← Overview

OmniNRF Configuration Reference

OmniNRF reads its operator configuration from `config/runtime.exs`, which maps environment variables onto the `:omninrf` application environment. Every operator-facing key is listed below with its environment variable and default.

Runtime Configuration

```
import Config

config :omninrf,
  sbi_addr: System.get_env("SBI_ADDR", "127.0.0.1"),
  sbi_port: String.to_integer(System.get_env("SBI_PORT", "7777")),
  mcc: System.get_env("MCC", "999"),
  mnc: System.get_env("MNC", "70"),
  heartbeat_timeout:
String.to_integer(System.get_env("HEARTBEAT_TIMEOUT", "30000")),
  prometheus_metrics_port:
String.to_integer(System.get_env("PROMETHEUS_PORT", "9568"))

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

Parameter Table

Parameter	Type	Required	Default
`sbi_scheme`	string	No	`http`
`sbi_addr`	string	No	`127.0.0.1`
`sbi_port`	integer	No	`7777`

Parameter	Type	Required	Default
`mcc`	string	No	`999`
`mnc`	string	No	`70`
`heartbeat_timeout`	integer (ms)	No	`30000`
`cleanup_interval`	integer (ms)	No	`10000`

Parameter	Type	Required	Default
`notification_max_attempts`	integer	No	`4`
`notification_backoff_ms`	integer (ms)	No	`500`
`discovery_validity_period`	integer (s)	No	`3600`

Parameter	Type	Required	Default
`plmn_list`	list	No	`[%{mcc: mcc, mnc: r`
`prometheus_metrics_port`	integer	No	`9568`
`logger format`	tuple	No	`{OmniLogger.JsonFo :format}`
`logger metadata`	atom/list	No	`:all`

> `SBI_ADDR` defaults to `127.0.0.1` (loopback). In any multi-host or containerised deployment this must be overridden to an address the other NFs can reach, otherwise no NF will be able to register or discover.

Heartbeat aging semantics

The NRF **receives** heartbeats from other NFs; it does not send them. Each time an NF sends a PATCH heartbeat (or any registration/update) for its instance, the NRF records a fresh "last seen" timestamp against that NF profile.

A background sweep runs every `cleanup_interval` milliseconds and removes any NF whose last-seen timestamp is older than `heartbeat_timeout` milliseconds. When an NF is aged out this way it is deregistered from the store and a `NF_DEREGISTERED` status notification is sent to any matching subscribers, so consumers stop being handed a dead endpoint.

On registration, if an NF does not include its own `heartBeatTimer`, the NRF fills one in equal to `heartbeat_timeout / 1000` seconds as a hint of how often it expects to be heartbeated. Operators should ensure NFs heartbeat well inside `heartbeat_timeout` (a value comfortably shorter than the timeout) so transient jitter does not cause a live NF to be expired.

The background sweep interval (`cleanup_interval`) and the staleness threshold (`heartbeat_timeout`) are independent. An NF is only removed on the first sweep that runs *after* its last-seen age has already crossed `heartbeat_timeout`, so the real worst-case time an NF lingers after going silent is up to `heartbeat_timeout + cleanup_interval`. Keep `cleanup_interval` well below `heartbeat_timeout` (the defaults are 10 s and 30 s) so this overshoot stays small; a `cleanup_interval` larger than `heartbeat_timeout` would leave dead NFs advertised for longer than intended.

plmn_list

`plmn_list` is the set of PLMNs **this NRF serves**, expressed as a list of `%{mcc, mnc}` maps. It is distinct from the `plmnList` carried inside each registered NF's own profile. That per-NF list is what discovery filters against, whereas `plmn_list` describes the NRF's own served network(s).

Sub-field	Type	Description
`mcc`	string	Mobile Country Code (3 digits)
`mnc`	string	Mobile Network Code (2 to 3 digits)

If unset, `plmn_list` defaults to a single entry built from the top-level `mcc`/`mnc` keys. Configure multiple entries for a shared / multi-operator NRF, for example:

```
config :omninrf,
  plmn_list: [%{mcc: "999", mnc: "70"}, %{mcc: "001", mnc: "01"}]
```

Discovery matching (the `plmn-id` query parameter) is evaluated against each NF's own `plmnList`, not against `plmn_list`. Per TS 29.510, a registered NF that advertises **no** `plmnList` is assumed to belong to the NRF's PLMN and therefore matches any `plmn-id` filter.

Management / OAM API

Alongside the 5G SBI, OmniNRF exposes an operator-facing REST management API for runtime inspection and control. It is served over **HTTPS** on the api_ex management listener (default port `8443`). Request and response bodies are JSON.

Method	Path	Description
GET	`/api/oam/config`	View the current runtime configuration (`sbi_addr`, `sbi_port`, `heartbeat_timeout`, `cleanup_interval`, `plmn_list`, `log_level`)
PATCH	`/api/oam/config/:id`	Update runtime configuration. Only `heartbeat_timeout`, `cleanup_interval`, and `log_level` may be changed; all supplied values are validated before any are applied (400 on an invalid value)
POST	`/api/oam/log_level`	Change the log level at runtime. Body `{"level": "..."}`; valid levels are `emergency`, `alert`, `critical`, `error`, `warning`, `warn`, `notice`, `info`, `debug` (400 otherwise)
POST	`/api/oam/heartbeat`	Manually refresh the heartbeat timestamp for an NF. Body `{"nf_instance_id": "..."}` (404 if the instance is not registered)
GET	`/api/oam/nf_uptime/:id`	Show registration/heartbeat age and status for one NF instance (404 if not registered)
POST	`/api/oam/nf_purge`	Trigger an immediate stale-instance purge sweep of the registry; reports how many NF instances were removed

Method	Path	Description
GET	`/api/oam/resources`	Show BEAM VM resource usage (memory, process/port counts, scheduler count, uptime, OTP/BEAM versions)
GET	`/api/oam/diagnostics`	Show BEAM VM diagnostics (node, uptime, connected nodes, process count, run queue, I/O and GC statistics)
PATCH	`/api/oam/nf_status/:id`	Update the stored status of an NF instance

The same listener also serves read-only status and inventory endpoints: `/api/status/api`, `/api/status/license`, `/api/status/nrf`, `/api/status/nf`, `/api/api/nf_instances`, `/api/api/subscriptions`, and `/api/api/statistics`.

Inspecting and purging NF instances

`GET /api/oam/nf_uptime/:id` is the direct way to see how close an NF is to being aged out. It reports:

Field	Meaning
`last_heartbeat_ms_ago`	Age of the NF's last-seen timestamp, in milliseconds
`heartbeat_timeout`	The current staleness threshold, in milliseconds
`nf_type`, `nf_status`	The NF's type and stored status

An NF is a candidate for removal once `last_heartbeat_ms_ago` exceeds `heartbeat_timeout`; the gap between the two is roughly how much longer the

NF may stay silent before the sweep expires it.

`POST /api/oam/nf_purge` runs the same stale-instance sweep as the background timer, immediately and on demand, and returns how many instances were removed. It only removes NFs that are *already* stale (last-seen older than `heartbeat_timeout`); it does not force out healthy NFs. Use it to reclaim the registry at once rather than waiting for the next scheduled sweep. To forcibly remove a specific live NF regardless of its heartbeat age, deregister it directly with `DELETE /api/api/nf_instances/:id`.

Logging

OmniNRF emits **structured JSON logs** via the `OmniLogger.JsonFormatter` formatter, with all logger metadata included (`metadata: :all`). Each log line carries per-request metadata such as the NF instance ID and the procedure being handled, which makes it straightforward to correlate registration, heartbeat, discovery, and notification events for a given NF.

The active log level can be changed at runtime without a restart, either through `POST /api/oam/log_level` or via the `log_level` key of `PATCH /api/oam/config/:id`.

← Overview

OmniNRF Metrics

Metrics are exposed at `http://{host}:{prometheus_metrics_port}/metrics` (default port `9568`) in Prometheus text format.

NRF Service Metrics

Metric	Type	Description	
`omni_nrf_nf_registered_count`	Gauge	Current number of registered NF instances, per type	`nf_`
`omni_nrf_active_nf_instances_count`	Gauge	Current number of active NF instances, per type. Emitted on every NF register, deregister, and heartbeat-expiry, so it reflects the live active set	`nf_`
`omni_nrf_registration_requests_count`	Counter	Total NF registration requests received (all types)	(nor
`omni_nrf_nf_registrations_total`	Counter	Total NF registrations, per type	`nf_`
`omni_nrf_nf_deregistrations_total`	Counter	Total NF deregistrations, per type. The `nf_type` label	`nf_`

Metric	Type	Description	
		is resolved from the stored NF profile	
`omni_nrf_heartbeats_total`	Counter	Total NF heartbeat requests, per type. The `nf_type` label is resolved from the stored NF profile	`nf_`
`omni_nrf_discovery_requests_count`	Counter	Total discovery requests received (all target types)	(nor
`omni_nrf_discovery_requests_total`	Counter	Total discovery requests, per target NF type	`tar`

BEAM VM Metrics

Metric	Type	Description
`beam_memory_total`	Gauge	Total BEAM memory (bytes)
`beam_memory_processes`	Gauge	Memory allocated to Erlang processes
`beam_memory_processes_used`	Gauge	Memory actually used by processes
`beam_memory_system`	Gauge	System (non-process) 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 (holds the NF and subscription stores)
`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 (seconds)

Useful PromQL

Registered NF instances by type:

```
omni_nrf_nf_registered_count
```

Registration request rate over 5 minutes:

```
rate(omni_nrf_registration_requests_count[5m])
```

Discovery request rate by target NF type:

```
sum by (target_nf_type)
(rate(omni_nrf_discovery_requests_total[5m]))
```

Heartbeat rate by NF type (a drop toward zero warns of NFs that have stopped heartbeating). The `nf_type` label is resolved from the stored profile, so the per-type breakdown is meaningful:

```
sum by (nf_type) (rate(omni_nrf_heartbeats_total[5m]))
```

Net registration churn (registrations minus deregistrations), per NF type. Both counters carry a real `nf_type`, so churn can be computed per type:

```
sum by (nf_type) (rate(omni_nrf_nf_registrations_total[5m]))
  - sum by (nf_type) (rate(omni_nrf_nf_deregistrations_total[5m]))
```

Registrations by NF type (this counter does carry a real `nf_type`):

```
sum by (nf_type) (rate(omni_nrf_nf_registrations_total[5m]))
```

> Metric name rendering: Telemetry names such as `omni_nrf.nf.registered.count` are exported with dots replaced by underscores. Confirm the exact exported names against a live `/metrics` scrape, as they can vary with exporter configuration.

← Overview

OmniNRF Operations Guide

OmniNRF implements the Network Repository Function (NRF) of the 5G Core. The NRF is the central registry and service directory of the Service Based Architecture (SBA): every other Network Function (NF) registers its profile with the NRF at start-up, keeps that registration alive with periodic heartbeats, and queries the NRF to discover the endpoints of the NFs it needs to talk to.

OmniNRF exposes two SBI services:

- **Nnrf_NFManagement** (`nnrf-nfm/v1`): NF profile registration, update, heartbeat, deregistration, retrieval, and NF status-change subscriptions/notifications.
- **Nnrf_NFDiscovery** (`nnrf-disc/v1`): resolution of target NF instances by type, service, S-NSSAI, DNN, and PLMN.

All NF profiles and subscriptions are held in in-memory (ETS) stores. There is no external database; the registry is rebuilt as NFs re-register after an NRF restart. A background sweep continuously ages out NF instances that have stopped sending heartbeats, so the registry only ever advertises NFs that are demonstrably alive.

3GPP Role and Specification References

Specification	Relevance
TS 23.501	System architecture, NRF role in the Service Based Architecture (clause 6.2.6)
TS 29.510	Nnrf_NFManagement and Nnrf_NFDiscovery services, NFProfile / SearchResult / SubscriptionData data models
TS 29.510 §5.2	Nnrf_NFManagement: registration, update, heartbeat (`heartBeatTimer`), deregistration, status subscriptions and notifications
TS 29.510 §5.3	Nnrf_NFDiscovery: NF instance discovery query and SearchResult
TS 29.510 §6.2.3.2.3.1	Discovery query parameters, result ordering and pagination

OmniNRF is the SBI **producer** for `Nnrf_NFManagement` and `Nnrf_NFDiscovery`. It does not itself register with any other NRF; it is the registry that everything else points at, so there is no outward `nrf_uri` in its operator configuration.

SBI Endpoints

The SBI listener is served over **HTTP/2 (cleartext, h2c)** at `http://{sbi_addr}:{sbi_port}`. Request and response bodies are JSON. PATCH bodies use `application/json-patch+json`; the NF instance list is returned as `application/3gppHal+json`.

Nnrf_NFManagement (`nnrf-nfm/v1`)

Method	Path	Description	Suc
PUT	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Register or re-register an NF (full NFProfile body)	201 Creat (new) 200 C (upda
PATCH	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Heartbeat (empty body) or JSON-Patch profile update	204 N Conte (hear / 200 (patc
DELETE	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Deregister an NF	204 N Conte
GET	`/nnrf-nfm/v1/nf-instances/{nfInstanceId}`	Retrieve a single NFProfile	200 C
GET	`/nnrf-nfm/v1/nf-instances`	List NF instances (UriList; supports `nf-type`, `limit`, `page-number`, `page-size`)	200 C
POST	`/nnrf-nfm/v1/subscriptions`	Subscribe to NF status notifications	201 Creat

Method	Path	Description	Suc
GET	`/nnrf-nfm/v1/subscriptions/{subscriptionId}`	Read back a single subscription	200 C
PATCH	`/nnrf-nfm/v1/subscriptions/{subscriptionId}`	Update a subscription (JSON-Patch)	200 C
DELETE	`/nnrf-nfm/v1/subscriptions/{subscriptionId}`	Unsubscribe	204 N Conte

Successful PUT/PATCH/GET responses on a single NF instance carry an `ETag` header. Clients may supply `If-Match` on PUT/PATCH to make the update conditional (412 Precondition Failed on mismatch).

Nnrf_NFDiscovery (`nnrf-disc/v1`)

Method	Path	Description	Success
GET	`/nnrf-disc/v1/nf-instances`	Discover NF instances by type and optional filters	200 OK

When a discovery result is paginated (a `limit` was supplied and more results remain), the response carries a `Link: <...>; rel="next"` header pointing at the next page.

Discovery Query Parameters

Parameter	Required	Description
`target-nf-type`	Yes	NF type being searched for (e.g. `UDM`, `SMF`, `PCF`)
`requester-nf-type`	Yes	NF type making the request (e.g. `AMF`)
`service-names`	No	Comma-separated service names; an NF matches if it offers any listed service
`plmn-id`	No	JSON-encoded PLMN ID, e.g. `{"mcc":"999","mnc":"70"}`. An NF with no `plmnList` matches any PLMN
`snssais`	No	JSON-encoded S-NSSAI (object or array), e.g. `[{"sst":1,"sd":"000001"}]`. Matched against the NF's `sNssais`, `allowedNssais`, and per-service `allowedNssais`
`dnn`	No	Data Network Name; matched against per-service `allowedDnns`
`limit`	No	Maximum number of results in this page
`offset`	No	Starting index into the (sorted) result set, used for pagination

Results are returned in a `SearchResult` (`validityPeriod`, `nfInstances`, `numNfInstComplete`). Only NFs with `nfStatus = REGISTERED` are returned.

Result ordering

Matching NFs are sorted so that the most preferred instance appears first, letting a consumer select `nfInstances[0]` and only fall back to later entries. The sort keys, in order, come from each NF's own registered profile:

Key	Direction	Meaning for an operator
`priority`	ascending (lower first)	Relative preference set on the NF profile. Lower wins; NFs with no `priority` sort last (treated as `65535`)
`capacity`	descending (higher first)	Advertised capacity of the NF. Among equal-priority NFs, the higher-capacity one is preferred
`load`	ascending (lower first)	Current load reported by the NF. Among otherwise-equal NFs, the less-loaded one is preferred

To steer consumers toward a particular producer, set a lower `priority` (or higher `capacity`) on that NF's registered profile; these are properties of the target NF, not of the NRF.

Filter matching rules

- **`service-names`**: an NF matches if it offers *any one* of the listed services.
- **`snssais`**: matched against the union of the NF's `sNssais`, its top-level `allowedNssais`, and each service's `allowedNssais`. An `sd` is only required to match when the query supplies one.
- **`plmn-id`** and **`dnn`**: matched against the NF's `plmnList` and per-service `allowedDnns` respectively.

For every optional filter, an NF that advertises *none* of that attribute is treated as matching any value for it (e.g. an NF with no `plmnList` matches any `plmn-id`). This "unconstrained means universal" rule follows TS 29.510 and keeps minimally-provisioned NFs discoverable.

Pagination

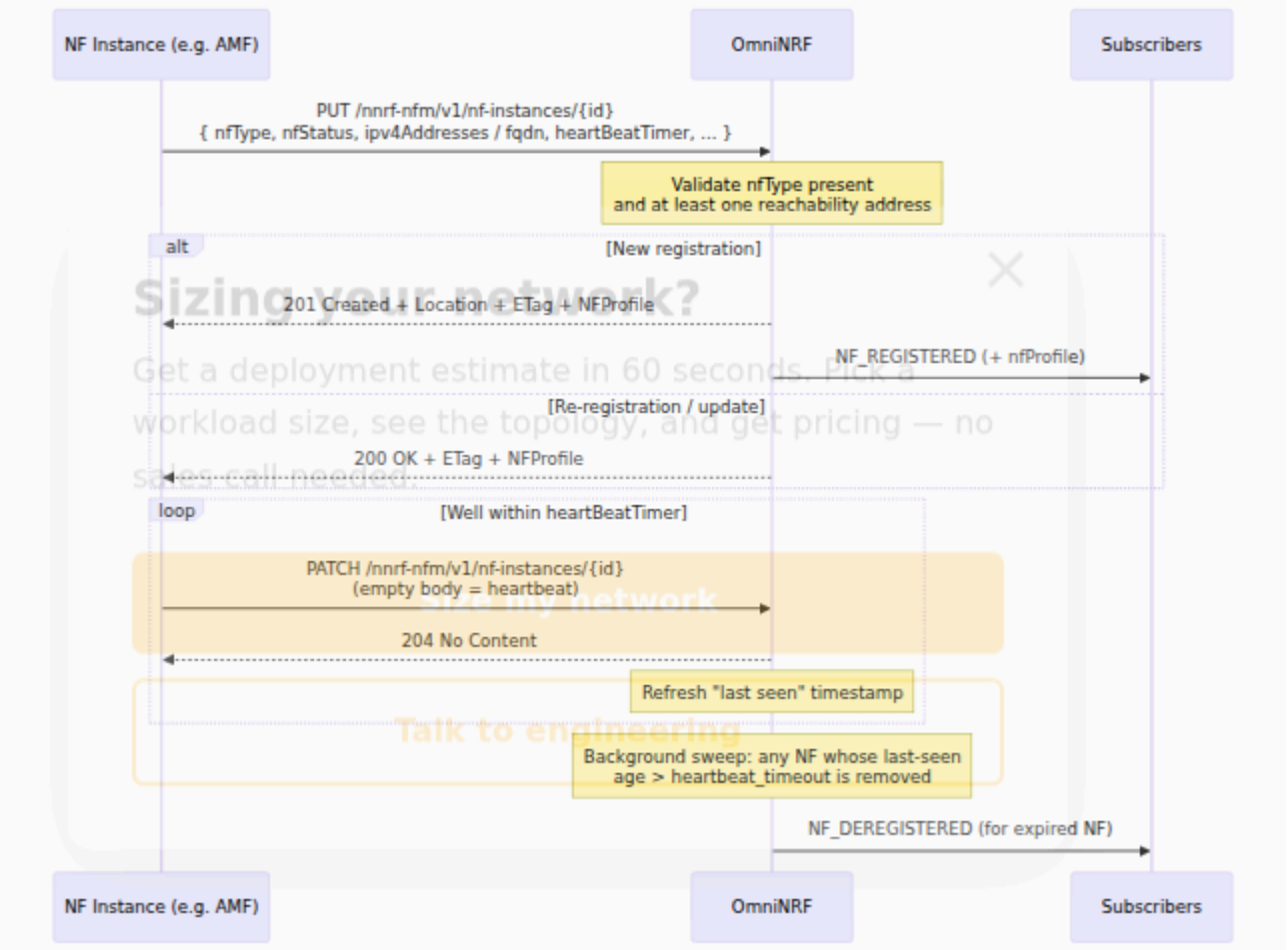
`limit` bounds the number of instances returned in one page; `offset` is the starting index into the sorted result set. When more results remain beyond the page, the response carries a `Link: <...>; rel="next"` header whose `offset` points at the next page. `numNfInstComplete` always reflects the full match

count before pagination, so a consumer can page until `offset + len(nfInstances)` reaches it.

> This SBI discovery pagination (`offset`/`limit` + `Link`) is separate from the NF-instance *list* endpoint's pagination under Nnrf_NFManagement, which instead uses `limit`/`page-number`/`page-size` over a `3gppHal+json` UriList.

Key Procedures

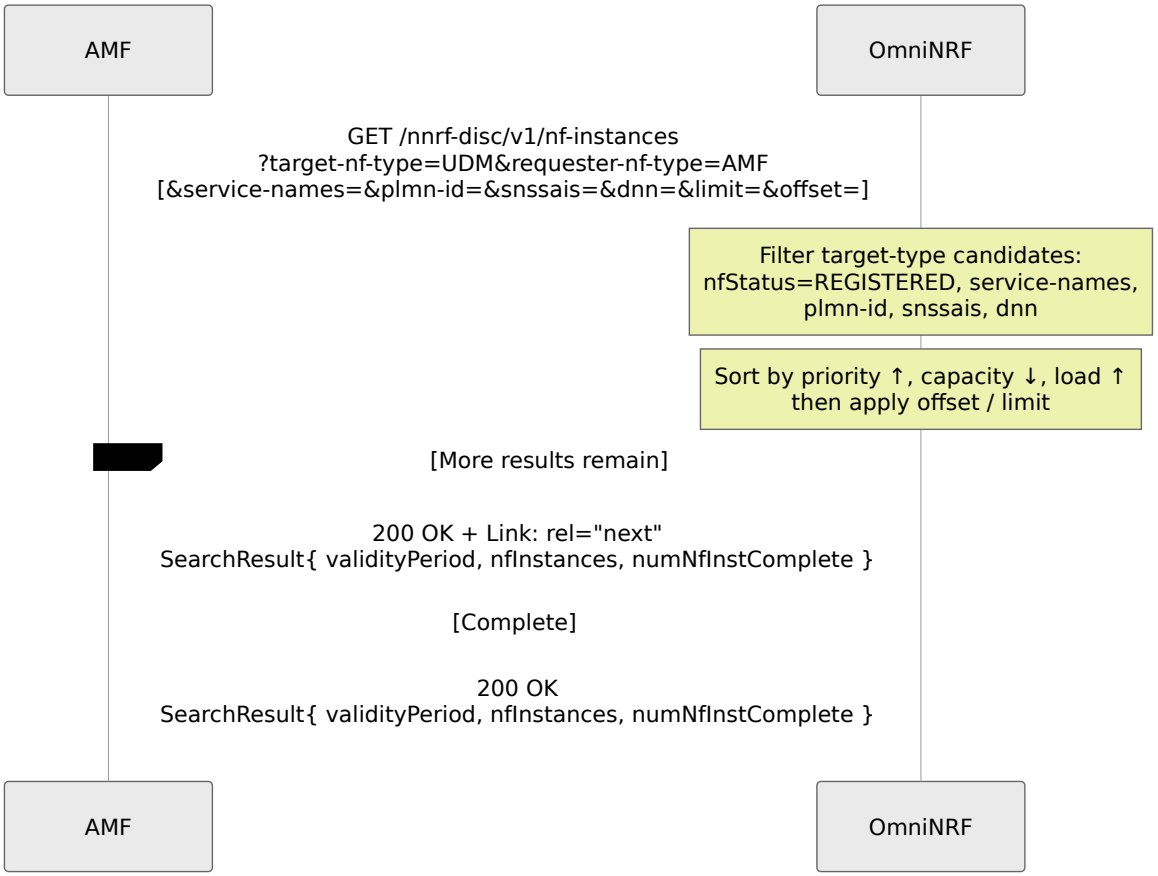
NF Registration and Heartbeat



Registration requires `nfType` and `nfStatus` in the body (a body missing either is rejected with 400 `MANDATORY_IE_MISSING`) **and** at least one reachability address (`fqdn`, `ipv4Addresses`, or `ipv6Addresses`). A body with no reachability address is rejected with 400 (bad request).

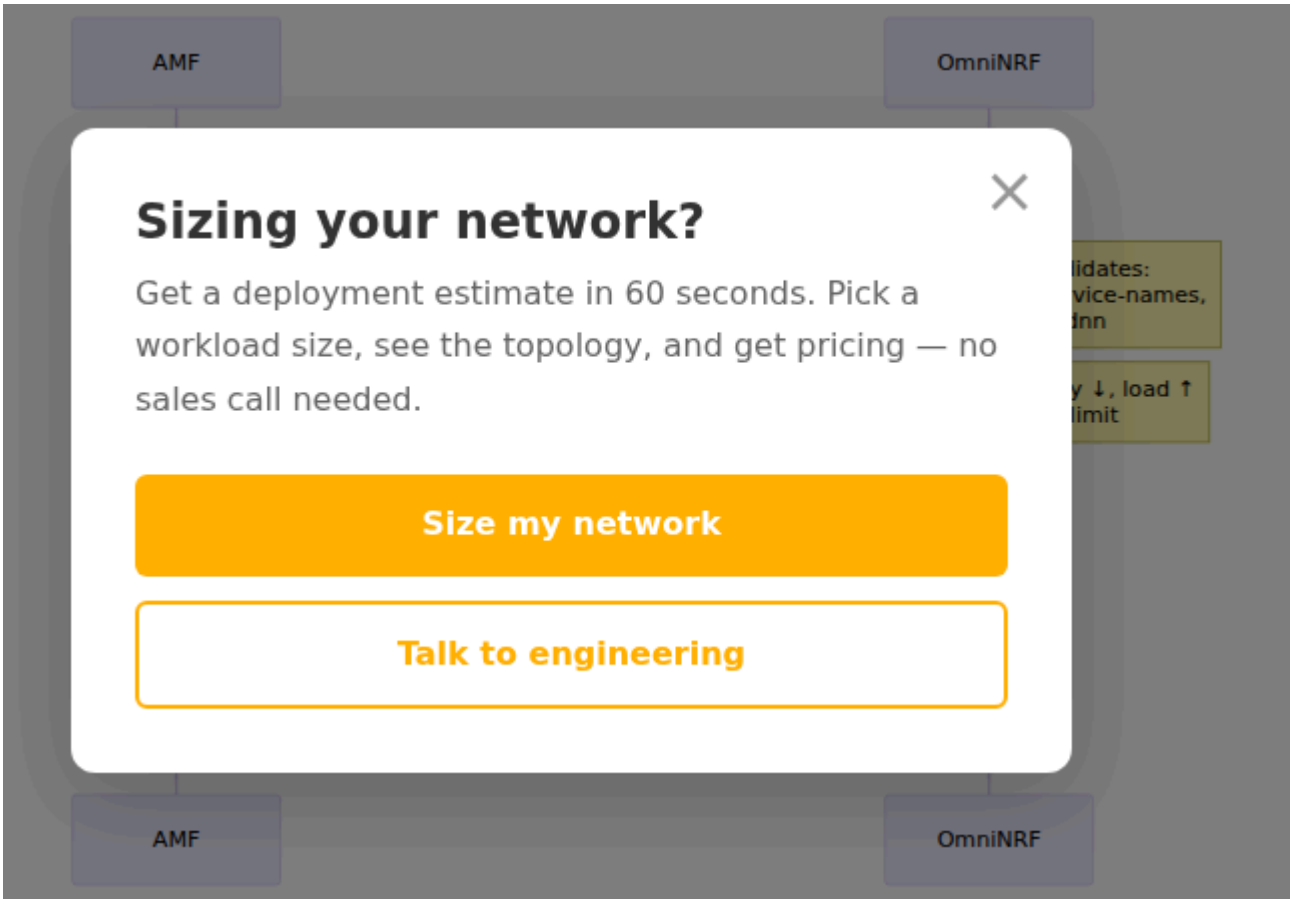
The NRF **receives** heartbeats from other NFs; it does not send them. Each time an NF sends a PATCH heartbeat (or any registration/update) for its instance, the NRF records a fresh "last seen" timestamp against that NF profile. A background sweep runs continuously and removes any NF whose last-seen timestamp is older than `heartbeat_timeout` milliseconds. When an NF is aged out this way it is deregistered from the store and a `NF_DEREGISTERED` status notification is sent to any matching subscribers, so consumers stop being handed a dead endpoint.

NF Discovery



Both `target-nf-type` and `requester-nf-type` are mandatory; omitting either yields 400. An empty `nfInstances` list (with `numNfInstComplete: 0`) is a valid response meaning nothing matched.

NF Status Subscriptions and Notifications



Notifications are delivered over the shared SBI HTTP/2 client (pooled) with a bounded retry: each notification is attempted up to `notification_max_attempts` times (default 4), using exponential backoff with base `notification_backoff_ms` (default 500 ms, doubling each retry). A subscriber that answers non-2xx or is briefly unreachable is retried rather than dropped after a single attempt; after the final failed attempt the NRF gives up and logs a warning. An unreachable subscriber is logged but never crashes the NRF. Event types (TS 29.510 clause 5.2.2.8) are mapped from NF status: `REGISTERED` to `NF_REGISTERED` (the full `nfProfile` is included), `DEREGISTERED`/`SUSPENDED` to `NF_DEREGISTERED`, other status changes to `NF_PROFILE_CHANGED`. A subscription with `subscrCond.nfType` only receives notifications for that NF type. If `validityTime` (ISO 8601) is supplied, the subscription is automatically deleted when it expires.

NF status transitions and how they reach subscribers

Trigger	Resulting notification	Also removed from registry?
NF registers (PUT, new)	`NF_REGISTERED` (with `nfProfile`)	No, added
NF profile changes (PATCH / OAM update to a non-terminal status)	`NF_PROFILE_CHANGED`	No
NF deregisters (DELETE)	`NF_DEREGISTERED`	Yes
NF aged out (missed heartbeats, background sweep)	`NF_DEREGISTERED`	Yes
Manual purge (`POST /api/oam/nf_purge`) of a stale NF	`NF_DEREGISTERED`	Yes

An **aged-out** NF and an explicitly **deregistered** NF are indistinguishable to subscribers: both surface as `NF_DEREGISTERED` and both leave the registry. The difference is purely the cause: deregistration is a deliberate DELETE from the NF itself, whereas ageing-out is the NRF removing an NF that stopped heartbeating. A **suspended** NF (`nfStatus = SUSPENDED`) is reported as `NF_DEREGISTERED` for notification purposes and, because only `REGISTERED` NFs are returned by discovery, is effectively hidden from consumers while its profile remains in the store.

← Overview

OmniNRF Troubleshooting

An NF vanished from discovery even though it is running

Almost always missed heartbeats. If an NF does not send a PATCH heartbeat within `heartbeat_timeout` (default 30 s), the background sweep expires it and it disappears from both retrieval and discovery. Check the NRF logs for `NRF: Expiring stale NF instance {id}`, and confirm the NF is heartbeating well inside the timeout. Raising `HEARTBEAT_TIMEOUT` gives more slack; the NF should also honour the `heartBeatTimer` returned in its registration response.

Registration rejected with 400

The NFProfile is missing a mandatory element or contains a malformed one. The NRF requires both `nfType` and at least one reachability address (`fqdn`, `ipv4Addresses`, or `ipv6Addresses`). A 400 with `MANDATORY_IE_MISSING` means `nfType`/`nfStatus` were absent from the PUT body; a 400 whose detail mentions reachability means no address was supplied.

Beyond those mandatory elements, the NRF also enforces deep structural validation of the rest of the profile and returns a 400 with cause `MANDATORY_IE_INCORRECT` when a present element is malformed (per TS 29.510). Common causes:

- `nfServices`: each service needs a `serviceInstanceId`, a `serviceName`, a valid `scheme`, a valid `nfServiceStatus`, and a non-empty `versions` array where each entry carries `apiVersionInUri`.
- `sNssais`: each entry needs an integer `sst` in the range 0 to 255, and an `sd` (when present) of exactly 6 hex digits.

- `plmnList`: entries must be well-formed PLMN objects.
- `capacity` and `priority` must be integers in the range 0 to 65535; `load` must be an integer in the range 0 to 100.

Absent optional elements are accepted; only a present-but-malformed element is rejected. The 400 `detail` names the offending field, so it points directly at what to correct.

PATCH rejected with 415 Unsupported Media Type

The PATCH Content-Type was set to something other than `application/json-patch+json`. Either send that media type for JSON-Patch updates, or send a genuinely empty heartbeat (no Content-Type is tolerated for heartbeats).

Update/heartbeat/delete returns 404

The `nfInstanceId` is not in the registry, typically because the NF was already aged out for missed heartbeats, or it never completed registration. Re-register with a PUT before patching or heartbeating.

PUT/PATCH returns 412 Precondition Failed

An `If-Match` header was sent whose ETag does not match the currently stored profile (the profile changed underneath the client). Re-`GET` the instance to obtain the current `ETag`, then retry.

Discovery returns an empty `nfInstances` list

- Confirm the target NF is registered and its stored `nfStatus` is `REGISTERED` (only registered NFs are returned).
- `target-nf-type` is matched case-sensitively against the stored `nfType`.
- If `plmn-id`, `snssais`, or `dnn` filters are supplied, verify they are well-formed and that the target NF's profile actually advertises them (an NF that advertises none of a given attribute matches any value for it).
- Ensure `requester-nf-type` is present. Its absence yields 400, not an empty list.

Status notifications are not being received

- Confirm `nfStatusNotificationUri` in the subscription is reachable from the NRF host.
- Delivery uses a bounded retry with exponential backoff (up to `notification_max_attempts` tries, default 4, with a base delay of `notification_backoff_ms`, default 500 ms, doubling each retry). A retry is logged as `NRF: Retrying status notification to ...`. If every attempt fails, the NRF gives up and logs `NRF: Giving up status notification to ... after N attempts`. Search the NRF logs for those two lines to see whether delivery was retried and eventually abandoned.
- If `subscrCond.nfType` is set, only status changes for that NF type are delivered.

Registry is empty after an NRF restart

Expected. The NF and subscription stores are in-memory (ETS). After a restart the registry repopulates as NFs re-register on their next registration/heartbeat cycle; there is no persistence to reload.