← Overview

OmniCHF Configuration Reference

Operator configuration is read from the application environment key `:omnichf`, populated at startup by `config/runtime.exs`. Every key is overridable by an OS environment variable. The example below shows the complete runtime configuration with the values applied when the corresponding environment variable is unset.

```
config :omnichf,
  # SBI (service-based interface) listener
  sbi_scheme: "http",
  sbi_addr:   "127.0.0.14",
  sbi_port:   7777,

  # NRF registration / heartbeat
  nrf_uri:    "http://127.0.0.1:7777",

  # Serving PLMN identity
  mcc: "999",
  mnc: "70",

  # Observability
  prometheus_metrics_port: 9568,

  # NRF heartbeat cadence
  heartbeat_interval: 10_000,

  # CGRateS rating / balance engine integration
  cgrates_enabled: true,
  cgrates_url:     "http://localhost:2080/jsonrpc",
  cgrates_tenant:  "cgrates.org",
  cgrates_timeout: 5000

# Structured JSON logging
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all
```

Core Parameters

Parameter	Type	Required	Default
`sbi_scheme`	String	No	`"http"`
`sbi_addr`	String	No	`"127.0.0.14"`
`sbi_port`	Integer	No	`7777`

Parameter	Type	Required	Default
`nrf_uri`	String	No	`"http://127.0.0.1:7777`
`mcc`	String	No	`"999"`
`mnc`	String	No	`"70"`
`prometheus_metrics_port`	Integer	No	`9568`

Parameter	Type	Required	Default
`heartbeat_interval`	Integer (ms)	No	`10000`

CGRateS Parameters

These keys control the integration with the external CGRateS rating and balance engine. See CGRateS Integration for behavioral detail.

Parameter	Type	Required	Default
`cgrates_enabled`	Boolean	No	`true` (`"true"`)
`cgrates_url`	String	No	`"http://localhost:2080/jsonrp`
`cgrates_tenant`	String	No	`"cgrates.org"`

Parameter	Type	Required	Default
`cgrates_timeout`	Integer (ms)	No	`5000`

> **Default when disabled at build time:** the compiled default (used if the application environment is never populated by `runtime.exs`) is `cgrates_enabled: false`. In a normal deployment `runtime.exs` sets it, and there the default is `true`. The test profile forces `cgrates_enabled: false`.

Advanced / Optional Keys

These keys are not populated by the default `runtime.exs` but are read from the `:omnichf` environment if present. They tune rating-group resolution, offline charging, and charging-ID generation. They have no dedicated environment variable and are set directly in the `:omnichf` config block.

Parameter	Type	Required	Default
`rating_groups`	Map	No	`%{}`
`default_rating_group`	Integer	No	`1`
`offline_charging_enabled`	Boolean	No	`false`
`cdr_output_dir`	String	No	`"/var/log/omnichf/cd`
`node_id`	Integer	No	`1`

Rating Groups

`rating_groups` maps a **DNN** (the map key, a string) to the **rating group** (the value, an integer) applied to that DNN's sessions. The resolved rating group appears in the `ratingGroup` field of `multipleUnitInformation` returned to the consumer and in the CDR `rating_group` field. Resolution is:

1. If the session's DNN is a key in `rating_groups`, use that value.
2. Otherwise use `default_rating_group`.

```
config :omnichf,
  rating_groups: %{
    "internet" => 10,
    "ims"      => 20,
    "iot"      => 30
  },
  default_rating_group: 1
```

Matching is on the exact DNN string. A DNN with no entry (and no `default_rating_group`) falls back to rating group `1`. A malformed `rating_groups` value (not a map) is ignored and the default is used, so a bad config never blocks charging.

Logging

Structured JSON logging is configured via the `:logger` `:default_formatter`.

Parameter	Type	Default	Description
`format`	Tuple	`{OmniLogger.JsonFormatter, :format}`	Log line formatter. Emits structured JSON, suitable for ingestion by a log shipper.
`metadata`	Atom / List	`:all`	Log metadata included in each line. `:all` emits all attached metadata (SUPI, DNN, PDU session ID, charging ID, procedure, etc.).

Management Interfaces

OmniCHF exposes operator-facing management surfaces alongside the SBI. These ports are relevant for monitoring and OAM but are not part of the Nchf service:

Interface	Port	Scheme	Purpose
SBI (Nchf)	`sbi_port` (default `7777`)	HTTP	5G inter-NF charging traffic.
Prometheus metrics	`prometheus_metrics_port` (default `9568`)	HTTP	Metrics scrape endp
Management API	`8443`	HTTPS	Status reads, session/statistics/CG health reads, CDR se and OAM actions (co CDR flush, log level, session abort, NRF r register).
Control Panel	`7443`	HTTPS	Web UI for resources configuration, licens logs.

Management / OAM API Endpoints

The Management API on port `8443` (HTTPS) exposes the following operator surfaces:

Method	Path	Purpose
`GET`	`/api/status/api`	Read API/service status.
`GET`	`/api/status/license`	Read license status.
`GET`	`/api/status/nrf`	Read NRF registration status.
`GET`	`/api/status/nf`	Read NF (node) status.
`GET`	`/api/sessions`	List active charging sessions.
`GET`	`/api/sessions/{id}`	Inspect a single charging session by `charging_data_ref`.
`GET`	`/api/statistics`	Read charging statistics (active session count, total charged volume, CGRateS reachability).
`GET`	`/api/health/cgrates`	Read CGRateS connectivity health.
`GET`	`/api/oam/config`	Read runtime configuration.
`PATCH`	`/api/oam/config/{id}`	Update a runtime configuration key.
`POST`	`/api/oam/charging_session`	Abort a charging session by `charging_data_ref`.
`GET`	`/api/oam/cdr`	Search completed CDRs. Query params: `supi`, `dnn`, `date_from`, `date_to` (all optional).
`POST`	`/api/oam/cdr/flush`	Force-flush accumulated usage as a CDR for one session (`{"charging_data_ref": "`

Method	Path	Purpose
		`<ref>"}`) or all tracked sessions (`{}` / `"all"`).
`POST`	`/api/oam/log_level`	Change the runtime log level.
`POST`	`/api/oam/nrf/reregister`	Trigger NRF re-registration.

> **Session-store visibility:** the session-facing management surfaces (`/api/sessions` , `/api/statistics` , `/api/oam/charging_session` , `/api/oam/cdr/flush`) present every live session. The live SBI charging path holds each session in a per-session **worker** process, and every worker session is mirrored into the in-memory **Context store** on create and update and removed on stop. The management surfaces query the worker registry first and fall back to Context, so a live session is always visible and actionable. See Session Store and Management Visibility for what this means operationally. CDR search (`/api/oam/cdr`) reads the offline CDR files and is independent of both stores.

← Overview

OmniCHF Metrics Reference

Metrics are exposed on `prometheus_metrics_port` (default `9568`) at `/metrics`.

Charging Metrics

Metric: `omni_chf_charging_initial_count` **Type**: Counter **Description**: Charging session INITIAL (Create) requests received. **Labels**: `supi`

Metric: `omni_chf_charging_update_count` **Type**: Counter **Description**: Charging session UPDATE requests received. **Labels**: `ref`

Metric: `omni_chf_charging_release_count` **Type**: Counter **Description**: Charging session RELEASE requests received. **Labels**: `ref`

Metric: `omni_chf_charging_creates_total` **Type**: Counter **Description**: Create operation outcomes. **Labels**: `result` (`success` / `failure`)

Metric: `omni_chf_charging_updates_total` **Type**: Counter **Description**: Update operation outcomes. **Labels**: `result` (`success` / `failure`)

Metric: `omni_chf_charging_releases_total` **Type**: Counter **Description**: Release operation outcomes. Released via the SBI with a successful CGRateS terminate (`success`), released via the SBI where the CGRateS `TerminateSession` failed but the CDR was still finalised (`failure`, surfaced to the consumer as a `500`), or force-finalised through the OAM CDR-flush endpoint (`oam_flush`). **Labels**: `result` (`success` / `failure` / `oam_flush`)

Metric: `omni_chf_sessions_active_count` **Type**: Gauge **Description**: Active charging sessions currently held in memory. Emitted on every Context store create and delete, which now includes every mirrored live worker session, so the gauge tracks the true active session count. **Labels**: none

```
# Charging session create rate
rate(omni_chf_charging_creates_total{result="success"}[5m])

# Create failure ratio
sum(rate(omni_chf_charging_creates_total{result="failure"}[5m]))
  / sum(rate(omni_chf_charging_creates_total[5m]))

# Currently active sessions
omni_chf_sessions_active_count
```

CGRateS Metrics

Metric: `omni_chf_cgrates_requests_total` **Type**: Counter **Description**: CGRateS JSON-RPC requests. **Labels**: `operation` (SessionS method), `result` (`success` / `failure`)

Metric: `omni_chf_cgrates_request_duration_ms` **Type**: Histogram **Description**: CGRateS JSON-RPC request duration in milliseconds. **Labels**: `operation` **Buckets**: 5, 10, 25, 50, 100, 250, 500, 1000, 2500

Metric: `omni_chf_cgrates_health` **Type**: Gauge **Description**: CGRateS connectivity health (`1` = reachable, `0` = down/disabled). **Labels**: none

```
# CGRateS p99 latency by operation
histogram_quantile(0.99,
  sum by (le, operation)
(rate(omni_chf_cgrates_request_duration_ms_bucket[5m])))

# CGRateS error rate
sum(rate(omni_chf_cgrates_requests_total{result="failure"}[5m]))

# CGRateS reachability (alert when 0)
omni_chf_cgrates_health
```

NRF Metrics

Metric: `omni_chf_nrf_registration_status` **Type**: Gauge **Description**: NRF registration status (`1` = registered, `0` = not registered). **Labels**: `nf_type`

```
omni_chf_nrf_registration_status
```

BEAM VM Metrics

Standard runtime health gauges are also exported.

Metric	Type	Description
`beam_memory_total`	Gauge	Total BEAM memory (bytes)
`beam_memory_processes` / `beam_memory_processes_used`	Gauge	Process memory allocated / used
`beam_memory_system`	Gauge	System (non-process) memory
`beam_memory_atom` / `beam_memory_atom_used`	Gauge	Atom memory allocated / used
`beam_memory_binary`	Gauge	Binary memory
`beam_memory_code`	Gauge	Code memory
`beam_memory_ets`	Gauge	ETS table memory
`beam_processes_count`	Gauge	Erlang process count
`beam_ports_count`	Gauge	Erlang port count
`beam_atom_count`	Gauge	Atom count
`beam_vm_uptime`	Gauge	VM uptime (seconds)

← Overview

OmniCHF Operations Guide

Table of Contents

Overview

OmniCHF implements the **Charging Function (CHF)** of the 5G Core. The CHF provides converged online and offline charging for 5G PDU sessions through the **Nchf_ConvergedCharging** service (3GPP TS 32.290 / TS 32.291 / TS 32.255).

A charging consumer, typically the **SMF**, creates a charging session for each PDU session, reports usage over the lifetime of that session, and finally releases it. OmniCHF translates each 5G charging request into a **CGRateS** SessionS JSON-RPC call for real-time credit authorization (online charging / quota management) and generates a **Charging Data Record (CDR)** on session release (offline charging).

Each charging session is owned by a dedicated **ChargingWorker** GenServer under a `DynamicSupervisor` (process-per-session architecture). All state for one session (SUPI, DNN, S-NSSAI, accumulated volume/duration, rating group, CGRateS session state, charging ID) is held in that worker process. One session

crashing does not affect others. Session state is held in memory and is rebuilt as consumers create new charging sessions after a restart.

Core responsibilities:

- Serve the `nchf-convergedcharging/v3` service on the SBI.
- Allocate a `chargingDataRef` per session and manage its INITIAL → UPDATE → TERMINATE lifecycle.
- Grant service units (GSU) in response to Requested Service Units (RSU), returning granted volume/time and quota-management **triggers** to the consumer.
- Bridge to CGRateS for rating and balance management (online charging).
- Emit CDRs to the application log and, optionally, to per-day offline charging files.
- Register with the NRF and maintain a heartbeat.

Charging Session Lifecycle

Each PDU session maps to one charging session, tracked by a `chargingDataRef` (UUID).

State	Trigger	Action
Created	`POST /chargingdata`	ChargingWorker spawned, CGRateS `InitiateSession` called, initial quota granted
Updated	`POST /chargingdata/{ref}/update`	CGRateS `UpdateSession` called, usage accumulated, sequence number incremented, new quota granted
Released	`POST /chargingdata/{ref}/release`	CGRateS `TerminateSession` called, CDR built and emitted, worker stopped

Session Store and Management Visibility

OmniCHF keeps active-session state in **two** in-memory stores that are kept in step, so the management surfaces see and can act on every live session:

Store	Backing	Holds	Role
Worker registry	`ChargingWorker` GenServer per session under a `DynamicSupervisor`, keyed in a `Registry`	Full per-session state (accumulated volume/duration, sequence number, CGRateS state)	Source of truth for a live session. Create spawns a worker; Update and Release resolve the worker first
Context store	A single in-memory `Agent` map keyed by `chargingDataRef`	A mirror of the same session attributes	Read model for observability and OAM surfaces

On the SBI, **Create** always spawns a worker, and **Update** and **Release** resolve that worker first, falling back to the Context store only if no worker is found (for example after a supervisor blip). Every live worker session is mirrored into the Context store on create and on each update, and is removed from it when the worker stops (normal release, abort, or crash). The management surfaces query the worker registry first and fall back to Context, so both stores present a consistent view.

Operationally this means:

- A session created normally is fully visible through the management surfaces: it appears in `GET /api/sessions` and in the statistics totals, and it can be aborted (`POST /api/oam/charging_session`) and CDR-flushed (`POST /api/oam/cdr/flush`) through the OAM endpoints while it is live.
- The active-session gauge, statistics, and session listings reflect the true live session set, not just a fallback subset.

- Both stores are held in memory and are repopulated as consumers create new sessions after a restart. Because sessions are held in a local registry, a `chargingDataRef` is owned by the instance that created it.

The `omni_chf_sessions_active_count` gauge and the `omni_chf_charging_creates_total` / `omni_chf_charging_releases_total` counters corroborate the live session count reported by `GET /api/sessions`.

3GPP Role and Specification References

Specification	Relevance
TS 23.501	5G System architecture: CHF role, converged charging concept
TS 32.240	Charging architecture and principles
TS 32.290	5G system charging architecture and message flows
TS 32.291	5G system charging: **Nchf_ConvergedCharging** service API (HTTP/2 SBI), ChargingDataRequest/Response data model
TS 32.255	5G data connectivity domain charging (PDU session charging)
TS 32.251	Charging ID format (`chargingId`), TS 32.251 clause 5.2.1.6
TS 32.298	CDR parameter descriptions and encoding
TS 29.500 / 29.501	5G SBI common framework and API design
TS 29.510	NF registration and heartbeat with the NRF

Nchf_ConvergedCharging procedure references (TS 32.291):

Procedure	Clause
Charging Data Create (INITIAL)	6.1.3.2.1
Charging Data Update (UPDATE)	6.1.3.2.2
Charging Data Release (TERMINATE)	6.1.3.2.3
Charging Notify (REAUTHORIZATION / ABORT_CHARGING)	6.1.6.2

SBI Endpoints

All Nchf endpoints are served under the base URL `{sbi_scheme}://{sbi_addr}:{sbi_port}` and the base path `/nchf-convergedcharging/v3`.

Method	Path	Description
`POST`	`/chargingdata`	Create a charging session (**INITIAL**). Allocates a `chargingDataRef`, starts a CGRateS session, grants initial quota.
`POST`	`/chargingdata/{chargingDataRef}/update`	Update a charging session (**UPDATE**). Reports usage and requests additional quota.
`POST`	`/chargingdata/{chargingDataRef}/release`	Release a charging session (**TERMINATE**). Reports final usage, generates a CDR, terminates the CGRateS session.
`POST`	`/notify`	Charging Notify callback trigger. Instructs OmniCHF to send a `REAUTHORIZATION` or `ABORT_CHARGING` notification to the consumer's

Method	Path	Description
		stored `notifyUri`.

On Release, a `500` (cause `SYSTEM_FAILURE`, title "Charging Termination Failed") is returned when the CGRateS `TerminateSession` fails. The CDR is still finalised and written, but the failure is surfaced so a billing-backend inconsistency (a CGRateS session left open) is not hidden behind a false `204`. See Release returns 500.

Request Validation

Every request body is validated for the following mandatory Information Elements before processing. A missing IE returns `400 Bad Request` with cause `MANDATORY_IE_MISSING`:

Mandatory IE	Type
`nfConsumerIdentification`	object
`invocationTimeStamp`	string (ISO 8601)
`invocationSequenceNumber`	integer

ChargingDataRequest: Key Fields Consumed

Field	Type	Used in	Descriptio
`subscriberIdentifier`	string	All	SUPI (e.g. `imsi-999700000000001`). Us CGRateS `Account`. Fa `nfConsumerIdentifi` if absent.
`notifyUri`	string	Create	Callback URI stored fo Charging Notify (REAUTHORIZATION / ABORT_CHARGING).
`pDUSessionChargingInformation`	object	All	PDU session details: `pduSessionInformati` PDU session ID, PDU t type, `qoSInformation` `networkSlicingInfo`.
`multipleUnitUsage`	array	Update, Release	Reported usage conta Every element's `usedUnitContainer` e summed for accumula volume/duration. Also requested `ratingGro`
`requestType`	string	All	`INITIAL_REQUEST`, `UPDATE_REQUEST`, or `TERMINATION_REQUEST`

ChargingDataResponse: Key Fields Returned

Field	Type	Description
`invocationSequenceNumber`	integer	`1` on Create; incremented on each Update.
`invocationTimeStamp`	string	ISO 8601 timestamp of the response.
`multipleUnitInformation`	array	Granted units: one entry with `resultCode: "SUCCESS"`, `grantedUnit` (`totalVolume`, `time`) and the resolved `ratingGroup`.
`triggers`	array	Quota-management triggers instructing the consumer when to report next: a `QUOTA_THRESHOLD` (IMMEDIATE_REPORT at half the granted volume) and a `TIME_LIMIT` (DEFERRED_REPORT at 3600 s).

CGRateS Integration

CGRateS is the external rating and balance engine. OmniCHF acts as a SessionS **client**, mapping each Nchf charging operation onto a SessionS JSON-RPC method:

Nchf operation	CGRateS SessionS method
Create (INITIAL)	`SessionSv1.InitiateSession`
Update (UPDATE)	`SessionSv1.UpdateSession`
Release (TERMINATE)	`SessionSv1.TerminateSession`
Health check	`SessionSv1.GetActiveSessions` (limit 1)

The `MaxUsage` value CGRateS returns is passed straight back to the consumer as the `grantedUnit` (both `totalVolume` and `time`).

Enabled vs Disabled Behavior

Aspect	`cgrates_enabled: true`	`cgrates_enabled: false` **(bypass mode)**
Credit authorization	Real: CGRateS rates the account and returns `MaxUsage`	None: a fixed grant of **86400** units is returned for every Create and Update
Balance impact	Subscriber balance is debited/reserved in CGRateS	No balance is touched
Terminate	`TerminateSession` finalizes the CGRateS session and CDR	No-op
Health check	Live probe against `cgrates_url`	Always reports `reachable: false`, `cgrates_url: "disabled"`
Log signature	`Initiating CGRateS session for ...`	`CGRateS integration disabled, returning default authorization` (warning)
Use case	Production online charging	Lab / integration testing without a rating engine

> In bypass mode OmniCHF still allocates `chargingDataRef`s, tracks sessions, increments sequence numbers, emits triggers, and generates CDRs; only the credit control is stubbed. This makes it safe to exercise the full SMF↔CHF flow without a CGRateS deployment.

Configuration of the CGRateS integration (`cgrates_enabled`, `cgrates_url`, `cgrates_tenant`, `cgrates_timeout`) is documented in the Configuration Reference.

5G-to-CGRateS Event Mapping

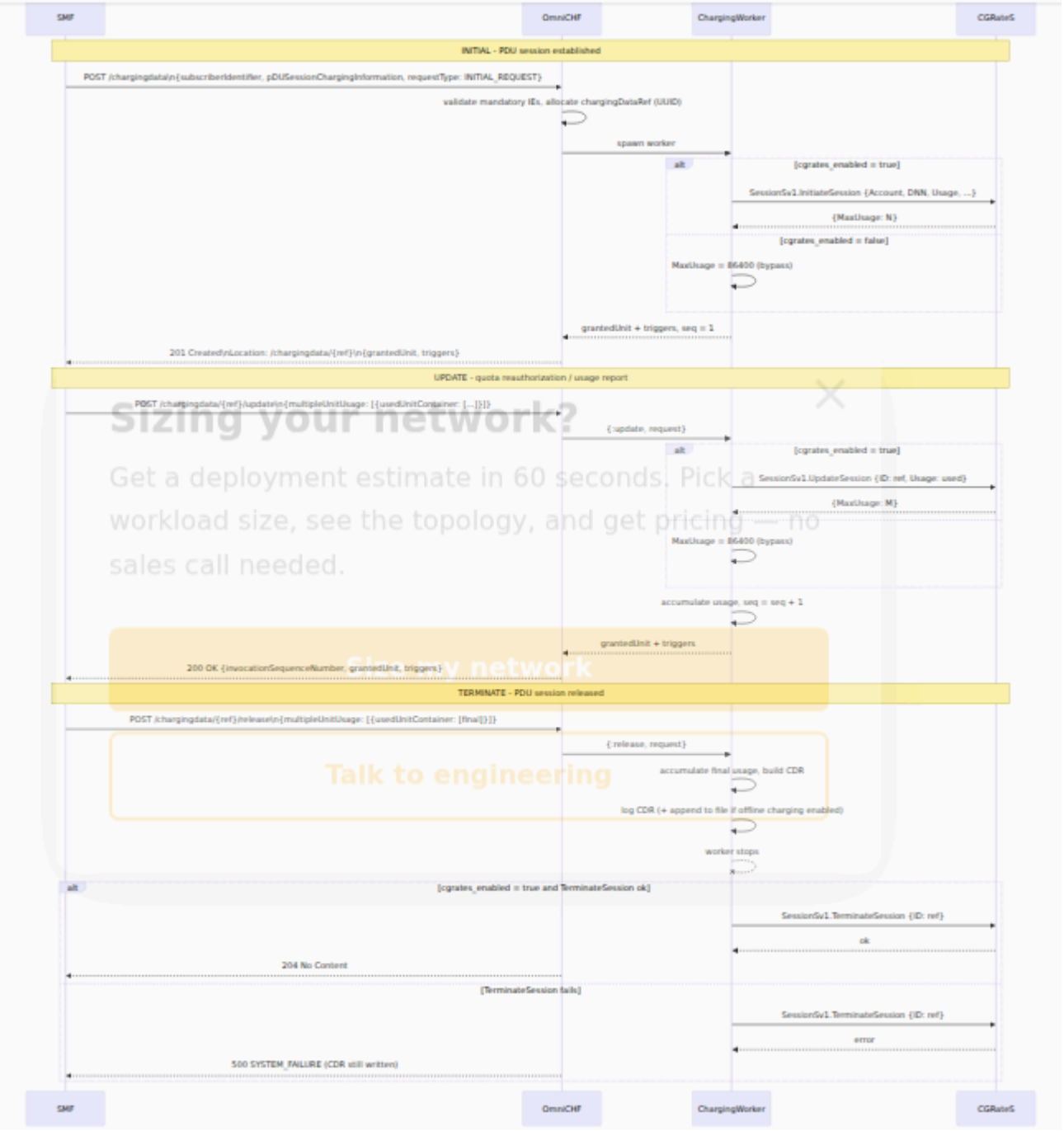
Each SessionS call carries an `Event` map built from the ChargingDataRequest:

CGRateS Event Field	Source	
`Account` / `Subject`	`subscriberIdentifier` (SUPI)	Falls back to `nfConsumerI` else `"unknow`
`Destination` / `DNN`	`pduSessionInformation.dnnId` (or `.dnn`)	Data Networ
`ToR`	derived from `pduType`	Always `*data`
`RequestType`	`requestType`	Mapped to `*`
`Category`	static	`"data"`.
`Usage`	`usedUnitContainer`	First non-zero then `uplinkV` `downlinkVolu`
`OriginID`	`chargingDataRef`	Unique per s
`OriginHost`	static	`"OmniCHF"`.
`SUPI`	`subscriberIdentifier`	5G extension
`S-NSSAI_SST` / `S-NSSAI_SD`	`pduSessionInformation.sNSSAI`	Defaults: SST
`5QI`	`qoSInformation.5qI` (or `fiveQI`)	Default `9`.
`RATType`	`pduSessionInformation.ratType`	Default `"NR"`
`PDUSessionID`	`pduSessionInformation.pduSessionID`	Default `0`.
`PDUSessionType`	`pduSessionInformation.pduType`	Default `"IPV`

CGRateS Event Field	Source	
`ChargingOperation`	operation	`Initial` / `Up`

Key Procedures

Converged Charging Session (INITIAL → UPDATE → TERMINATE)



> If CGRateS `TerminateSession` fails on release, OmniCHF **still** finalizes and writes the CDR, so the offline record is not lost. The consumer then receives `500` (cause `SYSTEM_FAILURE`) rather than a false `204`, because the CGRateS-side session may be left open and the billing backend may need

> reconciliation. On a successful terminate the consumer receives `204 No Content`.

Quota Management (RSU / GSU and Triggers)

Per TS 32.291, the consumer sends Requested Service Units (RSU) and OmniCHF returns Granted Service Units (GSU) plus **triggers** that tell the consumer when to come back:

The consumer re-authorizes (sends an UPDATE) when it hits the volume threshold or the time limit. OmniCHF can also proactively drive re-authorization or teardown via the `/notify` endpoint (`REAUTHORIZATION` / `ABORT_CHARGING`), which posts to the `notifyUri` supplied on the INITIAL request.

Charging Notify (REAUTHORIZATION / ABORT_CHARGING)

Per TS 32.291 clause 6.1.6.2, OmniCHF pushes a `ChargingNotifyRequest` to the consumer's stored `notifyUri`. The trigger is the operator-facing `POST /nchf-convergedcharging/v3/notify` endpoint, whose body carries the target `chargingDataRef` and the `notificationType`. OmniCHF looks up the session, resolves its `notifyUri`, and POSTs the notification to the consumer.

> The notification is best-effort toward the operator surface: a rejected or unreachable `notifyUri` is logged but does not change the `/notify` result once the session and `notifyUri` are resolved. A session with no stored `notifyUri` yields an internal `no_notify_uri` error.

CDR Fields

A CDR is built on release, logged at INFO (`CHF CDR: %{...}`), and, when `offline_charging_enabled` is `true`, appended as one JSON line to `cdr_YYYYMMDD.json`:

Field	Source
`record_type`	Static `"5G_PDU_SESSION"`
`supi`	Session context
`dnn`	Session context / `pduSessionInformation`
`snssai`	`{sst, sd}`
`qos_5qi`	`qoSInformation.5qI` (default `9`)
`rat_type`	`pduSessionInformation.ratType` (default `"NR"`)
`pdu_session_id` / `pdu_session_type`	Session context / request
`volume_uplink` / `volume_downlink` / `volume_total`	Accumulated from all `usedUnitContainer`s
`duration`	Reported `time`, else wall-clock difference
`start_time` / `end_time`	Session creation / release timestamps
`charging_data_ref`	Session UUID
`charging_id`	3GPP 32-bit charging ID (TS 32.251)
`recording_entity`	OmniCHF SBI base URL
`rating_group`	Resolved from DNN mapping

Offline CDR output (`offline_charging_enabled`, `cdr_output_dir`) and charging-ID composition (`node_id`) are configured in the Configuration Reference.

Running Offline Charging

To capture CDRs to file for mediation/billing:

1. Set `offline_charging_enabled: true` and point `cdr_output_dir` at a writable directory.
2. CDRs accumulate one JSON object per line into a per-day file named `cdr_YYYYMMDD.json` (the date is the UTC release date). A new file is opened each UTC day; there is no automatic rotation or retention beyond the daily filename split, so age out old files with your own housekeeping.
3. Each released session appends exactly one CDR. In-progress sessions hold their usage in the worker/context until release; they are not written to file until then.

If `offline_charging_enabled` is `false`, CDRs are still built and logged at INFO as `CHF CDR: %{...}`; only the file write is skipped. To capture CDRs without file output, ship those log lines with your log pipeline. See CDRs missing from offline files.

Locating and Searching CDRs

The Management API exposes `GET /api/oam/cdr` to search the written CDR files. It reads every `cdr_*.json` file under `cdr_output_dir` and filters on the supplied query parameters (all optional, combined with AND):

Query parameter	Matches	Format
`supi`	Exact CDR `supi`	e.g. `imsi-999700000000001`
`dnn`	Exact CDR `dnn`	e.g. `internet`
`date_from`	CDR file date `>=` this date	`YYYY-MM-DD` (hyphens optional)
`date_to`	CDR file date `<=` this date	`YYYY-MM-DD` (hyphens optional)

The date filters are applied to the **file name's date**, not to individual record timestamps, so a day's whole file is included or excluded as a unit. With no parameters the endpoint returns every CDR on disk. Because it reads files, `GET /api/oam/cdr` only returns records when `offline_charging_enabled` was on at release time.

Force-Flushing a CDR (OAM)

In normal operation a CDR is only produced when a session is released. `POST /api/oam/cdr/flush` lets an operator emit a CDR early for any live session (see Session Store and Management Visibility), for example to close out sessions the consumer never released:

Request body	Effect
`{"charging_data_ref": "<ref>"}`	Build and write a CDR for that one session, then finalise it. `404` if the ref is not found.
`{}` or `{"charging_data_ref": "all"}`	Flush every session currently tracked.

The flush builds the CDR from the session's accumulated usage plus the last `ChargingDataRequest` and reuses the same CDR writer as release; it does **not**

call CGRateS `TerminateSession`. The response reports `write_result` and `offline_charging_enabled` per session. If offline charging is disabled the CDR is built and the session finalised, but nothing is written to file (`write_result: "ok"` with no file output). Each flush increments `omni_chf_charging_releases_total{result="oam_flush"}`.

← Overview

OmniCHF Troubleshooting

400 Bad Request on any charging request

Symptoms: Consumer receives `400` with cause `MANDATORY_IE_MISSING`.

Possible causes:

- The request body is missing `nfConsumerIdentification`, `invocationTimeStamp`, or `invocationSequenceNumber`.

Resolution:

1. Confirm the consumer includes all three mandatory IEs on every operation (Create, Update, Release).
2. Check the `detail` field of the ProblemDetails response; it lists the missing IE names.

404 on Update or Release

Symptoms: Consumer receives `404` with cause `CHARGING_DATA_NOT_FOUND`.

Possible causes:

- OmniCHF restarted between Create and the later operation; session state is held in memory, so the session must be re-created after a restart.
- The `chargingDataRef` in the path is wrong.
- The session was already released (its worker has stopped).

Resolution:

1. Grep logs for `CHF Update: unknown ref=` / `CHF Release: unknown ref=`.
2. Confirm OmniCHF has not restarted since the session was created.
3. Confirm the consumer is echoing back the `chargingDataRef` from the Create `Location` header.

500 on Create with CGRateS enabled

Symptoms: Consumer receives `500` with cause `CHARGING_FAILED`.

Possible causes:

- CGRateS `InitiateSession` failed: unreachable engine, wrong tenant, or a CGRateS-side error.

Resolution:

1. Verify `cgrates_url` points to a reachable CGRateS `/jsonrpc` endpoint.
2. Verify `cgrates_tenant` matches the CGRateS tenant.
3. Check the `omni_chf_cgrates_health` gauge (`1` = up).
4. Review logs for `CGRateS InitiateSession failed ...`; the reason is one of `{:cgrates_error, msg}`, `{:http_error, status}`, or `{:http_error, reason}`.
5. Confirm the `SessionSv1` service is enabled in CGRateS.

Release returns 500

Symptoms: Consumer receives `500` with cause `SYSTEM_FAILURE` and title "Charging Termination Failed" on `POST /chargingdata/{ref}/release`.

Possible causes:

- The CGRateS `TerminateSession` failed during Release (unreachable engine, wrong tenant, or a CGRateS-side error).

Impact:

- The CDR is still built and written, so the offline record is not lost.
- The CGRateS-side session may be left open, so the billing backend can be inconsistent and needs reconciliation.

Resolution:

1. Grep logs for `CGRateS TerminateSession failed` on the affected `ref`.
2. Confirm CGRateS reachability with the `omni_chf_cgrates_health` gauge (`1` = up).
3. Reconcile any CGRateS session left open for that `ref` against the written CDR.
4. Correlate with `omni_chf_charging_releases_total{result="failure"}` to see the terminate-failure rate.

CGRateS health check fails while the daemon is up

Symptoms: `omni_chf_cgrates_health` reads `0` but CGRateS is running.

Possible causes:

- CGRateS is slow: the health check caps its timeout at `min(cgrates_timeout, 3000)` ms.
- Scheme mismatch (HTTP vs HTTPS) on `cgrates_url`.

Resolution:

1. Confirm `cgrates_url` scheme and port are correct.
2. If CGRateS is legitimately slow, investigate CGRateS load rather than raising the cap.
3. Look for `CGRateS unreachable at <url> ...` warnings.

High CGRateS latency

Symptoms: Slow charging operations; elevated `omni_chf_cgrates_request_duration_ms`.

Resolution:

1. Inspect the histogram p95/p99 by `operation`.
2. Scale CGRateS or the network path between OmniCHF and CGRateS.
3. Consider lowering `cgrates_timeout` so slow calls fail fast rather than blocking the session.

CDRs missing from offline files

Symptoms: `CHF CDR:` lines appear in the log, but no `cdr_YYYYMMDD.json` file is written.

Possible causes:

- `offline_charging_enabled` is `false` (default): CDRs are logged only.
- `cdr_output_dir` is not writable.

Resolution:

1. Set `offline_charging_enabled: true`.
2. Ensure `cdr_output_dir` exists and is writable; watch for `[CDR] Failed to write CDR to file` errors.
3. To capture CDRs without file output, ship the `CHF CDR:` log lines with your log pipeline.

Active session count not decreasing

Symptoms: `omni_chf_sessions_active_count` stays elevated.

Possible causes:

- The consumer never sends Release (sessions only clear on release or worker crash).
- Releases are returning `404` (see above) so the consumer considers them done while OmniCHF may not.

Resolution:

1. Correlate active count with Create vs Release rates.
2. Investigate `404` on Release, which usually indicates a prior restart.

NRF registration not maintained

Symptoms: `omni_chf_nrf_registration_status` reads `0`.

Resolution:

1. Verify `nrf_uri` is reachable from OmniCHF's `sbi_addr`.
2. Confirm `mcc` / `mnc` match the NRF PLMN configuration.
3. Confirm `sbi_addr` / `sbi_port` in the advertised profile are reachable by the NRF.
4. Review startup logs for NRF registration errors.