← 述

OmniCHF 置参考

操作员配置从应用程序环境键 `:omnichf` 中读取，在启动时由 `config/runtime.exs` 填充。每个键都可以通过操作系统环境变量覆盖。下面的示例显示了完整的运行时配置，以及在相应环境变量未设置时应用的值。

```
config :omnichf,
  # SBI（基于服务的接口）监听器
  sbi_scheme: "http",
  sbi_addr:   "127.0.0.14",
  sbi_port:   7777,

  # NRF 注册 / 心跳
  nrf_uri:    "http://127.0.0.1:7777",

  # 提供的 PLMN 身份
  mcc: "999",
  mnc: "70",

  # 可观察性
  prometheus_metrics_port: 9568,

  # NRF 心跳频率
  heartbeat_interval: 10_000,

  # CGRateS 计费 / 余额引擎集成
  cgrates_enabled: true,
  cgrates_url:     "http://localhost:2080/jsonrpc",
  cgrates_tenant:  "cgrates.org",
  cgrates_timeout: 5000

# 结构化 JSON 日志记录
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all
```

□□□□

□□	□□	□□	□□	□
`sbi_scheme`	□□□	□	`"http"`	`SBI_SCHE`
`sbi_addr`	□□□	□	`"127.0.0.14"`	`SBI_ADDR`
`sbi_port`	□□	□	`7777`	`SBI_PORT`
`nrf_uri`	□□□	□	`"http://127.0.0.1:7777"`	`NRF_URI`
`mcc`	□□□	□	`"999"`	`MCC`
`mnc`	□□□	□	`"70"`	`MNC`
`prometheus_metrics_port`	□□	□	`9568`	`PROMETHE`

参数	类型	必需	默认	[
`heartbeat_interval`	整数（毫秒）	否	`10000`	`HEARTBEA`

CGRateS 参数

这些键控制与外部 CGRateS 计费和余额引擎的集成。有关行为细节，请参见 CGRateS 集成。

参数	类型	必需	默认	环境
`cgrates_enabled`	布尔值	否	`true` (`"true"`)	`CGRATES_`
`cgrates_url`	字符串	否	`"http://localhost:2080/jsonrpc"`	`CGRATES_`
`cgrates_tenant`	字符串	否	`"cgrates.org"`	`CGRATES_`
`cgrates_timeout`	整数（毫秒）	否	`5000`	`CGRATES_`

> 在构建时禁用的默认值：编译的默认值（如果应用程序环境从未由 `runtime.exs` 填充）为 `cgrates_enabled: false`。在正常部署中，`runtime.exs` 设置它，并且默认值为 `true`。测试配置强制 `cgrates_enabled: false`。

高级 / 可选键

这些键不是由默认的 `runtime.exs` 填充的，但如果存在，则从 `:omnichf` 环境中读取。它们调整计费组解析、离线计费和计费 ID 生成。它们没有专用的环境变量，而是直接在 `:omnichf` 配置块中设置。

参数	类型	必需	默认	环境变量	
`rating_groups`	映射	否	`%{}`	无	DNN - 配一个键[multi 和 CDR
`default_rating_group`	整数	否	`1`	无	当在 ra DNN [
`offline_charging_enabled`	布尔值	否	`false`	无	当为 tr 了被记录[
`cdr_output_dir`	字符串	否	`"/var/log/omnichf/cdr"`	无	离线 CD （cdr_ JSON offli 为 tru
`node_id`	整数	否	`1`	无	用于根据 合 32 [点标识符[一值，以[

计费组

`rating_groups` 将 **DNN**（映射键，字符串）映射到应用于该 DNN 会话的计费组（值，整数）。解析的计费组出现在返回 ���消费者的 `multipleUnitInformation` 的 `ratingGroup` 字段和 CDR 的 `rating_group` 字段中。解析规则如下：

1. 如果会话的 DNN 是 `rating_groups` 中的键，则使用该值。
2. 否则使用 `default_rating_group`。

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

匹配基于精确的 DNN 字符串。没有条目的 DNN（且没有 `default_rating_group`）将回退到计费组 `1`。格式错误的 `rating_groups` 值（不是映射）将被忽略，使用默认值，因此错误的配置永远不会阻止计费。

日志记录

结构化 JSON 日志记录通过 `:logger` 的 `:default_formatter` 配置。

参数	类型	默认	描述
`format`	元组	`{OmniLogger.JsonFormatter, :format}`	日志行格式化器。发出结构化 JSON，适合被日志发送器摄取。
`metadata`	原子 / 列表	`:all`	每行包含的日志元数据。`:all` 发出所有附加的元数据（SUPI、DNN、PDU 会话 ID、计费 ID、过程等）。

管理接口

OmniCHF 在 SBI 旁边暴露操作员管理界面。这些端口与监控和 OAM 相关，但不是 Nchf 服务的一部分：

接口	端口	方案	目的
SBI (Nchf)	`sbi_port`（默认 `7777`）	HTTP	5G 互 NF 计费流量。
Prometheus 指标	`prometheus_metrics_port`（默认 `9568`）	HTTP	指标抓取端点。
管理 API	`8443`	HTTPS	状态读取、会话/统计/CGRateS 健康读取、CDR 搜索和 OAM 操作（配置、CDR 刷新、日志级别、会话中止、NRF 重新注册）。
控制面板	`7443`	HTTPS	资源、配置、许可证和日志的 Web UI。

管理 / **OAM API** 端点

8443 端口（HTTPS）的管理 API 暴露以下操作员界面：

方法	路径	目的
`GET`	`/api/status/api`	读取 API/服务状态。
`GET`	`/api/status/license`	读取许可证状态。
`GET`	`/api/status/nrf`	读取 NRF 注册状态。
`GET`	`/api/status/nf`	读取 NF（节点）状态。
`GET`	`/api/sessions`	列出活动计费会话。
`GET`	`/api/sessions/{id}`	通过 `charging_data_ref` 检查单个计费会话。
`GET`	`/api/statistics`	读取计费统计信息（活动会话计数、总计费量、CGRateS 可达性）。
`GET`	`/api/health/cgrates`	读取 CGRateS 连接健康。
`GET`	`/api/oam/config`	读取运行时配置。
`PATCH`	`/api/oam/config/{id}`	更新运行时配置键。
`POST`	`/api/oam/charging_session`	通过 `charging_data_ref` 中止计费会话。
`GET`	`/api/oam/cdr`	搜索已完成的 CDR。查询参数：`supi`、`dnn`、`date_from`、`date_to`（均为可选）。
`POST`	`/api/oam/cdr/flush`	强制刷新累积的使用量作为一个会话的 CDR（`{"charging_data_ref": "<ref>"}`）或所有跟踪的会话（`{}` / `"all"`）。
`POST`	`/api/oam/log_level`	更改运行时日志级别。
`POST`	`/api/oam/nrf/reregister`	触发 NRF 重新注册。

> 会话存储可见性：面向会话的管理界面（`/api/sessions`、`/api/statistics`、`/api/oam/charging_session`、`/api/oam/cdr/flush`）展示每个活动会话。实时的 SBI 计费路径在每个会话中保持每个会话在一个 工作者 进程中，并且每个工作者会话在创建和更新时被镜像到内存中的 上下文存储 中，并在停止时被移除。管理界面首先查询工作者注册表，然后回退到上下文，因此实时会话始终可见且可操作。有关这在操作上的意义，请参见 会话存储和管理可见性。CDR 搜索（`/api/oam/cdr`）读取离线 CDR 文件，并且与两个存储无关。

← 概述

OmniCHF 指标参考

指标在 `prometheus_metrics_port`（默认 `9568`）的 `/metrics` 上公开。

充电指标

指标：`omni_chf_charging_initial_count`

类型：计数器

描述：接收到的充电会话初始（创建）请求。

标签：`supi`

指标：`omni_chf_charging_update_count`

类型：计数器

描述：接收到的充电会话更新请求。

标签：`ref`

指标：`omni_chf_charging_release_count`

类型：计数器

描述：接收到的充电会话释放请求。

标签：`ref`

指标：`omni_chf_charging_creates_total`

类型：计数器

描述：创建操作结果。

标签：`result`（`success` / `failure`）

指标：`omni_chf_charging_updates_total`

类型：计数器

描述：更新操作结果。

标签：`result`（`success` / `failure`）

指标：`omni_chf_charging_releases_total`

类型：计数器

描述：释放操作结果。通过成功的 CGRateS 终止（`success`）通过 SBI 释放，或者在 CGRateS `TerminateSession` 失败但 CDR 仍然被最终确定的情况下通过 SBI 释放（`failure`，向消费者显示为 `500`），或者

通过 OAM CDR 清除端点强制最终确定（`oam_flush`）。
标签：`result`（`success` / `failure` / `oam_flush`）

指标：`omni_chf_sessions_active_count`
类型：测量值
描述：当前在内存中持有的活跃充电会话。每次上下文存储创建和删除时发出，包括每个镜像的实时工作会话，因此测量值跟踪真实的活跃会话计数。
标签：无

```
# 充电会话创建速率
rate(omni_chf_charging_creates_total{result="success"}[5m])

# 创建失败比例
sum(rate(omni_chf_charging_creates_total{result="failure"}[5m]))
  / sum(rate(omni_chf_charging_creates_total[5m]))

# 当前活跃会话
omni_chf_sessions_active_count
```

CGRateS 指标

指标：`omni_chf_cgrates_requests_total`
类型：计数器
描述：CGRateS JSON-RPC 请求。
标签：`operation`（SessionS 方法），`result`（`success` / `failure`）

指标：`omni_chf_cgrates_request_duration_ms`
类型：直方图
描述：CGRateS JSON-RPC 请求持续时间（毫秒）。
标签：`operation`
桶：5, 10, 25, 50, 100, 250, 500, 1000, 2500

指标：`omni_chf_cgrates_health`
类型：测量值
描述：CGRateS 连接健康状况（`1` = 可达，`0` = 不可达/禁用）。
标签：无

```
# CGRateS p99 延迟按操作
histogram_quantile(0.99,
  sum by (le, operation)
(rate(omni_chf_cgrates_request_duration_ms_bucket[5m])))

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

# CGRateS 可达性□当为 0 时警报）
omni_chf_cgrates_health
```

NRF 指标

指标：`omni_chf_nrf_registration_status`

类型：测量值

描述：NRF 注册状态（`1` = 已注册，`0` = 未注册）。

标签：`nf_type`

```
omni_chf_nrf_registration_status
```

BEAM VM 指标

标准运行时健康测量值也被导出。

指标	类型	描述
`beam_memory_total`	测量值	总 BEAM 内存（字节）
`beam_memory_processes` / `beam_memory_processes_used`	测量值	进程内存分配/使用
`beam_memory_system`	测量值	系统（非进程）内存
`beam_memory_atom` / `beam_memory_atom_used`	测量值	原子内存分配/使用
`beam_memory_binary`	测量值	二进制内存
`beam_memory_code`	测量值	代码内存
`beam_memory_ets`	测量值	ETS 表内存
`beam_processes_count`	测量值	Erlang 进程计数
`beam_ports_count`	测量值	Erlang 端口计数
`beam_atom_count`	测量值	原子计数
`beam_vm_uptime`	测量值	VM 启动时间（秒）

← 述

OmniCHF 操作指南

目录

1. 述
2. 充电会话生命周期
3. 3GPP 角色和规范参考
4. SBI 端点
5. CGRateS 集成
6. 关键程序

概述

OmniCHF 实现了 5G 核心的 充电功能 **(CHF)**。CHF 通过 **Nchf_ConvergedCharging** 服务（3GPP TS 32.290 / TS 32.291 / TS 32.255）提供 5G PDU 会话的在线和离线充电。

充电消费者，通常是 **SMF**，为每个 PDU 会话创建一个充电会话，在该会话的生命周期内报告使用情况，最后释放它。OmniCHF 将每个 5G 充电请求转换为 **CGRateS** SessionS JSON-RPC 调用，以进行实时信用授权（在线充电/配额管理），并在会话释放时生成 充电数据记录 **(CDR)**（离线充电）。

每个充电会话由一个专用的 **ChargingWorker** GenServer 在 `DynamicSupervisor` 下拥有（每会话一个进程架构）。一个��话的所有状态（SUPI、DNN、S-NSSAI、累计量/持续时间、计费组、CGRateS 会话状态、充电 ID）都保存在该工作进程中。一个会话崩溃不会影响其他会话。会话状态保存在内存中，并在消费者在重启后创建新的充电会话时重建。

核心职责：

- 在 SBI 上提供 `nchf-convergedcharging/v3` 服务。
- 为每个会话分配一个 `chargingDataRef` 并管理其 INITIAL → UPDATE → TERMINATE 生命周期。
- 响应请求的服务单元（RSU）授予服务单元（GSU），返回授予的量/时间和配额管理 触发器 给消费者。
- 通过 CGRateS 进行计费和余额管理（在线充电）。
- 将 CDR 发送到应用日志，并可选地发送到每日离线充电文件。

- 向 NRF 注册并保持心跳。

充电会话生命周期

每个 PDU 会话映射到一个充电会话，由 `chargingDataRef`（UUID）跟踪。

状态	触发器	动作
创建	`POST /chargingdata`	生成 ChargingWorker，调用 CGRateS `InitiateSession`，授予初始配额
更新	`POST /chargingdata/{ref}/update`	调用 CGRateS `UpdateSession`，累计使用情况，序列号递增，授予新配额
释放	`POST /chargingdata/{ref}/release`	调用 CGRateS `TerminateSession`，构建并发出 CDR，停止工作

会话存储和管理可见性

OmniCHF 在两个内存存储中保持活动会话状态，这两个存储保持同步，因此管理界面可以看到并对每个活动会话进行操作：

存储	支持	保存	角色
工作者注册表	每个会话下的 `ChargingWorker` GenServer，在 `DynamicSupervisor` 下，键入 `Registry`	每个会话的完整状态（累计量/持续时间、序列号、CGRateS 状态）	活动会话的真实来源。创建时生成工作者；更新和释放时优先解析工作者
上下文存储	一个单一的内存 `Agent` 映射，键入 `chargingDataRef`	相同会话属性的镜像	可观察性和 OAM 界面的读取模型

在 SBI 上，创建 始终生成一个工作者，而 更新 和 释放 首先解析该工作者，仅在找不到工作者时回退到上下文存储（例如在监督器故障后）。每个活动工作者会话在创建和每次更新时都被镜像到上下文存储中，并在工作者停止时从中移除（正常释放、终止或崩溃）。管理界面首先查询工作者注册表，然后回退到上下文，因此两个存储都呈现一致的视图。

这在操作上意味着：

- 正常创建的会话在管理界面中完全可见：它出现在 `GET /api/sessions` 和统计总数中，并且可以在活动期间通过 OAM 端点中止（`POST /api/oam/charging_session`）和 CDR 刷新（`POST /api/oam/cdr/flush`）。
- 活动会话计量器、统计信息和会话列表反映真实的活动会话集，而不仅仅是回退子集。
- 两个存储都保存在内存中，并在消费者在重启后创建新会话时重新填充。由于会话保存在本地注册表中，`chargingDataRef` 由创建它的实例拥有。

`omni_chf_sessions_active_count` 计量器和 `omni_chf_charging_creates_total` / `omni_chf_charging_releases_total` 计数器证实了 `GET /api/sessions` 报告的活动会话计数。

3GPP 角色和规范参考

规范	相关性
TS 23.501	5G 系统架构：CHF 角色，融合充电□念
TS 32.240	充电架构和原则
TS 32.290	5G 系统充电架构和消息流
TS 32.291	5G 系统充电：**Nchf_ConvergedCharging** 服务 API（HTTP/2 SBI），ChargingDataRequest/Response 数据模型
TS 32.255	5G 数据连接域充电（PDU 会话充电）
TS 32.251	充电 ID 格式（`chargingId`），TS 32.251 条款 5.2.1.6
TS 32.298	CDR 参数描述和编码
TS 29.500 / 29.501	5G SBI 通用框架和 API 设计
TS 29.510	NF 注册和与 NRF 的心跳

Nchf_ConvergedCharging 程序参考（TS 32.291）：

程序	条款
充电数据创建（INITIAL）	6.1.3.2.1
充电数据更新（UPDATE）	6.1.3.2.2
充电数据释放（TERMINATE）	6.1.3.2.3
充电通知（REAUTHORIZATION / ABORT_CHARGING）	6.1.6.2

SBI 端点

所有 Nchf 端点都在基本 URL `{sbi_scheme}://{sbi_addr}:{sbi_port}` 和基本路径 `/nchf-convergedcharging/v3` 下提供。

方法	路径	描述
`POST`	`/chargingdata`	创建一个充电会话（**INITIAL**）。分配一个 `chargingDataRef`，启动一个 CGRateS 会话，授予初始配额。
`POST`	`/chargingdata/{chargingDataRef}/update`	更新一个充电会话（**UPDATE**）。报告使用情况并请求额外配额。
`POST`	`/chargingdata/{chargingDataRef}/release`	释放一个充电会话（**TERMINATE**）。报告最终使用情况，生成 CDR，终止 CGRateS 会话。
`POST`	`/notify`	充电通知回调触发器。指示 OmniCHF 向消费者存储的 `notifyUri` 发送 `REAUTHORIZATION` 或 `ABORT_CHARGING` 通知。

在释放时，当 CGRateS `TerminateSession` 失败时，将���回 `500`（原因 `SYSTEM_FAILURE`，标题 "Charging Termination Failed"）。CDR 仍然会被最终确定并写入，但该错误会被显示，以便不隐藏账单后端不一致性（CGRateS 会话未关闭）。请参见 Release returns 500。

请求验证

在处理之前，每个请求体都会验证以下强制信息元素。缺少 IE 将返回 `400 Bad Request`，原因是 `MANDATORY_IE_MISSING`：

强制 **IE**	类型
`nfConsumerIdentification`	对象
`invocationTimeStamp`	字符串（ISO 8601）
`invocationSequenceNumber`	整数

ChargingDataRequest: 关键字段

字段	类型	用于	描述
`subscriberIdentifier`	字符串	所有	SUPI（例如 `imsi-999700000000001`）用作 CGRateS 的 `Account`。如果缺失，则[`nfConsumerIdentification.sup`
`notifyUri`	字符串	创建	存储的回调 URI，用于后续充电通知（REAUTHORIZATION / ABORT_CHARGING）。
`pDUSessionChargingInformation`	对象	所有	PDU 会话详细信息：`pduSessionInformation`（DNN、PDU 会话 ID、PDU 类型、RAT 类型、`qoSInformation`、`networkSlicingInfo.sNSSAI`）。
`multipleUnitUsage`	数组	更新、释放	报告的使用容器。每个元素的 `usedUnitContainer` 条目被求和以获得累[持续时间。还携带请求的 `ratingGroup`。
`requestType`	字符串	所有	`INITIAL_REQUEST`、`UPDATE_REQUEST` 或 `TERMINATION_REQUEST`。

ChargingDataResponse: 关键字段

字段	类型	描述
`invocationSequenceNumber`	整数	创建时为 `1`；每次更新时递增。
`invocationTimeStamp`	字符串	响应的 ISO 8601 时间戳。
`multipleUnitInformation`	数组	授予的单元：一个条目，带有 `resultCode: "SUCCESS"`、`grantedUnit`（`totalVolume`、`time`）和解析的 `ratingGroup`。
`triggers`	数组	配额管理触发器，指示消费者何时报告下一个：`QUOTA_THRESHOLD`（在授予量的一半时进行 IMMEDIATE_REPORT）和 `TIME_LIMIT`（在 3600 秒时进行 DEFERRED_REPORT）。

CGRateS 集成

CGRateS 是外部计费和余额引擎。OmniCHF 作为 SessionS 客户端，将每个 Nchf 充电操作映射到 SessionS JSON-RPC 方法：

Nchf 操作	**CGRateS SessionS** 方法
创建（INITIAL）	`SessionSv1.InitiateSession`
更新（UPDATE）	`SessionSv1.UpdateSession`
释放（TERMINATE）	`SessionSv1.TerminateSession`
健康检查	`SessionSv1.GetActiveSessions`（限制为 1）

CGRateS 返回的 `MaxUsage` 值直接返回给消费者作为 `grantedUnit`（`totalVolume` 和 `time`）。

启用与禁用行为

方面	`cgrates_enabled: true`	`cgrates_enabled: false`（旁路模式）
信用授权	实际：CGRateS 对账户进行计费并返回 `MaxUsage`	无：每次创建和更新返回固定的 **86400** 单位
余额影响	订阅者余额在 CGRateS 中被扣除/保留	不触及余额
终止	`TerminateSession` 完成 CGRateS 会话和 CDR	无操作
健康检查	针对 `cgrates_url` 的实时探测	始终报告 `reachable: false`，`cgrates_url: "disabled"`
日志签名	`Initiating CGRateS session for ...`	`CGRateS integration disabled, returning default authorization`（警告）
用例	生产在线充电	实验室/集成测试，无计费引擎

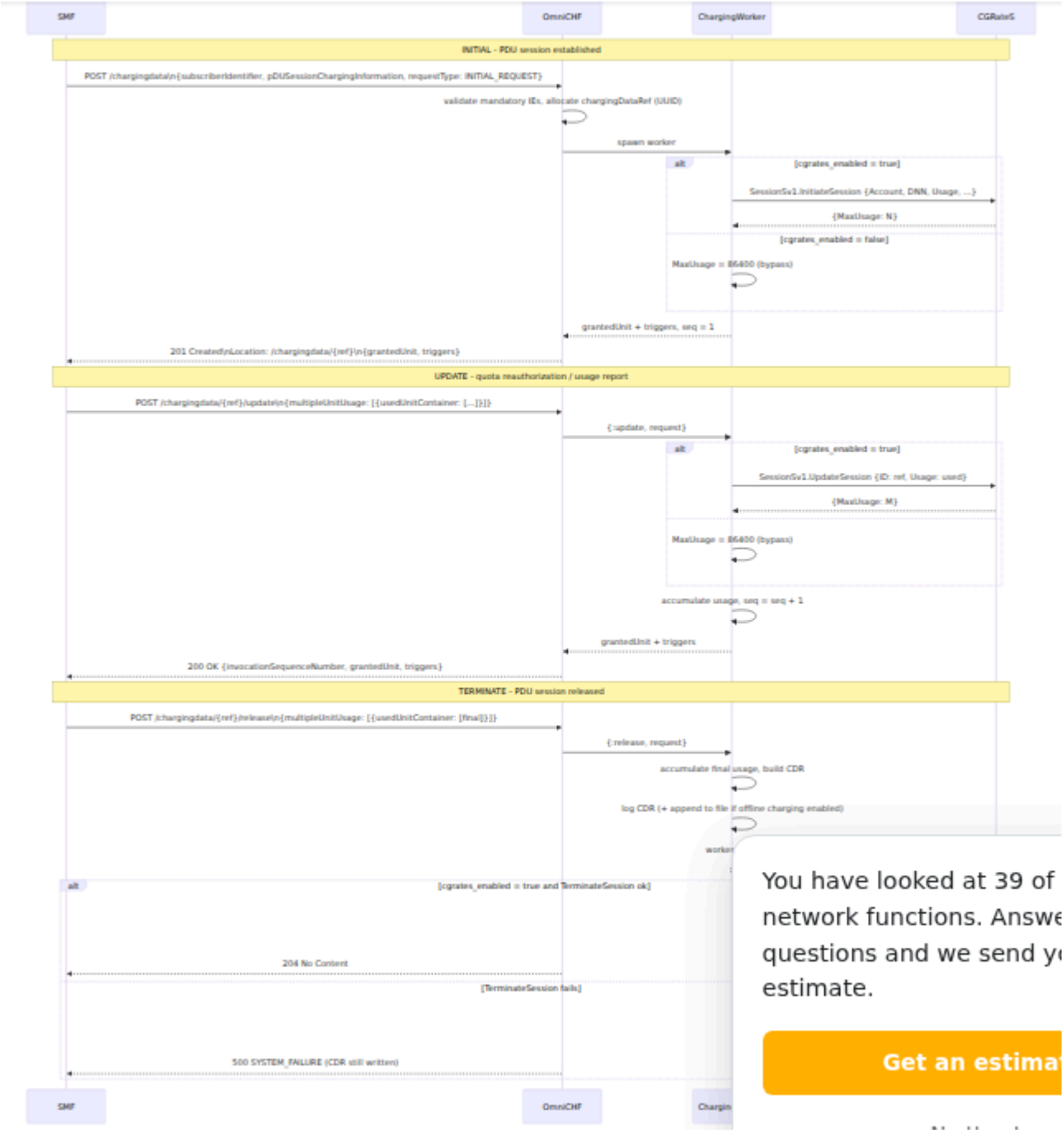
> 在旁路模式下，OmniCHF 仍然分配 `chargingDataRef`，跟踪会话，递增序列号，发出触发器，并生成 CDR；只有信用控制被存根。这使得在没有 CGRateS 部署的情况下安全地执行完整的 SMF↔CHF 流程。

CGRateS 集成的配置（`cgrates_enabled`、`cgrates_url`、`cgrates_tenant`、`cgrates_timeout`）在 配置参考 中进行了文档说明。

5G 到 CGRateS 事件映射

每个 SessionS 调用携带一个��� ChargingDataRequest 构建的 `Event` 映射：

CGRateS 事件字段	来源	
`Account` / `Subject`	`subscriberIdentifier`（SUPI）	如果缺失，则回退到 `nfConsumerI` 否则为 `"unknow`
`Destination` / `DNN`	`pduSessionInformation.dnnId`（或 `.dnn`）	数据网络名称。
`ToR`	从 `pduType` 派生	对于 PDU 会话始终
`RequestType`	`requestType`	映射到 `*prepai`
`Category`	静态	`"data"`。
`Usage`	`usedUnitContainer`	首先非零的 `totalV` `uplinkVolum` 最后是 `time`。
`OriginID`	`chargingDataRef`	每个会话唯一。
`OriginHost`	静态	`"OmniCHF"`。
`SUPI`	`subscriberIdentifier`	5G 扩展字段。
`S-NSSAI_SST` / `S-NSSAI_SD`	`pduSessionInformation.sNSSAI`	默认值: SST `1`, S
`5QI`	`qoSInformation.5qI`（或 `fiveQI`）	默认 `9`。
`RATType`	`pduSessionInformation.ratType`	默认 `"NR"`。
`PDUSessionID`	`pduSessionInformation.pduSessionID`	默认 `0`。
`PDUSessionType`	`pduSessionInformation.pduType`	默认 `"IPV4"`。
`ChargingOperation`	操作	`Initial` / `Up`



关键程序

融合充电会话（**INITIAL → UPDATE → TERMINATE**）

You have looked at 39 of
network functions. Answe
questions and we send y
estimate.
Get an estima

> 如果 CGRateS `TerminateSession` 在释放时失败，OmniCHF 仍然完成并写入 CDR，因此离线记录不会丢失。消费者随后收到 `500`（原因 `SYSTEM_FAILURE`），而不是虚假的 `204`，因为 CGRateS 端的会话可能仍然处于打开状态，账单后端可能需要进行对账。在成功终止时，消费者收到 `204 No Content`。

额管理（**RSU / GSU** 和触发器）

根据 TS 32.291，消费者发送请求的服务单元（RSU），OmniCHF 返回授予的服务单元（GSU）以及 触发器，指示消费者何时回来：

当达到量阈值或时间限制时，消费者会重新授权（发送更新）。OmniCHF 还可以通过 `/notify` 端点（`REAUTHORIZATION` / `ABORT_CHARGING`）主动驱动重新授权或拆除，该端点向 INITIAL 请求中提供的 `notifyUri` 发送 POST。

充电通知（**REAUTHORIZATION / ABORT_CHARGING**）

根据 TS 32.291 条款 6.1.6.2，OmniCHF 将 `ChargingNotifyRequest` 推送到消费者存储的 `notifyUri`。触发器是面向运营商的 `POST /nchf-convergedcharging/v3/notify` 端点，其主体携带目标 `chargingDataRef` 和 `notificationType`。OmniCHF 查找会话，解析其 `notifyUri`，并将通知 POST 到消费者。

> 通知是面向运营商界面的尽力而为：被拒绝或无法访问的 `notifyUri` 会被记录，但在会话和 `notifyUri` 被解析后不会改变 `/notify` 的结果。没有存储的 `notifyUri` 的会话会产生内部 `no_notify_uri` 错误。

CDR 字段

在释放时构建 CDR，以 INFO 级别记录（`CHF CDR: %{...}`），并且当 `offline_charging_enabled` 为 `true` 时，作为一行 JSON 附加到 `cdr_YYYYMMDD.json`：

字段	来源
`record_type`	静态 `"5G_PDU_SESSION"`
`supi`	会话上下文
`dnn`	会话上下文 / `pduSessionInformation`
`snssai`	`{sst, sd}`
`qos_5qi`	`qoSInformation.5qI`（默认 `9`）
`rat_type`	`pduSessionInformation.ratType`（默认 `"NR"`）
`pdu_session_id` / `pdu_session_type`	会话上下文 / 请求
`volume_uplink` / `volume_downlink` / `volume_total`	从所有 `usedUnitContainer` 中累计
`duration`	报告的 `time`，否则为实际时间差
`start_time` / `end_time`	会话创建/释放时间戳
`charging_data_ref`	会话 UUID
`charging_id`	3GPP 32 位充电 ID（TS 32.251）
`recording_entity`	OmniCHF SBI 基础 URL
`rating_group`	从 DNN 映射解析

离线 CDR 输出（`offline_charging_enabled`、`cdr_output_dir`）和充电 ID 组成（`node_id`）在 配置参考 中进行了配置。

运行离线充电

要捕获 CDR 到文件以进行调解/计费：

1. 设置 `offline_charging_enabled: true` 并将 `cdr_output_dir` 指向一个可写目录。
2. CDR 每行累积一个 JSON 对象到按天命名的文件 `cdr_YYYYMMDD.json`（日期为 UTC 释放日期）。每个 UTC 日都会打开一个新文件；没有自动轮换或保留，除了每日文件名拆分，因此请自行管理旧文件。
3. 每个释放的会话附加正好一个 CDR。进行中的会话在释放之前将其使用情况保存在工作者/上下文中；在此之前不会写入文件。

如果 `offline_charging_enabled` 为 `false`，CDR 仍然会构建并以 INFO 级别记录为 `CHF CDR: %{...}`；只有文件写入被跳过。要捕获没有文件输出的 CDR，请通过您的日志管道发送这些日志行。请参见 CDRs missing from offline files。

定位和搜索 CDR

管理 API 提供 `GET /api/oam/cdr` 来搜索已写入的 CDR 文件。它读取 `cdr_output_dir` 下的每个 `cdr_*.json` 文件，并根据提供的查询参数进行过滤（均为可选，与 AND 组合）：

查询参数	匹□	格式
`supi`	精确 CDR `supi`	例如 `imsi-999700000000001`
`dnn`	精确 CDR `dnn`	例如 `internet`
`date_from`	CDR 文件日期 `>=` 此日期	`YYYY-MM-DD`（可选连字符）
`date_to`	CDR 文件日期 `<=` 此日期	`YYYY-MM-DD`（可选连字符）

日期过滤器应用于 文件名的日期，而不是单个记录时间戳，因此一天的整个文件作为一个单位被包含或排除。没有参数时，该端点返回磁盘上的每个 CDR。因为它读取文件，`GET /api/oam/cdr` 仅在释放时 `offline_charging_enabled` 为开启时返回记录。

强制刷新 CDR（OAM）

在正常操作中，CDR 仅在会话释放时生成。`POST /api/oam/cdr/flush` 允许操作员提前为任何活动会话发出 CDR（请参见 会话存储和管理可见性），例如关闭消费者从未释放的会话：

请求体	效果
`{"charging_data_ref": "<ref>"}`	为该会话构建并写入 CDR，然后最终确定它。如果未找到该引用，则返回 `404`。
`{}` 或 `{"charging_data_ref": "all"}`	刷新当前跟踪的每个会话。

刷新从会话的累计使用情况以及最后的 `ChargingDataRequest` 构建 CDR，并重用与释放相同的 CDR 写入器；它不调用 CGRateS `TerminateSession`。响应报告每个会话的 `write_result` 和 `offline_charging_enabled`。如果离线充电被禁用，则构建 CDR 并最终确定会话，但不会写入文件（`write_result: "ok"`，没有文件输出）。每次刷新都会递增 `omni_chf_charging_releases_total{result="oam_flush"}`。

← 概述

OmniCHF 故障排除

任何充电请求上的 400 Bad Request

症状：消费者收到 `400`，原因是 `MANDATORY_IE_MISSING`。

可能的原因：

- 请求体缺少 `nfConsumerIdentification`、`invocationTimeStamp` 或 `invocationSequenceNumber`。

解决方案：

1. 确认消费者在每个操作（创建、更新、释放）中包含所有三个必需的 IE。
2. 检查 ProblemDetails 响应的 `detail` 字段；它列出了缺失的 IE 名称。

更新或释放时的 404

症状：消费者收到 `404`，原因是 `CHARGING_DATA_NOT_FOUND`。

可能的原因：

- OmniCHF 在创建和后续操作之间重启；会话状态保存在内存中，因此必须在重启后重新创建会话。
- 路径中的 `chargingDataRef` 错误。
- 会话已经被释放（其工作线程已停止）。

解决方案：

1. 在日志中查找 `CHF Update: unknown ref=` / `CHF Release: unknown ref=`。
2. 确认自会话创建以来 OmniCHF 没有重启。
3. 确认消费者回显来自创建的 `Location` 头中的 `chargingDataRef`。

启用 CGRateS 时的 500 错误

症状：消费者收到 `500`，原因是 `CHARGING_FAILED`。

可能的原因：

- CGRateS `InitiateSession` 失败：引擎不可达、租户错误或 CGRateS 端错误。

解决方案：

1. 验证 `cgrates_url` 指向可访问的 CGRateS `/jsonrpc` 端点。
2. 验证 `cgrates_tenant` 与 CGRateS 租户匹配。
3. 检查 `omni_chf_cgrates_health` 指标（`1` = 正常）。
4. 查看日志中 `CGRateS InitiateSession failed ...`；原因可能是 `{:cgrates_error, msg}`、`{:http_error, status}` 或 `{:http_error, reason}`。
5. 确认 CGRateS 中已启用 `SessionSv1` 服务。

释放返回 500

症状：消费者收到 `500`，原因是 `SYSTEM_FAILURE`，标题为 "Charging Termination Failed" 在 `POST /chargingdata/{ref}/release`。

可能的原因：

- CGRateS `TerminateSession` 在释放期间失败（引擎不可达、租户错误或 CGRateS 端错误）。

影响：

- CDR 仍然被构建并写入，因此离线记录不会丢失。
- CGRateS 端的会话可能保持打开状态，因此计费后端可能不一致，需要进行对账。

解决方案：

1. 在受影响的 `ref` 上查找日志中的 `CGRateS TerminateSession failed`。
2. 确认 CGRateS 可达性，查看 `omni_chf_cgrates_health` 指标（`1` = 正常）。
3. 针对已写入的 CDR 对任何为该 `ref` 保留的 CGRateS 会话进行对账。
4. 关联 `omni_chf_charging_releases_total{result="failure"}` 以查看终止失败率。

当守护进程运行时 CGRateS 健康检查失败

症状：`omni_chf_cgrates_health` 读取为 `0`，但 CGRateS 正在运行。

可能的原因：

- CGRateS 运行缓慢：健康检查将其超时限制为 `min(cgrates_timeout, 3000)` 毫秒。
- `cgrates_url` 上的方案不匹配（HTTP 与 HTTPS）。

解决方案：

1. 确认 `cgrates_url` 的方案和端口正确。
2. 如果 CGRateS 确实运行缓慢，请调查 CGRateS 负载，而不是提高上限。
3. 查找 `CGRateS unreachable at <url> ...` 警告。

CGRateS 延迟高

症状：充电操作缓慢；`omni_chf_cgrates_request_duration_ms` 升高。

解决方案：

1. 按 `operation` 检查直方图的 p95/p99。
2. 扩展 CGRateS 或 OmniCHF 与 CGRateS 之间的网络路径。
3. 考虑降低 `cgrates_timeout`，以便慢调用快速失败，而不是阻塞会话。

离线文件中缺少 CDR

症状：日志中出现 `CHF CDR:` 行，但未写入 `cdr_YYYYMMDD.json` 文件。

可能的原因：

- `offline_charging_enabled` 为 `false`（默认���）：仅记录 CDR。
- `cdr_output_dir` 不可写。

解决方案：

1. 设置 `offline_charging_enabled: true`。

2. 确保 `cdr_output_dir` 存在且可写；注意 `[CDR] Failed to write CDR to file` 错误。
3. 要捕获没有文件输出的 CDR，请通过您的日志管道发送 `CHF CDR:` 日志行。

活动会话计数未减少

症状：`omni_chf_sessions_active_count` 保持升高。

可能的原因：

- 消费者从未发送释放（会话仅在释放或工作线程崩溃时清除）。
- 释放返回 `404`（见上文），因此消费者认为它们已完成，而 OmniCHF 可能没有。

解决方案：

1. 将活动计数与创建和释放率关联。
2. 调查释放时的 `404`，这通常表示先前重启。

NRF 注册未维护

症状：`omni_chf_nrf_registration_status` 读取为 `0`。

解决方案：

1. 验证 `nrf_uri` 从 OmniCHF 的 `sbi_addr` 可达。
2. 确认 `mcc` / `mnc` 与 NRF PLMN 配置匹配。
3. 确认广告配置中的 `sbi_addr` / `sbi_port` 可被 NRF 访问。
4. 查看启动日志中的 NRF 注册错误。