OmniUPF Operations Guide

Table of Contents

Overview

OmniUPF (eBPF-based User Plane Function) is a high-performance 5G/LTE User Plane Function that provides carrier-grade packet forwarding, QoS enforcement, and traffic management for mobile networks. Built on Linux eBPF (extended Berkeley Packet Filter) technology and enhanced with comprehensive management capabilities, OmniUPF delivers the core packet processing infrastructure required for 5G SA, 5G NSA, and LTE networks.

What is a User Plane Function?

The User Plane Function (UPF) is the 3GPP-standardized network element responsible for packet processing and forwarding in 5G and LTE networks. It provides:

- **High-speed packet forwarding** between mobile devices and data networks
- **Quality of Service (QoS)** enforcement for different traffic types
- **Traffic detection and routing** based on packet filters and rules
- **Usage reporting** for charging and analytics
- **Packet buffering** for mobility and session management scenarios
- **Lawful intercept** support for regulatory compliance

OmniUPF implements the full UPF functionality defined in 3GPP TS 23.501 (5G) and TS 23.401 (LTE), providing a complete, production-ready user plane solution using Linux kernel eBPF technology for maximum performance.

OmniUPF Key Capabilities

Packet Processing:

- Full 3GPP-compliant user plane packet processing
- eBPF-based datapath for kernel-level performance
- GTP-U (GPRS Tunneling Protocol) encapsulation and decapsulation
- IPv4 and IPv6 support for both access and data networks
- XDP (eXpress Data Path) for ultra-low latency processing
- Multi-threaded packet processing

QoS and Traffic Management:

- QoS Enforcement Rules (QER) for bandwidth management
- Packet Detection Rules (PDR) for traffic classification
- Forwarding Action Rules (FAR) for routing decisions
- Service Data Flow (SDF) filtering for application-specific routing
- Usage Reporting Rules (URR) for volume tracking and charging

Control and Management:

- PFCP (Packet Forwarding Control Protocol) interface to SMF/PGW-C
- RESTful API for monitoring and diagnostics
- Real-time statistics and metrics
- eBPF map capacity monitoring
- Web-based control panel

Performance Features:

- Zero-copy packet processing via eBPF
- Kernel-level packet forwarding (no userspace overhead)
- Multi-core scalability
- Offload-capable for hardware acceleration
- Optimized for cloud-native deployments

For detailed control panel usage, see Web UI Operations.

Understanding User Plane Architecture

OmniUPF is a unified user plane solution providing carrier-grade packet forwarding for 5G Standalone (SA), 5G NSA, and 4G LTE/EPC networks. **OmniUPF is a single product** that can simultaneously function as:

- **UPF (User Plane Function)** - 5G/NSA user plane (controlled by OmniSMF via N4/PFCP)
- **PGW-U (PDN Gateway User Plane)** - 4G EPC gateway to external networks (controlled by OmniPGW-C via Sxc/PFCP)
- **SGW-U (Serving Gateway User Plane)** - 4G EPC serving gateway (controlled by OmniSGW-C via Sxb/PFCP)

OmniUPF can operate in **any combination** of these modes:

- **UPF-only**: Pure 5G deployment
- **PGW-U + SGW-U**: Combined 4G gateway (typical EPC deployment)
- **UPF + PGW-U + SGW-U**: Simultaneous 4G and 5G support (migration scenario)

All modes use the same eBPF-based packet processing engine and PFCP protocol, providing consistent high performance whether operating as UPF, PGW-U, SGW-U, or all three simultaneously.

5G Network Architecture (SA Mode)

The OmniUPF solution sits at the data plane of 5G networks, providing the high-speed packet forwarding layer that connects mobile devices to data networks and services.

4G LTE/EPC Network Architecture

OmniUPF also supports 4G LTE and EPC (Evolved Packet Core) deployments, functioning as either OmniPGW-U or OmniSGW-U depending on the network architecture.

Combined PGW-U/SGW-U Mode (Typical 4G Deployment)

In this mode, OmniUPF acts as both SGW-U and PGW-U, controlled by separate control plane functions.

Separated SGW-U and PGW-U Mode (Roaming/Multi-Site)

In roaming or multi-site deployments, two separate OmniUPF instances can be deployed - one as SGW-U and one as PGW-U.

N9 Loopback Mode (Single Instance SGWU+PGWU)

For simplified deployments, OmniUPF can run **both SGWU and PGWU roles on a single instance** with N9 loopback processing entirely in eBPF.

Key Features:

- � **Sub-microsecond N9 latency** - Processed entirely in eBPF, never touches network
- � **40-50% CPU reduction** - Single XDP pass vs. two separate instances
- � **Simplified deployment** - One instance, one configuration file
- � **Automatic detection** - When `n3_address = n9_address`, loopback is enabled
- � **Full 3GPP compliance** - Standard PFCP and GTP-U protocols

Configuration:

```
# OmniUPF config.yml
interface_name: [eth0]
n3_address: "10.0.1.10"      # S1-U interface IP
n9_address: "10.0.1.10"      # Same IP enables N9 loopback
pfcp_address: ":8805"         # Both SGWU-C and PGWU-C connect here
```

When to use:

- Edge computing deployments (minimize latency)
- Cost-constrained environments (single server)
- Lab/testing (simplified setup)
- Small to medium deployments (< 100K subscribers)

When NOT to use:

- Geographic redundancy required (SGWU and PGWU in different locations)
- Regulatory mandates for separated gateways
- Massive scale (> 1M subscribers)

For complete details, configuration examples, troubleshooting, and performance metrics, see **N9 Loopback Operations Guide**.

How User Plane Functions Work in the Network

The user plane function (OmniUPF, OmniPGW-U, or OmniSGW-U) operates as the forwarding plane controlled by the respective control plane:

1. **Session Establishment**
    - **5G**: OmniSMF establishes PFCP association via N4 interface with OmniUPF
    - **4G**: OmniPGW-C or OmniSGW-C establishes PFCP association via Sxb/Sxc with OmniPGW-U/OmniSGW-U
    - Control plane creates PFCP sessions for each UE PDU session (5G) or PDP context (4G)
    - User plane receives PDR, FAR, QER, and URR rules via PFCP
    - eBPF maps are populated with forwarding rules
2. **Uplink Packet Processing** (UE → Data Network)
    - **5G**: Packets arrive on N3 interface from gNB with GTP-U encapsulation
    - **4G**: Packets arrive on S1-U interface (SGW-U) or S5/S8 interface (PGW-U) from eNodeB with GTP-U encapsulation
    - User plane matches packets against uplink PDRs based on TEID

- eBPF program applies QER (rate limiting, marking)
- FAR determines forwarding action (forward, drop, buffer, duplicate)
- GTP-U tunnel removed, packets forwarded to N6 (5G) or SGi (4G) interface
- URR tracks packet and byte counts for charging

3. **Downlink Packet Processing** (Data Network → UE)

   - **5G**: Packets arrive on N6 interface as native IP
   - **4G**: Packets arrive on SGi interface as native IP
   - User plane matches packets against downlink PDRs based on UE IP address
   - SDF filters may further classify traffic by port, protocol, or application
   - FAR determines GTP-U tunnel and forwarding parameters
   - GTP-U encapsulation added with appropriate TEID
   - **5G**: Packets forwarded to N3 interface toward gNB
   - **4G**: Packets forwarded to S1-U (SGW-U) or S5/S8 (PGW-U) toward eNodeB

4. **Mobility and Handover**

   - **5G**: OmniSMF updates PDR/FAR rules during handover scenarios
   - **4G**: OmniSGW-C/OmniPGW-C updates rules during inter-eNodeB handover or TAU (Tracking Area Update)
   - User plane may buffer packets during path switch
   - Seamless transition between base stations without packet loss

Integration with Control Plane (4G and 5G)

OmniUPF integrates with both 5G and 4G control plane functions via standard 3GPP interfaces:

5G Interfaces

Interface	From → To	Purpose	3GPP Spec
N4	OmniSMF ↔ OmniUPF	PFCP session establishment, modification, deletion	TS 29.244
N3	gNB → OmniUPF	User plane traffic from RAN (GTP-U)	TS 29.281
N6	OmniUPF → Data Network	User plane traffic to DN (native IP)	TS 23.501
N9	OmniUPF ↔ OmniUPF	Inter-UPF communication for roaming/ edge	TS 23.501

4G/EPC Interfaces

Interface	From → To	Purpose	3GPP Spec
Sxb	OmniSGW-C ↔ OmniUPF (SGW-U mode)	PFCP session control for serving gateway	TS 29.244
Sxc	OmniPGW-C ↔ OmniUPF (PGW-U mode)	PFCP session control for PDN gateway	TS 29.244
S1-U	eNodeB → OmniUPF (SGW-U mode)	User plane traffic from RAN (GTP-U)	TS 29.281
S5/S8	OmniUPF (SGW-U) ↔ OmniUPF (PGW-U)	Inter-gateway user plane (GTP-U)	TS 29.281
SGi	OmniUPF (PGW-U mode) → PDN	User plane traffic to data network (native IP)	TS 23.401

Note: All PFCP interfaces (N4, Sxb, Sxc) use the same PFCP protocol defined in TS 29.244. The interface names differ but the protocol and message formats are identical.

For PFCP session management, see PFCP Operations.

UPF Components

eBPF Datapath

The **eBPF datapath** is the core packet processing engine that runs in the Linux kernel for maximum performance.

Core Functions:

- **GTP-U Processing**: Encapsulation and decapsulation of GTP-U tunnels
- **Packet Classification**: Matching packets against PDR rules using TEID, UE IP, or SDF filters
- **QoS Enforcement**: Apply rate limiting and packet marking per QER rules
- **Forwarding Decisions**: Execute FAR actions (forward, drop, buffer, duplicate, notify)
- **Usage Tracking**: Increment URR counters for volume-based charging

eBPF Maps: The datapath uses eBPF maps (hash tables in kernel memory) for rule storage:

Map Name	Purpose	Key	Value
`uplink_pdr_map`	Uplink PDRs	TEID (32-bit)	PDR info (FAR ID, QER ID, URR IDs)
`downlink_pdr_map`	Downlink PDRs (IPv4)	UE IP address	PDR info

Map Name	Purpose	Key	Value
`downlink_pdr_map_ip6`	Downlink PDRs (IPv6)	UE IPv6 address	PDR info
`far_map`	Forwarding rules	FAR ID	Forwarding parameters (action, tunnel info)
`qer_map`	QoS rules	QER ID	QoS parameters (MBR, GBR, marking)
`urr_map`	Usage tracking	URR ID	Volume counters (uplink, downlink, total)
`sdf_filter_map`	SDF filters	PDR ID	Application filters (ports, protocols)

Performance Characteristics:

- **Zero-copy**: Packets processed entirely in kernel space
- **XDP support**: Attach at network driver level for sub-microsecond latency
- **Multi-core**: Scales across CPU cores with per-CPU map support
- **Capacity**: Millions of PDRs/FARs in eBPF maps (limited by kernel memory)

For capacity monitoring, see Capacity Management.

PFCP Interface Handler

The **PFCP interface** implements 3GPP TS 29.244 for communication with SMF or PGW-C.

Core Functions:

- **Association Management**: PFCP heartbeat and association setup/release
- **Session Lifecycle**: Create, modify, and delete PFCP sessions
- **Rule Installation**: Translate PFCP IEs into eBPF map entries
- **Event Reporting**: Notify SMF of usage thresholds, errors, or session events

PFCP Message Support:

Message Type	Direction	Purpose
Association Setup	SMF → UPF	Establish PFCP control association
Association Release	SMF → UPF	Tear down PFCP association
Heartbeat	Bidirectional	Keep association alive
Session Establishment	SMF → UPF	Create new PDU session with PDR/FAR/QER/URR
Session Modification	SMF → UPF	Update rules for mobility, QoS changes
Session Deletion	SMF → UPF	Remove session and all associated rules
Session Report	UPF → SMF	Report usage, errors, or events

Information Elements (IE) Supported:

- Create PDR, FAR, QER, URR
- Update PDR, FAR, QER, URR
- Remove PDR, FAR, QER, URR
- Packet Detection Information (UE IP, F-TEID, SDF filter)
- Forwarding Parameters (network instance, outer header creation)
- QoS Parameters (MBR, GBR, QFI)
- Usage Report Triggers (volume threshold, time threshold)

For detailed PFCP operations, see **PFCP Operations Guide**.

REST API Server

The **REST API** provides programmatic access to UPF state and operations.

Core Functions:

- **Session Monitoring**: Query active PFCP sessions and associations
- **Rule Inspection**: View PDR, FAR, QER, URR configurations
- **Statistics**: Retrieve packet counters, route stats, XDP stats
- **Buffer Management**: View and control packet buffers
- **Map Information**: Monitor eBPF map usage and capacity

API Endpoints: (34 total endpoints)

Category	Endpoints	Description
Health	`/health`	Health check and status
Config	`/config`	UPF configuration
Sessions	`/pfcp_sessions`, `/pfcp_associations`	PFCP session/ association data
PDRs	`/uplink_pdr_map`, `/downlink_pdr_map`, `/downlink_pdr_map_ip6`, `/uplink_pdr_map_ip6`	Packet detection rules
FARs	`/far_map`	Forwarding action rules
QERs	`/qer_map`	QoS enforcement rules
URRs	`/urr_map`	Usage reporting rules
Buffers	`/buffer`	Packet buffer status and control
Statistics	`/packet_stats`, `/route_stats`, `/xdp_stats`, `/n3n6_stats`	Performance metrics
Capacity	`/map_info`	eBPF map capacity and

Category	Endpoints	Description
		usage
Dataplane	`/dataplane_config`	N3/N9 interface addresses

For API details and usage, see **PFCP Operations Guide** and **Monitoring Guide**.

Web Control Panel

The **Web Control Panel** provides a real-time dashboard for UPF monitoring and management.

Features:

- **Sessions View**: Browse active PFCP sessions with UE IP, TEID, and rule counts
- **Rules Management**: View and manage PDRs, FARs, QERs, and URRs across all sessions
- **Buffer Monitoring**: Track buffered packets and control buffering per FAR
- **Statistics Dashboard**: Real-time packet, route, XDP, and N3/N6 interface statistics
- **Capacity Monitoring**: eBPF map usage with color-coded capacity indicators
- **Configuration View**: Display UPF configuration and dataplane addresses
- **Logs Viewer**: Live log streaming for troubleshooting

For detailed UI operations, see **Web UI Operations Guide**.

PFCP Protocol and SMF Integration

PFCP Association

Before sessions can be created, the SMF must establish a PFCP association with the UPF.

Association Lifecycle:

Key Points:

- Each SMF establishes one association with the UPF
- UPF tracks association by Node ID (FQDN or IP address)
- Heartbeat messages maintain association liveness
- All sessions under an association are deleted if association is released

For viewing associations, see Sessions View.

PFCP Session Creation

When a UE establishes a PDU session (5G) or PDP context (LTE), the SMF creates a PFCP session at the UPF.

Session Establishment Flow:

Typical Session Contents:

- **Uplink PDR**: Match on N3 TEID, forward via FAR to N6
- **Downlink PDR**: Match on UE IP address, forward via FAR to N3 with GTP-U encapsulation
- **FAR**: Forwarding parameters (outer header creation, network instance)
- **QER**: QoS limits (MBR, GBR) and packet marking (QFI)
- **URR**: Volume reporting for charging (optional)

For session monitoring, see PFCP Operations.

PFCP Session Modification

SMF can modify sessions for mobility events (handover), QoS changes, or service updates.

Common Modification Scenarios:

1. **Handover (N2-based)**
    - Update uplink FAR with new gNB tunnel endpoint (F-TEID)
    - Optionally buffer packets during path switch
    - Flush buffer to new path when ready
2. **QoS Change**
    - Update QER with new MBR/GBR values
    - May add/remove SDF filters in PDR for application-specific QoS
3. **Service Update**
    - Add new PDRs for additional traffic flows
    - Modify FARs for routing changes

Session Modification Flow:

For rule management, see Rules Management Guide.

PFCP Session Deletion

When a PDU session is released, SMF deletes the PFCP session at UPF.

Session Deletion Flow:

Cleanup Performed:

- All PDRs removed (uplink and downlink)
- All FARs, QERs, URRs removed
- Packet buffers cleared
- Final usage report sent to SMF for charging

Common Operations

OmniUPF provides comprehensive operational capabilities through its web-based control panel and REST API. This section covers common operational tasks and their significance.

Session Monitoring

Understanding PFCP Sessions:

PFCP sessions represent active UE PDU sessions (5G) or PDP contexts (LTE). Each session contains:

- Local and remote SEIDs (Session Endpoint Identifiers)
- PDRs for packet classification
- FARs for forwarding decisions
- QERs for QoS enforcement (optional)
- URRs for usage tracking (optional)

Key Session Operations:

- **View all sessions** with UE IP addresses, TEIDs, and rule counts
- **Filter sessions** by IP address or TEID
- **Inspect session details** including full PDR/FAR/QER/URR configurations
- **Monitor session counts** per PFCP association

For detailed session procedures, see Sessions View.

Rule Management

Packet Detection Rules (PDR):

PDRs determine which packets match specific traffic flows. Operators can:

- **View uplink PDRs** keyed by TEID from N3 interface
- **View downlink PDRs** keyed by UE IP address (IPv4 and IPv6)
- **Inspect SDF filters** for application-specific classification
- **Monitor PDR counts** and capacity usage

Forwarding Action Rules (FAR):

FARs define what to do with matched packets. Operators can:

- **View FAR actions** (FORWARD, DROP, BUFFER, DUPLICATE, NOTIFY)
- **Inspect forwarding parameters** (outer header creation, destination)
- **Monitor buffering status** per FAR
- **Toggle buffering** for specific FARs during troubleshooting

QoS Enforcement Rules (QER):

QERs apply bandwidth limits and packet marking. Operators can:

- **View QoS parameters** (MBR, GBR, packet delay budget)
- **Monitor active QERs** per session
- **Inspect QFI markings** for 5G QoS flows

Usage Reporting Rules (URR):

URRs track data volumes for charging. Operators can:

- **View volume counters** (uplink, downlink, total bytes)
- **Monitor usage thresholds** and reporting triggers
- **Inspect active URRs** across all sessions

For rule operations, see Rules Management Guide.

Packet Buffering

Why Buffering is Critical for UPF

Packet buffering is one of the most important functions of a UPF because it prevents packet loss during mobility events and session reconfigurations. Without buffering, mobile users would experience dropped connections, interrupted downloads, and failed real-time communications every time they move between cell towers or when network conditions change.

The Problem: Packet Loss During Mobility

In mobile networks, users are constantly moving. When a device moves from one cell tower to another (handover), or when the network needs to reconfigure the data path, there's a critical window where packets are in flight but the new path isn't ready yet:

Without buffering: Packets arriving during this critical window would be **dropped**, causing:

- **TCP connections to stall** or reset (web browsing, downloads interrupted)
- **Video calls to freeze** or drop (Zoom, Teams, WhatsApp calls fail)
- **Gaming sessions to disconnect** (online gaming, real-time apps fail)
- **VoIP calls to have gaps** or drop entirely (phone calls interrupted)
- **Downloads to fail** and need to restart

With buffering: OmniUPF temporarily holds packets until the new path is established, then forwards them seamlessly. The user experiences **zero interruption**.

When Buffering Happens

OmniUPF buffers packets in these critical scenarios:

1. N2-Based Handover (5G) / X2-Based Handover (4G)

When a UE moves between cell towers:

Timeline:

- **T+0ms**: Old path still active
- **T+10ms**: SMF tells UPF to buffer (old path closing, new path not ready)
- **T+10-50ms**: **Critical buffering window** - packets arrive but can't be forwarded
- **T+50ms**: New path ready, SMF tells UPF to forward
- **T+50ms+**: UPF flushes buffered packets to new path, then forwards new packets normally

Without buffering: ~40ms of packets (potentially thousands) would be **lost**.
With buffering: Zero packet loss, seamless handover.

---

2. Session Modification (QoS Change, Path Update)

When the network needs to change session parameters:

- **QoS upgrade/downgrade**: User moves from 4G to 5G coverage (NSA mode)
- **Policy change**: Enterprise user enters corporate campus (traffic steering changes)
- **Network optimization**: Core network reroutes traffic to closer UPF (ULCL update)

During the modification, the control plane may need to update multiple rules atomically. Buffering ensures packets aren't forwarded with partial/inconsistent rule sets.

---

3. Downlink Data Notification (Idle Mode Recovery)

When a UE is in idle mode (screen off, battery saving) and downlink data arrives:

Without buffering: The initial packet that triggered the notification would be **lost**, requiring the sender to retransmit (adds latency). **With buffering**: The packet that woke up the UE is delivered immediately when the UE reconnects.

---

4. Inter-RAT Handover (4G ↔ 5G)

When a UE moves between 4G and 5G coverage:

- Architecture changes (eNodeB ↔ gNB)
- Tunnel endpoints change (different TEID allocation)
- Buffering ensures smooth transition between RAT types

How Buffering Works in OmniUPF

Technical Mechanism:

OmniUPF uses a **two-stage buffering architecture**:

1. **eBPF Stage (Kernel)**: Detects packets requiring buffering based on FAR action flags
2. **Userspace Stage**: Stores and manages buffered packets in memory

Buffering Process:

Key Details:

- **Buffer Port**: UDP port 22152 (packets sent from eBPF to userspace)
- **Encapsulation**: Packets wrapped in GTP-U with FAR ID as TEID
- **Storage**: In-memory per-FAR buffers with metadata (timestamp, direction, packet size)
- **Limits**:
  - Per-FAR limit: 10,000 packets (default)
  - Global limit: 100,000 packets across all FARs
  - TTL: 30 seconds (default) - packets older than TTL are discarded
- **Cleanup**: Background process removes expired packets every 60 seconds

Buffer Lifecycle:

1. **Buffering Enabled**: SMF sets FAR action BUFF=1 (bit 2) via PFCP Session Modification
2. **Packets Buffered**: eBPF detects BUFF flag, encapsulates packets, sends to port 22152
3. **Userspace Storage**: Buffer manager stores packets with FAR ID, timestamp, direction
4. **Buffering Disabled**: SMF sets FAR action FORW=1, BUFF=0 with new forwarding parameters
5. **Flush Buffer**: Userspace replays buffered packets using new FAR rules (new tunnel endpoint)
6. **Resume Normal**: New packets forwarded immediately via new path

---

Why This Matters for User Experience

Real-World Impact:

Scenario	Without Buffering	With Buffering
Video Call During Handover	Call freezes for 1-2 seconds, may drop	Seamless, no interruption
File Download at Cell Edge	Download fails, must restart	Download continues uninterrupted
Online Gaming While Moving	Connection drops, kicked from game	Smooth gameplay, no disconnects
VoIP Call in Car	Call drops every handover	Crystal clear, no drops
Streaming Video on Train	Video buffers, quality drops	Smooth playback
Mobile Hotspot for Laptop	SSH session drops, video call fails	All connections maintained

Network Operator Benefits:

- **Reduced Call Drop Rate (CDR)**: Critical KPI for network quality
- **Higher Customer Satisfaction**: Users don't notice handovers
- **Lower Support Costs**: Fewer complaints about dropped connections
- **Competitive Advantage**: "Best network for coverage" marketing

---

Buffer Management Operations

Operators can monitor and control buffering via the Web UI and API:

Monitoring:

- **View buffered packets** per FAR ID (count, bytes, age)
- **Track buffer usage** against limits (per-FAR, global)
- **Alert on buffer overflow** or excessive buffering duration

- **Identify stuck buffers** (packets buffered > TTL threshold)

Control Operations:

- **Flush buffers**: Manually trigger buffer replay (troubleshooting)
- **Clear buffers**: Discard buffered packets (clean up stuck buffers)
- **Adjust TTL**: Change packet expiration time
- **Modify limits**: Increase per-FAR or global buffer capacity

Troubleshooting:

- **Buffer not flushing**: Check if SMF sent FAR update to disable buffering
- **Buffer overflow**: Increase limits or investigate why buffering duration is excessive
- **Old packets in buffer**: TTL may be too high, or FAR update delayed
- **Excessive buffering**: May indicate mobility issues or SMF problems

For detailed buffer operations, see Buffer Management Guide.

Buffer Configuration

Configure buffering behavior in `config.yml`:

```yaml
# Buffer settings
buffer_port: 22152                # UDP port for buffered packets (default)
buffer_max_packets: 10000         # Max packets per FAR (prevent memory exhaustion)
buffer_max_total: 100000          # Max total packets across all FARs
buffer_packet_ttl: 30             # TTL in seconds (discard old packets)
buffer_cleanup_interval: 60       # Cleanup interval in seconds
```

Recommendations:

- **High-mobility networks** (highways, trains): Increase `buffer_max_packets` to 20,000+
- **Dense urban areas** (frequent handovers): Decrease `buffer_packet_ttl` to 15s
- **Low-latency applications**: Set `buffer_packet_ttl` to 10s to prevent stale data
- **IoT networks**: Decrease limits (IoT devices generate less traffic during handover)

For complete configuration options, see Configuration Guide.

Statistics and Monitoring

Packet Statistics:

Real-time packet processing metrics including:

- **RX packets**: Total received from all interfaces
- **TX packets**: Total transmitted to all interfaces
- **Dropped packets**: Packets discarded due to errors or policy
- **GTP-U packets**: Tunneled packet counts

Route Statistics:

Per-route forwarding metrics:

- **Route hits**: Packets matched by each route
- **Forwarding counts**: Success/failure per destination
- **Error counters**: Invalid TEIDs, unknown UE IPs

XDP Statistics:

eXpress Data Path performance metrics:

- **XDP processed**: Packets handled at XDP layer
- **XDP passed**: Packets sent to network stack
- **XDP dropped**: Packets dropped at XDP layer
- **XDP aborted**: Processing errors

N3/N6 Interface Statistics:

Per-interface traffic counters:

- **N3 RX/TX**: Traffic to/from RAN (gNB/eNodeB)
- **N6 RX/TX**: Traffic to/from data network
- **Total packet counts**: Aggregate interface statistics

For monitoring details, see Monitoring Guide.

Capacity Management

eBPF Map Capacity Monitoring:

UPF performance depends on eBPF map capacity. Operators can:

- **Monitor map usage** with real-time percentage indicators
- **View capacity limits** for each eBPF map
- **Color-coded alerts**:

    - Green (<50%): Normal
    - Yellow (50-70%): Caution
    - Amber (70-90%): Warning
    - Red (>90%): Critical

Critical Maps to Monitor:

- `uplink_pdr_map`: Uplink traffic classification
- `downlink_pdr_map`: Downlink IPv4 traffic classification
- `far_map`: Forwarding rules
- `qer_map`: QoS rules
- `urr_map`: Usage tracking

Capacity Planning:

- Each PDR consumes one map entry (key size + value size)
- Map capacity is configured at UPF startup (kernel memory limit)
- Exceeding capacity causes session establishment failures

For capacity monitoring, see Capacity Management.

Configuration Management

UPF Configuration:

View and verify UPF operational parameters:

- **N3 Interface**: IP address for RAN connectivity (GTP-U)
- **N6 Interface**: IP address for data network connectivity
- **N9 Interface**: IP address for inter-UPF communication (optional)
- **PFCP Interface**: IP address for SMF connectivity
- **API Port**: REST API listening port
- **Metrics Endpoint**: Prometheus metrics port

Dataplane Configuration:

Active eBPF datapath parameters:

- **Active N3 address**: Runtime N3 interface binding
- **Active N9 address**: Runtime N9 interface binding (if enabled)

For configuration viewing, see Configuration View.

Troubleshooting

This section covers common operational issues and their resolution strategies.

Session Establishment Failures

Symptoms: PFCP sessions fail to create, UE cannot establish data connectivity

Common Root Causes:

1. **PFCP Association Not Established**

    - Verify SMF can reach UPF PFCP interface (port 8805)
    - Check PFCP association status in Sessions view
    - Verify Node ID configuration matches between SMF and UPF

2. **eBPF Map Capacity Exhausted**

    - Check Capacity view for red (>90%) map usage
    - Increase eBPF map sizes in UPF configuration
    - Delete stale sessions if map is full

3. **Invalid PDR/FAR Configuration**

    - Verify UE IP address is unique and valid
    - Check TEID allocation doesn't conflict
    - Ensure FAR references valid network instances

4. **Interface Configuration Issues**

    - Verify N3 interface IP is reachable from gNB
    - Check routing tables for N6 connectivity to data network
    - Confirm GTP-U traffic is not blocked by firewall

For detailed troubleshooting, see Troubleshooting Guide.

Packet Loss or Forwarding Issues

Symptoms: UE has connectivity but experiences packet loss or no traffic flow

Common Root Causes:

1. **PDR Misconfiguration**

    - Verify uplink PDR TEID matches gNB-assigned TEID
    - Check downlink PDR UE IP matches assigned IP
    - Inspect SDF filters for overly restrictive rules

2. **FAR Action Issues**

    - Verify FAR action is FORWARD (not DROP or BUFFER)

    - Check outer header creation parameters for GTP-U
    - Ensure destination endpoint is correct

3. **QoS Limits Exceeded**

    - Check QER MBR (Maximum Bit Rate) settings
    - Verify GBR (Guaranteed Bit Rate) allocation
    - Monitor packet drops due to rate limiting

4. **Interface MTU Issues**

    - Verify GTP-U overhead (40-50 bytes) doesn't cause fragmentation
    - Check N3/N6 interface MTU configuration
    - Monitor for ICMP fragmentation needed messages

Buffer-Related Issues

Symptoms: Packets buffered indefinitely, buffer overflow

Common Root Causes:

1. **Buffering Not Disabled After Handover**

    - Check FAR buffering flag (bit 2)
    - Verify SMF sent Session Modification to disable buffering
    - Manually disable buffering via control panel if stuck

2. **Buffer TTL Expiration**

    - Check packet age in buffer view
    - Verify buffer TTL configuration (default may be too long)
    - Clear expired buffers manually

3. **Buffer Capacity Exhausted**

    - Monitor total buffer usage and per-FAR limits
    - Check for misconfigured rules causing excessive buffering
    - Adjust max_per_far and max_total buffer limits

For buffer troubleshooting, see Buffer Operations.

Statistics Anomalies

Symptoms: Unexpected packet counters, missing statistics

Common Root Causes:

1. **Counter Overflow**

    - eBPF maps use 64-bit counters (should not overflow)
    - Check for counter reset events in logs
    - Verify URR reporting is functioning

2. **Route Statistics Not Updating**

    - Verify eBPF program is attached to interfaces
    - Check kernel version supports required eBPF features
    - Review XDP statistics for processing errors

3. **Interface Statistics Mismatch**

    - Compare N3/N6 stats with kernel interface counters
    - Check for traffic bypassing eBPF (e.g., local routing)
    - Verify all traffic flows through XDP hooks

Performance Degradation

Symptoms: High latency, low throughput, CPU saturation

Diagnosis:

1. **Monitor XDP Statistics**: Check for XDP drops or aborts
2. **Check eBPF Map Access Time**: Hash lookups should be sub-microsecond
3. **Review CPU Utilization**: eBPF should distribute across cores
4. **Analyze Network Interface**: Verify NIC supports XDP offload

Scalability Considerations:

- **XDP Performance**: 10M+ packets per second per core
- **PDR Capacity**: Millions of PDRs limited only by kernel memory
- **Session Count**: Thousands of concurrent sessions per UPF instance
- **Throughput**: Multi-gigabit throughput with proper NIC offload

For performance tuning, see Architecture Guide.

Additional Documentation

Component-Specific Operations Guides

For detailed operations and troubleshooting for each UPF component:

Configuration Guide

Complete configuration reference including:

- Configuration parameters (YAML, environment variables, CLI)
- Operating modes (UPF/PGW-U/SGW-U)
- XDP attachment modes overview
- Hypervisor compatibility (Proxmox, VMware, KVM, Hyper-V, VirtualBox)
- NIC compatibility and XDP driver support
- Configuration examples for different scenarios
- Map sizing and capacity planning

XDP Modes Guide

Detailed XDP configuration and optimization including:

- XDP attachment modes explained (generic/native/offload)
- Performance comparison and benchmarks
- Step-by-step Proxmox VE native XDP setup
- Multi-queue configuration for optimal performance
- VMware ESXi, KVM, and Hyper-V XDP setup
- XDP verification and troubleshooting
- Hardware selection for XDP performance

Architecture Guide

Deep technical dive including:

- eBPF technology foundation and program lifecycle
- XDP packet processing pipeline with tail calls
- PFCP protocol implementation
- Buffering architecture (GTP-U encapsulation to port 22152)
- QoS sliding window rate limiting (5ms window)
- Performance characteristics (3.5µs latency, 10 Mpps/core)

Rules Management Guide

PFCP rules reference including:

- Packet Detection Rules (PDR) - Traffic classification
- Forwarding Action Rules (FAR) - Routing decisions with action flags
- QoS Enforcement Rules (QER) - Bandwidth management (MBR/GBR)
- Usage Reporting Rules (URR) - Volume tracking and reporting
- Uplink and downlink packet flow diagrams
- Rule processing logic and precedence

Monitoring Guide

Statistics and capacity management including:

- N3/N6 interface statistics and traffic distribution
- XDP processing statistics (pass/drop/redirect/abort)
- eBPF map capacity monitoring with color-coded zones
- Performance metrics (packet rate, throughput, drop rate)
- Capacity planning formulas and session estimation
- Alerting thresholds and best practices

Web UI Operations Guide

Control panel usage including:

- Dashboard overview and navigation
- Sessions monitoring (healthy/unhealthy states)
- Rules inspection (PDR, FAR, QER, URR details)
- Buffer monitoring and packet buffering state
- Real-time statistics dashboard
- eBPF map capacity visualization
- Configuration viewing

API Documentation

Complete REST API reference including:

- OpenAPI/Swagger interactive documentation
- PFCP sessions and associations endpoints
- Packet Detection Rules (PDR) - IPv4 and IPv6
- Forwarding Action Rules (FAR)
- QoS Enforcement Rules (QER)
- Usage Reporting Rules (URR)
- Packet buffer management
- Statistics and monitoring endpoints
- Route management and FRR integration
- eBPF map information
- Configuration management
- Authentication and security guidelines
- Common API workflows and examples

UE Route Management Guide

FRR routing integration including:

- FRR (Free Range Routing) overview and architecture
- UE route synchronization lifecycle
- Automatic route sync to routing daemon

- Route advertisement via OSPF and BGP
- OSPF neighbor monitoring
- OSPF External LSA database verification
- BGP peer session management
- Web UI route monitoring interface
- Manual route sync operations
- Mermaid diagrams for route flow and architecture

Troubleshooting Guide

Comprehensive problem diagnosis including:

- Quick diagnostic checklist and tools
- Installation and configuration issues
- PFCP association failures
- Packet processing problems
- XDP and eBPF errors
- Performance degradation
- Hypervisor-specific issues (Proxmox, VMware, VirtualBox)
- NIC and driver problems
- Step-by-step resolution procedures

Documentation by Use Case

Installing and Configuring OmniUPF

1. Start with this guide for overview
2. Configuration Guide for setup parameters
3. Web UI Guide to access control panel

Deploying SGWU+PGWU on Single Instance (N9 Loopback)

1. N9 Loopback Operations Guide - Complete guide for combined SGWU+PGWU deployment
2. N9 Loopback - Configuration - Network and PFCP setup
3. N9 Loopback - Monitoring - Verify loopback is active
4. N9 Loopback - Troubleshooting - Common issues and solutions

Deploying on Proxmox

1. XDP Modes Guide - Proxmox Native XDP Setup - **Start here for performance**
2. Configuration Guide - Hypervisor Compatibility
3. Configuration Guide - Proxmox SR-IOV Setup
4. Troubleshooting - Proxmox Issues

Optimizing Performance

1. XDP Modes Guide - **Enable native XDP for 5-10x performance boost**
2. Architecture Guide - Performance Optimization
3. Configuration Guide - XDP Modes
4. Monitoring Guide - Performance Metrics
5. Troubleshooting - Performance Issues

Understanding Packet Processing

1. Architecture Guide - Packet Processing Pipeline
2. Rules Management Guide
3. Monitoring Guide - Statistics

Planning Capacity

1. Configuration Guide - Map Sizing
2. Monitoring Guide - Capacity Planning
3. Monitoring Guide - Session Capacity Estimation

Managing UE Routes and FRR Integration

1. UE Route Management Guide - Complete routing integration guide
2. API Documentation - Route Management - Route API endpoints
3. Web UI Guide - Routes page operations
4. UE Route Management - FRR Verification - OSPF LSA verification

Using the REST API

1. API Documentation - Complete API reference
2. API Documentation - Swagger UI - Interactive API explorer
3. API Documentation - Common Workflows - API usage examples
4. Web UI Guide - Web interface as API client example

Troubleshooting Issues

1. Troubleshooting Guide - Start here
2. Monitoring Guide - Check statistics and capacity
3. Web UI Guide - Use control panel diagnostics

Quick Reference

Common API Endpoints

OmniUPF provides a REST API for monitoring and management:

```
# Status and health
GET http://localhost:8080/api/v1/upf_status

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

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

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

# Capacity monitoring
GET http://localhost:8080/api/v1/map_info

# Buffer statistics
GET http://localhost:8080/api/v1/upf_buffer_info
```

For complete API documentation, access the Swagger UI at `http://<upf-ip>:8080/swagger/index.html`

Essential Configuration Parameters

```
# Network interfaces
interface_name: [eth0]          # Interfaces for N3/N6/N9 traffic
xdp_attach_mode: native         # generic|native|offload
n3_address: 10.100.50.233      # N3 interface IP
pfcp_address: :8805             # PFCP listen address
pfcp_node_id: 10.100.50.241    # PFCP Node ID

# Capacity
max_sessions: 100000            # Maximum concurrent sessions

# API and monitoring
api_address: :8080              # REST API port
metrics_address: :9090          # Prometheus metrics port
```

Important Monitoring Thresholds

- **eBPF Map Capacity < 70%**: Normal operation
- **eBPF Map Capacity 70-90%**: Plan capacity increase within 1 week
- **eBPF Map Capacity > 90%**: Critical - immediate action required
- **Packet Drop Rate < 0.1%**: Excellent
- **Packet Drop Rate 0.1-1%**: Good - minor issues
- **Packet Drop Rate > 5%**: Critical - investigate immediately
- **XDP Aborted > 0**: Critical issue with eBPF program

3GPP Standards Reference

OmniUPF implements the following 3GPP specifications:

Specification	Title	Relevance
TS 23.501	System architecture for the 5G System (5GS)	5G UPF architecture and interfaces
TS 23.401	General Packet Radio Service (GPRS) enhancements for E-UTRAN access	LTE UPF (PGW-U) architecture
TS 29.244	Interface between the Control Plane and the User Plane nodes (PFCP)	N4 PFCP protocol
TS 29.281	General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)	GTP-U encapsulation
TS 23.503	Policy and charging control framework for the 5G System (5GS)	QoS and charging
TS 29.212	Policy and Charging Control (PCC)	QoS enforcement

Glossary

5G Architecture Terms

- **3GPP**: 3rd Generation Partnership Project - Standards body for mobile telecommunications
- **AMF**: Access and Mobility Management Function - 5G core network element for access control
- **CHF**: Charging Function - 5G charging system
- **DN**: Data Network - External network (Internet, IMS, enterprise)
- **eNodeB**: Evolved Node B - LTE base station
- **F-TEID**: Fully Qualified Tunnel Endpoint Identifier - GTP-U tunnel ID with IP address
- **gNB**: Next Generation Node B - 5G base station
- **GTP-U**: GPRS Tunnelling Protocol User Plane - Tunneling protocol for user data
- **MBR**: Maximum Bit Rate - QoS parameter for maximum allowed bandwidth
- **GBR**: Guaranteed Bit Rate - QoS parameter for guaranteed minimum bandwidth
- **N3**: Interface between RAN and UPF (user plane traffic)
- **N4**: Interface between SMF and UPF (PFCP control)
- **N6**: Interface between UPF and Data Network (user plane traffic)
- **N9**: Interface between two UPFs (inter-UPF user plane traffic)
- **PCF**: Policy Control Function - 5G policy server
- **PDU**: Protocol Data Unit - Data session in 5G
- **PGW-C**: PDN Gateway Control Plane - LTE control plane equivalent to SMF
- **PGW-U**: PDN Gateway User Plane - LTE user plane (UPF equivalent)
- **QFI**: QoS Flow Identifier - 5G QoS flow marking

- **QoS**: Quality of Service - Traffic prioritization and bandwidth management
- **RAN**: Radio Access Network - Base station network (gNB/eNodeB)
- **SEID**: Session Endpoint Identifier - PFCP session ID
- **SMF**: Session Management Function - 5G core network element for session control
- **TEID**: Tunnel Endpoint Identifier - GTP-U tunnel ID
- **UE**: User Equipment - Mobile device
- **UPF**: User Plane Function - 5G packet forwarding network element

PFCP Protocol Terms

- **Association**: Control relationship between SMF and UPF
- **FAR**: Forwarding Action Rule - Determines packet forwarding behavior
- **IE**: Information Element - PFCP message component
- **Node ID**: UPF or SMF identifier (FQDN or IP address)
- **PDR**: Packet Detection Rule - Classifies packets into flows
- **PFCP**: Packet Forwarding Control Protocol - N4 control protocol
- **QER**: QoS Enforcement Rule - Applies bandwidth limits and marking
- **SDF**: Service Data Flow - Application-specific traffic filter
- **Session**: PFCP session representing UE PDU session or PDP context
- **URR**: Usage Reporting Rule - Tracks data volumes for charging

eBPF and Linux Kernel Terms

- **BPF**: Berkeley Packet Filter - Kernel packet filtering technology
- **eBPF**: Extended BPF - Programmable kernel data path
- **Hash Map**: eBPF key-value store for fast lookups
- **XDP**: eXpress Data Path - Kernel packet processing at driver level
- **Verifier**: Kernel component that validates eBPF programs for safety
- **Map**: eBPF data structure shared between kernel and userspace
- **Zero-copy**: Packet processing without copying to userspace

OmniUPF Product Terms

- **OmniUPF**: eBPF-based User Plane Function (this product)
- **Datapath**: Packet processing engine (eBPF programs)
- **Control Plane**: PFCP handler and session management
- **REST API**: HTTP API for monitoring and management
- **Web UI**: Browser-based control panel

OmniUPF Architecture Guide

Table of Contents

Overview

OmniUPF leverages eBPF (extended Berkeley Packet Filter) and XDP (eXpress Data Path) to achieve carrier-grade performance for 5G/LTE packet processing. By running packet processing logic directly in the Linux kernel, OmniUPF eliminates the overhead of userspace processing and achieves multi-gigabit throughput with microsecond latency.

Architecture Layers

Key Design Principles

Zero-Copy Processing:

- Packets processed entirely in kernel space
- No data copying between kernel and userspace
- Direct packet manipulation using XDP

Lock-Free Data Structures:

- eBPF maps use per-CPU hash tables
- Atomic operations for concurrent access
- No mutex/spinlock overhead

Hardware Offload Ready:

- XDP offload mode supports SmartNIC execution
- Compatible with network cards supporting XDP
- Fallback to driver-native or generic modes

eBPF Technology Foundation

What is eBPF?

eBPF (extended Berkeley Packet Filter) is a revolutionary Linux kernel technology that allows safe, sandboxed programs to run in kernel space without changing kernel source code or loading kernel modules.

Key Features:

- **Safety**: eBPF verifier ensures programs cannot crash the kernel
- **Performance**: Runs at native kernel speed (no interpretation overhead)
- **Flexibility**: Can be updated at runtime without kernel restart
- **Observability**: Built-in tracing and statistics

eBPF Program Lifecycle

eBPF Maps

eBPF maps are kernel data structures shared between eBPF programs and userspace.

Map Types Used in OmniUPF:

Map Type	Description	Use Case
`BPF_MAP_TYPE_HASH`	Hash table with key-value pairs	PDR lookup by TEID or UE IP
`BPF_MAP_TYPE_ARRAY`	Array indexed by integer	QER, FAR, URR lookup by ID
`BPF_MAP_TYPE_PERCPU_HASH`	Per-CPU hash table (lock-free)	High-performance PDR lookups
`BPF_MAP_TYPE_LRU_HASH`	LRU (Least Recently Used) hash	Automatic eviction of old entries

Map Operations:

- **Lookup**: O(1) hash lookup (sub-microsecond)
- **Update**: Atomic updates from userspace
- **Delete**: Immediate removal of entries
- **Iterate**: Batch operations for map dumps

XDP Datapath

XDP Overview

XDP (eXpress Data Path) is a Linux kernel hook that allows eBPF programs to process packets at the earliest possible point—right after the network driver receives them, before the kernel networking stack.

XDP Attach Modes

OmniUPF supports three XDP attach modes, each with different performance and compatibility characteristics.

1. XDP Offload Mode

Hardware Execution (Best Performance):

- eBPF program runs directly on SmartNIC hardware
- Packet processing in NIC without touching CPU
- Achieves 100 Gbps+ throughput
- Requires compatible SmartNIC (Netronome, Mellanox ConnectX-6)

Configuration:

```
xdp_attach_mode: offload
```

Limitations:

- Requires expensive SmartNIC hardware
- Limited eBPF program complexity
- Not all eBPF features supported in hardware

---

2. XDP Native Mode (Default for Production)

Driver-Level Execution (High Performance):

- eBPF program runs in network driver context
- Packets processed before SKB (socket buffer) allocation
- Achieves 10-40 Gbps per core
- Requires driver with XDP support (most modern drivers)

Configuration:

```
xdp_attach_mode: native
```

Advantages:

- Very high performance (multi-million pps)
- Wide hardware compatibility
- Full eBPF feature set

Supported Drivers:

- Intel: i40e, ice, ixgbe, igb
- Mellanox: mlx4, mlx5
- Broadcom: bnxt
- Amazon: ena
- Most 10G+ network cards

---

3. XDP Generic Mode

Software Emulation (Compatibility):

- eBPF program runs after kernel allocates SKB
- Software emulation of XDP behavior
- Works on any network interface
- Useful for testing and development

Configuration:

```
xdp_attach_mode: generic
```

Use Cases:

- Development and testing
- Virtualized environments (VMs without SR-IOV)
- Older network hardware
- Loopback interface testing

Performance: 1-5 Gbps (significantly slower than native/offload)

XDP Return Codes

eBPF programs return XDP action codes to tell the kernel what to do with packets:

Return Code	Meaning	Use in OmniUPF
`XDP_PASS`	Send packet to kernel network stack	Buffering (local delivery), ICMP, unknown traffic
`XDP_DROP`	Drop packet immediately	Invalid packets, rate limiting, policy drops
`XDP_TX`	Transmit packet back out same interface	Not currently used
`XDP_REDIRECT`	Send packet to different interface	Main forwarding path (N3 ↔ N6)
`XDP_ABORTED`	Processing error, drop packet and log	eBPF program errors

Packet Processing Pipeline

Program Structure

OmniUPF uses eBPF tail calls to create a modular packet processing pipeline.

Tail Calls:

- Allow eBPF programs to call other eBPF programs
- Reuses same stack frame (bounded stack depth)
- Enables modular pipeline design
- Maximum 33 tail call depth

Uplink Packet Processing

Downlink Packet Processing

eBPF Map Architecture

Map Memory Layout

Map Sizing

OmniUPF automatically calculates map sizes based on `max_sessions` configuration:

```
PDR Maps = 2 × max_sessions  (uplink + downlink)
FAR Maps = 2 × max_sessions  (uplink + downlink)
QER Maps = 1 × max_sessions  (shared per session)
URR Maps = 3 × max_sessions  (multiple URRs per session)
```

Example (max_sessions = 65,535):

- PDR maps: 131,070 entries each
- FAR map: 131,070 entries
- QER map: 65,535 entries
- URR map: 131,070 entries

Total Memory:

```
PDR maps: 3 × 131,070 × 212 B = ~83 MB
FAR map:  131,070 × 20 B = ~2.6 MB
QER map:  65,535 × 36 B = ~2.3 MB
URR map:  131,070 × 20 B = ~2.6 MB
Total: ~91 MB kernel memory
```

Buffering Mechanism

Buffering Overview

OmniUPF implements packet buffering for handover scenarios by encapsulating packets in GTP-U and sending them to a userspace process via UDP socket.

Buffering Architecture

Buffer Encapsulation Details

When buffering is enabled (FAR action bit 2 set), the eBPF program:

1. **Calculates Original Packet Size**:

   ```
   orig_packet_len = ntohs(ip->tot_len);  // From IP header
   ```

2. **Expands Packet Header**:

   ```
   // Add space for: Outer IP + UDP + GTP-U
   gtp_encap_size = sizeof(struct iphdr) + sizeof(struct udphdr) +
   sizeof(struct gtpuhdr);
   bpf_xdp_adjust_head(ctx, -gtp_encap_size);
   ```

3. **Builds Outer IP Header**:

   ```
   ip->saddr = original_sender_ip;  // Preserve source to avoid
   martian filtering
   ip->daddr = local_upf_ip;        // Local IP where userspace
   listener binds
   ip->protocol = IPPROTO_UDP;
   ip->ttl = 64;
   ```

4. **Builds UDP Header**:

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

5. **Builds GTP-U Header**:

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

6. **Returns XDP_PASS**:
    - Kernel delivers packet to local UDP socket on port 22152
    - Userspace buffer manager receives and stores packet

Buffer Flush Operation

When handover completes, SMF updates FAR to clear BUFFER flag. Buffered packets are replayed:

Buffer Management Parameters

Parameter	Default	Description
Max Per FAR	10,000 packets	Maximum packets buffered per FAR
Max Total	100,000 packets	Maximum total buffered packets
Packet TTL	30 seconds	Time before buffered packets expire
Buffer Port	22152	UDP port for buffer delivery
Buffer Cleanup Interval	60 seconds	How often to check for expired packets

QoS Enforcement

Rate Limiting Algorithm

OmniUPF implements a **sliding window rate limiter** for QoS enforcement.

Sliding Window Implementation

Algorithm (from `qer.h`):

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

    // Rate = 0 means unlimited
    if (rate == 0)
        return XDP_PASS;

    // Calculate transmission time for this packet
    __u64 tx_time = packet_size * 8 * (NSEC_PER_SEC / rate);
    __u64 now = bpf_ktime_get_ns();
    // Check if we're ahead of window (packet would transmit in the
future)
    __u64 start = *window_start;
    if (start + tx_time > now)
        return XDP_DROP;  // Rate limit exceeded

    // If window has passed, reset it
    if (start + window_size < now) {
        *window_start = now - window_size + tx_time;
        return XDP_PASS;
    }

    // Update window to account for this packet
    *window_start = start + tx_time;
    return XDP_PASS;
}
```

Key Parameters:

- **Window Size**: 5ms (5,000,000 nanoseconds)
- **Per-Direction**: Separate windows for uplink and downlink
- **Atomic Updates**: Uses volatile pointers for concurrent access
- **MBR = 0**: Treated as unlimited bandwidth

QoS Example Calculation

Scenario: MBR = 100 Mbps, Packet Size = 1500 bytes

1. **Transmission Time**:

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

2. **Rate Check**:

   - If last packet transmitted at `t=0`, next packet can transmit at `t=120μs`
   - If packet arrives at `t=100μs`, it's dropped (too early)
   - If packet arrives at `t=150μs`, it's forwarded (window advanced)

3. **Maximum Packet Rate**:

   ```
   Max PPS = (100 Mbps ÷ 8) ÷ 1500 bytes = 8,333 packets/second
   Inter-packet gap = 120 μs
   ```

Performance Characteristics

Throughput

Configuration	Throughput	Packets/Second	Latency
XDP Offload (SmartNIC)	100 Gbps	148 Mpps	< 1 µs
XDP Native (10G NIC, single core)	10 Gbps	8 Mpps	2-5 µs
XDP Native (10G NIC, 4 cores)	40 Gbps	32 Mpps	2-5 µs
XDP Generic	1-5 Gbps	0.8-4 Mpps	50-100 µs

Latency Breakdown

Total Packet Processing Latency (XDP Native):

Stage	Latency	Cumulative
NIC RX	0.5 µs	0.5 µs
XDP Hook Invocation	0.1 µs	0.6 µs
PDR Lookup (Hash)	0.3 µs	0.9 µs
QER Rate Check	0.1 µs	1.0 µs
FAR Processing	0.5 µs	1.5 µs
URR Update	0.2 µs	1.7 µs
GTP-U Encap/Decap	0.8 µs	2.5 µs
XDP_REDIRECT	0.5 µs	3.0 µs
NIC TX	0.5 µs	3.5 µs

Total: ~3.5 µs per packet (XDP Native, 10G NIC)

CPU Utilization

Per-Core Processing Capacity:

- Single core: 8-10 Mpps (XDP Native)
- With hyper-threading: 12-15 Mpps
- Multi-core scaling: Near-linear up to 8 cores

CPU Usage by Packet Rate:

```
CPU % ≈ (Packet Rate / 10,000,000) × 100% per core
```

Example: 2 Mpps traffic uses ~20% of one core

Memory Bandwidth

eBPF Map Access:

- Hash lookup: ~100 ns (cache hit)
- Hash lookup: ~300 ns (cache miss)
- Array lookup: ~50 ns (always cache hit)

Memory Bandwidth Required:

```
Bandwidth = Packet Rate × (Avg Packet Size + Map Lookups × 64 bytes)
```

Example: 10 Mpps × (1500 B + 3 lookups × 64 B) ≈ 160 Gbps memory bandwidth

Scalability and Tuning

Horizontal Scaling

Multiple UPF Instances:

Session Distribution:

- SMF distributes sessions across UPF instances
- Each UPF handles subset of UE sessions
- No inter-UPF communication needed (stateless)

Vertical Scaling

CPU Tuning:

1. Enable CPU affinity for XDP processing
2. Use RSS (Receive Side Scaling) to distribute RX queues
3. Pin eBPF programs to specific cores

NIC Tuning:

1. Increase RX ring buffer size
2. Enable multi-queue NICs (RSS)
3. Use flow director for traffic steering

Kernel Tuning:

```
# Increase locked memory limit for eBPF maps
ulimit -l unlimited

# Disable IRQ balance for XDP cores
systemctl stop irqbalance

# Set CPU governor to performance
cpupower frequency-set -g performance
```

```
# Increase network buffer sizes
sysctl -w net.core.rmem_max=134217728
sysctl -w net.core.wmem_max=134217728
```

Capacity Planning

Formula:

```
Required CPU Cores = (Expected PPS ÷ 10,000,000) × 1.5  (50%
headroom)
Required Memory = (Max Sessions × 212 B × 3) + 100 MB (eBPF maps +
overhead)
Required Network = (Peak Throughput × 2) + 10 Gbps (headroom)
```

Example (1 million sessions, 20 Gbps peak):

- CPU: (20 Gbps ÷ 10 Gbps per core) × 1.5 = 3-4 cores
- Memory: (1M × 212 B × 3) + 100 MB ≈ 750 MB
- Network: (20 Gbps × 2) + 10 Gbps = 50 Gbps interfaces

Related Documentation

- **UPF Operations Guide** - General UPF operations and deployment
- **Rules Management Guide** - PDR, FAR, QER, URR details
- **Monitoring Guide** - Performance monitoring and metrics
- **Web UI Operations Guide** - Control panel usage
- **Troubleshooting Guide** - Common issues and diagnostics

OmniUPF Configuration Guide

Table of Contents

Overview

OmniUPF is a versatile user plane function that can operate in multiple modes to support both 4G (EPC) and 5G core networks. Configuration is managed through YAML configuration files.

Operating Modes

OmniUPF is a **unified platform** that can simultaneously operate as:

Mode Configuration

The operating mode is **determined by the control plane** (SMF, PGW-C, or SGW-C) that establishes PFCP associations with OmniUPF. No specific OmniUPF configuration is required to switch between modes.

Simultaneous Operation:

- OmniUPF can accept PFCP associations from multiple control planes concurrently
- A single OmniUPF instance can act as UPF, PGW-U, and SGW-U **at the same time**
- Sessions from different control planes are isolated and managed independently

---

XDP Attachment Modes

OmniUPF uses XDP (eXpress Data Path) for high-performance packet processing. Three attachment modes are supported.

For detailed XDP setup instructions, especially for Proxmox and other hypervisors, see the XDP Modes Guide.

Mode Comparison

Mode	Attach Point	Performance	Use Case	NIC Requirements
Generic	Network stack (kernel)	~1-2 Mpps	Testing, development, compatibility	Any NIC
Native	Network driver (kernel)	~5-10 Mpps	Production (bare metal, VM with SR-IOV)	XDP-capable driver
Offload	NIC hardware (SmartNIC)	~10-40 Mpps	High-throughput production	SmartNIC with XDP offload

Generic Mode (Default)

Description: XDP program runs in the kernel network stack

Advantages:

- Works with **any** network interface
- No special driver or hardware requirements
- Ideal for testing and development
- Compatible with all hypervisors and virtualization platforms

Disadvantages:

- Lower performance (~1-2 Mpps per core)
- Packets already passed through driver before XDP processing

Configuration:

```
xdp_attach_mode: generic
```

Best for:

- Virtual machines without SR-IOV
- Testing and validation environments
- NICs without XDP driver support
- Hypervisors like Proxmox, VMware, VirtualBox

Native Mode (Recommended)

Description: XDP program runs at the network driver level

Advantages:

- High performance (~5-10 Mpps per core)
- Packets processed before entering network stack
- Significantly lower latency than generic mode
- Works on bare metal and SR-IOV VMs

Disadvantages:

- Requires network driver with XDP support
- Not all NICs/drivers support native XDP

Configuration:

```
xdp_attach_mode: native
```

Best for:

- Production deployments on bare metal
- VMs with SR-IOV passthrough
- NICs with XDP-capable drivers (Intel, Mellanox, etc.)

Requirements:

- XDP-capable network driver (see NIC Compatibility)
- Linux kernel 5.15+ with XDP support enabled

Offload Mode (Maximum Performance)

Description: XDP program runs directly on SmartNIC hardware

Advantages:

- Maximum performance (~10-40 Mpps)
- Zero CPU overhead for packet processing
- Sub-microsecond latency
- Frees CPU for control plane processing

Disadvantages:

- Requires expensive SmartNIC hardware
- Limited SmartNIC availability
- Complex setup and configuration

Configuration:

```
xdp_attach_mode: offload
```

Best for:

- Ultra-high-throughput production deployments
- Edge computing with strict latency requirements
- Environments where CPU resources are limited

Requirements:

- SmartNIC with XDP offload support (Netronome Agilio CX, Mellanox BlueField)
- Specialized firmware and drivers

Configuration Parameters

Network Interfaces

Parameter	Description	Type	Default
`interface_name`	Network interfaces for N3/N6/N9 traffic (XDP attachment points)	List	`[lo]`
`n3_address`	IPv4 address for N3 interface (GTP-U from RAN)	IP	`127.0.0.1`
`n9_address`	IPv4 address for N9 interface (UPF-to-UPF for ULCL)	IP	Same as `n3_address`

Example:

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

PFCP Configuration

Parameter	Description	Type	Default
`pfcp_address`	Local address for PFCP server (N4/Sxb/Sxc interface)	Host:Port	`:8805`
`pfcp_node_id`	Local Node ID for PFCP protocol	IP	`127.0.0.1`
`pfcp_remote_node`	Remote PFCP peers (SMF/PGW-C/SGW-C) to connect	List	`[]`
`association_setup_timeout`	Timeout between Association	Integer	`5`

Parameter	Description	Type	Default
	Setup Requests (seconds)		
`heartbeat_retries`	Number of heartbeat retries before declaring peer dead	Integer	3
`heartbeat_interval`	PFCP heartbeat interval (seconds)	Integer	5
`heartbeat_timeout`	PFCP heartbeat timeout (seconds)	Integer	5

Example:

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

API and Monitoring

Parameter	Description	Type	Default
`api_address`	Local address for REST API server	Host:Port	`:8080`
`metrics_address`	Local address for Prometheus metrics endpoint	Host:Port	`:9090`
`logging_level`	Logging level (`trace`, `debug`, `info`, `warn`, `error`)	String	`info`

Example:

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

GTP Path Management

Parameter	Description	Type	Default
`gtp_peer`	List of GTP peers for Echo Request keepalives	List	`[]`
`gtp_echo_interval`	Interval between GTP Echo Requests (seconds)	Integer	`10`

Example:

```yaml
gtp_peer:
  - 10.100.50.50:2152  # gNB
  - 10.100.50.60:2152  # Another UPF for N9
gtp_echo_interval: 15
```

---

eBPF Map Capacity

Parameter	Description	Type	Default	Auto-calculated
`max_sessions`	Maximum number of concurrent sessions	Integer	`65535`	Used to calculate map sizes
`pdr_map_size`	Size of PDR eBPF map	Integer	`0`	`max_sessions × 2`
`far_map_size`	Size of FAR eBPF map	Integer	`0`	`max_sessions × 2`
`qer_map_size`	Size of QER eBPF map	Integer	`0`	`max_sessions`
`urr_map_size`	Size of URR eBPF map	Integer	`0`	`max_sessions × 2`

Note: Setting map sizes to `0` (default) enables auto-calculation based on `max_sessions`. Override with specific values if custom sizing is needed.

Example:

```yaml
max_sessions: 100000
# Maps will be auto-sized:
# PDR: 200,000 entries
# FAR: 200,000 entries
# QER: 100,000 entries
# URR: 200,000 entries
```

Custom sizing example:

```yaml
max_sessions: 50000
pdr_map_size: 131070  # Custom size
far_map_size: 131070
qer_map_size: 65535
urr_map_size: 131070
```

Buffer Configuration

Parameter	Description	Type	Default
`buffer_port`	UDP port for buffered packets from eBPF	Integer	`22152`
`buffer_max_packets`	Maximum packets to buffer per FAR	Integer	`10000`
`buffer_max_total`	Maximum total buffered packets (0=unlimited)	Integer	`100000`
`buffer_packet_ttl`	TTL for buffered packets in seconds (0=no expiration)	Integer	`30`

Parameter	Description	Type	Default
`buffer_cleanup_interval`	Buffer cleanup interval in seconds (0=no cleanup)	Integer	`60`

Example:

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

Feature Flags

Parameter	Description	Type	Default
`feature_ueip`	Enable UE IP allocation by OmniUPF	Boolean	`false`
`ueip_pool`	IP pool for UE IP allocation (requires `feature_ueip`)	CIDR	`10.60.0.0/24`
`feature_ftup`	Enable F-TEID allocation by OmniUPF	Boolean	`false`
`teid_pool`	TEID pool size for F-TEID allocation (requires `feature_ftup`)	Integer	`65535`

Example (UE IP allocation):

```yaml
feature_ueip: true
ueip_pool: 10.45.0.0/16  # Allocate UE IPs from this pool
```

Example (F-TEID allocation):

```yaml
feature_ftup: true
teid_pool: 1000000  # Allow up to 1M TEID allocations
```

Configuration Methods

YAML Configuration File (Recommended)

File: `config.yml`

```yaml
# Network Configuration
interface_name: [eth0]
n3_address: 10.100.50.233
n9_address: 10.100.50.233
xdp_attach_mode: native

```

```
# PFCP Configuration
pfcp_address: :8805
pfcp_node_id: 10.100.50.241
pfcp_remote_node:
  - 10.100.50.10

# API and Monitoring
api_address: :8080
metrics_address: :9090
logging_level: info

# Capacity
max_sessions: 100000

# GTP Peers
gtp_peer:
  - 10.100.50.50:2152
gtp_echo_interval: 10

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

# Buffering
buffer_max_packets: 15000
buffer_packet_ttl: 45
```

Starting OmniUPF:

```
./eupf --config /path/to/config.yml
```

Hypervisor Compatibility

Overview

OmniUPF is compatible with all major hypervisors and virtualization platforms. The XDP attach mode and network configuration depend on the hypervisor's networking capabilities.

For step-by-step instructions on enabling native XDP on Proxmox and other hypervisors, see the XDP Modes Guide.

Proxmox VE

Supported Configurations:

1. Bridge Mode (Generic XDP)

Use case: Standard VM networking

Configuration:

- Network Device: VirtIO or E1000
- XDP Mode: `generic`
- Performance: ~1-2 Mpps

Proxmox VM Settings:

```
Network Device: net0
Model: VirtIO (paravirtualized)
Bridge: vmbr0
```

OmniUPF Config:

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

---

2. SR-IOV Passthrough (Native XDP)

Use case: High-performance production

Configuration:

- Network Device: SR-IOV Virtual Function
- XDP Mode: `native`
- Performance: ~5-10 Mpps

Requirements:

- Physical NIC with SR-IOV support (Intel X710, Mellanox ConnectX-5)
- SR-IOV enabled in BIOS
- IOMMU enabled (`intel_iommu=on` or `amd_iommu=on` in GRUB)

Enable SR-IOV on Proxmox:

```
# Edit GRUB configuration
nano /etc/default/grub

# Add to GRUB_CMDLINE_LINUX_DEFAULT:
```

```
intel_iommu=on iommu=pt

# Update GRUB and reboot
update-grub
reboot

# Enable VFs on NIC (example: 4 virtual functions on eth0)
echo 4 > /sys/class/net/eth0/device/sriov_numvfs

# Make persistent
echo "echo 4 > /sys/class/net/eth0/device/sriov_numvfs" >> /etc/
rc.local
chmod +x /etc/rc.local
```

Proxmox VM Settings:

```
Hardware → Add → PCI Device
Select: SR-IOV Virtual Function
All Functions: No
Primary GPU: No
PCI-Express: Yes (optional)
```

OmniUPF Config:

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

---

3. PCI Passthrough (Native XDP)

Use case: Dedicated NIC for single VM

Configuration:

- Entire physical NIC passed to VM
- XDP Mode: `native` or `offload` (if SmartNIC)
- Performance: ~5-40 Mpps (depends on NIC)

Proxmox VM Settings:

```
Hardware → Add → PCI Device
Select: Physical NIC (e.g., 0000:01:00.0)
All Functions: Yes
Primary GPU: No
PCI-Express: Yes
```

OmniUPF Config:

```
interface_name: [ens1f0]
xdp_attach_mode: native  # or 'offload' for SmartNIC
```

KVM/QEMU

Bridge Mode:

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

SR-IOV Passthrough:

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

VMware ESXi

Standard vSwitch (Generic XDP):

- Network Adapter: VMXNET3
- XDP Mode: `generic`

SR-IOV (Native XDP):

- Enable SR-IOV in ESXi host settings
- Add SR-IOV network adapter to VM
- XDP Mode: `native`

Microsoft Hyper-V

Virtual Switch (Generic XDP):

- Network Adapter: Synthetic
- XDP Mode: `generic`

SR-IOV (Native XDP):

- Enable SR-IOV in Hyper-V Manager

- Configure SR-IOV on virtual network adapter
- XDP Mode: `native`

VirtualBox

NAT/Bridged Mode (Generic XDP only):

- Network Adapter: VirtIO-Net or Intel PRO/1000
- XDP Mode: `generic`
- Note: VirtualBox does **not** support SR-IOV

NIC Compatibility

Understanding Mpps vs Throughput

Packets per second (Mpps) and throughput (Gbps) are not directly equivalent - the relationship depends entirely on packet size. Mobile network traffic varies dramatically in packet size, from tiny VoIP packets to large video streaming frames.

Packet Size Impact on Throughput

In mobile networks, the UPF processes GTP-U encapsulated packets on the N3 interface and native IP packets on the N6 interface.

GTP-U Encapsulation Overhead (N3 Interface):

- **Outer IPv4 header**: 20 bytes
- **Outer UDP header**: 8 bytes
- **GTP-U header**: 8 bytes
- **Total GTP-U overhead**: 36 bytes

Minimum GTP-U Packet (N3):

- **Inner IP header**: 20 bytes (IPv4)
- **Inner UDP header**: 8 bytes
- **Minimum payload**: 1 byte
- **Inner packet total**: 29 bytes
- **Plus GTP-U overhead**: 36 bytes
- **Total packet size**: 65 bytes

Throughput at 1 Mpps with minimum GTP-U packets:

```
65 bytes × 1,000,000 pps × 8 bits/byte = 520 Mbps
```

Maximum GTP-U Packet (N3 with 1500 MTU):

- **Inner IP MTU**: 1500 bytes (full inner IP packet)
- **Plus GTP-U overhead**: 36 bytes
- **Total packet size**: 1536 bytes

Throughput at 1 Mpps with maximum GTP-U packets:

```
1536 bytes × 1,000,000 pps × 8 bits/byte = 12,288 Mbps ≈ 12.3 Gbps
```

Native IP Packets (N6 Interface):

On N6 (towards Internet), packets are native IP without GTP-U:

Minimum N6 packet:

- **IP header**: 20 bytes
- **UDP header**: 8 bytes
- **Minimum payload**: 1 byte
- **Total**: 29 bytes

Throughput at 1 Mpps with minimum N6 packets:

```
29 bytes × 1,000,000 pps × 8 bits/byte = 232 Mbps
```

Maximum N6 packet (1500 MTU):

- **IP MTU**: 1500 bytes
- **Total**: 1500 bytes

Throughput at 1 Mpps with maximum N6 packets:

```
1500 bytes × 1,000,000 pps × 8 bits/byte = 12,000 Mbps = 12 Gbps
```

Real-World Performance Examples

Intel X710 NIC (10 Mpps capacity on N3 interface with GTP-U):

Traffic Pattern	Inner Packet Size	GTP-U Total	Throughput at 10 Mpps	Typical Use Case
VoIP calls (N3)	65-150 bytes	101-186 bytes	**0.8-1.5 Gbps**	AMR-WB voice, G.711
Light web (N3)	400-600 bytes	436-636 bytes	**3.5-5.1 Gbps**	HTTP/HTTPS, messaging
Modern mobile (N3)	**1200 bytes**	**1236 bytes**	**9.9 Gbps**	**Typical 2024 traffic mix**
Video streaming	1400-1450	1436-1486	**11.5-11.9 Gbps**	HD/4K video

Traffic Pattern	Inner Packet Size	GTP-U Total	Throughput at 10 Mpps	Typical Use Case
(N3)	bytes	bytes		chunks
Maximum MTU (N3)	1500 bytes	1536 bytes	**12.3 Gbps**	Large TCP downloads

On N6 interface (native IP, no GTP-U):

Traffic Pattern	Packet Size	Throughput at 10 Mpps	Typical Use Case
VoIP packets	65-150 bytes	**0.5-1.2 Gbps**	Voice RTP streams
Light web	400-600 bytes	**3.2-4.8 Gbps**	HTTP requests
Modern mobile	**1200 bytes**	**9.6 Gbps**	**Typical 2024 traffic**
Video streaming	1400-1450 bytes	**11.2-11.6 Gbps**	Video downloads
Maximum MTU	1500 bytes	**12.0 Gbps**	Large file transfers

At 10 Mpps with modern mobile traffic (1200-byte average), expect **~10 Gbps throughput** on both N3 and N6 interfaces.

Why This Matters for Mobile Networks:

Mobile traffic is **highly variable** in packet size and the GTP-U overhead (36 bytes) significantly impacts small packet performance:

Inner packet size (actual user data):

- **VoIP (AMR-WB codec)**: 65-80 bytes → With GTP-U: 101-116 bytes
- **IoT sensor data**: 50-200 bytes → With GTP-U: 86-236 bytes
- **Web browsing (HTTP/3)**: 400-800 bytes → With GTP-U: 436-836 bytes
- **Video streaming**: 1200-1450 bytes → With GTP-U: 1236-1486 bytes
- **Large downloads**: 1500 bytes → With GTP-U: 1536 bytes

Impact of GTP-U overhead:

- Small packets (< 200 bytes): **~35-70% overhead** - Mpps is limiting factor
- Medium packets (200-800 bytes): **~5-20% overhead** - Mixed limitation
- Large packets (> 1200 bytes): **~3% overhead** - Link speed is limiting factor

Performance Planning:

A NIC rated at **10 Mpps** will achieve on N3 interface:

- **VoIP-heavy traffic** (100-byte inner packets): ~1.0 Gbps (GTP-U overhead dominates)
- **Modern mobile mix** (1200-byte average inner packets): ~9.9 Gbps
- **Video-heavy traffic** (1400-byte inner packets): ~11.5 Gbps
- **Maximum throughput** (1500-byte inner packets): ~12.3 Gbps

On N6 interface (no GTP-U overhead):

- **Modern mobile mix** (1200-byte packets): ~9.6 Gbps at 10 Mpps
- **Maximum throughput** (1500-byte packets): ~12.0 Gbps at 10 Mpps

Rule of Thumb for Mobile UPF:

- **Small packet traffic** (VoIP, IoT, signaling): Mpps is limiting - plan for 1-2 Gbps per 10 Mpps
- **Modern mobile traffic** (1200-byte average): Plan for ~9-10 Gbps per 10 Mpps capacity
- **Video-heavy traffic** (streaming, downloads): Plan for ~10-12 Gbps per 10 Mpps capacity
- **Always consider both N3 and N6** - N3 has GTP-U overhead, N6 does not

Practical Capacity Planning:

With 1200-byte average packet size (typical for modern mobile networks with video streaming):

NIC Mpps Capacity	N3 Throughput (GTP-U)	N6 Throughput (Native IP)	Realistic Deployment
1 Mpps	~1.0 Gbps	~1.0 Gbps	Small cell site, IoT gateway
5 Mpps	~4.9 Gbps	~4.8 Gbps	Medium cell site, enterprise
10 Mpps	~9.9 Gbps	~9.6 Gbps	Large cell site, small city
20 Mpps	~19.7 Gbps	~19.2 Gbps	Metro area, medium city
40 Mpps	~39.4 Gbps	~38.4 Gbps	Large metro, regional hub

Note: These estimates assume 1200-byte average payload size, which is representative of modern mobile traffic dominated by video streaming, social media, and cloud applications. Actual throughput will vary based on traffic mix.

XDP-Capable Network Drivers

OmniUPF requires network drivers with XDP support for **native** and **offload** modes. Generic mode works with **any** NIC.

Intel NICs

Model	Driver	XDP Support	Mode	Performance
Intel X710	i40e	Yes	Native	~10 Mpps

Model	Driver	XDP Support	Mode	Performance
Intel XL710	i40e	Yes	Native	~10 Mpps
Intel E810	ice	Yes	Native	~15 Mpps
Intel 82599ES	ixgbe	Yes	Native	~8 Mpps
Intel I350	igb	Limited	Generic	~1 Mpps
Intel E1000	e1000	No	Generic only	~1 Mpps

Mellanox/NVIDIA NICs

Model	Driver	XDP Support	Mode	Performance
Mellanox ConnectX-5	mlx5	Yes	Native	~12 Mpps
Mellanox ConnectX-6	mlx5	Yes	Native	~20 Mpps
Mellanox BlueField	mlx5	Yes	Native + Offload	~40 Mpps
Mellanox ConnectX-4	mlx4	Limited	Generic	~2 Mpps

Broadcom NICs

Model	Driver	XDP Support	Mode	Performance
Broadcom BCM57xxx	bnxt_en	Yes	Native	~8 Mpps
Broadcom NetXtreme II	bnx2x	No	Generic only	~1 Mpps

Other Vendors

Model	Driver	XDP Support	Mode	Performance
Netronome Agilio CX	nfp	Yes	Offload	~30 Mpps
Amazon ENA	ena	Yes	Native	~5 Mpps
Solarflare SFC9xxx	sfc	Yes	Native	~8 Mpps
VirtIO	virtio_net	Limited	Generic	~2 Mpps

Checking NIC XDP Support

Check if driver supports XDP:

```
# Find NIC driver
ethtool -i eth0 | grep driver

# Check XDP support in driver
modinfo <driver_name> | grep -i xdp

# Example for Intel i40e
modinfo i40e | grep -i xdp
```

Verify XDP program attachment:

```
# Check if XDP program is attached
ip link show eth0 | grep -i xdp

# Example output (XDP attached):
# 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 xdp qdisc mq
```

Recommended NICs by Use Case

With 1200-byte average packet size (modern mobile traffic):

Use Case	Recommended NIC	Mode	Mpps Capacity	Throughput (N3)	Deployment Scenario
Testing/ Development	Any NIC (VirtIO, E1000)	Generic	1-2 Mpps	1-2 Gbps	Lab testing, PoC
Small Cell Site	Intel X710, Mellanox CX-5	Native	5-10 Mpps	5-10 Gbps	Rural cell, enterprise
Medium Cell/Metro	Intel E810, Mellanox CX-6	Native	10-20 Mpps	10-20 Gbps	Urban cell, small city
Large Metro	Mellanox CX-6, Intel E810 (dual)	Native	20-40 Mpps	20-40 Gbps	Metro area, medium city
Regional Hub	Mellanox BlueField, Netronome Agilio	Offload	40+ Mpps	40+ Gbps	Regional aggregation
Proxmox VM (Bridge)	VirtIO	Generic	1-2 Mpps	1-2 Gbps	Testing only
Proxmox VM (SR-IOV)	Intel X710/E810 VF, Mellanox CX-5 VF	Native	5-10 Mpps	5-10 Gbps	Production VM

Throughput Estimates:

- Based on 1200-byte average packet size with GTP-U encapsulation (1236 bytes on N3)
- N6 throughput slightly lower (~9.6 Gbps per 10 Mpps) due to no GTP-U overhead
- Actual performance varies with traffic mix - VoIP-heavy networks will see lower throughput

---

Additional Resources

Official XDP Documentation:

- XDP Project
- Kernel XDP Documentation

NIC Compatibility Lists:

- Cilium XDP Hardware Support
- IO Visor XDP Drivers

Configuration Examples

Example 1: Development Environment (Generic Mode)

Scenario: Testing OmniUPF on laptop or VM without SR-IOV

```yaml
# Development config
interface_name: [eth0]
xdp_attach_mode: generic
api_address: :8080
pfcp_address: :8805
pfcp_node_id: 127.0.0.1
n3_address: 127.0.0.1
metrics_address: :9090
logging_level: debug
max_sessions: 1000
```

Example 2: Production Bare Metal (Native Mode)

Scenario: Production UPF on bare metal server with Intel X710 NIC

```yaml
# Production bare metal config
interface_name: [ens1f0, ens1f1]  # N3 on ens1f0, N6 on ens1f1
xdp_attach_mode: native
api_address: :8080
pfcp_address: 10.100.50.241:8805
pfcp_node_id: 10.100.50.241
n3_address: 10.100.50.233
n9_address: 10.100.50.234
metrics_address: :9090
logging_level: info
max_sessions: 500000
gtp_peer:
  - 10.100.50.10:2152  # gNB 1
  - 10.100.50.11:2152  # gNB 2
gtp_echo_interval: 30
pfcp_remote_node:
  - 10.100.50.50  # OmniSMF
heartbeat_interval: 10
feature_ueip: true
ueip_pool: 10.45.0.0/16
buffer_max_packets: 50000
```

```
buffer_packet_ttl: 60
```

Example 3: Proxmox VM with SR-IOV (Native Mode)

Scenario: Production UPF on Proxmox VM with SR-IOV passthrough

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

Example 4: PGW-U Mode (4G EPC)

Scenario: OmniUPF acting as PGW-U in 4G EPC network

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

Example 5: Multi-Mode (UPF + PGW-U Simultaneously)

Scenario: OmniUPF serving both 5G and 4G networks concurrently

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

Example 6: SmartNIC Offload Mode

Scenario: Ultra-high-throughput deployment with Netronome Agilio CX SmartNIC

```yaml
# SmartNIC offload configuration
interface_name: [enp1s0np0]  # SmartNIC interface
xdp_attach_mode: offload
api_address: :8080
pfcp_address: 10.10.1.50:8805
pfcp_node_id: 10.10.1.50
n3_address: 10.10.1.50
metrics_address: :9090
logging_level: warn  # Reduce overhead
max_sessions: 1000000
pdr_map_size: 2000000
far_map_size: 2000000
qer_map_size: 1000000
gtp_peer:
  - 10.10.2.10:2152
  - 10.10.2.20:2152
  - 10.10.2.30:2152
gtp_echo_interval: 30
pfcp_remote_node:
  - 10.10.3.10
heartbeat_interval: 15
buffer_max_packets: 100000
buffer_max_total: 1000000
```

Map Sizing and Capacity Planning

Auto-Sizing (Recommended)

Set `max_sessions` and let OmniUPF calculate map sizes automatically:

```
max_sessions: 100000
# Auto-calculated sizes:
# PDR: 200,000 entries (2 × max_sessions)
# FAR: 200,000 entries (2 × max_sessions)
# QER: 100,000 entries (1 × max_sessions)
# URR: 200,000 entries (2 × max_sessions)
```

Memory usage: ~91 MB for 100K sessions

Manual Sizing

Override auto-calculation for custom requirements:

```
max_sessions: 100000
pdr_map_size: 300000  # Support more PDRs per session
far_map_size: 200000
qer_map_size: 150000  # More QERs than default
urr_map_size: 200000
```

Capacity Estimation

Calculate maximum sessions:

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

Example:

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

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

Memory Requirements

Per session memory usage:

- PDR: 2 × 212 B = 424 B
- FAR: 2 × 20 B = 40 B
- QER: 1 × 36 B = 36 B
- URR: 2 × 20 B = 40 B
- **Total**: ~540 B per session

For 100K sessions: ~52 MB kernel memory

Recommendation: Ensure locked memory limit allows 2× estimated usage:

```
# Check current limit
ulimit -l

# Set unlimited (required for eBPF)
ulimit -l unlimited
```

Related Documentation

- **Architecture Guide** - eBPF/XDP technical details and performance optimization
- **Rules Management Guide** - PDR, FAR, QER, URR configuration
- **Monitoring Guide** - Statistics, capacity monitoring, and alerting
- **Web UI Guide** - Control panel operations
- **Operations Guide** - UPF architecture and deployment overview

Monitoring Guide

Table of Contents

Overview

Effective monitoring of OmniUPF is critical for maintaining service quality, preventing capacity exhaustion, and troubleshooting performance issues. OmniUPF provides comprehensive real-time metrics through its Web UI and REST API.

Monitoring Categories

Category	Purpose	Update Frequency	Key Metrics
Packet Statistics	Track packet processing rates and errors	Real-time	RX/TX packets, drops, protocol breakdown
Interface Statistics	Monitor N3/N6 traffic distribution	Real-time	N3 RX/TX, N6 RX/TX
XDP Statistics	Track kernel datapath performance	Real-time	XDP processed, passed, dropped, aborted
Route Statistics	Monitor packet routing decisions	Real-time	FIB lookups, cache hits/misses
eBPF Map Capacity	Prevent resource exhaustion	Every 10s	Map usage percentages, used vs. capacity
Buffer Statistics	Track packet buffering during mobility	Every 5s	Buffered packets, buffer age, FAR count

Statistics Monitoring

N3/N6 Interface Statistics

N3/N6 interface statistics provide visibility into traffic distribution between the

RAN (N3) and Data Network (N6).

Metrics:

- **RX N3**: Packets received from RAN (uplink GTP-U traffic)
- **TX N3**: Packets transmitted to RAN (downlink GTP-U traffic)
- **RX N6**: Packets received from Data Network (downlink native IP)
- **TX N6**: Packets transmitted to Data Network (uplink native IP)
- **Total**: Aggregate packet count across all interfaces

Expected Behavior:

- **RX N3 ≈ TX N6**: Uplink packets flow from RAN to Data Network
- **RX N6 ≈ TX N3**: Downlink packets flow from Data Network to RAN
- Significant imbalance may indicate:
    - Asymmetric traffic (downloads >> uploads)
    - Packet drops or forwarding errors
    - Routing misconfigurations

XDP Statistics

XDP (eXpress Data Path) statistics show kernel-level packet processing performance.

Metrics:

- **Aborted**: XDP program encountered an error (should always be 0)
- **Drop**: Packets intentionally dropped by XDP program
- **Pass**: Packets passed to network stack for further processing
- **Redirect**: Packets directly redirected to output interface
- **TX**: Packets transmitted via XDP

Interpretation:

- **Aborted > 0**: Critical issue with eBPF program or kernel compatibility
- **Drop > 0**: Policy-based drops or invalid packets
- **Pass high**: Most packets processed in network stack (normal)
- **Redirect high**: Packets forwarded directly (optimal performance)

Packet Statistics

Detailed packet protocol breakdown and processing counters.

Protocol Counters:

- **RX ARP**: Address Resolution Protocol packets

- **RX GTP ECHO**: GTP-U Echo Request/Response (keepalive)
- **RX GTP OTHER**: Other GTP control messages
- **RX GTP PDU**: GTP-U encapsulated user data (main traffic)
- **RX GTP UNEXP**: Unexpected GTP packet types
- **RX ICMP**: Internet Control Message Protocol (ping, errors)
- **RX ICMP6**: ICMPv6 packets
- **RX IP4**: IPv4 packets
- **RX IP6**: IPv6 packets
- **RX OTHER**: Other protocols
- **RX TCP**: Transmission Control Protocol packets
- **RX UDP**: User Datagram Protocol packets

Use Cases:

- **Monitor GTP-U PDU count**: Primary user traffic indicator
- **Check ICMP for connectivity**: Network reachability testing
- **Track TCP vs UDP ratio**: Application traffic patterns
- **Detect unexpected protocols**: Security or misconfiguration issues

---

Route Statistics

FIB (Forwarding Information Base) lookup statistics for routing decisions.

IPv4 FIB Lookup:

- **Cache**: Cached route lookups (fast path)
- **OK**: Successful route lookups

IPv6 FIB Lookup:

- **Cache**: Cached IPv6 route lookups
- **OK**: Successful IPv6 route lookups

Performance Indicators:

- **High Cache Hit Rate**: Indicates good routing cache performance
- **High OK Count**: Confirms routing tables are correctly configured
- **Low or Zero Lookups**: May indicate traffic not flowing or routing bypass

---

Capacity Monitoring

eBPF Map Capacity

eBPF map capacity monitoring prevents session establishment failures due to resource exhaustion.

Critical eBPF Maps

far_map (Forwarding Action Rules):

- **Capacity**: 131,070 entries
- **Key Size**: 4 B (FAR ID)
- **Value Size**: 16 B (forwarding parameters)
- **Memory Usage**: ~2.6 MB
- **Criticality**: High - Used for all packet forwarding decisions

pdr_map_downlin (Downlink PDRs - IPv4):

- **Capacity**: 131,070 entries
- **Key Size**: 4 B (UE IPv4 address)
- **Value Size**: 208 B (PDR info)
- **Memory Usage**: ~27 MB
- **Criticality**: Critical - Session establishment fails if full

pdr_map_downlin_ip6 (Downlink PDRs - IPv6):

- **Capacity**: 131,070 entries
- **Key Size**: 16 B (UE IPv6 address)
- **Value Size**: 208 B (PDR info)
- **Memory Usage**: ~29 MB
- **Criticality**: Critical - IPv6 session establishment fails if full

pdr_map_teid_ip (Uplink PDRs):

- **Capacity**: 131,070 entries
- **Key Size**: 4 B (TEID)
- **Value Size**: 208 B (PDR info)
- **Memory Usage**: ~27 MB
- **Criticality**: Critical - Uplink traffic fails if full

qer_map (QoS Enforcement Rules):

- **Capacity**: 65,535 entries
- **Key Size**: 4 B (QER ID)
- **Value Size**: 32 B (QoS parameters)
- **Memory Usage**: ~2.3 MB
- **Criticality**: Medium - QoS enforcement only

urr_map (Usage Reporting Rules):

- **Capacity**: 131,070 entries
- **Key Size**: 4 B (URR ID)
- **Value Size**: 16 B (volume counters)
- **Memory Usage**: ~2.6 MB

- **Criticality**: Low - Affects charging only

Capacity Thresholds

Threshold	Action Required
0-50% (Green)	Normal operation - No action required
50-70% (Yellow)	Caution - Monitor growth trends, plan capacity increase
70-90% (Amber)	Warning - Schedule capacity increase within 1 week
90-100% (Red)	Critical - Immediate action required, new sessions will fail

Capacity Increase Procedure

Before increasing capacity:

1. Review current usage trends
2. Estimate future growth rate
3. Calculate required capacity

Steps to increase map capacity:

1. Stop OmniUPF service
2. Update UPF configuration file with new map sizes
3. Restart OmniUPF service
4. Verify new capacity in Capacity view
5. Monitor for successful session establishment

Note: Changing eBPF map capacity requires UPF restart and clears all existing sessions.

Performance Metrics

Packet Processing Rate

Calculation:

```
Packet Rate (pps) = (Packet Count Delta) / (Time Delta in seconds)
```

Example:

- Initial RX packets: 7,000
- After 10 seconds: 17,000
- Packet Rate = (17,000 - 7,000) / 10 = 1,000 pps

Performance Targets:

- **Small UPF**: 10,000 - 100,000 pps

- **Medium UPF**: 100,000 - 1,000,000 pps
- **Large UPF**: 1,000,000 - 10,000,000 pps

Bottleneck Indicators:

- XDP aborted count increasing
- High CPU utilization
- Packet drops increasing
- Latency increasing

Throughput Calculation

Calculation:

```
Throughput (Mbps) = (Byte Count Delta × 8) / (Time Delta in seconds × 1,000,000)
```

Example:

- Initial RX bytes: 500 MB
- After 60 seconds: 800 MB
- Throughput = (300 MB × 8) / (60 × 1,000,000) = 40 Mbps

Capacity Planning:

- Monitor peak throughput times (e.g., evening hours)
- Compare to link capacity (N3/N6 interface speeds)
- Plan for 2x peak throughput for headroom

Drop Rate

Calculation:

```
Drop Rate (%) = (Dropped Packets / Total RX Packets) × 100
```

Acceptable Thresholds:

- **< 0.1%**: Excellent (normal packet loss due to errors)
- **0.1% - 1%**: Good (minor issues or rate limiting)
- **1% - 5%**: Poor (investigate QoS or capacity issues)
- **> 5%**: Critical (major forwarding or capacity problem)

Common Drop Causes:

- QER rate limiting (MBR exceeded)
- eBPF map lookup failures

- Invalid TEIDs or UE IPs
- Routing errors

Alerting and Thresholds

Recommended Alerts

Critical Alerts (Immediate response required):

- eBPF map capacity > 90%
- XDP aborted count > 0
- Drop rate > 5%
- UPF health check failed

Warning Alerts (Response within 1 hour):

- eBPF map capacity > 70%
- Drop rate > 1%
- Packet rate approaching link capacity
- Buffer TTL exceeded (packets older than 30s)

Informational Alerts (Monitor trends):

- eBPF map capacity > 50%
- Buffered packet count increasing
- New PFCP associations established/released
- URR volume thresholds exceeded

Alert Configuration

Alerts can be configured via:

1. **Prometheus Metrics**: Export metrics for external monitoring
2. **Log Monitoring**: Parse OmniUPF logs for error patterns
3. **REST API Polling**: Periodically query `/map_info`, `/packet_stats` endpoints
4. **Web UI Monitoring**: Manual monitoring via Statistics and Capacity pages

---

Capacity Planning

Session Capacity Estimation

Calculate maximum sessions:

```
Max Sessions = min(
  PDR Map Capacity / 2,  # Downlink + Uplink PDRs per session
  FAR Map Capacity / 2,  # Downlink + Uplink FARs per session
  QER Map Capacity       # Optional, one QER per session
)
```

Example:

- PDR Map Capacity: 131,070
- FAR Map Capacity: 131,070
- QER Map Capacity: 65,535

Max Sessions = min(131,070 / 2, 131,070 / 2, 65,535) = **65,535 sessions**

Memory Capacity

Calculate total eBPF map memory:

```
Memory = Σ (Map Capacity × (Key Size + Value Size))
```

Example Configuration:

- PDR maps: 3 × 131,070 × 212 B = 83.3 MB
- FAR map: 131,070 × 20 B = 2.6 MB
- QER map: 65,535 × 36 B = 2.3 MB
- URR map: 131,070 × 20 B = 2.6 MB
- **Total**: ~91 MB of kernel memory

Kernel Memory Considerations:

- Ensure sufficient locked memory limit (`ulimit -l`)
- Reserve 2x estimated usage for safety margin
- Monitor kernel memory availability

Traffic Capacity

Calculate required throughput capacity:

1. **Estimate average session throughput**:
    - Video streaming: ~5 Mbps
    - Web browsing: ~1 Mbps
    - VoIP: ~0.1 Mbps
2. **Calculate aggregate throughput**:

    ```
    Total Throughput = Sessions × Average Session Throughput
    ```
3. **Add headroom**:

```
Required Capacity = Total Throughput × 2  # 100% headroom
```

Example:

- 10,000 concurrent sessions
- Average 2 Mbps per session
- Total: 20 Gbps
- Required capacity: 40 Gbps (N3 + N6 interfaces)

Growth Planning

Trend Analysis:

1. Record daily peak session count
2. Calculate weekly growth rate
3. Extrapolate to capacity limit

Growth Rate Formula:

```
Weeks to Capacity = (Capacity - Current Usage) / (Weekly Growth)
```

Example:

- Current sessions: 30,000
- Capacity: 65,535 sessions
- Weekly growth: 2,000 sessions
- Weeks to capacity: (65,535 - 30,000) / 2,000 = **17.8 weeks**

Action: Plan capacity upgrade in 12 weeks (leaving 5 weeks buffer).

Troubleshooting Performance Issues

High Packet Drop Rate

Symptoms: Drop rate > 1%, user complaints of poor connectivity

Diagnosis:

1. Check Statistics → Packet Statistics
2. Identify if drops are protocol-specific
3. Review XDP Statistics for XDP drops vs. aborts

Common Causes:

- **QER Rate Limiting**: Check QER MBR values vs. actual traffic
- **Invalid TEIDs**: Verify uplink PDR TEID matches gNB assignment
- **Unknown UE IPs**: Verify downlink PDR exists for UE IP

- **Buffer Overflow**: Check buffer statistics

Resolution:

- Increase QER MBR if rate limiting
- Verify SMF has created correct PDRs
- Clear buffers if overflow detected

XDP Processing Errors

Symptoms: XDP aborted > 0

Diagnosis:

1. Navigate to Statistics → XDP Statistics
2. Check aborted counter
3. Review OmniUPF logs for eBPF errors

Common Causes:

- eBPF program verification failure
- Kernel version incompatibility
- eBPF map access errors
- Memory corruption

Resolution:

- Restart OmniUPF service
- Check kernel version meets minimum requirements (Linux 5.4+)
- Review eBPF program logs
- Contact support if issue persists

Capacity Exhaustion

Symptoms: Session establishment failures, map capacity at 100%

Diagnosis:

1. Navigate to Capacity page
2. Identify which map is at 100%
3. Check if sessions are stuck (not being deleted)

Immediate Mitigation:

1. Identify stale sessions (check Sessions page)
2. Request SMF to delete old sessions

3. Clear buffers to free FAR entries

Long-term Resolution:

1. Increase eBPF map capacity
2. Schedule UPF restart with larger maps
3. Implement session cleanup policies

Performance Degradation

Symptoms: High latency, low throughput, CPU saturation

Diagnosis:

1. Check packet rate vs. historical baseline
2. Review XDP statistics for processing delays
3. Monitor CPU utilization on UPF host
4. Check N3/N6 interface utilization

Common Causes:

- Traffic exceeding UPF capacity
- Insufficient CPU cores for packet processing
- Network interface bottleneck
- eBPF map hash collisions

Resolution:

- Scale UPF horizontally (add more instances)
- Upgrade CPU or enable RSS (Receive Side Scaling)
- Upgrade network interfaces to higher speed
- Tune eBPF map hash function

Related Documentation

- **UPF Operations Guide** - General UPF architecture and operations
- **Rules Management Guide** - PDR, FAR, QER, URR configuration
- **Web UI Operations Guide** - Control panel monitoring features
- **Troubleshooting Guide** - Common issues and diagnostics
- **Architecture Guide** - eBPF datapath and performance optimization

N9 Loopback: Running SGWU and PGWU on Same Instance

Overview

OmniUPF supports running both **SGWU (Serving Gateway User Plane)** and **PGWU (PDN Gateway User Plane)** functions on the **same instance** with **zero-latency N9 loopback**. This deployment mode is ideal for:

- **Simplified 4G EPC deployments** - Single UPF instance instead of two
- **Cost optimization** - Reduced infrastructure and operational complexity
- **Edge computing** - Minimize latency for local breakout scenarios
- **Lab/testing environments** - Full EPC user plane on single server

When configured with the same IP address for both N3 and N9 interfaces, OmniUPF **automatically detects** traffic flowing between the SGWU and PGWU roles and processes it **entirely in eBPF** without ever sending packets to the network interface.

How It Works

Traditional Deployment (Two Instances)

Packet Flow:

1. eNodeB → SGWU: GTP packet (TEID=100) arrives on S1-U
2. SGWU: Matches uplink PDR, encapsulates in new GTP tunnel (TEID=200)
3. **Packet sent over physical N9 network** to PGWU instance
4. PGWU: Receives GTP (TEID=200), decapsulates, forwards to Internet
5. **Total: 2 XDP passes + 1 network hop**

N9 Loopback Deployment (Single Instance)

Packet Flow with N9 Loopback:

1. eNodeB → SGWU role: GTP packet (TEID=100) arrives on S1-U
2. SGWU role: Matches uplink PDR
3. **Loopback detection:** Destination IP = local IP (10.0.1.10)
4. **In-place processing:** Update GTP TEID to 200 (PGWU session)
5. PGWU role: Decapsulates, forwards to Internet

6. **Total: 1 XDP pass, zero network hops**

Performance benefit: Sub-microsecond internal forwarding vs milliseconds for network round-trip

---

Packet Processing Details

Uplink Flow: eNodeB → SGWU → PGWU → Internet

Code Path: `cmd/ebpf/xdp/n3n6_entrypoint.c` lines 349-403

Key Steps:

1. **Receive:** GTP packet from eNodeB with TEID=100
2. **PDR Match:** Lookup uplink PDR for SGWU session (TEID=100)
3. **FAR Action:** Encapsulate in GTP with TEID=200, forward to 10.0.1.10
4. **Loopback Check:** `is_local_ip(10.0.1.10)` returns TRUE
5. **Update TEID:** Change `ctx->gtp->teid` from 100 to 200 (in kernel memory)
6. **Re-Process:** Lookup PDR for TEID=200 (PGWU session)
7. **FAR Action:** Remove GTP header, forward to Internet
8. **Route:** Send plain IP packet to N6 interface

Downlink Flow: Internet → PGWU → SGWU → eNodeB

Code Path: `cmd/ebpf/xdp/n3n6_entrypoint.c` lines 137-194 (IPv4), 265-322 (IPv6)

Key Steps:

1. **Receive:** Plain IP packet from Internet destined to UE (10.60.0.1)
2. **PDR Match:** Lookup downlink PDR by UE IP (PGWU session)
3. **FAR Action:** Encapsulate in GTP with TEID=200, forward to 10.0.1.10
4. **Loopback Check:** `is_local_ip(10.0.1.10)` returns TRUE
5. **Add GTP:** Encapsulate packet with TEID=200
6. **Re-Process:** Lookup PDR for TEID=200 (SGWU session)
7. **FAR Action:** Update GTP tunnel to eNodeB TEID=100
8. **Route:** Send GTP packet to S1-U interface (eNodeB)

Configuration

Requirements

Control Plane:

- **SGWU-C**: Must connect to OmniUPF PFCP interface (e.g., `192.168.1.10:8805`)
- **PGWU-C**: Must connect to **same** OmniUPF PFCP interface

Network:

- **Single IP address** for both N3 and N9 interfaces
- **Different IP addresses** for SGWU-C and PGWU-C (if running on same host, use different ports)

OmniUPF Configuration

config.yml:

```yaml
# Network interfaces
interface_name: [eth0]              # Single interface for S1-U and
N9
xdp_attach_mode: native             # Use native for best performance

# PFCP Interface
pfcp_address: ":8805"               # Listen on all interfaces, port
8805
pfcp_node_id: "192.168.1.10"        # OmniUPF's PFCP Node ID

# User Plane Interfaces
n3_address: "10.0.1.10"             # S1-U/N3 interface IP
n9_address: "10.0.1.10"             # N9 interface IP (SAME as N3)

# APIs
api_address: ":8080"                # REST API
metrics_address: ":9090"            # Prometheus metrics

# Resource Pools
ueip_pool: "10.60.0.0/16"           # UE IP address pool
teid_pool: 65535                    # TEID allocation pool

# Capacity
max_sessions: 100000                # Maximum concurrent UE sessions
```

Key Configuration:

- ⯑ **`n3_address` and `n9_address` MUST be identical** to enable loopback
- ⯑ Single PFCP listening address for both control planes
- ⯑ Sufficient `max_sessions` for combined SGWU + PGWU load

Control Plane Configuration

SGWU-C Configuration

```
# Point to OmniUPF PFCP interface
upf_pfcp_address: "192.168.1.10:8805"

# S1-U interface (same as OmniUPF n3_address)
sgwu_s1u_address: "10.0.1.10"

# N9 interface for forwarding to PGWU (same as OmniUPF)
sgwu_n9_address: "10.0.1.10"
```

PGWU-C Configuration

```
# Point to SAME OmniUPF PFCP interface
upf_pfcp_address: "192.168.1.10:8805"

# N9 interface (receives from SGWU)
pgwu_n9_address: "10.0.1.10"

# SGi interface for Internet connectivity
pgwu_sgi_address: "192.168.100.1"
```

Important:

- Both control planes connect to **same PFCP endpoint** (`:8805`)
- OmniUPF creates **separate PFCP associations** for SGWU-C and PGWU-C
- Sessions are isolated per control plane (tracked by Node ID)

Session Flow Example

UE Attach and PDU Session Establishment

Scenario: UE attaches to network, establishes data session

PFCP Sessions Created:

SGWU Session (from OmniSGW-C):

- **Uplink PDR:** Match TEID=100 (from eNodeB) → FAR: Encapsulate

TEID=200, dst=10.0.1.10
- **Downlink PDR:** Match TEID=200 (from PGWU) → FAR: Update tunnel TEID=100, forward to eNodeB

PGWU Session (from OmniPGW-C):

- **Uplink PDR:** Match TEID=200 (from SGWU) → FAR: Decapsulate, forward to Internet
- **Downlink PDR:** Match UE IP=10.60.0.1 → FAR: Encapsulate TEID=200, dst=10.0.1.10

Monitoring and Verification

Verify N9 Loopback is Active

Check XDP Logs:

```
# View real-time eBPF debug output
sudo cat /sys/kernel/debug/tracing/trace_pipe | grep loopback
```

Expected output:

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

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

Monitor Sessions via REST API

List PFCP Associations:

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

Expected output:

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

Verify two separate associations (one for SGWU-C, one for PGWU-C)

List Active Sessions:

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

Expected output:

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

Each UE has TWO sessions:

- Session from SGWU-C (TEID=100, S1-U interface)
- Session from PGWU-C (TEID=200, N9 interface)

Performance Metrics

Check Packet Statistics:

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

Key metrics:

- `xdp_processed`: Total packets processed in eBPF

- `xdp_pass`: Packets passed to network stack (should be zero for loopback traffic)
- `xdp_redirect`: Packets forwarded via XDP redirect
- `xdp_tx`: Packets transmitted (loopback traffic uses this)

For N9 loopback traffic:

- `xdp_pass` should be **minimal** (only non-loopback traffic)
- `xdp_tx` or `xdp_redirect` counts loopback forwarding

Troubleshooting

N9 Traffic Going to Network Instead of Loopback

Symptom: Packets sent to network interface, high latency

Root Cause: `n3_address` ≠ `n9_address`

Solution:

```
# WRONG:
n3_address: "10.0.1.10"
n9_address: "10.0.1.20"   # Different IP, no loopback!

# CORRECT:
n3_address: "10.0.1.10"
n9_address: "10.0.1.10"   # Same IP, enables loopback
```

Verification:

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

Should show:

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

PDR Not Found After Loopback

Symptom: Logs show `[n9-loopback] no PDR for destination TEID`

Root Cause: PGWU session not created or TEID mismatch

Diagnosis:

1. **Check PFCP Sessions:**

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

2. **Verify FAR Configuration:**

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

Solution: Ensure PGWU-C creates session with matching TEID that SGWU-C uses for N9 forwarding

---

High CPU Usage

Symptom: CPU usage higher than expected

Root Cause: eBPF program processing packets multiple times or excessive map lookups

Diagnosis:

```
# Check eBPF map access patterns
sudo bpftool map dump name pdr_map_teid_ip4 | wc -l
sudo bpftool map dump name far_map | wc -l
```

Solution:

- Increase `max_sessions` if map is full (causes lookup failures)
- Verify QER rate limiting is not causing drops and retransmits
- Check for excessive packet buffering

Packet Loss During Handover

Symptom: Packets dropped during eNodeB handover

Root Cause: Buffering not configured or insufficient buffer limits

Configuration:

```
buffer_port: 22152
buffer_max_packets: 20000      # Increase for high-mobility networks
buffer_max_total: 100000
buffer_packet_ttl: 30          # Adjust based on handover time
```

Verification:

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

Benefits of N9 Loopback

Performance

Metric	Two Instances	Single Instance (N9 Loopback)	Improvement
Latency	1-5 ms	< 1 μs	**1000x faster**
Throughput	Limited by network	Limited by CPU/memory	**2-3x higher**
CPU Usage	2× XDP passes + network stack	1× XDP pass	**40-50% reduction**
Packet Loss	Risk during network congestion	Zero (in-memory)	**Eliminated**

Operational

- **Simplified Deployment:** Single OmniUPF instance instead of two
- **Reduced Infrastructure:** Half the servers, network ports, IP addresses
- **Lower Complexity:** Single configuration, single monitoring endpoint
- **Cost Savings:** Reduced hardware, power, cooling, maintenance
- **Easier Troubleshooting:** Single packet trace, single eBPF debug output

Use Cases

Ideal For:

- ⍰ **Edge Computing:** Minimize latency for local breakout
- ⍰ **Small/Medium Deployments:** < 100K subscribers
- ⍰ **Lab/Testing:** Full EPC user plane on single VM
- ⍰ **Cost-Constrained:** Limited hardware budget

Not Recommended For:

- ⍰ **Geographic Redundancy:** SGWU and PGWU in different data centers
- ⍰ **Massive Scale:** > 1M subscribers (consider horizontal scaling)
- ⍰ **Regulatory Requirements:** Mandated separation of SGW and PGW

Comparison with Other Deployment Modes

Single Instance (N9 Loopback) vs. Separated Instances

Feature	N9 Loopback	Separated	Containers
Latency	⚡ < 1 µs	� 1-5 ms	� 5-20 ms
Throughput	⚡ 40+ Gbps	� 20+ Gbps	� 10+ Gbps
Infrastructure	� 1 server	� 2 servers	⚠ 1 server, 2 VMs
Complexity	� Simple	� Complex	⚠ Moderate
Cost	� Lowest	� Highest	⚠ Medium
Scaling	⚠ Vertical only	� Horizontal	� Horizontal
Redundancy	� Single point of failure	� Geographic redundancy	⚠ Local redundancy

Summary

N9 Loopback enables **carrier-grade 4G EPC user plane on a single OmniUPF instance** by processing SGWU→PGWU traffic entirely in eBPF without network hops. This provides:

- � **Sub-microsecond latency** for inter-gateway forwarding
- � **40-50% CPU reduction** compared to separated instances
- � **Simplified operations** - single instance, config, monitoring
- � **Lower cost** - half the infrastructure
- � **Full 3GPP compliance** - standard PFCP, GTP-U protocols

Configuration is automatic when `n3_address == n9_address` - no special flags or settings required. OmniUPF's eBPF datapath detects loopback conditions and processes packets inline.

For more information:

- **Configuration:** CONFIGURATION.md
- **Architecture:** ARCHITECTURE.md
- **Operations:** OPERATIONS.md
- **Troubleshooting:** TROUBLESHOOTING.md

Rules Management Guide

Table of Contents

Overview

OmniUPF uses a set of interconnected rules to classify, forward, shape, and track user plane traffic. These rules are installed by the SMF via PFCP and stored in eBPF maps for high-performance packet processing. Understanding these rules and their relationships is critical for operating and troubleshooting the UPF.

Rule Types

Rule Type	Purpose	Key Field	Installed By
PDR (Packet Detection Rule)	Classify packets into flows	TEID or UE IP	SMF via PFCP Session Establishment/Modification
FAR (Forwarding Action Rule)	Determine forwarding action	FAR ID	SMF via PFCP Session Establishment/Modification
QER (QoS Enforcement Rule)	Apply bandwidth limits and marking	QER ID	SMF via PFCP Session Establishment/Modification
URR (Usage Reporting Rule)	Track data volumes for charging	URR ID	SMF via PFCP Session Establishment/Modification

Rule Processing Flow

Packet Detection Rules (PDR)

Purpose

PDRs classify incoming packets into traffic flows. They are the entry point for all packet processing in the UPF.

PDR Structure

Uplink PDRs

Uplink PDRs match packets arriving on the N3 interface from the RAN.

Key Field: TEID (Tunnel Endpoint Identifier)

- 32-bit unsigned integer
- Assigned by SMF and signaled to gNB
- Unique per UE traffic flow

Value Fields:

- **FAR ID**: Reference to forwarding action rule
- **QER ID**: Reference to QoS enforcement rule (optional)
- **URR IDs**: List of usage reporting rules (optional)
- **Outer Header Removal**: Flag to remove GTP-U encapsulation

Lookup Process:

1. Extract TEID from GTP-U header
2. Hash lookup in `uplink_pdr_map` eBPF map
3. If match found, retrieve FAR ID, QER ID, and URR IDs
4. If no match, drop packet

Example:

```
TEID: 5678
FAR ID: 2
QER ID: 1
Outer Header Removal: False
SDF Mode: No SDF
```

Downlink PDRs

Downlink PDRs match packets arriving on the N6 interface from the data network.

Key Field: UE IP Address

- IPv4 address (32-bit) or IPv6 address (128-bit)
- Assigned by SMF during PDU session establishment
- Unique per UE

Value Fields:

- **FAR ID**: Reference to forwarding action rule
- **QER ID**: Reference to QoS enforcement rule (optional)
- **URR IDs**: List of usage reporting rules (optional)
- **SDF Mode**: Service Data Flow filter mode
    - `No SDF`: No filtering, all traffic matches
    - `SDF Only`: Only SDF-matched traffic is forwarded
    - `SDF + Default`: SDF-matched traffic uses specific rules, other traffic uses default FAR
- **SDF Filters**: Application-specific filters (ports, protocols, IP ranges)

Lookup Process:

1. Extract destination IP from packet header
2. Hash lookup in `downlink_pdr_map` (IPv4) or `downlink_pdr_map_ip6` (IPv6)
3. If match found, check SDF filters (if configured)
4. Retrieve FAR ID, QER ID, and URR IDs
5. If no match, drop packet

Example:

```
UE IP: 10.45.0.1
FAR ID: 1
QER ID: 1
Outer Header Removal: False
SDF Mode: No SDF
```

SDF Filters (Service Data Flow)

SDF filters provide application-specific traffic classification within a PDR.

Use Cases:

- Differentiate YouTube traffic from web browsing
- Apply different QoS to VoIP vs. best-effort data
- Route specific applications through different network paths

Filter Criteria:

- **Protocol**: TCP, UDP, ICMP
- **Port Range**: Destination ports (e.g., 443 for HTTPS, 5060 for SIP)
- **IP Address Range**: Specific destination networks
- **Flow Description**: 3GPP-defined flow templates

Example SDF Configuration:

```
PDR ID: 10
UE IP: 10.45.0.1
SDF Mode: SDF Only
SDF Filters:
  - Protocol: UDP, Ports: 5060-5061 → FAR ID 5 (VoIP FAR)
  - Protocol: TCP, Port: 443 → FAR ID 1 (Default FAR)
```

Forwarding Action Rules (FAR)

Purpose

FARs determine what to do with packets that match a PDR. They define forwarding actions, GTP-U encapsulation parameters, and destination endpoints.

FAR Structure

Action Flags

FAR actions are bitwise flags that can be combined:

Flag	Bit	Value	Description
FORWARD	1	2	Forward packet to destination
BUFFER	2	4	Store packet in buffer
DROP	0	1	Discard packet
NOTIFY	3	8	Send notification to control plane
DUPLICATE	4	16	Duplicate packet to multiple destinations

Common Action Combinations:

- `Action: 2 (FORWARD)` - Normal forwarding (most common)
- `Action: 6 (FORWARD + BUFFER)` - Forward and buffer during handover
- `Action: 4 (BUFFER)` - Buffer only (during path switch)
- `Action: 1 (DROP)` - Drop packet (rare, usually for policy enforcement)

Buffering Control

The BUFFER flag (bit 2) controls packet buffering during mobility events.

Buffering Operations:

- **Enable Buffer**: Set bit 2 of FAR action (Action |= 4)
- **Disable Buffer**: Clear bit 2 of FAR action (Action &= ~4)
- **Flush Buffer**: Replay all buffered packets using current FAR rules
- **Clear Buffer**: Discard all buffered packets without forwarding

Outer Header Creation

Determines whether GTP-U encapsulation should be added.

Uplink FAR (N3 → N6):

- Outer Header Creation: False
- Action: Remove GTP-U, forward native IP packet

Downlink FAR (N6 → N3):

- Outer Header Creation: True
- Remote IP: gNB IP address (e.g., 200.198.5.10)
- TEID: Tunnel ID for UE traffic
- Action: Add GTP-U header, forward to gNB

FAR Lookup in Web UI

The Rules Management page provides FAR lookup by ID:

Steps:

1. Navigate to Rules → FARs tab
2. Enter FAR ID in search field
3. Click "Lookup" to view FAR details

Displayed Information:

- FAR ID
- Action (numeric + decoded flags)
- Buffering status (ON/OFF)
- Outer Header Creation
- Remote IP address (with integer representation)
- TEID
- Transport Level Marking

QoS Enforcement Rules (QER)

Purpose

QERs apply Quality of Service parameters to traffic flows, including bandwidth limits and packet marking.

QER Structure

QoS Parameters

QFI (QoS Flow Identifier):

- 6-bit identifier for 5G QoS flows
- Values 1-9 are standardized (e.g., QFI 9 = default bearer)
- Used for packet marking in 5GC

Gate Status:

- **Open (0)**: Traffic allowed
- **Closed (non-zero)**: Traffic blocked

Maximum Bit Rate (MBR):

- Maximum allowed bandwidth for traffic flow
- Specified in kbps
- **MBR = 0**: No rate limit (unlimited)
- Traffic exceeding MBR is dropped

Guaranteed Bit Rate (GBR):

- Minimum bandwidth guaranteed for traffic flow
- Specified in kbps
- **GBR = 0**: Best-effort (no guarantee)
- **GBR > 0**: Prioritized flow with guaranteed bandwidth

QoS Flow Types

Best-Effort Flows (GBR = 0):

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

Guaranteed Flows (GBR > 0):

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

QoS Enforcement Algorithm

Usage Reporting Rules (URR)

Purpose

URRs track data volumes for charging, analytics, and policy enforcement. They maintain packet and byte counters that are reported to the SMF for charging records.

URR Structure

Volume Tracking

Uplink Volume:

- Bytes transmitted from UE to Data Network
- Measured after GTP-U decapsulation
- Includes IP header and payload

Downlink Volume:

- Bytes transmitted from Data Network to UE
- Measured before GTP-U encapsulation
- Includes IP header and payload

Total Volume:

- Sum of uplink and downlink volumes
- Used for total usage reporting

Usage Reporting Triggers

URRs can trigger reports based on:

Volume Threshold:

- Report when volume exceeds configured limit
- Example: Report every 1 GB of usage

Time Threshold:

- Report at periodic intervals
- Example: Report every 5 minutes

Event-Based:

- Report on session termination
- Report on QoS change
- Report on handover

Volume Display Formatting

The Web UI automatically formats volume in human-readable units:

Bytes	Display
0 - 1023	B (Bytes)
1024 - 1048575	KB (Kilobytes)
1048576 - 1073741823	MB (Megabytes)
1073741824 - 1099511627775	GB (Gigabytes)
1099511627776+	TB (Terabytes)

Example:

```
URR ID: 0
Uplink Volume: 12.3 KB
Downlink Volume: 9.0 KB
Total Volume: 21.3 KB
```

URR Reporting Flow

Rule Relationships

PDR → FAR → QER → URR Chain

Each PDR references a FAR, which may reference a QER and one or more URRs.

Example Session Configuration

Uplink PDR:

```
TEID: 5678
FAR ID: 2
QER ID: 1
URR IDs: [0]
Outer Header Removal: False
```

Downlink PDR:

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

FAR ID 1 (Downlink):

```
Action: 2 (FORWARD)
Outer Header Creation: True
Remote IP: 200.198.5.10
TEID: 5678
```

FAR ID 2 (Uplink):

```
Action: 2 (FORWARD)
Outer Header Creation: False
```

QER ID 1:

```
QFI: 9
MBR Uplink: 100000 kbps
MBR Downlink: 100000 kbps
GBR Uplink: 0 kbps
GBR Downlink: 0 kbps
```

URR ID 0:

```
Uplink Volume: 12.3 KB
Downlink Volume: 9.0 KB
Total Volume: 21.3 KB
```

Common Operations

View Rules for a Session

Via Sessions Page:

1. Navigate to Sessions
2. Find UE by IP or TEID
3. Click "Expand" to view all rules (PDR, FAR, QER, URR)

Via Rules Page:

1. Navigate to Rules
2. Use lookup by TEID (uplink) or UE IP (downlink) in PDR tab
3. Note the FAR ID, QER ID, URR IDs
4. Switch to FAR/QER/URR tabs to view referenced rules

Enable/Disable Buffering

Scenario: During handover, buffer packets to prevent loss

Steps:

1. Navigate to Rules → FARs
2. Enter FAR ID in search field
3. Click "Lookup"
4. If buffering is OFF, click "Enable Buffering"
5. Verify FAR action bit 2 is set (Action value increases by 4)

Alternative via Buffers Page:

1. Navigate to Buffers
2. View FARs with buffering enabled
3. Click "Disable Buffer" when handover completes

Monitor QoS Compliance

Check if traffic is being rate-limited:

1. Navigate to Rules → QERs
2. Find QER ID associated with UE session
3. Note MBR Uplink and MBR Downlink values
4. Compare with URR volume growth rate

Calculate Average Throughput:

```
Throughput (kbps) = (Volume Delta in bytes × 8) / (Time Delta in
seconds × 1000)
```

If throughput approaches MBR, traffic is being rate-limited.

Track Data Usage

Monitor URR volumes:

1. Navigate to Rules → URRs
2. View uplink, downlink, and total volumes
3. Sort by Total Volume to find highest users
4. Refresh periodically to observe volume growth

Use Cases:

- Verify charging integration
- Detect abnormal data usage
- Plan capacity based on traffic patterns

Troubleshooting

No Traffic Flowing

Check PDR:

1. Verify PDR exists for TEID (uplink) or UE IP (downlink)
2. Confirm FAR ID is valid
3. Check SDF filters aren't blocking traffic

Check FAR:

1. Verify FAR action is FORWARD (not DROP or BUFFER only)
2. Confirm outer header creation matches direction
3. Verify Remote IP and TEID are correct for downlink

Check QER:

1. Verify Gate Status is Open (0)
2. Check MBR is not too restrictive

Packets Being Dropped

Check QER Rate Limiting:

1. Navigate to Rules → QERs
2. Verify MBR is adequate for traffic load
3. Check URR volume growth matches expected throughput

Check FAR Action:

1. Navigate to Rules → FARs
2. Verify action is FORWARD, not DROP
3. Check buffering isn't stuck in BUFFER-only mode

Buffering Issues

Packets stuck in buffer:

1. Navigate to Buffers page
2. Check oldest packet timestamp
3. If > 30 seconds, handover may have failed
4. Manually flush or clear buffer
5. Disable buffering on FAR

Buffer overflow:

1. Check total packets vs. Max Total (default 100,000)
2. Check per-FAR packets vs. Max Per FAR (default 10,000)
3. Clear buffers if full
4. Investigate why buffering wasn't disabled

URR Not Tracking

Volume counters at zero:

1. Verify PDR references URR ID
2. Check that packets are matching PDR
3. Verify FAR is forwarding (not dropping) packets
4. Confirm URR ID exists in URR map

Volume not reporting to SMF:

1. Check PFCP Session Report configuration
2. Verify URR reporting triggers (volume/time thresholds)
3. Review logs for PFCP Session Report messages

Related Documentation

- **UPF Operations Guide** - Overview of OmniUPF architecture and components
- **PFCP Operations Guide** - PFCP session management and rule installation
- **Web UI Operations Guide** - Control panel usage for rule viewing
- **Monitoring Guide** - Statistics and capacity monitoring
- **Troubleshooting Guide** - Common issues and diagnostics

OmniUPF Troubleshooting Guide

Table of Contents

Overview

This guide provides systematic troubleshooting procedures for common OmniUPF issues. Each section includes symptoms, diagnosis steps, root causes, and resolution procedures.

Quick Diagnostic Checklist

Before deep troubleshooting, verify:

```
# 1. Check OmniUPF is running
systemctl status omniupf  # or ps aux | grep eupf

# 2. Check PFCP association
curl http://localhost:8080/api/v1/upf_pipeline

# 3. Check eBPF maps are loaded
ls /sys/fs/bpf/

# 4. Check XDP program is attached
ip link show | grep -i xdp

# 5. Check kernel logs for errors
dmesg | tail -50
journalctl -u omniupf -n 50
```

Diagnostic Tools

OmniUPF REST API

Check UPF status:

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

Check PFCP associations:

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

Check session count:

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

Check eBPF map capacity:

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

Check packet statistics:

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

Check XDP statistics:

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

---

eBPF Map Inspection

List all eBPF maps:

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

Show map details:

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

Count entries in map:

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

XDP Program Inspection

Check if XDP program is attached:

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

List all XDP programs:

```
bpftool net list
```

Show XDP program details:

```
bpftool prog show
```

Dump XDP statistics:

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

Network Debugging

Capture GTP-U traffic on N3:

```
tcpdump -i eth0 -n udp port 2152 -w /tmp/n3_traffic.pcap
```

Capture PFCP traffic on N4:

```
tcpdump -i eth0 -n udp port 8805 -w /tmp/pfcp_traffic.pcap
```

Monitor packet counters:

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

Check routing table:

```
ip route show
ip route get 10.45.0.100  # Check route for UE IP
```

Check ARP table:

```
ip neigh show
```

Installation Issues

Issue: "eBPF filesystem not mounted"

Symptoms:

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

Cause: eBPF filesystem not mounted

Resolution:

```
# Mount eBPF filesystem
sudo mount bpffs /sys/fs/bpf -t bpf

# Make persistent (add to /etc/fstab)
echo "bpffs /sys/fs/bpf bpf defaults 0 0" | sudo tee -a /etc/fstab

# Verify mount
mount | grep bpf
```

Issue: "Operation not permitted" when loading eBPF

Symptoms:

```
ERRO[0000] failed to load eBPF program: operation not permitted
```

Cause: Insufficient capabilities or locked memory limits

Resolution:

```
# Check current locked memory limit
ulimit -l

# Set unlimited locked memory (required for eBPF)
ulimit -l unlimited

# Make persistent (add to /etc/security/limits.conf)
echo "* soft memlock unlimited" | sudo tee -a /etc/security/
limits.conf
echo "* hard memlock unlimited" | sudo tee -a /etc/security/
limits.conf

# Run OmniUPF with required capabilities
sudo setcap cap_sys_admin,cap_net_admin,cap_bpf+eip /usr/bin/eupf

# Or run with sudo
sudo ./eupf
```

Issue: Kernel version too old

Symptoms:

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

Cause: Linux kernel version below minimum requirement

Resolution:

```
# Check kernel version
uname -r

# Upgrade kernel (Ubuntu/Debian)
sudo apt update
sudo apt install linux-generic-hwe-22.04
sudo reboot

# Verify new kernel
uname -r  # Should be >= 5.15.0
```

Issue: Missing libbpf dependency

Symptoms:

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

Cause: libbpf library not installed

Resolution:

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

# Verify installation
ldconfig -p | grep libbpf
```

Configuration Problems

Issue: Invalid configuration file

Symptoms:

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

Cause: YAML syntax error in config file

Resolution:

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

# Common issues:
# - Incorrect indentation (use spaces, not tabs)
# - Missing colons after keys
# - Unquoted strings with special characters
# - List items without hyphens

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

Issue: Interface name not found

Symptoms:

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

Cause: Configured interface does not exist

Resolution:

```
# List all network interfaces
ip link show

# Check interface status
ip addr show eth0

# If interface has different name, update config.yml:
interface_name: [ens1f0]  # Use actual interface name

# For VMs, check interface naming scheme
ls /sys/class/net/
```

Issue: Port already in use

Symptoms:

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

Cause: Port 8080, 8805, or 9090 already bound by another process

Resolution:

```
# Find process using port
sudo lsof -i :8080
sudo netstat -tulpn | grep :8080

# Kill conflicting process
sudo kill <PID>

# Or change OmniUPF port in config
api_address: :8081
pfcp_address: :8806
metrics_address: :9091
```

Issue: Invalid PFCP Node ID

Symptoms:

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

Cause: PFCP Node ID is not a valid IPv4 address

Resolution:

```
# Correct: Use IP address (not hostname)
pfcp_node_id: 10.100.50.241

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

PFCP Association Issues

Issue: No PFCP associations established

Symptoms:

- Web UI shows "No associations"
- SMF logs show "PFCP Association Setup failure"

Diagnosis:

```
# 1. Check if PFCP server is listening
sudo netstat -ulpn | grep 8805

# 2. Check firewall rules
sudo iptables -L -n | grep 8805
sudo ufw status

# 3. Capture PFCP traffic
tcpdump -i any -n udp port 8805 -vv

# 4. Check PFCP associations via API
curl http://localhost:8080/api/v1/upf_pipeline
```

Common Causes & Resolutions:

Firewall blocking PFCP

Resolution:

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

Wrong PFCP Node ID

Resolution:

```
# Set PFCP Node ID to correct N4 interface IP
pfcp_node_id: 10.100.50.241  # Must match IP on N4 network
```

Network unreachable to SMF

Resolution:

```
# Test connectivity to SMF
ping <SMF_IP>

# Check routing to SMF
ip route get <SMF_IP>

# Add route if missing
sudo ip route add <SMF_NETWORK>/24 via <GATEWAY>
```

SMF configured with wrong UPF IP

Resolution:

- Check SMF configuration for UPF address
- Ensure SMF has UPF's `pfcp_node_id` IP configured
- Verify SMF can route to UPF's N4 network

Issue: PFCP heartbeat failures

Symptoms:

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

Diagnosis:

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

# Monitor heartbeat logs
journalctl -u omniupf -f | grep heartbeat
```

Causes & Resolutions:

Network packet loss

Resolution:

```
# Check packet loss to SMF
ping -c 100 <SMF_IP> | grep loss

# If high loss, investigate network:
# - Check link status
# - Check switch/router health
# - Check for congestion
```

Heartbeat interval too aggressive

Resolution:

```
# Increase heartbeat interval
heartbeat_interval: 30  # Increase from 5 to 30 seconds
heartbeat_retries: 5    # Increase retries
heartbeat_timeout: 10   # Increase timeout
```

Packet Processing Problems

Issue: No packets flowing (RX/TX counters at 0)

Symptoms:

- Statistics page shows 0 RX/TX packets
- UE cannot establish data session

Diagnosis:

```
# 1. Check if XDP program is attached
ip link show eth0 | grep xdp

# 2. Check interface is UP
ip link show eth0

# 3. Capture traffic on interface
tcpdump -i eth0 -n -c 10

# 4. Check packet statistics
curl http://localhost:8080/api/v1/packet_stats
```

Resolutions:

XDP program not attached

Resolution:

```
# Restart OmniUPF to re-attach XDP
sudo systemctl restart omniupf

# Verify attachment
ip link show eth0 | grep xdp
bpftool net list
```

Interface down or no link

Resolution:

```
# Bring interface up
sudo ip link set eth0 up

# Check link status
ethtool eth0 | grep "Link detected"

# If link down, check physical connection or VM network config
```

Wrong interface configured

Resolution:

```
# Update config.yml with correct interface
interface_name: [ens1f0]  # Use actual interface name from 'ip link show'
```

Issue: Packets received but not forwarded (high drop rate)

Symptoms:

- RX counters increasing but TX counters not
- Drop rate > 1%

Diagnosis:

```
# Check drop statistics
curl http://localhost:8080/api/v1/xdp_stats | jq '.drop'

# Check route statistics
curl http://localhost:8080/api/v1/packet_stats | jq '.route_stats'

# Monitor packet drops
watch -n 1 'curl -s http://localhost:8080/api/v1/packet_stats | jq ".total_rx, .total_tx, .total_drop"'
```

Common Causes:

No PDR match (unknown TEID or UE IP)

Resolution:

```
# Check if sessions exist
curl http://localhost:8080/api/v1/sessions

# If no sessions, verify:
# - PFCP association is established
# - SMF has created sessions
# - Session establishment was successful

# Check PDR map entries
bpftool map dump name pdr_map_teid_ip | grep -c key
bpftool map dump name pdr_map_downlin | grep -c key
```

Routing failures

Resolution:

```
# Check FIB lookup failures
curl http://localhost:8080/api/v1/packet_stats | jq '.route_stats'

# Test routing for UE IP
ip route get 10.45.0.100

# Add missing route
sudo ip route add 10.45.0.0/16 dev eth1  # Route UE pool to N6
```

QER rate limiting

Resolution:

```
# Check QER statistics
curl http://localhost:8080/api/v1/sessions | jq '.[].qers'

# If MBR (Maximum Bit Rate) too low, request SMF to update QER
# Or check if traffic exceeds configured rates
```

Issue: One-way traffic (uplink works, downlink doesn't)

Symptoms:

- RX N3 packets but no TX N3 packets (downlink problem)
- RX N6 packets but no TX N6 packets (uplink problem)

Diagnosis:

```
# Check interface statistics
curl http://localhost:8080/api/v1/packet_stats | jq
'.interface_stats'

# Capture traffic on both interfaces
tcpdump -i eth0 -n udp port 2152 &  # N3
tcpdump -i eth1 -n not udp port 2152 &  # N6
```

Uplink Failure (RX N3, no TX N6):

Cause: No FAR action or routing issue to N6

Resolution:

```
# Check FAR has FORWARD action
```

```
curl http://localhost:8080/api/v1/sessions | jq '.[].fars[] |
select(.applied_action == 2)'

# Check N6 route exists
ip route get 8.8.8.8  # Test route to internet

# Add default route if missing
sudo ip route add default via <N6_GATEWAY> dev eth1
```

Downlink Failure (RX N6, no TX N3):

Cause: No downlink PDR or missing GTP encapsulation

Resolution:

```
# Check downlink PDR exists for UE IP
curl http://localhost:8080/api/v1/sessions | jq '.[].pdrs[] |
select(.pdi.ue_ip_address)'

# Verify FAR has OUTER_HEADER_CREATION
curl http://localhost:8080/api/v1/sessions | jq '.[].fars[] |
.outer_header_creation'

# Check gNB reachability
ping <GNB_N3_IP>
```

XDP and eBPF Issues

For detailed XDP configuration, mode selection, and troubleshooting, see the XDP Modes Guide.

Issue: XDP program failed to load

Symptoms:

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

Diagnosis:

```
# Check kernel XDP support
grep XDP /boot/config-$(uname -r)

# Should show:
# CONFIG_XDP_SOCKETS=y
# CONFIG_BPF=y
# CONFIG_BPF_SYSCALL=y
# Check dmesg for detailed error
dmesg | grep -i bpf
```

Causes & Resolutions:

Kernel lacks XDP support

Resolution:

```
# Rebuild kernel with XDP support or upgrade to newer kernel
# Ubuntu 22.04+ has XDP enabled by default
sudo apt install linux-generic-hwe-22.04
sudo reboot
```

XDP program verification failure

Resolution:

```
# Check OmniUPF logs for verifier errors
journalctl -u omniupf | grep verifier

# Common issues:
# - eBPF complexity exceeds limits (increase kernel limits)
# - Invalid memory access (bug in eBPF code)

# Increase eBPF verifier log level for debugging
sudo sysctl kernel.bpf_stats_enabled=1
```

Issue: XDP aborted count increasing

Symptoms:

- XDP stats show aborted > 0
- Packet drops increasing

Diagnosis:

```
# Check XDP aborted count
curl http://localhost:8080/api/v1/xdp_stats | jq '.aborted'

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

Cause: eBPF program encountered runtime error

Resolution:

```
# Check kernel logs for eBPF errors
dmesg | grep -i bpf

# Restart OmniUPF to reload eBPF program
sudo systemctl restart omniupf

# If issue persists, enable eBPF logging (requires rebuild):
# Build OmniUPF with BPF_ENABLE_LOG=1
```

Issue: eBPF map full (capacity exhausted)

Symptoms:

- Session establishment fails
- Map capacity at 100%

Diagnosis:

```
# Check map capacity
curl http://localhost:8080/api/v1/map_info | jq '.[] | {map_name,
capacity, used, usage_percent}'

# Identify full maps
curl http://localhost:8080/api/v1/map_info | jq '.[] |
select(.usage_percent > 90)'
```

Immediate Mitigation:

```
# 1. Identify stale sessions
curl http://localhost:8080/api/v1/sessions | jq '.[] | {seid,
uplink_teid, created_at}'

# 2. Request SMF to delete old sessions
# (via SMF admin interface or API)

# 3. Monitor map usage decrease
watch -n 5 'curl -s http://localhost:8080/api/v1/map_info | jq ".[] |
select(.map_name==\"pdr_map_downlin\") | .usage_percent"'
```

Long-term Resolution:

```
# Increase map capacity in config.yml
max_sessions: 200000  # Increase from 100000

# Or set individual map sizes
pdr_map_size: 400000
far_map_size: 400000
```

```
qer_map_size: 200000
```

Important: Changing map sizes requires OmniUPF restart and **clears all existing sessions**.

Performance Issues

Issue: Low throughput (below expected)

Symptoms:

- Throughput < 1 Gbps despite capable NIC
- High CPU utilization

Diagnosis:

```
# Check packet rate
curl http://localhost:8080/api/v1/packet_stats | jq '.total_rx,
.total_tx'

# Monitor CPU usage
top -bn1 | grep eupf

# Check NIC statistics
ethtool -S eth0 | grep -i drop

# Check XDP mode
ip link show eth0 | grep xdp
```

Resolutions:

Using generic XDP mode

Resolution:

```
# Switch to native mode for better performance
xdp_attach_mode: native  # Requires XDP-capable NIC/driver
```

Single-core bottleneck

Resolution:

```
# Enable RSS (Receive Side Scaling) on NIC
ethtool -L eth0 combined 4  # Use 4 RX/TX queues

# Verify RSS enabled
```

```
ethtool -l eth0

# Pin interrupts to specific CPUs
# See /proc/interrupts and use irqbalance or manual affinity
```

Buffer bloat

Resolution:

```yaml
# Reduce buffer limits to decrease latency
buffer_max_packets: 5000
buffer_packet_ttl: 15
```

Issue: High latency

Symptoms:

- Ping latency > 50ms
- User experience degradation

Diagnosis:

```
# Test latency to UE
ping -c 100 <UE_IP> | grep avg

# Check buffered packets
curl http://localhost:8080/api/v1/upf_buffer_info | jq
'.total_packets_buffered'

# Check route cache performance
curl http://localhost:8080/api/v1/packet_stats | jq '.route_stats'
```

Resolutions:

Packets being buffered excessively

Resolution:

```
# Check why packets are buffered
curl http://localhost:8080/api/v1/upf_buffer_info | jq '.buffers[] |
{far_id, packet_count, direction}'

# Clear buffers if stuck
# (restart OmniUPF or trigger PFCP session modification to apply FAR)
```

FIB lookup latency

Resolution:

```
# Ensure route cache is enabled (build-time option)
# Build with BPF_ENABLE_ROUTE_CACHE=1

# Optimize routing table
# Use fewer, more specific routes instead of many small routes
```

Issue: Packet drops under load

Symptoms:

- Drop rate increases with traffic
- RX errors on NIC

Diagnosis:

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

# Check ring buffer size
ethtool -g eth0

# Monitor drops in real-time
watch -n 1 'ethtool -S eth0 | grep -E "drop|miss"'
```

Resolution:

```
# Increase RX ring buffer size
ethtool -G eth0 rx 4096

# Increase TX ring buffer size
ethtool -G eth0 tx 4096

# Verify new settings
ethtool -g eth0
```

Hypervisor-Specific Issues

For step-by-step hypervisor configuration instructions, see the XDP Modes Guide.

Proxmox: XDP not working in VM

Symptoms:

- Cannot attach XDP program in native mode
- Only generic mode works

Cause: VM using bridged networking without SR-IOV

Resolution:

Option 1: Use generic mode (simplest)

```
xdp_attach_mode: generic
```

Option 2: Configure SR-IOV passthrough

```
# On Proxmox host:
# 1. Enable IOMMU
nano /etc/default/grub
# Add: intel_iommu=on iommu=pt
update-grub
reboot

# 2. Create VFs
echo 4 > /sys/class/net/eth0/device/sriov_numvfs

# 3. Assign VF to VM in Proxmox UI
# Hardware → Add → PCI Device → Select VF

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

VMware: Promiscuous mode required

Symptoms:

- Packets not received by OmniUPF

Cause: vSwitch blocking non-matching MAC addresses

Resolution:

```
# Enable promiscuous mode on vSwitch (in vSphere Client):
# 1. Select vSwitch → Edit Settings
# 2. Security → Promiscuous Mode: Accept
# 3. Security → MAC Address Changes: Accept
# 4. Security → Forged Transmits: Accept
```

VirtualBox: Performance very low

Symptoms:

- Throughput < 100 Mbps

Cause: VirtualBox does not support SR-IOV or native XDP

Resolution:

```
# Use generic mode (only option)
xdp_attach_mode: generic

# Optimize VirtualBox settings:
# - Use VirtIO-Net adapter (if available)
# - Enable "Allow All" promiscuous mode
# - Allocate more CPU cores to VM
# - Use bridged networking instead of NAT

# Consider migrating to KVM/Proxmox for better performance
```

NIC and Driver Issues

Issue: NIC driver does not support XDP

Symptoms:

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

Diagnosis:

```
# Check NIC driver
ethtool -i eth0 | grep driver

# Check if driver supports XDP
modinfo <driver_name> | grep -i xdp

# List XDP-capable interfaces
ip link show | grep -B 1 "xdpgeneric\|xdpdrv\|xdpoffload"
```

Resolution:

Option 1: Use generic mode

```
xdp_attach_mode: generic
```

Option 2: Update NIC driver

```
# Check for driver updates (Ubuntu)
sudo apt update
sudo apt install linux-modules-extra-$(uname -r)

# Or install vendor-specific driver
# Example for Intel:
# Download from https://downloadcenter.intel.com/
```

Option 3: Replace NIC

```
# Use XDP-capable NIC:
# - Intel X710, E810
# - Mellanox ConnectX-5, ConnectX-6
# - Broadcom BCM57xxx (bnxt_en driver)
```

Issue: Driver crashes or kernel panics

Symptoms:

- Kernel panic after attaching XDP
- NIC stops responding

Diagnosis:

```
# Check kernel logs
dmesg | tail -100

# Check for driver bugs
journalctl -k | grep -E "BUG:|panic:"
```

Resolution:

```
# 1. Update kernel and drivers
sudo apt update
sudo apt upgrade
sudo reboot

# 2. Disable XDP offload (use native only)
xdp_attach_mode: native

# 3. Use generic mode as workaround
xdp_attach_mode: generic
```

```
# 4. Report bug to NIC vendor or Linux kernel team
```

Session Establishment Failures

Issue: Session establishment fails

Symptoms:

- SMF reports session establishment failure
- UE cannot establish PDU session

Diagnosis:

```
# Check OmniUPF logs for session errors
journalctl -u omniupf | grep -i "session establishment"

# Check PFCP session count
curl http://localhost:8080/api/v1/sessions | jq 'length'

# Capture PFCP traffic during session establishment
tcpdump -i any -n udp port 8805 -w /tmp/pfcp_session.pcap
```

Common Causes:

Map capacity full

Resolution:

```
# Check map usage
curl http://localhost:8080/api/v1/map_info | jq '.[] |
select(.usage_percent > 90)'

# Increase capacity (see eBPF map full section above)
```

Invalid PDR/FAR parameters

Resolution:

```
# Check OmniUPF logs for validation errors
journalctl -u omniupf | grep -E "invalid|error" | tail -20

# Common issues:
# - Invalid UE IP address (0.0.0.0 or duplicate)
# - Invalid TEID (0 or duplicate)
# - Missing FAR for PDR
# - Invalid FAR action
# Verify SMF configuration and session parameters
```

Feature not supported (UEIP/FTUP)

Resolution:

```
# Enable required features if needed
feature_ueip: true  # UE IP allocation by UPF
ueip_pool: 10.60.0.0/16

feature_ftup: true  # F-TEID allocation by UPF
teid_pool: 100000
```

Buffering Issues

Issue: Packets stuck in buffer

Symptoms:

- Buffered packet count increasing
- Packets not delivered after handover

Diagnosis:

```
# Check buffer statistics
curl http://localhost:8080/api/v1/upf_buffer_info

# Check individual FAR buffers
curl http://localhost:8080/api/v1/upf_buffer_info | jq '.buffers[] |
{far_id, packet_count, oldest_packet_ms}'

# Monitor buffer size
watch -n 5 'curl -s http://localhost:8080/api/v1/upf_buffer_info | jq
".total_packets_buffered"'
```

Causes & Resolutions:

FAR never updated to FORWARD

Cause: SMF never sent PFCP Session Modification to apply FAR

Resolution:

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

# Action BUFF = 1 (buffering)
# Action FORW = 2 (forwarding)

# If stuck in BUFF state, request SMF to:
# - Send PFCP Session Modification Request
# - Update FAR with FORW action
```

Buffer TTL expired

Cause: Packets expired before FAR update

Resolution:

```
# Increase buffer TTL
buffer_packet_ttl: 60  # Increase from 30 to 60 seconds
```

Buffer overflow

Cause: Too many packets buffered per FAR

Resolution:

```
# Increase buffer limits
buffer_max_packets: 20000  # Per FAR
buffer_max_total: 200000   # Global limit
```

Advanced Debugging

Enable Debug Logging

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

```
# Restart OmniUPF with debug logging
sudo systemctl restart omniupf

# Monitor logs in real-time
journalctl -u omniupf -f --output cat
```

---

eBPF Program Tracing

```
# Trace eBPF program execution (requires bpftrace)
sudo bpftrace -e 'tracepoint:xdp:* { @[probe] = count(); }'
```

```
# Trace map operations
sudo bpftrace -e 'tracepoint:bpf:bpf_map_lookup_elem { printf("%s\n",
str(args->map_name)); }'
```

Packet Capture at XDP Level

```
# Capture packets before XDP (tcpdump)
tcpdump -i eth0 -w /tmp/before_xdp.pcap

# Capture packets after XDP (requires XDP_PASS)
tcpdump -i any -w /tmp/after_xdp.pcap

# Compare packet counts to identify drops
```

Getting Help

If troubleshooting steps do not resolve your issue:

1. **Collect diagnostic information**:

   ```
   # System info
   uname -a
   cat /etc/os-release

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

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

   # Network info
   ip addr > /tmp/network.txt
   ip route >> /tmp/network.txt
   ethtool eth0 >> /tmp/network.txt
   ```

2. **Report issue** with:

   - OmniUPF version
   - Linux kernel version
   - Network topology diagram
   - Configuration file (redact sensitive info)
   - Relevant log excerpts
   - Steps to reproduce

3. **Community support**:
    - GitHub Issues: https://github.com/edgecomllc/eupf/issues
    - Documentation: See related guides below

Related Documentation

- **Configuration Guide** - Configuration parameters and examples
- **Architecture Guide** - eBPF/XDP internals and performance tuning
- **Monitoring Guide** - Statistics, capacity, and alerting
- **Rules Management Guide** - PDR, FAR, QER, URR concepts
- **Operations Guide** - UPF architecture and overview

Web UI Operations Guide

Table of Contents

Overview

The OmniUPF Web UI provides a comprehensive control panel for real-time monitoring and management of the User Plane Function. The interface is built on Phoenix LiveView and provides:

- **Real-time visibility** into PFCP sessions and active PDU connections
- **Rules inspection** for PDR, FAR, QER, and URR across all sessions
- **Buffer management** for packet buffering during mobility events
- **Statistics monitoring** for packet processing, routes, and interfaces
- **Capacity tracking** for eBPF map usage and limits
- **Live log viewing** for troubleshooting

Architecture

The control panel communicates with multiple OmniUPF instances via their REST API to:

- Query PFCP sessions and associations
- Inspect packet detection and forwarding rules
- Monitor packet buffers and their status
- Access real-time statistics and performance metrics
- Track eBPF map capacity and utilization

Accessing the Control Panel

Default Access

The control panel is accessible via HTTPS on the OmniUPF management server:

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

Default Port: 443 (HTTPS with self-signed certificate)

Configuration

The control panel requires OmniUPF host configuration in `config/config.exs`:

Multiple UPF instances can be configured for multi-instance deployments:

The `upf_hosts` configuration defines which OmniUPF instances are available in the host selector dropdown throughout the UI.

Navigation

The control panel provides navigation tabs for each operational area:

- **Sessions** - `/sessions` - PFCP sessions and associations
- **Rules** - `/rules` - PDR, FAR, QER, URR rule inspection
- **Buffers** - `/buffers` - Packet buffer monitoring and control
- **Statistics** - `/statistics` - Packet, route, XDP, and interface statistics
- **Capacity** - `/capacity` - eBPF map usage and capacity monitoring
- **Config** - `/upf_config` - UPF configuration and dataplane addresses
- **Routes** - `/routes` - UE routes and routing protocol sessions (OSPF, BGP)
- **XDP Capabilities** - `/xdp_capabilities` - XDP mode support and performance capabilities
- **Logs** - `/logs` - Live log streaming

Sessions View

URL: `/sessions`

Features

The Sessions view displays all active PFCP sessions and associations from selected OmniUPF instances.

PFCP Associations Summary

Displays all active PFCP associations (control connections from SMF/PGW-C):

Column	Description
Node ID	SMF or PGW-C node identifier (FQDN or IP)
Address	SMF/PGW-C IP address for PFCP communication
Next Session ID	Next available PFCP session ID for this association

Purpose:

- Verify SMF connectivity to UPF
- Monitor number of control plane connections
- Track session ID allocation per association

Active Sessions Table

Displays all PFCP sessions representing active UE PDU sessions:

Column	Description
Local SEID	UPF-assigned session endpoint identifier
Remote SEID	SMF-assigned session endpoint identifier
UE IP	User equipment IPv4 or IPv6 address
TEID	GTP-U Tunnel Endpoint Identifier for uplink traffic
PDRs	Number of packet detection rules in session
FARs	Number of forwarding action rules in session
QERs	Number of QoS enforcement rules in session
URRs	Number of usage reporting rules in session
Actions	Expand button to view detailed rule information

Features:

- **Filter by IP**: Find sessions for specific UE IP address
- **Filter by TEID**: Find sessions by tunnel endpoint ID
- **Expand session**: View complete PDR/FAR/QER/URR JSON details
- **Auto-refresh**: Updates every 10 seconds

Expanded Session View:

When you click "Expand" on a session, the view shows:

- **Packet Detection Rules (PDRs)**: Complete JSON with TEID, UE IP, FAR ID, QER ID, SDF filters
- **Forwarding Action Rules (FARs)**: Action flags, outer header creation, destination endpoints
- **QoS Enforcement Rules (QERs)**: MBR, GBR, QFI, and other QoS parameters
- **Usage Reporting Rules (URRs)**: Volume counters (uplink, downlink, total bytes)

Use Cases

Verify UE Connectivity:

1. Navigate to Sessions view
2. Enter UE IP address in filter
3. Confirm session exists with correct TEID
4. Expand to verify PDR/FAR configuration

Monitor Session Count:

- Check total session count in header
- Compare across multiple UPF instances
- Track session growth over time

Troubleshoot Session Issues:

- Search for specific UE IP or TEID
- Expand session to inspect rule configuration
- Verify FAR forwarding parameters
- Check QER QoS settings

Real-time Updates

The Sessions view automatically refreshes every 10 seconds. A health check indicator shows UPF connectivity status:

- **HEALTHY** (green): UPF is reachable and responding
- **UNHEALTHY** (red): UPF is not reachable or not responding
- **UNKNOWN** (gray): Health status not yet determined

Rules Management

URL: `/rules`

The Rules view provides comprehensive inspection of all packet detection, forwarding, QoS, and usage reporting rules across all sessions.

PDR Tab - Packet Detection Rules

View and inspect all PDRs in the UPF:

Uplink PDRs (N3 → N6):

- **TEID**: GTP-U tunnel endpoint ID from gNB
- **FAR ID**: Associated forwarding action rule
- **QER ID**: Associated QoS enforcement rule (if any)

- **Outer Header Removal**: GTP-U decapsulation flag

Downlink PDRs (N6 → N3):

- **UE IP**: IPv4 or IPv6 address of user equipment
- **FAR ID**: Associated forwarding action rule
- **QER ID**: Associated QoS enforcement rule (if any)
- **SDF Mode**: Service data flow filter mode (none, sdf only, sdf + default)

IPv6 PDRs:

- Separate tables for IPv6 uplink and downlink PDRs
- Same structure as IPv4 but keyed by IPv6 addresses

FAR Tab - Forwarding Action Rules

View all FARs with their forwarding actions and parameters:

Column	Description
FAR ID	Unique forwarding rule identifier
Action	Forwarding action flags (FORWARD, DROP, BUFFER, DUPLICATE, NOTIFY)
Buffering	Current buffering status (Enabled/Disabled)
Destination	Outer header creation parameters (TEID, IP address)

FAR Action Flags:

- **FORWARD (1)**: Forward packet to destination
- **DROP (2)**: Discard packet
- **BUFFER (4)**: Store packet in buffer
- **NOTIFY (8)**: Send notification to control plane
- **DUPLICATE (16)**: Duplicate packet to multiple destinations

Buffering Toggle:

- Click "Enable Buffer" or "Disable Buffer" to toggle buffering flag
- Useful for troubleshooting handover scenarios
- Changes FAR action immediately in eBPF map

QER Tab - QoS Enforcement Rules

View QoS rules applied to traffic flows:

Column	Description
QER ID	Unique QoS rule identifier
MBR (Uplink)	Maximum bit rate for uplink traffic (kbps)
MBR (Downlink)	Maximum bit rate for downlink traffic (kbps)

Column	Description
GBR (Uplink)	Guaranteed bit rate for uplink traffic (kbps)
GBR (Downlink)	Guaranteed bit rate for downlink traffic (kbps)
QFI	QoS Flow Identifier (5G marking)

QoS Interpretation:

- **MBR = 0**: No rate limit
- **GBR = 0**: Best-effort (no guaranteed bandwidth)
- **GBR > 0**: Guaranteed bit rate flow (prioritized)

URR Tab - Usage Reporting Rules

View usage tracking rules and volume counters:

Column	Description
URR ID	Unique usage reporting rule identifier
Uplink Volume	Bytes sent from UE to data network
Downlink Volume	Bytes sent from data network to UE
Total Volume	Total bytes in both directions
Method	Reporting method (volume, time, event)

Volume Display:

- Automatically formatted (B, KB, MB, GB, TB)
- Real-time counters updated every refresh
- Used for charging and analytics

Filtering:

- Only shows URRs with non-zero volume
- Limited to 1000 most active URRs for performance

Use Cases

Inspect Traffic Classification:

1. Navigate to Rules → PDR tab
2. Search for specific TEID or UE IP
3. Verify PDR associates with correct FAR and QER

Troubleshoot Forwarding Issues:

1. Navigate to Rules → FAR tab
2. Locate FAR ID from session PDR
3. Verify action is FORWARD (not DROP or BUFFER)
4. Check outer header creation parameters

Monitor QoS Enforcement:

1. Navigate to Rules → QER tab
2. Verify MBR and GBR values match policy
3. Check QFI marking for 5G flows

Track Data Usage:

1. Navigate to Rules → URR tab
2. Sort by total volume to find highest users
3. Monitor volume growth over time
4. Verify charging integration

Buffer Management

URL: `/buffers`

Features

The Buffers view displays packet buffers maintained by the UPF during mobility events or path switches.

Total Statistics

Dashboard displays aggregate buffer statistics:

- **Total Packets**: Number of buffered packets across all FARs
- **Total Bytes**: Total buffered data size
- **Total FARs**: Number of FARs with buffered packets
- **Max Per FAR**: Maximum packets allowed per FAR
- **Max Total**: Maximum total buffered packets
- **Packet TTL**: Time-to-live for buffered packets (seconds)

Buffers by FAR

Table of all FARs with buffered packets:

Column	Description
FAR ID	Forwarding action rule identifier
Packet Count	Number of packets buffered for this FAR
Byte Count	Total bytes buffered for this FAR
Oldest Packet	Timestamp of oldest buffered packet
Newest Packet	Timestamp of newest buffered packet
Actions	Buffer control buttons (pill-style)

Buffer Control Actions

For each FAR with buffered packets, the following pill-style buttons are available:

Buffering Control:

- **Disable Buffer** (red): Turn off buffering for this FAR (updates FAR action flag)
- **Enable Buffer** (purple): Turn on buffering for this FAR

Buffer Operations:

- **Flush** (blue): Replay all buffered packets using current FAR rules
- **Clear** (gray): Delete all buffered packets without forwarding

Clear All Buffers:

- Red "Clear All" button in header
- Clears buffers for all FARs
- Requires confirmation

Use Cases

Monitor Handover Buffering:

1. During handover, verify packets are being buffered
2. Check FAR buffering status (should be enabled)
3. Monitor packet count and age

Complete Handover:

1. After path switch, click "Flush" to replay buffered packets
2. Verify packets are forwarded to new path
3. Click "Disable Buffer" to stop buffering

Clear Stuck Buffers:

1. Identify FARs with old buffered packets (check oldest timestamp)
2. Click "Clear" to discard stale packets
3. Or click "Disable Buffer" to prevent further buffering

Troubleshoot Buffer Overflow:

1. Check total packet count vs. max total
2. Identify FARs with excessive buffering
3. Verify SMF has sent session modification to disable buffering
4. Manually disable buffering if SMF command missed

Real-time Updates

The Buffers view automatically refreshes every 5 seconds to show current buffer status.

Statistics Dashboard

URL: `/statistics`

Features

The Statistics view provides real-time performance metrics from the OmniUPF datapath.

Packet Statistics

Aggregate packet processing counters:

- **RX Packets**: Total packets received on all interfaces
- **TX Packets**: Total packets transmitted on all interfaces
- **Dropped Packets**: Packets discarded due to errors or policy
- **GTP-U Packets**: Packets processed with GTP-U encapsulation

Use: Monitor overall UPF traffic load and packet drop rate

Route Statistics

Per-route forwarding metrics (if available):

- **Route hits**: Packets matched by each routing rule
- **Forwarding success**: Successfully forwarded packet count
- **Forwarding errors**: Failed forwarding attempts

Use: Identify busy routes and forwarding errors

XDP Statistics

eXpress Data Path performance metrics:

- **XDP Processed**: Total packets processed at XDP layer
- **XDP Passed**: Packets sent to network stack
- **XDP Dropped**: Packets dropped at XDP layer
- **XDP Aborted**: Processing errors in XDP program

Use: Monitor XDP performance and detect processing errors

XDP Drop Causes:

- Invalid packet format
- eBPF map lookup failure
- Policy-based drops
- Resource exhaustion

N3/N6 Interface Statistics

Per-interface traffic counters:

N3 Interface (RAN connectivity):

- **RX N3**: Packets received from gNB/eNodeB
- **TX N3**: Packets transmitted to gNB/eNodeB

N6 Interface (Data Network connectivity):

- **RX N6**: Packets received from data network (Internet/IMS)
- **TX N6**: Packets transmitted to data network

Total: Aggregate packet count across interfaces

Use: Monitor traffic balance and interface-specific issues

Use Cases

Monitor Traffic Load:

1. Check packet RX/TX rates
2. Verify traffic is flowing in both directions
3. Compare N3 vs N6 traffic (should be roughly equal)

Detect Packet Drops:

1. Check dropped packet counter
2. Review XDP dropped counter
3. Investigate cause in logs if drops are high

Performance Analysis:

1. Monitor XDP processed vs. passed ratio
2. Check for XDP aborts (indicates errors)
3. Verify N3/N6 interface traffic distribution

Capacity Planning:

1. Track packet rate over time
2. Compare to UPF capacity limits
3. Plan for scaling if approaching limits

Real-time Updates

Statistics automatically refresh every 10 seconds.

Capacity Monitoring

URL: `/capacity`

Features

The Capacity view displays eBPF map usage and capacity limits for all maps in the UPF datapath.

eBPF Map Usage Table

Table of all eBPF maps with usage information:

Column	Description
Map Name	eBPF map name (e.g., `uplink_pdr_map`, `far_map`)
Used	Number of entries currently in map
Capacity	Maximum entries allowed in map
Usage	Visual progress bar with percentage
Key Size	Size of map keys in bytes
Value Size	Size of map values in bytes

Color-Coded Usage Indicators

The usage progress bar is color-coded based on utilization:

- **Green (<50%)**: Normal operation, ample capacity
- **Yellow (50-70%)**: Caution, monitor growth
- **Amber (70-90%)**: Warning, plan capacity increase
- **Red (>90%)**: Critical, immediate action required

Critical Maps to Monitor

uplink_pdr_map:

- Stores uplink PDRs keyed by TEID
- One entry per uplink traffic flow
- **Critical**: Exhaustion prevents new session establishment

downlink_pdr_map / downlink_pdr_map_ip6:

- Stores downlink PDRs keyed by UE IP address
- One entry per UE IPv4/IPv6 address

- **Critical**: Exhaustion prevents new session establishment

far_map:

- Stores forwarding action rules keyed by FAR ID
- Shared across multiple PDRs
- **High Priority**: Affects forwarding decisions

qer_map:

- Stores QoS enforcement rules keyed by QER ID
- **Medium Priority**: Affects QoS but not basic connectivity

urr_map:

- Stores usage reporting rules keyed by URR ID
- **Low Priority**: Affects charging but not connectivity

Use Cases

Capacity Planning:

1. Monitor map usage trends over time
2. Identify which maps are growing fastest
3. Plan capacity increases before reaching limits

Prevent Session Establishment Failures:

1. Check PDR map usage before expected traffic surge
2. Increase map capacity if approaching limits
3. Monitor after capacity increase to verify

Troubleshoot Session Failures:

1. When session establishment fails, check Capacity view
2. If PDR maps are red (>90%), capacity is exhausted
3. Increase map capacity or clear stale sessions

Optimize Map Configuration:

1. Review key and value sizes
2. Calculate memory usage per map
3. Optimize map sizes based on actual usage patterns

Capacity Configuration

eBPF map capacities are configured at UPF startup in the UPF configuration file. Typical values:

- Small deployment: 10,000 - 100,000 entries per map
- Medium deployment: 100,000 - 1,000,000 entries per map
- Large deployment: 1,000,000+ entries per map

Memory Calculation:

```
Map Memory = (Key Size + Value Size) × Capacity
```

For example, a PDR map with 1 million entries and 64-byte values uses approximately 64 MB of kernel memory.

Real-time Updates

Capacity view automatically refreshes every 10 seconds.

Configuration View

URL: `/upf_config`

Features

The Configuration view displays UPF operational parameters and dataplane configuration.

UPF Configuration

Displays static UPF configuration:

- **PFCP Interface**: IP address and port for SMF/PGW-C connectivity
- **N3 Interface**: IP address for RAN (gNB/eNodeB) connectivity
- **N6 Interface**: IP address for data network connectivity
- **N9 Interface**: IP address for inter-UPF communication (optional)
- **API Port**: REST API listening port
- **Version**: OmniUPF software version

Dataplane (eBPF) Configuration

Displays active runtime dataplane parameters:

- **Active N3 Address**: Runtime N3 interface binding
- **Active N9 Address**: Runtime N9 interface binding (if enabled)

These values reflect the actual eBPF datapath configuration and may differ from static configuration if interfaces have been changed.

Use Cases

Verify UPF Connectivity:

1. Check N3 interface IP matches gNB configuration
2. Verify N6 interface can route to data network
3. Confirm PFCP interface is reachable from SMF

Troubleshoot Interface Issues:

1. Compare static config with dataplane active addresses
2. Verify interfaces are bound correctly
3. Check for interface configuration changes

Documentation and Audit:

1. Record UPF configuration for documentation
2. Verify deployment matches design specifications
3. Audit interface assignments

Routes View

URL: `/routes`

Features

The Routes view provides comprehensive monitoring of User Equipment (UE) IP routes and routing protocol sessions (OSPF and BGP).

Route Status Overview

Dashboard displays aggregate route statistics:

- **Status**: Routing enabled or disabled
- **Total Routes**: Total number of UE IP routes
- **Synced**: Number of successfully synced routes
- **Failed**: Number of routes that failed to sync

Active UE IP Routes

Table displaying all active User Equipment IP routes:

Column	Description
Index	Route index number
UE IP Address	IPv4 or IPv6 address assigned to the UE

Purpose:

- View all UE IP addresses that have routes configured
- Verify route distribution to routing protocols
- Monitor route synchronization status

OSPF Neighbors

Table of OSPF (Open Shortest Path First) protocol neighbors:

Column	Description
Neighbor ID	OSPF router identifier
Address	IP address of the OSPF neighbor
Interface	Interface used for OSPF adjacency
State	OSPF adjacency state (Full, Init, etc.)
Priority	OSPF priority value
Up Time	Duration the neighbor has been up
Dead Time	Time until neighbor is considered dead

OSPF States:

- **Full** (green): Fully adjacent and exchanging routing information
- **Other states** (yellow): Adjacency forming or incomplete

BGP Peers

Table of BGP (Border Gateway Protocol) peers:

Column	Description
Neighbor IP	IP address of the BGP peer
ASN	Autonomous System Number of the peer
State	BGP session state (Established, Idle, etc.)
Up/Down	Duration of current state
Prefixes Received	Number of route prefixes received from peer
Msg Sent	Total BGP messages sent to peer
Msg Rcvd	Total BGP messages received from peer

BGP States:

- **Established** (green): Active BGP session, exchanging routes
- **Other states** (red): Session down or establishing

The header also displays the local BGP Router ID and ASN when BGP is configured.

OSPF Redistributed Routes

Table showing OSPF External LSAs (Link State Advertisements) for redistributed

UE routes:

Column	Description
Link State ID	LSA identifier (typically the network address)
Mask	Network mask for the route
Advertising Router	Router ID advertising this external route
Metric Type	OSPF external metric type (E1 or E2)
Metric	OSPF cost metric for the route
Age	Time since LSA was originated (seconds)
Seq Number	LSA sequence number for versioning

Purpose:

- Verify UE routes are being redistributed into OSPF
- Monitor which router is advertising external routes
- Track LSA aging and updates

Route Control Actions

Sync Routes Button:

- Manually triggers route synchronization to FRR (Free Range Routing)
- Forces update of routing protocol with current UE routes
- Useful after configuration changes or to recover from sync failures

Refresh Button:

- Manually refresh all route information
- Updates OSPF neighbors, BGP peers, and route tables

Use Cases

Monitor Routing Protocol Health:

1. Navigate to Routes view
2. Check OSPF neighbor states (should be "Full")
3. Verify BGP peers are "Established"
4. Confirm expected number of neighbors/peers

Verify UE Route Distribution:

1. Check Active UE IP Routes table for specific UE
2. Scroll to OSPF Redistributed Routes section
3. Verify UE route appears in external LSAs
4. Confirm advertising router matches expected UPF

Troubleshoot Route Sync Issues:

1. Check Synced vs. Failed counters in status overview
2. If routes are failing, click "Sync Routes" button
3. Monitor error messages in red banner if sync fails
4. Check OSPF/BGP error messages in respective sections

Verify Multi-UPF Deployment:

1. Select different UPF instances from dropdown
2. Compare route counts across instances
3. Verify OSPF neighbors see each other
4. Check BGP peering relationships

Monitor Route Scaling:

1. Track total route count as UE sessions increase
2. Verify routes are distributed to routing protocols
3. Monitor OSPF LSA count growth
4. Check BGP prefix count received by peers

Real-time Updates

The Routes view automatically refreshes every 10 seconds to show current routing protocol status and UE routes.

Routing Integration

The Routes view integrates with FRR (Free Range Routing) running on the UPF:

- **OSPF**: Routes are redistributed as External Type-2 LSAs
- **BGP**: Routes are advertised to configured BGP peers
- **Sync mechanism**: REST API calls trigger vtysh commands to update FRR

XDP Capabilities View

URL: `/xdp_capabilities`

Features

The XDP Capabilities view displays eXpress Data Path (XDP) mode support, performance capabilities, and throughput calculations for the UPF dataplane.

Interface Configuration

Displays network interface and driver information:

Field	Description
Interface Name	Network interface used for XDP (e.g., eth0, ens1f0)

Field	Description
Driver	Network driver name (e.g., i40e, ixgbe, virtio_net)
Driver Version	Driver version string
Current Mode	Active XDP mode (DRV, SKB, or NONE)
Multi-Queue Count	Number of NIC queue pairs for parallel processing

XDP Modes

The view displays all XDP modes with their support status and performance characteristics:

XDP_DRV (Driver Mode):

- **Performance**: ~5-10 Mpps (millions of packets per second)
- **Description**: Native XDP support in driver, highest performance
- **Requires**: NIC driver with native XDP support (i40e, ixgbe, mlx5, etc.)
- **Status**: Supported if driver has XDP hooks
- **Indicator**: Green checkmark (✓) if supported, red X (✗) if not

XDP_SKB (Generic Mode):

- **Performance**: ~1-2 Mpps
- **Description**: Fallback mode using kernel network stack
- **Requires**: Any network interface
- **Status**: Always supported
- **Indicator**: Green checkmark (✓)

Current Mode Indicator:

- Blue dot next to the currently active XDP mode
- Shows which mode is actually in use

Unsupported Mode Reasons:

- If a mode is unsupported, the "Reason" field explains why
- Common reasons: driver lacks XDP support, interface type incompatibility

XDP Capabilities view showing interface configuration, supported modes, and the interactive Mpps throughput calculator

Recommendations

The view displays a colored recommendation banner based on current configuration:

Green (Optimal):

- "✓ Optimal: XDP_DRV mode enabled with native driver support"
- Highest performance mode is active

Yellow (Warning):

- "⚠ Consider upgrading to XDP_DRV mode for better performance"
- Running in generic mode when driver mode is available
- "⚠ Warning: XDP_DRV not supported by this driver"
- Hardware limitations prevent optimal performance

Blue (Informational):

- General information about XDP configuration

Mpps Performance Calculator

Interactive calculator to convert packet rate (Mpps) to throughput (Gbps):

Input Parameters

Packet Rate (Mpps):

- Range: 0.1 - 100 Mpps
- Default: Maximum Mpps for current XDP mode
- Represents millions of packets processed per second

Average Packet Size (bytes):

- Range: 64 - 9000 bytes
- Default: 1200 bytes (typical GTP packet)
- Includes full packet with GTP encapsulation

Quick Preset Buttons:

- **64B (min)**: Minimum Ethernet frame size
- **128B**: Small packets
- **256B**: Control plane or signaling
- **512B**: Medium-sized packets
- **1024B**: Large packets
- **1518B (max)**: Maximum Ethernet frame size without jumbo frames

Calculation Results

Total Throughput (Gbps):

- Wire-rate throughput including all headers
- Formula: `Gbps = Mpps × Packet_Size × 8 / 1000`
- Includes GTP, UDP, IP, and Ethernet headers

User Data Rate (Gbps):

- Actual user payload throughput
- Excludes ~50 bytes GTP encapsulation overhead
- Formula: `Gbps = Mpps × (Packet_Size - 50) / 1000`

Packet Rate:

- Displays Mpps and packets/sec with thousands separator
- Example: 10 Mpps = 10,000,000 packets/sec

Formula Display:

- Shows calculation breakdown step-by-step
- Example: `10 Mpps × 1200 bytes × 8 bits/byte ÷ 1000 = 96 Gbps`

Understanding Mpps

The view includes an explanation section covering:

What is Mpps:

- Millions of Packets Per Second
- Key metric for packet processing performance
- Independent of packet size

Relationship to Throughput:

- Same Mpps with larger packets = higher Gbps
- Same Mpps with smaller packets = lower Gbps
- Throughput depends on both rate and packet size

GTP Encapsulation Overhead:

- Ethernet header: 14 bytes
- IP header: 20 bytes (IPv4) or 40 bytes (IPv6)
- UDP header: 8 bytes
- GTP header: 8 bytes (minimum)
- Total typical overhead: ~50 bytes per packet

Use Cases

Evaluate XDP Performance:

1. Navigate to XDP Capabilities view
2. Check current XDP mode (should be DRV for best performance)
3. Note the Mpps performance range
4. Review recommendation banner

Calculate Expected Throughput:

1. Enter expected packet rate in Mpps
2. Enter average packet size for your traffic profile
3. Review calculated throughput in Gbps
4. Compare to link capacity or performance requirements

Optimize XDP Configuration:

1. Check if XDP_DRV mode is supported but not active
2. Review driver version and compatibility
3. Follow recommendation to upgrade to driver mode if available
4. Verify multi-queue count matches CPU cores

Capacity Planning:

1. Use calculator to determine required Mpps for target throughput
2. Compare to current XDP mode capabilities
3. Determine if hardware upgrade needed
4. Plan interface and driver selection for new deployments

Troubleshoot Performance Issues:

1. Verify XDP mode is DRV, not SKB
2. Check driver version for known performance issues
3. Verify multi-queue count is sufficient
4. Calculate if current mode supports required throughput

Performance Optimization Tips

Driver Mode (XDP_DRV):

- Use NICs with native XDP support (Intel i40e/ixgbe, Mellanox mlx5)
- Update NIC drivers to latest version
- Enable multi-queue (RSS) for parallel processing
- Tune NIC ring buffer sizes

Generic Mode (XDP_SKB):

- Acceptable for development and testing
- Not recommended for production high-throughput
- Consider hardware upgrade for production deployments

Multi-Queue Configuration:

- Number of queues should match or exceed CPU core count
- Enables parallel packet processing across cores
- Distributes load via RSS (Receive Side Scaling)

Real-time Updates

XDP Capabilities view refreshes every 30 seconds to update interface status and mode information.

Logs Viewer

URL: `/logs`

Features

View OmniUPF application logs in real-time from the control panel.

Features:

- Live log streaming via Phoenix LiveView
- Real-time updates as logs are generated
- Scrollable log history
- Useful for troubleshooting during active sessions

Log Levels

OmniUPF logs use standard Elixir Logger levels:

- **DEBUG**: Detailed diagnostic information
- **INFO**: General informational messages (default)
- **WARNING**: Warning messages for non-critical issues
- **ERROR**: Error messages for failures

Use Cases

Troubleshoot Session Establishment:

1. Open Logs view
2. Initiate session establishment from SMF
3. Watch for PFCP message logs and any errors

Monitor PFCP Communication:

1. View PFCP association setup messages
2. Track session creation/modification/deletion
3. Verify heartbeat messages

Debug Forwarding Issues:

1. Look for packet processing errors
2. Check eBPF map operation logs

3. Identify FAR/PDR configuration issues

Best Practices

Operational Guidelines

Monitoring:

- Regularly check Capacity view to prevent map exhaustion
- Monitor Statistics for unusual traffic patterns or drops
- Track session count growth over time
- Watch for XDP processing errors

Buffer Management:

- Monitor buffers during handover scenarios
- Clear stuck buffers if packets age beyond TTL
- Verify buffering is disabled after handover completes
- Use "Flush" instead of "Clear" to avoid packet loss

Session Management:

- Use filters to quickly locate specific UE sessions
- Expand sessions to verify rule configuration
- Compare sessions across multiple UPF instances
- Check health indicator before troubleshooting

Troubleshooting:

- Use Logs for real-time debugging
- Check Sessions view to verify UE connectivity
- Verify Rules configuration for traffic flows
- Monitor Statistics for packet drops or forwarding errors

Performance

- Control panel auto-refresh is 5-10 seconds depending on view
- Large session lists may take time to load
- Rules view filters by active entries (non-zero volumes for URRs)
- Buffer operations execute immediately on selected UPF

Related Documentation

- **PFCP Operations Guide** - PFCP session management and protocol details
- **Rules Management Guide** - PDR, FAR, QER, URR configuration
- **Monitoring Guide** - Statistics, metrics, and capacity planning
- **Routes Guide** - UE routing and FRR integration details

- **XDP Modes Guide** - Detailed XDP mode documentation and eBPF information
- **Troubleshooting Guide** - Common issues and diagnostics
- **UPF Operations Guide** - General UPF operations and architecture

XDP Attachment Modes for OmniUPF

Table of Contents

Overview

OmniUPF uses **XDP (eXpress Data Path)** for high-performance packet processing. XDP is a Linux kernel technology that allows packet processing programs (eBPF) to run at the earliest possible point in the network stack, providing microsecond-level latency and millions of packets per second throughput.

The XDP attachment mode determines **where** in the packet path the eBPF program executes:

Choosing the right XDP mode significantly impacts OmniUPF performance and determines whether you can achieve production-grade packet processing.

XDP Mode Comparison

Aspect	Generic Mode	Native Mode	Offload Mode
Attach Point	Linux network stack	Network driver	NIC hardware
Performance	~1-2 Mpps	~5-10 Mpps	~10-40 Mpps
Latency	~100 µs	~10 µs	~1 µs
CPU Usage	High	Medium	Low
NIC Requirements	Any NIC	XDP-capable driver	SmartNIC with XDP support

Aspect	Generic Mode	Native Mode	Offload Mode
Hypervisor Support	All hypervisors	Most (requires multi-queue)	Rare (PCI passthrough)
Use Case	Testing, development	**Production (recommended)**	High-throughput edge sites
Configuration	`xdp_attach_mode: generic`	`xdp_attach_mode: native`	`xdp_attach_mode: offload`

Recommendation: Use **native mode** for production deployments. Generic mode is only suitable for testing.

Generic Mode (Default)

Description

Generic XDP runs the eBPF program in the Linux network stack **after** the driver has processed the packet. This is the slowest XDP mode but works with any network interface.

Performance Characteristics

- **Throughput**: ~1-2 million packets per second (Mpps)
- **Latency**: ~100 microseconds per packet
- **CPU Overhead**: High (packet copied to kernel stack before XDP)

When to Use

- **Development and testing** only
- **Lab environments** where performance doesn't matter
- **Initial deployment** to verify functionality before optimizing

Configuration

```yaml
# config.yaml
interface_name: [eth0]
xdp_attach_mode: generic  # Default mode
```

Warning: Generic mode is **not suitable for production**. It will bottleneck at high packet rates and waste CPU resources.

Native Mode (Recommended for Production)

Description

Native XDP runs the eBPF program **inside the network driver**, before packets reach the Linux network stack. This provides near-hardware performance while maintaining kernel-level flexibility.

Performance Characteristics

- **Throughput**: ~5-10 million packets per second (Mpps) per core
- **Latency**: ~10 microseconds per packet
- **CPU Overhead**: Low (packet processed at driver level)
- **Scaling**: Linear scaling with CPU cores and NIC queues

When to Use

- **Production deployments** (recommended)
- **Carrier-grade networks** requiring high throughput
- **Edge computing** scenarios with performance requirements
- **Any deployment** where performance matters

NIC Driver Requirements

Native XDP requires a network driver with XDP support. Most modern NICs support native XDP:

Physical NICs (bare metal):

- Intel: `ixgbe` (10G), `i40e` (40G), `ice` (100G)
- Broadcom: `bnxt_en`
- Mellanox: `mlx4_en`, `mlx5_core`
- Netronome: `nfp` (with offload support)
- Marvell: `mvneta`, `mvpp2`

Virtual NICs (hypervisors):

- VirtIO: `virtio_net` (KVM, Proxmox, OpenStack) ✓
- VMware: `vmxnet3` ✓
- Microsoft: `hv_netvsc` (Hyper-V) ✓
- Amazon: `ena` (AWS) ✓
- SR-IOV: `ixgbevf`, `i40evf` (PCI passthrough) ✓

Note: VirtualBox does **not** support native XDP (use generic mode only).

Configuration

```yaml
# config.yaml
interface_name: [eth0]
xdp_attach_mode: native
```

Multi-Queue Requirement: For optimal performance, enable multi-queue on virtual NICs (see Proxmox section below).

Offload Mode (SmartNIC)

Description

Offload XDP runs the eBPF program **directly on the NIC hardware** (SmartNIC), completely bypassing the CPU for packet processing. This provides the highest performance but requires specialized hardware.

Performance Characteristics

- **Throughput**: ~10-40 million packets per second (Mpps)
- **Latency**: ~1 microsecond per packet
- **CPU Overhead**: Near-zero (processing on NIC)

When to Use

- **Ultra-high-throughput** deployments (10G+ per UPF instance)
- **Edge sites** with hardware acceleration
- **Cost-sensitive** deployments (reduce CPU requirements)

Hardware Requirements

Only Netronome Agilio SmartNICs currently support XDP offload:

- Netronome Agilio CX 10G/25G/40G/100G

Note: Offload mode requires **bare metal** or **PCI passthrough** - not available in standard VM configurations.

Configuration

```yaml
# config.yaml
interface_name: [eth0]
xdp_attach_mode: offload
```

Enabling Native XDP on Proxmox VE

Proxmox VE uses **VirtIO** network devices for VMs, which support native XDP via the `virtio_net` driver. However, you must enable **multi-queue** for optimal performance.

Step 1: Understanding the Requirement

Why Multi-Queue Matters:

- **Single queue** (default): All network traffic processed by one CPU core → bottleneck
- **Multi-queue**: Traffic distributed across multiple CPU cores → linear scaling

Step 2: Enable Multi-Queue in Proxmox

Option A: Via Proxmox Web UI

1. **Shutdown the VM completely** (not just reboot)

    - Select your VM in the Proxmox web interface
    - Click **Shutdown**

2. **Edit Network Device**

    - Go to **Hardware** tab
    - Click on your network device (e.g., `net0`)
    - Click **Edit**

3. **Set Multiqueue**

    - Find the **"Multiqueue"** field
    - Set to **8** (or match your vCPU count, max 16)
    - Click **OK**

4. **Start the VM**

    - Click **Start**

Option B: Via Proxmox Command Line

```
# SSH to your Proxmox host

# Find your VM ID
qm list
```

```
# Set multi-queue (replace XXX with your VM ID)
qm set XXX -net0 virtio=XX:XX:XX:XX:XX:XX,bridge=vmbr0,queues=8

# Example for VM 191 with MAC BC:24:11:1D:BA:00
qm set 191 -net0 virtio=BC:24:11:1D:BA:00,bridge=vmbr0,queues=8

# Shutdown the VM
qm shutdown XXX

# Wait for shutdown, then start
qm start XXX
```

Queue Count Recommendations:

- **4 queues**: Minimum for production (good for 2-4 vCPU VMs)
- **8 queues**: Recommended for most deployments (4-8 vCPU VMs)
- **16 queues**: Maximum for high-performance (8+ vCPU VMs)

Step 3: Verify Multi-Queue Inside VM

After VM restart, SSH into the VM and verify:

```
# Check queue configuration
ethtool -l eth0

# Expected output:
# Channel parameters for eth0:
# Combined:        8        <-- Should match your configured value

# Count actual queues
ls -1d /sys/class/net/eth0/queues/rx-* | wc -l
ls -1d /sys/class/net/eth0/queues/tx-* | wc -l

# Both should show 8 (or your configured value)
```

Step 4: Enable Native XDP in OmniUPF

Edit the OmniUPF configuration:

```
# Edit config file
sudo nano /etc/eupf/config.yaml
```

Change XDP mode:

```
# Before
xdp_attach_mode: generic

# After
```

```
xdp_attach_mode: native
```

Restart OmniUPF:

```
sudo systemctl restart eupf
```

Step 5: Verify Native XDP is Active

Check logs:

```
# View startup logs
journalctl -u eupf --since "1 minute ago" | grep -i "xdp\|attach"

# Expected output:
# xdp_attach_mode:native
# XDPAttachMode:native
# Attached XDP program to iface "eth0" (index 2)
```

Check via API:

```
# Query configuration
curl -s http://localhost:8080/api/v1/config | grep xdp_attach_mode

# Expected output:
# "xdp_attach_mode": "native",
```

Common Proxmox Issues

Issue: "Failed to attach XDP program"

Solution:

- Verify multi-queue is enabled (`ethtool -l eth0`)
- Check kernel version: `uname -r` (must be ≥ 5.15)
- Ensure VirtIO driver loaded: `lsmod | grep virtio_net`

Issue: Only 1 queue despite configuration

Solution:

- VM must be **fully shutdown** (not rebooted) for queue changes
- Use `qm shutdown XXX && sleep 5 && qm start XXX`
- Verify in Proxmox config: `grep net0 /etc/pve/qemu-server/XXX.conf`

Issue: Performance not improving with native mode

Solution:

- Check CPU pinning (avoid oversubscription)

- Monitor `top` - CPU usage should spread across cores
- Verify XDP stats: `curl http://localhost:8080/api/v1/xdp_stats`

Enabling Native XDP on Other Hypervisors

VMware ESXi / vSphere

VMware uses `vmxnet3` driver which supports native XDP.

Requirements:

- ESXi 6.7 or later
- vmxnet3 driver version 1.4.16+ in VM
- VM hardware version 14 or later

Enable Multi-Queue:

1. **Power off the VM**

2. **Edit VM settings**:

    - Right-click VM → Edit Settings
    - Network Adapter → Advanced
    - Set **Receive Side Scaling** to **Enabled**

3. **Edit .vmx file** (optional, for more queues):

    ```
    ethernet0.pnicFeatures = "4"
    ethernet0.multiqueue = "8"
    ```

4. **Start VM and verify**:

    ```
    ethtool -l ens192  # Check queue count
    ```

Configure OmniUPF:

```
interface_name: [ens192]  # VMware typically uses ens192
xdp_attach_mode: native
```

KVM / libvirt (Raw)

Enable Multi-Queue via virsh:

```
# Edit VM configuration
virsh edit your-vm-name
```

Add to network interface section:

```
<interface type='network'>
  <source network='default'/>
  <model type='virtio'/>
  <driver name='vhost' queues='8'/>
</interface>
```

Restart VM and verify:

```
ethtool -l eth0
```

Microsoft Hyper-V

Hyper-V uses `hv_netvsc` driver which supports native XDP.

Requirements:

- Windows Server 2016 or later
- Linux Integration Services 4.3+ in VM
- Generation 2 VM

Enable Multi-Queue:

PowerShell on Hyper-V host:

```
# Set VMQ (Virtual Machine Queue) - Hyper-V's multi-queue
Set-VMNetworkAdapter -VMName "YourVM" -VrssEnabled $true -VmmqEnabled
$true
```

Configure OmniUPF:

```
interface_name: [eth0]
xdp_attach_mode: native
```

VirtualBox

Warning: VirtualBox does **NOT** support native XDP.

Reason: VirtualBox network drivers (e1000, virtio-net) do not implement XDP hooks.

Workaround: Use generic mode only:

```
xdp_attach_mode: generic  # Only option for VirtualBox
```

Verifying XDP Mode

After configuring native XDP, verify it's working correctly:

1. Check OmniUPF Logs

```
# View recent logs
journalctl -u eupf --since "5 minutes ago" | grep -i xdp

# Look for:
# ✓ "xdp_attach_mode:native"
# ✓ "Attached XDP program to iface"
# ✗ "Failed to attach" or "falling back to generic"
```

2. Check via API

```
# Query configuration endpoint
curl -s http://localhost:8080/api/v1/config | jq .xdp_attach_mode

# Expected output:
# "native"
```

3. Check XDP Statistics

```
# View XDP processing stats
curl -s http://localhost:8080/api/v1/xdp_stats | jq

# Example output:
{
  "xdp_aborted": 0,      # Should be 0 (errors)
  "xdp_drop": 1234,      # Dropped packets
  "xdp_pass": 5678,      # Passed to stack
  "xdp_redirect": 9012,  # Redirected packets
  "xdp_tx": 3456         # Transmitted packets
}
```

4. Verify Driver Support

```
# Check if driver supports XDP
ethtool -i eth0 | grep driver

# For Proxmox/KVM: Should show "virtio_net"
# For VMware: Should show "vmxnet3"
# For Hyper-V: Should show "hv_netvsc"
```

5. Performance Test

Compare packet processing before and after:

```
# Monitor packet rate
watch -n 1 'curl -s http://localhost:8080/api/v1/packet_stats | jq
.rx_packets'
```

```
# Generic mode: ~1-2 Mpps
# Native mode: ~5-10 Mpps (5-10x improvement)
```

Troubleshooting XDP Issues

Issue: "Failed to attach XDP program" on Startup

Symptoms:

```
Error: failed to attach XDP program to interface eth0
```

Diagnosis:

1. **Check driver support**:

   ```
   ethtool -i eth0 | grep driver

   # If driver is not virtio_net/vmxnet3/hv_netvsc, native XDP
   won't work
   ```

2. **Check kernel version**:

   ```
   uname -r

   # Must be >= 5.15 for reliable XDP support
   ```

3. **Check for existing XDP programs**:

   ```
   ip link show eth0 | grep xdp

   # If another XDP program is attached, unload it first
   ip link set dev eth0 xdp off
   ```

Solution:

- Update kernel to 5.15+ if older
- Ensure virtio_net driver is loaded: `modprobe virtio_net`
- Fall back to generic mode if driver doesn't support native XDP

Issue: Native Mode Falls Back to Generic

Symptoms:

```
Warning: falling back to generic XDP mode
```

Diagnosis:

Check `dmesg` for driver errors:

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

Common causes:

1. **Driver doesn't support native XDP**:
    - VirtualBox drivers (no native XDP support)
    - Older NIC drivers
2. **Multi-queue not enabled**:
    - Check: `ethtool -l eth0`
    - Should show > 1 combined queue
3. **Kernel XDP support disabled**:

    ```
    # Check if XDP is enabled in kernel
    grep XDP /boot/config-$(uname -r)

    # Should show:
    # CONFIG_XDP_SOCKETS=y
    # CONFIG_BPF=y
    ```

Solution:

- Enable multi-queue (see Proxmox section)
- Update to supported driver
- Rebuild kernel with XDP support if necessary

Issue: Performance Not Improving with Native Mode

Symptoms: Native mode enabled but packet rate same as generic mode

Diagnosis:

1. **Verify multi-queue distribution**:

    ```
    # Check per-queue statistics
    ethtool -S eth0 | grep rx_queue

    # Traffic should be distributed across multiple queues
    ```

2. **Check CPU utilization**:

```
# Monitor CPU usage per core
mpstat -P ALL 1

# Should see load spread across multiple CPUs
```

3. **Verify XDP is actually running in native mode**:

   ```
   # Check bpftool (if available)
   sudo bpftool net list

   # Should show XDP attached to interface
   ```

Solution:

- Increase queue count (8-16 queues)
- Enable CPU pinning to prevent core migration
- Check for CPU oversubscription on hypervisor

Issue: XDP Program Aborted (xdp_aborted > 0)

Symptoms:

```
curl http://localhost:8080/api/v1/xdp_stats
{
  "xdp_aborted": 1234,  # Non-zero indicates errors
  ...
}
```

Diagnosis:

XDP aborted means the eBPF program hit an error during execution.

1. **Check eBPF verifier logs**:

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

2. **Check for map size limits**:

   ```
   # eBPF maps may be full
   curl http://localhost:8080/api/v1/map_info

   # Look for maps at 100% capacity
   ```

Solution:

- Increase eBPF map sizes in configuration
- Check for corrupted packets causing eBPF errors
- Verify Linux kernel eBPF support is complete

Issue: Multi-Queue Not Working on Proxmox

Symptoms: `ethtool -l eth0` shows only 1 queue despite configuration

Diagnosis:

1. **Check Proxmox VM config**:

   ```
   # On Proxmox host
   grep net0 /etc/pve/qemu-server/YOUR_VM_ID.conf

   # Should show: queues=8
   ```

2. **Verify VM was fully shutdown**:

   ```
   # On Proxmox host
   qm status YOUR_VM_ID

   # Must show "status: stopped" before starting
   ```

Solution:

```
# On Proxmox host
# Force shutdown and restart
qm shutdown YOUR_VM_ID
sleep 10
qm start YOUR_VM_ID

# Then check inside VM
ethtool -l eth0
```

Important: Changes to queue count require a **full VM shutdown**, not just a reboot from inside the VM.

Issue: Permission Denied When Attaching XDP

Symptoms:

```
Error: permission denied when attaching XDP program
```

Diagnosis:

XDP operations require `CAP_NET_ADMIN` and `CAP_SYS_ADMIN` capabilities.

Solution:

1. **Run OmniUPF as root** (or with capabilities):

   ```
   sudo systemctl restart eupf
   ```

2. **If using systemd**, verify service file has capabilities:

   ```
   # /lib/systemd/system/eupf.service
   [Service]
   CapabilityBoundingSet=CAP_NET_ADMIN CAP_SYS_ADMIN CAP_NET_RAW
   AmbientCapabilities=CAP_NET_ADMIN CAP_SYS_ADMIN CAP_NET_RAW
   ```

3. **If using Docker**, run with `--privileged`:

   ```
   docker run --privileged -v /sys/fs/bpf:/sys/fs/bpf ...
   ```

Performance Impact Summary

Real-world performance comparison for OmniUPF packet processing:

Scenario	Generic Mode	Native Mode	Improvement
Packet Rate	1.5 Mpps	8.2 Mpps	**5.5x faster**
Latency	95 µs	12 µs	**8x lower**
CPU Usage (1 Gbps)	85% (1 core)	15% (distributed)	**5x more efficient**
Max Throughput	~1.2 Gbps	~10 Gbps	**8x higher**

Recommendation: Always use **native mode** with **multi-queue enabled** for production deployments.

Hardware Recommendations for XDP

⚠ IMPORTANT: Before purchasing any hardware, consult with Omnitouch support to confirm it's 100% compatible with your specific configuration and deployment requirements.

Known Good NICs for Native XDP

These NICs are verified to support native XDP mode with OmniUPF:

Intel NICs (Recommended for Bare Metal)

Model	Speed	Driver	XDP Support	Notes
Intel X520	10GbE	ixgbe	Native ✓	Proven, widely available, good price/ performance

Model	Speed	Driver	XDP Support	Notes
Intel X710	10/40GbE	i40e	Native ✓	Excellent multi-queue support
Intel E810	100GbE	ice	Native ✓	Latest generation, best performance
Intel i350	1GbE	igb	Native ✓ (kernel 5.10+)	Good for lower bandwidth needs

Mellanox/NVIDIA NICs (High Performance)

Model	Speed	Driver	XDP Support	Notes
ConnectX-4	25/50/100GbE	mlx5	Native ✓	High throughput, good for edge computing
ConnectX-5	25/50/100GbE	mlx5	Native ✓	Excellent performance, hardware acceleration
ConnectX-6	50/100/200GbE	mlx5	Native ✓	Latest generation, best for ultra-high throughput
BlueField-2	100/200GbE	mlx5	Native ✓	SmartNIC with DPU capabilities

Broadcom NICs

Model	Speed	Driver	XDP Support	Notes
BCM57xxx series	10/25/50GbE	bnxt_en	Native ✓	Common in Dell/HP servers

Virtual NICs (VM Deployments)

Platform	NIC Type	Driver	XDP Support	Multi-Queue	Notes
Proxmox/KVM	VirtIO	virtio_net	Native ✓	Yes (configurable)	**Best for VMs**
VMware ESXi	vmxnet3	vmxnet3	Native ✓	Yes	Requires ESXi 6.7+
Hyper-V	Synthetic NIC	hv_netvsc	Native ✓	Yes	Windows Server 2016+
AWS	ENA	ena	Native ✓	Yes	EC2 metal instances
VirtualBox	Any	various	Generic only ⚠	No	Not recommended for production

NICs with Hardware Offload Support

True XDP hardware offload (eBPF runs on NIC):

Vendor	Model	Speed	Notes
Netronome	Agilio CX 10G	10GbE	Only confirmed XDP offload support
Netronome	Agilio CX 25G	25GbE	Requires special firmware
Netronome	Agilio CX 40G	40GbE	Very expensive (~$2,500-5,000)
Netronome	Agilio CX 100G	100GbE	Enterprise-grade only

Note: Hardware offload NICs are rare, expensive, and require bare metal deployment. Most deployments should use native XDP instead.

Tested Configurations

These configurations have been verified with OmniUPF in production:

Budget Option (1-10 Gbps)

- **NIC**: Intel X520 (10GbE dual-port)
- **Mode**: Native XDP
- **Throughput**: ~8-10 Gbps per UPF instance
- **Cost**: ~$100-200 (used/refurbished)

Mid-Range (10-50 Gbps)

- **NIC**: Intel X710 (40GbE) or Mellanox ConnectX-4 (25GbE)
- **Mode**: Native XDP
- **Throughput**: ~25-40 Gbps per UPF instance
- **Cost**: ~$300-800

High-End (50-100+ Gbps)

- **NIC**: Mellanox ConnectX-5/6 (100GbE)
- **Mode**: Native XDP
- **Throughput**: ~80-100 Gbps per UPF instance
- **Cost**: ~$1,000-2,500

VM Deployments (Proxmox/KVM)

- **NIC**: VirtIO with 8-16 queues
- **Mode**: Native XDP
- **Throughput**: ~5-10 Gbps per UPF instance
- **Cost**: No additional hardware cost

What NOT to Buy

Avoid these for production OmniUPF deployments:

NIC/Platform	Reason	Alternative
Realtek NICs	No XDP support, poor Linux drivers	Intel i350 or better
VirtualBox	No native XDP support	Migrate to Proxmox/KVM
Consumer-grade NICs	Limited queue support, unreliable	Server-grade Intel/ Mellanox
Very old NICs (<2014)	No XDP driver support	Intel X520 or newer

Pre-Purchase Checklist

Before buying hardware, verify:

1. ⯑ **Driver Support**: Check if Linux driver supports XDP

   ```
   # On similar system
   modinfo <driver_name> | grep -i xdp
   ```

2. ⯑ **Kernel Version**: Ensure kernel ≥ 5.15 for reliable XDP

   ```
   uname -r
   ```

3. ⯑ **Multi-Queue**: Verify NIC supports multiple queues (RSS/VMDq)

4. ⯑ **PCI Bandwidth**: Ensure PCIe slot has sufficient lanes

   - 10GbE: PCIe 2.0 x4 minimum
   - 40GbE: PCIe 3.0 x8 minimum
   - 100GbE: PCIe 3.0 x16 or PCIe 4.0 x8

5. ⯑ **Deployment Type**:

   - Bare metal: Physical NIC required
   - VM: VirtIO or SR-IOV support needed
   - Container: Host NIC configuration inherited

⚠ Don't buy hardware based solely on this guide - always confirm with Omnitouch support first!

Additional Resources

- **Configuration Guide**: CONFIGURATION.md - Complete configuration reference
- **Troubleshooting Guide**: TROUBLESHOOTING.md - Comprehensive problem diagnosis
- **Architecture Guide**: ARCHITECTURE.md - eBPF and XDP architecture

details
- **Monitoring Guide**: MONITORING.md - Performance monitoring and statistics

---

Quick Reference

Proxmox Native XDP Setup (TL;DR)

```
# On Proxmox host:
qm set <VM_ID> -net0 virtio=<MAC>,bridge=vmbr0,queues=8
qm shutdown <VM_ID> && sleep 10 && qm start <VM_ID>

# Inside VM:
ethtool -l eth0  # Verify 8 queues
sudo nano /etc/eupf/config.yaml  # Set: xdp_attach_mode: native
sudo systemctl restart eupf
journalctl -u eupf --since "1 min ago" | grep xdp  # Verify native
mode
```

Verify XDP Mode is Active

```
# Check configuration
curl -s http://localhost:8080/api/v1/config | grep xdp_attach_mode

# Check statistics
curl -s http://localhost:8080/api/v1/xdp_stats | jq

# Check queues
ethtool -l eth0
```

OmniUPF API Documentation

Overview

The OmniUPF API provides a RESTful interface for managing and monitoring the eBPF-based User Plane Function. The API enables real-time control and observability of:

- **PFCP Sessions**: Session lifecycle and association management
- **Packet Detection Rules (PDR)**: Traffic classification for uplink and downlink (IPv4 and IPv6)
- **Forwarding Action Rules (FAR)**: Packet forwarding, buffering, and dropping actions
- **QoS Enforcement Rules (QER)**: Quality of Service policies and rate limiting
- **Usage Reporting Rules (URR)**: Data volume tracking and reporting
- **Packet Buffers**: Packet buffering and replay functionality
- **Statistics**: Real-time metrics for packets, routes, XDP, and N3/N6 interfaces
- **Route Management**: UE route synchronization with FRR routing daemon
- **Configuration**: UPF and dataplane configuration management

Swagger API Documentation

The API is fully documented using **OpenAPI 3.0 (Swagger)** specification. The interactive Swagger UI provides:

- Complete endpoint documentation with request/response schemas
- Try-it-out functionality for testing API calls directly from the browser
- Schema definitions for all data models
- HTTP status codes and error responses

Interactive Swagger UI showing the OmniUPF API endpoints with detailed documentation.

Accessing Swagger UI

The Swagger documentation is available at:

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

For example: `http://10.98.0.20:8080/swagger/index.html`

API Base Path

All API endpoints are prefixed with:

```
/api/v1
```

```

## See Also

- [UE Route Management Documentation](./routes.md) - Detailed guide
on FRR integration and route synchronization
- [Operations Guide](../OPERATIONS.md) - Web UI operations and
monitoring
- [Swagger UI](http://10.98.0.20:8080/swagger/index.html) -
Interactive API documentation
```

UE Route Management

Related Documentation:

- API Documentation - Complete API reference including route management endpoints
- Operations Guide - Web UI operations and monitoring

Overview

The UPF (User Plane Function) integrates with **FRR (Free Range Routing)** to dynamically manage User Equipment (UE) IP routes. This integration ensures that as UE sessions are established or terminated, the routing infrastructure automatically adapts to reflect the current network topology.

What is FRR?

FRR (Free Range Routing) is a robust, open-source routing protocol suite for Linux and Unix platforms. It implements various routing protocols including BGP, OSPF, RIP, and others. In our deployment, FRR acts as the routing daemon that maintains the kernel routing table and can redistribute routes to other network elements.

Architecture

How Route Synchronization Works

Route Lifecycle

Automatic Synchronization

The UPF maintains an internal registry of all active UE IP addresses. When enabled, the route synchronization system:

1. **Monitors UE Sessions**: Tracks all active PFCP sessions and their associated UE IP addresses
2. **Maintains Route List**: Keeps an up-to-date list of routes that need to be in the routing table
3. **Syncs to FRR**: Automatically pushes route updates to the FRR daemon via its API
4. **Handles Failures**: Tracks sync status (synced/failed) for each route and retries as needed

FRR Setup

Configuration

FRR is deployed and configured using **Ansible templates** to establish the base routing parameters. You define the FRR configuration once as a **Jinja2 template** in your Ansible playbook, and Ansible automatically propagates it to all your UPF instances during deployment.

A typical FRR Jinja2 configuration template includes:

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

Deployment Model:

1. **Define Once**: Create the FRR Jinja2 template in your Ansible role (e.g., `roles/frr/templates/frr.conf.j2`)
2. **Configure Parameters**: Set variables in your Ansible inventory for each UPF host
3. **Deploy Everywhere**: Run the Ansible playbook to deploy FRR configuration to all UPF nodes
4. **Automatic Customization**: Ansible uses host-specific variables (IP addresses, router IDs, etc.) to customize each UPF's FRR configuration

Customizable Parameters in the Jinja2 template:

- **OSPF parameters**: Router ID, area configuration, authentication methods, network advertisements
- **BGP configuration**: ASN, neighbor relationships, route policies, communities
- **Route redistribution**: Which kernel routes to redistribute (e.g., `redistribute kernel`)
- **Route filtering**: Route maps, prefix lists, access lists
- **Interface settings**: OSPF/BGP interface parameters

UPF Integration: Once the base FRR configuration is deployed to each UPF instance, the UPF dynamically adds UE IP addresses as **/32 host routes** to the kernel routing table based on active PFCP sessions. These routes are then:

1. **Installed in the kernel routing table** by the UPF route sync engine
2. **Picked up by FRR** via the `redistribute kernel` directive
3. **Advertised to routing protocols** (OSPF, BGP) according to your FRR configuration
4. **Propagated to the network** so that UE traffic can be routed to this UPF instance

Key Points:

- **Set Once, Deploy Everywhere**: Define the FRR Jinja2 template once in Ansible, and it's automatically deployed to all UPF instances
- **Ansible handles static config**: The Jinja2 template sets up all routing protocol parameters (OSPF areas, BGP neighbors, authentication, route policies, etc.)
- **UPF handles dynamic routes**: Each UPF instance dynamically manages only the UE IP /32 routes based on its active PFCP sessions
- **Automatic route advertisement**: FRR on each UPF automatically redistributes the local UE routes according to your configured policies
- **Centralized management**: Update the Ansible template and re-run the playbook to change routing configuration across all UPFs simultaneously

Route Advertisement

Monitoring and Management

Web UI Integration

The UPF Control Panel provides a **Routes** page that displays:

- **Route Status**: Whether route synchronization is enabled or disabled
- **Total Routes**: Number of UE IP addresses being tracked
- **Sync Statistics**: Count of successfully synced routes and any failures

- **Active Routes**: Real-time list of all UE IP addresses currently in the routing table
- **OSPF Neighbors**: Live status of OSPF adjacencies with neighbor details
- **BGP Peers**: BGP session status and prefix statistics (when configured)
- **OSPF Redistributed Routes**: Complete view of external LSAs showing how UE routes are advertised

The Routes page provides comprehensive visibility into UE route synchronization, routing protocol neighbors, and redistributed route advertisements.

Manual Sync Operation

Administrators can trigger a manual route synchronization through the web UI using the **Sync Routes** button. This operation:

1. Re-reads the current list of active UE sessions from the UPF
2. Compares with FRR's routing table
3. Adds any missing routes
4. Removes any stale routes
5. Returns updated sync statistics

Route Flow

Benefits

- **Zero Touch Provisioning**: Routes are automatically managed without manual intervention
- **Dynamic Adaptation**: Network routing adapts in real-time to UE mobility and session changes
- **Scalability**: Supports thousands of concurrent UE routes
- **Resilience**: Failed sync operations are tracked and can be retried
- **Visibility**: Full visibility into route status through the web UI

Technical Details

API Endpoints

The UPF exposes the following route management endpoints:

- `GET /api/v1/routes` - List all tracked UE routes without syncing
- `POST /api/v1/routes/sync` - Sync routes to FRR and return updated list
- `GET /api/v1/route_stats` - Get detailed routing statistics
- `GET /api/v1/routing/sessions` - Get routing protocol sessions (OSPF neighbors, BGP peers)
- `GET /api/v1/ospf/database/external` - Get OSPF AS-External LSA database (redistributed routes)

> **See Also**: API Documentation - Route Management for complete endpoint details and examples

Route Format

Routes are stored and managed as simple IP addresses (e.g., `100.64.18.5`). The routing daemon handles the full route entry details including:

- Destination prefix/mask
- Gateway/next-hop
- Interface binding
- Metric and administrative distance

FRR Verification

OSPF External LSA Database

You can verify that UE routes are being properly redistributed into OSPF by examining the FRR OSPF Link State Database. External LSAs (Type 5) show routes that have been injected into OSPF from external sources.

FRR OSPF database showing external LSAs including UE route 100.64.18.5/32 being advertised as an E2 (External Type 2) route.

In the example above, you can see:

- **Network LSA (10.98.0.20)**: The UPF's own network advertisement
- **Router LSA (192.168.1.1)**: OSPF router advertisement
- **External LSAs**: Including the UE route `100.64.18.5` redistributed into OSPF with metric type E2 (External Type 2)

This verification confirms that:

1. The UPF is successfully tracking the UE IP address
2. The route sync engine has pushed the route to FRR
3. FRR has redistributed the route into OSPF
4. OSPF neighbors are receiving the route advertisements