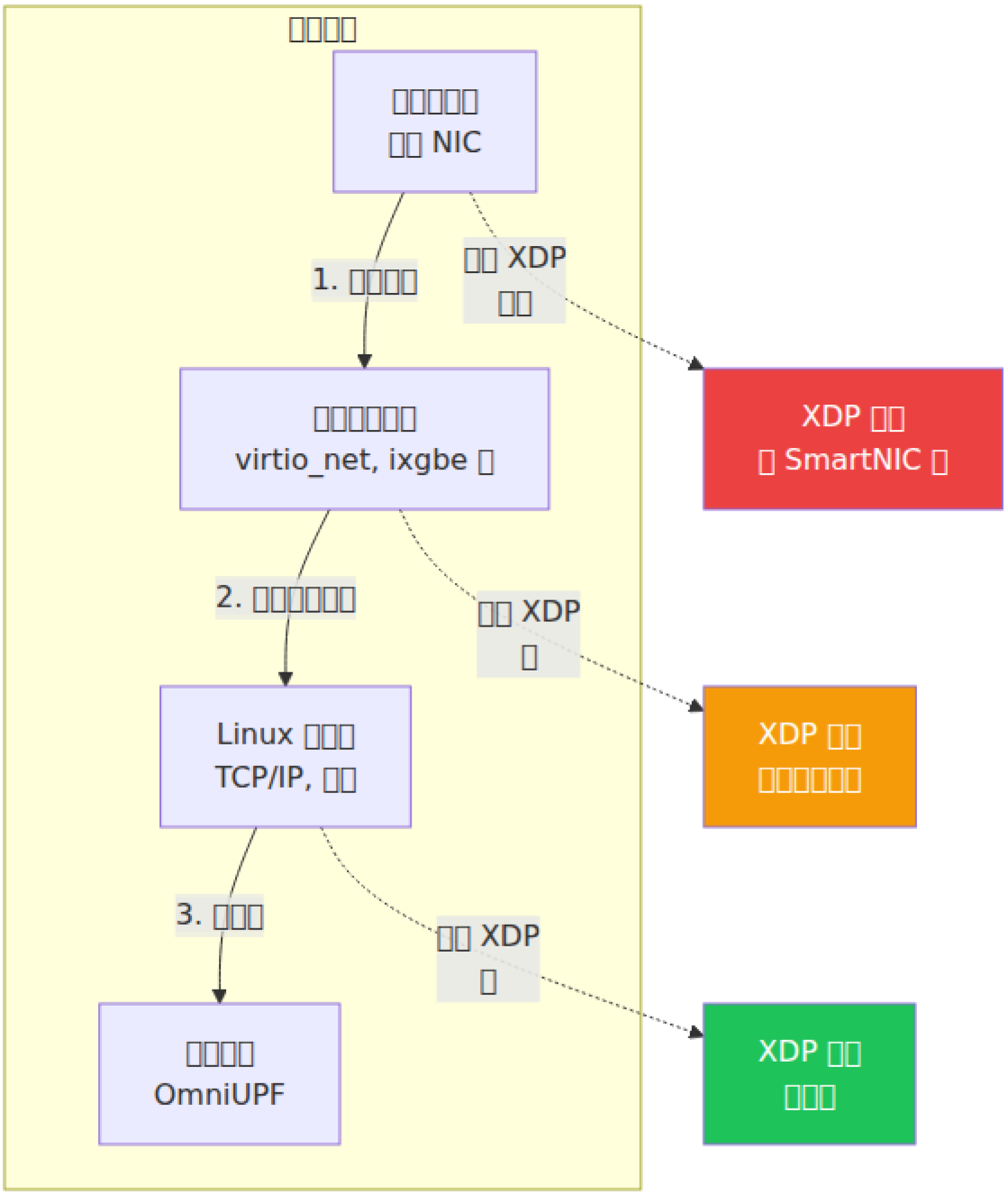
OmniUPF 的 XDP 附加模式

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）在网络栈中尽可能早的点运行，从而提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 的驱动程序	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 之后 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。

□□□□

- □□□□~1-2 □□□□□□□□ (Mpps)
- □□□□□□□□□ ~100 □□
- **CPU** □□□□□□□□□□ XDP □□□□□□□□□□

□□□□

- □□□□□□□□□
- □□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□

□□

```
# /etc/omniupf/runtime.exs
xdp_interfaces = "eth0"
xdp_attach_mode = "generic"    # □□□□
```

□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ CPU □□□

□□□□□□□□□□□□

□□

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

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
- 任何对性能有要求的部署

NIC 驱动程序要求

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`
- 美满科技：`mlx4_en`、`mlx5_core`
- Netronome：`nfp`（支持卸载）
- 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

```
# /etc/omniupf/runtime.exs
xdp_interfaces = "eth0"
xdp_attach_mode = "native"
```

多队列要求：为了获得最佳性能，在虚拟 NIC 上启用多队列（请参见下面的 Proxmox 部分）。

□□□□□SmartNIC□

□���

□□ XDP □□□ NIC □□□SmartNIC□□□□ eBPF □□□□□□□ CPU □□□□□□□

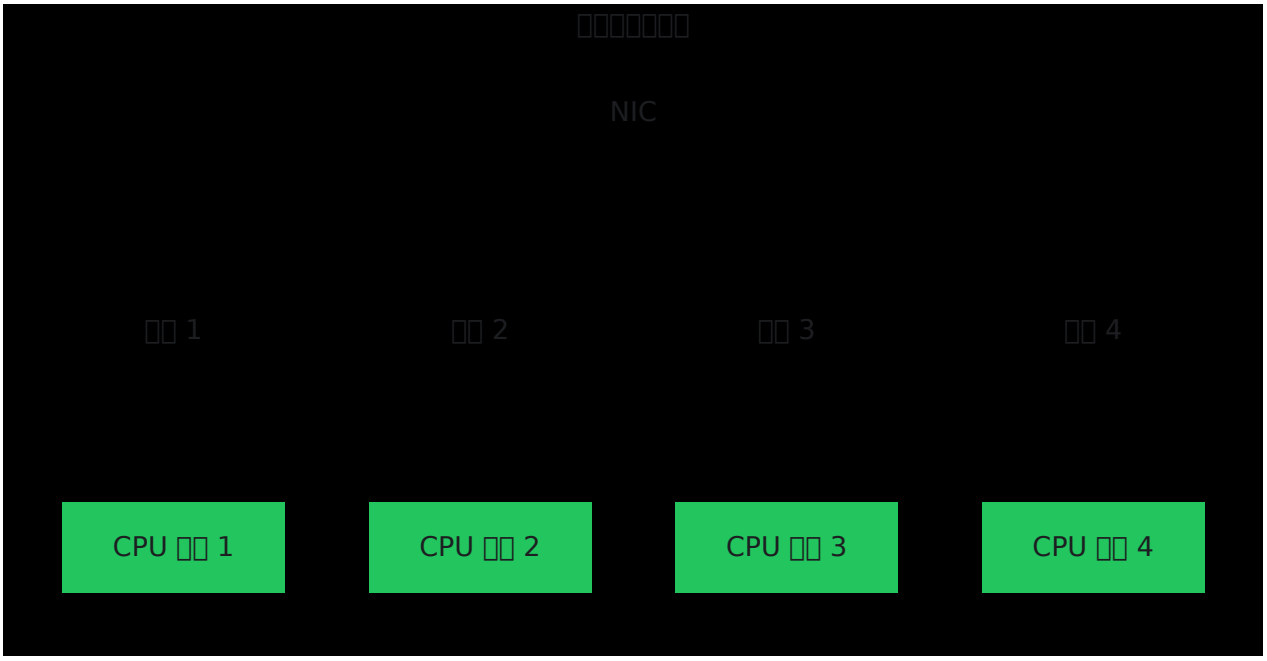
在 Proxmox VE 上启用本地 XDP

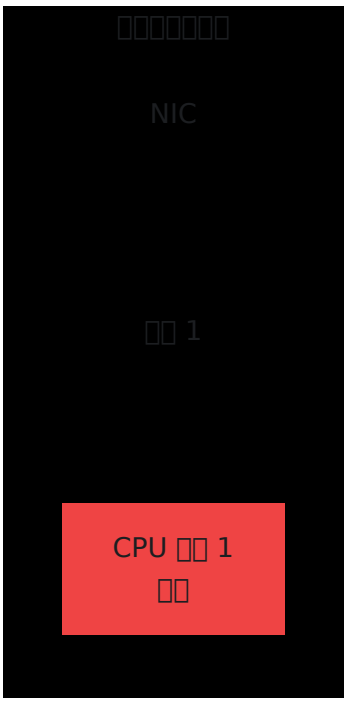
Proxmox VE 使用 **VirtIO** 网络设备用于虚拟机，这些设备通过 `virtio_net` 驱动程序支持本地 XDP。然而，您必须启用 多队列 以获得最佳性能。

第 1 步：理解要求

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





第 2 步：在 Proxmox 中创建虚拟机

方法 A：使用 Proxmox Web UI

1. 登录到您的 Proxmox Web 界面
   - 打开 Proxmox Web 界面并登录。
   - 点击 创建

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 □□□□□

第 3 步：在虚拟机内部验证多队列

虚拟机重启后，SSH 登录虚拟机并验证：

```
# 检查队列配置
ethtool -l eth0

# 预期输出：
# eth0 的通道参数：
# Combined:     8         <-- 应□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

# 两者都应显示 8（或您配置的值）
```

第 4 步：在 OmniUPF 中启用本地 XDP

编辑 OmniUPF 配置：

```
# 编辑配置文件
sudo nano /etc/omniupf/runtime.exs
```

更改 XDP 模式：

```
# 之前
xdp_attach_mode = "generic"

# 之后
xdp_attach_mode = "native"
```

重启 OmniUPF：

```
sudo systemctl restart omniupf
```

第 5 步：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",
```

常见的 Proxmox 问题

问题："附加 XDP 程序失败"

解决方案：

- 验证多队列是否启用（`ethtool -l eth0`）
- 检查内核版本：`uname -r`（必须 ≥ 5.15）
- 确保 VirtIO 驱动程序已加载：`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）才能更改队列
- 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未提高

解决方案：

- 检查 CPU 绑定（避免过度分配）
- 监控 `top` - CPU 使用率应分散在多个核心上
- 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+
- 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 接收端侧缩放 设置为 启用
3. 编辑 **.vmx** 文件（可选，用于更多队列）：

```
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**（在 `runtime.exs` 中）：

```
xdp_interfaces = "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
```

微软 **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+

- 生成 2 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**（在 `runtime.exs` 中）：

```
xdp_interfaces = "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 的唯一选项
```

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"
# ✗ "附加失败" 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,      # 传递到栈的数据包
  "xdp_redirect": 9012,  # 重定向的数据包
  "xdp_tx": 3456         # 发送的数据包
}
```

4. 验证驱动程序支持

```
# 检查驱动程序是否支持 XDP
ethtool -i eth0 | grep driver

# 对于 Proxmox/KVM: 应显示 "virtio_net"
# 对于 VMware: 应显示 "vmxnet3"
# 对于 Hyper-V: 应显示 "hv_netvsc"
```

5. 性能测试

比较启用前后的数据包处理:

```
# 监控数据包速率
watch -n 1 'curl -s http://localhost:8080/api/v1/packet_stats | jq .rx_packets'

# 通用模式: ~1-2 Mpps
# 本地模式: ~5-10 Mpps (提高 5-10 倍)
```

故障排除 XDP 问题

问题: 启动时 "附加 XDP 程序失败"

症状:

```
错误: 无法将 XDP 程序附加到接口 eth0
```

诊断:

1. 检查驱动程序支持:

```
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 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
- 如有必要，重新构建内核以支持 XDP

问题：使用本地模式时性能未提高

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 □□□□□□ eBPF □□□□□□□□□□□□□□□□

1. □□ **eBPF** □□□□□□□

```
dmesg | grep -i bpf | tail -20
```

2. □□□□□□□□□□

```
# eBPF □□□□□□□
curl http://localhost:8080/api/v1/map_info

# □□□□□□ 100% □□□
```

□□□□□□

- □□ eBPF □□□□
- □□□□□□□□□□□□□□□□ eBPF □□
- □□ Linux □□□□□□□□□□ eBPF

□□□Proxmox □□□□□□□□□

□□□□□□□□□□□□□□□

```
# 在 Proxmox 主机上
qm status YOUR_VM_ID

# 必须显示 "状态: 已停止" 然后才能启动
```

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

# 然后在虚拟机内部检查
ethtool -l eth0
```

重要: 更改队列数量需要 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 ...
```

性能影响摘要

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	提高 **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：

英特尔 **NIC**（推荐用于裸金属）

型号	速度	驱动程序	**XDP** 支持	备注
英特尔 **X520**	10GbE	ixgbe	本地 ✓	经过验证，广泛可用，性价比好
英特尔 **X710**	10/40GbE	i40e	本地 ✓	出色的多队列支持
英特尔 **E810**	100GbE	ice	本地 ✓	最新一代，最佳性能
英特尔 **i350**	1GbE	igb	本地 ✓（内核 5.10+）	适合较低带宽需求

美满科技**/NVIDIA NIC**（高性能）

型号	速度	驱动程序	**XDP** 支持	备注
ConnectX-4	25/50/100GbE	mlx5	本地 ✓	高吞吐量，适合边缘计算
ConnectX-5	25/50/100GbE	mlx5	本地 ✓	出色的性能，硬件加速
ConnectX-6	50/100/200GbE	mlx5	本地 ✓	最新一代，适合超高吞吐量
BlueField-2	100/200GbE	mlx5	本地 ✓	具有 DPU 功能的 SmartNIC

博通 **NIC**

型号	速度	驱动程序	**XDP** 支持	备注
BCM57xxx 系列	10/25/50GbE	bnxt_en	本地 ✓	常见于 Dell/HP 服务器

虚拟 **NIC**（虚拟机部署）

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）	虚拟机最佳选择
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（约 $2,500-5,000）
Netronome	Agilio CX 100G	100GbE	企业级仅

注意：硬件卸载 NIC 稀少且昂贵，且需要裸金属部署。大多数部署应使用本地 XDP。

测试配置

这些配置已在生产中与 OmniUPF 验证：

预算选项（**1-10 Gbps**）

- **NIC**□□□□ X520□10GbE □□□□
- □□□□□ XDP
- □□□□□□ UPF □□ ~8-10 Gbps
- □□□ ~$100-200□□□/□□□

□□□**10-50 Gbps**□

- **NIC**□□□□ X710□40GbE□□□□□□ ConnectX-4□25GbE□
- □□□□□ XDP
- □□□□□□ UPF □□ ~25-40 Gbps
- □□□ ~$300-800

□□□**50-100+ Gbps**□

- **NIC**□□□□□ ConnectX-5/6□100GbE□
- □□□□□ XDP
- □□□□□□ UPF □□ ~80-100 Gbps
- □□□ ~$1,000-2,500

□□□□□□□**Proxmox/KVM**□

- **NIC**□VirtIO□□□ 8-16 □□□
- □□□□□ XDP
- □□□□□□ UPF □□ ~5-10 Gbps
- □□□□□□□□□□□

□□□□□□□

□□□□□□□□ OmniUPF □□□

NIC/□□	□□	□□□
Realtek NICs	□ XDP □□□Linux □□□□□	□□□ i350 □□□
VirtualBox	□□□□□ XDP	□□□ Proxmox/KVM
□□□ **NICs**	□□□□□□□□□□□	□□□□□□□/□□□□
□□□□ **NICs**□<2014□	□ XDP □□□□□□	□□□ 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>

# 在虚拟机内部:
ethtool -l eth0  # 验证 8 个队列
sudo nano /etc/omniupf/runtime.exs  # 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 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 的用户平面功能。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：监控控制平面节点的关联和状态

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□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：按协议的实时数据包计数 (GTP, IP, TCP, UDP, ICMP, ARP)
- **XDP** 统计：数据路径性能指标 (通过、丢弃、重定向、终止)
- **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)

□□□□□

- **UE** □□□□□ UE IP □ gNB □□□□ (`GET /routes`)
- **FRR** □□□□□□□□□□□□□□□□□□□□ (`POST /routes/sync`)
- □□□□□□□□□□□□□□ (`GET /routing/sessions`)
- **OSPF** □□□□□□□ OSPF □□□□□□□□ (`GET /ospf/database/external`)

□□□

- **UPF** □□□□□□□□□□ (`GET /config`, `POST /config`)
- □□□□□□□□□□□□□□□□□□ (`GET /dataplane_config`)
- **XDP** □□□□□ XDP □□□□□□□□□ (`GET /xdp_capabilities`)
- **eBPF** □□□□□□□□□□□□□□□□ (`GET /map_info`)

Web UI □□

OmniUPF Web UI □□□ API □□□□□□□ API □□□□□□□□□□□□□ Web UI □□ □□□□□□□□□□□□ □□

Swagger API □□

□ API □□□□ **OpenAPI 3.0 (Swagger)** □□□□□□□□□□□□ Swagger UI □□□

- □□□□□□□□□□□□□/□□□□
- □□□□□□□□ API □□□□□□□
- □□□□□□□□□□□
- HTTP □□□□□□□□

□□□ 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 特性

分页

OmniUPF API 支持对返回大型数据集的端点进行分页。分页可以防止超时并减少查询数千个会话、PDR 或 URR 时的内存使用。

支持的分页样式：

1. 基于页面的分页（推荐）：
    - `page`: 页码（从 1 开始）
    - `page_size`: 每页项目数（默认: 100, 最大: 1000）
2. 基于偏移量的分页:
    - `offset`: 要跳过的项目数
    - `limit`: 要返回的项目数（最大: 1000）

示例请求:

```
# 基于页面: 获取每页 50 项的第二页
GET /api/v1/pfcp_sessions?page=2&page_size=50

# 基于偏移量: 跳过前 100 项, 返回接下来的 50 项
GET /api/v1/pfcp_sessions?offset=100&limit=50

# 默认行为（无分页参数）：前 100 项
GET /api/v1/pfcp_sessions
```

响应格式:

```
{
  "data": [
    { /* session object */ },
    { /* session object */ },
    ...
  ],
  "pagination": {
    "total": 5432,
    "page": 2,
    "page_size": 50,
    "total_pages": 109
  }
}
```

分页端点:

- `/api/v1/pfcp_sessions` - PFCP 会话列表

- `/api/v1/pfcp_associations` - PFCP □□□□
- `/api/v1/routes` - UE IP □□
- `/api/v1/uplink_pdr_map` - □□ PDR□□□□□□
- `/api/v1/uplink_pdr_map/full` - □□ PDR □□□ SDF □□□□□□
- `/api/v1/downlink_pdr_map` - □□ PDR IPv4□□□□□□
- `/api/v1/downlink_pdr_map/full` - □□ PDR IPv4 □□□ SDF □□□□□□
- `/api/v1/downlink_pdr_map_ip6` - □□ PDR IPv6□□□□□□
- `/api/v1/downlink_pdr_map_ip6/full` - □□ PDR IPv6 □□□ SDF □□□□□□
- `/api/v1/far_map` - □□□□□□
- `/api/v1/qer_map` - QoS □□□□
- `/api/v1/urr_map` - □□□□□□

□□□□□□□

- `GET /api/v1/buffer` - □□□□ FAR □□□□□□□□□
- `GET /api/v1/buffer/:far_id` - □□□□ FAR □□□□□
- `GET /api/v1/buffer/notifications` - □□ DLDR □□□□
- `DELETE /api/v1/buffer` - □□□□□  □□□
- `DELETE /api/v1/buffer/:far_id` - □□□□ FAR □□□
- `POST /api/v1/buffer/:far_id/flush` - □□□□□□□□□□□
- `POST /api/v1/buffer/:far_id/notify` - □□□□ DLDR □□

□□□□□

- `GET /api/v1/config` - □□□□ UPF □□
- `POST /api/v1/config` - □□ UPF □□□□□□□□□□□□□
- `GET /api/v1/dataplane_config` - □□□□□□□□□□

□□□□□□□

- `GET /api/v1/routes` - □□ UE □□
- `POST /api/v1/routes/sync` - □□□ FRR □□□□□
- `GET /api/v1/routing/sessions` - □□□□□□□□
- `GET /api/v1/ospf/database/external` - □□ OSPF □□ LSA □□□

□□□□□

- □□ Web UI □□□□□ `page_size=100`
- □□□□□□□□□ `page_size=1000`□□□□□□
- □□ `pagination.total_pages` □□□□□□□
- □□ `page_size` □□□ API □□□□□□□□

CORS □□

□□□□□□ (CORS) □□□□□ API □□□□□□□ Web UI □□□□□□□□□□□□□□□ API□

Prometheus □□

□□ REST API□OmniUPF □□ `/metrics` □□□□□□□ `:9090`□□□□ Prometheus □□□

□□□□□

- □□□□□□ PFCP □□□□□□□
- □□□□□□□□□□□
- XDP □□□□□
- □□□□
- eBPF □□□□□□□
- URR □□□□

□□□ □□□□ □□□□□□□□

□□□□

- **Web UI □□** - □□□ API □□□□□□□□□
- □□□□ - Prometheus □□□□
- **PFCP □□□□** - PFCP □□□□□□□□□
- □□□□□□ - PDR□FAR□QER□URR □□
- □□□□□□ - FRR □□□ UE □□
- □□□□ - □□□□□□□□□
- □□□□ - UPF □□□□
- **Swagger UI** - □□□ API □□□□ `localhost` □□□□□ UPF □□□

UE 路由管理

> 相关文档：
>
> - 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。

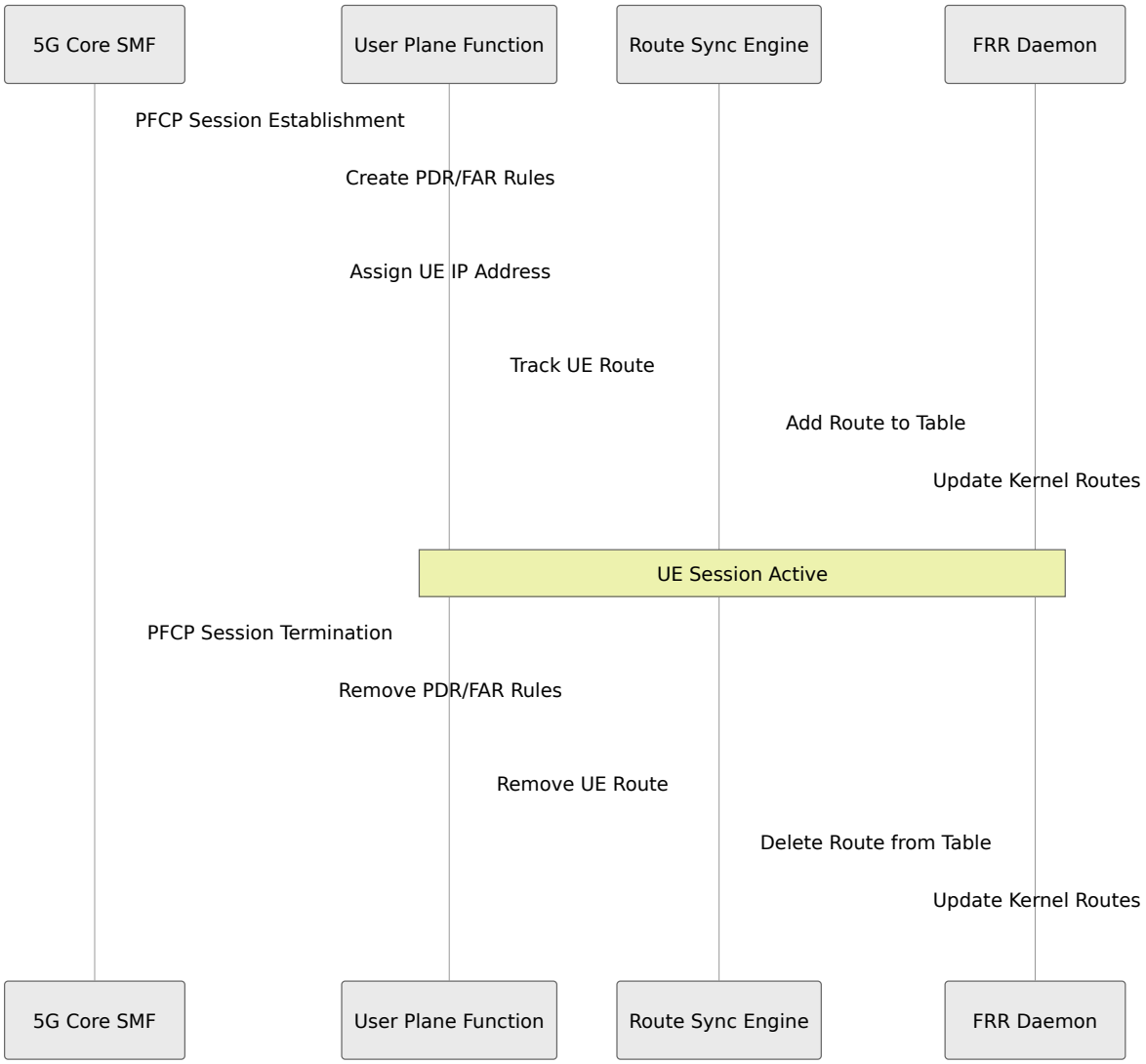
什么是 **FRR**？

FRR（自由范围路由）是一个强大且开源的路由协议套件，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 会话及其关联的 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 剧本中定义一次 FRR 配置作为 **Jinja2** 模板，Ansible 会在部署期间自动将其传播到所有 UPF 实例。

一个典型的 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 static
 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. 定义一次: 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: 在您的 Ansible 清单中为每个 UPF 主机设置变量
3. 全面部署: 运行 Ansible 剧本将 FRR 配置部署到所有 UPF 节点

4. □□□□□Ansible □□□□□□□□□□□IP □□□□□□ ID □□□□□□ UPF □ FRR □□

Jinja2 □□□□□□□□□□

- **OSPF** □□□□□□ ID□□□□□□□□□□□□□□□□
- **BGP** □□□ASN□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□`redistribute static` □□ UE □□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□OSPF/BGP □□□□

UPF □□□□□□□□ FRR □□□□□□□□ UPF □□□UPF □□□□□□ PFCP □□□□□ FRR vtysh □□□□□□ UE IP □□□□ □□□□□IPv4 □ /32□IPv6 □ /128□□□□□□□□□

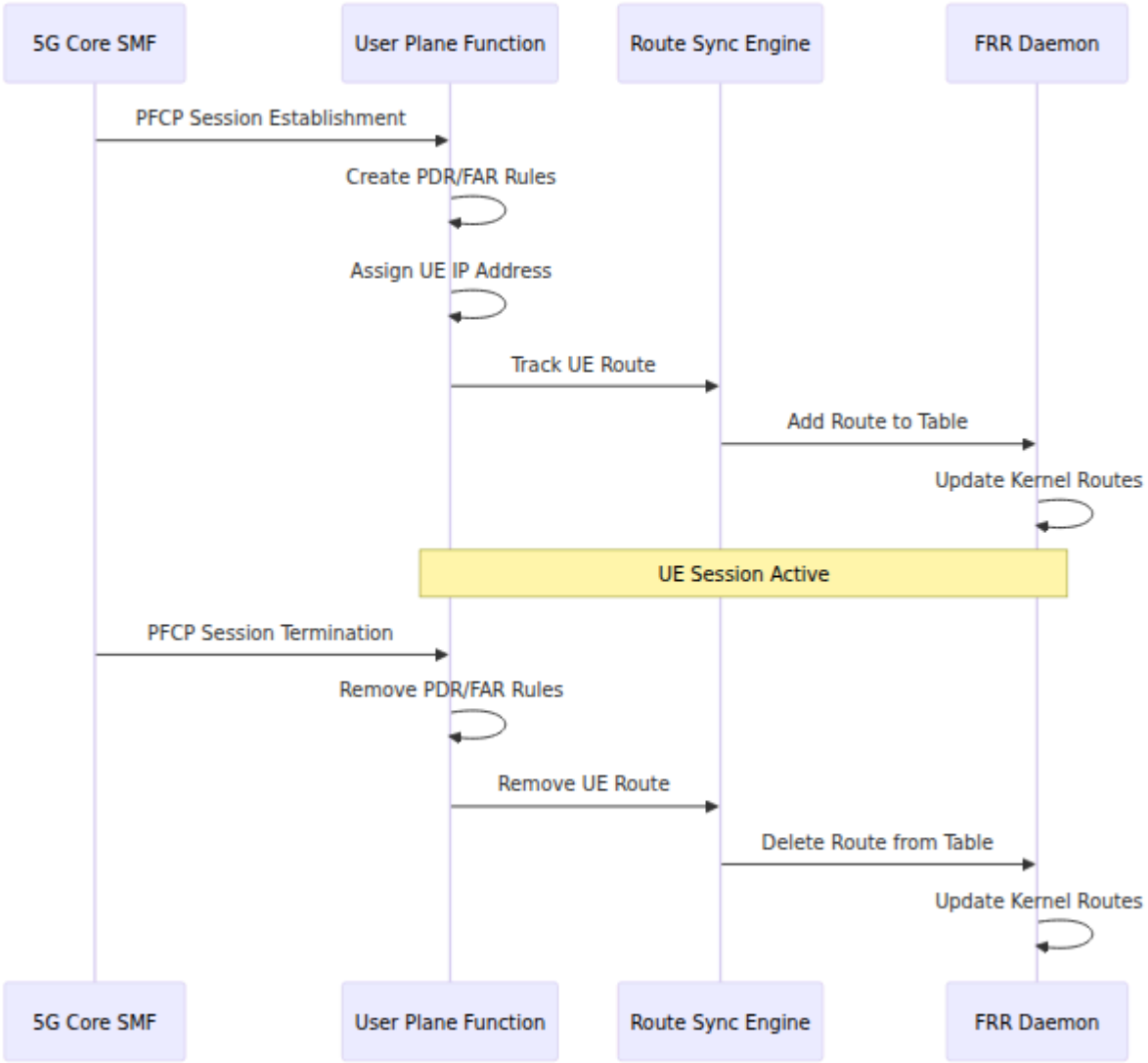
1. □□ **FRR** □□□□□□□□ UPF □□□□□□□□□ vtysh□
2. **□□ `redistribute static` □□□ FRR □□
3. □□□□ **FRR** □□□□□□□□□□OSPF□BGP□□□
4. □□□□□□□□ UE □□□□□□□□ UPF □□

> □□□UPF □□ FRR □ vtysh □□□□□□□□□□□ FRR □□□□□□□□□□□□□□□□□□□ OSPF/BGP □ □□□□ `redistribute static`□□□□ `redistribute kernel`□

□□□□

- □□□□□□□□□□□ Ansible □□□ FRR Jinja2 □□□□□□□□□□□□□□□ UPF □□
- **Ansible** □□□□□□□□Jinja2 □□□□□□□□□□□□□□□OSPF □□□BGP □□□□□□□□□□□□□
- **UPF** □□□□□□□□□□ UPF □□□□□□□□□□ PFCP □□□□□□ UE IP /32 □□
- □□□□□□□□□□ UPF □□ FRR □□□□□□□□□□□□□□□□□ UE □□
- □□□□□□□□ Ansible □□□□□□□□□□□□□□□□□□□□□□□□ UPF □□□□□

路由通告



监控和管理

Web UI 集成

UPF 控制面板提供一个 路由 页面，显示：

- 路由状态：路由同步是否启用或禁用
- 总路由：正在跟踪的 UE IP 地址数量
- 同步统计：成功同步的路由数量及任何失败
- 活动路由：当前路由表中所有 UE IP 地址的实时列表
- **OSPF** 邻居：与邻居详细信息的 OSPF 邻接状态

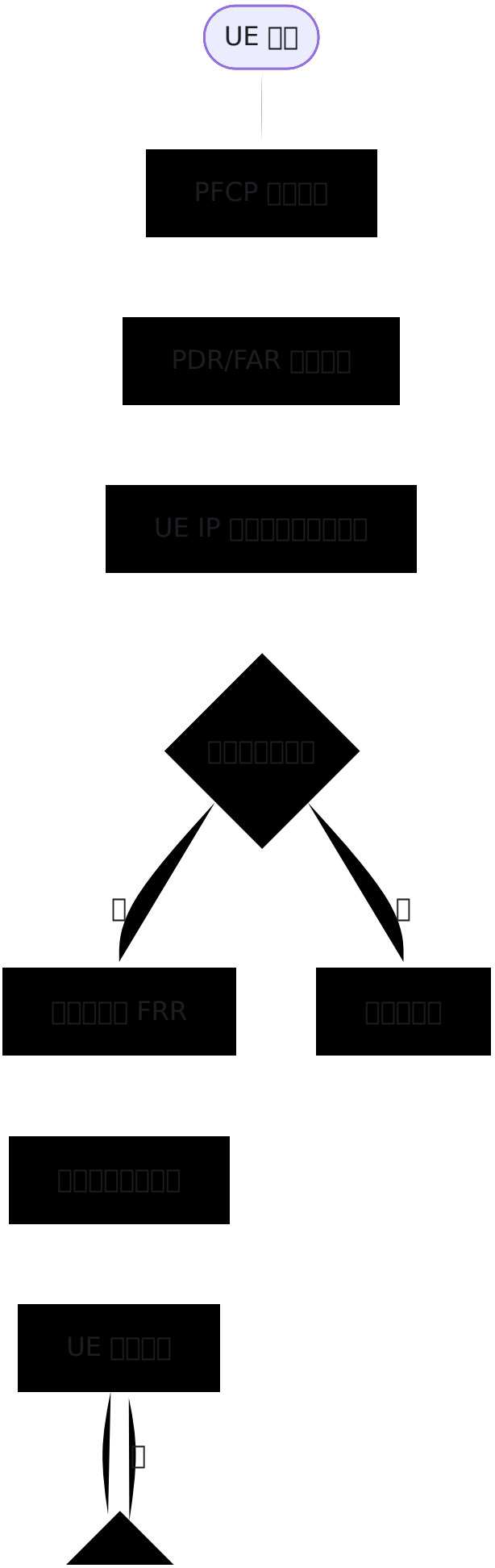
- **BGP** 对等体：BGP 会话状态和前缀统计（在配置时）
- **OSPF** 重新分发路由：显示 UE 路由如何被通告的外部 LSA 的完整视图

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

手动同步□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 中当前活动的 UE 会话列表
2. 与 FRR 的路由表进行比较
3. 添加任何缺失的路由
4. 移除任何过时的路由
5. 返回更新的同步统计


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

优势

- 零接触配置：路由自动管理，无需手动干预
- 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 以获取完整的端点详细信息和示例

路由格式

路由作为简单的 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

IPv6 支持

路由管理器支持 IPv4 和 IPv6 UE 地址：

地址类型	前缀长度	示例
IPv4	/32	`100.64.18.5/32`
IPv6	/128	`2001:db8::1/128`

对于 IPv6，请确保您的 FRR 配置包括适当的 OSPFv3 或 BGP IPv6 重新分发：

```
router ospf6
 redistribute static
```

或对于 BGP：

```
router bgp <asn>
 address-family ipv6 unicast
  redistribute static
```

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 自身的网络广告
- 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）指标类型重新分发到 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 邻居正在接收路由通告