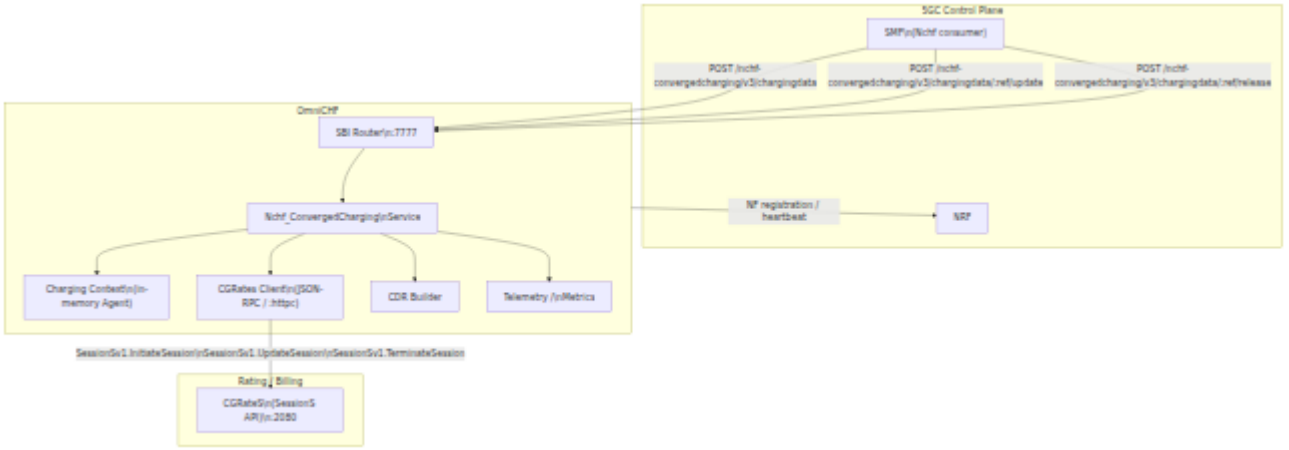
OmniCHF Operations Guide

Table of Contents

Component Overview

OmniCHF implements the Charging Function (CHF) network function defined in 3GPP TS 32.291. The CHF provides converged online and offline charging for 5G PDU sessions via the Nchf_ConvergedCharging service. It translates 5G charging requests into CGRateS SessionS JSON-RPC calls for credit authorization and session management, and generates Call Detail Records (CDRs) on session release.



Charging Session Lifecycle

Each PDU session maps to one charging session, tracked by a `chargingDataRef` (UUID). Session state is held in an in-memory Agent and is not persisted. A restart loses all active session state.

State	Trigger	Storage action
Created	POST /chargingdata	Context created, CGRateS InitiateSession called
Updated	POST /chargingdata/:ref/update	Context updated (usage accumulated, sequence incremented)
Released	POST /chargingdata/:ref/release	CDR built and logged, CGRateS TerminateSession called, context deleted

---

3GPP Role and Spec References

Item	Reference
CHF NF definition	3GPP TS 23.501 Section 6.2.16
Nchf_ConvergedCharging service	3GPP TS 32.291
Charging Data Create procedure	3GPP TS 32.291 Section 6.1.3.2.1
Charging Data Update procedure	3GPP TS 32.291 Section 6.1.3.2.2
Charging Data Release procedure	3GPP TS 32.291 Section 6.1.3.2.3
ChargingDataRequest / Response data model	3GPP TS 32.291 Section 6.1.6
CDR format for 5G PDU sessions	3GPP TS 32.290
SBI common framework	3GPP TS 29.500
NF registration with NRF	3GPP TS 29.510

SBI Endpoints

Base path: `/nchf-convergedcharging/v3`

Method	Path	Description
`POST`	`/chargingdata`	Create a charging session (Initial request). Allocates a `chargingDataRef`, starts a CGRateS session, and returns granted units.
`POST`	`/chargingdata/{chargingDataRef}/update`	Update a charging session (Interim request). Reports current usage and requests additional credit.
`POST`	`/chargingdata/{chargingDataRef}/release`	Release a charging session (Final request). Reports final usage, generates CDR, terminates CGRateS session.

ChargingDataRequest — Key Fields

Field	Type	Used in	Description
`subscriberIdentifier`	string	Create, Update, Release	SUPI (e.g., `imsi-999700000000001`). Used as CGRateS account identifier.
`nfConsumerIdentification`	object	Create	NF consumer info. Fallback source for SUPI if `subscriberIdentifie` is absent.
`pDUSessionChargingInformation`	object	Create, Update, Release	PDU session details: DNN, S-NSSAI, RAT type, QoS, PDU sessio ID and type.
`multipleUnitUsage`	array	Update, Release	Reported usage containers. First element's `usedUnitContainer` is used for volume and duration extraction.
`requestType`	string	All	`INITIAL_REQUEST`, `UPDATE_REQUEST`, or `TERMINATION_REQUEST`

ChargingDataResponse — Key Fields

Field	Type	Present	Description
`invocationSequenceNumber`	integer	Create, Update	Sequence number for this response. Hardcoded to `1` on Create (see CHF-M1). Incremented on each Update.
`invocationResult`	object	Create, Update	Always `{"resultCode": "SUCCESS"}` on the happy path.
`sessionId`	string	Create, Update	The `chargingDataRef` (UUID) allocated for this session.
`multipleUnitInformation`	array	Create, Update	Granted units. Contains one entry with `resultCode`, `grantedUnit` (totalVolume, time), and `ratingGroup` (hardcoded to `1`, see CHF-L2).

Configuration Reference

All parameters are set via the application environment (typically `config/runtime.exs`).

```
config :omnichf,
  sbi_scheme:       "http",
  sbi_addr:         "127.0.0.14",
  sbi_port:         7777,
  nrf_uri:          "http://127.0.0.10:7777",
  mcc:              "999",
  mnc:              "70",
  heartbeat_interval: 10_000,
  cgrates_enabled:  false,
  cgrates_url:      "http://localhost:2080/jsonrpc",
  cgrates_tenant:   "cgrates.org",
  cgrates_timeout:  5000
```

Parameter Table

Parameter	Default	Type	
`sbi_scheme`	`"http"`	string	Tran the
`sbi_addr`	`"127.0.0.14"`	string	IP ad HTT
`sbi_port`	`7777`	integer	TCP serv
`nrf_uri`	`"http://127.0.0.10:7777"`	string	Base used regi hea
`mcc`	`"999"`	string	Mob Use subr and netv
`mnc`	`"70"`	string	Mob Use subr and netv
`heartbeat_interval`	`10_000`	integer (ms)	Inte hea
`cgrates_enabled`	`false`	boolean	Mas CGR Whe CGR

Parameter	Default	Type	
			skip gran is re `true` whe insta
`cgrates_url`	`"http://localhost:2080/jsonrpc"`	string	JSON URL insta whe `cgra` `true`
`cgrates_tenant`	`"cgrates.org"`	string	CGR Pass API `Tena` mat conf CGR
`cgrates_timeout`	`5000`	integer (ms)	HTT for C calls calls `min` `3000` bloc end

CGRateS Integration Notes

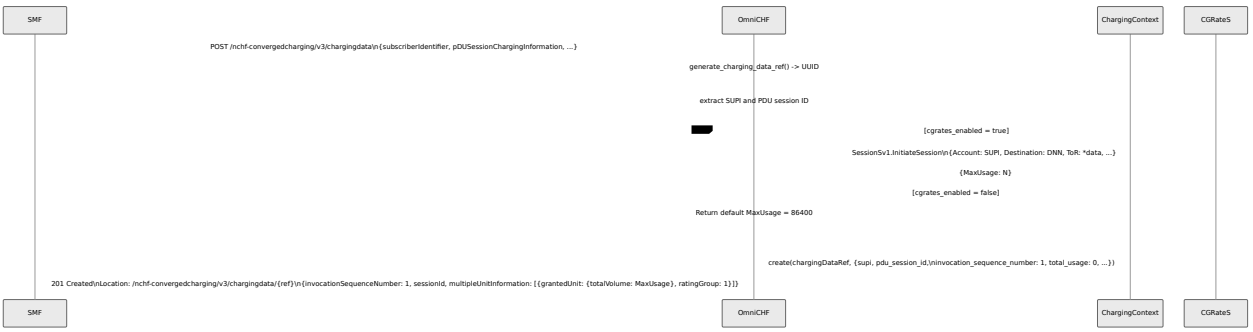
When `cgrates_enabled` is `false`, OmniCHF operates in **bypass mode**: all charging requests are accepted and granted 86,400 units (time or volume)

without any rating or authorization. This is suitable for lab and integration testing when CGRateS is not available.

CGRateS communication uses Erlang's built-in `:httpc` HTTP client (see limitation CHF-M5). This client does not support connection pooling. Under high load, each CGRateS request opens and closes a new HTTP connection, which may become a bottleneck.

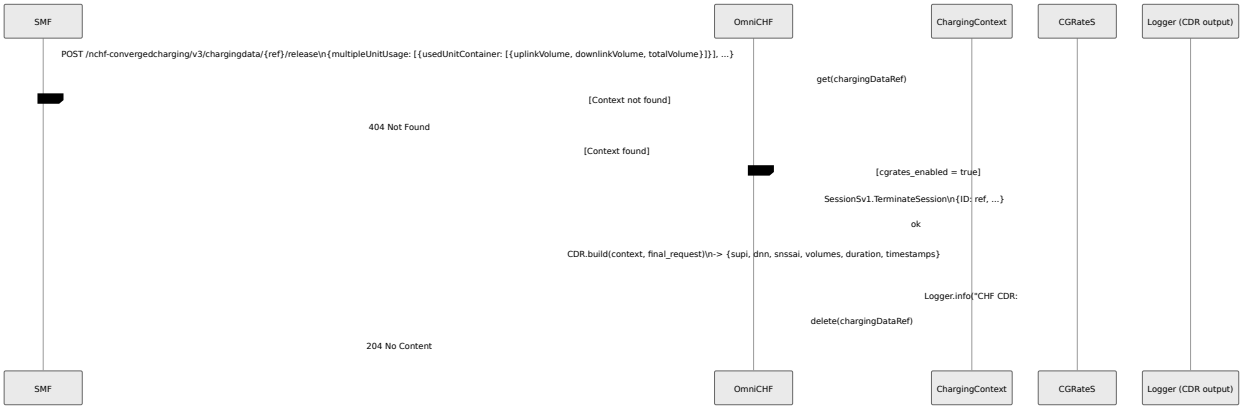
Key Procedures

Charging Session Create (Initial)



Charging Session Update (Interim)

Charging Session Release (Final)



CGRateS Event Mapping

OmniCHF maps 5G charging fields to CGRateS SessionS event fields as follows:

CGRateS Field	Source	
`Account`	`subscriberIdentifier` (SUPI)	Falls back `nfConsum`
`Subject`	`subscriberIdentifier` (SUPI)	Same as A
`Destination`	`pduSessionInformation.dnnId` or `.dnn`	Data Netv
`ToR`	`pduSessionInformation.pduType`	Always `*d`
`RequestType`	`requestType`	Always m
`Usage`	`usedUnitContainer.totalVolume` or sum of uplink+downlink or `time`	First non-z
`OriginID`	`chargingDataRef`	Unique UI
`OriginHost`	static `"OmniCHF"`	
`SUPI`	`subscriberIdentifier`	5G extens
`DNN`	`pduSessionInformation.dnnId`	5G extens
`S-NSSAI_SST`	`pduSessionInformation.sNSSAI.sst`	5G extens
`S-NSSAI_SD`	`pduSessionInformation.sNSSAI.sd`	5G extens
`5QI`	`pduSessionInformation.qoSInformation.5qI`	5G extens
`RATType`	`pduSessionInformation.ratType`	Default: `'`
`PDUSessionID`	`pduSessionInformation.pduSessionID`	5G extens
`PDUSessionType`	`pduSessionInformation.pduType`	Default: `'`

CDR Fields

CDRs are built on session release and emitted to the application log at INFO level. The CDR map contains:

Field	Source
`record_type`	Static: `"5G_PDU_SESSION"`
`supi`	Charging context
`dnn`	`pduSessionInformation.dnnId` or `.dnn`
`snssai`	`{sst, sd}` from `pduSessionInformation.sNSSAI`
`qos_5qi`	`pduSessionInformation.qoSInformation.5qI`
`rat_type`	`pduSessionInformation.ratType`
`pdu_session_id`	Charging context
`pdu_session_type`	`pduSessionInformation.pduType`
`volume_uplink`	`usedUnitContainer.uplinkVolume`
`volume_downlink`	`usedUnitContainer.downlinkVolume`
`volume_total`	`usedUnitContainer.totalVolume` (or uplink+downlink)
`duration`	`usedUnitContainer.time` (or wall clock diff if zero)
`start_time`	Session `created_at` timestamp
`end_time`	Wall clock at release time
`charging_data_ref`	Session UUID

Observability

Telemetry Events

Event	Measurements	Tags	Description
`[:omnichf, :charging, :initial]`	`count`	`supi`	Fired on every Create request
`[:omnichf, :charging, :update]`	`count`	`ref`	Fired on every Update request
`[:omnichf, :charging, :release]`	`count`	`ref`	Fired on every Release request
`[:omnichf, :charging, :creates]`	`count`	`result` (`success`/`failure`)	Create operation outcome
`[:omnichf, :charging, :updates]`	`count`	`result`	Update operation outcome
`[:omnichf, :charging, :releases]`	`count`	`result`	Release operation outcome
`[:omnichf, :cgrates, :request]`	`count`, `duration_ms`	`operation`, `result`	Per CGRateS JSON-RPC call

Event	Measurements	Tags	Description
`[:omnichf, :cgrates, :health]`	`status` (1/0)	—	CGRateS connectivity health
`[:omnichf, :sessions, :active]`	`count`	—	Gauge: active charging sessions
`[:omni5g, :nrf, :registration]`	`status` (1/0)	`nf_type`	NRF registration status

Prometheus Metrics

Charging Metrics

Metric	Type	Tags	Description
`omni_chf.charging.initial.count`	counter	`supi`	Charging session creates
`omni_chf.charging.update.count`	counter	`ref`	Charging session updates
`omni_chf.charging.release.count`	counter	`ref`	Charging session releases
`omni_chf.charging.creates.total`	counter	`result`	Total charging session creates
`omni_chf.charging.updates.total`	counter	`result`	Total charging session updates
`omni_chf.charging.releases.total`	counter	`result`	Total charging session releases
`omni_chf.sessions.active.count`	gauge	--	Number of active charging sessions

CGRateS Metrics

Metric	Type	Tags	Des
`omni_chf.cgrates.calls.count`	counter	`method`, `result`	CGR JSO call
`omni_chf.cgrates.latency.milliseconds`	gauge	--	CGR late
`omni_chf.cgrates.health`	gauge	--	CGR con hea (1= 0=c
`omni_chf.cgrates.requests.total`	counter	`operation`, `result`	Tota CGR JSO req
`omni_chf.cgrates.request.duration_ms`	distribution	`operation`	CGR req dur ms 5, 1 50, 250 100

NRF Metrics

Metric	Type	Tags	Description
`omni_chf.nrf.registration.status`	gauge	`nf_type`	NRF registration status (1=registered, 0=not)

BEAM VM Metrics

Metric	Type	Description
`beam.memory.total`	gauge	Total BEAM memory in bytes
`beam.memory.processes`	gauge	Memory used by Erlang processes
`beam.memory.processes_used`	gauge	Memory actually used by processes
`beam.memory.system`	gauge	System memory
`beam.memory.atom`	gauge	Total atom memory
`beam.memory.atom_used`	gauge	Used atom memory
`beam.memory.binary`	gauge	Binary memory
`beam.memory.code`	gauge	Code memory
`beam.memory.ets`	gauge	ETS table memory
`beam.processes.count`	gauge	Number of Erlang processes
`beam.ports.count`	gauge	Number of Erlang ports
`beam.atom.count`	gauge	Number of atoms
`beam.vm.uptime`	gauge	VM uptime in seconds

Log Patterns

Level	Pattern	Meaning
`info`	`CHF Create: ref=<UUID> supi=<SUPI> pdu_session=<N>`	Successful Create initiated
`info`	`CHF Update: ref=<UUID>`	Update request received
`info`	`CHF Release: ref=<UUID>`	Release request received
`info`	`CHF CDR: %{...}`	CDR emitted on release
`info`	`Initiating CGRateS session for <ref>, account: <SUPI>`	CGRateS InitiateSession sent
`info`	`CGRateS authorized <N> units for session <ref>`	Credit granted
`info`	`CGRateS session <ref> terminated successfully`	CGRateS TerminateSession OK
`warning`	`CGRateS integration disabled, returning default authorization`	Bypass mode active
`warning`	`CHF Update: unknown ref=<UUID>`	Update for non-existent session
`warning`	`CHF Release: unknown ref=<UUID>`	Release for non-existent session
`error`	`CHF Create failed: <reason>`	Create operation failed

Level	Pattern	Meaning
`error`	`CHF Update failed: <reason>`	Update operation failed
`error`	`CGRateS InitiateSession failed for <ref>: <reason>`	CGRateS error on create
`error`	`CGRateS HTTP error <status>: <body>`	Non-200 from CGRateS
`error`	`CGRateS HTTP request failed: <reason>`	Network error to CGRateS

Known Limitations

ID	Severity	Description
CHF-M1	Medium	`invocationSequenceNumber` is hardcoded to `1` in the Create (Initial) response. Per TS 32.291 the sequence number should start at 1 and increment on subsequent responses, which it does on Update. The issue is that if a consumer sends a Create with a `invocationSequenceNumber` in the request, that value is not inspected or validated.
CHF-M3	Medium	`invocationTimeStamp` is absent from `ChargingDataResponse`. Per TS 32.291 this field is mandatory in the response body. Strict consumers that require this field will receive an incomplete response.
CHF-M5	Medium	The CGRateS client uses Erlang's `:httpc` HTTP client rather than Finch. `:httpc` does not support connection pooling; each JSON-RPC call opens and closes a TCP connection. Under load (many concurrent charging sessions), CGRateS call latency will increase and the connection setup overhead becomes significant. Monitor `omni_chf.cgrates.request.duration_ms`.
CHF-L1	Low	No `triggers` field is included in `ChargingDataResponse`. Per TS 32.291, triggers can instruct the SMF to send an interim update on specific events (volume threshold, time threshold, QoS change). Without triggers, the SMF uses its own local policy to determine when to send updates.
CHF-L2	Low	`ratingGroup` in `multipleUnitInformation` is hardcoded to `1`. Real deployments typically have multiple rating groups per PDU session (one per service data flow). All usage is attributed to rating group 1

ID	Severity	Description
		regardless of the `ratingGroup` values in the request's `multipleUnitUsage`.
CHF-L3	Low	No `chargingId` in the 3GPP format is generated. Per TS 32.290, charging records should carry a chargingId that correlates with the PDU session ID assigned by the SMF. The `charging_data_ref` UUID is used instead, which may cause correlation issues in downstream billing systems that expect the 3GPP chargingId format.
CHF-L4	Low	The CDR record is missing the `chargingID` and `recordingEntity` fields required by TS 32.290. Downstream mediation or billing systems expecting these fields will need to tolerate their absence or be configured to treat them as optional.
CHF-L5	Low	Offline charging and CDR file output are not implemented. CDRs are emitted only to the application log via `Logger.info`. There is no file-based CDR output, no ASN.1 encoding, and no transfer to a billing domain gateway. For production offline charging, a log shipper (e.g., Fluentd, Vector) must collect the CHF CDR log lines and transform them for the billing system.

Troubleshooting

404 on Update or Release

The `chargingDataRef` does not match any active session in memory. Causes:

1. OmniCHF restarted between Create and Update/Release — all session state is in-memory and is lost on restart.
2. The SMF sent the wrong `chargingDataRef` in the path.

3. A Release was previously sent for this session, which deleted the context.

Check logs for `CHF Update: unknown ref=` or `CHF Release: unknown ref=` to confirm.

500 on Create with CGRateS enabled

The CGRateS call failed. Check:

1. Is `cgrates_url` pointing to a reachable CGRateS instance?
2. Is `cgrates_tenant` correct? Mismatched tenant names cause CGRateS to return an error response.
3. Check `omni_chf.cgrates.health` metric (1=up, 0=down).
4. Review logs for `CGRateS InitiateSession failed for <ref>: <reason>` — the reason will be one of: `{:cgrates_error, message}`, `{:http_error, status}`, or `{:http_error, reason}`.
5. Verify the CGRateS `SessionSv1` service is enabled in the CGRateS configuration.

CGRateS health check failing but daemon is running

The health check (via `SessionSv1.GetActiveSessions`) uses a 3-second capped timeout. If CGRateS is slow to respond, the health check may fail while the service is technically available. Check `cgrates_timeout` — the cap is `min(cgrates_timeout, 3000)`. Also confirm `cgrates_url` uses HTTP (not HTTPS) unless TLS is configured.

CDRs not appearing or incomplete

CDRs are written to the application log at INFO level (see limitation CHF-L5). To capture CDRs:

1. Ensure the application log level is set to `info` or lower.
2. Filter log lines containing `"CHF CDR:"` for post-processing.
3. Note that CDRs are missing `chargingID`, `recordingEntity` (CHF-L4), and will have `ratingGroup: 1` for all service data flows (CHF-L2).

High CGRateS call latency

Because CGRateS uses `:httpc` without connection pooling (CHF-M5), latency increases under load. To diagnose:

1. Check `omni_chf.cgrates.request.duration_ms` histogram for p99 latency.
2. If latency is high under concurrent load, reduce the number of concurrent charging sessions or consider deploying OmniCHF behind a load balancer with multiple instances.
3. As a workaround, set `cgrates_timeout` to a value lower than the expected worst-case CGRateS response time to prevent slow CGRateS calls from blocking the Elixir process pool.

Active sessions count not decreasing after releases

If `omni_chf.sessions.active.count` remains elevated after sessions should have been released:

1. Check for 404 responses on Release calls — if the SMF receives 404, it may not retry and the SMF considers the session released while OmniCHF may still have the context.
2. In the inverse case, if OmniCHF restarted and lost session state, contexts are gone but the SMF may still send Update/Release requests that result in 404. This is expected behaviour.

NRF registration not maintained

Check `omni_chf.nrf.registration.status` metric. If it reads 0:

1. Verify `nrf_uri` is correct and the NRF is reachable from OmniCHF's `sbi_addr`.
2. Check `mcc` and `mnc` match the NRF PLMN configuration.
3. Review application logs at startup for NRF registration errors.