REST API Guide

← Back to Main Documentation

This guide provides comprehensive documentation for the OmniSS7 **REST API** and **Swagger UI**.

Table of Contents

Overview

OmniSS7 exposes a REST API on **port 8080** for driving MAP (Mobile Application Part) and CAP (CAMEL Application Part) operations over the SS7 network, plus raw SCCP/M3UA injection for testing. The API handler (`APIhandler`, `lib/ss7_web/apihandler.ex`) defines **47 `POST /api/*` operation endpoints** alongside the metrics and Swagger endpoints. It allows you to:

- Trigger MAP requests (mobility, SMS, authentication, subscriber data, supplementary services, location services, GPRS, equipment)
- Drive CAP/CAMEL call-control dialogues

- Inject raw SCCP/M3UA payloads
- Monitor system metrics via Prometheus

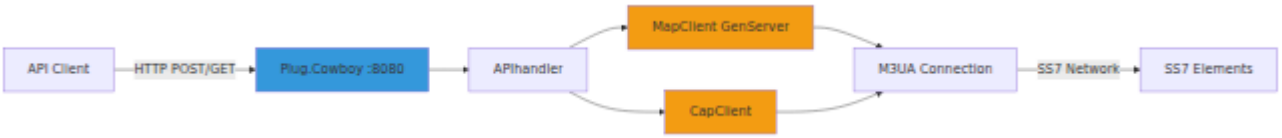
For per-operation request/response schemas and MAP opcodes, this guide cross-references the **MAP Client Guide**, which documents every operation in detail.

Ports and Transports

OmniSS7 runs **three distinct HTTP servers**. This guide covers the first one (:8080). The others are documented in the Configuration Reference.

Port	Transport	Server	Scope of this guide
8080	HTTP — `Plug.Cowboy`	`APIhandler`	**Yes** — MAP/CAP REST `/api/*`, `/metrics`, `/swagger`.
8087	HTTPS — `Bandit` (Phoenix)	`ControlPanelWeb.Endpoint`	No — web control panel UI. See Web UI Guide.
8445	HTTPS / TLS	`:api_ex` REST controllers	No — status/routing/links REST. See Configuration Reference.

API Architecture



HTTP Server Configuration

Server Details

The `/api/*`, `/metrics`, and `/swagger` endpoints are served by a `Plug.Cowboy` HTTP listener (`lib/application.ex`).

Parameter	Value	Configurable
Protocol	HTTP	No
IP Address	0.0.0.0 (all interfaces)	Via code only
Port	8080	Via code only
Transport	`Plug.Cowboy`	No

Access URL: `http://[server-ip]:8080`

> The control panel (:8087, Bandit) and the `:api_ex` REST controllers (:8445, TLS) are separate servers — see Ports and Transports.

Enabling/Disabling the HTTP Server

Control whether the :8080 HTTP server starts:

```
config :omniss7,
  start_http_server: true  # Set to false to disable
```

Default: `true` (enabled)

When Disabled: The :8080 HTTP server will not start, and the REST API / Swagger UI / metrics endpoint will be unavailable. The control panel (:8087) and `:api_ex` (:8445) servers are unaffected (they are gated by `start_web_interfaces`).

Swagger UI

The API includes a **Swagger UI** for interactive API documentation and testing.

Accessing Swagger UI

URL: `http://[server-ip]:8080/swagger`

Features:

- Interactive API documentation
- Try-it-out functionality for testing endpoints
- Request/response schemas
- Example payloads

Swagger JSON

The OpenAPI specification is available at:

URL: `http://[server-ip]:8080/swagger.json`

Use Cases:

- Import into Postman or other API clients
- Generate client libraries
- API documentation automation

API Endpoints

Every operation endpoint follows the pattern `POST /api/{operation}` and accepts a JSON request body. Requests are **synchronous**: the handler blocks until the SS7 response arrives or the operation times out (default **10 seconds** for awaited MAP operations — see Error Handling).

The tables below list all **47** operation endpoints grouped by category. For each operation's request fields, response schema, and MAP opcode, see the

corresponding section of the **MAP Client Guide**, which was refreshed alongside this guide and documents every operation in full.

> **Response codes (all endpoints):** `200` = success, `400` = bad/invalid request (missing or wrong-typed fields), `504` = timeout (no SS7 response within the operation's timeout), `500` = operation error (non-timeout), `503` = feature not enabled (USSD), `404` = unknown path. See Error Handling.

Mobility Management

Operation	Endpoint	Description
Update Location	`POST /api/updateLocation`	Register a subscriber's serving VLR with the HLR. In HLR mode this triggers the InsertSubscriberData (ISD) sequence.
Cancel Location	`POST /api/cancelLocation`	Instruct a VLR to delete a subscriber record.
Provide Roaming Number	`POST /api/prn`	Obtain an MSRN from the serving MSC.
Purge MS	`POST /api/purgeMS`	Mark a subscriber as purged at the HLR.
Send Identification	`POST /api/sendIdentification`	Retrieve IMSI/auth data from the previous VLR (TMSI supplied as hex).
Reset	`POST /api/reset`	HLR-initiated reset toward a VLR (unconfirmed — returns `{"status":"sent"}`).
Restore Data	`POST /api/restoreData`	VLR requests subscriber-data restoration from the HLR.

SMS

Operation	Endpoint	Description
Send Routing Info for SM	`POST /api/sri-for-sm`	Query the HLR for the serving node for SMS delivery.
MT-Forward SM	`POST /api/MT-forwardSM`	Deliver a mobile-terminated SM to the serving MSC/SGSN (`smsPDU` as hex).
MO-Forward SM	`POST /api/forwardSM`	Submit a mobile-originated SM (`smsPDU` as hex).
Send SM	`POST /api/sendSM`	Build and deliver an SMS-DELIVER from GSM-7 text.
Build Deliver PDU	`POST /api/deliverPDU`	Build an SMS-DELIVER PDU and return the hex (no SS7 send).
Report SM Delivery Status	`POST /api/reportSM-DeliveryStatus`	Report SM delivery outcome to the HLR.
Ready for SM	`POST /api/readyForSM`	Notify the HLR that an MS is reachable for SMS.
Alert Service Centre	`POST /api/alertServiceCentre`	Alert an SMSC that a subscriber is available.

Authentication

Operation	Endpoint	Description
Send Authentication Info	`POST /api/send-auth-info`	Retrieve authentication vectors from the HLR.
Send IMSI	`POST /api/sendIMSI`	Resolve an MSISDN to its IMSI at the HLR.

Subscriber Data

Operation	Endpoint	Description
Send Routing Info	`POST /api/sri`	Query the HLR for voice-call routing information.
Provide Subscriber Info	`POST /api/provideSubscriberInfo`	Request subscriber state/location from the VLR.
Delete Subscriber Data	`POST /api/deleteSubscriberData`	Remove subscriber data at the VLR (optional `withdraw` flags).
Any-Time Interrogation	`POST /api/anyTimeInterrogation`	gsmSCF queries subscriber info/location at the HLR.
Any-Time Subscription Interrogation	`POST /api/anyTimeSubscriptionInterrogation`	Query subscription data at the HLR.
Any-Time Modification	`POST /api/anyTimeModification`	Modify subscription data at the HLR.

Operation	Endpoint	Description
Note Subscriber Data Modified	`POST /api/noteSubscriberDataModified`	HLR notifies the gsmSCF of changed data.
Note MM-Event	`POST /api/noteMM-Event`	Report a mobility-management event to the gsmSCF.

Supplementary Services

Operation	Endpoint	Description
Interrogate SS	`POST /api/interrogateSS`	Interrogate the status of a supplementary service.
Register SS	`POST /api/registerSS`	Register a supplementary service (e.g. call forwarding / CFU).
Erase SS	`POST /api/eraseSS`	Erase a supplementary service.
Activate SS	`POST /api/activateSS`	Activate a supplementary service.
Deactivate SS	`POST /api/deactivateSS`	Deactivate a supplementary service.
Register Password	`POST /api/registerPassword`	Register an SS password (`ss_code` integer).
Get Password	`POST /api/getPassword`	Retrieve/verify an SS password.
Forward Check SS Indication	`POST /api/forwardCheckSS-Indication`	Unconfirmed check toward a VLR (returns `{"status":"sent"}`).

The five core SS operations (`interrogateSS`, `registerSS`, `eraseSS`, `activateSS`, `deactivateSS`) share a request shape: `ss_code` (integer) and `hlr_gt`, with optional `forwarded_to`, `no_reply_time`, and `basic_service`.

Location Services (LCS)

Operation	Endpoint	Description
Send Routing Info for LCS	`POST /api/sendRoutingInfoForLCS`	GMLC queries the HLR for LCS routing.
Provide Subscriber Location	`POST /api/provideSubscriberLocation`	Request a subscriber's location from the serving node.
Subscriber Location Report	`POST /api/subscriberLocationReport`	Report a subscriber's location to the GMLC.

GPRS

Operation	Endpoint	Description
Update GPRS Location	`POST /api/updateGprsLocation`	Register a subscriber's serving SGSN with the HLR.
Send Routing Info for GPRS	`POST /api/sendRoutingInfoForGprs`	Query the HLR for GPRS routing (GGSN selection).

Equipment

Operation	Endpoint	Description
Check IMEI	`POST /api/checkIMEI`	Query equipment status at the EIR.

USSD

Operation	Endpoint	Description
USSD Send (network-originated)	`POST /api/ussd/send`	Push a USSD message to a subscriber. Requires `ussd_gateway_enabled: true` (else `503`). See the USSD Gateway Guide.

CAP / CAMEL

Operation	Endpoint	Description
CAP InitialDP	`POST /api/cap/initialDP`	Trigger a CAMEL dialogue toward a gsmSCF. Returns an `otid` for follow-up ops.
CAP Connect	`POST /api/cap/connect`	Route the call to a destination (`destination_number` + `otid`).
CAP Continue	`POST /api/cap/continue`	Continue call processing (`otid`).
CAP Release Call	`POST /api/cap/releaseCall`	Release the call with a `cause` (`otid`).
CAP Apply Charging	`POST /api/cap/applyCharging`	Apply a charging window (`duration` + `otid`).

CAP follow-up operations carry the dialogue `otid` (hex, returned by `initialDP`) and respond with `{"status":"sent","otid":"<hex>"}`. See the CAMEL Gateway Guide.

Raw Injection (testing / diagnostics)

Operation	Endpoint	Description
Raw SCCP	`POST /api/raw/sccp`	Inject a hand-built SCCP payload (hex). Optional SCCP overrides (called/calling party, SSNs, OPC/DPC).
Raw M3UA	`POST /api/raw/m3ua`	Inject a hand-built M3UA payload (hex). Optional OPC/DPC/routing-context overrides.

Utility Endpoints

Endpoint	Method	Purpose
`/metrics`	GET	Prometheus metrics (text format).
`/swagger`	GET	Swagger UI.
`/swagger.json`	GET	OpenAPI specification.

Worked Examples

The following examples illustrate the request/response shape for representative operations. For the full field reference of every operation, see the MAP Client Guide.

SendRoutingInfo (SRI)

Retrieve routing information for establishing a call to a mobile subscriber.

Endpoint: `POST /api/sri`

Request Body:

```
{
  "msisdn": "1234567890",
  "gmsc": "5551234567"
}
```

Field	Type	Required	Description
`msisdn`	String	Yes	Called party MSISDN
`gmsc`	String	Yes	Gateway MSC Global Title

Response (200 OK):

```
{
  "result": {
    "imsi": "001001234567890",
    "msrn": "5551234999",
    "vlr_number": "5551234800"
  }
}
```

cURL Example:

```
curl -X POST http://localhost:8080/api/sri \
  -H "Content-Type: application/json" \
  -d '{"msisdn": "1234567890", "gmsc": "5551234567"}'
```

SendRoutingInfoForSM (SRI-for-SM)

Retrieve routing information for delivering an SMS to a mobile subscriber.

Endpoint: `POST /api/sri-for-sm`

Request Body:

```
{
  "msisdn": "1234567890",
  "service_center": "5551234567"
}
```

Field	Type	Required	Description
`msisdn`	String	Yes	Destination MSISDN
`service_center`	String	Yes	Service Center Global Title

cURL Example:

```
curl -X POST http://localhost:8080/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{"msisdn": "1234567890", "service_center": "5551234567"}'
```

UpdateLocation

Notify the HLR of a subscriber location change (VLR registration). In HLR mode this triggers the InsertSubscriberData (ISD) sequence.

Endpoint: `POST /api/updateLocation`

Request Body:

```
{
  "imsi": "001001234567890",
  "vlr": "5551234800"
}
```

Field	Type	Required	Description
`imsi`	String	Yes	Subscriber IMSI
`vlr`	String	Yes	VLR Global Title address

USSD Send (Network-Originated)

Push a USSD message to a subscriber. Requires `ussd_gateway_enabled: true`. For the full callback protocol and session lifecycle, see the USSD Gateway Guide.

Endpoint: `POST /api/ussd/send`

Request Body:

```
{
  "msisdn": "+254712345678",
  "text": "You have a pending bill. Reply 1 to pay.",
  "callback_url": "http://billing-app:9000/ussd"
}
```

Field	Type	Required	Description
`msisdn`	String	Yes	Destination subscriber MSISDN
`text`	String	Yes	USSD text to display (GSM 7-bit)
`callback_url`	String	Yes	URL to receive subscriber replies via HTTP POST

Response (200 OK):

```
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "status": "sent"
}
```

Error (503 — gateway disabled):

```
{
  "error": "USSD gateway not enabled"
}
```

Authentication

Current Status: The API does **not require authentication**.

Security Considerations:

- API is intended for internal/trusted network use
- Consider using firewall rules to restrict access
- For production deployments, consider implementing authentication middleware

---

Response Formats

All responses use **JSON** format.

Success Response

HTTP Status: 200 OK

Structure:

```
{
  "result": {
    // Operation-specific response data
  }
}
```

Error Response

HTTP Status:

- 400 Bad Request - Invalid or missing request fields
- 500 Internal Server Error - Operation failed (non-timeout error returned by the SS7 stack)
- 503 Service Unavailable - Feature not enabled (USSD gateway)
- 504 Gateway Timeout - Operation timed out (default 10 seconds)
- 404 Not Found - Invalid endpoint

Structure:

```
{
  "error": "timeout"
}
```

or

```
{
  "error": "invalid request"
}
```

Error Handling

Common Errors

Error	HTTP Code	Description	Solution
Invalid JSON	400	Request body is not valid JSON	Check JSON syntax
Missing/invalid fields	400	Required fields missing or wrong type (responses often include a `"required"` list)	Include all required parameters with correct types
Operation error	500	SS7 stack returned a non-timeout error	Inspect the `"error"` message; check SS7 event logs
Feature disabled	503	USSD gateway not enabled	Set `ussd_gateway_enabled: true`
Timeout	504	Operation exceeded its timeout (default 10s)	Check M3UA connectivity, HLR/VLR availability
Not Found	404	Invalid endpoint	Check endpoint URL

Timeout Behavior

Awaited operations use a **10-second timeout** by default:

1. Request sent to MapClient GenServer

2. Waits for response up to 10 seconds
3. If no response → returns 504 Gateway Timeout
4. If response received → returns 200 OK with result

Troubleshooting Timeouts:

- Check M3UA connection status (Web UI → M3UA page)
- Verify network element (HLR/VLR/MSC) is reachable
- Check routing configuration
- Review SS7 event logs for errors

Metrics (Prometheus)

The API exposes Prometheus metrics for monitoring.

Metrics Endpoint

URL: `http://[server-ip]:8080/metrics`

Format: Prometheus text format

Example Output:

```
# HELP map_requests_total Total MAP requests
# TYPE map_requests_total counter
map_requests_total{operation="sri"} 42
map_requests_total{operation="sri_for_sm"} 158
map_requests_total{operation="updateLocation"} 23

# HELP cap_requests_total Total CAP requests
# TYPE cap_requests_total counter
cap_requests_total{operation="initialDP"} 87
cap_requests_total{operation="requestReportBCSMEvent"} 91

# HELP map_request_duration_milliseconds Duration of MAP
request/responses in ms
# TYPE map_request_duration_milliseconds histogram
map_request_duration_milliseconds_bucket{operation="sri",le="10"}
5
map_request_duration_milliseconds_bucket{operation="sri",le="50"}
12
map_request_duration_milliseconds_bucket{operation="sri",le="100"}
35
...

# HELP map_pending_requests Number of pending MAP TID waiters
# TYPE map_pending_requests gauge
map_pending_requests 3

# HELP omniss7_license_status Current license status (1 = valid, 0
= invalid)
# TYPE omniss7_license_status gauge
omniss7_license_status 1
```

Available Metrics

Metric	Type	Labels	Descr
`map_requests_total`	Counter	`operation`	Total num MAP requ operation
`cap_requests_total`	Counter	`operation`	Total num CAP reque operation
`map_request_duration_milliseconds`	Histogram	`operation`	Request d millisecon
`map_pending_requests`	Gauge	-	Number o MAP trans
`ussd_requests_total`	Counter	`direction`	Total USSI requests (inbound/
`ussd_active_sessions`	Gauge	-	Number o USSD ses
`omniss7_license_status`	Gauge	-	Current lic status (1 = invalid)

Prometheus Configuration

Add to your `prometheus.yml`:

```yaml
scrape_configs:
  - job_name: 'omniss7'
    static_configs:
      - targets: ['server-ip:8080']
    metrics_path: '/metrics'
    scrape_interval: 15s
```

Example Requests

Python Example

```python
import requests
import json

# SRI-for-SM Request
url = "http://localhost:8080/api/sri-for-sm"
payload = {
    "msisdn": "1234567890",
    "service_center": "5551234567"
}

response = requests.post(url, json=payload, timeout=15)

if response.status_code == 200:
    result = response.json()
    print(f"Success: {result}")
elif response.status_code == 504:
    print("Timeout - no response from network")
else:
    print(f"Error: {response.status_code} - {response.text}")
```

JavaScript Example

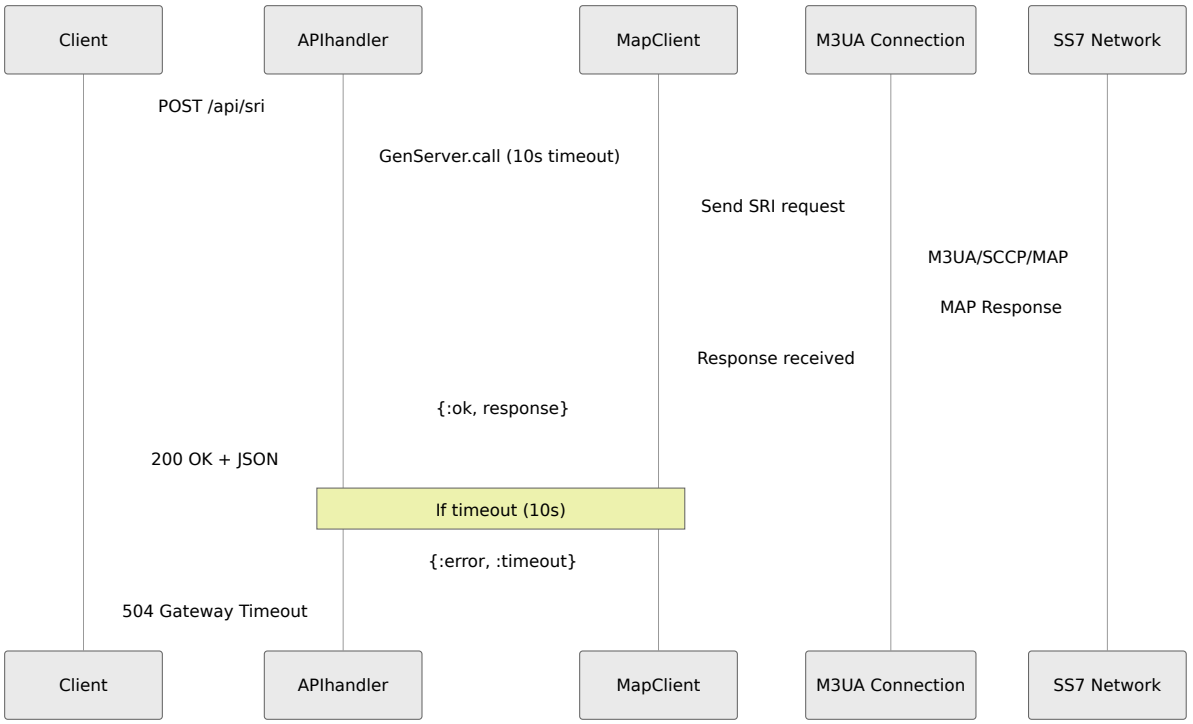
```
const axios = require('axios');

async function sendSRI() {
  try {
    const response = await
axios.post('http://localhost:8080/api/sri', {
      msisdn: '1234567890',
      gmsc: '5551234567'
    }, {
      timeout: 15000
    });

    console.log('Success:', response.data);
  } catch (error) {
    if (error.code === 'ECONNABORTED') {
      console.error('Timeout - no response from network');
    } else {
      console.error('Error:', error.response?.data ||
error.message);
    }
  }
}

sendSRI();
```

Bash/cURL Example

```
#!/bin/bash

# UpdateLocation Request
response=$(curl -s -w "\n%{http_code}" -X POST
http://localhost:8080/api/updateLocation \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "vlr": "5551234800"
  }')

http_code=$(echo "$response" | tail -n 1)
body=$(echo "$response" | sed '$d')

if [ "$http_code" -eq 200 ]; then
  echo "Success: $body"
elif [ "$http_code" -eq 504 ]; then
  echo "Timeout - no response from network"
else
  echo "Error $http_code: $body"
fi
```

Flow Diagrams

API Request Flow



Summary

The OmniSS7 REST API provides:

- **47 operation endpoints** across Mobility, SMS, Authentication, Subscriber Data, Supplementary Services, Location Services, GPRS, Equipment, USSD, CAP/CAMEL, and Raw injection.
- **Swagger UI** at `/swagger` for interactive documentation and testing.
- **Prometheus metrics** at `/metrics` for monitoring and observability.
- **10-second default timeout** for awaited operations (`504` on timeout).
- A `Plug.Cowboy` **HTTP server on port 8080**, toggled by `start_http_server` — separate from the control panel (:8087) and `:api_ex` REST controllers (:8445).

For per-operation request/response schemas and MAP opcodes, see the MAP Client Guide.

For Web UI access, see the Web UI Guide.

For configuration details, see the Configuration Reference.

Technical Reference (Appendix)

← Back to Main Documentation

Technical reference for SS7 protocols and OmniSS7 implementation.

SS7 Protocol Stack

MAP Operation Codes

Opcodes below are the MAP `local` operation codes per 3GPP TS 29.002. This table covers the operations OmniSS7 generates and/or handles; for the full set of client-triggerable operations and their request parameters, see the **MAP Client Guide**.

Operation	Opcode	Purpose
updateLocation	2	Register subscriber location (CS)
cancelLocation	3	Deregister subscriber from VLR/SGSN
provideRoamingNumber	4	Request MSRN
noteSubscriberDataModified	5	Notify gsmSCF of subscriber data change
insertSubscriberData	7	Push subscriber profile to VLR/SGSN
deleteSubscriberData	8	Withdraw subscriber data from VLR/SGSN
registerSS	10	Register a supplementary service (e.g. CFU)
eraseSS	11	Erase a supplementary service
activateSS	12	Activate a supplementary service
deactivateSS	13	Deactivate a supplementary service
interrogateSS	14	Interrogate supplementary service status
registerPassword	17	Register an SS password

Operation	Opcode	Purpose
getPassword	18	Request an SS password from the subscriber
sendRoutingInfo	22	Query call routing (SRI)
updateGprsLocation	23	Register packet-switched (SGSN) location
sendRoutingInfoForGprs	24	Query GPRS routing toward the GGSN
reset	37	HLR reset toward VLR/SGSN
forwardCheckSS-Indication	38	Indicate SS data should be re-checked
checkIMEI	43	Query EIR for equipment status
mt-forwardSM	44	Deliver SMS to subscriber (MT)
sendRoutingInfoForSM	45	Query SMS routing (SRI-for-SM)
mo-forwardSM	46	Forward SMS from subscriber (MO)
reportSM-DeliveryStatus	47	Report SMS delivery outcome to HLR
sendIdentification	55	Retrieve IMSI/auth from previous VLR

Operation	Opcode	Purpose
sendAuthenticationInfo	56	Request authentication vectors
restoreData	57	Restore subscriber data after VLR failure
sendIMSI	58	Resolve MSISDN to IMSI at the HLR
processUnstructuredSS-Request	59	Mobile-originated USSD request (inbound)
unstructuredSS-Request	60	Network-originated USSD request (outbound)
unstructuredSS-Notify	61	Network-originated USSD notification
anyTimeSubscriptionInterrogation	62	gsmSCF query of subscription data
alertServiceCentre	64	Alert SMSc that a subscriber is reachable
anyTimeModification	65	gsmSCF modification of subscriber data
readyForSM	66	Notify HLR subscriber is ready for SMS
purgeMS	67	Purge subscriber record from VLR/SGSN
provideSubscriberInfo	70	Request subscriber state/location

Operation	Opcode	Purpose
anyTimeInterrogation	71	gsmSCF query of subscriber state/location
provideSubscriberLocation	83	Request location estimate from serving node (LCS)
sendRoutingInfoForLCS	85	GMLC query for the subscriber's serving node (LCS)
subscriberLocationReport	86	Serving node reports a location to the GMLC (LCS)
noteMM-Event	89	Report a mobility-management event to gsmSCF

TCAP Message Types

- **BEGIN** - Start transaction
- **CONTINUE** - Mid-transaction
- **END** - Final response
- **ABORT** - Cancel transaction

SCCP Addressing

Global Title Formats

- **E.164** - International phone number (e.g., 447712345678)
- **E.212** - IMSI format (e.g., 234509876543210)

- **E.214** - Point code format

Subsystem Numbers (SSN)

- **SSN 6**: HLR
- **SSN 7**: VLR
- **SSN 8**: MSC/SMSC
- **SSN 9**: GMLC
- **SSN 10**: SGSN

SMS TPDU

Message Types

- **SMS-DELIVER** (MT) - Network to mobile
- **SMS-SUBMIT** (MO) - Mobile to network
- **SMS-STATUS-REPORT** - Delivery status
- **SMS-COMMAND** - Remote command

Character Encodings

- **GSM7** - 7-bit GSM alphabet (160 chars per SMS)
- **UCS2** - 16-bit Unicode (70 chars per SMS)
- **8-bit** - Binary data (140 bytes per SMS)

M3UA States

- **DOWN** - No SCTP connection
- **CONNECTING** - SCTP connecting
- **ASPUP_SENT** - Waiting for ASPUP ACK
- **INACTIVE** - ASP up but not active

- **ASPAC_SENT** - Waiting for ASPAC ACK
- **ACTIVE** - Ready for traffic

Common SS7 Point Codes

Point codes are typically 14-bit (ITU) or 24-bit (ANSI) values.

Example Format (ITU):

- Network: 3 bits
- Cluster: 8 bits
- Member: 3 bits

SCCP Error Codes

- **0** - No translation for address
- **1** - No translation for specific address
- **2** - Subsystem congestion
- **3** - Subsystem failure
- **4** - Unequipped user
- **5** - MTP failure
- **6** - Network congestion
- **7** - Unqualified
- **8** - Error in message transport

MAP Error Codes

Code	Error	Description
1	unknownSubscriber	Subscriber not in HLR
27	absentSubscriber	Subscriber not reachable
34	systemFailure	Network failure
35	dataMissing	Required data not available
36	unexpectedDataValue	Invalid parameter value

Related Documentation

- ← Back to Main Documentation
- STP Guide
- MAP Client Guide
- SMS Center Guide
- HLR Guide
- Common Features

CAMEL Gateway Configuration Guide

Overview

The **CAMEL Gateway (CAMELGW)** mode turns OmniSS7 into an Intelligent Network (IN) platform that provides real-time call control and charging using the CAMEL Application Part (CAP) protocol. OmniSS7 acts as the **gsmSCF** (Service Control Function), receiving triggers from the **gsmSSF** in the MSC/VLR/GMSC and driving charging decisions through CGrateS.

What is CAMEL?

CAMEL (Customized Applications for Mobile network Enhanced Logic) is a set of standards that work on either a GSM core network or a UMTS network. It lets operators provide services that require real-time control of calls, such as:

- **Prepaid calling** - Real-time balance checking and charging
- **Premium rate services** - Special billing for value-added services
- **Call routing control** - Dynamic destination routing based on time/location
- **Virtual private networks** - Corporate numbering plans
- **Call screening** - Allow/block calls based on criteria

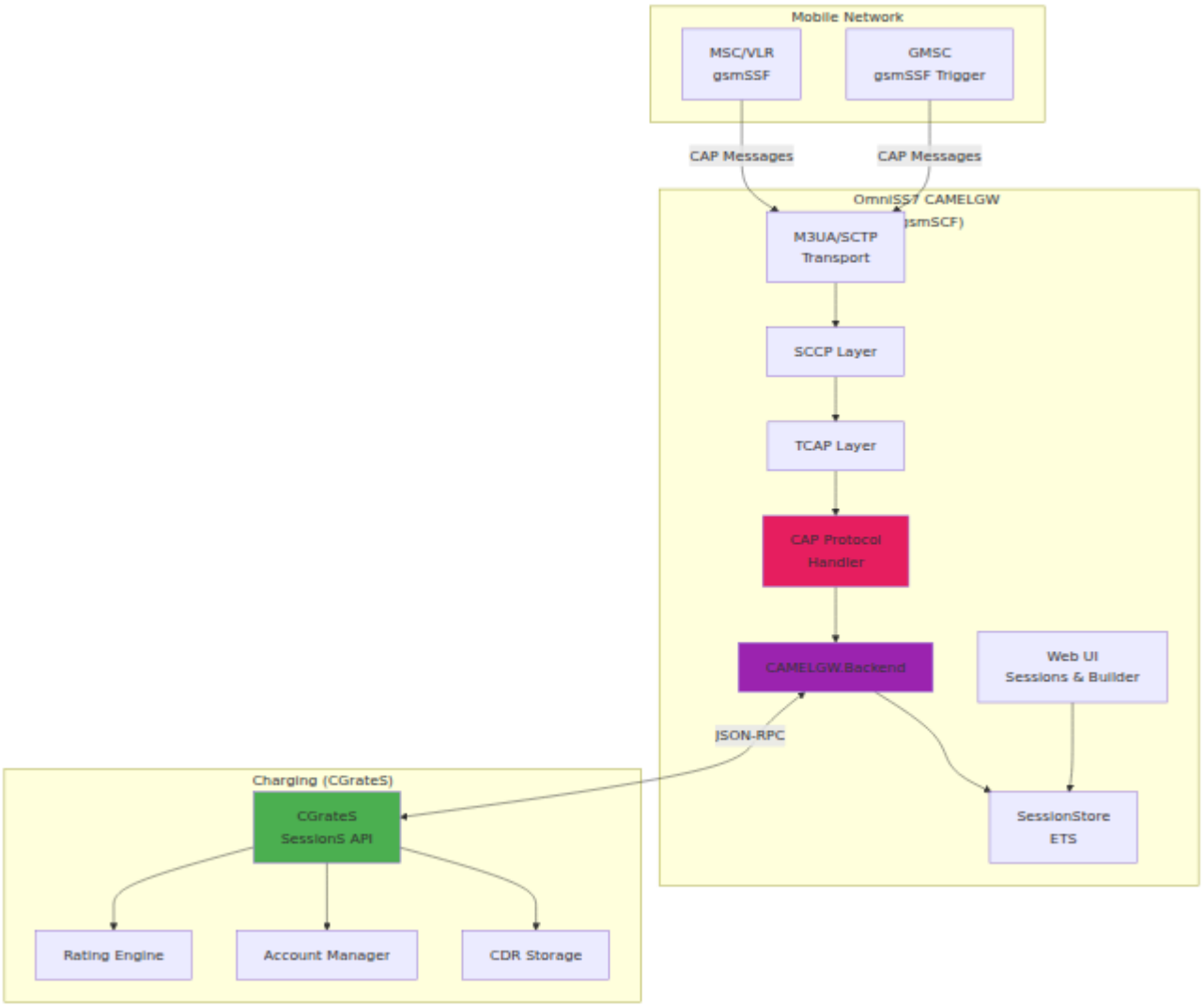
CAP Protocol Versions

OmniSS7 CAMELGW supports multiple CAP versions. The version controls the application-context OID carried in the TCAP dialogue:

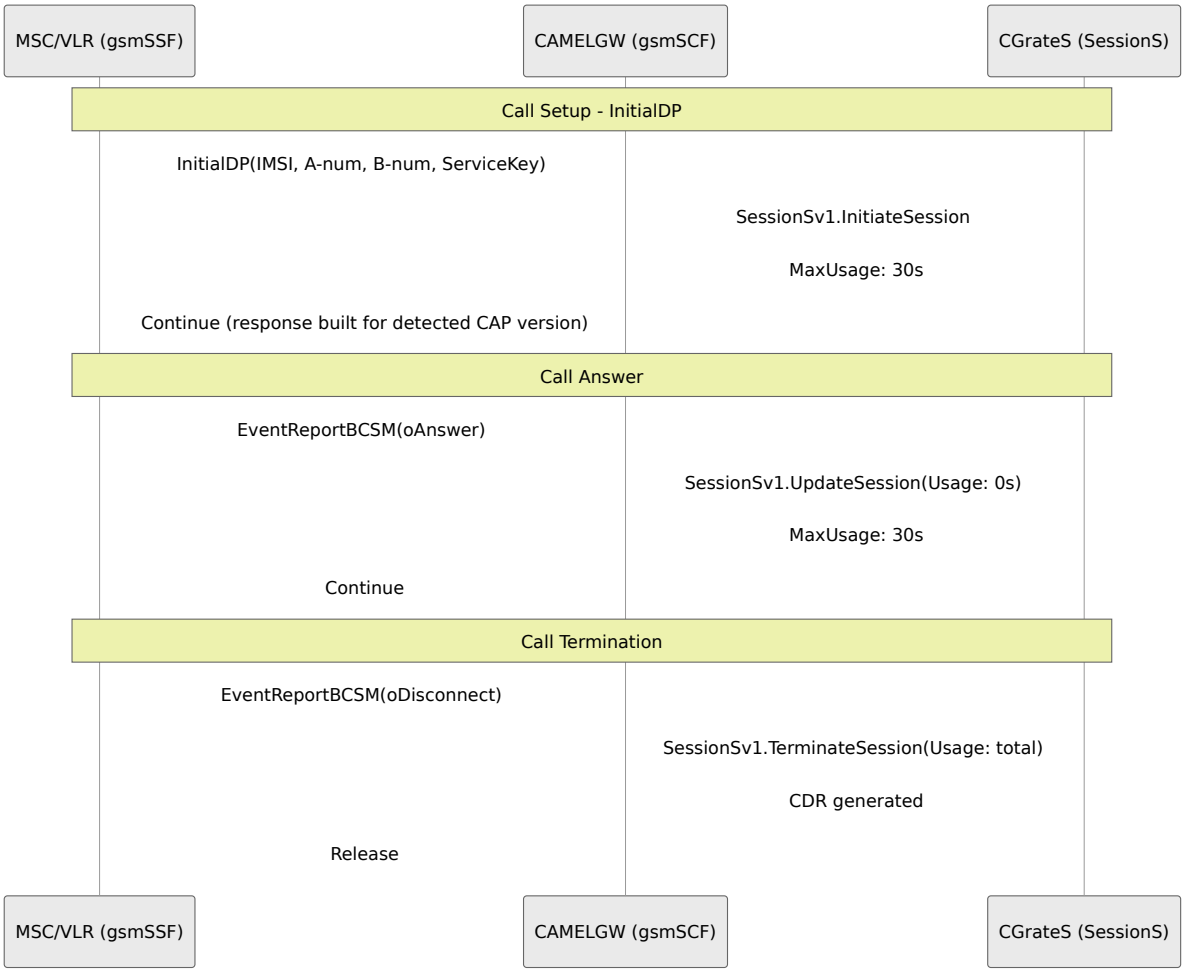
Version	Phase	Application-Context OID
CAP v1	CAMEL Phase 1	`0.4.0.0.1.0.50.0`
CAP v2	CAMEL Phase 2	`0.4.0.0.1.0.50.1`
CAP v3	CAMEL Phase 3	`0.4.0.0.1.21.3.4`
CAP v4	CAMEL Phase 4	`0.4.0.0.1.23.3.4`

Default: CAP v2 (most widely deployed). The incoming version is auto-detected from the InitialDP dialogue portion and reused when generating the response, so responses match each MSC's CAP version.

Architecture



Call Flow Example



Configuration

Prerequisites

- OmniSS7 installed and running
- M3UA connectivity to MSC/GMSC (gsmSSF)
- A CGrateS instance reachable over JSON-RPC (optional, for real-time charging)

Enable CAMEL Gateway Mode

CAMEL processing is controlled by two independent flags. `cap_client_enabled` brings up the CAP/M3UA client and CAMEL session components; `cgrates_enabled` turns on the CGrateS charging integration behind the backend. Configure these in `config/runtime.exs`:

```
config :omniss7,
  # Bring up the CAP/CAMEL client (M3UA ASP, SessionStore, request
handling)
  cap_client_enabled: true,

  # Other independent modes (set as needed for your deployment)
  map_client_enabled: false,
  smsc_mode_enabled: false,

  # CAP version used for outgoing requests / dialogue (:v1 | :v2 |
:v3 | :v4)
  cap_version: :v2,

  # CGrateS charging integration
  cgrates_enabled: true,

  # M3UA connection for CAP (connect as ASP toward the MSC/GMSC
gsmSSF)
  cap_client_m3ua: %{
    routing_context: 1,
    opc: 5013,
    dpc: 5011
  }
```

> **Note on `cap_client_m3ua`:** the CAMEL client reads only `:routing_context`, `:opc`, and `:dpc` from this map. Any additional keys are ignored by the CAP client. The M3UA ASP process name is fixed internally as `:camelgw_client_asp`.

> There is no `camelgw_mode_enabled`, `hlr_mode_enabled`, or `smsc_mode_enabled`-for-CAMEL flag. The only flags that gate CAMEL behaviour are `cap_client_enabled` and `cgrates_enabled`.

Mode Flags

Key	Default	Purpose
`cap_client_enabled`	`false`	Starts the CAP/CAMEL client (M3UA ASP, SessionStore, CAP request handling)
`cgrates_enabled`	`false`	Enables CGrateS charging integration in `CAMELGW.Backend`
`cap_version`	`:v2`	Default CAP version / dialogue OID for outgoing requests
`map_client_enabled`	`false`	Independent MAP client mode
`smsc_mode_enabled`	`false`	Independent SMSc mode (requires MAP client)

CGrateS Charging Parameters

Key	Default	Purpose
`cgrates_enabled`	`false`	Master switch for CGrateS integration
`cgrates_url`	`http://localhost:2080/jsonrpc`	CGrateS JSON-RPC endpoint
`cgrates_tenant`	`cgrates.org`	Tenant used on every SessionS event
`cgrates_request_type`	`*prepaid`	CGrateS request type (e.g. `*prepaid`, `*postpaid`)
`cgrates_timeout`	`5000`	HTTP request timeout in milliseconds

Configure Web UI Pages

The Web UI exposes the CAMEL pages through the control panel. These are configured in `config/config.exs`:

```
config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 Events"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "Peers"},
    {SS7.Web.CAMELSessionsLive, "/camel_sessions", "CAMEL
Sessions"},
    {SS7.Web.CAMELRequestLive, "/camel_request", "CAMEL Request
Builder"}
  ],
  page_order: [
    "/events", "/m3ua", "/camel_sessions", "/camel_request",
    "/application", "/configuration"
  ]
```

Page label	Route	Module
Peers	`/m3ua`	`SS7.Web.M3UAStatusLive`
CAMEL Sessions	`/camel_sessions`	`SS7.Web.CAMELSessionsLive`
CAMEL Request Builder	`/camel_request`	`SS7.Web.CAMELRequestLive`

CAP Operations Supported

CAP operations are identified by name and direction. Note that the codebase is internally inconsistent on some CAP opcode numbers (the event-log opcode map and the request generator disagree on a few operations), so operations below are described by **name and endpoint**. Where an opcode is cited, it is the one `cap_request_generator.ex` actually emits.

Incoming Operations (gsmSSF → gsmSCF)

Handled in `cap_request.ex` (`handle_request/1` dispatch on the decoded TCAP component opcode):

Operation	Handler	
InitialDP	`handle_initial_dp/1`	C n C 1 A 0 6 I
AssistRequestInstructions	`handle_assist_request_instructions/1`	A d 1
EventReportBCSM	`handle_event_report_bcsm/1`	E (o d u
Continue	`handle_continue/1`	A
ConnectToResource	`handle_connect_to_resource/1`	A
RequestReportBCSMEvent	`handle_request_report_bcsm_event/1`	A
ApplyCharging	`handle_apply_charging/1`	P d a
Connect	`handle_connect/1`	P d a
ReleaseCall	`handle_release_call/1`	A

Unhandled opcodes are answered with a TCAP error (Facility Not Supported / Unexpected Data Value).

Outgoing Operations (gsmSCF → gsmSSF)

Generated by `cap_request_generator.ex` and sent via `CapClient`. The opcodes below are the values the generator emits:

Operation	Opcode	Genera
InitialDP	0	`CapRequestGenerator.initial_dp`
Connect	20	`CapRequestGenerator.connect_re`
ReleaseCall	22	`CapRequestGenerator.release_ca`
RequestReportBCSMEvent	23	`CapRequestGenerator.request_re`
Continue	31	`CapRequestGenerator.continue_r`
ApplyCharging	35	`CapRequestGenerator.apply_char`
InitialDPSMS	60	`CapRequestGenerator.initial_dp`
NoteMM-Event	89 (MAP)	`CapRequestGenerator.note_mm_ev`

> **InitialDPSMS** (opcode 60) is for MO-SMS CAMEL control: sent by the smsSSF to the gsmSCF when an SMS-CSI / mo-sms-CSI is armed, before the short message is relayed to the SMSC. The arg is encoded with the generated CAP ASN.1 module and the dialogue carries the `id-ac-cap3-sms-AC` application context.
>
> **NoteMM-Event** is a MAP operation (opcode 89, not CAP) used for CAMEL M-CSI mobility notifications from the VLR to the gsmSCF. It is encoded with the MAP ASN.1 module (`:TCAPMessages.encode(:MapSpecificPDUs, …)`) and carries the `mm-EventReporting-v3` application context.

All CAP TCAP messages are encoded with `:GenericTCAP.encode(:GenericPDUs, …)`. Only the MAP-based NoteMM-Event uses `:TCAPMessages`.

Web UI Features

CAMEL Sessions Page

Title: CAMEL Sessions · **Route:** `/camel_sessions` (HTTPS, port 8087)

Real-time monitoring of active CAMEL call sessions:

- **Live session list** with auto-refresh
- **Session details** - OTID, Call ID, State, Duration
- **CAP Version** detected from InitialDP and stored per session
- **Call information** - IMSI, A-number, B-number, Service Key
- **State tracking** - `initiated`, `answered`, `terminated`

Table Columns: Call ID, State, Version, IMSI, Calling Number, Called Number, Service Key, Duration, Start Time, OTID

The same session data is available over the REST API at `/camel-sessions` (see Monitoring & Operations).

CAMEL Request Builder

Title: CAMEL Request Builder · **Route:** `/camel_request` (HTTPS, port 8087)

Interactive tool for building and sending CAP requests for testing. See the dedicated CAMEL Request Builder Guide for full details.

Request types offered by the form: InitialDP, Connect, ReleaseCall, RequestReportBCSMEvent, ApplyCharging, Continue.

CAP version selector: the form has a per-request CAP version dropdown (v1/v2/v3/v4 with their OIDs). The selection temporarily overrides the

configured default for that request and persists across form changes; the request history shows the version used in a **CAP Ver** column.

Advanced SCCP/M3UA options:

Field	Default	Notes
Called Party GT	`68988415011`	Destination Global Title
Calling Party GT	(configurable)	Originating Global Title
Called SSN	`146`	gsmSSF
Calling SSN	`146`	gsmSSF
OPC	`5013`	Originating Point Code
DPC	`5011`	Destination Point Code

Integration with CGrateS

OmniSS7 charges calls through the CGrateS **SessionS** JSON-RPC API. `CGrateS.SessionsClient` posts to the configured `cgrates_url` and maps CAP operations to SessionS methods:

CAP trigger	SessionS method
InitialDP	`SessionSv1.InitiateSession`
EventReportBCSM (oAnswer)	`SessionSv1.UpdateSession`
EventReportBCSM (oDisconnect)	`SessionSv1.TerminateSession`

Call Lifecycle with Charging

1. Call Initiation (InitialDP)

When the MSC sends InitialDP, CAMELGW:

1. **Detects the CAP version** from the dialogue portion (falls back to the configured default)
2. **Decodes the InitialDP** parameters (IMSI, calling/called numbers, service key)
3. **Calls CGrateS** `SessionSv1.InitiateSession` via `CAMELGW.Backend.handle_initial_dp/1`
4. **Reads** `MaxUsage` from the response (allowed call duration in seconds)
5. **Stores the session** in `CAMELGW.SessionStore` (ETS) keyed by OTID, including the CAP version and a call ID of the form `CAMEL-<hex OTID>`
6. **Responds to the MSC** using the detected CAP version

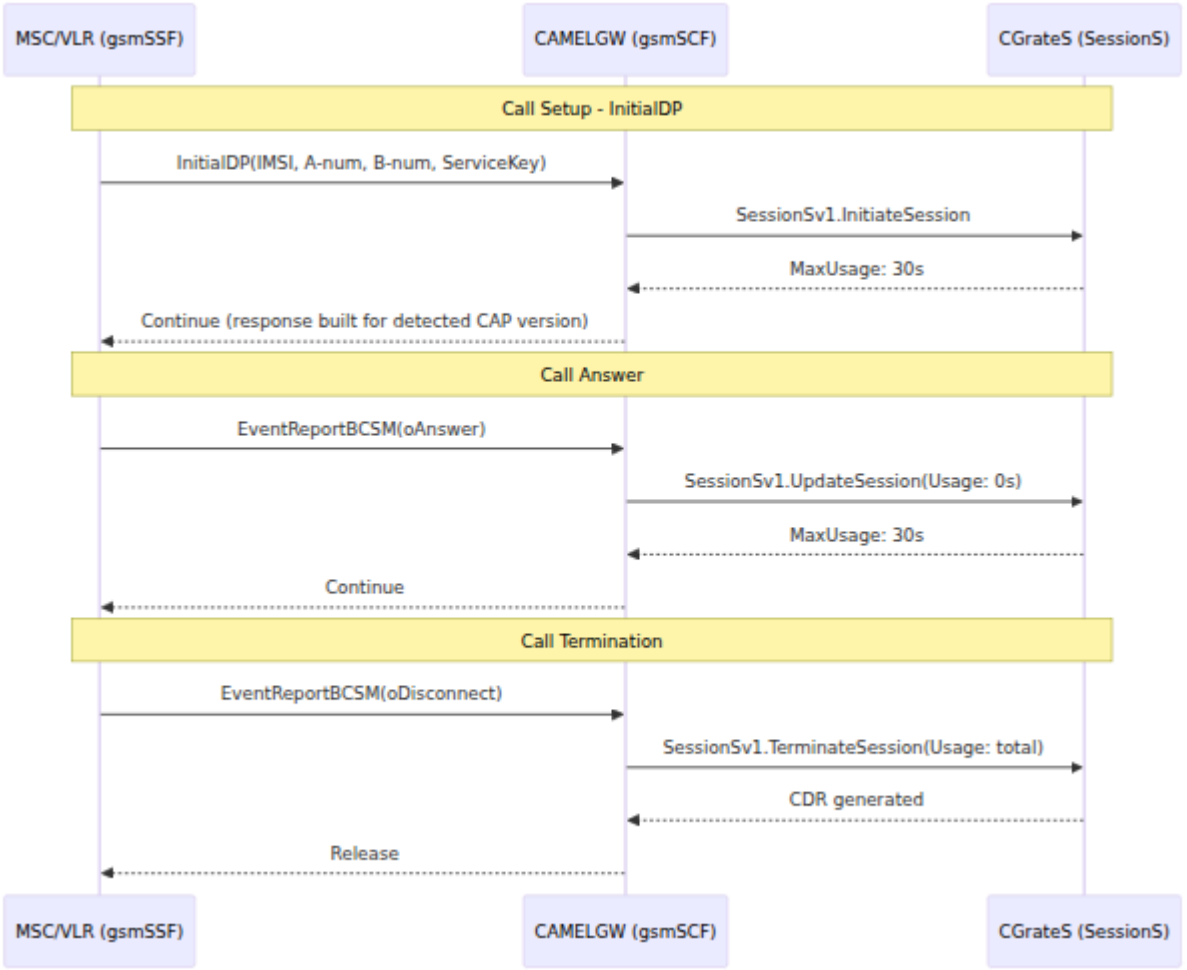
The account on the SessionS event is the subscriber **IMSI** when present, otherwise the **calling number**. The event also carries `Destination`, `ServiceKey`, `MSCAddress`, `CallingPartyNumber`, `CalledPartyNumber`, and `OriginID` (the call ID).

2. Call Answer (EventReportBCSM - oAnswer)

On `oAnswer`, the backend looks up the session, sets its state to `answered`, and calls `SessionSv1.UpdateSession` with usage 0 to start the debit loop. If CGrateS fails, the call is **not** dropped — the backend logs the error and continues.

3. Call Termination (EventReportBCSM - oDisconnect)

On `oDisconnect`, the backend computes the total duration from the session start time, sets state to `terminated`, calls `SessionSv1.TerminateSession` with the final usage, and deletes the session from the store. CDRs are generated by CGrateS. The session is cleaned up even if the terminate call errors.



CDR Fields from CAMEL

CDRs are produced by CGrateS and typically include:

- `Account` - IMSI (or calling number fallback)
- `Destination` - Called party number
- `OriginID` - Call identifier (`CAMEL-<hex OTID>`)
- `Usage` - Total call duration (seconds)
- `IMSI`, `CallingPartyNumber`, `CalledPartyNumber`
- `MSCAddress` - Serving MSC
- `ServiceKey` - CAMEL service key

Testing

Manual Testing with the Request Builder

1. Open `https://your-server:8087/camel_request`
2. Select **InitialDP**, set Service Key / Calling / Called numbers, optionally pick a CAP version, and send. Note the OTID generated.
3. Open `https://your-server:8087/camel_sessions` to see the session in state `initiated`.
4. Send **EventReportBCSM** with `oAnswer` to move the session to `answered`.
5. Send **ReleaseCall** with cause `16` (Normal) to terminate.

Testing with a Real MSC

Configure CAMEL service on your MSC/VLR (Service Key, gsmSCF Global Title) so it triggers InitialDP toward OmniSS7. Example for a Huawei MSC:

```
ADD CAMELSERVICE:
  SERVICEID=1,
  SERVICEKEY=100,
  GSMSCFADDR="55512341234",   # CAMELGW Global Title
  DEFAULTCALLHANDLING=CONTINUE;

ADD CAMELSUBSCRIBER:
  IMSI="310150123456789",
  SERVICEID=1,
  TRIGGERTYPE=TERMCALL;
```

Monitor Logs

OmniSS7 emits structured JSON logs (one object per line via `OmniLogger.JsonFormatter`), so you can filter on metadata fields:

```
# Tail and filter CAP events by the map_event field
tail -f /var/log/omniss7/omniss7.log | jq 'select(.map_event |
tostring | startswith("CAP:"))'
```

Monitoring & Operations

Prometheus Metrics

CAMELGW exposes metrics at `http://localhost:8080/metrics`.

CAP metrics:

- `cap_requests_total{operation}` - Total CAP requests by operation. Label values are snake_case, e.g. `initial_dp`, `connect`, `release_call`, `continue`, `apply_charging`.
- `cap_request_errors_total{operation}` - CAP request generation/send errors by operation.

MAP / API metrics:

- `map_requests_total{operation}` - Total MAP requests by operation
- `map_pending_requests` - Number of pending MAP TID waiters

M3UA STP metrics (if STP mode enabled):

- `m3ua_stp_routing_failures_total{reason}` - Routing failures by reason

```
# All CAP request counters
curl http://localhost:8080/metrics | grep cap_requests_total

# InitialDP only (note snake_case label)
curl http://localhost:8080/metrics | grep
'cap_requests_total{operation="initial_dp"}'
```

Health Checks

```
# M3UA peer status (TLS API on port 8445)
curl -k https://localhost:8445/m3ua-status

# Active CAMEL sessions (returns session_count + per-session
detail)
curl -k https://localhost:8445/camel-sessions
```

There is no `/api/ocs-status`, `/api/events`, or `/api/camel/sessions/count` endpoint. Session counts come from `/camel-sessions`.

Logging Configuration

Adjust the log level in `config/runtime.exs`:

```
config :logger, level: :info   # :debug | :info | :warning |
:error
```

The formatter is set to structured JSON in `config/config.exs`:

```
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all,
  truncate: :infinity
```

Troubleshooting

Issue: No CAP messages received

Symptoms: Request Builder works, but the MSC never sends InitialDP.

Check:

1. M3UA peer status: `curl -k https://localhost:8445/m3ua-status`
2. MSC CAMEL service configuration (Service Key, gsmSCF address)
3. SCCP routing (the gsmSCF Global Title must route to CAMELGW)
4. Firewall rules (allow SCTP port 2905)

Issue: CGrateS errors

Symptoms: `not_configured`, `http_error`, or `cgrates_error` in logs.

Check:

1. `cgrates_enabled: true` and `cgrates_url` point at a reachable JSON-RPC endpoint
2. The account (IMSI or calling number) exists and has balance in CGrateS
3. `cgrates_tenant` and `cgrates_request_type` match your CGrateS rating setup
4. `cgrates_timeout` is large enough for your CGrateS latency

On a charging failure during a call, CAMELGW continues the call rather than dropping it; the error is logged.

Issue: Session not found

Symptoms: EventReportBCSM logs "Session not found".

Cause: OTID mismatch or the session was already terminated.

Check: Verify the OTID in logs and ensure the DTID/OTID correlate across the dialogue.

Issue: Decode errors

Symptoms: "Failed to decode InitialDP" in logs.

Cause: CAP version mismatch or malformed message.

Check: Confirm the CAP version configuration matches the MSC, and capture a PCAP for analysis:

```
tcpdump -i eth0 -w cap_trace.pcap sctp port 2905
```

Summary

The CAMEL Gateway mode enables OmniSS7 to act as a gsmSCF with:

- **CAP protocol support** (v1/v2/v3/v4) with per-request version override
- **Real-time charging** via the CGrateS SessionS JSON-RPC API
- **Call control operations** (Connect, Release, Continue, ApplyCharging)
- **Session management** in ETS, keyed by OTID
- **Interactive testing** via the Web UI Request Builder
- **Live monitoring** of active sessions (Web UI and `/camel-sessions` API)
- **CDR generation** by CGrateS for billing and analytics

For more detail:

- CAMEL Request Builder Guide
- Technical Reference - CAP Operations

Product: OmniSS7 CAMEL Gateway **Documentation Version:** 1.1

CAMEL Request Builder

Overview

The **CAMEL Request Builder** is a LiveView page that builds and sends CAMEL/CAP requests for testing. It provides an interactive UI for creating InitialDP and follow-up CAMEL operations against a connected gsmSSF, and it logs every message to the event log in JSON.

Page title: CAMEL Request Builder **Route:** `/camel_request` **Access URL:** `https://your-server:8087/camel_request` (HTTPS)

Components

CAMEL Request Builder LiveView

`SS7.Web.CAMELRequestLive` renders the builder. Capabilities:

- Request type dropdown
- Dynamic form fields per request type
- **CAP version selector** (v1/v2/v3/v4) that overrides the configured default per request
- Collapsible Advanced SCCP/M3UA options (Global Titles, SSNs, OPC/DPC)
- Request history (last 20 requests) including a **CAP Ver** column
- Session tracking via OTID
- Success/error feedback and request size

Event Log with CAMEL Support

`lib/helpers/eventlog.ex` provides CAMEL/CAP logging (`paklog_camel/2` and helpers that look up CAP opcode names, extract OTID/DTID, and convert CAP PDUs to JSON). CAP operation names come from the module's internal opcode map, which currently contains **34 entries**.

A sample of that map:

```
0  => "initialDP"
1  => "assistRequestInstructions"
5  => "connect"
6  => "releaseCall"
7  => "requestReportBCSMEvent"
8  => "eventReportBCSM"
10 => "continue"
14 => "applyCharging"
15 => "applyChargingReport"
# ... 34 entries total
```

> **Opcode caveat:** the event-log opcode map and the outbound request generator (`cap_request_generator.ex`) do **not** agree on every opcode (for

> example, the event-log map labels opcode 14 as `applyCharging`, while the generator emits ApplyCharging as opcode 35). Treat the event-log map as a display/labelling aid for received messages, and treat `cap_request_generator.ex` as authoritative for what OmniSS7 actually sends. Operations are best identified by **name + endpoint**, not by a single opcode.

CapClient Logging

`CapClient` logs both incoming and outgoing CAP messages. Outgoing messages are logged from the SCCP/M3UA maker (`paklog_camel(&1, "outgoing")`); incoming messages are logged in the payload handler. This shares the event log with MAP events.

Configuration

The Request Builder page is registered in `config/config.exs` (not `runtime.exs`):

```
config :control_panel,
  use_additional_pages: [
    {SS7.Web.CAMELSessionsLive, "/camel_sessions", "CAMEL
Sessions"},
    {SS7.Web.CAMELRequestLive, "/camel_request", "CAMEL Request
Builder"}
  ],
  page_order: [
    # ...
    "/camel_sessions",
    "/camel_request"
  ]
```

Usage

Accessing the Request Builder

1. Navigate to `https://your-server:8087/camel_request`
2. Select the request type from the dropdown
3. Optionally choose a CAP version (defaults to the configured `cap_version`)
4. Fill in the required parameters
5. Optionally expand **Advanced SCCP/M3UA Options**
6. Click **Send [RequestType] Request**

Request Flow

InitialDP (new call)

1. Set Service Key (e.g. `100`)
2. Set Calling Number (A-party) and Called Number (B-party)
3. Send the request — a new OTID is generated and stored in the session for follow-up requests

Follow-up requests (Connect, ReleaseCall, etc.)

These reuse the stored OTID from InitialDP. If no OTID is active, the UI shows a warning and the request is not sent.

Request Types

The builder form offers six request types:

Request type	Parameters
InitialDP	Service Key (integer), Calling Number, Called Number
Connect	Destination Number
ReleaseCall	Cause Code (16 = Normal, 17 = Busy, 31 = Unspecified)
RequestReportBCSMEvent	BCSM events (e.g. oAnswer, oDisconnect)
ApplyCharging	Duration (seconds, 1–864000), Release on Timeout (boolean)
Continue	None (uses the active OTID)

> **Beyond the form:** two further operations exist in the request generator but are **not** exposed by the builder form — **InitialDPSMS** (opcode 60, `CapRequestGenerator.initial_dp_sms_request/5`) for MO-SMS CAMEL control, and **NoteMM-Event** (MAP opcode 89, `CapRequestGenerator.note_mm_event_request/4`) for M-CSI mobility notifications. See Additional Operations.

CAP Version Selection

The form includes a CAP version dropdown:

Option	Application-Context OID
CAP v1	`0.4.0.0.1.0.50.0`
CAP v2 (default)	`0.4.0.0.1.0.50.1`
CAP v3	`0.4.0.0.1.21.3.4`
CAP v4	`0.4.0.0.1.23.3.4`

Selecting a version temporarily overrides the configured default (`CAPVersion.set_default_version/1`) for the duration of the request, then restores the original. This controls the application-context OID placed in the TCAP dialogue of the outgoing message. The selection persists across form field changes, and the version used is recorded in the request history's **CAP Ver** column.

Advanced Options

Field	Default	Notes
Called Party GT	`68988415011`	Destination Global Title
Calling Party GT	(configurable)	Originating Global Title
Called SSN	`146`	gsmSSF
Calling SSN	`146`	gsmSSF
OPC	`5013`	Originating Point Code
DPC	`5011`	Destination Point Code

JSON Logging

Every CAMEL message is logged to the event log in JSON with direction, TCAP action, CAP operation name, SCCP addressing, OTID/DTID, and the full CAP PDU.

Example Log Entry

```
{
  "map_event": "CAP:initialDP",
  "direction": "outgoing",
  "tcap_action": "Begin",
  "otid": "A1B2C3D4",
  "sccp_called": {
    "SSN": 146,
    "GlobalTitle": {
      "Digits": "55512341234",
      "NumberingPlan": "isdn_tele",
      "NatureOfAddress_Indicator": "international"
    }
  },
  "event_message": "{ ... full CAP PDU ... }"
}
```

Because logs are emitted as JSON via `OmniLogger.JsonFormatter`, you can filter CAMEL traffic on the `map_event` field (prefix `CAP:`).

Request History & Session Tracking

The UI displays the last 20 requests with: timestamp, request type (color-coded badge), OTID (first 8 hex chars), status (sent/error), message size, and the **CAP Ver** used.

The current-session panel shows the active OTID and the last request size while a session is active.

Testing Workflow

1. **Start a call** — send InitialDP, get an OTID, the system creates a session.
2. **Control the call** — send RequestReportBCSMEvent, ApplyCharging, Connect, or ReleaseCall against the active OTID.
3. **View results** — check the request history, the **CAMEL Sessions** page, and the event log (filter on `CAP:`).

ApplyCharging — Call Duration Control

Overview

ApplyCharging sets a maximum call duration and optionally releases the call when that duration expires. It is typically used for prepaid charging or enforcing time limits.

Parameters

Duration (`maxCallPeriodDuration`)

- Integer, 1–864000 seconds (e.g. `60` = 1 minute, `300` = 5 minutes, `3600` = 1 hour)

Release on Timeout (`releaseIfDurationExceeded`)

- Boolean, default `true`
- `true`: release/disconnect the call when the duration expires
- `false`: notify the gsmSCF but keep the call active (allows the gsmSCF to act)

Message Structure

ApplyCharging is sent as a TCAP **Continue** carrying opcode **35** (the value emitted by `cap_request_generator.ex`). The argument (`ApplyChargingArg`) contains:

- `aChBillingChargingCharacteristics` → `timeDurationCharging` with `maxCallPeriodDuration` and the release flag
- `partyToCharge` (default: sendingSideID / leg1)

Example Usage

1. Send **InitialDP** (Service Key 100, calling/called numbers) → OTID `A1B2C3D4`
2. Send **ApplyCharging** (Duration 300, Release on Timeout true) → uses OTID `A1B2C3D4`
3. Send **Connect** (Destination) → uses OTID `A1B2C3D4`
4. After 300 seconds the network releases the call and the gsmSCF receives a disconnect notification.

Best Practices

1. **Send ApplyCharging before Connect** so charging is active when the call connects.
2. **Pair with RequestReportBCSMEvent** (oAnswer/oDisconnect) to track actual duration.
3. **Set reasonable durations** — typically 60–300 s for prepaid.
4. **Handle timeouts gracefully** when `release=false`.

Monitoring

Track ApplyCharging via the request history, the event log (`CAP:applyCharging`), the CAMEL Sessions page, and the `cap_requests_total{operation="apply_charging"}` metric.

Additional Operations

These operations are implemented in `cap_request_generator.ex` but are not part of the builder form.

InitialDPSMS (opcode 60)

`CapRequestGenerator.initial_dp_sms_request/5` builds an MO-SMS CAMEL trigger sent by the smsSSF to the gsmSCF when an SMS-CSI / mo-sms-CSI is armed, before the short message is relayed to the SMSC.

- Parameters: service key, calling party MSISDN, SM destination, IMSI, SMSC address (optional fields may be `nil`)
- The `InitialDPSMSArg` is encoded with the generated CAP ASN.1 module (`cap3` smsSSF-gsmSCF)
- The TCAP Begin carries the `id-ac-cap3-sms-AC` application context

NoteMM-Event (MAP opcode 89)

`CapRequestGenerator.note_mm_event_request/4` builds a CAMEL M-CSI mobility notification sent from the VLR to the gsmSCF when an armed mobility event occurs.

- Parameters: service key, mobility event (`:location_update_same_vlr`, `:location_update_other_vlr`, `:imsi_attach`, `:ms_initiated_imsi_detach`, `:network_initiated_imsi_detach`), IMSI, MSISDN
- Encoded with the **MAP** ASN.1 module via `:TCAPMessages.encode(:MapSpecificPDUs, …)`, carrying the `mm-EventReporting-v3` application context

> NoteMM-Event is a MAP operation, not a CAP operation — it is the one request here that does not use `:GenericTCAP`.

Implementation Details

State Management

- LiveView assigns hold the form state and the active OTID
- Request history is limited to 20 entries
- No auto-refresh — requests are sent manually

Request Generation

- Uses `CapRequestGenerator` to build TCAP/CAP structures
- CAP requests are encoded with `:GenericTCAP.encode(:GenericPDUs, …)` (works for both MAP and CAP transport). NoteMM-Event is the exception and uses `:TCAPMessages`.
- The result is wrapped in SCCP/M3UA via `CapClient.sccp_m3ua_maker/2`

Sending Mechanism

- Sent over M3UA to the `:camelgw_client_asp` ASP using routing context 1
- SCCP/M3UA encapsulation is automatic; SSNs default to 146 (gsmSSF)

Error Handling

- Form validation with user feedback
- Graceful handling of a missing OTID
- Parse and encoding errors surfaced in the UI and logs

Integration Notes

- Shares the event log with MAP events (`paklog`)
- Uses the same SCCP/M3UA infrastructure
- Works alongside the CAMEL Sessions page for monitoring
- Integrates with the existing M3UA routing

Common Features Guide

← Back to Main Documentation

This guide covers features common to all OmniSS7 operating modes.

Table of Contents

Web UI Overview

The Web UI is served over HTTPS on port `8087` (TLS is enforced via `force_ssl`), e.g. `https://localhost:8087/`.

Main Navigation

The menu is built from the pages enabled in `config/config.exs` and ordered by `page_order`. Common entries (menu label in bold):

- **Resources** (`/application`) - Application status and runtime information
- **Configuration** (`/configuration`) - System configuration viewer
- **SS7 Events** (`/events`) - Real-time SS7 signaling events and message logs
- **System Logs** (`/logs`) - Application log viewer
- **SS7 Client** (`/client`) - Interactive MAP request test client
- **Peers** (`/m3ua`) - M3UA/M2PA peer connections
- **Routing** (`/routing`) - Point Code and Global Title routes
- **Routing Test** (`/routing_test`) - Route resolution test tool
- **SMSc Links** (`/smsc_links`) - SMS backend links (SMSc)
- **SMSc Subscribers** (`/smsc_subscribers`) - Tracked SMS subscribers (SMSc)
- **HLR Links** (`/hlr_links`) - HLR backend links (HLR)
- **Active Subscribers** (`/subscribers`) - Tracked UpdateLocation subscribers

- **CAMEL Sessions** (`/camel_sessions`) - Live CAMEL call sessions (CAMEL GW)
- **CAMEL Request Builder** (`/camel_request`) - Interactive CAP request builder (CAMEL GW)
- **MSISDN/IMSI Test** (`/msisdn_imsi_test`) - Number translation test tool

Accessing the Web UI

1. Open your web browser
2. Navigate to the configured host on port 8087 (e.g., `https://localhost:8087/`)
3. View the system status dashboard

Swagger API Documentation

Interactive API documentation is served by the MAP operation API on HTTP port 8080:

```
http://your-server:8080/swagger
```

Web UI Configuration

The web UI endpoint is configured under `config :control_panel, ControlPanelWeb.Endpoint` in `config/config.exs`, and the navigation menu under `config :control_panel`:

```
config :control_panel, ControlPanelWeb.Endpoint,
  force_ssl: [rewrite_on: [:x_forwarded_proto]],
  server: true,
  url: [host: "0.0.0.0", path: "/"],
  https: [
    port: 8087,
    keyfile: "priv/cert/omnitouch.pem",
    certfile: "priv/cert/omnitouch.crt"
  ]

config :control_panel,
  page_order: [
    "/application", "/configuration", "/events", "/logs",
"/client",
    "/m3ua", "/routing", "/routing_test", "/smsc_links",
"/smsc_subscribers",
    "/hlr_links", "/subscribers", "/camel_sessions",
"/camel_request",
    "/msisdn_imsi_test"
  ]
```

Configuration Parameters:

Parameter	Type	Default
`ControlPanelWeb.Endpoint.url.host`	String	`"0.0.0.0"`
`ControlPanelWeb.Endpoint.https.port`	Integer	`8087`
`ControlPanelWeb.Endpoint.https.keyfile`	String	`"priv/cert/omnit`
`ControlPanelWeb.Endpoint.https.certfile`	String	`"priv/cert/omnit`
`ControlPanelWeb.Endpoint.force_ssl`	Keyword	`[rewrite_on: [:x_forwarded_pr`
`page_order`	List of Strings	see example above

> See the Configuration Reference for the full Control Panel schema.

Logger Configuration

Logs are emitted as structured JSON (one object per line) by `OmniLogger.JsonFormatter`, which includes all `Logger.metadata` so per-process context (STP routing OPC/DPC/GTs, SMS gateway to/from MSISDNs, transaction otid/operation) is carried automatically. This format is configured in `config/config.exs`:

```
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all,
  truncate: :infinity
```

The log level is set separately:

```
config :logger,
  level: :debug  # Options: :debug, :info, :warning, :error
```

Configuration Parameters:

Parameter	Type	Default
`:logger.level`	Atom	`:debug`
`:logger.default_formatter.format`	Tuple	`{OmniLogger.JsonF :format}`
`:logger.default_formatter.metadata`	Atom/List	`:all`
`:logger.default_formatter.truncate`	Integer/Atom	`:infinity`

API Documentation

OmniSS7 exposes two HTTP surfaces:

Surface	Port	Protocol	Contents
MAP operation API	`8080`	HTTP	MAP `/api/*` operations, `/swagger`, `/swagger.json`, `/metrics`
JSON status API (`api_ex`)	`8445`	HTTPS	Read-only status (`/api/status`, `/api/m3ua-status`, `/api/routing`, `/api/subscribers`, `/api/camel-sessions`, …)

MAP API Base URL

```
http://your-server:8080/api
```

Response Codes

- **200** - Success
- **400** - Bad Request
- **504** - Gateway Timeout

OpenAPI Specification

```
http://your-server:8080/swagger.json
```

Monitoring and Metrics

Prometheus Metrics Endpoint

```
http://your-server:8080/metrics
```

Key Metrics Categories

M3UA/SCTP Metrics:

- SCTP association state changes
- M3UA ASP state transitions
- Protocol data units sent/received

M2PA Metrics:

- Link state transitions (DOWN → ALIGNMENT → PROVING → READY)
- Messages and bytes sent/received per link
- Link-specific errors (decode, encode, SCTP)

STP Metrics:

- Messages received/sent per peer
- Routing failures by reason
- Traffic distribution across peers

MAP Client Metrics:

- MAP requests by operation type
- Request duration histograms
- Pending transactions gauge

CAP Metrics:

- CAP requests by operation type
- CAMEL gateway operations

SMSc Metrics:

- Queue depth
- Delivery rates
- Failed messages

Grafana Integration

OmniSS7 metrics are compatible with Prometheus and Grafana.

Best Practices

Security Recommendations

1. **Network Isolation**
    - Deploy in dedicated VLAN
    - Firewall rules to restrict access
    - Allow SCTP only from known addresses
2. **Web UI Security**
    - Enable TLS for production
    - Use reverse proxy with authentication
    - Restrict to management IPs
3. **API Security**
    - Implement rate limiting
    - Use API keys or OAuth
    - Log all requests for audit

Performance Tuning

1. **TPS Limits**

- Configure appropriate TPS
- Monitor system load
- Adjust SCTP buffers

2. **SMS Backend Tuning**
   - Keep the SMS backend API responsive (requests time out after 5 seconds)
   - Tune `auto_flush_tps` and `auto_flush_interval` to match backend capacity
   - Monitor queue depth and delivery rates
3. **M3UA Tuning**
   - Adjust SCTP heartbeat intervals
   - Configure timeout values
   - Use multiple links for redundancy

SCTP Multihoming for Network Redundancy

What is SCTP Multihoming?

SCTP Multihoming is a built-in feature of the SCTP protocol that allows a single M3UA connection to bind to multiple IP addresses on the same network interface or across different network interfaces. This provides automatic failover and redundancy at the transport layer.

Key Benefits:

- **Automatic Failover**: If one network path fails, SCTP automatically switches to an alternate path without dropping the connection
- **Zero Configuration Failover**: No application-level logic needed - SCTP handles path monitoring and failover

- **Improved Reliability**: Survive network failures, switch failures, or NIC failures
- **Load Balancing**: SCTP can distribute traffic across multiple paths (implementation-dependent)

How It Works

When you configure multiple IP addresses for an M3UA connection, SCTP:

1. **Binds to all IPs**: The socket binds to all configured IP addresses simultaneously
2. **Monitors paths**: SCTP continuously sends heartbeat packets on all paths to monitor their health
3. **Detects failures**: If heartbeats fail on the primary path, SCTP marks it as unreachable
4. **Automatic failover**: Traffic immediately switches to a backup path without application intervention
5. **Path recovery**: When the failed path recovers, SCTP detects it and marks it available again

Configuration

SCTP multihoming is configured by providing a **list of IP addresses** instead of a single IP tuple.

Single IP (Traditional)

```
# Single IP - no multihoming
local_ip: {10, 179, 4, 10}
```

Multiple IPs (Multihoming Enabled)

```
# Multiple IPs - multihoming enabled
# First IP is primary, subsequent IPs are backup paths
local_ip: [{10, 179, 4, 10}, {10, 179, 4, 11}]
```

Configuration Examples

> **Important Note for Server Role (Inbound Connections):**
>
> When configuring peers with `role: :server` (accepting inbound connections from multihomed peers), you must specify `remote_ip` as a **single tuple** - the IP address the remote peer uses to initiate the SCTP connection (SCTP INIT). Do NOT use a list.
>
> **Why?** The STP matches incoming connections based on the source IP of the SCTP INIT packet. SCTP will automatically discover the peer's other multihomed IP addresses during the association handshake. The list format for `remote_ip` is only valid for `role: :client` (outbound connections).
>
> **Examples:**
>
> ```
> # ✓ CORRECT - Server role with single remote_ip
> %{
>   role: :server,
>   remote_ip: {10, 0, 2, 100},  # Single tuple
>   # ...
> }
>
> # ✗ WRONG - Server role with list will NOT match incoming
> connections
> %{
>   role: :server,
>   remote_ip: [{10, 0, 2, 100}, {10, 0, 2,
> 101}],  # List won't work
>   # ...
> }
> ```

Example 1: STP Peer with Multihoming (Client Role - Outbound)

```
# STP mode peer configuration (OUTBOUND connection)
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "Partner_STP_Redundant",
      role: :client,  # Outbound - we initiate the connection
      # Multihoming: bind to two local IPs for redundancy
      local_ip: [{213, 57, 23, 200}, {213, 57, 23, 201}],
      local_port: 0,
      # Remote peer also supports multihoming - list is OK for
client role
      remote_ip: [{213, 57, 23, 100}, {213, 57, 23, 101}],
      remote_port: 2905,
      routing_context: 1,
      point_code: 100,
      network_indicator: :international
    }
  ]
```

Example 2: MAP Client with Multihoming

```
# MAP client mode with multihoming
config :omniss7,
  map_client_enabled: true,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # Multihoming: two local IPs for failover
    local_ip: [{10, 0, 0, 100}, {10, 0, 0, 101}],
    local_port: 2905,
    # Remote STP with multihoming support
    remote_ip: [{10, 0, 0, 1}, {10, 0, 0, 2}],
    remote_port: 2905,
    routing_context: 1
  }
```

Example 3: STP Listener with Multihoming

```
# Standalone STP server with multihoming
config :omniss7,
  sctp_handler: %{
    enabled: true,
    # Listen on multiple IPs for incoming connections
    local_ip: [{172, 16, 0, 10}, {172, 16, 0, 11}],
    local_port: 2905,
    point_code: 100
  }
```

Example 4: Server Role - Accepting Inbound from Multihomed Peer

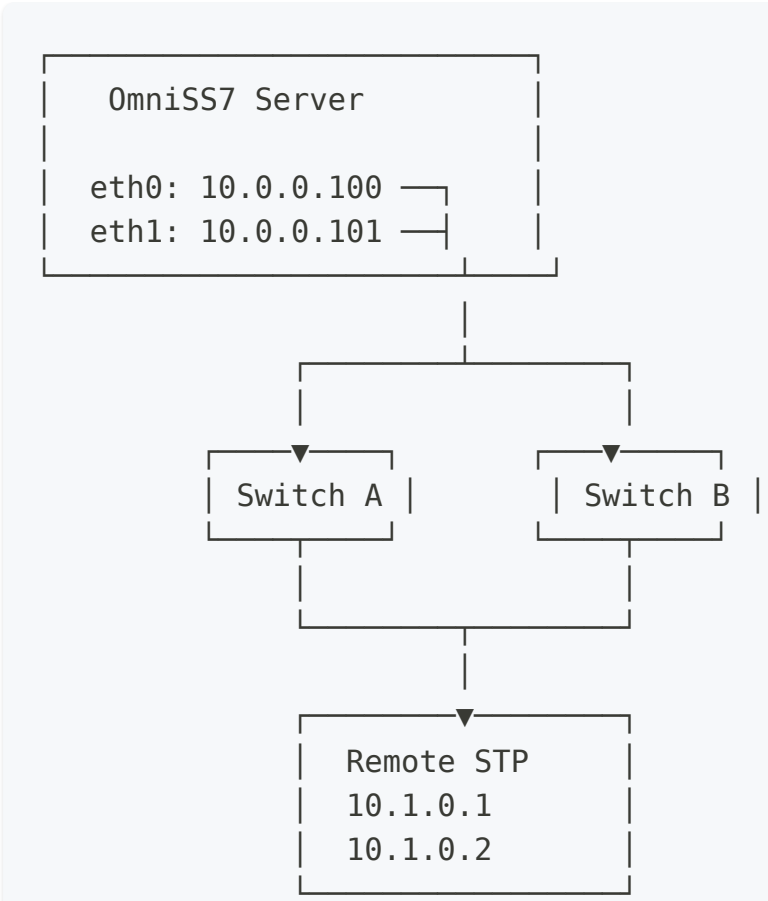
```
# Accepting inbound connection from a multihomed HLR
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "HLR",
      role: :server,  # INBOUND - HLR connects to us
      # Multihoming: our local IPs (list is OK)
      local_ip: [{10, 0, 1, 10}, {10, 0, 1, 11}],
      local_port: 2905,
      # IMPORTANT: Single tuple only - the IP the HLR uses to
initiate SCTP INIT
      # SCTP will auto-discover the HLR's other IPs during
handshake
      remote_ip: {10, 0, 2, 100},  # Primary IP only (NOT a list!)
      remote_port: 0,  # Accept from any source port
      routing_context: 1,
      point_code: 100,
      network_indicator: :international
    }
  ]
```

Example 5: Mixed Configuration (Backward Compatible)

```
# Mix of single and multi-homed peers
config :omniss7,
  m3ua_peers: [
    # Legacy peer - single IP
    %{
      peer_id: 1,
      name: "Legacy_STP",
      role: :client,
      local_ip: {10, 0, 0, 1},      # Single IP tuple
      local_port: 0,
      remote_ip: {10, 0, 0, 10},
      remote_port: 2905,
      routing_context: 1,
      point_code: 100
    },
    # New peer - multihoming
    %{
      peer_id: 2,
      name: "Redundant_STP",
      role: :client,
      local_ip: [{10, 0, 0, 2}, {10, 0, 0, 3}],  # IP list
      local_port: 0,
      remote_ip: [{10, 0, 0, 20}, {10, 0, 0, 21}],
      remote_port: 2905,
      routing_context: 2,
      point_code: 200
    }
  ]
```

Network Topology Scenarios

Scenario 1: Dual NICs (Common Deployment)



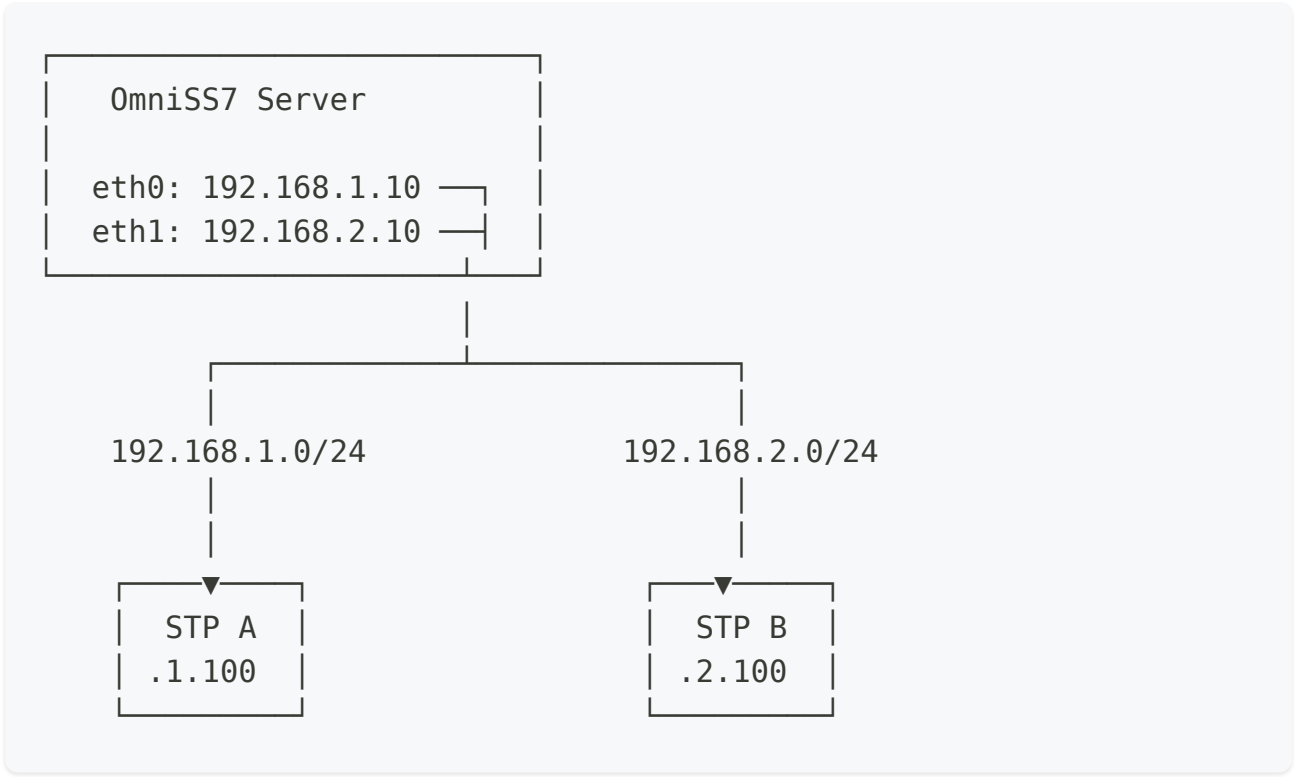
Configuration:

```
local_ip: [{10, 0, 0, 100}, {10, 0, 0, 101}]   # Both NICs
remote_ip: [{10, 1, 0, 1}, {10, 1, 0, 2}]      # Remote peer
```

Benefits:

- Survives failure of one NIC
- Survives failure of one switch
- Automatic failover in <1 second

Scenario 2: Multiple Subnets



Configuration:

```
local_ip: [{192, 168, 1, 10}, {192, 168, 2, 10}]
remote_ip: [{192, 168, 1, 100}, {192, 168, 2, 100}]
```

Benefits:

- Survives subnet failure
- Geographic redundancy possible
- Independent routing paths

Monitoring and Logging

When multihoming is enabled, you'll see log messages indicating the configuration:

Successful Multihoming

```
[info] SCTP client multihoming: bound 2 local IPs
[info] STP listener multihoming enabled: 2 local IPs bound
```

Path Failover Events

```
[warning] [MULTIHOMING] Path 10.0.0.100 is UNREACHABLE for peer
Partner_STP (assoc_id=1)
[info] [MULTIHOMING] Path 10.0.0.101 is now PRIMARY for peer
Partner_STP (assoc_id=1)
[info] [MULTIHOMING] Path 10.0.0.100 is now AVAILABLE for peer
Partner_STP (assoc_id=1)
```

Web UI Display

The Web UI automatically displays multihoming information:

M3UA Status Page:

- **Single IP**: Shows as `10.0.0.100`
- **Multiple IPs**: Shows as `10.0.0.100 (+1)` or `10.0.0.100 (+2)`
- **Details view**: Shows all IPs with primary/backup labels

Best Practices

1. Network Design

- **Use different NICs** for maximum redundancy
- **Different switches** to survive switch failures
- **Different subnets** if possible for routing diversity
- **Same datacenter initially** - test before geographic separation

2. IP Address Planning

- **First IP is primary** - ensure it's on the most reliable path
- **Order matters** - list IPs in order of preference
- **Consistent addressing** - use similar addressing schemes for troubleshooting

3. Testing Failover

```
# Disable primary interface to test failover
sudo ip link set eth0 down

# Monitor logs for failover
tail -f /var/log/omniss7.log | grep MULTIHOMING

# Re-enable interface
sudo ip link set eth0 up
```

4. Both Sides Should Support Multihoming

- **Optimal**: Both local and remote use multiple IPs
- **Acceptable**: Only one side uses multihoming
- **Note**: Redundancy is best when both endpoints support it

5. Firewall Configuration

```
# Allow SCTP on all multihoming IPs
iptables -A INPUT -p sctp --dport 2905 -s 10.0.0.0/24 -j ACCEPT
iptables -A INPUT -p sctp --dport 2905 -s 10.1.0.0/24 -j ACCEPT
```

Troubleshooting

Issue: Multihoming Not Working

Symptoms: Only primary IP is used, no failover

Checks:

1. Verify Erlang SCTP support: `erl -eval 'gen_sctp:open(9999, [binary, {ip, {127,0,0,1}}]).'`
2. Check kernel SCTP module: `lsmod | grep sctp`
3. Load SCTP if needed: `sudo modprobe sctp`
4. Verify both IPs are configured on system: `ip addr show`

Issue: Path Not Failing Over

Symptoms: Primary path marked down but traffic not switching

Checks:

1. Check SCTP heartbeat settings
2. Verify routing table has routes for all paths
3. Check firewall allows SCTP on all IPs
4. Review SCTP path monitoring logs

Issue: Frequent Path Flapping

Symptoms: Paths constantly switching between UP and DOWN

Checks:

1. Network instability - check physical links
2. SCTP heartbeat too aggressive - may need tuning
3. Firewall dropping SCTP heartbeats
4. MTU issues on one path

Performance Considerations

- **Minimal overhead**: SCTP heartbeats are small and infrequent
- **No application changes**: Multihoming is transparent to application layer
- **Fast failover**: Typically <1 second detection and failover
- **Automatic recovery**: No manual intervention needed

Compatibility

- **Backward compatible**: Single IP tuple format still works
- **Mixed deployments**: Can mix single-IP and multi-IP peers
- **All modes supported**: Works in STP, HLR, SMSc, and MAP Client modes
- **Erlang requirement**: Requires Erlang with SCTP support compiled in

Monitoring and Alerting

Key Metrics:

- M3UA connection state

- MAP request success rate
- API response times
- Message queue depth

Alert Thresholds:

- M3UA down > 1 minute
- MAP timeout rate > 10%
- Queue depth > 1000
- API error rate > 5%

Complete Configuration Reference

All Configuration Parameters

This section provides a complete reference of all available configuration parameters across all operating modes.

Logger Configuration (`:logger`)

```
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all,
  truncate: :infinity

config :logger,
  level: :debug  # :debug | :info | :warning | :error
```

Parameter	Type	Required	
`:logger.level`	Atom	No	`:debug`
`:logger.default_formatter.format`	Tuple	No	`{Omnil` `:forma`
`:logger.default_formatter.metadata`	Atom/List	No	`:all`
`:logger.default_formatter.truncate`	Integer/Atom	No	`:infi`

Web UI Configuration (`:control_panel`)

```
config :control_panel, ControlPanelWeb.Endpoint,
  force_ssl: [rewrite_on: [:x_forwarded_proto]],
  server: true,
  url: [host: "0.0.0.0", path: "/"],
  https: [
    port: 8087,
    keyfile: "priv/cert/omnitouch.pem",
    certfile: "priv/cert/omnitouch.crt"
  ]

config :control_panel,
  page_order: [
    "/application", "/configuration", "/events", "/logs",
"/client",
    "/m3ua", "/routing", "/routing_test", "/smsc_links",
"/smsc_subscribers",
    "/hlr_links", "/subscribers", "/camel_sessions",
"/camel_request",
    "/msisdn_imsi_test"
  ]
```

Parameter	Type	Required	
`ControlPanelWeb.Endpoint.url.host`	String	No	`"0.0.`
`ControlPanelWeb.Endpoint.https.port`	Integer	No	`8087`
`ControlPanelWeb.Endpoint.https.keyfile`	String	No	`"priv`
`ControlPanelWeb.Endpoint.https.certfile`	String	No	`"priv`
`ControlPanelWeb.Endpoint.force_ssl`	Keyword	No	`[rewr` `[:x_f`
`page_order`	List of Strings	No	see e

SCTP SocketHandler Configuration (`:omniss7`)

```
config :omniss7,
  sctp_handler: %{
    enabled: false,
    local_ip: {127, 0, 0, 1},
    local_port: 2905
  },
  enable_gt_routing: true,
  m3ua_peers: [...],
  m3ua_routes: [...],
  m3ua_gt_routes: [...]
```

Parameter	Type	Required	Default	Description
`sctp_handler.enabled`	Boolean	Yes	`false`	Enable STP mode at boot
`sctp_handler.local_ip`	Tuple	Yes	`{127, 0, 0, 1}`	IP to bind for incoming M3UA
`sctp_handler.local_port`	Integer	Yes	`2905`	SCTP port for M3UA
`enable_gt_routing`	Boolean	No	`false`	Enable Global Title routing

M3UA Peer Parameters:

Parameter	Type	Required	Description
`peer_id`	Integer	Yes	Unique peer identifier
`name`	String	Yes	Descriptive peer name
`role`	Atom	Yes	`:client` or `:server`
`local_ip`	Tuple or List	If `:client`	Local IP(s) to bind. Single: `{10, 0, 0, 1}` or List: `[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`local_port`	Integer	If `:client`	Local port (0 for dynamic)
`remote_ip`	Tuple or List	Yes	Remote peer IP(s). Single: `{10, 0, 0, 10}` or List: `[{10, 0, 0, 10}, {10, 0, 0, 11}]`
`remote_port`	Integer	If `:client`	Remote peer port
`routing_context`	Integer	Yes	M3UA routing context
`point_code`	Integer	Yes	SS7 point code
`network_indicator`	Atom	No	`:international` or `:national`

M3UA Route Parameters:

Parameter	Type	Required	Description
`dest_pc`	Integer	Yes	Destination point code
`peer_id`	Integer	Yes	Peer to route through
`priority`	Integer	Yes	Route priority (lower = higher priority)
`network_indicator`	Atom	No	`:international` or `:national`

M3UA GT Route Parameters:

Parameter	Type	Required	Description
`gt_prefix`	String	Yes	Global Title prefix to match
`peer_id`	Integer	Yes	Destination peer
`priority`	Integer	Yes	Route priority
`description`	String	No	Route description for logging
`source_ssn`	Integer	No	Match only if source SSN matches
`dest_ssn`	Integer	No	Rewrite destination SSN to this value

MAP Client Configuration (`:omniss7`)

```
config :omniss7,
  map_client_enabled: false,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},
    remote_port: 2905,
    routing_context: 1
  }
```

Parameter	Type	Required	Default
`map_client_enabled`	Boolean	Yes	`false`
`map_client_m3ua.mode`	String	Yes	`"ASP"`
`map_client_m3ua.callback`	Tuple	Yes	`{MapClient, :handle_paylo []}`
`map_client_m3ua.process_name`	Atom	Yes	`:map_client_a`
`map_client_m3ua.local_ip`	Tuple	Yes	-
`map_client_m3ua.local_port`	Integer	Yes	`2905`
`map_client_m3ua.remote_ip`	Tuple	Yes	-
`map_client_m3ua.remote_port`	Integer	Yes	`2905`
`map_client_m3ua.routing_context`	Integer	Yes	-

SMS Center Configuration (`:omniss7`)

```
config :omniss7,
  auto_flush_enabled: false,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: nil,
  auto_flush_tps: 10
```

Parameter	Type	Required	Default	Description
`auto_flush_enabled`	Boolean	No	`false`	Enable auto-flush of SMS queue
`auto_flush_interval`	Integer	No	`10000`	Queue poll interval (milliseconds)
`auto_flush_dest_smsc`	String/nil	No	`nil`	Filter by dest SMSC (nil = all)
`auto_flush_tps`	Integer	No	`10`	Max transactions per second

HTTP API Configuration (`:omniss7`)

The SMS backend now uses HTTP API instead of direct database connections.

```
config :omniss7,
  smsc_api_base_url: "https://10.5.198.200:8443",
  smsc_name: "omni-smsc01"  # Optional: defaults to hostname-
derived name
```

API Parameters:

Parameter	Type	Required	Default
`smsc_api_base_url`	String	Yes	`"https://10.5.198.200:8443"`
`smsc_name`	String	No	hostname-derived

API Endpoints Used:

- `POST /api/frontends` - Register this frontend instance with backend
- `POST /api/messages_raw` - Insert new SMS messages
- `GET /api/messages` - Retrieve message queue (with `smsc` header)
- `PATCH /api/messages/{id}` - Mark message as delivered
- `PUT /api/messages/{id}` - Update message status
- `POST /api/events` - Add event tracking
- `GET /api/status` - Health check endpoint

Frontend Registration:

The system automatically registers itself with the backend API on startup and re-registers approximately every 85 seconds. Registration includes:

- Frontend name and type (reported as `SS7`)
- Hostname
- Uptime in seconds
- Configuration details (JSON format)

Configuration Notes:

- SSL verification is disabled by default for self-signed certificates

- HTTP requests timeout after 5 seconds
- All timestamps are in ISO 8601 format
- The API uses JSON for request/response bodies

Related Documentation

- ← Back to Main Documentation
- STP Guide
- MAP Client Guide
- SMS Center Guide
- HLR Guide

OmniSS7 by Omnitouch Network Services

Configuration Reference

← Back to Main Documentation

This document provides a comprehensive reference for all OmniSS7 configuration parameters.

Table of Contents

Overview

OmniSS7 supports four operational modes, selected entirely through configuration:

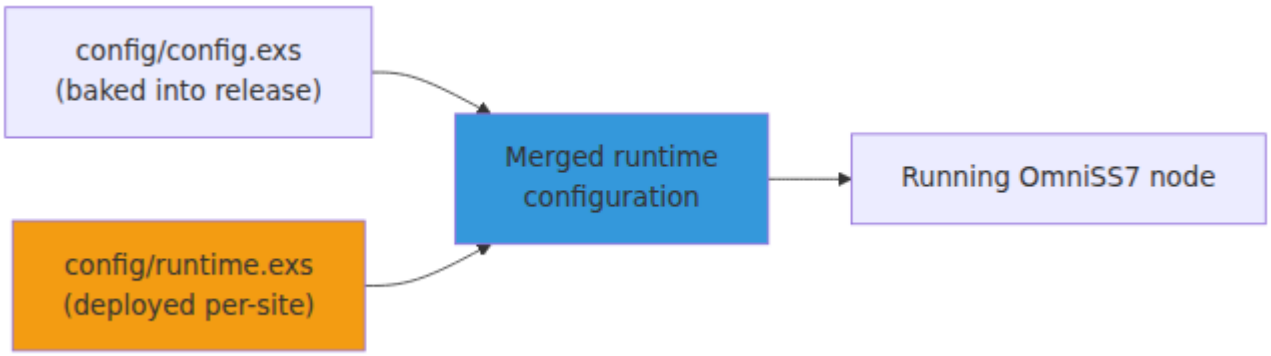
- **STP Mode** - Signal Transfer Point for routing
- **HLR Mode** - Home Location Register for subscriber management
- **SMSc Mode** - SMS Center for message delivery
- **CAMEL GW Mode** - CAMEL Gateway for intelligent call control

A single node may combine roles (for example HLR + SMSc) by enabling the relevant Operational Mode Flags.

Where Configuration Lives

OmniSS7 reads its configuration from two layers. Understanding which file owns a value determines where you change it on a deployed system.

File	Role	Edit on deployment?
`config/config.exs`	**Default / compile-time configuration** shipped with the release. Defines the control-panel pages, the `:api_ex` REST controllers, the logging formatter, and all baseline defaults.	No — part of the build.
`config/runtime.exs` (in the repo)	A **4-line test stub** only. In the source tree it sets up nothing beyond the test environment.	No.
`config/runtime.exs` (deployed)	The **per-deployment runtime configuration**, deployed separately onto each server (typically rendered from an Ansible/Jinja template). This is where mode flags, GTs, point codes, M3UA peers, and API backends are set for a given site.	**Yes — this is the operator-facing file.**



> **In short:** examples in this document use `config :omniss7, ...`. On a live node these go into the **deployed** `config/runtime.exs`. The `config/runtime.exs` committed to the repository is intentionally a stub

> used only by the automated test suite — do not mistake it for the production configuration.

All keys below are read via `Application.get_env(:omniss7, ...)` (or the relevant application) at runtime, so the defaults shown are the values used when a key is omitted.

Operational Mode Flags

Control which features are enabled.

Parameter	Type	Default	Description	Modes
`map_client_enabled`	Boolean	`false`	Enable MAP client and M3UA connectivity	All
`hlr_mode_enabled`	Boolean	`false`	Enable HLR-specific features	HLR
`smsc_mode_enabled`	Boolean	`false`	Enable SMSc-specific features	SMSc
`cap_client_enabled`	Boolean	`false`	Enable CAP client for CAMEL operations	CAMEL GW
`camelgw_mode_enabled`	Boolean	`false`	Enable CAMEL Gateway features	CAMEL GW
`ussd_gateway_enabled`	Boolean	`false`	Enable USSD Gateway (HTTP/JSON bridge)	USSD GW

Example:

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false
```

HLR Mode Parameters

Configuration for HLR (Home Location Register) mode.

HLR API Configuration

Parameter	Type	Default	Required
`hlr_api_base_url`	String	-	**Yes**
`hlr_api_status_path`	String	`"/api/status"`	No
`hlr_service_center_gt_address`	String	-	**Yes**
`smsc_service_center_gt_address`	String	-	**Yes**

> **SSL verification:** There is **no** `hlr_api_verify_ssl` key — it is not read anywhere in the code. TLS certificate verification for backend API calls is governed by the generic `:api` map's `verify_ssl` field (see API Backend SSL Verification).

Example:

```
config :omniss7,
  hlr_api_base_url: "https://10.180.2.140:8443",
  hlr_api_status_path: "/api/status",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112"
```

AlertServiceCenter Configuration

When a subscriber performs UpdateLocation, the HLR sends alertServiceCenter messages to configured SMSc GTs to indicate the subscriber is now reachable.

Parameter	Type	Default	Required	
`hlr_smsc_alert_gts`	List of Strings	`[]`	No	Lis Gl se ale no
`hlr_alert_location_expiry_seconds`	Integer	`172800`	No	Lo du se 48 to

Example:

```
config :omniss7,
  hlr_smsc_alert_gts: [
    "15559876543",
    "15559876544"
  ],
  hlr_alert_location_expiry_seconds: 172800  # 48 hours
```

MSISDN ↔ IMSI Mapping

Configuration for synthetic IMSI generation from MSISDNs. For detailed technical explanation of the mapping algorithm, see MSISDN ↔ IMSI Mapping in HLR Guide.

Parameter	Type	Default	Required	Description
`hlr_imsi_plmn_prefix`	String	`"50557"`	No	PLMN prefix (MCC+MNC) for synthetic IMSI generation
`hlr_msisdn_country_code`	String	`"61"`	No	Country code prefix for IMSI→MSISDN reverse mapping
`hlr_msisdn_nsn_offset`	Integer	`0`	No	Offset into MSISDN where NSN starts (typically length of country code)
`hlr_msisdn_nsn_length`	Integer	`9`	No	Length of National Subscriber Number to extract from MSISDN

Example (2-digit country code):

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",      # MCC 505 + MNC 57
  hlr_msisdn_country_code: "99",      # Example 2-digit country
code
  hlr_msisdn_nsn_offset: 2,           # Skip 2-digit country code
  hlr_msisdn_nsn_length: 9            # Extract 9-digit NSN
```

Example (3-digit country code):

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",      # MCC 505 + MNC 57
  hlr_msisdn_country_code: "999",     # Example 3-digit country
code
  hlr_msisdn_nsn_offset: 3,           # Skip 3-digit country code
  hlr_msisdn_nsn_length: 8            # Extract 8-digit NSN
```

Important: Set `nsn_offset` to the length of your country code to properly extract the NSN. For example:

- Country code "9" (1 digit) → `nsn_offset: 1`
- Country code "99" (2 digits) → `nsn_offset: 2`
- Country code "999" (3 digits) → `nsn_offset: 3`

InsertSubscriberData (ISD) Configuration

Configuration for subscriber provisioning data sent to VLRs during UpdateLocation. For detailed explanation of the ISD sequence and message flow, see InsertSubscriberData Configuration in HLR Guide.

Parameter	Type	Default	Required	
`isd_network_access_mode`	Atom	`:packetAndCircuit`	No	N t : : :
`isd_send_ss_data`	Boolean	`true`	No	S S S
`isd_send_call_barring`	Boolean	`true`	No	S C

Example:

```
config :omniss7,
  isd_network_access_mode: :packetAndCircuit,
  isd_send_ss_data: true,
  isd_send_call_barring: true
```

CAMEL Configuration

Configuration for CAMEL-based intelligent call routing. For detailed explanation of CAMEL integration and service keys, see CAMEL Integration in HLR Guide.

Parameter	Type	Default	Re
`camel_service_key`	Integer	`11_110`	No
`camel_trigger_detection_point`	Atom	`:termAttemptAuthorized`	No
`camel_gsmscf_gt_address`	String	(uses called GT)	No

Example:

```
config :omniss7,
  camel_service_key: 11_110,
  camel_trigger_detection_point: :termAttemptAuthorized
```

Home VLR Prefixes

Configuration for distinguishing home vs roaming subscribers. For detailed explanation of home/roaming detection and PRN operations, see Roaming Subscriber Handling in HLR Guide.

Parameter	Type	Default	Required	Description
`home_vlr_prefixes`	List	`["5551231"]`	No	VLR GT prefixes considered "home" network

Example:

```
config :omniss7,
  home_vlr_prefixes: ["5551231", "5551234"]
```

SMSc Mode Parameters

Configuration for SMS Center mode.

SMSc API Configuration

Parameter	Type	Default
`smsc_api_base_url`	String	`"https://10.5.198.200:8443"`
`smsc_api_status_path`	String	`"/api/status"`
`smsc_name`	String	`"{hostname}_SMSc"`
`smsc_service_center_gt_address`	String	-

> **SSL verification:** There is **no** `smsc_api_verify_ssl` key — it is not read anywhere in the code. TLS certificate verification for backend API calls is governed by the generic `:api` map's `verify_ssl` field (see API Backend SSL Verification).

Example:

```
config :omniss7,
  smsc_api_base_url: "https://10.179.3.219:8443",
  smsc_api_status_path: "/api/status",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112"
```

Note: Frontend registration occurs every **5 minutes** (hardcoded) via `SMS.FrontendRegistry` module.

Auto-Flush Configuration

Parameter	Type	Default	Required	Description
`auto_flush_enabled`	Boolean	`true`	No	Enable automati SMS queue processing
`auto_flush_interval`	Integer	`10_000`	No	Queue processir interval in milliseconds
`auto_flush_dest_smsc`	String	-	**Yes**	Destination SMS name for auto-fl
`auto_flush_tps`	Integer	`10`	No	Message proces rate (transactions/se

Example:

```
config :omniss7,
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10
```

STP Mode Parameters

Configuration for M3UA Signal Transfer Point mode. For detailed routing configuration and examples, see the STP Configuration Guide.

Standalone STP Server

Parameter	Type	Default	Required	Description
`sctp_handler.enabled`	Boolean	`false`	No	Enable standalone SCTP SocketHandl server
`sctp_handler.local_ip`	Tuple or List	`{127, 0, 0, 1}`	No	IP address(es to listen for connections. Single IP: `{10, 0, 0, 1}` or Multipl IPs for SCTP multihoming `[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`sctp_handler.local_port`	Integer	`2905`	No	Port to listen on
`sctp_handler.point_code`	Integer	-	**Yes** (if enabled)	This STP's own SS7 poi code

Example (Single IP):

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  }
```

Example (SCTP Multihoming):

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    # Multiple IPs for redundancy
    local_ip: [{10, 179, 4, 10}, {10, 179, 4, 11}],
    local_port: 2905,
    point_code: 100
  }
```

> **Note**: For detailed information on SCTP multihoming configuration and benefits, see SCTP Multihoming in Common Guide.

Global Title Routing

Parameter	Type	Default	Required	Description
`enable_gt_routing`	Boolean	`false`	No	Enable GT routing in addition to PC routing

Example:

```
config :omniss7,
  enable_gt_routing: true
```

M3UA/M2PA Peer Configuration

Peers are configured via the `m3ua_peers` list (supports both M3UA and M2PA protocols). For complete configuration examples, see the STP Configuration Guide and M2PA Protocol Support.

Common Peer Parameters:

Parameter	Type	Default	Required	Descri
`peer_id`	Integer	-	**Yes**	Unique p identifier
`name`	String	-	**Yes**	Descripti name
`protocol`	Atom	`:m3ua`	No	Protocol t `:m3ua` or
`role`	Atom	`:client`	No	Connecti `:client` `:server` or `:sgp`
`local_ip`	Tuple or List	-	**Yes**	Local IP address(binding
`local_port`	Integer	-	**Yes**	Local SC (M3UA: 2 M2PA: 35
`remote_ip`	Tuple or List	-	**Yes**	Remote I address(
`remote_port`	Integer	-	**Yes**	Remote S port
`routing_context`	Integer	-	No	M3UA rou context identifier only)
`point_code`	Integer	-	**Yes**	Local poi

Parameter	Type	Default	Required	Descri
`network_indicator`	Atom	`:international`	No	Network indicator `:interna` or `:nati`
`initiate_connection`	Boolean	`true`	No	Whether initiate S connecti

M2PA-Specific Parameters:

Parameter	Type	Default	Required	Description
`adjacent_point_code`	Integer	-	**Yes** (M2PA)	Adjacent peer's point code

Socket Management:

M2PA automatically uses SCTP.SocketHandler for shared socket management. All M2PA peers use the shared socket, which allows multiple peers to efficiently share the same SCTP port. For detailed configuration, see M2PA Socket Requirements.

Example (M3UA Peer):

```
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "HLR_East",
      protocol: :m3ua,
      role: :sgp,
      local_ip: {10, 179, 4, 10},
      local_port: 2905,
      remote_ip: {10, 179, 4, 20},
      remote_port: 2905,
      point_code: 100,
      network_indicator: :international
    }
  ]
```

Example (M2PA Peer):

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 3565,
    point_code: 100
  },
  m3ua_peers: [
    %{
      peer_id: 2,
      name: "M2PA_Link_STP_West",
      protocol: :m2pa,
      role: :client,
      local_ip: {10, 179, 4, 10},
      local_port: 3565,
      remote_ip: {10, 179, 4, 30},
      remote_port: 3565,
      point_code: 100,
      adjacent_point_code: 200
    }
  ]
```

M3UA Point Code Routes

Point Code routing configuration. Routes define which peer to use for reaching specific destination point codes.

Parameter	Type	Default	Required	Description
`m3ua_routes`	List of Maps	`[]`	No	List of point code routes. If not specified, routes are auto-generated from peer point codes.

Route Format: Each route in `m3ua_routes` must be a map with:

- `dest_pc`: Destination point code (Integer)
- `peer_id`: Peer ID to route through (Integer)
- `priority`: Route priority - lower value = higher priority (Integer)
- `network_indicator`: Network indicator (Atom): `:international` or `:national`

Example:

```
config :omniss7,
  m3ua_routes: [
    # Route to PC 100 via peer 1 (highest priority)
    %{dest_pc: 100, peer_id: 1, priority: 1, network_indicator:
:international},
    # Route to PC 200 via peer 2
    %{dest_pc: 200, peer_id: 2, priority: 1, network_indicator:
:international},
    # Load balancing: same dest_pc with different priorities
    %{dest_pc: 300, peer_id: 3, priority: 1, network_indicator:
:international},
    %{dest_pc: 300, peer_id: 4, priority: 2, network_indicator:
:international}
  ]
```

M3UA Global Title Routes

Global Title prefix-based routing with advanced SCCP parameter transformation. Longest prefix match is used first, then priority.

Parameter	Type	Default	Required	Description
`m3ua_gt_routes`	List of Maps	`[]`	No	List of Global Title routing rules with optional transformations

Route Format: Each route in `m3ua_gt_routes` must be a map with:

Basic Parameters:

- `gt_prefix`: Global Title prefix to match (String) - empty string matches all
- `peer_id`: Peer ID to route through (Integer) - use 0 to DROP traffic
- `priority`: Route priority - lower value = higher priority (Integer)
- `description`: Human-readable description (String)

Optional Matching Parameters:

- `source_ssn`: Match source SubSystem Number (Integer)
- `source_tt`: Match source Translation Type (Integer)
- `source_npi`: Match source Numbering Plan Indicator (Integer)
- `source_nai`: Match source Nature of Address Indicator (Integer)

Optional Transformation Parameters:

- `dest_ssn`: Transform SSN in forwarded message (Integer)
- `dest_tt`: Transform Translation Type in forwarded message (Integer)
- `dest_npi`: Transform Numbering Plan Indicator in forwarded message (Integer)
- `dest_nai`: Transform Nature of Address Indicator in forwarded message (Integer)

Common Values:

- **Translation Type (TT):** 0=Unknown, 1=International, 2=National, 3=Network Specific
- **Numbering Plan (NPI):** 0=Unknown, 1=ISDN(E.164), 6=Mobile(E.212)
- **Nature of Address (NAI):** 0=Unknown, 1=Subscriber, 3=National, 4=International
- **SubSystem Number (SSN):** 6=HLR, 7=VLR, 8=MSC, 9=EIR, etc.

Example:

```
config :omniss7,
  m3ua_gt_routes: [
    # Basic prefix routing
    %{gt_prefix: "1234", peer_id: 1, priority: 1, description: "US
numbers"},
    %{gt_prefix: "44", peer_id: 2, priority: 1, description: "UK
numbers"},

    # Translation Type transformation
    %{
      gt_prefix: "61",
      peer_id: 3,
      priority: 1,
      description: "Australian numbers: TT 0→1 transformation",
      source_tt: 0,  # Match TT=0 (Unknown)
      dest_tt: 1     # Transform to TT=1 (International)
    },

    # NPI transformation
    %{
      gt_prefix: "49",
      peer_id: 1,
      priority: 1,
      description: "German numbers: Mobile→ISDN NPI conversion",
      source_npi: 6,  # Match NPI=6 (Mobile/E.212)
      dest_npi: 1     # Transform to NPI=1 (ISDN/E.164)
    },

    # Combined transformation with SSN routing
    %{
      gt_prefix: "86",
      source_ssn: 8,   # Match SSN=8 (MSC)
      peer_id: 3,
      dest_ssn: 6,     # Rewrite to SSN=6 (HLR)
      priority: 1,
      description: "Chinese traffic: Full normalization",
      source_tt: 0,
      dest_tt: 2,
      source_npi: 6,
      dest_npi: 1,
      source_nai: 4,
      dest_nai: 3
    },
```

```
    # Default/Fallback route
    %{
      gt_prefix: "",
      peer_id: 1,
      priority: 99,
      description: "Default fallback route"
    }
  ]
```

CAMEL Gateway Mode Parameters

Configuration for CAMEL Gateway (CAP protocol) mode.

CAMEL Mode Flags

Enable CAMEL/CAP features (set `cap_client_enabled: true` and `camelgw_mode_enabled: true` in Operational Mode Flags).

CAP Protocol Configuration

Parameter	Type	Default	Required	Description
`cap_version`	Atom	`:v2`	No	CAP protocol version: `:v1`, `:v2`, `:v3`, or `:v4`
`camel_gsmscf_gt_address`	String	(uses called GT)	No	Default gsmSCF Global Title for CAMEL responses

CAP Version Mapping:

- `:v1` → Application Context OID: 0.4.0.0.1.0.50.0
- `:v2` → Application Context OID: 0.4.0.0.1.0.50.1 (default - most widely supported)
- `:v3` → Application Context OID: 0.4.0.0.1.21.3.4
- `:v4` → Application Context OID: 0.4.0.0.1.23.3.4

Note: Incoming requests are auto-detected from their application context OID and responses match the request version.

Example:

```
config :omniss7,
  cap_client_enabled: true,
  camelgw_mode_enabled: true,
  cap_version: :v2,
  camel_gsmscf_gt_address: "68988411553"
```

CGrateS Integration

Real-time charging integration with CGrateS for prepaid/postpaid billing.

Parameter	Type	Default	Required	Descr
`cgrates_enabled`	Boolean	`false`	No	Enable C integrati
`cgrates_url`	String	-	**Yes** (if enabled)	CGrateS endpoint
`cgrates_tenant`	String	`"cgrates.org"`	No	CGrateS identifier
`cgrates_request_type`	String	`"*prepaid"`	No	Charging `"*prepai` `"*postpa` `"*pseudo`
`cgrates_timeout`	Integer	`5000`	No	CGrateS timeout i milliseco

Example:

```
config :omniss7,
  cgrates_enabled: true,
  cgrates_url: "http://localhost:2080/jsonrpc",
  cgrates_tenant: "cgrates.org",
  cgrates_request_type: "*prepaid",
  cgrates_timeout: 5000
```

CAP M3UA Connection

CAMEL Gateway uses a separate M3UA connection for CAP operations.

Parameter	Type	Default	Required	Description
`cap_client_m3ua`	Map	-	**Yes**	CAP M3UA connection configuration (same structure as `map_client_m3ua`, including optional `opc` and `dpc` parameters)

Example:

```
config :omniss7,
  cap_client_m3ua: %{
    mode: "ASP",
    callback: {CapClient, :handle_payload, []},
    process_name: :camelgw_client_asp,
    local_ip: {10, 5, 198, 200},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 4,
    opc: 5013,    # Originating Point Code
    dpc: 5011     # Destination Point Code
  }
```

Global Title NAT Parameters

Global Title Network Address Translation allows different response GTs based on calling party prefix, called party prefix, or both. Rules are matched by weight (lower = higher priority), then prefix specificity (longer combined prefix = more specific). For detailed explanation and examples, see the Global Title NAT Guide.

Parameter	Type	Default	Required	Description
`gt_nat_enabled`	Boolean	`false`	No	Enable/disable GT NAT feature
`gt_nat_rules`	List of Maps	`[]`	**Yes** (if enabled)	List of GT NAT rules with prefix matching

Rule Format: Each rule in `gt_nat_rules` must be a map with:

- `calling_prefix`: String prefix to match against calling GT (optional)
- `called_prefix`: String prefix to match against called GT (optional)
- `weight`: Integer priority (lower = higher priority) - default: 100
- `response_gt`: Global Title to use in responses (required)

Matching Priority:

1. Rules are matched by `weight` (lower value = higher priority)
2. If weights are equal, longer combined prefix length wins
3. Both `calling_prefix` and `called_prefix` can be used together for precise matching

Example:

```
config :omniss7,
  gt_nat_enabled: true,
  gt_nat_rules: [
    # High priority: Match both calling from "8772" AND called to
"555"
    %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

    # Medium priority: Match only calling from "8772"
    %{calling_prefix: "8772", weight: 10, response_gt:
"68988411553"},

    # Medium priority: Match only called to "555"
    %{called_prefix: "555", weight: 10, response_gt:
"68988411554"},

    # Match only calling from "8773"
    %{calling_prefix: "8773", weight: 10, response_gt:
"68988411554"},

    # Wildcard fallback rule (matches everything, highest weight)
    %{weight: 100, response_gt: "68988411555"}
  ]
```

See Also: GT NAT Guide for detailed usage and examples.

M3UA Connection Parameters

M3UA connection configuration for MAP client mode. For detailed usage and examples, see the MAP Client Guide.

Parameter	Type	Default	Required
`map_client_m3ua.mode`	String	-	**Yes**
`map_client_m3ua.callback`	Tuple	-	**Yes**
`map_client_m3ua.process_name`	Atom	-	**Yes**
`map_client_m3ua.local_ip`	Tuple or List	-	**Yes**
`map_client_m3ua.local_port`	Integer	`2905`	**Yes**
`map_client_m3ua.remote_ip`	Tuple or List	-	**Yes**
`map_client_m3ua.remote_port`	Integer	`2905`	**Yes**
`map_client_m3ua.routing_context`	Integer	-	**Yes**
`map_client_m3ua.opc`	Integer	`5013`	No

Parameter	Type	Default	Required
`map_client_m3ua.dpc`	Integer	`5011`	No
`map_client_m3ua.receive_watchdog`	Boolean	`true`	No
`map_client_m3ua.receive_watchdog_idle`	Integer	`15`	No

Example (Single IP):

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1,
    opc: 5013,    # Originating Point Code (2-114-5)
    dpc: 5011     # Destination Point Code (2-114-3)
  }
```

> **Point Code Format**: Point codes in `X-Y-Z` format convert to integers as: `(X * 2048) + (Y * 8) + Z`. For example, `2-114-5` = `(2 * 2048) + (114 * 8) + 5` = `5013`.

Example (SCTP Multihoming):

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # Multiple local IPs for redundancy
    local_ip: [{10, 179, 4, 11}, {10, 179, 4, 12}],
    local_port: 2905,
    # Multiple remote IPs for STP redundancy
    remote_ip: [{10, 179, 4, 10}, {10, 179, 4, 20}],
    remote_port: 2905,
    routing_context: 1
  }
```

> **Note**: For detailed information on SCTP multihoming configuration and benefits, see SCTP Multihoming in Common Guide.

Receive Watchdog

The **receive watchdog** monitors SCTP connections for zombie sockets — associations that remain in an ESTABLISHED state at the OS level but where the remote end has silently stopped sending data. Without the watchdog, a dead connection may not be detected until the next send attempt fails.

Any received SCTP payload resets the idle timer, including M3UA BEAT responses, NOTIFY messages, and application data. Per RFC 4666 §3.8, M3UA BEAT is optional — SCTP already performs mandatory heartbeating at the transport layer. In environments where the remote SG does not send periodic application-layer traffic (and does not send M3UA BEATs), disabling the watchdog avoids unnecessary reconnect cycles.

Parameter	Type	Default	Description
`receive_watchdog`	Boolean	`true`	Enable or disable the receive watchdog. When `false`, idle connections are never torn down by the watchdog; SCTP transport-layer heartbeating still operates normally.
`receive_watchdog_idle`	Integer	`15`	Seconds of inactivity before the watchdog tears down the connection and triggers a reconnect. Has no effect when `receive_watchdog: false`.

Example — Disabling the watchdog for an ASP client:

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1,
    receive_watchdog: false        # Disable — remote SG does not
send periodic traffic
  }
```

Example — Customising the idle threshold:

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1,
    receive_watchdog: true,
    receive_watchdog_idle: 30      # Tear down after 30 s of
silence (default: 15 s)
  }
```

SCCP Subsystem Status

OmniSS7 answers SCCP-management Subsystem-Status-Tests (SST) from the real status of the queried local subsystem (ITU-T Q.714 §5.3.4). The `local_subsystems` key declares which SubSystem Numbers (SSNs) this node actually hosts. An SST for a listed SSN is answered `Subsystem-Allowed` (SSA); any other SSN is answered `Subsystem-Prohibited` (SSP).

Parameter	Type	Default	Required	Description
`local_subsystems`	List of Integers or `:all`	`:all`	No	In-service local SCCP SSNs used to answer SST. `:all` (the default) treats every subsystem as available — appropriate for a pure relay that hosts no subsystems. Operators that terminate specific SSNs should list them so SST reflects reality.

Common SSNs: 6 = HLR, 7 = VLR, 8 = MSC, 9 = EIR, 146 = gsmSCF/CAMEL.

Example:

```
config :omniss7,
  local_subsystems: [6, 7, 8, 146]
```

> Implemented in `lib/ss7_logic/subsystem_status.ex`.

Gateway Screening

Gateway screening (the SCCP firewall) permits or denies transit traffic before it is routed. Rules match on source peer, OPC, DPC, Service Indicator, SCCP Called/Calling-Party GTA prefix and SSN; the first matching enabled rule (lowest priority) decides `allow`/`deny`. When no rule matches, the default action below applies. Denied messages are dropped and counted as

`m3ua_stp_routing_failures_total{reason="screened"}`. See the STP Guide for the full model.

Parameter	Type	Default	Required	Description
`screening_default_action`	`:allow` or `:deny`	`:allow`	No	Action applied to a transit message when no screening rule matches. `:allow` (default) is a default-permit posture; `:deny` is default-deny (allowlist). Setting `:deny` makes screening active even with no rules configured.

Example:

```
config :omniss7,
  # Default-deny: only traffic matching an explicit allow rule is
forwarded.
  screening_default_action: :deny
```

Rules themselves are held in the `ss7_screen_rule` Mnesia table and managed programmatically; only the default action is set via config.

> Implemented in `lib/ss7_logic/screening.ex`.

USSD Gateway Parameters

The USSD Gateway bridges network-initiated and mobile-initiated USSD to an HTTP/JSON backend. Enable it with `ussd_gateway_enabled: true` (see Operational Mode Flags), then configure the `ussd_gateway` map. For the full callback protocol and session lifecycle, see the USSD Gateway Guide.

Parameter	Type	Default	Required	De
`ussd_gateway.routes`	List of Maps	`[]`	**Yes** (if enabled)	Shor callb rout Each `{pat` `<pre` `url:` `<cal` `URL>` **Lon mat pref** use shor as c fallb
`ussd_gateway.session_timeout_ms`	Integer	`180_000`	No	Max sess in m (3 m Sess term exce
`ussd_gateway.turn_timeout_ms`	Integer	`30_000`	No	Max (ms) a su repl mult dialo
`ussd_gateway.http_timeout_ms`	Integer	`5_000`	No	HTT requ time

Parameter	Type	Default	Required	De
				for c your appl
`ussd_gateway.max_text_length`	Integer	`182`	No	Max char USS strin bit). text trun warn logg

Routing: When a USSD code arrives, the gateway sorts configured routes by descending `pattern` length and selects the first whose `pattern` is a prefix of the dialled code — giving deterministic longest-prefix matching.

Parse error on line 3: ... B -->|pattern \"*100\"| C["POST to i ----------------------- ^ Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND', 'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND', 'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'STR'

Try again

Example:

```
config :omniss7,
  ussd_gateway_enabled: true,
  ussd_gateway: %{
    routes: [
      %{pattern: "*100", url: "http://balance-app:9000/ussd"},
      %{pattern: "*200", url: "http://topup-app:9000/ussd"},
      %{pattern: "*",    url: "http://default-app:9000/ussd"}
    ],
    session_timeout_ms: 180_000,
    turn_timeout_ms: 30_000,
    http_timeout_ms: 5_000,
    max_text_length: 182
  }
```

Runtime / Supervision Flags

These flags control how much of the supervision tree starts. They are useful for running OmniSS7 headless (as a library dependency) or for stripping the web tier in constrained deployments.

Parameter	Type	Default	Required	Desc
`start_web_interfaces`	Boolean	`true`	No	Start the contr `:api_ex` REST SS7 events log `false`, only the (`SS7.Events`) s overridden at b `ENABLE_WEB_IN` environment va or `false`/`0`).
`start_http_server`	Boolean	`true`	No	Start the Plug.C server on port `/api/*`, `/metr` See REST API S
`headless`	Boolean	`false`	No	Library-only mi supervision. Wh OmniSS7 starts codec/transact (MAP, CAP, USS dynamic super UI, no license c handler, no M3 Intended for er OmniSS7 codec dependency.
`vlr_cache_ttl_seconds`	Integer	`3600`	No	Time-to-live (se cached VLR loc delivery path (`lib/sms/smsc`

Example (headless library mode):

```
config :omniss7,
  headless: true
```

Example (no web tier, REST API only):

```
config :omniss7,
  start_web_interfaces: false,
  start_http_server: true
```

Infrastructure Parameters

Configuration for system infrastructure components including licensing, web interface, API server, and logging.

License Configuration

Parameter	Type	Default	Required
`license_client.license_server_api_urls`	List of Strings	-	**Yes**
`license_client.licensee`	String	-	**Yes**

Example:

```
config :license_client,
  license_server_api_urls: ["https://localhost:10443/api"],
  licensee: "Omnitouch Network Services Pty. Ltd."
```

Control Panel Web Interface

Parameter	Type	D
`control_panel.parent_application_readable_name`	String	`"OmniSS7 S`
`control_panel.use_additional_pages`	List of Tuples	`[]`
`control_panel.page_order`	List of Strings	`[]`
`ControlPanelWeb.Endpoint.url.host`	String	`"0.0.0.0"`
`ControlPanelWeb.Endpoint.https.port`	Integer	`8087`
`ControlPanelWeb.Endpoint.https.keyfile`	String	`"priv/cert/`
`ControlPanelWeb.Endpoint.https.certfile`	String	`"priv/cert/`

Example:

```
config :control_panel, ControlPanelWeb.Endpoint,
  url: [host: "0.0.0.0", path: "/"],
  https: [
    port: 8087,
    keyfile: "priv/cert/omnitouch.pem",
    certfile: "priv/cert/omnitouch.crt"
  ],
  parent_application_readable_name: "OmniSS7 Stack STP"

config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 Events"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "Peers"}
  ],
  page_order: ["/events", "/m3ua", "/application",
"/configuration"]
```

REST API Server

Parameter	Type	Default	Req
`start_http_server`	Boolean	`true`	No
`api_ex.api.port`	Integer	`8445`	No
`api_ex.api.listen_ip`	String	`"0.0.0.0"`	No
`api_ex.api.product_name`	String	`"OmniSS7"`	No
`api_ex.api.title`	String	`"API - OmniSS7"`	No
`api_ex.api.hostname`	String	`"localhost"`	No
`api_ex.api.enable_tls`	Boolean	`true`	No
`api_ex.api.tls_cert_path`	String	`"priv/cert/omnitouch.crt"`	No
`api_ex.api.tls_key_path`	String	`"priv/cert/omnitouch.pem"`	No

Example:

```
config :omniss7,
  start_http_server: true

config :api_ex,
  api: %{
    port: 8445,
    listen_ip: "0.0.0.0",
    product_name: "OmniSS7",
    title: "API - OmniSS7",
    hostname: "localhost",
    enable_tls: true,
    tls_cert_path: "priv/cert/omnitouch.crt",
    tls_key_path: "priv/cert/omnitouch.pem"
  }
```

Ports and transports at a glance:

Port	Transport	Server	Purpose
8080	HTTP (Plug.Cowboy)	`APIhandler`	MAP/CAP REST `/api`, `/metrics`, `/swagger` `/swagger.json`. Tog by `start_http_ser`
8087	HTTPS (Bandit/Phoenix)	`ControlPanelWeb.Endpoint`	Web control panel U Part of the web tier (`start_web_interf`
8445	HTTPS / TLS	`:api_ex` REST controllers	Status, routing CRU links, subscribers, CAMEL/SCTP/M3UA views. Part of the w tier.

API Endpoints:

- MAP/CAP REST API + Swagger: `http://[server-ip]:8080/...` (see API Guide)

- `:api_ex` REST controllers: `https://[server-ip]:8445/api/*`
- Web control panel: `https://[server-ip]:8087/`
- Prometheus metrics: `http://[server-ip]:8080/metrics`

API Backend SSL Verification

Outbound calls to backend HLR/SMSc APIs do **not** use per-target `*_verify_ssl` keys. Certificate verification is controlled by the generic `:api` map consumed by `SS7.Config.api_config/0` (`lib/ss7_config.ex`):

Parameter	Type	Default	Description
`api.base_url`	String	falls back to `smsc_api_base_url`	Base URL for the generic API client.
`api.timeout`	Integer	`5000`	Request timeout in milliseconds.
`api.verify_ssl`	Boolean	`false`	Whether to verify backend TLS certificates. **This is the single SSL-verify switch** — there is no `hlr_api_verify_ssl` or `smsc_api_verify_ssl`.

Example:

```
config :omniss7,
  api: %{
    base_url: "https://10.179.3.219:8443",
    timeout: 5000,
    verify_ssl: false
  }
```

Logging Configuration

Parameter	Type	Default	Requ
`logger.level`	Atom	`:debug`	No
`logger.backends`	List	`[:console]`	No
`logger.default_formatter.format`	String	-	No
`logger.default_formatter.metadata`	List	`[]`	No
`logger.default_formatter.truncate`	Atom/Integer	`:infinity`	No

Example:

```
config :logger,
  level: :debug,
  backends: [:console, SS7.Web.LoggerBackend]

config :logger, :default_formatter,
  format: "[$date] [$time] [$level] $message\n",
  metadata: [:error_code, :file],
  truncate: :infinity
```

Database Parameters

Configuration for Mnesia database persistence.

Parameter	Type	Default	Required	Description
`mnesia_storage_type`	Atom	`:disc_copies`	No	Mnesia storage type: `:disc_copies` or `:ram_copies`

Example:

```
config :omniss7,
  mnesia_storage_type: :disc_copies  # Production
  # mnesia_storage_type: :ram_copies  # Testing only
```

Storage Types:

- `:disc_copies` - Persistent disk storage (survives restarts) - **Recommended for production**
- `:ram_copies` - In-memory only (lost on restart) - For testing only

Mnesia Tables:

- `m3ua_peer` - M3UA peer connections
- `m3ua_route` - Point Code routes
- `m3ua_gt_route` - Global Title routes

Location: `Mnesia.{node_name}/` directory

Hardcoded Values

The following values are **hardcoded in the source code** and cannot be changed via configuration.

Timeouts

Value	Impact	Workaround
MAP request timeout: **10 seconds**	All MAP operations timeout after 10s	Modify source code
ISD timeout: **10 seconds**	Each ISD message times out after 10s	Modify source code

HTTP Server

Value	Impact	Workaround
HTTP IP: **0.0.0.0**	Metrics/Swagger server listens on all interfaces	Modify source code
HTTP Port: **8080**	Metrics/Swagger endpoint runs on port 8080	Modify source code

Registration Intervals

Value	Impact	Workaround
Frontend registration: **5 minutes**	SMSc registers with backend every 5 min	Modify source code

Web UI Auto-Refresh

Page	Interval
Routing Management	5 seconds
Active Subscribers	2 seconds

Configuration Examples

Minimal HLR Configuration

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  hlr_api_base_url: "https://10.180.2.140:8443",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112",

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

Minimal SMSc Configuration

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  smsc_api_base_url: "https://10.179.3.219:8443",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112",

  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

STP with Standalone Server

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,

  enable_gt_routing: true,
  mnesia_storage_type: :disc_copies,

  sctp_handler: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  },

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 10},
    local_port: 2906,
    remote_ip: {10, 179, 4, 11},
    remote_port: 2905,
    routing_context: 1
  }
```

Summary

Total Configuration Parameters: 85+

By Category:

- Operational Mode: 6 parameters
- HLR Mode: 17 parameters
- SMSc Mode: 8 parameters

- STP Mode: 5+ parameters (plus m3ua_peers, m3ua_routes, m3ua_gt_routes lists)
- CAMEL GW Mode: 14 parameters
- Global Title NAT: 2 parameters
- M3UA Connection: 8 parameters
- SCCP Subsystem Status: 1 parameter
- USSD Gateway: 5 parameters
- Runtime / Supervision: 4 parameters
- Infrastructure (License, Web, API, Logging): 23 parameters
- Database: 1 parameter

Required Parameters by Mode:

HLR Mode:

- `hlr_api_base_url`
- `hlr_service_center_gt_address`
- `smsc_service_center_gt_address`
- All `map_client_m3ua.*` parameters (8)

SMSc Mode:

- `smsc_api_base_url`
- `smsc_service_center_gt_address`
- `auto_flush_dest_smsc` (if auto-flush enabled)
- All `map_client_m3ua.*` parameters (8)

STP Mode:

- `sctp_handler.point_code` (if SCTP handler enabled)
- `sctp_handler.local_ip`
- `sctp_handler.local_port`

CAMEL GW Mode:

- `cgrates_url` (if CGrateS enabled)
- All `cap_client_m3ua.*` parameters (8)

Infrastructure:

- `license_client.license_server_api_urls`
- `license_client.licensee`

Related Documentation

- **HLR Guide** - HLR-specific configuration
- **SMSc Guide** - SMSc-specific configuration
- **STP Guide** - STP routing configuration
- **API Guide** - REST API reference
- **USSD Gateway Guide** - USSD Gateway configuration and HTTP callback protocol
- **Web UI Guide** - Web interface documentation

Global Title NAT Guide

Overview

Global Title Network Address Translation (GT NAT) is a feature that allows OmniSS7 to respond with different Global Title addresses based on the calling party's GT prefix, the called party's GT prefix, or a combination of both. This is essential when operating with multiple Global Titles and needing to ensure responses use the correct GT based on which network or peer is calling and/or which GT they called.

What's New (Enhanced GT NAT)

The GT NAT feature has been enhanced with powerful new capabilities:

New Features

1. **Called Party Prefix Matching**: Rules can now match on `called_prefix` in addition to `calling_prefix`
2. **Combined Matching**: Rules can match on both calling AND called prefixes simultaneously
3. **Weight-Based Prioritization**: Rules now use a `weight` field (lower = higher priority) instead of just prefix length
4. **Flexible Matching**: You can now create rules with:
   - Only calling prefix
   - Only called prefix
   - Both calling and called prefixes
   - Neither (wildcard/fallback rule)

New Rule Format

Required fields:

- `weight`: Integer priority (lower = higher priority)
- `response_gt`: The GT to respond with

Optional fields (at least one recommended for specific matching):

- `calling_prefix`: Match on calling party GT prefix
- `called_prefix`: Match on called party GT prefix

Example:

```
gt_nat_rules: [
  # Specific rule with both prefixes - highest priority
  %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

  # Specific rules - medium priority
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},
  %{called_prefix: "555", weight: 10, response_gt: "333333"},

  # Wildcard fallback - lowest priority
  %{weight: 100, response_gt: "999999"}
]
```

Use Cases

Multi-Network Operation

When you have multiple peer networks and each expects responses from a specific GT:

- **Network A** calls your GT `111111` and expects responses from `111111`
- **Network B** calls your GT `222222` and expects responses from `222222`

Without GT NAT, you would need separate instances or complex routing. With GT NAT, a single OmniSS7 instance can handle this intelligently.

Roaming Scenarios

When operating as an HLR or SMSc with roaming agreements:

- **Home network** subscribers use GT `555000`
- **Roaming partner 1** uses GT `555001`
- **Roaming partner 2** uses GT `555002`

GT NAT ensures each partner receives responses from the correct GT they're configured to route to.

Testing and Migration

During network migrations or testing:

- Gradually migrate traffic from old GT to new GT
- Maintain both GTs during transition period
- Route responses based on which GT the caller used

How It Works

Address Translation Flow

1. **Incoming Request**: OmniSS7 receives an SCCP message with:
    - Called Party GT: `55512341112` (your GT)
    - Calling Party GT: `877234567` (their GT)
2. **GT NAT Lookup**: System checks calling GT `877234567` against configured prefix rules
3. **Prefix Matching**: Finds longest matching prefix (e.g., `8772` matches `877234567`)
4. **Response GT Selection**: Uses `response_gt` from matched rule (e.g., `55512341112`)

5. **Response Sent**: SCCP response uses:
    - Called Party GT: `877234567` (reversed - their GT)
    - Calling Party GT: `55512341112` (NAT'd GT)

Affected Response Types

GT NAT applies to multiple layers of the SS7 stack:

SCCP Layer (All Responses)

- SCCP Called/Calling GT addresses in all response messages
- ISD (InsertSubscriberData) acknowledgments
- UpdateLocation responses
- Error responses

MAP Layer (Operation-Specific)

- **SRI-for-SM Responses**: `networkNode-Number` (SMSc GT address)
- **UpdateLocation**: `hlr-Number` in responses
- **InsertSubscriberData**: HLR GT in ISD messages

Configuration

Basic Configuration

Add to `config/runtime.exs`:

```
config :omniss7,
  # Enable GT NAT
  gt_nat_enabled: true,

  # Define GT NAT rules
  gt_nat_rules: [
    # Rule 1: Calls from prefix "8772" get response from
"55512341112"
    %{calling_prefix: "8772", response_gt: "55512341112"},

    # Rule 2: Calls from prefix "8773" get response from
"55512341111"
    %{calling_prefix: "8773", response_gt: "55512341111"},

    # Default rule (empty prefix matches everything)
    %{calling_prefix: "", response_gt: "55512311555"}
  ]
```

Configuration Parameters

For complete configuration reference, see Global Title NAT Parameters in Configuration Reference.

Parameter	Type	Required	Description
`gt_nat_enabled`	Boolean	Yes	Enable/disable GT NAT feature
`gt_nat_rules`	List of Maps	Yes (if enabled)	List of prefix matching rules

Rule Format

Each rule is a map with the following keys:

```
%{
  calling_prefix: "8772",      # (Optional) Prefix to match
against calling GT
  called_prefix: "555",        # (Optional) Prefix to match
against called GT
  weight: 10,                  # (Required) Priority value (lower
= higher priority)
  response_gt: "55512341112"   # (Required) GT to use in responses
}
```

Rule Fields:

- **`calling_prefix`** (Optional): String prefix to match against incoming calling GT
  - Matching is done by `String.starts_with?/2`
  - Empty string `""` or `nil` acts as wildcard (matches any calling GT)
  - Can be omitted to match any calling GT
- **`called_prefix`** (Optional): String prefix to match against incoming called GT
  - Matching is done by `String.starts_with?/2`
  - Empty string `""` or `nil` acts as wildcard (matches any called GT)
  - Can be omitted to match any called GT
- **`weight`** (Required): Integer priority value
  - Lower weight = higher priority (processed first)
  - Must be >= 0
  - Used as primary sorting criterion for matching rules
- **`response_gt`** (Required): The Global Title address to use in responses
  - Must be a valid E.164 number string
  - Should match one of your configured GTs

At least one of `calling_prefix` or `called_prefix` should be specified for specific routing. Both can be omitted for a wildcard/fallback rule.

Rule Matching Logic

Rules are evaluated by **weight first (ascending), then by combined prefix specificity**:

Matching Algorithm:

1. Filter rules where all specified prefixes match
   - If `calling_prefix` is set, it must match the calling GT
   - If `called_prefix` is set, it must match the called GT
   - If both are set, both must match
   - If neither is set, rule acts as a wildcard
2. Sort matching rules by:
   - **Primary**: Weight (ascending - lower values first)
   - **Secondary**: Combined prefix length (descending - longer = more specific)
3. Return the first matching rule

Examples:

```
# Example rules
gt_nat_rules: [
  # Weight 1: Highest priority - matches both prefixes
  %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

  # Weight 10: Medium priority - specific rules
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  #
Calling only
  %{called_prefix: "555", weight: 10, response_gt: "333333"},    #
Called only

  # Weight 100: Lowest priority - wildcard fallback
  %{weight: 100, response_gt: "444444"}  # Matches everything
]

# Matching examples:
# Calling: "877234567", Called: "555123" -> "111111" (weight 1,
both match)
# Calling: "877234567", Called: "999999" -> "222222" (weight 10,
calling only)
# Calling: "999999999", Called: "555123" -> "333333" (weight 10,
called only)
# Calling: "999999999", Called: "888888" -> "444444" (weight 100,
wildcard)
```

Examples

Example 1: Two Network Partners

Scenario: You operate an SMSc with two network partners. Each expects responses from a different GT.

```
config :omniss7,
  gt_nat_enabled: true,

  # Default SMSc GT (used when GT NAT is disabled or no rule
matches)
  smsc_service_center_gt_address: "5551000",

  # GT NAT rules for partners
  gt_nat_rules: [
    # Partner A (prefix 4412) expects responses from GT 5551001
    %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

    # Partner B (prefix 4413) expects responses from GT 5551002
    %{calling_prefix: "4413", weight: 10, response_gt: "5551002"},

    # Default: use standard SMSc GT (wildcard fallback)
    %{weight: 100, response_gt: "5551000"}
  ]
```

Traffic Flow:

```
Incoming SRI-for-SM from 44121234567:
  Called GT: 5551001 (your GT that Partner A uses)
  Calling GT: 44121234567 (Partner A's GT)

GT NAT Lookup:
  "44121234567" matches prefix "4412"
  Selected response_gt: "5551001"

Response SRI-for-SM to 44121234567:
  Called GT: 44121234567 (reversed)
  Calling GT: 5551001 (NAT'd)
  networkNode-Number: 5551001 (in MAP response)
```

Example 2: HLR with Regional GTs

Scenario: National HLR with different GTs per region.

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "555000",  # Default HLR GT

  gt_nat_rules: [
    # Northern region VLRs (prefix 5551)
    %{calling_prefix: "5551", weight: 10, response_gt: "555100"},

    # Southern region VLRs (prefix 5552)
    %{calling_prefix: "5552", weight: 10, response_gt: "555200"},

    # Western region VLRs (prefix 5553)
    %{calling_prefix: "5553", weight: 10, response_gt: "555300"},

    # Default for other regions (wildcard)
    %{weight: 100, response_gt: "555000"}
  ]
```

Example 3: Migration Scenario

Scenario: Migrating from old GT to new GT gradually.

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "123456789",  # Old GT (default)

  gt_nat_rules: [
    # Migrated networks (already updated their configs)
    %{calling_prefix: "555", weight: 10, response_gt:
"987654321"},  # New GT
    %{calling_prefix: "666", weight: 10, response_gt:
"987654321"},  # New GT

    # Everyone else still uses old GT (wildcard)
    %{weight: 100, response_gt: "123456789"}  # Old GT
  ]
```

Example 4: Called Party Prefix Matching (NEW)

Scenario: You have multiple GTs for different services, and want to respond with the correct GT based on which GT was called.

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # When they call your SMS GT (5551xxx), respond with that GT
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # When they call your Voice GT (5552xxx), respond with that GT
    %{called_prefix: "5552", weight: 10, response_gt: "555200"},

    # When they call your Data GT (5553xxx), respond with that GT
    %{called_prefix: "5553", weight: 10, response_gt: "555300"},

    # Default fallback
    %{weight: 100, response_gt: "555000"}
  ]
```

Traffic Flow:

```
Incoming request to Called GT: 555100 (your SMS GT)
Calling GT: 441234567 (any caller)

GT NAT Lookup:
  Called GT "555100" matches prefix "5551"
  Selected response_gt: "555100"

Response uses Calling GT: 555100 (matches what they called)
```

Example 5: Combined Calling + Called Prefix Matching (ADVANCED)

Scenario: Different partners call different GTs, and you want fine-grained control.

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # Partner A calling your SMS GT - highest priority (weight 1)
    %{calling_prefix: "4412", called_prefix: "5551", weight: 1,
response_gt: "555101"},

    # Partner B calling your SMS GT - highest priority (weight 1)
    %{calling_prefix: "4413", called_prefix: "5551", weight: 1,
response_gt: "555102"},

    # Anyone calling your SMS GT - medium priority (weight 10)
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # Partner A calling any GT - medium priority (weight 10)
    %{calling_prefix: "4412", weight: 10, response_gt: "555200"},

    # Default fallback - low priority (weight 100)
    %{weight: 100, response_gt: "555000"}
  ]
```

Matching Examples:

```
# Partner A calls SMS GT
Calling: "441234567", Called: "555100"
→ Matches weight 1 rule (both prefixes) → "555101"

# Partner A calls Voice GT
Calling: "441234567", Called: "555200"
→ Matches weight 10 rule (calling only) → "555200"

# Unknown caller calls SMS GT
Calling: "999999999", Called: "555100"
→ Matches weight 10 rule (called only) → "555100"

# Unknown caller calls Voice GT
Calling: "999999999", Called: "555200"
→ Matches weight 100 wildcard → "555000"
```

Operational Modes

GT NAT works across all OmniSS7 operational modes:

HLR Mode

GT NAT affects:

- UpdateLocation responses (HLR GT in response)
- InsertSubscriberData messages (HLR GT as calling party)
- SendAuthenticationInfo responses
- Cancel Location responses

For more information on HLR operations, see the HLR Configuration Guide.

Configuration:

```
config :omniss7,
  hlr_mode_enabled: true,
  hlr_service_center_gt_address: "5551234567",  # Default HLR GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "331", weight: 10, response_gt:
"5551234568"},  # France
    %{calling_prefix: "44", weight: 10, response_gt:
"5551234569"},   # UK
    %{weight: 100, response_gt: "5551234567"}  # Default wildcard
  ]
```

SMSc Mode

GT NAT affects:

- SRI-for-SM responses (`networkNode-Number` field) - see SRI-for-SM Details
- MT-ForwardSM acknowledgments

For more information on SMSc operations, see the SMSc Configuration Guide.

Configuration:

```
config :omniss7,
  smsc_mode_enabled: true,
  smsc_service_center_gt_address: "5559999",  # Default SMSc GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "1", weight: 10, response_gt: "5559991"},
# North America
    %{calling_prefix: "44", weight: 10, response_gt: "5559992"},
# UK
    %{calling_prefix: "86", weight: 10, response_gt: "5559993"},
# China
    %{weight: 100, response_gt: "5559999"}  # Default wildcard
  ]
```

CAMEL Gateway Mode

GT NAT affects:

- All SCCP-level responses (gsmSCF GT as Calling Party)
- CAMEL/CAP operation responses (InitialDP, EventReportBCSM, etc.)
- RequestReportBCSMEvent acknowledgments
- ApplyCharging responses
- Continue responses

Configuration:

```
config :omniss7,
  camelgw_mode_enabled: true,
  camel_gsmscf_gt_address: "55512341112",  # Default gsmSCF GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "555", weight: 10, response_gt:
"55512341111"},  # Network A
    %{calling_prefix: "666", weight: 10, response_gt:
"55512311555"},  # Network B
    %{weight: 100, response_gt: "55512341112"}  # Default wildcard
  ]
```

Use Case: When operating as a gsmSCF (Service Control Function) for multiple networks, each network's gsmSSF may expect responses from a specific gsmSCF GT. GT NAT ensures the correct GT is used based on which gsmSSF is calling.

Logging and Debugging

Enable GT NAT Logging

GT NAT includes automatic logging of all translations:

```
# In logs, you'll see:
[info] GT NAT [SRI-for-SM response]: Calling GT 877234567 ->
Response GT 55512341112
[info] GT NAT [UpdateLocation ISD]: Calling GT 331234567 ->
Response GT 55512341111
[info] GT NAT [MAP BEGIN response]: Calling GT 441234567 ->
Response GT 55512311555
```

The context field shows where the NAT was applied:

- `"SRI-for-SM response"` - In SRI-for-SM handler
- `"UpdateLocation ISD"` - In InsertSubscriberData messages
- `"UpdateLocation END"` - In UpdateLocation END response

- `"MAP BEGIN response"` - Generic MAP BEGIN responses
- `"ISD ACK"` - ISD acknowledgment
- `"HLR error response"` - Error response from HLR
- `"CAMEL response"` - CAMEL/CAP operation responses (gsmSCF)

Validation

The system validates GT NAT configuration at startup:

```
# Check GT NAT config
iex> GtNat.validate_config()
{:ok, [
  %{calling_prefix: "8772", weight: 10, response_gt:
"55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt:
"55512341111"}
]}

# Check if enabled
iex> GtNat.enabled?()
true

# Get all rules
iex> GtNat.get_rules()
[
  %{calling_prefix: "8772", weight: 10, response_gt:
"55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt:
"55512341111"}
]
```

Testing GT NAT

Test GT NAT logic programmatically:

```
# Test translation with calling GT only (called_gt is nil)
iex> GtNat.translate_response_gt("877234567", nil, "default_gt")
"55512341112"

# Test translation with both calling and called GT
iex> GtNat.translate_response_gt("877234567", "555123",
"default_gt")
"55512341112"

# Test with logging (nil called GT)
iex> GtNat.translate_response_gt_with_logging("877234567", nil,
"default_gt", "test")
# Logs: GT NAT [test]: Calling GT 877234567 -> Response GT
55512341112
"55512341112"

# Test with logging (both GTs)
iex> GtNat.translate_response_gt_with_logging("877234567",
"555123", "default_gt", "test")
# Logs: GT NAT [test]: Calling GT 877234567, Called GT 555123 ->
Response GT 55512341112
"55512341112"

# Test no match (returns default)
iex> GtNat.translate_response_gt("999999999", "888888",
"default_gt")
"default_gt"
```

Troubleshooting

Issue: GT NAT Not Working

Check 1: Is it enabled?

```
iex> Application.get_env(:omniss7, :gt_nat_enabled)
true  # Should be true
```

Check 2: Are rules configured?

```
iex> Application.get_env(:omniss7, :gt_nat_rules)
[%{calling_prefix: "8772", response_gt: "55512341112"}, ...]  #
Should return list
```

Check 3: Check logs Search for "GT NAT" in logs to see if translations are happening.

Issue: Wrong GT in Responses

Symptom: Responses use unexpected GT address

Cause: Rule prefix matching might be too broad or default rule is catching traffic

Solution: Review rule weights and prefixes:

```
# BAD: Wildcard with low weight (catches everything first)
gt_nat_rules: [
  %{weight: 1, response_gt: "111111"},               # This
matches everything first!
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"}  #
Never reached
]

# GOOD: Specific rules with lower weight, wildcard with higher
weight
gt_nat_rules: [
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  #
Specific, low weight
  %{weight: 100, response_gt: "111111"}  # Wildcard, high weight
(fallback)
]
```

Issue: GT NAT Not Applied to Specific Message Type

Symptom: Some responses use NAT'd GT, others don't

Current Coverage:

- □ SCCP Calling GT (all responses)
- □ SRI-for-SM responses (networkNode-Number)
- □ UpdateLocation ISD messages (HLR GT)
- □ UpdateLocation END responses
- □ ISD acknowledgments
- □ MAP BEGIN responses

If a specific message type isn't using GT NAT, it may not be implemented yet. Check the source code or contact support.

Performance Considerations

Lookup Performance

GT NAT uses simple prefix matching with O(n) complexity where n is the number of rules.

Performance tips:

- Keep rule count under 100 for best performance
- Use specific prefixes to reduce rule count
- Default rule (empty prefix) should be last

Benchmark (typical system):

- 10 rules: < 1µs per lookup
- 50 rules: < 5µs per lookup
- 100 rules: < 10µs per lookup

Memory Usage

Each rule requires ~100 bytes of memory:

- 10 rules ≈ 1 KB

- 100 rules ≈ 10 KB

Best Practices

1. Always Include a Wildcard Fallback Rule

```
gt_nat_rules: [
  %{calling_prefix: "8772", weight: 10, response_gt: "111111"},
  %{calling_prefix: "8773", weight: 10, response_gt: "222222"},
  %{weight: 100, response_gt: "default_gt"}  # Always have a
wildcard with high weight
]
```

2. Use Meaningful Prefixes and Weights

```
# GOOD: Clear, specific prefixes with appropriate weights
%{calling_prefix: "331", weight: 10, response_gt: "..."}  # France
%{calling_prefix: "44", weight: 10, response_gt: "..."}   # UK

# BAD: Overly broad prefixes or confusing weights
%{calling_prefix: "3", weight: 5, response_gt: "..."}     # Too
many countries
%{calling_prefix: "331", weight: 100, response_gt: "..."}  #
Weight should be lower for specific rules
```

3. Document Your Rules

```
gt_nat_rules: [
  # Partner XYZ - UK network (GT range: 4412xxxxxxx)
  # Weight 10: Standard partner priority
  %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

  # Partner ABC - France network (GT range: 33123xxxxxx)
  # Weight 10: Standard partner priority
  %{calling_prefix: "33123", weight: 10, response_gt: "5551002"}
]
```

4. Test Before Deployment

```
# Test in iex before deploying
iex> GtNat.translate_response_gt("44121234567", nil, "default")
"5551001"  # Expected result

# Test with called GT
iex> GtNat.translate_response_gt("44121234567", "555123",
"default")
"5551001"  # Expected result
```

5. Monitor Logs

Enable INFO level logging to see all GT NAT translations in production.

Integration with Other Features

STP Mode

GT NAT works independently of STP routing. STP routes based on point codes and destination GTs, while GT NAT handles response addressing.

For more information on STP routing, see the STP Configuration Guide.

CAMEL Integration

GT NAT is **fully integrated** with CAMEL/CAP operations:

SCCP Layer:

- Calling Party GT in all CAMEL responses
- Automatically applied based on incoming gsmSSF GT

Configuration:

- `camel_gsmscf_gt_address` - Default gsmSCF GT (optional)
- If not configured, uses the Called Party GT from incoming request

- GT NAT rules override the default based on calling party prefix

Example:

```
# When gsmSSF 555123456 calls your gsmSCF
# Incoming: Called=55512341112, Calling=555123456
# GT NAT lookup: "555" -> response_gt="55512341111"
# Response: Called=555123456, Calling=55512341111
```

Load Balancing

GT NAT can be combined with M3UA load balancing for advanced traffic management.

Migration Guide

Enabling GT NAT on Existing System

1. **Prepare Configuration**

   ```
   # Add to runtime.exs (keep disabled initially)
   config :omniss7,
     gt_nat_enabled: false,  # Start disabled
     gt_nat_rules: [
       # Your rules here with weights
       %{calling_prefix: "877", weight: 10, response_gt:
   "111111"},
       %{weight: 100, response_gt: "999999"}  # Wildcard fallback
     ]
   ```

2. **Test Configuration**

```
# Validate config compiles
mix compile

# Test in iex
iex -S mix
iex> GtNat.validate_config()
```

3. **Enable in Staging**

```
gt_nat_enabled: true  # Change to true
```

4. **Monitor Logs**

```
tail -f log/omniss7.log | grep "GT NAT"
```

5. **Deploy to Production**
    - Deploy during maintenance window
    - Monitor first 24 hours closely
    - Have rollback plan ready (set `gt_nat_enabled: false`)

Support

For issues or questions:

- Check logs for "GT NAT" messages
- Validate config with `GtNat.validate_config()`
- Review this guide's troubleshooting section
- Contact OmniSS7 support with log excerpts

See Also

- HLR Guide - HLR mode configuration
- SMSC Guide - SMSc mode configuration

- STP Guide - STP routing configuration
- Configuration Reference - Complete config reference

HLR Configuration Guide

← Back to Main Documentation

This guide provides configuration for using OmniSS7 as a **Home Location Register (HLR/HSS)** with **OmniHSS** as the backend subscriber database.

OmniHSS Integration

OmniSS7 HLR mode functions as an SS7 signaling frontend that interfaces with **OmniHSS**, a full-featured Home Subscriber Server (HSS) backend. This architecture separates concerns:

- **OmniSS7 (HLR Frontend)**: Handles all SS7/MAP protocol signaling, SCCP routing, and network communication
- **OmniHSS (HSS Backend)**: Manages subscriber data, authentication, provisioning, and advanced features

Why OmniHSS?

OmniHSS provides carrier-grade subscriber management with features including:

- **Multi-IMSI Support**: Each subscriber can have multiple IMSIs associated with a single MSISDN for international roaming, network switching, and eSIM provisioning
- **Flexible Authentication**: Support for both Milenage (3G/4G/5G) and COMP128 (2G) authentication algorithms
- **Circuit & Packet Session Tracking**: Independent tracking of CS (circuit-switched) and PS (packet-switched) network registrations
- **Advanced Provisioning**: Customizable service profiles, supplementary services, and CAMEL subscription data

- **API-First Design**: RESTful HTTP API for integration with billing, CRM, and provisioning systems
- **Real-time Updates**: Location tracking, session management, and authentication vector generation

All subscriber data, authentication credentials, and service configurations are stored and managed in OmniHSS. OmniSS7 queries OmniHSS via HTTPS API calls to respond to MAP operations like UpdateLocation, SendAuthenticationInfo, and SendRoutingInfo.

Important: OmniSS7 HLR mode is a **signaling frontend only**. All subscriber management logic, authentication algorithms, provisioning rules, and database operations are handled by OmniHSS. This guide covers the SS7/MAP protocol configuration in OmniSS7. For information about subscriber provisioning, authentication configuration, service profiles, and administrative operations, **refer to the OmniHSS documentation**.

Multi-IMSI Support

OmniHSS natively supports Multi-IMSI configurations, allowing a single subscriber (identified by MSISDN) to have multiple IMSIs. This enables:

- **International Roaming Profiles**: Different IMSIs for different regions to reduce roaming costs
- **eSIM Multi-Profile**: Multiple network profiles on a single eSIM-capable device
- **Network Switching**: Seamless switching between networks without changing MSISDN
- **Dual SIM Coordination**: Coordination across multiple physical or virtual SIMs
- **Testing & Development**: Multiple test IMSIs pointing to the same subscriber

How it works:

- Each IMSI has its own authentication credentials (Ki, OPc, algorithm)
- Each IMSI can have independent circuit and packet session registrations

- Subscriber services and profiles can be shared or customized per-IMSI
- OmniSS7 queries OmniHSS by IMSI, and OmniHSS returns the appropriate subscriber data
- Billing systems can track usage per-IMSI while associating all IMSIs to a single account

Example Multi-IMSI scenario:

```
Subscriber MSISDN: +1-555-123-4567
├─ IMSI 1: 310260123456789 (US Home Network - Milenage auth)
├─ IMSI 2: 208011234567890 (France Roaming Profile - Milenage
auth)
└─ IMSI 3: 440201234567891 (UK Roaming Profile - COMP128 auth)
```

All three IMSIs can be used independently for network registration, but they all belong to the same subscriber account. OmniHSS manages the IMSI-to-subscriber mapping and ensures proper authentication and provisioning for each IMSI.

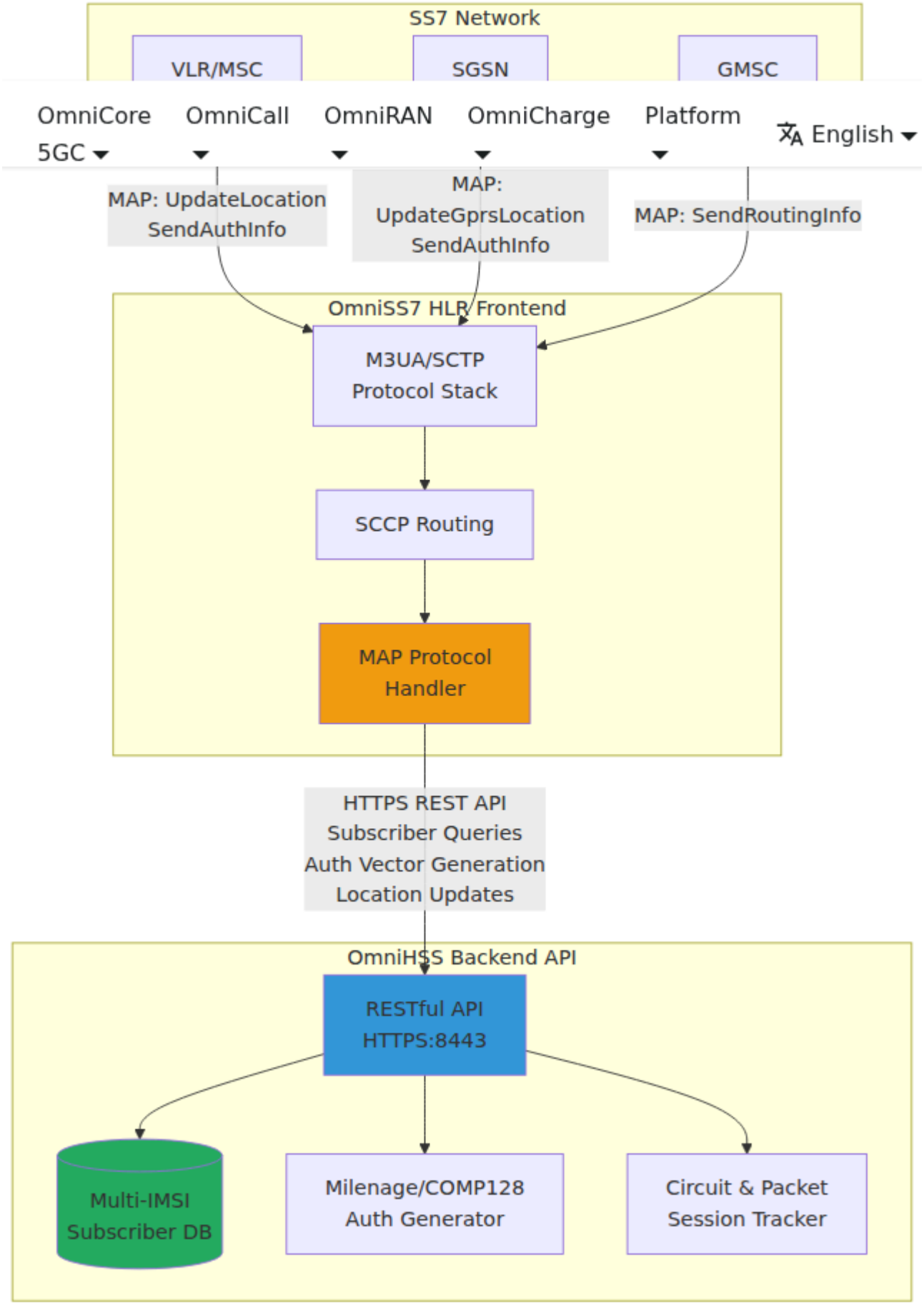
Table of Contents

What is HLR Mode?

HLR Mode enables OmniSS7 to function as a Home Location Register for:

- **Subscriber Management**: Store and manage subscriber data
- **Authentication**: Generate authentication vectors for network access
- **Location Tracking**: Process location updates from VLRs
- **Routing Information**: Provide routing info for calls and SMS

HLR Architecture


SS7 Network
VLR/MSC
SGSN
GMSC
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
English
MAP: UpdateLocation SendAuthInfo
MAP: UpdateGprsLocation SendAuthInfo
MAP: SendRoutingInfo
OmniSS7 HLR Frontend
M3UA/SCTP Protocol Stack
SCCP Routing
MAP Protocol Handler
HTTPS REST API
Subscriber Queries
Auth Vector Generation
Location Updates
OmniHSS Backend API
RESTful API HTTPS:8443
Multi-IMSI Subscriber DB
Milenage/COMP128 Auth Generator
Circuit & Packet Session Tracker

Enabling HLR Mode

OmniSS7 can operate in different modes (STP, HLR, SMSc). The mode is selected through a set of mode flags and the associated configuration in **`config/config.exs`**, the application's default configuration file.

Switching to HLR Mode

To run OmniSS7 as an HLR, set the mode flags so that HLR features are enabled and SMSc features are disabled, then add the HLR-specific configuration:

1. **Open** `config/config.exs`
2. **Set** the mode flags under `config :omniss7`:
   - `map_client_enabled: true` — required to send/receive MAP operations
   - `hlr_mode_enabled: true` — enables HLR-specific handling
   - `smsc_mode_enabled: false` — disables SMSc features
3. **Add** the HLR configuration parameters (API endpoint, GT addresses, ISD/CAMEL settings, M3UA connection) shown below
4. **Customize** the parameters for your deployment
5. **Restart** the application for the changes to take effect

> **Note**: `config/runtime.exs` is a minimal stub used for automated testing only — it does **not** contain operational mode definitions. All mode and HLR configuration lives in `config/config.exs` (with environment-specific overrides in files such as `config/test.exs`). Production deployments supply their own `config/config.exs`.

HLR Mode Configuration

The complete HLR configuration (in `config/config.exs`) looks like this:

```
config :omniss7,
  # Mode flags - Enable HLR features only
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  # OmniHSS Backend API Configuration
  hlr_api_base_url: "https://10.180.2.140:8443",

  # HLR Service Center GT Address for SMS operations
  hlr_service_center_gt_address: "1234567890",

  # MSISDN ↔ IMSI Mapping Configuration
  # See: MSISDN ↔ IMSI Mapping section for details
  hlr_imsi_plmn_prefix: "50557",
  hlr_msisdn_country_code: "61",
  hlr_msisdn_nsn_offset: 0,
  hlr_msisdn_nsn_length: 9,

  # InsertSubscriberData Configuration
  # Network Access Mode: :packetAndCircuit, :packetOnly, or
:circuitOnly
  isd_network_access_mode: :packetAndCircuit,

  # Send ISD #2 (Supplementary Services data)
  isd_send_ss_data: true,

  # Send ISD #3 (Call Barring data)
  isd_send_call_barring: true,

  # CAMEL Configuration (for SendRoutingInfo responses)
  # Service Key for CAMEL service initiation
  camel_service_key: 11_110,

  # CAMEL Trigger Detection Point
  # Options: :termAttemptAuthorized, :tBusy, :tNoAnswer, :tAnswer
  camel_trigger_detection_point: :termAttemptAuthorized,

  # Home VLR Prefixes
  # List of VLR address prefixes that are considered "home"
network
  # If subscriber's VLR starts with one of these prefixes, use
standard SRI response
```

```
  # Otherwise, subscriber is roaming and we need to send PRN to
get MSRN
  # Defaults to [] (no prefixes = all VLRs treated as roaming) if
omitted
  home_vlr_prefixes: ["555123"],

  # M3UA Connection Configuration
  # Connect as ASP for receiving MAP operations (UpdateLocation,
SendAuthInfo, etc.)
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # Local endpoint (HLR system)
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    # Remote STP endpoint
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

Configuration Parameters to Customize

For a complete reference of all configuration parameters, see the Configuration Reference.

Parameter	Type	Default
`hlr_api_base_url`	String	*Required*
`hlr_service_center_gt_address`	String	*Required*
`smsc_service_center_gt_address`	String	*Required*
`hlr_smsc_alert_gts`	List	`[]`
`hlr_alert_location_expiry_seconds`	Integer	`172800`
`hlr_imsi_plmn_prefix`	String	`"50557"`
`hlr_msisdn_country_code`	String	`"61"`

Parameter	Type	Default
`hlr_msisdn_nsn_offset`	Integer	`0`
`hlr_msisdn_nsn_length`	Integer	`9`
`isd_network_access_mode`	Atom	`:packetAndCircuit`
`isd_send_ss_data`	Boolean	`true`
`isd_send_call_barring`	Boolean	`true`
`camel_service_key`	Integer	`11_110`
`camel_trigger_detection_point`	Atom	`:termAttemptAuthorized`
`home_vlr_prefixes`	List	`[]`

Parameter	Type	Default
`local_ip`	Tuple	*Required*
`local_port`	Integer	`2905`
`remote_ip`	Tuple	*Required*
`remote_port`	Integer	`2905`
`routing_context`	Integer	`1`

What Happens When HLR Mode is Enabled

When `hlr_mode_enabled: true`, the web UI will show:

- □ **SS7 Events** - Event logging
- □ **SS7 Client** - MAP operation testing
- □ **Peers** - Connection status (M3UA/SCTP peers)
- □ **HLR Links** - HLR API status + subscriber management ← *HLR-specific*
- □ **Resources** - System monitoring
- □ **Configuration** - Config viewer

The **Routing**, **Routing Test**, and **SMSc Links** tabs will be hidden.

Important Notes

- **Required Configuration**: The `hlr_service_center_gt_address` parameter is **mandatory**. The application will fail to start if it is not configured.
- **OmniHSS Backend**: The OmniHSS API backend must be accessible at the configured `hlr_api_base_url`

- **API Request Timeout**: All OmniHSS API requests have a **hardcoded 5-second timeout**
- **MAP Request Timeout**: All MAP requests (SRI, UpdateLocation, SendAuthInfo, etc.) have a **hardcoded 10-second timeout**
- **ISD Timeout**: Each InsertSubscriberData (ISD) message in an UpdateLocation sequence has a **hardcoded 10-second timeout**
- M3UA connection to STP is required for receiving MAP operations
- After changing modes, you must restart the application for changes to take effect
- **Web UI**: See the Web UI Guide for information on using the web interface
- **API Access**: See the API Guide for REST API documentation and Swagger UI access

---

Subscriber Database

OmniHSS manages all subscriber data including identities, authentication credentials, service profiles, and location information. OmniSS7 retrieves this data via RESTful API calls.

OmniHSS Subscriber Model

OmniHSS stores comprehensive subscriber information:

- **Multiple IMSIs per subscriber**: Support for Multi-IMSI configurations (eSIM, roaming profiles, network switching)
- **Authentication credentials**: Ki, OPc, and algorithm selection (Milenage or COMP128)
- **Service profiles**: Subscriber category, allowed services, QoS parameters
- **Location tracking**: Current VLR/MSC (circuit session) and SGSN/GGSN (packet session) independent tracking
- **CAMEL subscription data**: Service keys, trigger points, and gsmSCF addresses
- **Supplementary services**: Call forwarding, barring, waiting, CLIP/CLIR configurations

- **Administrative state**: Enabled/disabled, service restrictions, expiration dates

---

Authentication Vectors

Generate Auth Vectors

OmniHSS generates authentication vectors using the Milenage or COMP128 algorithms based on each subscriber's configured authentication method. When OmniSS7 receives **sendAuthenticationInfo** MAP requests:

1. OmniSS7 extracts the IMSI from the MAP request
2. OmniSS7 calls the OmniHSS API to generate authentication vectors
3. OmniHSS retrieves the subscriber's Ki and OPc credentials
4. OmniHSS generates the requested number of vectors (RAND, XRES, CK, IK, AUTN)
5. OmniSS7 encodes the vectors into MAP format and returns them to the requesting VLR/SGSN

OmniHSS API Integration

OmniSS7 communicates with OmniHSS via HTTPS REST API to retrieve subscriber information, update location data, and generate authentication vectors:

```
config :omniss7,
  hlr_api_base_url: "https://omnihss-server:8443"
```

When OmniSS7 receives MAP operations from the SS7 network, it queries OmniHSS to:

- **Retrieve subscriber data** by IMSI or MSISDN
- **Generate authentication vectors** using stored Ki/OPc credentials

- **Update circuit session location** when subscribers perform UpdateLocation
- **Check subscriber status** and service entitlements

Location Updates

Update Location Processing

When receiving **updateLocation** MAP requests, OmniSS7 coordinates with OmniHSS to register the subscriber at a new VLR:

1. **Extract location info** from UpdateLocation request (IMSI, new VLR GT, new MSC GT)
2. **Query OmniHSS** to verify subscriber exists and is enabled
3. **Update circuit session** in OmniHSS with new VLR/MSC location
4. **Send InsertSubscriberData (ISD)** messages to provision the subscriber at the new VLR
5. **Return UpdateLocation response** to VLR (includes HLR GT from `hlr_service_center_gt_address`)
6. **Send alertServiceCenter** to configured SMSc GTs (if `hlr_smsc_alert_gts` is populated)

Note: The `hlr_service_center_gt_address` configuration parameter specifies the HLR's Global Title that is returned in UpdateLocation responses. This allows the VLR/MSC to identify and route messages back to this HLR.

Alert Service Center Integration

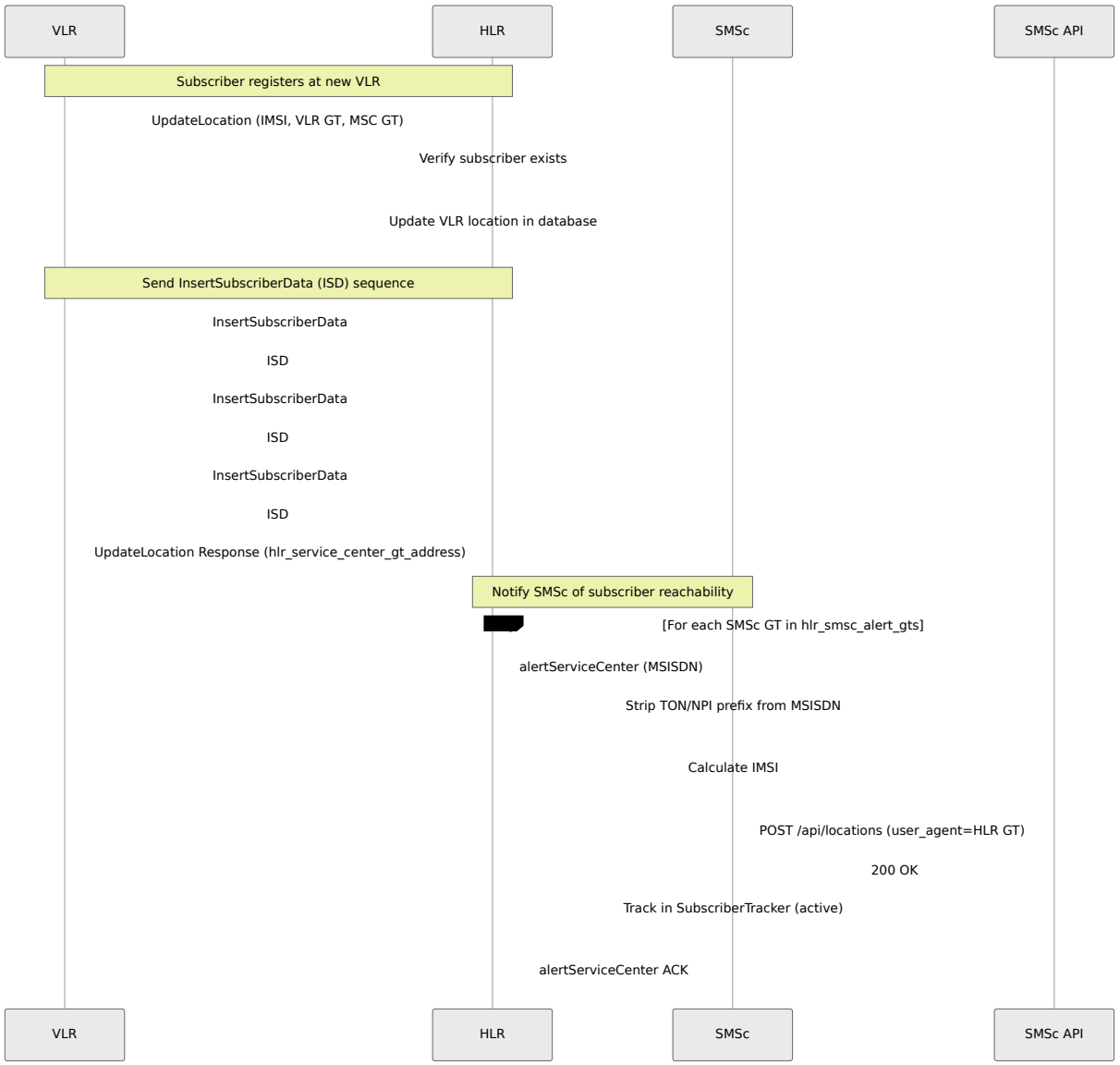
After a successful UpdateLocation, the HLR can automatically notify SMSc systems that a subscriber is now reachable by sending **alertServiceCenter** (MAP opcode 64) messages. For information on how the SMSc handles these alerts, see Alert Service Center Handling in SMSc Guide.

Configuration

Configure the list of SMSc Global Titles to notify:

```
config :omniss7,
  # List of SMSc GTs to send alertServiceCenter after
UpdateLocation
  hlr_smsc_alert_gts: [
    "15559876543",
    "15559876544"
  ],

  # Location expiry time when SMSc receives alertServiceCenter
(default: 48 hours)
  hlr_alert_location_expiry_seconds: 172800
```

Flow Diagram



Behavior

When a subscriber performs UpdateLocation:

1. HLR sends alertServiceCenter to **each** SMSc GT in the `hlr_smsc_alert_gts` list
2. Message includes the subscriber's MSISDN
3. HLR uses `hlr_service_center_gt_address` as the calling party GT
4. SCCP addressing: calling SSN=6 (HLR), called SSN=8 (SMSc)

The SMSc receives the alert and:

- **Strips TON/NPI prefix** from MSISDN (e.g., "19123123213" → "123123213")
- Marks the subscriber as reachable in its location database (via POST to /api/locations)
- **Sets `user_agent` field** to the HLR GT when calling the API (for tracking which HLR sent the alert)
- Sets location expiry time based on `hlr_alert_location_expiry_seconds`
- Tracks the subscriber in the SMSc Subscriber Tracker for monitoring

Testing

Use the **Active Subscribers** page in the Web UI to manually send alertServiceCenter messages for testing:

1. Navigate to the "Active Subscribers" tab
2. Find the "Test Alert Service Center" section
3. Enter MSISDN, SMSc GT, and HLR GT (defaults are pre-populated from config)
   - SMSc GT defaults to first entry in `hlr_smsc_alert_gts`
   - HLR GT defaults to `hlr_service_center_gt_address`
4. Click "Send alertServiceCenter"

This is useful for testing SMSc alert handling without requiring a full UpdateLocation flow. The form uses `phx-blur` validation to avoid showing errors while typing.

InsertSubscriberData (ISD) Configuration

After a successful UpdateLocation, the HLR sends subscriber provisioning data to the VLR using **InsertSubscriberData** (ISD) messages. The ISD configuration allows you to customize what data is sent and how.

For configuration parameter reference, see ISD Configuration in Configuration Reference.

ISD Sequence

The HLR can send up to 3 sequential ISD messages:

1. **ISD #1** (Always sent) - Basic subscriber data:
    - IMSI
    - MSISDN
    - Subscriber category
    - Subscriber status (serviceGranted)
    - Bearer service list
    - Teleservice list
    - Network access mode
2. **ISD #2** (Optional) - Supplementary Services (SS) data:
    - Call forwarding settings (unconditional, busy, no reply, not reachable)
    - Call waiting
    - Call hold
    - Multi-party service
    - Supplementary service status and features
3. **ISD #3** (Optional) - Call Barring data:
    - Barring of all outgoing calls (BAOC)
    - Barring of outgoing international calls (BOIC)
    - Access restriction data

Configuration Options

```
# InsertSubscriberData Configuration
# Network Access Mode: :packetAndCircuit, :packetOnly, or
:circuitOnly
isd_network_access_mode: :packetAndCircuit,

# Send ISD #2 (Supplementary Services data)
isd_send_ss_data: true,

# Send ISD #3 (Call Barring data)
isd_send_call_barring: true,
```

Network Access Mode

The `isd_network_access_mode` parameter controls what type of network access the subscriber is allowed:

Value	Description	Use Case
`:packetAndCircuit`	Both packet-switched (GPRS/LTE) and circuit-switched (voice)	Default - Full service subscribers
`:packetOnly`	Packet-switched only (data/LTE)	Data-only SIM cards, IoT devices
`:circuitOnly`	Circuit-switched only (voice/SMS)	Legacy devices, voice-only plans

Controlling ISD Messages

You can control which ISD messages are sent based on your network requirements:

Send all ISDs (Default - Full feature set):

```
isd_send_ss_data: true,
isd_send_call_barring: true,
```

Send only basic subscriber data (Minimal provisioning):

```
isd_send_ss_data: false,
isd_send_call_barring: false,
```

Send basic + supplementary services (No call barring):

```
isd_send_ss_data: true,
isd_send_call_barring: false,
```

ISD Flow Example

When UpdateLocation is received:

```
VLR → HLR: UpdateLocation (BEGIN)
HLR → VLR: InsertSubscriberData #1 (CONTINUE) - Basic data
VLR → HLR: ISD #1 ACK (CONTINUE)
HLR → VLR: InsertSubscriberData #2 (CONTINUE) - SS data [if
enabled]
VLR → HLR: ISD #2 ACK (CONTINUE)
HLR → VLR: InsertSubscriberData #3 (CONTINUE) - Call barring [if
enabled]
VLR → HLR: ISD #3 ACK (CONTINUE)
HLR → VLR: UpdateLocation Response (END)
```

If `isd_send_ss_data` or `isd_send_call_barring` are set to `false`, those ISD messages are skipped, and the UpdateLocation END is sent sooner.

Best Practices

- **Default Configuration**: Use `:packetAndCircuit` and enable all ISDs for maximum compatibility
- **IoT/M2M**: Use `:packetOnly` and disable SS data/call barring for data-only devices
- **Interoperability**: Some older VLRs may not support all supplementary services - disable `isd_send_ss_data` if encountering issues

- **Performance**: Disabling unused ISDs reduces message overhead and speeds up location updates

---

CAMEL Integration

CAMEL Configuration for SendRoutingInfo

When responding to **SendRoutingInfo** (SRI) requests from a GMSC (Gateway MSC), the HLR can instruct the GMSC to invoke CAMEL services for intelligent call routing and service control.

For configuration parameter reference, see CAMEL Configuration in Configuration Reference.

What is CAMEL?

CAMEL (Customized Applications for Mobile network Enhanced Logic) is a protocol that enables intelligent network services in GSM/UMTS networks. It allows network operators to implement value-added services like:

- Prepaid billing
- Call screening and barring
- Virtual Private Networks (VPN)
- Premium rate services
- Call forwarding with custom logic
- Location-based services

Configuration Options

```
# CAMEL Configuration (for SendRoutingInfo responses)
# Service Key for CAMEL service initiation
camel_service_key: 11_110,

# CAMEL Trigger Detection Point
# Options: :termAttemptAuthorized, :tBusy, :tNoAnswer, :tAnswer
camel_trigger_detection_point: :termAttemptAuthorized,
```

Service Key

The `camel_service_key` identifies which CAMEL service should be invoked at the gsmSCF (Service Control Function). This is a numeric identifier configured in your network:

Service Key	Typical Use Case
`11_110`	Prepaid terminating call control (default)
`100`	Originating prepaid service
`200`	Call forwarding with custom logic
`300`	Virtual Private Network (VPN)
Custom	Operator-specific services

Configuration Example:

```
# For prepaid terminating call control
camel_service_key: 11_110,

# For VPN service
camel_service_key: 300,
```

Trigger Detection Point

The `camel_trigger_detection_point` specifies when the CAMEL service should be triggered during call setup:

Detection Point	Description	When Triggered
`:termAttemptAuthorized`	Call attempt authorized (default)	Before call is routed to subscriber
`:tBusy`	Terminating busy	When subscriber is busy
`:tNoAnswer`	Terminating no answer	When subscriber doesn't answer
`:tAnswer`	Terminating answer	When subscriber answers the call

Configuration Examples:

Standard prepaid control (trigger before routing):

```
camel_trigger_detection_point: :termAttemptAuthorized,
```

Custom busy handling (trigger when busy):

```
camel_trigger_detection_point: :tBusy,
```

Answer-based billing (trigger on answer):

```
camel_trigger_detection_point: :tAnswer,
```

SRI Response with CAMEL

When configured, SendRoutingInfo responses include CAMEL subscription information:

```
GMSC → HLR: SendRoutingInfo (BEGIN)
HLR → GMSC: SRI Response (END) with:
  - IMSI
  - VLR number
  - Subscriber state
  - CAMEL routing info:
    * Service Key: 11_110
    * gsmSCF Address: <configured address>
    * Trigger Detection Point: termAttemptAuthorized
    * Default Call Handling: continueCall

GMSC contacts gsmSCF at trigger point to execute CAMEL service
```

Best Practices

- **Production Networks**: Use standardized service keys agreed upon with your gsmSCF provider
- **Testing**: Use `:termAttemptAuthorized` for most comprehensive testing
- **Prepaid Services**: Service key `11_110` is a common industry standard for prepaid terminating calls
- **Fallback Handling**: `defaultCallHandling: :continueCall` ensures calls proceed if gsmSCF is unreachable

---

Roaming Subscriber Handling

Home VLR vs Roaming VLR Detection

When the HLR receives a **SendRoutingInfo** (SRI) request, it needs to determine whether the subscriber is on a "home" VLR (within your network) or on a roaming VLR (visiting another network). The behavior differs based on this determination:

For configuration parameter reference, see Home VLR Prefixes in Configuration Reference.

- **Home VLR**: Return standard SRI response with CAMEL routing information

- **Roaming VLR**: Send a Provide Roaming Number (PRN) request to obtain an MSRN, then return it in the SRI response

Configuration

```
# Home VLR Prefixes
# List of VLR address prefixes that are considered "home" network
# If subscriber's VLR address starts with one of these prefixes,
use standard SRI response
# Otherwise, subscriber is roaming and we need to send PRN to get
MSRN
home_vlr_prefixes: ["555123"],
```

Configuration Example:

```
# Single home network
home_vlr_prefixes: ["555123"],

# Multiple home networks (e.g., different regions or subsidiaries)
home_vlr_prefixes: ["555123", "555124", "555125"],
```

How It Works

1. Home Subscriber Flow (Standard)

When the subscriber's VLR address starts with a configured home prefix:

```
GMSC → HLR: SendRoutingInfo (MSISDN: "1234567890")
HLR queries backend API for subscriber data
HLR checks VLR address: "5551234567"
HLR determines: VLR starts with "555123" → Home network
HLR → GMSC: SRI Response with CAMEL routing info:
  - IMSI
  - VLR number: "5551234567"
  - gsmSCF address (MSC): "5551234501"
  - CAMEL service key: 11_110
  - Trigger detection point: termAttemptAuthorized
```

2. Roaming Subscriber Flow (PRN Required)

When the subscriber's VLR address does NOT match any home prefix:

```
GMSC → HLR: SendRoutingInfo (MSISDN: "1234567890")
HLR queries backend API for subscriber data
HLR checks VLR address: "49170123456"
HLR determines: VLR doesn't start with "555123" → Roaming
HLR → MSC: ProvideRoamingNumber (PRN):
  - MSISDN: "1234567890"
  - IMSI: "999999876543210"
  - MSC number: "49170123456"
  - GMSC address: "5551234501"
MSC → HLR: PRN Response with MSRN: "49170999888777"
HLR → GMSC: SRI Response with routing info:
  - IMSI
  - VLR number: "49170123456"
  - Roaming Number (MSRN): "49170999888777"
```

Response Structure Differences

Home Subscriber SRI Response

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :camelRoutingInfo, %{
      gmscCamelSubscriptionInfo: %{
        "t-CSI": %{
          serviceKey: 11_110,
          "gsmSCF-Address": "5551234501",
          defaultCallHandling: :continueCall,
          "t-BcsmTriggerDetectionPoint": :termAttemptAuthorized
        }
      }
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "5551234567"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

Roaming Subscriber SRI Response

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :routingInfo, %{
      roamingNumber: "49170999888777"  # MSRN from PRN
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "49170123456"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

Provide Roaming Number (PRN) Operation

PRN Request Structure

The PRN request sent to the MSC/VLR contains:

Field	Source	Description
MSISDN	SRI request	Subscriber's phone number
IMSI	HLR API	Subscriber's IMSI
MSC Number	HLR API	MSC serving the roaming subscriber (`serving_msc`)
GMSC Address	SRI request	GMSC making the original SRI request
Call Reference Number	Static	Call reference identifier
Supported CAMEL Phases	Static	CAMEL phases supported by GMSC

PRN Response Handling

The HLR expects a PRN response containing:

- **MSRN** (Mobile Station Roaming Number): A temporary number allocated by the visited network for routing the call

Error Handling:

- If PRN times out → Returns error 27 (Absent Subscriber) in SRI response
- If PRN fails → Returns error 27 (Absent Subscriber) in SRI response
- If MSRN cannot be extracted → Returns error 27 (Absent Subscriber) in SRI response

Configuration Examples

Single Home Network Operator

```
# All VLR addresses starting with "555123" are considered home
home_vlr_prefixes: ["555123"],
```

- VLR `5551234567` → Home (CAMEL response)
- VLR `5551235001` → Home (CAMEL response)
- VLR `49170123456` → Roaming (PRN + MSRN response)

Multi-Region Operator

```
# Multiple home networks across different regions
home_vlr_prefixes: ["555123", "555124", "555125"],
```

- VLR `5551234567` → Home (region 1)
- VLR `5552341234` → Home (region 2)
- VLR `5553411111` → Home (region 3)
- VLR `44201234567` → Roaming (international)

Testing Configuration

For testing PRN functionality, set an empty list to treat all VLRs as roaming:

```
# All VLRs are treated as roaming (for testing PRN flow)
home_vlr_prefixes: [],
```

Best Practices

- **Prefix Selection**: Use the shortest unique prefix that identifies your network's VLRs (e.g., country code + network code)
- **Multiple Prefixes**: Include all VLR prefixes in your network, including different regions and subsidiaries
- **Roaming Agreements**: Ensure PRN is properly supported by roaming partner networks
- **Testing**: Test both home and roaming scenarios thoroughly before production deployment

- **Monitoring**: Monitor PRN timeout rates to identify connectivity issues with roaming partners

Troubleshooting

Symptom: All subscribers treated as roaming

- **Cause**: `home_vlr_prefixes` not configured or prefixes don't match VLR addresses
- **Solution**: Check VLR addresses in your database and update prefixes accordingly

Symptom: PRN requests timing out

- **Cause**: Network connectivity issues to roaming partner MSC/VLR
- **Solution**: Verify M3UA/SCCP routing to remote MSC addresses

Symptom: Invalid MSRN in SRI response

- **Cause**: PRN response format from roaming partner doesn't match expected structure
- **Solution**: Review PRN response logs and adjust `extract_msrn_from_prn/1` if needed

---

HLR Operations

Supported MAP Operations

OmniSS7 dispatches inbound MAP operations by opcode (independent of TCAP phase). The following inbound operations are handled:

Opcode	Operation	Purpose
2	`updateLocation`	Register VLR location (triggers ISD sequence + optional alertServiceCenter)
23	`updateGprsLocation`	Register SGSN (packet-switched) location
3	`cancelLocation`	Deregister from old VLR
7	`insertSubscriberData`	Receive subscriber profile (e.g. when acting as VLR-side)
22	`sendRoutingInfo` (SRI)	Provide MSRN/CAMEL routing for calls (home vs roaming)
45	`sendRoutingInfoForSM` (SRI-for-SM)	Provide IMSI + SMSC GT for SMS routing
44	`MT-Forward-SM`	Mobile-terminated SMS delivery
46	`MO-Forward-SM`	Mobile-originated SMS reception
47	`reportSM-DeliveryStatus`	SMS-GMSC delivery status report
64	`alertServiceCenter`	Subscriber-reachable alert to SMSc
66	`readyForSM`	MSC/SGSN signals subscriber is ready for SMS
56	`sendAuthenticationInfo`	Generate authentication vectors
57	`restoreData`	VLR subscriber-data restoration

Opcode	Operation	Purpose
67	`purgeMS`	Purge a subscriber record at the HLR
59	`processUnstructuredSS-Request`	USSD (handled via USSD gateway when enabled)
71	`anyTimeInterrogation` (ATI)	Subscriber location/state interrogation
10	`registerSS`	Supplementary-service registration (e.g. Call Forwarding Unconditional)

Unhandled opcodes are answered with a `facilityNotSupported` (21) error.

Response Field Mapping

This section details where each field in HLR responses comes from.

SendRoutingInfo (SRI) Response

Purpose: Provides routing information for incoming calls to a subscriber.

The HLR provides two different response types based on whether the subscriber is on a home VLR or roaming:

Home Subscriber Response (CAMEL Routing)

Used when the subscriber's VLR address starts with a configured `home_vlr_prefixes` value.

Response Structure:

Field	Source	Description
IMSI	OmniHSS API	Subscriber's IMSI from OmniHSS database
VLR Number	OmniHSS API	Current VLR serving the subscriber (`circuit_session.assigned_vlr`)
Subscriber State	Static	Always `notProvidedFromVLR`
extendedRoutingInfo	-	Type: `camelRoutingInfo`
gsmSCF Address	OmniHSS API	MSC serving the subscriber (`circuit_session.assigned_msc`)
Service Key	config.exs	CAMEL service identifier (`camel_service_key`)
Trigger Detection Point	config.exs	When to trigger CAMEL (`camel_trigger_detection_point`)
CAMEL Capability Handling	Static	CAMEL phase support level
Default Call Handling	Static	Fallback if gsmSCF unreachable

Roaming Subscriber Response (MSRN Routing)

Used when the subscriber's VLR address does NOT match any configured `home_vlr_prefixes` value.

Response Structure:

Field	Source	Description	
IMSI	OmniHSS API	Subscriber's IMSI from OmniHSS database	`"`
VLR Number	OmniHSS API	Current VLR serving the subscriber (`circuit_session.assigned_vlr`)	`"`
Subscriber State	Static	Always `notProvidedFromVLR`	`:`
extendedRoutingInfo	-	Type: `routingInfo`	-
Roaming Number (MSRN)	PRN Response	MSRN obtained from ProvideRoamingNumber request	`"`

Routing Decision Logic:

```
1. OmniSS7 receives SendRoutingInfo request
2. OmniSS7 queries subscriber data from OmniHSS API
3. OmniSS7 checks VLR address against home_vlr_prefixes:

   If VLR starts with home prefix:
     → Return CAMEL routing info (home subscriber flow)

   If VLR does NOT match any home prefix:
     → Send ProvideRoamingNumber (PRN) to MSC
     → Extract MSRN from PRN response
     → Return routing info with MSRN (roaming subscriber flow)
```

Data Flow:

- OmniSS7 queries OmniHSS for subscriber information
- OmniHSS returns IMSI, current VLR/MSC location, and subscriber state
- OmniSS7 uses this data to construct the MAP response

Configuration Requirements:

```
# In config/config.exs
home_vlr_prefixes: ["555123"],  # List of home VLR prefixes
```

Error Responses:

- If `serving_vlr` and `serving_msc` are `null`: Returns error 27 (Absent Subscriber)
- If subscriber not found: Returns error 1 (Unknown Subscriber)
- If PRN request times out (roaming case): Returns error 27 (Absent Subscriber)
- If PRN response invalid (roaming case): Returns error 27 (Absent Subscriber)

UpdateLocation Response with InsertSubscriberData

Purpose: Registers subscriber at new VLR and provisions subscriber data.

UpdateLocation END Response

Field	Source	Description	Example
HLR Number	config.exs	This HLR's Global Title (`hlr_service_center_gt_address`)	`"5551234568"`
TCAP Message Type	Static	Final response after all ISDs	`END`

InsertSubscriberData #1 (Basic Subscriber Data)

Field	Source	Description	Example
IMSI	Request	From UpdateLocation request	`"999999876543210`
MSISDN	OmniHSS API	Subscriber's phone number from OmniHSS	`"555123456"`
Category	Static	Subscriber category	`"\n"` (0x0A)
Subscriber Status	Static	Service status	`:serviceGranted`
Bearer Service List	Static	Supported bearer services	`[<<31>>]`
Teleservice List	Static	Supported teleservices	`[<<17>>, "!", "\""]`
Network Access Mode	config.exs	Packet/circuit access (`isd_network_access_mode`)	`:packetAndCircui`

InsertSubscriberData #2 (Supplementary Services) - Optional

Field	Source	Description	Controlled By
Provisioned SS	Static	Supplementary services data	`isd_send_ss_data: true`
Call Forwarding	Static	Forwarding configurations (unconditional, busy, no reply, not reachable)	Config enabled
Call Waiting	Static	Call waiting service status	Config enabled
Multi-party Service	Static	Conference call support	Config enabled

ISD #2 includes:

- Call forwarding unconditional (SS code 21)
- Call forwarding on busy (SS code 41)
- Call forwarding on no reply (SS code 42)
- Call forwarding on not reachable (SS code 62)
- Call waiting (SS code 43)
- Multi-party service (SS code 51)
- CLIP/CLIR services

InsertSubscriberData #3 (Call Barring) - Optional

Field	Source	Description	Controlled By
Call Barring Info	Static	Call barring configurations	`isd_send_call_barring: true`
BAOC	Static	Barring of All Outgoing Calls (SS code 146)	Config enabled
BOIC	Static	Barring of Outgoing International Calls (SS code 147)	Config enabled
Access Restriction Data	Static	Network access restrictions	Config enabled

ISD Sequence Control:

- ISD #1: **Always sent** - Contains essential subscriber data
- ISD #2: Sent only if `isd_send_ss_data: true` in config/config.exs
- ISD #3: Sent only if `isd_send_call_barring: true` in config/config.exs

SendRoutingInfoForSM (SRI-for-SM) Response

Purpose: Provides MSC/SMSC routing information for SMS delivery. When an SMSc needs to deliver an SMS to a subscriber, it sends a SRI-for-SM request to the HLR to determine where to route the message.

Response Structure:

Field	Source	Description	How Generated
IMSI	Calculated	Synthetic IMSI derived from MSISDN	`PLMN_PREFIX + zero_padded_MSISDN`
Network Node Number	config.exs	SMSC GT address for SMS routing	`smsc_service_center_gt_address`

Configuration Parameters (from `config/config.exs`):

```
# Service Center GT Address (returned in SRI-for-SM responses)
# This tells the requesting SMSc where to send MT-ForwardSM
messages
smsc_service_center_gt_address: "5551234567",  # Required

# MSISDN ↔ IMSI Mapping Configuration
# PLMN prefix: MCC (001 = Test Network) + MNC (01 = Test Operator)
hlr_imsi_plmn_prefix: "001001",                 # Only config
parameter needed!
```

MSISDN ↔ IMSI Mapping

Configuration Parameters:

These parameters control how OmniSS7 generates synthetic IMSIs from MSISDNs for SRI-for-SM responses:

- **`hlr_imsi_plmn_prefix`**: The MCC+MNC prefix to use when constructing synthetic IMSIs (e.g., "50557" for MCC=505, MNC=57)
- **`hlr_msisdn_country_code`**: Country code to prepend when doing reverse IMSI→MSISDN mapping (e.g., "61" for Australia, "1" for USA/Canada)
- **`hlr_msisdn_nsn_offset`**: Character position where the National Subscriber Number (NSN) starts within the MSISDN (typically 0 if MSISDN doesn't

include country code, or length of country code if it does)
- `hlr_msisdn_nsn_length`: Number of digits to extract from the MSISDN as the NSN

For additional configuration details, see MSISDN ↔ IMSI Mapping in Configuration Reference.

Why is MSISDN to IMSI Mapping Needed?

The MAP protocol for **SendRoutingInfoForSM** (SRI-for-SM) requires the HLR to return an **IMSI** (International Mobile Subscriber Identity) in its response. However, the requesting SMSc only knows the subscriber's **MSISDN** (phone number).

In a traditional network:

- The SMSc sends SRI-for-SM with the destination MSISDN (e.g., "5551234567")
- The HLR must look up the subscriber in its database to find their IMSI
- The HLR returns the IMSI in the SRI-for-SM response
- The SMSc then uses this IMSI when sending MT-ForwardSM to the MSC/VLR

OmniSS7's Approach - Synthetic IMSIs:

Instead of maintaining a full subscriber database with MSISDN-to-IMSI mappings, OmniSS7 uses a simple encoding scheme to **calculate** synthetic IMSIs directly from the MSISDN. This approach provides two key benefits:

1. **Privacy**: Real subscriber IMSIs stored in the HLR database are never exposed in SRI-for-SM responses sent over the SS7 network
2. **Simplicity**: No need to query the HLR database for IMSI lookups during SRI-for-SM operations - the IMSI is calculated on-the-fly from the MSISDN

How It Works:

MSISDNs are encoded directly into the subscriber portion of the IMSI (the digits after MCC+MNC):

```
IMSI = PLMN_PREFIX + zero_padded_MSISDN
```

Where:

- **PLMN_PREFIX**: MCC + MNC (e.g., "001001" for Test Network)
- **MSISDN**: All numeric digits from the phone number
- **Zero Padding**: Left-padded with zeros to fill IMSI to exactly 15 digits

Step-by-Step Example:

```
# Configuration
plmn_prefix = "001001"  # MCC 001 + MNC 01

# Input: MSISDN from SRI-for-SM request (TBCD decoded)
msisdn = "555123456"    # 9 digits

# Step 1: Calculate available space for subscriber number
subscriber_digits = 15 - String.length("001001")  # = 9 digits

# Step 2: Left-pad MSISDN with zeros to fill subscriber portion
padded_msisdn = String.pad_leading("555123456", 9, "0")  # =
"555123456" (no padding needed)

# Step 3: Concatenate PLMN prefix + padded MSISDN
imsi = "001001" <> "555123456"  # = "001001555123456" (exactly 15
digits)
```

Complete Examples:

Input MSISDN	PLMN Prefix	Subscriber Digits Available	Padded MSISDN	Final IMSI
`"555123456"`	`"001001"` (6)	9	`"555123456"`	`"001001555123`
`"99"`	`"001001"` (6)	9	`"000000099"`	`"001001000000`
`"999999999"`	`"001001"` (6)	9	`"999999999"`	`"001001999999`
`"91123456789"`	`"001001"` (6)	9	`"555123456"`	`"001001555123`

Edge Case Handling:

- **Short MSISDNs**: Left-padded with zeros (e.g., `"99"` → `"000000099"`)
- **Long MSISDNs**: Rightmost digits are kept, leftmost digits are truncated (e.g., `"91123456789"` → `"555123456"`)
- **IMSI Length**: Always exactly 15 digits

Reverse Mapping (IMSI → MSISDN):

The SMSc can reverse this mapping to convert IMSIs back to MSISDNs:

```
# Input: IMSI from SRI-for-SM response
imsi = "001001555123456"

# Step 1: Strip PLMN prefix
plmn_prefix = "001001"
subscriber_portion = String.slice(imsi, 6, 9)  # = "555123456"

# Step 2: Remove leading zeros to get actual MSISDN
msisdn = String.replace_leading(subscriber_portion, "0", "")  # =
"555123456"
```

Reverse Mapping Examples:

Input IMSI	PLMN Prefix	Subscriber Portion	Remove Leading Zeros	Final MSISDI
`"001001555123456"`	`"001001"`	`"555123456"`	`"555123456"`	`"5551234`
`"001001000000099"`	`"001001"`	`"000000099"`	`"99"`	`"99"`
`"001001999999999"`	`"001001"`	`"999999999"`	`"999999999"`	`"9999999`

Properties of This Mapping:

- □ **Deterministic**: Same MSISDN always produces same IMSI
- □ **Reversible**: Can convert back from IMSI to MSISDN
- □ **Minimal Configuration**: Only requires `hlr_imsi_plmn_prefix`
- □ **Privacy-Preserving**: Real IMSIs never exposed
- □ **No Database Lookup**: Fast calculation, no API calls needed
- □ **Always 15 Digits**: IMSI is always exactly 15 digits

MSISDN Input Handling:

When the HLR receives a SRI-for-SM request, the MSISDN undergoes TBCD decoding:

1. **TBCD Decode**: Convert binary TBCD to string (may include TON/NPI prefix like "91")
2. **Extract Digits**: Keep only numeric digits, strip any non-digit characters
3. **Normalize**: If longer than available space, take rightmost digits; if shorter, left-pad with zeros
4. **Encode**: Concatenate PLMN prefix + normalized MSISDN

Security Considerations:

The synthetic IMSIs returned in SRI-for-SM responses are purely for routing purposes. They are NOT the real IMSIs stored in the HLR subscriber database. This provides an additional layer of privacy protection, as real subscriber IMSIs are only exposed when absolutely necessary (e.g., during UpdateLocation or SendAuthenticationInfo operations that require real authentication vectors).

Response Flow:

```
1. SMSc → HLR: SRI-for-SM Request
   - MSISDN (TBCD): "91123456789" (includes TON/NPI)

2. HLR Processing:
   - TBCD decode: "91123456789"
   - Extract digits: "91123456789" (11 digits)
   - Fit to 9 digits: "555123456" (rightmost 9)
   - Add PLMN: "001001" + "555123456" = "001001555123456"
   - Get SMSC GT from config: "5551234567"

3. HLR → SMSc: SRI-for-SM Response
   - IMSI: "001001555123456" (synthetic, always 15 digits)
   - Network Node Number: "5551234567" (where to send MT-
ForwardSM)

4. SMSc sends MT-ForwardSM to "5551234567" with IMSI
"001001555123456"
```

Configuration:

The following parameters are used in `config/config.exs`:

```
# PLMN prefix: MCC (001 = Test Network) + MNC (01 = Test Operator)
hlr_imsi_plmn_prefix: "001001",

# NSN Extraction (if MSISDNs include country code)
hlr_msisdn_country_code: "1",      # Used for reverse mapping
(IMSI→MSISDN)
hlr_msisdn_nsn_offset: 1,          # Skip 1-digit country code
hlr_msisdn_nsn_length: 10          # Extract 10-digit NSN
```

NSN Extraction Configuration:

If your MSISDNs include the country code (e.g., `"68988000088"` instead of just `"88000088"`), you must configure NSN extraction:

- **`hlr_msisdn_nsn_offset`**: Position where NSN starts (typically the length of your country code)
- **`hlr_msisdn_nsn_length`**: Number of digits in the NSN

Examples:

Example	Country Code	MSISDN Example	nsn_offset	nsn_length	I Ext
1-digit CC	`"9"`	`"95551234567"`	`1`	`10`	`"555`
2-digit CC	`"99"`	`"99412345678"`	`2`	`9`	`"412`
3-digit CC	`"999"`	`"99988000088"`	`3`	`8`	`"880`

How It Works:

1. **MSISDN → IMSI**: Extract NSN from MSISDN, pad with leading zeros, concatenate with PLMN prefix

```
MSISDN: "99988000088"
NSN: String.slice("99988000088", 3, 8) = "88000088"
Padded NSN: "088000088" (9 digits)
IMSI: "547050" + "088000088" = "547050088000088"
```

2. **IMSI → MSISDN**: Strip PLMN prefix, remove leading zeros, prepend country code

```
IMSI: "547050088000088"
Subscriber portion: "088000088"
Remove zeros: "88000088"
MSISDN: "+999" + "88000088" = "+99988000088"
```

API Requirements: None - SRI-for-SM uses calculated values and

config only. No backend API calls are required.

Field Source Summary

Source Type	Description	Examples
OmniHSS API	Dynamic data from OmniHSS subscriber database	IMSI, MSISDN, serving VLR/MSC from circuit_session
config.exs	OmniSS7 configuration parameters	`smsc_service_center_gt_address`, `camel_service_key`, `isd_network_access_mode`
Static	Hardcoded values in response generator	Subscriber status, bearer services, SS codes
Request	Fields extracted from incoming MAP request	IMSI from UpdateLocation, MSISDN from SRI
Calculated	Derived values using logic	Synthetic IMSI in SRI-for-SM (hlr_imsi_prefix + NSN)

Configuration Dependencies

Required in config/config.exs:

- `hlr_service_center_gt_address` - Used in UpdateLocation responses

- `smsc_service_center_gt_address` - Used in SRI-for-SM responses (where MT-ForwardSM should be routed)

Optional in config/config.exs (with defaults):

- `camel_service_key` - Default: `11_110`
- `camel_trigger_detection_point` - Default: `:termAttemptAuthorized`
- `isd_network_access_mode` - Default: `:packetAndCircuit`
- `isd_send_ss_data` - Default: `true`
- `isd_send_call_barring` - Default: `true`
- `hlr_imsi_plmn_prefix` - Default: `"001001"` (PLMN prefix for MSISDN↔IMSI mapping)

Required from OmniHSS:

OmniHSS must provide REST API endpoints for:

- Subscriber lookup by IMSI and MSISDN
- Circuit session location updates (VLR/MSC assignment)
- Authentication vector generation
- Subscriber status and service profile queries

Related Documentation

OmniSS7 Documentation:

- ← Back to Main Documentation
- Common Features Guide
- MAP Client Guide
- Technical Reference
- Configuration Reference

OmniHSS Documentation: For subscriber management, provisioning, authentication configuration, and administrative operations, refer to the **OmniHSS product documentation**. OmniHSS contains all the subscriber

database logic, authentication algorithms, service provisioning rules, and Multi-IMSI management capabilities.

OmniSS7 by Omnitouch Network Services

MAP Client Configuration Guide

← Back to Main Documentation

This guide covers operating OmniSS7 as a **MAP Client** — connecting to an SS7 network as an Application Server Process (ASP) and issuing MAP (Mobile Application Part) operations toward HLRs, MSC/VLRs, SGSNs, GMLCs and other core-network elements. Protocol behaviour follows 3GPP TS 29.002.

Table of Contents

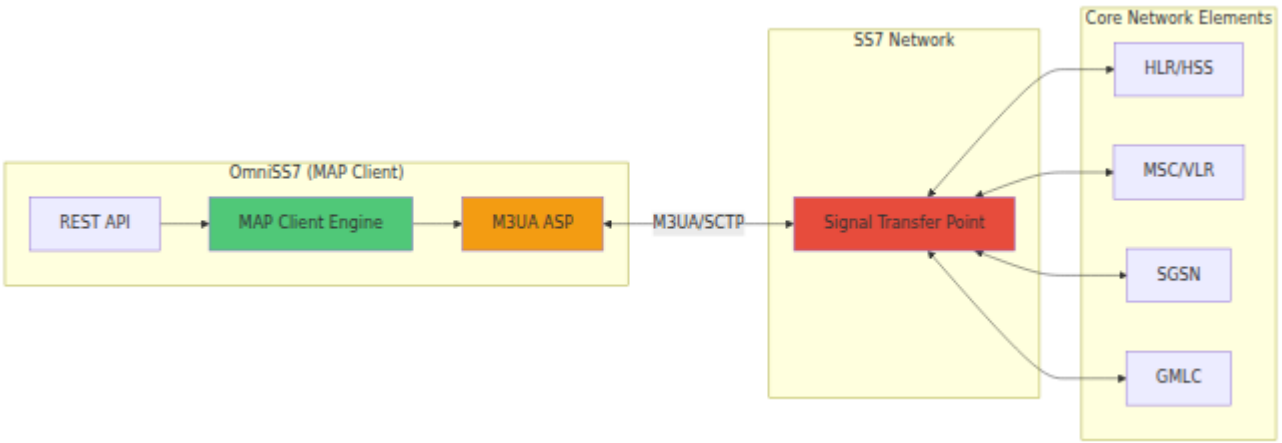
What is MAP Client Mode?

MAP Client Mode lets OmniSS7 connect as an **Application Server Process (ASP)** to an M3UA peer (STP or SGP) over SCTP and exchange **MAP** messages. Each operation is exposed as an HTTP endpoint: a request to the API is encoded as a TCAP/MAP dialogue, carried over SCCP and M3UA to the SS7 network, and the network's response is decoded and returned synchronously to the caller.

Typical uses include:

- **Mobility management** — Update Location, Cancel Location, Purge MS, Provide Roaming Number
- **Call handling** — Send Routing Info, Provide Subscriber Info
- **SMS** — SRI-for-SM, MO/MT-ForwardSM, delivery-status reporting
- **Authentication** — Send Authentication Info
- **Subscriber data** — Insert/Delete Subscriber Data, Any-Time Interrogation
- **Supplementary services** — register/erase/activate/interrogate SS, password handling
- **Location services (LCS)** and **GPRS** routing

Network Architecture



Enabling MAP Client Mode

MAP Client mode is configured in `config/runtime.exs`. For the full M3UA connection reference see M3UA Connection Parameters.

```
config :omniss7,
  # Enable MAP Client mode
  map_client_enabled: true,

  # M3UA connection for the MAP Client (connects as an ASP to a
remote STP/SGP)
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},
    remote_port: 2905,
    routing_context: 1
  }
```

`map_client_m3ua` Parameters

Parameter	Type	Required	Default	Description
`mode`	String	Yes	-	M3UA role. `"ASP"` for client mode (connects out to an STP/SGP).
`callback`	Tuple	Yes	-	`{Module, Function, Args}` invoked for each inbound M3UA payload. For MAP Client this is `{MapClient, :handle_payload, []}`.
`process_name`	Atom	Yes	-	Registered name of the M3UA ASP process.
`local_ip`	Tuple	Yes	-	Local SCTP bind address as an IP tuple, e.g. `{10, 0, 0, 100}`.
`local_port`	Integer	No	2905	Local SCTP port. 2905 is the IANA-registered M3UA port.
`remote_ip`	Tuple	Yes	-	Remote STP/SGP IP address as a tuple.
`remote_port`	Integer	No	2905	Remote STP/SGP SCTP port.

Parameter	Type	Required	Default	Description
`routing_context`	Integer	No	-	M3UA Routing Context value, when the peer requires one.

> Top-level keys: `map_client_enabled` (Boolean, default `false`) turns the MAP Client on; `map_client_m3ua` (Map) holds the connection above.

The SCCP addressing applied to outgoing operations (Calling/Called Party Global Titles, SSNs, point codes) is configured separately — see the Configuration Reference.

MAP Operations Reference

Every operation below is reachable as `POST /api/<endpoint>`. Opcodes are the MAP operation codes from 3GPP TS 29.002. Requests are synchronous: the API blocks until the SS7 response arrives or the request times out (see Response Codes).

Mobility Management

Operation	Endpoint	Opcode	Description
Update Location	`/api/updateLocation`	2	Register a subscriber's serving VLR with the HLR.
Cancel Location	`/api/cancelLocation`	3	Instruct a VLR to delete a subscriber record.
Provide Roaming Number	`/api/prn`	4	Obtain an MSRN from the serving MSC.
Purge MS	`/api/purgeMS`	67	Mark a subscriber as purged at the HLR.
Send Identification	`/api/sendIdentification`	55	Retrieve IMSI/authentication data from the previous VLR.
Reset	`/api/reset`	37	HLR-initiated reset toward a VLR.
Restore Data	`/api/restoreData`	57	VLR requests subscriber data restoration from the HLR.

Call Handling & Subscriber Data

Operation	Endpoint	Opcode	Desc
Send Routing Info	`/api/sri`	22	Quer HLR f voice routi infor
Provide Subscriber Info	`/api/provideSubscriberInfo`	70	Requ subs state from
Delete Subscriber Data	`/api/deleteSubscriberData`	8	Remo subs data VLR.
Any-Time Interrogation	`/api/anyTimeInterrogation`	71	gsmS queri subs info/l at th
Any-Time Subscription Interrogation	`/api/anyTimeSubscriptionInterrogation`	62	Quer subs data HLR.
Any-Time Modification	`/api/anyTimeModification`	65	Modi subs data HLR.

Operation	Endpoint	Opcode	Desc
Note Subscriber Data Modified	`/api/noteSubscriberDataModified`	5	HLR r the g of ch data.
Note MM-Event	`/api/noteMM-Event`	89	Repo mobi mana even gsmS

SMS

Operation	Endpoint	Opcode	Description
Send Routing Info for SM	`/api/sri-for-sm`	45	Query the HLR for the serving node for SMS delivery.
MT-Forward SM	`/api/MT-forwardSM`	44	Deliver a mobile-terminated SM to the serving MSC/SGSN.
MO-Forward SM	`/api/forwardSM`	46	Submit a mobile-originated SM to the SMSC.
Send SM (PDU)	`/api/sendSM`, `/api/deliverPDU`	44	Build and deliver an SMS-DELIVER PDU.
Report SM Delivery Status	`/api/reportSM-DeliveryStatus`	47	Report SM delivery outcome to the HLR.
Ready for SM	`/api/readyForSM`	66	Notify the HLR that an MS is reachable for SMS.
Alert Service Centre	`/api/alertServiceCentre`	64	Alert an SMSC that a subscriber is available.

Authentication & Equipment

Operation	Endpoint	Opcode	Description
Send Authentication Info	`/api/send-auth-info`	56	Retrieve authentication vectors from the HLR.
Send IMSI	`/api/sendIMSI`	58	Resolve an MSISDN to its IMSI at the HLR.
Check IMEI	`/api/checkIMEI`	43	Query equipment status at the EIR.

Supplementary Services

Operation	Endpoint	Opcode	Description
Register SS	`/api/registerSS`	10	Register a supplementary service (e.g. call forwarding / CFU).
Erase SS	`/api/eraseSS`	11	Erase a supplementary service.
Activate SS	`/api/activateSS`	12	Activate a supplementary service.
Deactivate SS	`/api/deactivateSS`	13	Deactivate a supplementary service.
Interrogate SS	`/api/interrogateSS`	14	Interrogate the status of a supplementary service.
Register Password	`/api/registerPassword`	17	Register an SS password.
Get Password	`/api/getPassword`	18	Retrieve/verify an SS password.
Forward Check SS Indication	`/api/forwardCheckSS-Indication`	38	Unconfirmed check toward a VLR.

Location Services (LCS) & GPRS

Operation	Endpoint	Opcode	Description
Send Routing Info for LCS	`/api/sendRoutingInfoForLCS`	85	GMLC queries the HLR for LCS routing.
Provide Subscriber Location	`/api/provideSubscriberLocation`	83	Request a subscriber's location from the serving node.
Subscriber Location Report	`/api/subscriberLocationReport`	86	Report a subscriber's location to the GMLC.
Send Routing Info for GPRS	`/api/sendRoutingInfoForGprs`	24	Query the HLR for GPRS routing (GGSN selection).
Update GPRS Location	`/api/updateGprsLocation`	23	Register a subscriber's serving SGSN with the HLR.

> **LCS scope:** the Location Services operations are **outbound (requester) only** — the MAP Client triggers them toward an HLR, serving node or GMLC. The node does not act as an LCS *server* (it will not answer an

> incoming ProvideSubscriberLocation or process an inbound SubscriberLocationReport). The encoders carry the mandatory information elements plus the core options noted above; the broader LCS parameter set (QoS, priority, GAD shapes, codeword, reference number) is not yet exposed.

CAP / CAMEL & Raw Injection

Operation	Endpoint	Description
CAP InitialDP	`/api/cap/initialDP`	Trigger a CAMEL dialogue toward a gsmSCF.
CAP Connect / Continue / ReleaseCall / ApplyCharging	`/api/cap/connect`, `/api/cap/continue`, `/api/cap/releaseCall`, `/api/cap/applyCharging`	CAMEL call-control operations. See the CAMEL Gateway Guide.
Raw SCCP	`/api/raw/sccp`	Inject a hand-built SCCP payload (testing/diagnostics).
Raw M3UA	`/api/raw/m3ua`	Inject a hand-built M3UA payload (testing/diagnostics).

Common Operation Examples

Send Routing Info for SM (SRI-for-SM)

Queries the HLR for the serving node used to deliver an SMS. For how the HLR side processes this, see SRI-for-SM in the HLR Guide.

Endpoint: `POST /api/sri-for-sm`

Request:

```
{
  "msisdn": "447712345678",
  "serviceCenter": "447999123456"
}
```

Response:

```
{
  "result": {
    "imsi": "234509876543210",
    "locationInfoWithLMSI": {
      "networkNode-Number": "447999555111"
    }
  }
}
```

cURL:

```
curl -X POST http://localhost/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{"msisdn": "447712345678", "serviceCenter": "447999123456"}'
```

Send Routing Info (SRI)

Endpoint: `POST /api/sri`

Request:

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456"
}
```

Provide Roaming Number (PRN)

Endpoint: `POST /api/prn`

Request:

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456",
  "msc_number": "447999555111",
  "imsi": "234509876543210"
}
```

Send Authentication Info

Endpoint: `POST /api/send-auth-info`

Request:

```
{
  "imsi": "234509876543210",
  "vectors": 5
}
```

Response:

```
{
  "result": {
    "authenticationSetList": [
      {
        "rand": "0123456789ABCDEF0123456789ABCDEF",
        "xres": "ABCDEF0123456789",
        "ck": "0123456789ABCDEF0123456789ABCDEF",
        "ik": "FEDCBA9876543210FEDCBA9876543210",
        "autn": "0123456789ABCDEF0123456789ABCDEF"
      }
    ]
  }
}
```


Update Location

Registers a subscriber's serving VLR with the HLR, which responds with the Insert Subscriber Data sequence. See Location Updates in the HLR Guide.

Endpoint: `POST /api/updateLocation`

Request:


```
{
  "imsi": "234509876543210",
  "vlr": "447999555111"
}
```


Any-Time Interrogation (ATI)

Endpoint: `POST /api/anyTimeInterrogation`

Request:


```
{
  "hlr_gt": "447999123456",
  "msisdn": "447712345678"
}
```

Operation Categories

Sending Requests via API

Using Swagger UI

The Swagger UI provides an interactive interface for every endpoint.

1. Navigate to `http://your-server/swagger`.
2. Select an endpoint (e.g. `/api/sri-for-sm`) and click **Try it out**.
3. Fill in the request body and click **Execute**.
4. The decoded SS7 response is shown below.

The raw OpenAPI document is available at `http://your-server/swagger.json`.

API Response Codes

Code	Meaning
200	Success — the decoded result is in the response body.
400	Bad request — invalid or missing parameters.
504	Gateway timeout — no SS7 response within the request timeout (default 10 seconds).

Some operations are unconfirmed (fire-and-forget) and return immediately once the message is sent.

MAP Client Metrics

All metrics are exported on the Prometheus endpoint (`/metrics`).

Metric: `map_requests_total` **Type:** Counter **Description:** Total MAP requests sent. **Labels:** `operation` — e.g. `sri`, `sri_for_sm`, `prn`, `authentication_info`, `update_location`.

Metric: `map_request_errors_total` **Type:** Counter **Description:** Total MAP requests that resulted in an error or timeout. **Labels:** `operation`.

Metric: `map_request_duration_milliseconds` **Type:** Histogram **Description:** Distribution of MAP request round-trip times. **Labels:** `operation`.

Metric: `map_pending_requests` **Type:** Gauge **Description:** MAP requests currently awaiting a response.

Example Prometheus Queries

```
# SRI-for-SM requests in the last hour
increase(map_requests_total{operation="sri_for_sm"}[1h])

# Average SRI response time
rate(map_request_duration_milliseconds_sum{operation="sri"}[5m])
  / rate(map_request_duration_milliseconds_count{operation="sri"}
[5m])

# Error rate per operation
sum(rate(map_request_errors_total[5m])) by (operation)

# Current pending requests
map_pending_requests
```

Troubleshooting MAP Client

Requests Time Out (504)

Symptoms: The API returns 504; no response from the HLR/MSC.

Possible causes:

- The M3UA association to the STP/SGP is not ACTIVE.
- The STP has no route for the destination Global Title.
- Incorrect SCCP addressing (Called/Calling Party GT, SSN) or Routing Context.

Resolution:

1. Confirm the M3UA association status on the dashboard or via the STP Guide status views.
2. Verify network connectivity (SCTP) to the STP.
3. Confirm the destination Global Title is routable and the destination subsystem (e.g. HLR SSN 6) is in service.
4. Review logs for SCCP return messages (see below).

SCCP Return / Error Messages

Symptoms: Logs show SCCP Unitdata Service (UDTS) return messages.

Common return causes:

- **No Translation for Address** — the Global Title is not in the STP routing table.
- **Subsystem Failure** — the destination subsystem (e.g. HLR SSN 6) is unavailable.
- **Network Failure / Congestion** — transient network problem.

Resolution:

- Confirm the STP routing/translation configuration for the destination GT.

- Verify the destination subsystem is operational.
- Retry after congestion clears.

Related Documentation

- ← Back to Main Documentation
- Configuration Reference — all configuration parameters
- Common Features Guide — Web UI, API, monitoring
- STP Guide — routing and Global Title Translation
- HLR Guide — HLR-side operation processing
- SMS Center Guide — SMS delivery
- CAMEL Gateway Guide — CAP/CAMEL operations

OmniSS7 by Omnitouch Network Services

SMS Center (SMSc) Configuration Guide

← Back to Main Documentation

This guide provides detailed configuration for using OmniSS7 as an **SMS Center (SMSc)** frontend with **OmniMessage** as the backend message store and delivery platform.

OmniMessage Integration

OmniSS7 SMSc mode functions as an SS7 signaling frontend that interfaces with **OmniMessage**, a carrier-grade SMS platform. This architecture separates concerns:

- **OmniSS7 (SMSc Frontend)**: Handles all SS7/MAP protocol signaling, SCCP routing, and network communication
- **OmniMessage (SMS Backend)**: Manages message storage, queuing, retry logic, delivery tracking, and routing decisions

Why OmniMessage?

OmniMessage provides carrier-grade SMS messaging capabilities with features including:

- **Message Queue Management**: Persistent storage with configurable retry logic and priority queuing
- **Delivery Tracking**: Real-time delivery status, delivery reports (DLR), and failure reason tracking
- **Multi-SMSc Support**: Multiple frontend instances can connect to a single OmniMessage backend for load balancing and redundancy
- **Routing Intelligence**: Advanced routing rules based on destination, sender, message content, and time of day

- **Rate Limiting**: Per-route TPS (transactions per second) controls to prevent network congestion
- **API-First Design**: RESTful HTTP API for integration with billing systems, customer portals, and third-party applications
- **Analytics & Reporting**: Message volume statistics, delivery success rates, and performance metrics

All message data, delivery state, and routing configurations are stored and managed in OmniMessage. OmniSS7 queries OmniMessage via HTTPS API calls to retrieve pending messages, update delivery status, and register as an active frontend.

Important: OmniSS7 SMSc mode is a **signaling frontend only**. All message routing logic, queue management, retry algorithms, delivery tracking, and business rules are handled by OmniMessage. This guide covers the SS7/MAP protocol configuration in OmniSS7. For information about message routing, queue configuration, delivery reports, rate limiting, and analytics, **refer to the OmniMessage documentation**.

Table of Contents

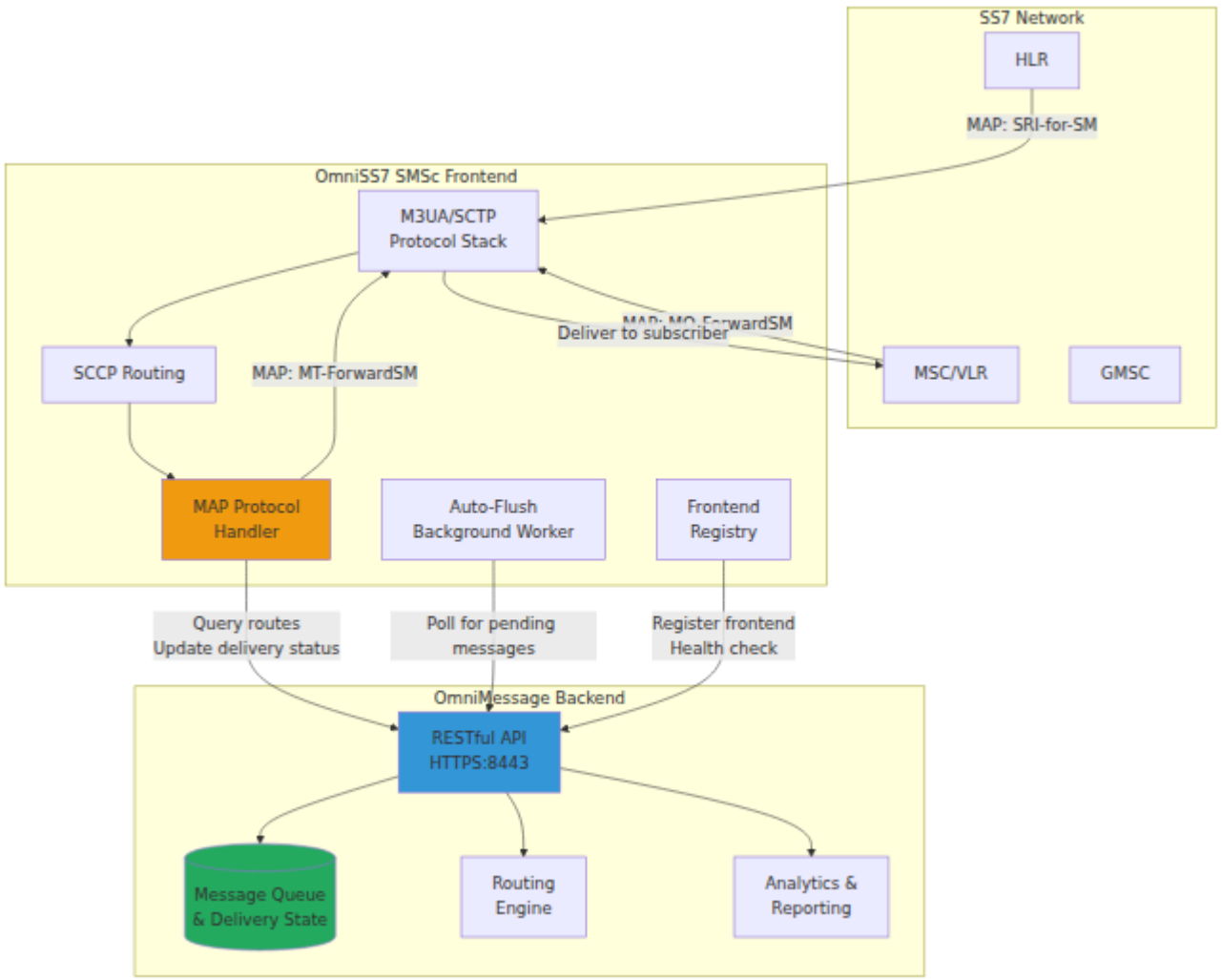
What is SMS Center Mode?

Note: This section covers OmniSS7's SS7 signaling configuration only. For message routing rules, queue management, delivery tracking, and business logic configuration, see the **OmniMessage product documentation**.

SMS Center Mode enables OmniSS7 to function as an SMSc for:

- **MT-SMS Delivery**: Mobile-Terminated SMS delivery to subscribers
- **MO-SMS Handling**: Mobile-Originated SMS reception and routing
- **Message Queuing**: Database-backed message queue with retry logic
- **Auto-Flush**: Automatic SMS delivery from queue
- **Delivery Reports**: Track message delivery status

SMS Center Architecture



Enabling SMSc Mode

OmniSS7 can operate in different modes (STP, HLR, SMSc). The mode is selected through a set of mode flags and the associated configuration in **`config/config.exs`**, the application's default configuration file.

Switching to SMSc Mode

To run OmniSS7 as an SMSc, set the mode flags so that SMSc features are enabled, then add the SMSc-specific configuration:

1. **Open** `config/config.exs`
2. **Set** the mode flags under `config :omniss7`:
   - `map_client_enabled: true` — required for routing/MAP capabilities
   - `smsc_mode_enabled: true` — enables SMSc-specific handling
   - `hlr_mode_enabled: false` — disables HLR features
3. **Add** the SMSc configuration parameters (API endpoint, SMSc name, GT address, auto-flush settings, M3UA connection) shown below
4. **Customize** the parameters for your deployment
5. **Restart** the application for the changes to take effect

> **Note**: `config/runtime.exs` is a minimal stub used for automated testing only — it does **not** contain operational mode definitions. All mode and SMSc configuration lives in `config/config.exs` (with environment-specific overrides in files such as `config/test.exs`). Production deployments supply their own `config/config.exs`.

SMSc Mode Configuration

The complete SMSc configuration (in `config/config.exs`) looks like this:

```
config :omniss7,
  # Mode flags - Enable STP + SMSc features
  # Note: map_client_enabled is true because SMSc needs routing
capabilities
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  # OmniMessage Backend API Configuration
  smsc_api_base_url: "https://10.179.3.219:8443",
  # SMSc identification for registration with backend
  smsc_name: "ipsmgw",
  # Service Center GT Address for SMS operations
  smsc_service_center_gt_address: "5551234567",

  # Auto Flush Configuration (background SMS queue processing)
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  # M3UA Connection Configuration
  # Connect as ASP for sending/receiving MAP SMS operations
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    # Local endpoint (SMSc system)
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    # Remote STP endpoint
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

The web UI pages are registered separately under `config :control_panel` in `config/config.exs`. The default `use_additional_pages` list registers all OmniSS7 pages (~16 entries); pages that don't apply to the active mode are filtered out at runtime. Note the peer-status page is labelled **"Peers"**:

```
config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 Events"},
    {SS7.Web.TestClientLive, "/client", "SS7 Client"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "Peers"},
    {SS7.Web.SCTPConnectionsLive, "/sctp", "SCTP Connections"},
    {SS7.Web.RoutingLive, "/routing", "Routing"},
    {SS7.Web.RoutingTestLive, "/routing_test", "Routing Test"},
    {SS7.Web.SmscLinksLive, "/smsc_links", "SMSc Links"},
    {SS7.Web.SmscSubscribersLive, "/smsc_subscribers", "SMSc
Subscribers"},
    {SS7.Web.HlrLinksLive, "/hlr_links", "HLR Links"},
    # ... additional pages (Active Subscribers, CAMEL, Logs,
MSISDN/IMSI Test, etc.)
  ],
  page_order: [
    "/application", "/configuration", "/events", "/logs",
"/client", "/m3ua",
    "/routing", "/routing_test", "/smsc_links",
"/smsc_subscribers", "/hlr_links",
    # ... remaining ordered paths
  ]
```

Configuration Parameters to Customize

For a complete reference of all configuration parameters, see the Configuration Reference.

Parameter	Type	Default	Des
`smsc_api_base_url`	String	*Required*	**OmniM** backend endpoin
`smsc_name`	String	`"{hostname}_SMSc"`	Your SM for regis
`smsc_service_center_gt_address`	String	*Required*	Service Global T
`auto_flush_enabled`	Boolean	`true`	Enable queue p
`auto_flush_interval`	Integer	`10_000`	Queue p interval millisec
`auto_flush_dest_smsc`	String	`nil`	Destina name fi flush (`n` all pend messag
`auto_flush_tps`	Integer	`10`	Messag rate (transac
`local_ip`	Tuple	*Required*	Your SM IP addre
`local_port`	Integer	`2905`	Local SC
`remote_ip`	Tuple	*Required*	STP IP a SS7 con

Parameter	Type	Default	Des
`remote_port`	Integer	`2905`	Remote
`routing_context`	Integer	`1`	M3UA r context

What Happens When SMSc Mode is Enabled

When `smsc_mode_enabled: true` and `map_client_enabled: true`, the web UI will show:

- **SS7 Events** - Event logging
- **SS7 Client** - MAP operation testing
- **Peers** - Connection status (M3UA/SCTP peers)
- **Routing** - Route table management (STP enabled)
- **Routing Test** - Route testing (STP enabled)
- **SMSc Links** - SMSc API status + SMS queue management ← *SMSc-specific*
- **Resources** - System monitoring
- **Configuration** - Config viewer

The **HLR Links** tab will be hidden.

Important Notes

- SMSc mode requires `map_client_enabled: true` for routing capabilities
- **OmniMessage Backend**: The OmniMessage API backend must be accessible at the configured `smsc_api_base_url`
- **Frontend Registration**: The system automatically registers with OmniMessage every **~85 seconds** (`@registration_interval` = 85,000 ms) via the `SMS.FrontendRegistry` module
- **API Request Timeout**: All OmniMessage API requests have a **hardcoded 5-second timeout**

- **MAP Request Timeout**: SRI-for-SM has a 10-second dialog timeout. MT-ForwardSM has a **30-second dialog timeout** to accommodate slow VLR responses on 2G/CS networks
- Auto-flush automatically processes the SMS queue in the background
- M3UA connection to STP is required for sending/receiving MAP SMS operations
- After changing modes, you must restart the application for changes to take effect
- **Web UI**: See the Web UI Guide for information on using the web interface
- **API Access**: See the API Guide for REST API documentation and Swagger UI access

---

HTTP API Configuration

OmniMessage Backend Setup

OmniSS7 communicates with OmniMessage via HTTPS REST API to manage message delivery, track subscriber state, and register as an active frontend:

```
config :omniss7,
  # OmniMessage API base URL
  smsc_api_base_url: "https://10.5.198.200:8443",
  # SMSC name identifier for registration (defaults to
hostname_SMSc if empty)
  smsc_name: "omni-smsc01",
  # Service Center GT Address for SMS operations
  smsc_service_center_gt_address: "5551234567"
```

Configuration Parameters:

Parameter	Type	Required	Default
`smsc_api_base_url`	String	Yes	`"https://localhc`
`smsc_name`	String	No	`""` (uses `"{hostname}_SMSc"`
`smsc_service_center_gt_address`	String	No	`"5551234567"`

Frontend Registration

The system automatically registers itself with OmniMessage on startup and **re-registers every ~85 seconds** via the `SMS.FrontendRegistry` module. This allows OmniMessage to:

- Track active frontends for load balancing
- Monitor uptime and health status

- Collect configuration information
- Manage distributed SMS routing across multiple frontends

Implementation Details:

- **Registration Interval**: ~85 seconds (`@registration_interval 85_000`, hardcoded)
- **Process**: Started automatically when `smsc_mode_enabled: true`

Registration Payload:

```
{
  "frontend_name": "omni-smsc01",
  "configuration": "{...}",
  "frontend_type": "SS7",
  "hostname": "smsc-server01",
  "uptime_seconds": 12345
}
```

Note: The frontend name is taken from the `smsc_name` configuration parameter. If not set, it defaults to `"{hostname}_SMSc"`.

OmniMessage API Communication

When OmniSS7 receives MAP operations from the SS7 network or processes the message queue, it communicates with OmniMessage to:

- **Register as an active frontend** and report health status
- **Submit mobile-originated (MO) messages** received from subscribers
- **Retrieve mobile-terminated (MT) messages** from the queue for delivery
- **Update delivery status** with success/failure reports
- **Query routing information** for message forwarding

Endpoint	Method	Purpose	Request
`/api/frontends`	POST	Register frontend instance	`{"frontend_nar "configuratior "frontend_type "hostname": ". "uptime_second`
`/api/messages_raw`	POST	Insert new SMS message (raw TPDU)	`{"rp_originat "...", "rp_destinatic "...", "direct "message_body" "send_time": " "smsc_node_nam "source_smsc": "source_type":`
`/api/messages`	GET	Get message queue	Header: `smsc:`
`/api/messages/{id}`	PATCH	Mark message as delivered	`{"deliver_time "dest_smsc": "`
`/api/messages/{id}`	PUT	Update message status	`{"dest_smsc":`
`/api/locations`	POST	Insert/update subscriber location	`{"msisdn": ". "...", "locati "ims_capable": "csfb": false, "...", "user_a "...", "ran_lo "...", "imei": "registered":`
`/api/events`	POST	Add event tracking	`{"message_id" "name": "...", "description":`

Endpoint	Method	Purpose	Request
`/api/status`	GET	Health check (path is configurable via `smsc_api_status_path`, default `/api/status`; polled by the SMSc Links page, not by `SMSc.APIClient`)	-

API Response Format

All API responses use JSON format with the following conventions:

- **Success responses**: HTTP 200-201 with JSON body containing result data
- **Error responses**: HTTP 4xx/5xx with error details in response body
- **Timestamps**: ISO 8601 format (e.g., `"2025-10-21T12:34:56Z"`)
- **Message IDs**: Integer or string identifiers

API Client Modules

The SMS system consists of three main modules:

1. SMSc.APIClient

Main API client module providing all HTTP API communication with OmniMessage:

- `frontend_register/4` - Register frontend with OmniMessage
- `insert_message/1` - Insert raw SMS message (takes a single message map; posts to `/api/messages_raw`)
- `insert_location/9` - Insert/update subscriber location data
- `get_message_queue/2` - Retrieve pending messages from queue
- `mark_dest_smsc/3` - Mark message as delivered or failed
- `add_event/3` - Add event tracking for messages
- `flush_queue/2` - Process pending messages (SRI-for-SM + MT-forwardSM)

- `auto_flush/2` - Continuous queue processing loop

2. SMS.FrontendRegistry

Handles periodic frontend registration with the backend:

- Automatically registers on startup
- Re-registers every ~85 seconds
- Uses `smsc_name` from config (falls back to hostname)
- Collects system configuration and uptime information

3. SMS.Utils

Utility functions for SMS operations:

- `generate_tp_scts/0` - Generate SMS timestamp in TPDU format

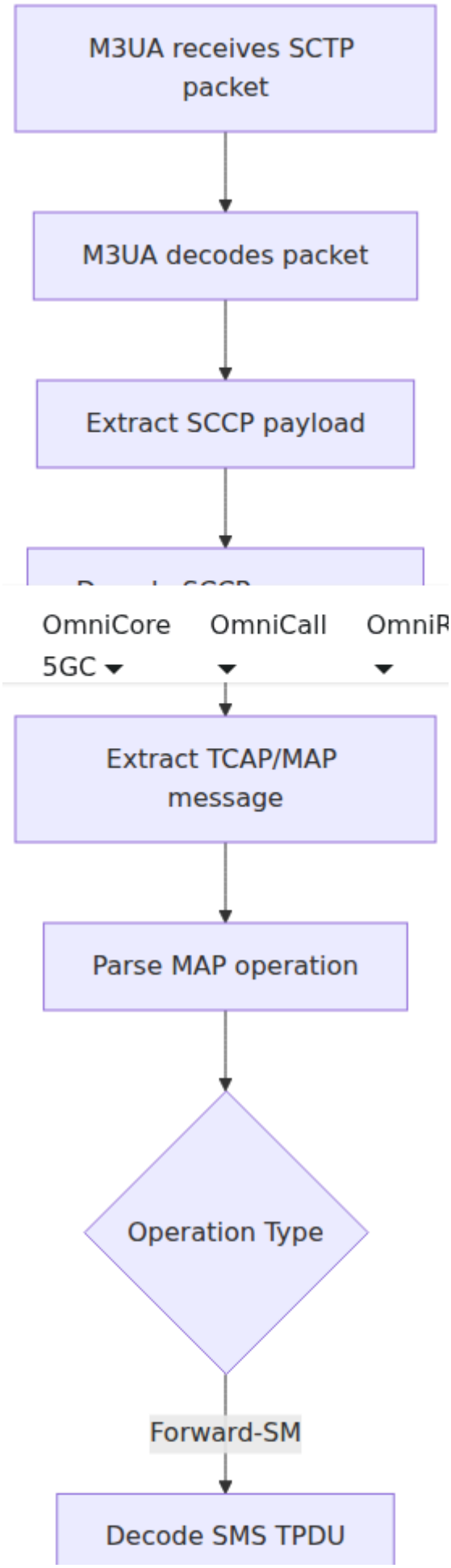
SMS Message Flows

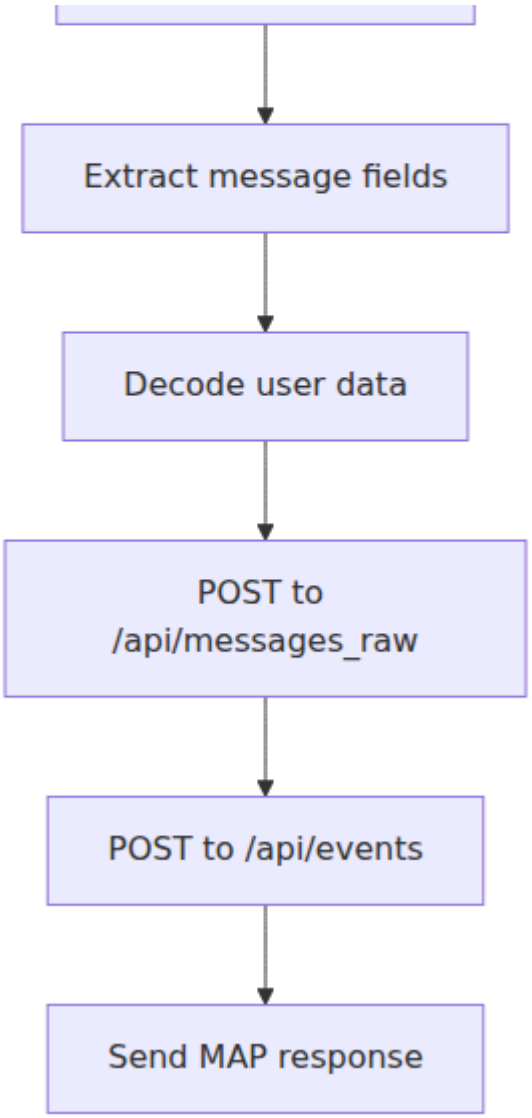
Incoming SMS Flow (Mobile-Originated)


M3UA receives SCTP packet
M3UA decodes packet
Extract SCCP payload
Decode SCCP message
Extract TCAP/MAP message
Parse MAP operation
Operation Type
Forward-SM
Decode SMS TPDU

Outgoing SMS Flow (Mobile-Terminated)

When an HSS/HLR API is configured (`hlr_api_base_url`), the IP-SM-GW can deliver MT-SMS to **on-net subscribers** without an SRI-for-SM round-trip by querying the HSS API directly for the subscriber's real IMSI and serving VLR. A **VLR cache** in the Subscriber Tracker avoids repeated HSS lookups for the same subscriber.


M3UA receives SCTP packet
M3UA decodes packet
Extract SCCP payload
OmniCore 5GC
OmniCall
Extract TCAP/MAP message
Parse MAP operation
Operation Type
Forward-SM
Decode SMS TPDU



Key Steps Explained:

- **VLR Cache Check**: Before querying the HSS API, the system checks the Subscriber Tracker's VLR cache for a valid entry (IMSI + VLR address within TTL). A cache hit skips the HSS API entirely. See VLR Address Caching for details.
- **HSS API Direct Delivery (on-net)**: When `hlr_api_base_url` is configured, the IP-SM-GW queries the HSS API with the destination MSISDN to get the subscriber's **real IMSI** and **serving VLR address**. If both are available, the MT-ForwardSM is sent directly to the VLR — bypassing SRI-for-SM entirely.
- **Absent Subscriber (on-net)**: If the HSS API finds the subscriber but no serving VLR is assigned, the message is marked as failed with an expired

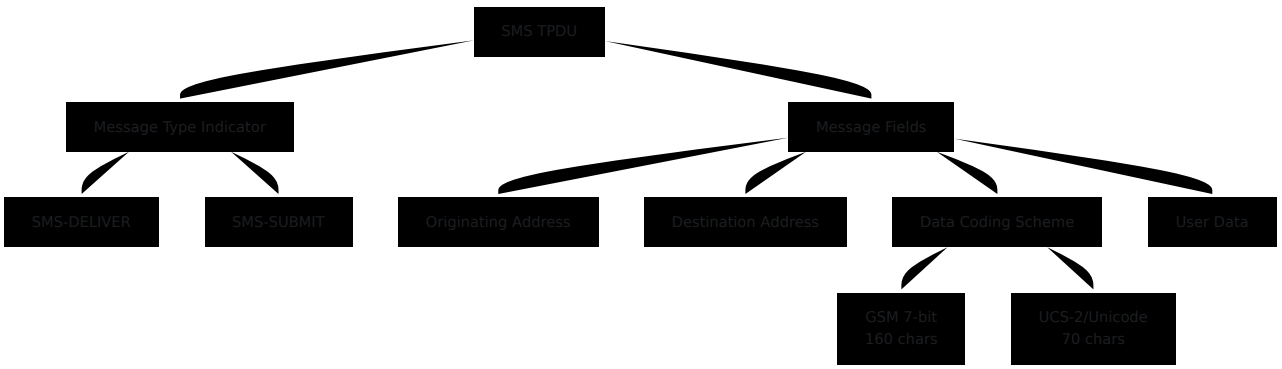
location entry. Delivery retries when the HLR sends an alertServiceCenter after the subscriber re-registers.

- **SRI-for-SM Fallback (off-net)**: When the HSS API is not configured or the subscriber is not found in the HSS, the standard SRI-for-SM flow is used. The HLR responds with a synthetic IMSI and network node number. See SRI-for-SM in HLR Guide.

- **VLR Clear on Failure**: When MT-ForwardSM delivery fails (timeout, SCCP error, etc.), the cached VLR address is cleared so the next attempt re-queries the HSS API for a fresh VLR.

- **MT-ForwardSM Timeout**: The MT-ForwardSM dialog has a **30-second timeout** to accommodate slower 2G/CS VLR responses.

SMS TPDU Structure



Alert Service Center Handling

The SMSc can receive **alertServiceCenter** messages from the HLR to track subscriber reachability status.

For information on how the HLR sends alertServiceCenter messages, see Alert Service Center Integration in HLR Guide.

What is alertServiceCenter?

When a subscriber performs an UpdateLocation at the HLR (i.e., registers with a new VLR/MSC), the HLR can notify SMSc systems that the subscriber is now reachable by sending an **alertServiceCenter** (MAP opcode 64) message.

Configuration

The location expiry time is configured in the HLR:

```
config :omniss7,
  # Location expiry time when SMSc receives alertServiceCenter
(default: 48 hours)
  hlr_alert_location_expiry_seconds: 172800
```

Behavior

When the SMSc receives an alertServiceCenter message:

1. **Decode MSISDN**: Extract the subscriber's MSISDN from the message (TBCD format)
2. **Strip TON/NPI prefix**: Remove common prefixes like "19", "11", "91" (e.g., "19123123213" → "123123213")
3. **Resolve IMSI**: If `hlr_api_base_url` is configured, query the HSS API to resolve the subscriber's **real IMSI**. If the HSS lookup fails or no API is configured, fall back to generating a synthetic IMSI using the same mapping as SRI-for-SM. Using the real IMSI is critical for on-net MT delivery to 2G/CS subscribers — the MSC/VLR requires the correct IMSI in the MT-ForwardSM.
4. **POST to /api/locations**: Update location database with:
   - `msisdn`: Subscriber's phone number (cleaned)
   - `imsi`: Synthetic IMSI
   - `location`: SMSc name (e.g., "ipsmgw")

    - `expires`: Current time + `hlr_alert_location_expiry_seconds`
    - `csfb`: true (subscriber reachable via Circuit-Switched Fallback)
    - `ims_capable`: false (this is 2G/3G CS registration, not IMS/VoLTE)
    - `user_agent`: HLR GT that sent the alert (for tracking)
    - `ran_location`: "SS7"
5. **Track in SMSc Subscriber Tracker**: Record the subscriber with HLR GT, status=active, message counters at 0
6. **Send ACK**: Reply to HLR with alertServiceCenter acknowledgment

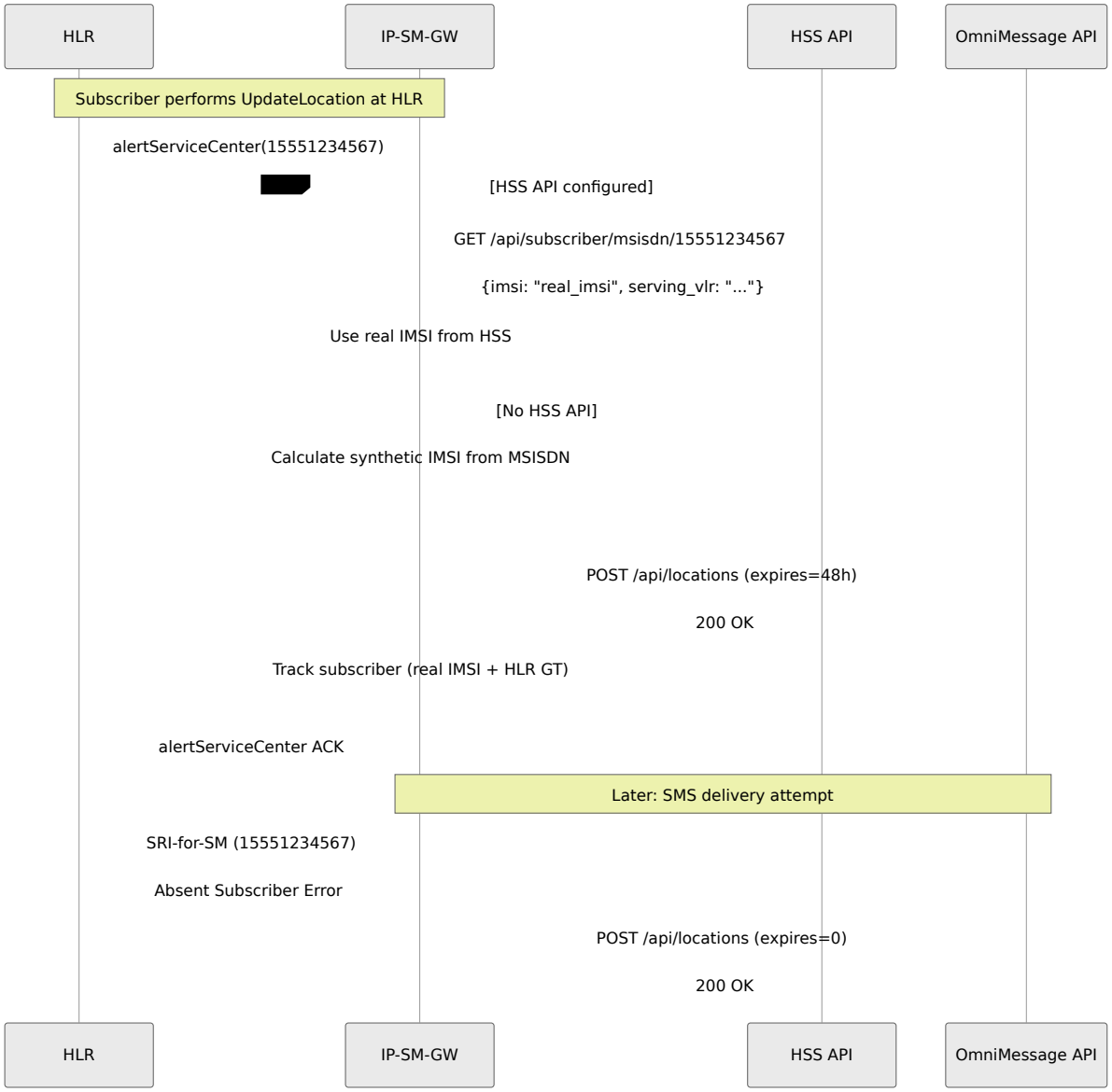
Absent Subscriber Handling

When the SMSc attempts to deliver a message and receives an "absent subscriber" error during SRI-for-SM (for more on SRI-for-SM, see SRI-for-SM in HLR Guide):

1. **Detect absence**: SRI-for-SM returns `absentSubscriberDiagnosticSM` error
2. **Expire location**: POST to /api/locations with `expires=0` to mark subscriber as unreachable
3. **User agent**: Set to "SS7_AbsentSubscriber" to identify the source
4. **Update tracker**: Mark subscriber as `failed` in SMSc Subscriber Tracker

This ensures the location database and tracker accurately reflect subscriber reachability status.

Flow Diagram



API Endpoint

POST /api/locations

```
{
  "msisdn": "15551234567",
  "imsi": "001010123456789",
  "location": "ipsmgw",
  "ims_capable": false,
  "csfb": true,
  "expires": "2025-11-01T12:00:00Z",
  "user_agent": "15551111111",
  "ran_location": "SS7",
  "imei": "",
  "registered": "2025-10-30T12:00:00Z"
}
```


Note: The `user_agent` field contains the HLR GT that sent the alertServiceCenter, allowing the SMSc to track which HLR is providing location updates.

For absent subscribers, `expires` is set to current time (immediate expiry).

Loop Prevention

The SMSc implements **automatic loop prevention** to avoid infinite message routing loops when messages originate from SS7 networks, while still allowing legitimate MO→MT delivery between on-net subscribers.

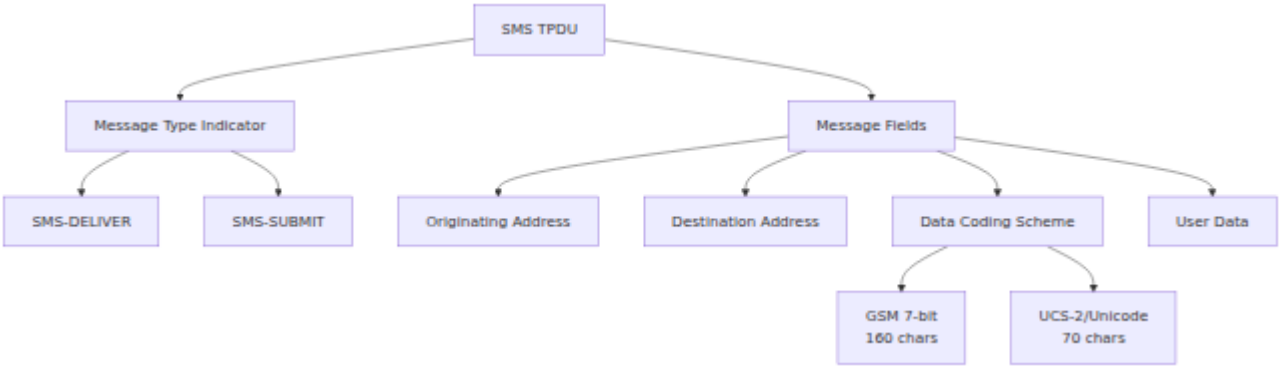
Why Loop Prevention is Important

When the SMSc receives mobile-originated (MO) SMS messages from the SS7 network, it inserts them into the message queue with a `source_smsc` field identifying their origin (e.g., `"SS7_MO_15551234567"`). Without loop prevention, these messages could be:

1. Received from SS7 network → Queued with `source_smsc` containing "SS7"
2. Retrieved from queue → Processed for delivery
3. Sent back to SS7 network → Creating a loop

How It Works

The loop prevention logic distinguishes between **on-net** and **off-net** destinations. Messages from SS7 are only blocked when the destination subscriber is **not on-net** — this prevents transit-routing loops while allowing legitimate MO→MT delivery between subscribers on the same network (e.g., MSC-to-MSC via the IP-SM-GW).



On-Net Detection

When a message originates from SS7, the system checks whether the destination subscriber is on-net before applying loop prevention:

1. **VLR Cache** (fastest): Check if the subscriber has a cached VLR address with valid IMSI in the Subscriber Tracker
2. **HSS API** (authoritative): If `hlr_api_base_url` is configured and the cache misses, query the HSS API by MSISDN. If the subscriber exists in the HSS, they are on-net.

If neither check finds the subscriber, the destination is considered **off-net** and the message is blocked.

Implementation

When processing messages from the queue, the SMSc checks the `source_smsc` field:

- **If `source_smsc` contains "SS7" and destination is off-net**:
    - Message is skipped

    - Event added: `"Loop Prevention"` with description noting the destination is off-net
    - Message marked as failed via PUT request
    - Logged with warning level
- **If `source_smsc` contains "SS7" and destination is on-net**:
    - Message processed normally — this is a legitimate MO→MT flow
    - Delivery proceeds via the HSS API / VLR cache path
- **If `source_smsc` does not contain "SS7"**:
    - Message processed normally (e.g., messages from the web API, SMPP, or other non-SS7 sources)

Source SMSC Values

Messages can have various `source_smsc` values:

Source	Example Value	Action
SS7 Network (MO-FSM), off-net dest	`"SS7_MO_15551234567"`	**Skipped** - Loop prevention
SS7 Network (MO-FSM), on-net dest	`"SS7_MO_15551234567"`	**Allowed** - On-net MO→MT delivery
External API/SMPP	`"ipsmgw"` or `"api_gateway"`	Processed normally
Other SMSc	`"smsc-node-01"`	Processed normally

Event Tracking

When a message is skipped due to loop prevention, an event is recorded:

```
{
  "message_id": 12345,
  "name": "Loop Prevention",
  "description": "Message skipped - source_smsc
'SS7_MO_15551234567' contains 'SS7' and destination is off-net,
preventing message loop"
}
```

This event is visible in:

- **Web UI**: SS7 Events page (`/events`)
- **Database**: `events` table via API
- **Logs**: Warning level log entries

Configuration

Loop prevention is **always enabled** and cannot be disabled. This is a critical safety feature to prevent network disruption from message loops. The on-net bypass requires both `smsc_mode_enabled: true` and either the VLR cache or `hlr_api_base_url` to be configured.

Example Scenarios

Scenario 1: MO-SMS from SS7 to off-net subscriber (blocked)

```
1. Mobile phone → MSC/VLR → IP-SM-GW (via MO-ForwardSM)
2. IP-SM-GW receives MO-FSM, inserts to queue: source_smsc =
"SS7_MO_2471900"
3. Auto-flush retrieves message from queue
4. Destination not in VLR cache or HSS → off-net
5. IP-SM-GW detects "SS7" in source_smsc + off-net dest → SKIP
6. Event logged: "Loop Prevention"
7. No SRI-for-SM or MT-ForwardSM sent (loop prevented)
```

Scenario 2: MO-SMS from SS7 to on-net subscriber (allowed)

```
1. Mobile phone → MSC/VLR → IP-SM-GW (via MO-ForwardSM)
2. IP-SM-GW receives MO-FSM, inserts to queue: source_smsc =
"SS7_MO_2471900"
3. Auto-flush retrieves message from queue
4. Destination found in HSS API → on-net subscriber
5. IP-SM-GW allows delivery despite SS7 origin
6. HSS returns real IMSI + VLR → MT-ForwardSM sent directly to VLR
```

SMSc Subscriber Tracking

The SMSc includes a **Subscriber Tracker** GenServer that maintains real-time state for subscribers based on alertServiceCenter messages and message delivery attempts.

Purpose

The tracker provides:

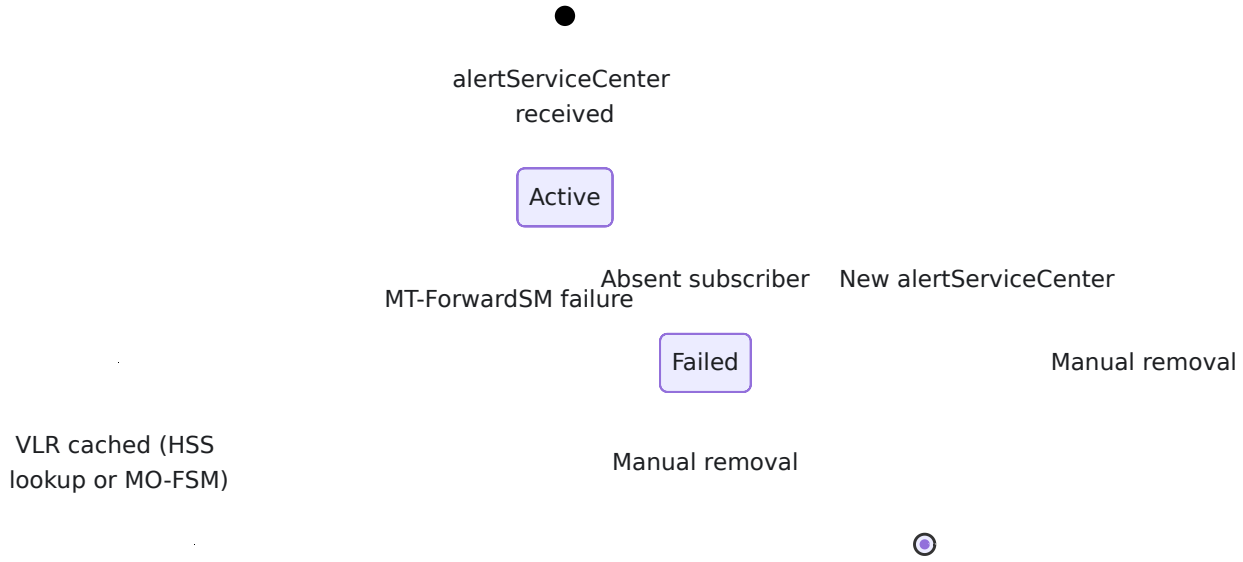
- **Reachability monitoring**: Which subscribers are currently reachable
- **HLR tracking**: Which HLR sent the alertServiceCenter for each subscriber
- **Message counters**: Number of messages sent/received per subscriber
- **Failure tracking**: Mark subscribers as failed when delivery attempts fail
- **Web UI visibility**: Real-time dashboard showing all tracked subscribers

Tracked Information

For each subscriber, the tracker stores:

Field	Description	Example
`msisdn`	Subscriber's phone number (key)	"15551234567"
`imsi`	Subscriber's IMSI (real from HSS if available)	"001010123456789"
`hlr_gt`	HLR GT that sent alertServiceCenter	"15551111111"
`messages_sent`	Count of MT-FSM messages sent	5
`messages_received`	Count of MO-FSM messages received	2
`status`	`:active` or `:failed`	`:active`
`updated_at`	Unix timestamp of last update	1730246400
`vlr_address`	Cached serving VLR GT address	"14155550100"
`vlr_cached_at`	Unix timestamp when VLR was last cached	1730246400

State Transitions



Behavior

When alertServiceCenter is received:

- Create or update subscriber entry
- Set `status = :active`
- Record HLR GT
- Resolve real IMSI from HSS API (if configured), otherwise use synthetic IMSI
- Preserve existing VLR cache and message counters

When HSS API returns subscriber with VLR:

- Cache the VLR address and real IMSI in the tracker
- Subsequent deliveries use the cached VLR without re-querying the HSS

When MO-ForwardSM is received:

- Cache the SCCP calling GT as the subscriber's VLR address (the calling GT on an MO-ForwardSM is the VLR that relayed the message)
- Creates a tracker entry if one does not exist

When MT-ForwardSM delivery fails:

- Clear the cached VLR address so the next delivery attempt re-queries the HSS API

When SRI-for-SM succeeds:

- Increment `messages_sent` counter
- Update `updated_at` timestamp

When SRI-for-SM fails:

- Set `status = :failed`
- Keep in tracker for monitoring

When subscriber is removed:

- Delete from ETS table
- No longer appears in Web UI

Web UI - SMSc Subscribers Page

Path: `/smsc_subscribers` **Auto-refresh**: Every 2 seconds

Note: This page is only available when running in SMSc mode. After enabling the SMSc configuration in `config/config.exs`, you must restart the application for the route to become available.

The **SMSc Subscribers** page provides real-time monitoring of all tracked subscribers:

Features

1. **Subscriber Table**
    - MSISDN, IMSI, HLR GT, VLR GT (cached serving VLR address)
    - Messages sent/received counters
    - Status badge (Active/Failed) with color coding
    - Last updated timestamp and duration
    - Remove button for individual subscribers

2. **Summary Statistics**
    - Total tracked subscribers
    - Count of active subscribers
    - Count of failed subscribers
    - Number of unique HLRs
3. **Actions**
    - Clear All: Remove all tracked subscribers
    - Remove: Remove individual subscriber

Example View

```
┌──────────────────────────────────────────────────────────────────────────────
│ SMSc Tracked Subscribers                                          Total:
├──────────────────────────────────────────────────────────────────────────────
│ MSISDN          IMSI              HLR GT         VLR GT         Msg
│                                                                   S/
├──────────────────────────────────────────────────────────────────────────────
│ 15551234567     001010123456789   15551111111    14155550100    5/2
│ 15559876543     001010987654321   15551111111    -              0/0
│ 15551112222     001010111222233   15552222222    14155550200    3/1
└──────────────────────────────────────────────────────────────────────────────

Summary:  Total: 3  |  Active: 2  |  Failed: 1  |  Unique HLRs: 2
```

API Functions

The tracker exposes these functions for programmatic access:

```
# Called when alertServiceCenter is received
SMSc.SubscriberTracker.alert_received(msisdn, imsi, hlr_gt)

# Increment message counters
SMSc.SubscriberTracker.message_sent(msisdn)
SMSc.SubscriberTracker.message_received(msisdn)

# Mark as failed (SRI-for-SM failure)
SMSc.SubscriberTracker.mark_failed(msisdn)

# Remove from tracking
SMSc.SubscriberTracker.remove_subscriber(msisdn)

# Query functions
SMSc.SubscriberTracker.get_active_subscribers()
SMSc.SubscriberTracker.get_subscriber(msisdn)
SMSc.SubscriberTracker.count_subscribers()
SMSc.SubscriberTracker.clear_all()
```

Integration

The tracker is automatically integrated with:

- **alertServiceCenter handler**: Calls `alert_received/3` on successful location update
- **SRI-for-SM handler**: Increments `messages_sent` on successful routing
- **Absent subscriber handler**: Calls `mark_failed/1` when subscriber is absent
- **Unknown subscriber errors**: Calls `mark_failed/1` when SRI-for-SM fails

---

VLR Address Caching

When operating as an IP-SM-GW with an HSS API configured, the Subscriber Tracker caches VLR addresses to avoid querying the HSS on every MT delivery attempt.

How It Works

The VLR cache is populated from two sources:

1. **HSS API lookup**: When the delivery pipeline queries the HSS API and gets a subscriber with a serving VLR, the IMSI and VLR address are cached in the Subscriber Tracker.
2. **MO-ForwardSM**: When an MO-ForwardSM arrives from a subscriber, the SCCP calling party GT (which is the VLR/MSC that relayed the message) is cached as the subscriber's VLR address.

On subsequent MT delivery attempts, the cache is checked first. If a valid entry exists (both IMSI and VLR present, within TTL), the HSS API is skipped entirely.

Cache Invalidation

The VLR cache is cleared when:

- **MT-ForwardSM delivery fails** (timeout, SCCP error, VLR rejection) — the cached VLR may be stale
- **TTL expires** — configurable via `vlr_cache_ttl_seconds`, entries older than the TTL are treated as cache misses

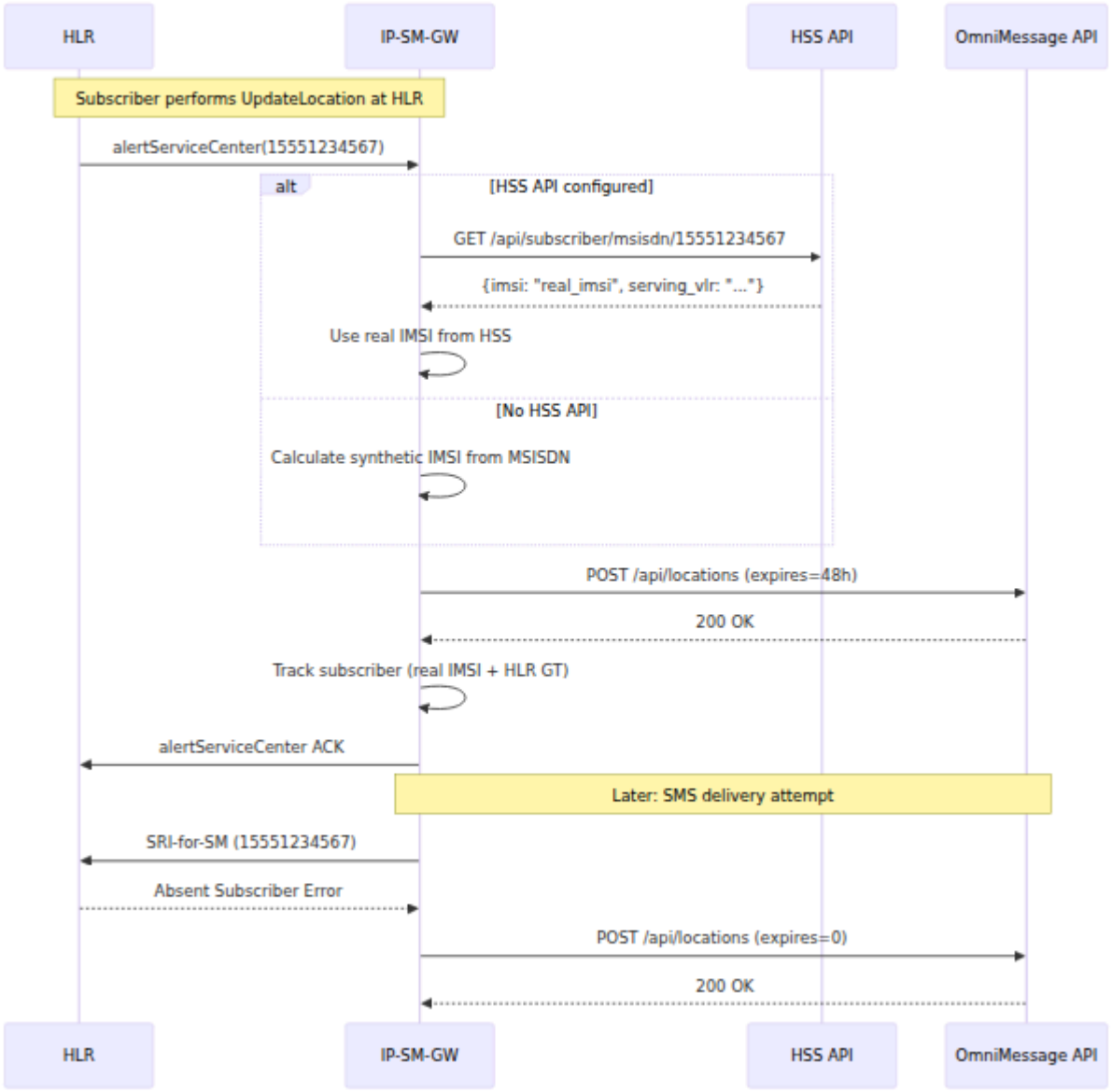
The cache is **not** cleared on alertServiceCenter — the existing VLR entry is preserved since the subscriber may still be reachable at the same VLR.

Configuration

```
config :omniss7,
  # HSS/HLR API endpoint (required for VLR caching and on-net
delivery)
  hlr_api_base_url: "https://10.179.2.140:8443",

  # VLR cache TTL in seconds (default: 3600 = 1 hour)
  vlr_cache_ttl_seconds: 3600
```

Parameter	Type	Required	Default	Description
`hlr_api_base_url`	String	No	`nil`	HSS/HLR API base URL. When set, enables on-net subscriber lookup, VLR caching, and direct MT delivery. When `nil`, all deliveries use the SRI-for-SM flow.
`vlr_cache_ttl_seconds`	Integer	No	`3600`	Maximum age in seconds for a cached VLR entry to be considered valid. After this period, the next delivery attempt re-queries the HSS API. Lower values increase HSS API traffic but improve accuracy for mobile subscribers.

Sequence Diagram



Auto-Flush SMS Queue

The **Auto-Flush** service automatically processes pending SMS messages.

For configuration parameter reference, see Auto-Flush Configuration in Configuration Reference.

Configuration

```
config :omniss7,
  auto_flush_enabled: true,           # Enable/disable auto-flush
  auto_flush_interval: 10_000,        # Poll interval in
milliseconds
  auto_flush_dest_smsc: nil,          # Filter: nil = all
  auto_flush_tps: 10                  # Max transactions per
second
```

How It Works

1. **Polling**: Every `auto_flush_interval` milliseconds, queries API for pending messages
2. **Filtering**: Optionally filter by `auto_flush_dest_smsc`
3. **Rate Limiting**: Process up to `auto_flush_tps` messages per cycle
4. **Delivery**: For each message:
   - Send **SRI-for-SM** (Send Routing Info for Short Message) to HLR to get routing info
     - The HLR returns a synthetic IMSI calculated from the MSISDN
     - The HLR returns the SMSC GT address where MT-ForwardSM should be sent
     - See SRI-for-SM Details in HLR Guide for complete documentation
   - On success, send **MT-forwardSM** to MSC/VLR
   - Update message status via API (delivered/failed)
   - Add event tracking via API

> **Technical Deep Dive**: For a complete explanation of how SRI-for-SM works, including MSISDN to IMSI mapping, service center GT address configuration, and the privacy-preserving synthetic IMSI generation, see the SRI-for-SM section in the HLR Configuration Guide.

SMSc Metrics

OmniSS7 exposes Prometheus metrics. SMS activity is tracked through the shared MAP metrics — there are no SMSc-specific queue/delivery counters. The metrics relevant to SMS operations are:

Metric	Type	Labels	Relevar SM
`map_requests_total`	Counter	`operation`	Counts MA operation relevant operation include `sri_for_` `forward_` `MTforwar` and `prn`.
`map_request_duration_milliseconds`	Histogram	`operation`	Duration request/re cycles, inc SRI-for-SM MT-Forwa

The full set of declared metrics (not all SMS-specific) is: `map_requests_total`, `cap_requests_total`, `ussd_requests_total`, `ussd_active_sessions`, `map_pending_requests`, `omniss7_license_status`, and `map_request_duration_milliseconds`.

Example Queries:

```
# Rate of MT-ForwardSM operations over the last 5 minutes
rate(map_requests_total{operation="MTforward_sm"}[5m])

# Rate of SRI-for-SM operations over the last 5 minutes
rate(map_requests_total{operation="sri_for_sm"}[5m])

# Average MT-ForwardSM duration (last 5 minutes)
rate(map_request_duration_milliseconds_sum{operation="MTforward_sm"}
[5m]) /
rate(map_request_duration_milliseconds_count{operation="MTforward_sm"
[5m])
```

Troubleshooting SMSc

Issue: Messages Not Delivering

Checks:

1. Verify auto-flush is enabled
2. Check database connection
3. Monitor logs for errors
4. Verify M3UA connection is ACTIVE
5. Check TPS limits

Issue: High Queue Depth

Possible Causes:

- TPS limit too low
- HLR timeout issues
- Network connectivity problems
- Invalid destination numbers

Solutions:

- Increase `auto_flush_tps`
- Check HLR availability
- Review failed message logs

MT-forwardSM API

Send SMS via API

API Endpoint: `POST /api/MT-forwardSM`

All four fields are required. Field names are snake_case (note `destination_service_centre` uses the British spelling). `smsPDU` is the hex-encoded SMS TPDU.

Request:

```
{
  "imsi": "234509876543210",
  "destination_service_centre": "447999555111",
  "originating_service_center": "447999123456",
  "smsPDU":
"040B917477218345F60000157030185714 0C0BD4F29C0E9281C4E1F11A"
}
```

Response:

On success (HTTP 200), the response contains a single `result` key holding the formatted TCAP result from the MT-ForwardSM dialog (there is no `message_id`, and the value is not the literal string `"success"`):

```
{
  "result": "<formatted TCAP result>"
}
```

If the MAP dialog times out, the endpoint returns HTTP 504 with `{"error": "timeout"}`. A malformed/missing-field request returns HTTP 400 with `{"error": "invalid request"}`.

Related Documentation

OmniSS7 Documentation:

- ← Back to Main Documentation
- HLR Configuration Guide - HLR mode setup and operations
  - **SRI-for-SM Technical Details** - Complete documentation on MSISDN to IMSI mapping and service center configuration
- Common Features Guide - Web UI, API, Monitoring
- MAP Client Guide - MAP operations
- Technical Reference - Protocol specifications

OmniMessage Documentation: For message routing configuration, queue management, delivery tracking, rate limiting, and analytics, refer to the **OmniMessage product documentation**. OmniMessage contains all the message routing logic, queue retry algorithms, delivery report handling, and business rules engine.

M3UA & M2PA STP Configuration Guide

← Back to Main Documentation

This guide provides detailed configuration for using OmniSS7 as a **Signaling Transfer Point (STP)**.

Table of Contents

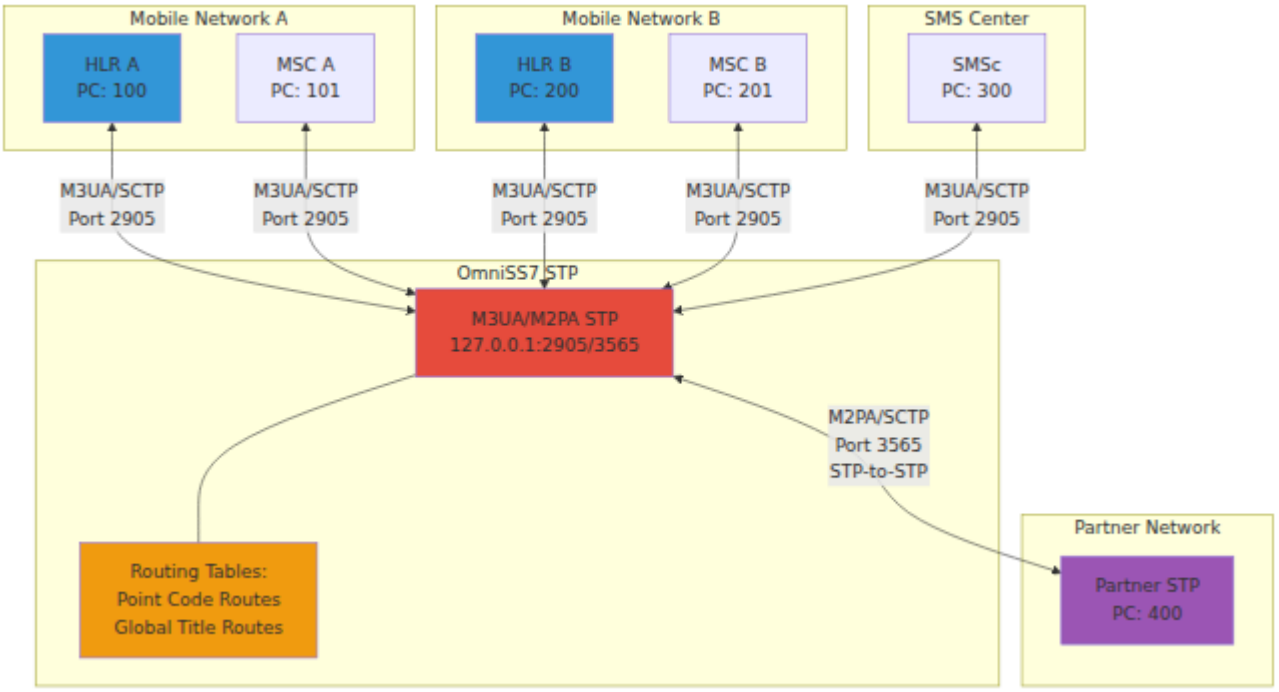
What is a Signaling Transfer Point (STP)?

A **Signaling Transfer Point (STP)** is a critical network element in SS7 and IP-based signaling networks that routes signaling messages between network nodes.

STP Functions

- **Message Routing**: Routes SS7 signaling traffic based on destination Point Code (PC) or Global Title (GT)
- **Protocol Translation**: Bridges traditional SS7 networks with IP-based M3UA/SCTP networks
- **Load Distribution**: Distributes traffic across multiple destinations using priority-based routing
- **Network Gateway**: Connects different signaling networks and service providers
- **Topology Hiding**: Can rewrite addresses to hide internal network topology

STP Network Diagram



STP Network Roles Explained

ASP (Application Server Process)

- **Role**: Client connecting to a remote SGP/STP
- **Direction**: Outbound connection

- **Use Case**: Your STP connects to a partner network's STP

SGP (Signaling Gateway Process)

- **Role**: Server accepting connections from ASPs
- **Direction**: Inbound connection
- **Use Case**: Partner networks connect to your STP

AS (Application Server)

- **Definition**: Logical grouping of one or more ASPs
- **Purpose**: Provides redundancy and load sharing
- **Use Case**: Multiple ASPs serving the same destination

Interfacing with TDM Networks via Signaling Gateways

OmniSS7 is an IP-based signaling platform that uses SIGTRAN protocols (M3UA/M2PA over SCTP). To exchange signaling with legacy **TDM-based SS7 networks**, you need a **Signaling Gateway (SGW)** to bridge the two worlds.

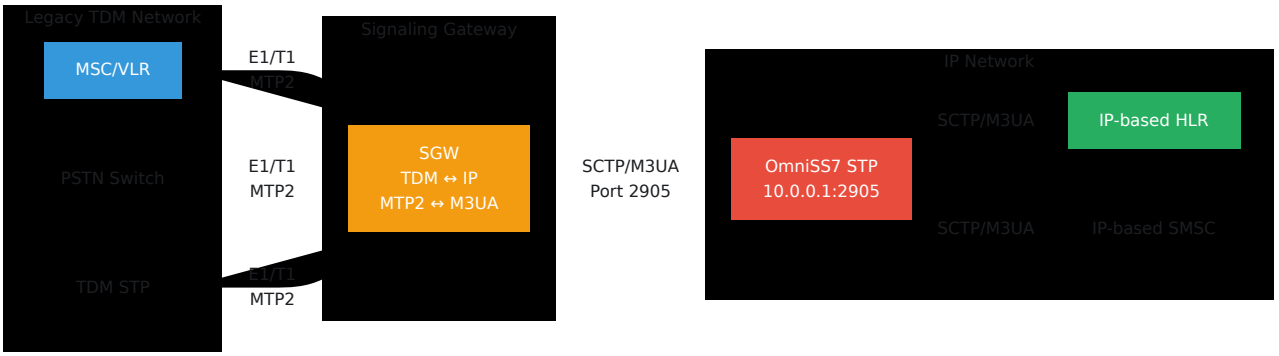
What is a Signaling Gateway?

A **Signaling Gateway (SGW)** is a network element that converts SS7 signaling between:

- **TDM side**: Traditional SS7 using MTP1/MTP2/MTP3 over E1/T1 links
- **IP side**: SIGTRAN using M3UA or M2PA over SCTP/IP

The SGW acts as a protocol translator, allowing IP-based applications like OmniSS7 to communicate with legacy TDM equipment (switches, STPs, HLRs) that don't support IP signaling.

TDM to IP Architecture



Protocol Stack Comparison

Layer	TDM SS7	Signaling Gateway	IP (SIGTRAN)
User Part	SCCP/TCAP/MAP	← Transparent →	SCCP/TCAP/MAP
Network	MTP3	Conversion	M3UA/M2PA
Link	MTP2	Termination	SCTP
Physical	MTP1 (E1/T1)	Termination	IP/Ethernet

The SGW terminates MTP1/MTP2 on the TDM side and presents the MTP3 user data over M3UA or M2PA on the IP side. The upper layers (SCCP, TCAP, MAP, CAP) pass through transparently.

Connecting OmniSS7 to a Signaling Gateway

OmniSS7 connects to a Signaling Gateway as an **ASP (Application Server Process)**, while the SGW acts as an **SGP (Signaling Gateway Process)**.

Configuration Example

```
config :omniss7,
  # Connect to Signaling Gateway as ASP
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :sgw_connection,
    # Local endpoint (OmniSS7)
    local_ip: {10, 0, 0, 1},
    local_port: 0,                    # Dynamic local port
    # Signaling Gateway endpoint
    remote_ip: {10, 0, 0, 100},       # SGW IP address
    remote_port: 2905,                # SGW M3UA port
    routing_context: 1                # As assigned by SGW
  },

  # Or configure as a peer for STP routing
  peers: [
    %{
      peer_id: 1,
      name: "TDM_Gateway",
      role: :client,                  # OmniSS7 initiates
connection
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 100},     # SGW IP address
      remote_port: 2905,
      routing_context: 1,
      point_code: 100,                # Point code range behind
SGW
      network_indicator: :international
    }
  ],

  # Route TDM network point codes through the gateway
  m3ua_routes: [
    %{
      dest_pc: 100,                   # TDM MSC point code
      peer_id: 1,                     # Route via SGW peer
      priority: 1,
      network_indicator: :international
    },
    %{
      dest_pc: 200,                   # TDM HLR point code
```

```
      peer_id: 1,
      priority: 1,
      network_indicator: :international
    }
  ]
```

SGW Configuration Considerations

When configuring your Signaling Gateway to work with OmniSS7:

Parameter	Description	Typical Value
Remote IP	OmniSS7's IP address	Your server IP
Remote Port	OmniSS7's SCTP port	2905
Routing Context	AS identifier at the SGW	Assigned by SGW admin
Point Codes	TDM network point codes accessible via SGW	Network-specific
Traffic Mode	ASP traffic handling mode	Override or Loadshare

Deployment Scenarios

Scenario 1: IP Application Accessing TDM Network

OmniSS7 sends MAP queries (SRI-SM, PRN) to TDM-based HLRs via the SGW:

```
OmniSS7 (ASP) → SGW (SGP) → TDM Network → HLR
     M3UA          MTP2           MTP3
```

Scenario 2: OmniSS7 as STP Between IP and TDM

OmniSS7 routes traffic between IP-based network elements and TDM networks:

```
IP SMSC → OmniSS7 STP → SGW → TDM HLR
              ↓
          IP-based HLR
```

Scenario 3: Dual SGW for Redundancy

Connect to two Signaling Gateways for high availability:

```
peers: [
  %{
    peer_id: 1,
    name: "SGW_Primary",
    role: :client,
    remote_ip: {10, 0, 0, 100},
    remote_port: 2905,
    point_code: 100,
    # ... other config
  },
  %{
    peer_id: 2,
    name: "SGW_Backup",
    role: :client,
    remote_ip: {10, 0, 0, 101},
    remote_port: 2905,
    point_code: 100,
    # ... other config
  }
],

m3ua_routes: [
  # Primary route via SGW_Primary
  %{dest_pc: 100, peer_id: 1, priority: 1, network_indicator:
:international},
  # Backup route via SGW_Backup
  %{dest_pc: 100, peer_id: 2, priority: 2, network_indicator:
:international}
]
```

Enabling M3UA STP Mode

OmniSS7 can operate in different modes. To use it as an STP, you provide STP routing configuration under the `:omniss7` application key.

Where Configuration Lives

File	Purpose
`config/config.exs`	Default, always-loaded configuration (web UI, API listener on port 8445, logging)
`config/stp_example_runtime.exs`	A complete, ready-to-edit **STP example** (`sctp_handler`, peers, PC routes, GT routes)
`config/runtime.exs`	Per-deployment runtime overrides. In this repository it is a minimal stub; production servers deploy their own

> **Note**: Earlier releases shipped three commented-out mode blocks inside `config/runtime.exs`. That layout no longer exists. The STP example now lives in its own file, `config/stp_example_runtime.exs`.

Switching to STP Mode

To stand up an STP, copy the example configuration into the config that your deployment loads at runtime:

1. **Open** `config/stp_example_runtime.exs` and review the STP example (SCTP handler, peers, PC routes, GT routes).
2. **Copy** the `config :omniss7, ...` block into your deployment's `config/runtime.exs` (or merge the relevant keys).
3. **Customize** the IP addresses, ports, point codes, peers, and routes for your network.

4. **Restart** the application: `iex -S mix`

STP Mode Configuration

The STP example in `config/stp_example_runtime.exs` configures an inbound STP listener (`sctp_handler`), enables GT routing, and defines peers plus PC and GT routes:

```
config :omniss7,
  # SCTP listener (inbound STP server) — see Standalone STP Mode
below
  sctp_handler: %{
    enabled: true,
    local_ip: {127, 0, 0, 1},
    local_port: 2905,
    point_code: 101            # This STP's own point code
  },

  # Enable Global Title routing (in addition to PC routing)
  enable_gt_routing: true,

  # M3UA peers, PC routes, and GT routes follow
  m3ua_peers: [ %{peer_id: 1, name: "STP_West", role: :client,
...} ],
  m3ua_routes: [ %{dest_pc: 100, peer_id: 1, priority: 1,
network_indicator: :international} ],
  m3ua_gt_routes: [ %{gt_prefix: "44", peer_id: 2, priority: 1,
description: "UK numbers"} ]
```

See Configuring M3UA Peers, Point Code Routing, and Global Title Routing below for the full parameter set.

Configuration Parameters to Customize

For a complete reference of all configuration parameters, see the Configuration Reference.

These are the top-level STP keys under `config :omniss7`:

Parameter	Type	Default	Description	Example
`sctp_handler`	Map	*Optional*	Inbound STP listener configuration (see Standalone STP Mode)	`%{enabled: true, local_port: 2905, point_code: 101}`
`enable_gt_routing`	Boolean	`false`	Enable Global Title routing (in addition to PC routing)	`true`
`m3ua_peers`	List	`[]`	M3UA / M2PA peer definitions (see Configuring M3UA Peers)	—
`m3ua_routes`	List	`[]`	Point Code routes (see Point Code Routing)	—
`m3ua_gt_routes`	List	`[]`	Global Title routes (see Global Title Routing)	—

> **Tip**: Use SCTP multihoming by providing a list of IP addresses for `local_ip` and/or `remote_ip` to enable automatic failover. See SCTP Multihoming Guide.

What Happens When STP Mode is Enabled

Once peers and routes are configured, the web UI shows:

- **SS7 Events** - Event logging
- **M3UA** - Connection status
- **Routing** - Route table management ← *STP-specific*
- **Routing Test** - Route testing ← *STP-specific*
- **Resources** - System monitoring
- **Configuration** - Config viewer

Important Notes

- SCTP protocol (IP protocol 132) must be allowed through firewalls
- Default M3UA port is 2905 (industry standard)
- Ensure sufficient system resources for handling routing traffic
- **Routing Persistence**: All routes configured via the Web UI or API are stored in **Mnesia database** and **survive restarts**
- **Configuration Merge**: Routes from `runtime.exs` are loaded at startup and merged with Mnesia routes
- After changing configuration, you must restart the application for changes to take effect
- **Web UI**: See the Web UI Guide for managing routes via the web interface
- **API Access**: See the API Guide for REST API documentation and Swagger UI access

---

Standalone STP Mode

You can run a **standalone M3UA STP server** that listens for incoming connections. This is the mode used by the example in `config/stp_example_runtime.exs`.

Enabling Standalone STP

Add this configuration under `config :omniss7` in your runtime config:

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    local_ip: {127, 0, 0, 1},    # IP address to listen on
    local_port: 2905,             # Port to listen on
    point_code: 100               # This STP's own point code
  }
```

STP Configuration Parameters

Parameter	Type	Default	Description	Example
`enabled`	Boolean	`false`	Enable standalone STP server	`true`
`local_ip`	Tuple	`{127, 0, 0, 1}`	IP address to listen for connections	`{0, 0, 0, 0}`
`local_port`	Integer	`2905`	Port to listen on	`2905`
`point_code`	Integer	*Required*	This STP's own SS7 point code	`100`

When to Use Standalone STP

- **Pure Routing**: When you only need M3UA routing without MAP client functionality
- **Central STP**: To create a central signaling router for multiple network elements
- **Hub Architecture**: Connect multiple HLRs, MSCs, and SMSCs through a central STP

Note: You can enable both `map_client_m3ua` and `sctp_handler` simultaneously if you need both outbound connections and inbound STP functionality.

Routing Table Persistence (Mnesia)

All routing tables (peers, Point Code routes, and Global Title routes) are stored in a **Mnesia database** for persistence.

How Routing Works

1. **Runtime.exs Routes**: Routes defined in `config/runtime.exs` under `peers` (or legacy `m3ua_peers`), `m3ua_routes`, and `m3ua_gt_routes` are loaded at application startup
2. **Web UI Routes**: Routes added via the Web UI Routing page are stored in Mnesia
3. **Route Merge**: On restart, runtime.exs routes are merged with existing Mnesia routes (no duplicates)
4. **Persistence**: All routes configured via Web UI **survive application restarts**

Mnesia Storage Type

Control how routing tables are stored. For more details on database configuration, see Database Parameters in Configuration Reference.

```
config :omniss7,
  mnesia_storage_type: :disc_copies  # or :ram_copies for testing
```

Storage Type	Description	Persistence	Use Case
`:disc_copies`	Disk-backed storage (default)	**Survives restarts**	Production environments
`:ram_copies`	In-memory only	Lost on restart	Testing, development

Default: `:disc_copies`

Mnesia Database Location

Mnesia stores routing tables in the application's Mnesia directory:

- **Location**: `Mnesia.{node_name}/` (e.g., `Mnesia.nonode@nohost/`)
- **Tables** (all created in `M3UARouting.init_database`):

Table	Purpose
`m3ua_peer`	M3UA / M2PA connection endpoints
`m3ua_route`	Point Code routes (DPC → peer, with mask support)
`m3ua_gt_route`	Global Title routes (prefix → peer, with SSN/TT/NPI/NAI matching and rewrites)
`m3ua_dest_state`	Per-destination availability state driven by MTP3 signalling-route management (TFP/TFA/TFR, Q.704 §13)
`ss7_screen_rule`	Gateway-screening (SCCP firewall) permit/deny rules

Managing Routes

You have three options for managing routes:

1. **Runtime.exs** - Static configuration loaded at startup
2. **Web UI** - Interactive route management (see Web UI Guide)
3. **REST API** - Programmatic route management (see API Guide)

Best Practice: Use `runtime.exs` for base configuration and the Web UI for dynamic route changes during operation.

Configuring M3UA Peers

Peers represent M3UA connection endpoints (other STPs, HLRs, MSCs, SMSCs). Add peers to `config/runtime.exs`.

Peer Configuration Example

> **Note**: The configuration key `peers` is the current standard. The legacy key `m3ua_peers` is still supported for backwards compatibility.

```
config :omniss7,
  peers: [
    # Outbound connection to Partner STP (role: :client)
    %{
      peer_id: 1,                          # Unique identifier
      name: "Partner_STP_West",            # Descriptive name
      role: :client,                       # :client for
outbound, :server for inbound
      local_ip: {10, 0, 0, 1},            # Local IP to bind
      local_port: 0,                       # 0 = dynamic port
assignment
      remote_ip: {10, 0, 0, 10},          # Remote peer IP
      remote_port: 2905,                   # Remote peer port
      routing_context: 1,                  # M3UA routing context
      point_code: 100,                     # Point code of this
peer
      network_indicator: :international    # :international or
:national
    },

    # Connection to Local HLR (role: :client)
    %{
      peer_id: 2,
      name: "Local_HLR",
      role: :client,
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 20},
      remote_port: 2905,
      routing_context: 2,
      point_code: 200,
      network_indicator: :international
    },

    # Inbound connection from Remote MSC (role: :server)
    # For :server role, STP waits for incoming connection
    %{
      peer_id: 3,
      name: "Remote_MSC",
      role: :server,                       # Accept inbound
connection
      remote_ip: {10, 0, 0, 30},          # Expected source IP
      remote_port: 2905,                   # Expected source port
```

```
(0 = accept from any port)
      routing_context: 3,
      point_code: 300,
      network_indicator: :international
    },

    # Inbound connection with dynamic source port (no port
filtering)
    %{
      peer_id: 4,
      name: "Dynamic_Client",
      role: :server,
      remote_ip: {10, 0, 0, 40},           # Expected source IP
      remote_port: 0,                       # 0 = accept
connections from any source port
      routing_context: 4,
      point_code: 400,
      network_indicator: :international
    }
  ]
```

Peer Configuration Parameters

Parameter	Type	Required	Description
`peer_id`	Integer	Yes	Unique numeric identifier for the peer
`name`	String	Yes	Human-readable name for logs and monitoring
`role`	Atom	Yes	`:client` (outbound) or `:server` (inbound)
`local_ip`	Tuple or List	Yes (client)	Local IP address(es) to bind. Single: `{10, 0, 0, 1}` or Multiple for SCTP multihoming: `[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`local_port`	Integer	Yes (client)	Local port (0 for dynamic)
`remote_ip`	Tuple or List	Yes	**Client role**: Single tuple or list for multihomed remote peer. **Server role**: Single tuple only - the IP the remote peer connects FROM (see note below)
`remote_port`	Integer	Yes	Remote peer port (0 for inbound = accept any source port)
`routing_context`	Integer	Yes	M3UA routing context identifier
`point_code`	Integer	Yes	SS7 point code of this peer

Parameter	Type	Required	Description
`network_indicator`	Atom	No	`:international` or `:national`

> **SCTP Multihoming for Server Role (Inbound Connections)**: When accepting inbound connections from a multihomed remote peer, you only need to specify the **single IP address** that the remote peer uses to initiate the SCTP connection (SCTP INIT). This should be a **single tuple**, not a list. SCTP will automatically discover the peer's other multihomed IP addresses during the association handshake. The list format for `remote_ip` is only used for **client role** when connecting to a multihomed remote peer.
>
> **SCTP Multihoming for Client Role (Outbound Connections)**: For network redundancy when connecting to remote peers, you can configure multiple IP addresses for both `local_ip` and `remote_ip` using lists. This enables automatic failover if one network path fails. See SCTP Multihoming Guide for detailed configuration examples and best practices.

Source Port Filtering for Inbound Connections

For **inbound connections** (role: `:server`), the `remote_port` parameter controls source port filtering:

- **Specific Port** (e.g., `remote_port: 2905`): Only accept connections from that exact source port
  - Provides additional security by validating the source port
  - Use when the remote peer uses a fixed source port
- **Any Port** (`remote_port: 0`): Accept connections from any source port
  - Useful when the remote peer uses dynamic/ephemeral source ports
  - Only validates the source IP address
  - More flexible but slightly less secure

Example:

```
# Accept only from 10.5.198.200:2905 (specific port)
%{
  peer_id: 1,
  name: "Strict_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 2905,
  # ... other config
}

# Accept from 10.5.198.200 with any source port
%{
  peer_id: 2,
  name: "Flexible_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 0,  # Accept from any source port
  # ... other config
}
```

M2PA Protocol Support

OmniSS7 supports both **M3UA** and **M2PA** protocols for SS7 signaling transport.

What is M2PA?

M2PA (MTP2 User Peer-to-Peer Adaptation Layer) is an IETF-standardized protocol (RFC 4165) for transporting SS7 MTP3 messages over IP networks using SCTP.

M3UA vs M2PA: Key Differences

Feature	M3UA	M2PA
Architecture	Client/Server (ASP/SGW)	Peer-to-Peer
Use Case	Gateway between SS7 and IP	Direct point-to-point links
Link State Management	Application-level (ASPUP/ASPAC)	MTP2-style (Alignment, Proving, Ready)
Sequence Numbers	No inherent sequencing	24-bit BSN/FSN for ordered delivery
Typical Deployment	SS7-to-IP gateway, STP	Direct signaling links between nodes
RFC	RFC 4666	RFC 4165

Protocol Selection Guidance

Recommendation: Use M3UA by default. Only use M2PA when specifically required.

When to Use M3UA (Recommended)

M3UA is the recommended protocol for most deployments:

- **STP Deployments**: Standard signaling transfer point implementations
- **Gateway Functions**: Bridging SS7 networks with IP-based signaling
- **Network Element Connections**: Connecting HLRs, MSCs, SMSCs, and other network elements to your STP
- **Signaling Gateway (SGW)**: Central gateway accepting connections from multiple Application Servers

- **Flexible Topologies**: Client/server architectures with centralized control
- **Multi-vendor Networks**: Widely supported industry standard (RFC 4666)

Use M3UA for connecting network elements (HLR, MSC, SMSC, VLR, etc.) to your STP.

When to Use M2PA (Special Cases Only)

M2PA should only be used in specific scenarios:

- **STP-to-STP Links**: Direct point-to-point connections between Signal Transfer Points in a multi-STP network
- **Legacy TDM Replacement**: Replacing traditional SS7 TDM links when the remote system specifically requires M2PA
- **MTP2 Compatibility Required**: When connecting to legacy systems that mandate MTP2-style link state management
- **Partner Requirement**: When a partner or interconnect specifically requires M2PA protocol

Important: Do not use M2PA for connecting network elements (HLR, MSC, SMSC) to your STP - use M3UA instead. M2PA is designed for STP-to-STP interconnections where both sides operate as routing nodes.

Configuring M2PA Peers

M2PA peers are configured the same way as M3UA peers, with an additional `protocol` parameter.

M2PA Peer Configuration

Add M2PA peers to your `peers` configuration in `config/runtime.exs` (both M3UA and M2PA peers share the same configuration section, differentiated by the `protocol` parameter):

Key Parameters for M2PA:

Parameter	Value	Description
`protocol`	`:m2pa`	Specifies M2PA protocol (defaults to `:m3ua` if omitted)
`role`	`:client` or `:server`	Connection direction
`local_port`	Integer	Local SCTP port (standard M2PA port is **3565**)
`remote_port`	Integer	Remote SCTP port (standard M2PA port is **3565**)
`point_code`	Integer	Your point code
`adjacent_point_code`	Integer	Remote peer's point code (M2PA-specific)
`send_sltm`	Boolean	Controls SLTM behavior (see SLTM Configuration below)
`network_indicator`	Atom	`:international` or `:national` - must match remote peer

Note: M2PA uses **port 3565** as the industry standard (different from M3UA's port 2905).

SLTM Configuration

SLTM (Signaling Link Test Message) and **SLTA (Signaling Link Test Acknowledgment)** are MTP3 maintenance messages used to verify end-to-end connectivity after an M2PA link reaches the READY state. Per ITU-T Q.707, one side sends SLTM and the other responds with SLTA to confirm the link is operational.

SLTM Behavior

The `send_sltm` parameter controls which side initiates the SLTM test:

`send_sltm` Value	Behavior
`true`	We send SLTM when link becomes READY, wait for SLTA
`false`	We wait for peer to send SLTM, respond with SLTA
Not set (default)	Follows Q.707: SCTP responder sends SLTM, SCTP initiator waits

Default Behavior (Q.707 Compliant):

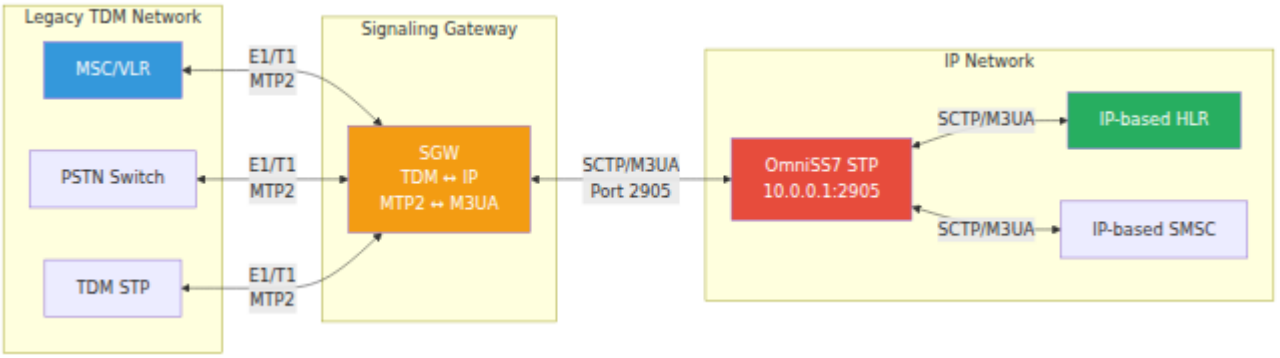
- If `initiate_connection: true` (SCTP initiator/client) → We wait for peer to send SLTM
- If `initiate_connection: false` (SCTP responder/server) → We send SLTM

When to Override SLTM Defaults

Override the default behavior when connecting to equipment that doesn't follow Q.707 conventions:

```
# Example: Force our side to send SLTM regardless of SCTP role
%{
  peer_id: 100,
  name: "Partner_STP",
  protocol: :m2pa,
  role: :client,
  local_ip: {10, 0, 0, 1},
  local_port: 3565,
  remote_ip: {10, 0, 0, 2},
  remote_port: 3565,
  point_code: 7415,
  adjacent_point_code: 15528,
  network_indicator: :international,
  send_sltm: true  # Override: We send SLTM even though we're the
SCTP initiator
}
```

SLTM Message Flow



Troubleshooting SLTM Issues

Symptom: Link reaches READY but no traffic flows

Both sides may be waiting for the other to send SLTM. Check logs for:

```
"waiting for peer to send SLTM"
```

Resolution: Configure `send_sltm: true` on one side to break the deadlock.

Symptom: SLTM sent but no SLTA received

Possible causes:

1. `adjacent_point_code` is incorrect (SLTM sent with wrong DPC)
2. Network indicator mismatch (`:international` vs `:national`)
3. Remote peer not configured to respond to our point code

Resolution: Verify `adjacent_point_code` matches the STP's own point code (configured in `sctp_handler.point_code`).

M2PA Link States

M2PA links progress through several states during initialization:

1. **Down** - No connection established
2. **Alignment** - Initial synchronization phase (~1 second)
3. **Proving** - Link quality verification (~2 seconds)
4. **Ready** - Link active and ready for traffic

The link state progression ensures reliable signaling before traffic is exchanged.

Managing M2PA Peers via Web UI

The **Routing** page in the Web UI provides full support for managing M2PA peers:

1. **Navigate** to the Routing page
2. **Select** the "Peers" tab
3. **Click** "Add New Peer"
4. **Choose** "M2PA (RFC 4165)" from the Protocol dropdown
5. **Fill in** the peer configuration:
    - Peer Name (descriptive identifier)
    - Protocol: M2PA
    - Role: client or server
    - Point Code (your PC)
    - Local/Remote IP addresses
    - Local/Remote ports (typically 3565 for M2PA)
    - Network Indicator (international or national)
6. **Click** "Save Peer"

The peers table displays the protocol type with color coding:

- **Blue** - M3UA peers
- **Green** - M2PA peers

M2PA Routing Behavior

M2PA peers integrate seamlessly with OmniSS7's routing system:

- **Point Code Routes**: Work identically for M2PA and M3UA
- **Global Title Routes**: Fully supported on M2PA links
- **Route Priority**: M2PA and M3UA peers can be mixed in the same routing tables
- **Message Relay**: Messages can arrive on M2PA and be routed to M3UA, and vice versa

M2PA Metrics

M2PA provides comprehensive Prometheus metrics for monitoring link health and traffic:

Traffic Metrics:

- `m2pa_messages_sent_total` - Total MTP3 messages sent per link
- `m2pa_messages_received_total` - Total MTP3 messages received per link
- `m2pa_bytes_sent_total` - Total bytes sent over M2PA
- `m2pa_bytes_received_total` - Total bytes received over M2PA

All traffic metrics are labeled by: `link_name`, `point_code`, `adjacent_pc`

Link State Metrics:

- `m2pa_link_state_changes_total` - Link state transitions (DOWN → ALIGNMENT → PROVING → READY)
  - Labels: `link_name`, `from_state`, `to_state`

Error Metrics:

- `m2pa_errors_total` - Total errors by type
  - `decode_error` - M2PA message decode failures
  - `encode_error` - M2PA message encode failures
  - `sctp_send_error` - SCTP transmission failures
  - Labels: `link_name`, `error_type`

Access Metrics:

- Prometheus endpoint: `http://your-server:8080/metrics`
- Metrics auto-register on application startup

M2PA Best Practices

1. **Port Selection**: Use port 3565 for M2PA (industry standard)
2. **Link Monitoring**: Monitor link state changes via metrics
3. **Firewall Rules**: Ensure SCTP (IP protocol 132) is allowed

4. **Point Codes**: Ensure adjacent point codes are correctly configured on both sides
5. **Network Indicator**: Must match between peers (international or national)
6. **Testing**: Use the Routing Test page to verify connectivity after configuration

M2PA Socket Requirements

M2PA uses SCTP.SocketHandler shared sockets. All M2PA peers automatically use the SCTP.SocketHandler for socket management, which allows multiple peers to efficiently share the same SCTP port.

Requirements:

- SCTP.SocketHandler must be enabled and running
- The shared socket must be configured before M2PA peers start

Example:

```
# Enable SCTP.SocketHandler
sctp_handler: %{
  enabled: true,
  local_ip: {10, 179, 4, 10},
  local_port: 3565,
  point_code: 100
}

# M2PA peers automatically use the shared socket
peers: [
  %{
    peer_id: 1,
    name: "M2PA_Link_1",
    protocol: :m2pa,
    role: :client,
    local_ip: {10, 179, 4, 10},
    local_port: 3565,
    remote_ip: {10, 179, 4, 20},
    remote_port: 3565,
    point_code: 100,
    adjacent_point_code: 200
  },
  %{
    peer_id: 2,
    name: "M2PA_Link_2",
    protocol: :m2pa,
    role: :client,
    local_ip: {10, 179, 4, 10},
    local_port: 3565,           # Same port shared across peers
    remote_ip: {10, 179, 4, 30},
    remote_port: 3565,
    point_code: 100,
    adjacent_point_code: 300
  }
]
```

SCTP Requirements:

- Ordered delivery (no `unordered` flag)
- PPID 5 (M2PA identifier per RFC 4165)

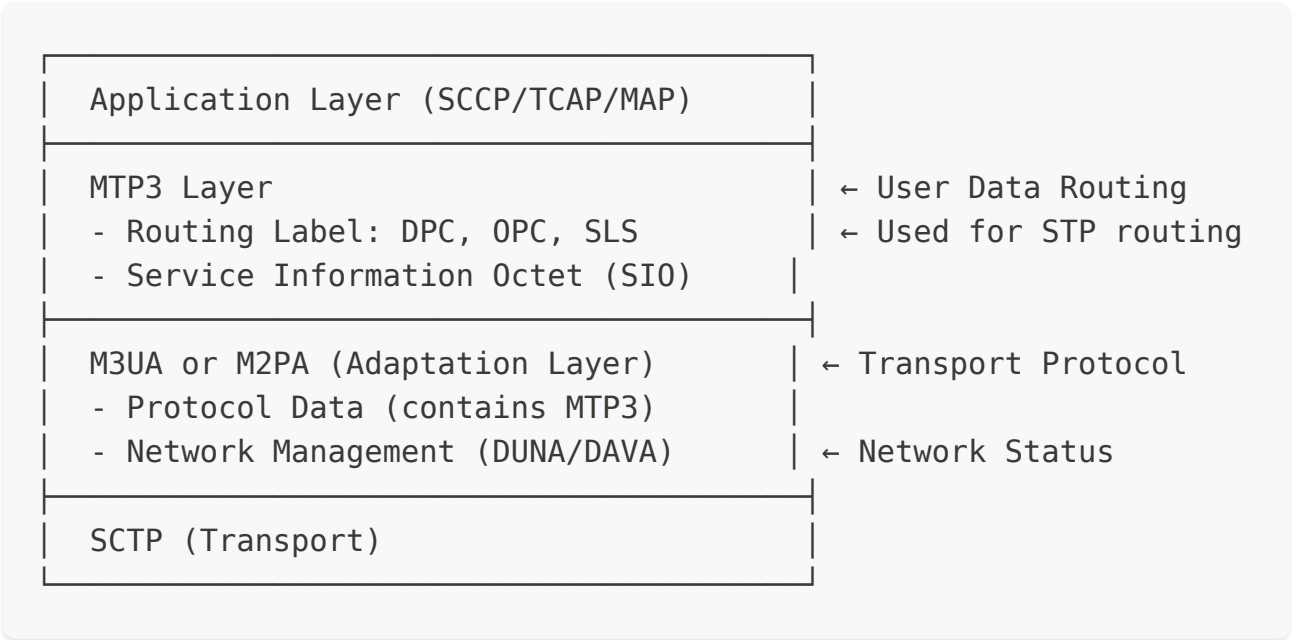
Configuring Point Code Routing

Point Code routing directs messages based on the **Destination Point Code (DPC)** in the MTP3 header.

Understanding Point Codes in SS7 Protocol Stack

Point codes exist at different layers of the SS7 protocol stack. Understanding this distinction is important:

Protocol Stack Layers:



Two Types of Point Codes:

1. **MTP3 Layer Point Codes** (Used for Routing):
    - Located in the MTP3 routing label (DPC, OPC)
    - Present in M3UA Protocol Data parameter (tag 528)
    - Present in M2PA User Data messages
    - **STP uses these DPC values for routing decisions**
    - These determine where the message is ultimately delivered
2. **M3UA Layer Point Codes** (Used for Network Management):

    - Present in M3UA management messages (DUNA, DAVA, SCON, DUPU)
    - Indicate affected point codes for network status
    - Tell peers which destinations are available/unavailable
    - Not used for routing user data

How STP Routing Works:

- **For M3UA DATA messages**: STP extracts the MTP3 message from the Protocol Data parameter (tag 528), which contains the MTP3 routing label (DPC, OPC, SLS). The DPC from the MTP3 layer is used to look up routes.
- **For M2PA User Data messages**: STP extracts the MTP3 message from the M2PA user data field, then reads the DPC from the MTP3 routing label.
- **M3UA management messages**: Network management messages (DUNA, DAVA, SCON) contain affected point codes at the M3UA layer for signaling network status between peers.

Basic Point Code Routes

Add routes to `config/runtime.exs`:

```
config :omniss7,
  m3ua_routes: [
    # Route all traffic for PC 100 to peer 1 (Partner STP)
    %{
      dest_pc: 100,                   # Destination point code
      peer_id: 1,                     # Peer to route through
      priority: 1,                    # Priority (lower = higher
priority)
      network_indicator: :international
      # mask: 14                      # Optional: defaults to 14
(exact match)
    },

    # Route all traffic for PC 200 to peer 2 (Local HLR)
    %{
      dest_pc: 200,
      peer_id: 2,
      priority: 1,
      network_indicator: :international
    },

    # Load balancing example: PC 300 with primary and backup
routes
    %{
      dest_pc: 300,
      peer_id: 3,                     # Primary route
      priority: 1,
      network_indicator: :international
    },
    %{
      dest_pc: 300,
      peer_id: 4,                     # Backup route (higher
priority number)
      priority: 2,
      network_indicator: :international
    }
  ]
```

Note: The `mask` field is optional and defaults to `14` (exact point code match). Only specify `mask` when you need range-based routing (see Point Code Masks section below).

Routing Logic

1. STP receives M3UA DATA or M2PA User Data message
2. STP extracts the **MTP3 message** from the Protocol Data (M3UA) or User Data (M2PA) field
3. STP reads the **Destination Point Code (DPC)** from the MTP3 routing label
4. Looks up routing table for matching DPC (considering masks)
5. If multiple routes exist, selects the route with **most specific mask** (highest mask value), then **lowest priority number**
6. Wraps the MTP3 message in M3UA DATA or M2PA User Data for the destination peer
7. Routes the message to the corresponding peer
8. If the selected peer is down, tries the next highest priority route

Point Code Masks

Point codes are 14-bit values (range 0-16383). By default, routes match a single point code exactly (mask `/14`). However, you can use **point code masks** to create routes that match **ranges** of point codes.

Understanding Masks

The mask specifies how many **most significant bits** must match between the route's destination PC and the incoming message's DPC. The remaining bits can be any value, creating a range of matching point codes.

Mask Reference Table:

Mask	Point Codes Matched	Use Case
`/14`	1 PC (exact match)	Single destination (default)
`/13`	2 PCs	Small range
`/12`	4 PCs	Small range
`/11`	8 PCs	Small range
`/10`	16 PCs	Medium range
`/9`	32 PCs	Medium range
`/8`	64 PCs	Medium range
`/7`	128 PCs	Medium-large range
`/6`	256 PCs	Large range
`/5`	512 PCs	Large range
`/4`	1,024 PCs	Very large range
`/3`	2,048 PCs	Very large range
`/2`	4,096 PCs	Extremely large range
`/1`	8,192 PCs	Half of all PCs
`/0`	16,384 PCs	All PCs (default/fallback route)

Point Code Mask Examples

Note: The `mask` field is **optional** in all examples. If omitted, it defaults to `14` (exact match).

Example 1: Single Point Code (Default Behavior)

```
# Without mask field (recommended for single PC)
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  network_indicator: :international
}
# Mask defaults to 14 - Matches: Only PC 1000

# Explicit mask (same result)
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  mask: 14,                              # Explicit exact match
  network_indicator: :international
}
# Matches: Only PC 1000
```

Example 2: Small Range

```
%{
  dest_pc: 1000,
  peer_id: 2,
  priority: 1,
  mask: 12,                              # Matches 4 PCs
  network_indicator: :international
}
# Matches: PC 1000, 1001, 1002, 1003
```

Example 3: Medium Range

```
%{
  dest_pc: 1000,
  peer_id: 3,
  priority: 1,
  mask: 8,                                  # Matches 64 PCs
  network_indicator: :international
}
# Matches: PC 1000-1063 (64 consecutive point codes)
```

Example 4: Default/Fallback Route

```
%{
  dest_pc: 0,
  peer_id: 4,
  priority: 10,                             # Low priority (high
number)
  mask: 0,                                  # Matches all PCs
  network_indicator: :international
}
# Matches: All point codes (0-16383)
# Use as a catch-all/default route with low priority
```

Combining Specific and Masked Routes

You can combine specific routes with masked routes for flexible routing:

```
config :omniss7,
  m3ua_routes: [
    # Specific route for PC 1000 (takes precedence)
    %{
      dest_pc: 1000,
      peer_id: 1,
      priority: 1,
      network_indicator: :international
      # mask defaults to 14 (exact match)
    },

    # Range route for PCs 1000-1063
    %{
      dest_pc: 1000,
      peer_id: 2,
      priority: 1,
      mask: 8,                          # Matches 64 PCs
      network_indicator: :international
    },

    # Default/fallback route for all other PCs
    %{
      dest_pc: 0,
      peer_id: 3,
      priority: 10,                     # Low priority
      mask: 0,                          # Matches all PCs
      network_indicator: :international
    }
  ]
```

Routing Decision for DPC 1000:

1. Matches mask `/14` route (PC 1000 exactly) - **Selected** (most specific)
2. Also matches mask `/8` route (PC 1000-1063 range) - Ignored (less specific)
3. Also matches mask `/0` route (all PCs) - Ignored (least specific)

Routing Decision for DPC 1015:

1. Does not match mask `/14` route (PC 1000 only)
2. Matches mask `/8` route (PC 1000-1063 range) - **Selected** (most specific match)

3. Also matches mask `/0` route (all PCs) - Ignored (less specific)

Routing Decision for DPC 5000:

1. Does not match mask `/14` route
2. Does not match mask `/8` route
3. Matches mask `/0` route (all PCs) - **Selected** (only match, fallback route)

Best Practices

1. **Omit `mask` for Single Destinations**: For exact point code matches, omit the `mask` field entirely (defaults to `/14`)
2. **Use `/14` Explicitly Only When Needed**: Only specify `mask: 14` when you need to make it clear in documentation or when mixing with range routes
3. **Use Range Masks for Network Blocks**: Route entire network segments to specific peers with masks `/0` through `/13`
4. **Use `/0` as Fallback**: Create a default route with low priority to catch unmatched traffic
5. **Most Specific Wins**: The routing engine always selects the most specific (highest mask value) matching route first
6. **Priority as Tiebreaker**: If multiple routes have the same mask, lowest priority number wins

---

Configuring Global Title (GT) Routing

Global Title routing enables **content-based routing** using phone numbers or IMSI values instead of point codes. For advanced Global Title address translation based on calling/called party, see the Global Title NAT Guide.

Prerequisites

- Enable GT routing: `enable_gt_routing: true` in `config/runtime.exs`

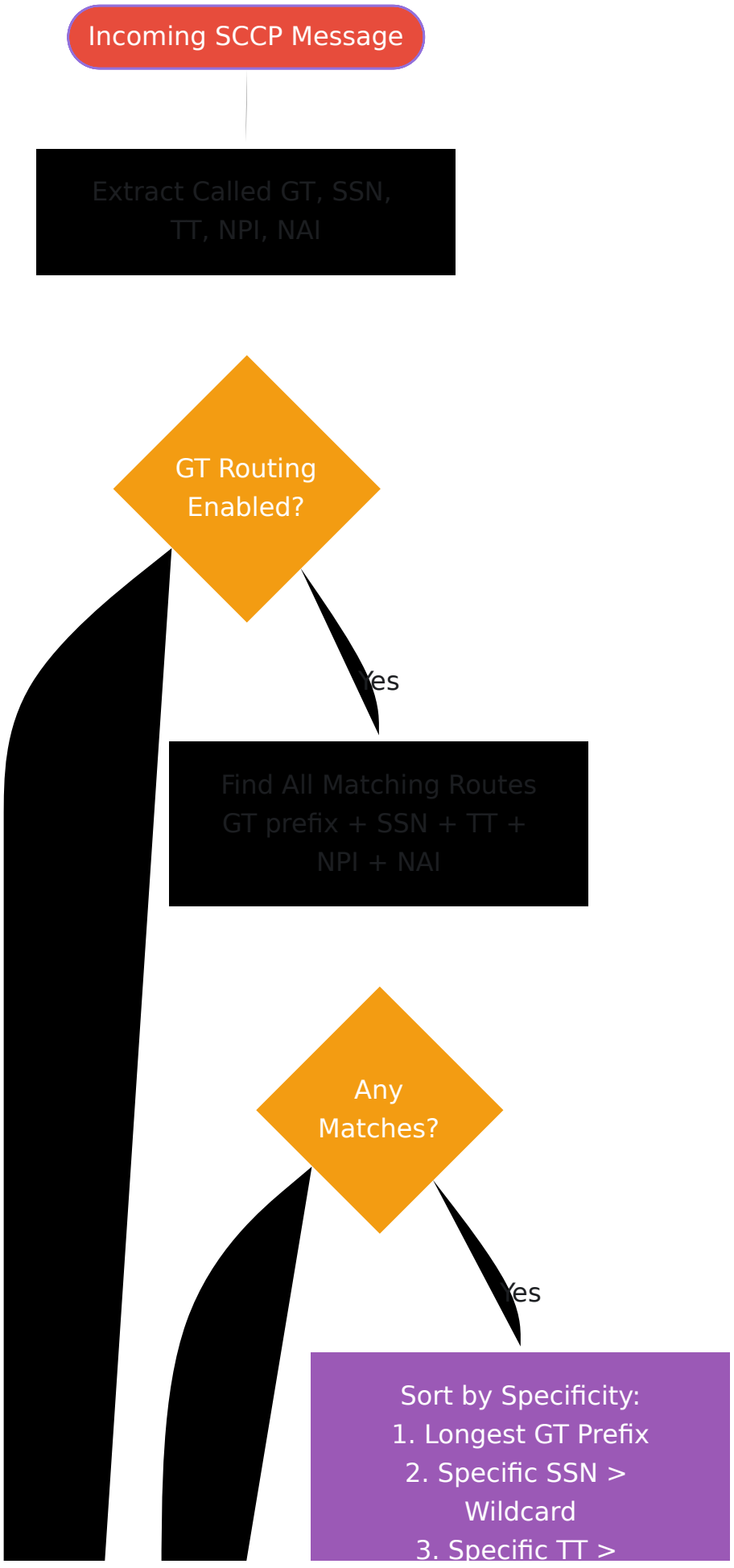
GT Route Configuration

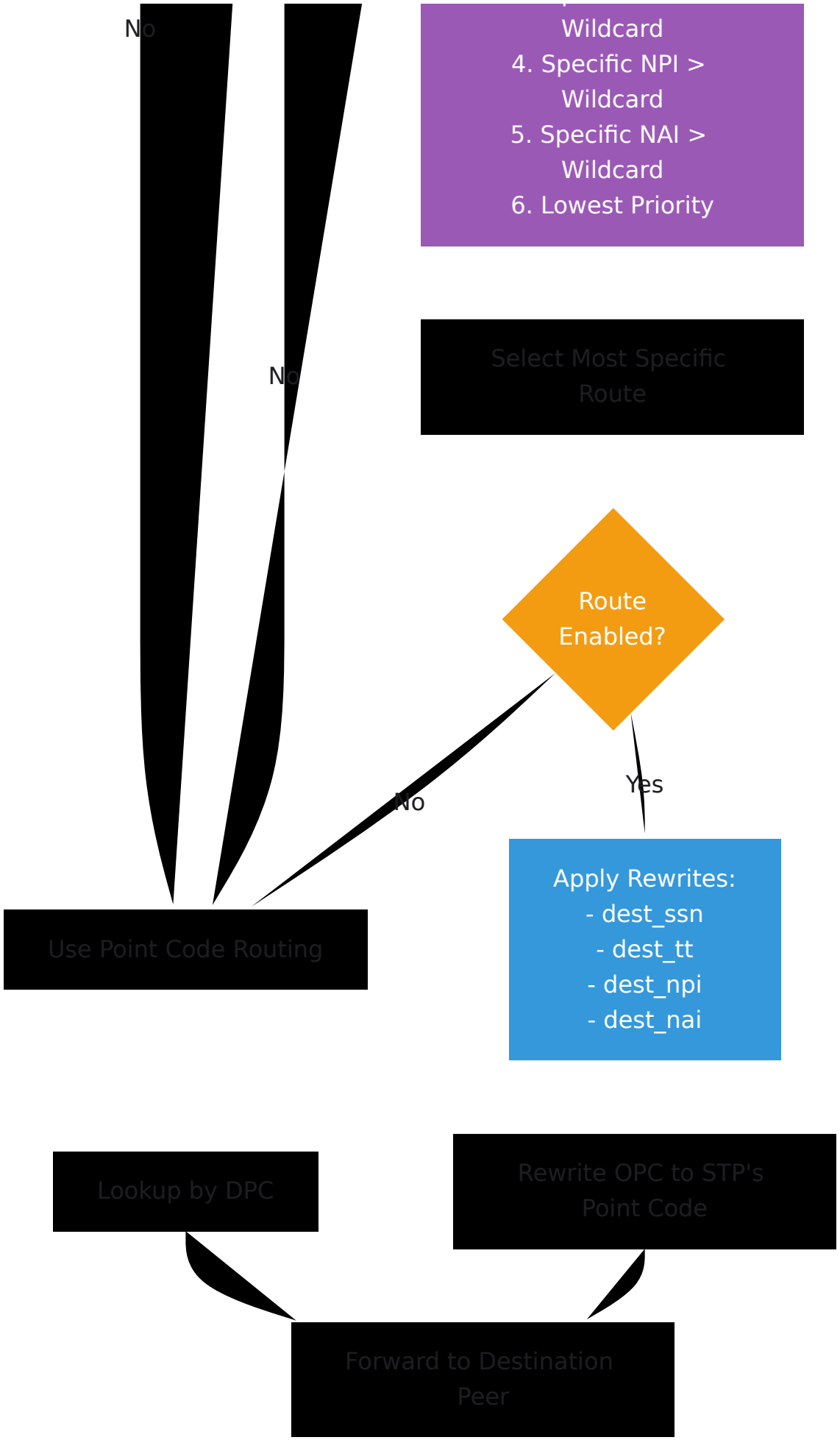
```
config :omniss7,
  # Enable GT routing
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # Route all UK numbers (prefix 44) to peer 1
    %{
      gt_prefix: "44",                # Global Title prefix to
match
      peer_id: 1,                     # Destination peer
      priority: 1,                    # Priority (lower = higher)
      description: "UK numbers"       # Description for logging
    },

    # Route US numbers (prefix 1) to peer 2
    %{
      gt_prefix: "1",
      peer_id: 2,
      priority: 1,
      description: "US numbers"
    },

    # More specific route: UK mobile numbers starting with 447
    %{
      gt_prefix: "447",               # Longest prefix match wins
      peer_id: 3,
      priority: 1,
      description: "UK mobile numbers"
    },

    # SSN-specific routing (optional)
    %{
      gt_prefix: "555",
      source_ssn: 8,                  # Only match if source SSN
= 8 (SMSC)
      peer_id: 4,
      dest_ssn: 6,                    # Rewrite destination SSN
to 6 (HLR)
      priority: 1,
      description: "SMS traffic for 61 prefix"
```

```
    }
  ]
```

GT Routing Logic

The GT routing algorithm follows this decision process:


Incoming SCCP Message
Extract Called GT, SSN, TT, NPI, NAI
GT Routing Enabled?
Yes
Find All Matching Routes GT prefix + SSN + TT + NPI + NAI
Any Matches?
Yes
Sort by Specificity:
1. Longest GT Prefix
2. Specific SSN > Wildcard
3. Specific TT >


No
Wildcard
4. Specific NPI >
Wildcard
5. Specific NAI >
Wildcard
6. Lowest Priority
No
Select Most Specific
Route
Route
Enabled?
No
Yes
Use Point Code Routing
Apply Rewrites:
- dest_ssn
- dest_tt
- dest_npi
- dest_nai
Lookup by DPC
Rewrite OPC to STP's
Point Code
Forward to Destination
Peer

Routing Steps:

1. **Longest Prefix Match**: The STP finds all GT routes where the prefix matches the beginning of the Global Title
    - Example: GT "447712345678" matches both "44" and "447", but "447" wins (longest match)
2. **SSN Matching** (Optional):
    - If `source_ssn` is specified, the route only matches when the SCCP Called Party SSN equals that value
    - If `source_ssn` is `nil`, the route matches any SSN (wildcard)
3. **TT/NPI/NAI Matching** (Optional):
    - If `source_tt`, `source_npi`, or `source_nai` are specified, routes must match those indicators
    - `nil` values act as wildcards (match any value)
4. **Specificity-Based Selection**:
    - Routes with more specific matching criteria win over wildcards
    - Priority order: GT Prefix Length → SSN → TT → NPI → NAI → Priority Number
5. **Indicator Rewriting** (Optional):
    - If `dest_ssn`, `dest_tt`, `dest_npi`, or `dest_nai` are specified, the STP rewrites those indicators
    - Useful for protocol normalization and network interconnection
6. **Fallback to Point Code**:
    - If no GT route matches, the STP falls back to Point Code routing using the DPC

Advanced GT Routing: Translation Type, NPI, and NAI

In addition to GT prefix and SSN matching, the STP supports routing and transformation based on SCCP Global Title indicators:

- **Translation Type (TT)**: Identifies the numbering plan and address type
- **Numbering Plan Indicator (NPI)**: Defines the numbering plan (e.g., ISDN, Data, Telex)
- **Nature of Address Indicator (NAI)**: Specifies the address format (e.g., International, National, Subscriber)

Matching (Source Indicators)

Routes can match on incoming message indicators:

- `source_tt`: Match messages with specific Translation Type
- `source_npi`: Match messages with specific Numbering Plan Indicator
- `source_nai`: Match messages with specific Nature of Address Indicator
- `nil` value = wildcard (matches any value)

Transformation (Destination Indicators)

Routes can rewrite indicators when forwarding:

- `dest_tt`: Transform Translation Type to new value
- `dest_npi`: Transform Numbering Plan Indicator to new value

- `dest_nai`: Transform Nature of Address Indicator to new value
- `nil` value = preserve original value (no transformation)

Specificity-Based Selection

When multiple routes match, the most specific route is selected using this priority order:

1. Longest GT prefix match
2. Specific source SSN over wildcard SSN
3. Specific source TT over wildcard TT
4. Specific source NPI over wildcard NPI
5. Specific source NAI over wildcard NAI
6. Lowest priority number

Configuration Examples

```
config :omniss7,
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # Example 1: Match and transform Translation Type
    %{
      gt_prefix: "44",
      peer_id: 1,
      source_tt: 0,      # Match TT=0 (Unknown)
      dest_tt: 3,        # Transform to TT=3 (National)
      priority: 1,
      description: "UK numbers: TT 0→3 transformation"
    },

    # Example 2: Match specific NPI and transform NAI
    %{
      gt_prefix: "1",
      peer_id: 2,
      source_npi: 1,     # Match NPI=1 (ISDN/Telephony)
      source_nai: 4,     # Match NAI=4 (International)
      dest_nai: 3,       # Transform to NAI=3 (National)
      priority: 1,
      description: "US numbers: International→National NAI"
    },

    # Example 3: Combined SSN and indicator routing
    %{
      gt_prefix: "33",
      source_ssn: 8,     # Match SMSC traffic
      source_tt: 0,      # Match TT=0
      dest_ssn: 6,       # Rewrite SSN to HLR
      dest_tt: 2,        # Transform to TT=2
      dest_npi: 1,       # Set NPI=1 (ISDN)
      dest_nai: 4,       # Set NAI=4 (International)
      peer_id: 3,
      priority: 1,
      description: "French SMS: Full normalization"
    },

    # Example 4: Wildcard TT, specific NPI
    %{
      gt_prefix: "49",
      source_tt: nil,    # Match any TT (wildcard)
```

```
      source_npi: 6,     # Match NPI=6 (Data)
      dest_npi: 1,       # Transform to NPI=1 (ISDN)
      peer_id: 4,
      priority: 1,
      description: "German data network normalization"
    }
  ]
```

Common TT/NPI/NAI Values

Translation Type (TT):

- `0` = Unknown
- `1` = International
- `2` = National
- `3` = Network Specific

Numbering Plan Indicator (NPI):

- `0` = Unknown
- `1` = ISDN/Telephony (E.164)
- `3` = Data (X.121)
- `4` = Telex (F.69)
- `6` = Land Mobile (E.212)

Nature of Address Indicator (NAI):

- `0` = Unknown
- `1` = Subscriber Number
- `2` = Reserved for National Use
- `3` = National Significant Number
- `4` = International Number

Routing Decision Example

For an incoming message with:

- GT: "447712345678"

- SSN: 8
- TT: 0
- NPI: 1
- NAI: 4

With these configured routes:

```
# Route A: Wildcard TT
%{gt_prefix: "447", peer_id: 1, priority: 1}

# Route B: Specific TT
%{gt_prefix: "447", source_tt: 0, peer_id: 2, priority: 1}

# Route C: Specific TT + NPI
%{gt_prefix: "447", source_tt: 0, source_npi: 1, peer_id: 3,
priority: 1}
```

Result: Route C is selected (most specific: matches GT + TT + NPI)

The message is forwarded with indicators transformed per Route C's `dest_tt`, `dest_npi`, `dest_nai` values.

GT Routing Examples

Called GT	Source SSN	TT	NPI	NAI	Matched Route	Reason
447712345678	6	-	-	-	"447" → peer 3	Longest prefix match
441234567890	6	-	-	-	"44" → peer 1	Prefix match, no more specific route
12125551234	6	-	-	-	"1" → peer 2	Prefix match for US numbers
555881234567	8	-	-	-	"555" (SSN 8) → peer 4	GT + SSN match, rewrites SSN to 6
555881234567	6	-	-	-	"555" (SSN wildcard) → peer X	GT match, no SSN rewrite
441234567890	6	0	1	4	"44" (TT=0) → peer 1	GT + TT match, transforms TT to 3
12125551234	8	0	1	4	"1" (TT=0, NPI=1, NAI=4)	Most specific: GT+TT+NPI+NAI match

Practical Use Cases for TT/NPI/NAI Routing

1. **Network Interconnection Normalization**

- Different networks may use different indicator conventions
- Transform indicators at the interconnection point to ensure compatibility
- Example: Partner network uses TT=0 for international, your network uses TT=1

2. **Protocol Conversion**
   - Convert between numbering plans when routing between different network types
   - Example: Route from mobile network (NPI=6) to PSTN (NPI=1)
3. **Address Format Standardization**
   - Normalize all incoming traffic to use consistent NAI values
   - Example: Convert all international format (NAI=4) to national format (NAI=3) for domestic routing
4. **Carrier-Specific Routing**
   - Route based on translation type to different service providers
   - Example: TT=0 routes to Carrier A, TT=2 routes to Carrier B
5. **Legacy System Integration**
   - Modern systems might use different indicator values than legacy systems
   - Transform at the STP to maintain backward compatibility

SCCP Transit Handling

Beyond selecting a destination peer, the STP performs the SCCP-layer transit functions required of a true Signalling Connection Control Point. These apply to the connectionless message classes the STP relays: **UDT** (0x09), **XUDT** (0x11), and **LUDT** (0x13).

Route-on-SSN After Global Title Translation

After a Global Title Translation (GTT) resolves a message to a final destination that is addressed by subsystem, the STP sets the Called Party **routing indicator to route-on-SSN** before forwarding (ITU-T Q.714 §2.2.2.1). This lets SSN-only endpoints accept the message, because once GTT has been performed the next node should route on the resolved point code and SSN rather than repeat the translation.

This happens automatically whenever a GT route is matched and applied — no configuration is required.

Hop-Counter Enforcement (XUDT / LUDT)

XUDT and LUDT messages carry an **SCCP hop counter** that guards against routing loops between SCCP relay nodes. On every transit hop the STP:

1. **Decrements** the hop counter in the forwarded message (Q.714 §2.4).
2. If decrementing would take the counter to **zero**, the message is **discarded** rather than forwarded (Q.714 §2.8.6).
3. A discard increments `m3ua_stp_routing_failures_total{reason="hcounter_violation"}`.
4. If the original message requested return-on-error, an `XUDTS`/`LUDTS` message-return is generated back to the originator (see below).

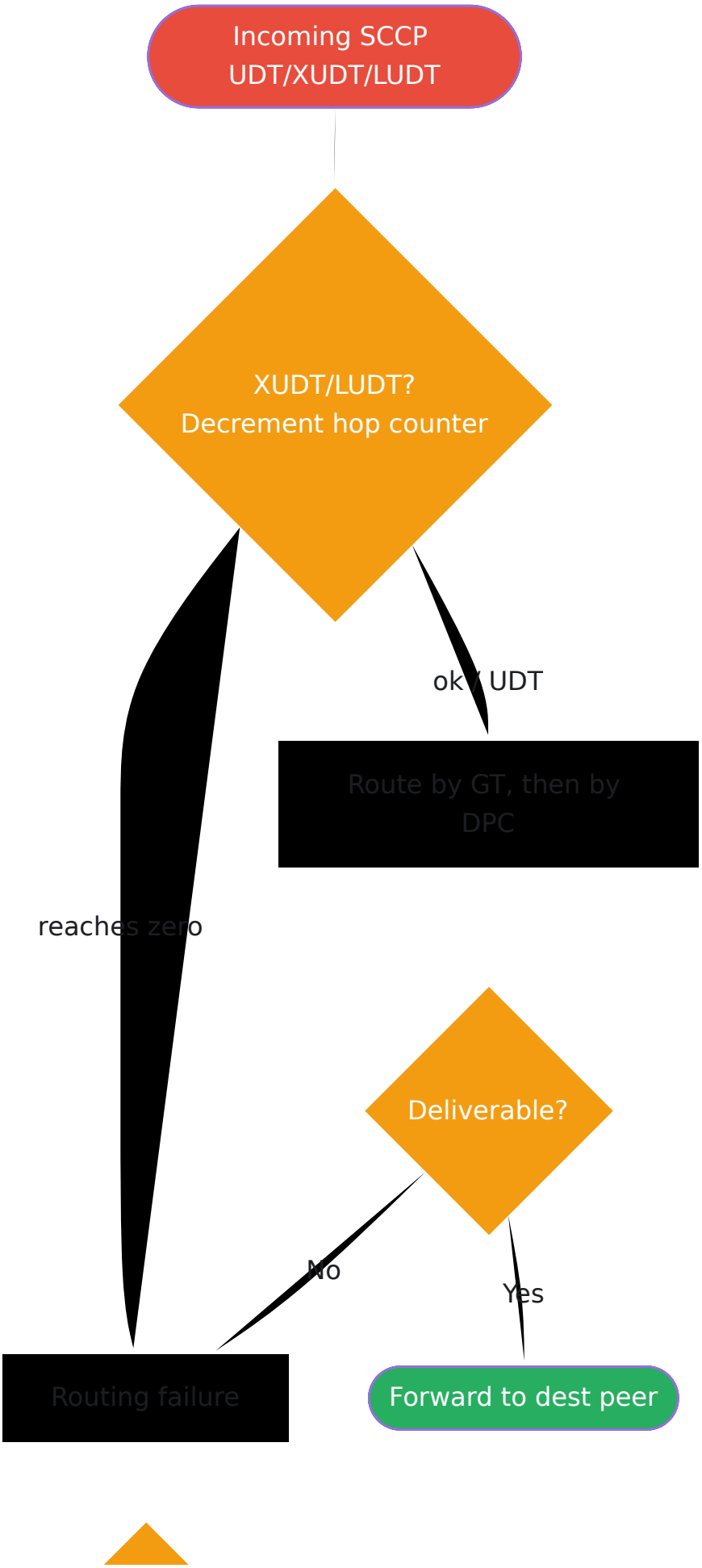
UDT has no hop counter and is forwarded unchanged at this step.

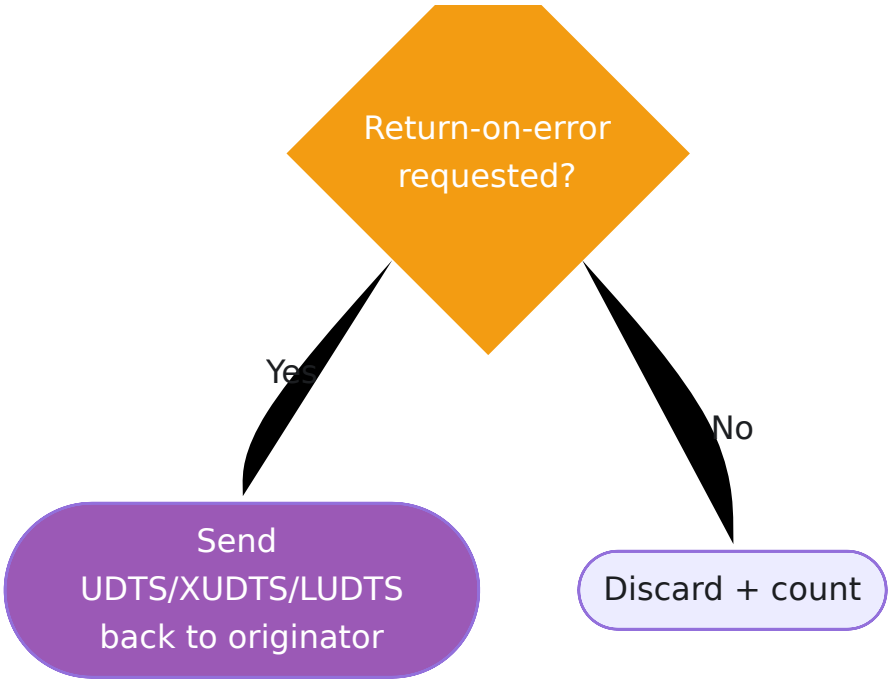
Message-Return on Routing Failure (UDTS / XUDTS / LUDTS)

When a connectionless message cannot be delivered (no route, prohibited destination, hop-counter exhausted, etc.) **and the originator set the return-on-error option**, the STP builds an SCCP message-return and sends it back toward the originator (ITU-T Q.714 §4.2):

Original message	Returned as
UDT (0x09)	UDTS (0x0A)
XUDT (0x11)	XUDTS (0x12)
LUDT (0x13)	LUDTS (0x14)

The return-cause carried in the message reflects the failure reason (for example, an MTP failure for a prohibited destination, or a hop-counter violation for an exhausted counter). If the originator did **not** request return-on-error, the message is simply discarded and counted.


Incoming SCCP
UDT/XUDT/LUDT
XUDT/LUDT?
Decrement hop counter
ok / UDT
Route by GT, then by
DPC
reaches zero
Deliverable?
No
Yes
Routing failure
Forward to dest peer



Destination State (Signalling-Route Management)

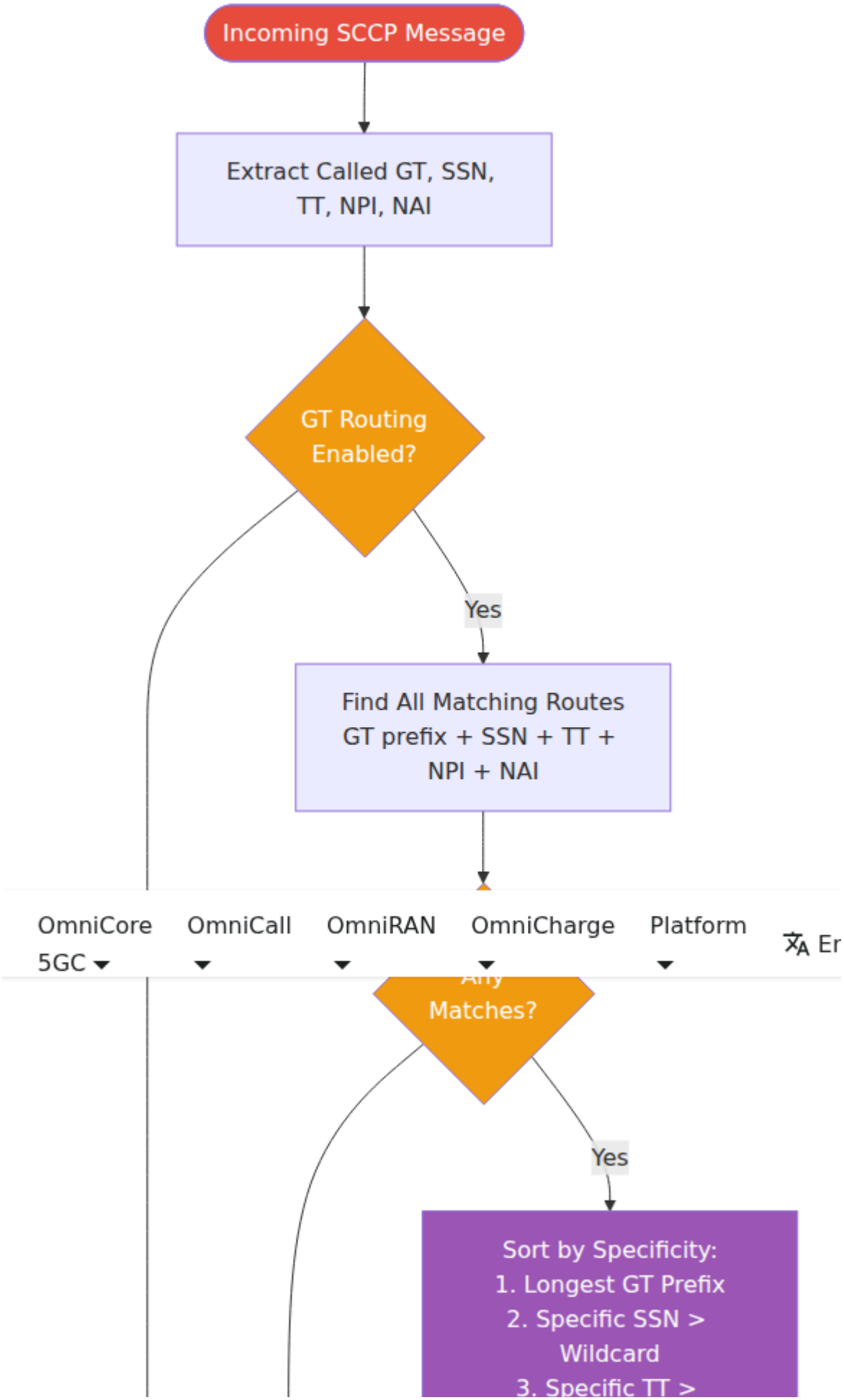
The STP tracks the availability of each destination point code using MTP3 signalling-route-management messages, per ITU-T Q.704 §13. State is held in the `m3ua_dest_state` Mnesia table; the absence of a row means the destination is **allowed** (the default).

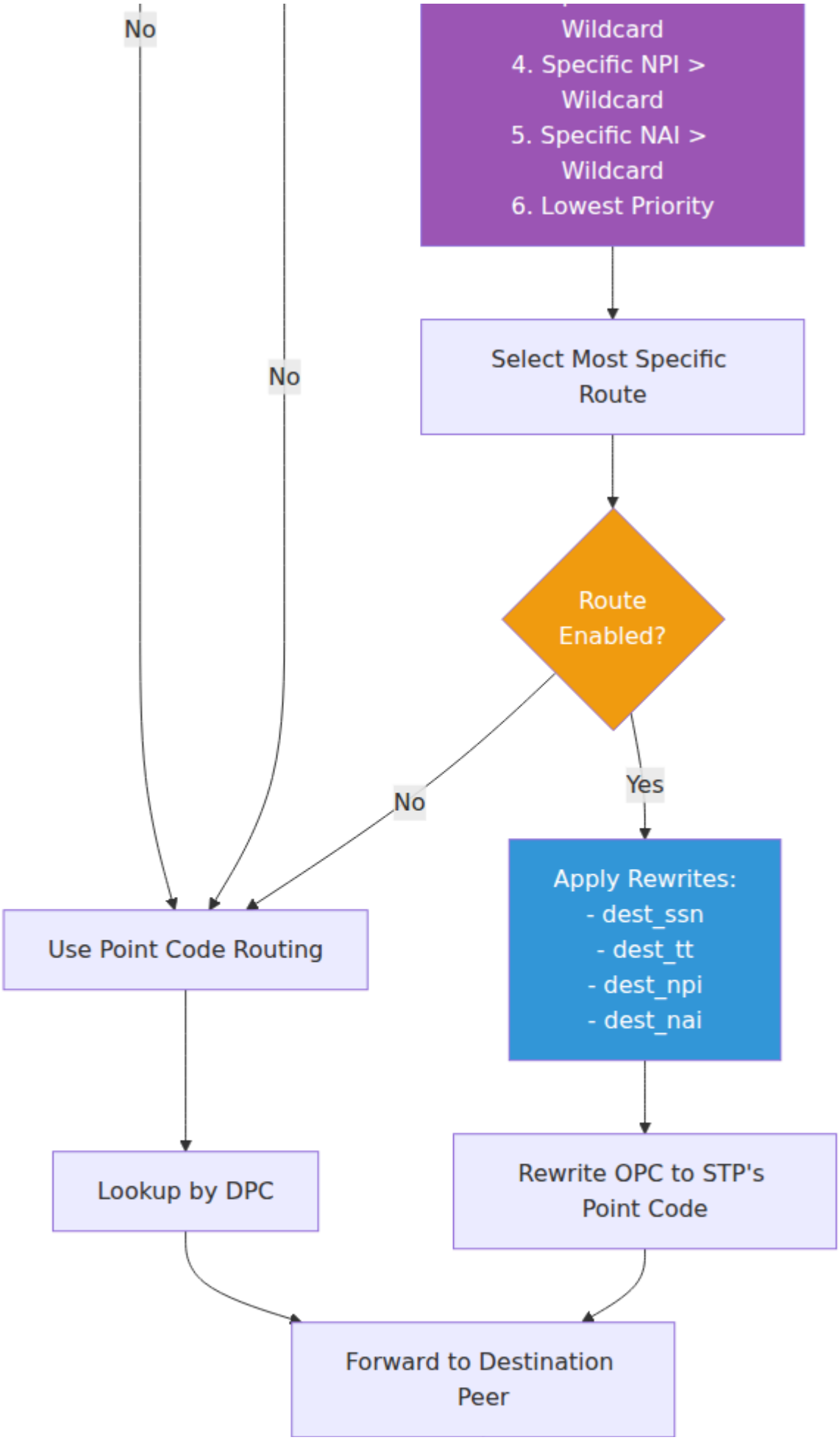
TFP / TFA / TFR Handling

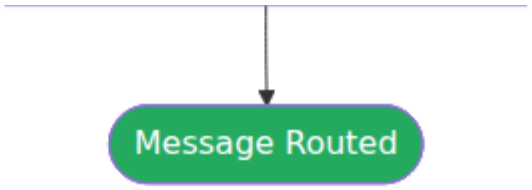
Message	Meaning	Effect on STP state
TFP (Transfer Prohibited)	A destination is no longer reachable via this route	Destination marked `:prohibited`
TFA (Transfer Allowed)	A destination is reachable again	Destination state cleared (back to allowed)
TFR (Transfer Restricted)	A destination should be used only when no alternative exists	Destination marked `:restricted`

Effect on Routing

- During a Point Code lookup, the STP first checks destination state. A destination marked **transfer-prohibited** is skipped — the lookup returns `{:error, :prohibited}` and the message is not routed there (Q.704 §13.2.3). This increments `m3ua_stp_routing_failures_total{reason="prohibited"}`.
- **DAUD** (Destination Audit) requests are answered from this real state rather than with a blanket "available": reachable destinations are reported with **DAVA**, and unreachable or prohibited destinations with **DUNA** (RFC 4666 §3.4.3).


Incoming SCCP Message
Extract Called GT, SSN, TT, NPI, NAI
GT Routing Enabled?
Yes
Find All Matching Routes GT prefix + SSN + TT + NPI + NAI
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
Er
Matches?
Yes
Sort by Specificity:
1. Longest GT Prefix
2. Specific SSN > Wildcard
3. Specific TT >


No
Wildcard
4. Specific NPI >
Wildcard
5. Specific NAI >
Wildcard
6. Lowest Priority
No
Select Most Specific
Route
Route
Enabled?
No
Yes
Use Point Code Routing
Apply Rewrites:
- dest_ssn
- dest_tt
- dest_npi
- dest_nai
Lookup by DPC
Rewrite OPC to STP's
Point Code
Forward to Destination
Peer



Gateway Screening (SCCP Firewall)

Gateway screening lets the STP **permit or deny transit traffic** before it is routed. Without it the STP forwards every message unconditionally; with it, operators can block unwanted signalling at the gateway — a core SS7 security control.

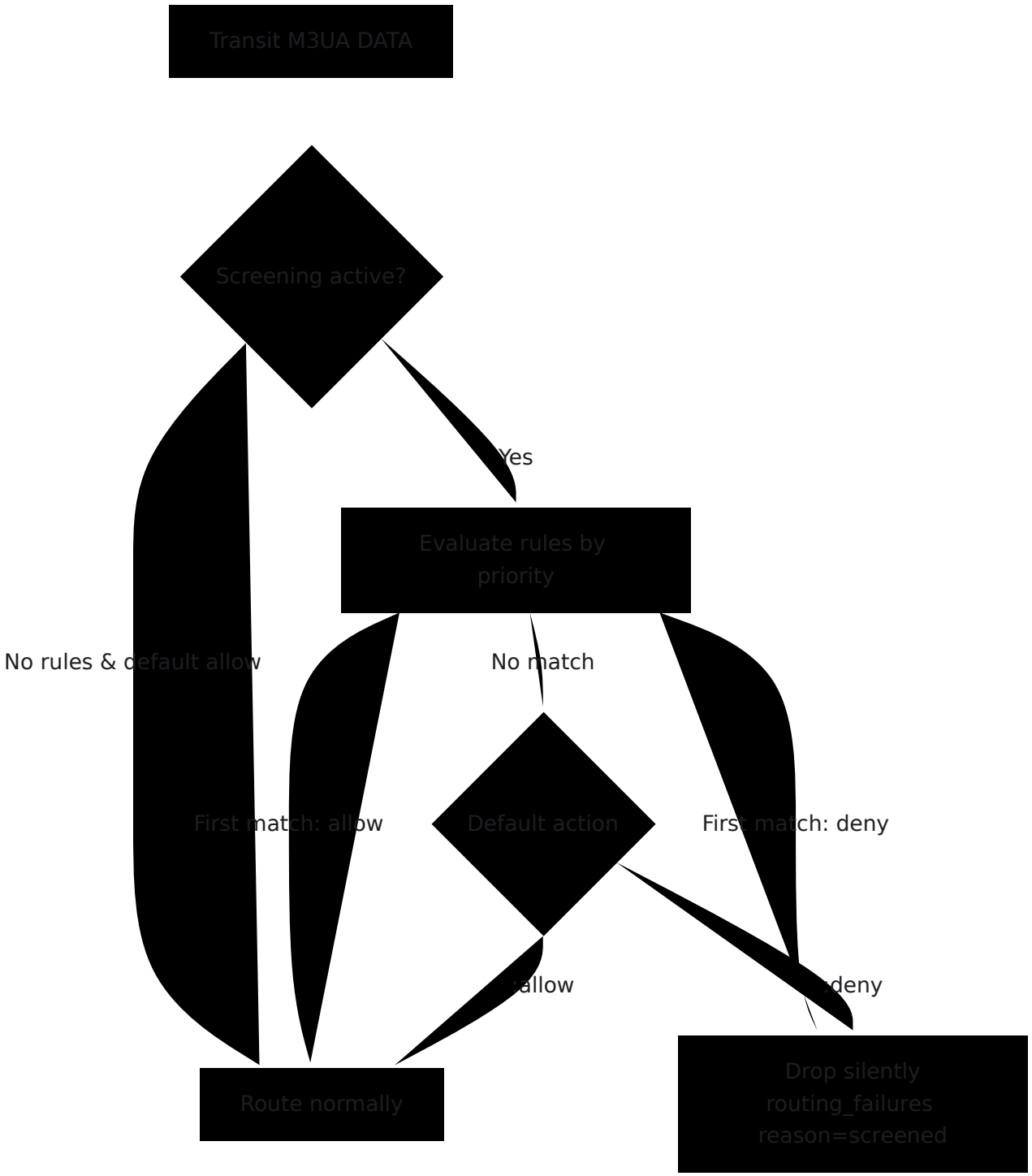
Each transit M3UA DATA message is evaluated against an ordered set of rules. A rule matches on any combination of the attributes below (an unset attribute is a wildcard):

Attribute	Matches
Source peer	The peer the message arrived from
OPC	Originating Point Code
DPC	Destination Point Code
Service Indicator	MTP3 SI
Called-Party GTA prefix	SCCP Called Party Global Title (prefix match)
Calling-Party GTA prefix	SCCP Calling Party Global Title (prefix match)
SSN	SCCP Subsystem Number

The **first matching enabled rule** (lowest `priority` value first) decides the outcome — `allow` or `deny`. If no rule matches, the configurable **default action** applies (`config :omniss7, :screening_default_action`, default `:allow`).

A denied message is **dropped silently** — no UDTS/return is generated (a firewall must not leak the block back to the sender) — and counted as `m3ua_stp_routing_failures_total{reason="screened"}`.

Behaviour



Screening is skipped entirely (zero hot-path cost) when no rules are configured **and** the default action is `:allow`; it becomes active as soon as a rule exists or the default action is set to `:deny`.

Default Action

Parameter	Type	Default	Description
`screening_default_action`	`:allow` | `:deny`	`:allow`	Action applied when no rule matches. Set to `:deny` for a default-deny (allowlist) posture.

```
config :omniss7,
  # Block everything except what rules explicitly allow.
  screening_default_action: :deny
```

Rules are stored in the `ss7_screen_rule` Mnesia table and managed programmatically (each rule carries a priority, the match attributes above, an `allow`/`deny` action and an `enabled` flag). See the Configuration Reference for the default-action parameter.

Route Management Features

Disabling Routes

Routes can be temporarily disabled without deleting them. This is useful for testing, maintenance, or traffic management.

Enabled Flag

Both Point Code and Global Title routes support an optional `enabled` flag:

```
config :omniss7,
  m3ua_routes: [
    # Active route
    %{
      dest_pc: 100,
      peer_id: 1,
      priority: 1,
      network_indicator: :international,
      enabled: true  # Route is active (default if omitted)
    },

    # Disabled route (not evaluated during routing)
    %{
      dest_pc: 200,
      peer_id: 2,
      priority: 1,
      network_indicator: :international,
      enabled: false  # Route is disabled
    }
  ],

  m3ua_gt_routes: [
    # Disabled GT route
    %{
      gt_prefix: "44",
      peer_id: 1,
      priority: 1,
      description: "UK numbers - temporarily disabled",
      enabled: false
    }
  ]
```

Default Behavior:

- If `enabled` is not specified, routes default to `enabled: true`
- Disabled routes are completely skipped during route lookup
- Use the Web UI to toggle routes on/off without editing config

Use Cases:

- Testing traffic flow changes

- Temporary maintenance windows
- A/B testing different routing paths
- Gradual rollout of new routes

DROP Routes: Preventing Routing Loops

DROP routes (with `peer_id: 0`) silently discard traffic instead of forwarding it. This prevents routing loops and enables advanced traffic filtering.

Configuring DROP Routes

```
config :omniss7,
  m3ua_routes: [
    # DROP route for specific point code
    %{
      dest_pc: 999,
      peer_id: 0,        # peer_id=0 means DROP
      priority: 1,
      network_indicator: :international
    }
  ],

  m3ua_gt_routes: [
    # DROP route for GT prefix
    %{
      gt_prefix: "999",
      peer_id: 0,        # peer_id=0 means DROP
      priority: 99,
      description: "Block test range"
    }
  ]
```

How DROP Routes Work

When a message matches a DROP route:

1. The routing engine identifies `peer_id: 0`
2. The message is **silently discarded** (not forwarded)

3. An **INFO log** is generated: `"DROP route matched for DPC 999"` or `"DROP route matched for GT 999"`
4. The routing lookup returns `{:error, :dropped}`

Important: Dropped traffic is logged at INFO level for monitoring and troubleshooting.

Common Use Case: Prefix Whitelisting

One of the most powerful uses of DROP routes is **prefix whitelisting** - allowing only specific numbers within a large range while blocking all others.

The Pattern:

1. Create a DROP route for the entire prefix with **high priority number** (e.g., 99)
2. Create specific allow routes for individual numbers with **low priority numbers** (e.g., 1)
3. Since lower priority numbers are evaluated first, allowed routes match before the DROP route
4. Any number not explicitly allowed gets caught by the DROP route

Example Scenario:

You have a GT prefix `1234` that represents a range of 10,000 numbers (1234000000 - 1234999999), but you only want to route 3 specific numbers: `1234567890`, `1234555000`, and `1234111222`.

```
config :omniss7,
  m3ua_gt_routes: [
    # DROP route with HIGH priority number (evaluated last)
    %{
      gt_prefix: "1234",
      peer_id: 0,              # DROP
      priority: 99,            # High number = low priority =
evaluated last
      description: "Block all 1234* except whitelisted numbers"
    },

    # Specific allow routes with LOW priority numbers (evaluated
first)
    %{
      gt_prefix: "1234567890",
      peer_id: 1,              # Route to peer 1
      priority: 1,             # Low number = high priority =
evaluated first
      description: "Allowed number 1"
    },
    %{
      gt_prefix: "1234555000",
      peer_id: 1,
      priority: 1,
      description: "Allowed number 2"
    },
    %{
      gt_prefix: "1234111222",
      peer_id: 1,
      priority: 1,
      description: "Allowed number 3"
    }
  ]
```

Routing Behavior:

Incoming GT	Matching Routes	Selected Route	Action
1234567890	✅ "1234567890" (priority 1) ✅ "1234" DROP (priority 99)	"1234567890" (most specific, highest priority)	**Routed to peer 1**
1234555000	✅ "1234555000" (priority 1) ✅ "1234" DROP (priority 99)	"1234555000" (most specific, highest priority)	**Routed to peer 1**
1234111222	✅ "1234111222" (priority 1) ✅ "1234" DROP (priority 99)	"1234111222" (most specific, highest priority)	**Routed to peer 1**
1234999999	✅ "1234" DROP (priority 99)	"1234" DROP (only match)	**Dropped + logged**
1234000000	✅ "1234" DROP (priority 99)	"1234" DROP (only match)	**Dropped + logged**

Result:

- ✅ Only 3 specific numbers are routed to peer 1
- ✅ All other `1234*` numbers are silently dropped
- ✅ All dropped traffic is logged for monitoring

Logs Generated:

```
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for GT 1234000000
```

DROP Routes for Point Codes

The same whitelist pattern works for Point Code routing:

```
config :omniss7,
  m3ua_routes: [
    # DROP entire range /8 (64 point codes: 1000-1063)
    %{
      dest_pc: 1000,
      peer_id: 0,
      priority: 99,
      mask: 8,
      network_indicator: :international
    },

    # Allow specific PCs
    %{dest_pc: 1010, peer_id: 1, priority: 1, network_indicator:
:international},
    %{dest_pc: 1020, peer_id: 1, priority: 1, network_indicator:
:international},
    %{dest_pc: 1030, peer_id: 1, priority: 1, network_indicator:
:international}
  ]
```

Result: Only PCs 1010, 1020, and 1030 are routed. All other PCs in the 1000-1063 range are dropped.

Monitoring DROP Routes

Check Logs:

```
# Monitor for dropped traffic
tail -f logs/app.log | grep "DROP route matched"

# Expected output:
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for DPC 1050
```

Via Web UI:

- Navigate to **System Logs** tab
- Filter by `INFO` level

- Search for "DROP route matched"

Best Practices:

1. ⚠ Monitor logs regularly to ensure DROP routes aren't blocking legitimate traffic
2. □ Use descriptive `description` fields to document why routes are dropped
3. □ Use high priority numbers (90-99) for DROP routes to ensure they're catch-all routes
4. □ Test DROP route behavior before deploying to production
5. □ Set up alerts for unexpected increases in dropped traffic

---

Advanced Routing: SSN-Based Routing and Rewriting

Subsystem Numbers (SSN)

Subsystem Numbers identify the application layer:

- **SSN 6**: HLR (Home Location Register)
- **SSN 7**: VLR (Visitor Location Register)
- **SSN 8**: MSC (Mobile Switching Center) / SMSC (SMS Center)
- **SSN 9**: GMLC (Gateway Mobile Location Center)

SSN-Based Routing Example

Route SMS traffic to different HLR based on number prefix:

```
m3ua_gt_routes: [
  # Route SMS for UK numbers to UK HLR, rewrite SSN from 8 (SMSC)
to 6 (HLR)
  %{
    gt_prefix: "44",
    source_ssn: 8,                    # Match incoming SSN 8
(SMSC)
    peer_id: 1,
    dest_ssn: 6,                      # Rewrite to SSN 6 (HLR)
    priority: 1,
    description: "UK SMS to HLR"
  },

  # Route voice traffic for UK numbers (SSN 6) without rewriting
  %{
    gt_prefix: "44",
    source_ssn: 6,                    # Match incoming SSN 6 (HLR)
    peer_id: 1,
    dest_ssn: nil,                    # No SSN rewrite
    priority: 1,
    description: "UK voice traffic"
  }
]
```

Testing STP Routing Configuration

After configuring peers and routes, verify your configuration:

1. Check Peer Status

Via Web UI:

- Navigate to http://localhost
- Check M3UA Status page
- Verify peers show **Status: ACTIVE**

Via IEx Console:

```
# Get all peer statuses
SCTP.SocketHandler.get_peers_status()

# Expected output:
# [
#   %{peer_id: 1, name: "Partner_STP_West", status: :active,
point_code: 100, ...},
#   %{peer_id: 2, name: "Local_HLR", status: :active, point_code:
200, ...}
# ]
```

2. Test Point Code Routing

```
# Send test M3UA message to DPC 100
test_payload = <<1, 2, 3, 4>>  # Dummy payload
SCTP.SocketHandler.route_by_pc(100, test_payload, 0)

# Check logs for routing decision
# Expected log: "Routing message: OPC=... -> DPC=100 via peer 1"
```

3. Test Global Title Routing

```
# Look up GT route manually
M3UARouting.lookup_peer_by_gt("447712345678")

# Returns {:ok, peer, rewrites}. `peer` is the 13-element
m3ua_peer record;
# `rewrites` is a MAP of indicator rewrites (nil = keep original):
# {:ok, {:m3ua_peer, 3, "UK_Mobile_Peer", ...},
#       %{dest_ssn: nil, dest_tt: nil, dest_npi: nil, dest_nai:
nil}}

# Look up GT route with SSN (a route that rewrites SSN 8 → 6)
M3UARouting.lookup_peer_by_gt("555881234567", 8)

# The SSN rewrite appears in the rewrites map, NOT as a bare
integer:
# {:ok, {:m3ua_peer, 4, "SMS_HLR_Peer", ...},
#       %{dest_ssn: 6, dest_tt: nil, dest_npi: nil, dest_nai:
nil}}
```

4. Monitor Routing Metrics

Access Prometheus metrics at `/metrics`

Key metrics:

```
# Messages received per peer
m3ua_stp_messages_received_total{peer_name="Partner_STP_West",point_c
1523

# Messages sent per peer
m3ua_stp_messages_sent_total{peer_name="Local_HLR",point_code="200"}

# Routing failures (by reason)
m3ua_stp_routing_failures_total{reason="no_route"} 5
m3ua_stp_routing_failures_total{reason="no_gt_route"} 2
m3ua_stp_routing_failures_total{reason="no_active_peer"} 1
m3ua_stp_routing_failures_total{reason="prohibited"} 3
m3ua_stp_routing_failures_total{reason="hcounter_violation"} 4
m3ua_stp_routing_failures_total{reason="screened"} 7
```

STP Metrics and Monitoring

Available Metrics

Per-Peer Traffic Metrics:

- `m3ua_stp_messages_received_total` - Total messages received from each peer
  - Labels: `peer_name`, `point_code`
- `m3ua_stp_messages_sent_total` - Total messages forwarded to each peer
  - Labels: `peer_name`, `point_code`

Routing Failure Metrics:

- `m3ua_stp_routing_failures_total` - Count of routing failures by reason
  - Labels: `reason`

`reason`	Meaning
`no_route`	No Point Code route found for the destination
`no_gt_route`	No Global Title route found, and PC routing also failed
`no_active_peer`	A route matched, but its peer is unknown/not active
`prohibited`	The destination is transfer-prohibited (a TFP was received, Q.704 §13)
`hcounter_violation`	The SCCP hop counter reached zero on an XUDT/LUDT (Q.714 §2.8.6)

Metric Interpretation

- **High message counts**: Indicates active traffic flow
- **Routing failures**: Indicates missing routes, an unreachable destination, or a looping message
  - `no_route`: No Point Code route found for destination
  - `no_gt_route`: No Global Title route found, and PC routing also failed
  - `no_active_peer`: Route exists but the target peer is down/unknown
  - `prohibited`: Destination marked transfer-prohibited by signalling-route management
  - `hcounter_violation`: Hop counter exhausted; the message was discarded (and returned if return-on-error was set)

Troubleshooting with Metrics

Scenario: No traffic reaching destination

1. Check if messages are being received:

```
m3ua_stp_messages_received_total{peer_name="Source_Peer"} > 0
```

2. Check if messages are being sent:

```
m3ua_stp_messages_sent_total{peer_name="Dest_Peer"} > 0
```

3. Check for routing failures:

```
m3ua_stp_routing_failures_total{reason="no_route"} > 0
```

Solution: If routing failures are high, add missing routes in configuration.

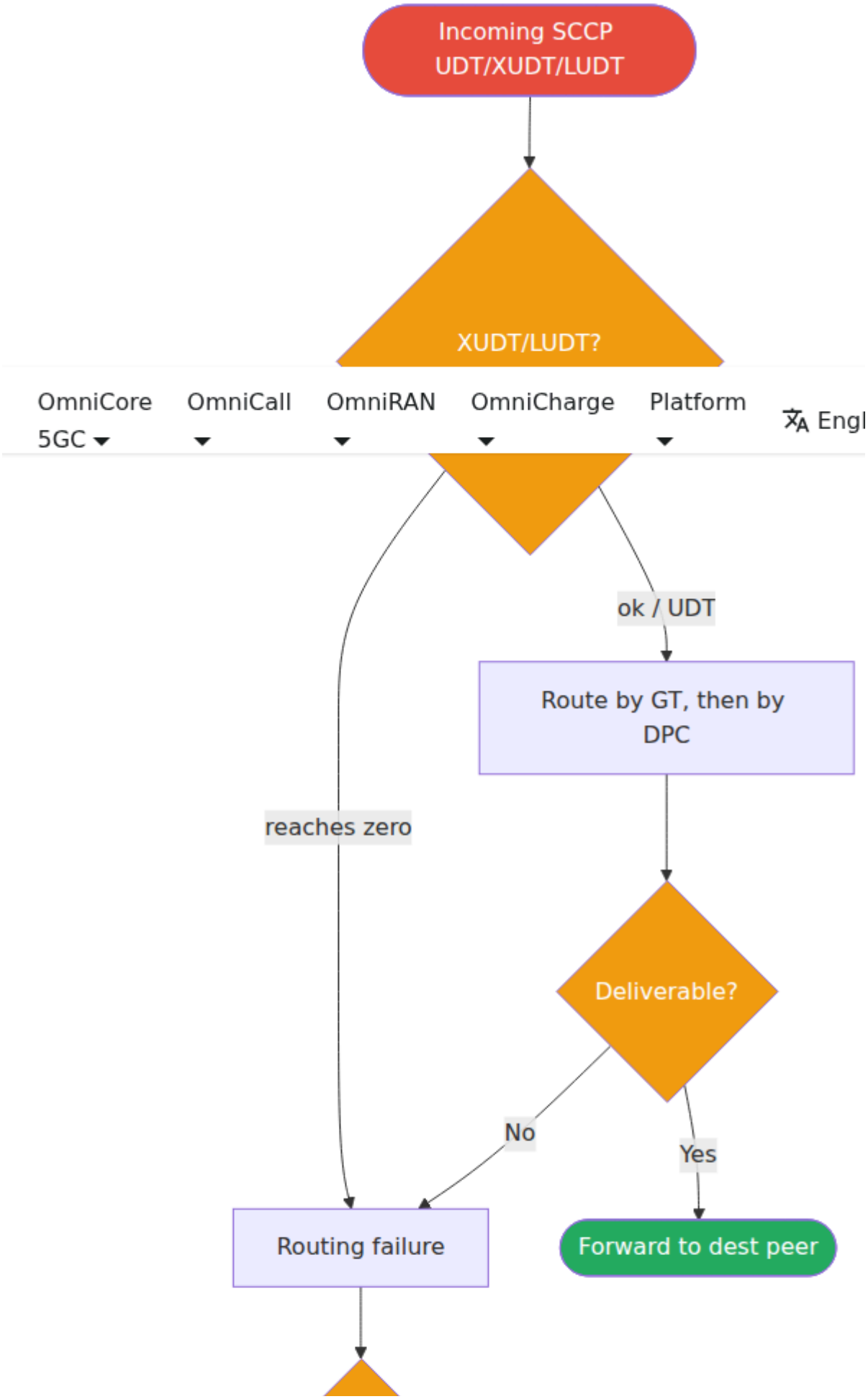
M3UA Peer Status Monitoring

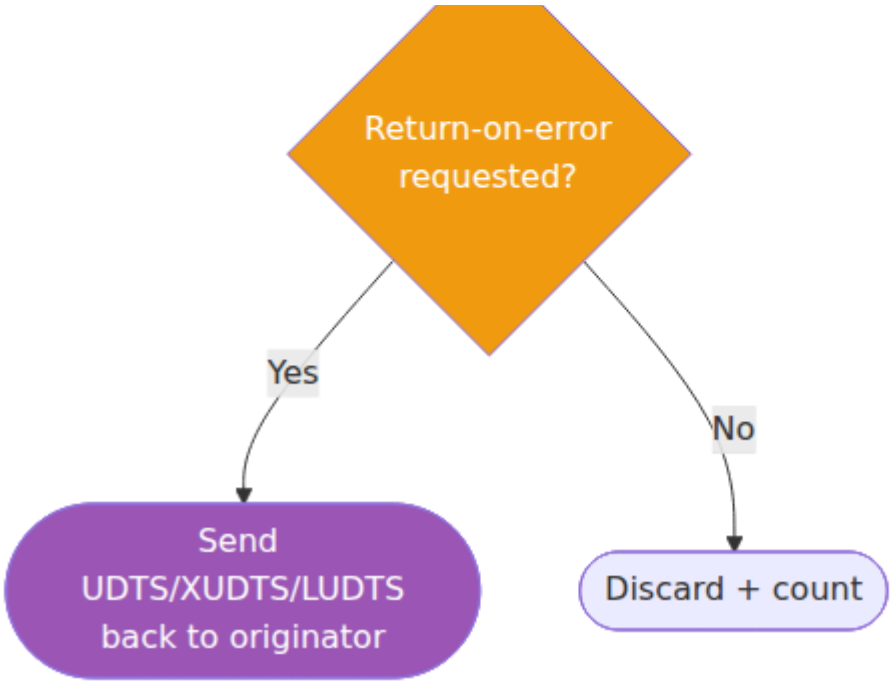
Understanding M3UA

M3UA (MTP3 User Adaptation Layer) is a protocol that allows SS7 signaling to be transported over IP networks using SCTP.

M3UA Connection States

M3UA connections progress through several states:


Incoming SCCP UDT/XUDT/LUDT
XUDT/LUDT?
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
Engl
ok / UDT
Route by GT, then by DPC
reaches zero
Deliverable?
No
Yes
Routing failure
Forward to dest peer



State Descriptions:

- **DOWN** - No SCTP connection
- **CONNECTING** - SCTP connection in progress
- **ASPUP_SENT** - Waiting for ASPUP acknowledgment
- **INACTIVE** - ASP is up but not actively carrying traffic (can transition to ACTIVE via ASPAC)
- **ASPAC_SENT** - Waiting for ASPAC acknowledgment
- **ACTIVE** - Ready for traffic, fully operational
- **ASPDOWN_SENT** - Graceful shutdown in progress

Monitoring M3UA Peers via Web UI

The Web UI provides real-time monitoring of M3UA peer connections.

Accessing M3UA Status Page:

1. Navigate to the Web UI home page
2. Click on "M3UA Status" in the navigation menu
3. The page auto-refreshes every second

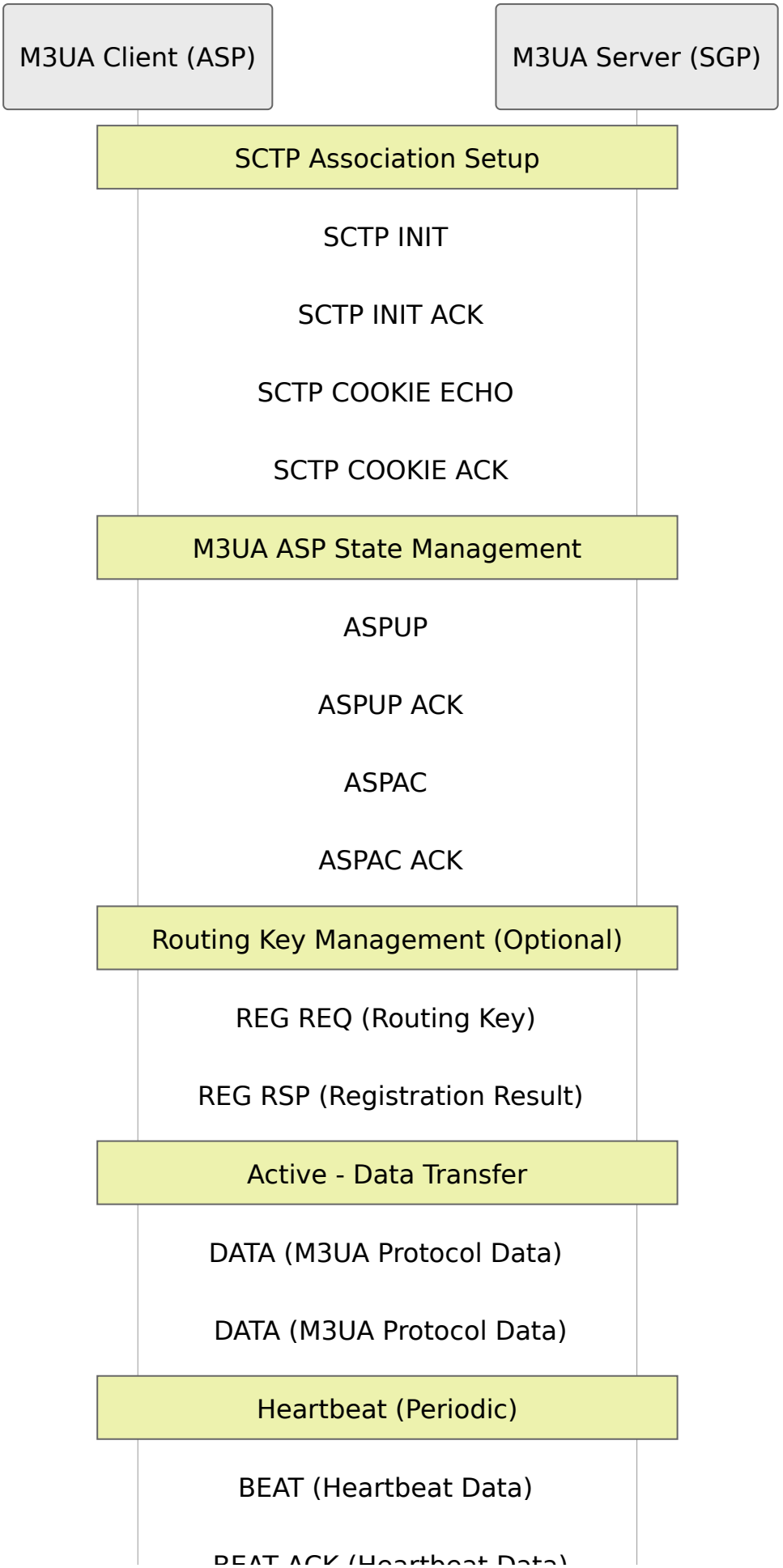
M3UA Status Table:

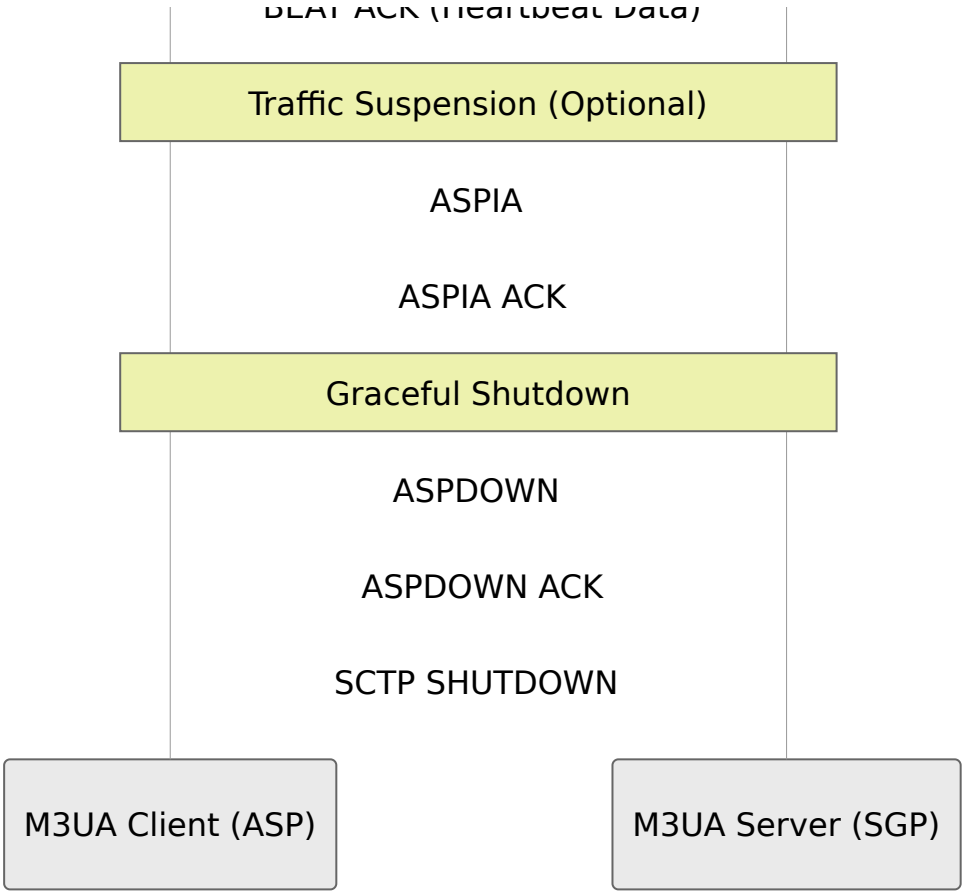
Column	Description
Name	Connection name (e.g., `testASP`)
PID	Process identifier
Status	UP (green) or DOWN (red)
ASP State	Current M3UA state (e.g., ACTIVE, INACTIVE)
Assoc/SCTP	SCTP association state
Local	Local IP:Port
Remote	Remote IP:Port
RC	Routing Context ID

Status Indicators:

- **Green (UP)** - Connection is active and healthy
- **Red (DOWN)** - Connection is down or unavailable
- **ASP State** - Shows current M3UA connection state
- **Assoc/SCTP** - Shows SCTP association status

M3UA Message Flow


M3UA Client (ASP)
M3UA Server (SGP)
SCTP Association Setup
SCTP INIT
SCTP INIT ACK
SCTP COOKIE ECHO
SCTP COOKIE ACK
M3UA ASP State Management
ASPUP
ASPUP ACK
ASPAC
ASPAC ACK
Routing Key Management (Optional)
REG REQ (Routing Key)
REG RSP (Registration Result)
Active - Data Transfer
DATA (M3UA Protocol Data)
DATA (M3UA Protocol Data)
Heartbeat (Periodic)
BEAT (Heartbeat Data)



RFC 4666 M3UA Message Support

OmniSS7 implements the full set of M3UA message types defined in RFC 4666:

Class	Type	Message	Direction	Behaviour
0 (MGMT)	0	ERR	Both	Logs error code and diagnostic info
0 (MGMT)	1	NTFY	Both	Logs notification
1 (XFER)	1	DATA	Both	Routes payload via PC or GT
2 (SSNM)	1	DUNA	Receive	Logs destination unavailable
2 (SSNM)	2	DAVA	Receive	Logs destination available
2 (SSNM)	3	DAUD	Receive	Responds per destination state: DAVA for reachable PCs, DUNA for unreachable/prohibited PCs
2 (SSNM)	4	SCON	Receive	Logs congestion
2 (SSNM)	5	DUPU	Receive	Logs user part unavailable
2 (SSNM)	6	DRST	Receive	Logs destination restricted
3 (ASPSM)	1	ASPUP	Both	Sends/receives ASPUP ACK
3 (ASPSM)	2	ASPDN	Both	Sends ASPDN ACK, tears down peer routes

Class	Type	Message	Direction	Behaviour
3 (ASPSM)	3	BEAT	Both	Echoes heartbeat data in BEAT ACK
3 (ASPSM)	4	ASPUP ACK	Receive	Transitions peer to UP
3 (ASPSM)	5	ASPDN ACK	Receive	Transitions peer to DOWN
3 (ASPSM)	6	BEAT ACK	Receive	Acknowledges heartbeat
4 (ASPTM)	1	ASPAC	Both	Sends/receives ASPAC ACK, activates routes
4 (ASPTM)	2	ASPIA	Both	Sends ASPIA ACK, transitions peer to INACTIVE
4 (ASPTM)	3	ASPAC ACK	Receive	Transitions peer to ACTIVE
4 (ASPTM)	4	ASPIA ACK	Receive	Transitions peer to INACTIVE
9 (RKM)	1	REG REQ	SGP receive	Parses routing key, responds with REG RSP
9 (RKM)	2	REG RSP	ASP receive	Logs registration result
9 (RKM)	3	DEREG REQ	SGP receive	Responds with DEREG RSP
9 (RKM)	4	DEREG RSP	ASP receive	Logs deregistration result

Heartbeat (BEAT/BEAT ACK): When the STP receives a BEAT message, it echoes the heartbeat data back in a BEAT ACK. This allows peers to detect link failures without relying solely on SCTP-level timeouts.

Routing Key Management (RKM): When an ASP sends a REG REQ, the SGP parses the routing key (local RK identifier, routing context, traffic mode, DPC) and responds with a REG RSP containing the registration status. This enables dynamic route provisioning without requiring static configuration.

ASP Down (ASPDN): When the SGP receives an ASPDN, it responds with ASPDN ACK, marks the peer as DOWN, and removes associated routes from the routing table.

ASP Inactive (ASPIA): When the SGP receives an ASPIA, it responds with ASPIA ACK and transitions the peer to INACTIVE, pausing traffic without tearing down the association.

M3UA Routing Context Handling

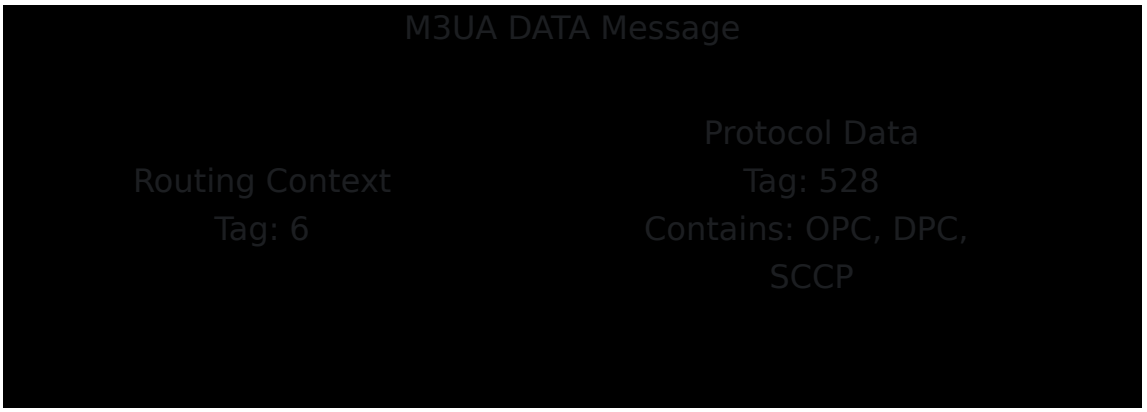
The **Routing Context (RC)** is a 32-bit identifier in M3UA that identifies the Application Server (AS) context at each endpoint. Understanding how RC is handled at different points in the signaling path is important for correct network configuration.

What is Routing Context?

Per RFC 4666 Section 3.3.1, the Routing Context:

- Uniquely identifies an Application Server (AS) at an endpoint
- Is carried in M3UA DATA messages (parameter tag 6)
- Allows a Signaling Gateway to determine which AS should receive a message
- Is meaningful per-hop, not end-to-end

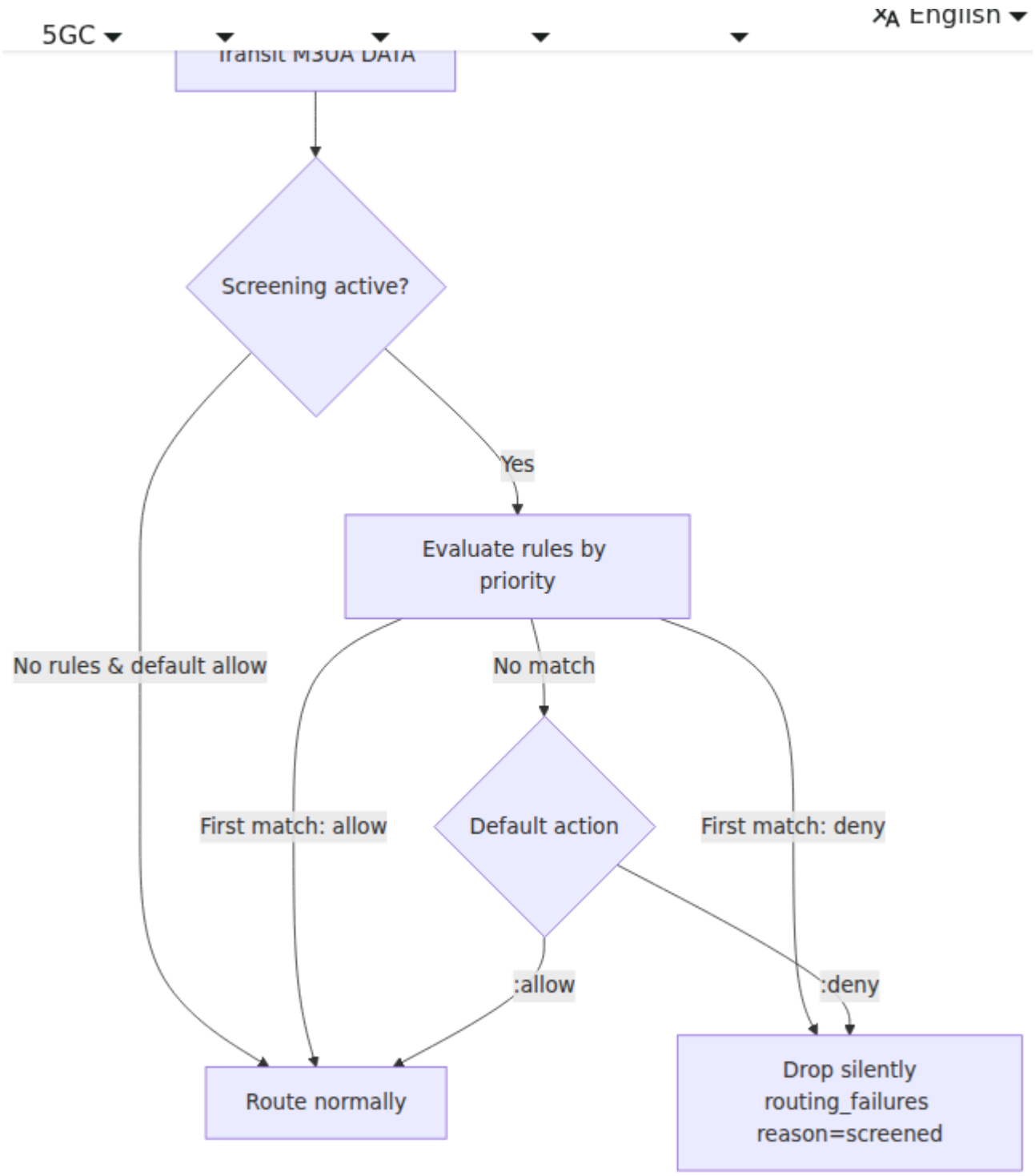
Routing Context in M3UA Protocol



The Routing Context parameter (tag 6) appears in M3UA DATA messages alongside the Protocol Data parameter (tag 528) which contains the actual MTP3/SCCP payload.

How OmniSS7 Handles Routing Context

OmniSS7 handles Routing Context differently depending on the component and message direction:



MAP Client and CAP Client

For application-layer clients (MAP and CAP), routing context is handled as follows:

Scenario	RC Handling	Configuration
Response to incoming request	Mirror the incoming RC back in the response	Automatic - no configuration needed
Client-initiated request (SRI, PRN, IDP, etc.)	Use configured RC from application config	Set `routing_context` in `map_client_m3ua` or `cap_client_m3ua`
No RC in incoming request	Omit RC from response	Maintains backward compatibility
No RC configured for client requests	Omit RC from outgoing message	Maintains backward compatibility

Configuration Example:

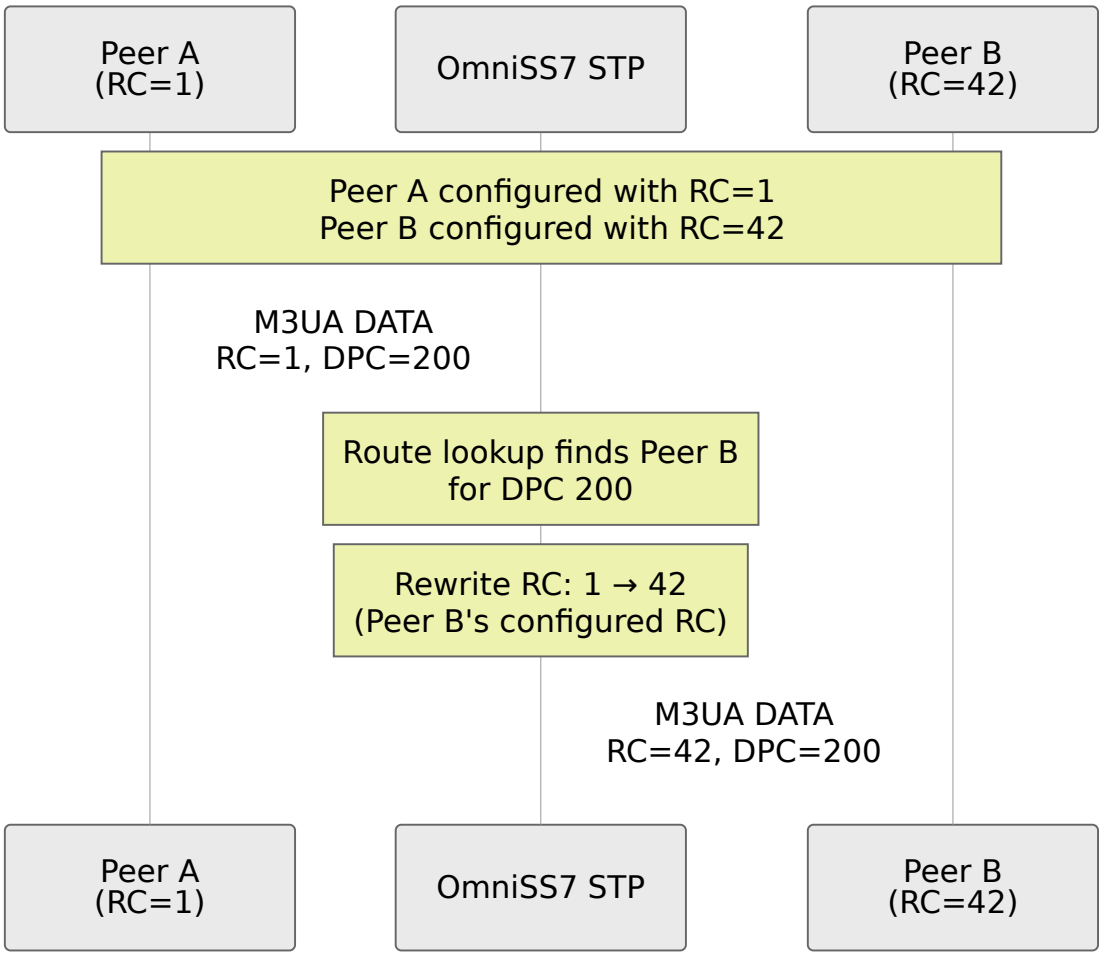
```
config :omniss7,
  # MAP Client RC configuration
  map_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 0, 0, 1},
    local_port: 2905,
    remote_ip: {10, 0, 0, 10},
    remote_port: 2905,
    routing_context: 1    # Used for client-initiated MAP requests
(SRI, PRN, etc.)
  },

  # CAP Client RC configuration
  cap_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 0, 0, 1},
    local_port: 2906,
    remote_ip: {10, 0, 0, 20},
    remote_port: 2906,
    routing_context: 2    # Used for client-initiated CAP requests
(InitialDP, etc.)
  }
```

STP Routing Context Rewriting

When the STP forwards messages between peers, it **rewrites** the Routing Context to the destination peer's configured RC. This is correct behavior per RFC 4666 because:

1. RC identifies the AS context at each endpoint
2. Different peers may be associated with different AS contexts
3. The STP translates RC values when routing between different ASs

STP RC Rewriting Flow:



Configuration:

Each peer's routing context is configured in the peer definition:

```
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "Network_A_STP",
      role: :server,
      remote_ip: {10, 0, 0, 100},
      remote_port: 2905,
      routing_context: 1,        # RC expected by/from this peer
      point_code: 100,
      network_indicator: :international
    },
    %{
      peer_id: 2,
      name: "Network_B_HLR",
      role: :client,
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 200},
      remote_port: 2905,
      routing_context: 42,       # RC expected by this peer
      point_code: 200,
      network_indicator: :international
    }
  ]
```

When a message arrives from Peer 1 (RC=1) destined for Peer 2, the STP:

1. Routes the message based on DPC (200 → Peer 2)
2. Rewrites RC from 1 to 42 (Peer 2's configured RC)
3. Forwards the message to Peer 2

RC Handling Summary

Component	Incoming Request	Outgoing Response	Client-Initiated
MAP Client	Extract RC	Mirror incoming RC	Use configured RC
CAP Client	Extract RC	Mirror incoming RC	Use configured RC
STP	Extract RC	N/A	Rewrite to dest peer's RC

When RC is Omitted

If a peer has no `routing_context` configured (or it's set to `nil`):

- **MAP/CAP Client responses**: RC parameter is omitted from response
- **MAP/CAP Client requests**: RC parameter is omitted from request
- **STP forwarding**: RC is not rewritten (original RC preserved or omitted)

This maintains backward compatibility with peers that don't use or expect routing context.

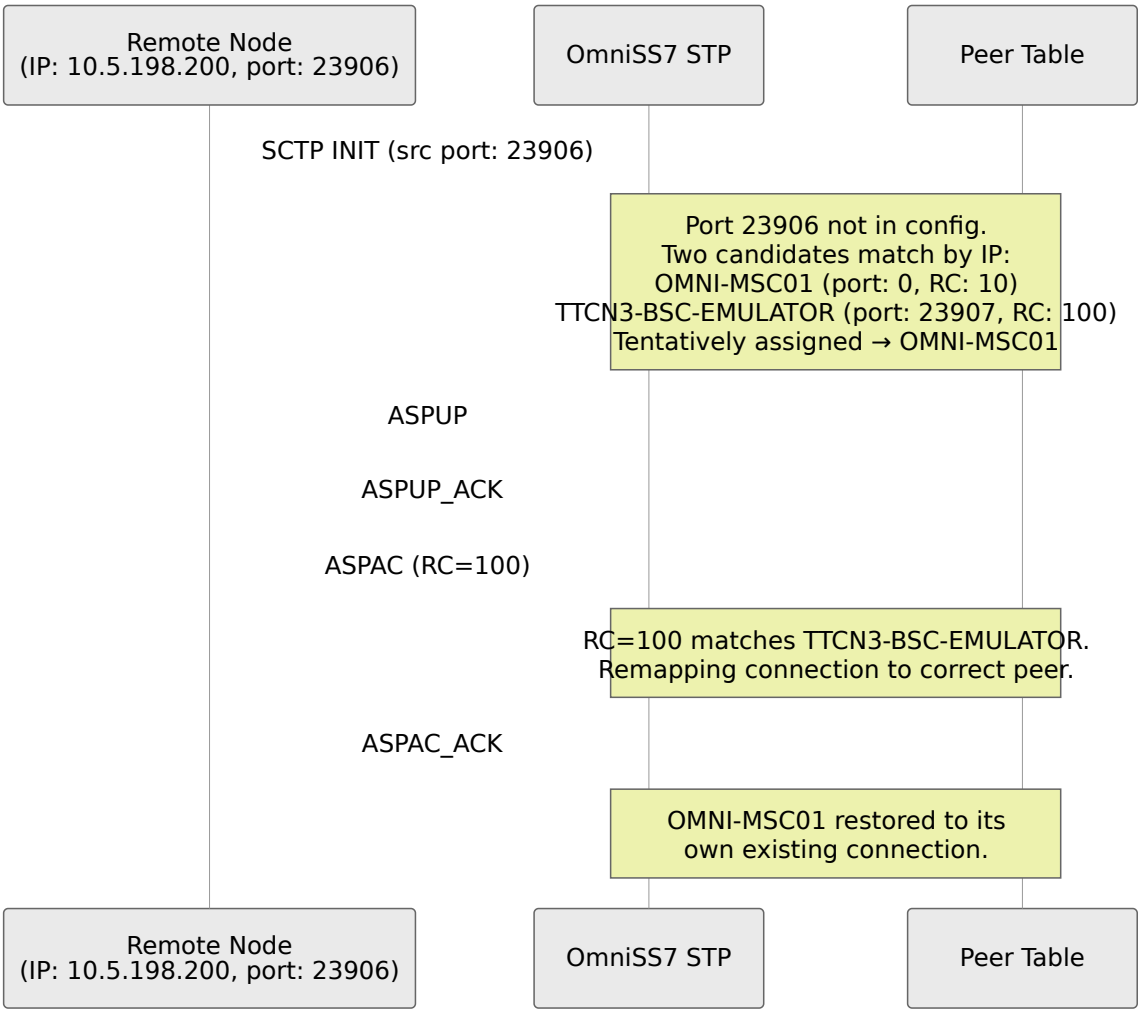
Same-IP Multi-Peer Disambiguation

When two configured peers share the same remote IP address — for example, a production MSC and a test emulator running on the same host — OmniSS7 cannot determine the correct peer identity from the source IP alone at the moment an inbound SCTP connection arrives. A peer configured with `remote_port: 0` (wildcard) will match any connection from that IP, which can cause a new connection from the other peer to be tentatively assigned to the wrong entry.

OmniSS7 resolves this automatically using the **M3UA ASPAC routing context**. The connection is accepted and tentatively assigned to the best-matching peer (specific port match is preferred over wildcard), and when the

remote end sends an `ASPAC` message carrying a routing context, the system compares that RC against each peer's configured `routing_context`. If the RC matches a different peer, the connection is silently remapped to the correct one.

How It Works



Log Output

When multiple peers match an inbound connection, OmniSS7 logs all candidates so operators can verify disambiguation is working correctly:

```
Inbound connection from 10.5.198.200:23906 matches peers: ["OMNI-
MSC01", "TTCN3-BSC-EMULATOR"] — tentatively using "OMNI-MSC01"
until ASPAC RC confirms identity
```

After the ASPAC routing context is received:

```
ASPAC RC=100 matches peer 9 (TTCN3-BSC-EMULATOR), but connection
was assigned to peer 8 (OMNI-MSC01) — remapping
```

Configuration Requirements

For same-IP peer disambiguation to work correctly, **each peer must have a unique `routing_context` configured**. If two peers on the same IP share the same RC (or both have `nil`), the remapping cannot determine which peer is correct.

```
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 8,
      name: "OMNI-MSC01",
      role: :server,
      remote_ip: {10, 5, 198, 200},
      remote_port: 0,              # Wildcard — accepts any source
port
      routing_context: 10,         # Must be unique across peers
sharing this IP
      point_code: 500,
      network_indicator: :national
    },
    %{
      peer_id: 9,
      name: "TTCN3-BSC-EMULATOR",
      role: :server,
      remote_ip: {10, 5, 198, 200},
      remote_port: 23907,          # Expected source port (may
differ for ephemeral ports)
      routing_context: 100,        # Different RC — used to
identify this peer after ASPAC
      point_code: 193,
      network_indicator: :national
    }
  ]
```

> **Note on ephemeral source ports**: Port matching is exact — if a peer is configured with `remote_port: 23907` but connects from port 23906, that

> peer is excluded from the initial match and the wildcard peer wins instead. This is expected. The ASPAC routing context is the authoritative identity signal; the configured `remote_port` on a peer is only used for the initial connection assignment heuristic, not for final identity. Ensure each peer has a distinct `routing_context` and the remapping will resolve correctly regardless of which source port the remote uses.

Troubleshooting M3UA Connections

Issue: Connection Won't Establish

Symptoms:

- Status shows DOWN
- No SCTP association

Checks:

1. Verify network connectivity: `ping remote_ip`
2. Check firewall allows SCTP (protocol 132)
3. Verify remote STP/SGP is listening on correct port
4. Check `remote_ip` and `remote_port` in config
5. Review application logs for SCTP errors

Issue: Connection Established but ASP Not Active

Symptoms:

- SCTP association exists
- ASP state stuck in INACTIVE or ASPUP_SENT

Checks:

1. Verify routing context matches remote configuration
2. Check remote STP accepts your point code
3. Review logs for ASPUP/ASPAC rejections
4. Verify no authentication/security requirements

Issue: Data Not Flowing

Symptoms:

- ASP state shows ACTIVE
- No messages being routed

Checks:

1. Verify routing context in messages
2. Check SCCP addressing (GT format, SSN values)
3. Verify routing tables configured correctly
4. Review `/events` page for SCCP errors
5. Check point code routing at STP level

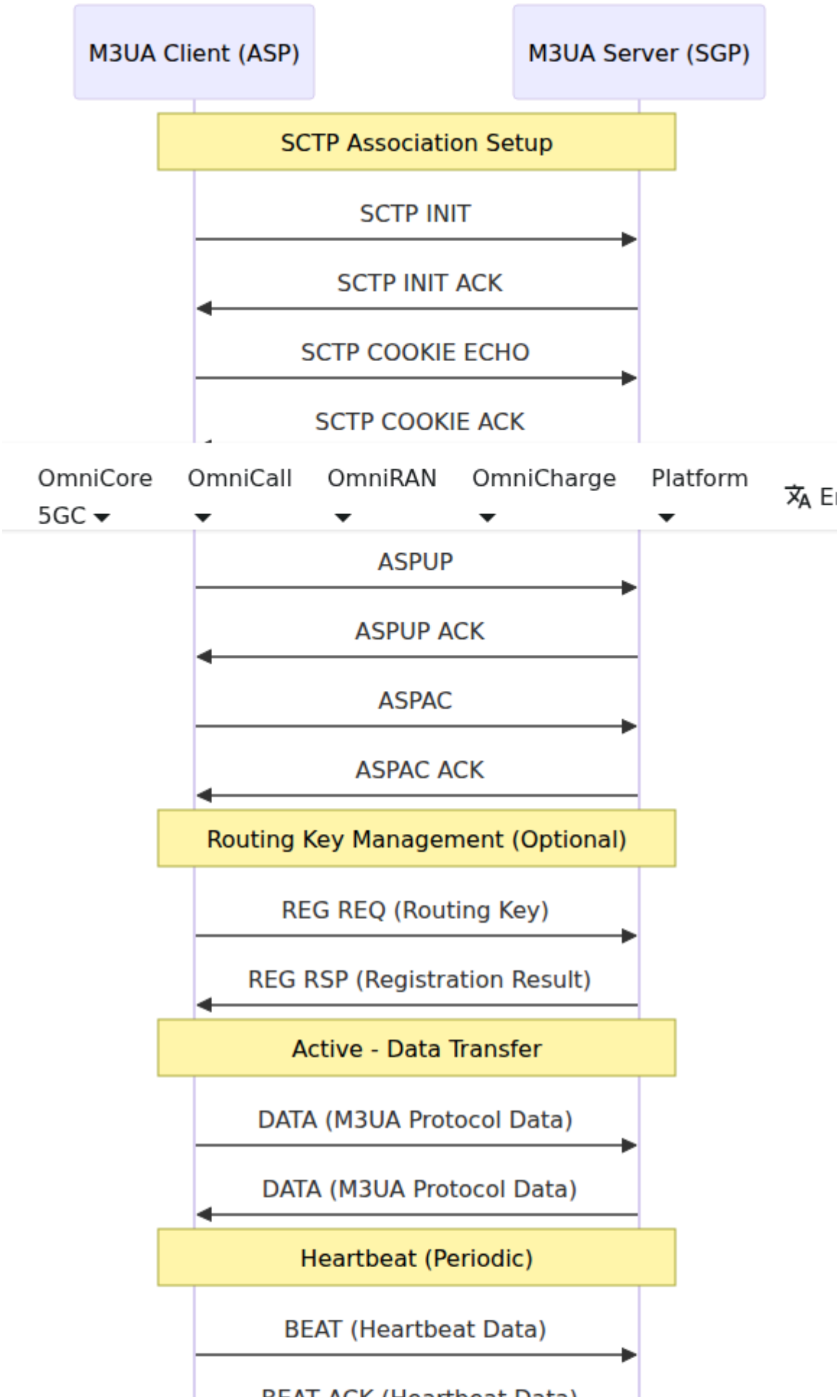
M2PA Peer Status Monitoring

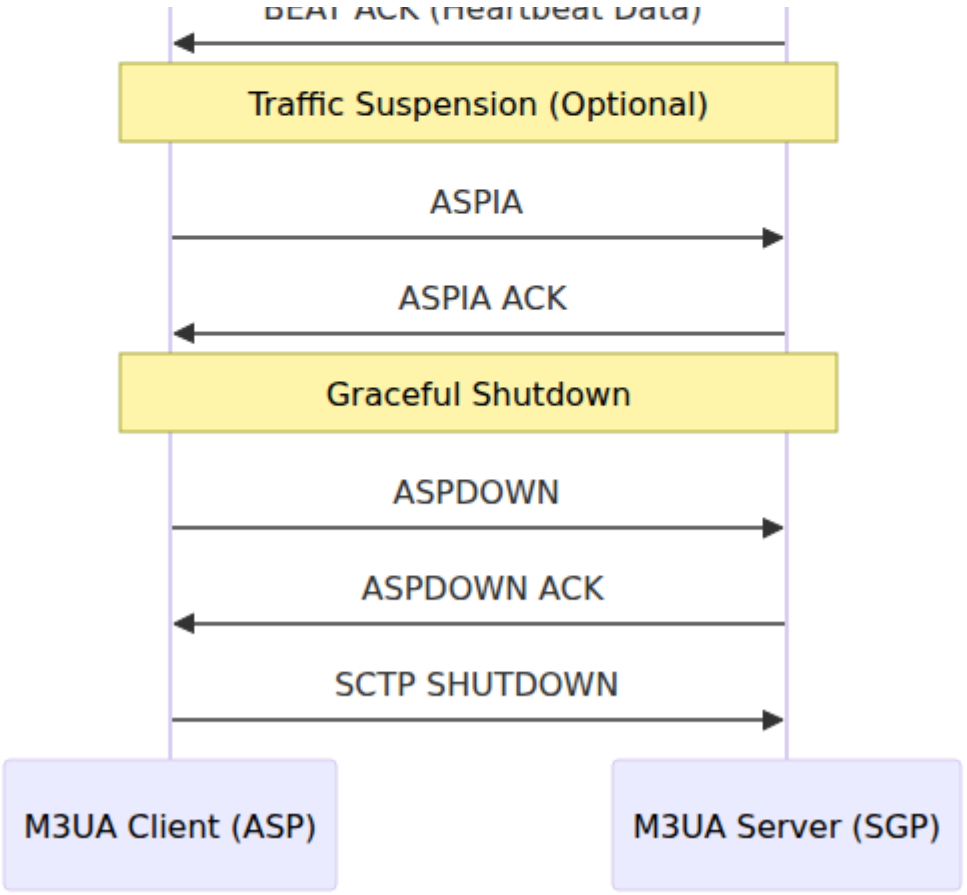
Understanding M2PA

M2PA (MTP2 User Peer-to-Peer Adaptation Layer) is a protocol defined in RFC 4165 that provides point-to-point MTP3 message transport over SCTP. Unlike M3UA which uses an ASP/SGP architecture, M2PA provides peer-to-peer links similar to traditional TDM SS7 links.

M2PA Link States

M2PA links progress through several states during establishment:


M3UA Client (ASP)
M3UA Server (SGP)
SCTP Association Setup
SCTP INIT
SCTP INIT ACK
SCTP COOKIE ECHO
SCTP COOKIE ACK
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
ASPUP
ASPUP ACK
ASPAC
ASPAC ACK
Routing Key Management (Optional)
REG REQ (Routing Key)
REG RSP (Registration Result)
Active - Data Transfer
DATA (M3UA Protocol Data)
DATA (M3UA Protocol Data)
Heartbeat (Periodic)
BEAT (Heartbeat Data)



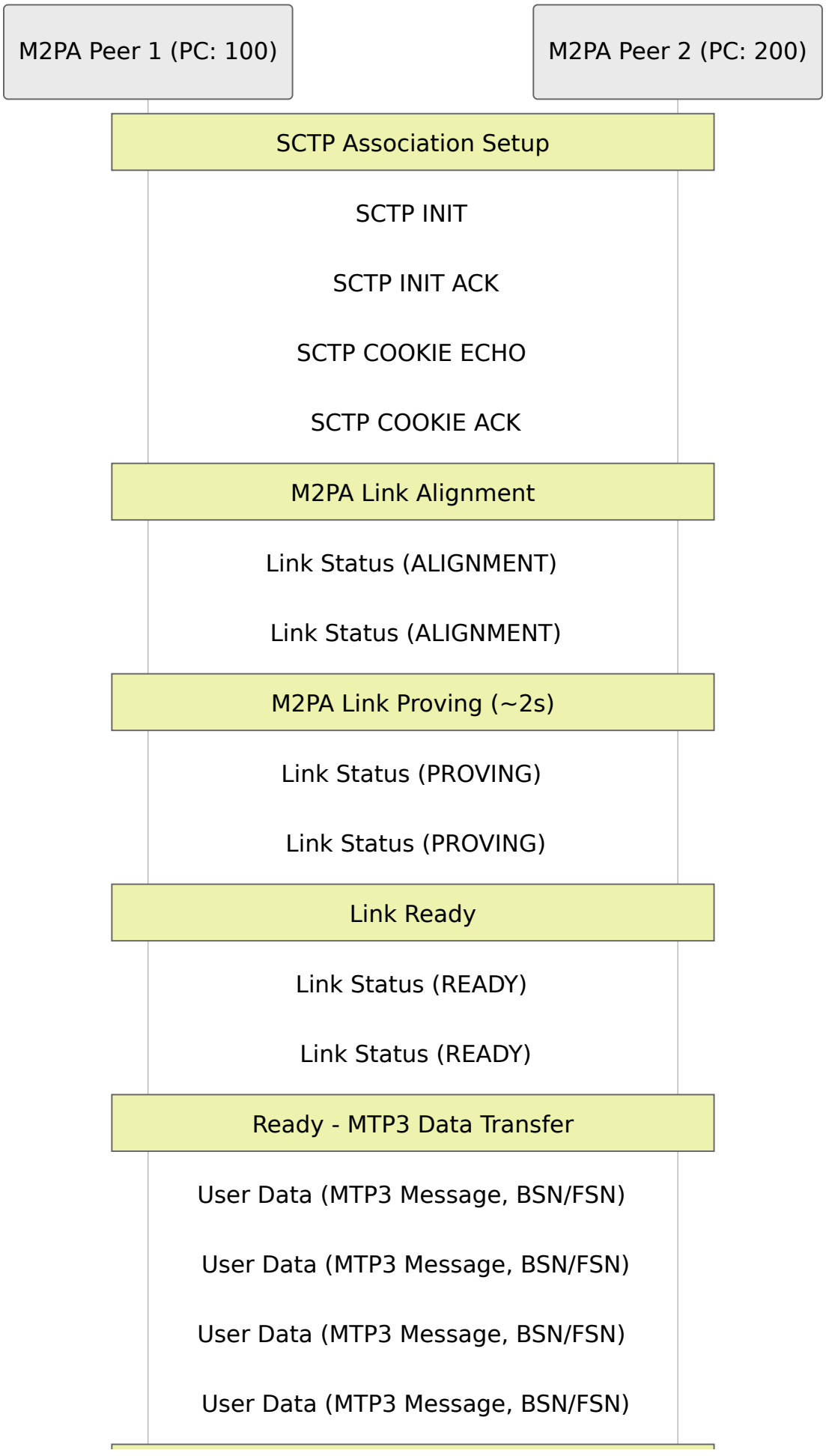
State Descriptions:

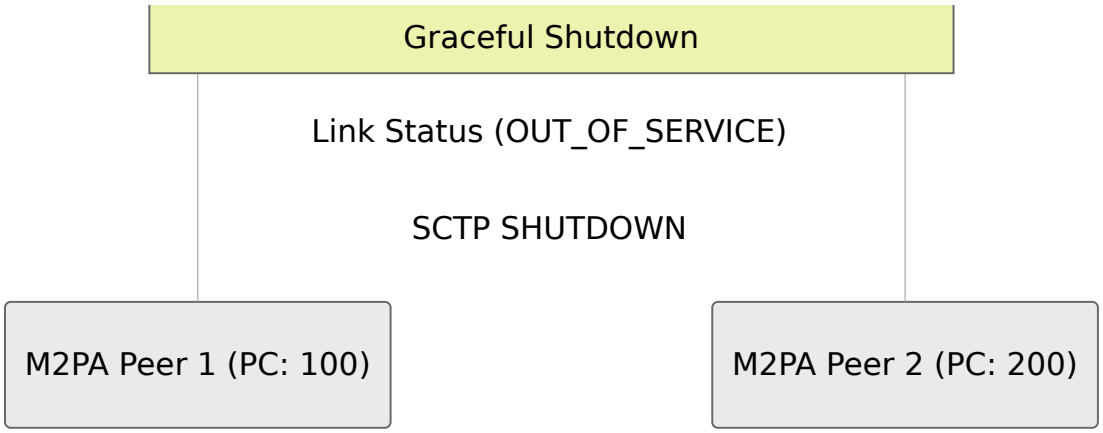
- **DOWN** - No SCTP connection, link inactive
- **CONNECTING** - SCTP association in progress
- **ALIGNMENT** - Link Status messages exchanged (~1 second)
- **PROVING** - Link proving period, testing link integrity (~2 seconds)
- **READY** - Link operational, ready for MTP3 user data transfer
- **ALIGNMENT (re-entry)** - Link status change requires re-alignment

Link State Progression:

1. **SCTP Connection**: Establishes SCTP association (DOWN → CONNECTING)
2. **Alignment**: Exchanges Link Status messages to synchronize (CONNECTING → ALIGNMENT)
3. **Proving**: Tests link reliability and sequence number synchronization (ALIGNMENT → PROVING)
4. **Ready**: Link becomes operational for data transfer (PROVING → READY)

M2PA Message Flow


M2PA Peer 1 (PC: 100)
M2PA Peer 2 (PC: 200)
SCTP Association Setup
SCTP INIT
SCTP INIT ACK
SCTP COOKIE ECHO
SCTP COOKIE ACK
M2PA Link Alignment
Link Status (ALIGNMENT)
Link Status (ALIGNMENT)
M2PA Link Proving (~2s)
Link Status (PROVING)
Link Status (PROVING)
Link Ready
Link Status (READY)
Link Status (READY)
Ready - MTP3 Data Transfer
User Data (MTP3 Message, BSN/FSN)
User Data (MTP3 Message, BSN/FSN)
User Data (MTP3 Message, BSN/FSN)
User Data (MTP3 Message, BSN/FSN)



Monitoring M2PA Peers via Web UI

The Web UI provides real-time monitoring of M2PA peer connections.

Accessing Routing Management Page:

1. Navigate to the Web UI home page
2. Click on "Routing Management" in the navigation menu
3. View the "M3UA/M2PA Peers" table

M2PA Peer Table:

Column	Description
Peer ID	Unique peer identifier
Name	Peer name (e.g., `M2PA_Link_STP_A`)
Protocol	Shows "M2PA" in green
Point Code	Local point code
Adj. PC	Adjacent peer point code
Local	Local IP:Port (typically port 3565)
Remote	Remote IP:Port
Status	Link state (e.g., READY, ALIGNMENT, DOWN)

Status Indicators:

- **READY (Green)** - Link is operational and passing traffic
- **ALIGNMENT (Yellow)** - Link is aligning, not yet ready
- **PROVING (Yellow)** - Link is in proving state
- **DOWN (Red)** - Link is down or unavailable

Troubleshooting M2PA Connections

Issue: Link Stuck in ALIGNMENT

Symptoms:

- Link state shows ALIGNMENT for extended period
- No progression to PROVING or READY

Checks:

1. Verify both sides are configured with correct point codes

2. Check SCTP firewall allows protocol 132
3. Verify `point_code` and `adjacent_point_code` are correctly set
4. Review application logs for Link Status message errors
5. Ensure remote peer is also in ALIGNMENT state

Issue: Link Stuck in PROVING

Symptoms:

- Link reaches PROVING but doesn't transition to READY
- Proving period exceeds 2-3 seconds

Checks:

1. Verify network stability (no packet loss)
2. Check for SCTP association errors
3. Review logs for sequence number mismatches
4. Ensure remote peer is also in PROVING state
5. Verify SCTP multihoming isn't causing routing issues

Issue: Link Flapping (DOWN ↔ READY)

Symptoms:

- Link repeatedly cycles between READY and DOWN
- Frequent re-alignments

Checks:

1. Check network connectivity stability
2. Verify SCTP heartbeat settings
3. Review firewall session timeout settings
4. Check for MTU/fragmentation issues
5. Verify no duplicate IP addresses

Issue: Data Not Flowing

Symptoms:

- Link state shows READY
- No MTP3 messages being transferred

Checks:

1. Verify routing tables include routes to this peer
2. Check MTP3 point code routing is configured
3. Review DPC values in messages match expected routes
4. Check `/events` page for routing errors
5. Verify sequence numbers (BSN/FSN) are incrementing

Related Documentation

- ← Back to Main Documentation
- Common Features Guide - Web UI, API, Monitoring
- MAP Client Guide - Sending MAP requests
- SMS Center Guide - SMS delivery
- Technical Reference - Protocol specifications

OmniSS7 by Omnitouch Network Services

USSD Gateway Guide

← Back to Main Documentation

This guide covers the OmniSS7 **USSD Gateway**, which bridges SS7/MAP USSD dialogues to HTTP/JSON callbacks, enabling 3rd-party developers to build USSD applications with a simple HTTP endpoint.

Table of Contents

Overview

The USSD Gateway handles two directions of USSD traffic:

- **Mobile-Originated (Inbound)** — A subscriber dials a short code (e.g. `*100#`). The gateway receives the MAP `processUnstructuredSS-Request` (opcode 59), forwards it to your HTTP callback, and relays your response back over SS7.
- **Network-Originated (Outbound)** — Your application pushes a USSD message to a subscriber via the REST API. The gateway sends a MAP

`unstructuredSS-Request` (opcode 60) over SS7 and routes the subscriber's reply to your callback.

Both directions support **multi-turn dialogues** — interactive menus where the subscriber replies and receives follow-up prompts.

Key Characteristics

Property	Value
Transport	Synchronous HTTP POST per turn
Encoding	GSM 7-bit default alphabet (DCS 0x0F) per 3GPP TS 23.038
Max text length	182 characters (configurable)
Session tracking	Gateway-generated UUID per dialogue
Authentication	None (trusts the SS7 network)
Routing	Short code prefix matching to callback URLs

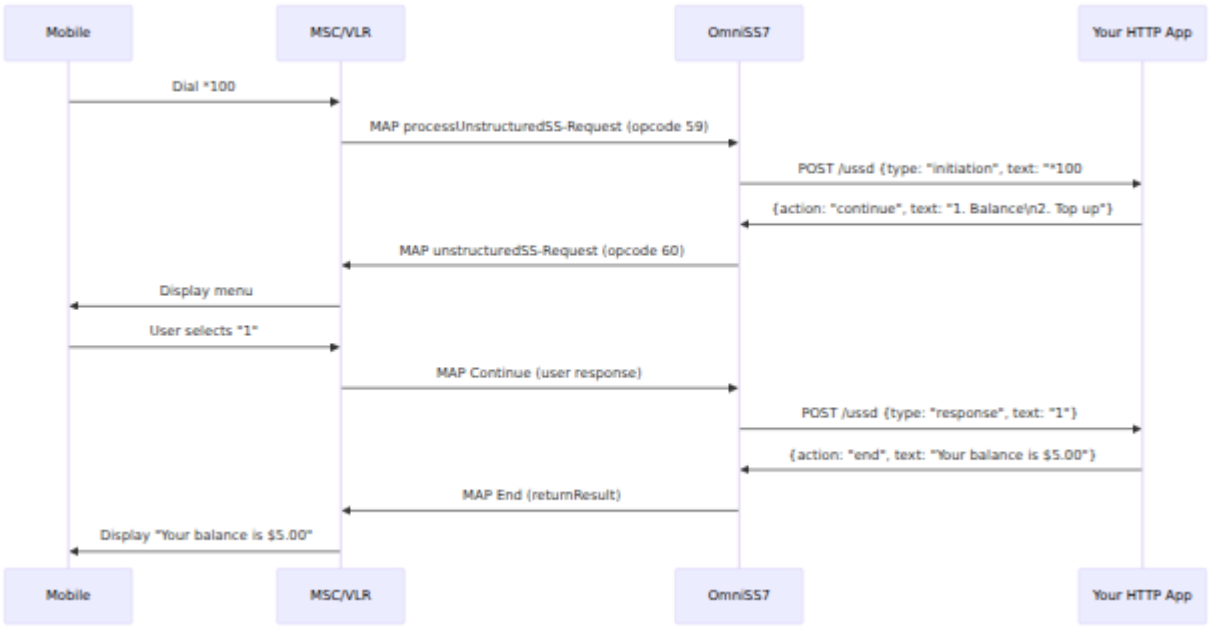
3GPP References

Specification	Relevance
3GPP TS 23.090	USSD Stage 2 — architecture and procedures
3GPP TS 24.090	USSD Stage 3 — protocol details
3GPP TS 29.002	MAP protocol — USSD-Arg, USSD-Res, opcodes 59/60/61
3GPP TS 23.038	GSM 7-bit default alphabet and data coding scheme

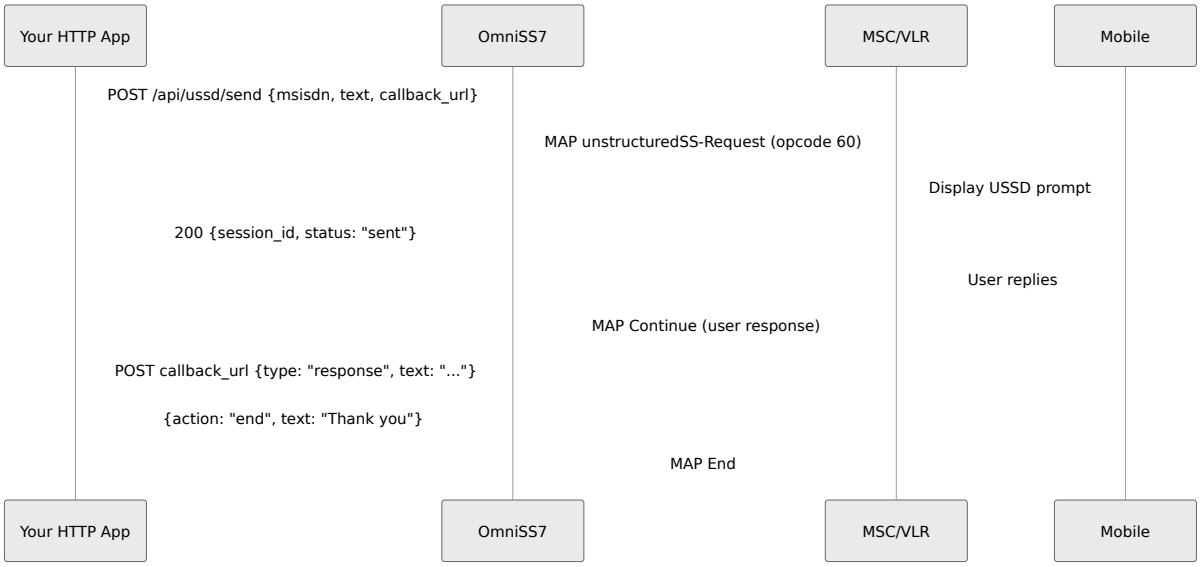
---

Architecture

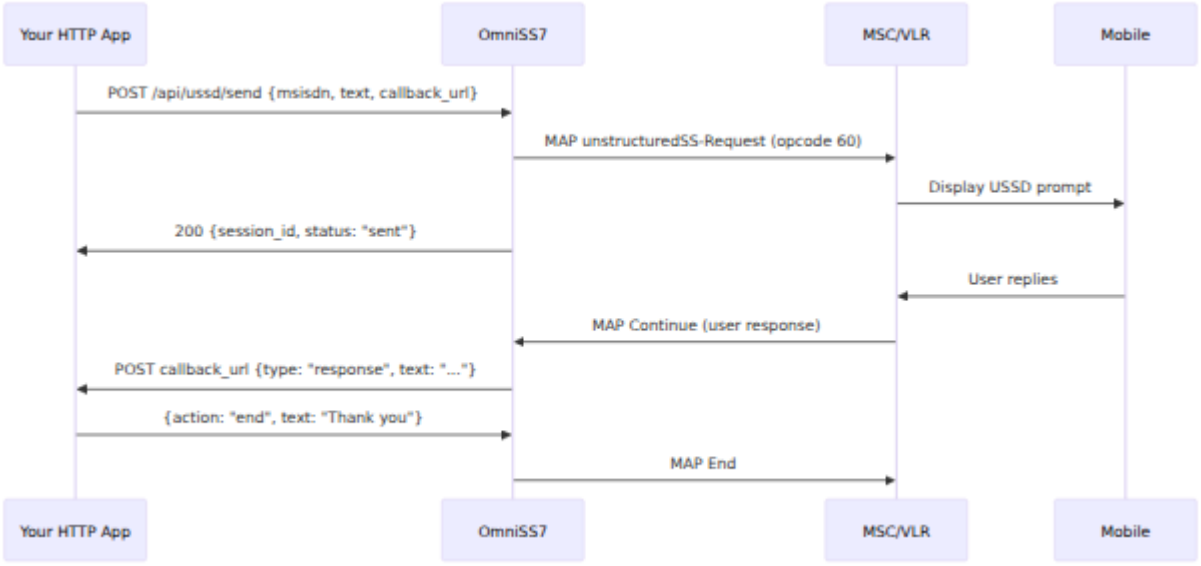
Mobile-Originated Flow (Inbound)



Network-Originated Flow (Outbound Push)



Component Overview



Enabling the USSD Gateway

The USSD Gateway requires **MAP Client mode** to be enabled, plus its own feature flag.

```
config :omniss7,
  map_client_enabled: true,
  ussd_gateway_enabled: true
```

The gateway also requires a working M3UA connection (see MAP Client Guide for M3UA setup).

Configuration

USSD Gateway Parameters

```
config :omniss7,
  ussd_gateway_enabled: true,
  ussd_gateway: %{
    # Short code routing — longest prefix match
    routes: [
      %{pattern: "*100", url: "http://balance-app:9000/ussd"},
      %{pattern: "*200", url: "http://topup-app:9000/ussd"},
      %{pattern: "*",    url: "http://default-app:9000/ussd"}
    ],

    # Session timeouts
    session_timeout_ms: 180_000,    # Total session lifetime (3
minutes)
    turn_timeout_ms: 30_000,        # Max wait for subscriber
reply per turn (30 seconds)

    # HTTP callback settings
    http_timeout_ms: 5_000,         # Timeout for HTTP POST to
your app (5 seconds)

    # Text limits
    max_text_length: 182            # GSM 7-bit max (truncates
with warning if exceeded)
  }
```

Parameter Reference

Parameter	Type	Required	Default	De
`ussd_gateway_enabled`	Boolean	Yes	`false`	Ma swi the Gat fea
`ussd_gateway.routes`	List of Maps	Yes	`[]`	Sho pre rou rule ent `pat` (pre stri `url` (ca URI Lon pre win
`ussd_gateway.session_timeout_ms`	Integer	No	`180_000`	Ma tot dur mil Ses ter wit if e
`ussd_gateway.turn_timeout_ms`	Integer	No	`30_000`	Ma tim for

Parameter	Type	Required	Default	Des
				sub rep mu dia mil
`ussd_gateway.http_timeout_ms`	Integer	No	`5_000`	HT req tim call you app in mil Cov con res tim
`ussd_gateway.max_text_length`	Integer	No	`182`	Ma cha in a tex Tex exc this tru and wa log

Route Parameters

Each entry in the `routes` list is a map:

Parameter	Type	Required	Description
`pattern`	String	Yes	Short code prefix to match. Use `"*"` as a catch-all fallback. Longer prefixes take priority.
`url`	String	Yes	HTTP endpoint URL to receive callback POSTs for matching short codes.

Route Matching

Routes are matched by **longest prefix first**. For the dial string `*100#`:

1. `"*100"` matches (length 4) — selected
2. `"*10"` matches (length 3) — skipped, shorter
3. `"*"` matches (length 1) — fallback

If no route matches, the gateway returns a MAP error to the mobile and logs a warning.

HTTP Callback Protocol

Your application receives HTTP POST requests from the gateway and responds with JSON instructions.

Request from Gateway to Your Application

Content-Type: `application/json`

First turn (session initiation):

```
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "msisdn": "+254712345678",
  "type": "initiation",
  "text": "*100#",
  "turn": 1
}
```

Subsequent turns (subscriber replied):

```
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "msisdn": "+254712345678",
  "type": "response",
  "text": "1",
  "turn": 2
}
```

Request Fields

Field	Type	Description
`session_id`	String	Gateway-generated UUID. Unique per USSD dialogue. Use this to correlate turns.
`msisdn`	String	Subscriber's MSISDN (if available from the MAP message). May be empty for some networks.
`type`	String	`"initiation"` for the first turn, `"response"` for subsequent subscriber replies.
`text`	String	The dial string (e.g. `*100#`) on initiation, or the subscriber's input (e.g. `1`) on response.
`turn`	Integer	Turn counter starting at 1. Increments with each subscriber interaction.

Response from Your Application

Your application must respond with JSON containing an `action` and `text`:

Continue (show menu, wait for subscriber input):

```
{
  "action": "continue",
  "text": "1. Balance\n2. Top up\n3. Transfer"
}
```

End (show final message, close session):

```
{
  "action": "end",
  "text": "Your balance is $5.00"
}
```

Response Fields

Field	Type	Required	Description
`action`	String	Yes	`"continue"` to keep the session open and wait for subscriber input, or `"end"` to display a final message and close the session.
`text`	String	Yes	Text to display on the subscriber's handset. Maximum length governed by `max_text_length` (default 182). Use `\n` for line breaks.

Network-Originated USSD (Push API)

Push a USSD message to a subscriber from your application.

Endpoint

`POST /api/ussd/send`

Request

```
{
  "msisdn": "+254712345678",
  "text": "You have a pending bill. Reply 1 to pay.",
  "callback_url": "http://billing-app:9000/ussd"
}
```

Request Fields

Field	Type	Required	Description
`msisdn`	String	Yes	Destination subscriber MSISDN in international format.
`text`	String	Yes	Initial USSD text to display. Encoded as GSM 7-bit.
`callback_url`	String	Yes	URL to receive the subscriber's reply via the standard callback protocol.

Response

Success (200 OK):

```
{
  "session_id": "xyz-789-abc-123",
  "status": "sent"
}
```

Error responses:

HTTP Status	Body	Cause
400	`{"error": "invalid request", "required": ["msisdn", "text", "callback_url"]}`	Missing one or more required fields, or the body is not valid JSON
500	`{"error": "{:gsm7_encode_failed, ...}"}`	Text contains characters not in the GSM 7-bit alphabet. The encode failure is surfaced through the generic error branch, so the status is **500**, not 400.
500	`{"error": "..."}`	Any other send failure (e.g. M3UA connectivity). The `error` field carries the inspected internal reason.
503	`{"error": "USSD gateway not enabled"}`	`ussd_gateway_enabled` is `false`

> **Note on validation vs. send errors:** Field-validation problems (missing keys, malformed JSON) return **400** **`invalid request`**. Once validation passes, any failure from the send path — including GSM 7-bit encoding failures — is reported as **500** with the inspected reason in the `error` field.

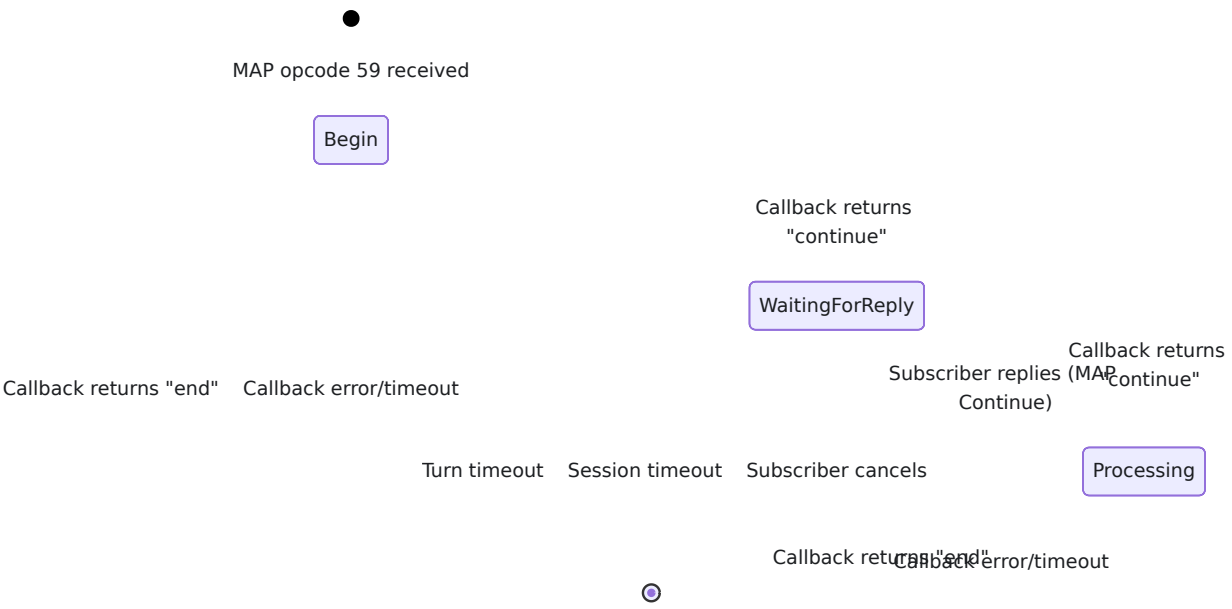
cURL Example

```
curl -X POST http://localhost:8080/api/ussd/send \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "+254712345678",
    "text": "You have a pending bill. Reply 1 to pay.",
    "callback_url": "http://billing-app:9000/ussd"
  }'
```

Session Lifecycle

Each USSD dialogue is tracked as a **session** with a unique `session_id`.

Session States



Single-Turn Session

If your callback returns `"end"` on the first turn, no persistent session is created. The gateway sends a MAP End with the result and returns immediately.

Multi-Turn Session

If your callback returns `"continue"`, the gateway:

1. Creates a **Session GenServer** registered in `UssdGateway.Registry`
2. Sends a MAP Continue with opcode 60 (unstructuredSS-Request) to the mobile
3. Waits for the subscriber's reply (up to `turn_timeout_ms`)
4. Forwards the reply to your callback
5. Repeats until your callback returns `"end"` or a timeout occurs

Timeout Behaviour

Timeout	Default	Effect
Turn timeout	30 seconds	If the subscriber doesn't reply within this window, the session is terminated with a MAP error.
Session timeout	3 minutes	Total session lifetime. Terminates the session regardless of activity.
HTTP callback timeout	5 seconds	If your application doesn't respond in time, the gateway sends a MAP error to the mobile and terminates the session.

Error Handling

The gateway handles failures gracefully and always attempts to send a MAP error response to the mobile so the subscriber sees a meaningful message rather than a network timeout.

Scenario	Gateway Action	MAP Error Code
HTTP callback timeout or 5xx	Terminate session, send MAP End with error	34 (systemFailure)
Invalid JSON from callback	Terminate session, send MAP End with error	34 (systemFailure)
USSD text exceeds `max_text_length`	Truncate text, log warning, continue normally	N/A (truncated, not an error)
Subscriber timeout (no reply)	Terminate session, send MAP End with error	34 (systemFailure)
No route matches short code	Send MAP End with error, log warning	34 (systemFailure)
Session GenServer crash	Session dies, subscriber sees network timeout	N/A (process exit)
USSD gateway not enabled	Return Facility Not Supported	21 (facilityNotSupported)

Metrics and Monitoring

The USSD Gateway exposes Prometheus metrics on the standard `/metrics` endpoint (port 8080).

USSD Metrics

Metric: `ussd_requests_total` **Type**: Counter **Description**: Total USSD requests processed **Labels**:

- `direction` — `"inbound"` (mobile-originated) or `"outbound"` (network-originated push)

Metric: `ussd_active_sessions` **Type**: Gauge **Description**: Declared as a gauge for the number of active USSD sessions.

> **Caveat:** This gauge is **declared but never updated** in the current build — it always reads `0`. Do not rely on it for live session counts. Track session activity via `ussd_requests_total` instead.

Metric: `map_request_duration_milliseconds` **Type**: Histogram **Description**: Duration of USSD send operations in milliseconds **Labels**:

- `operation` — `"ussd_send"` for outbound push requests

Example Prometheus Queries

```
# USSD request rate by direction (inbound vs outbound)
rate(ussd_requests_total[5m])

# Outbound USSD latency (p95)
histogram_quantile(0.95,
rate(map_request_duration_milliseconds_bucket{operation="ussd_send"}
[5m]))
```

Example Callback Server

Python (Flask)

```
from flask import Flask, request, jsonify

app = Flask(__name__)
sessions = {}

@app.route('/ussd', methods=['POST'])
def ussd():
    data = request.json
    session_id = data['session_id']
    text = data['text']
    turn = data['turn']

    if data['type'] == 'initiation':
        sessions[session_id] = {'state': 'main_menu'}
        return jsonify({
            'action': 'continue',
            'text': 'Welcome!\n1. Check balance\n2. Buy
airtime\n3. Transfer'
        })

    state = sessions.get(session_id, {}).get('state')

    if state == 'main_menu':
        if text == '1':
            del sessions[session_id]
            return jsonify({
                'action': 'end',
                'text': 'Your balance is $5.00'
            })
        elif text == '2':
            sessions[session_id]['state'] = 'buy_airtime'
            return jsonify({
                'action': 'continue',
                'text': 'Enter amount:'
            })
        else:
            del sessions[session_id]
            return jsonify({
```

```
                'action': 'end',
                'text': 'Invalid option. Goodbye.'
            })

    elif state == 'buy_airtime':
        del sessions[session_id]
        return jsonify({
            'action': 'end',
            'text': f'You purchased ${text} airtime. Thank you!'
        })

    return jsonify({'action': 'end', 'text': 'Session expired.'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9000)
```

Node.js (Express)

```
const express = require('express');
const app = express();
app.use(express.json());

const sessions = new Map();

app.post('/ussd', (req, res) => {
  const { session_id, text, type } = req.body;

  if (type === 'initiation') {
    sessions.set(session_id, { state: 'main_menu' });
    return res.json({
      action: 'continue',
      text: 'Welcome!\n1. Check balance\n2. Buy airtime'
    });
  }

  const session = sessions.get(session_id);
  if (!session) {
    return res.json({ action: 'end', text: 'Session expired.' });
  }

  if (session.state === 'main_menu' && text === '1') {
    sessions.delete(session_id);
    return res.json({ action: 'end', text: 'Your balance is $5.00'
});
  }

  sessions.delete(session_id);
  return res.json({ action: 'end', text: 'Goodbye.' });
});

app.listen(9000, () => console.log('USSD callback on port 9000'));
```

Troubleshooting

USSD Dial Returns "Service Not Available"

Symptoms: Subscriber dials a short code and immediately gets a network error.

Possible causes:

- `ussd_gateway_enabled` is `false`
- No route matches the dialled short code
- M3UA connection is down

Resolution:

1. Verify `ussd_gateway_enabled: true` in configuration
2. Check that a route pattern matches the short code (remember to include `*` catch-all)
3. Check M3UA peer status in the Web UI (Peers page)

Callback Not Receiving Requests

Symptoms: Gateway logs show USSD begin but your application never receives the HTTP POST.

Possible causes:

- Callback URL is unreachable from the OmniSS7 host
- Firewall blocking outbound HTTP from OmniSS7
- Callback application not running

Resolution:

1. Test connectivity: `curl -v http://your-app:9000/ussd` from the OmniSS7 host
2. Check firewall rules for outbound HTTP
3. Verify your callback application is listening on the configured port

Sessions Timing Out Prematurely

Symptoms: Multi-turn sessions end with "systemFailure" before the subscriber can reply.

Possible causes:

- `turn_timeout_ms` is too short for your subscriber base
- `http_timeout_ms` is too short for your application's processing time
- Network latency between OmniSS7 and your callback server

Resolution:

1. Increase `turn_timeout_ms` (default 30 seconds should suffice for most cases)
2. Increase `http_timeout_ms` if your application needs more processing time
3. Deploy callback server close to OmniSS7 to reduce latency

GSM 7-bit Encoding Errors

Symptoms: `gsm7_encode_failed` errors in logs, or a **500** response from `/api/ussd/send` whose `error` field contains `{:gsm7_encode_failed, ...}`.

Possible causes:

- Text contains characters outside the GSM 7-bit default alphabet (e.g. emoji, CJK characters)

Resolution:

- Restrict USSD text to the GSM basic character set: ASCII letters, digits, common punctuation, and a few Greek/Nordic characters
- See 3GPP TS 23.038 Section 6.2.1 for the complete character table

---

Related Documentation

- **API Guide** — Full REST API reference (all endpoints including `/api/ussd/send`)
- **MAP Client Guide** — M3UA connection setup required for USSD
- **Configuration Reference** — All configuration parameters
- **Common Features Guide** — Web UI, monitoring, and Prometheus setup

Web UI Guide

← Back to Main Documentation

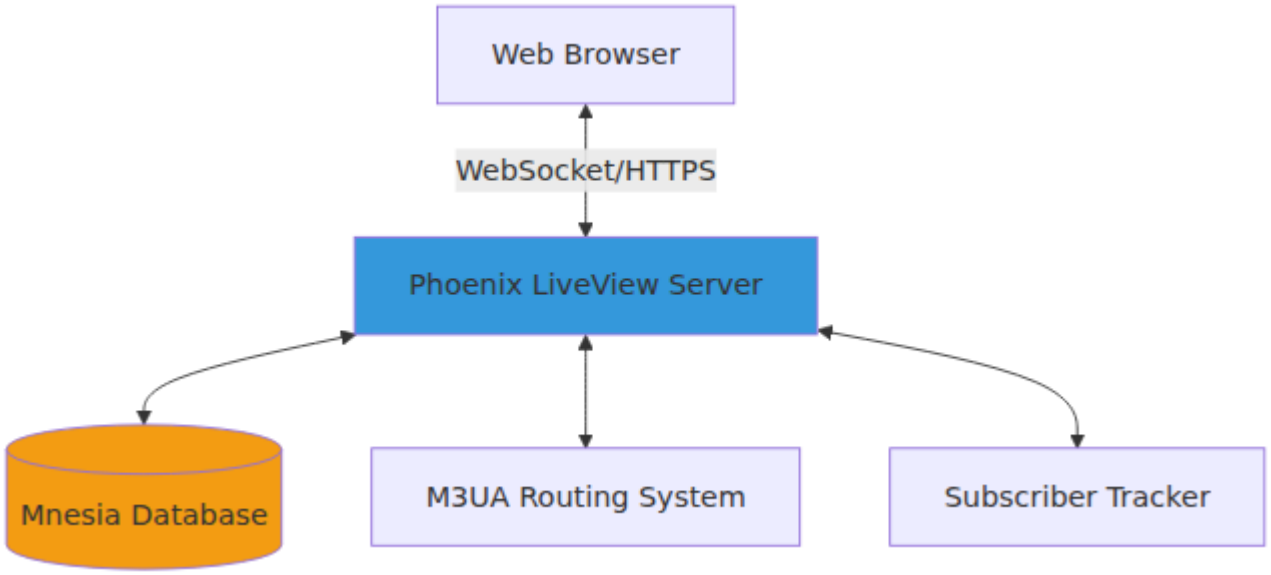
This guide provides comprehensive documentation for using the OmniSS7 **Web UI** (Phoenix LiveView interface).

Table of Contents

Overview

The OmniSS7 Web UI is a **Phoenix LiveView** application that provides real-time monitoring and management capabilities. The available pages depend on which operational mode is active (STP, HLR, or SMSc).

Web UI Architecture



Server Configuration

Parameter	Value
Protocol	HTTPS (`force_ssl` enabled)
Port	`8087` (configured under `config :control_panel, ControlPanelWeb.Endpoint` → `https: [port: 8087]`)
Default IP	`0.0.0.0` (listens on all interfaces)
Certificate	`priv/cert/omnitouch.crt`
Private key	`priv/cert/omnitouch.pem`
Adapter	Bandit (Phoenix)

Access URL: `https://[server-ip]:8087`

Accessing the Web UI

Prerequisites

1. **SSL Certificates**: Ensure valid SSL certificates are present in `priv/cert/`:
   - `omnitouch.crt` - Certificate file
   - `omnitouch.pem` - Private key file
2. **Application Running**: Start the application with `iex -S mix`
3. **Firewall**: Ensure port `8087` is open for HTTPS traffic

Available Pages by Mode

The page label shown here is exactly the label registered in `config/config.exs` (`use_builtin_pages` / `use_additional_pages`). Mode columns indicate the operational mode (STP/HLR/SMSc) in which a page is typically relevant; runtime configuration filters the page list per deployment.

Page (route)	Label	STP Mode	HLR Mode	SMSc Mode	Descript
`/application`	**Resources**	✅	✅	✅	System resources and monitorir
`/configuration`	**Configuration**	✅	✅	✅	Configura viewer
`/license`	**License**	✅	✅	✅	License status
`/events`	**SS7 Events**	✅	✅	✅	Event logging a SCCP message capture
`/logs`	**System Logs**	✅	✅	✅	Live applicatic log viewe
`/client`	**SS7 Client**	✅	✅	✅	Manual M operation testing
`/m3ua`	**Peers**	✅	✅	✅	M3UA pe ASP connectic status
`/sctp`	**SCTP Connections**	✅	✅	✅	Live SCTF associatic status

Page (route)	Label	STP Mode	HLR Mode	SMSc Mode	Descript
`/routing`	**Routing**	□	□	□	M3UA routing ta managen
`/routing_test`	**Routing Test**	□	□	□	Route tes and validatior
`/hlr_links`	**HLR Links**	□	□	□	HLR API status an subscribe managen
`/subscribers`	**Active Subscribers**	□	□	□	Real-time subscribe location tracking (HLR)
`/smsc_links`	**SMSc Links**	□	□	□	SMSc API status an queue managen
`/smsc_subscribers`	**SMSc Subscribers**	□	□	□	Real-time subscribe tracking (SMSc)
`/camel_sessions`	**CAMEL Sessions**	□	□	□	Live CAMEL/C dialogue monitorir

Page (route)	Label	STP Mode	HLR Mode	SMSc Mode	Descript
`/camel_request`	**CAMEL Request Builder**	□	□	□	Interactiv CAP requ builder
`/msisdn_imsi_test`	**MSISDN/IMSI Test**	□	□	□	MSISDN ↔ IMSI look conversio testing

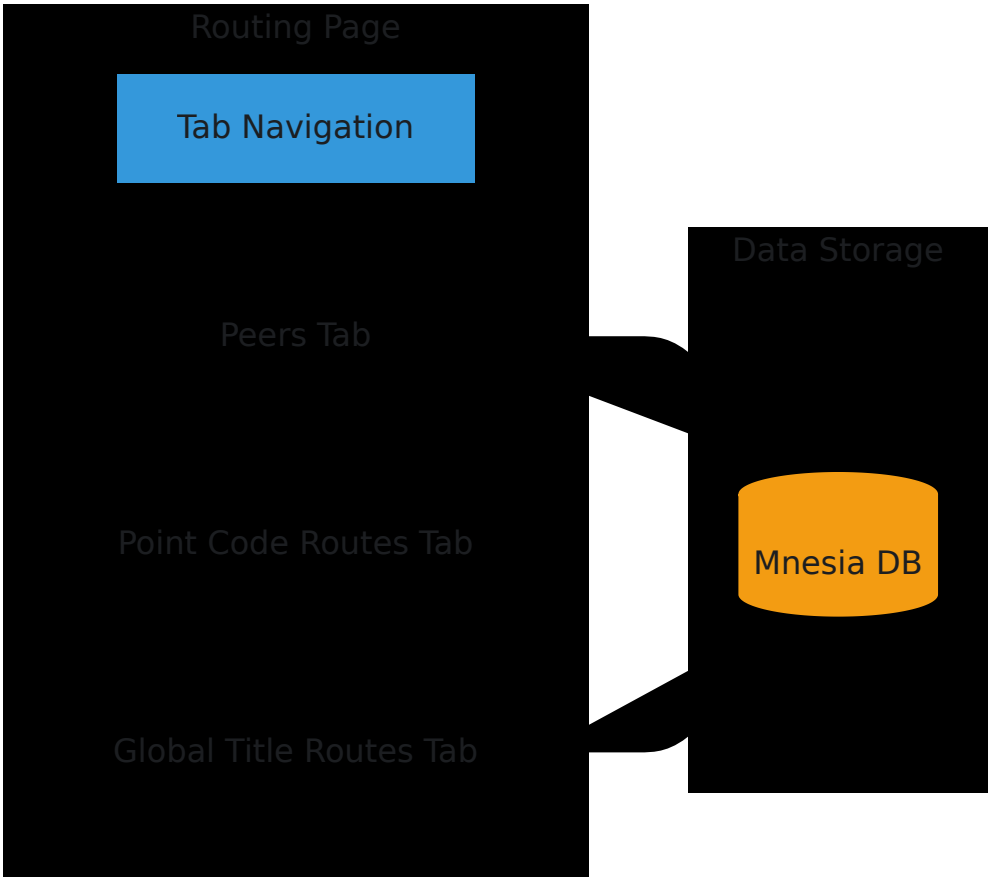
> The M3UA page is labelled **"Peers"** in the navigation (route `/m3ua`).

Routing Management

Page: `/routing` **Modes**: STP, SMSc **Auto-Refresh**: Every 5 seconds

The Routing Management page provides a tabbed interface for managing M3UA routing tables.

Page Layout



Peers Tab

Manage M3UA peer connections (other STPs, HLRs, MSCs, SMSCs).

Peer Table Columns

Column	Description	Example
ID	Unique peer identifier	`1`
Name	Human-readable peer name	`"STP_West"`
Role	Connection role	`client`, `server`, `stp`
Point Code	Peer's SS7 point code	`100`
Remote	Remote IP:Port	`10.0.0.10:2905`
Status	Connection status	`active`, `aspup`, `down`
Actions	Edit/Delete buttons	-

Adding a Peer

1. **Click** the Peers tab
2. **Fill in** the form fields:
   - **Peer ID**: Auto-generated if left empty
   - **Peer Name**: Descriptive name (required)
   - **Role**: Select `client`, `server`, or `stp`
   - **Point Code**: SS7 point code (required)
   - **Local IP**: Your system's IP address
   - **Local Port**: 0 for dynamic port assignment
   - **Remote IP**: Peer's IP address
   - **Remote Port**: Peer's port (typically 2905)
   - **Routing Context**: M3UA routing context ID
   - **Network Indicator**: `international` or `national`
3. **Click** "Add Peer"

Persistence: Peer is immediately saved to Mnesia and survives restart.

Editing a Peer

1. **Click** the "Edit" button on the peer row
2. **Modify** the form fields as needed
3. **Click** "Update Peer"

Note: If you change the Peer ID, the old peer is deleted and a new one is created.

Deleting a Peer

1. **Click** the "Delete" button on the peer row
2. **Confirm** the deletion (all routes using this peer will also be removed)

Peer Status Indicators

Status	Color	Description
`active`	Green	Peer is connected and routing messages
`aspup`	Yellow	ASP is up but not yet active
`down`	Red	Peer is disconnected

Point Code Routes Tab

Configure routing rules based on destination Point Codes.

Route Table Columns

Column	Description	Example
Destination PC	Target point code (zone.area.id format)	`1.2.3` (100)
Mask	Subnet mask for PC matching	`/14` (exact), `/8` (range)
Peer ID	Target peer for this route	`1`
Peer Name	Name of target peer	`"STP_West"`
Priority	Route priority (1 = highest)	`1`
Network	Network indicator	`international`
Actions	Edit/Delete buttons	-

Adding a Point Code Route

1. **Click** the "Point Code Routes" tab
2. **Fill in** the form fields:
   - **Destination Point Code**: Enter as `zone.area.id` (e.g., `1.2.3`) or integer (0-16383)
   - **Mask**: Select mask `/14` for exact match, lower values for ranges
   - **Peer ID**: Select target peer from dropdown
   - **Priority**: Enter priority (1 = highest, default)
   - **Network Indicator**: Select `international` or `national`
3. **Click** "Add Route"

Point Code Format: You can enter point codes in two formats:

- **3-8-3 Format**: `zone.area.id` (e.g., `1.2.3`)
- **Integer Format**: `0-16383` (e.g., `1100`)

The system automatically converts between formats.

Understanding Masks

Point codes are 14-bit values (0-16383). The mask specifies how many most significant bits must match:

Mask	PCs Matched	Use Case
`/14`	1 (exact match)	Route to specific destination
`/13`	2 PCs	Small range
`/8`	64 PCs	Medium range
`/0`	All 16,384 PCs	**Default/fallback route**

Examples:

- `PC 1000 /14` → Matches only PC 1000
- `PC 1000 /8` → Matches PC 1000-1063 (64 consecutive PCs)
- `PC 0 /0` → Matches all point codes (default route)

Point Code Mask Reference Card

The web page includes an interactive reference showing all mask values and their ranges.

Global Title Routes Tab

Configure routing rules based on SCCP Global Title addresses.

Requirement: Global Title routing must be enabled in configuration:

```
config :omniss7,
  enable_gt_routing: true
```

Route Table Columns

Column	Description	Example
GT Prefix	Called party GT prefix (empty = fallback)	`"1234"`, `""`
Source SSN	Match on called party SSN (optional)	`6` (HLR), `any`
Peer ID	Target peer	`1`
Peer	Peer name	`"HLR_West (1)"`
Dest SSN	Rewrite SSN when forwarding (optional)	`6`, `preserve`
Priority	Route priority	`1`
Description	Route description	`"US numbers"`
Actions	Edit/Delete buttons	-

Adding a Global Title Route

1. **Click** the "Global Title Routes" tab
2. **Fill in** the form fields:
   - **GT Prefix**: Leave empty for fallback route, or enter digits (e.g., `"1234"`)
   - **Source SSN**: Optional - filter by called party SSN
   - **Peer ID**: Select target peer
   - **Dest SSN**: Optional - rewrite SSN when forwarding
   - **Priority**: Route priority (1 = highest)
   - **Description**: Human-readable description
3. **Click** "Add Route"

Fallback Routes: If GT Prefix is empty, the route acts as a catch-all for GTs that don't match any other route.

Common SSN Values

The page includes a reference card with common SSN values:

SSN	Network Element
6	HLR (Home Location Register)
7	VLR (Visitor Location Register)
8	MSC (Mobile Switching Center)
9	EIR (Equipment Identity Register)
10	AUC (Authentication Center)
142	RANAP
145	gsmSCF (Service Control Function)
146	SGSN

SSN Rewriting

- **Source SSN**: Match on the Called Party SSN in incoming messages
- **Dest SSN**: If set, rewrites the Called Party SSN when forwarding
  - Empty = preserve original SSN
  - Value = replace with this SSN

Use Case: Route messages with SSN=6 (HLR) to a peer, and rewrite to SSN=7 (VLR) on the outgoing side.

Routing Table Persistence

All routes are stored in Mnesia and survive application restarts.

How Routes Persist

1. **Web UI Changes**: All add/edit/delete operations are immediately saved to Mnesia

2. **Application Restart**: Routes are loaded from Mnesia on startup
3. **Runtime.exs Merge**: Static routes from `config/runtime.exs` are merged with Mnesia routes (no duplicates)

Route Priority

When multiple routes match a destination:

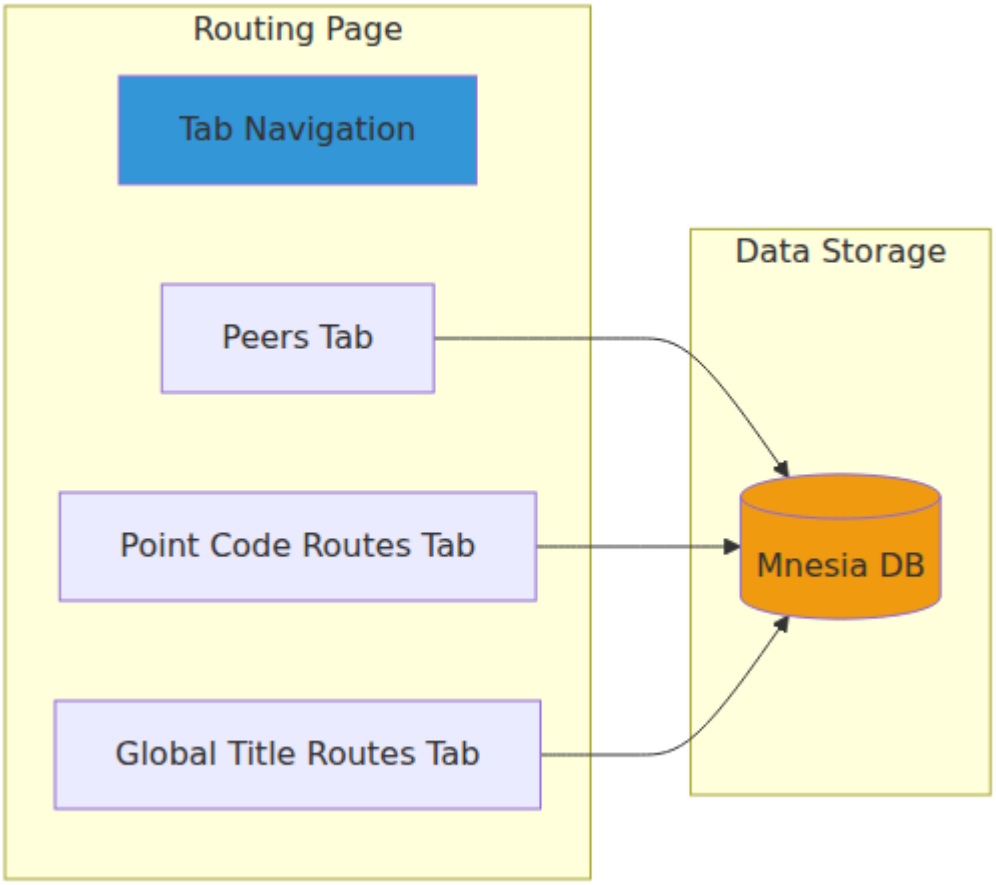
1. **More Specific First**: Higher mask values (more specific) take precedence
2. **Priority Field**: Lower priority numbers route first (1 = highest priority)
3. **Peer Status**: Only routes to `active` peers are used

Active Subscribers

Page: `/subscribers` **Mode**: HLR only **Auto-Refresh**: Every 2 seconds

Displays real-time tracking of subscribers who have sent UpdateLocation requests.

Page Features



Subscriber Table Columns

Column	Description	Example
IMSI	Subscriber IMSI	`"50557123456789"`
VLR Number	Current VLR GT address	`"555123155"`
MSC Number	Current MSC GT address	`"555123155"`
Updated At	Last UpdateLocation timestamp	`"2025-10-25 14:23:45 UTC"`
Duration	Time since registration	`"2h 15m 34s"`

Statistics Summary

When subscribers are present, a summary card displays:

- **Total Active**: Total number of registered subscribers
- **Unique VLRs**: Number of distinct VLR addresses
- **Unique MSCs**: Number of distinct MSC addresses

Clearing Subscribers

Clear All Button: Removes all active subscribers from the tracker.

Confirmation: Requires confirmation before clearing (cannot be undone).

Use Case: Clear stale subscriber records after network maintenance or testing.

Auto-Refresh

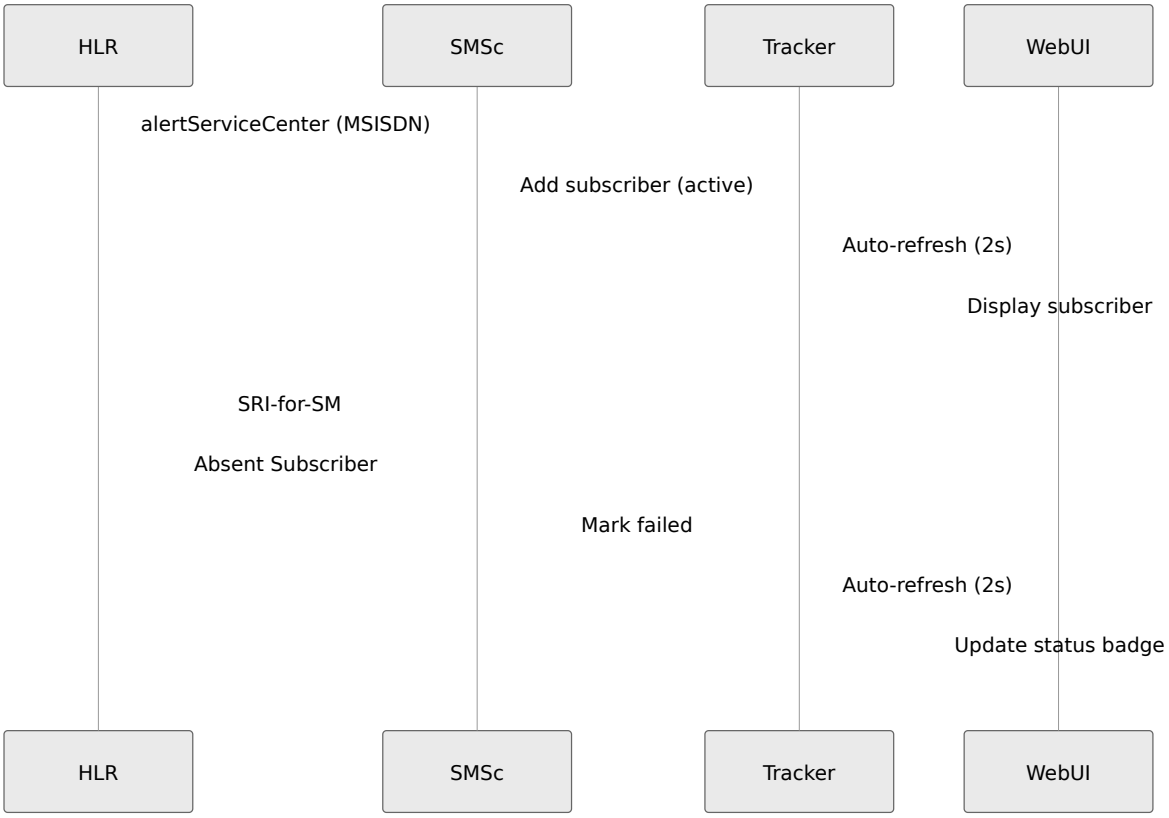
The page automatically refreshes every **2 seconds** to show real-time subscriber updates.

SMSc Subscribers

Page: `/smsc_subscribers` **Mode**: SMSc only **Auto-Refresh**: Every 2 seconds

Displays real-time tracking of subscribers based on alertServiceCenter messages received from HLRs, message delivery status, and failure tracking.

Page Features



Subscriber Table Columns

Column	Description	Example
MSISDN	Subscriber's phone number	`"15551234567"`
IMSI	Subscriber IMSI	`"001010123456789"`
HLR GT	HLR GT that sent alertServiceCenter	`"15551111111"`
Msgs Sent	Count of MT-FSM messages sent	`5`
Msgs Rcvd	Count of MO-FSM messages received	`2`
Status	`Active` or `Failed` (color-coded)	`● Active`
Last Updated	Last update timestamp	`"2025-10-30 14:23:45 UTC"`
Duration	Time since last update	`"15m 34s"`

Status Indicators

- ● **Active** (Green): Subscriber is reachable, last alertServiceCenter received successfully
- ○ **Failed** (Red): Last delivery attempt failed (SRI-for-SM or absent subscriber error)

Statistics Summary

When subscribers are present, a summary card displays:

- **Total Tracked**: Total number of tracked subscribers

- **Active**: Number of subscribers with active status
- **Failed**: Number of subscribers with failed status
- **Unique HLRs**: Number of distinct HLRs sending alerts

Managing Subscribers

Remove Button: Removes individual subscriber from tracking.

Clear All Button: Removes all tracked subscribers.

Confirmation: Clear All requires confirmation before clearing (cannot be undone).

Use Case:

- Remove stale entries after network issues
- Clear test data after development
- Monitor which HLRs are sending alerts

Message Counters

The tracker automatically increments counters:

- **Messages Sent**: Incremented when SRI-for-SM succeeds and MT-FSM is sent
- **Messages Received**: Incremented when MO-FSM is received from subscriber

Auto-Refresh

The page automatically refreshes every **2 seconds** to show real-time subscriber and status updates.

Common Operations

Searching and Filtering

Currently, the Web UI does not include built-in search/filter functionality. To find specific routes:

1. Use your browser's find function (Ctrl+F / Cmd+F)
2. Search for peer names, point codes, or GT prefixes

Bulk Operations

To perform bulk route changes:

1. **Option 1**: Use the REST API for programmatic access
2. **Option 2**: Edit `config/runtime.exs` and restart the application
3. **Option 3**: Use the Web UI for individual route changes

Export/Import

Note: The Web UI does not currently support exporting or importing routing tables. Routes are:

- Stored in Mnesia database files
- Configured in `config/runtime.exs`

To backup routes:

1. **Mnesia**: Backup the `Mnesia.{node_name}/` directory
2. **Config**: Version control `config/runtime.exs`

Auto-Refresh Behavior

Different pages have different refresh intervals:

Page	Refresh Interval	Reason
Routing Management	5 seconds	Route changes are infrequent
Active Subscribers	2 seconds	Subscriber state changes frequently
M3UA Status	Varies by page	Connection state monitoring

WebSocket Connection: All pages use Phoenix LiveView WebSocket connections for real-time updates.

Network Interruption: If the WebSocket connection is lost, the page will attempt to reconnect automatically.

Troubleshooting

Page Not Loading

1. **Check HTTPS Certificate**: Ensure `priv/cert/omnitouch.crt` and `priv/cert/omnitouch.pem` are present
2. **Verify Port 8087**: Check firewall rules allow HTTPS traffic on port `8087`
3. **Application Running**: Confirm application is running with `iex -S mix`
4. **Browser Console**: Check for SSL certificate errors (self-signed cert warnings)

Routes Not Persisting

1. **Check Mnesia Storage**: Verify `mnesia_storage_type: :disc_copies` in config
2. **Mnesia Directory**: Ensure Mnesia directory is writable
3. **Check Logs**: Look for Mnesia errors in application logs

Auto-Refresh Not Working

1. **WebSocket Connection**: Check browser console for WebSocket errors
2. **Network**: Verify stable network connection
3. **Page Reload**: Try refreshing the page (F5)

Related Documentation

- **STP Guide** - Detailed routing configuration
- **HLR Guide** - Subscriber management
- **API Guide** - REST API for programmatic access
- **Configuration Reference** - All configuration parameters

Summary

The OmniSS7 Web UI provides intuitive, real-time management of routing tables and subscriber tracking:

 Real-time Updates - Auto-refresh keeps data current **Persistent Storage** - Mnesia ensures routes survive restarts **Role-Based UI** - Pages adapt to operational mode (STP/HLR/SMSc) **Interactive Management** - Add, edit, delete routes without restart **Status Monitoring** - Live connection and peer status

For advanced operations or automation, see the API Guide.