Advanced Routing

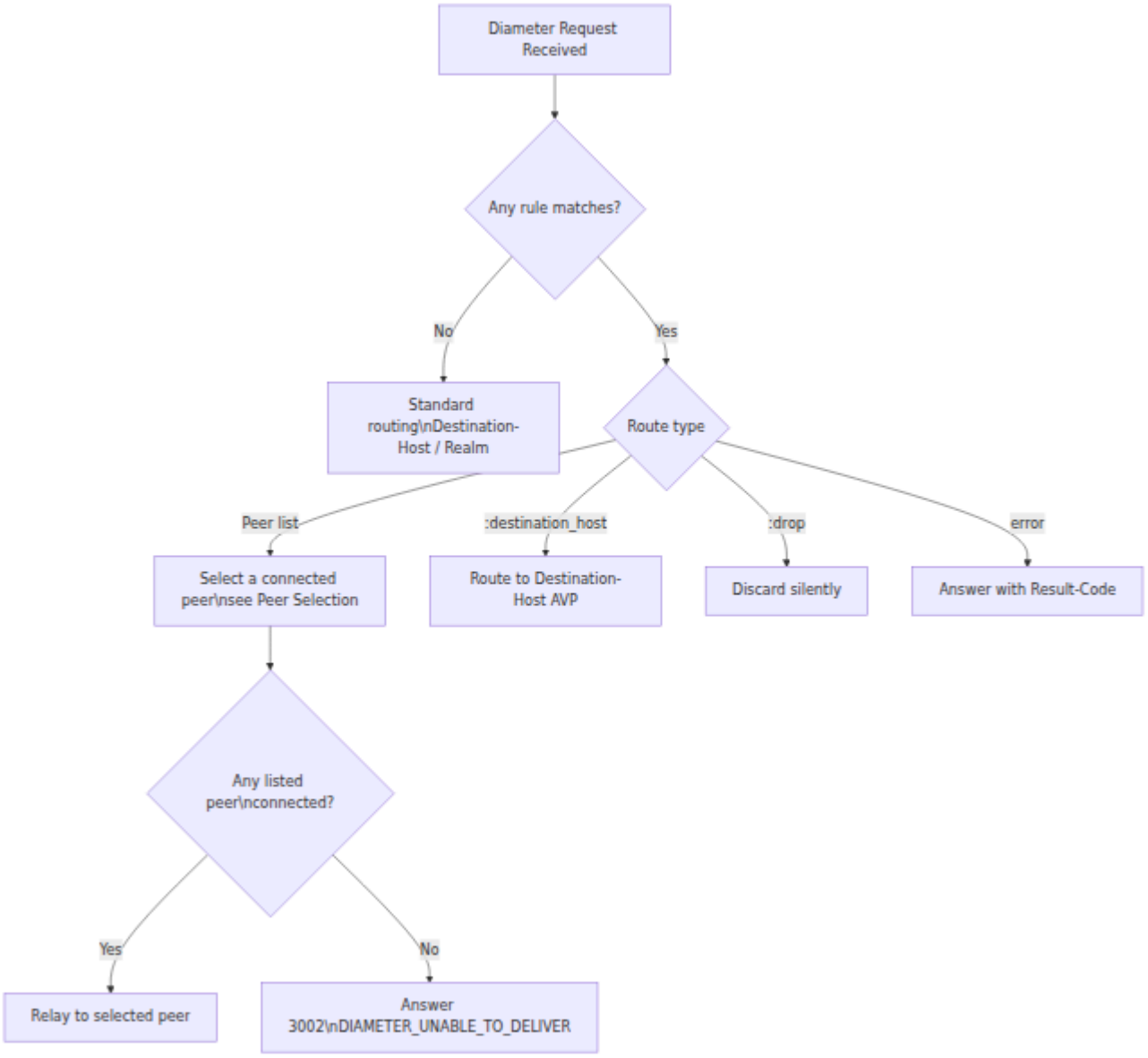
Advanced Routing lets the DRA make routing decisions based on the content of each Diameter request rather than **Destination-Realm** alone. **Rules** match on properties of the message (originating peer, Application-Id, Command-Code, AVP values) and, when matched, direct the request to a chosen set of peers or a special destination.

Beyond simply selecting a peer, a rule can distribute traffic across several peers for **failover** and **load balancing**, using per-peer `priority` and `weight`. For stateful applications such as Gx (PCRF) and Gy (OCS), Advanced Routing preserves **session affinity** so that mid-session messages always return to the peer that owns the session.

Advanced Routing evaluates **inbound Diameter request packets only**. Answer packets follow the established session routing back to the originating peer and are never re-evaluated against these rules. To rewrite message content, see Advanced Transform; for realm/host-based routing that runs when no rule matches, see Standard Routing.

How It Works

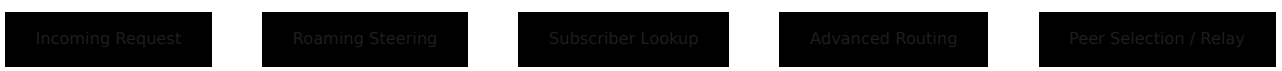
Every inbound request is evaluated against the configured rules **in order, from top to bottom**. The **first** rule whose filters match wins and decides the route — subsequent rules are not evaluated. If no rule matches, the request continues with standard **Destination-Host** / **Destination-Realm** routing.



A matched rule that resolves only to peers which are **not currently connected** is answered with **DIAMETER_UNABLE_TO_DELIVER (3002)**. The request is never silently handed back to standard routing, which would deliver it to the wrong node and return a misleading result.

Pipeline Position

Advanced Routing runs after Roaming Steering and Subscriber Lookup, and its decision overrides the peer that standard routing would otherwise have selected.



Rule Processing

Rules are evaluated top to bottom, and the **first matching rule wins**. Each rule combines its filters according to its `match` mode, and applies its `route` action as soon as it matches. If no rule matches, the DRA falls back to standard routing.

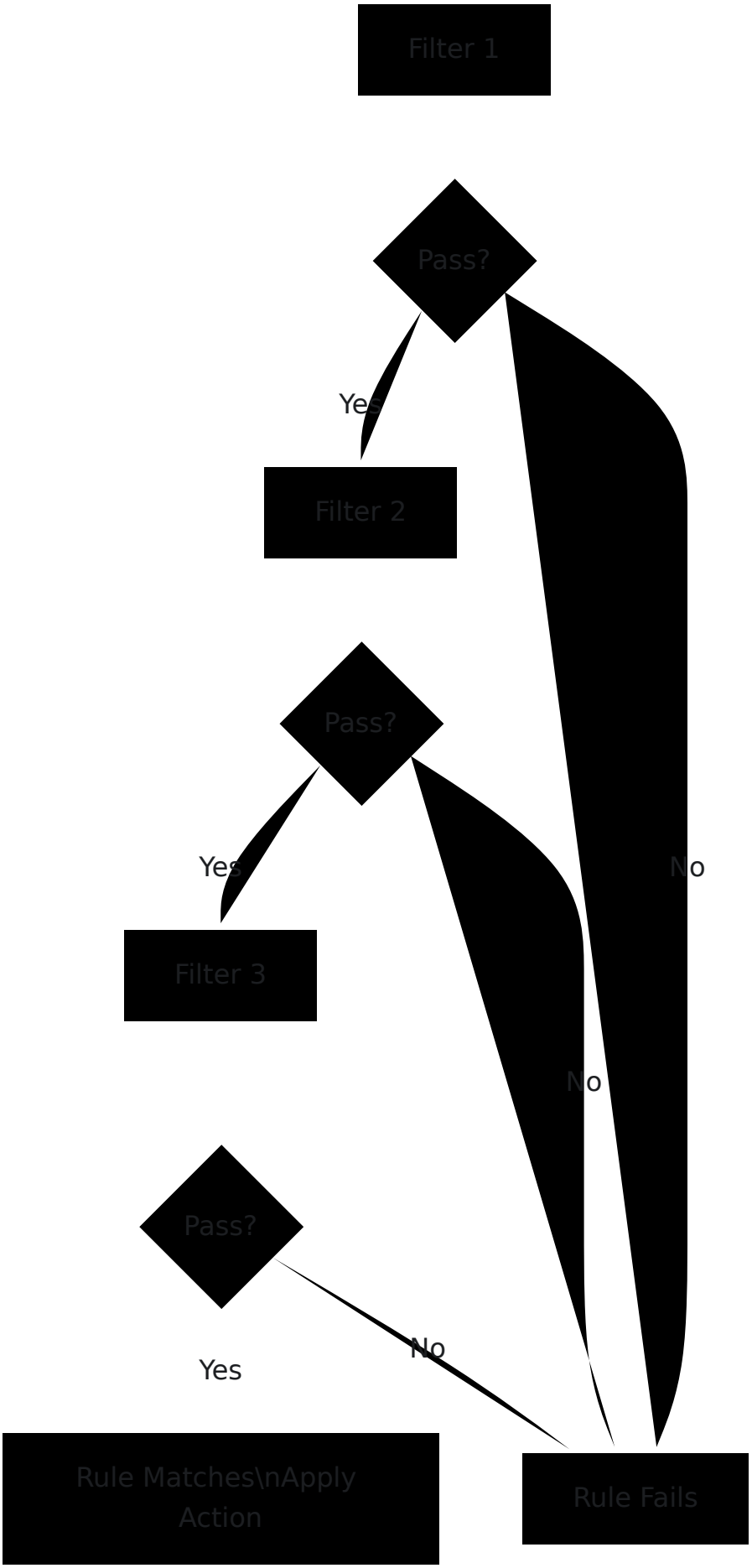
1. Rules are evaluated **in order from top to bottom** as defined in configuration.
2. Filters within a rule are combined according to the rule's `match` mode (`:all`, `:any`, or `:none`).
3. The **first matching rule wins** — subsequent rules are not evaluated.
4. If no rule matches, standard routing is used.

Match Modes

The `match` parameter determines how a rule's filters are combined.

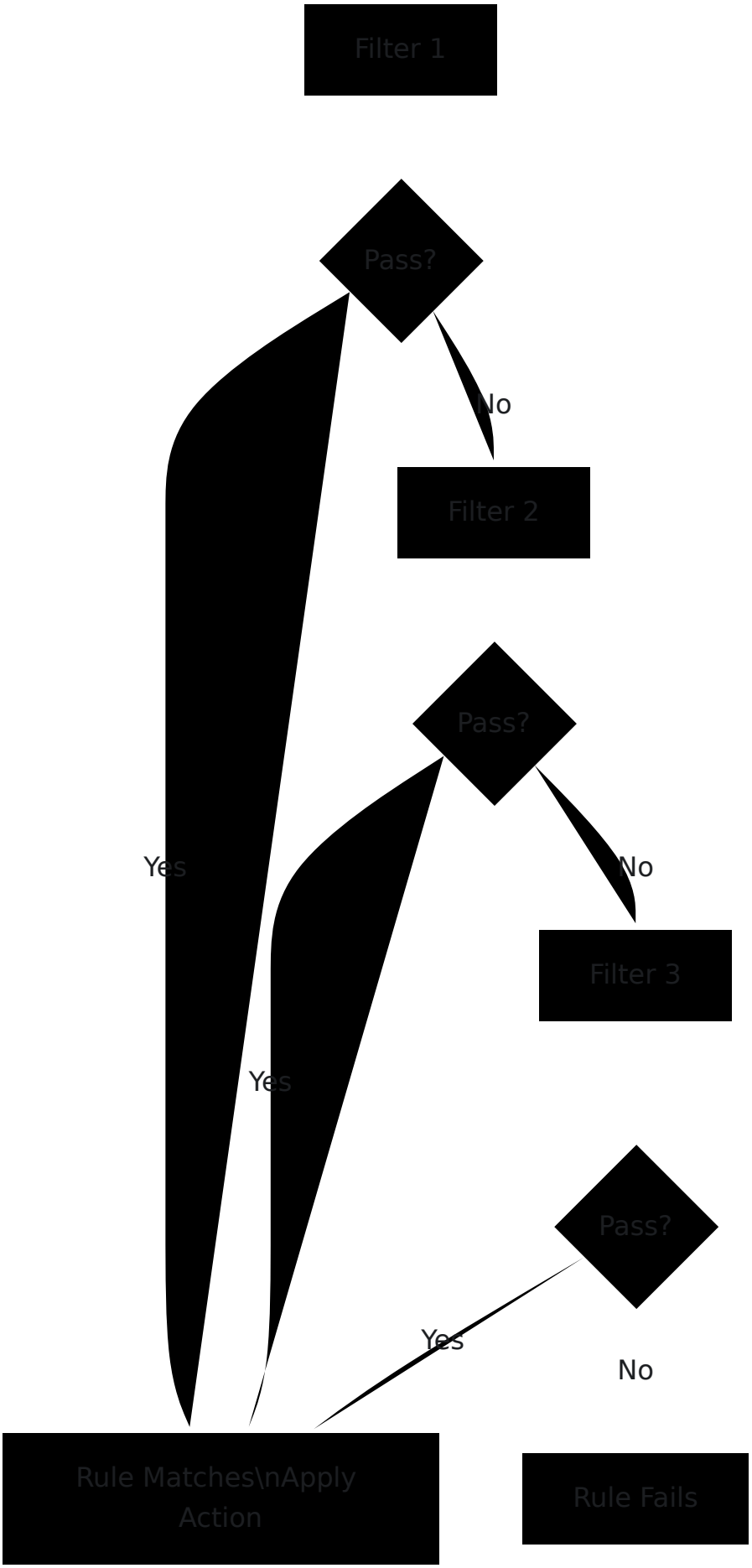
`:all` (AND logic)

Every filter must match for the rule to succeed. With three filters, `filter1 AND filter2 AND filter3` must all be true.


Filter 1
Pass?
Yes
Filter 2
Pass?
Yes
Filter 3
Pass?
Yes
No
No
No
Rule Matches\nApply Action
Rule Fails

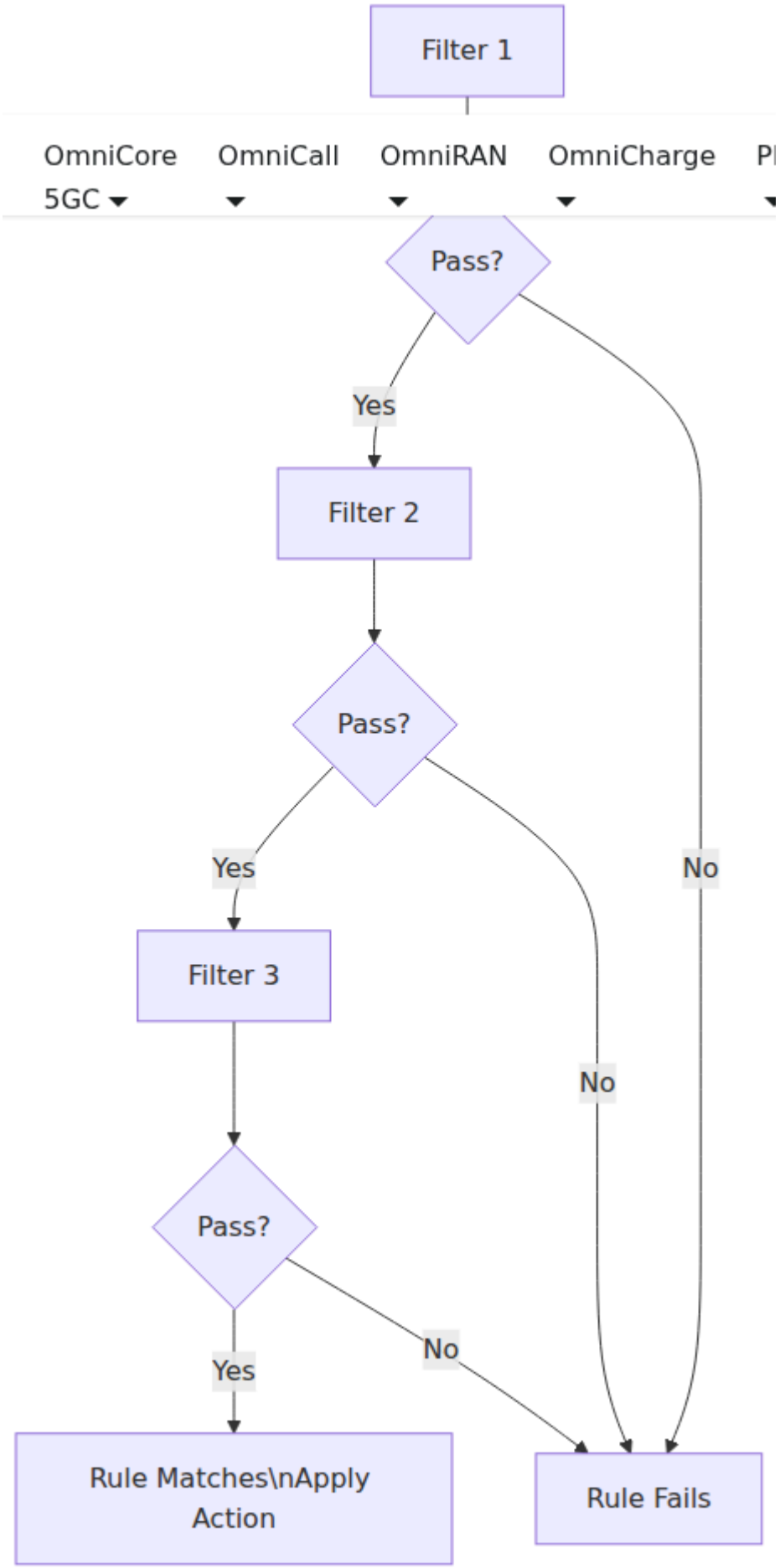
`:any` (OR logic)

At least one filter must match. With three filters, `filter1 OR filter2 OR filter3` succeeds if any one passes.


Filter 1
Pass?
No
Filter 2
Pass?
Yes
No
Filter 3
Yes
Pass?
Yes
No
Rule Matches\nApply Action
Rule Fails

`:none` (NOR logic)

No filter may match — inverse matching. With three filters, `NOT filter1 AND NOT filter2 AND NOT filter3` must all hold (every filter must fail) for the rule to succeed.


Filter 1
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Pass?
Yes
Filter 2
Pass?
Yes
Filter 3
Pass?
Yes
Rule Matches\nApply Action
No
Rule Fails
No
No

Peer Selection

When a rule routes to a peer list, the DRA only ever selects among peers that are **currently connected**. How it chooses between them depends on how the peers are written.

Bare hostnames

A list of plain hostnames follows the global `peer_selection_algorithm` (`:failover` or `:random`) configured under `diameter`. This is the original behaviour and is unchanged. Peers must be defined in the DRA's Diameter peer configuration, named by their **exact** hostname (case-sensitive), and currently connected — disconnected peers are skipped.

```
route: %{peers: ["hss01.example.com", "hss02.example.com"]}
```

- `:failover` — always the first connected peer in the list (deterministic primary, then secondary).
- `:random` — a random connected peer (even load spreading).

Priority and weight

For per-rule failover and load balancing, peers are written as maps with `priority` and `weight`. This is independent of the global algorithm.

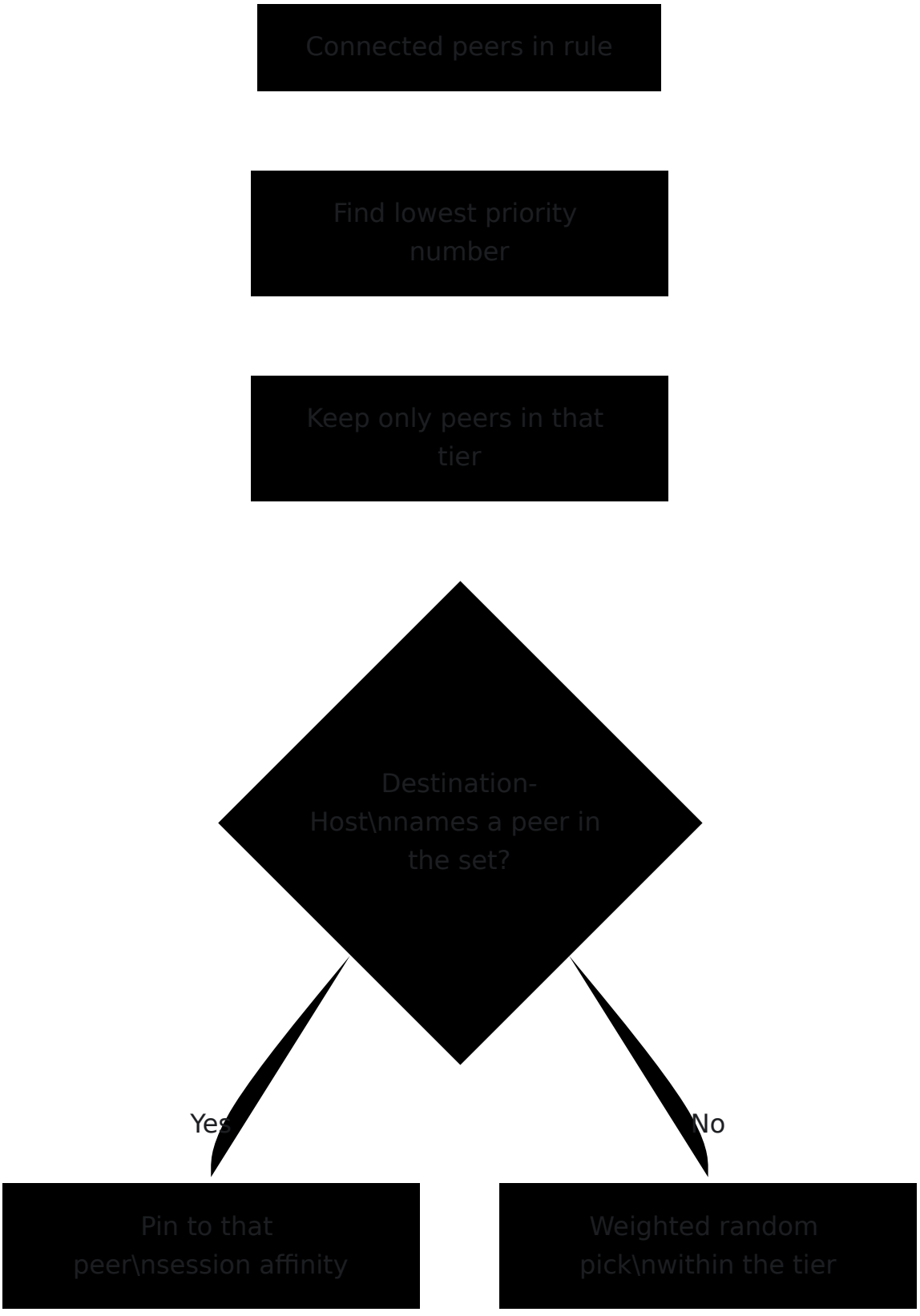
```
route: %{
  peers: [
    %{host: "pcrf01.example.com", priority: 10, weight: 50},
    %{host: "pcrf02.example.com", priority: 10, weight: 50},
    %{host: "pcrf-dr.example.com", priority: 20, weight: 100}
  ]
}
```

Selection is **"priority picks the tier, weight splits the load"**:

1. Only connected peers are considered.

2. The **lowest priority number** present wins the tier (lower = preferred, as with DNS SRV records). Higher-numbered tiers act as backups and are only used once every connected peer in a lower tier is gone — this provides **failover**.
3. Within the winning tier, one peer is chosen at random **weighted by `weight`**. Equal weights spread load evenly; unequal weights bias the split — this provides **load balancing**.

`priority` and `weight` each default to `100` when omitted.



This single model expresses every common topology:

Goal	How to express it
Pure failover	Distinct descending priorities (e.g. 10, 20, 30); weight irrelevant
Pure load balancing	One priority for all; equal weights
Weighted load balancing	One priority for all; weights in the desired ratio (e.g. 70 / 30)
Load balancing with backup	Two peers at priority 10, a third at priority 20

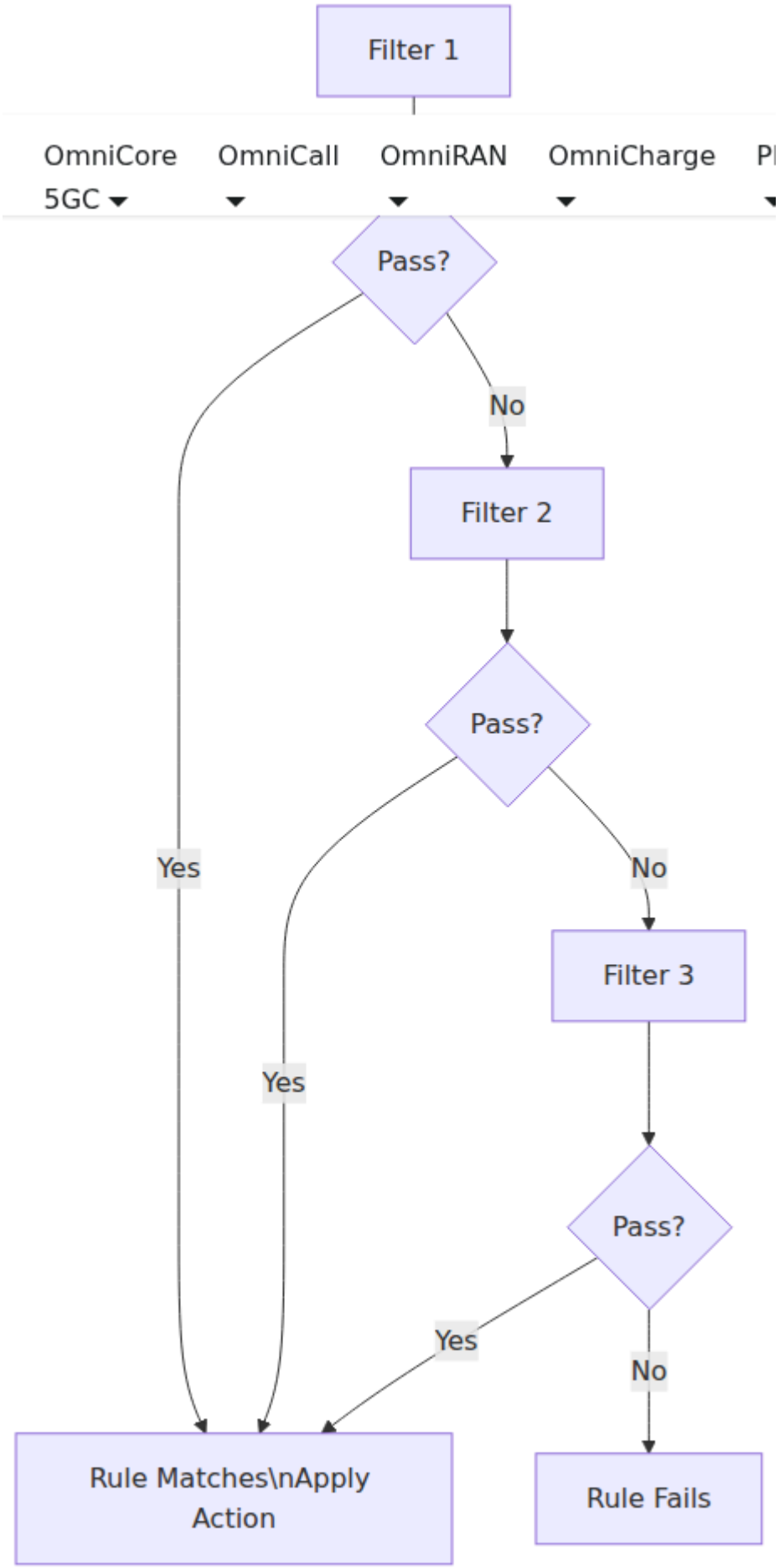
Session Affinity (Gx / Gy)

Stateful applications carry one **Session-Id** for the life of a session, and mid-session messages must reach the **same** server that handled the session-initiating request. If they were re-balanced to a different peer, that peer would reject them with **DIAMETER_UNKNOWN_SESSION_ID (5002)**.

Advanced Routing preserves affinity using the standard Diameter mechanism, the **Destination-Host AVP (293)**. When a rule uses priority/weight peers:

- If the request carries a Destination-Host that names one of the rule's connected peers, the request is **pinned** to that peer, overriding priority and weight.
- If there is no such Destination-Host, the request is load balanced normally.

The session-initiating request (e.g. Gx CCR-Initial) carries no Destination-Host, so it is load balanced across the tier. The answer returns the chosen server's Origin-Host, which the client then places in Destination-Host on all subsequent requests (CCR-Update, CCR-Terminate) — pinning them to that server automatically. In Gx/Gy the request type is distinguished by the **CC-Request-Type AVP (416)**.


Filter 1
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Pass?
No
Filter 2
Pass?
Yes
No
Filter 3
Yes
Pass?
Yes
No
Rule Matches\nApply Action
Rule Fails

This affinity applies to priority/weight peer lists. Bare-hostname lists follow the global algorithm and are not affected by this behaviour.

Configuration

Advanced Routing is configured under `module_advanced_routing` in `config/runtime.exs`.

```
module_advanced_routing: %{
  # Enable or disable the module
  enabled: false,

  # Rules are evaluated in order; the first match wins
  rules: [
    %{
      rule_name: "route_s6a_air_to_hss",
      match: :all,
      filters: [
        {:application_id, 16_777_251},
        {:command_code, 318}
      ],
      route: %{
        peers: ["hss01.example.com"]
      }
    }
  ]
}
```

Module Parameters

Parameter	Type	Required	Default	Description
`enabled`	Boolean	No	`false`	Master switch for the module. When `false`, no rules are evaluated and all traffic uses standard routing.
`rules`	List	Yes	`[]`	Ordered list of routing rules. Evaluated top to bottom; the first match wins. See Rule Parameters.

Rule Parameters

Parameter	Type	Required	Default	Description
`rule_name`	String	Yes	—	A descriptive, unique name for the rule. Shown in logs and the management UI.
`match`	Atom	Yes	—	How filters combine: `:all` (logical AND), `:any` (logical OR), or `:none` (logical NOR). See Match Modes.
`filters`	List	Yes	—	The conditions a message must meet. See Filters.
`route`	Map or Atom	Yes	—	Where matching traffic is sent. A peer list, or a special destination. See Route Destinations.

Filters

A filter is a tuple of `{filter_type, value}`. The filters available to Advanced Routing are:

Filter	Example	Description
`:via_peer`	`{:via_peer, "mme01.example.com"}`	The Origin-Host of the peer the request was received from (the most recent hop).
`:application_id`	`{:application_id, 16_777_251}`	The Diameter Application-Id. See Application IDs.
`:command_code`	`{:command_code, 318}`	The Diameter Command-Code.
`:avp`	`{:avp, {1, "999999000000001"}}`	An AVP value, written as `{avp_code, value}`.

Filter values

- **List (OR) values.** Any filter value may be a list, which is treated as a logical OR over its members — for example `{:command_code, [317, 318]}` matches command code 317 or 318, and `{:via_peer, ["dra01.example.com", "dra02.example.com"]}` matches either peer.
- **Regex values.** AVP filter values (and list members) support regular expressions via a regex sigil, e.g. `{:avp, {1, ~r"999001.*"}}`. Members can be mixed, e.g. `{:avp, {1, ["505571234567", ~r"999001.*", ~r"505057.*"]}}`.
- **Presence check (`:any`).** A value of `:any` matches when the attribute is simply present with any value — for example `{:avp, {264, :any}}` matches whenever the Origin-Host AVP exists, and `{:via_peer, :any}` matches any originating peer.

AVP filter limitations

AVP filters are recommended for **simple** AVPs only (such as User-Name, Origin-Host, Destination-Realm). **Grouped AVPs are not supported** and will not match, and complex binary values are not supported. Always use the `{:avp, {code, value}}` form.

Route Destinations

The `route` parameter takes either a peer-list map or one of the special destination atoms/tuples below.

Destination	Example	Description
Peer list	`%{peers: [...]}`	Route to one of the listed peers. Entries may be bare hostnames or priority/weight maps. See Peer Selection.
Destination-Host	`:destination_host`	Route to the peer named in the message's **Destination-Host AVP (293)**. If the AVP is absent, the rule does not apply and standard routing is used.
Drop	`:drop`	Silently discard the message with no response sent to the requesting peer. Use for blackholing unwanted traffic (test subscribers, unwanted ranges).
Error	`{:error, 3004}`	Respond with a Diameter error answer carrying the given Result-Code. The DRA populates Origin-Host, Origin-Realm and Session-Id automatically; the message is not forwarded to any peer.

For a **relay** application, only 3xxx (protocol) result codes are strictly valid on generated answers (`3002`, `3003`, `3004`, `3007`). Some deployments also use `5012` (DIAMETER_UNABLE_TO_COMPLY) to reject specific commands; see the reference tables for meanings.

Peer Parameters (priority/weight form)

Parameter	Type	Required	Default	Description
`host`	String	Yes	—	The peer's Origin-Host. Must exactly match the peer's configured Diameter identity (case-sensitive).
`priority`	Integer	No	`100`	The tier. Lower numbers are preferred; higher-numbered tiers are used only when every connected peer in a lower tier is unavailable.
`weight`	Integer	No	`100`	The share of traffic within a tier. Traffic is split across a tier's peers in proportion to their weights. A weight of `0` excludes a peer unless every peer in the tier is `0`.

Configuration Examples

Failover to a Backup Peer

Route S6a Authentication-Information-Requests to a primary HSS, falling back to a disaster-recovery HSS only when the primary is down.

```
%{
  rule_name: "s6a_air_failover",
  match: :all,
  filters: [
    {:application_id, 16_777_251},
    {:command_code, 318}
  ],
  route: %{
    peers: [
      %{host: "hss01.example.com", priority: 10},
      %{host: "hss-dr.example.com", priority: 20}
    ]
  }
}
```

How it works: While `hss01` is connected, every request goes to it. If `hss01` disconnects, requests fail over to `hss-dr`. If both are down, requests are answered with 3002.

Even Load Balancing

Spread Gx traffic evenly across two PCRFs.

```
%{
  rule_name: "gx_load_balance",
  match: :all,
  filters: [
    {:application_id, 16_777_238}
  ],
  route: %{
    peers: [
      %{host: "pcrf01.example.com", priority: 10, weight: 50},
      %{host: "pcrf02.example.com", priority: 10, weight: 50}
    ]
  }
}
```

How it works: New sessions (CCR-Initial) are split roughly 50/50 between the two PCRFs. Mid-session messages are pinned to whichever PCRF handled the

session, via the Destination-Host they carry.

Weighted Load Balancing with a Backup

Send more traffic to a larger PCRF, less to a smaller one, and keep a backup for when both are unavailable.

```
%{
  rule_name: "gx_weighted_with_backup",
  match: :all,
  filters: [
    {:application_id, 16_777_238}
  ],
  route: %{
    peers: [
      %{host: "pcrf01.example.com", priority: 10, weight: 70},
      %{host: "pcrf02.example.com", priority: 10, weight: 30},
      %{host: "pcrf-dr.example.com", priority: 20, weight: 100}
    ]
  }
}
```

How it works: While either priority-10 PCRF is connected, new sessions are split 70/30 between them. Only when **both** priority-10 PCRFs are down does traffic move to `pcrf-dr`. Established sessions remain pinned to their server for their lifetime.

Route by Originating Peer

Route S6a traffic that arrives via specific upstream DRAs to a local HSS pair — useful during a migration or cutover.

```
%{
  rule_name: "s6a_via_peer_to_local_hss",
  match: :all,
  filters: [
    {:application_id, 16_777_251},
    {:via_peer, ["dra01.example.com", "dra02.example.com"]},
    {:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}}
  ],
  route: %{
    peers: ["hss01.example.com", "hss02.example.com"]
  }
}
```

How it works: Only S6a requests that arrived from `dra01` or `dra02` and carry the matching Origin-Realm (AVP 296) are routed to the local HSS nodes; everything else falls through to later rules or standard routing.

Pattern-Matched Roaming by IMSI

Route inbound roaming traffic based on IMSI (User-Name, AVP 1) patterns.

```
%{
  rule_name: "inbound_s6a_roaming",
  match: :all,
  filters: [
    {:application_id, 16_777_251},
    {:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}},
    {:avp, {1, ["505571234567", ~r"999001.*"]}}
  ],
  route: %{
    peers: ["dra01.example.com", "dra02.example.com"]
  }
}
```

How it works: Matches S6a messages from the given Origin-Realm whose User-Name is exactly `505571234567` or begins with `999001`, and routes them to the designated peers. The list value applies OR logic across the exact string and the regex.

Honour the Destination-Host AVP

Route to whatever peer the sender named, for matching traffic only.

```
%{
  rule_name: "honour_destination_host",
  match: :all,
  filters: [
    {:avp, {1, [~r"90199.*"]}}
  ],
  route: :destination_host
}
```

How it works: When the IMSI matches, the request is sent to the peer named in its Destination-Host AVP (293). If the Destination-Host AVP is missing, the match is treated as a failure and standard routing applies.

Drop Unwanted Traffic

Silently discard test-range traffic.

```
%{
  rule_name: "drop_test_subscribers",
  match: :all,
  filters: [
    {:application_id, 16_777_251},
    {:avp, {1, [~r"999999.*"]}}
  ],
  route: :drop
}
```

How it works: S6a messages whose IMSI begins with `999999` are dropped with no response. Useful for filtering test traffic or blocking specific subscriber ranges.

Overload Protection with an Error Answer

Return DIAMETER_TOO_BUSY to a specific overloaded peer.

```
%{
  rule_name: "rate_limit_high_volume_peer",
  match: :all,
  filters: [
    {:via_peer, "mme-overloaded-01.example.com"},
    {:application_id, 16_777_251}
  ],
  route: {:error, 3004}
}
```

How it works: S6a traffic from the named peer is answered with **DIAMETER_TOO_BUSY (3004)**, signalling the peer to back off. Nothing is forwarded upstream.

Block a Specific Command

Reject a particular command type with an explanatory error rather than dropping silently.

```
%{
  rule_name: "block_purge_requests",
  match: :all,
  filters: [
    {:application_id, 16_777_251},
    {:command_code, 321}
  ],
  route: {:error, 5012}
}
```

How it works: S6a Purge-UE-Requests (command 321) are answered with **DIAMETER_UNABLE_TO_COMPLY (5012)**, selectively disabling that operation while leaving other commands untouched.

Reading the Active Configuration

The active rules can be read over the management API at `GET /api/rules`. The response includes the routing rules, the transform rules, and the global

`routing_algorithm`.

For each rule that routes to a peer list, the API reports how selection will behave:

- `route.selection` is `"priority_weight"` when the rule uses priority/weight maps, otherwise the global algorithm (`"failover"` or `"random"`).
- In `priority_weight` mode, each peer is returned with its effective `priority` and `weight`, with defaults filled in so the distribution is explicit.


```
{
  "status": "success",
  "response": {
    "routing_algorithm": "random",
    "routing_rules": [
      {
        "rule_name": "gx_weighted_with_backup",
        "match": "all",
        "route": {
          "selection": "priority_weight",
          "peers": [
            { "host": "pcrf01.example.com", "priority": 10,
"weight": 70 },
            { "host": "pcrf02.example.com", "priority": 10,
"weight": 30 },
            { "host": "pcrf-dr.example.com", "priority": 20,
"weight": 100 }
          ]
        }
      }
    ],
    "transform_rules": []
  }
}
```


Best Practices

1. **Specificity first.** Order rules from most specific to most general, since the first match wins.

2. **Common matches early.** Place the highest-volume matches near the top to reduce evaluation overhead.
3. **Validate regex.** Test regular-expression patterns against captured messages before deployment.
4. **Name clearly.** Use descriptive `rule_name` values so logs and the management UI are self-explanatory.
5. **Prefer priority/weight for stateful apps.** Use priority/weight peer maps for Gx/Gy so session affinity is applied automatically.

Troubleshooting

Sessions Rejected with UNKNOWN_SESSION_ID

Symptoms: Gx or Gy sessions establish correctly but mid-session updates are rejected by the server with Result-Code 5002.

Possible causes:

- The clients (PCEF/OCF) are not setting Destination-Host on mid-session requests, so the DRA cannot pin them.
- A rule is routing this traffic using bare hostnames with the global `:random` algorithm, which does not apply session affinity.

Resolution:

1. Confirm that mid-session requests carry a Destination-Host naming the server that answered the initial request.
2. Convert the rule's peer list to priority/weight maps so session affinity is applied automatically.

Matched Traffic Answered with 3002

Symptoms: Requests that should be routed by a rule are answered with DIAMETER_UNABLE_TO_DELIVER.

Possible causes:

- None of the peers listed in the matched rule are currently connected.
- The hostnames in the rule do not exactly match the peers' Origin-Host values.

Resolution:

1. Check peer connection state in the management UI or logs.
2. Ensure the `host` values in the rule exactly match the peers' Origin-Host (case-sensitive).

A Rule Is Not Taking Effect

Symptoms: Traffic that should match a rule is routed by standard routing instead.

Possible causes:

- `enabled` is set to `false`.
- An earlier rule in the list matches first and wins the route.
- A filter value does not match (for example a case mismatch on an AVP value, or an unsupported grouped AVP).

Resolution:

1. Verify `enabled: true` and that the DRA was restarted after the change.
2. Review rule order — the first matching rule wins.
3. Confirm the filter values against a captured message, and remember grouped AVPs cannot be matched.

Reference

Diameter Result Codes

Code	Name	Description	Reference
3002	DIAMETER_UNABLE_TO_DELIVER	No connected peer is available to deliver the request	RFC 6733 §7.1.3
3003	DIAMETER_REALM_NOT_SERVED	No configured route serves the request's realm	RFC 6733 §7.1.3
3004	DIAMETER_TOO_BUSY	The node is overloaded and cannot process the request	RFC 6733 §7.1.3
5002	DIAMETER_UNKNOWN_SESSION_ID	The server does not recognise the Session-Id (e.g. a mid-session message reached the wrong server)	RFC 6733 §7.1.5
5012	DIAMETER_UNABLE_TO_COMPLY	The request cannot be honoured (used to reject	RFC 6733 §7.1.5

Code	Name	Description	Reference
		specific commands)	

Relevant AVPs

AVP	Code	Description	Reference
Destination-Host	293	Names the specific peer a request must be delivered to; the basis for session affinity	RFC 6733 §6.5
Session-Id	263	Identifies a session for its lifetime	RFC 6733 §8.8
CC-Request-Type	416	Distinguishes INITIAL, UPDATE and TERMINATION requests in Gx/Gy	RFC 4006 §8.3

Application IDs

ID	Interface	Description	Reference
16777238	Gx	Policy control between PCEF and PCRF	3GPP TS 29.212
16777251	S6a/S6d	MME/SGSN to HSS authentication and subscription management	3GPP TS 29.272
4	Gy	Online charging between OCF and OCS	RFC 4006

Related Documentation

- Standard Routing — Destination-Host / Destination-Realm routing used when no rule matches.
- Advanced Transform — Content rewriting for requests and answers, sharing the same filter engine.
- Configuration Reference — Full `config/runtime.exs` parameter reference, including the global `peer_selection_algorithm`.

Advanced Transform

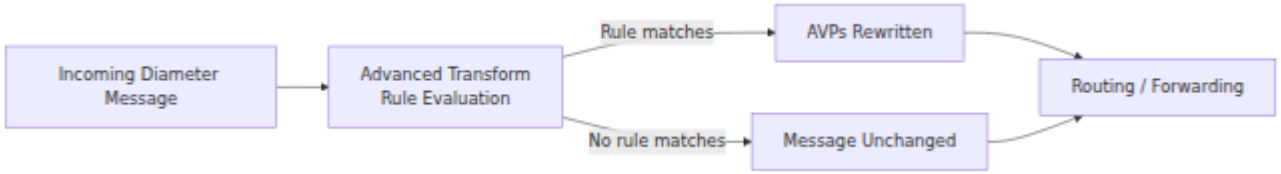
The **Advanced Transform** module rewrites Attribute-Value Pairs (**AVPs**) inside Diameter messages that match operator-defined criteria. It lets you correct, translate, or normalise message contents as they pass through OmniDRA without changing the sending or receiving network elements.

Overview

Advanced Transform evaluates a list of **rules** against each Diameter message. A rule combines a set of **filters** (the match criteria) with a **transform** (the change to apply). When a rule matches, its transform is applied to the message and rule processing stops.

Three transform actions are available: `:edit` changes the value of an existing AVP, `:remove` deletes AVP(s) by code, and `:overwrite` rewrites AVP(s) by name (including grouped AVPs). See Transform Actions.

Advanced Transform is closely related to, and shares its rule-processing model with, the routing engine. See Advanced Routing for how routing decisions are made.

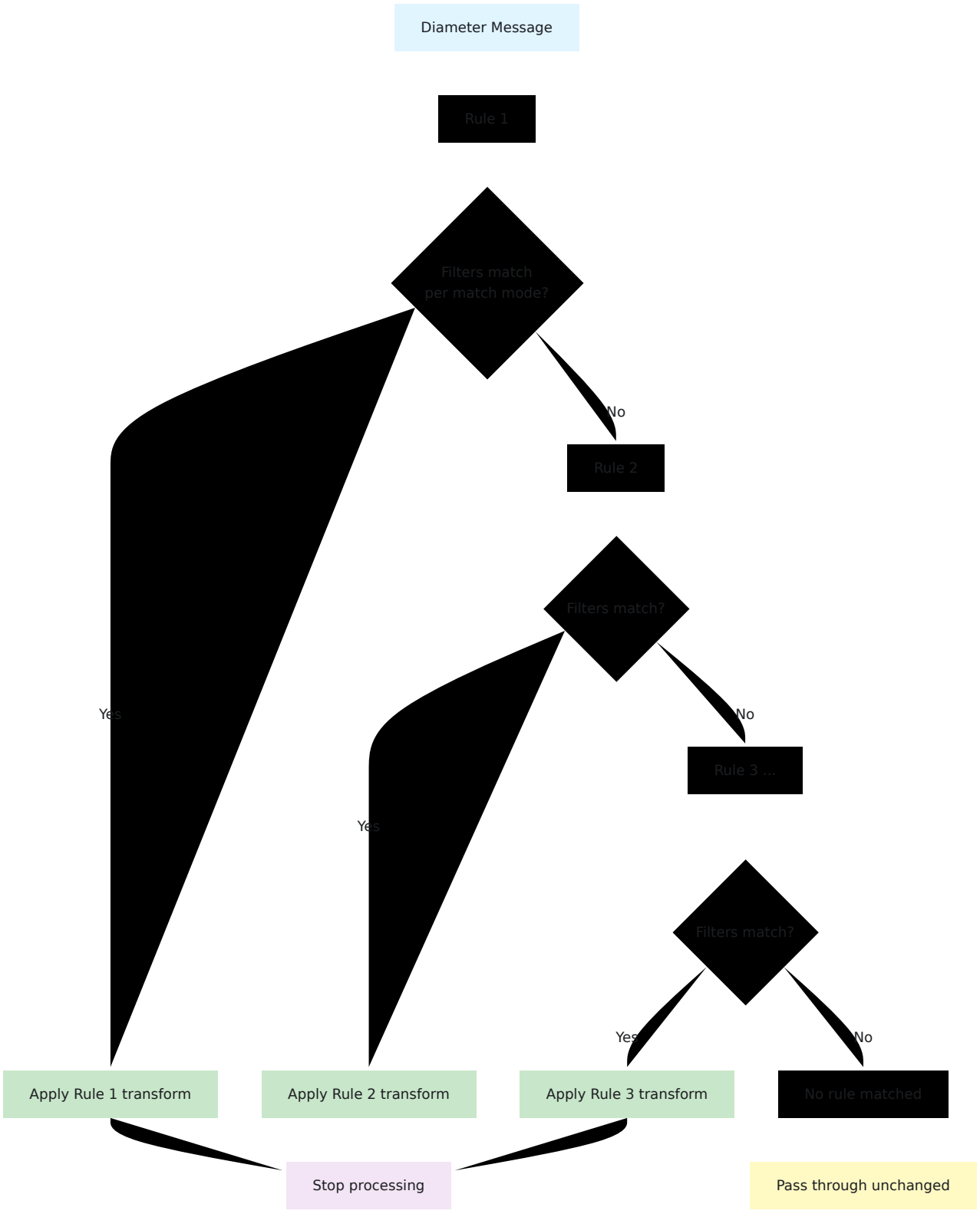


How Rules Are Processed

Rules are evaluated **in order, from top to bottom**, as they appear in the configuration. The **first matching rule wins**: as soon as a rule's filters match, its transform is applied and no further rules are considered. If no rule matches, the message passes through **transparently** with no modification.

Within a single rule, the `match` parameter controls how the individual filters combine:

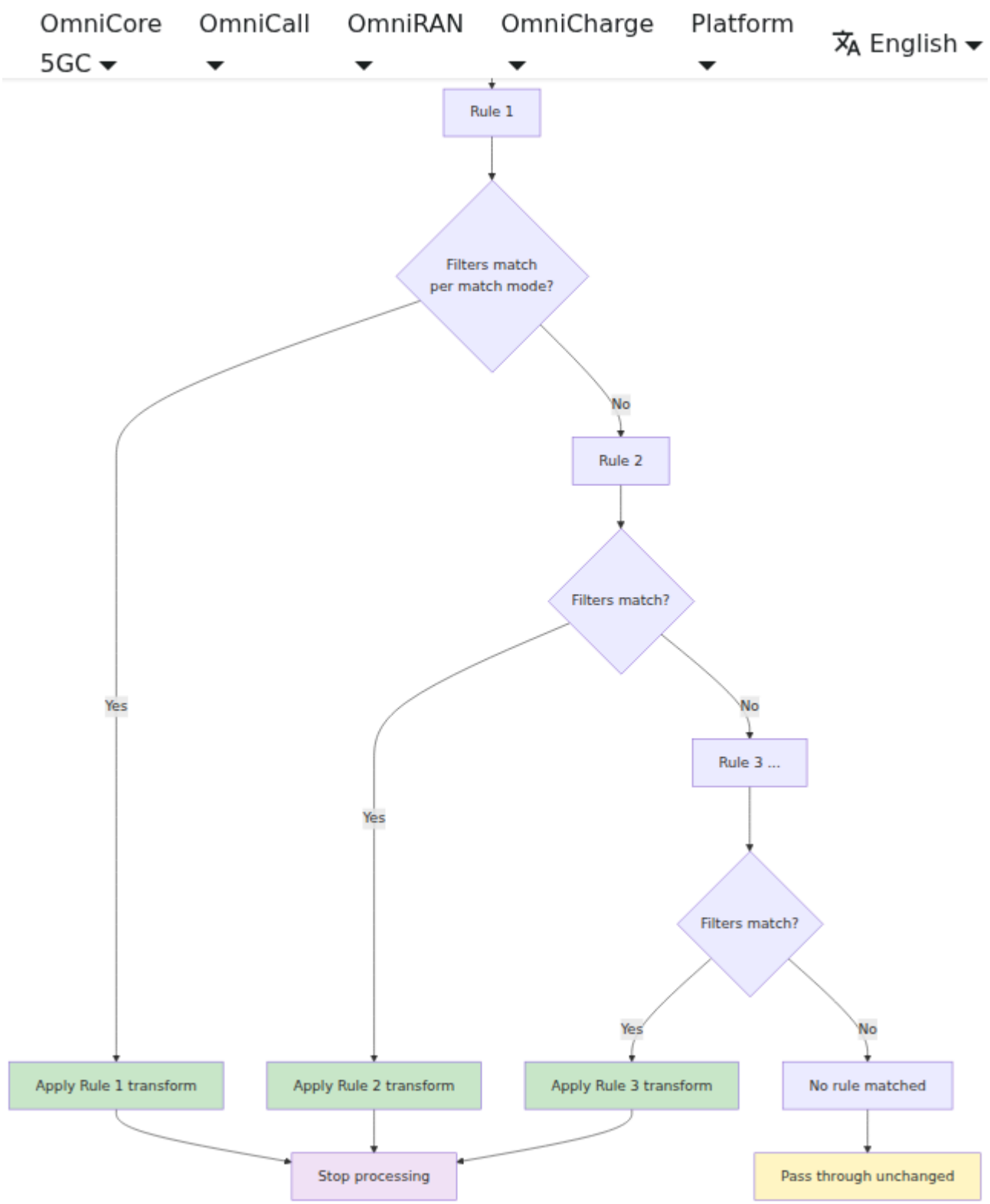
- `:all` — logical **AND**. Every filter must pass for the rule to match.
- `:any` — logical **OR**. At least one filter must pass for the rule to match.
- `:none` — logical **NOR**. No filter may pass for the rule to match (inverse matching).



Relative to routing, transform rules operate on the message contents and do not themselves select a destination. A rule may rewrite an AVP (for example a realm or identity) that the routing engine subsequently uses to forward the message.

Match Mode Evaluation

The three match modes evaluate the same filter list differently:



Filters

Filters are the conditions that decide whether a rule applies to a message. Six filter types are available.

Filter	Format	Description
`:packet_type`	`{:packet_type, :request}`	Matches the message **direction**. Accepts `:request` or `:answer` only; a list of values is **not** accepted. See Packet-Type Filtering.
`:application_id`	`{:application_id, <id>}`	Matches the Diameter Application-Id of the message (for example `16777251` for S6a/S6d). See Application IDs.
`:command_code`	`{:command_code, <code>}`	Matches the Diameter Command Code of the message (for example `318` for AIR). See Command Codes.
`:avp`	`{:avp, {<code>, <value>}}`	Matches an AVP by its numeric code and expected value. See AVP Filtering.
`:to_peer`	`{:to_peer, "hss01.example.com"}`	Matches the **resolved destination peer** of a request. Only matches **request** packets — it never matches answers. Accepts a list of hostnames (OR). See Peer Filtering.
`:from_peer`	`{:from_peer, "hss01.example.com"}`	Matches the **peer that answered** a message. Only matches **answer** packets — it never matches requests.

Filter	Format	Description
		Accepts a list of hostnames (OR). See Peer Filtering.

Packet-Type Filtering

The `:packet_type` filter restricts a rule to one direction of traffic. It accepts exactly one of two values:

- `:request` — matches only Diameter **request** messages.
- `:answer` — matches only Diameter **answer** messages.

Unlike `:application_id`, `:command_code`, and `:avp`, the `:packet_type` filter does **not** accept a list of values.

AVP Filtering

The `:avp` filter is structured as `{avp_id, avp_value}`, where `avp_id` is the numeric AVP code and `avp_value` is the value to compare against, expressed as a string.

The `:avp` filter is intended for **simple AVPs** such as `User-Name` and `Origin-Host`. **Grouped AVPs are not supported** and will not match, and complex binary values will not match.

Values may be matched in several ways:

- **Exact string** — `{:avp, {1, "999999000000001"}}` matches only that exact value.
- **List of values** — `{:avp, {1, ["50557...", "505057..."]}}` matches if the AVP equals **any** value in the list (OR logic within the list).
- **Regular expression** — `{:avp, {1, [~r"9999999.*"]}}` matches values against a regex pattern.

Use the `~r"pattern"` syntax for regular-expression matching:

Pattern	Matches
`~r"999001.*"`	`User-Name`/IMSI values starting with `999001`
`~r"^310[0-9]{3}.*"`	Values with specific MCC/MNC prefixes
`~r".*test$"`	Values ending with `test`

Peer Filtering

The `:to_peer` and `:from_peer` filters match against a **peer hostname** rather than the message contents. They are direction-specific and complementary:

- **`:to_peer`** matches the **resolved destination peer** that a request is about to be forwarded to. It only ever matches **request** packets; against an answer it never matches.
- **`:from_peer`** matches the **peer that answered** a message. It only ever matches **answer** packets; against a request it never matches.

Both filters accept either a single hostname or a **list of hostnames**, in which case the filter matches if the peer equals **any** hostname in the list (OR logic).

Filter	Example	Matches
`:to_peer`	`{:to_peer, "hss01.example.com"}`	Requests routed to peer `hss01.example.com`
`:to_peer`	`{:to_peer, ["hss01.example.com", "hss02.example.com"]}`	Requests routed to **either** listed peer
`:from_peer`	`{:from_peer, "hss01.example.com"}`	Answers received from peer `hss01.example.com`

Transform Actions

The transform is described by the `transform` map. Its `action` key selects **one of three** transformations. Which actions are permitted depends on the **direction** of the message:

Action	What it does	Requests	Answers
`:edit`	Rewrites the **value** of an existing AVP.	Yes	No
`:remove`	Deletes the named AVP(s) from the message.	Yes	Yes
`:overwrite`	Rewrites AVP(s) by name, including **grouped** AVPs, using a decoding dictionary.	Yes	Yes

The `:edit` Action

```
transform: %{
  action: :edit,
  avps: [{:avp, {1, "999999000000002"}}]
}
```

`:edit` behaves as follows:

- It **modifies the value** of the named AVP in the Diameter message to the supplied value.
- It only affects AVPs that **already exist** in the message. If the targeted AVP is not present, **no change is made** and the message is passed on as received.
- Each entry in `avps` is `{:avp, {<code>, <new_value>}}` and sets that AVP to the new value.
- `:edit` applies to **requests only**. To rewrite AVPs on answers, or to reach grouped AVPs, use `:overwrite`.

The `:remove` Action

```
transform: %{
  action: :remove,
  avps: [{:avp, {264, :any}}]
}
```

`:remove` behaves as follows:

- It **deletes** the AVP(s) named in `avps` from the message.
- Matching is by **AVP code only** — the value in each `avps` entry is **ignored** for removal. Use `:any` as the value to make this explicit.
- It applies to both **requests and answers**.

The `:overwrite` Action

```
transform: %{
  action: :overwrite,
  dictionary: :diameter_gen_3gpp_s6a,
  avps: [{:avp, {1, "999999000000002"}}]
}
```

`:overwrite` behaves as follows:

- It **rewrites** the named AVP(s) by name, using the supplied `dictionary` to decode and re-encode the message.
- Because the message is fully decoded, `:overwrite` can reach **grouped / record AVPs** that `:edit` cannot.
- It **requires** a `dictionary` key naming the Diameter dictionary that describes the message (for example `:diameter_gen_3gpp_s6a`).
- It applies to both **requests and answers**.

Transform Map Parameters

Parameter	Type	Required	Default	Description
`action`	Atom	Yes	-	The transformation to apply: `:edit` (rewrite an existing AVP value — **requests only**), `:remove` (delete AVP(s) by code — requests and answers), or `:overwrite` (rewrite AVP(s) by name, including grouped AVPs — requests and answers).
`avps`	List	Yes	-	List of AVP targets. For `:edit`/`:overwrite`, each entry is `{:avp, {<code>, <new_value>}}` and sets that AVP to `<new_value>`. For `:remove`, only `<code>` is used and the value is ignored (use `:any`).
`dictionary`	Atom	Only for `:overwrite`	`:not_used`	The Diameter dictionary used to decode/re-encode the message so named (grouped) AVPs can be overwritten, e.g. `:diameter_gen_3gpp_s6a` Ignored by `:edit` and `:remove`.

Configuration

Advanced Transform is configured under `module_advanced_transform` in `config/runtime.exs`. The module is disabled by default and does nothing until `enabled` is set to `true`.

```
module_advanced_transform: %{
  enabled: true,
  rules: [
    %{
      rule_name: "transform_user_name",
      match: :all,
      filters: [
        {:application_id, 16_777_251},
        {:command_code, 318},
        {:avp, {1, "999999000000001"}}
      ],
      transform: %{
        action: :edit,
        avps: [{:avp, {1, "999999000000002"}}]
      }
    }
  ]
}
```

Module Parameters

Parameter	Type	Required	Default	Description
`enabled`	Boolean	No	`false`	Enables the module. When `false`, no rules are evaluated and all messages pass through unchanged.
`rules`	List	Yes (when enabled)	-	Ordered list of transform rules. Evaluated top to bottom; the first matching rule wins. See Rule Parameters.

Rule Parameters

Each entry in `rules` is a map with the following keys.

Parameter	Type	Required	Default	Description
`rule_name`	String	Yes	-	Unique, descriptive identifier for the rule. Used for operational clarity and monitoring.
`match`	Atom	Yes	-	How filters combine: `:all` (AND), `:any` (OR), or `:none` (NOR). See How Rules Are Processed.
`filters`	List	Yes	-	List of filter conditions that must be satisfied per the `match` mode. See Filters.
`transform`	Map	Yes	-	The change to apply when the rule matches. See Transform Map Parameters.

Examples

Example 1: Rewrite a User-Name on S6a AIR Requests

```
module_advanced_transform: %{
  enabled: true,
  rules: [
    %{
      rule_name: "transform_user_name",
      match: :all,
      filters: [
        {:application_id, 16_777_251},
        {:command_code, 318},
        {:avp, {1, "999999000000001"}}
      ],
      transform: %{
        action: :edit,
        avps: [{:avp, {1, "999999000000002"}}]
      }
    }
  ]
}
```

How it works: With `match: :all`, every filter must pass. The rule matches only S6a (`16777251`) Authentication-Information-Request (`318`) messages whose `User-Name` (AVP `1`) equals `999999000000001`. When matched, the `:edit` action rewrites `User-Name` to `999999000000002`. If a message carries no `User-Name` AVP, no change is made.

Use case: Correcting or normalising a specific subscriber identity as it transits the DRA — for example, mapping a legacy IMSI to its current value during a migration.

Example 2: Realm Rewriting Based on IMSI Range

```
module_advanced_transform: %{
  enabled: true,
  rules: [
    %{
      rule_name: "rewrite_destination_realm_roaming_partner",
      match: :all,
      filters: [
        {:avp, {296, "epc.mnc057.mcc505.3gppnetwork.org"}},
        {:avp, {1, [~r"50557.*"]}}
      ],
      transform: %{
        action: :edit,
        avps: [{:avp, {283, "epc.mnc030.mcc310.3gppnetwork.org"}}]
      }
    }
  ]
}
```

How it works: The rule matches when the `Origin-Realm` (AVP `296`) equals the roaming partner's realm **and** the `User-Name`/IMSI (AVP `1`) begins with `50557`. It then rewrites the `Destination-Realm` (AVP `283`) so the routing engine forwards the message toward the correct partner network. Because the first matching rule wins, place more specific IMSI ranges above broader ones.

Use case: Directing roaming or MVNO subscriber traffic to the correct hosted core network based on their IMSI range and originating realm.

Example 3: Strip an AVP From Answers From a Specific Peer

```
module_advanced_transform: %{
  enabled: true,
  rules: [
    %{
      rule_name: "strip_origin_host_from_partner_answers",
      match: :all,
      filters: [
        {:packet_type, :answer},
        {:from_peer, "hss01.partner.example.com"},
        {:command_code, 318}
      ],
      transform: %{
        action: :remove,
        avps: [{:avp, {264, :any}}]
      }
    }
  ]
}
```

How it works: The `:packet_type` filter limits the rule to **answers**, `:from_peer` limits it to answers received from `hss01.partner.example.com`, and `:command_code` limits it to AIA (`318`). When all three match, the `:remove` action deletes the `Origin-Host` (AVP `264`) regardless of its value — removal matches on AVP code only.

Use case: Removing an AVP that a partner network populates incorrectly, or that should not be forwarded downstream, from its answers.

Example 4: Overwrite a Grouped AVP on S6a Requests

```
module_advanced_transform: %{
  enabled: true,
  rules: [
    %{
      rule_name: "overwrite_user_name_s6a",
      match: :all,
      filters: [
        {:packet_type, :request},
        {:application_id, 16_777_251},
        {:command_code, 318}
      ],
      transform: %{
        action: :overwrite,
        dictionary: :diameter_gen_3gpp_s6a,
        avps: [{:avp, {1, "999999000000002"}}]
      }
    }
  ]
}
```

How it works: The rule matches S6a (`16777251`) AIR (`318`) **requests**. The `:overwrite` action decodes the message with the `:diameter_gen_3gpp_s6a` dictionary, rewrites the named AVP(s), and re-encodes it. Because the full message is decoded, `:overwrite` can reach AVPs nested inside grouped/record structures that `:edit` cannot touch.

Use case: Correcting an AVP that lives inside a grouped AVP — for example a field within a subscription-data or vendor-specific group — where a simple `:edit` would not match.

Reference Tables

Application IDs

ID	Interface	Description	Reference
16777251	S6a/S6d	MME/SGSN ↔ HSS authentication and subscription	3GPP TS 29.272

Command Codes

Code	Command	Interface	Reference
318	Authentication-Information-Request/Answer (AIR/AIA)	S6a/S6d	3GPP TS 29.272 §7.2.5
316	Update-Location-Request/Answer (ULR/ULA)	S6a/S6d	3GPP TS 29.272 §7.2.3

Common AVP Codes

Code	AVP	Description	Reference
1	User-Name	Subscriber identity (IMSI on S6a).	3GPP TS 29.272 / RFC 6733 §8.14
264	Origin-Host	Diameter Identity of the message originator.	RFC 6733 §6.3
283	Destination-Realm	Realm the message is destined for; used for routing.	RFC 6733 §6.6
296	Origin-Realm	Realm of the message originator.	RFC 6733 §6.4

Best Practices

1. **Specificity** — Order rules from most specific to most general. The first matching rule wins.
2. **Performance** — Place the most commonly matched rules first to reduce evaluation overhead.
3. **Testing** — Validate regular-expression patterns before deployment.
4. **Naming** — Use descriptive `rule_name` values for operational clarity and monitoring.
5. **Monitoring** — Track rule match rates to confirm rules behave as expected.

Related Documentation

- Advanced Routing — Destination selection using the same rule-processing model.

DRA Architecture Overview

OmniDRA is a **Diameter Routing Agent (DRA)** that sits at the centre of a Diameter signalling network, relaying request and answer messages between clients such as MME/SGSN nodes and back-end services such as the HSS, PCRF, and OCS. It is built in Elixir/Erlang on the OTP `:diameter` library and operates as a **Diameter Relay**, forwarding messages between peers without terminating the applications they carry.

Table of Contents

What the DRA Is

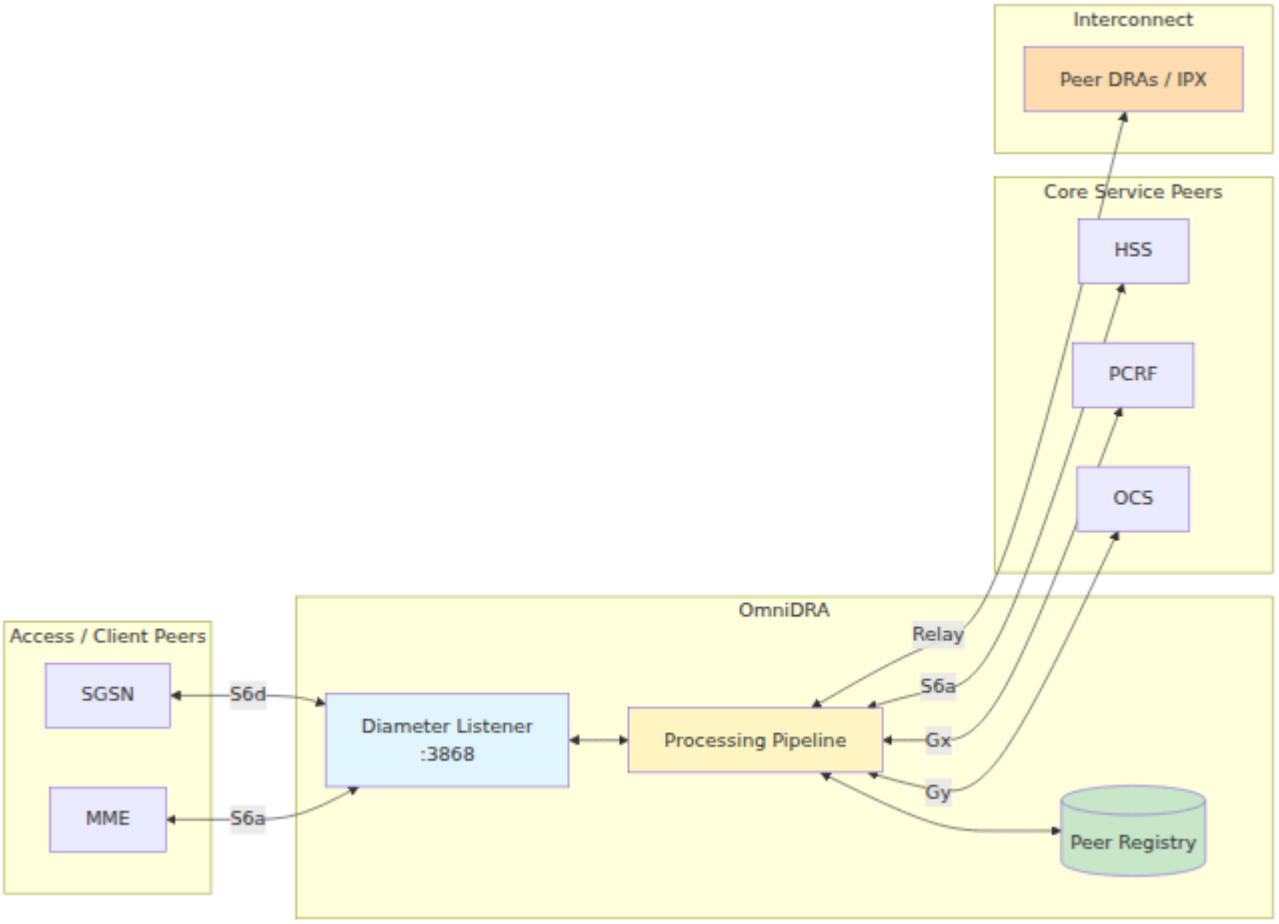
A **Diameter Relay** receives Diameter messages on a listening socket, decides which connected peer should handle each message, and forwards it unchanged (aside from the routing AVPs the relay itself manages). Because OmniDRA runs as a relay, it does not implement the application logic of the interfaces it carries — it can route **S6a**, **Gx**, **Gy**, and any other Diameter application transparently, as long as the appropriate peers advertise support for them.

Key characteristics:

- **Standards-based routing** — Request routing follows the priority mechanism of the Diameter Base Protocol (RFC 6733), using **Destination-Host**, **Destination-Realm**, and **Application-Id** to select a peer.
- **Stateful answer correlation** — Answers always return along the reverse path of their request, correlated by the **Hop-by-Hop Identifier**.
- **Optional processing modules** — Roaming steering, subscriber lookup, advanced routing, and transform modules extend the standard behaviour without replacing it.
- **Security pipeline** — Rate limiting, Diameter firewalling, AVP sanitisation, and topology hiding protect the network at the signalling edge.

Network Context

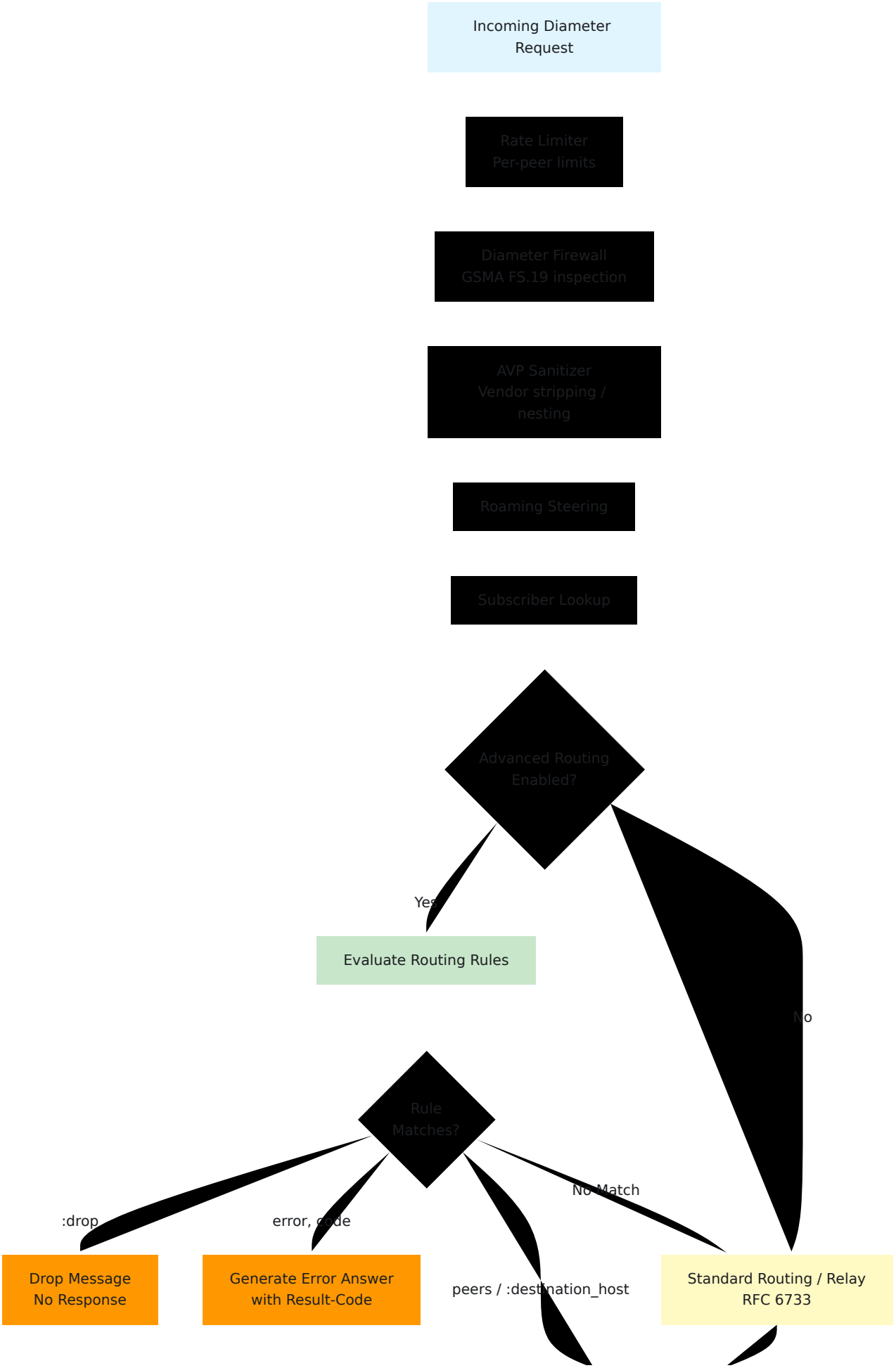
The DRA is deployed between the **access-side** signalling nodes (such as MME and SGSN) and the **core-side** services (such as HSS, PCRF, and OCS). Every peer establishes a Diameter connection to the DRA; the DRA never originates application traffic of its own.



The DRA listens for incoming Diameter connections on TCP/SCTP port `3868` (the standard port defined by RFC 6733). Peers connecting to the DRA advertise the realms and applications they support during **Capabilities Exchange** (CER/CEA), and the DRA records this in its peer registry for use in routing decisions.

Request-Processing Pipeline

Each incoming request passes through an ordered pipeline. Security and steering stages run first, followed by the routing stages. Answers bypass this pipeline entirely — see Message Relaying.


Incoming Diameter Request
Rate Limiter Per-peer limits
Diameter Firewall GSMA FS.19 inspection
AVP Sanitizer Vendor stripping / nesting
Roaming Steering
Subscriber Lookup
Advanced Routing Enabled?
Yes
Evaluate Routing Rules
No
Rule Matches?
:drop
error, code
No Match
peers / :destination_host
Drop Message No Response
Generate Error Answer with Result-Code
Standard Routing / Relay RFC 6733

Pipeline Stages

Stage	Purpose
Rate Limiter	Enforces per-peer request rate limits to protect downstream services from overload.
Diameter Firewall	Inspects requests against GSMA FS.19 low-layer and Category 1/2/3 checks. See Security.
AVP Sanitizer	Strips disallowed vendor AVPs and enforces AVP nesting-depth limits.
Roaming Steering	Applies operator-defined steering to influence which network a roaming subscriber is directed to.
Subscriber Lookup	Resolves subscriber-specific routing context prior to rule evaluation.
Advanced Routing	Evaluates rule-based routing that can drop, error, or direct a request to specific peers. See Advanced Routing.
Standard Routing / Relay	Falls back to RFC 6733 priority routing when no advanced rule applies. See Standard Routing.

Stage	Purpose
Rx Session Binding	Redirects an Rx request to the PCRF holding its session. The only stage that runs *after* Advanced Routing, because an Rx `Session-Termination-Request` carries nothing else to route on. See Rx Session Binding.
Topology Hiding	Rewrites or strips identifying AVPs (Route-Record, Origin-Host/Realm) on outbound messages. See Security.

When Advanced Routing produces a `:drop` outcome, the request is discarded with no response. An `error` outcome generates a Diameter answer carrying the configured **Result-Code**. Otherwise, matched requests are forwarded to the specified peers or the **Destination-Host** peer, and unmatched requests fall through to standard routing.

Rx Session Binding is the one exception to Advanced Routing having the last word, and it is deliberately narrow: it only ever redirects a request that was already going to be relayed, and only for Rx. A `:drop` or `error` outcome, and an explicit `Destination-Host`, are all left untouched.

Standard Routing Priority

When a request reaches standard routing, the DRA selects a peer using the priority mechanism of RFC 6733 §6.1:

1. **Destination-Host AVP (293)** — If present, the DRA routes directly to the named peer. This is the highest-priority mechanism; if that peer is not connected, routing fails.
2. **Destination-Realm AVP (283)** — If Destination-Host is absent, the DRA selects a connected peer advertising the target realm, load-balancing across matching peers.
3. **Application-Id** — Candidate peers are further filtered to those advertising support for the message's application, as learned during Capabilities Exchange.

The full standard routing behaviour, including the peer-selection algorithm, is documented in Standard Routing.

Peer Registry

The DRA maintains an in-memory **peer registry** that tracks every peer connection and the capabilities it advertised during Capabilities Exchange. The registry is the authoritative source for routing decisions: only peers in the **connected** state are eligible to receive traffic, and disconnected or down peers are automatically excluded.

For each connected peer, the registry records:

- The peer's **Diameter Identity** (Origin-Host / FQDN).
- The **realms** the peer advertises.
- The **Application-Ids** the peer supports.
- The current **connection state**.

Because realm and application support are learned dynamically from CER/CEA, adding capacity for a given interface is a matter of connecting an additional peer that advertises the relevant realm and application — the DRA begins load-balancing to it automatically.

Message Relaying

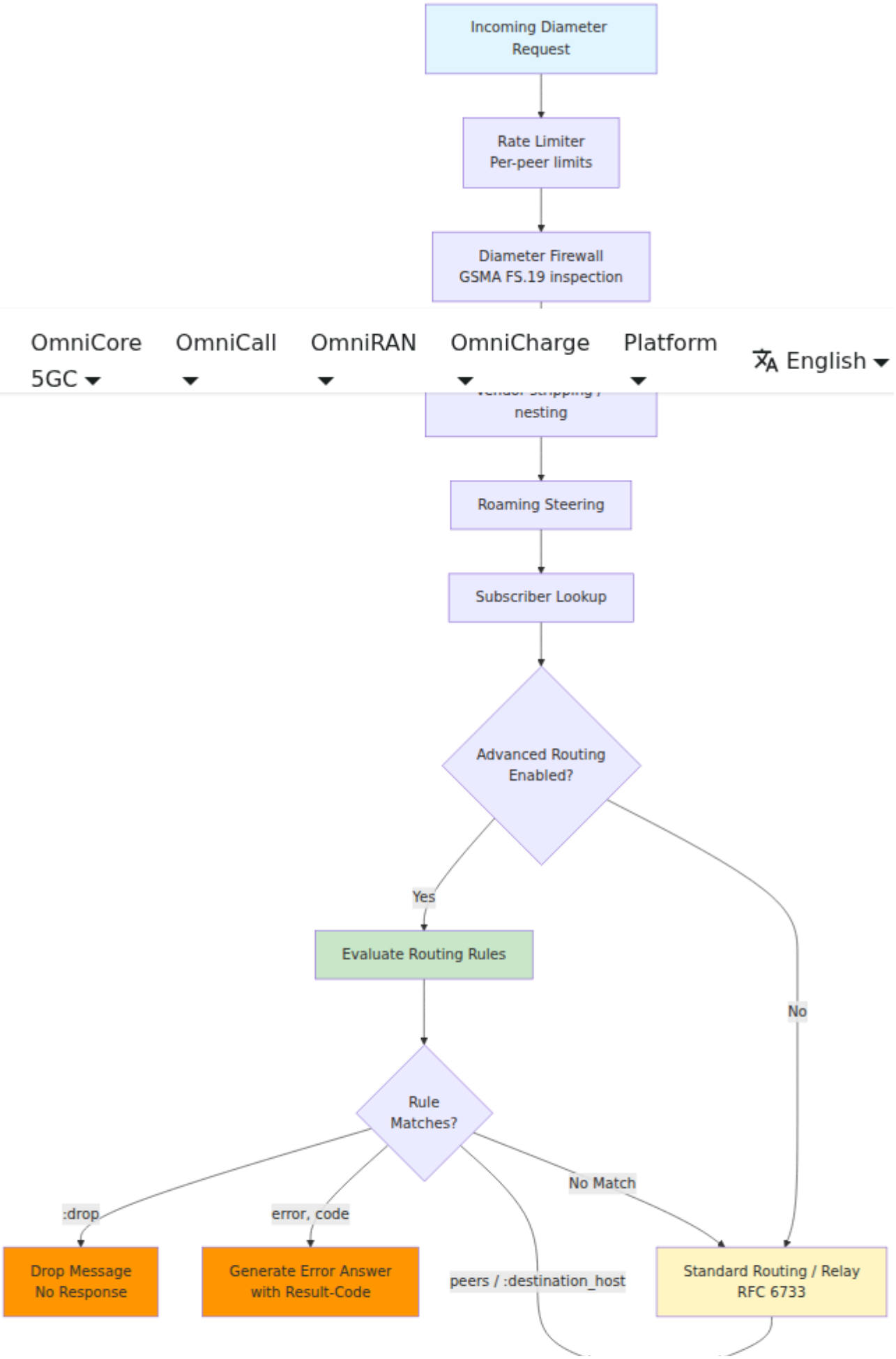
The DRA relays two fundamentally different classes of message, and treats them differently.

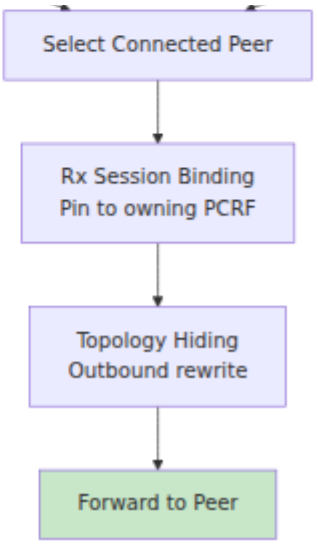
Requests

Requests traverse the full processing pipeline. The DRA selects a destination peer, updates the routing AVPs it manages, and forwards the message. Because the DRA is a relay, the application payload is passed through unchanged.

Answers

Answers never re-enter the routing pipeline. Instead, the DRA correlates each answer to its originating request and returns it along the reverse path.


Incoming Diameter Request
Rate Limiter Per-peer limits
Diameter Firewall GSMA FS.19 inspection
nesting
Roaming Steering
Subscriber Lookup
Advanced Routing Enabled?
Yes
No
Evaluate Routing Rules
Rule Matches?
:drop
error, code
No Match
peers / :destination_host
Drop Message No Response
Generate Error Answer with Result-Code
Standard Routing / Relay RFC 6733



The relaying rules for answers are:

- **Session-based routing** — Answers always follow the reverse path of their request.
- **End-to-End ID preservation** — The **End-to-End Identifier** remains unchanged across every hop.
- **Hop-by-Hop routing** — The **Hop-by-Hop Identifier** (which changes at each hop) is used to maintain routing state and match an answer back to the peer that sent the request.
- **No rule evaluation** — The DRA does not evaluate routing rules or inspect AVP contents for answers.
- **Stateful correlation** — Internal routing tables track which peer sent each request, and this state is cleaned up once the answer is relayed.

Answers are handled this way because answer routing is deterministic: the Diameter protocol requires an answer to return to the originating peer along the established request path. This preserves correct session management and prevents routing loops. See RFC 6733 §6.2 for answer message routing details.

Reference Tables

Common 3GPP Application IDs

ID	Interface	Description	Reference
16777251	S6a/S6d	MME/SGSN ↔ HSS authentication and subscription	3GPP TS 29.272
16777238	Gx	PCEF ↔ PCRF policy and charging control	3GPP TS 29.212
4	Gy	Online charging (credit control) ↔ OCS	3GPP TS 32.299

Common Routing AVPs

Code	Name	Description	Reference
293	Destination-Host	Explicit host-level routing target (highest priority)	RFC 6733 §6.5
283	Destination-Realm	Realm used to select a connected peer	RFC 6733 §6.4
282	Route-Record	Records the identity of relays a message has traversed	RFC 6733 §6.7.1

Further Reading

- **Standard Routing** — RFC 6733 priority routing, peer selection, and answer routing in detail.

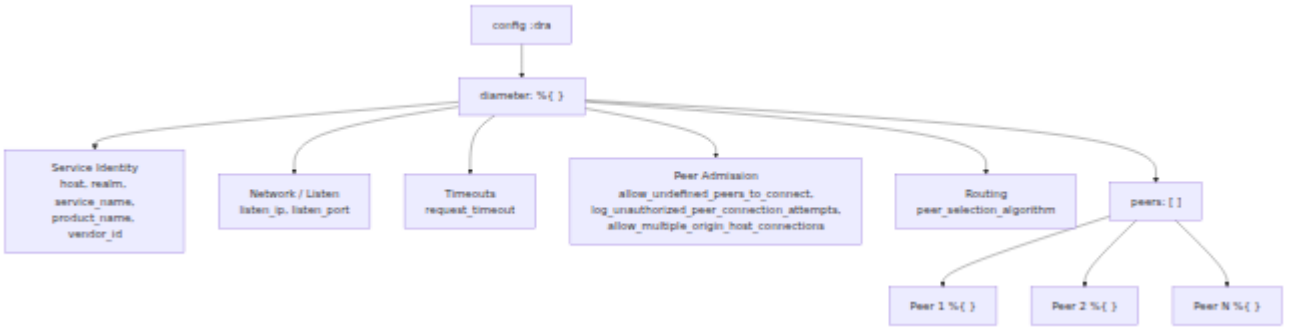
- **Advanced Routing** — Rule-based routing, drop/error outcomes, and peer targeting.
- **Configuration** — Base DRA configuration, transports, and module settings.
- **Security** — Rate limiting, GSMA FS.19 Diameter firewall, AVP sanitisation, and topology hiding.

Base DRA Configuration

The **Base DRA Configuration** defines the OmniDRA's identity, network listeners, timeouts, peer-admission policy, and the list of Diameter peers it connects to. This block is the foundation for every routing decision the agent makes and is declared under `config :dra, diameter: %{...}` in `config/runtime.exs`.

Overview

All base settings live in a single `diameter` map. The map contains a set of top-level keys that describe the DRA itself, followed by a `peers` list where each entry describes one Diameter peer.



Configuration Structure

```
config :dra,
  diameter: %{
    service_name: :omnitouch_dra,
    listen_ip: "127.0.0.2",
    listen_port: 3868,
    host: "dra01",
    realm: "production.omnitouch",
    product_name: "OmniDRA",
    vendor_id: 10415,
    request_timeout: 5000,
    allow_undefined_peers_to_connect: false,
    log_unauthorized_peer_connection_attempts: true,
    allow_multiple_origin_host_connections: false,
    peer_selection_algorithm: :random,
    peers: [
      # Peer configurations...
    ]
  }
```

Top-Level Parameters

These keys configure the DRA's own identity, its network listener, timeouts, and its peer-admission and peer-selection policy. Every key appears exactly once in the `diameter` map.

Parameter	Type	Required	Def
`service_name`	Atom	Yes	-
`listen_ip`	String or List	Yes	-
`listen_port`	Integer	Yes	-
`host`	String	Yes	-
`realm`	String	Yes	-
`product_name`	String	Yes	-
`vendor_id`	Integer	Yes	-

Parameter	Type	Required	Def
`request_timeout`	Integer	No	`500`
`allow_undefined_peers_to_connect`	Boolean	Yes	`fal`
`log_unauthorized_peer_connection_attempts`	Boolean	Yes	`tru`
`allow_multiple_origin_host_connections`	Boolean	Yes	`fal`

Parameter	Type	Required	Def
`peer_selection_algorithm`	Atom	Yes	-
`peer_settle_time_ms`	Integer	No	`0`
`watchdog_timer_ms`	Integer	No	`300`
`peers`	List	Yes	-

SCTP Socket Buffer Size

The socket buffer size is a **top-level** `:dra` key, configured with `config :dra, diameter_socket_buffer_size:` - it is **not** a member of the `diameter:` map.

```
config :dra, diameter_socket_buffer_size: 4_194_304

config :dra,
  diameter: %{
    # ... base configuration ...
  }
```

Parameter	Type	Required	Default	
`diameter_socket_buffer_size`	Integer	No	`4194304` (4 MB)	SCTP soc size in by key (`con` `diameter` not a me map. On `:diamete` Multihom

Peer Selection

When more than one configured peer matches the routing criteria for a request, `peer_selection_algorithm` determines which peer receives it. This directly affects how the DRA distributes traffic and behaves during peer outages; see Standard Routing for the full routing decision flow, and Advanced Routing for rule-based overrides.

Value	Behaviour
`:random`	A matching peer is chosen at random for each request, spreading load across all eligible peers.
`:failover`	The first matching peer is preferred; subsequent peers are used only when higher-priority peers are unavailable.

Peer Configuration

Each entry in the `peers` list is a map describing a single Diameter peer - its identity, its network address, the transport it uses, and whether the DRA dials the peer or waits to be dialled.

```
%{
  host: "diameter-benchmark-server.production.omnitouch",
  realm: "production.omnitouch",
  ip: "127.0.0.3",
  port: 3868,
  transport: :diameter_tcp,
  initiate_connection: true
}
```

Per-Peer Parameters

Parameter	Type	Required	Default	Description
`host`	String	Yes	-	The peer's Diameter Identity (FQDN This value mus match exactly (case-sensitive for routing and for Advanced Routing rules t apply.
`realm`	String	Yes	-	The peer's Diameter realn
`ip`	String	Yes	-	The peer's primary IP address used f the connection
`additional_ips`	List	No	-	List of IP addresses for a multi-homed SCTP association; fa back to `ip` when unset. Only meaningf with `transport :diameter_sct` See SCTP Multihoming.

Parameter	Type	Required	Default	Description
`port`	Integer	Yes	-	The peer's Diameter port (typically `3868`
`transport`	Atom	Yes	-	Transport protocol for the connection: `:diameter_tcp` for TCP, or `:diameter_sct` for SCTP. See the Transports reference belo
`source_ip`	String	No	`listen_ip`	Local IP addres to bind when t DRA dials this peer (outboun connections only). Defaults to the service `listen_ip`.
`source_port`	Integer	No	-	Local port to bind when the DRA dials this peer (outboun connections only). When unset, the OS assigns an ephemeral por Set an explicit value when tw peers share th

Parameter	Type	Required	Default	Description
				same destination IP and port, so each connectio uses a distinct local socket.
`initiate_connection`	Boolean	Yes	-	When `true`, th DRA connects out to the peer (client role). When `false`, the DRA waits for the peer to connect (serve role). See Connection Modes.

> **SCTP multihoming:** peers using `:diameter_sctp` can bind multiple local and remote addresses for resilience. Multihoming settings and per-peer address lists are covered in SCTP Multihoming.

Transports

Transport value	Protocol	Reference
`:diameter_tcp`	TCP	RFC 6733 (Diameter Base)
`:diameter_sctp`	SCTP	RFC 4960 (SCTP) - see SCTP Multihoming

Connection Modes

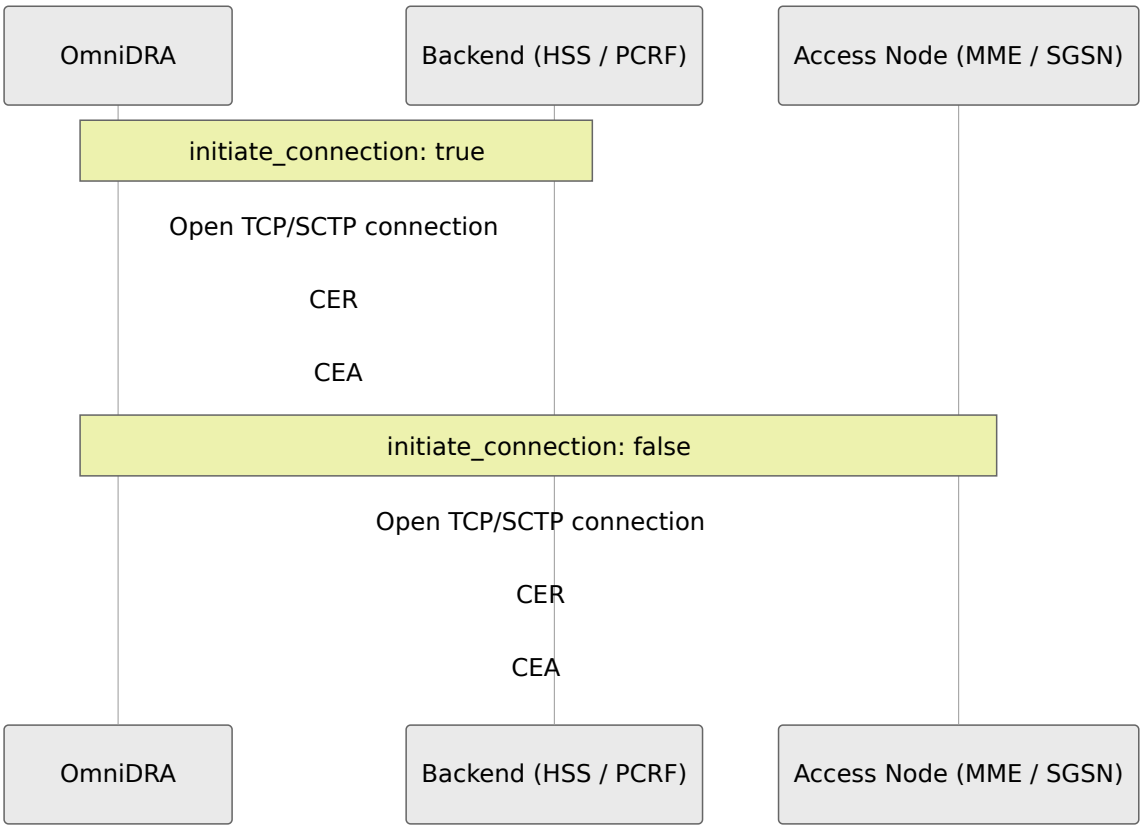
Whether the DRA dials a peer or waits to be dialled is controlled by `initiate_connection`.

Initiate Connection (`initiate_connection: true`)

- The DRA acts as the Diameter client and initiates the TCP/SCTP connection to the peer.
- Typically used for backend systems such as an HSS or PCRF.
- The DRA will retry the connection if the peer is unreachable.

Accept Connection (`initiate_connection: false`)

- The DRA acts as the Diameter server and waits for the peer to connect.
- Typically used for access-side nodes such as an MME, SGSN, or P-GW.
- The peer must be present in `peers`, unless `allow_undefined_peers_to_connect` is set to `true`.
- Inbound peers are differentiated by the `Origin-Host` they advertise in the CER, so two Diameter nodes sharing one source IP are matched independently as long as each presents a distinct `host`.



Complete Example

The following example configures a DRA that accepts connections from an MME over SCTP and dials an HSS and a PCRF over TCP.

```
config :dra,
  diameter: %{
    service_name: :mvno_dra,
    listen_ip: "10.100.1.10",
    listen_port: 3868,
    host: "dra01",
    realm: "mvno.example.com",
    product_name: "OmniDRA",
    vendor_id: 10415,
    request_timeout: 5000,
    allow_undefined_peers_to_connect: false,
    log_unauthorized_peer_connection_attempts: true,
    allow_multiple_origin_host_connections: false,
    peer_selection_algorithm: :random,
    peers: [
      # MME - DRA waits for the MME to connect
      %{
        host: "mme01.operator.example.com",
        realm: "operator.example.com",
        ip: "10.100.2.15",
        port: 3868,
        transport: :diameter_sctp,
        initiate_connection: false
      },
      # HSS - DRA initiates the connection from an explicit local
source port
      %{
        host: "hss01.mvno.example.com",
        realm: "mvno.example.com",
        ip: "10.100.3.141",
        port: 3868,
        source_port: 13868,
        transport: :diameter_tcp,
        initiate_connection: true
      },
      # PCRF - DRA initiates the connection
      %{
        host: "pcrf01.mvno.example.com",
        realm: "mvno.example.com",
        ip: "10.100.3.22",
        port: 3868,
        transport: :diameter_tcp,
        initiate_connection: true
```

```
      }
    ]
  }
```

Operational Notes

- **Hostname matching** - Peer `host` values are matched exactly and case-sensitively. Rules in Advanced Routing must reference the same `host` string configured here.
- **Capabilities exchange** - On connection, the DRA and each peer exchange supported applications and identities via CER/CEA. The `host`, `realm`, `product_name`, and `vendor_id` values are advertised at this stage.
- **Vendor-ID 10415** - This is the standard 3GPP `Vendor-Id` and is the expected value for 3GPP Diameter interfaces.
- **Peer selection** - When multiple peers match a request, `peer_selection_algorithm` decides which one is used; see Standard Routing.

Security Considerations

- Set `allow_undefined_peers_to_connect: false` in production so that only known peers may connect.
- Enable `log_unauthorized_peer_connection_attempts: true` to record rejected connection attempts for security monitoring.
- Ensure firewall rules align with the configured `listen_ip` and `listen_port`.

Related Documentation

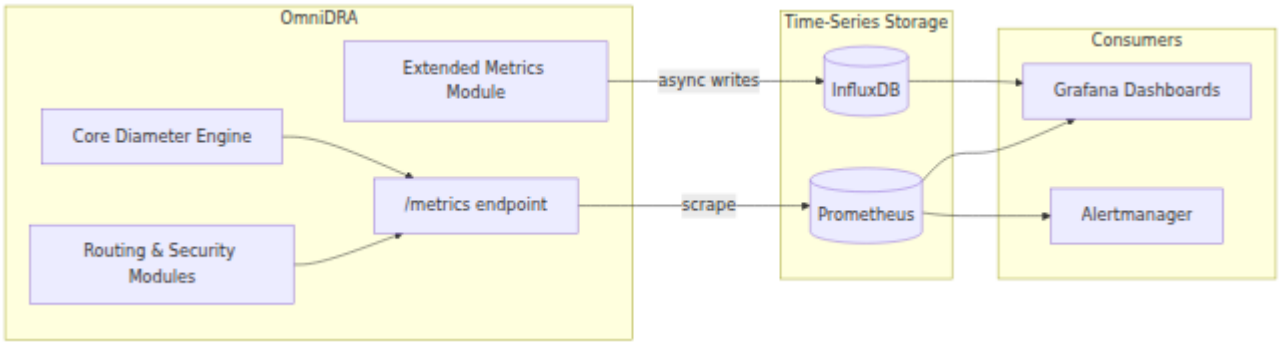
- SCTP Multihoming - Multihomed SCTP transport and per-peer address configuration.
- Standard Routing - How the DRA selects peers and routes requests, including `peer_selection_algorithm` behaviour.
- Advanced Routing - Rule-based routing overrides keyed on peer `host` values.

Metrics & Monitoring

OmniDRA exposes comprehensive **Prometheus** metrics covering peer connectivity, message rates, success and error outcomes, response latency, dropped traffic, and unauthorized connection attempts. This page documents every metric, its labels, and ready-to-use PromQL queries for building dashboards and alerts.

Observability Flow

All metrics are published in Prometheus exposition format at the `/metrics` endpoint. Prometheus scrapes this endpoint on a regular interval, stores the samples, and dashboards or alerting tools query the resulting time series.



Standard Prometheus metrics are scraped from `/metrics`. The **Extended Metrics Module** publishes high-cardinality, subscriber-level telemetry directly to **InfluxDB** to keep the Prometheus series count manageable (see Extended Metrics Module).

> **BEAM/Erlang runtime metrics** are also exposed on `/metrics` but are not documented here. All counters are cumulative and should be queried with `rate()` to obtain per-second rates.

Core Diameter Metrics

These metrics are always available and describe peer health and message flow through the DRA.

Peer Connectivity

Peer Status

- **Metric**: `diameter_peer_status`
- **Type**: Gauge
- **Description**: Whether the peer is connected (`1`) or not (`0`).

Labels:

Label	Description
`origin_host`	Peer's Diameter Identity (FQDN).
`ip`	Peer's IP address.

Example queries:

```
# Check if specific peer is connected
diameter_peer_status{origin_host="hss01.example.com"}

# Count disconnected peers
count(diameter_peer_status == 0)
```

For diagnosing peers that flap or never reach a connected state, see Troubleshooting.

Supported Applications

- **Metric**: `diameter_peer_supported_applications`
- **Type**: Gauge
- **Description**: The applications a peer advertised during capabilities exchange (CER/CEA), when known.

Labels:

Label	Description
`origin_host`	Peer's Diameter Identity (FQDN).
`supported_applications`	The applications the peer advertised as supported.

Example queries:

```
# Applications advertised by a specific peer
diameter_peer_supported_applications{origin_host="hss01.example.com"}
```

Message Rates

Message Count

- **Metric**: `diameter_peer_message_count_total`
- **Type**: Counter
- **Description**: Total number of Diameter messages exchanged with peers.

Labels:

Label	Description
`origin_host`	Peer's Diameter Identity.
`received_from`	Peer the message was received from.
`application_id`	Diameter Application-Id. See Application IDs.
`cmd_code`	Diameter Command-Code. See Command Codes.
`application_name`	Human-readable application name (e.g. `3GPP_S6a`).
`cmd_name`	Human-readable command name (e.g. `AIR`).
`direction`	`request` or `response`.

Example queries:

```
# S6a AIR request rate from specific MME
rate(diameter_peer_message_count_total{
  cmd_code="318",
  direction="request",
  origin_host="mme01.example.com"
}[5m])

# Total message rate by application
sum by (application_name)
(rate(diameter_peer_message_count_total[5m]))
```

Success & Error Rates

Response Result Codes

- **Metric**: `diameter_peer_message_result_code_count_total`
- **Type**: Counter
- **Description**: Total number of Diameter responses by result code.

Labels:

Label	Description
`origin_host`	Original requester.
`routed_to`	Peer that sent the answer.
`application_id`	Diameter Application-Id.
`cmd_code`	Diameter Command-Code.
`application_name`	Application name.
`cmd_name`	Command name.
`result_code`	Diameter Result-Code or Experimental-Result-Code.

Example queries:

```
# Success rate for S6a AIR requests
rate(diameter_peer_message_result_code_count_total{
  cmd_code="318",
  result_code="2001"
}[5m])

# Error rate by result code
sum by (result_code) (
  rate(diameter_peer_message_result_code_count_total{
    result_code!="2001"
  }[5m])
)
```

Common Result Codes:

Code	Name	Reference
`2001`	DIAMETER_SUCCESS	RFC 6733 §7.1.1
`3002`	DIAMETER_UNABLE_TO_DELIVER	RFC 6733 §7.1.3
`3003`	DIAMETER_REALM_NOT_SERVED	RFC 6733 §7.1.3
`3004`	DIAMETER_TOO_BUSY	RFC 6733 §7.1.3
`5001`	DIAMETER_AUTHENTICATION_REJECTED	RFC 6733 §7.1.5
`5004`	DIAMETER_INVALID_AVP_VALUE	RFC 6733 §7.1.5
`5012`	DIAMETER_UNABLE_TO_COMPLY	RFC 6733 §7.1.5

Latency

Response Delay

- **Metric**: `diameter_peer_last_response_delay`
- **Type**: Gauge
- **Description**: Most recent response delay in milliseconds (DRA → Peer → DRA).

Labels:

Label	Description
`origin_host`	Original requester.
`routed_to`	Peer that sent the answer.
`application_name`	Application name.
`cmd_name`	Command name.

Example queries:

```
# Most recent response delay from HSS
diameter_peer_last_response_delay{routed_to="hss01.example.com"}

# Highest recent response delay across S6a peers
max(diameter_peer_last_response_delay{application_name="3GPP_S6a"})
```

> This is a gauge holding the most recent delay per peer, not a histogram, so quantile functions such as `histogram_quantile` do not apply.

Unanswered Requests

Unanswered Request Count

- **Metric**: `diameter_peer_unanswered_request_count_total`
- **Type**: Counter
- **Description**: Requests sent but not answered within the timeout period.

Labels:

Label	Description
`origin_host`	Original requester.
`routed_to`	Peer that didn't answer.
`application_id`	Diameter Application-Id.
`cmd_code`	Diameter Command-Code.
`application_name`	Application name.
`cmd_name`	Command name.

Example queries:

```
# Unanswered request rate
rate(diameter_peer_unanswered_request_count_total[5m])

# Identify problematic peers
topk(5, sum by (routed_to) (
  rate(diameter_peer_unanswered_request_count_total[5m])
))
```

Unauthorized Connections

Unauthorized Connection Attempts

- **Metric**: `diameter_peer_unauthorized_connection_count_total`
- **Type**: Counter
- **Description**: Connection attempts from unauthorized peers.

Labels:

Label	Description
`origin_host`	Unauthorized peer's claimed identity.
`supported_applications`	Applications advertised by the peer.
`peer_ip`	IP address of the connection attempt.

Example queries:

```
# Unauthorized connection attempts
rate(diameter_peer_unauthorized_connection_count_total[5m])

# Alert on unauthorized access
diameter_peer_unauthorized_connection_count_total > 0
```

For deeper detail on security-module metrics (firewall, rate limiting, topology hiding, AVP sanitization), see Security.

Advanced Routing Module Metrics

These metrics describe outcomes produced by advanced routing rules. See the security module documentation in Security for related traffic-control telemetry.

Dropped Traffic

Drop Count

- **Metric**: `diameter_advanced_routing_drop_count_total`
- **Type**: Counter
- **Description**: Requests dropped by the advanced routing `:drop` action (no response sent).

Labels:

Label	Description
`application_id`	Diameter Application-Id.
`cmd_code`	Diameter Command-Code.
`application_name`	Application name.
`cmd_name`	Command name.

Example queries:

```
# Drop rate by command
sum by (cmd_name) (
  rate(diameter_advanced_routing_drop_count_total[5m])
)

# Total dropped traffic
sum(rate(diameter_advanced_routing_drop_count_total[5m]))
```

When traffic is dropped:

- An advanced routing rule matches with a **drop** action.
- The message is silently discarded.
- No answer message is sent to the requesting peer.

Rate Limiting & Overload

Error Responses

- **Metric**: `diameter_advanced_routing_error_count_total`
- **Type**: Counter
- **Description**: Error responses generated by the advanced routing error action.

Labels:

Label	Description
`result_code`	Diameter Result-Code sent in the error response.
`application_id`	Diameter Application-Id.
`cmd_code`	Diameter Command-Code.
`application_name`	Application name.
`cmd_name`	Command name.

Example queries:

```
# Error responses by result code
sum by (result_code) (
  rate(diameter_advanced_routing_error_count_total[5m])
)

# DIAMETER_TOO_BUSY responses
rate(diameter_advanced_routing_error_count_total{
  result_code="3004"
}[5m])

# Rate limiting or overload detection
diameter_advanced_routing_error_count_total{
  result_code="3004"
} > 100
```

When error responses are generated:

- An advanced routing rule matches with an **error** action for a specified Result-Code.
- The DRA generates a Diameter answer with the specified Result-Code.
- The answer is sent back to the requesting peer (not forwarded).

Common Error Codes Used:

Code	Name	Meaning
`3002`	DIAMETER_UNABLE_TO_DELIVER	Routing failure.
`3003`	DIAMETER_REALM_NOT_SERVED	Realm not supported.
`3004`	DIAMETER_TOO_BUSY	Overload protection.
`5012`	DIAMETER_UNABLE_TO_COMPLY	General error.

Extended Metrics Module

The **Extended Metrics Module** provides advanced telemetry and analytics for analyzing Diameter traffic patterns beyond the standard metrics. It currently tracks LTE subscriber attach attempts by correlating S6a Authentication Information Request (AIR) and Answer (AIA) message pairs.

Because these metrics carry **high-cardinality** fields such as IMSI, they are written asynchronously to **InfluxDB** rather than exposed on the Prometheus `/metrics` endpoint. This keeps the Prometheus series count bounded and avoids metric explosion.

Configuration

The module is disabled by default and configured under `config/runtime.exs`:

```
module_extended_metrics:
  enabled: true
  attach_attempt_reporting_enabled: true
```

Parameter	Type	Required	Default	De
`enabled`	Boolean	No	`false`	Se to the Ex Me Mo Wh `fa` mo co by
`attach_attempt_reporting_enabled`	Boolean	No	`false`	En tra rep LTE att (S(AI

InfluxDB Connection

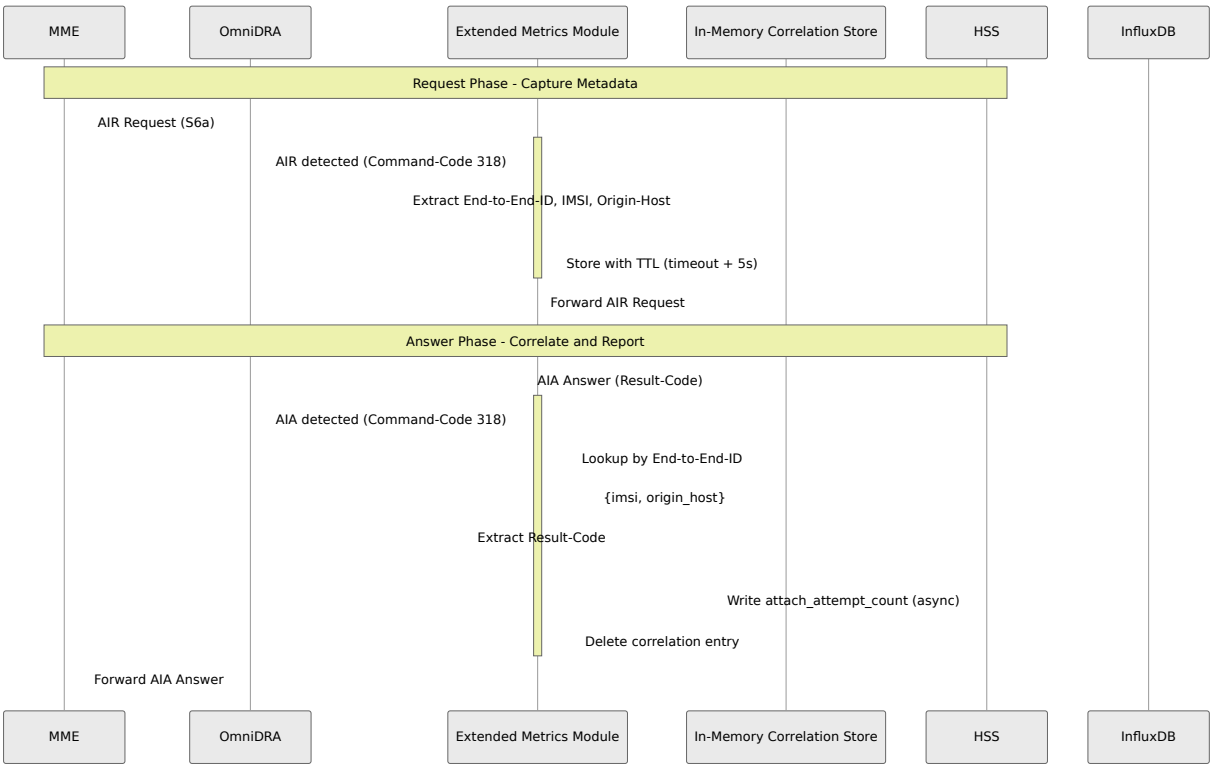
The Extended Metrics Module writes asynchronously to InfluxDB via an Instream connection (Instream); configure the target under `config :dra, DRA.Metrics.InfluxDB`.

```
config :dra, DRA.Metrics.InfluxDB,
  auth: [username: "example_user", password: "example_password"],
  database: "dra",
  host: "127.0.0.15",
  port_udp: 8089,
  writer: Instream.Writer.Line
```

Parameter	Type	Required	Default	Descri
`auth`	Keyword list	No	-	InfluxDB cre `[username: password: .`
`database`	String	Yes	-	Target Influx database th the metrics measureme
`host`	String	Yes	-	Hostname o address of t server.
`port_udp`	Integer	No	`8089`	UDP port us line-protocol InfluxDB. Us InfluxDB's U port (`8089` b not the HTT (`8086`).
`writer`	Module	No	`Instream.Writer.Line`	Instream wr module; `Instream.W` writes using protocol.

Attach Attempt Tracking

The module monitors S6a AIR/AIA pairs, correlating each request with its answer using the Diameter End-to-End-ID, then emits a subscriber-level metric enriched with the originating peer and the HSS result code.



Measurement: `attach_attempt_count`

Fields:

Field	Description
`imsi`	The subscriber IMSI, taken from the User-Name AVP.

Tags:

Tag	Description
`origin_host`	The peer that originated the attach request.
`result_code`	The Diameter result code from the HSS response.

Result codes written to this measurement are standard Diameter codes:

Code	Name
`2001`	DIAMETER_SUCCESS
`5001`	DIAMETER_AUTHENTICATION_REJECTED
`5004`	Diameter AVP unsupported

See RFC 6733 for the complete result code list.

Use Cases

- **Attach Success Rate Analysis** - Track successful versus failed attach attempts by result code.
- **IMSI-Level Troubleshooting** - Identify individual subscribers experiencing attach failures.
- **Network Performance Monitoring** - Monitor attach attempt patterns by origin (MME/SGSN).
- **Roaming Analytics** - Analyze inbound roaming attach success rates.

Operational Notes

- Attach attempt metrics only track S6a AIR/AIA pairs (Application-Id `16777251`, Command-Code `318`).
- Request metadata expires based on the configured request timeout plus 5 seconds.
- Metric processing is asynchronous to avoid blocking message flow.
- The module operates independently from routing and transform modules.

Reference Tables

Application ID Reference

ID	Application	Description	Reference
`16777251`	S6a/S6d	MME/SGSN ↔ HSS authentication and subscription	3GPP TS 29.272
`16777238`	Gx	PCEF ↔ PCRF policy and charging control	3GPP TS 29.212
`16777216`	Cx	I-CSCF/S-CSCF ↔ HSS IMS registration	3GPP TS 29.229

Command Code Reference

Code	Command	Application	Reference
`318`	AIR/AIA (Authentication-Information)	S6a	3GPP TS 29.272

Notes

- All standard metrics are available at the `/metrics` endpoint in Prometheus format.
- The **Extended Metrics Module** provides additional S6a-specific metrics via InfluxDB.
- **BEAM/Erlang runtime metrics** are also exposed but not documented here.
- All counters are cumulative; query them with `rate()` for per-second rates.

- High-cardinality tags (such as IMSI) are used only in the Extended Metrics Module to avoid metric explosion.

Related Documentation

- Troubleshooting - Diagnosing connectivity, routing, and delivery issues.
- Security - Detail on security-module metrics (firewall, rate limiting, topology hiding, AVP sanitization).

Peer Connectivity & Settle Time

Controls how quickly a Diameter peer becomes routable after it connects, and prevents traffic being sent to peers that are flapping (repeatedly connecting and disconnecting).

Background

Before OmniDRA routes to a peer it must complete the capabilities exchange (CER/CEA) and be promoted to service. By default, the RFC 3539 watchdog holds a *reconnecting* peer in the REOPEN state until it has passed successive watchdog (DWR/DWA) exchanges paced by the watchdog timer **Tw** — roughly 30 seconds with the standard 30000 ms default — even though CER/CEA already succeeded.

OmniDRA removes that implicit delay (the watchdog promotes reconnecting peers immediately) and replaces it with a **settle time**: an explicit, configurable hold applied after the peer connects. During the hold the peer is connected but not routable; if it disconnects within the window the hold is cancelled and no traffic is ever sent to it.

Configuration

Both keys are optional and live in the `:diameter` config map. With neither set, peers route as soon as they connect.

```
config :dra,
  diameter: %{
    # ...service settings...
    peer_settle_time_ms: 5000,
    watchdog_timer_ms: 30_000
  }
```

Parameter	Type	Required	Default	Description
`peer_settle_time_ms`	Integer	No	`0`	How long (ms) a newly-connected peer must stay up before traffic is routed to it. `0` routes immediately. A positive value holds the peer for that duration and cancels the hold if it disconnects during the window.
`watchdog_timer_ms`	Integer	No	`30000`	Override for the RFC 3539 watchdog timer **Tw**, the DWR/DWA liveness interval. Lower it for faster dead-peer detection at the cost of more watchdog traffic.

Set the settle time no higher than the worst-case flap interval you need to ride out — a longer value increases flap protection but also delays a healthy peer's return after a legitimate restart.

Monitoring

A peer in its settle window reports **down** in the `diameter_peer_status` gauge (`0` = down/not yet routable, `1` = routable) until it becomes routable, so the metric reflects what can actually be routed.

Come-up produces these log lines:

Log message	When
`peer connected, settling before routable`	Peer connected, settle window started (includes `settle_time_ms`).
`peer disconnected during settle window; never became routable`	Peer dropped before settling; no traffic was routed.
`destination peer is settling; request not yet routable...`	A request arrived for a peer still settling. Answered `DIAMETER_UNABLE_TO_DELIVER (3002)`.
`no suitable peer to route request...`	Request unroutable to a destination that is **not** settling (down, unknown, or misconfigured).

The last two distinguish an expected, transient settle from a genuinely missing destination.

Reference

- RFC 3539 — Diameter watchdog state machine and the watchdog timer (Tw).
- RFC 6733 — Diameter base protocol (CER/CEA, DWR/DWA).

Roaming Steering (SoR)

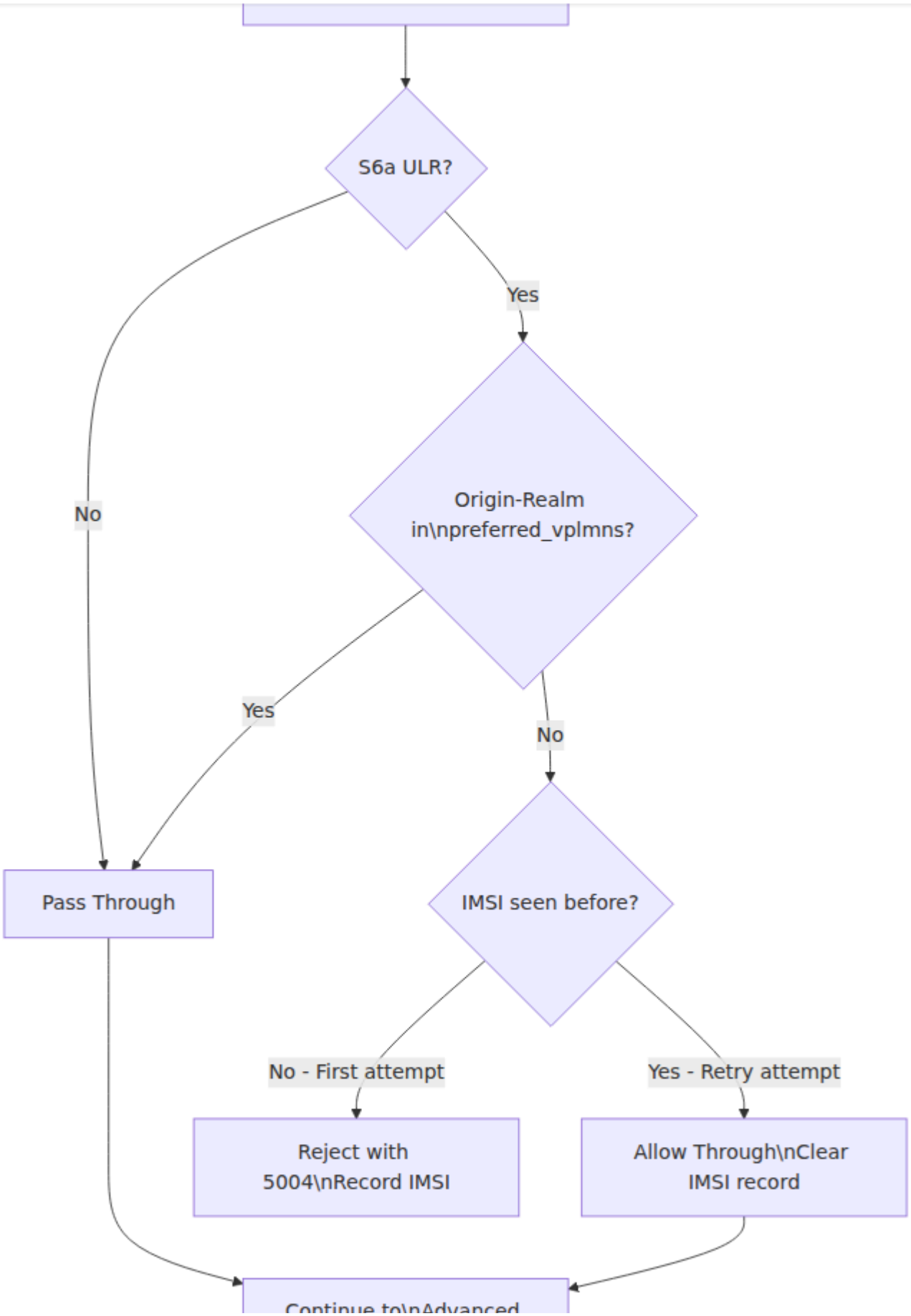
Steering of Roaming (SoR) allows the HPLMN to influence which VPLMN a roaming subscriber attaches to. When a subscriber attempts to register via a non-preferred VPLMN, the DRA intercepts the S6a Update-Location-Request (ULR) and rejects it with **DIAMETER_ERROR_ROAMING_NOT_ALLOWED (5004)**. This causes the UE to detach and re-attach, ideally selecting a preferred VPLMN.

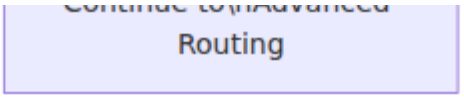
If preferred coverage is unavailable and the subscriber returns to the same non-preferred VPLMN, the module allows the request through on the subsequent attempt.

SoR is defined in 3GPP TS 29.272 Section 5.2.1.1 and 3GPP TS 23.122.

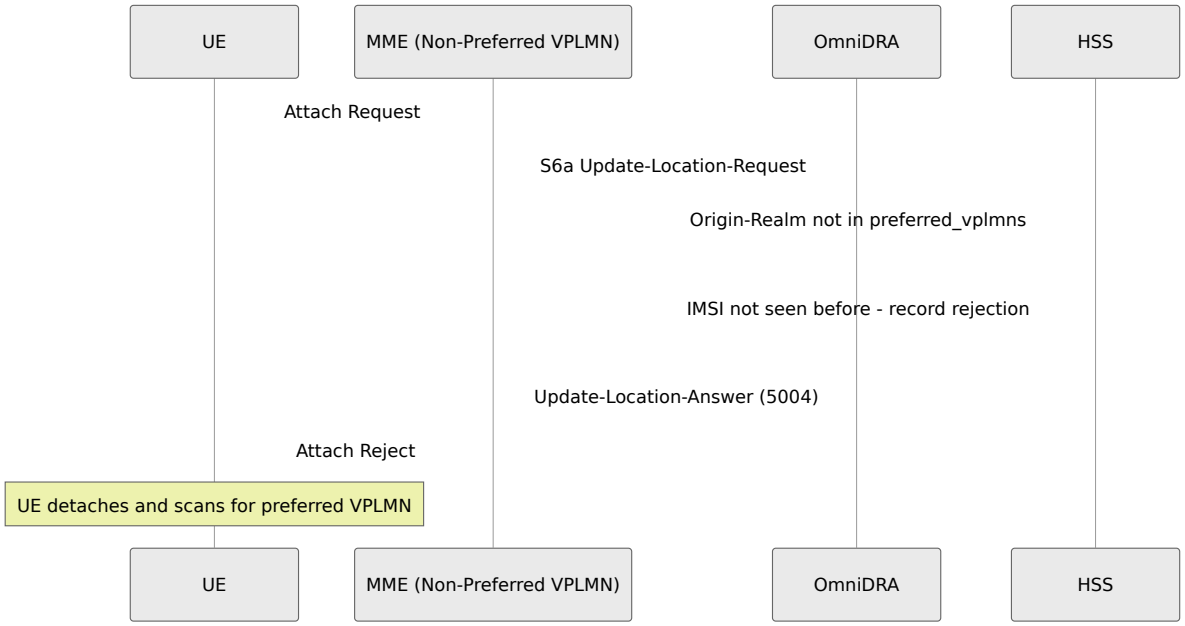
How It Works

The module operates within the DRA request processing pipeline, executing **before** Advanced Routing. Only S6a Update-Location-Requests (Application-Id `16777251`, Command-Code `316`) are evaluated. All other Diameter messages pass through unmodified.

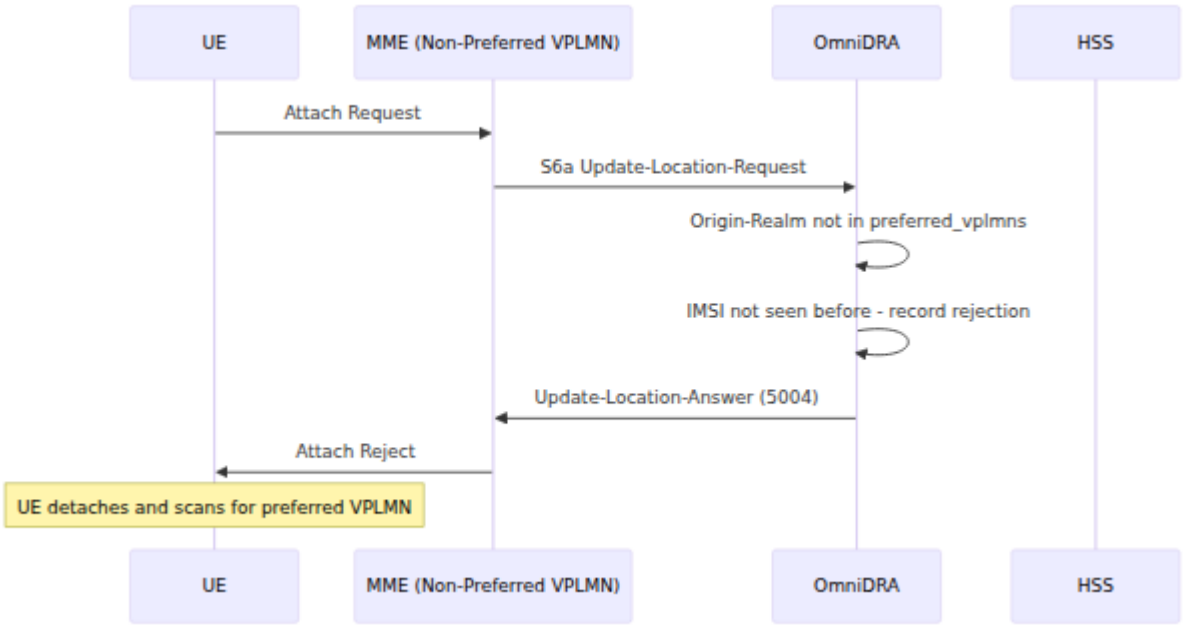

S6a ULR?
Yes
No
Origin-Realm in\npreferred_vplmns?
Yes
No
Pass Through
IMSI seen before?
No - First attempt
Yes - Retry attempt
Reject with 5004\nRecord IMSI
Allow Through\nClear IMSI record
Continue to\nAdvanced



Sequence: First Attempt (Rejected)



Sequence: Second Attempt (Allowed)



Pipeline Position

Roaming Steering runs **before** Advanced Routing in the request processing pipeline. If SoR rejects a request, Advanced Routing is never reached for that message.

Subscriber Tracking

The module tracks each rejected subscriber by IMSI. Each record includes a rejection count and a timestamp. Records are automatically cleaned up after the configured TTL expires.

When a tracked IMSI has reached the `max_rejections` threshold and sends another ULR, the request is allowed through and the tracking record is deleted.

If no further ULR arrives within the TTL window, the record expires and is removed during the next cleanup cycle. A subsequent ULR from the same IMSI will be treated as a first attempt and rejected again.

Configuration

The module is configured under `module_roaming_steering` in `config/runtime.exs`.

```
module_roaming_steering: %{
  # Enable or disable the module
  enabled: false,

  # Number of rejections before allowing through
  max_rejections: 1,

  # Time in seconds before a tracked IMSI record expires
  rejection_ttl_seconds: 300,

  # Diameter Result-Code returned in the rejection answer
  rejection_result_code: 5004,

  # List of Origin-Realm values considered preferred VPLMNs
  # ULRs from these realms are always allowed through
  preferred_vplmns: [
    "epc.mnc001.mcc001.3gppnetwork.org",
    "epc.mnc002.mcc001.3gppnetwork.org"
  ]
}
```

Parameters

Parameter	Type	Required	Default	
`enabled`	Boolean	Yes	`false`	Enable or disab module. When through unmoc
`children`	List	No	`[]`	Processes start when enabled. `[DRA.Router.M` the IMSI-trackir
`max_rejections`	Integer	No	`1`	Number of time before allowing `1`, the first ULR allowed.
`rejection_ttl_seconds`	Integer	No	`300`	Time in second expires and is c does not retry attempt is trea controls the cle
`rejection_result_code`	Integer	No	`5004`	Diameter Resu rejection answe DIAMETER_ERR per 3GPP TS 29
`preferred_vplmns`	List	No	`[]`	List of Origin-R preferred VPLM peers with a m always allowed

Configuration Examples

Reject Once, Then Allow

The default behaviour. Subscribers hitting a non-preferred VPLMN are rejected once. If they return, the ULR is forwarded to the HSS.

```
module_roaming_steering: %{
  enabled: true,
  max_rejections: 1,
  rejection_ttl_seconds: 300,
  rejection_result_code: 5004,
  preferred_vplmns: [
    "epc.mnc001.mcc310.3gppnetwork.org",
    "epc.mnc002.mcc310.3gppnetwork.org"
  ]
}
```

How it works: A ULR arriving from an MME in `epc.mnc003.mcc310.3gppnetwork.org` (not in the preferred list) is rejected with Result-Code 5004. The UE detaches and attempts to find a preferred network. If it returns to the same non-preferred VPLMN within 300 seconds, the ULR is forwarded normally.

Use case: Standard roaming steering where the operator has preferred roaming agreements with specific partner networks.

Multiple Rejections Before Allow

For scenarios where the operator wants the UE to make multiple attempts before falling back to a non-preferred VPLMN.

```
module_roaming_steering: %{
  enabled: true,
  max_rejections: 3,
  rejection_ttl_seconds: 600,
  rejection_result_code: 5004,
  preferred_vplmns: [
    "epc.mnc001.mcc310.3gppnetwork.org"
  ]
}
```

How it works: The subscriber is rejected three times before being allowed through on the fourth attempt. The TTL is extended to 600 seconds to accommodate the additional retry cycles.

Use case: Dense urban areas where preferred coverage exists but may require multiple attach attempts to acquire.

Metrics

Rejections

Metric: `diameter.roaming_steering.reject.count` **Type**: Counter
Description: Incremented each time a ULR is rejected by the SoR module.
Labels:

- `origin_realm` - Origin-Realm of the non-preferred VPLMN
- `result_code` - Diameter Result-Code returned in the rejection
- `imsi` - IMSI of the rejected subscriber

Allowed (Preferred VPLMN)

Metric: `diameter.roaming_steering.allow.count` **Type**: Counter
Description: Incremented when a ULR is allowed through. **Labels**:

- `origin_realm` - Origin-Realm of the originating VPLMN
- `reason` - Why the request was allowed: `preferred_vplmn` or `max_rejections_reached`
- `imsi` - IMSI of the subscriber (present only when reason is `max_rejections_reached`)

Error Responses

Metric: `diameter.roaming_steering.error.count` **Type**: Counter
Description: Incremented when the processor returns an error answer message due to a SoR rejection. **Labels**:

- `result_code` - Diameter Result-Code
- `application_id` - Diameter Application-Id (numeric)
- `cmd_code` - Diameter Command-Code (numeric)
- `application_name` - Human-readable application name
- `cmd_name` - Human-readable command name

Troubleshooting

Subscribers Always Rejected (Never Allowed Through)

Symptoms: Subscribers on non-preferred VPLMNs are repeatedly rejected and never successfully attach.

Possible causes:

- `rejection_ttl_seconds` is too short, causing the tracking record to expire before the subscriber retries
- `max_rejections` is set higher than the number of retries the UE performs before giving up

Resolution:

1. Increase `rejection_ttl_seconds` to accommodate the UE's retry timing
2. Verify `max_rejections` is set to a value the UE can realistically reach within its retry cycle

Subscribers on Preferred VPLMNs Being Rejected

Symptoms: ULRs from preferred partner networks are being rejected with 5004.

Possible causes:

- The Origin-Realm of the preferred VPLMN is not listed in `preferred_vplmns`

- The Origin-Realm string does not exactly match (case-sensitive)

Resolution:

1. Check the peer's Origin-Realm in the DRA logs at connection time
2. Ensure the exact Origin-Realm string is included in the `preferred_vplmns` list

Module Not Taking Effect

Symptoms: ULRs from non-preferred VPLMNs are being forwarded without rejection.

Possible causes:

- `enabled` is set to `false`
- The module process is not running (check supervisor)

Resolution:

1. Verify `enabled: true` in the configuration
2. Confirm the DRA was restarted after the configuration change

Reference

Diameter Codes

Code	Name	Description	Refer
5004	DIAMETER_ERROR_ROAMING_NOT_ALLOWED	Subscriber not permitted to roam in this VPLMN	3GPP 29.27 Sectic 7.4.4

S6a Commands

Command-Code	Name	Description	Reference
316	Update-Location-Request/Answer (ULR/ULA)	Sent by the MME to the HSS during attach to update the subscriber's serving node	3GPP TS 29.272 Section 7.2.3

Application IDs

ID	Interface	Description	Reference
16777251	S6a/S6d	MME/SGSN to HSS authentication and subscription management	3GPP TS 29.272

Rx Session Binding

Rx Session Binding keeps every message of an Rx session on the PCRF that owns it. It exists because Rx is the one interface whose session-terminating request carries nothing to route on.

The Problem

3GPP TS 29.214 §5.6.5 makes `Destination-Host` **optional** in the Rx `Session-Termination-Request`, and defines **no** `Subscription-Id` and **no** `Framed-IP-Address` in the message at all. That second part is the structural bit: those AVPs are not merely absent in practice, they are not in the message definition, so no conformant AF can supply them.

Whether the STR carries a `Destination-Host` is therefore up to the AF. Kamailio's `ims_qos` does not send one. A real capture from this network shows:

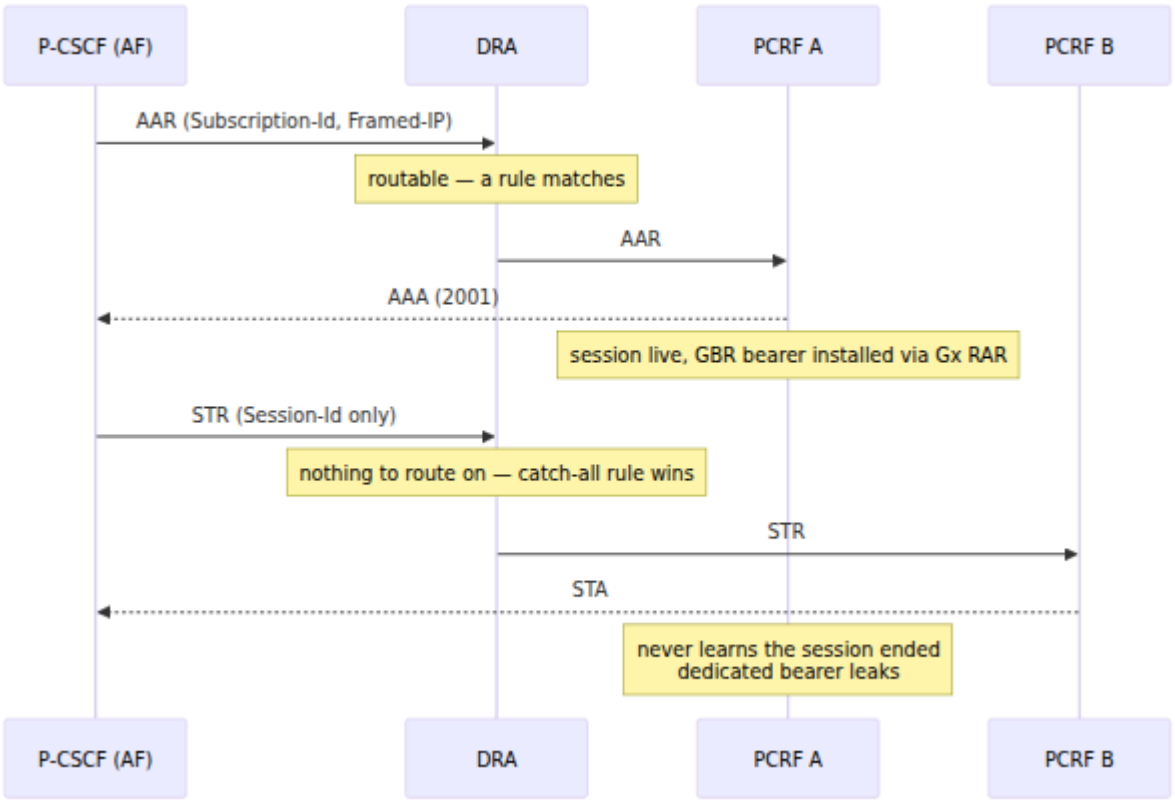
`Session-Id`, `Origin-Host`, `Origin-Realm`, `Destination-Realm`, `Auth-Application-Id`, `Vendor-Specific-Application-Id`, `AF-Application-Identifier`, `Termination-Cause`.

The `Session-Id` is the AF's own opaque handle — per RFC 6733 §8.8 it is globally unique and stable for the life of the session, but it identifies neither the subscriber nor the server.

Compare this with the `AA-Request` that opened the session, which carries `Subscription-Id` and `Framed-IP-Address` and can therefore be routed by an Advanced Routing rule. Compare it also with **Gx**, whose `CC-Request` ABNF (TS 29.212 §5.6.2) includes `[ Destination-Host ]`, `*[ Subscription-Id ]` **and** `[ Framed-IP-Address ]` — which is exactly why Gx needs no equivalent of this module.

TS 29.213 §7.3.5 puts the burden on the DRA regardless: a client "shall be capable of sending every message of a session to the DRA", with bypass only something that "may be configured".

With a single PCRF this asymmetry is invisible. With more than one behind the DRA it is not:



This failure is silent. RFC 6733 §8.4.2 prescribes no result code for an STR naming an unknown session — §8.1's state machine handles "STR received" from *any* state with "send STA, clean up" — so a PCRF answering `2001` for a session it never had is conformant, and that is what real implementations do. Do not expect result codes to reveal this. (`5002 DIAMETER_UNKNOWN_SESSION_ID` is prescribed for the **ASA**, §8.5.2, not the STA.)

The same applies, less visibly, to a *pool* of interchangeable PCRFs. A rule routing to `peers: [a, b]` selects per message, so an AAR and its STR can land on different members of the same pool.

When You Need It

- Migrating between PCRFs, or running two in parallel for a per-subscriber trial.
- Any Advanced Routing rule that sends Rx to more than one peer.

You do **not** need it with a single PCRF, and it is disabled by default.

How It Works

Message	Action
`AA-Answer` (265, `Result-Code` 2001)	Bind `Session-Id` → the PCRF that answered
`AA-Request` (265)	Route to the bound PCRF
`Session-Termination-Request` (275)	Route to the bound PCRF
`Session-Termination-Answer` (275)	Release the binding — **only** if it came from the bound PCRF and its `Result-Code` is final (2xxx or 5xxx)

Bindings Follow Facts, Not Intentions

Binding happens on the **answer**, never on the request. A request only says where the DRA *tried* to send something; the answer proves a session exists on that node. Three consequences:

- An AAR answered with an error, or never answered, leaves no binding.
- A session is released when its STA arrives, not when its STR is routed. Retransmission after a failover is explicitly sanctioned by RFC 6733 §5.5.4; releasing early would misroute the retry — the very failure this module prevents.
- A binding is never rewritten by a request, so a transient peer outage cannot silently migrate a live session to a different PCRF.

Release requires **both** that the STA came from the peer that owns the binding, and that its `Result-Code` is final. RFC 6733 §8.4.2 has the server release its resources when it sends the STA, so 2xxx and 5xxx are equally final; a 3xxx/4xxx is transient and the AF may retransmit, so the binding survives for the retry.

The ownership half is not defensive tidiness. Without it the module fails in the exact scenario it was built for: the owner goes unreachable, the STR falls through to another PCRF, that PCRF answers `2001` for a session it never had, and the binding to the owner is destroyed while its bearer is still up — leaving you worse off than with no module, because teardown now *looks* pinned. It is the mirror of `insert_new` on the bind side: a second PCRF can neither steal a binding nor destroy one.

What It Will Not Override

Routing is only ever redirected away from a relay decision. Everything else is a deliberate outcome and is left alone:

Outcome	Source	Honoured
`:discard`	a `route: :drop` rule	yes
`{:answer_message, code}`	a `route: {:error, code}` rule	yes
`{:answer_message, 3002}`	Advanced Routing failing closed with no connected peer	yes
explicit `Destination-Host`	the AF naming its server (RFC 6733 §6.1.5)	yes

The cost is that a session cannot be torn down while every peer its rules point at is unreachable. That is the correct trade: an operator draining a node, or failing closed on purpose, is not asking to be routed around.

Ordering

This is the only stage that runs **after** Advanced Routing. It has to: an STR can only ever match a catch-all rule, and the binding must be able to beat it.

The answer-path observer runs **before** Advanced Transform, which can rewrite or remove `Session-Id` and `Result-Code` on answers; the binding must be keyed on the `Session-Id` the AF actually used. It is deliberately insensitive to topology hiding: the answering peer is read from its Diameter capabilities, not from an `Origin-Host` AVP, so rewriting AVPs cannot mislead it.

Configuration

```
config :dra,
  module_rx_session_binding: %{
    enabled: true,
    children: [DRA.Router.Modules.RxSessionBinding],
    ttl_seconds: 43_200
  }
```

Key	Purpose
`enabled`	Off by default. Independent of `module_subscriber_lookup` — the two solve different problems and either may be wanted alone.
`ttl_seconds`	Leak guard for sessions whose STA never arrives. **Not** a session timer.

Size `ttl_seconds` **above** the AF's `Authorization-Lifetime` so a long call is never unbound mid-session. A P-CSCF advertising `Authorization-Lifetime: 36000` needs more than 10 hours; the 12-hour default suits it. Sweeps run every `ttl_seconds / 2` (minimum 60s), so a leaked binding is reclaimed within 1.5 × TTL.

Observability

Three telemetry events, all carrying `session_id`:

Event	Metadata
`[:diameter, :rx_session_binding, :bind]`	`serving_host`
`[:diameter, :rx_session_binding, :route]`	`serving_host`
`[:diameter, :rx_session_binding, :release]`	—

Plus `[:diameter, :rx_session_binding, :reap]` with a `count`, emitted only when the TTL sweep finds expired bindings. A non-zero reap count means STAs are going missing and is worth alerting on.

`RxSessionBinding.count/0` returns the number of live bindings; `RxSessionBinding.all/0` returns them as `{session_id, origin_host, inserted_at}`.

Limitations

- **Bindings are in-memory and node-local.** A DRA restart, or a sibling crash under the `:one_for_all` supervisor, drops them. In-flight sessions then fall back to whatever the rules say for the remainder of the call. Bindings are not shared between DRAs, so an AF must reach the same DRA for both halves of a session.
- **The adjacent peer is what gets bound.** In relay mode the DRA may append a Route-Record but not insert a `Destination-Host` (RFC 6733 §6.1.9), so a PCRF sitting behind another agent cannot be pinned — only that agent can.
- **A late AA-Answer can resurrect a released session.** If an AAR-update's answer is processed after the STA, `insert_new` recreates the row. The session is over, so this is a leaked row rather than a misroute (Session-Ids are not reused — RFC 6733 §8.8); it is bounded by `ttl_seconds` and shows up in the `:reap` counter.
- **A session cannot be released while Advanced Routing is failing closed** for Rx, because `{:answer_message, 3002}` is indistinguishable from an operator's deliberate `route: {:error, 3002}`. Keep every PCRF that holds live sessions in the rule's peer list and this does not arise.

- **`children` is required in config.** `DRA.Supervisor` only starts a module's processes when the config carries a `children` list. With `enabled: true` and no `children`, the GenServer never starts, every entry point degrades to "no binding", and nothing warns you.
- **Only the AF → PCRF direction is bound.** PCRF-initiated `RAR` and `ASR` carry a mandatory `Destination-Host` (RFC 6733 §8.3.1, §8.5.1) and route on that, so they need no binding.
- **Only the Rel-7+ Rx application id (16777236).** Sessions on the legacy Rel-6 Rx/Gq id (16777229) are not bound.
- **Telemetry is emitted but not exported.** The four events below have no `Telemetry.Metrics` definitions, so nothing reaches Prometheus today — the same gap the pre-existing `subscriber_lookup` events have. Use `RxSessionBinding.count/0` from a remote shell until that is wired up.

SCTP Multihoming

SCTP multihoming provides network redundancy for Diameter transport by allowing each endpoint to bind to multiple IP addresses within a single association. If the primary network path fails, SCTP automatically fails over to an alternative path without disrupting the Diameter session.

Why Multihoming Matters

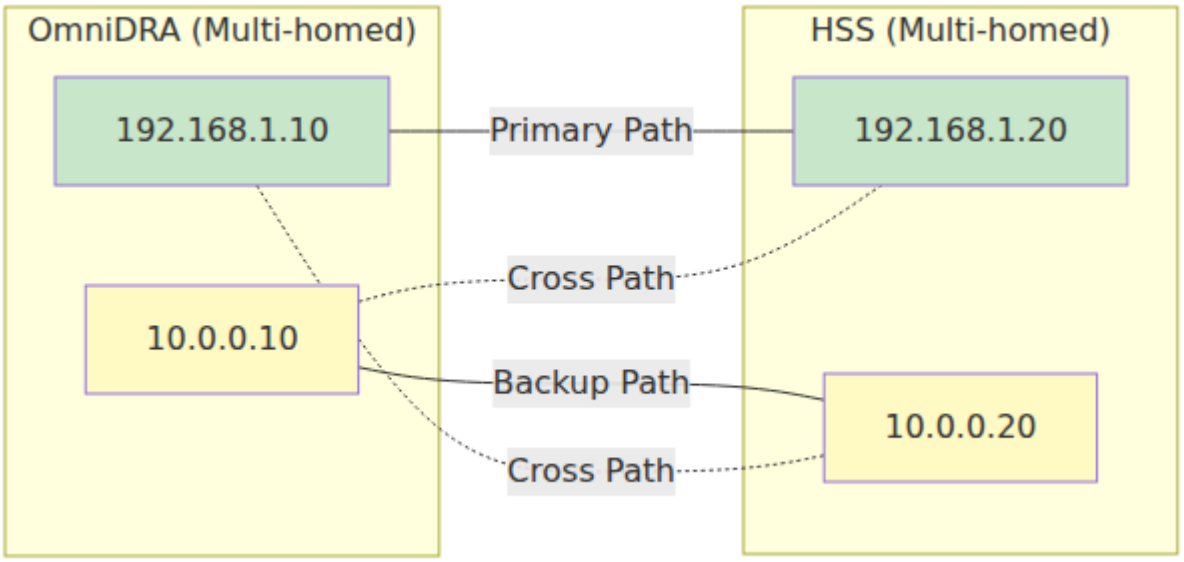
Diameter over SCTP is defined in RFC 6733, and the multihoming behaviour described here is a native capability of **SCTP** itself, specified in RFC 4960. Where TCP binds an association to a single source/destination IP pair, SCTP allows an association to span multiple IP addresses on each side. This gives OmniDRA transport-layer resilience against the loss of an individual network path, NIC, switch, or routed segment - without tearing down or re-establishing the Diameter connection.

Key characteristics:

- SCTP **heartbeats** continuously monitor all network paths in the association.
- **Automatic failover** occurs at the kernel level if the primary path becomes unreachable.
- No Diameter session disruption occurs during path switchover - the association and its Diameter state persist.
- Path selection is handled automatically by the operating system kernel.

How It Works

A multi-homed association is established between OmniDRA and a peer, with each endpoint advertising several IP addresses. The kernel selects a primary path for data transmission and treats the remaining paths as standby routes, promoting one of them automatically if the primary fails.



In the diagram above, the **Primary Path** (green) carries traffic under normal conditions. The **Backup Path** (yellow) and the **Cross Paths** (dashed) are all part of the same SCTP association and remain available for immediate failover.

Configuration

Multihoming is configured in `config/runtime.exs` in two places: the local listen addresses that OmniDRA binds to, and the remote addresses of each multi-homed peer. See the Configuration Reference for the full configuration structure.

DRA Listen Addresses

The `listen_ip` field controls which local IP addresses OmniDRA binds to for incoming connections. It accepts either a single IP string (backward compatible) or a list of IP strings to enable multihoming.

```
%{
  # Single IP (backward compatible)
  listen_ip: "192.168.1.10",

  # Multiple IPs for SCTP multihoming
  listen_ip: ["192.168.1.10", "10.0.0.10"],

  listen_port: 3868
}
```

Parameter	Type	Required	Default	Description
`listen_ip`	String or List of Strings	Yes	-	Local IP address(es) OmniDRA binds to for incoming connections. A single string binds one address; a list of strings enables SCTP multihoming across all listed addresses. For **TCP transport**, only the first IP in the list is used. For **SCTP transport**, all listed IPs are bound to the association.
`listen_port`	Integer	Yes	-	Local port for incoming Diameter connections. Standard port is 3868 per RFC 6733.

Backward compatibility: A single IP string remains fully supported. Existing single-homed configurations continue to work unchanged; multihoming is opt-in by supplying a list.

Peer Configuration

Each peer entry declares the peer's remote address(es). The `ip` field carries the primary address and is always required. The optional `additional_ips` field lists all addresses used for the multi-homed association.

```
peers: [
  %{
    host: "hss01.example.com",
    realm: "example.com",
    ip: "192.168.1.20",                              # Primary IP
(required)
    additional_ips: ["192.168.1.20", "10.0.0.20"],  # All IPs for
multihoming
    port: 3868,
    transport: :diameter_sctp,
    initiate_connection: true
  }
]
```

Parameter	Type	Required	Default	Description
`host`	String	Yes	-	Peer's Diameter Identity (FQDN). Must exactly match the peer's configured identity.
`realm`	String	Yes	-	Peer's Diameter Realm. Used for routing decisions.
`ip`	String	Yes	-	Peer's primary IP address. Always required, including for backward compatibility with single-homed configurations.
`additional_ips`	List of Strings	No	-	All IP addresses used for the multi-homed SCTP association. Optional - if omitted, only `ip` is used. For SCTP, include the primary IP in this list. For TCP, this field is ignored (TCP

Parameter	Type	Required	Default	Description
				does not support multihoming).
`port`	Integer	Yes	-	Peer's Diameter port. Standard port is 3868 per RFC 6733.
`transport`	Atom	Yes	-	Transport protocol for the peer connection: `:diameter_sctp` or `:diameter_tcp`. Multihoming requires `:diameter_sctp`.
`initiate_connection`	Boolean	Yes	-	When `true`, OmniDRA actively initiates the connection to the peer. When `false`, OmniDRA waits for the peer to connect.

Backward compatibility: The `ip` field alone describes a single-homed peer. Omitting `additional_ips` leaves the peer single-homed, so existing peer configurations require no changes.

Complete Example

The following configuration binds OmniDRA to two local addresses and connects to one multi-homed HSS and one single-homed MME.

```
config :dra,
  diameter: %{
    service_name: :omnitouch_dra,
    listen_ip: ["192.168.1.10", "10.0.0.10"],  # Multihomed DRA
    listen_port: 3868,
    host: "dra01",
    realm: "example.com",
    product_name: "OmniDRA",
    vendor_id: 10415,
    request_timeout: 5000,
    peer_selection_algorithm: :random,
    allow_undefined_peers_to_connect: false,
    peers: [
      # Multihomed HSS connection
      %{
        host: "hss01.example.com",
        realm: "example.com",
        ip: "192.168.1.20",
        additional_ips: ["192.168.1.20", "10.0.0.20"],
        port: 3868,
        transport: :diameter_sctp,
        initiate_connection: true
      },
      # Single-homed MME (backward compatible)
      %{
        host: "mme01.example.com",
        realm: "example.com",
        ip: "192.168.1.30",
        port: 3868,
        transport: :diameter_sctp,
        initiate_connection: false
      }
    ]
  }
```

The `listen_ip`, `listen_port`, and all peer parameters are documented in the tables above. The remaining top-level parameters control OmniDRA's Diameter identity and peer-selection behaviour and are documented in the Configuration Reference:

Parameter	Type	Required	Default	D
`service_name`	Atom	Yes	-	Int of Or Di se
`host`	String	Yes	-	Or ow Ide (F
`realm`	String	Yes	-	Or ow Re
`product_name`	String	Yes	-	Pr ad pe ca ex
`vendor_id`	Integer	Yes	-	Ve ad pe is ve ide
`request_timeout`	Integer	No	`5000`	Ma tin mi to pe be

Parameter	Type	Required	Default	D
				ou re
`peer_selection_algorithm`	Atom	Yes	-	Alg us am pe rou
`allow_undefined_peers_to_connect`	Boolean	Yes	`false`	Wh on list ma Wh un are ac
`peers`	List	Yes	-	Lis co co Se Co

SCTP Socket Buffers

OmniDRA sets an explicit socket receive and send buffer on every SCTP transport via the top-level `diameter_socket_buffer_size` key (default 4 MB). Unlike `listen_ip` and `peers`, this key sits at the **top level** of the `:dra` application config, **not** inside the `diameter:` map.

Without an explicit buffer, the SCTP socket inherits a small default `SO_RCVBUF`, which pins the SCTP receive window to roughly one MTU and causes zero-window stalls under load. The Linux kernel caps `SO_RCVBUF` at

`net.core.rmem_max` (and the send buffer at `net.core.wmem_max`) - to realise buffer sizes larger than that cap, raise those sysctls on the host.

```
config :dra,
  # Top-level key - NOT inside the diameter: map. Applied to every
SCTP transport.
  diameter_socket_buffer_size: 4 * 1024 * 1024,  # 4 MB
(recbuf/sndbuf)
  diameter: %{
    # ... service and peer configuration ...
  }
```

Parameter	Type	Required	Default	
`diameter_socket_buffer_size`	Integer	No	`4194304` (4 MB)	Socket re bytes ap (sets bot affects `:` for TCP. V `net.core` require r

Requirements

For SCTP multihoming to function correctly across both endpoints:

- The **SCTP kernel module** must be loaded on the host (the `lksctp-tools` package on Linux).
- All configured IP addresses must be routable to and from the peer.
- Firewall rules must permit SCTP traffic on every configured IP address.
- Both endpoints should be configured for multihoming to achieve full redundancy. A multi-homed OmniDRA connected to a single-homed peer only gains path redundancy on the OmniDRA side.

Limitations

- **TCP does not support multihoming.** With `:diameter_tcp` transport, only the primary IP is used - `listen_ip` uses the first IP in the list, and peer `additional_ips` is ignored.
- **Path failover timing** depends on kernel SCTP parameters (heartbeat interval, retransmission thresholds), which are tuned outside OmniDRA at the operating-system level.

References

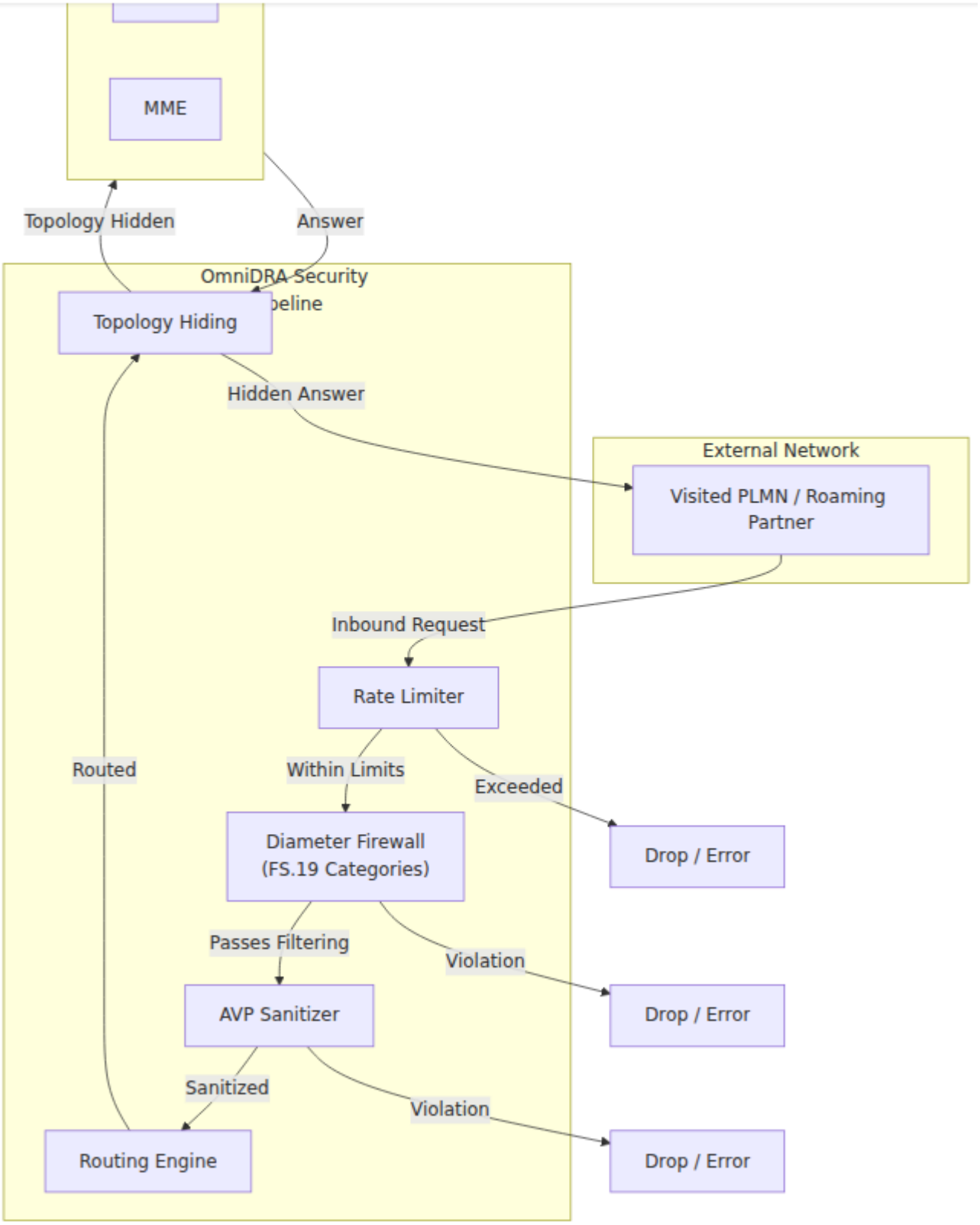
- RFC 4960 - Stream Control Transmission Protocol (SCTP)
- RFC 6733 - Diameter Base Protocol
- Configuration Reference

Diameter Security

OmniDRA provides a comprehensive suite of Diameter security modules aligned with **GSMA FS.19** (Diameter Interconnect Security, v10.0) and **GSMA FS.21** (Interconnect Signalling Security Recommendations, v12.0). These modules protect the network against signalling-level attacks on Diameter interconnect interfaces.

Each module is independently configurable, can be enabled or disabled without affecting the others, and emits its own telemetry events for monitoring and alerting.

Architecture Overview



Processing Order

Inbound Diameter requests pass through the security modules in the following order. A message rejected at any stage is never forwarded to subsequent stages.

Order	Module	Direction	Purpose
1	Rate Limiter	Inbound	Volumetric flood protection
2	Diameter Firewall	Inbound	FS.19 protocol and content filtering
3	AVP Sanitizer	Inbound	AVP validation and stripping
4	Routing Engine	-	Standard Diameter routing
5	Topology Hiding	Outbound	Network topology concealment

Actions

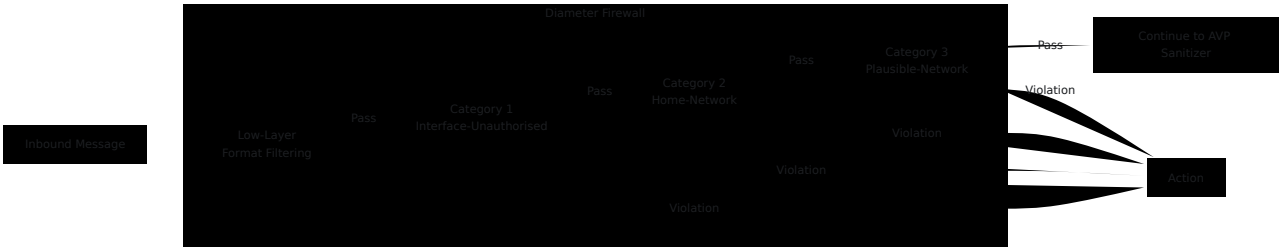
All security modules support configurable actions when a violation is detected. Actions are configured per-module (and per-category for the Diameter Firewall).

Action	Behaviour	Diameter Result
`{:error, 3002}`	Respond with a Diameter error answer	DIAMETER_UNABLE_TO_DELIVER
`{:error, 3004}`	Respond with a Diameter error answer	DIAMETER_TOO_BUSY
`{:error, 3007}`	Respond with a Diameter error answer	DIAMETER_APPLICATION_UNSUPPORTED
`:drop`	Silently discard the message	No response sent
`:log_only`	Log the violation but allow the message through	Message continues

---

Diameter Firewall

The Diameter Firewall implements the four filtering layers defined in **GSMA FS.19 Section 3.3** and the protocol-agnostic packet categorisation from **FS.21 Section 7**. Messages are evaluated through each layer in order; a violation at any layer stops further processing.



Low-Layer Format Filtering

GSMA Reference: FS.19 Section 3.3.4, FS.21 Section 7.3.1

Low-layer filtering detects protocol-level violations and basic anti-spoofing attempts without needing to understand upper-layer application semantics. This layer catches malformed messages before they reach deeper inspection.

Checks performed:

Check	FS.19 Reference	Description
AVP Duplication	Section 3.3.4, 4.8.1	Detects duplicate instances of AVPs that must appear at most once (e.g. Origin-Host, Origin-Realm). Prevents AVP doubling evasion attacks.
Session-Id Ordering	Section 3.3.4	Validates that Session-Id (AVP 263) is the first AVP in the message, per RFC 6733 Section 8.8.
Destination-Host in Answers	Section 3.3.4	Rejects answer messages that contain a Destination-Host AVP, which is a protocol violation that may indicate filter evasion.
Mandatory AVP Validation	Section 3.3.5	Validates that required AVPs are present for each command code (e.g. ULR must contain User-Name and Visited-PLMN-Id per 3GPP TS 29.272 Section 7.2.3).

```
low_layer: %{
  enabled: true,
  action: {:error, 3002},
  # AVP codes that must not appear more than once (FS.19 Section
3.3.4)
  # 264 = Origin-Host, 296 = Origin-Realm, 283 = Destination-
Realm, 293 = Destination-Host
  single_instance_avps: [264, 296, 283, 293],
  # Session-Id must be the first AVP (RFC 6733 Section 8.8)
  enforce_session_id_first: true,
  # Answer messages must not contain Destination-Host (FS.19
Section 3.3.4)
  reject_destination_host_in_answers: true,
  # Mandatory AVPs per command code (3GPP TS 29.272)
  # 1 = User-Name (IMSI), 1407 = Visited-PLMN-Id, 264 = Origin-
Host, 296 = Origin-Realm
  mandatory_avps: %{
    316 => [1, 1407, 264, 296],
    318 => [1, 1407, 264, 296]
  }
}
```

Category 1 — Interface-Unauthorised Packet Filtering

GSMA Reference: FS.19 Section 3.3.5, FS.21 Section 7.2.1

Category 1 filtering ensures that only authorised Diameter Application IDs and Command Codes are accepted on each interconnect interface. This prevents external access to internal-only interfaces (e.g. blocking Sh commands over an S6a interface) and enforces that roaming partners only send message types covered by their roaming agreements.

The whitelist approach follows the FS.19 recommendation: **block all Diameter messages except those explicitly required for a given interface**.

Whitelists can be configured globally (applying to all peers) or per-peer, allowing different roaming partners to have different permitted message sets.

```
category_1: %{
  enabled: true,
  action: {:error, 3007},
  whitelists: %{
    # Per-peer whitelist — restrict this partner to ULR and AIR
only (FS.19 Section 3.3.5)
    "restricted-partner.roaming.com" => %{
      16_777_251 => [316, 318]
    },
    # Default whitelist — standard S6a commands permitted for all
other peers
    all: %{
      # S6a/S6d (FS.19 Section 3.3.5, Table 2)
      16_777_251 => [316, 317, 318, 319, 320, 321, 323],
      # S13 — ME Identity Check (FS.19 Section 3.3.5)
      16_777_252 => [324],
      # S6c — SMS via HSS (FS.19 Section 3.3.5.1)
      16_777_312 => [8388647, 8388648],
      # SGd — SMS via MME (FS.19 Section 3.3.5.2)
      16_777_313 => [8388645, 8388646]
    }
  }
}
```

Common S6a/S6d Whitelist:

Command Code	Name	Direction	Reference
316	Update-Location-Request/Answer	MME → HSS	3GPP TS 29.272 §7.2.3
317	Cancel-Location-Request/Answer	HSS → MME	3GPP TS 29.272 §7.2.7
318	Authentication-Information-Request/Answer	MME → HSS	3GPP TS 29.272 §7.2.5
319	Insert-Subscriber-Data-Request/Answer	HSS → MME	3GPP TS 29.272 §7.2.9
320	Delete-Subscriber-Data-Request/Answer	HSS → MME	3GPP TS 29.272 §7.2.11
321	Purge-UE-Request/Answer	MME → HSS	3GPP TS 29.272 §7.2.13
323	Notify-Request/Answer	MME → HSS	3GPP TS 29.272 §7.2.15

Common Application IDs for roaming interconnect (FS.19 Section 3.3.5):

Application ID	Interface	Reference
16777251	S6a/S6d	3GPP TS 29.272
16777252	S13	3GPP TS 29.272
16777312	S6c	3GPP TS 29.338
16777313	SGd	3GPP TS 29.338
16777255	SLg	3GPP TS 29.172
16777267	S9	3GPP TS 29.215

Category 2 — Home-Network Packet Filtering

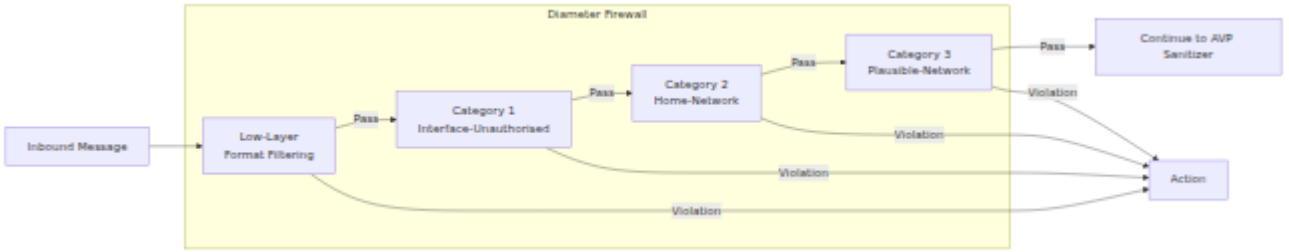
GSMA Reference: FS.19 Section 3.3.6, FS.21 Section 7.2.2

Category 2 filtering protects home subscribers from being targeted by messages arriving from the interconnect. Messages on protected command codes are inspected for subscriber identity (IMSI in the User-Name AVP, MSISDN in AVP 701). If the identity matches a home subscriber prefix, the message is rejected — legitimate traffic for home subscribers should originate from within the home network, not from external peers.

This prevents attacks where an external entity sends location updates, subscriber data requests, or authentication queries targeting the operator's own subscribers.

```
# Top-level: define home subscriber prefixes
home_imsi_prefixes: ["31338", "31339"],
home_msisdn_prefixes: ["+1313"],

category_2: %{
  enabled: true,
  action: {:error, 3002},
  # S6a command codes that carry subscriber identity (FS.19
Section 3.3.6, Table 12)
  protected_command_codes: [316, 317, 318, 319, 320, 321, 323]
}
```



Category 3 — Plausible-Network Packet Filtering

GSMA Reference: FS.19 Section 3.3.7, FS.21 Section 7.2.3

Category 3 filtering detects implausible location changes by tracking the last-seen network for each subscriber (IMSI). When an Update-Location-Request arrives from a different visited network than the previous one, the time elapsed is compared against a configurable threshold. A realm change occurring faster than physically possible indicates a potential attack (e.g. spoofed location updates).

This module maintains stateful per-IMSI tracking using an ETS table with automatic cleanup of stale entries (entries older than 24 hours are removed periodically).

Checks performed:

Check	FS.19 Reference	Description
Previous Location Check	Section 3.3.7.1	Compares the Origin-Realm of the current ULR with the last-seen Origin-Realm for that IMSI
Velocity/Time Check	Section 3.3.7.2	Flags realm changes that occur within a configurable minimum time window

```
category_3: %{
  enabled: true,
  # Start with log_only to observe patterns before enforcing
(FS.19 Section 3.3.7)
  action: :log_only,
  # Maximum plausible velocity in km/h (FS.19 Section 3.3.7.2)
  max_velocity_kmh: 1200,
  # Minimum seconds between ULRs from different realms for the
same IMSI
  min_time_between_updates_seconds: 2
}
```

Configuration

The Diameter Firewall is enabled at the top level, with each filtering layer configured independently. The config snippets shown above within each category section are combined under a single `module_diameter_firewall` key:

```
config :dra,
  module_diameter_firewall: %{
    enabled: true,
    home_imsi_prefixes: ["31338", "31339"],
    home_msisdn_prefixes: ["+1313"],
    low_layer: %{ ... },      # See Low-Layer Format Filtering
above
    category_1: %{ ... },     # See Category 1 above
    category_2: %{ ... },     # See Category 2 above
    category_3: %{ ... }      # See Category 3 above
  }
```

Top-Level Parameters

Parameter	Type	Required	Default	
`enabled`	Boolean	Yes	`false`	Enable or disabl `false`, all messa
`children`	List	No	`[]`	Processes starte module is enabl `[DRA.Router.Mo` which is require
`home_imsi_prefixes`	List	No	`[]`	List of IMSI prefi by Category 2 fi
`home_msisdn_prefixes`	List	No	`[]`	List of MSISDN p Used by Categor

Low-Layer Parameters

Parameter	Type	Required	Default
`enabled`	Boolean	No	`true`
`action`	Action	No	`{:error, 3002}`
`single_instance_avps`	List	No	`[264, 296, 283, 293]`
`enforce_session_id_first`	Boolean	No	`true`

Parameter	Type	Required	Default
`reject_destination_host_in_answers`	Boolean	No	`true`
`mandatory_avps`	Map	No	`%{}`

Category 1 Parameters

Parameter	Type	Required	Default	Description
`enabled`	Boolean	No	`true`	Enable or disable Category
`action`	Action	No	`{:error, 3007}`	Action to take when a Cate violation is detected. Defa with DIAMETER_APPLICATION_U
`whitelists`	Map	Yes	-	Map of peer hostname (or permitted Application ID / Code combinations. The `:` provides a default whitelis specific entries take priorit

Category 2 Parameters

Parameter	Type	Required	Default	Descriptio
`enabled`	Boolean	No	`true`	Enable or disable Category 2 filtering.
`action`	Action	No	`{:error, 3002}`	Action to take when a home subscriber i targeted from interconnec
`protected_command_codes`	List	No	`[]`	Command codes that trigger hom subscriber identity checks. Typically the S6a command codes that carry subscriber identity.

Category 3 Parameters

Parameter	Type	Required	Default
`enabled`	Boolean	No	`false`
`action`	Action	No	`:log_only`
`max_velocity_kmh`	Integer	No	`1200`

Parameter	Type	Required	Default
`min_time_between_updates_seconds`	Integer	No	`2`

Telemetry Events

Event	Description
`[:diameter, :firewall, :low_layer, :block]`	Low-layer format violation detected and blocked
`[:diameter, :firewall, :category_1, :block]`	Category 1 violation detected and blocked
`[:diameter, :firewall, :category_2, :block]`	Category 2 violation detected and blocked
`[:diameter, :firewall, :category_3, :block]`	Category 3 violation detected and blocked
`[:diameter, :firewall, :pass, :count]`	Message passed all firewall checks

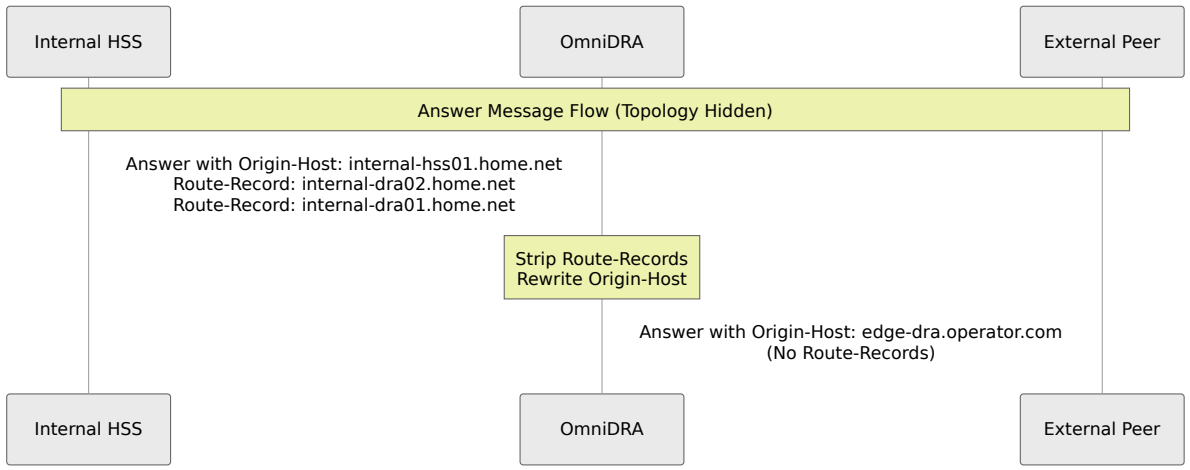
All events include metadata: `origin_host`, `application_id`, `command_code`, `application_name`, `command_name`, and `reason`.

Topology Hiding

GSMA Reference: FS.19 Section 2.4, Section 3.4; FS.21 Section 3.6

The Topology Hiding module prevents internal network topology from being exposed to external peers. When Diameter messages traverse multiple internal nodes, they accumulate Route-Record AVPs and carry Origin-Host/Origin-Realm values that reveal internal hostnames, network structure, and node counts. This information can be used by attackers to map the internal network for targeted attacks.

Topology hiding operates on the **outbound path** — after routing decisions have been made, before messages are forwarded to the destination peer.



Features

Feature	FS.19 Reference	Description
Route-Record Stripping	Section 2.4	Removes all Route-Record AVPs (282) that reveal internal routing paths and node hostnames
Origin-Host Rewriting	Section 3.4	Replaces the Origin-Host AVP with the DRA's own identity (or a custom value) on answer messages
Origin-Realm Rewriting	Section 3.4	Optionally replaces the Origin-Realm AVP to hide internal realm structure
Per-Peer Control	-	Apply topology hiding selectively — only to external peers, or to all peers

```
module_topology_hiding: %{
  enabled: true,
  # Strip Route-Record AVPs revealing internal paths (FS.19
Section 2.4)
  strip_route_records: true,
  # Rewrite Origin-Host on answers to hide internal node names
(FS.19 Section 3.4)
  rewrite_origin_host: %{enabled: true, replacement: :self},
  # Optionally hide internal realm structure
  rewrite_origin_realm: %{enabled: false, replacement: :self},
  # Apply to all external peers, or list specific hostnames
  external_peers: :all
}
```

Parameters

Parameter	Type	Required	Default	Description
`enabled`	Boolean	Yes	`false`	Enable or disable the Topology Hiding module.
`strip_route_records`	Boolean	No	`true`	Remove all Route-Record AVPs (code 282) from messages before forwarding to external peers.
`rewrite_origin_host`	Map	No	See below	Controls Origin-Host rewriting on answer messages.
`rewrite_origin_realm`	Map	No	See below	Controls Origin-Realm rewriting on answer messages.
`external_peers`	`:all` or List	No	`:all`	Peers considered external. Topology hiding is only

Parameter	Type	Required	Default	Description
				applied to messages destined for these peers. Use `:all` for interconnect deployments. Use a list of hostnames for selective application.

Rewrite Parameters (apply to both `rewrite_origin_host` and `rewrite_origin_realm`):

Parameter	Type	Required	Default	Description
`enabled`	Boolean	No	varies	Enable rewriting for this AVP. Default is `true` for Origin-Host, `false` for Origin-Realm.
`replacement`	`:self` or String	No	`:self`	Replacement value. `:self` uses the DRA's own identity from the `diameter` config (`host.realm`). A string value is used as-is.

Telemetry Events

Event	Description
`[:diameter, :topology_hiding, :route_record, :stripped]`	Route-Record AVPs removed. Measurement includes `count` of AVPs stripped.
`[:diameter, :topology_hiding, :origin_host, :rewritten]`	Origin-Host AVP rewritten
`[:diameter, :topology_hiding, :origin_realm, :rewritten]`	Origin-Realm AVP rewritten

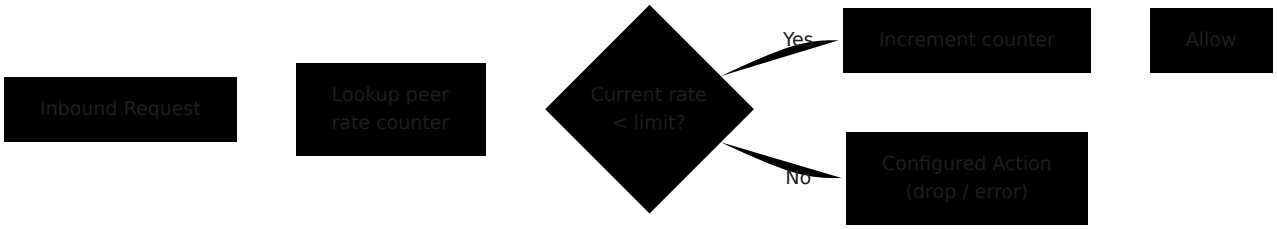
Rate Limiter

GSMA Reference: FS.19 Section 3.4 (Availability / DoS Protection)

The Rate Limiter enforces per-peer message rate limits to protect against volumetric attacks and message flooding. It operates as the first security check in the pipeline — before any message parsing or content inspection — to shed excess load as early as possible.

Rate limits are tracked per-peer using a sliding window counter. Each peer has an independent counter that resets every second.

```
module_rate_limiter: %{
  enabled: true,
  # Default limit for all peers (FS.19 Section 3.4)
  default_max_requests_per_second: 1000,
  default_action: {:error, 3004},
  # Per-peer overrides
  peer_limits: %{
    "high-volume-partner.roaming.com" => %
{max_requests_per_second: 5000, action: {:error, 3004}},
    "restricted-peer.roaming.com" => %{max_requests_per_second:
100, action: :drop}
  }
}
```



Parameters

Parameter	Type	Required	Default	
`enabled`	Boolean	Yes	`false`	En mo
`children`	List	No	`[]`	Pro su `[D` so ru
`default_max_requests_per_second`	Integer	No	`1000`	De se pe
`default_action`	Action	No	`{:error, 3004}`	De its res
`peer_limits`	Map	No	`%{}`	Ma co th

Per-Peer Override Parameters:

Parameter	Type	Required	Default	Description
`max_requests_per_second`	Integer	No	Inherits default	Maximum requests per second for this specific peer.
`action`	Action	No	Inherits default	Action when this peer exceeds its rate limit.

Telemetry Events

Event	Description
`[:diameter, :rate_limiter, :throttled]`	A message was rate limited. Measurements include `current_rate` and `limit`.
`[:diameter, :rate_limiter, :allowed]`	A message was within rate limits.

AVP Sanitizer

GSMA Reference: FS.19 Section 3.3.4, 4.8.1, 4.8.2; FS.21 Section 3.6

The AVP Sanitizer validates and sanitizes Diameter AVPs at the interconnect boundary. It addresses the FS.21 Section 3.6 recommendations for handling protocol-level manipulation attacks that operate below the level of the FS.19 filtering categories.

Features

Feature	GSMA Reference	Description
Unknown Vendor AVP Stripping	FS.21 Section 3.6	Removes AVPs from non-whitelisted vendors. Prevents injection of proprietary AVPs that may trigger unintended behaviour in backend nodes.
Grouped AVP Nesting Depth	FS.21 Section 3.6	Enforces a maximum nesting depth for grouped AVPs. Prevents stack overflow attacks using deeply nested AVP structures.

```
module_avp_sanitizer: %{
  enabled: true,
  action: {:error, 3002},
  # Only allow standard vendor AVPs on interconnect (FS.21 Section
3.6)
  # 0 = IETF, 10415 = 3GPP, 13019 = ETSI, 5535 = 3GPP2
  allowed_vendor_ids: [0, 10415, 13019, 5535],
  strip_unknown_vendor_avps: true,
  # Prevent stack overflow via deeply nested grouped AVPs (FS.21
Section 3.6)
  max_avp_nesting_depth: 10
}
```

Parameters

Parameter	Type	Required	Default	Desc
`enabled`	Boolean	Yes	`false`	Enable or AVP Sanit module.
`action`	Action	No	`{:error, 3002}`	Action to a nesting violation Vendor st silently re offending without b message.
`allowed_vendor_ids`	List	No	`[0, 10415, 13019, 5535]`	List of ve permitted interconn from othe are stripp
`strip_unknown_vendor_avps`	Boolean	No	`true`	Enable st AVPs from not in the `allowed_` list.
`max_avp_nesting_depth`	Integer	No	`10`	Maximum depth of AVP nesti Messages this are s the config `action`.

Common Vendor IDs:

Vendor ID	Organisation	Notes
0	IETF	Standard Diameter AVPs defined in RFCs
10415	3GPP	All 3GPP-defined AVPs (S6a, Gx, Rx, etc.)
13019	ETSI	ETSI-defined AVPs
5535	3GPP2	3GPP2-defined AVPs (CDMA interworking)

Telemetry Events

Event	Description
`[:diameter, :avp_sanitizer, :unknown_vendor, :stripped]`	One or more AVPs from unknown vendors were stripped
`[:diameter, :avp_sanitizer, :nesting_depth, :violation]`	A message exceeded the maximum AVP nesting depth
`[:diameter, :avp_sanitizer, :pass, :count]`	Message passed all sanitization checks

Deployment Recommendations

Recommended Enablement Order

When deploying the security modules for the first time, enable them incrementally to avoid disrupting live traffic:

1. **Rate Limiter** — Start with generous limits and monitor traffic patterns. Tighten limits once baseline rates are understood.

2. **AVP Sanitizer** — Low risk of false positives. Enable stripping and nesting checks.
3. **Diameter Firewall (Category 1)** — Define whitelists based on roaming agreements. Start with known-good Application ID / Command Code combinations.
4. **Diameter Firewall (Low-Layer)** — Enable protocol conformance checks.
5. **Diameter Firewall (Category 2)** — Configure home IMSI/MSISDN prefixes.
6. **Topology Hiding** — Enable Route-Record stripping first, then Origin-Host rewriting.
7. **Diameter Firewall (Category 3)** — Enable with `:log_only` action to observe location patterns before enforcing.

Defence in Depth

These modules implement the **defence in depth** principle described in FS.19 Section 3.4. Each layer addresses a different class of attack:

Attack Class	Primary Defence	Secondary Defence
Volumetric DoS	Rate Limiter	Diameter Firewall (all categories)
Interface Abuse	Category 1	AVP Sanitizer
Home Subscriber Targeting	Category 2	Category 1 (interface restriction)
Location Spoofing	Category 3	Category 2 (home sub check)
Topology Discovery	Topology Hiding	-
AVP Injection	AVP Sanitizer	Low-Layer Filtering
Protocol Evasion (AVP Doubling)	Low-Layer Filtering	AVP Sanitizer

GSMA Document Cross-Reference

Module / Feature	FS.19 v10.0	FS.21 v12.0
Low-Layer Format Filtering	Section 3.3.4	Section 7.3.1
Category 1 Filtering	Section 3.3.5, Annex B.3.3	Section 7.2.1
Category 2 Filtering	Section 3.3.6, Annex B.3.4	Section 7.2.2, Section 16
Category 3 Filtering	Section 3.3.7, Annex B.3.5	Section 7.2.3
Topology Hiding	Section 2.4, Section 3.4	Section 3.6
Rate Limiting	Section 3.4	-
AVP Sanitization	Section 4.8.1, 4.8.2	Section 3.6
Defence in Depth	Section 3.4	Section 3.15
Filtering Categories (Protocol-Agnostic)	Annex A	Section 7

Standard Diameter Routing

Standard Diameter Routing is the DRA's default request-forwarding behaviour, applied whenever no Advanced Routing rule matches an incoming request. It follows the priority-based routing mechanism defined in the Diameter Base Protocol (RFC 6733), selecting a destination from AVP contents and the DRA's set of connected peers.

When Standard Routing Applies

The DRA always attempts standard routing as its base behaviour. If the Advanced Routing module is enabled and one of its rules matches the request, that rule decides the outcome instead (route to named peers, route by `Destination-Host`, generate an error answer, or silently drop). When Advanced Routing is disabled, or no rule matches, the request falls through to the standard routing precedence described here.

One narrow exception applies after both: Rx Session Binding, when enabled, redirects an Rx request to the PCRF that holds its session. It only ever overrides a relay decision — a `:drop`, an error answer, and an explicit `Destination-Host` all stand.

Content-based overrides — routing on AVP values such as IMSI patterns, realm translation, and originating-peer decisions — are covered in Advanced Routing. **Session affinity** via the `Destination-Host` AVP is also detailed there.

Routing Decision Precedence

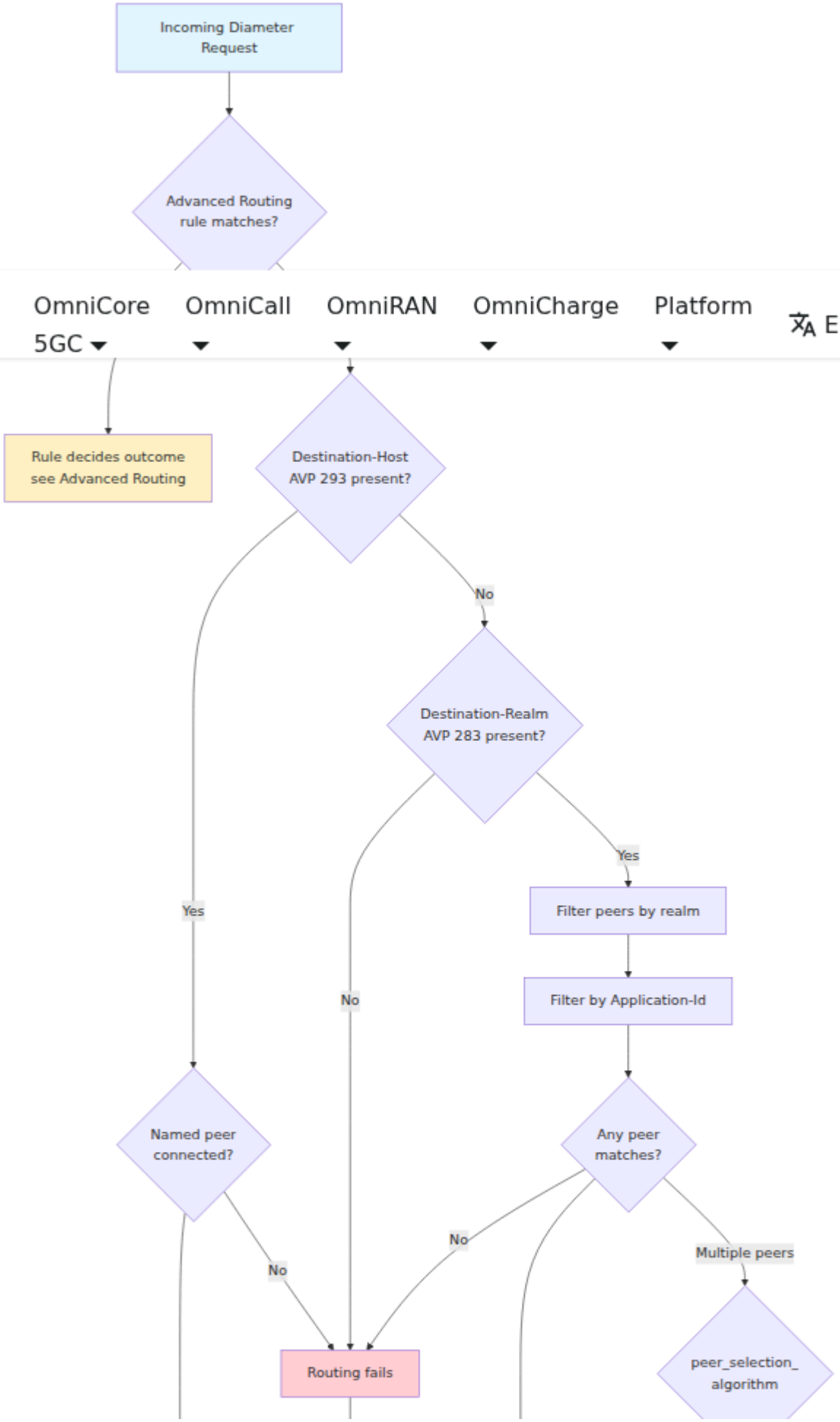
Standard routing evaluates each request against a fixed precedence order. The first applicable mechanism wins.

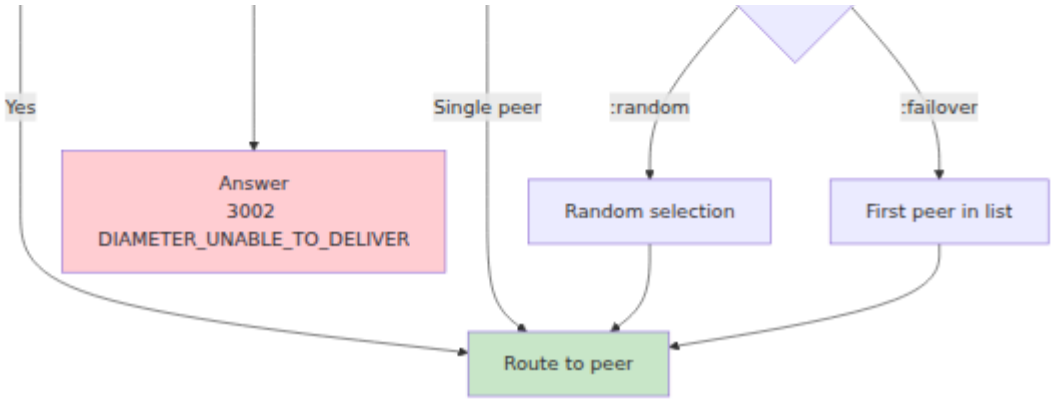
1. **`Destination-Host` AVP (293)** — The highest-priority mechanism. If the request carries a `Destination-Host` AVP, the DRA routes directly to that

host. This provides explicit, host-level routing control. If the named peer is not connected, routing fails.

2. **`Destination-Realm` AVP (283) + `Application-Id`** — If `Destination-Host` is absent, the DRA selects a connected peer that advertises support for the target **realm**. Candidate peers are then filtered by the message's **Application-Id**, so that only peers advertising support for that application (learned during Capabilities Exchange) remain. When more than one peer matches, the configured peer selection algorithm chooses between them.
3. **Relay / routing failure** — If neither a reachable `Destination-Host` nor a matching realm-plus-application peer is available, the request cannot be delivered and the DRA returns a `3002 DIAMETER_UNABLE_TO_DELIVER` answer.

The following flowchart shows the full decision path, including the point at which Advanced Routing (when enabled) may intercept the request before standard routing runs.


Incoming Diameter Request
Advanced Routing rule matches?
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
English
Rule decides outcome see Advanced Routing
Destination-Host AVP 293 present?
No
Destination-Realm AVP 283 present?
Yes
Filter peers by realm
Filter by Application-Id
Any peer matches?
Multiple peers
peer_selection_ algorithm
No
Yes
Named peer connected?
No
No
Routing fails



Only peers in the **connected** state are eligible at any stage. Disconnected or down peers are automatically excluded from realm and application filtering, and a `Destination-Host` request aimed at a disconnected peer fails immediately.

Peer Selection

When multiple connected peers match the realm and Application-Id criteria, the global `peer_selection_algorithm` setting determines which peer receives the request. This is a service-wide setting; it is not configured per realm or per peer.

```
config :dra, :service, %{
  # ... DRA identity and network settings ...
  peer_selection_algorithm: :random
}
```

Parameter	Type	Required	Default	Description
`peer_selection_algorithm`	Atom	No	`:random`	Load-balancing strategy applied when multiple connected peers match a request. `:random` distributes requests by selecting a random peer from the matching set. `:failover` always selects the first peer in the configured list, using the remaining peers only when the first is unavailable.

- **`:random`** — Spreads load evenly across all matching peers. Suitable when peers are interchangeable and equal-cost.

- `:failover` — Prefers a single primary peer, treating the ordered peer list as a priority sequence. Suitable for active/standby designs where a specific peer should carry traffic while it remains connected.

See Configuration for the complete service configuration block and peer definitions.

When No Peer Is Available

If the precedence evaluation cannot identify a reachable destination — because the `Destination-Host` peer is disconnected, no connected peer advertises the target realm, no matching peer supports the Application-Id, or the request lacks the AVPs needed to make a decision — the DRA cannot forward the request.

In this case the DRA returns an answer with `Result-Code` set to `3002 DIAMETER_UNABLE_TO_DELIVER`, as defined in RFC 6733 §7.1.5. This informs the originating peer that the request was received but could not be delivered to a suitable next hop, allowing the originator to apply its own retry or failover logic.

Reference

Result Codes

Code	Name	Description	Reference
2001	DIAMETER_SUCCESS	The request was successfully completed	RFC 6733 §7.1.1
3002	DIAMETER_UNABLE_TO_DELIVER	The request could not be delivered to a suitable next hop; no matching, connected peer was available	RFC 6733 §7.1.3

Common AVP Codes

Code	AVP Name	Type	Usage	Reference
1	User-Name	UTF8String	Subscriber identifier (IMSI in 3GPP)	RFC 6733 §8.14
264	Origin-Host	DiameterIdentity	Originating peer hostname	RFC 6733 §6.3
268	Result-Code	Unsigned32	Standard result code	RFC 6733 §7.1
283	Destination-Realm	DiameterIdentity	Target realm	RFC 6733 §6.6
293	Destination-Host	DiameterIdentity	Target host (optional)	RFC 6733 §6.5
296	Origin-Realm	DiameterIdentity	Source realm	RFC 6733 §6.4
297	Experimental-Result	Grouped	Vendor-specific result code	RFC 6733 §7.6

Common Command Codes

Command codes are part of the Diameter message header, not AVPs.

Code	Command Name	Description	Reference
257	CER/CEA	Capabilities-Exchange-Request/Answer	RFC 6733 §5.3
258	RAR/RAA	Re-Auth-Request/Answer	RFC 6733 §8.3
274	ASR/ASA	Abort-Session-Request/Answer	RFC 6733 §8.5
275	STR/STA	Session-Termination-Request/Answer	RFC 6733 §8.4
280	DWR/DWA	Device-Watchdog-Request/Answer	RFC 6733 §5.5
282	DPR/DPA	Disconnect-Peer-Request/Answer	RFC 6733 §5.4
316	ULR/ULA	Update-Location-Request/Answer (S6a)	3GPP TS 29.272
317	CLR/CLA	Cancel-Location-Request/Answer (S6a)	3GPP TS 29.272
318	AIR/AIA	Authentication-Information-Request/Answer (S6a)	3GPP TS 29.272
321	PUR/PUA	Purge-UE-Request/Answer (S6a)	3GPP TS 29.272

Common 3GPP Application IDs

Peers advertise their supported Application-Ids during Capabilities Exchange. Standard routing filters candidate peers by the request's Application-Id.

Application-Id	Interface	Description	Reference
16777251	S6a/S6d	MME/SGSN ↔ HSS authentication and subscription data	3GPP TS 29.272
16777252	S13/S13'	MME ↔ EIR equipment identity check	3GPP TS 29.272
16777238	Gx	PCEF ↔ PCRF policy and charging control	3GPP TS 29.212
16777267	S9	Home PCRF ↔ Visited PCRF roaming policy	3GPP TS 29.215
16777272	S6b	PDN-GW ↔ 3GPP AAA for non-3GPP access	3GPP TS 29.273
16777216	Cx	I-CSCF/S-CSCF ↔ HSS IMS registration	3GPP TS 29.229
16777217	Sh	AS ↔ HSS IMS user data	3GPP TS 29.329
16777255	SLg	MME/SGSN ↔ GMLC location services	3GPP TS 29.172
16777291	SLh	GMLC ↔ HSS location subscriber info	3GPP TS 29.173
16777302	Sy	PCRF ↔ OCS session binding	3GPP TS 29.219
16777310	S6m	MTC-IWF ↔ HSS/HLR for M2M devices	3GPP TS 29.336

Application-Id	Interface	Description	Reference
16777312	S6c	SMS-SC/IP-SM-GW ↔ HSS SMS routing	3GPP TS 29.338
16777345	S6t	SCEF ↔ HSS monitoring events	3GPP TS 29.336
16777236	Rx	AF ↔ PCRF media authorization	3GPP TS 29.214

Charging applications (Gy/Ro) follow 3GPP TS 32.299.

Related Documentation

- Configuration — Complete service and peer configuration reference.
- Advanced Routing — Content-based routing rules, session affinity via `Destination-Host`, and rule-driven overrides that take precedence over standard routing.

Subscriber Lookup Function (SLF)

The Subscriber Lookup Function (SLF) dynamically learns which network element is serving each subscriber by observing Diameter signaling traffic passing through the DRA. It uses these learned bindings to route subsequent requests — such as location service queries — directly to the correct serving node without requiring static routing rules.

This is particularly useful for SLg/SLh location service requests, where the GMLC needs to reach the serving MME for a given subscriber but has no prior knowledge of which MME that is.

The SLF concept is described in 3GPP TS 29.172 and 3GPP TS 29.173 for location services, with subscriber registration defined in 3GPP TS 29.272.

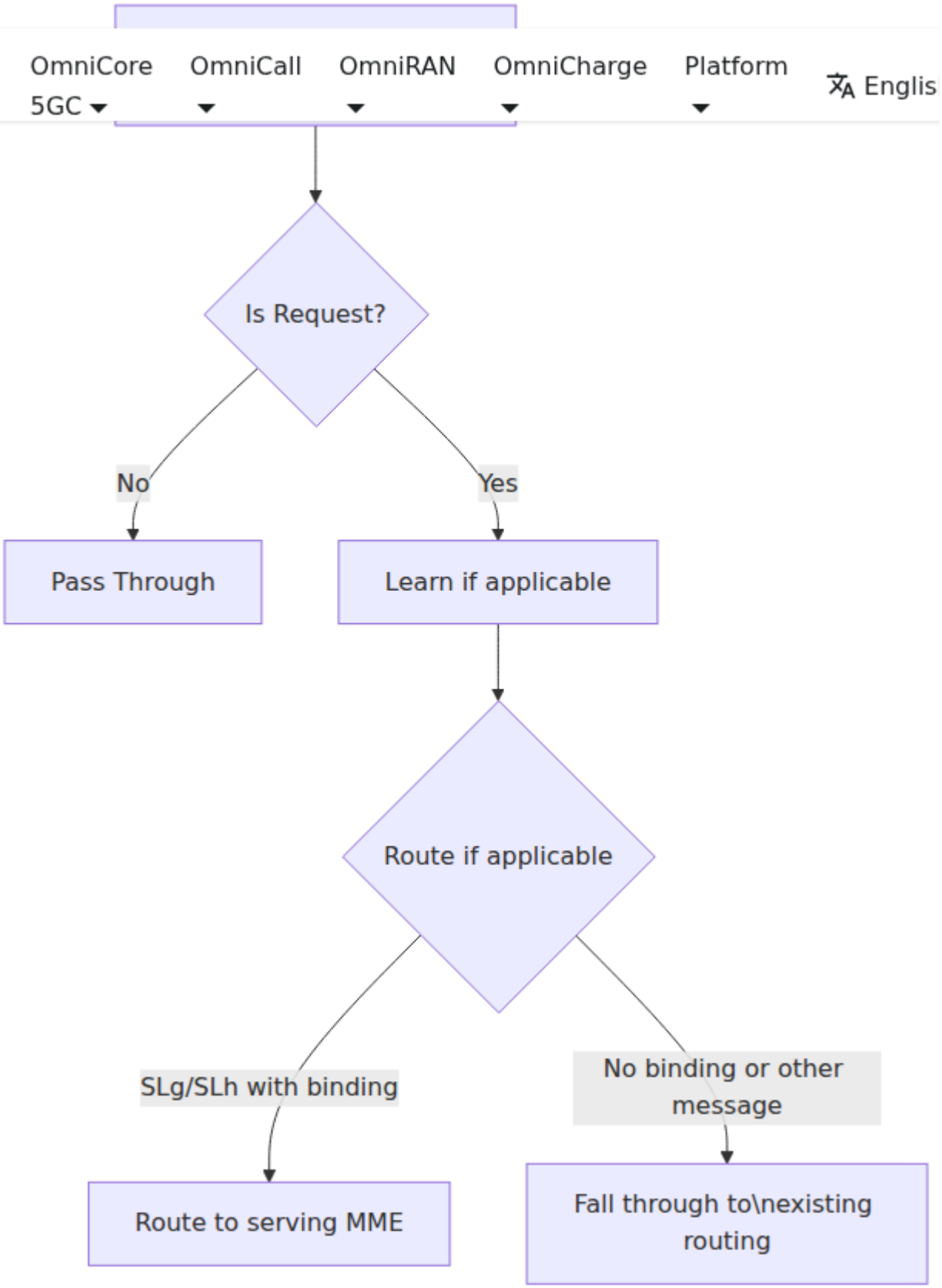
How It Works

The module operates in two phases: **learning** and **routing**.

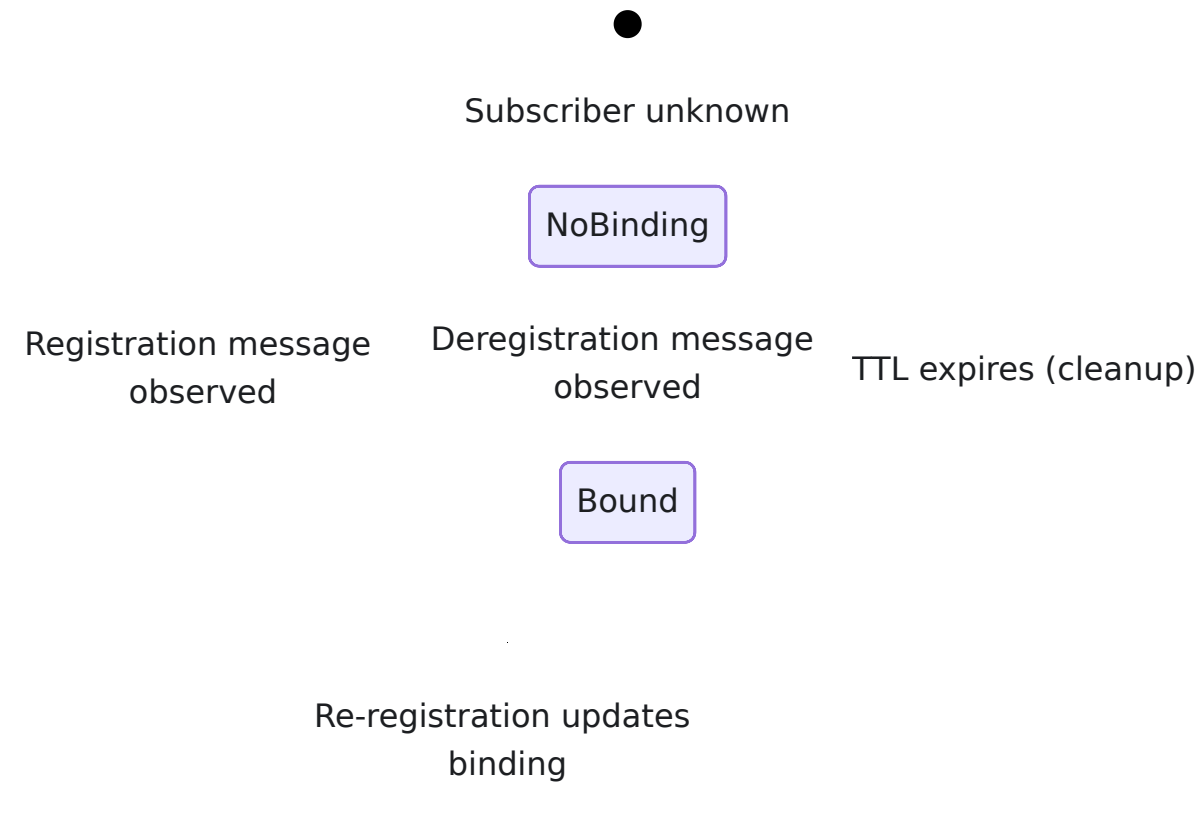
During **learning**, the module passively observes registration and deregistration messages flowing through the DRA. When a subscriber registers (e.g. via an S6a Update-Location-Request), the module records which network element is now serving that subscriber.

During **routing**, when a request arrives that needs to reach a subscriber's serving node (e.g. an SLg Provide-Location-Request), the module looks up the binding and routes directly to the correct peer.

If no binding exists for a subscriber, the request falls through to the existing routing logic (including Advanced Routing).

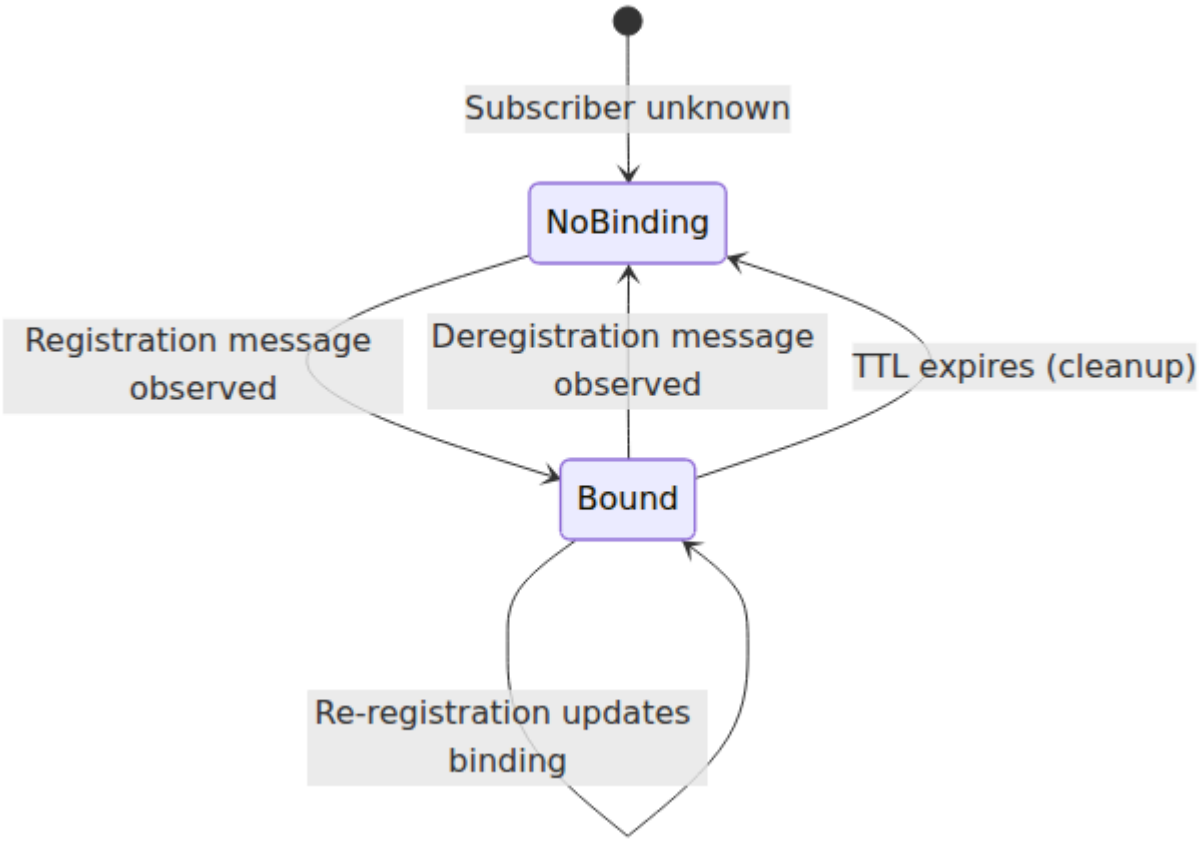

Is Request?
No
Yes
Pass Through
Learn if applicable
Route if applicable
SLg/SLh with binding
No binding or other message
Route to serving MME
Fall through to\nexisting routing

Binding Lifecycle



Learning: Messages That Create Bindings

The module learns subscriber-to-serving-node bindings from the following Diameter messages:



Interface	Message	Command-Code	Binding Created	IMSI Source
S6a	Update-Location-Request (ULR)	316	`serving_mme`	User-Name AVP (1)
Gx	Credit-Control-Request Initial (CCR-I)	272 (CC-Request-Type=1)	`serving_pgw`	Subscription-Id AVP (443)
Cx	Server-Assignment-Request (SAR)	301 (Server-Assignment-Type=1)	`serving_cscf`	Public-Identity AVP (601)

Learning: Messages That Remove Bindings

Interface	Message	Command-Code	Binding Removed
S6a	Cancel-Location-Request (CLR)	317	`serving_mme`
S6a	Purge-UE-Request (PUR)	321	`serving_mme`
Gx	Credit-Control-Request Termination (CCR-T)	272 (CC-Request-Type=3)	`serving_pgw`
Cx	Server-Assignment-Request (SAR)	301 (Server-Assignment-Type=4)	`serving_cscf`

When the last binding for a subscriber is removed, the entire subscriber record is deleted from the table.

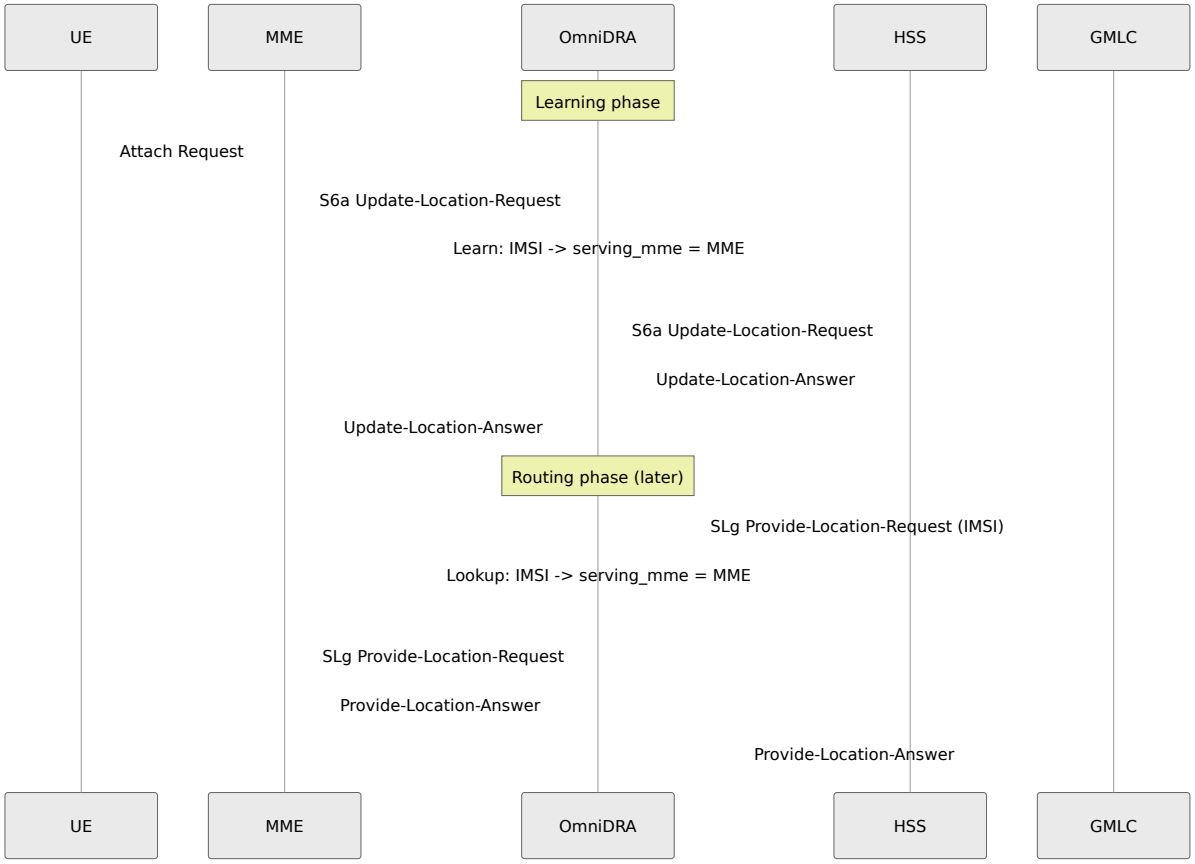
Routing: Messages That Use Bindings

Interface	Message	Command-Code	Binding Used
SLg	Provide-Location-Request (PLR)	8388620	`serving_mme`
SLh	LCS-Routing-Info-Request (RIR)	8388622	`serving_mme`

Requests routed via a learned binding are relayed to the serving node using the base relay timeout from the `diameter.request_timeout` setting (see Peer

Connectivity or Configuration). The module does not define its own timeout for these routed requests.

Example Flow: Location Service Request



Pipeline Position

Subscriber Lookup runs **before** Advanced Routing in the request processing pipeline. If SLF finds a binding and overrides the route, Advanced Routing still runs but the SLF route takes precedence.

Configuration

The module is configured under `module_subscriber_lookup` in `config/runtime.exs`.

```
module_subscriber_lookup: %{
  # Enable or disable the module
  enabled: false,

  # How long to keep bindings before expiring (seconds)
  binding_ttl_seconds: 86400
}
```

Parameters

Parameter	Type	Required	Default	De
`enabled`	Boolean	Yes	`false`	Enable or disable module. When `fa` through unmodifi learned.
`children`	List	No	`[]`	Processes started supervisor when `[DRA.Router.Mod` so the binding-tal
`binding_ttl_seconds`	Integer	No	`86400`	Time in seconds b binding expires a cleanup. Default at half the TTL int 60 seconds.

Configuration Examples

Standard Deployment

Suitable for most networks where subscribers re-register within 24 hours.

```
module_subscriber_lookup: %{
  enabled: true,
  binding_ttl_seconds: 86400
}
```

How it works: The module learns MME/PGW/CSCF bindings from passing traffic and holds them for 24 hours. SLg and SLh requests are automatically routed to the correct serving MME. Bindings are refreshed whenever a new registration message is observed for the same subscriber.

Use case: Networks with location service (LCS) requirements where the GMLC needs to reach the serving MME without static subscriber-to-MME mappings.

High-Churn Environment

For networks with frequent subscriber mobility where stale bindings should expire quickly.

```
module_subscriber_lookup: %{
  enabled: true,
  binding_ttl_seconds: 3600
}
```

How it works: Bindings expire after 1 hour. This is appropriate when subscribers frequently move between MMEs and stale bindings would cause misdirected location requests. The cleanup interval runs every 30 minutes.

Use case: Dense urban networks or events with high subscriber mobility.

Metrics

Binding Updates

Metric: `diameter.subscriber_lookup.binding.update` **Type**: Counter
Description: Incremented each time a subscriber binding is created or

updated. **Labels**:

- `imsi` - IMSI of the subscriber
- `binding_type` - Type of binding: `serving_mme`, `serving_pgw`, or `serving_cscf`
- `serving_host` - Origin-Host of the serving network element

Binding Deletions

Metric: `diameter.subscriber_lookup.binding.delete` **Type**: Counter **Description**: Incremented each time a subscriber binding is explicitly removed via a deregistration message. **Labels**:

- `imsi` - IMSI of the subscriber
- `binding_type` - Type of binding removed

Routed Requests

Metric: `diameter.subscriber_lookup.route.count` **Type**: Counter **Description**: Incremented each time an SLg/SLh request is successfully routed using a learned binding. **Labels**:

- `imsi` - IMSI of the subscriber
- `binding_type` - Binding type used for routing (currently always `serving_mme`)
- `serving_host` - Origin-Host of the peer the request was routed to

Troubleshooting

Location Requests Not Being Routed to Serving MME

Symptoms: SLg Provide-Location-Requests or SLh LCS-Routing-Info-Requests are not being routed to the expected MME, falling through to default routing instead.

Possible causes:

- The subscriber has not registered through this DRA, so no binding exists
- The binding has expired (TTL exceeded)
- The serving MME peer is not connected to this DRA
- `enabled` is set to `false`

Resolution:

1. Verify the module is enabled in the configuration
2. Check that S6a ULR traffic for the subscriber flows through this DRA (bindings are only learned from observed traffic)
3. Verify `binding_ttl_seconds` is long enough to cover the gap between registration and location request
4. Confirm the serving MME is connected as a peer

Bindings Not Being Learned

Symptoms: The binding table remains empty despite S6a/Gx/Cx traffic passing through the DRA.

Possible causes:

- The module is not enabled
- The module process is not running (check supervisor)
- Messages do not contain a valid IMSI in the expected AVP

Resolution:

1. Verify `enabled: true` in the configuration
2. Confirm the DRA was restarted after the configuration change
3. Check DRA debug logs for `SubscriberLookup: Learned` entries to confirm binding activity

Stale Bindings Causing Misrouted Requests

Symptoms: Location requests are routed to an MME that is no longer serving the subscriber.

Possible causes:

- The Cancel-Location-Request (CLR) or Purge-UE-Request (PUR) did not pass through this DRA
- `binding_ttl_seconds` is set too high for the network's mobility pattern

Resolution:

1. Reduce `binding_ttl_seconds` to match the expected subscriber re-registration interval
2. Ensure all S6a deregistration traffic flows through this DRA

Reference

Application IDs

ID	Interface	Description	Reference
16777251	S6a/S6d	MME/SGSN to HSS authentication and subscription management	3GPP TS 29.272
16777238	Gx	PCEF to PCRF policy and charging control	3GPP TS 29.212
16777216	Cx	I-CSCF/S-CSCF to HSS IMS registration	3GPP TS 29.229
16777255	SLg	GMLC to MME location services	3GPP TS 29.172
16777291	SLh	GMLC to HSS/DRA LCS routing info	3GPP TS 29.173

Command Codes

Code	Name	Interface	Description
272	Credit-Control-Request/Answer (CCR/CCA)	Gx	Session-level policy and charging control
301	Server-Assignment-Request/Answer (SAR/SAA)	Cx	IMS registration and deregistration
316	Update-Location-Request/Answer (ULR/ULA)	S6a	Subscriber location update at MME
317	Cancel-Location-Request/Answer (CLR/CLA)	S6a	HSS-initiated location cancellation
321	Purge-UE-Request/Answer (PUR/PUA)	S6a	MME-initiated UE purge
8388620	Provide-Location-Request/Answer (PLR/PLA)	SLg	Location service request to serving MME
8388622	LCS-Routing-Info-Request/Answer (RIR/RIA)	SLh	LCS routing info lookup

Binding Types

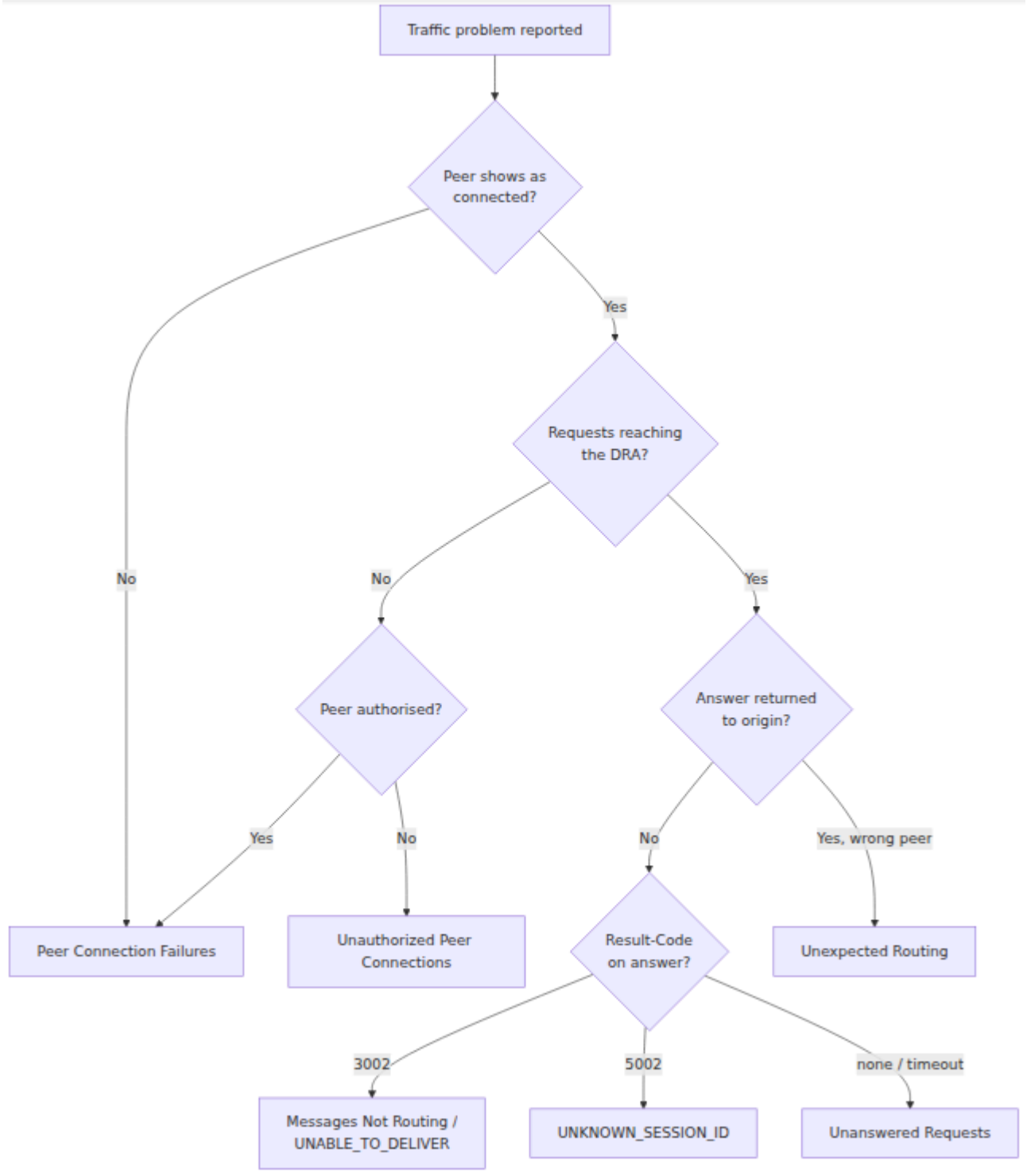
Binding Type	Learned From	Used By	Description
`serving_mme`	S6a ULR	SLg PLR, SLh RIR	The MME currently serving the subscriber
`serving_pgw`	Gx CCR-I	—	The PGW handling the subscriber's session (reserved for future routing)
`serving_cscf`	Cx SAR (Registration)	—	The S-CSCF serving the subscriber's IMS registration (reserved for future routing)

Troubleshooting

This guide covers the issues most commonly encountered when operating OmniDRA. Each entry lists the symptoms an operator observes, the likely causes, and an ordered set of steps to isolate and resolve the problem. For metric names referenced throughout, see Metrics.

Diagnosis Decision Tree

Use this flow to narrow down where a problem sits before diving into a specific section.


Traffic problem reported
Peer shows as connected?
No
Yes
Requests reaching the DRA?
No
Yes
Peer authorised?
Answer returned to origin?
Yes
No
No
Yes, wrong peer
Peer Connection Failures
Unauthorized Peer Connections
Result-Code on answer?
Unexpected Routing
3002
5002
none / timeout
Messages Not Routing / UNABLE_TO_DELIVER
UNKNOWN_SESSION_ID
Unanswered Requests

Routing Rules

Rule Not Matching

Symptoms: A configured advanced routing rule never fires; traffic falls through to standard routing or to a later rule instead.

Possible causes:

- One or more filter conditions do not match the actual message content.
- The AVP code in the filter does not match the AVP used by the Diameter application in question.
- A regular expression pattern is incorrect or over-/under-anchored.
- The message type is outside the rule's `match` scope (for example, a request-only filter applied to an answer).
- The wrong filter type is used for the module.

Resolution:

1. Verify every filter condition against a captured message. All conditions in a rule must match for the rule to fire.
2. Confirm the AVP codes match the Diameter application. See the AVP code reference in Advanced Routing.
3. Test regular-expression patterns independently against real AVP values before deploying them.
4. Ensure the message type falls within the rule's `match` scope.
5. Review the available filter types in Advanced Routing and confirm you are using the correct one for the module.

Unexpected Routing

Symptoms: Requests are forwarded, but to a different peer than intended, or standard routing is used when a rule was expected to apply.

Possible causes:

- Rule ordering. Advanced routing uses first-match-wins, so an earlier, broader rule can shadow a later, more specific one.
- The intended peer name is incorrect or the peer is not reachable.
- Two rules have overlapping filters and the wrong one wins.
- No rule matched, so Standard Routing took over.

Resolution:

1. Review rule ordering. Place more specific rules before broader ones, since the first matching rule wins. See Advanced Routing.
2. Verify peer names are spelled exactly as configured and that the target peer is connected (`diameter_peer_status` reports `1`).
3. Check for conflicting rules with overlapping filters and reorder or tighten them.
4. Confirm the fall-through behaviour described in Standard Routing matches what you observe when no rule applies.

Transform Not Applied

Symptoms: A routing rule matches, but an AVP add/edit/remove transform does not take effect on the message.

Possible causes:

- The AVP code targeted by the transform is incorrect for the use case.
- An `edit` action was used on an AVP that does not exist in the message (`edit` modifies an existing AVP but never creates one).
- The rule's filters do not match the intended message flow, so the transform is never reached.
- The packet-type filter (`:request` vs `:answer`) excludes the message.
- The action is not supported for the packet type (`edit` applies to requests only).

Resolution:

1. Confirm the AVP codes are correct for your use case. See the AVP code reference in Advanced Routing.

2. For an `edit` action, verify the AVP already exists in the message. Use `add` if the AVP may be absent.
3. Check that the rule's filters match the intended message flow.
4. Verify any packet-type filter (`:request` vs `:answer`) is correct for the direction you are transforming.
5. Confirm the action is supported for the packet type. `edit` only works on requests.

Peer Connectivity

Peer Connection Failures

Symptoms: A configured peer never reaches connected state. `diameter_peer_status{origin_host="..."}` remains `0`, and routing to that peer fails.

Possible causes:

- Firewall blocking the configured `listen_port` (default `3868`) or the peer's port.
- Incorrect peer `ip`, `port`, or `transport` in the peer configuration.
- Mismatched transport: one side is configured for `:diameter_tcp` and the other for `:diameter_sctp`.
- The peer's advertised Origin-Host does not match the configured `host` (identities must match exactly).
- `initiate_connection` is set the same way on both ends, so neither side dials, or both do.
- The peer does not advertise a supported application during Capabilities Exchange (CER/CEA).
- For an outbound peer, the peer service is not running or not listening.

Resolution:

1. Confirm firewall rules permit traffic on `listen_port` and the peer's port for the configured transport. SCTP requires the kernel SCTP module and

firewall rules to allow SCTP, not just TCP.

2. Verify the peer's `ip`, `port`, and `transport` values exactly match the peer's actual endpoint.
3. Ensure both ends use the same transport protocol.
4. Confirm the peer's advertised Origin-Host matches the configured `host`. Diameter Identities must match exactly for routing.
5. Check `initiate_connection` roles. Exactly one side should initiate: set `initiate_connection: true` on the side that dials (typically the DRA toward an HSS or PCRF) and `false` on the side that waits (typically toward an MME, SGSN, or P-GW).
6. Verify the peer advertises at least one application the DRA serves during CER/CEA. A peer with no common application will not be usable for routing.
7. For an outbound peer, confirm the remote service is running and listening. The DRA retries unreachable peers automatically.

Peer Flapping / Traffic Sent Too Early After Reconnect

Symptoms: After a peer reconnects, requests are dispatched to it and immediately fail or time out while the peer is still stabilising; alternatively, a peer that recently recovered appears usable but drops traffic.

Possible causes:

- The peer is unstable at the network layer and is repeatedly reconnecting (flapping).
- The application-level settle time is too short for this peer, so traffic is routed before the peer is truly ready.
- Conversely, an excessively long settle time delays legitimate recovery and extends the outage window.

Resolution:

1. Investigate the underlying transport instability first (network path, SCTP path failover, or peer-side restarts). A flapping peer is a symptom, not a routing-config problem.

2. Tune the settle time so a freshly-connected peer must stay up for a defined interval before traffic is routed to it. A peer that disconnects inside this window is never added to the routable set, so traffic is kept off flapping peers.

```
config :dra, :gateway,
  # Hold a freshly-connected peer this many ms before routing
traffic to it.
  # 0 = route immediately once CER/CEA completes.
  peer_settle_time_ms: 2_000,

  # Optional override (ms) for the RFC 3539 watchdog timer Tw.
  # watchdog_timer_ms: 30_000
```

Parameter	Type	Required	Default	Description
`peer_settle_time_ms`	Integer	No	`0`	Milliseconds a newly-connected peer must remain up before the DRA routes traffic to it. `0` routes immediately once CER/CEA completes. If the peer disconnects during the window, it is never marked routable, so traffic is kept off flapping peers.
`watchdog_timer_ms`	Integer	No	`30000`	Optional override for the RFC 3539 watchdog timer Tw, the Device-Watchdog (DWR/DWA) liveness

Parameter	Type	Required	Default	Description
				interval. Leave unset unless you have a specific reason to change the liveness cadence.

3. `diameter_peer_status` reflects actual routability, so a peer inside its settle window still reports as not-yet-routable in metrics. Use this to confirm the settle behaviour is doing what you expect.

4. Raise `peer_settle_time_ms` for chronically flapping peers to suppress premature dispatch; keep it low (or `0`) for stable peers to minimise recovery time.

> The DRA promotes a reconnecting peer to routable immediately once the application-level settle time elapses, rather than waiting out the RFC 3539 watchdog's implicit reconnect delay. The settle time above is the single, explicit control over how long a come-up is held.

Unauthorized Peer Connections

Symptoms: A peer cannot connect and is rejected, or `diameter_peer_unauthorized_connection_count_total` increments with an unexpected `origin_host` / `peer_ip`.

Possible causes:

- The connecting peer is not present in the `peers` configuration and `allow_undefined_peers_to_connect` is `false`.
- The peer's advertised Origin-Host does not match any configured `host`.
- A genuinely unauthorized or misconfigured node is attempting to connect.

Resolution:

1. Confirm whether the peer should be allowed. If so, add it to the `peers` list with its exact `host`, `realm`, `ip`, `port`, and `transport`.
2. Verify the peer's advertised Origin-Host matches the configured `host` exactly. A mismatch is treated as an unknown peer.
3. Keep `allow_undefined_peers_to_connect: false` in production and add peers explicitly. Only enable it in controlled environments where any peer may connect.
4. Keep `log_unauthorized_peer_connection_attempts: true` so rejected attempts are logged for security review.
5. Investigate the `peer_ip` and claimed `origin_host` labels on `diameter_peer_unauthorized_connection_count_total`. A sustained rate from an unknown source warrants a security investigation.

Parameter	Type	Required
`allow_undefined_peers_to_connect`	Boolean	No
`log_unauthorized_peer_connection_attempts`	Boolean	No

SCTP Association Issues

Symptoms: An SCTP peer fails to establish an association, or an established association drops or fails over unexpectedly. TCP peers on the same host connect normally.

Possible causes:

- The kernel SCTP module is not loaded on the host.
- Firewall rules permit TCP but not SCTP on the configured addresses.
- One or more addresses in the multihoming set are not routable from the peer.
- The primary IP is not included in the peer's `additional_ips` list.
- Only one endpoint is configured for multihoming, so full redundancy is not achieved.
- Path failover timing is governed by kernel SCTP parameters and differs from expectations.

Resolution:

1. Confirm the SCTP kernel module is loaded (the `lksctp-tools` package on Linux).
2. Ensure firewall rules allow SCTP traffic on every configured IP, not only TCP.
3. Verify all IP addresses in the multihoming set are routable in both directions between the DRA and the peer.
4. For an SCTP peer, include the primary IP in the `additional_ips` list; for TCP peers only `ip` is used, as TCP does not support multihoming.
5. Configure both endpoints for multihoming for full path redundancy.
6. If failover timing is not as expected, review kernel SCTP parameters. Path failover timing depends on them, not on DRA configuration.
7. See SCTP Multihoming for the complete transport configuration and its limitations.

Message Delivery

Messages Not Routing / 3002 UNABLE_TO_DELIVER

Symptoms: Requests arrive at the DRA but are answered with Result-Code `3002` (DIAMETER_UNABLE_TO_DELIVER), or are not forwarded to the expected destination at all.

Possible causes:

- No connected peer matches the request's Destination-Host or Destination-Realm.
- The target peer is disconnected, or is inside its settle window and not yet routable.
- The target peer does not advertise support for the application in the request.
- An advanced routing rule selected a peer that is not reachable.
- The requested realm is not served by any peer (this yields `3003`, DIAMETER_REALM_NOT_SERVED, rather than `3002`).

Resolution:

1. Confirm at least one peer matching the Destination-Host or Destination-Realm is connected. `diameter_peer_status` for that peer should read `1`.

2. Check whether the target peer is inside its settle window. A peer that has just reconnected is intentionally not routable until `peer_settle_time_ms` elapses. See Peer Flapping.

3. Verify the peer advertises the application in the request during CER/CEA. Peers that do not advertise the application are excluded from selection.

4. If an advanced routing rule targets a specific peer, confirm that peer is connected. Disconnected peers are skipped. See Advanced Routing.

5. Distinguish `3002` from `3003`: `3002` means no reachable peer could be selected, while `3003` means no peer serves the requested realm. Correct

realm configuration if you observe `3003`.

6. Track the rate of `3002` answers to spot systemic delivery failures:

```
rate(diameter_peer_message_result_code_count_total{result_code="3
[5m])
```

Unanswered Requests (Timeouts)

Symptoms: Requests are forwarded to a peer but no answer returns within the timeout. `diameter_peer_unanswered_request_count_total` increments and `diameter_peer_last_response_delay` climbs.

Possible causes:

- The peer received the request but did not answer within `request_timeout` (default `5000` ms).
- The peer is overloaded or slow.
- A network path is degraded (relevant for SCTP multihoming during failover).
- The peer processed the request but the answer was lost in transit.

Resolution:

1. Identify the slow or silent peers:

```
topk(5, sum by (routed_to) (
  rate(diameter_peer_unanswered_request_count_total[5m])
))
```

2. Compare against `diameter_peer_last_response_delay` for the same peer to see whether it is slow rather than silent.

3. Review the peer's own load and health. Sustained timeouts usually indicate a downstream problem, not a DRA problem.

4. For SCTP peers, check for path failover events that coincide with the timeouts. See SCTP Multihoming.

5. Adjust `request_timeout` only if the peer's legitimate processing time genuinely exceeds the default. Increasing it also extends the Extended Metrics TTL (`request_timeout` + 5 seconds).

Parameter	Type	Required	Default	Description
`request_timeout`	Integer	No	`5000`	Milliseconds to wait for an answer before the request/answer pair is considered timed out.

UNKNOWN_SESSION_ID on Stateful Gx / Gy

Symptoms: Stateful sessions (Gx policy control, Gy online charging) fail with Result-Code `5002` (DIAMETER_UNKNOWN_SESSION_ID). Typically an update (CCR-U) or termination (CCR-T) is rejected even though the initial (CCR-I) succeeded.

Possible causes:

- A mid-session request was routed to a different peer than the one holding the session state.
- The peer holding the session restarted and lost its session table.
- A peer reconnected and was routed to before its state was restored (settle-window interaction).
- Load balancing spread requests for one Session-Id across multiple peers.

Resolution:

1. Ensure all requests carrying the same Session-Id reach the same stateful peer. For Gx and Gy, session affinity is essential: mid-session messages must land on the peer that answered CCR-I. Prefer `Destination-Host`-based routing or an advanced rule that pins a Session-Id family to one peer over realm-based load balancing. See Standard Routing and Advanced Routing.

2. Review the `peer_selection_algorithm`. `:random` distributes across matching peers and can break session affinity for stateful applications; `:failover` sends to the first peer in the list. For stateful interfaces, route on Destination-Host so affinity is explicit rather than relying on load balancing.

3. Check whether the target peer restarted around the time the `5002` answers began. A peer that lost its session table will reject in-flight sessions; those sessions must be re-established by the client.

4. If a peer had recently reconnected, confirm `peer_settle_time_ms` is large enough that traffic is only routed once the peer is genuinely ready. See Peer Flapping.

5. Track the rate of `5002` answers to correlate with peer events:

```
rate(diameter_peer_message_result_code_count_total{result_code="5
[5m])
```

Reference: Common Result Codes

Code	Name	Meaning	Reference
2001	DIAMETER_SUCCESS	Request completed successfully.	RFC 6733 §7.1.1
3002	DIAMETER_UNABLE_TO_DELIVER	No reachable peer could be selected to deliver the request.	RFC 6733 §7.1.3
3003	DIAMETER_REALM_NOT_SERVED	No peer serves the requested realm.	RFC 6733 §7.1.3
3004	DIAMETER_TOO_BUSY	The selected peer is overloaded.	RFC 6733 §7.1.3
5002	DIAMETER_UNKNOWN_SESSION_ID	The peer does not recognise the Session-Id (lost or misrouted stateful session).	RFC 6733 §7.1.5
5012	DIAMETER_UNABLE_TO_COMPLY	The request cannot be processed as presented.	RFC 6733 §7.1.5

Related Documentation

- Metrics - Prometheus metrics and example queries referenced above.
- Standard Routing - RFC 6733 priority routing, peer selection, and answer routing.
- Advanced Routing - Rule-based routing, filters, transforms, and peer targeting.
- SCTP Multihoming - SCTP transport, multihoming, and path failover.